ECEN 5823-001 / -001B

Mobile Computing & IoT Security

Lecture #4

08 September 2017

Be Boulder.

University of Colorado Boulder

Agenda

- Class announcements
- Office Hours
- Review of Simplicity Exercise – Will postpone to Tuesday the 12th
- Managing Energy Modes Assignment
  - Objective: Become more familiar with the Silicon Labs' Simplicity development system as well as learn the different Blue Gecko energy modes and how to manage them.
- Documentation style sheet
- Interrupts
- LETIMER0
- Wireless Networks - Infrastructure

Class Announcements

- Quiz #2 is due at 11:59pm on Sunday, September 10th, 2017
- Slack is set up
  - Everyone that submitted the Simplicity Exercise has been invited to the ioteff17 slack team
  - The following Slack channels have been setup for the next assignment
    - managingenergymodes
    - letimer0
- Managing Energy Mode Assignment is due at 11:59pm on Wednesday, September 13th, 2017

Office Hours

- Savitha - Tue & Thu - 11 AM to 1 PM
- Raj - Mon & Wed - 12 PM to 2 PM
- Location: ECEE 285 (edited)

Managing Energy Modes Assignment

Program documentation style sheet

- Can be found in the course D2L Content Section under Course Content
  - MCIOTS Style document – Spring 2017.pdf
- The purpose of this document is to set a minimum set of rules of program code documentation to enable:
  - Understanding of code
  - Enable the course instructing team to assist with code when requested
  - Grade the programming assignments
- Not following this style sheet will result in reduction of programming assignment scores

Additional comments on coding style

```
/**************************************************************************//**
 * @file sleep.c
 ******************************************************************************
 * @section License
 * <b>(C) Copyright 2015 Silicon Labs, http://www.silabs.com</b>
 ******************************************************************************
 *
 * Permission is granted to anyone to use this software for any purpose,
 * including commercial applications, and to alter it and redistribute it
 * freely, subject to the following restrictions:
 *
 * 1. The origin of this software must not be misrepresented; you must not
 *    claim that you wrote the original software.
 * 2. Altered source versions must be plainly marked as such, and must not be
 *    misrepresented as being the original software.
 * 3. This notice may not be removed or altered from any source distribution.
 *
 * DISCLAIMER OF WARRANTY/LIMITATION OF REMEDIES: Silicon Labs has no
 * obligation to support this Software. Silicon Labs is providing the
 * Software "AS IS", with no express or implied warranties of any kind,
 * including, but not limited to, any implied warranties of merchantability
 * or fitness for any particular purpose or warranties against infringement
 * of any proprietary rights of a third party.
 *
 * Silicon Labs will not be liable for any consequential, incidental, or
 * special damages, or any other relief, or for any claim by any third party,
 * arising from your use of this Software.
 *
 ******************************************************************************/
```

Additional comments on coding style

For the instructing team to provide help and suggestions with programming assignments, the following documentation must be included. Without this documentation, it is very difficult and time consuming to review and provide help.

1. No use of Init TypeDef defaults. The constructs must be fully specified by one of the following methods. The instructing team does not memorize or know the defaults of the Init TypeDef defaults.

```
LETIMER_Init_TypeDef    LETIMER0_init;
int intFlags;
int Comp0_init;
int Comp1_init;
int LETIMER0_prescaler;
int ULFRCO_count_calibrated;

LETIMER0_init.bufTop = false;
LETIMER0_init.comp0Top = true;
LETIMER0_init.debugRun = false;
LETIMER0_init.enable = false;
LETIMER0_init.out0Pol = 0;
LETIMER0_init.out1Pol = 0;
LETIMER0_init.repMode = letimerRepeatFree;
LETIMER0_init.rtcComp0Enable = false;
LETIMER0_init.rtcComp1Enable = false;
LETIMER0_init.ufoa0 = letimerUFOANone;
LETIMER0_init.ufoa1 = letimerUFOANone;

LETIMER_Init(LETIMER0, &LETIMER0_init);
```

Or,

MCIOTS Style Sheet – Fall 2016

```
/* Set configurations for LETIMER 0 */
 const LETIMER_Init_TypeDef letimerInit =
 {
 .enable         = true,
 .debugRun       = false,
 .rtcComp0Enable = false,
 .rtcComp1Enable = false,
 .comp0Top       = true,
 .bufTop         = false,
 .out0Pol        = 0,
 .out1Pol        = 0,
 .ufoa0          = letimerUFOAPwm,
 .ufoa1          = letimerUFOAPulse,
 .repMode        = letimerRepeatFree
 };

 /* Initialize LETIMER */
 LETIMER_Init(LETIMER0, &letimerInit);
```

2. No register bit manipulation with direct bit or hex values where appropriate and possible. Use of using the enumerations provided by the MCU/SOC provider to address bit names and fields. Similar to the use of Init TypeDef defaults, the instruction team does not memorize the value of each bit field in a register.

~~LETIMER0->IEN |= 0x00000004; Not acceptable~~
LETIMER0->IEN |= LETIMER_IEN_UF; Acceptable

Energy Profiler providing erratic results

- If your Energy profiler is measuring current in the sleep mode < than 1nA which is not possible, you may need to reset the Energy profiler and Simplicity to obtain correct results. Below is the suggested process to reset your environment:

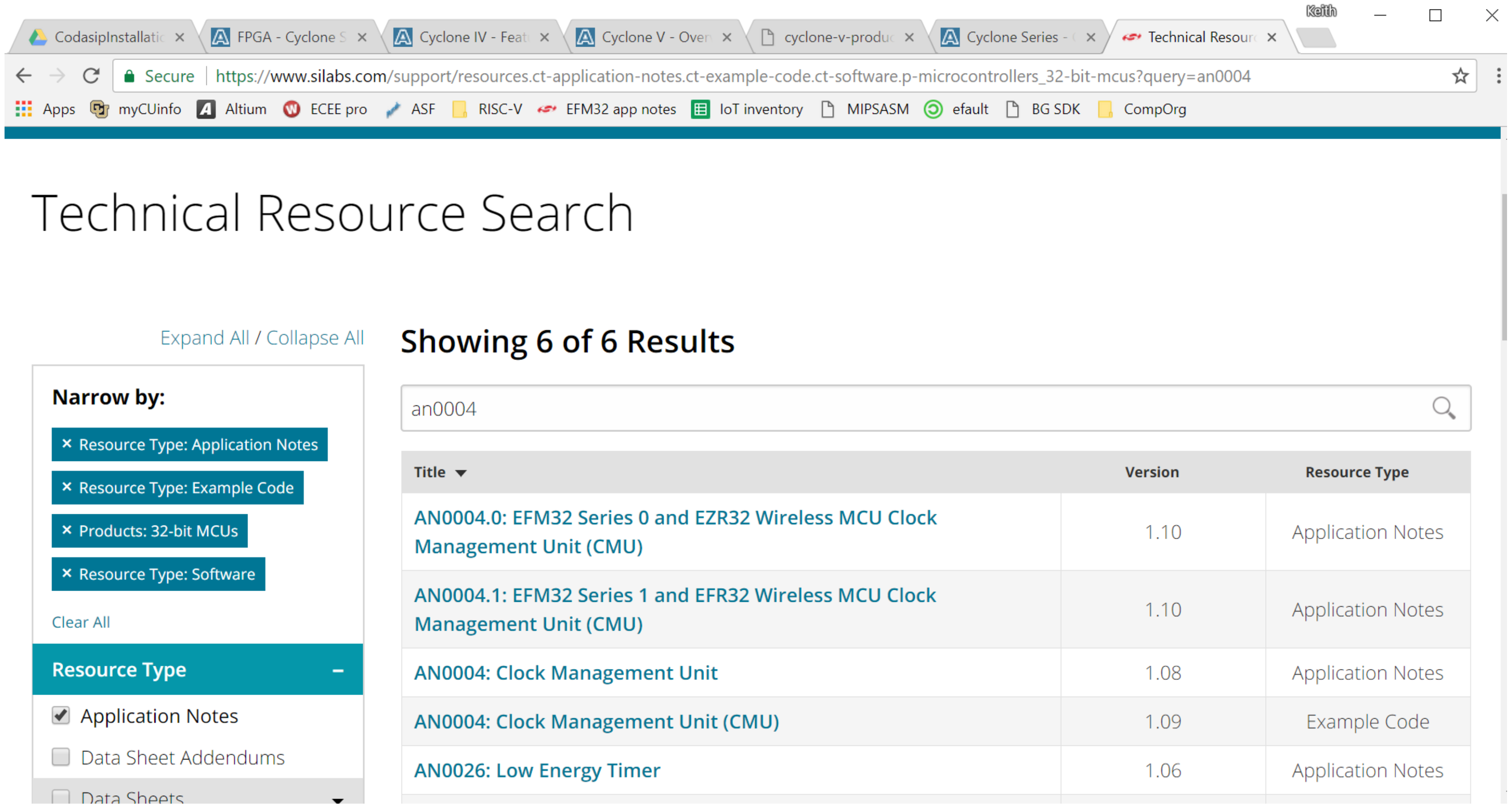
  1. Quit Studio.
  2. Unplug your device from PC, then re-plug it.
  3. Start Studio. That should solve the problem.

  This problem usually happens when you plug in your device while Studio is running.

http://www.silabs.com/products/mcu/Pages/32-bit-mcu-application-notes.aspx

Application notes

+ Software examples



Technical Resource Search

Expand All / Collapse All

Narrow by:

× Resource Type: Application Notes
× Resource Type: Example Code
× Products: 32-bit MCUs
× Resource Type: Software

Clear All

Resource Type

[✓] Application Notes
[ ] Data Sheet Addendums
[ ] Data Sheets

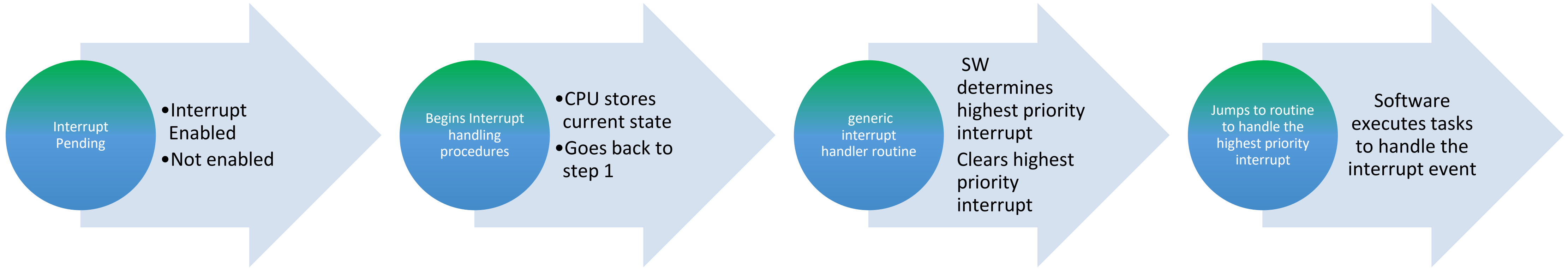
Showing 6 of 6 Results

an0004

Title	Version	Resource Type
AN0004.0: EFM32 Series 0 and EZR32 Wireless MCU Clock Management Unit (CMU)	1.10	Application Notes
AN0004.1: EFM32 Series 1 and EFR32 Wireless MCU Clock Management Unit (CMU)	1.10	Application Notes
AN0004: Clock Management Unit	1.08	Application Notes
AN0004: Clock Management Unit (CMU)	1.09	Example Code
AN0026: Low Energy Timer	1.06	Application Notes

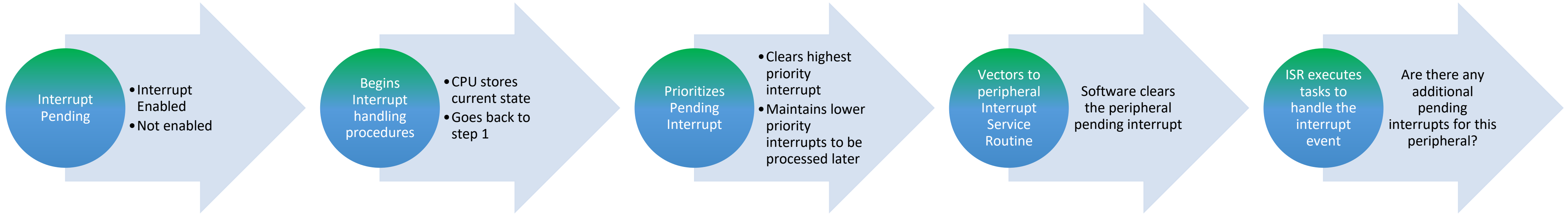
Interrupt Requests (IRQ)

- Generic steps of a controller receiving an interrupt
- Enabled, Prioritize, Vector to Interrupt Service Routine


Interrupt Pending
•Interrupt Enabled
•Not enabled
Begins Interrupt handling procedures
•CPU stores current state
•Goes back to step 1
generic interrupt handler routine
SW determines highest priority interrupt
Clears highest priority interrupt
Jumps to routine to handle the highest priority interrupt
Software executes tasks to handle the interrupt event

ARM Cortex-M4 Interrupt process

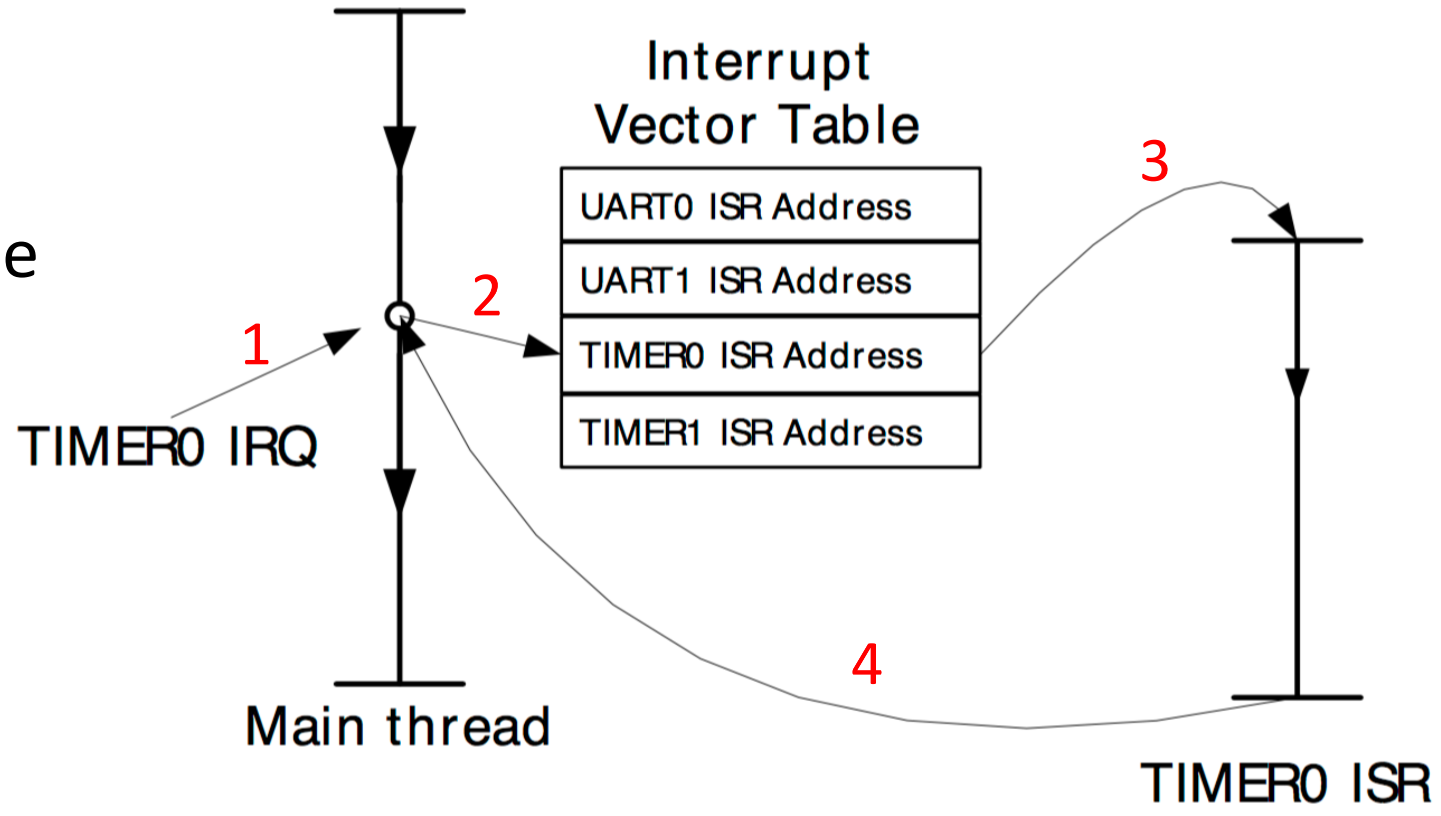
- Similar to the generic interrupt process, but there are multiple specific ISRs available
- These ISRs more efficient direct the controller to the required Interrupt Handle



Interrupt Pending
• Interrupt Enabled
• Not enabled
Begins Interrupt handling procedures
• CPU stores current state
• Goes back to step 1
Prioritizes Pending Interrupt
• Clears highest priority interrupt
• Maintains lower priority interrupts to be processed later
Vectors to peripheral Interrupt Service Routine
Software clears the peripheral pending interrupt
ISR executes tasks to handle the interrupt event
Are there any additional pending interrupts for this peripheral?

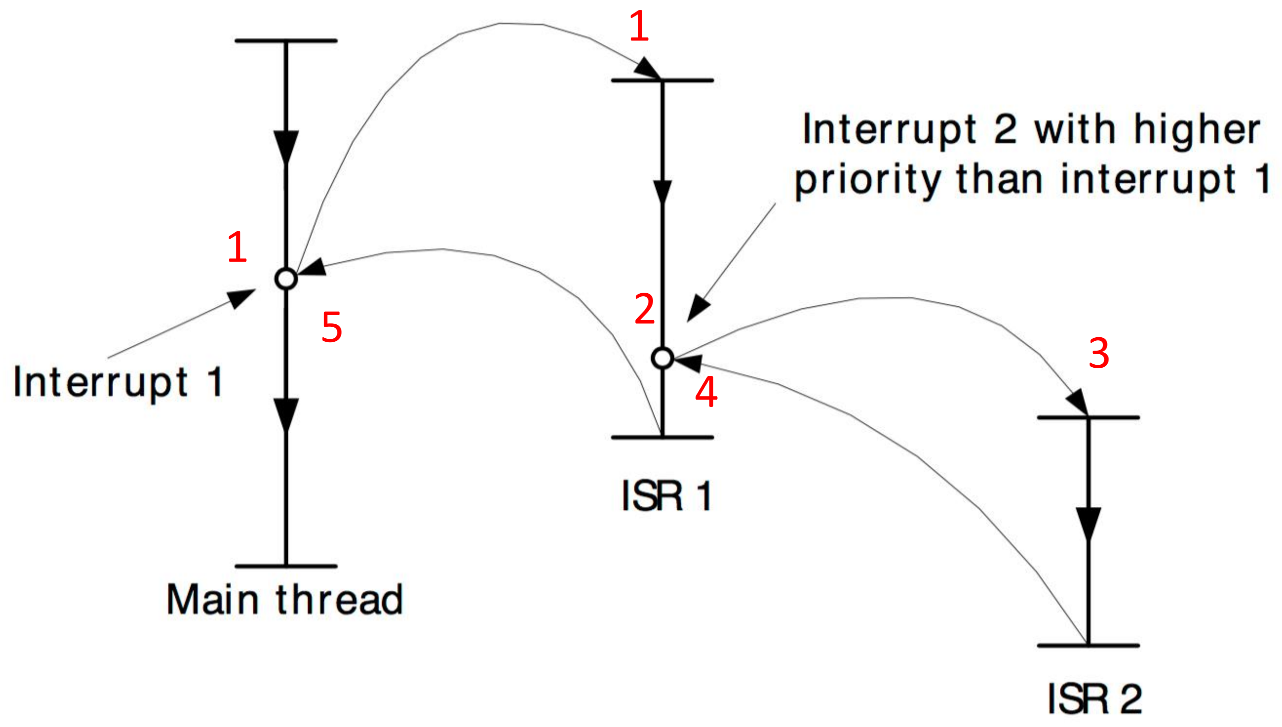
ARM Cortex-M4

Single Interrupt Example



ARM Cortex-M4 Nested Interrupt example

1. CPU is interrupt with a low priority interrupt
2. While in the low priority ISR, a higher priority interrupt occurs
3. System jumps to the higher priority ISR in the middle of the lower priority ISR
4. The higher priority routine completes and returns to the initial ISR
5. The low priority routine completes and returns to where the CPU left off



ARM Cortex-M4 Interrupt priority

- How do you determine which interrupt can interrupt an Interrupt Service Routine (ISR)?
  - Prioritizing the Interrupts
    - Lower-latency interrupts can interrupt a lower-priority ISR to service the most time critical interrupts such as servicing a UART before its buffer overflows
    - A higher priority interrupt will be handled before a lower priority interrupt if they occur simultaneously
- The CPU will continue where it left off after all the pending interrupts have been serviced

ARM Cortex-M4 internal interrupts

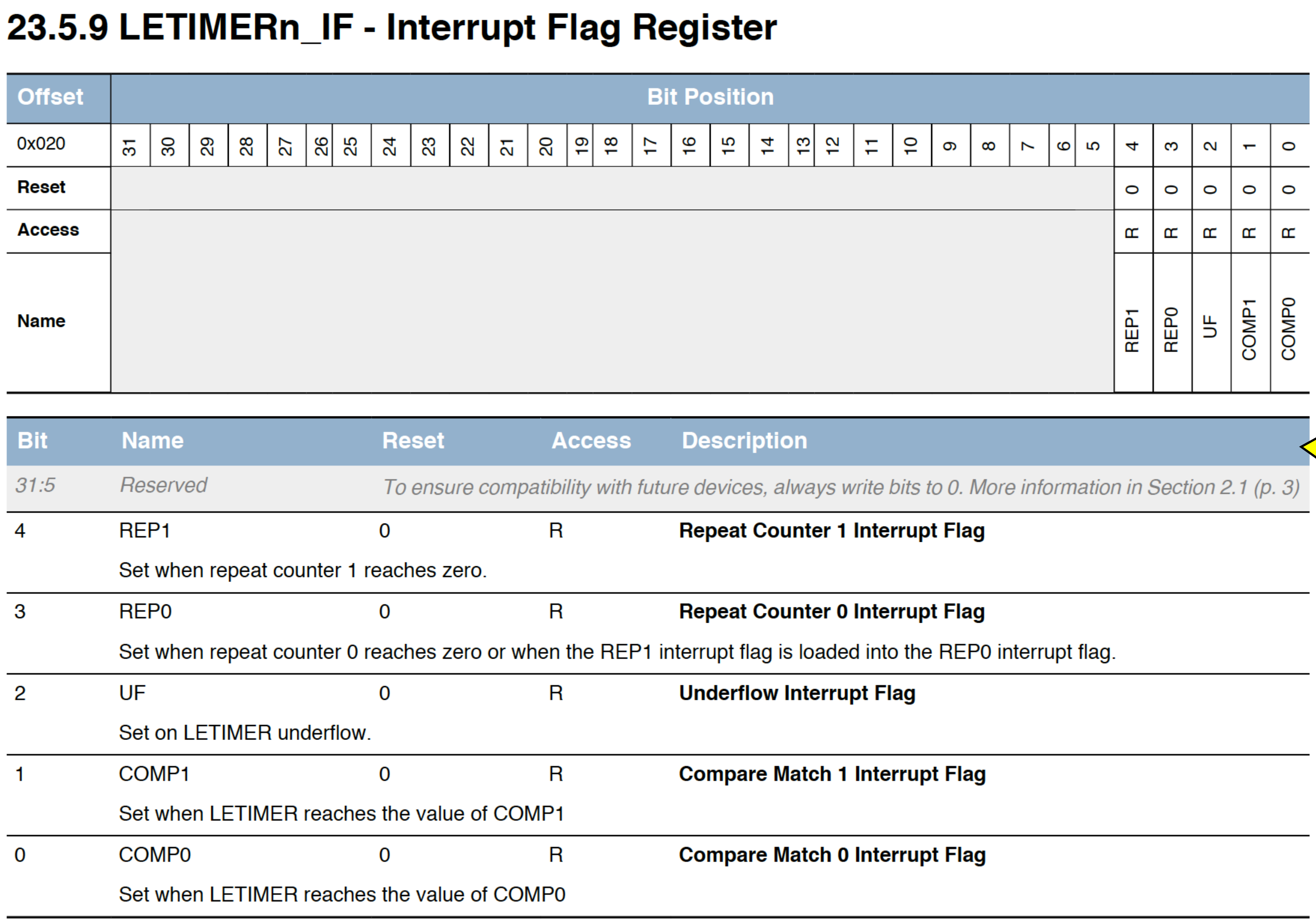
```
__isr_vector:
    .long    __StackTop           /* Top of Stack */
    .long    Reset_Handler        /* Reset Handler */
    .long    NMI_Handler          /* NMI Handler */
    .long    HardFault_Handler    /* Hard Fault Handler */
    .long    MemManage_Handler    /* MPU Fault Handler */
    .long    BusFault_Handler     /* Bus Fault Handler */
    .long    UsageFault_Handler   /* Usage Fault Handler */
    .long    Default_Handler      /* Reserved */
    .long    Default_Handler      /* Reserved */
    .long    Default_Handler      /* Reserved */
    .long    Default_Handler      /* Reserved */
    .long    SVC_Handler          /* SVCall Handler */
    .long    DebugMon_Handler     /* Debug Monitor Handler */
    .long    Default_Handler      /* Reserved */
    .long    PendSV_Handler       /* PendSV Handler */
    .long    SysTick_Handler      /* SysTick Handler */
```

Silicon Labs Gecko peripheral interrupts

```
/* External interrupts */
    .long    DMA_IRQHandler    /* 0 - DMA */
    .long    GPIO_EVEN_IRQHandler    /* 1 - GPIO_EVEN */
    .long    TIMER0_IRQHandler    /* 2 - TIMER0 */
    .long    USART0_RX_IRQHandler    /* 3 - USART0_RX */
    .long    USART0_TX_IRQHandler    /* 4 - USART0_TX */
    .long    ACMP0_IRQHandler    /* 5 - ACMP0 */
    .long    ADC0_IRQHandler    /* 6 - ADC0 */
    .long    DAC0_IRQHandler    /* 7 - DAC0 */
    .long    I2C0_IRQHandler    /* 8 - I2C0 */
    .long    GPIO_ODD_IRQHandler    /* 9 - GPIO_ODD */
    .long    TIMER1_IRQHandler    /* 10 - TIMER1 */
    .long    TIMER2_IRQHandler    /* 11 - TIMER2 */
    .long    USART1_RX_IRQHandler    /* 12 - USART1_RX */
    .long    USART1_TX_IRQHandler    /* 13 - USART1_TX */
    .long    USART2_RX_IRQHandler    /* 14 - USART2_RX */
    .long    USART2_TX_IRQHandler    /* 15 - USART2_TX */
```

```
.long    UART0_RX_IRQHandler    /* 16 - UART0_RX */
.long    UART0_TX_IRQHandler    /* 17 - UART0_TX */
.long    LEUART0_IRQHandler    /* 18 - LEUART0 */
.long    LEUART1_IRQHandler    /* 19 - LEUART1 */
.long    LETIMER0_IRQHandler    /* 20 - LETIMER0 */
.long    PCNT0_IRQHandler    /* 21 - PCNT0 */
.long    PCNT1_IRQHandler    /* 22 - PCNT1 */
.long    PCNT2_IRQHandler    /* 23 - PCNT2 */
.long    RTC_IRQHandler    /* 24 - RTC */
.long    CMU_IRQHandler    /* 25 - CMU */
.long    VCMP_IRQHandler    /* 26 - VCMP */
.long    LCD_IRQHandler    /* 27 - LCD */
.long    MSC_IRQHandler    /* 28 - MSC */
.long    AES_IRQHandler    /* 29 - AES */
```

Peripheral IRQ generation



23.5.9 LETIMERn_IF - Interrupt Flag Register

Offset	Bit Position																															
0x020	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																												0	0	0	0	0
Access																												R	R	R	R	R
Name																												REP1	REP0	UF	COMP1	COMP0

Bit	Name	Reset	Access	Description
31:5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	REP1	0	R	**Repeat Counter 1 Interrupt Flag**
	Set when repeat counter 1 reaches zero.			
3	REP0	0	R	**Repeat Counter 0 Interrupt Flag**
	Set when repeat counter 0 reaches zero or when the REP1 interrupt flag is loaded into the REP0 interrupt flag.			
2	UF	0	R	**Underflow Interrupt Flag**
	Set on LETIMER underflow.			
1	COMP1	0	R	**Compare Match 1 Interrupt Flag**
	Set when LETIMER reaches the value of COMP1			
0	COMP0	0	R	**Compare Match 0 Interrupt Flag**
	Set when LETIMER reaches the value of COMP0			

- A peripheral interrupt to the system will only occur when:
  - The interrupt conditions sets its bit in the IF register, and
  - The corresponding bit in the IEN register is set
  - And, the Interrupt has been enabled through the MCU NVIC, Nested Vector Interrupt Controller

NVIC – Nested Vector Interrupt Controller

- Integrated in the ARM Cortex-M processor
  - Each IRQ will set a pending bit in the NVIC register when asserted
  - An interrupt to the Interrupt Service Routine will occur only if this interrupt is Enabled in the NVIC
  - The pending bit will automatically be cleared by hardware when the corresponding ISR is entered
  - NOTE: The interrupt flag in the peripheral Interrupt Flag registers are not automatically cleared when the ISR is entered

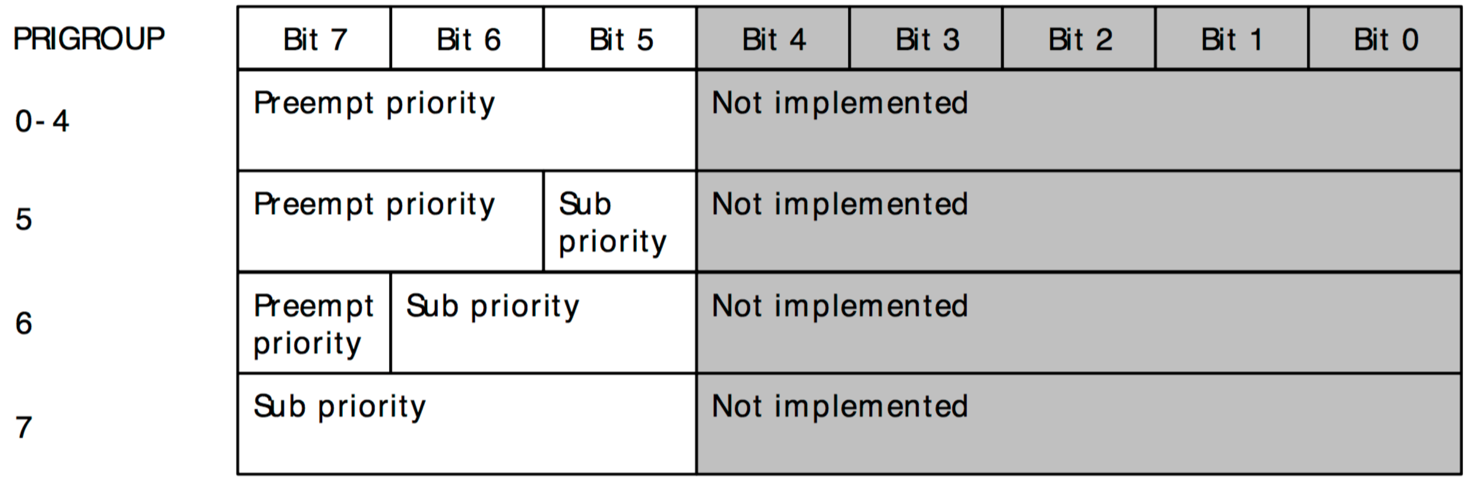
Interrupt priority

- Each IRQ has 3 bits in the Priority Level Registers (IPRn) that control the interrupt priority
- These bits can be configured to two types of priority:
  - Preempt – determines whether an interrupt can be executed when the processor is already running another ISR
  - And, sub priority – determines which interrupt is vectored to if two interrupts have the same preempt priority
    - If the preempts have the same sub priority, the interrupt with the lower IRQ number will be handled first (ex. IRQ0 has highest priority out of reset)

Interrupt Priority Register

- The number of bits for preempt and sub priority are defined by the bits set in the AIRC register

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				

How to insure atomic instruction operation?

- In concurrent programming, an **operation** (or set of **operations**) is **atomic**, linearizable, indivisible or uninterruptible if it appears to the rest of the system to occur instantaneously. Atomicity is a guarantee of isolation from concurrent processes. (definition by Google)
  - All interrupts disabled, CORE_ATOMIC_IRQ_DISABLE();
  - Atomic operation
  - All interrupts renabled, CORE_ATOMIC_IRQ_ENABLE();

Best practices in ISR code writing

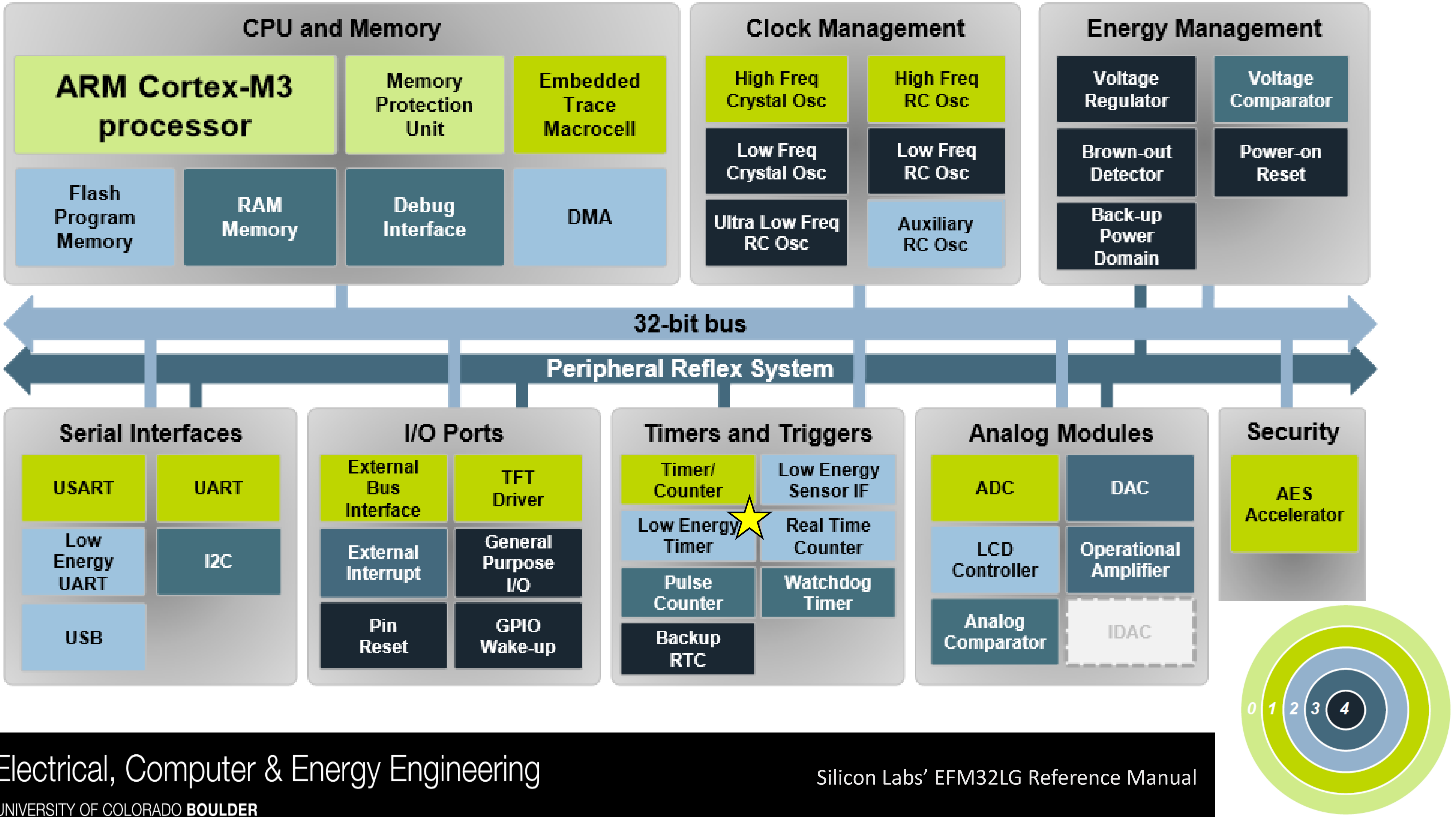
Clear pending interrupts immediate in the ISR so that if another interrupt occurs, you will not be clearing an interrupt that has not been processed

```
void peripheral_IRQHandler() {
        int intFlags;
        intFlags = Peripheral_IntGet(Peripheral); //determine pending interrupts
        Peripheral_IntClear(Peripheral, intFlags);
        /*ISR handling code based on interrupts set in intFlag*/
}
```

Best practices in ISR code writing – part 2

If an interrupt routine needs to be atomic or it cannot be interrupted by another interrupt, interrupts through the NVIC should be disabled and then re-enabled

```
void peripheral_IRQHandler() {
        int intFlags;
        CORE_ATOMIC_IRQ_DISABLE();
        intFlags = Peripheral_IntGet(Peripheral); //determine pending interrupts
        Peripheral_IntClear(Peripheral, intFlags);
        /*ISR handling code based on interrupts set in intFlag*/
        CORE_ATOMIC_IRQ_DISABLE();
}
```

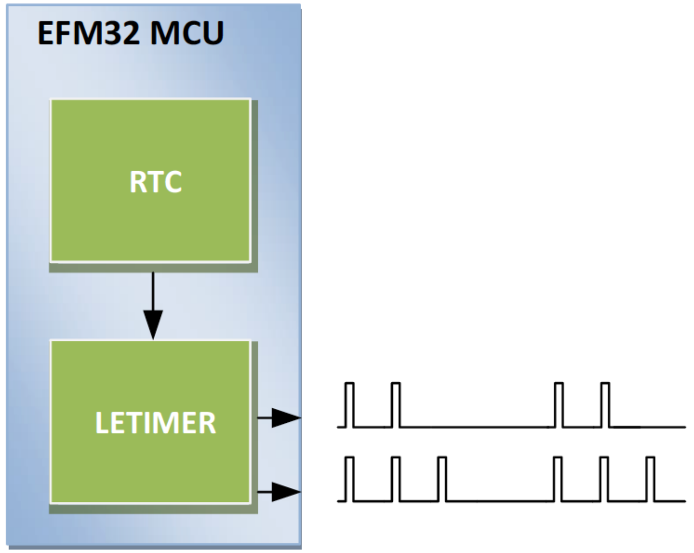


Silicon Labs' EFM32LG Reference Manual

Low Energy Timer

Low Energy Timer Highlights

- **16-bit counter, 8-bit repeat**
- **Clocked from LFXO/LFRCO**
- **Waveform generation**
- **Duty cycle control of external components/sensors**
- **Availabled down to Deep Sleep (EM2)**

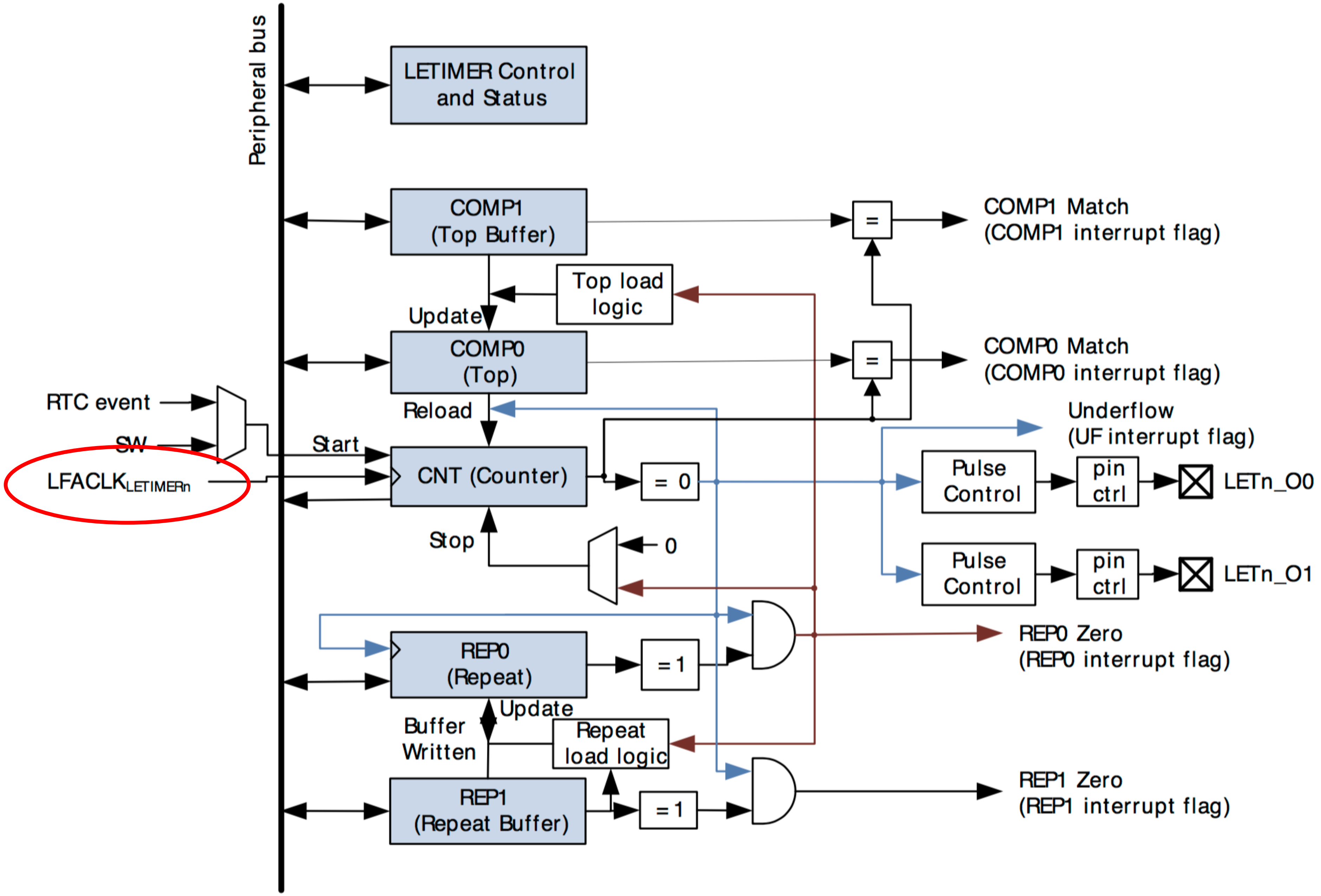


Blue Gecko - Available down in EM3 using the ULFRC0

Low Energy Timer

- a 16-bit down counter which is clocked off the LFA clock branch
- The top of the counter can be set to COMP0 or to 0xFFFF upon underflow, reaching 0
- Interrupts can be generated on count matches to COMP0, COMP1, and Underflow
  - The Interrupt status can be found in the LETIMER0 IF register
- The LETIMER clock frequency is defined by the following equation based on the LETIMERn 4-bit prescaler value in the CMU_LFAPRESC0 register
    - LETIMERfrequency = LFACLKfrequency / 2^LETIMERn

Low Energy Timer



Setting up the LETIMER0

- First, the clock tree to the LETIMER0 must be established
  - Without establishing the clock tree, all writes to the LETIMER0 registers will not occur
  - Pseudo code in the CMU setup routine to enable the LETIMER0 clock tree:
    - If using LFXO, enable the LFXO using the CMU_OscillatorEnable routine
    - Select the appropriate Low Frequency clock for the LFA clock tree depending on lowest energy mode for the LETIMER0
      - If EM0 – EM2, use CMU_ClockSelectSet to select the LFXO for LFA
      - If EM3, use the CMU_ClockSelectSet to select ULFRCO for LFA
    - Enable the Low Frequency clock tree by using the CMU_ClockEnable for CORELE
    - Lastly, enable the LFA clock tree to the LETIMER0 using the CMU_ClockEnable for the LETIMER0

Setting up the LETIMER0

- Second, the LETIMER0 must be set up
  - Define all variables in the LETIMER_Init_TypeDef to configure the LETIMER0 to perform as desired (disable the LETIMER0 at this time)
  - Then initialize the LETIMER0 using the LETIMER_Init command
  - If required, writing directly to the CMU->LFAPRESCO register, update the LFA prescaler
  - Program the COMP0 and COMP1 register with the values required to obtain the functionality desired using LETIMER_CompareSet command
  - Wait for the LETIMER0 synch bit is cleared before proceeding by accessing the register LETIMER0->SYNCBUSY

Setting up the LETIMER0

- Third, the LETIMER0 interrupts must be enabled
  - Clear all interrupts from the LETIMER0 to remove any interrupts that may have been set up inadvertently by accessing the LETIMER0->IFC register or the emlib routine
  - Enable the appropriate LETIMER0 interrupts by setting the appropriate bits in the LETIMER0->IEN register or using an emlib routine
  - Set the appropriate BlockSleep mode for this peripheral based on the system configuration such as going to EM3 or limiting to a higher EM level such as EM1 or EM2
  - Enable interrupts to the CPU by enabling the LETIMER0 in the Nested Vector Interrupt Control register using NVIC_EnableIRQ(LETIMER0_IRQn);

Setting up the LETIMER0

- Fourth, the LETIMER0 interrupt handler must be included
  - Routine name must match the vector table name:

    ```
    Void LETIMER0_IRQHandler(void) {
    }
    ```

  - Inside this routine, you add the functionality that is desired for the LETIMER0 interrupts
  - Note: Most timers are meant to repeat, so an unBlockSleep call most likely will not be needed in the LETIMER0 interrupt handler

Setting up the LETIMER0

- Lastly, enable the LETIMER0 when it is desired to have the peripheral to start operation
  - Can enable the LETIMER0 by writing directly to the LETIMER0 or using the emlib routin LETIMER_Enable(LETIMER0, true);

LETIMER0 Compare Registers

- The LETIMER has two compare match registers, LETIMERn_COMP0 and LETIMERn_COMP1
  - Each of these compare registers are capable of generating an interrupt when the counter value LETIMERn_CNT becomes equal to their value.
  - When LETIMERn_CNT becomes equal to the value of LETIMERn_COMP0, the interrupt flag COMP0 in LETIMERn_IF is set, and when LETIMERn_CNT becomes equal to the value of LETIMERn_COMP1, the interrupt flag COMP1 in LETIMERn_IF is set.
    - Setting the correct count value with the known period of the clock used by LETIMER, the period of the LETIMER can be divided into an On Duty Cycle and an Off Duty Cycle

LETIMER0 Top Value

- If COMP0TOP in LETIMERn_CTRL is set, the value of LETIMERn_COMP0 acts as the top value of the timer, and LETIMERn_COMP0 is loaded into LETIMERn_CNT on timer underflow.
  - A specific period of the LETIMER0 can be set by COMP0TOP being set and the correct count value programmed into COMP0 if the clock period is known for the LETIMER
- Else, the timer wraps around to 0xFFFF. The underflow interrupt flag UF in LETIMERn_IF is set when the timer reaches zero
  - The period with COMP0TOP is defined by 0xFFFF * the clock period used for LETIMER

LETIMER Buffered Top Value

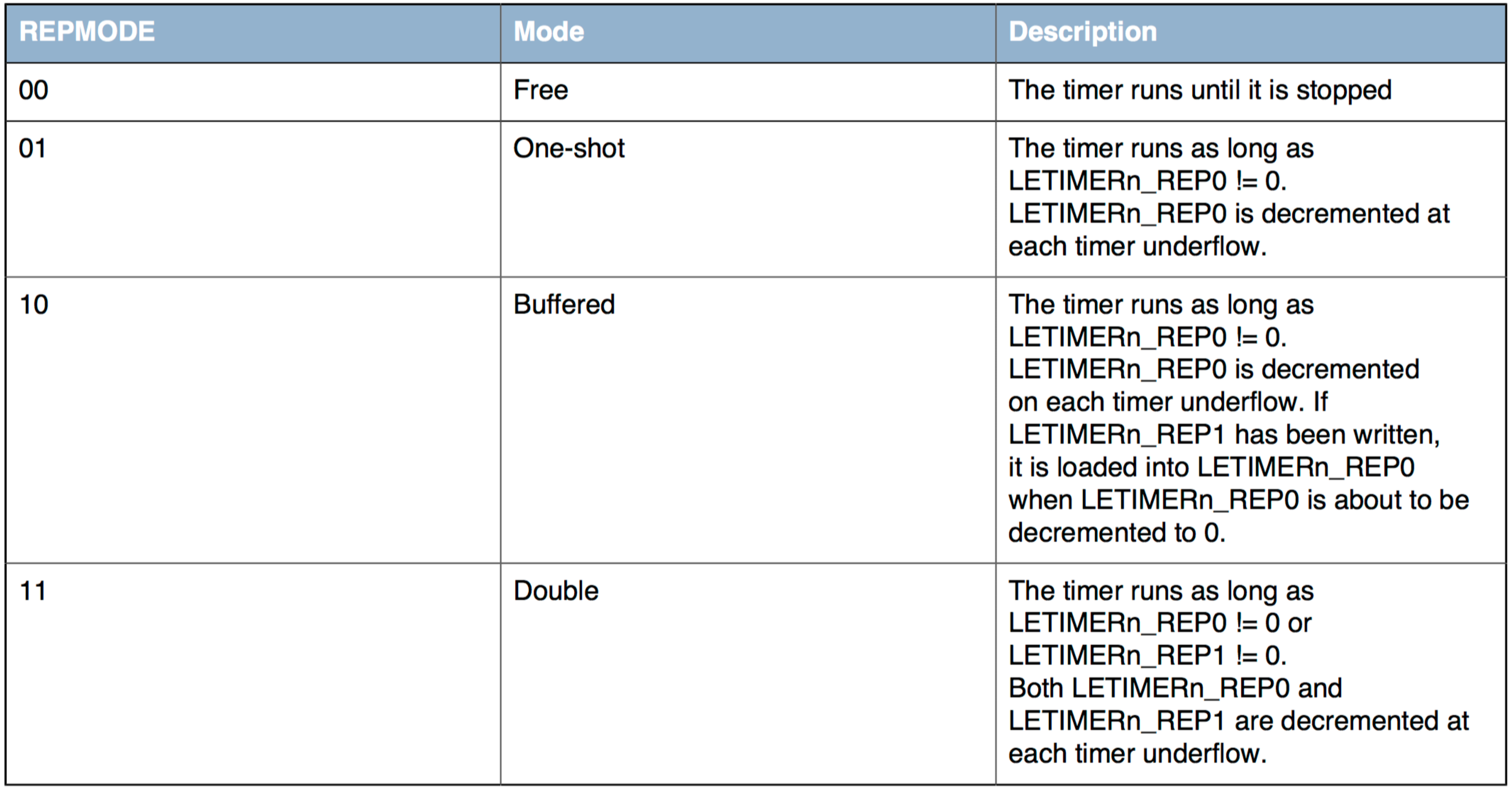
- If BUFTOP in LETIMERn_CTRL is set, the value of LETIMERn_COMP0 is buffered by LETIMERn_COMP1
- In this mode, the value of LETIMERn_COMP1 is loaded into LETIMERn_COMP0 every time LETIMERn_REP0 is about to decrement to 0
- This can for instance be used in conjunction with the buffered repeat mode to generate continually changing output waveforms
- Write operations to LETIMERn_COMP0 have priority over buffer loads

Setting up the LETIMER0

- Lastly, enable the LETIMER0 when it is desired to have the peripheral to start operation
  - Can enable the LETIMER0 by writing directly to the LETIMER0 or using the emlib routin LETIMER_Enable(LETIMER0, true);

LETIMER Repeat Modes

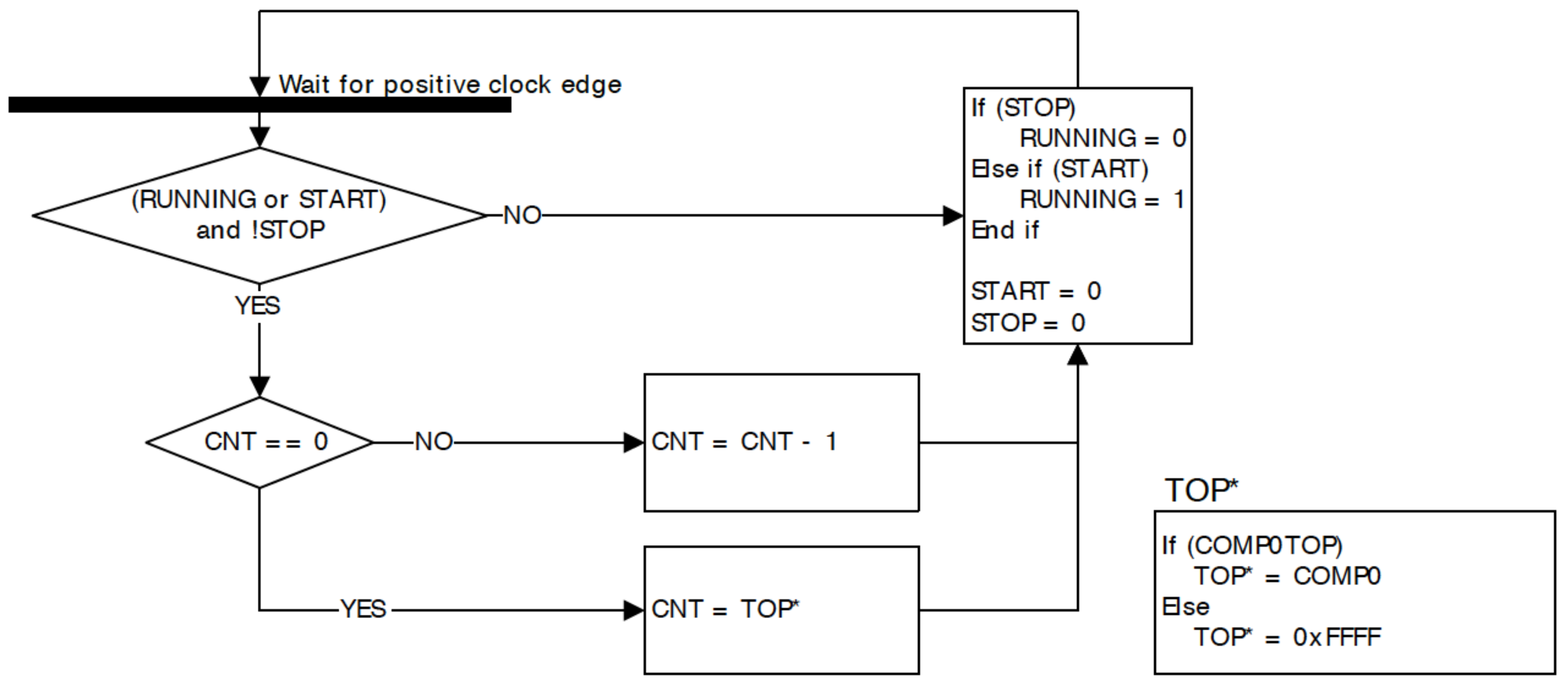
Table 2.1. LETIMER Repeat Modes



REPMODE	Mode	Description
00	Free	The timer runs until it is stopped
01	One-shot	The timer runs as long as LETIMERn_REP0 != 0. LETIMERn_REP0 is decremented at each timer underflow.
10	Buffered	The timer runs as long as LETIMERn_REP0 != 0. LETIMERn_REP0 is decremented on each timer underflow. If LETIMERn_REP1 has been written, it is loaded into LETIMERn_REP0 when LETIMERn_REP0 is about to be decremented to 0.
11	Double	The timer runs as long as LETIMERn_REP0 != 0 or LETIMERn_REP1 != 0. Both LETIMERn_REP0 and LETIMERn_REP1 are decremented at each timer underflow.

Free Running flow diagram

Figure 23.2. LETIMER State Machine for Free-running Mode

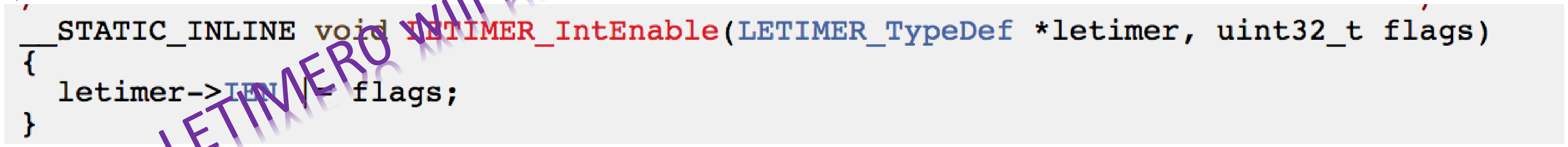


Wait for positive clock edge
(RUNNING or START) and !STOP
NO
YES
If (STOP) RUNNING = 0 Else if (START) RUNNING = 1 End if START = 0 STOP = 0
CNT == 0
NO
CNT = CNT - 1
YES
CNT = TOP*
TOP*
If (COMP0TOP) TOP* = COMP0 Else TOP* = 0xFFFF

Silicon Labs' EFM32 reference manual

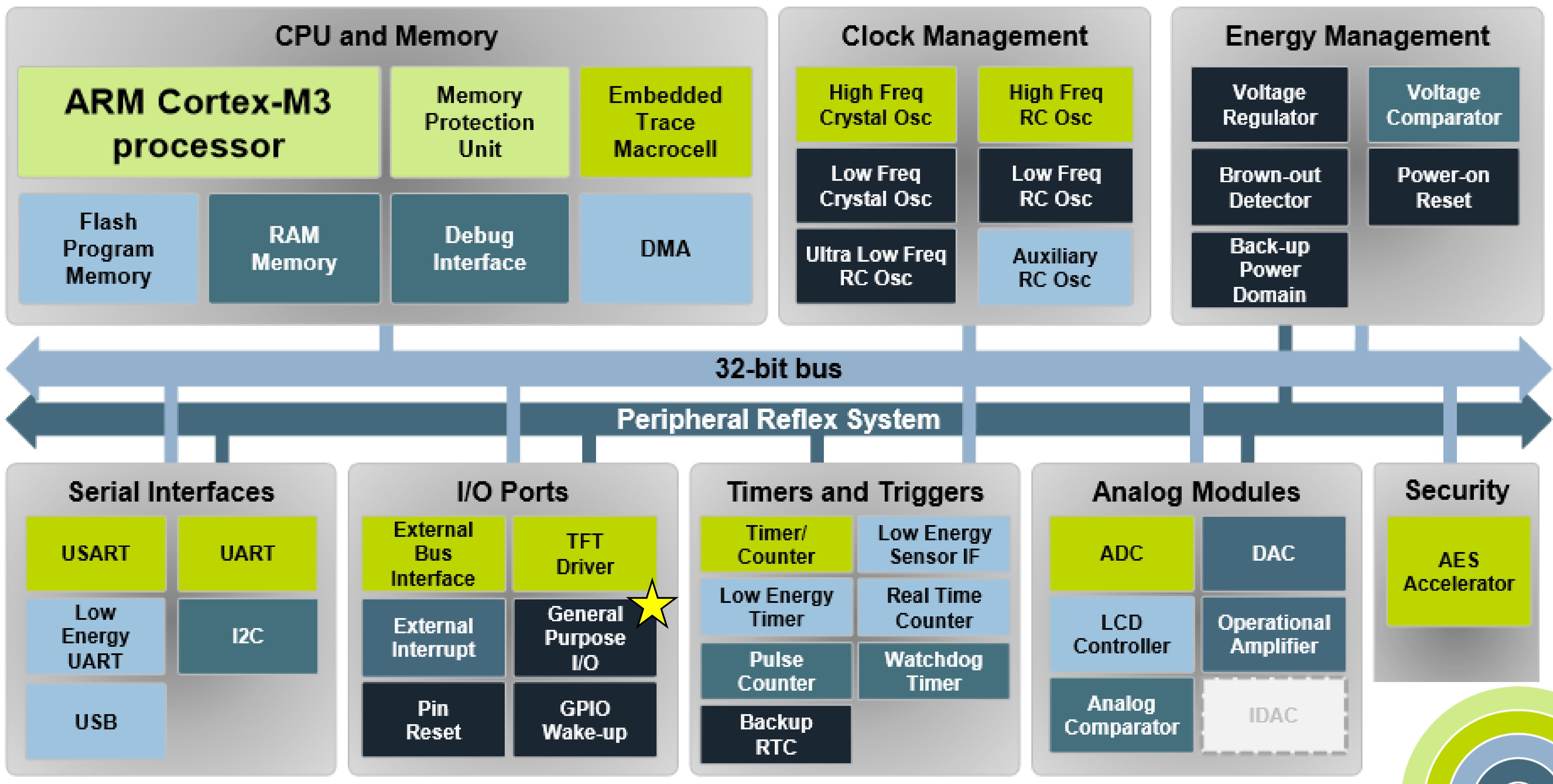
LETIMER interrupt emlib routine examples

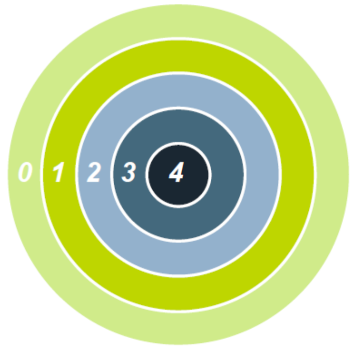
- There are 5 interrupts available for LETIMER0
  - REP0, REP1, COMP0, COMP1, and UL
- emlib routine to enable interrupts
  - LETIMER_IntEnable(LETIMER_TypeDef *letimer, unit32_flags);
- emlib routine to disable interrupts
  - LETIMER_IntDisable(LETIMER_TypeDef *letimer, unit32_flags);
- emlib routine to clear interrupts
  - LETIMER_IntClear(LETIMER_TypeDef *letimer, unit32_flags);
- example



```
__STATIC_INLINE void LETIMER_IntEnable(LETIMER_TypeDef *letimer, uint32_t flags)
{
  letimer->IEN |= flags;
}
```

LETIMER0 will be used for LETIMER_TypeDef for the Blue Gecko

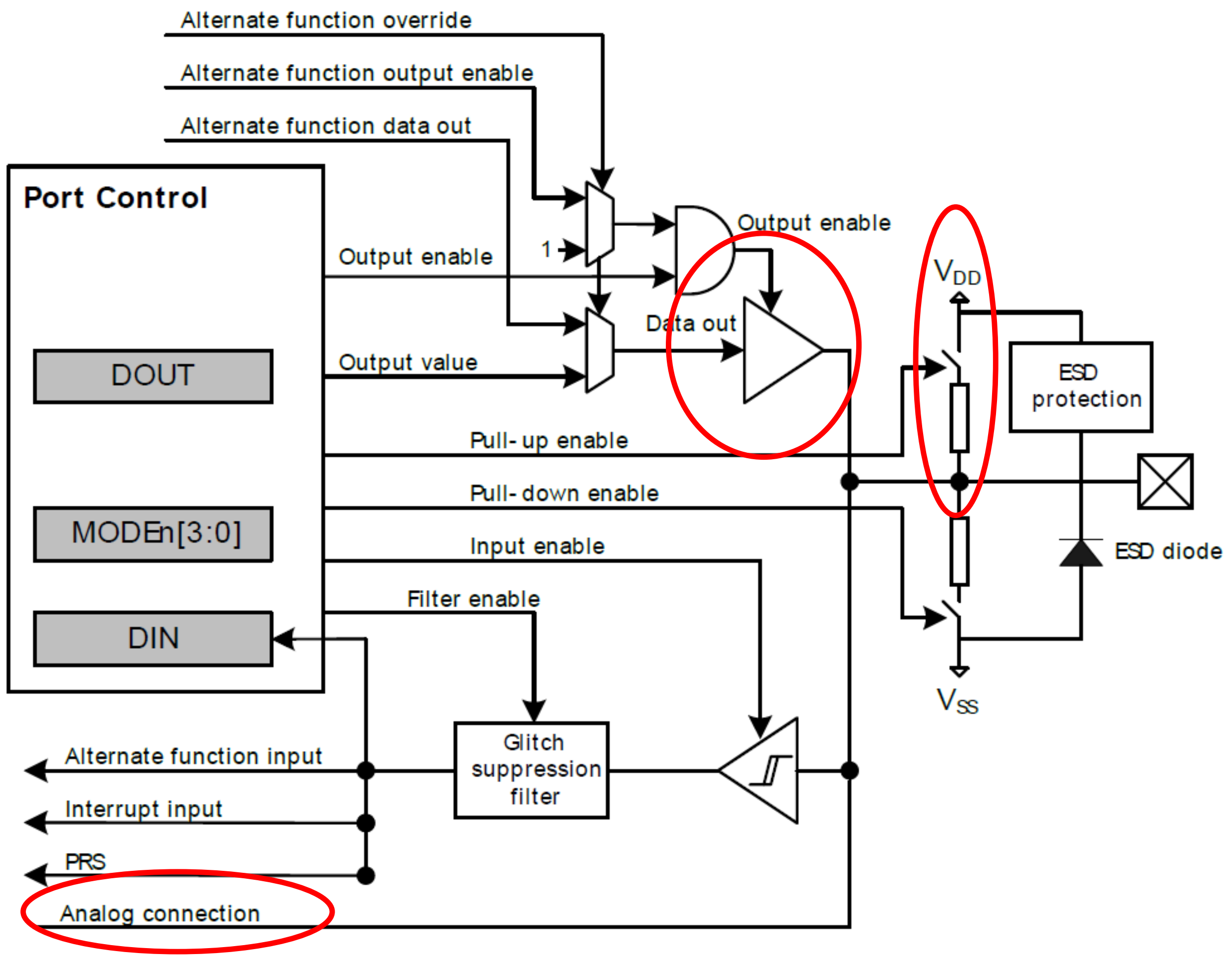




Silicon Labs' EFM32LG Reference Manual

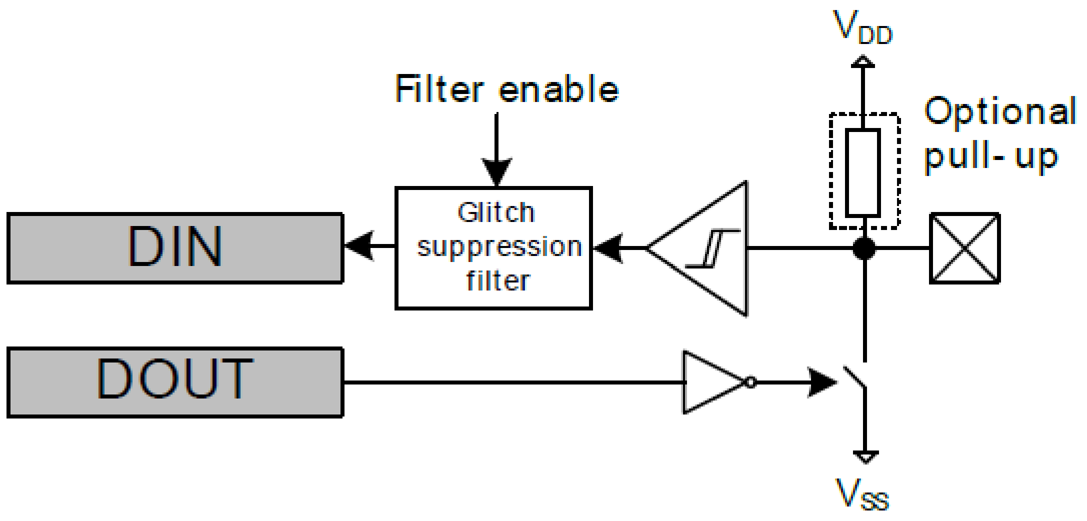
GPIO Peripheral

- Individual configuration for each pin
  - Tristate (reset state)
  - Push-pull
  - Open-drain
  - Pull-up resistor
  - Pull-down resistor
  - Four drive strength modes
    - HIGH
    - STANDARD
    - LOW
    - LOWEST



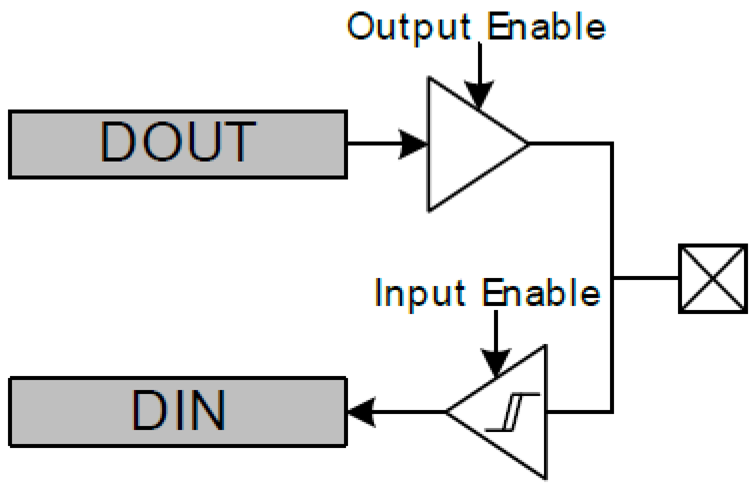
GPIO – Open drain

- Common configuration when multiple sources may drive the bus
  - I2C



GPIO – Push/Pull configuration

- Common Output or input configuration when a single source is driving the net
- In output mode, it can be used to:
  - Source current to an LED by driving the positive node of the LED
  - Or, sink current from an LED by connecting to ground the negative/ground node of the LED

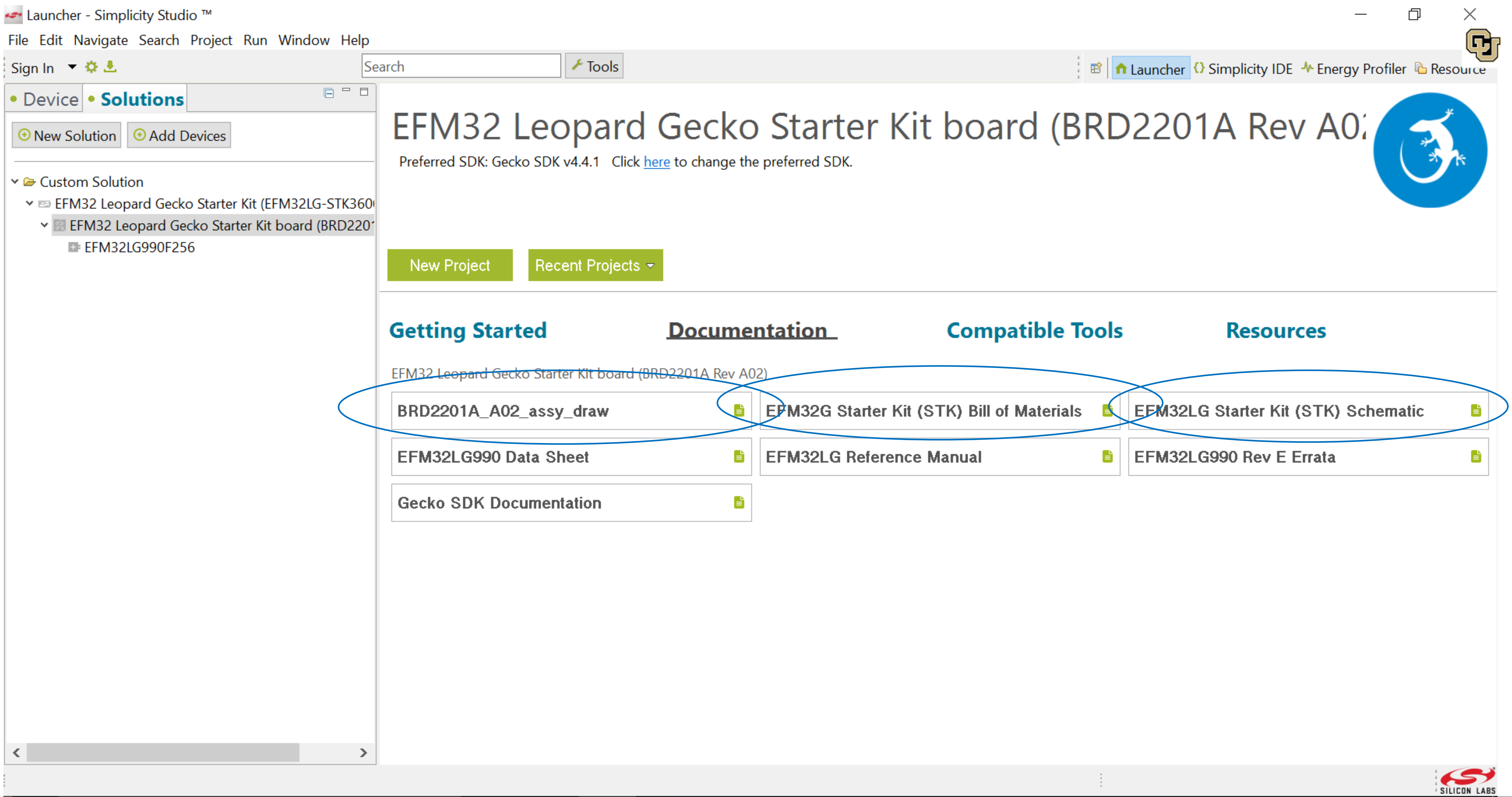


Setting up the GPIO

- First, the clock tree to the GPIO must be established
  - Without establishing the clock tree, all writes to the GPIO registers will not occur
  - Pseudo code in the CMU setup routine to enable the GPIO clock tree:
    - Lastly, enable the GPIO clocking using the CMU_ClockEnable for the GPIO

Setting up the GPIO

- Second, the GPIO must be set up
  - Specifying the pins
    - Both the Port and Pin # is required
    - For the Leopard Gecko, use the schematic from Simplicity
  - What is the function of the pins?
    - Push-Pull
    - Open drain
    - Etc.
  - Program the functionality of the GPIO pin using GPIO_PinModeSet
  - Program drive strength of the GPIO pin using GPIO_DriveModeSet



Launcher - Simplicity Studio ™

File Edit Navigate Search Project Run Window Help

Sign In | Search | Tools | Launcher | Simplicity IDE | Energy Profiler | Resource

Device | Solutions

New Solution | Add Devices

- Custom Solution
  - EFM32 Leopard Gecko Starter Kit (EFM32LG-STK360
    - EFM32 Leopard Gecko Starter Kit board (BRD220
      - EFM32LG990F256

EFM32 Leopard Gecko Starter Kit board (BRD2201A Rev A0

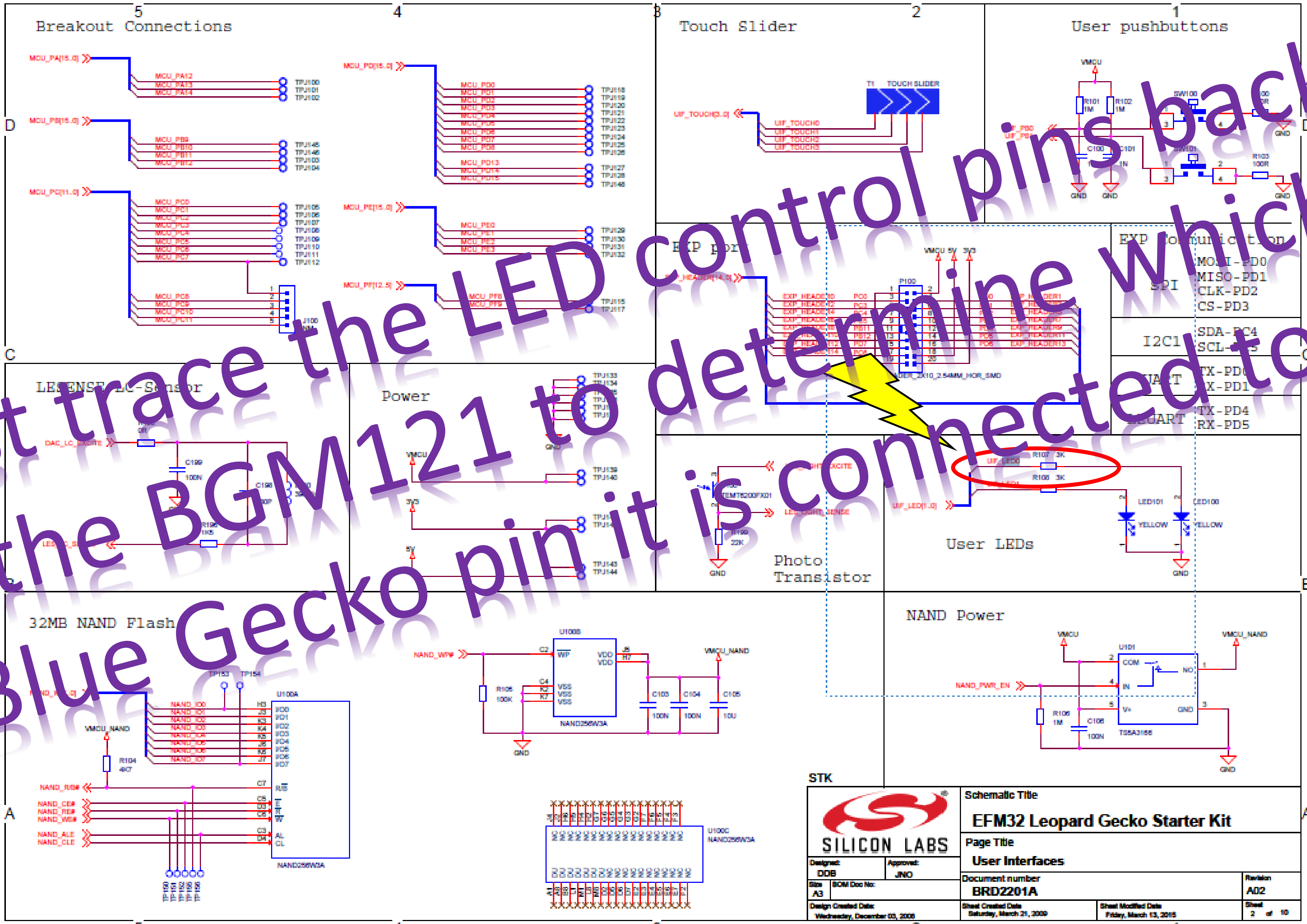
Preferred SDK: Gecko SDK v4.4.1 Click here to change the preferred SDK.

New Project | Recent Projects

Getting Started | Documentation | Compatible Tools | Resources

EFM32 Leopard Gecko Starter Kit board (BRD2201A Rev A02)

BRD2201A_A02_assy_draw	EFM32G Starter Kit (STK) Bill of Materials	EFM32LG Starter Kit (STK) Schematic
EFM32LG990 Data Sheet	EFM32LG Reference Manual	EFM32LG990 Rev E Errata
Gecko SDK Documentation		



Must trace the LED control pins back to the BGM121 to determine which Blue Gecko pin it is connected to

Setting up the GPIO

- Third, the GPIO interrupts must be enabled if needed
  - Clear all interrupts from the GPIO to remove any interrupts that may have been set up inadvertently by accessing the GPIO->IFC register or the emlib routine
  - GPIO_IntConfig emlib command to set GPIO interrupts
  - Enable the appropriate GPIO interrupts by setting the appropriate bits in the GPIO->IEN register or using an emlib routine
    - There are two interrupt vectors (handlers) for the GPIO
      - Even GPIO pins
      - Odd GPIO pins
  - No need to set BlockSleep mode since GPIO pins work EM0 thru EM3
  - A subset of GPIO pins works down to EM4
  - Enable interrupts to the CPU by enabling the GPIO in the Nested Vector Interrupt Control register using NVIC_EnableIRQ(GPIO_EVEN_IRQn); or NVIC_EnableIRQ(GPIO_ODD_IRQn);

Setting up the GPIO

- Fourth, the GPIO interrupt handler must be included
  - Routine name must match the vector table name:

```
Void GPIO_EVEN_IRQHandler(void) {
}
Or
Void GPIO_ODD_IRQHandler(void) {
}
```

  - Inside this routine, you add the functionality that is desired for the GPIO interrupts

GPIO – Input / Output

- To read a GPIO pin, direct register access can be used or emlib routine GPIO_PinInGet
- Setting a GPIO pin output:
  - Programming a one or “high” with GPIO_PinOutSet
  - Programming a zero or “low” with GPIO_PinOutClear