EFM®32

... the world's most energy friendly microcontrollers

Driving Electronic Paper Displays (E-paper)

AN0063 - Application Note

Introduction

This application note shows how to drive an Electronic Paper Display (EPD) with the EFM32. The application note makes use of an EPD extension board made by Pervasive Displays which is connected to the EFM32GG-STK3700 Starter Kit.

This application note includes:

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

1 Electronic Paper Displays

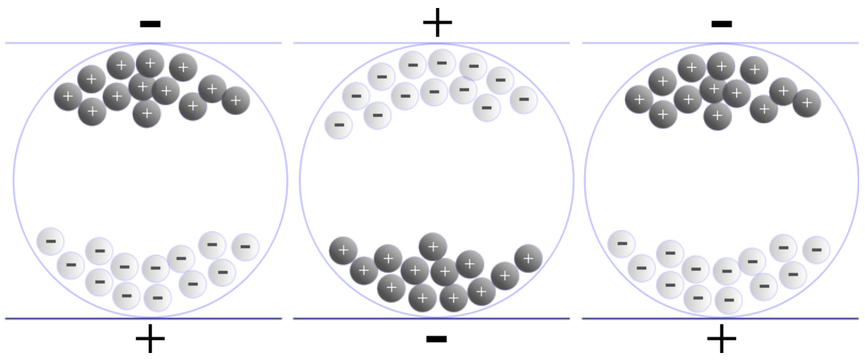
Electronic Paper Displays (EPDs) are types of displays that are *reflective* and *bistable*. Reflective in this case means that they rely solely on ambient light and does not use a backlight. Bistable is the property of retaining an image even when no power is connected.

EPDs are commonly used in e-readers, industrial signage and electronic shelf labels. Their properties are ideal for applications which do not update the image frequently. Since the display draws *no* current when showing a static image, they allow for a very long battery lifetime.

1.1 EPD Cell

The pixels in an EPD 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 1.1. EPD cells



1.2 Benefits of EFM32

Since EPDs have the ability to draw no current, the current consumption of the MCU and rest of the application becomes very visible. The flexible Energy Modes of the EFM32 allows the MCU to always draw as little current as possible. In many EPD applications Energy Mode 4 can be used, where the MCU consumption can be as low as 20 nA.

Memory is important to save frame buffers and images. The EFM32 has large memory options, both for Flash and SRAM. At the time of writing the largest options include 1 MB of Flash and 128 kB of SRAM.

EPDs require longer time to update in cold environments. The EFM32 contains an internal temperature sensor that can be used to adjust the time.

1.3 Considerations

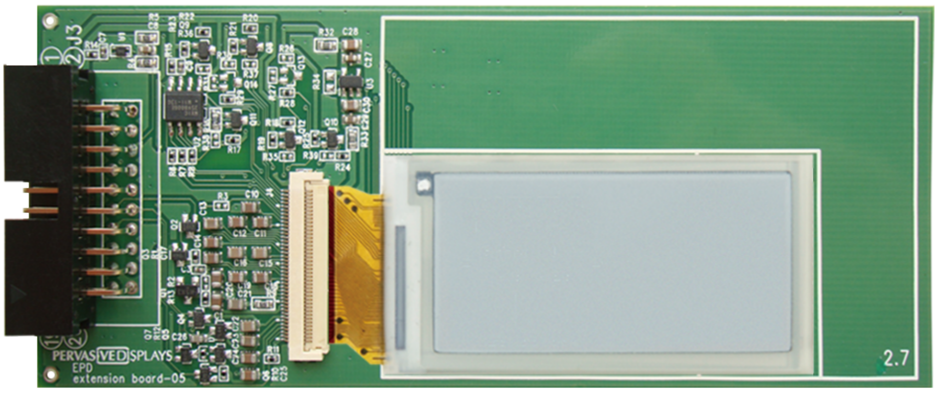
Even though the EPDs draw no current while showing a static image, they do require a significant amount of current while updating the display. The display update itself also takes a long time (typically 1-2 seconds). EPDs are therefore not suited to applications which require a high update frequency.

The contrast ratio (CR) does decay after a while. Pervasive Displays specifies a contrast ratio decay of CR 9:1 -> 7:1 over three weeks for their panels. [pdi_faq]

Ghosting is the effect of seeing artifacts of a previous image on the display. EPDs are by nature prone to ghosting effects. When a pixel is updated it can happen that not all of the particles are able to move to the other side of the EPD cell. To the user it looks like a grey shade of the old frame is embedded in the new frame. To get around this problem, a frame is usually written multiple times to the display.

2 EPD Extension Board

Figure 2.1. Pervasive Displays EPD Extension Board



This application note makes use of the EPD extension board made available by Pervasive Displays (PDI) to illustrate how to drive an e-paper display with an EFM32 microcontroller. The EPD extension board is available for sale and the full schematics and documentation can be downloaded for free. Documentation and ordering information are found on their webpage [pdi_kit].

The extension board includes:

- Flexible flat panel connector to connect display
- 8 MB Flash to store images
- Temperature sensor
- 20-pin header to connect MCU

The flat panel connector is compatible with 1.44", 2.0" and 2.7" EPD panels. The signals required to drive the panel are routed out on the 20-pin header. The board comes with a 20-pin connector cable which terminates in standard jumper wires on the opposite side. This allows the EPD board to easily connect to the EFM32 via the starter kit expansion header, see Figure 2.2 (p. 4). Table 2.1 (p. 5) shows how to connect the EPD extension board to the EFM32GG-STK3700 Starter Kit.

Figure 2.2. EPD and STK connected

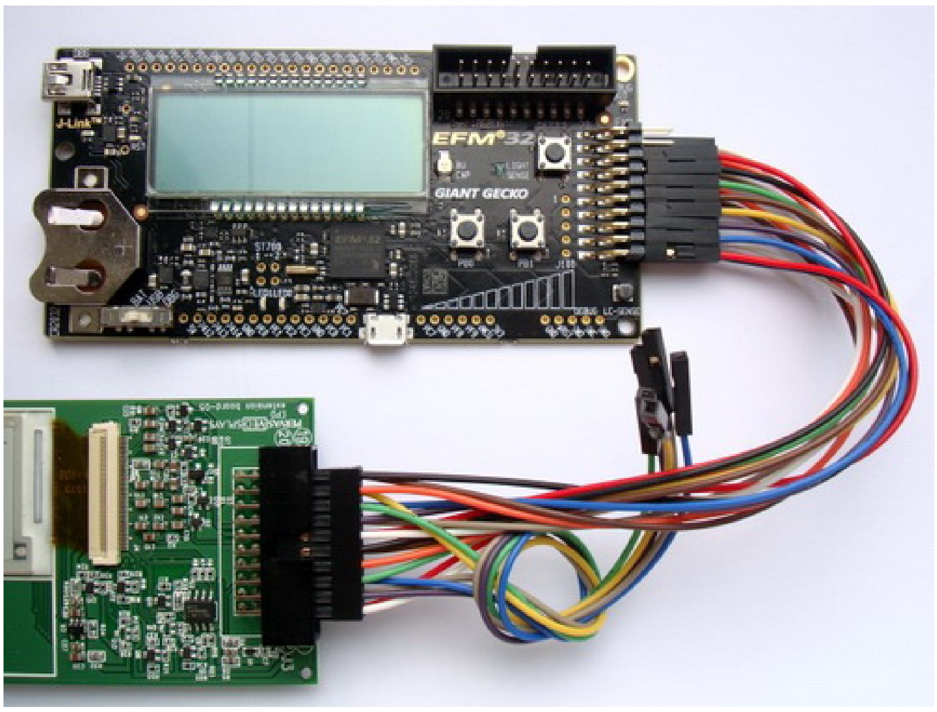


Table 2.1. Connection Table

EPD Pin	Function	EFM32 Pin	STK Pin	Color
1	VDD	VMCU/PC4 [1]	2/7[1]	Red
2	LED	PC0	3	White
3				Grey
4				Purple
5				Blue
6	Temperature	PD4	12	Green
7	SPI_CLK	PD2	8	Yellow
8	BUSY	PC6	15	Orange
9	PWM	PD3	10	Brown
10	/RESET	PD5	14	Black
11	PANEL_ON	PD6	16	Red
12	DISCHARGE	PD7	17	White
13	BORDER	PB12	13	Grey
14	SPI_MISO	PD1	6	Purple
15	SPI_MOSI	PD0	4	Blue
16				Green
17				Yellow
18	FLASH_CS	PB11	11	Orange
19	/EPD_CS	PC5	9	Brown
20	GND	GND	1	Black

[1] Pin can be connected to PC4 when running the 'EPD Demo' example to save power. This pin will be pulled low when display is not updating to remove all power to the EPD board. When running the 'PDI Kit Demo' example the pin must be connected directly to VMCU.

3 Driving an EPD with EFM32

This chapter will explain what the MCU needs to do in order to update the EPD panel. All of the information in this chapter is based on the documentation released by Pervasive Displays [cog_timing]. See this documentation for details about each step including SPI commands and timing.

3.1 Chip-on-Glass Driver

The EPD panels from PDI come with an integrated Chip-on-Glass (COG) driver that controls the lines and columns on the panel. The COG has a 3-wire SPI interface that is used to accept commands from the MCU.

3.2 Charge Pump

When writing a new image to the display, the panel itself requires a large voltage across it in order to drive the pixels. To achieve this without an external voltage supply a charge-pump circuit is used. The MCU must provide a PWM signal between 100 - 300 kHz in order to charge up this voltage.

3.3 Image Update Sequence

To draw an image on the EPD panel, the MCU must:

1. Power up and initialize the COG.
2. Write the image data in multiple passes to avoid ghosting and improve contrast.
3. Power off the COG driver.

During initialization the MCU communicates with the COG by sending SPI commands and must adhere to specific timings laid out by the COG documentation. The MCU must also provide the PWM waveform as part of the initialization sequence.

The MCU should keep two frame buffers in memory: the new image to be displayed, and the old image currently visible on the panel. During image update the panel is updated in four stages:

1. The current image is inverted
2. The entire panel is drawn white
3. The inverse of new image is drawn
4. The new image is drawn

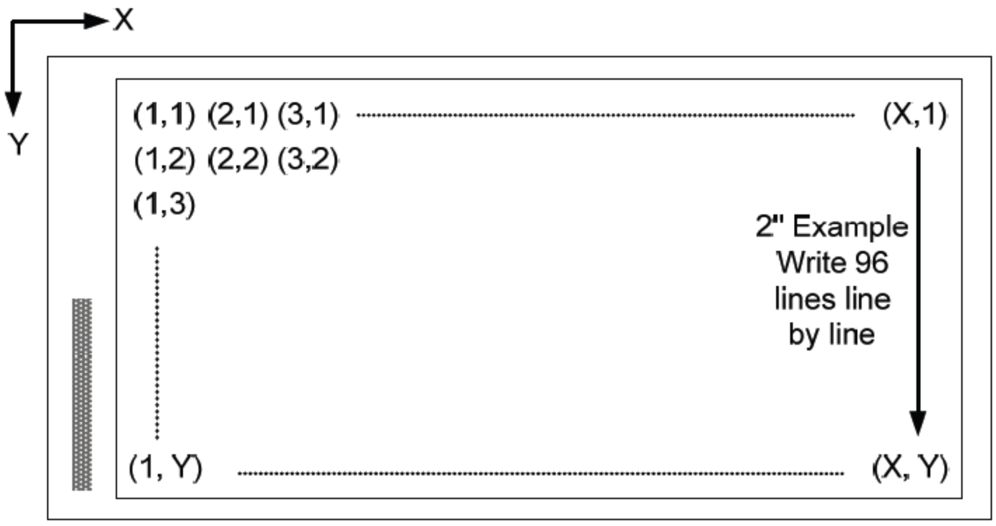
During each stage the same frame is written multiple times to the display. The number of times a frame should be rewritten is dependent on both the type of display and the current temperature. A colder environment makes the particles within the pixel cells move more slowly and therefore requires the panel to be rewritten more. If the application should work in varying temperature conditions the temperature should be measured before writing to the display. The EPD extension board comes with a temperature sensor for this purpose. The EFM32 also has an internal temperature sensor that can be used.

After the frame has been written, the COG should be powered down. This process also has a specific sequence of commands and timings the MCU must obey. During this sequence the pixel registers are first cleared and then the charge-pump capacitors are discharged. After the power-down sequence is finished, all power to the panel can be removed and the image will be kept on the panel.

3.4 Frame Buffer

The COG expects each frame in a special format. A pixel is labeled (x,y) where (1,1) is the upper left of the display. Each pixel is described by 2 bits which are either BLACK (0b11), WHITE (0b10) or NOTHING (0b01). Sending a NOTHING value means to leave the current pixel as is.

Figure 3.1. EPD Frame (from [cog_timing])



During a frame update, pixel data is sent one line at a time. Each line starts with all the even pixels beginning from the right (e.g. for a 200 px wide display: (200,y), (198,y) ...). Then follows the scan lines which indicate which row to update. The scan line format is a sequence of bits where pairs of bits are mapped to a line. The two bits should be 0b11 for the current line and 0b00 for all others (e.g. when writing the second line the scan lines will start 001100...). After the scan lines comes all the odd pixels starting from the left ((1,y), (3,y) ....).

3.5 Rendering with emWin

emWin is a graphics library made by Segger which is built for embedded processors and microcontrollers. The library is licensed for free to all EFM32 customers and available as a static library through Simplicity Studio.

To use this library to render frames on the EPD, emWin must be configured to draw a black-and-white (1 bpp) image to a frame buffer in RAM. This frame buffer is then converted to the EPD format when transmitting it to the COG.

Documentation for emWin can be found on the Segger website: http://www.segger.com/emwin.html

Note

The current version of emWin (v5.16) has a bug in the driver which only handles display sizes (width, height) that are a multiple of 32. Therefore the emWin frame buffer must be rounded up to the nearest compatible size. When the frame is converted to EPD format, only the pixels that will fit are copied.

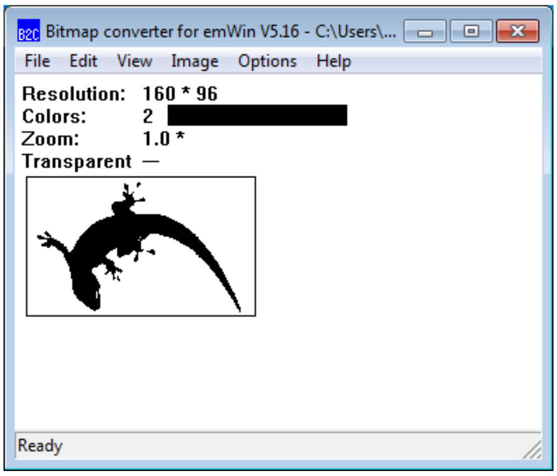
3.5.1 Converting Images

emWin includes a PC tool to convert images to a format that can be used by emWin. The BmpCvt.exe (Bitmap Converter) tool can be found in the Simplicity Studio folder under reptile\emwin\exe.

The tool can open common image formats (GIF, BMP, PNG) and save the output as a normal C file that can be included and compiled into the MCU application.

When using the Bitmap Converter to prepare an image for use on an EPD, first convert the image to black and white [Image -> Convert Into -> BW (1 BPP)]. Then, scale the image to fit on the screen: [Image -> Scale]. When done, click [File -> Save As] and make sure the file type is set to "C bitmap file (*.c)". When prompted to select a format, choose "1 bit per pixel" and no palette.

Figure 3.2. emWin Bitmap Converter


Bitmap converter for emWin V5.16 - C:\Users\...
File Edit View Image Options Help

Resolution: 160 * 96
Colors: 2
Zoom: 1.0 *
Transparent —

Ready

4 Power Consumption

The EPD only consumes power during a display update. A display update however takes a significant amount of time, typically between 1-2 seconds at room temperature. During this time the MCU must complete the power-up and power-down sequences and transmit frames repeatedly to the panel.

To optimize power consumption during a display update, the MCU should be put in its optimal Energy Mode at all times. When performing CPU intensive tasks like rendering a new image, the MCU should run at a high frequency. During the update stages, when the frame is transmitted over SPI to the EPD panel the clock frequency can be slower since the stage should nevertheless run for a predetermined amount of time. During delay periods, when the MCU is waiting for the COG to be ready, the MCU can be in a sleep mode.

The main parameter that can be optimized to save power during a display update is the 'stage time'. This is how long each stage should take. The PDI documentation [pdi_ref] defines sensible defaults for each panel along with a 'temperature factor' that is used to extend this time for colder environments. The stage time parameter for each panel is configured in the top of the cog.c file, while the temperature factor is defined in temp.c.

Figure 4.1. EPD + MCU power consumption

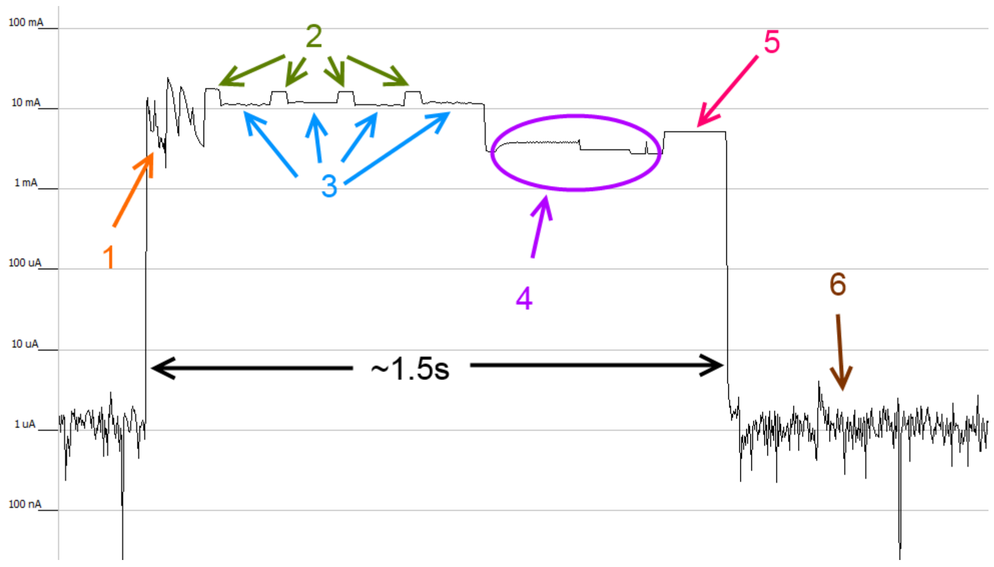


Figure 4.1 (p. 9) shows a breakdown of the power consumption of the MCU + EPD during a display update. This figure is taken with the energyAware Profiler while the MCU is running the 'EPD Demo' example in this application note. Referring to the numbers in the figure:

1. Initialization sequence and PWM to power up the COG and charge-pump.
2. EPD frame is calculated from the two frame buffers in memory.
3. EPD frame is transmitted to the panel over SPI.
4. COG shutdown sequence.
5. Discharging capacitors.
6. Power is cut to the panel. MCU is in EM2.

4.1 Choosing a Sleep Mode

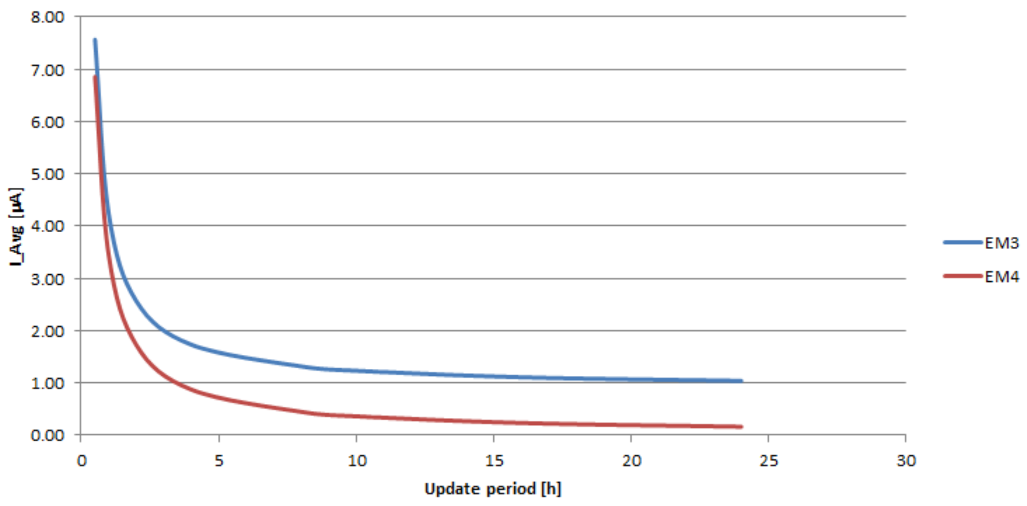
Since EPDs do not require power or interaction while displaying a static image, this enables the possibility to use EM4 as a sleep mode. Whether it makes sense to use EM4 depends on the application, specifically if other tasks needs to be performed during sleep, and the update frequency.

In EM4 the EFM32 can draw as little as 20 nA of current and wake up on a GPIO interrupt (NB: only on selected pins, see reference manual). However, when waking up from EM4, the MCU has to go through a full reset and run the startup/initialization code gain, which will add to the average current consumption. Therefore, it is applications which have long sleep periods that gains the most by using EM4.

The theoretical advantage of using EM4 is shown in Figure 4.2 (p. 10) . The graph is calculated assuming a simple application that stays in a sleep mode and updates the display based on a periodic GPIO interrupt. The horizontal axis is the update period and the vertical axis shows the average current consumption. At short update periods (high update frequency) the current consumption is dominated by the active mode current consumption of the MCU and the current the EPD draws when updating the image. In this case, the sleep mode current can be ignored. However, as the update period gets longer, the averge current consumption becomes dominated by the sleep mode current consumption.

The values used to calculate Figure 4.2 (p. 10) is using data sheet numbers for the EFM32GG990F1024 current consumption and measurements on the 'EPD Demo' example in this application note using a 2" panel (an update takes about 1.5s and consumes on average about 8 mA of current).

Figure 4.2. Average current consumption (MCU + EPD)



5 Software Examples

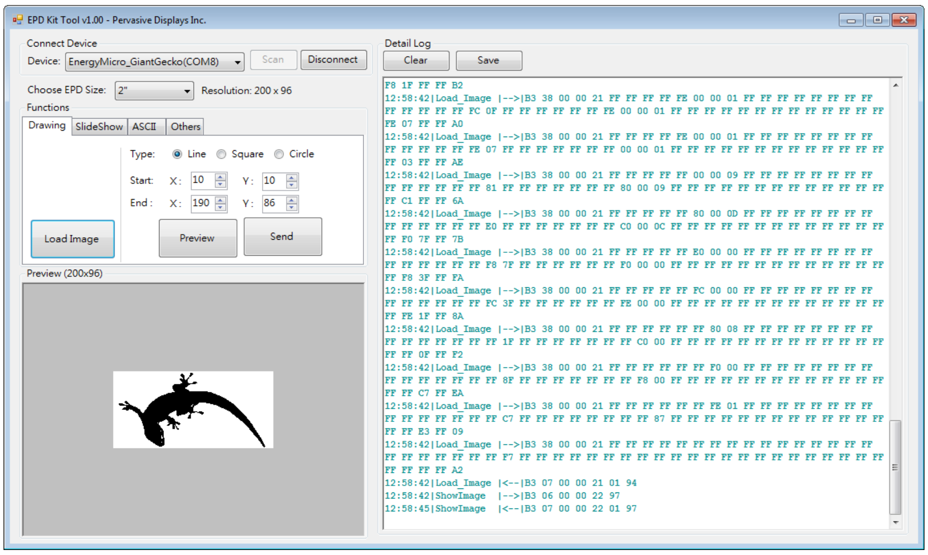
This application note comes with two software examples.

5.1 PDI Kit Demo

This example is the demo application provided by Pervasive Displays and available from their website [pdi_kit]. On the same website it is possible to download the EPD Kit Tool which can upload images directly from the PC (Figure 5.1 (p. 11)). When running this example, connect a USB cable to the Micro-USB connector at the bottom center of the EFM32GG-STK3700 Starter Kit.

The PC tool communicates with the EFM32 over USB and can transmit pictures, text and geometric shapes to the MCU. The MCU sends these images to the EPD.

Figure 5.1. EPD Kit Tool



Device: EnergyMicro_GiantGecko(COM8)

Choose EPD Size: 2"   Resolution: 200 x 96

Type: [✓ Line] Square Circle

Start: X: 10 Y: 10

End : X: 190 Y: 86

```
F8 1F FF FF B2
12:58:42|Load_Image |-->|B3 38 00 00 21 FF FF FF FF FE 00 00 01 FF FF FF FF FF FF FF FF
FF FF FF FF FF FC 0F FF FF FF FF FF FF FE 00 00 01 FF FF FF FF FF FF FF FF FF FF FF FF FF
FE 07 FF FF A0
12:58:42|Load_Image |-->|B3 38 00 00 21 FF FF FF FF FE 00 00 01 FF FF FF FF FF FF FF FF
FF FF FF FF FF FE 07 FF FF FF FF FF FF FF 00 00 01 FF FF FF FF FF FF FF FF FF FF FF FF FF
FF 03 FF FF AE
12:58:42|Load_Image |-->|B3 38 00 00 21 FF FF FF FF FF 00 00 09 FF FF FF FF FF FF FF FF
FF FF FF FF FF FF 81 FF FF FF FF FF FF FF 80 00 09 FF FF FF FF FF FF FF FF FF FF FF FF FF
FF C1 FF FF 6A
12:58:42|Load_Image |-->|B3 38 00 00 21 FF FF FF FF FF 80 00 0D FF FF FF FF FF FF FF FF
FF FF FF FF FF FF E0 FF FF FF FF FF FF FF C0 00 0C FF FF FF FF FF FF FF FF FF FF FF FF FF
FF F0 7F FF 7B
12:58:42|Load_Image |-->|B3 38 00 00 21 FF FF FF FF FF E0 00 00 FF FF FF FF FF FF FF FF
FF FF FF FF FF FF F8 7F FF FF FF FF FF FF F0 00 00 FF FF FF FF FF FF FF FF FF FF FF FF FF
FF F8 3F FF FA
12:58:42|Load_Image |-->|B3 38 00 00 21 FF FF FF FF FF FC 00 00 FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FC 3F FF FF FF FF FF FF FE 00 00 FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FE 1F FF 8A
12:58:42|Load_Image |-->|B3 38 00 00 21 FF FF FF FF FF FF 80 08 FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF 1F FF FF FF FF FF FF FF C0 00 FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF 0F FF F2
12:58:42|Load_Image |-->|B3 38 00 00 21 FF FF FF FF FF FF F0 00 FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF 8F FF FF FF FF FF FF FF F8 00 FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF C7 FF EA
12:58:42|Load_Image |-->|B3 38 00 00 21 FF FF FF FF FF FF FE 01 FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF C7 FF FF FF FF FF FF FF FF 87 FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF E3 FF 09
12:58:42|Load_Image |-->|B3 38 00 00 21 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF F7 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF A2
12:58:42|Load_Image |<--|B3 07 00 00 21 01 94
12:58:42|ShowImage  |-->|B3 06 00 00 22 97
12:58:45|ShowImage  |<--|B3 07 00 00 22 01 97
```

Note that the example is by default configured for a 2" display. If another size is used it is necessary to first change the following line in image_data.h

```
#define USE_EPD_Type USE_EPD200
```

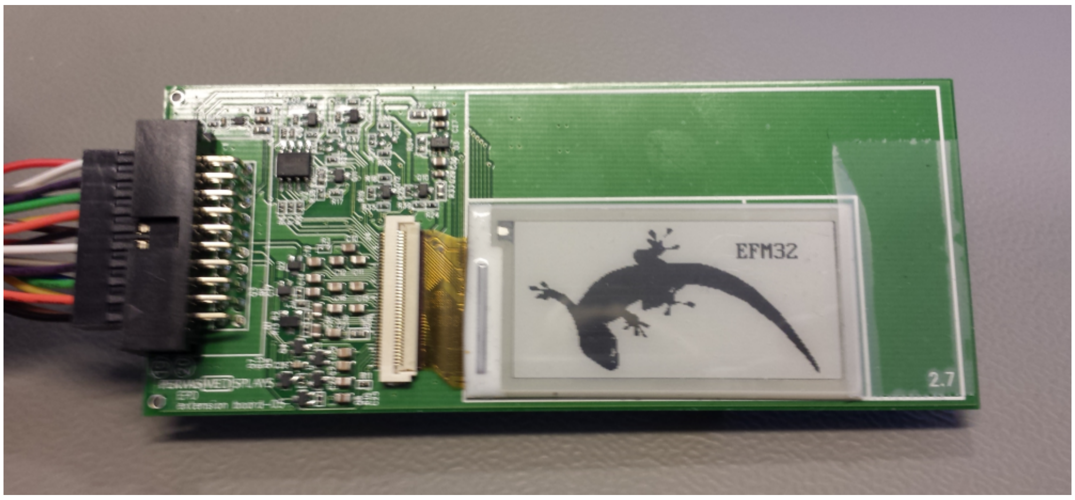
Project files are only available for IAR.

5.2 EPD Demo

This example is created by Silicon Labs based on the Pervasive Displays documentation [pdi_ref]. The example shows how to use emWin to render frames that are put on the EPD. This example only uses the internal RAM and Flash to store images and utilizes the internal temperature sensor in order to adjust the update time.

This example also powers the entire EPD Extension Board from one of the EFM32 GPIO pins. This allows the MCU to completely shut off power to the board between display updates. Note that this requires the VDD pin of the EPD Extension Board to be connected to a GPIO pin of the EFM32 instead of the VMCU pin (different from the PDI demo, see Table 2.1 (p. 5)).

Figure 5.2. EPD Demo



The demo toggles between a sequence of predefined screens. Push button PB0 on the EFM32GG-STK3700 jumps to the next screen and PB1 toggles on/off slideshow mode where the screen changes automatically every few seconds.

Note that by default the example is configured for a 2" display. If another size is used it is necessary to first change the following line in config.h:

```
#define PANEL_200
```

The code has been tested on 2" and 2.7" displays.

6 References

[epd_kit] . *Pervasive Displays EPD Kit Homepage*. http://pervasivedisplays.com/kits/ext_kit.

[epd_ref] . *RePaper Webpage*. http://repaper.org.

[pdi_faq] . *Pervasive Displays FAQ*. http://www.pervasivedisplays.com/technology/about-e-paper.

[cog_timing] . *E-paper Display Cog Driver Interface Timing*. http://www.pervasivedisplays.com/LiteratureRetrieve.aspx?ID=138408.

7 Revision History

7.1 Revision 1.02

2014-05-07

Updated example code to CMSIS 3.20.5

Changed source code license to Silicon Labs license

Added project files for Simplicity IDE

Removed makefiles for Sourcery CodeBench Lite

Changed descriptors.h to Silicon Labs VID/PID

Added Silicon Labs USB CDC driver for Windows

Fixed support for 1.44" displays

7.2 Revision 1.01

2013-10-14

New cover layout

7.3 Revision 1.00

2013-08-12

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

List of Figures

List of Tables

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