ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

Cross-compile with OpenSTLinux SDK

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

Contents

Stable: 16.01.2020 - 14:27 / Revision: 16.01.2020 - 14:23

Stable: 16.01.2020 - 14:27 / Revision: 16.01.2020 - 14:28

A quality version of this page, approved on *16 January 2020*, was based off this revision.

Contents

1 Article purpose

The pieces of software delivered as source code within the OpenSTLinux Developer Package (for example the Linux kernel) can be modified. External out-of-tree Linux kernel modules, and pieces of applicative software (for example Linux applications) can also be developed thanks to this Developer Package, and loaded onto the board.

The build of all these pieces of software by means of the SDK for OpenSTLinux distribution, and the deployment on-target of the resulting images is explained below.

Warning

To use the cross-compilation efficiently with the OpenSTLinux SDK, it is recommended that you read the Developer Package article relative to the Series of your STM32 microprocessor: Category: Developer Package

1.1 Modifying the Linux kernel

Prerequisites:

- the SDK is installed
- the SDK is started up
- the Linux kernel is installed

The *<Linux kernel installation directory>/README.HOW_TO.txt* helper file gives the commands to:

configure the Linux kernel

cross-compile the Linux kernel

deploy the Linux kernel (that is, update the software on board)

You can refer to the following simple examples:

- Modification of the kernel configuration
- Modification of the device tree
- Modification of a built-in device driver
- Modification of an external in-tree module

1.2 Adding external out-of-tree Linux kernel modules

Prerequisites:

- the SDK is installed
- the SDK is started up
- the Linux kernel is installed

Most device drivers (or modules) in the Linux kernel can be compiled either into the kernel itself (built-in, or internal module) or as Loadable Kernel Modules (LKMs, or external modules) 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).

External Linux kernel modules are compiled taking reference to a Linux kernel source tree and a Linux kernel configuration file (*.config*).
Thus, a makefile for an external Linux kernel module points to the Linux kernel directory that contains the source code and the configuration file, with the **"-C <Linux kernel path>"** option.
This makefile also points to the directory that contains the source file(s) of the Linux kernel module to compile, with the **"M=< Linux kernel module path>"** option.

A generic makefile for an external out-of-tree Linux kernel module looks like the following:

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

# Object file(s) to be built
obj-m := <module source file(s)>.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 generated objects
DESTDIR ?= <Linux kernel installation directory>

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

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

install:
	$(MAKE) -C $(KERNEL_DIR) M=$(PWD) INSTALL_MOD_PATH=$(DESTDIR) modules_install

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

Such module is then cross-compiled with the following commands:

```
$ make clean
$ make
$ make install
```

You can refer to the following simple example:

- Addition of an external out-of-tree module

1.3 Adding Linux user space applications

Prerequisites:

- the SDK is installed
- the SDK is started up

Once a suitable cross-toolchain (OpenSTLinux SDK) is installed, it is easy to develop a project outside of the OpenEmbedded build system.
There are different ways to use the SDK toolchain directly, among which Makefile and Autotools.
Whatever the method, it relies on:

- the sysroot that is associated with the cross-toolchain, and that contains the header files and libraries needed for generating binaries (see target sysroot)

- the environment variables created by the SDK environment setup script (see SDK startup)

You can refer to the following simple example:

- Addition of a "hello world" user space application

1.4 Modifying the U-Boot

Prerequisites:

- the SDK is installed
- the SDK is started up
- the U-Boot is installed

The *<U-Boot installation directory>/README.HOW_TO.txt* helper file gives the commands to:

cross-compile the U-Boot

deploy the U-Boot (that is, update the software on board)

You can refer to the following simple example:

- Modification of the U-Boot

1.5 Modifying the TF-A

Prerequisites:

- the SDK is installed
- the SDK is started up
- the TF-A is installed

The *<TF-A installation directory>/README.HOW_TO.txt* helper file gives the commands to:

cross-compile the TF-A

deploy the TF-A (that is, update the software on board)

You can refer to the following simple example:

- Modification of the TF-A

1.6 Modifying the OP-TEE

Prerequisites:

- the SDK is installed
- the SDK is started up
- the OP-TEE is installed

The *<OP-TEE installation directory>/README.HOW_TO.txt* helper file gives the commands to:

cross-compile the OP-TEE

deploy the OP-TEE (that is, update the software on board)

Linux® is a registered trademark of Linus Torvalds.

Software development kit (A programming package that enables a programmer to develop applications for a specific platform.)

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

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

Trusted Firmware for Arm Cortex-A

Open Portable Trusted Execution Environment

Stable: 17.06.2020 - 15:26 / Revision: 16.01.2020 - 13:43

A quality version of this page, approved on *17 June 2020*, was based off this revision.

This category groups together all articles related to a Developer Package (whatever the microprocessor device and the board).

The Developer Package is specified in the Which Package better suits your needs article.

Pages in category "Developer Package"

The following 3 pages are in this category, out of 3 total.

- How to cross-compile with the Developer Package
- STM32MP1 Developer Package
- STM32MP1 Developer Package for Android

Stable: 16.04.2021 - 10:21 / Revision: 16.04.2021 - 07:12

A quality version of this page, approved on *16 April 2021*, was based off this revision.

Contents

1 Article purpose

This article provides simple examples for the Developer Package of the OpenSTLinux distribution, that illustrate cross-compilation with the SDK:

- modification of software elements delivered as source code (for example the Linux kernel)
- addition of software (for example the Linux kernel module or user-space applications)

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

Information

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

2 Prerequisites

The prerequisites from the Cross-compile with OpenSTLinux SDK article must be executed, and the cross-compilation and deployment of any piece of software, as explained in that article, is known.

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.

Information

If you encounter a problem with any of the commands in this article, remember that the README. HOW_TO.txt helper files, from the Linux kernel, U-Boot and TF-A installation directories, are **the** build references.

Information

Regarding the Linux kernel examples, it is considered that the Linux kernel has been setup, configured and built a first time in a dedicated build directory (*<Linux kernel build directory>* later in this page) different from the source code directory (*<Linux kernel source directory>* later in this document).

3 Modifying the Linux kernel configuration

3.1 Preamble

Warning

Please read carefully and pay attention to the following point before modifying the Linux kernel configuration

The Linux kernel configuration option that you want to modify might be used by external out-of-tree Linux kernel modules (for example the GPU kernel driver), and these should then be recompiled. These modules are, by definition, outside the kernel tree structure, and are not delivered in the Developer Package source code; it is not possible to recompile them with the Developer Package. Consequently, if the Linux kernel is reconfigured and recompiled with this option then deployed on the board, the external out-of-tree Linux kernel modules might no longer be loaded.

There are two possible situations:

- This is not a problem for the use cases on which you are currently working. In this case you can use the Developer Package to modify and recompile the Linux kernel.
- This is a problem for the use cases on which you are currently working. In this case you need to switch on the STM32MP1 Distribution Package, and after having modified the Linux kernel configuration, use it to rebuild the whole image (that is, not only the Linux kernel but also the external out-of-tree Linux kernel modules).

Example:

- Let's assume that the FUNCTION_TRACER and FUNCTION_GRAPH_TRACER options are activated to install the ftrace Linux kernel feature
- This feature is used to add tracers in the whole kernel, including the external out-of-tree Linux kernel modules

1. The Developer Package is used to reconfigure and recompile the Linux kernel, and to deploy it on the board
    1. The external out-of-tree Linux kernel modules are not recompiled. This is the case for the GPU kernel driver
    2. Consequently, the Linux kernel fails to load the GPU kernel driver module. However, even if the display no longer works, the Linux kernel boot succeeds, and the setup is sufficient, for example, to debug use cases involving an Ethernet or USB connection
2. The Distribution Package is used to reconfigure the Linux kernel, and to rebuild and deploy the whole image on the board
    1. The external out-of-tree Linux kernel modules are recompiled, including the GPU kernel driver
    2. Consequently, the Linux kernel succeeds in loading the GPU kernel driver module. The display is available.

3.2 Simple example

This simple example modifies the value defined for the contiguous memory area (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
```

STM32MP157C-EV1

```
[    0.000000] cma: Reserved 128 MiB at 0xe8000000
```

STM32MP157C-DK2

```
[    0.000000] cma: Reserved 128 MiB at 0xd2000000
```

- Go to the <Linux kernel build directory>

```
PC $> cd <Linux kernel build directory>
```

- 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 .config
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
```

STM32MP157C-EV1

```
[    0.000000] cma: Reserved 256 MiB at 0xd8000000
```

STM32MP157C-DK2

```
[    0.000000] cma: Reserved 256 MiB at 0xc2000000
```

4 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 green LED (LD3 for STM32MP157C-EV1, LD5 for STM32MP157C-DK2) is disabled
- Go to the <Linux kernel source directory>

```
PC $> cd <Linux kernel source directory>
```

STM32P157C-EV1

- Edit the *arch/arm/boot/dts/stm32mp15xx-edx.dtsi* device tree source file
- Add the lines highlighted below

```
        led {
                compatible = "gpio-leds";
                blue {
                        label = "heartbeat";
                        gpios = <&gpiod 9 GPIO_ACTIVE_HIGH>;
                        linux,default-trigger = "heartbeat";
                        default-state = "off";
                };
                 green {
                         label = "stm32mp:green:user";
                         gpios = <&gpioa 14 GPIO_ACTIVE_LOW>;
                         default-state = "on";
                  };
        };
```

STM32MP157C-DK2

- Edit the *arch/arm/boot/dts/stm32mp15xx-dkx.dtsi* device tree source file
- Add the lines highlighted below

```
        led {
                compatible = "gpio-leds";
                blue {
                        label = "heartbeat";
                        gpios = <&gpiod 11 GPIO_ACTIVE_HIGH>;
                        linux,default-trigger = "heartbeat";
                        default-state = "off";
                };
                 green {
                         label = "stm32mp:green:user";
                         gpios = <&gpioa 14 GPIO_ACTIVE_LOW>;
                         default-state = "on";
                  };
        };
```

- Go to the <Linux kernel build directory>

```
PC $> cd <Linux kernel build directory>
```

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

```
PC $> make dtbs
PC $> cp arch/arm/boot/dts/stm32mp157*.dtb install_artifact/boot/
```

- Update the device tree blobs on the board

```
PC $> scp install_artifact/boot/stm32mp157*.dtb root@<board ip address>:/boot/
```

Information

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

- Reboot the board

```
Board $> reboot
```

- Check that the user green LED (LD3 for STM32MP157C-EV1, LD5 for STM32MP157C-DK2) is **enabled** (green)

5 Modifying a built-in Linux kernel device driver

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

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

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

- Go to the <Linux kernel source directory>

```
PC $> cd <Linux kernel source directory>
```

- 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 <Linux kernel build directory>

```
PC $> cd <Linux kernel build directory>
```

- Cross-compile the Linux kernel (please check the load address in the *README.HOW_TO.txt* helper file)

```
PC $> make uImage LOADADDR=0xC2000040
PC $> cp arch/arm/boot/uImage install_artifact/boot/
```

- Update the Linux kernel image on board

```
PC $> scp install_artifact/boot/uImage root@<board ip address>:/boot/
```

Information

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

- Reboot the board

```
Board $> reboot
```

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

```
Board $> dmesg | grep -i stm_drm_platform_probe
[    2.995125] [drm] Simple example - stm_drm_platform_probe
```

6 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).

6.1 Modifying an external in-tree Linux kernel module

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

- Go to the <Linux kernel source directory>

```
PC $> cd <Linux kernel source directory>
```

- 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 <Linux kernel build directory>

```
PC $> cd <Linux kernel build directory>
```

- Cross-compile the Linux kernel modules

```
PC $> make modules
PC $> make INSTALL_MOD_PATH="./install_artifact" modules_install
```

- Remove the link on "install_artifact/lib/modules/<kernel version>/"

```
PC $> rm install_artifact/lib/modules/<kernel version>/build
PC $> rm install_artifact/lib/modules/<kernel version>/source
```

- Optionally, strip kernel modules (to reduce the size of each kernel modules)

```
PC $>  find . -name "*.ko" | xargs $STRIP --strip-debug --remove-section=.comment --
remove-section=.note --preserve-dates
```

- Update the vivid kernel module on the board (please check the kernel version <kernel version>)

```
PC $> scp install_artifact/lib/modules/<kernel version>/kernel/drivers/media/platform
/vivid/vivid.ko root@<board ip address>:/lib/modules/<kernel version>/kernel/drivers/media
/platform/vivid/
```

OR

```
PC $> scp -r install_artifact/lib/modules/* root@<board ip address>:/lib/modules/
```

- 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
```

6.2 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.

- Prerequisite: the Linux source code is installed, and the Linux kernel has been cross-compiled
- Go to the working directory that contains all the source code (that is, the directory that contains the Linux kernel, U-Boot and TF-A source code directories)

```
PC $> cd <tag>/sources/arm-<distro>-linux-gnueabi
```

- Export to KERNEL_SRC_PATH the path to the Linux kernel build directory that contains both the Linux kernel source code and the configuration file (*.config*)

```
PC $> export KERNEL_SRC_PATH=$PWD/<Linux kernel build directory>/
```

Example:

**PC $>** export KERNEL_SRC_PATH=$PWD/linux-stm32mp-5.4.31/build

- 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*

Information

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 ?=  $(KERNEL_SRC_PATH)

# Path to the directory that contains the generated objects
DESTDIR ?= $(KERNEL_DIR)/install_artifact

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

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

install:
	$(MAKE) -C $(KERNEL_DIR) M=$(PWD) INSTALL_MOD_PATH=$(DESTDIR) modules_install

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

- Cross-compile the kernel module example

```
PC $> make clean
PC $> make
PC $> make install
```

- Go to the <Linux kernel build directory>

```
PC $> cd <Linux kernel build directory>
```

- The generated kernel module example is in: *install_artifact/lib/modules/<kernel version>/extra/kernel_module_example.ko*
- Remove the link on "install_artifact/lib/modules/<kernel version>/"

```
PC $> rm install_artifact/lib/modules/<kernel version>/build
PC $> rm install_artifact/lib/modules/<kernel version>/source
```

- Optionally, strip kernel modules (to reduce the size of each kernel modules)

```
PC $>  find . -name "*.ko" | xargs $STRIP --strip-debug --remove-section=.comment --
remove-section=.note --preserve-dates
```

- Push this kernel module example on board (please check the kernel version <kernel version>)

```
PC $> ssh root@<board ip address> mkdir -p /lib/modules/<kernel version>/extra
PC $> scp install_artifact/lib/modules/<kernel version>/extra/kernel_module_example.ko
root@<board ip address>:/lib/modules/<kernel version>/extra
```

OR

```
PC $> scp -r install_artifact/lib/modules/* root@<board ip address>:/lib/modules/
```

- 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
```

7 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
```

STM32MP157C-EV1

```
[...]
U-Boot 2020.01-stm32mp-r1 (Jan 06 2020 - 20:56:31 +0000)
CPU: STM32MP157CAA Rev.B
Model: STMicroelectronics STM32MP157C eval daughter on eval mother
Board: stm32mp1 in trusted mode (st,stm32mp157c-ev1)
[...]
```

STM32MP157C-DK2

```
[...]
U-Boot 2020.01-stm32mp-r1 (Jan 06 2020 - 20:56:31 +0000)
CPU: STM32MP157CAC Rev.B
Model: STMicroelectronics STM32MP157C-DK2 Discovery Board
Board: stm32mp1 in trusted mode (st,stm32mp157c-dk2)
[...]
```

- Go to the <U-Boot source directory>

```
PC $> cd <U-Boot source directory>
```

```
Example:
PC $> cd u-boot-stm32mp-2020.01-r0/u-boot-stm32mp-2020.01
```

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

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

        [...]
        puts("\n");
        printf("U-Boot simple example\n");
        [...]
```

```
        return 0;
}
```

- Get the list of supported configurations with the following command

```
PC $> make -f $PWD/../Makefile.sdk help
```

- Cross-compile the U-Boot: trusted boot for STM32MP157C-EV1 and STM32MP157C-DK2

```
PC $> make -f $PWD/../Makefile.sdk all UBOOT_CONFIGS=stm32mp15_trusted_defconfig,trusted,
u-boot.stm32
```

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

```
PC $> cd build-trusted/
```

- 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
```

Information

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 Feb  8 08:57 bootfs -> ../../sdb4
lrwxrwxrwx 1 root root 10 Feb  8 08:57 fsbl1 -> ../../sdb1
lrwxrwxrwx 1 root root 10 Feb  8 08:57 fsbl2 -> ../../sdb2
lrwxrwxrwx 1 root root 10 Feb  8 08:57 rootfs -> ../../sdb6
lrwxrwxrwx 1 root root 10 Feb  8 08:57 ssbl -> ../../sdb3
lrwxrwxrwx 1 root root 10 Feb  8 08:57 userfs -> ../../sdb7
lrwxrwxrwx 1 root root 10 Feb  8 08:57 vendorfs -> ../../sdb5
```

- Copy the U-Boot binary to the dedicated partition

STM32MP157C-EV1

```
PC $> dd if=u-boot-stm32mp157c-ev1-trusted.stm32 of=/dev/sdb3 bs=1M conv=fdatasync
```

STM32MP157C-DK2

```
PC $> dd if=u-boot-stm32mp157c-dk2-trusted.stm32 of=/dev/sdb3 bs=1M conv=fdatasync
```

- Reset the U-Boot shell

```
STM32MP> reset
```

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

STM32MP157C-EV1

```
[...]
U-Boot 2020.01-stm32mp-r1 (Jan 06 2020 - 20:56:31 +0000)
CPU: STM32MP157CAA Rev.B
Model: STMicroelectronics STM32MP157C eval daughter on eval mother
Board: stm32mp1 in trusted mode (st,stm32mp157c-ev1)
U-Boot simple example
[...]
```

STM32MP157C-DK2

```
[...]
U-Boot 2020.01-stm32mp-r1 (Jan 06 2020 - 20:56:31 +0000)
CPU: STM32MP157CAC Rev.B
Model: STMicroelectronics STM32MP157C-DK2 Discovery Board
Board: stm32mp1 in trusted mode (st,stm32mp157c-dk2)
U-Boot simple example
[...]
```

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

8 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:    PMIC version = 0x10
INFO:    Using SDMMC
[...]
```

- Go to the <TF-A source directory>

```
PC $> cd <TF-A source directory>
```

```
 Example:
 PC $> cd tf-a-stm32mp-2.2-r1-r0/tf-a-stm32mp-2.2.r1
```

- Edit the *./plat/st/stm32mp1/bl2_plat_setup.c* source file
- Add a log information in the *print_reset_reason* function

```
static void print_reset_reason(void)
{
        [...]
        INFO("Reset reason (0x%x):\n", rstsr);

        INFO("TF-A simple example\n");
        [...]
}
```

- Get the list of supported configurations with the following command

```
PC $> make -f $PWD/../Makefile.sdk help
```

- Cross-compile the TF-A: trusted boot for STM32MP157C-EV1 and STM32MP157C-DK2

```
PC $> make -f $PWD/../Makefile.sdk all TF_A_CONFIG=trusted
```

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

```
PC $> cd ../build/trusted
```

- 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
```

Information

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): *sdb1* and *sdb2* (as backup) here

```
PC $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 10 Feb  8 10:01 bootfs -> ../../sdb4
lrwxrwxrwx 1 root root 10 Feb  8 10:01 fsbl1 -> ../../sdb1
lrwxrwxrwx 1 root root 10 Feb  8 10:01 fsbl2 -> ../../sdb2
lrwxrwxrwx 1 root root 10 Feb  8 10:01 rootfs -> ../../sdb6
lrwxrwxrwx 1 root root 10 Feb  8 10:01 ssbl -> ../../sdb3
lrwxrwxrwx 1 root root 10 Feb  8 10:01 userfs -> ../../sdb7
lrwxrwxrwx 1 root root 10 Feb  8 10:01 vendorfs -> ../../sdb5
```

- Copy the TF-A binary 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*)

STM32MP157C-EV1

```
PC $> dd if=tf-a-stm32mp157c-ev1-trusted.stm32 of=/dev/sdb1 bs=1M conv=fdatasync
```

Information

In case you get a *permission denied* you can set more permission on */dev/sdb1* : *sudo chmod 777 /dev /sdb1*

STM32MP157C-DK2

```
PC $> dd if=tf-a-stm32mp157c-dk2-trusted.stm32 of=/dev/sdb1 bs=1M conv=fdatasync
```

- Reset the U-Boot shell

In the U-Boot shell, press Ctrl+C prior to get hand back.

```
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:    PMIC version = 0x10
INFO:    Using SDMMC
[...]
```

9 Adding a "hello world" user space example

Thanks to the OpenSTLinux SDK, it is easy to develop a project outside of the OpenEmbedded build system. This chapter shows how to compile and execute a simple "hello world" example.

9.1 Source code file

- Go to the working directory that contains all the source codes (i.e. directory that contains the Linux kernel, U-Boot and TF-A source code directories)

```
PC $> cd <tag>/sources/arm-<distro>-linux-gnueabi
```

- 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);
}
```

9.2 Cross-compilation

Three ways to use the OpenSTLinux SDK to cross-compile this user space example are proposed below: (1) command line (2) makefile-based project (3) autotools-based project.

9.2.1 Command line

This method allows quick cross-compilation of a single-source code file. It applies if the project has only one file.
The cross-development toolchain is associated with the sysroot that contains the header files and libraries needed for generating binaries that run on the target architecture (see SDK for OpenSTLinux distribution#Native and target sysroots).
The sysroot location is specified with the *--sysroot* option.

The sysroot location must be specified using the --sysroot option. The *CC* environment variable created by the SDK already includes the *--sysroot* option that points to the SDK sysroot location.

```
PC $> echo $CC
arm-ostl-linux-gnueabi-gcc -march=armv7ve -mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -
mcpu=cortex-a7 --sysroot=<SDK installation directory>/SDK/sysroots/cortexa7t2hf-neon-
vfpv4-ostl-linux-gnueabi
```

- Create the directory in which the generated binary is to be stored

```
PC $> mkdir -p install_artifact install_artifact/usr install_artifact/usr/local
install_artifact/usr/local/bin
```

- Cross-compile the single source code file for the user space example

```
PC $> $CC hello_world_example.c -o ./install_artifact/usr/local/bin/hello_world_example
```

9.2.2 Makefile-based project

For this method, the cross-toolchain environment variables established by running the cross-toolchain environment setup script are subject to general *make* rules.
For example, see the following environment variables:

```
PC $> echo $CC
arm-ostl-linux-gnueabi-gcc -march=armv7ve -mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -
mcpu=cortex-a7 --sysroot=<SDK installation directory>/SDK/sysroots/cortexa7t2hf-neon-
vfpv4-ostl-linux-gnueabi
PC $> echo $CFLAGS
-O2 -pipe -g -feliminate-unused-debug-types
PC $> echo $LDFLAGS
-Wl,-O1 -Wl,--hash-style=gnu -Wl,--as-needed
PC $> echo $LD
arm-ostl-linux-gnueabi-ld --sysroot=<SDK installation directory>/SDK/sysroots
/cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
```

- Create the makefile for this user space example: *Makefile*

Information

All the indentations in a makefile are tabulations

```
PROG = hello_world_example
SRCS = hello_world_example.c
OBJS = $(SRCS:.c=.o)
```

```
CLEANFILES = $(PROG)
INSTALL_DIR = ./install_artifact/usr/local/bin

# Add / change option in CFLAGS if needed
# CFLAGS += <new option>

$(PROG):  $(OBJS)
        $(CC) $(CFLAGS) -o $(PROG) $(OBJS)

.c.o:
        $(CC) $(CFLAGS) -c $< -o $@

all: $(PROG)


clean:
        rm -f $(CLEANFILES) $(patsubst %.c,%.o, $(SRCS)) *~

install: $(PROG)
        mkdir -p $(INSTALL_DIR)
        install $(PROG) $(INSTALL_DIR)
```

- Cross-compile the project

```
PC $> make
PC $> make install
```

9.2.3 Autotools-based project

This method creates a project based on GNU autotools.

- Create the makefile for this user space example: *Makefile.am*

```
bin_PROGRAMS = hello_world_example
hello_world_example_SOURCES = hello_world_example.c
```

- Create the configuration file for this user space example: *configure.ac*

```
AC_INIT(hello_world_example,0.1)
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CC
AC_PROG_INSTALL
AC_OUTPUT(Makefile)
```

- Generate the local aclocal.m4 files and create the configure script

```
PC $> aclocal
PC $> autoconf
```

- Generate the files needed by GNU coding standards (for compliance)

```
PC $> touch NEWS README AUTHORS ChangeLog
```

- Generate the links towards SDK scripts

```
PC $> automake -a
```

- Cross-compile the project

```
PC $> ./configure ${CONFIGURE_FLAGS}
PC $> make
PC $> make install DESTDIR=./install_artifact
```

9.3 Deploy and execute on board

- Check that the generated binary for this user space example is in: *./install_artifact/usr/local/bin/hello_world_example*
- Push this binary onto the board

```
PC $> scp -r install_artifact/* root@<board ip address>:/
```

- Execute this user space example

```
Board $> cd /usr/local/bin
Board $> ./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
```

10 Tips

10.1 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 Jan 23 17:00 bootfs -> ../../mmcblk0p4
lrwxrwxrwx 1 root root 15 Jan 23 17:00 fsbl1 -> ../../mmcblk0p1
lrwxrwxrwx 1 root root 15 Jan 23 17:00 fsbl2 -> ../../mmcblk0p2
lrwxrwxrwx 1 root root 15 Jan 23 17:00 rootfs -> ../../mmcblk0p6
lrwxrwxrwx 1 root root 15 Jan 23 17:00 ssbl -> ../../mmcblk0p3
lrwxrwxrwx 1 root root 15 Jan 23 17:00 userfs -> ../../mmcblk0p7
lrwxrwxrwx 1 root root 15 Jan 23 17:00 vendorfs -> ../../mmcblk0p5
```

- 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

Graphics Processing Units

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

Power Management Integrated Circuit

Software development kit (A programming package that enables a programmer to develop applications for a specific platform.)

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

A quality version of this page, approved on *7 January 2021*, was based off this revision.

Contents

1 Article purpose

This article aims to give **general information** about the software development kit (SDK) for the OpenSTLinux distribution.

Information

To install and use efficiently the last release of the OpenSTLinux SDK, please read the Developer Package article relative to the Series of your STM32 microprocessor: Category:Developer Package

2 Introduction

The **software development kit (SDK)** for the OpenSTLinux distribution is a customization of the Yocto SDK[1], which provides a **stand-alone cross-development toolchain** and libraries tailored to the contents of a specific image. The OpenSTLinux SDK is part of the STM32MPU Embedded Software Developer Package.

The **SDK might be generated**, through the STM32MPU Embedded Software Distribution Package, during the compilation of a software release, which guarantees the alignment of this SDK with the software images (binaries) built for the Starter Package of the STM32MPU Embedded Software: see SDK development cycle model.

It provides a more "traditional" toolchain experience than the full Yocto project (OpenEmbedded) development environment available through the Distribution Package of the STM32MPU Embedded Software.

It simplifies the workflow for application developers: it has no dependency on the Yocto project used for its generation (Distribution Package), and can be installed on any host machine. Note that many SDKs can coexist on the same host machine.

3 Why use the SDK, and how?

The OpenSTLinux SDK gives developers an efficient development cycle (compilation, deployment on target, and debug).

Using this SDK, developers take advantage of the Yocto project development environment (to quickly develop, deploy and test applications, or any other piece of software, as part of images running hardware), without having to understand all the Yocto project mechanisms that might seem somewhat complex.

3.1 SDK development cycle model

A developer can install the SDK on a machine (host PC), and use it to develop within any piece of software (for example, an application, kernel drivers or kernel modules).

Basically, the developer has to:

- get the software images (binaries) of the software release associated with the SDK (see Starter Package)
- install the SDK for the targeted hardware (see SDK installation)
- run the SDK environment setup script (see SDK startup)
- develop and test the piece of software

When the development is finished (the source code is ready to be shared with other developers), it should be integrated into the whole software. For this, the Distribution Package must be used.

Through the Distribution Package, new images (binaries) and a new SDK are generated (see How to create an SDK for OpenSTLinux distribution).

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

4 SDK content

The OpenSTLinux SDK is based on the **standard** Yocto project SDK.

A standard SDK consists of the following:

- a cross-development toolchain: this toolchain contains a compiler, linker, debugger, and various miscellaneous tools
- libraries, headers, and symbols (target and native sysroots): the libraries, headers, and symbols are specific to the image (that is, they match the image)
- an environment setup script: this *.sh file, once run, sets up the cross-development environment by defining variables and preparing it for SDK use

4.1 Cross-development toolchain

The cross-development toolchain consists of a cross-compiler, a cross-linker and a cross-debugger that are used to:

- develop user-space applications for targeted hardware
- modify a software component that already exists in the images, and that is delivered as source code in the Developer Package (for example the Linux kernel or U-Boot)

This cross-development toolchain is created by running a toolchain installer script (see SDK installation).

It works with a matching target sysroot (see below).

4.2 Native and target sysroots

The native and target sysroots contain the required headers and libraries for generating binaries that run on the target architecture.

The target sysroot is based on the target root file system image that is built through the Distribution Package of the STM32MPU Embedded Software and uses the same metadata configuration as that used to build the cross-toolchain.

For any software baseline, this process guarantees the alignment between:

- the content (source code) of the Distribution Package
- the target root file system image (binary) of the Starter Package
- the target sysroot (headers and libraries) of the Developer Package
- the configuration of the cross-toolchain of the Developer Package

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

5 SDK installation

The OpenSTLinux SDK is installed on the host development machine by running the *.sh installation script.

The tarball file (*SDK-[...].tar.xz*) that contains this script is named as follows: **SDK-<host machine>-<version>.tar.xz** where:

<host machine>	Host machine on which the SDK is installed: • x86_64 (only 64-bit host machines are supported)
<version>	Software release version; example: • openstlinux-5.4-dunfell-mp1-20-11-12

Example:

- en.SDK-x86_64-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz

The steps for the OpenSTLinux SDK installation, are:

- Download, on the host machine, the SDK tarball file (*SDK-[...].tar.xz*)
- Decompress the tarball file

```
$ tar xvf en.SDK-[...].tar.xz
```

- The **installation script** is named:

  <image>-<distro>-<machine>-<host machine>-toolchain-<Yocto release>-<version>.sh

  where:

<image>	Image name; example: • st-image-weston
<distro>	Distribution name; example: • openstlinux-weston
<machine>	Machine name; example: • stm32mp1
<host machine>	Host machine on which the SDK is installed: • x86_64 (only 64-bit host machines are supported)
<Yocto release>	Release number of the Yocto Project; example: • 3.1 (aka dunfell)
<version>	Software release version; example: • openstlinux-5.4-dunfell-mp1-20-11-12

  Example:

  - st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-3.1-openstlinux-5.4-dunfell-mp1-20-11-12.sh

- If necessary, change the permissions on the installation script so that it is executable:

```
$ chmod +x <image>-<distro>-<machine>-<host machine>-toolchain-<Yocto release>-
<version>.sh
```

- The SDK is self-contained and by default is installed into */opt/st/<machine>/<Yocto release>-<version>*

  Example:

  - /opt/st/stm32mp1/3.1-openstlinux-5.4-dunfell-mp1-20-11-12

  However, running the installation script with the *-d* option allows an installation directory to be chosen

  Check that the write permissions in the installation directory (either the default one, or the customized one) are granted

Information

Recommendation: for an STM32MPU Embedded Software release, install the software image from the Starter Package, the SDK and the source codes from the Developer Package in the same top directory; indeed, these packages are linked

- Run the installation script

```
$ ./<image>-<distro>-<machine>-<host machine>-toolchain-<Yocto release>-<version>.sh
```

Example (with an installation directory */local/SDK/<Yocto release>-<version>* different from the default one)

```
$ ./st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-2.6-openstlinux-20-02-19.
sh -d /local/SDK/2.6-openstlinux-20-02-19
ST OpenSTLinux - Weston - (A Yocto Project Based Distro) SDK installer version 2.6-
openstlinux-20-02-19
==========================================================================================
============================
You are about to install the SDK to "/local/SDK/2.6-openstlinux-20-02-19". Proceed[Y/n]? Y
Extracting SDK.....................................................................
.........................done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the
environment setup script e.g.
 $ . /local/SDK/2.6-openstlinux-20-02-19/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-
linux-gnueabi
```

The OpenSTLinux SDK install is now complete.

Refer to Standard SDK directory structure for details of the resulting directory structure of the installed SDK.

6 SDK startup

To use an installed SDK, its environment setup script must be run.

This setup script is located in the SDK installation directory (per default, */opt/st/<machine>/<Yocto release>-<version>*).

It must be run once in each new working terminal.

This **environment setup script** is named:

environment-setup-<target>-<distro>-linux-gnueabi

Where:

<target>	Target architecture for cross-toolchain; example: • cortexa7t2hf-neon-vfpv4
<distro>	Distribution name; example: • openstlinux_weston

Example

- environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi

- Run the environment setup script

```
 $ source <SDK installation directory path>/environment-setup-<target>-<distro>-linux-gnueabi
```

Example: here, the SDK installation directory (*/local/SDK/<Yocto release>-<version>*) is different from the default one

```
 $ source /local/SDK/3.1-openstlinux-20-11-12/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
```

- Many environment variables are then defined:

```
SDKTARGETSYSROOT - the path to the sysroot used for cross-compilation
PKG_CONFIG_PATH - the path to the target pkg-config files
CONFIG_SITE - a GNU autoconf site file preconfigured for the target
CC - the minimal command and arguments to run the C compiler
CXX - the minimal command and arguments to run the C++ compiler
CPP - the minimal command and arguments to run the C preprocessor
AS - the minimal command and arguments to run the assembler
LD - the minimal command and arguments to run the linker
GDB - the minimal command and arguments to run the GNU Debugger
STRIP - the minimal command and arguments to run 'strip', which strips symbols
RANLIB - the minimal command and arguments to run 'ranlib'
OBJCOPY - the minimal command and arguments to run 'objcopy'
OBJDUMP - the minimal command and arguments to run 'objdump'
AR - the minimal command and arguments to run 'ar'
NM - the minimal command and arguments to run 'nm'
TARGET_PREFIX - the toolchain binary prefix for the target tools
```

```
      CROSS_COMPILE - the toolchain binary prefix for the target tools
      CONFIGURE_FLAGS - the minimal arguments for GNU configure
      CFLAGS - suggested C flags
      CXXFLAGS - suggested C++ flags
      LDFLAGS - suggested linker flags when you use CC to link
      CPPFLAGS - suggested preprocessor flags
```

The OpenSTLinux SDK is started.

7 References

- Yocto Project Application Development and Extensible Software Development Kit (eSDK)

Software development kit (A programming package that enables a programmer to develop applications for a specific platform.)

Linux® is a registered trademark of Linus Torvalds.

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

also known as

GNU debugger, a portable debugger that runs on many Unix-like systems

Stable: 25.09.2020 - 09:10 / Revision: 25.09.2020 - 09:08

A quality version of this page, approved on *25 September 2020*, was based off this revision.

This article describes how to get and use the **Developer Package** of the **STM32MPU Embedded Software** for any development platform of the **STM32MP1 family** (STM32MP15 boards), in order to modify some of its pieces of software, or to add applications on top of it.

It lists some **prerequisites** in terms of knowledges and development environment, and gives the **step-by-step** approach to download and install the STM32MPU Embedded Software components for this Package.

Finally, it proposes some guidelines to upgrade (add, remove, configure, improve...) any piece of software.

Contents

1 Developer Package content

If you are not yet familiar with the **STM32MPU Embedded Software** distribution and its **Packages**, please read the following articles:

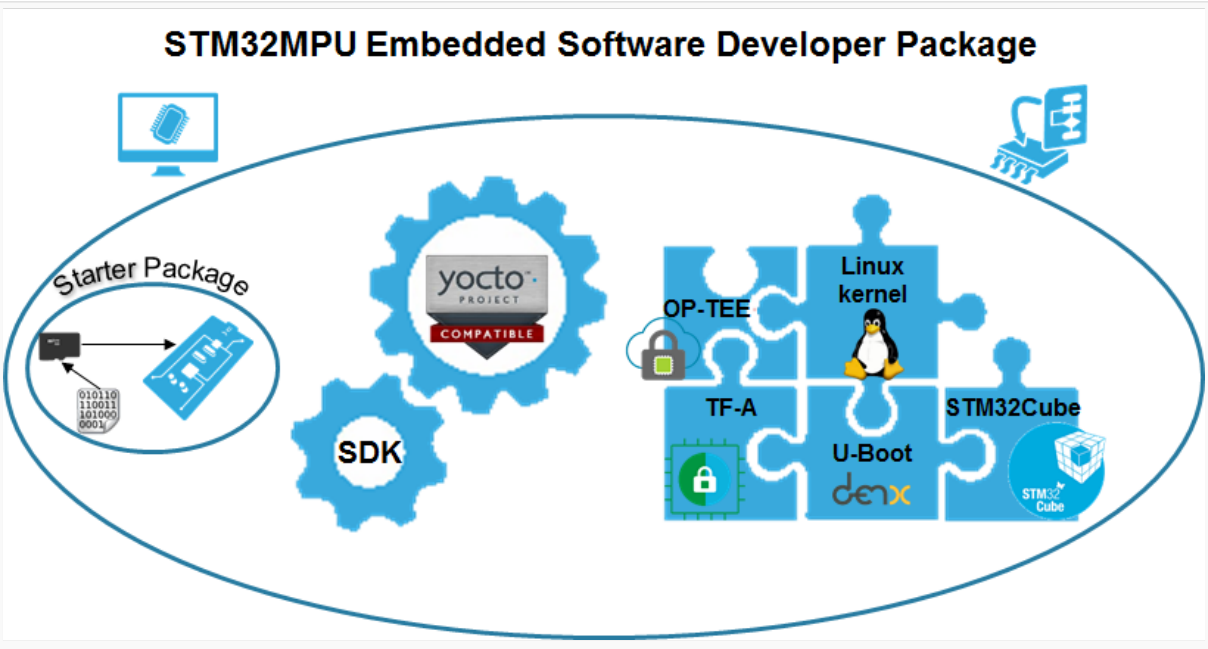
- Which STM32MPU Embedded Software Package better suits your needs (and especially the Developer Package chapter)
- STM32MPU Embedded Software distribution

If you are already familiar with the Developer Package for the STM32MPU Embedded Software distribution, the fast links to essential commands might interest you.

To sum up, this **Developer Package** provides:

- for the **OpenSTLinux distribution** (development on Arm®Cortex®-A processor):
  - the **software development kit** (SDK), based on Yocto SDK, for cross-development on an host PC
  - the following pieces of software in **source code**:
    - Linux® kernel
    - U-Boot
    - Trusted Firmware-A (TF-A)
    - optionally, Open source Trusted Execution Environment (OP-TEE)
  - the **debug symbol files** for Linux® kernel, U-Boot and TF-A
- for the **STM32Cube MPU Package** (development on Arm®Cortex®-M processor):
  - all pieces of software (BSP, HAL, middlewares and applications) in **source code**
  - the **integrated development environment (IDE)** (STM32CubeIDE)

Note that, the application frameworks for the OpenSTLinux distribution are not available as source code in this Package.



Components of the Developer Package

2 Developer Package step-by-step overview

The steps to get the STM32MPU Embedded Software Developer Package ready for your developments, are:

Checking the prerequisites

Installing the Starter Package for your board

Installing the components to develop software running on Arm®Cortex®-A (OpenSTLinux distribution)

Installing the SDK (**mandatory** for any development on Arm®Cortex®-A)

Installing the Linux kernel (**mandatory only** if you plan to modify the Linux kernel or to add external out-of-tree Linux kernel modules)

Installing the U-Boot (**mandatory only** if you plan to modify the U-Boot)

Installing the TF-A (**mandatory only** if you plan to modify the TF-A)

Installing the TF-A-SSP (**mandatory only** if you plan to modify the TF-A SSP)

Installing the debug symbol files (**mandatory only** if you plan to debug Linux® kernel, U-Boot or TF-A with GDB)

Installing the components to develop software running Arm Cortex-M (STM32Cube MPU Package)

Installing STM32CubeIDE (**mandatory** for any development on Arm®Cortex®-M)

Installing the STM32Cube MPU Package (**mandatory only** if you plan to modify the Cube firmware)

Once these steps are achieved, you are able to:

- develop software running on Arm Cortex-A
  - Modifying the Linux kernel
  - Adding external out-of-tree Linux kernel modules
  - Adding Linux user space applications
  - Modifying the U-Boot
  - Modifying the TF-A
- develop software running on Arm Cortex-M4

3 Checking the prerequisites

3.1 Knowledges

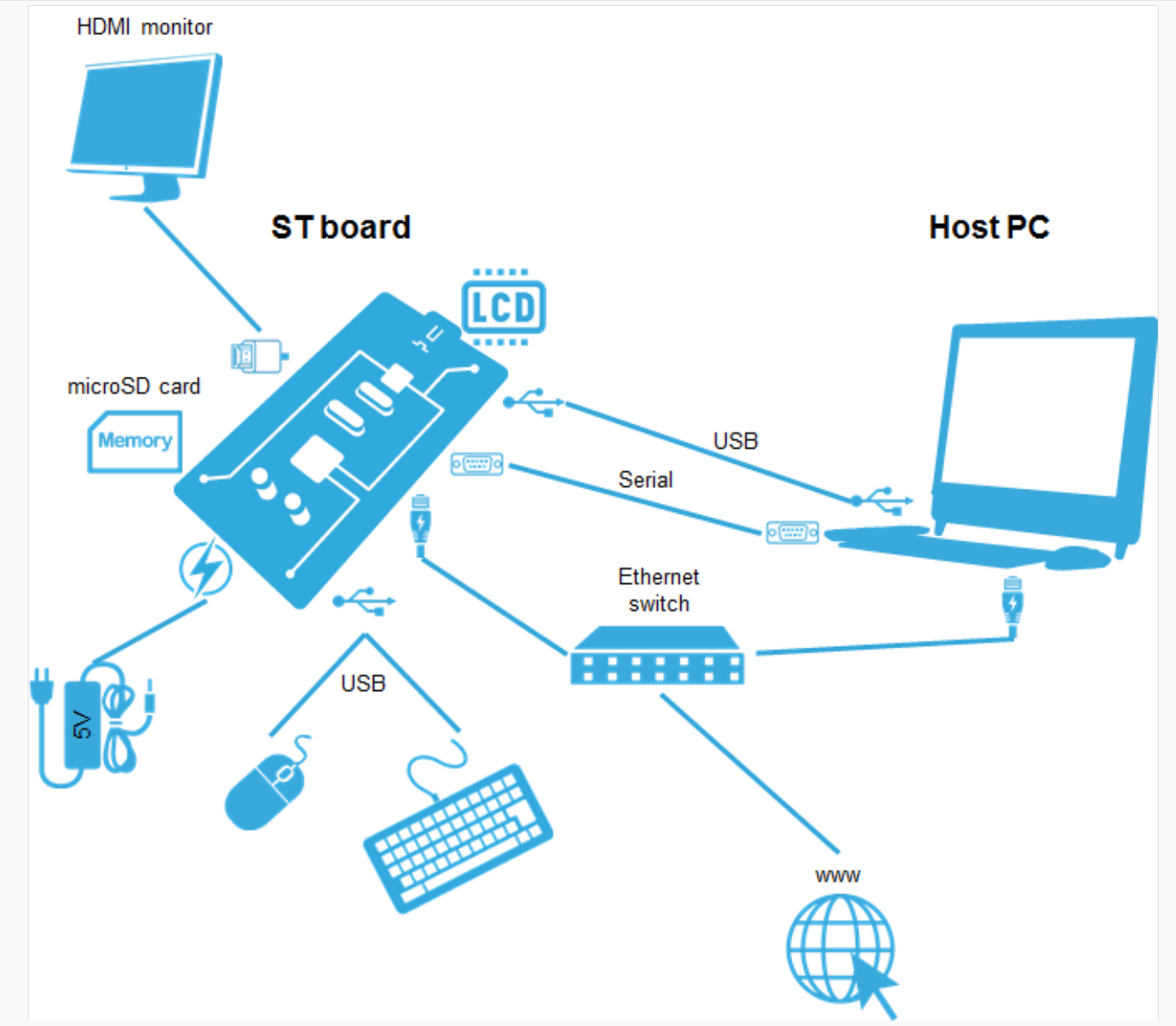
The STM32MP1 Developer Package aims at enriching a Linux-based software for the targeted product: basic knowledges about Linux are recommended to make the most of this Package.

Having a look at the STM32MPU Embedded Software architecture overview is also highly recommended.

3.2 Development setup

The recommended setup for the development PC (host) is specified in the following article: PC prerequisites.

Whatever the development platform (board) and development PC (host) used, the range of possible development setups is illustrated by the picture below.



Development setup for Developer and Distribution Packages

The following components are **mandatory**:

- Host PC for cross-compilation and cross-debugging, installed as specified above
- Board assembled and configured as specified in the associated Starter Package article
- Mass storage device (for example, microSD card) to load and update the software images (binaries)

The following components are **optional**, but **recommended**:

- A serial link between the host PC (through Terminal program) and the board for traces (even early boot traces), and access to the board from the remote PC (command lines)
- An Ethernet link between the host PC and the board for cross-development and cross-debugging through a local network. This is an alternative or a complement to the serial (or USB) link
- A display connected to the board, depending on the technologies available on the board: DSI LCD display, HDMI monitor (or TV) and so on
- A mouse and a keyboard connected through USB ports

Additional optional components can be added by means of the connectivity capabilities of the board: cameras, displays, JTAG, sensors, actuators, and much more.

4 Installing the Starter Package

Before developing with the Developer Package, **it is essential to start up your board thanks to its Starter Package**. All articles relative to Starter Packages are found in Category:Starter Package: find the one that corresponds to your board, and follow the installation instructions (if not yet done), before going further.

In brief, it means that:

- your board boots successfully
- the flashed image comes from the same release of the STM32MPU Embedded Software distribution than the components that will be downloaded in this article

Thanks to the Starter Package, **all Flash partitions are populated**.

Then, with the Developer Package, it is possible to modify or to upgrade the partitions independently one from the others.

For example, if you only want to modify the Linux kernel (part of *bootfs* partition), installing the SDK and the Linux kernel are enough; no need to install anything else.

5 Installing the components to develop software running on Arm Cortex-A (OpenSTLinux distribution)

5.1 Installing the SDK

Optional step: **it is mandatory only if you want to modify or add software running on Arm Cortex-A (e.g. Linux kernel, Linux user space applications...).**

The SDK for OpenSTLinux distribution provides a stand-alone cross-development toolchain and libraries tailored to the contents of the specific image flashed in the board. If you want to know more about this SDK, please read the SDK for OpenSTLinux distribution article.

- The STM32MP1 SDK is delivered through a tarball file named : **en.SDK-x86_64-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz**
- Download and install the STM32MP1 SDK.

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

Warning

To download a package, it is recommended to be logged in to your "myst" account [1]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package SDK - STM32MP15-Ecosystem-v2.1.0 release
Download	You need to be logged on *my.st.com* before accessing the following link: en.SDK-x86_64-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz

	• Uncompress the tarball file to get the SDK installation script
	`tar xvf en.SDK-x86_64-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz`
	• If needed, change the permissions on the SDK installation script so that it is executable
	`$ chmod +x stm32mp1-openstlinux-5.4-dunfell-mp1-20-11-12/sdk/st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-3.1-openstlinux-5.4-dunfell-mp1-20-11-12.sh`
	• Run the SDK installation script
	  • Use the *-d <SDK installation directory absolute path>* option to specify the absolute path to the directory in which you want to install the SDK (*<SDK installation directory>*)
	  • If you follow the proposition to organize the working directory, it means:

	STM32MP1 Developer Package SDK - STM32MP15-Ecosystem-v2.1.0 release

Installation	`$ ./stm32mp1-openstlinux-5.4-dunfell-mp1-20-11-12/sdk/st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-3.1-openstlinux-5.4-dunfell-mp1-20-11-12.sh -d <working directory absolute path>/Developer-Package/SDK`
	• A successful installation outputs the following log:
	`ST OpenSTLinux - Weston - (A Yocto Project Based Distro) SDK installer version 3.1-openstlinux-5-4-dunfell-mp1-20-11-12`
	`==========`
	`You are about to install the SDK to "<working directory absolute path>/Developer-Package/SDK". Proceed [Y/n]?`
	`Extracting SDK.....`
	`.....`
	`.....`
	`.....done`
	`Setting it up...done`
	`SDK has been successfully set up and is ready to be used.`
	`Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.`
	`$ . <working directory absolute path>/Developer-Package/SDK/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi`

Release note	Details about the content of the SDK are available in the **associated** STM32MP15 ecosystem release note. If you are interested in older releases, please have a look into the section Archives.

- The SDK is in the *<SDK installation directory>*:

```
<SDK installation directory>                                          SDK for
OpenSTLinux distribution: details in Standard SDK directory structure article
├── environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi  Environment setup script
 for Developer Package
├── site-config-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
├── sysroots
│   ├── cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi                Target sysroot
(libraries, headers, and symbols)
│   │   └── [...]
│   └── x86_64-ostl_sdk-linux                                     Native sysroot
(libraries, headers, and symbols)
│       └── [...]
└── version-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
```

Warning

Now that the SDK is installed, please do not move or rename the *<SDK installation directory>*.

5.1.1 Starting up the SDK

The SDK environment setup script must be run once in each new working terminal in which you cross-compile:

```
PC $> source <SDK installation directory>/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-
linux-gnueabi
```

The following checkings allow to ensure that the environment is correctly setup:

- Check the target architecture

```
PC $> echo $ARCH
arm
```

- Check the toolchain binary prefix for the target tools

```
PC $> echo $CROSS_COMPILE
arm-ostl-linux-gnueabi-
```

- Check the C compiler version

```
PC $> $CC --version
arm-ostl-linux-gnueabi-gcc (GCC) <GCC version>
[...]
```

- Check that the SDK version is the expected one

```
PC $> echo $OECORE_SDK_VERSION
<expected SDK version>
```

If any of these commands fails or does not return the expected result, please try to reinstall the SDK.

5.2 Installing the Linux kernel

Optional step: it is mandatory only if you want to modify the Linux kernel (configuration, device tree, driver...), or to add external out-of-tree Linux kernel modules.

Prerequite: the SDK is installed.

5.2.1 Downloading the Linux kernel

- The STM32MP1 Linux kernel is delivered through a tarball file named **en.SOURCES-kernel-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz** for STM32MP157x-EV1 and STM32MP157x-DKx boards.

- Download and install the STM32MP1 Linux kernel

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

Warning

To download a package, it is recommended to be logged in to your "myst" account [2]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package Linux kernel - STM32MP15-Ecosystem-v2.1.0 release
Download	You need to be logged on to *my.st.com* before accessing the following link en.SOURCES-kernel-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz

Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, this means:
	`$ cd <working directory path>/Developer-Package`
	• Download the tarball file in this directory
	• Uncompress the tarball file to get the Linux kernel (Linux kernel source code, ST patches, ST configuration fragments...):
	`PC $> $ tar xvf en.SOURCES-kernel-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz`
	`PC $> $ cd stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12/sources/arm-ostl-linux-gnueabi/linux-stm32mp-5.4.56-r0`
	`PC $> $ tar xvf linux-5.4.56.tar.xz`

Release note	Details of the content of the Linux kernel are available in the **associated** STM32MP15 OpenSTLinux release note. If you are interested in older releases, please have a look into the section Archives.

- The **Linux kernel installation directory** is in the *<Developer Package installation directory>/stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12/sources/arm-ostl-linux-gnueabi* directory, and is named *linux-stm32mp-<kernel version>*:

```
linux-stm32mp-5.4.56-r0      Linux kernel installation directory
├── [*].patch               ST patches to apply during the Linux kernel preparation (see
next chapter)
├── fragment-[*].config     ST configuration fragments to apply during the Linux kernel
configuration (see next chapter)
├── linux-5.4.56            Linux kernel source code directory
├── linux-5.4.56.tar.xz     Tarball file of the Linux kernel source code
├── README.HOW_TO.txt       Helper file for Linux kernel management: reference for Linux
kernel build
└── series                  List of all ST patches to apply
```

5.2.2 Building and deploying the Linux kernel for the first time

It is mandatory to execute once the steps specified below before modifying the Linux kernel, or adding external out-of-tree Linux kernel modules.

The partitions related to the Linux kernel are:

- the *bootfs* partition that contains the Linux kernel U-Boot image (*uImage*) and device tree
- the *rootfs* partition that contains the Linux kernel modules

The Linux kernel might be cross-compiled, either in the source code directory, or in a dedicated directory different from the source code directory.
This last method is recommended as it clearly separates the files generated by the cross-compilation from the source code files.

Information

The README_HOWTO.txt helper file is **THE** reference for the Linux kernel build

Warning

The SDK must be started

Open the *<Linux kernel installation directory>/README.HOW_TO.txt* helper file, and execute its instructions to:

setup a software configuration management (SCM) system (*git*) for the Linux kernel (optional but recommended)

prepare the Linux kernel (applying the ST patches)

configure the Linux kernel (applying the ST fragments)

cross-compile the Linux kernel

deploy the Linux kernel (i.e. update the software on board)

The Linux kernel is now installed: let's modify the Linux kernel, or add external out-of-tree Linux kernel modules.

5.3 Installing the U-Boot

Optional step: it is mandatory only if you want to modify the U-Boot.

Prerequite: the SDK is installed.

5.3.1 Downloading the U-Boot

- The STM32MP1 U-Boot is delivered through a tarball file named **en.SOURCES-u-boot-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz** for STM32MP157x-EV1 and STM32MP157x-DKx boards.
- Download and install the STM32MP1 U-Boot

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

Warning

To download a package, it is recommended to be logged in to your "myst" account [3]. If, trying to

download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package U-Boot - STM32MP15-Ecosystem-v2.1.0 release
Download	You need to be logged on to *my.st.com* before accessing the following link en.SOURCES-u-boot-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz

Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, this means:
	`$ cd <working directory path>/Developer-Package`
	• Download the tarball file in this directory
	• Uncompress the tarball file to get the U-Boot (U-Boot source code, ST patches and so on):
	`PC $> $ tar xvf en.SOURCES-u-boot-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz`
	`PC $> $ cd stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12/sources/arm-ostl-linux-gnueabi/u-boot-stm32mp-2020.01.r2-r0`
	`PC $> $ tar xvf u-boot-stm32mp-2020.01.r2-r0.tar.gz`

Release note	Details of the content of the U-Boot are available in the **associated** STM32MP15 OpenSTLinux release note. If you are interested in older releases, please have a look into the section Archives.

- The **U-Boot installation directory** is in the *<Developer Package installation directory>/stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12/sources/arm-ostl-linux-gnueabi* directory, and is named *u-boot-stm32mp-<U-Boot version>*:

```
u-boot-stm32mp-2020.01.r2-r0
├── [*].patch                          U-Boot installation directory
├── u-boot-stm32mp-2020.01.r2          ST patches to apply during
├── Makefile.sdk                       the U-Boot preparation (see
├── README.HOW_TO.txt                  next chapter)
├── series                             U-Boot source code directory
└── u-boot-stm32mp-2020.01.r2-r0.tar.  Makefile for the U-Boot
gz                                     compilation
                                       Helper file for U-Boot
                                       management: reference for U-
                                       Boot build
                                       List of all ST patches to
                                       apply
                                       Tarball file of the U-Boot
                                       source code
```

5.3.2 Building and deploying the U-Boot for the first time

It is mandatory to execute once the steps specified below before modifying the U-Boot.

As explained in the boot chain overview, the trusted boot chain is the default solution delivered by STMicroelectronics.

Within this scope, the partition related to the U-Boot is the *ssbl* one that contains the U-Boot and its device tree blob.

Information

The README_HOWTO.txt helper file is **THE** reference for the U-Boot build

Warning

The SDK must be started

Open the *<U-Boot installation directory>/README.HOW_TO.txt* helper file, and execute its instructions to:

setup a software configuration management (SCM) system (*git*) for the U-Boot (optional but recommended)

prepare the U-Boot (applying the ST patches)

cross-compile the U-Boot

deploy the U-Boot (i.e. update the software on board)

The U-Boot is now installed: let's modify the U-Boot.

5.4 Installing the TF-A

Optional step: it is mandatory only if you want to modify the TF-A.

Prerequite: the SDK is installed.

5.4.1 Downloading the TF-A

- The STM32MP1 TF-A is delivered through a tarball file named **en.SOURCES-tf-a-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz** for STM32MP157x-EV1 and STM32MP157x-DKx boards.
- Download and install the STM32MP1 TF-A

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

Warning

To download a package, it is recommended to be logged in to your "myst" account [4]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package TF-A - STM32MP15-Ecosystem-v2.1.0 release
Download	You need to be logged on *my.st.com* before accessing the following link: en.SOURCES-tf-a-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz

	STM32MP1 Developer Package TF-A - STM32MP15-Ecosystem-v2.1.0 release

Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, it means:
	`$ cd <working directory path>/Developer-Package`
	• Download the tarball file in this directory
	• Uncompress the tarball file to get the TF-A (TF-A source code, ST patches...):
	`PC $> $ tar xvf en.SOURCES-tf-a-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz`
	`PC $> $ cd stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12/sources/arm-ostl-linux-gnueabi/tf-a-stm32mp-2.2.r2-r0`
	`PC $> $ tar xvf tf-a-stm32mp-2.2.r2-r0.tar.gz`

Release note	Details about the content of the TF-A are available in the **associated** STM32MP15 OpenSTLinux release note. If you are interested in older releases, please have a look into the section Archives.

• The **TF-A installation directory** is in the *<Developer Package installation directory>/stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12/sources/arm-ostl-linux-gnueabi* directory, and is named *tf-a-stm32mp-<TF-A version>*:

```
tf-a-stm32mp-2.2.r2-r0                TF-A installation directory
├── [*].patch                         ST patches to apply during the TF-A preparation (see
next chapter)
├── tf-a-stm32mp-2.2.r2               TF-A source code directory
├── Makefile.sdk                      Makefile for the TF-A compilation
├── README.HOW_TO.txt                 Helper file for TF-A management: reference for TF-A
build
├── series                            List of all ST patches to apply
└── tf-a-stm32mp-2.2.r2-r0.tar.gz     Tarball file of the TF-A source code
```

5.4.2 Building and deploying the TF-A for the first time

It is mandatory to execute once the steps specified below before modifying the TF-A.

As explained in the boot chain overview, the trusted boot chain is the default solution delivered by STMicroelectronics.

Within this scope, the partition related to the TF-A is the *fsbl* one.

Information

The README_HOWTO.txt helper file is **THE** reference for the TF-A build

Warning

The SDK must be started

Open the *<TF-A installation directory>/README.HOW_TO.txt* helper file, and execute its instructions to:

setup a software configuration management (SCM) system (*git*) for the TF-A (optional but recommended)

prepare the TF-A (applying the ST patches)

cross-compile the TF-A

deploy the TF-A (i.e. update the software on board)

The TF-A is now installed: let's modify the TF-A.

5.5 Installing the TF-A-SSP

Optional step: it is mandatory only if you want to modify the TF-A to use Secure Secret provisioning feature.

Prerequite: the SDK is installed.

5.5.1 Downloading the TF-A-SSP

- The STM32MP1 TF-A-SSP is delivered through a tarball file named **en.SOURCES-tf-a-ssp-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz**
- Download and install the STM32MP1 TF-A-SSP

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

Warning

To download a package, it is recommended to be logged in to your "myst" account [5]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package TF-A SSP - STM32MP15-Ecosystem-v2.1.0 release
Download	You need to be logged on *my.st.com* before accessing the following link: en.SOURCES-tf-a-ssp-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz
Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, it means: `$ cd <working directory path>/Developer-Package` • Download the tarball file in this directory • Uncompress the tarball file to get the TF-A SSP (TF-A SSP source code, ST patches...):

	STM32MP1 Developer Package TF-A SSP - STM32MP15-Ecosystem-v2.1.0 release
	PC $> $ tar xvf en.SOURCES-tf-a-ssp-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz **PC $>** $ cd stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12/sources/arm-ostl-linux-gnueabi/tf-a-stm32mp-ssp-2.2.r2-r0 **PC $>** $ tar xvf tf-a-stm32mp-ssp-2.2.r2-r0.tar.gz
Release note	Details about the content of the TF-A SSP are available in the **associated** STM32MP15 OpenSTLinux release note. If you are interested in older releases, please have a look into the section Archives.

- The **TF-A SSP installation directory** is in the *<Developer Package installation directory>/stm32mp1-openstlinux-20-11-12/sources/arm-ostl-linux-gnueabi* directory, and is named *tf-a-stm32mp-ssp-<TF-A version>*:

```
tf-a-stm32mp-ssp-2.2.r2-r0
├── [*].patch
├── tf-a-stm32mp-ssp-2.2.r2
├── Makefile.sdk
├── README.HOW_TO.txt
├── series
└── tf-a-stm32mp-ssp-2.2.r2-r0.tar.gz
```

```
TF-A SSP installation directory
ST patches to apply during the TF-A SSP preparation (see next chapter)
TF-A SSP source code directory
Makefile for the TF-A SSP compilation
Helper file for TF-A SSP management: reference for TF-A SSP build
List of all ST patches to apply
Tarball file of the TF-A SSP source code
```

5.5.2 Building the TF-A-SSP for the first time

It is mandatory to execute once the steps specified below before modifying the TF-A SSP.

Information

The README_HOWTO.txt helper file is **THE** reference for the TF-A SSP build

Warning

The SDK must be started

Open the *<TF-A SSP installation directory>/README.HOW_TO.txt* helper file, and execute its instructions to:

setup a software configuration management (SCM) system (*git*) for the TF-A SSP (optional but recommended)

prepare the TF-A SSP (applying the ST patches)

cross-compile the TF-A SSP

The TF-A SSP is now built.

5.6 Installing the OP-TEE

Optional step: **it is mandatory only if you want to modify the OP-TEE.**

Prerequite: the SDK is installed.

5.6.1 Downloading the OP-TEE

- The STM32MP1 OP-TEE is delivered through a tarball file named **en.SOURCES-optee-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz** for STM32MP157x-EV1 and STM32MP157x-DKx boards.
- Download and install the STM32MP1 OP-TEE

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

Warning

To download a package, it is recommended to be logged in to your "myst" account [6]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package OP-TEE - STM32MP15-Ecosystem-v2.1.0 release
Download	You need to be logged on *my.st.com* before accessing the following link: en.SOURCES-optee-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz
Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, it means: `$ cd <working directory path>/Developer-Package` • Download the tarball file in this directory • Uncompress the tarball file to get the OP-TEE (OP-TEE source code, ST patches...):

	STM32MP1 Developer Package OP-TEE - STM32MP15-Ecosystem-v2.1.0 release
	PC $> $ tar xvf en.SOURCES-optee-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz **PC $>** $ cd stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12/sources/arm-ostl-linux-gnueabi/optee-os-stm32mp-3.9.0.r2-r0 **PC $>** $ tar xvf optee-os-stm32mp-3.9.0.r2-r0.tar.gz
Release note	Details about the content of the OP-TEE are available in the **associated** STM32MP15 OpenSTLinux release note. If you are interested in older releases, please have a look into the section Archives.

- The **OP-TEE installation directory** is in the *<Developer Package installation directory>/stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12/sources/arm-ostl-linux-gnueabi* directory, and is named *optee-os-stm32mp-<OP-TEE version>*:

```
optee-os-stm32mp-3.9.0.r2-r0
├── [*].patch
├── optee-os-stm32mp-3.9.0.r2
├── Makefile.sdk
├── optee-os-stm32mp-3.9.0.r2-r0.tar.gz
├── README.HOW_TO.txt
└── series
```

```
OP-TEE installation directory
ST patches to apply during the OP-TEE preparation (see next chapter)
OP-TEE source code directory
Makefile for the OP-TEE compilation
Tarball file of the OP-TEE source code
Helper file for OP-TEE management: reference for OP-TEE build
List of all ST patches to apply
```

5.6.2 Building and deploying the OP-TEE for the first time

It is mandatory to execute once the steps specified below before modifying the OP-TEE.

As explained in the boot chain overview, the trusted boot chain is the default solution delivered by STMicroelectronics.

Within this scope, the partition related to the OP-TEE is the *fsbl* one.

Information

The README_HOWTO.txt helper file is **THE** reference for the OP-TEE build

Warning

The SDK must be started

Open the *<OP-TEE installation directory>/README.HOW_TO.txt* helper file, and execute its instructions to:

setup a software configuration management (SCM) system (*git*) for the OP-TEE (optional but recommended)

prepare the OP-TEE (applying the ST patches)

cross-compile the OP-TEE

deploy the OP-TEE (i.e. update the software on board)

The OP-TEE is now installed: let's modify the OP-TEE.

5.7 Installing the debug symbol files

Optional step: **it is mandatory only if you want to debug Linux® kernel, U-Boot or TF-A with GDB.**

5.7.1 Downloading the debug symbol files

- The STM32MP1 debug symbol files is delivered through a tarball file named **en.DEBUG-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz** for STM32MP157x-EV1 and STM32MP157x-DKx boards.
- Download and install the STM32MP1 debug symbol files

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

Warning

To download a package, it is recommended to be logged in to your "myst" account [7]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package debug symbol files - STM32MP15-Ecosystem-v2.1.0 release
Download	You need to be logged on to *my.st.com* before accessing the following link en.DEBUG-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz

Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, this means:
	`$ cd <working directory path>/Developer-Package`
	• Download the tarball file in this directory
	• Uncompress the tarball file to get the debug symbol files (for Linux kernel, U-Boot, TF-A and OP-TEE OS):
	`PC $> $ tar xvf en.DEBUG-stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12.tar.xz`

ST Restricted – Subject to Non-Disclosure Agreement – Do not copy

	release
Release note	If you are interested in older releases, please have a look into the section Archives.

- The debug symbol files are in the *<Developer Package installation directory>/stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12/images/stm32mp1* directory:

```
stm32mp1
├── arm-trusted-firmware
│   ├── tf-a-bl2-optee.elf
│   ├── tf-a-bl2-serialboot.elf
│   ├── tf-a-bl2-trusted.elf
│   ├── tf-a-bl32-serialboot.elf
│   └── tf-a-bl32-trusted.elf
├── bootloader
│   ├── u-boot-stm32mp157a-dk1-optee.elf
│   ├── u-boot-stm32mp157a-dk1-trusted.
elf
│   ├── u-boot-stm32mp157a-ev1-optee.elf
│   ├── u-boot-stm32mp157a-ev1-trusted.
elf
│   ├── u-boot-stm32mp157c-dk2-optee.elf
│   ├── u-boot-stm32mp157c-dk2-trusted.
elf
│   ├── u-boot-stm32mp157c-ev1-optee.elf
│   ├── u-boot-stm32mp157c-ev1-trusted.
elf
│   ├── u-boot-stm32mp157d-dk1-optee.elf
│   ├── u-boot-stm32mp157d-dk1-trusted.
elf
│   ├── u-boot-stm32mp157d-ev1-optee.elf
```

```
Debug symbol file for TF-A
→ TF-A for OP-TEE boot stage
Debug symbol file for TF-A
→ fsbl for flasher
Debug symbol file for TF-A
→ TF-A for OP-TEE boot stage
Debug symbol file for TF-A
→ secure monitor for flasher
Debug symbol file for TF-A
→ runtime software stage


Debug symbol file for U-Boot
 → STM32MP15 Discovery kits
Debug symbol file for U-Boot
 → STM32MP15 Discovery kits
Debug symbol file for U-Boot
 → STM32MP15 Evaluation
boards
Debug symbol file for U-Boot
 → STM32MP15 Evaluation
boards
Debug symbol file for U-Boot
 → STM32MP15 Discovery kits
Debug symbol file for U-Boot
 → STM32MP15 Discovery kits
Debug symbol file for U-Boot
 → STM32MP15 Evaluation
boards
Debug symbol file for U-Boot
 → STM32MP15 Evaluation
boards
Debug symbol file for U-Boot
 → STM32MP15 Discovery kits
Debug symbol file for U-Boot
 → STM32MP15 Discovery kits
Debug symbol file for U-Boot
 → STM32MP15 Evaluation
boards
Debug symbol file for U-Boot
 → STM32MP15 Evaluation
boards
Debug symbol file for U-Boot
```

```
│   ├── u-boot-stm32mp157d-ev1-trusted.elf
│   ├── u-boot-stm32mp157f-dk2-optee.elf
│   ├── u-boot-stm32mp157f-dk2-trusted.elf
│   ├── u-boot-stm32mp157f-ev1-optee.elf
│   └── u-boot-stm32mp157f-ev1-trusted.elf
├── kernel
│   └── vmlinux
└── optee
    ├── tee-stm32mp157a-dk1-optee.elf
    ├── tee-stm32mp157a-ev1-optee.elf
    ├── tee-stm32mp157c-dk2-optee.elf
    ├── tee-stm32mp157c-ev1-optee.elf
    ├── tee-stm32mp157d-dk1-optee.elf
    ├── tee-stm32mp157d-ev1-optee.elf
    ├── tee-stm32mp157f-dk2-optee.elf
    └── tee-stm32mp157f-ev1-optee.elf
```

```
→ STM32MP15 Discovery kits
Debug symbol file for U-Boot
→ STM32MP15 Discovery kits
Debug symbol file for U-Boot
→ STM32MP15 Evaluation
boards
Debug symbol file for U-Boot
→ STM32MP15 Evaluation
boards


Debug symbol file for Linux
kernel


Debug symbol file for OP-TEE
OS → STM32MP15 Discovery
kits
Debug symbol file for OP-TEE
OS → STM32MP15 Evaluation
boards
Debug symbol file for OP-TEE
OS → STM32MP15 Discovery
kits
Debug symbol file for OP-TEE
OS → STM32MP15 Evaluation
boards
Debug symbol file for OP-TEE
OS → STM32MP15 Discovery
kits
Debug symbol file for OP-TEE
OS → STM32MP15 Evaluation
boards
Debug symbol file for OP-TEE
OS → STM32MP15 Discovery
kits
Debug symbol file for OP-TEE
OS → STM32MP15 Evaluation
boards
```

5.7.2 Using the debug symbol files

These files are used to debug the Linux® kernel, U-Boot or TF-A with GDB. Especially, the Debug OpenSTLinux BSP components chapter explains how to load the debug symbol files in GDB.

6 Installing the components to develop software running on Arm Cortex-M4 (STM32Cube MPU Package)

6.1 Installing STM32CubeIDE

Optional step: it is needed if you want to modify or add software running on Arm Cortex-M.

The table below explains how to download and install STM32CubeIDE which addresses STM32 MCU, and also provides support for Cortex-M inside STM32 MPU.

It is available on Linux® and Windows® host PCs, but **it is NOT on macOS®.**

	STM32CubeIDE for Linux host PC	STM32CubeIDE for Windows® host PC

Download	**Version 1.5.0**
	• Download the preferred all-in-one Linux installer from my.st.com
	  • *Generic Linux Installer - STM32CubeIDE-Lnx*
	  • *RPM Linux Installer - STM32CubeIDE-RPM*
	  • *Debian Linux Installer - STM32CubeIDE-DEB*
		Version 1.5.0
		• Download the all-in-one Windows installer from my.st.com
		  • *Windows Installer - STM32CubeIDE-Win*

Installation guide	• Please refer to *STM32CubeIDE Installation guide (UM2563)* available on my.st.com.	

User manual	• When the installation is over, please see additional information about the STM32CubeIDE in my.st.com:
	  • *STM32CubeIDE quick start guide (UM2553)*
	  • *Getting started with projects based on the STM32MP1 Series in STM32CubeIDE (AN5360)*	

Detailed release note	• Details about the content of this tool version are available in Release Notes *STM32CubeIDE release v1.5.0* from my.st.com	

Minor releases may then be available from update site, please check chapter 10 in (UM2609) for more information on how to update STM32CubeIDE.

6.2 Installing the STM32Cube MPU Package

Optional step: it is mandatory only if you want to modify the STM32Cube MPU Package.

Prerequite: the STM32CubeIDE is installed.

- The STM32CubeMP1 Package is delivered through an archive file named **en.STM32Cube_FW_MP1_V1.3.0.zip**.

- Download and install the STM32CubeMP1 Package

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

Warning

To download a package, it is recommended to be logged in to your "myst" account [8]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

.

	STM32MP1 Developer Package STM32CubeMP1 Package - v2.1.0 release
Download	You need to be logged on *my.st.com* before accessing the following link: en.STM32Cube_FW_MP1_V1.3.0.zip

Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, it means:
	`$ cd <working directory path>/Developer-Package`
	• Download the archive file in this directory
	• Uncompress the archive file to get the STM32CubeMP1 Package:
	`$ unzip en.STM32Cube_FW_MP1_V1.3.0.zip`

Release note	Details about the content of the STM32CubeMP1 Package are available in the *STM32Cube_FW_MP1_V2.1.0/Release_Notes.html* file. If you are interested in older releases, please have a look into the section Archives.

- The **STM32CubeMP1 Package installation directory** is in the *<Developer Package installation directory>* directory, and is named *STM32Cube_FW_MP1_V1.2.0*:

```
STM32Cube_FW_MP1_V1.3.0                          STM32CubeMP1 Package: details in STM32
CubeMP1 Package content article
├── Drivers
│   ├── BSP                                      BSP drivers for the supported
STM32MP1 boards
│   │   └── [...]
│   ├── CMSIS
│   │   └── [...]
│   └── STM32MP1xx_HAL_Driver                    HAL drivers for the supported
STM32MP1 devices
│       └── [...]
├── _htmresc
│   └── [...]
├── License.md
├── Middlewares
```

```
│   └── [...]
├── package.xml
├── Projects
│   ├── STM32CubeProjectsList.html                  List of examples and applications for
STM32CubeMP1 Package
│   ├── STM32MP157C-DK2                             Set of examples and applications →
STM32MP15 Discovery kits
│   │   └── [...]
│   └── STM32MP157C-EV1                             Set of examples and applications →
STM32MP15 Evaluation boards
│       └── [...]
├── Readme.md
├── Release_Notes.html                              Release note for STM32CubeMP1 Package
└── Utilities
    └── [...]
```

The STM32Cube MPU Package is now installed: let's develop software running on Arm Cortex-M4.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

7 Developing software running on Arm Cortex-A7

7.1 Modifying the Linux kernel

Prerequisites:

- the SDK is installed
- the SDK is started up
- the Linux kernel is installed

The *<Linux kernel installation directory>/README.HOW_TO.txt* helper file gives the commands to:

configure the Linux kernel

cross-compile the Linux kernel

deploy the Linux kernel (that is, update the software on board)

You can refer to the following simple examples:

- Modification of the kernel configuration
- Modification of the device tree
- Modification of a built-in device driver
- Modification of an external in-tree module

7.2 Adding external out-of-tree Linux kernel modules

Prerequisites:

- the SDK is installed
- the SDK is started up
- the Linux kernel is installed

Most device drivers (or modules) in the Linux kernel can be compiled either into the kernel itself (built-in, or internal module) or as Loadable Kernel Modules (LKMs, or external modules) 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).

External Linux kernel modules are compiled taking reference to a Linux kernel source tree and a Linux kernel configuration file (*.config*).
Thus, a makefile for an external Linux kernel module points to the Linux kernel directory that contains the source code and the configuration file, with the **"-C <Linux kernel path>"** option.
This makefile also points to the directory that contains the source file(s) of the Linux kernel module to compile, with the **"M=< Linux kernel module path>"** option.

A generic makefile for an external out-of-tree Linux kernel module looks like the following:

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

# Object file(s) to be built
obj-m := <module source file(s)>.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 generated objects
```

```
DESTDIR ?= <Linux kernel installation directory>

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

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

install:
        $(MAKE) -C $(KERNEL_DIR) M=$(PWD) INSTALL_MOD_PATH=$(DESTDIR) modules_install

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

Such module is then cross-compiled with the following commands:

```
$ make clean
$ make
$ make install
```

You can refer to the following simple example:

- Addition of an external out-of-tree module

7.3 Adding Linux user space applications

Prerequisites:

- the SDK is installed
- the SDK is started up

Once a suitable cross-toolchain (OpenSTLinux SDK) is installed, it is easy to develop a project outside of the OpenEmbedded build system.
There are different ways to use the SDK toolchain directly, among which Makefile and Autotools.
Whatever the method, it relies on:

- the sysroot that is associated with the cross-toolchain, and that contains the header files and libraries needed for generating binaries (see target sysroot)
- the environment variables created by the SDK environment setup script (see SDK startup)

You can refer to the following simple example:

- Addition of a "hello world" user space application

7.4 Modifying the U-Boot

Prerequisites:

- the SDK is installed
- the SDK is started up
- the U-Boot is installed

The *<U-Boot installation directory>/README.HOW_TO.txt* helper file gives the commands to:

cross-compile the U-Boot

deploy the U-Boot (that is, update the software on board)

You can refer to the following simple example:

- Modification of the U-Boot

7.5 Modifying the TF-A

Prerequisites:

- the SDK is installed
- the SDK is started up
- the TF-A is installed

The *<TF-A installation directory>/README.HOW_TO.txt* helper file gives the commands to:

cross-compile the TF-A

deploy the TF-A (that is, update the software on board)

You can refer to the following simple example:

- Modification of the TF-A

7.6 Modifying the OP-TEE

Prerequisites:

- the SDK is installed
- the SDK is started up
- the OP-TEE is installed

The *<OP-TEE installation directory>/README.HOW_TO.txt* helper file gives the commands to:

cross-compile the OP-TEE

deploy the OP-TEE (that is, update the software on board)

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

8 Developing software running on Arm Cortex-M4

8.1 How to create a Cube project from scratch or open/modify an existing one from STM32Cube MPU package

Please refer to STM32CubeMP1 Package article.

9 Fast links to essential commands

If you are already familiar with the Developer Package for the STM32MPU Embedded Software distribution, fast links to the essential commands are listed below.

Information

With the links below, you will be redirected to other articles; use the *back* button of your browser to come back to these fast links

Link to the command
Starter Packages
Essential commands of the STM32MP15 Evaluation board Starter Package
Essential commands of the STM32MP15 Discovery kit Starter Package
SDK
Download and install the latest SDK
Start the SDK
Linux kernel
Download and install the latest Linux kernel
Helper file for the **Linux kernel** build, and update on board
U-Boot
Download and install the latest U-Boot
Helper file for the **U-Boot** build, and update on board
TF-A
Download and install the latest TF-A
Helper file for the **TF-A** build, and update on board
TF-A SSP
Download and install the latest TF-A SSP
Helper file for the **TF-A SSP** build, and update on board
OP-TEE
Download and install the latest OP-TEE
Helper file for the **OP-TEE** build, and update on board
Linux user space
Simple user space application
STM32Cube MPU Package

Link to the command
Download and install the latest STM32CubeMP1 Package
Create or modify a Cube project

10 How to go further?

Now that your developments are ready, you might want to switch to the STM32MP1 Distribution Package, in order to create your own distribution and to generate your own SDK and image.

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Cortex®

Software development kit (A programming package that enables a programmer to develop applications for a specific platform.)

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

Open Portable Trusted Execution Environment

Microprocessor Unit

Board support package

Hardware Abstraction Layer

(Software)Integrated development/design/debugging environment

Secure Secret Provisioning

Secure secrets provisioning

GNU dedugger, a portable debugger that runs on many Unix-like systems

Display Serial Interface (MIPI® Alliance standard)

High-Definition Multimedia Interface (HDMI standard)

debug and test protocol, named from the Joint Test Action Group that developed it

Operating System

Microcontroller Unit (MCUs have internal flash memory and are intended to operate with a minimum amount of external support ICs. They commonly are a self-contained, system-on-chip (SoC) designs.)

Cortex Microcontroller Software Interface Standard

Stable: 01.03.2021 - 10:54 / Revision: 01.03.2021 - 10:53

A quality version of this page, approved on *1 March 2021*, was based off this revision.

Contents

1 Das U-Boot

Das U-Boot ("the Universal Boot Loader" or U-Boot) is an open-source bootloader that can be used on ST boards to initialize the platform and load the Linux® kernel.

- Official website: https://www.denx.de/wiki/U-Boot
- Official manual: U-Boot project documentation and https://www.denx.de/wiki/DULG/Manual
- Official **source code** is available under git repository at [1]

Read the README file before starting using U-Boot. It covers the following topics:

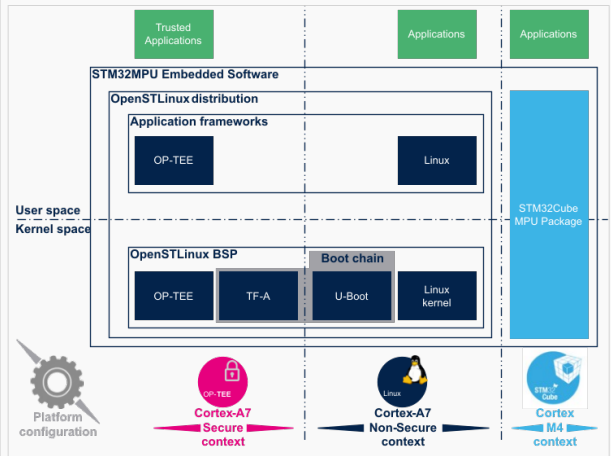
- source file tree structure
- description of CONFIG defines
- instructions for building U-Boot
- brief description of the Hush shell
- list of common environment variables

Do go further, read the documentations available in doc/ and the documentation generated by `make htmldocs` [1].

2 U-Boot overview

The STM32 MPU boot chain uses Trusted Firmware-A (TF-A) as FSBL and U-Boot as SSBL.

The same U-Boot source can also generate an alternate FSBL named SPL. The boot chain becomes: SPL as FSBL and U-Boot as SSBL.



Zoom out to STM32MPU Embedded Software

Warning

This alternate boot chain with SPL cannot be used for product development.

2.1 SPL: alternate FSBL

2.1.1 SPL description

The **U-Boot SPL** or **SPL** is an alternate first stage bootloader (FSBL).
It is a small binary (bootstrap utility) generated from the U-Boot source and stored in the internal limited-size embedded RAM.
SPL main features are the following:

- It is loaded by the ROM code.
- It performs the initial CPU and board configuration (clocks and DDR memory).
- It loads the SSBL (U-Boot) into the DDR memory.

2.1.2 SPL restrictions

Warning

SPL cannot be used for product development.

SPL is provided only as an example of the simplest SSBL with the objective to support upstream U-Boot development. However, several known limitations have been identified when SPL is used in conjunction with the minimal secure monitor provided within U-Boot for basic boot chain. These limitations apply to:

- power management
- secure access to registers
- limited features (STM32CubeProgrammer / boot from NAND Flash memory)
- SCMI support for clock and reset (not compatible with latest Linux kernel device tree)

There is no workaround for these limitations.

2.1.3 SPL execution sequence

SPL executes the following main steps in SYSRAM:

- **board_init_f()**: driver initialization including DDR initialization (mininimal stack and heap: CONFIG_SPL_STACK_R_MALLOC_SIMPLE_LEN)
- configuration of heap in DDR memory (CONFIG_SPL_SYS_MALLOC_F_LEN)
- **board_init_r()**: initialization of the other drivers activated in the SPL device tree
- loading and execution of U-Boot (or Kernel in Falcon mode[2]: README.falcon).

2.2 U-Boot: SSBL

2.2.1 U-Boot description

U-Boot is the second-stage bootloader (SSBL) of boot chain for STM32 MPU platforms.

SSBL main features are the following:

- It is configurable and expendable.
- It features a simple command line interface (CLI), allowing users to interact over a serial port console.
- It provides scripting capabilities.
- It loads the kernel into RAM and gives control to the kernel.
- It manages several internal and external devices such as NAND and NOR Flash memories, Ethernet and USB.
- It supports the following features and commands:
  - File systems: FAT, UBI/UBIFS, JFFS
  - IP stack: FTP
  - Display: LCD, HDMI, BMP for splashcreen
  - USB: host (mass storage) or device (DFU stack)

2.2.2 U-Boot execution sequence

U-Boot executes the following main steps in DDR memory:

- **Pre-relocation** initialization (common/board_f.c): minimal initialization (such as CPU, clock, reset, DDR and console) running at the CONFIG_SYS_TEXT_BASE load address.
- **Relocation**: copy of the code to the end of DDR memory.
- **Post-relocation initialization**:(common/board_r.c): initialization of all the drivers.
- **Command execution** through autoboot (CONFIG_AUTOBOOT) or console shell.
  - Execution of the boot command (by default bootcmd=CONFIG_BOOTCOMMAND):

  for example, execution of the command `bootm` to:
    - load and check images (such as kernel, device tree and ramdisk)
    - fixup the kernel device tree
    - install the secure monitor (optional) or
    - pass the control to the Linux kernel (or to another target application)

3 U-Boot configuration

The U-Boot binary configuration is based on

- **Kbuild infrastructure** (as in Linux Kernel, you can use `make menuconfig` in U-Boot)

The configurations are based on:

  - options defined in Kconfig files (CONFIG_ compilation flags)
  - the selected configuration file: configs/stm32mp*_defconfig

- **other compilation flags** defined in include/configs/stm32mp*.h (these flags are progressively migrated to Kconfig)

The file name is configured through CONFIG_SYS_CONFIG_NAME.

For STM32MP15x lines, the include/configs/stm32mp1.h file is used.

- DeviceTree: U-Boot binaries include a device tree blob that is parsed at runtime

All the configuration flags (prefixed by CONFIG_) are described in the source code, either in the README file or in the documentation directory .

For example, CONFIG_SPL activates the SPL compilation.

Hence to compile U-Boot, select the <target> and the device tree for the board in order to choose a predefined configuration.

Refer to #U-Boot_build for examples.

3.1 Kbuild

Like the kernel, the U-Boot build system is based on configuration symbols (defined in Kconfig files). The selected values are stored in a **.config** file located in the build directory, with the same makefile target. .

Proceed as follows:

- Select a predefined configuration (defconfig file in configs directory) and generate the first **.config**:

```
PC $> make <config>_defconfig.
```

- Change the U-Boot compile configuration (modify .config) by using one of the following five `make` commands:

```
PC $> make menuconfig --> menu based program
PC $> make config  --> line-oriented configuration
PC $> make xconfig --> QT program[3]
PC $> make gconfig --> GTK program
PC $> make nconfig --> ncurse menu based program
```

You can then compile U-Boot with the updated .config.

Warning: the modification is performed locally in the build directory. It will be lost after a `make distclean`.

Save your configuration to be able to use it as a defconfig file:

```
PC $> make savedefconfig
```

This target saves the current config as a defconfig file in the build directory. It can then be compared with the predefined configuration (configs/stm32mp*defconfig).

The other makefile targets are the following:

```
 PC $> make help
....
Configuration targets:
  config          - Update current config utilising a line-oriented program
  nconfig         - Update current config utilising a ncurses menu based
                    program
  menuconfig          - Update current config utilising a menu based program
  xconfig          - Update current config utilising a Qt based front-end
  gconfig          - Update current config utilising a GTK+ based front-end
  oldconfig          - Update current config utilising a provided .config as base
  localmodconfig  - Update current config disabling modules not loaded
  localyesconfig  - Update current config converting local mods to core
  defconfig          - New config with default from ARCH supplied defconfig
  savedefconfig   - Save current config as ./defconfig (minimal config)
  allnoconfig           - New config where all options are answered with no
  allyesconfig           - New config where all options are accepted with yes
  allmodconfig           - New config selecting modules when possible
  alldefconfig    - New config with all symbols set to default
  randconfig          - New config with random answer to all options
  listnewconfig   - List new options
  olddefconfig           - Same as oldconfig but sets new symbols to their
                   default value without prompting
```

3.2 Device tree

Refer to doc/README.fdt-control for details.

The board device tree has the same binding as the kernel. It is integrated within the U-Boot binaries:

- By default, it is appended at the end of the code (CONFIG_OF_SEPARATE).
- It can be embedded in the U-Boot binary (CONFIG_OF_EMBED). This is particularly useful for debugging since it enables easy .elf file loading.

A default device tree is available in the defconfig file (by setting CONFIG_DEFAULT_DEVICE_TREE).

You can either select another supported device tree using the DEVICE_TREE make flag. For stm32mp boards, the corresponding file is <dts-file-name>.dts in arch/arm/dts/stm32mp*.dts , with <dts-file-name> set to the full name of the board:

```
PC $> make DEVICE_TREE=<dts-file-name>
```

or provide a device tree blob (dtb file) resulting from the dts file compilation, by using the EXT_DTB option:

```
PC $> make EXT_DTB=boot/<dts-file-name>.dtb
```

The SPL device tree is also generated from this device tree. However to reduce its size, the U-Boot makefile uses the fdtgrep tool to parse the full U-Boot DTB and identify all the drivers required by SPL.

To do this, U-Boot uses specific device-tree flags to determine if the associated driver is initialized prior to U-Boot relocation and /or if the associated node is present in SPL :

- **u-boot,dm-pre-reloc** => present in SPL, initialized before relocation in U-Boot
- **u-boot,dm-pre-proper** => initialized before relocation in U-Boot
- **u-boot,dm-spl** => present in SPL

In the device tree used by U-Boot, these flags **need to be added in all the nodes** used in SPL or in U-Boot before relocation, and for all used handles (clock, reset, pincontrol).

To obtain a device tree file <dts-file-name>.dts that is identical to the Linux kernel one, these U-Boot properties are only added for ST boards in the add-on file <dts-file-name>-u-boot.dtsi. This file is automatically included in <dts-file-name>.dts during device tree compilation (this is a generic U-Boot Makefile behavior).

4 U-Boot command line interface (CLI)

Refer to U-Boot Command Line Interface.

If CONFIG_AUTOBOOT is activated, you have CONFIG_BOOTDELAY seconds (2s by default, 1s for ST configuration) to enter the console by pressing any key, after the line below is displayed and bootcmd is executed (CONFIG_BOOTCOMMAND):

```
Hit any key to stop autoboot:  2
```

4.1 Commands

The commands are defined in cmd/*.c . They are activated through the corresponding **CONFIG_CMD_*** configuration flag.

Use the `help` command in the U-Boot shell to list the commands available on your device:

```
Board $> help
```

Below the list of all commands extracted from U-Boot Manual (**not-exhaustive**):

- Information Commands
  - bdinfo - prints Board Info structure
  - coninfo - prints console devices and information
  - flinfo - prints Flash memory information
  - iminfo - prints header information for application image
  - help - prints online help
- Memory Commands
  - base - prints or sets the address offset
  - crc32 - checksum calculation
  - cmp - memory compare
  - cp - memory copy
  - md - memory display
  - mm - memory modify (auto-incrementing)
  - mtest - simple RAM test
  - mw - memory write (fill)
  - nm - memory modify (constant address)
  - loop - infinite loop on address range
- Flash Memory Commands
  - cp - memory copy
  - flinfo - prints Flash memory information
  - erase - erases Flash memory
  - protect - enables or disables Flash memory write protection
  - mtdparts - defines a Linux compatible MTD partition scheme
- Execution Control Commands
  - source - runs a script from memory
  - bootm - boots application image from memory

- go - starts application at address 'addr'
- Download Commands
  - bootp - boots image via network using BOOTP/TFTP protocol
  - dhcp - invokes DHCP client to obtain IP/boot params
  - loadb - loads binary file over serial line (kermit mode)
  - loads - loads S-Record file over serial line
  - rarpboot- boots image via network using RARP/TFTP protocol
  - tftpboot- boots image via network using TFTP protocol
- Environment Variables Commands
  - printenv- prints environment variables
  - saveenv - saves environment variables to persistent storage
  - setenv - sets environment variables
  - run - runs commands in an environment variable
  - bootd - default boot, that is run 'bootcmd'
- Flattened Device Tree support
  - fdt addr - selects the FDT to work on
  - fdt list - prints one level
  - fdt print - recursive printing
  - fdt mknode - creates new nodes
  - fdt set - sets node properties
  - fdt rm - removes nodes or properties
  - fdt move - moves FDT blob to new address
  - fdt chosen - fixup dynamic information
- Special Commands
  - i2c - I2C sub-system
- Storage devices
- Miscellaneous Commands
  - echo - echoes args to console
  - reset - performs a CPU reset
  - sleep - delays the execution for a predefined time
  - version - prints the monitor version

To add a new command, refer to doc/README.commands .

4.2 U-Boot environment variables

The U-Boot behavior is configured through environment variables.

Refer to Manual and README / Environment Variables.

On the first boot, U-Boot uses a default environment embedded in the U-Boot binary. You can modify it by changing the content of CONFIG_EXTRA_ENV_SETTINGS in your configuration file (for example ./include/configs/stm32mp1.h) (see README / - Default Environment).

This environment can be modified and saved in the boot device. When it is present, it is loaded during U-Boot initialization:

- To boot from *e*•MMC/SD card (CONFIG_ENV_IS_IN_MMC): at the end of the partition indicated by config field "u-boot,mmc-env-partition" in device-tree (partition named "ssbl" for ST boards).
- To boot from NAND Flash memory (CONFIG_ENV_IS_IN_UBI): in the two UBI volumes "config" (CONFIG_ENV_UBI_VOLUME) and "config_r" (CONFIG_ENV_UBI_VOLUME_REDUND).

- To boot from NOR Flash memory (CONFIG_ENV_IS_IN_SPI_FLASH): the u-boot_env mtd parttion (at offset CONFIG_ENV_OFFSET).

4.2.1 env command

The `env` command allows displaying, modifying and saving the environment in U-Boot console.

```
Board $> help env
env - environment handling commands

Usage:
env default [-f] -a - [forcibly] reset default environment
env default [-f] var [...] - [forcibly] reset variable(s) to their default values
env delete [-f] var [...] - [forcibly] delete variable(s)
env edit name - edit environment variable
env exists name - tests for existence of variable
env print [-a | name ...] - print environment
env print -e [name ...] - print UEFI environment
env run var [...] - run commands in an environment variable
env save - save environment
env set -e name [arg ...] - set UEFI variable; unset if 'arg' not specified
env set [-f] name [arg ...]
```

Example: proceed as follows to restore the default environment and save it. This is useful after a U-Boot upgrade:

```
Board $> env default -a
Board $> env save
```

4.2.2 bootcmd

"bootcmd" variable is the autoboot command. It defines the command executed when U-Boot starts (CONFIG_BOOTCOMMAND).

For stm32mp, CONFIG_BOOTCOMMAND="run bootcmd_stm32mp":

```
Board $> env print bootcmd
 bootcmd=run bootcmd_stm32mp
```

"bootcmd_stm32mp" is a script that selects the command to be executed for each boot device (see ./include/configs/stm32mp1.h), based on generic distro scripts:

- To boot from a serial/usb device: execute the `stm32prog` command.
- To boot from an *e*•MMC, SD card: boot only on the same device (bootcmd_mmc...).
- To boot from a NAND Flash memory: boot on ubifs partition on the NAND memory (bootcmd_ubi0).
- To boot from a NOR Flash memory: use the SD card (on SDMMC 0 on ST boards with bootcmd_mmc0)

```
Board $> env print bootcmd_stm32mp
```

You can then change this configuration:

- either permanently in your board file
  - default environment by CONFIG_EXTRA_ENV_SETTINGS (see ./include/configs/stm32mp1.h)
  - change CONFIG_BOOTCOMMAND value in your defconfig

```
CONFIG_BOOTCOMMAND="run bootcmd_mmc0"
```

```
CONFIG_BOOTCOMMAND="run distro_bootcmd"
```

- or temporarily in the saved environment:

```
Board $> env set bootcmd run bootcmd_mmc0
Board $> env save
```

Note: To reset the environment to its default value:

```
Board $> env default bootcmd
Board $> env save
```

4.3 Generic Distro configuration

Refer to doc/README.distro for details.

This feature is activated by default on ST boards (CONFIG_DISTRO_DEFAULTS):

- one boot command (bootmcd_xxx) exists for each bootable device.
- U-Boot is independent from the Linux distribution used.
- bootcmd is defined in ./include/config_distro_bootcmd.h

When DISTRO is enabled, the command that is executed by default is include/config_distro_bootcmd.h :

```
bootcmd=run distro_bootcmd
```

This script tries any device found in the 'boot_targets' variable and executes the associated bootcmd.

Example for mmc0, mmc1, mmc2, pxe and ubifs devices:

```
bootcmd_mmc0=setenv devnum 0; run mmc_boot
bootcmd_mmc1=setenv devnum 1; run mmc_boot
bootcmd_mmc2=setenv devnum 2; run mmc_boot
bootcmd_pxe=run boot_net_usb_start; dhcp; if pxe get; then pxe boot; fi
bootcmd_ubifs0=setenv devnum 0; run ubifs_boot
```

U-Boot searches for an **extlinux.conf** configuration file for each bootable device. This file defines the kernel configuration to be used:

- bootargs
- kernel + device tree + ramdisk files (optional)
- FIT image

4.4 U-Boot scripting capabilities

"Script files" are command sequences that are executed by the U-Boot command interpreter. This feature is particularly useful to configure U-Boot to use a real shell (hush) as command interpreter.

See U-Boot script manual for an example.

5 U-Boot build

5.1 Prerequisites

- a PC with Linux and tools:
  - see PC_prerequisites
  - #ARM cross compiler
- U-Boot source code
  - the latest STMicroelectronics U-Boot version
    - tar.xz file from Developer Package (for example STM32MP1) or from latest release on ST github [4]
    - from GITHUB[5], with `git` command

```
PC $> git clone https://github.com/STMicroelectronics/u-boot
```

  - from the Mainline U-Boot in official GIT repository [6]

```
PC $> git clone https://source.denx.de/u-boot/u-boot.git
```

5.2 ARM cross compiler

A cross compiler [7] must be installed on your Host (X86_64, i686, ...) for the ARM targeted Device architecture. In addition, the $PATH and $CROSS_COMPILE environment variables must be configured in your shell.

You can use gcc for ARM, available in:

- the SDK toolchain (see Cross-compile with OpenSTLinux SDK)

PATH and CROSS_COMPILE are automatically updated.

- an existing package

For example, install gcc-arm-linux-gnueabihf on Ubuntu/Debian: (**PC $>** sudo apt-get.

- an existing toolchain:
  - latest gcc toolchain provided by arm (https://developer.arm.com/open-source/gnu-toolchain/gnu-a/downloads/)
  - gcc v7 toolchain provided by linaro: (https://www.linaro.org/downloads/)

For example, to use *gcc-arm-9.2-2019.12-x86_64-arm-none-linux-gnueabihf.tar.xz* from arm, extract the toolchain in $HOME and update your environment with:

```
PC $> export PATH=$HOME/gcc-arm-9.2-2019.12-x86_64-arm-none-linux-gnueabihf/bin:$PATH
PC $> export CROSS_COMPILE=arm-none-linux-gnueabihf-
```

For example, to use **gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi.tar.xz**
from https://releases.linaro.org/components/toolchain/binaries/7.2-2017.11/arm-linux-gnueabi/
Unzip the toolchain in $HOME and update your environment with:

```
PC $> export PATH=$HOME/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/bin:$PATH
PC $> export CROSS_COMPILE=arm-linux-gnueabi-
```

5.3 Compilation

In the U-Boot source directory, select the defconfig for the <target> and the <device tree> for your board and then execute the `make all` command:

```
PC $> make <target>_defconfig
PC $> make DEVICE_TREE=<device tree> all
```

Use help to list other targets than all:

```
PC $> make help
```

Optionally

- **KBUILD_OUTPUT** can be used to change the output build directory in order to compile several targets in the source directory. For example:

```
PC $> export KBUILD_OUTPUT=<path>
```

- **DEVICE_TREE** can also be exported to your environment when only one board is supported. For example:

```
PC $> export DEVICE_TREE=<device-tree>
```

The result is the following:

```
PC $> export KBUILD_OUTPUT=<path>
PC $> export DEVICE_TREE=<device tree>
PC $> make <target>_defconfig
PC $> make all
```

Examples from STM32MP15 U-Boot:

The boot chain for STM32MP15x lines use **stm32mp15_trusted_defconfig**:

```
PC $> make stm32mp15_trusted_defconfig
PC $> make DEVICE_TREE=stm32mp157f-dk2 all
```

```
PC $> export KBUILD_OUTPUT=../build/stm32mp15_trusted
PC $> export DEVICE_TREE=stm32mp157c-ev1
PC $> make stm32mp15_trusted_defconfig
PC $> make all
```

5.4 Output files

The resulting U-Boot files are located in your build directory (U-Boot or KBUILD_OUTPUT).

The U-Boot generated files when TF-A is used as FSBL, with or without OP-TEE:

- **u-boot.stm32** : U-Boot binary with STM32 image header, loaded by TF-A

The STM32 image format (*.stm32) is managed by mkimage U-Boot tools and Signing_tool. It is requested by ROM code and TF-A (see STM32 header for binary files for details).

The files used to debug with gdb are

- u-boot : elf file for U-Boot

6 References

- https://u-boot.readthedocs.io/en/stable/index.html
- https://www.denx.de/wiki/pub/U-Boot/MiniSummitELCE2013/2013-ELCE-U-Boot-Falcon-Boot.pdf
- https://en.wikipedia.org/wiki/Xconfig
- https://github.com/STMicroelectronics/u-boot/releases
- https://github.com/STMicroelectronics/u-boot
- https://source.denx.de/u-boot/u-boot.git or https://github.com/u-boot/u-boot
- https://en.wikipedia.org/wiki/Cross_compiler

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

Linux® is a registered trademark of Linus Torvalds.

First Stage Boot Loader

Secondary Program Loader, *Also known as* ***U-Boot SPL***

Second Stage Boot Loader

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today for volatile random-acces ssemiconductor memories.)

Read Only Memory

Central processing unit

Doubledata rate (memory domain)

Flash memories combine high density and cost effectiveness of EPROMs with the electrical erasability of EEPROMs. For this reason, the Flash memory market is one of the most exciting areas of the semiconductor industry today and new applications requiring in system reprogramming, such as cellular telephones, automotive engine management systems, hard disk drives, PC BIOS software for Plug & Play, digital TV, set top boxes, fax and other modems, PC cards and multimedia CD-ROMs, offer the prospect of very high volume demand.

System control and management interface

Microprocessor Unit

High-Definition Multimedia Interface (HDMI standard)

Device Firmware Upgrade

Device Tree Binary (or Blob)

Memory Technology Device

Trivial File Transfer Protocol (https://en.wikipedia.org/wiki/Trivial_File_Transfer_Protocol)

Dynamic Host Configuration Protocol (See https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol for more details)

Inter-Integrated Circuit (Bi-directional 2-wire bus standard for efficient inter-IC control.)

MultimediaCard

SD memory card (https://www.sdcard.org)

Serial Peripheral Interface

Flattened uImage Tree is a packaging format used by U-Boot

Software development kit (A programming package that enables a programmer to develop applications for a specific platform.)

Trusted Firmware for Arm Cortex-A

Open Portable Trusted Execution Environment