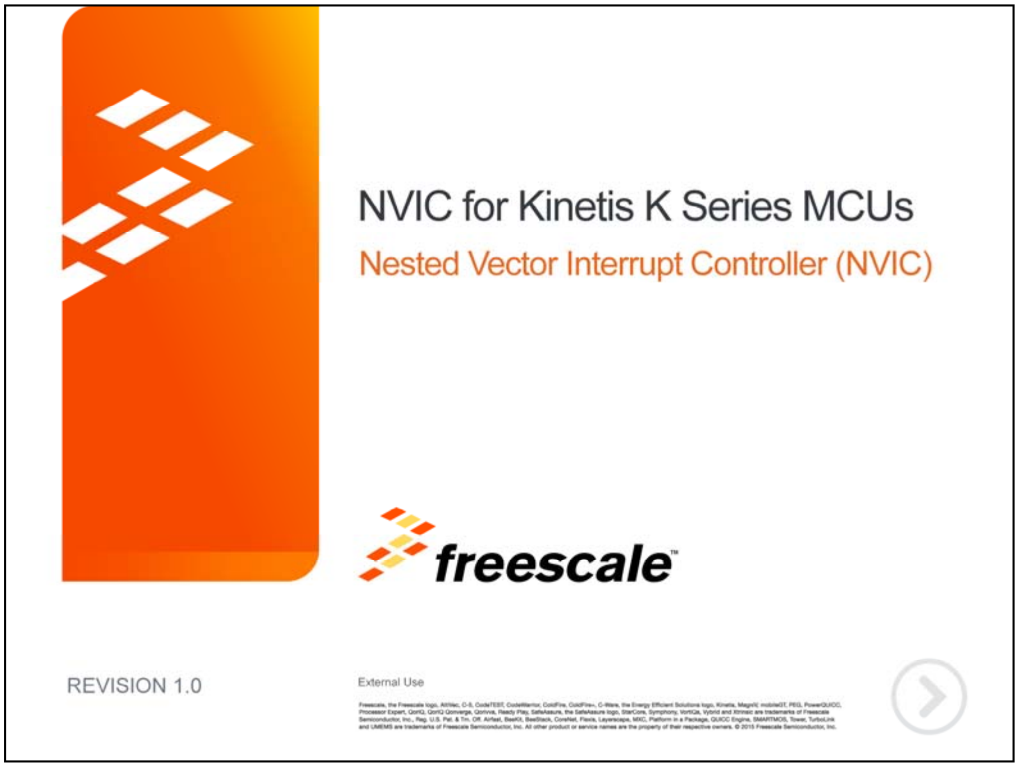


NVIC for Kinetis K Series MCUs

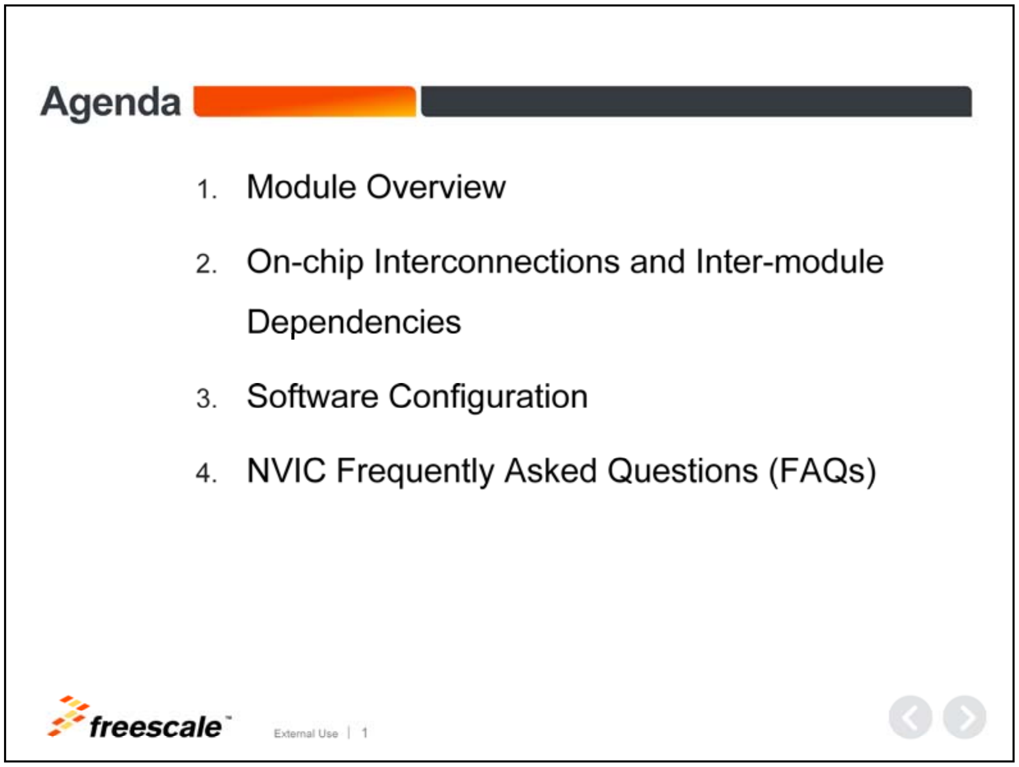
Nested Vector Interrupt Controller (NVIC)

freescale

REVISION 1.0

External Use

Hello, and welcome to this presentation of the Nested Vector Interrupt Controller – or NVIC – module for Kinetis K series MCUs. In this session, you'll learn about the NVIC, its main features and the application benefits of leveraging this function.

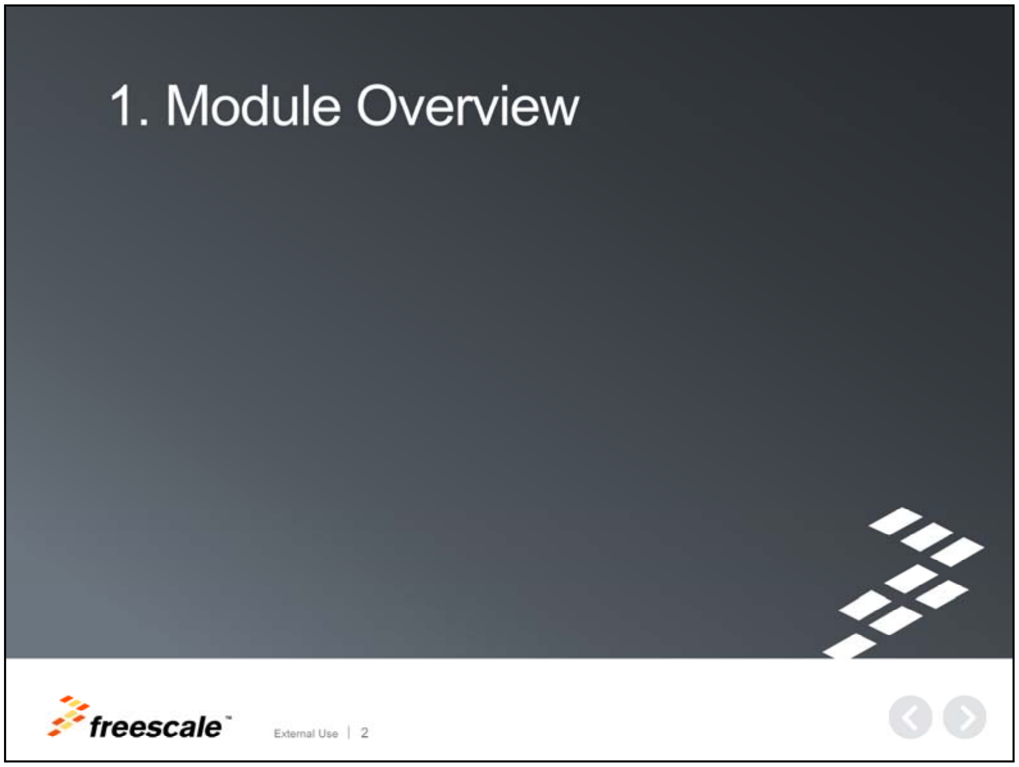


Agenda

1. Module Overview
2. On-chip Interconnections and Inter-module Dependencies
3. Software Configuration
4. NVIC Frequently Asked Questions (FAQs)

In this presentation, we'll cover:

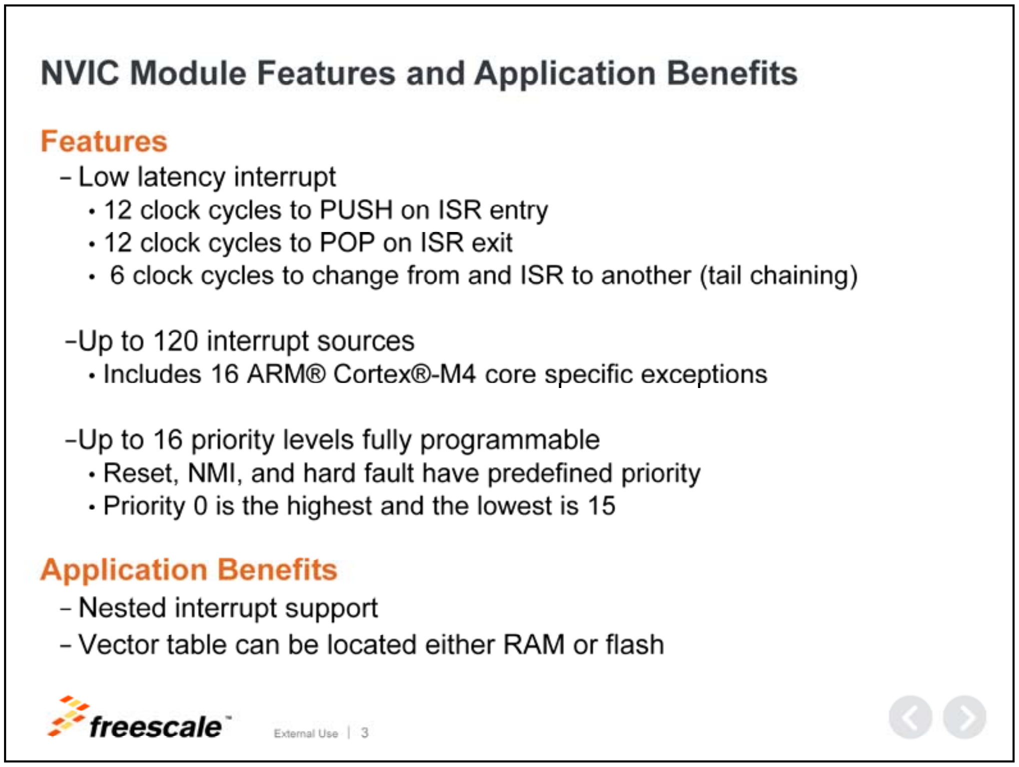
• An overview of the NVIC module itself
• The on-chip interconnections and inter-module dependencies
• Software configuration
• And some frequently asked questions



1. Module Overview

External Use | 2

Let’s first begin with an overview of the module.



NVIC Module Features and Application Benefits

Features

- Low latency interrupt
  - 12 clock cycles to PUSH on ISR entry
  - 12 clock cycles to POP on ISR exit
  - 6 clock cycles to change from and ISR to another (tail chaining)
- Up to 120 interrupt sources
  - Includes 16 ARM® Cortex®-M4 core specific exceptions
- Up to 16 priority levels fully programmable
  - Reset, NMI, and hard fault have predefined priority
  - Priority 0 is the highest and the lowest is 15

Application Benefits

- Nested interrupt support
- Vector table can be located either RAM or flash

freescale External Use | 3

NVIC Module Features and Application Benefits

Features

•The NVIC module is located within the ARM® Cortex®-M4 core and provides low latency interrupt servicing by taking only 12 clock cycles to start or exit the interrupt service routine, or ISR. In case there are two pending interrupts, it will take 6 clock cycles

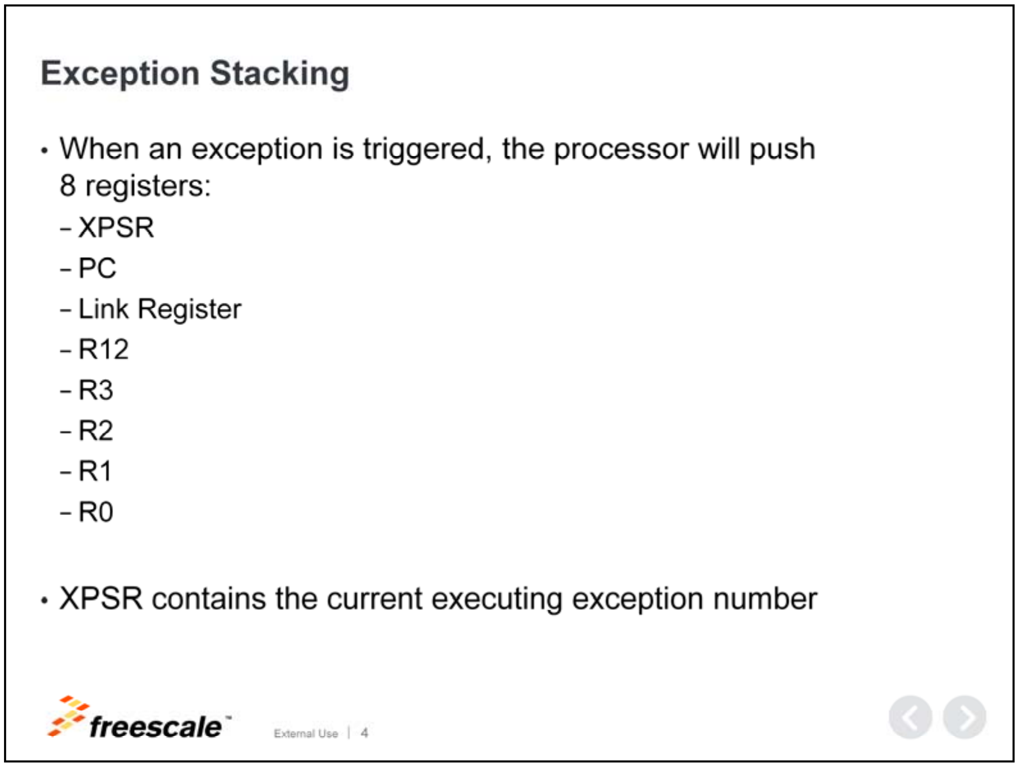
• There are up to 120 interrupt sources on the NVIC implementation for Kinetis devices. The first 16 interrupt sources are dedicated to the ARM Cortex-M4 core

• The NVIC module supports up to 16 interrupt priority levels for peripherals. However, the priority level for the ARM Cortex-M4 core exceptions are fixed

Application benefits, include:

• Automatic nested interrupt support is provided for embedded systems, so low priority interrupt requests are delayed when high priority requests are pending

• The vector table can be relocated from Flash to RAM for applications such as bootloaders



Exception Stacking

- When an exception is triggered, the processor will push 8 registers:
  - XPSR
  - PC
  - Link Register
  - R12
  - R3
  - R2
  - R1
  - R0
- XPSR contains the current executing exception number

External Use | 4

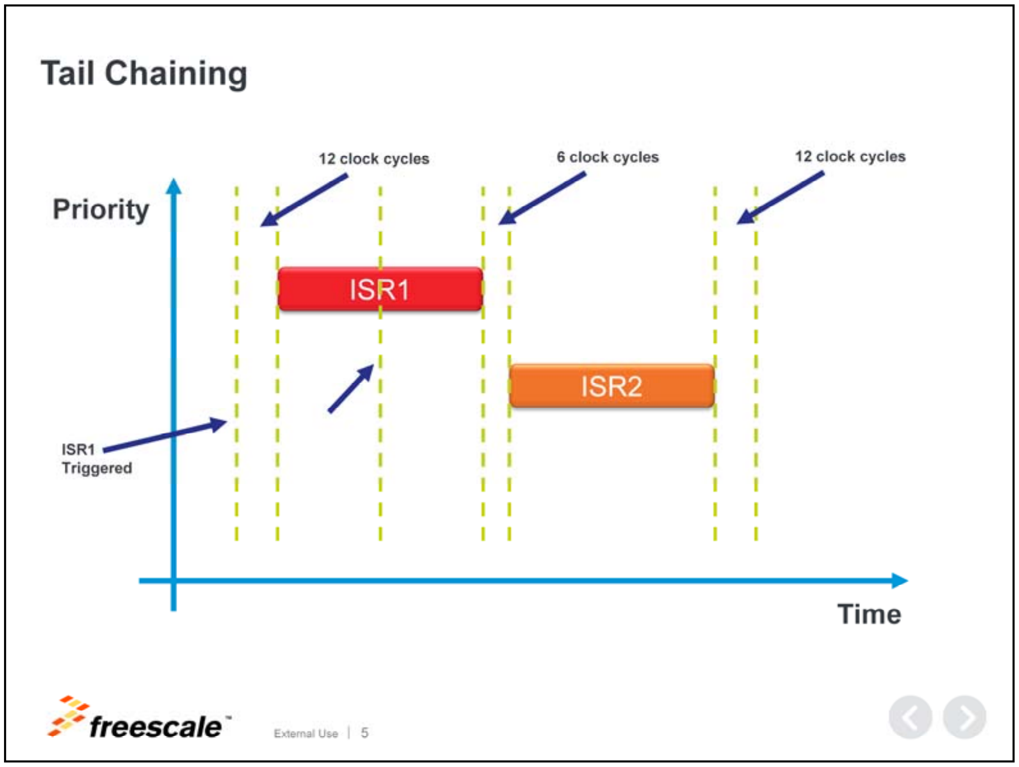
Exception Stacking

Exceptions are the interrupts that come from the core. Kinetis K MCUs are based on ARM® Cortex®-M4 cores.

When an exception is triggered, the ARM Cortex-M4 processor will start the exception process where 8 registers are pushed onto the stack before fetching the ISR address.

The XPSR is the processor status register and it contains the ALU flags, execution status, and the currently executing interrupt number.

You may be asking, why these 8 registers? The answer is: These registers are the ones used by the compiler to pass parameters to C functions. Because of this, the ISR handlers are common C functions, and no compiler directive or special instructions are necessary, such as RTI or RTE.

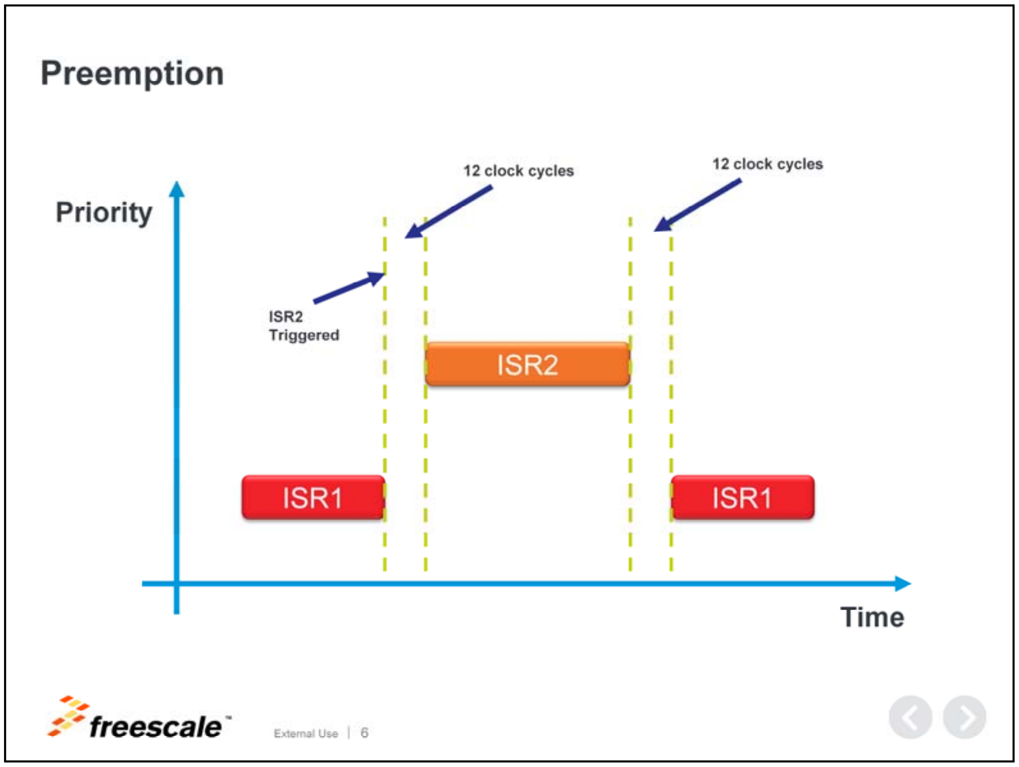


Tail Chaining

One of the features that the NVIC module provides is tail chaining. This helps reduce ISR execution time by avoiding register stacking when two or more interrupts are pending.

Now, let's assume that we have an application with two interrupts enabled:

- The higher priority interrupt is triggered and it will take 12 clock cycles to start executing the ISR. During these 12 clock cycles, the CPU will save the 8 registers discussed earlier and fetch the ISR address.
- During ISR1 execution, a lower or same priority interrupt is pending.
- When the ISR1 execution is complete, the ISR2 execution will start. Changing from ISR1 to ISR2 will only take 6 clock cycles since there is no register stacking.
- Once ISR2 is complete, the exit process will recover the 8 registers saved during entry. This takes 12 clock cycles.


Preemption

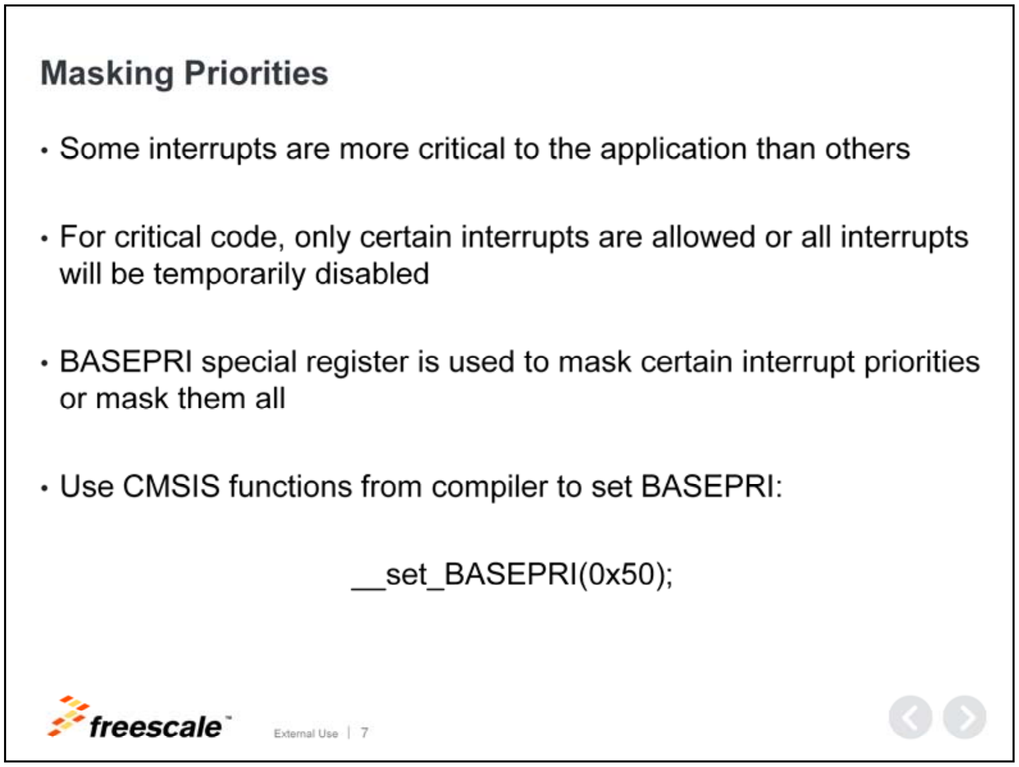
External Use | 6

Preemption

Preemption allows a higher priority ISR to execute by interrupting a lower priority ISR.

Let’s consider a case where there is a low priority ISR being executed when a higher priority interrupt is triggered:

- At this point, the CPU will stop executing ISR1 and go to ISR2. However, please note that in order to return to ISR1, the registers must be saved, hence, it takes 12 clock cycles to start ISR2.
- When ISR2 is complete, ISR1 resumes execution.



Masking Priorities

- Some interrupts are more critical to the application than others
- For critical code, only certain interrupts are allowed or all interrupts will be temporarily disabled
- BASEPRI special register is used to mask certain interrupt priorities or mask them all
- Use CMSIS functions from compiler to set BASEPRI:

```
__set_BASEPRI(0x50);
```

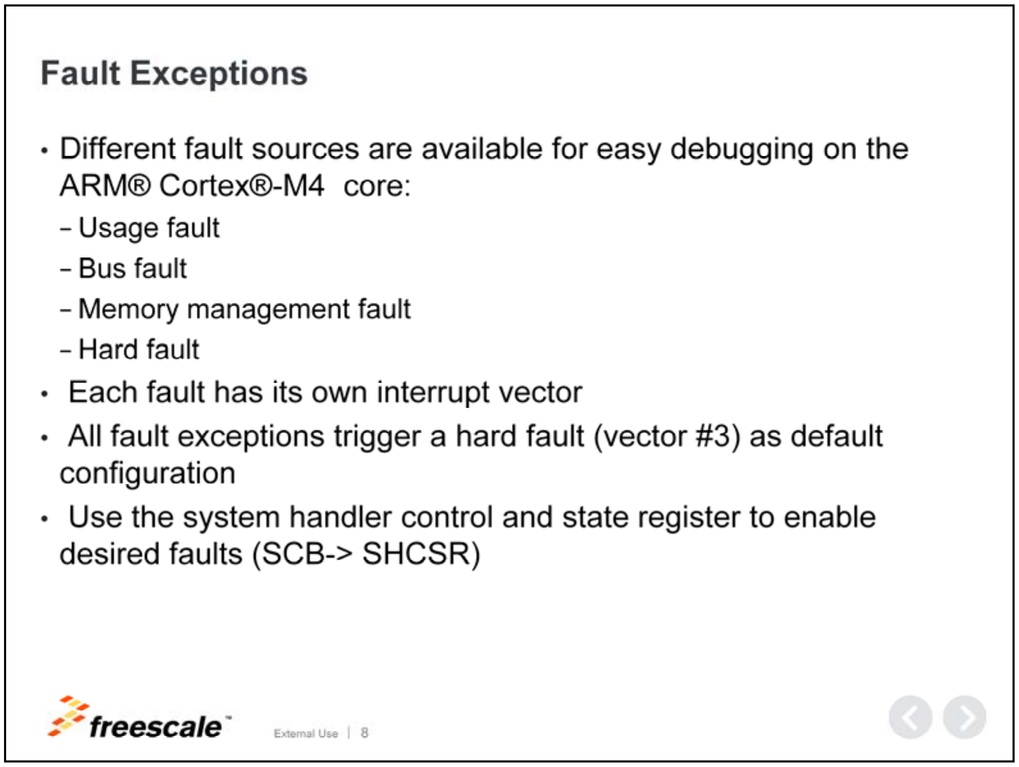
External Use | 7

Masking Priorities

There are some applications in which critical code must be either atomic or interrupted only by the highest priority interrupts in the system.

In order to achieve this, the NVIC provides a mechanism to mask interrupts with lower priorities by using the base-priority (BASEPRI) special register.

This is accomplished through CMSIS function set base-priority (__set_BASEPRI). The __set_BASEPRI function is used to set the masking level. Calling this function with a value of 5 means that only interrupts with a level of 0 to 4 are allowed. Please note that the priority bits are implemented in the most-significant nibble of the register.


Fault Exceptions

- Different fault sources are available for easy debugging on the ARM® Cortex®-M4 core:
  - Usage fault
  - Bus fault
  - Memory management fault
  - Hard fault
- Each fault has its own interrupt vector
- All fault exceptions trigger a hard fault (vector #3) as default configuration
- Use the system handler control and state register to enable desired faults (SCB-> SHCSR)

External Use | 8

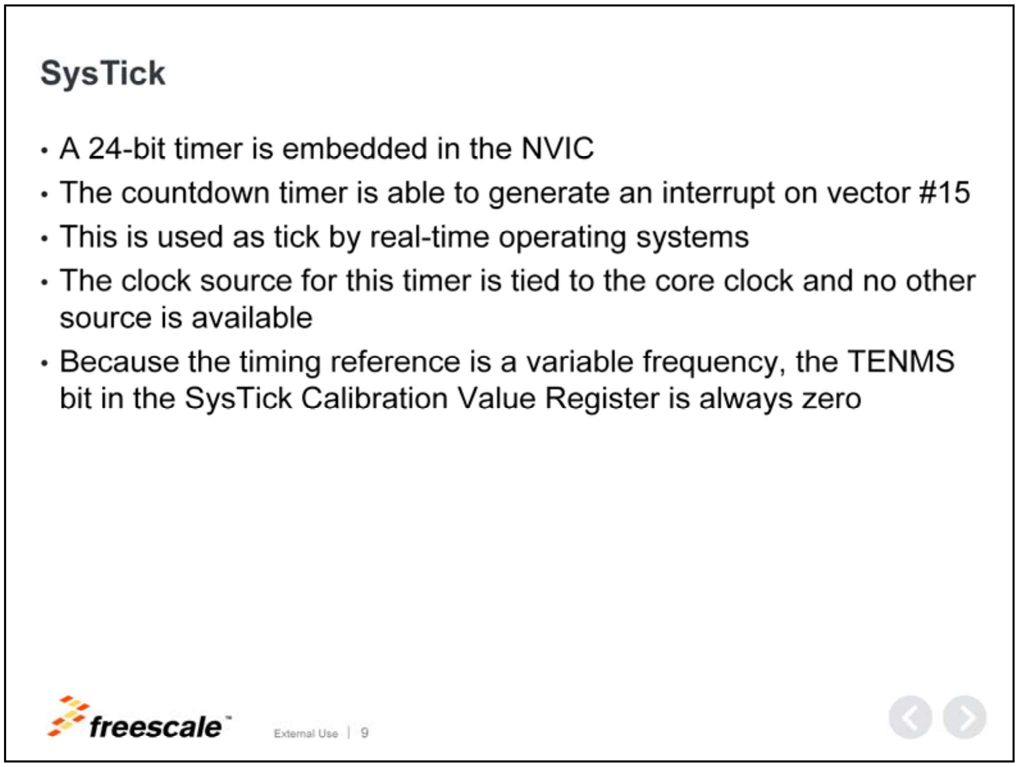
Fault Exceptions

Within the first 16 interrupt sources, there are four fault exceptions that are useful for debugging issues.

The fault sources within the ARM® Cortex®-M4 core are:

- Usage fault
- Bus fault
- Memory management fault
- Hard fault

Each of the fault exceptions has its own interrupt vector, allowing you to choose the cause of the fault in your system. Only the hard fault exception is enabled out of reset, and is triggered for usage, bus, or memory management faults until those exceptions are enabled explicitly in the system control block register.



SysTick

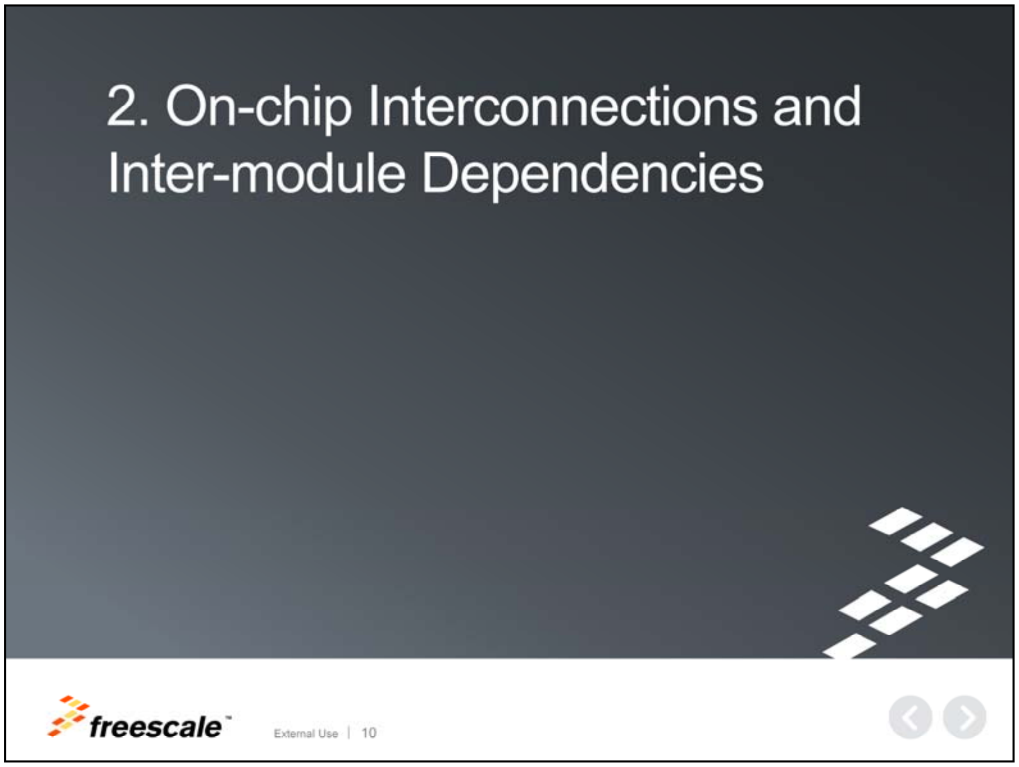
- A 24-bit timer is embedded in the NVIC
- The countdown timer is able to generate an interrupt on vector #15
- This is used as tick by real-time operating systems
- The clock source for this timer is tied to the core clock and no other source is available
- Because the timing reference is a variable frequency, the TENMS bit in the SysTick Calibration Value Register is always zero

freescale External Use | 9

SysTick

The NVIC module provides a 24-bit timer called SysTick, which can be used as a time base for the real-time operating systems , or as a system time base for periodic tasks in a bare metal environment.

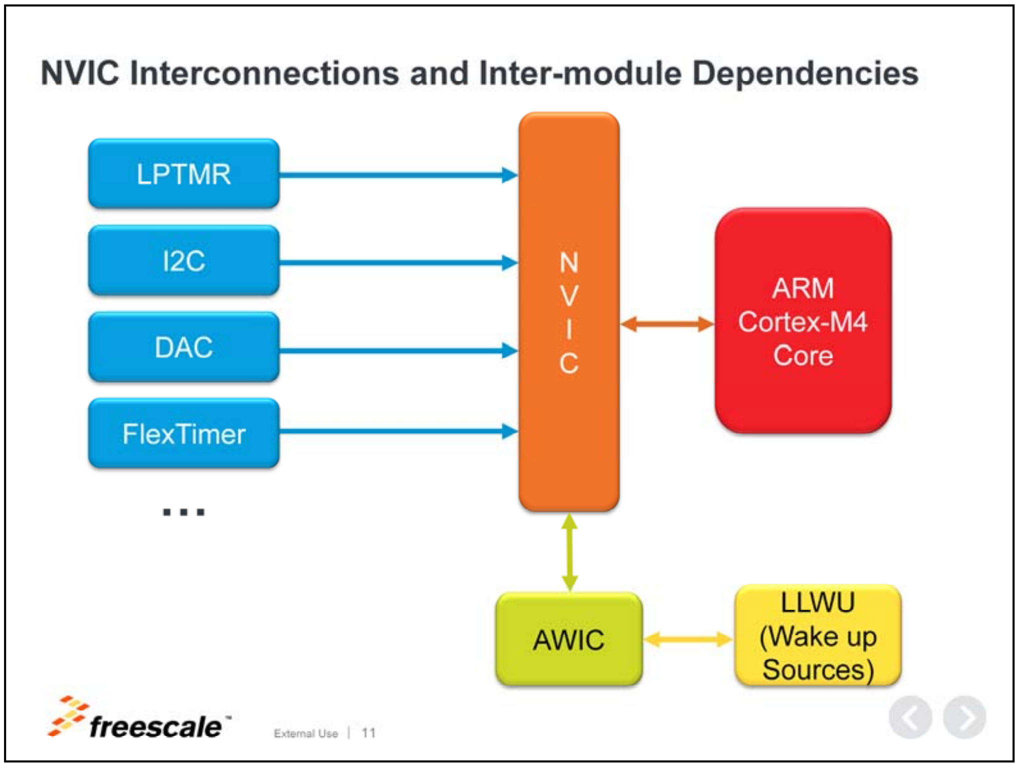
On Kinetis MCUs, the calibration feature of SysTick is not available because the timer clock source is fixed to use the processor core clock. There is no option for an external reference.



2. On-chip Interconnections and Inter-module Dependencies

External Use | 10

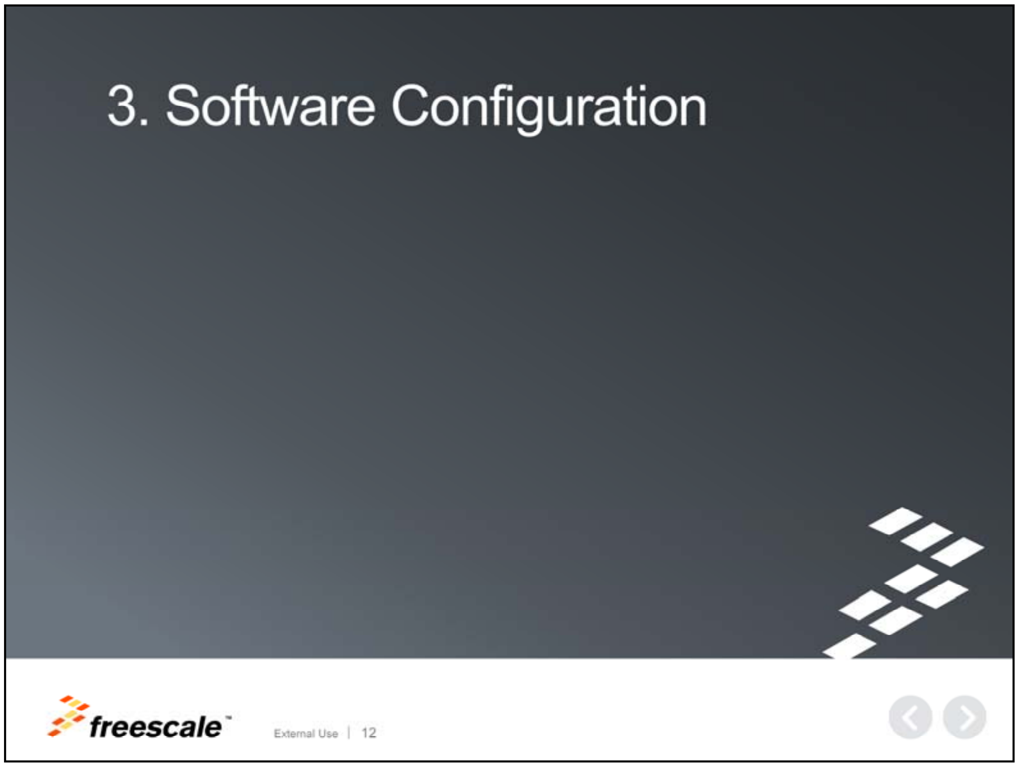
Now, let’s talk about on-chip interconnection and inter-module dependencies.



NVIC Interconnections and Inter-module Dependencies

NVIC Interconnections and Inter-module Dependencies

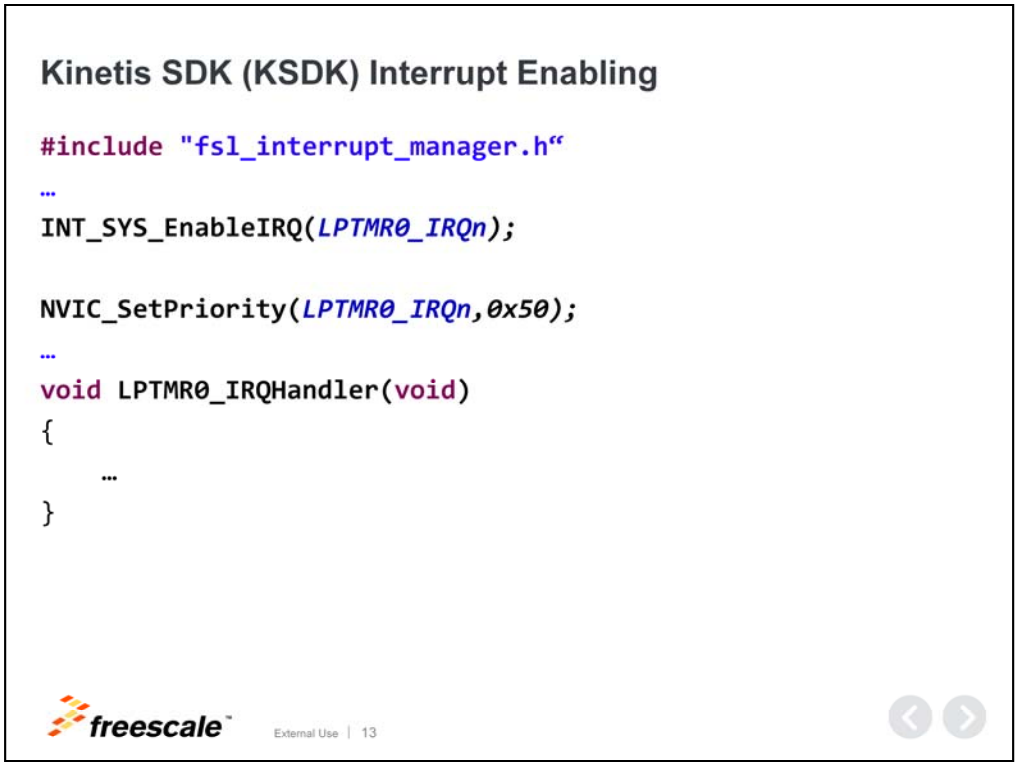
- The NVIC module provides methods to generate interrupts to the ARM® Cortex®-M4 core.
- All peripherals capable of generating an interrupt are also connected to the NVIC.
- The asynchronous wakeup interrupt controller, or AWIC, is used when the MCU goes into certain sleep modes. Since NVIC will remain static, the AWIC will act as the interrupt controller in those modes via the low-leakage wakeup unit module, abbreviated LLWU, which controls wake up sources.



3. Software Configuration

External Use | 12

Now, software configuration.


Kinetis SDK (KSDK) Interrupt Enabling

```
#include "fsl_interrupt_manager.h“
…
INT_SYS_EnableIRQ(LPTMR0_IRQn);

NVIC_SetPriority(LPTMR0_IRQn,0x50);
…
void LPTMR0_IRQHandler(void)
{
    …
}
```

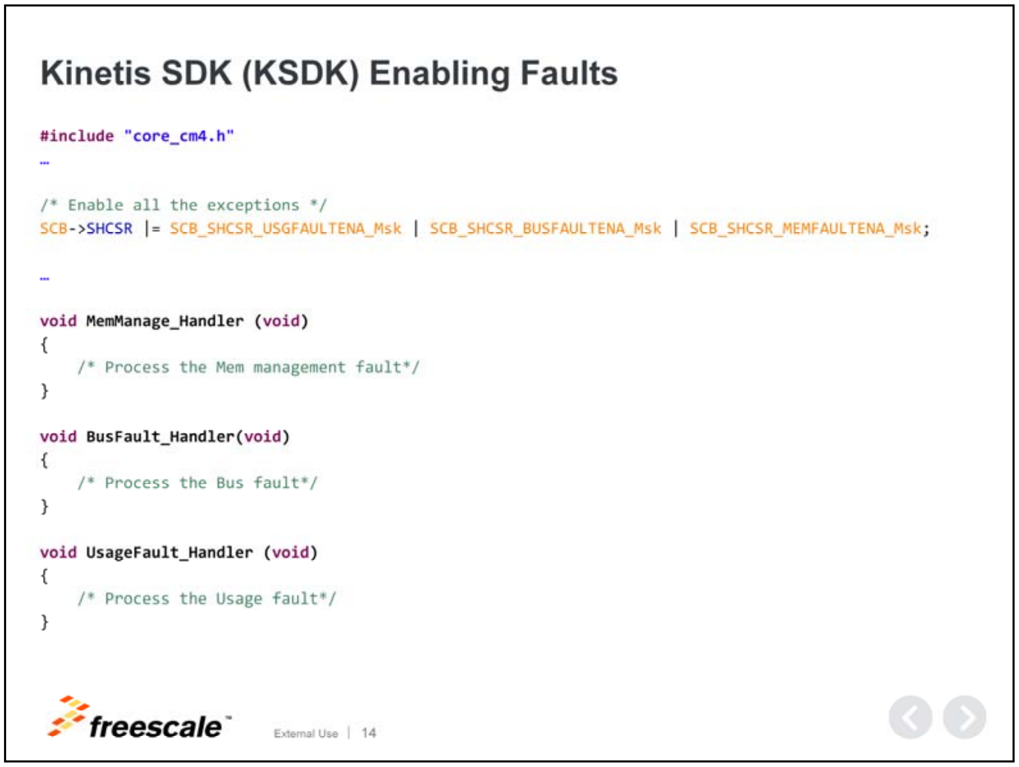
freescale External Use | 13

Kinetis SDK (KSDK) Interrupt Enabling

The Kinetis SDK provides peripheral drivers that implement interrupt handling. It also provides an API to enable interrupts for the NVIC.

In the following example, we enable a low-power interrupt, set the interrupt priority, and create an interrupt handler placeholder:

- First, you'll need to include the Kinetis SDK interrupt manager header file since this provides necessary APIs.
- The enable IRQ API passes in the peripheral IRQ number via a parameter. For ease of use, Kinetis SDK provides an interface to the IRQ number that is ARM® Cortex®-M4 core CMSIS compliant.
- Configuration of the interrupt priority is left to the application and the ARM® Cortex®-M4 core CMSIS API must be used. In this example, the timer is set to priority 5.
- Finally, implement the ISR handler with the required processing and application specific tasks.


Kinetis SDK (KSDK) Enabling Faults

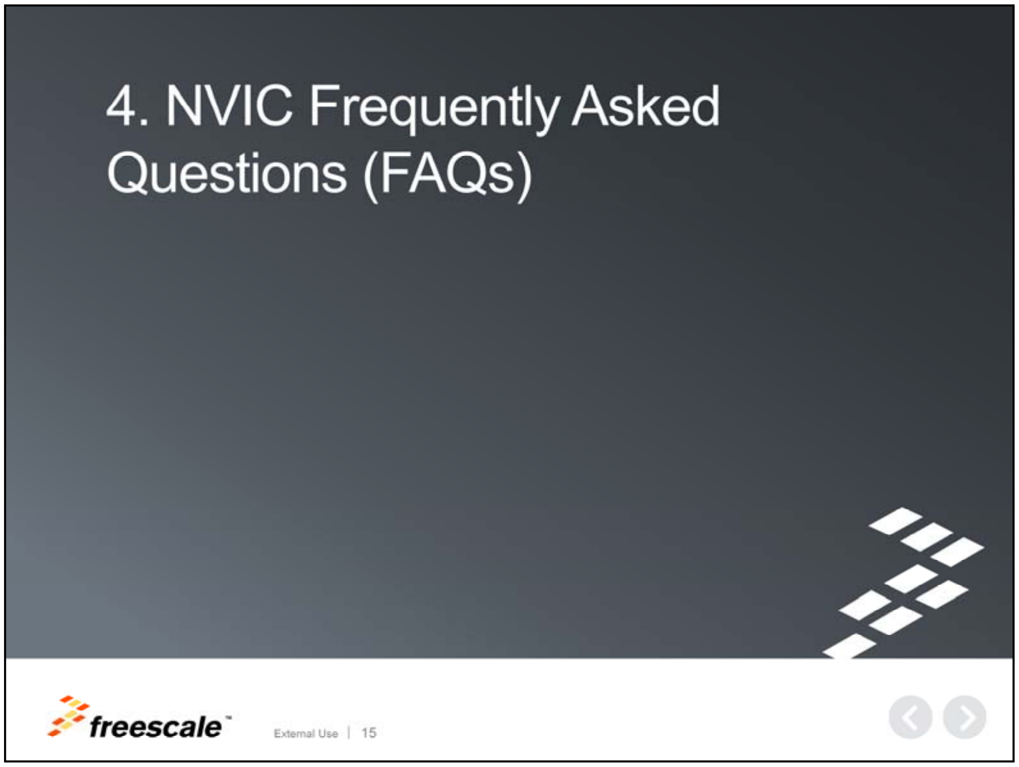
```
#include "core_cm4.h"
…

/* Enable all the exceptions */
SCB->SHCSR |= SCB_SHCSR_USGFAULTENA_Msk | SCB_SHCSR_BUSFAULTENA_Msk | SCB_SHCSR_MEMFAULTENA_Msk;

…

void MemManage_Handler (void)
{
    /* Process the Mem management fault*/
}

void BusFault_Handler(void)
{
    /* Process the Bus fault*/
}

void UsageFault_Handler (void)
{
    /* Process the Usage fault*/
}
```

External Use | 14

Kinetis SDK (KSDK) Enabling Faults

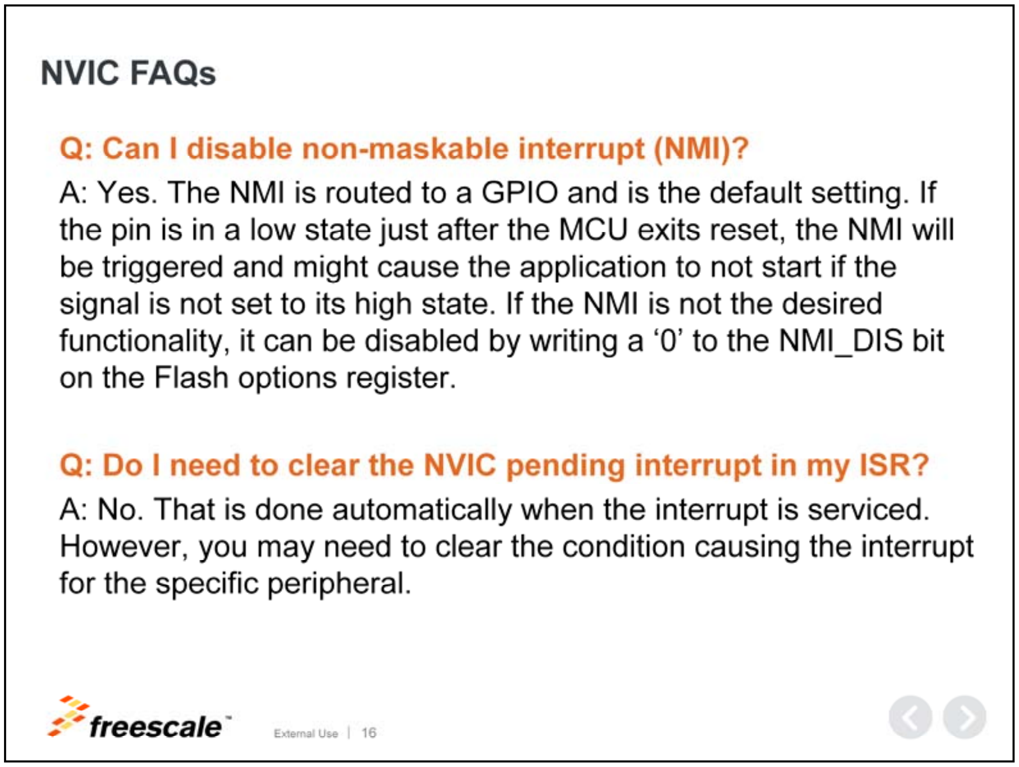
Now here's an example on how to enable all the faults:

- The Kinetis SDK does not provide an interface to the system control block, or the SCB, which means that the registers must be written by the application. So first, enable the required faults. In this example, all faults are enabled.
- The fault handlers are not implemented because these are application specific. The application must implement them and add the desired processing based on its needs.



4. NVIC Frequently Asked Questions (FAQs)

Finally, some frequently asked questions.



NVIC FAQs

Q: Can I disable non-maskable interrupt (NMI)?

A: Yes. The NMI is routed to a GPIO and is the default setting. If the pin is in a low state just after the MCU exits reset, the NMI will be triggered and might cause the application to not start if the signal is not set to its high state. If the NMI is not the desired functionality, it can be disabled by writing a ‘0’ to the NMI_DIS bit on the Flash options register.

Q: Do I need to clear the NVIC pending interrupt in my ISR?

A: No. That is done automatically when the interrupt is serviced. However, you may need to clear the condition causing the interrupt for the specific peripheral.

freescale External Use | 16

NVIC FAQs

Q: Can I disable non-maskable interrupt?

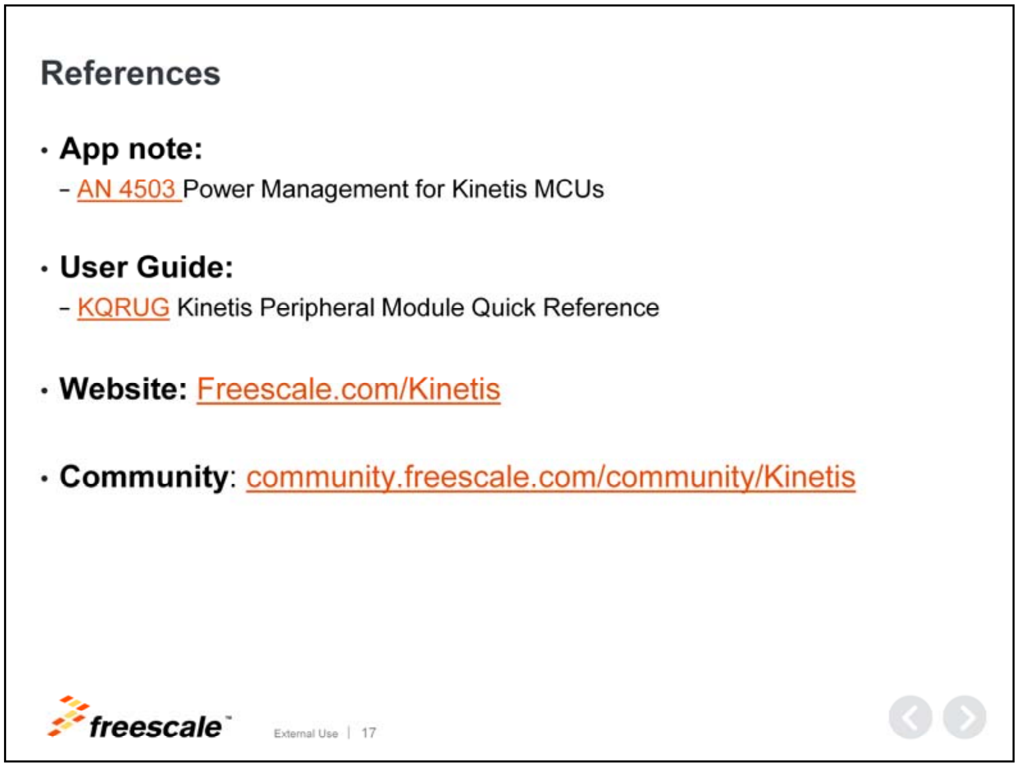
A: Yes, the NMI signal is multiplexed on a GPIO and is the default function for that pin.

NMI is active low and is level sensitive with priority -2. This means the only exception capable of interrupting it is a system reset. The NMI signal is available after CPU reset, so if the pin is connected to a signal that is on logic ‘0’, the NMI ISR will be triggered continuously until the pin is logic ‘1’, which can cause a system lockup.

NMI functionality can be disabled by clearing the NMI_disable bit in the Flash options register.

Q: Do I need to clear the NVIC pending interrupt in my ISR?

A: No, the pending interrupt in the NVIC is cleared automatically when the interrupt is serviced. However, you may need to clear the condition causing the interrupt for the specific peripheral.



References

- **App note:**
  - AN 4503 Power Management for Kinetis MCUs
- **User Guide:**
  - KQRUG Kinetis Peripheral Module Quick Reference
- **Website:** Freescale.com/Kinetis
- **Community**: community.freescale.com/community/Kinetis

freescale External Use | 17

References

The application note listed here outlines the role of the NVIC with Kinetis MCU Power Management. Chapter 3 of the Kinetis Quick Reference Guide provides a more in depth look at the NVIC module.

For more information, we invite you to also visit us on the web at Freescale.com/Kinetis and check out our community page.

www.Freescale.com