Category:Trusted Firmware-A (TF-A)

Contents

1. Category:Trusted Firmware-A (TF-A) 3
2. Clock device tree configuration - Bootloader specific 4
3. STM32MP15 TF-A 17
4. TF-A - Flash memory configuration 27
5. TF-A - How to debug 38
6. TF-A overview 42

Stable: 17.06.2020 - 15:27 / Revision: 16.01.2020 - 13:10

Category:Trusted Firmware-A (TF-A)

Stable: 17:06:2020 - 10:27 / Revision: 16:01:2020 - 13:10

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

This category groups together all articles related to software components managing the Trusted Firmware-A (TF-A) (with "A" meaning Arm®Cortex®-A).

Trusted Firmware for Arm Cortex-A

Pages in category "Trusted Firmware-A (TF-A)"

The following 5 pages are in this category, out of 5 total.

- TF-A overview
- STM32MP15 TF-A
- Clock device tree configuration - Bootloader specific
- TF-A - Flash memory configuration
- TF-A - How to debug

Stable: 09.12.2020 - 13:13 / Revision: 07.12.2020 - 12:45

A quality version of this page, approved on *9 December 2020*, was based off this revision.

Contents

1 Article purpose 5
2 DT bindings documentation 6
3 DT configuration 7
3.1 DT configuration (STM32 level) 7
3.2 DT configuration (board level) 7
3.2.1 Clock node 7
3.2.1.1 Optional properties for "clk-lse" and "clk-hse" external oscillators 8
3.2.1.2 DT configuration for HSE 8
3.2.1.3 DT configuration for LSE 9
3.2.1.4 Optional property for "clk-hsi" internal oscillator 9
3.2.1.5 Clock node example 10
3.2.2 STM32MP1 clock node 11
3.2.2.1 Defining clock source distribution with st,clksrc property 11
3.2.2.2 Defining clock dividers with st,clkdiv property 12
3.2.2.3 Defining peripheral PLL frequencies with st,pll property 12
3.2.2.4 Defining peripheral kernel clock tree distribution with st,pkcs property 13
3.2.2.5 HSI and CSI clocks calibration 14
4 How to configure the DT using STM32CubeMX 15
5 References 16

1 Article purpose

This article describes the specific **RCC** internal peripheral configuration done by the first stage bootloader:

- TF-A for the Trusted boot chain
- U-Boot SPL DDR interactive mode for the DDR tuning tool

Regarding OP-TEE when it is embedded in the device, OP-TEE OS is booted by TF-A BL2, it is booted by TF-A BL2 bootstage. OP-TEE relies on TF-A BL2 bootstage for the RCC clock tree initial configuration. This article explicitily mentions OP-TEE when in information applies to OP-TEE secure world configuration.

Warning

This article explains how to configure the clock tree in the RCC at boot time.
You can then refer to the clock device tree configuration article to understand how to derive each internal peripheral clock tree in Linux®OS from the RCC clock tree.

The configuration is performed using the device tree mechanism that provides a hardware description of the RCC peripheral.

This clock tree is only used in the device tree of the boot chain FSBL; so in the TF-A device tree for OpenSTLinux official delivery (or in SPL only for the DDR tuning tool).

Even if the clock tree information is also present in the U-Boot device tree, it is not used during boot by this SSBL.

2 DT bindings documentation

The bootloader clock device tree bindings correspond to the vendor clock DT bindings used by the clk-stm32mp1 driver of the FSBL (TF-A or U-Boot SPL for DDR interactive mode), it is based on:

- binding described in Clock_device_tree_configuration
- bootloader specific properties described in #DT configuration

This binding document explains how to write the device tree files for clocks on the bootloader side:

- TF-A: *tf-a/docs/devicetree/bindings/clock/st,stm32mp1-rcc.txt*"[1]
- U-Boot SPL for DDR interactive mode: *doc/device-tree-bindings/clock/st,stm32mp1.txt*[2]

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 STM32MP1 clock nodes are located in *stm32mp151.dtsi*[3] (see Device tree for more explanations):

- fixed-clock defined in clock node
- RCC node for #STM32MP1 clock node: clock generation and distribution.

```
/ {
...
        clocks {
                clk_hse: clk-hse {
                        #clock-cells = <0>;
                        compatible = "fixed-clock";
                        clock-frequency = <24000000>;
                };

...
          };
...
        soc {
...
                rcc: rcc@50000000 {
                        compatible = "st,stm32mp1-rcc", "syscon";
                        reg = <0x50000000 0x1000>;
                        #clock-cells = <1>;
                        #reset-cells = <1>;
                        interrupts = <GIC_SPI 5 IRQ_TYPE_LEVEL_HIGH>;
                };
...

        };

};
```

Please refer to clock device tree configuration for the bindings common with Linux® kernel.

3.2 DT configuration (board level)

3.2.1 Clock node

Note: this section applies to OP-TEE that gets input clocks frequency value from the device tree description it boots upon.

The clock tree is also based on five fixed clocks in the clock node. They are used to define the state of associated STM32MP1 oscillators:

- clk-lsi
- clk-lse

- clk-hsi
- clk-hse
- clk-csi

Please refer to clock device tree configuration for detailed information.

At boot time, the clock tree initialization performs the following tasks:

- enabling of the oscillators present in the device tree and not disabled (node with status="disabled"),
- disabling of the HSI oscillator if the node is absent or disabled (HSI is always activated by the ROM code).

3.2.1.1 *Optional properties for "clk-lse" and "clk-hse" external oscillators*

For external oscillator HSE and LSE, the default clock configuration is an external crystal/ceramic resonator.

Four optional fields are supported:

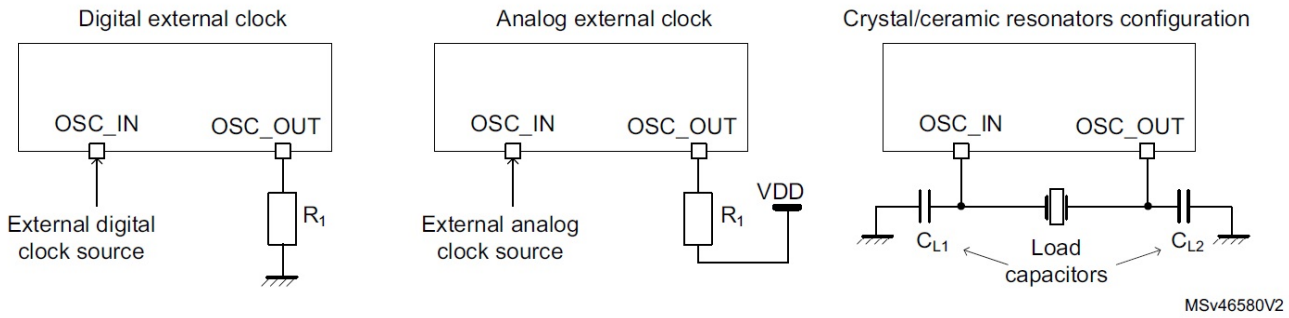
- "st,bypass" configures the external analog clock source (set HSEBYP, LSEBYP),
- "st,digbypass" configures the external digital clock source (set DIGBYP and HSEBYP, LSEBYP),
- "st,css" activates the clock security system (HSECSSON, LSECSSON),
- "st,drive" (LSE only) contains the value of the drive for the oscillator (see LSEDRV_ defined in the file *stm32mp1-clksrc.h*[4]).

3.2.1.2 *DT configuration for HSE*

The HSE can accept an external crystal/ceramic or external clock source on OSC_IN, digital or analog : the user needs to select the correct frequency and the correct configuration in the device tree, corresponding to the hardware setup.

All the ST boards are using a digital external clock configuration (so device tree with = st,digbypass).

For example with the same 24MHz frequency, we have 3 configurations:



- Digital external clock = st,digbypass

```
        / {
                clocks {
                        clk_hse: clk-hse {
                                #clock-cells = <0>;
                                compatible = "fixed-clock";
                                clock-frequency = <24000000>;
                                st,digbypass;
                        };
        };
```

- Analog external clock = st,bypass

```
        / {
                clocks {
                        clk_hse: clk-hse {
                                #clock-cells = <0>;
```

```
                                compatible = "fixed-clock";
                                clock-frequency = <24000000>;
                                st,bypass;
                        };
        };
```

- Crystal/ ceramic resonators configuration

```
        / {
                clocks {
                        clk_hse: clk-hse {
                                #clock-cells = <0>;
                                compatible = "fixed-clock";
                                clock-frequency = <24000000>;
                        };
        };
```

3.2.1.3 DT configuration for LSE

Below an example of LSE on board file with 32,768kHz crystal resonators, the drive set to medium high and with activated clock security system.

```
        / {
                clocks {
                        clk_lse: clk-lse {
                                #clock-cells = <0>;
                                compatible = "fixed-clock";
                                clock-frequency = <32768>;
                                st,css;
                                st,drive = <LSEDRV_MEDIUM_HIGH>;
                        };
        };
```

3.2.1.4 Optional property for "clk-hsi" internal oscillator

The HSI clock frequency is internally fixed to 64 MHz for the STM32MP15 devices.

In the device tree, clk-hsi is the clock after HSIDIV divider (more information on clk_hsi can be found in the RCC chapter in the reference manual).
As a result the frequency of this fixed clock is used to compute the expected HSIDIV for the clock tree initialization.

Below an example with HSIDIV = 1/1:

```
        / {
                clocks {
                        clk_hsi: clk-hsi {
                                #clock-cells = <0>;
                                compatible = "fixed-clock";
                                clock-frequency = <64000000>;
                        };
        };
```

Below an example with HSIDIV = 1/2:

```
        / {
                clocks {
                        clk_hsi: clk-hsi {
                                #clock-cells = <0>;
                                compatible = "fixed-clock";
                                clock-frequency = <32000000>;
                        };
        };
```

3.2.1.5 *Clock node example*

Note: this section applies to OP-TEE OS clock drivers.

An example of clocks node with:

- all oscillators switched on (HSE, HSI, LSE, LSI, CSI)
- HSI at 64MHZ (HSIDIV = 1/1)
- HSE using a digital external clock at 24MHz
- LSE using an external crystal at 32.768kHz (the typical frequency)

We highlight the customized parts:

```
/ {
        clocks {
                clk_hse: clk-hse {
                        #clock-cells = <0>;
                        compatible = "fixed-clock";
                        clock-frequency = <24000000>;
                        st,digbypass;
                };

                clk_hsi: clk-hsi {
                        #clock-cells = <0>;
                        compatible = "fixed-clock";
                        clock-frequency = <64000000>;
                };

                clk_lse: clk-lse {
                        #clock-cells = <0>;
                        compatible = "fixed-clock";
                        clock-frequency = <32768>;
                };

                clk_lsi: clk-lsi {
                        #clock-cells = <0>;
                        compatible = "fixed-clock";
                        clock-frequency = <32000>;
                };

                clk_csi: clk-csi {
                        #clock-cells = <0>;
                        compatible = "fixed-clock";
                        clock-frequency = <4000000>;
                };
        };
};
```

So the resulting board device tree, based on SoC device tree "stm32mp151.dtsi", is :

```
#include "stm32mp151.dtsi"
&clk_hse {
        clock-frequency = <24000000>;
        st,digbypass;
};

&clk_hsi {
        clock-frequency = <64000000>;
};

&clk_lse {
        clock-frequency = <32768>;
};
```

It is the configuration used by TF-A for STM32MP15 boards.

3.2.2 STM32MP1 clock node

Please refer to clock device tree configuration for information on how to specify the number of cells in a clock specifier.

The bootloader performs a global clock initialization, as described below. The information related to a given board can be found in the board specific device tree files listed in clock node.

The bootloader uses other properties for RCC node ("st,stm32mp1-rcc" compatible):

- secure-status: related to TZEN bit configuration in RCC security register that allows to restrict RCC and PWR registers write access
- st,clksrc: clock source configuration array
- st,clkdiv: clock divider configuration array
- st,pll: specific PLL configuration
- st,pkcs: peripheral kernel clock distribution configuration array.

All the available clocks are defined as preprocessor macros in *stm32mp1-clks.h*[5] and can be used in device tree sources.

Note: this section partially applies to OP-TEE OS clock drivers in that OP-TEE OS clock drivers consider only property *secure-status* over those listed above.

3.2.2.1 Defining clock source distribution with st,clksrc property

This property can be used to configure the clock distribution tree. When used, it must describe the whole distribution tree.

There are nine clock source selectors for the STM32MP15 devices. They must be configured in the following order: MPU, AXI, MCU, PLL12, PLL3, PLL4, RTC, MCO1, and MCO2.

The clock source configuration values are defined by the CLK_<NAME>_<SOURCE> macros located in *stm32mp1-clksrc.h*[4].

Example:

```
        st,clksrc = <
                CLK_MPU_PLL1P
                CLK_AXI_PLL2P
                CLK_MCU_PLL3P
                CLK_PLL12_HSE
                CLK_PLL3_HSE
                CLK_PLL4_HSE
                CLK_RTC_LSE
                CLK_MCO1_DISABLED
                CLK_MCO2_DISABLED
        >;
```

3.2.2.2 *Defining clock dividers with st,clkdiv property*

This property can be used to configure the value of the clock main dividers. When used, it must describe the whole clock divider tree.

There are 11 dividers values for the STM32MP15 devices. They must be configured in the following order: MPU, AXI, MCU, APB1, APB2, APB3, APB4, APB5, RTC, MCO1 and MCO2.

Each divider value uses the DIV coding defined in the RCC associated register, RCC_xxxDIVR. In most cases, this value is the following:

- 0x0: not divided
- 0x1: division by 2
- 0x2: division by 4
- 0x3: division by 8
- ...

Note that the coding differs for RTC MCO1 and MCO2:

- 0x0: not divided
- 0x1: division by 2
- 0x2: division by 3
- 0x3: division by 4
- ...

Example:

```
        st,clkdiv = <
                1 /*MPU*/
                0 /*AXI*/
                0 /*MCU*/
                1 /*APB1*/
                1 /*APB2*/
                1 /*APB3*/
                1 /*APB4*/
                2 /*APB5*/
                23 /*RTC*/
                0 /*MCO1*/
                0 /*MCO2*/
        >;
```

3.2.2.3 *Defining peripheral PLL frequencies with st,pll property*

This property can be used to configure PLL frequencies.

The PLL children nodes for PLL1 to PLL4 (see reference manual for details) are associated with an index from 0 to 3 (st,pll@0 to st,pll@3).

PLL2, PLL3 or PLL4 are off when their associated nodes are absent or deactivated.

The configuration of PLL1, the source clock of Cortex-A7 core, with st,pll@0 node, is optional as TF-A automatically selects the most suitable operating point for the platform (please refer to How to change the CPU frequency). The node st,pll@0 node should be absent; it is only used if you want to override the PLL1 properties computed by TF-A (clock spreading for example).

Below the available properties for each PLL node:

- cfg contains the PLL configuration parameters in the following order: DIVM, DIVN, DIVP, DIVQ, DIVR, output.

  DIVx values are defined as in RCC:

  - 0x0: bypass (division by 1)

- 0x1: division by 2
- 0x2: division by 3
- 0x3: division by 4
- ...

Output contains a bitfield for each output value (1:ON / 0:OFF)

- BIT(0) output P : DIVPEN
- BIT(1) output Q : DIVQEN
- BIT(2) output R : DIVREN

Note: PQR(p,q,r) macro can be used to build this value with p, q, r = 0 or 1.

- frac: fractional part of the multiplication factor (optional, when absent PLL is in integer mode).
- csg contains the clock spreading generator parameters (optional) in the following order: MOD_PER, INC_STEP and SSCG_MODE.

MOD_PER: modulation period adjustment

INC_STEP: modulation depth adjustment

SSCG_MODE: Spread spectrum clock generator mode, defined in *stm32mp1-clksrc.h*[4]:

- SSCG_MODE_CENTER_SPREAD = 0
- SSCG_MODE_DOWN_SPREAD = 1

Example:

```
        pll2: st,pll@1 {
                   compatible = "st,stm32mp1-pll";
                   reg = <1>;
                cfg = < 1 43 1 0 0 PQR(0,1,1) >;
                csg = < 10 20 1 >;
        };
        pll3: st,pll@2 {
                   compatible = "st,stm32mp1-pll";
                   reg = <2>;
                cfg = < 2 85 3 13 3 0 >;
                csg = < 10 20 SSCG_MODE_CENTER_SPREAD >;
        };
        pll4: st,pll@3 {
                   compatible = "st,stm32mp1-pll";
                   reg = <3>;
                cfg = < 2 78 4 7 9 3 >;
        };
```

3.2.2.4 *Defining peripheral kernel clock tree distribution with st,pkcs property*

This property can be used to configure the peripheral kernel clock selection.

It is a list of peripheral kernel clock source identifiers defined by the CLK_<KERNEL-CLOCK>_<PARENT-CLOCK> macros in the *stm32mp1-clksrc.h*[4] header file.

st,pkcs may not list all the kernel clocks. No specific order is required.

Example:

```
        st,pkcs = <
                CLK_STGEN_HSE
                CLK_CKPER_HSI
```

```
                CLK_USBPHY_PLL2P
                CLK_DSI_PLL2Q
                CLK_I2C46_HSI
                CLK_UART1_HSI
                CLK_UART24_HSI
        >;
```

3.2.2.5 *HSI and CSI clocks calibration*

Note: this section applies to OP-TEE OS clock calibration support.

The calibration is an optional feature that can be enabled from the device tree. It allows requesting the HSI or CSI clock calibration by several means:

- SiP SMC service
- Periodic calibration every X seconds
- Interrupt raised by the MCU

This feature requires that a hardware timer is assigned to the calibration sequence.

A dedicated interrupt must be defined using "mcu_sev" name to start a calibration on detection of an interrupt raised by the MCU.

- st,hsi-cal: used to enable HSI clock calibration feature.
- st,csi-cal; used to enable CSI clock calibration feature.
- st,cal-sec: used to enable periodic calibration at specified time intervals from the secure monitor. The time interval must be given in seconds. If not specified, a calibration is only processed for each incoming request.

Example:

```
        &rcc {
                st,hsi-cal;
                st,csi-cal;
                st,cal-sec = <15>;
                secure-interrupts = <GIC_SPI 144 IRQ_TYPE_LEVEL_HIGH>,
                                    <GIC_SPI 145 IRQ_TYPE_LEVEL_HIGH>;
                interrupt-names = "mcu_sev", "wakeup";
        };
```

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:

- docs/devicetree/bindings/clock/st,stm32mp1-rcc.txt TF-A clock binding information file
- doc/device-tree-bindings/clock/st,stm32mp1.txt U-Boot SPL for DDR interactive mode clock binding information file
- fdts/stm32mp151.dtsi (for TF-A), arch/arm/dts/stm32mp15-no-scmi.dtsi (for U-Boot SPL for DDR interactive mode): STM32MP151 device tree files
- 4.04.14.24.3 include/dt-bindings/clock/stm32mp1-clksrc.h (for TF-A), include/dt-bindings/clock/stm32mp1-clksrc.h (for U-Boot SPL for DDR interactive mode): STM32MP1 DT bindings clock source files
- include/dt-bindings/clock/stm32mp1-clks.h (for TF-A), include/dt-bindings/clock/stm32mp1-clks.h (for U-Boot SPL for DDR interactive mode): STM32MP1 DT bindings clock identifier files

Doubledata rate (memory domain)

Open Portable Trusted Execution Environment

Operating System

Trusted Firmware for Arm Cortex-A

Boot Loader stage 2

Reset and Clock Control

Linux® is a registered trademark of Linus Torvalds.

First Stage Boot Loader

Secondary Program Loader, *Also known as* ***U-Boot SPL***

Second Stage Boot Loader

Device Tree

Generic Interrupt Controller

Serial Peripheral Interface

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

Read Only Memory

High Speed External oscillator (STM32 clock source)

Low Speed External oscillator (STM32 clock source)

Low Speed Internal oscillator (STM32 clock source)

Multi Speed Internal oscillator (STM32 clock source)

Microprocessor Unit

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.)

Real Time Clock

Cortex®

System Time Generator

Display Serial Interface (MIPI® Alliance standard)

Silicon Provider

Secure Monitor Call

Stable: 08.01.2021 - 14:59 / Revision: 08.01.2021 - 14:59

A quality version of this page, approved on *8 January 2021*, was based off this revision.

Contents

1 Article Purpose 18
2 Overview 19
3 Developer Package 20
3.1 Install sources 20
3.2 Official source tree 20
3.3 Build Process 20
3.3.1 Initialize the cross compile environment 20
3.3.2 TF-A Build flags 20
3.4 Build command 21
3.5 Final image 21
4 Distribution Package 23
4.1 Access sources 23
5 Update software on board 24
5.1 Partitioning of binaries 24
5.2 Update via SDCARD 24
5.3 Update via USB mass storage on U-boot 25
5.4 Update your boot device via STM32CubeProgrammer 25
6 Secure secret provisioning 26
6.1 Developer Package 26
6.1.1 Install sources 26
6.1.2 Additional Flags 26
6.1.3 Build command 26
6.1.4 Final image 26
6.2 Distribution Package 26
6.2.1 Access sources 27

1 Article Purpose

This section details the process used to build TF-A from sources and to deploy it on your target.
The build example is based on the OpenSTLinux environment:

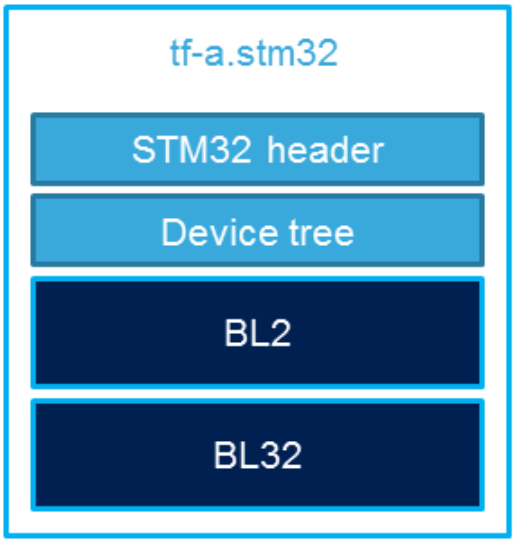
- Developer Package
- Distribution Package

2 Overview

TF-A is the FSBL for the ST trusted boot chain. It must be configured or updated depending on your platform.

Cross compilation of TF-A is only required if it is to be modified. By default, in the Starter Package, the TF-A image is named: tf-<board>-trusted.stm32.
If changes are made, you must rebuild TF-A and update all the FSBL partitions of your boot device with this new image. A second FSBL image is used as a backup image.

The build process creates a full STM32 image that can be used for Flash integrating a specific header.
This trusted firmware-A image contains a device tree, a BL2 and a BL32 stage.
These binaries are built in a single step during the build process.



3 Developer Package

3.1 Install sources

The Developer Package contains OpenSTLinux and TF-A sources: TF-A Installation

3.2 Official source tree

Download source code from the official Trusted Firmware-A github.

```
PC $> git clone https://github.com/ARM-software/arm-trusted-firmware.git
```

Warning

The STM32MP1 platform is not yet fully upstreamed. Depending on the version used, some features may not be available.

For a full feature software, a STMicroelectronics github is available:

```
PC $> git clone https://github.com/STMicroelectronics/arm-trusted-firmware.git
```

3.3 Build Process

3.3.1 Initialize the cross compile environment

Setup Cross compile environment

3.3.2 TF-A Build flags

A Makefile is provided with the developer package that includes the mandatory flags to build the Trusted Firmware-A for STM32MP15.

Mandatory flags:

- ARM_ARCH_MAJOR=7: the major version of ARM Architecture to target (STM32MP15 is ARMv7 architecture based)
- ARCH=aarch32: specify aarch32 architecture to be built
- PLAT=stm32mp1: builds an stm32mp1 platform
- DTB_FILE_NAME=<fdt file name>.dtb: this must be defined to build the proper target and include the correct DTB file into the final file
- AARCH32_SP=<monitor>
  - sp_min: builds the BL32 secure monitor if required
  - optee: do not include BL32 and prepare BL2 for optee-specific load.
- The boot device(s) you use, one (or more) of:
  - STM32MP_EMMC=1

- STM32MP_SDMMC=1
- STM32MP_RAW_NAND=1
- STM32MP_SPI_NAND=1
- STM32MP_SPI_NOR=1

- Or a programming interface (you cannot use AARCH32_SP=optee with those flags):
  - STM32MP_UART_PROGRAMMER=1
  - STM32MP_USB_PROGRAMMER=1

Optional flags:

- DEBUG=1: add debug information in all binaries
- V=1: print verbose compilation traces

3.4 Build command

From the Developer Package tarball, a Makefile.sdk is present and must be used to build the target. It automatically sets the proper configuration for the TF-A build.

```
PC $> make -f Makefile.sdk TF_A_CONFIG=trusted TFA_DEVICETREE=<board>
```

The latest version of the helper file is also available in GitHub: README_HOWTO.txt .

Warning

The DTB_FILE_NAME flag must be set to select the correct board configuration.

The device tree file for the target must be located in fdts folder (<board>.dts)

If no Makefile.sdk exists, you must add your own environment flags:

```
PC $> unset LDFLAGS;
PC $> unset CFLAGS;
```

Then you will have to compile 2 TF-A binaries: one for flash programming (USB or UART), one for device boot (SD-card, eMMC , SPI-NOR, SPI-NAND or parallel NAND (through FMC)):

```
 PC $> make ARM_ARCH_MAJOR=7 ARCH=aarch32 PLAT=stm32mp1 AARCH32_SP=sp_min DTB_FILE_NAME=<b
oard>.dtb STM32MP_UART_PROGRAMMER=1 STM32MP_USB_PROGRAMMER=1
 PC $> make ARM_ARCH_MAJOR=7 ARCH=aarch32 PLAT=stm32mp1 AARCH32_SP=sp_min DTB_FILE_NAME=<b
oard>.dtb STM32MP_EMMC=1 STM32MP_SDMMC=1 STM32MP_RAW_NAND=1 STM32MP_SPI_NAND=1 STM32MP_SPI
_NOR=1
```

It is advised to remove from the command line the devices you do not use to boot, to ensure that the built binary will fit in the SYSRAM on startup.

3.5 Final image

Final image is available for Flash or SD card update in the corresponding folder:

```
build/<target>/<debug|release>/tf-a-<target>.stm32
Ex:
build/stm32mp1/debug/tf-a-stm32mp157c-ev1.stm32
```

4 Distribution Package

For an OpenSTLinux distribution, the TF-A image is built in release mode by default. The yocto recipe can be found in:

```
meta-st/meta-st-stm32mp/recipes-bsp/trusted-firmware-a/tf-a-stm32mp_<version>.bb
```

If you want to modify the TF-A code source, use the following steps starting from an already downloaded and built OpenSTLinux distribution.

4.1 Access sources

You can use devtool to access the source.

```
PC $> cd <baseline root directory>
PC $> devtool modify tf-a-stm32mp sources/boot/tf-a
```

By going to the sources/boot/tf-a folder, you can manage and modify the TF-A sources. To rebuild it, go back to the build-<distribution> folder and launch the TF-A recipe:

```
PC $> bitbake tf-a-stm32mp
```

The final image is deployed in the image default output folder.

5 Update software on board

5.1 Partitioning of binaries

The TF-A build provides a binary named tf-a-stm32mp157c-<board>.stm32 that MUST be copied to a dedicated partition named "fsblX" (X depends on the number of needed backups in the Flash).

Warning

TF-A must be located in the first partition of your boot device.

You can just update the first partition for a simple test, but all backup partitions must contain the same image at the end.

5.2 Update via SDCARD

If you use an SD card, you can simply update TF-A using the dd command on your host.
Plug your SD card into the computer and copy the binary to the dedicated partition; on an SDCard/USB disk the "fsbl1" partition is partition 1:

```
- SDCARD: /dev/mmcblkXp1 (where X is the instance number)
- SDCARD via USB reader: /dev/sdX1 (where X is the instance number)
```

- Linux

```
PC $> dd if=<tf-a file> of=/dev/<device partition> bs=1M conv=fdatasync
```

Information

To find the partition associated to a specific label, just plug the

SDCARD/USB disk into your PC and call the following command:

```
PC $> ls -l /dev/disk/by-partlabel/
total 0
lrwxrwxrwx 1 root root 10 Jan 17 17:38 bootfs -> ../../mmcblk0p4
lrwxrwxrwx 1 root root 10 Jan 17 17:38 fsbl1 -> ../..
/mmcblk0p1          → FSBL1 (TF-A)
lrwxrwxrwx 1 root root 10 Jan 17 17:38 fsbl2 -> ../..
/mmcblk0p2          → FSBL2 (TF-A backup – same content as FSBL)
lrwxrwxrwx 1 root root 10 Jan 17 17:38 rootfs -> ../../mmcblk0p5
lrwxrwxrwx 1 root root 10 Jan 17 17:38 ssbl -> ../..
/mmcblk0p3           → SSBL (U-Boot)
lrwxrwxrwx 1 root root 10 Jan 17 17:38 userfs -> ../../mmcblk0p6
```

- Windows

CoreUtils [1] that includes the dd command is available for Windows.

5.3 Update via USB mass storage on U-boot

See How to use USB mass storage in U-Boot

Follow the previous section to put tf-a-<board>.stm32 onto SDCard/USB disk

5.4 Update your boot device via STM32CubeProgrammer

Refer to the STM32CubeProgrammer documentation to update your target.

6 Secure secret provisioning

A specific TF-A build is required to manage SSP.

A dedicated branch (named **<version>-stm32mp-ssp**) is delivered on top of the official TF-A release that contains the specific Makefile for the TF-A SSP.
The TF-A SSP is a subset part of the TF-A that only includes:

- BL2 device tree
- BL2 image with limited support to the serial link device.

6.1 Developer Package

6.1.1 Install sources

The Developer Package contains OpenSTLinux and TF-A-SSP sources: TF-A-SSP Installation

Warning

The SSP is a specific ST feature and will never be upstreamed.

6.1.2 Additional Flags

Mandatory flags to build the TF-A SSP are:

- **STM32MP_SSP=1**

For the serial link storage

- STM32MP_UART_PROGRAMMER=1
- STM32MP_USB_PROGRAMMER=1

6.1.3 Build command

```
 PC $> make ARM_ARCH_MAJOR=7 ARCH=aarch32 PLAT=stm32mp1 DTB_FILE_NAME=<board>.dtb STM32MP_
SSP=1 STM32MP_UART_PROGRAMMER=1 STM32MP_USB_PROGRAMMER=1
```

6.1.4 Final image

Final image is available in the corresponding folder:

```
build/<target>/<debug|release>/tf-a-ssp-<target>.stm32
Ex:
build/stm32mp1/debug/tf-a-ssp-stm32mp157c-ev1.stm32
```

6.2 Distribution Package

For an OpenSTLinux distribution, the TF-A SSP image is **not** built in release mode by default. The yocto recipe can be found in:

```
meta-st/meta-st-stm32mp/recipes-bsp/trusted-firmware-a/tf-a-stm32mp-ssp_<version>.bb
```

If you want to modify the TF-A SSP code source, use the following steps starting from an already downloaded and built OpenSTLinux distribution.

6.2.1 Access sources

You can use devtool to access the source.

```
PC $> cd <baseline root directory>
PC $> devtool modify tf-a-stm32mp-ssp sources/boot/tf-a_ssp
```

By going to the sources/boot/tf-a_ssp folder, you can manage and modify the TF-A sources. To rebuild it, go back to the build-<distribution> folder and launch the TF-A recipe:

```
PC $> bitbake tf-a-stm32mp-ssp
```

The final image is deployed in the image default output folder.

- http://gnuwin32.sourceforge.net/packages/coreutils.htm

Trusted Firmware for Arm Cortex-A

First Stage Boot Loader

Boot Loader stage 2

Boot Loader stage 3-2

Device Tree Binary (or Blob)

Serial Peripheral Interface

Universal Asynchronous Receiver/Transmitter

Secure digital

former spelling for e•MMC ('e' in italic)

SD memory card (https://www.sdcard.org)

Flash memory shortened to gain space in titles, tables and block diagrams

Linux® is a registered trademark of Linus Torvalds.

Second Stage Boot Loader

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

Secure Secret Provisioning

Secure secrets provisioning

Stable: 12.10.2020 - 12:02 / Revision: 12.10.2020 - 12:01

A quality version of this page, approved on *12 October 2020*, was based off this revision.

Contents

1 Article purpose 29
2 Framework overview 30
2.1 Components description 30
3 DT configuration 31

4 Raw NAND Flash memory 32
4.1 Raw NAND DT configuration 32
4.2 Raw NAND device configuration 32
4.2.1 Raw NAND examples 32
5 SPI NAND flash memory 34
5.1 SPI NAND DT configuration 34
5.2 SPI NAND device configuration 34
5.2.1 SPI NAND framework 34
5.2.2 SPI NAND example 34
6 SPI NOR Flash memory 36
6.1 SPI NOR DT configuration 36
6.2 SPI NOR device configuration 36
6.2.1 SPI NOR framework 36
6.2.2 SPI NOR example 36
7 How to configure the DT using STM32CubeMX 37
8 References 38

1 Article purpose

This article explains how to configure the TF-A MTD frameworks:

- SPI NOR
- SPI NAND
- SPI MEM
- FMC NAND

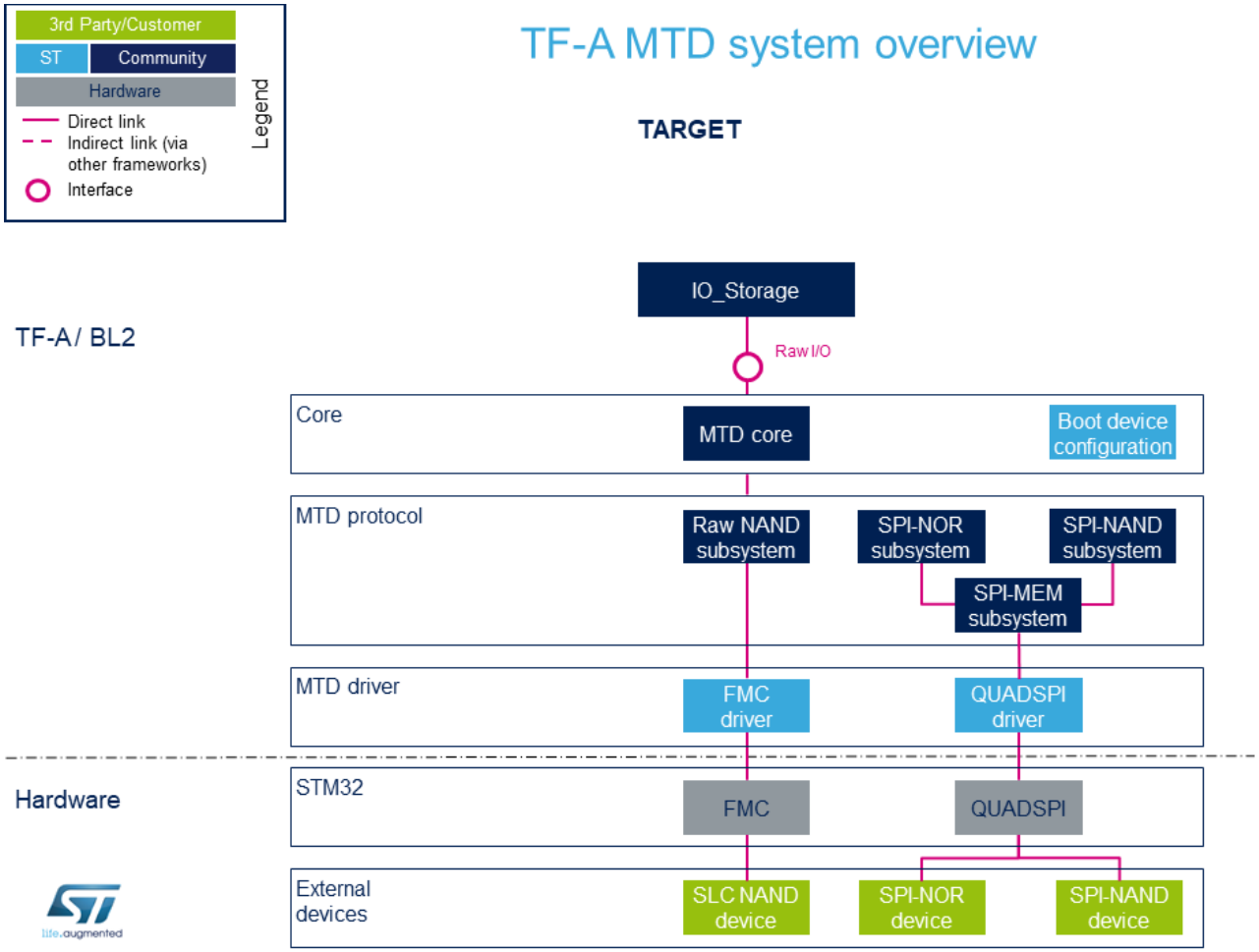
These frameworks represent the memory-access organisation.

They use two different configurations:

- Device tree configuration
- Boot device configuration

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 Framework overview



2.1 Components description

- **IO_Storage**. The IO storage provides an abstraction layer to access storage devices.
- **Boot device configuration**. The Boot device configuration is a platform specific add-on to manage Flash memory settings.
- **MTD core**. The MTD core provides an abstraction layer for raw Flash memories.
- **Raw NAND subsystem**. The Raw NAND protocol is used in the MTD subsystem for interfacing NAND Flash memories.
- **SPI-MEM subsystem**. The SPI-MEM protocol is used in the MTD subsystem for interfacing all kinds of SPI memories (NORs, NANDs).
- **SPI-NAND subsystem**. The SPI-NAND protocol is used in the MTD subsystem for interfacing SPI NAND Flash memories.
- **SPI-NOR subsystem**. The SPI-NOR protocol is used in the MTD subsystem for interfacing SPI NOR Flash memories.
- **FMC driver** / **FMC** (Hardware) . Please refer to the FMC internal peripheral.
- **QUADSPI driver** / **QUADSPI** (Hardware). Please refer to the QUADSPI internal peripheral.

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.

Pinctrl device tree configuration (and optionally to Pinctrl overview) must be added in #DT configuration (board level).

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

4 Raw NAND Flash memory

4.1 Raw NAND DT configuration

For the DT bindings, refer to FMC_device_tree_configuration.

For the DT configuration (STM32 level), refer to FMC DT Configuration at STM32 level.

For the DT configuration (board level), refer to FMC DT Configuration at board level.

Warning

Only the device required to load images must be declared as a child node.

4.2 Raw NAND device configuration

Raw NAND access uses the raw NAND framework. Some additional parameters are required by the raw NAND framework to address the memory:

- the page size
- the block size
- the number of blocks per device
- the bus width (8 or 16 bits)
- the ECC algorithm (HW BCH8/BCH4/Hamming algorithms are available). The default ECC used is ECC NONE (no error correction).

Some memories are ONFI[1] compliant. In that case, the required parameters can be directly read from the parameter description table.

For the others, the user must correctly fill-out the OTP configuration.

4.2.1 Raw NAND examples

- ONFI raw NAND with ECC override (default from parameter table is BCH4, force to BCH8).

```
OTP Word 9 : 0x18000
```

Use OTP	Page size		Block size		Block numbers (N * 256)								Width	ECC																	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- 8-bit raw NAND (page size: 4 Kbytes, block size: 256 Kbytes, device size: 1024 Mbytes, ECC requirement: BCH4)

```
OTP Word 9 : 0xA0808000
```

Use OTP	Page size		Block size		Block numbers (N * 256)								Width	ECC																	
1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- 16-bit raw NAND (page size: 8 Kbytes, block size: 512 Kbytes, device size: 2048 Mbytes, ECC requirement: ON-DIE)

```
OTP Word 9 : 0xc0860000
```

Use OTP	Page size		Block size		Block numbers (N * 256)								Width	ECC																	
1	1	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5 SPI NAND flash memory

5.1 SPI NAND DT configuration

For the DT bindings, refers to the QUADSPI_device_tree_configuration.

For the DT configuration (STM32 level), refer to QUADSPI DT Configuration at STM32 level.

For the DT configuration (board level), refer to QUADSPI DT Configuration at board level.

Warning

Only the device required to load images must be declared as a child node

5.2 SPI NAND device configuration

SPI NAND and SPI MEM frameworks are used to address such memories.

5.2.1 SPI NAND framework

SPI NAND framework requires additional parameters:

- the page size
- the block size
- the number of blocks per device
- the number of planes per device.

These parameters must be correctly filled out by the user in OTP configuration.

By default, the **READ FROM CACHE x1** command is used (opcode: **0x03**). It is possible to override this command in the platform configuration to improve memory-access performance .

5.2.2 SPI NAND example

- SPI NAND (page size: 2 Kbytes, block size: 128 Kbytes, device size: 256 Mbytes, 2 planes)

```
OTP Word 9 : 0x80404000
```

Use OTP	Page size		Block size		Block numbers (N * 256)								Not Used			2 Planes														
1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

- Update the **READ FROM CACHE x1** command to use the **READ FROM CACHE x4** command (opcode: **0x6B**) defined in platform file[2]

```
device->spi_read_cache_op.cmd.opcode = SPI_NAND_OP_READ_FROM_CACHE_4X;
device->spi_read_cache_op.cmd.buswidth = SPI_MEM_BUSWIDTH_1_LINE;
device->spi_read_cache_op.addr.nbytes = 2U;
device->spi_read_cache_op.addr.buswidth = SPI_MEM_BUSWIDTH_1_LINE;
device->spi_read_cache_op.dummy.nbytes = 1U;
device->spi_read_cache_op.dummy.buswidth = SPI_MEM_BUSWIDTH_1_LINE;
device->spi_read_cache_op.data.buswidth = SPI_MEM_BUSWIDTH_4_LINE;
device->spi_read_cache_op.data.dir = SPI_MEM_DATA_IN;
```

6 SPI NOR Flash memory

6.1 SPI NOR DT configuration

For the DT bindings, refer to the QUADSPI_device_tree_configuration.

For the DT configuration (STM32 level), refer to QUADSPI DT Configuration at STM32 level.

For the DT configuration (board level), refer to QUADSPI DT Configuration at board level.

Warning

Only the device required to load images must be declared as a child node.

6.2 SPI NOR device configuration

SPI NOR and SPI MEM frameworks are used to address such memories.

6.2.1 SPI NOR framework

SPI NOR framework requires additional parameter:

- Device size

This parameter needs to be defined in the platform configuration file<ref_name="boot_device">.

By default, the **READ** command is used (opcode: **0x03**). It is possible to override this command in the platform configuration to improve memory-access performance .

6.2.2 SPI NOR example

- SPI NOR (device size: 64 Mbytes)

```
        device->size = SZ_64M;                                  --> Device size
```

- Update the **READ** command to use **QREAD** command (opcode: **0x6B**) defined in platform file[2]

```
        device->read_op.cmd.opcode = SPI_NOR_OP_READ_1_1_4;
        device->read_op.cmd.buswidth = SPI_MEM_BUSWIDTH_1_LINE;
        device->read_op.addr.nbytes = 3U;
        device->read_op.addr.buswidth = SPI_MEM_BUSWIDTH_1_LINE;
        device->read_op.dummy.nbytes = 1U;
        device->read_op.dummy.buswidth = SPI_MEM_BUSWIDTH_1_LINE;
        device->read_op.data.buswidth = SPI_MEM_BUSWIDTH_4_LINE;
        device->read_op.data.dir = SPI_MEM_DATA_IN;
```

7 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 might 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 the STM32CubeMX user manual for further information.

8 References

Please refer to the following links for additional information:

- http://www.onfi.org/specifications
- 2.02.1 plat/st/stm32mp1/stm32mp1_boot_device.c

Trusted Firmware for Arm Cortex-A

Memory Technology Device

Serial Peripheral Interface

input/output

Flash memories combine high density and cost effectiveness of EPROMs with the electrical erasability of EEPROMs. For this reason, the Flash memory market is one of the most exciting areas of the semiconductor industry today and new applications requiring in system reprogramming, such as cellular telephones, automotive engine management systems, hard disk drives, PC BIOS software for Plug & Play, digital TV, set top boxes, fax and other modems, PC cards and multimedia CD-ROMs, offer the prospect of very high volume demand.

Device Tree

Elliptic curve cryptography

Error Correction Capability

Open NAND Flash interface (The ONFI working group, acronym for Open NAND Flash Interface, was founded in 2005. The group’s mission consists in creating a common industry standard for NAND Flash interfaces, to simplify integration of NAND Flash memory into consumer electronics (CE) devices and computing platforms. ST is one of the co-founder companies together with Hynix, Intel, Micron, Phison and Sony.)

One Time Programmed

Stable: 08.01.2021 - 15:47 / Revision: 08.01.2021 - 15:47

A quality version of this page, approved on *8 January 2021*, was based off this revision.

Contents

1 Article Purpose 39
2 Debugging 40
2.1 TF-A Version number 40
2.2 Debug with traces 40
2.3 Debug with GDB 40
2.3.1 Debug boot sequence 40
2.3.2 Debugging during runtime execution 41

1 Article Purpose

This article explains how to debug TF-A firmware.
Debug is specifically linked to the TrustZone environment.

There are two main ways to debug TF-A, using traces inside the code, or by using JTAG to access the secure world. The focus here is on the solution integrated in OpenSTLinux: debug over gdb (ST-Link or JTAG based)

2 Debugging

2.1 TF-A Version number

The starting point for debugging is to identify the TF-A version used in the target. Debug and release versions are displayed on the console with the following format:

```
NOTICE: BL2: v2.0(debug):<tag>
```

- v2.0 is the TF-A version used.
- (debug) : Build mode enable
- <tag> : Git reference resulting from the **git describe** command at build time

2.2 Debug with traces

TF-A allows RELEASE and DEBUG modes to be enabled:

- RELEASE mode builds with default LOG_LEVEL to 20, which only prints ERROR and NOTICE traces
- DEBUG mode enables the -g build flag (for debug build object), and defaults LOG_LEVEL to 40

With the debug LOG_LEVEL, you can add console traces such as ERROR, NOTICE, WARNING or INFO. Please refer to **include/common/debug.h**.

Warning

You cannot build code with LOG_LEVEL set to 50 (the highest level); there are too many traces and TF-A does not fit in the SYSRAM. If required, change some VERBOSE settings to INFO.

For both modes, you must ensure that the UART is properly configured:

- BL2: UART traces are enabled by default
- BL32: UART traces are enabled by default

Traces and errors are available on the console defined in the chosen node of the device tree by the stdout-path property:

```
chosen {
        stdout-path = "serial0:115200n8";
};
```

2.3 Debug with GDB

TF-A runs in SYSRAM and is executed in CPU secure state. One can debug TF-A through JTAG using an ST-Link or the JTAG output, depending on the board.

2.3.1 Debug boot sequence

The BL2 boot stage is only executed during a boot sequence. To break into BL2, connect to the target and break.

Load symbols to the correct offset:

```
(gdb) add-symbol-file <path_to_build_folder>/tf-a-bl2.elf (or bl2.elf) <load_address>
```

BL2 and BL32 load addresses can be found in the generated **tf-a-stm32mp157c-<board>.map** file:

```
...
 *fill*         0x000000002ffce000     0x5000
                0x000000002ffd3000                __BL2_IMAGE_START__ = .  -> BL2 Load
address
 *(.bl2_image*)
 .bl2_image     0x000000002ffd3000    0x1166c build/stm32mp1/stm32mp1.o
                0x000000002ffe466c                __BL2_IMAGE_END__ = .
                0x0000000000024b00                . = 0x24b00
 *fill*         0x000000002ffe466c     0x2994
                0x000000002ffe7000                __BL32_IMAGE_START__ = . -> BL32 Load
address
 *(.bl32_image*)
 .bl32_image    0x000000002ffe7000    0x1744c build/stm32mp1/stm32mp1.o
                0x000000002fffe44c                __BL32_IMAGE_END__ = .
                0x000000002fffe44c                __DATA_END__ = .
                0x000000002fffe44c                __TF_END__ = .
...
```

In this example:

- BL2 load address is 0x2ffd3000.
- BL32 load address is 0x2ffe7000.

You can load all your symbols directly:

```
(gdb) add-symbol-file <path_to_build_folder>/tf-a-bl2.elf 0x2ffd3000
(gdb) add-symbol-file <path_to_build_folder>/tf-a-bl32.elf 0x2ffe7000
```

Using the Wrapper_for_FSBL_images, you will be able to debug the initial boot sequence. Set your hardware breakpoint and reset:

```
(gdb) hb bl2_entrypoint
(gdb) monitor reset halt
(gdb) continue
```

2.3.2 Debugging during runtime execution

Once U-Boot or the Linux kernel is running, you cannot access secure memory or regions, but you can break, set a hardware breakpoint into SMC handler (in BL32). GDB breaks once it has switched into the secure world and reached the break instruction. Once halted in the secure monitor, GDB can access secure resources as IPs or memory. For example, one can break into kernel:

```
(gdb) hb stm32mp1_svc_smc_handler
(gdb) continue
```

On the first SMC call occurence GDB breaks the stm32mp1_svc_smc_handler() entry in BL32.

TrustZone®

debug and test protocol, named from the Joint Test Action Group that developed it

spelling for older versions of STLink, ST in-circuit debugger and programmer for the STM8 and STM32 microcontroller families

Boot Loader stage 2

Universal Asynchronous Receiver/Transmitter

Boot Loader stage 3-2

GNU dedugger, a portable debugger that runs on many Unix-like systems

Central processing unit

Linux® is a registered trademark of Linus Torvalds.

Secure Monitor Call

Stable: 17.02.2021 - 19:40 / Revision: 16.02.2021 - 16:25

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

Contents

1 Trusted Firmware-A 43
2 Architecture 44
3 Boot loader stages 46
3.1 BL1 46
3.2 BL2 46
3.3 BL32 46
4 References 47

1 Trusted Firmware-A

Trusted Firmware-A is a reference implementation of secure-world software provided by Arm®. It was first designed for Armv8-A platforms, and has been adapted to be used on Armv7-A platforms by STMicroelectronics. Arm is transferring the Trusted Firmware project to be managed as an open-source project by Linaro.[1]

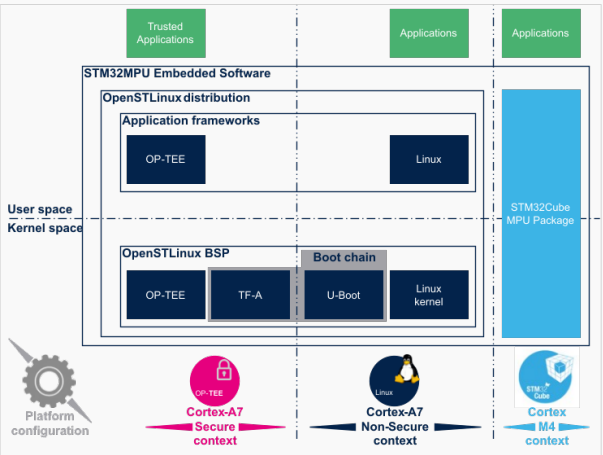
It is used as the first-stage boot loader (FSBL) on STM32 MPU platforms when using the trusted boot chain.

The code is open source, under a BSD-3-Clause licence, and can be found on github [2], including an up-to-date documentation about Trusted Firmware-A implementation [3].

Trusted Firmware-A also implements a secure monitor with various Arm interface standards:

- The power state coordination interface (PSCI) [4]
- Trusted board boot requirements (TBBR) [5]
- SMC calling convention [6]
- System control and management interface [7]

Trusted Firmware-A is usually shortened to TF-A.



Zoom out to STM32MPU Embedded Software

2 Architecture

The global architecture of TF-A is explained in the Trusted Firmware-A design [8] document.

TF-A is divided into several binaries, each with a dedicated main role. For 32-bit Arm processors (AArch32), it is divided into four steps (in order of execution):

- Boot loader stage 1 (BL1) application processor trusted ROM
- Boot loader stage 2 (BL2) trusted boot firmware
- Boot loader stage 3-2 (BL32) runtime software
- Boot loader stage 3-3 (BL33) non-trusted firmware

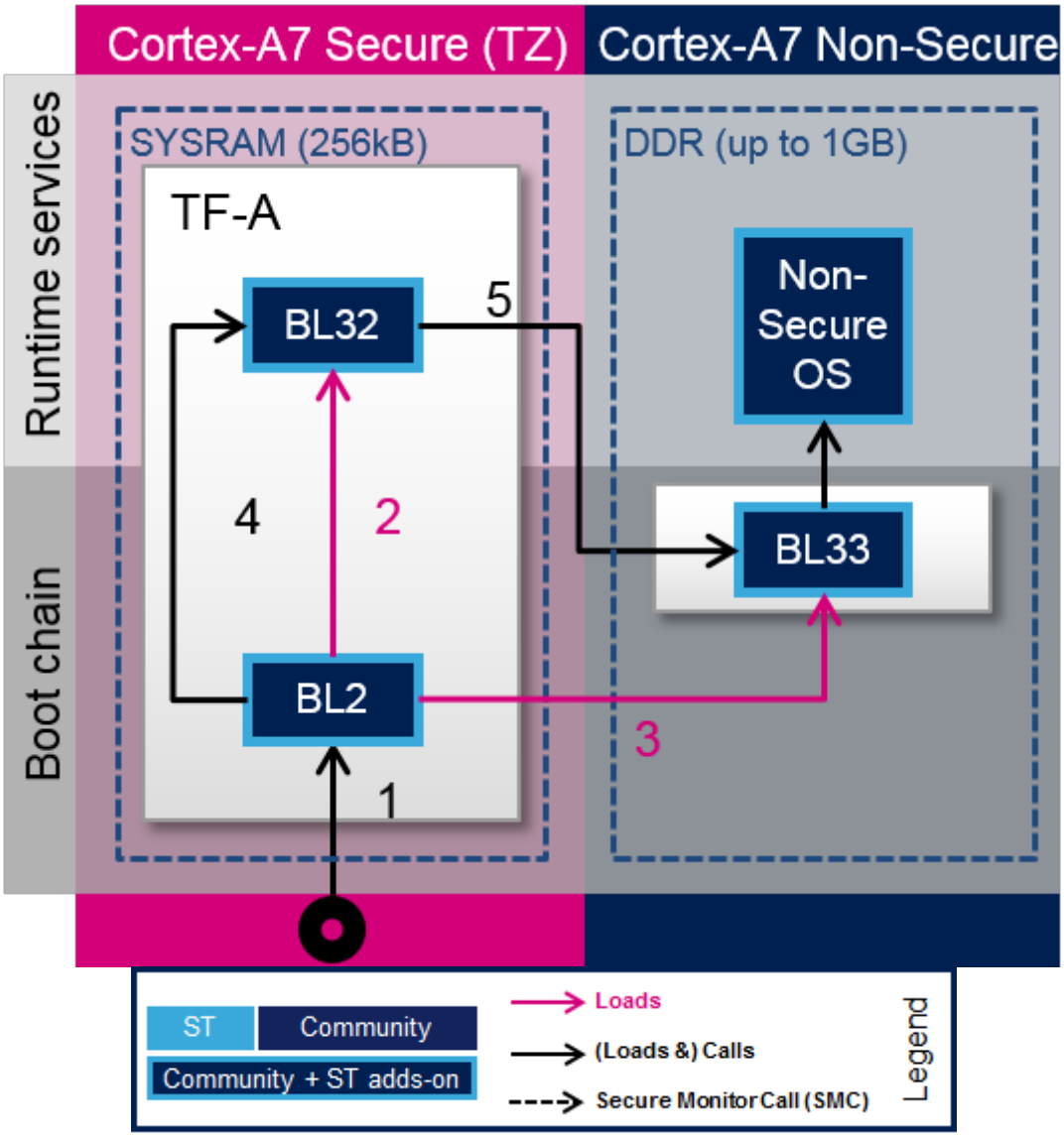
BL1, BL2 and BL32 are parts of TF-A.

BL1 is now optional, and can be removed by enabling the compilation flag: BL2_AT_EL3. It is then removed for the STM32MP1, as all BL1 tasks are done by ROM code, or BL2.

BL33 is outside of TF-A. This is the first non-secure code loaded by TF-A. During the boot sequence, this is the secondary stage boot loader (SSBL). For STM32 MPU platforms, the SSBL is U-Boot by default.

TF-A can manage its configuration with a device tree, as this is the case on STM32MP1. It is a reduced version of the Linux kernel one, with only the devices used during boot. It can be configured with STM32CubeMX.

In STMicroelectronics' implementation, the 2 binaries, BL2 and BL32, and the device tree are put together in a single binary, to be loaded at once to the SYSRAM by the ROM code.



TF-A loading steps:

1. ROM code loads TF-A binary and calls BL2
2. BL2 prepares BL32
3. BL2 loads BL33
4. BL2 calls BL32
5. BL32 calls BL33

3 Boot loader stages

3.1 BL1

BL1 is the first stage executed, and is designed to act as ROM code; it is loaded and executed in internal RAM. It is not used for the STM32MP1. As the STM32MP1 has its own proprietary ROM code, this part can be removed and BL2 is then the first TF-A binary to be executed.

3.2 BL2

BL2 (trusted boot firmware) is in charge of loading the next-stage images (secure and non secure). To achieve this role, BL2 has to initialize all the required peripherals.

It has to initialize the security components.
For the STM32MP15, these security peripherals are:

- boot and security, and OTP control (BSEC internal peripheral)
- extended TrustZone protection controller (ETZPC internal peripheral)
- TrustZone address space controller for DDR (TZC internal peripheral)

BL2 is also in charge of initializing the DDR and clock tree.

The boot peripheral has to be initialized.
On the STM32MP15, it can be one of the following:

- SD-card via the SDMMC internal peripheral
- eMMC via the SDMMC internal peripheral
- NAND via the FMC internal peripheral
- NOR via the QUADSPI internal peripheral

USB (OTG internal peripheral) or UART(USART internal peripheral) are used when Flashing, see STM32CubeProgrammer for more details.

BL2 also integrates image verification and authentication. Authentication is achieved by calling BootROM verification services.

At the end of its execution, after having loaded BL32 and the next boot stage (BL33), BL2 jumps to BL32.

3.3 BL32

BL32 provides runtime secure services. In TF-A, the BL32 default implementation is SP-MIN solution. It is described in the TF-A functionality list [3] as: "A minimal AArch32 Secure Payload (SP-MIN) to demonstrate PSCI [4] library integration with AArch32 EL3 Runtime Software."

This minimal implementation can be replaced with a trusted OS or trusted environment execution (TEE), such as OP-TEE. Both solutions (SP-MIN or OP-TEE) are supported by STMicroelectronics for STM32MP1.

BL32 acts as a secure monitor and thus provides secure services to non-secure OSs. These services are called by non-secure software with secure monitor calls [6].

This code is in charge of standard service calls, like PSCI [4].

It also provides STMicroelectronics dedicated services, to access secure peripherals. On the STM32MP1, these services are used to access RCC internal peripheral, PWR internal peripheral, RTC internal peripheral or BSEC internal peripheral.

4 References

- https://www.trustedfirmware.org/
- https://github.com/ARM-software/arm-trusted-firmware
- 3.03.1 readme.rst
- 4.04.14.2 http://infocenter.arm.com/help/topic/com.arm.doc.den0022d/Power_State_Coordination_Interface_PDD_v1_1_DEN0022D.pdf
- Arm DEN0006C-1
- 6.06.1 http://infocenter.arm.com/help/topic/com.arm.doc.den0028b/ARM_DEN0028B_SMC_Calling_Convention.pdf
- http://infocenter.arm.com/help/topic/com.arm.doc.den0056a/DEN0056A_System_Control_and_Management_Interface.pdf
- https://trustedfirmware-a.readthedocs.io/en/latest/design/index.html

First Stage Boot Loader

Microprocessor Unit

Power State Coordination Interface

Secure Monitor Call

Trusted Firmware for Arm Cortex-A

Boot Loader stage 1

Read Only Memory

Boot Loader stage 2

Boot Loader stage 3-2

Boot Loader stage 3-3

Second Stage Boot Loader

Linux® is a registered trademark of Linus Torvalds.

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today for volatile random-acces ssemiconductor memories.)

One Time Programmed

TrustZone®

Doubledata rate (memory domain)

Secure digital

former spelling for e•MMC ('e' in italic)

Universal Asynchronous Receiver/Transmitter

Secure Payload minimal

Operating System

Trusted Execution Environment

Open Portable Trusted Execution Environment