Security Installer Demo

Table of Contents

Table of Figures

1. Introduction

The following document describes the steps to activate security features on S32K3XX NXP platforms. This application demonstrates how to provision HSE FW on a device first time delivered to customer and can be configured to run multiple examples of HSE services usage to do an initial configuration, run cryptographic services, run SHE Command App services, run monotonic counter related services, update the FW, enable secure boot and program UTEST area for application debug process configuration and life cycle advancement. This application can also be used to read various attributes of firmware like Current Life Cycle, Firmware Version, Debug Auth method etc. and write various attributes as well like Program ADKP key, Advance LC of product, Update Debug Auth Mode etc.

Disclaimer: This application contains demonstrative code that should not be used in the production as it is. S32K344 devices are being used as reference for explanation of the features in the document and illustrative pictures used in the document, readers are advised to relate to their device accordingly and check the device datasheet. IN NO EVENT SHALL NXP OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

2. Glossary and Terms

The table below lists the abbreviations used throughout the document

Table 1. Definitions and Abbreviations used in this document

Abbreviations	Description
ADK/P	Application Debug Key / Password; one-time programmable parameter
AES	Advanced Encryption Standard; a cipher primitive
ASB	Advanced Secure Boot
BSB	Basic Secure Boot
CR Table	Core Reset Table
CUST_DEL	Device Life Cycle customer delivery
HSE	Hardware Security Engine
ID	Identifier
IN_FIELD	Device Life Cycle is IN_FIELD
IVT	Image Vector Table
LC	Device Life Cycle; used to limit by design the configuration and debug / test possibilities of the device for in-field usage
MAC	Message Authentication Code
MCU_PROD	Device Life Cycle MCU Production
NVM	Non-Volatile Memory; typically Flash that can be embedded in the device or externally connected to the device
OEM_PROD	Device Life Cycle OEM Production
OTA	Over The Air
POR	Power On Reset
RAM	Random Access Memory
SBAF	Secure Boot Assist Flash
SMR	Secure Memory Region

Terms	Description
Application Image	Binary image containing the user application code. This is stored in plain format and optionally signed if secure boot is enabled.
HSE FW	HSE firmware code: It provides cryptographic security services for the entire platform. HSE FW image delivered is encrypted and signed (see HSE FW Pink image definitions).
HSE FW Pink image	HSE FW image delivered by NXP. This is encrypted and authenticated with keys known by NXP. Authentication (signature generation) is done with an asymmetric algorithm.
HSE FW Image header tag	The first byte of an image header, also known as marker. 0xDA – Full Mem HSE firmware 0xDB – AB SWAP HSE Firmware 0xDC – Encrypted Secure-BAF Image
IVT image	First 256 bytes of data read by HSE firmware after reset. Contains some attributes configured by secure BAF and references to the other images used by secure BAF/HSE FW at boot time.

3. Package Contents

The demo package has parent directory as *"HSE_DEMOAPP_S32K3XX"* and it contains the following artifacts:

Note: Demo app and secure boot app images can be generated by either GHS or S23DS tool.

- **Additional Documentation** found in the *"\docs\"* folder

  a) *NXP_HSE_FW_FAQ.pdf* – contains Frequently Asked Question for user.

  b) NXP_HSE_FW_Key_Storage_Calculator.xlsx – file to calculate NVM key storage and RAM key storage

- **Binary Files** *found in the "\images\" folder*

  a) *ivt.bin* – IVT image for non-secure\secure boot.

- **Demo app projects files** is found in the *"HSE_DEMOAPP_S32K3XX\demo_security_installer\Projects"* folder:

  a) *GHS\demo_app\S32K3xx\S32K3x4\standard\demo_app_m7_0.gpj* – Top GHS project for the demo app. Select the *demo_app_m7_0*.gpj file directory as per your device.

  b) *GHS\SecureBootApp\secure_boot_app.gpj*– secure boot app GHS project.

  c) *S32DS\demo_app\S32K3xx\S32K3xx_demo_app* – Top S32DS project for demo app.

  d) *S32DS\SecureBootApp* – secure boot app S32DS project

- **Source Code** is found in *"HSE_DEMOAPP_S32K3XX\demo_security_installer\src"* folder:

  a) *demo_app* - demo app, secure boot app and SHE command app source code.

- **scripts found in the** *"\scripts\"* folder:

Note: Some of the script for e.g.: menu.cmm offer same features but the images loaded during demo app execution is generated from either GHS tool or S32DS image respectively.

Table 2. Scripts details

File Name	Description
activate_passive_block.cmm	Script to request HSE to activate passive block
adkp_key_input.txt	Text document for ADKP input, this is updated by user before programming ADKP.
advance_LC.cmm	Script to request HSE to advance life cycle.
attach_core_m7_0.cmm	script to attach to M7-0 core i.e. application core.
attribute_programming.cmm	script contains option of read and write attribute, called from run_all_test.cmm script.
attribute_read.cmm	script offers read attributes request to hse from demo app.
attribute_write.cmm	script offers write attributes request to hse from demo app.
challenge.txt	Text file updated with challenge read from debugger
debug_App_ADKP.py	Python script used for debug authorization
check_fw_install_status.cmm	script checks whether firmware installed or not.
check_secure_boot_app_status.cmm	script checks whether secure boot app was successful or not.
debug_authorization.cmm	script prompts user to input two things: debug auth mode and user rights that is configured for device before starting debug authorization. The script also read ADKP input from adkp_key_input.txt.
EraseKeys.cmm	script for erasing NVM data.

File Name	Description
fw_update.cmm	this script writes new firmware at BLOCK 0 location 0x422000.
hse_fw_installation.cmm	scripts prompt user to select from two types of firmware installation offered by demo app: fw install with ivt and fw install without ivt.
load_app_symbols	script maps demo application symbols.
load_secure_boot_app_symbols	script maps secure boot application symbols.
menu.cmm	script offers all features supported in demo app, this is the first script user should run from t32.exe.
monotonic_counter.cmm	offers monotonic counter feature.
mu_attribute.cmm	script offers MU1 configuration.
program_hse.cmm	script loads only firmware pink image in device in case of fw install without IVT option.
program_hse_ivt_and_demo_app	script loads ivt, demo app, secure boot app and firmware pink image in device in case of firmware install with IVT.
program_hsefwusageflag.cmm	script enables HSE FW feature flag in UTEST location 0x1B000000 when user want to use HSE firmware.
run_all_tests.cmm	script offers different features supported by this demo app.
S32_JTAG_WRITE.cmm	this script is used for debug authorization.
program_app.cmm	script only loads IVTs, demo app and secure boot app image in the device in case when user do not want to load hse fw image again since install is already done.
she_services.cmm	This script is used to offer she command app based services

File Name	Description
secure_boot.cmm	script offers three different types of secure boot: Basic Secure Boot, Advanced Secure Boot and SHE secure boot.
monotonic_counter_configure2.cmm	script prompts user to configure RPB value in watch window.
monotonic_counter_increment2.cmm	script prompts user to increment counter.
monotonic_counter_read_display.cmm	script prompts user to read counter.
monotonic_counter_select.cmm	Script prompts user to select counter number.
S32_JTAG_READ.cmm	Script to read JTAG response
S32_JTAG_READ_UID.cmm	Scrip to read UID from JTAG
S32_JTAG_WRITE.cmm	Script to write password on JTAG
S32_SDAP_READ.cmm	Script to read SDAP
S32_SDAP_READ_UID.cmm	Script to read SDAP UID
S32_SDAP_WRITE.cmm	Script to write SDAP
system_setup.cmm	Script to setup basic steps required for programming images
t32_wrapper_func.py	Python script for wrapper functions required for trace32
uid.txt	Text file for printing UID

4. Setup

This section describes demo app dependencies, available setups and application configuration options for HSE use cases demos.

4.1. HSE Firmware

The demo app needs the HSE FW interface files and encrypted HSE FW-IMG. Download the compatible HSE release from HSE release package and copy the subdirectories *"hse"* and *"interface"* inside directory from the package as shown below.

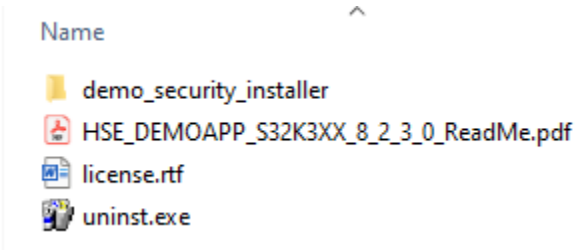


Figure 1. Folder Setup

Important: This is just an illustrated image and actual image may vary.

Note:

- *"hse\bin"* folder should contain pink image.
- This demo app is independent of either FULL MEM or AB SWAP HSE firmware *"interface"* files hence the user can take interface folder from any of the above HSE firmware release type.

4.2. Tools

To generate the application image, programming it along with HSE FW and/or running and debugging it, two independent setups are available:

4.2.1. S32 Design Studio and Trace32

- S32 Design Studio for compilation and demo app binary file generation
- Trace32 Lauterbach debugger for programming the image, running and debugging programed/loaded applications.

Version Used

This application has been developed and validated using:

- S32 Design Studio for S32 Platform, Version: 3.4, Build id: 201217
- For S32 DS, following package is used: S32K3xx_development_package, Version: 3.4.0
- Trace32 PowerView for ARM, Software: Nightly Build(64-bit), Version: N.2020.06.000122933 and Build: 122933

Import and Build project using S32 Design Studio

Below are the steps involved while importing project using S32 DS.

- Click on “File” from the Menu.

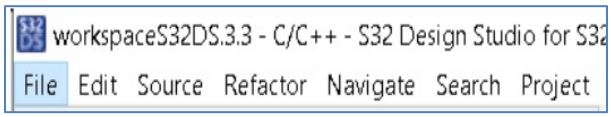


Figure 2. S32 Design Studio Menu

- Click on “Import”.

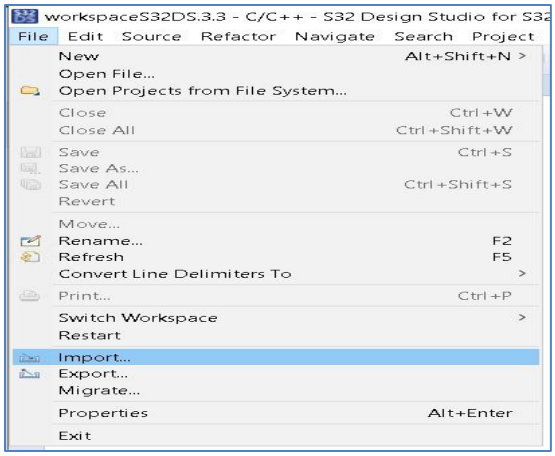


Figure 3. S32DS Project Import

- From “General” select “Existing Projects into Workspace” and then click OK.

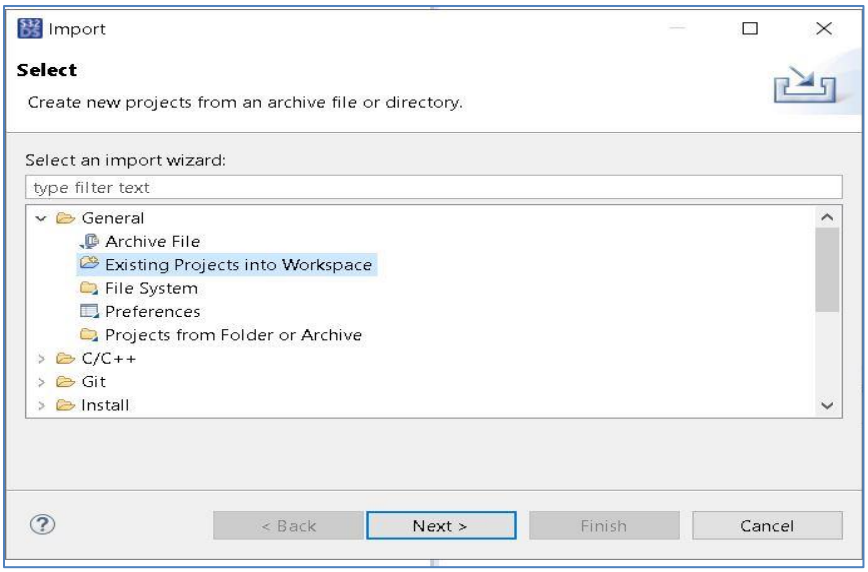


Figure 4. S32DS Import Existing project into Workspace

- "Select root directory" and Browse where demo project is stored and then click OK.

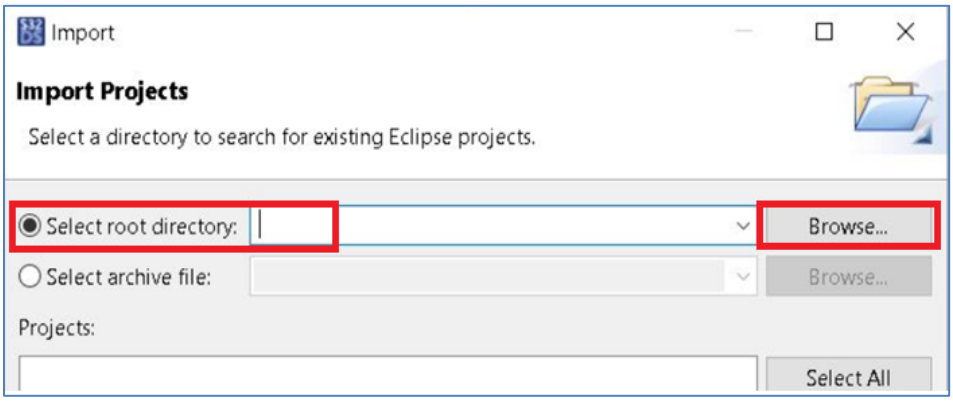


Figure 5. S32DS Browse the project

- Ensure all projects are imported as shown in Figure 7.

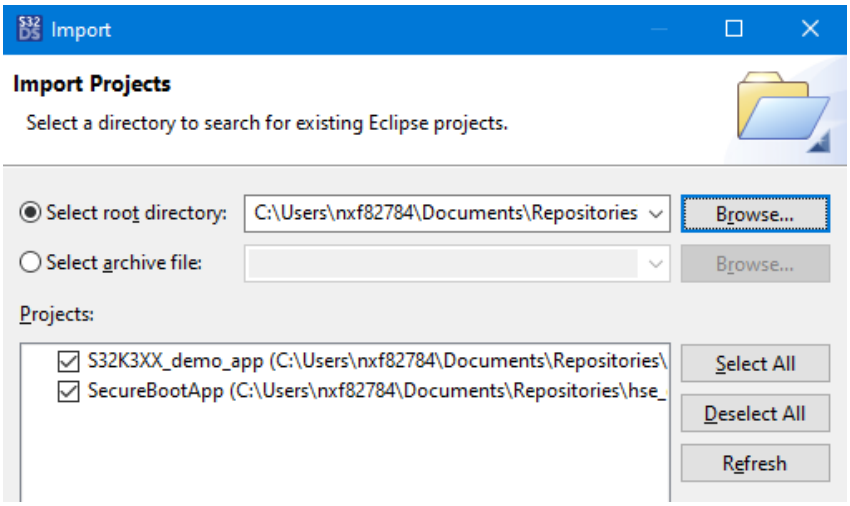


Figure 6. S32DS 2 projects imported.

- Click on "Finish".
- Verify the project also being displayed in the project explorer.

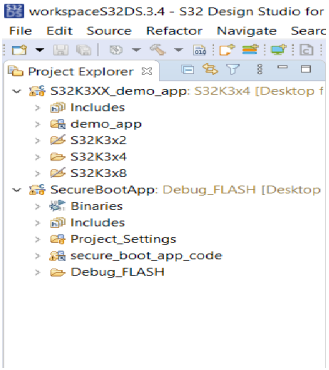


Figure 7. S32DS Project explorer

- Right click the project "*S32K3XX_demo_app*" and then select active build configuration according to your device e.g. 'S32K3x4' for S32K344.

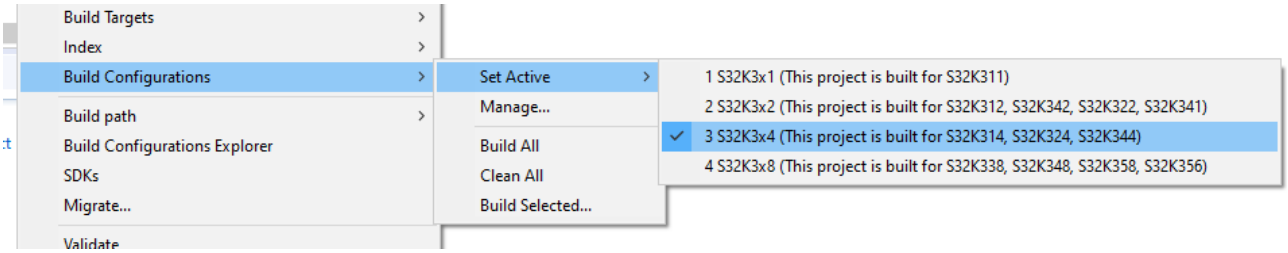


Figure 8. S32DS Build Configurations

Important: This is just an illustrated image and actual image may vary according to the selected device.

- Right click on any selected project and select the options of 'Clean' and 'Build' the project.

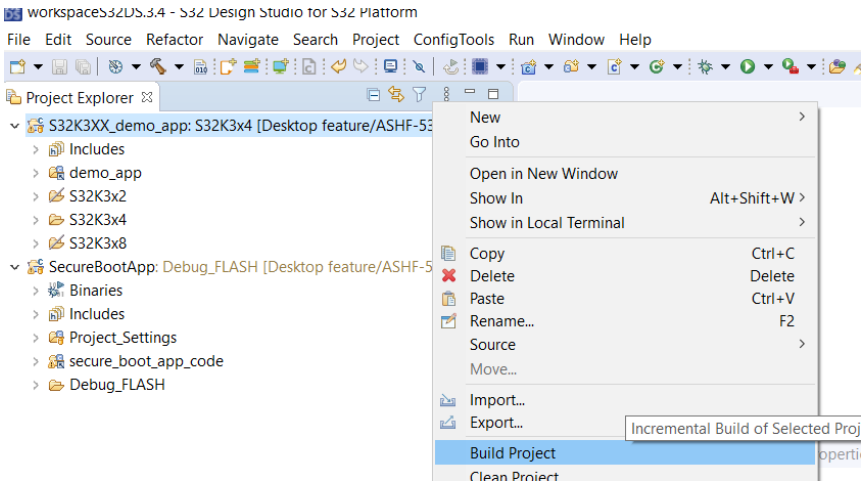


Figure 10. S32DS Build Project

- Verify the project build status in the 'console' view.

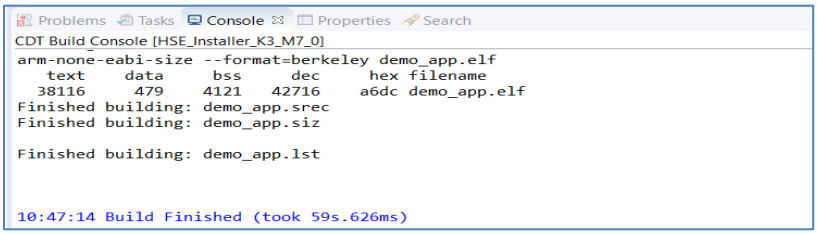


Figure 9. S32DS Build Status 'Console' project HSE_Installer_K3_M7_0

- Similarly, build the *"SecureBootApp"* project by right clicking on project name.

4.2.2. Multi GHS and Trace32

GHS Multi project for compilation and demo app binary file generation.

Trace32 Lauterbach debugger for programming the device, running, and debugging programed application/loaded applications directly in RAM.

Version Used

This application has been developed and validated using:

- GHS Multi v7.1.4, Compiler v2017.1.4
- Trace32 Power View for ARM, Software: Nightly Build(64-bit),Version: N.2022.03.000145572 and Build: 145572Build project using GHS

To build project using GHS, considering use is inside HSE_DEMOAPP_S32K3XX folder, below are the steps:

- Go to the path of *demo_app_m7_0.*gpj file according to your device: e.g. "\Projects\GHS\demo_app\S32K3xx\S32K3x4" for S32K344.
- The build for S32K3x4 caters to all devices of this family like S32K314, S32K324 and S32K344. Similarly, The build for S32K3x2 caters to all devices of this family like S32K312, S32K322and S32K342. Same is applicable for S32K3x8, S32K3x6 and S32K3x1.
- Select project: demo_app_m7_0.gpj

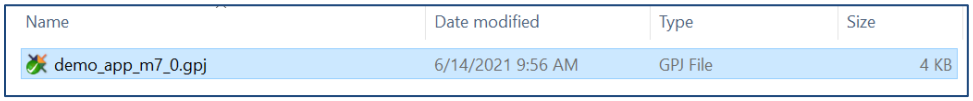


Figure 10. Open Demo App project using GHS

- Open the project using multi-GHS IDE or double click it.
- Click on 'Build' from the top Menu and then click 'Rebuild'.

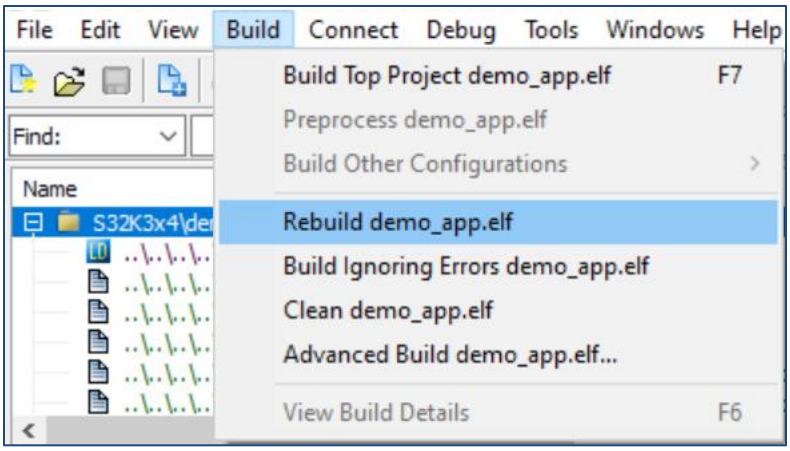


Figure 11. GHS Build top Menu

- Make sure project build status is successful as shown in Figure 13.

```
Compiling hse_mu_config.c because hse_mu_config.o does not exist
Compiling hse_debug_key.c because hse_debug_key.o does not exist
Compiling hse_extend_cust_security_policy.c because hse_extend_cust_security_policy.o does not exist
Compiling hse_fw_update.c because hse_fw_update.o does not exist
Compiling global_variables.c because global_variables.o does not exist
Compiling hse_config.c because hse_config.o does not exist
Compiling utest.c because utest.o does not exist
Linking demo_app.elf because host_flash.o has changed
Done
Build successful (Wed Jul 28 12:12:44 2021)
```

Figure 12. GHS Build status

- Above build steps should be followed for *"SecureBootApp"* GHS projects also.

5. Device configuration

5.1. Features Configuration

Following table lists down the status of various features when device “LC = CUST_DEL”. Many features are configurable directly or indirectly by application software.

Feature	Configuration Information	Status	Configurability
OTA functionality	Samples are always shipped in the OTA disabled configuration	Disabled	Yes. Can be enabled by application by requesting to HSE firmware or SBAF
HSE Firmware usage feature flag	This flag indicates whether firmware can be installed in the device or not.	Disabled	Yes. Can be enabled by programming in the UTEST location.
SBAF firmware	Samples are always shipped with SBAF programmed	Programmed	No
HSE Firmware	HSE firmware is always programmed by user. Encrypted and signed firmware image are always delivered to user.	Not programmed	Yes. Can be installed by the application software. Application software needs to add the address of

Feature	Configuration Information	Status	Configurability
			encrypted firmware image in IVT.
Image Vector Table	No IVT is programmed at any of the IVT locations.	Not programmed	Yes. Can be programmed by application software.
SWT0	Application core watchdog SWT0 is disabled.	Disabled	Yes. It can be optionally enabled by SBAF if SWT bit is configured in boot configuration word.
Boot Sequence	Boot sequence is non-secure boot i.e. application is booted by SBAF without any authentication.	Non secure boot	Yes. It can be changed to secure boot by programming the BOOT_SEQ bit in IVT.
Life Cycle	Samples are always delivered in LC=CUST_DEL	Customer Delivery	Yes. It can be changed to OEM_PROD and IN_FIELD by application through SBAF and HSE firmware.

Feature	Configuration Information	Status	Configurability
Clock Source	Fast Internal Reference clock	FIRC	Yes.
Application debug authorization mode	Debug Authorization mode of application cores is password based. It can be changed to challenge response mode by programming the configuration in UTEST.	Password based approach	Yes. Debug authorization mode can be changed to challenge response mode by requesting to HSE firmware.
Application core debug status	Debug of application cores is enabled in customer delivery	Enabled	No

5.1.1. HSE FW feature Usage flag

The user can enable HSE FW usage for application by programming HSE FW feature usage flag at 0x1B000000 location of 8 bytes. UTEST is one-time programmable only. On the device delivered to customer, the value at this location should be unprogrammed so that the SBAF assumes the default configuration i.e., the HSE firmware cannot be installed in the system and application does not intend to use the HSE FW on the device. If any value is written anything, then the application informs SBAF that it wants to use HSE FW in the device.

In demo code, following is implemented:

- The code first checks if HSE FW feature is already programmed or not by reading 8 bytes value from 0x1B000000 location and comparing it with 0xFFFFFFFFFFFFFFFF, which is default value.

- If the value present at above mentioned location is the default value, then it is concluded that HSE FW feature flag is not enabled.
- In this case, if user selects to enable HSE FW feature flag from trace32, then the code writes value 0xAABBCCDDDDCCBBAA value at specified address.
- If the value at HSE FW feature flag address is not the default value, then it is concluded that HSE FW feature flag is already enabled and there is nothing to do.

In demo app, user can enable HSE FW feature flag by following below steps:

- Open T32 (t32marm.exe) with Lauterbach connected to your board.
- Run menu.cmm script from Trace32 as shown below:

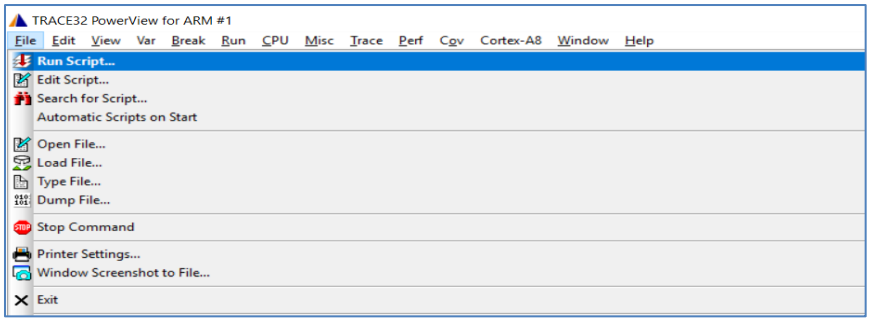


Figure 13. Run script option from Trace32

Note: Once user selects either of the option mentioned below, further any images are loaded as per that i.e. if user selects GHS then user has to make sure that secure boot application and demo application images is already built using GHS compiler and similarly when user selects S32DS.

Figure 14. First script to be run

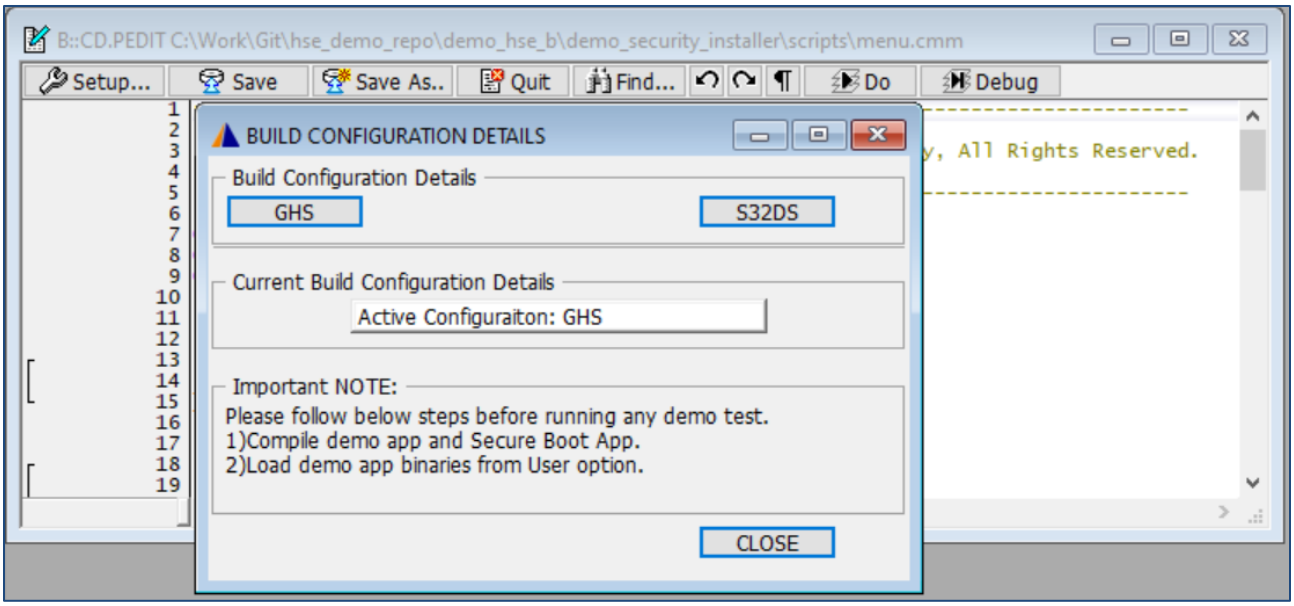


Figure 15. Options to be selected to register build configurations

- In case, user do not intend to install HSE and only enable HSE FW feature flag, then user should select “Load demo app binary” option as shown below.

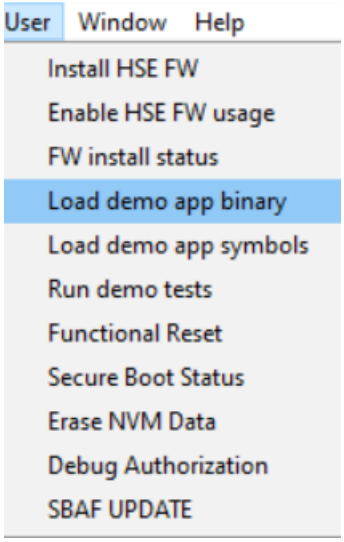


Figure 16. Load Demo app binary option

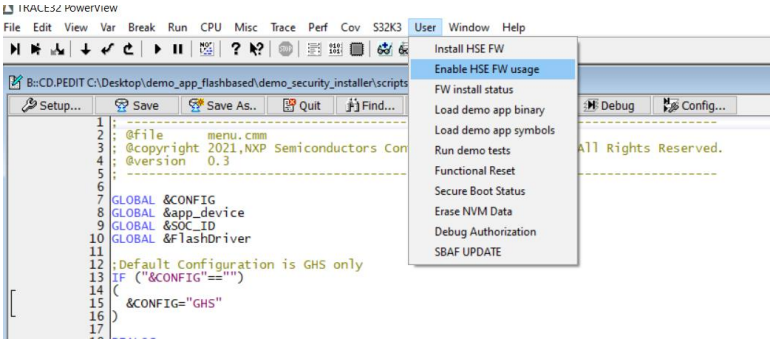


Figure 17. HSE FW Feature Flag Enable

- Once above step is completed, functional reset is issued. After that, user should select "Enable HSE FW usage" option as shown above.
- Confirmation from user is asked if user wants to proceed directly and check the output or wants to debug the code.

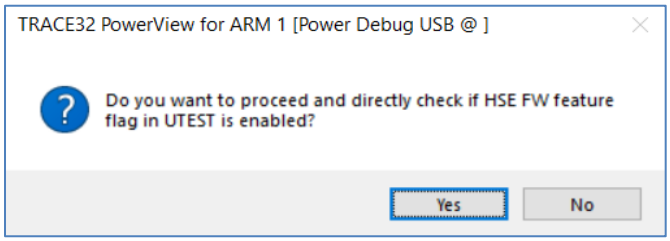


Figure 18. Confirmation Pop-up

- If user selects Yes to proceed, then the user is informed with a pop-up when HSE FW feature flag is written as shown below if default value is present.

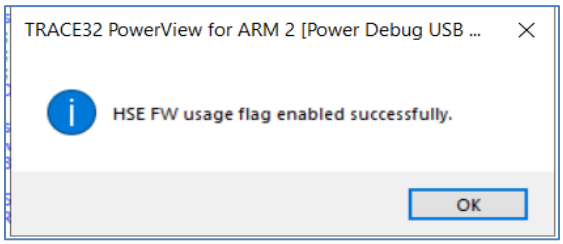


Figure 19. HSE FW usage flag Output

- If HSE FW feature flag is already programmed, then user is notified of the same with below message.

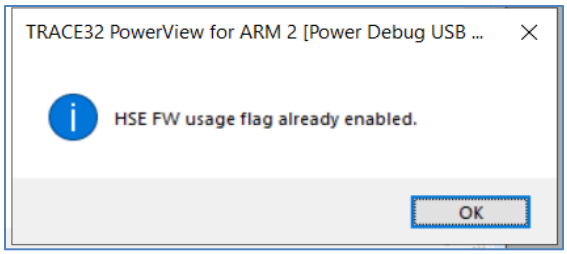


Figure 20. HSE FW usage flag already programmed output

- For confirming, user can take memory dump of 0x1B000000 address. The value should be written as shown in below Figure 20.

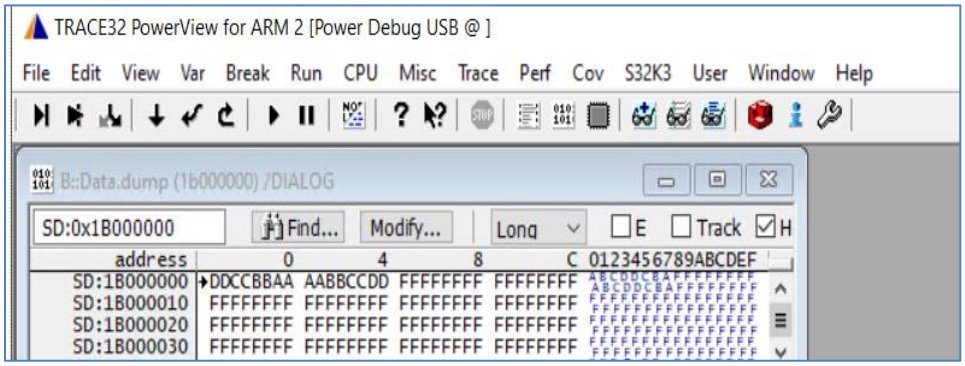


Figure 21. HSE FW feature usage flag

Note: This is the first step that user should execute before installing HSE FW. This is one-time activity only, and the user should not run this step again once executed.

5.2. Memory Regions required by HSE

Refer HSE Firmware Reference Manual available with the release.

5.3. HSE FW installation

5.3.1. Relevant Terms

- IVT
- HSE FW
- Demo application Image
- HSE FW image-header tag
- HSE FW pink image

For more details about these terms see Glossary and Terms

5.3.2. Description

The HSE FW installation is a one-time process and once HSE FW is in the system, it can only be updated further. For more information, refer to HSE Firmware Reference Manual.

This package supports two out of three ways of HSE firmware installation:

- HSE FW installation with IVT
- HSE FW installation without IVT

The user can execute either one of the installations by following below steps:

HSE FW installation with IVT

The Full Mem HSE FW image and AB SWAP HSE FW image should have header marker value as 0xDA and 0xDB respectively. In the demo app, the script programs

- IVT at 0x00400000

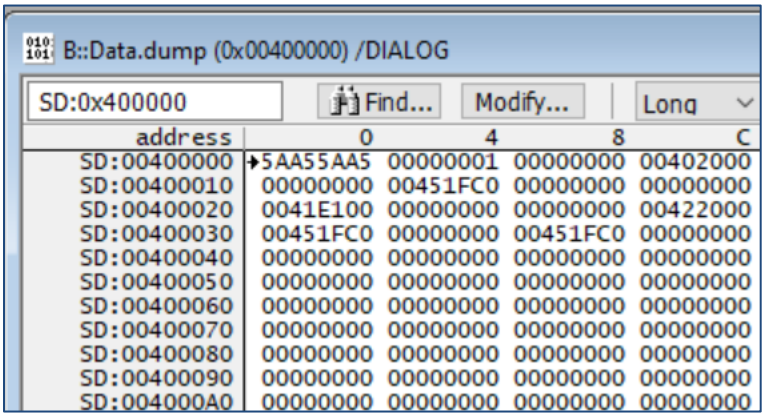


Figure 22. IVT programmed

- Full Mem or AB SWAP HSE FW pink image at 0x00422000

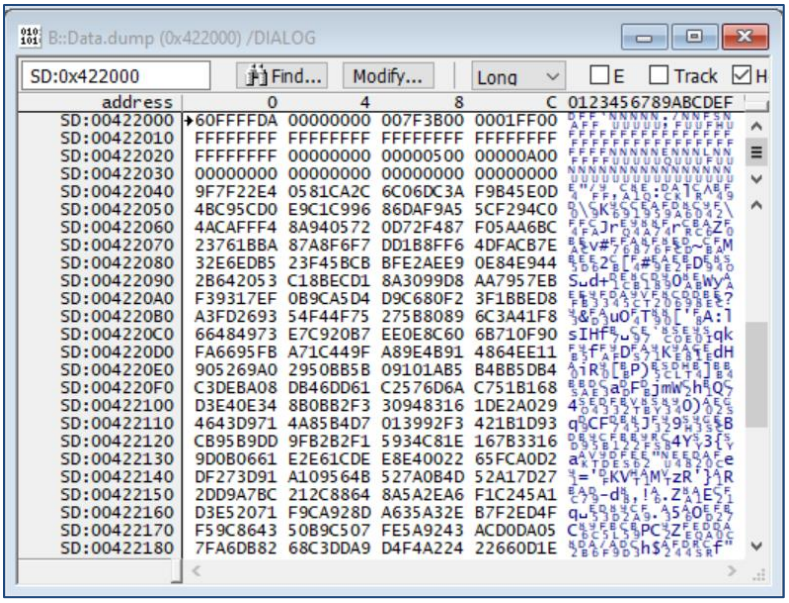


Figure 23. Full Mem HSE FW programmed

Important: This is just an illustrated image and actual image may vary.

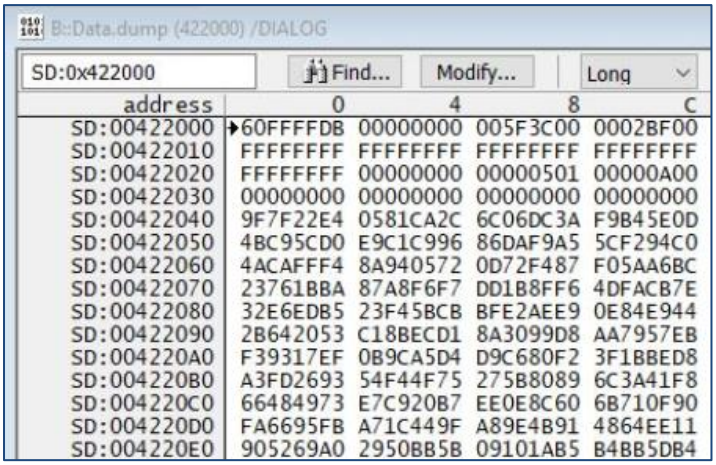


Figure 24. AB SWAP HSE FW image programmed

Important: This is just an illustrated image and actual image may vary.

- Demo app image at 0x00402000

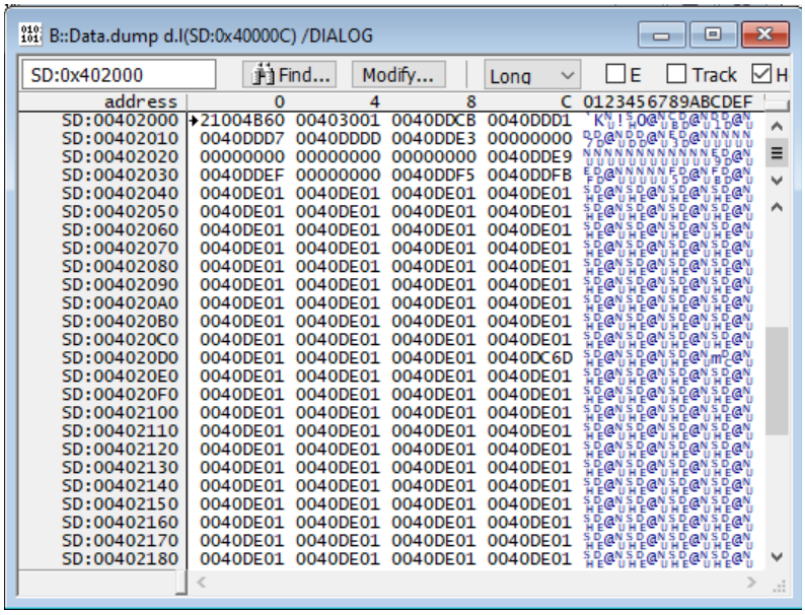


Figure 25. Demo application programmed

- Secure boot app image along with app header at location 0x00451FC0.

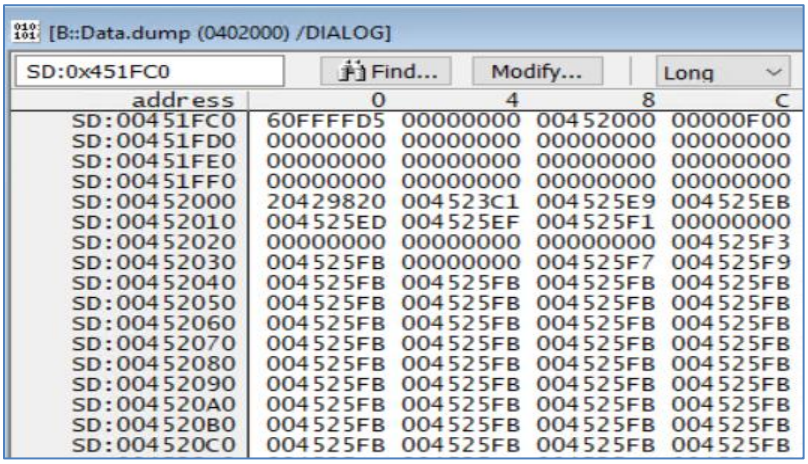


Figure 26. Secure boot application programmed

Follow below steps for installation process:

- Keep secure boot app and demo app images ready.
- Click on “User” and select “Install HSE FW” option from the drop-down as shown below:

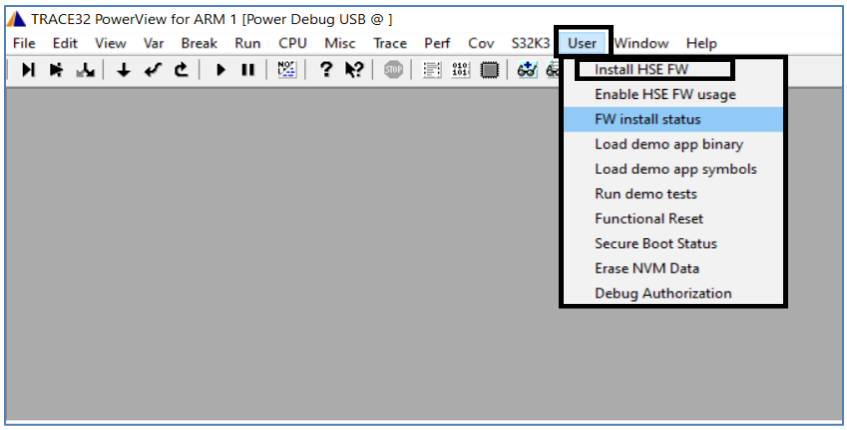


Figure 27. Main menu

- Following the above, the user is prompted with another pop-up as shown below:

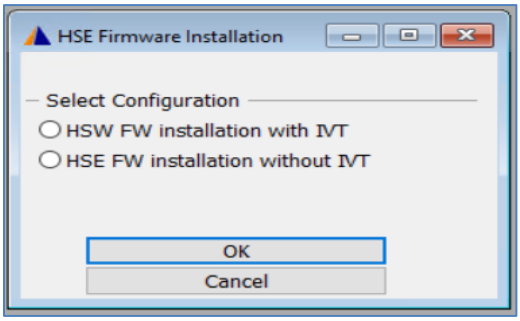


Figure 28. Firmware installation options

- Select HSE FW installation with IVT as shown in Figure 27 and click “OK”.
- The user must select HSE firmware pink image to be loaded on device.
- If the user selects AB SWAP HSE FW pink image for installation, a confirmation message appears for user as shown below.

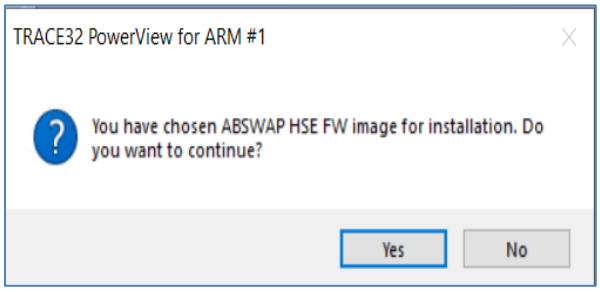


Figure 29 AB SWAP FW pink image selection

- After loading HSE, the user is informed of completion of loading images as shown below:

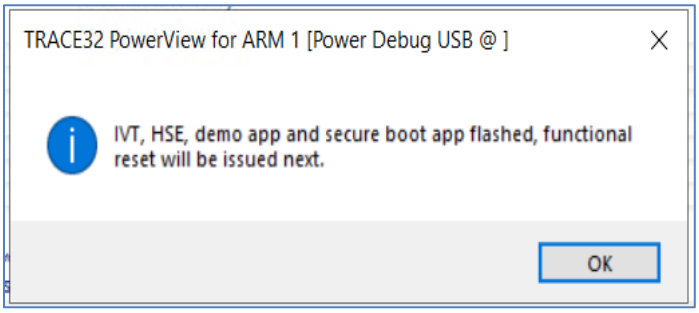


Figure 30. Pop-up informing user of completing of loading images

- After functional reset is issued, the user should check the firmware install status by selecting "FW install status" from the main "User" menu. Depending upon the type of HSE FW selected while installing, either "Full Mem HSE FW installed" or "AB SWAP HSE FW installed" pop-up is displayed, one of which, is shown in Figure 30.

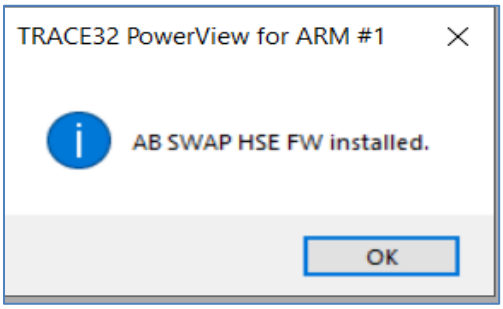


Figure 31. HSE FW installed.

- FW install is said to be successful when HSE FW init bit 0 at location 0x4038C107 is set to 1 as shown below.

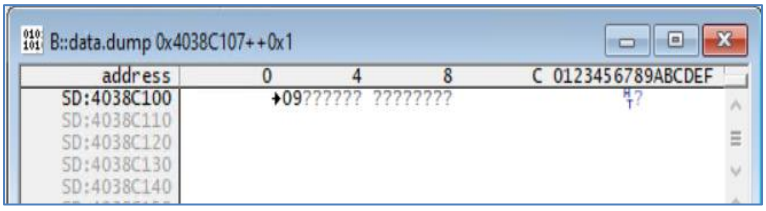


Figure 32. HSE FW install bit

- HSE FW version for AB SWAP HSE FW is also printed on Watch Window.

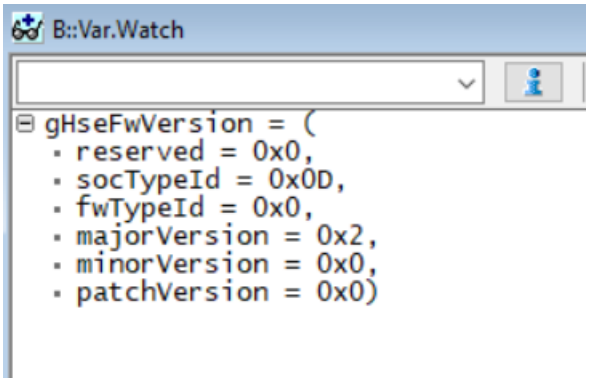


Figure 33. Example of HSE FW version (Actual version may change)

HSE FW installation without IVT

In this case, either FULL MEM or AB SWAP HSE FW pink image is flashed at BLOCK 0 of location 0x00400000 and no IVT is present in code flash as shown below:

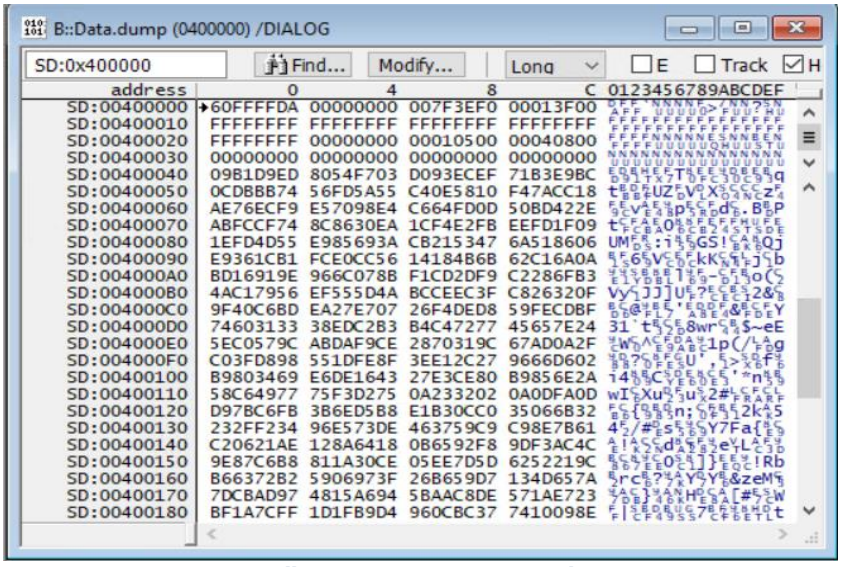


Figure 34. Full Mem HSE FW programmed at BLOCK 0

Important: This is just an illustrated image and actual image may vary.

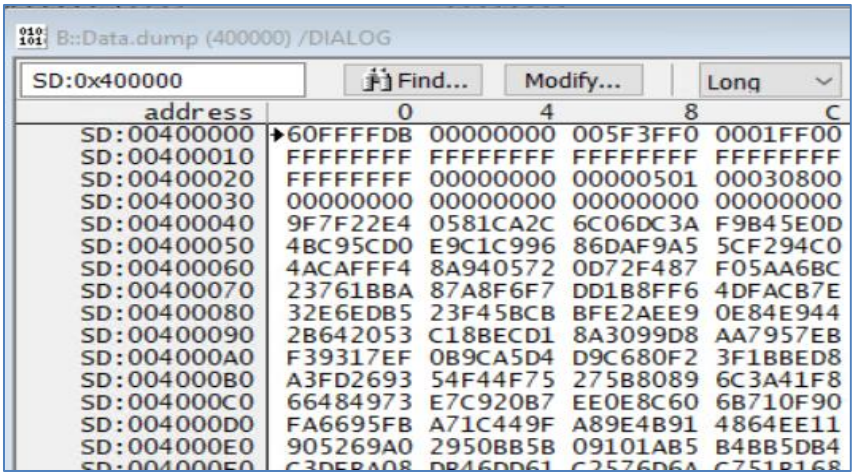


Figure 35. AB SWAP HSE FW programmed at BLOCK 0

Important: This is just an illustrated image and actual image may vary.

Follow below steps for installation:

- Keep secure boot app and demo app images ready.
- Click on “User” and select “Install HSE FW” option from the drop-down as shown in Figure 27.
- Select Full HSE FW installation without IVT as shown in Figure 28 and click ok.
- The user must select HSE firmware pink image to be loaded on device.
- The user is informed of completion of images loading, as shown below:

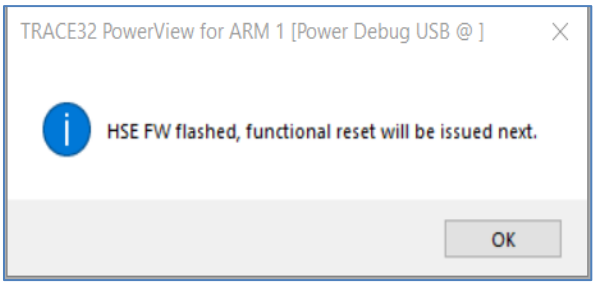


Figure 36. Pop-up informing user of completion of loading image

- FW install is said to be successful when HSE FW init bit 0 at location 0x4038C107 is set to 1 as shown in Figure 32.
- HSE FW version for AB SWAP HSE FW is printed on Watch Window.

6. HSW FW Features

This demo app offers following features of HSE firmware, also shown in Figure 37.

- HSE Attribute Programming
- HSE Cryptographic services
- HSE FW Update
- Secure Boot
- Monotonic Counter Feature
- SHE Services Feature

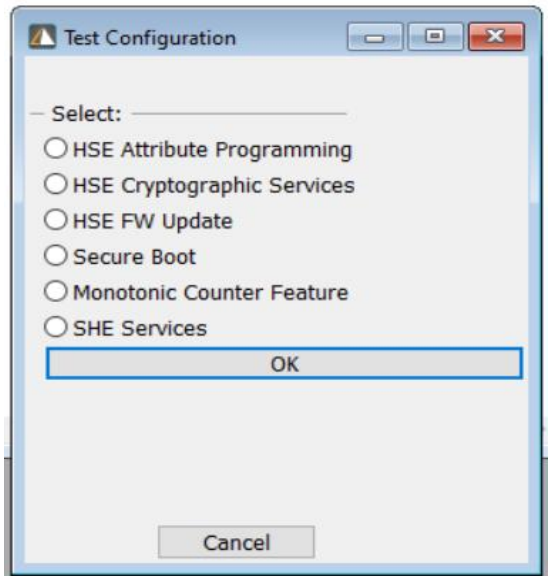


Figure 37. HSE FW features

Each feature example function execution is given as follows:

6.1. HSE Attribute Programming

The demo application offers Attribute Read and Attribute Write both. To execute this example feature, select “Run demo tests” option as shown in Figure 17. The user is prompted with lists of HSE FW features that can be tested as shown in Figure 37, select “HSE Attribute Programming” option from it.

Now the user can choose what kind of attribute it wants to run from below options:

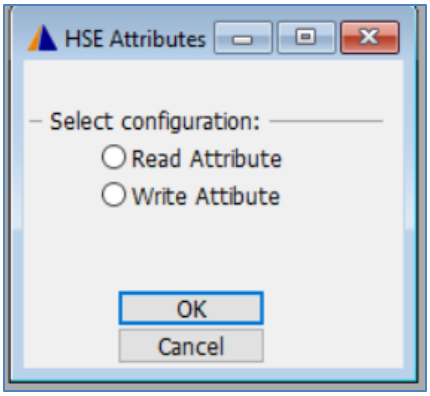


Figure 38. Attribute options

6.1.1. Read Attribute

This demo application offers following attributes that can be read:

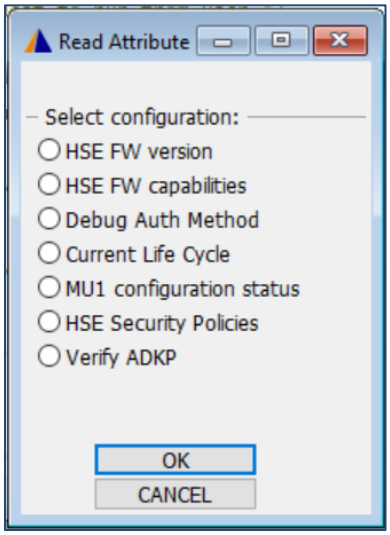


Figure 39. Read Attribute options

HSE FW capabilities

HSE FW capabilities is demonstrated to show how read attribute works. The same is applicable for rest of the attributes except for *"Verify ADKP"*.

- The user should select HSE FW capabilities option and click ok.
- After this, the user gets the output printed in Watch window along with pop-up as shown below:

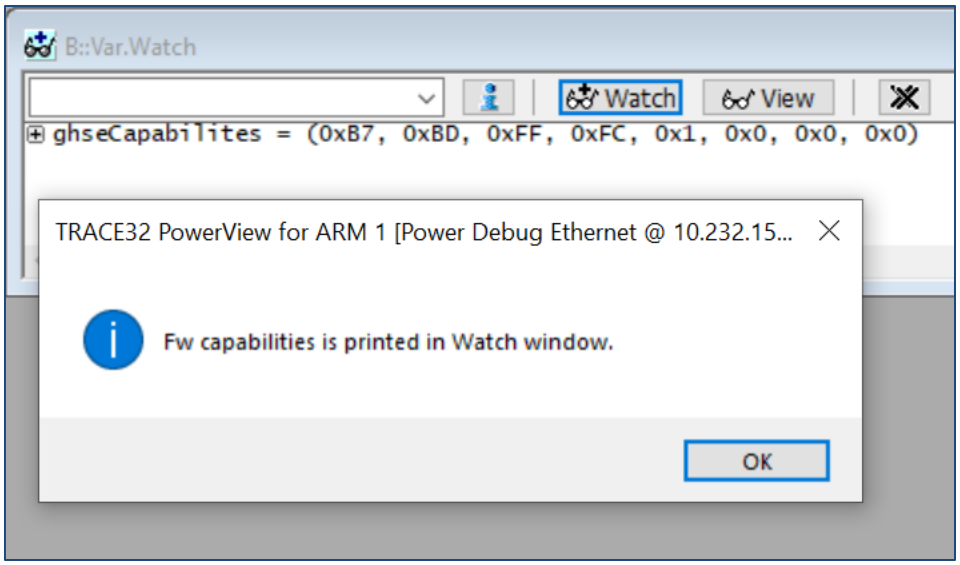


Figure 40. Watch Window and pop-up message.

Important: This is just an illustrated image and actual image may vary.

Verify ADKP

This option offers user to verify programmed ADKP key in the device against the text input in which ADKP is given. To run this feature, follow below steps

- Select “Verify ADKP” option and then click ok. Below message pops-up in which user must confirm the ADKP in text file which is used for checking if ADKP programmed in the device is as per the file by calculating hash of ADKP key in text file and reading ADKP hash from the device and comparing both. If the hashes are equal, then ADKP programmed is as per text file else not.

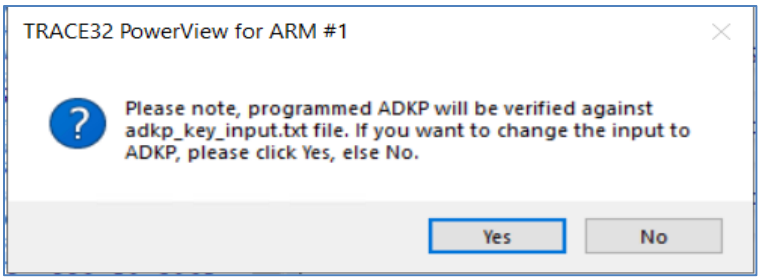


Figure 41. Confirmation from user to confirm ADKP in text file

- If the user wants to update ADKP, select Yes from Figure 41, text file opens, and user can edit it. If user wants to proceed then select No from the Figure 41.
- If the programmed ADKP is not as per text file, then Figure 42 message pops up and if ADKP programmed in the device is same as in text file, then Figure 43 message pops up.

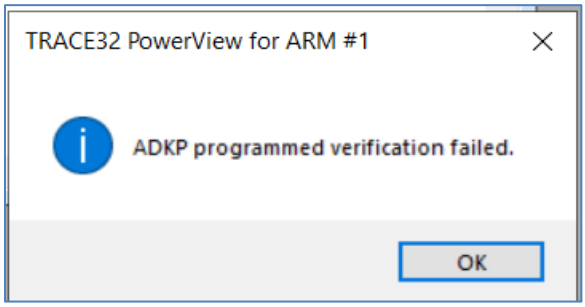


Figure 42. ADKP programmed in the device is different from the one in text file or ADKP is not programmed.

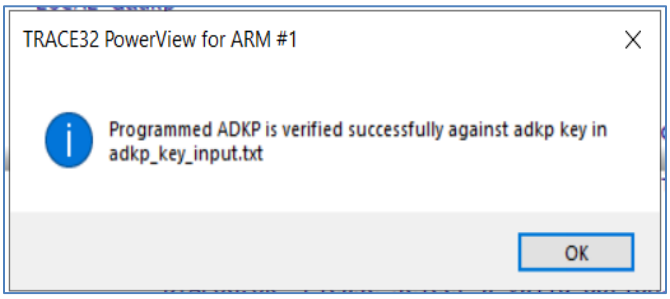


Figure 43. ADKP programmed in the device is same as in text file.

- During this time, GMAC of IVT, required for non-secure and secure boot, is calculated appended at the end of IVT. This IVT is saved in code memory as a backup required for boot authentication as shown in below figures.
- For example, ADKP = 0x000102030405060708090a0b0c0d0e0f and enableADKm =0, below are the GMAC calculated and appended for non-secure and secure boot IVT:

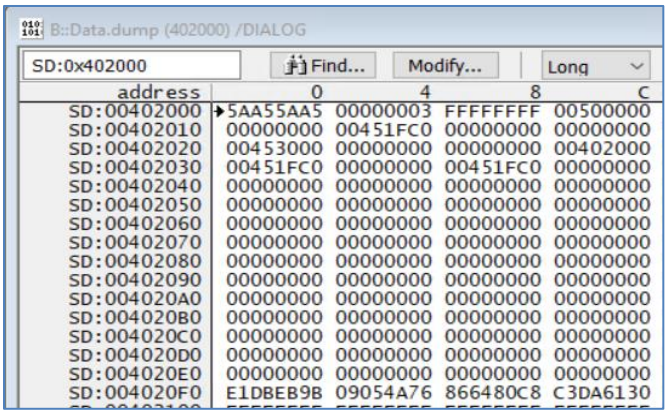


Figure 44. Non-secure boot IVT backup with GMAC appended

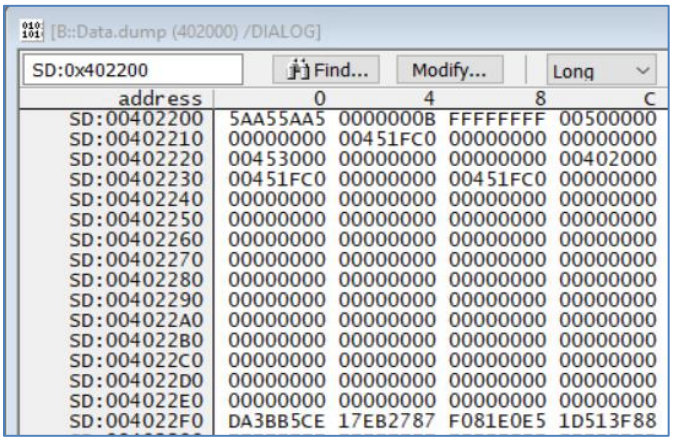


Figure 45. Secure boot IVT backup with GMAC appended

- When user enables boot authentication i.e. IVT is authenticated by SBAF during boot, the backup IVT with GMAC appended, depending upon the type of boot i.e. secure or non-secure, is loaded onto BLOCK 0 IVT address.

  Note: For GMAC calculation, ADKP should be programmed.

  For enableADKm = 1, following is the flowchart:

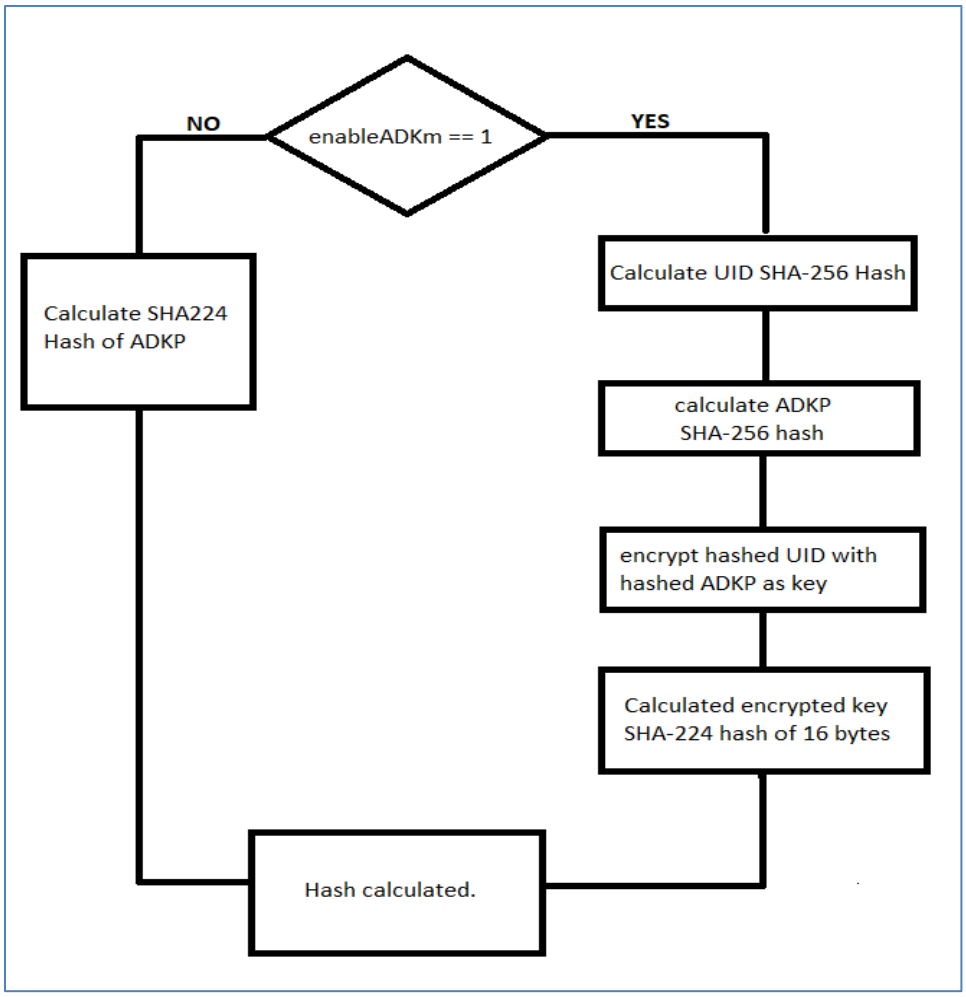


Figure 46. Verify ADKP when enableADKPm=1

6.1.2. Write Attribute

This demo application offers following attributes that can be written:

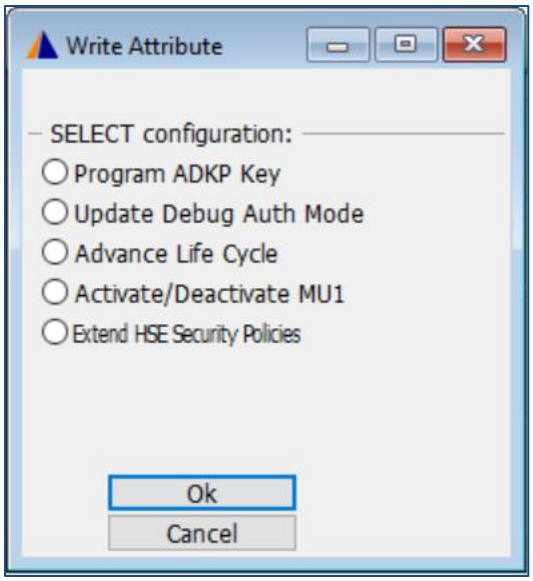


Figure 47. Write Attribute options

Program ADKP key

Here, only 1 example, Program ADKP Key is demonstrated to show how write attribute works. The same is applicable for rest of the attributes too. ADKP key programming feature allows user to update a 128-bit application debug password/key (ADK/P) at UTEST location 0x1B000370 which is required for debug authorization.

Note: Option 1,2,3 and 5 are one-time programmable attributes only and these operations cannot be reverted.

Follow below steps for Programming ADKP key:

- The user should select first option and click ok.
- This demo app takes ADKP key input from a text document named “adkp_key_input.txt”. The user is requested to update the text document with AKDP key if the user wants its own ADKP key to be programmed in the device. For this, when user clicks ok in Step 1, the user is prompted with another pop-up for updating the text document before sending write attribute request as shown below:

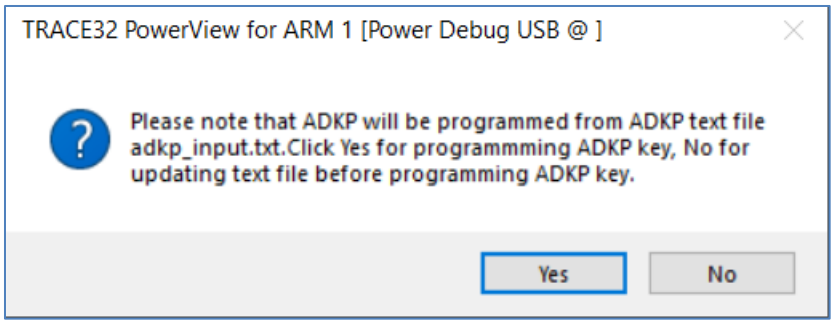


Figure 48. User prompted to update ADKP key in text document

- If the user wants to continue with whatever ADKP key is already there in text document, then user should click Yes. If the user wants to update the text document before ADKP program request, then user should click No.
- Once the user clicks No, user have text document open up as shown in Figure 50. The input for text document is in hex format only (Figure 50).

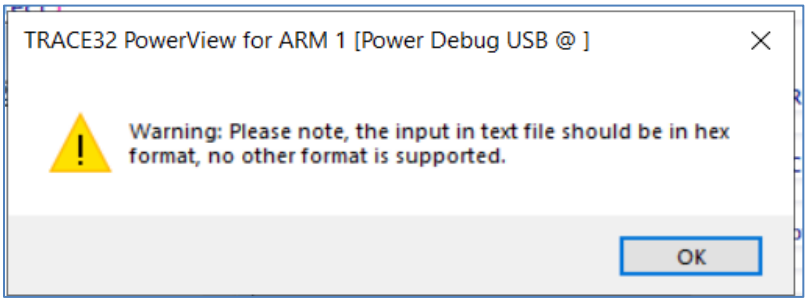


Figure 49. Pre-requisite for updating text document

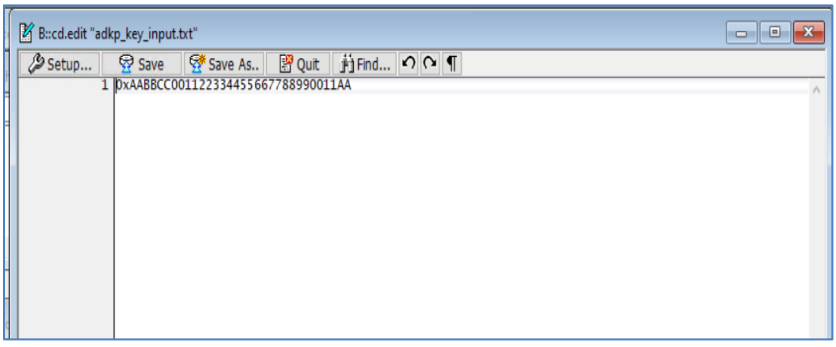


Figure 50. ADKP key input text file

- After updating text document, user is requested to close it and then again run attribute programming from Run demo tests option in User menu and then repeat Step 1 and 2.
- This time, user should select yes for Step 3.
- Now, user is prompted to either continue and directly check the output without debugging the demo code or not continue and debug the code.
- If the user clicks Yes, then user is informed of the output directly as shown in Figure 51. If the user clicks No, then user can debug the demo code, and check the output themselves as shown in Figure 52.

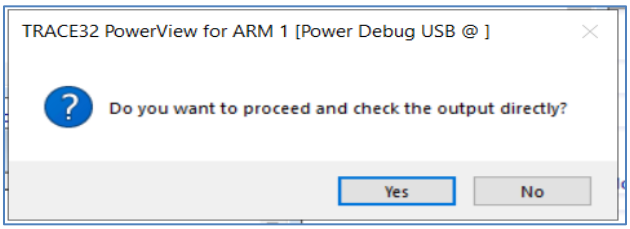


Figure 51. User prompted for checking the output or not

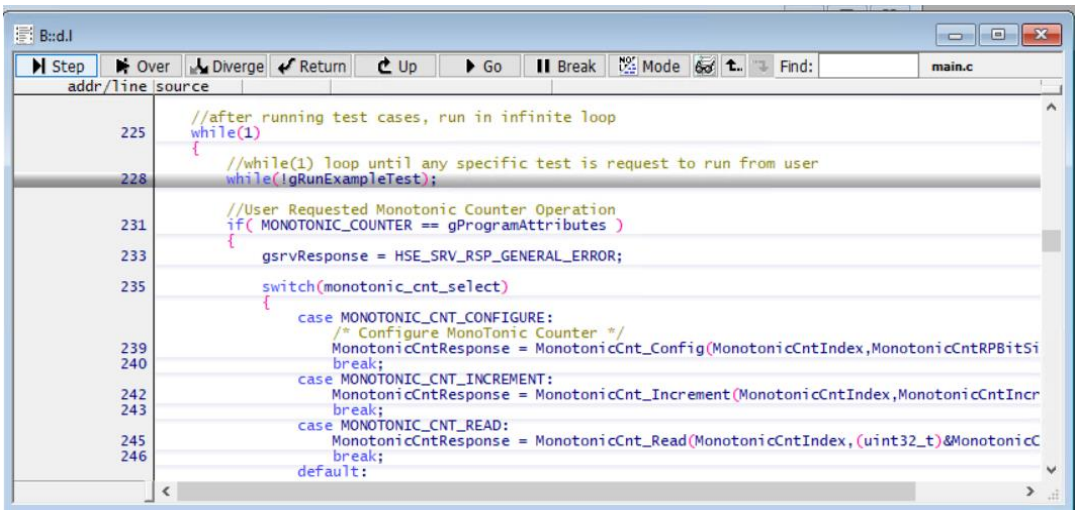


Figure 52. User selects No and wants to debug the code.

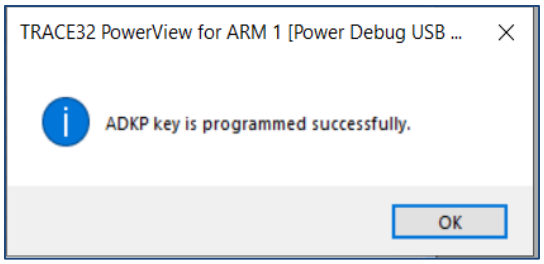


Figure 53. ADKP key programmed

- If ADKP key is already programmed and user requested to program ADKP key again, then user is notified of the same with below message.

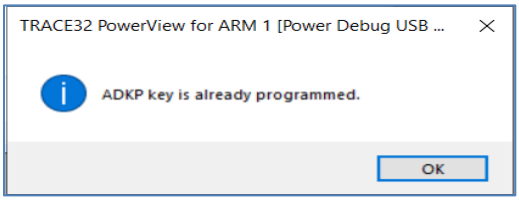


Figure 54. ADKP already programmed.

- For Debug auth mode and update Life Cycle attributes, ADKP key must be programmed, or else user is not be allowed to program these two attributes. In case, user has not programed ADKP key, the user is informed about the same with below message.

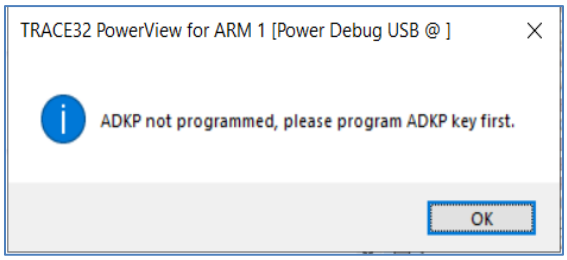


Figure 55. ADKP key not programmed

Update Secure Life Cycle

Life Cycle can only be promoted and not demoted. Once ADKP is programmed (mandatory step), life cycle can be advanced from CUST_DEL (by default) to any other (either OEM_PROD or IN_FIELD). This is one-time programmable only and this operation cannot be reverted. From Figure 47, select option “Advance Life Cycle”. The user is informed of response as shown in below figure, if LC is advanced, it is reflected after POR.

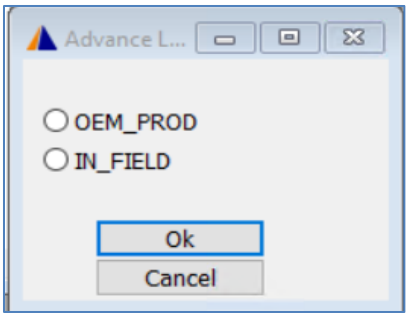


Figure 56. LC advance options

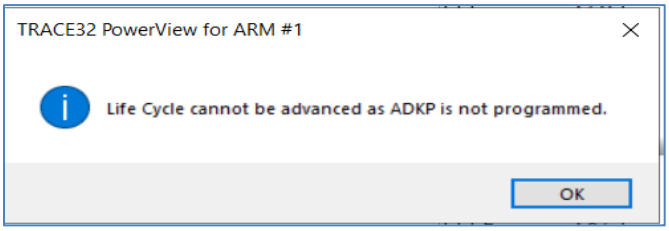


Figure 57. LC advance negative response reason 1

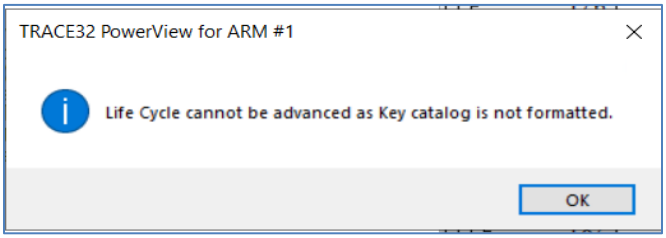


Figure 58. LC advance negative response reason 2

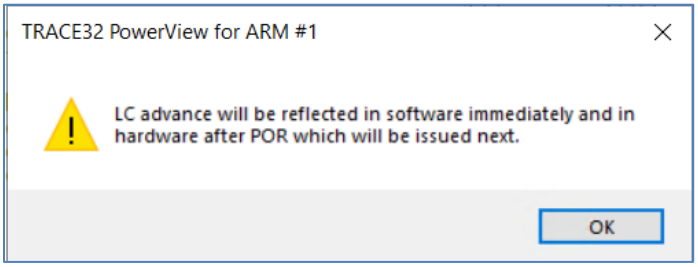


Figure 59. LC advance positive response

Configure MU1

This allows user to configure MU1 which can be activated or de-activated as per user needs. To run this feature, select fourth option and click ok. This further prompt user to either activate or de-activate the MU1 as shown in below figure:

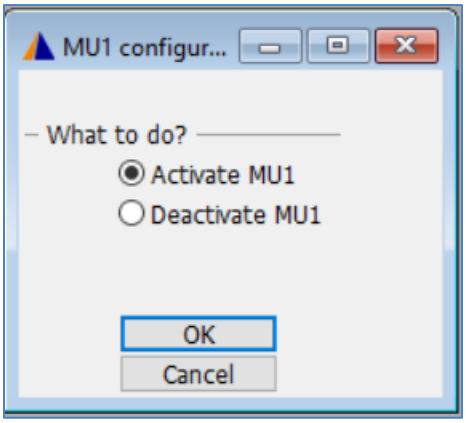


Figure 60. MU1 activate or deactivate options

The user can verify if MU1 was configured as per its needs by reading the MU1 configuration (select Option 5 from Figure 39).

Extend HSE Security Policies

The user can optionally diversify ADKP key with the device's UID before being written in secure NVM. This allows to provision a device-dependent debug key (or password) and to use ADKP as a master debug key: the device-dependent key can be calculated based on the UID and the knowledge of the master key is which is never shared. This diversification option is controlled by the HSE system attribute ADKP_MASTER: when set, the input value to the set attribute service is encrypted with a 256-bit hash of the device's UID before being saved in secure NVM.

The user can enable ADKP_MASTER also known as enableADKm bit and StartAsUser bit by selecting "Extend HSE Security Policies" option from Figure 47. The user is further prompted to configure enableADKm bit enable or disable and configure StartAsUser bit enable or disable as shown in below figures:

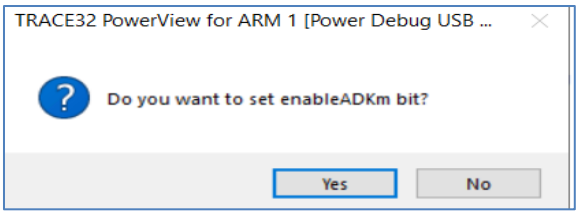


Figure 61. Configure enableADKm

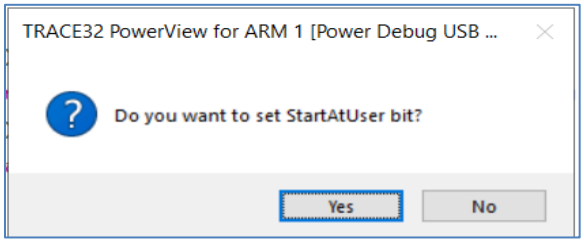
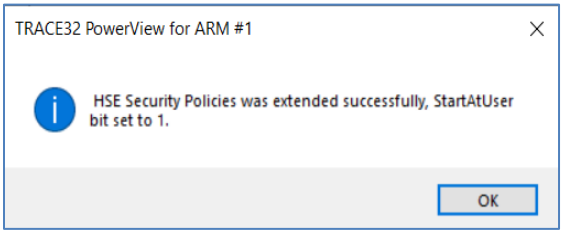


Figure 62. Configure StartAsUser



or

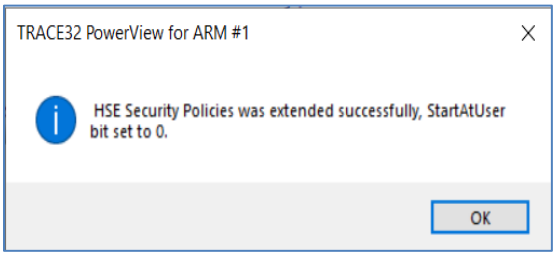


Figure 63. Extend HSE Security policies positive response

Note: This operation is one-time programmable only and cannot be reverted. Also, ADKP key should not be programmed before extending CUST_DEL security policy. If user tries to execute this operation after programming ADKP key, the operation is not be allowed and below message is printed.

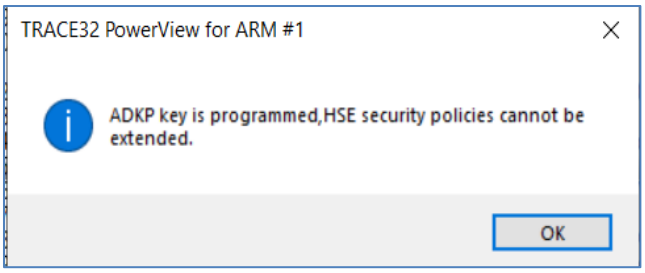


Figure 64. Extend Security Policies not allowed

6.2. HSE Cryptographic Services

This application offers following cryptographic services for demonstration.

- AES keys related example: The demo application does the following operations in this example case:
  - Import Symmetric Keys
  - AES ECB encrypt and decrypt
  - AES Fast CMAC generate and verify along with fast CMAC time calculated (50 Requests).
- ECC keys related example:
  - Import ECC keys
  - ECDSA sign and verify
  - Export ECC compressed keys, ECC raw keys and ECC uncompressed keys
- Key derivation related example:

Key derivation using KDF algo PBKDF2

Key derivation using KDF algo HKDF PRF

Key derivation using KDF algo TLS 1.2 PRF

- Session Keys related example:
  - Generate ECC key pair
  - Import ECC keys
  - Compute DH Shared Secret

- Derive Key using SP800_108 KDF
- Extract from the derived key material 2 keys: 192-bits AES and 256-bits AES.
- AES GCM encrypt and decrypt

- Fast CMAC with Counter:
  - Monotonic Counter Configure
  - Monotonic Counter Increment
  - Fast CMAC Generate with Counter
  - Fast CMAC Verify with Counter
- Burmester Desmedt Protocol:
  - Load ECC Curve
  - Example for Node 1
  - Example for Node 2
  - Example for Node 3
- Sys Authorization related tests:
  - Set the CUST_SUPER_USER authorization key
  - Switch to IN_FIELD User and erase keys
  - Authorize as CUST_SUPER_USER and erase keys
- Update NVM keys related tests:
  - Import Symmetric key and AES GMAC generate keys
- Hash Asynchronous related tests: Sends request asynchronously.

Files which cover above sub-features are explained below:

- HSE configuration services (`hse_config.c`) – file contains the code to format NVM and RAM keys catalog before provisioning them in cryptographic services.
- HSE cryptographic examples (`hse_crypto.c`) – file containing code of all the cryptographic services mentioned above.

To run this feature, follow below steps:

- From Figure 37, select “HSE Cryptographic Services” option and click “OK”.
- This prompts user for either directly running all cryptographic services and displaying output or for debugging the code as shown below:

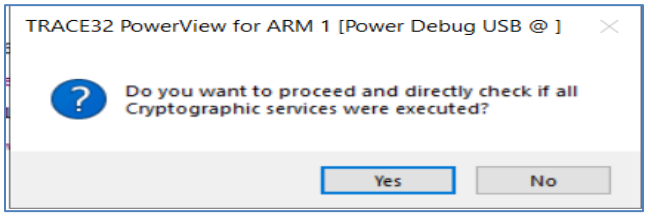


Figure 65. User prompt for direct output display

- If the user clicks ok, the direct output is displayed along with Fast CMAC generate and verify time in microseconds in watch window as shown below:

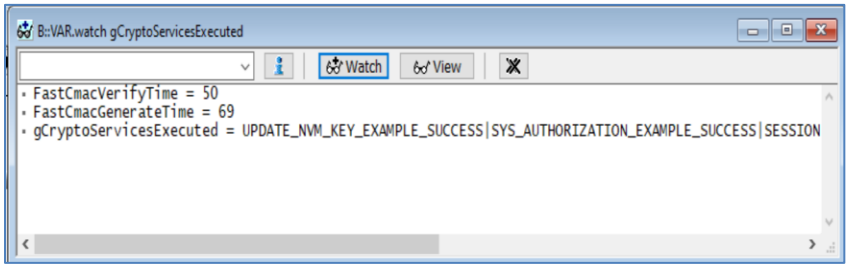


Figure 66. Cryptographic Services output

NOTE: In case, all the cryptographic services are not executed successfully, user should try to issue functional reset and then erase the NVM data, and then try running the Cryptographic services again.

6.3. Secure BAF Update

HSE Firmware offers an option to update the Secure-BAF on the device. User needs to select the option shown in image below in order to update the SBAF of the device.

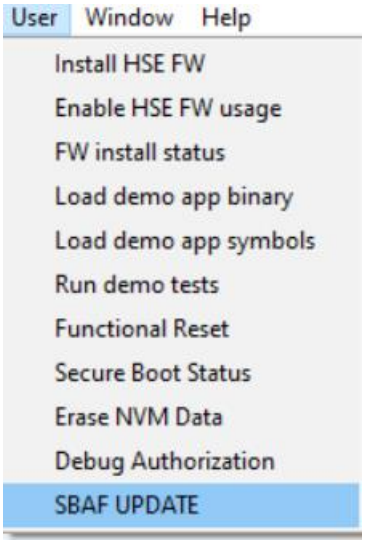


Figure 67. SBAF Update option

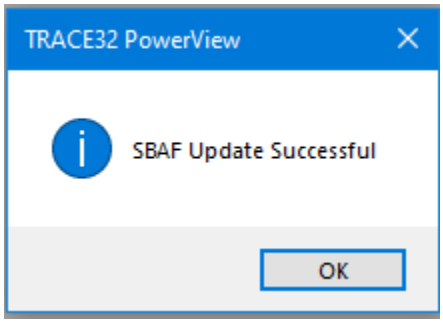


Figure 68. SBAF Update success dialog message

User can check the updated SBAF version on address 0x4039c020.

The following figures show the latest Secure BAF version for various devices:

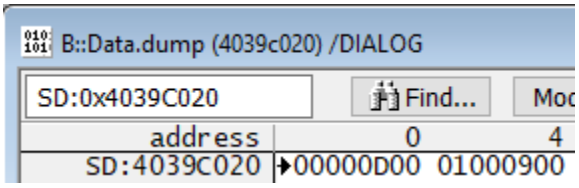


Figure 69: S32K3X2 Standard SBAF Version

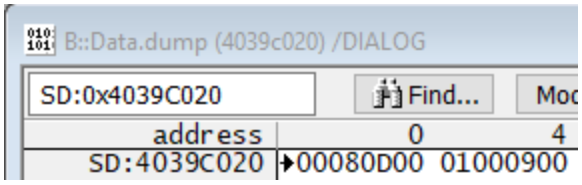


Figure 70: S32K3X2 GM SBAF Version

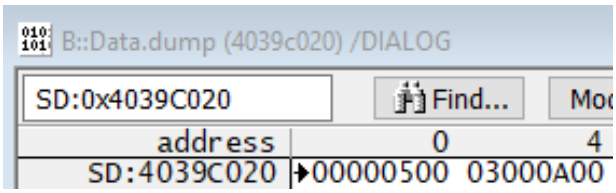


Figure 71. S32K3X4 Standard SBAF Version

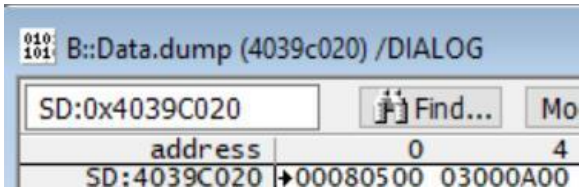


Figure 72: S32K3X4 GM SBAF Version

6.4. HSE FW Update

Once the user has installed the HSE FW in the device, reinstallation is not allowed. New HSE FW version can only be updated through FW update process.

HSE FW can be updated by 2 options:

- One Shot Mechanism
- Streaming Mode Mechanism

One Shot mechanism of firmware update requires entire encrypted image of HSE firmware delivered by NXP to be programmed on Application memory. The firmware update is requested once by the application and the complete firmware update takes place at once.

Streaming Mode mechanism of firmware update requires only a chunk of firmware image to be present on the Application memory. The size of chunk is shared by the application while requesting the firmware update service. The streaming mode happens in 3 stages – init, update and finish. The Init and Finish commands are called only once, while update can be called repeated times, depending on the size of firmware (Refer demo app code for more information). The firmware update completes only after the finish is requested to the firmware.

HSE FW can be updated for 3 different configurations:

- FULL MEM (OTA Disabled) to FULL MEM (OTA Disabled)
- FULL MEM (OTA Disabled) to AB SWAP (OTA Enabled)
- AB SWAP (OTA Enabled) to AB SWAP (OTA Enabled)

In the Full Mem configuration the entire Flash memory is seen as one continuous memory block.

In the AB Swap configuration the Flash memory splits into two block of equal size,

- Active block
- Passive block

In case of AB Swap device, the active partition is the target execution block of flash. The passive partition of flash is the region where an update is done. After updating OTA counter in passive partition and on next reset, passive partition becomes active partition and vice-versa. During firmware update, the firmware backs up itself. Demo app is designed to demonstrate how the user is supposed to back up the applications in passive partition during the Activate Passive Block. For more detail on OTA feature in the hardware, please refer device reference manual.

When the Flash configuration is set to AB_SWAP, it is not possible to go back to the FULL_MEM configuration. Hence, only an AB_SWAP HSE Firmware image can be used in this configuration.

The user can distinguish between Full Mem configuration and AB Swap configuration Firmware with four bytes of header.

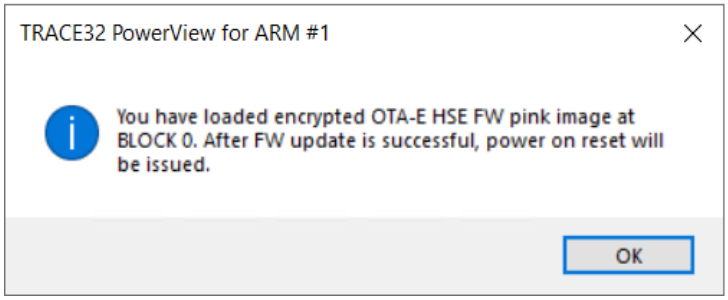


Figure 73. AB SWAP HSE FW image selected for fw update

6.4.1. Encrypted HSE FW Header Format

The HSE Firmware is delivered for a specific Flash memory configuration. The first four bytes in the FW-IMG define the required Flash configuration as described in the below tables:

Table 3. HSE-IMG header for Full Mem HSE firmware configuration

Byte 0	Byte 1	Byte 2	Byte 3
0xDA	0xFF	0xFF	0x60

Table 4. HSE-IMG header for AB SWAP HSE firmware configuration

Byte 0	Byte 1	Byte 2	Byte 3
0xDB	0xFF	0xFF	0x60

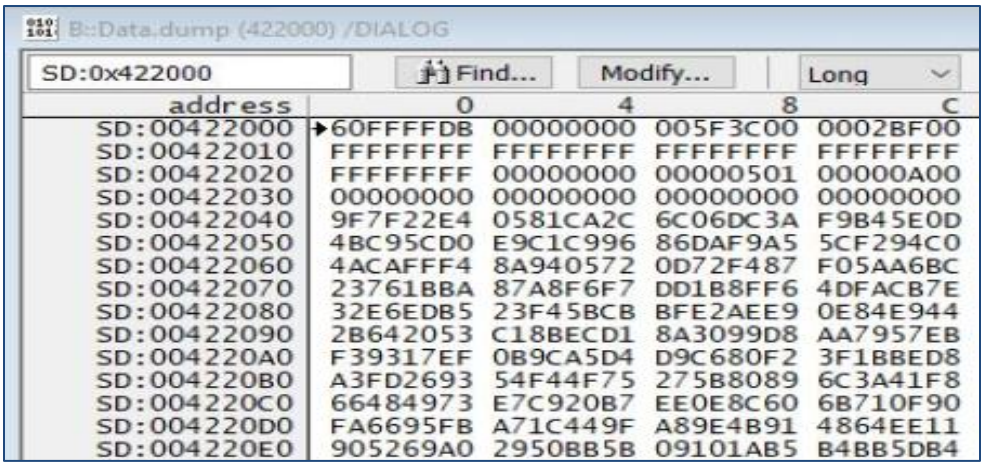


Figure 74. AB SWAP HSE FW selected for FW update

Below figure shows reserved member value of the HSE_FW that was installed in the system when OTA functionality is not enabled.

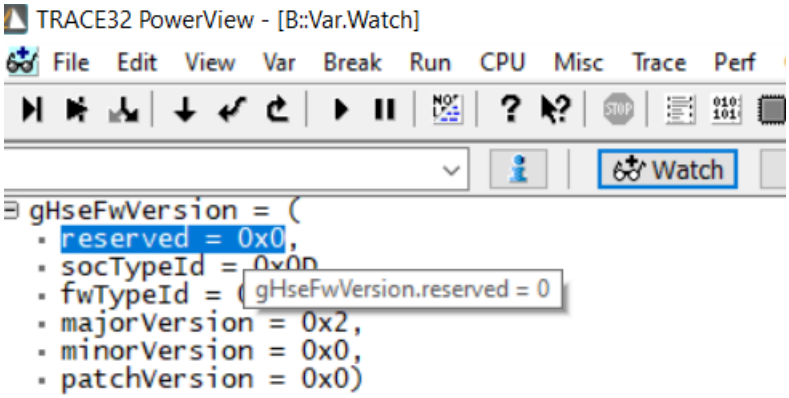


Figure 75. Firmware version reserved member id value when Full Mem HSE FW installed

Below figure shows reserved member value of the HSE_FW that was installed in the system when OTA functionality is enabled.

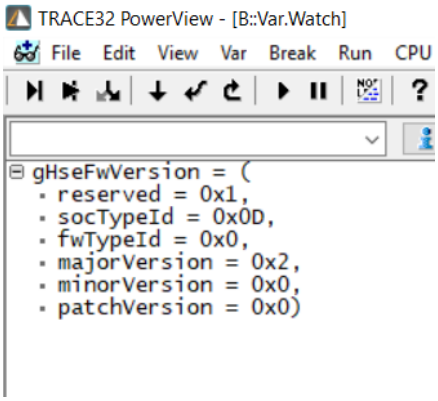


Figure 76. Firmware version reserved member id value when AB SWAP HSE FW installed

The demo app has an option that whenever the user tries to update from AB Swap to Full Mem, the user gets pop-up of not allowed operation.

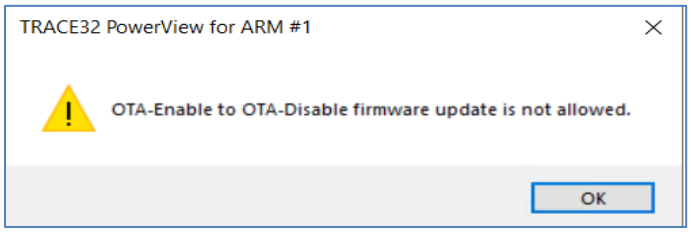


Figure 77. AB Swap to Full Mem fw update

Illustrating AB Swap HSE FW installation. As mentioned earlier, the images are for illustration and explaining the concept taking S32K344 as reference, the actual device addressing may vary according to the device specs. For more details on device specs, refer to HSE Firmware Reference Manual.

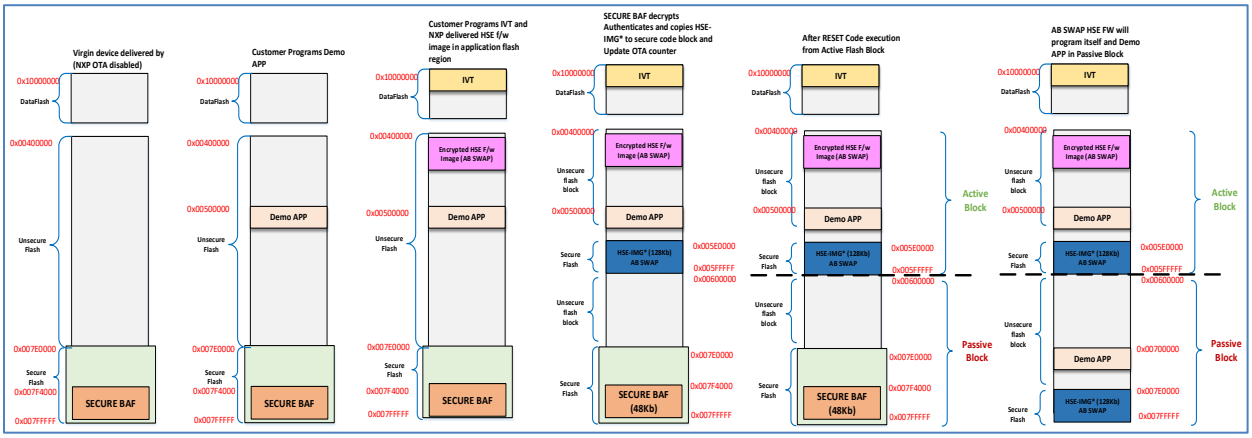


Figure 78. AB SWAP HSE FW installation for first time

6.4.2. FULL MEM to FULL MEM Update

Illustrating the HSE FW update from FULL_MEM to FULL MEM configuration

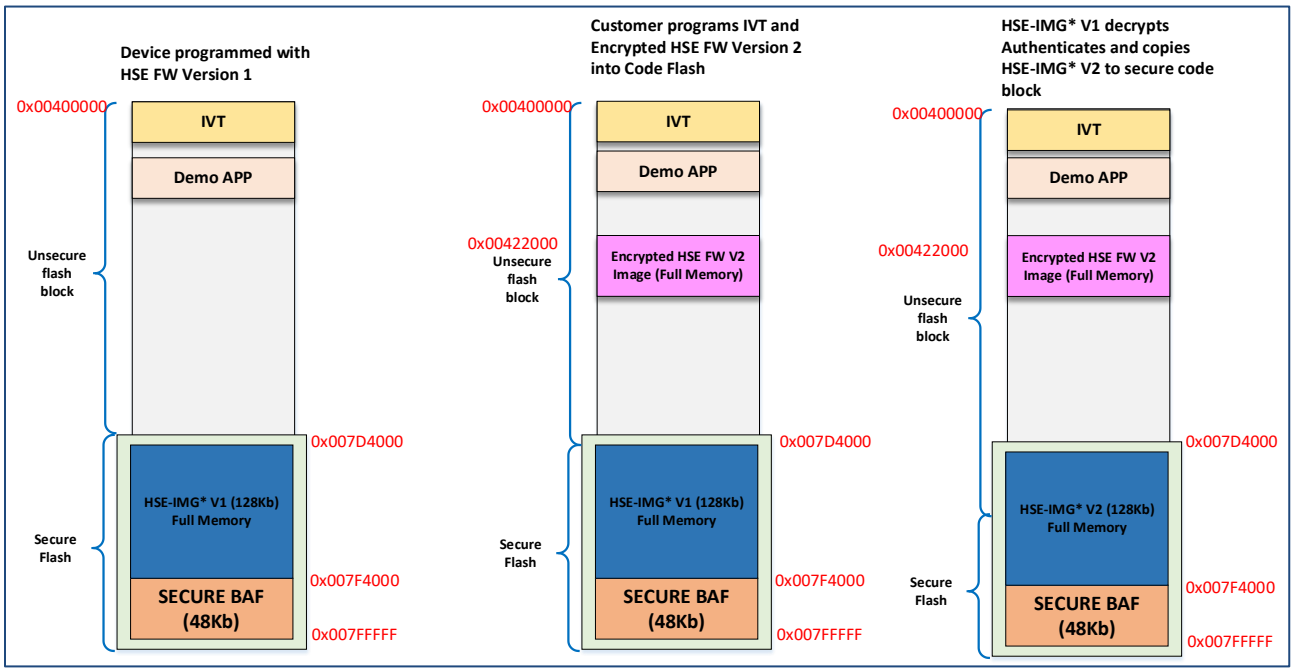
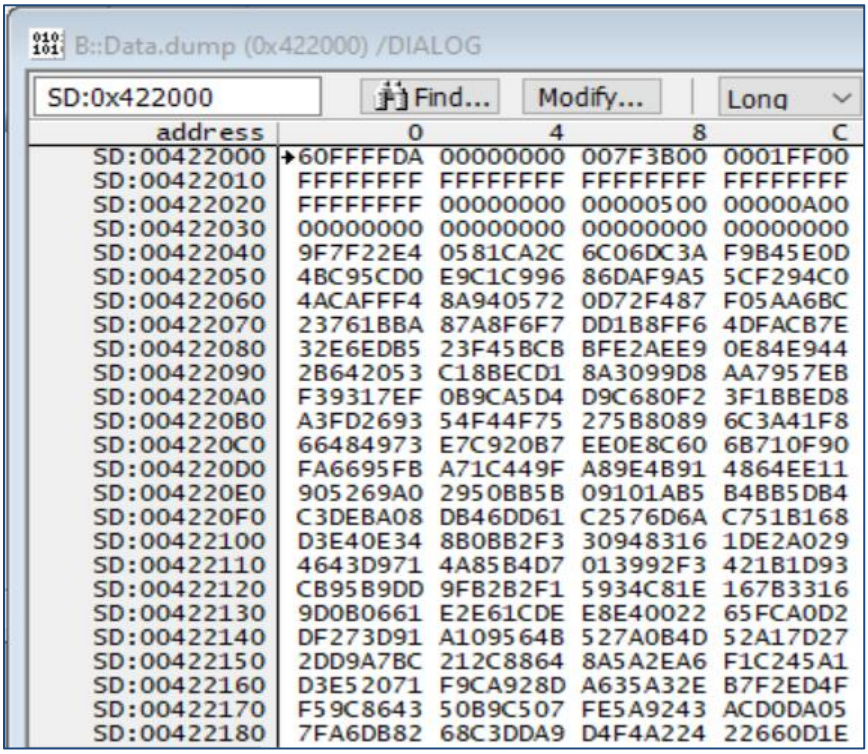


Figure 79. FULL MEM to FULL MEM HSE FW update (the actual device addressing may vary according to the device specs)

Steps:

This demo application offers HSE Firmware update feature via One shot or in Streaming Mode. This document covers how HSE Firmware Update done via Streaming mode. The application triggers firmware update service request to HSE FW. To execute HSE FW update feature, the demo app does the following through the fw_update.cmm script:

- Stores HSE FW pink image in flash memory which needs to be updated.



address	0	4	8	C
SD:00422000	60FFFFDA	00000000	007F3B00	0001FF00
SD:00422010	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
SD:00422020	FFFFFFFF	00000000	00000500	00000A00
SD:00422030	00000000	00000000	00000000	00000000
SD:00422040	9F7F22E4	0581CA2C	6C06DC3A	F9B45E0D
SD:00422050	4BC95CD0	E9C1C996	86DAF9A5	5CF294C0
SD:00422060	4ACAFFF4	8A940572	0D72F487	F05AA6BC
SD:00422070	23761BBA	87A8F6F7	DD1B8FF6	4DFACB7E
SD:00422080	32E6EDB5	23F45BCB	BFE2AEE9	0E84E944
SD:00422090	2B642053	C18BECD1	8A3099D8	AA7957EB
SD:004220A0	F39317EF	0B9CA5D4	D9C680F2	3F1BBED8
SD:004220B0	A3FD2693	54F44F75	275B8089	6C3A41F8
SD:004220C0	66484973	E7C920B7	EE0E8C60	6B710F90
SD:004220D0	FA6695FB	A71C449F	A89E4B91	4864EE11
SD:004220E0	905269A0	2950BB5B	09101AB5	B4BB5DB4
SD:004220F0	C3DEBA08	DB46DD61	C2576D6A	C751B168
SD:00422100	D3E40E34	8B0BB2F3	30948316	1DE2A029
SD:00422110	4643D971	4A85B4D7	013992F3	421B1D93
SD:00422120	CB95B9DD	9FB2B2F1	5934C81E	167B3316
SD:00422130	9D0B0661	E2E61CDE	E8E40022	65FCA0D2
SD:00422140	DF273D91	A109564B	527A0B4D	52A17D27
SD:00422150	2DD9A7BC	212C8864	8A5A2EA6	F1C245A1
SD:00422160	D3E52071	F9CA928D	A635A32E	B7F2ED4F
SD:00422170	F59C8643	50B9C507	FE5A9243	ACD0DA05
SD:00422180	7FA6DB82	68C3DDA9	D4F4A224	22660D1E

Figure 80. New HSE FW flashed for FW update process

- Provide the address of new HSE FW pink image i.e., 0x00422000 to the code.

In code, to confirm that HSE FW update was successful, following is implemented:

- First read version of the current HSE FW on the device and store it in a variable.
- After the Firmware update is requested to HSE, it polls for HSE FW init bit to be set and the response should be HSE_SRV_RSP_OK that confirms HSE FW update is successful.
- After HSE FW update is successful, read again the version of the HSE FW on the device.
- Note: For LC=CUST_DEL, HSE FW update with lower version is also allowed.
- If the HSE FW update is not successful same is informed to the user.

Following are the steps for running this scenario, if Full Mem HSE FW is installed:

- Click on “User” from menu bar, select “Run demo tests” from and click ok.

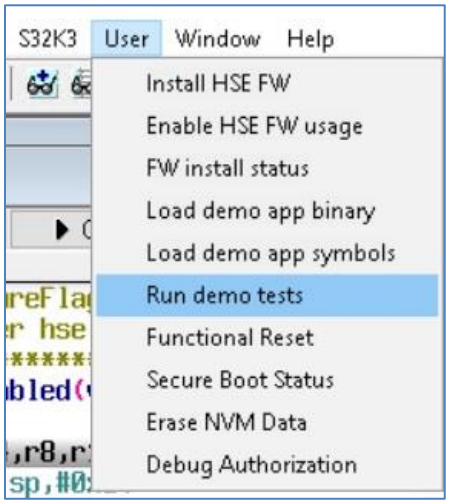


Figure 81. Select “Run Demo Tests”

- Select “HSE FW Update” option from Run demo tests as below and click ok.

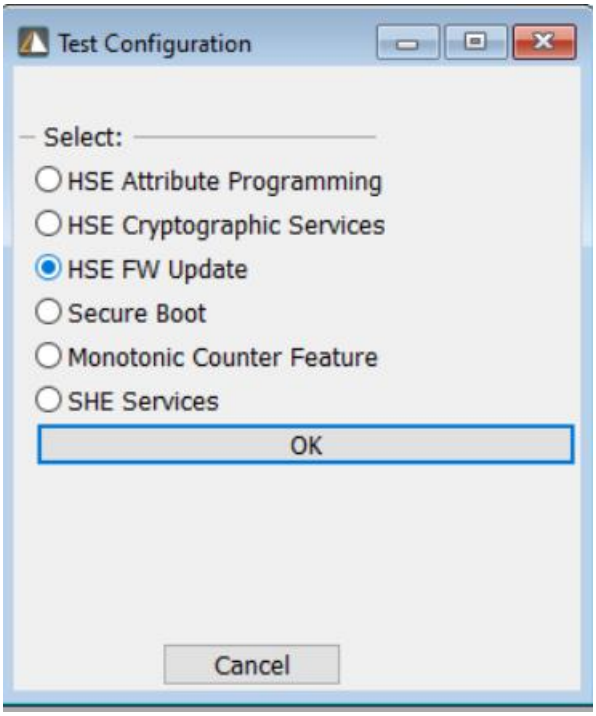


Figure 82. Select “HSE FW Update“

- Select “Update HSE FW” and click OK.

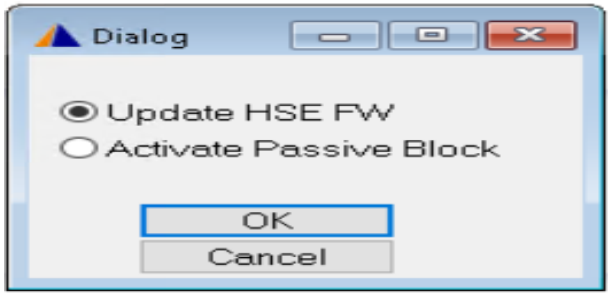


Figure 83. Select “HSE FW”

- Select mode of update “Streaming Mode” and click OK.

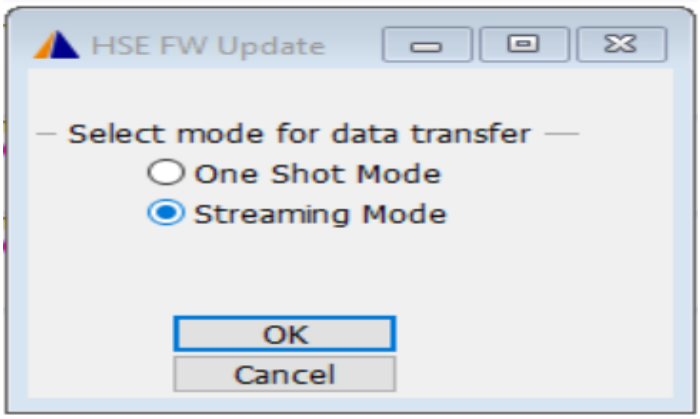


Figure 84. Select “Streaming Mode”

- Select the streaming size “512 Bytes” and click OK.

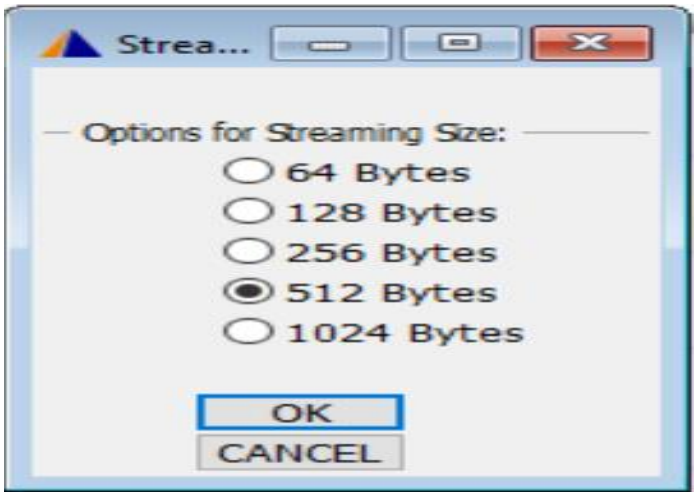


Figure 85. Firmware Streaming Size

- After that, fw_update.cmm script is called, A window pops up to flash FW pink image in code flash. Select the new HSE FW encrypted pink image that you want to update on the device.

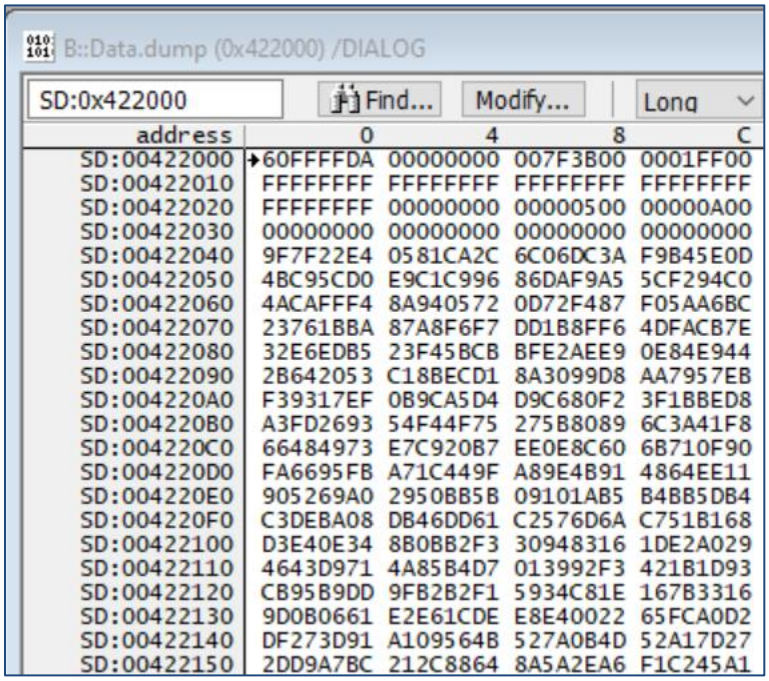


Figure 86. Firmware loaded in code flash

- Once new HSE FW is loaded, user is prompted to confirm if directly check the output or manually run the fw update as shown in below figure.

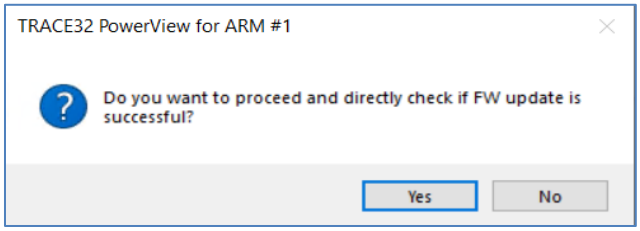


Figure 87. FW update process

- If firmware update is successful, below message pops-up.

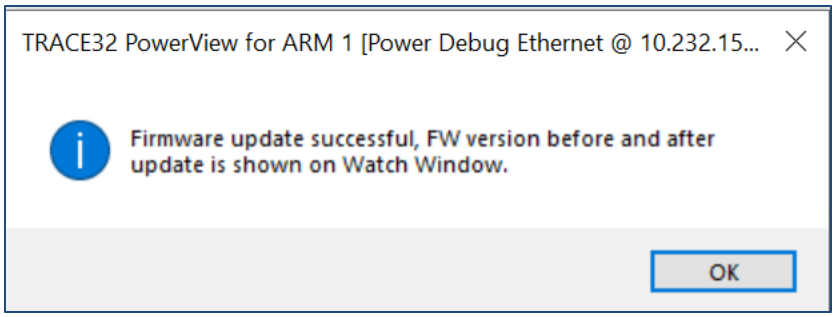


Figure 88. Firmware update success

- Demo app provisions user to verify the previous HSE FW version number and current FW installed status by displaying the same on the screen as shown below:

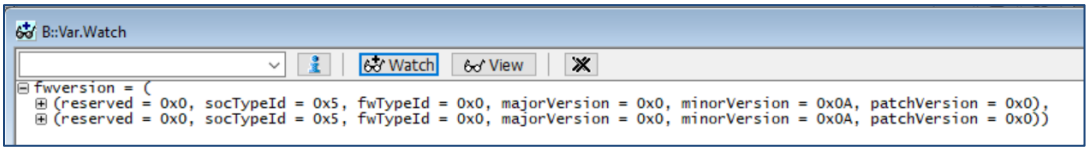


Figure 89. Firmware version before and after update

6.4.3. FULL MEM to AB SWAP HSE FW Update

Illustrating the change from FULL_MEM to AB_SWAP configuration:

Figure 90. FULL MEM to AB SWAP HSE FW Update

Steps:

To update AB Swap HSE FW through demo APP, User must follow below steps,

- Click on “User” Menu.

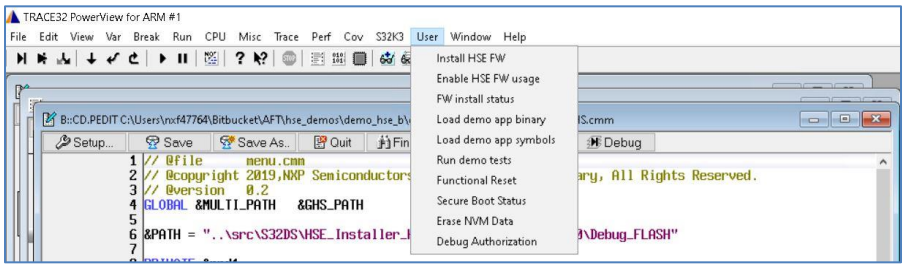


Figure 91. ‘User’ Menu

- Select “Run Demo tests”.

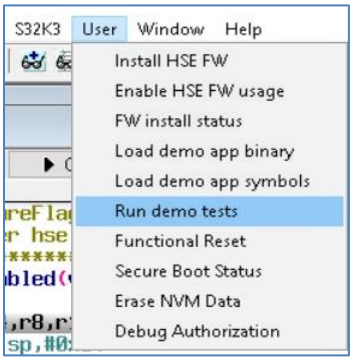


Figure 92. Select “Run Demo Tests”

- Select “HSE FW Update.”

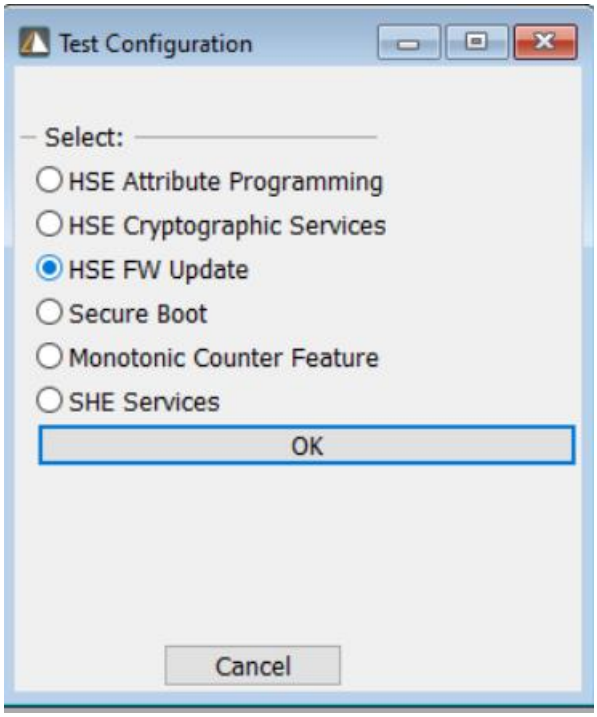


Figure 93. Select "HSE FW Update"

- Select "Update HSE FW". Click OK.

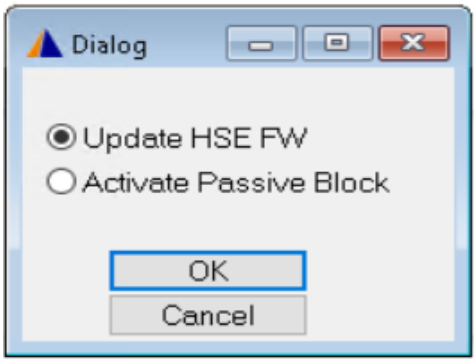
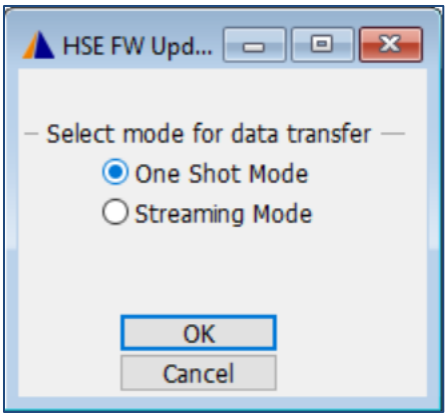


Figure 94. Proceed for Installation

- Select "One Shot Mode". Click OK.



- Browse for encrypted HSE FW pink image.
- After image is selected Dialog is open which confirms that user has selected AB Swap HSE FW for update.

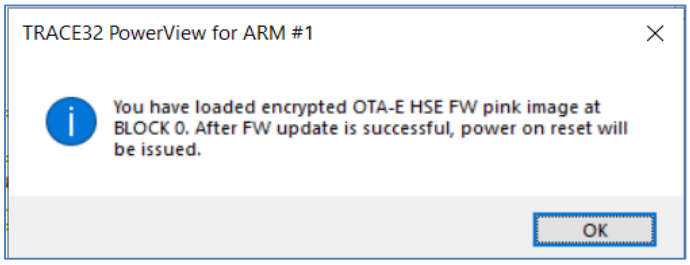


Figure 95. Dialog Confirmation

- If the user wants to directly check the output, then user should click yes else No as shown below.

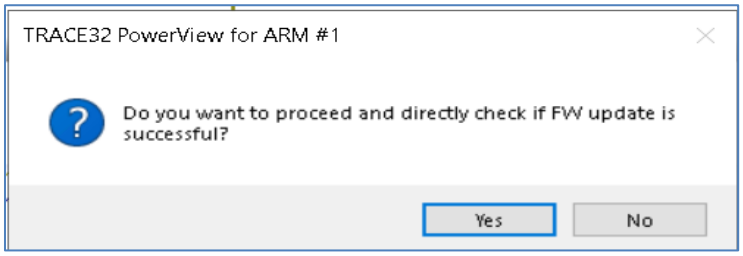


Figure 96. FW update output

- After update is successful, dialog opens with successful message. After power on reset, the FW version is updated as shown below.

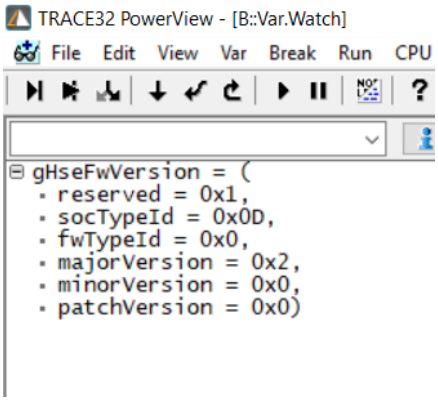


Figure 97. Watch Window showing Updated HSE FW version.

- User can also verify the FW installed Status in the Menu option.

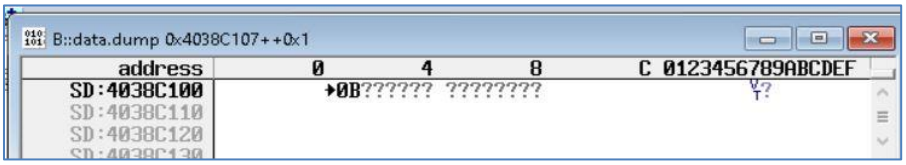


Figure 98 AB Swap HSE FW installed Status Code

6.4.4. AB SWAP to AB SWAP FW update

The steps for HSE Firmware update in AB_SWAP configuration are same as above.

6.4.5. Activate Passive Block

Illustrating the HSE Firmware update in AB_SWAP configuration:

Figure 99. AB SWAP to AB SWAP HSE FW Update

Below steps explains activation of Passive Partition with the help of demo APP.

- Follow same steps from Step 1 to Step 4 as given in “Full Mem to AB Swap Update”.
- Select “Activate Passive Block” as shown in below snapshot and click OK.

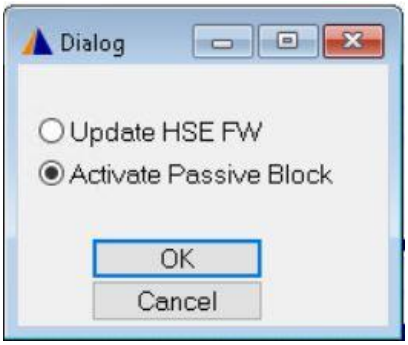


Figure 100. Firmware Update Dialog

- If user wants to directly check the output of activate passive partition request, then user should click Yes else No

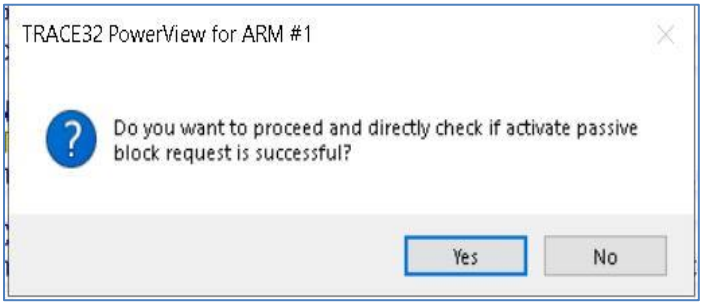


Figure 101. Dialog Confirmation

- After Passive Partition is activated confirmation dialog appears and functional reset is issued after this.

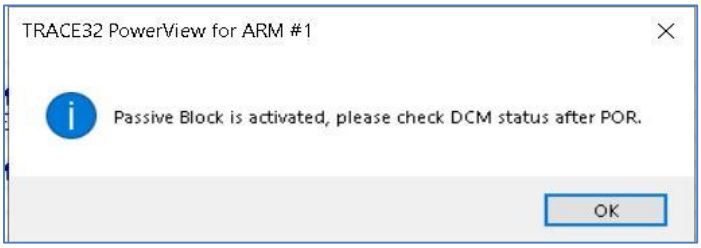


Figure 102. Dialog Confirmation

- Demo App demonstrates the flow by copying the content of active partition to passive partition with same configurations.

- After functional reset, passive partition become active partition while active partition become passive partition.
- User can also verify that Demo App is copied in Passive partition before switching. Post switching HSE FW and Demo App are executed from new active partition.

6.5. Secure Boot Demo

6.5.1. Description

To elevate the security level of the device, all active assets in flash memory must be authenticated to perform secure boot, ensuring therefore the integrity of images that controls platform's resources. For application images, three mechanisms are available for configuring the secure boot, also shown in figure 104.

- Basic Secure Boot,
- Advanced Secure Boot,
- SHE based Advanced Secure Boot

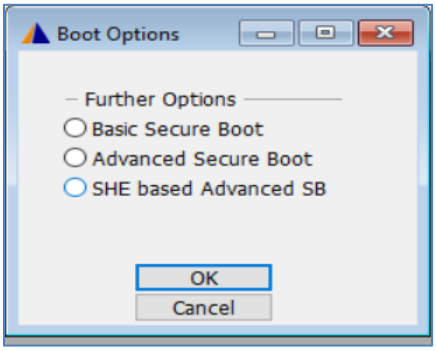


Figure 103. Secure boot options

Note: Please note that secure boot application image (application header is part of the app) is loaded at installation time only or while loading demo app binaries

Following are the addresses where secure boot app image is loaded:

- secure_boot_app - (includes application header of 64 bytes at offset 0) – 0x00451FC0

6.5.2. Basic Secure Boot (BSB)

In this case, the HSE FW enables only one application core at a time. It extracts the application header address from IVT programmed at one of the IVT locations and then authenticate application header and code using AES-GMAC algorithm and ADK/P SHA256 hash key. Application header length is of 64 bytes and application code address starts from application header start address + application header length. To enable secure boot, the code also update BOOT_SEQ to 1 value.

After issuing functional reset, the HSE FW authenticates the application and once authentication is successful that application core is enabled and booted securely.

Note: The base secure boot can only be enabled when SMR and CR table is not installed and BOOT_SEQ bit in Boot configuration word (BCW) in UTEST area is set to 1. Set BOOT_SEQ to 0 for non-secure boot.

There are some pre-requisites that needs to be satisfied before initiating basic secure boot:

- ADK/P value should be programmed by host application.

In demo code, following is implemented (enableADKPm bit is disabled)

- Calculates GMAC of application of 16 bytes using AES-GMAC algorithm in RAM. Before GMAC calculation:

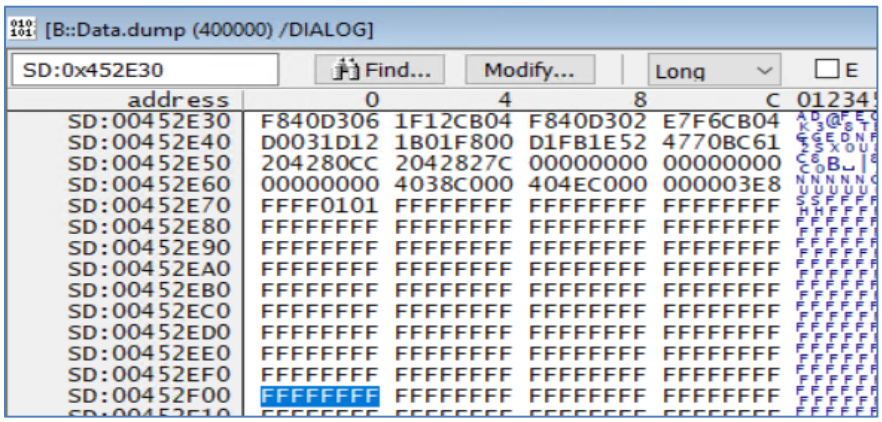


Figure 104. Secure boot app end address without GMAC

- Copy back this generated GMAC from RAM to secure boot app address to append it to the end of the application. After GMAC calculation:

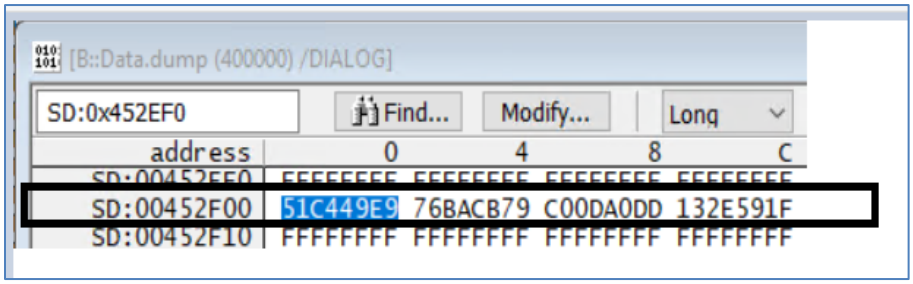


Figure 105. Secure boot app end address with GMAC

- Send verify request to HSE to validate the GMAC calculated.
- Replace non-secure boot IVT with secure boot IVT required for BSB at 0x00400000 location in which BOOT_SEQ is set to 1.

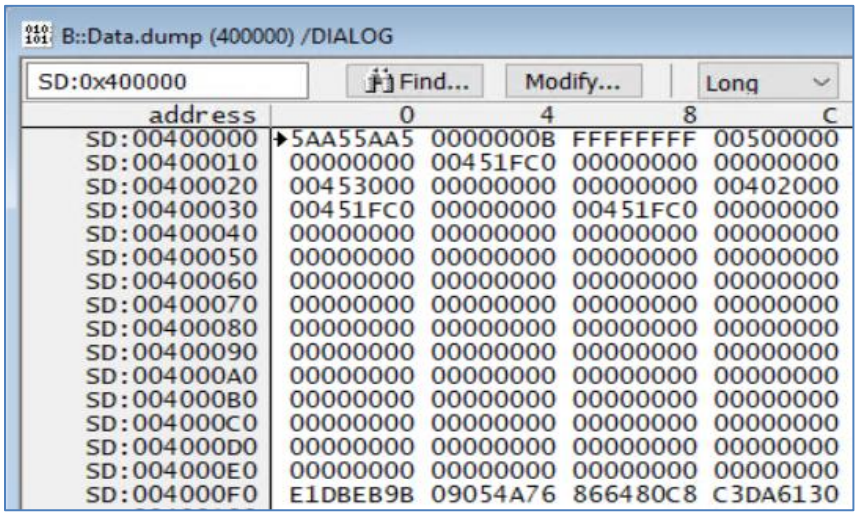


Figure 106. IVT for BSB flashed at BLOCK 0

- Once above configuration is done, basic secure boot configuration is said to be completed.

To run this example feature, follow below steps:

- Select “Secure Boot” option from Figure 37 click ok.
- Further options of Secure boot pop-ups as shown in Figure 103, select Basic Secure boot and click ok.
- In case, ADKP is not programmed, the user is warned about it that ADKP is not programmed, and one should program ADKP first to run basic secure boot.
- If ADKP is programmed, then user is asked to either continue and directly check if secure boot was configured or wants to debug the code.
- If user selects Yes to continue, then after base secure boot configuration which includes generating GMAC and enabling secure boot, following message pops-up and functional reset is issued after that.

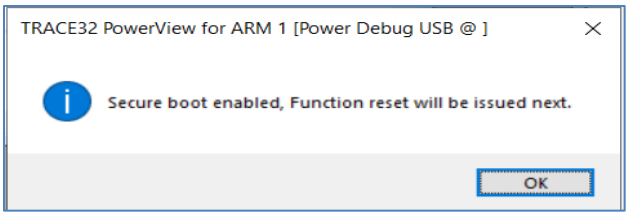


Figure 107. Secure boot configuration done

- After functional reset is issued, select “Secure boot status” option as give in Figure 27. This checks if secure boot was successful by loading secure boot app symbols and checking the secure boot status in the code as shown below:

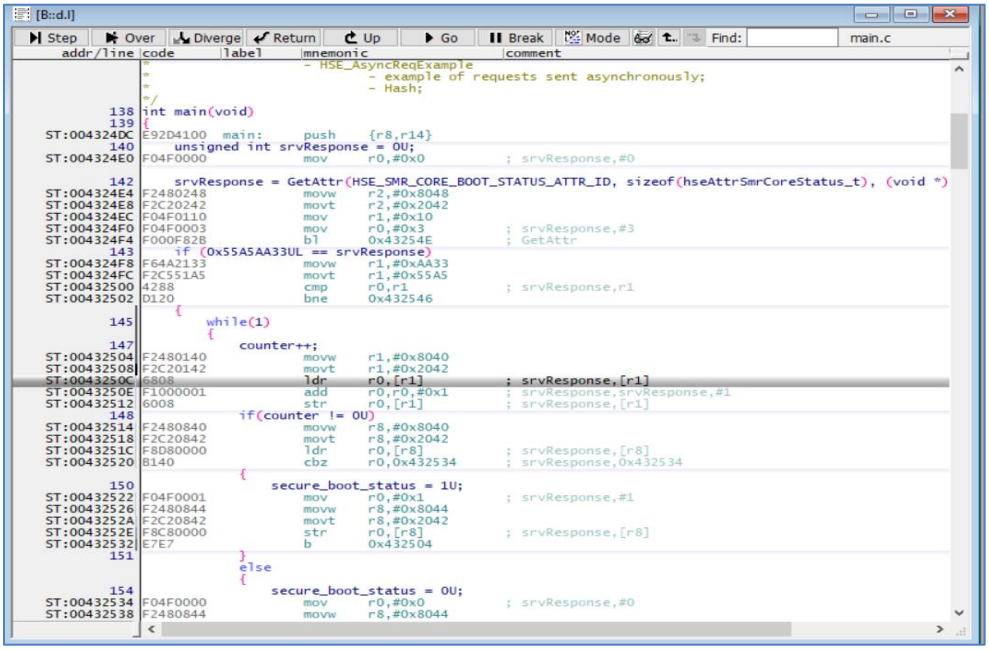


Figure 108. Secure boot app code

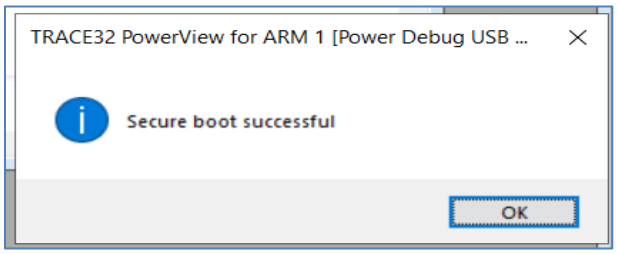


Figure 109. Secure boot status message

- After user has verified the secure boot status and wants to re-run the demo tests, first user needs to load demo application image by selecting “Load demo app binary” option from Figure 17 and click “ok”.
- Once all binaries are loaded, functional reset is issued and then user can re-run any test case.

6.5.3. Advanced Secure Boot (ASB)

In advanced secure booting, the HSE FW can boot multiple application cores which uses Secure Memory Regions (SMR) entries that are linked with Core Reset (CR) entries configuration that together define application cores behavior. These configurations are done via HSE FW services and are stored in internal data flash memory.

The ASB is initiated by HSE FW when CR table entries are present, or the entries are more than zero. The HSE FW then parse SMR/CR tables.

In demo app code, following is implemented for Advanced Secure Boot:

- Install and verify SMR#0 to SMR#4 and CR0 with following options for multiple crypto algorithms in configurable options such as:
    a. AES CMAC
    b. AES GMAC
    c. HMAC
    d. RSA-PKCS
    e. ECC
- This demo application can be configured in any cipher combination mentioned above.
- Secure boot application is booted with core 0.
- Enable secure boot by setting BOOT_SEQ=1 in IVT in boot configuration word.

Pre-requisites before ASB can be executed are as follows:

- The host should be granted with Super User (SU) rights.
- LC should be CUST_DEL and subsequent SMRs and CRs should be empty.

To run this example feature:

- Go to User menu and select “Run demo tests” option, click ok and then select “Secure Boot” option and click ok.
- Again select “Advanced Secure Boot” option as given in Figure 103 and click ok.
- Select any cipher options among the mentioned options and click ok.

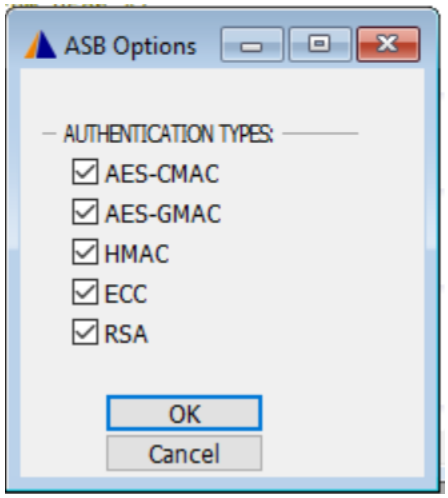


Figure 110. ASB Cipher Options

- The Ciphers are mapped to SMR Entries as follows
  a. SMR#0 → AES-CMAC
  b. SMR#1 → AES-GMAC
  c. SMR#2 → HMAC
  d. SMR#3 → ECC
  e. SMR#4 → RSA-PKCS

- User is prompted for option to either proceed and directly check the output or debug the code.

- If user selects Yes to proceed then user should wait for advanced secure boot configuration to be completed.
- Once completed, user is notified of the same with message as shown in Figure 112.
- Issue functional reset by opting for “Functional Reset” option from User menu.
- After issuing reset, to check if advanced secure boot was successful, the user should opt for “Secure Boot status” option from User menu and click ok.
- This loads secure boot app symbols and check from which core and address was Advanced secure boot done as shown in Figure 110.

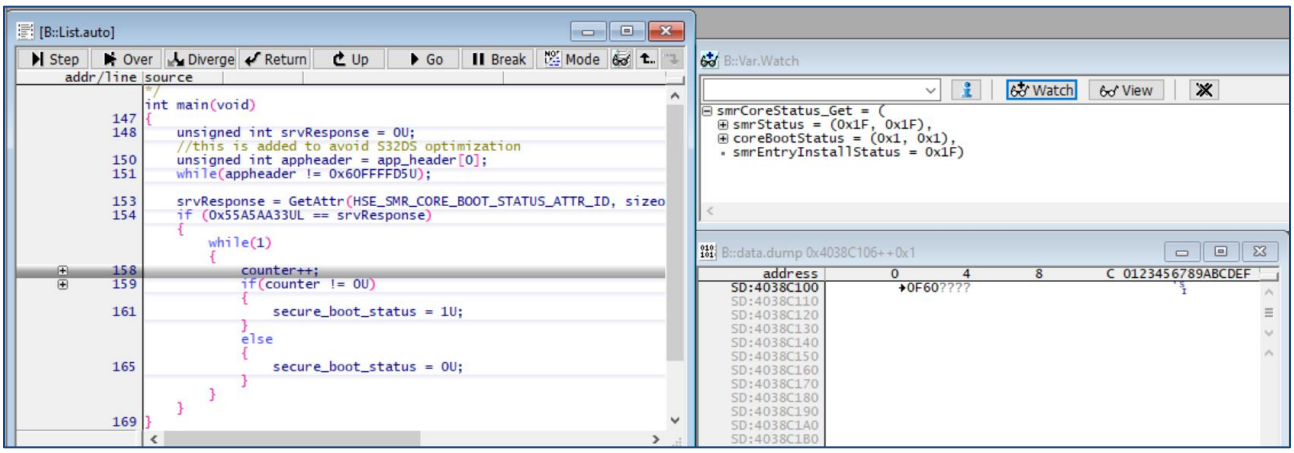


Figure 111. Advanced Secure Boot status

- After user has verified the secure boot status and wants to re-run the demo tests, first user needs to load demo application image by selecting “Load demo app binary” option from Figure 24 and click ok.
- Once all binaries are loaded, functional reset is issued and then user can re-run any test case.

6.5.4. SHE based Secure boot

HSE FW does SHE based boot operation using SMR and CR tables. Only SMR #0 should be used to implement SHE secure boot.

In demo app code, SHE secure boot is implemented in following sequence:

- Format SHE NVM and RAM key catalog.

- Load BOOT_MAC key in SHE_NVM_BOOT_MAC key target handle.
- Install SMR#0 with SHE_NVM_BOOT_MAC key handle and HSE_KEY_TYPE_SHE key type and AES CMAC as crypto algorithm.
- Install CR#0.
- Enable secure boot by setting BOOT_SEQ=1 in IVT in boot configuration word.

To run this example feature:

- After NVM data is erased successfully, again go to User menu and select “Run demo tests” option, click ok and then select “Secure Boot” option and click ok.
- Again select “SHE Secure Advanced SB” option as given in Figure 103 and click ok.
- User is prompted for option to either proceed and directly check the output or debug the code.
- If user selects Yes to proceed then user should wait for SHE secure boot configuration to be completed.
- Once completed, user is notified of the same with message as shown in Figure 101.
- Issue functional reset by opting for “Functional Reset” option from User menu.
- After issuing reset, to check if SHE secure boot was successful, the user should opt for “Secure Boot status” option from User menu and click ok.
- This loads secure boot app symbols and check from which core and address was SHE secure boot done and also checks SHE_SECURE_BOOT_FINISHED,INIT and OK bits to confirm that secure boot was successful as shown in below figure.

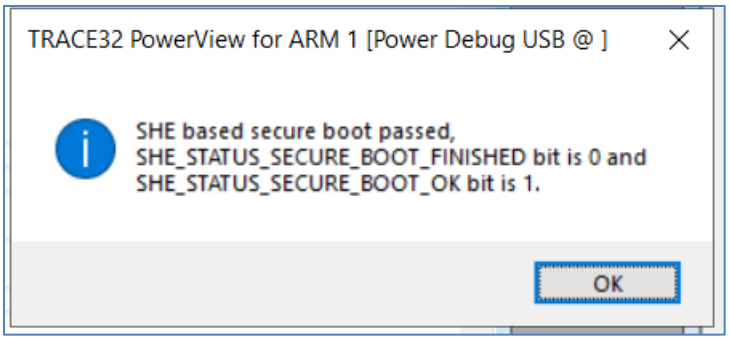


Figure 112. SHE secure boot status

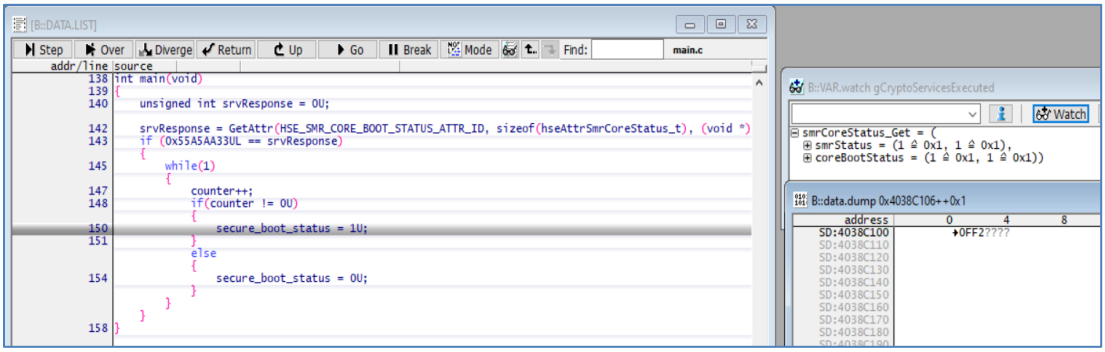


Figure 113. secure boot app code

- After user has verified the secure boot status and wants to re-run the demo tests, first user needs to load demo application image by selecting “Load demo app binary” option from Figure 17 and click ok.
- Once all binaries are loaded, functional reset is issued and then user can re-run any test case.

6.6. Monotonic Counter

HSE supports 16 monotonic counters, each counter is 64 bits wide and associated a counter index from 0 to 15. By default, all the counters are disabled. The monotonic counter (MC) consists of 2 separate bitfields: Rollover Protection (RP) + Volatile Counter (VC).

HSE stores the monotonic counter in data flash each time the Rollover Protection (RP) is updated. The objective is to reduce the rate at which NVM programming operations occur. If the monotonic counter is already configured and configuration is called again, HSE re-configures the counter with new RP value and reset it to 0.

NOTE:

Rollover protection minimum allowed value = 32

Rollover protection maximum allowed value = 64

Rollover protection value must be multiple of 8

If Rollover Protection Bit Size is 64bits, HSE stores the MC in flash each time is updated.

HSE erases a flash sector after 31 Rollover Protection updates in data flash.

The flash erases cycles are limited. The application should configure each monotonic counter depending on the MC update rate and the number of enabled counters. The monotonic counter configuration is stored in data flash each time configure monotonic counter service (hseConfigSecCounterSrv_t) is called.

Example:

Let's consider the Rollover protection = 40 and MC = 0x0000000000000001.

This means Rollover Protection (40bits) + Volatile Counter (24bits).

The monotonic counter (MC) is stored in flash if the incremental value is >= 0xFFFFFF. Otherwise, the counter is updated but not stored.

MC = 0x0000000000000001+0xFFFFFF = 0x0000000001000000 (RP was changed)

6.6.1. Configure Monotonic Counter

The purpose of this service is to enable the monotonic counter and configure the Rollover Protection bitfield size. If the monotonic counter is already configured and configuration is called again, HSE re-configures the counter with the new Rollover Protection (RP) and reset it to 0.

To configure counter in Demo Application, User need to follow below steps.

- Select “Run demo tests” option and select Monotonic Counter Feature and click ok.
- Select ‘Monotonic Counter Feature’ and then press OK.

Figure 114. Monotonic Counter Feature

- Monotonic Counter(MC) Operation window is opened.

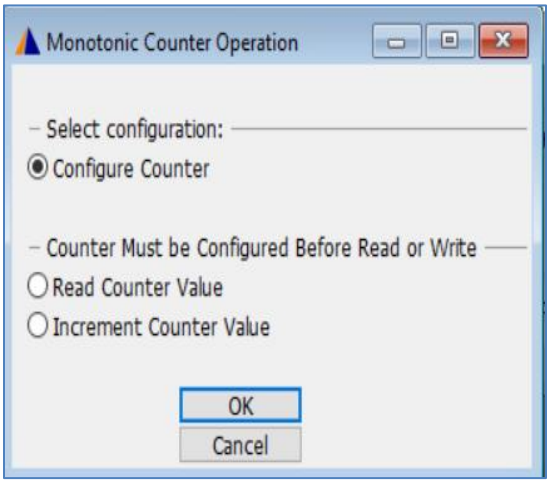


Figure 115. Monotonic Counter configures

- Select “Configure Counter” and then click OK.
- Monotonic Counter Selection window is open.
- Select the counter number which you want to configure and then Press OK.

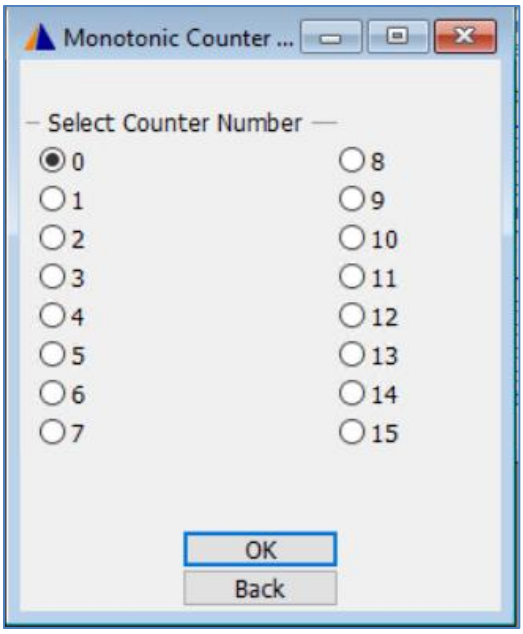


Figure 116. Monotonic Counter Configure option

- Next window with Roll Over Protection configuration is open.
- User need to double click on “MonotonicCntRPBitSize” and then enter value between 32 to 64 (and multiple of 8).
- Press Enter. Entered value is displayed in watch window and command line.

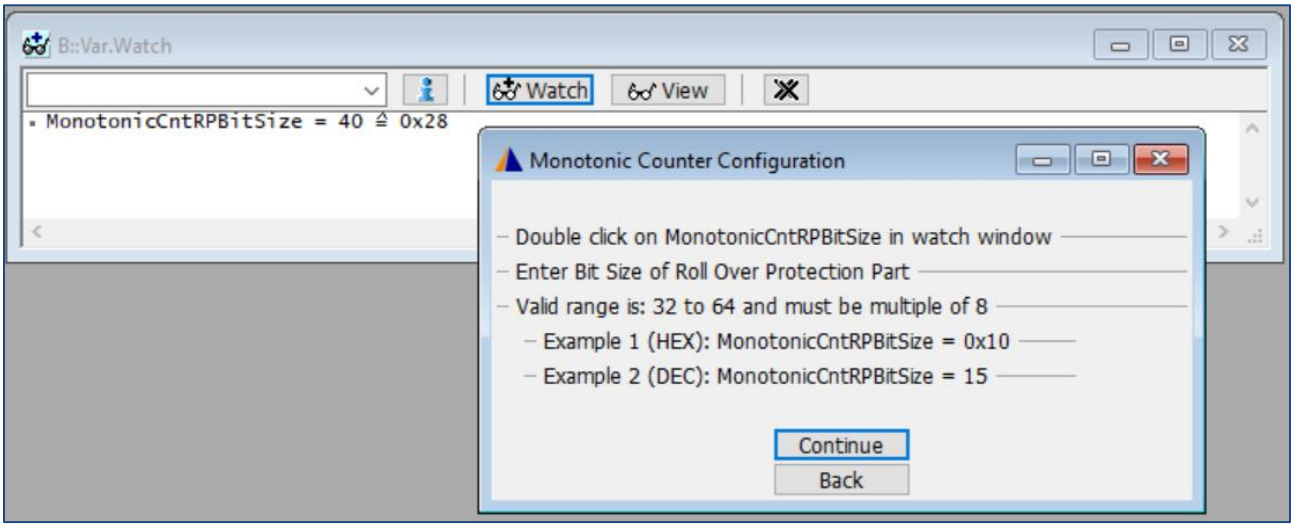


Figure 117. Update Monotonic Counter RPB in watch window

- For Example, value 64 is entered which is being displayed in watch window.

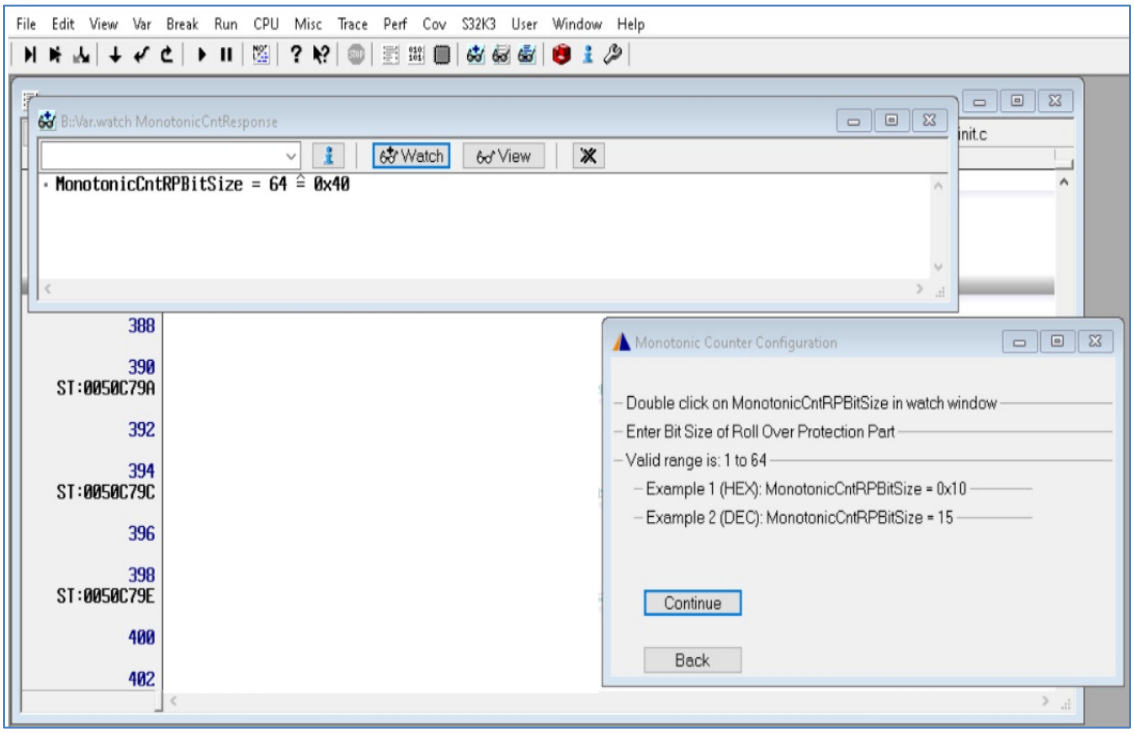


Figure 118. Continue after RPB update

- Click on Continue.
- After this Counter is configured. Main screen with watch window shows the response code.
- Symbol “MonotonicCntResponse” holds response code. For Success, Response Code is : 0x55A5AA33

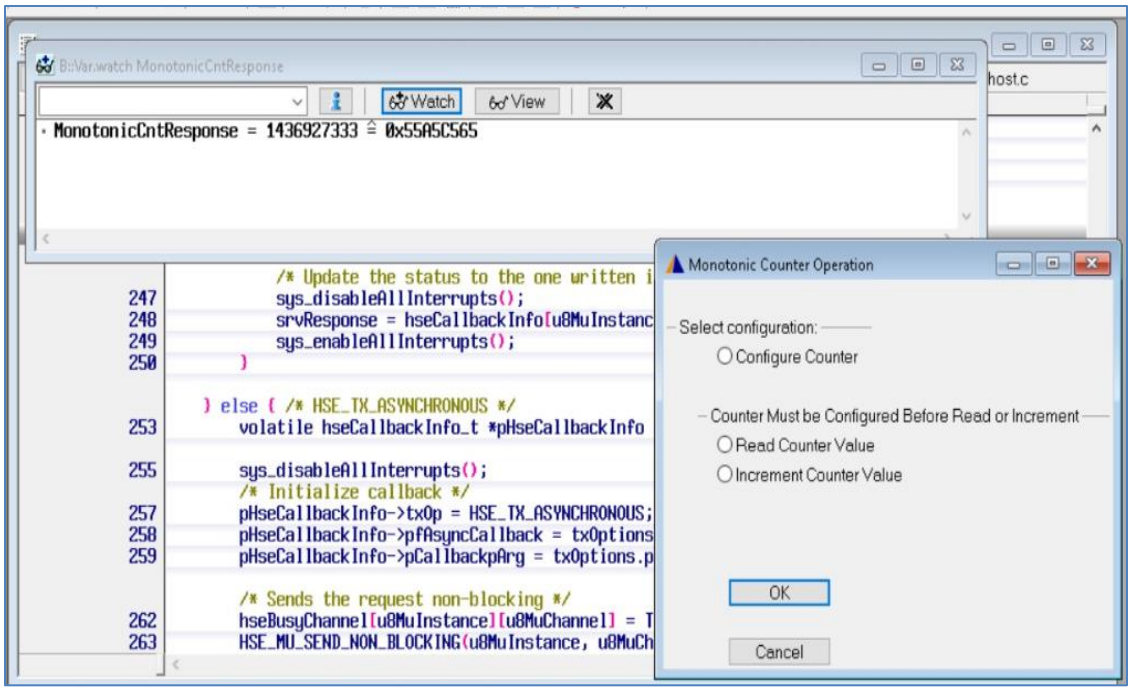


Figure 119. RPB response

6.6.2. Increment Monotonic Counter Value

Below are the steps to increment monotonic counter value.

- Select “Increment Counter Value” and then Press OK

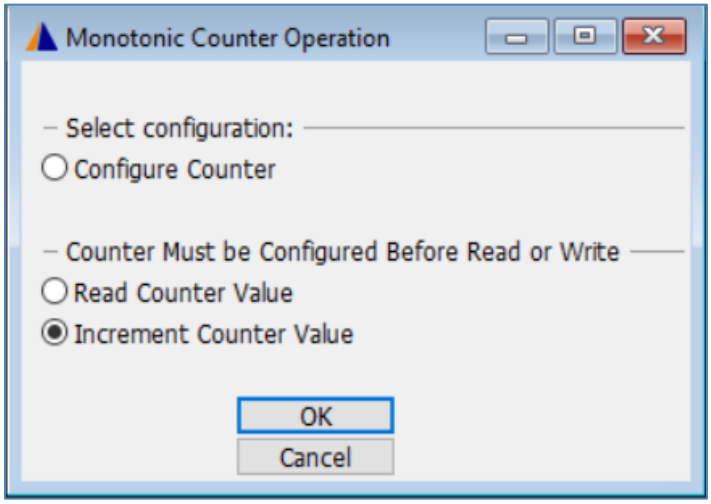


Figure 120. Monotonic Counter Increment Option

- Select Counter Number and then Press OK.

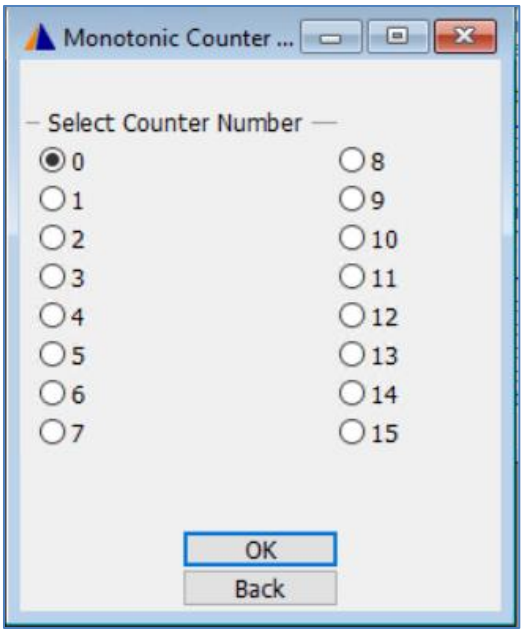


Figure 121. Monotonic Counter number option

- Next window is open where user need to enter value by which counter can be increment.
- Double click on “MonotonicCntIncrementValue” and then enter the desired value.
- Press Enter. Entered value is displayed in watch window.

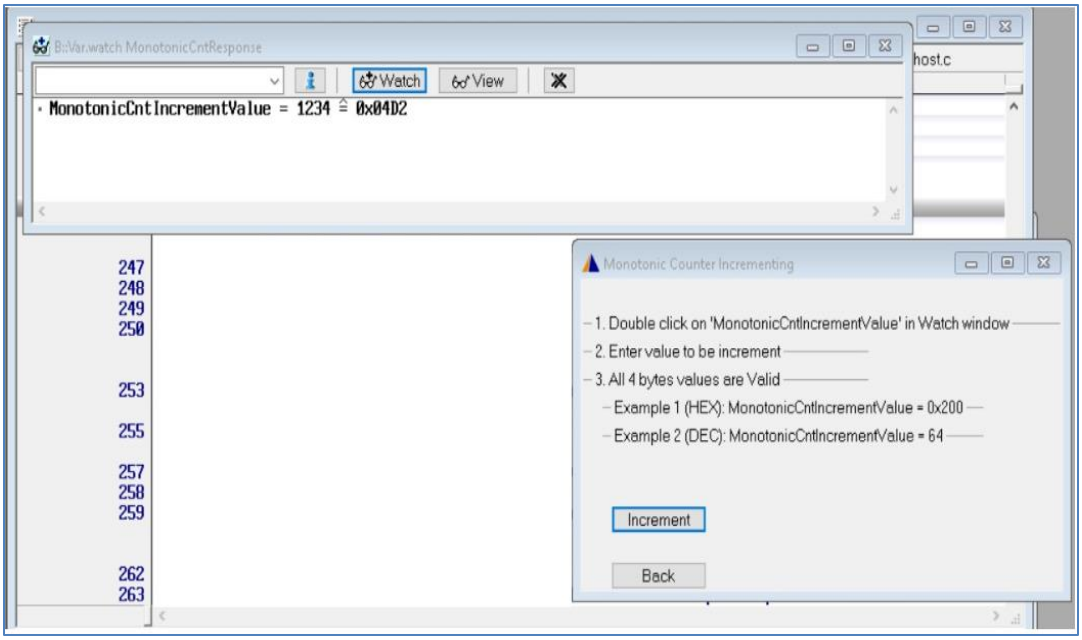


Figure 122. Update Monotonic Counter RPB value

- Click on Increment.
- Value should be updated. Main screen with watch window shows the response code.
- Symbol “MonotonicCntResponse” holds response code. For Success, Response Code is : 0x55A5AA33

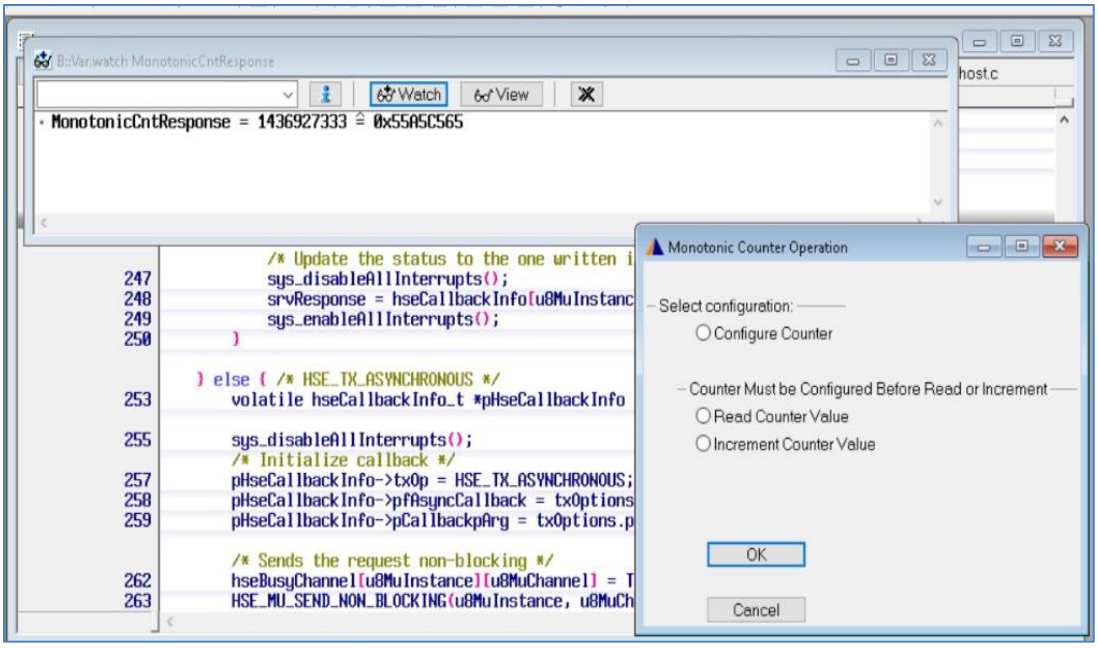


Figure 123. Monotonic Counter RPB response

6.6.3. Read Monotonic Counter Value

- To read counter follow the below steps :
- Select Read counter Value and click ok

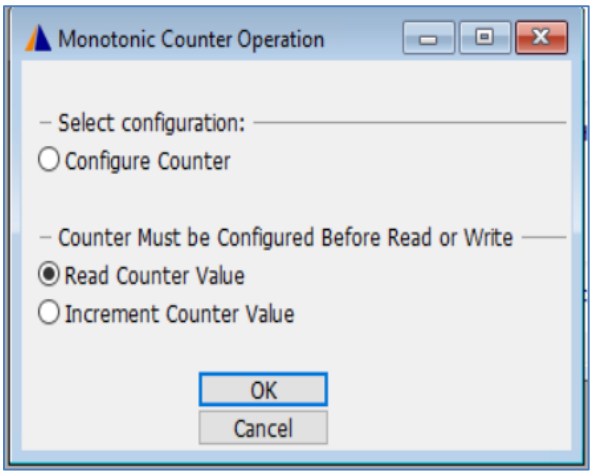


Figure 124. Monotonic Counter Read Option

- Select Counter number which you want to Read. And then Press OK.

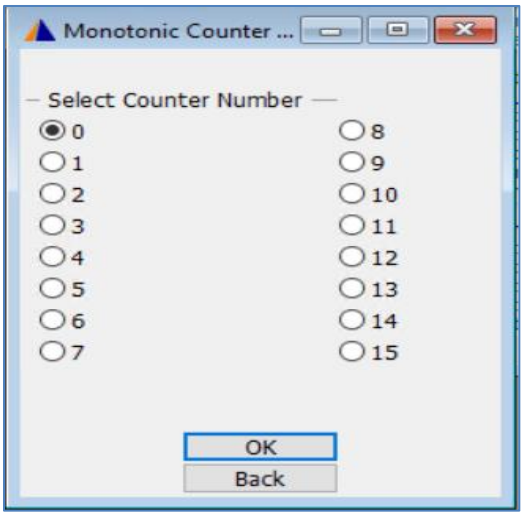


Figure 125. Monotonic Counter number option

- Counter value is displayed in watch window and in command line with Response code.
- Symbol “MonotonicCntResponse” holds response code. For Success, Response Code is : 0x55A5AA33
- Symbol “MonotonicCntReadValue” holds counter value.

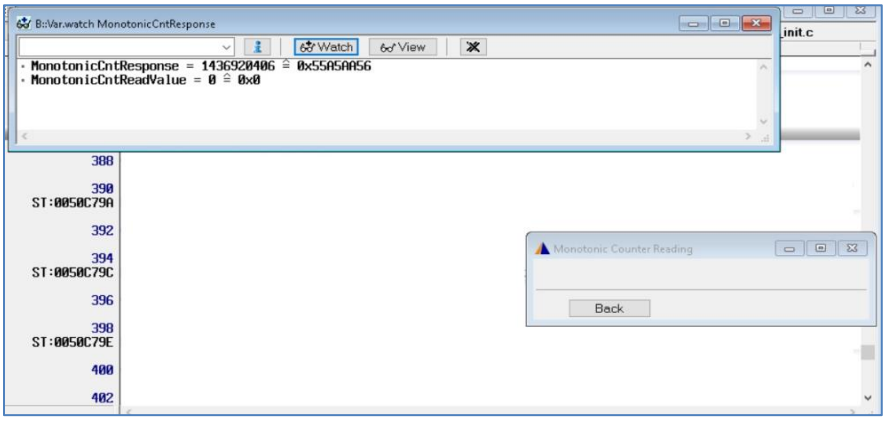


Figure 126. Monotonic Counter Read response

7. Debug Authorization

This feature enables application debug authorization when LC is advanced to either IN_FIELD or OEM_PROD. There is total four ways in which debug authorization can happen.

When ADKP_MASTER bit (also known as enableADKm bit) (the method to provision ADKP in secure NVM) is set to 0, i.e. the default configuration, the input value is ADKP (mode is either password mode or challenge-response mode) and is written “as is” in secure NVM

When ADKP_MASTER bit is set to 1, the input value is considered as a master debug key and is diversified with the device’s UID before being programed i.e., the value for password mode or challenge-response is calculated along the unique identifier (UID) of the device hence the value always is unique for every device. For more details, Refer to HSE Firmware Reference Manual.

Note: Before running this example feature, the user should not program ADKP key as if ADKP key is already programmed, then CUST_DEL security policy cannot be exceeded.

To run this feature, the user should have extended CUST_DEL security policy and programmed ADKP key also.

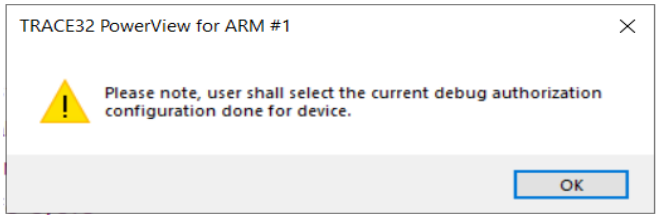


Figure 127. Warning

If device LC is advanced from CUST_DEL and debugger access is disabled, then the user can use this feature to execute debug authorization to open access of debugger to code.

Select “Debug authorization” option from User menu. This prompts user to select the configuration that is done for debug auth mode and enableADKm bit as shown below:

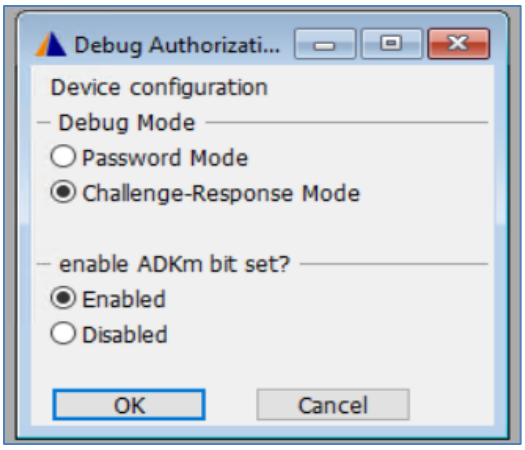


Figure 128. Debug Authorization configuration input from user

- The ADKP key input for debug authorization is taken from adkp_key_input.txt itself.
- Once the user selects the configuration of debug mode and enable ADKm bit, the user should wait for debugger to open access.
- If debug mode password and ADKm bit disabled is selected, then following steps are followed for debug authorization:
  - Try to access the HSE FW code via debugger.
  - The debugger runs the authorization process via JTAG.
  - It asks for loading 16-byte value which is given input from text document and which is compared with 16byte ADK/P value programmed in UTEST.
  - If the value matches, the debugger is granted the access of the code else reset is issued and again debug authorization is tried.

7.1. Configuration

To run debug authorization, the user must do following steps to configure trace32 for it. Following software should be installed:

- Microsoft Visual C++ Redistributable Package
- Python 3.6 (32bit) with following packages:
- Pycryptodome: pip install pycryptodome

- enum34: pip install enum34

Note: If the user has "crypto" lib already installed in "*C:\Python36\Lib\site-packages*" path, then from command line, user first should uninstall this library and then delete the "crypto" folder from *"C:\Python36\Lib\site-packages"* path and then again from command line, execute "pip install cipher" command. This install "Crypto" libs again.

Following points must be taken care by user:

- Edit *C:\T32\config.t32* and check following points, if any missing please add it: RCL, PACKLEN and PORT are set as given below:

RCL=NETASSIST

PACKLEN=1024

PORT=20000

- C:\T32\bin\windows64\t32marm.exe to be used for debug authorization as above settings are required.
- If the user machine has T32 32-bit installed, then user should edit t32_wrapper_func.py script in scripts folder and update command from

dll_name = *"t32api64.dll"* to

dll_name = *"t32api.dll"*

7.2. Commands supported

Following configuration are supported in debug authorization in this demo app:

- When user selects debug mode as "Password" and enableADKm bit as "Disabled", the command that is called is debug_App_ADKP -PASS -MSTR_DEB_KEY.
- When user selects debug mode as "Challenge Response" and enableADKm bit as "Disabled", the command that is called is debug_App_ADKP -CHL_RSP -MSTR_DEB_KEY.

- When user selects debug mode as “Password” and enableADKm bit as “Enabled”, the command that is called is debug_App_ADKP -PASS -DER_DEB_KEY, and
- When user selects debug mode as “Challenge Response” and enableAdKm bit as “Enabled” the command that is called is debug_App_ADKP -CHL_RSP -DER_DEB_KEY

Note: User should load symbols manually once debug authorization is successful.

7.3. Limitations

Following are the limitations with debug authorization of the demo app:

- This tool is not supported for HSE Firmware Feature Flag Disabled.
- This tool is not supported for opening HSE Debug.
- This tool is only supported for JDC(JTAG) Based Debug Authorization and not for SD_DAP Based Debug Authorization.

7.4. Commands supported

Following configuration are supported in debug authorization in this demo app:

- When user selects debug mode as “Password”, the command that is called is :

py -3 debug_App_ADKP -PASS -MSTR_DEB_KEY.

- When user selects debug mode as “Challenge Response”, the command that is called is :

py -3 debug_App_ADKP -CHL_RSP -MSTR_DEB_KEY.

Note: User should load symbols manually once debug authorization is successful.

8. Erase NVM Data

This service is used if the user wants to erase NVM data and wants to re-format NVM keys. To execute this feature, the user should select *“Erase NVM data”* option from user menu as shown in Figure 17 and click ok. This prompts user to either proceed and directly check the output if the

data was erased or debug the code. If user selects, yes to proceed, then user should wait for output. In case of success, following message pops up along with the status of test case status in watch window as shown below:

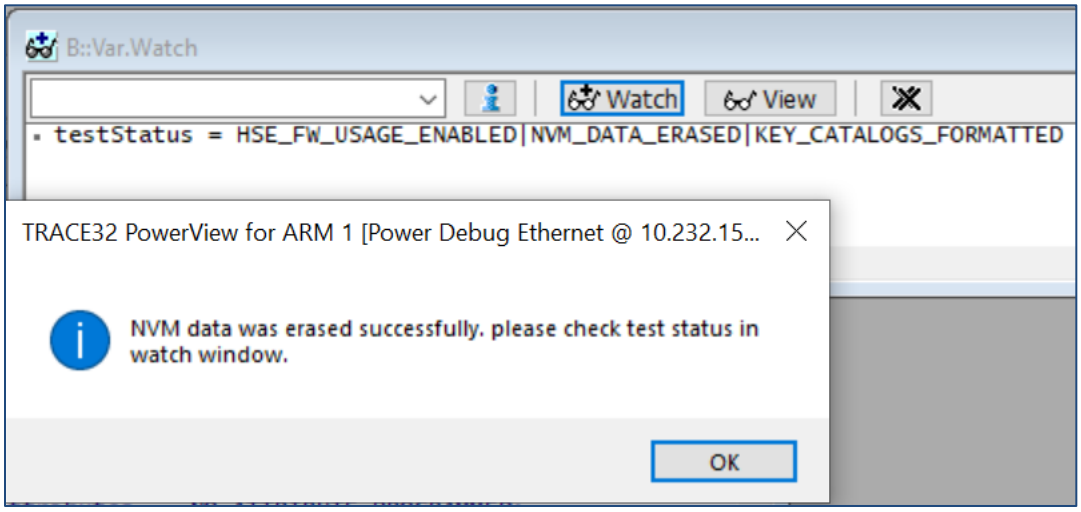


Figure 129. Erase NVM data status

9. SHE Command Example

Description

This examples maps the SHE commands mentioned in SHE Functional Specification to HSE. It demonstrates the use of all these commands using standard test vectors as per SHE Verification Specification.

Table 5. Supported SHE commands

SHE Command	Mapped Command
CMD_ENC_ECB	she_cmd_enc_ecb
CMD_ENC_CBC	she_cmd_dec_ecb
CMD_DEC_ECB	she_cmd_enc_cbc
CMD_DEC_CBC	she_cmd_dec_cbc
CMD_GENERATE_MAC	cmd_generate_mac
CMD_VERIFY_MAC	cmd_verify_mac
CMD_LOAD_KEY	cmd_load_key
CMD_LOAD_PLAIN_KEY	cmd_load_plain_key
CMD_EXPORT_RAM_KEY	cmd_export_ram_key
CMD_EXTEND_SEED	cmd_extend_seed
CMD_RND	cmd_rnd
CMD_GET_ID	cmd_get_id
CMD_DEBUG	cmd_debug_chal_auth

The tests and test steps are briefly explained later in this chapter.

Files which cover the above features are as follows:

- hse_she_command_main.c : This file contains the code for implementing the SHE Commands.
- hse_memory_update_protocol : This file contains the function to implement the SHE Memory Update Protocol

- hse_host_test.c : This file contains the functions to test the commands implemented in the hse_she_command_main.c file
- hse_host_wrappers.c : This file contains the wrappers and helper functions which we are using in our application.
- hse_host_commom.c : This file contains the functions to send the request to HSE, and other utility functions.

The following services are tested for demonstration of SHE Commands.

9.1. Execute SHECommandApp

- Follow below steps to run SHE command application.
- Click on “User” and select “Load SHE command app” option from the drop-down as shown below:

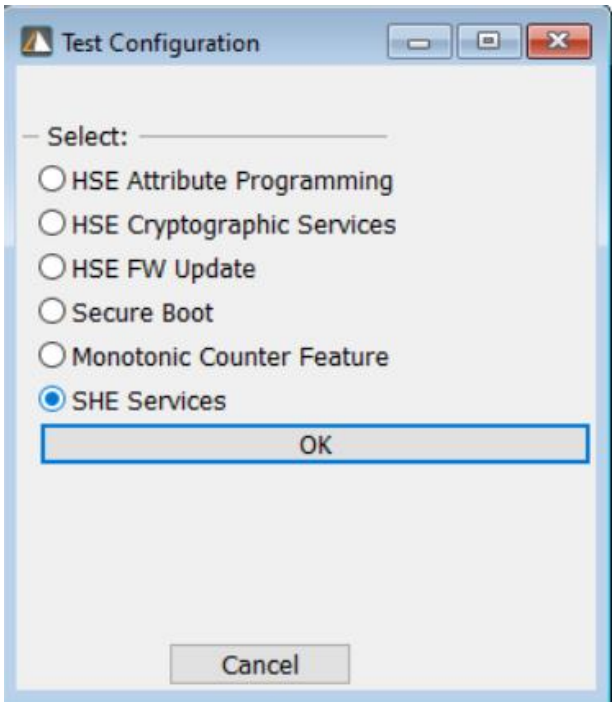


Figure 130. Select SHE command app option

- A dialog box appears, select OK.

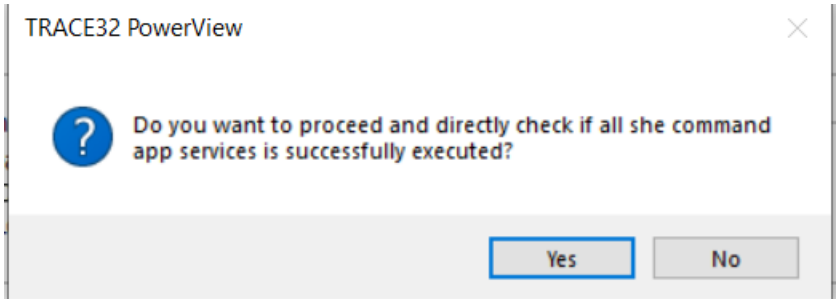


Figure 131. pop-up

- SHE Command App is in running state now, and if all the services will be executed successfully, then the following window will pop-up.

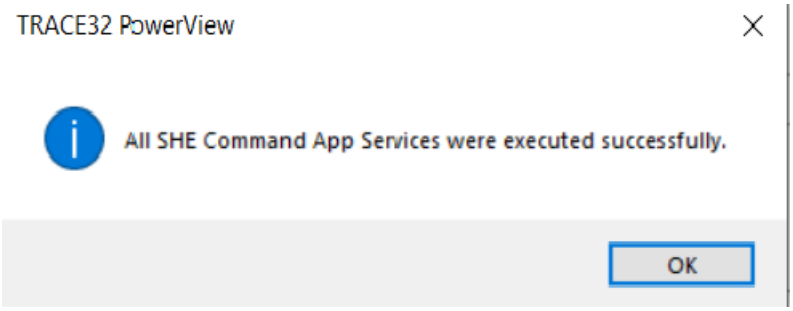


Figure 132. SHE Commands app status

- In case, any of SHE Command App service failed, then the following window will pop-up as shown in image below:

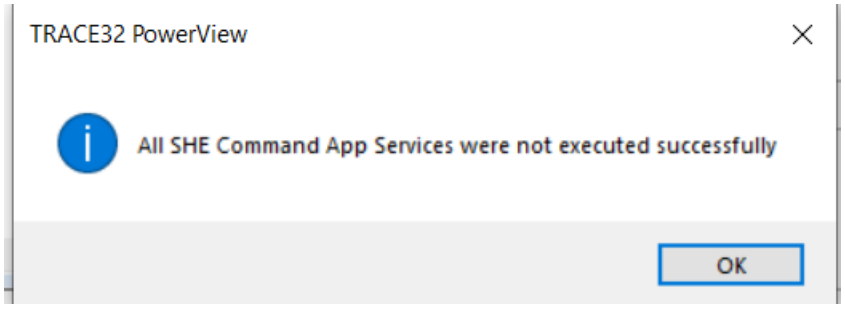


Figure 133. SHE Commands app status

- Run d.l command in command line. The code should be visible to the user as shown in figure below:

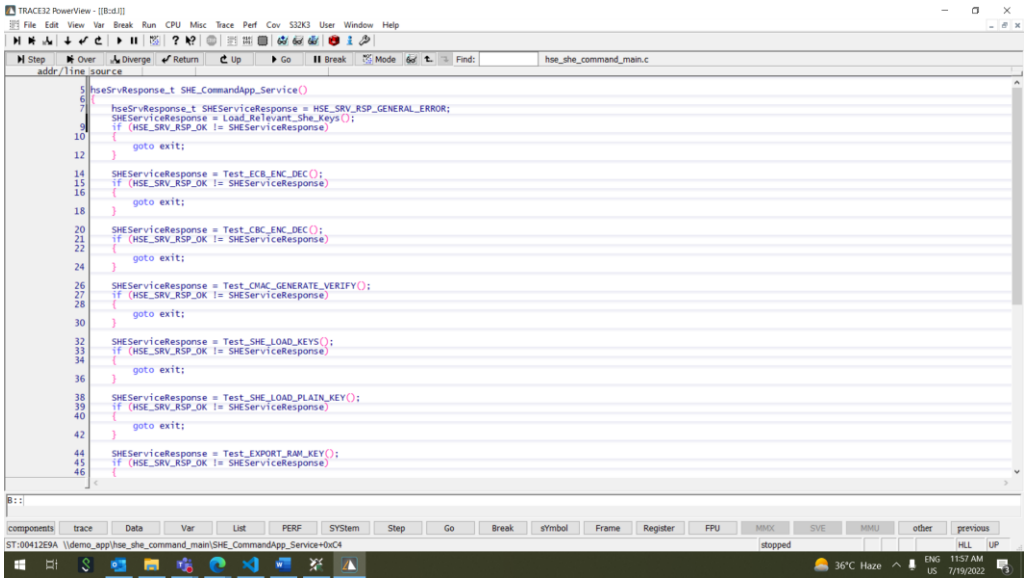


Figure 134. SHE Commands app code

The following services are tested for demonstration of SHE Commands.

9.2. Load the relevant Keys

- The required keys to be used for testing AES ECB Encryption/Decryption, AES CBC Encryption/Decryption and CMAC Generation/Verification are uploaded in the first step with the respective flag values. The tests are performed once we've successfully uploaded the keys.

Load the Master ECU Key

- If the response is not successful, Erase the SHE Key Catalog. Erase operation is done after we have done the Authorization successfully.
- Send the Load Key request again for Master ECU Key

- The operation must be successful this time, if it is not successful still, then erase NVM data and repeat the above three steps.
- Load NVM_Key_1, used for testing AES-ECB Encryption/ Decryption Command
- Load NVM_Key_2, used for testing AES-CBC Encryption/ Decryption Command
- Load NVM_Key_3, used for testing CMAC Generation/Verification Commands. The KEY_USAGE_FLAG usage flag type is set for this key to enable MAC Generation and Verification

9.3. AES ECB Encryption/Decryption Test

- AES ECB Encryption is performed and verified with the given test vectors, using the NVM_KEY_1, which we've already loaded.
- AES ECB Decryption performed and verified with the given test vectors.

9.4. AES CBC Encryption/Decryption Test

- AES CBC Encryption is performed and verified with the given test vectors, using the NVM_KEY_2, which we've already loaded.
- AES CBC Decryption is performed and verified with the given test vectors.

9.5. CMAC Generation/Verification Test

- AES Fast CMAC Generation and Verification using NVM_KEY_3.

9.6. Load Key Command Test

- NVM_KEY_4 is loaded to verify this test; Master ECU Key is used for authentication. The parameters M1, M2 and M3 have been calculated during runtime.
- The successful update of the key is verified by comparing the value of M4 and M5.

9.7. Test Export RAM Key Command

- Load the RAM Key using the standard SHE Memory Update Protocol, the export RAM key command does not give successful response.

- Load the RAM Key in plain format, the Export RAM key is successful this time.

9.8. Test Extend Seed Command

- Tests the Random Number Generation Request for DRG4.

9.9. Test Random Number Request

- Tests the Random Number Generation Request for DRG3.

9.10. Test SHE Get ID Command

- In this test we verify the SHE Get ID Command, the command returns the UID , the value of sReg and a mac value.
- The value of UID is verified by calculating the MAC and comparing it against the MAC value we received from the Get UID command.

9.11. Test Debug Challenge and Authentication Commands

- Downgrade to User Rights (To avoid error if we've already got Super User rights)
- Run the Debug Challenge Command to get the challenge value.
- Derive key from MASTER_ECU_KEY with DEBUG_KEY_C for challenge response.
- K = KDF (KEYMASTER_ECU_KEY, DEBUG_KEY_C).
- Generate MAC over the challenge value using the derived key.
- Send the Debug Authentication command, with the MAC obtained in the above step.
- Test TRNG Command.
- Generate random number using DRNG4.

10. APPENDIX

10.1. Appendix 1 IVT Information

This defines IVT information that is scanned by SBAF after device is out of reset and that must be programmed at one of the following location by the application:

- 0x00400000
- 0x00500000
- 0x00600000
- 0x00700000
- 0x10000000

Table 6. IVT Image Structure

Address offset	Size (Bytes)	Name	Comments
0h	4	Image Vector Table Marker	Magic number showing the start of the IVT. Value = 0x5AA55AA5.
4h	4	Boot Configuration Word	Configuration data used by device to select various boot configuration.
8h	4	Reserved	-
Ch	4	CM7_0 APP start address	Boot address of CM7_0 APP in code flash area; must honor core VTOR register alignment restrictions; used by SBAF when BOOT_SEQ bit is 0.
10h	4	Reserved	-
14h	4	CM7_1 APP start address	Boot address of CM7_1 APP in code flash area; must honor core VTOR register alignment restrictions; used by SBAF when BOOT_SEQ bit is 0.

Address offset	Size (Bytes)	Name	Comments
18h	4	Reserved	-
1Ch	4	CM7_2 APP start address	Boot address of CM7_2 APP in code flash area; must honor core VTOR register alignment restrictions; used by SBAF when BOOT_SEQ bit is 0.
20h	4	Reserved	-
24h	4	LC configuration address	Address of Configuration word that allows User to Advance LC.
28h	4	Reserved	-
2Ch	4	NXP supplied encrypted HSE FW image address	Start address of encrypted HSE FW image supplied by NXP.
30h	4	App_StartAddress	Start Address of Application Core for secure Boot
34h	12	Reserved	-
40h	4	App_backupStartAddress	Backup Start Address of Application Core for secure Boot
44h	4	Length	Length of Recovery Application
48h	168	Reserved	
F0h	16	GMAC	The AES-GMAC of the entire IVT structure calculated using ADKP-SHA.

Table 7. IVT Boot Configuration Word

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
							X			APP_ SWT _INIT	PLL bit	BOOT_ SEQ		CM7_1_ ENABLE	CM7_0_ ENABLE

Note: For S32K3xx devices, in IVT, only pointer to LC is provided and not the actual

11. Revision History

VERSION	DATE	CHANGE DESCRIPTION
1.0	30-09-2020	S32DS project configuration, debug authorization, ivt authentication, ABSWAP and XRDC configuration added
1.1	05-10-2020	Directory name of the demo app folder updated, and Package content section updated.
1.2	30-07-2021	Design changes w.r.t EAR2 Release of HSE Firmware Initial support added for multiple K3 platforms Debug Auth tool for Python 2.7 is migrated to Python 3.6+ SMR Installation supports multiple cipher schemes (ASB) FCA features like Fast CMAC with Counter and Burmester Desmedt added HSE Firmware Update supports Streaming Operation of multiple sizes Unified scripts for GHS and S32DS.
1.3	16-08-2021	Support added for S32K312. Changes in MU addresses added for S32K312. JTAG changed to DAP.

VERSION	DATE	CHANGE DESCRIPTION
1.4	6-09-2021	Demo app parent directory name updated.
1.5	8-10-2021	Demo app parent directory name updated.
1.6	6-12-2021	Demo app parent directory name updated. Updated logic to select MU1 base address for S32K3X2 devices. Updated logic to select host core in python script.
1.7	12-1-2022	Demo app parent directory name updated. Added copyright to source code
1.8	19-7-2022	Demo App code restructured for better readability Names of file structures updated Added support of export keys for ECC Build Configuration modifications in various .cmm scripts
1.9	25-08-2022	Added KDF (PBKDF2, HKDF and TLS 1.2) examples