Reset device tree configuration

Contents

1. Reset device tree configuration 3
2. Device tree 10
3. Non secure RCC configuration 15
4. RCC internal peripheral 15
5. Reset overview 21
6. STM32CubeMX 28

Stable: 17.07.2020 - 12:55 / Revision: 29.06.2020 - 08:00

Stable: 17.07.2020 - 12:39 / Revision: 29.06.2020 - 08:00

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

Contents

1 Article purpose 4
2 Reset controller providers 5
3 DT bindings documentation 6
4 DT configuration 7
4.1 DT configuration (STM32 level) 7
4.1.1 STM32MP1 RCC Reset node 7
4.1.2 SCMI Reset Domain node 7
4.2 DT configuration (board level) 8
5 How to configure the DT using STM32CubeMX 9
6 References 10

1 Article purpose

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

The configuration is performed using the device tree mechanism that provides a hardware description of the RCC peripheral used by the reset-stm32mp1 Linux driver and by the Reset framework.

When RCC TZEN or MCKPROT security hardening is enabled, Linux RCC reset driver is not the only reset controller provider. The SCMI server embedded in the secure firmware is also a reset controller provider through Linux SCMI reset device driver. See more information in section STM32MP15 SCMI reset domains below.

2 Reset controller providers

There are 2 reset controller providers in STM32MP1. Each are represented by node(s) in the device tree description. Their node must define a value for specifier *#reset-cells*.

- STM32MP1 RCC reset controllers, most of the system reset controllers actually.
- SCMI reset domains are reset controllers registered by the SCMI drivers.

The STM32MP1 uses SCMI reset domains to abstract RCC secure reset controllers.

This article describes the device configuration where RCC TZEN security hardening is enabled. Specificities to explore when RCC TZEN hardening is disabled are discussed in non-secure RCC configuration article.

3 DT bindings documentation

The Reset device tree bindings are composed of:

- generic DT bindings[1] used by the Reset framework.
- vendor Reset DT bindings[2] used by the reset-stm32mp1 driver: this binding document explains how to write device tree files for reset.
- generic SCMI DT bindings[3] used by the SCMI reset domain protocol support.

4 DT configuration

4.1 DT configuration (STM32 level)

The device tree description of the STM32 SoC includes reset controllers exposed by RCC reset controller device driver and SCMI reset domain device driver.

4.1.1 STM32MP1 RCC Reset node

The device tree defines the RCC reset controllers device as a node with *compatible = "st,stm32mp1-rcc"* or *"st,stm32mp1-rcc-secure"* node.

- *"st,stm32mp1-rcc-secure"* complies with configuration where RCC TZEN secure hardening is enabled.
- *"st,stm32mp1-rcc"* complies with configuration where RCC TZEN secure hardening is disabled.

The node defines *#reset-cells = <1>;*.
STM32MP1 RCC Reset controllers are identified by a single 32-bit ID. Valid values for the IDs are defined in STM32MP1 Reset DT bindings[4].

The STM32MP1 RCC Reset node is the same node as the STM32MP1 RCC Clock (they share same hardware IP) and is located in the *stm32mp151.dtsi*[5], hence one can also see *#clock-cells = <1>;* in the node. See the Device tree for further explanation.

```
rcc: rcc@50000000 {
       compatible = "st,stm32mp1-rcc-secure", "st,stm32mp1-rcc", "syscon";
        #clock-cells = <1>;
        #reset-cells = <1>;
        reg = <0x50000000 0x1000>;
        ...
};
```

This device tree part is related to STM32MP1 microprocessors. It must be kept as-is, without being modified by the end-user.

4.1.2 SCMI Reset Domain node

The device tree defines SCMI reset domains using *compatible = "arm,scmi"* nodes with subnodes specifying protocol@16 (*reg = <0x16>*) together with specifier '#reset-cells = 1. *The reset consumer uses node phandle (*scmi0_reset *in example below) together with a reset ID based macros* RST_SCMIx_* *defined in STM32MP1 Reset DT bindings*[6] *to identify a SCMI reset domain related to an agent interface.*

```
scmi-0 {
        compatible = "arm,scmi";
        #address-cells = <1>;
        #size-cells = <0>;

        scmi0_reset: protocol@16 {
               reg = <0x16>;
               #reset-cells = <1>;
       };
};
```

This device tree part is related to STM32MP1 microprocessors. It must be kept as-is, without being modified by the end-user.

4.2 DT configuration (board level)

If a Linux driver needs a reset signal, it should be declared in its DT node as shown below:
resets = <phandle> : List of phandle and reset specifier pairs, one pair for each reset signal that affects the device, or that the device manages.

- Example:

```
i2c2: i2c@40013000 {
        compatible = "st,stm32f7-i2c";
        reg = <0x40013000 0x400>;
        clocks = <&rcc I2C2_K>;
        resets = <&rcc I2C2_R>;
       ...
};

i2c4: i2c@5c002000 {
        compatible = "st,stm32f7-i2c";
        reg = <0x5c002000 0x400>;
        clocks = <&scmi0_clk CK_SCMI0_I2C4>;
        resets = <&scmi0_reset RST_SCMI0_I2C4>;
       ...
};
```

5 How to configure the DT using STM32CubeMX

The STM32CubeMX tool can be used to configure the STM32MP1 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, which are preserved from one generation to another. Refer to STM32CubeMX user manual for further information.

6 References

Please refer to the following links for additional information:

- Documentation/devicetree/bindings/reset/reset.txt , Reset device tree bindings
- Documentation/devicetree/bindings/reset/st%2Cstm32mp1-rcc.txt , STM32MP1 Reset device tree bindings
- Documentation/devicetree/bindings/arm/arm,scmi.txt SCMI DT bindings
- Documentation/devicetree/bindings/reset/st%2Cstm32mp1-rcc.txt STM32MP1 Reset DT bindings
- stm32mp151.dtsi STM32MP151 device tree file
- Documentation/devicetree/bindings/reset/st%2Cstm32mp1-rcc.txt STM32MP1 Reset DT bindings

Operating System

Reset and Clock Control

System control and management interface

Device Tree

Stable: 19.03.2021 - 08:52 / Revision: 19.03.2021 - 08:49

A quality version of this page, approved on *19 March 2021*, was based off this revision.

Contents

1 Purpose 11
1.1 Source files 11
1.2 Bindings 11
1.3 Build 11
1.4 Tools 12
2 STM32 13
3 How to go further 14
4 References 15

1 Purpose

The objective of this chapter is to give general information about the device tree.
An extract of the **device tree specification**[1] explains it as follows:

"A device tree is a tree data structure with nodes that describe the devices in a system. Each node has property/value pairs that describe the characteristics of the device being represented. Each node has exactly one parent except for the root node, which has no parent. ... Rather than hard coding every detail of a device into an operating system, many aspect of the hardware can be described in a data structure that is passed to the operating system at boot time."

In other words, a device tree describes the hardware that can not be located by probing. For more information, please refer to the device tree specification[1]

1.1 Source files

- **.dts:** The device tree source (DTS). This format is a textual representation of a device tree in a form that can be processed by DTC (Device Tree Compiler) into a binary device tree in the form expected by software components: Linux® Kernel, U-Boot and TF-A.

- **.dtsi:** Source files that can be included from a DTS file.

1.2 Bindings

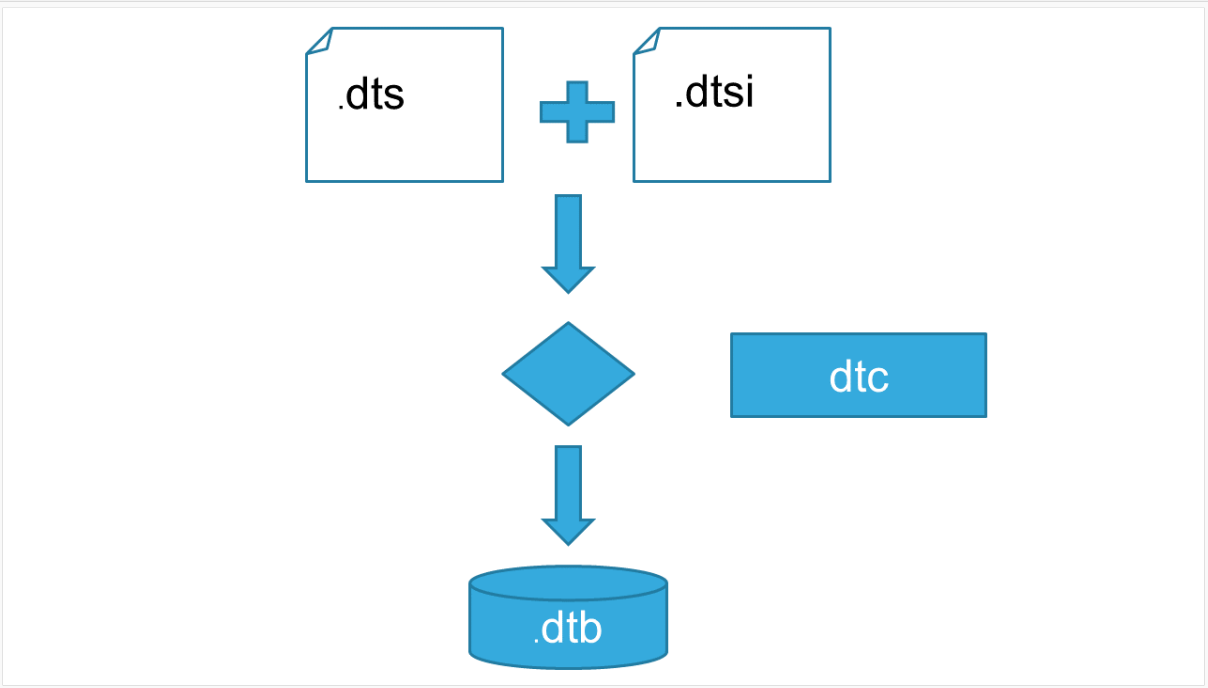
The device tree data structures and properties are named **bindings**. Those bindings are described in:

- The Device tree specification[1] for generic bindings.
- The software component documentations:
  - Linux® Kernel: Linux kernel device tree bindings
  - U-Boot: doc/device-tree-bindings/
  - TF-A: TF-A device tree bindings

1.3 Build

- A tool named DTC (Device Tree Compiler) allows compiling the DTS sources into a binary.
- input file: the *.dts* file described in section above.
- output file: the *.dtb* file described in section above.
- More information are available in DTC manual[2].

- DTC source code is located here[3]. DTC tool is also available directly in particular software components: **Linux Kernel, U-Boot, TF-A ...**. For those components, the device tree building is directly integrated in the component build process.



Device tree build process

If dts files use some defines, dts files should be preprocessed before being compiled by DTC.

1.4 Tools

The device tree compiler offers also some tools:

- **fdtdump:** Print a readable version of a flattened device tree file (dtb)
- **fdtget:** Read properties from a device tree
- **fdtput:** Write properties to a device tree
- ...

There are several ways to get those tools:

- In the device tree compiler project source code[3]
- Directly in software components: **Kernel, u-boot, tf-a ...**
- Available in Debian package[4]

2 STM32

For STM32MP1, the device tree is used by three software components: Linux® kernel, U-Boot and TF-A.

The device tree is part of the OpenSTLinux distribution. It can also be generated by STM32CubeMX tool.

To have more information about the device tree usage on STM32MP1 (how the device tree source files are split, how to find the device tree source files per software components, how is STM32CubeMX generating the device tree ...) see STM32MP15 device tree page.

3 How to go further

- Device Tree for STM32MP [5]
- Device Tree Reference[6] - eLinux.org
- Device Tree usage[7] - eLinux.org

4 References

- 1.01.11.2 https://github.com/devicetree-org/devicetree-specification/releases/tag/v0.2(latest) ,Device tree specification
- https://git.kernel.org/pub/scm/utils/dtc/dtc.git/tree/Documentation/manual.txt(master) ,DTC manual
- 3.03.1 https://git.kernel.org/pub/scm/utils/dtc/dtc.git(master) ,DTC source code
- https://packages.debian.org/search?keywords=device-tree-compiler(master) ,DTC debian package
- https://www.youtube.com/watch?v=a9CZ1Uk3OYQ, Device Tree for STM32MP
- Device Tree Reference, eLinux.org
- Device Tree Usage, eLinux.org

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

Linux® is a registered trademark of Linus Torvalds.

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

Trusted Firmware for Arm® Cortex®-A

Stable: Not stable / Revision: 31.03.2021 - 07:05

Invalid target: no reviewed revision corresponds to the given ID.

Return to Non secure RCC configuration.

Stable: 25.09.2020 - 09:10 / Revision: 25.09.2020 - 09:09

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

Contents

1 Article purpose 16
2 Peripheral overview 17
2.1 Features 17
2.2 Security support 17
3 Peripheral usage and associated software 18
3.1 Boot time 18
3.2 Runtime 18
3.2.1 Overview 18
3.2.2 Software frameworks 18
3.2.3 Peripheral configuration 19
3.2.4 Peripheral assignment 19
4 How to go further 20
5 References 21

1 Article purpose

The purpose of this article is to:

- briefly introduce the RCC peripheral and its main features
- indicate the level of security supported by this hardware block
- explain, when necessary, how to configure the RCC peripheral.

2 Peripheral overview

The **RCC** peripheral is used to control the internal peripherals, as well as the **reset** signals and **clock** distribution. The RCC gets several internal (LSI, HSI and CSI) and external (LSE and HSE) clocks. They are used as clock sources for the hardware blocks, either directly or indirectly, via the four PLLs (PLL1, PLL2, PLL3 and PLL4) that allow to achieve high frequencies.

2.1 Features

Refer to the STM32MP15 reference manuals for the complete list of features, and to the software components, introduced below, to see which features are really implemented.

2.2 Security support

The RCC is a **secure** peripheral. There are two levels of security, which are controlled via two bits in the RCC_TZCR register (only accessible in secure mode):

- **TZEN** allows to set some RCC registers in secure mode, in particular registers for configuring PLL1 and PLL2, in order to secure a TrustZone perimeter for the Cortex®-A7 secure core and its peripherals.
- **MCKPROT** allows extending the TZEN secure clock control perimeter to PLL3 and to the MCU subsystem, so to the Cortex® -M4 and its bus clock.

Please note that all RCC registers can be read from the non-secure world.

3 Peripheral usage and associated software

3.1 Boot time

The RCC security level differs for each boot chain:

- the trusted boot chain sets TZEN to 1 an MCKPROT to 0
- the basic boot chain sets TZEN to 0 an MCKPROT to 0

The RCC is used by all the boot components: the ROM code, the FSBL, the SSBL and up to the Linux® kernel. Nevertheless, the main initialization step is performed by the FSBL that is responsible for the clock tree initialization: it consists in configuring all the input clocks, the PLL and the clock sources that are selected as kernel clocks for all peripherals. The whole configuration is carried out by the device tree.

The STM32CubeMX tool allows configuring in one place the clock tree that will be applied at boot time and used at runtime, so it is highly recommended to use it to generate your device tree. Moreover, the STM32CubeMX integrates all the information documented in the STM32MP15 reference manuals, making this configuration step straighforward.

3.2 Runtime

3.2.1 Overview

The RCC peripheral is shared at runtime:

- the Arm®Cortex®-A7 secure core controls all the secure registers (refer to TZEN and MCKPROT bit descriptions) through the RCC OP-TEE driver. The access to some secure registers from the Cortex®-A7 non-secure core can be achieved via runtime secure services implemented in the secure monitor (from the OP-TEE if it is present, otherwise from the TF-A).
- the Arm®Cortex®-A7 non-secure core controls the clock management via the clock framework, and the reset management via the reset framework in Linux®.
- the Arm®Cortex®-M4 core controls all the clock and reset managements in STM32Cube with the RCC HAL driver

Concurrent control from each context is possible because the above managements are performed via independent registers.

3.2.2 Software frameworks

Domain	Peripheral	Software components			Comment
OP-TEE	Linux	STM32Cube			
Power & Thermal	RCC	OP-TEE RCC driver	Reset framework Clock framework	STM32Cube RCC driver	

3.2.3 Peripheral configuration

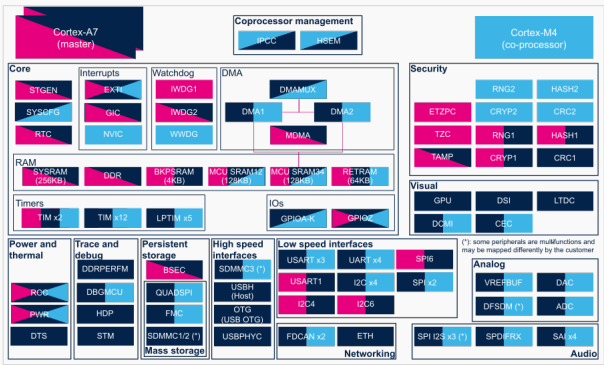
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration can be done alone via the STM32CubeMX tool for all internal peripherals, and then manually completed (particularly for external peripherals), according to the information given in the corresponding software framework article.

3.2.4 Peripheral assignment

Check boxes illustrate the possible peripheral allocations supported by STM32 MPU Embedded Software:

- means that the peripheral can be assigned () to the given runtime context.
- is used for system peripherals that cannot be unchecked because they are statically connected in the device.

Refer to How to assign an internal peripheral to a runtime context for more information on how to assign peripherals manually or via STM32CubeMX.
The present chapter describes STMicroelectronics recommendations or choice of implementation. Additional possiblities might be described in STM32MP15 reference manuals
.



Internal peripherals

Domain	Peripheral	Runtime allocation				Comment
Instance	Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Power & Thermal	RCC	RCC				

4 How to go further

The RCC is interfaced with the HDP internal peripheral, thus offering the flexibility to monitor the main RCC state signals on the debug pins.

Please refer to the STM32MP15 reference manuals for the full list of signals that can be monitored.

5 References

Reset and Clock Control

Low Speed Internal oscillator (STM32 clock source)

High Speed Internal oscillator (STM32 clock source) or High Speed Synchronous Serial Interface (MIPI® Alliance standard)

Multi Speed Internal oscillator (STM32 clock source)

Low Speed External oscillator (STM32 clock source)

High Speed External oscillator (STM32 clock source)

TrustZone®

Arm® and TrustZone® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Cortex®

Microcontroller Unit (MCUs have internal flash memory and are intended to operate with a minimum amount of external support ICs. They commonly are a self-contained, system-on-chip (SoC) designs.)

Read Only Memory

First Stage Boot Loader

Second Stage Boot Loader

Linux® is a registered trademark of Linus Torvalds.

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Open Portable Trusted Execution Environment

Stable: 15.10.2019 - 14:08 / Revision: 15.10.2019 - 14:08

A quality version of this page, approved on *15 October 2019*, was based off this revision.

The Linux® kernel reset framework offers a generic API for reset lines management at platform level.

Contents

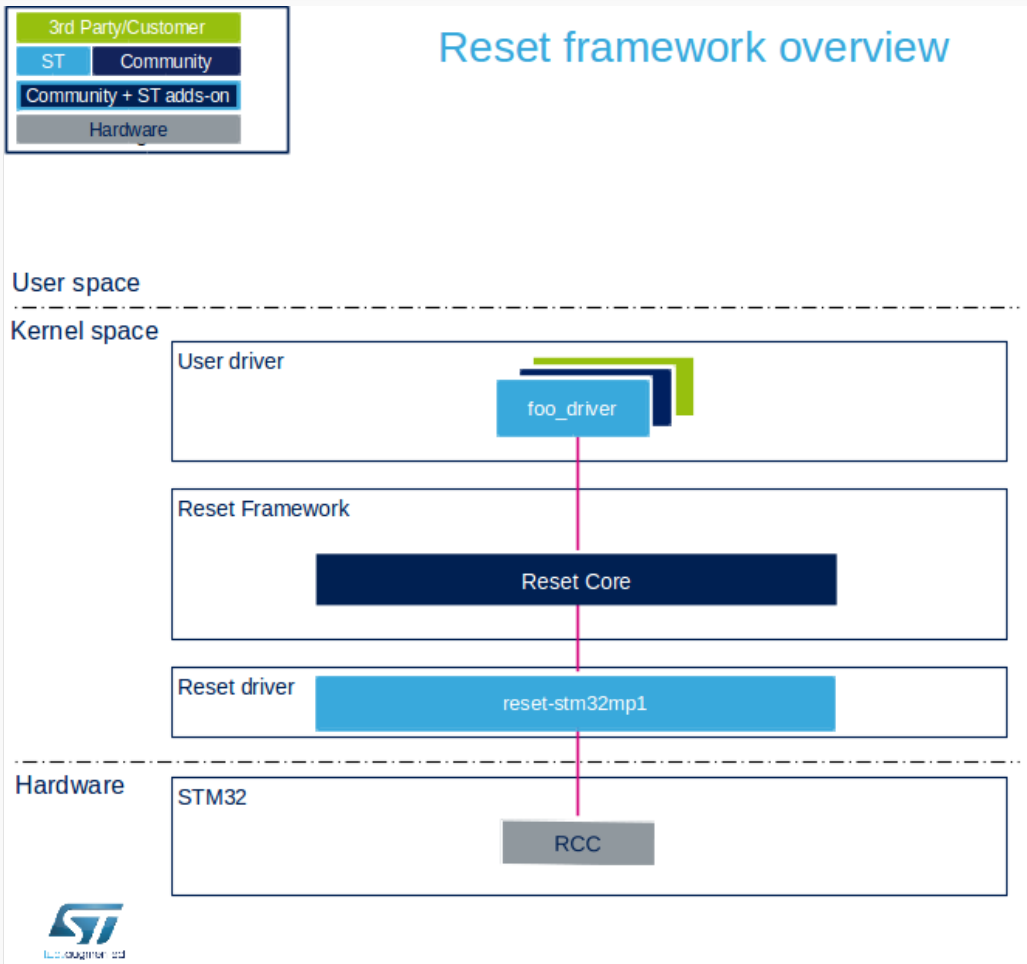
1 Framework purpose 22
2 System overview 23
2.1 Component description 23
2.2 API description 23
3 Configuration 25
3.1 Kernel configuration 25
3.2 Device tree configuration 25
4 How to use the Reset framework 26
5 Source code location 27
6 References 28

1 Framework purpose

The purpose of this article is to introduce the Linux kernel Reset Framework. It provides general information and, based on concrete IP use cases, explains how to use it.

The Linux Reset framework offers a generic API for abstracting and controlling the different reset lines of the platform. Reset lines can be internal system reset signals or external lines such as a GPIO to reset an external device.

2 System overview



Reset Overview

2.1 Component description

- **reset core:** kernel reset framework designed for abstracting device reset handling via GPIOs or internal reset controller modules.
- **reset-stm32mp1:** STM32MP1 RCC specific reset driver
- **RCC:** Reset and clock controller that generates the different internal reset lines on STM32MP1.

2.2 API description

Documentation on the reset interface can be found in kernel documentation folder: https://www.kernel.org/doc/Documentation/devicetree/bindings/reset/reset.txt

- Kernel reset API

To get the line reset from DT:

- **devm_reset_control_get_exclusive:** looks up and obtains an exclusive reference to a reset controller (cannot be shared with another driver).
- **devm_reset_control_get_shared:** looks up and obtains a shared reference to reset a controller (can be shared with another driver).
- **devm_reset_control_get:** same as devm_reset_control_get_exclusive.
- **devm_reset_control_get_optional:** same as devm_reset_control_get except that the reference name is optional.
- **devm_reset_control_get_optional_exclusive:** same as devm_reset_control_get_exclusive except that the reference name is optional.
- **devm_reset_control_get_optional_shared:** same as devm_reset_control_get_shared except that the reference name is optional.

To configure the reset line and retrieve its status from DT:

- **reset_control_assert:** asserts the reset line
- **reset_control_deassert:** deasserts the reset line
- **reset_control_status:** returns if the line is asserted or not

3 Configuration

3.1 Kernel configuration

By default, the activation of the Reset framework configuration (CONFIG_RESET_CONTROLER) is selected in the kernel configuration, through the Linux Menuconfig tool: Menuconfig or how to configure kernel (CONFIG_ARCH_STM32=y, CONFIG_ARCH_MULTI_V7=y):

3.2 Device tree configuration

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

4 How to use the Reset framework

To control a peripheral reset line, a driver must first get a reference on the reset handler thanks to its device tree node. The driver can then assert and deassert the reset line by calling the reset framework API.

Below an example:

```
foo: foo@adcdefgh {
       compatible = "foo-driver";
       ...
        resets = <&rcc FOO_R>;
       ...
};
```

Before using the reset specified in the device tree node, the foo driver has to request it at probe execution.

```
static int foo_probe(struct platform_device *pdev)
{
        struct reset_control *rst;
        ...
        rst = devm_reset_control_get_exclusive(&pdev->dev, NULL);
        if (IS_ERR(rst)) {
                ret = PTR_ERR(rest);
                dev_err(&pdev->dev, "reset get failed: %d\n", ret);
                goto err_master_put;
        }
        ...
```

The reset can then be asserted or deasserted using the reset framework API:

```
        ...
        reset_control_assert(rst);
        udelay(2);
        reset_control_deassert(rst);
        ...
```

5 Source code location

Reset core file [1]

STM32MP1 reset file [2]

STM32MP1 reset interface file [3]

6 References

- drivers/reset/core.c
- drivers/reset/reset-stm32mp1.c
- include/dt-bindings/reset/stm32mp1-resets.h

Linux® is a registered trademark of Linus Torvalds.

Application programming interface

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.)

Reset and Clock Control

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