ARM and Embedded System
22EC62
Module 2

ARM Embedded Software Development

Contents

Inside ARM

- There are many different things inside a microcontroller. In many microcontrollers, the processor takes less than 10% of the silicon area, and the rest of the silicon die is occupied by other components such as:
- Program memory (e.g., flash memory)
- SRAM
- Peripherals
- Internal bus infrastructure
- Clock generator (including Phase Locked Loop), reset generator, and distribution network for these signals
- Voltage regulator and power control circuits
- Other analog components (e.g., ADC, DAC, voltage reference circuits)
- I/O pads
- Support circuits for manufacturing tests, etc.

Development suites

- With more than 10 different vendors selling C compiler suites for Cortex-M micro controllers, The current available choices included various products from the following vendors:
- Keil Microcontroller Development Kit (MDK-ARM)
- ARM DS-5 (Development Studio 5)
- IAR Systems (Embedded Workbench for ARM Cortex-M)
- Red Suite from Code Red Technologies (acquired by NXP in 2013)
- Mentor Graphics Sourcery CodeBench (formerly CodeSourcery Sourcery g++)
- mbed.org
- Altium Tasking VX-toolset for ARM Cortex-M
- Rowley Associates (CrossWorks)
- Coocox
- Texas Instruments Code Composer Studio (CCS)
- Raisonance RIDE
- Atollic TrueStudio
- GNUCompiler Collection (GCC)
- ImageCraft ICCV8
- Cosmic Software C Cross Compiler for Cortex-M
- mikroElektronika mikroC
- Arduino
- there are development suites for other languages. For example: • Oracle Java ME Embedded • IS2T MicroEJ Java virtual machine • mikroElektronika mikroBasic, mikroPascal

Development boards

- A number of low-cost development boards are designed to work with particular development suites.
- For example, the “mbed.org” development boards, a low-cost solution for rapid software prototyping, are designed to work with the mbed development platform.
- for example, companies like Keil (an example is show in Figure 2.1), IAR Systems, and Code Red Technologies all have a number of development boards available.
- the Keil MDK-ARM even supports device-level simulation for some of the popular Cortex-M microcontrollers.

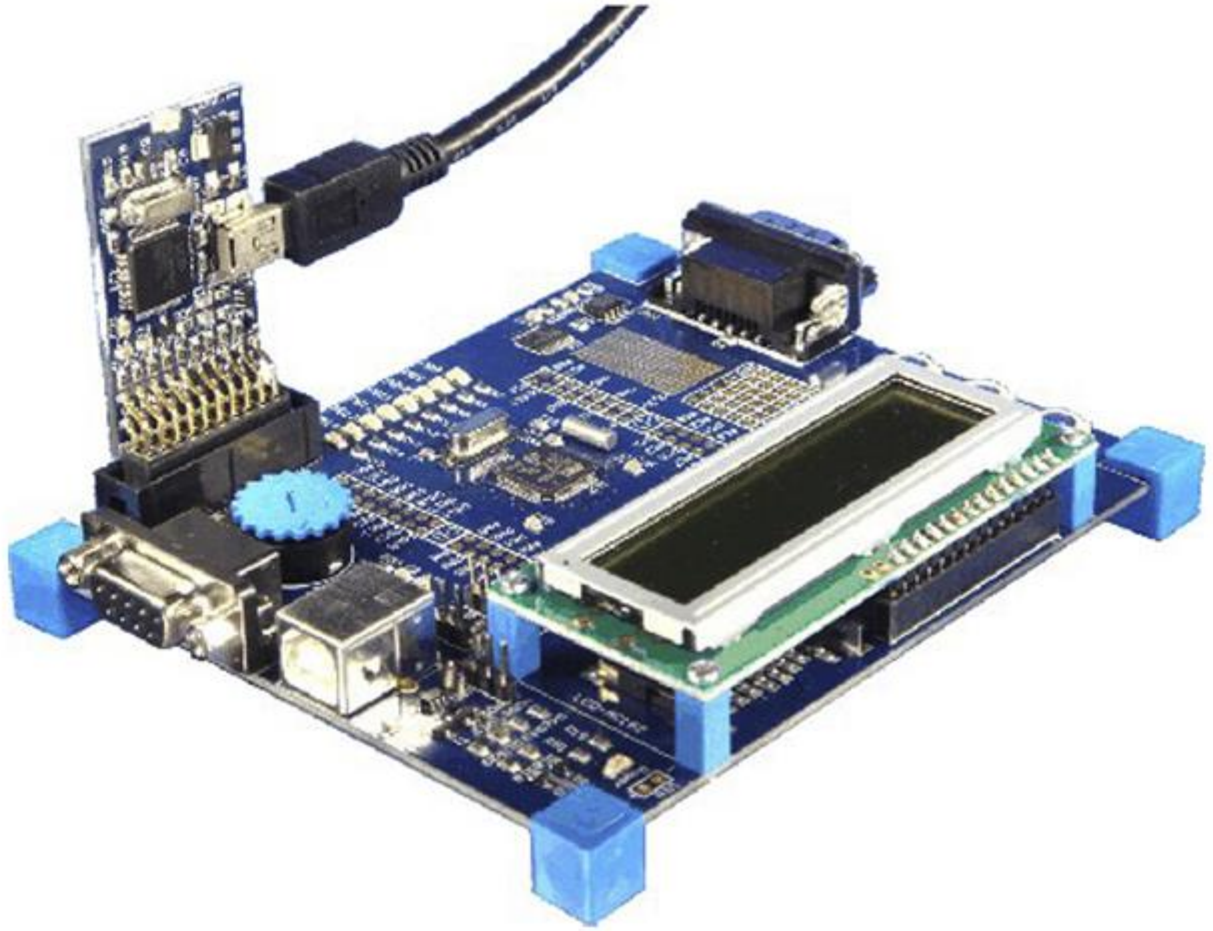

FIGURE 2.1

A Cortex-M3 development board from Keil (MCBSTM32)

Development adaptors

- Most C compiler vendors have their own debug adaptor products. For example, Keil has the ULINK product family (Figure 2.2), and IAR provides the I-Jet product.
- Most development suites also support third-party debug adaptors. Note that different vendors might have different terminologies for these debug adaptors, for example, debug probe, USB-JTAG adaptor, JTAG/SW Emulator, JTAG In-Circuit Emulator (ICE), etc.

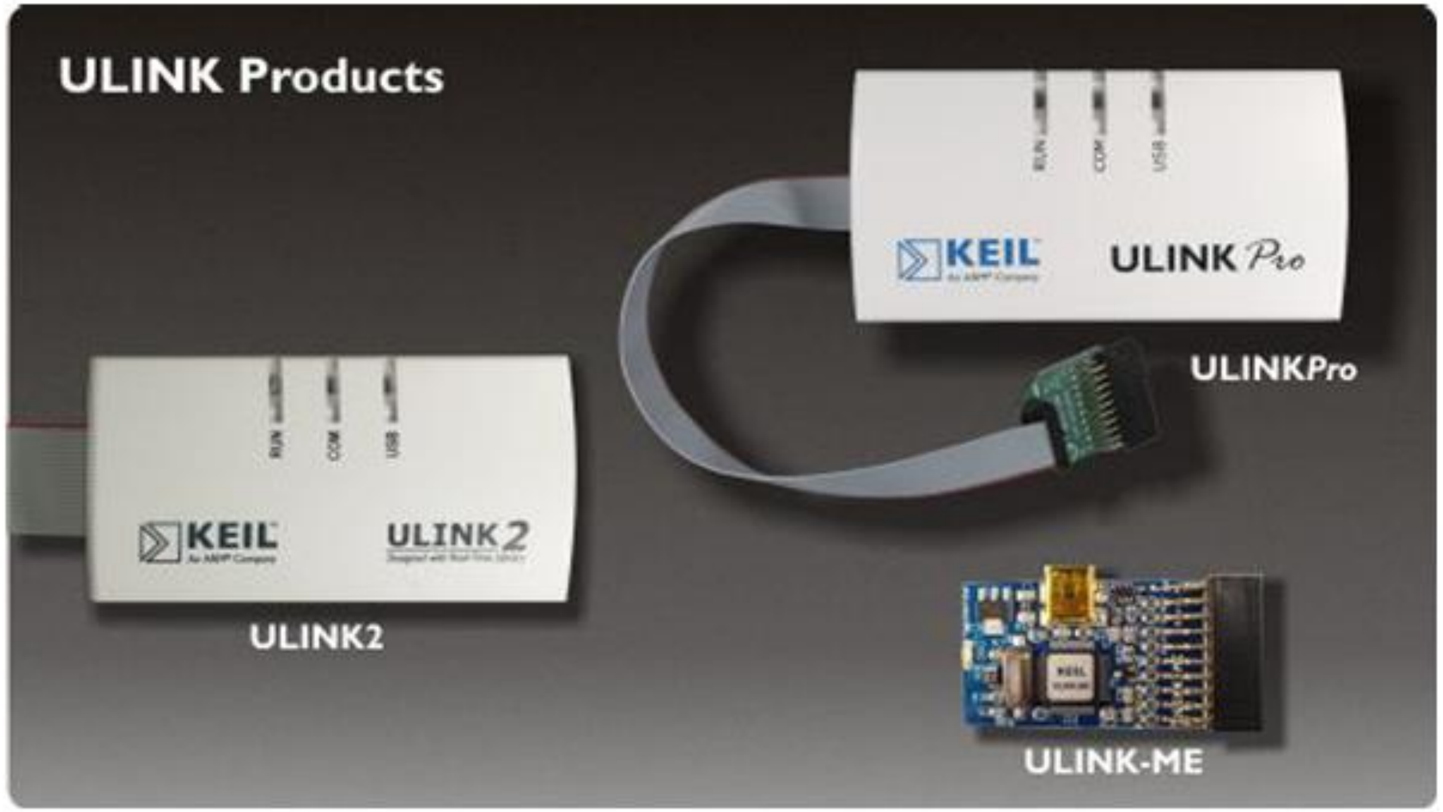

FIGURE 2.2

The Keil ULINK debug adaptor family

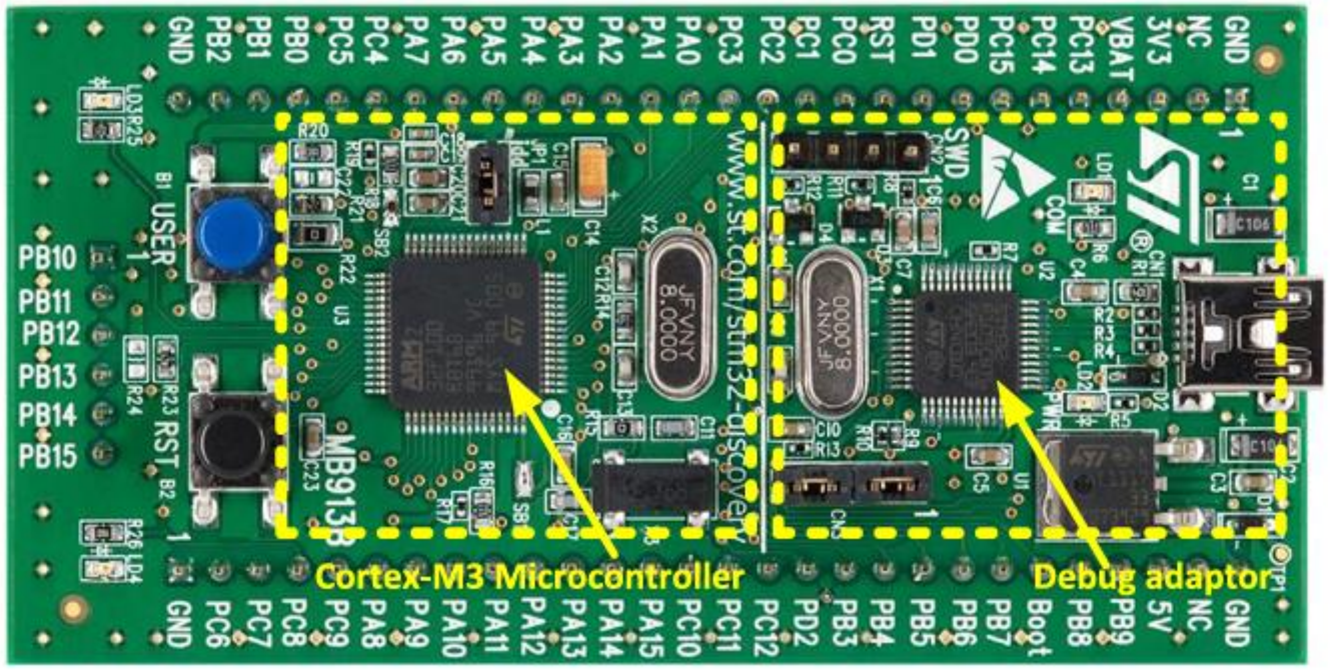

FIGURE 2.3

An example of development board with USB debug adaptor — STM32 Value Line Discovery

- STMicroelectronics (e.g., STM32 Value Line Discovery; Figure 2.3), NXP, Energy Micro, etc. Many of these onboard USB adaptors are also supported by mainstream commercial development suites. So you can start developing software for the Cortex-M microcontrollers with a tiny budget.
- The built-in USB debug adaptor can also be used to connect to other development boards. You can also find "open source" versions of such debug adaptors. The CMSIS-DAP from ARM and CoLink from Coocox are two examples.

Software device drivers

- The term device driver here is quite different from its meaning in a PC environment. In order to help microcontroller software developers, microcontroller vendors usually provide header files and C codes that include:
- Definitions of peripheral registers
- Access functions for configuring and accessing the peripherals
- Adding these files to your software projects, you can access various peripheral functions via function calls and access peripheral registers easily.
- If you want to, you can also create modified versions of the access functions based on the methods shown in the driver code and optimize them for your application.

Software development flow

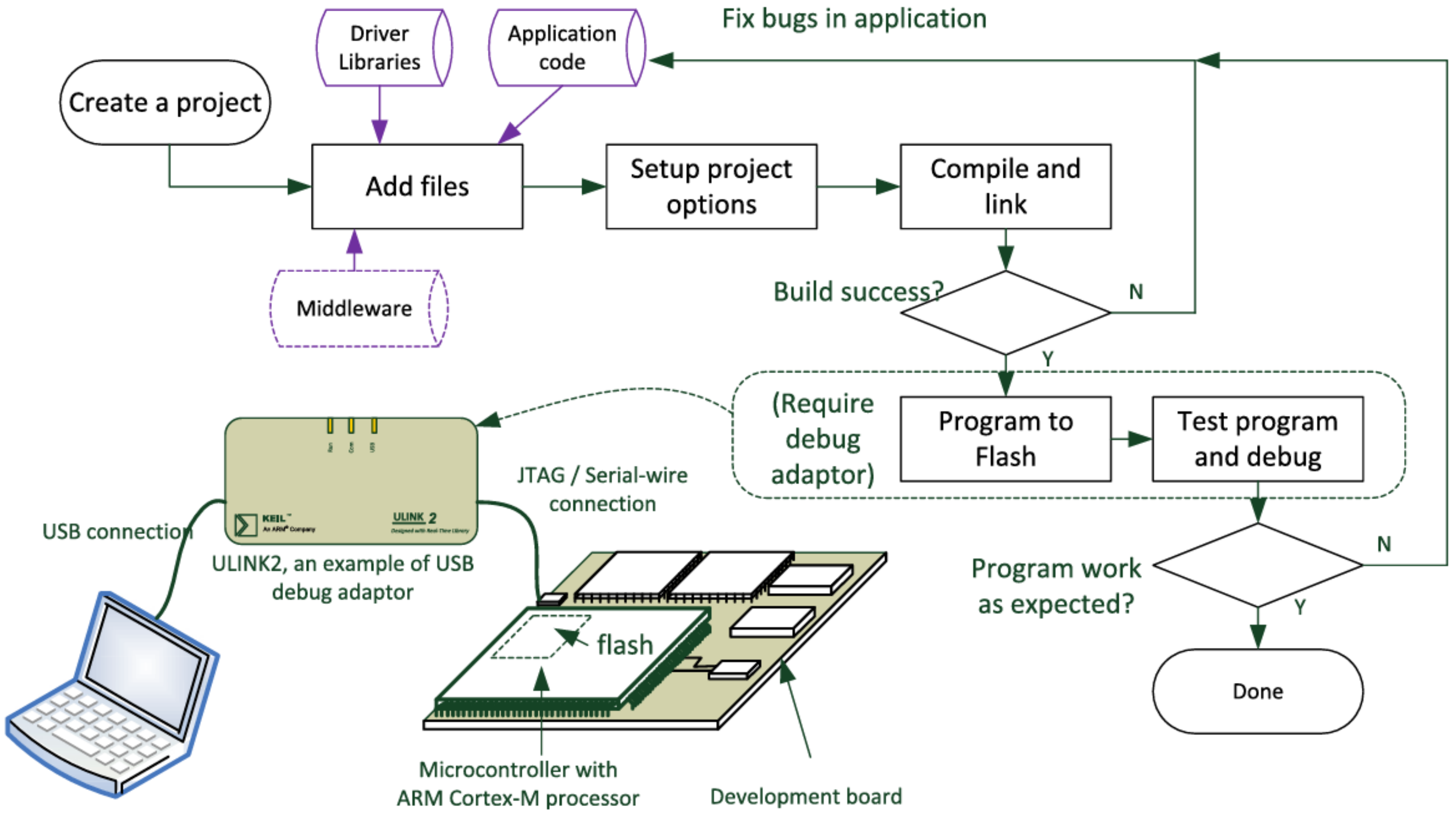

FIGURE 2.4

A simplified software development flow

- Create Project
- Add files to project
- Setup project options - compiler optimization options, memory map, and output file types for debug and code download
- Compile and link
- Flash programming
- Execute program and debug

Compiling

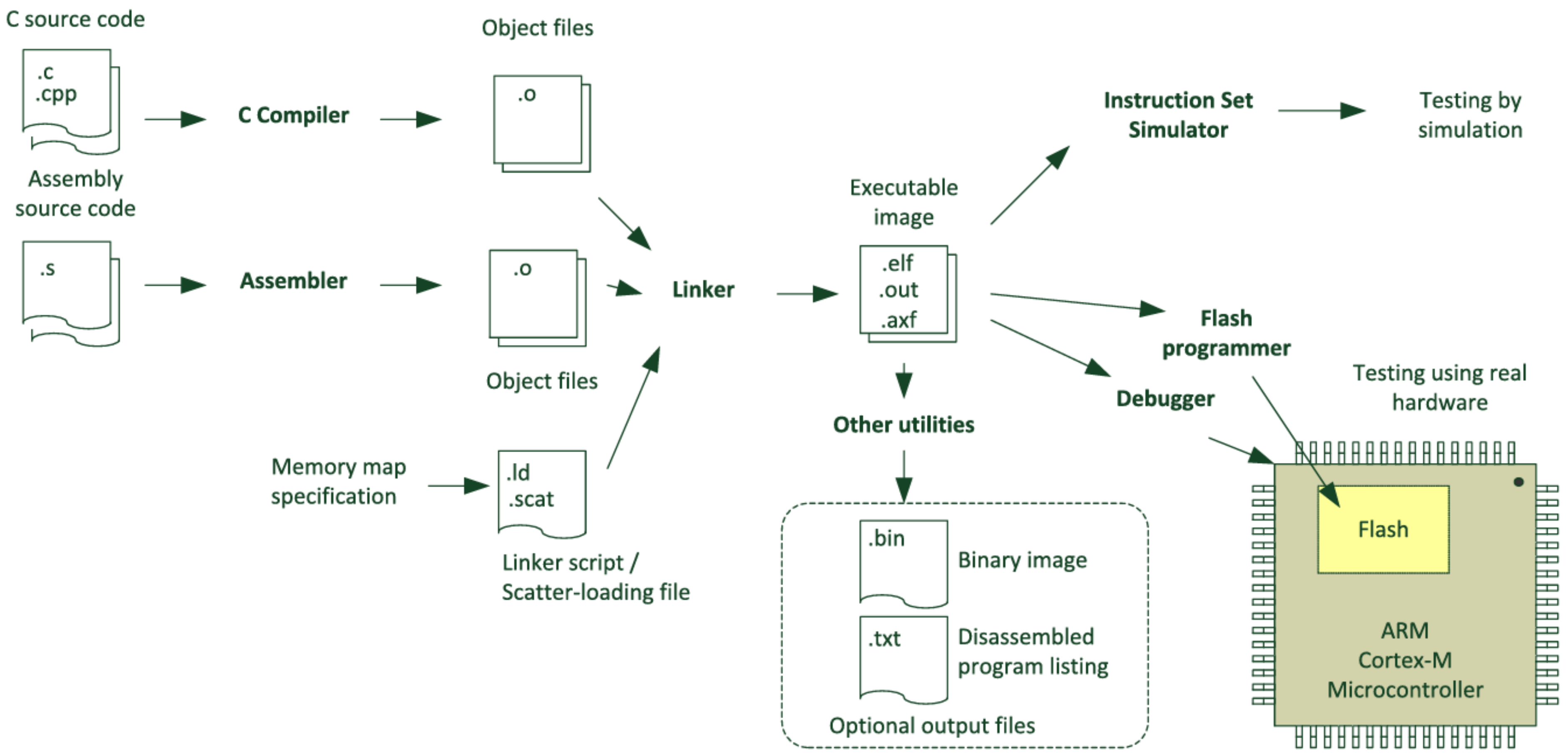

FIGURE 2.5

Common software compilation flow

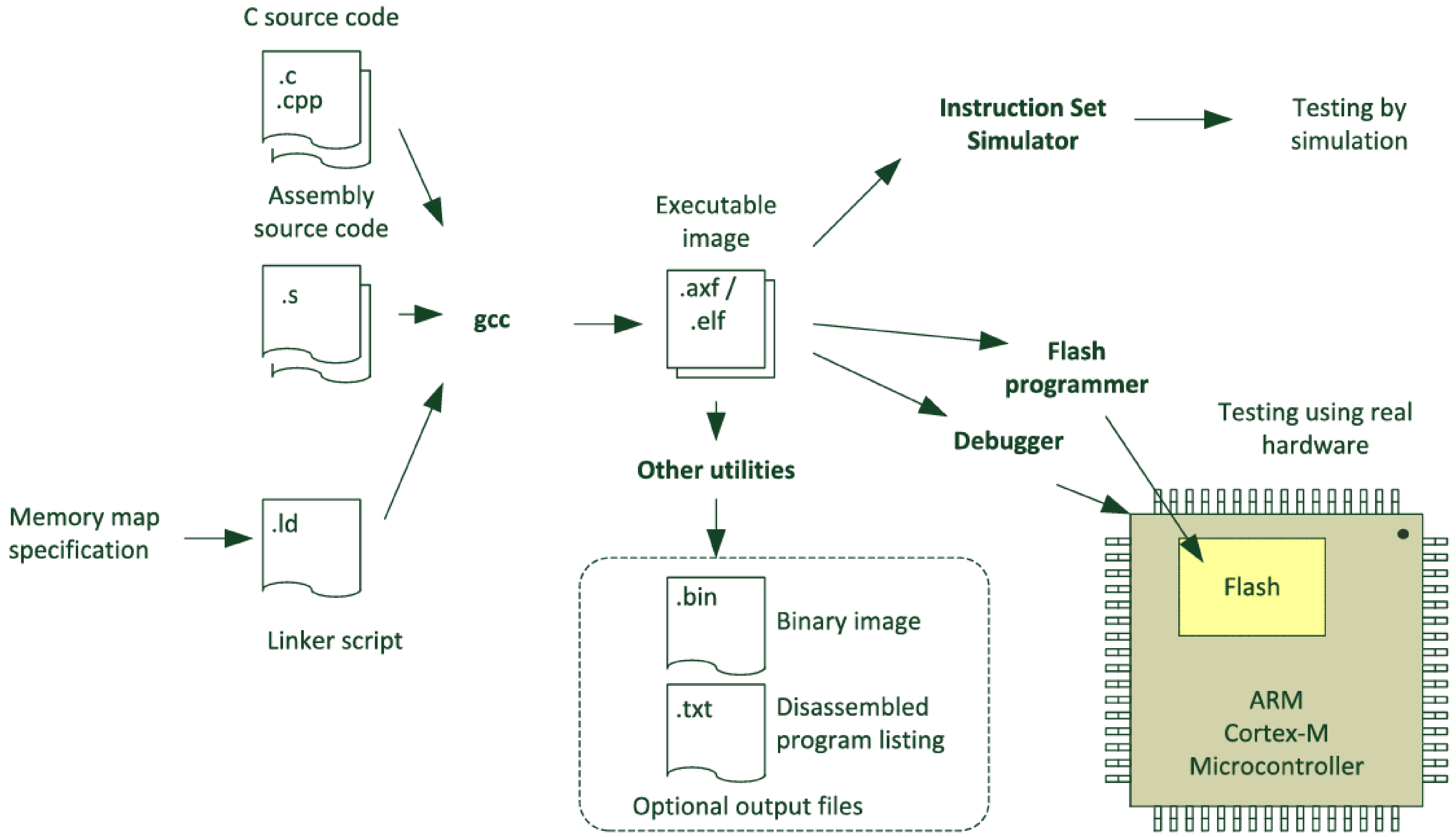

FIGURE 2.6

Common software compilation flow for GNU toolchain

Table 2.1 Various Tools You Can Find in a Development Suite

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

Software flow

- Two methods of Software flow –
- **Polling -** For very simple applications, the processor can wait until there is data ready for pro cessing, process it, and then wait again. This is very easy to setup and works fine for simple tasks.
- Figure 7 shows a simple polling program flow chart.
- A microcontroller will have to serve multiple interfaces and there fore be required to support multiple processes.
- The polling program flow method can be expanded to support multiple processes easily (Figure 2.8).
- This arrangement is sometimes called a “super-loop.”

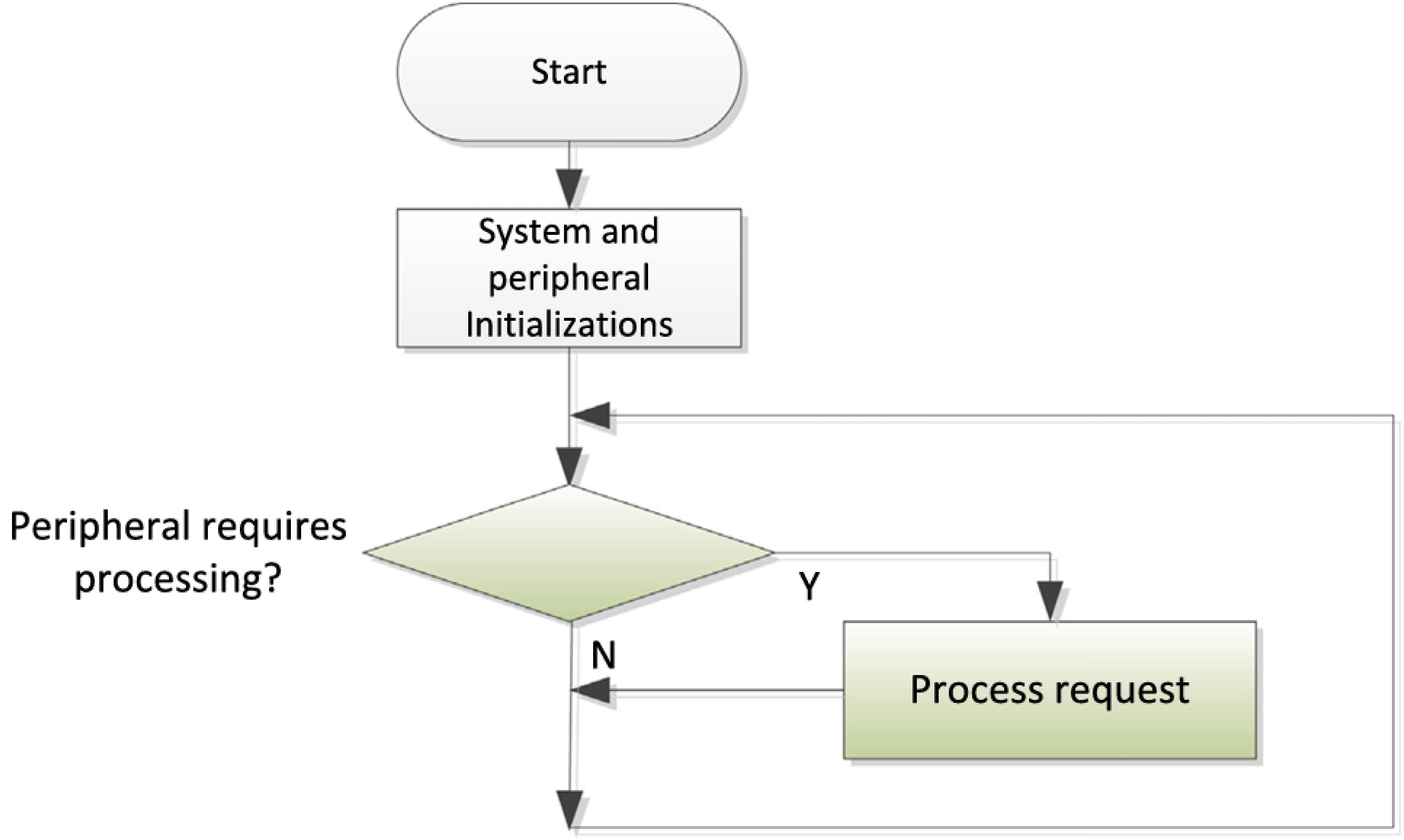
Start
System and peripheral Initializations
Peripheral requires processing?
Y
N
Process request

FIGURE 2.7

Polling method for simple application processing

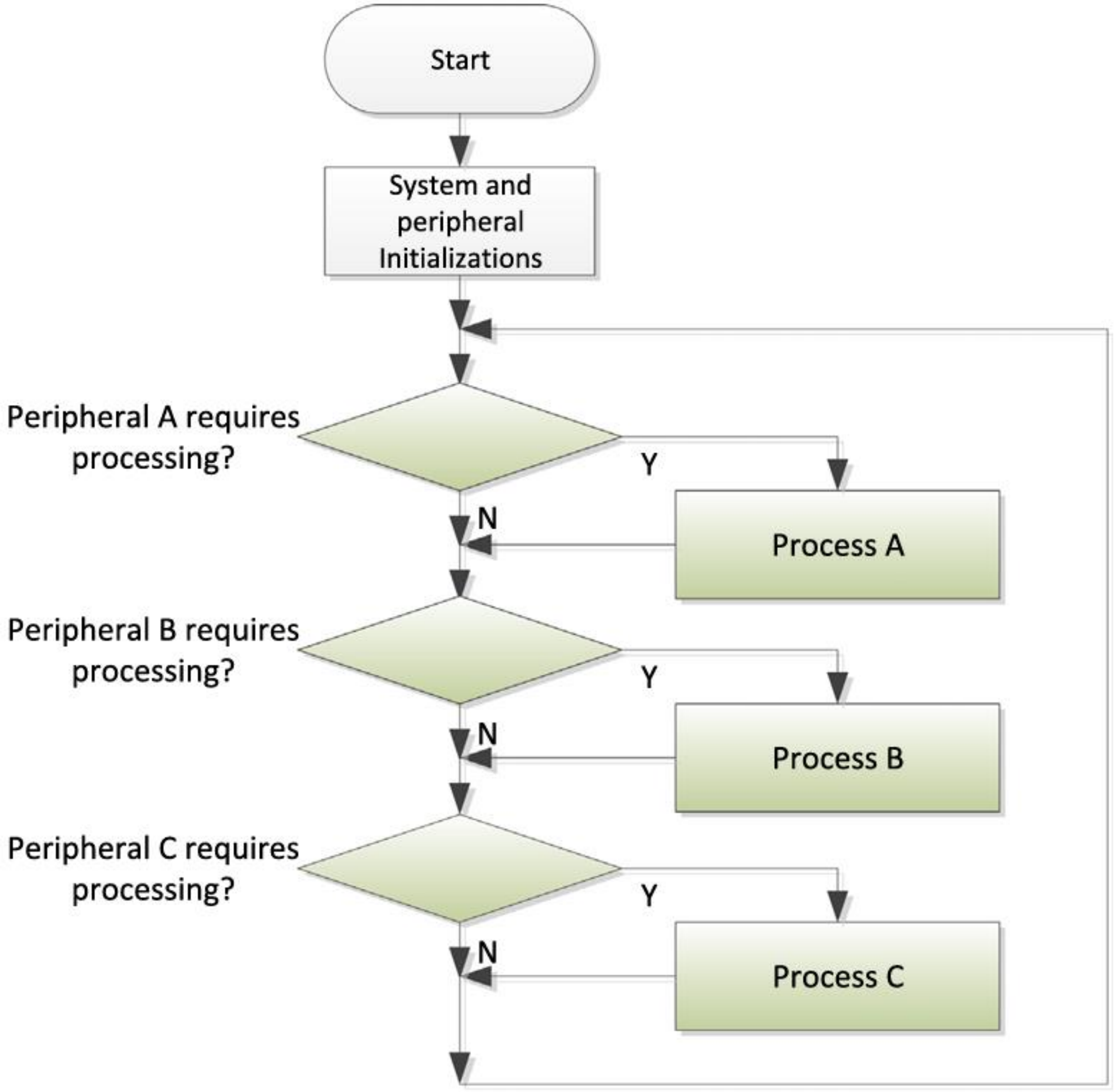
Start
System and peripheral Initializations
Peripheral A requires processing?
Y
N
Process A
Peripheral B requires processing?
Y
N
Process B
Peripheral C requires processing?
Y
N
Process C

FIGURE 2.8

Polling method for application with multiple devices that need processing

Software flow

- The polling method works well for simple applications, but it has several disadvantages.
- For example, when the application gets more complex, the polling loop design might get very difficult to maintain.
- It is difficult to define priorities be tween different services using polling e you might end up with poor responsiveness,
- A peripheral requesting service might need to wait a long time while the processor is handling less important tasks.
- Another main disadvantage of the polling method is that it is not energy efficient. Lots of energy is wasted during the polling when service is not required.

Software flow

- **Interrupt driven**
- All microcontrollers have some sort of sleep mode support to reduce power, in which the peripheral can wake up the processor when it requires a service (Figure 2.9). This is commonly known as an interrupt-driven application.
- In an interrupt-driven application, interrupts from different peripherals can be assigned with different interrupt priority levels.
- For example, important/critical peripherals can be assigned with a higher priority level so that if the interrupt arrives when the processor is servicing a lower priority interrupt,
- The execution of the lower priority interrupt service is suspended, allowing the higher priority interrupt service to start immediately. This arrangement allows much better responsiveness.

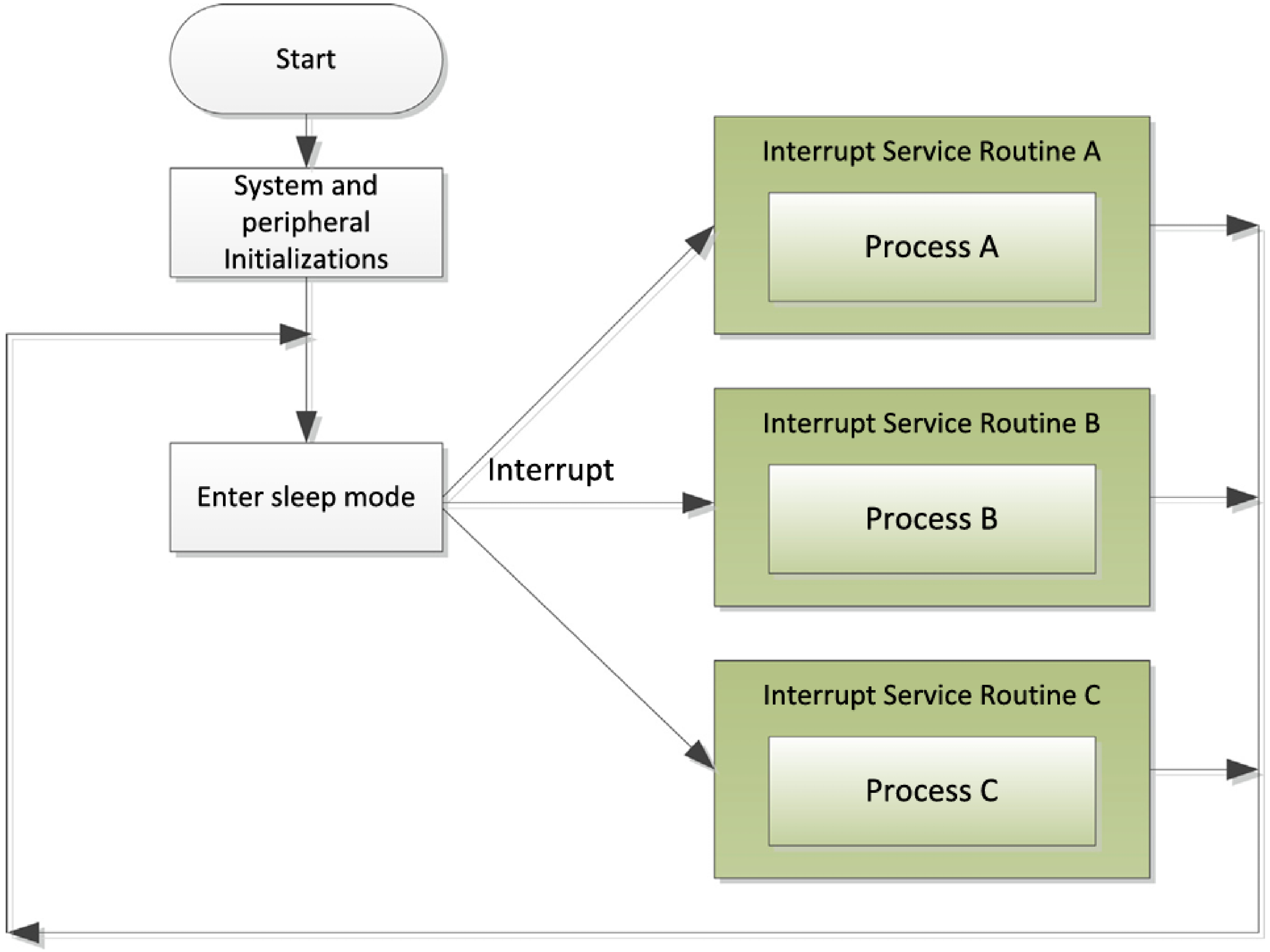
Start
System and peripheral Initializations
Enter sleep mode
Interrupt
Interrupt Service Routine A
Process A
Interrupt Service Routine B
Process B
Interrupt Service Routine C
Process C

FIGURE 2.9

Simple interrupt-driven application

- the processing of data from peripheral services can be partitioned into two parts: the first part needs to be done quickly, and the second part can be carried out a little bit later.
- In such situations we can use a mixture of interrupt-driven and polling methods to construct the program.
- When a peripheral requires service, it triggers an interrupt request as in an interrupt-driven application.
- Once the first part of the interrupt service is carried out, it updates some software variables so that the second part of the service can be executed in the polling-based application code (Figure 2.10)
- Using this arrangement, we can reduce the duration of high-priority interrupt handlers so that lower priority interrupt services can get served quicker. At the same time, the processor can still enter sleep mode to save power when no servicing is needed.

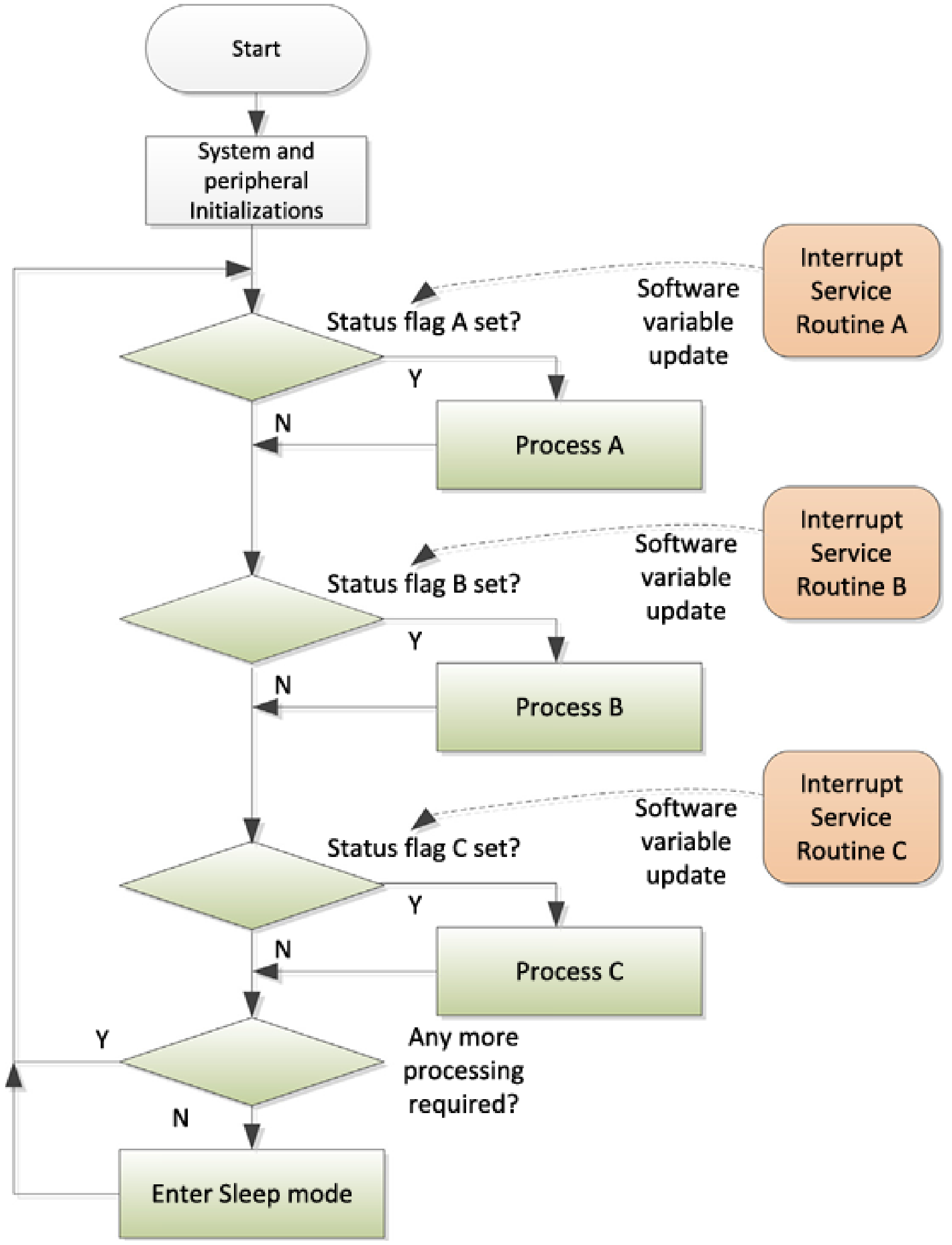

FIGURE 2.10

Application with both polling method and interrupt-driven arrangement

Microcontroller Interface

- Unlike programming for PCs, most embedded applications do not have a rich GUI where as many development boards might have an LCD screen or couple of LEDs and buttons.
- A UART is easy to use, and allows more information to be passed to the developer quickly. The Cortex-M3/M4 processor does not have a UART as standard, but most microcontroller vendors have included a UART peripheral in their microcontroller designs.
- Most modern computers do not have a UART interface (COM port) anymore, so you might need to use a USB-to-UART adaptor cable to handle this communication (**a TTL-to-RS232 adaptor in your development setup to convert the signal’s voltage**)

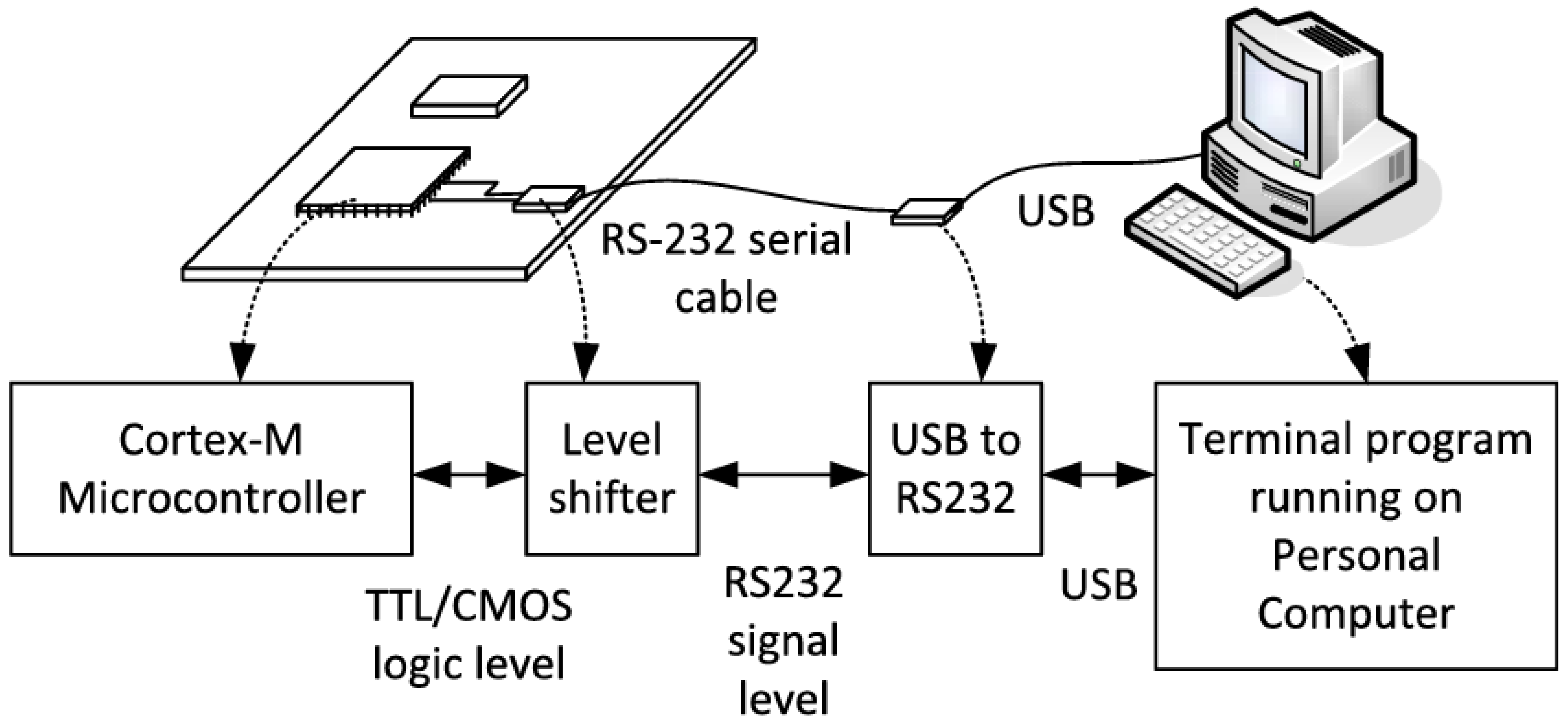

FIGURE 2.12

Using a UART to communicate with a PC via USB

- If the microcontroller you use has a USB interface, you can use this to communicate with a PC using USB.
- **For example**, you can use a Virtual COM port solution for text-based communication with a terminal program running on a computer.
- It requires more effort in setting up the software but allows the microcontroller hardware to interface with the PC directly, avoiding the cost of the RS232 adaptors.
- If you are using commercial debug adaptors like the Keil ULINK2, Segger J-LINK, or similar, you can use a feature called Instrumentation Trace Macrocell (ITM) to transfer messages to the debug host (the PC running the debugger) and display the messages in the development environment.
- This does not require any extra hardware and does not require much software overhead. It allows the peripheral interfaces to be free for other purposes.

The Cortex microcontroller software interface standard (CMSIS)

- Embedded systems are also becoming more and more complex, and the amount of effort in developing and testing the software has increased substantially.
- In order to reduce development time as well as reducing the risk of having defects in products, software reuse is becoming more and more common.
- In addition, the complexity of the embedded systems has also increased the use of third party software solutions.
- Some form of standardization of the way the soft ware infrastructure works becomes necessary to ensure software compatibility with various development tools and between different software solutions.

- For Example - an embedded software project might involve software components from many different sources:
- Software developed by in house developers
- Software reused from other projects
- Device-driver libraries from microcontroller vendors
- Embedded OSs
- Other third-party software products such as communication protocol stacks
- the interoperability of various software components becomes critical.

Introduction of CMSIS

- ARM worked with various microcontroller vendors, tools vendors, and software solution providers to develop CMSIS, a software frame work covering most Cortex-M processors and Cortex-M microcontroller products.
- The aims of CMSIS include:
- Enhanced software reusability e makes it easier to reuse software code in different Cortex-M projects, reducing time to market and verification efforts.
- Enhanced software compatibility e by having a consistent software infrastructure (e.g., API for processor core access functions, system initialization method, common style for defining peripherals), software from various sources can work together, reducing the risk in integration.
- Easy to learn e the CMSIS allows easy access to processor core features from the C language. In addition, once you learn to use one Cortex-M microcontroller product, starting to use another Cortex-M product is much easier because of the consistency in software setup.

- Toolchain independent e CMSIS-compliant device drivers can be used with various compilation tools, providing much greater freedom.
- Openness - the source code for CMSIS core files can be downloaded and accessed by everyone, and everyone can develop software products with CMSIS.

Additional CMSIS Projects

- **CMSIS-Core (Cortex-M processor support)** - a set of APIs for application or middleware developers to access the features on the Cortex-M processor regardless of the microcontroller devices or toolchain used.
- Currently the CMSIS processor support includes the Cortex-M0, Cortex-M0þ, Cortex-M3, and Cortex-M4 processors and SecurCore products like SC000 and SC300. Users of the Cortex-M1 can use the Cortex-M0 version because they share the same architecture.
- **CMSIS-DSP library** - in 2010 the CMSIS DSP library was released, supporting many common DSP operations such as FFT and filters. The CMSIS-DSP is intended to allow software developers to create DSP applications on Cortex-M microcontrollers easily.
- **CMSIS-SVD** - the CMSIS System View Description is an XML-based file format to describe peripheral set in microcontroller products. Debug tool vendors can then use the CMSIS SVD files prepared by the microcontroller vendors to construct peripheral viewers quickly.
- **CMSIS-RTOS** - the CMSIS-RTOS is an API specification for embedded OS running on Cortex-M microcontrollers. This allows middleware and application code to be developed for multiple embedded OS platforms, and allows better reusability and portability.
- **CMSIS-DAP** – the CMSIS-DAP (Debug Access Port) is a reference design for a debug interface adaptor, which supports USB to JTAG/Serial protocol conversions.
- This allows low-cost debug adaptors to be developed which work for multiple development toolchains.

Areas of standardization in CMSIS-Core

- **Standardized definitions for the processor’s peripherals** - These include the reg isters in the Nested Vector Interrupt Controller (NVIC), a system tick timer in the processor (SysTick), an optional Memory Protection Unit (MPU), various programmable registers in the System Control Block (SCB), and some software programmable registers related to debug features.
- **Standardized access functions to access processor’s features** - These include various functions for interrupt control using NVIC, and functions for accessing special registers in the processors.
- **Standardized functions for accessing special instructions easily** - The Cortex-M processors support a number of instructions for special purposes (e.g., Wait For-Interrupt, WFI, for entering sleep mode).

- **Standardized function names for system exception handlers** - A number of system exception types are presented in the architecture for the Cortex-M processors.
- **Standardized functions for system initialization** - Most modern feature-rich microcontroller products require some configuration of clock circuitry and power management registers before the application starts.
- **Standardized software variables for clock speed information** - This might not be obvious, but often our application code does need to know what clock frequency the system is running at.
- **The CMSIS-Core also provides:** A common platform for device-driver libraries e Each device-driver library has the same look and feel, making it easier for beginners to learn how to use the devices.

Organization of CMSIS-Core

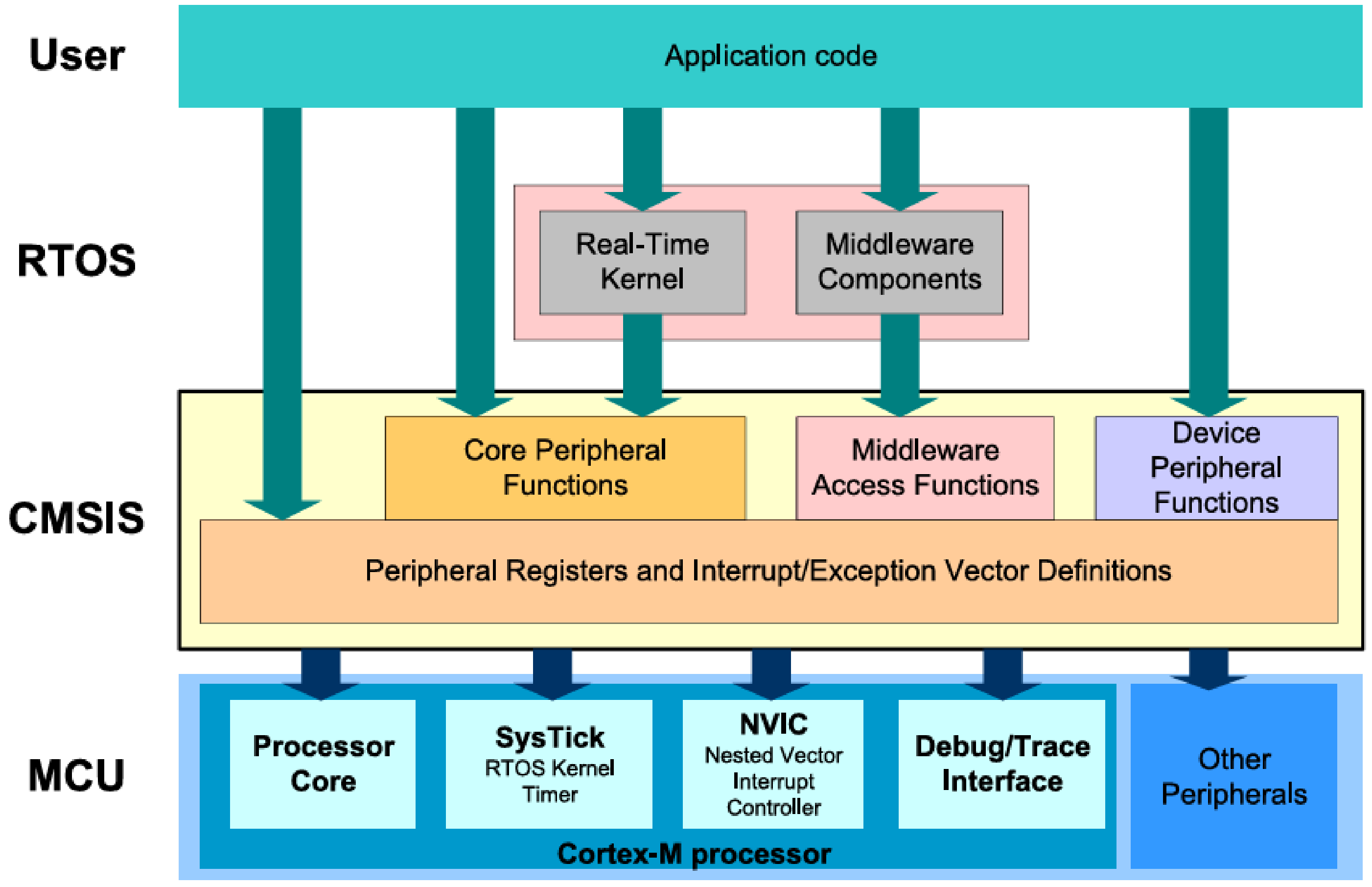
User
Application code
RTOS
Real-Time Kernel
Middleware Components
CMSIS
Core Peripheral Functions
Middleware Access Functions
Device Peripheral Functions
Peripheral Registers and Interrupt/Exception Vector Definitions
MCU
Processor Core
SysTick
RTOS Kernel Timer
NVIC
Nested Vector Interrupt Controller
Debug/Trace Interface
Cortex-M processor
Other Peripherals

FIGURE 2.13

CMSIS-Core structure

- The CMSIS files are integrated into device-driver library packages from microcon troller vendors.
- CMSIS files can be defined into multi layers:
- **Core Peripheral Access Layer** - Name definitions, address definitions, and helper functions to access core registers and core peripherals.
- **Device Peripheral Access Layer** - Name definitions, address definitions of peripheral registers, as well as system implementations including interrupt assignments, exception vector definitions, etc. This is device specific (note: multiple devices from the same vendor might use the same file set).
- **Access Functions for Peripherals** - The driver code for peripheral accesses. This is vendor specific and is optional.
- **Middleware Access Layer** - This layer does not exist in current version of CMSIS. The idea is to develop a set of APIs for interfacing common peripherals such as UART, SPI, and Ethernet. If this layer exists, developers of middleware can develop their applications based on this layer to allow software to be ported between devices easily.

Use CMSIS-Core

- The CMSIS files are included in the device-driver packages provided by the micro controller vendors i.e., CMSIS-compliant device-driver libraries.
- **Add source files to project.** This includes:
- Device-specific, toolchain-specific startup code, in the form of assembly or C
- Device-specific device initialization code (e.g., system_.c)
- Additional vendor-specific source files for peripheral access functions. This is **optional**

- Add header files into search path of the project. This includes:
- A device-specific header file for peripheral registers definitions and interrupt assignment definitions. (e.g., .h)
- A device-specific header file for functions in device initialization code (e.g., system_.h)
- A number of processor-specific header files (e.g., core_cm3.h, core_cm4.h; they are generic for all microcontroller vendors)
- Optionally additional vendor-specific header files for peripheral access functions
- In some cases the development suites might also have some of the generic CMSIS support files pre-installed.

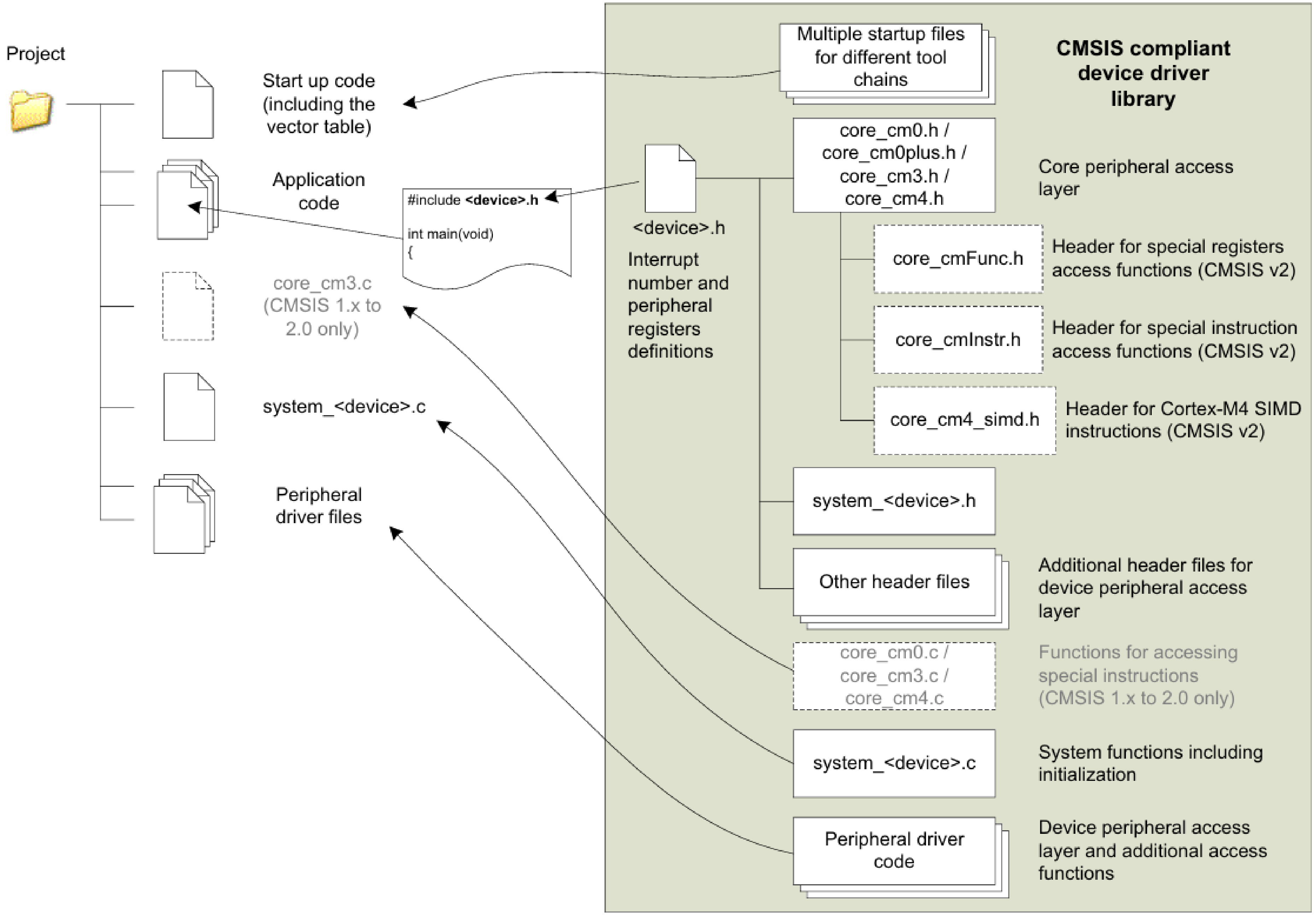
Project
Start up code (including the vector table)
Application code
core_cm3.c (CMSIS 1.x to 2.0 only)
system_<device>.c
Peripheral driver files
#include <device>.h
int main(void)
{
<device>.h
Interrupt number and peripheral registers definitions
CMSIS compliant device driver library
Multiple startup files for different tool chains
core_cm0.h / core_cm0plus.h / core_cm3.h / core_cm4.h
Core peripheral access layer
core_cmFunc.h
Header for special registers access functions (CMSIS v2)
core_cmInstr.h
Header for special instruction access functions (CMSIS v2)
core_cm4_simd.h
Header for Cortex-M4 SIMD instructions (CMSIS v2)
system_<device>.h
Other header files
Additional header files for device peripheral access layer
core_cm0.c / core_cm3.c / core_cm4.c
Functions for accessing special instructions (CMSIS 1.x to 2.0 only)
system_<device>.c
System functions including initialization
Peripheral driver code
Device peripheral access layer and additional access functions

FIGURE 2.14

Using CMSIS-Core in a project

A typical project setup using a CMSIS device-driver package

- Inside the device-driver package obtained from the microcontroller vendor, including the CMSIS generic files.
- The names of some of these files depend on the actual microcontroller device name chosen by the microcontroller vendor (indicated as in the diagram).
- When the device-specific header file is included in the application code, it automatically includes additional header files, therefore you need to set up the project search path for the header files in order to compile the project correctly.

Benefits of CMSIS-Core

- The main advantage is much better software portability and reusability:
- It can be migrated to another device from the same vendor with a different Cortex-M processor very easily.
- CMSIS-Core made it easier for a Cortex-M microcontroller project to be migrated to another device from a different vendor.
- CMSIS allows software to be much more future proof because embedded soft ware developed today can be reused on other Cortex-M products in the future.
- The CMSIS-Core also allows faster time to market.

- It is easier to reuse software code from previous projects.
- All CMSIS-compliant device drivers have a similar structure, learning to use a new Cortex-M microcontroller is much easier.
- The CMSIS code has been tested by many silicon vendors and software developers around the world. It is compliant with Motor Industry Software Reliability Association (MISRA).
- It reduces the validation effort required, as there is no need to develop and test your own processor feature access functions.
- Starting from CMSIS 2.0, a DSP library is included that provides tested, opti mized DSP functions. The DSP library code is available as a free download and can be used by software developers free of charge.

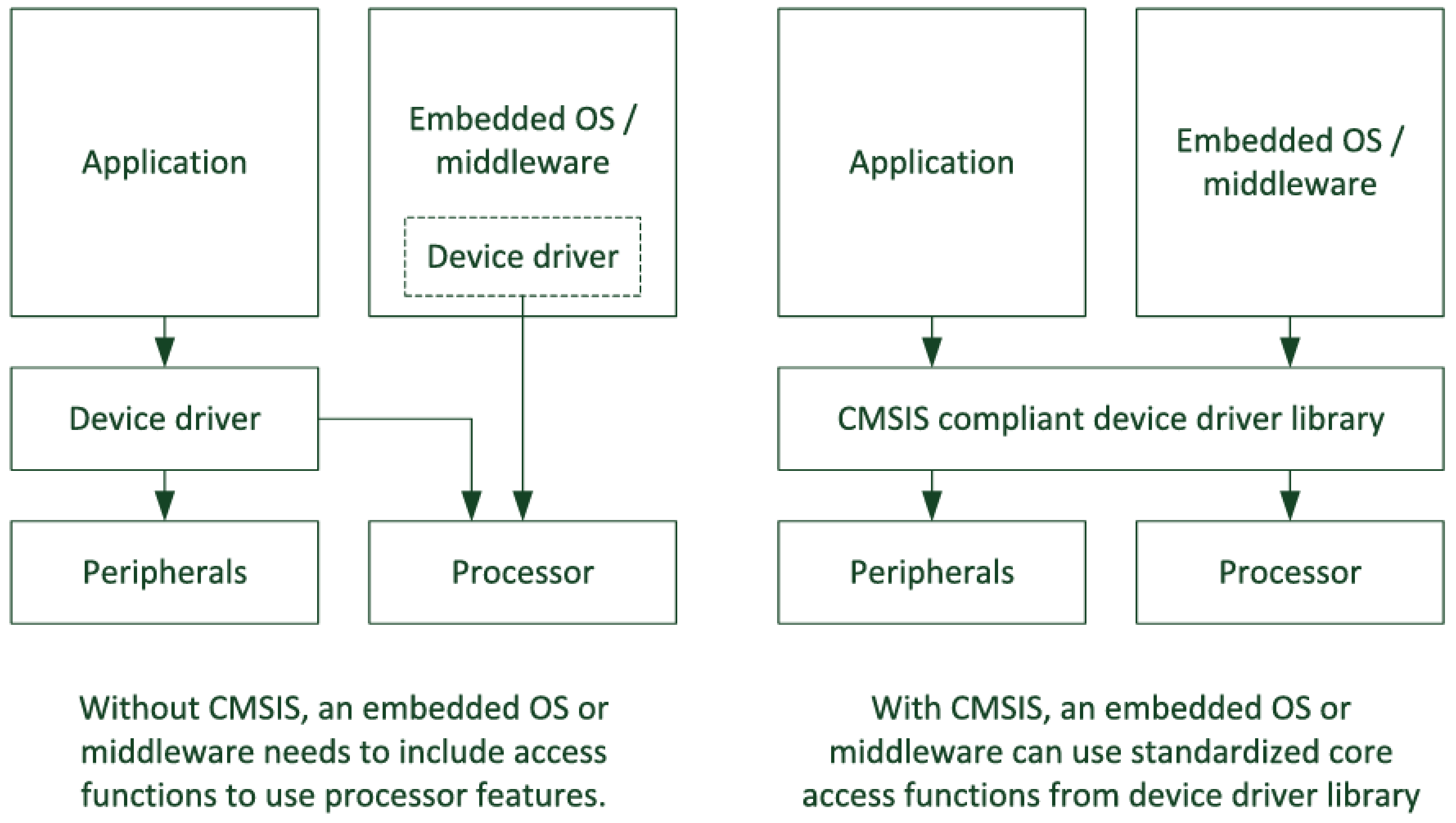
Application
Embedded OS / middleware
Device driver
Device driver
Peripherals
Processor
Without CMSIS, an embedded OS or middleware needs to include access functions to use processor features.
Application
Embedded OS / middleware
CMSIS compliant device driver library
Peripherals
Processor
With CMSIS, an embedded OS or middleware can use standardized core access functions from device driver library

FIGURE 2.15

CMSIS-Core avoids the need for middleware or OS to carry their own driver code

Advantage of CMSIS

- By using processor core access functions from CMSIS, embedded OS, and middleware can work with device-driver libraries from various microcontroller vendors, including future products that are yet to be released.
- Since CMSIS is designed to work with various toolchains, many software products can be designed to be toolchain independent.
- Without CMSIS, middleware might need to include a small set of driver func tions for accessing processor peripherals such as the interrupt controller.
- Such an arrangement increases the program size, and might cause compatibility issues with other software products.

- CMSIS is supported by multiple compiler toolchain vendors.
- CMSIS has a small memory footprint (less than 1KB for all core access functions and a few bytes of RAM for several variables).

thank you
danke
謝謝
ngiyabonga
teşekkür ederim
gracias
merci
go raibh maith agat
obrigado
спасибо
dziękuję
bedankt
sukriya
kop khun krap
mochchakkeram
tapadh leat
terima kasih
감사합니다
ありがとう
arigatô
grazie
dakujem
takk
мерси
рахмат
Баярлалаа
dank je
mersi
kia ora
barka
welalin
tack
spas
faafetai lava
vinaka
спасибі
blagodaram
misaotra
matondo
paldies
grazzi
mahalo
хвала
asante
manana
obrigada
tenki
murakoze
chokrane
djiere dieuf
tau
дякую
mamnun
sulpay
taiku
chnorakaloutioun
gratias ago
gracies
akun
dankon
aciū
kiitos
dankie
nanni
nandri
dhanyavad
hvala
mauruuru
köszönöm
bayarlalaa
gracie
enkosi
sobodi
dėkuji
mèsi
didi madloba
sagolun
najis tuke
kam sah hamnida
rahmat
তোমাকে ধন্যবাদ
xiexie
tanemirt
rahmet
ευχαριστώ
diolch
dhanyavadagalu
shukriya
merce
trugarez