Hardware random overview

Contents

Stable: 17.02.2021 - 19:52 / Revision: 17.02.2021 - 19:51

Stable: 17.02.2021 - 10:31 | Revision: 17.02.2021 - 10:31

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

This article gives information about the Linux® hardware random framework.

Contents

1 Article Purpose

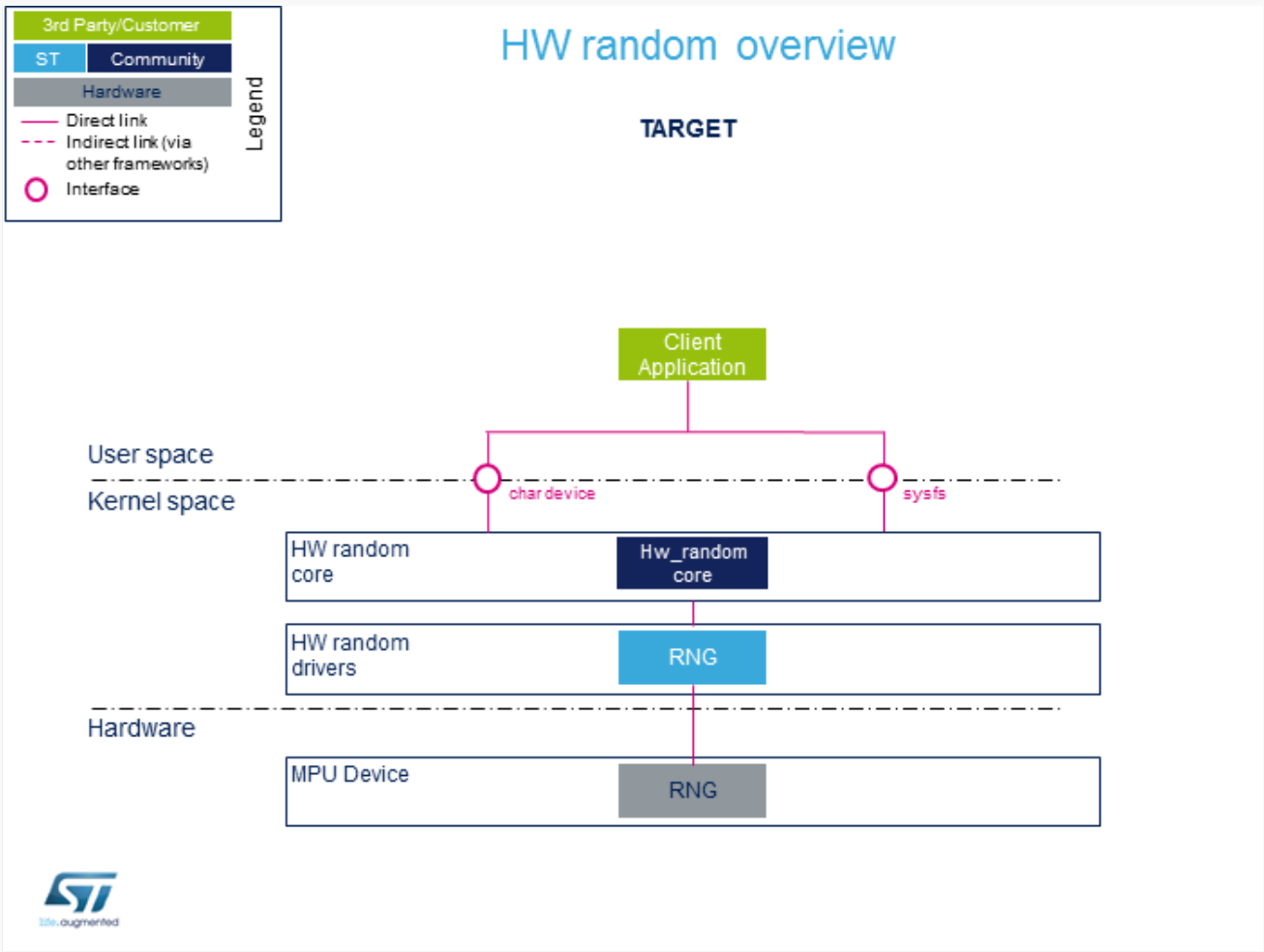
This article gives information about the hardware random (HWRNG) framework.

2 Framework purpose

The Hardware random framework is integrated in the kernel. It provides access to RNG peripherals and focuses on supporting the hardware number generator.

3 System overview

The HW random framework allows retrieving random numbers in userland.



Hw Random

3.1 Component description

- **HW random core** (Kernel space)

Generic interface in kernel space. This layer is in charge of creating the character device (char device) and sysfs to access hw_random.

- **RNG** (Kernel space)

Hardware random Linux® drivers handling the HW blocks.

- **RNG** (Hardware)

HW blocks handling the RNG peripheral.

3.2 API description

The Hardware random framework uses char device API[1] ioctl operations. For additional information, refer to:

- sysfs interface.
- Kernel Documentation directory[2]

4 Configuration

4.1 Kernel configuration

The Hardware random support is activated by default in ST deliveries. No specific configuration is required apart from enabling or disabling peripheral support using Linux® Menuconfig tool. Refer to Menuconfig or how to configure kernel and select:

```
[*] Device Drivers --->
  [*] Character devices --->
    [*] Hardware Random Number Generator Core support --->
      [*] STMicroelectronics STM32 random number generator
```

4.2 Device tree configuration

DT configuration can be done thanks to the STM32CubeMX.

A detailed device tree configuration is described in RNG device tree configuration.

5 How to use the framework

The framework provides external interfaces from userland : How to control RNG.

5.1 How to use from char device

The community tool for using Hardware random framework is rng_tools[3] which provides a complete set of utilities related to random number generators:

- **rngd**: runs a background daemon that opens /dev/hwrng file (default) to connect and retrieve random numbers.
- **rngtest**: runs different tests that check the entropy and verify the compliance regarding FIPS 140-2 standard.

5.2 How to use from sysfs

Available devices compatible with Hardware framework can be listed using sysfs commands:

```
Board $> cat /sys/class/misc/hw_random/rng_available
stm32-rng
```

The selected device is shown here:

```
Board $> cat /sys/class/misc/hw_random/rng_current
stm32-rng
```

To select a different device:

```
Board $> echo "stm32-rng"> /sys/class/misc/hw_random/rng_current
```

6 How to trace and debug the framework

Light information on the framework can be accessed by using sysfs.

By default, the framework does not provide any specific debug output or dynamic debugging tool.

7 Source code location

Hardware random drivers and framework are available here[4].

8 To go further

Code examples are directly available from rng-tools[3] github.

9 References

- https://bootlin.com/doc/legacy/accessing-hardware/accessing-hardware.pdf
- Documentation/admin-guide/hw_random.rst
- 3.03.1 Rng_tools source code
- drivers/char/hw_random , Hw_random sources

Random Number Generator

System File System (See https://en.wikipedia.org/wiki/Sysfs for more details)

Application programming interface

Device Tree

Stable: 03.02.2020 - 08:42 / Revision: 03.02.2020 - 08:27

A quality version of this page, approved on *3 February 2020*, was based off this revision.

Contents

1 Purpose

Hardware random framework offers the interface to control RNG devices from userspace.
This article shows two ways to control a RNG in userspace:

- using /dev/random command to generate a random number
- using rng-tools to validate the RNG

2 RNG control through /dev/random

/dev/random is a special file that can be used to generate random numbers based on a pseudo-random generator. It uses noise collected from device drivers and hardware random sources to generate data. od (octal dump) command is used to extract the number of bytes and display the decimal number.
Ex: - Random number (0 - 255):

```
Board $> od -An -N1 -i /dev/random
        172
```

- Random number (0 - 65535):

```
Board $> od -An -N2 -i /dev/random
        20041
```

3 RNG control through rng-tools

rng_tools[1] is a set of tools related to random number generation.
rng-tools will connect to the hardware random number generator through /dev/hwrng. rngtest is a basic test that checks data using FIPS 140-2 tests[2] which is a security requirement test for cryptographic module compliance.

```
Board $> rngtest -c 100 </dev
/hwrng

rngtest 5
Copyright (c) 2004 by Henrique de Moraes Holschuh
This is free software; see the source for copying conditions.  There is NO warranty; not
even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

rngtest: starting FIPS tests...
rngtest: bits received from input: 2000032
rngtest: FIPS 140-2 successes: 100
rngtest: FIPS 140-2 failures: 0
rngtest: FIPS 140-2(2001-10-10) Monobit: 0
rngtest: FIPS 140-2(2001-10-10) Poker: 0
rngtest: FIPS 140-2(2001-10-10) Runs: 0
rngtest: FIPS 140-2(2001-10-10) Long run: 0
rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
rngtest: input channel speed: (min=33.154; avg=33.656; max=34.217) Kibits/s
rngtest: FIPS tests speed: (min=21.193; avg=23.180; max=23.403) Mibits/s
rngtest: Program run time: 58114432 microseconds
```

It is normal for any random generator to fail in small number of tests, but failures must not exceed around 10.

4 References

- https://git.kernel.org/pub/scm/utils/kernel/rng-tools/rng-tools.git/
- https://en.wikipedia.org/wiki/FIPS_140-2

Random Number Generator
Stable: 11.02.2021 - 11:10 / Revision: 19.01.2021 - 10:34

A quality version of this page, approved on *11 February 2021*, was based off this revision.

Contents

1 Linux configuration genericity

The process of building a kernel has two parts: configuring the kernel options and building the source with those options.

The Linux® kernel configuration is found in the generated file: .config.

.config is the result of configuring task which is processing platform defconfig and fragment files if any.

For OpenSTLinux distribution the defconfig is located into the kernel source code and fragments into stm32mp BSP layer :

- arch/arm/configs/**multi_v7_defconfig**

Every new kernel version brings a bunch of new options, we do not want to back port them into a specific defconfig file each time the kernel releases, so we use the same defconfig file based on ARM SoC v7 architecture.
STM32MP1 specificities are managed with fragments config files.

- meta-st/meta-st-stm32mp/recipes-kernel/linux/linux-stm32mp/<kernel version>/**fragment-*.config**

.config result is located in the build folder:

- build-openstlinuxweston-stm32mp1/tmp-glibc/work/stm32mp1-ostl-linux-gnueabi/linux-stm32mp/4.14-48/linux-stm32mp1-standard-build/**.config**

To modify the kernel options, it is not recommended to edit this file directly.

- A user runs either a text-mode :

```
   PC $> make config
starts a character based question and answer session (Figure 1)
```

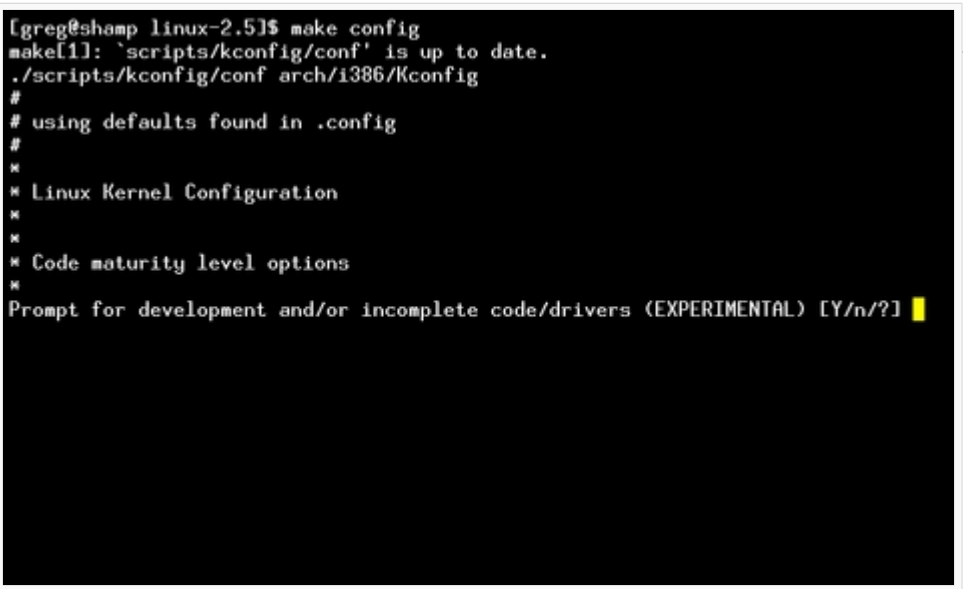


Figure 1. Configuring the kernel with make config

```
   PC $> make
menuconfig
starts a terminal-
oriented
configuration tool
(using ncurses)
(Figure 2)
The ncurses text
version is more
popular and is run
with the make
menuconfig option.
Wikipedia Menuconfig[1
] also explains how
to "navigate" within
the configuration
menu, and highlights
main key strokes.
```

- or a graphical kernel

configurator :

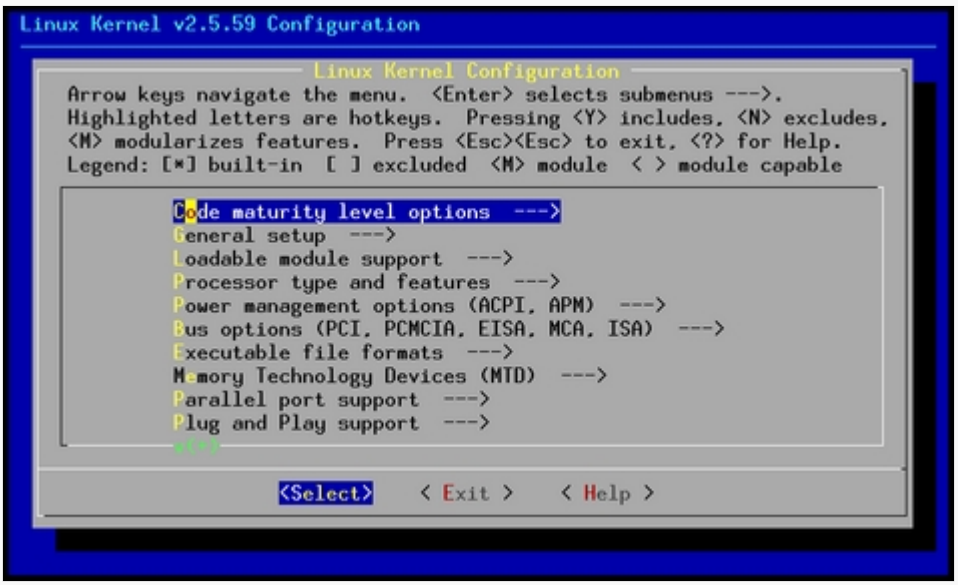


Figure 2. Make menuconfig makes it easier to back up and correct mistakes

PC $> make xconfig starts a X based configuration tool (Figure 3)

Ultimately these configuration tools edit the .config file.

An option indicates either some driver is built into the kernel ("=y") or will be built as a module ("=m") or is not selected.

The unselected state can either be indicated by a line starting with "#" (e.g. "# CONFIG_SCSI is not set") or by the absence of the relevant line from the .config file.

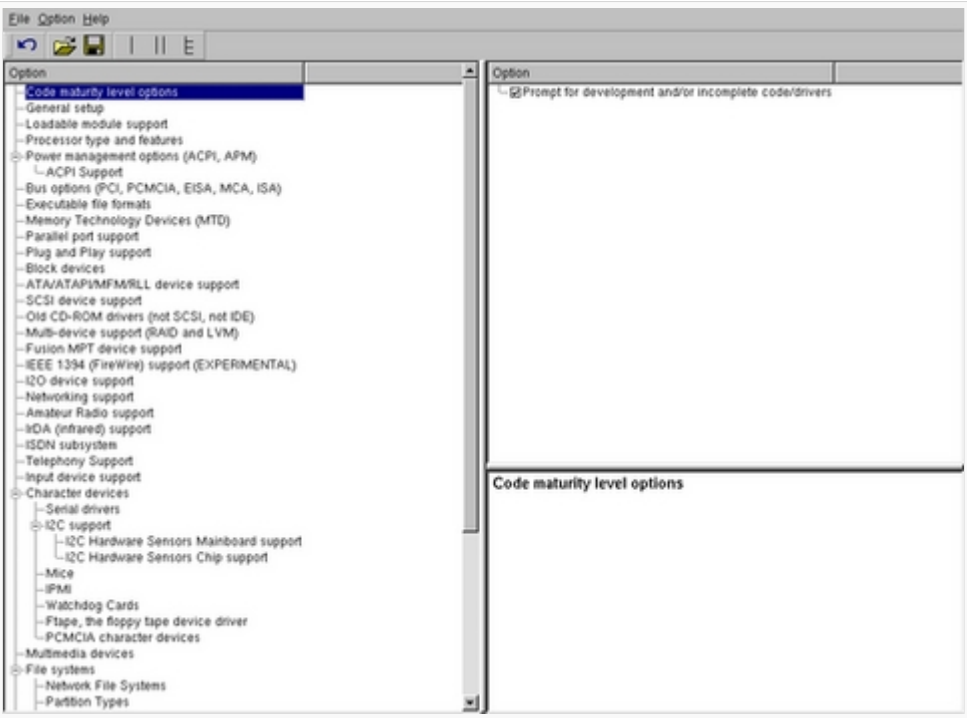


Figure 3. The Qt-Based make xconfig

The 3 states of the main selection option for the SCSI subsystem (which actually selects the SCSI mid level driver) follow. Only one of these should appear in an actual .config file:

```
CONFIG_SCSI=y
CONFIG_SCSI=m
# CONFIG_SCSI is not
set
```

2 Menuconfig and Developer Package

For this use case, the prerequesite is that OpenSTLinux SDK has been installed and configured.

To verify if your cross-compilation environment has been put in place correctly, run the following command:

```
PC $> set | grep CROSS
CROSS_COMPILE=arm-ostl-linux-gnueabi-
```

For more details, refer to *<Linux kernel installation directory>/README.HOW_TO.txt* helper file (the latest version of this helper file is also available in GitHub: README.HOW_TO.txt).

- Go to the <Linux kernel build directory>

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

- Save initial configuration (to identify later configuration updates)

```
PC $> make arch=ARM savedefconfig
Result is stored in defconfig file
PC $> cp defconfig defconfig.old
```

- Start the Linux kernel configuration menu

```
PC $> make arch=ARM menuconfig
```

- Navigate forwards or backwards directly between feature
  - un/select, modify feature(s) you want
  - When the configuration is OK : exit and save the new configuration

```
useful keys to know:
enter: enter in config subdirectory
space: hit several times to either select [*], select in module [m] or unselect [ ]
/: to search for a keyword, this is usefull to navigate in tree
?: to have more information on selected line
```

- Compare the old and new config files after operating modifications with menuconfig

```
PC $> make arch=ARM savedefconfig
```

Retrieve configuration updates by comparing the new defconfig and the old one

```
PC $> meld defconfig defconfig.old
```

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

```
PC $> make arch=ARM uImage LOADADDR=<loadaddr of kernel>
PC $> cp arch/arm/boot/uImage install_artifact/boot/
```

- Update the Linux kernel image on board

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

Information

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

- Reboot the board

```
Board $> cd /boot; sync; systemctl reboot
```

Note that this use case modifies the configuration file in the Linux kernel build directory, not in the Linux kernel source directory: this is a temporary modification useful for a prototyping.

- To make this temporary modification permanent, the delta between defconfig and defconfig.old must be saved in a configuration fragment file (fragment-*.config) based on fragment.cfg file, and the Linux kernel configuration/compilation steps must be re-executed (as explained in the README.HOW_TO.txt helper file).

3 Menuconfig and Distribution Package

- Start the Linux kernel configuration menu

```
PC $> bitbake virtual/kernel -c menuconfig
```

- Navigate forwards or backwards directly between feature
  - un/select, modify feature(s) you want
  - When the configuration is OK : exit and save the new configuration

```
useful keys to know:
enter: enter in config subdirectory
space: hit several times to either select [*], select in module [m] or unselect [ ]
/: to search for a keyword, this is usefull to navigate in tree
?: to have more information on selected line
```

- Cross-compile the Linux kernel

```
PC $> bitbake virtual/kernel
```

- Update the Linux kernel image on board

```
PC $> scp <build dir>/tmp-glibc/deploy/images/<machine name>/uImage root@<board ip
address>:/boot
```

Information

If the */boot* mounting point does not exist yet, please see how to create a mounting point

- Reboot the board

```
Board $> cd /boot; sync; systemctl reboot
```

Note that this use case modifies the configuration file in the Linux kernel build directory, not in the Linux kernel source directory: this is a temporary modification useful for a prototyping.

- To make this temporary modification permanent, it must be saved in a configuration fragment file (fragment-*.config) based on **fragment.cfg** file, and the Linux kernel configuration/compilation steps must be re-executed: **bitbake <name of kernel recipe>**.

4 References

- Wikipedia Menuconfig

Board support package

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

Stable: 13.05.2020 - 08:40 / Revision: 13.05.2020 - 08:39

A quality version of this page, approved on *13 May 2020*, was based off this revision.

Contents

1 Article purpose

This article explains how to configure the **RNG** internal peripheral when it is assigned to the Linux®OS. In that case, it is controlled by the Hardware random framework.

The configuration is performed using the device tree mechanism that provides a hardware description of the RNG peripheral, used by the STM32 RNG Linux driver.

If the peripheral is assigned to another execution context, refer to How to assign an internal peripheral to a runtime context article for guidelines on peripheral assignment and configuration.

2 DT bindings documentation

The *RNG* is represented by the *STM32 RNG device tree bindings*[1]

3 DT configuration

This hardware description is a combination of the **STM32 microprocessor** device tree files (*.dtsi* extension) and **board** device tree files (*.dts* extension). See the Device tree for an explanation of the device tree file split.

STM32CubeMX can be used to generate the board device tree. Refer to How to configure the DT using STM32CubeMX for more details.

3.1 DT configuration (STM32 level)

The RNG node is declared in stm32mp151.dtsi[2]. It describes the hardware register address, clock and reset.

```
 rng1: rng@54003000 {                                          Comments
         compatible = "st,stm32-rng";
         reg = <0x54003000 0x400>;                              --> Register location
and length
         clocks = <&scmi0_clk CK_SCMI0_RNG1>;
         resets = <&scmi0_reset RST_SCMI0_RNG1>;
         status = "disabled";
 };
```

Warning

This device tree part is related to STM32 microprocessors. It must be kept as is, without being modified by the end-user.

3.2 DT configuration (board level)

This part is used to enable the RNG used on a board which is done by setting the **status** property to **okay**.

A clock-error-detect property is available depending the clock choosen for entropy. It can be enabled to manage the clock detection.

3.3 DT configuration examples

```
&rng1 {
        status = "okay";
        clock-error-detect;
};
```

4 How to configure the DT using STM32CubeMX

The STM32CubeMX tool can be used to configure the STM32MPU device and get the corresponding platform configuration device tree files.
The STM32CubeMX may not support all the properties described in the above DT bindings documentation paragraph. If so, the tool inserts **user sections** in the generated device tree. These sections can then be edited to add some properties and they are preserved from one generation to another. Refer to STM32CubeMX user manual for further information.

5 References

Please refer to the following links for additional information:

- Device tree bindings
- STM32MP151 device tree

Linux® is a registered trademark of Linus Torvalds.

Operating System

Random Number Generator

Device Tree

Stable: 23.09.2020 - 13:22 / Revision: 12.06.2020 - 13:25

A quality version of this page, approved on *23 September 2020*, was based off this revision.

1 STM32CubeMX overview

This article describes STM32CubeMX, an official STMicroelectronics graphical software configuration tool.

The STM32CubeMX application helps developers to use the STM32 by means of a user interface, and guides the user through to the initial configuration of a firmware project.

It provides the means to:

- configure pin assignments, the clock tree, or internal peripherals
- simulate the power consumption of the resulting project
- configure and tune DDR parameters
- generate HAL initialization code for Cortex-M4
- generate the Device Tree for a Linux kernel, TF-A and U-Boot firmware for Cortex-A7

It uses a rich library of data from the STM32 microcontroller portfolio.

The application is intended to ease the initial development phase by helping developers to select the best product in terms of features and power.

2 STM32CubeMX main features

- Peripheral and middleware parameters

  Presents options specific to each supported software component

- Peripheral assignment to processors

  Allows assignment of each peripheral to Cortex-A Secure, Cortex-A Non-Secure, or Cortex-M processors

- Power consumption calculator

  Uses a database of typical values to estimate power consumption, DMIPS, and battery life

- Code generation

  Makes code regeneration possible, while keeping user code intact

- Pinout configuration

  Enables peripherals to be chosen for use, and assigns GPIO and alternate functions to pins

- Clock tree initialization

  Chooses the oscillator and sets the PLL and clock dividers

- DDR tunning tool

  Ensures the configuration, testing, and tuning of the MPU DDR parameters. Using U-Boot-SPL Embedded Software.

3 How to get STM32CubeMX

Please, refer to the following link STM32CubeMX to find STM32CubeMX, the Release Note, the User Manual and the product specification.

Doubledata rate (memory domain)

Hardware Abstraction Layer

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Trusted Firmware for Arm Cortex-A

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

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

Microprocessor Unit