Energy Micro University

UM006 - Energy Modes

This lesson aims to provide insight into the different energy modes of EFM32 microcontrollers. The different modes have fewer features available as the mode goes from 0 to 4 and hence decreasing energy consumption. Choosing the mode that provides the desired functionality and uses the least energy makes the application energy friendly.

Concepts that will be introduced include:

- **Energy Modes (EM0 to EM4)**
- **Energy Management Unit (EMU)**
- **Clock Management Unit (CMU)**
- **Advanced Energy Monitoring (AEM)**
- **energyAware Profiler**

This lesson includes:

- **This PDF document**
- **Source files**
  - **Example C code**
  - **Multiple IDE projects**

1 Saving Energy

In battery powered microcontroller applications, energy saving is essential. By reducing the overall energy need, the mean time between battery charging/replacement can be significantly increased. Applying the following principles will drastically reduce the energy consumption:

- Turn off unused modules / peripherals
- Disable clocks to unused modules / peripherals
- Reduce clock frequency

The EFM32 supports extensive usage of all these principles. It is important to note that even though reducing the current consumption will reduce the power needed, this might not be the best overall energy strategy. If half the power is needed, but the MCU has to work three times as long to perform the task, the reduction in power actually increased the overall energy consumption.

1.1 Turn off unused modules/peripherals

At a given time in every microcontroller application, there are modules/peripherals which are not used. By turning these off, current will be saved.

If a module is used part-time (e.g. the ADC performs a temperature measurement each second), it will be more energy efficient to only enable the module when it is used and keep it turned off otherwise. This way, extra current is only consumed when necessary and idle current is kept at a minimum.

This also applies to the CPU itself. If the core is idle (e.g. waiting for data reception), it can be turned off and energy is saved. This is one of the main features of the energy modes of the EFM32.

Remember to take startup/stop overhead into consideration, both in terms of time and power consumption.

1.2 Disable clocks to unused modules/peripherals

Even though a module/peripheral is disabled (e.g. TIMER0 is stopped), energy will still be consumed in that module if its clock is running. Therefore, it is important to turn off the clocks of all unused modules.

1.3 Reduce clock frequency

Current consumption scales with clock frequency. In general, a certain task or module should be operated at the lowest possible frequency. E.g. if a timer is to give an interrupt each ms, it is better to run it at a few kHz, rather than several MHz. This can easily be implemented using the prescaling functionality in the Clock Management Unit (CMU).

Likewise, one approach regarding the CPU frequency is that it should be so low that the CPU is not idle (some margin should be added). However, in many cases it is by far better to complete the current tasks quickly and then enter a suitable energy mode until new tasks must be handled. The different energy modes of the EFM32 are optimized for this purpose.

2 Energy Modes

The different energy modes of the EFM32 differ in energy consumption, CPU activity, active peripherals and clocks. A presentation of each energy mode is given below.

2.1 Run Mode (Energy Mode 0)

This is the default mode, i.e. the startup mode as well as the mode the MCU returns to after being in other Energy Modes. In this mode the CPU fetches and executes instructions from flash or RAM, and all peripherals may be enabled.

2.2 Sleep Mode (Energy Mode 1)

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. By using the Peripheral Reflex System (PRS) and DMA, several operations can be performed autonomously. 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.

Sleep Mode is entered by executing either the "Wait for Interrupt (WFI)" or the "Wait for Event (WFE)" instruction. This will turn off the CPU. Alternatively, Energy Mode 1 can be entered by calling the emlib function

```
EMU_EnterEM1()
```

2.3 Deep Sleep Mode (Energy Mode 2)

In the Deep Sleep Mode no high frequency oscillators are running, i.e. only asynchronous or low frequency peripherals are available. This mode is extremely energy efficient, and may be used for a wide range of useful cases. A few examples:

- The LCD controller drives an LCD.
- The Low Energy Universal Asynchronous Receiver/Transmitter (LEUART) is receiving or transmitting a byte.
- The RTC is counting and will wake up the CPU after a programmed number of ticks.
- An Analog Comparator (ACMP) is comparing a voltage to a programmed threshold.
- GPIO is checking for transitions on an I/O line.

Deep Sleep Mode is entered by first setting the SLEEPDEEP bit in the System Control Register (SCR) and then executing either the "Wait for Interrupt (WFI)" or the "Wait for Event (WFE)" instruction. Energy Mode 2 can also be entered by calling the emlib function

```
EMU_EnterEM2(bool restore)
```

The argument sets whether or not clocks and oscillators are to be restored at wakeup.

2.4 Stop Mode (Energy Mode 3)

Stop Mode differ from Deep Sleep Mode in that no oscillator (except the ULFRCO to the watch dog) is running.

Modules available in Stop Mode are:

- Inter-Integrated Circuit Interface (I^2C) Address Check
- Watchdog
- Asynchronous Pin Interrupt
- Analog Comparator (ACMP)
- Voltage Comparator (VCMP)
- The RTC is counting and will wake up the CPU after a programmed number of ticks (if ULFRCO is selected).
- GPIO asynchronous interrupts

Stop Mode is entered the same way as Deep Sleep Mode, except that also the low frequency oscillators are turned off. The oscillators must be disabled "manually" by software prior to entering Stop Mode. Energy Mode 3 can also be entered by calling the emlib function

```
EMU_EnterEM3(bool restore)
```

The argument sets whether or not clocks and oscillators are to be restored at wakeup.

2.5 Shut Off Mode (Energy Mode 4)

In Shut Off Mode the controller is completely shut down and the current consumption is as low as 20nA. The only way to exit this energy mode is to assert the reset line, cycle the power or by an EM4 wakeup. In this context the word "assert" means setting a pin high or low to make something happen. In the case of asserting the reset line this means setting the pin low. In addition to the these three, the Giant Gecko can exit EM4 by an BURTC interrupt.

Shut Off Mode is entered by writing 0x3 and 0x2 four times to the EM4CTRL field in the EMU_CTRL register of the Energy Management Unit (EMU). Alternatively, Energy Mode 4 can be entered by calling the emlib function

```
EMU_EnterEM4()
```

2.6 Energy mode transitions

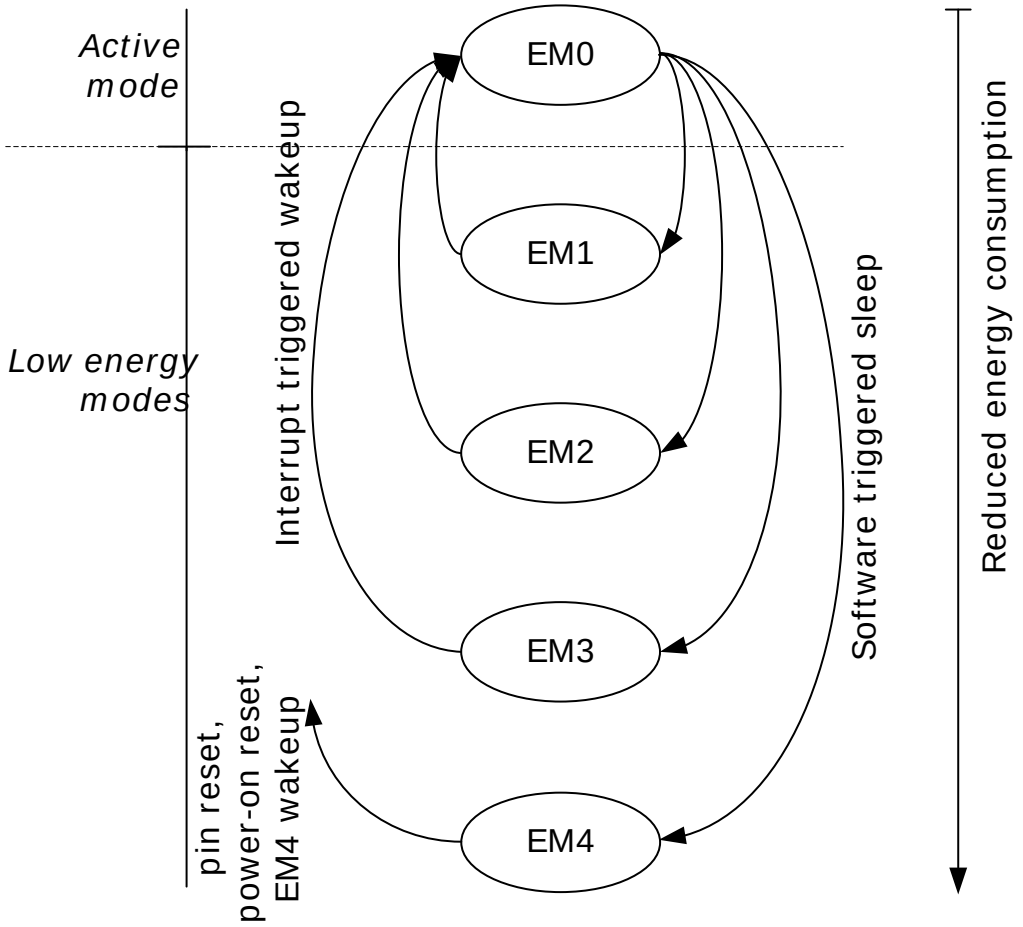
Figure 2.1 (p. 5) shows the transitions between different energy modes. After reset the EMU will always start in EM0. A transition from EM0 to another energy mode is always initiated by software. EM0 is the highest activity mode, in which all functionality is available. EM0 is therefore also the mode with highest energy consumption.

The low energy modes EM1 through EM4 result in less functionality being available, and therefore also reduced energy consumption. The Cortex-M3 is not executing instructions in any low energy mode. Each low energy mode provides different energy consumptions associated with it, for example because a different set of peripherals are enabled or because these peripherals are configured differently.

A transition from EM0 to a low energy mode can only be triggered by software.

A transition from EM1 – EM3 to EM0 can be triggered by an enabled interrupt or event. In addition, a chip reset will return the device to EM0. A transition from EM4 can only be triggered by a pin reset, power-on reset, or EM4 wakeup. Pin reset is done by asserting the reset line, power-on reset is when power has been cut off and then reconnected, while EM4 wakeup is a reset that can be set to happen for certain pins in EM4. It is possible to choose whether the pins should be set low or high for the wakeup. The `GPIO_EM4WUEN` register is used to enable EM4 wakeup for available pins, e.g A0, A6, C9, F1, F2, E13. The associated polarity, i.e. whether the reset will be caused by setting the corresponding pin high or low, is set in the `GPIO_EM4WUPOL` register.

Figure 2.1. EMU Energy Mode Transitions



No direct transitions between EM1, EM2 or EM3 are available, as can also be seen from Figure 2.1 (p. 5) . Instead, a wakeup will transition back to EM0, in which software can enter any other low energy mode.

2.7 Energy Management Unit (EMU)

The EMU (Energy Management Unit) handles the different low energy modes in the EFM32 microcontrollers. The EMU features Energy Mode control from software, flexible wakeup from low energy modes and low wakeup time. The EMU is available as a peripheral on the peripheral bus. The energy management state machine is triggered from the Cortex-M3 and controls the internal voltage regulators, oscillators, memories and interrupt systems in the low energy modes. Events from the interrupt or reset systems can in turn cause the energy management state machine to return to its active state.

3 Clock and Oscillator Control

The current consumption is highly dependent on clock frequency. Selecting correct oscillator and frequency is therefore a very important aspect in low energy applications. The following sections will discuss different modes and settings for clocks and oscillators.

3.1 Clock Management Unit (CMU)

The Clock Management Unit (CMU) is responsible for controlling the oscillators and clocks on-board the EFM32. The CMU provides the capability to turn on and off the clock on an individual basis to all peripheral modules in addition to enable/disable and configure 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 are inactive.

There are four clock sources available:

- 1-28 MHz High Frequency RC Oscillator (HFRCO)
- 4-48 MHz High Frequency Crystal Oscillator (HFXO)
- 32.768 kHz Low Frequency RC Oscillator (LFRCO)
- 1 kHz Ultra Low Frequency RC Oscillator (ULFRCO)
- 32.768 kHz Low Frequency Crystal Oscillator (LFXO)

The oscillators is what drive the clocks. The crystal oscillators are external, and the RC oscillators are internal. The clocks can be scaled in the prescaler to obtain the preferred frequency. The system clocks are:

3.1.1 HFCLK - High Frequency Clock

HFCLK is the selected High Frequency Clock. This clock is used by the CMU and drives the two prescalers that generate HFCORECLK and HFPERCLK. The HFCLK can be driven by a high-frequency oscillator (HFRCO or HFXO) or one of the low-frequency oscillators (LFRCO or LFXO). By default the HFRCO is selected.

3.1.2 HFCORECLK - High Frequency Core Clock

HFCORECLK is a prescaled version of HFCLK. This clock drives the Core Modules, which consists of the CPU and modules that are tightly coupled to the CPU, e.g. MSC, DMA etc.

3.1.3 HFPERCLK - High Frequency Peripheral Clock

Like HFCORECLK, HFPERCLK is also a potentially prescaled version of HFCLK. This clock drives the High-Frequency Peripherals.

3.1.4 LFACLK - Low Frequency A Clock

LFACLK is the selected clock for the Low Energy A Peripherals. The low energy peripherals are divided into A and B peripherals. The partition is shown in Figure 3.1 (p. 8) There are four selectable sources for LFACLK: LFRCO, LFXO, HFCORECLK/2 and ULFRCO.

3.1.5 LFBCLK - Low Frequency B Clock

LFBCLK is the selected clock for the Low Energy B Peripherals. There are four selectable sources for LFBCLK: LFRCO, LFXO, HFCORECLK/2 and ULFRCO.

3.1.6 PCNTnCLK - Pulse Counter n Clock

Each available pulse counter is driven by its own clock, PCNTnCLK where n is the pulse counter instance number. Each pulse counter can be configured to use an external pin (PCNTn_S0) or LFACLK as PCNTnCLK.

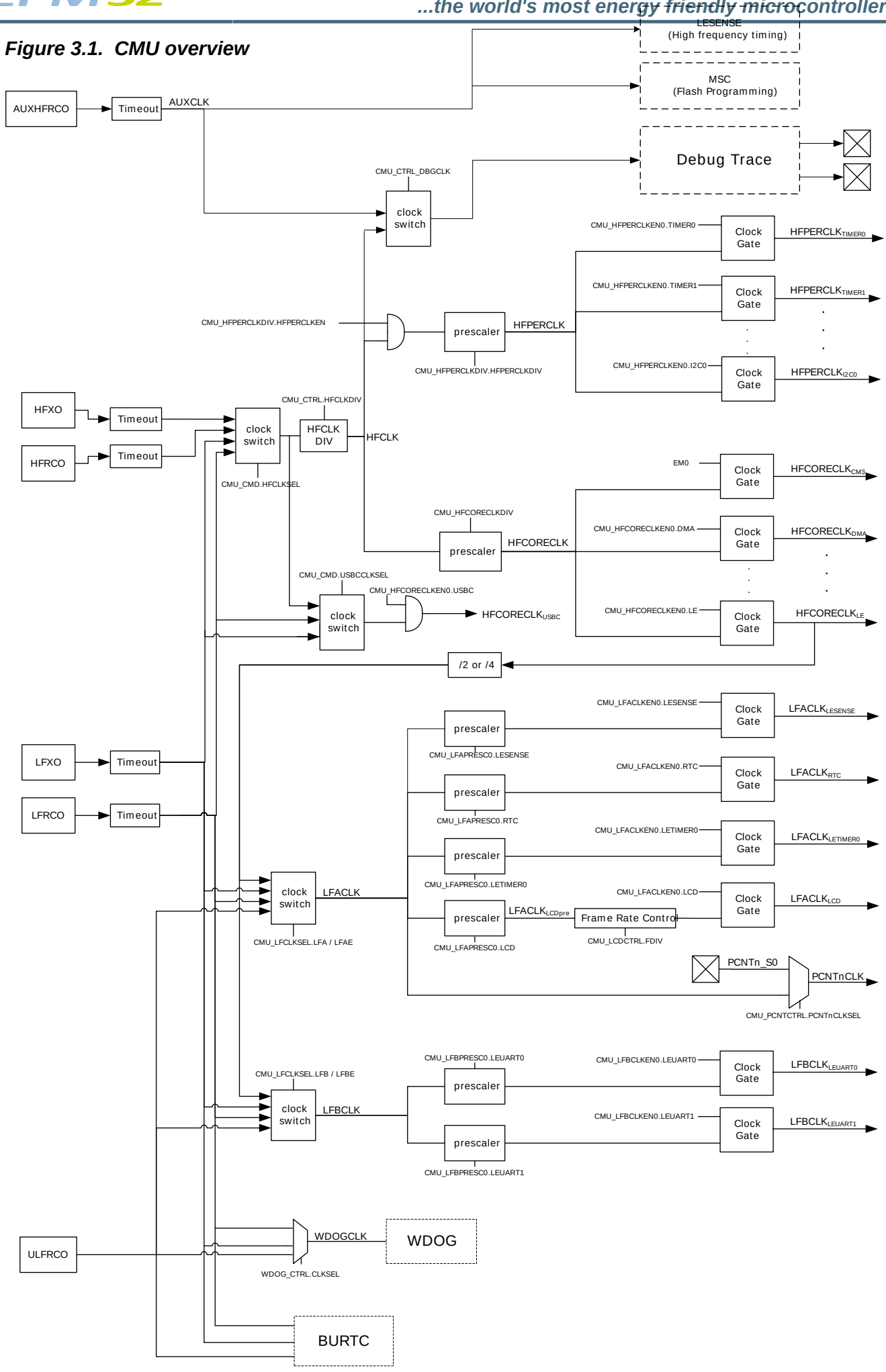
3.1.7 WDOGCLK - Watchdog Timer Clock

The Watchdog Timer (WDOG) can be configured to use one of three different clock sources: LFRCO, LFXO or ULFRCO. ULFRCO (Ultra Low Frequency RC Oscillator) is a separate 1 kHz RC oscillator that also runs in EM3.

3.1.8 AUXCLK - Auxiliary Clock

AUXCLK is a 1-28 MHz clock driven by a separate RC oscillator, AUXHFRCO. This clock is used for flash programming, debug trace, and LESENSE operation. During flash programming, or if needed by LESENSE, this clock will be active. If the AUXHFRCO has not been enabled explicitly by software, the MSC or LESENSE module will automatically start and stop it.

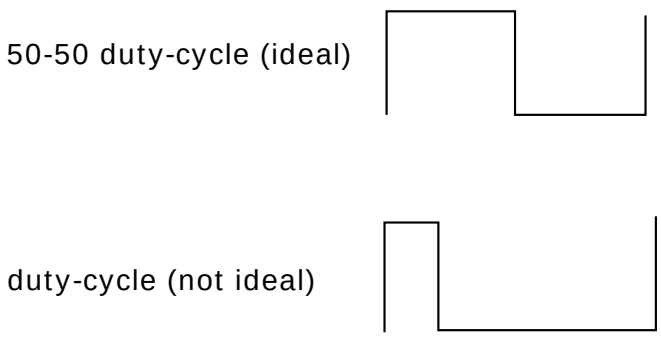
Figure 3.1. CMU overview



3.2 Enabling Oscillators/Setting Clock Source

Oscillators are enabled and disabled through the CMU_OSCENCMD register in the CMU. Each oscillator has one enable and one disable field in this register (e.g. LFXOEN and LFXODIS). The CMU_STATUS register holds two flags for each oscillator; enabled and ready (e.g. LFXOENS and LFXORDY). Each enabled-flag is set when the corresponding oscillator is turned on, whereas the ready-flag is set when the oscillator is ready to be used as a clock source. Until the ready-flag is set, the oscillator output may be incorrect, both with respect to frequency and duty-cycle. A duty-cycle is shown in the following figure, and should ideally be a 50-50 duty-cycle.

Figure 3.2. Duty cycles



If an oscillator is used before this flag is set, behavior may become unpredictable/undefined.

Out of reset, the High Frequency RC Oscillator (HFRCO) is set as source for the Core and high-speed peripherals (HFCLK domain). For the Low Energy (LE) clock domains (A and B), no oscillator is enabled or selected.

To change the source of a clock, do the following:

1. Enable the desired oscillator by setting the corresponding bit in the CMU_OSCENCMD register in the Clock Management Unit (CMU). Alternatively, you can use the emlib function

   ```
   CMU_OscillatorEnable(CMU_Osc_TypeDef osc, bool enable, bool wait)
   ```

   The first argument is the oscillator to be configured, the second is whether to turn the oscillator on or off and the third argument is whether the function should wait for the oscillator to become ready before returning.
2. Wait until the ready-flag of the oscillator in the CMU_STATUS register is set. This is not necessary when `wait` is set to `true` when using `CMU_OscillatorEnable()`
3. Change clock source to the new oscillator. For the HFCLK, this is done in the CMU_CMD register. For the LE domains, use the CMU_LFCLKSEL register. Alternative, you can use the emlib function

   ```
   void CMU_ClockSelectSet(CMU_Clock_TypeDef clock, CMU_Select_TypeDef ref)
   ```

   The first argument is which clock to configure and the second argument is which oscillator to use.

3.3 HFRCO Band Setting

The extreme frequency tuning range of the HFRCO is a major advantage, and should be used to minimize the energy consumption of any application. The following frequencies may be set [MHz]: 1 - 7 - 11 - 14 - 21 - 28. Frequency band is selected using the BAND field in the CMU_HFRCOCTRL register.

3.4 Enabling or Disabling a Clock to a Module / Peripheral

A module's clock may be enabled or disabled using the corresponding bit in the appropriate CLKEN register in the CMU. The following registers are used for this operation:

- CMU_HFCORECLKEN0: High-speed and core modules (DMA, EBI, AES and interface to LE peripherals).
- CMU_HFPERCLKEN0: Used for the regular peripherals (e.g. TIMERS, ADC, USART, ...).
- CMU_LFACLKEN0 / CMU_LFBCLKEN0: Used for the LE peripherals (e.g. RTC, LEUART, LETIMER, ...).

Please see the EFM32 Reference Manual for details on which register and field to use for a specific module or peripheral.

3.5 Clock Prescaling

The clocks of the EFM32 microcontrollers may be prescaled (divided by a given factor) in the CMU. The core clock prescaling is set in the CMU_HFCORECLKDIV register, and is common to the CPU as well as the other core modules (e.g. AES, DMA, EBI). The prescaling factor for the regular peripherals is also common and set in the CMU_HFPERCLKDIV register. However, for the LE peripherals, prescaling is set individually. E.g. RTC prescaling may be set 128, whereas the LEUART1 clock is not prescaled (division factor is 1). LE peripheral prescaling is set in CMU_LFAPRESC0 and CMU_LFBPRESC0.

4 Energy Debugging

It is possible to visualize the energy consumption of a system and relate it to the software running on the microcontroller by using Advanced Energy Monitoring (AEM). This is a part of the Energy Micro kits. Real time information about current consumption can be displayed by the energyAware Profiler by connecting the EFM32 kit to a computer through a USB port. The AEM together with the profiler will achieve advanced energy debugging.

4.1 Advanced Energy Monitoring (AEM)

The AEM is built into the STK, enabling the developer to track a system's energy consumption while the application is running and therefore with real (not estimated) current values. As these currents are measured, other components can also be included, to give a complete view of the systems energy consumption.

The dynamic range of the on-board AEM goes from a minimum of 100 nA to a maximum of 50 mA. This covers the needs of most energy sensitive applications and it is important to notice that it measures not only the current drawn by the MCU but also the current drawn by other system components, as long as they are supplied from the power rail monitored by the AEM.

The EFM32 can be set to output program counter samples through the Serial Wire Viewer Output (SWO) pin which is sent together with the AEM data (current, voltage and time). Enabling this feature increases power consumption slightly which should be accounted for when debugging the device.

4.2 energyAware Profiler

The energyAware Profiler is an advanced energy debugging tool that complements the STK. This software tool gets data from the AEM on the kits via USB and displays information in a current vs. time graphical representation.

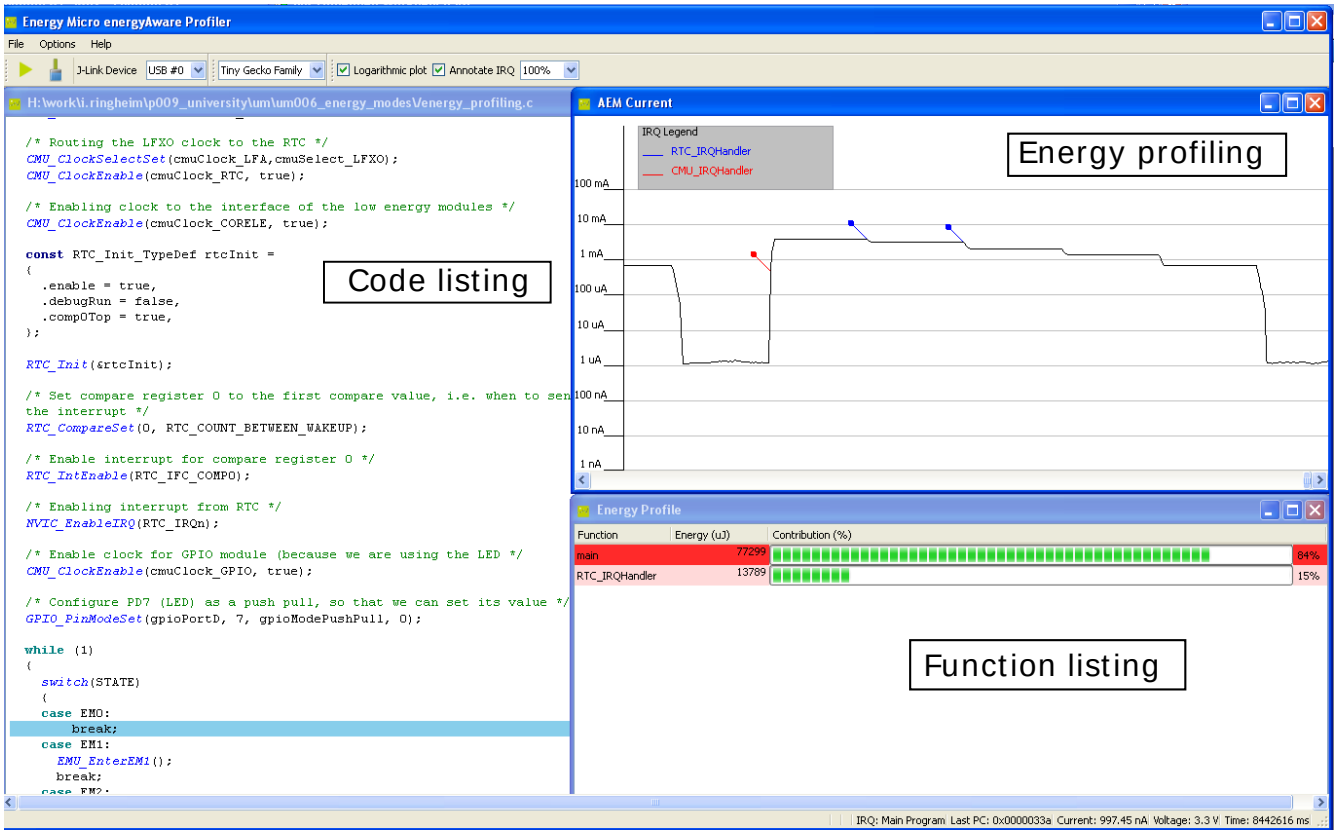
The final result of code compilation is an object file (*.out) that follows the ELF (Executable and Linkable Format) file standard and contains DWARF (Debug With Arbitrary Record Format) debug information. This file must be loaded into the Profiler so that the code trace feature works properly.

4.3 An energyAware Profiler example; `1_energy_profiling`

We look at the first code example of the attached example source code, `1_energy_profiling`. The example shows LED toggling in different energy modes. The chip starts in EM2 with the LED turned off. Then an interrupt wakes the chip up, and it enters EM0 and the LED is turned on. Then a new interrupt is sent, and the LED is turned off. The next interrupt sends the chip to EM1 and turns on the LED. This continues in the different energy modes. The code has to be compiled to create the object (*.out) file.

The energyAware Profiler can be found in Simplicity Studio. When opening this there is a short guidance of how to start. First you have to upload the object file. This can be found in the project folder under Debug->Exe (*.out). To be able to see the function calls, the function written in the guidance, `setupSWO()`, must be copied into the code you are running, and must be called early in the `main()` program. This is already done in this example. You then press "play" to visualize the energy consumption in the "AEM Current" window. The following figure shows an example of running a program:

Figure 4.1. energyAware Profiler



The following figure shows the graphical visualization of the energy consumption:

Figure 4.2. Energy consumption, current vs. time, for the energy profiling example

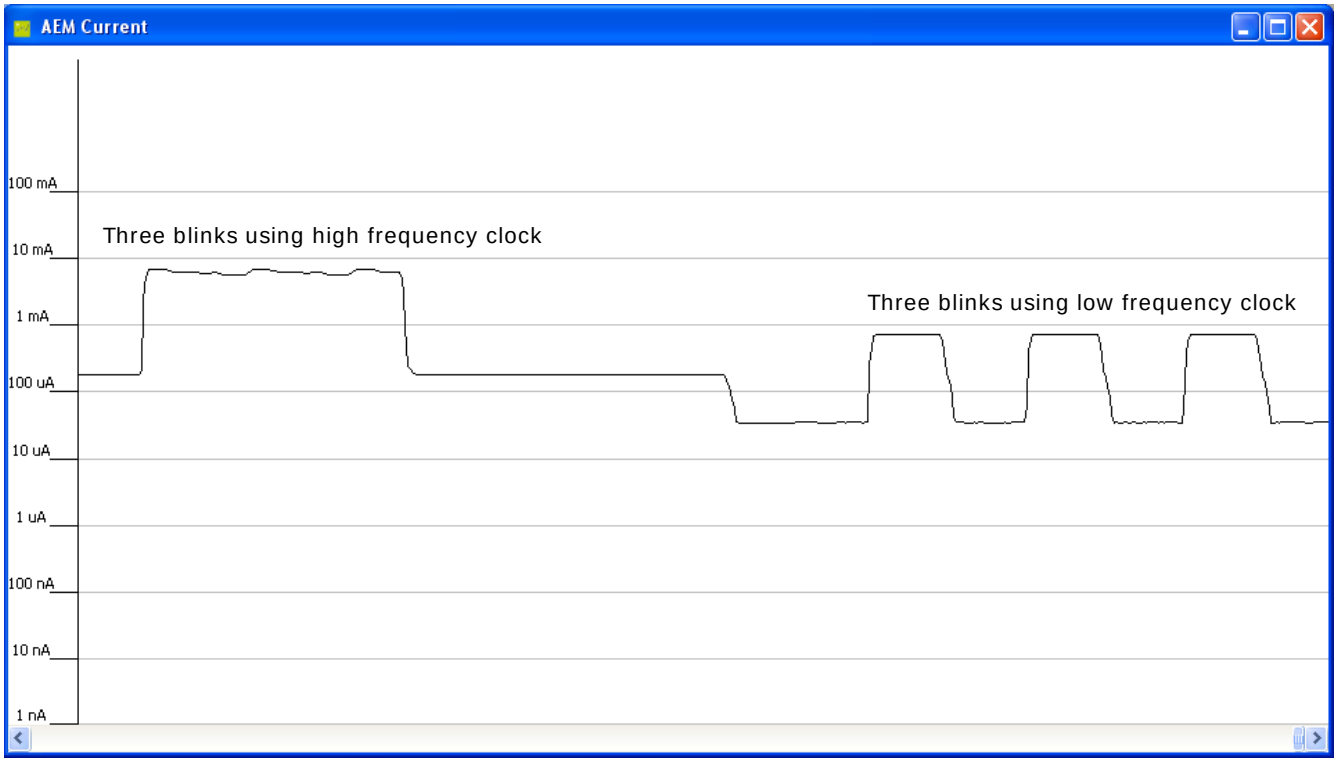
4.4 Energy Modes Example

The `2_energy_modes` example toggles the different Energy Modes each time Push Button 0 is pressed.

4.5 Core Clock Frequency Example

We look at another example from the attached source code; `3_core_clock`. Here the LED is blinking three times first using a high frequency clock. Then we switch to a low frequency clock and let the LED blink three times. A delay is included in the function that makes the LED blink such that the time between each blink is roughly the same using the two different clocks. The difference in energy consumption is shown in the following figure:

Figure 4.3. Energy consumption, current vs. time, for the core clock frequency example



5 Summary

This lesson has introduced the concepts of energy friendly MCU programming. By entering low energy modes, the energy consumption is dramatically reduced. The peripherals can work independently of the CPU and wakes it up when needed using an interrupt. The principles used to save energy is to turn off unused peripherals, disable clocks to unused peripherals and to reduce clock frequencies.

6 Exercises

Frameworks and solutions to the exercises below are provided together with the other example code.

6.1 Exercise 1: Choosing the right Energy Mode

In this exercise your task is to enter as low energy modes as possible while still being able to run through to the eternal while loop at the end. The places to enter your code is marked in the `main`. When the GPIO interrupt is enabled Push Button 0 has to be manually pressed to generate the interrupt. The USART will transmit a data and generate an interrupt even if the proper pin is not connected to anything.

7 Revision History

7.1 Revision 1.00

2011-06-22

Initial revision.

7.2 Revision 1.10

2012-07-27

Updated for Giant Gecko STK.

A Disclaimer and Trademarks

A.1 Disclaimer

Energy Micro AS intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Energy Micro products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Energy Micro reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Energy Micro shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Energy Micro. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Energy Micro products are generally not intended for military applications. Energy Micro products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

A.2 Trademark Information

Energy Micro, EFM32, EFR, logo and combinations thereof, and others are the registered trademarks or trademarks of Energy Micro AS. ARM, CORTEX, THUMB are the registered trademarks of ARM Limited. Other terms and product names may be trademarks of others.

B Contact Information

B.1 Energy Micro Corporate Headquarters

Postal Address	Visitor Address	Technical Support
Energy Micro AS P.O. Box 4633 Nydalen N-0405 Oslo NORWAY	Energy Micro AS Sandakerveien 118 N-0484 Oslo NORWAY	support.energymicro.com Phone: +47 40 10 03 01

www.energymicro.com
Phone: +47 23 00 98 00
Fax: + 47 23 00 98 01

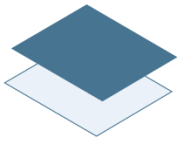
B.2 Global Contacts

Visit **www.energymicro.com** for information on global distributors and representatives or contact **sales@energymicro.com** for additional information.

Americas	Europe, Middle East and Africa	Asia and Pacific
www.energymicro.com/americas	www.energymicro.com/emea	www.energymicro.com/asia

Table of Contents

List of Figures



ENERGY®
micro

Energy Micro AS
Sandakerveien 118
P.O. Box 4633 Nydalen
N-0405 Oslo
Norway

www.energymicro.com