... the world's most energy friendly microcontrollers

Energy Modes

AN0007 - Application Note

Introduction

This application note describes strategies to reduce current consumption as well as how to enter different energy modes.

Additionally the prime number calculation code used in current consumption benchmarking is included.

This application note includes:

- This PDF document
- Source files (zip)
  - Example C-code
  - Multiple IDE projects

1 Energy Saving

1.1 General

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

- Use appropriate Energy Modes
- Exploit low energy peripherals
- Turn off unused modules / peripherals
- Disable clocks to unused modules / peripherals
- Reduce clock frequency
- Lower the operating voltage

The EFM32 supports extensive usage of all these principles. This will be explained in this and the following chapters.

1.2 Use Appropriate Energy Modes

The most effective way to save energy is to spend as little time as possible in active mode. The EFM32 features 5 tailored Energy Modes that allows the application to always be in an energy optimal state.

1.3 Make Use of Low Energy Peripherals

All the peripherals on EFM32 are built with energy consumption in mind. The peripherals are available in various energy modes. Whenever possible, select an appropriate peripheral and let it do the work, while the CPU goes to sleep (or performs other tasks). A few examples:

- Use RTC and go to sleep instead of busy-waiting
- Use DMA to transfer data between memory and U(S)ART
- Use the Low Energy Sensor Interface (LESENSE) to monitor a sensor instead of waking up and polling

See the System Overview chapter in the Reference Manual for a device to see which peripherals are available in different energy modes.

1.4 Turn off Unused Modules / Peripherals

At a given time in every microcontroller application, there are modules / peripherals which are not used. Turn these off to save energy.

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. E.g. when using the ADC, it requires some time to warm up before starting a conversion.

1.5 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 for all unused modules. The EFM32 fully supports this, please see the attached example and the Clock Management Unit (CMU) chapter in the EFM32 reference manual for details.

1.6 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 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, please see the Energy Modes chapter for a further description.

1.7 Lower the Operating Voltage

By lowering the operating voltage, energy consumption is further reduced. The EFM32 family of microcontrollers can run with full functionality on low voltages. The absolute minimum ratings be found in the data sheet for each device.

2 Energy Modes

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

By using the *emlib* functions, it is extremely simple to switch between different energy modes.

2.1 Run Mode (Energy Mode 0)

This is the default mode. 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.

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 Low Energy Sensor Interface (LESENSE) is monitoring a sensor
- The LCD controller drives an LCD.
- The LEUART is receiving or transmitting a byte.
- The I2C is performing address match check.
- 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 IO 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.

emlib function: `EMU_EnterEM2()`

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:

- I2C Address Check
- Watchdog

- Asynchronous Pin Interrupt
- Analog Comparator (ACMP)
- Voltage Comparator (VCMP)

Stop Mode is entered the same way as Deep Sleep Mode, except that also the low frequency oscillators are turned off.

emlib function: `EMU_EnterEM3()`

2.5 Shut Off Mode (Energy Mode 4)

In Shut Off Mode the EFM32 is completely shut down and the current consumption is as low as 20nA. The only way to exit this energy mode is with a reset.

Shut Off Mode is entered by writing 0x3 and then 0x2 four times to the EM4CTRL field in the EMU_CTRL register of the Energy Management Unit (EMU).

emlib function: `EMU_EnterEM4()`

2.6 Waking Up

From EM1 to EM3 the EFM32 will wake up (enter EM0) on any interrupt. When waking up, the Interrupt Service Routine (ISR) for that interrupt will be called first, and then code execution will start from where the CPU went to sleep.

However, waking up from EM4 is only possible through a reset. To facilitate timed and event-based wakeups from EM4, the EFM32 provides a Backup Real-Time Counter (BURTC) which is able to run even in EM4. Selected GPIO pins can also trigger a reset. Waking up from EM4 requires some additional configuration which must be done in software, before going to sleep.

Note

The Backup Real-Time Counter is not available on all devices, please consult the reference manual for your device.

2.6.1 Timed Wakeup From EM4

To perform a timed wakeup from EM4 some registers has to be configured before going to sleep. In `EMU_EM4CONF`:

- An oscillator must be selected. If different from ULFRCO software must also make sure the oscillator is running and is stable before going to sleep.
- Wakeup from BURTC interrupt must be enabled.
- The voltage regulator must be enabled in EM4.

In the BURTC registers:

- BURTC must be enabled in EM4.
- BURTC interrupt must be enabled.
- A clock source and optional prescaling must be selected.

2.6.2 GPIO Wakeup From EM4

If the application needs to wake up from a GPIO pin (other than the reset pin) the following must be configured in the GPIO registers:

- The pin must be set to input.

- Active polarity of the reset signal must be set in `GPIO_EM4WUPOL`.
- Wake up from EM4 must be enabled in `GPIO_EM4WUEN`.

3 Clock and Oscillator Control

3.1 General

As previously mentioned, 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.

3.2 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.

Note

Until the ready-flag is set, the oscillator output may be incorrect, both with respect to frequency and duty-cycle. 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, please do the following:

1. Enable the desired oscillator by setting the corresponding bit in the CMU_OSCENCMD register in the CMU.
2. Wait until the ready-flag of the oscillator in the CMU_STATUS register is set.
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.

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 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 Energymodes Software Example

4.1 General

The attached example illustrates how to enter different energy modes, how to enable different oscillators, enable / disable clocks and set up prescaling. To illustrate the importance of only enabling needed clocks and oscillators, the software starts off by turning on "everything". Then, more and more energy is saved by disabling clocks and oscillators, and by entering energy modes.

Note

When in debug mode the MCU will not go below EM1. When running this example exit debug mode and reset the MCU after flashing it.

4.2 Program Flow

The program goes through the following states. After each change of settings, a few seconds of waiting is inserted to make the current consumption visible. Use the energyAware Profiler to see how the current consumption changes.

- **All Clocks Enabled** Every oscillator on the EFM32 is turned on. The HFCLK source is set to HFXO, which is the fastest oscillator. In addition, clocks to all core modules and regular peripherals are enabled.
- **All Clocks Disabled** The HFCLK source is set back to HFRCO. All unused oscillators are turned off, and clocks to unused modules / peripherals are disabled.
- **Core Clock Downscaled** The core clock frequency is reduced by selecting the 7 MHz frequency band. In addition, the core clock is prescaled with factor 4, i.e. the core frequency is 1.75 MHz.
- **Sleep Mode** The clock to TIMER0 is enabled, the clock to the regular peripherals is prescaled by 512 (maximum) and TIMER0 is set to issue an interrupt when it wraps. Then Sleep Mode is entered until the timer wakes up the device.
- **Deep Sleep Mode** The clock to the interface of the LE peripherals is enabled. Then the RTC is set up to issue an interrupt after a few seconds. Then Deep Sleep Mode is entered.
- **Stop Mode / Shut Off Mode** At the end of the program either EM3 or EM4 is entered and the program stays in this mode.

5 EM4 Wakeup Software Example

This example shows how to do a wake up from EM4. It will also be shown how to make use of the 512 bytes of available memory that can be retained in EM4.

5.1 DK example

This example is made for the EFM32GG-DK3750/EFM32GG-DK3650 Development Kit. The example will repeatedly enter EM4 and sleep for 5 seconds. A retention register is used to keep track of the number of times the MCU has woken up from a BURTC interrupt.

It is also possible to wake up by connecting pin PF2 to GND. A separate register is used to keep track of how many times the MCU has woken up from a GPIO pin.

Every time the MCU wakes up, the reset cause (reset pin, GPIO or BURTC) will be printed to the display. The number of times woken up (for either GPIO or BURTC) is also printed. Make sure the AEM mode is "EFM32" to enable the display (toggled with the AEM button).

5.2 STK example

This example is made for the EFM32GG-STK3700/EFM32GG-STK3600 Starter Kit. The example will repeatedly enter EM4 and sleep for 5 seconds. A retention register is used to keep track of the number of times the MCU has woken up from a BURTC interrupt.

It is also possible to wake up by connecting pin PF2 to GND. A separate register is used to keep track of how many times the MCU has woken up from a GPIO pin. The PF2 pin is labelled SWO on the starter kit and is found near the bottom right corner.

Every time the MCU wakes up, the reset cause (reset pin, GPIO or BURTC) will be printed to the LCD. The number of times woken up (for either GPIO or BURTC) is also printed.

6 Prime Number Software Example

6.1 General

The prime number software example is used for current consumption benchmarking. It sets up the EFM32 to forever execute a prime calculation algorithm from flash.

6.2 Setup

The code makes the EFM32 run from the High Frequency Crystal Oscillator (HFXO) and disables all other oscillators. Then the Suppressed Conditional Branch Target Prefetch (SCBTP) option in the MSC module is enabled before the prime calculation algorithm is started.

7 Revision History

7.1 Revision 1.09

2014-05-07

Updated example code to CMSIS 3.20.5

Changed to Silicon Labs license on code examples

Added example projects for Simplicity IDE

Removed example makefiles for Sourcery CodeBench Lite

7.2 Revision 1.08

2013-10-14

New cover layout

7.3 Revision 1.07

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

7.4 Revision 1.06

2012-11-12

Adapted software projects to new kit-driver and bsp structure.

7.5 Revision 1.05

2012-08-13

Adapted software projects to new driver file structure.

7.6 Revision 1.04

2012-07-19

Added EM4 wakeup example with BURTC and data retention

Fixed a bug where Prime Example would be stuck in while loop

Added software support for Tiny and Giant Gecko STK's.

7.7 Revision 1.03

2012-04-20

Adapted software projects to new peripheral library naming and CMSIS_V3.

7.8 Revision 1.02

2012-03-13

Fixed makefile-error for CodeSourcery projects.

7.9 Revision 1.01

2010-11-16

Changed example folder structure, removed build and src folders.

Added chip-init function.

7.10 Revision 1.00

2010-09-20

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

A.2 Trademark Information

B Contact Information

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701

Please visit the Silicon Labs Technical Support web page:
http://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.

Table of Contents

silabs.com

ZERO
ARM Cortex-M0+

TINY
ARM Cortex-M3

GECKO
ARM Cortex-M3

LEOPARD
ARM Cortex-M3

GIANT
ARM Cortex-M3

WONDER
ARM Cortex-M4