# ECEN 5823-001
# Internet of Things Embedded Firmware

Lecture #4

06 September 2018

Be Boulder.

University of Colorado Boulder

## Agenda

- Class Announcements
- Simplicity Exercise Review
- Homework #1 assigned:  Managing Energy Modes
- Managing Energy  Modes
- Interrupts
- LETIMER0

## Class Announcements

- Quiz #2 is due at 11:59 on Sunday, September 9th, 2018
- Assignment #1: Managing Energy Modes is due, Saturday, September 15th, at 11:59pm via Canvas
  - Depending how far we get in today's lecture, I may move it to Saturday the 15th

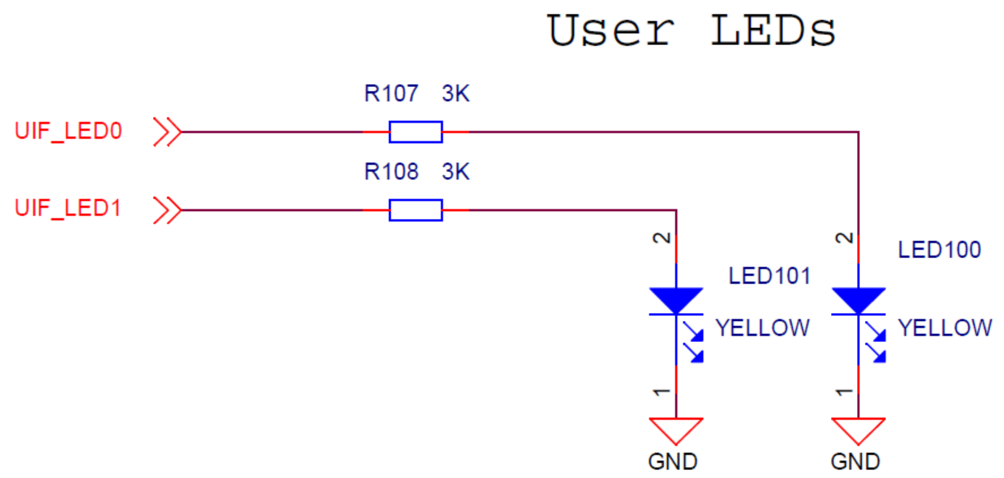
# Simplicity Exercise Rubric

1. Total points for this exercise is 5 points
   a. 3.5 pts for the questions
   b. 1.5 pts of the code

2. Question scoring. Max score is 3.5 pts.
   a. Question 1: Full credit if values provided in range 0.42 to 0.53mA (0.5 pts)
   b. Question 2: Full credit if values provided in range 0.42 to 0.53mA (0.5 pts)
   c. Question 3: There is no difference in LED current between weak and strong drive (0.5 pts)
      i. Specifying just due to the LED voltage drop, (+0.5 pts)
      ii. Specifying that due to the LED voltage drop and the series resistor the current is below the weak drive capability (+1pt)
   d. Question 4: somewhere close to 4.40-4.90mA (0.5 pts)
   e. Question 5: approximately 0.10-0.20mA less than answer to question 4 (0.5 pts)

3. Functional code delivered per exercise. Max score is 1.5 pts.
   a. If the code is coming from a Windows machine, I would like it be delivered via exporting the project. For Linux and Mac OS based systems, we may need to rely on just the .c and .h files.
   b. Code is functional (1pt)
   c. Code is running as specified which is 1 LED with weak drive (0.5 pts)

# Why is there no difference in current?



$$I = \frac{(Vsupply - Vf_{led})}{Resistance}$$

$$I = \frac{(3.3 - 2.0)}{3K}$$

$$I = \frac{1.3}{3K}$$

$$I = 433uA$$

**Electrical & Optical Properties:**

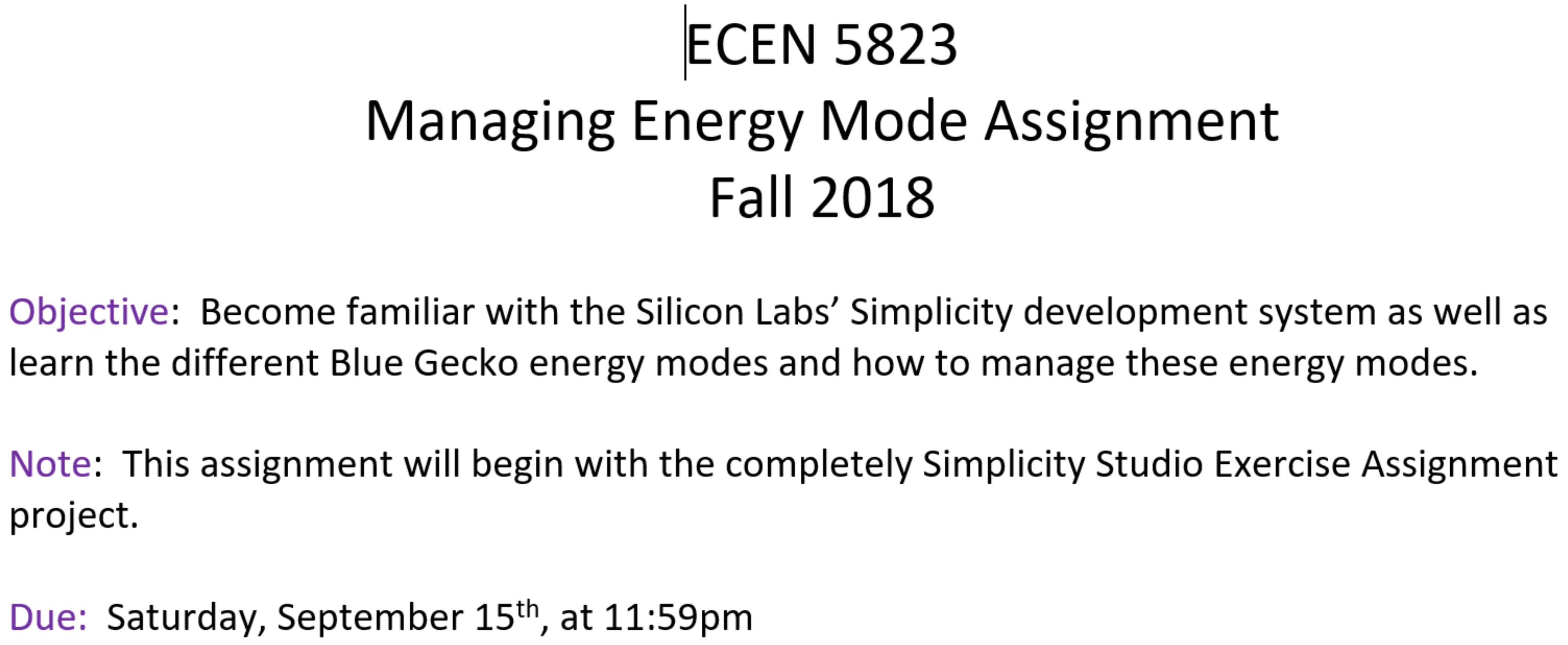
| Properties | Test conditions | | Value | | | Unit |
|---|---|---|---|---|---|---|
| | | | min. | typ. | max. | |
| Peak Wavelength | 20 mA | $\lambda_{Peak}$ | | 595 | | nm |
| Dominant Wavelength | 20 mA | $\lambda_{Dom}$ | | 590 | | nm |
| Luminous Intensity | 20 mA | $I_V$ | 90 | 120 | | mcd |
| Forward Voltage | 20 mA | $V_F$ | | 2 | 2.4 | V |
| Spectral Bandwidth | 20 mA | $\Delta\lambda$ | | 15 | | nm |
| Reverse Current | 5 V | $I_{REV}$ | | | 10 | µA |
| Viewing Angle Phi 0° | 20 mA | $2\theta_{50\%}$ | | 140 | | ° |

**enum GPIO_DriveStrength_TypeDef**

GPIO drive strength.

| Enumerator | |
|---|---|
| gpioDriveStrengthWeakAlternateWeak | GPIO weak 1mA and alternate function weak 1mA |
| gpioDriveStrengthWeakAlternateStrong | GPIO weak 1mA and alternate function strong 10mA |
| gpioDriveStrengthStrongAlternateWeak | GPIO strong 10mA and alternate function weak 1mA |
| gpioDriveStrengthStrongAlternateStrong | GPIO strong 10mA and alternate function strong 10mA |

## Assignment #1 assigned



ECEN 5823
Managing Energy Mode Assignment
Fall 2018

Objective: Become familiar with the Silicon Labs' Simplicity development system as well as learn the different Blue Gecko energy modes and how to manage these energy modes.

Note: This assignment will begin with the completely Simplicity Studio Exercise Assignment project.

Due: Saturday, September 15th, at 11:59pm

## Homework #1 Demo

## IMPORTANT PROGRAMMING NOTE!!!

- If you disable both of the HF clock sources, HFXO and HFRCO, you will "brick" your dev kit

I know, I have done it!

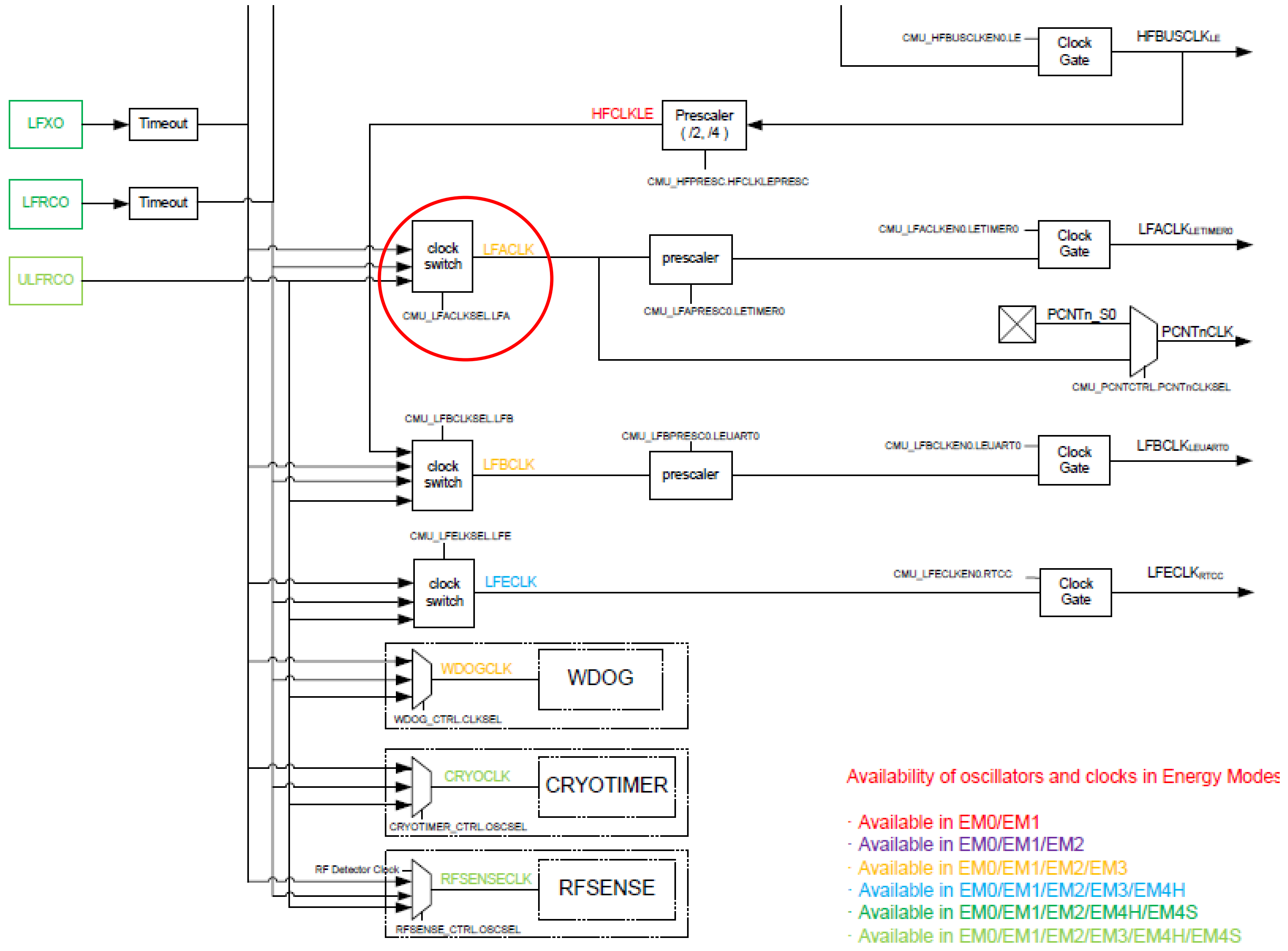
- Before disabling the HFXO, insure that you first enable the HFRCO and select the HFRCO as the HF clock source!

## Enabling a peripheral

- Before programming a peripheral, what aspects of the clock tree must be configured?
  - Set the oscillator frequency
  - Enable its oscillator
  - Configure the oscillator to the peripheral clock branch
  - Program the peripheral prescaler
  - Enable the peripheral clock

# Blue Gecko Low Frequency Clock Tree

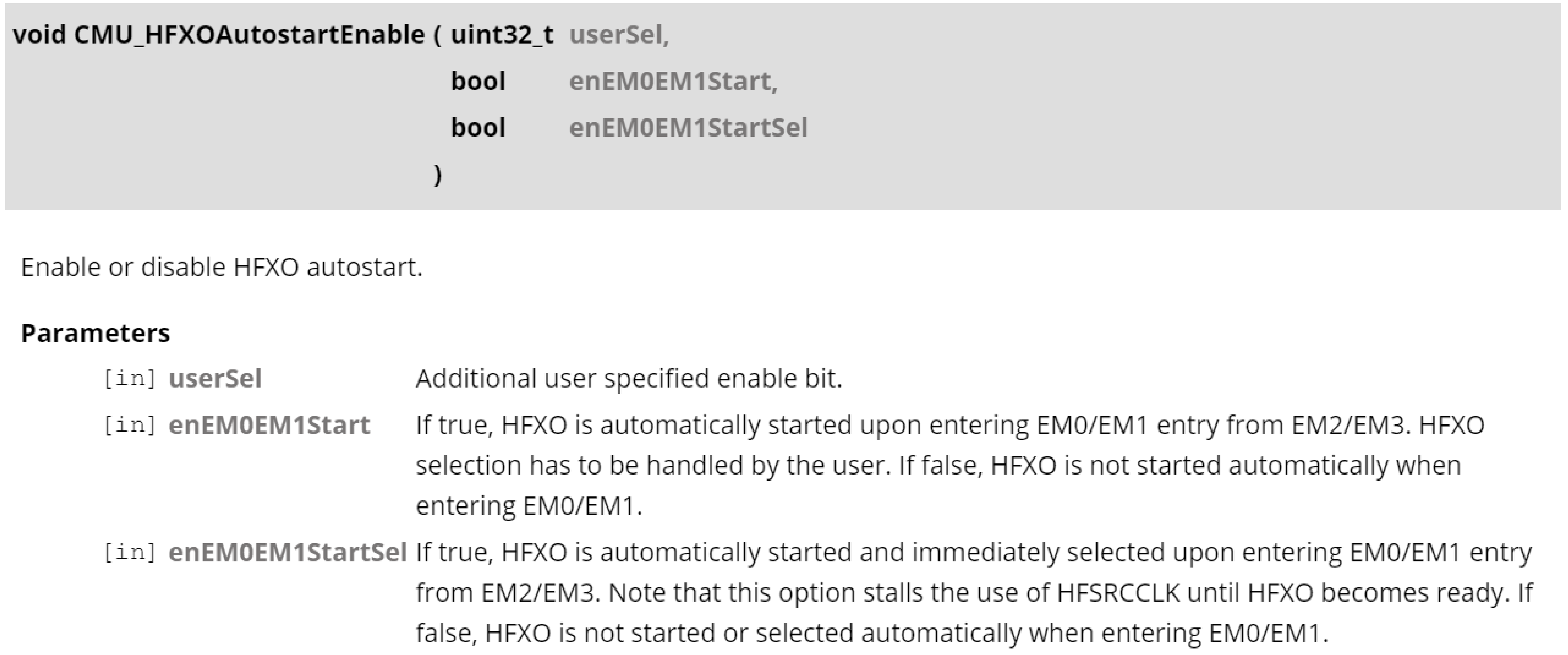
## Clock Source Selection



## HFXO/AUXHFXO Band Selection

- The default HF clock source is set to the HFXO @ 38 MHz
- The HFXO can be changed to 1, 2, 4, 7, 13, 16, 19, 26, and 38 MHz
- emlib routine to set the HFRCO band:
  - CMU_HFRCOBandSet(CMU_HFRCOBand_TypeDef band)
- There is also a AUXHFRCO that can be set using
  - CMU_AUXHFRCOBandSet(CMU_HFRCOBand_TypeDef band)

## HFXO AutostartEnable



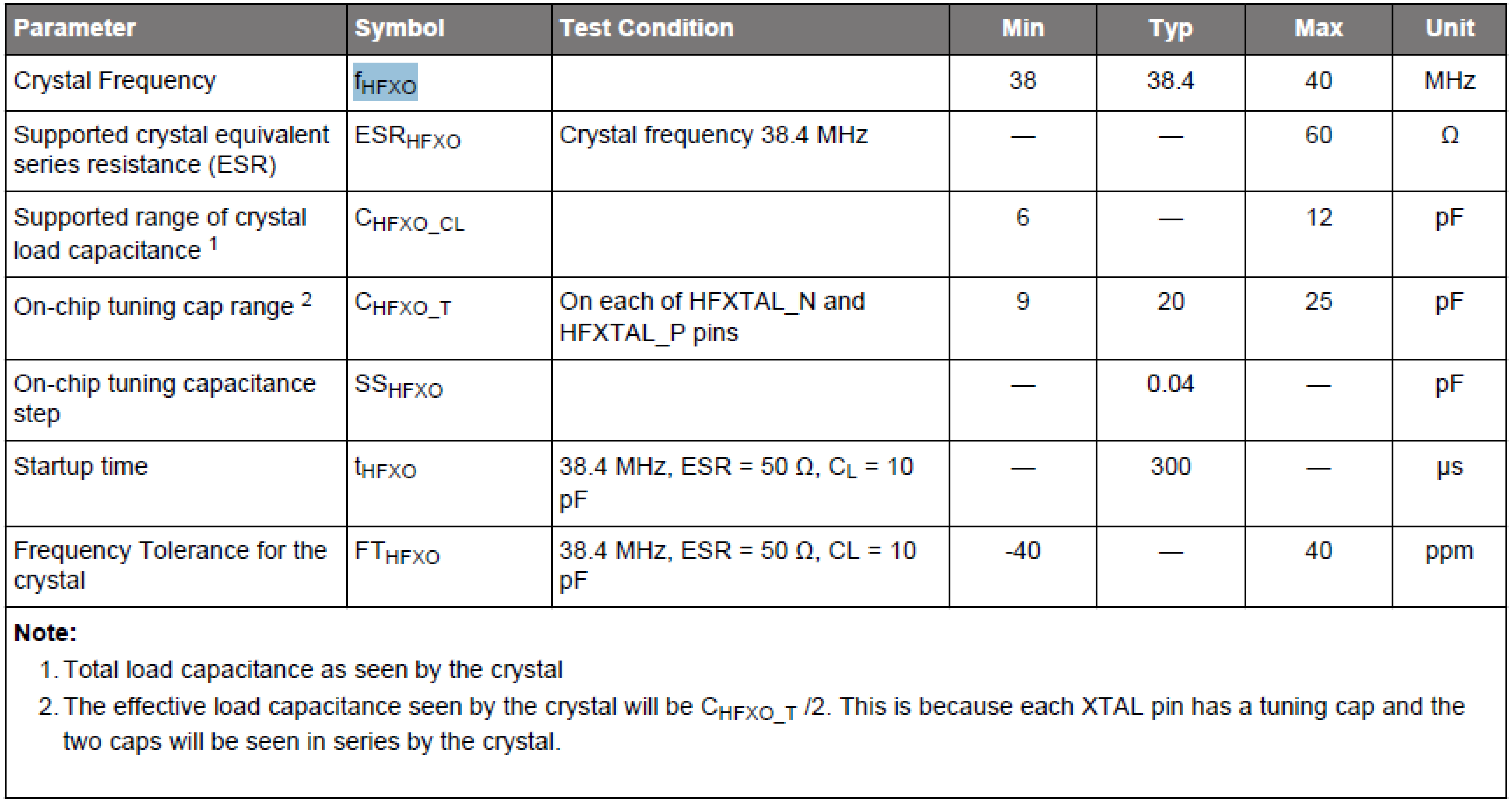
```
void CMU_HFXOAutostartEnable ( uint32_t  userSel,
                               bool      enEM0EM1Start,
                               bool      enEM0EM1StartSel
                             )
```

Enable or disable HFXO autostart.

**Parameters**

| | | |
|---|---|---|
| [in] | **userSel** | Additional user specified enable bit. |
| [in] | **enEM0EM1Start** | If true, HFXO is automatically started upon entering EM0/EM1 entry from EM2/EM3. HFXO selection has to be handled by the user. If false, HFXO is not started automatically when entering EM0/EM1. |
| [in] | **enEM0EM1StartSel** | If true, HFXO is automatically started and immediately selected upon entering EM0/EM1 entry from EM2/EM3. Note that this option stalls the use of HFSRCCLK until HFXO becomes ready. If false, HFXO is not started or selected automatically when entering EM0/EM1. |

# HFXO StartUp Time



| Parameter | Symbol | Test Condition | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| Crystal Frequency | $f_{HFXO}$ | | 38 | 38.4 | 40 | MHz |
| Supported crystal equivalent series resistance (ESR) | $ESR_{HFXO}$ | Crystal frequency 38.4 MHz | — | — | 60 | Ω |
| Supported range of crystal load capacitance [1] | $C_{HFXO\_CL}$ | | 6 | — | 12 | pF |
| On-chip tuning cap range [2] | $C_{HFXO\_T}$ | On each of HFXTAL_N and HFXTAL_P pins | 9 | 20 | 25 | pF |
| On-chip tuning capacitance step | $SS_{HFXO}$ | | — | 0.04 | — | pF |
| Startup time | $t_{HFXO}$ | 38.4 MHz, ESR = 50 Ω, $C_L$ = 10 pF | — | 300 | — | µs |
| Frequency Tolerance for the crystal | $FT_{HFXO}$ | 38.4 MHz, ESR = 50 Ω, CL = 10 pF | -40 | — | 40 | ppm |

**Note:**

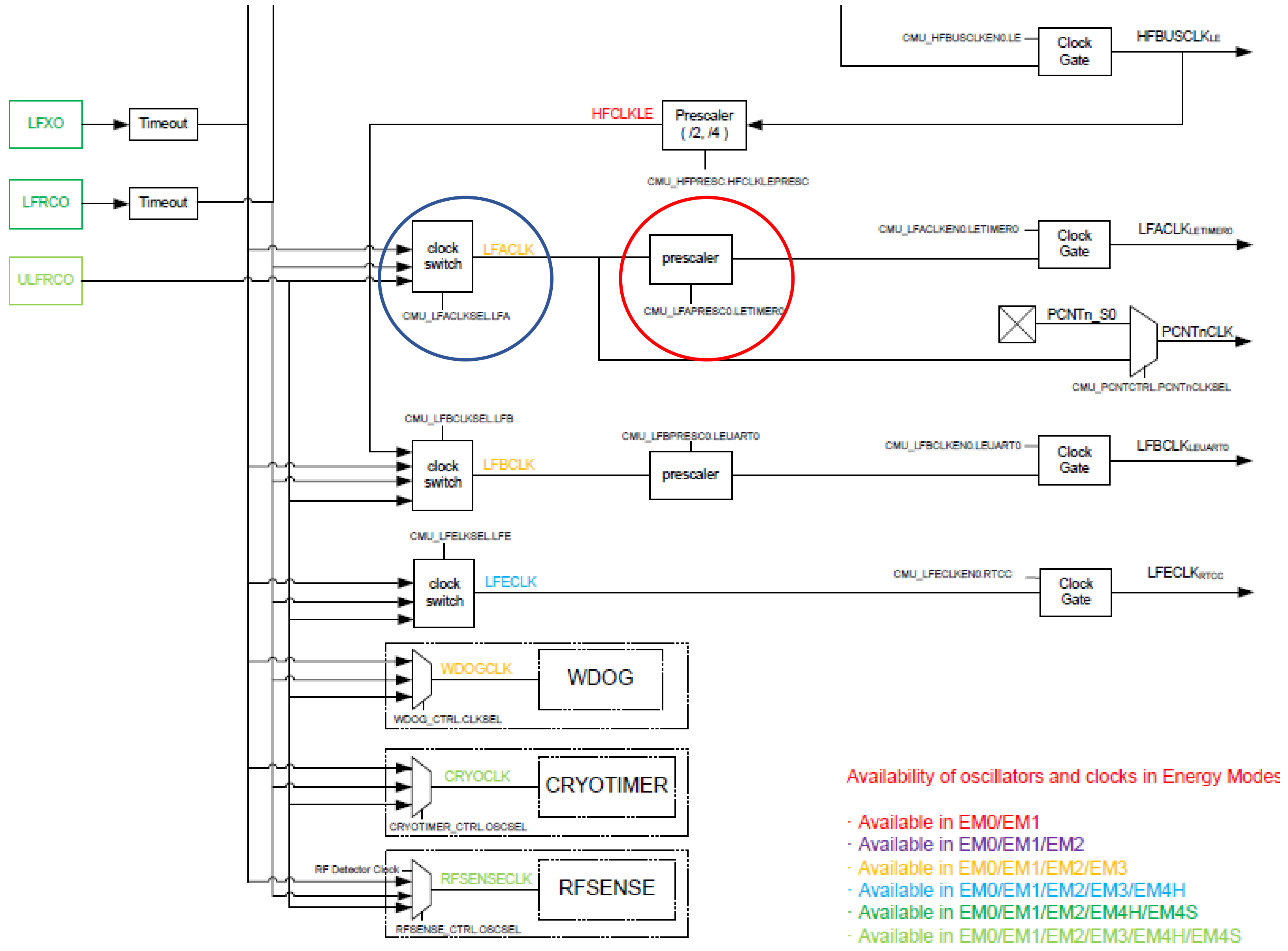
1. Total load capacitance as seen by the crystal
2. The effective load capacitance seen by the crystal will be $C_{HFXO\_T}$ /2. This is because each XTAL pin has a tuning cap and the two caps will be seen in series by the crystal.

# Enabling a peripheral – Clock Source

- A clock source must be selected to the peripheral clock tree branch
- emlib routine to enable/disable an oscillator:
  - CMU_OscillatorEnable(CMU_Osc_TypeDef osc, bool enable, bool wait);
    - Setting wait to true, this routine will not return to the program until the clock source has been stabilized
- emlib routine to select a clock source:
  - CMU_ClockSelectSet(CMU_Clock_TypeDef clock, CMU_Select_TypeDef ref)
    - The clock parameter is one of the clock branches (HF, LFA, LFB)
    - Ref is one of the clock sources (HFRCO, HFXO, LFRCO, LFXO, HFCORECLK/2, ULFRCO)

# Blue Gecko Low Frequency Clock Tree

## Prescaler

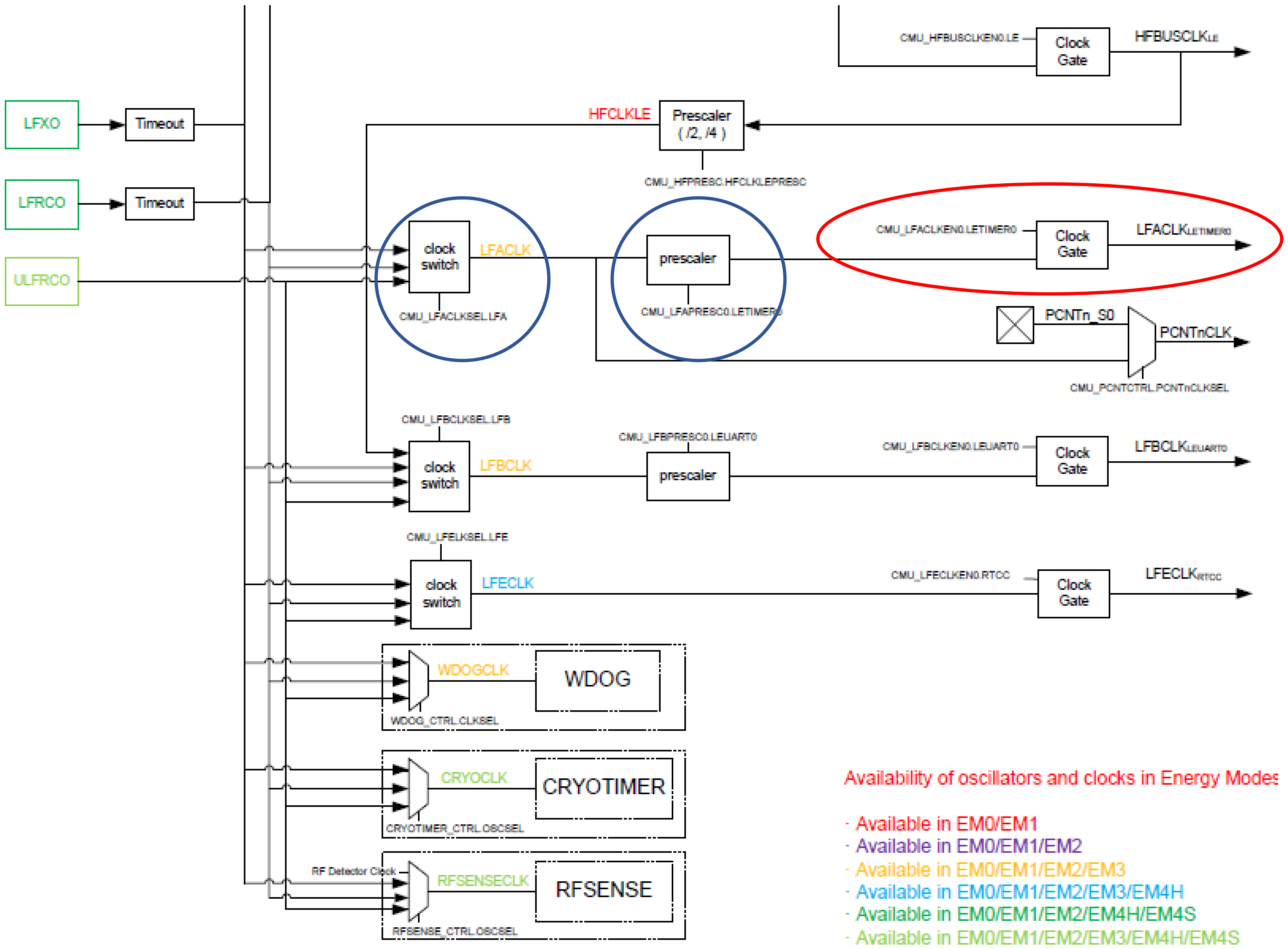


## Prescaling

- emlib to set the clock branch prescaler
  - CMU_ClockDivSet(CMU_Clock_TypeDef clock, CMU_ClkDiv_TypeDef div)
  - Note that not all clocks can be prescaled or prescaled to the same level

## Blue Gecko Low Frequency Clock Tree

## Enabling the peripheral clock

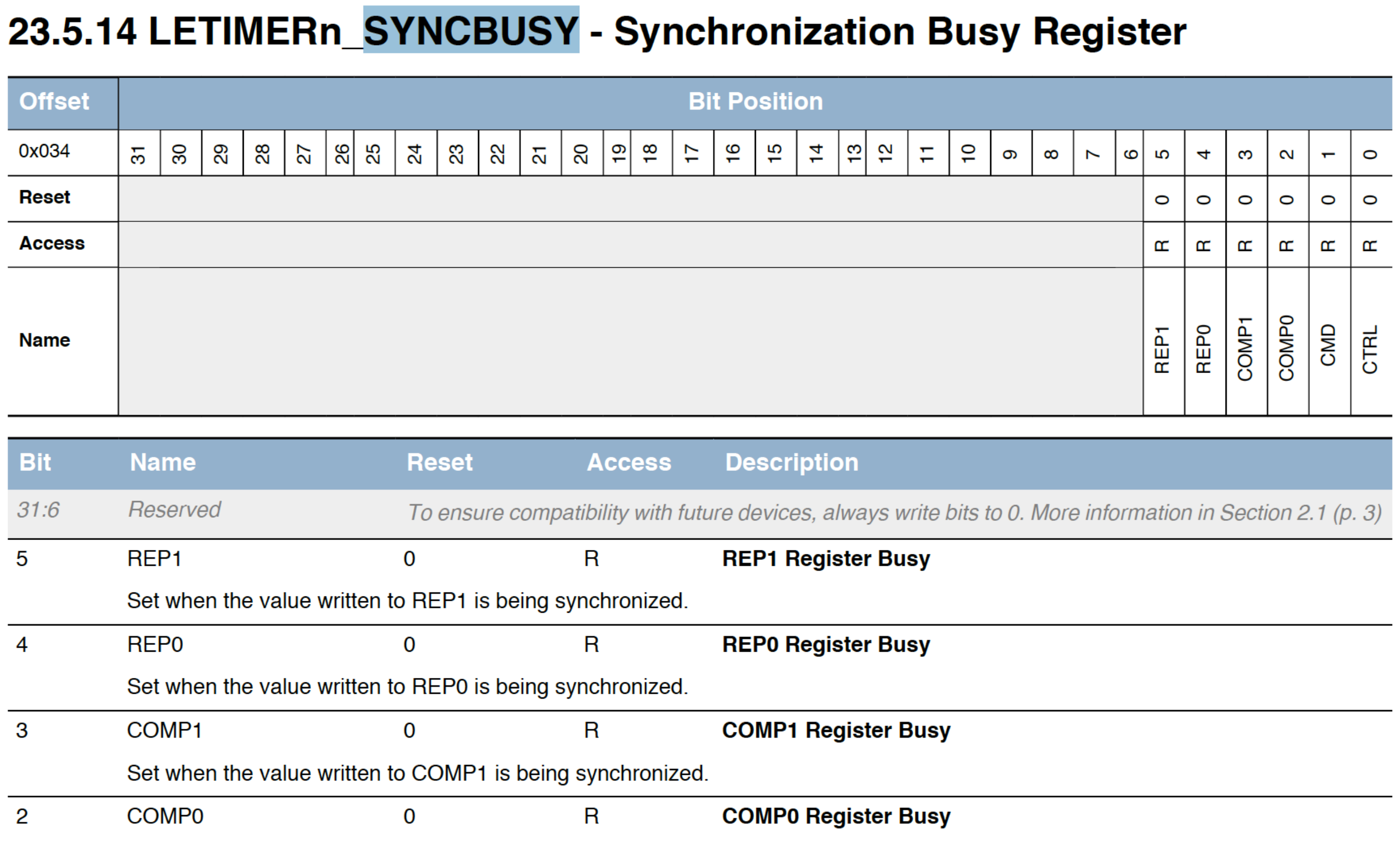


## Peripheral clocks

- Based on the idea of a Low Energy micro controller, all peripherals have their clocks turned-off at reset. Each peripheral clock needs to be enabled individually before initializing and using the peripheral
- emlib routine to enable a peripherals clock:
  - CMU_ClockEnable(CMU_Clock_TypeDef clock, bool enable)
    - Clock is the peripheral to be clocked and true to enable the clock
- Note: For Low Energy peripherals, the LE clock for all LE peripherals need to be enabled using the same emlib function above

# Peripheral clocks

- Since the Low Energy clocks are running at a much slower and possibly an asynchronous clock to the CPU HFCORE clock, writing data to the LE register may need to wait for any previous write to complete or to be synchronized



**23.5.14 LETIMERn_SYNCBUSY - Synchronization Busy Register**

| Offset | Bit Position | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x034 | 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 | 0 |
| **Access** | | | | | | | | | | | | | | | | | | | | | | | | | | | R | R | R | R | R | R |
| **Name** | | | | | | | | | | | | | | | | | | | | | | | | | | | REP1 | REP0 | COMP1 | COMP0 | CMD | CTRL |

| Bit | Name | Reset | Access | Description |
|---|---|---|---|---|
| *31:6* | *Reserved* | *To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)* | | |
| 5 | REP1 | 0 | R | **REP1 Register Busy** |
| | Set when the value written to REP1 is being synchronized. | | | |
| 4 | REP0 | 0 | R | **REP0 Register Busy** |
| | Set when the value written to REP0 is being synchronized. | | | |
| 3 | COMP1 | 0 | R | **COMP1 Register Busy** |
| | Set when the value written to COMP1 is being synchronized. | | | |
| 2 | COMP0 | 0 | R | **COMP0 Register Busy** |

Silicon Labs' App note AN0004
Silicon Labs' EFM3LG reference manual

## Energy Modes

- To save energy, the Blue Gecko can be placed in an appropriate energy state for the current activity requirements
  - EM0: run mode
  - EM1: sleep mode
  - EM2: deep sleep mode
  - EM3: stop mode
  - EM4: shutoff

# EM0 – run mode

- This is the default mode. In this mode the CPU fetches and executes instructions from flash or RAM, and all peripherals may be enabled.
    - Cortex-M4 is executing code and consuming as little as 211uA/MHz when running code from flash.
    - High and low frequency clock trees are active
    - All peripheral functionality is available
    - Consuming as little as 130 µA/MHz (In low power mode)
        - Equated to ~4.992 mA @ 38.4MHz

# EM1 – sleep mode

- The highest consuming peripheral is asleep.  What is the highest energy consuming peripheral?
- All peripherals, as well as RAM and Flash are available. The EFM32 has extensive support for operation in this mode. For example, the timer may repeatedly trigger an ADC conversion at a given instant. When the conversion is complete, the result is moved by the DMA to RAM. When a given number of conversions have been performed, the DMA may wake up the CPU using an interrupt.
  - MCU clock tree is inactive
  - Cortex-M4 is in sleep mode, not executing instructions.  Clocks to the CPU gated off
  - High and low frequency clock trees are active
  - All peripheral functionality is available
  - Current consumption is only 65 µA/MHz
    - Equates to 2.496 mA @ 38.4MHz

# EM2 – deep sleep mode

- This is the first level into the low power energy modes. Most of the high frequency peripherals are disabled or have reduced functionality.  Memory and registers retain their values.
  - Cortex-M4 is in sleep mode. Clocks to the core are off
  - High frequency clock tree is inactive
  - Low frequency clock tree are still active
  - The following low frequency peripherals are available
    - LCD, RTC, LETIMER, PCNT, LEUART, I2C, LESENSE, OPAMP, USB, WDOG and ACM
  - How does the Blue Gecko wakeup back to EM0?
    - Peripheral interrupt, reset pin, power on reset, asynchronous pin interrupt, I2C address recognition, or ACMP edge interrupt
  - How does the Blue Gecko wakeup back to EM1?
    - DMA request
    - Part returns to EM2 Deep Sleep when transfers are complete
  - RAM and register values are preserved
  - Current consumption as low as 3.3 µA typically

# EM3 – stop mode

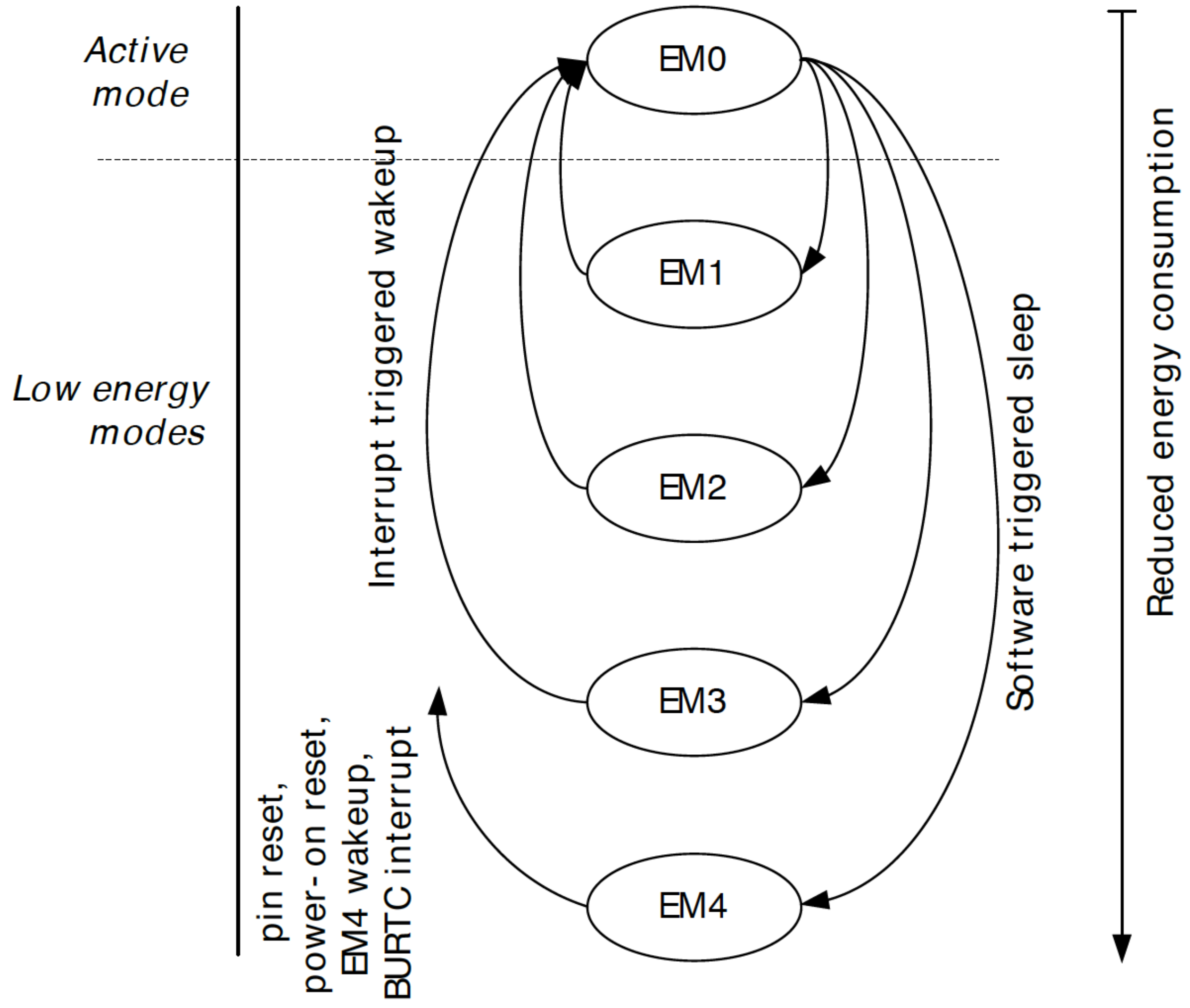
- This low energy mode has both high frequency and low frequency clocks stopped. Most peripherals are disabled or have reduced functionality. Memory and registers retain their values.
  - Cortex-M4 is in sleep mode. Clocks to the core are off
  - High frequency clock tree is inactive
  - What else becomes inactive?
    - Low frequency clock tree are inactive
  - The following low frequency peripherals are available
    - ACMP, asynchronous external interrupt, PCNT, and I2C can wake-up the device
  - Wakeup to EM0 Active through
    - Peripheral interrupt, reset pin, power on reset, asynchronous pin interrupt, I2C address recognition, or ACMP edge interrupt
  - RAM and register values are preserved
  - Current consumption is only 2.8 µA typically

## EM4 – Shutoff

- EM4 Shutoff is the lowest energy mode of the part. There is no retention except for GPIO PAD state upon register set. Wakeup from EM4 Shutoff requires a reset to the system, returning it back to EM0 Active
  - The following is the only functionality available in EM4
    - pin reset
    - GPIO pin wake-up
    - GPIO pin retention
    - Backup RTC (including retention RAM)
    - Power-On Reset
    - All pins are put into their reset state unless specified to retain state in EM4
  - Current is down to 0.65 uA typically

**Figure 10.2. EMU Energy Mode Transitions**

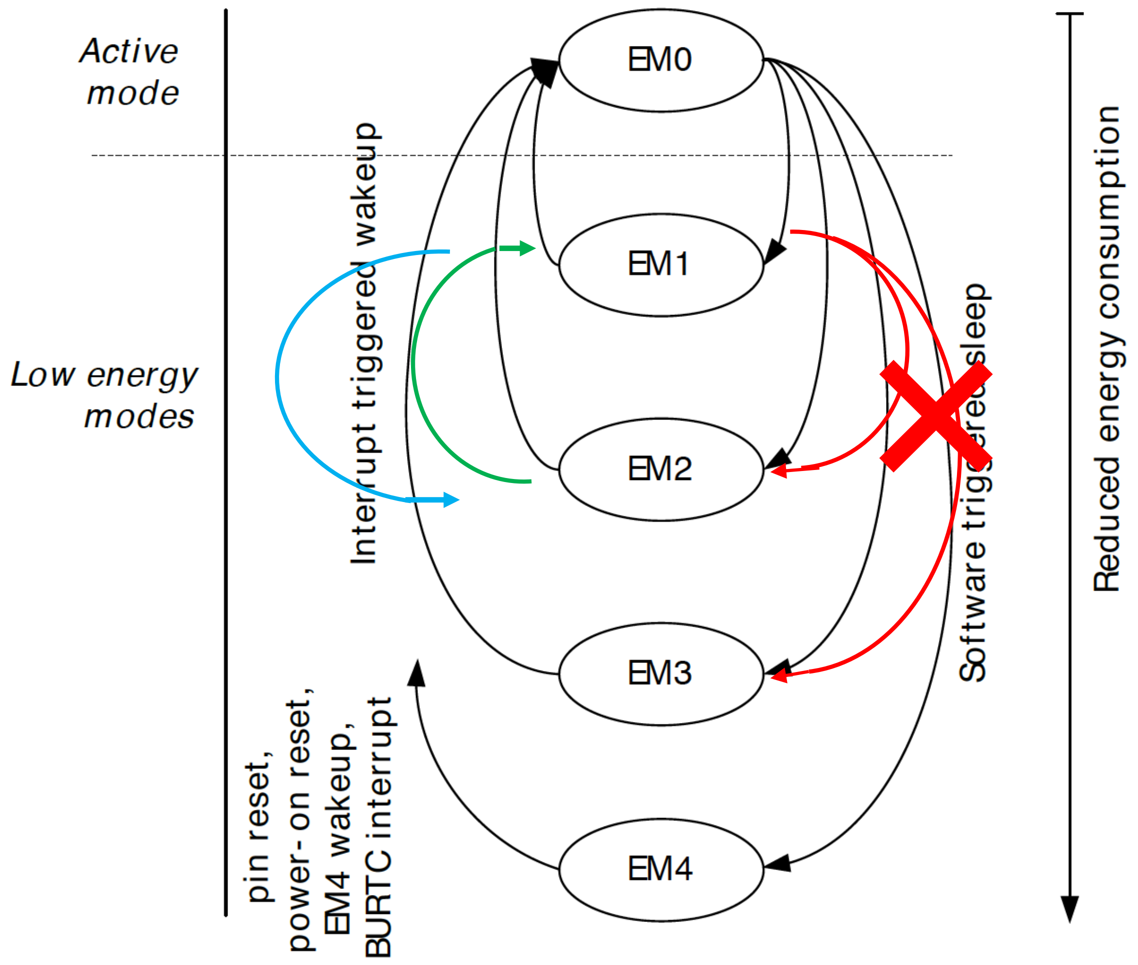
# Silicon Lab's standard Energy Mode flow chart



Silicon Labs' EFM32LG Reference Manual

# Blue Gecko special Energy Mode flows

- The Low Energy UART, LEUART, can change DMA states from EM2 to EM1 to enable DMA transfers
  - Once the DMA transfer is completed, the system will go back to EM2
- The ADC can run as low as EM3 and wake up the DMA to pull data out of its FIFOs


Active mode
Low energy modes
EM0
EM1
EM2
EM3
EM4
Interrupt triggered wakeup
Software triggered sleep
Reduced energy consumption
pin reset, power-on reset, EM4 wakeup, BURTC interrupt

## Managing Blue Gecko’s energy mode

- Managing which energy mode the Blue Gecko can enter based on which peripheral is active by creating a sleep() routine
- What information would the sleep() routine require to know which Energy Mode to go to?
- Each active peripheral signifies its lowest active energy state by setting a global variable / count using the below routine:
  - blockSleepMode(EMx); where x is 0-4 indicating first low power state not to go into
- Each active peripheral is responsible to release its block on an energy state when it becomes no longer active
  - unblockSleepMode(EMx); where x is 0-4 indicating first low power state not to go into

# blockSleepMode();

```
/** Block the microcontroller from sleeping below a certain mode
 *
 * This will block sleep() from entering an energy mode below the one given.
 * -- To be called by peripheral HAL's --
 *
 * After the peripheral is finished with the operation, it should call unblock with the same state
 *
 */
void blockSleepMode(sleepstate_enum minimumMode)
{
  CORE_ATOMIC_IRQ_DISABLE();();
  sleep_block_counter[minimumMode]++;
  CORE_ATOMIC_IRQ_ENABLE();
}
```

# unblockSleepMode();

```
/** Unblock the microcontroller from sleeping below a certain mode
 *
 * This will unblock sleep() from entering an energy mode below the one given.
 * -- To be called by peripheral HAL's --
 *
 * This should be called after all transactions on a peripheral are done.
 */
void unblockSleepMode(sleepstate_enum minimumMode)
{
   CORE_ATOMIC_IRQ_DISABLE();
   if(sleep_block_counter[minimumMode] > 0) {
      sleep_block_counter[minimumMode]--;
   }
   CORE_ATOMIC_IRQ_ENABLE();
}
```

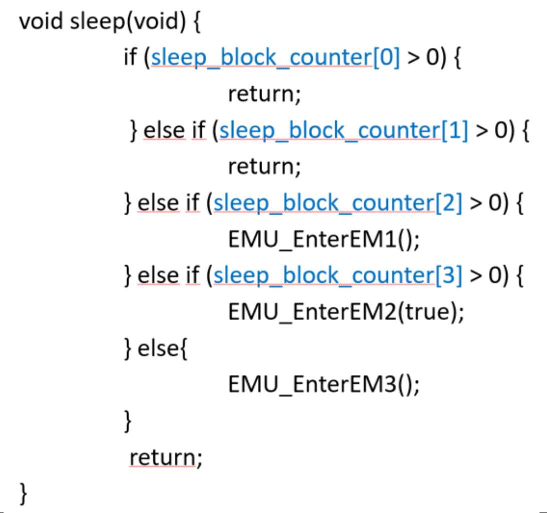
# sleep();

```
void sleep(void) {
        if (sleep_block_counter[0] > 0) {
                return;                                  // stay in EM0
        } else if (sleep_block_counter[1] > 0) {
                return;                                  // EM1 is blocked, so go into EM0
        } else if (sleep_block_counter[2] > 0) {
                EMU_EnterEM1();                          // EM2 is blocked, so go into EM1
        } else if (sleep_block_counter[3] > 0) {
                EMU_EnterEM2(true);                      // EM3 is blocked, so go into EM2
        } else{
                EMU_EnterEM3();                          // Don‘t go into EM4
        }
        return;
}
```

# Example pseudo code outline

```
void peripheral_call() {
        blockSleepMode(EMx);
        peripheral routine …
        enable peripheral_call_interrupt;
}

void peripheral_IRQHandler() {
        disable peripheral_call_interrupt;
        peripheral interrupt routine …
        unblockSleepMode(EMx);
}

int main() {
        CHIP_Init();
        peripheral initialization routine();

        peripheral_call();

        while(1) {
                sleep();
        }

}
```

```
void blockSleepMode(sleepstate_enum
minimumMode)
{
   CORE_ATOMIC_IRQ_DISABLE();
   sleep_block_counter[minimumMode]++;
   CORE_ATOMIC_IRQ_ENABLE();
}

void unblockSleepMode(sleepstate_enum minimumMode)
{
   CORE_ATOMIC_IRQ_DISABLE();
   if(sleep_block_counter[minimumMode] > 0) {
      sleep_block_counter[minimumMode]--;
   }
   CORE_ATOMIC_IRQ_ENABLE();
}
```



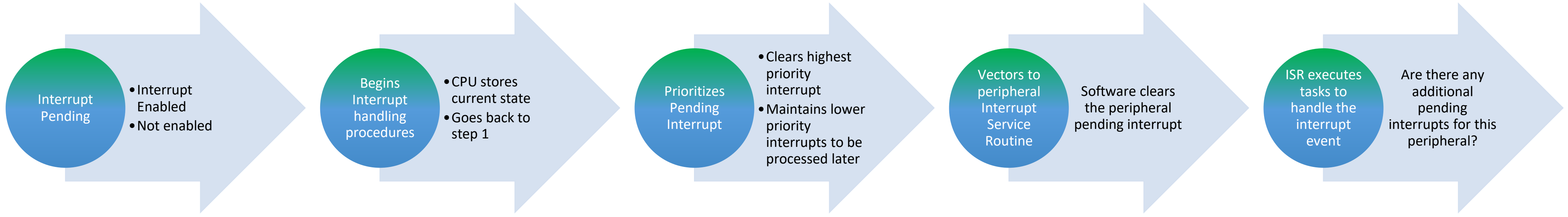
```
void sleep(void) {
        if (sleep_block_counter[0] > 0) {
                return;
        } else if (sleep_block_counter[1] > 0) {
                return;
        } else if (sleep_block_counter[2] > 0) {
                EMU_EnterEM1();
        } else if (sleep_block_counter[3] > 0) {
                EMU_EnterEM2(true);
        } else{
                EMU_EnterEM3();
        }
        return;
}
```

# Energy Optimization - Interrupts

- Polling with while-loops can be a useful way to halt CPU processing at a certain stage in a program until a certain condition has been met such as waiting for an oscillator to stabilize or for incoming data on a UART connection. However, a while loop where the CPU continuously checks for a certain condition is not very power efficient.
- Interrupts allows the CPU to go sleep while a condition is waiting to be met such as getting a result from the ADC which is very energy efficient.

# ARM Cortex-M4 Interrupt process

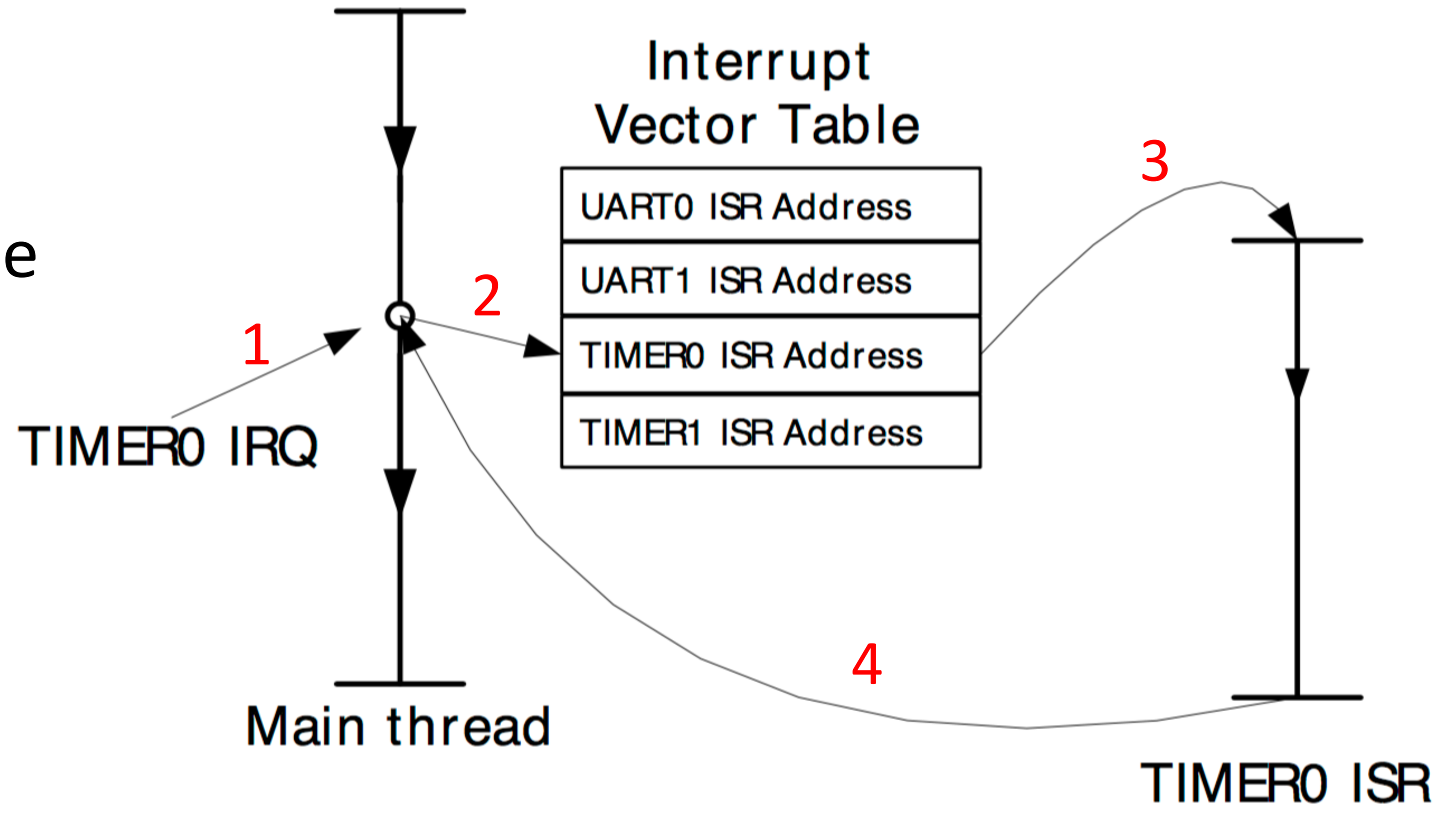
- Similar to the generic interrupt process, but there are multiple specific ISRs available
- These ISRs more efficient direct the controller to the required Interrupt Handle



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?

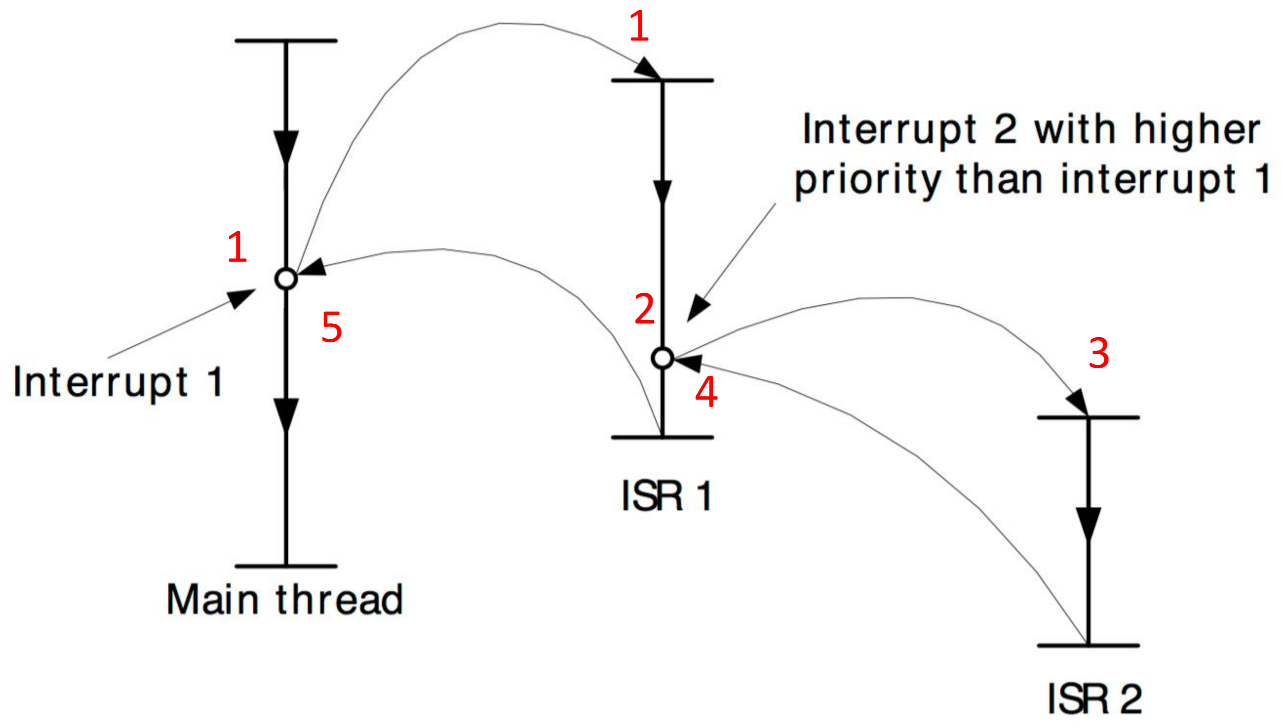
# ARM Cortex-M4

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



Silicon Labs' App note AN0039

# 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

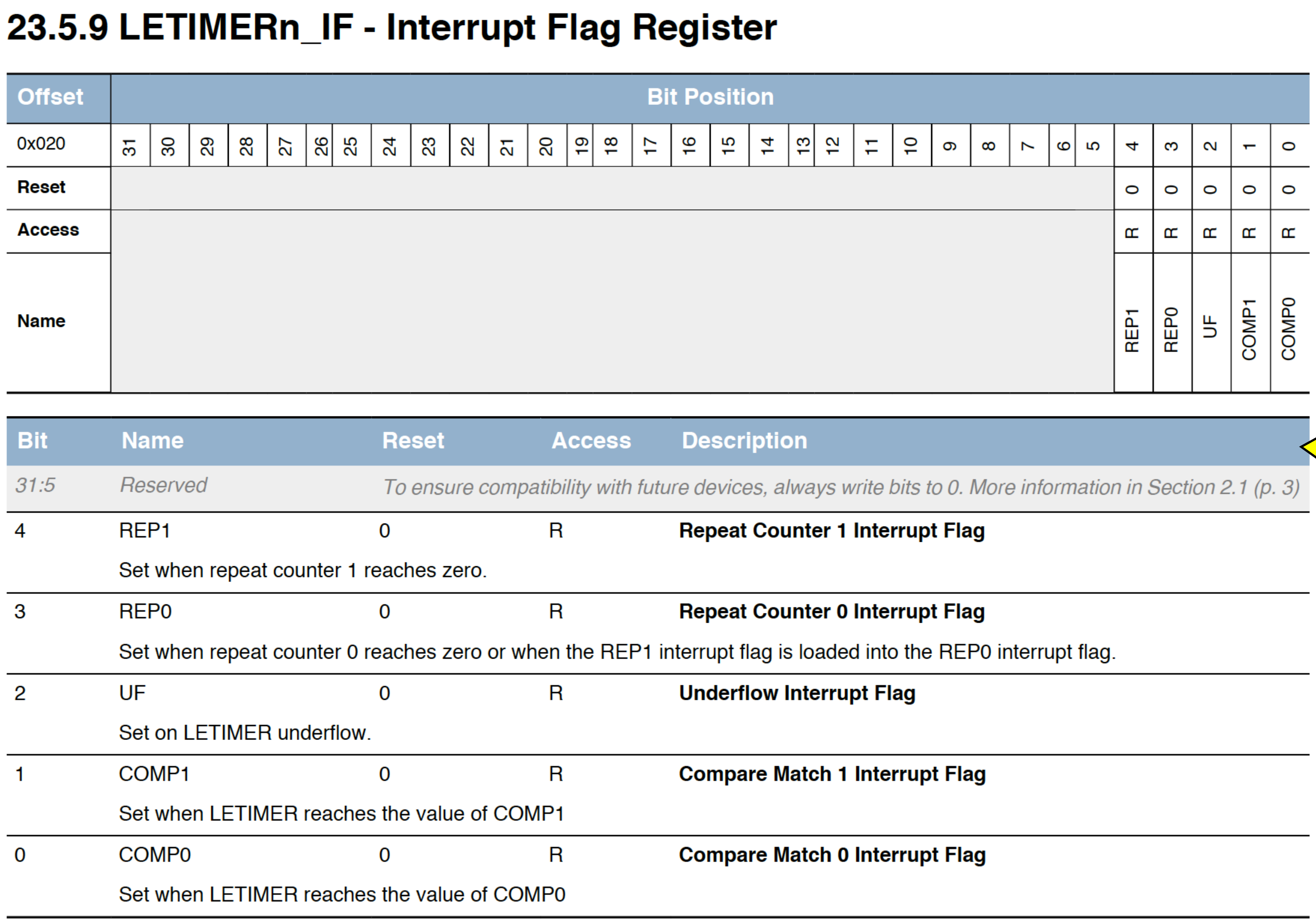
```
__isr_vector:
    .long    __StackTop            /* Top of Stack */
    .long    Reset_Handler         /* Reset Handler */
    .long    NMI_Handler           /* NMI Handler */
    .long    HardFault_Handler     /* Hard Fault Handler */
    .long    MemManage_Handler     /* MPU Fault Handler */
    .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

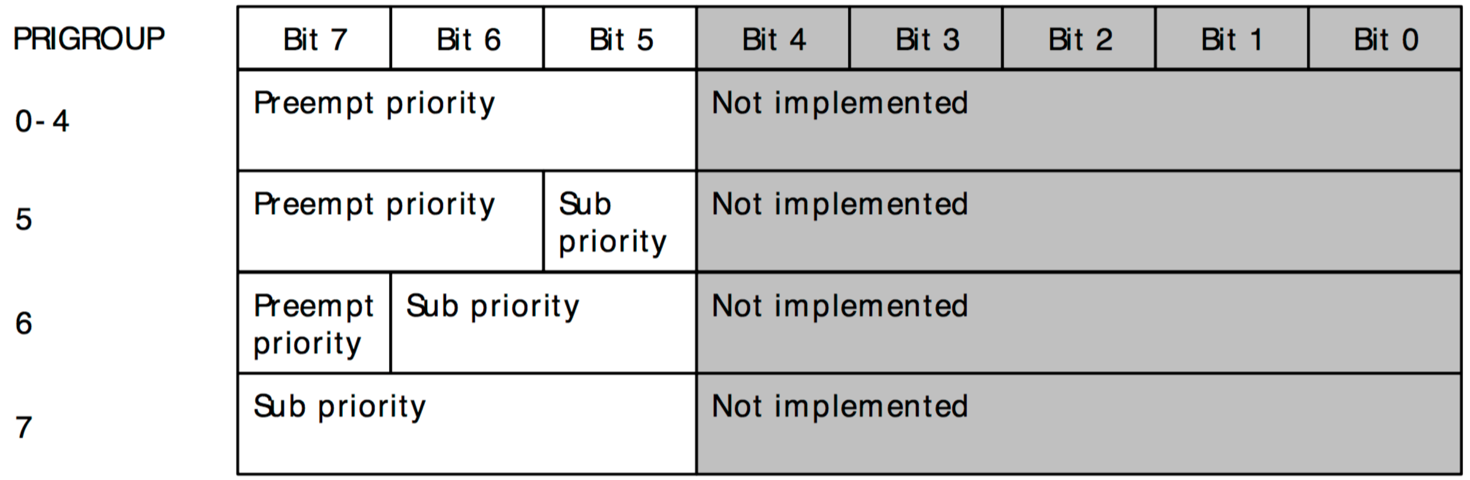
## Interrupt priority

- 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)
- What must we do to make an interrupt service routine, ISR, atomic?
  - All interrupts disabled, CORE_ATOMIC_IRQ_DISABLE();
  - Atomic operation
  - All interrupts renabled, CORE_ATOMIC_IRQ_ENABLE();

## Best practices in ISR code writing

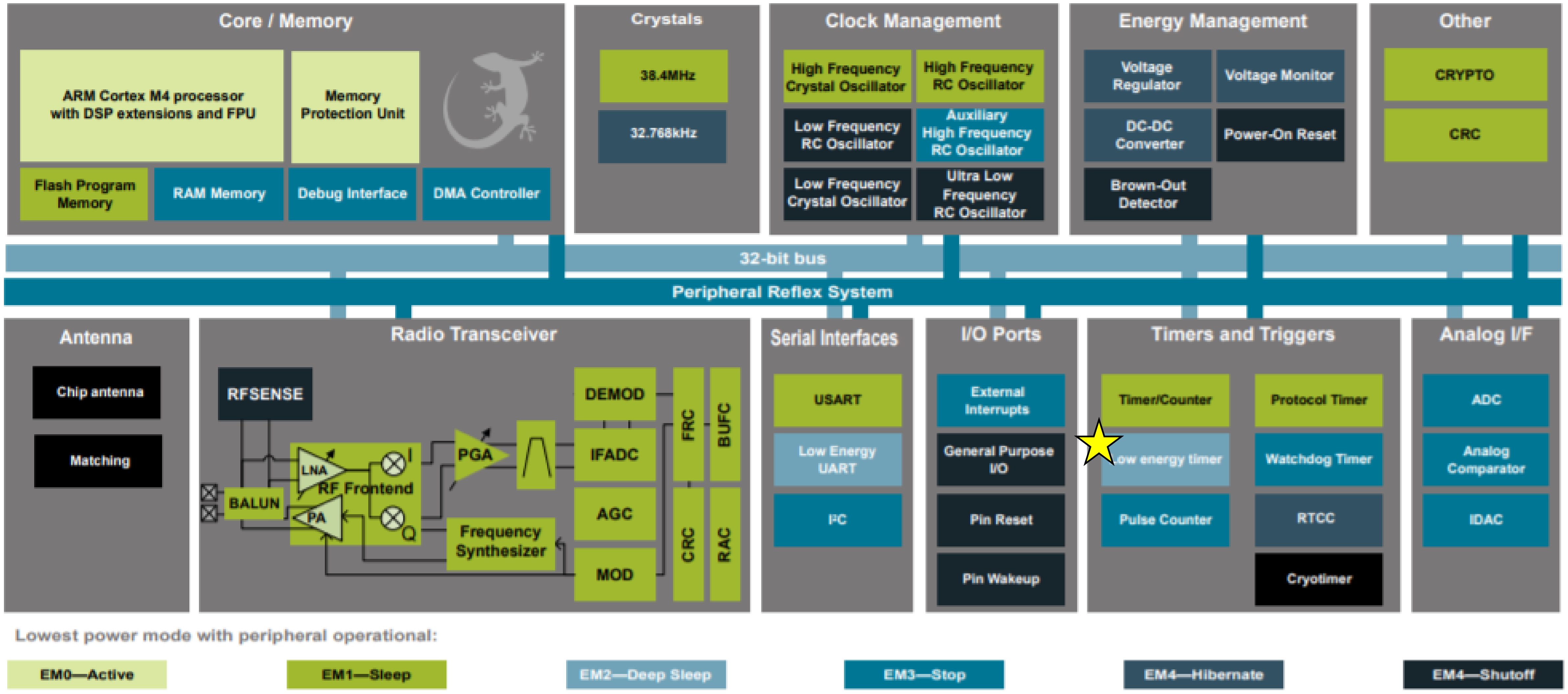
Clear pending interrupts immediately in the ISR so that if another interrupt occurs, you will not be clearing an interrupt that has not been processed

```
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

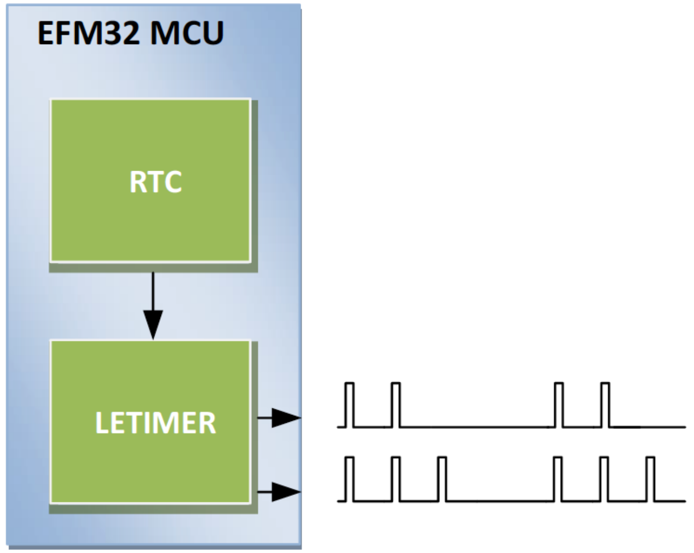
```
void peripheral_IRQHandler() {
        int intFlags;
        CORE_ATOMIC_IRQ_DISABLE();
        intFlags = Peripheral_IntGet(Peripheral); //determine pending interrupts
        Peripheral_IntClear(Peripheral, intFlags);
        /*ISR handling code based on interrupts set in intFlag*/
        CORE_ATOMIC_IRQ_DISABLE();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


Core / Memory
ARM Cortex M4 processor with DSP extensions and FPU
Memory Protection Unit
Flash Program Memory
RAM Memory
Debug Interface
DMA Controller
Crystals
38.4MHz
32.768kHz
Clock Management
High Frequency Crystal Oscillator
High Frequency RC Oscillator
Low Frequency RC Oscillator
Auxiliary High Frequency RC Oscillator
Low Frequency Crystal Oscillator
Ultra Low Frequency RC Oscillator
Energy Management
Voltage Regulator
Voltage Monitor
DC-DC Converter
Power-On Reset
Brown-Out Detector
Other
CRYPTO
CRC
32-bit bus
Peripheral Reflex System
Antenna
Chip antenna
Matching
Radio Transceiver
RFSENSE
DEMOD
IFADC
AGC
MOD
FRC
BUFC
CRC
RAC
PGA
LNA
RF Frontend
BALUN
PA
I
Q
Frequency Synthesizer
Serial Interfaces
USART
Low Energy UART
I²C
I/O Ports
External Interrupts
General Purpose I/O
Pin Reset
Pin Wakeup
Timers and Triggers
Timer/Counter
Protocol Timer
Low energy timer
Watchdog Timer
Pulse Counter
RTCC
Cryotimer
Analog I/F
ADC
Analog Comparator
IDAC
Lowest power mode with peripheral operational:
EM0—Active
EM1—Sleep
EM2—Deep Sleep
EM3—Stop
EM4—Hibernate
EM4—Shutoff

## 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)

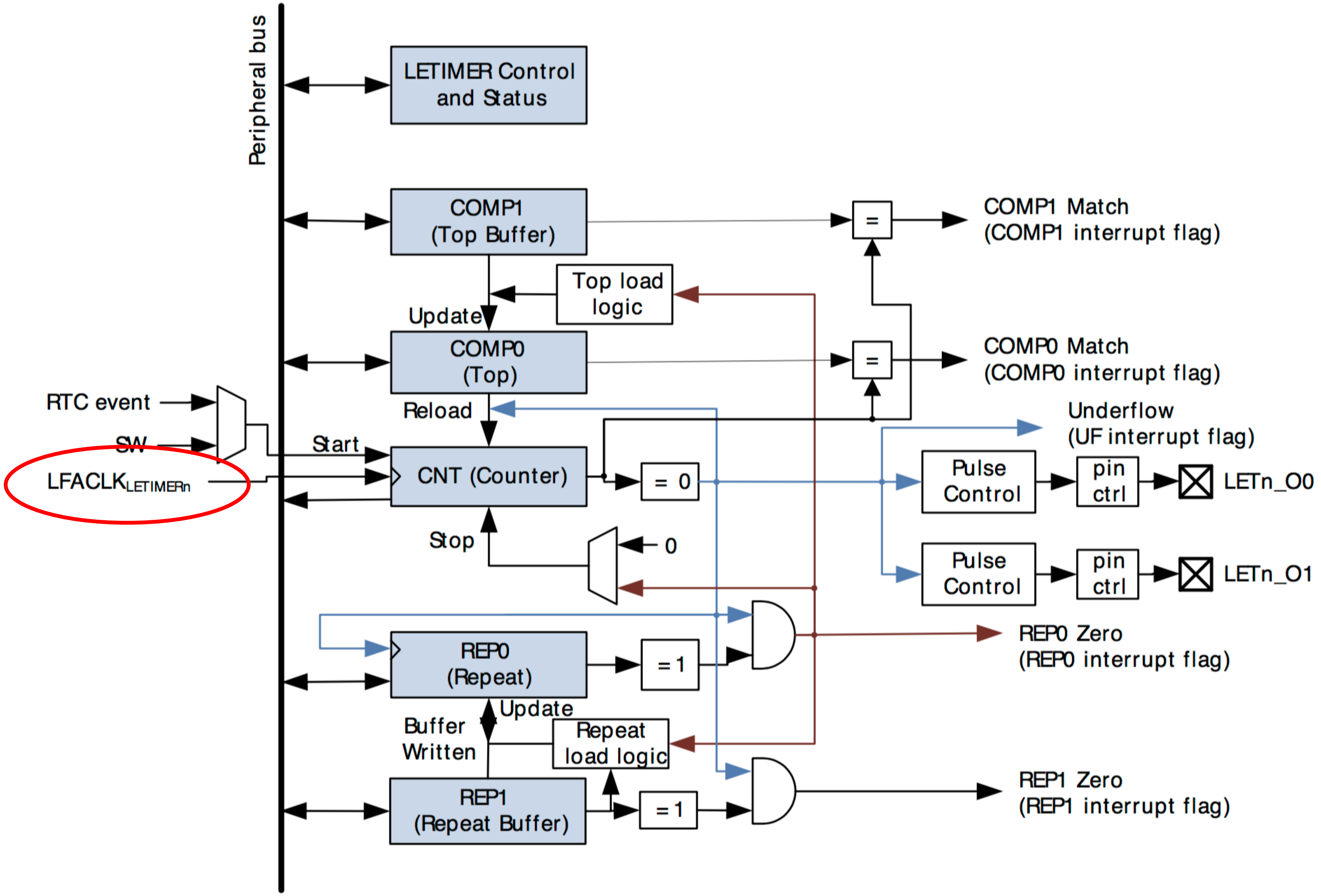


Blue Gecko - Available down in EM3 using the ULFRC0

# 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



Peripheral bus
LETIMER Control and Status
COMP1 (Top Buffer)
Top load logic
Update
COMP0 (Top)
Reload
RTC event
SW
LFACLKLETIMERn
Start
CNT (Counter)
Stop
0
= 0
=
=
COMP1 Match (COMP1 interrupt flag)
COMP0 Match (COMP0 interrupt flag)
Underflow (UF interrupt flag)
Pulse Control
pin ctrl
LETn_O0
Pulse Control
pin ctrl
LETn_O1
REP0 (Repeat)
= 1
REP0 Zero (REP0 interrupt flag)
Update
Buffer Written
Repeat load logic
REP1 (Repeat Buffer)
= 1
REP1 Zero (REP1 interrupt flag)