# How to cross-compile with the Developer Package

## 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.

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.

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.

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

**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 0xf0000000
```

*STM32MP157C-DK2*

```
[    0.000000] cma: Reserved 128 MiB at 0xd4000000
```

- 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 0xe0000000
```

*STM32MP157C-DK2*

```
[    0.000000] cma: Reserved 256 MiB at 0xcc000000
```

## 4 Modifying the Linux kernel device tree

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

- With the board started; check that the user green LED (LD3 for STM32MP157C-EV1, LD5 for STM32MP157C-DK2) is disabled
- Go to the <Linux kernel source directory>

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

*STM32P157C-EV1*

- Edit the *arch/arm/boot/dts/stm32mp157c-ed1.dts* device tree source file
- Add the lines highlighted below

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

*STM32MP157C-DK2*

- Edit the *arch/arm/boot/dts/stm32mp157a-dk1.dts* device tree source file
- Add the lines highlighted below

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

- Go to the <Linux kernel build directory>

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

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

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

- Update the device tree blobs on the board

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

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/
```

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

- Reboot the board

```
Board $> reboot
```

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

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

## 6 Modifying/adding an external Linux kernel module

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

### 6.1 Modifying an external in-tree Linux kernel module

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

- Go to the <Linux kernel source directory>

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

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

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

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

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

        return ret;
}
```

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

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

        return 0;
}
```

- Go to the <Linux kernel build directory>

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

- Cross-compile the Linux kernel modules

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

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

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

OR

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

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

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

- Insert the vivid kernel module into the Linux kernel

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

- Remove the vivid kernel module from the Linux kernel

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

### 6.2 Adding an external out-of-tree Linux kernel module

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

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

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

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

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

Example:
**PC $>** export KERNEL_SRC_PATH=$PWD/linux-stm32mp-4.19-r0/build

- Create a directory for this kernel module example

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

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

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

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

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

static void __exit kernel_module_example_exit(void)
```

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

All the indentations in a makefile are tabulations

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

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

# Path to the directory that contains the Linux kernel source code
# and the configuration file (.config)
KERNEL_DIR ?=  $(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:
```

- 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*
- 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/<
```

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 2018.11-stm32mp-r2

CPU: STM32MP157CAA Rev.B
Model: STMicroelectronics STM32MP157C eval daughter on eval mother
Board: stm32mp1 in trusted mode (st,stm32mp157c-ev1)
[...]
```

*STM32MP157C-DK2*

```
[...]
U-Boot 2018.11-stm32mp-r2

CPU: STM32MP157CAC Rev.B
Model: STMicroelectronics STM32MP157C-DK2 Discovery Board
Board: stm32mp1 in trusted mode (st,stm32mp157c-dk2)
[...]
```

- Go to the <U-Boot source directory>

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

```
Example:
PC $> cd u-boot-stm32mp-2018.11-r0/u-boot-2018.11
```

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

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

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

```
        return 0;
}
```

- Get the list of supported configurations with the following command

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

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

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

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

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

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

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

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

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

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

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

```
PC $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 10 Feb  8 08:57 bootfs -> ../../sdb4
lrwxrwxrwx 1 root root 10 Feb  8 08:57 fsbl1 -> ../../sdb1
lrwxrwxrwx 1 root root 10 Feb  8 08:57 fsbl2 -> ../../sdb2
lrwxrwxrwx 1 root root 10 Feb  8 08:57 rootfs -> ../../sdb6
lrwxrwxrwx 1 root root 10 Feb  8 08:57 ssbl -> ../../sdb3
lrwxrwxrwx 1 root root 10 Feb  8 08:57 userfs -> ../../sdb7
lrwxrwxrwx 1 root root 10 Feb  8 08:57 vendorfs -> ../../sdb5
```

- Copy the U-Boot binary to the dedicated partition

*STM32MP157C-EV1*

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

*STM32MP157C-DK2*

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

- Reset the U-Boot shell

```
STM32MP> reset
```

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

*STM32MP157C-EV1*

```
[...]
U-Boot 2018.11-stm32mp-r2

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 2018.11-stm32mp-r2

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:    Using SDMMC
[...]
```

- Go to the <TF-A source directory>

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

```
 Example:
 PC $> cd tf-a-stm32mp-2.0-r0/arm-trusted-firmware-2.0
```

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

- Add a log information in the *print_reset_reason* function

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

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

- Get the list of supported configurations with the following command

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

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

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

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

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

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

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

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

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

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
```

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

*STM32MP157C-DK2*

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

- Reset the U-Boot shell

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

```
STM32MP> reset
```

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

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

## 9 Adding a "hello world" user space example

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

### 9.1 Source code file

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

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

- Create a directory for this user space example

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

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

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

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

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

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

## 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-openstlinux_weston-linux-gnueabi-gcc -march=armv7ve -mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7 --sysroot=<
```

- 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-openstlinux_weston-linux-gnueabi-gcc -march=armv7ve -mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7 --sysroot=<
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-openstlinux_weston-linux-gnueabi-ld --sysroot=<SDK installation directory>/SDK/sysroots/cortexa7t2hf-neon-vfpv4-openstli
```

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

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)
```

- Cross-compile the project

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

#### 9.2.3 Autotools-based project

This method creates a project based on GNU autotools.

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

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

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

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

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

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

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

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

- Generate the links towards SDK scripts

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

- Cross-compile the project

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

### 9.3 Deploy and execute on board

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

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

- Execute this user space example

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

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

## 10 Tips

### 10.1 Creating a mounting point

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

- Find the partition label associated with the boot file system

```
Board $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 15 Jan 23 17:00 bootfs -> ../../mmcblk0p4
lrwxrwxrwx 1 root root 15 Jan 23 17:00 fsbl1 -> ../../mmcblk0p1
lrwxrwxrwx 1 root root 15 Jan 23 17:00 fsbl2 -> ../../mmcblk0p2
lrwxrwxrwx 1 root root 15 Jan 23 17:00 rootfs -> ../../mmcblk0p6
lrwxrwxrwx 1 root root 15 Jan 23 17:00 ssbl -> ../../mmcblk0p3
lrwxrwxrwx 1 root root 15 Jan 23 17:00 userfs -> ../../mmcblk0p7
lrwxrwxrwx 1 root root 15 Jan 23 17:00 vendorfs -> ../../mmcblk0p5
```

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

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