Cross-compile with OpenSTLinux SDK

Contents

Stable: 17.11.2021 - 16:11 / Revision: 16.11.2021 - 11:23

Stable: 17.11.2021 - 16:46 / Revision: 17.11.2021 - 15:58

The content format pdf is not supported by the content model wikitext.

Return to Main Page.

Stable: 17.11.2021 - 16:46 / Revision: 17.11.2021 - 15:58

You do not have permission to edit this page, for the following reasons:

- The action you have requested is limited to users in one of the groups: Administrators, Editors, Reviewers, Selected_editors, ST_editors.
- The action "Read pages" for the draft version of this page is only available for the groups ST_editors, ST_readers, Selected_editors, sysop, reviewer

---

You can view and copy the source of this page.

<noinclude> [[Category:SDK]] {{PublicationRequestId | 7646 | 2018-06-04 | PhilipS}} ==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]]}} </noinclude> ===Modifying the Linux kernel=== Prerequisites: * the [[STM32MP1 Developer Package#Installing the SDK|SDK is installed]] * the [[STM32MP1 Developer Package#Starting up the SDK|SDK is started up]] * the [[STM32MP1 Developer Package#Installing the Linux kernel|Linux kernel is installed]] {{highlight|'''The ''<Linux kernel installation directory>/README.HOW_TO.txt'' helper file gives the commands to:'''}}
  configure the Linux kernel
  cross-compile the Linux kernel
 </ /span> deploy the Linux kernel (that is, update the software on board)
 You can refer to the following simple examples: * [[How to cross-compile with the Developer Package#Modifying the Linux kernel configuration|Modification of the kernel configuration]] * [[How to cross-compile with the Developer Package#Modifying the Linux kernel device tree|Modification of the device tree]] * [[How to cross-compile with the Developer Package#Modifying a built-in Linux kernel device driver|Modification of a built-in device driver]] * [[How to cross-compile with the Developer Package#Modifying an external in-tree Linux kernel module|Modification of an external in-tree module]] ===Adding external out-of-tree Linux kernel modules=== Prerequisites: * the [[STM32MP1 Developer Package#Installing the SDK|SDK is installed]] * the [[STM32MP1 Developer Package#Starting up the SDK|SDK is started up]] * the [[STM32MP1 Developer Package#Installing the Linux kernel|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: <pre> # 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 </pre> Such module is then cross-compiled with the following commands: $ make clean $ make $ make install You can refer to the following simple example: * [[How to cross-compile with the Developer Package#Adding an external out-of-tree Linux kernel module|Addition of an external out-of-tree module]] ===Adding Linux user space applications=== Prerequisites: * the [[STM32MP1 Developer Package#Installing the SDK|SDK is installed]] * the [[STM32MP1 Developer Package#Starting up the SDK|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 [[SDK for OpenSTLinux distribution#Native and target sysroots|target sysroot]]) * the environment variables created by the SDK environment setup script (see [[SDK for OpenSTLinux distribution#SDK startup|SDK startup]]) You can refer to the following simple example: * [[How to cross-compile with the Developer Package#Adding a "hello world" user space example|Addition of a "hello world" user space application]] ===Modifying the U-Boot=== Prerequisites: * the [[STM32MP1 Developer Package#Installing the SDK|SDK is installed]] * the [[STM32MP1 Developer Package#Starting up the SDK|SDK is started up]] * the [[STM32MP1 Developer Package#Installing the U-Boot|U-Boot is installed]] {{highlight|'''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: * [[How to cross-compile with the Developer Package#Modifying the U-Boot|Modification of the U-Boot]] ===Modifying the TF-A=== Prerequisites: * the [[STM32MP1 Developer Package#Installing the SDK|SDK is

Boot]] ===Modifying the TF-A=== Prerequisites: * the [[STM32MP1 Developer Package#Installing the SDK|SDK is installed]] * the [[STM32MP1 Developer Package#Starting up the SDK|SDK is started up]] * the [[STM32MP1 Developer Package#Installing the TF-A|TF-A is installed]] {{highlight|"'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: * [[How to cross-compile with the Developer Package#Modifying the TF-A|Modification of the TF-A]] ===Modifying the OP-TEE=== Prerequisites: * the [[STM32MP1 Developer Package#Installing the SDK|SDK is installed]] * the [[STM32MP1 Developer Package#Starting up the SDK|SDK is started up]] * the [[STM32MP1 Developer Package#Installing the OP-TEE|OP-TEE is installed]] {{highlight|"'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)

Templates used on this page:

- Template:Highlight (view source)
- Template:Info (view source)
- Template:STDarkBlue (view source)

Return to Main Page.