


# *Energy Micro University*

## UM007 - Energy Optimization

**This lesson presents general energy optimization techniques as well as EFM32 specific features that can greatly reduce the power consumption of your applications. We will also see that different types of applications require different techniques - when it is unfeasible to enter low Energy Modes the use of DMA and PRS helps reducing the active energy consumption.**

**Energy saving techniques that will be covered include:**

- **Energy Modes**
- **Interrupts**
- **Choosing oscillators**
- **Prescaling**
- **Choosing the right peripheral**
- **DMA and PRS**

**This lesson includes:**

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

ENERGY micro

# 1 Energy Optimization

The goal of energy optimization is to identify energy critical parts of a program and reduce their energy consumption to a minimum. Some concepts, including Energy Modes, were introduced in the last lesson. In this lesson we will revisit those general strategies as well as a few new ones. The two following chapters will present examples of how to energy optimize your programs. They will cover how to implement the strategies presented below as well as show how big of an impact they have on energy consumption. Chapter 2 (p. 4) will look at an application with infrequent tasks, i.e. most of the time is spent sleeping, while Chapter 3 (p. 9) will look at an application with frequent tasks, i.e. little or no time is spent sleeping.

## 1.1 Energy Modes

Energy Modes is a convenient way of turning off unused peripherals. By entering the lowest Energy Mode possible, the more energy demanding peripherals will be turned off and thereby saving energy. The Energy Modes range from the default EM0, where the CPU is on and all peripherals can be turned on, to EM4, where only a few features are available and hence the power consumption is very low. Please consult the reference manual of your device to see what features are available in each of the Energy Modes.

## 1.2 Interrupts

Interrupts combined with Energy Modes can offer great improvements over regular polling. Polling does not only keep the CPU from doing other work, it also consumes a lot of energy since the CPU is operating in EM0. By configuring the peripherals to produce interrupts when appropriate, the CPU is free to do other work or a low Energy Mode can be entered.

## 1.3 Oscillators

If the application is not timing critical, the use of the default RC oscillators instead of external crystal oscillators will both reduce the power consumption as well as the time to restore oscillators after waking up from low Energy Modes. The MCU will always wake up using the default RC oscillators, but if crystal oscillators are enabled and they are set to be restored when entering EM2 or EM3, extra time is needed to start and switch to the crystal oscillators at wakeup. Changing to a oscillator with lower frequency will also reduce the power consumption, but not necessarily the energy consumption if the time to complete the task increases excessively and that time could have been spent sleeping with the oscillator turned of.

## 1.4 Clock frequency

Instead of changing the oscillator, the clock of a peripheral can be prescaled so that the peripheral uses less power. Nevertheless, this might not be beneficial if the task takes much longer to complete.

## 1.5 Choosing the right peripheral

Some functionality is available in two or more peripherals with different energy consumption. The low energy modules are clocked by low frequency oscillators, therefore they cannot work as fast as the regular modules. The low energy modules might only have a subset of the features that are available in the regular module. The Low Energy UART can for instance only transfer at rates of up to 9600 symbols per second, but the regular UART supports much higher rates. Choosing the proper module is therefore a trade-off between functionality and energy consumption. In the following chapters we will see how choosing the RTC is beneficial for infrequent tasks, while the TIMER is needed for frequent tasks.

## 1.6 PRS and DMA

PRS and DMA allows events to be triggered and data to be transferred without CPU intervention. This allows the CPU to do other work, or a low Energy Mode can be entered. Often no lower Energy Mode

than EM1 can be entered, making it a poor choice if the application is suited for long sleep periods in EM2 or EM3. If the application would otherwise be in EM0 to do the same tasks, PRS and DMA would require less energy. PRS and DMA are also much more predictable than managing things in software.

# 2 Optimizing infrequent tasks

Some tasks only have to be done occasionally; here we will look at how to measure the internal temperature of the MCU about once every second. We will start with a naive implementation using polling. We will then improve the implementation while still retaining the same functionality by using

- interrupts
- choosing different peripherals
- choosing different oscillators
- prescaling

This will reduce the average energy consumption by a factor of several hundred. To illustrate how low it is possible to go while still being able to sample and calculate the temperature from time to time, the Ultra Low Frequency RC Oscillator will be used. It cannot drive the LCD, so it has to be turned off making it difficult to compare it with the other implementations, but the core functionality is still retained while using less than 1 µA.

## 2.1 Naive implementation using Timer

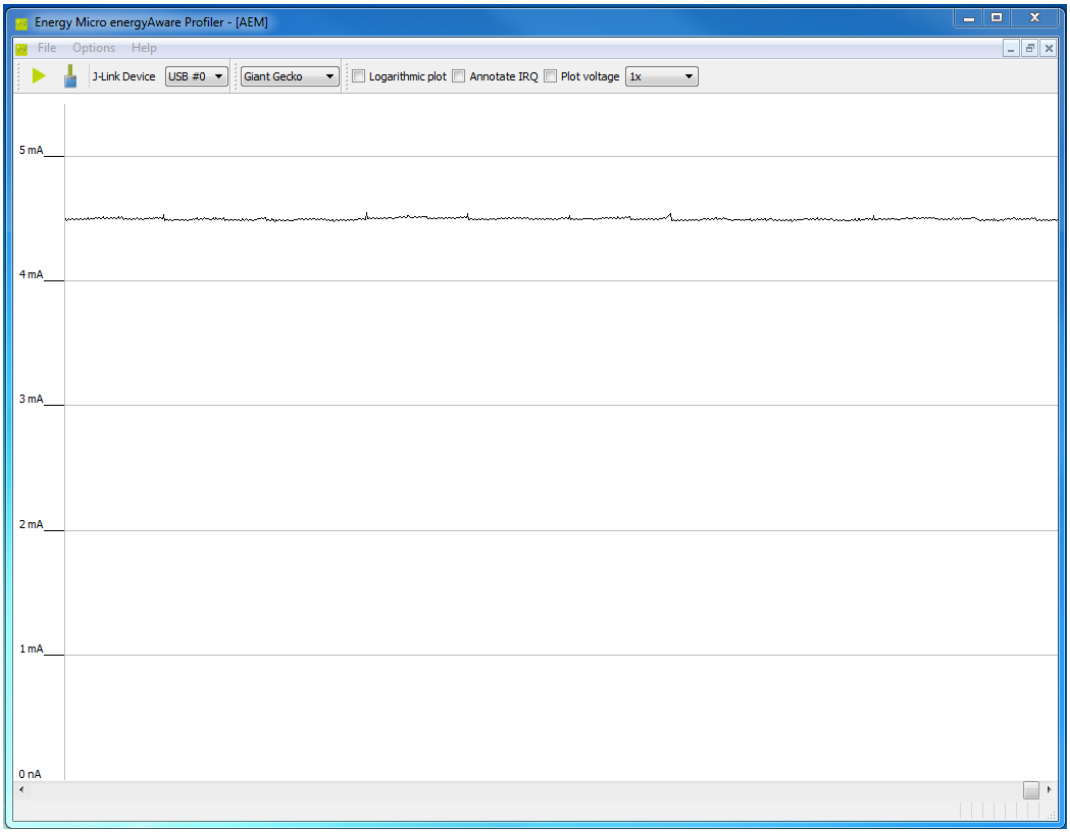
Example `1_timer` implements the desired functionality using the ADC and the TIMER. Polling is used both to check if the ADC sampling is complete and to check if the TIMER has overflowed.

Average consumption: ~4.5 mA

***Example 2.1. Pseudocode for `1_timer`***

```
while(1)
{
  adc_start();
  while(adc_is_sampling);
  show_results();
  while(timer_has_not_overflowed);
}
```

***Figure 2.1. Energy profile of example `1_timer`***




## 2.2 Improved version using TIMER and interrupts

Example `2_timer_interrupt` improves the energy consumption by using interrupts instead of polling. The TIMER produces an interrupt on overflow, which wakes up the CPU that trigger a new ADC sampling. Energy Mode 1 can then be entered while waiting for the ADC to finish and produce an interrupt. The result is computed and displayed before EM1 is entered again.

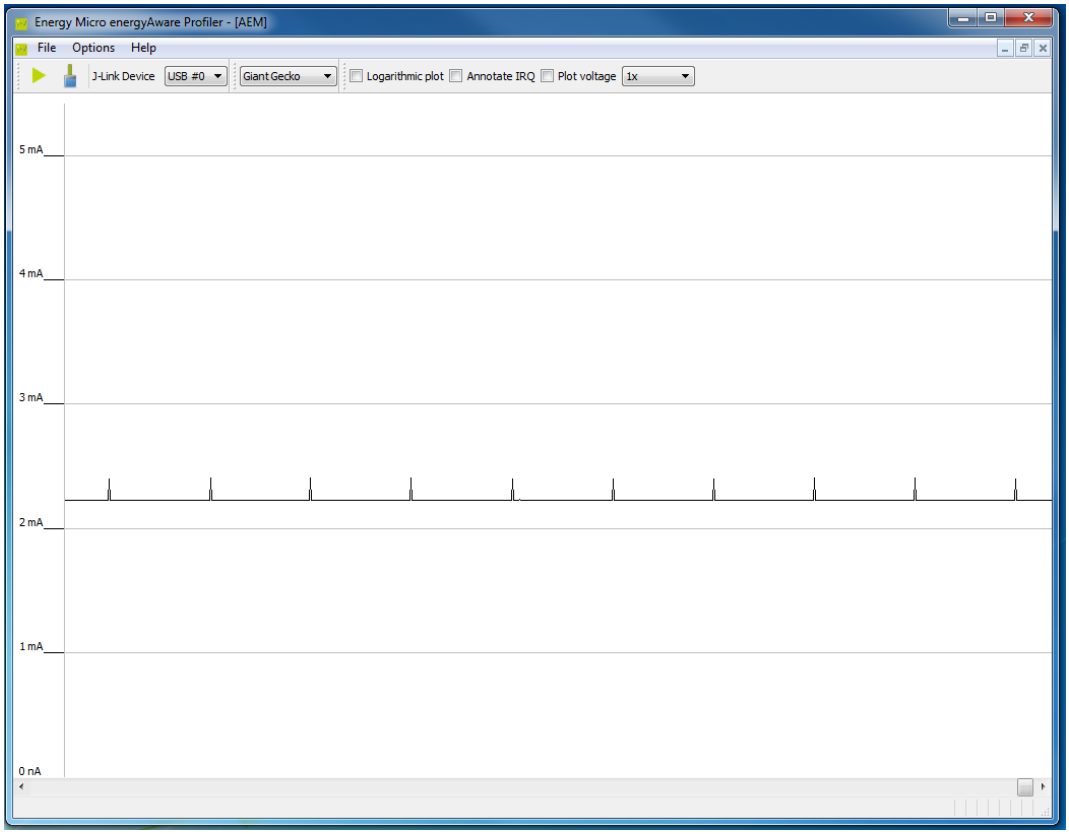
Average consumption: ~2.2 mA

Improvement: ~2.05

***Example 2.2. Pseudocode for `2_timer_interrupt`***

```
ADC_IRQ() { clear_interrupt(); }
TIMER_IRQ() { clear_interrupt(); }
while(1)
{
  adc_start();
  EnterEM1();
  show_results();
  EnterEM1();
}
```

***Figure 2.2. Energy profile of example `2_timer_interrupt`***




## 2.3 Further improvements using RTC and interrupts

By using the RTC with the Low Frequency RC Oscillator (LFRCO), `3_rtc_interrupt` can enter EM2 instead of EM1 that was used in the previous example. Prescaling of the RTC is also used to reduce energy consumption even further.

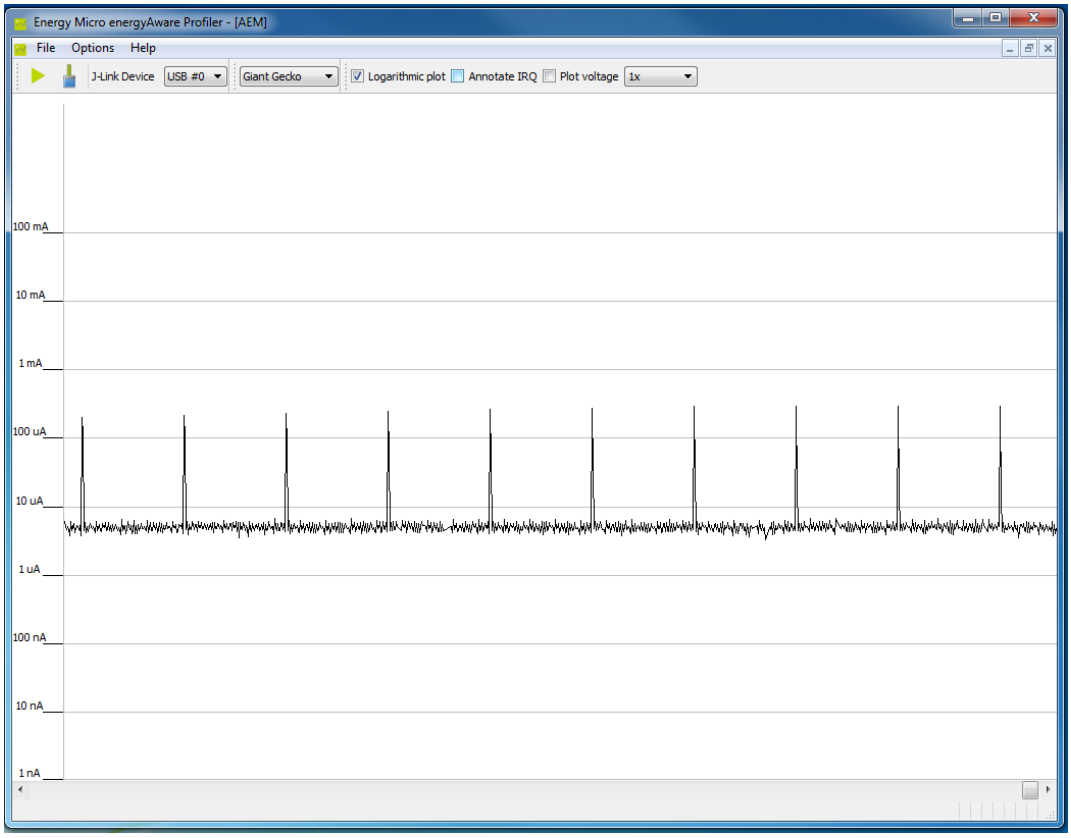
Average consumption: ~7.9 µA

Improvement: ~570

***Example 2.3. Pseudocode for `3_rtc_interrupt`***

```
ADC_IRQ() { clear_interrupt(); }
RTC_IRQ() { clear_interrupt(); }
rtc_setup() { prescale_rtc(); }
while(1)
{
  adc_start();
  EnterEM1();
  show_results();
  EnterEM2();
}
```

***Figure 2.3. Energy profile of example `3_rtc_interrupt`***




## 2.4 Alternative implementation using ULFRCO

By using ULFRCO instead of LFRCO example `4_rtc_ulfrco` achieves even lower energy consumption than the previous example. Unfortunately the ULFRCO cannot drive the LCD, so the power consumption cannot be compared directly with the previous examples. The ULFRCO is very inaccurate; it should only be used if timing is not important.

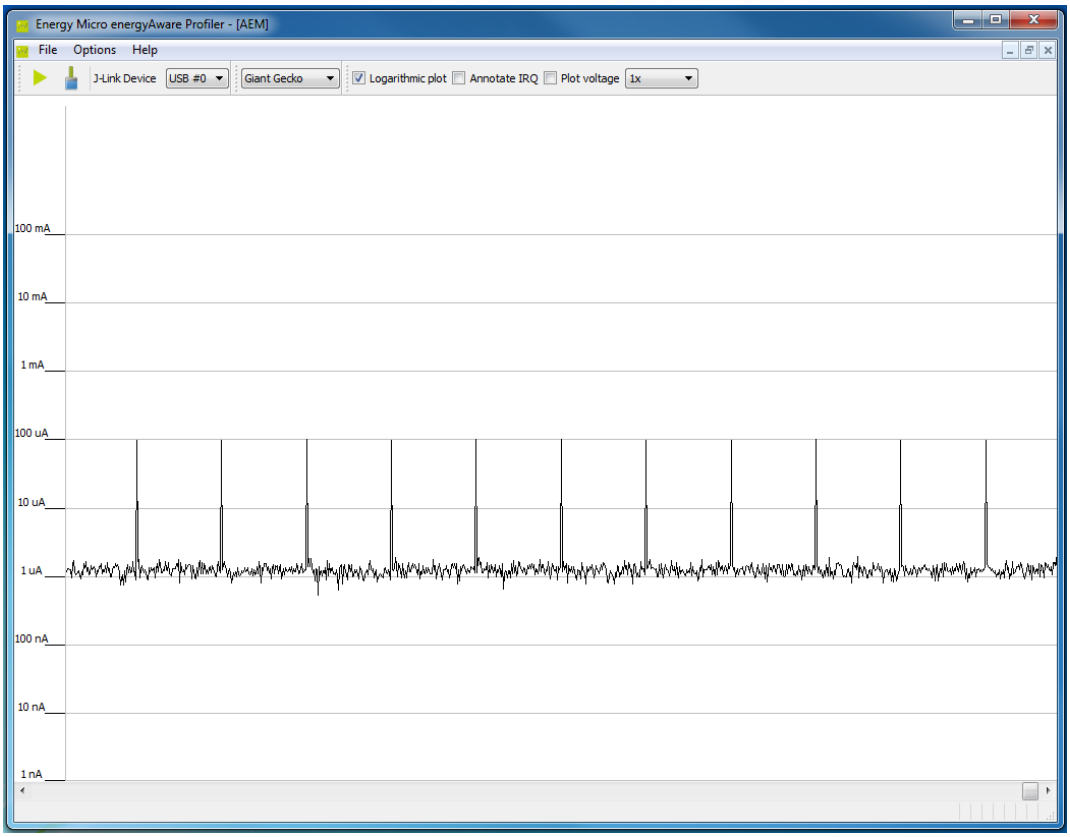
Average consumption: ~ µA

Improvement: ~1800 (Mainly due to turning off the LCD!)

***Example 2.4. Pseudocode for `4_rtc_ulfrco`***

```
ADC_IRQ() { clear_interrupt(); }
RTC_IRQ() { clear_interrupt(); }
rtc_setup() { use_ULFRCO_and_prescale_rtc(); }
while(1)
{
  adc_start();
  EnterEM1();
  compute_results();
  EnterEM3();
}
```

***Figure 2.4. Energy profile of example `4_rtc_ulfrco`***



# 3 Optimizing frequent tasks

Some tasks have to be done very frequently, as a result little or no time is available to sleep. The previous strategy of staying in the lowest possible Energy Mode is therefore unfeasible. We will look at how to sample the internal temperature thousands of times per second, place the samples in a buffer and then compute the average temperature. Occasionally the result will be displayed on the LCD. We cannot choose any of the low frequency oscillators, since the application would not be able to finish the tasks on time. We therefore need the TIMER using the HFRCO oscillator to have a chance of finishing on time.

## 3.1 100k samples per second using interrupts

Example `5_hf_timer` uses interrupts to sample 100k times per second and computing the average in a deferred software interrupt. Both the ADC and TIMER uses interrupts, this allows the CPU to do other work like computing the average of the buffers while waiting. The debug version is actually too slow to finish on time, so the optimized version is needed just to get a properly working program. The optimized version also shows that by finishing tasks earlier the CPU can sleep more and hence the energy consumption is reduced.

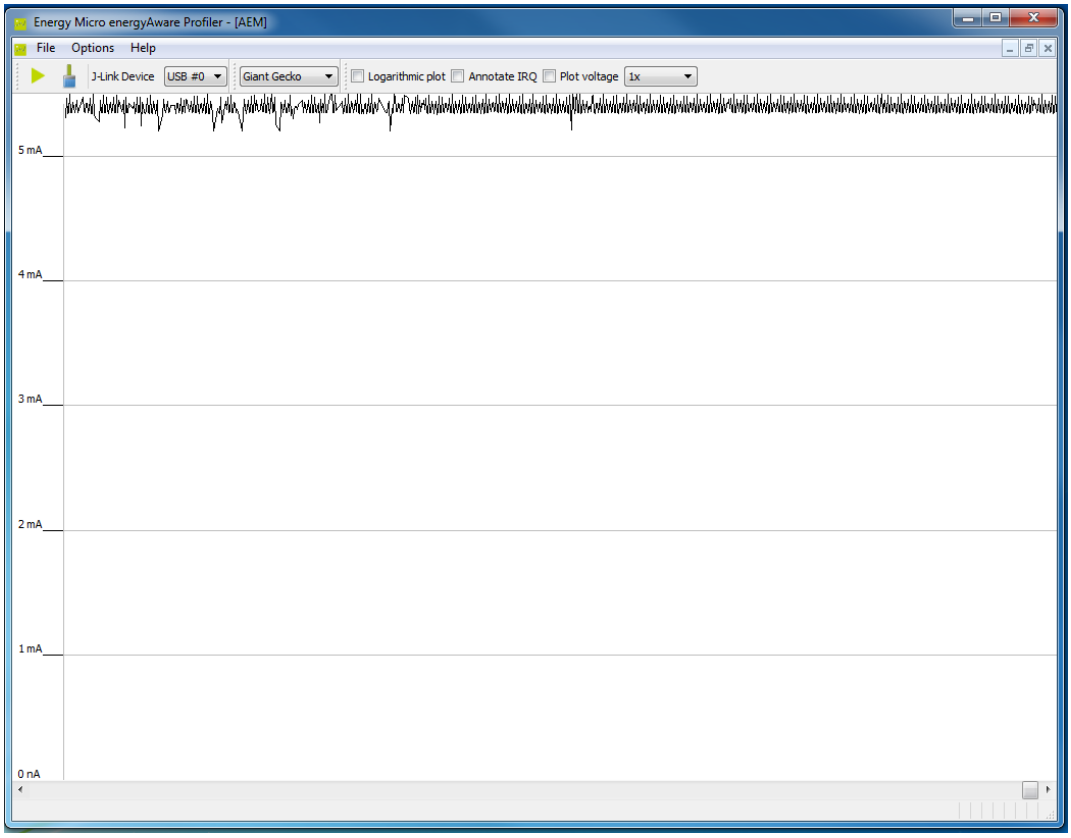
***Example 3.1. Pseudocode for `5_hf_timer`***

```
TIMER_IRQ() { adc_start(); }
ADC_IRQ()
{
  add_ADC_result_to_current_buffer();
  if ( buffer_is_full() ) set_software_interrupt();
}
Software_IRQ() { compute_and_show_results(); }
while(1)
{
  Enter_EM1();
}
```

### 3.1.1 Debug version

Average consumption: ~5.4 mA

***Figure 3.1. Energy profile of example `5_hf_timer` (debug)***



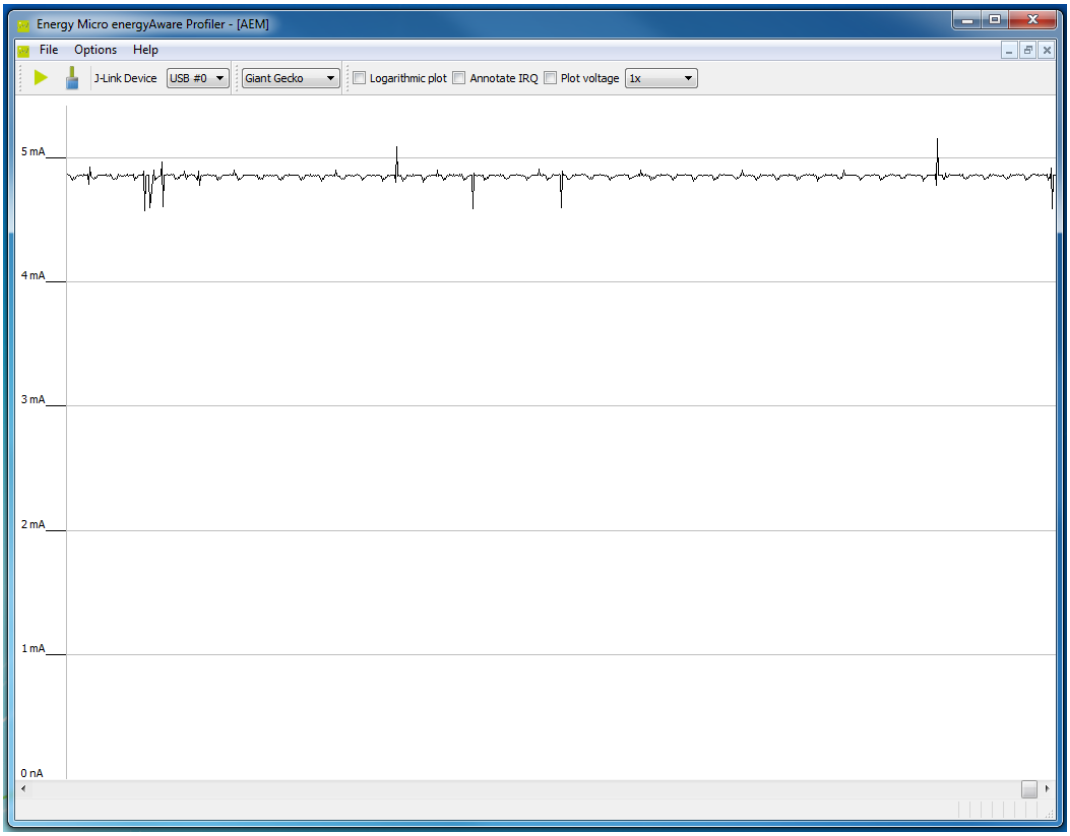

## 3.1.2 High code optimization

To get an optimized version you change the mode from debug mode to release mode in IAR.

Average consumption: ~4.9 mA

Improvement: ~1.1

***Figure 3.2. Energy profile of example `5_hf_timer` (optimized)***

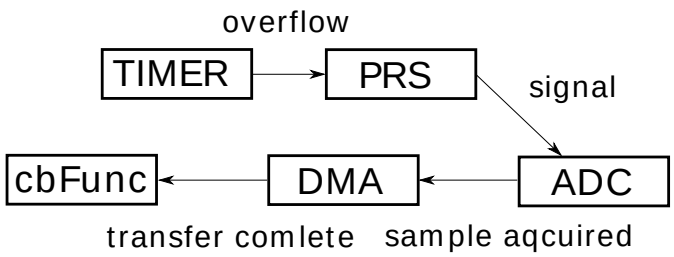



## 3.2 100k samples per second using DMA and PRS

Both the debug and optimized versions of example `6_hf_timer_dma_prs` use less energy than the one using interrupts. Since DMA and PRS can perform task without CPU intervention the CPU can sleep more. DMA and PRS are also more robust than relying on software to manage things. Since the CPU does less the gain from the debug to the optimized version is also smaller than for the interrupt example. Less work for the CPU also increases its chance of finishing on time - in this case both the debug and the optimized versions are able to do so. DMA/PRS is therefore not only more energy friendly in this case it is also more reliable than using interrupts.

An overview of the DMA/PRS setups is shown in Figure 3.3 (p. 11). The TIMER is set to overflow 100k times per second. The TIMER notifies the ADC via PRS to start sampling at each overflow. The ADC notifies the DMA when the sample has been acquired. The DMA then transfers the result from the ADC to RAM and triggers the callback function when done. The callback function takes care of post-processing.

***Figure 3.3. Overview of DMA/PRS setup***



***Example 3.2. Pseudocode for `6_hf_timer_dma_prs`***

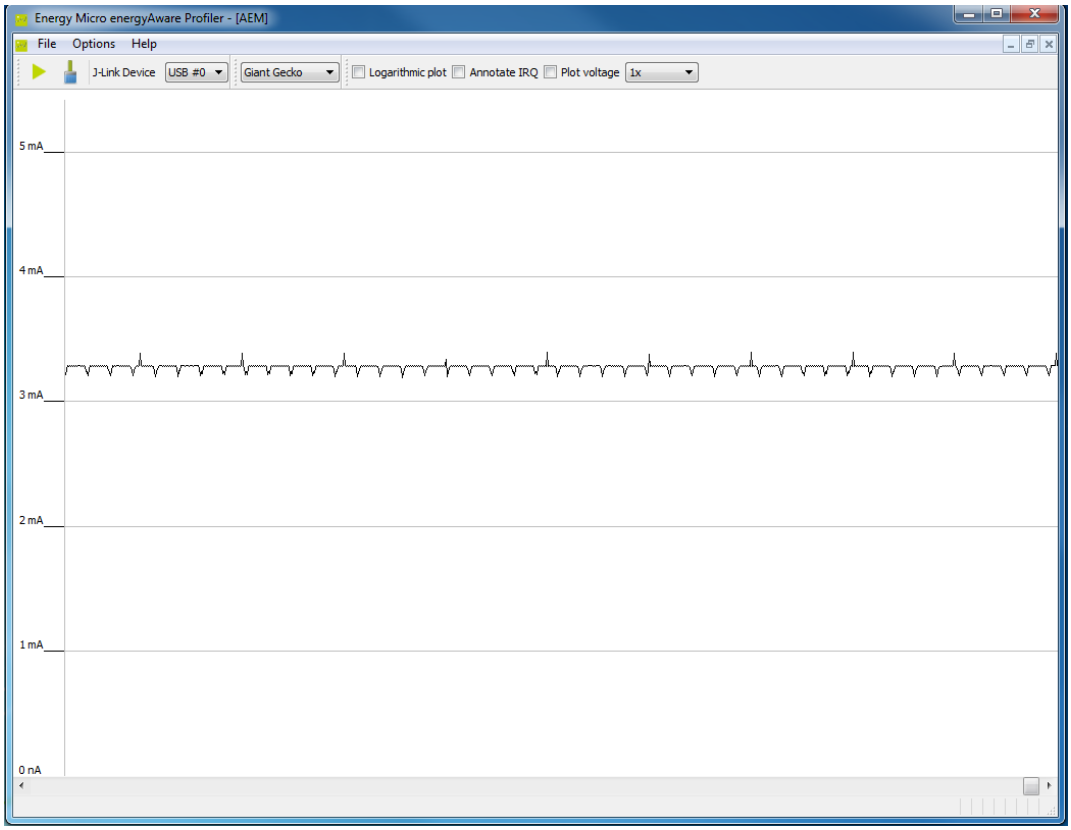
```
timer_setup() { enable_prs(); }
adc_setup()
{
  set_dma_to_transfer_aqcuired_data_to_ram();
}
dma_callback()
{
  swap_buffers;
  set_software interrupt()
}
Software_IRQ() { compute_and_show_results(); }
while(1)
{
  Enter_EM1();
}
```

## 3.2.1 Debug version

Average consumption: ~3.3 mA

Improvement: ~1.6

***Figure 3.4. Energy profile of example `6_hf_timer_dma_prs` (debug)***

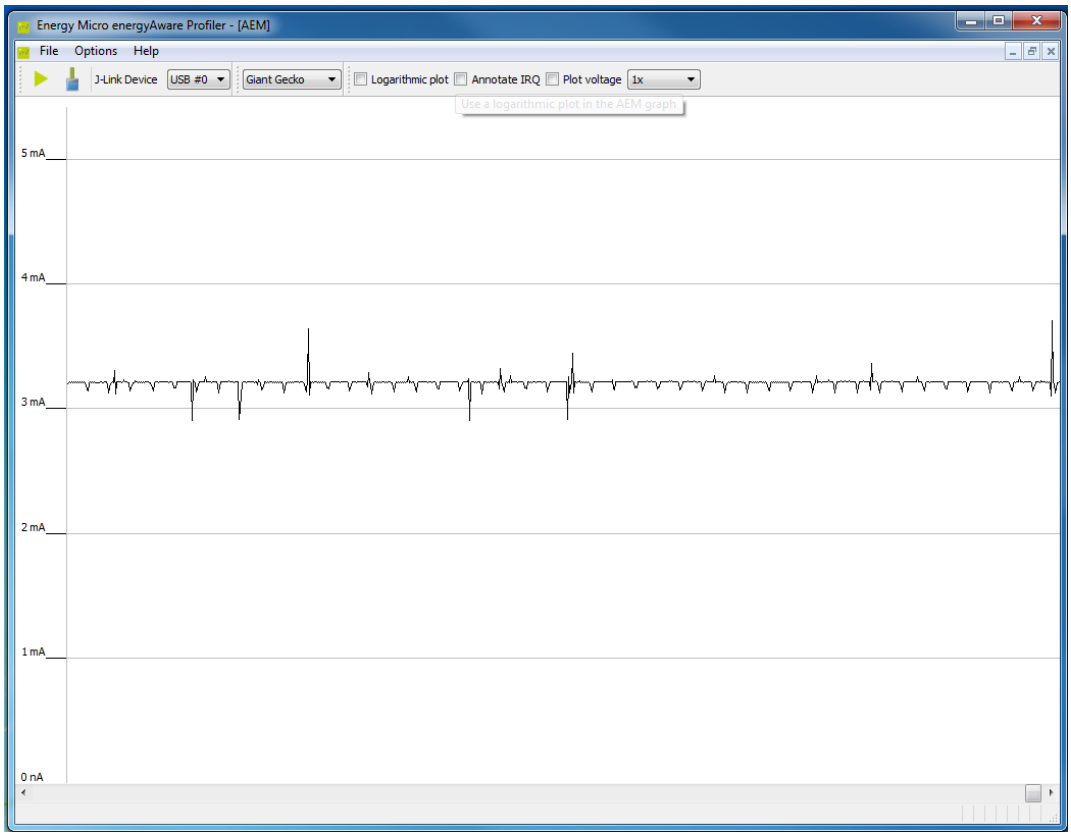



## 3.2.2 High code optimization

Average consumption: ~3.2 mA

Improvement: ~1.7

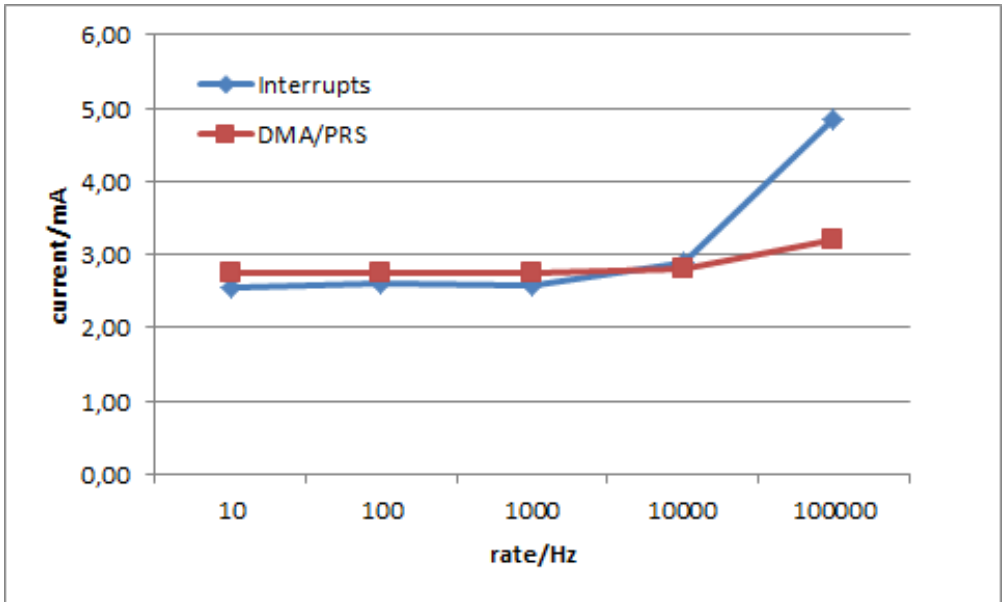
***Figure 3.5. Energy profile of example `6_hf_timer_dma_prs` (optimized)***




## 3.3 Choosing between interrupts and DMA/PRS

There is a trade-off between the extra cost of DMA/PRS and the amount of energy they can save. In the figure below the different consumptions for the optimized versions of `5_hf_timer` and `6_hf_timer_dma_prs` are plotted against the number of samples per second which is set with the define `ADC_SAMPLE_RATE`. The results show that the cost is about the same for 10 000 samples per second, but for higher rates DMA/PRS is the best choice.

***Figure 3.6. Energy consumption vs. samples per second***



# 4 Summary

In this lesson we have looked at how to greatly reduce the power consumption by use of interrupts, Energy Modes, changing oscillators and prescaling. When the application cannot spend significant time in low Energy Modes a different approach using DMA and PRS offers a robust and energy friendly alternative to regular interrupts.

# 5 Exercises

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

## 5.1 Exercise 1: Optimize a naive implementation of a stopwatch

Exercise `7_exercise_1_naive` implements a very simple stopwatch that waits for Push Button 0 to be pressed before starting the timekeeping. When Push Button 0 is pressed again, the timekeeping will be stopped and the result will be displayed on the LCD for some time before the LCD is turned off. The implementation has several shortcomings both in terms of functionality and energy consumption. One of which is that it is not possible to restart the stopwatch before after the delay showing the results. Furthermore polling is unreliable to check for button state and update time. Your task is to rewrite the program in `7_exercise_1_improved` using interrupts, Energy Modes and prescaling to increase its robustness and improve the energy consumption by a factor of at least 200.

# 6 Revision History

## 6.1 Revision 1.00

2011-06-22

Initial revision.

## 6.2 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<br>P.O. Box 4633 Nydalen<br>N-0405 Oslo<br>NORWAY | Energy Micro AS<br>Sandakerveien 118<br>N-0484 Oslo<br>NORWAY | support.energymicro.com<br>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

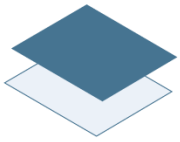
Visit **www.energymicro.com** for information on global distributors and representatives or contact **sales@energymicro.com** for additional information.

| 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

# List of Examples



ENERGY®
micro

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

www.energymicro.com