SOFTWARE GUIDE

Display Evaluation Kit

K_MSP430

For

4.0", 4.7", 4.9", and 10.7" Plastic Logic Displays

Part No.: 303002, 303004, 303006, 303010

Containing Part No. 301008 and 301002, 301014 or 301015

Revision 1
February 15th 2018

Revision Status	Date	Author	Reason of Modification
1	15-Feb-2018	RP	Initial Version

1 Table of Contents

2 Introduction

This document provides an introduction and overview to the code intended to be used by customers wishing to drive Plastic Logic displays and associated controller hardware from a single chip microcontroller platform. The project is in active development and feedback is welcomed on new features or issues found in the code or documentation. Please send feedback via your sales/support representative.

3 Scope

This document does not attempt to describe the detailed operation of any particular microcontroller or Epson display controller as this information is readily available, or may require an NDA to disclose. Prior experience with embedded programming is expected and discussion will focus on the specifics of this code base. The code is able to drive a slideshow of pre-rendered images in the PGM file format to a chosen display. The code focusses on functionality and does not pretend to implement best practice for any specific microcontroller. Data transfer speed improvements are planned for subsequent releases. The code attempts to strike a balance between minimizing microcontroller resource usage while preserving portability, good coding practices and the provision of good debug support (e.g. use of assertions).

4 Licensing

The majority of the software in this codebase was written by Plastic Logic and is currently licensed under a restrictive license for early adopters. For the avoidance of confusion: This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. Some key support functionality is provided by third parties who have their own licenses. The third party components are: FatFs – A FAT file-system driver. This is used to access configuration and image data stored on a micro SD card on the reference microcontroller hardware. The license for FatFS can be found here: http://elm-chan.org/fsw/ff/en/appnote.html#license, it is not restrictive. Sample code - This is sample source code made freely available by the microcontroller vendor. The copyright notices vary from source file to source file but are not restrictive other than limiting the use of such processor specific sample code to a given range of processor devices. Please see Appendix A for license text.

5 Glossary

CCS

Code Composer Studio, an integrated development environment from Texas Instruments that can be used to develop code for the MSP430 microcontroller

DAC

Digital to analogue converter. Converts a digitally-encoded value to an analogue signal

EPD

Electrophoretic display. Such displays retain the last image driven to them and only require power to change the image

EPDC

Electrophoretic display controller. A specialized display timing controller required for updating electrophoretic displays. The EPDC is responsible for applying the correct waveform to each pixel, according to the current and target images

FFC

Flexible flat cable (http://en.wikipedia.org/wiki/Flat_Flex_Cable)

GPIO

General-purpose input/output. A user-controllable pin that can be defined at runtime as either an input or an output

Hummingbird Z6

An evaluation board from Plastic Logic that interfaces between a Plastic Logic small display (e.g. S040_T1.1) and a host processor board (e.g. Parrot or Parrot + Ruddock2)

Hummingbird Z7

An evaluation board from Plastic Logic that interfaces between a Plastic Logic bracelet display (e.g. S049_T1.1) and a host processor board (e.g. Parrot or Parrot + Ruddock2)

HVPMIC

High voltage power management IC. A chip that converts a single (typically battery) voltage into the various higher voltages required by the display

I2C

Inter-Integrated Circuit, a standard two-wire multimaster serial bus intended for communication with low-speed peripherals

Mercury board

An interface board from Plastic Logic that connects a Plastic Logic 10.7" display (e.g. D107_T3.1) to a Raven board via a 50-way FFC

MSP430

A low-power microcontroller from Texas Instruments

Parrot board

An evaluation board from Plastic Logic containing a MSP430 microcontroller

PGM

A portable graphics file format

PIL

Python Imaging Library, adds image processing support to Python

PNG

A graphics file format which uses lossless data compression

Raven board

An evaluation board from Plastic Logic that interfaces between a Plastic Logic 10.7" display (e.g. D107_T3.1) and a host processor board (e.g. Parrot or Parrot + Ruddock2)

Ruddock2 board

An evaluation board from Plastic Logic that interfaces between the Parrot board and one of the display interface boards (Raven, Hummingbird Z6/Z7)

S1D13524

An EPD controller chip from Epson, designed for use with displays up to a resolution of 4096x4096 pixels. Also supports color displays

S1D13541

A combined EPD controller chip and source driver from Epson, designed for use with displays up to a resolution of 854x480 pixels

SPI

Serial Peripheral Interface, a standard four-wire serial bus that operates in full duplex mode

USCI

Universal Serial Communication Interface. MSP430 serial communications interface that supports multiple serial communication modes with one hardware module

VCOM

Display-specific common electrode voltage. Each Plastic Logic display is supplied with the correct voltage that must be applied by the control electronics

Waveform

A display-specific data file that defines how the display updates

Z6

See Hummingbird Z6

Z7

See Hummingbird Z7

6 Configuring the Code

The code includes a number of features and demonstrations that can be configured at run time via the use of settings in the config.txt file.

6.1 Configuration of the display interface board type and display type

The following example defines a Raven board with D107_T3.1 display:

```
# Set one of the following to 1 to manually select the platform.
# This will be used if no platform can be discovered at runtime.
# CONFIG_PLAT_RAVEN < Raven board
# CONFIG_PLAT_Z6 < Hummingbird Z6.x board
# CONFIG_PLAT_Z7 < Hummingbird Z7.x board
board CONFIG_PLAT_RAVEN
# Set this to manually specify the display type when it could not be
detected
# at run-time. This is especially useful for displays without an EEPROM
such
# as S049_T1.1. */
display_type D107_T3.1
```

6.2 Configuration of how display-specific data is used

All Plastic Logic displays require display-specific information such as waveform data and VCOM voltage. Some displays contain an EEPROM that can be used to store this information; alternatively the information can be provided on the SD card. The following settings define where the information will be read from:

```
# Each display has a type and some associated data such as a VCOM voltage and
# waveform library. This can either be stored in the display EEPROM or on the
# SD card. The display type may also be manually specified with
# CONFIG_DISPLAY_TYPE.
#
# Set data_source to one of the following values in order to choose where the data
# should be read from:
# CONFIG_DISP_DATA_EEPROM_ONLY, < Only use display EEPROM
# CONFIG_DISP_DATA_SD_ONLY, < Only use SD card
# CONFIG_DISP_DATA_EEPROM_SD, < Try EEPROM first, then SD card
# CONFIG_DISP_DATA_SD_EEPROM < Try SD card first, then EEPROM
data_source CONFIG_DISP_DATA_EEPROM_SD
```

6.3 Configuration of I2C master

A number of components are configured and accessed via I2C. The following setting defines the device used as the I2C master:

```
# Default I2C master mode used with CONFIG_HWINFO_DEFAULT
# I2C_MODE_NONE, /* invalid mode */
# I2C_MODE_HOST, /* use the host */
# I2C_MODE_DISP, /* use SPI-I2C bridge on the display (S1D13541) */
# I2C_MODE_S1D13524, /* use SPI-I2C bridge on the S1D13524 */
# I2C_MODE_SC18IS6XX, /* not currently supported */
i2c_mode I2C_MODE_HOST
```

The code also includes a number of features and demonstrations that can be configured at compile time via the use of preprocessor directives in the config.h file.

6.4 Configuration of how hardware information is used

The Plastic Logic display interface boards (Raven, Hummingbird Z6/Z7) contain an EEPROM that can be used to store board-specific calibration data and other relevant information. The following settings define whether or not the code will use this information and whether or not to use a default if the information is not available:

```
/** Set to 1 to use the VCOM and hardware info stored in board EEPROM */
#define CONFIG_HWINFO_EEPROM 1
/** Set to 1 to use default VCOM calibration settings if HW info EEPROM
data
* cannot be used (either not programmed, or hardware fault, or
* CONFIG_HWINFO_EEPROM is not defined). If set to 0, the system will not be
* able to work without valid EEPROM data. */
#define CONFIG_HWINFO_DEFAULT 1
```

6.5 Configuration of serial interface

A serial interface is supported via the USB port (the Parrot board is fitted with a TUSB3410 USB to serial port controller). Alternatively a FTDI active serial-to-USB cable can be plugged into a pin header on the Parrot board. The code can be configured to route all standard output to the serial port rather than back to the debugger. This allows debug output still to be seen when no debugger is attached. The following setting defines whether stdout and stderr are sent to the serial port or the debugger:

```
/** Set to 1 to have stdout, stderr sent to serial port */
#define CONFIG_UART_PRINTF 1
```

6.6 Power mode demonstration

The following setting can be used to configure a demonstration of power state transitions:

```
/** Set to 1 to use the power state transition demo rather than the
slideshow */
#define CONFIG_DEMO_POWERMODES 1
```

6.7 Pattern demonstration

The following settings can be used to display a checker-board pattern of the specified size:

```
/** Set to 1 to use the pattern demo rather than the slideshow */
#define CONFIG_DEMO_PATTERN 1 /** Not intended for S049_T1.1 displays */
#define CONFIG_DEMO_PATTERN_SIZE 32 /** Size of checker-board */
```

7 Running the Code

Once the code has been configured and built in Code Composer Studio, the resulting binary can be transferred to the Parrot board using the MSP-FET430UIF USB-JTAG programmer. Depending on the configuration, you should now be able to see one of the following: ▪ A slideshow of stock images from the 0:/<Display-Type>/img folder being shown on the display until execution is halted (with or without power sequencing). The slideshow will skip any files that do not have the extension ".pgm"

- A sequence of images defined by the 0:/<Display-Type>/img/slides.txt file
- A checkerboard image

7.1 Error codes

If a fatal error occurs while running the code, the type of error is indicated via the status LED. Specifically the status LED will be flashed on/off a number of times, followed by a delay, after which the pattern will repeat. The error types are as follows (see also assert.h):

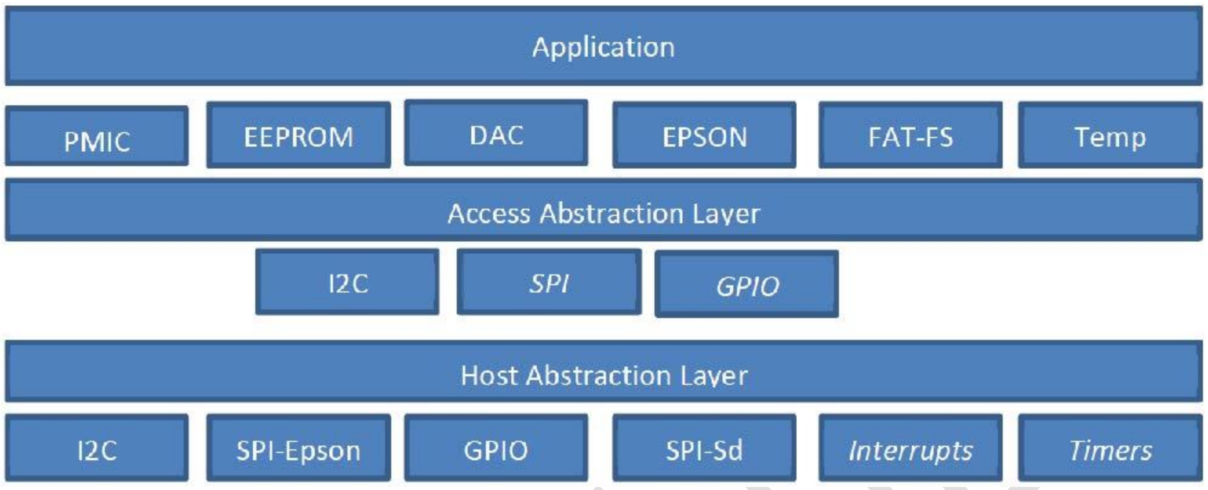
Flashes	**Description**
1	General error initialising GPIO
2	Error initialising MSP430 comms
3	Error reading HWINFO EEPROM. Could be a comms error or a content error
4	Error initialising I2C (Epson)
5	Error reading display information. Could be many errors (comms error, content error, missing or invalid file, etc). Also depends on preprocessor settings
6	Error initialising HVPSU. Most likely to be a comms error, but could indicate a failed PMIC
7	Error initialising EPDC. Could be many errors (comms error, EPDC failure, failed to load init code, failed on one of several commands needed to initialise the EPDC, failed to load waveform, etc)
8	Failed while running application. Multiple causes for this, depending on application that is running. Most likely failures are due to missing/invalid files or hardware problems such as POK or comms failure
9	Failed assert statement (debug use only)
10	Failed to read the config file
0 (off)	Undefined error
0 (on)	No error

Additional information relating to the error can be obtained by inspecting stderr via the debugger or the serial port (depending on how CONFIG_UART_PRINTF has been defined).

8 Code Structure

8.1 Overview

The diagram below shows an overview of the code base.



Things to note are:

1. The **Application** sits right on top of the common components. There is no layer that abstracts a complete display system that can be manipulated by calling methods on it.

2. The **Access Abstraction Layer** exists because the Epson controllers contain a number of resources, e.g. I2C master, SPI master, and on chip GPIOs that the Application layer may want to use. This abstraction layer allows the application to access either a host CPU resource or one contained in the Epson controller without needing to know its location once initialized . Currently only support for I2C is implemented.

3. The **Host Abstraction Layer** allows for porting to different CPUs, either members of the same family or different architectures. Interrupts and Timers are not mandatory for the sample code to work.

8.2 Platform Neutral Components

8.2.1 File System

The micro SD card uses a FAT/FAT16 file system for data storage (not FAT-32). In order to minimize code and data size the FatFs driver is configured to support Read-Only operations, to reuse memory aggressively and not to support long filenames. This has some small impact on access time and transfer speed for the data within files. Long filenames can be used when writing files to the SD card from a PC however the FatFs code can only use the 8.3 compatible filenames. These names can be displayed under Windows by entering dir /x

e.g.:
21/05/2011 07:01 8,863,336 NVWGF2~1.DLL nvwgf2umx.dll

SD Card path	Contents
`0:/<display-type>`	Root of the subtree for the selected display type
`0:/<display-type>/bin/Ecode.bin`	Epson controller initialization file for display type
`0:/<display-type>/img/*.pgm`	Image files to be displayed
`0:/<display-type>/img/slides.txt`	Optional sequence file
`0:/<display-type>/display/vcom`	VCOM voltage for display
`0:/<display-type>/display/waveform.bin`	Waveform for the display (S1D13541)
`0:/<display-type>/display/waveform.wbf`	Waveform for the display (S1D13524)

The VCOM and waveform data for each display should be stored on the display's EEPROM where applicable The Plastic Logic reference code uses the data stored on the EEPROM by default and will search on the SD card if the EEPROM does not contain valid data. This behavior can be changed by modifying the CONFIG_DISP_DATA_xxx preprocessor definitions in the config.txt file. For the best results, it is advisable to use the EEPROM-based data as this is tuned for each display.

8.2.2 EPDC API and Epson implementations

The pl/epdc.h header file defines an abstract interface to an E-Paper Display Controller implementation. There are currently two Epson implementations (S1D13524 and S1D13541), which internally share some overlap. This will generate the appropriate SPI data transfers and control various GPIOs to operate the EPDC. Utility functions provide higher level functions on top of command transfer layer. These functions support initialization code and waveform loading, frame buffer RAM fill, image data transfer and power state transition control.

Note

Epson name the SPI data signals with respect to the controller. Hence DI (DataIn) => MOSI, and DO (DataOut) => MISO. To prepare the controller for operation it is necessary to send two files to it: 1. A controller initialization file which customizes the controller's behavior to the type of display it is going to drive, e.g. resolution, driver configuration, clock timings. 2. A waveform data file which provides display specific timing information required to maximize the performance and image quality of a display.

8.2.3 Epson S1D135xx I2C Interface

The Epson controllers provide an SPI to I2C bridge that can be used to communicate with I2C peripherals instead of using an I2C interface on the host processor. The I2C interface abstraction defined in pl/i2c.h allows higher level software to communicate using either method once an interface has been initialized. The bridge results in a slower overall I2C data rate than a host I2C interface would achieve due to the overhead of communicating over SPI to manage the transfer. However, in normal use the amount of I2C traffic is limited to one-time device configuration.

Note

Some peripherals, the MAXIM 17135 PMIC specifically, have inbuilt timeouts which can be triggered when Epson command tracing is taking place and the Epson I2C bridge is in use.

8.2.4 Temperature Measurement

The accurate measurement of temperature is important to obtain the best image quality from the display. The temperature is used to select the correct waveform used to drive the display. It is common for display updates to take longer at lower temperatures due to the physical attributes of the display media. The S1D13524 and S1D13541 have differing methods of handling temperature measurement. These are exposed in the code as “modes” (pl_epdc_temp_mode in pl/epdc.h):

1. **Manual** – The application software will obtain the temperature from some other component, e.g. the PMIC and pass it to the controller.

2. **Internal** – The display controller will use its built-in temperature sensor, if it has one, to measure the temperature. The S1D13541 controller contains such a temperature sensor, which requires an external NTC thermistor to be fitted (as shown on the Z6 and Z7 reference schematics).

3. **External** – The display controller will communicate directly with an LM75-compatible I2C temperature sensor to obtain the temperature.

To trigger the acquisition or processing of temperature data the controller's update_temp() function is called (either s1d13524_update_temp() or s1d13541_update_temp()). On completion a new temperature will be in effect. On the S1D13541 controller an indication that the waveform data must be reloaded is given if the temperature measured has moved outside the range of the currently cached waveform data. Currently the Internal mode is implemented for the S1D13541 and the External mode is implemented for the S1D13524. The code contains appropriate hooks for implementing the Manual mode if required.

8.2.5 Putting it all Together

The source code contains examples of how to drive a number of different display interface boards. The main.c file contains hardware definitions and the main_init() function which goes through a top-level initialization sequence. This is common to all Plastic Logic reference hardware combinations. It calls functions in probe.c to determine any run-time configuration and initialize the software and hardware accordingly. When porting to a specific product design, typically the main_init() function and associated hardware definitions (i.e. GPIOs) would be tailored to only take care of the hardware features available on the product. The probe.c functions are here mainly for run-time dynamic configuration, which may not be applicable to a fixed and optimized product so initialization function calls may be typically be picked from probe.c and called directly in main_init().

8.3 Host Abstraction Layer

The host abstraction layer isolates the platform neutral code from the underlying platform hardware. The abstraction layers are kept as self-contained and thin as practical. While interrupts and timers are listed their availability is not required to create a working system.

8.3.1 Host GPIO Interface

The GPIO interface provides a way to reserve and define a GPIO pin on the host processor at run time. On small microcontrollers pins are typically either GPIOs or connected to a single special purpose hardware unit e.g. an I2C unit. Some, or all, of the GPIOs supported may be able to generate interrupts. The GPIO interface records which GPIOs are already defined but not the mode in which they are configured. This allows the code to trap errors where a pin is defined multiple times, or used before being defined. GPIO pins are typically used to control the power sequence hardware and manipulate signals in the serial and parallel interface to the Epson controller.

8.3.2 Host I2C Interface

The host I2C interface provides access to an I2C interface physically attached to the host processor. Only a single I2C interface is supported by the code. A host I2C interface may not be required if the system is configured to use the Epson SPI-I2C bridge feature instead. Examples of devices connected to I2C include the HVPMIC, temperature sensors, and EEPROMs.

8.3.3 Host SPI Interface – Epson

The host SPI-Epson interface provides access to an SPI interface that is connected to the Epson controller when it is operating in serial interface mode. On short cables this interface has been operated at 20MHz successfully. In general the Epson controller should be placed on its own SPI bus due to the need to keep the chip selected for the entire duration of the image data transfer operation which may be up to 1MB.

8.3.4 Host SPI Interface – SD Card

The host SPI-SDCard interface provides access to an SPI interface that is connected to the SD Card. The SD Card is operated at 20MHz. If additional hardware is available in the host processor the SD Card could be operated in 4 bit parallel mode for improved data transfer speed.

8.3.5 Host Interrupt Interface

The interrupt interface supports the processing of interrupts. The code currently does not use interrupts but the first usage will be for notifying the code that the Epson is ready to accept a new command by the assertion of the HRDY line. The abstraction is still to be defined

8.3.6 Host Timer Interface

The timer interface provides platform specific timer implementations. Currently delays are coded as busy loops. A more power efficient mechanism will follow in a future release.

8.4 MSP430 Specific Host Interfaces

8.4.1 GPIO Interface

This is the reference implementation for the GPIO host interface and can be found in msp430/msp430-gpio.c. It supports the configuration of all features on all pins that can be configured. It is only possible to configure one pin at a time in a port. It is not possible to define the configuration of multiple pins in a port with one call – e.g. when defining an 8 bit bus as output or input. The code attempts to verify the request as much as it can. Much of the error checking code can be disabled once the porting process to a new platform has been completed and the platform configuration is stable.

8.4.2 I2C Interface

A single I2C interface is supported. I2C is only supported in USCI modules and the chosen UCSI module is defined in the msp430/mlsp430-i2c.c source file by setting the macros USCI_UNIT and USCI_CHAN as required. The code will then reconfigure itself to reference the correct I2C unit. In addition to specifying which UCSI module to use the I2C SDA and SCL pins need to be connected to the USCI unit by defining the appropriate pins as PL_GPIO_SPECIAL in the pl_gpio_config_list() call.

8.4.3 SPI Interface – Epson

SPI is supported in both USCI_A and USCI_B modules and the chosen USCI module is defined in the msp430/msp430-spi.c source file by setting the macros USCI_UNI and USCI_CHAN as required. The code will then reconfigure itself to reference the correct SPI unit. In addition to specifying which USCI module to use the SPI_CLK, SPI_MOSI and SPI_MISO pins need to be connected to the USCI unit by defining the appropriate pins as PL_GPIO_SPECIAL in the pl_gpio_config_list() call. Note that it is possible to use both the USCI_A and USCI_B units, i.e. USCI_A0 and USCI_B0 are physically different hardware units. A single SPI interface is supported for Epson controller communications. Multiple controllers can be connected to this bus and are selected using their chip select lines as required. This interface runs at 20Mbps reliably. Due to the need to keep the Epson chip selected for the duration of the image data transfer the Epson controller must be placed on a separate bus to the SD card so that multiple blocks can be read from the SD card.

8.4.4 SPI Interface – SD Card

SPI is supported in both USCI_A and USCI_B modules and the chosen USCI module is defined in the msp430/msp430-sdcard.c source file by setting the macros USCI_UNI and USCI_CHAN as required. The code will then reconfigure itself to reference the correct SPI unit. In addition to specifying which USCI module to use the SPI_CLK, SPI_MOSI and SPI_MISO pins need to be connected to the USCI unit by defining the appropriate pins as PL_GPIO_SPECIAL in the pl_gpio_config_list() call. Note that it is possible to use both the USCI_A and USCI_B units. i.e. USCI_A0 and USCI_B0 are physically different hardware units. A single SPI interface is supported for transferring data from the micro SD card slot. This interface runs at 20Mbps reliably.

8.4.5 UART Interface

UART mode is supported in the USCI_A module and the code handling this can be found in the msp430\msp430-uart.c source file. In the sample code, the UART interface is used only to handle stderr and stdout, and then only if CONFIG_UART_PRINTF is defined (in config.h). In Code Composer Studio it is not possible simply to override putc() and puts(), and instead a device has to be registered (see msp430_uart_register_files()).

8.4.6 Porting the Existing Code to another MSP430 Processor

Porting the existing code to a design which requires a different pin out is relatively straightforward. The necessary configuration information is mainly contained in the main.c file. To reconfigure the reference code follow the sequence below:

1. Determine which USCI units will be used in the new configuration. Ensure the unit is suitable for its intended purpose
2. Determine which pins are associated with the chosen USCI units
3. Determine which pins will be used for the Epson SPI signals HRDY, HDC, and RESET
4. Determine which pin(s) will be used for the Epson SPI chip select
5. Determine which pins may be necessary to control the power supplies
6. In each of the msp430/msp430-spi.c, msp430/msp430-sdcard.c, mps430/msp430-i2c.c and msp430/msp430-uart.c
    a. Define USCI_UNIT and USCI_CHAN as required
    b. Modify the definitions for the pins so they match the chosen UCSI unit, e.g.:

```
#define USCI_UNIT B
#define USCI_CHAN 0
// Pins from MSP430 connected to the SD Card
#define SD_CS MSP430_GPIO(5,5)
#define SD_SIMO MSP430_GPIO(3,1)
#define SD_SOMI MSP430_GPIO(3,2)
#define SD_CLK MSP430_GPIO(3,3)
```

7. In main.c, define the Epson SPI interface signals, e.g.:

```
// Remaining Epson interface pins
```

```
#define EPSON_HDC MSP430_GPIO(1,3)
#define EPSON_HRDY MSP430_GPIO(2,7)
#define EPSON_RESET MSP430_GPIO(5,0)
```

8. In main.c, define the power control and Epson chip select pins, e.g.:

```
#define B_HWSW_CTRL MSP430_GPIO(1,2)
#define B_POK MSP430_GPIO(1,0)
#define B_PMIC_EN MSP430_GPIO(1,1)
#define EPSON_CS_0 MSP430_GPIO(3,6)
```

Recompile the code and it has now been retargeted to the new pin assignments.

9 Supported Hardware

9.1 Hardware Components

This section lists the hardware components commonly found on boards intended to drive Plastic Logic displays that require software drivers.

9.1.1 Microchip EEPROMs

The code supports I2C EEPROMs up to 64KB in size. The code currently supports two I2C EEPROM types:
1. **24LC014** – this is a small 128B EEPROM fitted to later display interface boards and is used to store power supply calibration data. This permits accurate VCOM voltages to be achieved when the display interface board is swapped. It also stores other hardware configuration information.

2. **24AA256** – this is a 32KB EEPROM found on some display types. It is intended to store waveform information so that the necessary information to drive a display travels with the display. This allows the system to ensure the correct waveform information is used for the display. Since waveforms can exceed 32KB in size, the data stored on this EEPROM is compressed using the LZSS compression algorithm. EEPROM types can be added by extending the table that defines the device characteristics (in i2c-eeprom.c) and extending the enumeration of EEPROM types (in i2c-eeprom.h).

9.1.2 LM75 Temperature Sensor

The LM75 temperature sensor is a configurable I2C temperature sensor that can measure temperature autonomously at programmable intervals. It can be used when the temperature measuring facilities of the PMICs cannot be used for some reason. The measured temperature register can be read automatically by the Epson controllers.

9.1.3 Maxim MAX17135 HVPMIC

The Maxim PMIC is used on boards primarily intended to drive the 10.7" displays. Its key features are:

- I2C interface for configuration of power sequence timings
- Hardware signals for PowerUp/Down, PowerGood and PowerFault
- I2C commands for PowerUp/Down and power supply monitoring
- Inbuilt 8bit VCOM DAC
- Inbuilt LM75 compatible temperature sensor with automatic temperature sensing

9.1.4 TI TPS65185 HVPMIC

The TI PMIC is used on boards intended to drive the small displays. Its key features are:

- I2C interface for configuration of power sequence timings

- Hardware signals PowerUp/Down, PowerGood and PowerFault
- I2C commands for PowerUp/Down and power supply monitoring
- Inbuilt 9bit VCOM DAC
- Inbuilt LM75 compatible temperature sensor with on demand temperature sensing.

9.2 Epson Controllers

Epson have a range of controllers designed to support the output of images onto electrophoretic displays (EPD). The controllers differ in the size of display they can support, whether they have external or internal frame buffer memory, on-board or external power supplies and support for color displays. The controllers can be accessed via SPI or a 16-bit parallel data bus. In addition to the main EPD functionality the controllers contain a varying collection of useful hardware units that may be required in a system fitted with an electrophoretic display. For example, an I2C master, SPI master, GPIO ports, and internal temperature sensor. Which options are available will ultimately depend on the controller selected and how it is connected to the display and other system components. The code supports the Epson **S1D13524** and **S1D13541** controllers in various configurations. The S1D13524 controller supports large (up to 2560x2048 greyscale pixels or 2560x2048 RGBW color sub-pixels) displays and is fitted to a circuit board with its external SDRAM. The S1D13541 controller supports smaller displays (up to 854x480 pixels greyscale) and is physically bonded to the display module.

9.2.1 Power State Management Epson S1D13541

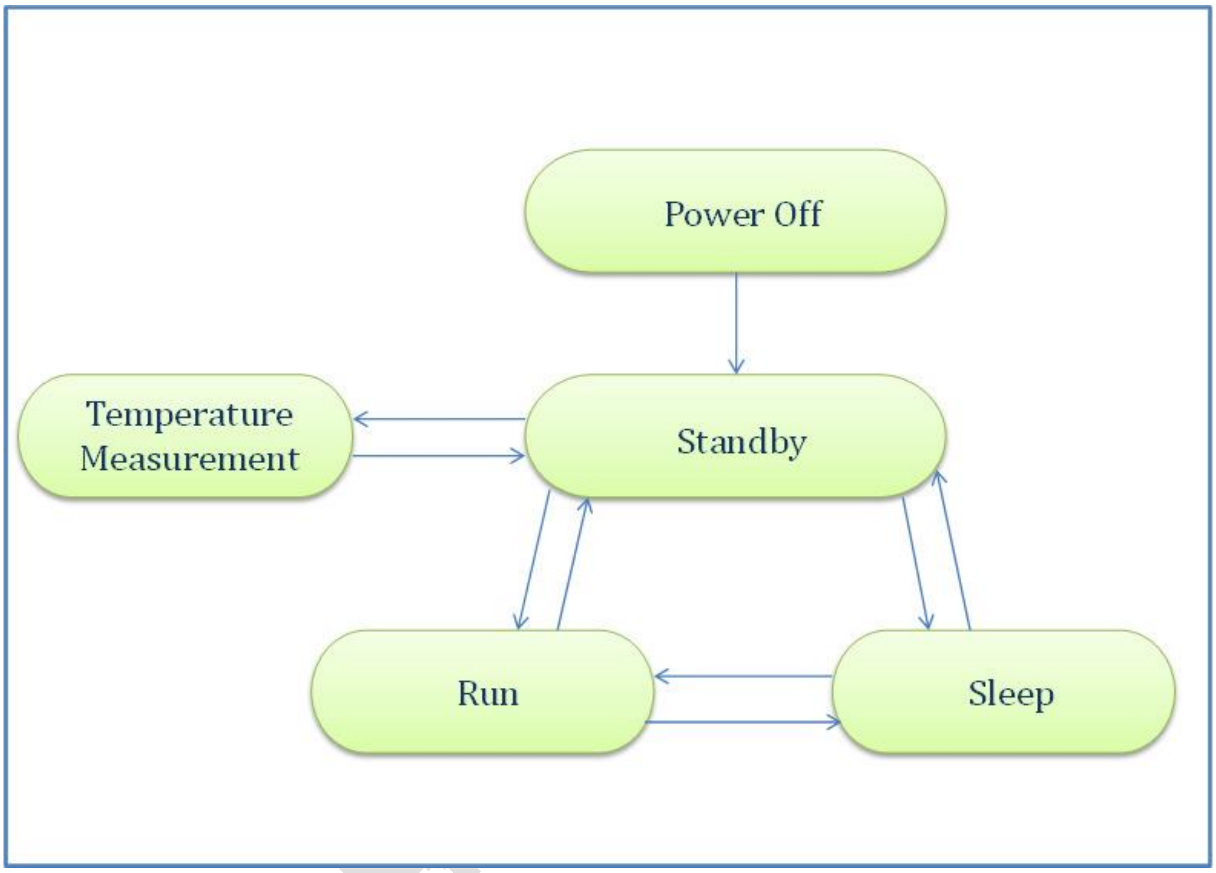
The Epson S1D13541 controller can be configured to one of several power states; helping to minimize power use when appropriate.

These power states are:

- **Power Off**
  - Clock chip disabled
  - 3V3 power to S1D13541 disabled
- **Standby**
  - Can be set from SLEEP or RUN mode
  - Clock chip enabled
  - Power save status bit set to 0
  - Source/gate driver powered off
- **Run**
  - Can be set from SLEEP or STANDBY mode
  - Clock chip enabled
  - Power save status bit set to 1
  - Source/gate driver powered on
- **Sleep**

- Can be set from RUN or STANDBY mode
- Clock chip disabled
- Source/gate driver powered off
- Power save status bit set to 0

The figure below shows the possible power state transitions.



Below is a breakdown of the actions that must be taken for each of the power state transitions.

9.2.1.1 Run -> Standby

1. STBY command (CMD(0x04), no parameters) issued to Epson controller
2. Wait for HRDY = 1

9.2.1.2 Sleep -> Standby

1. Set CLK_EN GPIO true to re-enable clock
2. Set REG[0x0006] bit 8 to 1 for normal power supply
3. STBY command (CMD(0x04), no parameters) issued to Epson controller
4. Wait for HRDY = 1

9.2.1.3 Run/Standby -> Sleep

1. SLP command (CMD(0x05), no parameters) issued to Epson controller
2. Wait for HRDY = 1
3. Set REG[0x0006] bit 8 to 0 for minimum power supply

4. Set CLK_EN GPIO to false to disable clock

9.2.1.4 Standby -> Run

1. RUN command (CMD(0x02), no parameters) issued to Epson controller
2. Wait for HRDY = 1

9.2.1.5 Sleep -> Run

1. Set CLK_EN GPIO to true to re-enable clock
2. Set REG[0x0006] bit 8 to 1 for normal power supply
3. RUN command (CMD(0x02), no parameters) issued to Epson controller
4. Wait for HRDY = 1

9.2.1.6 Run/Standby/Sleep -> Power Off

1. SLP command (CMD(0x05), no parameters) issued to Epson controller 2. Set CLK_EN GPIO to false to disable clock
2. Set 3V3_EN GPIO to false to disable 3V3 power supply

Note

Any data in the image buffer will be lost when going into off mode. If the current displayed image is to be retained when powering back up, the contents of the image buffer should be copied to a suitable location (e.g. an SD card) before continuing with the power off. This image can then be loaded back into the image buffer when coming out of power off mode.

9.2.1.7 Power Off -> Standby

Note

After each of the following commands, the host should wait for HRDY to be 1 before continuing.

1. Set 3V3_EN GPIO to true to enable 3V3 power supply
2. Set CLK_EN GPIO to true to enable clock
3. INIT_CMD_SET command (CMD(0x00 + Epson Instruction Code Binaries)) issued to Epson controller
4. INIT_SYS_STBY command (CMD(0x06, no parameters) issued to Epson controller
5. Set Protect Key Code to REG[0x042C] and REG[0x042E]
6. BST_WR_MEM command (CMD(0x1D) + Waveform Storage Address) to start loading waveform data
7. WR_REG command (CMD(0x11), 0x154 + Waveform) to load waveform data
8. BST_END_MEM command (CMD(0x1E), no parameters) to end loading waveform data
9. RUN command (CMD(0x02), no parameters) issued to Epson controller
10. UPD_GDRV_CLR command (CMD(0x37), no parameters)
11. WAIT_DSPE_TRG command (CMD(0x28), no parameters)
12. S1D13541 is initialized into known state

The EPD Panel and Image Buffer should now be initialized to a known state; either the standard white initialization waveform, or image data copied to a safe medium before power off was called.

9.2.1.8 Power State Demo

A power state demo can be launched using the Plastic Logic reference code by including the following in

config.h:

```
#define CONFIG_DEMO_POWERMODES 1
```

This demo will transition through the power states with the following behavior:

- Go into RUN mode
- Load an image into the image buffer
- Update the display
- Go into SLEEP mode for 2 seconds
- Go into STANDBY mode for 2 seconds
- Go into RUN mode
- Update the display (with image data retained from the previous update)
- Go into POWER OFF mode (CLKI and 3V3 disabled) for 2 seconds
- Go through power on initialise

9.3 Plastic Logic Evaluation Hardware

9.3.1 Display Types

The code supports the following Plastic Logic display types. Additional displays will be supported as required.

Display Type	Resolution	Notes
D107_T2.1	1280x960	External Controller Requires the Mercury display connector board
D107_T3.1	1280x960	External Controller Requires the Hermes 3.0 display connector board
S047_T2.1	800x450	External Controller Requires the Helios display connector board
S040_T1.1	400x240	Bonded Controller 4.0" @115ppi
S049_T1.1	720x120	Bonded Controller 4.9" @150ppi
S079_T1.1	768x192	External Controller Requires the Hermes 2.0 display connector board
S115_T1.1	1380x96	External Controller Requires the Hermes 2.0 display connector board
D054_T1.1	680x155	Bonded Controller 5.4" @130ppi

9.3.2 Parrot - MSP430 Processor Board

The Parrot board docks with the Ruddock2 motherboard to provide access to the display interfaces. It has the same form factor and connector pin out as a BeagleBone allowing the processors to be easily swapped for evaluation or development work. The Parrot board can also be used without the Ruddock2 by connecting it directly to the Z6, Z7 or Raven boards via the 24-pin "serial" interface.

The board has the following features:

- MSP430F5438A, clocked at 20MHz
- A 32KHz oscillator for low power operation
- micro SD card socket
- On-board reset switch
- JTAG programming header (an adapter may be required to mate with the MSP-FET430UIF programmer)
- All 100 processor pins available on debug headers
- On-board power regulation and power socket (can also be powered from USB)
- The board has 1 LED for power good and another connected to a pin on the processor for status indication
- 24-pin "serial" interface to Z6, Z7 and Raven boards
- Provision for an SPI daisy-chain of MSP430 boards using 2 SPI channels (upstream and downstream)

9.3.3 Ruddock2

The Ruddock2 board is a motherboard that sits between a processor board, currently either BeagleBone or a microcontroller (MSP430) and the display interface board. It provides signal routing from the processor to the interface connectors together with some LEDs and switches that can be used to configure the software or create a user interface. The board allows the Epson serial, parallel and TFT interfaces to be used depending on the interface board and controller selected. The processor board can disable all power from the Ruddock2 under software control allowing hardware components, e.g. display interface boards, to be safely exchanged. The board has a 128B EEPROM which can be used as non-volatile storage if required.

9.3.4 HB Z6/Z7

The Z6 and Z7 are intended to drive a S1D13541 small display controller which is bonded to the display itself. The boards differ in the display connector used. The Z7 board is used to drive the S049_T1.1 bracelet display and the Z6 is used to drive all other small Plastic Logic displays. The boards have a TI PMIC and a 128B EEPROM for storing power supply calibration data. The VCOM DAC in the PMIC is used to set the VCOM value for the display. All versions of the Z7 board have the provision to turn off 3V3 power to the display controller; this feature is absent on version 6.1

of the Z6 but has been introduced as of version 6.3, along with the ability to control the clock enable and PMIC wake signals.

9.3.5 Raven

The Raven board is designed to drive 10.7" D107_T2.1, D107_T3.1 and S047_T2.1 displays. The board has an Epson S1D13524 controller and associated memory, a Maxim PMIC, a 128B EEPROM for storing power supply calibration data and an LM75 temperature sensor. The VCOM DAC in the PMIC is used to set the VCOM value for the display. The board has input connectors that allow it to be controlled via the Serial host interface (SPI) or Parallel host interface. Additionally the signals to support data transfer using the TFT interface are available. The board has 5 test pads which bring out the 5 Epson GPIO pins found on the S1D13524.

10 Relevant Data Sheets and Additional Resources

10.1 Plastic Logic Documents

Detailed schematics and design documents are available from Plastic Logic for all of the boards supported by this software. Other relevant documents are:

Electronics for small displays

An overview of the electronics required to drive displays

Plastic Logic Software Manual

A detailed description of the EEPROM data formats and some associated software tools

How to integrate VCOM calibration

A detailed description of calibrating and setting the VCOM voltage

10.2 Third Party Datasheets and Resources

TI TPS65185 HVPMIC

http://www.ti.com/product/tps65185

Maxim MAX17135 HVPMIC

http://datasheets.maximintegrated.com/en/ds/MAX17135.pdf

LM75 temperature sensor

http://www.ti.com/lit/ds/snos808o/snos808o.pdf

Microchip 24LC014 EEPROM

http://ww1.microchip.com/downloads/en/DeviceDoc/21809G.pdf

Microchip 24AA256 EEPROM

http://ww1.microchip.com/downloads/en/DeviceDoc/21203Q.pdf

Epson S1D13524 EPDC (NDA required for full datasheet)

http://vdc.epson.com/index.php?option=com_docman&task=doc_download&gid=1768&Itemid=99

Epson S1D13541 EPDC (NDA required for full datasheet)

no public datasheet

TI MSP430 tools and software

http://www.ti.com/lsds/ti/microcontroller/16-bit_msp430/tools_software.page

11 Appendix A - License Text

The license text for third party components is reproduced below:

11.1 FatFs

11.2 Texas Instruments