PLASTIC LOGIC

Plastic Logic Software Manual

Description: Plastic Logic Software Manual
Date: 07 April 2014
Revision: v009

PLASTIC LOGIC

Table of contents

1 Introduction

Thank you for reading the Plastic Logic Software Manual. You'll find in this document all you need to know to use the **Plastic Logic Display Evaluation Kits (PLDEK)** and especially the software designed to run on them.

Plastic Logic maintains a collection of C and Python libraries and applications especially designed for **E-Paper** called the **Plastic Logic Software Development Kit** (PLSDK). A brief introduction is available in this manual, and some separate developer manuals are available with full library API documentation. You can get the source code for the PLSDK on Github.

Evaluation kits typically run a GNU/Linux system, and kernel drivers for the Plastic Logic hardware and associated E-Paper controllers are also maintained by Plastic Logic. You can also get the source code of the Plastic Logic Linux kernels on Github.

Some other evaluation kits run on low-power micro-controllers (MCU) without any actual operating system but some reference software from Plastic Logic to operate various E-Paper displays. Please see the Plastic Logic MCU Manual for more details specific to these kits.

1.1 E-Paper Considerations

E-Paper displays behave differently from most current common display technologies (LCD, OLED, CRT...). A first difference is that a sequence of signals (Waveforms) need to be generated in order to **update each pixel from one state to another**. This means some extra work has to be done by the software to describe *how* to update the pixels, not just *what* colour the pixels should show.

Another important point is that **the image remains on the display** after the power has been turned off. In other words, the display only needs to be powered while changing its contents. This implies that the software typically turns the display power on and off automatically before and after each update.

1.2 Display Evaluation Kits (PLDEK)

The Plastic Logic Display Evaluation Kits are flexible platforms which can be used to build some prototypes using E-Paper technology. They provide expansion connectors to integrate other application components, and their hardware designs can be used as references for commercial products. Likewise, the software installed on the evaluation kits enables a wide variety of application demonstrators, proofs of concepts and building bricks for definitive product solutions. The libraries take into account all the general E-Paper aspects as well as Plastic Logic displays specificities in order to get the most out of the technology.

The main hardware platforms currently used are:

- **BeagleBone** board (TI OMAP3) in conjunction with Plastic Logic expansion boards for the Epson S1D13541 and S1D13524 E-Paper controllers
- **Freescale i.MX5** EVK which has a powerful built-in hardware E-Paper controller and can drive large displays with advanced hardware graphical acceleration
- **TI MSP430** MCU on the Parrot board, designed by Plastic Logic

2 E-Paper Display Operation

2.1 Waveforms

A *waveform* in the context of EPDs is a series of voltages that need to be applied to a pixel in order to change its value. There are three main *waveform types* defined by Plastic Logic, each with different properties. Each waveform is characterised by a combination of an update mode and a number of possible pixel values (greyscale or colour depth). Waveforms are identified in the PLSDK with the following path-like syntax:

`type/depth/options`

If only `type` is provided, then the default `depth` and `options` will be used whenever possible. Only a few special waveforms use `options`. The three main waveforms listed below provide a simple rule of thumb which covers most of the usual application cases:

- `refresh`

Each pixel is refreshed, which causes a flashing effect but achieves best resulting image quality. This should normally be used over the whole screen to clear any ghosting produced by previous updates. Using the `refresh` waveform over a limited display area usually causes some edge artefact so it should be avoided.

By default, the `refresh` waveform uses **full greyscale/colour depth** for maximum image quality.

- `delta`

Only the pixels which values need to change are refreshed. This is useful to show shapes that are not rectangular or to produce animation effects. It can also be used when turning pages in a document with a minimal flashing effect, although this will build up some ghosting and edge artefact which can eventually be cleared with a `refresh` update.

By default, the `delta` waveform uses **full greyscale/colour depth** for maximum image quality.

- `delta/mono` (also refered to as `mono`)

This waveform only supports pure **black and white** and does not cause any flashing as it is a *direct drive waveform*. Like the standard `delta` waveform, only the pixels that need to change are updated. The main difference being that only target full black and white pixels are updated, intermediate target grey levels are left unchanged.

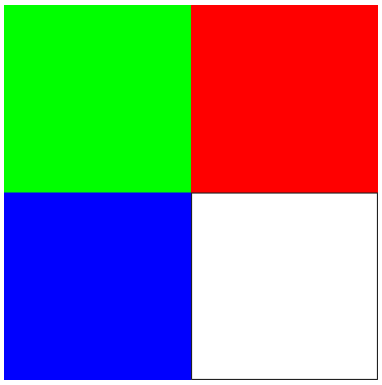
It is faster than the full greyscale/color `refresh` and `delta` waveforms, so it usually constitutes a good choice to make fast and responsive user interface elements such as menus, handwriting tools and virtual keyboards.

Primary and secondary colours

When used with a colour display (fitted with one of the Colour Filter Arrays), the `mono` waveform can be used to show white, black but also primary (red, green, blue) and secondary (cyan, magenta, yellow) colours. These eight *colours* are recommended in order to build efficient user interfaces.

2.2 Colour Filter Arrays

Colour EPD is achieved by applying a **C**olour **F**ilter **A**rray on top of the basic monochrome EPD. This typically converts each pixel into a red, green, blue or white sub-pixel as shown in the example below:



Green, red, blue and white CFA (standard name: `GR_BW`)

The logical resolution of the display is then halved on both horizontal and vertical axes. So for a native monochrome resolution of 1280x960, the equivalent colour resolution becomes 640x480.

Many CFA configurations are possible. The naming convention is to use a letter for each colour on a row of sub-pixels, and each row is separated by an underscore "_". For example, the name used to identify the CFA shown above is `GR_BW`.

The following CFA modes are supported by this `plsdk` version:

- `GR_BW`:

  Green, red, blue and white 2x2 pattern. This is the standard configuration.

- `GR_BW_TR180`:

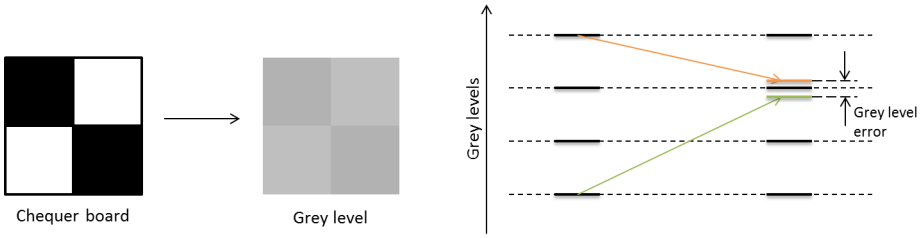
  Same as GR_BW on the bottom half of the screen, and WB_RG on the top half (split a half of the vertical resolution). This is typically used with 1x2 tiled displays where the top one is rotated by 180 degrees relatively to the bottom one.

The PLSDK supports Plastic Logic's CFA configurations. For example, when the `GR_BW` CFA mode is enabled, all the graphical operations will populate 4 actual sub-pixels for each logical pixel as seen by the application.

2.3 E-Paper Visual Artefacts

2.3.1 Ghosting

Ghosting manifests itself with remnants of the previous state of the display after some pixels have been updated. This is due to an error between the ideal target value of a pixel and its actual luminance; it is intrinsic to any EPD technology. The `refresh` waveforms with full greyscale or colour depth are not direct drive; they produce a flashing effect to reduce ghosting by first driving the pixels to full black or white before the target grey level.



Effects of ghosting using a direct drive waveform (no flashing)

2.3.2 Edge artefact

E-Paper media is composed of many more small particles than the number of pixels. When driving one pixel, particles on the edges of neighbouring pixels can also be altered if they are not actively driven. The result is particularly visible along shapes with a high contrast contour, for example black text on white background. Driving only the black pixels will interfere with the surrounding white pixels, changing their state slightly. This manifests itself with a reminiscent outline refered to as *edge artefact*, as illustrated on the figure below.

Edge artefact shown after deleting a shape with a sharp contour

This typically occurs when turning a page in a text document with a `delta` waveform. The simplest way to avoid this issue is to perform a full screen update with a `refresh` waveform, so all the pixels are actively driven in order to prevent any interference between them. It is also sometimes possible to hide this issue by making the update area match an existing black frame.

2.4 Temperature Waveform Compensation

The characteristics of the E-Paper media vary significantly with temperature. For this reason, the waveform data needs to be adjusted when the temperature changes. A waveform library contains waveform tables, each of them corresponding to a given temperature range. The EPD controller needs to have the correct waveform table loaded in memory matching the display temperature before starting any display update.

This feature can be implemented in several different ways. Generally speaking, two modes can be distinguished:

Automatic temperature control

This is the typical mode of operation; the EPD control system (hardware and associated software driver) has a means of capturing the display temperature and automatically loading the waveform table with a matching temperature range.

Manual temperature control

In this case, the EPD control system relies on the user to take the action to either provide the temperature measurement or the waveform table data.

2.5 VCOM Calibration

2.5.1 About the VCOM voltage

The VCOM voltage is applied to a common electrode for all pixels on top of electrophoretic displays (EPD). If the value is too high, a Plastic Logic EPD will look lighter; if the voltage is too low the display will look darker.

Another visual effect of an in correct VCOM is that pixels which are not actively driven during a display update will progressively fade to black or white if the applied voltage is respectively too low or too high.

For more information about the VCOM voltage, please read the separate document "How to integrate VCOM calibration".

2.5.2 VCOM and Hardware EEPROM format v1

This section details the version 1 of the format of the VCOM calibration and hardware information EEPROM as it can be found on Hummingbird Z6.x and Z7.x boards for example. It contains necessary information to enable the software to operate the hardware correctly: how to **calibrate the VCOM voltage**, how to **control the high-voltage** display power supplies and **which signals are available** on the board interface.

The EEPROM device follows the 24c01 standard configuration, with a total size of 128 bytes.

A few general conventions:

- Numbers are stored in big-endian format.
- The CRC16 follows the reference implementation as published by Plastic Logic; see pldata.crc16 for more details.
- Strings are always null-terminated, so for a storage length of 9 bytes the maximum string length is 8 effective characters.

The VCOM and hardware EEPROM format v1 is defined below:

Offset	Size	Field name	Description
Version			
0	1	VERSION	Unsigned 8-bit integer set to 1.
DAC characteristics			
1	2	DAC_X1	Signed 16-bit integer with the first DAC register value (usually 25% of full range i.e. 64 for 8-bit DAC).
3	2	DAC_Y1	Signed 16-bit integer with the VCOM voltage in mV associated with DAC_X1.
5	2	DAC_X2	Signed 16-bit integer with the second DAC register value (usually 75% of full range i.e. 191 for 8-bit DAC).
7	2	DAC_Y2	Signed 16-bit integer with the VCOM voltage associated with DAC_X2.
Gate PSUs			
9	4	VGPOS	Signed 32-bit integer with the VGPOS voltage in mV.
13	4	VGNEG	Signed 32-bit integer with the VGNEG voltage in mV.
17	4	IDEAL_VGSWING	Ideal gate swing voltage in mV.
Hardware information			
21	9	BOARD_TYPE	String with the board type.
30	1	BOARD_VER_MAJ	Board major version number.
31	1	BOARD_VER_MIN	Board minor version number.
32	1	VCOM_MODE	VCOM calibration mode identifier.
33	1	HV_PMIC	HV-PMIC part identifier.
34	1	VCOM_DAC	VCOM DAC part identifier.
35	1	VCOM_ADC	VCOM ADC part identifier.
36	1	IO_CONFIG	Configuration identifier for the set of I/O signals.
37	1	I2C_MODE	Configuration identifier for the I2C bus master for display hardware devices.
38	1	TEMP_SENSOR	Temperature sensor chip identifier.
39	1	FRAME_BUFFER	Frame buffer memory identifier.
40	1	EPDC_REF	ePDC part reference dentifier.
41	2	ADC_SCALE_1	Signed 16-bit integer value with the dividend of the ratio between the VGSWING voltage and the one measured by the ADC; set to 1 if no ADC is used.
43	2	ADC_SCALE_2	Signed 16-bit integer value of the divisor to be used with ADC_SCALE_1; set to 1 if no ADC is used.
Checksum			
45	2	CRC	CRC-16 checksum of all the data (offsets 0 to 45).

See Programming EEPROMs for instructions on how to store this data in an EEPROM device.

VCOM_MODE

- **0:** No VCOM calibration available.
- **1:** Closed-loop dynamic VCOM calibration with an ADC.
- **2:** Open-loop static VCOM calibration based on factory measurements of the characteristics of the VCOM PSU.

HV_PMIC

- **0:** No HV-PMIC available
- **1:** Maxim MAX17135
- **2:** TI TPS65185

VCOM_DAC

- **0:** No VCOM DAC available.
- **1:** Discrete Maxim MAX5820 (8 bits)
- **2:** Integrated Maxim MAX17135 (8 bits)
- **3:** Integrated TI TPS65185 (9 bits)

VCOM_ADC

- **0:** No VCOM ADC available.
- **1:** Maxim MAX11607

IO_CONFIG

- **0:** No standard I/O configuration defined for this board.
- **1:** Serial connection as on Hummingbird Z6.0, 24-way connector defined as follows:

  - **1-4:** VMAIN
  - **5:** HDC
  - **6:** SCL
  - **7:** SDA
  - **8:** SPI_CLK
  - **9:** SPI_MOSI
  - **10:** SPI_CS
  - **11:** SPI_MISO
  - **12:** HRDY
  - **13:** HIRQ
  - **14:** RST_
  - **15:** HVSW_CTRL
  - **16:** PMIC_EN
  - **17:** PMIC_POK
  - **18:** PMIC_FLT
  - **19:** RESERVED1
  - **20:** RESERVED2
  - **21-24:** GND

2: Serial connection as on Hummingbird Z7.0, same as I/O definitions **1:** but with this extra signal:

19: 3V3_EN

3: Newer serial connection as on more recent Hummingbird Z6.3 and Z7.2, same as I/O definitions **1:** but with these extra signals:

19: 3V3_EN

20: BB_SLEEP

4: Serial connection as on Robin v1.0, 10-way connector defined as follows:

1-2: GND

3-4: VBATT

5: SPI_CS

6: HIRQ

7: SPI_CS2

8: SPI_MOSI

9: SPI_MISO

10: SPI_CLK

I2C_MODE

0: No I2C devices.

1: Host CPU I2C master bus.

2: Display integrated I2C master bus (typically S1D13541).

3: On-board S1D13524 I2C master bus.

4: NXP SC18IS600/601 SPI to I2C bridge.

TEMP_SENSOR

0: No separate temperature sensor device. Other components such as the HV-PMIC or EPD controllers may stil be present on the board and they contain their own temperature sensors.

1: LM75 compatible I2C temperature sensor device.

FRAME_BUFFER

0: No frame buffer memory device.

1: Micron MT46H16M32LF 512Mb, 64MB (4 Meg * 32 bits * 4 banks)

EPDC

0: No ePDC device.

1: Epson S1D13524

2: Epson S1D13541

2.6 Display Types

Each display design has a unique type identifier. This is based on parameters which define how the display can be used within a system, such as its resolution, signals timing and controller compatibility. The table below summarizes all the display types currently supported by the reference software:

Type	Resolution	Active area	Pixel density	Controller	EEPROM
Type11	1280x960	10.7"	150ppi	None	24C256
Type16	320x240	4.7" / 2.7"	85ppi / 150ppi	S1D13541	24C256
Type18	400x240	4.0"	115ppi	S1D13541	24C256
Type19	720x120	4.9"	150ppi	S1D13541 (COP)	None

2.7 Display Data

Each display has a corresponding set of data, which needs to be known to the control system. This data can be stored in any non-volatile memory available on the system, and some displays include an EEPROM for that purpose.

display identifier

Each display produced in the factory has a unique serial number. This is stored as a string and its format is not currently documented.

display type

The display type (see Display Types) is a string to identify the display design. Its main purpose is to let the control system configure itself appropriately, especially in terms of resolution and timing constraints for the data signals.

VCOM voltage

This is the VCOM voltage determined as part of the manufacturing process. See VCOM Calibration for more information about its purpose and how it is used by the control system.

waveform library

This usually takes the form of either a set of files or a binary blob with the waveform data in a format compatible with the control system.

2.7.1 Display Data File Layout

Whenever represented as a file system, the display data is organised as follows:

id:	String with the display identifier.
type:	String with one of the Display types.
waveform.bin:	Binary waveform library in a suitable format for the control system.
waveform.bin.md5:	String with the MD5 sum of the binary waveform library.
voltage/vcom:	String with the VCOM voltage im mV.

This file layout can be used on an SD card or in a display EEPROM using the SquashFS file system.

2.7.2 Display EEPROM binary format v1

A few general conventions:

- Numbers are stored in big-endian format.
- Strings are always null-terminated, so for a storage length of 64 bytes the maximum string length is 63 effective characters.
- The version number needs to be incremented each time a new format is published.
- The CRC16 is computed using the reference implementation as published by Plastic Logic. See pldata.crc16 for mode details.
- The LZSS compression algorithm is used following the reference implementation as published by Plastic Logic. It takes two parameters: `ei` and `ej`. These need to match during compression and decompression. See lzss and Python lzss module for more details.

The display EEPROM binary format v1 is defined below:

Offset	Size	Field name	Description
vermagic			
0	4	SIGNATURE	Magic number: 0x46574C50 or "PLWF" in ASCII.
4	2	VERSION	Unsigned 16-bit integer set to 1.
info			
6	64	DISPLAY_ID	String with the display identifier.
70	64	DISPLAY_TYPE	String with one of the Display types.
134	4	VCOM	Signed 32-bit integer with the VCOM voltage as measured with an ideal VGSWING which varies with the design being used.
138	16	WVF_MD5	MD5 checksum of the original waveform data as a 128-bit integer. This can also be used as a unique identifier of the waveform library.
154	4	WVF_FULL_LENGTH	Full length of the original uncompressed waveform data as an unsigned 32-bit integer.
158	4	WVF_DATA_LENGTH	Length of the compressed waveform data as an unsigned 32-bit integer.
162	64	WVF_ID	String with the waveform identifier generated during manufacturing.
226	64	TARGET	String with the name of the target controller type (S1D13541, S1D13524...)
290	2	INFO_CRC	CRC-16 checksum of the **info** section (offsets 6 to 290).
waveform			
292	N	WVF_DATA	Waveform library data compressed with the LZSS algorithm and following parameters: ei=7, ej=4. The length N is stored in the `WAVEFORM_DATA_LENGTH` field.
292+N	2	WVF_DATA_CRC	CRC-16 checksum of the **waveform** section (offsets 292 to 292+N).

See Programming EEPROMs for instructions on how to store this data in an EEPROM device.

2.8 Programming EEPROMs

The table below contains the I2C and EEPROM options corresponding to various hardware configurations:

EEPROM device	Mode	Size	I2C address
Standard display (see Display Data and Display Types)	24c256	32kB	0x54 (or 0x55)
Z6-Z7 boards (VCOM calibration data)	24c01	128B	0x50
Ruddock v2 runtime options	24c01	128B	0x52

Note: The I2C bus used on BeagleBone is always `/dev/i2c-3`, unless the Epson S1D13541 (display with a Module B v2.2) or the S1D13524 (Raven board) is configured to act as an SPI-I2C bridge in which case the I2C bus is usually `/dev/i2c-5`.

The plhwtools application, part of the PLSDK, allows to programme I2C EEPROMs on Linux systems from the command line. It can also be used to read the data back from EEPROMs. For full details, see the eeprom command documentation.

Sample commands:

```
# To programme the display EEPROM on a BeagleBone + HB Z6
plhwtools -b /dev/i2c-3 -a 0x54 eeprom 24c256 f2e display-blob.bin

# To programme the factory VCOM on a BeagleBone + HB Z6
plhwtools -b /dev/i2c-3 -a 0x50 eeprom 24c01 f2e vcom-blob.bin

# To make a back-up of the display EEPROM on a BeagleBone + HB Z6
plhwtools -b /dev/i2c-3 -a 0x54 eeprom 24c256 e2f display-blob-backup.bin
```

Note: The I2C bus can be stored in the PLSDK Configuration File to simplify `plhwtools` command line options:

```
[libplhw]

i2c-bus=/dev/i2c-3
```

3 Plastic Logic Software Development Kit

3.1 About

The Plastic Logic Software Development Kit contains a series of readily available command-line applications to demonstrate the operation of E-Paper Plastic Logic displays. It is based on portable C libraries which can be used on various hardware architectures, GNU/Linux and Android operating systems.

It also contains a Python package to enable fast development of E-Paper applications and expand the possibilities using standard Python libraries as well as the Python Imaging Library (PIL).

3.2 Applications

The PLSDK contains the following list of command-line applications to exercise all the Plastic Logic hardware.

- eptest: Draw test patterns and run animations on E-Paper.
- epview: Document viewer with full E-Paper support.
- plinput: Input device utility.
- plottest: Handwriting with arbitrary input data on E-Paper.
- plhwtools: Low-level Plastic Logic hardware control.
- lzss: Encode and decode files using the LZSS algorithm.

Note about colours

A colour is either one of the following keywords:

```
black white red green blue cyan magenta yellow
```

or a 24-bit hexadecimal value representing the red, green and blue channels:

```
0x102030
```

for red=16, green=32 and blue=48 in decimal.

Note about ePDC modes

There is currently no standard Linux software interface to control e-paper displays, so the PLSDK has a dynamic module feature to use any e-paper display controller (ePDC) architecture. Each module has a unique name which is then used to look for a `mod_plepaper_EPDC_MODE.so` file to be loaded at runtime. When not specified, applications will automatically detect which module can be used on the platform by iterating through common modes. In order to implement a customer-specific module, just compile a file with the correct symbol and use the same naming convention. Usual ePDC modes are:

noep

do not use any ePDC (built-in, not in a .so file and default when no other ePDC mode can be found)

epson

Epson ePDC family using Plastic Logic kernel drivers (S1D13541, S1D13524)

imx

Freescale i.MX508

plepdc

Plastic Logic generic ePDC interface

Note about waveforms

Waveform options are strings as described in Waveforms.

3.2.1 eptest

The test patterns and animations produced by `eptest` are automatically scaled to the display resolution. They also show examples of how to use the various E-Paper control system specificities such as Waveforms, regions and synchronous updates.

Usage:

```
eptest <OPTIONS> <TEST_NAME> <TEST_OPTIONS>
```

OPTIONS

`-h`	Show help message and exit.
`-v`	Show the version, copyright and license information, and exit.
`-l`	Log the frame buffer and ePDC information.
`-d DEVICE`	Specify the frame buffer device to use. If not specified, the usual devices will be searched (`/dev/fb0`, `/dev/graphics/fb0` etc...).
`-e DEVICE`	Specify the ePDC device to use. This relies on the implementation of the ePDC mode being used.
`-c`	Clear the screen before running the test. If not specified, test patterns will be drawn on top of each other.
`-m EPDC_MODE`	Specify which ePDC mode to use to control the ePDC. By default, this is discovered automatically by probing the system. The corresponding module is then loaded.
`-r ANGLE`	Set the display rotation to `ANGLE` degrees. Valid values are 0, 90, 180 and 270.
`-f CFA`	Enable the colour filter array (CFA) specified by the `CFA` identifier or the default standard `GR_BW` one otherwise. This is useful for colour displays only.
`-O CONFIG_FILE`	Use configuration file with the specified `CONFIG_FILE` path. If not specified, the default path (typically `~/.plsdk.ini` or `/etc/plsdk.ini`) is used.

TEST_NAME and **TEST_OPTIONS**

The table below shows the list of tests with their specific options.

When no TEST_NAME is specified, the default test is `palette`.

TEST_NAME	TEST_OPTIONS	Description
palette		Draw a palette test pattern with all grey levels and colours.
frame		Draw a checkered boarder frame, useful to show the mono waveform and regional updates.
regions		Draw a series of regions with different grey levels. The updates should overlap in time but not in coordinates.
radar	delay, colour	Draw a rotating line, waiting for `delay` milliseconds between each step. The `colour` option is the colour to draw the lines, with the special value `rgbw` which uses the primary colours in turns. The defaults are 10ms and black.
check	dimension, colour, waveform	Draw a checkered board on the full screen, each square having the given `dimension` in pixels and the given colour on white background, and update the display with the given `waveform`. The defaults are the horizontal resolution divided by 40, black squares and refresh waveform.
fill	colour, waveform	Fill the screen with the given colour and perform a display update with the given `waveform`. The default colour is white and the default waveform is refresh.
fastmono		Show an animation using the "fast/mono" waveform (only available on a limited number of platforms).
highlight		Similar to `fastmono`, show an animation using the "highlight/4GL" waveform.
text	foreground, background, message	Draw several pieces of text using built-in fonts, or a given `message`. The default colour values for `foreground` and `background` are black on white.
none		Do not run any test. Useful when only testing the libraries initialisation.
update	waveform	Update the display with the given `waveform`. The default is refresh.

3.2.2 epview

Usage:

```
epview <OPTIONS> [FILE ...]
```

Show the documents contained in the `FILE...` files with full e-paper support. If no `FILE` argument is provided, data is read from stdin and the `-t` option must be set. It is not possible to mix stdin with file names, and the `-j` (loop) and `-g` (pause) options are not supported when using `stdin`.

OPTIONS

`-h`	Show help message and exit.
`-v`	Show the version, copyright and license information, and exit.
`-l`	Log the frame buffer and ePDC information.

`-d DEVICE`	Specify the frame buffer device to use. If not specified, the usual devices will be searched (`/dev/fb0`, `/dev/graphics/fb0` etc...).
`-e DEVICE`	Specify the ePDC device to use. This relies on the implementation of the ePDC mode being used.
`-c`	Clear the screen before running the test. If not specified, test patterns will be drawn on top of each other.
`-m EPDC_MODE`	Specify which ePDC mode to use to control the ePDC. By default, this is discovered automatically by probing the system. The corresponding module is then loaded.
`-r ANGLE`	Set the display rotation to `ANGLE` degrees. Valid values are 0, 90, 180 and 270.
`-f CFA`	Enable the colour filter array (CFA) specified by the `CFA` identifier or the default standard `GR_BW` one otherwise. This is useful for colour displays only.
`-p`	Wait for the display power to be turned off after the display update.
`-s`	Enable synchronous mode to wait for the new display update to have startd.
`-j`	Keep showing the documents in an infinite loop (ignored when reading from `stdin`).
`-g PAUSE`	Wait for `PAUSE` milliseconds before showing the next document. The default is 2000 for 2 seconds (ignored when reading from `stdin`).
`-W`	Enumerate the available Waveforms, do not show any document and exit.
`-w WAVEFORM`	Specify the waveform path to use with `WAVEFORM` (see Waveforms).
`-b BACKGROUND`	Fill the background of the screen with the given `BACKGROUND` colour. If not specified, the background is not filled at all.
`-t DOC_TYPE`	Specify the document type DOC_TYPE. This is required when reading data from stdin or when using raw formats. The following document types are currently supported: **png** PNG images
`-o OFFSET`	Start showing the document with the top-left corner located in coordinates `OFFSET_X`, `OFFSET_Y`. By default, the document is shown in the top-left corner of the display (0, 0) - unless a rotation angle is provided.

`-a ALIGNMENT`	Align the image using the vertical and horizontal alignment values in `ALIGNMENT`, separated by a comma. They must be either one or two values from these two sets:
	left, center, right
	top, middle, bottom
	A few examples:
	`left,middle`
	`center,middle`
	`bottom`
	Any offset specified by -o will be ignored if -a conflicts with it.

`-c CROP`	Crop the input image to only show a part of it using `LEFT,TOP,RIGHT,BOTTOM` coordinates for `CROP`. These values are relative to the image, not the screen. The offset of the pixels being shown doesn't change when a crop area used.
`-O CONFIG_FILE`	Use configuration file with the specified `CONFIG_FILE` path. If not specified, the default path (typically `~/.plsdk.ini` or `/etc/plsdk.ini`) is used.

3.2.3 plinput

Usage:

```
plinput <OPTIONS>
```

Listen for input events on a given input device and print them out.

To be completed in a future version

3.2.4 plottest

Usage:

```
plottest <OPTIONS>
```

Listen for input events from a given mouse-like input device and draw thin lines on the screen between the received coordinates.

To be completed in a future version

3.2.5 plhwtools

Usage:

```
plhwtools <OPTIONS> <COMMAND_NAME> <COMMAND_ARGUMENTS>
```

OPTIONS

`-h COMMAND`	Show the help message and exit. If `COMMAND` is provided, only show the help message for the given command.
`-v`	Show the version, copyright and license information and exit.
`-b I2C_BUS_DEVICE`	Specify the I2C bus device to be used, typically `/dev/i2c-X` where X is the I2C bus number.
`-a I2C_ADDRESS`	Specify the I2C address of the device to be used with the command. This only applies to commands that use a single I2C device.
`-o COMMAND_OPTIONS`	Optional argument string which can be used by the command. Please see each command help for more specific details.

COMMAND_NAME

cpld

Control Plastic Logic CPLD over I2C register interface

max17135

Control MAX17135 HV-PMIC (timings, switches, VCOM register)

tps65185

Control TI65185 HV-PMIC (timings, switches, VCOM register)

dac

Control MAX5820 DAC power and register value

adc

Read MAX11607 ADC values

pbtn

Push button test procedure using I2C GPIO expander

eeprom

Read/write/test display EEPROM

power

Run full power on/off sequence using multiple devices

epdc

Access to low-level EPD control system interface

COMMAND_ARGUMENTS vary with each command, see detailed description below.

`cpld` ARGUMENTS

When called with no arguments, the CPLD firmware version and board identifier are displayed. Not all Plastic Logic hardware systems have a compatible CPLD. Currently, only the **Hummingbird Z3.1** does, which is a **Freescale i.MX5** EVK expansion board. All the following switches accept an optional extra argument to set their state to `on` or `off`. If no extra argument is provided, then their current state is printed on stdout.

Switches:

hv

display high-voltage supply enable

`gate`

gate drivers VDD enable

vcom_close

VCOM switch close (`off` to open, `on` to close)

vcom_psu

VCOM power supply enable

bpcom_clamp

BPCOM clamp enable

Other:

version

Print the CPLD version as a plain decimal number on stdout.

Examples:

```
plhwtools cpld version
plhwtools cpld hv on
plhwtools cpld hv off
```

max17135 ARGUMENTS

This command can be used to control a Maxim MAX17135 HV-PMIC. With no arguments, all the HV-PMIC status information is logged. The following arguments can be used:

timing TIMING_NUMBER TIMING_VALUE_MS

set a timing register `TIMING_NUMBER` value to `TIMING_VALUE_MS` in milliseconds

timings TIMINGS

set all the timings (8 of them) in one command

vcom [VCOM_REGISTER_VALUE]

set the VCOM register value (0-255) or print out the current value

[en, cen, cen2] [on, off]

switch on/off the HV power supplies (`en`, `cen` or `cen2`)

> ***Caution!***
>
> It is important to respect the power sequence specified in the display source and gate drivers; not doing so can damage the display. Use with care.

Examples:

```
# query the current state of the chip
plhwtools max17135

# set the VCOM value, power up and down sequence timings and turn the HV on
plhwtools max17135 vcom 80
plhwtools max17135 timings 12 7 2 23 2 14 12 7
plhwtools max17135 en on
```

```
# query the state of the chip again and turn the HV off
plhwtools max17135
plhwtools max17135 en off
```

tps65185 ARGUMENTS

This command can be used to control a TI TPS65185 HV-PMIC. With no arguments, all the HV-PMIC status information is logged. The following arguments can be used:

`vcom [VCOM_REGISTER_VALUE]`

set the VCOM register value (0-511) or print out the current value if no `VCOM_REGISTER_VALUE` is provided

`seq [up, down] [VDDH VPOS VEE VNEG STROBE1 STROBE2 STROBE3 STROBE4]`

set the power up or down sequence timings; each voltage (`VDDH..VNEG`) takes a strobe number between 1 and 4, and each strobe (`STROBE1..STROBE4`) is a delay of either 3, 6, 9 or 12 milliseconds

`active`

set the power mode to "active", which turns the HV on and waits for the sequence to be completed by polling the HV-PMIC ENABLE.ACTIVE register bit

`standby`

set the power mode to "standby", which turns the HV off and waits for the sequence to be completed by polling the HV-PMIC ENABLE.STANDBY register bit

`en [vgneg, vee, vpos, vddh, vcom, v3p3] [on, off]`

set an individual power rail enable status to on or off, or log the current state if no on or off value is provided

> ***Caution!***
>
> It is important to respect the power sequence specified in the display source and gate drivers; not doing so can damage the display. Use with care.

Examples:

```
# query the current state of the chip
plhwtools tps65185

# set the VCOM value, power up sequence timings and turn the HV on
plhwtools tps65185 vcom 150
plhwtools tps65185 seq up 1 2 3 4 3 6 9 12
plhwtools tps65185 active

# check the new state of the VEE power rail
plhwtools tps65185 en vee

# disable only VEE
plhwtools tps65185 en vee off
```

```
# turn the HV off (standby mode)
plhwtools tps65185 standby
```

dac CHANNEL ACTION

CHANNEL

either A or B to select the channel

ACTION

either a value between 0 and 255 to write to the DAC register and change the output voltage, or one of the following power control commands:

on

turn the power on

off

turn the power off and let the output floating

off1k

turn the power off and pull the output to GND with 1K

off100k

turn the power off and pull the output to GND with 100K value between 0 and 255: set the output of the given channel

Examples:

```
plhwtools dac A on
plhwtools dac A 128
```

adc REFERENCE CHANNEL

With no arguments, the default reference voltage is used and all channels are converted to volts and displayed. When a reference voltage is specified but no channel is selected, then all the channels are shown. When a channel is selected, then a plain floating point voltage is printed on stdout.

REFERENCE

The reference voltage can be any of the following:

internal

use the internal reference voltage (default)

external

use the external reference voltage

vdd

use VDD as reference voltage

CHANNEL

The channel can be selected using a number starting from 0, or the special value `vcom` can be used with the standard VCOM channel number.

Examples:

```
plhwtools adc
```

pbtn

No arguments, just a small procedure to manually test buttons switches on an I2C GPIO expander.

eeprom MODE COMMAND

This command requires 2 arguments: `MODE` and `COMMAND`. The `MODE` is to specify which kind of EEPROM is being used. Typical values with Plastic Logic hardware are `24c256` for 32 kB display EEPROMs and `24c01` for 128 bytes factory VCOM EEPROMs. Then the `COMMAND` argument is the action to perform with the EEPROM.

MODE

The following EEPROM modes are supported:

```
24c01 24c02 24c08 24c16 24c32 24c64 24c128 24c256 24c512 24c1024
```

COMMAND

The following commands are supported:

full_rw

write random data, read it back and compare

e2f FILE_NAME

dump EEPROM contents to a file, or stdout by default

f2e FILE_NAME

dump file contents or stdin by default to EEPROM

COMMAND_OPTIONS

The EEPROM command options use the following format:

```
-o opt1=value1,opt2=value2
```

The options are:

i2c_block_size=SIZE

size of the I2C transaction blocks, 96 by default and can be set higher for increased performance if the underlying I2C driver supports bigger blocks

page_size=SIZE

size of an EEPROM page, if not standard

zero_padding

with `f2e`, pad the rest of the EEPROM with zeros

skip=LENGTH

skip LENGTH bytes when either reading or writing.

data_size

maximum size of the EEPROM to use, instead of full capacity

addr=CONFIG

configuration key to look for the EEPROM I2C address in the configuration file; typical values are `eeprom-i2c-addr-display` and `eeprom-i2c-addr-vcom`

Example Configuration File settings:

```
[libplhw]
```

```
i2c-bus=/dev/i2c-5

[plhwtools]

eeprom-i2c-addr-display=0x54
eeprom-i2c-addr-vcom=0x50
```

Examples:

```
plhwtools -a 0x50 eeprom 24c256 f2e blob.bin
plhwtools -a 0x52 eeprom 24c01 full_rw
plhwtools -a 0x54 eeprom 24c256 e2f eeprom-dump.bin
plhwtools -o addr=eeprom-i2c-addr-display eeprom 24c256 e2f display-dump.bin
```

`power` ARGUMENTS

`on [seq] [vcom]`

turn the power on, with optional sequence name (seq0 by default) and optional HV-PMIC VCOM value (0-255 using the MAX17135 HV-PMIC)

`off [seq]`

turn the power off, with optional sequence name (seq0 by default)

Examples:

```
plhwtools power on seq0 102
plhwtools power off
```

epdc

This command is used to access the low-level interface to electrophoretic display controllers (ePDC) via the PLSDK `libplepaper` library.

`opt OPT [VALUE]`

Set a hardware option `OPT` to the given numerical `VALUE` or print its current value if none. Supported option identifiers for `OPT` are:

`power_off_delay_ms`

delay in milliseconds between end of display update and display HV power off

`clear_on_exit`

clear the screen when the ePDC is shut down

`temperature_auto`

enable automatic temperature handling

`temperature`

temperature value in degrees Celcius when using manual mode

Examples:

```
plhwtools epdc opt clear_on_exit 1
plhwtools epdc opt temperature_auto 0
plhwtools epdc opt temperature 12
```

3.2.6 lzss

Usage:

```
lzss <OPTIONS> COMMAND INPUT_FILE OUTPUT_FILE
```

OPTIONS

`-h`	Show the help message and exit.
`-v`	Show the version, copytight and license information, and exit.
`-i N`	Set the compression buffer size parameter. The size in bytes is `2^(N + 1)` although all not values are valid. The default is 10 (2048 bytes).
`-j N`	Set the reading buffer size parameter. This is not a working buffer but the read-only input data examined to produce the compressed data. The size in bytes is `(2^N) + 1`. The default is 4 (17 bytes).

> ***Important***
>
> The `-i` and `-j` parameter values are not stored in the output file, and they must match between encoding and decoding. So if the standard values are not used when encoding the data, the same options need to be passed when decoding it again.

COMMAND

e

Encode the `INPUT_FILE` and store the encoded data in `OUTPUT_FILE`.

d

Decode the `INPUT_FILE` and store the original data in `OUTPUT_FILE`.

test

Run an in-memory test and print the result.

Examples:

```
lzss test
lzss e original encoded.lzss
lzss d encoded.lzss new-original
lzss e -i7 -j4 original encoded-smaller-buffer.lzss
lzss d -i7 -j4 encoded-smaller-buffer.lzss new-original
```

3.3 Configuration File

The PLSDK can use a configuration file to provide system-wide settings. It is in a key-value format and stored in the user home directory or the global settings directory. Typical locations are:

```
~/.plsdk.ini
/etc/plsdk.ini
/system/etc/plsdk.ini
```

Unless a specific configuration file is specified, the PLSDK libraries will automatically look first for `~/.plsdk.ini`, then `/etc/plsdk.ini`.

Each section of the file corresponds to a PLSDK library or application. A basic template file `plsdk.ini` is included with the source distribution. This is a short example of how it can be used:

```
# Plastic Logic Software Development Kit - Sample configuration file

[libpldraw]

# Use second frame buffer device (default is fb0)
fb=/dev/fb1

# Rotate the screen by 90 degrees clockwise
rotation=90


[libplhw]

# Default BeagleBone I2C bus to access the hardware devices from user-land:
i2c-bus=/dev/i2c-3


[plhwtools]

# Standard Type18 EEPROM I2C address
eeprom-i2c-addr-display=0x54

# Standard Hummingbird Z6/Z7 VCOM PSU calibration EEPROM I2C address
eeprom-i2c-addr-vcom=0x50

# Arbitrary user-defined EEPROM I2C address (any key can be used)
my-arbitrary-i2c-eeprom-address=0x23
```

All currently supported configuration options are described in the table below, for each library or application:

Key	Usual values	Description
libplepaper - E-Paper control library		
mode	noep, epson, imx, plepdc	EPD controller back-end module identifier - see ePDC mode.
libpldraw - Frame buffer drawing library		
fb	/dev/fb0	Path to frame buffer device.
rotation	0, 90, 180, 270	Rotation angle to use for all drawing operations.
cfa	none, GR_BW, GR_BW_TR180	Colour Filter Array itendifier.
font-name	Courier, Lucida	Default font name and size to use with text drawing operations. This uses fonts built statically into the library.
font-size	8, 10, 12, 14, 18, 24	
crop-left	0	The crop margins are expressed in numbers of pixels from each edge of the display. This means the resulting active area dimensions vary with the display resolution.
crop-top	0	
crop-right	0	
crop-bottom	0	
libplhw - Plastic Logic hardware control library		
i2c-bus	/dev/i2c-1	Path to the I2C bus device.
MAX17135-address	0x48	Maxim MAX17135 HV-PMIC I2C address
TPS65185-address	0x68	TI TPS65185 HV-PMIC I2C address
MAX5820-address	0x39	Maxim MAX5820 ADC I2C address
MAX116xx-address	0x34	Maxim MAX116xx DAC I2C address
CPLD-address	0x70	Plastic Logic CPLD I2C address
pbtn-address	0x21	Push-button Philips PCF8574 GPIO expander I2C address

3.4 Development

This section provides instructions to get the PLSDK source code from public repositories, build it and install it. For some complementary information, please see PLSDK on Github

It also deals with how the PLSDK applications can be written in C and in Python, and how new features can be added to it.

3.4.1 Getting the source code

The PLSDK relies on `repo` and `git` to manage all its source code. First you will need to install `git` and potentially several other packages depending on your Linux distribution. On Debian or Ubuntu systems, run this command:


```
$ sudo apt-get install git
```

Then to install `repo`, follow the instructions on the Android Developers page. In a nutshell:

```
$ mkdir ~/bin
$ PATH=~/bin:$PATH
$ curl https://dl-ssl.google.com/dl/googlesource/git-repo/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
```

Then to get the latest source code from Github:

```
$ mkdir -p ~/src/plsdk
$ cd ~/src/plsdk
$ repo init -u git://github.com/plasticlogic/manifest.git
$ repo sync
```

To update your source code in the future and keep up-to-date, run:

```
$ repo sync
```

Alternatively, to get a specific release number, for example plsdk-011:

```
$ repo init -m plsdk-011.xml
$ repo sync
```

3.4.2 Building for GNU/Linux

To cross-compile for ARM, a third-party toolchain needs to be installed first. The a few environment variables need to be defined, which can easily be achived with these scripts:

```
$ . plsdk/fast-builder/envsetup.sh # for cross-compiling, adjust paths
$ . plsdk/fast-builder/native-envsetup.sh # for native build
```

Then to build:

```
$ make -j5 # adjust -j for the number of parallel builds
```

The output files are located in the following directories ($ARCH is the target architecture name, typically `arm` or `x86_64`):

```
out/$ARCH/bin # install in /usr/bin on the target
out/$ARCH/lib # install in /usr/lib on the target
out/$ARCH/include # public C header files, install in /usr/include
out/$ARCH/python2.7 # depends on your target Python version, in /usr/lib/
```

3.4.3 Building for Android

Building for Android is achieved with the Android Native Development Kit (NDK). Instructions to install it can be found on the Android NDK page.

A different repo manifest file `android-ndk.xml` needs to be used to build with the Android NDK. If the source tree has not yet been initialised, run:

```
repo init -u git@github.com:plasticlogic/manifest.git -b master -m android-ndk.xml
```

Otherwise, to simply switch to the other manifest file:

```
repo init -m android-ndk.xml
rm -rf plsdk
```

Then to actually update the source tree:

```
repo sync
```

Then to build the source code, run:

```
$ ndk-build -j5
```

The output files can be found in the `libs/armeabi/` directory.

3.4.4 E-Paper modules: `mod_plepaper`

The `libplepaper` library uses run-time plug-in modules which are compiled in the form of shared objects (`.so`) files. Each module implements a back-end interface to control a particular kind of E-Paper controller. For example, the PLSDK standard back-ends include one for the i.MX5 integrated controller and one for the Epson S1D135x1.

An exception to this is the built-in `noep` mode, which doesn't use any real E-Paper controller. It can be used for tests or simulation on non-E-Paper displays.

Creating a new back-end module

Arbitrary modules can be created and used without modifying the `libepepaper`. This can be achived with the following steps:

1. start with an empty C file in the PLSDK source tree, for example `mod_plepaper/mod_plepaper_example`
2. include `<libplepaper.h>`
3. add a line with `PLEPAPER_VERSION;` which is used for run-time binary compatibility checks
4. implement functions to use your E-Paper controller such as `update`, `set_opt`, `get_opt`, `free`, `wait_power`, `log_info`
5. define a waveform description look-up table
6. write a `PLEPAPER_MOD_INIT(epdc, dev)` function which will be called by the `libepaper` library when the module is loaded to allocate any private resource and populate the `epdc` abstract API structure with function pointers and other parameters
7. compile into `mod_plepaper_example.so`
8. install on the target system in the standard library path

Waveform description look-up table

This is an array of `struct epdc_wfdef` structures, each containing the information for a waveform type available in the E-Paper controller waveform library:

```
struct epdc_wfdef {
        const char *type;           /**< waveform type */
        const char *palette;        /**< waveform palette */
        const char *opt;            /**< waveform option */
        int epdc_id;                /**< waveform ID */
        unsigned duration_ms;       /**< waveform duration in ms */
};
```

This is related to the Waveforms paths defined with `type/palette/opt`. A pointer to this table needs to be set in the `epdc->wflist` interface to let the `plep_get_wfid` function search for a matching entry and pass it to `epdc->update`. The `epdc_id` can be set to any arbitrary value to uniquely identify the waveform at the controller level. The `duration_ms` is used when the module doesn't support synchronous updates and make the sleep instead.

This is the example table defined in `epdc-noep.c`:

```
static struct epdc_wfdef g_wflist[] = {
        { PLEP_REFRESH,     NULL,       NULL,           0, 1000 },
        { PLEP_REFRESH,     PLEP_MONO,  NULL,           1,  500 },
        { PLEP_DELTA,       NULL,       NULL,           2, 1000 },
        { PLEP_DELTA,       PLEP_MONO,  NULL,           3,  500 },
        { PLEP_FAST,        PLEP_MONO,  PLEP_3SUBF,     4,   80 },
        { PLEP_FAST,        PLEP_MONO,  PLEP_6SUBF,     5,  140 },
        { PLEP_FAST,        PLEP_MONO,  PLEP_9SUBF,     6,  200 },
        { PLEP_HIGHLIGHT,   PLEP_4GL,   NULL,           7,   80 },
};
```

The **order of the entries** in the table matters as the first matching waveform will be used. When looking up for a `refresh` waveform, the first entry will be used in this case. If the first two entries were swapped, that is to say `refresh/mono` was first and `refresh` second, then the `refresh/mono` would be picked instead. So as a rule, the waveform identifiers with `NULL` should be placed before those with defined variants: ``refresh` should be before `refresh/mono`, which should also be placed before `refresh/mono/some_option` if it existed.

Using the new module

The module should now be ready to be used with the `example` mode name. The main usage is to create a `plep` object to control the E-Paper:

```
struct plep *plep = plep_init(NULL, "example", NULL);
```

With standard PLSDK applications, use the `-m` option to specify the E-Paper mode:

```
eptest -m example
```

3.4.5 Python

The PLSDK contains a Python package `plsdk`, see the separate Python PLSDK manual for the full API documentation. This can also be used in conjunction with the Python Imaging Library as included in the PLSDK external components. The following example shows the main features of the Python `plsdk`:

```
from PIL import Image
import plsdk

# Create a Display object
d = plsdk.Display()

# Clear the screen
d.clear()

# Use mono waveform
d.wf = d.ep.mono

# Fill a rectangle with black
d.rectangle(((d.fb.xres / 4), (d.fb.yres / 4),
             (d.fb.xres * 3 / 4), (d.fb.yres * 3 / 4)))

# Disable automatic updates
d.auto_update = False

# Draw diagonal lines across the screen
d.line((0, 0, (d.fb.xres - 1), (d.fb.yres - 1)))
d.line(((d.fb.xres - 1), 0, 0, (d.fb.yres - 1)))

# Update now
d.update()

# Show an image in the top-left corner using special PIL fbdump method
Image.open("my-picture.jpeg").convert("RGB").fbdump(d.fb)

# Update the display with a refresh waveform
d.update(wf=d.ep.refresh)

# Enable CFA for a colour display
d.fb.cfa = d.fb.CFA_GR_BW

# Show a colour picture
Image.open("my-colour-picture.jpeg").convert("RGB").fbdump(d.fb)

# Update the display with a refresh waveform
d.update(wf=d.ep.refresh)
```

3.5 Advanced Operations

This section contains instructions to perform actions that are not typically needed when using the Plastic Logic evaluation kits but can be useful in some specific cases.

`pldata`

The PLSDK comes with a `pldata` component dedicated to generating Plastic Logic data structures, such as EEPROM binary blobs, LZSS compression, checksum computations etc... This is automatically built with the other PLSDK applications and libraries, but it can also be used as a stand-alone component as it has no dependencies other than standard C and Python v2.7 libraries.

Note: The Python scripts used in the examples below assume the script is located in the current directory, to keep the syntax simple. On a PL evaluation kit, they are typically stored in `/usr/lib/python2.7/pldata` so they can be called this way for example:

```
python /usr/lib/python2.7/pldata/some_script.py
```

If you don't want to get or build the complete PLSDK source code, you can just clone the `pldata` repository from Github:

```
git clone https://github.com/plasticlogic/pldata.git
```

This repository contains the following items:

liblzss

C library to perform LZSS encoding/decoding; its API is documented in the PLSDK API manual

lzss

command line application to encode/decode data using the LZSS algorithm

Python lzss module

Python module implemented in C to use the liblzss in Python

pldata.crc16

Python module to generate CRC16 checksums; can also be run from the command line

pldata.mkwfblob

Python module to generate binary display EEPROM blob (mainly contains the waveform library and some information about the display module); can also be run from the command line

pldata.mkvcomblob

Python module to generate binary VCOM PSU calibration EEPROM blob; can also be run from the command line

pldata.dump

Python script to be run on the command line and dump the contents of a blob with a known format

3.5.1 Python lzss module

The PLSDK contains a `pldata/lzss` component which itself includes a C library `liblzss`, a C application lzss and a Python module also named `lzss` and implemented as a C Python extension. This is automatically built with the rest of the PLSDK, but it can also be built and installed independently using standard Python `distutils` and the `setup.py` script. To install it on any operating system with Python installed and `distutils`, run the following commands (as `root` user or using `sudo`):

```
cd pldata/lzss/python
python setup.py install
```

You can then run the included test script to check it all works as expected:

```
python test.py
```

This is a small sample code to read the contents of a file and compress it in a memory buffer:

```
import lzss

data = open('file_name', 'rb').read()
compressed = lzss.encode(data)
```

3.5.2 pldata.crc16

Command line usage:

```
python crc16.py [-h] [--init INIT] [--poly POLY] [--check CHECK] file
```

Compute CRC16 checksum and print it out.

positional arguments:

file

Input file path

optional arguments:

`-h, --help`	show this help message and exit
`--init INIT`	Initial CRC value (16-bit hex)
`--poly POLY`	Polynomial (16-bit hex)
`--check CHECK`	Value to check against (16-bit hex)

crc16 module

The module contains only one useful function to be used by other scripts: `crc16.compute`. Below is a sample code to compute the checksum of the contents of a file:

```
import crc16

data = open('file-name', 'rb').read()
crc = crc16.compute(data)
```

3.5.3 pldata.mkwfblob

Produce a Plastic Logic display EEPROM binary blob following the specified Display EEPROM binary format.

> ***Attention!***
>
> This section contains instructions to perform actions that are outside the scope of normal operation of the Plastic Logic evaluation kits. However, they are provided here for remote maintenance and to enable specific product variants of the Plastic Logic reference designs.
>
> Normal display usage does not require to generate display EEPROM blobs as they are normally generated by Plastic Logic for each display module as part of the manufacturing process. Use with

care, and make sure you create a back-up of the original display EEPROM contents (see sample plhwtools commands) before replacing it with your own generated one.

Command line usage:

```
python mkwfblob.py [-h] [--debug] [--output OUTPUT] [--check CHECK]
                   display_id display_type vcom waveform waveform_id target
```

positional arguments:

display_id
display identifier

display_type
display type

vcom
VCOM voltage in mV

waveform
path to the binary waveform library

waveform_id
waveform identifier

target
target ePDC controller

optional arguments:

`-h, --help`	show this help message and exit
`--debug`	enable debug messages
`--output OUTPUT`	output file, default is display-blob.bin
`--check CHECK`	only check agains existing blob file

Example with dummy values:

```
python mkwfblob.py "1234657890" Type18 4500 waveform.bin "WvfId" S1D13541
python mkwfblob.py "1234657890" Type18 4500 waveform.bin "WvfId" S1D13541 \
  --check=display-blob.bin
```

mkwfblob module

The `mkwfblob.build_blob` function can be used in other Python scripts. It produces a pldata.blob.Blob instance, which can then be used to pack or unpack the data. For example, to create a display EEPROM blob:

```
from pldata import mkwfblob

sample_display_data = {
  'display_id': "DisplayIdentifier 1234567890",
  'display_type': "Type18",
  'vcom': 4500,
```

```
  'target': "S1D13541",
  'wvf_id': "TestWaveformIdentifier ABCDXYZ",
  'wvf': open("waveform.bin", 'rb').read(),
}

blob_data = mkwfblob.build_blob().pack(sample_display_data)
open('display-eeprom.bin', 'wb').write(blob_data)
```

3.5.4 pldata.mkvcomblob

Produce PL flat VCOM calibration EEPROM binary blob following the specified VCOM and Hardware EEPROM format.

> ***Attention!***
>
> Normal usage of Plastic Logic reference hardware does not require to generate VCOM EEPROM blobs as EEPROMs are already programmed on the evaluation kit hardware. Use with care, and make sure you create a back-up of the original VCOM PSU EEPROM contents (see sample plhwtools commands) before replacing it with your own generated one.
>
> When implementing a specific product design, this may be needed in order to generate the VCOM PSU calibration data binary blob, which may be stored in any persistent memory available on the system.

Command line usage:

```
python mkvcomblob.py [-h] [--debug] [--output OUTPUT] [--check CHECK]
                     [--vcom_mode VCOM_MODE] [--hv_pmic HV_PMIC]
                     [--vcom_dac VCOM_DAC] [--vcom_adc VCOM_ADC]
                     [--io_config IO_CONFIG] [--i2c_mode I2C_MODE]
                     [--temp_sensor TEMP_SENSOR] [--frame_buffer FRAME_BUFFER]
                     [--epdc EPDC] [--adc_scale_1 ADC_SCALE_1]
                     [--adc_scale_2 ADC_SCALE_2]
                     dac_x1 dac_y1 dac_x2 dac_y2 vgpos vgneg ideal_vgswing
                     board_type board_ver_maj board_ver_min
```

positional arguments:

dac_x1
: first DAC register value (hexadecimal)

dac_y1
: VCOM voltage associated with dac_x1 in mV

dac_x2
: second DAC register value (hexadecimal)

dac_y2
: VCOM voltage associated with dac_x2 in mV

vgpos
: VGPOS voltage in mV

vgneg
VGNEG voltage in mV

ideal_vgswing
Ideal gate swing voltage in mV

board_type
board type

board_ver_maj
board major version number

board_ver_min
board minor version number

optional arguments:

`-h, --help`	show this help message and exit
`--debug`	enable debug messages
`--output OUTPUT`	output file, default is vcom-blob.bin
`--check CHECK`	only check against existing blob file
`--vcom_mode`	VCOM calibration mode
`--hv_pmic`	HV-PMIC part identifier
`--vcom_dac`	VCOM DAC part identifier
`--vcom_adc`	VCOM ADC part identifier
`--io_config`	configuration of the I/O signals
`--i2c_mode`	I2C mode
`--temp_sensor`	temperature sensor chip identifier
`--frame_buffer`	frame buffer memory identifier
`--epdc_ref`	ePDC part reference identifier
`--adc_scale_1`	dividend of the ratio between VGSWING voltage and that measured by the ADC
`--adc_scale_2`	divisor to be used with ADC_SCALE_1

Example with dummy values for a 9-bit VCOM DAC and Hummingbird Z6:

```
python mkvcomblob.py 0x7F 4200 0x17D 12500 25000 -32000 3333 hbz6 6 3 --vcom_mode 1 --hv_pmic 2
      --vcom_dac 3 --vcom_adc 0 --io_config 1 --i2c_mode 2 --temp_sensor 0
      --frame_buffer 0 --epdc 2 --adc_scale_1 1 --adc_scale_2 1 --output hbz6-1.blob
python mkvcomblob.py 0x7F 4200 0x17D 12500 25000 -32000 3333 hbz6 6 3 --vcom_mode 1 --hv_pmic 2
      --vcom_dac 3 --vcom_adc 0 --io_config 1 --i2c_mode 2 --temp_sensor 0
      --frame_buffer 0 --epdc 2 --adc_scale_1 1 --adc_scale_2 1 --output hbz6-1.blob --check=vcom-blob.bin
```

mkvcomblob module

The following example creates the same binary blob with a python script, using the `mkvcomblob.build_blob` function:

```
from pldata import mkvcomblob

sample_vcom_psu_data = {
    'dac_x1': 0x7F,
    'dac_y1': 4200,
    'dac_x2': 0x17D,
```

```
    'dac_y2': 12500,
    'vgpos': 25000,
    'vgneg': -32000,
    'ideal_vgswing': 5432,
    'board_type': "parrot",
    'board_ver_maj': 1,
    'board_ver_min': 2,
    'vcom_mode': 1,
    'hv_pmic': 2,
    'vcom_dac': 3,
    'vcom_adc': 1,
    'io_config': 1,
    'i2c_mode': 2,
    'temp_sensor': 0,
    'frame_buffer': 0,
    'epdc_ref': 1,
    'adc_scale_1': 1,
    'adc_scale_2': 1
    }

blob_data = mkvcomblob.build_blob().pack(sample_vcom_psu_data)
open('vcom-eeprom.bin', 'wb').write(blob_data)
```

3.5.5 pldata.dump

Dump on stdout the contents of a blob in a known format.

Command line usage:

```
python dump.py [-h] [--version VERSION] [--maxlen MAXLEN]
               {display_wf,vcom_psu} file
```

positional arguments:

format {display_wf,vcom_psu}

blob format name

file

path to the blob file

optional arguments:

`-h, --help`	show this help message and exit
`--version VERSION`	optional format version number
`--maxlen MAXLEN`	maximum item dump length (0 for no limit)

Examples:

```
python dump.py display_wf display-blob.bin
python dump.py vcom_psu vcom-blob.bin
```

4 Plastic Logic Linux kernels

See Plastic Logic Linux kernels on Github.

4.1 BeagleBone

Linux kernel for PLDEKs based on BeagleBone.

4.2 Freescale i.MX

Linux kernel for PLDEKs based on Freescale i.MX platforms.

5 Appendices

5.1 Previous EEPROM binary formats

5.1.1 VCOM and Hardware EEPROM format v0

Offset	Size	Field name	Description
Version			
0	1	VERSION	Unsigned 8-bit integer set to 0.
DAC and VCOM PSU			
1	2	DAC_X1	Signed 16-bit integer with the first DAC register value (usually 25% of fully range i.e. 64 for 8-bit DAC).
3	2	DAC_Y1	Signed 16-bit integer with the VCOM voltage in mV associated with DAC_X1.
5	2	DAC_X2	Signed 16-bit integer with the second DAC register value (usually 75% of fully range i.e. 191 for 8-bit DAC).
7	2	DAC_Y2	Signed 16-bit integer with the VCOM voltage associated with DAC_X2.
Gate PSUs			
9	4	VGPOS	Signed 32-bit integer with the VGPOS voltage in mV.
13	4	VGNEG	Signed 32-bit integer with the VGNEG voltage in mV.
Checksum			
17	2	CRC	CRC-16 checksum of all the above data (offsets 0 to 17).