Cross-compile with OpenSTLinux SDK

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

Contents

1 Article purpose

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

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

Warning

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

1.1 Modifying the Linux kernel

Prerequisites:

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

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

configure the Linux kernel

cross-compile the Linux kernel

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

You can refer to the following simple examples:

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

1.2 Adding external out-of-tree Linux kernel modules

Prerequisites:

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

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

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

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

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

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

# Path to the directory that contains the Linux kernel source code
# and the configuration file (.config)
KERNEL_DIR ?= <Linux kernel path>

# Path to the directory that contains the generated objects
DESTDIR ?= <Linux kernel installation directory>

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

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

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

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

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

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

You can refer to the following simple example:

- Addition of an external out-of-tree module

1.3 Adding Linux user space applications

Prerequisites:

- the SDK is installed
- the SDK is started up

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

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

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

You can refer to the following simple example:

- Addition of a "hello world" user space application

1.4 Modifying the U-Boot

Prerequisites:

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

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

cross-compile the U-Boot

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

You can refer to the following simple example:

- Modification of the U-Boot

1.5 Modifying the TF-A

Prerequisites:

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

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

cross-compile the TF-A

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

You can refer to the following simple example:

- Modification of the TF-A

1.6 Modifying the OP-TEE

Prerequisites:

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

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

cross-compile the OP-TEE

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

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