Application Note 237

Migrating from 8051 to Cortex™ Microcontrollers

Document number: ARM DAI 0237

Issued: July 2010

Application Note 237
Migrating from 8051 to Cortex Microcontrollers

Release information

The following changes have been made to this Application Note.

Change history

Date	Issue	Change
July 2010	A	First release

Proprietary notice

Confidentiality status

This document is Open Access. This document has no restriction on distribution.

Feedback on this Application Note

If you have any comments on this Application Note, please send email to errata@arm.com giving:

- the document title
- the document number
- the page number(s) to which your comments refer
- an explanation of your comments.

General suggestions for additions and improvements are also welcome.

ARM web address

http://www.arm.com

Table of Contents

1 Introduction

The ARM Cortex™-M range of microcontroller cores are high performance, low cost and low power 32-bit RISC processors. Currently, the range includes the Cortex-M3, Cortex-M4 and Cortex-M0 cores (the Cortex-M1 is similar in functionality and is targeted at implementation in FPGA devices). Cortex-M processors differ from other processors in ARM's range in that they execute only Thumb-2 instructions and do not support the ARM instruction set. They are based on the ARMv7-M architecture and have an efficient Harvard architecture 3-stage pipeline core. They also feature hardware divide and low-latency ISR (Interrupt Service Routine) entry and exit.

As well as the CPU core, the Cortex-M processors include a number of other components. These include a Nested Vectored Interrupt Controller (NVIC), an optional Memory Protection Unit (MPU), Timer, Debug Access Port (DAP) and optional trace facilities. They have a fixed memory map.

In this document, we only refer to the standard 8051 architecture. There are several extended versions of the architecture (e.g. 8052) which are support additional features (e.g. extra status flags, ability to address more memory etc.). These are not considered.

We compare the 8051 primarily with Cortex-M3 devices as these form the bulk of the microcontrollers available using Cortex-M cores. This document also contains an additional comparison with the Cortex-M0, which is in many respects a subset of the Cortex-M3, providing a lower cost solution, albeit at a lower performance point.

1.1 Why change to Cortex-M3

There are many reasons to take the decision to base a new design on a device incorporating a Cortex-M3 processor. Most, if not all, of these reasons also apply to the decision to migrate an existing product to Cortex-M3.

- **Higher performance**
  While exact performance is dependent on the individual device and implementation, the Cortex-M3 processor is capable of providing 1.25DMIPS/MHz at clock speeds up to 135MHz.
- **More memory**
  Since the Cortex-M3 is a full 32-bit processor, including 32-bit address and data buses, it has a 4GB address space. Within the fixed address space, up to 512MB is available for optimized code execution from flash (further code segments can be placed in other memory regions) and up to 2GB for RAM (either on or off chip). Significant space is also allocated for peripherals, system control registers and debug support.
- **Modern tools**
  The Cortex-M3 is well supported by a wide range of tools from many suppliers. In particular, the RealView Developer Suite (RVDS) and Keil Microcontroller Developer Kit (MDK) from ARM provide full support for Cortex-M3. Models are also available to accelerate software development.
- **Can program in C**
  Unlike many microcontrollers, the Cortex-M3 can be programmed entirely in C. This includes exception handling, reset and initialization as well as application software. Doing away with assembly code improves portability, maintainability and debugging.
- **More efficient interrupt handling**
  The interrupt architecture of the Cortex-M3 is designed for efficient interrupt entry and exit and also to minimize interrupt latency. The integrated Nested Vectored Interrupt Controller supports hardware prioritization, pre-emption and dispatch of external and internal interrupts. The core also supports late arrival, tail-chaining and nesting with minimal software intervention.

- **Future proof**
  The Cortex-M3 will meet the needs of the majority of today's microcontroller applications but, crucially, it provides an upwards migration path to the rest of the ARM architecture family of products. The Cortex-M4 adds a number of extra instructions providing floating-point and SIMD capability while retaining complete binary compatibility with Cortex-M3. Since programming is entirely in C, achieving extra performance by migrating to, for instance, Cortex-R4 is realistic and achievable with minimal engineering effort.

- **Use more capable OS/scheduler**
  The architecture of the Cortex-M3 provides excellent support for many standard RTOS's and schedulers. OS's can make use of the privileged "Handler" mode to provide inter-process isolation and protection. The built-in SysTick timer is ideal for system synchronization and can also function as a watchdog.

- **Better consistency between suppliers**
  Using a microcontroller based on an industry-standard architecture reduces risk by ensuring that products available from different suppliers are highly consistent and standardized. The engineering effort involved in moving from one supplier to another is minimized. The Cortex Microcontroller Software Interface Standard (CMSIS) is a widely-adopted API for many standard functions, making migration between suppliers easier.

- **Better debug facilities**
  The Cortex-M3 supports full in-circuit debug using standard debug adapters. There is full support for breakpointing, single-stepping and program trace as well as standard instrumentation features.

1.2 Cortex-M3 products

See www.onarm.com for the most comprehensive list of available Cortex-M3 devices, supporting technology and development tools.

1.3 References and Further Reading

Application Note 179 – Cortex-M3 Embedded Software Development, ARM DAI0179B, ARM Ltd.

Cortex Microcontroller Software Interface Standard (see www.onarm.com).

Cortex-M3 User Guide Reference Material, ARM DUI0450A, ARM Ltd.

Cortex-M3 Technical Reference Manual, ARM DDI0337G, ARM Ltd.

ARMv7-M Architecture Reference Manual, ARM DDI0403D, ARM Ltd.

2 Cortex-M3 Features

The Cortex-M3 processor core is highly configurable by the chip developer. For this reason, not all of the features described in this section will be available (or identically configured) on all Cortex-M3 microcontrollers. It is important to check carefully the feature set of the device you are using.

2.1 Nested Vectored Interrupt Controller (NVIC)

Depending on the silicon manufacturer's implementation, the NVIC can support up to 240 external interrupts with up to 256 different priority levels, which can be dynamically reprioritized. It supports both level and pulse interrupt sources. The processor state and a subset of core registers are automatically saved by hardware on interrupt entry and is restored on interrupt exit. The NVIC also supports tail-chaining of interrupts, allowing efficient handling of multiple pending interrupts.

The use of an NVIC in the Cortex-M3 means that the vector table for a Cortex-M3 is very different to previous ARM cores. The Cortex-M3 vector table contains the address of the exception handlers and ISR, not instructions as most other ARM cores do. The initial stack pointer and the address of the reset handler must be located at 0x0 and 0x4 respectively. These values are then loaded into the appropriate CPU registers at reset.

The NVIC also incorporates a standard SysTick timer which can be used as a one-shot timer, repeating timer, or system wake-up/watchdog timer.

A separate (optional) Wake-up Interrupt Controller (WIC) is also available. In low power modes, the rest of the chip can be powered down leaving only the WIC powered.

2.2 Memory Protection Unit (MPU)

The MPU is an optional component of the Cortex-M3. It provides support for protecting regions of memory through enforcing privilege and access rules. It supports up to 8 different regions which can be split into a further 8 sub-regions, with each sub-region being one eighth the size of a region.

2.3 Debug Interface

There are two different debug interfaces supported: the Serial Wire JTAG Debug Port (SWJ-DP) and the Serial Wire Debug Port (SW-DP). Your Cortex-M3 implementation might contain either or both of these depending on the silicon manufacturer's implementation.

2.4 Program and Data Trace

There are several possible components which may be implemented to allow tracing of program execution and data access. Printf-style program tracing may also be available. Check the documentation for your device to determine which features are available.

2.5 Fixed Memory Map

Unlike most previous ARM cores, the overall layout of the memory map of a device based around the Cortex-M3 is fixed. This allows easy porting of software between different systems based on the Cortex-M3. The address space is split into a number of different sections and is discussed further in section 3.3.1 below.

3 8051 and Cortex-M3 Compared

As you would expect, direct comparisons between two such different products are difficult. Both are available in many different configurations. The Cortex-M3 is arguably more standardized than the 8051, which exists in several quite different variants (some of which are obsolete). However, the Cortex-M3 offers several configurable options to the chip vendor (e.g. number of interrupts and depth of priority scheme, memory protection, debug configuration etc.).

In both cases, there is a huge variety of devices with a multitude of different peripheral sets, targeted at different end applications.

For the purposes of comparison, we have selected two devices from chip manufacturer Atmel. For the 8051 architecture, we will look at the AT89C51RD2; for the Cortex-M3, the SAM3S1A.

	AT8951C51RD2	**SAM3S1A**
Architecture	8051 (8052)	ARMv7-M (Cortex-M3)
Program memory (flash)	64 Kbytes	64 Kbytes
Data memory (RAM)	2 Kbytes	16 Kbytes
EEPROM	2 Kbytes	N
Memory Protection Unit	N	Y
Max clock frequency	60 MHz	64 MHz
GPIO pins	48	34
ADC	N	8-channel x 12-bit
Timers	3 x 16-bit	3 x 16-bit + SysTick
Watchdog timer	Y	Y
SPI	1	1
I2C	N	1
USART	1 (UART)	2
PWM	Y	Y
RTC	Y (up to 5 x 8-bit)	4 channel x 16-bit
External interrupt sources	2 (+ 7 internal)	34 (+ 16 internal)
Interrupt prioritization	4 levels	16 levels
Vectored Interrupt Controller	Y (two separately-vectored external interrupts)	Y
Power-saving modes	Idle/Power-Down	Sleep/Wait/Backup
DMA	N	4-channel
Debug port	ONCE mode	Serial sire or JTAG debug
Voltage Detection	Y	Y

3.1 Programmer's model

3.1.1 Register set

The two processors are significantly different with regards to the register set.

The 8051 has two 8-bit registers (ACC and B) which are generally used in arithmetic and logic instructions. Additionally, ACC is used as source and destination for most memory access instructions. There is also a 16-bit address register, DPTR, which is used exclusively for addressing external memory.

There are also four banks of eight 8-bit registers which are mapped into the first 32 bytes of internal RAM. These are byte and bit addressable. The bank in current use is selected via the RS0 and RS1 bits in the PSW (see below).

The Cortex-M3 has 16 32-bit registers, R0-R15. R0-R12 are generally available for essentially all instructions. R13 is used as the Stack Pointer, R14 as the Link Register (for subroutine and exception return) and R15 as the Program Counter. None of the Coretx-M3 core registers are directly addressable.

Since the Cortex-M3's registers are all 32-bit, it supports 32-bit arithmetic and logic operations efficiently. Such operations have to be synthesized from smaller ones when programming the 8051.

3.1.2 Status registers

The 8051 PSW register contains Carry (CY), Auxiliary Carry (AC) and Overflow (OV) flags (set on results from ALU operations), RS0 and RS1 (used to select one of four register banks in current use) and a parity flag, P (set by the hardware to indicate whether there are an even or odd number of set bits in the accumulator on each instruction cycle).

The Cortex-M3 Program Status Register (xPSR) is a single 32-bit register with several aliases, each providing a view of a different subset of the contents. From the user point of view, the Application Program Status Register (APSR) contains the ALU status flags. For operating system and exception handling use, the Interrupt Program Status Register (IPSR) contains the number of the currently executing interrupt (or zero if none is currently active). The Execution Program Status Register (EPSR) contains bits which reflect execution status and is not directly accessible.

3.1.3 Instruction set

The 8051 supports an 8-bit instruction set. Instructions are variable length and can extend to include operands in the instruction stream, varying from one to three bytes.

The Cortex-M3 supports a subset of the Thumb-2 instruction set. Instruction are either 16-bits or 32-bits in size. One crucial difference is that instructions cannot extend to include operand data. Any information for the instruction must be encoded within the instruction itself. One consequence of this is that it is not possible to specify arbitrary 32-bit constants or addresses in instructions so other methods must be used for this. C compilers supporting the Cortex-M3 all support this transparently to the programmer.

3.1.4 Operating modes

The 8051 does not support any concept of multiple operating modes.

The Cortex-M3 supports two modes, Thread mode (used for user processes) and Handler mode (used for handling exceptions and automatically entered when an exception is entered).

The ARMv7-M architecture also defines the concept of "privilege." Unprivileged execution limits or excludes access to some resources (for instance, unprivileged code is unable to mask or unmask interrupts). Handler mode execution is always privileged. By default, Thread mode will also execute with privilege but the programmer may configure thread mode to execute without privilege. This configuration can be used to provide a degree of system protection from errant or malicious programs.

3.1.5 Stack

The 8051 supports a Full Ascending stack using a single stack pointer, SP. This is constrained to like in internal RAM. SP is initialized to 0x07, meaning that the first item pushed on to the stack goes to address 0x08. The stack grows upwards from this point. It cannot grow beyond the end of internal RAM at 0xFF. This means that the 8051 only supports 248 bytes of stack space.

Since the internal RAM region is partially shared with the register banks (bank 0 is located from 0x0 to 0x7, banks 1-3 from 0x08 to 0x1F) and the bit-addressable area (0x20 to 0x 2F), using the full range of 248 bytes of stack space requires foregoing three of the register banks and the whole of the bit-addressable area. It is more common, therefore, for SP to be reset by the program to start at 0x30 during initialization. This restricts stack space to 192 bytes.

All stack accesses on the 8051 are byte-sized.

The 8051 stack pointer is typically initialized to the address one byte below the start of the allocated stack area so that the first push uses the first byte of the area (since the stack model is Full Ascending the stack pointer is incremented before the first store).

The Cortex-M3 supports a Full Descending stack addressed by the current stack pointer (see below). This stack can be located anywhere in RAM. Typically, for best performance, it will be located in internal SRAM. Stack size is limited only by the available RAM space.

The Cortex-M3 stack pointer is typically initialized to the word above the top of the allocated stack area. Since the stack model is Full Descending, the stack pointer is decremented before the first store, thus placing the first word on the stack at the top of the allocated region.

All stack accesses on the Cortex-M3 are word-sized.

3.2 Exceptions and interrupts

The 8051 supports five interrupt sources: IE0 and IE1 (external hardware interrupts INT0 and INT1), TF0 and TF1 (internal timer overflow interrupts) and RI/TI (receive and transmit interrupts from the integrated UART, handled as a single interrupt source. Each can be configured as high or low priority (two priority levels). Within the two priority groups, priority of individual interrupts is fixed. An interrupt of higher priority will pre-empt a lower priority interrupt. External interrupts may be configured as level or edge sensitive. Interrupts may be individually and/or globally enabled and disabled.

The Cortex-M3 has an integrated Nested Vectored Interrupt Controller which supports between 1 and 240 separate external interrupt courses. There are up to 256 priority levels. Individual implementations may configure the number of interrupts and the number of priority levels which are supported so it is important to check the manual for the target device to determine the exact configuration. The Cortex-M3 also supports an external Non-Maskable Interrupt (NMI) and several internal interrupts (e.g. HardFault, SVC etc.).

3.2.1 Interrupt prioritization and pre-emption

The 8051 supports two priority levels. These are configured for each interrupt via a single bit within the Interrupt Priority (IP) register. Interrupts of a higher priority will interrupt an currently executing interrupt. All interrupts can be masked in software

The Cortex-M3 priority scheme is significantly more flexible. Up to 8-bits (256 levels) of priority are supported though devices are able to implement fewer than the maximum number. Additionally, the priority levels may be subdivided into pre-emption priority and subpriority by setting a binary point position at which the two divide. For instance, on a system with 6 bits of priority, 8 levels of pre-emption priority and 8 levels of sub-priority may be configured by setting the binary point at bit 5. All external interrupts, except NMI, can be masked in software.

The priority of NMI is fixed.

3.2.2 External interrupts

The 8051 supports two external interrupts. INT0 is higher priority than INT1.

The Cortex-M3, as discussed above, supports up to 240 external interrupts, all of which may have separately configured priority, and a Non-Maskable Interrupt (NMI).

3.2.3 Internal interrupts

The 8051 recognizes internal interrupts relating to the two system timers and to the integrated UART. Each of the two timers can generate an interrupt on overflow. The UART TI (Transmit Interrupt) and RI (Receive Interrupt) conditions are combined to a single interrupt.

The Cortex-M3 supports a larger list of internal interrupts and exception events. These indicate errors (e.g. bus faults, undefined instructions), debug events and software interrupts.

3.2.4 Vector table

The 8051 vector table is located at a fixed address in the internal RAM area. Each of the five interrupt handlers is started at a fixed address within the vector table.

The Cortex-M3 vector table is located, by default, at address 0x00000000. It can be relocated during initialization to a location in either Code or RAM regions. Locating the vector table in internal SRAM can provide higher performance. Within the vector table, each entry contains the starting address of the corresponding handler routine.

3.2.5 Interrupt handlers

The 8051 requires that interrupt handler routines return with a Return from Interrupt (RETI) instruction. This means that such routines must either be written in assembler or flagged as handlers using some suitable keyword attribute if programming in a high level language.

The Cortex-M3 supports all exception entry and exit sequences in hardware and thus allows interrupt routines to be standard C functions, compliant with the ARM Architecture Procedure Call Standard (AAPCS). Any compliant function can be installed in the vector table as a handler simply by referencing its address.

3.3 Memory

The 8051 supports 64k of instruction memory (4k on-chip and up to 60k off-chip or 64k external) and up to 64k of external data memory. Internal data memory is limited to 128 bytes, some of which is given over to the register banks and the bit-addressable area. The external data memory is accessed via the special MOVX instruction.

The 8-bit 8051 stack pointer is restricted to the portion of the internal RAM between 0x08 and 0xFF, though to use all of this space the programmer must forego use of two of the register banks and also the bit-addressable area. It is more normal, therefore, to restrict the stack pointer to the region from 0x30 upwards, leaving only 192 bytes of stack space.

Both processors support bitwise access to memory or “bit-banding”. The 8051 bit-addressable area is 16 bytes long, contained within the internal RAM region, and can be directly accessed either as bytes or bits. Additionally, several bits within the Special Function Register (SFR) region can be directly accessed as bits. However, the transfer can only be between the memory bit and the Carry flag (C). The Cortex-M3 supports bit addressing across 1MB of each of the RAM and Peripheral regions. This is achieved by aliasing a further 32MB of address space in each region for direct atomic bit accesses in the bit-band region.

3.3.1 Memory map

The Cortex-M3 memory map is summarized in the table below. This is a unified address space covering both program and data regions as well as peripherals and system control registers.

Address	Region	Address	Detail
0x0000 0000	Code	Code memory (e.g. flash, ROM etc.)	
0x2000 0000	SRAM	0x2000 0000 - 0x200F FFFF	Bit band region
		0x2200 0000 - 0x23FF FFFF	Bit band alias
0x4000 0000	Peripheral	0x4000 0000 - 0x400F FFFF	Bit band region
		0x4200 0000 - 0x43FF FFFF	Bit band alias
0x6000 0000 - 0x9FFF FFFF	External RAM	For external RAM	
0xA000 0000 - 0xDFFF FFFF	External device	External peripherals or shared memory	
0xE000 0000 - 0xE003 FFFF	Private Peripheral Bus – Internal	0xE000 0000 - 0xE000 2FFF	ITM, DWT, FPB
		0xE000 E000 - 0xE000 EFFF	System Control Space
0xE004 0000 - 0xE00F FFFF	Private Peripheral Bus – External	0xE004 0000 - 0xE004 1FFF	TPIU, ETM
		0xE004 2000 - 0xE00F EFFF	MPU, NVIC etc
0xE010 0000 to 0xFFFF FFFF	Vendor-specific	For vendor-specific use	

The memory map implemented in the SAM3S1A is as follows. Regions which are not indicated are unimplemented.

Address	Region	Address	Detail
0x0000 0000	Code	0x0000 0000 - 0x007F FFFF	Internal boot memory
		0x0040 0000 - 0x0801 EFFF	64k Internal flash memory
		0x0080 0000 - 0x1FFF F7FF	Internal ROM
		0x1FFF F800 - 0x1FFF F80F	Option bytes
0x2000 0000	SRAM	0x2000 0000 - 0x2000 03FF	16k SRAM (bit banded)
		0x2200 0000 - 0x2200 03FF	Bit band alias region for SRAM
0x4000 0000	Peripheral	0x4000 0000 - 0x400E 25FF	Peripherals (bit banded)
		0x4200 0000 - 0x420E 25FF	Bit band alias for peripherals
0xE000 0000 - 0xE00F FFFF	Internal peripherals	0xE000 0000 - 0xE000 2FFF	ITM, DWT, FPB
		0xE000 E000 - 0xE000 EFFF	System Control Space
		0xE004 0000 - 0xE004 1FFF	TPIU, ETM
		0xE004 2000 - 0xE00F EFFF	SysTick, NVIC etc

The boot ROM and internal ROM contain a simple boot monitor, flash programming algorithms and some support code. The system may be configured (via hardware signals sensed at reset) to boot from the flash region rather than starting from the beginning of the internal boot ROM at address 0x0.

Since the amount of SRAM implemented on the SAM3S1A is wholly contained within the bit band region, all SRAM on this device is bit-addressable. Likewise, all peripherals in the Peripheral region are bit-addressable.

3.3.2 Memory protection

The 8051 does not support any form of memory protection either in hardware or software. All program instructions have access to all areas of memory.

The Cortex-M3 supports an optional Memory Protection Unit (MPU). When implemented, this allows access to memory to be partitioned into regions. Access to each region may then be restricted based on the current operating mode. This allows software developers to implement memory access schemes aimed at providing a degree of protection to the system from inerrant or malicious software applications.

When porting from 8051 to Cortex-M3 it may be desirable to take account of any memory protection features offered by the Cortex-M3 device but it is not necessary. At reset, the

MPU is disabled and the default memory map as described above applies. No memory protection configuration is required unless the MPU is to be enabled.

However, if the Cortex-M3 device incorporates caches and/or write buffers (these are not part of the architecture and, if present, are implemented externally) the cache and buffer policies are contained within the MPU configuration. In this case, it may be desirable for performance reasons to configure and enable the MPU.

The SAM3S1A device implements the standard MPU but does not include any cache or write buffer functionality.

3.3.3 Access types

8051 supports byte and bit accesses to memory. Bit accesses are supported via the bit-addressable region. Since the largest item which can be loaded from memory is 8-bits and the destination is also 8-bits, the sign of the loaded value is not important.

The Cortex-M3 is a 32-bit processor and all internal registers are 32-bit. Memory transfers of 8-bit bytes, 16-bit halfwords and 32-bit words are supported. In the case of bytes and halfwords, the programmer needs to specify whether the loaded value is to be treated as signed or unsigned. In the case of signed loads, the loaded value is sign-extended to create a 32-bit signed value in the destination register; in the case of unsigned loads, the upper part of the register is cleared to zero.

The Cortex-M3 also has instructions which transfer doublewords and also Load and Store Multiple instructions which transfer multiple words in a single instruction to and from a contiguous block of memory.

3.3.4 Bit banding

The 8051 bit-addressable region supports atomic bit accesses to 16 bytes within the internal RAM area. In additional, many bits within the Special Function Register area can be directly accessed as bits. However, a special version of the MOV instruction must be used to do this and the transfer can only be to or from the Carry flag (C).

The Cortex-M3 scheme is more flexible in that bit access is provided to two 1MB regions of memory, one within the internal SRAM region and the other in the peripheral region. A further 32MB of address space is reserved for bit accesses and each word within these regions aliases to a specific bit within the corresponding bit-band region. Reading from the alias region returns a word containing the value of the corresponding bit; writing to bit 0 of a word in the alias region results in an atomic read-modify-write of the corresponding bit within the bit-band region. Thus, bit accesses can be achieved using standard memory access instructions across a very wide region of memory, covering both data RAM and peripheral space.

In the case of the SAM3S1A, the internal RAM is smaller than the bitband region so atomic bitwise access is possible across all available RAM using standard load and store instructions.

The ARM/Keil development tools support bit-banding via compiler extensions.

3.4 Debug

8051-based devices support a range of debug options. In many cases a dedicated In-Circuit Emulator must be connected in place of the processor in order to debug an application. The Atmel at89C51RD2 supports a mode called ONCE (On-Chip Emulation mode) which allows debug in-circuit. This involves placing the device into a special mode in which an external emulator can be connected.

Cortex-M3 devices are debugged via a standard JTAG or Serial-Wire Debug (SWD) connector. A simple, standardized external connector is required to interface to the host system. The SAM3S1A supports both types of debug connection. Within the core, a Flash Patch Breakpoint unit (FPB) provides the ability to implement breakpoints and field code patches, a Data Watchpoint and Trace unit (DWT) allows for data access watchpoints and data tracing, a Instrumentation and Trace Macrocell (ITM) supports printf-style debugging even when no suitable output device is present (the output from ITM is carried over the JTAG interface to the host system when connected).

Since the debug features and the debug connection are standardized across all Cortex-M3 devices, any compatible external debug adapter (sometimes called a JTAG probe) may be used, in conjunction with any available debug application supporting Cortex-M3.

In addition, the uVision simulator from Keil and the MPLAB IDE provide software simulation of target devices. In the case of uVision, this can include simulating external components at board level.

3.5 Power management

The 8051 supports two power-saving modes: Power-Down Mode and Idle Mode. These are entered by setting the PD and IDL bits within the Power Control (PCON) register. The effect on system operation and the resulting power-saving is device-dependent. The AT91C51RD2 device under consideration stops the CPU clock in Idle mode but continues to clock peripherals; in Power-Down mode, all clocks are stopped. In both modes, the CPU status, contents of Special Function registers and contents of RAM are retained. Exit from either mode is via an enabled interrupt or reset.

Architecturally, the Cortex-M3 supports Sleep and Deep Sleep modes. The exact power saving which can be realized in these two modes depends to a great extent on the external logic implemented by the processor manufacturer. Developers should consult the manual for the device they are using for further details if required.

In Sleep mode, external logic is usually configured to stop the processor clock, minimizing power consumption. The power and clock to the NVIC is maintained, so that an exception can exit sleep mode.

In Deep Sleep mode, the processor can be powered down completely, usually leaving only the external Wakeup Interrupt Controller (WIC) active. The WIC will wake the processor if any unmasked external interrupt is detected.

Sleep mode can be entered in the following ways.

- **Sleep-now**
  The Wait-For-Interrupt (WFI) or Wait-For-Event (WFE) instructions cause the processor to enter Sleep mode immediately. Exit is on detection of an interrupt or debug event.
- **Sleep-on-exit**
  Setting the SLEEPONEXIT bit within the System Control Register (SCR) causes the processor to enter Sleep mode when the last pending ISR has exited. In this case, the exception context is left on the stack so that the exception which wakes the processor can be processed immediately.

In addition, Deep Sleep mode can be entered by setting the SLEEPDEEP bit in the SCR. On entry to Sleep mode, if this bit is set, the processor indicates to the external system that deeper sleep is possible. The behavior of the external system as a result is device-specific – you should consult the documentation for the device you are using to determine exactly what action is taken.

3.6 Extensions implemented in the Atmel AT91C51RD2

As stated earlier, this device is compatible with the standard 8052 architecture (8052 is a slight enhancement of the 8051 standard providing, among other things, slightly more internal RAM space and an extra timer). However, the manufacturer has implemented some enhancements to the standard. Note that these enhancements are proprietary and may not be supported by all software development tools.

- **Dual Data Pointer**
  There are two instances of the data pointer register DPTR. These can be selected via a single bit in the AUXR configuration register. This makes for more flexibility in addressing external data memory.
- **Expanded RAM (XRAM)**
  The EXTRAM bit in the AUXR configuration register allows certain instructions to access an extended region of on-chip RAM. In the particular device considered in this document, this region is 1792 bytes.
- **4-level interrupt priority system**
  An additional Interrupt Priority Register (IPH) allows the standard 2-level priority scheme to be extended to 4 levels.

There are also some enhancements to the standard UART and Timer functions provided by standard 8051 parts.

4 Migrating a software application

When making comparisons which depend on the development tools in use, we ARM's RealView Microcontroller Development Kit (MDK). This easily available toolset has variants supporting Cortex-M3, Cortex-M3 and 8051.

4.1 General considerations

4.1.1 Operating mode

The Cortex-M3 will reset in Thread mode, executing as privileged. Handler mode (also privileged) is automatically entered when handling any exceptions which occur. Unless the programmer takes explicit steps to configure the core differently, the current mode is transparent and can this essentially be ignored when migrating software.

Since the 8051 does not support more multiple operating modes and has no concept or privilege, there is generally nothing for the developer to do here.

In order to take advantage of the protection offered by the privileged execution Thread mode can be configured to be unprivileged by setting CONTROL[0]. Unprivileged execution is prohibited from carrying out some system operations, e.g. masking interrupts.

4.1.2 Stack configuration

The 8051 stack is fixed and does not need to be configured by the programmer. It starts at a fixed address and is constrained to lie within on-chip RAM. The Cortex-M3 allows the stack to be placed anywhere in a suitable region of RAM.

The Cortex-M3 takes the initial value for the Main Stack Pointer (SP_main) from the first word in the vector table. This must be initialized to an area of RAM. Ideally this should be internal SRAM for best performance. Unless configured otherwise (see below) the Cortex-M3 will use this single stack pointer in both Thread and Handler modes. Since the 8051 only supports a single stack pointer, this is the simplest configuration to use when migrating to Cortex-M3.

The initial value for the Cortex-M3 stack pointer is typically placed in the vector table automatically once the tools have been configured with the correct address map information. The exact method for doing this varies between tools.

The Cortex-M3 can be configured to use the separate Process Stack Pointer (SP_process) when in Thread Mode. This is done by writing to the CONTROL[1] bit. Setting this configuration allows separate stacks to be used for normal execution and exception handling. The initial value for the Process Stack Pointer must be set during system initialization.

When compiling for Cortex-M3, the C compiler will place all local variables and function parameters on the stack. This means that there are no problems with re-entrant functions as with 8051 but stack usage is likely to be higher than for the same program running on an 8051. Sufficient stack space must therefore be provided. When allocating stack space, ensure that you take account of any usage required by exceptions.

Compilers targeting 8051 allocate automatic variables in static data memory (due to the restricted stack model) so stack usage will generally be lower but, correspondingly, RAM usage may be higher.

4.1.3 Memory map

Unless an MPU is present and enabled, the default memory map described above is used. When migrating an application from 8051 it is not usually necessary to implement any memory map configuration.

Microcontrollers using a Cortex-M3 processor can be built with many different memory devices. Usually, there will be some internal Flash or ROM (mapped to the CODE region) and internal SRAM (in the SRAM region). Any peripherals will be mapped to the Peripheral region. There may also be some external RAM.

Consult the manual for your chosen device to determine exactly what memories have been implemented and how they are mapped.

In any event, the system control registers and standard peripherals (such as the SysTick timer etc) will be located in the standard location.

4.1.4 Code and data placement

The 8051 architecture strictly segregates code and data within the memory map. Further, it requires that data items be assigned to one of the available memory regions.

data	Internal on-chip RAM, directly addressed using 8-bit addresses and restricted to 128 bytes or less.
idata	Internal on-chip RAM, indirectly accessed using 8-bit addressing. Limited to 256 bytes, the lower half of which overlaps the data region.
bdata	Internal on-chip bit-addressable RAM.
xdata	External data memory, accessed using 16-bit addressing and restricted to 64Kbytes or less,
pdata	One page of external data memory, limited to 256 bytes and accessed using 8-bit addressing.

In simple systems, much or all of this assignment is carried out implicitly by specifying a memory model, In all models, executable instructions are always allocated to code memory. MDK supports the following standard models for data allocation:

SMALL	All variables placed in internal on-chip RAM (data region)
COMPACT	All variables placed in a single 256-byte page of external memory (pdata region)
LARGE	All variables placed in external data memory (xdata region)

The ARMv7-M architecture of the Cortex-M3 support a single, unified address space. All memory access instructions can be used to address items in all regions of memory. This is much more flexible, allowing code and data to be essentially freely allocated across all available memory regions. The compiler and linker automatically handle run-time initialization of volatile regions from the contents of ROM following reset.

On-chip memory (including many control registers) can be accessed using bit addressing on an 8051. The Cortex-M3 provides two regions, one in RAM and one in the peripheral region, which support this. In both cases, a read-modify-write operation to a single bit is atomic. In the ARM case, standard memory access instructions are used with an aliased address to achieve the bit access and specialized bitwise addressing modes are not required as with the 8051.

One frequent requirement is the need to place a data item or code sequence at an absolute location. When using the Keil tools on the 8051, this is done using the “at” function qualifier. This means that these absolute addresses are encoded in the C source code – this can obviously lead to portability and maintenance problems.

When developing for Cortex-M3 parts, all code and data is relocatable with the final placement in memory being decided at link time. The input configuration to the linker (called a “scatter control file”) allows code and data segments to be placed at absolute addresses. It is not necessary to specify any addresses in the C source code, this improving maintainability.

4.1.5 Data types and alignment

When programming in a high-level language, the natural data type of the underlying machine is not necessarily important. The C compiler will take care of mapping high-level data types to low-level arithmetic operations and storage units.

Both compilers support a variety of types, as listed in the table.

Type	Cortex-M3	8051	Notes
`char`	8-bit signed	8-bit signed	
`short`	16-bit	16-bit	
`int`	32-bit	16-bit	int is smaller on 8051
`long`	32-bit	32-bit	
`long long`	64-bit	N/A	
`float`	32-bit	32-bit	
`double`	64-bit	32-bit	8051 has no 64-bit floating point type
`long double`	64-bit	N/A	

The Cortex-M3 is a 32-bit architecture and, as such, handles 32-bit types very efficiently. In particular, 8-bit and 16-bit types are less efficiently manipulated, although they will save on data memory if this is an issue.

The sbit keyword is used to declare items which are bit-addressable on 8051. This is unnecessary in the Cortex-M3 as bit-banding provides bit-addressable accesses via standard memory access instructions to aliased addresses. See section 5.2 for more details.

Cortex-M3 devices, in common with other ARM architecture devices, generally require that data be aligned on natural boundaries. For instance, 32-bit objects should be aligned on 32-bit (word) boundaries and so on. The core supports unaligned accesses only in the code and data regions (not the peripheral regions) though there is a slight performance penalty involved since the hardware performs two separate bus transactions. Access to unaligned data items can easily be support in C using the "__packed" attribute.

Since data memory is always accessed as bytes, 8051 devices have no such alignment restrictions. However, being restricted to byte-wise accesses does mean that performance is significantly lower when dealing with multi-byte quantities.

4.1.6 Pointers

8051 has memory-specific and generic pointer types. Generic pointers are stored using three bytes, the first of which encodes the memory type. Larger and slower. Memory-specific pointers can only be used to access the declared type of memory. They are quicker and smaller but less flexible.

There is no need for any such distinction when coding for Cortex-M3. Since all pointers are 32 bits in size and are held in 32-bit registers, the address of any location in memory can be held within a single register and all pointers are treated identically. Any such pointer qualifiers can and should be removed from code which is being ported.

4.1.7 Function declarations

8051 requires that re-entrant functions be declared as such. This is primarily because function parameters are statically allocated in memory to avoid using the limited stack space (only return addresses are stacked). This makes functions inherently non-reentrant.

Also the memory model used by the function may be declared if different from the program-wide default.

Interrupt handlers may be written in C for the 8051 and are usually declared using the “interrupt” qualifier. This identifies the routine as an interrupt handler and also associates it with a specific interrupt source. It is also necessary to specify (via the “using” keyword) which register bank the handler will use in order to avoid corrupting foreground registers.

None of this is required for Cortex-M3. Interrupt handlers are declared and coded exactly as standard C routines. The hardware which handles exception of the Cortex-M3, together with the register usage rules in the ARM Architecture Procedure Call Standard, automatically ensures that all foreground registers are preserved.

4.2 Tools configuration

Existing 8051 applications may have been developed with any of the wide range of available tool chains supporting 8051. Many of these will come from vendors (e.g. Keil) who also provide ARM support. In these cases, the simplest option may be to remain with the same tool vendor and purchase a different variant of the tool. The clear advantage here is that many tool-specific features will remain unchanged.

In general, very little of the configuration of the tools will need to change beyond the following.

* Memory map, code and data placement
* Any options which relate to particular target 8051 devices, platforms, processors or boards. When deciding on the ARM options, it is good practice to be as specific as possible with respect to the processor and architecture you are using.
* If your application uses floating point, then you will need to configure carefully for either hardware floating point or soft emulation.

There is also the option of using the ARM RealView tools. Clearly, more significant reconfiguration will be required here and you should refer to the RealView documentation (all available on ARM’s website) for further information on this.

4.3 Startup

8051 devices begin execution at the rest vector, located at address 0x0000. The stack pointer is automatically initialized to address 0x0007 in the internal RAM region (this means that the first byte on the stack is stored at address0x0008). The reset vector addresses a startup routine which is responsible for initializing the state of the system e.g. disabling interrupts, initializing peripherals, clearing memory etc.

Generally, the standard startup software provided with the tools will initialize the C runtime environment and then jump automatically to the entry of the C code at main().

Cortex-M3 devices take the initial value for the Main Stack Pointer from the first word of the vector table (at address 0x00000000) and then begin execution by jumping to the address contained in the reset vector (at address 0x00000004).

The C startup code will initialize the runtime environment and then call main().

If you choose not to use the startup code provided with the Cortex-M3 development tools, then the entire routine may be coded in C.

4.4 Interrupt handling

8051 devices have 8 interrupt sources, mapped to six handlers via a fixed vector table. Each vector table entry typically contains a LJMP instruction directing execution to the relevant handler. There is a simple two-level priority scheme for internal interrupts allowing each to be assigned high or low priority. This is configured via a single bit for each interrupt in the IP (Interrupt Priority) register. Within these levels, the underlying priority scheme is fixed.

In order to use the two external interrupts (which are shared with I/O pins in Port 3), the pins must be configured as inputs. Whether they are edge or level sensitive must also be configured via the IT bits in the TCON register.

The NVIC on Cortex-M3 devices supports a much more general priority scheme and also provides vectoring in hardware. All interrupt sources have separate vectors defined in the vector table. In parallel with saving context on the stack, the NVIC reads the relevant vector address from the table and directs program execution directly to the start of the correct handler.

Writing interrupt handlers

Interrupt handler routines for 8051 devices must be identified in C source code using the “interrupt” keyword. In addition, the register bank to be used by the interrupt handler must be specified using the “using” keyword.

Cortex-M3 interrupt handlers are standard C functions and there are no special C coding rules for them.

Vector table generation

8051 interrupt vectors are at fixed locations in memory. Typically, a LJMP instruction to the appropriate handler is located at this absolute address using the “at” keyword in assembler or C source code.

On Cortex-M3 devices, the vector table is typically defined by an array of C function pointers. This is then located at the correct address by the linker.

See below for an example.

Interrupt configuration

In order to enable an 8051 interrupt, the following must be set correctly:

- Individual Interrupt Enable bit in IE (Interrupt Enable) register set to 1
- EA (Enable All) bit in IE register set to 1
- For external interrupts, the relevant pins must be configured as inputs and also configured correctly for edge or level sensitivity

On Cortex-M3 devices, the NVIC must be configured appropriately:

- Interrupt Set Enable register
- PRIMASK must be clear to enable interrupts

This is achieved using CMSIS intrinsic functions __enable_irq().and NVIC_EnableIRQ(). See the CMSIS documentation for more detail. CMSIS functions are also available for setting and clearing pending interrupts, checking the current active interrupt and configuring priority.

Faults

The 8051 does not have a standard fault-handling mechanism for handling events such as an attempt to access an invalid memory location. The resulting value will not necessarily be valid or useful but the program itself will not generate an error.

In contrast, the Cortex-M3 will generate a BusFault exception in this situation. This requires the programmer to implement at least a minimal handler for this event.

The four types of fault are:

- MemManage
  This is used for memory protection faults determined by the Memory protection Unit (if implemented) or by fixed system constraints on the memory map.
- BusFault
  This results from a memory access error at hardware level e.g. an attempt to access an invalid memory location.
- UsageFault
  This indicates an attempt to use an instruction in an invalid way or carry out an invalid operation. Examples include execution of an undefined instruction, division by zero.
- HardFault
  This is generally used for unrecoverable error conditions.

It is advisable to implement at least a minimal handler for each of these events even if they are not expected to occur in normal operation.

4.5 Timing and delays

It is common, when programming for 8051 devices, to use NOP instructions as a way of consuming time. The execution time of NOP instructions is easily determined from the system clock speed.

When developing for Cortex-M3 devices this cannot be relied on as the pipeline is free to “fold out” NOP instructions from the instruction stream. When this happens, they do not consume time at all. Deterministic delays in Cortex-M3 systems must therefore make use of a timer peripheral (e.g. the internal SysTick timer).

4.6 Peripherals

4.6.1 Standard peripherals

On the Cortex-M3 it is usual to define a structure covering the System Control Space. All of the system control registers are located in this region. The standard peripherals (NVIC, SysTick etc.) are all controlled via registers in this area.

4.6.2 Custom peripherals

Custom peripherals on Cortex-M3 microcontrollers are generally memory-mapped within the defined peripheral region in the memory map. The usual method of access to these is to define C structures which describe the relevant registers. These can then be located at absolute addresses at link time via the scatter control file.

4.7 Power Management

ANSI C cannot generate the WFI and WFE instructions directly. If you are using ARM/Keil development tools, the compiler includes portable intrinsic functions for generating these. You could also use the CMSIS intrinsic functions __WFE() and __WFI() in your source code. If suitable device-driver functions are provided by the vendor, these can also be used.

4.8 C Programming

- Don’t bother with static parameters or overlays. Cortex-M3 does not suffer from data RAM size constraints like the 8051.
- Unless memory size is a constraint, don’t bother with data types smaller than one 32-bit word. They are less efficient on Cortex-M3.
- Any qualifiers relating to memory areas (e.g. idata, xdata, bdata, pdata etc) can be removed.
- Since Cortex-M3 pointers can address items in any memory region, there is no need to declare objects as “near” or “far”. All such qualifiers should be removed.
- There is no need to specify which register bank is to be used by interrupt handlers.
- No need to designate interrupt handlers using any special keywords. Though it is regarded as good programming practice to declare interrupt handlers using the __irq keyword when using the ARM/Keil tools to develop for Cortex-M3.
- It is unlikely that you will encounter any data alignment problems but the __packed keyword should be used to resolve any which do arise. Any data items which are actually or potentially unaligned should be declared as __packed. Be particularly careful with pointers.
- There is no need to specify any particular memory model (e.g. small, compact, large etc.)
- Any assembler in your 8051 source will need to be re-written. Typically it should be rewritten in C rather than translated to ARM assembler – this will be easier and more portable. Since modern C compilers produce very efficient code for 32-bit targets like the Cortex-M3, writing your code directly in C will usually be the best solution.
- Any #pragma directives will need to be either removed or replaced. Those associated with associated with placement code or data in C source code must be removed.
- There is no need to declare functions as “reentrant” when compiling for Cortex-M3. Nor is there any need to specify which register bank is used by a function.
- By default, many mathematical functions on 8051 default to 32-bit single-precision floating point. The corresponding ARM functions default to 64-bit. To retain the same precision, the function name may need to be changed e.g. sin() changed to sinf().

5 Examples

5.1 Vector tables and exception handlers

5.1.1 In assembler

The examples below show definition of vector tables and placeholders for exception handlers when writing in assembly code. Note that it is possible to avoid C startup for Cortex-M3 systems completely.

8051

```
$INCLUDE (reg_c51.INC)

; reset vector jumps to initialization
org 000h
ljmp begin

; example vectors
org 03h
ljmp extint0

org 23h
ljmp serial_IT

; main program starting at address 0x0100
org 0100h

begin:
;
JMP $

; example serial interrupt handler

serial_IT:

JNB RI,EMIT_IT ; test RI flag
CLR RI         ; clear RI flag
MOV A,SBUF     ; read UART data
MOV SBUF,A     ; write UART data

LJMP END_IT    ; jump to end
EMIT_IT:
CLR TI         ; clear TI flag

END_IT:
RETI           ; end with RETI instruction

extint0:
; write INT0 handler here

RETI


end
```

Cortex-M3

```
; Vector Table Mapped to Address 0 at Reset
 AREA    RESET, DATA, READONLY
 EXPORT  __Vectors

__Vectors
 DCD   __initial_sp      ; Initial SP
 DCD     Reset_Handler  ; Reset Handler
 DCD     NMI_Handler    ; NMI Handler
 DCD     HardFault_Handler
 DCD     MemManage_Handler
 DCD     BusFault_Handler
 DCD     UsageFault_Handler
 DCD     0, 0, 0, 0       ; Reserved
 DCD     SVC_Handler
 DCD     DebugMon_Handler
 DCD     0                ; Reserved
 DCD     PendSV_Handler
 DCD     SysTick_Handler

; External Interrupts from here
 DCD     InterruptHandler0
 DCD     InterruptHandler1
 DCD     InterruptHandler2

; etc

 AREA    |.text|, CODE, READONLY
; Reset Handler

Reset_Handler   PROC
                EXPORT  Reset_Handler
                IMPORT  __main
                LDR     R0, =__main
                BX      R0
                ENDP

InterruptHandler0
; write your IRQ0 handler here
;
 BX lr

 END
```

5.1.2 In C

These two examples show how the same is achieved when coding in C.

8051

```
#include "reg_c51.h"
char uart_data;

/* standard startup sequence calls main
 * following initialization
 */
void main (void)
{

/* write main program here */

}

/* serial interrupt handler */
void serial_IT(void) interrupt 4
{
  /* serial interrupt handler here */
}
```

Cortex-M3

```
/* Filename: exceptions.c */
typedef void(* const ExecFuncPtr)(void);

/* Place table in separate section */
#pragma arm section rodata="vectortable"

ExecFuncPtr exception_table[] =
{
(ExecFuncPtr)&Image$$ARM_LIB_STACKHEAP$$ZI$
$Limit,                    /* Initial SP */
    (ExecFuncPtr)__main,   /* Initial PC */
    NMIException,
    HardFaultException,
    MemManageException,
    BusFaultException,
    UsageFaultException,
    0, 0, 0, 0,             /* Reserved */
    SVCHandler,
    DebugMonitor,
    0,                      /* Reserved */
    PendSVC,
    SysTickHandler,

    /* Configurable interrupts from here */
    InterruptHandler0,
    InterruptHandler1,
    InterruptHandler2
    /*
     * etc.
     */
};

/* One example exception  handler */
#pragma arm section
void SysTickHandler(void)
{
    printf("---- SysTick Interrupt ----");
}


In Scatter Control File:

LOAD_REGION 0x00000000 0x00200000
{
  ;; 256 exceptions (256*4 bytes == 0x400)
  VECTORS 0x0 0x400
  {
    exceptions.o (vectortable, +FIRST)
  }
}
```

5.2 Bit banding

As described above, both devices support bit access to certain areas of memory. In both cases, bit accesses are atomic.

The 8051 supports this through a direct bit addressing mode in many instructions. Individual bits within many of the Special Function Registers and within the internal RAM memory can be addressed like this.

Cortex-M3 devices support bit access via a different method entirely. Within, for example, the SRAM region of the memory map, 1MB is designated as the “bit band region”. A second 32MB region, called the bit band alias region, is used to access the bits within the bit band region. Bit 0 of each word in the alias region is mapped within the memory system to a single bit within the bit band region. Bits can be read and written. Reading or writing any bit other than bit 0 in a word in the alias region has no effect.

A simple formula converts from bit address to aliased word address.

```
word_addr   = bit_band_base
            + (byte_offset x 32)
            + (bit_number x 4)
```

C macros can then be easily defined to automate this process. For example

```
#define BITBAND_SRAM(a,b) ((BITBAND_SRAM_BASE \
                        + (a - BITBAND_SRAM_REF) * 32 \
                        + (b * 4)))
```

A similar macro can be defined for the peripheral region.

Individual bits can then be accessed using sequences like this.

```
#define MAILBOX   0x20004000
#define MBX_B7    *((volatile unsigned int *) \
                  (BITBAND_SRAM(MAILBOX,7)))

a = MBX_B7;
```

The C compiler provides direct support for bit-banding via the “bitband” attribute which can be used with structure bitfields. The resultant code will use bitband accesses for all single-bit bitfields, For example:

```
typedef struct {
  char i : 1;
  int j : 2;
  int k : 3;
} BB __attribute__((bitband));

BB bb __attribute__((at(0x20000004)));

void foo()
{
  bb.i = 1; /* will use bitband access */
}
```

Note the use of the “at” attribute to place the structure at an absolute address within the bitband region. This could also be achieved at link-time using e.g. scatter-loading.

Note that this latter method is not portable since the “bitband” keyword is an ARM-proprietary feature.

5.3 Access to peripherals

There are device-specific header files for all supported 8051 devices. These are included with most, if not all, development tools for 8051 (e.g. Keil). These header files define all registers and systems constants e.g. available memory size etc.

Similarly, header files are usually provided for Cortex-M3 devices. You can obtain these either from the device supplier or use those included with many development tools. Keil MDK-ARM includes header files for most common devices.

Developers for Cortex-M3 platforms should be aware of the Cortex Microcontroller Software Interface Standard (CMSIS). This defines a standard software application interface for many standard peripherals (e.g. SysTick, NVIC) and system functions (e.g. enable/disable interrupts) on Cortex-M3 platforms. This covers the set of standard peripherals and core functions. Most Cortex-M3 device manufacturers supply additional CMSIS-compliant header files which provide definitions for all device-specific functions and peripherals.

6 Comparison with Cortex-M0

So far we have compared the 8051 architecture against a Cortex-M3 device. However, there is a lower-cost, lower-power Cortex-M core available in the form of the Cortex-M0.

Introduced by ARM in 2009, the Cortex-M0 implements a subset of the Cortex-M3, in terms of architecture and functionality, in a much smaller silicon footprint. It is therefore capable of operating at much lower power and can be purchased at significantly lower cost.

The Cortex-M0 support a smaller instruction set than the Cortex-M3 (56 instructions as compared to 101) and has a smaller number of external interrupt sources (up to 32 as compared to the maximum of 240 supported by Cortex-M3).

NXP LPC1114 is a typical Cortex-M0 microcontroller. Its largest variant provides 32k of on-chip flash memory and 8k of on-chip SRAM.

From a software developer’s point of view, especially when writing in C, the two cores can be treated as essentially identical.

The two cores support the same memory map, exception model and register set.