fluepdot

@fluepke (Fabian Luepke)

Jan 02, 2023

USAGE

Hardware project for controlling legacy flipdot panels used by BVG

Note: This project was exhibited at 36C3. A large display was constructed from many flipdots. Pictures of the installation.

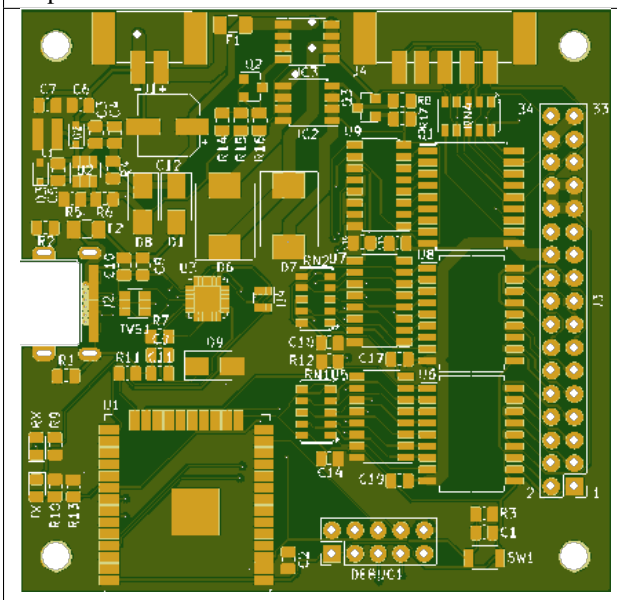
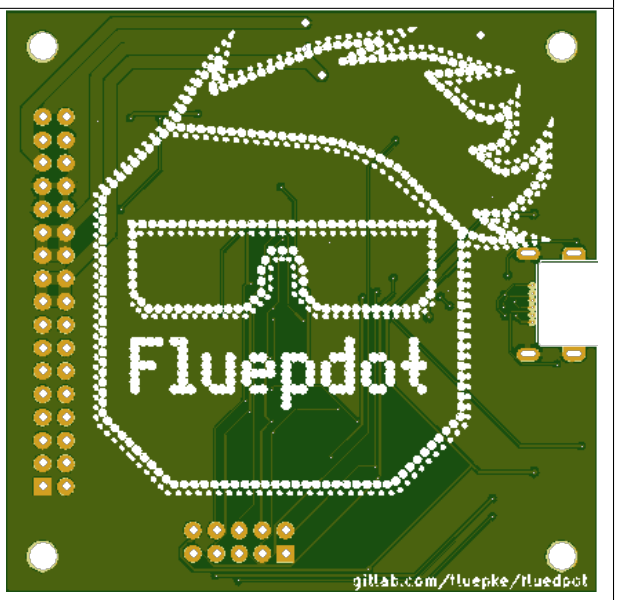
fluepdot is a hardware and C project for controlling flip disc panels which where in use by the Berlin public transport operator BVG.

CHAPTER

ONE

HARDWARE

Up to 5 flipdot panels are managed by one *fluepboard*. It is equipped with a dual core **ESP32** that supports **WLAN** and **Bluetooth**. An **RS485** transceiver can be used for wired installations. You only have to solder the THT (through hole technology) connectors which are shipped together with the fluepboard as a kit.

Top	Bottom
	

CHAPTER

TWO

SOFTWARE

The *fluepboard* comes pre-flashed with a firmware, that allows for various ways of interacting with the flipdot.

Features:

- CLI for configuration and testing purposes
- mDNS for simple service discovery
- HTTP API for framebuffer manipulations
- SNMP for monitoring and framebuffer manipulations
- BT LE support for framebuffer manipulations
- C flipdot library for custom applications

2.1 Getting started

This short guide explains how to get from receiving a *fluepdot* kit to flipping pixels the first time.

2.1.1 Hardware assembly

> **Warning:** Electrons'n'stuff can be dangerous. Use your brain and a proper 12V power supply rated for at least 3 amperes.

Perform the following steps:

0. Solder the components **J1** (USB port), **J2** (10 pins, 2.54mm, power supply, RS485 and select) and **J3** (34 pins, flat cable connector) to the board.
1. Attach the flatband cable to the boards and the fluepboard.
2. Connect the panel's select pins to the PCB. Pinout is given in the rendering. The order is left (0) to right (5), assuming panel's "top" is where the very long connector with huge amounts of pins is.

   In case you mess with the panel order, there are CLI commands for changing the order in software.
3. Provide 12V and at least 3A to the PCB as displayed on the pinout above.
4. Connect the *fluepboard* via USB to some computer.

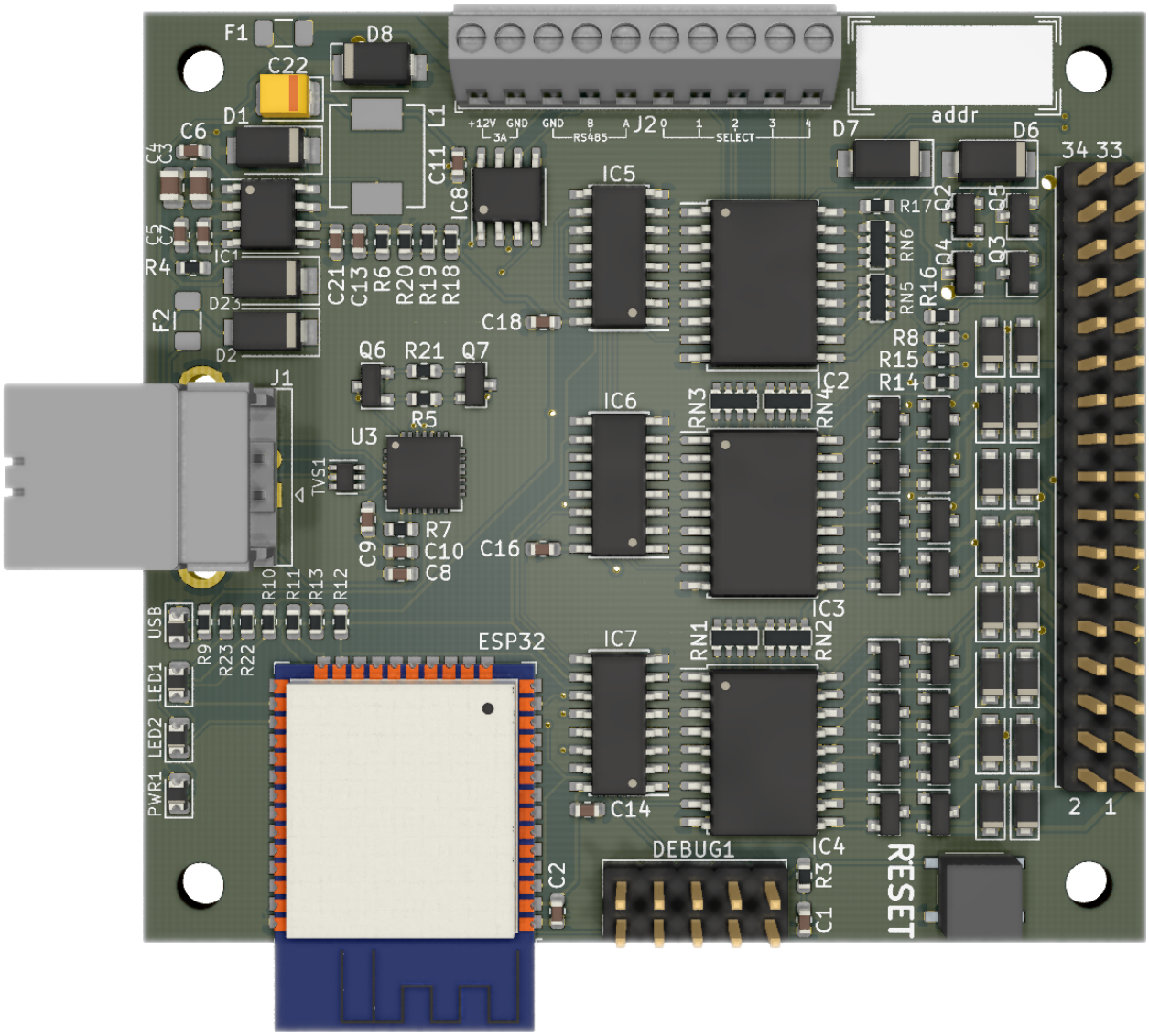


Fig. 1: J3 notch must point to the right, to the PCB's border.

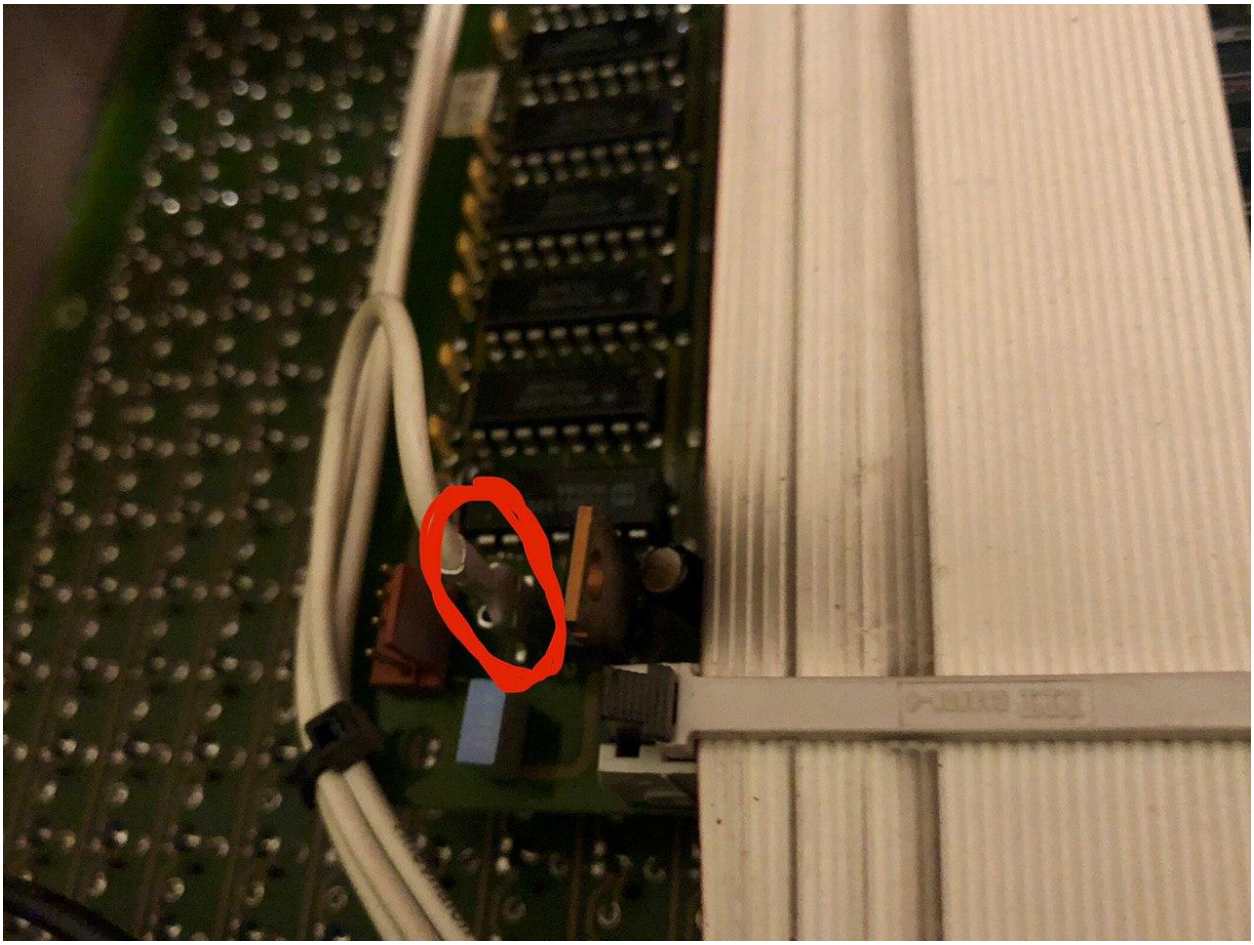


Fig. 2: Panel select pin (highlighted)

2.1.2 Software

The *fluepboard* has the USB vendor id **0x1209** and product id **0x4223** assigned by pid.codes.

Please configure your system (e.g. by writing a udev rule) to allow access to the serial port.

USB-to-serial chip in use is a **CP2102N**. Users of linux based operating systems don't need to install any drivers.

Note: Serial interface params

- **Baudrate**: 115200
- **Data Bits**: 8
- **Parity**: None
- **Stop Bits**: 1

---

0. Connect to the serial interface

```
screen /dev/ttyUSB0 115200
```

1. Configure the fluepdot panel layout

```
TODO
```

2. *Optional*: Set a hostname

```
config_hostname yet_another_iot_device.local
```

3. *Optional*: Connect to a wireless LAN

```
config_wifi_station  <ssid> [<password>]
 Configure station mode
       <ssid>  WiFi SSID
   <password>  WiFi Password
```

> **Warning:** Absolutely no attempt to protect your wireless LAN's credentials is made. Everyone having access to the fluepboard can retrieve them easily.
>
> The fluepboard's firmware intentionally does not provide means of **authentication**. It is up to you as a network operator to restrict access (e.g. by **firewalling**).

4. Save the configuration and reboot

```
config_show
config_save
reboot
```

5. Clear and set all pixels as a test, be prepared for noise

```
flipdot_clear --invert
flipdot_clear
```

2.2 Commandline interface

For debugging and configuration purposes the *fluepboard* firmware provides a simple CLI.

The serial interface is available by connecting the *fluepboard* to a computer via USB.

Serial interface works at 115200/8-N-1[1].

Note: After performing configuration changes, please issue a `config_save` followed by a `reboot` in order to apply the configuration.

2.2.1 Available commands

Type `help` to get a list of available commands:

```
help
  Print the list of registered commands

ping  [-W <t>] [-i <t>] [-s <n>] [-c <n>] [-Q <n>] <host>
  send ICMP ECHO_REQUEST to network hosts
  -W, --timeout=<t>  Time to wait for a response, in seconds
  -i, --interval=<t>  Wait interval seconds between sending each packet
  -s, --size=<n>  Specify the number of data bytes to be sent
  -c, --count=<n>  Stop after sending count packets
  -Q, --tos=<n>  Set Type of Service related bits in IP datagrams
        <host>  Host address

host  <host>
  Perform a DNS lookup
        <host>  Host to look up

traceroute  <host>
  Perform a traceroute
        <host>  Host to traceroute

reboot
  Perform a software reset

show_version
  Get version of chip and SDK

show_tasks
  Get information about running tasks

config_save
  Save the current system configuration to flash

config_load
  Load the system configuration from flash

config_show
  Show the current system configuration
```

(continues on next page)

[1] See https://en.wikipedia.org/wiki/8-N-1

(continued from previous page)

```
config_reset
  Factory reset the system configuration

config_wifi_ap  <ssid> [<password>]
  Configure AP mode
        <ssid>  WiFi SSID
    <password>  WiFi Password

config_wifi_station  <ssid> [<password>]
  Configure station mode
        <ssid>  WiFi SSID
    <password>  WiFi Password

config_hostname  <hostname>
  Set system hostname
    <hostname>  Hostname

config_panel_layout  <panel_size> [<panel_size>]...
  Configure panel count and sizes
  <panel_size>  Panel size

flipdot_clear  [--invert]
  Clear the flipdot
      --invert  Set all pixels to white instead of black

show_fonts
  List installed fonts

render_font  [-x <int>] [-y <int>] [-f <font>] <text>
  Render some text given some font
  -x, --X=<int>  Depending on aligned, either left, center or right edge of target.
  -y, --Y=<int>  Upper edge of the target area.
  -f, --font=<font>  Name of font to use
        <text>  Text to display
```

2.2.2 Usage examples

- Connect to a wireless network

```
config_wifi_station NetworkName Password
config_show
config_save
reboot
```

- Connect to an unencrypted wireless network

```
config_wifi_station NetworkName
config_show
config_save
reboot
```

- MTU test

```
ping fritz.box -s 1472
```

- Cycle all pixels

```
flipdot_clear
flipdot_clear --invert
```

2.2.3 Footnotes

2.3 SNMP

To serious business people and for optimal integration into your existing enterprise network, the *fluepboard* firmware offers SNMP integration.

The SNMP implementation provides **monitoring**, full control over **framebuffer** and rendering options as well as an **IF-MIB**, **IP-MIB** and **TCP/UDP-MIB** implementation.

You might find the SNMP implementation useful in combination with the prometheus snmp_exporter.

You can find the **MIB** (Mangement information base) file in `util/FLUEPDOT.mib`.

2.3.1 Communities

Private (write-only) community private

Public (read-only) community public

Version v2c

2.3.2 Tree

```
snmptranslate -m ./util/FLUEPDOT.mib -Tp .1.3.6.1.4.1.54722
```

```
+--fluepke(54722)
|
+--projects(1)
   |
   +--fluepdot(1)
      |
      +--framebuffer(1)
      |  |
      |  +-- -R-- Integer32 width(1)
      |  +-- -R-- Integer32 height(2)
      |  |
      |  +--pixelsTable(3)
      |     |
      |     +--pixelEntry(1)
      |        |  Index: pixelX, pixelY
      |        |
      |        +-- -R-- Integer32 pixelX(1)
      |        |        Range: 0..255
      |        +-- -R-- Integer32 pixelY(2)
      |        |        Range: 0..255
      |        +-- -RW- EnumVal   pixelState(3)
      |                 Values: dark(0), bright(1)
      |
      +--panels(2)
```

(continues on next page)

(continued from previous page)

```
|  |
|  +-- -R-- Integer32 panelCount(1)
|  |        Range: 0..5
|  |
|  +--panelTable(2)
|     |
|     +--panelTableEntry(1)
|        |  Index: panelIndex
|        |
|        +-- -R-- Integer32 panelIndex(1)
|        |        Range: 0..5
|        +-- -R-- Integer32 panelWidth(2)
|        |        Range: 20..25
|        +-- -R-- Integer32 panelX(3)
|                 Range: 0..255
|
+--renderingOptions(3)
|  |
|  +--delayTable(1)
|  |  |
|  |  +--delayEntry(1)
|  |     |  Index: column
|  |     |
|  |     +-- -R-- Integer32 column(1)
|  |     |        Range: 0..255
|  |     +-- -RW- Integer32 columnPreDelay(2)
|  |     +-- -RW- Integer32 columnSetDelay(3)
|  |     +-- -RW- Integer32 columnClearDelay(4)
|  |
|  +--panelOrderTable(2)
|  |  |
|  |  +--panelOrderEntry(1)
|  |     |  Index: orderIndex
|  |     |
|  |     +-- -R-- Integer32 orderIndex(1)
|  |     |        Range: 0..5
|  |     +-- -RW- Integer32 panelOrderIndex(2)
|  |              Range: 0..5
|  |
|  +-- -RW- EnumVal   renderingMode(3)
|           Values: full(0), differential(1)
|
+-- -R-- Counter64 pixelsFlipped(4)
+-- --W- Integer32 dirtyBit(69)
```

2.3.3 Usage examples

- Set the pixel *x=23, y=4* to bright

```
snmpset -v 2c -c private -m ./util/FLUEPDOT.mib fluepdot0.cluster.ap-
↪south-1.yolo.network FLUEPDOT-MIB::pixelState.23.4 i bright
```

- Set the dirty bit (aka tell the *fluepboard* to render framebuffer contents)

```
snmpset -v 2c -c private -m ./util/FLUEPDOT.mib fluepdot0.cluster.ap-
↪south-1.yolo.network FLUEPDOT-MIB::dirtyBit.0 i 1
```

(continues on next page)

(continued from previous page)

- Get the number of flipped pixels

```
snmpget -v 2c -c public -m ../util/FLUEPDOT.mib 192.168.178.94 FLUEPDOT-
↪MIB::pixelsFlipped.0
```

2.4 HTTP API

2.4.1 Framebuffer

For simple integration into your projects, the *fluepboard* firmwares ships with a simple to use HTTP API.

Framebuffer encoding

Framebuffers are **ASCII**-encoded. The only allowed characters are **<space>** `0x20`, **X** `0x58` and **``\n``** `\0x0A`.

A **bright** (set) pixel is encoded as an X.

A **dark** (cleared) pixel is encoded as a space character.

Each line in a framebuffer has **exactly** the same amount of characters, which is **exactly** the *fluepdot's* width plus one newline character.

Each framebuffer has **exactly** 16 lines.

Lines are terminated by a single `\n`. No carriage return is used.

Framebuffer manipulations

GET /framebuffer

Params

GET None

POST None

Gets the current framebuffer encoded as explained above. This endpoint can be used to calculate the framebuffer dimensions, thus there is no seperate endpoint for retrieving the geometry.

POST /framebuffer

Params

GET None

POST Raw framebuffer encoded as explained above

Draw the posted framebuffer to the *fluepdot*.

GET /pixel

Params

GET x - The x coordinate y - The y coordinate

Get the current pixel at the given coordinate encoded as above

POST /pixel

Params

GET

x The x coordinate (ascii encoded decimal value)

y The y coordinate (ascii encoded decimal value)

Sets the pixel at the given coordinate to bright.

DELETE /pixel

Params

GET

x The x coordinate (ascii encoded decimal value)

y The y coordinate (ascii encoded decimal value)

Set the pixel at the give coordinate to dark.

POST /framebuffer/text

Params

POST Text to display

GET

x The x coordinate to render text to

y The y coordinate to render text to

font The font's name to use for rendering text. Retrieve a list of fonts at *GET /fonts*

Render a string

GET /fonts

Returns a list of installed fonts

For each font there are two lines: First the **full_name** and then **short_name** which is used with the *POST /framebuffer/text* eAPI endpoint.

2.4.2 Rendering options

Rendering mode

GET /rendering/mode

Retuns an ASCII printed integer which value is defined as follows:

enum flipdot_rendering_mode_t

Values:

FULL = 0
Always render the full framebuffer without skipping anything.

DIFFERENTIAL = 1
Redraw only those pixels that changed.

PUT /rendering/mode

Params

GET None

POST ASCII printed integer, which value is to be interpreted as stated above.

Rendering timings encoding

Rendering timings are **ASCII**-encoded. The only allowed characters are 0-9 (`0x30-0x39`) and `\\n`.

For each *fluepdot* column, there are **exactly** 3 rows:

1. **Pre delay**

   How long to wait (50 microseconds steps) before rendering to the column.

2. **Clear delay**

   How long to power the coils in order to clear the column (in 50 microseconds steps).

3. **Set delay**

   How long to power the coils in order to set the column (in 50 microseconds steps)

Each line has **exactly** 5 characters and one trailing `\\n`. You have to pad with zeros. Each line contains the decimal value in ascii-encoded form.

> **Warning:** Powering a coil repeatedly for a long duration *might* cause the coil to overheat and or fail. Decreasing the timings might result in higher **framerate**, but might result in not all pixels flipping.
>
> Usually **1600uS** are enough to reliably flip all pixels. This is the **default**.

GET /rendering/timings

Returns the timing configuration encoded as explained above.

POST /rendering/timings

Params

GET None

POST Timing configuration as explained above.

Set the timing configuration.

2.5 Firmware update

2.5.1 Using service utility

The *fluepboard*'s firmware can be easily updated using a **standalone** binary for GNU/Linux (amd64) that is generated by the *Gitlab CI* build process of this project.

You can get the latest version here.

Make sure your current user has permission to use the serial port (in this example */dev/ttyUSB0*), then run:

```
./service_utility -serial.port /dev/ttyUSB0
```

This utility will flash: * the partition table * the bootloader * the factory application

2.5.2 Using esptool.py

You can also flash the firmware using esptool.py. Download and unpack the compiled firmware, then run:

```
esptool.py --chip esp32 --before=default_reset --after=hard_reset write_
↪flash --flash_mode dio --flash_freq 40m --flash_size 4MB 0x8000 partition_
↪table/partition-table.bin 0x1000 bootloader/bootloader.bin 0x10000 flipdot-
↪firmware.bin
```

2.6 Compiling

A multistage **Docker** environment is used to build firmware and documentation.

1. Clone the repository *recursively*.

```
git clone --recursive https://gitlab.com/fluepke/fluepdot.git
```

2. Build or pull the build container.
   - If you have a high bandwidth internet connection downloading is fastest (~ 10 GB):

```
docker pull fluepke/fluepdot-build-environment
```

   - If you have a low bandwidth internet connection building is faster (still ~ 3 GB)

```
docker build --force-rm -t fluepke/fluepdot-build-environment -f␣
↪Dockerfile .
```

3. Build the second stage container

```
docker build --force-rm -t fluepdot -f Dockerfile.build .
```

4. Run the second stage container and flash the image (assuming your flipdot's serial device is at `/dev/ttyUSB0`)

```
docker run -d --name fluepdot --device /dev/ttyUSB0:/fluepdot-device:rwm
docker exec -it fluepdot \
    -w "/fluepdot/software/firmware" \
    -e ESPTOOL_PORT='/fluepdot-device' \
    -e ESPTOOL_BAUD='480000' \
    make flash
```

2.7 Flipdot API reference

2.7.1 Flipdot

Flipdot structs

`struct flipdot_t`
Main flipdot data structure Passed as a pointer to most of the flipdot functions.

Public Members

EventGroupHandle_t **`event_group`**
Used for inter task communication with the flipdot task.

TaskHandle_t ***`task`**
Flipdot rendering task Performs the following actions, when `FLIPDOT_FRAMEBUFFER_DIRTY_BIT` in `event_group` was set:

- Delete `framebuffer_internal_old`
- Copy `framebuffer_internal` to `framebuffer_internal_old`
- Delete `framebuffer_internal`
- Copy `framebuffer` to `framebuffer_internal`
- Delete `internal_rendering_options`
- Copy `rendering_options` to `internal_rendering_options`
- Render the flipdot as configured

framebuffer_t ***`framebuffer`**
You can manipulate this framebuffer as you please.

Once done, call `flipdot_set_dirty_flag`

See *flipdot_set_dirty_flag*

framebuffer_t ***framebuffer_internal**
Internal framebuffer, used to avoid race conditions when rendering.

framebuffer_t ***framebuffer_internal_old**
Previous internal framebuffer, used for differential rendering.

flipdot_panel_t ***panels**
Panel configuration.

uint8_t **panel_count**
Number of attached panels.

uint8_t **width**
Total flipdot width in pixels.

flipdot_rendering_options_t ***rendering_options**
Rendering options.

flipdot_rendering_options_t ***internal_rendering_options**
Internal rendering options, used internally to avoid race conditions when rendering.

bool **power_status**
Flipdot power status.

Note Use `flipdot_set_power` to manipulate this member

See *flipdot_set_power*

spi_device_handle_t **spi_device_handle**
SPI device handle.

Note SPI is used for controlling shift registers IC5, IC6 and IC7.

flipdot_io_state_t **io**
Flipdot GPIO state, used for shifting out IO states via SPI to IC5, IC6 and IC7.

unsigned long long **pixels_flipped**
Counts the pixels that changed their color since startup.

SemaphoreHandle_t **semaphore**
Used for mutual exclusive hardware access.

struct flipdot_panel_t
Location and size of a flipdot panel.

A flipdot consists of up to 5 panels. Width of a panel is typically 20 or 25 pixels. Height is hardcoded to be 16.

Public Members

uint8_t **width**
Panel width.

uint8_t **x**
Panel offset.

struct flipdot_configuration_t
Used to initialize a flipdot.

See *flipdot_initialize*

Public Members

uint8_t **panel_count**
Number of attached panels.

uint8_t **panel_size**[(5)]
Width of the attached panels.

Flipdot functions

esp_err_t **flipdot_initialize** (*flipdot_t *flipdot*, *flipdot_configuration_t *flipdot_configuration*)
Initializes a flipdot for use and starts the flipdot task.

Note To actually use the flipdot you have to power it up using flipdot_set_power.

See *flipdot_set_power*

Return

- `ESP_OK`: success
- `ESP_ERR_INVALID_ARG`: invalid arguments (i.e. null pointers)
- `ESP_ERR_NO_MEM`: allocating internal buffers failed
- other: failure

Parameters

- `flipdot`: Flipdot to initialize, must not be NULL
- `flipdot_configuration`: Configuration to apply, must not be NULL

esp_err_t **flipdot_set_power** (*flipdot_t *flipdot*, bool *power_status*)
Power on / off the flipdot.

Note This does not stop the rendering task

Return

- ESP_OK success
- ESP_TIMEOUT timeout waiting for flipdot->semaphore

Parameters

- `flipdot`: The flipdot handle
- `power_status`:
  - True power on the flipdot
  - False power off the flipdot

bool **flipdot_get_power** (*flipdot_t *flipdot*)
Returns if the flipdot is powered on.

Parameters

- `flipdot`: The flipdot handle

esp_err_t **flipdot_set_dirty_flag** (*flipdot_t *flipdot*)
Notfies the flipdot task, that the framebuffer is dirty and needs redrawing.

2.7.2 Framebuffer

Framebuffer struct

struct framebuffer_t
A framebuffer with bottom left corner as x=0, y=0.

Public Members

uint16_t ***columns**
16 pixels high columns

uint8_t **width**
Framebuffer width.

esp_err_t **flipdot_framebuffer_init** (*framebuffer_t *framebuffer*, uint8_t *width*)
Initialize a framebuffer.

Return

- `ESP_ERR_INVALID_ARG`: framebuffer was NULL or width was 0
- `ESP_ERR_NO_MEM`: failed to allocate memory

Parameters

- `framebuffer`: The framebuffer_t* to initialize
- `width`: Framebuffer width. Must be > 0

void **flipdot_framebuffer_free** (*framebuffer_t *framebuffer*)
Free a framebuffer_t* Framebuffer is freed and the pointer set to NULL to avoid use after free bugs.

Parameters

- `framebuffer`: The framebuffer to free

Framebuffer manipulation

void **flipdot_framebuffer_clear** (*framebuffer_t *framebuffer*)
Clear a framebuffer (set all pixels dark)

Parameters

- `framebuffer`: The framebuffer to clear

bool **flipdot_framebuffer_get_pixel** (*framebuffer_t *framebuffer*, uint8_t *x*, uint8_t *y*)
Get a single pixel value.

See *framebuffer_t* for an explanation of the coordinate system

See *framebuffer_t* for an explanation of the coordinate system

Return

- true Pixel is set (bright)
- false Pixel is not set (dark) or out of bounds read or framebuffer was NULL

Parameters

- `framebuffer`: The framebuffer to retrieve pixel value from
- `x`: X location,

Parameters

- `y`: Y location,

esp_err_t **flipdot_framebuffer_set_pixel**(*framebuffer_t* **framebuffer*, uint8_t *x*, uint8_t *y*, bool *value*)

Set a single pixel value.

See *framebuffer_t* for an explanation of the coordinate system

See *framebuffer_t* for an explanation of the coordinate system

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG out of bounds or framebuffer was NULL

Parameters

- `framebuffer`: The framebuffer to manipulate
- `x`: X location,

Parameters

- `y`: Y location,

Parameters

- `value`:
  - true Pixel is set (bright)
  - false Pixel is not set (dark) or out of bounds read

Framebuffer de/encoding

char ***flipdot_framebuffer_encode_line**(*framebuffer_t* **framebuffer*, char **dst*, size_t *dst_len*, uint8_t *y*)

ASCII encode a horizontal framebuffer line.

Return

- NULL failure
- char* Pointer to the encoded line

Note

See *flipdot_framebuffer_encode_pixel* is used for ASCII encoding single pixels

Parameters

- `framebuffer`: The framebuffer to encode
- `dst`: Destination char buffer
- `dst_len`: Destination buffer size, must be framebuffer->width + 1 to accomodate the trailing 0 character

esp_err_t **flipdot_framebuffer_decode_line**(*framebuffer_t* *framebuffer, char *src, size_t src_len, uint8_t y)

ASCII decode a horizontal framebuffer line.

Return

- ESP_OK success
- ESP_INVALID_ARG framebuffer was null, or src_len != framebuffer->width + 1

Note

See *flipdot_framebuffer_decode_pixel* is used for ASCII decoding single pixels

Parameters

- `framebuffer`: The framebuffer to decode into
- `src`: Source char buffer
- `src_len`: Source buffer size, must be framebuffer->width + 1 to accomodate the trailing 0 character

char **flipdot_framebuffer_encode_pixel**(bool *value*)

ASCII encode a single pixel.

Return

-

See FLIPDOT_PIXEL_SET pixel is set

-

See FLIPDOT_PIXEL_CLEAR pixel is not set

Parameters

- `value`: The pixel value to encode

bool **flipdot_framebuffer_decode_pixel**(char *src*)

ASCII decode a single pixel.

Return

- true pixel is set (src == FLIPDOT_PIXEL_SET)
- false pixel is not set (all other cases)

Parameters

- `src`: Character to decode into a pixel value

esp_err_t **flipdot_framebuffer_printf**(*framebuffer_t* *framebuffer)

Printf the given framebuffer to STDOUT.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG framebuffer was NULL
- ESP_ERR_NO_MEM could not allocate enough memory

Parameters

- `framebuffer`: The framebuffer to print

Framebuffer diffing

esp_err_t **flipdot_framebuffer_compare** (*framebuffer_t* *a, *framebuffer_t* *b, unsigned int *diff)
Compares two framebuffers.

Return

- ESP_INVALID_ARG a was NULL and / or b was NULL and / or diff was NULL and / or a->width != b->width
- ESP_OK success

Parameters

- `a`: Framebuffer 1
- `b`: Framebuffer 2
- `diff`: Pointer to an integer to store the number of changed pixels

esp_err_t **flipdot_framebuffer_compare_partial** (*framebuffer_t* *a, *framebuffer_t* *b, uint8_t *start_x*, uint8_t *width*, unsigned int *diff)
Compares two framebuffers partially.

Return

- ESP_INVALID_ARG a was NULL and / or b was NULL and / or diff was NULL and / or a->width != b->width and / or a->width < start_x + width
- ESP_OK success

Parameters

- `a`: Framebuffer 1
- `b`: Framebuffer 2
- `start_x`: X offset from where to start comparing
- `width`: Width of the view port to compare
- `diff`: Pointer to an integer to store the number of changed pixels

2.7.3 Rendering options

struct flipdot_rendering_options_t
Flipdot rendering options.

Public Members

uint8_t **width**
Flipdot width.

uint8_t **panel_count**
Number of panels attached to flipdot.

flipdot_rendering_mode_t **mode**
Flipdot rendering mode.

flipdot_rendering_delay_options_t ***delay_options**
Flipdot timing options.

uint8_t ***panel_order***
Order in which panels are rendered.

struct flipdot_rendering_delay_options_t
Flipdot timing options.

Public Members

uint16_t **pre_delay**
How long to wait before rendering a column (in 10ms counts)

uint16_t **clear_delay**
How long to power the row high side drivers (in 10ms counts)

uint16_t **set_delay**
How long to power the column clear low side driver (in 10ms counts)

enum flipdot_rendering_mode_t
Values:

FULL = 0
Always render the full framebuffer without skipping anything.

DIFFERENTIAL = 1
Redraw only those pixels that changed.

esp_err_t **flipdot_rendering_options_initialize**(*flipdot_rendering_options_t* **rendering_options*, uint8_t *panel_count*, uint8_t *width*)
Allocates internal data structures and initializes them with reasonable defaults.

Return

- `ESP_ERR_INVALID_ARG`: rendering_options was NULL, panel_count was 0 or width was 0
- `ESP_ERR_NO_MEM`: could not allocate memory for internal data structures
- `ESP_OK`: success

Parameters

- `options`: The *flipdot_rendering_options_t* to initialize, must not be NULL.
- `panel_count`: Number of panels available, must be larger than 0
- `width`: Number of columns available, must be larger than 0

esp_err_t **flipdot_rendering_options_copy**(*flipdot_rendering_options_t* **dest*, *flipdot_rendering_options_t* **src*)
Copies rendering options from src to dest.

Parameters

- `dest`: Allocate using `calloc(1, sizeof(flipdot_rendering_options_t))`
- `src`: Source

void **flipdot_rendering_options_free**(*flipdot_rendering_options_t* **rendering_options*)
Free a flipdot_rendering_options_t*.

Parameters

- `rendering_options`: flipdot_rendering_options_t* to free.

INDEX

D

F