CMSIS-SVD environment and scripts

Contents

Stable: 21.02.2020 - 08:34 / Revision: 10.02.2020 - 11:04

Stable: 21.02.2020 - 08:54 / Revision: 10.02.2020 - 11:04

A quality version of this page, approved on *21 February 2020*, was based off this revision.

Contents

1 Article purpose

This article describes how to read or write to the STM32MPU internal peripherals registers using CMSIS-SVD file through GDB.

2 Information about CMSIS-SVD files

Extracted from Arm® Keil® web page[1]:

> *The CMSIS System View Description format(CMSIS-SVD) formalizes the description of the system contained in Arm Cortex-M processor-based microcontrollers, in particular, the memory mapped registers of peripherals. The detail contained in system view descriptions is comparable to the data in device reference manuals. The information ranges from high level functional descriptions of a peripheral all the way down to the definition and purpose of an individual bit field in a memory mapped register.*

Arm® also specifies on an other web page that CMSIS-SVD[2] is not restricted to only for Cortex®-M:

> ***For every supported microcontroller**, debuggers can provide detailed views to the device peripherals that display the current register state.*

3 Python scripts environment for GDB

Below information is related to the Android™ distribution
Not yet applicable for Android

3.1 CMSIS-SVD file parser

A CMSIS-SVD file parser is proposed by Paul Osborne and shared is on github: https://github.com/posborne/cmsis-svd.

This package is reused as is. Add STM32MPU microprocessor if it is missing from the CMSIS-SVD files.

For licensing, following information is provided:
In general, the following rules apply:

- *Under data, the license from each Vendor is provided along with the SVDs from that vendor. Please review this license before use of the SVDs contained therein. Look for files named the following for license information.*
- *All other code is licensed under the terms of the Apache License v2.0 (See LICENSE-APACHE).*

In our case, we will only add new files in data/STMicro sub-directory, so STMicroelectronics license is applied.

This package is part of the OpenSTLinux Developer Package.

3.2 Python scripts for GDB

svd-tools[3] github repository provides a gdb-svd python module for gdb which adds some instructions to:

- get information on peripherals | registers | fields
- get registers values | register fields
- set registers values | register fields

For more information, refer to the README[4] file of the repository.

This package is part of the OpenSTLinux Developer Package.

3.3 Setting environment

As pre-requisites, the gdb setup must be installed and running.

As CMSIS-SVD scripts are also part of the OpenSTLinux Developer Package, the environment is set by using the instructions below :

```
# Get the sysroot path from your SDK
(gdb) show environment OECORE_NATIVE_SYSROOT
 OECORE_NATIVE_SYSROOT = <path_to_sysroot>
(gdb) cd <path_to_sysroot>
(gdb) source usr/share/svd-tools/gdb-svd.py
```

4 Python scripts usage for GDB

4.1 Load svd file for the requested SoC

The **svd** instruction loads the appropriate SOC svd files.

Here is an example for the STM32MP15xxx:

```
(gdb) svd usr/share/cmsis-svd/cmsis_svd/data/STMicro/STM32MP15xxx.svd
```

Note: for all svd instructions described below, instruction completion for register name is working.

4.2 Getting peripheral / register / field information

This instruction displays any peripheral, register, and field information.
The last parameter can be part of the name, a filter is applied with the string.

```
(gdb) svd info <peripheral> [register] [field]
```

Please refer to the following examples:

- Getting information on all available ADC peripherals

(gdb) svd info ADC

```
+Peripherals-+------------+--------+-----------------------------+
| name       | base       | access | description                 |
+------------+------------+--------+-----------------------------+
| ADC2       | 0x48003100 | None   | ADC2                        |
| ADC        | 0x48003000 | None   | ADC                         |
| ADC_common | 0x48003300 | None   | Analog-to-Digital Converter |
+------------+------------+--------+-----------------------------+
```

- Getting information on all ADC2 registers beginning by S

```
(gdb) svd info ADC2 ADC_S
+Registers--+------------+------------+---------------------------------+
| name      | address    | access     | description                     |
+-----------+------------+------------+---------------------------------+
| ADC_SMPR1 | 0x48003114 | read-write | ADC sample time register 1      |
| ADC_SMPR2 | 0x48003118 | read-write | ADC sample time register 2      |
| ADC_SQR1  | 0x48003130 | read-write | ADC regular sequence register 1 |
| ADC_SQR2  | 0x48003134 | read-write | ADC regular sequence register 2 |
| ADC_SQR3  | 0x48003138 | read-write | ADC regular sequence register 3 |
| ADC_SQR4  | 0x4800313c | read-write | ADC regular sequence register 4 |
+-----------+------------+------------+---------------------------------+
```

- Getting information on all fields (peripheral: ADC2 register: CR) beginning by J

```
(gdb) svd info ADC2 ADC_CR J
+Fields----+----------+--------+-------------+
| name     | [msb:lsb] | access | description |
+----------+----------+--------+-------------+
| JADSTART | [3:3]     | None   | JADSTART    |
| JADSTP   | [5:5]     | None   | JADSTP      |
+----------+----------+--------+-------------+
```

4.3 Getting register(s) value(s)

This instruction provides the register list and field values.

```
(gdb) svd get [peripheral] [register]
```

Please refer to the following examples:

- Getting all the information for an entire peripheral

```
(gdb) svd get DLYBSD1
+Registers---+------------+------------
+-----------------------------------------------------------+
| name       | address    | value      |
fields                                                      |
+------------+------------+------------
+-----------------------------------------------------------+
| DLYB_CR    | 0x58006000 | 0x00000000 | DEN[0:0]=0x0 SEN[1:1]
=0x0                                   |
| DLYB_CFGR  | 0x58006004 | 0x80000000 | SEL[3:0]=0x0 UNIT[14:8]=0x0 LNG[27:16]=0x0 LNGF
[31:31]=0x1 |
| DLYB_VERR  | 0x580063f4 | 0x00000000 | MINREV[3:0]=0x0 MAJREV[7:4]
=0x0                              |
| DLYB_IPIDR | 0x580063f8 | 0x00000000 | ID[31:0]
=0x0                                                |
| DLYB_SIDR  | 0x580063fc | 0x00000000 | SID[31:0]
=0x0                                               |
+------------+------------+------------
+-----------------------------------------------------------+
```

- Getting information for just one register

```
(gdb) svd get DLYBSD1 DLYB_CFGR
+Registers--+-----------+-----------
+-----------------------------------------------------------+
| name      | address    | value      |
fields                                                      |
+-----------+------------+------------
+-----------------------------------------------------------+
| DLYB_CFGR | 0x58008004 | 0x00000000 | SEL[3:0]=0x0 UNIT[14:8]=0x0 LNG[27:16]=0x0 LNGF
[31:31]=0x0 |
+-----------+------------+------------
+-----------------------------------------------------------+
```

4.4 Writing register value

This instruction sets the register value.

```
(gdb) svd set <peripheral> <register> [field] <value>
```

Please refer to the following examples.

- Set register values

```
(gdb) svd set QUADSPI QUADSPI_LPTR 0x50
```

- Set field values

```
(gdb) svd set QUADSPI QUADSPI_LPTR TIMEOUT 0x10
```

5 References

- http://www.keil.com/pack/doc/CMSIS/SVD/html/index.html
- https://developer.arm.com/embedded/cmsis
- https://github.com/1udo6arre/svd-tools
- https://github.com/1udo6arre/svd-tools/blob/master/README.md

Stable: 17.11.2021 - 16:07 / Revision: 05.11.2021 - 11:37

A quality version of this page, approved on *17 November 2021*, was based off this revision.

Contents

1 Article purpose

This article provides the basic information needed to start using the **GDB**[1] application tool.

It explains how to use this GNU debugger tool connected to your ST board target via Ethernet or via ST-LINK, and how to perform cross-debugging (IDE, gdb viewer tool or command line) on Arm® Cortex®-A7 side for Linux® application, Linux® kernel (including external modules), or Arm® Cortext®-M4 firmware.

2 Introduction

The following table provides a brief description of the tool, as well as its availability depending on the software packages:

✓: this tool is either present (ready to use or to be activated), or can be integrated and activated on the software package.

✗: this tool is not present and cannot be integrated, or it is present but cannot be activated on the software package.

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	**Category**	**Purpose**	**Starter Package**	**Developer Package**	**Distribution Package**	**Starter Package**	**Developer Package**	**Distribution Package**
GDB	Debugging tools	The GNU Project debugger, **GDB**[1], allows monitoring program execution, or what the program was doing at the moment it crashed.	✗*	✓	✗**	✗	✗	✗
			** Cross compile gdb and openocd binaries are required and only available from Developer Package.* *** It is recommended to use the Developer Package to run the gdb debug session, which provided all dependencies*			**Warning** The STM32MPU distribution for Android™ is not yet available in the v3 ecosystem releases: please refer to the GDB page for the v2 ecosystem releases (in archived wiki).		

The GDB can perform four main types of actions (plus other corollary actions) to help you detect bugs when running your software or application:

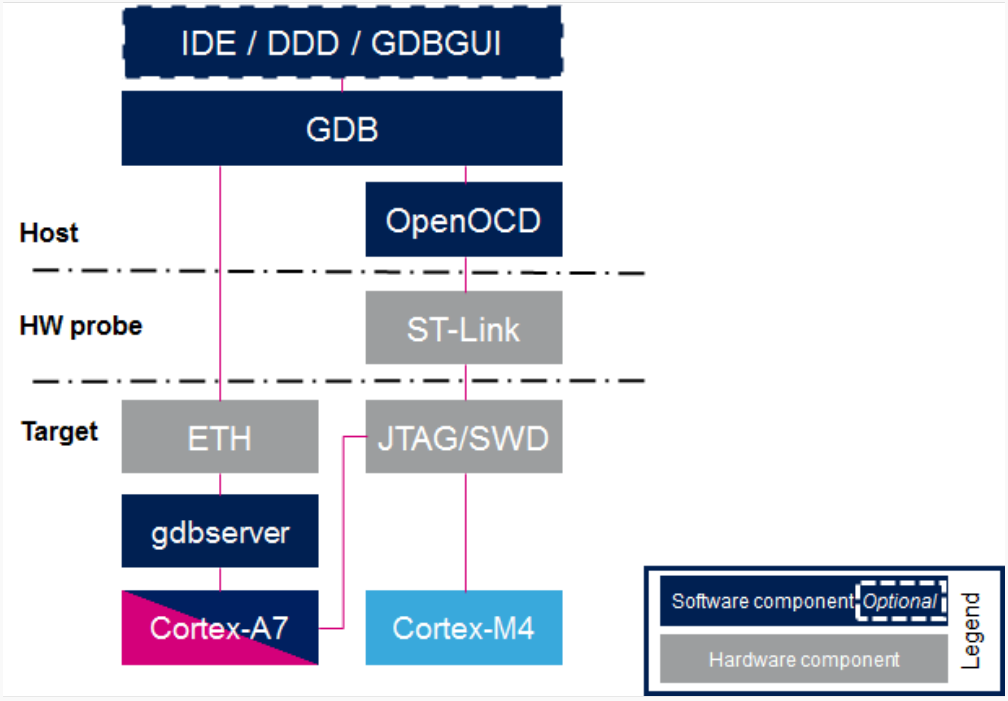
- Start the program, specifying anything that might affect its behaviour.
- Make the program stop on specific conditions.
- Examine what happened when the program stopped.
- Update the program in order to test a bug correction, and jump to the next one.

3 Overview of GDB setup for STM32MPU

3.1 GDB setup paths

Two paths can be used in the STM32MPU environment for GDB setup:

- **gdb gdbserver** path through Ethernet, **used for Cortex-A7 Linux applications**.
  In that case, two software components are required, one on the target and the other on the host PC.
- **gdb JTAG/SWD** path through OpenOCD and ST-LINK, **used both for Cortex-M4 firmware and Cortex-A7 Linux kernel**.
  In that case, only one software component is required on the host PC.



GDB paths scheme

Two components are included in OpenSTLinux Developer Package for GDB setup:

- **gdbserver**: **embedded on target rootfs** and used as remote access for a host connection
- **arm-ostl-linux-gnueabi-gdb**: **embedded on host PC side**, cross-compiled gdb binary that manages the connexion between the host computer and the target board

3.2 JTAG and SWD debug port

The STM32MPU features two debug ports through the embedded CoreSight™ component that implements an external access port for connecting debugging equipment:

- A 5-pin standard JTAG interface (JTAG-DP)
- A 2-pin (clock + data) “serial-wire debug” port (SW-DP)

These two modes are mutually exclusive since they share the same I/O pins.

Refer to STM32MP15 reference manuals for information related to JTAG and SWD.

4 Installing the GDB tool

This tool is made of two parts, one on the host PC, and a second on the target (only for debugging Linux applications).

4.1 Installing the GDB tool on your host PC

Below is the list of required components:

- The cross-compiled GDB binary
- The OpenOCD binary and configuration files
- The symbols files of all BSP components (TF-A, U-Boot and Linux kernel), corresponding to the images of the OpenSTLinux Starter Package.

4.1.1 Using the STM32MPU Embedded Software distribution

4.1.1.1 Developer Package

Only the Developer Package can be used, since it is the only one which provides all the required components.

4.2 Installing the GDB on your target board

On the target board, only the gdbserver binary is required for debugging Linux applications.

It is available by default within the Starter Package, which provides images linked to the Developer Package.

5 Getting started

This chapter describes the two ways for debugging OpenSTLinux BSP components (TF-A, U-Boot and Linux kernel including external modules), Linux applications and Cortex-M4 firmware: by using GDB commands or by using a GDB user interface such as gdbgui, DDD or IDE.

5.1 Prerequisites

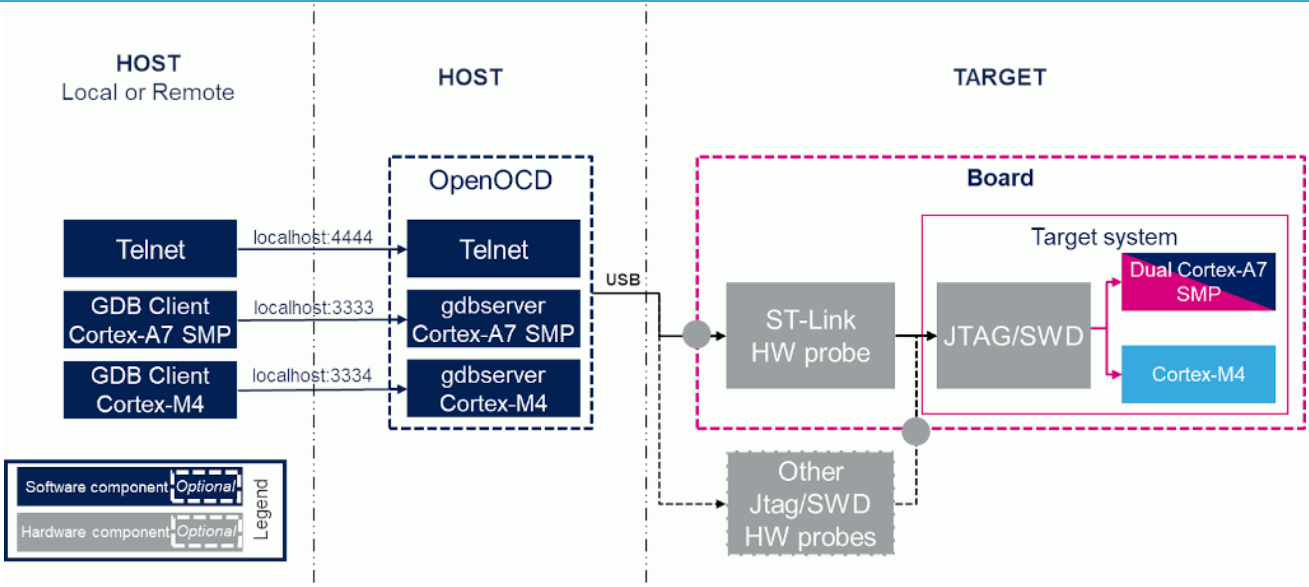
The target board is up and running.

5.2 Debug OpenSTLinux BSP components

5.2.1 Setting up GDB / OpenOCD debug path environment

- **Architecture**

The figure below shows the architecture corresponding to the GDB/OpenOCD connected to Cortex-A7 and Cortex-M4 cores.
Note: The ST-LINK probes available on the STM32MP1 board can be used through a USB interface, as well as any other external probes through the Trace or JTag connector.



- **Prerequisites**

  The Developer Package must be installed. It provides the SDK, the debug symbol files and the source files for TF-A, U-Boot and Linux kernel (refer to STM32MP1 Developer Package).

  The debug symbol files contain the symbols for the TF-A, U-Boot and Linux kernel binaries (from the Starter Package image) that have been flashed on the board.

- **Environment configuration files**

  To speed up environment setup for debugging with GDB, download two configuration files, and install them on your PC (*under the home directory: $HOME/gdbscripts/*). You can then **customize** them:

  - Setup.gdb: main configuration file in which you can define the debug context you want to use (Refer to Boot chain overview for details). Possible combinations are:

    - Trusted boot chain:

Debug mode	**(1) Cortex-A7 TF-A(BL2)**	**(2) Cortex-A7 TF-A(BL32)**	**(3) Cortex-A7 SSBL(U-Boot)**	**(4) Cortext-A7 Linux kernel**
(0) - Boot	✓	✓	✓	✓
(				

Debug mode	(1) Cortex-A7 TF-A(BL2)	(2) Cortex-A7 TF-A(BL32)	(3) Cortex-A7 SSBL(U-Boot)	(4) Cortext-A7 Linux kernel
1)-Running target	✗	✗	✓	✓

- Trusted boot chain with OP-TEE:

Debug mode	(1) Cortex-A7 TF-A(BL2)	(2) Cortex-A7 OP-TEE	(3) Cortex-A7 SSBL(U-Boot)	(4) Cortext-A7 Linux kernel
(0)-Boot	✓	✓	✓	✓
(1				

Debug mode	(1) Cortex-A7 TF-A(BL2)	(2) Cortex-A7 OP-TEE	(3) Cortex-A7 SSBL(U-Boot)	(4) Cortext-A7 Linux kernel
) - Running target	⊗	⊗	✓	✓

- File:Path env py.txt: customization file to define all the paths to source and symbol files, which can either be directly retrieved from the **Developer Package** (refer to the Example of directory structure for Packages), or result from a new compilation. **In both cases, update the paths corresponding to your environment.**
Please read carefully the comments provided in this file to help you updating source and symbol paths.

Store these files locally on your host PC, **check for name cast (Setup.gdb and Path_env.py)** and update them accordingly.

5.2.2 Configuring GDB and OpenOCD for attachment on a running target

When the target board is running, you can attach the GDB only during one of following phases:

5.2.2.1 U-Boot execution phase

Select the right configuration in Setup.gdb:

```
# Set debug phase:
#        1: Attach at Cortex-A7 / TF-A(BL2)
#        2: Attach at Cortex-A7 / TF-A(BL32) or OP-TEE
#        3: Attach at Cortex-A7 / U-Boot
#        4: Attach at Cortex-A7 / Linux kernel
set $debug_phase = 3
```

```
#        0: Attach at boot
#        1: Attach running target
set $debug_mode = 1
```

```
# Set debug trusted bootchain:
#        0: TF-A BL2 / TF-A BL32 / U-Boot / Linux kernel
#        1: TF-A BL2 / OP-TEE / U-Boot / Linux kernel
set $debug_trusted_bootchain = 0 or 1 #depending on your software boot chain configuration
```

When the configuration is complete, jump to Running OpenOCD and GDB.

5.2.2.2 *Linux kernel execution phase*

Select the right configuration in Setup.gdb:

```
# Set debug phase:
#        1: Attach at Cortex-A7 / TF-A(BL2)
#        2: Attach at Cortex-A7 / TF-A(BL32) or OP-TEE
#        3: Attach at Cortex-A7 / U-Boot
#        4: Attach at Cortex-A7 / Linux kernel
set $debug_phase = 4
```

```
#        0: Attach at boot
#        1: Attach running target
set $debug_mode = 1
```

```
# Set debug trusted bootchain:
#        0: TF-A BL2 / TF-A BL32 / U-Boot / Linux kernel
#        1: TF-A BL2 / OP-TEE / U-Boot / Linux kernel
set $debug_trusted_bootchain = 0 or 1 #depending on your software boot chain configuration
```

When the configuration is complete, jump to Running OpenOCD and GDB.

5.2.3 Configuring GDB and OpenOCD for attachment on boot

You can attach the GDB during target boot only in the following cases:

- TF-A(BL2) boot case;
- TF-A(BL32) boot case;
- OP-TEE boot case;
- U-Boot boot case;
- Linux kernel boot case.

To handle the cases above, the FSBL image has to be wrapped through the tool stm32wrapper4dbg. This operation allows the debugger to halt the target at the very first instruction of FSBL.

Warning

In these cases, the target board will automatically reboots when the GDB starts.

5.2.3.1 *TF-A(BL2) boot case*

In that case, the GDB breaks in bl2_entrypoint function.

Select the right configuration in Setup.gdb:

```
# Set debug phase:
#        1: Attach at Cortex-A7 / TF-A(BL2)
#        2: Attach at Cortex-A7 / TF-A(BL32) or OP-TEE
#        3: Attach at Cortex-A7 / U-Boot
#        4: Attach at Cortex-A7 / Linux kernel
set $debug_phase = 1
```

```
#        0: Attach at boot
#        1: Attach running target
set $debug_mode = 0
```

```
# Set debug trusted bootchain:
#        0: TF-A BL2 / TF-A BL32 / U-Boot / Linux kernel
#        1: TF-A BL2 / OP-TEE / U-Boot / Linux kernel
set $debug_trusted_bootchain = 0 or 1 #depending on your software boot chain configuration
```

When this operation is complete, jump to Running OpenOCD and GDB.

5.2.3.2 TF-A(BL32) boot case

In that case, the GDB breaks in sp_min_entrypoint function.

Select the right configuration in Setup.gdb:

```
# Set debug phase:
#        1: Attach at Cortex-A7 / TF-A(BL2)
#        2: Attach at Cortex-A7 / TF-A(BL32) or OP-TEE
#        3: Attach at Cortex-A7 / U-Boot
#        4: Attach at Cortex-A7 / Linux kernel
set $debug_phase = 2
```

```
#        0: Attach at boot
#        1: Attach running target
set $debug_mode = 0
```

```
# Set debug trusted bootchain:
#        0: TF-A BL2 / TF-A BL32 / U-Boot / Linux kernel
#        1: TF-A BL2 / OP-TEE / U-Boot / Linux kernel
set $debug_trusted_bootchain = 0
```

When this operation is complete, jump to Running OpenOCD and GDB.

5.2.3.3 OP-TEE boot case

In that case, the GDB breaks in _start function.

Select the right configuration in Setup.gdb:

```
# Set debug phase:
#        1: Attach at Cortex-A7 / TF-A(BL2)
#        2: Attach at Cortex-A7 / TF-A(BL32) or OP-TEE
#        3: Attach at Cortex-A7 / U-Boot
#        4: Attach at Cortex-A7 / Linux kernel
set $debug_phase = 2
```

```
#        0: Attach at boot
#        1: Attach running target
set $debug_mode = 0
```

```
# Set debug trusted bootchain:
#        0: TF-A BL2 / TF-A BL32 / U-Boot / Linux kernel
#        1: TF-A BL2 / OP-TEE / U-Boot / Linux kernel
set $debug_trusted_bootchain = 1
```

When this operation is complete, jump to Running OpenOCD and GDB.

5.2.3.4 U-Boot boot case

In that case, the GDB breaks in in _start function.

Select the right configuration in Setup.gdb:

```
# Set debug phase:
#        1: Attach at Cortex-A7 / TF-A(BL2)
#        2: Attach at Cortex-A7 / TF-A(BL32) or OP-TEE
#        3: Attach at Cortex-A7 / U-Boot
#        4: Attach at Cortex-A7 / Linux kernel
set $debug_phase = 3
```

```
#        0: Attach at boot
#        1: Attach running target
set $debug_mode = 0
```

```
# Set debug trusted bootchain:
#        0: TF-A BL2 / TF-A BL32 / U-Boot / Linux kernel
#        1: TF-A BL2 / OP-TEE / U-Boot / Linux kernel
set $debug_trusted_bootchain = 0 or 1 #depending on your software boot chain configuration
```

When this operation is complete, jump to Running OpenOCD and GDB.

5.2.3.5 Linux kernel boot case

In that case, the GDB breaks in stext function.

Select the right configuration in Setup.gdb:

```
# Set debug phase:
#        1: Attach at Cortex-A7 / TF-A(BL2)
#        2: Attach at Cortex-A7 / TF-A(BL32) or OP-TEE
#        3: Attach at Cortex-A7 / U-Boot
#        4: Attach at Cortex-A7 / Linux kernel
set $debug_phase = 4
```

```
#        0: Attach at boot
#        1: Attach running target
set $debug_mode = 0
```

```
# Set debug trusted bootchain:
#        0: TF-A BL2 / TF-A BL32 / U-Boot / Linux kernel
#        1: TF-A BL2 / OP-TEE / U-Boot / Linux kernel
set $debug_trusted_bootchain = 0 or 1 #depending on your software boot chain configuration
```

When this operation is complete, jump to Running OpenOCD and GDB.

5.2.4 Running OpenOCD and GDB

- Prerequisites

Before running OpenOCD and GDB, check that the target board is in the right state.

For all configurations except GDB attachment to a running SSBL (U-Boot), the board has to operate in OpenSTLinux running mode.

In case of attachment to a running SSBL (U-Boot) configuration, the board target must be in U-Boot console mode:

```
#Reboot the target board

#Press any key to stop at U-Boot execution when booting the board
STM32MP> ...
STM32MP> Hit any key to stop autoboot:  0
STM32MP>
```

When you are in the expected configuration, two different consoles must be started: **one for OpenOCD** and **one for GDB**.
The SDK environment must be installed on both console terminals.

- OpenOCD console

```
# First console for starting openocd with configuration file
PC $> source <Your_SDK_path>/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
PC $> openocd -f <board.cfg>
```

Possible target configuration files for *<board.cfg>*:

Target board	Adapter	SWD mode board.cfg	JTAG mode board.cfg
STM32MP157C-EV1	ST-LINK *	board /stm32mp15x_ev1_stlink_swd.cfg	board /stm32mp15x_ev1_stlink_jtag.cfg
		board	

Target board	Adapter	SWD mode board.cfg	JTAG mode board.cfg
STM32MP157C-EV1	U-LINK2	/stm32mp15x_ev1_ulink2_swd.cfg	board /stm32mp15x_ev1_ulink2_jtag.cfg
STM32MP157C-EV1	J-LINK	board /stm32mp15x_ev1_jlink_swd.cfg	board/stm32mp15x_ev1_jlink_jtag.cfg
STM32MP157X-DK2	ST-LINK *	board/stm32mp15x_dk2.cfg	⊗**

** Both v2 and v3 are supported.*

*** JTAG wires are not connected in DK2.*

Note: It is recommended to use SWD, which is faster than JTAG.

- GDB console

Warning

The GDB must be executed from the directory where the scripts have been installed (i.e. $HOME /gdbscripts/)

```
# Second console for starting the GDB
PC $> cd $HOME/gdbscripts/
PC $> source <Your_SDK_path>/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
PC $> $GDB -x Setup.gdb
```

5.2.5 To know more about Linux kernel debug with GDB

Please refer to Debugging the Linux kernel using the GDB.

5.2.6 Access to STM32MP registers

5.2.6.1 Using gdb command line

- The following monitoring commands can be used to read a register value:

```
(gdb) monitor mdb <phys_address> [count] #Display target memory as 8-bit bytes
(gdb) monitor mdh <phys_address> [count] #Display target memory as 16-bit bytes
(gdb) monitor mdw <phys_address> [count] #Display target memory as 32-bit bytes
```

For example: Read RCC_MP_APB1ENSETR register on STM32MP1 to check RCC APB1 peripheral enable status.

```
(gdb) monitor mdw phys 0x50000a00 #full 32bits value result requested
0x50000a00: 00010000 --> UART4 is enable as explained in STM32MP15 reference manuals
```

- The following monitoring commands can be used to set a register value:

```
(gdb) monitor mwb <phys_address> <value> [count] #Write byte(s) to target memory
(gdb) monitor mwh <phys_address> <value> [count] #Write 16-bit half-word(s) to target
memory
(gdb) monitor mww <phys_address> <value> [count] #Write 32-bit word(s) to target memory
```

For example: Write RCC_MP_APB1ENCLRR register on STM32MP1 to clear the UART4 RCC of APB1 peripheral, then reenable it:

```
(gdb) monitor mww phys 0x50000a04 0x00010000 #full 32bits value given

# You can then check that UART4 is disable by reading the RCC_MP_APB1ENSETR register:
(gdb) monitor mdw phys 0x50000a00
0x50000a00: 00000000

# You can also check that the console is disabled by going on running the GDB
(gdb) c

# When the GBD has stopped running, you can re-enable UART4 RCC:
(gdb) monitor mww phys 0x50000a00 0x00010000
```

5.2.6.2 *Using CMSIS-SVD environment*

The CMSIS-SVD environment is useful to get detailed information on registers, such as name and bit descriptions.

It is based on python scripts and svd files which contain the description of all registers.

Refer to CMSIS-SVD environment and scripts for more details.

5.3 Debug Cortex-M4 firmware with GDB

The Arm Cortex-M4 core firmware can also be debugged using the GDB in command line (without IDE).

Either engineering boot mode and production boot mode are supported.

Please refer to the **Hardware Description** (Category:STM32 MPU boards) of your board for information on the Boot mode selection switch.

5.3.1 Debug Cortex-M4 firmware in engineering boot mode

As in previous chapter Running OpenOCD and GDB, both OpenOCD and GDB have to be started on separate consoles.

OpenOCD has to be executed in the same way as in the mentionned chapter.

GDB, instead, has to be executed with a different command line, without the script but with the path of ELF file containing the firmware to be debugged:

```
# Second console for starting the GDB
PC $> source <Your_SDK_path>/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
PC $> $GDB $HOME/path/to/file.elf
```

Then, the following commands has to be typed in the GDB console to start the execution of a firmware and halt it at the beginning of main():

```
(gdb) target extended-remote localhost:3334
(gdb) load
(gdb) thbreak main
(gdb) continue
```

Once the execution halts at main(), GDB will return the prompt allowing the debug of the firmware.

5.3.2 Debug Cortex-M4 firmware in production boot mode

In production mode the firmware is started by Linux, independently from GDB. Nevertheless, GDB can set a breakpoint before the firmware is started by Linux; thus the firmware will halt at the breakpoint, allowing GDB to debug the firmware.

Please check in Linux remoteproc framework overview how to start and stop a firmware using Linux remoteproc framework.

At first, verify that no firmware is running on Cortex-M4 or, eventually, stop it.

Then, start OpenOCD and GDB as in previous chapter Debug Cortex-M4 firmware in engineering boot mode.

Type the following commands in the GDB console:

```
(gdb) target extended-remote localhost:3334
(gdb) thbreak main
(gdb) continue
```

Finally, let Linux start the firmware. The firmware execution will halt at main() and GDB will return the prompt allowing the debug of the firmware.

Warning

Use on GDB command line the exact same ELF file that is used by Linux to read and run the firmware. If the files are not the same, the symbols on GDB will not match the firmware in execution.

5.4 Debug Linux application with gdbserver

5.4.1 Enable debug information

Once your program is built using the sdk toolchain, make sure that the **-g** option is enabled to debug your program and add the necessary debug information.

*Note: If an issue occurs during debugging, you can also force gcc to "not optimize code" using the **-O0** option.*

- Example of a simple test program build:

```
PC $> $CC -g -o myappli myappli.c
```

- Example based on Hello World: refer to "hello world" user space example

Edit and update the makefile for the user space example:

```
...
# Add / change option in CFLAGS if needed
-# CFLAGS += <new option>
+ CFLAGS += -g
...
```

5.4.2 Remote debugging using gdbserver

In this setup, an **ethernet link must be set between the host PC and the target board**.

Once your program is installed on the target (using ssh or copied from an SDcard), you can start debugging it.

- On target side: based on "Hello world" user space example

```
Board $> cd /usr/local/bin
Board $> ls
   hello_world_example
Board $> gdbserver host:1234 hello_world_example
  Process main created; pid = 11832 (this value depends on your target)
  Listening on port 1234
```

- Your target waits for remote PC connection, and then starts debugging.
- Launch the GDB command from your source file folder (easier source loading)

The SDK environment must be installed.

```
PC $> cd <source file folder path>
PC $> ls
    hello_world_example  hello_world_example.c  hello_world_example.o
kernel_install_dir  Makefile
PC $> source <Your_SDK_path>/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
PC $> $GDB
   GNU gdb (GDB) X.Y.Z
   ...
   This GDB was configured as "--host=x86_64-ostl_sdk-linux --target=arm-ostl-linux-
gnueabi".
   ...
(gdb)
```

- Connect to the target and load the source file:

```
(gdb) target remote <IP_Addr_of_Board>:1234
  Remote debugging using <IP_Addr_of_Board>:1234
  Reading /home/root/test from remote target...
  ...
(gdb) break 16 (line number in the source file)
(gdb) continue
```

- The target program breaks on the breakpoint. Proceed until the end of the program:

```
(gdb) continue
  Continuing.
  [Inferior 1 (process 16204) exited normally]
(gdb) quit
```

5.5 User interface application

5.5.1 Text user interface (TUI) mode

This user interface mode is the first step before using the graphical UI as GDBGUI or DDD.

The TUI can be very useful to map source code with instruction.

Please go through the online documentation [2][3].

5.5.2 Debugging with GDBGUI

Please refer to the dedicated gdbgui article.

5.5.3 Debugging with DDD

GNU DDD is a graphical front-end for command-line debuggers. Please refer to dedicated web page for details[4].

5.5.4 Debugging with IDE

Please refer to STM32CubeIDE.

6 To go further

6.1 Useful GDB commands

When using the GDB in command line mode, it is important to know some basic GDB commands, such as run software, set breakpoints, execute step by step, print variables and display information.

Please refer to GDB commands article.

6.2 Core dump analysis using GDB

The core dump generated for an application crash can be analysed by using the GDB.

Developer Package components, such as SDK and symbol files, must installed. Please refer to STM32MP1 Developer Package .

The symbol file containing the debug symbol of the application in which the crash occurred must also be available.

- First enable the SDK environment:

```
PC $> source <Your_SDK_path>/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
```

- Then play the core dump with GDB:

```
PC $> $GDB <path_to_the_binary> <path_to_the_core_dump_file>
```

6.3 Tips

- Managing multiple debug setups

To manage multiple debug setups for different software versions, create different Path_env.py files with different names, and call the expected file in the Setup.gdb file:

```
...
#################################################################

# Set environment configuration
#Path_env.py
source Path_env_dk2_18_12_14.py

#################################################################
...
```

7 References

- [1.01.1] https://www.gnu.org/software/gdb
- https://sourceware.org/gdb/onlinedocs/gdb/TUI.html#TUI
- https://ftp.gnu.org/old-gnu/Manuals/gdb/html_chapter/gdb_19.html
- https://www.gnu.org/software/ddd
- Useful external links

Document link	Document Type	Description
Using kgdb, kdb and the kernel debugger internals	User Guide	KGDB documentation guide
Welcome to the GDB Wiki	User guide	GDB Wiki
Building GDB and GDBserver for cross debugging	User Guide	Explain how to build gdb for target and host
A GDB Tutorial with Examples	Training	Debugging a simple application

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

User Interface

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

A quality version of this page, approved on *17 November 2021*, was based off this revision.

This article describes how to get and use the **Developer Package** of the **STM32MPU Embedded Software** for any development platform of the **STM32MP1 family** (STM32MP15 boards), in order to modify some of its pieces of software, or to add applications on top of it.

It lists some **prerequisites** in terms of knowledges and development environment, and gives the **step-by-step** approach to download and install the STM32MPU Embedded Software components for this Package.

Finally, it proposes some guidelines to upgrade (add, remove, configure, improve...) any piece of software.

Contents

1 Developer Package content

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

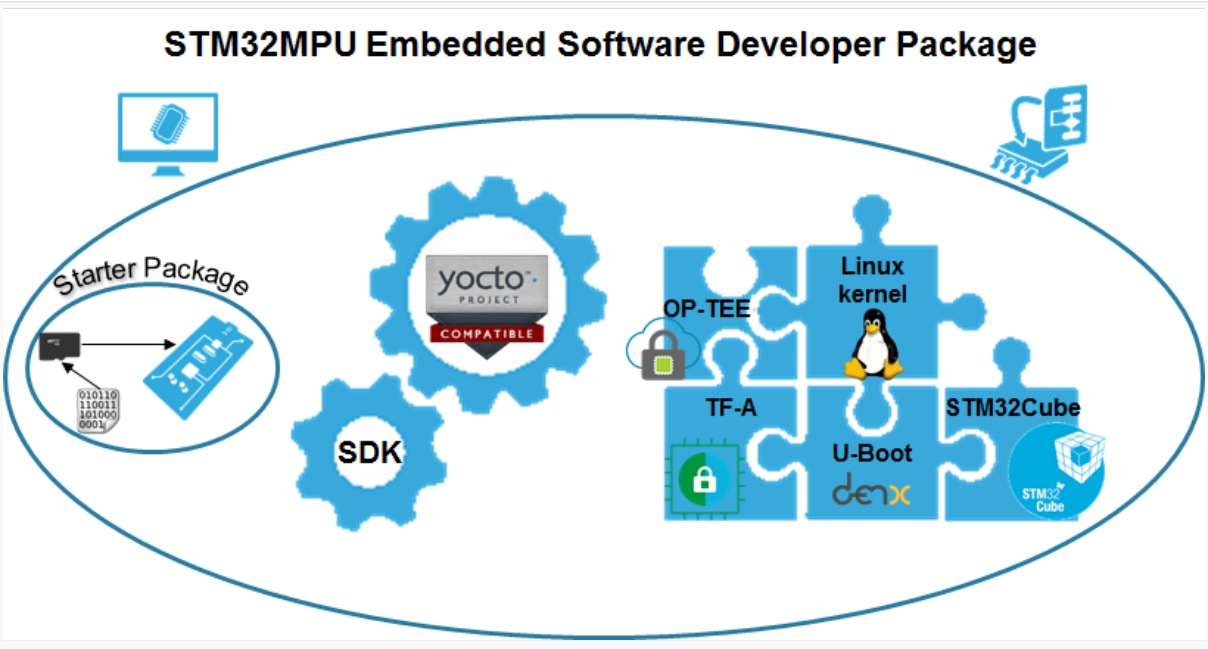
- Which STM32MPU Embedded Software Package better suits your needs (and especially the Developer Package chapter)
- STM32MPU Embedded Software distribution

If you are already familiar with the Developer Package for the STM32MPU Embedded Software distribution, the fast links to essential commands might interest you.

To sum up, this **Developer Package** provides:

- for the **OpenSTLinux distribution** (development on Arm® Cortex®-A processor):
  - the **software development kit** (SDK), based on Yocto SDK, for cross-development on an host PC
  - the following pieces of software in **source code**:
    - Linux® kernel
    - U-Boot
    - Trusted Firmware-A (TF-A)
    - optionally, Open source Trusted Execution Environment (OP-TEE)
  - the **debug symbol files** for Linux® kernel, U-Boot and TF-A
- for the **STM32Cube MPU Package** (development on Arm® Cortex®-M processor):
  - all pieces of software (BSP, HAL, middlewares and applications) in **source code**
  - the **integrated development environment (IDE)** (STM32CubeIDE)

Note that, the application frameworks for the OpenSTLinux distribution are not available as source code in this Package.



Components of the Developer Package

2 Developer Package step-by-step overview

The steps to get the STM32MPU Embedded Software Developer Package ready for your developments, are:

Checking the prerequisites

Installing the Starter Package for your board

Installing the components to develop software running on Arm® Cortex®-A (OpenSTLinux distribution)

Installing the SDK (**mandatory** for any development on Arm® Cortex®-A)

Installing the Linux kernel (**mandatory only** if you plan to modify the Linux kernel or to add external out-of-tree Linux kernel modules)

Installing the U-Boot (**mandatory only** if you plan to modify the U-Boot)

Installing the TF-A (**mandatory only** if you plan to modify the TF-A)

Installing the debug symbol files (**mandatory only** if you plan to debug Linux® kernel, U-Boot or TF-A with GDB)

Installing the components to develop software running Arm Cortex-M (STM32Cube MPU Package)

Installing STM32CubeIDE (**mandatory** for any development on Arm® Cortex®-M)

Installing the STM32Cube MPU Package (**mandatory only** if you plan to modify the Cube firmware)

Once these steps are achieved, you are able to:

- develop software running on Arm Cortex-A
  - Modifying the Linux kernel
  - Adding external out-of-tree Linux kernel modules
  - Adding Linux user space applications
  - Modifying the U-Boot
  - Modifying the TF-A
- develop software running on Arm Cortex-M4

3 Checking the prerequisites

3.1 Knowledges

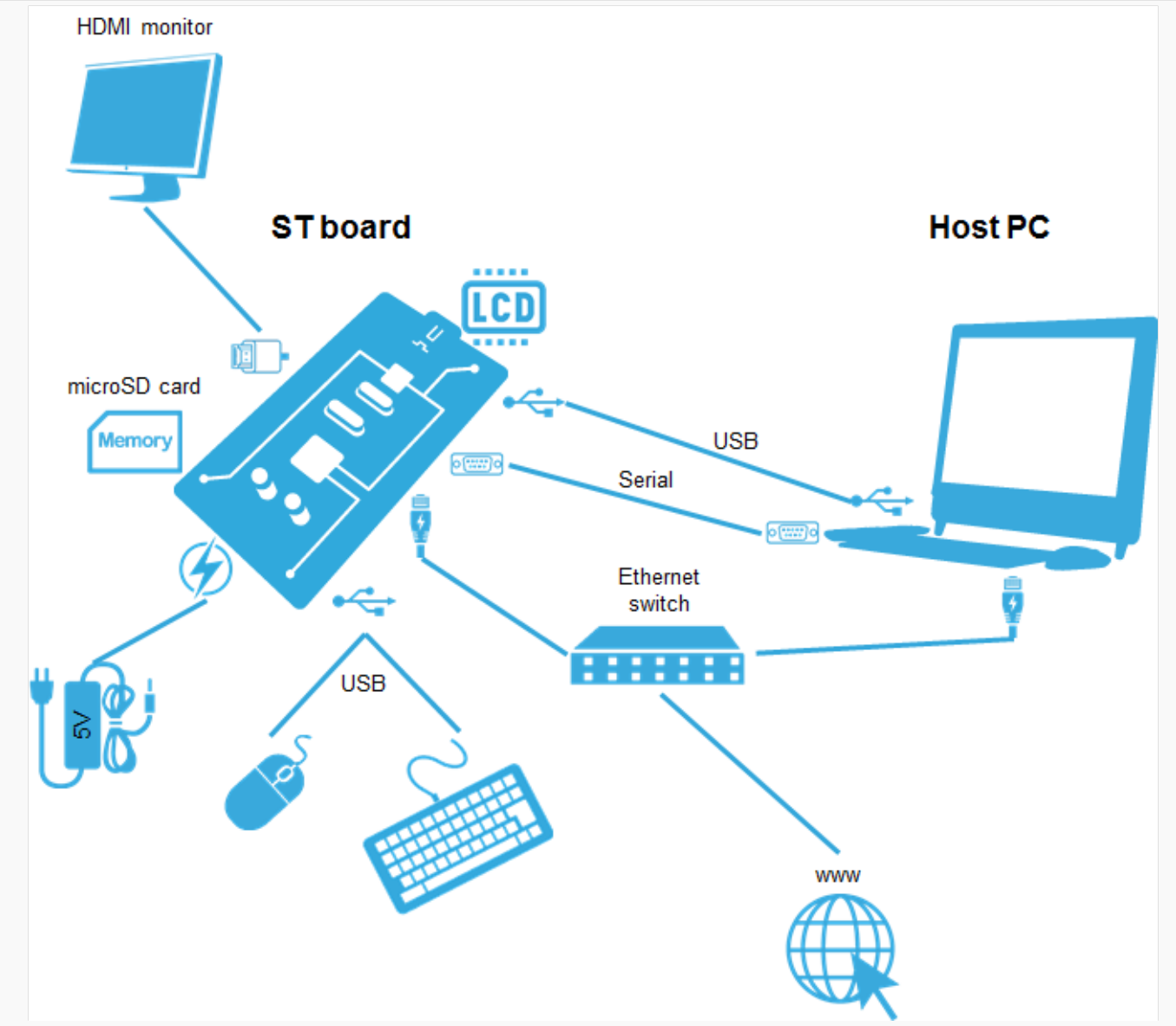
The STM32MP1 Developer Package aims at enriching a Linux-based software for the targeted product: basic knowledges about Linux are recommended to make the most of this Package.

Having a look at the STM32MPU Embedded Software architecture overview is also highly recommended.

3.2 Development setup

The recommended setup for the development PC (host) is specified in the following article: PC prerequisites.

Whatever the development platform (board) and development PC (host) used, the range of possible development setups is illustrated by the picture below.



Development setup for Developer and Distribution Packages

The following components are **mandatory**:

- Host PC for cross-compilation and cross-debugging, installed as specified above
- Board assembled and configured as specified in the associated Starter Package article
- Mass storage device (for example, microSD card) to load and update the software images (binaries)

The following components are **optional**, but **recommended**:

- A serial link between the host PC (through Terminal program) and the board for traces (even early boot traces), and access to the board from the remote PC (command lines)
- An Ethernet link between the host PC and the board for cross-development and cross-debugging through a local network. This is an alternative or a complement to the serial (or USB) link
- A display connected to the board, depending on the technologies available on the board: DSI LCD display, HDMI monitor (or TV) and so on
- A mouse and a keyboard connected through USB ports

Additional optional components can be added by means of the connectivity capabilities of the board: cameras, displays, JTAG, sensors, actuators, and much more.

4 Installing the Starter Package

Before developing with the Developer Package, **it is essential to start up your board thanks to its Starter Package**. All articles relative to Starter Packages are found in Category:Starter Package: find the one that corresponds to your board, and follow the installation instructions (if not yet done), before going further.

In brief, it means that:

- your board boots successfully
- the flashed image comes from the same release of the STM32MPU Embedded Software distribution than the components that will be downloaded in this article

Thanks to the Starter Package, **all Flash partitions are populated**.

Then, with the Developer Package, it is possible to modify or to upgrade the partitions independently one from the others.

For example, if you only want to modify the Linux kernel (part of *bootfs* partition), installing the SDK and the Linux kernel are enough; no need to install anything else.

5 Installing the components to develop software running on Arm Cortex-A (OpenSTLinux distribution)

5.1 Installing the SDK

Optional step: it is mandatory only if you want to modify or add software running on Arm Cortex-A (e.g. Linux kernel, Linux user space applications...).

The SDK for OpenSTLinux distribution provides a stand-alone cross-development toolchain and libraries tailored to the contents of the specific image flashed in the board. If you want to know more about this SDK, please read the SDK for OpenSTLinux distribution article.

- The STM32MP1 SDK is delivered through a tarball file named : **en.SDK-x86_64-stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17.tar.xz**
- Download and install the STM32MP1 SDK.

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

Warning

To download a package, it is recommended to be logged in to your "myst" account [1]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package SDK - STM32MP15-Ecosystem-v3.1.0 release
Download	You need to be logged on *my.st.com* before accessing the following link: en.SDK-x86_64-stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17.tar.xz

	• Uncompress the tarball file to get the SDK installation script
	`PC $> tar xvf en.SDK-x86_64-stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17.tar.xz`
	• If needed, change the permissions on the SDK installation script so that it is executable
	`PC $> chmod +x stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/sdk/st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-3.1.11-openstlinux-5.10-dunfell-mp1-21-11-17.sh`
	• Run the SDK installation script
	  • Use the *-d <SDK installation directory absolute path>* option to specify the absolute path to the directory in which you want to install the SDK (*<SDK installation directory>*)
	  • If you follow the proposition to organize the working directory, it means:

	STM32MP1 Developer Package SDK - STM32MP15-Ecosystem-v3.1.0 release
Installation	**PC $>** ./stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/sdk/st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-3.1.11-openstlinux-5.10-dunfell-mp1-21-11-17.sh -d <working directory absolute path>/Developer-Package/SDK • A successful installation outputs the following log: ST OpenSTLinux - Weston - (A Yocto Project Based Distro) SDK installer version 3.1.11-openstlinux-5.10-dunfell-mp1-21-11-17 ========== You are about to install the SDK to "<working directory absolute path>/Developer-Package/SDK". Proceed [Y/n]? Extracting SDK.....done Setting it up...done SDK has been successfully set up and is ready to be used. Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g. $> . <working directory absolute path>/Developer-Package/SDK/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
Release note	Details about the content of the SDK are available in the **associated** STM32MP15 ecosystem release note. If you are interested in older releases, please have a look into the section Archives.

- The SDK is in the *<SDK installation directory>*:

```
<SDK installation directory>                                    SDK for OpenSTLinux
distribution: details in Standard SDK directory structure article
├── environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi  Environment setup script
 for Developer Package
├── site-config-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
├── sysroots
│   ├── cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi              Target sysroot
(libraries, headers, and symbols)
│   │   └── [...]
│   └── x86_64-ostl_sdk-linux                                   Native sysroot
(libraries, headers, and symbols)
│       └── [...]
└── version-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
```

Warning

Now that the SDK is installed, please do not move or rename the *<SDK installation directory>*.

5.1.1 Starting up the SDK

The SDK environment setup script must be run once in each new working terminal in which you cross-compile:

```
PC $> source <SDK installation directory>/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
```

The following checkings allow to ensure that the environment is correctly setup:

- Check the target architecture

```
PC $> echo $ARCH
arm
```

- Check the toolchain binary prefix for the target tools

```
PC $> echo $CROSS_COMPILE
arm-ostl-linux-gnueabi-
```

- Check the C compiler version

```
PC $> $CC --version
arm-ostl-linux-gnueabi-gcc (GCC) <GCC version>
[...]
```

- Check that the SDK version is the expected one

```
PC $> echo $OECORE_SDK_VERSION
<expected SDK version>
```

If any of these commands fails or does not return the expected result, please try to reinstall the SDK.

5.2 Installing the Linux kernel

Optional step: **it is mandatory only if you want to modify the Linux kernel (configuration, device tree, driver...), or to add external out-of-tree Linux kernel modules.**

Prerequite: the SDK is installed.

5.2.1 Downloading the Linux kernel

- The STM32MP1 Linux kernel is delivered through a tarball file named **en.SOURCES-kernel-stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17.tar.xz** for STM32MP157x-EV1 and STM32MP157x-DKx boards.

- Download and install the STM32MP1 Linux kernel

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

Warning

To download a package, it is recommended to be logged in to your "myst" account [2]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package Linux kernel - STM32MP15-Ecosystem-v3.1.0 release
Download	You need to be logged on to *my.st.com* before accessing the following link en.SOURCES-kernel-stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17.tar.xz

Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, this means:
	`PC $> cd <working directory path>/Developer-Package`
	• Download the tarball file in this directory
	• Uncompress the tarball file to get the Linux kernel (Linux kernel source code, ST patches, ST configuration fragments...):
	`PC $> tar xvf en.SOURCES-kernel-stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17.tar.xz`
	`PC $> cd stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/sources/arm-ostl-linux-gnueabi/linux-stm32mp-5.10.61-stm32mp-r2-r0`
	`PC $> tar xvf linux-5.10.61.tar.xz`

Release note	Details of the content of the Linux kernel are available in the **associated** STM32MP15 OpenSTLinux release note. If you are interested in older releases, please have a look into the section Archives.

- The **Linux kernel installation directory** is in the *<Developer Package installation directory>/stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/sources/arm-ostl-linux-gnueabi* directory, and is named *linux-stm32mp-<kernel version>*:

```
linux-stm32mp-5.10.61-r2          Linux kernel installation directory
├── [*].patch                         ST patches to apply during the Linux kernel
preparation (see next chapter)
├── fragment-[*].config                 ST configuration fragments to apply during the
Linux kernel configuration (see next chapter)
├── linux-5.10.61                    Linux kernel source code directory
├── linux-5.10.61.tar.xz            Tarball file of the Linux kernel source code
├── README.HOW_TO.txt                 Helper file for Linux kernel management: reference
for Linux kernel build
└── series                            List of all ST patches to apply
```

5.2.2 Building and deploying the Linux kernel for the first time

It is mandatory to execute once the steps specified below before modifying the Linux kernel, or adding external out-of-tree Linux kernel modules.

The partitions related to the Linux kernel are:

- the *bootfs* partition that contains the Linux kernel U-Boot image (*uImage*) and device tree
- the *rootfs* partition that contains the Linux kernel modules

The Linux kernel might be cross-compiled, either in the source code directory, or in a dedicated directory different from the source code directory.
This last method is recommended as it clearly separates the files generated by the cross-compilation from the source code files.

Information

The README_HOWTO.txt helper file is **THE** reference for the Linux kernel build

Warning

The SDK must be started

Open the *<Linux kernel installation directory>/README.HOW_TO.txt* helper file, and execute its instructions to:

setup a software configuration management (SCM) system (*git*) for the Linux kernel (optional but recommended)

prepare the Linux kernel (applying the ST patches)

configure the Linux kernel (applying the ST fragments)

cross-compile the Linux kernel

deploy the Linux kernel (i.e. update the software on board)

The Linux kernel is now installed: let's modify the Linux kernel, or add external out-of-tree Linux kernel modules.

5.3 Installing the U-Boot

Optional step: it is mandatory only if you want to modify the U-Boot.

Prerequite: the SDK is installed.

5.3.1 Downloading the U-Boot

- The STM32MP1 U-Boot is delivered through a tarball file named **en.SOURCES-u-boot-stm32mp1-openstlinux-5-10-dunfell-mp1-21-11-17_tar.xz** for STM32MP157x-EV1 and STM32MP157x-DKx boards.
- Download and install the STM32MP1 U-Boot

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

Warning

To download a package, it is recommended to be logged in to your "myst" account [3]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package U-Boot - STM32MP15-Ecosystem-v3.1.0 release
Download	You need to be logged on to *my.st.com* before accessing the following link en.SOURCES-u-boot-stm32mp1-openstlinux-5-10-dunfell-mp1-21-11-17_tar.xz

Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, this means:
	`PC $> cd <working directory path>/Developer-Package`
	• Download the tarball file in this directory
	• Uncompress the tarball file to get the U-Boot (U-Boot source code, ST patches and so on):
	`PC $> tar xvf en.SOURCES-u-boot-stm32mp1-openstlinux-5-10-dunfell-mp1-21-11-17_tar.xz`
	`PC $> cd stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/sources/arm-ostl-linux-gnueabi/u-boot-stm32mp-v2020.10-stm32mp-r2-r0`
	`PC $> tar xvf u-boot-stm32mp-v2020.10-stm32mp-r2-r0.tar.gz`

Release note	Details of the content of the U-Boot are available in the **associated** STM32MP15 OpenSTLinux release note. If you are interested in older releases, please have a look into the section Archives.

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

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

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

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

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

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

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

5.3.2 Building and deploying the U-Boot for the first time

It is mandatory to execute once the steps specified below before modifying the U-Boot.

As explained in the boot chain overview, the trusted boot chain is the default solution delivered by STMicroelectronics.

Within this scope, the partition related to the U-Boot is the *ssbl* one that contains the U-Boot and its device tree blob.

Information

The README_HOWTO.txt helper file is **THE** reference for the U-Boot build

Information

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

Warning

The SDK must be started

Open the *<U-Boot installation directory>/README.HOW_TO.txt* helper file, and execute its instructions to:

setup a software configuration management (SCM) system (*git*) for the U-Boot (optional but recommended)

prepare the U-Boot (applying the ST patches)

cross-compile the U-Boot

deploy the U-Boot (i.e. update the software on board)

The U-Boot is now installed: let's modify the U-Boot.

5.4 Installing the TF-A

Optional step: it is mandatory only if you want to modify the TF-A.

Prerequite: the SDK is installed.

5.4.1 Downloading the TF-A

- The STM32MP1 TF-A is delivered through a tarball file named **en.SOURCES-tf-a-stm32mp1-openstlinux-5-10-dunfell-mp1-21-11-17_tar.xz** for STM32MP157x-EV1 and STM32MP157x-DKx boards.
- Download and install the STM32MP1 TF-A

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

Warning

To download a package, it is recommended to be logged in to your "myst" account [4]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package TF-A - STM32MP15-Ecosystem-v3.1.0 release
Download	You need to be logged on *my.st.com* before accessing the following link: en.SOURCES-tf-a-stm32mp1-openstlinux-5-10-dunfell-mp1-21-11-17_tar.xz

Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, it means:
	`PC $> cd <working directory path>/Developer-Package`
	• Download the tarball file in this directory
	• Uncompress the tarball file to get the TF-A (TF-A source code, ST patches...):
	`PC $> tar xvf en.SOURCES-tf-a-stm32mp1-openstlinux-5-10-dunfell-mp1-21-11-17_tar.xz`
	`PC $> cd stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/sources/arm-ostl-linux-gnueabi/tf-a-stm32mp-v2.4-stm32mp-r2-r0`
	`PC $> tar xvf tf-a-stm32mp-v2.4-stm32mp-r2-r0.tar.gz`

Release note	Details about the content of the TF-A are available in the **associated** STM32MP15 OpenSTLinux release note. If you are interested in older releases, please have a look into the section Archives.

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

```
tf-a-stm32mp-v2.4-stm32mp-r2-r0                          TF-A installation directory
    ├── [*].patch                                ST patches to apply during the TF-A preparation
(see next chapter)
    ├── tf-a-stm32mp-v2.4-stm32mp-r2                      TF-A source code directory
    ├── Makefile.sdk                           Makefile for the TF-A compilation
    ├── README.HOW_TO.txt                      Helper file for TF-A management: reference for
TF-A build
    ├── series                                    List of all ST patches to apply
    └── tf-a-stm32mp-v2.4-stm32mp-r2-r0.tar.gz        Tarball file of the TF-A source code
```

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

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

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

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

5.4.2 Building and deploying the TF-A for the first time

It is mandatory to execute once the steps specified below before modifying the TF-A.

As explained in the boot chain overview, the trusted boot chain is the default solution delivered by STMicroelectronics.

Within this scope, the partition related to the TF-A is the *fsbl* one.

Information

The README_HOWTO.txt helper file is **THE** reference for the TF-A build

Information

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

Warning

The SDK must be started

Open the *<TF-A installation directory>/README.HOW_TO.txt* helper file, and execute its instructions to:

setup a software configuration management (SCM) system (*git*) for the TF-A (optional but recommended)

prepare the TF-A (applying the ST patches)

cross-compile the TF-A

deploy the TF-A (i.e. update the software on board)

The TF-A is now installed: let's modify the TF-A.

5.5 Installing the OP-TEE

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

Prerequite: the SDK is installed.

5.5.1 Downloading the OP-TEE

- The STM32MP1 OP-TEE is delivered through a tarball file named **en.SOURCES-optee-stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17.tar.xz** for STM32MP157x-EV1 and STM32MP157x-DKx boards.
- Download and install the STM32MP1 OP-TEE

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

Warning

To download a package, it is recommended to be logged in to your "myst" account [5]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package OP-TEE - STM32MP15-Ecosystem-v3.1.0 release
Download	You need to be logged on *my.st.com* before accessing the following link: en.SOURCES-optee-stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17.tar.xz

Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, it means:
	`PC $> cd <working directory path>/Developer-Package`
	• Download the tarball file in this directory
	• Uncompress the tarball file to get the OP-TEE (OP-TEE source code, ST patches...):
	`PC $> tar xvf en.SOURCES-optee-stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17.tar.xz`
	`PC $> cd stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/sources/arm-ostl-linux-gnueabi/optee-os-stm32mp-3.12.0-stm32mp-r2-r0`
	`PC $> tar xvf optee-os-stm32mp-3.12.0-stm32mp-r2-r0.tar.gz`

Release note	Details about the content of the OP-TEE are available in the **associated** STM32MP15 OpenSTLinux release note. If you are interested in older releases, please have a look into the section Archives.

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

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

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

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

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

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

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

5.5.2 Building and deploying the OP-TEE for the first time

It is mandatory to execute once the steps specified below before modifying the OP-TEE.

As explained in the boot chain overview, the trusted boot chain is the default solution delivered by STMicroelectronics. Within this scope, the partition related to the OP-TEE is the *fsbl* one.

Information

The README_HOWTO.txt helper file is **THE** reference for the OP-TEE build

Information

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

Warning

The SDK must be started

Open the *<OP-TEE installation directory>/README.HOW_TO.txt* helper file, and execute its instructions to:

setup a software configuration management (SCM) system (*git*) for the OP-TEE (optional but recommended)

prepare the OP-TEE (applying the ST patches)

cross-compile the OP-TEE

deploy the OP-TEE (i.e. update the software on board)

The OP-TEE is now installed: let's modify the OP-TEE.

5.6 Installing the debug symbol files

Optional step: it is mandatory only if you want to debug Linux® kernel, U-Boot or TF-A with GDB.

5.6.1 Downloading the debug symbol files

- The STM32MP1 debug symbol files is delivered through a tarball file named **en.DEBUG-stm32mp1-openstlinux-5-10-dunfell-mp1-21-11-17_tar.xz** for STM32MP157x-EV1 and STM32MP157x-DKx boards.
- Download and install the STM32MP1 debug symbol files

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

Warning

To download a package, it is recommended to be logged in to your "myst" account [6]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

	STM32MP1 Developer Package debug symbol files - STM32MP15-Ecosystem-v3.1.0 release
Download	You need to be logged on to *my.st.com* before accessing the following link en.DEBUG-stm32mp1-openstlinux-5-10-dunfell-mp1-21-11-17_tar.xz

Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, this means:
	`PC $> cd <working directory path>/Developer-Package`
	• Download the tarball file in this directory
	• Uncompress the tarball file to get the debug symbol files (for Linux kernel, U-Boot, TF-A and OP-TEE OS):
	`PC $> tar xvf en.DEBUG-stm32mp1-openstlinux-5-10-dunfell-mp1-21-11-17_tar.xz`

Release note	If you are interested in older releases, please have a look into the section Archives.

- The debug symbol files are in the *<Developer Package installation directory>*/stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/images/stm32mp1 *directory:*

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

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

5.6.2 Using the debug symbol files

These files are used to debug the Linux® kernel, U-Boot or TF-A with GDB. Especially, the Debug OpenSTLinux BSP components chapter explains how to load the debug symbol files in GDB.

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

6.1 Installing STM32CubeIDE

Optional step: it is needed if you want to modify or add software running on Arm Cortex-M.

The table below explains how to download and install STM32CubeIDE which addresses STM32 MCU, and also provides support for Cortex-M inside STM32 MPU.

STM32 MPU support inside STM32CubeIDE is available on Linux® and Windows® host PCs, but **it is NOT on macOS®.**

	STM32CubeIDE for Linux® host PC	STM32CubeIDE for Windows® host PC

Download	**Version 1.6.1**
	• Download the preferred all-in-one Linux installer from my.st.com
	  • *Generic Linux Installer - STM32CubeIDE-Lnx*
	  • *RPM Linux Installer - STM32CubeIDE-RPM*
	  • *Debian Linux Installer - STM32CubeIDE-DEB*
		Version 1.6.1
		• Download the all-in-one Windows installer from my.st.com
		  • *Windows Installer - STM32CubeIDE-Win*

Installation guide	• Refer to *STM32CubeIDE installation guide (UM2563)* available on my.st.com.	

User manual	• When the installation is completed, see additional information about STM32CubeIDE in my.st.com:
	  • *STM32CubeIDE quick start guide (UM2553)*
	  • *Getting started with projects based on the STM32MP1 Series in STM32CubeIDE (AN5360)*	

Detailed release note	• Details about the content of this tool version are available in the *STM32CubeIDE release v1.6.1* release note from my.st.com	

Minor releases may be available from the update site. Check chapter 10 in (UM2609) for more information on how to update STM32CubeIDE.

6.2 Installing the STM32Cube MPU Package

Optional step: it is mandatory only if you want to modify the STM32Cube MPU Package.

Prerequite: the STM32CubeIDE is installed.

- The STM32CubeMP1 Package is delivered through an archive file named **en.STM32Cube_FW_MP1_V1-5-0.zip**.

- Download and install the STM32CubeMP1 Package

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

Warning

To download a package, it is recommended to be logged in to your "myst" account [7]. If, trying to download, you encounter a “403 error”, you could try to empty your browser cache to workaround the problem. We are working on the resolution of this problem.
We apologize for this inconvenience

.

	STM32MP1 Developer Package STM32CubeMP1 Package - v3.1.0 release
Download	You need to be logged on *my.st.com* before accessing the following link: en.STM32Cube_FW_MP1_V1-5-0.zip

Installation	• Go to the host PC directory in which you want to install the Developer Package (*<Developer Package installation directory>*); if you follow the proposition to organize the working directory, it means:
	`PC $> cd <working directory path>/Developer-Package`
	• Download the archive file in this directory
	• Uncompress the archive file to get the STM32CubeMP1 Package:
	`PC $> unzip en.STM32Cube_FW_MP1_V1-5-0.zip`

Release note	Details about the content of the STM32CubeMP1 Package are available in the *STM32Cube_FW_MP1_V1.4.0/Release_Notes.html* file. If you are interested in older releases, please have a look into the section Archives.

- The **STM32CubeMP1 Package installation directory** is in the *<Developer Package installation directory>* directory, and is named *STM32Cube_FW_MP1_V1.5.0*:

```
STM32Cube_FW_MP1_V1.5.0                          STM32CubeMP1 Package: details in STM32CubeM
P1 Package content article
├── Drivers
│   ├── BSP                                  BSP drivers for the supported STM32MP1 boards
│   │   └── [...]
│   ├── CMSIS
│   │   └── [...]
│   └── STM32MP1xx_HAL_Driver                  HAL drivers for the supported STM32MP1
devices
│       └── [...]
├── _htmresc
│   └── [...]
├── License.md
├── Middlewares
│   └── [...]
```

```
├── package.xml
├── Projects
│   ├── STM32CubeProjectsList.html        List of examples and applications for
STM32CubeMP1 Package
│   ├── STM32MP157C-DK2                        Set of examples and applications →
STM32MP15 Discovery kits
│   │   └── [...]
│   └── STM32MP157C-EV1                        Set of examples and applications →
STM32MP15 Evaluation boards
│       └── [...]
├── Readme.md
├── Release_Notes.html                       Release note for STM32CubeMP1 Package
└── Utilities
    └── [...]
```

The STM32Cube MPU Package is now installed: let's develop software running on Arm Cortex-M4.

7 Developing software running on Arm Cortex-A7

7.1 Modifying the Linux kernel

Prerequisites:

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

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

configure the Linux kernel

cross-compile the Linux kernel

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

You can refer to the following simple examples:

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

7.2 Adding external out-of-tree Linux kernel modules

Prerequisites:

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

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

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

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

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

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

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

# Path to the directory that contains the generated objects
```

```
DESTDIR ?= <Linux kernel installation directory>

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

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

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

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

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

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

You can refer to the following simple example:

- Addition of an external out-of-tree module

7.3 Adding Linux user space applications

Prerequisites:

- the SDK is installed
- the SDK is started up

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

- the sysroot that is associated with the cross-toolchain, and that contains the header files and libraries needed for generating binaries (see target sysroot)
- the environment variables created by the SDK environment setup script (see SDK startup)

You can refer to the following simple example:

- Addition of a "hello world" user space application

7.4 Modifying the U-Boot

Prerequisites:

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

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

cross-compile the U-Boot

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

You can refer to the following simple example:

- Modification of the U-Boot

7.5 Modifying the TF-A

Prerequisites:

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

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

cross-compile the TF-A

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

You can refer to the following simple example:

- Modification of the TF-A

7.6 Modifying the OP-TEE

Prerequisites:

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

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

cross-compile the OP-TEE

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

8 Developing software running on Arm Cortex-M4

8.1 How to create a Cube project from scratch or open/modify an existing one from STM32Cube MPU package

Please refer to STM32CubeMP1 Package article.

9 Fast links to essential commands

If you are already familiar with the Developer Package for the STM32MPU Embedded Software distribution, fast links to the essential commands are listed below.

Information

With the links below, you will be redirected to other articles; use the *back* button of your browser to come back to these fast links

Link to the command
Starter Packages
Essential commands of the STM32MP15 Evaluation board Starter Package
Essential commands of the STM32MP15 Discovery kit Starter Package
SDK
Download and install the latest SDK
Start the SDK
Linux kernel
Download and install the latest Linux kernel
Helper file for the **Linux kernel** build, and update on board
U-Boot
Download and install the latest U-Boot
Helper file for the **U-Boot** build, and update on board
TF-A
Download and install the latest TF-A
Helper file for the **TF-A** build, and update on board
OP-TEE
Download and install the latest OP-TEE
Helper file for the **OP-TEE** build, and update on board
Linux user space
Simple user space application
STM32Cube MPU Package
Download and install the latest STM32CubeMP1 Package
Create or modify a Cube project

10 How to go further?

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

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

Universal Asynchronous Receiver/Transmitter