ECEN 5023-001, -001B, -740

Mobile Computing & IoT Security

Lecture #2

19 January 2017

Be Boulder.

University of Colorado Boulder

Agenda

- Class Announcements
- Low Power versus Low Energy Design
- What Makes a Low Energy Microcontroller
- Clock tree
- How to address register's and their bits
- Silicon Labs' Exercise
- Energy modes

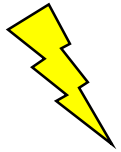
Class Announcements

- Quiz #1 is due at 11:59pm on Sunday, January 22nd , 2017
- Survey is due at 11:59pm on Sunday, January 22nd , 2017
- Register for ESE lab card access at: https://goo.gl/forms/YaBXQHATHELA2FIk2
- Waiting list is 2 students
  - Wait list students should be able to take the D2L quiz as well as submit assignments via the D2L drop box
  - Enrollment caps should be adjusted on Monday the 23rd or Tuesday the 24th
- Silicon Labs' STK3600 evaluation boards will be available after class

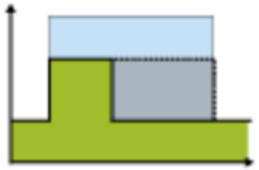
Low Energy

Energy = Power x Time

- Battery Capacity is measured in Energy such as mA-h
- To increase Batter Life, Energy must be reduced
  - Decrease Power
  - Decrease Time
- Low Energy firmware design minimizes both Power and Time



How does a microcontroller reduce energy?



- Reduce ON Time:
  - Highest energy consuming peripheral in a microcontroller is the CPU
  - A higher computational CPU will reduce the time required for a fix amount of work
  - Thus, a higher computational CPU => Reduced computational time
  - Resulting in minimizing CPU on time minimizes energy

How does a microcontroller reduce energy?

- MSP430
  - DMIPS/MHz = 0.288
  - 330uA/MHz active
  - Or $\frac{uA/MHz}{DMIPS/MHz}$
  - Or $\frac{330uA/MHz}{0.288DMIPS/MHz}$
  - 1,146uA/DMIP

Comparison of MSP430 and MSP432

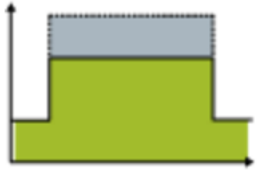
	MSP430	MSP430X	MSP432
Address space	16 bits	20 bits	32 bits
Memory address space	64kB	1MB	4GB
Clock speed	25 MHz		48 MHz
Floating Point	soft		IEEE754 32 bit FPU
Typical Dhrystone 2.1 (DMIPS/MHz)	0.288 [3]		1.196
ULPBench low power score	120		167.4

- Silicon Labs' Leopard Gecko Cortex-M3
  - DMIPS/MHz = 1.25
  - 216uA/MHz
  - 173uA/DMIP
- Cortex M3 is 662% more energy efficient than the MSP430
  - The Cortex-M3 is more energy efficient than the MSP430 not just due to the current per MHz (~ 50%), but due to the 400+% higher computational performance

https://en.wikipedia.org/wiki/TI_MSP430, https://en.wikipedia.org/wiki/TI_MSP432, http://www.ti.com/lit/ds/symlink/msp430f1611.pdf, http://www.silabs.com/Support%20Documents/TechnicalDocs/EFM32LG990.pdf

How does a microcontroller reduce energy?

- Reduce Power:
  - Very low active power consumption
    - Specifications can be in uA/MHz or mA per clock speed



Compared to the MSP430 from the previous slide at 330uA/MHz, the Cortex-M3 is a lower energy MCU architecture

- Silicon Labs' EFM32LG (Leopard Gecko) Cortex-M3

14 MHz HFRCO, all peripheral clocks disabled, V_{DD}= 3.0 V, T_{AMB}=25°C		216		µA/ MHz

- NXP LPC15xx Cortex-M3

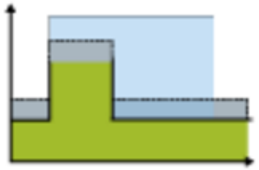
I_{DD}	supply current	Active mode; code `while(1){}` executed from flash;				
		system clock = 12 MHz; default mode; V_{DD} = 3.3 V	[3][4][5] [7][8]	-	4.3	-	mA
		system clock = 12 MHz; low-current mode; V_{DD} = 3.3 V	[3][4][5] [7][8]	-	2.7	-	mA

  - 2.7mA / 12MHz = 0.225mA/MHz or 225uA/MHz

http://www.silabs.com/Support%20Documents/TechnicalDocs/EFM32LG990.pdf
http://www.nxp.com/documents/data_sheet/LPC15XX.pdf
http://www.nxp.com/documents/user_manual/UM10736.pdf

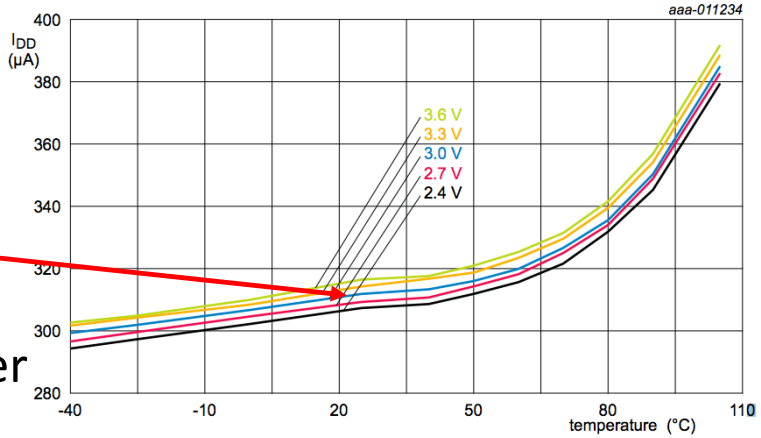
How does a microcontroller reduce energy?

- Reduce Power:
  - Ultra-Low power sleep modes
    - Silicon Labs' EFM32LG (Leopard Gecko) Cortex-M3



I_{EM2}	EM2 current	EM2 current with RTC prescaled to 1 Hz, 32.768 kHz LFRCO, V_{DD}= 3.0 V, T_{AMB}=25°C		0.95[1]	1.7[1]	µA

    - NXP LPC15xx Cortex-M3
      - Deep-sleep mode is similar EM2 above
      - 310uA at 25C
    - If being in sleep mode is a significant period of operation, then the Leopard Gecko may be a better choice



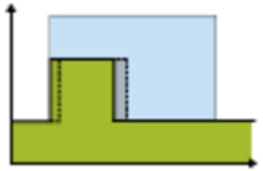
Conditions: BOD disabled; all oscillators and analog blocks disabled. Use API power_mode_configure() with mode parameter set to DEEP_SLEEP and peripheral parameter set to 0xFF.

Fig 21. Deep-sleep mode: Typical supply current I_{DD} versus temperature for different supply voltages V_{DD}

http://www.silabs.com/Support%20Documents/TechnicalDocs/EFM32LG990.pdf
http://www.nxp.com/documents/data_sheet/LPC15XX.pdf
http://www.nxp.com/documents/user_manual/UM10736.pdf

How does a microcontroller reduce energy?

- Reduce Power:
  - Fast wake up times from sleep or low energy modes
    - Time to wake up from a low energy mode is wasted energy -> no work is being accomplished
      - Silicon Labs' EFM32LG (Leopard Gecko) Cortex-M3

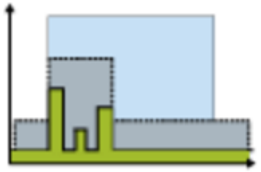


t_{EM20}	Transition time from EM2 to EM0		2		µs

- NXP LPC15xx Cortex-M3
  - "In Deep-sleep mode, the LPC15xx is in Sleep-mode and all peripheral clocks and all clock sources are off except for the IRC. The IRC output is disabled unless the IRC is selected as input to the watchdog timer. "

http://www.silabs.com/Support%20Documents/TechnicalDocs/EFM32LG990.pdf
http://www.nxp.com/documents/data_sheet/LPC15XX.pdf
http://www.nxp.com/documents/user_manual/UM10736.pdf

How does a microcontroller reduce energy?

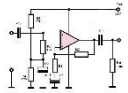
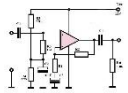


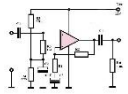
- Reduce Power:
  - Autonomous Peripherals
    - Highest energy consuming peripheral in a microcontroller is the CPU
    - Do more work with the CPU off, the more energy efficient
      - Silicon Labs' EFM32LG (Leopard Gecko) Cortex-M3
        - Peripherals operating in EM2 mode (not complete list):
          - USB, I2C, Low Energy UART, Low Energy Sense, Low Energy Timer, Interrupts, DAC, Analog Comparators, Voltage Comparators
      - NXP LPC15xx Cortex-M3
        - Peripherals operating in Deep-Sleep mode (not complete list):
          - USB, I2C, SPI, USART, RTC, Interrupts

http://www.silabs.com/Support%20Documents/TechnicalDocs/EFM32LG990.pdf
http://www.nxp.com/documents/data_sheet/LPC15XX.pdf
http://www.nxp.com/documents/user_manual/UM10736.pdf

What are the characteristics that the firmware engineer can take advantage?

- Low Energy Microcontroller characteristics:
  - Higher Computational CPU
  - Very Low Active Power Consumption
  - Ultra Low Power Sleep Modes
  - Fast wake up times from sleep or low energy modes
  - Autonomous Peripherals




Additional considerations to select a Low Energy Microcontroller

- Advanced autonomous peripheral functions:
  - Passive sensor state machines - LESENSE
  - Peripheral Intercommunication - PRS
- Well architected Energy Modes
- Energy or Current monitors

Advanced autonomous peripheral functions

- Passive sensor state machines
  - State machine equates to no CPU resources required
    - Highest energy peripheral, CPU, can be turned off
  - Passive equates to no active energy required
    - Energy is only required when sensor is turned on
  - Silicon Labs Leopard Gecko's Low Energy Sensor Interface (LESENSE)
    - Up to 16 passive sensor
    - Excites, Measures, and Decodes passive sensors without the CPU
    - Operates down to Energy Mode 2 with the CPU turned off

Advanced autonomous peripheral functions

- Peripheral Intercommunication
  - Create new autonomous functions by combining individual autonomous peripherals
  - Increase functionality without CPU involvement allowing the microcontroller to remain in a low energy state
  - Silicon Labs Leopard Gecko Peripheral Reflex System
    - 12 interconnect channels
    - Any producing peripheral can connect to any consuming peripheral
    - Example of ”new” autonomous function
      - LESENSE ambient light sensor can trigger an ADC conversion whose result is stored in memory via DMA
      - Complete operation can occur while in EM1 with no CPU involvement
        - 938uA typically versus 3,024uA in EM0 with the CPU

Well architected Energy Modes

- Well architected Energy Modes equates to the appropriate peripherals for your application that will enable you to remain in the lowest possible energy mode for the longest period of time
  - For example, if an Analog Comparator is required to monitor the system to initiate an activity, having this peripheral available in a low energy state would be important
    - The Silicon Labs Leopard Gecko enables Analog Comparator down to EM3

I_{EM3}	EM3 current	V_{DD}= 3.0 V, T_{AMB}=25°C		0.65	1.3	µA
		V_{DD}= 3.0 V, T_{AMB}=85°C		2.65	4.0	µA

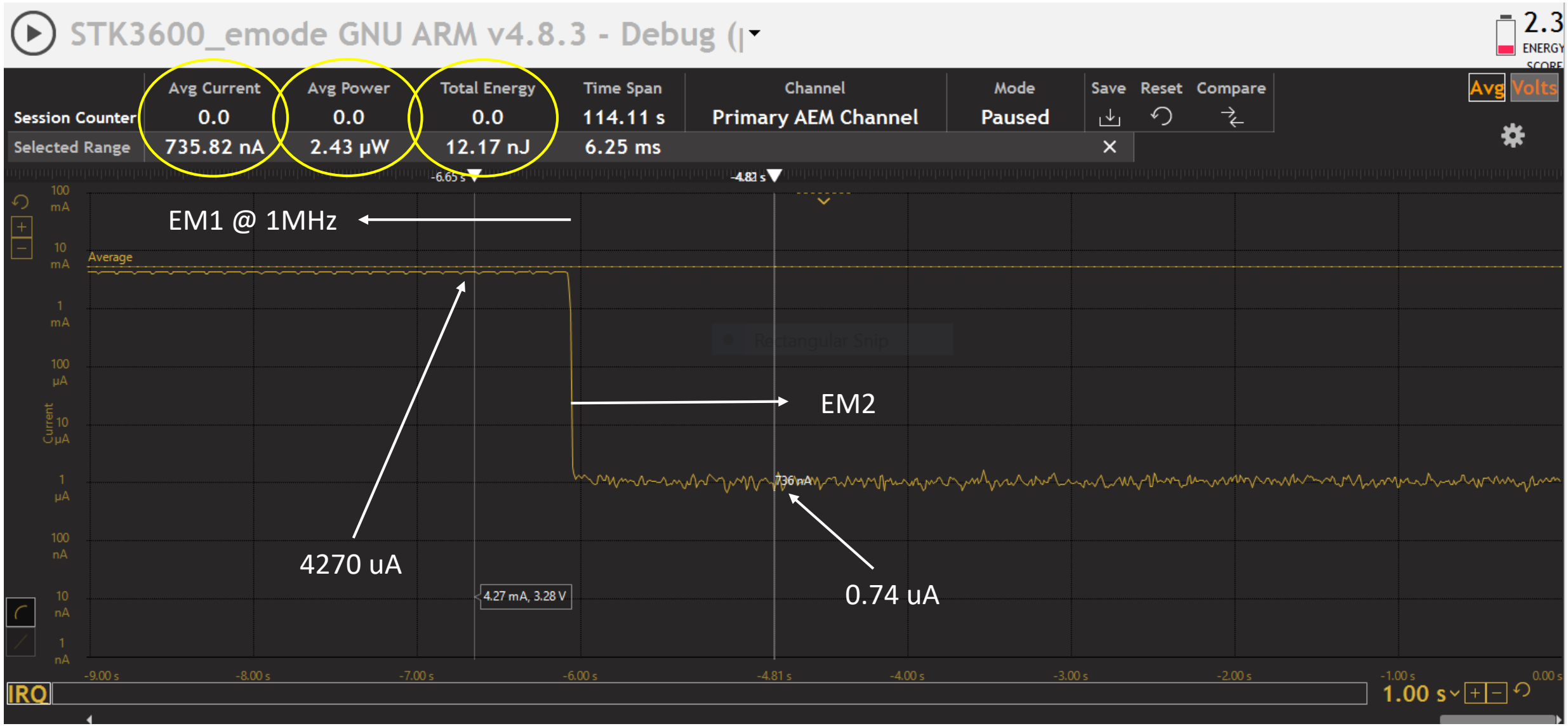
    - While the NXP LPC15xx enables its Analog peripherals only to Sleep Mode

system clock = 12 MHz; low-current mode; V_{DD} = 3.3 V	[3][4][5] [7][8]	-	2.7	-	mA

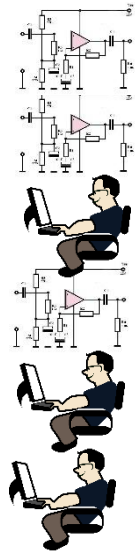
    - Silicon Labs' ACMP lower energy mode is typically 42 times more energy efficient

Energy or Current Monitors

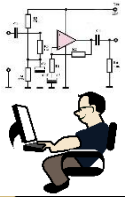
- Energy or Current Monitors provide real time data on the use of energy
- Can be used to determine which routines consume a significant amount of energy by using code correlation
  - Enables the firmware engineer to pinpoint where the code is spending energy
  - And, thus, focus attention to reduce energy in those routines
- Verifies the energy efficiency of the firmware design


STK3600_emode GNU ARM v4.8.3 - Debug
2.3
ENERGY
Avg Current
Avg Power
Total Energy
Time Span
Channel
Mode
Save
Reset
Compare
Avg
Volts
Session Counter
0.0
0.0
0.0
114.11 s
Primary AEM Channel
Paused
Selected Range
735.82 nA
2.43 µW
12.17 nJ
6.25 ms
-6.65 s
-4.81 s
EM1 @ 1MHz
Average
EM2
4270 uA
0.74 uA
4.27 mA, 3.28 V
736 nA
Current
100 mA
10 mA
1 mA
100 µA
10 µA
1 µA
100 nA
10 nA
1 nA
-9.00 s
-8.00 s
-7.00 s
-6.00 s
-4.81 s
-4.00 s
-3.00 s
-2.00 s
-1.00 s
0.00 s
IRQ
1.00 s

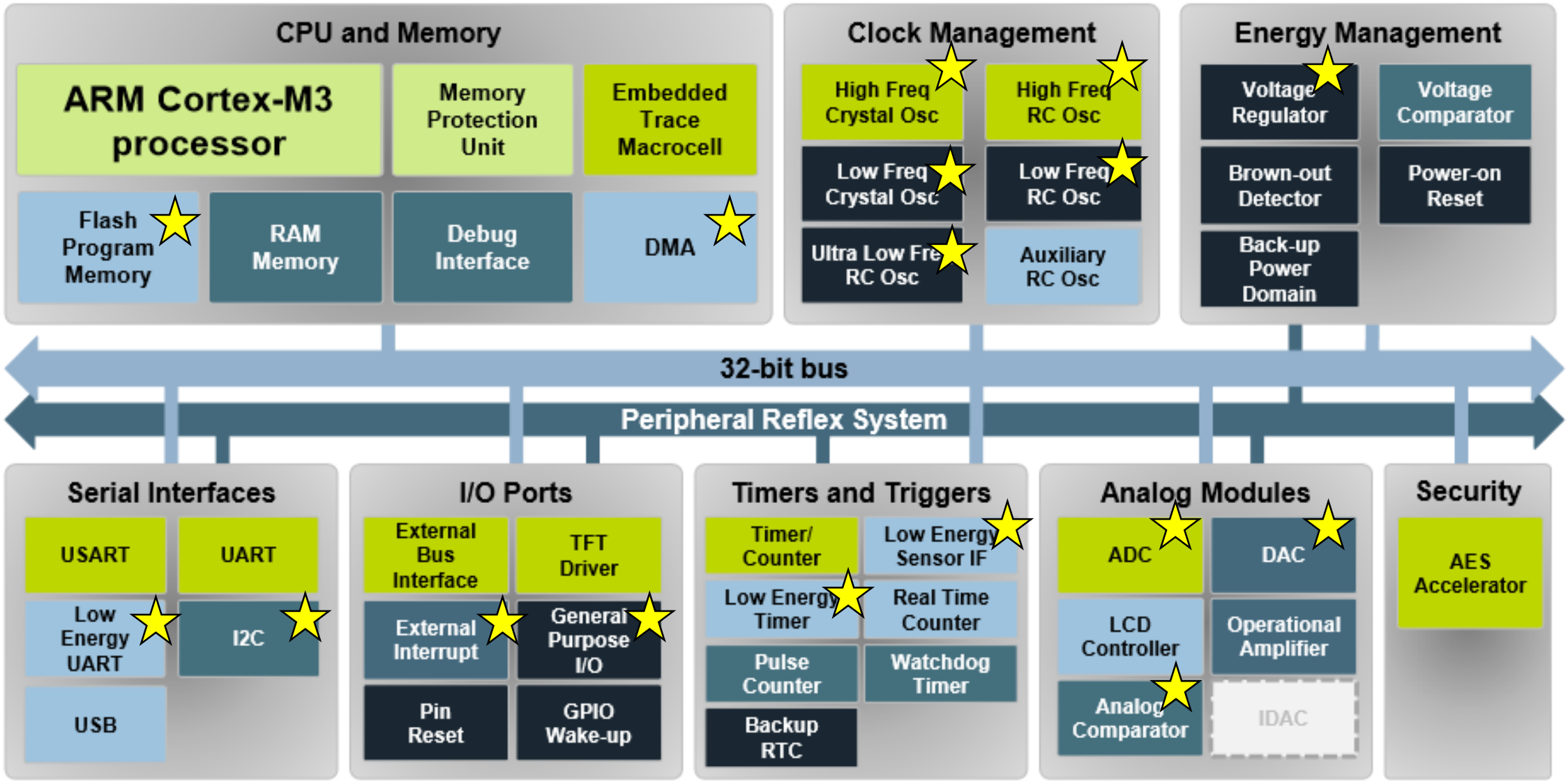
What are the characteristics that the firmware engineer can take advantage?



- Low Energy Microcontroller characteristics:
  - Higher Computational CPU
  - Very Low Active Power Consumption
  - Ultra Low Power Sleep Modes
  - Fast wake up times from sleep or low energy modes
  - Autonomous Peripherals
- Advanced autonomous peripheral functions:
  - Passive sensor state machines - LESENSE
  - Peripheral Intercommunication - PRS

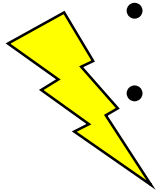


- Well architected Energy Modes
- Energy or Current monitors


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 Fre 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

Ideal CMOS FET

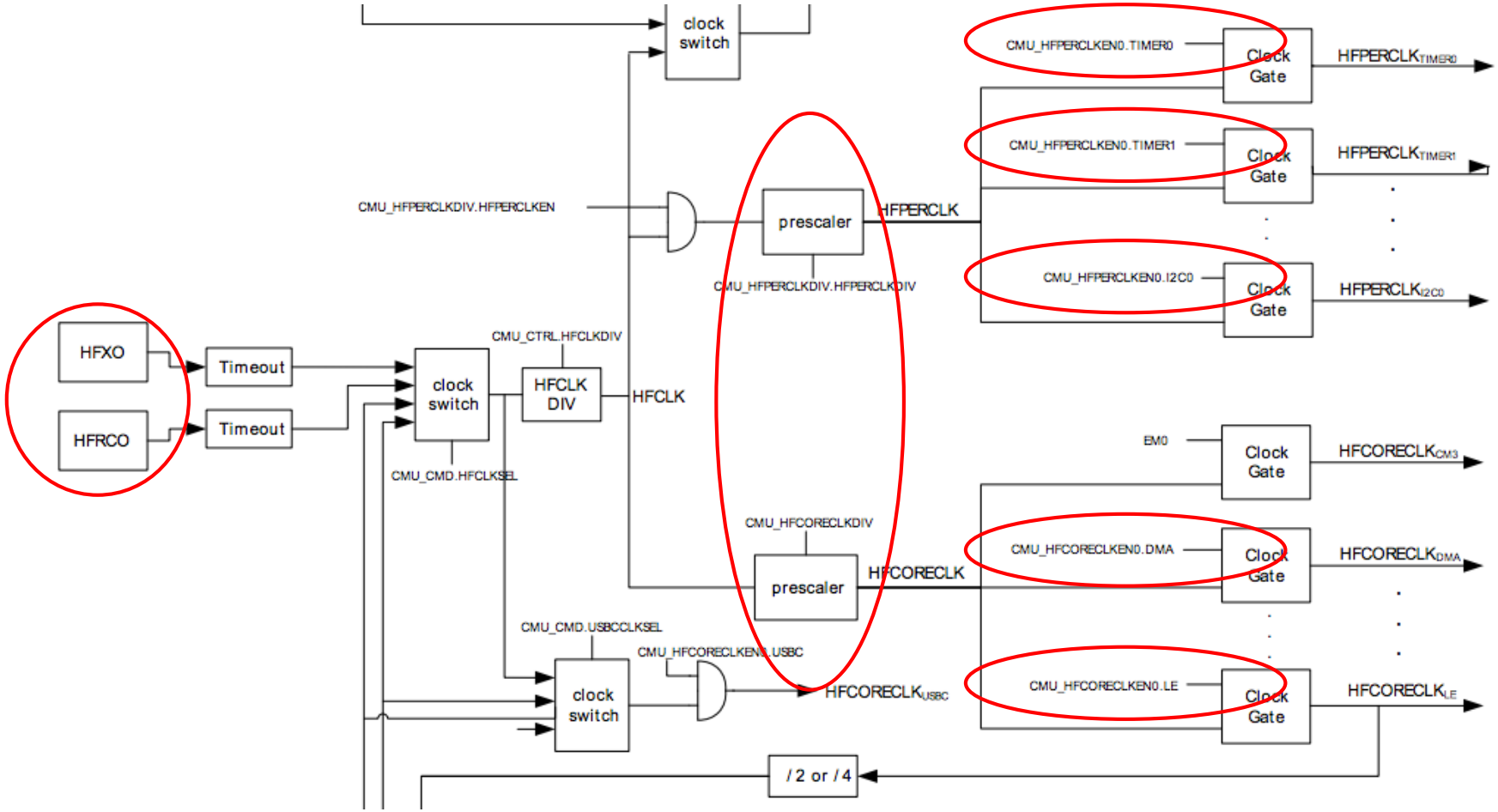
- CMOS logic reduces power consumption because no current flows (ideally), and thus no power is consumed, except when the inputs to logic gates are being switched. CMOS accomplishes this current reduction by complementing every nMOSFET with a pMOSFET and connecting both gates and both drains together.
- 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



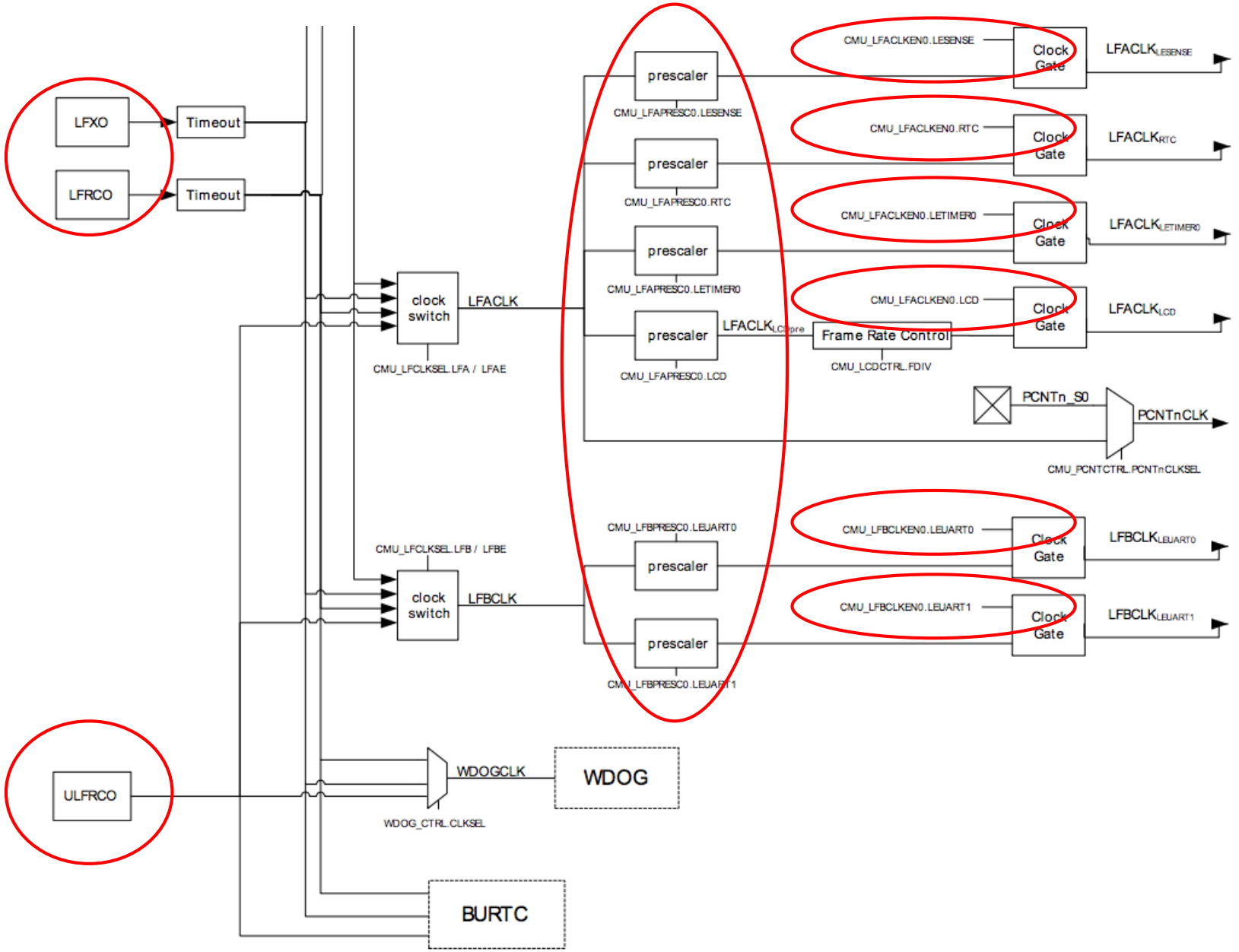
Clock tree assumptions

- Goal of the Silicon Labs' Leopard Gecko is low power / low energy applications
- To achieve this goal:
  - Clock sources of various frequencies, tolerances, and power are available
    - 1-28 MHz High Frequency RC Oscillator (HFRCO)
    - 4-48 MHz High Frequency Crystal Oscillator (HFXO)
    - 32.768 Hz Low Frequency RC Oscillator (LFRCO)
    - 32.768 Hz Low Frequency Crystal Oscillator (LFXO)
    - 1 kHz Ultra Low Frequency RC Oscillator (ULFRCO)
  - Prescalers on each clock tree to optimize energy to required performance
  - Clock enables to only consume energy on peripherals required for the application
    - By default, each peripheral clock is disabled / turned off

Leopard Gecko High Frequency Clock Tree

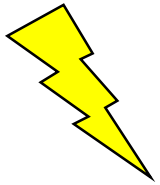


Leopard Gecko Low Frequency Clock Tree



CMU – Clock Management Unit

- The Clock Management Unit (CMU) is responsible for controlling the oscillators and clocks in the EFM32.
  - 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.

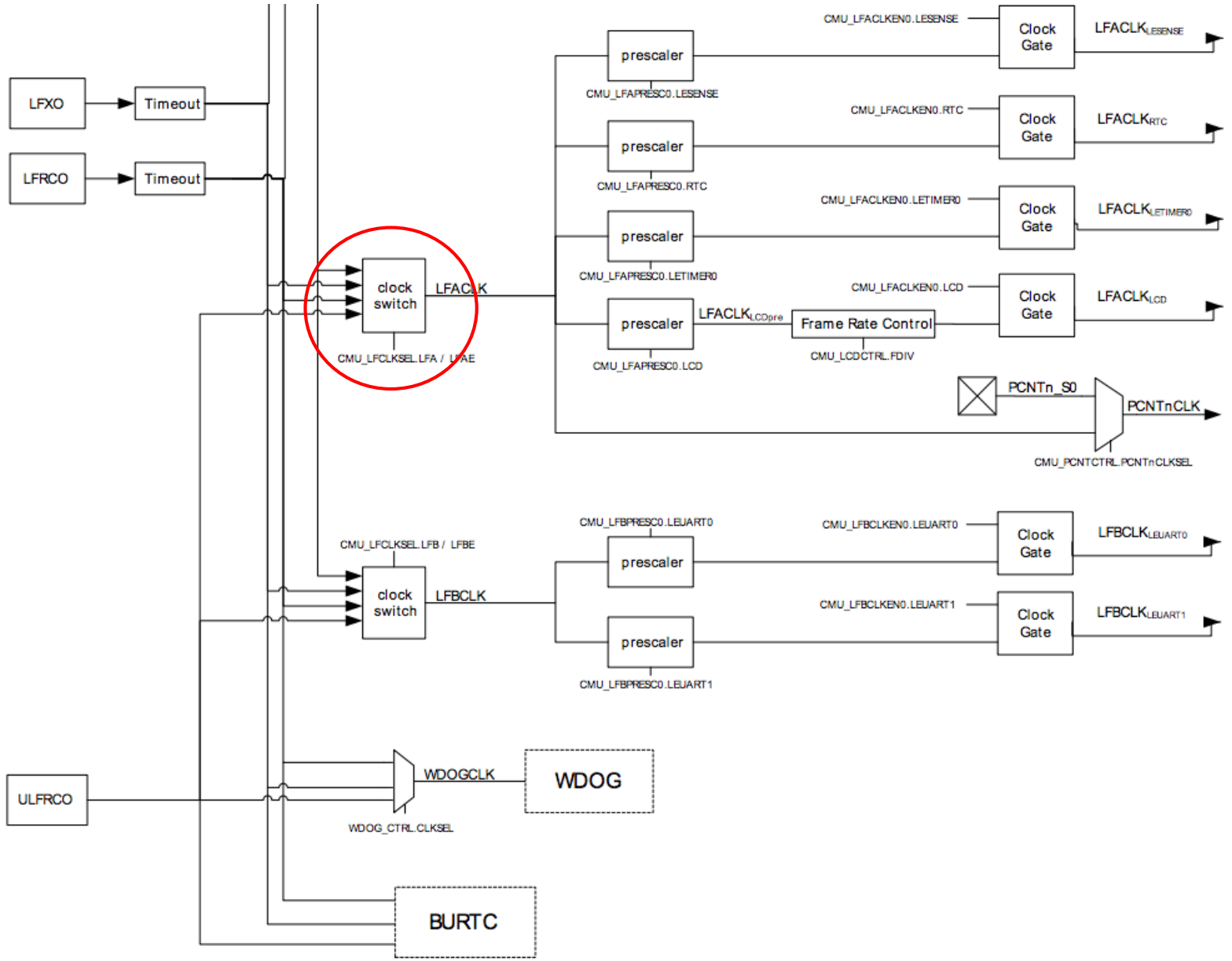


Enabling a peripheral

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

Leopard Gecko Low Frequency Clock Tree

Clock Source Selection

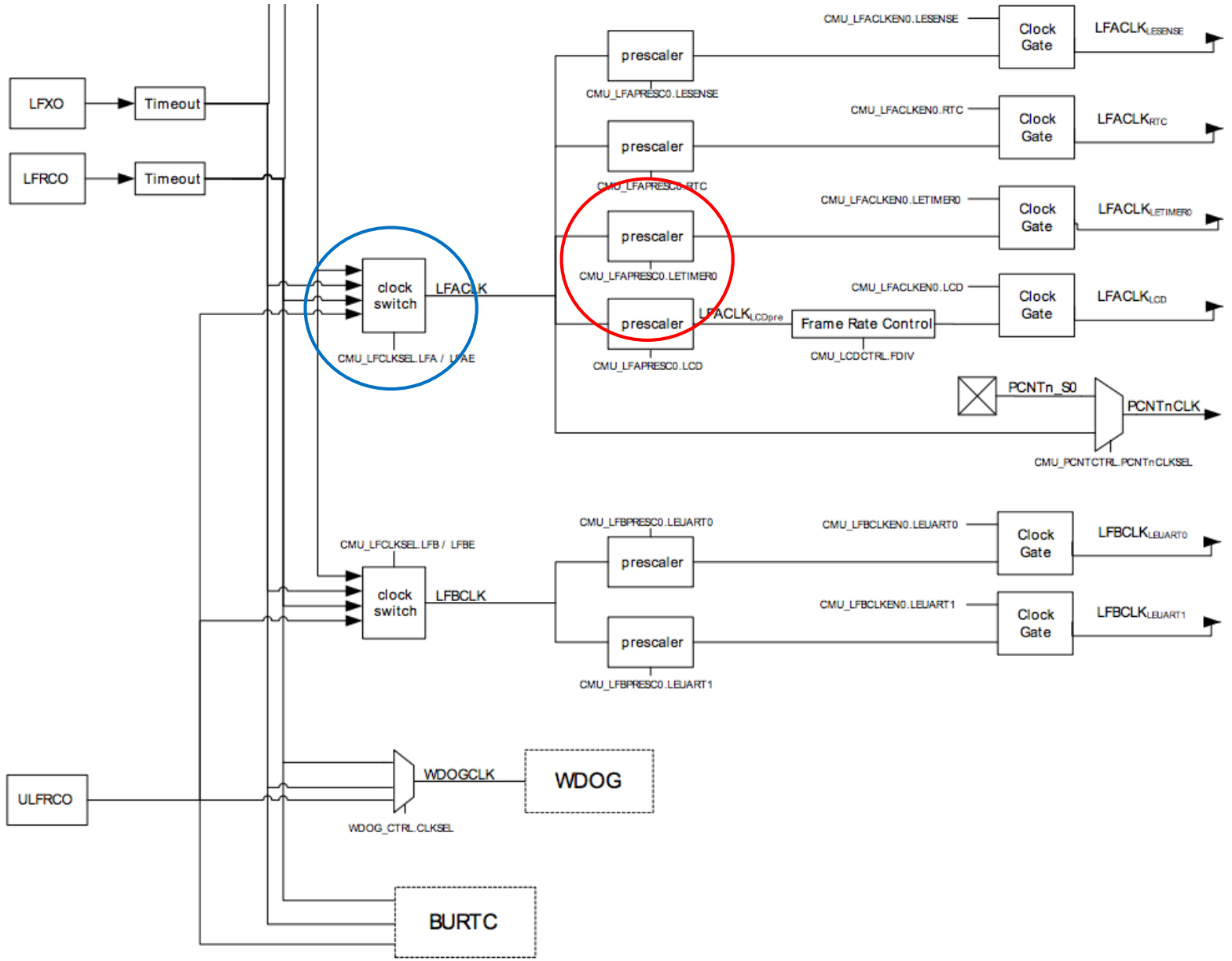


Enabling a peripheral – Clock Source

- 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)

Leopard Gecko Low Frequency Clock Tree

Prescaler

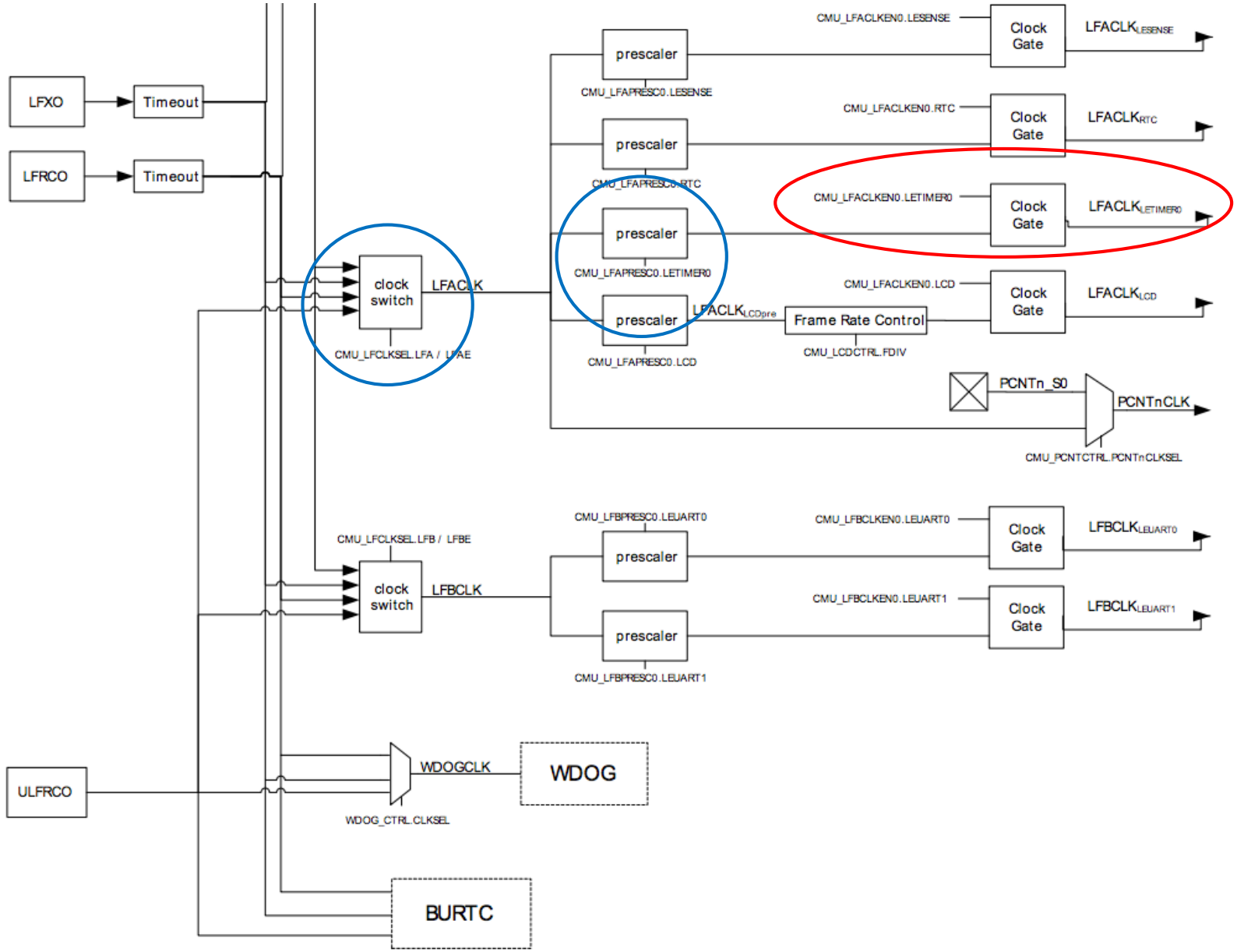


Prescaling

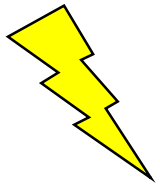
- emlib to set the clock branch prescaler
  - CMU_ClockDivSet(CMU_Clock_TypeDef clock, CMU_ClkDiv_TypeDef div)
  - Note that not all clocks can be prescaled or prescaled to the same level

Leopard 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

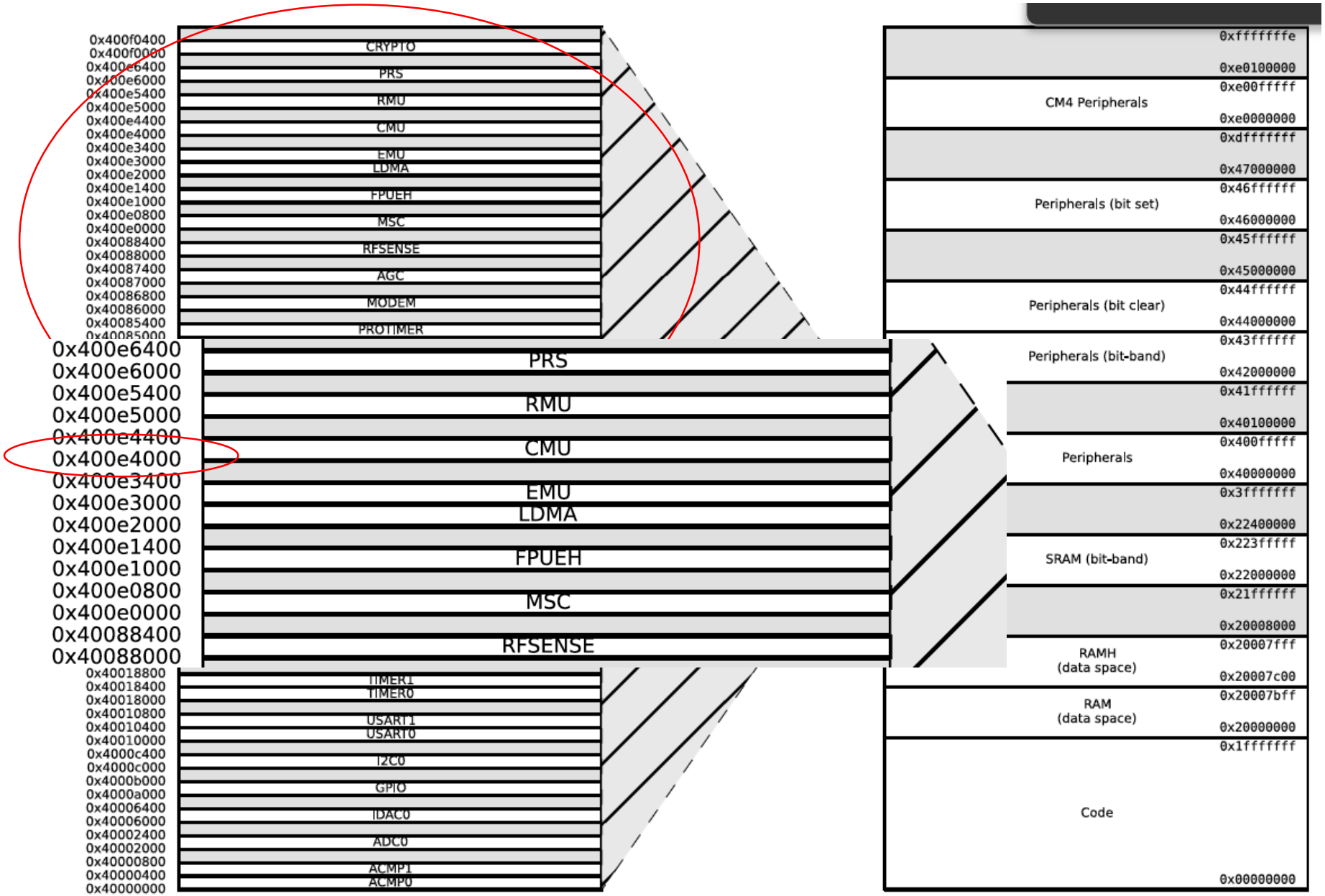
Offset	Bit Position																											
0x034	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											R	R	R	R	R	R
Name																											REP1	REP0	COMP1	COMP0	CMD	CTRL

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**

Accessing registers directly using C-code

CMU base address is:

0x400e4000



Accessing registers directly using C-code

CMU Interrupt Enable register offset address 0x0AC

Offset	Name	Type	Description
0x000	CMU_CTRL	RW	CMU Control Register
0x010	CMU_HFRCOCTRL	RWH	HFRCO Control Register
0x018	CMU_AUXHFRCOCTRL	RW	AUXHFRCO Control Register
0x020	CMU_LFRCOCTRL	RW	LFRCO Control Register
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 0x400e8000
  - IEN register offset + 0x000000aC
  - CMU->IEN = 0x400e80aC

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

11.5.26 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 0x400e8000
  - IEN register offset + 0x000000aC
  - CMU->IEN = 0x400e80aC
- 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 0x400e80aC 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

11.5.28 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;

HFRCO Band Selection

- The default HF clock source is set to the HFRCO @ 14 MHz
- The HFRCO can be changed to 1, 7, 11, 21, or 28 MHz
- Note: Higher the frequency does not mean faster operation. Above 16MHz, the flash wait state needs to be increased from 0 to 1.
- emlib routine to set the HFRCO band:
  - CMU_HFRCOBandSet(CMU_HFRCOBand_TypeDef band)

Example project from fall 2016

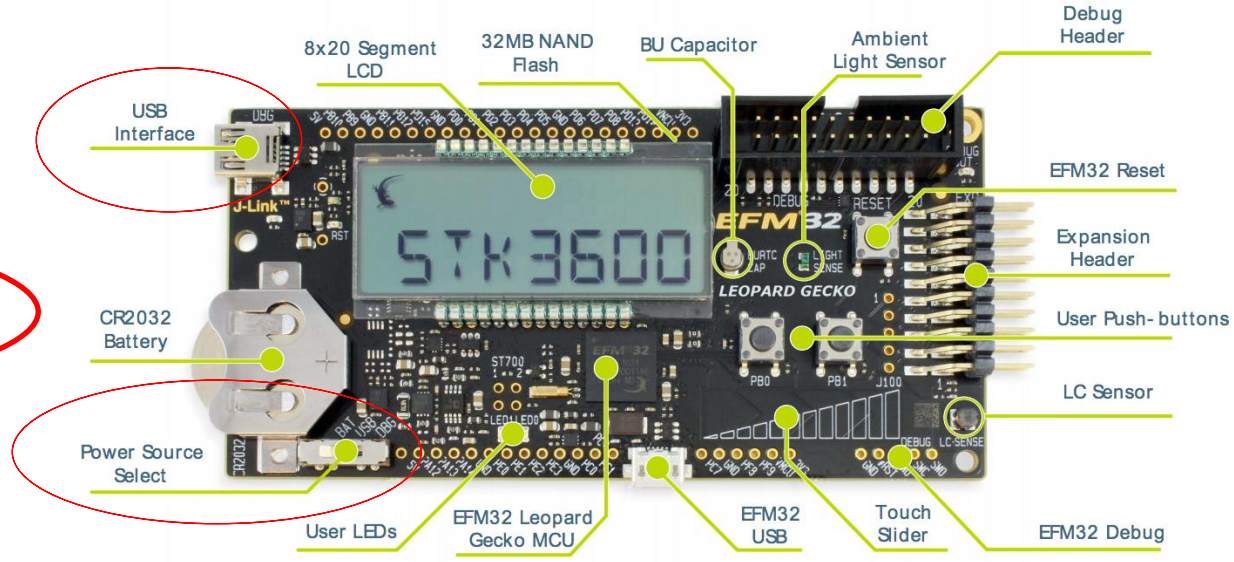
Simplicity Studio Exercise

Objective: Install and become familiarized with the Silicon Labs' Simplicity Studio development environment

Instructions:

1. Install Silicon Labs' Simplicity Studio 4 development environment. You can download the software from the following site:
   a. http://www.silabs.com/products/mcu/Pages/simplicity-studio.aspx
   b. Insure that you select all EFM32 files as a minimum
2. Connect your Silicon Labs' Leopard Gecko STK3600 starter kit, USB Interface, to the computer
3. Insure that the Power Source Select is to the DBG position

Due: Wednesday, January 25th, 2017 at 11:59pm



Silicon Lab STK3600 User Manual

Simplicity Studio Exercise

4. Insure you are in the Launcher view, and click on Solutions in the upper left corner, and then click on Add Devices. In the Add Devices window, type stk3600 to select the Leopard Gecko 3600 starter kit or click on the EFM32 Leopard Gecko Starter Kit board in detected hardware.

5. On the left hand column under Solutions, click on the stk3600 to make it the active Solution.

6. In the middle window under "Software Examples," locate STK3600_blink example and click it

7. Click Yes to create the example project.

8. The program should open the Simplicity IDE and highlight blink.c of the example program.

Simplicity Studio Exercise

9. Now, click on “Run” in the toolbar, and then click on “Profile.”
   a. Simplicity IDE should begin to compile the project, and then download and flash the code into Leopard Gecko on the STK3600
   b. After flashing the microcontroller, the code should begin to run, and the LED on the STK3600 should begin to flash
   c. Simplicity should now open the Energy Profiler

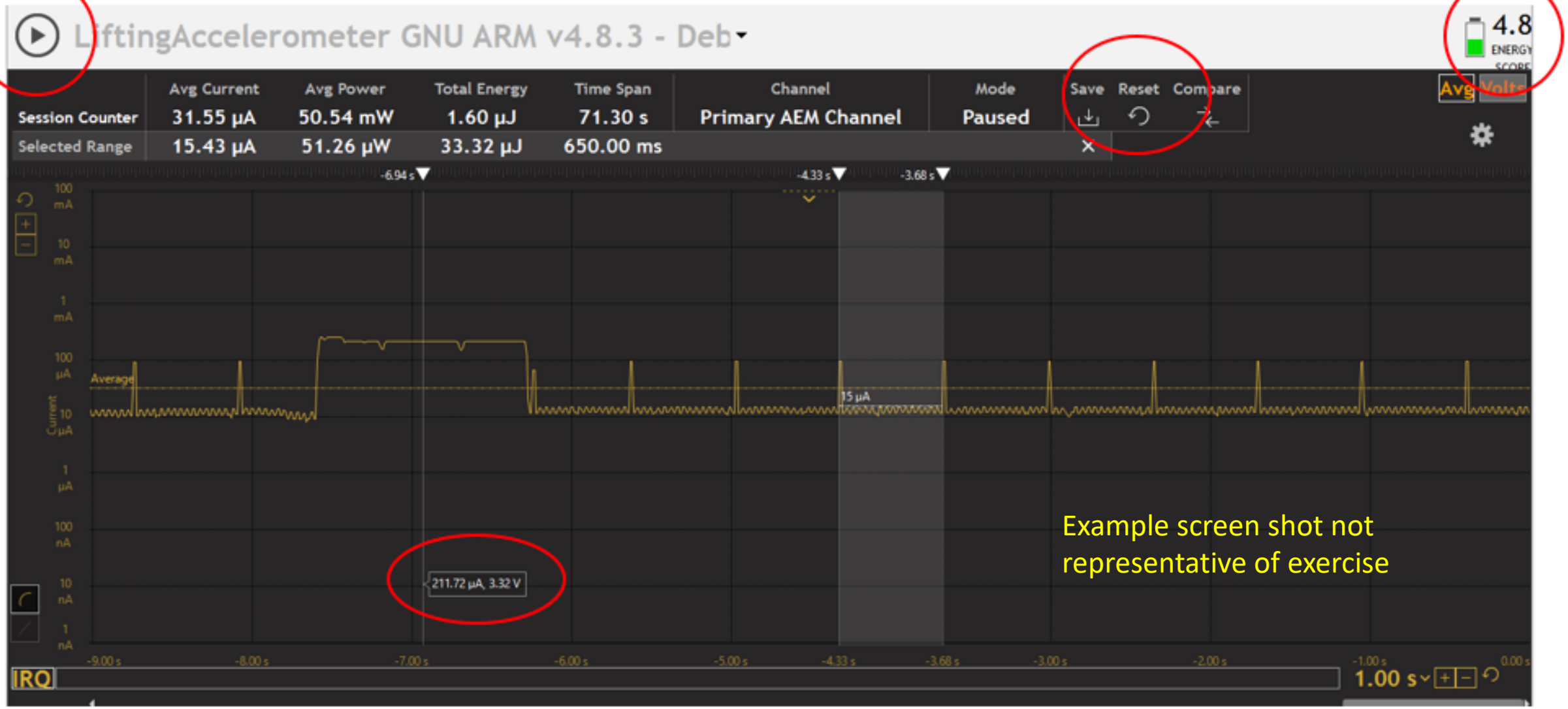
10. In the Energy Profiler, click the pause button towards the upper left corner.

11. Click once somewhere after the program has started. Towards the bottom of the marker, the instantaneous current measurement can be found.

12. Click the play button towards the upper left corner to restart the Energy Profiler measurements.

13. Towards the right end of the session counter, click on the counter clockwise arrow to reset the session counters and the Energy Profiler score. Wait 30 seconds after resetting to determine the Energy Score.

Simplicity Studio Exercise



Example screen shot not representative of exercise

Simplicity Studio Exercise

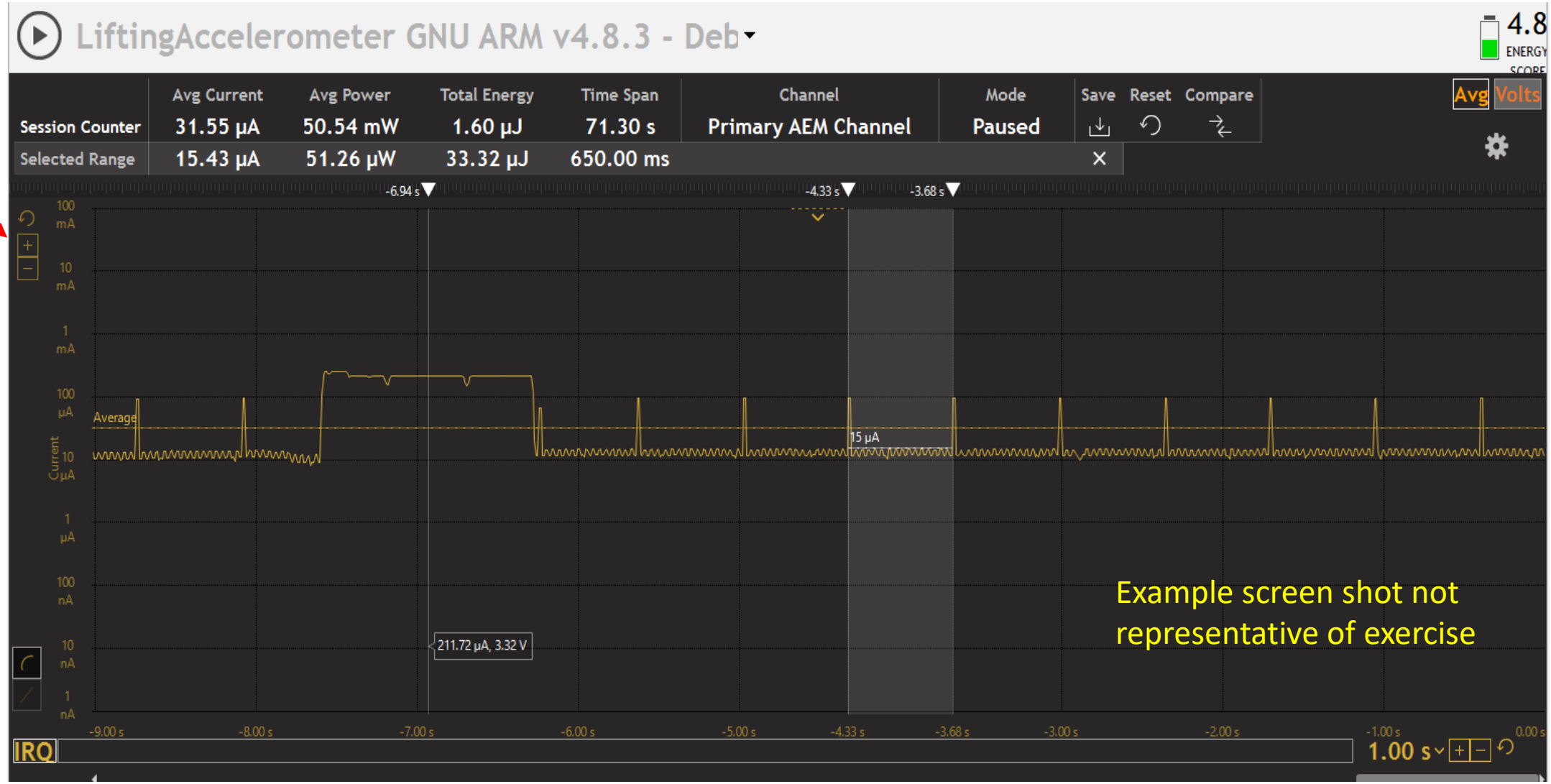
14. Now go back into the Simplicity IDE, and comment out the following line of code in the Blinking LED Blink.c routine

    BSP_LedToggle(1); => //BSP_LED_Toggle(1);

15. Now click on Run for Simplicity Studio to compile, flash, and start the updated program.

16. Find in the Energy Profiler where the LED0 is ON. You may need to zoom in the vertical scale by clicking the following button.

Simplicity Studio Exercise



Simplicity Studio Exercise

17. After commenting out the code to toggle LED1, what is the instantaneous current when LED0 is on?

18. After commenting out the code to toggle LED1, what is the instantaneous current when LED0 is off?

19. After commenting out the code to toggle LED1, what is the Energy Score after resetting the Energy Profiler and awaiting 30 seconds?

Simplicity Studio Exercise - Questions

- In a separate document to be placed in the drop box with the program code, please answer the following questions:
  - Using the Energy Profiler, what is the instantaneous current measured once the program has begun without any modification to the sample code?
  - What is the Energy Score after resetting the Energy Profiler and waiting 30 seconds without any modification to the sample code?
  - After commenting out the code to toggle LED1, what is the instantaneous current measured while LED0 is off?
  - After commenting out the code to toggle LED1, what is the instantaneous current measured while LED0 is on?
  - After commenting out the code to toggle LED1, what is the Energy Score after resetting the Energy Profiler and waiting 30 seconds?

Simplicity Studio Exercise - Deliverables

- In the D2L drop box for this assignment, please include two files:
  - Answers to the 5 assignment questions
  - Sample program with the modifications assigned to answer question 5
    - Easiest way to submit is copying the code from the Simplicity Studio program and pasting it into a Word or notepad file

Today's Summary

- Class Announcements
- Low Power versus Low Energy Design
- What Makes a Low Energy Microcontroller
- Clock tree
- How to address register's and their bits
- Energy modes
- Assignment

Discussion topics for next lecture

- Class Announcements
- Quiz review
- Keeping track of the Energy Mode
- Designing Interrupt Based Code Using Low Energy Timer
- Mobile/Pervasive Adaptive Computing System Considerations
- Network considerations
- Documentation style sheet