Migrating to Cortex-M3 Microcontrollers: an RTOS Perspective

Microcontroller devices based on the ARM® Cortex™-M3 processor specifically target real-time applications that run several tasks in parallel. The Keil™ RTX Real-Time Kernel has been optimised for Cortex-M3 processor-based microcontrollers and as a result it runs on them twice as fast as on comparable ARM7TDMI® processor-based microcontrollers. This article discusses the new benefits of Cortex-M3 processor-based microcontrollers, the modifications done to RTX and the results achieved.

Requirements of Real-Time Systems

Most microcontroller-based embedded systems are real-time systems that have to respond within predefined time limits to external events. This is a key constraint in the design of both the hardware and software of these systems.

Thanks to the performance and functionality of 32-bit microcontrollers, real-time embedded systems often have a single microcontroller device. This enables low unit cost, but creates challenges for software developers, as a single application must respond timely to a larger number of increasingly complex peripherals.

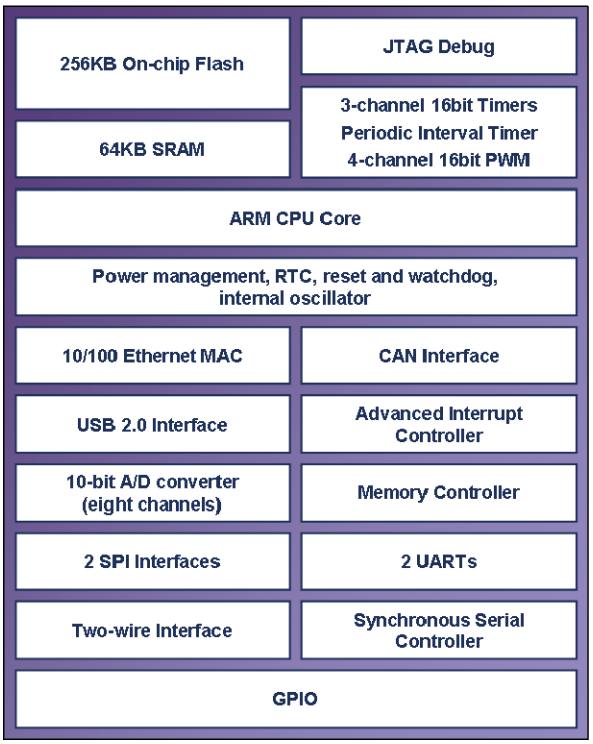

Figure 1: Block diagram of a typical 32-bit microcontroller

Real-time software is normally structured as a number of tasks running concurrently, with task scheduling handled by a real-time kernel. The infrastructure provided by the real-time kernel enables developers to define and control easily the timeliness of their software.

The main requirements for the hardware are high performance, which enables more tasks to be executed in parallel, and low interrupt latency, which enables fast reaction to external events.

Cortex-M3 processor-based Microcontroller Benefits for Real-Time Systems

Cortex-M3 processor-based microcontrollers delivers higher performance and lower interrupt latency than microcontrollers based on the older, but still popular, ARM7TDMI processor.

	ARM7TDMI	Cortex-M3
Instruction set	Thumb®/ARM	Thumb-2
Memory interface	Single interface, data read/write takes 3 cycles	Separate instruction and data bus interfaces, single cycle data read/write
Pipeline	3-stage	3-stage with branch speculation
Multiplication	8-bit hardware multiplier, result in up to 5 cycles	8/32-bit hardware multiplier, result in 1 cycle
Division	In software, result in 100s of cycles	Hardware divider, result in up to 12 cycles
Bit manipulation	Read – modify - write	Single instruction

Table 1: Characteristics of ARM7TDMI and Cortex-M3 processor

The Cortex-M3 processor implements the 16/32-bit Thumb-2 instruction set. Building C/C++ code for the Thumb-2 instruction set results in fewer instructions than building it for Thumb technology. Since most Thumb and Thumb-2 instructions are executed in a single cycle, Cortex-M3 microcontrollers can execute the same program in fewer cycles.

The ARM7TDMI processor also supports the 32-bit ARM instruction set, which delivers similar performance as Thumb-2 technology at the expense of larger code size. In practice, using ARM instructions is not common in microcontroller-based systems because of unit cost constraints.

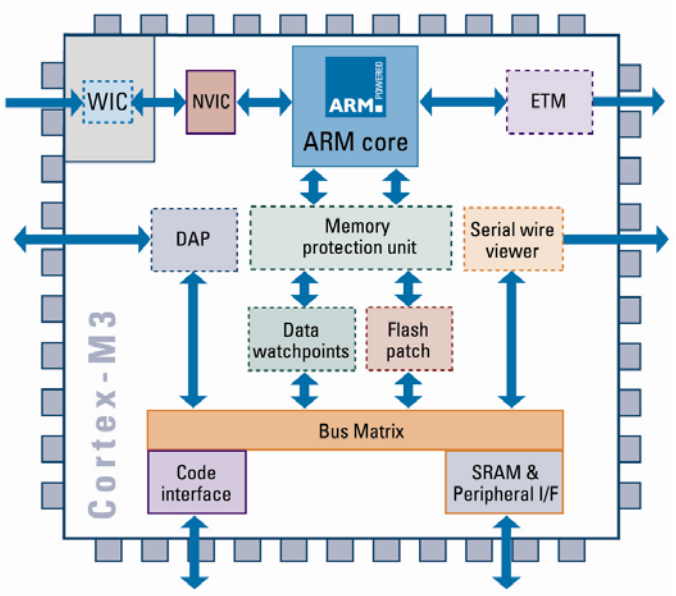

Figure 2: Cortex-M3 processor block diagram

	ARM7TDMI	Cortex-M3
Interrupt controller	External to processor	Integrated Nested Vectored Interrupt Controller
Interrupt handlers	One fast (nFIQ) and one slow (nIRQ)	One handler per interrupt source
RTOS system timer	Uses one timer of the microcontroller	Uses integrated “SysTick” timer on the processor
System calls	SWI instruction (interrupts disabled)	SVC instruction (interrupts enabled)

Table 2: Exceptions on ARM7TDMI and Cortex-M3 processors

By integrating the interrupt controller in the processor, Cortex-M3 processor-based microcontrollers have one interrupt vector entry and interrupt handler per interrupt source. This avoids the need for re-entrant interrupt handlers, which have a negative effect on interrupt latency.

The Cortex-M3 processor also accelerates the execution of interrupt handlers by incorporating logic to automatically save its general purpose and status registers in the stack when an interrupt arrives. The Cortex-M3 processor is comparatively even more efficient by tail-chaining interrupts that arrive at the same time, as shown in Figure 3.

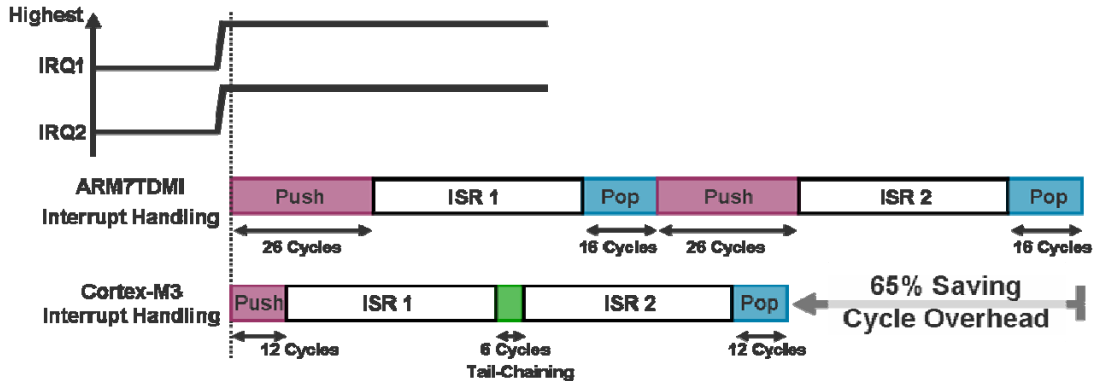

Figure 3: Tail-chaining on Cortex-M3 processor

The Keil RTX Real-Time Kernel

RTX is a real-time kernel for ARM7™ and ARM9™ family, and Cortex-M3 processor-based devices. RTX helps create real-time programs by solving scheduling, maintenance and timing issues. RTX allows flexible scheduling of system resources like CPU and memory and offers several ways to communicate between tasks.

The main advantages of RTX are its low cost, small code size and low and predictable response time to interrupts.

RTX Features	ARM7, ARM9 or Cortex-M3
Defined Tasks	Unlimited
Max Active Tasks	250
User Task Priorities	1 - 254
Scheduling	Round-robin, pre-emptive, co-operative
Mailboxes, Mutex, User Timers	Unlimited
Semaphores, Signals	Unlimited

Table 3: RTX Features

RTX is fully integrated in the Keil™ Microcontroller Development Kit (MDK). MDK includes RTX as a royalty-free library. The source code for RTX, a flash file system and protocol stacks for TCP/IP, USB and CAN are available in the Keil Real-Time Library (RL-ARM).

Real-time programs written using RTX require only including a special header file and linking the RTX library. Creating a task can be done by simply adding the __task keyword to a C function. All the options in RTX can be configured with a single C source file.

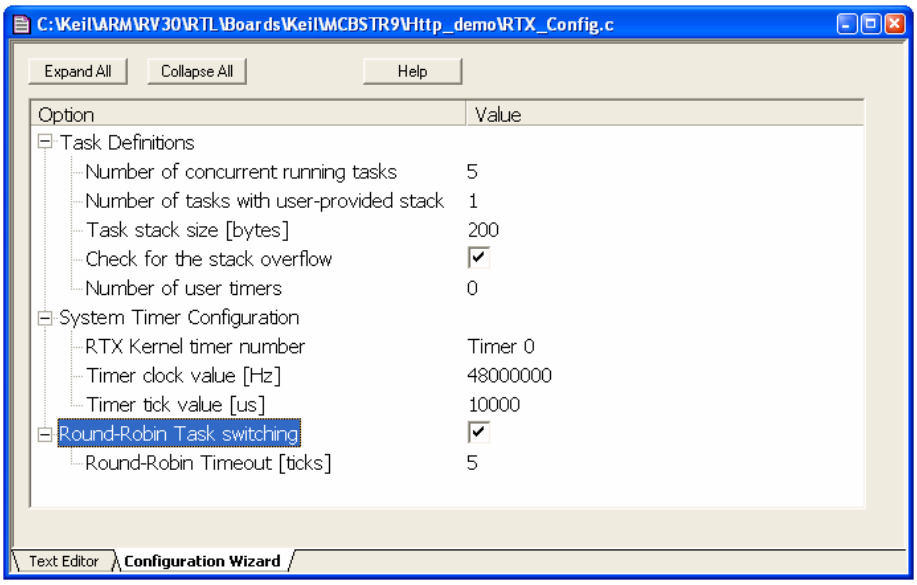

Figure 4: RTX Configuration

MDK also provides RTX debug awareness with dialogs that show the status of different tasks and how they are scheduled over time.

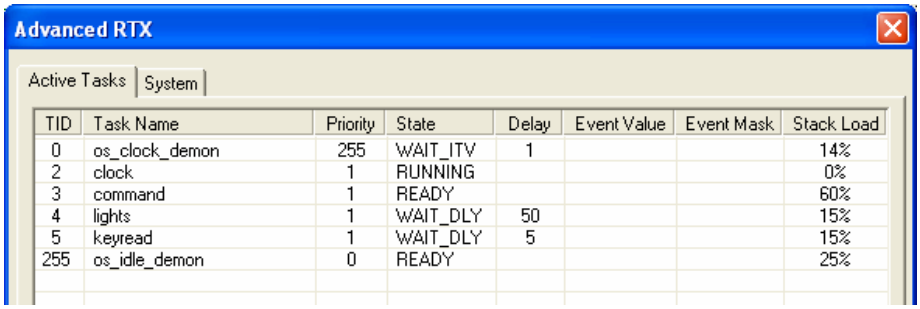

Figure 5: RTX awareness in Keil MDK

Porting RTX to Cortex-M3 Processor-based Microcontrollers

Rebuilding and running existing RTX C code on a Cortex-M3 processor-based microcontroller automatically brought a performance increase. Improved interrupt latency was achieved by modifying RTX to use the new Cortex-M3 processor programmer's model for exceptions.

Interrupts

On RTX for the ARM7TDMI processor a single re-entrant interrupt handler is shared for all the peripherals in the system. This allows high priority peripherals to interrupt the processor while it is handling interrupts from low priority peripherals. However, re-entrant interrupts require long sequences of code at the entry and exit of the interrupt handler, which is costly in terms of code size, performance and responsiveness.

The RTX interrupt handlers have been modified to use the new interrupt structure of Cortex-M3 processor-based microcontrollers. The RTX kernel only includes code for a single interrupt source, the system timer. When handling other interrupts the processor does not run any RTX code.

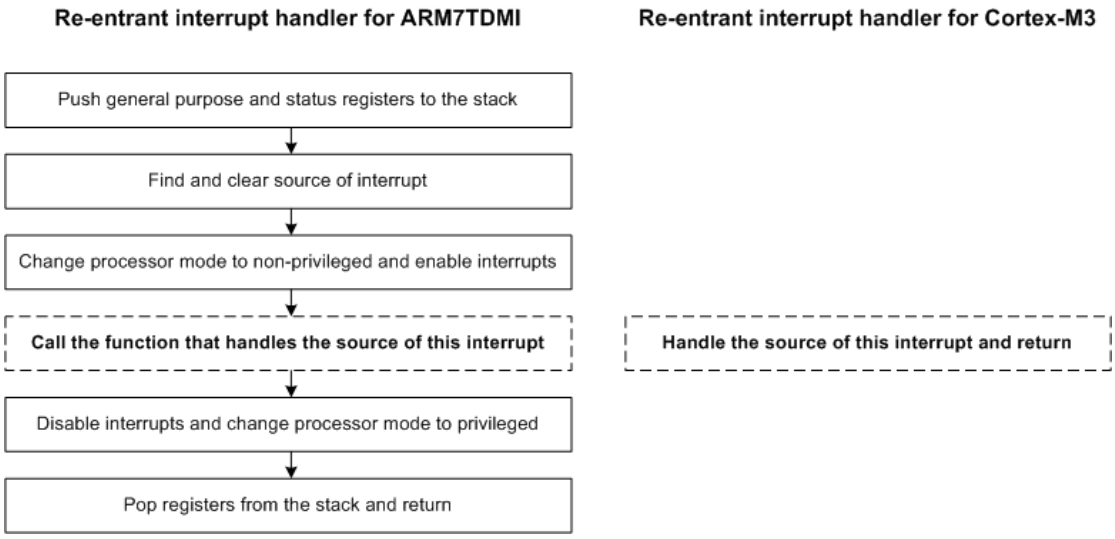

Figure 6: RTX interrupt handling on ARM7 and Cortex-M3 processors

System Calls

The RTX kernel always runs in privileged mode, as it needs access to resources that are only available in this mode. Therefore, kernel calls are implemented with instructions that cause an exception and put the processor in privileged mode.

On RTX for the ARM7TDMI processor, calls are implemented with SWI instructions, so while system calls are handled the processor cannot respond to interrupts. In order to keep low interrupt latency only a few short system calls can be implemented. As a result, some kernel functionality cannot run as system calls and a “demon” task is required to deal with it.

On RTX for the Cortex-M3 processor, all system calls are implemented with SVC instructions, so interrupts continue to be handled while the processor executes the SVC exception hander. RTX for Cortex-M3 processor is not only more responsive to external events, but also its code is better structured and easier to understand.

Comparison: RTX on ARM7TDMI and Cortex-M3 Processor-based Microcontrollers

The following tables compare RTX running on an ARM7TDMI processor-based LPC2138 microcontroller at 60MHz and a Cortex-M3 processor-based STM32F103RB microcontroller at 72MHz.

The first area of comparison is size. RTX for the Cortex-M3 processor is 0.5 Kbytes smaller and requires 20 bytes less RAM thanks to its simpler programmer's model for exceptions.

RTX Size (bytes)	ARM7TDMI	Cortex-M3
Code	<4.5 K (Thumb)	<4.0 K (Thumb-2)
RAM for Kernel	446	428
RAM for a Task	TaskStackSize + 52	TaskStackSize + 52
RAM for a Mailbox	MaxMessage * 4 + 16	MaxMessages * 4 + 16
RAM for a Semaphore	8	8
RAM for a Mutex	12	12
RAM for a User Timer	8	8

Table 4: RTX Memory Size for ARM7TDMI and Cortex-M3 processors

The second area of comparison is performance. The microcontroller clock frequency accounts for a 20 percent performance increase on the STM32F103RB microcontroller. The remaining performance increase up to two fold is mostly due to the processor's instruction set and implementation.

RTX Function	ARM7TDMI	Cortex-M3	Improvement
Initialize system, start task	46.2	22.1	2.1x
Create defined task	17.0	8.1	2.1x
Create defined task and switch task	19.1	9.3	2.1x
Delete task	9.3	4.8	1.9x
Task switch	6.6	3.9	1.7x
Set event	2.4	1.9	1.3x
Send message	4.5	2.5	1.8x
Context switch time	<7 µsec	<4 µsec	1.7x
Maximum interrupt latency	4.2	0.1	n/a

Table 5: RTX Performance on ARM7TDMI and Cortex-M3 processors

The maximum interrupt latency is high in the case of the ARM7TDMI processor-based microcontroller because it requires a re-entrant interrupt handler and because interrupts are disabled while the kernel is handling system calls. The interrupt latency is only up to 12 cycles for the Cortex-M3 processor-based microcontrollers.

In summary, the Cortex-M3 processor brings considerable advantages for developers of real-time applications. The Keil RTX Real-Time Kernel has been updated to make use of these advantages, which has resulted in faster performance, smaller code size and faster response to external events.

References

Further details on Cortex-M3 processor and other ARM products: http://www.arm.com.
Further details on RTX and other Keil products: http://www.keil.com.
Further details on the benchmark used in this article: http://www.keil.com/support/man/docs/rlarm/rlarm_ar_timing_spec.htm.