# Implementing Low Power Graphical User Interfaces with Microcontrollers

## Introduction to Low Power Displays

For any MCU application that requires user interaction, visual feedback is essential. For some applications a simple LED can be enough, while full-color high definition displays define the other end of the spectrum. Most embedded devices will be somewhere in between, and traditionally leaning towards the LED solution. Segment LCDs have long been a popular choice. They are cheap, easy to integrate and consume very little power. Graphical displays (active-matrix displays) have traditionally been off-limits, either because of price, CPU processing power or they simply consume too much power for an embedded platform.

But, in recent years there have been breakthroughs in low power displays, marking a change in what is possible with a strict power budget. This white paper will focus on two new display technologies: electronic paper displays (EPDs) and Memory LCDs and how to best control these with a microcontroller.

## Low Power Display Technologies

### Electronic Paper

Electronic paper is actually a family of display technologies, where the most common form of EPDs are electrophoric displays. The pixels in an electrophoric display are made up of tiny transparent capsules. The capsules are filled with a dark-colored oil. The oil contains titanium dioxide particles which are white and negatively charged. Electrodes are placed on both sides (front and back) of the capsules. If a positive voltage is applied to the front electrode, the particles are drawn towards the front electrode. The pixel appears white because light is reflected off the white particles. In the opposite case the particles are drawn to the back electrode and the pixel appears black because light is absorbed by the oil.

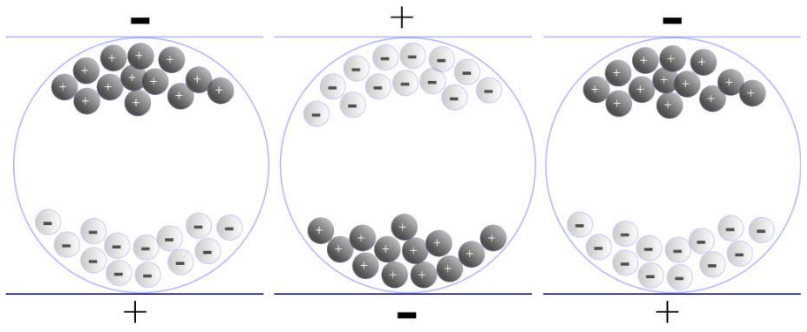


**Figure 2: Electrophoric display cells [Gerald Senarclens de Grancy / Wikimedia Commons]**

Common for these technologies is that the display is purely reflective. They require no backlight, but rely only on ambient light. They have excellent contrast in sunlight or common indoor lighting, but in dark environments an external light source is needed. The EPDs are *bistable* – once an image has been written to the screen it will keep this image, even when power is removed. The displays do not draw *any* power when showing a static image. This makes them particularly suitable for applications where the image content does not change frequently.

Even though EPDs consume no power when keeping a static image, the current consumption when updating the image can be significant. And the update time for an EPD is usually relatively long, around one second. EPDs are also prone to 'ghosting' – part of the old image can be seen after a refresh. This is often solved by updating the display twice, thus further adding to the power consumption and update time.

### Memory LCD

Memory LCDs are a new type of reflective dot-matrix display. In each pixel there is an embedded 1-bit memory circuit. Once a pixel has been written it will retain this value, thus removing the need for constantly refreshing the display like a conventional LCD. The effect of this is a very low power consumption.

Memory LCDs have a fast response time. The entire frame can typically be updated at up to 60 Hz, which means dynamic content like animations and video are possible. The displays can be powered from standard 3.3 V or 5 V voltage sources which allows for very easy integration because they can be powered directly from the microcontroller supply or GPIO pins.

The drawback compared to EPDs, is that memory LCDs are not bistable. The display needs power to retain the image. Memory LCDs also need to invert the polarity across the panel regularly. Failing to do so would cause a charge buildup which eventually can cause the pixels to be stuck in one position. The MCU needs to provide a switching waveform which periodically toggles the polarity inversion, typically at least 1 Hz. During the polarity switch the current consumption spikes briefly.

## MCU Challenges When Driving a Low-Power Graphical Display

Since the displays discussed here have almost negligible current consumption in standby mode the **overall current consumption is determined by the MCU** or other parts of the system. Thus there is much to gain in terms of battery life by choosing the correct MCU and designing the software in an energy efficient way.

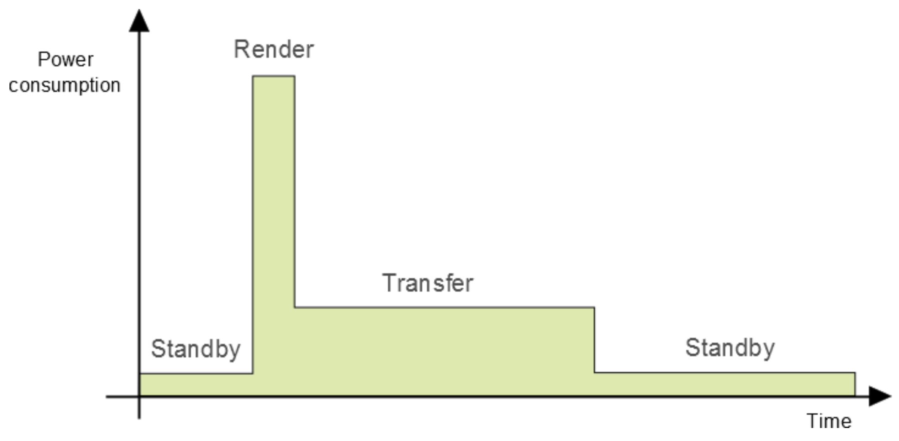


Figure 3: MCU power consumption profile

The overall power consumption of a graphical display MCU application can be divided in three main components:

1. Active mode power consumption while rendering an image
2. Update power consumption while updating the display (sending pixels)
3. Standby consumption while displaying a static image

Figure 3 shows an example of a power consumption profile. Standby mode naturally has the lowest current consumption. The render step is the only step which requires the CPU to be active and thus draws the most power, but the transfer step can be longer since it depends on the speed of the display. This figure only accounts for the MCU power consumption and not the display itself. The figure will look similar when including power consumption of a Memory LCD, only offset by the current drawn by the display since this value is mostly constant. For an EPD there is a large contribution when the display is updated, but no contribution for standby.

**Rendering a Frame**
Creating a user interface is more complex when using a graphical display than a more traditional approach like for instance, a segmented LCD. Printing a number to a segmented LCD is a relatively simple process. The same operation with a graphical display however, involves loading a font, selecting the appropriate characters for the number, rendering these to the correct location on screen and then transferring the pixels over a serial interface. Rendering more complex objects can be quite CPU intensive, especially when the pixel resolution is large.

**Updating the display**
Updating the display will typically mean sending pixels over an SPI interface. The time this takes will largely be determined by the display. Since this time period can be long it is not desired to keep the MCU in active mode. It is thus important to have DMA support so the frame buffer can be sent without CPU intervention. Even so, if the frame buffer is large it might require several DMA cycles to send the entire frame buffer which means the CPU has to wake up and reconfigure DMA during the transfer.

**Standby Mode**
During 'standby mode', where the display shows a static image, it is of course important that the MCU is consuming as little power as possible. If the display is infrequently updated the total energy consumption will actually be determined by the standby mode power consumption. Even in standby mode, it might still be required that the MCU keeps track of time or is able to respond to events, such as a key press or touch event from a user. The display itself may even require interaction in this mode. For example, Memory LCDs require the polarity inversion waveform in this mode. Furthermore the MCU might have to keep the frame buffer in memory, which means that also RAM has to be powered in this mode.

**Memory requirements**
In addition to the low power constraints there are extra memory requirements when interfacing a graphical display. To illustrate, a 320x240 QVGA screen with only 1 bit per pixel (i.e. a monochrome display) requires 9.4 kB of RAM to store *one* frame buffer. In addition the application will usually need a graphical drawing library, adding to the code size footprint. As EPD color displays are starting to enter the market, the memory requirements become even more severe.

**Optimizing power consumption**
It can be difficult to know exactly what part of an application is responsible for the power consumption. If the hardware layout is complex and consists of many components, figuring out the respective power drains is non-trivial. The same problem is found in software. For a large program, which routine is responsible for the majority of power consumption? Where should optimizing efforts be put? As projects become larger, these questions become increasingly difficult to answer.

For a display application an obvious parameter is the update rate. More display updates will necessarily cause a higher power consumption. But how high rate can be tolerated within the battery life time spec? Is it the rendering or update step which is drawing the most power? What is the base line power consumption in standby mode? These are all important questions for a system designer.

The problem is that this can be difficult to measure because it usually requires external tools like oscilloscopes and multimeters and the measurements are difficult to relate directly to problems in software.

## An Energy Efficient Solution

This chapter is aimed at showing how modern 32-bit low power microcontrollers can help solve the challenges outlined above. By using the EFM32 MCU as a reference, several features will be presented that allow a graphical display application to be energy efficient and achieve a long battery life time.

### Standby Mode

For an EPD application no interaction at all is required from the MCU when the display is showing a static image. The EFM32 can then be in its lowest sleep mode, drawing only 20 nA and is ready to wake up an asynchronous event, such as a key press. In case a timed wakeup is needed, an RTC can be added for a total current consumption of 400 nA.

With a Memory LCD the MCU is still responsible for generating the polarity inversion waveform in standby mode. On most MCUs the solution would involve using an RTC to generate a periodic interrupt. In the interrupt service routine the CPU would then toggle the polarity of a GPIO pin. On the EFM32 however, the whole process can be done in Deep Sleep mode. The EFM32 includes a Low Energy Timer (LETIMER) which can run in Deep Sleep and can autonomously generate a waveform and output this on a pin. The MCU current consumption for this mode is about 1 µA and this includes full CPU and RAM retention.  In case some of the RAM is not needed, the EFM32 provides capability to power down individual RAM blocks to reduce power consumption even further.

### Updating the Display

The fully featured EFM32 DMA can operate in sleep mode and consumes only 8 µA/MHz. It can transfer data from memory to any of the communication peripherals (USART/UART/EBI/I2C) which means that no matter which interface the display is using, the EFM32 can perform a low power transfer to it.

The EFM32 DMA also has a feature called 2D Copy, which is ideal for transferring large frame buffers. With this feature it is possible to copy a rectangular area (such as a part of an image) from a larger buffer. However, the main use of this feature in conjunction with low power displays is that it is capable of transferring larger amounts of memory than a normal DMA transfer. By using 2D Copy it is possible to transfer frame buffers of nearly any size in one DMA cycle, thus removing the need to wake up the CPU and reconfigure the DMA descriptors.

During transfer, the high frequency clock needs to run in order for the DMA and communication peripherals to operate. However, the speed can usually be slower than when the CPU is active. The flexible frequency bands of the internal RC oscillator allow the EFM32 MCU to save power by clocking down the frequency to match the transfer speed.

**Rendering the Image**

The powerful ARM Cortex-M3 core allows frames to be rendered quickly, allowing the CPU to go back to sleep earlier. The EFM32 implementation of the core is extremely efficient, consuming down to 150 µA/MHz in active mode. Since the core is a widely used ARM standard, it can take advantage of the huge ARM ecosystem of tools and vendors.

Energy Micro provides the powerful emWin graphics library from Segger free of charge to all of its customers. This library is optimized for both size and speed to provide optimal performance for MCU applications. It supports nearly any display on the market and drivers can be custom made if desired.

The EFM32 comes with large internal flash and RAM options up to 1 MB / 128 kB, allowing several large frames to be stored in memory. Internal memory is advantageous for both speed and cost compared to using external memory, but the EFM32 also provides a fast External Bus Interface (EBI) if external memory is needed.

**Energy Debugging**

Optimizing an application for energy consumption can be difficult. That is why Energy Micro has introduced the concept of *Energy Debugging*. All Energy Micro development kits (including the starter kits) include Advanced Energy Monitoring (AEM). The AEM continuously measures current consumption and supply voltage. This data can be transferred to a PC in real-time over the same USB cable used for normal debugging.

On the PC the data is collected and displayed on the screen by the energyAware Profiler. The data is also correlated directly with debug information, which means a developer can get information about what code is running at a given point in time. For instance, when the developer sees a current spike in the profiler output he or she can click directly on the graph and the line of code which was executing at that time will be highlighted.

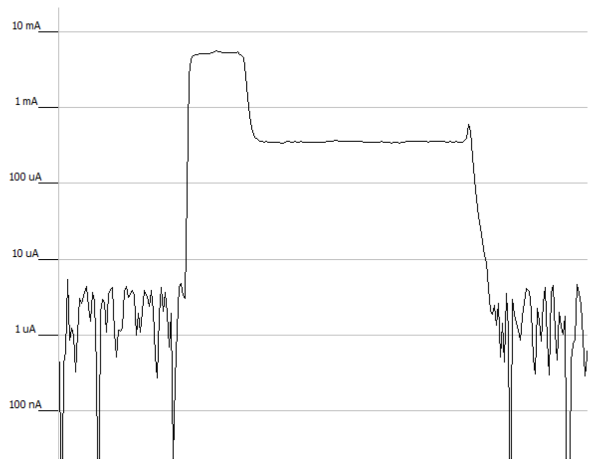


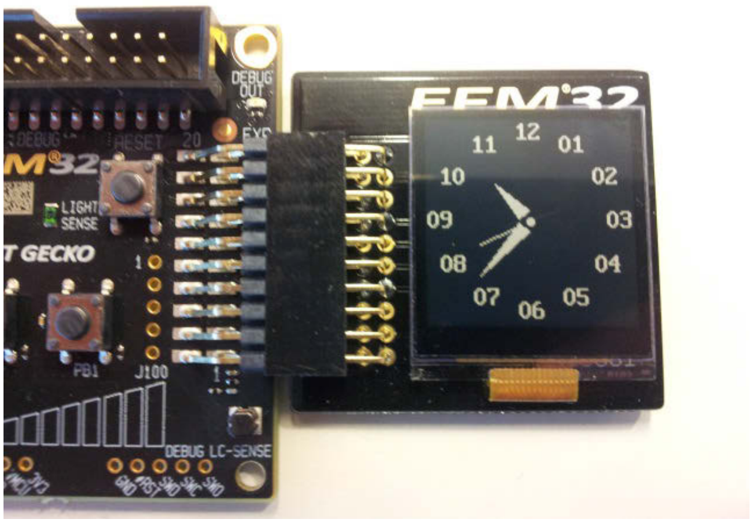
Figure 4: Energy profiling a Smart Watch application

An example plot from the energyAware Profiler is shown in Figure 4. The figure is taken from a smart watch application using a Memory LCD (see the next section). Most of the time is spent in Deep Sleep mode where only the RTC is running. The view is zoomed in on a spike which occurs every time the display is updated. At the beginning the MCU is deep sleep (standby mode), then entering active mode while the CPU is rendering the new frame. Sleep mode is then entered while DMA is transferring the finished frame from memory to the USART peripheral which sends the pixel over SPI to the display. The small spike at the end of the transfer is the interrupt telling the MCU that the DMA transfer is finished before the MCU goes back to standby mode.

## Application Example: Smart Watch

As an example of a low power GUI design, consider a smart watch. Wearable as a wrist watch it needs to run of a small coin-cell battery to keep the form factor and weight down. By using a graphical display the functionality can extend beyond simply showing the time. It could for example be connected to a pulse meter, showing the heart rate. It could show news or RSS feeds, weather data or any imaginable data the user might be interested in.

The display of choice for this application will be a memory LCD since the display will be updated quite frequently. Figure 5 shows an example of such an application. The example is using a 128x128 pixel memory LCD from Sharp. The implementation is an analog watch face which updates once every second to move the pointers. The average *total* power consumption of both MCU and display is only 13.5 µA. With a digital interface which is simpler to draw, the current consumption can be reduced to 4.4 µA, still updating every second.



## Summary

The combination of high processing power of the 32-bit ARM Cortex-M core at very low power consumption and ultra-low power and flexible sleep modes of the EFM32 makes it a great fit for creating low power graphical user interfaces with the latest display technologies. Adding to this is a set of custom-built low power peripherals and integrated energy debugging features, which helps developers create power critical applications with ease.

## More Information

Energy Micro provides both code examples and reference designs to help customers quickly get started with their display application. For more information about how to drive displays with the EFM32 MCU and tips on how to improve energy consumption see the following application notes and white papers (all available through Simplicity Studio or the Energy Micro website):

- AN0047 Interfacing Graphical Displays
- AN0048 Energy Optimized Display Application
- WP0002 Energy Debugging