ECEN 5023-001, -001B

Internet of Things Embedded Firmware

Lecture #3

5 September 2017

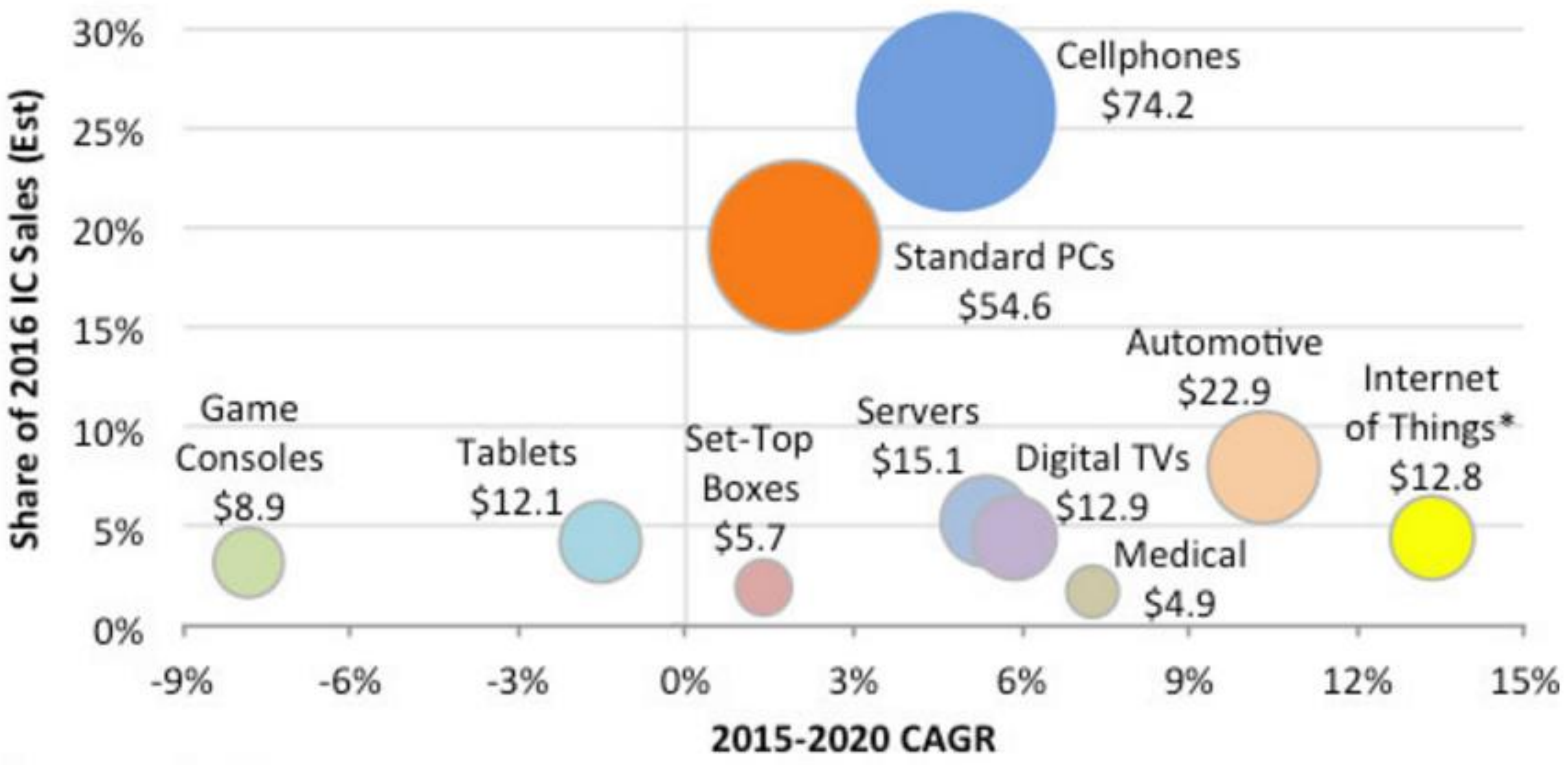
Agenda

- Class Announcements
- Reading Assigned
- Quiz 2 Assigned
- Quiz 1 review
- Directly Accessing Registers
- Clock Trees
- Energy Modes
- Keeping Track of the Energy State
- Interrupts
- LETIMER0

Class Announcements

- Simplicity Exercise is due at 11:59pm on September 6th, 2017
- Quiz #2 is due at 11:59pm on Sunday, September 10th, 2017

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



*Covers only the Internet connection portion of systems.

Source: IC Insights

Reading List

Below is a list of required reading for this course. Questions from these readings plus the lectures from August 28th, 2017 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
http://www.sersc.org/journals/IJAST/vol40/4.pdf

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

1. "Silicon Labs' Energy Modes App note – AN0007
   - Available on D2L as well as in Simplicity
2. "Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems," Adam Dunkels, Oliver Schmidt, Tiemo Voigt, Muneeb Ali
   - http://muneebali.com/pubs/dunkels06protothreads.pdf
3. EFM32 CMU application note - AN0004
   - Available on D2L as well as in Simplicity
4. EFM32 GPIO application note - AN0012
   - Available on D2L as well as in Simplicity
5. EFM32 Low Energy Timer LETIMER application note - AN0026
   - Available on D2L as well as in Simplicity

Important web link below. It will take you to the Silicon Labs' Hardware Abstraction Layer home page for the Silicon Labs' EFR32BG1 family of products:
- http://devtools.silabs.com/dl/documentation/doxygen/5.1.0/

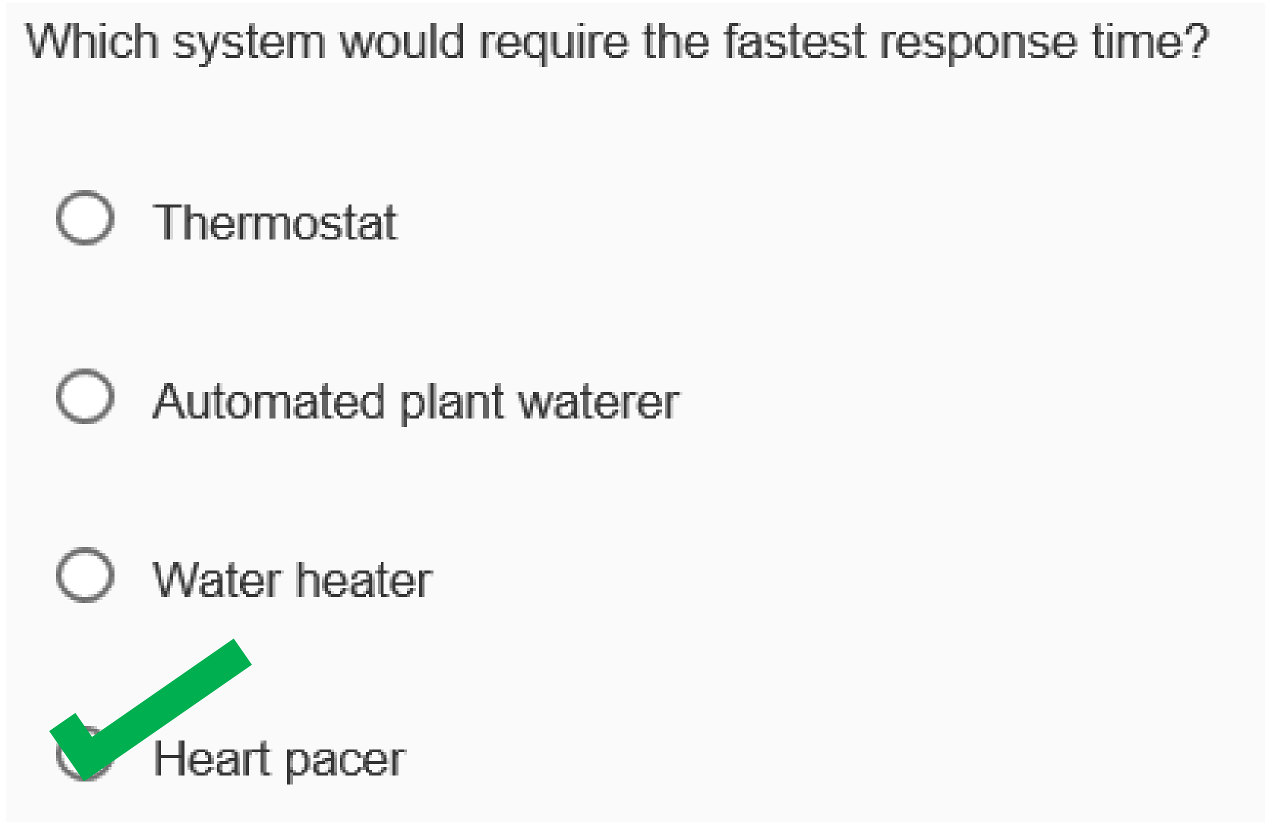
Class Survey Results

- 28 students responded. I will use the results as a representative of the entire class.
  - Question 1: 46% have little or no experience in assemble programming
  - Question 2: 14% have little or no experience in C programming
  - Question 3: 46% have little or no experience in C++ programming
  - Question 4: 39% have little or no experience in coding to the metal
  - Question 5: 29% have little or no experience in programming a micro-controller
  - Question 6: 14% have little or no experience in using an IDE
  - Question 7: 36% have little or no experience in using an Eclipse based IDE
  - Question 8: 36% have little or no experience in using break points in an IDE debugger
  - Question 9: 45% have little or no experience in using the IDE register view

Class Survey Results

- 28 students responded. I will use the results as a representative of the entire class.
  - Question 10: Programming peripheral with no experience
    - Bluetooth or radio: 89%
    - USB: 75%
    - FLASH: 57%
  - Question 11: Programming peripheral with little experience
    - DMA: 50%
    - SPI: 39%
    - Interrupt Controller 39%
  - Question 12: Programming peripheral with moderate experience
    - TIMERS: 54%
    - ADC: 50%
    - SPI: 46%
  - Question 13: Programming peripheral with expert experience
    - TIMERS: 43%
    - ADC 32%
    - UART 18%
  - Question 14: 82% have taken ECEN 5813
  - Question 15: 25% have taken ECEN 5613

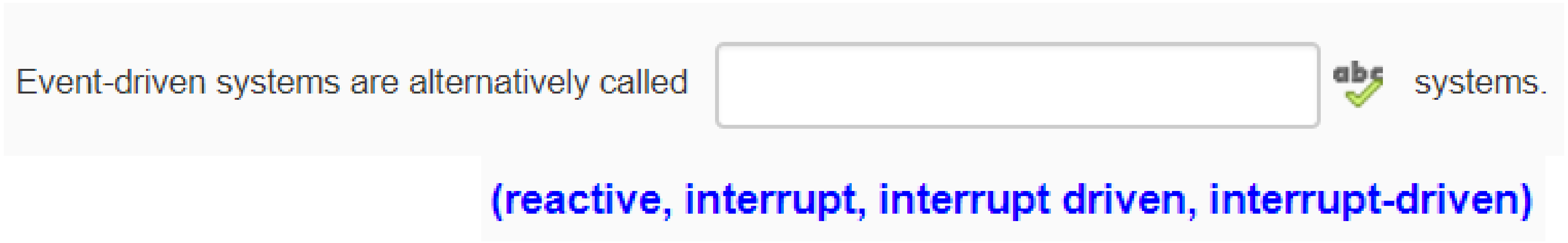
Quiz



Which system would require the fastest response time?

- Thermostat
- Automated plant waterer
- Water heater
- [✓ Heart pacer]

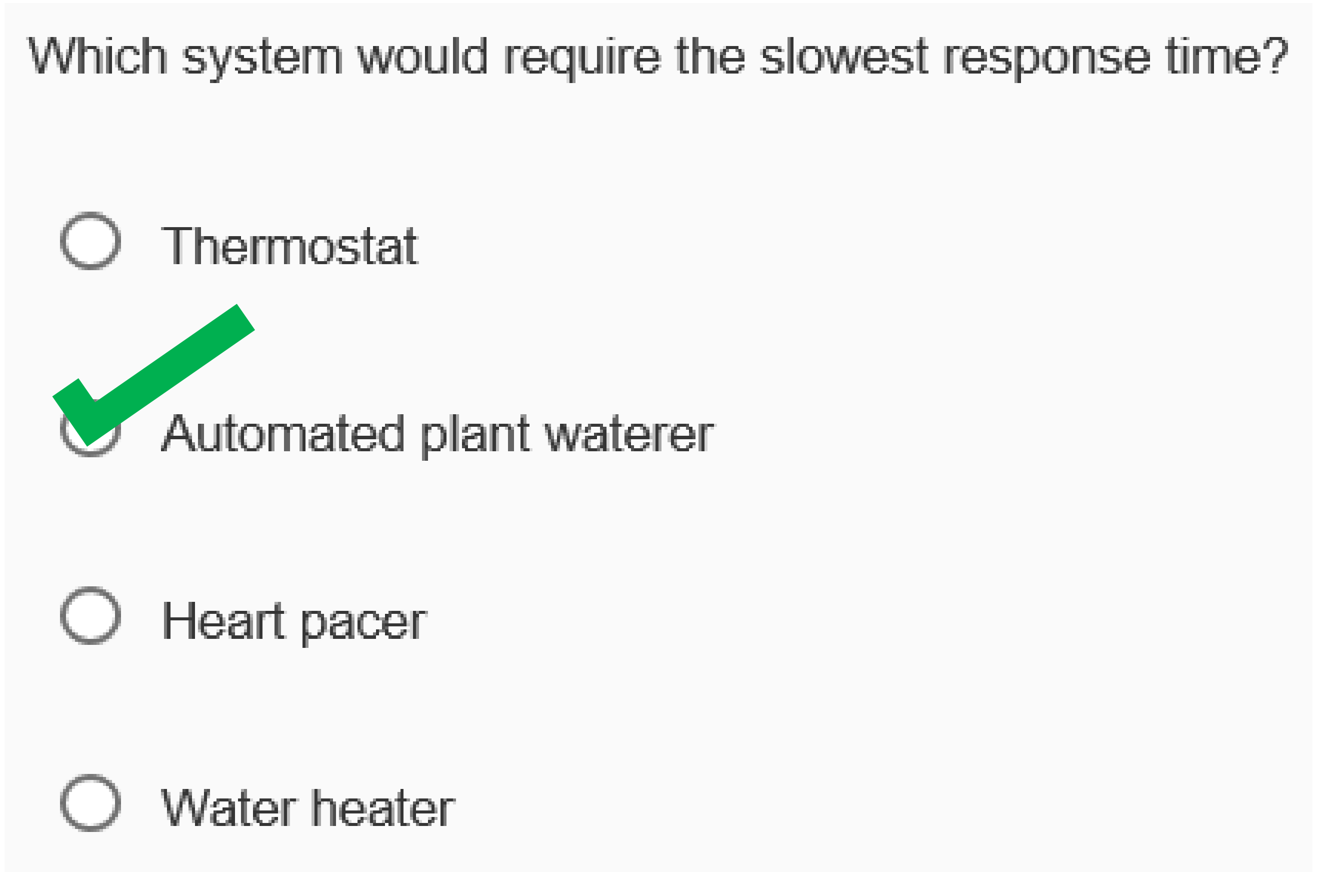
Quiz



Event-driven systems are alternatively called ______ systems.

(reactive, interrupt, interrupt driven, interrupt-driven)

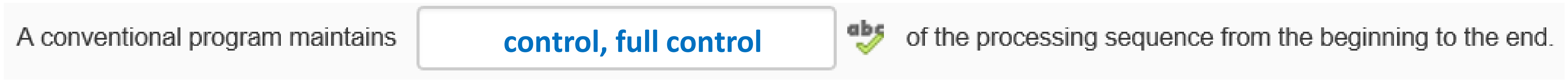
Quiz



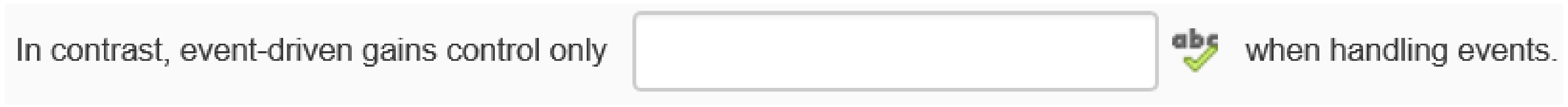
Which system would require the slowest response time?

- ○ Thermostat
- [✓ Automated plant waterer]
- ○ Heart pacer
- ○ Water heater

Quiz



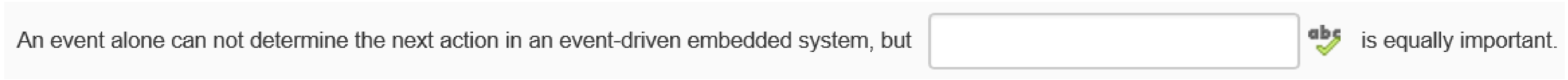
A conventional program maintains control, full control of the processing sequence from the beginning to the end.



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

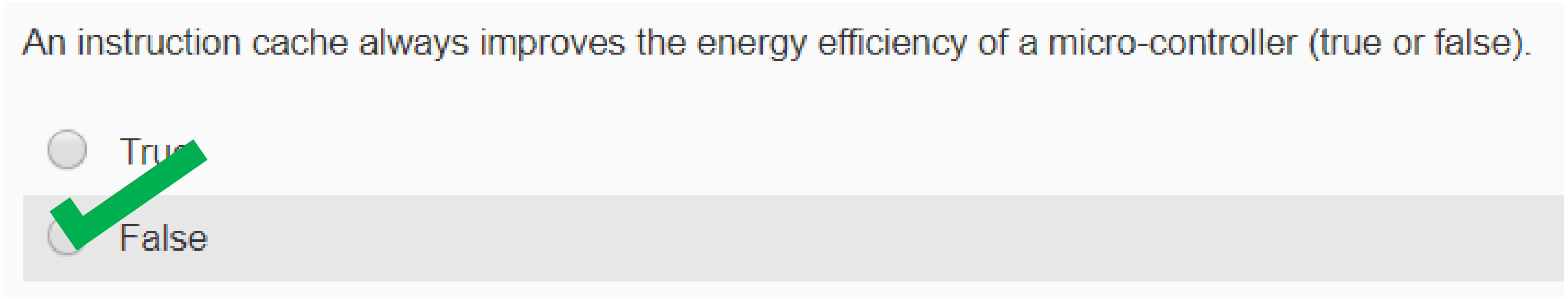
sporadically, sporadic, temporary, temporarily, occasionally

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

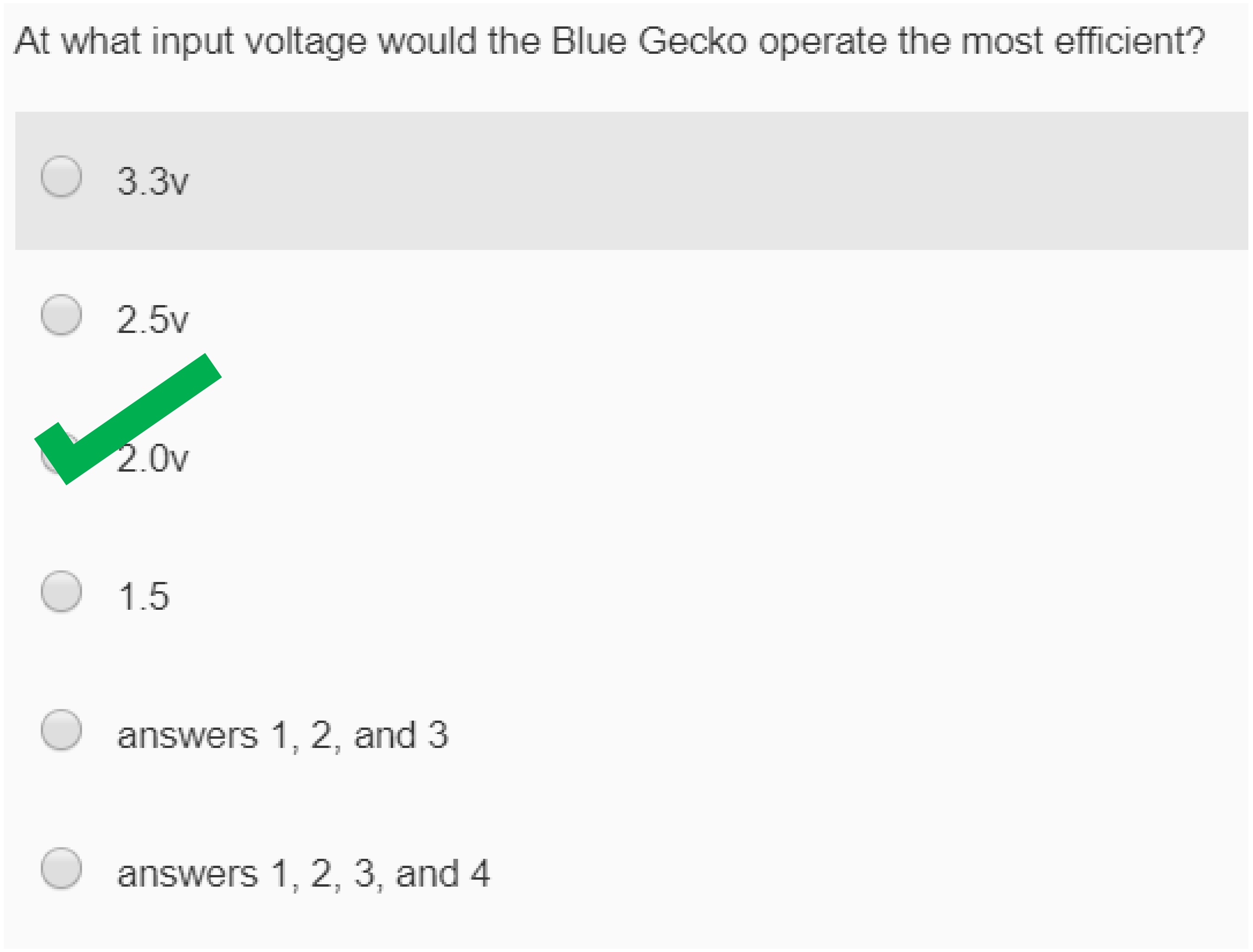
Quiz



An instruction cache always improves the energy efficiency of a micro-controller (true or false).

True [✓ False]

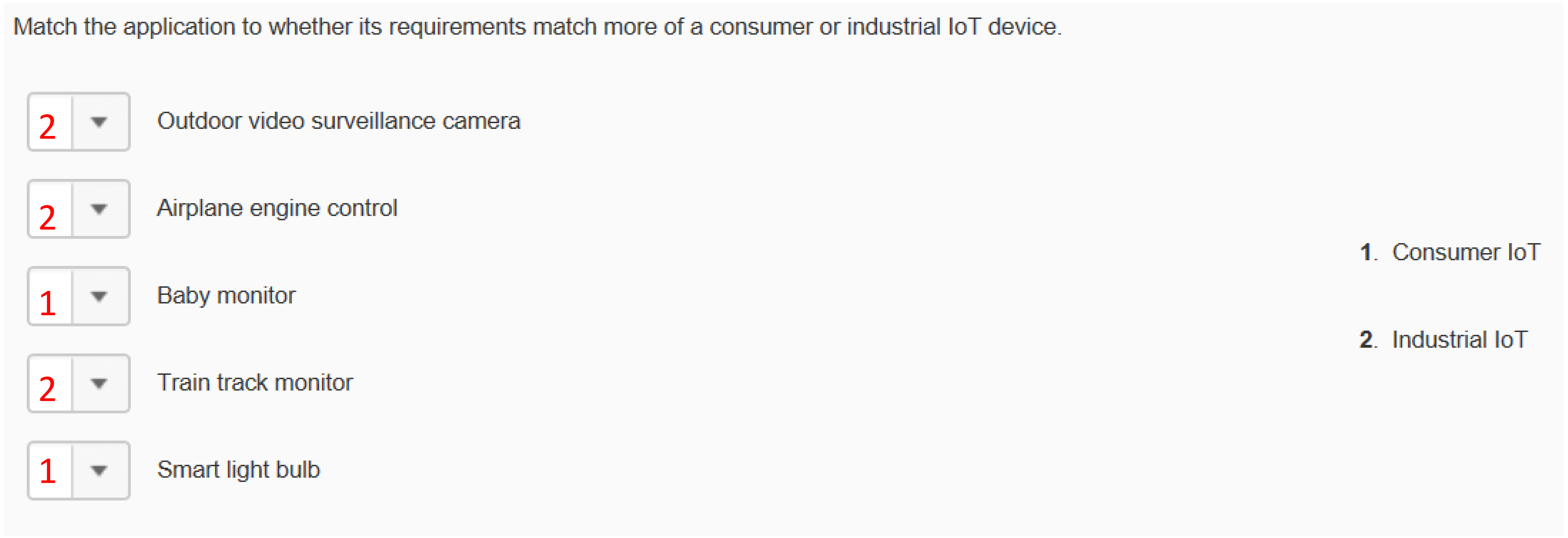
Quiz



At what input voltage would the Blue Gecko operate the most efficient?

- 3.3v
- 2.5v
- [✓ 2.0v]
- 1.5
- answers 1, 2, and 3
- answers 1, 2, 3, and 4

Quiz

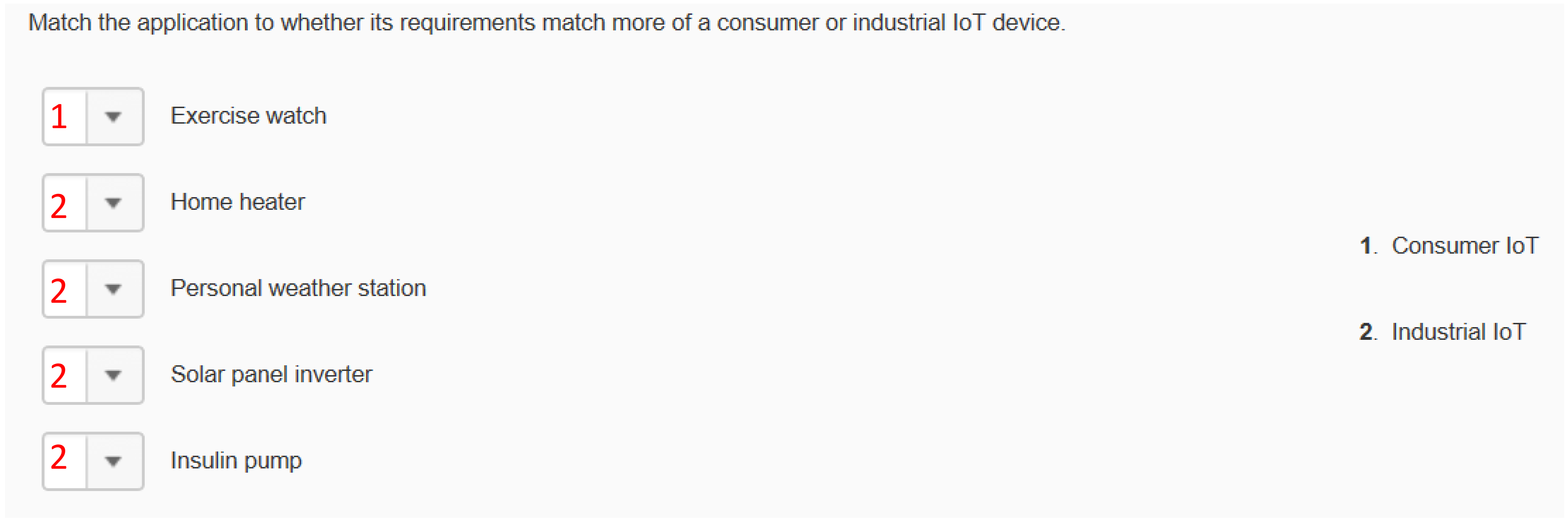


Match the application to whether its requirements match more of a consumer or industrial IoT device.

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



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

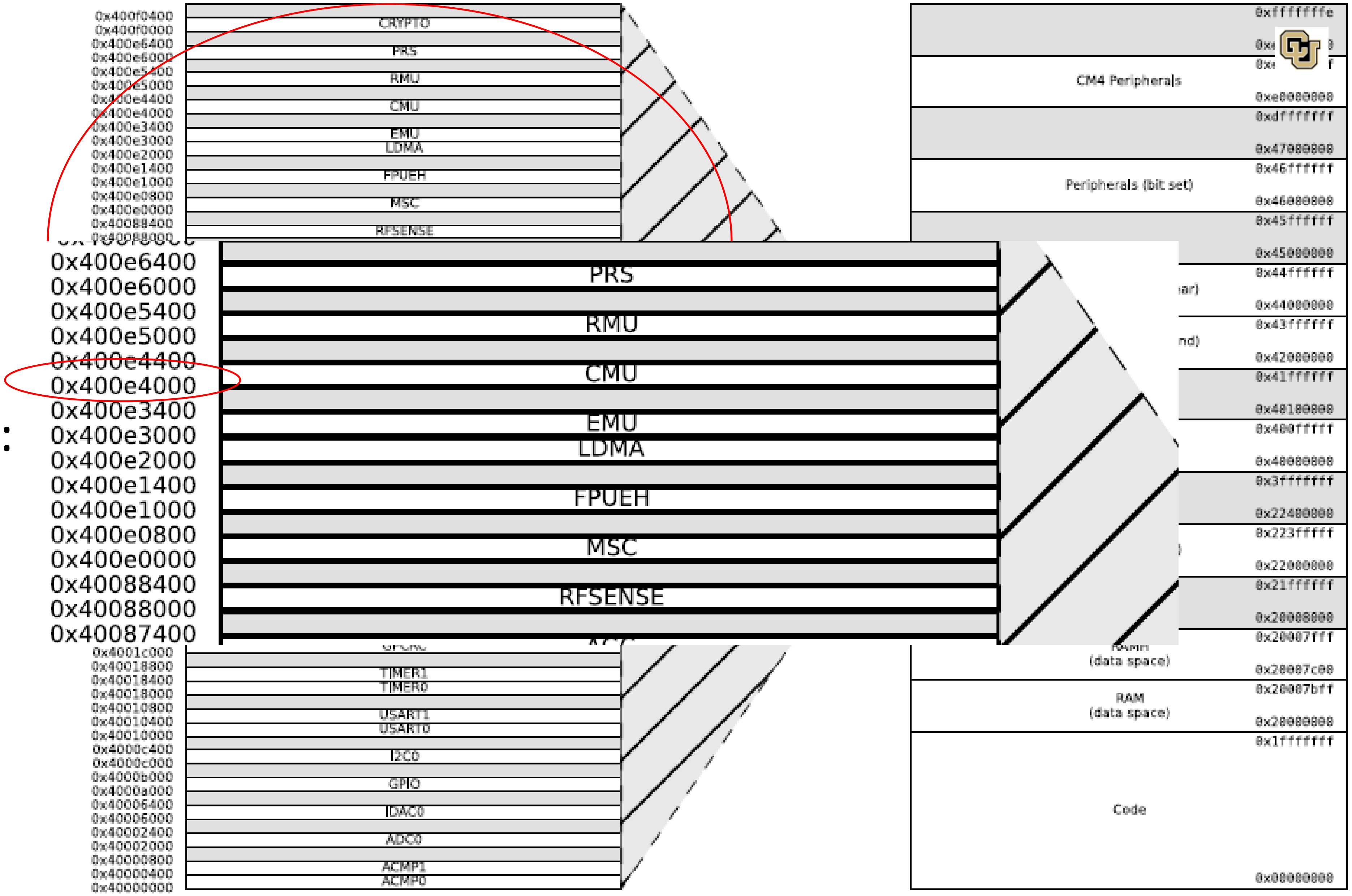
1	Exercise watch
2	Home heater
2	Personal weather station
2	Solar panel inverter
2	Insulin pump

1. Consumer IoT
2. Industrial IoT

Accessing registers directly using C-code

CMU base address is:

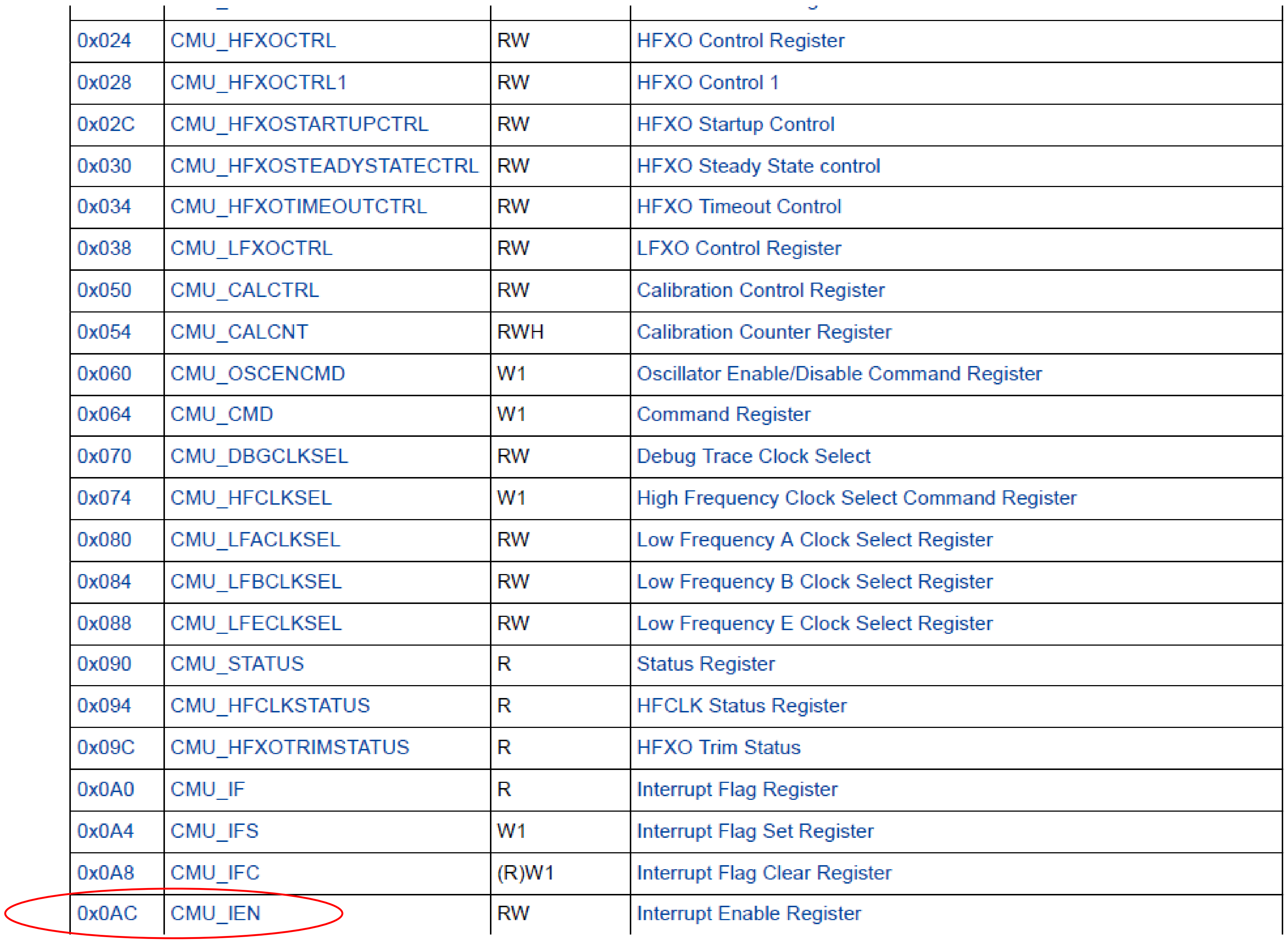
0x400e4000



Silicon Labs' EFR32BG1 data sheet

Accessing registers directly using C-code

CMU Interrupt Enable register offset address 0x0AC



0x024	CMU_HFXOCTRL	RW	HFXO Control Register
0x028	CMU_HFXOCTRL1	RW	HFXO Control 1
0x02C	CMU_HFXOSTARTUPCTRL	RW	HFXO Startup Control
0x030	CMU_HFXOSTEADYSTATECTRL	RW	HFXO Steady State control
0x034	CMU_HFXOTIMEOUTCTRL	RW	HFXO Timeout Control
0x038	CMU_LFXOCTRL	RW	LFXO Control Register
0x050	CMU_CALCTRL	RW	Calibration Control Register
0x054	CMU_CALCNT	RWH	Calibration Counter Register
0x060	CMU_OSCENCMD	W1	Oscillator Enable/Disable Command Register
0x064	CMU_CMD	W1	Command Register
0x070	CMU_DBGCLKSEL	RW	Debug Trace Clock Select
0x074	CMU_HFCLKSEL	W1	High Frequency Clock Select Command Register
0x080	CMU_LFACLKSEL	RW	Low Frequency A Clock Select Register
0x084	CMU_LFBCLKSEL	RW	Low Frequency B Clock Select Register
0x088	CMU_LFECLKSEL	RW	Low Frequency E Clock Select Register
0x090	CMU_STATUS	R	Status Register
0x094	CMU_HFCLKSTATUS	R	HFCLK Status Register
0x09C	CMU_HFXOTRIMSTATUS	R	HFXO Trim Status
0x0A0	CMU_IF	R	Interrupt Flag Register
0x0A4	CMU_IFS	W1	Interrupt Flag Set Register
0x0A8	CMU_IFC	(R)W1	Interrupt Flag Clear Register
0x0AC	CMU_IEN	RW	Interrupt Enable Register

Accessing registers directly using C-code

- CMU->IEN

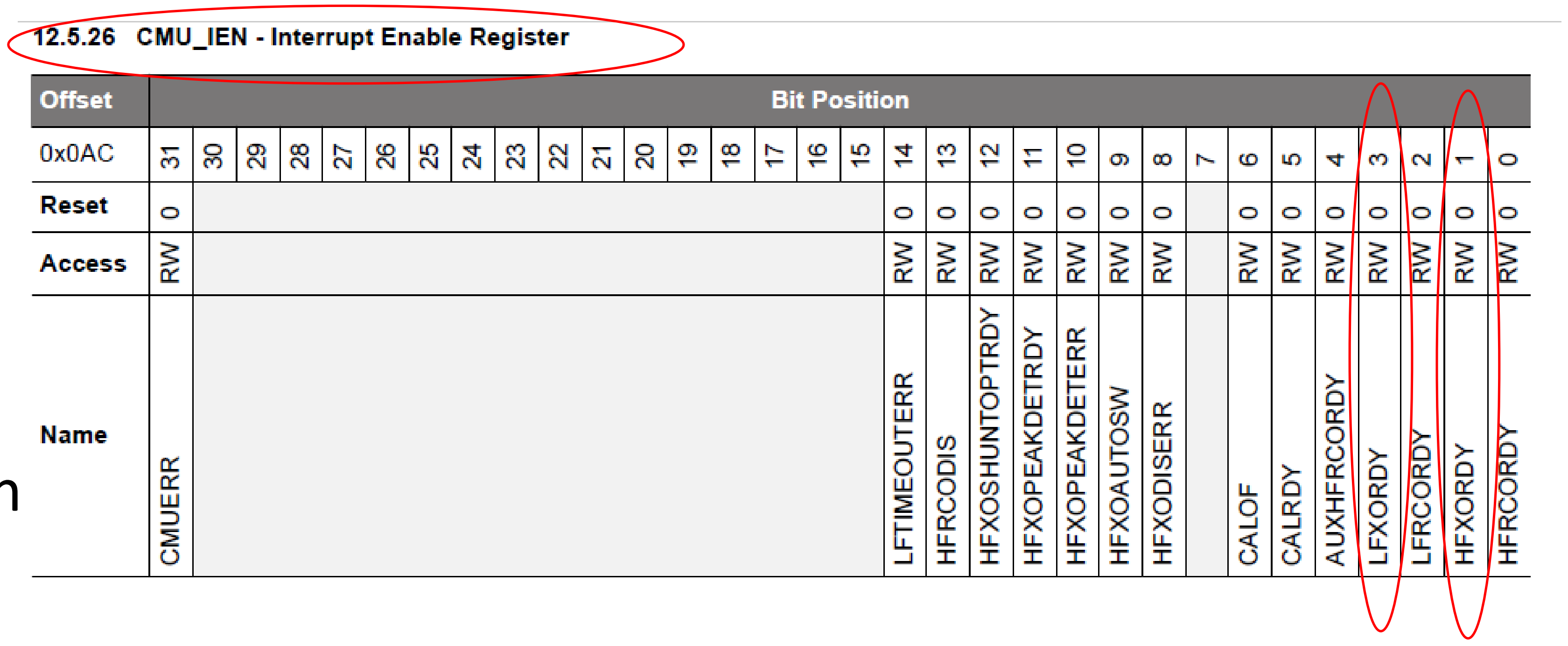
• CMU base address	0x400e4000
• IEN register offset	+ 0x000000aC
• CMU->IEN	= 0x400e40aC

Accessing a register bit using C-code

- HFXO ready bit is bit 1
- LFXO ready bit is bit 3

- Register name bit equals a 1 in the correct bit position

- CMU_IEN_HFXORDY is
  - 0b00000010
- CMU_IEN_LFXORDY is
  - 0b00001000



12.5.26 CMU_IEN - Interrupt Enable Register

Offset	Bit Position																															
0x0AC	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	0	0		0	0	0	0	0	0	0
Access	RW																	RW	RW	RW	RW	RW	RW	RW		RW	RW	RW	RW	RW	RW	RW
Name	CMUERR																	LFTIMEOUTERR	HFRCODIS	HFXOSHUNTOPTRDY	HFXOPEAKDETRDY	HFXOPEAKDETERR	HFXOAUTOSW	HFXODISERR		CALOF	CALRDY	AUXHFRCORDY	LFXORDY	LFRCORDY	HFXORDY	HFRCORDY

Accessing registers directly using C-code

- CMU->IEN
  - CMU base address 0x400e4000
  - IEN register offset + 0x000000aC
  - CMU->IEN = 0x400e40aC
- CMU_IEN_HFXORDY | CMU_IEN_LFXORDY
  - CMU_IEN_HFXORDY 0b00000010
  - CMU_IEN_LFXORDY 0b00001000
  - CMU_IEN_HFXORDY | CMU_IEN_LFXORDY 0b00001010
- CMU->IEN = CMU_IEN_HFXORDY | CMU_IEN_LFXORDY;
  - CMU Interrupt Enable Register value at 0x400e40aC now is 0b00001010

Not clearing bits when you are adding setting a bit in a register

- Assume LETIMER0 already has the UF bit set in the Interrupt Enable Register
  - LETIMER0 currently is set to 0x00000004 (UF bit set)
- Now, lets also enable the COMP1 interrupt to the LETIMER0 peripheral

  ✗ LETIMER0->IEN = LETIMER_IEN_COMP1;
    - LETIMER0 now is set to 0x00000002 (Overwrote UF interrupt bit)
- These are correct syntaxes to add a bit and not overwrite a bit
  - LETIMER0->IEN = LETIMER0->IEN | LETIMER_IEN_COMP1;
  - LETIMER0->IEN |= LETIMER_IEN_COMP1;
  - LETIMER_IntEnables(LETIMER0, LETIMER_IEN_COMP1);

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

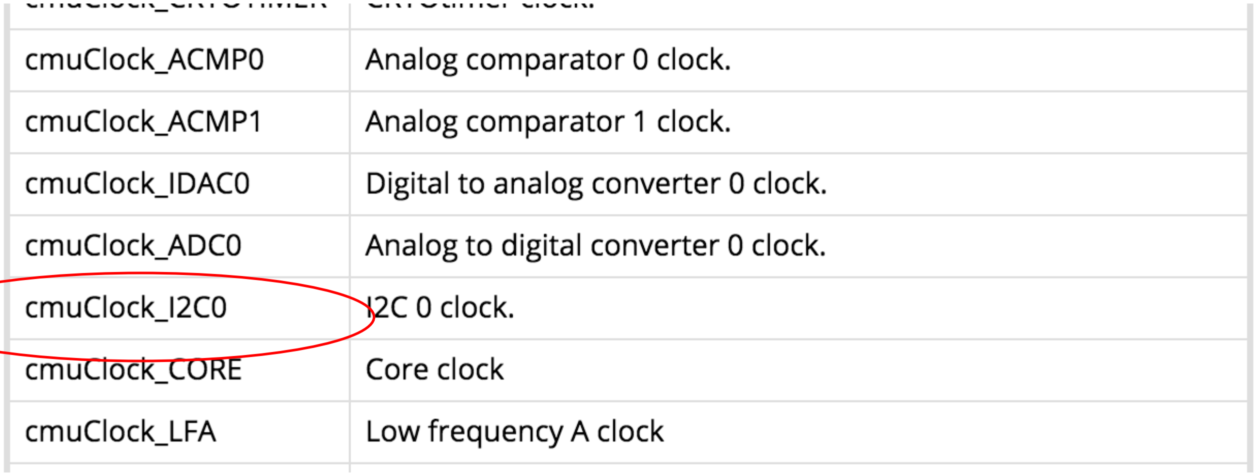
Clearing a single bit(s) without disturbing others in a register

- Assume LETIMER0 already has both the COMP1 & UF bits set in the Interrupt Enable Register
  - LETIMER0 currently is set to 0x00000006 (UF & COMP1 bits set)
- Now, lets disable the UF interrupt to the LETIMER0 peripheral
  - ✖ LETIMER0->IEN = ~LETIMER_IEN_UF;
    - LETIMER0 now is set to 0xFFFFFFFB (Overwrote all bits)
- These are correct syntaxes to add a bit and not overwrite a bit
  - LETIMER0->IEN = LETIMER0->IEN & ~LETIMER_IEN_UF;
  - LETIMER0->IEN &= ~LETIMER_IEN_UF;
  - LETIMER_IntDisables(LETIMER0, LETIMER_IEN_UF);

```
__STATIC_INLINE void LETIMER_IntDisable(LETIMER_TypeDef *letimer, uint32_t flags)
{
  letimer->IEN &= ~flags;
}
```

Enabling a peripheral clock

- Example: Enabling the clock to the I2C0 peripheral
- Emlib routine:
  - **void, CMU_ClockEnable(CMU_Clock_TypeDef** clock, **bool** enable **)**
  - CMU_Clock_TypeDef

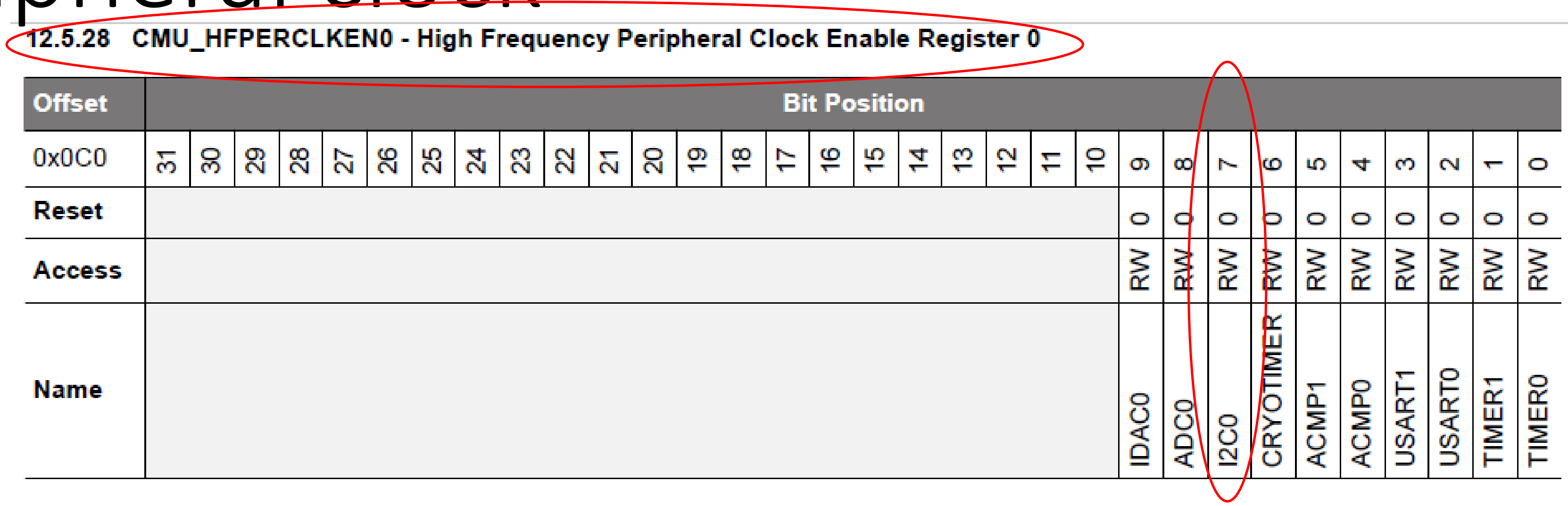


cmuClock_ACMP0	Analog comparator 0 clock.
cmuClock_ACMP1	Analog comparator 1 clock.
cmuClock_IDAC0	Digital to analog converter 0 clock.
cmuClock_ADC0	Analog to digital converter 0 clock.
cmuClock_I2C0	I2C 0 clock.
cmuClock_CORE	Core clock
cmuClock_LFA	Low frequency A clock

  - **Bool**
    - **true** to enable clocking to the I2C0
  - CMU_ClockEnable(cmuClock_I2C0, true);

Enabling a peripheral clock

- Direct register access

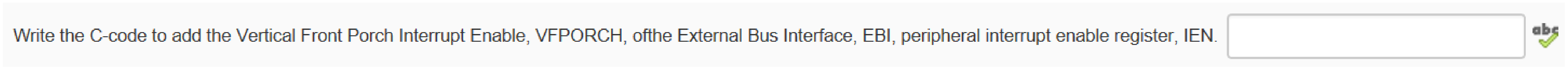


12.5.28 CMU_HFPERCLKEN0 - High Frequency Peripheral Clock Enable Register 0

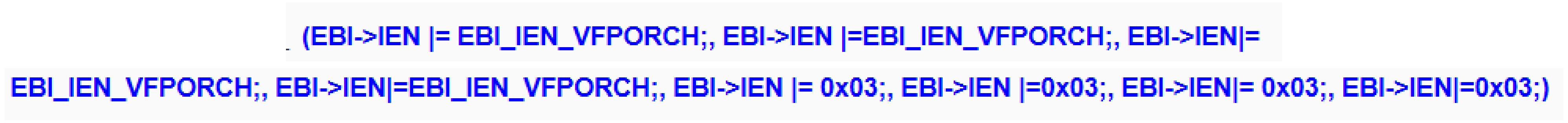
Offset	Bit Position																															
0x0C0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																							0	0	0	0	0	0	0	0	0	0
Access																							RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Name																							IDAC0	ADC0	I2C0	CRYOTIMER	ACMP1	ACMP0	USART1	USART0	TIMER1	TIMER0

- CMU->HFPERCLKEN0 = CMU->HFPERCLKEN0 + CMU_HFPERCLKEN0_I2C0;
- Or,
- CMU->HFPERCLKEN0 |= CMU_HFPERCLKEN0_I2C0;
- Example to disable:
  - CMU->HFPERCLKEN0 &= ~CMU_HFPERCLKEN0_I2C0;

Quiz

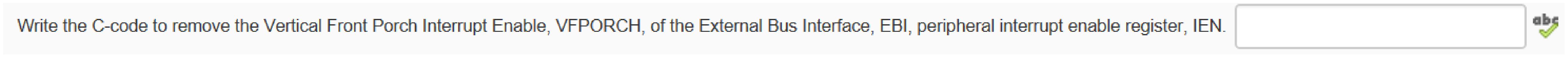


Write the C-code to add the Vertical Front Porch Interrupt Enable, VFPORCH, ofthe External Bus Interface, EBI, peripheral interrupt enable register, IEN.



(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 &= 0xfffffff7;, EBI->IEN &=0xfffffff7;, EBI->IEN&= 0xfffffff7;,

EBI->IEN&=0xfffffff7;)

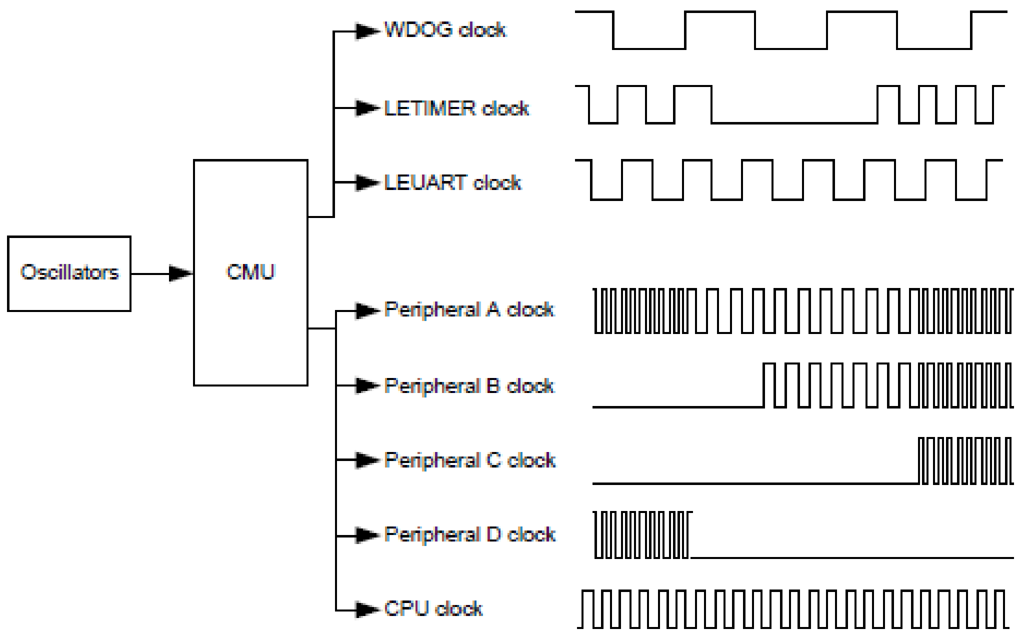
https://youtu.be/VMj3j8GU2SM

Ideal CMOS FET

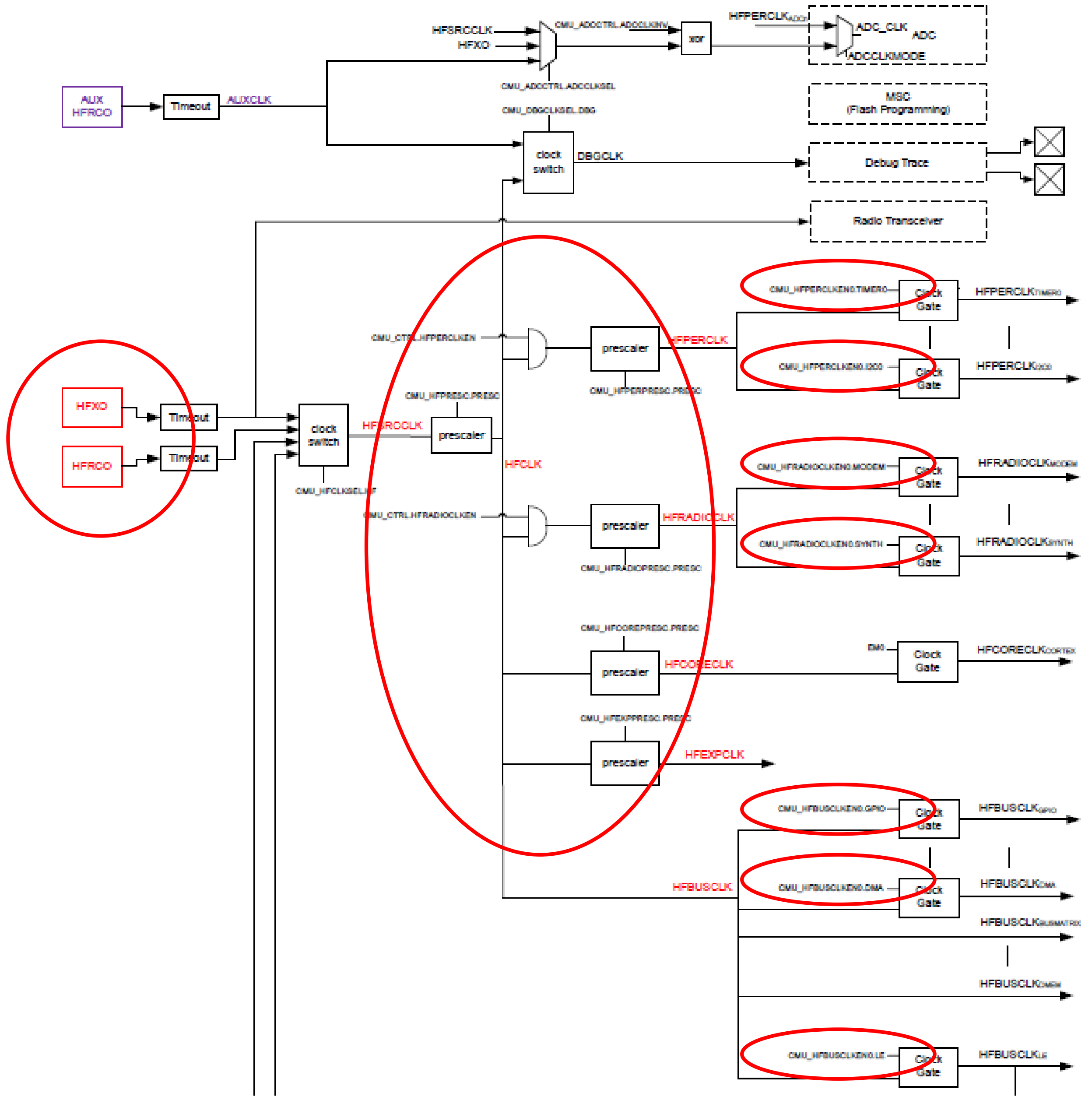
- CMOS logic reduces power consumption because no current flows (ideally), and thus no power is consumed, except when the inputs to logic gates are being switched. CMOS accomplishes this current reduction by complementing every nMOSFET with a pMOSFET and connecting both gates and both drains together.
- Thus, power/energy is reduced when:
  - CMOS logic is not clocked or switched
  - Lower switching frequency will result in reduced current/power/energy
    - Caveat: Only if it does not extend the time the CPU remains in EM0

Clock tree assumptions

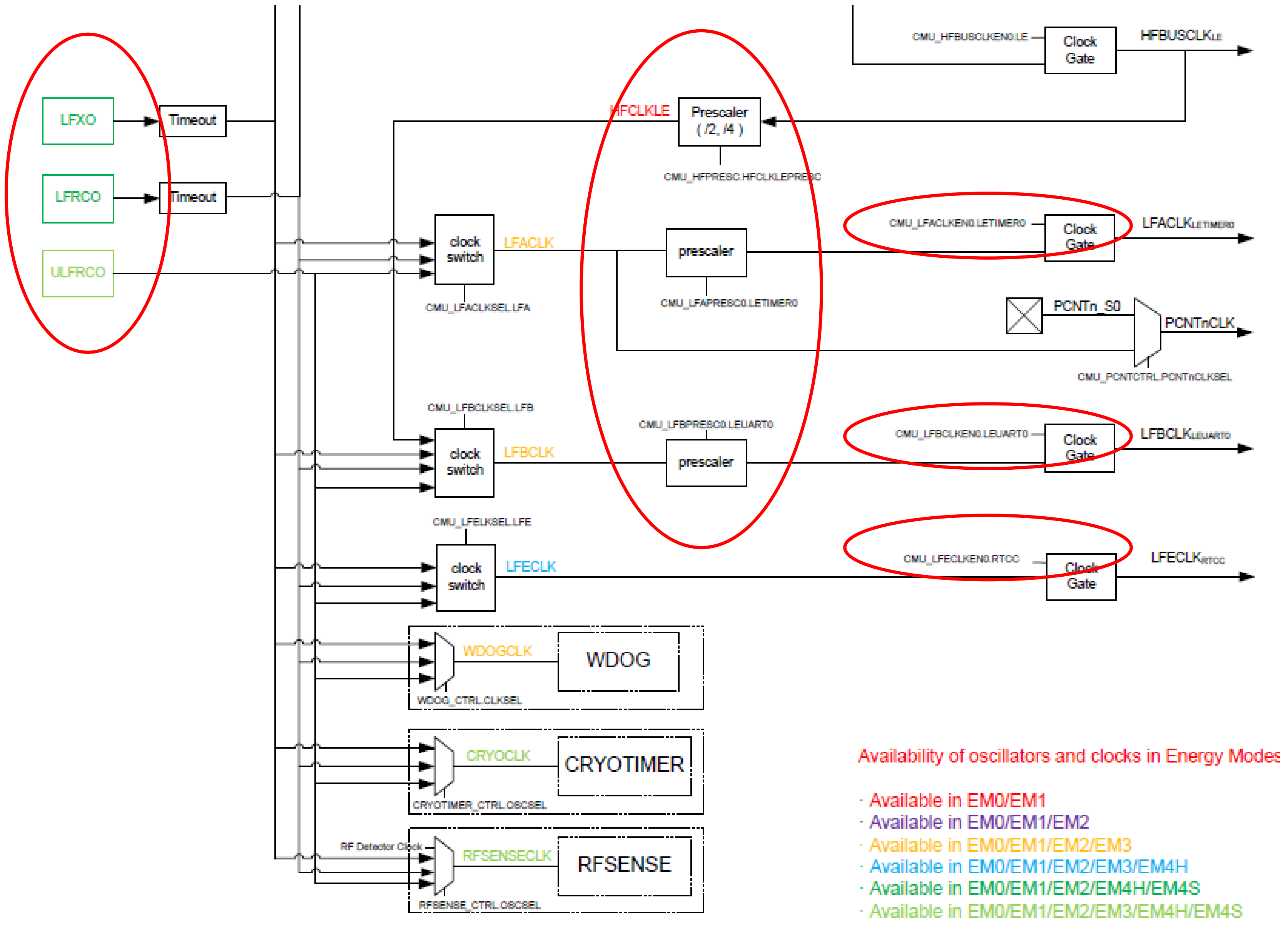
- Goal of the Silicon Labs' Blue Gecko is low power / low energy applications
- To achieve this goal:
  - Clock sources of various frequencies, tolerances, and p available
    - 38 MHz - 40 MHz High Frequency Crystal Oscillator (HFXO)
    - 1 MHz - 38 MHz High Frequency RC Oscillator (HFRCO)
    - 1 MHz - 38 MHz Auxiliary High Frequency RC Oscillator (AU
    - 32768 Hz Low Frequency Crystal Oscillator (LFXO)
    - 32768 Hz Low Frequency RC Oscillator (LFRCO)
    - 1000 Hz Ultra Low Frequency RC Oscillator (ULFRCO)
  - Prescalers on each clock tree to optimize energy to req performance
  - Clock enables to only consume energy on peripherals required for the application
    - By default, each peripheral clock is disabled / turned off



Blue Gecko High Frequency Clock Tree



Blue Gecko High Frequency Clock Tree



CMU – Clock Management Unit

- The Clock Management Unit (CMU) is responsible for controlling the oscillators and clocks in the EFR32.
  - Provides the capability to turn on and off the clock on an individual basis to all peripheral modules
  - Enables/Disables and configures the available oscillators.
  - The high degree of flexibility enables software to minimize energy consumption in any specific application by not wasting power on peripherals and oscillators that do not need to be active.

IMPORTANT PROGRAMMING NOTE!!!

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

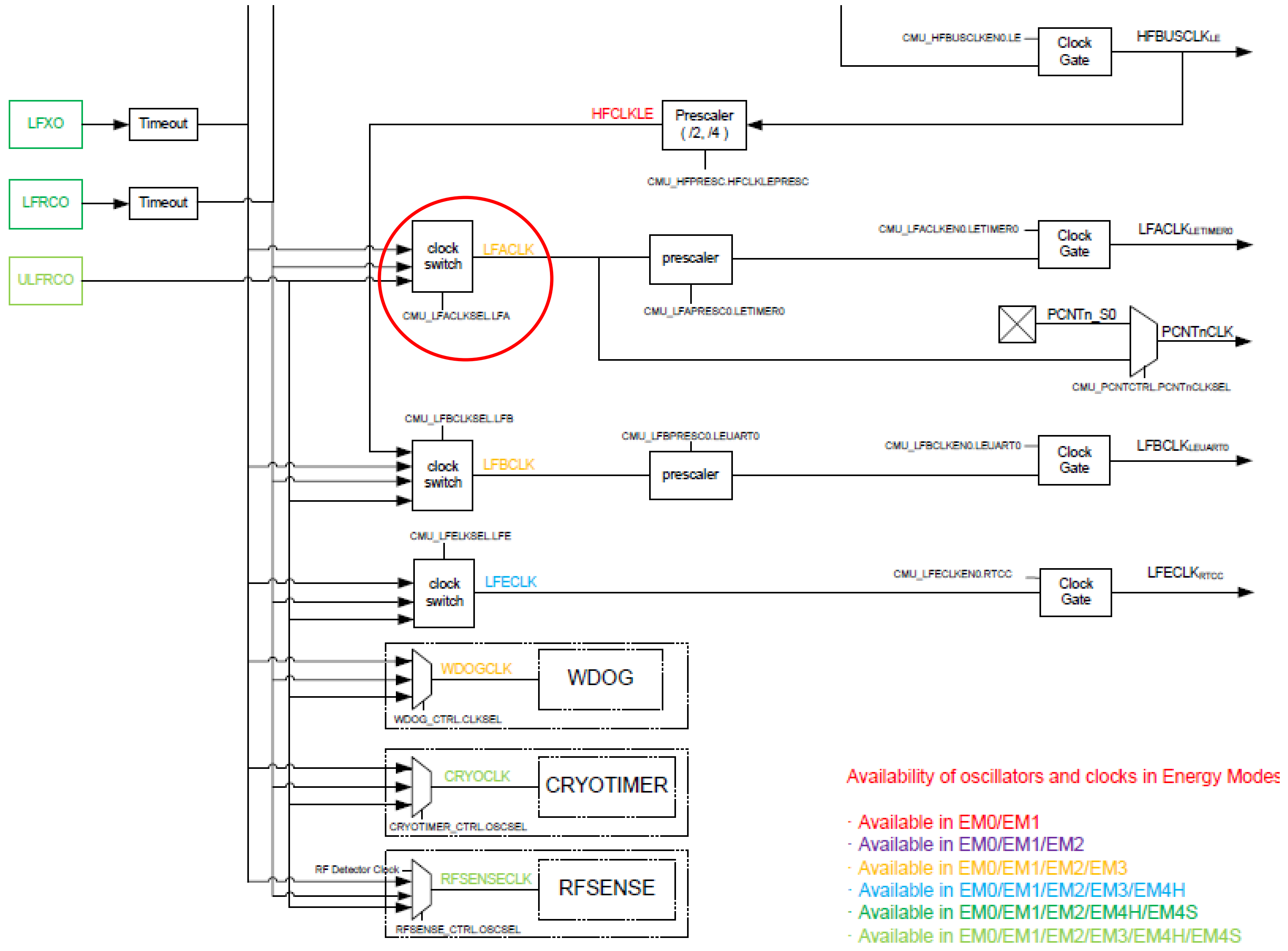
- Before disabling the HFXO, insure that you first enable the HFRCO and select the HFRCO as the HF clock source!

Enabling a peripheral

- Before programming a peripheral, its clock tree must be configured and enabled
  - Set the oscillator frequency
  - Enable its oscillator
  - Configure the oscillator to the clock tree
  - Program the peripheral prescaler
  - Enable the peripheral clock

Blue Gecko Low Frequency Clock Tree

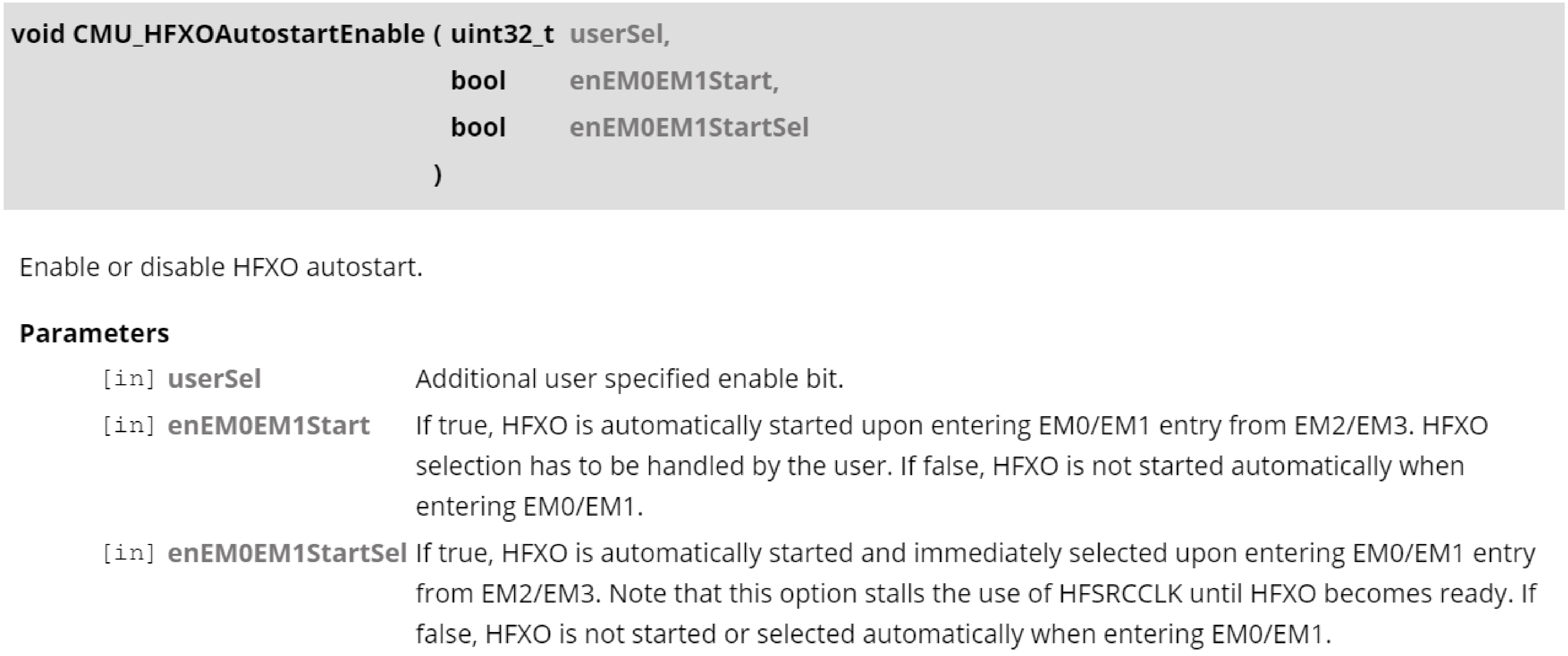
Clock Source Selection



HFRCO/AUXHFRCO Band Selection

- The default HF clock source is set to the HFRCO @ 19 MHz
- The HFRCO 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



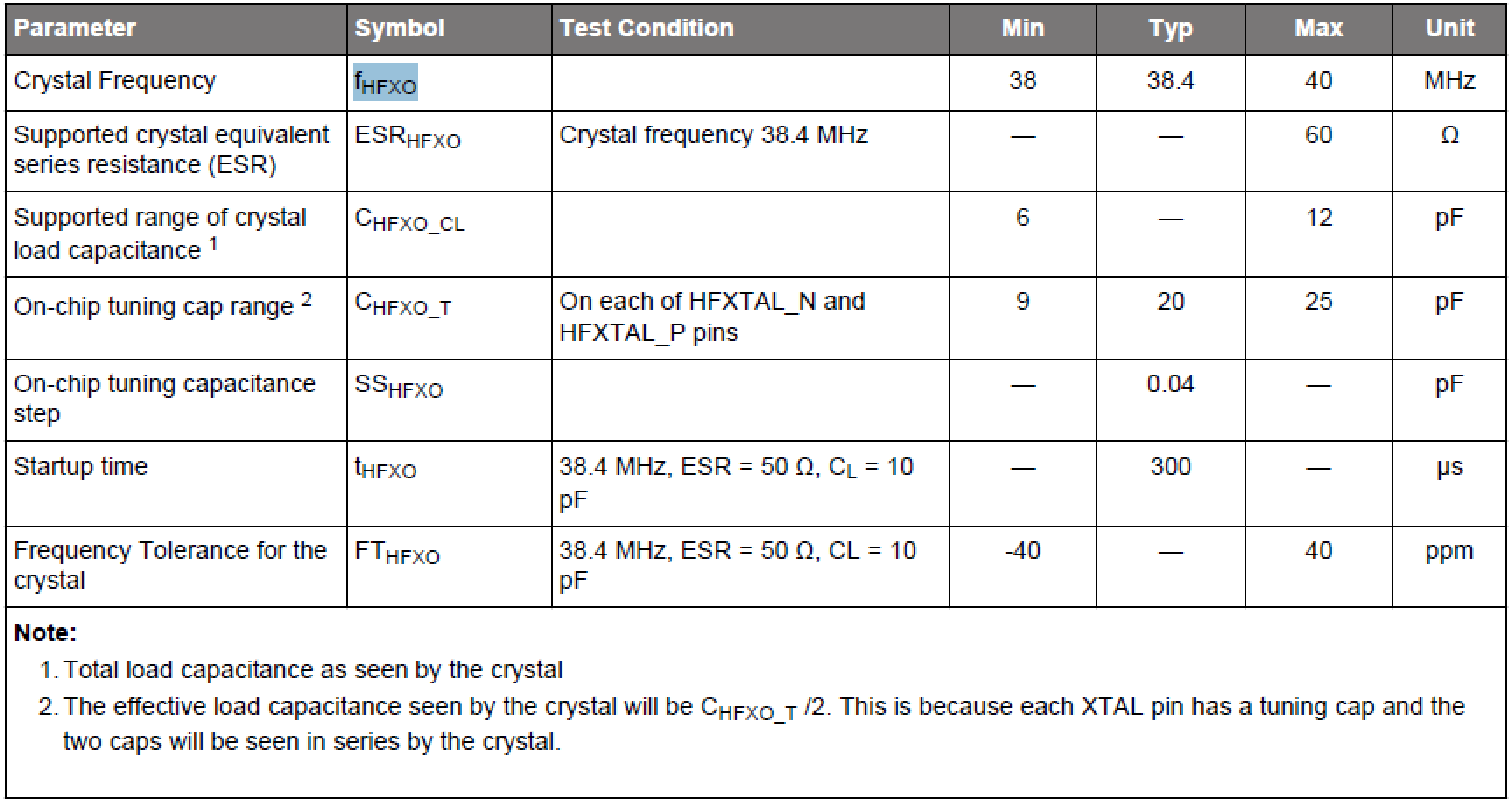
```
void CMU_HFXOAutostartEnable ( uint32_t  userSel,
                               bool      enEM0EM1Start,
                               bool      enEM0EM1StartSel
                             )
```

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:

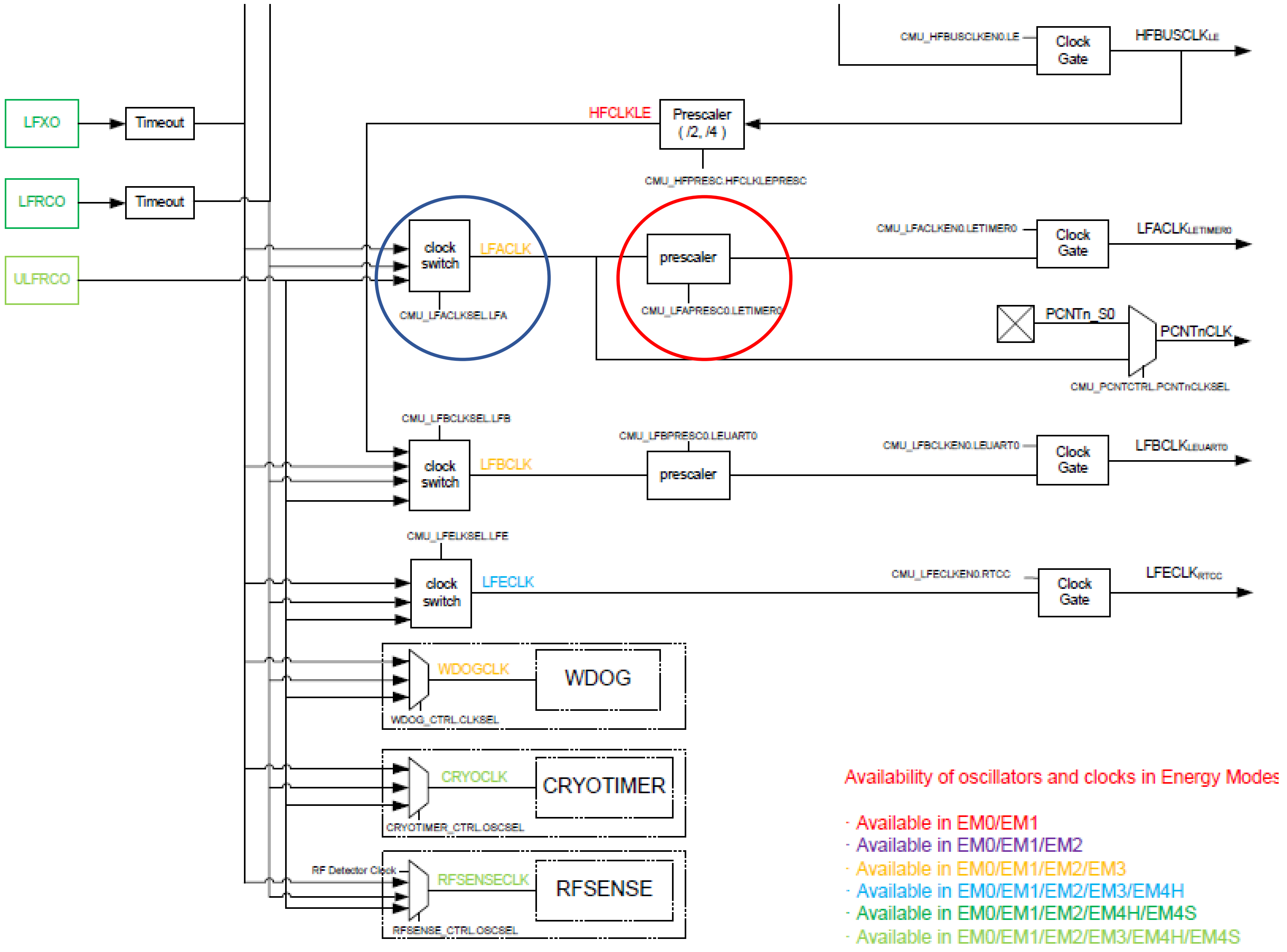
1. Total load capacitance as seen by the crystal
2. The effective load capacitance seen by the crystal will be C_{HFXO_T} /2. This is because each XTAL pin has a tuning cap and the two caps will be seen in series by the crystal.

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

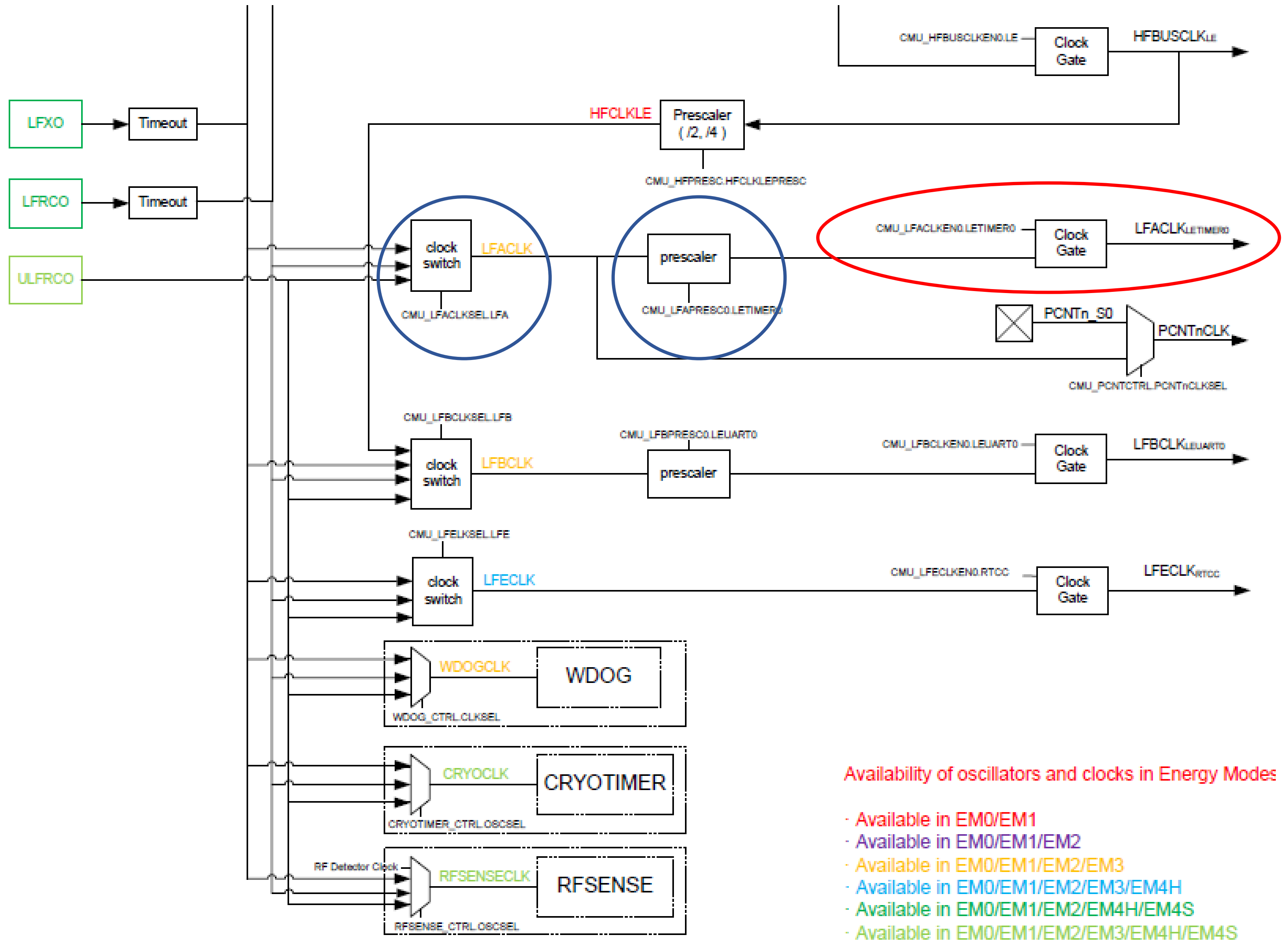


Prescaling

- 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

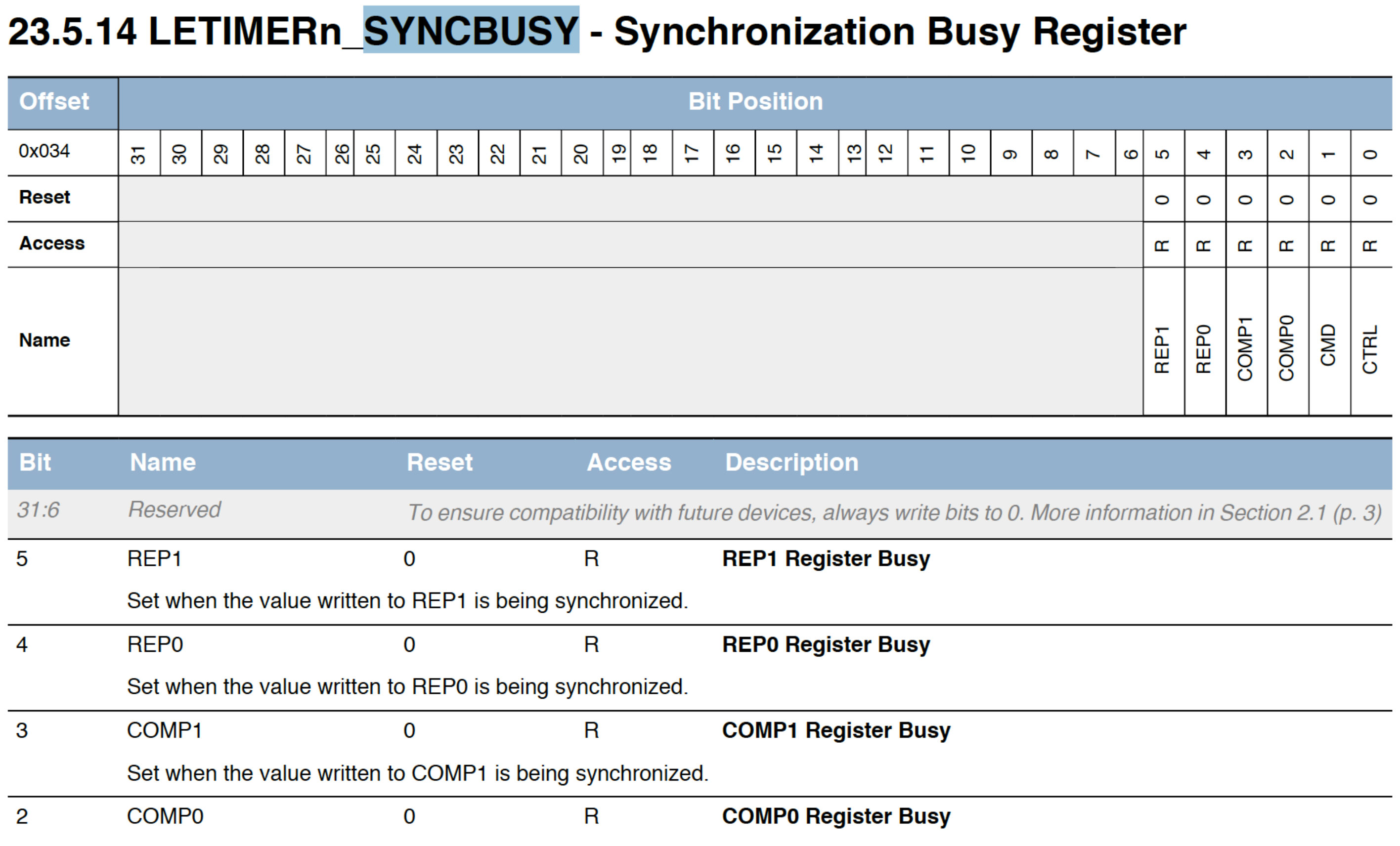


Peripheral clocks

- 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

- 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-M4 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 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-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 3.3 µA typically

EM3 – stop mode

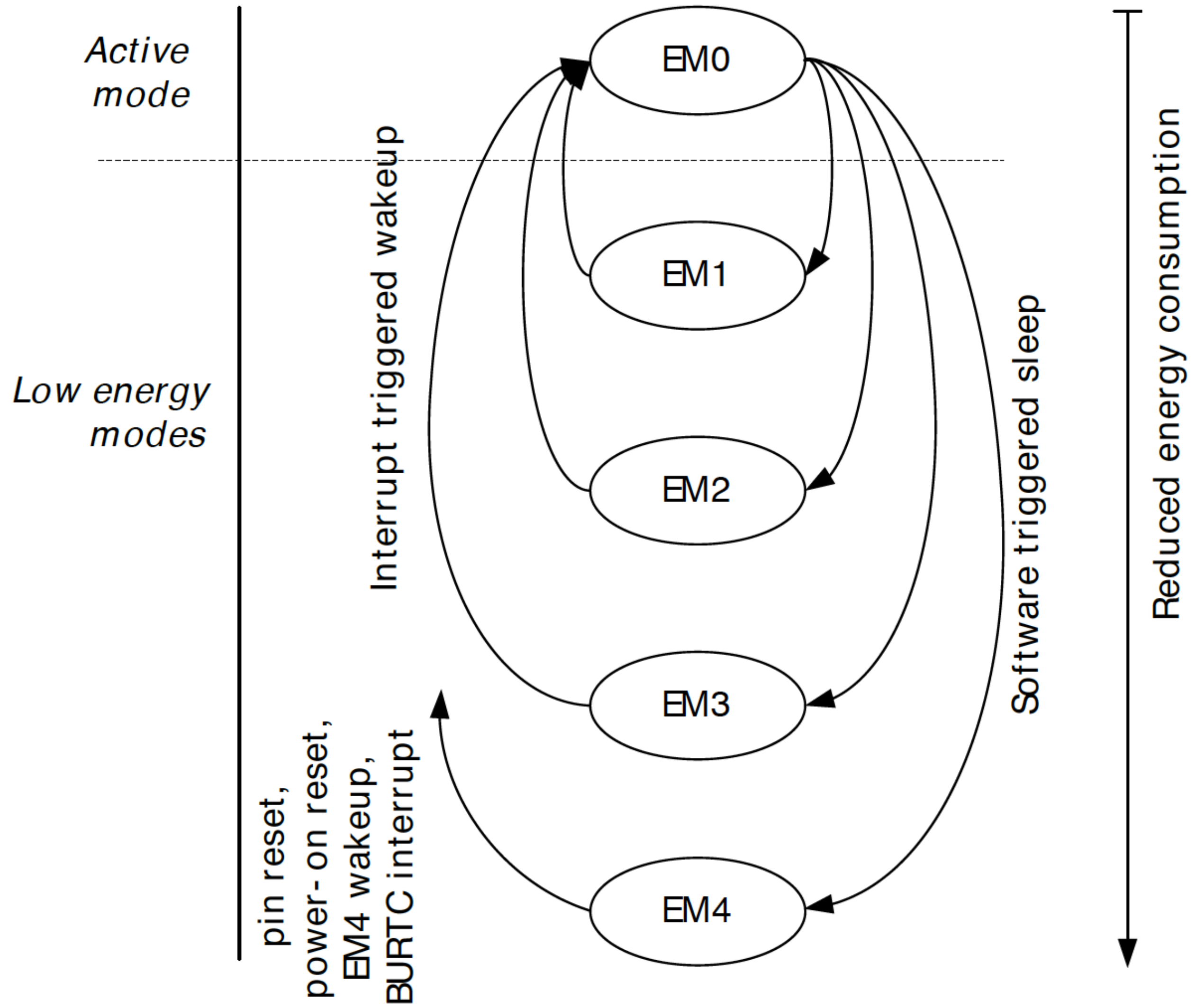
- This low energy mode has both high frequency and low frequency clocks stopped. Most peripherals are disabled or have reduced functionality. Memory and registers retain their values.
  - Cortex-M4 is in sleep mode. Clocks to the core are off
  - High frequency clock tree is inactive
  - Low frequency 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

Silicon Lab’s standard Energy Mode flow chart

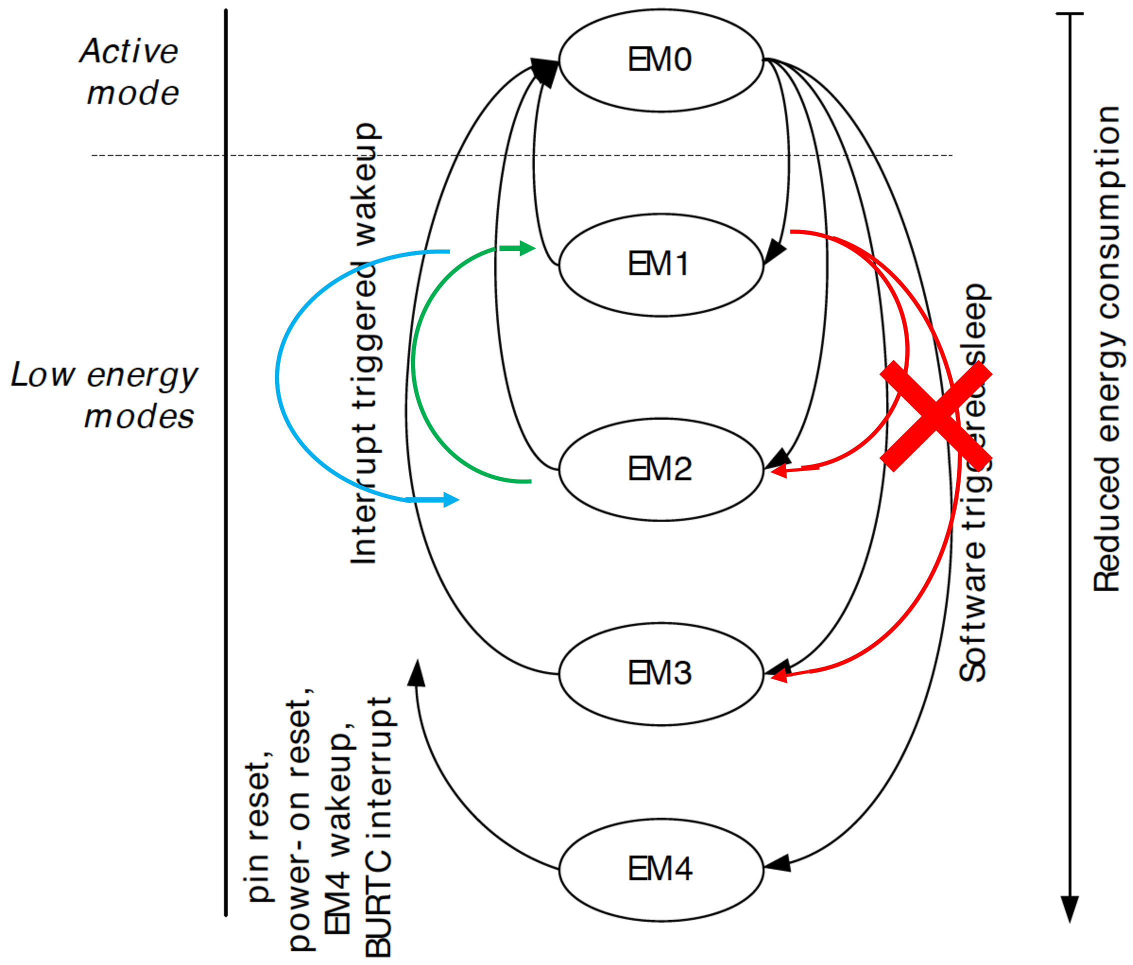
Figure 10.2. EMU Energy Mode Transitions


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

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
- 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)
{
  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();

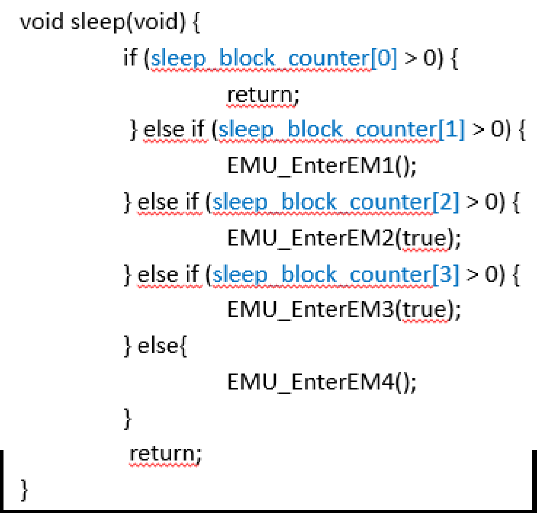
```
void sleep(void) {
        if (sleep_block_counter[0] > 0) {
                return;                              // Blocked everying below EM0, so just return
        } else if (sleep_block_counter[1] > 0) {
                EMU_EnterEM1();                      // Blocked everyithing below EM1, enter 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)
{
   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) {
                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.