ECEN 5023-001, -001B, -740

Mobile Computing & IoT Security

Lecture #3

24 January 2017

Be Boulder.

University of Colorado Boulder

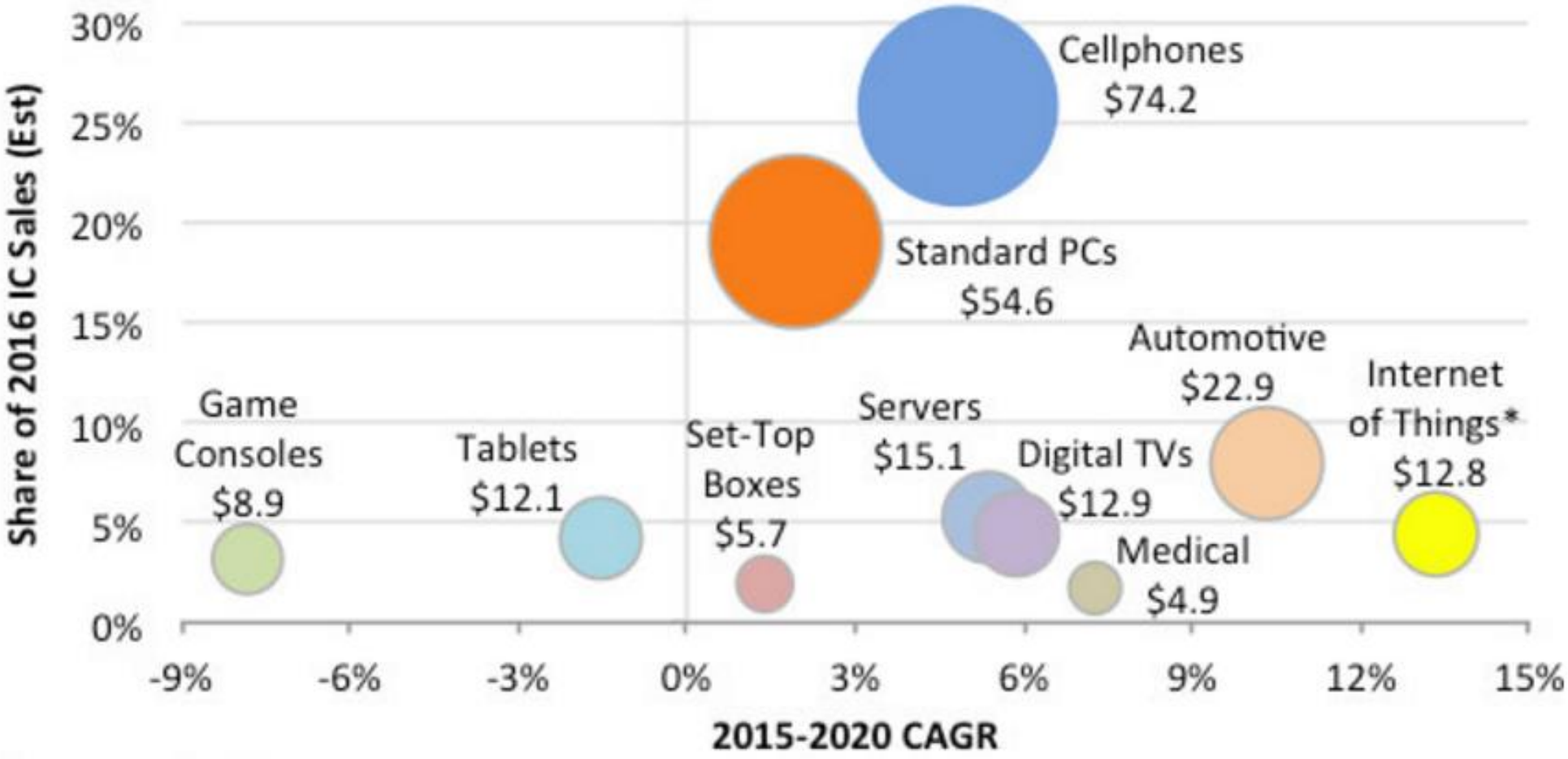
Agenda

- Class Announcements
- Quiz 1 review
- Energy Modes
- Keeping Track of the Energy State
- Interrupts
- LETIMER0
- Reading List
- Quiz 2 assigned

Class Announcements

- Simplicity Exercise is due at 11:59pm on Wednesday the 25th, 2017
- Quiz #2 is due at 11:59pm on Sunday, January 29th, 2017
- Register for ESE lab card access https://goo.gl/forms/YaBXQHATHELA2FIk2

IC End-Use Markets ($B) and Growth Rates



*Covers only the Internet connection portion of systems.

Source: IC Insights

Class Survey Results

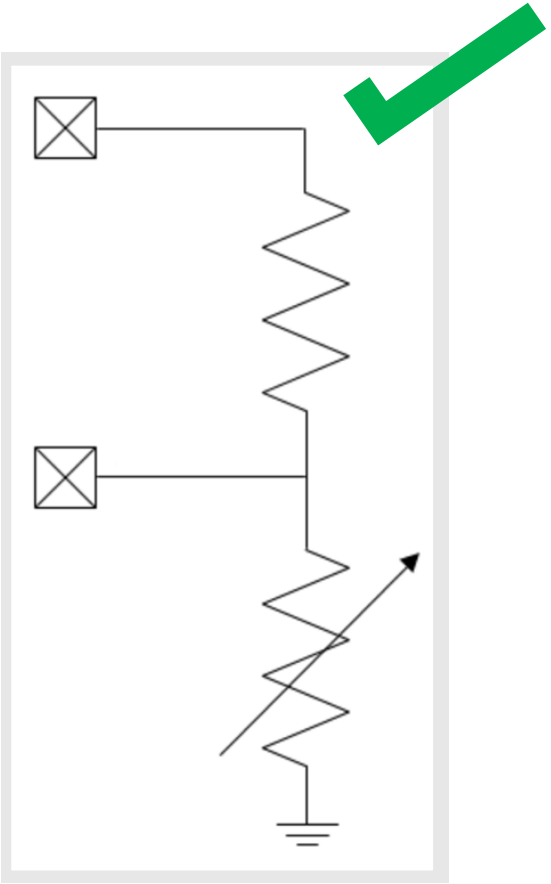
- 18 students responded. I will use the results as a representative of the entire class.
  - Question 1: 39% have taken ESD
  - Question 2: 0% are currently taking ESD
  - Question 3: 50% have taken ESE
  - Question 4: 5% are currently taking ESE
  - Question 5: 94% have used a microcontroller IDE
  - Question 6: 83% have used a microcontroller IDE debugger
  - Question 7: 72% have used a microcontroller IDE register views
  - Question 8: 83% are moderate to expert C programmers
  - Question 9: 50% are moderate C++ programmers
  - Question 10: 22% have beginning to moderate Bluetooth Classic experience
  - Question 11: 28% have beginning to moderate Bluetooth Low Energy experience

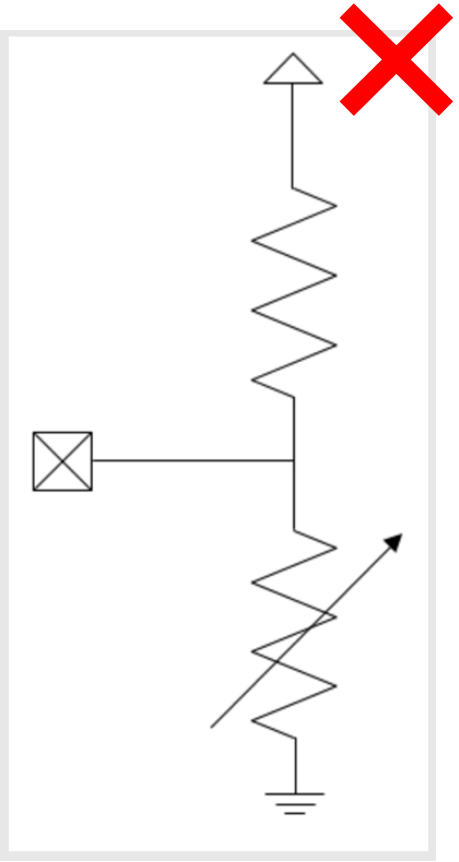
Quiz

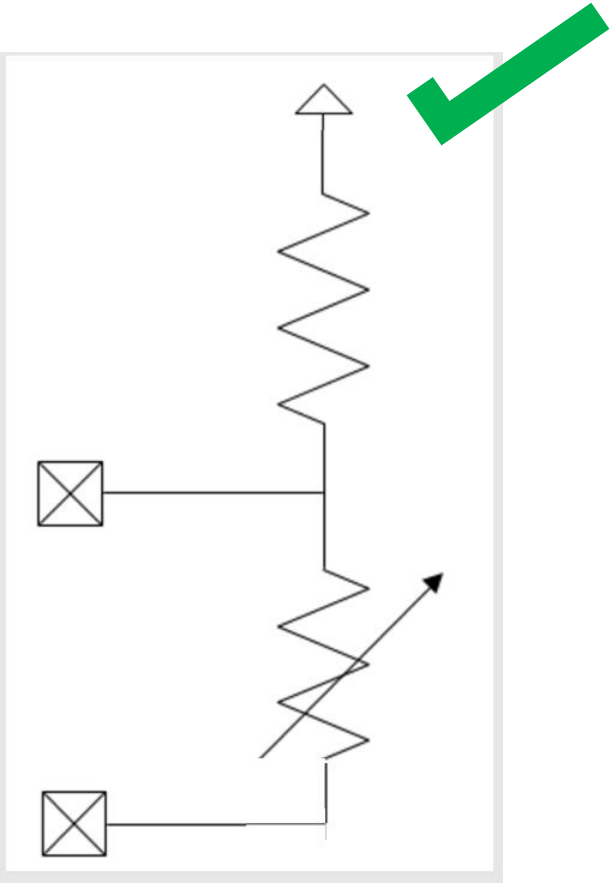
In selecting an energy source, mobility, lifetime , cost, and form factor need to be considered.

Select all possible analog thermistor circuits which can minimize energy by enabling duty cycling of the passive sensor.

Quiz







Quiz

In which Energy Mode of the Leopard Gecko are the Low-Frequency peripherals available?

(All of the particular peripheral must be available in that Energy Mode)

- [x] EM0
- [x] EM1
- [x] EM2
- [ ] EM3
- [ ] EM4

Quiz

In which Energy Mode of the Leopard Gecko are the Asynchronous peripherals available?

(All of the particular peripheral must be available in that Energy Mode)

- [x] EM0
- [x] EM1
- [x] EM2
- [x] EM3
- [ ] EM4

Quiz

In which Energy Mode of the Leopard Gecko are the High-Frequency peripherals available?

(All of the particular peripheral must be available in that Energy Mode)

- [x] EM0
- [x] EM1
- [ ] EM2
- [ ] EM3
- [ ] EM4

Quiz

hardware, peripheral, peripherals

To enable the CPU to be off as much as possible to save energy, the cpu must _____ to the _____ in the microcontroller.

offload, offloaded, offload tasks, offloaded tasks, offload task, offloaded task, be offloaded, delegate task

Quiz

Which system would require the fastest response time?

- Thermostat
- Automated plant waterer
- Water heater
- Heart pacer

Quiz

Which system would require the slowest response time?

- Thermostat
- Automated plant waterer
- Heart pacer
- Water heater

Quiz

Match the application to the most appropriate CPU architecture.

2	Mixed application	1. Cortex-M4
3	Control Logic	2. Cortex-M3
1	Signal processing	3. Cortex-M0

Quiz

A conventional program maintains **control, full control** of the processing sequence from the beginning to the end.

In contrast, event-driven gains control only _____ when handling events.

sporadically, sporadic, temporary, temporarily, occasionally

Quiz

An event alone can not determine the next action in an event-driven embedded system, but _____ is equally important.

context, the context, the current context, current context, state

Quiz

True of false, a state machine increases the different paths through the program code.

- True
- False

Quiz

True of false, a state machine increases the complexity of testing at each branch point.

- ○ True
- ✓ False

Quiz

Match the application to whether its requirements match more of a consumer or industrial IoT device.

1	Exercise watch
2	Home heater
2	Personal weather station
2	Solar panel inverter
2	Insulin pump

1. Consumer IoT
2. Industrial IoT

Quiz

Match the application to whether its requirements match more of a consumer or industrial IoT device.

2	Outdoor video surveillance camera
2	Airplane engine control
1	Baby monitor
2	Train track monitor
1	Smart light bulb

1. Consumer IoT
2. Industrial IoT

Quiz

Write the C-code to add the Vertical Front Porch Interrupt Enable, VFPORCH, ofthe External Bus Interface, EBI, peripheral interrupt enable register, IEN.

(EBI->IEN |= EBI_IEN_VFPORCH;, EBI->IEN |=EBI_IEN_VFPORCH;, EBI->IEN|=
EBI_IEN_VFPORCH;, EBI->IEN|=EBI_IEN_VFPORCH;, EBI->IEN |= 0x03;, EBI->IEN |=0x03;, EBI->IEN|= 0x03;, EBI->IEN|=0x03;)

Quiz

Write the C-code to remove the Vertical Front Porch Interrupt Enable, VFPORCH, of the External Bus Interface, EBI, peripheral interrupt enable register, IEN.

(EBI->IEN &= ~EBI_IEN_VFPORCH;, EBI->IEN &=~EBI_IEN_VFPORCH;,

EBI->IEN&= ~EBI_IEN_VFPORCH;, EBI->IEN&=~EBI_IEN_VFPORCH;, EBI->IEN &= ~0x03;, EBI->IEN &=~0x03;, EBI->IEN&= ~0x03;,

, EBI->IEN&=~0x03;, EBI->IEN &= 0xffffff7;, EBI->IEN &=0xffffff7;, EBI->IEN&= 0xffffff7;,

EBI->IEN&=0xffffff7;)

Drone/Bot for Responsive Residence Hall Monitoring

Bhallaji
bhve0693@colorado.edu

Mounika
moed4346@colorado.edu

Virag
viga7710@colorado.edu

Drone fall 2016 project

Energy Modes

- To save energy, the Leopard 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-M3 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 211 µA/MHz
    - Equated to ~3.0 mA @ 14MHz

EM1 – sleep mode

- In Sleep Mode the clock to the CPU is disabled. 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-M3 is in sleep mode, not executing instructions. Clocks to the core are off
  - High and low frequency clock trees are active
  - All peripheral functionality is available
  - Current consumption is only 63 µA/MHz
    - Equates to 0.9 mA @ 14MHz

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-M3 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
  - Wakeup to EM0 Active through
    - Peripheral interrupt, reset pin, power on reset, asynchronous pin interrupt, I2C address recognition, or ACMP edge interrupt
  - Wakeup to EM1 Sleep through
    - DMA request
    - Part returns to EM2 Deep Sleep when transfers are complete
  - RAM and register values are preserved
  - Current consumption as low as 0.95 µA with RTC enabled
    - Energy Profile should see ~1.0 – 1.4 uA while in this mode

EM3 – stop mode

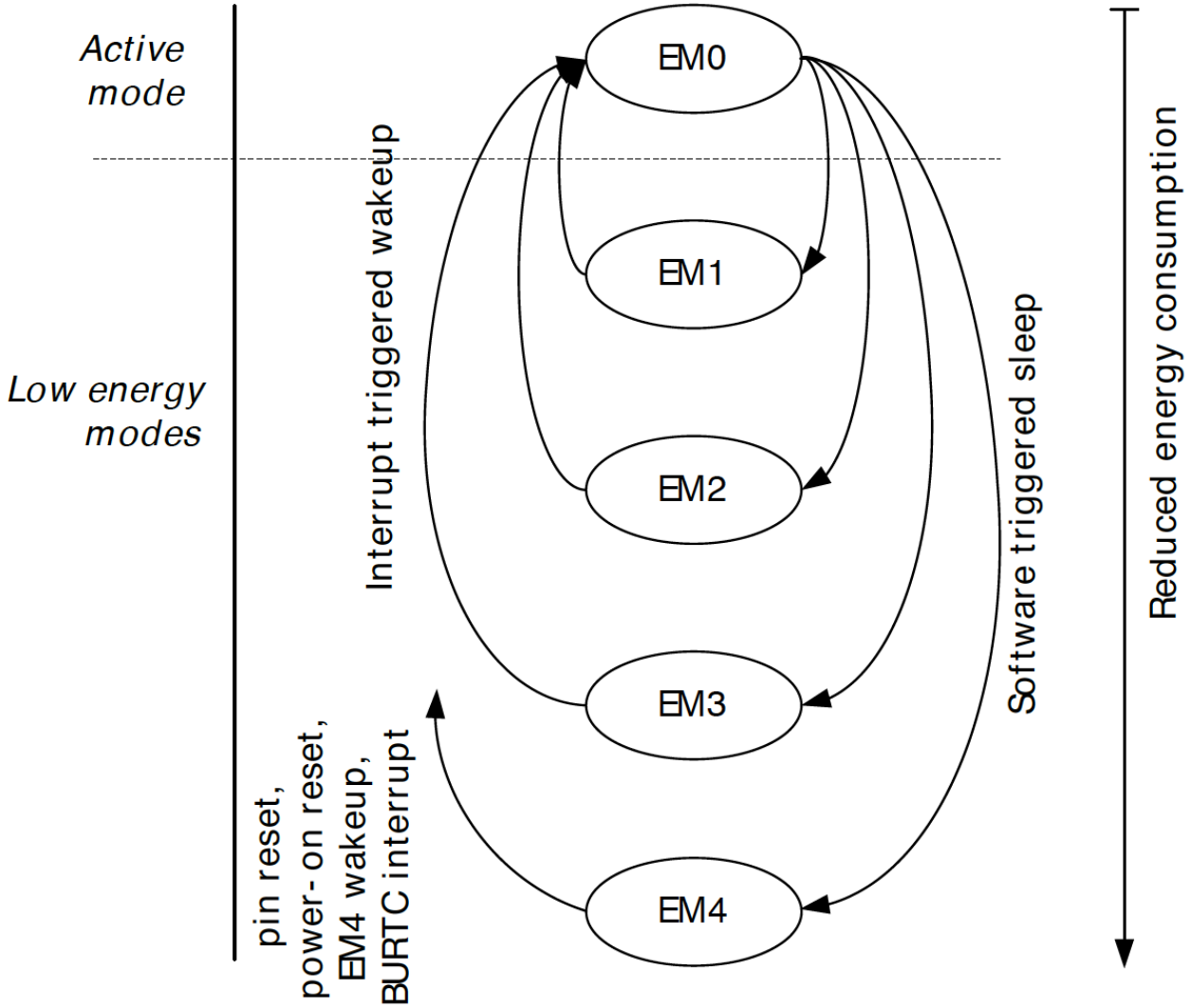
- This low energy mode has both high frequency and low frequency clocks stopped. Most peripherals are disabled or have reduced functionality. Memory and registers retain their values.
  - Cortex-M3 is in sleep mode. Clocks to the core are off
  - High frequency clock tree is inactive
  - Low frequency 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 0.65 µA
    - Energy Profile should see < 1.0 uA while in this mode

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 20 nA

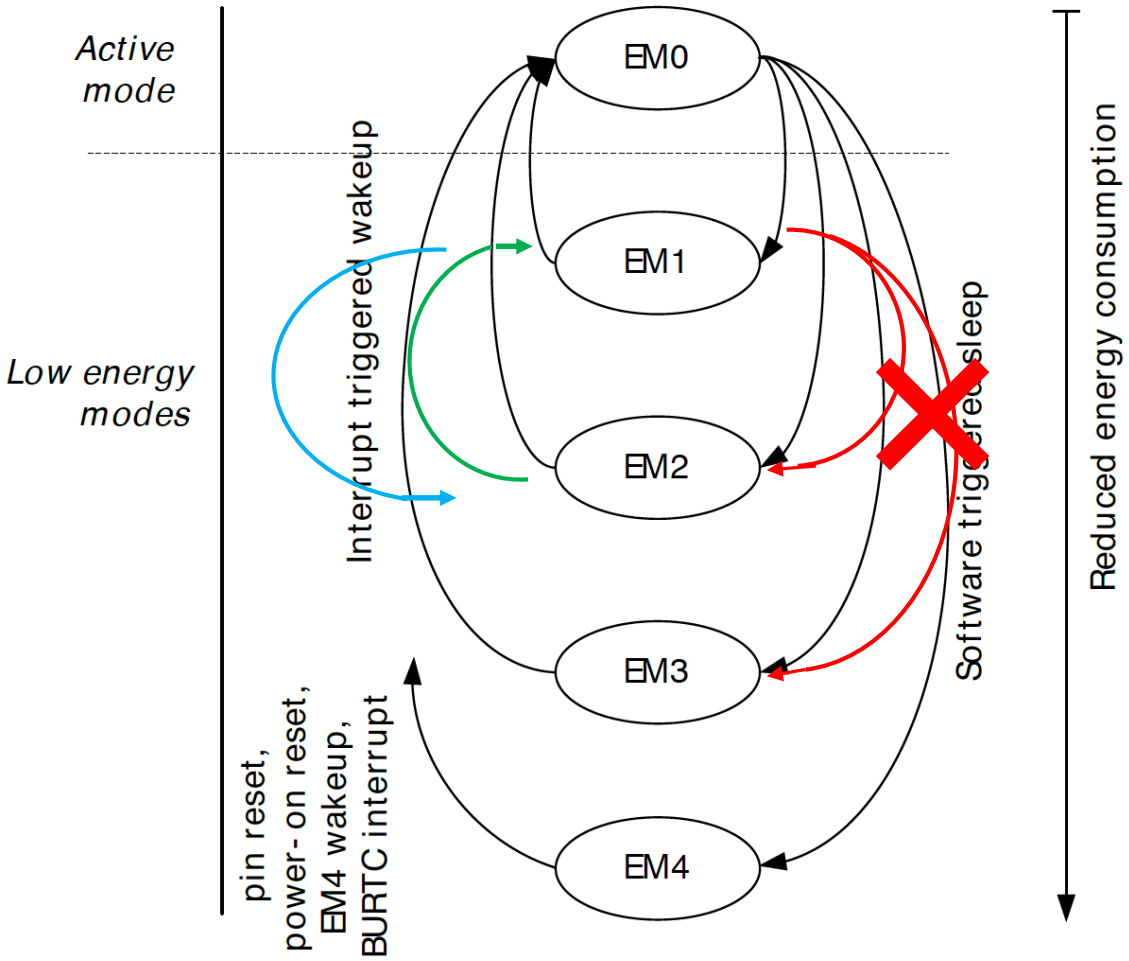
Figure 10.2. EMU Energy Mode Transitions

Silicon Lab's standard Energy Mode flow chart



Leopard 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



Managing Leopard Gecko's energy mode

- Managing which energy mode the Leopard Gecko can enter based on which peripheral is active by creating a sleep() routine
- 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 lowest possible active state
- 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 lowest possible active state

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)
{
    INT_Disable();
    sleep_block_counter[minimumMode]++;
    INT_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)
{
    INT_Disable();
    if(sleep_block_counter[minimumMode] > 0) {
        sleep_block_counter[minimumMode]--;
    }
    INT_Enable();
}
```

sleep();

```
void sleep(void) {
        if (sleep_block_counter[0] > 0) {
                return;                         // Blocked everying below EM0, so just return
         } else if (sleep_block_counter[1] > 0) {
                EMU_EnterEM1();                 // Blocked everything below EM1, enter EM1
        } else if (sleep_block_counter[2] > 0) {
                EMU_EnterEM2(true);             // Blocked everything below EM2, enter EM2
        } else if (sleep_block_counter[3] > 0) {
                EMU_EnterEM3(true);             // Blocked everything below EM3, enter EM3
        } else{
                EMU_EnterEM4();                 // Nothing is blocked, enter 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)
{
    INT_Disable();
    sleep_block_counter[minimumMode]++;
    INT_Enable();
}

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

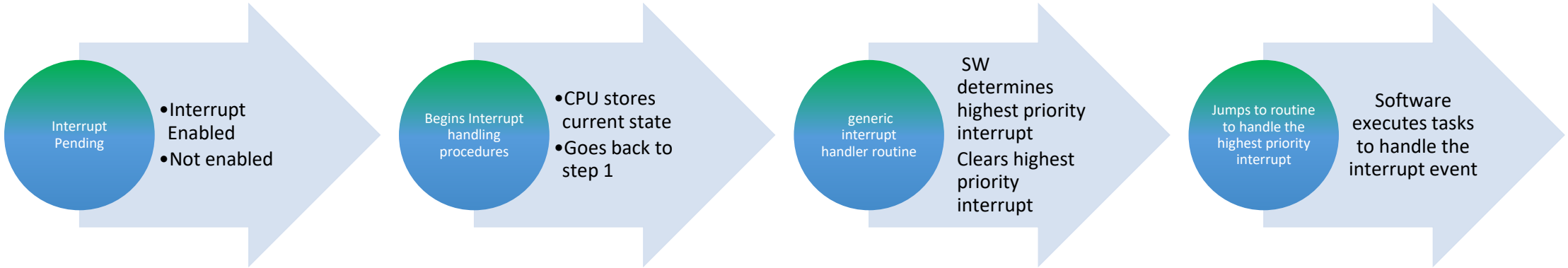
```
void sleep(void) {
        if (sleep_block_counter[0] > 0) {
                return;
        } else if (sleep_block_counter[1] > 0) {
                EMU_EnterEM1();
        } else if (sleep_block_counter[2] > 0) {
                EMU_EnterEM2(true);
        } else if (sleep_block_counter[3] > 0) {
                EMU_EnterEM3(true);
        } else{
                EMU_EnterEM4();
        }
        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.

Interrupt Requests (IRQ)

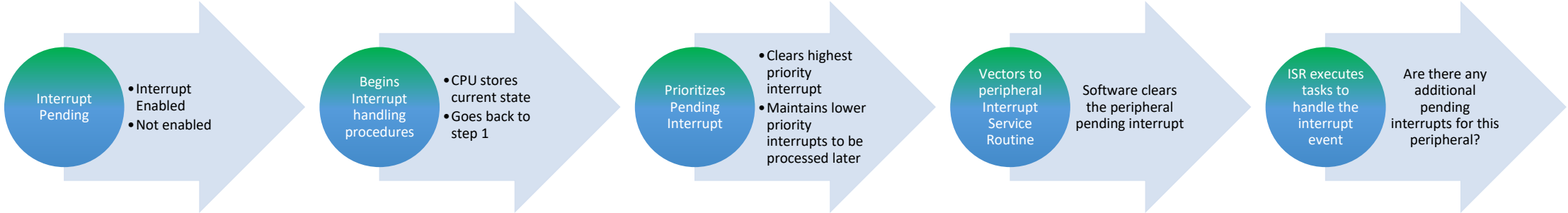
- Generic steps of a controller receiving an interrupt
- Enabled, Prioritize, Vector to Interrupt Service Routine



Silicon Labs' App note AN0039

ARM Cortex-M3 Interrupt process

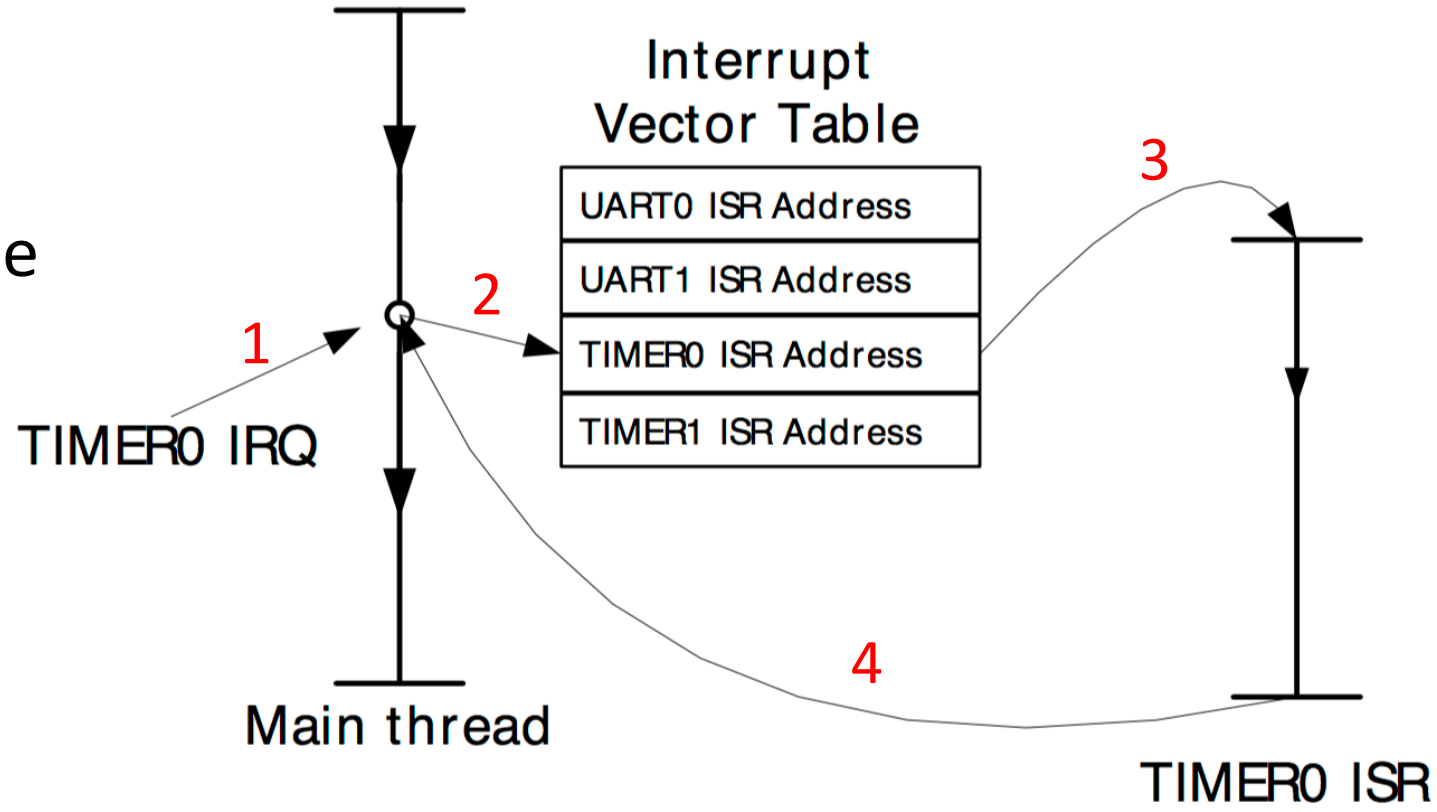
- Similar to the generic interrupt process, but there are multiple specific ISRs available
- These ISRs more efficient direct the controller to the required Interrupt Handle



Silicon Labs' App note AN0039

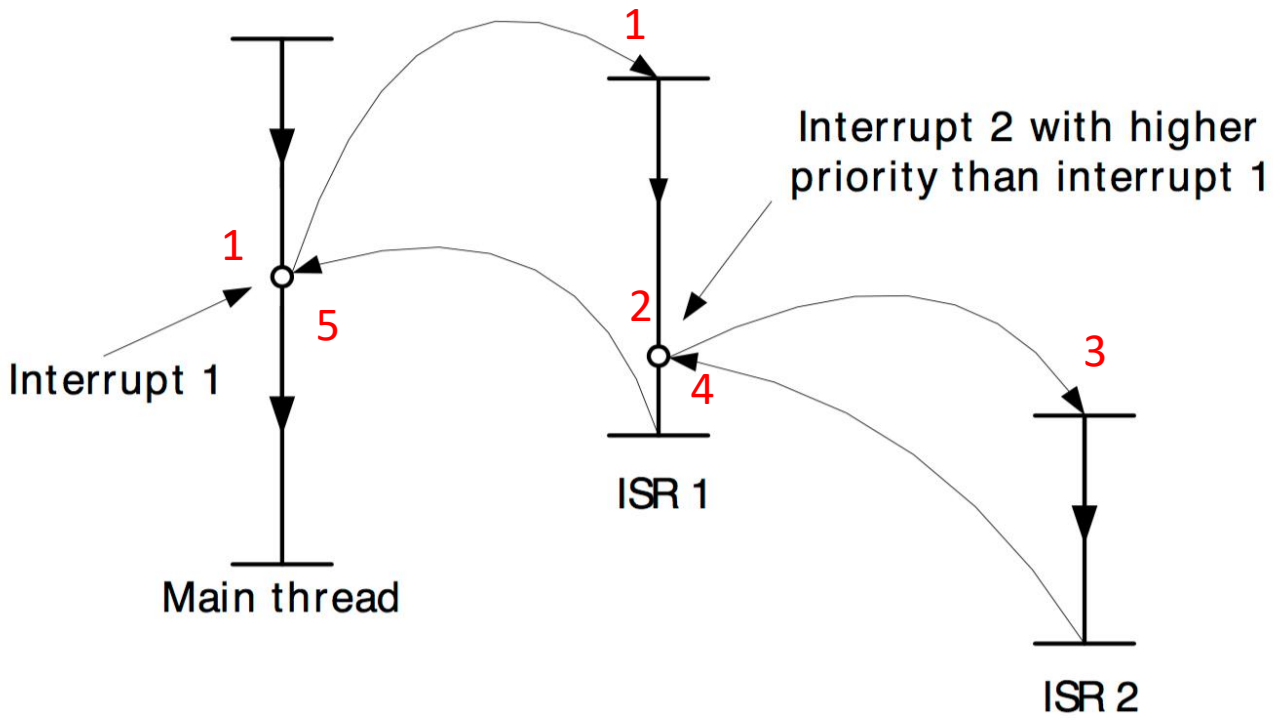
ARM Cortex-M3

Single Interrupt Example



ARM Cortex-M3 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-M3 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-M3 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 Set when repeat counter 1 reaches zero.	0	R	**Repeat Counter 1 Interrupt Flag**
3	REP0 Set when repeat counter 0 reaches zero or when the REP1 interrupt flag is loaded into the REP0 interrupt flag.	0	R	**Repeat Counter 0 Interrupt Flag**
2	UF Set on LETIMER underflow.	0	R	**Underflow Interrupt Flag**
1	COMP1 Set when LETIMER reaches the value of COMP1	0	R	**Compare Match 1 Interrupt Flag**
0	COMP0 Set when LETIMER reaches the value of COMP0	0	R	**Compare Match 0 Interrupt Flag**

- 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, INT_Disable()
  - Atomic operation
  - All interrupts renabled, INT_Enable()

Best practices in ISR code writing

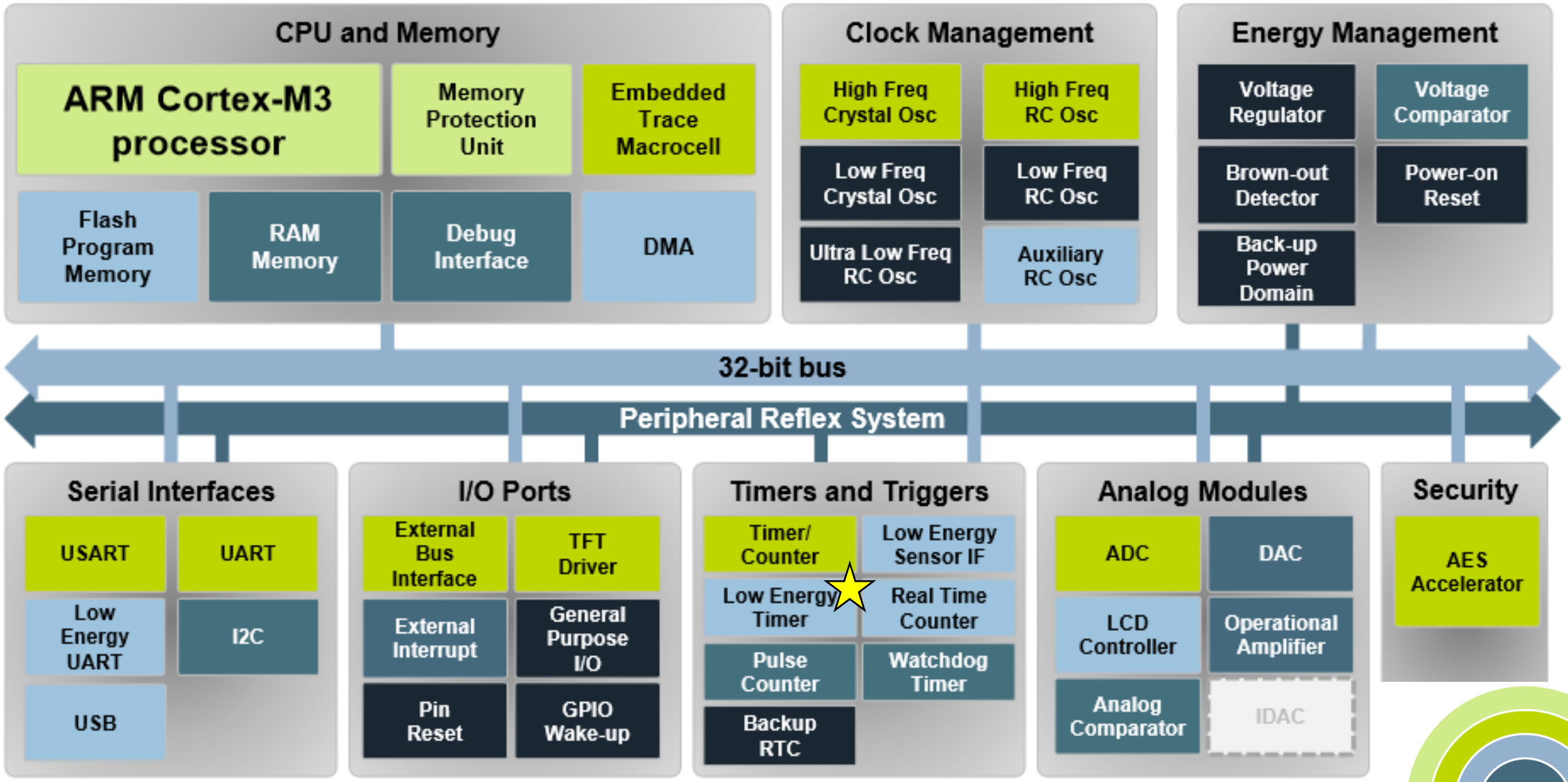
Clear pending interrupts immediate in the ISR so that if another interrupt occurs, you will not be clearing an interrupt that has not been processed

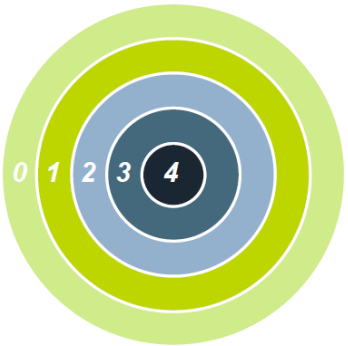
```
void peripheral_IRQHandler() {
    int intFlags;
    intFlags = Peripheral_IntGet(Peripheral); //determine pending interrupts
    Peripheral_IntClear(Peripheral, intFlags);
    /*ISR handling code based on interrupts set in intFlag*/
}
```

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

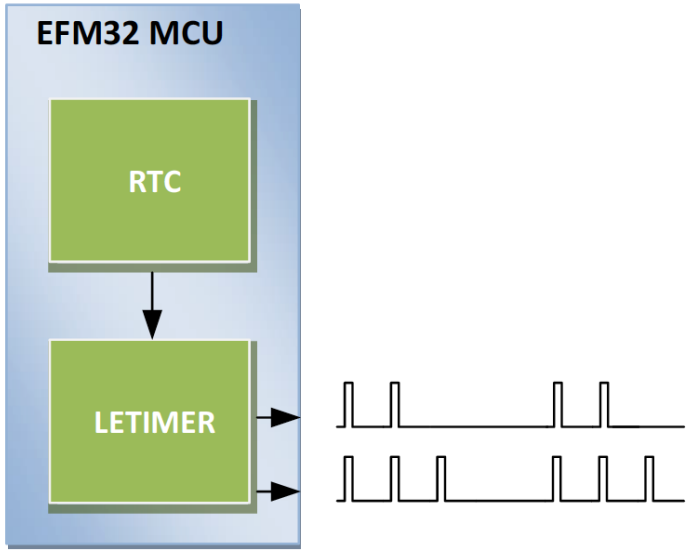

CPU and Memory
ARM Cortex-M3 processor
Memory Protection Unit
Embedded Trace Macrocell
Flash Program Memory
RAM Memory
Debug Interface
DMA
Clock Management
High Freq Crystal Osc
High Freq RC Osc
Low Freq Crystal Osc
Low Freq RC Osc
Ultra Low Freq RC Osc
Auxiliary RC Osc
Energy Management
Voltage Regulator
Voltage Comparator
Brown-out Detector
Power-on Reset
Back-up Power Domain
32-bit bus
Peripheral Reflex System
Serial Interfaces
USART
UART
Low Energy UART
I2C
USB
I/O Ports
External Bus Interface
TFT Driver
External Interrupt
General Purpose I/O
Pin Reset
GPIO Wake-up
Timers and Triggers
Timer/ Counter
Low Energy Sensor IF
Low Energy Timer
Real Time Counter
Pulse Counter
Watchdog Timer
Backup RTC
Analog Modules
ADC
DAC
LCD Controller
Operational Amplifier
Analog Comparator
IDAC
Security
AES Accelerator


0
1
2
3
4

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)

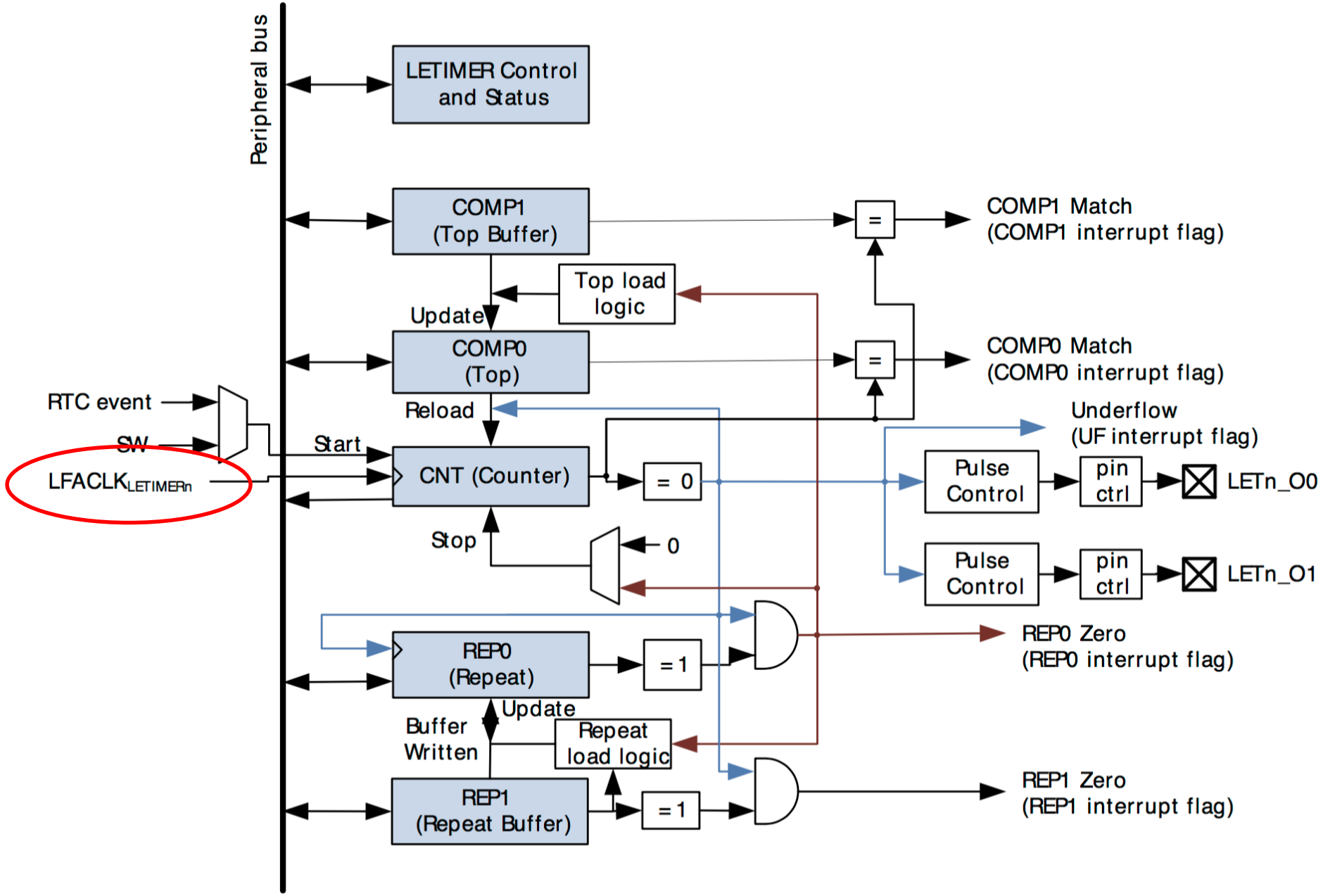


Leopard Gecko - Available down in EM3 using the ULFRC0

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



Silicon Labs' App note AN0026

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);

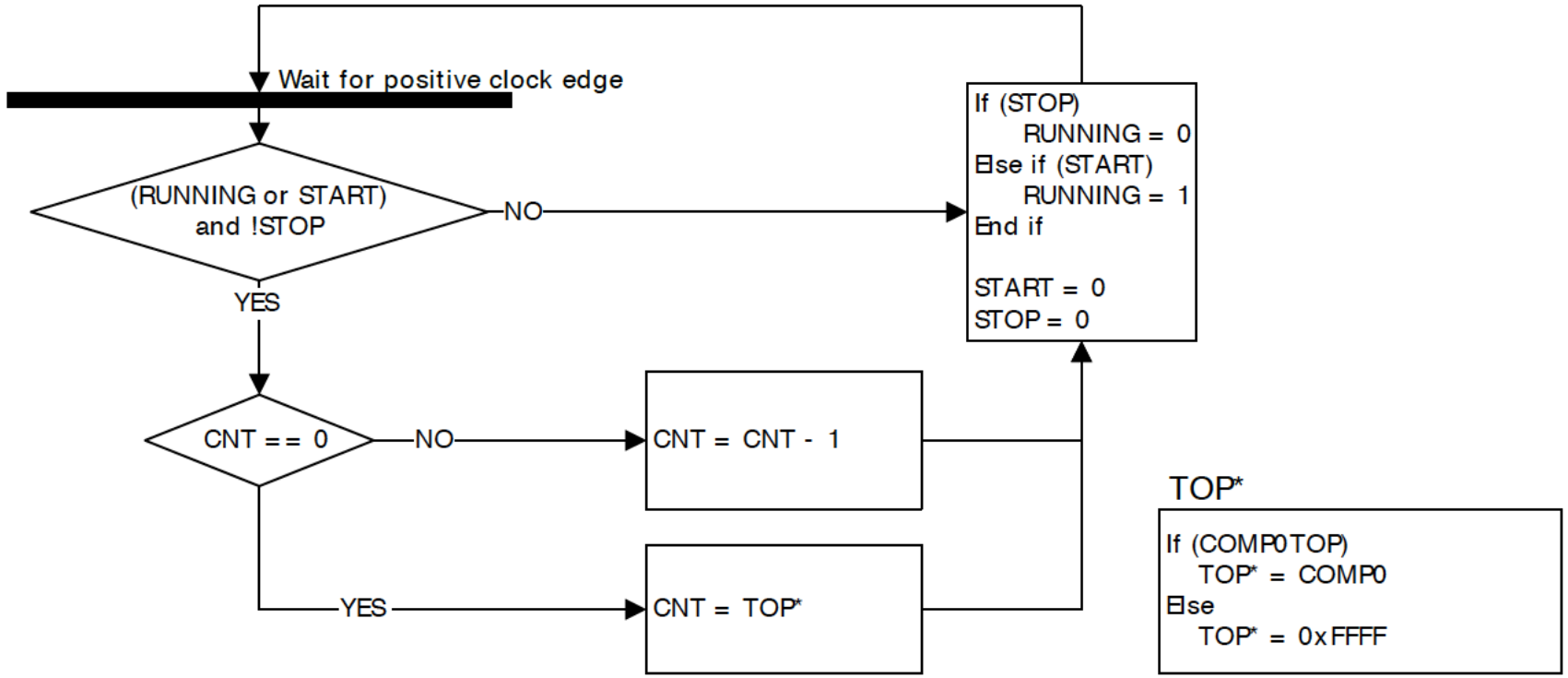
LETIMER Repeat Modes

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



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 Leopard Gecko

Reading List

Below is a list of required reading for this course. Questions from these readings plus the lectures from August 23rd, 2016 onward will be on the weekly quiz.

- "Testing and Debugging Concurrency Bugs in Event-Driven Programs," Guy Martin Tchamgoue, Kyong-Hoon Kim, and Yong-Kee Jim
  https://www.silabs.com/Support%20Documents/TechnicalDocs/manage-the-iot-on-an-energy-budget.pdf

Recommended readings. These readings will not be on the weekly quiz, but will be helpful in the class programming assignments and course project.

- "Silicon Labs' Energy Modes App note – AN0007"
  http://www.silabs.com/Support%20Documents/TechnicalDocs/AN0007.pdf

- "Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems," Adam Dunkels, Oliver Schmidt, Tiemo Voigt, Muneeb Ali
  http://muneebali.com/pubs/dunkels06protothreads.pdf

- EFM32 CMU application note - AN0004
  http://www.silabs.com/Support%20Documents/TechnicalDocs/AN0004.pdf

- EFM32 GPIO application note - AN0012
  http://www.silabs.com/Support%20Documents/TechnicalDocs/AN0012.pdf

- EFM32 Low Energy Timer LETIMER application note - AN0026
  http://www.silabs.com/Support%20Documents/TechnicalDocs/AN0026.pdf

Important web link below. It will take you to the Silicon Labs' application note home page for the Silicon Labs' EFM32 family of products:
http://www.silabs.com/products/mcu/Pages/32-bit-mcu-application-notes.aspx

Quiz 2

- Due by 11:59pm on Sunday, January 29th, 2017
- Questions will be from the required reading plus lectures from January 17th, 2017 onward

Today's Summary

- Simplicity Studio Tutorial
- Quiz 1 review
- Keeping Track of the Energy State
- Interrupts
- LETIMER0
- GPIO
- Reading List
- Quiz 2 assigned

Discussion topics for next lecture

- Review of Simplicity Exercise
- GPIO peripheral
- Synchronous and Asynchronous Routines
- Mobile/Pervasive Adaptive Computing System Considerations
- Network considerations
- Documentation style sheet
- Programming Assignment #1
  - Objective: Become familiar with the Silicon Labs' Simplicity development system as well as learn the different Leopard energy modes and how to manage these energy modes.