Category:How to build software

Contents

Stable: 17.06.2020 - 15:26 / Revision: 12.12.2019 - 13:22

Category:How to build software

Stable: 17:00, 17 June 2020 Revision: 12:12, 2 June 2020

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

This category groups together articles explaining how to build software for the STM32MPU Embedded Software distribution, and the STM32 MPUs microprocessor devices and boards.

Pages in category "How to build software"

The following 6 pages are in this category, out of 6 total.

- How to build Linux kernel user space tools
- How to compile the device tree with the Developer Package
- How to create an SDK for OpenSTLinux distribution
- How to cross-compile with the Developer Package
- How to cross-compile with the Distribution Package
- How to integrate an external software package

Stable: 22.04.2020 - 12:38 / Revision: 22.04.2020 - 12:37

A quality version of this page, approved on *22 April 2020*, was based off this revision.

Contents

1 Article purpose

This article provides the basic information needed to build the user space tools available on the Linux® kernel.

2 Introduction

The Linux kernel provides some user-space tools that are available in the tools directory [1] of the source tree.

These tools are not compiled by default when compiling the Linux kernel for the target board. They can be compiled independently, depending on the user's needs.

3 Installing the trace and debug tool on your target board

3.1 Using the STM32MPU Embedded Software distribution

3.1.1 Developer Package

Prerequisites, please ensure:

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

The available user space tools can be listed by using the following commands in the Linux kernel source root path:

```
PC $> cd <Linux_kernel_source_path>
PC $> make tools/help O="<Linux_kernel_build_dir>" (optional)
Possible targets:

 acpi                   - ACPI tools
 cgroup                 - cgroup tools
 cpupower               - a tool for all things x86 CPU power
 firewire               - the userspace part of nosy, an IEEE-1394 traffic sniffer
 freefall               - laptop accelerometer program for disk protection
 gpio                   - GPIO tools
 hv                     - tools used when in Hyper-V clients
 iio                    - IIO tools
 kvm_stat               - top-like utility for displaying kvm statistics
 leds                   - LEDs  tools
 liblockdep             - user-space wrapper for kernel locking-validator
 bpf                    - misc BPF tools
 perf                   - Linux performance measurement and analysis tool
 selftests              - various kernel selftests
 spi                    - spi tools
 objtool                - an ELF object analysis tool
 tmon                   - thermal monitoring and tuning tool
 turbostat              - Intel CPU idle stats and freq reporting tool
 usb                    - USB testing tools
 virtio                 - vhost test module
 vm                     - misc vm tools
 wmi                        - WMI interface examples
 x86_energy_perf_policy - Intel energy policy tool
```

Note: some tools are made for specific platforms (ARM, x86, RISC, and so on), so cannot be used on STM32MPU systems

The following basic steps must be done :

- Compiling the application:

  - Refer to *<Linux kernel installation directory>/README.HOW_TO.txt* helper file to know how to compile (the latest version of this helper file is also available in GitHub: README.HOW_TO.txt).

  - Ensure at least that the kernel configuration file is generated (.config) (information available in README.HWO_TO file previously mentioned)

  - Compile the expected ***tool (i.e. iio, spi...)***

```
PC $> cd <Linux_kernel_source_path>/tools
PC $> make <tool> [O=<Linux_kernel_build_dir>]
```

Note: The 'O' option can be used to specify the output directory

- Deploying the application on a board:

  -The binary is generated in the directory path *<Linux_kernel_build_dir>/**<tool>***

  -Push it onto the board.

```
PC $> scp <tool_binary> root@<board ip address>:/<dest_path>
```

PS: please ensure that the <dest_path> is known in the $PATH to execute the tool binary from anywhere on the target board.

3.1.2 Distribution Package

There is currently no recipe to build the Linux kernel user space tools, so the Developer package has to be used.

4 References

- /tools

Linux® is a registered trademark of Linus Torvalds.

Central processing unit

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Industrial I/O Linux subsystem

Executable and linkable file

Stable: 01.12.2020 - 16:59 / Revision: 01.12.2020 - 15:24

A quality version of this page, approved on *1 December 2020*, was based off this revision.

Contents

1 Purpose

This article explains how to update the boot chain (trusted mode) for a "custom" device tree.
In particular, STM32CubeMX can generate a "custom" device tree.

This article describes how to update the device tree compiled (DTB) part of the boot binaries.

2 Rationale

There are various rationale for using a custom *device tree*, such as:

- the description of a new and private board
- the swapping of some internal peripherals from Cortex®-M side to Cortex-A side (or the opposite)

3 Prerequisites

Information

Even if STMicroelectronics strongly recommends to use a Linux® environment, the steps described in this article can be executed in a WSL2 (Windows Sub-system Linux 2) environment.

Compiling a new device tree means updating three software components belonging to the complete boot chain (trusted mode), Trusted Firmware A (TF-A), u-boot, and Linux kernel.

The material required to update the above software components is the following:

- Starter package:
  - the flashlayout as well as the images to flash, provided within the **en.FLASH-stm32mp1-openstlinux<YY>-<MM>-<DD>.tar.xz** file (download it from st.com)
- Developer package:
  - the component sources and patches, provided within the **en.SOURCES-stm32mp1-openstlinux-<YY>-<MM>-<DD>.tar.xz** file (download it from st.com)
  - the SDK toolchain, provided within the **en.SDK-x86_64-stm32mp1-openstlinux-<YY>-<MM>-<DD>.tar.xz** file (download it from st.com)
- the STM32CubeProgrammer, which is the tool used to flash the images and binaries into the target.

- **Custom device tree sources**:
  - In the rest of this document, we assume that the custom device tree is generated by STM32CubeMX and stored in a *MyDeviceTree_fromCubeMX.tar.xz* tarball with following file tree:

```
MyDeviceTree_fromCubeMX
|-- kernel
|   |-- stm32mp157c-mydevicetree-mx.dts
|-- tf-a
|  |-- stm32mp15-mx.h
|  |-- stm32mp157c-mydevicetree-mx.dts
|-- u-boot
|  |-- stm32mp15-mx.h
|  |-- stm32mp157c-mydevicetree-mx-u-boot.dtsi
|  |-- stm32mp157c-mydevicetree-mx.dts
```

- Make sure the hardware configuration described in PC_prerequisites#Linux PC **has been executed** (even with a WSL2 setup)

4 Preparing your environment

It is recommended to organize the numerous inputs described in #Prerequisites in your environment.
First create a dedicated *WORKDIR* under your *HOME* folder and copy there all the inputs listed in #Prerequisites:

PC $> cd $HOME
PC $> mkdir WORKDIR
PC $> cd WORKDIR
**PC $>** export WORKDIR="$PWD"
PC $> tar --strip-components=1 -xf *<FLASH-st-image-weston-openstlinux-weston-stm32mp1.tar.xz>* -C $WORKDIR/
PC $> tar --strip-components=1 -xf *<SOURCES-st-image-weston-openstlinux-weston-stm32mp1.tar.xz>* -C $WORKDIR/
PC $> tar --strip-components=1 -xf *<SDK-st-image-weston-openstlinux-weston-stm32mp1.tar.xz>* -C $WORKDIR/
PC $> tar xf *<MyDeviceTree_fromCubeMX.tar.xz>* -C $WORKDIR/

Then proceed with the SDK installation.

The commands described in the rest of the document must be run in an SDK environment context: (Starting_up_the_SDK).

5 Updating the kernel device tree

Since 'extlinux.conf' explicitly points to the DTB, just update the kernel device tree by replacing the DTB file of the '/boot' partition. The path used must be something like '/boot/<devicetree>.dtb'.

The following chapters describe the procedure to generate and copy the new DTB into the target.

5.1 Kernel : unpack and patch sources

Information

The procedure below is an extract of the README.HOW_TO.txt file which is available in *$WORKDIR /sources/arm-ostl-linux-gnueabi/linux-stm32mp-**. Please notice a *grep "$>" README.HOW_TO.txt* describes the few commands needed to build the artifact of kernel

Run the following command into a shell:

PC $> pushd $WORKDIR
PC $> mkdir -p kernel
PC $> tar xf sources/arm-ostl-linux-gnueabi/linux-stm32mp-*/linux-*.tar.xz -C kernel
PC $> mv kernel/linux-* kernel/kernel-sources/
PC $> pushd kernel/kernel-sources/
PC $> for p in $(ls -1 ../../sources/arm-ostl-linux-gnueabi/linux-stm32mp-*/*.patch); do patch -p1 < $p; done
PC $> popd
PC $> popd

5.2 Kernel : copy the DTS into the source code

PC $> pushd $WORKDIR
PC $> cp -r MyDeviceTree_fromCubeMX/kernel/* kernel/kernel-sources/arch/arm/boot/dts/
PC $> popd

5.3 Kernel : regenerate the kernel DTB

Information

The procedure below is an extract of the README.HOW_TO.txt file which is available in *$WORKDIR /sources/arm-ostl-linux-gnueabi/linux-stm32mp-**. Please notice a *grep "$>" README.HOW_TO.txt* describes the few commands needed to build the artifact of kernel

PC $> pushd $WORKDIR/kernel/kernel-sources
**PC $>** make O="$PWD/../build" multi_v7_defconfig
PC $> for f in `ls -1 ../../sources/arm-ostl-linux-gnueabi/linux-stm32mp-*/fragment*.config`; do scripts/kconfig/merge_config.sh -m -r -O $PWD/../build $PWD/../build/.config $f; done
**PC $>** yes "" | make ARCH=arm oldconfig O="$PWD/../build"
**PC $>** make stm32mp157c-mydevicetree-mx.dtb LOADADDR=0xC2000040 O="$PWD/../build"

```
PC $> popd
PC $> ls -l $WORKDIR/kernel/build/arch/arm/boot/dts/stm32mp157c-mydevicetree-mx.dtb
```

5.4 Kernel : copy the DTB into bootfs

First of all #Updating bootfs with the new DTB so that it is taken it into account at the next boot of the target.

Then, if needed, #Updating extlinux for the target according to this new DTB filename. This is only required if the filename of the generated DTB is different from the one used by extlinux to boot.

6 Updating the u-boot device tree

To update the u-boot device tree, replace the DTB part of the u-boot binary.
Adding a new device tree to the u-boot source code forces the Makefile to regenerate a new u-boot.stm32 containing the new DTS.
The following chapters describe the procedure to update the u-boot device tree.

6.1 U-boot : unpack and patch sources

Information

The procedure below is an extract of the README.HOW_TO.txt file which is available in *$WORKDIR /sources/arm-ostl-linux-gnueabi/u-boot-stm32mp-**. Please notice a *grep "$>" README.HOW_TO.txt* describes the few commands needed to build the artifact of u-boot

PC $> pushd $WORKDIR
PC $> mkdir -p u-boot
PC $> tar xf sources/arm-ostl-linux-gnueabi/u-boot-stm32mp-*/v*.tar.gz -C u-boot
PC $> mv u-boot/u-boot* u-boot/u-boot-sources/
PC $> pushd u-boot/u-boot-sources
PC $> for p in ../../sources/arm-ostl-linux-gnueabi/u-boot-stm32mp-*/*.patch; do patch -p1 < $p; done
PC $> popd

6.2 U-boot : copy the DTS in the u-boot source code

PC $> pushd $WORKDIR
PC $> cp MyDeviceTree_fromCubeMX/u-boot/* u-boot/u-boot-sources/arch/arm/dts/
PC $> popd

Starting from *u-boot* 2019.04 version the device tree to be compiled must be explicitly added to the *dts Makefile*, *u-boot/u-boot-sources/arch/arm/dts/Makefile*:

```
dtb-y += stm32mp157c-mydevicetree-mx.dtb
targets += $(dtb-y)
```

6.3 U-boot : regenerate u-boot.stm32

Information

The procedure below is an extract of the README.HOW_TO.txt file which is available in *$WORKDIR /sources/arm-ostl-linux-gnueabi/u-boot-stm32mp-**. Please notice a *grep "$>" README.HOW_TO.txt* describes the few commands needed to build the artifact of u-boot

PC $> pushd $WORKDIR/u-boot/u-boot-sources
PC $> make stm32mp15_*<config>*_defconfig

<config> : could be *trusted* or *basic* according the boot type

PC $> make DEVICE_TREE=*<device tree>* all

<device tree> : is the device tree just copied, i.e.: stm32mp157c-mydevicetree-mx

PC $> popd

PC $> ls -l $WORKDIR/u-boot/u-boot-sources/**u-boot.stm32**

6.4 U-boot : copy the u-boot into the target

- Because of 'extlinux' and before flashing the new u-boot.stm32, make sure #Updating extlinux is compliant with the 'compatible' value in the DTS file.
- Then flash the u-boot.stm32 into the 'ssbl' partition of the target using STM32CubeProgrammer.

7 Updating the TF-A device tree

To update the TF-A device tree, replace the DTB part of the TF-A binary.
The TF-A binary allocates a 'fixed' area for the DTB, just after the 'mkimage" headers. If the DTB is smaller than the reserved area, the remaining memory is padded with zero.

Below the procedure to generate TF-A with a new DTB and then flash it on the target:

7.1 TF-A : unpack and patch sources

Information

The procedure below is an extract of the README.HOW_TO.txt file which is available in *$WORKDIR /sources/arm-ostl-linux-gnueabi/tf-a-stm32mp-**. Please notice a *grep "$>" README.HOW_TO.txt* describes the few commands needed to build the artifact of tf-a

PC $> pushd $WORKDIR
PC $> mkdir -p tf-a
PC $> tar xf sources/arm-ostl-linux-gnueabi/tf-a-stm32mp-[0-9]*/*.tar.gz -C tf-a
PC $> mv tf-a/tf-a-* tf-a/tf-a-sources
PC $> pushd tf-a/tf-a-sources
PC $> for p in ../../sources/arm-ostl-linux-gnueabi/tf-a-stm32mp-[0-9]*/*.patch; do patch -p1 < $p; done
PC $> popd
PC $> popd

7.2 TF-A : copy the DTS into the source code

PC $> pushd $WORKDIR
PC $> cp -r MyDeviceTree_fromCubeMX/tf-a/* tf-a/tf-a-sources/fdts/
PC $> popd

7.3 TF-A : regenerate TF-A.stm32

Information

The procedure below is an extract of the README.HOW_TO.txt file which is available in *$WORKDIR /sources/arm-ostl-linux-gnueabi/tf-a-stm32mp-**. Please notice a *grep "$>" README.HOW_TO.txt* describes the few commands needed to build the artifact of tf-a

PC $> pushd $WORKDIR/tf-a/tf-a-sources
PC $> make -f ../../sources/arm-ostl-linux-gnueabi/tf-a-stm32mp-[0-9]*-r0/Makefile.sdk TFA_DEVICETREE=*<device tree>* TF_A_CONFIG=*<config>* all

<config> : could be ***trusted*** or *serialboot* according the boot type
<device tree> : is the device tree just copied, i.e.: ***stm32mp157c-mydevicetree-mx***

PC $> popd

PC $> ls -l $WORKDIR/tf-a/build/*<config>*/tf-a-*<device tree>*-*<config>*.stm32

7.4 TF-A : copy the DTB in target/bootfs

Then flash the tf-a-*.stm32 into the 'fsbl1' and 'fsbl2' partitions of the target with STM32CubeProgrammer

Warning

'fsbl1' and 'fsbl2' are two redondant partitions and so, they should have same content

8 Update methods

8.1 Updating bootfs

There are two methods to update bootfs

- On an up and running target

  PC $> scp stm32mp157c-mydevicetree-mx.dtb root@<Target_IP>:/boot/

- Directly into 'bootfs' image

You do not need to have a target up and running. Only the "st-image-bootfs-openstlinux-weston-stm32mp1.ext4" file is required. To modify an 'ext4' file, a loopback mount, avaible within any Linux Distribution (even through WSL2), is required:

PC $> mkdir -p $WORKING/bootfs

PC $> mount -o loop <st-image-bootfs-openstlinux-weston-stm32mp1.ext4> $WORKING/bootfs

##Then copy the new dtb file at the root of $WORKING/bootfs

PC $> umount $WORKING/bootfs

PC $> sync

PC $> dd if=<st-image-bootfs-openstlinux-weston-stm32mp1.ext4> of=/dev/mmblk0p4 conv=fdatasync

Information

The '/dev/mmcblk0p4' is in case of the sdcard is inserted in dedicated drive of the PC, using an USB sdcard reader will probably create /dev/sdb4 entry.

8.2 Updating extlinux

8.2.1 extlinux basics

extlinux describes how u-boot boots. Updating **extlinux** consists in updating the extlinux.conf:

- In case of an DK-2 board booting from the sdcard. A **stm32mp157c-dk2_extlinux.conf** file is located in */boot/mmc0_extlinux/*,
- otherwise if there is no specific extlinux.conf for your board then the **extlinux.conf** is taking into account.

extlinux.conf is the description of a boot menu with one or several entries; 'DEFAULT' selects the default entry.
Below an example of **extlinux.conf**:

```
menu title Select the boot mode
MENU BACKGROUND ../splash.bmp
TIMEOUT 5
DEFAULT stm32mp157c-mydevicetree-mx
LABEL stm32mp157c-dk2-sdcard
        KERNEL /uImage
        FDTDIR /
        APPEND root=PARTUUID=e91c4e10-16e6-4c0e-bd0e-77becf4a3582 rootwait rw
console=ttySTM0,115200
LABEL stm32mp157c-dk2-a7-examples-sdcard
        KERNEL /uImage
        FDT /stm32mp157c-dk2-a7-examples.dtb
        APPEND root=PARTUUID=e91c4e10-16e6-4c0e-bd0e-77becf4a3582 rootwait rw
console=ttySTM0,115200
LABEL stm32mp157c-dk2-m4-examples-sdcard
        KERNEL /uImage
        FDT /stm32mp157c-dk2-m4-examples.dtb
        APPEND root=PARTUUID=e91c4e10-16e6-4c0e-bd0e-77becf4a3582 rootwait rw
console=ttySTM0,115200
LABEL stm32mp157c-mydevicetree-mx
       KERNEL /uImage
       FDT /stm32mp157c-mydevicetree-mx.dtb
       APPEND root=PARTUUID=e91c4e10-16e6-4c0e-bd0e-77becf4a3582 rootwait rw
console=ttySTM0,115200
```

Please update/add the highlighted lines according to what have been compiled in chapter 5, 6 and/or 7:

- DEFAULT: This is the default 'LABEL' to boot
- LABEL : The entry 'LABEL' is the value of 'compatible' of the DTS file compiled with u-boot.

  The 'compatible' value is at head of the DTS file and looks like : "st,stm32mp157c-mydevicetree-mx"
- FDT : The path from /boot of the kernel DTB to use

8.2.2 Updating extlinux

Updating 'extlinux' consists in modifying the extlinux.conf. There are two ways to do this:

- On an up and running target

Open an ssh connection to the target or use a direct connection with a tty terminal. Then use an vi editor to modify the extlinux.conf file.

Do not forget to synchronize the file system before rebooting the target:

Board $> sync

- Into 'bootfs' image directly

You do not need to have a target up and running. Only the "st-image-bootfs-openstlinux-weston-stm32mp1.ext4" file is required. To modify an 'ext4' file, a loopback mount tool, avaible in any Linux Distribution (even through WSL2), is needed:

```
PC $> mkdir -p $WORKING/bootfs
PC $> mount -o loop <st-image-bootfs-openstlinux-weston-stm32mp1.ext4> $WORKING/bootfs
##Then edit the extlinux.conf file (for WSL2 use a 'Linux' type editor; vi, ...)
##Once extlinux.conf up-to-date, umount loopback and flash the bootfs into sdcard with STM32CubeProgrammer
```

Device Tree Binary (or Blob)

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Trusted Firmware for Arm Cortex-A

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

Device Tree Source (in software context) or Digital Temperature Sensor (in peripheral context)

Stable: 17.11.2020 - 16:27 / Revision: 30.07.2020 - 08:18

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

When an OpenSTLinux distribution has been modified, it is pertinent to build a new software development package that integrates the modifications, and to redistribute this SDK to developers (see SDK development cycle model).

1 Prerequisites

The Distribution Package relative to your STM32 microprocessor Series is installed: Category:Distribution Package.

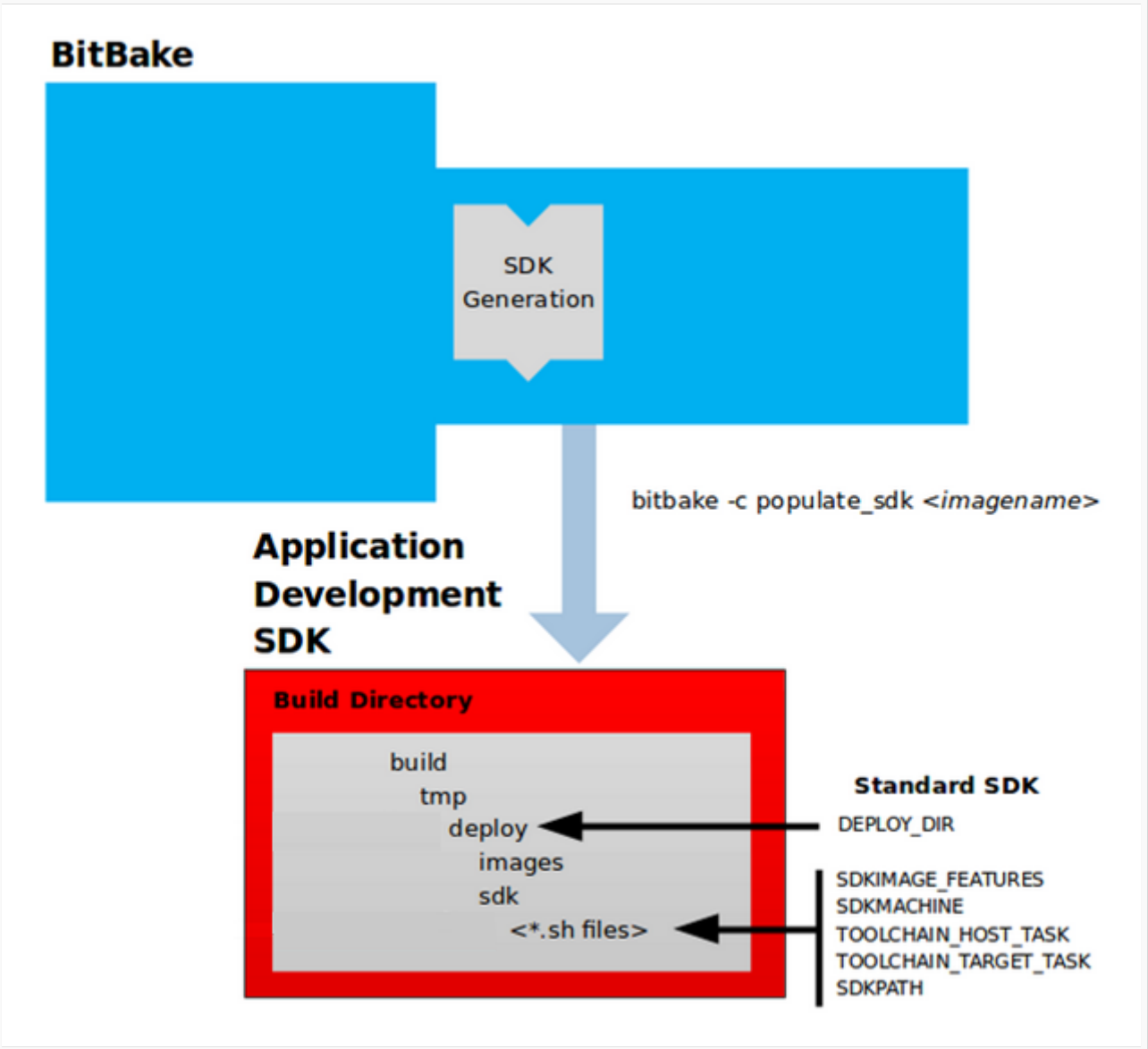
On the installation:

- some pieces of software might have been modified or integrated
- the build environment script has been executed
- the selected image has been rebuilt

2 SDK generation

The OpenEmbedded build system uses BitBake to generate the software development package (SDK) installation script.

For more information about the SDK, see the SDK for OpenSTLinux distribution article.



Standard SDK generation
Source: Yocto Project[1]

The *do_populate_sdk* task helps to create the standard SDK and handles two parts: a target part and a host part. The target part is built for the target hardware and includes libraries and headers. The host part is the part of the SDK that runs on the host machine.

- Check that the build environment script has been executed, and that the current directory is the build directory of the OpenSTLinux distribution (for example, *openstlinux-20-06-24/build-openstlinuxweston-stm32mp1')*
- Generate the SDK installation files (including the installation script) for a standard SDK with the following command :

```
PC $> bitbake -c populate_sdk <image>
```

Where:

<image>	Image name; example: • st-image-weston

Example:

PC $> bitbake -c populate_sdk st-image-weston

- The SDK installation files (*<image>-<distro>-<machine>-<host machine>-toolchain-<Yocto release>-snapshot.**) are written to the *deploy/sdk* directory inside the build directory *build-<distro>-<machine>* as shown in the figure above

Where:

<host machine>	Host machine on which the SDK is generated • x86_64 (64-bit host machine; this is the only supported value)
<Yocto release>	Release number of the Yocto Project; example: • 3.1 (aka dunfell)

Example

```
PC $> ls tmp-glibc/deploy/sdk/
st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-3.1-snapshot.host.manifest
st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-3.1-snapshot.license
st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-3.1-snapshot-license_content.
html
st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-3.1-snapshot.sh
st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-3.1-snapshot.target.manifest
st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-3.1-snapshot.testdata.json
```

The main final output is the cross-development toolchain installation script (*.sh* file), which includes the environment setup script.

Note that several OpenEmbedded variables exist that help configure these files. The following list shows the variables associated with a standard SDK:

```
    DEPLOY_DIR: points to the deploy directory.
    SDKMACHINE: specifies the architecture of the machine on which the cross-development
tools are run to create packages for the target hardware.
    SDKIMAGE_FEATURES: lists the features to include in the "target" part of the SDK.
    TOOLCHAIN_HOST_TASK: lists packages that make up the host part of the SDK (that is,
the part that runs on the SDKMACHINE). This variable allows packages other than the
default ones to be added.
    TOOLCHAIN_TARGET_TASK: lists packages that make up the target part of the SDK (that
is, the part built for the target hardware).
    SDKPATH: Defines the default SDK installation path offered by the installation script.
```

3 Reference list

- http://www.yoctoproject.org/documentation

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

also known as

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

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

Contents

1 Article purpose

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

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

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

Information

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

2 Prerequisites

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

The board and the host machine are connected through an Ethernet link, and a remote terminal program is started on the host machine: see How to get Terminal.

The target is started, and its IP address (<board ip address>) is known.

Information

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

Information

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

3 Modifying the Linux kernel configuration

3.1 Preamble

Warning

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

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

There are two possible situations:

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

Example:

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

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

3.2 Simple example

This simple example modifies the value defined for the contiguous memory area (CMA) size.

- Get the current value of the CMA size (128 Mbytes here) through the analysis of the target boot log

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

STM32MP157C-EV1

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

STM32MP157C-DK2

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

- Go to the <Linux kernel build directory>

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

- Start the Linux kernel configuration menu: see Menuconfig or how to configure kernel
- Navigate to "Device Drivers - Generic Driver Options"
  - select "Size in Megabytes"
  - modify its value to 256
  - exit and save the new configuration
- Check that the configuration file (*.config*) has been modified

```
PC $> grep -i CONFIG_CMA_SIZE_MBYTES .config
CONFIG_CMA_SIZE_MBYTES=256
```

- Cross-compile the Linux kernel: see Menuconfig or how to configure kernel
- Update the Linux kernel image on board: see Menuconfig or how to configure kernel
- Reboot the board: see Menuconfig or how to configure kernel
- Get the new value of the CMA size (256 Mbytes) through the analysis of the target boot log

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

STM32MP157C-EV1

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

STM32MP157C-DK2

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

4 Modifying the Linux kernel device tree

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

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

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

STM32P157C-EV1

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

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

STM32MP157C-DK2

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

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

- Go to the <Linux kernel build directory>

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

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

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

- Update the device tree blobs on the board

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

Information

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

- Reboot the board

```
Board $> reboot
```

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

5 Modifying a built-in Linux kernel device driver

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

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

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

- Go to the <Linux kernel source directory>

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

- Edit the *./drivers/gpu/drm/stm/drv.c* source file
- Add a log information in the *stm_drm_platform_probe* function

```
static int stm_drm_platform_probe(struct platform_device *pdev)
{
        struct device *dev = &pdev->dev;
        struct drm_device *ddev;
        int ret;
        [...]

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

        return 0;
        [...]
}
```

- Go to the <Linux kernel build directory>

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

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

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

- Update the Linux kernel image on board

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

Information

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

- Reboot the board

```
Board $> reboot
```

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

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

6 Modifying/adding an external Linux kernel module

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

6.1 Modifying an external in-tree Linux kernel module

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

- Go to the <Linux kernel source directory>

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

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

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

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

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

        return ret;
}
```

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

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

        return 0;
}
```

- Go to the <Linux kernel build directory>

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

- Cross-compile the Linux kernel modules

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

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

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

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

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

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

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

OR

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

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

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

- Insert the vivid kernel module into the Linux kernel

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

- Remove the vivid kernel module from the Linux kernel

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

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

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

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

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

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

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

Example:

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

- Create a directory for this kernel module example

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

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

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

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

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

static void __exit kernel_module_example_exit(void)
{
        printk(KERN_INFO "Kernel module example: goodbye from STMicroelectronics\n");
}

module_init(kernel_module_example_init);
```

```
module_exit(kernel_module_example_exit);

MODULE_DESCRIPTION("STMicroelectronics simple external out-of-tree Linux kernel module
example");
MODULE_AUTHOR("Jean-Christophe Trotin <jean-christophe.trotin@st.com>");
MODULE_LICENSE("GPL v2");
```

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

Information

All the indentations in a makefile are tabulations

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

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

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

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

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

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

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

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

- Cross-compile the kernel module example

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

- Go to the <Linux kernel build directory>

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

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

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

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

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

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

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

OR

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

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

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

- Insert the kernel module example into the Linux kernel

```
Board $> modprobe kernel_module_example
[18167.821725] Kernel module example: hello world from STMicroelectronics
```

- Remove the kernel module example from the Linux kernel

```
Board $> rmmod kernel_module_example
[18180.086722] Kernel module example: goodbye from STMicroelectronics
```

7 Modifying the U-Boot

This simple example adds unconditional log information when U-Boot starts. Within the scope of the trusted boot chain, U-Boot is used as second stage boot loader (SSBL).

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

```
Board $> reboot
```

STM32MP157C-EV1

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

STM32MP157C-DK2

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

- Go to the <U-Boot source directory>

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

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

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

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

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

```
        return 0;
}
```

- Get the list of supported configurations with the following command

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

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

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

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

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

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

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

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

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

Information

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

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

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

- Copy the U-Boot binary to the dedicated partition

STM32MP157C-EV1

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

STM32MP157C-DK2

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

- Reset the U-Boot shell

```
STM32MP> reset
```

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

STM32MP157C-EV1

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

STM32MP157C-DK2

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

8 Modifying the TF-A

This simple example adds unconditional log information when the TF-A starts. Within the scope of the trusted boot chain, TF-A is used as first stage boot loader (FSBL).

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

```
Board $> reboot
[...]
INFO:      System reset generated by MPU (MPSYSRST)
INFO:    PMIC version = 0x10
INFO:    Using SDMMC
[...]
```

- Go to the <TF-A source directory>

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

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

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

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

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

- Get the list of supported configurations with the following command

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

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

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

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

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

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

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

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

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

Information

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

- On the host machine, check the partition associated with the first stage boot loader (*fsbl1* and *fsbl2* as backup): *sdb1* and *sdb2* (as backup) here

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

- Copy the TF-A binary to the dedicated partition; to test the new TF-A binary, it might be useful to keep the old TF-A binary in the backup FSBL (*fsbl2*)

STM32MP157C-EV1

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

Information

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

STM32MP157C-DK2

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

- Reset the U-Boot shell

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

```
STM32MP> reset
```

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

```
[...]
INFO:      System reset generated by MPU (MPSYSRST)
INFO:    TF-A simple example
INFO:    PMIC version = 0x10
INFO:    Using SDMMC
[...]
```

9 Adding a "hello world" user space example

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

9.1 Source code file

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

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

- Create a directory for this user space example

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

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

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

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

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

        printf("\nUser space example: hello world from STMicroelectronics\n");
        setbuf(stdout,NULL);
        while (i--) {
                printf("%i ", i);
                sleep(1);
        }
        printf("\nUser space example: goodbye from STMicroelectronics\n");

        return(0);
}
```

9.2 Cross-compilation

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

9.2.1 Command line

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

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

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

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

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

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

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

9.2.2 Makefile-based project

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

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

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

Information

All the indentations in a makefile are tabulations

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

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

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

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

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

all: $(PROG)


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

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

- Cross-compile the project

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

9.2.3 Autotools-based project

This method creates a project based on GNU autotools.

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

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

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

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

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

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

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

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

- Generate the links towards SDK scripts

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

- Cross-compile the project

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

9.3 Deploy and execute on board

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

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

- Execute this user space example

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

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

10 Tips

10.1 Creating a mounting point

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

- Find the partition label associated with the boot file system

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

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

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

Linux® is a registered trademark of Linus Torvalds.

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

Trusted Firmware for Arm Cortex-A

Graphics Processing Units

Light-emitting diode

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Direct Rendering Manager (kernel module that gives direct hardware access to DRI clients, find more information on official DRI web site http://dri.freedesktop.org/wiki/DRM)

Second Stage Boot Loader

Central processing unit

USB On-The-Go (Capability/type of USB port, acting primarily as USB device, to also act as USB host. Also known as USB OTG.)

User-space Mode Setting

First Stage Boot Loader

Microprocessor Unit

Power Management Integrated Circuit

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

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

Contents

1 Article purpose

This article provides simple examples for the Distribution Package of the OpenSTLinux distribution, that illustrate the cross-compilation with the devtool and BitBake tools:

- modification with Linux® Kernel (configuration, device tree, driver, ...)
- modification of an external in-tree Linux Kernel module
- modification of U-Boot
- modification of TF-A
- addition of software

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

All the examples described on this page use devtool and/or bitbake from OpenEmbededdded, see Open Embedded - devtool for more information.

Information

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

2 Prerequisites

The prerequisites from Installing the OpenSTLinux distribution must be executed.

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.

3 Modification with kernel

3.1 Preamble

To start modification with a module, you need to initialize your Distribution Package environment.

```
PC $> cd <working directory path of distribution>
PC $> DISTRO=openstlinux-weston MACHINE=stm32mp1 source layers/meta-st/scripts/envsetup.sh
```

You are now in the build directory, identified by <build dir> in the following paragraphs.

Initialize devtool for kernel component:

```
PC $> devtool modify virtual/kernel
NOTE: Starting bitbake server...
NOTE: Creating workspace layer in /mnt/internal_storage/oetest/oe_openstlinux_rocko/build-
openstlinuxweston-stm32mp1/workspace
NOTE: Enabling workspace layer in bblayers.conf
Parsing recipes: 100%
|##################################################################################|
Time: 0:00:54
Parsing of 2401 .bb files complete (0 cached, 2401 parsed). 3282 targets, 88 skipped, 0
masked, 0 errors.
NOTE: Mapping virtual/kernel to linux-stm32mp
NOTE: Resolving any missing task queue dependencies
...
```

Information

For the case of virtual/<something> component, you need to get the name of mapping between virtual component and associated recipe. In this example, the name of kernel recipe is indicated in the trace, but you can also get it by calling **devtool status**

A minority of devtool command supports the virtual/<something> component, like devtool modify, it is why you need to get the recipe name associated to virtual/component.
In this example, the name of kernel recipe is indicated in the trace (*linux-stm32mp*)

Information

The source code of kernel is located on <build dir>/workspace/sources. You can change the path where the source code is extracted by customizing the devtool modify command

Information

For all the work around the kernel, we strongly encourage some usage for deploying the binaries on board

- Kernel image, device tree: use bitbake deploy command and scp (see #modifying kernel configuration))
- Kernel module: use devtool deploy-target by passing by a temporary directory (see #Modifying/adding an external Linux kernel module)

The difference of usage comes from the number of files to deploy on board. When there is only one or two files to put on board, the easiest way to do it is to deploy only the desired files .

3.2 Modifying kernel configuration

This simple example modifies the kernel configuration via menuconfig for the 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
[    0.000000] cma: Reserved 128 MiB at 0xf0000000
```

- 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 <build dir>/workspace/sources/<name of kernel
recipe>/.config.new
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
[    0.000000] cma: Reserved 256 MiB at 0xe0000000
```

3.3 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 LED (LD3) is disabled
- Go to the <build dir>/workspace/sources/<name of kernel recipe>/ directory

```
PC $> cd <build dir>/workspace/sources/<name of kernel recipe>/
```

- Edit the *arch/arm/boot/dts/stm32mp157c-ed1.dts* Device Tree Source file for evaluation board or

Edit the *arch/arm/boot/dts/stm32mp157c-dk2.dts* Device Tree Source file for discovery board or

- Change the status of the "stm32mp:green:user" led to "okay", and set its default state to "on"

```
        led {
                compatible = "gpio-leds";

                status = "okay";
```

```
                red {
                        label = "stm32mp:red:status";
                        gpios = <&gpioa 13 GPIO_ACTIVE_LOW>;

                        status = "disabled";
                };
                green {
                        label = "stm32mp:green:user";
                        gpios = <&gpioa 14 GPIO_ACTIVE_LOW>;
                        default-state = "on";

                        status = "okay";
                };
        };
```

- Go to the build directory

```
PC $> cd <build dir>
```

If this update is accepted, then to report in other paragraph of this page}}

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

```
PC $> bitbake virtual/kernel -C compile
```

Information

we use here bitbake command instead of **devtool build** because the **build** makes **compile**, **compile_kernemodules** and **install** commands whereas we only need only **compile** command to generate the kernel image and the device tree

- Update the device tree blobs on the board

```
PC $> scp <build dir>/tmp-glibc/deploy/images/<machine name>/*.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 $> cd /boot; sync; systemctl reboot
```

- Check that the user LED (LD3) is **enabled** (green)

3.4 Modifying a built-in Linux kernel device driver

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

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

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

- Go to the <build dir>/workspace/sources/<name of kernel recipe>/

```
PC $> cd <build dir>/workspace/sources/<name of kernel recipe>/
```

- 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 build directory

```
PC $> cd <build dir>
```

- Cross-compile the Linux kernel

```
PC $> bitbake virtual/kernel -C compile
```

- Update the Linux kernel image on board

```
PC $> scp <build dir>/tmp-glibc/deploy/images/<machine name>/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 $> cd /boot; sync; systemctl reboot
```

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

```
Board $> dmesg | grep -i stm_drm_platform_probe
[    5.005833] [drm] Simple example - stm_drm_platform_probe
```

3.5 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).

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

- Go to the <build dir>/workspace/sources/<name of kernel recipe>/

```
PC $> cd <build dir>/workspace/sources/<name of kernel recipe>/
```

- 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 build directory

```
PC $> cd <build dir>
```

- Cross-compile the Linux kernel modules

```
PC $> bitbake virtual/kernel -C compile
```

- Update the vivid kernel module on the board

```
PC $> devtool deploy-target -Ss <name of kernel recipe> root@<board ip address>:/
```

- 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
```

4 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.

- 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 ?= <Linux kernel path>

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

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

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

- Add a new recipe to the workspace

```
PC $> cd <build dir>
PC $> devtool add mymodule kernel_module_example/
```

- Adapt recipe to kernel module build

```
PC $> devtool edit-recipe mymodule
```

Modify the recipe according the following changes (see highlighted lines)

```
 1 # Recipe created by recipetool
 2 # This is the basis of a recipe and may need further editing in order to be fully
functional.
 3 # (Feel free to remove these comments when editing.)
 4
 5 # Unable to find any files that looked like license statements. Check the accompanying
 6 # documentation and source headers and set LICENSE and LIC_FILES_CHKSUM accordingly.
 7 #
 8 # NOTE: LICENSE is being set to "CLOSED" to allow you to at least start building - if
 9 # this is not accurate with respect to the licensing of the software being built (it
10 # will not be in most cases) you must specify the correct value before using this
11 # recipe for anything other than initial testing/development!
12 LICENSE = "CLOSED"
13 LIC_FILES_CHKSUM = ""
14
15 # No information for SRC_URI yet (only an external source tree was specified)
16 SRC_URI = ""
17
18 # NOTE: this is a Makefile-only piece of software, so we cannot generate much of the
19 # recipe automatically - you will need to examine the Makefile yourself and ensure
20 # that the appropriate arguments are passed in.
21 DEPENDS = "virtual/kernel"
22 inherit module
23
24 EXTRA_OEMAKE  = "ARCH=arm"
25 EXTRA_OEMAKE += "KERNEL_DIR=${STAGING_KERNEL_BUILDDIR}"
26
27 S = "${WORKDIR}"
28
29 do_configure () {
30         # Specify any needed configure commands here
31         :
32 }
33
34 do_compile () {
35          # You will almost certainly need to add additional arguments here
36         oe_runmake
37 }
```

```
38
39 do_install () {
40         # NOTE: unable to determine what to put here - there is a Makefile but no
41         # target named "install", so you will need to define this yourself
42         install -d ${D}/lib/modules/${KERNEL_VERSION}
43         install -m 0755 ${B}/kernel_module_example.ko ${D}/lib/modules/${KERNEL_VERSION
}/
44 }
```

- Go to the build directory

```
PC $> cd <build dir>
```

- Generated kernel module example

```
PC $> devtool build mymodule
```

- Push this kernel module example on board

```
PC $> devtool deploy-target -Ss mymodule root@<board ip address>
```

- 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
```

5 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
[...]
U-Boot <U-Boot version>

CPU: STM32MP1 rev1.0
Model: STMicroelectronics STM32MP157C [...]
Board: stm32mp1 in trusted mode
[...]
```

- Go to the build directory

```
PC $> cd <build dir>
```

- Search U-boot recipe

```
PC $> devtool search u-boot*
u-boot-stm32mp-extlinux  Generate 'extlinux.conf' file for U-boot
u-boot-stm32mp        Universal Boot Loader for embedded devices for stm32mp
```

On this example, the recipe name is **u-boot-stm32mp**

- Start to work with u-boot

```
PC $> devtool modify u-boot-stm32mp
```

```
Example:
PC $> cd <build dir>/workspace/sources/u-boot-stm32mp
```

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

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

        [...]

        printf("Board: stm32mp1 in %s mode\n", mode);
        printf("U-Boot simple example\n");

        return 0;
}
```

- Cross-compile the U-Boot: trusted boot

```
PC $> devtool build u-boot-stm32mp
PC $> bitbake u-boot-stm32mp -c deploy
```

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

```
PC $> cd <build dir>/tmp-glibc/deploy/images/<machine name>/
```

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

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

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

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

Information

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

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

```
PC $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 10 Jan 17 18:05 bootfs -> ../../sdc4
lrwxrwxrwx 1 root root 10 Jan 17 18:05 fsbl1 -> ../../sdc1
lrwxrwxrwx 1 root root 10 Jan 17 18:05 fsbl2 -> ../../sdc2
lrwxrwxrwx 1 root root 10 Jan 17 18:05 rootfs -> ../../sdc5
lrwxrwxrwx 1 root root 10 Jan 17 18:05 ssbl -> ../../sdc3
lrwxrwxrwx 1 root root 10 Jan 17 18:05 userfs -> ../../sdc6
```

- Copy the binary (u-boot.stm32) to the dedicated partition

```
PC $> dd if=u-boot-<board name>-trusted.stm32 of=/dev/sdc3 bs=1M conv=fdatasync
```

(here u-boot-stm32mp157c-ev1-trusted.stm32 or u-boot-stm32mp157c-dk2-trusted.stm32)

- Reset the U-Boot shell

```
STM32MP> reset
```

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

```
[...]
U-Boot <U-Boot version>

CPU: STM32MP1 rev1.0
Model: STMicroelectronics STM32MP157C [...]
Board: stm32mp1 in trusted mode
U-Boot simple example
[...]
```

6 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 build directory

```
PC $> cd <build dir>
```

- Search TF-A recipe

```
PC $> devtool search tf-a*
babeltrace            Babeltrace - Trace Format Babel Tower
libunistring          Library for manipulating C and Unicode strings
lttng-tools           Linux Trace Toolkit Control
gettext               Utilities and libraries for producing multi-lingual messages
glibc                 GLIBC (GNU C Library)
tf-a-stm32mp          Trusted Firmware-A for STM32MP1
gnutls                GNU Transport Layer Security Library
gstreamer1.0          GStreamer 1.0 multimedia framework
harfbuzz              Text shaping library
glibc-locale          Locale data from glibc
kbd                   Keytable files and keyboard utilities
```

On this example, the recipe name is **tf-a-stm32mp**

- Start to work with tf-a

```
PC $> devtool modify tf-a-stm32mp
```

- Go to <build dir>/workspace/sources/tf-a-stm32mp

```
PC $> cd <build dir>/workspace/sources/tf-a-stm32mp
```

- Edit the *./plat/st/stm32mp1/bl2_io_storage.c* source file
- Add a log information in the *stm32mp1_io_setup* function

```
void stm32mp1_io_setup(void)
{
        int io_result;
        [...]
```

```
        /* Add a trace about reset reason */
        print_reset_reason();

        INFO("TF-A simple example");

        [...]
}
```

- Cross-compile the TF-A

```
PC $> devtool build tf-a-stm32mp
PC $> bitbake tf-a-stm32mp -c deploy
```

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

```
PC $> cd <build dir>/tmp-glibc/deploy/images/<machine name>/
```

- 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): *sdc1* and *sdc2* (as backup) here

```
PC $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 10 Jan 17 18:05 bootfs -> ../../sdc4
lrwxrwxrwx 1 root root 10 Jan 17 18:05 sfsbl1 -> ../../sdc1
lrwxrwxrwx 1 root root 10 Jan 17 18:05 sfsbl2 -> ../../sdc2
lrwxrwxrwx 1 root root 10 Jan 17 18:05 rootfs -> ../../sdc5
lrwxrwxrwx 1 root root 10 Jan 17 18:05 ssbl -> ../../sdc3
lrwxrwxrwx 1 root root 10 Jan 17 18:05 userfs -> ../../sdc6
```

- Copy the binary (tf-a-stm32mp157c-ev1-trusted.stm32) 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*)

```
PC $> dd if=tf-a-<board name>-trusted.stm32 of=/dev/sdc1 bs=1M conv=fdatasync
```

(here tf-a-stm32mp157c-ev1-trusted.stm32 or tf-a-stm32mp157c-dk2-trusted.stm32)

- Reset the U-Boot shell

```
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
[...]
```

7 Adding a "hello world" user space example

This chapter shows how to compile and execute a simple "hello world" example.

- 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);
}
```

- Add a new recipe to the workspace

```
PC $> cd <build dir>
PC $> devtool add myhelloword hello_world_example/
```

- Adapt recipe

```
PC $> devtool edit-recipe myhelloword
```

Modify the recipe according the following changes (see highlighted lines)

```
 1 # Recipe created by recipetool
 2 # This is the basis of a recipe and may need further editing in order to be fully
functional.
 3 # (Feel free to remove these comments when editing.)
 4
```

```
 5 # Unable to find any files that looked like license statements. Check the accompanying
 6 # documentation and source headers and set LICENSE and LIC_FILES_CHKSUM accordingly.
 7 #
 8 # NOTE: LICENSE is being set to "CLOSED" to allow you to at least start building - if
 9 # this is not accurate with respect to the licensing of the software being built (it
10 # will not be in most cases) you must specify the correct value before using this
11 # recipe for anything other than initial testing/development!
12 LICENSE = "CLOSED"
13 LIC_FILES_CHKSUM = ""
14
15 # No information for SRC_URI yet (only an external source tree was specified)
16 SRC_URI = ""
17
18 # NOTE: no Makefile found, unable to determine what needs to be done
19
20 do_configure () {
21         # Specify any needed configure commands here
22         :
23 }
24
25 do_compile () {
26         # Specify compilation commands here
27         cd ${S}
28         ${CC} hello_world_example.c -o hello_word_example
29 }
30
31 do_install () {
32         # Specify install commands here
33         install -d ${D}${bindir}
34         install -m 755 ${S}/hello_word_example  ${D}${bindir}/
35 }
```

- Compile binary

```
PC $> devtool build myhelloword
```

- Push this binary on board

```
PC $> devtool deploy-target -s myhelloword root@<board ip address>
```

- Execute this user space example

```
Board $> /usr/bin/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
```

8 Tips

8.1 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 Dec 13 12:31 bootfs -> ../../mmcblk0p4
lrwxrwxrwx 1 root root 15 Dec 13 12:31 fsbl1 -> ../../mmcblk0p1
lrwxrwxrwx 1 root root 15 Dec 13 12:31 fsbl2 -> ../../mmcblk0p2
lrwxrwxrwx 1 root root 15 Dec 13 12:31 rootfs -> ../../mmcblk0p5
lrwxrwxrwx 1 root root 15 Dec 13 12:31 ssbl -> ../../mmcblk0p3
lrwxrwxrwx 1 root root 15 Dec 13 12:31 userfs -> ../../mmcblk0p6
```

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

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

Linux® is a registered trademark of Linus Torvalds.

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

Trusted Firmware for Arm Cortex-A

Light-emitting diode

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Direct Rendering Manager (kernel module that gives direct hardware access to DRI clients, find more information on official DRI web site http://dri.freedesktop.org/wiki/DRM)

Second Stage Boot Loader

Central processing unit

USB On-The-Go (Capability/type of USB port, acting primarily as USB device, to also act as USB host. Also known as USB OTG.)

User-space Mode Setting

First Stage Boot Loader

Microprocessor Unit

Stable: 19.10.2020 - 14:26 / Revision: 19.10.2020 - 14:26

A quality version of this page, approved on *19 October 2020*, was based off this revision.

Contents

1 Article purpose

This article provides guidelines to install a software package containing binaries (e.g. tools) not present by default in the OpenSTLinux distribution.
This installation can be done either by using a Yocto recipe in the scope of a STM32MPU Embedded Software Distribution Package or either by cloning an existing tarball or repo git in the scope of a STM32MPU Embedded Software Developer Package.

2 Using the STM32MPU Embedded Software Developer Package

Following basic steps to be done :

- Downloading application source code via git clone (tarball or repo git)
- Compiling the application (very often with basic **make** command)
  - Please ensure SDK environment for compiling Linux application setup is ready
  - Refer to *<Linux kernel installation directory>/README.HOW_TO.txt* helper file to know how to compile (the latest version of this helper file is also available in GitHub: README.HOW_TO.txt).
- Installing the application (very often with basic **make install** command)
  - Refer to application README file to know how to install
- Deploying the application on board

Check where the generated binaries/libraries have been installed and push them onto the board.

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

3 Using the STM32MPU Embedded Software Distribution Package

Following steps to be done:

- Identifying the recipe building and installing the application/package you need

  - More often search this recipe either in layer openembedded-core/meta/ either in layers meta-openembedded/meta-*/

  - If no existing recipe, you have to create and copy it in your own customer layer: refer to "Writing a New Recipe"[1] and How to create your own distribution

  - Check if the layer containing the recipe is currently in use and so can be compiled, if not you have this error :

```
PC $> bitbake <recipe name>
ERROR: Nothing PROVIDES '<recipe name>. Close matches:
```

- Checking if the layer containing the recipe is currently in use

The command to show the layers currently in your build:

```
PC $> bitbake-layers show-layers
```

It outputs a list of the layers currently in use, and their priorities. If a package exists in two or more layers, it will be built from the layer with the higher priority.

If the recipe is not contained in the list of layers, you must add the layer containing this recipe inside the bbalyers.conf.sample of the distro you have selected.

For instance for distro openstlinux, bblayers file is located here :

```
 * meta-st/meta-st-openstlinux/conf/template/bblayers.conf.sample
```

So for instance, to add meta-qt5 layer, just copy/paste this line in bblayers file :

```
 * ADDONSLAYERS += "${@'${OEROOT}/meta-qt5' if os.path.isfile('${OEROOT}/meta-qt5/conf
/layer.conf') else ''}"
```

- Installing the package for the targetted image, for instance :

```
PC $> echo 'IMAGE_INSTALL_append += "<recipe name>"' >> meta-st/meta-st-openstlinux
/recipes-st/images/st-image-weston.bbappend
```

- Rebuilding the image:

```
PC $> bitbake st-image-weston
```

- Updating your board with new image binaries. See Flashing the built image.

4 References

- Writing a New Recipe