How to cross-compile with the Distribution Package

Contents

1 Article purpose

This article provides simple examples for the Distribution Package of the OpenSTLinux distribution, that illustrate the cross-compilation with the devtool and BitBake tools:

- modification with Linux® Kernel (configuration, device tree, driver, ...)
- modification of an external in-tree Linux Kernel module
- modification of U-Boot
- modification of TF-A
- addition of software

These examples also show how to deploy the results of the cross-compilation on the target, through a network connection to the host machine.

All the examples described on this page use devtool and/or bitbake from OpenEmbededded, see OpenEmbedded - devtool and BitBake cheat sheet for more information.

There are many ways to achieve the same result; this article aims to provide at least one solution per example. You have the liberty to explore other methods that are better adapted to your development constraints.

2 Prerequisites

The prerequisites from Installing the OpenSTLinux distribution must be executed.

The board and the host machine are connected through an Ethernet link, and a remote terminal program is started on the host machine: see How to get Terminal.

The target is started, and its IP address (<board ip address>) is known.

3 Modification with kernel

3.1 Preamble

To start modification with a module, you need to initialize your Distribution Package environment.

```
PC $> cd <working directory path of distribution>
PC $> DISTRO=openstlinux-weston MACHINE=stm32mp1 source layers/meta-st/scripts/envsetup.sh
```

You are now in the build directory, identified by <build dir> in the following paragraphs.

Initialize devtool for kernel component:

```
PC $> devtool modify virtual/kernel
NOTE: Starting bitbake server...
NOTE: Creating workspace layer in /mnt/internal_storage/oetest/oe_openstlinux_rocko/build-
openstlinuxweston-stm32mp1/workspace
NOTE: Enabling workspace layer in bblayers.conf
Parsing recipes: 100%
|##################################################################################|
Time: 0:00:54
Parsing of 2401 .bb files complete (0 cached, 2401 parsed). 3282 targets, 88 skipped, 0
masked, 0 errors.
NOTE: Mapping virtual/kernel to linux-stm32mp
NOTE: Resolving any missing task queue dependencies
...
```

For the case of virtual/<something> component, you need to get the name of mapping between virtual component and associated recipe. In this example, the name of kernel recipe is indicated in the trace, but you can also get it by calling **devtool status**

A minority of devtool command supports the virtual/<something> component, like devtool modify, it is why you need to get the recipe name associated to virtual/component.
In this example, the name of kernel recipe is indicated in the trace (*linux-stm32mp*)

The source code of kernel is located on <build dir>/workspace/sources. You can change the path where the source code is extracted by customizing the devtool modify command

For all the work around the kernel, we strongly encourage some usage for deploying the binaries on board

- Kernel image, device tree: use bitbake deploy command and scp (see #modifying kernel configuration))
- Kernel module: use devtool deploy-target by passing by a temporary directory (see #Modifying/adding an external Linux kernel module)

The difference of usage comes from the number of files to deploy on board. When there is only one or two files to put on board, the easiest way to do it is to deploy only the desired files .

3.2 Modifying kernel configuration

This simple example modifies the kernel configuration via menuconfig for the CMA size.

- Get the current value of the CMA size (128 Mbytes here) through the analysis of the target boot log

```
Board $> dmesg | grep -i cma
[    0.000000] cma: Reserved 128 MiB at 0xf0000000
```

- Start the Linux kernel configuration menu: see Menuconfig or how to configure kernel
- Navigate to "Device Drivers - Generic Driver Options"
  - select "Size in Megabytes"
  - modify its value to 256
  - exit and save the new configuration
- Check that the configuration file (*.config*) has been modified

```
PC $> grep -i CONFIG_CMA_SIZE_MBYTES <build dir>/workspace/sources/<name of kernel
recipe>/.config.new
CONFIG_CMA_SIZE_MBYTES=256
```

- Cross-compile the Linux kernel: see Menuconfig or how to configure kernel
- Update the Linux kernel image on board: see Menuconfig or how to configure kernel
- Reboot the board: see Menuconfig or how to configure kernel
- Get the new value of the CMA size (256 Mbytes) through the analysis of the target boot log

```
Board $> dmesg | grep -i cma
[    0.000000] cma: Reserved 256 MiB at 0xe0000000
```

3.3 Modifying the Linux kernel device tree

This simple example modifies the default status of a user LED.

- With the board started; check that the user LED (LD3) is disabled
- Go to the <build dir>/workspace/sources/<name of kernel recipe>/ directory

```
PC $> cd <build dir>/workspace/sources/<name of kernel recipe>/
```

- Edit the *arch/arm/boot/dts/stm32mp157c-ed1.dts* Device Tree Source file for evaluation board or

Edit the *arch/arm/boot/dts/stm32mp157c-dk2.dts* Device Tree Source file for discovery board or

- Change the status of the "stm32mp:green:user" led to "okay", and set its default state to "on"

```
        led {
                compatible = "gpio-leds";

                status = "okay";
                red {
                        label = "stm32mp:red:status";
                        gpios = <&gpioa 13 GPIO_ACTIVE_LOW>;
```

```
                        status = "disabled";
                };
                green {
                        label = "stm32mp:green:user";
                        gpios = <&gpioa 14 GPIO_ACTIVE_LOW>;
                        default-state = "on";

                        status = "okay";
                };
        };
```

- Go to the build directory

```
PC $> cd <build dir>
```

If this update is accepted, then to report in other paragraph of this page}}

- Generate the device tree blobs (**.dtb*)

```
PC $> bitbake virtual/kernel -C compile
```

> we use here bitbake command instead of **devtool build** because the **build** makes **compile**, **compile_kernemodules** and **install** commands whereas we only need only **compile** command to generate the kernel image and the device tree

- Update the device tree blobs on the board

```
PC $> scp <build dir>/tmp-glibc/deploy/images/<machine name>/*.dtb root@<board ip
address>:/boot
```

> If the */boot* mounting point doesn't exist yet, please see how to create a mounting point

- Reboot the board

```
Board $> cd /boot; sync; systemctl reboot
```

- Check that the user LED (LD3) is **enabled** (green)

3.4 Modifying a built-in Linux kernel device driver

This simple example adds unconditional log information when the display driver is probed.

- Check that there is no log information when the display driver is probed

```
Board $> dmesg | grep -i stm_drm_platform_probe
Board $>
```

- Go to the <build dir>/workspace/sources/<name of kernel recipe>/

```
PC $> cd <build dir>/workspace/sources/<name of kernel recipe>/
```

- Edit the *./drivers/gpu/drm/stm/drv.c* source file
- Add a log information in the *stm_drm_platform_probe* function

```
static int stm_drm_platform_probe(struct platform_device *pdev)
{
        struct device *dev = &pdev->dev;
        struct drm_device *ddev;
        int ret;
        [...]

        DRM_INFO("Simple example - %s\n", __func__);

        return 0;
        [...]
}
```

- Go to the build directory

```
PC $> cd <build dir>
```

- Cross-compile the Linux kernel

```
PC $> bitbake virtual/kernel -C compile
```

- Update the Linux kernel image on board

```
PC $> scp <build dir>/tmp-glibc/deploy/images/<machine name>/uImage root@<board ip
address>:/boot
```

If the */boot* mounting point doesn't exist yet, please see how to create a mounting point

- Reboot the board

```
Board $> cd /boot; sync; systemctl reboot
```

- Check that there is now log information when the display driver is probed

```
Board $> dmesg | grep -i stm_drm_platform_probe
[    5.005833] [drm] Simple example - stm_drm_platform_probe
```

3.5 Modifying/adding an external Linux kernel module

Most device drivers (modules) in the Linux kernel can be compiled either into the kernel itself (built-in/internal module) or as Loadable Kernel Modules (LKM/external module) that need to be placed in the root file system under the /lib/modules directory. An external module can be in-tree (in the kernel tree structure), or out-of-tree (outside the kernel tree structure).

This simple example adds an unconditional log information when the virtual video test driver (vivid) kernel module is probed or removed.

- Go to the <build dir>/workspace/sources/<name of kernel recipe>/

```
PC $> cd <build dir>/workspace/sources/<name of kernel recipe>/
```

- Edit the *./drivers/media/platform/vivid/vivid-core.c* source file
- Add log information in the *vivid_probe* and *vivid_remove* functions

```
static int vivid_probe(struct platform_device *pdev)
{
        const struct font_desc *font = find_font("VGA8x16");
        int ret = 0, i;
        [...]

        /* n_devs will reflect the actual number of allocated devices */
        n_devs = i;

        pr_info("Simple example - %s\n", __func__);

        return ret;
}
```

```
static int vivid_remove(struct platform_device *pdev)
{
        struct vivid_dev *dev;
        unsigned int i, j;
        [...]

        pr_info("Simple example - %s\n", __func__);

        return 0;
}
```

- Go to the build directory

```
PC $> cd <build dir>
```

- Cross-compile the Linux kernel modules

```
PC $> bitbake virtual/kernel -C compile
```

- Update the vivid kernel module on the board

```
PC $> devtool deploy-target -Ss <name of kernel recipe> root@<board ip address>:/
```

Take care to have enought place available on board to receive the binaries deployed via **deploy-target** command.

The binaries (mainly kernel modules) are not stripped on **deploy-target** command and take a lot of place.

- Update dependency descriptions for loadable kernel modules, and synchronize the data on disk with memory

```
Board $> /sbin/depmod -a
Board $> sync
```

- Insert the vivid kernel module into the Linux kernel

```
Board $> modprobe vivid
[...]
[ 3412.784638] Simple example - vivid_probe
```

- Remove the vivid kernel module from the Linux kernel

```
Board $> rmmod vivid
[...]
[ 3423.708517] Simple example - vivid_remove
```

4 Adding an external out-of-tree Linux kernel module

This simple example adds a "Hello World" external out-of-tree Linux kernel module to the Linux kernel.

- Create a directory for this kernel module example

```
PC $> mkdir kernel_module_example
PC $> cd kernel_module_example
```

- Create the source code file for this kernel module example: *kernel_module_example.c*

```
// SPDX-identifier: GPL-2.0
/*
 * Copyright (C) STMicroelectronics SA 2018
 *
 * Authors: Jean-Christophe Trotin <jean-christophe.trotin@st.com>
 *
 */

#include <linux/module.h>    /* for all kernel modules */
#include <linux/kernel.h>    /* for KERN_INFO */
#include <linux/init.h>      /* for __init and __exit macros */

static int __init kernel_module_example_init(void)
{
        printk(KERN_INFO "Kernel module example: hello world from STMicroelectronics\n");
        return 0;
}

static void __exit kernel_module_example_exit(void)
{
        printk(KERN_INFO "Kernel module example: goodbye from STMicroelectronics\n");
}

module_init(kernel_module_example_init);
module_exit(kernel_module_example_exit);

MODULE_DESCRIPTION("STMicroelectronics simple external out-of-tree Linux kernel module
example");
MODULE_AUTHOR("Jean-Christophe Trotin <jean-christophe.trotin@st.com>");
MODULE_LICENSE("GPL v2");
```

- Create the makefile for this kernel module example: *Makefile*

All the indentations in a makefile are tabulations

```
# Makefile for simple external out-of-tree Linux kernel module example

# Object file(s) to be built
obj-m := kernel_module_example.o

# Path to the directory that contains the Linux kernel source code
# and the configuration file (.config)
```

```
KERNEL_DIR ?= <Linux kernel path>

# Path to the directory that contains the source file(s) to compile
PWD := $(shell pwd)

default:
        $(MAKE) -C $(KERNEL_DIR) M=$(PWD) modules

clean:
        $(MAKE) -C $(KERNEL_DIR) M=$(PWD) clean
```

- Add a new recipe to the workspace

```
PC $> cd <build dir>
PC $> devtool add mymodule kernel_module_example/
```

- Adapt recipe to kernel module build

```
PC $> devtool edit-recipe mymodule
```

Modify the recipe according the following changes (see highlighted lines)

```
 1 # Recipe created by recipetool
 2 # This is the basis of a recipe and may need further editing in order to be fully
functional.
 3 # (Feel free to remove these comments when editing.)
 4
 5 # Unable to find any files that looked like license statements. Check the accompanying
 6 # documentation and source headers and set LICENSE and LIC_FILES_CHKSUM accordingly.
 7 #
 8 # NOTE: LICENSE is being set to "CLOSED" to allow you to at least start building - if
 9 # this is not accurate with respect to the licensing of the software being built (it
10 # will not be in most cases) you must specify the correct value before using this
11 # recipe for anything other than initial testing/development!
12 LICENSE = "CLOSED"
13 LIC_FILES_CHKSUM = ""
14
15 # No information for SRC_URI yet (only an external source tree was specified)
16 SRC_URI = ""
17
18 # NOTE: this is a Makefile-only piece of software, so we cannot generate much of the
19 # recipe automatically - you will need to examine the Makefile yourself and ensure
20 # that the appropriate arguments are passed in.
21 DEPENDS = "virtual/kernel"
22 inherit module
23
24 EXTRA_OEMAKE  = "ARCH=arm"
25 EXTRA_OEMAKE += "KERNEL_DIR=${STAGING_KERNEL_BUILDDIR}"
26
27 S = "${WORKDIR}"
28
29 do_configure () {
30         # Specify any needed configure commands here
31         :
32 }
33
34 do_compile () {
35          # You will almost certainly need to add additional arguments here
36         oe_runmake
37 }
```

```
38
39 do_install () {
40         # NOTE: unable to determine what to put here - there is a Makefile but no
41         # target named "install", so you will need to define this yourself
42         install -d ${D}/lib/modules/${KERNEL_VERSION}
43         install -m 0755 ${B}/kernel_module_example.ko ${D}/lib/modules/${KERNEL_VERSION
}/
44 }
```

- Go to the build directory

```
PC $> cd <build dir>
```

- Generated kernel module example

```
PC $> devtool build mymodule
```

- Push this kernel module example on board

```
PC $> devtool deploy-target -Ss mymodule root@<board ip address>
```

- Update dependency descriptions for loadable kernel modules, and synchronize the data on disk with memory

```
Board $> /sbin/depmod -a
Board $> sync
```

- Insert the kernel module example into the Linux kernel

```
Board $> modprobe kernel_module_example
[18167.821725] Kernel module example: hello world from STMicroelectronics
```

- Remove the kernel module example from the Linux kernel

```
Board $> rmmod kernel_module_example
[18180.086722] Kernel module example: goodbye from STMicroelectronics
```

5 Modifying the U-Boot

This simple example adds unconditional log information when U-Boot starts. Within the scope of the trusted boot chain, U-Boot is used as second stage boot loader (SSBL).

- Have a look at the U-Boot log information when the board reboots

```
Board $> reboot
[...]
U-Boot <U-Boot version>

CPU: STM32MP1 rev1.0
Model: STMicroelectronics STM32MP157C [...]
Board: stm32mp1 in trusted mode
[...]
```

- Go to the build directory

```
PC $> cd <build dir>
```

- Search U-boot recipe

```
PC $> devtool search u-boot*
u-boot-stm32mp-extlinux  Generate 'extlinux.conf' file for U-boot
u-boot-stm32mp        Universal Boot Loader for embedded devices for stm32mp
```

On this example, the recipe name is **u-boot-stm32mp**

- Start to work with u-boot

```
PC $> devtool modify u-boot-stm32mp
```

```
Example:
PC $> cd <build dir>/workspace/sources/u-boot-stm32mp
```

- Edit the *./board/st/stm32mp1/stm32mp1.c* source file
- Add a log information in the *checkboard* function

```
int checkboard(void)
{
        char *mode;

        [...]

        printf("Board: stm32mp1 in %s mode\n", mode);
        printf("U-Boot simple example\n");

        return 0;
}
```

- Cross-compile the U-Boot: trusted boot

```
PC $> devtool build u-boot-stm32mp
PC $> bitbake u-boot-stm32mp -c deploy
```

- Go to the directory in which the compilation results are stored

```
PC $> cd <build dir>/tmp-glibc/deploy/images/<machine name>/
```

- Reboot the board, and hit any key to stop in the U-boot shell

```
Board $> reboot
[...]
Hit any key to stop autoboot:  0
STM32MP>
```

- Connect a USB cable between the host machine and the board via the USB OTG ports
- In the U-Boot shell, call the USB mass storage function

```
STM32MP> ums 0 mmc 0
```

for more information about the usage of U-Boot UMS functionality, see How to use USB mass storage in U-Boot

- On the host machine, check the partition associated with the secondary stage boot loader (*ssbl*): *sdc3* here

```
PC $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 10 Jan 17 18:05 bootfs -> ../../sdc4
lrwxrwxrwx 1 root root 10 Jan 17 18:05 fsbl1 -> ../../sdc1
lrwxrwxrwx 1 root root 10 Jan 17 18:05 fsbl2 -> ../../sdc2
lrwxrwxrwx 1 root root 10 Jan 17 18:05 rootfs -> ../../sdc5
lrwxrwxrwx 1 root root 10 Jan 17 18:05 ssbl -> ../../sdc3
lrwxrwxrwx 1 root root 10 Jan 17 18:05 userfs -> ../../sdc6
```

- Copy the binary (u-boot.stm32) to the dedicated partition

```
PC $> dd if=u-boot-<board name>-trusted.stm32 of=/dev/sdc3 bs=1M conv=fdatasync
```

(here u-boot-stm32mp157c-ev1-trusted.stm32 or u-boot-stm32mp157c-dk2-trusted.stm32)

- Reset the U-Boot shell

```
STM32MP> reset
```

- Have a look at the new U-Boot log information when the board reboots

```
[...]
U-Boot <U-Boot version>

CPU: STM32MP1 rev1.0
Model: STMicroelectronics STM32MP157C [...]
Board: stm32mp1 in trusted mode
U-Boot simple example
[...]
```

6 Modifying the TF-A

This simple example adds unconditional log information when the TF-A starts. Within the scope of the trusted boot chain, TF-A is used as first stage boot loader (FSBL).

- Have a look at the TF-A log information when the board reboots

```
Board $> reboot
[...]
INFO:      System reset generated by MPU (MPSYSRST)
INFO:    Using SDMMC
[...]
```

- Go to the build directory

```
PC $> cd <build dir>
```

- Search TF-A recipe

```
PC $> devtool search tf-a*
babeltrace            Babeltrace - Trace Format Babel Tower
libunistring          Library for manipulating C and Unicode strings
lttng-tools           Linux Trace Toolkit Control
gettext               Utilities and libraries for producing multi-lingual messages
glibc                 GLIBC (GNU C Library)
tf-a-stm32mp          Trusted Firmware-A for STM32MP1
gnutls                GNU Transport Layer Security Library
gstreamer1.0          GStreamer 1.0 multimedia framework
harfbuzz              Text shaping library
glibc-locale          Locale data from glibc
kbd                   Keytable files and keyboard utilities
```

On this example, the recipe name is **tf-a-stm32mp**

- Start to work with tf-a

```
PC $> devtool modify tf-a-stm32mp
```

- Go to <build dir>/workspace/sources/tf-a-stm32mp

```
PC $> cd <build dir>/workspace/sources/tf-a-stm32mp
```

- Edit the *./plat/st/stm32mp1/bl2_io_storage.c* source file
- Add a log information in the *stm32mp1_io_setup* function

```
void stm32mp1_io_setup(void)
{
        int io_result;
        [...]
```

```
        /* Add a trace about reset reason */
        print_reset_reason();

        INFO("TF-A simple example");

        [...]
}
```

- Cross-compile the TF-A

```
PC $> devtool build tf-a-stm32mp
PC $> bitbake tf-a-stm32mp -c deploy
```

- Go to the directory in which the compilation results are stored

```
PC $> cd <build dir>/tmp-glibc/deploy/images/<machine name>/
```

- Reboot the board, and hit any key to stop in the U-boot shell

```
Board $> reboot
[...]
Hit any key to stop autoboot:  0
STM32MP>
```

- Connect a USB cable between the host machine and the board via the USB OTG ports
- In the U-Boot shell, call the USB mass storage function

```
STM32MP> ums 0 mmc 0
```

for more information about the usage of U-boot ums functionality see How to use USB mass storage in U-Boot

- On the host machine, check the partition associated with the first stage boot loader (*fsbl1* and *fsbl2* as backup): *sdc1* and *sdc2* (as backup) here

```
PC $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 10 Jan 17 18:05 bootfs -> ../../sdc4
lrwxrwxrwx 1 root root 10 Jan 17 18:05 sfsbl1 -> ../../sdc1
lrwxrwxrwx 1 root root 10 Jan 17 18:05 sfsbl2 -> ../../sdc2
lrwxrwxrwx 1 root root 10 Jan 17 18:05 rootfs -> ../../sdc5
lrwxrwxrwx 1 root root 10 Jan 17 18:05 ssbl -> ../../sdc3
lrwxrwxrwx 1 root root 10 Jan 17 18:05 userfs -> ../../sdc6
```

- Copy the binary (tf-a-stm32mp157c-ev1-trusted.stm32) to the dedicated partition; to test the new TF-A binary, it might be useful to keep the old TF-A binary in the backup FSBL (*fsbl2*)

```
PC $> dd if=tf-a-<board name>-trusted.stm32 of=/dev/sdc1 bs=1M conv=fdatasync
```

(here tf-a-stm32mp157c-ev1-trusted.stm32 or tf-a-stm32mp157c-dk2-trusted.stm32)

- Reset the U-Boot shell

```
STM32MP> reset
```

- Have a look at the new TF-A log information when the board reboots

```
[...]
INFO:      System reset generated by MPU (MPSYSRST)
INFO:    TF-A simple example
INFO:    Using SDMMC
[...]
```

7 Adding a "hello world" user space example

This chapter shows how to compile and execute a simple "hello world" example.

- Create a directory for this user space example

```
PC $> mkdir hello_world_example
PC $> cd hello_world_example
```

- Create the source code file for this user space example: *hello_world_example.c*

```
// SPDX-identifier: GPL-2.0
/*
 * Copyright (C) STMicroelectronics SA 2018
 *
 * Authors: Jean-Christophe Trotin <jean-christophe.trotin@st.com>
 *
 */

#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv)
{
        int i =11;

        printf("\nUser space example: hello world from STMicroelectronics\n");
        setbuf(stdout,NULL);
        while (i--) {
                printf("%i ", i);
                sleep(1);
        }
        printf("\nUser space example: goodbye from STMicroelectronics\n");

        return(0);
}
```

- Add a new recipe to the workspace

```
PC $> cd <build dir>
PC $> devtool add myhelloword hello_world_example/
```

- Adapt recipe

```
PC $> devtool edit-recipe myhelloword
```

Modify the recipe according the following changes (see highlighted lines)

```
 1 # Recipe created by recipetool
 2 # This is the basis of a recipe and may need further editing in order to be fully
functional.
 3 # (Feel free to remove these comments when editing.)
 4
```

```
 5 # Unable to find any files that looked like license statements. Check the accompanying
 6 # documentation and source headers and set LICENSE and LIC_FILES_CHKSUM accordingly.
 7 #
 8 # NOTE: LICENSE is being set to "CLOSED" to allow you to at least start building - if
 9 # this is not accurate with respect to the licensing of the software being built (it
10 # will not be in most cases) you must specify the correct value before using this
11 # recipe for anything other than initial testing/development!
12 LICENSE = "CLOSED"
13 LIC_FILES_CHKSUM = ""
14
15 # No information for SRC_URI yet (only an external source tree was specified)
16 SRC_URI = ""
17
18 # NOTE: no Makefile found, unable to determine what needs to be done
19
20 do_configure () {
21         # Specify any needed configure commands here
22         :
23 }
24
25 do_compile () {
26         # Specify compilation commands here
27         cd ${S}
28         ${CC} hello_world_example.c -o hello_word_example
29 }
30
31 do_install () {
32         # Specify install commands here
33         install -d ${D}${bindir}
34         install -m 755 ${S}/hello_word_example  ${D}${bindir}/
35 }
```

- Compile binary

```
PC $> devtool build myhelloword
```

- Push this binary on board

```
PC $> devtool deploy-target -s myhelloword root@<board ip address>
```

- Execute this user space example

```
Board $> /usr/bin/hello_world_example

User space example: hello world from STMicroelectronics
10 9 8 7 6 5 4 3 2 1 0
User space example: goodbye from STMicroelectronics
```

8 Tips

8.1 Creating a mounting point

The objective is to create a mounting point for the boot file system (bootfs partition)

- Find the partition label associated with the boot file system

```
Board $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 15 Dec 13 12:31 bootfs -> ../../mmcblk0p4
lrwxrwxrwx 1 root root 15 Dec 13 12:31 fsbl1 -> ../../mmcblk0p1
lrwxrwxrwx 1 root root 15 Dec 13 12:31 fsbl2 -> ../../mmcblk0p2
lrwxrwxrwx 1 root root 15 Dec 13 12:31 rootfs -> ../../mmcblk0p5
lrwxrwxrwx 1 root root 15 Dec 13 12:31 ssbl -> ../../mmcblk0p3
lrwxrwxrwx 1 root root 15 Dec 13 12:31 userfs -> ../../mmcblk0p6
```

- Attach the boot file system found under */dev/mmcblk0p4* in the directory */boot*

```
Board $> mount /dev/mmcblk0p4 /boot
```

Linux® is a registered trademark of Linus Torvalds.

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Trusted Firmware for Arm® Cortex®-A

Light-emitting diode

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Direct Rendering Manager (kernel module that gives direct hardware access to DRI clients, find more information on official DRI web site http://dri.freedesktop.org/wiki/DRM)

Second Stage Boot Loader

Central processing unit

USB On-The-Go (Capability/type of USB port, acting primarily as USB device, to also act as USB host. Also known as USB OTG.)

User-space Mode Setting

First Stage Boot Loader

Microprocessor Unit