Using e-Paper Displays

by Tim Blythman

Electronic paper or e-Paper displays (also known as E-Ink) are used in devices like e-Book readers and even to show product prices on the shelves in some shops. These displays are now becoming available as electronic modules, making them usable by hobbyists. In this article, we explain what they do, how to use them and where to get them.

E-Paper displays have very high contrast and good daylight readability with a wide viewing angle, and usually, require no power to maintain the display once set.

So they are well-suited to applications where display updates are infrequent.

While some e-Paper displays can show colours, most are black and white only, although this limitation also results in good contrast and keeps the control scheme simple.

We bought an e-Paper display, tested it out and wrote code to drive it from both an Arduino and Micromite.

Read on to see if an e-Paper display is something you would like to add to your next project!

How it works

While there are variations to the technology, many displays are based on electrostatically charged coloured particles.

Sometimes these are particles with one black side and one white side; in other cases, they are light particles suspended in a dark liquid.

An applied electric field rotates or moves the particles so that the apparent colour changes. Once the display has been updated, the displayed image will remain indefinitely (or at least until the display is powered up again and commanded to change) – see Fig.1.

The ability to hold the last state with no power consumption makes e-Paper displays ideal for e-Book readers or price displays. The high contrast ratio means that no backlighting is required, and practically zero power is consumed overall.

Thus e-Book readers can run for up to a month between charges, and shelf price displays can operate from a tiny button cell.

Limitations

Of course, if e-Paper displays had no downsides, we'd be seeing them everywhere. They cost more than monochrome LCD with a similar resolution and availability (at least to individuals) is still limited.

Also, as they are optimised for infrequent updates, they don't cope well with fast updates. The unit we tested took around 300ms for a so-called 'partial' refresh and over a second for a full refresh. So they're definitely not suitable for video playback.

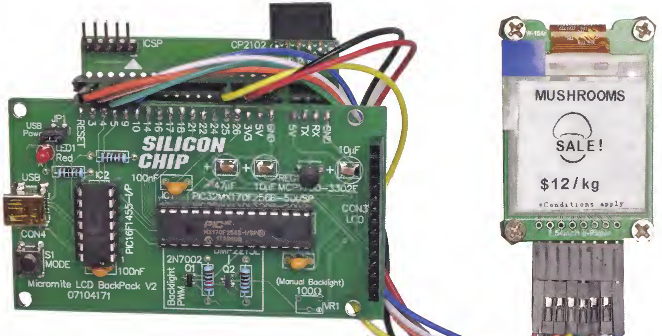
The difference between a partial and full refresh does not relate to whether some or all of the screen is refreshed, but rather how effectively the refresh occurs. A partial refresh is quicker, but may not entirely flip all of the pixels, resulting in 'ghosting' from the previous image.

A full refresh takes longer but is more thorough. If you have ever seen an e-Book reader updating and noticed that the display flashes from all black to all white before settling on a final image, that is a full refresh and it ensures that there are no remnants of the previous display left behind.

Colour e-Paper displays exist but are quite expensive. Interestingly, they use a subtractive colour system based on cyan, magenta and yellow (like printed books and magazines) rather than the additive system used by TVs and computer monitors, which mix red, green and blue light.

Many e-Paper controller ICs use high voltages to drive the display. Since electric field strength is proportional to voltage, it makes sense that a display driven with higher voltages will provide more effective updates.

We measured around 20V on our test



This shows the e-Paper display hooked up to a Micromite BackPack (though it could just as easily be an Arduino, Raspberry Pi or anything else that supports the SPI interface.) This is just one of the demonstration programs that we've written to demonstrate the text and graphics capabilities of the e-Paper. (No, we haven't gone crazy and started selling mushrooms on special at $12/kg – we're not sure how many we'd sell at that price anyway . . .)

module while the display was active. The data sheet includes a reference design which specifies a 25V-rated capacitor and an inductor-based boost circuit.

We found that the 3.3V rail on the Micromite sagged quite badly (down to 2.7V) while the display was updating, and the measured current draw was over 300mA.

Clearly, the low power requirement is subject to the proviso that there may be brief bursts of high current while the display is being updated.

We think a charge-pump boost circuit may be better suited to this application, as the current needed to flip the pixels should be quite small.

Display use with no backlighting assumes that there is adequate ambient light for viewing the display.

For an e-Paper display to be useful in low light conditions, a separate source of illumination would be required, potentially negating the low power benefit (although it still may be more efficient than a backlit display, as less light would be required thanks to the high contrast).

Our e-Paper module

The module we tested is one of the smaller types available, with a 1.54in diagonal display having a square, 200x200 pixel active area. It has an 8-way header for control. The overall module measures 34x50mm and comes with a tapped spacer in each corner for mounting.

This close-up of the display shows that the pixels have quite blurry edges. There are also some small black dots visible on the white region. These are almost impossible to see at normal reading distances.

We sourced our unit from an online store at siliconchip.com.au/link/aapo, but several similar 200x200 pixel displays are available from other sources, and appear to use the same controller and command set.

The 8-way electrical header mentioned above consists of a set of pads spaced apart by 0.1in (2.54mm), to which we soldered a header socket, so we could use jumper wires for prototyping. But you could also plug it into a breadboard or into a socket on stripboard or an etched PCB.

The eight pins are for 3.3V power and ground, plus the SPI control bus (MOSI, SCK and CS) and a data/command (DC) control line, as well as a RESET input and BUSY pin. While most of these are found on other SPI-based display modules (eg, LCDs), the BUSY pin is not something we've seen before.

Fig.2 shows the reference schematic from the display data sheet (siliconchip.com.au/link/aapp). The controller IC is an IL3820, and we found its data sheet, too. See siliconchip.com.au/link/aapq.

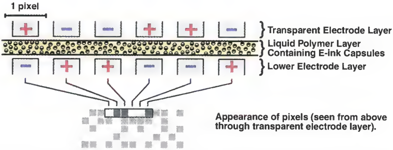


Fig.1: a typical e-Paper display consists of contrasting coloured capsules suspended between the electrodes. An applied electric field causes particles to move or rotate and the displayed colour to change.

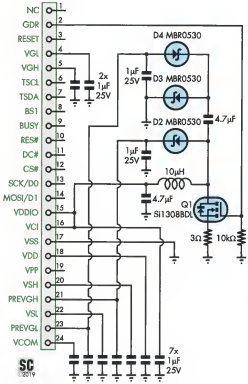


Fig.2: this reference schematic for the IL3820 e-Paper controller IC indicates that the controller doesn't need much external circuitry other than the boost circuit to generate a higher voltage for refreshing the display, and a handful of bypass capacitors.

This controller supports displays up to 320x240 pixels, as well as multiple serial and parallel data formats. Hence the I/O pins take on different roles depending on the data format.

On our module, the BS1 line of this IC is broken out to a small slide switch which can be used to toggle between 9-bit and 8-bit SPI mode. We have used 8-bit mode for our examples, which corresponds to the slide switch being set to the '0' position.

The display data sheet notes that the controller should not be interrupted while the display is being updated. Since this can take over a second, the BUSY pin provides a simple means to monitor when the controller is ready. The microcontroller can resume other tasks and check the BUSY pin to determine when the display controller is ready for another command.

Getting it going

We used an ESP8266-based, Arduino-compatible D1 Mini board for further testing. This is a WiFi-capable board which can be programmed using the Arduino IDE. We're using this because it has 3.3V I/O pins, which suits the I/O and power requirements of the e-Paper module.

It would be tricky to drive it using an Arduino with 5V I/Os like an Uno.

The supplier of the module provided a link to an open-source library for working with the displays. We have included this in our software download bundle. The library supports ESP8266 boards.

As is often the case, using the library was not straightforward. The library supports many different displays, but none of these were an exact match for the display we were using.

The library provides example code for around a dozen displays, including two with the same 200x200 pixel resolution as ours. Trying these, we were able to see some activity on the display, but it appeared to be a corrupted or distorted image.

Looking further into the library, we found that these two displays do not use the IL3820 controller IC. We found another example sketch that did use the IL3820, but it did was intended for a lower-resolution display than ours. It worked, but was not able to refresh the entire screen.

Given these two examples, we were confident we could write our own interface code from scratch and tried to do so. As well as using this library as a reference, we also had the aforementioned data sheet.

Display quirks

The 'quirks' we found are due to the nature of e-Paper displays. These are quite different from liquid crystal displays (LCDs). Like LCDs, the e-Paper displays need to be issued a series of commands at power-up before they are ready to show text or images.

Firstly, the display controller needs to be told how large the display is. While it may seem like a small detail, it's not something we've had to with other display controllers. As we mentioned, the IL3820 controller can work with displays up to 320x240 pixels, while our display is only 200x200 pixels.

We also found reference to a waveform lookup table (LUT) which needed to be loaded into the display. The library code examples actually had two LUT arrays, each 30 bytes long, labelled "full refresh" and "partial refresh".

The LUT waveform controls the display update sequence, so which array you use determines whether you get a full or partial display update.

There is a reference in the IL3820 datasheet as to what voltages these values correspond to, but the values from the library worked well enough that we did not try to change them.

The boost circuit shown in Fig.2 also needs to be activated by sending a command to the controller.

Given the high current consumption that we saw while the boost circuit was running, we tried turning this on immediately before sending the refresh command, and found that this worked well.

Our example code does this too.

Like many other displays, drawing is done by selecting an area of pixels within the display and then streaming bitmap data into that area.

As we've previously alluded, though, merely sending the new pixel data does not cause the display to update.

There is another short sequence of commands which updates the actual display based on the data which is in its memory buffer. It is this sequence which triggers the actual display refresh.

To shut down the boost circuit and save power, after the refresh sequence

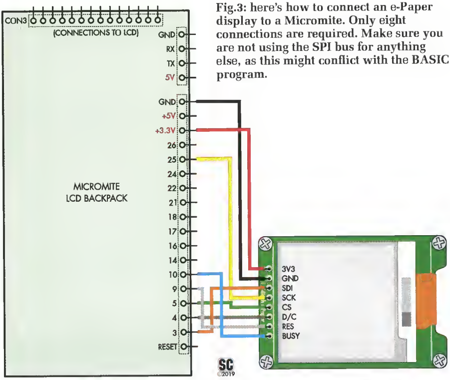


Fig.3: here's how to connect an e-Paper display to a Micromite. Only eight connections are required. Make sure you are not using the SPI bus for anything else, as this might conflict with the BASIC program.

is complete, we shut down the controller by pulling the reset pin low.

We found one more thing that was not obvious from reading the data sheet. There are two RAM buffers on the controller, and it alternates between them each time the display is refreshed.

Thus, it is quite easy to alternate between two images by doing nothing more than sending repeated refresh sequences.

Our code

We're providing two code examples, one for Arduino and one for Micromite. They both drive the display in the same manner.

When you run this code, the display first shows what appears to be various shades of grey, although the mid-shades are actually alternating patterns of light and dark pixels. The display has a nominal resolution of 184 DPI, which is around 7 pixels per millimetre, so dithering works quite well to produce intermediate tones. You have to be very close to the display to see the pixel patterns.

After a short pause, it shows the second display page, which is a comparison between two fonts and also shows the difference between white-on-black and black-on-white text.

We think that the black-on-white text is easier to read, perhaps because of its similarity to black ink printed on white paper which we are so familiar with.

The next page is full of text in a tiny font. Each character is around 1.5mm high, much smaller than the text you might find in a book or newspaper. The text is quite legible, although you may need to squint to read it.

The fourth page has larger text and is quite easy to read. You will have to look closely to see the individual pixels.

The next page is designed to look like what might be displayed on an electronic price ticket. There are different sizes of text and a bitmap image too. We used an online tool to convert images to C code for the Arduino example. It is at: www.digole.com/tools/PicturetoC_Hex_converter.php

For the Micromite example, we had to convert this data to a 32-bit format to simplify the code, which was an extra step, as well as converting it to a format suitable for MMBasic.

The final page display is similar in that it also shows an electronic price ticket, although this example uses the two RAM buffers to flash a banner across the image. As noted above, once the two RAM buffers have been filled, the refresh sequence is all that is needed to alternate between them.

Between each example page, the display is shut down (by pulling the reset pin low), then the code waits for a fixed period before repeating the initialisation code, to restart the display before the next update.

Connecting it up

To try out our example code, you will need a display and also a microcontroller module to connect it to.

We provided a link (above) to the online store where we bought ours. We have not tried any others, but if you find another 200x200 pixel e-Paper display which uses the IL3820 controller and has an eight-way connector, then there's a good chance that our code will work with it.

We have used the hardware SPI ports to drive the displays in both the Micromite and Arduino examples. These, and the other necessary connections, are noted near the top of the sample code. You can also refer to Figs.3 & 4 and the table of connections (Table 1) to wire up the display to your microcontroller.

The module will only work at 3.3V,

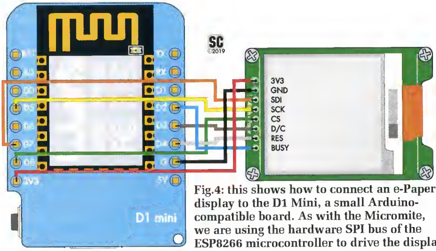


Fig.4: this shows how to connect an e-Paper display to the D1 Mini, a small Arduino-compatible board. As with the Micromite, we are using the hardware SPI bus of the ESP8266 microcontroller to drive the display.

so if using an Arduino board, make sure it's a type with 3.3V I/Os.

Loading the examples

Once you have made the necessary connections, you can try out our code.

Our example code does not need any external libraries to work (although the Arduino example has some included files in the sketch folder for fonts and images).

Open the code and upload it to your microcontroller board. You should see the display cycle through the different test screens described earlier.

Writing your own code

To write your own code, have a look at our examples and follow the sequence between two locations where the reset pin is pulled low.

Note that the module draws a reasonably high current while the boost circuit is running, which is switched on by the EPAPERINIT/epaperInit() function and then off when the reset pin is pulled low.

So we recommend that you run this complete sequence without interruption, minimising the time the boost circuit is active.

e-Paper display	Micromite BackPack V2	Arduino D1 Mini
3V3	3V3	3V3
GND	GND	G
SDI	3	D7
SCK	25	D5
CS	5	D8
D/C	4	D3
RES	9	D4
BUSY	10	D2

Table 1: e-Paper display connections required by example code

The display controller receives rows of eight pixels at a time, so there are only two orientations that can be used (normal and rotated 180°), although this should not cause any problems due to the square shape of the display – there is no 'landscape' or 'portrait' mode!

To see the effects of a full refresh versus a partial refresh, replace all of the

EPAPERSETFULLREFRESH/ epaperSetFullRefresh()

commands with

EPAPERSETPARTIALREFRESH/ epaperSetPartialRefresh()

commands.

What to do with an e-Paper display

We were impressed with how easy it was to get this display up and running, and we hope to find some good ideas as to how this type of display can be used in a practical project.

It is well-suited to the electronic Tide Chart we presented last July (siliconchip.com.au/Article/11142) as this only requires very infrequent display updates.

The e-Paper display would also be good for a weather display or even a web-connected public transport timetable, for similar reasons.

They would work well as programmable name badges, perhaps not even needing a power source while they are being worn.

We're dubious about using them in battery-powered applications as they seem to have very high peak current draw, despite being able to operate with practically zero power draw the rest of the time.

However, once the display is on the e-Paper it stays there until it is re-written, so you don't have to worry about continually supplying power to the module. SC