Category:Developer Package

Contents

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

Stable: 17.06.2020 - 10:20 | Revision: 16.01.2020 - 13:48

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: 17.11.2021 - 16:13 / Revision: 16.11.2021 - 17:52

A quality version of this page, approved on *17 November 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>
```

- Edit the device tree source file to add the lines highlighted below

STM32P157C-EV1	arch/arm/boot/dts/**stm32mp15xx-edx**.dtsi	`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";`
		`};`
		`};`

STM32MP15	arch/arm/boot/dts/**stm32m**	`led {`
		`compatible = "gpio-leds";`
		`blue {`
		`label = "heartbeat";`
		`gpios = <&gpiod 11 GPIO_ACTIVE_HIGH>;`
		`linux,default-trigger = "heartbeat";`
		`default-state = "off";`

7C-DK2	p15xx-dkx.dtsi	<pre>};
		green {
		label = "stm32mp:green:user";
		gpios = <&gpioa 14 GPIO_ACTIVE_LOW>;
		default-state = "on";
		};
		};</pre>

- 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/test-drivers/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/test-drivers
/vivid/vivid.ko root@<board ip address>:/lib/modules/<kernel version>/kernel/drivers/media
/test-drivers/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>/
```

- 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 *<U-Boot version> (<U-Boot build time flag>)* CPU: **STM32MP157CAA Rev.B** Model: STMicroelectronics STM32MP157C eval daughter on eval mother Board: stm32mp1 in trusted mode (st,**stm32mp157c-ev1**) [...]
STM32MP157C-DK2	[...] U-Boot *<U-Boot version> (<U-Boot build time flag>)* 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:

- Edit the *./board/st/stm32mp1/stm32mp1.c* source file to 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
```

stm32 legacy images case building (no FIP)

- Enable CONFIG_STM32MP15x_STM32IMAGE in your defconfig (i.e. *configs/stm32mp15_trusted_defconfig*)
- Cross-compile the U-Boot: here trusted boot for **STM32MP157C-EV1** and **STM32MP157F-DK2**

```
PC $> make -f $PWD/../Makefile.sdk all UBOOT_CONFIG=trusted
UBOOT_DEFCONFIG=stm32mp15_trusted_defconfig UBOOT_BINARY=u-boot.dtb DEVICETREE="
stm32mp157c-ev1 stm32mp157f-dk2" ENABLE_FIP=0
```

- Go to the directory where the compilation results are stored

```
PC $> cd ../deploy/
```

- 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*): *sdb3* 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`

or

FIP images case building (Default)

- As mentionned in help, OpenSTLinux has activated FIP by default, so the "FIP_artifacts" folder should be specified to allow to update the FIP binary with the new U-Boot binary. When using the "SOURCES-xxxx.tar.gz" from Developer package the "FIP_DEPLOYDIR_ROOT" variable should be set as below:

```
PC $> export FIP_DEPLOYDIR_ROOT=$PWD/../../FIP_artifacts
```

- Note that to keep the original FIP binary, the "FIP_DEPLOYDIR_FIP" variable should be set as below:

```
PC $> export FIP_DEPLOYDIR_FIP=$PWD/../deploy/fip
```

- Cross-compile the U-Boot: here trusted boot for **STM32MP157C-EV1** and **STM32MP157F-DK2**

```
PC $> make -f $PWD/../Makefile.sdk all UBOOT_CONFIG=trusted
UBOOT_DEFCONFIG=stm32mp15_trusted_defconfig UBOOT_BINARY=u-boot.dtb DEVICETREE="
stm32mp157c-ev1 stm32mp157f-dk2"
```

- Go to the directory where the compilation results are stored

```
PC $> cd $FIP_DEPLOYDIR_FIP
```

- 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 that contains the FIP binary (*fip*): *sdb3* 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 fip -> ../../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 FIP binary to the dedicated partition

STM32MP157C-EV1	`PC $> dd if=fip-stm32mp157c-ev1-trusted.bin of=/dev/sdb3 bs=1M conv=fdatasync`
STM32MP157C-DK2	`PC $> dd if=fip-stm32mp157c-dk2-trusted.bin 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 <U-Boot version> (<U-Boot build time flag>)`
	`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 <U-Boot version> (<U-Boot build time flag>)`
	`CPU: STM32MP157CAC Rev.B`
	`Model: STMicroelectronics STM32MP157C-DK2 Discovery Board`
	`Board: stm32mp1 in trusted mode (st,stm32mp157c-dk2)`
	`U-Boot simple example`
	`[...]`

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:

- Edit the *./plat/st/stm32mp1/bl2_plat_setup.c* source file to 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
```

stm32 legacy images case building (no FIP)

- Update following lines in the Makefile.sdk

```
TF_A_DEVICETREE_serialboot ?= stm32mp157a-dk1 stm32mp157d-dk1 stm32mp157c-dk2 stm32mp157f-
dk2 stm32mp157c-ed1 stm32mp157f-ed1 stm32mp157a-ev1 stm32mp157c-ev1 stm32mp157d-ev1
stm32mp157f-ev1
TF_A_EXTRA_OPTFLAGS_serialboot ?= AARCH32_SP=sp_min STM32MP_UART_PROGRAMMER=1
STM32MP_USB_PROGRAMMER=1 STM32MP_USE_STM32IMAGE=1
TF_A_BINARY_serialboot ?= tf-a
TF_A_MAKE_TARGET_serialboot ?=
```

- Cross-compile the TF-A: here trusted boot for **STM32MP157C-EV1** and **STM32MP157C-DK2** for microSD card and UART

```
PC $> make -f $PWD/../Makefile.sdk TF_A_DEVICETREE="stm32mp157c-ev1 stm32mp157c-dk2"
TF_A_CONFIG="trusted serialboot" ELF_DEBUG_ENABLE='1' ENABLE_FIP=0 all
```

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

```
PC $> cd ../deploy
```

- 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`
STM32MP157C-DK2	`PC $> dd if=tf-a-stm32mp157c-dk2-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*

- 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
[...]
```

or

FIP images case building (Default)

- Cross-compile the TF-A: here trusted boot for **STM32MP157C-EV1** and **STM32MP157C-DK2** for microSD card and UART

```
PC $> make -f $PWD/../Makefile.sdk TF_A_DEVICETREE="stm32mp157c-ev1 stm32mp157c-dk2"
TF_A_CONFIG="trusted sdcard uart" ELF_DEBUG_ENABLE='1' all
```

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

```
PC $> cd ../deploy
```

- 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 fip -> ../../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-sdcard.stm32 of=/dev/sdb1 bs=1M conv=fdatasync`
STM32MP157C-DK2	`PC $> dd if=tf-a-stm32mp157c-dk2-sdcard.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*

- 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
[...]
```

- Note that according to the modification done on TF-A source code, the FIP binary may need also to be updated. In such case, to cross-compile the TF-A trusted boot for STM32MP157C-EV1 and STM32MP157C-DK2 for microSD card with FIP binary update, follow the steps:
  - The "FIP_artifacts" folder should be specified to allow to update the FIP binary with the new U-Boot binary. When using the "SOURCES-xxxx.tar.gz" from Developer package the "FIP_DEPLOYDIR_ROOT" var should be set as below:

```
PC $> export FIP_DEPLOYDIR_ROOT=$PWD/../../FIP_artifacts
```

  - To keep the original FIP binary, the "FIP_DEPLOYDIR_FIP" var should be set as below:

```
PC $> export FIP_DEPLOYDIR_FIP=$PWD/../deploy/fip
```

  - Launch cross-compilation (trusted boot for STM32MP157C-EV1 and STM32MP157C-DK2 for microSD):

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

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

```
PC $> cd $FIP_DEPLOYDIR_FIP
```

  - 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 second stage boot loader that contains the FIP binary (*fip*): *sdb3"*

```
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 Highlight|fsbl1 -> ../../sdb1
lrwxrwxrwx 1 root root 10 Feb  8 10:01 Highlight|fsbl2 -> ../../sdb2
lrwxrwxrwx 1 root root 10 Feb  8 10:01 rootfs -> ../../sdb6
lrwxrwxrwx 1 root root 10 Feb  8 10:01 fip -> ../../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 FIP binary to the dedicated partition

STM32MP157C-EV1	`PC $> dd if=fip-stm32mp157c-ev1-trusted.bin of=/dev/sdb3 bs=1M conv=fdatasync`
STM32MP157C-DK2	`PC $> dd if=fip-stm32mp157c-dk2-trusted.bin of=/dev/sdb3 bs=1M conv=fdatasync`

Information

In case you get a *permission denied* you can set more permission on */dev/sdb3* : *sudo chmod 777 /dev/sdb3*

- Reset the U-Boot shell

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

```
STM32MP> reset
```

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 fip -> ../../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
```

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

User-space Mode Setting

Universal Asynchronous Receiver/Transmitter

Stable: 17.11.2021 - 16:14 / Revision: 16.11.2021 - 17:54

A quality version of this page, approved on *17 November 2021*, 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:

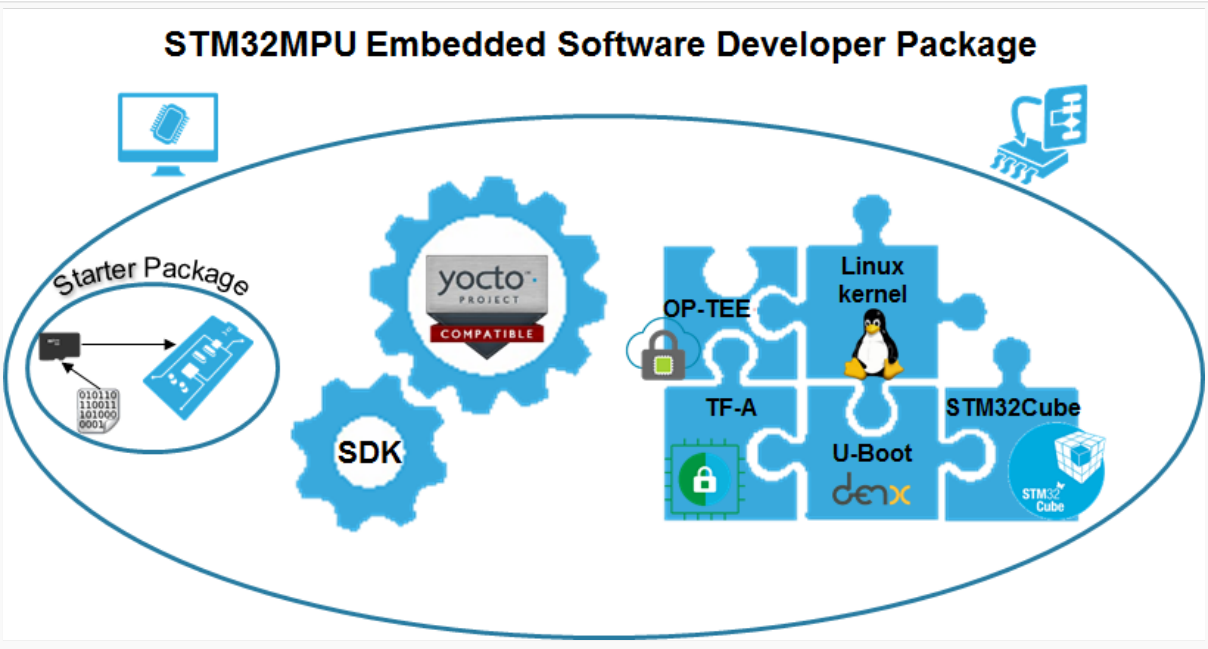
- Which STM32MPU Embedded Software Package better suits your needs (and especially the Developer Package chapter)
- STM32MPU Embedded Software distribution

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 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

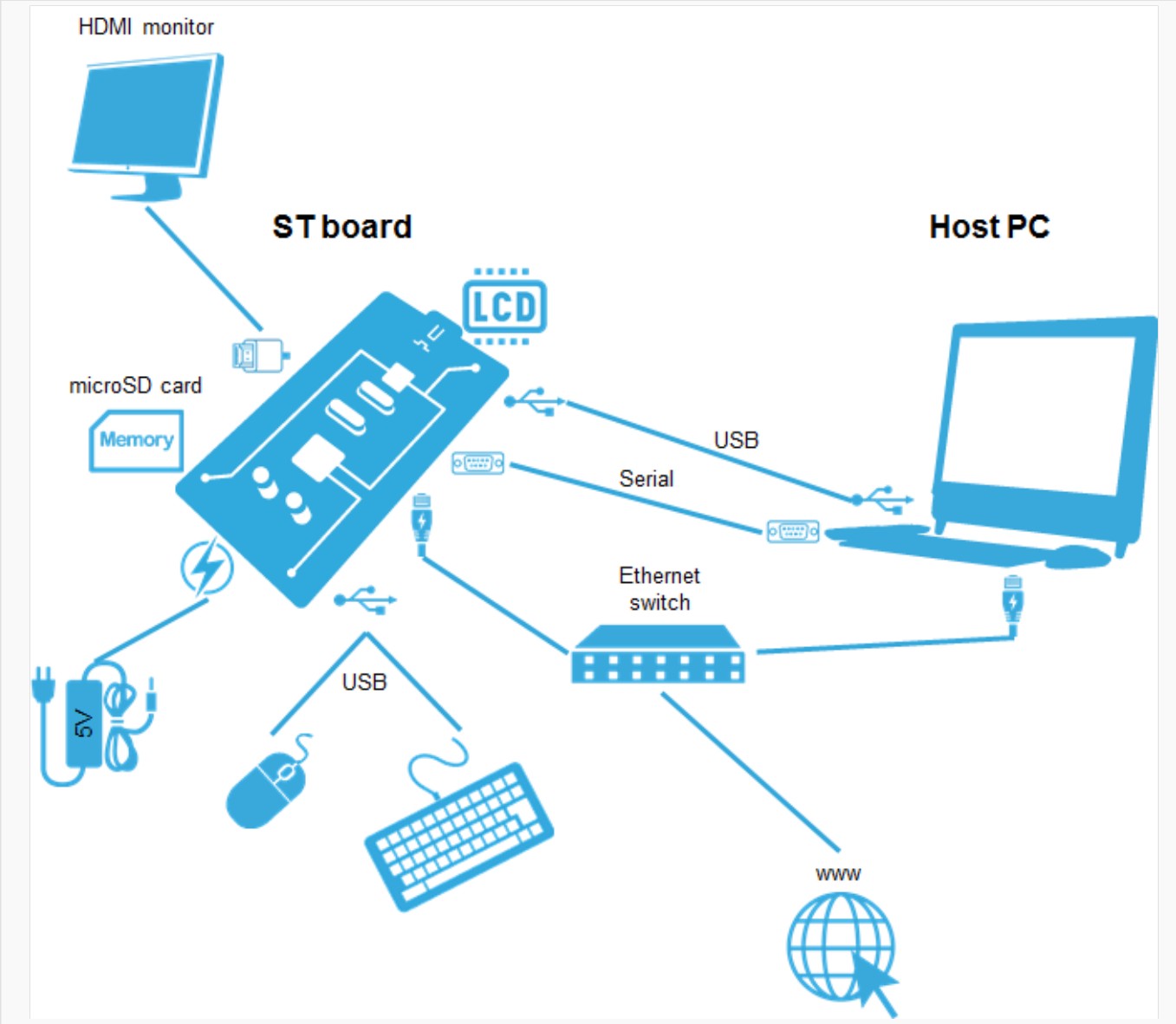
The STM32MP1 Developer Package aims at enriching a Linux-based software for the targeted product: basic knowledges about Linux are recommended to make the most of this Package.

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.



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-5.10-dunfell-mp1-21-11-17.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-v3.1.0 release
Download	You need to be logged on *my.st.com* before accessing the following link: en.SDK-x86_64-stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17.tar.xz

	• Uncompress the tarball file to get the SDK installation script
	`PC $> tar xvf en.SDK-x86_64-stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17.tar.xz`
	• If needed, change the permissions on the SDK installation script so that it is executable
	`PC $> chmod +x stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/sdk/st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-3.1.11-openstlinux-5.10-dunfell-mp1-21-11-17.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-v3.1.0 release
Installation	**PC $>** ./stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/sdk/st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-3.1.11-openstlinux-5.10-dunfell-mp1-21-11-17.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 3.1.11-openstlinux-5.10-dunfell-mp1-21-11-17 ========== 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-5.10-dunfell-mp1-21-11-17.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-v3.1.0 release
Download	You need to be logged on to *my.st.com* before accessing the following link en.SOURCES-kernel-stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17.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:
	`PC $> 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-5.10-dunfell-mp1-21-11-17.tar.xz`
	`PC $> cd stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/sources/arm-ostl-linux-gnueabi/linux-stm32mp-5.10.61-stm32mp-r2-r0`
	`PC $> tar xvf linux-5.10.61.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-5.10-dunfell-mp1-21-11-17/sources/arm-ostl-linux-gnueabi* directory, and is named *linux-stm32mp-<kernel version>*:

```
linux-stm32mp-5.10.61-r2          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-5.10.61                    Linux kernel source code directory
├── linux-5.10.61.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-5-10-dunfell-mp1-21-11-17_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-v3.1.0 release
Download	You need to be logged on to *my.st.com* before accessing the following link en.SOURCES-u-boot-stm32mp1-openstlinux-5-10-dunfell-mp1-21-11-17_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:
	`PC $> 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-5-10-dunfell-mp1-21-11-17_tar.xz`
	`PC $> cd stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/sources/arm-ostl-linux-gnueabi/u-boot-stm32mp-v2020.10-stm32mp-r2-r0`
	`PC $> tar xvf u-boot-stm32mp-v2020.10-stm32mp-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.

- In the *<Developer Package installation directory>/stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/sources/arm-ostl-linux-gnueabi* directory
  - The **U-Boot installation directory** is named *u-boot-stm32mp-<U-Boot version>*:

```
u-boot-stm32mp-v2020.10-stm32mp-r2-r0                          U-Boot installation directory
    ├── [*].patch                                    ST patches to apply during the U-Boot
preparation (see next chapter)
    ├── u-boot-stm32mp-v2020.10-stm32mp-r2                  U-Boot source code directory
    ├── Makefile.sdk                                  Makefile for the U-Boot compilation
    ├── README.HOW_TO.txt                            Helper file for U-Boot management: refere
nce for U-Boot build
    ├── series                                           List of all ST patches to apply
    └── u-boot-stm32mp-v2020.10-stm32mp-r2-r0.tar.gz        Tarball file of the U-Boot
source code
```

  - The **FIP artifacts directory** is named *FIP_artifacts*:

```
FIP_artifacts
├── arm-trusted-firmware
│   ├── bl32
│   │   ├── stm32mp157a-dk1-bl32.dtb        Device tree for TF-A → STM32MP15 Discovery
kits
│   │   ├── stm32mp157a-ev1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   ├── stm32mp157c-dk2-bl32.dtb        Device tree for TF-A → STM32MP15 Discovery
kits
│   │   ├── stm32mp157c-ed1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
```

```
│   │   ├── stm32mp157c-ev1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   ├── stm32mp157d-dk1-bl32.dtb        Device tree for TF-A → STM32MP15 Discovery
kits
│   │   ├── stm32mp157d-ev1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   ├── stm32mp157f-dk2-bl32.dtb        Device tree for TF-A → STM32MP15 Discovery
kits
│   │   ├── stm32mp157f-ed1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   ├── stm32mp157f-ev1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   └── tf-a-bl32-stm32mp15.bin        Binary file for bl32 stage
│   └── fwconfig
│       ├── stm32mp157a-dk1-fw-config-optee.dtb               Device tree for FW config
→ STM32MP15 Discovery kits
│       ├── stm32mp157a-dk1-fw-config-trusted.dtb        Device tree for FW config →
STM32MP15 Discovery kits
│       ├── stm32mp157a-ev1-fw-config-optee.dtb               Device tree for FW config
→ Evaluation boards
│       ├── stm32mp157a-ev1-fw-config-trusted.dtb        Device tree for FW config →
Evaluation boards
│       ├── stm32mp157c-dk2-fw-config-optee.dtb               Device tree for FW config
→ STM32MP15 Discovery kits
│       ├── stm32mp157c-dk2-fw-config-trusted.dtb        Device tree for FW config →
STM32MP15 Discovery kits
│       ├── stm32mp157c-ed1-fw-config-optee.dtb               Device tree for FW config
→ Evaluation boards
│       ├── stm32mp157c-ed1-fw-config-trusted.dtb        Device tree for FW config →
Evaluation boards
│       ├── stm32mp157c-ev1-fw-config-optee.dtb               Device tree for FW config
→ Evaluation boards
│       ├── stm32mp157c-ev1-fw-config-trusted.dtb        Device tree for FW config →
Evaluation boards
│       ├── stm32mp157d-dk1-fw-config-optee.dtb               Device tree for FW config
→ STM32MP15 Discovery kits
│       ├── stm32mp157d-dk1-fw-config-trusted.dtb        Device tree for FW config →
STM32MP15 Discovery kits
│       ├── stm32mp157d-ev1-fw-config-optee.dtb               Device tree for FW config
→ Evaluation boards
│       ├── stm32mp157d-ev1-fw-config-trusted.dtb        Device tree for FW config →
Evaluation boards
│       ├── stm32mp157f-dk2-fw-config-optee.dtb               Device tree for FW config
→ STM32MP15 Discovery kits
│       ├── stm32mp157f-dk2-fw-config-trusted.dtb        Device tree for FW config →
STM32MP15 Discovery kits
│       ├── stm32mp157f-ed1-fw-config-optee.dtb               Device tree for FW config
→ Evaluation boards
│       ├── stm32mp157f-ed1-fw-config-trusted.dtb        Device tree for FW config →
Evaluation boards
│       ├── stm32mp157f-ev1-fw-config-optee.dtb               Device tree for FW config
→ Evaluation boards
│       └── stm32mp157f-ev1-fw-config-trusted.dtb        Device tree for FW config →
Evaluation boards
├── optee
│   ├── tee-header_v2-stm32mp157a-dk1.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-header_v2-stm32mp157a-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-header_v2-stm32mp157c-dk2.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-header_v2-stm32mp157c-ed1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-header_v2-stm32mp157c-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-header_v2-stm32mp157d-dk1.bin        Binary file for OP-TEE OS → STM32MP15
```

```
Discovery kits
│   ├── tee-header_v2-stm32mp157d-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-header_v2-stm32mp157f-dk2.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-header_v2-stm32mp157f-ed1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-header_v2-stm32mp157f-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157a-dk1.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pageable_v2-stm32mp157a-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157c-dk2.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pageable_v2-stm32mp157c-ed1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157c-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157d-dk1.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pageable_v2-stm32mp157d-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157f-dk2.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pageable_v2-stm32mp157f-ed1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157f-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pager_v2-stm32mp157a-dk1.bin       Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pager_v2-stm32mp157a-ev1.bin       Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pager_v2-stm32mp157c-dk2.bin       Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pager_v2-stm32mp157c-ed1.bin       Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pager_v2-stm32mp157c-ev1.bin       Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pager_v2-stm32mp157d-dk1.bin       Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pager_v2-stm32mp157d-ev1.bin       Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pager_v2-stm32mp157f-dk2.bin       Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pager_v2-stm32mp157f-ed1.bin       Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   └── tee-pager_v2-stm32mp157f-ev1.bin       Binary file for OP-TEE OS → STM32MP15
Evaluation boards
├── u-boot
│   ├── u-boot-nodtb-stm32mp15.bin
│   ├── u-boot-stm32mp157a-dk1-trusted.dtb        Device tree for U-Boot → STM32MP15
Discovery kits
│   ├── u-boot-stm32mp157a-ev1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
│   ├── u-boot-stm32mp157c-dk2-trusted.dtb        Device tree for U-Boot → STM32MP15
Discovery kits
│   ├── u-boot-stm32mp157c-ed1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
│   ├── u-boot-stm32mp157c-ev1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
│   ├── u-boot-stm32mp157d-dk1-trusted.dtb        Device tree for U-Boot → STM32MP15
Discovery kits
│   ├── u-boot-stm32mp157d-ev1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
│   ├── u-boot-stm32mp157f-dk2-trusted.dtb        Device tree for U-Boot → STM32MP15
```

```
Discovery kits
│   ├── u-boot-stm32mp157f-ed1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
│   └── u-boot-stm32mp157f-ev1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
```

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

Information

Note that FIP images format is used by default. To generate legacy images (.stm32) click here

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-5-10-dunfell-mp1-21-11-17_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-v3.1.0 release
Download	You need to be logged on *my.st.com* before accessing the following link: en.SOURCES-tf-a-stm32mp1-openstlinux-5-10-dunfell-mp1-21-11-17_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:
	`PC $> 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-5-10-dunfell-mp1-21-11-17_tar.xz`
	`PC $> cd stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/sources/arm-ostl-linux-gnueabi/tf-a-stm32mp-v2.4-stm32mp-r2-r0`
	`PC $> tar xvf tf-a-stm32mp-v2.4-stm32mp-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.

- In the *<Developer Package installation directory>/stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/sources/arm-ostl-linux-gnueabi* directory,
  - The **TF-A installation directory** is named *tf-a-stm32mp-<TF-A version>*:

```
tf-a-stm32mp-v2.4-stm32mp-r2-r0                          TF-A installation directory
    ├── [*].patch                                ST patches to apply during the TF-A preparation
(see next chapter)
    ├── tf-a-stm32mp-v2.4-stm32mp-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-v2.4-stm32mp-r2-r0.tar.gz        Tarball file of the TF-A source code
```

  - The **FIP artifacts directory** is named *FIP_artifacts*:

```
FIP_artifacts
├── arm-trusted-firmware
│   ├── bl32
│   │   ├── stm32mp157a-dk1-bl32.dtb        Device tree for TF-A → STM32MP15 Discovery
kits
│   │   ├── stm32mp157a-ev1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   ├── stm32mp157c-dk2-bl32.dtb        Device tree for TF-A → STM32MP15 Discovery
kits
│   │   ├── stm32mp157c-ed1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   ├── stm32mp157c-ev1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   ├── stm32mp157d-dk1-bl32.dtb        Device tree for TF-A → STM32MP15 Discovery
kits
│   │   ├── stm32mp157d-ev1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   ├── stm32mp157f-dk2-bl32.dtb        Device tree for TF-A → STM32MP15 Discovery
kits
│   │   ├── stm32mp157f-ed1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   ├── stm32mp157f-ev1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   └── tf-a-bl32-stm32mp15.bin        Binary file for bl32 stage
│   └── fwconfig
│       ├── stm32mp157a-dk1-fw-config-optee.dtb               Device tree for FW config
→ STM32MP15 Discovery kits
│       ├── stm32mp157a-dk1-fw-config-trusted.dtb        Device tree for FW config →
STM32MP15 Discovery kits
│       ├── stm32mp157a-ev1-fw-config-optee.dtb               Device tree for FW config
→ Evaluation boards
│       ├── stm32mp157a-ev1-fw-config-trusted.dtb        Device tree for FW config →
Evaluation boards
│       ├── stm32mp157c-dk2-fw-config-optee.dtb               Device tree for FW config
→ STM32MP15 Discovery kits
│       ├── stm32mp157c-dk2-fw-config-trusted.dtb        Device tree for FW config →
STM32MP15 Discovery kits
│       ├── stm32mp157c-ed1-fw-config-optee.dtb               Device tree for FW config
→ Evaluation boards
│       ├── stm32mp157c-ed1-fw-config-trusted.dtb        Device tree for FW config →
Evaluation boards
│       ├── stm32mp157c-ev1-fw-config-optee.dtb               Device tree for FW config
→ Evaluation boards
│       ├── stm32mp157c-ev1-fw-config-trusted.dtb        Device tree for FW config →
Evaluation boards
│       ├── stm32mp157d-dk1-fw-config-optee.dtb               Device tree for FW config
→ STM32MP15 Discovery kits
│       ├── stm32mp157d-dk1-fw-config-trusted.dtb        Device tree for FW config →
STM32MP15 Discovery kits
│       ├── stm32mp157d-ev1-fw-config-optee.dtb               Device tree for FW config
→ Evaluation boards
│       ├── stm32mp157d-ev1-fw-config-trusted.dtb        Device tree for FW config →
Evaluation boards
│       ├── stm32mp157f-dk2-fw-config-optee.dtb               Device tree for FW config
→ STM32MP15 Discovery kits
│       ├── stm32mp157f-dk2-fw-config-trusted.dtb        Device tree for FW config →
STM32MP15 Discovery kits
│       ├── stm32mp157f-ed1-fw-config-optee.dtb               Device tree for FW config
→ Evaluation boards
│       ├── stm32mp157f-ed1-fw-config-trusted.dtb        Device tree for FW config →
Evaluation boards
│       ├── stm32mp157f-ev1-fw-config-optee.dtb               Device tree for FW config
→ Evaluation boards
│       └── stm32mp157f-ev1-fw-config-trusted.dtb        Device tree for FW config →
```

```
Evaluation boards
├── optee
│   ├── tee-header_v2-stm32mp157a-dk1.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-header_v2-stm32mp157a-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-header_v2-stm32mp157c-dk2.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-header_v2-stm32mp157c-ed1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-header_v2-stm32mp157c-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-header_v2-stm32mp157d-dk1.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-header_v2-stm32mp157d-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-header_v2-stm32mp157f-dk2.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-header_v2-stm32mp157f-ed1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-header_v2-stm32mp157f-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157a-dk1.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pageable_v2-stm32mp157a-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157c-dk2.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pageable_v2-stm32mp157c-ed1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157c-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157d-dk1.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pageable_v2-stm32mp157d-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157f-dk2.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pageable_v2-stm32mp157f-ed1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157f-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pager_v2-stm32mp157a-dk1.bin       Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pager_v2-stm32mp157a-ev1.bin       Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pager_v2-stm32mp157c-dk2.bin       Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pager_v2-stm32mp157c-ed1.bin       Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pager_v2-stm32mp157c-ev1.bin       Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pager_v2-stm32mp157d-dk1.bin       Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pager_v2-stm32mp157d-ev1.bin       Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pager_v2-stm32mp157f-dk2.bin       Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pager_v2-stm32mp157f-ed1.bin       Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   └── tee-pager_v2-stm32mp157f-ev1.bin       Binary file for OP-TEE OS → STM32MP15
Evaluation boards
├── u-boot
│   ├── u-boot-nodtb-stm32mp15.bin
│   ├── u-boot-stm32mp157a-dk1-trusted.dtb       Device tree for U-Boot → STM32MP15
Discovery kits
```

```
│   ├── u-boot-stm32mp157a-ev1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
│   ├── u-boot-stm32mp157c-dk2-trusted.dtb        Device tree for U-Boot → STM32MP15
Discovery kits
│   ├── u-boot-stm32mp157c-ed1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
│   ├── u-boot-stm32mp157c-ev1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
│   ├── u-boot-stm32mp157d-dk1-trusted.dtb        Device tree for U-Boot → STM32MP15
Discovery kits
│   ├── u-boot-stm32mp157d-ev1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
│   ├── u-boot-stm32mp157f-dk2-trusted.dtb        Device tree for U-Boot → STM32MP15
Discovery kits
│   ├── u-boot-stm32mp157f-ed1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
│   └── u-boot-stm32mp157f-ev1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
```

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

Information

Note that FIP images format is used by default. To generate legacy images (.stm32) click here

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 OP-TEE

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

Prerequite: the SDK is installed.

5.5.1 Downloading the OP-TEE

- The STM32MP1 OP-TEE is delivered through a tarball file named **en.SOURCES-optee-stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17.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 [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 OP-TEE - STM32MP15-Ecosystem-v3.1.0 release
Download	You need to be logged on *my.st.com* before accessing the following link: en.SOURCES-optee-stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17.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:
	`PC $> 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...):
	`PC $> tar xvf en.SOURCES-optee-stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17.tar.xz`
	`PC $> cd stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/sources/arm-ostl-linux-gnueabi/optee-os-stm32mp-3.12.0-stm32mp-r2-r0`
	`PC $> tar xvf optee-os-stm32mp-3.12.0-stm32mp-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.

- In the *<Developer Package installation directory>/stm32mp1-openstlinux-5.10-dunfell-mp1-21-03-31/sources/arm-ostl-linux-gnueabi* directory,
  - The **OP-TEE installation directory** is named *optee-os-stm32mp-<OP-TEE version>*:

```
optee-os-stm32mp-3.12.0.r2-r0                    OP-TEE installation directory
    ├── [*].patch                          ST patches to apply during the OP-TEE
preparation (see next chapter)
    ├── optee-os-stm32mp-3.12.0-stm32mp-r2          OP-TEE source code directory
    ├── Makefile.sdk                      Makefile for the OP-TEE compilation
    .
```

```
    ├── optee-os-stm32mp-3.12.0-stm32mp-r2-r0.tar.gz       Tarball file of the OP-TEE
source code
    ├── README.HOW_TO.txt                            Helper file for OP-TEE management: refere
nce for OP-TEE build
    └── series                                            List of all ST patches to apply
```

- The **FIP artifacts directory** is named *FIP_artifacts*:

```
FIP_artifacts
├── arm-trusted-firmware
│   ├── bl32
│   │   ├── stm32mp157a-dk1-bl32.dtb        Device tree for TF-A → STM32MP15 Discovery
kits
│   │   ├── stm32mp157a-ev1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   ├── stm32mp157c-dk2-bl32.dtb        Device tree for TF-A → STM32MP15 Discovery
kits
│   │   ├── stm32mp157c-ed1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   ├── stm32mp157c-ev1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   ├── stm32mp157d-dk1-bl32.dtb        Device tree for TF-A → STM32MP15 Discovery
kits
│   │   ├── stm32mp157d-ev1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   ├── stm32mp157f-dk2-bl32.dtb        Device tree for TF-A → STM32MP15 Discovery
kits
│   │   ├── stm32mp157f-ed1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   ├── stm32mp157f-ev1-bl32.dtb        Device tree for TF-A → STM32MP15 Evaluation
boards
│   │   └── tf-a-bl32-stm32mp15.bin        Binary file for bl32 stage
│   └── fwconfig
│       ├── stm32mp157a-dk1-fw-config-optee.dtb              Device tree for FW config
→ STM32MP15 Discovery kits
│       ├── stm32mp157a-dk1-fw-config-trusted.dtb        Device tree for FW config →
STM32MP15 Discovery kits
│       ├── stm32mp157a-ev1-fw-config-optee.dtb              Device tree for FW config
→ Evaluation boards
│       ├── stm32mp157a-ev1-fw-config-trusted.dtb        Device tree for FW config →
Evaluation boards
│       ├── stm32mp157c-dk2-fw-config-optee.dtb              Device tree for FW config
→ STM32MP15 Discovery kits
│       ├── stm32mp157c-dk2-fw-config-trusted.dtb        Device tree for FW config →
STM32MP15 Discovery kits
│       ├── stm32mp157c-ed1-fw-config-optee.dtb              Device tree for FW config
→ Evaluation boards
│       ├── stm32mp157c-ed1-fw-config-trusted.dtb        Device tree for FW config →
Evaluation boards
│       ├── stm32mp157c-ev1-fw-config-optee.dtb              Device tree for FW config
→ Evaluation boards
│       ├── stm32mp157c-ev1-fw-config-trusted.dtb        Device tree for FW config →
Evaluation boards
│       ├── stm32mp157d-dk1-fw-config-optee.dtb              Device tree for FW config
→ STM32MP15 Discovery kits
│       ├── stm32mp157d-dk1-fw-config-trusted.dtb        Device tree for FW config →
STM32MP15 Discovery kits
│       ├── stm32mp157d-ev1-fw-config-optee.dtb              Device tree for FW config
→ Evaluation boards
│       ├── stm32mp157d-ev1-fw-config-trusted.dtb        Device tree for FW config →
Evaluation boards
│       ├── stm32mp157f-dk2-fw-config-optee.dtb              Device tree for FW config
→ STM32MP15 Discovery kits
│       ├── stm32mp157f-dk2-fw-config-trusted.dtb        Device tree for FW config →
```

```
STM32MP15 Discovery kits
│       ├── stm32mp157f-ed1-fw-config-optee.dtb              Device tree for FW config
→ Evaluation boards
│       ├── stm32mp157f-ed1-fw-config-trusted.dtb       Device tree for FW config →
Evaluation boards
│       ├── stm32mp157f-ev1-fw-config-optee.dtb              Device tree for FW config
→ Evaluation boards
│       └── stm32mp157f-ev1-fw-config-trusted.dtb       Device tree for FW config →
Evaluation boards
├── optee
│   ├── tee-header_v2-stm32mp157a-dk1.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-header_v2-stm32mp157a-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-header_v2-stm32mp157c-dk2.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-header_v2-stm32mp157c-ed1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-header_v2-stm32mp157c-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-header_v2-stm32mp157d-dk1.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-header_v2-stm32mp157d-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-header_v2-stm32mp157f-dk2.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-header_v2-stm32mp157f-ed1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-header_v2-stm32mp157f-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157a-dk1.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pageable_v2-stm32mp157a-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157c-dk2.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pageable_v2-stm32mp157c-ed1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157c-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157d-dk1.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pageable_v2-stm32mp157d-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157f-dk2.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pageable_v2-stm32mp157f-ed1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pageable_v2-stm32mp157f-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pager_v2-stm32mp157a-dk1.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pager_v2-stm32mp157a-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pager_v2-stm32mp157c-dk2.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pager_v2-stm32mp157c-ed1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pager_v2-stm32mp157c-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pager_v2-stm32mp157d-dk1.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
│   ├── tee-pager_v2-stm32mp157d-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   ├── tee-pager_v2-stm32mp157f-dk2.bin        Binary file for OP-TEE OS → STM32MP15
Discovery kits
```

```
│   ├── tee-pager_v2-stm32mp157f-ed1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
│   └── tee-pager_v2-stm32mp157f-ev1.bin        Binary file for OP-TEE OS → STM32MP15
Evaluation boards
├── u-boot
│   ├── u-boot-nodtb-stm32mp15.bin
│   ├── u-boot-stm32mp157a-dk1-trusted.dtb        Device tree for U-Boot → STM32MP15
Discovery kits
│   ├── u-boot-stm32mp157a-ev1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
│   ├── u-boot-stm32mp157c-dk2-trusted.dtb        Device tree for U-Boot → STM32MP15
Discovery kits
│   ├── u-boot-stm32mp157c-ed1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
│   ├── u-boot-stm32mp157c-ev1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
│   ├── u-boot-stm32mp157d-dk1-trusted.dtb        Device tree for U-Boot → STM32MP15
Discovery kits
│   ├── u-boot-stm32mp157d-ev1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
│   ├── u-boot-stm32mp157f-dk2-trusted.dtb        Device tree for U-Boot → STM32MP15
Discovery kits
│   ├── u-boot-stm32mp157f-ed1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
│   └── u-boot-stm32mp157f-ev1-trusted.dtb        Device tree for U-Boot → STM32MP15
Evaluation boards
```

5.5.2 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

Information

Note that FIP images format is used by default. To generate legacy images (.stm32) please add *ENABLE_FIP=0* in the build command line

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.

5.6 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.**

5.6.1 Downloading the debug symbol files

- The STM32MP1 debug symbol files is delivered through a tarball file named **en.DEBUG-stm32mp1-openstlinux-5-10-dunfell-mp1-21-11-17_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 [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 debug symbol files - STM32MP15-Ecosystem-v3.1.0 release
Download	You need to be logged on to *my.st.com* before accessing the following link en.DEBUG-stm32mp1-openstlinux-5-10-dunfell-mp1-21-11-17_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:
	`PC $> 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-5-10-dunfell-mp1-21-11-17_tar.xz`

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-5.10-dunfell-mp1-21-11-17/images/stm32mp1 *directory:*

```
stm32mp1
├── arm-trusted-firmware
│   ├── bl32
│   │   └── debug
│   │       └── tf-a-bl32-stm32mp15.elf         Debug symbol
file for TF-A BL32
│   └── debug
│       ├── tf-a-bl2-emmc.elf                        Debug
symbol file for TF-A → TF-A for emmc boot stage
│       ├── tf-a-bl2-nand.elf                        Debug
symbol file for TF-A → TF-A for nand boot stage
│       ├── tf-a-bl2-nor.elf                       Debug
symbol file for TF-A → TF-A for nor boot stage
│       ├── tf-a-bl2-sdcard.elf                  Debug symbol
file for TF-A → TF-A for SDcard boot stage
│       ├── tf-a-bl2-uart.elf                        Debug
symbol file for TF-A → TF-A for UART downloading boot stage
│       └── tf-a-bl2-usb.elf                       Debug
symbol file for TF-A → TF-A for USB downloading boot stage
├── kernel
│   ├── config-5.10.61                               Reference
Config file for Linux kernel
│   └── vmlinux                                  Debug symbol
file for Linux kernel
├── optee
│   └── debug
│       ├── tee-stm32mp157a-dk1.elf                  Debug
symbol file for OP-TEE OS → STM32MP15 Discovery kits
│       ├── tee-stm32mp157a-ev1.elf                  Debug
symbol file for OP-TEE OS → STM32MP15 Evaluation boards
│       ├── tee-stm32mp157c-dk2.elf                  Debug
symbol file for OP-TEE OS → STM32MP15 Discovery kits
│       ├── tee-stm32mp157c-ed1.elf                  Debug
symbol file for OP-TEE OS → STM32MP15 Evaluation boards
│       ├── tee-stm32mp157c-ev1.elf                  Debug
symbol file for OP-TEE OS → STM32MP15 Evaluation boards
│       ├── tee-stm32mp157d-dk1.elf                  Debug
symbol file for OP-TEE OS → STM32MP15 Discovery kits
│       ├── tee-stm32mp157d-ev1.elf                  Debug
symbol file for OP-TEE OS → STM32MP15 Evaluation boards
│       ├── tee-stm32mp157f-dk2.elf                  Debug
symbol file for OP-TEE OS → STM32MP15 Discovery kits
│       ├── tee-stm32mp157f-ed1.elf                  Debug
symbol file for OP-TEE OS → STM32MP15 Evaluation boards
│       └── tee-stm32mp157f-ev1.elf                  Debug
symbol file for OP-TEE OS → STM32MP15 Evaluation boards
└── u-boot
    └── debug
        ├── u-boot-stm32mp157a-dk1-trusted.elf        Debug
symbol file for U-Boot → STM32MP15 Discovery kits
        ├── u-boot-stm32mp157a-ev1-trusted.elf        Debug
symbol file for U-Boot → STM32MP15 Evaluation boards
        ├── u-boot-stm32mp157c-dk2-trusted.elf        Debug
symbol file for U-Boot → STM32MP15 Discovery kits
        ├── u-boot-stm32mp157c-ed1-trusted.elf        Debug
```

```
symbol file for U-Boot → STM32MP15 Evaluation boards
        ├── u-boot-stm32mp157c-ev1-trusted.elf          Debug
symbol file for U-Boot → STM32MP15 Evaluation boards
        ├── u-boot-stm32mp157d-dk1-trusted.elf          Debug
symbol file for U-Boot → STM32MP15 Discovery kits
        ├── u-boot-stm32mp157d-ev1-trusted.elf          Debug
symbol file for U-Boot → STM32MP15 Evaluation boards
        ├── u-boot-stm32mp157f-dk2-trusted.elf          Debug
symbol file for U-Boot → STM32MP15 Discovery kits
        ├── u-boot-stm32mp157f-ed1-trusted.elf          Debug
symbol file for U-Boot → STM32MP15 Evaluation boards
        └── u-boot-stm32mp157f-ev1-trusted.elf          Debug
symbol file for U-Boot → STM32MP15 Evaluation boards
```

5.6.2 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.

STM32 MPU support inside STM32CubeIDE 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 1.8.0**
	• 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 1.8.0
		• Download the all-in-one Windows installer from my.st.com
		  • *Windows Installer - STM32CubeIDE-Win*

Installation guide	• Refer to *STM32CubeIDE installation guide (UM2563)* available on my.st.com.	
User manual	• When the installation is completed, see additional information about STM32CubeIDE in my.st.com: • *STM32CubeIDE quick start guide (UM2553)*	
Detailed release note	• Details about the content of this tool version are available in the *STM32CubeIDE release v1.8.0* release note from my.st.com	

Minor releases may be available from the update site. 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_V1-5-0.zip**.
- Download and install the STM32CubeMP1 Package

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 STM32CubeMP1 Package - v3.1.0 release
Download	You need to be logged on *my.st.com* before accessing the following link: en.STM32Cube_FW_MP1_V1-5-0.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:
	`PC $> cd <working directory path>/Developer-Package`
	• Download the archive file in this directory
	• Uncompress the archive file to get the STM32CubeMP1 Package:
	`PC $> unzip en.STM32Cube_FW_MP1_V1-5-0.zip`

Release note	Details about the content of the STM32CubeMP1 Package are available in the *STM32Cube_FW_MP1_V1.4.0/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_V1.5.0*:

```
STM32Cube_FW_MP1_V1.5.0                          STM32CubeMP1 Package: details in STM32CubeM
P1 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.

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)

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
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
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.

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

Universal Asynchronous Receiver/Transmitter

Stable: 16.03.2021 - 16:11 / Revision: 11.03.2021 - 16:15

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

This article describes how to obtain and use the **Developer Package** of the **STM32MPU Embedded Software for Android™** for any **STM32MP1 family** development platform (STM32MP15 boards) in order to develop applications on it.

It details some **prerequisite** knowledge and the development environment, and gives **step-by-step** instructions to download and install the STM32MPU Embedded Software for Android components for the Package.

Finally, it gives guidelines on upgrading (to add, remove, configure, or improve) any piece of software.

Warning

The STM32MPU distribution for Android™ is not yet available in the v3 ecosystem releases: please refer to the STM32 MP1 Developer Package for Android page for the v2 ecosystem releases (in archived wiki).

Contents

1 Developer Package content

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

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

In summary, this **Developer Package** provides:

- for the **STM32MPU distribution for Android™** (development on Arm® Cortex®-A processor):
  - the **software development kit** (SDK) update associated with Android Studio IDE, adding a service to connect your application to the coprocessor (customization)
- for the **STM32Cube MPU Package** (development on Arm® Cortex®-M processor):
  - **source code** for all pieces of software (BSP, HAL, middleware and applications)
  - the **integrated development environment (IDE)** (STM32CubeIDE)
  - a **pre-configured project for CubeMx** including default resources allocations for the Arm® Cortex®-M4
  - a **multicore application example using the coprocessor service** (Android application linked with a Arm® Cortex®-M4 firmware)

2 Prerequisite knowledge

The STM32MP1 Developer Package aims to enrich Linux-based software for the targeted product. A basic knowledge of Linux and Android is recommended in order to make the most of this Package. Reading the STM32MPU Embedded Software for Android architecture overview is also highly recommended.

3 Installing the components to develop software running on Arm Cortex-A (STM32MPU distribution for Android)

3.1 Installing the Android Studio IDE

The IDE is available in the Android developer site[1].

3.2 Installing the SDK update

The official SDK for Android must be loaded through Android Studio using the SDK manager, selecting the correct version for the Starter package version used.

The SDK can then be updated by replacing the *android.jar* file (JAVA archive file for Android) of the loaded SDK with the one provided within the Developer package (adding the coprocessor service classes). Refer to Install SDK in Android Studio.

Warning

The STM32MPU distribution for Android™ is not yet available in the v3 ecosystem releases: please refer to the STM32 MP1 Developer Package for Android page for the v2 ecosystem releases (in archived wiki).

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

4.1 Installing STM32CubeMX

Please refer to the STM32CubeMX page.

4.2 Installing STM32CubeIDE

Please refer to the STM32CubeIDE page.

5 Developing multi-core applications using the coprocessor service (CoproManager)

STM32MP1 Platforms for Android propose an environment for developing Android applications running on the Arm Cortex-A7, combined with a remote firmware offloaded in the Arm Cortex-M4.
This is based in the coprocessor service. Please check How to use coprocessor service for Android page for details.

A STCoproM4Example project, containing both Android application and Cortext-M4 firmware, is given with the STM32MP1 Developer Package for Android, and is used to illustrate below chapters.

5.1 Software running on Arm Cortex-M4

It is possible to develop Arm Cortex-M4 software based on the default resources allocated in the Linux kernel device tree (detailed in Default resources allocation for Arm Cortex-M4 in Developer Package for Android).

5.1.1 Prerequisites

Main components to be used:

- STM32CubeMX (See Installing STM32CubeMX)
- STM32CubeIDE (See Installing STM32CubeIDE)

5.1.2 Creating project from CubeMx

It's required to configure your project using STM32CubeMX. It's possible to start from the provided STM32CubeMX project `.ioc` file which reserve (not activate and not configure) the available Cortex-M4 resources as listed in Default resources allocation for Arm Cortex-M4 in Developer Package for Android.

Warning

The STM32MPU distribution for Android™ is not yet available in the v3 ecosystem releases: please refer to the STM32MP1 Developer Package for Android page for the v2 ecosystem releases (in archived wiki).

You can also refer to the configured STM32CubeMX project `copro_m4example.ioc` associated to the STCoproM4Example firmware project available on GitHub: STCoproM4Example.

5.1.3 Building your project using STM32CubeIDE

At this stage, it's possible to open the generated project using STM32CubeIDE. Pleaser refer to STM32CubeIDE user guide to understand it's usage.

The default IPC mechanism is based on a virtual UART which must be used to receive commands from the Cortex-A7 and transmit results. It's possible to refer to the example available in GitHub: STCoproM4Example firmware project.

Warning

The generated sources include CA7 device tree files are not needed in Android context. As mentioned, only the listed reserved Cortex-M4 resources can be used. Otherwise, it's required to use the STM32MP1 Distribution Package for Android.

At this stage your project has been built and a generated `.elf` file is available.

5.1.4 Loading your generated image in the device

The download of the firmware `.elf` image (Arm Cortex-M4 firmware) on the target is performed using ADB.

Information

All firmware `.elf` images using the coprocessor service must be downloaded in the `/vendor/firmware/copro/` directory.

To load the Arm Cortex-M4 firmware image on the target, execute the following commands using a terminal:

```
PC $> adb root; adb remount
PC $> adb push <path_to>/<project image>.elf /vendor/firmware/copro/
PC $> adb reboot
```

5.1.5 Testing your project

There are several solutions to test your project on the device (using STM32CubeIDE in engineering mode, using test binary, using Android application).

It's also possible to load, start and send commands manually.

Open a console:

```
PC $> adb root
PC $> adb shell
Board $> ...
```

Configure the firmware to be loaded by Linux in the Cortex-M4:

```
Board $>:/ # echo copro/<project image>.elf > /sys/class/remoteproc/remoteproc0/firmware
```

Start the firmware:

```
Board $>:/ # echo start > /sys/class/remoteproc/remoteproc0/state
```

Check that all required resources have been reserved without error

```
Board $>:/ # dmesg
```

Example of `dmesg` trace with `copro/copro_m4example.elf`:

```
[ 5183.938679] remoteproc remoteproc0: powering up m4
[ 5183.947729] remoteproc remoteproc0: Booting fw image copro/copro_m4example.elf, size
3107616
[ 5183.989105] rproc-srm-core mlahb:m4@10000000:m4_system_resources: bound mlahb:
m4@10000000:m4_system_resources:timer@40000000 (ops rproc_srm_dev_ops)
[ 5184.001239] rproc-srm-core mlahb:m4@10000000:m4_system_resources: bound mlahb:
m4@10000000:m4_system_resources:timer@40005000 (ops rproc_srm_dev_ops)
[ 5184.014541] rproc-srm-core mlahb:m4@10000000:m4_system_resources: bound mlahb:
m4@10000000:m4_system_resources:serial@4000f000 (ops rproc_srm_dev_ops)
```

```
[ 5184.027795] rproc-srm-core mlahb:m4@10000000:m4_system_resources: bound mlahb:
m4@10000000:m4_system_resources:i2c@40015000 (ops rproc_srm_dev_ops)
[ 5184.043446] rproc-srm-core mlahb:m4@10000000:m4_system_resources: bound mlahb:
m4@10000000:m4_system_resources:dac@40017000 (ops rproc_srm_dev_ops)
[ 5184.055294] rproc-srm-core mlahb:m4@10000000:m4_system_resources: bound mlahb:
m4@10000000:m4_system_resources:spi@44004000 (ops rproc_srm_dev_ops)
[ 5184.068340] rproc-srm-core mlahb:m4@10000000:m4_system_resources: bound mlahb:
m4@10000000:m4_system_resources:dma@48001000 (ops rproc_srm_dev_ops)
[ 5184.081449] rproc-srm-core mlahb:m4@10000000:m4_system_resources: bound mlahb:
m4@10000000:m4_system_resources:adc@48003000 (ops rproc_srm_dev_ops)
[ 5184.094565] rproc-srm-core mlahb:m4@10000000:m4_system_resources: bound mlahb:
m4@10000000:m4_system_resources:hash@4c002000 (ops rproc_srm_dev_ops)
[ 5184.107796] rproc-srm-core mlahb:m4@10000000:m4_system_resources: bound mlahb:
m4@10000000:m4_system_resources:rng@4c003000 (ops rproc_srm_dev_ops)
[ 5184.120918] rproc-srm-core mlahb:m4@10000000:m4_system_resources: bound mlahb:
m4@10000000:m4_system_resources:crc@4c004000 (ops rproc_srm_dev_ops)
[ 5184.134054] rproc-srm-core mlahb:m4@10000000:m4_system_resources: bound mlahb:
m4@10000000:m4_system_resources:cryp@4c005000 (ops rproc_srm_dev_ops)
[ 5184.147277] rproc-srm-core mlahb:m4@10000000:m4_system_resources: bound mlahb:
m4@10000000:m4_system_resources:qspi@58003000 (ops rproc_srm_dev_ops)
[ 5184.160489] rproc-srm-core mlahb:m4@10000000:m4_system_resources: bound mlahb:
m4@10000000:m4_system_resources:m4_led (ops rproc_srm_dev_ops)
[ 5184.173700] remoteproc0#vdev0buffer: assigned reserved memory node vdev0buffer@10042000
[ 5184.182181] virtio_rpmsg_bus virtio0: rpmsg host is online
[ 5184.182335] virtio_rpmsg_bus virtio0: creating channel rpmsg-tty-channel addr 0x0
[ 5184.187346] remoteproc0#vdev0buffer: registered virtio0 (type 7)
[ 5184.200672] remoteproc remoteproc0: remote processor m4 is now up
[ 5184.204352] rpmsg_tty virtio0.rpmsg-tty-channel.-1.0: new channel: 0x400 -> 0x0 :
ttyRPMSG0
```

Send command:

```
Board $>:/ # echo <command> > /dev/ttyRPMSG0
```

Receive returned data (to be executed in another console before sending the command):

```
Board $>:/ # od -c < /dev/ttyRPMSG0
```

Stop the firmware:

```
Board $>:/ # echo stop > /sys/class/remoteproc/remoteproc0/state
```

5.2 Android application running on Arm Cortex-A7

Please refer to the Android developer guide[2] for generic information about standard Android application development.

Then based on the SDK delivered within the Developer Package for Android (see Installing the SDK update), this is possible to develop an application which can use the coprocessor service (CoproManager), in order to interact with remote Arm Cortex-M4 core software.

Coprocessor service detailed information and API are described in dedicated page. Please see How to use coprocessor service for Android.

As example, you can refer to the STCoproM4Example application available on GitHub.

6 How to go further

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

7 References

- https://developer.android.com/studio
- https://developer.android.com/guide

Universal Asynchronous Receiver/Transmitter

Stable: 23.06.2020 - 14:55 / Revision: 12.06.2020 - 10:12

A quality version of this page, approved on *23 June 2020*, was based off this revision.

Click on the link below that corresponds to your targeted distribution, to discover the software Packages (Starter, Developer and Distribution) delivered for the STM32 MPU microprocessor devices:

Which STM32MPU Embedded Software Package better suits your needs

Which STM32MPU Embedded Software Package for Android better suits your needs