Getting started with Trusted Firmware-M (TF-M) on PSOC™ Control C3M/P8, C3M/P7 MCUs

About this document

Scope and purpose

This application note explains how to get started with Trusted Firmware-M (TF-M) on the PSC3M/P8, PSC3M/P7 devices. It describes how to build a TF-M-based application, add TF-M to an existing application, and use the services offered by TF-M with ModusToolbox™.

Intended audience

This is an introductory document for getting started with TF-M that is intended for users who are familiar with application development with PSOC™ Edge in ModusToolbox™ and PSOC™ Edge Protect Bootloader. For more information, see AN242836 – Getting started with PSOC™ Control C3M/P8, C3M/P7 MCUs security [2].

Table of contents

About this document . . . 1
Table of contents . . . 2
1 Introduction . . . 4
2 Getting started with TF-M . . . 5
2.1 Prerequisites . . . 5
2.2 Boot flow . . . 5
2.3 Application creation and programming procedure . . . 6
2.3.1 Create basic TF-M based application . . . 6
2.3.2 Add Edge Protect Bootloader to TF-M application . . . 9
2.3.3 Image signing and programming . . . 11
2.4 Application structure and build artifacts . . . 13
2.4.1 Application structure . . . 13
2.4.2 Build artifacts . . . 14
2.5 SPE logging . . . 15
3 Add TF-M to an existing application . . . 18
3.1 Add TF-M libraries to the application . . . 18
3.2 Update memory and protection configuration . . . 22
3.3 Add Edge Protect Bootloader . . . 26
3.4 Add TF-M interface to use TF-M services . . . 33
4 TF-M architecture . . . 35
4.1 Intercommunication between SPE and NSPE . . . 36
4.2 Isolation in TF-M . . . 36
4.3 Profiles in TF-M . . . 37
5 Services offered by TF-M . . . 39
5.1 Cryptography . . . 39
5.2 Secured storage . . . 40
5.2.1 Internal trusted storage (ITS) . . . 40
5.2.2 Protected storage (PS) . . . 40
5.3 Initial attestation service . . . 40
5.4 Platform service . . . 41
5.5 IFX Extension Partition . . . 41
6 TF-M memory map . . . 43
6.1 Changing the TF-M memory map . . . 44
7 Secure boot and image upgrade . . . 45
8 Device setup for PSA certification . . . 46
References . . . 48

Table of contents

Glossary 49

Revision history 51

Trademarks 52

Disclaimer 53

1 Introduction

Trusted Firmware-M (TF-M) is an open-source initiative that implements the Arm® Platform Security Architecture (PSA) specifications. This implementation enables chips, real-time operating systems, and devices to become PSA Certified. TF-M leverages the security features of the MCU hardware to establish an isolated secure processing environment (SPE), where sensitive information can be securely processed and stored. In Armv8-M and Armv8.1-M architectures, TF-M relies on the isolation capability of Arm® TrustZone®, while in architectures earlier than Armv8-M, physical core isolation is necessary.

TF-M consists of the following components:

- **TF-M core:** TF-M core manages isolation among partitions of SPE and between the SPE and non-secure processing environment (NSPE). It manages communication among partitions of SPE and between SPE and NSPE. The TF-M core also manages the execution of the SPE partitions and the NSPE application
- **Secure services:** TF-M includes secure services such as Crypto, Internal Trusted Storage (ITS), Protected Storage (PS), firmware update, attestation, and platform. The secure services are utilized by NSPE application via PSA APIs. The secure services are organized into a cohesive software unit referred to as "partition"

Figure 1 (source: TF-M website) illustrates the high-level design of TF-M. For more information about TF-M, see Trusted Firmware-M documentation.

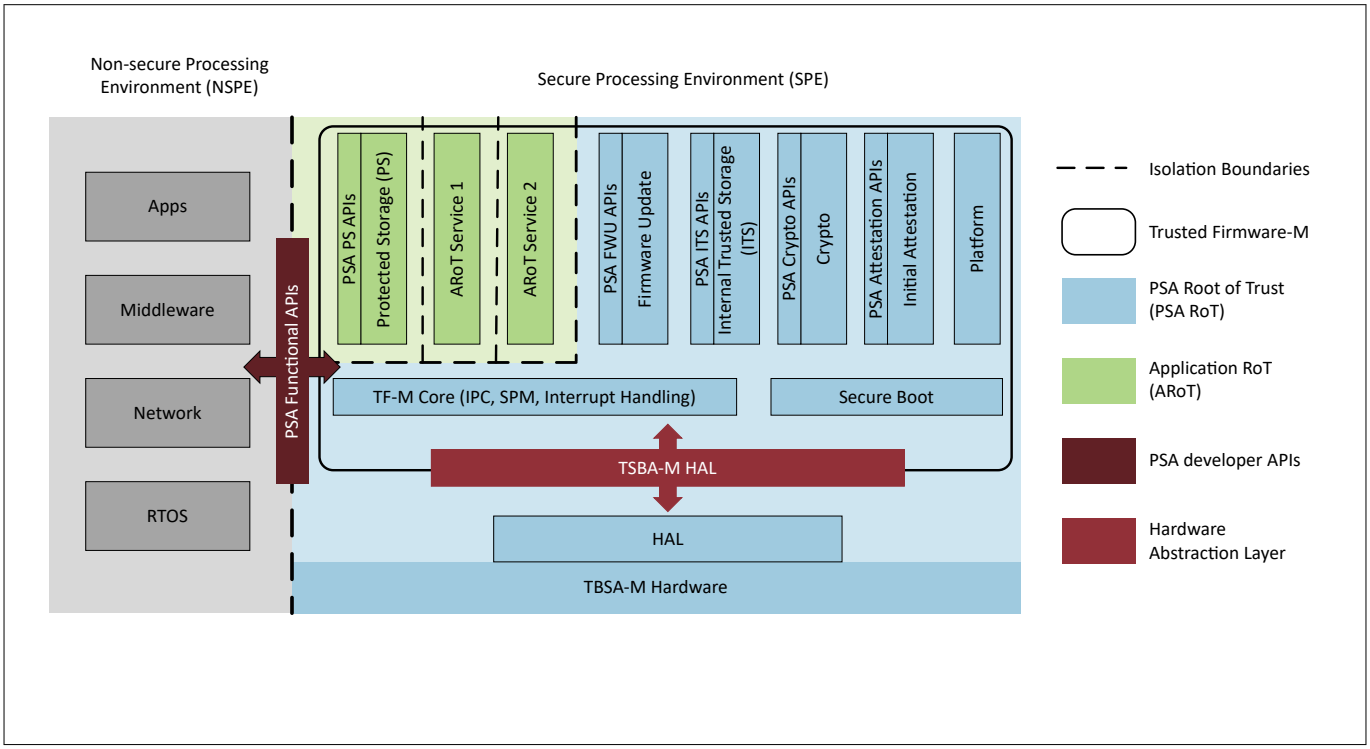


Figure 1 TF-M overview (source: TF-M website)

2 Getting started with TF-M

This section describes how to build an application based on TF-M in ModusToolbox™. A starter template application "PSOC™ Control C3M8 MCU: Basic TF-M application" is available in ModusToolbox™.

2.1 Prerequisites

It is assumed that the reader is familiar with PSOC™ Control application development with ModusToolbox™ software. If you are new to PSOC™ Control application development software, see the AN242836 – Getting started with PSOC™ Control C3M/P8, C3M/P7 MCUs security [2] application note for getting started with application development.

TF-M is designed to work together with the Edge Protect Bootloader, so the reader is expected to have a basic understanding of the Edge Protect Bootloader. If you are new to Edge Protect Bootloader, see AN242836 – Getting started with PSOC™ Control C3M/P8, C3M/P7 MCUs security [2].

2.2 Boot flow

When the PSOC™ Control device comes out of reset, the immutable boot firmware (BootROM) is executed first. The BootROM then initializes the device, then launches the first OEM application, that is the Edge Protect Bootloader. The Edge Protect Bootloader authenticates the TF-M (main M33 SPE), main M33 NSPE, and two PPCA applications and transfers control to TF-M. TF-M performs necessary device initialization, sets up isolation boundaries using various protection units, and transfers control to the main M33 NSPE.

Figure 2 summarizes the above-described boot flow:

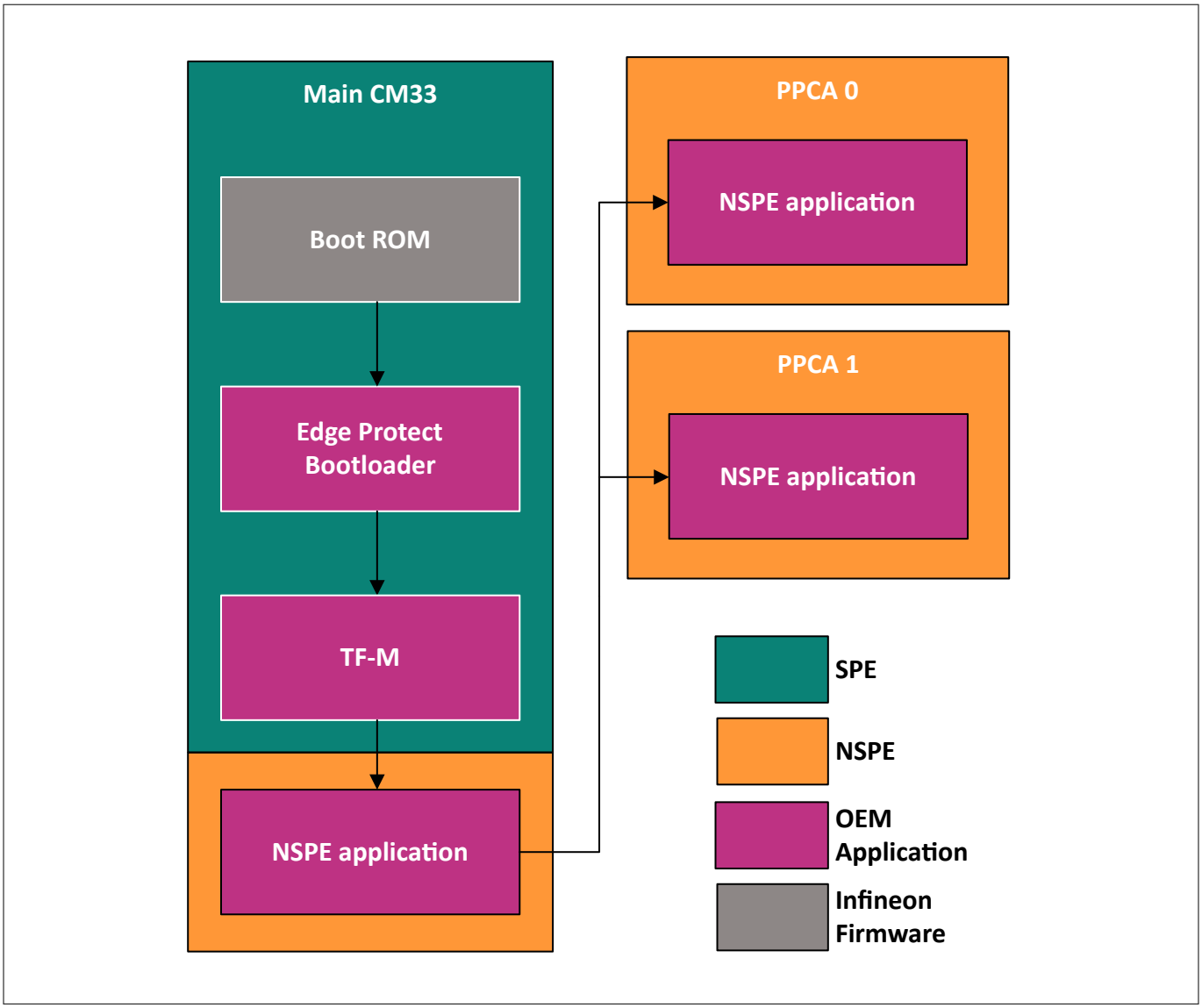


Figure 2 **Boot flow in PSOC™ Control MCU with Edge protect Bootloader**

2.3 Application creation and programming procedure

This section provides step-by-step instructions to get started with PSOC™ Control TF-M application. It provides guidance on how to create the PSOC™ Control C3M8 MCU: Basic TF-M application and use it with Edge Protect Bootloader. If you need to add TF-M to an existing application, see Add TF-M to an existing application section for instructions.

2.3.1 Create basic TF-M based application

Follow these steps to create a new PSOC™ Control C3M8 MCU: Basic TF-M application [7]

1. Create PSOC™ Control C3M8 MCU: Basic TF-M application for `KIT_PSC3M8_EVK` as shown below. The example is available under the **Security** section of the code example category

***Note**: Windows has a 260-character path length limit. A longer path to the workspace may exceed this limit and cause build failure with TF-M application. Ensure that the path `<Workspace_Path>\<TF-M_app_name>\` (including back slashes) must be less than or equal to 32-characters. For example, `C:\TF-M_workspace` can be used as workspace path and TF-M application name can be set as `TF-M`.*

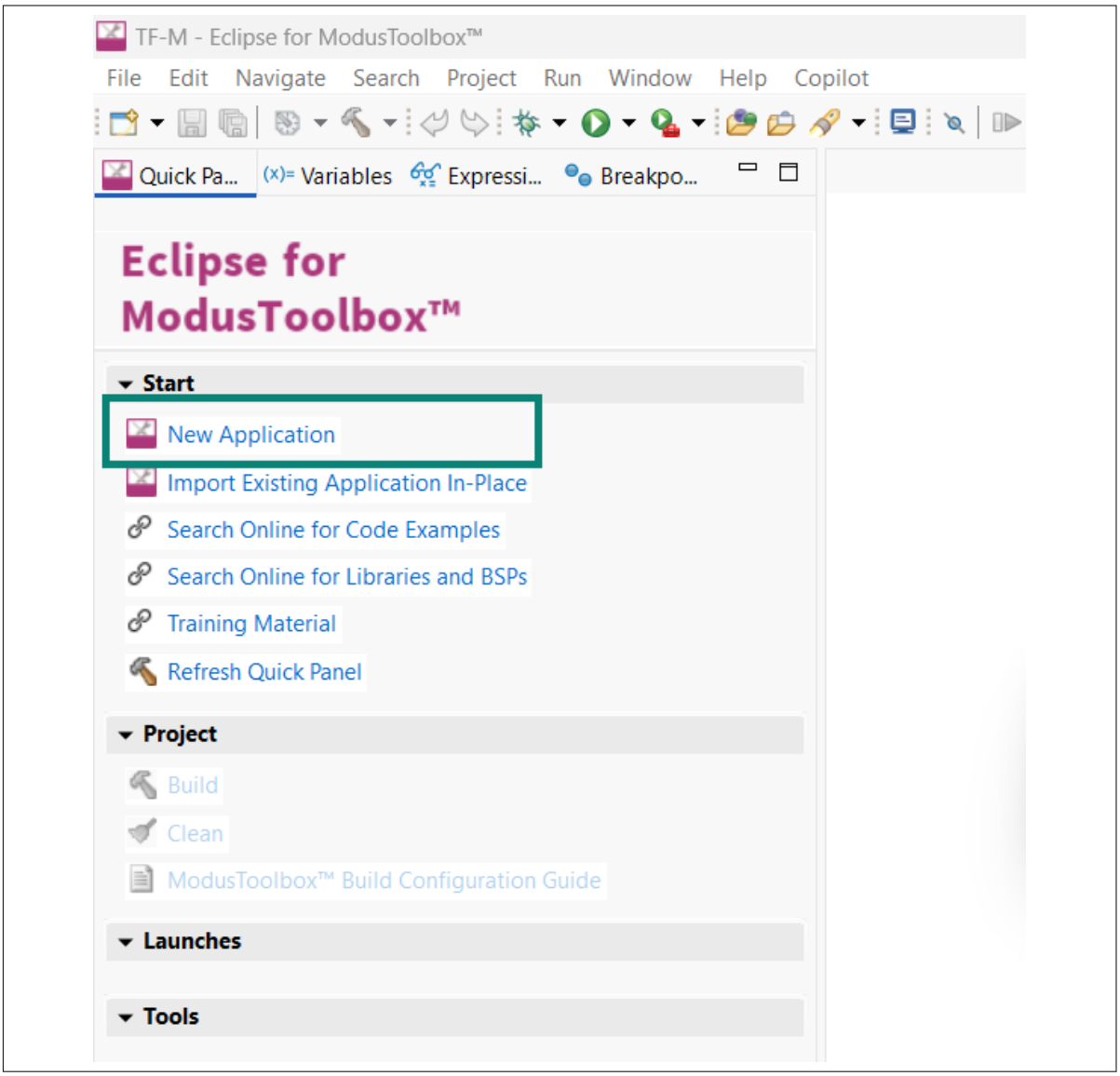


Figure 3 Select create new application

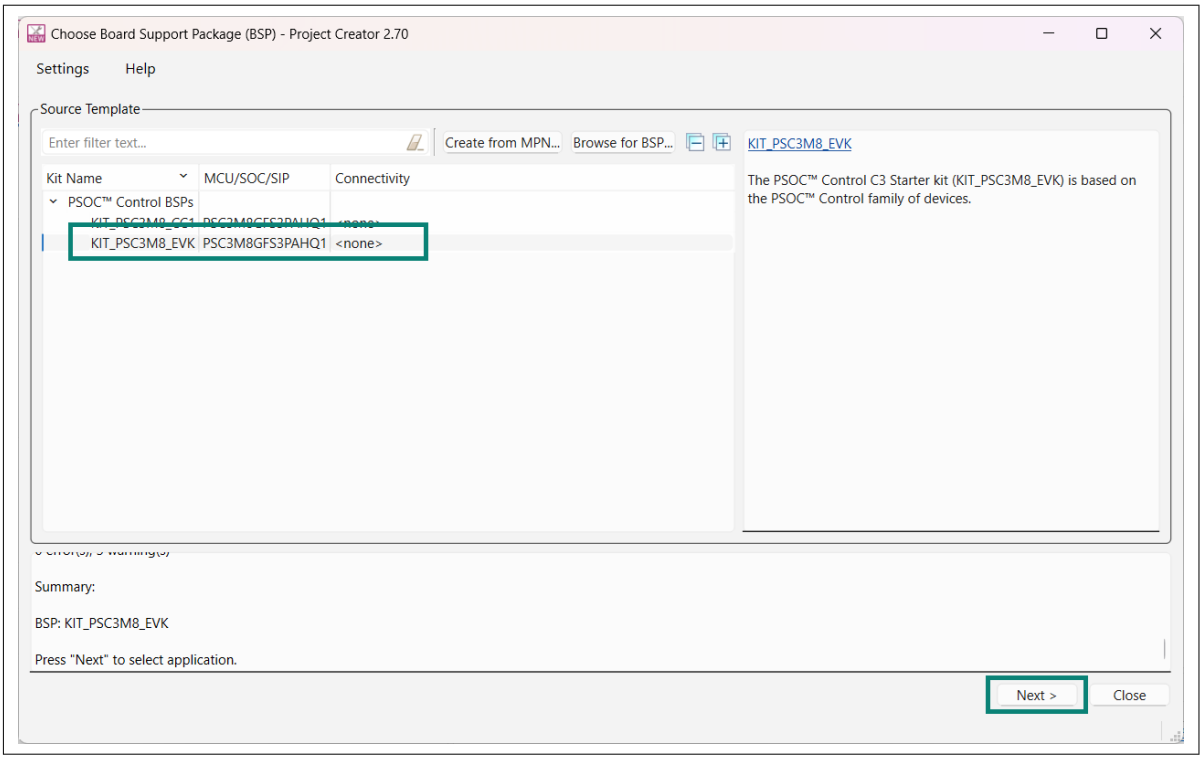


Figure 4 Select BSP for PSOC™ Control MCU

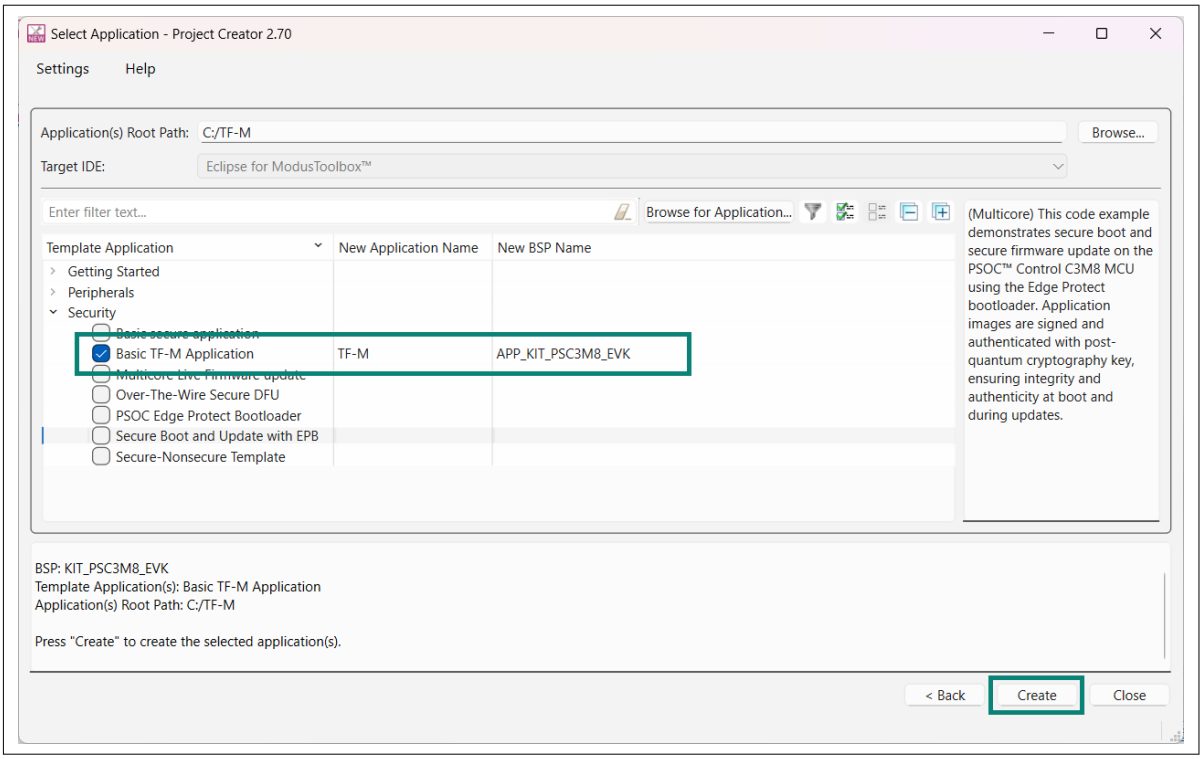


Figure 5 Creation of Basic TF-M application for PSOC™ Control MCU

2. After creating the TF-M application, install the Python dependencies required to build TF-M. The requirements are listed in `mtb_shared/ifx-tf-m/release-vx.y.z/tools/requirements.txt`. Execute the following command to install the dependencies

```
pip install -r <path_to_mtb_shared>/ifx-tf-m/release-vx.y.z/tools/requirements.txt
```

2.3.2 Add Edge Protect Bootloader to TF-M application

The TF-M-based application is designed to work together with the Edge Protect Bootloader. The Edge Protect Bootloader is available as a separate code example with the title PSOC™ Edge Protect Bootloader in ModusToolbox™. Follow the steps provided below to integrate Edge Protect Bootloader in the TF-M application.

1. Create the PSOC™ Edge Protect Bootloader application for `KIT_PSC3M8_EVK` BSP that is also used for creating TF-M application as shown below. The example is available under the **Security** section of the code example category

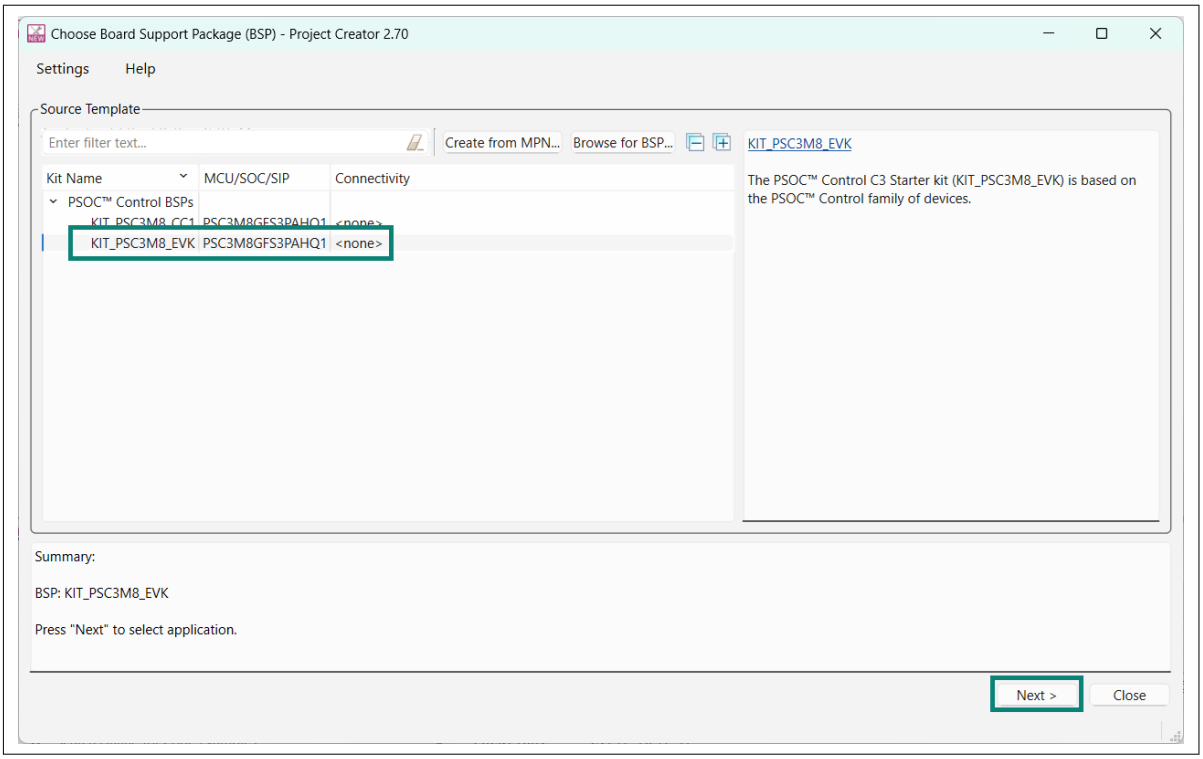


Figure 6 Select BSP for PSOC™ Control MCU

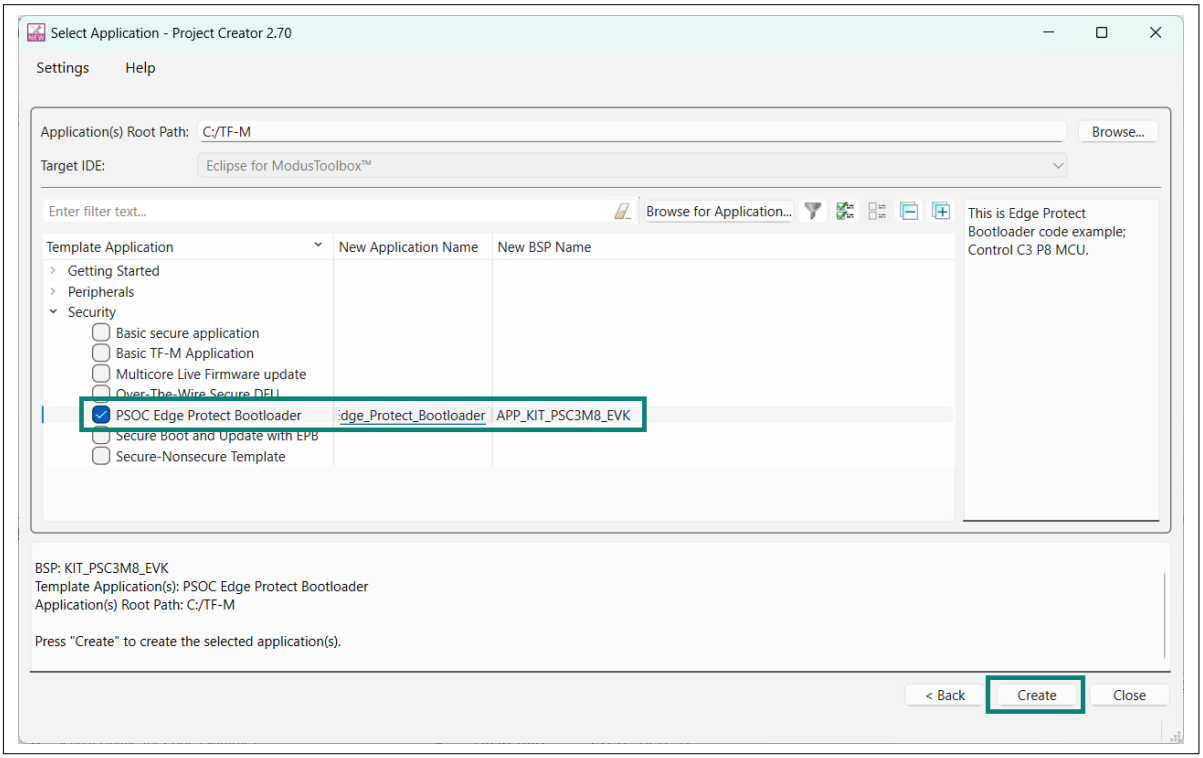


Figure 7 Creation of Edge Protect Bootloader application

Verify that both the Edge Protect Bootloader and the TF-M application are created successfully and visible in the IDE Project Explorer.

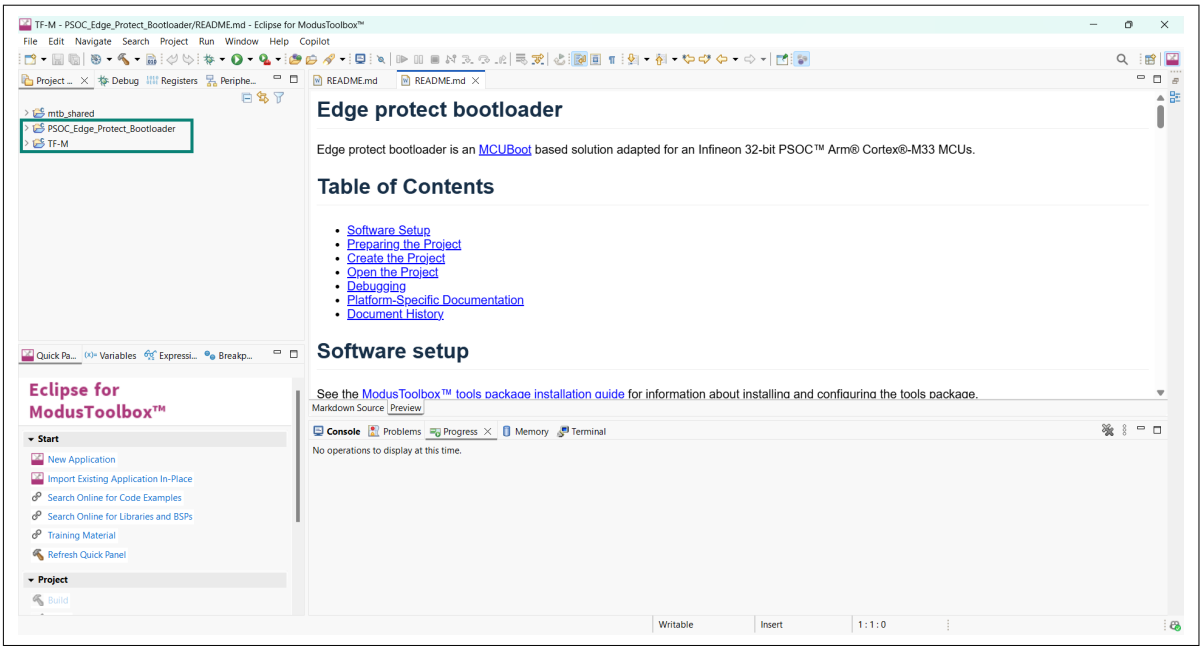


Figure 8 Eclipse IDE Project Explorer

2. Open `PSOC_Edge_Protect_Bootloader/common.mk` and update `MEMORY_MAP` variable to the path of the memory map JSON file `overwrite_multi4_flash.json` provided with TF-M code example in `TF-M/memory_map` folder and save the file

```
MEMORY_MAP?=../../TF-M/memory_map/overwrite_multi4_flash.json
```

2.3.3 Image signing and programming

This code example configures the Edge Protect Bootloader (EPB) to authenticate four (TF-M, main M33 non-secure, PPCA0 and PPCA1) signed images. The EPB supports signing/verification with ECDSA-256/384/521, LMS_SHA256_M32_H10, XMSS_SHA2_10_256, and ML-DSA-87 algorithms. The code example uses ECDSA521 as the default. The instructions below show how to generate signing keys and configure the Edge Protect Bootloader (EPB) for image authentication.

1. Generate ECDSA521 key pair for EPB signing using Edge Protect Tools. Edge Protect Tools is a command-line tool used to perform the functions needed for key signing, key generation, OEM certificate creation, device provisioning, and so on. The tool is executed through a shell:

```
edgeprotecttools create-key --key-type ECDSA-P521 -o keys/oem_private_key_0.pem keys/
oem_public_key_0.pem
```

2. Open `TF-M/common.mk` and update `IMAGE_SIGNING_KEY` variable to the path of your generated private key file and save the file

```
IMAGE_SIGNING_KEY=<key_path>/keys/oem_private_key_0.pem
```

3. Open `PSOC_Edge_Protect_Bootloader/platforms/PSC3_P8/feature_config.json` and set the following options:
   - Set "validation_key_type" to "ECDSA-521"
   - Set "validation_key" to generated public key path `"<key_path>/keys/oem_public_key_0.pem"`

- Set "validate_boot" to true
- Set "validate_upgrade" to true

After completing the configuration, build and program the EPB and TF-M application. Select the EPB and TF-M application in Project Explorer and click on Program application from Quick Panel to build and program applications as shown in the below figure.

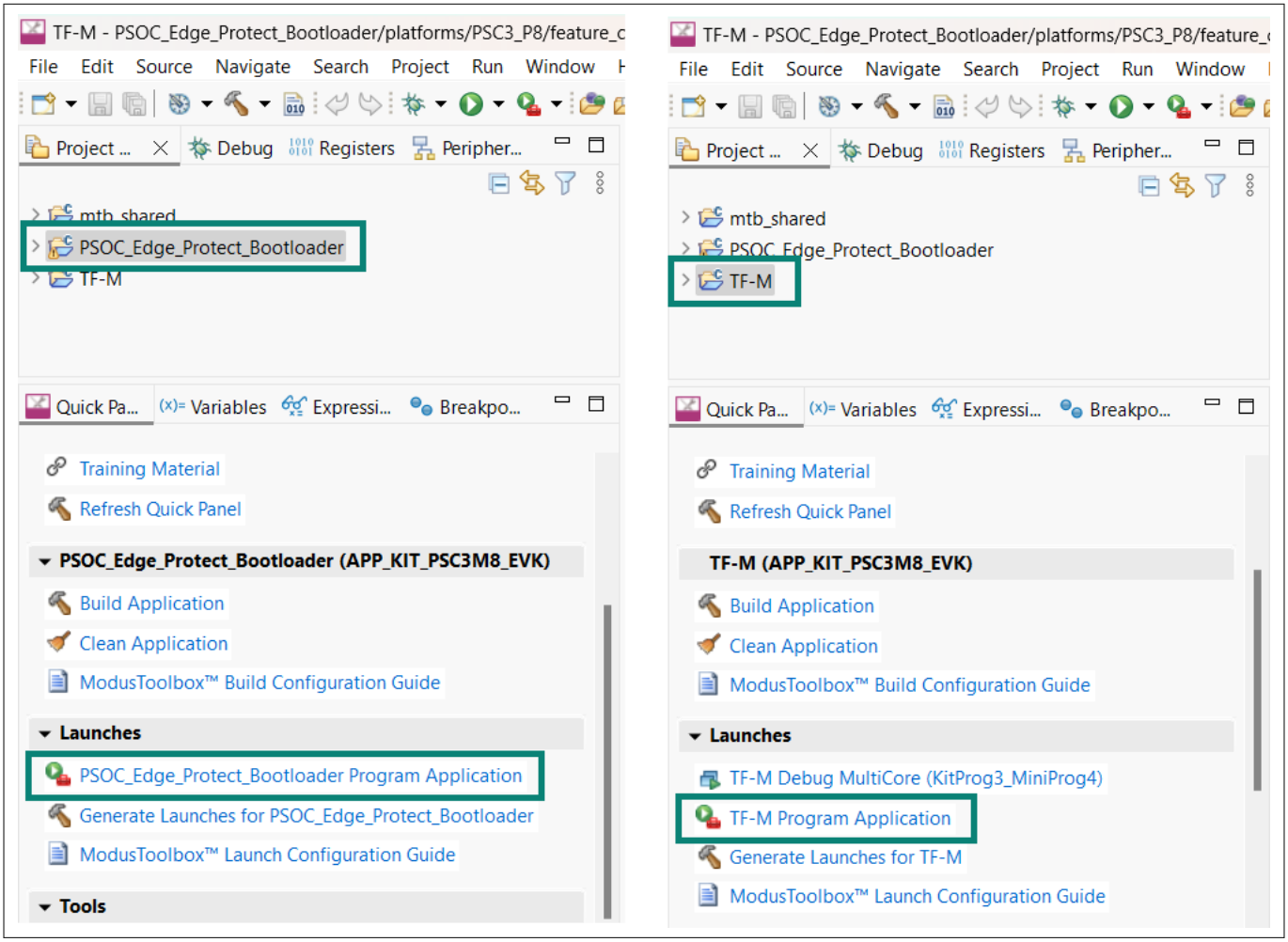


Figure 9 Build and Program TF-M application

After successfully programming the application, the BootROM launches the EPB. The EPB verifies the signatures of the next four images (TF-M, main M33 non-secure, PPCA0 and PPCA1), and launches the TF-M image. TF-M initializes the SPE and transfers control to the main M33 NSPE. The main M33 NSPE enables PPCA 0/1. Upon successful launch, you will see the following logs on the serial terminal as shown in Figure 10:

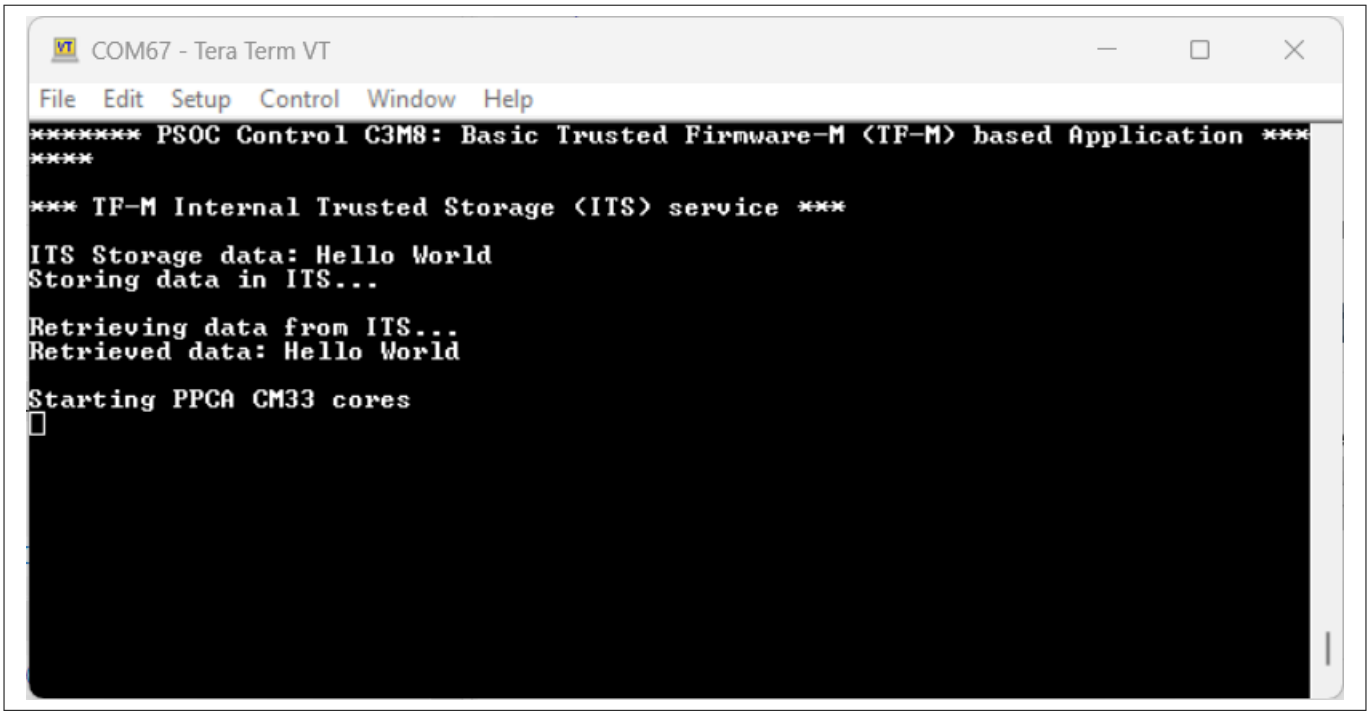


Figure 10 PSOC™ Control basic TF-M application logs

2.4 Application structure and build artifacts

2.4.1 Application structure

The PSOC™ Control basic TF-M application consists of four projects:

- main_cm33_s
- main_cm33_ns
- ppca_cm33_0
- ppca_cm33_1

main_cm33_s: This is a CM33 secure project that contains the TF-M. TF-M is available in source code format and is placed in the `ifx-tf-m` library that can be managed using the Library Manager offered by ModusToolbox™. The library is placed in the `<Workspace_Path>/mtb_shared` directory and functions as an independent, self-contained application. It contains all the necessary components and dependencies required to operate independently. The `main_cm33_s` project directory contains only the essential Makefile and dependencies; it does not contain any source code. Any source code placed in this directory will be excluded from the build process. Instead, ModusToolbox™ compiles the TF-M application provided within the `ifx-tf-m` library.

main_cm33_ns: This is the main CM33 non-secure project. This project has the TF-M NSPE interface and can call PSA APIs to request secure services from TF-M. The TF-M interface is available in a separate `ifx-tf-m-ns` library that is placed in `<Workspace_Path>/mtb_shared` directory. The interface must be initialized by the main CM33 non-secure project before requesting service from TF-M.

ppca_cm33_0: This is the PPCA 0 non-secure project. This project does not have the TF-M NSPE interface, that is, this project cannot request services from TF-M.

ppca_cm33_1: This is the PPCA 1 non-secure project. This project does not have the TF-M NSPE interface, that is, this project cannot request services from TF-M.

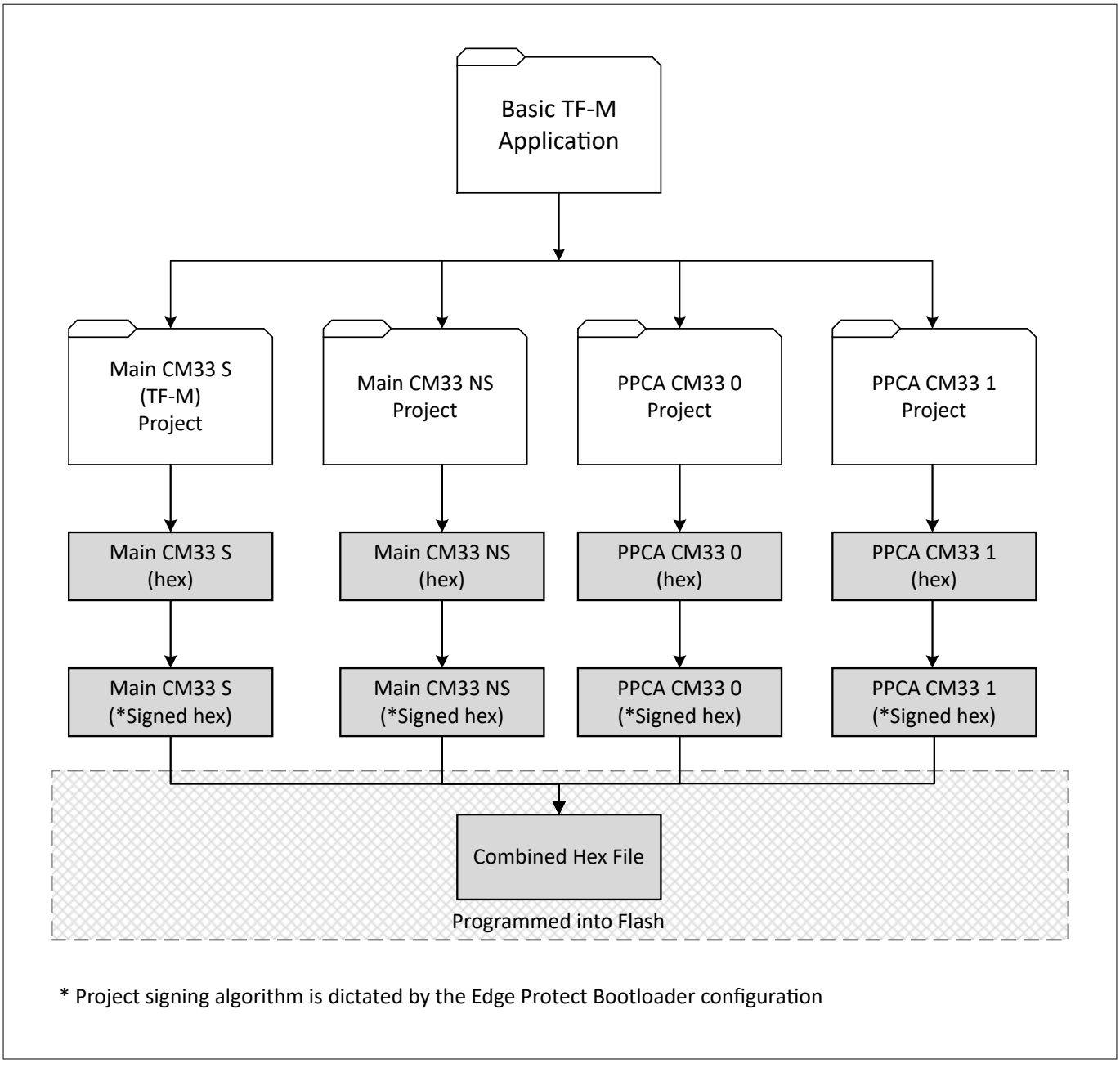


Figure 11 TF-M application structure

2.4.2 Build artifacts

The build artifacts (hex, bin, elf, etc.) of each project of PSOC™ Edge basic TF-M application are placed into their individual project directories.

- **main_cm33_s:** When building TF-M, the normal ModusToolbox™ application build flow is bypassed while building TF-M (main_cm33_s). Instead, TF-M uses `CMake` as the build system generator and Ninja as the build system. The output of `CMake` is placed in the `main_cm33_s/build/APP_KIT_PSC3M8_EVK/Debug` directory. The build system compiles the TF-M source code from the `mtb-shared/ifx-tf-m` directory and generates the `tfm_s.hex`, `tfm_s.bin`, `tfm_s.map`, `tfm_s.elf`, and `tfm_s.axf` artifact files and places them in the `main_cm33_s/build/APP_KIT_PSC3M8_EVK/Debug/bin` directory. The object files (.o) and library files (.a) are placed in `main_cm33_s/build/APP_KIT_PSC3M8_EVK/Debug` directory. The `tfm_s.hex`, `tfm_s.bin` and `tfm_s.elf` artifacts are copied to the `main_cm33_s/build/APP_KIT_PSC3M8_EVK/Debug` directory and renamed as `main_cm33_s.hex`, `main_cm33_s.bin` and `main_cm33_s.elf` by TF-M build system.

The TF-M build system also installs the essential non-secure interface files in `<TF-M_app_name>/install`. These files enable a non-secure interface to request secure services from TF-M. As part of the build process, ModusToolbox™ copies the `main_cm33_s.hex` to `<TF-M_app_name>/build/project_hex` directory

- **main_cm33_ns, ppca_cm33_0 and ppca_cm33_1:** Like any other ModusToolbox™ application, the artifacts of the build process are placed in their respective `build/APP_KIT_PSC3M8_EVK/Debug` directories and the hex files are copied to `<TF-M_app_name>/build/project_hex` directory

After building all four projects, ModusToolbox™ looks for all four `<APPNAME>.hex`, signs the individual images and merges them to create a `app_combined.hex` file, which is the combination of all four projects. The signing and merging of all four projects is governed by `signer_combiner.json` that is located in `<TF-M_app_name>/configs` directory.

2.5 SPE logging

TF-M provides an option to log status, exceptions, and faults from SPE that are useful to debug the application. The logs can be enabled by adding the below configuration in `<Workspace_Path>/<TF-M_app_name>/main_cm33_s/Makefile`:

```
TFM_CONFIGURE_EXT_OPTIONS+= -DTFM_EXCEPTION_INFO_DUMP=ON -DPLATFORM_EXCEPTION_INFO=ON
-DIFX_FAULTS_INFO_DUMP=ON -DTFM_SPM_LOG_LEVEL=LOG_LEVEL_VERBOSE
-DTFM_PARTITION_LOG_LEVEL=LOG_LEVEL_VERBOSE
```

When the logs are enabled, TF-M logs all the Secure Partition Manager (SPM) and all other partition's status, faults, and exceptions. TF-M dumps the exception stack frame, type of fault, the bus master that caused the fault, etc. through UART. For example, if the M33 NSPE tries to change the protection context (PC) of the M33 CPU, it leads to a PPC fault that will be logged on the UART terminal. By default, SPE logs are enabled in the code example.

Serial Communication Block 2 (SCB2) UART lines are connected to KitProg3 on the KIT_PSC3M8_EVK Evaluation Kit; therefore, it is used for logging by TF-M. When logging is enabled in TF-M, SCB2 must be assigned to the M33 secure (M33S) domain, that is, it is marked as secure and cannot be accessed directly by NSPEs. This configuration is set from the Peripheral Protection Controller (PPC) configuration, available under the System tab in Device Configurator. It is possible to use some other instance of SCB for logging. To enable logging in such cases:

1. Enable the desired SCB instance with the `IFX_TFM_SPM_UART` alias and set the personality to UART. Configure the clock and Rx/Tx lines accordingly
2. Update the Peripheral Protection Controller (PPC) protection settings of the new SCB instance by assigning it to the M33S domain through PPC configuration in the device configurator. Ensure that Rx/Tx pins have a secure attribute set to Secure access

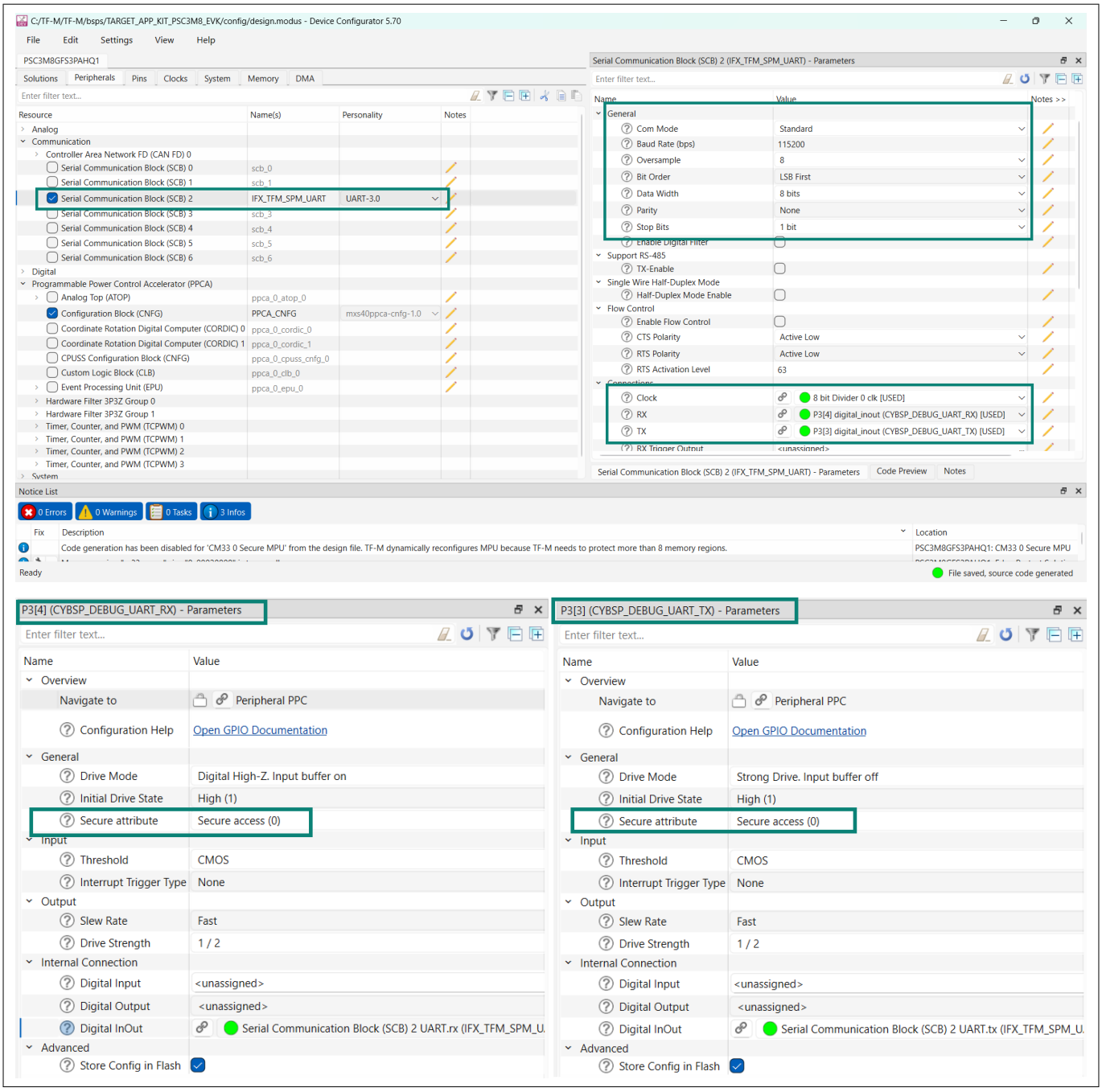


Figure 12 Logging SCB configuration

It is important to note that the secure SCB cannot be accessed by NSPE for logging or communication. If accessed, it will result in a fault. To prevent such faults, users can choose any of the following options:

- **Use platform service (for logging):** TF-M offers a platform service for message logging. The logging service transfers the message from NSPE to TF-M (SPE) where TF-M uses the secure SCB2 to log the messages. This is the default logging method used in the TF-M code example. See the Platform service section for instruction on how to use the platform partition's logging service
- **Use other SCB (for logging and communication):** Use any other SCB than secure SCB for NSPE logging or communication. Configure the new SCB with the desired personality (UART/I2C/SPI) along with its associated Rx/Tx pins. The SCB must be assigned to a non-secure domain (M33_M55 domain) with PPC configuration (available under System tab in Device Configurator) and the pin's secure attribute must be set as non-secure access to make it accessible to NSPEs

See ModusToolbox™ Device Configurator user guide for more information about PPC configurations in Device Configurator.

3 Add TF-M to an existing application

TF-M can be added to an existing application using Library Manager provided by ModusToolbox™. Adding TF-M to an existing project involves the following steps:

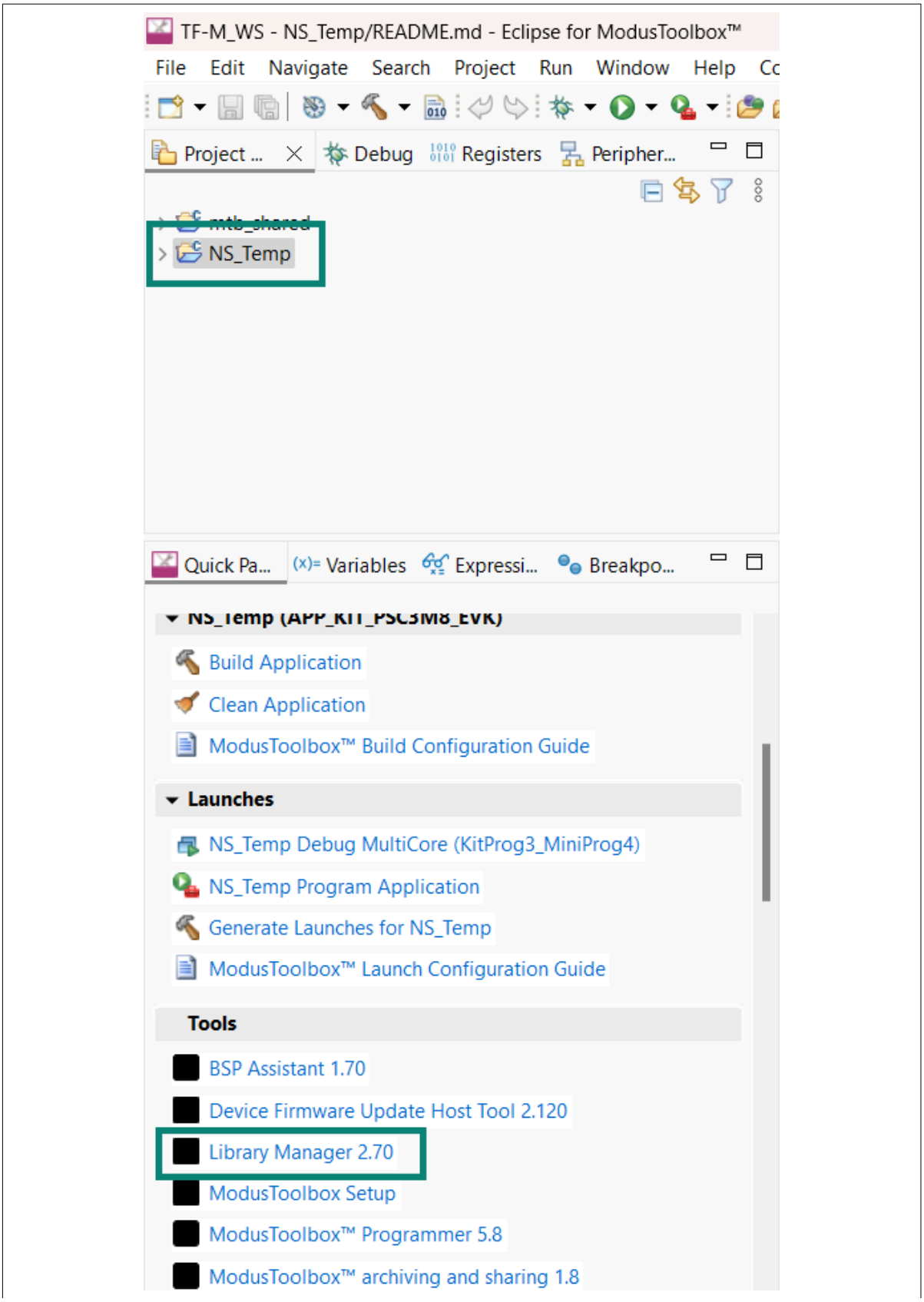
- Add TF-M libraries to the application
- Update the application's memory and protection configuration
- Add and configure Edge Protect Bootloader
- Initialize the TF-M interface in NSPE

***Note**: After adding the TF-M library, all source code placed in the `main_cm33_s` directory will be ignored by the build process. Instead the TF-M source code available in the `ifx-tf-m` library is built.*

3.1 Add TF-M libraries to the application

TF-M source code is available in the `ifx-tf-m` library and the NSPE interface is provided by the `ifx-tf-m-ns` library. The following steps show how to add TF-M to PSOC™ Control C3M/P8 MCU: Secure-Non-Secure template code example [9], but the same steps are applicable to other applications.

1. Select your application and open Library Manager


TF-M_WS - NS_Temp/README.md - Eclipse for ModusToolbox™
File Edit Navigate Search Project Run Window Help
Project ...
Debug
Registers
Peripher...
mtb_shared
NS_Temp
Quick Pa...
Variables
Expressi...
Breakpo...
Build Application
Clean Application
ModusToolbox™ Build Configuration Guide
Launches
NS_Temp Debug MultiCore (KitProg3_MiniProg4)
NS_Temp Program Application
Generate Launches for NS_Temp
ModusToolbox™ Launch Configuration Guide
Tools
BSP Assistant 1.70
Device Firmware Update Host Tool 2.120
Library Manager 2.70
ModusToolbox Setup
ModusToolbox™ Programmer 5.8
ModusToolbox™ archiving and sharing 1.8

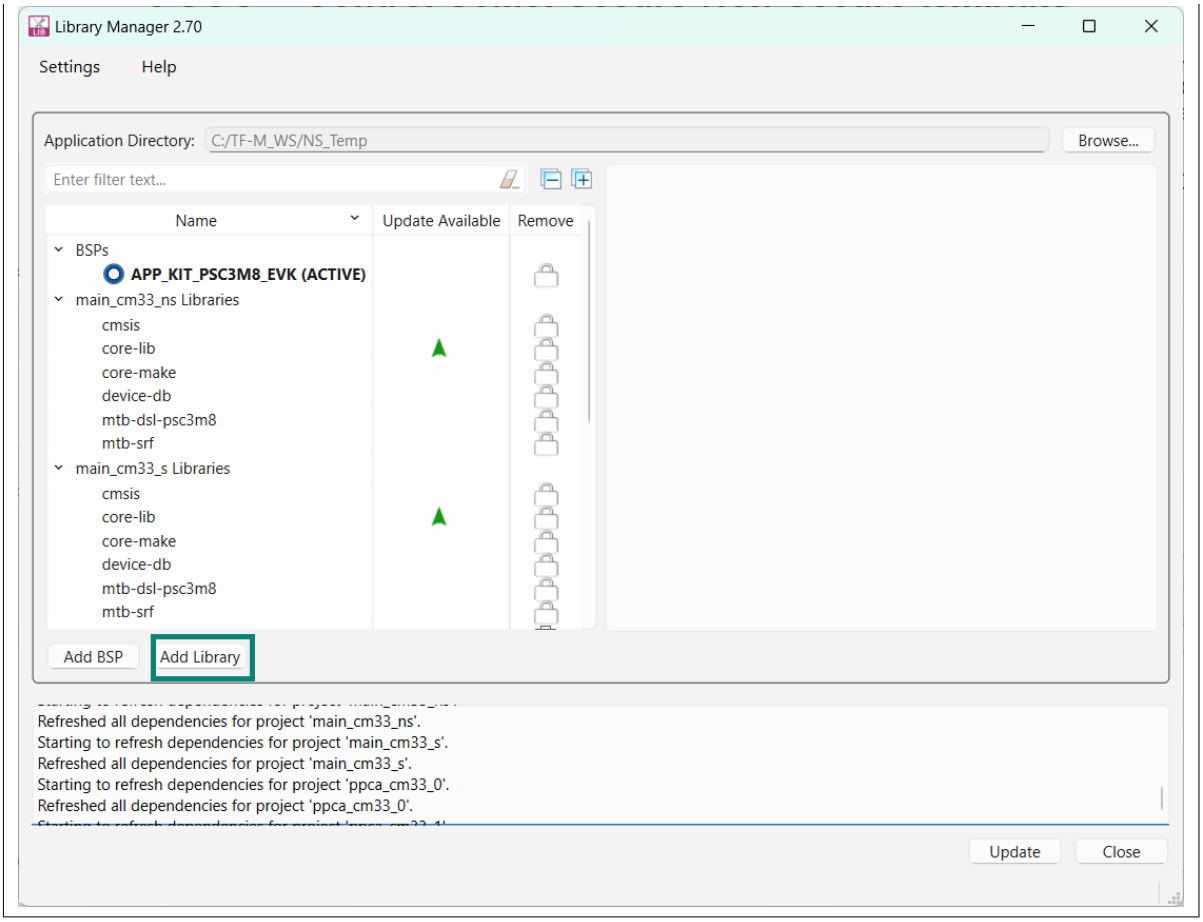


Figure 13 Open Library Manager

2. Add `ifx-tf-m` library to `main_cm33_s` project as per target device

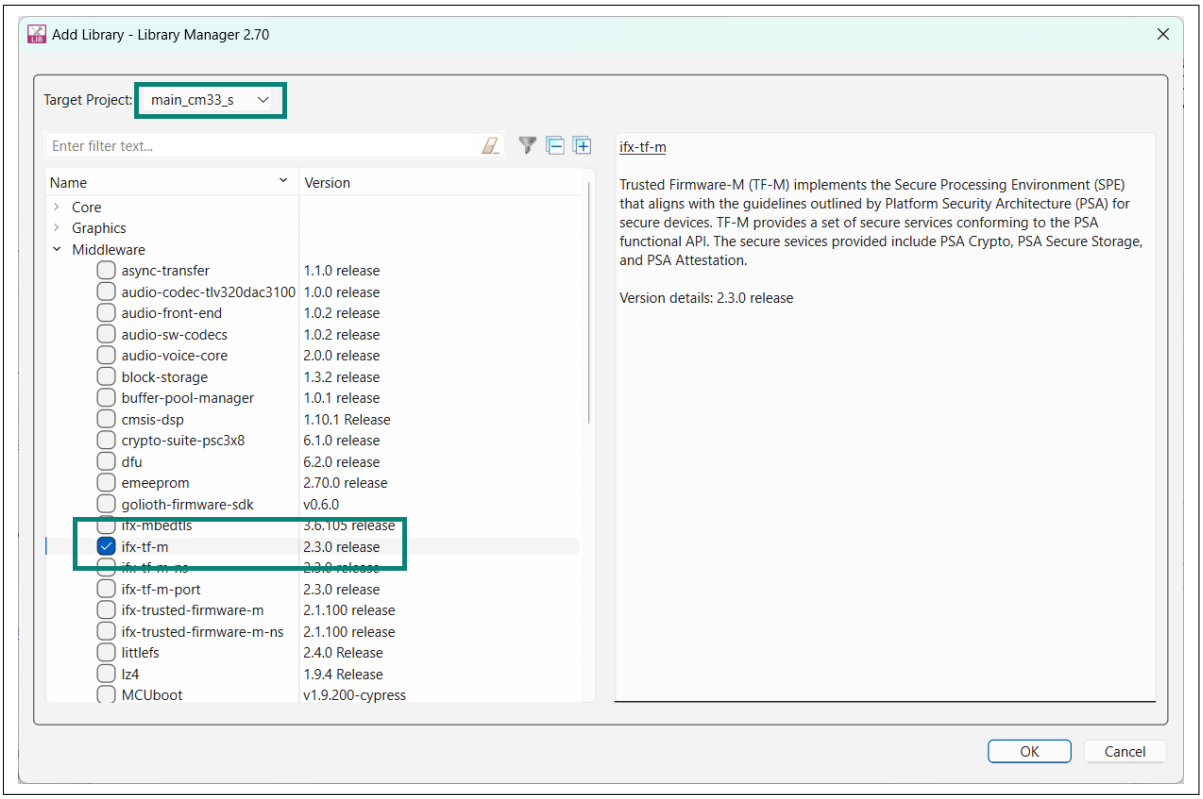


Figure 14 Add TF-M library

3. Add TF-M NS library to `main_cm33_ns` project (M33 NSPE)

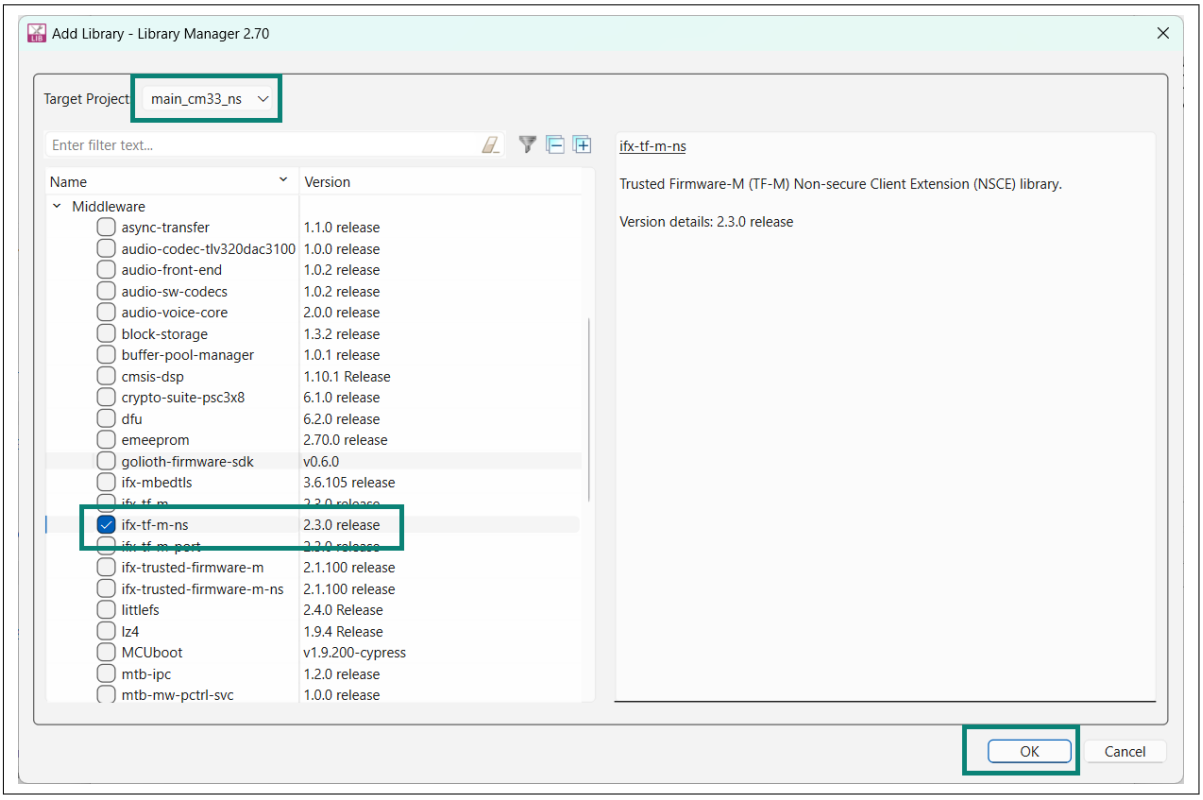


Figure 15 Add TF-M to M33 NSPE

4. Update the application after adding the necessary libraries

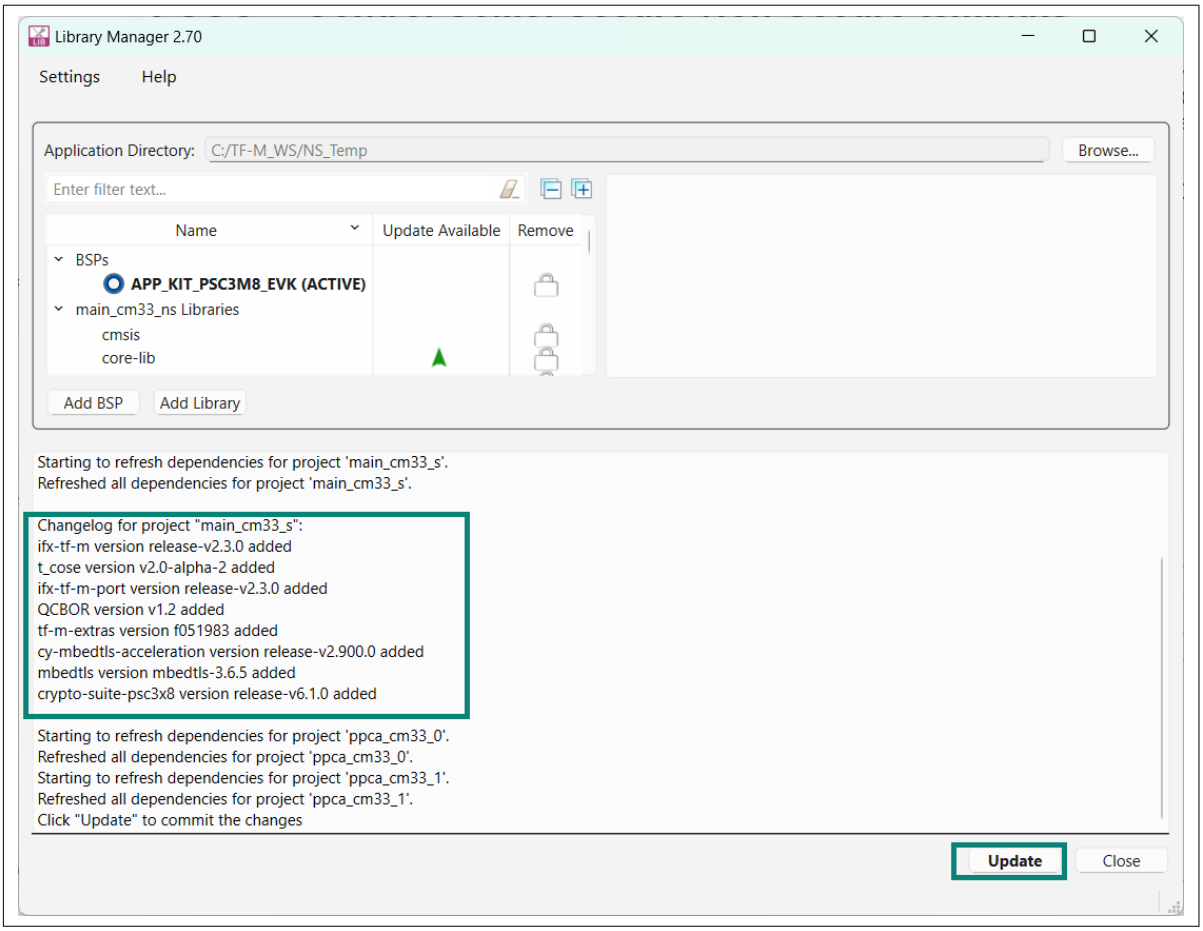


Figure 16 Update application libraries

5. Install the Python dependencies required to build TF-M. The requirements are listed in `mtb_shared/ifx-tf-m/release-vx.y.z/tools/requirements.txt`. Execute the following command to install the dependencies

```
pip install -r <path_to_mtb_shared>/ifx-tf-m/release-vx.y.z/tools/requirements.txt
```

6. Update the NSC veneer path in `NS_Temp/common.mk`

```
NSC_VENEER=../install/interface/lib/s_veneers.o
```

7. Update the attestation boot data buffer size in TF-M. Open `mtb_shared/ifx-tf-m/release-vx.y.z/secure_fw/partitions/initial_attestation/attest_boot_data.c` and update `MAX_BOOT_STATUS` to 1024

3.2 Update memory and protection configuration

After adding TF-M libraries, the next step is to add the necessary memory regions and update the protection configuration. This can be achieved with Edge Protect Configurator.

1. Select your application and open Edge Protect Configurator

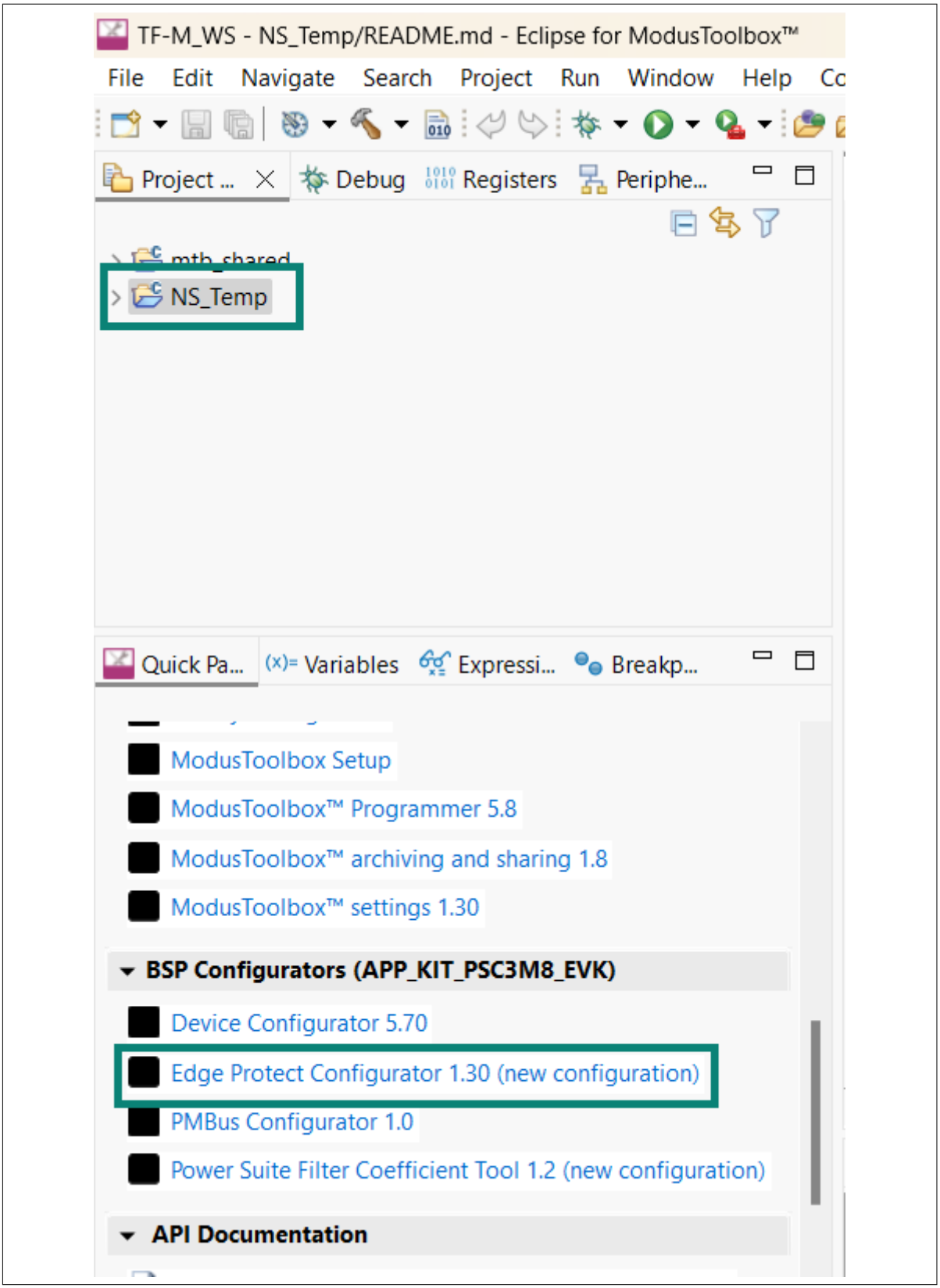


Figure 17 Open Edge Protect Configurator

2. After opening the Edge Protect Configurator, the Edge Protect Configurator provides an option to add the default memory regions and update security configurations for TF-M. Select yes to update the application with these changes. Save the changes and close Edge Protect Configurator

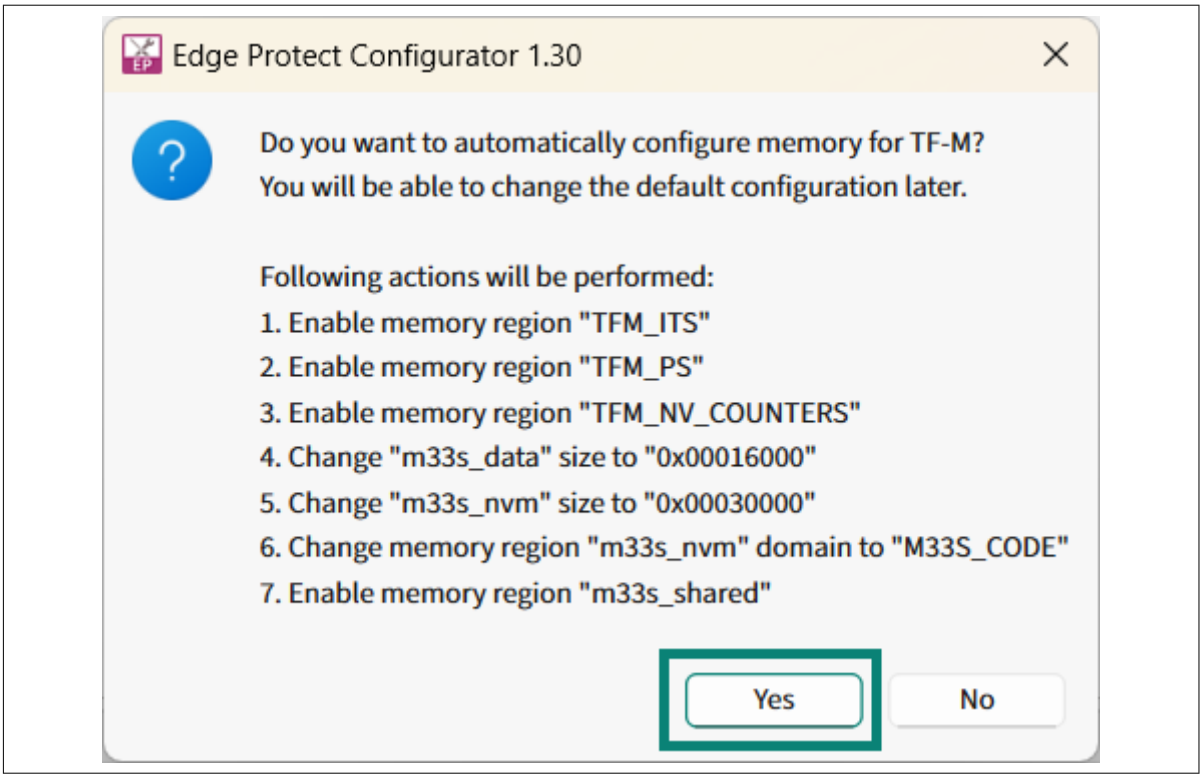


Figure 18 Update application with Edge Protect Configurator

3. Optionally you can select which TF-M profile, isolation level, and partitions will be enabled. Additionally, user-defined Application RoT partitions can be created from Edge Protect Configurator

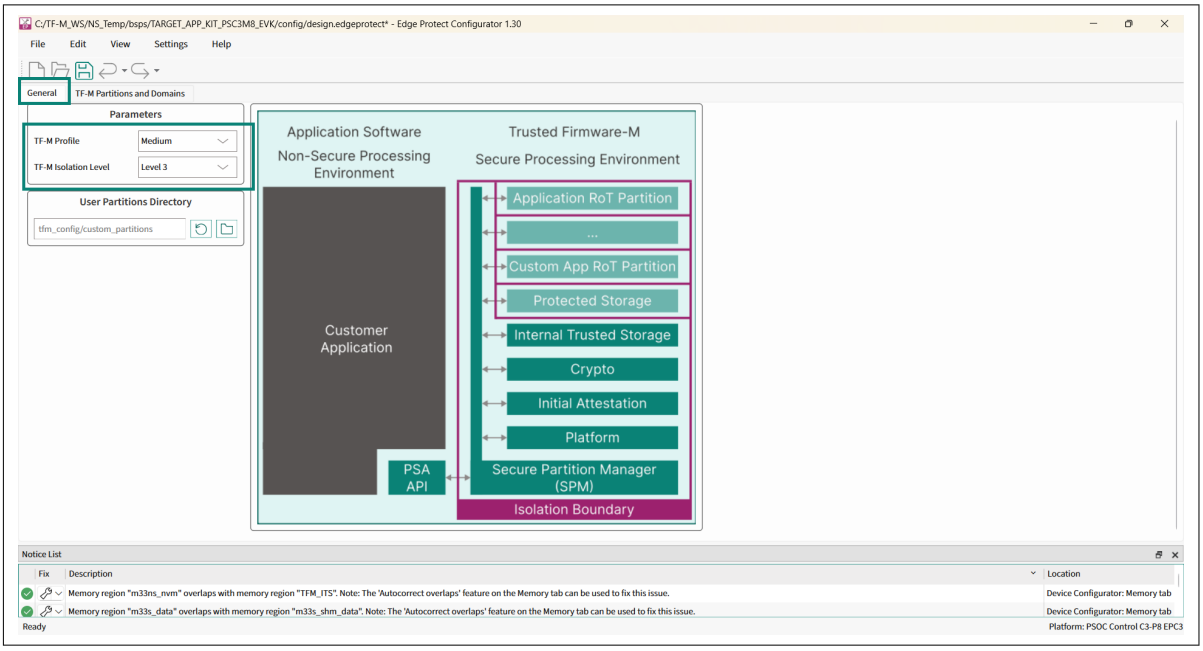


Figure 19 Change TF-M Profile and Isolation Level

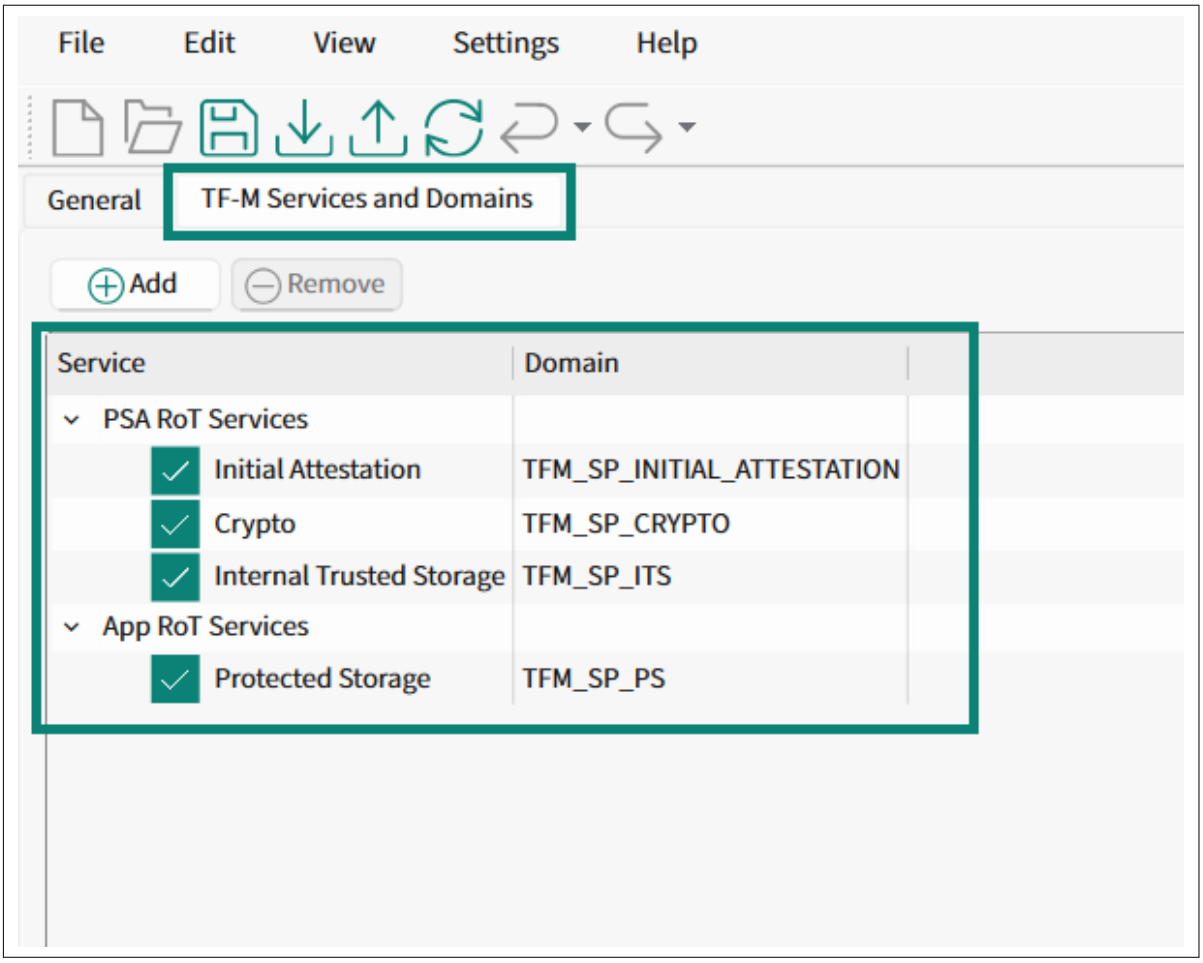


Figure 20 Enable/Disable TF-M Partitions

4. Open device Configurator from Quick Panel and select fix PPC configuration

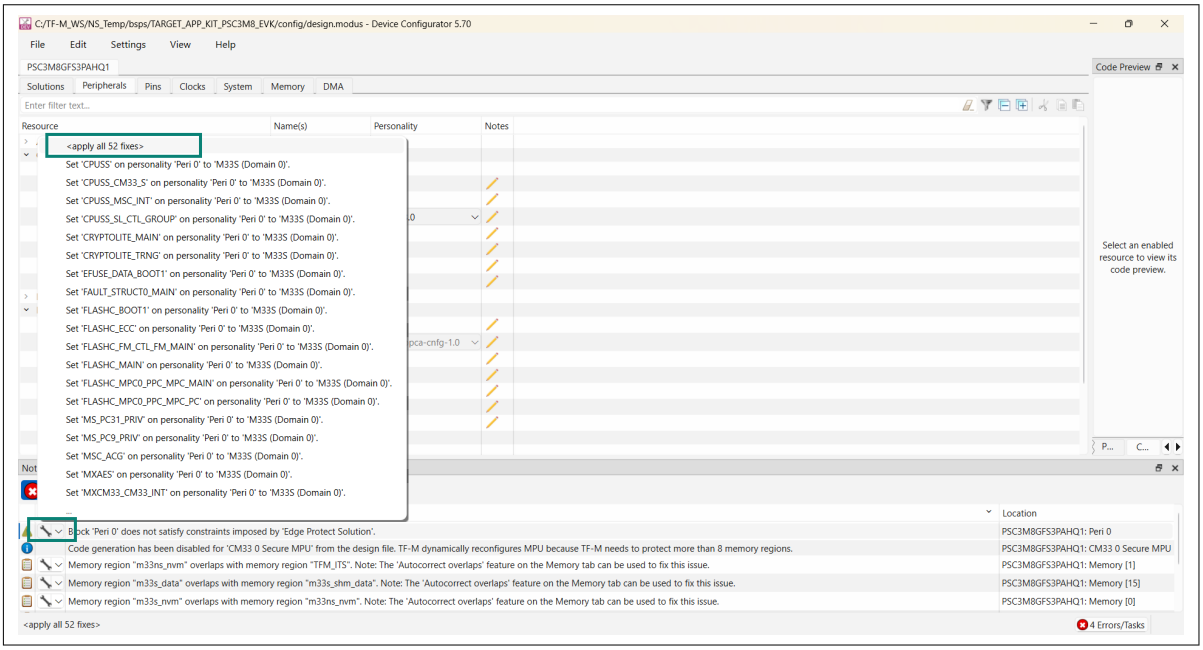


Figure 21 Fix PPC configuration

5. Navigate to the memory tab in Device Configurator and update the memory map as shown below

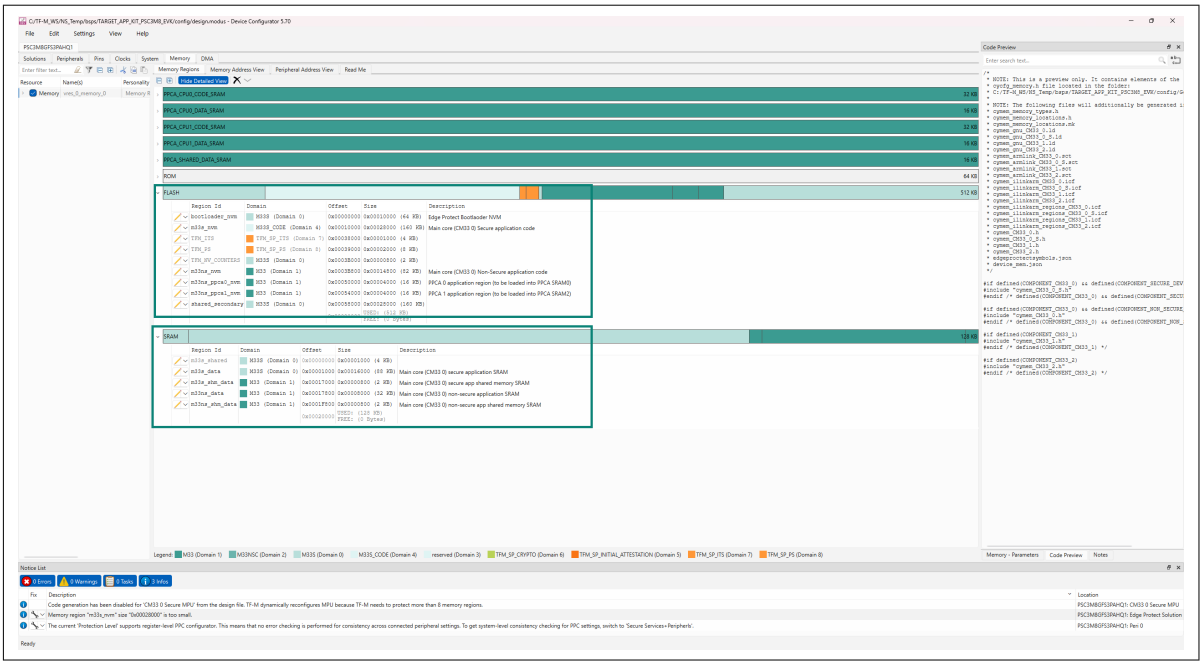


Figure 22 Update Memory map

3.3 Add Edge Protect Bootloader

1. Create Edge Protect Bootloader application as described in Add Edge Protect Bootloader to TF-M application

2. By default, EPB uses `PSOC_Edge_Protect_Bootloader/platforms/PSC3_P8/memory_maps/overwrite_single_flash.json` memory map. Update the EPB memory map configuration to align with the TF-M application's memory map. A reference memory map is provided below

```
{
    "bootloader":
    {
        "bootloader_area":
        {
            "address"           : "0x32000000",
            "size"              : "0x10000"
        },

        "ram":
        {
            "address"           : "0x34001000",
            "size"              : "0xE000"
        },

        "shared_data":
        {
            "address"           : "0x34000000",
            "size"              : "0x1000"
        }
    },

    "application_1":
    {
        "slots":
        {
            "boot"              : "0x32010000",
            "upgrade"           : "0x32058000",
            "size"              : "0x28000"
        },

        "ram":
        {
            "address"           : "0x34001000",
            "size"              : "0xE000"
        }
    },

    "application_2":
    {
        "slots":
        {
            "boot"              : "0x3203B800",
            "upgrade"           : "0x32058400",
            "size"              : "0x14800"
        },

        "ram":
        {
```

```
            "address"           : "0x34001000",
            "size"              : "0xE000"
        }
    },

    "application_3":
    {
        "slots":
        {
            "boot"              : "0x32050000",
            "upgrade"           : "0x32058800",
            "size"              : "0x4000"
        },

        "ram":
        {
            "address"           : "0x34001000",
            "size"              : "0xE000"
        }
    },

    "application_4":
    {
        "slots":
        {
            "boot"              : "0x32054000",
            "upgrade"           : "0x32058C00",
            "size"              : "0x4000"
        },

        "ram":
        {
            "address"           : "0x34001000",
            "size"              : "0xE000"
        }
    }

}
```

3. Generate key pair for image signing/verification and update the `PSOC_Edge_Protect_Bootloader/platforms/PSC3_P8/feature_config.json` as described in Image signing and programming to enable image authentication
4. To sign the images with the above mentioned key-pair, update the `<Non-secure_Template_App>/configs/signer_combiner.json` as shown below

```
{
    "schema-version": 1.0,
    "content":
    [
        {
            "name": "Secure Image signing",
            "description": "Signs the input file and converts it to the MCUboot format",
            "enabled": true,
            "commands":
            [
                {
                    "command": "sign",
                    "inputs":
                    [
                        {
                            "description": "Secure application",
                            "file": "../build/project_hex/main_cm33_s.hex",
                            "signing-key": "../keys/oem_private_key_0.pem",
                            "header-size": "0x400",
                            "slot-size": "{{ CYMEM_CM33_0_S_m33s_nvm_SIZE }}",
                            "image-version": "1.0.0",
                            "fill-value": "0x00",
                            "min-erase-size": "0x200",
                            "signature-encoding": "asn1",
                            "pubkey-encoding": "der",
                            "pubkey-format": "hash",
                            "hex-address": "{{ CYMEM_CM33_0_S_m33s_nvm_S_START }}",
                            "align": "1",
                            "boot-record": "TF-M_image",
                            "pad": true,
                            "overwrite-only": true,
                            "security-counter" : "1"
                        }
                    ],
                    "outputs":
                    [
                        {
                            "description": "Signed secured application",
                            "file": "../build/project_hex/main_cm33_s_signed.hex",
                            "format": "ihex"
                        }
                    ],
                    "extra_config":
                    [
                        {
                            "project": "main_cm33_s",
                            "debug_config_name": "main_cm33_s",
```

```
                    "default": false,
                    "build_dependency": "project"
                }
            ]
        }
    ]
},
{
    "name": "Non-Secure Image signing",
    "description": "Signs the input file and converts it to the MCUboot format",
    "enabled": true,
    "commands":
    [
        {
            "command": "sign",
            "inputs":
            [
                {
                    "description": "Non-Secure application",
                    "file": "../build/project_hex/main_cm33_ns.hex",
                    "signing-key": "../keys/oem_private_key_0.pem",
                    "header-size": "0x400",
                    "slot-size": "{{ CYMEM_CM33_0_S_m33ns_nvm_SIZE }}",
                    "image-version": "1.0.0",
                    "fill-value": "0x00",
                    "min-erase-size": "0x200",
                    "signature-encoding": "asn1",
                    "pubkey-encoding": "der",
                    "pubkey-format": "hash",
                    "hex-address": "{{ CYMEM_CM33_0_S_m33ns_nvm_S_START }}",
                    "align": "1",
                    "boot-record": "TF-M_NSPE",
                    "pad": true,
                    "overwrite-only": true,
                    "security-counter" : "1"
                }
            ],
            "outputs":
            [
                {
                    "description": "Signed non-secure application",
                    "file": "../build/project_hex/main_cm33_ns_signed.hex",
                    "format": "ihex"
                }
            ],
            "extra_config":
            [
                {
                    "project": "main_cm33_ns",
                    "debug_config_name": "main_cm33_ns",
                    "default": false,
                    "build_dependency": "project"
                }
```

```
                ]
            }
        ]
    },
    {
        "name": "PPCA_0 Image signing",
        "description": "Signs the input file and converts it to the MCUboot format",
        "enabled": true,
        "commands":
        [
            {
                "command": "sign",
                "inputs":
                [
                    {
                        "description": "PPCA0 Non-Secure application",
                        "file": "../build/project_hex/ppca_cm33_0.hex",
                        "signing-key": "../keys/oem_private_key_0.pem",
                        "header-size": "0x400",
                        "slot-size": "{{ CYMEM_CM33_0_S_m33ns_ppca0_nvm_SIZE }}",
                        "image-version": "1.0.0",
                        "min-erase-size": "0x200",
                        "signature-encoding": "asn1",
                        "pubkey-encoding": "der",
                        "pubkey-format": "hash",
                        "hex-address": "{{ CYMEM_CM33_0_S_m33ns_ppca0_nvm_S_START }}",
                        "align": "1",
                        "boot-record": "PPCACM330",
                        "pad": true,
                        "overwrite-only": true,
                        "security-counter" : "1"
                    }
                ],
                "outputs":
                [
                    {
                        "description": "Signed secured application",
                        "file": "../build/project_hex/ppca_cm33_0_signed.hex",
                        "format": "ihex"
                    }
                ],
                "extra_config":
                [
                    {
                        "project": "ppca_cm33_0",
                        "debug_config_name": "ppca_cm33_0",
                        "default": false,
                        "build_dependency": "project"
                    }
                ]
            }
        ]
    },
```

```
        {
            "name": "PPCA_1 Image signing",
            "description": "Signs the input file and converts it to the MCUboot format",
            "enabled": true,
            "commands":
            [
                {
                    "command": "sign",
                    "inputs":
                    [
                        {
                            "description": "PPCA1 Non-Secure application",
                            "file": "../build/project_hex/ppca_cm33_1.hex",
                            "signing-key": "../keys/oem_private_key_0.pem",
                            "header-size": "0x400",
                            "slot-size": "{{ CYMEM_CM33_0_S_m33ns_ppca1_nvm_SIZE }}",
                            "image-version": "1.0.0",
                            "min-erase-size": "0x200",
                            "signature-encoding": "asn1",
                            "pubkey-encoding": "der",
                            "pubkey-format": "hash",
                            "hex-address": "{{ CYMEM_CM33_0_S_m33ns_ppca1_nvm_S_START }}",
                            "align": "1",
                            "boot-record": "PPCACM331",
                            "pad": true,
                            "overwrite-only": true,
                            "security-counter" : "1"
                        }
                    ],
                    "outputs":
                    [
                        {
                            "description": "Signed secured application",
                            "file": "../build/project_hex/ppca_cm33_1_signed.hex",
                            "format": "ihex"
                        }
                    ],
                    "extra_config":
                    [
                        {
                            "project": "ppca_cm33_1",
                            "debug_config_name": "ppca_cm33_1",
                            "default": false,
                            "build_dependency": "project"
                        }
                    ]
                }
            ]
        },
        {
            "name": "merge",
            "enabled": true,
            "commands":
```

```
            [
                {
                    "command": "merge",
                    "inputs":
                    [
                        {
                            "file": "./../build/project_hex/main_cm33_s_signed.hex"
                        },
                        {
                            "file": "./../build/project_hex/main_cm33_ns_signed.hex"
                        },
                        {
                            "file": "./../build/project_hex/ppca_cm33_0_signed.hex"
                        },
                        {
                            "file": "./../build/project_hex/ppca_cm33_1_signed.hex"
                        }
                    ],
                    "outputs":
                    [
                        {
                            "file": "./../build/app_combined.hex",
                            "format": "ihex",
                            "overlap": "ignore"
                        }
                    ]
                }
            ]
        }
    ]
}
```

3.4 Add TF-M interface to use TF-M services

To use the secure services provided by TF-M, the interface must be initialized to enable communication between the NSPE and SPE. The initialization function for the NSPE interface, `tfm_ns_interface_init()`, must be called by the Main CM33 NSPE. This can be achieved by including the `tfm_ns_interface.h` and `os_wrapper/`

common.h header files in the Main CM33 NSPE code. The below code snippet shows how to initialize the TF-M interface on the Main CM33 NSPE. Once the Interface is initialized, Main M33 NSPE can use TF-M services.

```
#include "tfm_ns_interface.h"
#include "os_wrapper/common.h"


int main(void)
{
    cy_rslt_t result;
    uint32_t rslt;

    /* Device initialization */
    result = cybsp_init();
    ...
    ...

    /* Initialize TF-M interface */
    rslt = tfm_ns_interface_init();
    if(rslt != OS_WRAPPER_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* User application */
    ...
    ...
}
```

***Note**:* *The PSOC™ Control C3M/P8 MCU: Secure-Non-Secure template code example [9] logs through an NSC (Non-Secure Callable) logging function. After TF-M is enabled, the NSC region is owned by TF-M veneers, so the previous NSC logger is no longer available. To keep logging working, use one of the following:*

- *Use TF-M's logging service*
- *Use a Non-Secure UART (SCB) for logging*

See SPE logging for more information on logging.

4 TF-M architecture

Figure 23 provide a high-level view of the TF-M. It shows where each component resides, runs, and how it interfaces with other components.

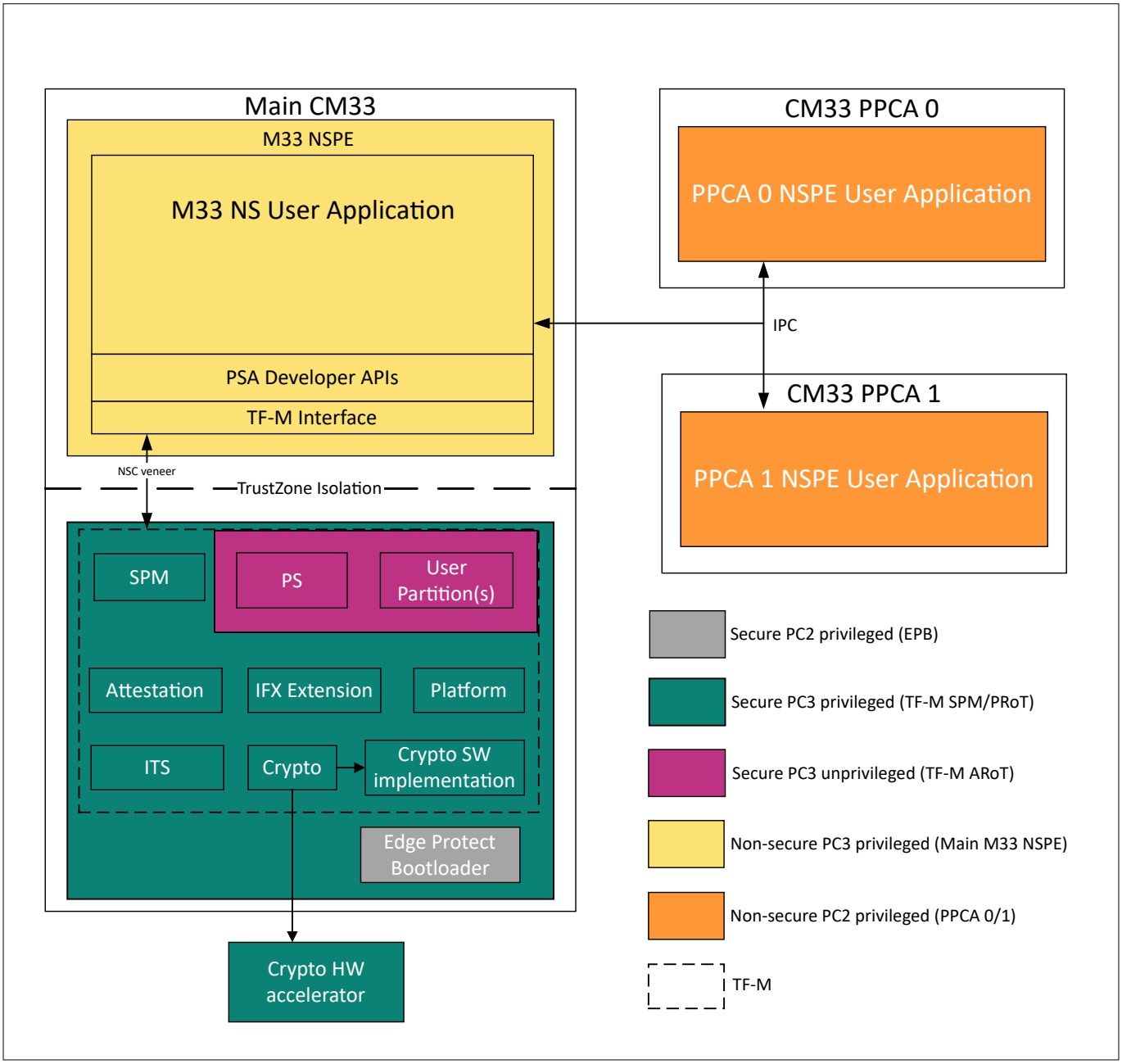


Figure 23 PSOC™ Control C3M/P8, C3M/P7 TF-M architecture

PSC3M/P8, PSC3M/P7 is a tricore MCU. The main CM33 implements the TrustZone® extension, while the two PPCA CM33 do not have TrustZone® implemented and therefore always runs in non-secure (NS) mode. TF-M runs in the secure mode/secure processing environment (SPE) on the main CM33 CPU. The SPE is further partitioned into two domains: PSA Root of Trust (RoT) and the application RoT.

The PSA RoT consists of the following components:

- Edge Protect Bootloader (not part of TF-M)
- TF-M Secure Partition Manager (SPM)
- TF-M PSA RoT partitions
  - Crypto

- Platform
- Attestation
- ITS
- IFX Extension

The application RoT domain houses the following:

- Protected storage partition
- User-defined partition (optional): This is where user-defined services can be implemented in a new partition as part of the application RoT

These partitions can be enabled/disabled from Edge Protect Configurator as per-application needs. The Edge Protect Configurator also provides an option to create new user-defined application RoT partitions. See the Edge Protect Configurator user guide [3] for more information.

The Secure Partition Manager (SPM) and secure services are implemented in separate partitions. The partitions execute in privilege levels as described in Table 1. The SPM is a core component of TF-M that manages the isolation boundaries between these partitions. SPM sets up the execution environment for other secure partitions and NSPEs then transfers control to M33 NSPE. When NSPE requests for secure service, SPM manages the state transitions and schedules the execution of pending requests from NSPE.

Infineon also provides Secure Request Framework (SRF) to facilitate secure access to certain system resources from the NSPE with security aware Peripheral Driver Library (PDL). In the Infineon port of TF-M, the SRF is seamlessly integrated into a PSA RoT partition, IFX Extension Partition. Out of the box, SRF provides support for serial memory interface (SMIF), real-time clock (RTC), system power management (SysPM), and system clock (SysClk) modules. The support for these security aware PDL APIs are offered as a service by IFX Extension Partition of TF-M. A comprehensive list of supported APIs can be found within the respective PDL and SRF documentation.

4.1 Intercommunication between SPE and NSPE

PSC3M/P8, PSC3M/P7 devices have NSPE on the main CM33. The communication between the main M33 NSPE and the main CM33 SPE (TF-M) is facilitated through non-secure callable (NSC) functions. Non-secure callable functions contain the special secure gateway (SG) instruction placed in the NSC memory. When the NSC function call is performed from the NS software, it executes the SG instruction that performs the state change from NS to S and the control is transferred to TF-M, where the secure partition manager (SPM) handles the scheduling of requested services. The PPCA CM33 0/1 do not interact with TF-M.

4.2 Isolation in TF-M

The PSA Firmware Framework-M (PSA FF-M) specification defines three isolation levels to ensure data confidentiality. These three levels of isolation address the varying security needs of devices with varying capabilities.

- **Level 1 isolation:** The isolation boundary exists between the SPE and NSPE
- **Level 2 isolation:** The isolation boundary exists between the SPE and NSPE as well as between the PSA RoT and App RoT
- **Level 3 isolation:** The isolation boundary exists between the SPE and NSPE, between the PSA RoT and App RoT, and between each App RoT partition

The PSC3M/P8, PSC3M/P7 devices are designed to meet PSA Certified Level 3 security requirements with L3 isolation being the default isolation level. The default isolation level can be changed from Edge Protect Configurator. To establish these isolation boundaries between domains effectively, SPM uses protection mechanisms such as TrustZone®, memory protection unit (MPU), memory protection controller (MPC), and peripheral protection controller (PPC).

The TrustZone® is a core component of the CM33 CPU that partitions the address space of the entire device using the Implementation Defined Attribution Unit (IDAU) and Secure Attribution Unit (SAU) in to secure and non-secure regions.

In addition to TrustZone®, PSOC™ Edge implements Infineon's proprietary protection contexts (PC) for additional security. Protection context isolates shared resources in a system from different software components as well as from multiple masters. It adds an additional security attribute to the bus masters over the existing NS attribute of TrustZone®. PSOC™ Control supports eight PCs with each software layer executed using a predefined PC. The PC and NS attribute is used by MPC and PPC to ensure isolation of memory and peripherals between different software components.

For granular access control of memories, TF-M utilizes MPC and MPU for the following:

- The MPC divides memory into fixed-size blocks and provides access control permissions for Read/Write/Non-secure access for each protection context to ensure data confidentiality among partitions
- The MPU controls code execution allowance and access to memory by unprivileged code

Table 1 summarize the execution privilege level of each firmware component.

Table 1 Privilege level of software components

Software component	NS attribute	PC	Priv attribute
Edge Protect Bootloader	S	PC2	Priv
TF-M SPM	S	PC3	Priv
PSA RoT	S	PC3	Priv
App RoT	S	PC3	Unpriv
Main CM33 NSPE	NS	PC3	Priv
PPCA CM33 0/1 NS application	NS	PC2	Priv

TF-M isolates peripherals using the PPC. The PPC provides access restrictions of peripherals among software components based on attributes such as non-secure, privileged, and protection context.

The protection attributes for MPC (read/write/non-secure/PC) and PPC (non-secure/privileged/PC) are grouped together and abstracted in a protection domain. For example, the M33S (secure) protection domain defines secure read and write access to memory and secure privileged access to peripherals (PPC regions) for bus masters with PC2, PC3 and PC7. The Device Configurator allows users to create up to 16 protection domains. Each memory region and PPC region in the device is assigned a protection domain through the Device Configurator. During device initialization, the MPC and PPC are configured such that the memory regions and peripherals have appropriate access permissions as per their assigned protection domain.

The Edge Protect Configurator provides the necessary default protection domains required for TF-M. The Edge Protect Configurator works in conjunction with the Device Configurator to assign these domains to memory regions associated with TF-M as well as to peripherals. The security permission for each protection domain can be viewed from the Systems tab in Device Configurator. See Edge Protect Configurator user guide [3] and ModusToolbox™ Device Configurator user guide for more information.

4.3 Profiles in TF-M

The features of TF-M are highly customizable. To cater to devices with different capabilities and their use cases, TF-M provides a set of profiles. A profile is a group of configuration options used for building TF-M. TF-M provides four profiles; small, medium, and ARoT less and large. Table 2 provides a brief summary of the features offered in each TF-M profile. Out of the four profiles, Infineon's port of TF-M for PSC3M/P8, PSC3M/P7 devices support medium profile. The TF-M profile can be changed from Edge Protect Configurator.

Table 2 TF-M profiles

	Small (not supported)	Medium	ARoT less (not supported)	Large (not supported)
Isolation level	Level 1	Level 2	Level 1	Level 3
Crypto capabilities	• Symmetric cipher suite • SHA 256 hashing	• Symmetric and asymmetric cipher suite • SHA 256 hashing	• Symmetric and asymmetric cipher suite • SHA 256 hashing	• Larger symmetric and asymmetric cipher suite • SHA 256/384/512 hashing
Services/partitions	• ITS • Initial Attestation (symmetric) • Crypto	• ITS • Initial Attestation (asymmetric) • PS • Crypto	• ITS • Initial Attestation (asymmetric) • FWU • Crypto	• ITS • Initial Attestation (asymmetric) • PS • Crypto • FWU (not supported)

The configuration options defined by a TF-M profile can be customized further as per the application's requirements. The following table compares the default configuration used by Infineon's port of TF-M for PSC3M/P8, PSC3M/P7 devices:

Table 3 PSC3M/P8, PSC3M/P7 default TF-M configurations

Configuration	Default value
Profile	Medium
Isolation level	Level 3
Services/Partitions	• ITS • Initial attestation • PS • Crypto • Platform • IFX Extension
NV counters location	Flash

5 Services offered by TF-M

The following services are offered by TF-M to the SPE and NSPE:

5.1 Cryptography

TF-M provides a cryptography service as an implementation of the PSA crypto API in a PSA RoT secure partition within TF-M. This implementation utilizes the `MbedTLS` library. The crypto service can be used by applications running in the NSPE or by other services of TF-M (running in the SPE) through PSA crypto APIs.

TF-M can be configured and built for four different profiles, each providing a different level of functionality. These profiles offer the following cryptography capabilities:

- **Small:** Provides only symmetric cipher, SHA256, and HMAC (not supported)
- **Medium:** Provides symmetric and asymmetric cryptography (as recommended in (RFC 7925) and CoAP (RFC 7252)), along with SHA, HMAC, and asymmetric key algorithms for signing or verification, and public-key cryptography-based key exchange
- **ARoT less:** Same as medium profile (not supported)
- **Large:** Provides symmetric and asymmetric cryptography as defined in TLS 1.2, SHA, HMAC, asymmetric key algorithms for signing or verification, public-key cryptography-based key exchange, and key derivation (not supported)

TF-M for PSC3M/P8, PSC3M/P7 devices is built with the medium profile. The crypto implementation for the PSC3M/P8, PSC3M/P7 devices uses the following:

- Infineon's Crypto Suite A6 Library
- MxCryptoLite hardware accelerator
- Software implementation from the MbedTLS library

The selection to use the Infineon's Crypto Suite A6 Library, MxCryptoLite hardware accelerator, or software implementation is defined at compile-time configuration. Infineon's Crypto Suite A6 is a hardened Crypto Library that uses software implementation, MxAES and MxCryptoLite cryptographic accelerators to provide crypto services for AES and CMAC algorithms. By default, Crypto Suite support is enabled in TF-M. Crypto Suite library support can be disabled or enabled by passing the following Make configuration in `proj_cm33_s/Makefile`

```
TFM_CONFIGURE_OPTIONS+= -DIFX_CRYPTOSUITE_ENABLED:BOOL=OFF/ON
```

By default, MxCryptoLite hardware accelerator support is enabled in TF-M. For crypto algorithms other than AES and CMAC, the selection to use the MxCryptoLite hardware accelerator or software implementation is defined by passing the following Make configuration in `proj_cm33_s/Makefile`.

```
TFM_CONFIGURE_OPTIONS+= -DIFX_MBEDTLS_ACCELERATION_ENABLED:BOOL=ON/OFF
```

It is recommended to use the cryptography service with Crypto Suite Library enabled. If built without Crypto Suite Library, the resulting system cannot meet PSA L3 requirements.

The cryptography service uses ITS to store the persistent keys while the volatile keys are stored in an SRAM region, which is allocated to Crypto partition.

In addition to the predefined profiles, TF-M offers the flexibility to customize the crypto capabilities by adding or removing specific algorithms that align with the application's requirements. This can be achieved by defining a customized crypto configuration header file and setting the

`TFM_MBEDCRYPTO_PLATFORM_EXTRA_CONFIG_PATH` CMake variable with the path of the header file. The following snippet shows how the CMake variable can be set via `proj_cm33_s/Makefile`.

```
TFM_CONFIGURE_OPTIONS+= -DTFM_MBEDCRYPTO_PLATFORM_EXTRA_CONFIG_PATH="Path to config file"
```

5.2 Secured storage

TF-M provides the following two services for secure storage of assets:

5.2.1 Internal trusted storage (ITS)

ITS is a PSA RoT service designed for storing the most security-critical device data, such as cryptographic keys, in the internal storage. In PSC3M/P8, PSC3M/P7 devices, an internal flash NVM is used to store the ITS assets. By default, ITS supports the storage of 10 assets, with a maximum asset size of 512 bytes.

5.2.2 Protected storage (PS)

PS is an application RoT service that offers encrypted storage of assets in the internal flash. PS uses AES-CCM based AEAD encryption to protect the assets. By default, PS supports the storage of 10 assets, with a maximum asset size of 2 KB.

5.3 Initial attestation service

The initial attestation service serves as a key mechanism to prove the genuineness of the device. Upon request, TF-M uses symmetric crypto to create an initial attestation token (IAT) that is compliant with the IETF entity attestation token (EAT) specification. The attestation token contains a list of claims that serve as evidence to prove the identity of the device. The following list of claims are provided in the attestation token:

- boot seed
- challenge
- client_id
- implementation_id
- instance_id
- profile_id
- security_lifecycle
- sw_components (for each image)
  - measurement_description
  - measurement_value
  - signer_id
  - sw_component_type
  - sw_component_version

The attestation token includes software component measurements for the following images:

- Edge Protect Bootloader
- Main CM33 SPE (TF-M)
- Main CM33 NSPE
- PPCA CM33 0
- PPCA CM33 1

This list of claims is encoded in concise binary object representation (CBOR) format and signed by initial attestation private key (256-bit ECC key) as per the CBOR object signing and encryption (COSE) specification (RFC 8152). A certification verification service verifies the token with the initial attestation public key.

The initial attestation service relies on the Edge Protect Bootloader for boot measurements of all images to generate the attestation token. To get boot measurements, the boot measurements feature must be enabled. This can be enabled in the Edge Protect Bootloader Solution by defining the shared_data region in SRAM within the "bootloader" section of EPB's memory map JSON file. The boot measurements are placed in the above defined SRAM region. See `overwrite_multi4_flash.json` provided with PSOC™ Control C3M8 MCU: Basic TF-M application.

5.4 Platform service

The platform service leverages the platform-specific features of the MCU and offers them to the NSPE via the platform partition of TF-M. TF-M for PSC3M/P8, PSC3M/P7 devices offer the following platform services:

Non-volatile counters: TF-M offers up to three non-volatile counters of four bytes each to the NSPE. The NSPE can use the platform service to either read or increment the NV counters. Include the `tfm_platform_api.h` header file in the NSPE project and use the `tfm_platform_nv_counter_increment()` and `tfm_platform_nv_counter_read()` APIs to use the service.

System reset: The registers required to reset the system are protected and cannot be accessed by the NSPE. The system reset service provides a way for the NSPE application to reset the platform. To reset the device, include the `tfm_platform_api.h` header file in the NSPE project and call the `tfm_platform_system_reset()` API function.

NSPE logging: The logging platform service can be used by the NSPE to log messages on the serial terminal using UART. This service can be quite useful when only one SCB is available and both the TF-M and NSPE applications need to log. Before using the NSPE logging service, secure SPE logging in TF-M must be enabled. To enable logging, add the following Make configuration in the secure (`main_cm33_s`) project Makefile.

```
TFM_CONFIGURE_EXT_OPTIONS+= -DTFM_EXCEPTION_INFO_DUMP=ON -DPLATFORM_EXCEPTION_INFO=ON
-DIFX_FAULTS_INFO_DUMP=ON -DTFM_SPM_LOG_LEVEL=LOG_LEVEL_VERBOSE
-DTFM_PARTITION_LOG_LEVEL=LOG_LEVEL_VERBOSE
```

Enable the desired SCB instance with the `IFX_TFM_SPM_UART` name in Device Configurator. See SPE logging to set up the SCB for logging. To use this service, the NSPE application must include the `ifx_platform_api.h` header and call the `ifx_platform_log_msg()` API.

5.5 IFX Extension Partition

The IFX Extensions Secure Partition is an Infineon-defined TF-M partition that provides a general-purpose secure extension point. It fulfills two roles:

- **Secure Request Framework (SRF) gateway**: Routes SRF requests from non-secure code through TF-M's security boundary into the SRF execution context.
- **User extension point**: Hosts project-specific secure functionality that does not fit within existing TF-M standard partitions.

The partition is registered as a PSA Root of Trust (PSA-RoT) component and is callable from both the secure and non-secure domains. Through SRF, it exposes security-aware driver implementations for modules, such as SMIF, RTC, SysClk, and SysPM. Refer to the SRF and PDL documentation for the complete set of supported APIs.

The partition supports registration of custom SRF modules in addition to the built-in ones. To add a user module:

1. Set `TFM_CONFIGURE_EXT_OPTIONS+=-DIFX_EXT_SP_REGISTER_USER_SRF_MODULE=ON` in the `main_cm33_s/Makefile`. With this option enabled, the partition invokes user module registration during its initialization
2. Create `ifx_user_srf_module.c` in `main_cm33_s/imports/ifx-tf-m-port/ifx_ext_sp`. Define a global symbol of type `mtb_srf_module_s_t` named `ifx_user_srf_module`. Implement the secure-side functions for each operation required by your module. Alternatively, integrate your module directly by modifying the partition sources
3. At TF-M startup, TF-M calls `mtb_srf_module_register()` to register all modules. If additional initialization is needed, add it to `ifx_ext_sp_init()` in `main_cm33_s/imports/ifx-tf-m-port/ifx_ext_sp/ifx_ext_sp_mngr.c` before the function returns
4. In the NSPE, provide APIs that wrap `ifx_mtb_srf_call()`. These wrappers invoke the IFX Extension Secure Partition service and dispatch the request to the appropriate secure-side user SRF implementation

6 TF-M memory map

This section describes the memory configuration and usage associated with TF-M for the PSC3M/P8, PSC3M/P7 devices.

Figure 24 illustrates the default memory layout for basic Trusted Firmware-M (TF-M) based application, showing the division of various memories for the secure and non-secure processing environments for code, data, and shared regions.

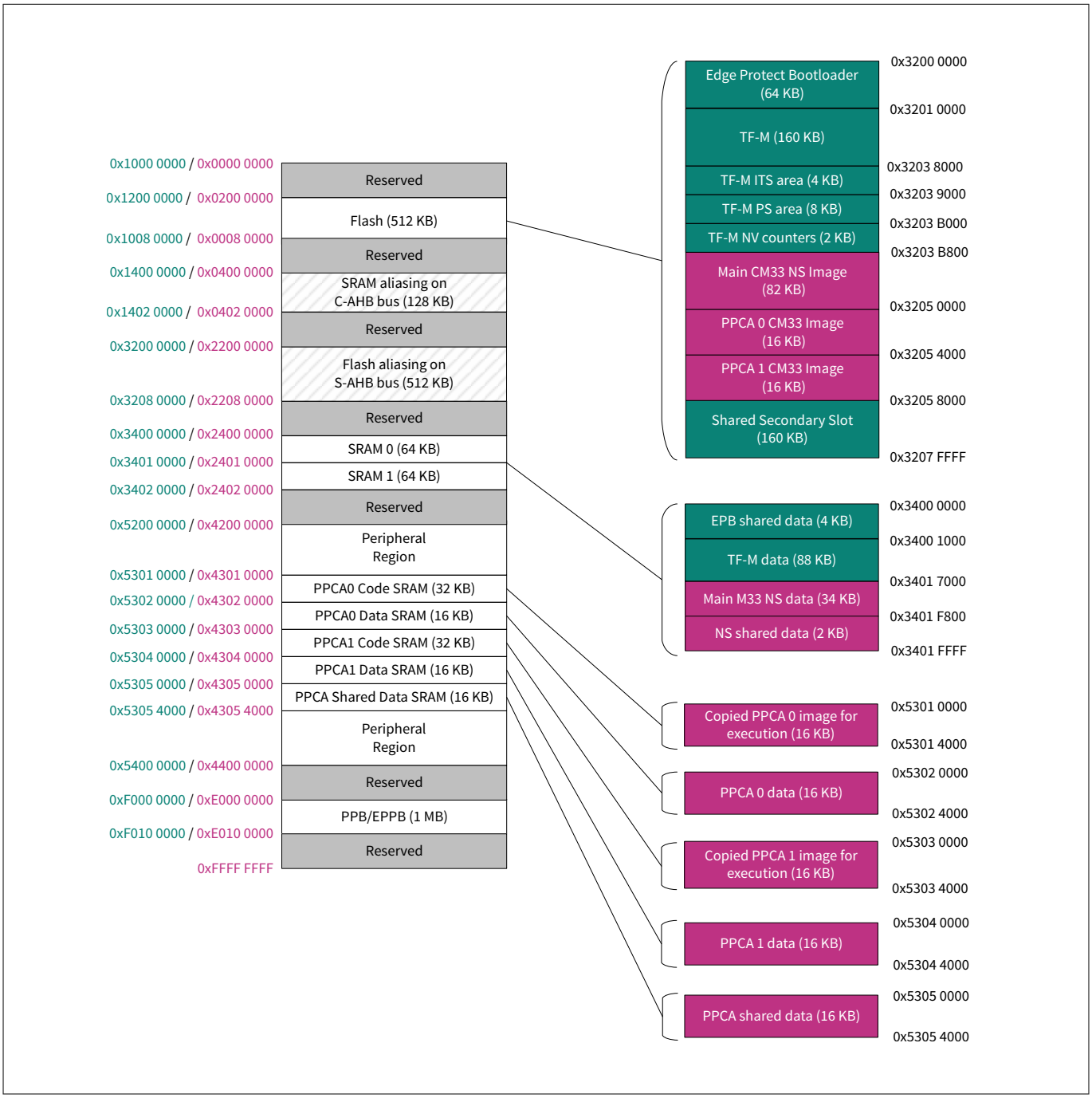


Figure 24 TF-M memory map

- Internal flash
  - PSC3M/P8, PSC3M/P7 devices include 512 KB of internal flash

  - The first 283 KB is configured as secure: 64 KB for the EPB, 160 KB for the TF-M image, 4 KB for ITS storage, 8 KB for PS storage, and 2 KB for NV counters
  - The remaining flash is partitioned as follows:
    - Non-secure regions for the Main CM33 NSPE image (82 KB) and the PPCA CM33 0/1 images (16 KB each)
    - A secure shared secondary slot for image upgrades (160 KB)
- SRAM
  - The first 4 KB of SRAM0 is secure and reserved for measured boot data produced by the EPB; this shared data is essential for the Initial Attestation service
  - The next 88 KB (60 KB in SRAM0 and 28 KB in SRAM1) is secure and dedicated to TF-M data
  - The subsequent 34 KB of SRAM1 is non-secure for the Main CM33 NS image, and the final 2 KB is used for data sharing
- PPCA SRAMs
  - PSC3M/P8, PSC3M/P7 devices provide five PPCA SRAM blocks. All are non-secure and available to PPCA non-secure applications

6.1 Changing the TF-M memory map

The TF-M application memory map can be customized to satisfy the requirements of the use case. The default TF-M map can be changed by updating the memory sections in the memory tab of the Device Configurator. The Device Configurator allows changing the size and location of memory regions as well as creating new regions and assigning protection domains to them. Refer to the memory tab section in the Device Configurator user guide for detailed instructions on modifying the memory map.

When the TF-M application's memory map changes, update the Edge Protect bootloader's memory map JSON and the TF-M application's signer combiner JSON so the bootloader's memory map stays in line with the application, and signed images are generated for the correct addresses.

7 Secure boot and image upgrade

By default, image authentication is disabled in BootROM firmware. Enabling secured boot enables the Chain of Trust (CoT) where each software is authenticated by the preceding software. In PSC3M/P8, PSC3M/P7 devices, the Root of Trust (RoT) is anchored in the Immutable BootROM firmware, which is executed by Main CM33. To establish the full chain of trust:

- Enable secured boot in BootROM
- Enable secured boot in the Edge Protect Bootloader and sign the bootloader
- Enable image signing in PSC3M/P8, PSC3M/P7 TF-M application

To enable secure boot, first transfer device ownership to the OEM and then update the OEM policy to turn on secure boot. For detailed procedures, see the “Enable secure boot” section of AN242836 – Getting started with PSOC™ Control C3M/P8, C3M/P7 MCUs security [2].

To enable signing in the Edge Protect Bootloader, follow the instructions in the “Secure boot” subsection under Edge Protect Bootloader in AN242836 – Getting started with PSOC™ Control C3M/P8, C3M/P7 MCUs security [2].

Edge Protect Bootloader image authentication and TF-M application image signing are already enabled in the basic TF-M code example. No additional steps are required. If you are adding TF-M to an existing application and need to enable Edge Protect Bootloader image authentication and application signing, refer to the “Secure boot” subsection under Edge Protect Bootloader and the “Code signing and verification” section in AN242836 – Getting started with PSOC™ Control C3M/P8, C3M/P7 MCUs security [2].

The Edge Protect Bootloader manages image upgrade and supports two upgrade modes:

- **overwrite:** The Edge Protect Bootloader copies the image from the secondary slot and overwrites the primary slot. This is the default upgrade mode in the PSOC™ Control basic TF-M code example
- **swap:** The Edge Protect Bootloader uses a scratch area to swap the images from primary and secondary slot. The upgrade mode requires two extra memory regions, a scratch region for performing the image swapping operation and a swap status region to track the progress and state of the swap operation

For guidance on selecting the upgrade mode, refer to “Edge Protect Bootloader features” in AN242836 – Getting started with PSOC™ Control C3M/P8, C3M/P7 MCUs security [2].

8 Device setup for PSA certification

This procedure prepares the KIT_PSC3M8_EVK for PSA L3 certification testing. Ensure that all the required assets listed in Table 4 are available before starting. Where referenced, follow the respective application notes for detailed step-by-step instructions.

Table 4 Asset list

Asset Name	Version
ROM	SROM Boot: 1.0.0.13087
Flashboot	FlashBoot: 1.0.0.13982
Edge Protect Bootloader	Edge Protect Bootloader code example [8]: release v1.0.0.3695 mcuboot: release-v1.9.600.16033 ifx-wolfcrypto: v5.9.2.7056036 ifx-mbedtls: v3.6.606.1509 cy-mbedtls-acceleration: v3.1.0.1926 crypto-suite-psc3: v6.1.2.24
TF-M	ifx-tf-m: v2.3.400.19504 ifx-tf-m-port: v2.3.400.19504 ifx-tf-m-ns: v2.3.400.19504 mtb-dsl-psc3m8: v1.2.0.558 mtb-srf: v1.2.1.1597 crypto-suite-psc3: v6.1.2.24 cy-mbedtls-acceleration: v3.1.0.1926 TF-PSA-Crypto: v1.1.0 cmsis: v6.2.0
Edge Protect Security Suite	Edge Protect Security Suite: v2.3.0.958 Edge Protect Tools: v2.3.0.16368
ModusToolbox™ Programmer	ModusToolbox™ Programmer: v1.9.0.2261 OpenOCD: v5.19.0.4782

Create a TF-M and EPB applications:

- Create a basic TF-M application targeting the KIT_PSC3M8_EVK BSP, as described in Create basic TF-M based application section
- The example uses the following default TF-M configurations:
  - Isolation level: 3
  - FIH: High
  - Profile: Medium
  - All partitions are enabled: Crypto, Internal Trusted Storage, Initial attestation, Protected Storage, Platform
- Create the Edge Protect Bootloader application and configure the bootloader memory map by following Add Edge Protect Bootloader to TF-M application section
- Ensure that both the TF-M and the Edge Protect Bootloader code examples use the above specified asset version. If they do not, update them through the Library Manager

Transfer of ownership

- By default, device ownership is held by Infineon
- Transfer ownership as described in the "Getting started" section of AN242836 – Getting started with PSOC™ Control C3M/P8, C3M/P7 MCUs security [2]

Configure Edge Protect Bootloader:

- Open `PSOC_Edge_Protect_Bootloader/platforms/PSC3_P8/feature_config.json` and set the following options:
  - Set "validation_key_type" to one of the following: ECDSA-256/ECDSA-384/ECDSA-521/ LMS_SHA256_M32_H10/XMSS_SHA2_10_256/ML-DSA-87
  - Set "validation_key" to production OEM ROT public key 0 (generated during device ownership transfer) path `"<key_path>/keys/oem_public_key_0.pem"`
  - Set "validate_boot" to true
  - Set "validate_upgrade" to true
  - Set "hw_rollback_prot" to true
  - Set "fault_injection_hardening" to high

Enable secure boot by BootROM

- Enable secure boot by following the instructions from “Enable secure boot” section of AN242836 – Getting started with PSOC™ Control C3M/P8, C3M/P7 MCUs security [2]
- Sign the Edge Protect Bootloader by following the instructions from “Secure boot” subsection under Edge Protect Bootloader in AN242836 – Getting started with PSOC™ Control C3M/P8, C3M/P7 MCUs security [2]

Build and program application:

- In Eclipse IDE, select the Edge Protect Bootloader Project Explorer and click "Program Application" from the Quick panel to build and program the Bootloader. Similarly, program the TF-M application

Transition to Production LCS

- Prepare the assets to transition device to Production LCS as described in "Transition to Production LCS" section of AN242836 – Getting started with PSOC™ Control C3M/P8, C3M/P7 MCUs security [2]
- Debug ports are enabled by default. To prevent unauthorized access, enable authenticated debug:
  - Follow the "Debug token" section of AN242836 – Getting started with PSOC™ Control C3M/P8, C3M/P7 MCUs security [2] to enable authenticated debug
  - Set CM33 AP to "Disable". This configures the CM33 AP as closed but can be opened with a valid debug token/certificate
  - Keep SYS_AP set to Enabled (default). This allows:
    - Loading RAM applications (re-provisioning, key revocation, etc.)
    - Loading the debug certificate required to open CM33_AP
  - As mentioned in "Debug token" section:
    - Generate the debug key pair
    - Set the debug public key path in OEM policy and transitioning to Production LCS
    - Generate the debug certificate (required for programming and debugging)

***Note**: To perform an image update, the upgrade images must be built with an updated image version and hardware security counter value. Use the signer combiner JSON file to set these values.*

References

Application notes

[1] Infineon Technologies AG: *AN241270 – Getting started with PSOC™ Control C3M/P8, C3M/P7 MCUs on ModusToolbox™ software*; Available online

[2] Infineon Technologies AG: *AN242836 – Getting started with PSOC™ Control C3M/P8, C3M/P7 MCUs security*; Available online

User guide

[3] Infineon Technologies AG: *002-39923: ModusToolbox™ Edge Protect Configurator user guide*; Available online

Datasheet

[4] Infineon Technologies AG: *002-39793: PSOC™ Control C3 - PSC3P7xx, PSC3M7xx, PSC3P8xx, PSC3M8xx*; Available online

Reference manuals

[5] Infineon Technologies AG: *002-42141: PSOC™ Control C3 (PSC3P7xx, PSC3M7xx, PSC3P8xx, PSC3M8xx) architecture reference manual*; Available online

[6] Infineon Technologies AG: *PSOC™ Control C3 (PSC3P7xx, PSC3M7xx, PSC3P8xx, PSC3M8xx) registers reference manual*

Code examples

[7] Infineon Technologies AG: *PSOC™ Control C3M8 MCU: Basic TF-M application*; Available online

[8] Infineon Technologies AG: *PSOC™ Control C3M8 MCU: Edge Protect Bootloader*; Available online

[9] Infineon Technologies AG: *PSOC™ Control C3M8 MCU: Secure-Non-Secure template*; Available online

***Note**: For more documents related to PSC3M/P8, PSC3M/P7 devices, see PSOC™ Control C3 Performance Line webpage.*

Glossary

Arm® TrustZone®
An Arm® technology used for isolating SPE and NSPE address space.

BootROM
Immutable boot firmware responsible for the secure boot process. It interprets the OEM policy and boots the first OEM code based on the policy configuration. The Root of Trust (RoT) is anchored in BootROM.

Edge Protect Bootloader (MCUboot)
An open-source secure bootloader library for 32-bit MCUs aiming to define a common infrastructure for the bootloader and system flash layout on microcontroller systems, and to provide a secure bootloader that enables easy software upgrade.

ModusToolbox™
It is an Eclipse-based embedded design platform for IoT designers that offers a single, coherent, and familiar design experience. The platform combines the industry's most deployed Wi-Fi and Bluetooth® technologies with the lowest power, most flexible MCUs, and best-in-class sensing capabilities.

MPC
Memory protection controller. This hardware block is used for access control to memory based on protection context (PC) and secure/non-secure (S/NS) attributes.

MPU
Memory Protection Unit. These hardware blocks are used to permit or restrict access to sections of memory or peripherals for CPU bus masters.

NSC
Non-secure callable (NSC) is a special type of secure region which is permitted to hold secure gateway (SG) instructions that allows state transition from non-secure to secure state.

NSPE
Non-secure processing environment. It is a space that is not isolated from other software running on the device. Data from the NSPE can be accessed by other software. Typically, the NSPE has limited access to memory and peripherals.

PC
Protection context. An Infineon proprietary access control mechanism that adds an additional layer of security over TrustZone®.

PPC
Peripheral protection controller. These hardware blocks are used for isolating peripheral regions based on protection context (PC), S/NS attributes, and privileged/unprivileged attributes.

PSA
Arm® Platform Security Architecture. Arm® PSA is a comprehensive security framework developed to enhance the security of IoT devices by providing standardized practices, reference implementations, and certification processes.

PSA Certified
PSA Certified program offers a low-cost, time-saving way for implementing security. PSA Certified framework ensures security is designed into a device right from hardware to software.

PSOC™

It is a programmable, embedded design platform that includes a CPU, such as the 32-bit Arm® Cortex®-M0, with both analog and digital programmable blocks. This platform accelerates embedded system design with reliable, easy-to-use solutions such as touch sensing, and enables low-power designs.

SPE

Secure processing environment. It is an isolated and protected space where data can be processed and stored securely. Data from the SPE is not accessible to the outside world but SPE has full access to the system's resources.

TF-M

Trusted Firmware-M. It is an open-source initiative by Arm® that implements the secure processing environment (SPE) for Armv8-M, Armv8.1-M architectures, and dual-core platforms.

TSBA-M

Arm® Trusted Base System Architecture. It is an architecture for designing and implementing secure devices which is part of Arm® PSA

Revision history

Document revision	Date	Description of changes
**	2026-07-30	Initial release
*A	2026-09-21	Web release

Trademarks

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc., and any use of such marks by Infineon is under license.

PSOC™, formerly known as PSoC™, is a trademark of Infineon Technologies. Any references to PSoC™ in this document or others shall be deemed to refer to PSOC™.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2026-09-21

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2026 Infineon Technologies AG

All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference

IFX-oqs1776828522200

Important notice

Products which may also include samples and may be comprised of hardware or software or both ("Product(s)") are sold or provided and delivered by Infineon Technologies AG and its affiliates ("Infineon") subject to the terms and conditions of the frame supply contract or other written agreement(s) executed by a customer and Infineon or, in the absence of the foregoing, the applicable Sales Conditions of Infineon. General terms and conditions of a customer or deviations from applicable Sales Conditions of Infineon shall only be binding for Infineon if and to the extent Infineon has given its express written consent.

For the avoidance of doubt, Infineon disclaims all warranties of non-infringement of third-party rights and implied warranties such as warranties of fitness for a specific use/purpose or merchantability. Infineon shall not be responsible for any information with respect to samples, the application or customer's specific use of any Product or for any examples or typical values given in this document.

If this document is marked as 'Preliminary', 'Target', 'Draft' or 'Proposal', Infineon reserves the right to change all information given in this document at any time without notice. Subject to the development and release of a Product for series supply by Infineon, the technical specifications of the Product are set forth in the relevant datasheet provided by Infineon without such marking.

The data contained in this document is exclusively intended for technically qualified and skilled customer representatives. It is the responsibility of the customer to evaluate the suitability of the Product for the intended application and the customer's specific use and to verify all relevant technical data contained in this document in the intended application and the customer's specific use. The customer is responsible for properly designing, programming, and testing the functionality and safety of the intended application, as well as complying with any legal requirements related to its use.

Unless otherwise explicitly approved by Infineon, Products may not be used in any application where a failure of the Products or any consequences of the use thereof can reasonably be expected to result in personal injury.

However, the foregoing shall not prevent the customer from using any Product in such fields of use that Infineon has explicitly designed and sold it for, provided that the overall responsibility for the application lies with the customer. Infineon expressly reserves the right to use its content for commercial text and data mining (TDM) according to applicable laws, e.g. Section 44b of the German Copyright Act (UrhG). If the Product includes security features:

Because no computing device can be absolutely secure, and despite security measures implemented in the Product, Infineon does not guarantee that the Product will be free from intrusion, data theft or loss, or other breaches ("Security Breaches"), and Infineon shall have no liability arising out of any Security Breaches.

If this document includes or references software:

The software is owned by Infineon under the intellectual property laws and treaties of the United States, Germany, and other countries worldwide. All rights reserved. Therefore, you may use the software only as provided in the software license agreement accompanying the software.

If no software license agreement applies, Infineon hereby grants you a personal, non-exclusive, non-transferable license (without the right to sublicense) under its intellectual property rights in the software (a) for software provided in source code form, to modify and reproduce the software solely for use with Infineon hardware products, only internally within your organization, and (b) to distribute the software in binary code form externally to end users, solely for use on Infineon hardware products. Any other use, reproduction, modification, translation, or compilation of the software is prohibited.

For further information on the Product, technology, delivery terms and conditions, and prices, please contact your nearest Infineon office or visit www.infineon.com.