CHAPTER 1

Introduction to the Cortex-M Processor Family

Cortex Profiles

In 2004, ARM introduced its new Cortex family of processors. The Cortex processor family is subdivided into three different profiles. Each profile is optimized for different segments of embedded systems applications.

Figure 1.1
The Cortex processor family has three profiles—application, real time, and microcontroller.

The Cortex-A profile has been designed as a high-end application processor. Cortex-A processors are capable of running feature-rich operating systems such as WinRT and Linux. The key applications for Cortex-A are consumer electronics such as smart phones, tablet computers, and set-top boxes. The second Cortex profile is Cortex-R. This is the real-time profile that delivers a high-performance processor which is the heart of an application-specific device. Very often a Cortex-R processor forms part of a "system-on-chip" design that is focused on a specific task such as hard disk drive (HDD) control, automotive engine management, and medical devices. The final profile is Cortex-M or the microcontroller profile. Unlike earlier ARM CPUs, the Cortex-M processor family has been designed specifically for use within a small microcontroller. The Cortex-M processor currently comes in five variants: Cortex-M0, Cortex-M0+ , Cortex-M1, Cortex-M3, and Cortex-M4. The Cortex-M0 and Cortex-M0+ are the smallest processors in the family. They allow silicon

manufacturers to design low-cost, low-power devices that can replace existing 8-bit microcontrollers while still offering 32-bit performance. The Cortex-M1 has much of the same features as the Cortex-M0 but has been designed as a "soft core" to run inside an Field Programmable Gate Array (FPGA) device. The Cortex-M3 is the mainstay of the Cortex-M family and was the first Cortex-M variant to be launched. It has enabled a new generation of high-performance 32-bit microcontrollers which can be manufactured at a very low cost. Today, there are many Cortex-M3-based microcontrollers available from a wide variety of silicon manufacturers. This represents a seismic shift where Cortex-M-based microcontrollers are starting to replace the traditional 8\16-bit microcontrollers and even other 32-bit microcontrollers. The highest performing member of the Cortex-M family is the Cortex-M4. This has all the features of the Cortex-M3 and adds support for digital signal processing (DSP) and also includes hardware floating point support for single precision calculations.

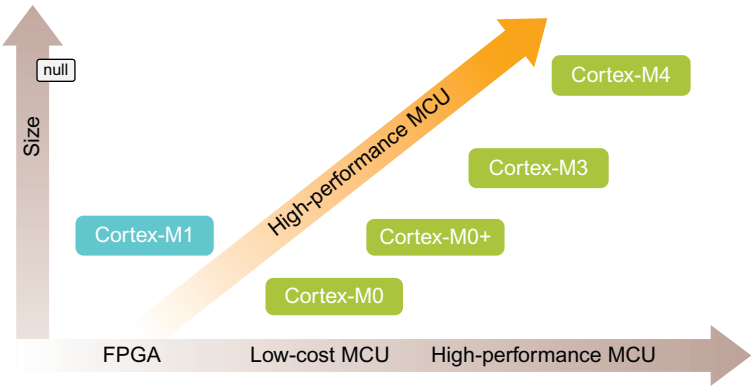

Figure 1.2
The Cortex-M profile has five different variants with a common programmers model.

In the late 1990s, various manufacturers produced microcontrollers based on the ARM7 and ARM9 CPUs. While these microcontrollers were a huge leap in performance and competed in price with existing 8/16-bit architectures, they were not always easy to use. A developer would first have to learn how to use the ARM CPU and then have to understand how a specific manufacturer had integrated the ARM CPU into their microcontroller system. If you have moved to another ARM-based microcontroller you might have gone through another learning curve of the microcontroller system before you could confidently start development. Cortex-M changes all that; it is a complete Microcontroller Unit (MCU) architecture, not just a CPU core. It provides a standardized bus interface, debug architecture, CPU core, interrupt structure, power control, and memory protection. More importantly, each Cortex-M processor is the same across all manufacturers, so once you have learned to use one

Cortex-M-based processor you can reuse this knowledge with any other manufacturers of Cortex-M microcontrollers. Also within the Cortex-M family, once you have learned the basics of how to use a Cortex-M3, then you can use this experience to develop using a Cortex-M0, Cortex-M0+ , or a Cortex-M4 device. Through this book, we will use the Cortex-M3 as a reference device and then look at the differences between Cortex-M3 and Cortex-M0, Cortex-M0+ , and Cortex-M4, so that you will have a practical knowledge of all the Cortex-M processors.

Cortex-M3

Today, the Cortex-M3 is the most widely used of all the Cortex-M processors. This is partly because it has been available not only for the longest period of time but also it meets the requirements for a general-purpose microcontroller. This typically means it has a good balance between high performance, low power consumption, and low cost.

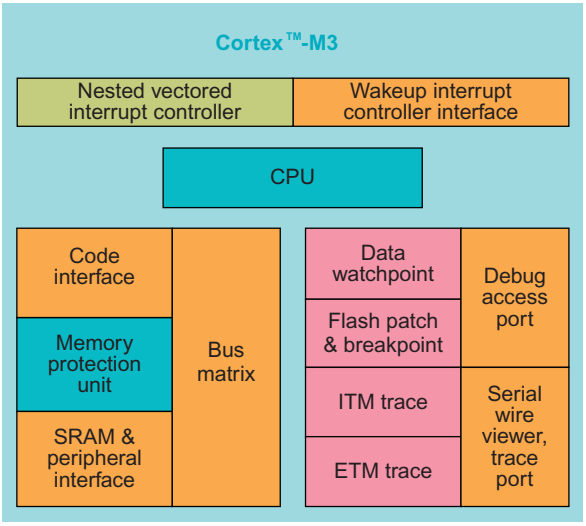

Figure 1.3
The Cortex-M3 was the first Cortex-M device available. It is a complete processor for a general-purpose microcontroller.

The heart of the Cortex-M3 is a high-performance 32-bit CPU. Like the ARM7, this is a reduced instruction set computer (RISC) processor where most instructions will execute in a single cycle.

Figure 1.4
The Cortex-M3 CPU has a three-stage pipeline with branch prediction.

This is partly made possible by a three-stage pipeline with separate fetch, decode, and execute units.

Figure 1.5
The Cortex-M3 CPU can execute most instructions in a single cycle. This is achieved by the pipeline executing one instruction, decoding the next, and fetching a third.

So while one instruction is being executed, a second is being decoded, and a third is being fetched. The same approach was used on the ARM7. This is great when the code is going in a straight line, however, when the program branches, the pipeline must be flushed and refilled with new instructions before execution can continue. This made branches on the ARM7 quite expensive in terms of processing power. However, the Cortex-M3 and Cortex-M4 include an instruction to fetch unit that can handle speculative branch target fetches which can reduce the bench penalty. This helps the Cortex-M3 and Cortex-M4 to have a sustained processing power of 1.25 DMIPS/MHz. In addition, the processor has a hardware integer math unit with hardware divide and single cycle multiply. The Cortex-M3 processor also includes a nested vector interrupt unit (NVIC) that can service up to 240 interrupt sources. The NVIC provides fast deterministic interrupt handling and from an interrupt being raised to reaching the first line of "C" in the interrupt service routine takes just 12 cycles every time. The NVIC also contains a standard timer called the systick timer. This is a 24-bit countdown timer with an auto reload. This timer is present on all of the different Cortex-M processors. The systick timer is used to provide regular periodic interrupts. A typical use of this timer is to provide a timer tick for small footprint real-time operating systems (RTOS). We will have a look at such an RTOS in Chapter 6. Also next to the NVIC is the wakeup interrupt controller (WIC); this is a small area of the Cortex-M processor that is kept alive when the processor is in low-power mode. The WIC can use the interrupt signals from the microcontroller peripherals to wake up the Cortex-M processor from a low-power mode. The WIC can be implemented in various ways and in some cases does not require a clock to function; also, it can be in a separate power region from the main Cortex-M processor. This allows 99% of the Cortex-M processor to be placed in a low-power mode with just minimal current being used by the WIC.

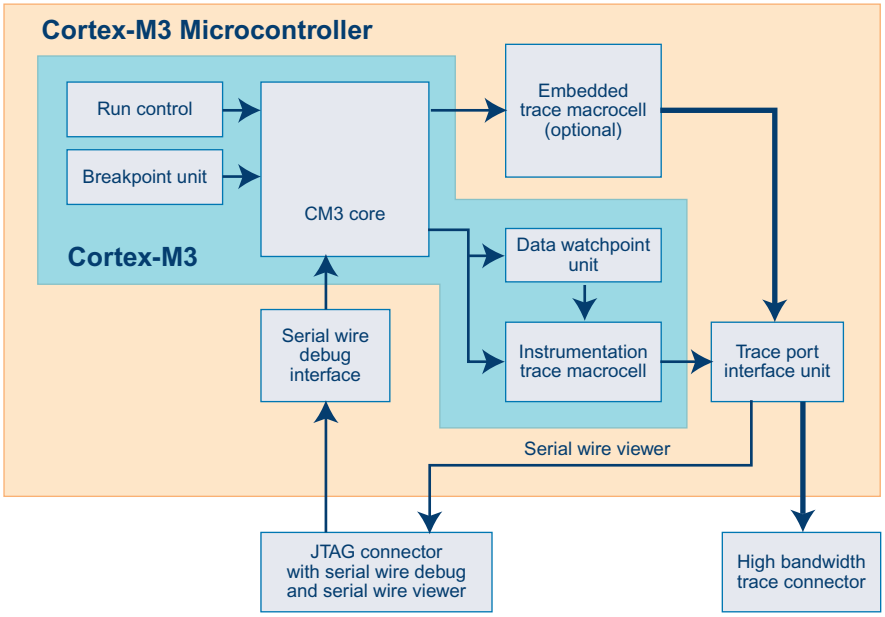

Figure 1.6
The Cortex-M debug architecture is consistent across the Cortex-M family and contains up to three real-time trace units in addition to the run control unit.

The Cortex-M family also has a very advanced debug architecture called CoreSight. The earlier ARM7/9 processors could be debugged through a joint test action group (JTAG) debug interface. This provided a means to download the application code into the on-chip flash memory and then exercise the code with basic run/stop debugging. While a JTAG debugger provided a low-cost way of debugging, it had two major problems. The first was a limited number of breakpoints, generally two with one being required for single stepping code and secondly, when the CPU was executing code the microcontroller became a black box with the debugger having no visibility to the CPU, memory, or peripherals until the microcontroller was halted. The CoreSight debug architecture within the Cortex-M processors is much more sophisticated than the old ARM7 or ARM9 processors. It allows up to eight hardware breakpoints to be placed in code or data regions. CoreSight also provides three separate trace units that support advanced debug features without intruding on the execution of the Cortex CPU. The Cortex-M3 and Cortex-M4 are always fitted with a data watchpoint and trace (DWT) unit and an instrumentation trace macrocell (ITM) unit. The debug interface allows a low-cost debugger to view the contents of memory and peripheral registers "on the fly" without halting the CPU, and the DWT can export a number of watched data, everything that is accessed by the processor, without stealing any cycles from the CPU. The second trace unit is called the instrumentation trace. This trace unit provides a debug

communication method between the running code and the debugger user interface. During development, the standard IO channel can be redirected to a console window in the debugger. This allows you to instrument your code with printf() debug messages which can then be read in the debugger while the code is running. This can be useful for trapping complex runtime problems. The instrumentation trace is also very useful during software testing as it provides a way for a test harness to dump data to the PC without needing any specific hardware on the target. The instrumentation trace is actually more complex than a simple UART, as it provides 32 communication channels which can be used by different resources within the application code. For example, we can provide extended debug information about the performance of an RTOS by placing the code in the RTOS kernel that uses an instrumentation trace channel to communicate with the debugger. The final trace unit is called the embedded trace macrocell (ETM). This trace unit is an optional fit and is not present on all Cortex-M devices. Generally, a manufacturer will fit the ETM on their high-end microcontrollers to provide extended debug capabilities. The ETM provides instruction trace information that allows the debugger to build an assembler and High level language trace listing of the code executed. The ETM also enables more advanced tools such as code coverage monitoring and timing performance analysis. These debug features are often a requirement for safety critical and high integrity code development.

Advanced Architectural Features

The Cortex-M3 and Cortex-M4 can also be fitted with another unit to aid high integrity code execution. The memory protection unit allows developers to segment the Cortex-M memory map into regions with different access privileges. We will look at the operating modes of the Cortex-M processor in Chapter 5, but to put it in simple terms, the Cortex CPU can execute the code in a privileged mode or a more restrictive unprivileged mode. The memory protection unit (MPU) can define privileged and unprivileged regions over the 4 GB address space (i.e., code, ram, and peripheral). If the CPU is running in unprivileged mode and it tries to access a privileged region of memory, the MPU will raise an exception and execution will vector to the MPU fault service routine. The MPU provides hardware support for more advanced software designs. For example, you can configure the application code so that an RTOS and low-level device drivers have full privileged access to all the features of the microcontroller while the application code is restricted to its own region of code and data. Like the ETM, the MPU is an optional unit which may be fitted by the manufacturers during design of the microcontroller. The MPU is generally found on high-end devices which have large amounts of flash memory and SRAM. Finally, the Cortex-M3 and Cortex-M4 are interfaced to the rest of the microcontroller through a Harvard bus architecture. This means that they have a port for fetching instructions and constants from code memory and a second port for accessing SRAM and peripherals. We will look at the bus interface more closely in Chapter 5, but in essence, the Harvard bus

architecture increases the performance of the Cortex-M processor but does not introduce any additional complexity for the programmer.

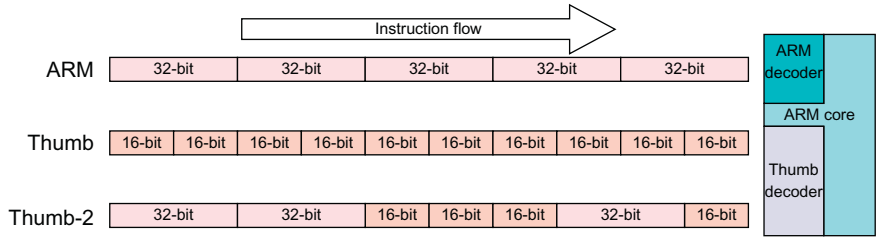

Figure 1.7

Earlier ARM CPUs had two instruction sets, ARM (32 bit) and Thumb (16 bit). The Cortex-M processors have an instruction set called Thumb-2 which is a blend of 16- and 32-bit instructions.

The earlier ARM CPUs, ARM7 and ARM9, supported two instruction sets. This code could be compiled either as 32-bit ARM code or as 16-bit Thumb code. The ARM instruction set would allow code to be written for maximum performance, while Thumb code would achieve a greater code density. During development, the programmer had to decide which function should be compiled with the ARM 32-bit instruction set and which should be built using the Thumb 16-bit instruction set. The linker would then interwork the two instruction sets together. While the Cortex-M processors are code compatible with the original Thumb instruction set, they are designed to execute an extended version of the Thumb instruction set called Thumb-2. Thumb-2 is a blend of 16- and 32-bit instructions that has been designed to be very C friendly and efficient. For even the smallest Cortex-M project, all of the code can be written in a high-level language, typically C, without any need to use an assembler.

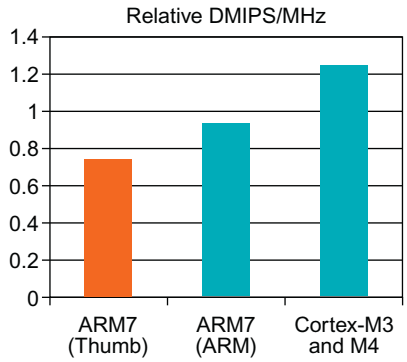

Figure 1.8

The Cortex-M3 and Cortex-M4 Thumb-2 instruction set achieves higher performance levels than either the Thumb or the ARM instruction set running on the ARM7.

With the Thumb-2 instruction set, the Cortex-M3 provides 1.25 DMIPS/MHz, while the ARM7 CPU using the ARM 32-bit instruction set achieves 0.95 DMIPS/MHz, and the Thumb 16-bit instruction set on ARM7 is 0.7 DMIPS/MHz. The Cortex-M CPU has a number of hardware accelerators such as single cycle multiply, hardware division, and bit field manipulation instructions that help boost its performance compared to ARM7-based devices.

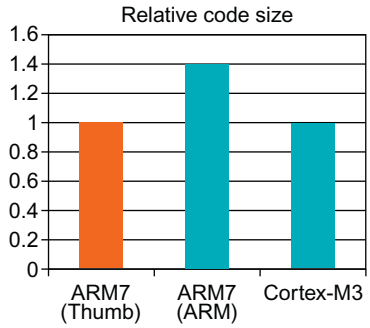

Figure 1.9
The Thumb-2 instruction set of the Cortex-M3 and Cortex-M4 achieves the same code density as the ARM7 Thumb (16 bit) instruction set.

The Thumb-2 instruction set is also able to achieve excellent code density that is comparable to the original 16-bit Thumb instruction set while delivering more processing performance than the ARM 32-bit instruction set.

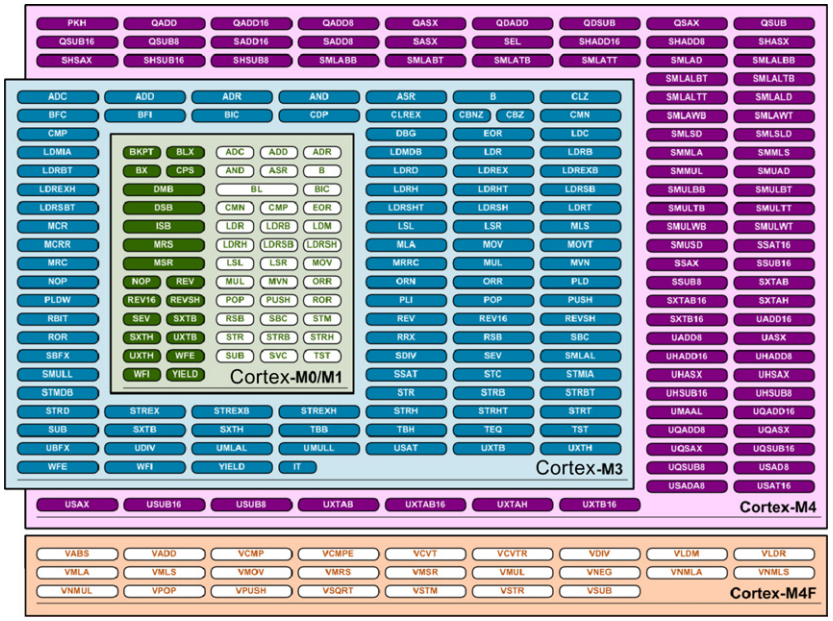

Figure 1.10
The Thumb-2 instruction set scales from 56 instructions on the Cortex-M0 and Cortex-M0+ to up to 169 instructions on the Cortex-M4.

All of the Cortex-M processors use the Thumb-2 instruction set. The Cortex-M0 uses a subset of just 56 instructions and the Cortex-M4 adds the DSP, single instruction multiple data (SIMD), and floating point instructions.

Cortex-M0

The Cortex-M0 was introduced a few years after the Cortex-M3 was released and was in general use. The Cortex-M0 is a much smaller processor than the Cortex-M3 and can be as small as 12 K gates in minimum configuration. The Cortex-M0 is typically designed into microcontrollers that are intended to be very low-cost devices and\or intended for low-power operation. However, the important thing is that once you understand the Cortex-M3, you will have no problem using the Cortex-M0; the differences are mainly transparent to high-level languages.

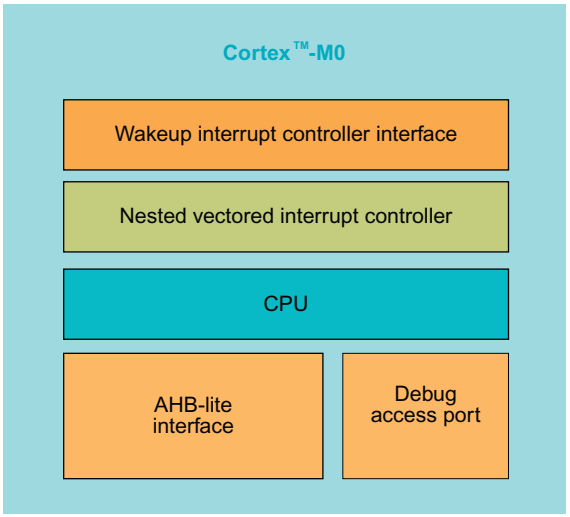

Figure 1.11
The Cortex-M0 is a reduced version of the Cortex-M3 while still keeping the same programmers model.

The Cortex-M0 processor has a CPU that can execute a subset of the Thumb-2 instruction set. Like the Cortex-M3, it has a three-stage pipeline but no branch speculation fetch, therefore branches and jumps within the code will cause the pipeline to flush and refill before execution can resume. The Cortex-M0 also has a von Neumann bus architecture, so there is a single path for code and data. While this makes for a simple design, it can become a bottleneck and reduce performance. Compared to the Cortex-M3, the Cortex-M0 achieves 0.84 DMIPS/MHz, which while less than the Cortex-M3 is still about the same as an ARM7 which has three times the gate count. So, while the Cortex-M0 is at the bottom

end of the Cortex-M family, it still packs a lot of processing power. The Cortex-M0 processor has the same NVIC as the Cortex-M3, but it is limited to a maximum of 32 interrupt lines from the microcontroller peripherals. The NVIC also contains the systick timer that is fully compatible with the Cortex-M3. Most RTOS that run on the Cortex-M3 and Cortex-M4 will also run on the Cortex-M0, though the vendor will need to do a dedicated port and recompile the RTOS code. As a developer, the biggest difference you will find between using the Cortex-M0 and the Cortex-M3 is its debug capabilities. While on the Cortex-M3 and Cortex-M4 there is extensive real-time debug support, the Cortex-M0 has a more modest debug architecture. On the Cortex-M0, the DWT unit does not support data trace and the ITM is not fitted, so we are left with basic run control (i.e., run, halt, single stepping and breakpoints, and watchpoints) and on-the-fly memory/peripheral accesses. This is still an enhancement from the JTAG support provided on ARM7 and ARM9 CPUs.

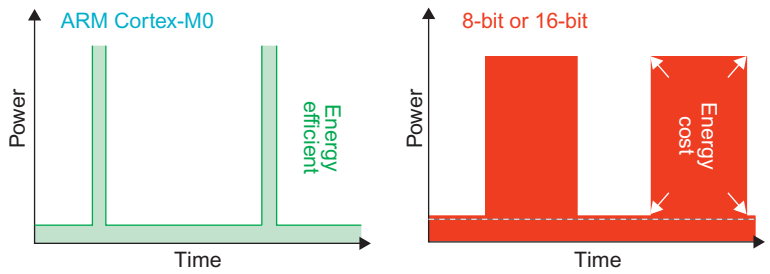

Figure 1.12
The Cortex-M0 is designed to support low-power standby modes. Compared to an 8- or 16-bit MCU, it can stay in sleep mode for much more time because it needs to execute fewer instructions than an 8/16-bit device to achieve the same result.

While the Cortex-M0 is designed to be a high performance microcontroller processor it has a relatively low gate count. This makes it ideal for both low cost and low power devices. The typical power consumption of the Cortex-M0 is 16 μW/MHz when running and almost zero when in its low-power sleep mode. While other 8- and 16-bit architectures can also achieve similar low-power figures, they need to execute far more instructions than the Cortex-M0 to achieve the same end result. This means extra cycles and extra cycles would mean more power consumption. If we pick a good example for Cortex-M0 such as a 16×16 multiply, then the Cortex-M0 can perform this calculation in one cycle. In comparison, an 8-bit typical architecture like the 8051 will need at least 48 cycles and a 16-bit architecture will need 8 cycles. This is not only a performance advantage but also an energy efficiency advantage as well.

Table 1.1: Number of Cycles Taken for a 16 × 16 Multiply against Typical 8- and 16-bit Architectures

8-Bit Example (8051)	16-Bit Example	ARM Cortex-M

MOV A, XL ; 2 bytes
MOV B, YL ; 3 bytes
MUL AB; 1 byte
MOV R0, A; 1 byte
MOV R1, B; 3 bytes
MOV A, XL ; 2 bytes
MOV B, YH ; 3 bytes
MUL AB; 1 byte
ADD A, R1; 1 byte
MOV R1, A; 1 byte
MOV A, B ; 2 bytes
ADDC A, #0 ; 2 bytes
MOV R2, A; 1 byte
MOV A, XH ; 2 bytes
MOV B, YL ; 3 bytes
MUL AB; 1 byte
ADD A, R1; 1 byte
MOV R1, A; 1 byte
MOV A, B ; 2 bytes
ADDC A, #0 ; 2 bytes
MOV R2, A; 1 byte
MOV A, XH ; 2 bytes
MOV B, YH ; 3 bytes
MUL AB; 1 byte
ADD A, R2; 1 byte
MOV R2, A; 1 byte
MOV A, B ; 2 bytes
ADDC A, R2 ; 1 bytes
MOV R2, A; 1 byte
MOV A, XH ; 2 bytes
MOV B, YH ; 3 bytes
MUL AB; 1 byte
ADD A, R2; 1 byte
MOV R2, A; 1 byte
MOV R3, A; 1 byte
	MOV R1,&MulOp1
	MOV R2,&MulOp2
	MOV SumLo,R3
	MOV SumHi,R4
	
	(Memory mapped multiply unit)
		MULS r0,r1,r0

Time: 48 clock cycles* **Code size:** 48 bytes	**Time:** 8 clock cycles **Code size:** 8 bytes	**Time:** 1 clock cycle **Code size:** 2 bytes

*cycle count for a single cycle 8051 processor.

Like the Cortex-M3, the Cortex-M0 also has the WIC feature. While the WIC is coupled to the Cortex-M0 processor, it can be placed in a different power domain within the microcontroller.

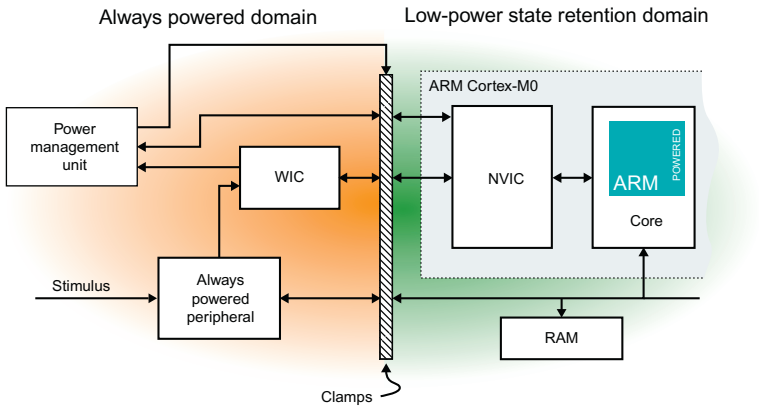

Figure 1.13
The Cortex-M processor is designed to enter low-power modes. The WIC can be placed in a separate power domain.

This allows the microcontroller manufacturer to use their expertise to design very low-power devices where the bulk of the Cortex-M0 processor is placed in a dedicated low-power domain which is isolated from the microcontroller peripheral power domain. These kinds of architected sleep states are critical for designs that are intended to run from batteries.

Cortex-M0+

The Cortex-M0+ processor is the latest generation low-power Cortex-M core. It has complete instruction set compatibility with the Cortex-M0 allowing you to use the same compiler and debug tools. As you might expect, the Cortex-M0+ has some important enhancements over the Cortex-M0.

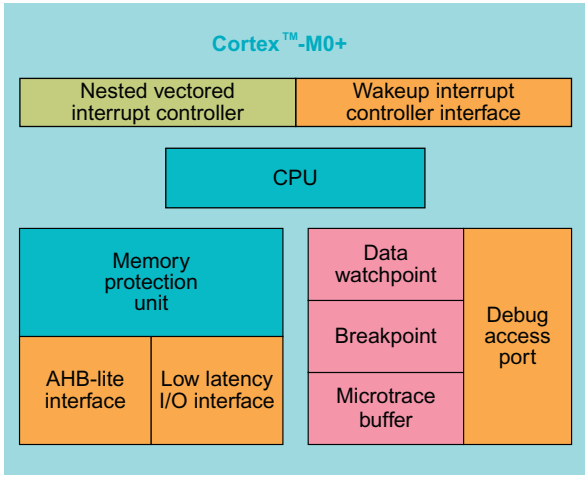

Figure 1.14
The Cortex-M0+ is fully compatible with the Cortex-M0. It has more advanced features such as more processing power and lower power consumption.

The defining feature of the Cortex-M0+ is its power consumption, which is just 11 μW/MHz compared to 16 μW/MHz for the Cortex-M0 and 32 μW/MHz for the Cortex-M3. One of the Cortex-M0+ key architectural changes is a move to a two-stage pipeline. When the Cortex-M0 and Cortex-M0+ execute a conditional branch, the instructions in the pipeline are no longer valid. This means that the pipeline must be flushed every time there is a branch. Once the branch has been taken the pipeline must be refilled to resume execution. While this impacts on performance it also means accessing the flash memory and each access costs energy as well as time. By moving to a two-stage pipeline, the number of flash memory accesses and hence the runtime energy consumption is also reduced.

Figure 1.15
The Cortex-M0+ has a two-stage pipeline compared to the three-stage pipeline used in other Cortex-M processors.

Another important feature added to the Cortex-M0+ is a new peripheral I/O interface that supports single cycle access to peripheral registers. The single cycle I/O interface is a standard part of the Cortex-M0+ memory map and uses no special instructions or paged addressing. Registers located within the I/O interface can be accessed by normal C pointers from within your application code. The I/O interface allows faster access to peripheral

registers with less energy use while being transparent to the application code. The single cycle I/O interface is separate from the advanced high speed bus (AHB) lite external bus interface, so it is possible for the processor to fetch instructions via the AHB lite interface while making a data access to the peripheral registers located within the I/O interface.

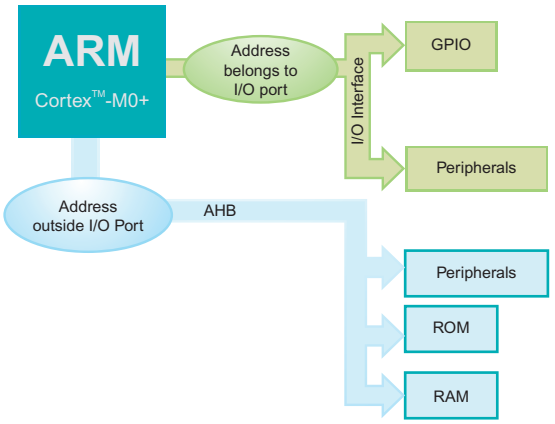

Figure 1.16
The I/O port allows single cycle access to General Purpose IO (GPIO) and peripheral registers.

The Cortex-M0+ is designed to support fetching instructions from 16-bit flash memories. Since most of the Cortex-M0+ instructions are 16 bit, this does not have a major impact on performance but does make the resulting microcontroller design simpler, smaller, and consequently cheaper. The Cortex-M0+ has some Cortex-M3 features missing on the original Cortex-M0. This includes the MPU, which we will look at in Chapter 5, and the ability to relocate the vector table to a different position in memory. These two features provide improved operating system (OS) support and support for more sophisticated software designs with multiple application tasks on a single device.

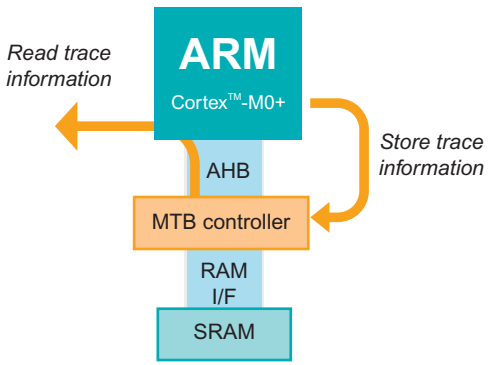

Figure 1.17
The microtrace buffer (MTB) can be configured to record executed instructions into a section of user SRAM. This can be read and displayed as an instruction trace in the PC debugger.

The Cortex-M0+ also has an improved debug architecture compared to the Cortex-M0. As we will see in Chapter 8, it supports the same real-time access to peripheral registers and SRAM as the Cortex-M3 and Cortex-M4. In addition, the Cortex-M0+ has a new debug feature called MTB. The MTB allows executed program instructions to be recorded into a region of SRAM setup by the programmer during development. When the code is halted this instruction trace can be downloaded and displayed in the debugger. This provides a snapshot of code execution immediately before the code was halted. While this is a limited trace buffer, it is extremely useful for tracking down elusive bugs. The MTB can be accessed by standard JTAG/serial wire debug adaptor hardware, for which you do not need an expensive trace tool.

Cortex-M4

While the Cortex-M0 can be thought of as a Cortex-M3 minus some features, the Cortex-M4 is an enhanced version of the Cortex-M3. The additional features on the Cortex-M4 are focused on supporting DSP algorithms. Typical algorithms are transforms such as fast Fourier transform (FFT), digital filters such as finite impulse response (FIR) filters, and control algorithms such as a Proportional Internal Differential (PID) control loop. With its DSP features, the Cortex-M4 has created a new generation of ARM-based devices that can be characterized as digital signal controllers (DSC). These devices allow you to design devices that combine microcontroller type functions with real-time signal processing. In Chapter 7, we will look at the Cortex-M4 DSP extensions in more detail and also how to construct software that combines real-time signal processing with typical event-driven microcontroller code.

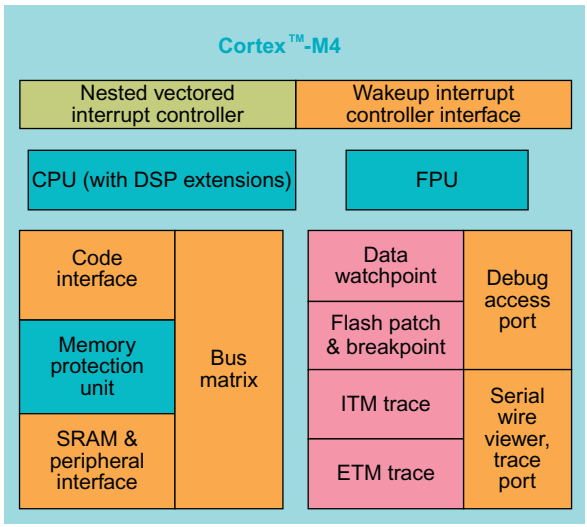

Figure 1.18
The Cortex-M4 is fully compatible with the Cortex-M3 but introduces a hardware floating point unit (FPU) and additional DSP instructions.

The Cortex-M4 has the same basic structure as the Cortex-M3 with the same CPU programmers modes, NVIC, CoreSight debug architecture, MPU, and bus interface. The enhancements over the Cortex-M3 are partly to the instruction set where the Cortex-M4 has additional DSP instructions in the form of SIMD instructions. The hardware multiply accumulate (MAC) has also been improved so that many of the 32 × 32 arithmetic instructions are single cycle.

Table 1.2: Single Cycle MAC Instructions on the Cortex-M4

Operation	Instructions
16 x 16 = 32	SMULBB, SMULBT, SMULTB, SMULTT
16 x 16 + 32 = 32	SMLABB, SMLABT, SMLATB, SMLATT
16 x 16 + 64 = 64	SMLALBB, SMLALBT, SMLALTB, SMLALTT
16 x 32 = 32	SMULWB, SMULWT
(16 x 32) + 32 = 32	SMLAWB, SMLAWT
(16 x 16) ± (16 x 16) = 32	SMUAD, SMUADX, SMUSD, SMUSDX
(16 x 16) ± (16 x 16) + 32 = 32	SMLAD, SMLADX, SMLSD, SMLSDX
(16 x 16) ± (16 x 16) + 64 = 64	SMLALD, SMLALDX, SMLSLD, SMLSLDX
32 x 32 = 32	MUL
32 ± (32 x 32) = 32	MLA, MLS
32 x 32 = 64	SMULL, UMULL
(32 x 32) + 64 = 64	SMLAL, UMLAL
(32 x 32) + 32 + 32 = 64	UMAAL
32 ± (32 x 32) = 32 (upper)	SMMLA, SMMLAR, SMMLS, SMMLSR
(32 x 32) = 32 (upper)	SMMUL, SMMULR

DSP Instructions

The Cortex-M4 has a set of SIMD instructions aimed at supporting DSP algorithms. These instructions allow a number of parallel arithmetic operations in a single processor cycle.

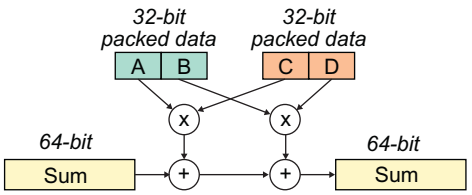

Figure 1.19
The SIMD instructions can perform multiple calculations in a single cycle.

The SIMD instructions work with 16- or 8-bit data which has been packed into 32-bit word quantities. So, for example, we can perform two 16-bit multiplies and sum the result into a 64-bit word. It is also possible to pack the 32-bit works with 8-bit data and perform a quad 8-bit addition or subtraction. As we will see in Chapter 7, the SIMD instructions can be used to vastly enhance the performance of DSP algorithms such as digital filters that are basically performing lots of multiply and sum calculations on a pipeline of data.

The Cortex-M4 processor may also be fitted with a hardware FPU. This choice is made at the design stage by the microcontroller vendor, so like the ETM and MPU you will need to check the microcontroller datasheet to see if it is present. The FPU supports single precision floating point arithmetic calculations using the IEEE 754 standard.

Table 1.3: Cortex-M4 FPU Cycle Times

Operation	Cycle count
Add/Subtract	1
Divide	14
Multiply	1
Multiply accumulate (MAC)	3
Fused MAC	3
Square root	14

On small microcontrollers, floating point math has always been performed by software libraries provided by the compiler tool. Typically, such libraries can take hundreds of instructions to perform a floating point multiply. So, the addition of floating point hardware that can do the same calculation in a single cycle gives an unprecedented performance boost. The FPU can be thought of as a coprocessor that sits alongside the Cortex-M4 CPU. When a calculation is performed, the floating point values are transferred directly from the FPU registers to and from the SRAM memory store, without the need to use the CPU registers. While this may sound involved the entire FPU transaction is managed by the compiler. When you build an application for the Cortex-M4, you can compile code to automatically use the FPU rather than software libraries. Then any floating point calculations in your C code will be carried out on the FPU.

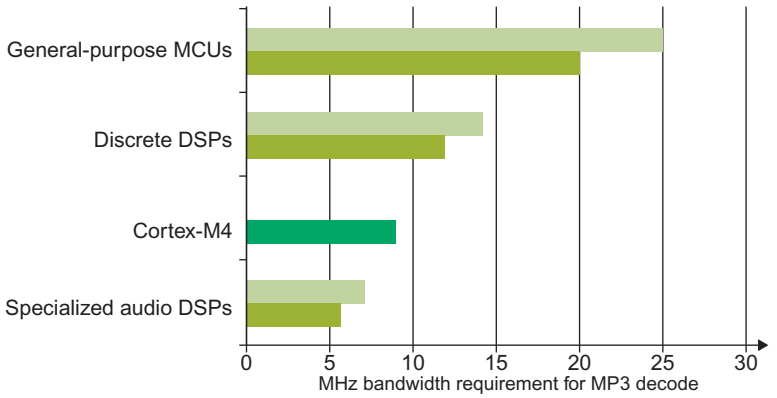

Figure 1.20
MP3 decode benchmark.

With optimized code, the Cortex-M4 can run DSP algorithms far faster than the standard microcontrollers and even some dedicated DSP devices. Of course, the weasel word here is "optimized." This means having a good knowledge of the processor and the DSP algorithm you are implementing and then hand coding the algorithm by making use of compiler intrinsics to get the best level of performance. Fortunately, ARM provides a full open source DSP library that implements many commonly required DSP algorithms as easy to use library functions. We will look at using this library in Chapter 7.