EPDiy

Release 2.0.0

Valentin Roland

Jan 22, 2024

QUICKSTART:

CHAPTER

ONE

ABOUT

EPDiy is a driver board for e-Paper (or E-ink) displays.

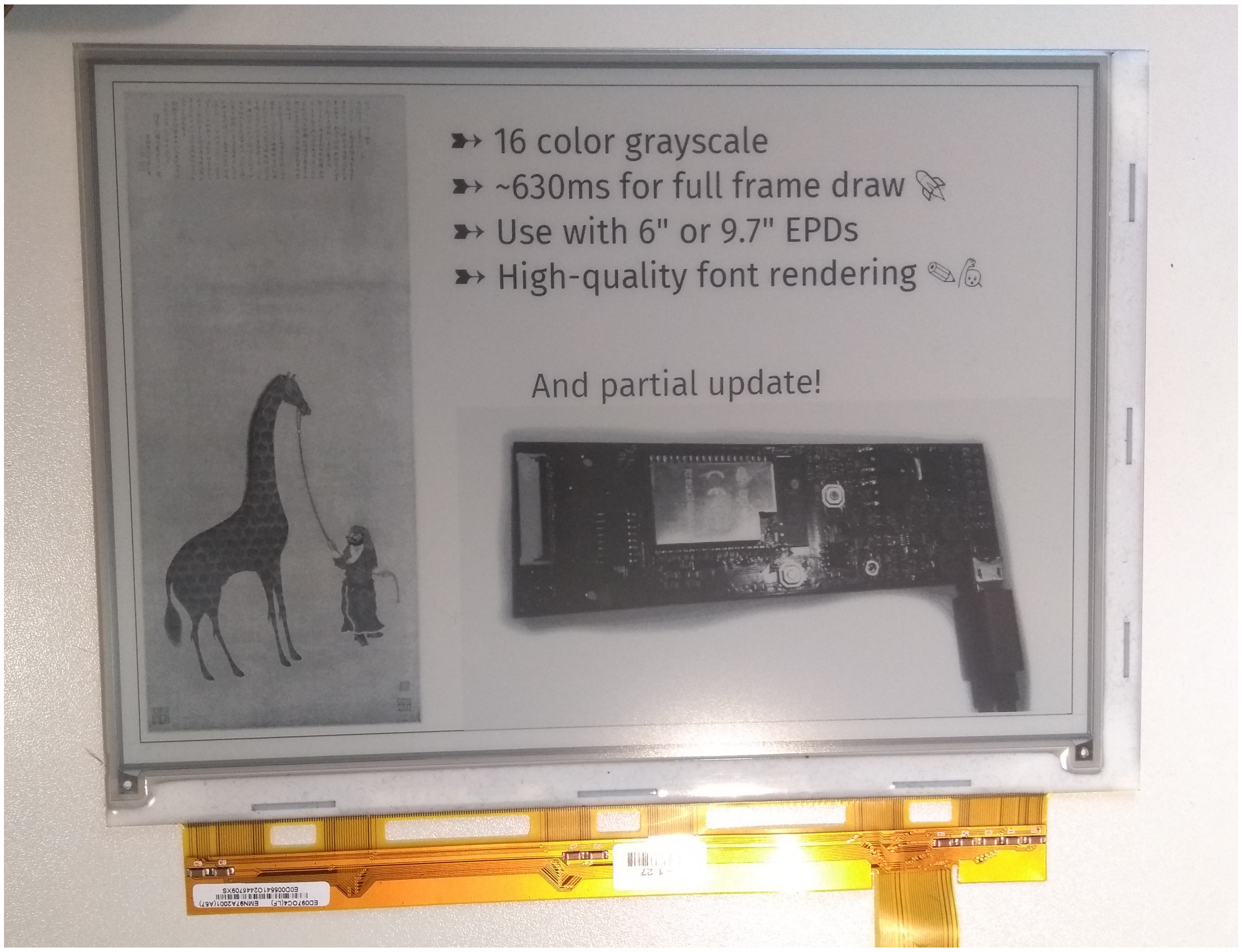
1.1 Display Types

The EPDiy driver board targets multiple E-Paper displays. As the driving method for all matrix-based E-ink displays seems to be more or less the same, only the right connector and timings are needed. A table of supported displays is mainained in the `README.md` file.

Some of the supported displays are showcased below.

1.1.1 ED097OC4

The ED097OC4 was the original target of this project. It is an 9.7 inch screen, with a resolution of 1200 * 825 pixels (150dpi). It is fairly available on Ebay and AliExpress, for around 30$ to 35$. There is also a lower contrast version (ED097OC1) which also works.



1.1.2 ED060SC4

This is a 6 inch display, with a 800 * 600 resolution. With 150dpi as well, it has about half the total display area of the ED097OC4. To connect this display, the 39-pin connector on the back has to be populated. It is also the display a lot of experimentation was done with (see Thanks To), so there are alternative controllers available. Besides the obvious difference in size, this display is cheaper (~20$) and also refreshes slightly faster than the ED097OC4.

1.1.3 ED097TC2

Information on this display should be taken with a grain of salt. One of the displays I ordered as ED097OC4 came as ED097TC2, and upon testing it also exhibited noticably better contrast and a more responsive electro-phoretic medium. The ribbon connector looked like a ED097TC2 as well, or like the *9.7 inch screens offered by Waveshare <https://www.waveshare.com/9.7inch-e-Paper.htm>* (which is sold for a lot more). If you are on the lookout for such a display keep in mind the authenticity of my sample is disputable and resolution and connector type should be double-checked.

16 color grayscale
~630ms for full frame draw
Use with 6" or 9.7" EPDs
High-quality font rendering
And partial update!

CHAPTER

TWO

GETTING STARTED

2.1 Getting your Board

At the current point in time, there is no official way to buy an epdiy board. Fortunately, it is quite easy to order your own. There are many PCB prototype services that will manufacture boards for a low price.

To use one of those services, upload the "Gerber files", usually provided as a zip file, to the service. You can find all the available hardware listed on the Hardware Page.

2.1.1 Choosing and Finding Parts

The parts needed to assemble the epdiy board are listed in the *BOM.csv* file. Commodity parts like resistors, capacitors, coils and diodes are interchangable, as long as they fit the footprint. When in doubt, use the parts listed in the BOM file.

However, some parts are not as common, especially the connectors. Most of them can be found on sites like eBay or AliExpress.

Tips:

- Use the WROVER-B module instead of other WROVER variants. This module exhibits a low deep sleep current and is proven to work.
- The LT1945 voltage booster seems to be out of stock with some distributors, but they are available cheaply from AliExpress.

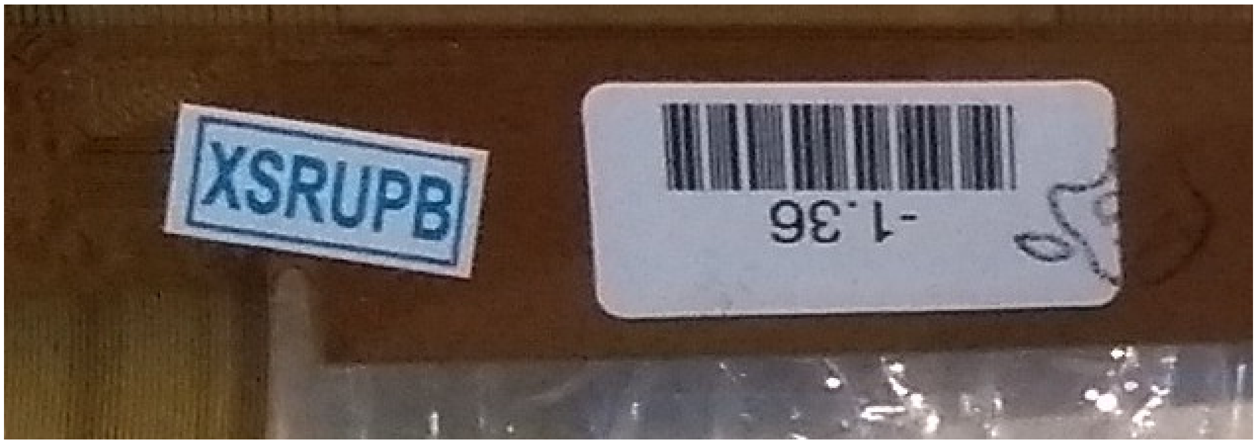
2.2 Calibrate VCOM

Note: Only for old boards

This is only needed with boards prior to revision 6. From revision 6 onwards, VCOM can be set in software via `epd_set_vcom(..)`.

EPaper displays use electrical fields to drive colored particles. One of the required voltages, VCOM (Common Voltage) is display-dependent and must be calibrated for each display individually.

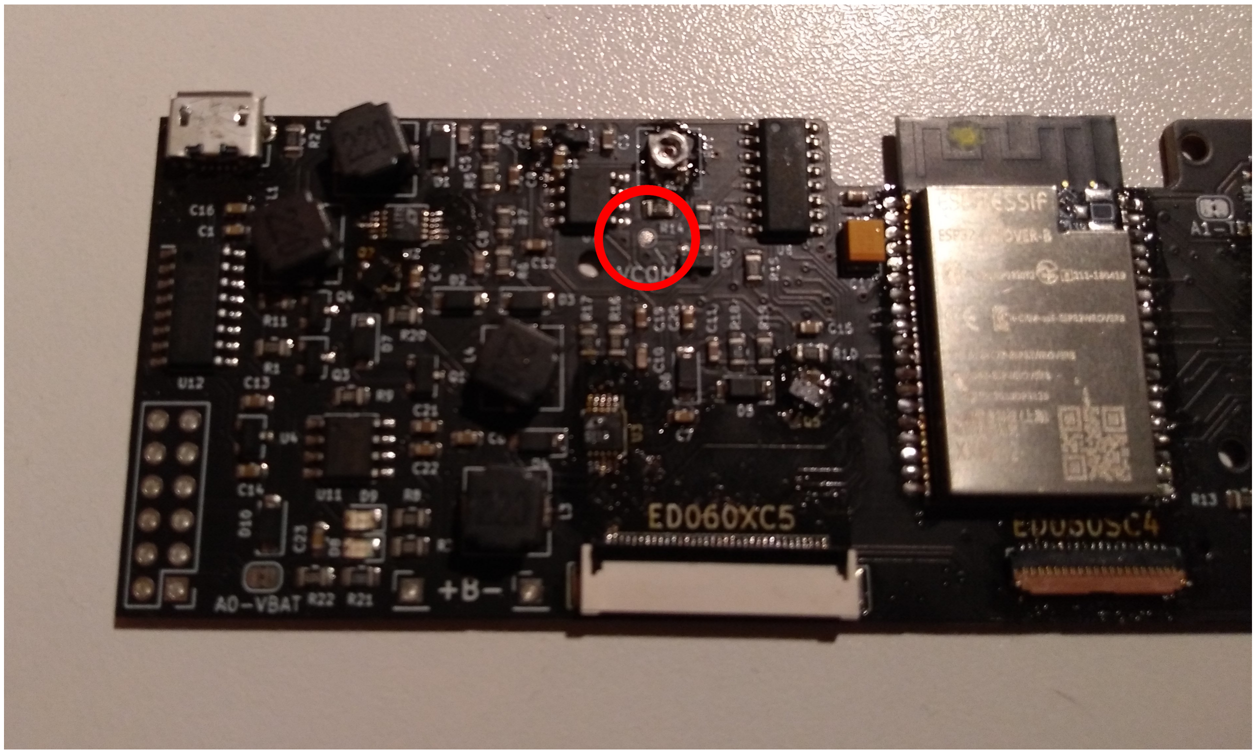
Fortunately, the VCOM voltage is usually printed on the display, similar to this:

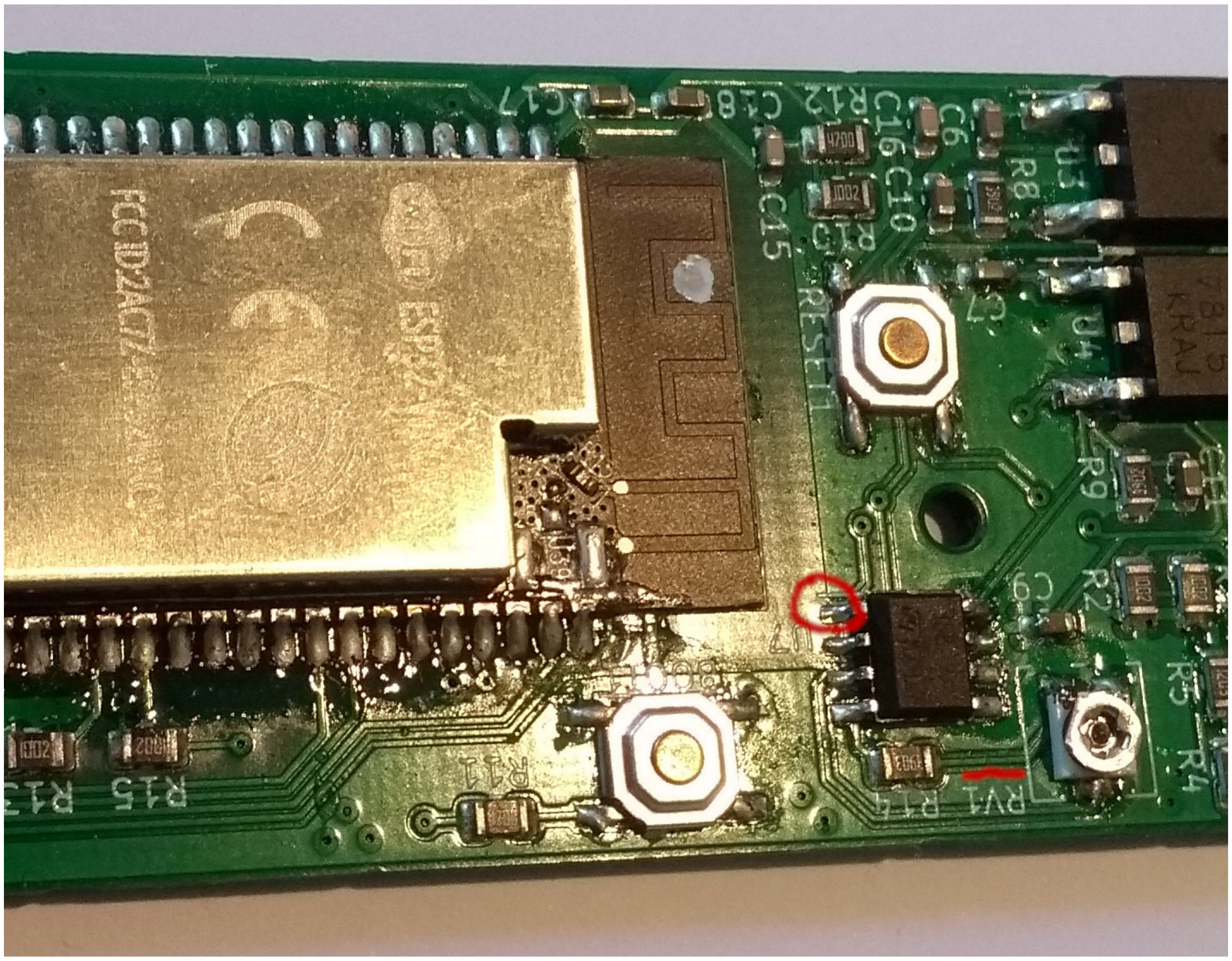


The VCOM value is usually between -1V and -3V.

For the v6 board, you can enter the desired VCOM value in `make menuconfig`. No interaction is required.

For the older models, use the trimmer marked `RV1`. You can measure the VCOM on the VCOM test pad (if your board has one) or directly at the amplifier:





Note: Although most boards do not have it yet, image quality for partial updates can be improved by adding a (at least) 4.7uF capacitor between VCOM and GND. When adding this capacitor, take care with the polarity as VCOM is negative!

2.3 Flashing the demo

First, connect you board with the computer. In the output of `lsusb` you should find something like:

```
Bus 001 Device 048: ID 1a86:7523 QinHeng Electronics HL-340 USB-Serial adapter
```

This means the serial adapter is working and there a serial like `/dev/ttyUSB0` should appear.

Next, make sure you have the Espressif IoT Development Framework installed. The current stable (and tested) version is 4.0. For instructions on how to get started with the IDF, please refer to their documentation.

Then, clone the `epdiy` git repository (and its submodules):

```
git clone --recursive https://github.com/vroland/epdiy
```

Now, (after activating the IDF environment) you should be able to build the demo:

```
cd examples/demo/
idf.py build
```

Hold down the BOOT button on your board, while quickly pressing the RESET button. The ESP module is now in boot mode. Upload the demo program to the board with

```
idf.py flash -b 921600 && idf.py monitor
```

Pressing RESET a second time should start the demo program, which will output some information on the serial monitor.

With the **board power turned off**, connect your display. Power on the board. You should now see the demo output on your display.

2.4 Use with esp-idf

The neccessary functionality for driving an EPD display is encapsulated in the `components/epdiy` IDF component. To use it in you own project, simply copy the `epdiy` folder to your project-local `components` directory. The component sould be automatically detected by the framework, you can now use

```
#include "epdiy.h"
```

to use the EPD driver's *Library API*.

2.4.1 Selecting a Board and Display Type

With epdiy 2.0.0, the display type and board are set via `epd_init()`.

2.4.2 Enable SPI RAM

The ESP32-WROVER-B comes with an additional 8MB external PSRAM, where the `epdiy` is going to store ~2MB for its internal frame buffers. Since it is dynamically allocated from the heap, and the built-in SRAM of ~160KB is insufficient, we need to enable external SPI RAM first.

Open the `menuconfig` again (see above) and navigate to `Component config -> ESP32-Specific -> Support for external, SPI-connected RAM` and enable it.

2.5 Use with Arduino

Epdiy can be used as an Arduino library. Additionally, epdiy comes with board definitions for its supported boards, which must be installed separately. To install epdiy to work with the Arduino IDE (>= 1.8), place the downloaded repository into your Arduino libraries folder.

Alternatively, it is possible to use the Arduino APIs as an IDF component, which allows you to use the Arduino ecosystem (Except for a different build process). This gives you full access to ESP-IDF options.

CHAPTER

THREE

FONTS, IMAGES, WAVEFORMS

The ESP32 is, although fairly capable, still a microcontroller. Thus, with memory and computational resources limited, it is useful to do as much of the processing for displaying fonts and images on a computer.

Epdiy comes with scripts that convert fonts, images and waveforms to C headers, that you can then simply *#include* in your project.

3.1 Generating Font Files

Fonts can only be used by the driver in a special header format (inspired by the Adafruit GFX library), which need to be generated from TTF fonts. For this purpose, the `scripts/fontconvert.py` utility is provided. .. code-block:

```
fontconvert.py [-h] [--compress] [--additional-intervals ADDITIONAL_INTERVALS] name size␣
↪fontstack [fontstack ...]
```

The following example generates a header file for Fira Code at size 10, where glyphs that are not found in Fira Code will be taken from Symbola: .. code-block:

```
./fontconvert.py FiraCode 10 /usr/share/fonts/TTF/FiraCode-Regular.ttf /usr/share/fonts/
↪TTF/Symbola.ttf > ../examples/terminal/main/firacode.h
```

You can change which unicode character codes are to be exported by specifying additional ranges of unicode code points with `--additional-intervals`. Intervals are written as `min,max`. To add multiple intervals, you can specify the `--additional-intervals` option multiple times. .. code-block:

```
./fontconvert.py ... --additional-intervals 0xE0A0,0xE0A2 --additional-intervals 0xE0B0,
↪0xE0B3 ...
```

The above command would add two addtitional ranges.

You can enable compression with `--compress`, which reduces the size of the generated font but comes at a performance cost.

If the generated font files with the default characters are too large for your application, you can modify `intervals` in `fontconvert.py`.

3.2 Generating Images

The process for converting images is very similar to converting fonts. Run the `scripts/imgconvert.py` script with an input image, an image name and an output image. .. code-block:

```
imgconvert.py [-h] -i INPUTFILE -n NAME -o OUTPUTFILE [-maxw MAX_WIDTH] [-maxh MAX_
↪HEIGHT]
```

The image is converted to grayscale scaled down to match fit into `MAX_WIDTH` and `MAX_HEIGHT` (1200x825 by default). For accurate grayscale it is advisable to color-grade and scale the image with a dedicated tool before converting it.

`OUTPUTFILE` will be a C header with the following constants defined:

- `{NAME}_width` is the width of the image
- `{NAME}_height` is the height of the image
- `{NAME}_data` is the image data in 4 bit-per-pixel grayscale format.

3.3 Converting Waveforms

Note: Waveform Timings and V7

Epdiy builtin waveforms currently use variable frame timings to reduce the number of update cycles required. This is currently not implemented in V7. Hence, for best results it is recommended to use Vendor waveforms where available, which use constant frame timings.

In comercial applications, displays are driven with information in so-called *Waveform Files*. These specify how which pulses to apply to the pixel to transition from one gray tone to another. Unfortunately, they are display-specific and proprietary. However, while they are not freely available, they can be obtained through a number of ways:

- Being a large customer of E-Ink. Unfortunately not doable for mere mortals.
- Finding them scattered around the internet. Examples include the MobileRead forums or the NXP Support forum.
- Extracting from e-Reader firmware.
- Extracting from a flash chip that comes with some displays. More on this can be found here.

Waveforms usually come with a `*.wbf` file extension.

If you have a matching waveform file for your display, it can be converted to a waveform header that's usable by epdiy. The advantage of using vendor waveforms include the availability of all implemented modes in the waveform file, support of a wide range of temperatures and more accurate grayscale-to-grayscale transitions.

As a first step, the waveform data is extracted from the original waveform file and stored in JSON format. This can be done using a modified version of inkwave by Marc Juul.

Once a matching JSON file is obtained, the `scripts/waveform_hdrgen.py` utility can be used to generate a waveform header, which can be included in your project.

```
waveform_hdrgen.py [-h] [--list-modes] [--temperature-range TEMPERATURE_RANGE] [--export-
↪modes EXPORT_MODES] name
```

With the `--list-modes` option, a list of all included modes is printed. `name` specifies a name for the generated `EpdWaveform` object. Additionally, the temperature range and modes to export can be limited in order to reduce file size. An example for the usage of this script can be found in the top-level `Makefile` of the epdiy repository.

CHAPTER

FOUR

TIPS & TRICKS

4.1 Temperature Dependence

The display refresh speed depends on the environmental temperature. Thus, if your room temperature is significantly different from ~22°C, grayscale accuracy might be affected when using the builtin waveform. This can be mitigated by using a different timing curve, but this would require calibrating the display timings at that temperature. If you did this for some temperature other than room temperature, please submit a pull request!

4.2 Deep Sleep Current

Board Revision V5 is optimized for running from a battery thanks to its low deep sleep current consumption. In order to achieve the lowest possible deep sleep current, call

```
epd_deinit()
```

before going to deep sleep. This will de-initialize the I2S peripheral used to drive the diplay and bring the pins used by epdiy to a low-power state. You should be able to achieve a deep-sleep current of less than 13µA. If your deep-sleep current is much higher, please check your attached peripherals. With some modules, you have to isolate GPIO 12 before going to deep sleep:

```
rtc_gpio_isolate(GPIO_NUM_12)
```

4.3 Adding a New Display

This section is work-in-progress.

- Add display definitions in `displays.c` and `epd_display.h`.
- Include waveform in `bulitin_waveforms.c`
- Calibrate timing curve in `scripts/generate_epdiy_waveforms.py`.
- Add to the list of displays to build waveforms for in `Makefile`
- Document

CHAPTER

FIVE

LIBRARY API

5.1 Highlevel API

High-level API for drawing to e-paper displays.

The functions in this library provide a simple way to manage updates of e-paper display. To use it, follow the steps below:

1. First, we declare a global object that manages the display state.

```
EpdiyHighlevelState hl;
```

2. Then, the driver and framebuffers must be initialized.

```
epd_init(EPD_LUT_1K);
hl = epd_hl_init(EPD_BUILTIN_WAVEFORM);
```

3. Now, we can draw to the allocated framebuffer, using the draw and text functions defined in `epdiy.h`. This will not yet update the display, but only its representation in memory.

```
// A reference to the framebuffer
uint8_t* fb = epd_hl_get_framebuffer(&hl);

// draw a black rectangle
EpdRect some_rect = {
    .x = 100,
    .y = 100,
    .width = 100,
    .height = 100
};
epd_fill_rect(some_rect, 0x0, fb);

// write a message
int cursor_x = 100;
int cursor_y = 300;
epd_write_default(&FiraSans, "Hello, World!", &cursor_x, &cursor_y, fb);

// finally, update the display!
int temperature = 25;
epd_poweron();
```

(continues on next page)

(continued from previous page)

```
EpdDrawError err = epd_hl_update_screen(&hl, MODE_GC16, temperature);
epd_poweroff();
```

That's it! For many application, this will be enough. For special applications and requirements, have a closer look at the `epdiy.h` header.

5.1.1 Colors

Since most displays only support 16 colors, we're only using the upper 4 bits (nibble) of a byte to detect the color.

```
char pixel_color = color & 0xF0;
```

So keep in mind, when passing a color to any function, to always set the upper 4 bits, otherwise the color would be black.

Possible colors are `0xF0` (white) through `0x80` (median gray) til `0x00` (black).

Defines

`EPD_BUILTIN_WAVEFORM`

Functions

EpdiyHighlevelState **`epd_hl_init`**(const *EpdWaveform* *waveform)

Initialize a state object. This allocates two framebuffers and an update buffer for the display in the external PSRAM. In order to keep things simple, a chip reset is triggered if this fails.

Parameters

- **`waveform`** – The waveform to use for updates. If you did not create your own, this will be `EPD_BUILTIN_WAVEFORM`.

Returns

An initialized state object.

uint8_t ***`epd_hl_get_framebuffer`**(*EpdiyHighlevelState* *state)

Get a reference to the front framebuffer. Use this to draw on the framebuffer before updating the screen with *`epd_hl_update_screen()`*.

enum *EpdDrawError* **`epd_hl_update_screen`**(*EpdiyHighlevelState* *state, enum *EpdDrawMode* mode, int temperature)

Update the EPD screen to match the content of the front frame buffer. Prior to this, power to the display must be enabled via *`epd_poweron()`* and should be disabled afterwards if no immediate additional updates follow.

Parameters

- **`state`** – A reference to the *`EpdiyHighlevelState`* object used.
- **`mode`** – The update mode to use. Additional mode settings like the framebuffer format or previous display state are determined by the driver and must not be supplied here. In most cases, one of `MODE_GC16` and `MODE_GL16` should be used.
- **`temperature`** – Environmental temperature of the display in °C.

Returns
`EPD_DRAW_SUCCESS` on sucess, a combination of error flags otherwise.

enum *EpdDrawError* **`epd_hl_update_area`**(*EpdiyHighlevelState* *state, enum *EpdDrawMode* mode, int temperature, *EpdRect* area)

Update an area of the screen to match the content of the front framebuffer. Supplying a small area to update can speed up the update process. Prior to this, power to the display must be enabled via `epd_poweron()` and should be disabled afterwards if no immediate additional updates follow.

Parameters

- **`state`** – A reference to the `EpdiyHighlevelState` object used.
- **`mode`** – See `epd_hl_update_screen()`.
- **`temperature`** – Environmental temperature of the display in °C.
- **`area`** – Area of the screen to update.

Returns
`EPD_DRAW_SUCCESS` on sucess, a combination of error flags otherwise.

void **`epd_hl_set_all_white`**(*EpdiyHighlevelState* *state)

Reset the front framebuffer to a white state.

Parameters

- **`state`** – A reference to the `EpdiyHighlevelState` object used.

void **`epd_fullclear`**(*EpdiyHighlevelState* *state, int temperature)

Bring the display to a fully white state and get rid of any remaining artifacts.

struct **`EpdiyHighlevelState`**

#include <epd_highlevel.h> Holds the internal state of the high-level API.

Public Members

uint8_t ***`front_fb`**

The “front” framebuffer object.

uint8_t ***`back_fb`**

The “back” framebuffer object.

uint8_t ***`difference_fb`**

Buffer for holding the interlaced difference image.

bool ***`dirty_lines`**

Tainted lines based on the last difference calculation.

const *EpdWaveform* ***`waveform`**

The waveform information to use.

5.2 Complete API

A driver library for drawing to an EPD.

Defines

`EPD_MODE_DEFAULT`

The default draw mode (non-flashy refresh, whith previously white screen).

Enums

enum **`EpdInitOptions`**

Global EPD driver options.

Values:

enumerator **`EPD_OPTIONS_DEFAULT`**

Use the default options.

enumerator **`EPD_LUT_1K`**

Use a small look-up table of 1024 bytes. The EPD driver will use less space, but performance may be worse.

enumerator **`EPD_LUT_64K`**

Use a 64K lookup table. (default) Best performance, but permanently occupies a 64k block of internal memory.

enumerator **`EPD_FEED_QUEUE_8`**

Use a small feed queue of 8 display lines. This uses less memory, but may impact performance.

enumerator **`EPD_FEED_QUEUE_32`**

Use a feed queue of 32 display lines. (default) Best performance, but larger memory footprint.

enum **`EpdDrawMode`**

The image drawing mode.

Values:

enumerator **`MODE_INIT`**

An init waveform. This is currently unused, use *`epd_clear()`* instead.

enumerator **`MODE_DU`**

Direct Update: Go from any color to black for white only.

enumerator **`MODE_GC16`**

Go from any grayscale value to another with a flashing update.

enumerator **`MODE_GC16_FAST`**

Faster version of `MODE_GC16`. Not available with default epdiy waveforms.

enumerator **`MODE_A2`**

Animation Mode: Fast, monochrom updates. Not available with default epdiy waveforms.

enumerator **`MODE_GL16`**

Go from any grayscale value to another with a non-flashing update.

enumerator **`MODE_GL16_FAST`**

Faster version of `MODE_GL16`. Not available with default epdiy waveforms.

enumerator **`MODE_DU4`**

A 4-grayscale version of `MODE_DU`. Not available with default epdiy waveforms.

enumerator **`MODE_GL4`**

Arbitrary transitions for 4 grayscale values. Not available with default epdiy waveforms.

enumerator **`MODE_GL16_INV`**

Not available with default epdiy waveforms.

enumerator **`MODE_EPDIY_WHITE_TO_GL16`**

Go from a white screen to arbitrary grayscale, quickly. Exclusively available with epdiy waveforms.

enumerator **`MODE_EPDIY_BLACK_TO_GL16`**

Go from a black screen to arbitrary grayscale, quickly. Exclusively available with epdiy waveforms.

enumerator **`MODE_EPDIY_MONOCHROME`**

Monochrome mode. Only supported with 1bpp buffers.

enumerator **`MODE_UNKNOWN_WAVEFORM`**

enumerator **`MODE_PACKING_8PPB`**

1 bit-per-pixel framebuffer with 0 = black, 1 = white. MSB is left is the leftmost pixel, LSB the rightmost pixel.

enumerator **`MODE_PACKING_2PPB`**

4 bit-per pixel framebuffer with 0x0 = black, 0xF = white. The upper nibble corresponds to the left pixel. A byte cannot wrap over multiple rows, images of uneven width must add a padding nibble per line.

enumerator **`MODE_PACKING_1PPB_DIFFERENCE`**

A difference image with one pixel per byte. The upper nibble marks the “from” color, the lower nibble the “to” color.

enumerator **PREVIOUSLY_WHITE**

Assert that the display has a uniform color, e.g. after initialization. If `MODE_PACKING_2PPB` is specified, a optimized output calculation can be used. Draw on a white background

enumerator **PREVIOUSLY_BLACK**

See `PREVIOUSLY_WHITE`. Draw on a black background

enum **EpdRotation**

Display software rotation. Sets the rotation for the purposes of the drawing and font functions Use epd_set_rotation(EPD_ROT_*) to set it using one of the options below Use *epd_get_rotation()* in case you need to read this value

Values:

enumerator **EPD_ROT_LANDSCAPE**

enumerator **EPD_ROT_PORTRAIT**

enumerator **EPD_ROT_INVERTED_LANDSCAPE**

enumerator **EPD_ROT_INVERTED_PORTRAIT**

enum **EpdDrawError**

Possible failures when drawing.

Values:

enumerator **EPD_DRAW_SUCCESS**

enumerator **EPD_DRAW_INVALID_PACKING_MODE**

No valid framebuffer packing mode was specified.

enumerator **EPD_DRAW_LOOKUP_NOT_IMPLEMENTED**

No lookup table implementation for this mode / packing.

enumerator **EPD_DRAW_STRING_INVALID**

The string to draw is invalid.

enumerator **EPD_DRAW_NO_DRAWABLE_CHARACTERS**

The string was not empty, but no characters where drawable.

enumerator **EPD_DRAW_FAILED_ALLOC**

Allocation failed.

enumerator **EPD_DRAW_GLYPH_FALLBACK_FAILED**

A glyph could not be drawn, and not fallback was present.

enumerator **EPD_DRAW_INVALID_CROP**

The specified crop area is invalid.

enumerator **EPD_DRAW_MODE_NOT_FOUND**

No such mode is available with the current waveform.

enumerator **EPD_DRAW_NO_PHASES_AVAILABLE**

The waveform info file contains no applicable temperature range.

enumerator **EPD_DRAW_INVALID_FONT_FLAGS**

An invalid combination of font flags was used.

enumerator **EPD_DRAW_EMPTY_LINE_QUEUE**

The waveform lookup could not keep up with the display output.

Reduce the display clock speed.

enum **EpdFontFlags**

Font drawing flags.

Values:

enumerator **EPD_DRAW_BACKGROUND**

Draw a background.

Take the background into account when calculating the size.

enumerator **EPD_DRAW_ALIGN_LEFT**

Left-Align lines.

enumerator **EPD_DRAW_ALIGN_RIGHT**

Right-align lines.

enumerator **EPD_DRAW_ALIGN_CENTER**

Center-align lines.

Functions

void **epd_init**(const EpdBoardDefinition *board, const EpdDisplay_t *display, enum *EpdInitOptions* options)

Initialize the ePaper display

const EpdDisplay_t ***epd_get_display**()

Get the configured display.

int **epd_width**()

Get the EPD display's witdth.

int **epd_height**()

Get the EPD display's height.

void **epd_set_vcom**(uint16_t vcom)

Set the display common voltage if supported.

Voltage is set as absolute value in millivolts. Although VCOM is negative, this function takes a positive (absolute) value.

float **epd_ambient_temperature**()

Get the current ambient temperature in °C, if the board has a sensor.

Get the current ambient temperature in °C, if supported by the board. Requires the display to be powered on.

enum *EpdRotation* **epd_get_rotation**()

Get the display rotation value

void **epd_set_rotation**(enum *EpdRotation* rotation)

Set the display rotation: Affects the drawing and font functions

int **epd_rotated_display_width**()

Get screen width after rotation

int **epd_rotated_display_height**()

Get screen height after rotation

void **epd_deinit**()

Deinit the ePaper display

void **epd_poweron**()

Enable display power supply.

void **epd_poweroff**()

Disable display power supply.

void **epd_clear**()

Clear the whole screen by flashing it.

void **epd_clear_area**(*EpdRect* area)

Clear an area by flashing it.

Parameters

- **area** – The area to clear.

void **epd_clear_area_cycles**(*EpdRect* area, int cycles, int cycle_time)

Clear an area by flashing it.

Parameters

- **area** – The area to clear.
- **cycles** – The number of black-to-white clear cycles.
- **cycle_time** – Length of a cycle. Default: 50 (us).

EpdRect **epd_full_screen**()

Returns

Rectancle representing the whole screen area.

void **epd_copy_to_framebuffer**(*EpdRect* image_area, const uint8_t *image_data, uint8_t *framebuffer)

Draw a picture to a given framebuffer.

Parameters

- **`image_area`** – The area to copy to. `width` and `height` of the area must correspond to the image dimensions in pixels.
- **`image_data`** – The image data, as a buffer of 4 bit wide brightness values. Pixel data is packed (two pixels per byte). A byte cannot wrap over multiple rows, images of uneven width must add a padding nibble per line.
- **`framebuffer`** – The framebuffer object, which must be *`epd_width()`* `/ 2 *` *`epd_height()`* large.

void **`epd_draw_pixel`**(int x, int y, uint8_t color, uint8_t *framebuffer)

Draw a pixel a given framebuffer.

Parameters

- **`x`** – Horizontal position in pixels.
- **`y`** – Vertical position in pixels.
- **`color`** – The gray value of the line (see *Colors*);
- **`framebuffer`** – The framebuffer to draw to,

void **`epd_draw_hline`**(int x, int y, int length, uint8_t color, uint8_t *framebuffer)

Draw a horizontal line to a given framebuffer.

Parameters

- **`x`** – Horizontal start position in pixels.
- **`y`** – Vertical start position in pixels.
- **`length`** – Length of the line in pixels.
- **`color`** – The gray value of the line (see *Colors*);
- **`framebuffer`** – The framebuffer to draw to, which must be *`epd_width()`* `/ 2 *` *`epd_height()`* bytes large.

void **`epd_draw_vline`**(int x, int y, int length, uint8_t color, uint8_t *framebuffer)

Draw a horizontal line to a given framebuffer.

Parameters

- **`x`** – Horizontal start position in pixels.
- **`y`** – Vertical start position in pixels.
- **`length`** – Length of the line in pixels.
- **`color`** – The gray value of the line (see *Colors*);
- **`framebuffer`** – The framebuffer to draw to, which must be *`epd_width()`* `/ 2 *` *`epd_height()`* bytes large.

void **`epd_fill_circle_helper`**(int x0, int y0, int r, int corners, int delta, uint8_t color, uint8_t *framebuffer)

void **`epd_draw_circle`**(int x, int y, int r, uint8_t color, uint8_t *framebuffer)

Draw a circle with given center and radius

Parameters

- **`x`** – Center-point x coordinate
- **`y`** – Center-point y coordinate
- **`r`** – Radius of the circle in pixels

- **color** – The gray value of the line (see *Colors*);
- **framebuffer** – The framebuffer to draw to,

void **epd_fill_circle**(int x, int y, int r, uint8_t color, uint8_t *framebuffer)

Draw a circle with fill with given center and radius

Parameters

- **x** – Center-point x coordinate
- **y** – Center-point y coordinate
- **r** – Radius of the circle in pixels
- **color** – The gray value of the line (see *Colors*);
- **framebuffer** – The framebuffer to draw to,

void **epd_draw_rect**(*EpdRect* rect, uint8_t color, uint8_t *framebuffer)

Draw a rectanle with no fill color

Parameters

- **rect** – The rectangle to draw.
- **color** – The gray value of the line (see *Colors*);
- **framebuffer** – The framebuffer to draw to,

void **epd_fill_rect**(*EpdRect* rect, uint8_t color, uint8_t *framebuffer)

Draw a rectanle with fill color

Parameters

- **rect** – The rectangle to fill.
- **color** – The gray value of the line (see *Colors*);
- **framebuffer** – The framebuffer to draw to,

void **epd_draw_line**(int x0, int y0, int x1, int y1, uint8_t color, uint8_t *framebuffer)

Draw a line

Parameters

- **x0** – Start point x coordinate
- **y0** – Start point y coordinate
- **x1** – End point x coordinate
- **y1** – End point y coordinate
- **color** – The gray value of the line (see *Colors*);
- **framebuffer** – The framebuffer to draw to,

void **epd_draw_triangle**(int x0, int y0, int x1, int y1, int x2, int y2, uint8_t color, uint8_t *framebuffer)

Draw a triangle with no fill color

Parameters

- **x0** – Vertex #0 x coordinate
- **y0** – Vertex #0 y coordinate
- **x1** – Vertex #1 x coordinate

- **y1** – Vertex #1 y coordinate
- **x2** – Vertex #2 x coordinate
- **y2** – Vertex #2 y coordinate
- **color** – The gray value of the line (see *Colors*);
- **framebuffer** – The framebuffer to draw to,

void **epd_fill_triangle**(int x0, int y0, int x1, int y1, int x2, int y2, uint8_t color, uint8_t *framebuffer)

Draw a triangle with color-fill

Parameters

- **x0** – Vertex #0 x coordinate
- **y0** – Vertex #0 y coordinate
- **x1** – Vertex #1 x coordinate
- **y1** – Vertex #1 y coordinate
- **x2** – Vertex #2 x coordinate
- **y2** – Vertex #2 y coordinate
- **color** – The gray value of the line (see *Colors*);
- **framebuffer** – The framebuffer to draw to,

EpdFontProperties **epd_font_properties_default**()

The default font properties.

void **epd_get_text_bounds**(const *EpdFont* *font, const char *string, const int *x, const int *y, int *x1, int *y1, int *w, int *h, const *EpdFontProperties* *props)

Get the text bounds for string, when drawn at (x, y). Set font properties to NULL to use the defaults.

EpdRect **epd_get_string_rect**(const *EpdFont* *font, const char *string, int x, int y, int margin, const *EpdFontProperties* *properties)

Returns a rect with the bounds of the text

Parameters

- **font** – : the font used to get the character sizes
- **string** – pointer to c string
- **x** – : left most position of rectangle
- **y** – : top most point of the rectangle
- **margin** – : to be pllied to the width and height

Returns

EpdRect with x and y as per the original and height and width adjusted to fit the text with the margin added as well.

enum *EpdDrawError* **epd_write_string**(const *EpdFont* *font, const char *string, int *cursor_x, int *cursor_y, uint8_t *framebuffer, const *EpdFontProperties* *properties)

Write text to the EPD.

enum *EpdDrawError* **epd_write_default**(const *EpdFont* *font, const char *string, int *cursor_x, int *cursor_y, uint8_t *framebuffer)

Write a (multi-line) string to the EPD.

const *EpdGlyph* ***epd_get_glyph**(const *EpdFont* *font, uint32_t code_point)

Get the font glyph for a unicode code point.

void **epd_push_pixels**(*EpdRect* area, short time, int color)

Darken / lighten an area for a given time.

Parameters

- **area** – The area to darken / lighten.
- **time** – The time in us to apply voltage to each pixel.
- **color** – 1: lighten, 0: darken.

enum *EpdDrawError* **epd_draw_base**(*EpdRect* area, const uint8_t *data, *EpdRect* crop_to, enum *EpdDrawMode* mode, int temperature, const bool *drawn_lines, const *EpdWaveform* *waveform)

Base function for drawing an image on the screen. If It is very customizable, and the documentation below should be studied carefully. For simple applications, use the epdiy highlevel api in "epd_higlevel.h".

Parameters

- **area** – The area of the screen to draw to. This can be imagined as shifting the origin of the frame buffer.
- **data** – A full framebuffer of display data. It's structure depends on the chosen `mode`.
- **crop_to** – Only draw a part of the frame buffer. Set to *`epd_full_screen()`* to draw the full buffer.
- **mode** – Specifies the Waveform used, the framebuffer format and additional information, like if the display is cleared.
- **temperature** – The temperature of the display in °C. Currently, this is unused by the default waveforms at can be set to room temperature, e.g. 20-25°C.
- **drawn_lines** – If not NULL, an array of at least the height of the image. Every line where the corresponding value in `lines` is `false` will be skipped.
- **waveform** – The waveform information to use for drawing. If you don't have special waveforms, use `EPD_BUILTIN_WAVEFORM`.

Returns

`EPD_DRAW_SUCCESS` on sucess, a combination of error flags otherwise.

EpdRect **epd_difference_image_cropped**(const uint8_t *to, const uint8_t *from, *EpdRect* crop_to, uint8_t *interlaced, bool *dirty_lines, bool *previously_white, bool *previously_black)

Calculate a `MODE_PACKING_1PPB_DIFFERENCE` difference image from two `MODE_PACKING_2PPB` (4 bit-per-pixel) buffers. If you're using the epdiy highlevel api, this is handled by the update functions.

Parameters

- **to** – The goal image as 4-bpp (`MODE_PACKING_2PPB`) framebuffer.
- **from** – The previous image as 4-bpp (`MODE_PACKING_2PPB`) framebuffer.
- **crop_to** – Only calculate the difference for a crop of the input framebuffers. The `interlaced` will not be modified outside the crop area.
- **interlaced** – The resulting difference image in `MODE_PACKING_1PPB_DIFFERENCE` format.

- **dirty_lines** – An array of at least *epd_height()*. The positions corresponding to lines where to and from differ are set to true, otherwise to false.
- **previously_white** – If not NULL, it is set to true if the considered crop of the from-image is completely white.
- **previously_black** – If not NULL, it is set to true if the considered crop of the from-image is completely black.

Returns

The smallest rectangle containing all changed pixels.

EpdRect **epd_difference_image**(const uint8_t *to, const uint8_t *from, uint8_t *interlaced, bool *dirty_lines)

Simplified version of *epd_difference_image_cropped()*, which considers the whole display frame buffer.

See *epd_difference_image_cropped()* for details.

uint8_t **epd_get_pixel**(int x, int y, int fb_width, int fb_height, const uint8_t *framebuffer)

Return the pixel color of a 4 bit image array x,y coordinates of the image pixel fb_width, fb_height dimensions

Returns

uint8_t 0-255 representing the color on given coordinates (as in epd_draw_pixel)

void **epd_draw_rotated_image**(*EpdRect* image_area, const uint8_t *image_buffer, uint8_t *framebuffer)

Draw an image reading pixel per pixel and being rotation aware (via epd_draw_pixel)

void **epd_draw_rotated_transparent_image**(*EpdRect* image_area, const uint8_t *image_buffer, uint8_t *framebuffer, uint8_t transparent_color)

Draw an image reading pixel per pixel and being rotation aware (via epd_draw_pixel) With an optional transparent color (color key transparency)

void **epd_set_lcd_pixel_clock_MHz**(int frequency)

Override the pixel clock when using the LCD driver for display output (Epdiy V7+). This may result in draws failing if it's set too high!

This method can be used to tune your application for maximum refresh speed, if you can guarantee the driver can keep up.

struct **EpdRect**

#include <epdiy.h> An area on the display.

Public Members

int **x**

Horizontal position.

int **y**

Vertical position.

int **width**

Area / image width, must be positive.

int **height**

Area / image height, must be positive.

struct **EpdFontProperties**

#include <epdiy.h> Font properties.

Public Members

uint8_t **fg_color**

Foreground color.

uint8_t **bg_color**

Background color.

uint32_t **fallback_glyph**

Use the glyph for this codepoint for missing glyphs.

enum *EpdFontFlags* **flags**

Additional flags, reserved for future use.

5.3 Internals

Internal definitions and auxiliary data types.

Unless you want to extend the library itself (Which you are very welcome to do), you will most likely not need to know about this file.

Defines

MINIMUM_FRAME_TIME

minimal draw time in ms for a frame layer, which will allow all particles to set properly.

MONOCHROME_FRAME_TIME

Frame draw time for monochrome mode in 1/10 us.

Variables

const *EpdWaveform* **epdiy_ED060SC4**

const *EpdWaveform* **epdiy_ED097OC4**

const *EpdWaveform* **epdiy_ED047TC1**

const *EpdWaveform* **epdiy_ED047TC2**

const *EpdWaveform* **`epdiy_ED097TC2`**

const *EpdWaveform* **`epdiy_ED060XC3`**

const *EpdWaveform* **`epdiy_ED060SCT`**

const *EpdWaveform* **`epdiy_ED133UT2`**

struct **`EpdWaveformPhases`**

Public Members

int **`phases`**

const uint8_t ***`luts`**

const int ***`phase_times`**

If we have timing information for the individual phases, this is an array of the on-times for each phase. Otherwise, this is NULL.

struct **`EpdWaveformMode`**

Public Members

uint8_t **`type`**

uint8_t **`temp_ranges`**

EpdWaveformPhases const ****`range_data`**

struct **`EpdWaveformTempInterval`**

Public Members

int **`min`**

int **`max`**

struct **`EpdWaveform`**

Public Members

uint8_t **num_modes**

uint8_t **num_temp_ranges**

EpdWaveformMode const ****mode_data**

EpdWaveformTempInterval const ***temp_intervals**

struct **EpdGlyph**

#include <epd_internals.h> Font data stored PER GLYPH.

Public Members

uint16_t **width**

Bitmap dimensions in pixels.

uint16_t **height**

Bitmap dimensions in pixels.

uint16_t **advance_x**

Distance to advance cursor (x axis)

int16_t **left**

X dist from cursor pos to UL corner.

int16_t **top**

Y dist from cursor pos to UL corner.

uint32_t **compressed_size**

Size of the zlib-compressed font data.

uint32_t **data_offset**

Pointer into EpdFont->bitmap.

struct **EpdUnicodeInterval**

#include <epd_internals.h> Glyph interval structure.

Public Members

uint32_t **`first`**

The first unicode code point of the interval.

uint32_t **`last`**

The last unicode code point of the interval.

uint32_t **`offset`**

Index of the first code point into the glyph array.

struct **`EpdFont`**

#include <epd_internals.h> Data stored for FONT AS A WHOLE.

Public Members

const uint8_t ***`bitmap`**

Glyph bitmaps, concatenated.

const *EpdGlyph* ***`glyph`**

Glyph array.

const *EpdUnicodeInterval* ***`intervals`**

Valid unicode intervals for this font.

uint32_t **`interval_count`**

Number of unicode intervals.

bool **`compressed`**

Does this font use compressed glyph bitmaps?

uint16_t **`advance_y`**

Newline distance (y axis)

int **`ascender`**

Maximal height of a glyph above the base line.

int **`descender`**

Maximal height of a glyph below the base line.

5.4 Board-Specific Extensions

Board-specific functions that are only conditionally defined.

Functions

void **epd_powerdown_lilygo_t5_47**()

This is a Lilygo47 specific function

This is a work around a hardware issue with the Lilygo47 *epd_poweroff()* turns off the epaper completely however the hardware of the Lilygo47 is different than the official boards. Which means that on the Lilygo47 this disables power to the touchscreen.

This is a workaround to allow to disable display power but not the touch screen. On the Lilygo the epd power flag was re-purposed as power enable for everything. This is a hardware thing.

Please use *epd_poweroff()* and *epd_deinit()* whenever you sleep the system. The following code can be used to sleep the lilygo and power down the peripherals and wake the unit on touch. However is should be noted that the touch controller is not powered and as such the touch coordinates will not be captured. Arduino specific code:

```
epd_poweroff();
epd_deinit();
esp_sleep_enable_ext1_wakeup(GPIO_SEL_13, ESP_EXT1_WAKEUP_ANY_HIGH);
esp_deep_sleep_start();
```

Warning: This workaround may still leave power on to epd and as such may cause other problems such as grey screen.

void **epd_powerdown**() __attribute__((deprecated))

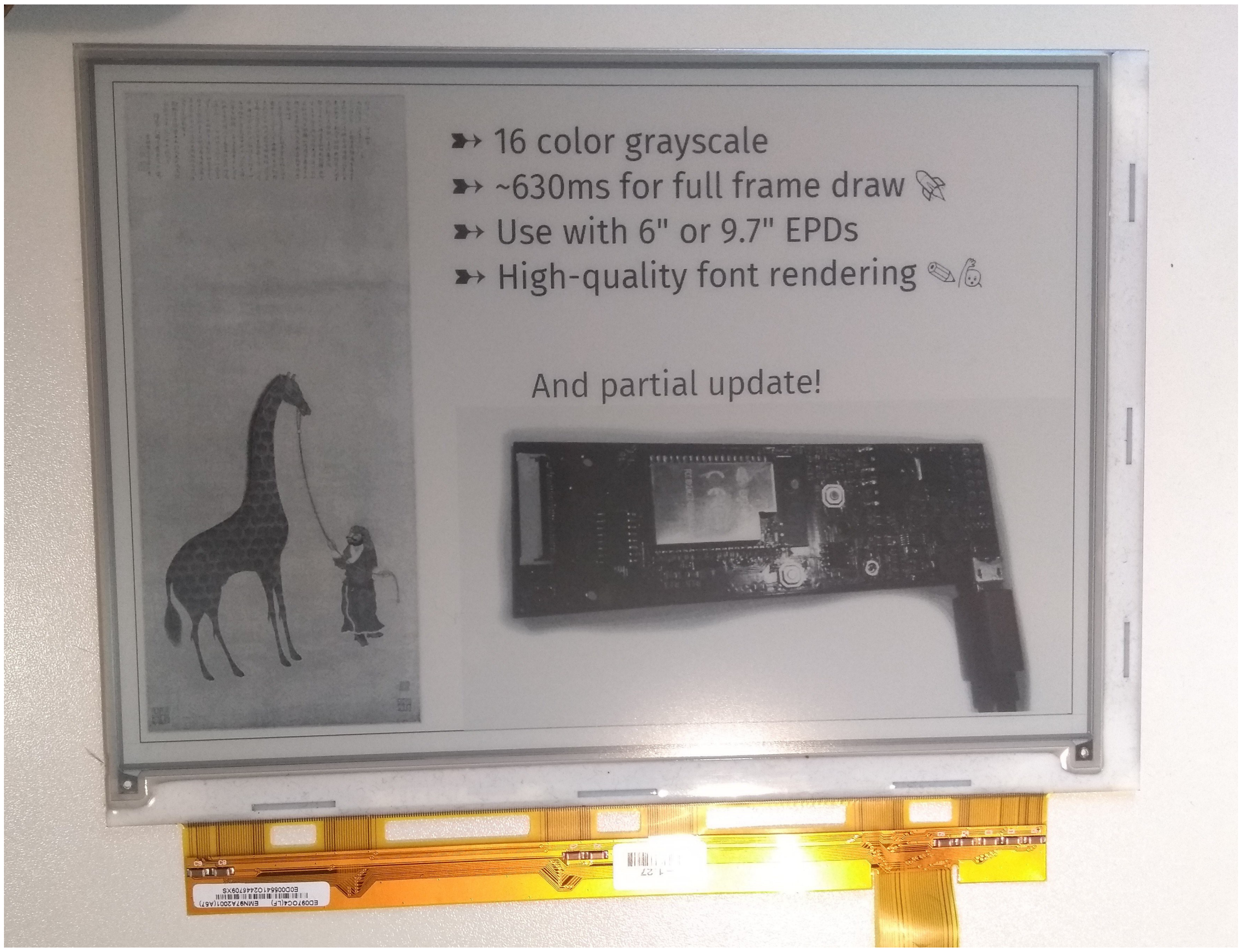
EPDiy is a driver board which talks to affordable E-Paper (or E-Ink) screens, which are usually sold as replacement screens for E-Book readers. Why are they interesting?

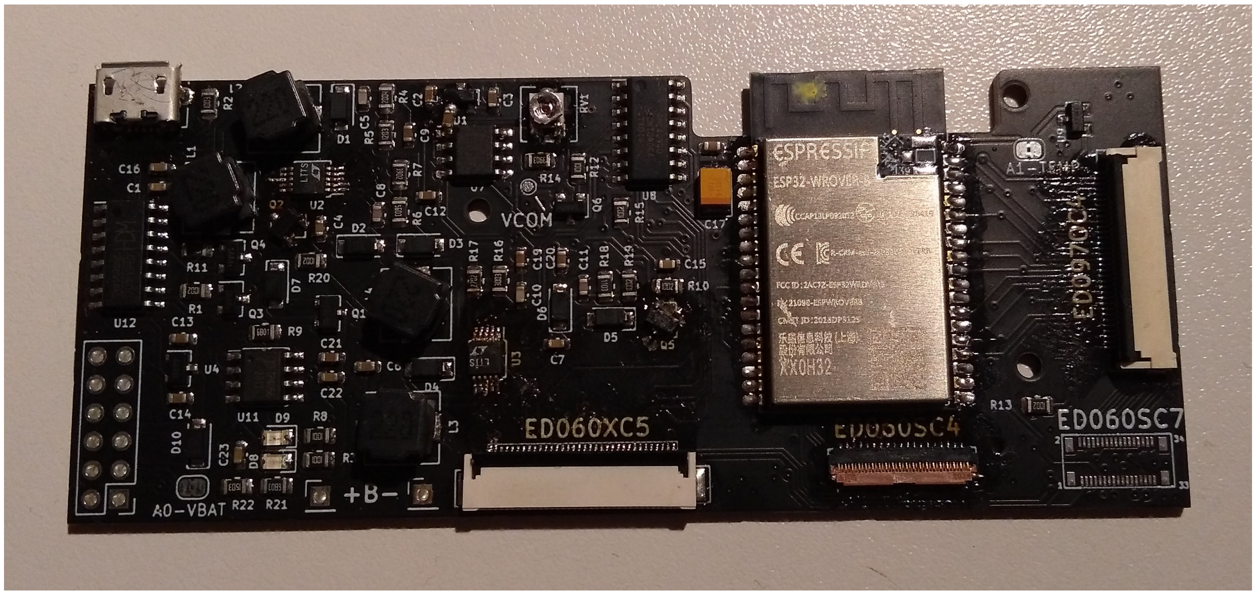
- Easy on the eyes and paper-like aesthetics
- No power consumption when not updating
- Sunlight-readable

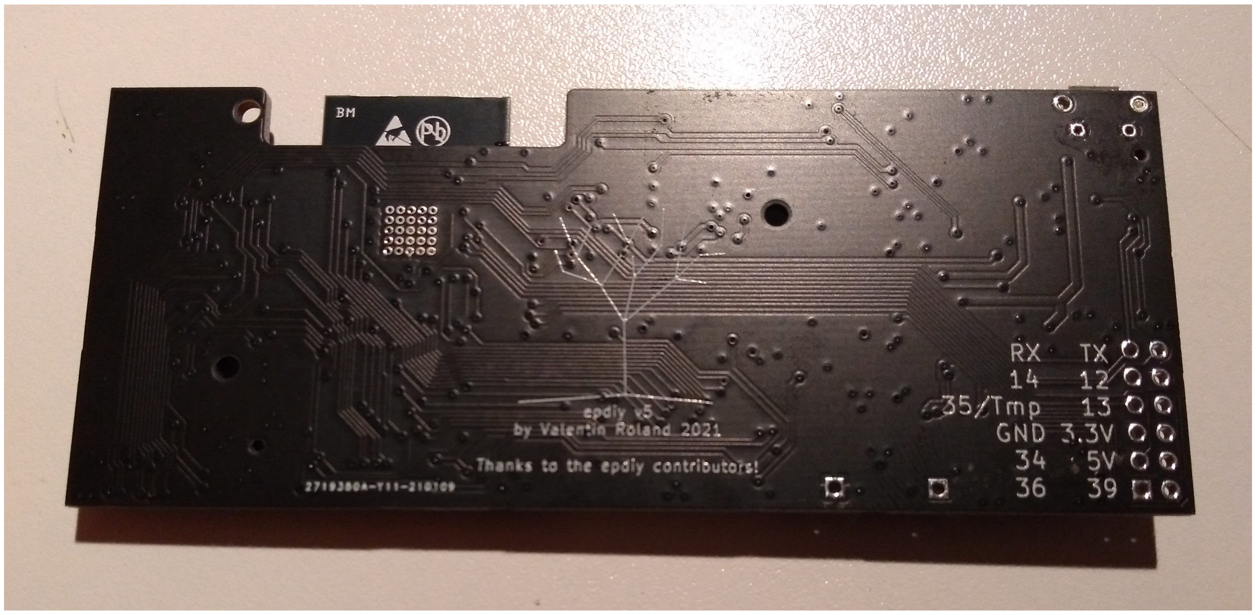
Ready-made DIY modules for this size and with 4bpp (16 Grayscale) color support are currently quite expensive. This project uses Kindle replacement screens, which are available for 20$ (small) / 30$ (large) on ebay!

The EPDiy driver board targets multiple E-Paper displays. As the driving method for all matrix-based E-ink displays seems to be more or less the same, only the right connector and timings are needed. The EPDiy PCB features a 33pin and a 39pin connector, which allow to drive the following display types: ED097OC4, ED060SC4, ED097TC2

Getting Started


16 color grayscale
~630ms for full frame draw
Use with 6" or 9.7" EPDs
High-quality font rendering
And partial update!


VCOM
ED060XC5
ED060SC4
ED060SC7
ED097OC4
A0-VBAT
+B-
ESPRESSIF
ESP32-WROVER-B


BM
epdiy v5
by Valentin Roland 2021
Thanks to the epdiy contributors!
RX TX
14 12
35/Tmp 13
GND 3.3V
34 5V
36 39

INDEX