# chili-buildroot-sdk manual

v. 2015.02-1.0

# Contents

# Main Page

## chiliboard

## Buildroot SDK

# Chiliboard/Getting started

This document describes in a few steps how to get started with chiliboard. The description is based on the latest Buildroot SDK.

## Download and unpack Buildroot SDK

Download the latest Buildroot SDK tarball from ftp server:

```
ftp://chiliboard.org/sdk
```

to the local directory, and unpack it:

```
$ tar -xjf chili-buildroot-sdk-2015.02-1.0.tar.bz2
```

## Prepare SD card

The bootable SD card for chiliboard contains two partitions: boot (fatfs) and rootfs (extfs).

### SD card partitioning

To partition your SD card, follow the bellow steps:

- Insert the SD card into PC.
- Run command

```
$ dmesg
```

  to determine which device is assigned to the SD card.

  **Be careful, because the use of a wrong device can erase data from other media.**

- Run commands:

```
$ cd buildroot_sdk_path/scripts
$ sudo ./mkcard.sh /dev/sdX
```

where **/dev/sdX** is the determined previously device assigned to the SD card (for example /def/sdc).

### File copying

To copy filesystem to the SD card, follow the bellow steps:

- Mount SD card partitions. By default Ubuntu 12.04 mount partitions in /media, while Ubuntu 14.04 in /media/user.
- Change the current directory:

```
$ cd buildroot_sdk_path/images
```

- Determine which version of chiliboard you have and copy the apropriate bootloader images to the boot partition. Details about the images for different hardware configurations are in SDK Content. In the example bellow are copied files for chiliboard with 256 MB NAND (FLASH) and 128 MB RAM:

```
$ cp -t /media/boot u-boot-nand256-ram128/MLO u-boot-nand256-ram128/u-boot.img uEnv.txt
```

- Next, unpack apropriate filesystem tarball to /rootfs partiton:

```
$ sudo tar -xz -C /media/rootfs -f rootfs_qt5.tar.gz
```

- Finally, copy kernel image and device-tree file to boot directory on the rootfs partition. Choose the appropriate dtb file and copy it with the name of **am335x-chiliboard.dtb** to the target:

```
$ cp zImage /media/rootfs/boot
$ cp am335x-chiliboard-nand256-ram128.dtb /media/rootfs/boot/am335x-chiliboard.dtb
```

## Connect chiliboard to PC

To boot chiliboard from the SD card set the boot-switches BOOT[4:0] to 00011. To be able to use the debug console, connect chiliboard to PC via USB cable or serial port (depending on the chiliboard version). When you use a serial port, use an additional power supply.

## Install and open serial port terminal

To communicate with chiliboard first install the serial port termianl on your PC, for example Minicom. Under Ubuntu install with:

```
$ sudo apt-get install minicom
```

Next, run and configure Minicom:

```
$ sudo minicom -s
```

Depending on the chiliboard version and the serial interface set the appropriate serial port (usually /dev/ttyUSB0 or /dev/ttyS0). Other parameters: bit rate - 115200, data bits - 8, no parity, one stop bit, without flow control. Save the configuration and then run Minicom with command:

```
$ sudo minicom
```

Additional information:

```
http://processors.wiki.ti.com/index.php/Setting_up_Minicom_in_Ubuntu
https://help.ubuntu.com/community/Minicom
```

## Insert the SD card and boot system

Insert the newly programmed SD card into chiliboard and the system will automatically boot. When chiliboard is powered off, push the power button. In the terminal is visible current log of the system boot. After loading, log in as **root**. By default, **no password**

## Power off

**Remember to properly close system.** The rapid removal of power can corrupt the file system. Before turning off power close properly system using command:

```
$ poweroff
```

or press power button.

# Chiliboard/Using GPIO and LED

chiliboard has two built-in user LED's, and a lot of unused pins that can be used as general-purpose input/output lines (GPIO).

## User LED's

User leds can be controlled by **/sys/class/leds** directory, and are visible as **led0** and **led1** subdirectories. By default led0 uses heartbeat trigger. To turn on the **led1**, type:

```
echo 1 > /sys/class/leds/led1/brightness
```

and to turn off

```
echo 0 > /sys/class/leds/led1/brightness
```

## GPIO

GPIO lines can be controlled from the user space via the directory **/sys/class/gpio**.

For example, to set the GPIO1_8 pin to 1 do:

- Calculate the index of the pin as 1*32+8 = 40 and export it to the gpio subsystem using:

```
echo 40 > /sys/class/gpio/export
```

  In the **/sys/class/gpio** will be created gpio40 directory representing selected pin.

- Set the direction of the line:

```
echo out > /sys/class/gpio/gpio40/direction
```

- Then the value of the line

echo 1 > /sys/class/gpio/gpio40/value

# Chiliboard/LCD cape

This document describes how tu use LCD cape with chiliboard. To test the LCD cape should be used rootfs image with Qt library and device tree configuration am335x-chiliboard-*-lcd.dtb.

## LCD panel

In the LCD cape is used LCD panel with resolution 800x640 and 24-bit color. By default, for display is used DRM lcd driver, which also emulates fbdev through **/dev/fb0**. For details about the driver, see http://processors.wiki.ti.com/index.php/Linux_Core_LCD_Controller_User_Guide.

To drive LCD backlight is used TPS65217 boost converter. To change the brightness, type:

```
$ echo value > /sys/class/backlight/tps65217-bl/brightness
```

where value is a number from 0 to 100.

By default, in the system is enabled display blanking, to disable blanking type:

```
$ echo 0 > /sys/class/graphics/fb0/blank
```

and to enable:

```
$ echo 4 > /sys/class/graphics/fb0/blank
```

## Touchscreen

To handle the touchscreen events is used tslib library. When you first run the system with LCD cape, you should calibrate the touchscreen using:

```
$ ts_calibrate
```

After calibration, the touchscreen can be tested by:

```
$ ts_test
```

when the panel is too sensitive, modify the parameters in the file /etc/ts.conf.

## Graphics accelerator SGX530

Processors AM3354, AM3358 and AM3359 have built-in graphics accelerator PoverVR SGX530, which is supported in this SDK. To load SGX drivers execute script ti-gfx:

```
$ ti-gfx start
```

and to unload:

```
$ ti-gfx stop
```

To permanently enable SGX drivers loading when the system is loaded, copy script /usr/sbin/ti-gfx to the directory /etc/init.d/ as S80ti-gfx.

To test SGX drivers run demos:

```
$ /usr/bin/OGLES2ChameleonMan
$ /usr/bin/OGLES2MagicLantern
```

For details about SGX drivers, see http:/ / processors. wiki. ti. com/ index. php/ AM35x-OMAP35x_Graphics_SDK_Getting_Started_Guide.

## Using Qt4 library

To run programs using Qt4 should be built rootfs with Qt4 library. How to configure buildroot with Qt4 see Building customized image. Before running the Qt4 program should be exported the variable enabling support for the touchscreen handled by tslib:

```
$ export QWS_MOUSE_PROTO=Tslib:/dev/input/event0
```

Running the sample application:

```
$ /usr/share/qt/demos/deform/deform -qws
```

## Using Qt5 library

To run programs using Qt5 should be built rootfs with Qt5 library. How to configure buildroot with Qt5 see Building customized image.

### Using linuxfb

Example applications using fbdev

```
$ cd /usr/lib/qt/examples
$ ./touch/dials/dials -platform linuxfb -plugin tslib:/dev/input/event0
$ ./widgets/mainwindows/menus/menus -platform linuxfb -plugin tslib:/dev/input/event0
```

### Using hardware graphics acceleration

To run application using touchscreen (handled by tslib) using eglfs platform (OpenGL ES 2.0) should be disabled input on eglfs by setting environment variable:

```
$ export QT_QPA_EGLFS_DISABLE_INPUT=1
```

Examples of applications using eglfs:

```
$ cd /usr/lib/qt/examples
$ ./touch/dials/dials -platform eglfs -plugin tslib:/dev/input/event0
$ ./opengl/hellogl2/hellogl2 -platform eglfs -plugin tslib:/dev/input/event0
$ ./webkitwidgets/fancybrowser/fancybrowser -platform eglfs -plugin tslib:/dev/input/event0
$ ./webkitwidgets/browser/browser -platform eglfs -plugin tslib:/dev/input/event0
$ ./webkitwidgets/previewer/previewer -platform eglfs -plugin tslib:/dev/input/event0
```

Useful links for Qt5

```
http://doc.qt.io/qt-5/embedded-linux.html
http://doc.qt.io/QtForDeviceCreation/qtee-customization.html
```

### Buttons

Buttons located on the LCD cape are handled by MCP23008 expander and are available in user space as GPIOs. Buttons are maped with indexes from 504 to 508. To be able to read them, you must first export them:

```
$ echo 504 > /sys/class/gpio/export
$ echo 505 > /sys/class/gpio/export
$ echo 506 > /sys/class/gpio/export
$ echo 507 > /sys/class/gpio/export
$ echo 508 > /sys/class/gpio/export
```

And to read the state of the selected button type:

```
$ cat /sys/class/gpio/gpioX/value
```

where X is the index of the buuton. For example, for the first button: $ cat /sys/class/gpio/gpio504/value

# Buildroot SDK/SDK content

The chili-buildroot-sdk is software development kit for platforms based on the chiliSOM module. The SDK is developed, built and verified on Ubuntu LTS 12.04 and 14.04.

## chili-buildroot-sdk-2015.02-1.0 release notes

SDK is based on the following software components:

- Kernel - TI version 3.14.35
- u-boot - TI version 2014.07
- TI 3D Graphics SDK 05.01.01.02
- Buildroot 2015.02
- gcc 4.8.4
- binutils 2.24
- Qt 4.8.6 and Qt 5.4

## SDK directory structure

The chili-buildroot-sdk-2015.02-1.0 contains the following directories and files:

- scripts
  - mkcard.sh
- images
  - u-boot-nand0-ram128
    - MLO, u-boot.img
  - u-boot-nand256-ram128
    - MLO, u-boot.img
  - u-boot-nand256-ram256
    - MLO, u-boot.img
  - u-boot-nand512-ram256
    - MLO, u-boot.img

  - am335x-chiliboard-nand0-ram128.dtb
  - am335x-chiliboard-nand256-ram128.dtb
  - am335x-chiliboard-nand256-ram256.dtb
  - am335x-chiliboard-nand256-ram256-lcd.dtb
  - am335x-chiliboard-nand512-ram512.dtb
  - am335x-chiliboard-nand512-ram512-lcd.dtb
  - zImage
  - rootfs_minimal.tar.gz
  - rootfs_qt4.tar.gz
  - rootfs_qt5.tar.gz
  - uEnv.txt
- chili-buildroot-2015.02

Description of the content:

- **images** - bootloader, kernel images and root file system tarballs for all hardware configurations.
- The configurations are determined by adding a prefix to the file name in the following format: -nandX-ramY, where X and Y is the amount of NAND (FLASH) and RAM defined in megabytes.
- Subdirectory **u-boot*** contain **MLO** and **u-boot.img** images that should be copied to boot partition of the SD card.
- File **uEnv.txt** should be copied to the boot partition.
- File **am335x-chiliboard-*.dtb** is the kernel device-tree configuration and should be copied (with changed name to **am335x-chiliboard.dtb**) to /boot directory of the rootfs partition.
- File **zImage** is the kernel image and also should be copied to the /boot directory.
- File **rootfs_*.tar.gz** is a tarball of rootfs file system and should be unpacked to rootfs partition.
- Directory **chili-buildroot-2015.02** contains all files of the Buildroot with modifications for chili platform.

## u-boot sources

u-boot source files are located in:

- buildroot-2015.02/board/grinn/ti-u-boot-v2014.07.tar.gz - TI u-boot tarball
- buildroot-2015.02/board/grinn/chiliboard/patches/u-boot - patches for chili platforms

## Kernel sources

Linux kernel source files are located in:

- buildroot-2015.02/board/grinn/ti-linux-3.14.35.tar.gz - TI kernel tarball
- buildroot-2015.02/board/grinn/chiliboard/patches/linux - patches for chili platforms

## Buildroot configurations

In the chili-buildroot-sdk are defined the following minimal configurations:

```
chiliboard_nand0_ram128_defconfig
chiliboard_nand256_ram128_defconfig
chiliboard_nand256_ram256_defconfig
chiliboard_nand256_ram256_lcd_defconfig
chiliboard_nand512_ram512_defconfig
chiliboard_nand512_ram512_lcd_defconfig
```

All configurations include a toolchain, kernel and u-boot configuration, basic file system, and does not contain any additional packets. These configurations can be used as a base for building complex user images..

To build custom image, see Building customized image.

For the build of rootfs tarballs (available in the directory images) are used configurations:

```
chiliboard_nand256_ram128_defconfig
chiliboard_nand256_ram128_qt4_defconfig
```

chiliboard_nand256_ram128_qt5_defconfig

# Buildroot SDK/Building customized image

This document describes how to build your own system image using Buildroot. This description is based on the latest chili Buildroot SDK.

## How to build customized system image

- Download the latest chili-buildroot-sdk*.tar.bz2 archive from ftp server and unpack it. For details, see Getting started. Or clone chili-buildroot from the git server. This is the best way because the git repository contains the latest sources.

```
$ git clone git://git.grinn-global.com/chili-buildroot
```

- Change the current directory by:

```
$ cd buildroot_sdk_path/chili-buildroot*
```

  or (git clone):

```
$ cd chili-buildroot
```

- Select the basic configuration of the Buildroot. Currently, the following configurations are defined:

```
chiliboard_nand0_ram128_defconfig
chiliboard_nand256_ram128_defconfig
chiliboard_nand256_ram256_defconfig
chiliboard_nand256_ram256_lcd_defconfig
chiliboard_nand512_ram512_defconfig
chiliboard_nand512_ram512_lcd_defconfig
```

  There are also defined two complex configurations using Qt library:

```
chiliboard_nand256_ram128_qt4_defconfig
chiliboard_nand256_ram128_qt5_defconfig
```

  Configuration details can be found in SDK Content. The appropriate configuration should be selected by command (eg. for 256MB NAND and 128 MB RAM):

```
$ make chiliboard_nand256_ram256_defconfig
```

- Then modify the build options and select the required packages using:

```
$ make menuconfig
```

After modification, save the configuration.

- Finally, build all:

```
$ make
```

- By default, all final files will be created in the directory:

```
buildroot_path/output/images/
```

How to use these files, see Getting started.

## Default system configuration

Below are shown the Buildroot settings used in all configurations for chiliboard.

**System configuration**

```
Init system --> systemd
*** /dev management using udev (from systemd) ***
```

**Target packages**

```
Hardware handling --->
[*]   dbus
[*]   evtest
[*]   i2c-tools
```

```
Filesystem and flash utilities -->
[*] mtd, jffs2 and ubi/ubifs tools
      *** MTD tools selection ***
[ ]   docfdisk (NEW)
[ ]   doc_loadbios (NEW)
[*]   flashcp (NEW)
[*]   flash_erase (NEW)
[*]   flash_lock (NEW)
[ ]   flash_otp_dump (NEW)
[ ]   flash_otp_info (NEW)
[ ]   flash_otp_lock (NEW)
[ ]   flash_otp_write (NEW)
[*]   flash_unlock (NEW)
[ ]   ftl_check (NEW)
[ ]   ftl_format (NEW)
[ ]   jffs2dump (NEW)
[*]   mkfs.jffs2
[*]   mkfs.ubifs
[*]   mtd_debug (NEW)
[*]   nanddump (NEW)
[*]   nandtest (NEW)
[*]   nandwrite (NEW)
[ ]   nftldump (NEW)
[ ]   nftl_format (NEW)
[ ]   recv_image (NEW)
[ ]   rfddump (NEW)
[ ]   rfdformat (NEW)
[ ]   serve_image (NEW)
[ ]   sumtool (NEW)
[*]   mtdinfo (NEW)
[*]   ubiattach (NEW)
[*]   ubicrc32 (NEW)
[*]   ubidetach (NEW)
[*]   ubiformat (NEW)
[*]   ubimkvol (NEW)
[*]   ubinfo (NEW)
[*]   ubinize (NEW)
[*]   ubirename (NEW)
[*]   ubirmvol (NEW)
[*]   ubirsvol (NEW)
[*]   ubiupdatevol (NEW)
[*]   ubiblock (NEW)
```

## Additional packages used in images with Qt library

Touchscreen access library.

**Target packages**

```
Libraries --->
 Hardware handling  --->
  [*]   tslib
```

Support for hardware graphics accelerator SGX530 (used with Qt5).

**Target packages**

```
Hardware handling --->
 [*] ti-gfx
 [ ]   enable debug support
 [*]   install demos
 [*]   install eglimage version of libraries
       Target (es8.x (AM335x))  --->
```

SSH client and server.

**Target packages**

```
Networking applications --->
 [*] dropbear
 [*]   client programs
 [ ]   disable reverse DNS lookups
 [*]   optimize for size
 [ ]   log dropbear access to wtmp
 [ ]   log dropbear access to lastlog
```

ALSA support.

**Target packages**

```
Audio and video applications  --->
 [*] als-utils
     ALSA utils selection
 [*]    alsaconf
 [*]    alsactl
 [*]    alsamixer
 [*]    amixer
 [*]    aplay/arecord
 [*]    speaker-test
```

## Configuration for Qt4 library

**Target packages**

```
Graphic libraries and applications (graphic/text)  --->
 Qt --->
--- Qt
      Qt installation (Qt embedded)  --->
[ ]   Compile with debug support
[*]   Compile and install Qt demos (with code)
[ ]   Install translation files
[*]   Compile and install Qt examples (with code)
      Library type (Shared library)  --->
[*]   Approve free license
()    Config file
[ ]   Compatibility with Qt3
-*-   Gui Module
        Pixel depths  --->
        Fonts  --->
        freetype2 support (no freetype2 support)  --->
[*]     Enable GIF support
[ ]     Enable libmng support
        JPEG support (Use Qt bundled libjpeg)  --->
        PNG support (Use Qt bundled libpng)  --->
        TIFF support (Use Qt bundled libtiff)  --->
      zlib support (Qt zlib)  --->
[ ]   SQL Module  ----
      Graphics drivers  --->
         [*] Linux Framebuffer
         [ ] Transformed
         [ ] Qt Virtual Framebuffer
         [ ] VNC
         [ ] multiscreen
             *** directfb Qt driver not available (need directfb) ***
      Mouse drivers  --->
         [ ] pc
         [ ] linuxtp
         [*] linux input
         [*] tslib
         [ ] qvfb
              *** Mouse Options ***
         [ ] Hide the mouse cursor
      Keyboard drivers  --->
      *** Phonon module needs gstreamer ***
[ ]   DBus Module
[*]   XML Module
[ ]     XML Patterns Module
[ ]   Multimedia Module
[ ]   SVG Module
-*-   Network Module
[*]   WebKit Module
[*]   STL support
[ ]   Enable OpenSSL support
[*]   Script Module
[ ]   Script Tools Module
[ ]   Test Module
```

## Configuration for Qt5 library

**Target packages**

```
Graphic libraries and applications (graphic/text)  --->
 Qt5 --->
--- Qt5
-*-   qt5base
[*]     Approve free license
[*]     Compile and install examples (with code)
-*-     concurrent module
[*]     MySQL Plugin
[*]     PostgreSQL Plugin
    SQLite 3 support (Qt SQLite)  --->
-*-     gui module
-*-       widgets module
-*-       OpenGL support
            OpenGL API (OpenGL ES 2.0+)  --->
[*]         opengl module
[*]       linuxfb support
[ ]       directfb support
          *** X.org XCB backend available if X.org is enabled ***
[*]       eglfs support
(linuxfb) Default graphical platform
[ ]       fontconfig support
[*]       GIF support
[*]       JPEG support
[*]       PNG support
-*-     DBus module
-*-     Enable ICU support
[*]     Enable Tslib support
[*]   qt5connectivity
-*-   qt5declarative
-*-     quick module
[*]   qt5enginio
[*]   qt5graphicaleffects
[*]   qt5imageformats
[*]   qt5multimedia
[*]   qt5quick1
[*]   qt5quickcontrols
-*-   qt5script
[*]   qt5sensors
[*]   qt5serialport
[*]   qt5svg
[*]   qt5webkit
[*]     qt5webkit examples
[*]   qt5websockets
```

-*- qt5xmlpatterns

# Buildroot SDK/Cross-compilation toolchain

This document describes how to create a cross-compilation toolchain, and how to compile your own programs using the Buildroot tools.

## Buildroot toolchain

To compile your own programs first you need toolchain and set of libraries generated by Buildroot. The toolchain uses as a sysroot the target filesystem generated during building of the rootfs image. Therefore, you must first build rootfs with all the required packages (libraries). For details of how to build a rootfs image, see Building customized image.

By default toolchain is located in directory:

```
buildroot_path/output/host/
```

## Compile your own programs

The simplest way to use the Buildroot toolchain for building your own applications is to add **buildroot_path/output/host/usr/bin/** to your **PATH** environment variable and then to use **arm-buildroot-linux-gnueabihf-gcc**, **arm-buildroot-linux-gnueabihf-objdump**, **arm-buildroot-linux-gnueabihf-ld**, etc.

It is possible to relocate the toolchain - but then --sysroot must be passed every time the compiler is called to tell where the libraries and header files are.

To set the environment variables to build your own application, in some cases, may be useful the script:

```
SDK_PATH="buildroot_path/output/host"
export SDK_PATH_NATIVE=$SDK_PATH
```

```
export
SDK_PATH_TARGET=$SDK_PATH//usr/arm-buildroot-linux-gnueabihf/sysroot
export PATH=$SDK_PATH_NATIVE/usr/bin:$PATH
export TOOLCHAIN_PREFIX=arm-buildroot-linux-gnueabihf-
export CPATH=$SDK_PATH_TARGET/usr/include:$CPATH
export PKG_CONFIG_SYSROOT_DIR=$SDK_PATH_TARGET
export PKG_CONFIG_PATH=$SDK_PATH_TARGET/usr/lib/pkgconfig
export PKG_CONFIG_ALLOW_SYSTEM_LIBS=1
export CC=${TOOLCHAIN_PREFIX}gcc
export CXX=${TOOLCHAIN_PREFIX}g++
export GDB=${TOOLCHAIN_PREFIX}gdb
export CPP="${TOOLCHAIN_PREFIX}gcc -E"
export NM=${TOOLCHAIN_PREFIX}nm
export AS=${TOOLCHAIN_PREFIX}as
export AR=${TOOLCHAIN_PREFIX}ar
export RANLIB=${TOOLCHAIN_PREFIX}ranlib
export OBJCOPY=${TOOLCHAIN_PREFIX}objcopy
export OBJDUMP=${TOOLCHAIN_PREFIX}objdump
export STRIP=${TOOLCHAIN_PREFIX}strip
export CONFIGURE_FLAGS="--target=arm-buildroot-linux-gnueabihf
--host=arm-buildroot-linux-gnueabihf --build=i686-linux
--with-libtool-sysroot=$SDK_PATH_TARGET"
export CPPFLAGS=" -march=armv7-a -marm -mthumb-interwork
-mfloat-abi=hard -mfpu=neon -mtune=cortex-a8
--sysroot=$SDK_PATH_TARGET"
export CFLAGS="$CPPFLAGS"
export CXXFLAGS="$CPPFLAGS"
```

export LDFLAGS=" --sysroot=$SDK_PATH_TARGET"

# Buildroot SDK/Using Qt4 and Qt5 library

- To run programs using Qt4 or Qt5 should be built rootfs with Qt4 or Qt5 library. How to configure buildroot with Qt see Building customized image. Libraries Qt4 and Qt5 can not be used simultaneously on the same rootfs.
- To see how to run programs using Qt open LCD cape.
- The method of compiling user programs using Buildroot toolchain is described in Cross-compilation toolchain.

# Buildroot SDK/Linux kernel development

## Toolchain for kernel development

To develop the Linux kernel in the Buildroot tree first build all system, or only kernel using command:

```
$ make linux
```

Next you can modify the kernel sources, which are located in the directory:

```
buildroot_path/output/build/linux-custom
```

To work with kernel sources export the path to the Buildroot toolchain:

```
$ export PATH=$PATH:buildroot_path/output/host/usr/bin
```

and change the current directory:

```
$ cd buildroot_path/output/build/linux-custom
```

## Useful commands for working with kernel

Below are listed the most useful commands used for working with kernel.

Removing object files:

```
$ make ARCH=arm CROSS_COMPILE=arm-buildroot-linux-gnueabihf- clean
```

Configuration:

```
$ make ARCH=arm CROSS_COMPILE=arm-buildroot-linux-gnueabihf- menuconfig
```

Building:

```
$ make ARCH=arm CROSS_COMPILE=arm-buildroot-linux-gnueabihf-
```

Installation of modules:

```
$ make ARCH=arm CROSS_COMPILE=arm-buildroot-linux-gnueabihf- INSTALL_MOD_PATH=./modules_path modules_install
```

Compiling dts file: $ make ARCH=arm CROSS_COMPILE=arm-buildroot-linux-gnueabihf- am335x-chiliboard-nand256-ram128.dtb

# Article Sources and Contributors

**Main Page** *Source*: http://mediawikigg/wiki/index.php?oldid=55 *Contributors*: 1 anonymous edits

**Chiliboard/Getting started** *Source*: http://mediawikigg/wiki/index.php?oldid=51 *Contributors*: 8 anonymous edits

**Chiliboard/Using GPIO and LED** *Source*: http://mediawikigg/wiki/index.php?oldid=52 *Contributors*: 2 anonymous edits

**Chiliboard/LCD cape** *Source*: http://mediawikigg/wiki/index.php?oldid=53 *Contributors*: 3 anonymous edits

**Buildroot SDK/SDK content** *Source*: http://mediawikigg/wiki/index.php?oldid=47 *Contributors*: 11 anonymous edits

**Buildroot SDK/Building customized image** *Source*: http://mediawikigg/wiki/index.php?oldid=48 *Contributors*: 3 anonymous edits

**Buildroot SDK/Cross-compilation toolchain** *Source*: http://mediawikigg/wiki/index.php?oldid=39 *Contributors*: 1 anonymous edits

**Buildroot SDK/Using Qt4 and Qt5 library** *Source*: http://mediawikigg/wiki/index.php?oldid=49 *Contributors*: 2 anonymous edits

**Buildroot SDK/Linux kernel development** *Source*: http://mediawikigg/wiki/index.php?oldid=50 *Contributors*: 5 anonymous edits