BUILD GUIDE

Display Evaluation Kit

K_i.MX6

for

4.7", 7.9", 10.7", 11.5" and 15.4"

Part No.: 303008, 303012, 303013, 303014

Containing Part No. 301028 and 301038

Revision 1

15-February-2018

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

Contents

Get the Source

The PL Linux has to be compiled on a Linux environment (i.e. Ubuntu 1404).

A VM will work fine.

To load the Plasticlogic github repositories, install git first:

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

CD into a folder of your choice and untar the source archives to a folder

```
$ mkdir ~/myBuild && cd ~/myBuild
$ tar -xzf linux-3.14.52.tar.gz ~/myBuild/kernel
$ tar -xzf plsdk_3.14.52.tar.gz ~/myBuild/plsdk
$ tar -xzf i.MX6SL-freescale-debian-setup.tar.gz ~/myBuild/imx6sl-setup
```

or clone the plasticlogic github repositories

```
$ mkdir ~/myBuild && cd ~/myBuild
$ git clone https://github.com/plasticlogic/linux.git -b pl-imx-3.14.52-hbz3.3 kernel
$ git clone https://github.com/plasticlogic/imx-setup.git -b debian-imx6sl imx6sl-setup
```

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:

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
```

Build PL linux 3.14.52 kernel

Install the arm-linux-gnueabi toolchain

```
$ sudo apt-get install libc6-armel-cross libc6-dev-armel-cross
$ sudo apt-get install binutils-arm-linux-gnueabi
$ sudo apt-get install libncurses5-dev
$ sudo apt-get install gcc-arm-linux-gnueabi
$ sudo apt-get install g++-arm-linux-gnueabi
$ sudo apt-get install lzop uboot-mkimage u-boot-tools
```

Build the kernel

```
$export CC="arm-linux-gnueabi-"
$ make -C kernel imx_v7_pl_defconfig ARCH=arm CROSS_COMPILE=$CC
LOADADDR=0x80008000
$ make -C kernel uImage modules dtbs modules_install
INSTALL_MOD_PATH=../modules LOCALVERSION= ARCH=arm CROSS_COMPILE=$CC
LOADADDR=0x80008000
$ cp -f kernel/arch/arm/boot/uImage modules/uImage
$ cp -f kernel/arch/arm/boot/dts/imx6sl-evk-hummingbird-z33.dtb
modules/imx6sl-evk-hummingbird-z33.dtb
```

Build PLSDK

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
```

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/
```

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

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
```

Build the filesystem

```
$ sudo apt-get install qemu-user-static debootstrap binfmt-support
$ targetdir=rootfs
$ distro=jessie
$ mkdir $targetdir
$ sudo debootstrap --arch=armhf --foreign $distro $targetdir
$ sudo cp /usr/bin/qemu-arm-static $targetdir/usr/bin/
$ sudo cp /etc/resolv.conf $targetdir/etc
$ sudo chroot $targetdir
```

Inside the chroot set up the environment again

```
$ distro=jessie
$ export LANG=C
$ /debootstrap/debootstrap --second-stage
```

setup support files and apt configuration

```
$ cat <<EOT > /etc/apt/sources.list
deb http://ftp.uk.debian.org/debian $distro main contrib non-free
deb-src http://ftp.uk.debian.org/debian $distro main contrib non-free
deb http://ftp.uk.debian.org/debian $distro-updates main contrib non-free
deb-src http://ftp.uk.debian.org/debian $distro-updates main contrib non-free
deb http://security.debian.org/debian-security $distro/updates main contrib
non-free
deb-src http://security.debian.org/debian-security $distro/updates main
contrib non-free
EOT
```

Update Debian package database:

```
$ apt-get update
```

Install locales

```
$ apt-get install locales dialog
$ dpkg-reconfigure locales
```

Install some useful packages inside the chroot (add desired packages here)

```
$ apt-get install openssh-server ntpdate usbutils i2c-tools mtd-utils devio
```

Set a root password

```
$ passwd
```

Build a basic network interface file

```
$ echo <<EOT >> /etc/network/interfaces
allow-hotplug eth0
# The loopback interface
auto lo
iface lo inet loopback
# Wireless interfaces
iface wlan0 inet dhcp
        wireless_mode managed
        wireless_essid any
        wpa-driver wext
        wpa-conf /etc/wpa_supplicant.conf
# Wired or wireless interfaces
auto eth0
iface eth0 inet dhcp
iface eth1 inet dhcp

# Ethernet/RNDIS gadget (g_ether)
# ... or on host side, usbnet and random hwaddr
iface usb0 inet static
        address 192.168.1.10
        netmask 255.255.255.0
        network 192.168.1.0
        gateway 192.168.1.1
EOT
```

Set the hostname

```
$ echo nameme > /etc/hostname
```

Enable the serial console, Debian sysvinit way

```
$ echo T0:2345:respawn:/sbin/getty -L ttymxc0 115200 vt100 >> /etc/inittab
$ exit
```

Outside the chroot shell, remove the files

```
$ sudo rm $targetdir/etc/resolv.conf
$ sudo rm $targetdir/usr/bin/qemu-arm-static
```

Copy the PLSDK, the kernel modules and the imx6sl-setup to the rootfs

```
#remember: targetdir=~/myBuild/rootfs
$ cp -rf ~/myBuild/plsdk/out/arm/* $targetdir/usr/
$ cp -rf ~/myBuild/modules/lib $targetdir/
$ cp -rf ~/myBuild/imx6sl-setup /* $targetdir/
```

Copy the built rootfs to a SD card partition

Assuming /dev/sdb2 is the SD card root partition (as provided by PL)

```
$ mkdir ~/myBuild/sd_rootfs
$ mount /dev/sdb2 ~/myBuild/sd_rootfs
$ rm -rf ~/myBuild/sd_rootfs/*
$ cp -rf ~/myBuild/rootfs ~/myBuild/sd_rootfs
```

Note

The Kernel image (`uImage_3.14.52`) and the device tree binary (`imx6sl-evk-hummingbird-z34.dtb`) must be placed on the root folder of the USERDATA partition (mmcblk1p1)

Appendix

U-Boot environment:

The PL Linux uses u-boot 2014-04 with the following settings:

```
baudrate=115200
bootargs=console=ttymxc0,115200 keypad video=mxcepdcfb:PL_STD
video=mxc_elcdif_fb:off loglevel=8 root=/dev/mmcblk1p2 rootwait
bootargs_base=setenv bootargs console=ttymxc0,115200 keypad
video=mxcepdcfb:PL_STD video=mxc_elcdif_fb:off loglevel=8
bootargs_mmc=setenv bootargs ${bootargs} root=/dev/mmcblk1p2 rootwait
bootcmd=run bootcmd_mmc_file
bootcmd_3.0.35=run bootargs_base bootargs_mmc; fatload mmc
${mmcdev}:${mmcpart} ${loadaddr} uImage_3.0.35; bootm ${loadaddr}
bootcmd_mmc=run bootargs_base bootargs_mmc; mmc dev 1; mmc read ${loadaddr}
0x800 0x2000; mmc read ${fdt_addr} 0x2800 0x100; bootm ${loadaddr} -
${fdt_addr}
bootcmd_mmc_file=run bootargs_base bootargs_mmc; run loadfdt; run loadkernel;
bootm ${loadaddr} - ${fdt_addr}
bootdelay=3
ethact=FEC
ethaddr=00:04:9f:03:2c:06
ethprime=FEC
fastboot_dev=mmc1
fdt_addr=0x83000000
fdt_file= imx6sl-evk-hummingbird-z34.dtb
fdt_high=0xffffffff
filesize=674620
initrd_high=0xffffffff
kernel=uImage_3.10.53
loadaddr=0x80800000
loadfdt=fatload mmc ${mmcdev}:${mmcpart} ${fdt_addr} ${fdt_file}
loadkernel=fatload mmc ${mmcdev}:${mmcpart} ${loadaddr} ${kernel}
mmcdev=1
mmcpart=1
splashpos=m,m
stderr=serial
stdin=serial
stdout=serial
```