EFM32
... the world's most energy friendly microcontrollers

Energy Optimized Display Application

AN0048 - Application Note

Introduction

This application note shows how the EFM32 can be used to create an application that requires a high resolution graphical display, yet still maintains a low power consumption.

This application note includes:

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

ARM ZERO TINY GECKO LEOPARD GIANT WONDER

SILICON LABS

1 Introduction

In recent years, the rapidly advancing field of display technologies have paved the way for creating rich and powerful user applications. From smartphones, e-book readers to TV screens, the trend is moving toward larger and larger displays with higher resolution. However, for an MCU application, a powerful display can often be off-limits, either because of price, CPU processing power or power budget.

In this application note we show how to use the EFM32's energy saving capabilities together with a Sharp low-power matrix memory LCD to create a powerful display application. The application is capable of driving a 128x128 pixel display drawing as little as 2 µA while showing a static image. Even when updating the frame every second the current consumption can be lower than 5 µA.

2 Application Description

The application demonstrated in this application note is a wrist watch. The user will be able to toggle between two modes: a *digital* or *analog* watch face. In the first mode the display will be used to display an analog watch face with moving pointers for hours, minutes and seconds. The latter displays the current time with numbers. The two modes are shown in Figure 2.1 (p. 3) and Figure 2.2 (p. 3) .

Figure 2.1. The analog watch face

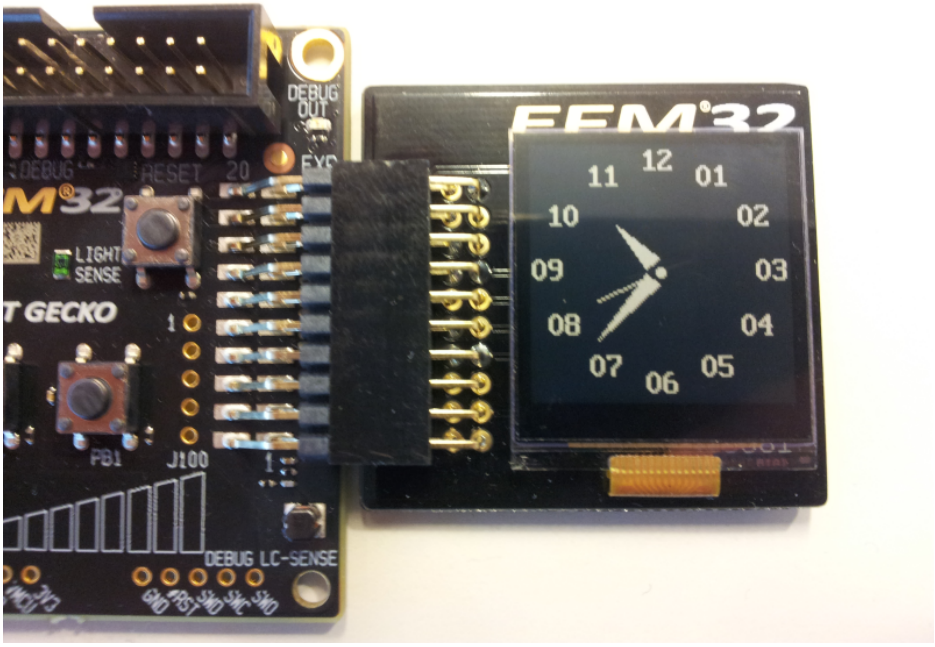
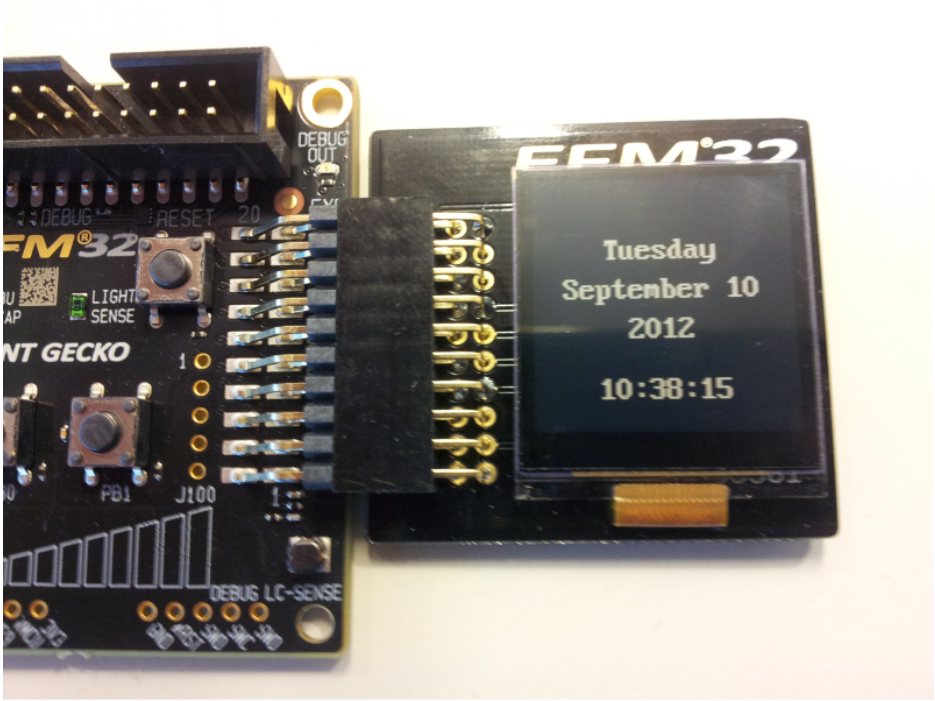


Figure 2.2. The digital watch face



2.1 Display

The display used in this example is a Sharp Memory LCD, model number LS013B7DH03. Memory LCD is a new techonology where the pixels does not have to be periodically refreshed like a regular LCD

panel. Once the pixels have been written to the display they will keep their configuration, only drawing a small amount of current.

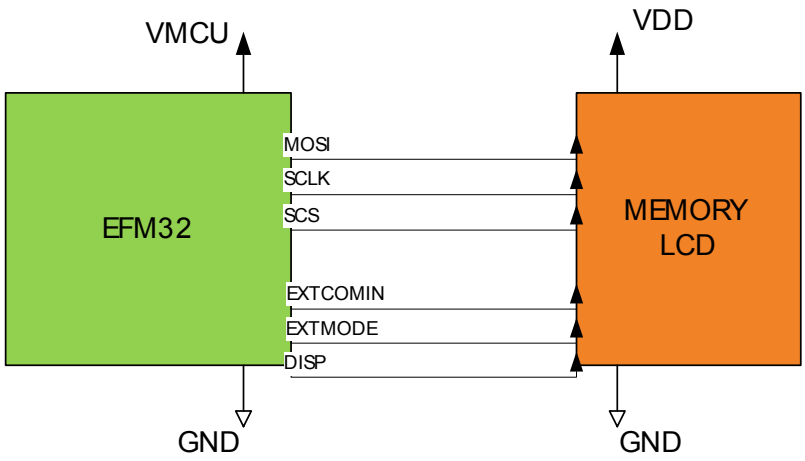
The display is a 1.28", 128x128 pixels monochrome display, with a 3-wire SPI interface. Apart from the SPI interface, the display requires a 3.3V power supply, and 3 extra pins named EXTMODE, EXTCOMIN and DISP (explained below).

The EXTMODE pin controls how *polarity inversion* is controlled. The display requires that the polarity across the Liquid Crystal Cell is reversed at a constant frequency. This polarity inversion prevents charge building up within the cell. If EXTMODE is LOW the polarity inversion is toggled by sending a special command over SPI. If it is HIGH polarity inversion is controlled by the EXTCOMIN pin.

If EXTMODE is HIGH the polarity inversion is armed for every rising edge of the EXTCOMIN pin. The actual polarity inversion is triggered at the next transision of SCS. The toggling frequency should be at least 1 Hz. If EXTMODE is LOW this pin is ignored.

The DISP pin toggles the display on or off (without the pixels losing their state). When LOW the display is off, when HIGH the display is on.

Figure 2.3. Display Interface



2.1.1 SPI Protocol

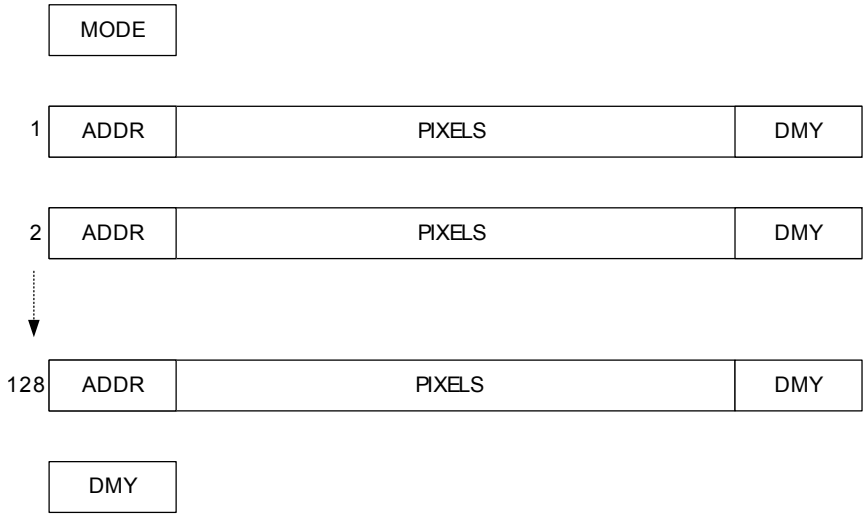
The SPI protocol consists of three modes. At the start of each SPI transfer, a mode command (1 byte) is sent first. The modes are:

- Update Image
- Toggle Polarity Inversion
- Clear Display

Each display update is initiated by sending the Update Image command. Following the command is N lines of data. Each line begins with a byte telling which line to update (the address byte). After the address byte follows 16 bytes (=128 bits) of pixel data followed by one byte of dummy data before the next address byte. After the last line, the display controller expects 2 bytes of dummy data before the transmission is ended. See Figure 2.4 (p. 5) .

The total number of SPI cycles needed for a full frame update is: 16 + 128 * (128 + 16) = 18448. The maximum baud rate is 1.1MHz which means the maximum update frequency is 60Hz. In other words, the minimum time to update the entire display is 16.8ms.

Figure 2.4. SPI protocol



2.2 Drawing Library

The emWin library from SEGGER is used to draw the frames. This library is provided free of charge to all Energy Micro customers and is installed through Simplicity Studio. The library is configured to draw directly to a frame buffer in memory. Updating the actual display from this frame buffer is done by a separate part of the program (discussed later).

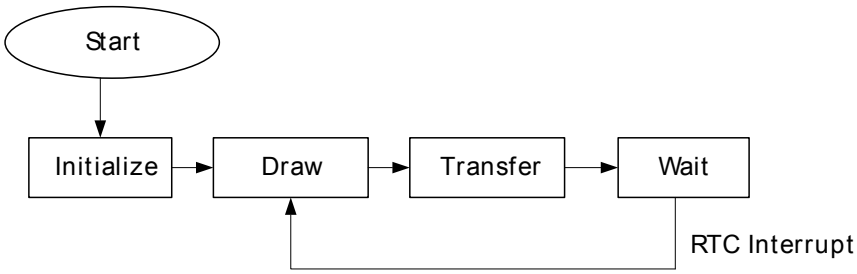
Note

For more information about emWin, see application note AN0047 and the SEGGER website, http://www.segger.com/emwin.html

2.3 Program Flow

The application program flow is depicted in Figure 2.5 (p. 5). At the start of the program the internal time reference is initialized and the RTC is set to generate an interrupt every second. The frame is then drawn with emWin to the frame buffer in memory. When the new frame is ready, it is transferred to the display over the SPI interface. The MCU then enters sleep mode and waits for the next RTC interrupt. On RTC interrupt, time is incremented by one second before the frame is redrawn and transmitted again.

Figure 2.5. Program flow



3 Power Saving Techniques

The EFM32 contains many features to help reduce power consumption. This chapter goes through the steps taken to reduce the power consumption of the watch example application. The steps include taking advantage of the EFM32's integrated peripherals, which are all designed for minimum energy consumption, using the tailored energy modes optimally and designing the software in such a way that the CPU spends as little time as possible in active mode.

3.1 Using DMA to Feed SPI

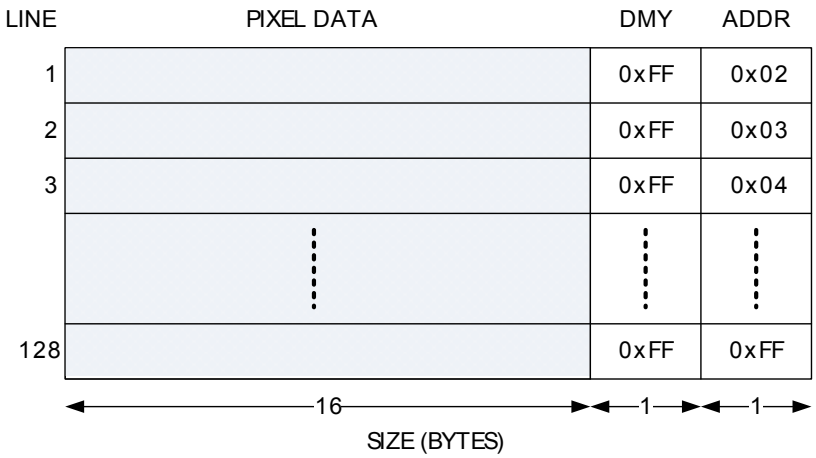
If the display is being updated by software, the CPU has to be awake for the entire duration of the transfer. As discussed in Section 2.1.1 (p. 4) , this time is significant. However, by using DMA the CPU can be sleeping while the display is being updated.

To make this work, also the dummy bytes and line addresses have to be encoded in memory. Figure Figure 3.1 (p. 6) shows this is implemented. Each line of pixels is followed 8 dummy bits and 8 address bits before the next line. When the display is to be updated, the CPU first has to send the command byte and address of the first line and then start the DMA transfer. The DMA will then take care of sending all the necessary data including control signals to the display, while the CPU is sleeping.

Note

To make emWin draw correctly to this frame buffer the display width is set to 144 pixels, but before any drawing is done the view is *clipped* to 128 pixels

Figure 3.1. Frame buffer structure



3.2 Removing Redundant SPI Transfers

In both display modes (analog and digital watch), there are parts of the screen which remains the same between frames. The display does not require the entire frame to be sent at each update. Instead only the rows which has been changed since the last frame will be sent.

For the analog watch this means we only need to figure out which pointers have moved and what area have they swept across. If a pointer has length L, angle t with the "twelve o'clock position" and is fixed at the origin, the first row touched by a pointer is given by Equation 3.1 (p. 6). The last row is found in the same way and only the interval [R_min,R_max] is sent over the SPI interface.

First row touched by an analog watch pointer

$$R_{MIN} = \min(HEIGHT / 2, HEIGHT / 2 - L * \cos(t))$$
$$R_{MAX} = \max(HEIGHT / 2, HEIGHT / 2 - L * \cos(t)) \quad (3.1)$$

For the digital watch, the algorithm is even simpler and the number of rows sent simply depend on the font height.

3.3 Efficient use of Energy Modes

Table 3.1. Tasks and energy modes

	DRAW	TRANSFER	WAIT
Active modules	RTC, CPU	RTC, USART, DMA	RTC
Lowest energy mode	EM0	EM1	EM2

One of the most critical parameters when designing an energy efficient MCU application is how long time is spent in each energy mode. Table 3.1 (p. 7) shows the different tasks of this application along with which modules are required for each task. The last row shows the lowest energy modes where the modules in use are available.

3.4 Adjusting Clock Frequency

During the TRANSFER task when transmitting the frame from memory to display, the high frequency (HF) clock has to be active. The display's highest baud rate is 1.1MHz. Maximum bit rate for the USART module is half the HF clock frequency. Consequently if the HF clock is higher than 2.2MHz during transfer the MCU is wasting energy in unused clock cycles.

The LFRCO clock can be configured to run on several different frequencies. The clock can be changed by software, even when the CPU is running from the LFRCO. See the device Reference Manual for which frequency bands are available.

There is only one option less than 2.2 MHz, so this option (1 MHz) is selected. This means the actual SPI transfer will only run at 500 kHz. This choice implies that the MCU spends more than twice the amount of time on the SPI transfer. However, the total energy consumption is still lower than selecting 7 MHz. The higher speed does not make up for the wasted clock cycles.

3.5 Let LETIMER Toggle Polarity Inversion

The Sharp Memory LCD display requires that the pin EXTCOMIN is toggled at a constant frequency, to avoid charge buildup. The normal way of implementing this on an MCU is to let the RTC trigger an interrupt at this frequency and toggle a GPIO pin in the interrupt handler. But the EFM32 has a peripheral, the LETIMER, which can do this process automatically, without having to wake up the CPU.

The LETIMER has two output pins. On each of these it is possible to specify a an action whenever the counter overflows. To comply with the specifications of the display, LETIMER is set to overlfow at 1 Hz and toggle the output polarity at each overflow. The EFM32 can therefore be in EM2 all the time between frame changes.

3.6 Render Next Frame During SPI Transfer

To reduce the time spent outside EM2, it is possible to render the next frame during the transfer of the current one. To be able to do this, double frame buffers are needed. While frame buffer 1 is being transferred, the next frame is rendered to frame buffer 2. For the next frame the roles are switched.

With this scheme the Draw and Transfer are done in parallel. The MCU has to stay in EM0 until the drawing task is active. When the drawing is complete, the MCU can enter EM1 while the rest of transfer completes.

3.7 Precalculating / Prerendering

Since many of the frames will be similar it makes sense to precalculate and prerender some of the elements. In this way we avoid doing the same calculations over and over for every frame. The tradeoff is that we are using more memory. As long as enough memory is available this is an efficient way to reduce the time spent in active mode.

For the analog watch face there are 3 polygons (the pointers) which needs to be rotated before they can be drawn. The rotation uses trigonometric functions which include floating point calculations. For MCUs without a floating point unit (FPU) these operations has to be done in software and requires many clock cycles. At initialization, these polygons are rotated and the results are stored in a table. At every new frame the CPU only has to look up the correct value in this table instead of performing the calculation again.

Note

The floating point operations are CPU intensive on all devices with a Cortex-M3 core as these devices does not contain a Floating Point Unit (FPU). Some of the newer EFM32 devices, such as the Wonder Gecko, will use a Cortex-M4F core which contains an FPU. Another way to speed up these calculations is to use the fixed point library provided by ARM.

In many display applications some parts of the image are static. Normally a background image is drawn to the screen first and then other elements are drawn on top of this. If the background image never changes, this image can be stored in a separate buffer and copied to the frame buffer before drawing the frame. Thus a complex drawing operation can be reduced to a simple `memcpy()` call.

Note

The background image copy could also have been done with DMA. However, since the frame buffer is quite small (only 640 words), it is actually more efficient to let the CPU perform the copy operation.

4 Energy Profiling

The energyAware Profiler is an extremely useful tool when designing an energy critical application. The watch application has several different tasks (DRAW, TRANSFER, WAIT), each contributing to the total energy consumption. With the energyAware Profiler it is easy to see which tasks is contributing the most to the energy consumption and thus which parts of the application should be the focus for optimization.

4.1 Analog Watch

Figure 4.1. Profiling analog watch before optimization

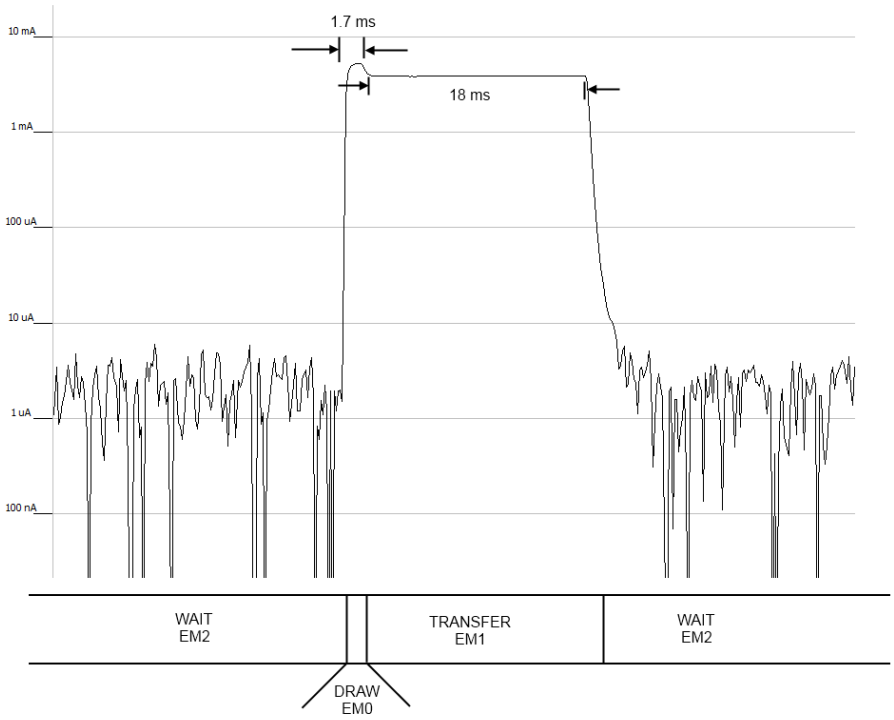
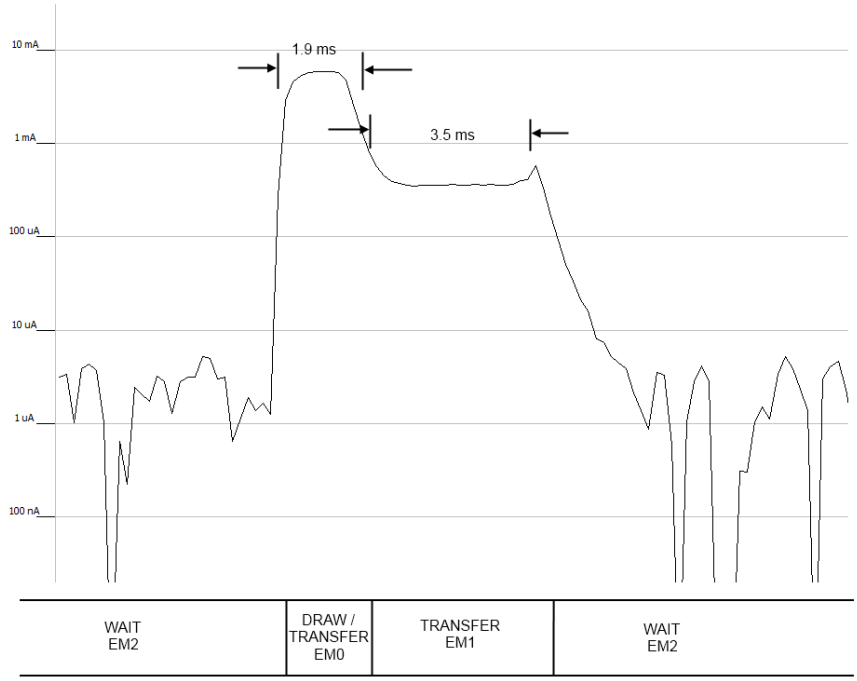


Figure 4.1 (p. 9) shows the profiler output for the analog watch. The profiler output is mostly flat with a base current consumption of 2 µA (from the WAIT task) and a peak at every second where the frame is updated. The view in Figure 4.1 (p. 9) is zoomed in on one peak. In this case the CPU both the drawing and transfer step is done in EM0 and the core frequency is always 14 MHz. The current consumption is about 5 mA in the DRAW task and 3.9 mA in the TRANSFER task.

Figure 4.2 (p. 10) shows the same view after optimization. The current consumption during the TRANSFER step is greatly reduced to about 360 µA.

Figure 4.2. Profiling analog watch after optimization



The average current consumption (over a 60sec interval) is shown in Table 4.1 (p. 10). Each row corresponds to a different set of enabled optimizations. The lowest consumption, while drawing a new frame and updating the display every second, is 26.9 µA.

Table 4.1. Analog watch average current consumption with different optimizations

Double buffer	Redundant rows	Clock freq	DMA SPI	Current consumption
				79.0 µA
			x	47.3 µA
		x	x	22.7 µA
	x	x	x	13.8 µA
x	x	x	x	13.5 µA

4.2 Digital Watch

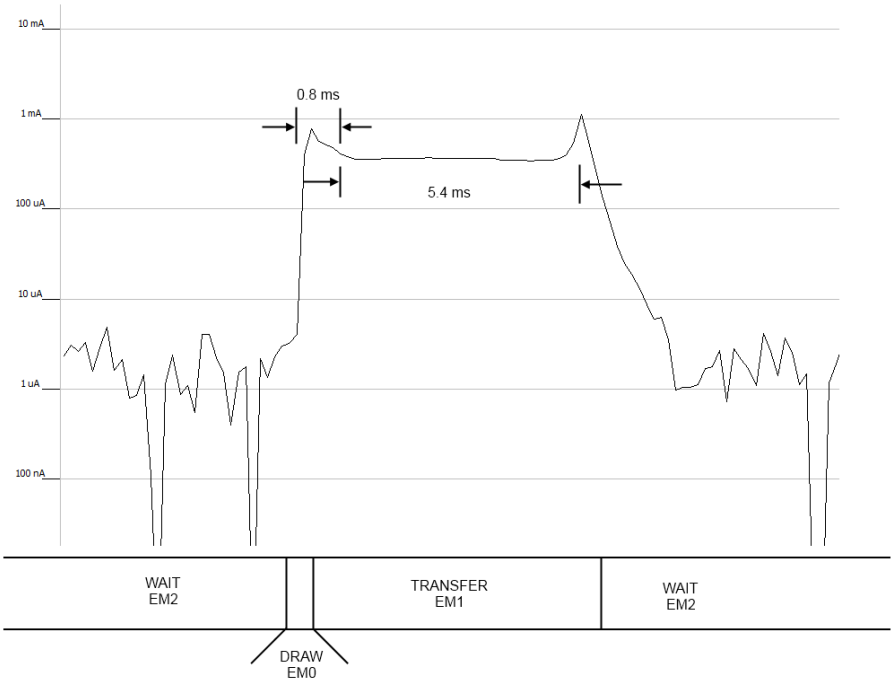
The digital watch can be made even more energy efficient than the analog. The two main reasons are:

- Less of the display has to be updated
- The drawing operations are much simpler

While the analog watch needs to update up to half the display for each frame, the digital watch only needs to update the lines displaying the time. For the font used, this is only 16 lines. On the digital watch there are no overlapping graphical elements. There is no need for redrawing the background, the current time can simply be redrawn on top of the previous frame.

The profiler output for the digital watch is shown in Figure 4.3 (p. 11) and the average current consumption is only 4.4 µA.

Figure 4.3. Profiling the digital watch



5 Touch Slider Demo

This example is a larger application and meant as a demo for the STK3700 with the MemoryLCD plugin board. It is an extension of digital/analog watch example discussed until now. In addition to the two watch faces this demo also shows an animated video, a static image and real-time plot of the light sensor. The touch slider on the Starter Kit can be used to switch between the screens, much in the same way as on a smartphone.

In addition to the analog/digital watch faces, the following applications are demonstrated in this example:

Video

The video screen loops a prerendered video of 60 frames at 25 FPS. The images have been created with the emWin Bitmap Converter and prerendered to Flash.

Average current consumption: 380 µA

Static Image

A static image is shown to illustrate that no CPU intervention is required to keep this image on the screen. The EFM32 can continuously stay in EM2 while the LETIMER is toggling the EXTCOMIN pin and LESENSE is monitoring the touch slider. The image is only sent once to the display.

Average current consumption: 1.9 µA

Light Sensor

The light sensor on the STK is measured repeatedly and the result is plotted on the display. This example takes advantage of the fact that very little of the screen is updated with each new data point and thus very few lines have to be sent to the display for each measurement.

Average current consumption: ~40 µA

5.1 Code Overview

The rendering code is divided into modules called *screens*. The screens.c file manages the different screens calls the functions to render the active screen. Each screen module must provide a few callback functions outlined below. When switching between screens, the functions in slide.c are called instead to take care of sliding animation.

Each screen can use an animation timer to request frame updates. The animation timer simply generates an interrupt at a specified interval and sets a flag. This flag is checked in the drawing loop which triggers a frame update.

The file caplesense.c contains the driver for the capacitive slider. This file uses LESENSE to measure the capacitive touch buttons and interpolates the values to give a position along the slider.

5.1.1 Screen Callback Functions

The following callback functions are used when drawing a screen. To implement a new screen these functions must be defined and the `screens` array must be updated with the appropriate function pointers.

preview()
This function will draw the image that will be used when switching screens

prepare()
This function is called before a screen is made active. It is typically used to initialize variables or start the animation timer.

draw()

This function will render the actual screen. The draw() method is called repeatedly when the screen is active. The draw() function should also update the drawing limits.

finish()

The function is called when the screen is made inactive, i.e. the user has switched to another screen. This is typically used to clean up variables or stop the animation timer.

getLimits()

Retrieves the limits, the first and last row of the display that needs to be updated. This is used by the DMA transfer to avoid sending unnecessary rows

6 References

[1] Sharp Memory LCD LS013B7DH03 Data Sheet

[2] Sharp LS013B4DN02 Application Note
http://www.sharpmemorylcd.com/resources/LS013B4DN02_Application_Info.pdf

7 Revision History

7.1 Revision 1.04

2013-09-03

New cover layout

7.2 Revision 1.02

1 Revision 1.03

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

2013-01-29

Added new touch demo software example

Fixed bug where wrong weekday was shown

7.3 Revision 1.01

2012-11-12

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

7.4 Revision 1.00

2012-09-10

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

Silicon Laboratories 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 Silicon Laboratories 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. Silicon Laboratories 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. Silicon Laboratories 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 Silicon Laboratories. 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. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories 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

Silicon Laboratories Inc., Silicon Laboratories, the Silicon Labs logo, Energy Micro, EFM, EFM32, EFR, logo and combinations thereof, and others are the registered trademarks or trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

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

1. Introduction 2
2. Application Description 3
    2.1. Display 3
    2.2. Drawing Library 5
    2.3. Program Flow 5
3. Power Saving Techniques 6
    3.1. Using DMA to Feed SPI 6
    3.2. Removing Redundant SPI Transfers 6
    3.3. Efficient use of Energy Modes 7
    3.4. Adjusting Clock Frequency 7
    3.5. Let LETIMER Toggle Polarity Inversion 7
    3.6. Render Next Frame During SPI Transfer 7
    3.7. Precalculating / Prerendering 8
4. Energy Profiling 9
    4.1. Analog Watch 9
    4.2. Digital Watch 10
5. Touch Slider Demo 12
    5.1. Code Overview 12
6. References 14
7. Revision History 15
    7.1. Revision 1.04 15
    7.2. Revision 1.02 15
    7.3. Revision 1.01 15
    7.4. Revision 1.00 15
A. Disclaimer and Trademarks 16
    A.1. Disclaimer 16
    A.2. Trademark Information 16
B. Contact Information 17
    B.1. 17

List of Figures

2.1. The analog watch face 3
2.2. The digital watch face 3
2.3. Display Interface 4
2.4. SPI protocol 5
2.5. Program flow 5
3.1. Frame buffer structure 6
4.1. Profiling analog watch before optimization 9
4.2. Profiling analog watch after optimization 10
4.3. Profiling the digital watch 11

List of Tables

3.1. Tasks and energy modes 7
4.1. Analog watch average current consumption with different optimizations 10

List of Equations

3.1. First row touched by an analog watch pointer 6

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