Cutting Through the Confusion with ARM Cortex-M Interrupt Priorities

Miro Samek●February 26, 2016

4 Tweet

- ARM
- Software Development
- Microcontroller

The insanely popular ARM Cortex-M processor offers very versatile interrupt priority management, but unfortunately, the multiple priority numbering conventions used in managing the interrupt priorities are often counter-intuitive, inconsistent, and confusing, which can lead to bugs. In this post I attempt to explain the subject and cut through the confusion.

The Inverse Relationship Between Priority Numbers and Urgency of the Interrupts

The most important fact to know is that ARM Cortex-M uses the “reversed” priority numbering scheme for interrupts, where priority zero corresponds to the highest urgency interrupt and higher numerical values of priority correspond to lower urgency. This numbering scheme poses a constant threat of confusion, because any use of the terms “higher priority” or “lower priority” immediately requires clarification, whether they represent the numerical value of priority, or perhaps, the urgency of an interrupt.

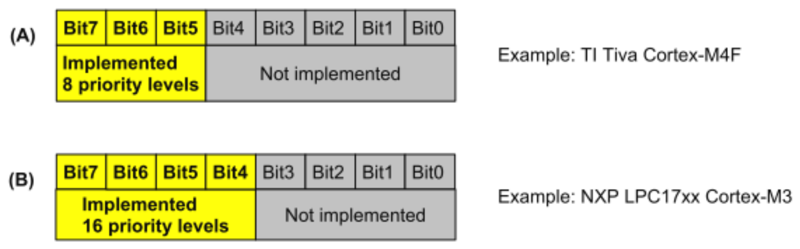
This article is available in PDF format for easy printing

NOTE: To avoid this confusion, in the rest of this post, the term “*priority*” means the numerical value of interrupt priority in the ARM Cortex-M convention. The term “*urgency*” means the capability of an interrupt to preempt other interrupts. A higher-urgency interrupt (lower priority number) can preempt a lower-urgency interrupt (higher priority number).

Interrupt Priority Configuration Registers in the NVIC

The number of priority levels in the ARM Cortex-M core is configurable, meaning that various silicon vendors can implement different number of priority bits in their chips. However, there is a minimum number of interrupt priority bits that need to be implemented, which is 2 bits in ARM Cortex-M0/M0+ and 3 bits in ARM Cortex-M3/M4.

But here again, the most confusing fact is that the priority bits are implemented in the **most-significant** bits of the priority configuration registers in the NVIC (Nested Vectored Interrupt Controller). The following figure illustrates the bit assignment in a priority configuration register for 3-bit implementation (part A), such as TI Tiva MCUs, and 4-bit implementation (part B), such as the NXP LPC17xx ARM Cortex-M3 MCUs.


(A)
Bit7 Bit6 Bit5 Bit4 Bit3 Bit2 Bit1 Bit0
Implemented 8 priority levels
Not implemented
Example: TI Tiva Cortex-M4F
(B)
Bit7 Bit6 Bit5 Bit4 Bit3 Bit2 Bit1 Bit0
Implemented 16 priority levels
Not implemented
Example: NXP LPC17xx Cortex-M3

Interrupt priory registers with 3 bits of priority (A), and 4 bits of priority (B)

The relevance of the bit representation in the NVIC priority register is that this creates **another priority numbering scheme**, in which the numerical value of the priority is shifted to the left by the number of unimplemented priority bits. If you ever write directly to the priority registers in the NVIC, you must remember to use this convention.

NOTE: The interrupt priorities don't need to be uniquely assigned, so it is perfectly legal to assign the same interrupt priority to many interrupts in the system. That means that your application can service many more interrupts than the number of interrupt priority levels.

NOTE: Out of reset, all interrupts and exceptions with configurable priority have the same default priority of zero. This priority number represents the highest-possible interrupt urgency.

Interrupt Priority Numbering in the CMSIS

The Cortex Microcontroller Software Interface Standard (**CMSIS**) provided by ARM Ltd. is the recommended way of programming Cortex-M microcontrollers in a portable way. The CMSIS standard provides the function **NVIC_SetPriority(IRQn, priority)** for setting the interrupts priorities.

However, it is very important to note that the '**priority**' argument of this function must **not** be shifted by the number of unimplemented bits, because the function performs the shifting by (8 - __NVIC_PRIO_BITS) internally, before writing the value to the appropriate priority configuration register in the NVIC. The number of implemented priority bits __NVIC_PRIO_BITS is defined in CMSIS for each ARM Cortex-M device.

For example, calling NVIC_SetPriority(7, 6) will set the priority configuration register corresponding to IRQ#7 to 1100,0000 binary on ARM Cortex-M with 3-bits of interrupt priority and it will set the same register to 0110,0000 binary on ARM Cortex-M with 4-bits of priority.

NOTE: The confusion about the priority numbering scheme used in the NVIC_SetPriority() is further promulgated by various code examples on the Internet and even in reputable books. For example the book "The Definitive Guide to ARM Cortex-M3, Second Edition", ISBN 979-0-12-382091-4, Section 8.3 on page 138 includes a call NVIC_SetPriority(7, 0xC0) with the intent to set priority of IR#7 to 6. This call is incorrect and at least in CMSIS version 3.x will set the priority of IR#7 to zero.

Preempt Priority and Subpriority

The interrupt priority registers for each interrupt is further divided into two parts. The upper part (most-significant bits) is the preempt priority, and the lower part (least-significant bits) is the subpriority. The number of bits in each part of the priority registers is configurable via the Application Interrupt and Reset Control Register (AIRC, at address 0xE000ED0C).

The **preempt priority** level defines whether an interrupt can be serviced when the processor is already running another interrupt handler. In other words, preempt priority determines if one interrupt can preempt another.

The **subpriority** level value is used only when two exceptions with the same preempt priority level are pending (because interrupts are disabled, for example). When the interrupts are re-enabled, the exception with the lower subpriority (higher urgency) will be handled first.

In most applications, I would highly recommended to assign all the interrupt priority bits to the preempt priority group, leaving no priority bits as subpriority bits, which is the default setting out of reset. Any other configuration complicates the otherwise direct relationship between the interrupt priority number and interrupt urgency.

NOTE: Some third-party code libraries (e.g., the STM32 driver library) change the priority grouping configuration to non-standard. Therefore, it is highly recommended to explicitly re-set the priority grouping to the default by calling the CMSIS function **NVIC_SetPriorityGrouping(0U)** after initializing such external libraries.

Disabling Interrupts with PRIMASK and BASEPRI Registers

Often in real-time embedded programming it is necessary to perform certain operations **atomically** to prevent data corruption. The simplest way to achieve the atomicity is to briefly disable and re-enabe interrupts.

The ARM Cortex-M offers two methods of disabling and re-enabling interrupts. The simplest method is to set and clear the interrupt bit in the PRIMASK register. Specifically, disabling interrupts can be achieved with the “CPSID i” instruction and enabling interrupts with the “CPSIE i” instruction. This method is simple and fast, but it disables all interrupt levels indiscriminately. This is the only method available in the ARMv6-M architecture (Cortex-M0/M0+).

However, the more advanced ARMv7-M (Cortex-M3/M4/M4F) provides additionally the BASEPRI special register, which allows you to disable interrupts more selectively. Specifically, you can disable interrupts only with urgency lower than a certain level and leave the higher-urgency interrupts not disabled at all. (This feature is sometimes called "zero interrupt latency".)

NOTE: The BASEPRI register cannot disable interrupt priority 0 (highest urgency). In fact, writing 0 into BASEPRI re-enables all interrupt priority levels.

The CMSIS provides the function **__set_BASEPRI(priority)** for changing the value of the BASEPRI register. The function uses the hardware convention for the 'priority' argument, which means that the priority must be shifted left by the number of unimplemented bits (8 - __NVIC_PRIO_BITS).

NOTE: The priority numbering convention used in **__set_BASEPRI(priority)** is thus **different** than in the NVIC_SetPriority(priority) function, which expects the “priority” argument not shifted.

For example, if you want to selectively block interrupts with priority number higher or equal to 6, you could use the following code:

```
// code before critical section
__set_BASEPRI(6U << (8 - __NVIC_PRIO_BITS));
// critical section
__set_BASEPRI(0U); // remove the BASEPRI masking
// code after critical section
```

Next post by Miro Samek:
Beyond the RTOS: A Better Way to Design Real-Time Embedded Software

You might also like... (promoted content)

Make IP protocols available on any Embedded USB device

Free Webinar: Verifying RTOS Applications using Deep Insight Analysis

Comments:

- Comments
- Write a Comment
  Select to add a comment

To post reply to a comment, click on the 'reply' button attached to each comment. To post a new comment (not a reply to a comment) check out the 'Write a Comment' tab at the top of the comments.

Sign in

Username | Password | Sign in

Remember me

Forgot username or password? | Create account

You might also like...

Subscribe to this Blog

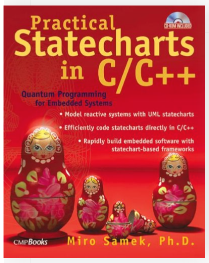
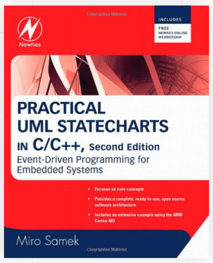
Receive a notification when **Miro Samek** publishes a new article:

Your Email Subscribe

About Miro Samek

Dr. Miro M. Samek is the creator of the open source QP active object frameworks and the free QM graphical modeling tool. He is also the founder of Quantum Leaps (state-machine.com), an open source company dedicated to bringing quantum leaps of innovation to embedded systems programming by making software and tools that enable widespread adoption of event-driven active object frameworks, hierarchical state machines (UML statecharts), design by contract, rapid prototyping, modeling, and automatic code generation.

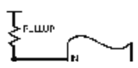
Books by Miro Samek

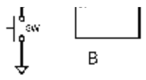



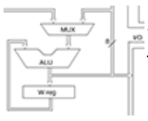
Popular Posts by Miro Samek

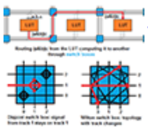
- Embedded Toolbox: Source Code Whitespace Cleanup
- Embedded Toolbox: Programmer's Calculator
- Embedded Toolbox: Windows GUI Prototyping Toolkit
- Beyond the RTOS: A Better Way to Design Real-Time Embedded Software

Blogs - Hall of Fame

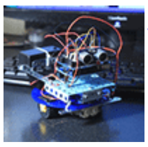

Introduction to Microcontrollers
Mike Silva




Important Programming Concepts (Even on Embedded Systems)
Jason Sachs


How FPGAs Work and Why You'll Buy One
Yossi Krenin

MSP430 Launchpad Tutorial
Enrico Garante


Arduino Robotics
Lonnie Honeycutt

Free PDF Downloads

Real-time Image Processing on Low Cost Embedded Computers

CPU Memory - What Every Programmer Should Know About Memory

Getting Started with C Programming for the ATMEL AVR Microcontrollers

All FREE PDF Downloads

Quick Links

- Home
- Blogs
- Tweets
- Forums
- Jobs
- #IoT
- Tutorials
- Books
- Free PDFs
- Vendors
- Code Snippets
- comp.arch.embedded

About EmbeddedRelated.com

- Advertise
- Contact

Social Networks

EmbeddedRelated.com DSPRelated.com Electronics-Related.com
FPGARelated.com