EFM32®

... the world's most energy friendly microcontrollers

EFM32 Interrupt Handling

AN0039 - Application Note

Introduction

This application note is an introduction to interrupts and wake-up handling in the EFM32. It includes ways to optimize for low latency wake-up, interrupt prioritization and energy saving operation.

This application note includes:

- This PDF document
- Source files (zip)
  - Example C code
  - Multiple IDE projects

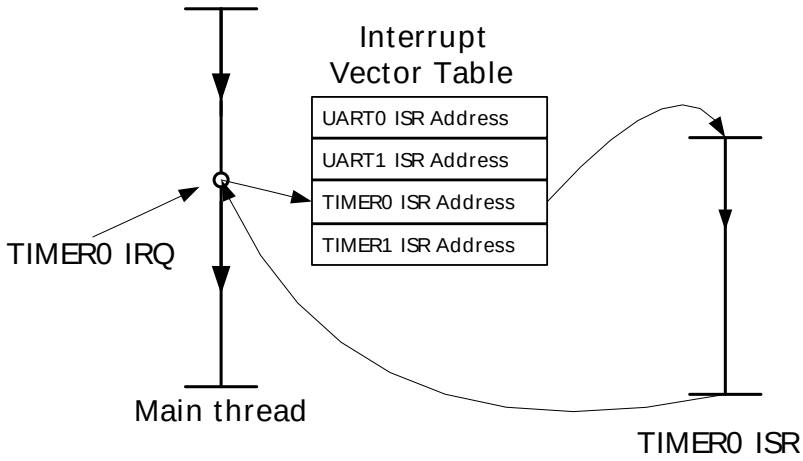
1 Interrupt Theory

Interrupts are a commonly used technique in microcontrollers allowing CPU-external systems to indicate need for change in CPU execution. Instead of using polling loops to monitor these kinds of events and wasting valuable processing time, interrupts do not require any action from the CPU unless they are triggered.

When an Interrupt Request (IRQ) is received, the CPU will store its current state before it switches to the Interrupt Service Routine (ISR) (Figure 1.1 (p. 2)). In older architectures there was only one ISR and SW needed to determine which source triggered the IRQ. In modern architectures like the ARM Cortex-M in the EFM32, each IRQ has its own ISR. The starting address for each ISR is stored in an interrupt vector table.

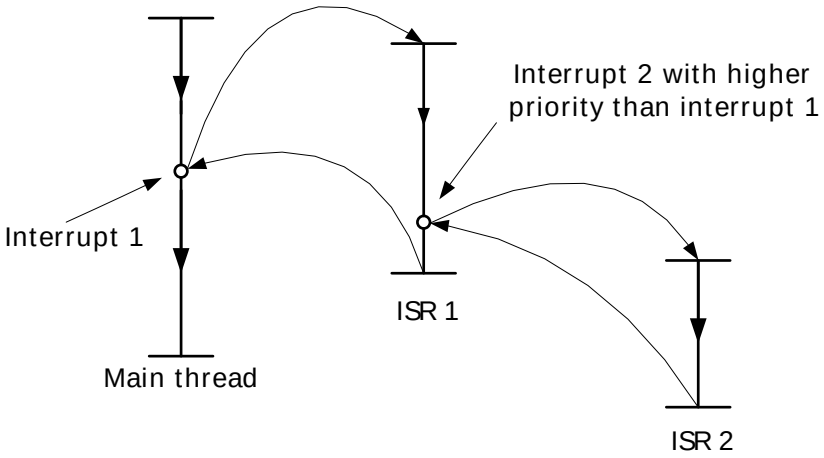
When an interrupt is triggered, the CPU automatically jumps to the corresponding address in the vector table which contains an address to the relevant ISR. The service routine then executes the tasks needed to handle the event before the CPU returns to where it left off before the interrupt was received. A common way to acknowledge the interrupt is to clear the interrupt source in the interrupt handler resulting in the IRQ being de-asserted.

Figure 1.1. Basic Interrupt Operation



As more than one interrupt can be triggered at the same time, interrupt priorities can be assigned to the different IRQs. This allows lower latency for the most important interrupts for real-time control etc. In interrupt controllers that support nesting, it is also possible for a high priority interrupt handler to start immediately even if another lower priority handler is executing. The CPU will then continue where it left off in the lower priority handler once it is done servicing the higher priority interrupt. Figure 1.2 (p. 3) shows an example where a higher priority interrupt (2) is serviced in the middle of the lower priority interrupt handler (1). The lookup in the vector table is done every time an IRQ is triggered, but left out in the figure for simplicity.

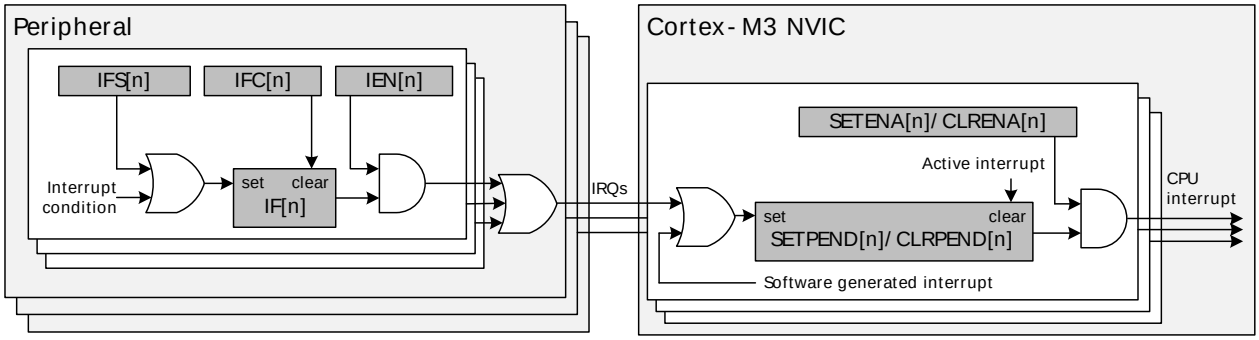
Figure 1.2. Nested Interrupts



2 Interrupts in the EFM32

The Nested Vector Interrupt Controller (NVIC) in the ARM Cortex-M processor in the EFM32 evaluates the IRQ lines and switches the CPU execution to the triggered IRQs address in the vector table. Figure 2.1 (p. 4) shows an overview of how interrupts are handled in the EFM32. Most of the peripherals in the EFM32 can generate interrupts and control one or more interrupt lines (IRQ) each.

Figure 2.1. Interrupt overview



2.1 Peripheral IRQ Generation

As shown in Figure 2.1 (p. 4) each IRQ line can be triggered by one or more interrupt flags (IF). Normally these interrupt flags will be set by a hardware condition (e.g. timer overflow), but SW can also set and clear these directly by writing to the IFS (Interrupt Flag Set register) or IFC (Interrupt Flag Clear register). The Interrupt Enable (IEN) register allows masking of interrupt flags that should not trigger the IRQ. To acknowledge an interrupt the Interrupt Flag corresponding to the event should be cleared (through the IFC register) in the ISR. The OR function between the interrupt flags ensures that the IRQ line is asserted as long as there are unmasked interrupt flags that have not been acknowledged. As an example, the TIMER0 peripheral contains 8 interrupt flags:

Example 2.1. TIMER0 interrupt flags/conditions

- OF - Overflow
- UF - Underflow
- CC0 - Compare Match/Input Capture on Channel 0
- CC1 - Compare Match/Input Capture on Channel 1
- CC2 - Compare Match/Input Capture on Channel 2
- ICBOF0 - Input Capture Buffer Overflow on Channel 0
- ICBOF1 - Input Capture Buffer Overflow on Channel 1
- ICBOF2 - Input Capture Buffer Overflow on Channel 2

These interrupt flags can be read through TIMER0_IF. The TIMER0_IFC and TIMER0_IFS registers are used to clear and set the interrupt flags, while TIMER0_IEN masks the flags not contributing to the IRQ. More detailed information on how to generate IRQs in the EFM32 peripherals are given in the reference manual for the device, as well as in practical examples in application notes targeted for the specific peripheral.

2.2 The Nested Vector Interrupt Controller (NVIC)

The Nested Vector Interrupt Controller (NVIC) is an integrated part of the ARM Cortex-M processor, supporting both Cortex-internal interrupts (Hard fault, SysTick etc.) and up to 240 peripheral interrupt requests (IRQs). In the EFM32, IRQs are generated by peripherals such as TIMERs and GPIOs as a

response to events internal to, or acting upon the MCU. The NVIC's handling of the IRQs is controlled by memory mapped registers (System Control Space). More information on the NVIC can be found in the EFM32 Cortex-M3 Reference Manual.

As shown in Figure 2.1 (p. 4), each IRQ will set a Pending bit when asserted. This pending bit will generate an interrupt request to the CPU if the corresponding enable bit (SETENA[n] in Figure 2.1 (p. 4)) is also set. Note that the pending bit will be automatically cleared by hardware when the corresponding ISR is entered.

Table 2.1 (p. 6) shows the interrupt vector table for the EFM32TG devices. The vector table is common for all devices within the same device series (e.g. EFM32TG), but will vary between device series (e.g EFM32LG devices have a different table than EFM32G devices). The interrupts generated internally in the Cortex-M have negative IRQ numbers, while the peripheral IRQs start at 0.

Table 2.1. EFM32 Interrupts

IRQ #	Source
-	Reset
-14	Non-maskable interrupt (NMI)
-13	Hard fault
-12	Memory management fault
-11	Bus fault
-10	Usage fault
-5	SVCall
-2	PendSV
-1	SysTick
0	DMA
1	GPIO_EVEN
2	TIMER0
3	USART0_RX
4	USART0_TX
5	ACMP0/ACMP1
6	ADC0
7	DAC0
8	I2C0
9	GPIO_ODD
10	TIMER1
11	USART1_RX
12	USART1_TX
13	LESENSE
14	LEUART0
15	LETIMER0
16	PCNT0
17	RTC
18	CMU
19	VCMP
20	LCD
21	MSC
22	AES

2.2.1 Interrupt Priority

Each IRQ in the EFM32 has 3 bits in the Priority Level Registers (IPRn) controlling interrupt priority. A low value means a high priority. These bits are used to configure two types of priority:

- Preempt priority

- Sub priority

The preempt priority level determines whether an interrupt can be executed when the processor is already running another ISR. The sub priority is only evaluated if two interrupts have the same preempt priority and are pending at the same time. The interrupt with the higher sub priority will then be handled first. If two pending interrupts have the same preempt and sub priority, the interrupt with the lower IRQ number will be handled first. By default, the priority of all interrupts is 0 (highest) out of reset. There is no need to write special wrapper code to handle nested interrupts. You only need to configure the appropriate priority for each IRQ.

The PRIGROUP bits in the AIRC Register controls how many of the priority bits are used to encode the preempt priority and how many are used for sub priority as shown in Figure 2.2 (p. 7). By default PRIGROUP is set to 0 and all 3-bits are therefore used for preempt priority.

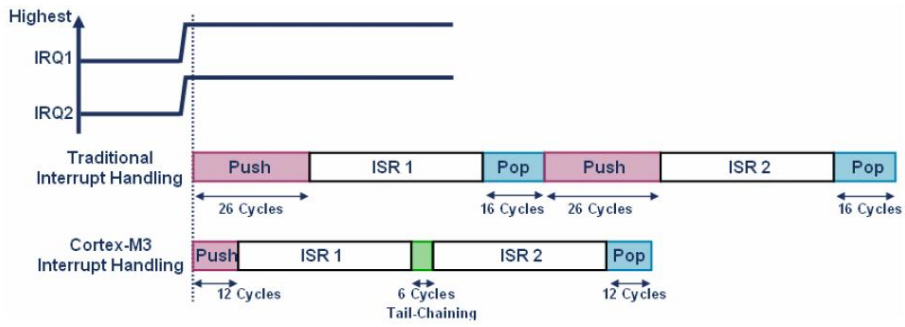
Figure 2.2. Definition of Priority Fields in Priority Level Register

PRIGROUP	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0-4	Preempt priority			Not implemented				
5	Preempt priority		Sub priority	Not implemented				
6	Preempt priority	Sub priority		Not implemented				
7	Sub priority			Not implemented				

2.2.2 Interrupt Sequencing

Before an ISR can be entered the CPU registers must pushed to the stack (stacking), which takes 12 clock cycles (when flash is configured to 0 wait states). Returning from the ISR also takes 12 clock cycles as the CPU state must be restored (unstacking). For ISRs following immediately after (tail-chaining), or nested inside another ISR, the ARM Cortex-M improves latency by not stacking and unstacking fully between the ISRs. This reduces the latency between the handlers to only 6 clock cycles as shown in Figure 2.3 (p. 7) .

Figure 2.3. Latency when entering interrupt handlers



2.3 Sleep Operation

If there is no other work for the CPU to do while waiting for an interrupt to trigger, energy can be saved by putting the CPU to sleep until the interrupt triggered. In the EFM32 there are two ways of going to

sleep, the Wait For Interrupt (WFI) and the Wait For Event (WFE) instructions. After executing one of these instructions the CPU stays in sleep mode until one of the following conditions occur:

- An enabled interrupt request is asserted
- A Debug Entry request occurs, if Debug is enabled.

The SLEEPDEEP bit in the System Control Register (SCR) controls whether the processor uses Sleep or Deep Sleep/Stop Mode as its low power mode according to Table 2.2 (p. 8). Energy Mode 4/ Shut Off Mode is entered by register commands to the Energy Management Unit (EMU) and will require a reset to wake-up thus not allowing regular interrupt operation. For more information on the functional and power consumption differences between the Energy Modes, please refer to the reference manual for the relevant EFM32 device.

Table 2.2. Entering Energy Modes with WFI/WFE

SLEEPDEEP	Low Frequency Oscillator	Resulting Energy Mode when running WFI/WFE
0	NA	Energy Mode 1/Sleep Mode
1	Running	Energy Mode 2/Deep Sleep Mode
1	Stopped	Energy Mode 3/Stopped Mode

2.3.1 ISR Handling With Wait For Interrupt (WFI)

The Wait For Interrupt instruction (__WFI();), causes immediate entry to EM1/2/3. When an enabled IRQ is asserted, the CPU wakes up and executes the associated ISR. When the ISR is done, the main thread continues at the point it was before entering sleep mode.

2.3.2 Low Latency Wake-up With Wait For Event (WFE)

The Wait For Event instruction (__WFE();), causes immediate entry to EM1/2/3. The CPU will wake up once it receives an event. In the EFM32 an event can be generated by a pending interrupt if the Send Event on Pending Interrupt bit (SEVONPEND)in the System Control Register is set. Note that if the interrupt is also enabled, the pending interrupt will also cause the ISR to be executed after waking up. However, if the interrupt is not enabled, the CPU will skip the ISR and continue execution immediately from the WFE instruction. You will then be able to react to the interrupt event faster than if using the ISR as the storing of the CPU state (12 clock cycles) for the ISR is skipped. This allows wake-up in only 1 clock cycle from EM1 and only 2µs from EM2/3. Note though that the interrupt pending bit is only cleared automatically when entering the ISR, so when waking from an event without the ISR, you must first clear the interrupt flag in the peripheral and then clear the pending bit manually before entering a sleep mode again.

Whenever an event occurs, the event status register bit will be set. This is the bit that the WFE command is waiting for. It is important to note that this bit is also set when regular interrupts are executed. If this bit has been set by interrupts executed earlier in the application, and the WFE instruction is executed, the device will immediately wake up again as the event status bit is already set. This bit is not readable directly by software. To make sure that this bit is cleared, it is therefore recommended to run the following sequence before the WFE used to go to sleep:

Example 2.2. Clearing event status bit:

```
__SEV(); /* Set event register bit */
__WFE(); /* Clear the event status bit */

/* __WFE() can now be used to go to sleep */
```

Note that when using other interrupts at the same time as using WFE, the event status bit must be cleared every time an interrupt executes.

2.3.3 Continuous Sleep With SLEEPONEXIT

Normally when the ISR is done, CPU execution returns to where it left off. However, if the SLEEPONEXIT bit in the System Control Register (SCR) is set to 1 the device enters sleep (depth set by DEEPSLEEP bit) directly when finishing the ISR without returning to main. This feature is useful for applications where the program would otherwise enter sleep repeatedly between interrupts. With SLEEPONEXIT set, only one WFI instruction would need to be run and execution would not return to main until the SLEEPONEXIT bit is cleared in the ISR after a desirable condition has been met. Note that if the SLEEPONEXIT bit is changed as the last instruction in the ISR, a Data Synchronization Barrier (DSB assembly instruction) must be run to make sure that the new value is registered before the ISR is exited.

3 CMSIS

The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a common software standard for all Cortex-M devices across all vendors. This means that features, such as the NVIC, which are present in all Cortex microcontrollers, are handled in the same way. The example below shows an ISR for TIMER0 written according to the CMSIS standard. This function is called by the interrupt vector table, which can be found in the startup file for the device (e.g. startup_efm32tg.s).

Example 3.1. A TIMER0 ISR:

```
void TIMER0_IRQHandler(void)
{
  /* Clear the interrupt flag in the beginning */
  TIMER_IntClear(TIMER0, TIMER_IF_OF);

  /* More code */
}
```

As the NVIC is a part of the ARM Cortex-M, CMSIS functions are also provided to handle things like enabling IRQs, (done with the NVIC_EnableIRQ-function in the example below) and configuring priorities. The peripherals (like TIMER0) are not an integrated part of the Cortex and can vary from device to device. CMSIS does not dictate a set of functions for these, although a common naming standard is specified. Energy Micro does however provide a full function library for all peripherals called **emlib**.

Clearing of interrupt flags (TIMER_IntClear-function above) and handling of interrupt enable bits (TIMER_IntEnable-function below) are all handled by the **emlib** functions (functions for TIMER0 are found in *efm32_timer.h*). Please note that the **emlib** and CMSIS functions themselves are shared between all EFM32 devices and individual differences between the sub-series are handled internally in the functions.

Example 3.2. Enabling TIMER0 Overflow Interrupt:

```
/* Enable TIMER0 interrupt in NVIC */
NVIC_EnableIRQ(TIMER0_IRQn);

/* Enable TIMER0 IRQ on Overflow */
TIMER_IntEnable(TIMER0, TIMER_IF_OF);
```

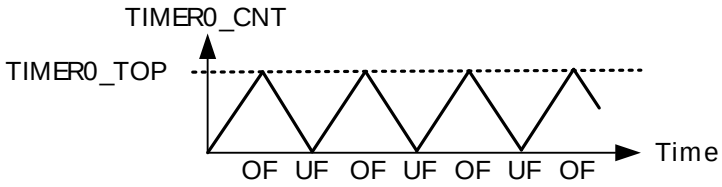
4 Software examples

The included examples are written for the Tiny and Giant Gecko starter kits and their onboard devices EFM32TG840F32 and EFM32GG990F1024, respectively. Source files and projects for several IDEs are also included with this application note.

4.1 WFI

In the **wait_for_interrupt** example project, TIMER0 is set to Up/Down mode and is configured to trigger interrupts when an overflow or underflow occurs. The TOP value is set to give alternating overflow and underflow interrupts at 1 second intervals (shown in Figure 4.1 (p. 11)). The ISR for TIMER0 checks which interrupt flag triggered and writes "OVER" or "UNDER" to the LCD display accordingly. Notice that the interrupt flag register is copied to a variable early in the ISR and then only the interrupt flags that are set are cleared. The copy of the interrupt flags is then evaluated to take appropriate action. Clearing the interrupt flags early in the ISR minimizes the time the flags are set, as other interrupt conditions occurring during this period will be lost as the flags are already set. Even though the interrupt timing in this example is predictable, this structure is still good practice and should be adhered to whenever possible.

Figure 4.1. TIMER0 in Up/Down Count Mode

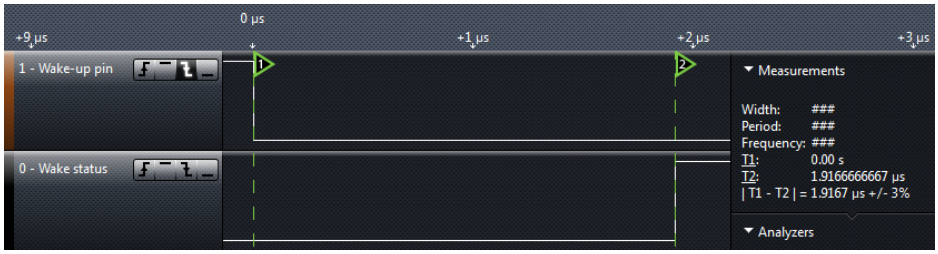


4.2 Reduced Wake-up Latency with SEVONPEND and WFE

In the **wait_for_event** example project, a GPIO pin connected to a LED is toggled every time a falling edge is detected on PB0. The device is in Energy Mode 3 between GPIO interrupts. SEVONPEND is set to generate an event when the IRQ from the GPIO sets the interrupt as pending. The interrupt is not enabled, so the event will only wake-up the device from EM3 and no ISR is called. The LED is toggled immediately after the wake-up. The plot from a logic analyzer in Figure 4.2 (p. 11) shows that it takes less than 2 µs from the time the Wake-up pin goes low to the time the Wake status pin toggles. Immediately after the LED is toggled, the interrupt flag and the pending vector is cleared, making the device ready to wake up from the next event.

On the Tiny Gecko Starter Kit(EFM32TG_STK3300) the LED is located on PD7 and Push button 0 is located on PD8. On the Giant Gecko STK(EFM32GG_STK3700) they are located on PE2 and PB9, respectively.

Figure 4.2. Wake-up Latency From EM2/EM3



4.3 Sleep-on-Exit

The **sleep_on_exit** example project sets up TIMER0 in Up/Count mode to give overflow interrupts every second. The SLEEPONEXIT bit is set and the device is put in EM1. Execution will then not return to main until SLEEPONEXIT is cleared. The ISR increments an interrupt counter which is shown on the LCD every time it is run. After 5 interrupts the SLEEPONEXIT bit is cleared in the ISR and the CPU returns to main where "DONE" is written to the LCD.

4.4 Sub Priority

In the **sub_priority** example project, TIMER0 and TIMER1 are set up to start at the same time and count to 1000, triggering their overflow interrupts in the same cycle. PRIGROUP is configured so all interrupt bits are used to set sub priority. Initially both interrupts have the same default sub priority and TIMER0 ISR will then be executed first as it has the lowest numbered IRQ. Before the TIMERs reach their second overflow interrupts (also in the same cycle), the sub priority for the TIMER0 ISR has been decreased and the TIMER1 ISR will then be executed first. Both ISRs will concatenate a "0" (for TIMER0) or a "1" (for TIMER1) to a string every time they are called. The string is printed at the end of the program to show the order the ISRs were executed in.

4.5 Preempt Priority

In the **preempt_priority** example project, TIMER0 and TIMER1 are set up to trigger overflow interrupts, but they are not enabled. The TIMER0 overflow interrupt is then triggered by setting the overflow interrupt flag in SW. In the TIMER0 ISR, the TIMER1 overflow interrupt is triggered in the same way. PRIGROUP is configured so all interrupt bits are used to set preempt priority. Initially both interrupts have the same default preempt priority and the TIMER1 interrupt triggered in the TIMER0 ISR will then wait until the TIMER0 ISR finishes before starting.

Before TIMER0 overflow is triggered a second time, the preempt priority for the TIMER0 ISR has been decreased. The TIMER1 ISR will then start immediately when triggered in the TIMER0 ISR and the rest of the TIMER0 ISR will complete after the TIMER1 ISR is finished. The TIMER1 ISR will concatenate a "0" to a string every time it is called. The TIMER0 ISR will concatenate an "A" to the string before triggering the TIMER1 overflow interrupt and a "B" after. The string is printed then printed at the end of the program to show the order the ISRs were executed in. Figure 1.2 (p. 3) shows the simplified execution order of a nested interrupt.

4.6 Interrupt Disable

The **interrupt_disable** example project shows a how to disable interrupts temporarily to allow several evaluations to be done in one atomic operation, without the risk of an interrupt hitting in between to corrupt the process. In this example enabling the LFRCO and waiting for the interrupt to trigger when it is stable. In the ISR, a global variable (lfrcoReady) is set to true which is then checked in main. While this variable is false, the device is sent repeatedly to EM1. The repeated check ensures that no other interrupt triggering will incorrectly cause the program to proceed without LFRCO being ready. If this check was done without disabling the interrupts first, an LFRCO-ready interrupt firing after the check for lfrcoReady, but before the sleep instructions, could lock the whole program indefinitely as there might not be any subsequent interrupts to wake the device from sleep.

To disable and enable interrupts safely the em_int.h library is used. The library uses a lock level counter to keep track of how many times `INT_Disable()` and `INT_Enable()` has been called. `INT_Disable()` disables interrupts and increments the counter while `INT_Enable()` decrements the counter and enables interrupts only if the counter reached zero. This ensures that interrupts will not be enabled prematurely if nested disabling and enabling of interrupts is used. Notice that the CPU wakes up when the interrupt is set pending even though the global interrupts have been disabled. The ISR is however not entered until the lock level counter is decremented to zero.

5 Further Reading

Details on the interrupt capabilities of each peripheral in the EFM32 devices can be found in the reference manual for the device. Cortex-internal parts, such as the NVIC, is documented in the EFM32 Cortex-M3 Reference Manual. For a more narrative introduction to the Cortex-M3, "The Definitive Guide To The ARM Cortex-M3" by Joseph Yiu is recommended.

6 Revision History

6.1 Revision 1.04

2013-09-03

New cover layout

6.2 Revision 1.03

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

Specified how to clear event status bit in PDF document and in wait_for_event examples.

6.3 Revision 1.02

2012-11-12

Adapted software projects to new kit-driver and bsp structure.

Updated the Interrupt Disable project to use the em_int.h library.

Added software projects for the Giant Gecko STK.

Added note on using DSB with SLEEPONEXIT.

6.4 Revision 1.01

2012-04-20

Adapted software projects to new peripheral library naming and CMSIS_V3.

6.5 Revision 1.00

2012-02-08

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

A.2 Trademark Information

Silicon Laboratories Inc., Silicon Laboratories, the Silicon Labs logo, Energy Micro, EFM, EFM32, EFR, logo and combinations thereof, and others are the registered trademarks or trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

B Contact Information

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701

Please visit the Silicon Labs Technical Support web page:
http://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.

Table of Contents

List of Figures

List of Tables

List of Examples

silabs.com


ZERO
ARM Cortex-M0+
TINY
ARM Cortex-M3
GECKO
ARM Cortex-M3
LEOPARD
ARM Cortex-M3
GIANT
ARM Cortex-M3
WONDER
ARM Cortex-M4