# BUILD GUIDE

## Display Evaluation Kit

## K_MSP430

## Microcontroller Platform for 1.1", 1.38", 2.1", and 3.1" for Ultrachip EPD driver

## Part No.: 303001

**Containing Part No. 301006 and 301013**

Revision 2

5-June-2018

| Revision Status | Date | Author | Reason of Modification |
|---|---|---|---|
| 1 | 15-Feb-2018 | RP | Initial Version |
| 2 | 05-Jun-2018 | RP | Fix Git Paths and Settings |

# 1 Table of Contents

# 2 Introduction

This document provides an introduction how to set up a build environment and how to set up the development software. 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 Glossary

**CCS**

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

**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)

**Mercury / Hermes / Sojus 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

**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

# 4 Getting Started

This section covers setting up the hardware and software so that a given display type can be driven. Please follow the steps outlined in order to setup and build the software.

## 4.1 Get the Code

Get the source code, either by downloading an archive or cloning the repository:

```
git clone https://github.com/plasticlogic/pl-uc8156-mcu-epd.git
```

It is recommended to use the most recent version branch. For compatibility reasons, the former versions are available as well.

## 4.2 Code Composer Studio Setup

The Plastic Logic reference code project uses Texas Instruments' Code Composer Studio IDE v5.5. This section details the steps necessary to set up Code Composer Studio to build and debug the code.

### *Important*

This project uses **DOS line endings**. The .gitattributes file tells Git to automatically convert text file line endings to DOS at commit time. To avoid complications on Unix-like operating systems (Linux, MacOSX...) please configure your text editor to use DOS line endings.

1. Install Code Composer Studio from TI on MS Windows or GNU/Linux
2. Create new CCS project
   - Open the menu **File -> New -> CCS Project**
   - Project name: **pl-mcu-epd** (for example)
   - Project location: Ensure the location is different to that of the cloned repository
   - Output: **Executable**
   - Device family: **MSP430**
   - Variant: **MSP430F5438A** (for PL Parrot v1.x boards)
   - Connection: **TI MSP430 USB1** (uses an MSP-FET430UIF USB-JTAG interface)
   - Project template: **Empty Project**
3. Import the source code
   - Open the menu **File -> Import -> General**
   - To import a clone repository, choose **File System**
   - To import a source archive, choose **Archive File**

   ### *Note*

   When using a source archive, the source will need to be moved out of the extra sub-directory that is created.
   - Browse to select your pl-mcu-epd cloned repository or archive file
   - Select all the files (checkbox near pl-mcu-epd)
   - Click on **Finish**
4. Compiler configuration

Open the menu **Project -> Properties**

Click on the **Show Advanced Settings** link in the bottom left of the properties window

The following project settings need to be modified:

**CCS Build -> MSP430 Compiler -> Debug Options**

**Debugging Model:** Should be set to **'Full Symbolic Debug'**

**CCS Build -> MSP430 Compiler -> Advanced Options -> Language Options:**

Check the box for **Enable Support for GCC Extensions**

**CCS Build -> MSP430 Compiler -> Advanced Options -> Library Function**

Set the level of printf support to **nofloat**

**CCS Build -> MSP430 Compiler -> ULP Advisor:**

Disable the following items by unchecking the associated box:

**2: Software (SW) delay**

**5: Processing/power intensive operations**

**CCS Build -> MSP430 Compiler -> Advanced Options -> Diagnostic Options:**

Check the box for **Emit Diagnostic Identifier Numbers**

**CCS Build -> MSP430 Linker -> Basic Options**

**Set C system stack size: 512**

**Heap size for C/C++ dynamic memory allocation: 10000**

**C/C++ General -> Paths and Symbols**

**Includes:** Add the following paths to the includes list (check all three boxes in the creation dialog window)

/${ProjName}/msp430

/${ProjName}

**Includes:** Re-order the includes list so that the order is as below:

/${ProjName}/msp430

${CCS_BASE_ROOT}/msp430/include

/${ProjName}

${CG_TOOL_ROOT}/include

5. . Setup config.txt

Finally, a config.txt file must be placed on the SD-Card. A sample config file is supplied for each supported display type. More information on the various code configuration options can be found in the section Configuring the Code.

## 4.3 SD Card Setup

The micro SD card for the processor board must be formatted as a FAT/FAT16 file-system (not FAT32). The SD card contents (initialization data and images) can be retrieved from the Plastic Logic GitHub repository

```
git clone https://github.com/plasticlogic/pl-uc8156-mcu-sd-card.git
```

Unzip this archive and place the resulting files on the SD card so that the root directory of the file-system contains the folders S011_T1.1, S014_T1.1, etc. The Software reads the waveform and the VCOM value automatically from the Display Processors memory.

Place the micro SD card in the micro SD card socket on the processor board.

## 4.4 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. You should now be able to see a slideshow of stock images from the 0:/<Display-Type>/img folder being shown on the display until execution is halted.

The slideshow will skip any files that do not have the extension ".pgm"

## 4.5 Toolchains

### 4.5.1 Code Composer Studio

Code Composer Studio has been used extensively during development of the code in conjunction with the MSP-FET430UIF USB/JTAG programmer. Both have proved to be extremely reliable in use. There is a free version of the tools which restrict the size of code they will generate to 16KB. The full version can be evaluated free for 90 days. The current configuration of the code is too large to fit within the 16K limit, however by removing some features, e.g. Fat file system support then the free version may be sufficient. A very useful feature of the IDE is the ability to use standard printf type functions and have the output displayed in a console window within the IDE. In order for this to work the amount of memory set aside for the stack and heap must be increased and the io functionality must be enabled in the project build configuration. A small amount of source code in the platform common layer was taken from Plastic Logics equivalent Linux drivers. The code uses anonymous unions extensively and in order to get the code to compile it was necessary to add a compiler flag (--gcc) to tell it to behave more like gcc.

### 4.5.2 msp430-gcc

There is an open source msp430 tool chain available “msp430-gcc”. Some work has been done to support this tool chain but the work is not yet complete. Much of the code compiles cleanly however there are some issues related to pragmas used to declare interrupt handlers. Full support for this tool chain will depend on customer demand.