MODULE -1

ARM MICROCONTROLLER

The microcontroller market is vast, with more than 20 billion devices per year estimated to be shipped in 2010. A bewildering array of vendors, devices, and architectures is competing in this market. The requirement for higher performance microcontrollers has been driven globally by the industry's changing needs; for example, microcontrollers are required to handle more work without increasing a product's frequency or power. In addition, microcontrollers are becoming increasingly connected, whether by Universal Serial Bus (USB), Ethernet, or wireless radio, and hence, the processing needed to support these communication channels and advanced peripherals are growing.

1.1 INTRODUCTION

The ARM Cortex™-M3 processor, the first of the Cortex generation of processors released by ARM in 2006, was primarily designed to target the 32-bit microcontroller market. The Cortex-M3 processor provides excellent performance at low gate count and comes with many new features previously available only in high-end processors.

The Cortex-M3 addresses the requirements for the 32-bit embedded processor market in the following ways:

Greater performance efficiency: allowing more work to be done without increasing the frequency or power requirements.

Low power consumption: enabling longer battery life, especially critical in portable products including wireless networking applications.

Enhanced determinism: guaranteeing that critical tasks and interrupts are serviced as quickly as possible and in a known number of cycles.

Improved code density: ensuring that code fits in even the smallest memory footprints.

Ease of use: providing easier programmability and debugging for the growing number of 8-bit and 16-bit users migrating to 32 bits.

Lower cost solutions: reducing 32-bit-based system costs close to those of legacy 8-bit and 16-bit devices and enabling low-end, 32-bit microcontrollers to be priced at less than US$1 for the first time.

Wide choice of development tools: from low-cost or free compilers to full-featured development suites from many development tool vendors.

1.1.1 BACKGROUND OF ARM AND ARM ARCHITECTURE

ARM was formed in 1990 as Advanced RISC Machines Ltd., a joint venture of Apple Computer, Acorn Computer Group, and VLSI Technology. In 1991, ARM introduced the ARM6 processor family, and VLSI became the initial licensee. Subsequently, additional companies, including Texas Instruments, NEC, Sharp, and ST Microelectronics, licensed the ARM processor designs, extending the applications of ARM processors

into mobile phones, computer hard disks, personal digital assistants (PDAs), home entertainment systems, and many other consumer products.

THE CORTEX-M3 PROCESSOR VERSUS CORTEX-M3-BASED MCUs

The Cortex-M3 processor is the central processing unit (CPU) of a microcontroller chip. In addition, a number of other components are required for the whole Cortex-M3 processor-based microcontroller. After chip manufacturers license the Cortex-M3 processor, they can put the Cortex-M3 processor in their silicon designs, adding memory, peripherals, input/output (I/O), and other features. Cortex-M3 processor-based chips from different manufacturers will have different memory sizes, types, peripherals, and features. This book focuses on the architecture of the processor core. For details about the rest of the chip, readers are advised to check the particular chip manufacturer's documentation.

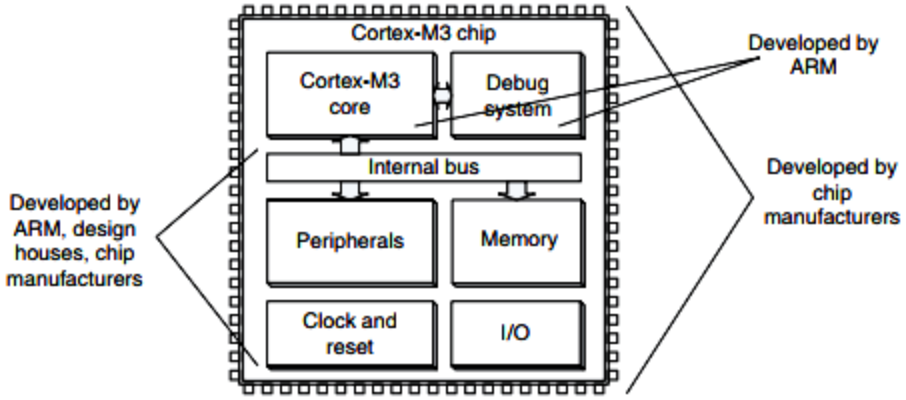


FIGURE 1.1

The Cortex-M3 Processor versus the Cortex-M3-Based MCU.

WHAT ARE ARM CORTEX M SERIES MICROCONTROLLER?

The Cortex-M3 and Cortex-M4 processors use a 32-bit architecture. Internal registers in the register bank, the data path, and the bus interfaces are all 32 bits wide. The Instruction Set Architecture (ISA) in the Cortex-M processors is called the Thumb ISA and is based on Thumb-2 Technology which supports a mixture of 16-bit and 32-bit instructions. The Cortex-M3 and Cortex-M4 processors have:

• Three-stage pipeline design

• Harvard bus architecture with unified memory space: instructions and data use the same address space

• 32-bit addressing, supporting 4GB of memory space

• On-chip bus interfaces based on ARM AMBA (Advanced Microcontroller Bus Architecture) Technology, which allow pipelined bus operations for higher throughput

• An interrupt controller called NVIC (Nested Vectored Interrupt Controller) supporting up to 240 interrupt requests and from 8 to 256 interrupt priority levels (dependent on the actual device implementation)

• Support for various features for OS (Operating System) implementation such as a system tick timer, shadowed stack pointer

• Sleep mode support and various low power features

• Support for an optional MPU (Memory Protection Unit) to provide memory protection features like programmable memory, or access permission control
• Support for bit-data accesses in two specific memory regions using a feature called Bit Band
• The option of being used in single processor or multi-processor designs

The ISA used in Cortex-M3 and Cortex-M4 processors provides a wide range of instructions:

• General data processing, including hardware divide instructions
• Memory access instructions supporting 8-bit, 16-bit, 32-bit, and 64-bit data, as well as instructions for transferring multiple 32-bit data
• Instructions for bit field processing
• Multiply Accumulate (MAC) and saturate instructions
• Instructions for branches, conditional branches and function calls
• Instructions for system control, OS support, etc.

In addition, the Cortex-M4 processor also supports:

• Single Instruction Multiple Data (SIMD) operations
• Additional fast MAC and multiply instructions

DIFFERENCES BETWEEN MICROPROCESSOR AND MICROCONTROLLER

ARM does not make microcontrollers. ARM designs processors and various com ponents that silicon designers need and licenses these designs to various silicon design companies including microcontroller vendors. Typically, we call these designs “Intellectual Property” (IP) and the business model is called IP licensing.

In a typical microcontroller design, the processor takes only a small part of the sil icon area. The other areas are taken up by memories, clock generation (e.g., PLL) and distribution logic, system bus, and peripherals (hardware units like I/O interface units, communication interface, timers, ADC, DAC, etc.) as shown in Figure 1.2.

Although many microcontroller vendors use ARM Cortex-M processors as their choice of CPU, the memory system, memory map, peripherals, and operation characteristics (e.g., clock speed and voltage) can be completed differently from one product to another. This allows microcontroller manufacturers to add additional features in their products and differentiate their products from others on the market.

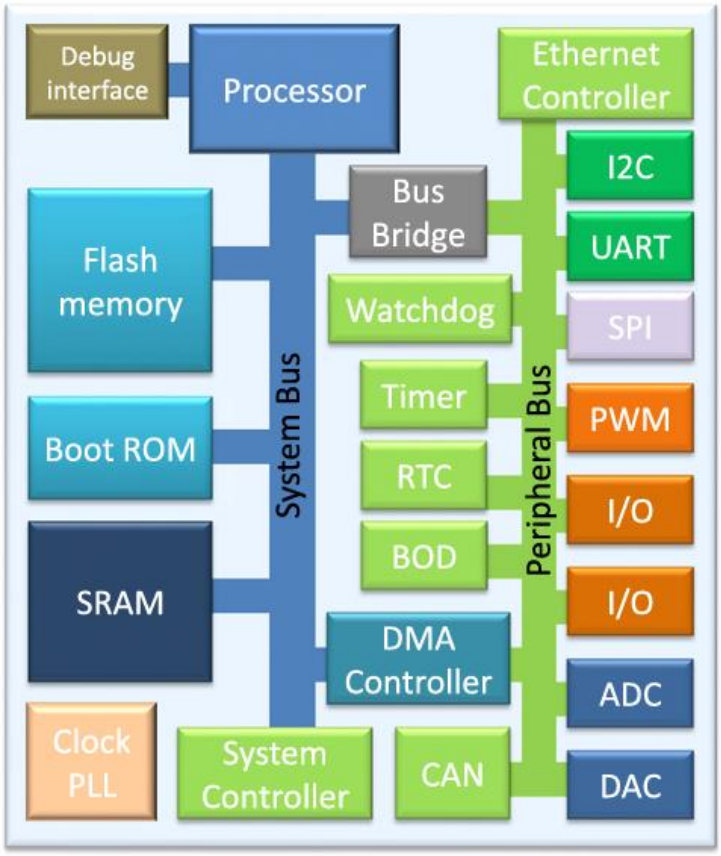


FIGURE 1.2

A microcontroller contains many different blocks

ARM and the Microcontroller vendors

Currently there are more than 15 silicon vendors using ARM Cortex-M3 or Cortex-M4 processors in microcontroller products. There are also some other companies that use Cortex-M3 or Cortex-M4 for SoC designs, others companies that use only Cortex-M0 or Cortex-M0+ processors.

(Current Cortex-M3/M4 microcontroller vendors include: Analog Devices, Atmel, Cypress, EnergyMicro, Freescale, Fujitsu, Holtek, Infineon, Microsemi, Milandr, NXP, Samsung, Silicon Laboratories, ST Microelectronics, Texas Instrument, and Toshiba.)

After a company licenses the Cortex-M processor design, ARM provides the design source code of the processor in a language called Verilog-HDL (Hardware Description Language). The design engineers in these companies then add their own design blocks like peripherals and memories, and use various EDA tools to convert the whole design from Verilog-HDL and various other forms into a transistor level chip layout. ARM also provides other Intellectual Property (IP) products, and some can be used by these companies in their microcontroller products (see Figure 1.3).

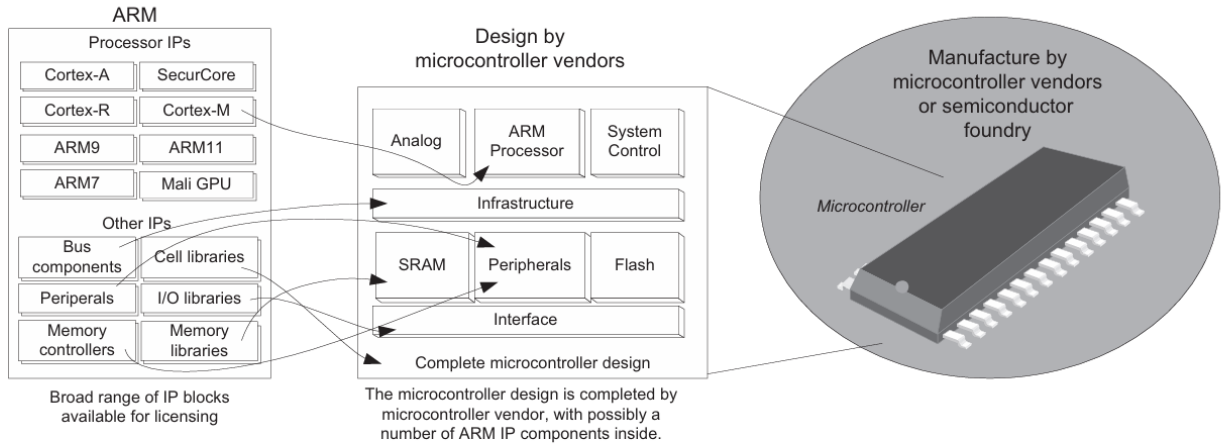


FIGURE 1.3

A microcontroller might contain multiple ARM IP products

For example:

• Design of the cell libraries such as logic gates and memories (ARM Physical IP products)

• Peripherals and AMBA infrastructure components (Cortex-M System Design Kit (CMSDK), ARM CoreLink IP products)

• Additional debug components for linking debug systems in multi-processor design (ARM CoreSight IP products)

For example, ARM provides a product called the Cortex-M System Design Kit (CMSDK), a design kit for Cortex-M processor with AMBA infrastructure components, baseline peripherals, example systems, and example software. This allows chip designers to start using the Cortex-M processors quickly and reduces the total chip development effort with reusable IP.

ARM has various software development platforms such as the Keil Microcontroller Development Kit (MDK-ARM) and ARM Development Studio 5 (DS-5). These software development suites contain compilers, debuggers, and instruction set simulators.

SELECTING CORTEX M3 and M4 MICROCONTROLLER

There are many factors to be considered when selecting a microcontroller device for a product. For example:

• Peripherals and interface features

• Memory size requirements of the application

• Low power requirements

• Performance and maximum frequency

• Chip package

• Operation conditions (voltage, temperature, electromagnetic interference)

• Cost and availability

- Software development tool support and development kits
- Future upgradability
- Firmware packages and firmware security
- Availability of application notes, design examples, and support

There are no golden rules on how to select the best microcontroller. All of the factors depend on your target applications as well as your project's situation. Some of the factors, like cost and product availability, might vary from time to time.

ADVANTAGES

Low power

Compared to other 32-bit processor designs, Cortex-M processors are relatively small. The Cortex-M processor designs are also optimized for low power consumption. In addition, the Cortex-M processors also include support for sleep mode features and can be used with various advanced ultra-low power design technologies. All these allow the Cortex-M processors to be used in various ultra-low power microcontroller products.

Performance

The Cortex-M3 and Cortex-M4 processors can deliver over 3 CoreMark/MHz and 1.25 DMIPS/MHz (based on the Dhrystone 2.1 benchmark). This allows Cortex-M3 and Cortex-M4 microcontrollers to handle many complex and demanding applications with a much slower clock speed to reduce power consumption.

Energy efficiency

Combining low power and high-performance characteristics, the Cortex-M3 and Cortex-M4 processors have excellent energy efficiency with a limited supply of energy to stay in sleep mode for longer durations of time, enabling longer battery life in portable products.

Code density

The Thumb ISA provides excellent code density. This means that to achieve the same tasks, you need a smaller program size. It reduces cost and power consumption by using a microcontroller with smaller flash memory size, and chip manufacturers can produce microcontroller chips with smaller packages.

Interrupts

The Cortex-M3 and Cortex-M4 processors have a configurable interrupt controller design, which can support up to 240 vectored interrupts and multiple levels of interrupt priorities (from 8 to 256 levels). The interrupt processing capability makes the Cortex-M processors suitable for many real-time control applications.

Ease of use, C friendly

The Cortex-M processors are very easy to use. In fact, they are easier than compared to many 8-bit processors because Cortex-M processors have a simple, linear memory map, and there are no special architectural restrictions, which you often find in 8-bit microcontrollers (e.g., memory banking, limited stack levels, non-re-entrant code, etc.).

Scalability

The Cortex-M processor family allows easy scaling of designs from low-cost, simple microcontrollers costing less than a dollar to high-end microcontrollers. Due to the consistency of the processor architecture, you only need one tool chain and you can reuse your software easily.

Debug

The Cortex-M processors include many debug features that allow you to analyze design problems easily. In multiple processor designs, the debug system of each Cortex-M processor can be linked together to share debug connections.

OS support

The Cortex-M processors are designed with OS applications in mind. A number of features are available to make OS implementation easier and make OS operations more efficient. Currently there are over 30 embedded OSs available for Cortex-M processors. **Versatile system**

The Cortex-M3 and Cortex-M4 processors support a number of system features such as bit addressable memory range (bit band feature) and MPU (Memory Protection Unit).

Software portability and reusability

Since the architecture is very C friendly, you can program almost everything in stan dard ANSI C. One of ARM's initiatives called CMSIS makes programming for Cortex-M processor based products even easier by providing standard header files and an API for standard Cortex-M processor functions.

This allows better software reusability and also makes porting application code easier.

Choices (devices, tools, OS, etc.) One of the best things about using Cortex-M microcontrollers number amount of available choices. Besides the thousands of microcontroller devices available, you also have a wide range of coins on software development/debug tools, embedded OS, middleware, etc

APPLICATIONS OF ARM CORTEX M3:

Microcontrollers:

The Cortex-M processor family is ideally suited for micro controller products. This includes low-cost microcontrollers with small memory sizes and high performance microcontrollers with high operation speeds. These microcontrollers can be used in consumer products, from toys to electrical appliances, or even specialized products for Information Technology (IT), industrial, or even medical systems.

Automotive:

Another application for the Cortex-M3 and Cortex-M4 processors is in the automotive industry. As these processors offer great performance, very high energy efficiency, and low interrupt latency, they are ideal for many real-time control systems. In addition, the flexibility of the processor design (e.g., it supports up to 240 interrupt sources, optional MPU) makes it ideal for highly integrated ASSPs (Application Specific Standard Products) for the automotive industry. The MPU feature also provides robust memory protection, which is required in some of these applications.

Data communications:

The processor's low power and high efficiency, coupled with instructions in Thumb-2 for bit-field manipulation, make the Cortex-M3 and Cortex-M4 processors ideal for many communication applications, such as Bluetooth and ZigBee.

Industrial control:

In industrial control applications, simplicity, fast response, and reliability are key factors. Again, the interrupt support features on Cortex-M3 and Cortex-M4 processors, including their deterministic behavior, automatic nested interrupt handling, MPU, and enhanced fault-handling, make them strong candidates in this area.

Consumer products:

In many consumer products, a high-performance microprocessor (or several) is used. The Cortex-M3 and Cortex-M4 processors, being small, are highly efficient and low in power, and at the same time provide the performance required for handling complex GUIs on LCD panels and various communication protocols.

Systems-on-Chips (SoC):

In some high-end application processor designs, Cortex-M processors are used in various subsystems such as audio processing engines, power management systems, FSM (Finite State Machine) replacement, I/O control task off loading, etc. Mixed signal designs: In the IC design world, the digital and analog designs are converging. While microcontrollers contain more and more analogue components (e.g., ADC, DAC), some analog ICs such as sensors, PMIC (Power Management IC), and MEMS (Microelectromechanical Systems) now

also include processors to provide additional intelligence. The low power capability and small gate count characteristics of the Cortex-M processors make it possible for them to be integrated on mixed signal IC designs

1.1.2 ARCHITECTURE VERSIONS

Over the years, ARM has continued to develop new processors and system blocks. These include the popular ARM7TDMI processor and, more recently, the ARM1176TZ(F)-S processor, which is used in high-end applications such as smart phones. The evolution of features and enhancements to the processors over time has led to successive versions of the ARM architecture. Note that architecture version numbers are independent from processor names. For example, the ARM7TDMI processor is based on the ARMv4T architecture (the T is for Thumb® instruction mode support).

Over the past several years, ARM extended its product portfolio by diversifying its CPU development, which resulted in the architecture version 7 or v7. In this version, the architecture design is divided into three profiles:

• The **A profile** is designed for high-performance open application platforms.

• The **R profile** is designed for high-end embedded systems in which real-time performance is needed.

• The **M profile** is designed for deeply embedded microcontroller-type systems.

• **A Profile (ARMv7-A)**: Application processors which are designed to handle complex applications such as high-end embedded operating systems (OSs) (e.g., Symbian, Linux, and Windows Embedded). These processors requiring the highest processing power, virtual memory system support with memory management units (MMUs), and, optionally, enhanced Java support and a secure program execution environment. Example products include high-end mobile phones and electronic wallets for financial transactions.

• **R Profile (ARMv7-R)**: Real-time, high-performance processors targeted primarily at the higher end of the real-time1 market—those applications, such as high-end breaking systems and hard drive controllers, in which high processing power and high reliability are essential and for which low latency is important.

• **M Profile (ARMv7-M)**: Processors targeting low-cost applications in which processing efficiency is important and cost, power consumption, low interrupt latency, and ease of use are critical, as well as industrial control applications, including real-time control systems.

1.2 THUMB-2 TECHNOLOGY

The Thumb-23 technology extended the Thumb Instruction Set Architecture (ISA) into a highly efficient and powerful instruction set that delivers significant benefits in terms of ease of use, code size, and performance. The extended instruction set in Thumb-2 is a superset of the previous 16-bit Thumb instruction set, with additional 16-bit instructions alongside 32-bit instructions. It allows more complex operations to be carried out in the Thumb state, thus allowing higher efficiency by reducing the number of states switching between ARM state and Thumb state. Focused on small memory system devices such as microcontrollers and reducing the size of the processor, the Cortex-M3 supports only the Thumb-2 (and traditional Thumb)

instruction set. Instead of using ARM instructions for some operations, as in traditional ARM processors, it uses the Thumb-2 instruction set for all operations. As a result, the Cortex-M3 processor is not backward compatible with traditional ARM processors. Nevertheless, the Cortex-M3 processor can execute almost all the 16-bit Thumb instructions, including all 16-bit Thumb instructions supported on ARM7 family processors, making application porting easy. With support for both 16-bit and 32-bit instructions in the Thumb-2 instruction set, there is no need to switch the processor between Thumb state (16-bit instructions) and ARM state (32-bit instructions).

For example, in ARM7 or ARM9 family processors, you might need to switch to ARM state if you want to carry out complex calculations or a large number of conditional operations and good performance is needed, whereas in the Cortex-M3 processor, you can mix 32-bit instructions with 16-bit instructions without switching state, getting high code density and high performance with no extra complexity.

The Thumb-2 instruction set is a very important feature of the ARMv7 architecture. Compared with the instructions supported on ARM7 family processors (ARMv4T architecture), the Cortex-M3 processor instruction set has a large number of new features. For the first time, hardware divide instruction is available on an ARM processor, and a number of multiply instructions are also available on the Cortex-M3 processor to improve data-crunching performance. The Cortex-M3 processor also supports unaligned data accesses, a feature previously available only in high-end processors.

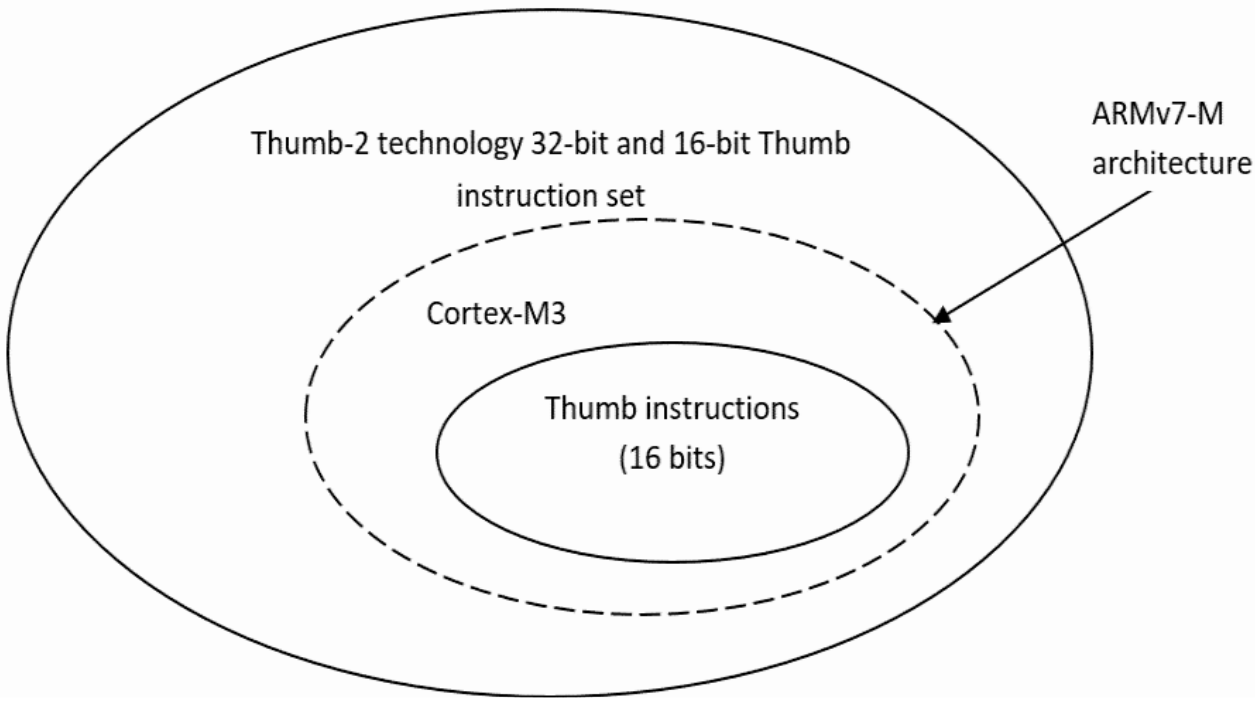


Fig1. The Relationship between the Thumb Instructions Set in Thumb-2 Technology and the Traditional Thumb

1.3 Cortex-M3 Processor Applications

With its high performance and high code density and small silicon footprint, the Cortex-M3 processor is ideal for a wide variety of applications:

- **Low-cost microcontrollers:** The Cortex-M3 processor is ideally suited for low-cost microcontrollers, which are commonly used in consumer products, from toys to electrical appliances.
  It is a highly competitive market due to the many well-known 8-bit and 16-bit microcontroller products on the market. Its lower power, high performance, and ease-of-use advantages enable embedded developers to migrate to 32-bit systems and develop products with the ARM architecture.
- **Automotive:** Another ideal application for the Cortex-M3 processor is in the automotive industry. The Cortex-M3 processor has very high-performance efficiency and low interrupt latency, allowing it to be used in real-time systems.
  The Cortex-M3 processor supports up to 240 external vectored interrupts, with a built-in interrupt controller with nested interrupt supports and an optional MPU, making it ideal for highly integrated and cost-sensitive automotive applications.
- **Data communications:** The processor's low power and high efficiency, coupled with instructions in Thumb-2 for bit-field manipulation, make the Cortex-M3 ideal for many communications applications, such as Bluetooth and ZigBee.
- **Industrial control:** In industrial control applications, simplicity, fast response, and reliability are key factors.
  Cortex-M3 processor's interrupt feature, low interrupt latency, and enhanced fault-handling features make it a strong candidate in this area.
- **Consumer products:** In many consumer products, a high-performance microprocessor (or several of them) is used. The Cortex-M3 processor, being a small processor, is highly efficient and low in power and supports an MPU enabling complex software to execute while providing robust memory protection.

1.3.1 FUNDAMENTALS OF CORTEX M3

The Cortex™-M3 is a 32-bit microprocessor. It has a 32-bit data path, a 32-bit register bank, and 32-bit memory interfaces (see Figure 2). The processor has a Harvard architecture, which means that it has a separate instruction bus and data bus. This allows instructions and data accesses to take place at the same time, and as a result of this, the performance of the processor increases because data accesses do not affect the instruction pipeline. This feature results in multiple bus interfaces on Cortex-M3, each with optimized usage and the ability to be used simultaneously. The instruction and data buses share the same memory space called as unified memory system. *(In other words, you cannot get 8 GB of memory space just because you have separate bus interfaces.)*

1.4 ARCHITECTURE OF ARM CORTEX M3

The Cortex-M3 processor is a 32-bit processor, with a 32-bit wide data path, register bank and memory interface. There are 13 general-purpose registers, two stack pointers, a link register, a program counter and a number of special registers including a program status register.

The Cortex-M3 core contains a decoder for traditional Thumb and new Thumb-2 instructions, an advanced ALU with support for hardware multiply and divide, control logic, and interfaces to the other components of the processor.

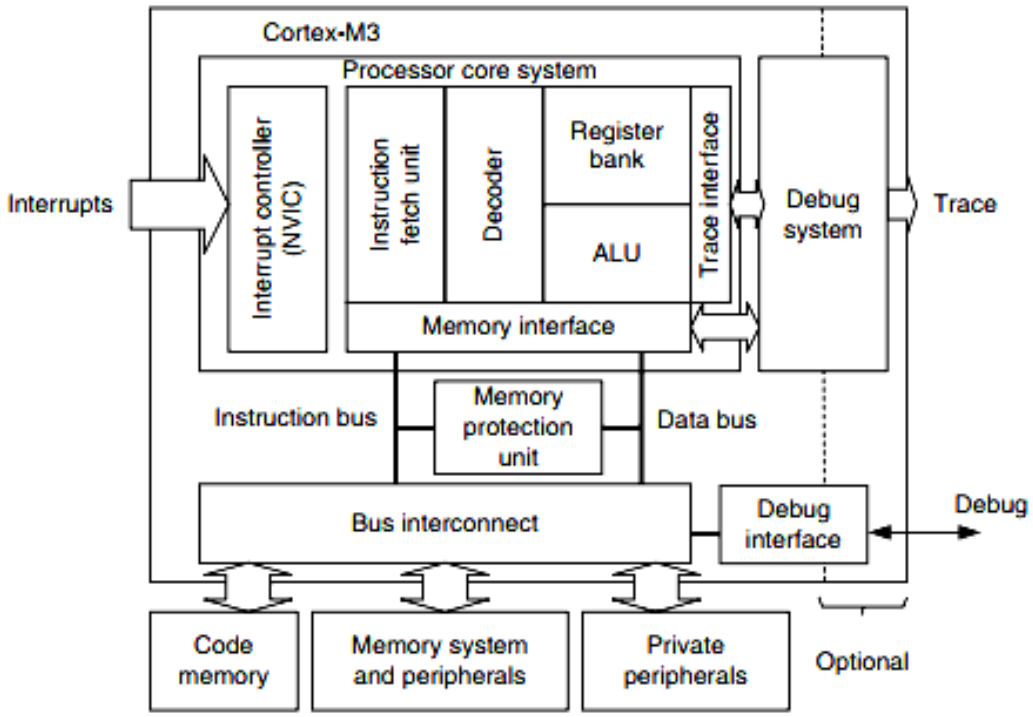


Fig2. Simplified view of Arm Cortex M3 architecture.

The Cortex-M3 processor is a 32-bit processor, with a 32-bit wide data path, register bank and memory interface. There are 13 general-purpose registers, two stack pointers, a link register, a program counter and a number of special registers including a program status register.

The Cortex-M3 processor is a memory mapped system with a simple, fixed memory map for up to 4 gigabytes of addressable memory space with predefined, dedicated addresses for code (code space), SRAM(memory space), external memories/devices and internal/external peripherals. There is also a special region to provide for vendor specific addressability.

The MPU is an optional component of the Cortex-M3 processor that can improve the reliability of an embedded system by protecting critical data used by the operating system from user applications, separating processing tasks by disallowing access to each other's data, disabling access to memory regions, allowing memory regions to be defined as read-only and detecting unexpected memory accesses that could potentially break the system.

The highly configurable NVIC is an integral part of the Cortex-M3 processor and provides the processor's outstanding interrupt handling abilities. In its standard implementation it supplies a Non Maskable Interrupt (NMI) and 32 general purpose physical interrupts with 8 levels of pre-emption priority. It can be configured to anywhere between 1 and 240 physical interrupts with up to 256 levels of priority though simple synthesis choices.

The debug access into a Cortex-M3 processor based system is through the Debug Access Port (DAP) that can be implemented as either a Serial Wire Debug Port (SW-DP) for a two-pin (clock and data) Interface or a Serial Wire JTAG Debug Port (SWJ-DP) that enables either JTAG or SW protocol to be used. The SWJ-DP

defaults to JTAG mode on power reset and can be made to switch protocols with a specific control sequence provided by the external debug hardware.

The Cortex-M3 processor bus matrix connects the processor and debug interface to the external buses; the 32-bit AMBA® AHB-Lite based ICode, DCode and System interfaces and the 32-bit AMBA APB™ based Private Peripheral Bus (PPB). The bus matrix also implements unaligned data accesses and bit banding.

1.5 REGISTERS

The Cortex-M3 processor has registers R0 through R15 (see figure 3). R13 (the stack pointer) is banked, with only one copy of the R13 visible at a time.

R0–R12: General-Purpose Registers R0–R12 are 32-bit general-purpose registers for data operations. Some 16-bit Thumb® instructions can only access a subset of these registers (low registers, R0–R7).

R13: Stack Pointers. The Cortex-M3 contains two stack pointers (R13). They are banked so that only one is visible at a time.
The two stack pointers are as follows:
• Main Stack Pointer (MSP): The default stack pointer, used by the operating system (OS) kernel and exception handlers
• Process Stack Pointer (PSP): Used by user application code.

R14: The Link Register When a subroutine is called, the return address is stored in the link register.

R15: The Program Counter The program counter is the current program address. This register can be written to control the program flow.

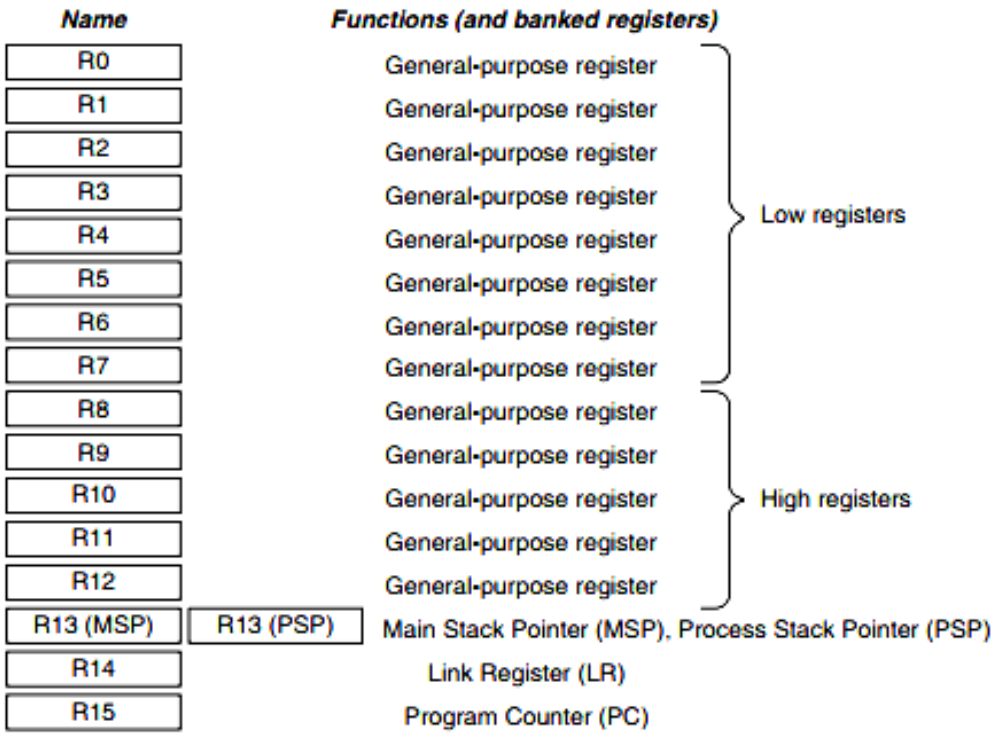


Fig. 3 Registers in the Cortex-M3

1.5.1 Stack Pointer R13

R13 is the stack pointer (SP). In the Cortex-M3 processor, there are two SPs. This duality allows two separate stack memories to be set up. When using the register name R13, you can only access the current SP; the other one is inaccessible unless you use special instructions to move to special register from general-purpose register (MSR) and move special register to general-purpose register (MRS).

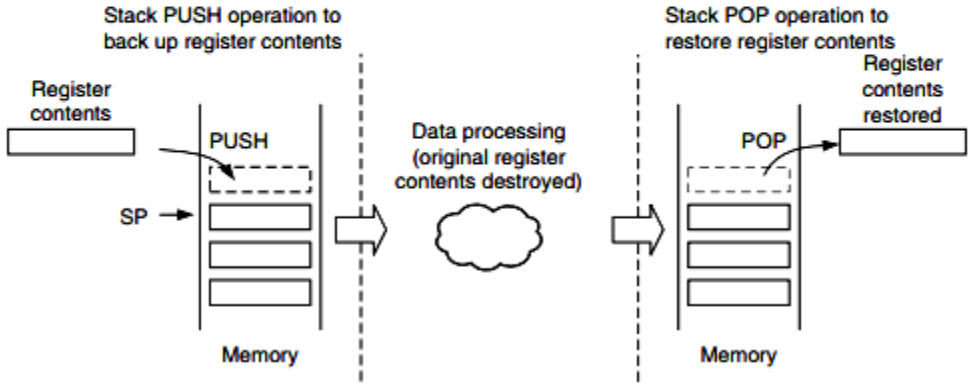
The two SPs are as follows:

• **Main Stack Pointer (MSP) or SP_main** in ARM documentation: This is the default SP; it is used by the operating system (OS) kernel, exception handlers, and all application codes that require privileged access.

• **Process Stack Pointer (PSP) or SP_process** in ARM documentation: This is used by the base-level application code (when not running an exception handler).

STACK PUSH AND POP

Stack is a memory usage model. It is simply part of the system memory, and a pointer register (inside the processor) is used to make it work as a first-in/last-out buffer. The common use of a stack is to save register contents before some data processing and then restore those contents from the stack after the processing task is done.



In the Cortex-M3, the instructions for accessing stack memory are PUSH and POP. The assembly language syntax is as follows (text after each semicolon [;] is a comment):

```
PUSH {R0}    ; R13=R13-4, then Memory[R13] = R0
POP {R0}     ; R0 = Memory[R13], then R13 = R13 + 4
```

1.5.2 Link Register R14

R14 is the link register (LR). Inside an assembly program, you can write it as either R14 or LR. LR is used to store the return program counter (PC) when a subroutine or function is called—for example, when you're using the branch and link (BL) instruction:

```
main : Main program
     ...
     BL function1 ; Call function1 using Branch with Link instruction.
                  ; PC = function1 and
                  ; LR = the next instruction in main
     ...
function1
     ...          ; Program code for function 1
     BX LR        ; Return
```

1.5.3 Program Counter R15

R15 is the PC. You can access it in assembler code by either R15 or PC. Because of the pipelined nature of the Cortex-M3 processor, when you read this register, you will find that the value is different than the location of the executing instruction, normally by 4.

0x1000 : MOV R0, PC ; R0 = 0x1004

In other instructions like literal load (reading of a memory location related to current PC value), the effective value of PC might not be instruction address plus 4 due to alignment in address calculation. But the PC value is still at least 2 bytes ahead of the instruction address during execution.

1.6 Special Registers

The Cortex-M3 processor also has a number of special registers. They are as follows:

• Program Status registers (PSRs)

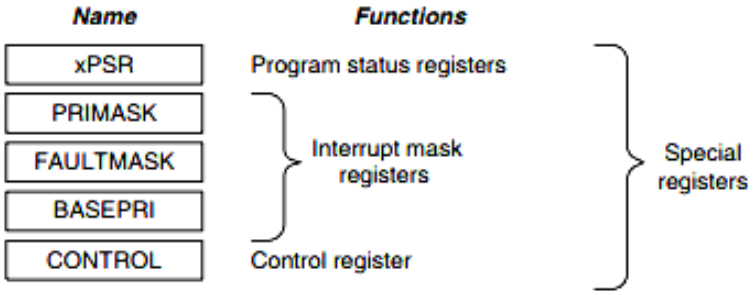
• Interrupt Mask registers (PRIMASK, FAULTMASK, and BASEPRI)

• Control register (CONTROL) These registers have special functions and can be accessed only by special instructions. They cannot be used for normal data processing.

```
MRS <reg>, <special_reg>; Read special register
MSR <special_reg>, <reg>; write to special register
```

MRS , ; move from special register to register

MSR , ; move register to special register



Register	Function
xPSR	Provide arithmetic and logic processing flags (zero flag and carry flag), execution status, and current executing interrupt number
PRIMASK	Disable all interrupts except the nonmaskable interrupt (NMI) and hard fault
FAULTMASK	Disable all interrupts except the NMI
BASEPRI	Disable all interrupts of specific priority level or lower priority level
CONTROL	Define privileged status and stack pointer selection

1.6.1 PROGRAM STATUS REGISTERS

The PSRs are subdivided into three status registers:

1. Application Program Status register (APSR)
2. Interrupt Program Status register (IPSR)
3. Execution Program Status register (EPSR)

The three PSRs can be accessed together or separately using the special register access instructions MSR and MRS. When they are accessed as a collective item, the name xPSR is used. You can read the PSRs using the MRS instruction. You can also change the APSR using the MSR instruction, but EPSR and IPSR are read-only.

For example:

MRS r0, APSR ; Read Flag state into R0

MRS r0, IPSR ; Read Exception/Interrupt state

MRS r0, EPSR ; Read Execution state

MSR APSR, r0 ; Write Flag state

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
APSR	N	Z	C	V	Q											
IPSR												Exception number				
EPSR						ICI/IT	T			ICI/IT						

Fig4. Program Status Registers (PSRs) in the Cortex-M3.

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
xPSR	N	Z	C	V	Q	ICI/IT	T			ICI/IT		Exception number				

Fig5. Combined Program Status Registers (xPSR) in the Cortex-M3.

Table 3.1 Bit Fields in Cortex-M3 Program Status Registers

Bit	Description
N	Negative
Z	Zero
C	Carry/borrow
V	Overflow
Q	Sticky saturation flag
ICI/IT	Interrupt-Continuable Instruction (ICI) bits, IF-THEN instruction status bit
T	Thumb state, always 1; trying to clear this bit will cause a fault exception
Exception number	Indicates which exception the processor is handling

1.6.2 PRIMASK, FAULTMASK, AND BASEPRI REGISTERS

The PRIMASK and BASEPRI registers are useful for temporarily disabling interrupts in timing-critical tasks. An OS could use FAULTMASK to temporarily disable fault handling when a task has crashed. In this scenario, a number of different faults might be taking place when a task crashes. Once the core starts cleaning up, it might not want to be interrupted by other faults caused by the crashed process. Therefore, the FAULTMASK gives the OS kernel time to deal with fault conditions.

Table 3.2 Cortex-M3 Interrupt Mask Registers

Register Name	Description
PRIMASK	A 1-bit register, when this is set, it allows nonmaskable interrupt (NMI) and the hard fault exception; all other interrupts and exceptions are masked. The default value is 0, which means that no masking is set.
FAULTMASK	A 1-bit register, when this is set, it allows only the NMI, and all interrupts and fault handling exceptions are disabled. The default value is 0, which means that no masking is set.
BASEPRI	A register of up to 8 bits (depending on the bit width implemented for priority level). It defines the masking priority level. When this is set, it disables all interrupts of the same or lower level (larger priority value). Higher priority interrupts can still be allowed. If this is set to 0, the masking function is disabled (this is the default).

1.6.3 CONTROL REGISTER

In the Cortex-M3, the CONTROL [1] bit is always 0 in handler mode. However, in the thread or base level, it can be either 0 or 1.

Table 3.3 Cortex-M3 Control Register

Bit	Function
CONTROL[1]	Stack status: 1 = Alternate stack is used 0 = Default stack (MSP) is used If it is in the thread or base level, the alternate stack is the PSP. There is no alternate stack for handler mode, so this bit must be 0 when the processor is in handler mode.
CONTROL[0]	0 = Privileged in thread mode 1 = User state in thread mode If in handler mode (not thread mode), the processor operates in privileged mode.

➢ **CONTROL[1]**

o In the Cortex-M3, the CONTROL[1] bit is always 0 in handler mode. However, in the thread or base level, it can be either 0 or 1.

o This bit is writable only when the core is in thread mode and privileged.

o In the user state or handler mode, writing to this bit is not allowed.

o Aside from writing to this register, another way to change this bit is to change bit 2 of the LR when in exception return.

➢ **CONTROL[0]**

o The CONTROL[0] bit is writable only in a privileged state.

o Once it enters the user state, the only way to switch back to privileged is to trigger an interrupt and change this in the exception handler.

To access the control register in assembly, the MRS and MSR instructions are used:
MRS r0, CONTROL ; Read CONTROL register into R0
MSR CONTROL, r0 ; Write R0 into CONTROL register

1.7 OPERATION MODES

The Cortex-M3 processor has two modes and two privilege levels.

• The operation modes - **thread mode and handler mode-** determine whether the processor is running a normal program or running an exception handler like an interrupt handler or system exception handler.

• **The privilege levels (privileged level and user level)** provide a mechanism for safeguarding memory accesses to critical regions as well as providing a basic security model.

• Software in the privileged access level can switch the program into the user access level using the control register.

• When an exception takes place, the processor will always switch back to the privileged state and return to the previous state when exiting the exception handler.

• A user program cannot change back to the privileged state by writing to the control register.

• It has to go through an exception handler that programs the control register to switch the processor back into the privileged access level when returning to thread mode

• It can be used in conjunction with privilege levels to protect critical memory locations, such as programs and data for OS.

	Privileged	*User*
When running an exception handler	Handler mode	
When not running an exception handler (e.g., main program)	Thread mode	Thread mode

Fig6. Operation Modes and Privilege Levels in Cortex-M3.

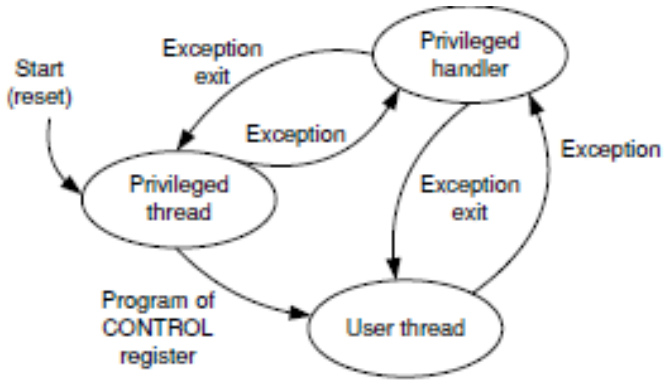


Fig7. Allowed Operation Mode Transitions.

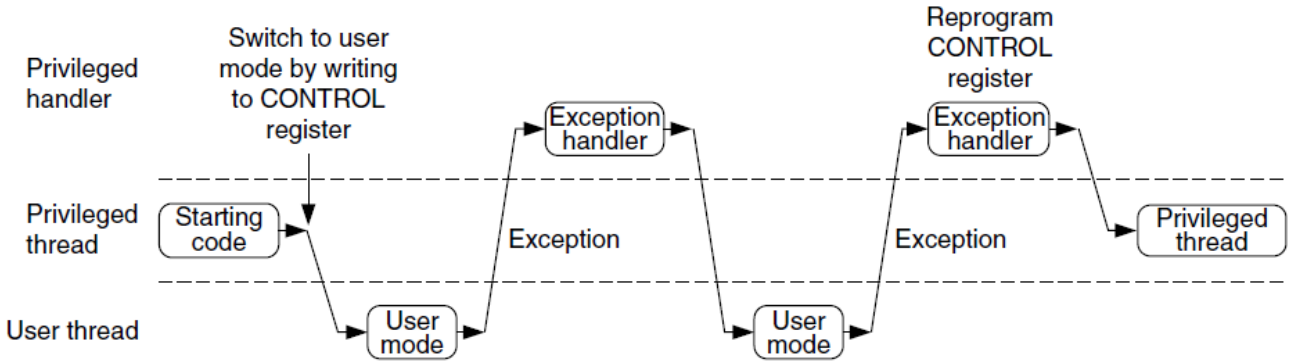


Fig8. Switching of Operation Mode by Programming the Control Register or by Exceptions.

In simple applications, there is no need to separate the privileged and user access levels.

• In these cases, there is no need to use user access level and no need to program the control register.

• Its recommended to separate the user application stack from the kernel stack memory to avoid the possibility of crashing a system caused by stack operation errors in user programs.

• With this arrangement, the user program (running in thread mode) uses the PSP, and the exception handlers use the MSP. The switching of SPs is automatic upon entering or leaving the exception handlers.

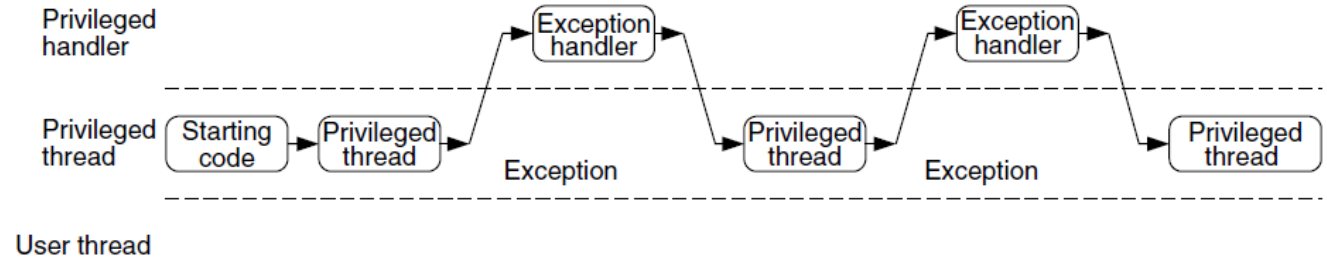


Fig9. Simple Applications Do Not Require User Access Level in Thread Mode

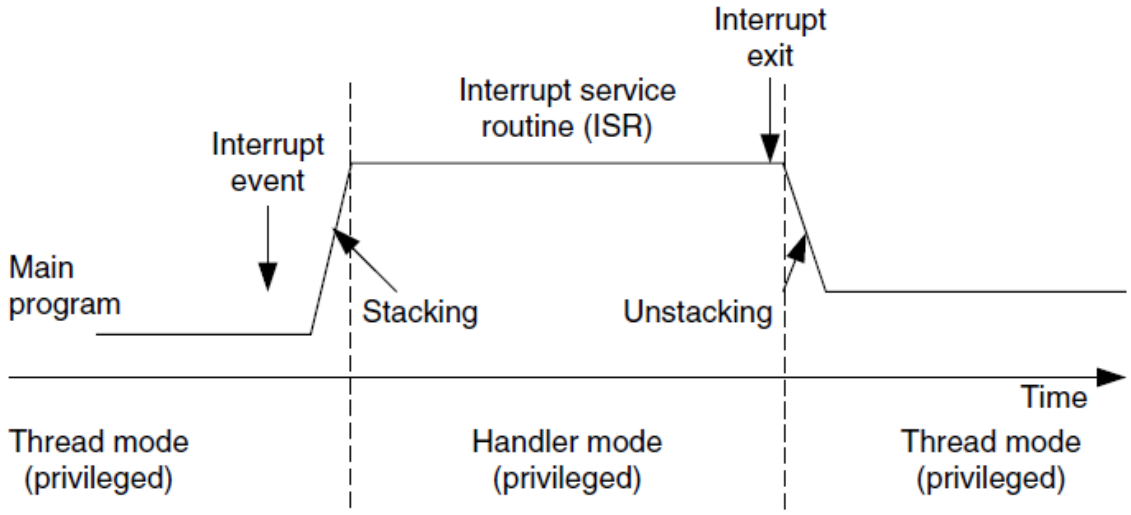


Fig10. Switching Processor Mode at Interrupt.

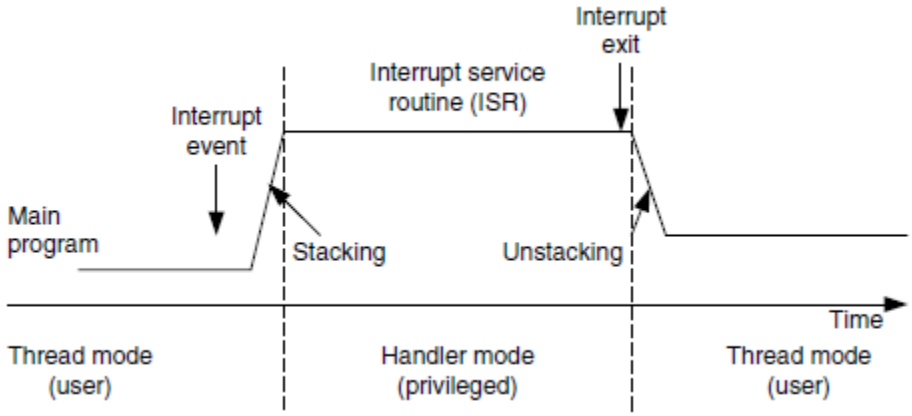


Fig11.Switching Processor Mode and Privilege Level at Interrupt.

1.8STACK IMPLEMENTATION

The Cortex-M3 uses a full-descending stack operation model. The SP points to the last data pushed to the stack memory, and the SP decrements before a new PUSH operation.

1.8.1 ONE REGISTER IN EACH STACK OPERATION

Main program

```
    ...
; R0 = X, R1 = Y, R2 = Z
BL    function 1
```

Subroutine

```
function 1
    PUSH    {R0-R2} ; Store R0, R1, R2 to stack
    ... ; Executing task (R0, R1 and R2
      ; could be changed)
    POP     {R0-R2} ; restore R0, R1, R2
    BX      LR    ; Return
```

```
; Back to main program
; R0 = X, R1 = Y, R2 = Z
... ; next instructions
```

1.8.2 MULTIPLE REGISTER STACK

Main program

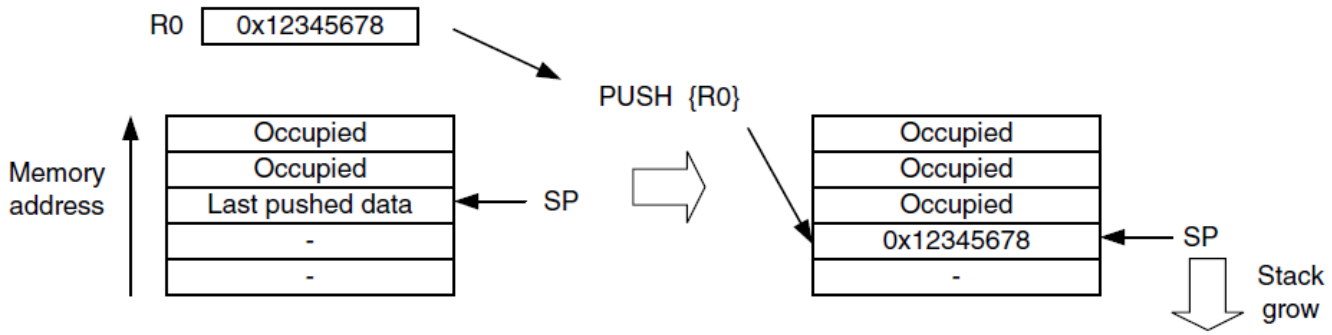
```
    ...
; R0 = X, R1 = Y, R2 = Z
BL    function 1
```

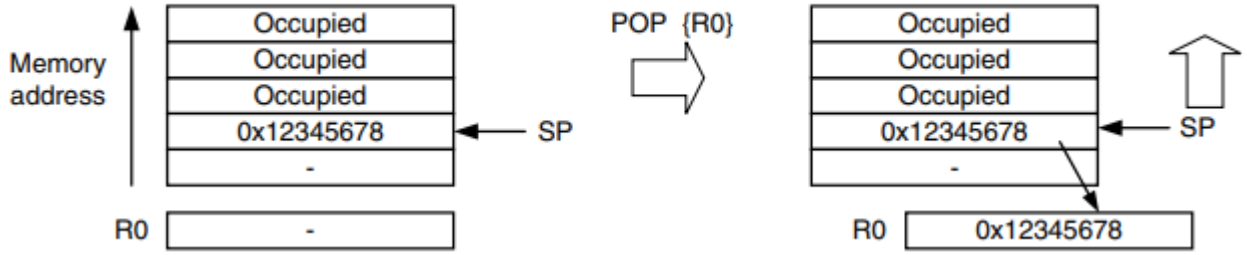
Subroutine

```
function 1
    PUSH    {R0-R2} ; Store R0, R1, R2 to stack
    ... ; Executing task (R0, R1 and R2
      ; could be changed)
    POP     {R0-R2} ; restore R0, R1, R2
    BX      LR    ; Return
```

```
; Back to main program
; R0 = X, R1 = Y, R2 = Z
... ; next instructions
```

1.8.3 COMBINING POP AND RETURN






```
Main program
    ...
; R0 = X, R1 = Y, R2 = Z
BL    function 1

Subroutine

function 1
    PUSH    {R0-R2, LR} ; Save registers
                ; including link register
    ... ; Executing task (R0, R1 and R2
        ; could be changed)
    POP     {R0-R2, PC} ; Restore registers and
                ; return

; Back to main program
; R0 = X, R1 = Y, R2 = Z
... ; next instructions
```

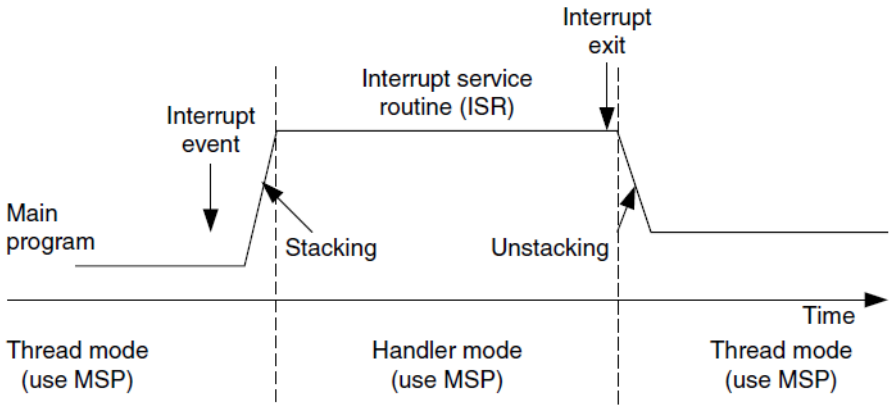
For POP operations, the data is read from the memory location pointed by SP, and then, the SP is incremented. The contents in the memory location are unchanged but will be overwritten when the next PUSH operation takes place Each PUSH/POP operation transfers 4 bytes of data (each register contains 1 word, or 4 bytes), the SP decrements/increments by 4 at a time or a multiple of 4 if more than 1 register is pushed or popped.
In the Cortex-M3, R13 is defined as the SP. When an interrupt takes place, a number of registers will be pushed automatically, and R13 will be used as the SP for this stacking process. Similarly, the pushed registers will be restored/popped automatically when exiting an interrupt handler, and the SP will also be adjusted.

1.8.4 THE TWO-STACK MODEL IN THE CORTEX-M3

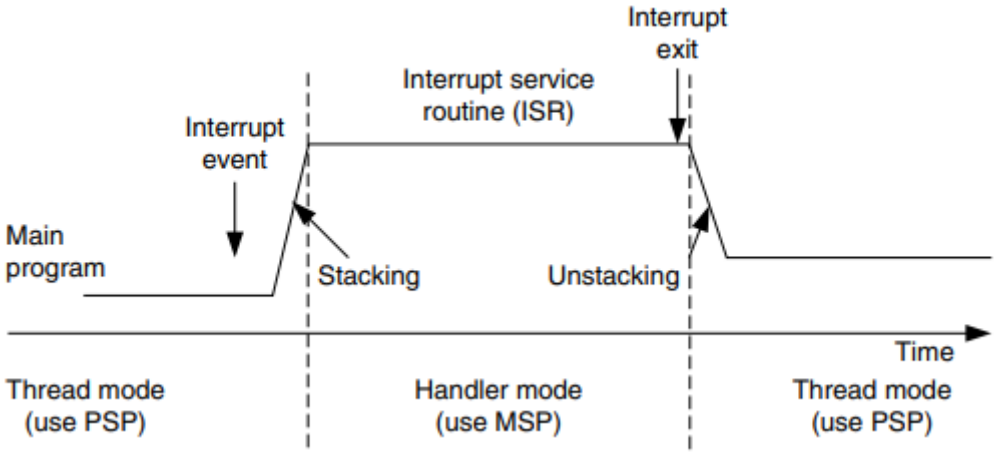
The Cortex-M3 has two SPs: the MSPS and the PSP.
- The SP register to be used is controlled by the control register[1].
- When CONTROL[1] is 0, the MSP is used for both thread mode and handler mode. In this arrangement, the main program and the exception handlers share the same stack memory region. This is the default setting after power-up.

When the CONTROL [1] is 1, the PSP is used in thread mode. In this arrangement, the main program and the exception handler can have separate stack memory regions.

Control [1] = 0, Both Thread Level and Handler Use Main Stack.



Control [1]=1, Thread Level Uses Process Stack and Handler Uses Main Stack.

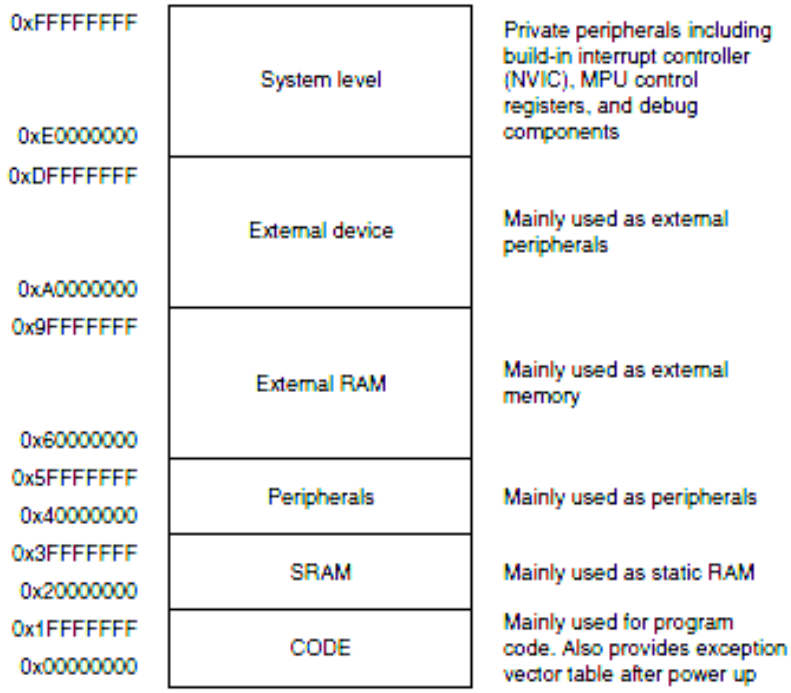


Both Thread Level and Handler Use Main Stack
It is possible to perform read/write operations directly to the MSP and PSP, without any confusion of which R13 you are referring to. Provided that you are in privileged level, you can access MSP and PSP values:
In general, it is not recommended to change the stack address

```
x = __get_MSP(); // Read the value of MSP
__set_MSP(x); // Set the value of MSP
x = __get_PSP(); // Read the value of PSP
__set_PSP(x); // Set the value of PSP
```

```
MRS R0, MSP  ; Read Main Stack Pointer to R0
MSR MSP, R0  ; Write R0 to Main Stack Pointer
MRS R0, PSP  ; Read Process Stack Pointer to R0
MSR PSP, R0  ; Write R0 to Process Stack Pointer
```

1.9 MEMORY MAP



The Cortex-M3 has a predefined memory map.

❖ This allows the built-in peripherals, such as the interrupt controller and the debug components, to be accessed by simple memory access instructions.

❖ Thus, most system features are accessible in C program code.

❖ The predefined memory map also allows the Cortex-M3 processor to be highly optimized for speed and ease of integration in system-on-a-chip (SoC) designs.

❖ Overall, the 4 GB memory space can be divided into ranges.

❖ The Cortex-M3 design has an internal bus infrastructure optimized for this memory usage.

❖ In addition, the design allows these regions to be used differently. For example, data memory can still be put into the CODE region, and program code can be executed from an external Random Access Memory (RAM) region.

1.10 BUILT IN NESTED VECTORED INTERRUPT CONTROLLER

The Cortex-M3 processor includes an interrupt controller called the Nested Vectored Interrupt Controller (NVIC). It is closely coupled to the processor core and provides a number of features as follows:

• Nested interrupt support

• Vectored interrupt support

- Dynamic priority changes support
- Reduction of interrupt latency
- Interrupt masking

1.10.1 NESTED INTERRUPT SUPPORT

The NVIC provides nested interrupt support. All the external interrupts and most of the system exceptions can be programmed to different priority levels. When an interrupt occurs, the NVIC compares the priority of this interrupt to the current running priority level. If the priority of the new interrupt is higher than the current level, the interrupt handler of the new interrupt will override the current running task.

1.10.2 VECTORED INTERRUPT SUPPORT

The Cortex-M3 processor has vectored interrupt support. When an interrupt is accepted, the starting address of the interrupt service routine (ISR) is located from a vector table in memory. There is no need to use software to determine and branch to the starting address of the ISR. Thus, it takes less time to process the interrupt request.

1.10.3 DYNAMIC PRIORITY CHANGES SUPPORT

Priority levels of interrupts can be changed by software during run time. Interrupts that are being serviced are blocked from further activation until the ISR is completed, so their priority can be changed without risk of accidental reentry.

1.10.4 REDUCTION OF INTERRUPT LATENCY

The Cortex-M3 processor also includes a number of advanced features to lower the interrupt latency. These include automatic saving and restoring some register contents, reducing delay in switching from one ISR to another, and handling of late arrival interrupts.

1.10.5 INTERRUPT MASKING

Interrupts and system exceptions can be masked based on their priority level or masked completely using the interrupt masking registers BASEPRI, PRIMASK, and FAULTMASK. They can be used to ensure that time-critical tasks can be finished on time without being interrupted. The system-level memory region contains the interrupt controller and the debug components. These devices have fixed addresses. By having fixed addresses for these peripherals, you can port applications between different Cortex-M3 products much more easily.

1.11 EXCEPTIONS AND INTERRUPTS

 The Cortex-M3 supports a number of exceptions, including a fixed number of system exceptions and a number of interrupts, commonly called *IRQ.*

 The number of interrupt inputs on a Cortex-M3 microcontroller depends on the individual design.

 The typical number of interrupt inputs is 16 or 32. However, you might find some microcontroller designs with more (or fewer) interrupt inputs

 Besides the interrupt inputs, there is also a nonmaskable interrupt (NMI) input signal.

 The actual use of NMI depends on the design of the microcontroller or system-on-chip (SoC) product you use.

 In most cases, the NMI could be connected to a watchdog timer or a voltage-monitoring block that warns the processor when the voltage drops below a certain level.

 The NMI exception can be activated any time, even right after the core exits reset.

 A number of the system exceptions are fault-handling exceptions that can be triggered by various error conditions.

Table 3.4 Exception Types in Cortex-M3

Exception Number	Exception Type	Priority	Function
1	Reset	–3 (Highest)	Reset
2	NMI	–2	Nonmaskable interrupt
3	Hard fault	–1	All classes of fault, when the corresponding fault handler cannot be activated because it is currently disabled or masked by exception masking
4	MemManage	Settable	Memory management fault; caused by MPU violation or invalid accesses (such as an instruction fetch from a nonexecutable region)
5	Bus fault	Settable	Error response received from the bus system; caused by an instruction prefetch abort or data access error
6	Usage fault	Settable	Usage fault; typical causes are invalid instructions or invalid state transition attempts (such as trying to switch to ARM state in the Cortex-M3)
7–10	—	—	Reserved
11	SVC	Settable	Supervisor call via SVC instruction
12	Debug monitor	Settable	Debug monitor
13	—	—	Reserved
14	PendSV	Settable	Pendable request for system service
15	SYSTICK	Settable	System tick timer
16–255	IRQ	Settable	IRQ input #0–239

1.12 VECTOR TABLES

1. When an exception event takes place on the Cortex-M3 and is accepted by the processor core, the corresponding exception handler is executed.

2. To determine the starting address of the exception handler, a vector table mechanism is used.

3. The *vector table is an array of word data inside the system* memory, each representing the starting address of one exception type.

4. The vector table is relocatable, and the relocation is controlled by a relocation register in the NVIC.

5. After reset, this relocation control register is reset to 0; therefore, the vector table is located in address 0x0 after reset.

6. For example, if the reset is exception type 1, the address of the reset vector is 1 times 4 (each word is 4 bytes), which equals 0x00000004, and NMI vector (type 2) is located in 2 × 4 = 0x00000008.

7. The address 0x00000000 is used to store the starting value for the MSP.

8. The LSB of each exception vector indicates whether the exception is to be executed in the Thumb state. Because the Cortex-M3 can support only Thumb instructions, the LSB of all the exception vectors should be set to 1.

Table 3.5 Vector Table Definition after Reset

Exception Type	Address Offset	Exception Vector
18–255	0x48–0x3FF	IRQ #2–239
17	0x44	IRQ #1
16	0x40	IRQ #0
15	0x3C	SYSTICK
14	0x38	PendSV
13	0x34	Reserved
12	0x30	Debug monitor
11	0x2C	SVC
7–10	0x1C–0x28	Reserved
6	0x18	Usage fault
5	0x14	Bus fault
4	0x10	MemManage fault
3	0x0C	Hard fault
2	0x08	NMI
1	0x04	Reset
0	0x00	Starting value of the MSP

1.12 THE BUS INTERFACE

The main bus interfaces are as follows:

- Code memory buses
- System bus
- Private peripheral bus

The code memory region access is carried out on the code memory buses, which physically consist of two buses, one called I-Code and other called D-Code.

The system bus is used to access memory and peripherals. This provides access to the Static Random Access Memory (SRAM), peripherals, external RAM, external devices, and part of the system level memory regions.

The private peripheral bus provides access to a part of the system-level memory dedicated to private peripherals, such as debugging components

1.13 MPU

- The Cortex-M3 has an optional MPU.
- This unit allows access rules to be set up for privileged access and user program access.
- When an access rule is violated, a fault exception is generated, and the fault exception handler will be able to analyze the problem and correct it, if possible.
- The MPU can be used in various ways.
- In common scenarios, the OS can set up the MPU to protect data used by the OS kernel and other privileged processes to be protected from untrusted user programs.
- The MPU can also be used to make memory regions read-only, to prevent accidental erasing of data or to isolate memory regions between different tasks in a multitasking system.
- Overall, it can help make embedded systems more robust and reliable.

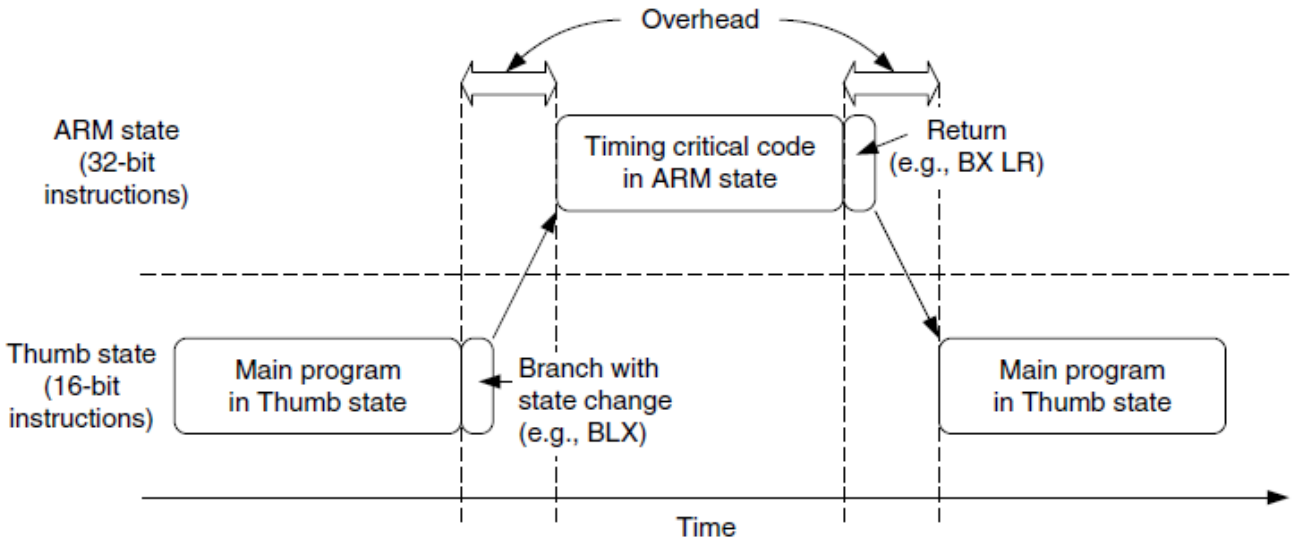
1.14 THUMB-2 INSTRUCTION SET

- The Cortex-M3 supports the Thumb-2 instruction set.
- This is one of the most important features of the Cortex-M3 processor because it allows 32-bit instructions and 16-bit instructions to be used together for high code density and high efficiency.
- It is flexible and powerful yet easy to use.
- In previous ARM processors, the central processing unit (CPU) had two operation states: a 32-bit ARM state and a 16-bit Thumb state.

 In the ARM state, the instructions are 32 bits and can execute all supported instructions with very high performance.
 In the Thumb state, the instructions are 16 bits, so there is a much higher instruction code density, but the Thumb state does not have all the functionality of ARM instructions and may require more instructions to complete certain types of operations.

1.14.1 BENEFITS OF THUMB-2 INSTRUCTIONS

 No state switching overhead, saving both execution time and instruction space
 No need to separate ARM code and Thumb code source files, making software development and maintenance easier
 It's easier to get the best efficiency and performance, in turn making it easier to write software, because there is no need to worry about switching code between ARM and Thumb to try to get the best density/performance



Debugging Support:

The Cortex-M3 processor includes a number of debugging features, such as program execution controls, including halting and stepping, instruction breakpoints, data watchpoints, registers and memory accesses, profiling, and traces. The debugging hardware of the Cortex-M3 processor is based on the CoreSight™ architecture.
Unlike traditional ARM processors, the CPU core itself does not have a Joint Test Action Group (JTAG) interface. Instead, a debug interface module is decoupled from the core, and a bus interface called the Debug Access Port (DAP) is provided at the core level. Through this bus interface, external debuggers can access control registers to debug hardware as well as system memory, even when the processor is running. The control of this bus interface is carried out by a Debug Port (DP) device.
The DPs currently available are the Serial-Wire JTAG Debug Port (SWJ-DP) (supports the traditional JTAG protocol as well as the Serial-Wire protocol) or the SW-DP (supports the Serial-Wire protocol only). A JTAG-DP module from the ARM CoreSight product family can also be used. Chip manufacturers can choose to attach one of these DP modules to provide the debug interface.
Chip manufacturers can also include an Embedded Trace Macrocell (ETM) to allow instruction trace. Trace information is output via the Trace Port Interface Unit (TPIU), and the debug host (usually a Personal Computer [PC]) can then collect the executed instruction information via external trace capturing hardware.

Stack Memory Operations:

In the Cortex-M3, besides normal software-controlled stack PUSH and POP, the stack PUSH and POP operations are also carried out automatically when entering or exiting an exception/interrupt handler. In this section, we examine the software stack operations.

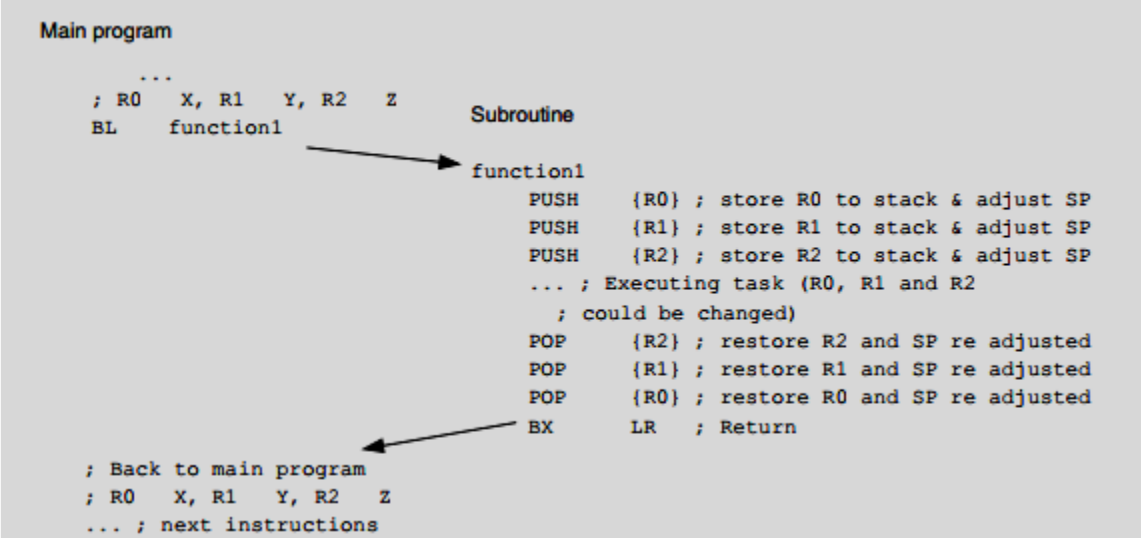
Operation:

In general, stack operations are memory write or read operations, with the address specified by an SP. Data in registers is saved into stack memory by a PUSH operation and can be restored to registers later by a POP operation. The SP is adjusted automatically in PUSH and POP so that multiple data PUSH will not cause old stacked data to be erased.

The function of the stack is to store register contents in memory so that they can be restored later, after a processing task is completed. For normal uses, for each store (PUSH), there must be a corresponding read (POP), and the address of the POP operation should match that of the PUSH operation. When PUSH/POP instructions are used, the SP is incremented/decremented automatically. When program control returns to the main program, the R0–R2 contents are the same as before.

Notice the order of PUSH and POP: The POP order must be the reverse of PUSH. These operations can be simplified, thanks to PUSH and POP instructions allowing multiple load and store. In this case, the ordering of a register POP is automatically reversed by the processor. You can also combine RETURN with a POP operation. This is done by pushing the LR to the stack and popping it back to PC at the end of the subroutine.

Cortex-M3 Stack Implementation:



Reset Sequence:

After the processor exits reset, it will read two words from memory

• Address 0x00000000: Starting value of R13 (the SP)

• Address 0x00000004: Reset vector (the starting address of program execution; LSB should be set to 1 to indicate Thumb state)

This differs from traditional ARM processor behavior. Previous ARM processors executed program code starting from address 0x0. Furthermore, the vector table in previous ARM devices was instructions.

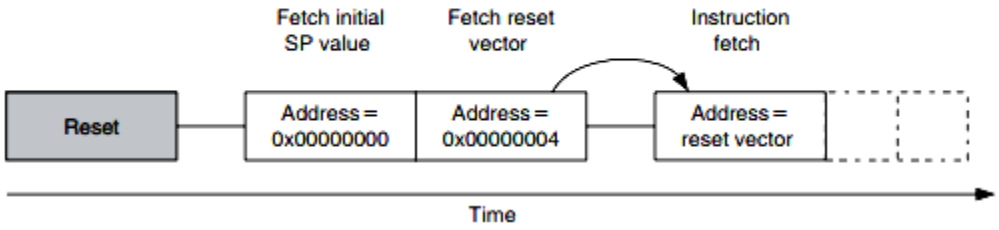


Fig3.Reset sequence

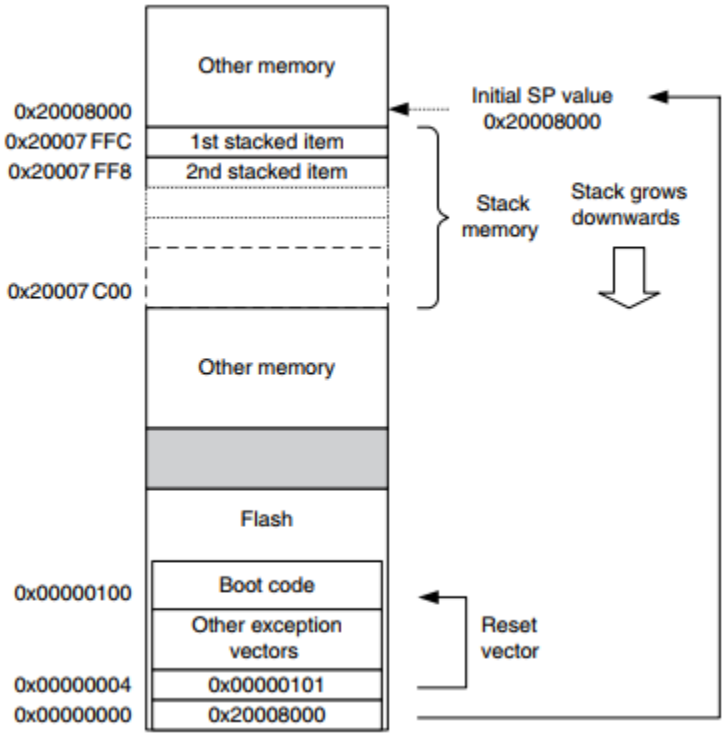


FIG.4 Initial Stack Pointer Value and Initial Program Counter Value Example.

In the Cortex-M3, the initial value for the MSP is put at the beginning of the memory map, followed by the vector table, which contains vector address values. (The vector table can be relocated to another location later, during program execution.) In addition, the contents of the vector table are address values not branch instructions. The first vector in the vector table (exception type 1) is the reset vector, which is the second piece of data fetched by the processor after reset. Because the stack operation in the Cortex-M3 is a full descending stack (SP decrement before store), the initial SP value should be set to the first memory after the top of the stack region. For example, if you have a stack memory range from 0x20007C00 to 0x20007FFF (1 KB), the initial stack value should be set to 0x20008000.

QUESTION BANK
MODULE 1

1. With a neat diagram explain the architecture of ARM Cortex M3 microcontroller. (6)
2. Give the applications Cortex-m3 processor. (6)
3. Explain the operation modes of ARM Cortex M3. (6)
4. Give the memory map of Cortex M3. (4)
5. Briefly describe the functions of the various units with the architectural block diagram of ARM Cortex M3. (6)
6. Discuss the functions of R0 to R15 and other special registers in Cortex M3. (7)
7. Describe the functions of exceptions with a vector table and priorities. (6)
8. Explain two stack model and reset sequence in ARM Cortex M3. (7)
9. With a neat diagram explain the thumb-2 set architecture in comparison with thumb and ARM. (4)
10. Discuss various profiles of ARM processors. (3)
11. Bring out the differences between 1) RISC and CISC architecture 2) Von Neumann and Harvard architecture and 3) microprocessors and microcontrollers. (6)
12. Write a short note on interrupts and exceptions supported by Cortex M3. (3)
13. Give the two stack model in Cortex M3. (6)
14. Write short note on reset sequence. (4)

MODULE -2

ARM EMBEDDED SOFTWARE DEVELOPMENT

2.1 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.

2.2 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

2.3 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.

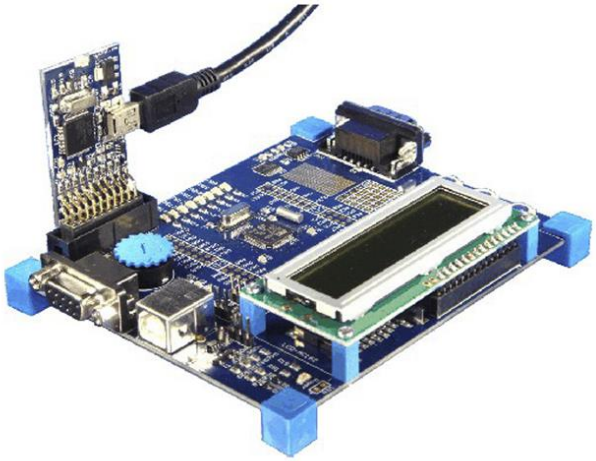


FIGURE 2.1

A Cortex-M3 development board from Keil (MCBSTM32)

2.4 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.

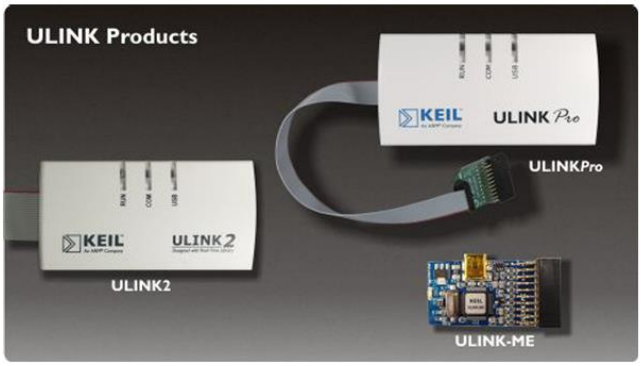


FIGURE 2.2

The Keil ULINK debug adaptor family

- 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.

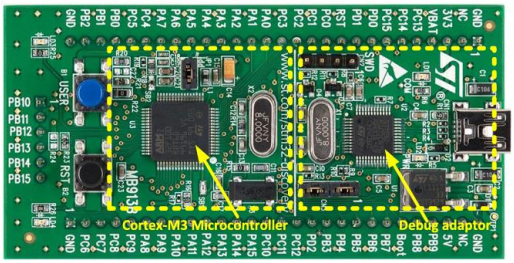


FIGURE 2.3
An example of development board with USB debug adaptor – STM32 Value Line Discovery

2.5 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.

2.6 Software development flow

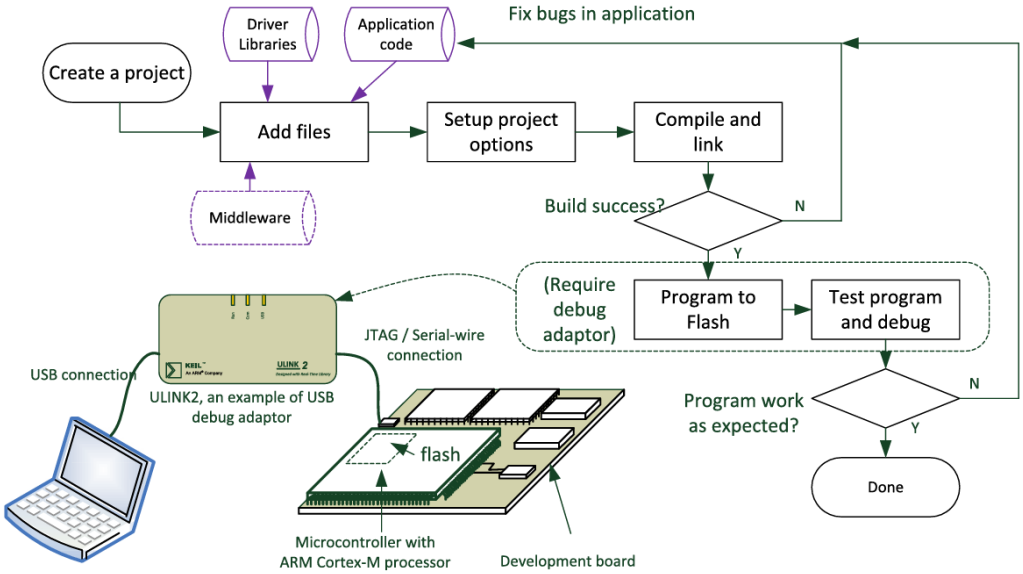


FIGURE 2.4
A simplified software development flow

The software development flow depends on the compiler suite you use. Assuming that you are using a compiler suite with Integrated Development Environment (IDE), the software development flow (as shown in Figure 2.4) usually involves:

Create project - you need to create a project that will specify the location of source files, compile target, memory configurations, compilation options, and so on. Many IDEs have a project creation wizard for this step.

Add files to project - you need to add the source code files required by the project. You might also need to specify the path of any included header files in the project options. Obviously you might also need to create new program source code files and write the program. Note that you should be able to reuse a number of files from the device-driver library to reduce the effort in writing new files. This includes startup code, header files, and some of the peripheral control functions.

Setup project options - In most cases, the project file created allows a number of project options such as compiler optimization options, memory map, and output f ile types. Based on the development board and debug adaptor you have, you might also need to setup options for debug and code download.

Compile and link - In most cases, a project contains a number of files that are compiled separately. After the compilation process, each source file will have a corresponding object file. In order to generate the final combined executable image, a separate linking process is required. After the link stage, the IDE can also generate the program image in other file formats for the purpose of programming the image to the device.

Flash programming - Almost all of the Cortex-M microcontrollers use flash memories for program storage. After a program image is created, we need to download the program to the flash memory of the microcontroller.

Execute program and debug - After the compiled program is downloaded to the microcontroller, you can then run the program and see if it works.

2.7 Compiling

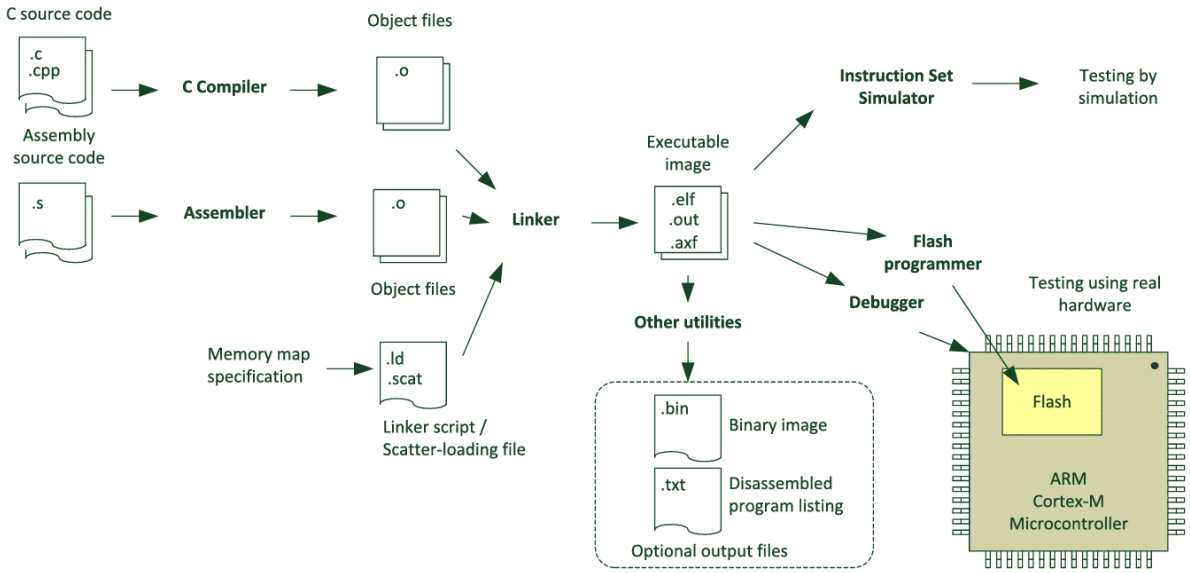


FIGURE 2.5

Common software compilation flow

First, we assume that you are developing your project using C programming language. This is the most commonly used programming language for microcon troller software development. Your project might also contain some assembly language files; for example, startup code that is supplied by microcontroller vendors. In most cases, the compilation process will be similar to the one shown in Figure 2.5. Most development suites contain

the tools listed in Table 2.1. Different development tools have different ways to specify the layout of the pro gram and data memory in the microcontroller system. In ARM toolchains, you can use a file type called scatter-loading file, or in the case of Keil MDK-ARM, the scatter-loading file can be generated automatically by the mVision development environment. For some other ARM toolchains, you can also use command line options to specify the locations of ROM and RAM.

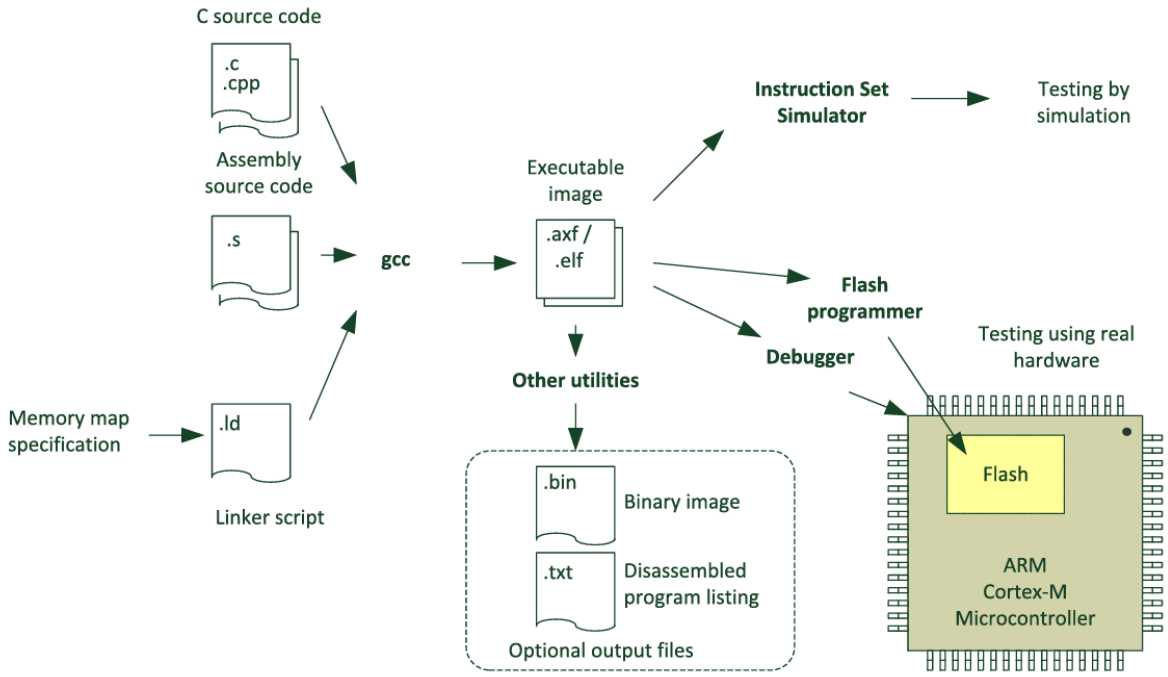


FIGURE 2.6

Common software compilation flow for GNU toolchain

In a GNU-based toolchain, the memory specification is handled by linker scripts. These scripts are typically included in the installation of commercial gcc toolchains. However, some gcc users might have to create these files themselves. A later chapter of this book contains examples for compiling programs using gcc, which covers more information on linker scripts. When using the GNU gcc toolchain, it is common to compile the whole application in one go instead of separating the compilation and linking stages (Figure 2.6). The gcc compilation automatically invokes the linker and assembler if needed. This arrangement ensures that the details of the required parameters and libraries are passed on to the linker correctly. Using the linker as a separate step can be error prone and therefore is not recommended by most gcc tool vendors.

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

2.8 Software flow

- Two methods of Software flow –
- Polling - For very simple applications, the processor can wait until there is data ready for processing, 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.”

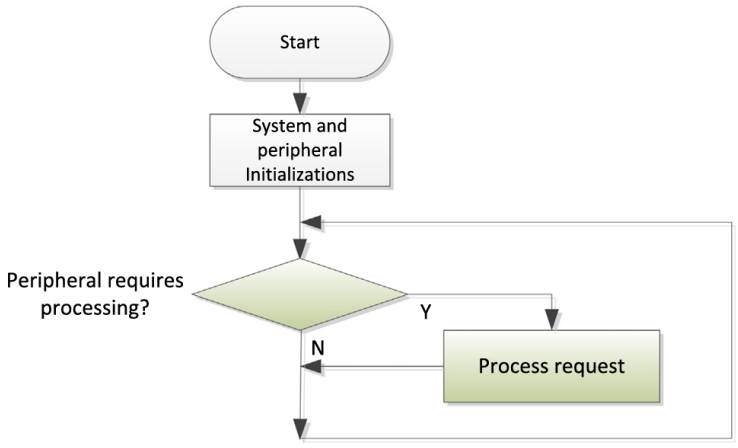


FIGURE 2.7
Polling method for simple application processing

- 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 - 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.

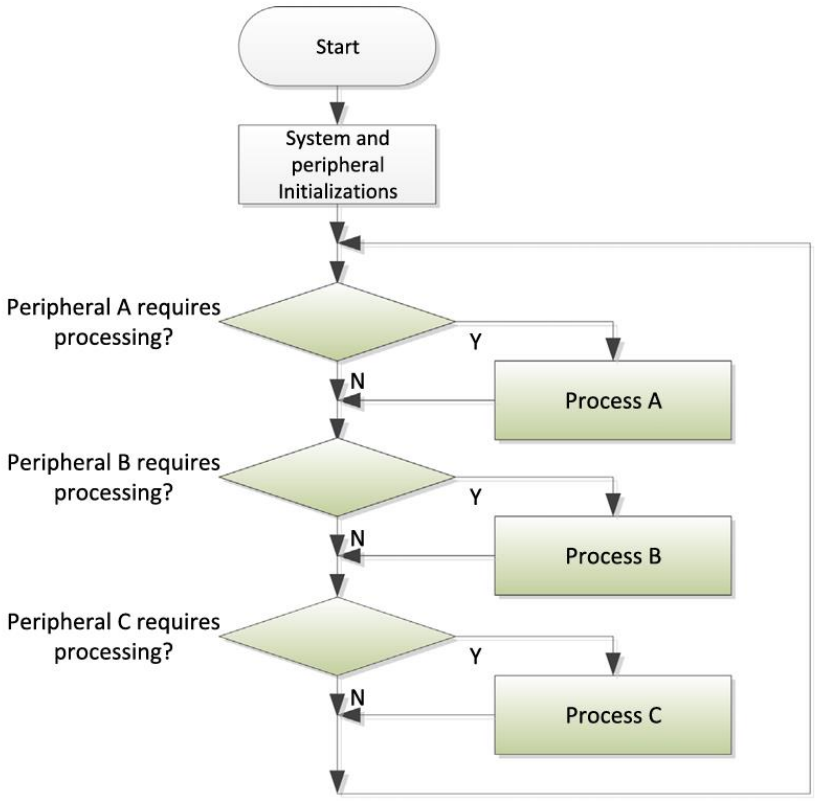


FIGURE 2.8

Polling method for application with multiple devices that need processing

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.

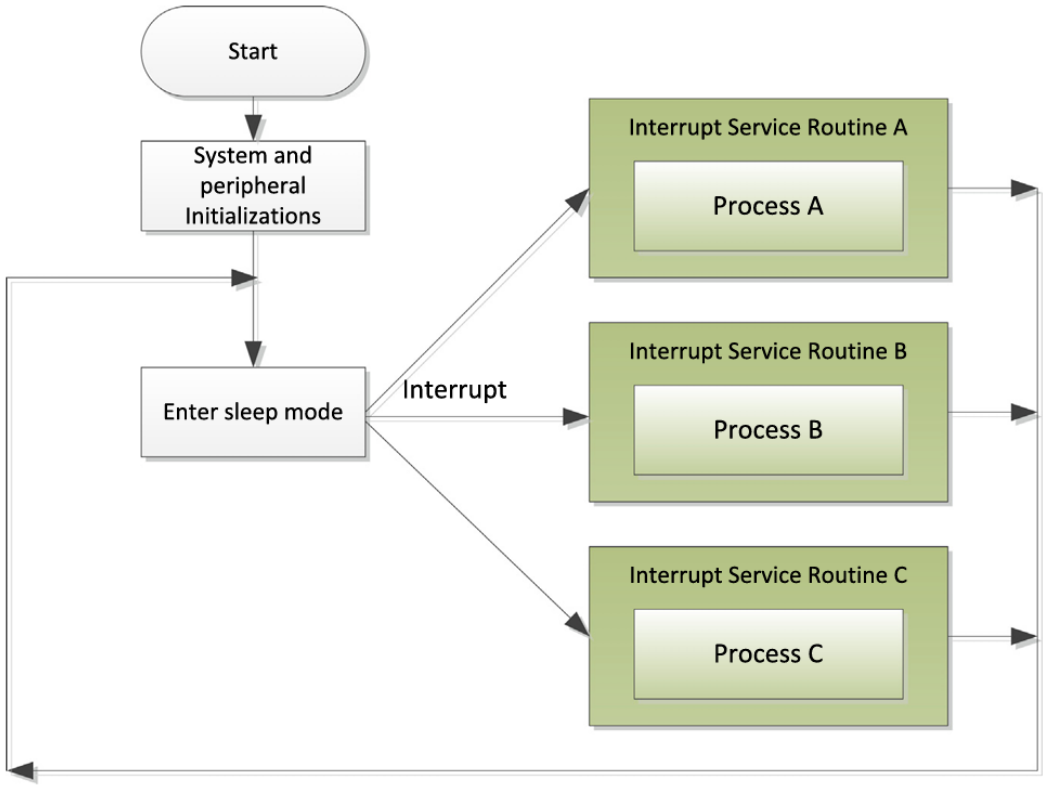


FIGURE 2.9

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.

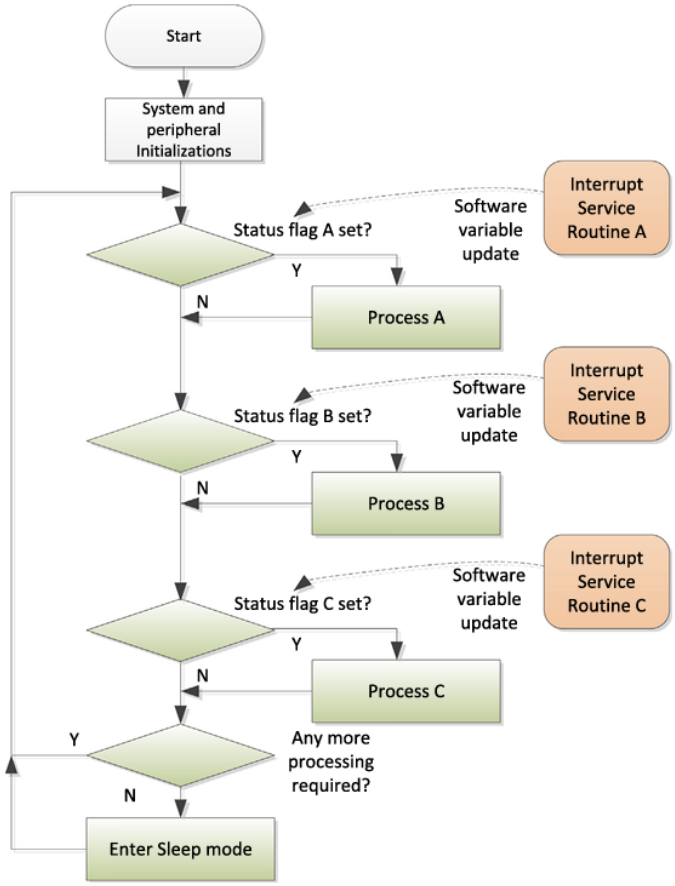


FIGURE 2.10

Application with both polling method and interrupt-driven arrangement

2.9 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)
- 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.

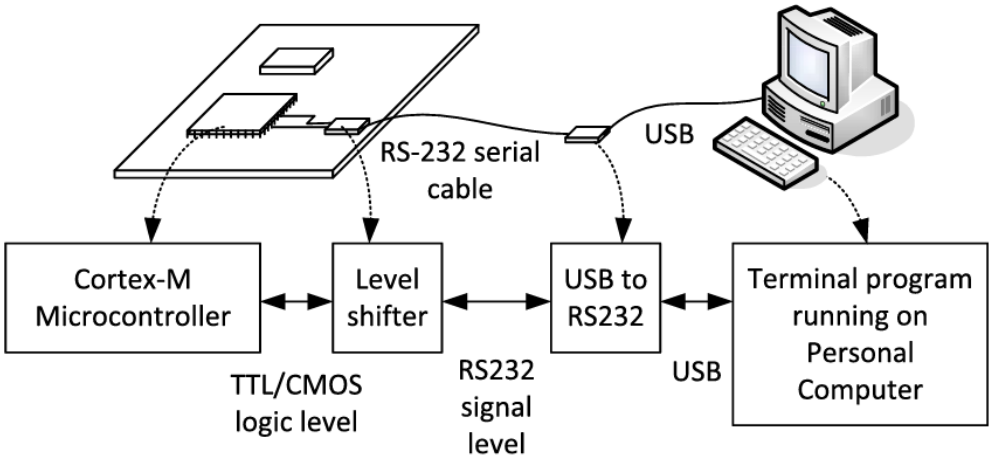


FIGURE 2.12

Using a UART to communicate with a PC via USB

2.10 The Cortex Microcontroller Software Interface Standard

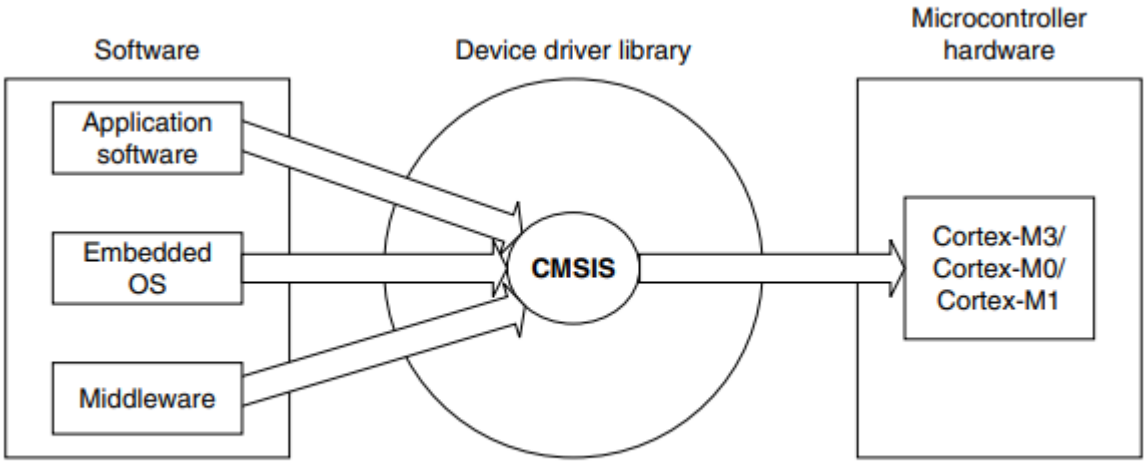


FIGURE 10.6

CMSIS Provides a Standardized Access Interface for Embedded Software Products.

The Cortex-M3 microcontrollers are gaining momentum in the embedded application market, as more and more products based on the Cortex-M3 processor and software that support the Cortex-M3 processor are emerging. At the end of 2008, there were more than five C compiler vendors, and more than 15 embedded Operating Systems (OS) supporting the Cortex-M3 processor. There are also a number of companies providing embedded

software solutions, including codecs, data processing libraries, and various software and debug solutions. The CMSIS was developed by ARM to allow users of the Cortex-M3 microcontrollers to gain the most benefit from all these software solutions and to allow them to develop their embedded application quickly and reliably (see Figure 10.6).

2.11 Introduction to CMSIS

The CMSIS was started in 2008 to improve software usability and inter-operability of ARM microcontroller software. It is integrated into the driver libraries provided by silicon vendors, providing a standardized software interface for the Cortex-M3 processor features, as well as a number of common system and I/O functions. The library is also supported by software companies including embedded OS vendors and compiler vendors.
The aims of CMSIS are to:
• improve software portability and reusability
• enable software solution suppliers to develop products that can work seamlessly with device libraries from various silicon vendors
• allow embedded developers to develop software quicker with an easy-to-use and standardized software interface
• allow embedded software to be used on multiple compiler products
• avoid device driver compatibility issues when using software solutions from multiple sources
The first release of CMSIS was available from fourth quarter of 2008 and has already become part of the device driver library from microcontroller vendors. The CMSIS is also available for Cortex-M0.

2.12 Use CMSIS – core

- **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.
- **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.

Organisation of CMSIS Core

- 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.

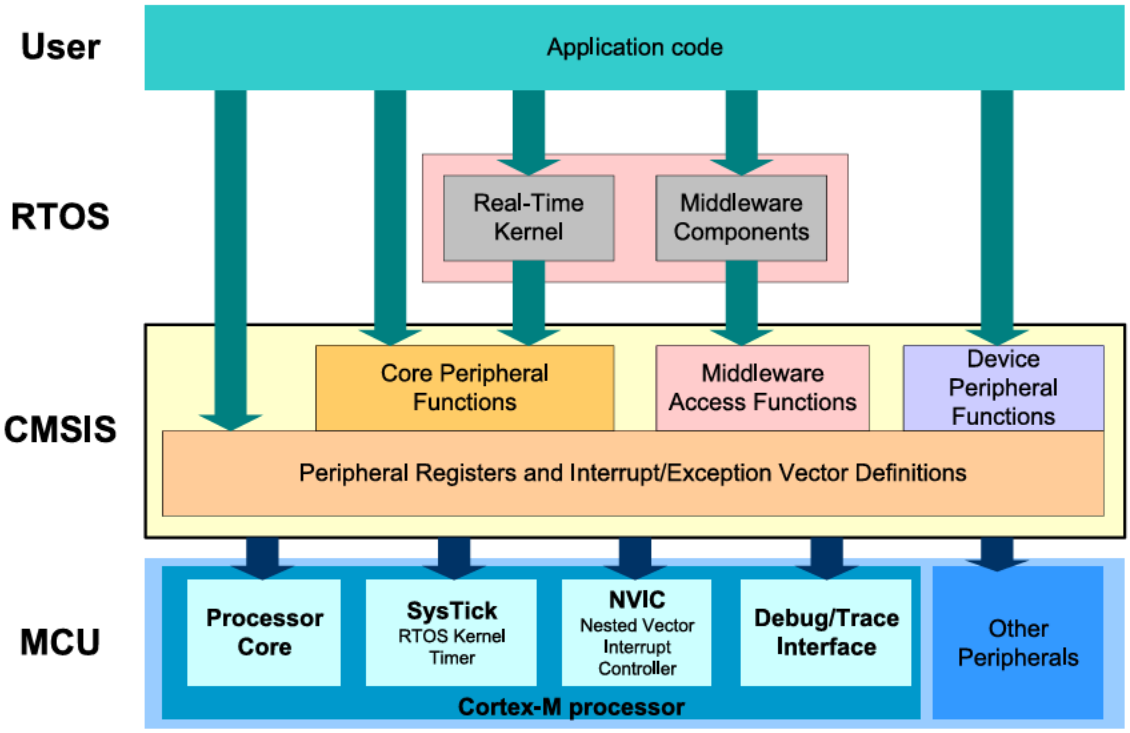


FIGURE 2.13

CMSIS-Core structure

Use of 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.

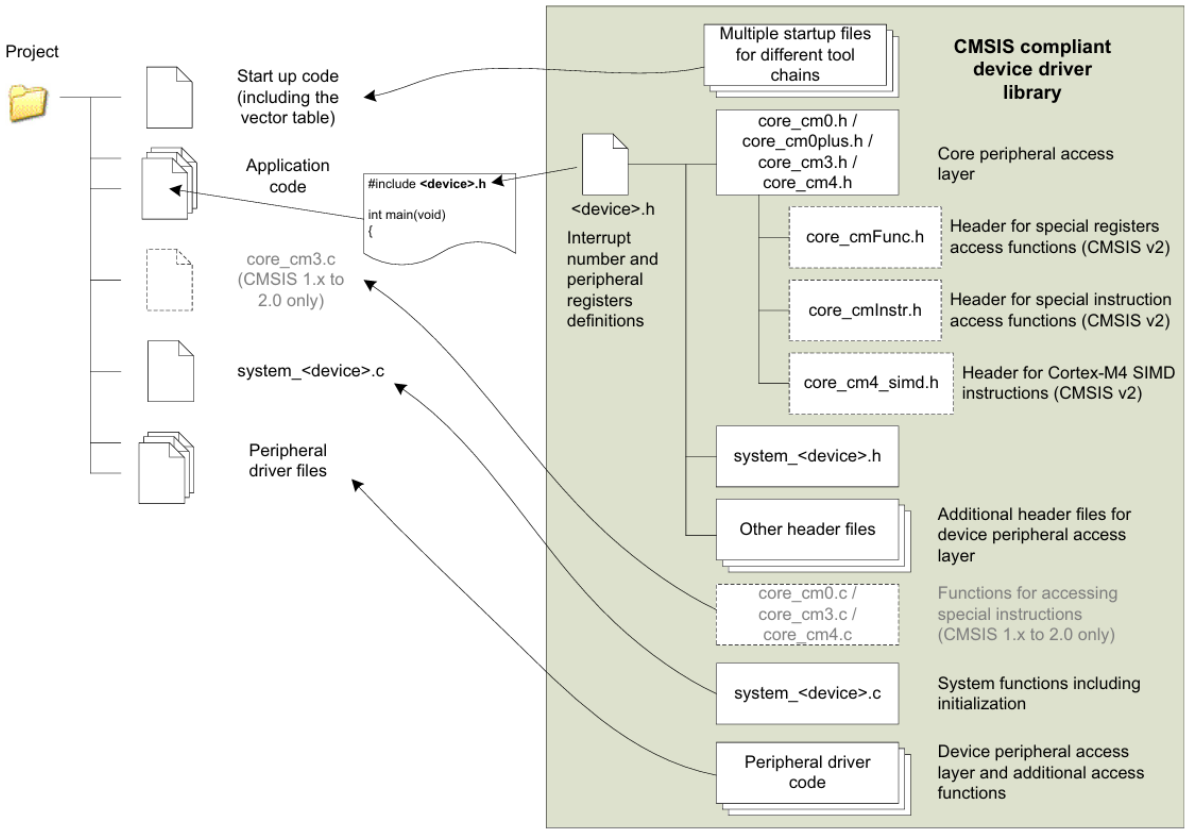


FIGURE 2.14

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.

2.13 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 software 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.
- 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 functions 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).

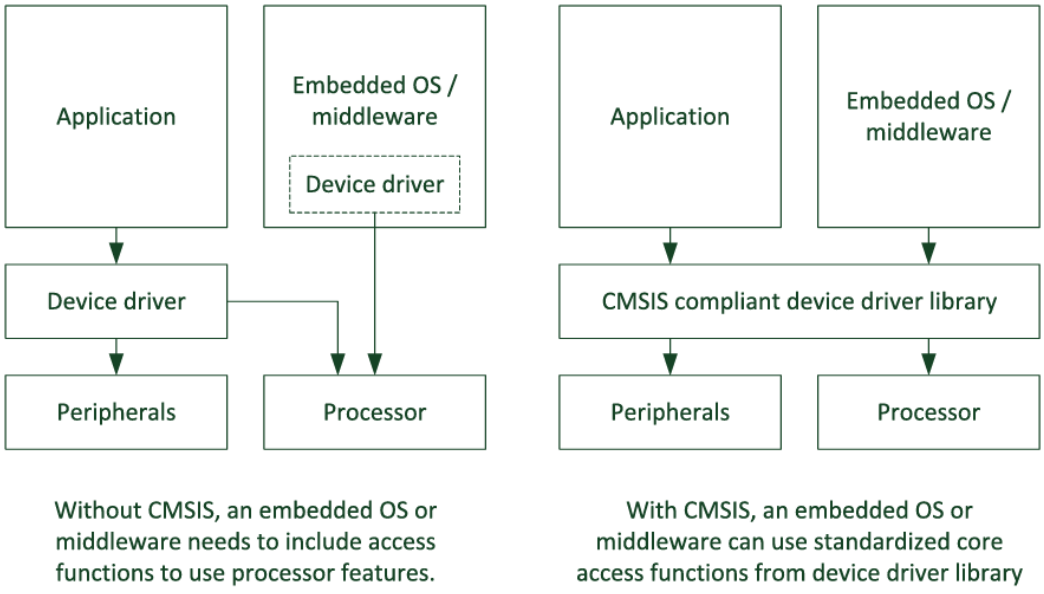


FIGURE 2.15

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

Module 2

1. With a diagram, explain the organization of CMSIS and its benefits.
2. List out the components inside ARM. Explain.
3. With a neat diagram, Explain common software compilation flow.
4. Explain simplified software development flow.
5. Describe various tools available in Development suite for Cortex M3.
6. Explain different methods of software flow with a neat flow chart.
7. Explain the Microcontroller interface using a UART to communicate with a PC via USB.
8. Explain areas of standardization in CMSIS – core.
9. Explain common software compilation flow with GNU Toolchain.
10. With a neat flowchart, explain the application with polling method and interrupt driven arrangement.