Security Installer Guideline

Table of Contents

1. Introduction

The following document describes the steps to activate security features on NXP platforms by provisioning the HSE FW on a virgin device.

Disclaimer: This guide has demonstrative purposes and the steps described should not be used in production. 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

Abbreviations	Description
ADKP	Application Debug Key/Password
ASB	Advanced Secure Boot
BSB	Basic Secure Boot
CR	Core Reset
IVT	Initialization Vector Table
MRK	Master Root Key
OTFAD	On The Fly AES Decryption
SMR	Secure Memory Region

Terms	Description
Application image	A binary image containing the user application code. This is stored in plain format and optionally signed if secure boot is enabled.
Blob image	A binary image that defines the layout of external flash memory. Contains all images used by BootROM/HSE FW to boot and configure the system at start-up.
Backup image	A binary image that is part of the blob image. Loaded by BootROM in case the primary image fails (e.g. gets corrupted). Address locations are specified in the IVT image.
DCD/Self-Test image(s)	Binary image(s) which allow peripherals configuration and sanity checks to ensure system consistency. See Appendix 11 for an example.
Device-specific key	A key derived for Master Root Key. On rev 1 it is used for encrypting and/or signing artifacts (e.g. IVT/DCD, Application image in BSB). Any such image would be tight to the device it was generated on and would not be portable on other devices.

HSE FW	HSE firmware code: together with BootROM, it provides security services for the entire platform. In the blob image, HSE FW image is encrypted and signed (see HSE FW Pink and Blue 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 Blue image	HSE FW image generated from the pink image via a firmware update request to HSE FW. This is encrypted and authenticated with a device-specific key using a symmetric algorithm.
Image header tag	The first byte of an image part of the blob image. Each such image can be uniquely identified by this byte.
IVT image	The first segment (256 bytes) of the blob image. Contains some attributes configured by BootROM and references to the other images used by BootROM/HSE FW at boot time; for more details see *IVT Information*.
Primary image	A binary image that is part of the blob image. First choice for BootROM to load the image contents. Address locations are specified in IVT image.
SYS_IMG	An encrypted image that stores the HSE persistent assets (NVM attributes, keys, and SMR/CR entries). On flash-less devices, this image is part of the blob image and is stored in the external memory.

3. IMPORTANT - Scope

This document is an extract from HSE DEMOAPP package. If a version compatible with this release is available, **download the DEMOAPP package** and follow the steps described in the root Readme file.

This HSE FW release is compatible with **HSE_DEMOAPP_S32G2XX_x.2.51.0.**

4. HSE FW Installation

4.1. Relevant Terms

- Blob image
- IVT image
- DCD/Self-Test image(s)
- HSE FW
- SYS_IMG
- Application image(s)
- Image header tag
- Primary image
- Backup image

For details about these terms see *Glossary and Terms*

4.2. Description

The first step in installing the HSE FW on a device is to generate the blob image. The blob defines the layout in flash of the images that will be loaded and booted by HSE core (via BootROM or, in secure boot, HSE FW).

This section describes the steps to generate a blob image containing the IVT image configured with addresses for DCD, HSE FW, Application, SYS_IMG and their backups, along with other fields in HSE FW config (*IVT Information*). The blob image will also contain the associated binary images, except for SYS-Image for which space is just reserved. The DCD image does system RAM initialization before the application is loaded. Additionally, QSPI reconfiguration parameters are written at offset 0x200 in the blob image to boost the booting process. The host application used for this setup is the demo application compiled with *APP_HSE_FW_INSTALL_NO_SEC_BOOT* configuration (already provided in the package). The blob will then be written in QSPI flash after which the HSE and application will be booted (non-secure).

4.3. Generating the binary file of the application

The following steps will guide you through generating the application binaries in S32 Design Studio. To generate them with GHS instead, see *GHS* - Generating the binary file of the application.

NOTE: After the application binary will be written to the QSPI flash as part of the blob image, you may need to recompile it in other modes. To be able to debug the flashed application with symbols afterwards, make sure the ELF file obtained in this section is not lost after a rebuild.

In the next chapters, we will assume that the copied ELF of the flashed image is available as *demo_app\images\%PLATFORM%\HSE_DEMO_%PLATFORM%.elf*.

1) In S32 Design Studio > *File* > *Open Projects from File System...*;

2) Click on *Directory* in the new open window and select the *HSE_DEMO_%PLATFORM%* project > *OK*. Select the *HSE_DEMO_%PLATFORM%* project from the discovered projects and press *Finish*;

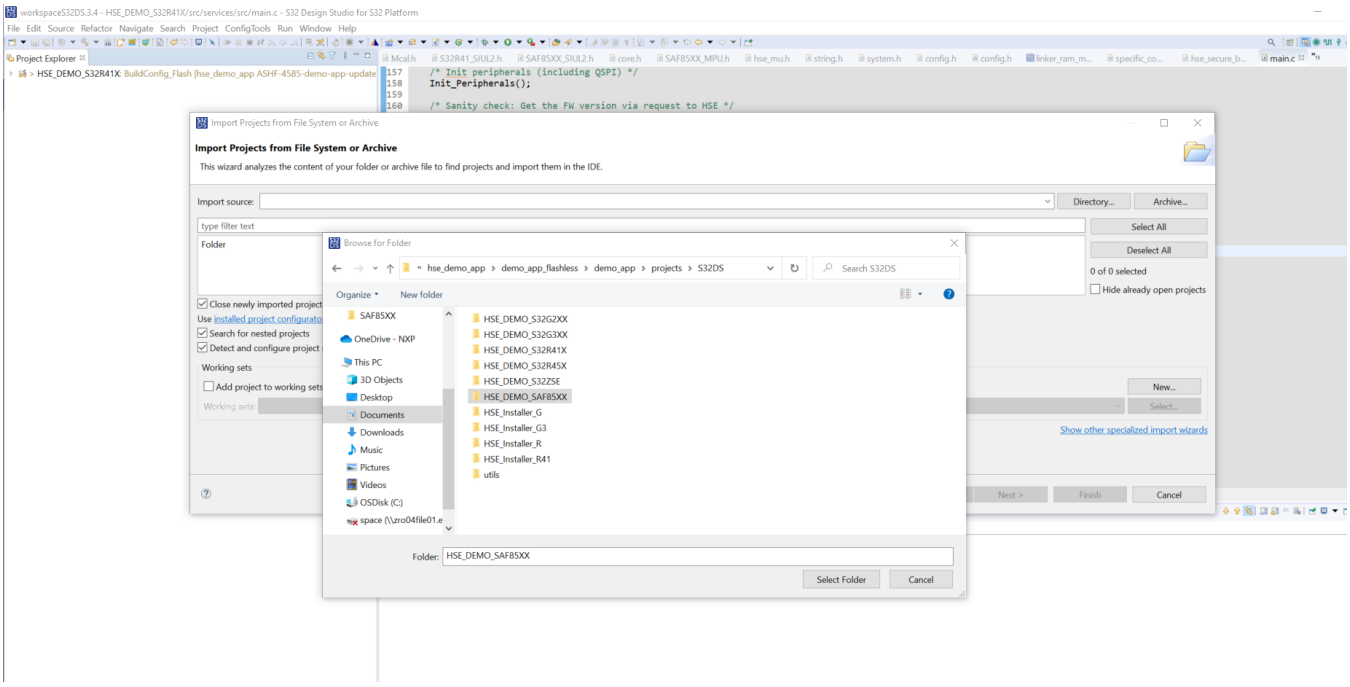


Figure 1 Import S32 DS project and build

3) To build the project, right-click on the project and click *Build Project* or click on the build icon on the left bottom side. The binary is created in the *BuildConfig_Flash* directory in the Design Studio Project location.

NOTE: The generated binary does not contain the ".bss" and ".stack" sections mapped in system RAM. Before accessing this section the system RAM must be initialized (by including a DCD image doing system RAM initialization in the blob – see Appendix 10) or by updating the initialization functions to write in chunks of 64-bits from 64-bits aligned addresses). Otherwise, an ECC error might be reported.

4.4. IVT configuration and blob image generation

This section describes how to configure the IVT and generate the blob image in S32 Design Studio. To perform these steps using a Python script instead, see *Python - IVT configuration and blob image generation*.

1) Open IVT configuration tool and select the *HSE_DEMO_%PLATFORM%* project;
2) Fill in the paths as shown in the snapshot below:
    - DCD image (e.g. *dcd_init_sram.bin, dcd_init_all_realtime_sram_and_no_pll_hse_case.bin, dcd_set_gpio25_and_init_sram.bin*, etc.). Initializes system RAM using SRAMCR(s) and, depending on the platform, sets additional peripherals. It will be run by BootROM before any image is loaded. Address value example: 0x400;
    - HSE/HSE backup path should use the HSE FW pink image binary (e.g. *%PLATFORM%_hse_fw_[full_version]_pb[date].pink*). Address values example: 0x1000 and 0x5B000;
    - Application Bootloader (and backup) path should be set to use *HSE_DEMO_%PLATFORM%.bin*. Address values example: 0xCD000 and 0x121000;
    - (optional) Check *Configure QuadSPI parameters* and fill the path to the dedicated binary to enable fast QSPI configuration at boot (e.g. *qspi_macronix_ddr_octal_dll_bypass_133MHz.bin*);
3) Fill in HSE FW Configuration information: expand *HSE FW Configuration* subsection under HSE and fill in the addresses for SYS-IMG pointer and backup, flash types (set to QSPI) and flash page size (for QSPI – 0x1000) – **Figure 2**. SYS-IMG pointers will reference a reserved space in blob (44KB each) which will be written at run-time by application after publishing it via HSE service. Address values example: 0xB5000 and 0xC1000;
4) Add header to the application images. After selecting the Application Bootloader binary, three fields will be displayed to be filled in: RAM start pointer, RAM entry pointer, and Code Length. These fields need to be filled in with the addresses the binary was linked to (0x34000000 for SAF85, 0x25000000 for ZE, 0x34080000 for the rest). Also, make sure the Code Length field is aligned to 8 bytes. Export Image -> *app_with_header.bin*. Import the new image (with header appended) – **Figure 3**;
5) Press the *Export Blob Image* button and select where to export it – **Figure 4**.

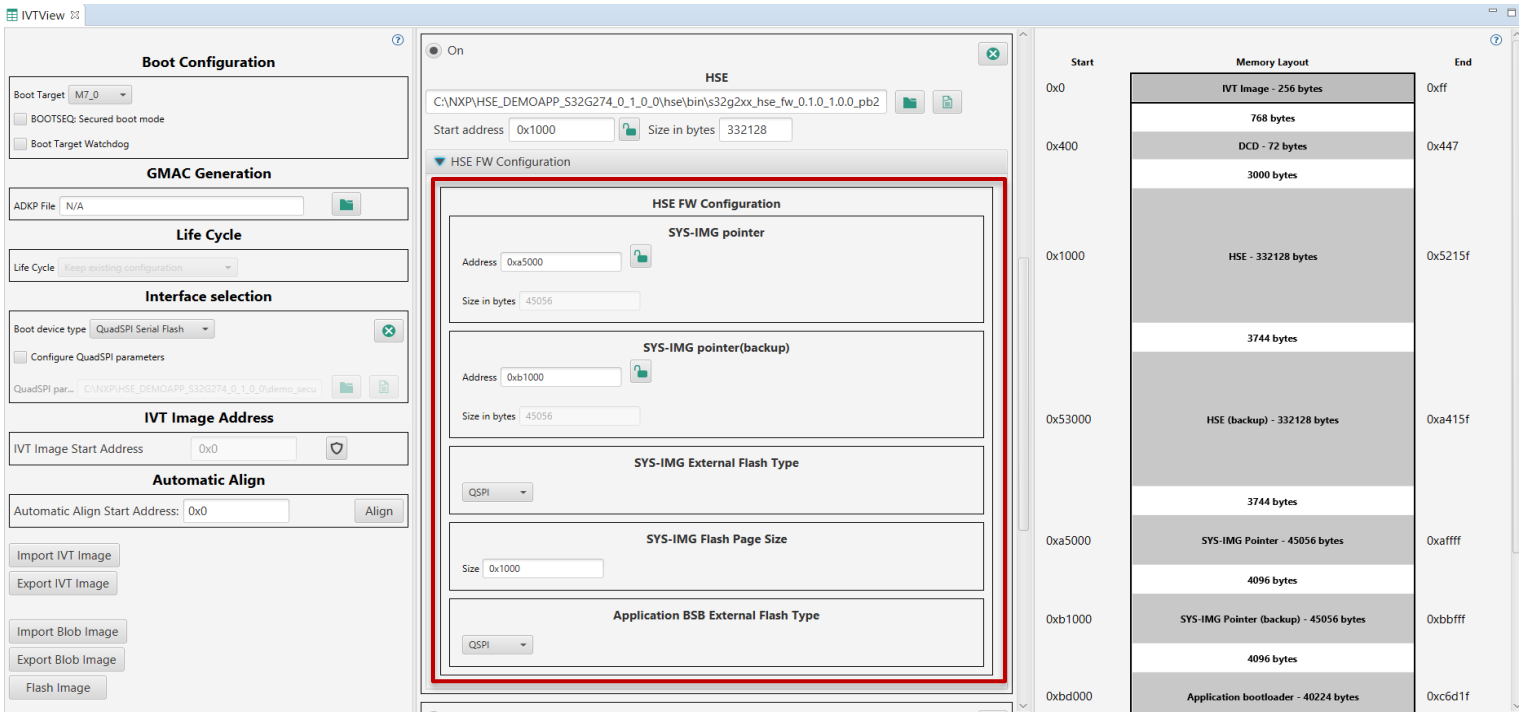


Figure 2 Fill HSE FW Configuration (step 3)

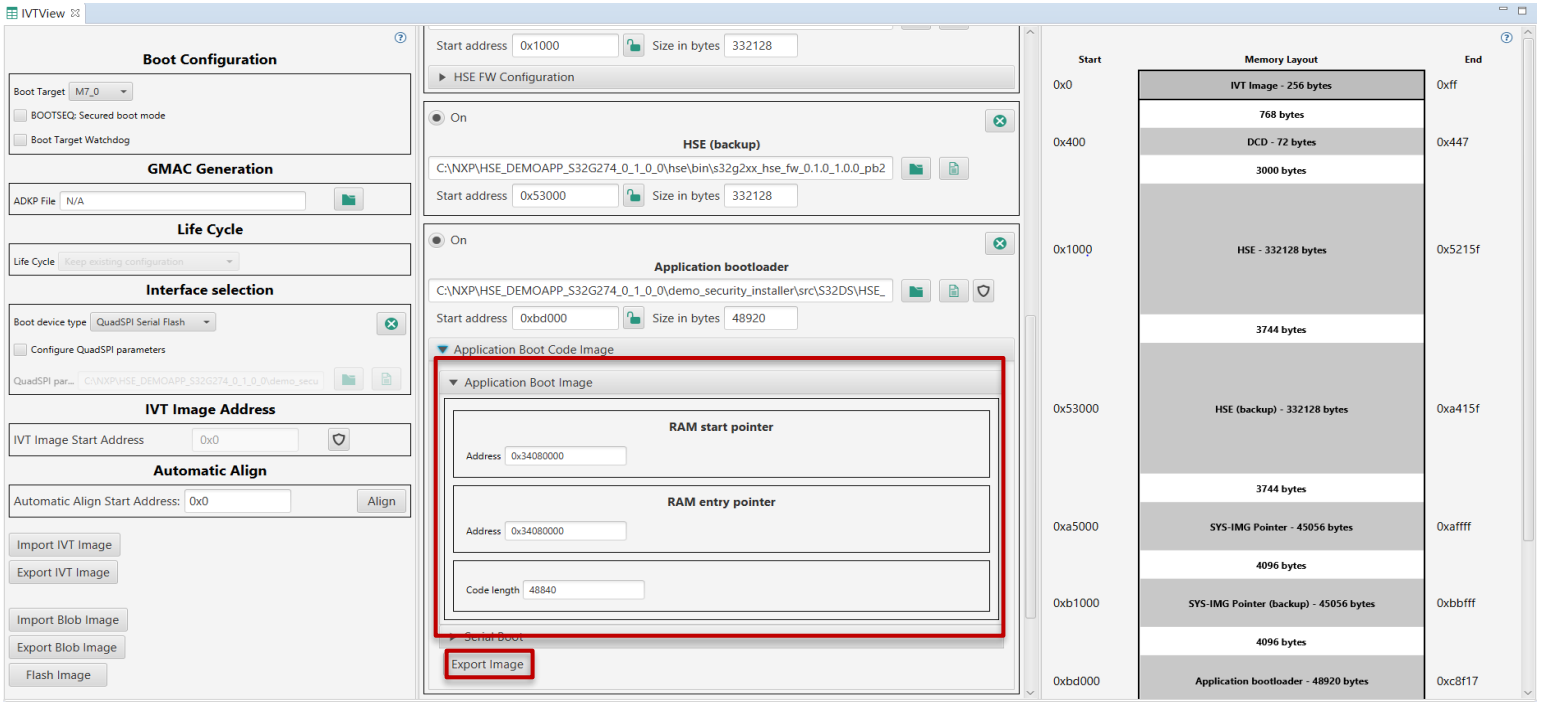


Figure 3 Generate application header (step 4)

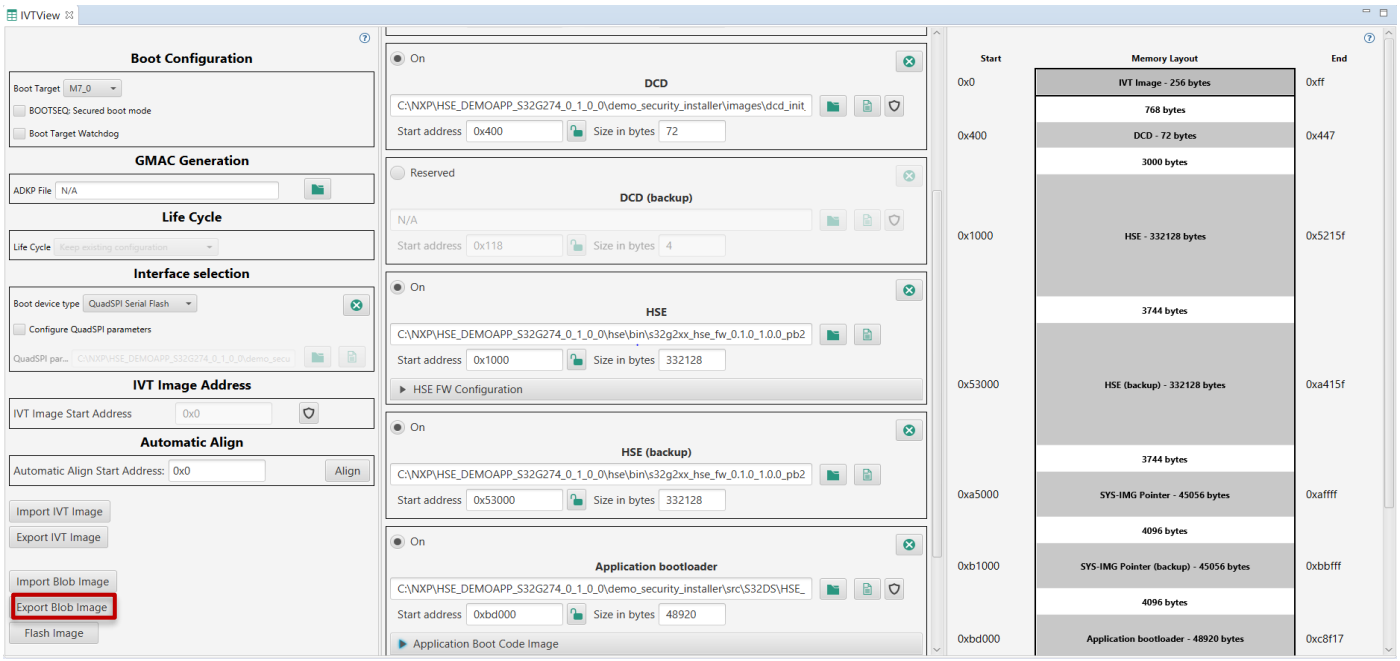


Figure 4 Generate blob image (step 5)

NOTE: For configuring the IVT with custom addresses for the images that are part of the blob, the following items should be taken into consideration to have all examples function correctly:

- HSE, HSE backup, SYS-IMG, SYS-IMG backup addresses must be aligned to 0x1000 (application limitation);
- For Secure Boot examples, 96 bytes after Application Bootloader image are used to store the signatures (16 for BSB, the next 80 bytes for ASB); these 96 bytes should be free. If the header is added during blob configuration using IVT tool, the 16 bytes for BSB are automatically reserved as part of *app_with_header.bin*;
- For On-Demand ASB (enabled by default on ASB example) some space for the dummy code of M7_1 is reserved after the primary image of M7_0 (checkout *hse_secure_boot.c*);
- For automatic alignment of the images use Align option within IVT tool; however, take the above constraints into consideration.

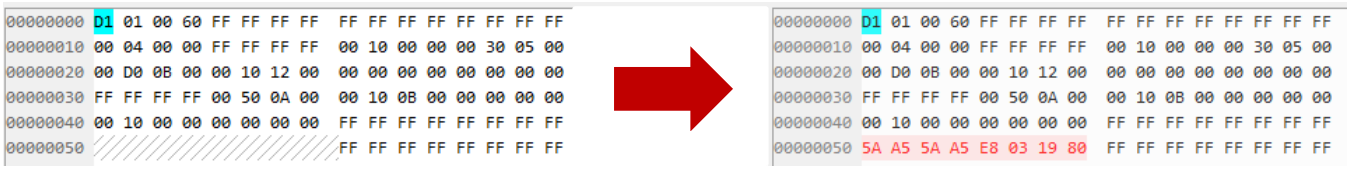
4.5. [S32G-RDB] Updating the IVT with VDD_EFUSE configuration

To ensure HSE functions correctly after SYS_IMG or HSE FW blue image update, the VDD_EFUSE pin must be connected to 1.8V during these processes. These can be achieved via different methods (more details in Appendix 4). The recommended way to do it is via VDD_EFUSE configuration within IVT (more details about the fields in *IVT Information*), which currently is not supported in S32DS.

To include these fields into the generated blob image, one can either edit the blob in a hex editor and add the necessary information manually, or run the python script to update the blob image:

```
python3 blob.py -o update -j json\%PLATFORM%\update_ivt_with_efuse_config.json -b
..\..\..\images\%PLATFORM%\blob.bin
```

The resulting image should have the IVT VDD_EFUSE configuration bits updated as follows:



NOTE: This is applicable only for boards having this support in HW and not having the VDD_EFUSE connected by default to 1.8V. Similarly, the GPIO pin used to power on the VDD_EFUSE pin may differ, which may imply updates on the VDD_EFUSE configuration values themselves. This is an example for S32G RDB. Some platforms may not have this support and would need to set the VDD_EFUSE as described in Appendix 4.

4.6. Flashing the blob and booting from QSPI

The below steps assume a setup with S32 Design Studio and an S32 Debug Probe. For a setup with TRACE32 PowerView for ARM and a Lauterbach PowerDebug interface, see *TRACE32 - Flashing the blob and booting from QSPI*.

Program the blob image to the QuadSPI flash:

1) Power off the board and configure it to boot from the serial interface – *Appendix 2*;
2) Power on the board and flash the device with the blob image created previously. With S32 Design Studio, this can be achieved in any of the following ways:
   a. Using the S32 Flash Tool within IVT view of S32 Design Studio: Click on the *Flash Image* button in *IVT View* to program the blob with via UART – see **Figure 5**.
   b. Using the flash programmer (via JTAG):

i. Update the *"HSE_DEMO_%PLATFORM%_<build_configuration>_S32Debug_Flash_blob_image"* debug configuration, as indicated in Appendix 5, and provide the path to the blob image instead of an ELF path. By default, the configuration will be *"<build directory>/blob_s32<g2x/r45>x.bin"*. If you choose a location external to the project, a *"Cannot find project for selected C/C++ application"* warning will be displayed at the top of the window, but will not affect the flashing process;

ii. Run the above configuration. Switching to the *Debug* perspective is not necessary at this point, so you can decline if prompted. The *Progress Information* window will be opened and will show the message *"Flashing operation completed."* upon completion.

Alternatively, you can use the flash programmer tool from the GDB command line. For a description of the procedure, refer to the *HOWTO_Command_Line_JTAG_flash_programming_with_S32_Debug_Probe_on_S32<G274A/R45>_EVB.pdf* document, which can be found in *<S32DS_install_dir>\S32DS\help\resources\howto\<g2/r45>dev* after installing the S32<G2/R4>XX development package.

NOTE: When using the flash programmer tool from the command line:

- make sure to also add *<S32DS_install_dir>\S32DS\build_tools\msys32\mingw32\lib\python2.7\site-packages* to *PYTHONPATH*;
- the *_SOC_NAME* parameter needs to be defined before *"py flash()"*, e.g.:

  py _SOC_NAME="S32G274A"

- if the probe is connected through USB, *_PROBE_IP* can be set to the MAC address of the probe, e.g.:

  py _PROBE_IP="00:04:9F:06:1F:EF"

3) Power off the board and configure it along with the RCON pads to boot from QSPI – Appendix 3

NOTE: In case 2) b. above, JTAG flash programming does not require serial communication and thus can be performed even without a UART cable. However, the steps 1) and 3) above are recommended in this case as well, to avoid potential issues caused by QSPI settings of the currently loaded application.

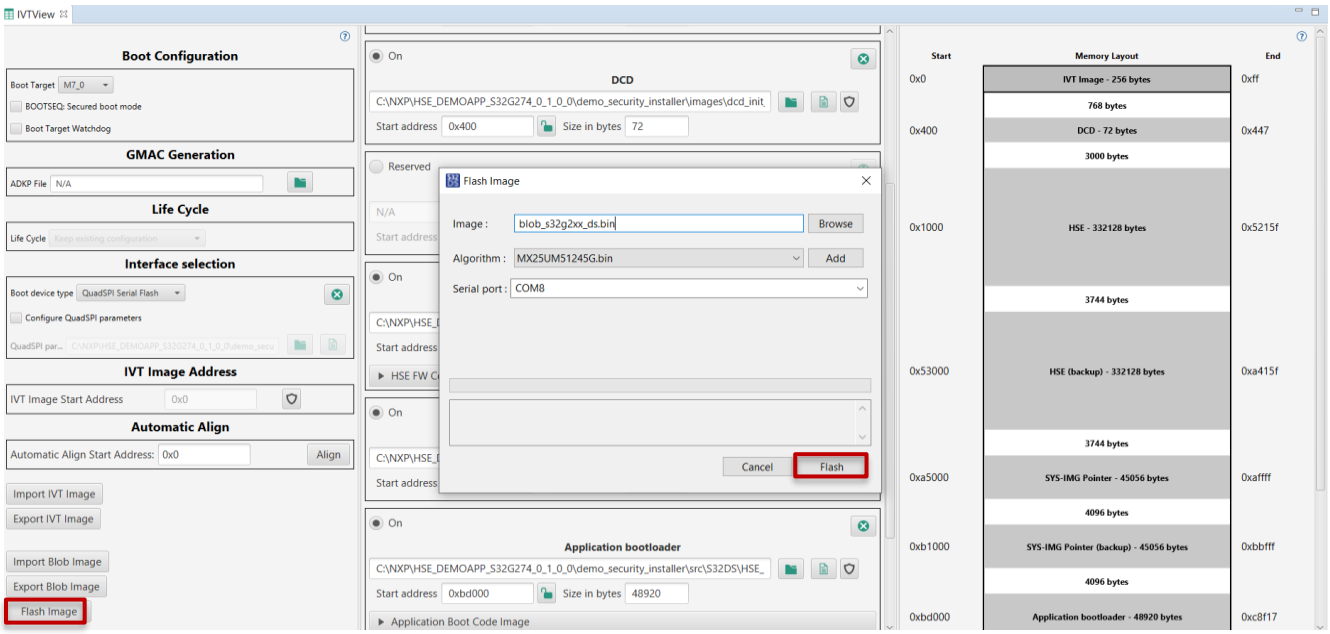


Figure 5 Flash the blob (step 2a)

4.7. Run the examples

1) Make sure VDD_EFUSE pin is configured by either SW/HW (Appendix 4);
2) (optional) With the UART cable connected between board and host, configure a serial terminal: 8N1 without FC (default in PuTTY), baud rate 115200;
3) Power on the board (configured for QSPI boot);
4) (optional) Check the output on the serial terminal to confirm that the application booted correctly and the HSE is up and running – Figure 6. With this acknowledgement, step 7) can be skipped;

Figure 6 HSE up and running

5) Update the *"HSE_DEMO_%PLATFORM%_M7_0_<build configuration>_S32Debug_Load_symbols"* debug configuration, as indicated in Appendix 5. If the binary files were generated with Design Studio, *"C/C++ Application"* can be left to the default value: *"<build directory>|HSE_DEMO_%PLATFORM%.elf"*. Otherwise, update this field with the absolute path of the ELF file;

6) To start a debug session which also loads the symbols of the flashed application, run the updated **_Load_symbols* debug configuration. If prompted, allow the launch to open *Debug* perspective;
7) In the *Debug* view, expand the debug configuration item, select the one with the ELF name and click the *Suspend* (⏸) button on the toolbar to pause the application execution;
8) If you haven't already done so in step 3) using the terminal emulator, validate that the HSE FW version has been correctly retrieved from the HSE – **Figure 7**:
    a. Open the *Expressions* view from *Window* > *Show View* > *Other* > *Debug* > *Expressions*;
    b. Add the *gGetVersionExample* and *gHseFwVersion* variables in the table, using *Add new expression*.
9) Skip the infinite loop by setting *gZero* to 1 and explore the examples. The value of *gZero* can be changed in any of the following ways:
    a. Add the variable in the *Expressions* view as described in step 7) and edit its *Value* cell;
    b. Place the mouse cursor over the variable name in the code. This will display a hover similar to the *Expression* view, but only with a *gZero* entry, allowing to edit its value – **Figure 8**.
    c. Issue the *"set gZero = 1"* command in the GDB console (the *Debugger Console* view). Use *"print gZero"* for validation.

    For more information about the *Debug* perspective, see the section *"Using the debugger"* in the S32 Design Studio User Guide.

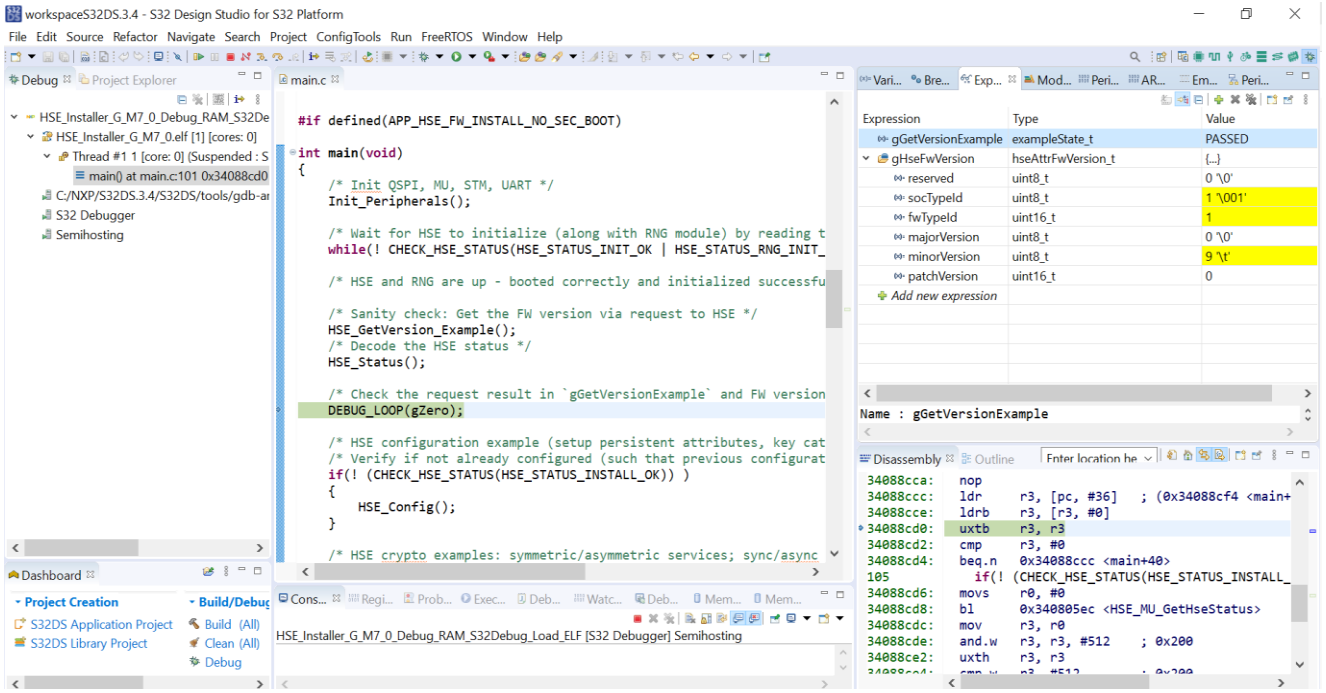


Figure 7 HSE up and running S32 Debug

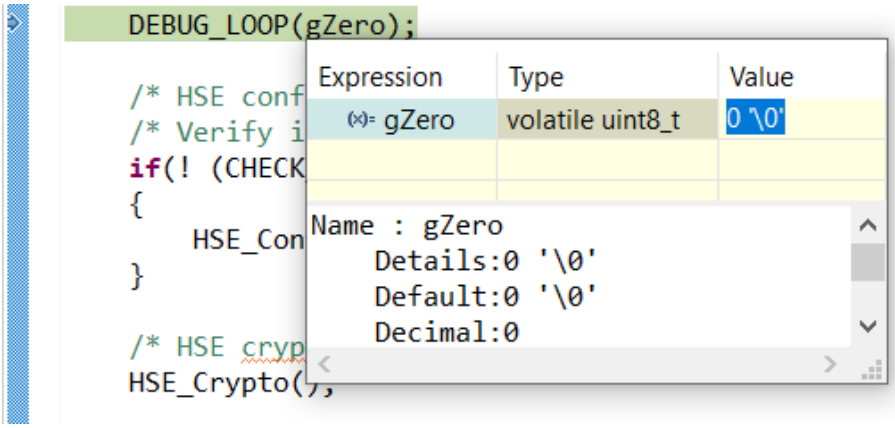


Figure 8 Edit gZero

5. Appendices

The following pages contain useful information for configuring and running this application.

Appendix 1. IVT Information

NOTE: Not all the IVT fields are described below. For more details, use HSE Firmware Reference Manual.

IVT offset

Boot device type	Image vector table offset		
	HSE-H	HSE-M (R41/SAF85)	
SD/MMC/eMMC	1000h	N/A	
QuadSPI Serial Flash	0h	0h	

IVT image header

Byte 0	Byte 1	Byte 2	Byte 3
TAG = D1h	Length = (100h)		Version = (60h)

IVT image structure

Address	Size (Bytes)	Name	Comments
0h	4	IVT header	Header showing the start of the IVT.
4h	4	Reserved 1	
8h	4	Self-Test DCD Pointer	Pointer to the start of the configuration data used for BIST.
Ch	4	Self-Test DCD Pointer (backup)	Pointer to the start of the backup configuration data used for BIST.
10h	4	DCD Pointer	Pointer to the start of DCD configuration data.
14h	4	DCD Pointer (backup)	Pointer to the start of backup DCD configuration data.
18h	4	HSE Firmware Flash Start Pointer	Pointer to the start of the HSE firmware in flash memory.
1Ch	4	HSE Firmware Flash Start Pointer (backup)	Pointer to the start of the backup HSE firmware in flash memory.
20h	4	Application Boot Code Flash Start Pointer	Pointer to the start of the application boot code in flash memory.
24h	4	Application Boot Code Flash Start Pointer (backup)	Pointer to the start of the backup application boot code in flash memory.

28h	4	Boot Configuration Word	Configuration data used to select the boot configuration. This includes bit(s) to indicate if HSE firmware is encrypted.
2Ch	4	Life-Cycle Configuration Word	Configuration data used for advancing LifeCycle. For the configuration to remain the same, this word needs to be set to 0.
30h	4	DEBUG_CONFIG	Debug Configuration used to enable debugging on application core from first instruction
34h	48	HSE FW CONFIG	HSE f/w Configuration (see table below)
64h	128	Reserved 2	
E4h	12	IV	Random IV. If Random IV is not used, bytes are treated as reserved.
F0h	16	GMAC	MAC over data from IVT Header till end of Reserved2 section.

IVT Boot Configuration word

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
							Reserved								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					Reserved							BOOT_SEQ	SWT	BOOT_TARGET	

Life-Cycle Configuration word

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Reserved	IN_FIELD	OEM_PROD

HSE FW CONFIG

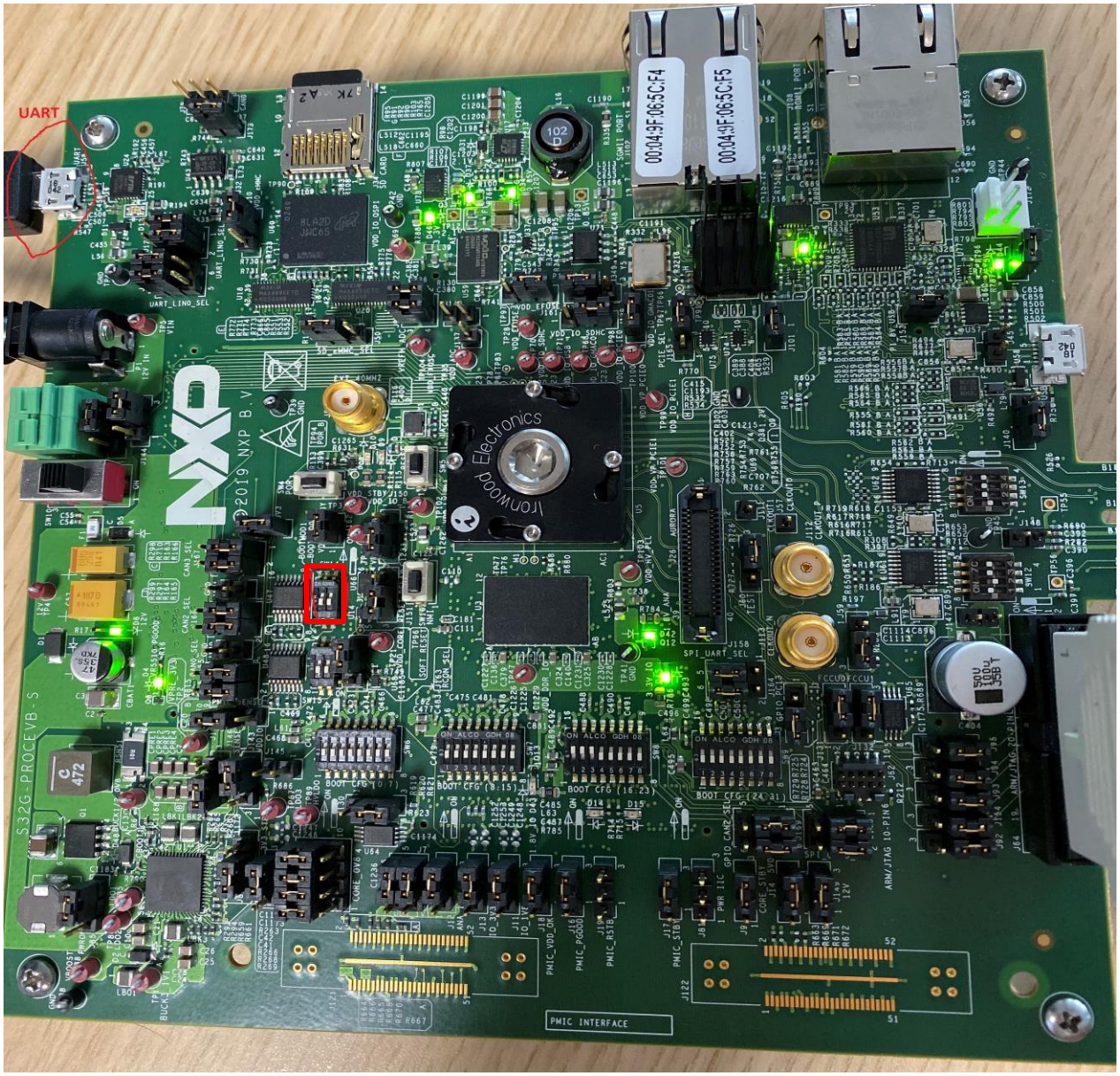
IVT Address offset	Size (bytes)	Name	Comments
34h	4	SYS-IMG Pointer	Pointer to the start of SYS-IMG file (primary).
38h	4	SYS-IMG Pointer (backup)	Pointer to the start of SYS-IMG back-up file.
3Ch	4	SYS-IMG External Flash Type	QSPI:0 / SD card:2 / MMC:3
40h	4	SYS-IMG Flash Page Size	Erasable page size of external flash.
44h	4	App BSB External Flash Type	External Flash Type for Application: QSPI:0 / SD card:2 / MMC:3 Shall be used only if Basic Secure Booting is used:
48h	8	Reserved	Reserved
50h	4	VDD_EFUSE marker	VDD_EFUSE enable/disable. To consider the VDD_EFUSE word must have the value 0xA55AA55A.
54h	4	VDD_EFUSE word	Configuration for Delay, MSCR number and polarity
58h	12	Reserved	Reserved

VDD_EFUSE Configuration word

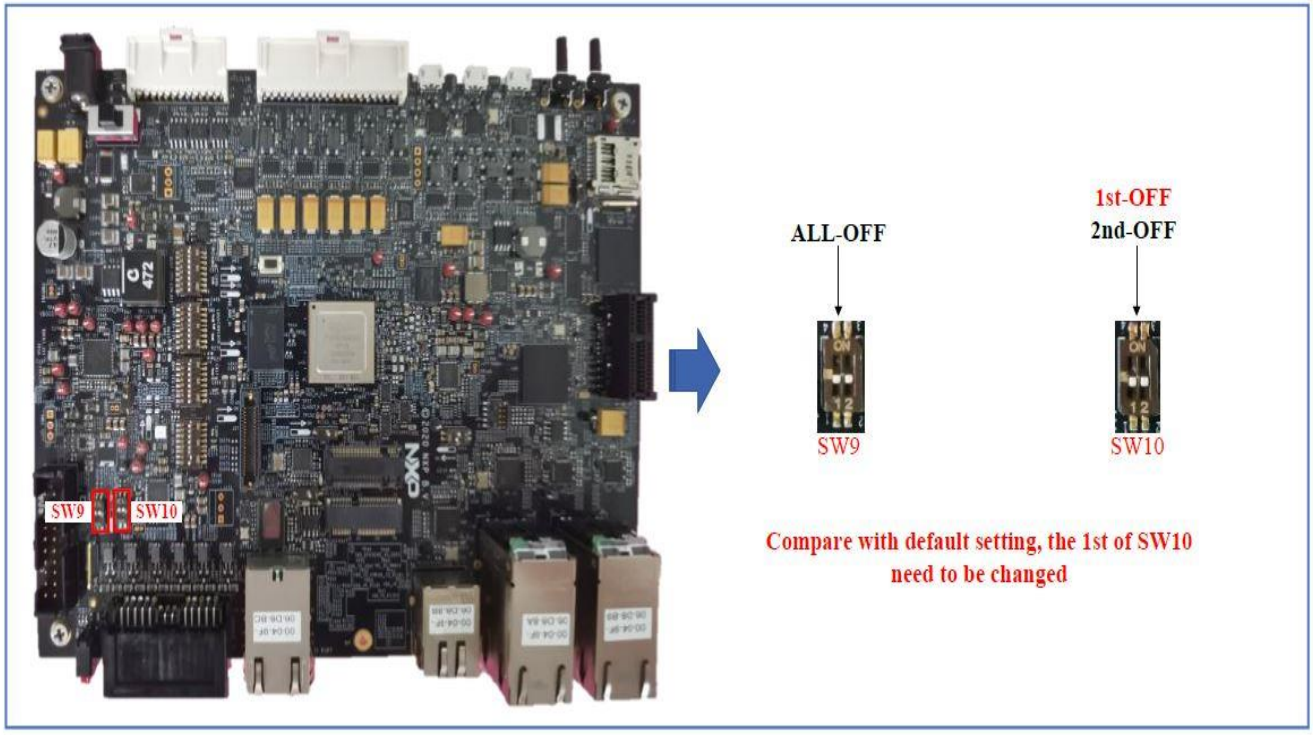
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IO Polarity	GPIO MSCR value														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
delay in us															

Appendix 2. Board configuration for boot from serial interface (UART)

2.1. S32G EVB



2.2. S32G2 RDB2



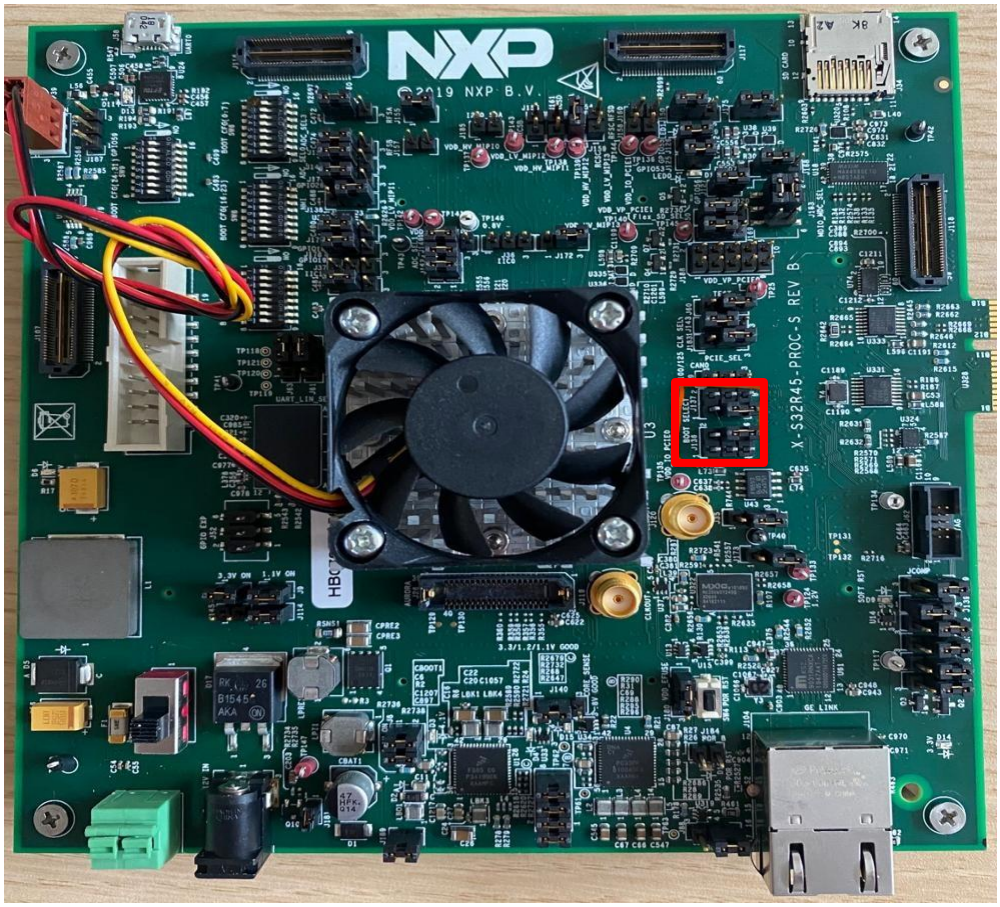
2.3. S32R45 EVB

Board settings for BMOD Pins:

	Value	
BOOT_MODEx	1	0
BOOT_MODE1 (BMODE2 RM)	J136_2_4	J136_6_4: Close
BOOT_MODE0 (BMODE1 RM)	J137_1_3	J137_3_5: Close

For Serial boot with differential mode: Set BOOT_MODE1 to 0 and BOOT_MODE0 to 0.

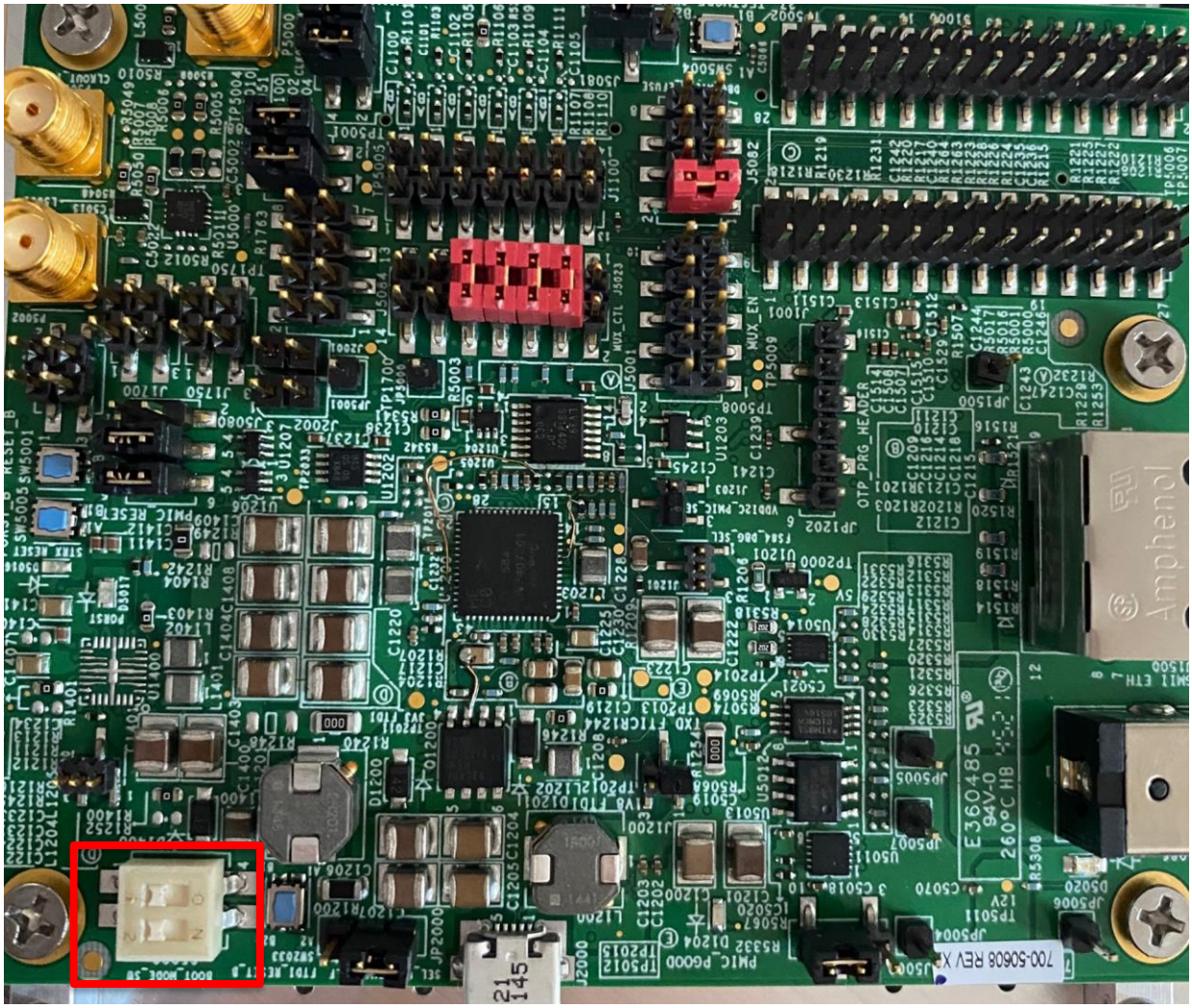
For Serial boot with crystal mode: Set BOOT_MODE1 to 1 and BOOT_MODE0 to 0.



2.4. SAF85 X-STRX-SKT_WG-V4

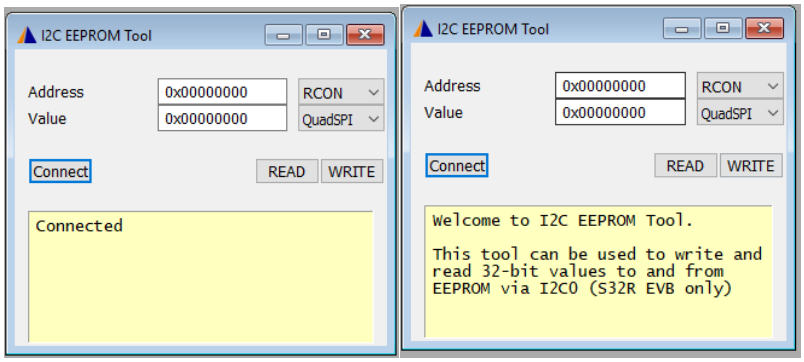
1. Move the buttons of the white Boot Mode microswitch under the JTAG connector like this:

- Left button to position 1 (down)
- Right button to position 2 (down)



2. Ensure the BOOT_CFG1, stored in EEPROM, is 0:

Run the cmm script **S32_Serial_RCON_I2C_EEPROM.cmm** from the **scripts** folder. The below dialog window will open:



Click **Connect**. The script will initialize the debug session and configure the SIUL and I2C_0 modules for interaction with the EEPROM module and will display '**Connected**' status.

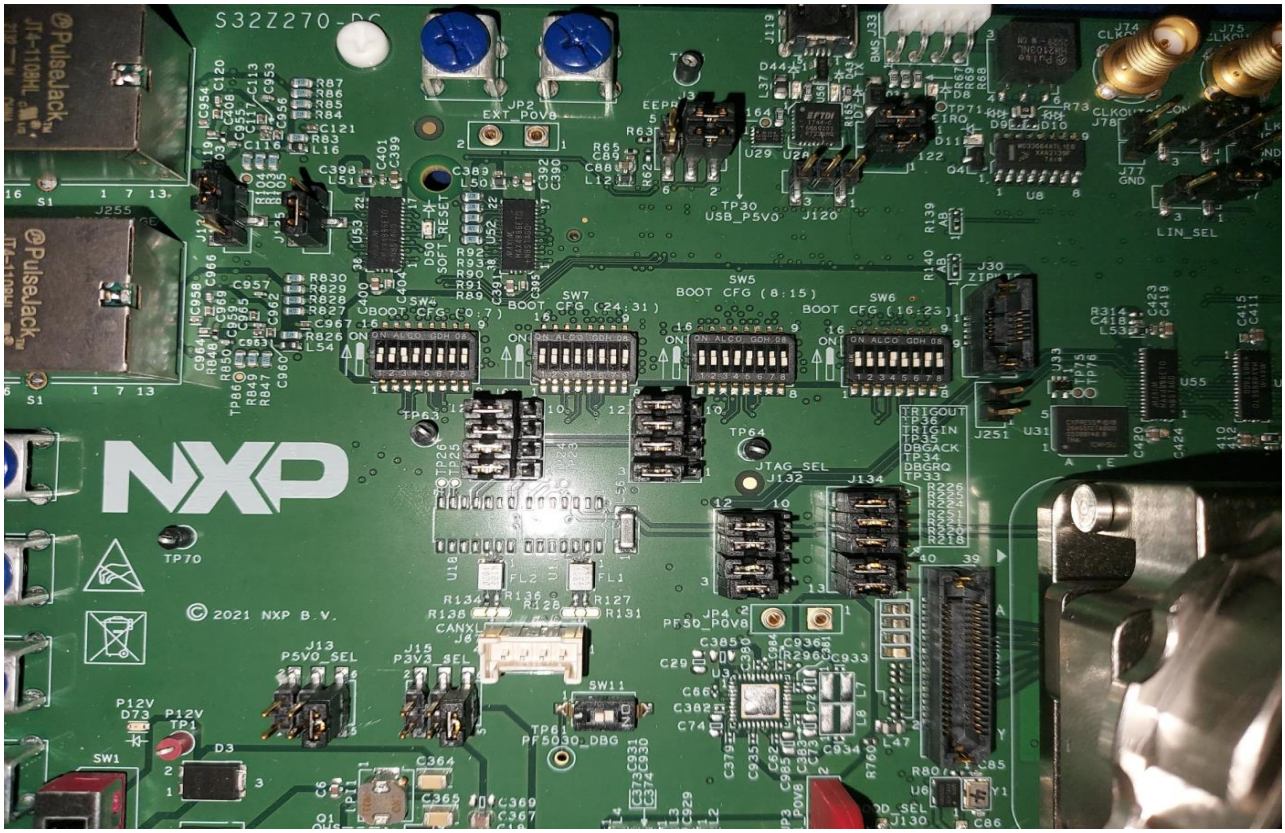
Set the Address to **0x00000000** and click on **READ** button. If the value returned in the status window is **0x0000000**, you can close the script window. There is nothing more to do with it.

If the value returned in the status window is **not** 0x0000000, please set Address to 0x00000000 and Value to 0x00000000 and hit WRITE button.

2.5. S32Z270-DC SCH-50921 REVA

Configure the S32ZE board to boot in serial mode. In order to do this set the following jumpers:

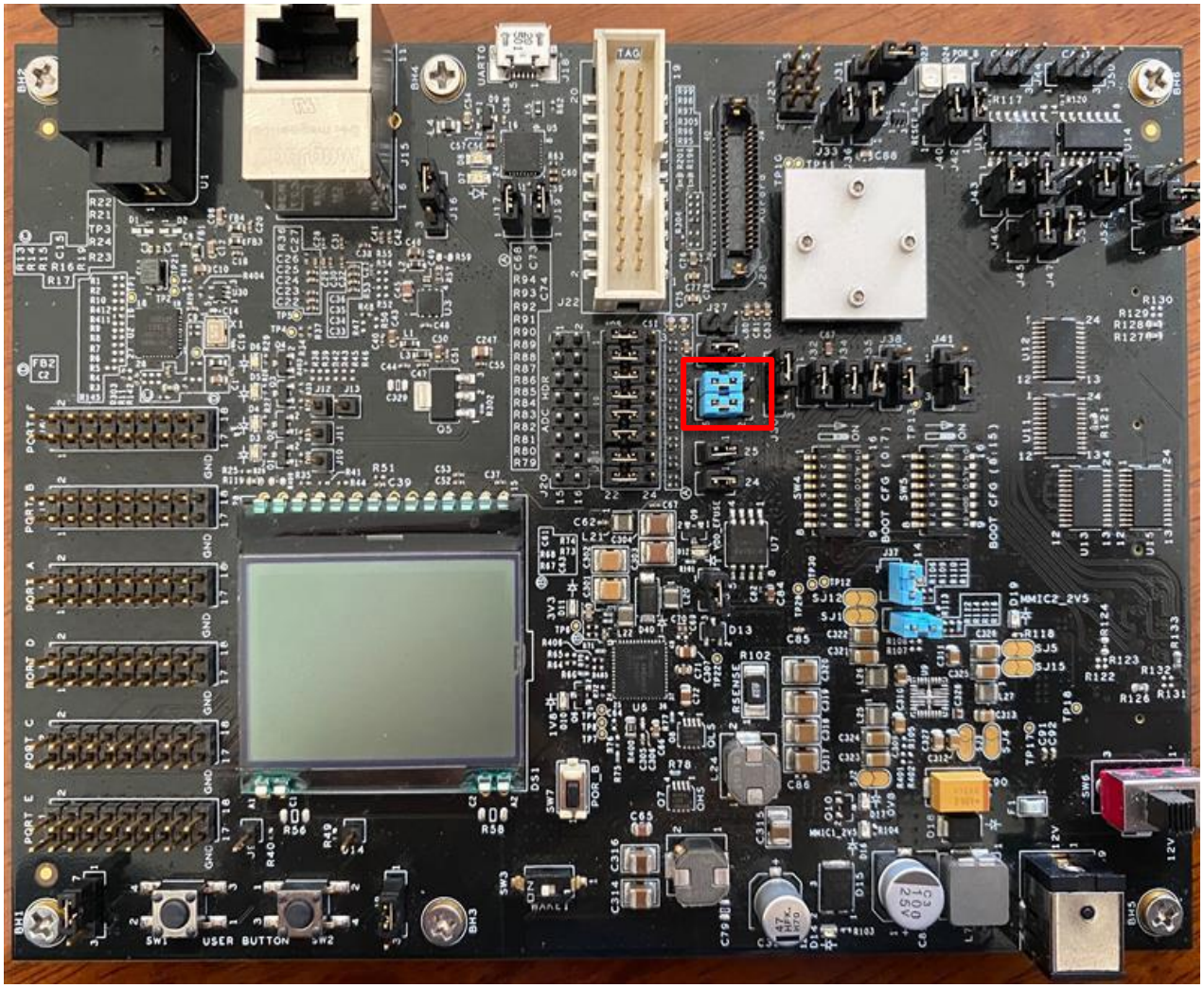
- J124: 2-3
- J125: 1-2
- J247: 1-2

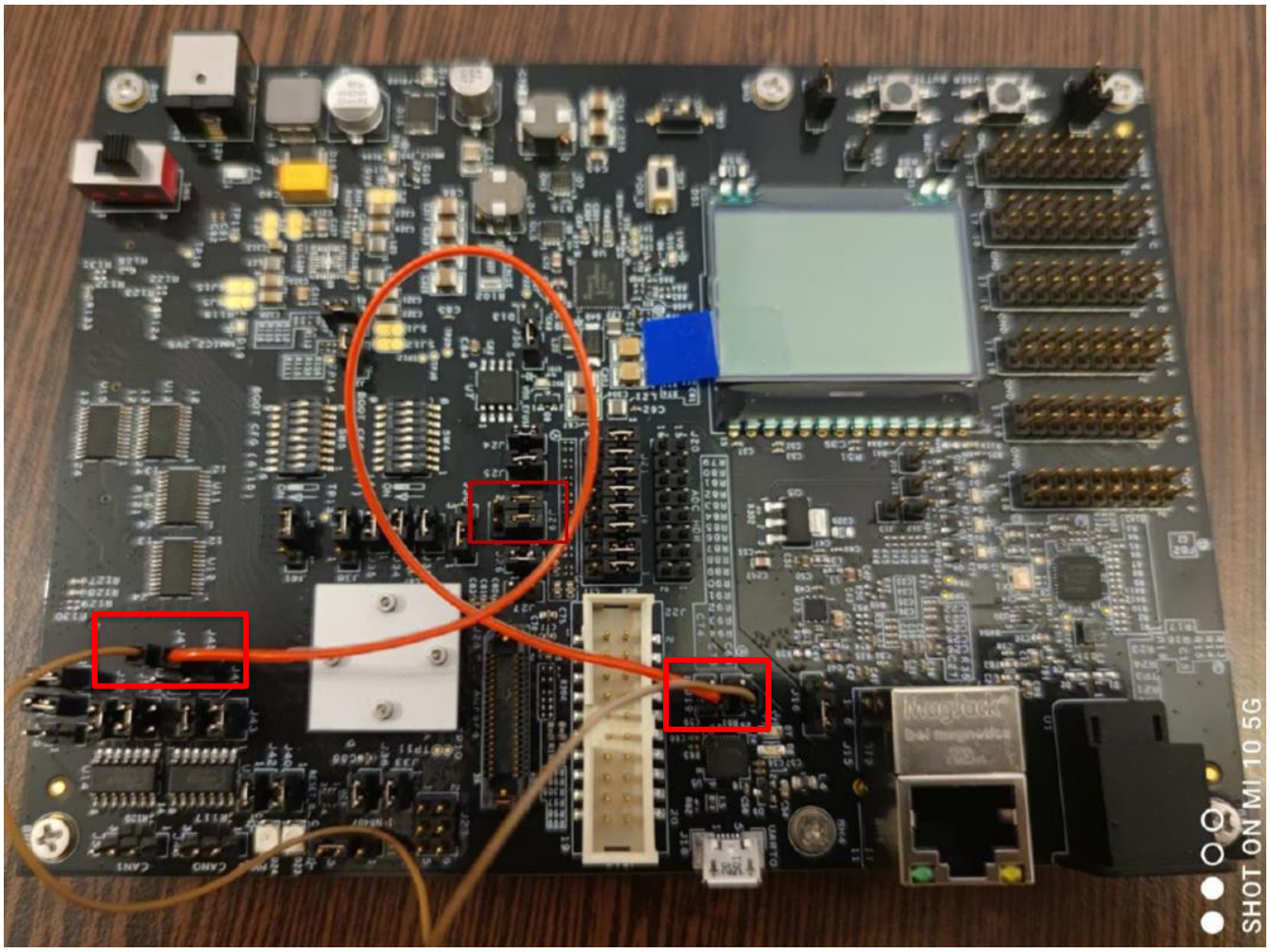


2.6. S32R41-EVB SCH-48194 REV B

- Configure the board with all RCON pads to 0.
- Configure the board in serial boot mode (J29 **3-5**, 4-6).
- (Optional) If using S32DS/S32FlashTool to program the images to QSPI flash, additional jumpers must be set (as shown in the second Figure):
    - J47:1 – J17:2
    - J49:1 – J19:2

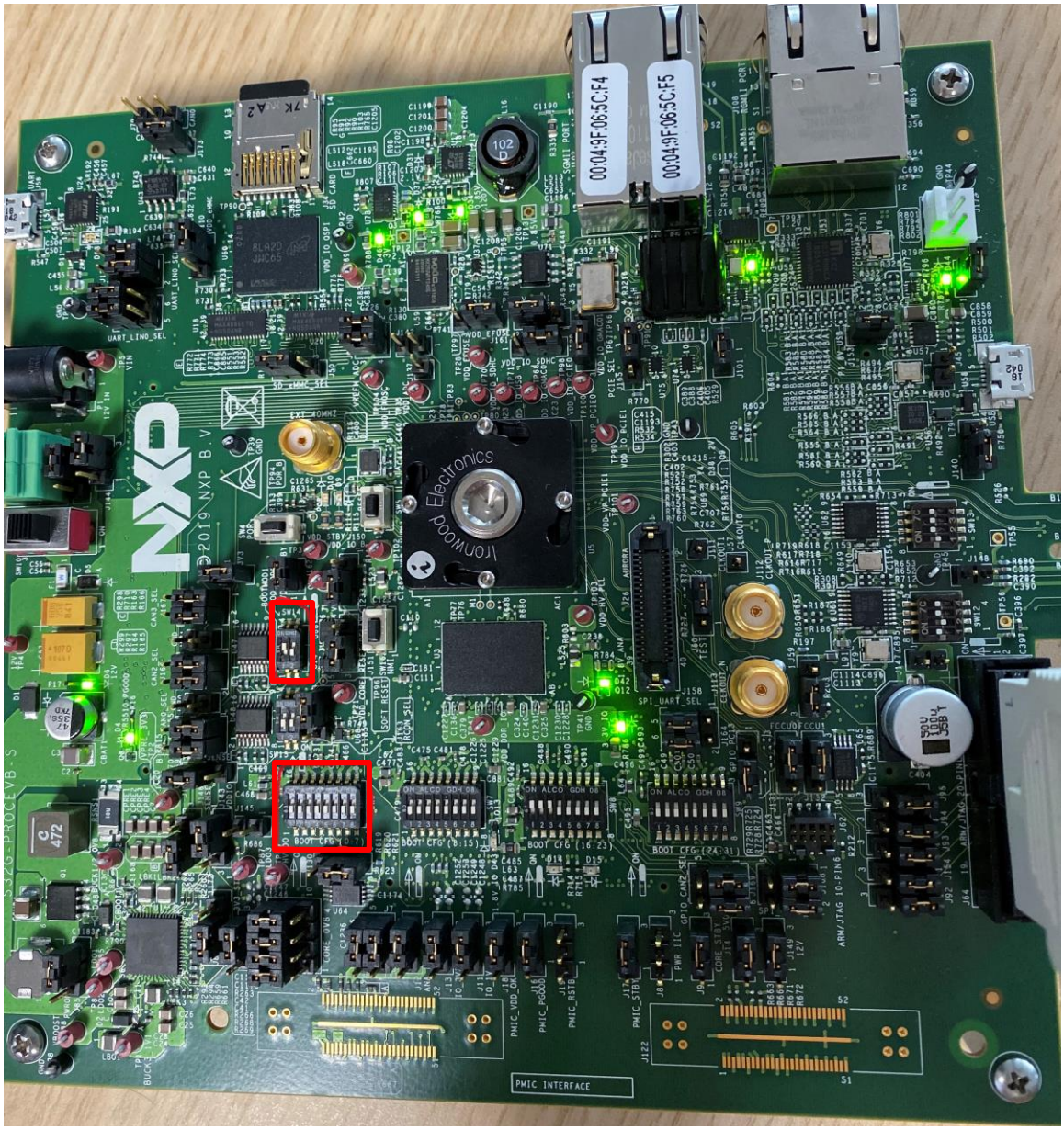
NOTE: This configuration for the UART is incompatible with the one used in the driver from the application. As such the debug messages are not visible in the console (e.g. Putty) when running the examples.



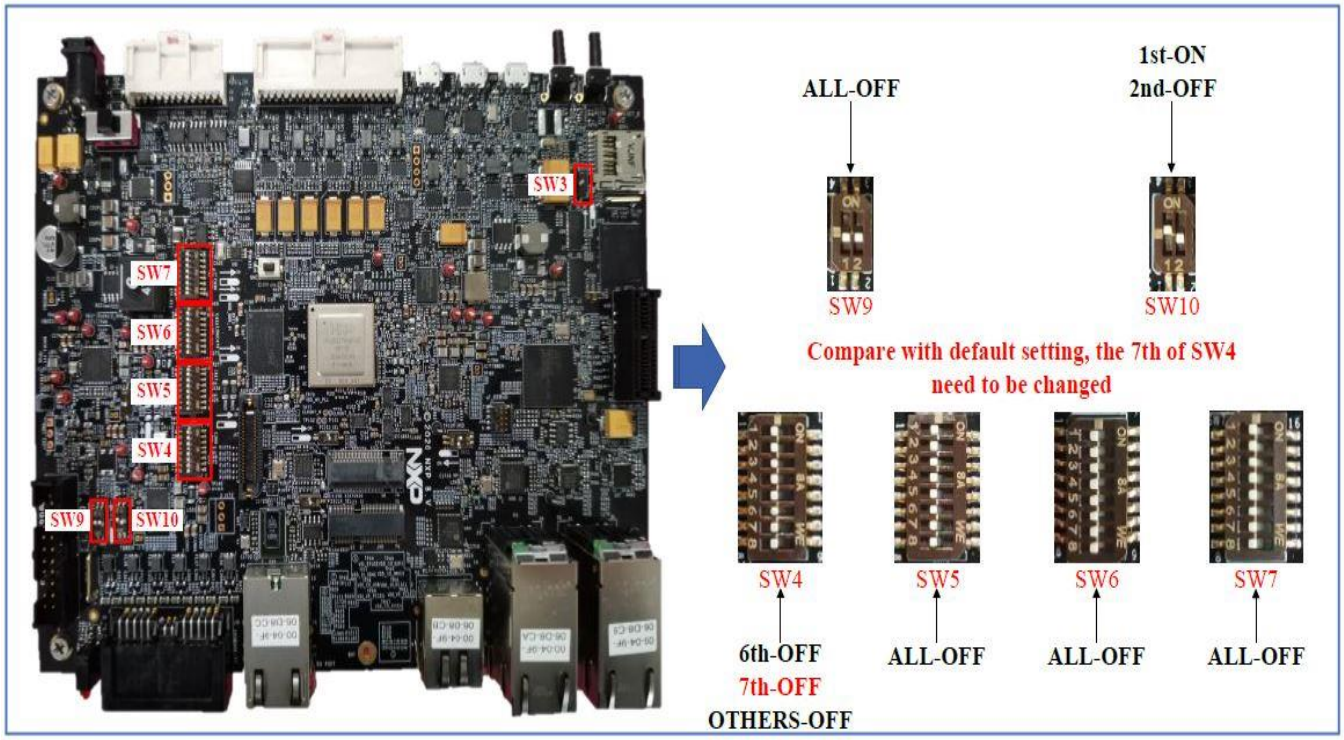

SHOT ON MI 10 5G

Appendix 3. Board configuration for boot from RCON, QSPI external flash

3.1. S32G EVB



3.2. S32G2 RDB2



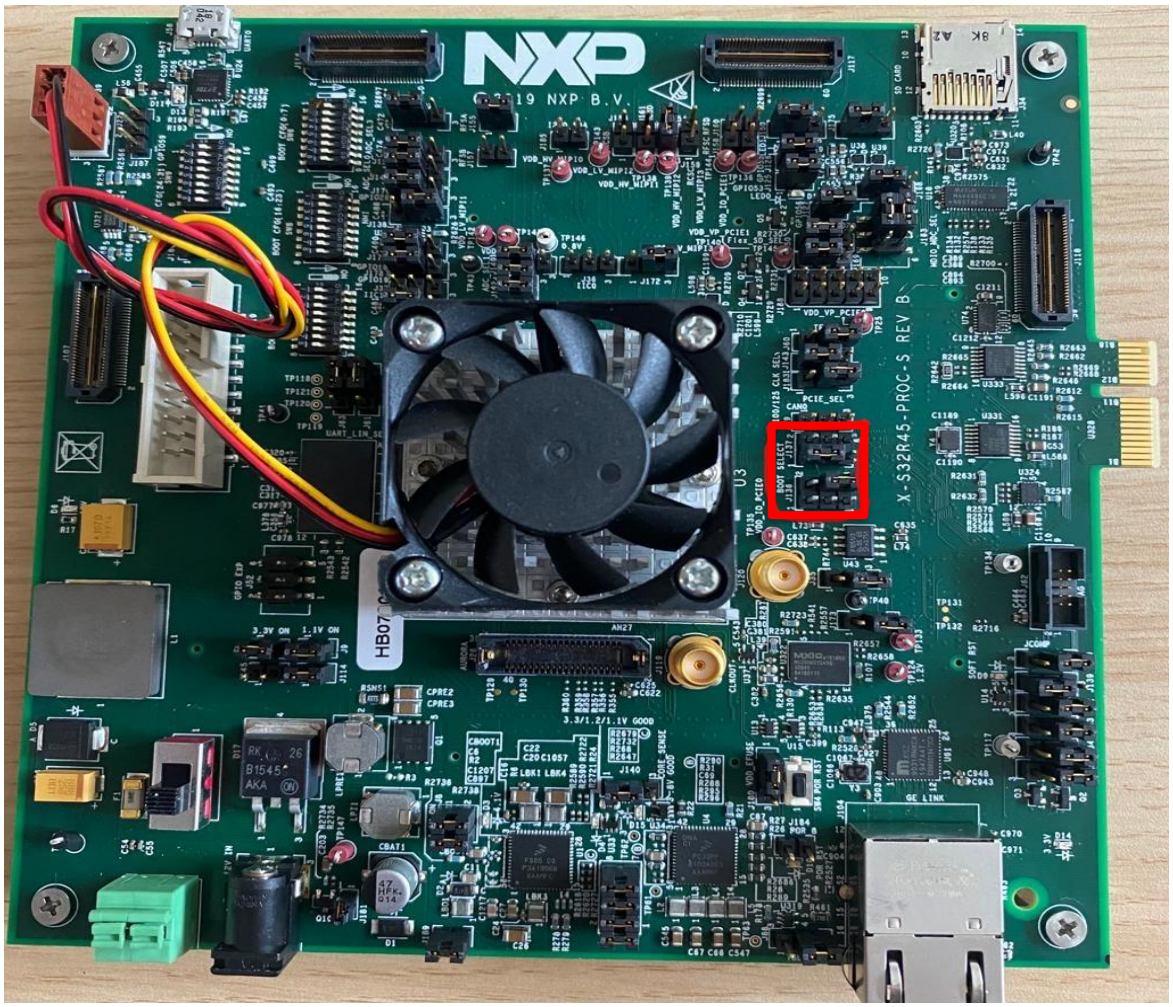
3.3. S32R45 EVB

Board settings for BMOD Pins:

	Value	
BOOT_MODEx	1	0
BOOT_MODE1 (BMODE2 RM)	J136_2_4	J136_6_4: Close
BOOT_MODE0 (BMODE1 RM)	J137_1_3: Close	J137_3_5

For Booting from RCON: Set BOOT_MODE1 to 0 and BOOT_MODE0 to 1.

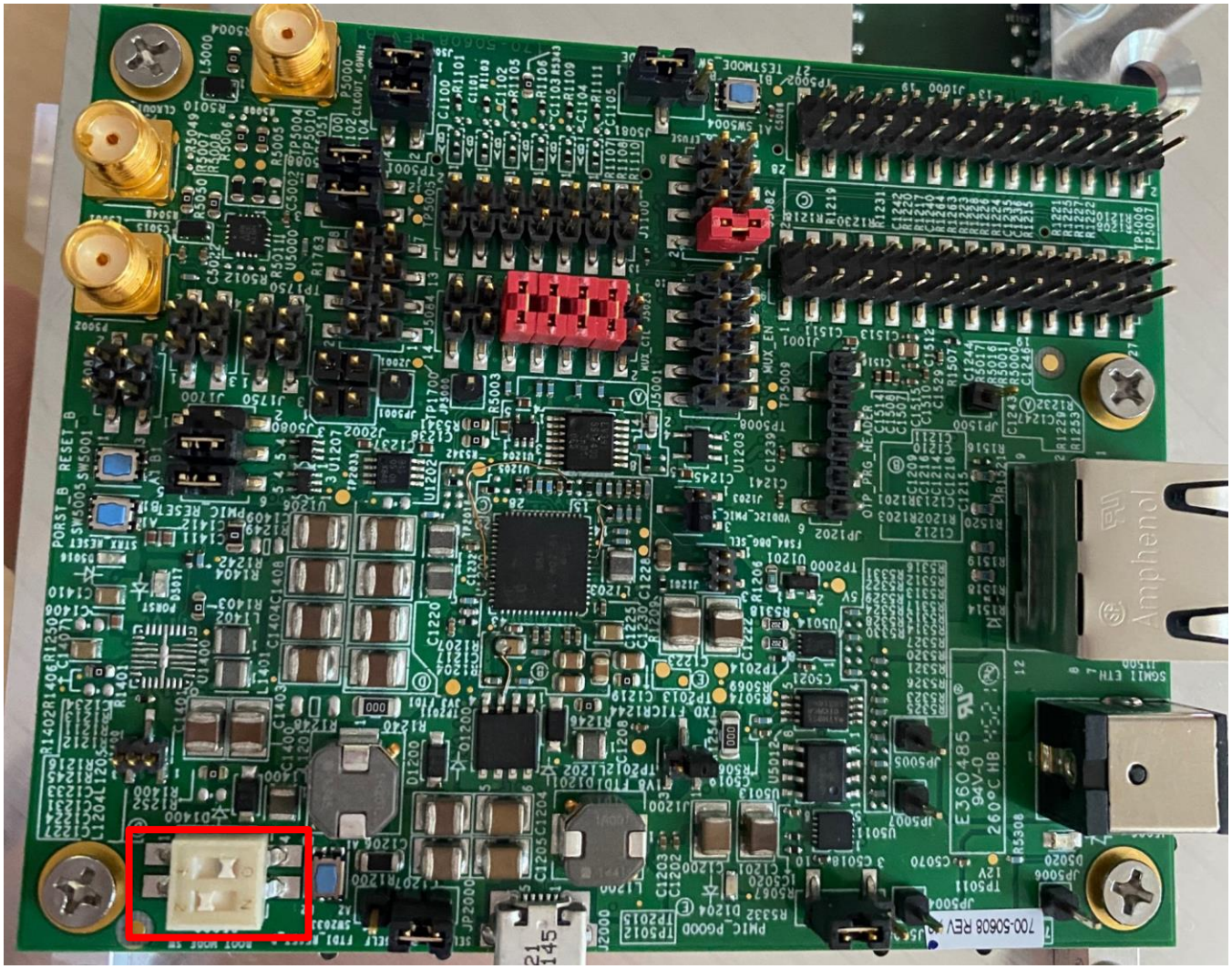
RCON pads: All set to 0



3.4. SAF85 X-STRX-SKT_WG-V4

Move the buttons of the white Boot Mode microswitch under the JTAG connector like this:

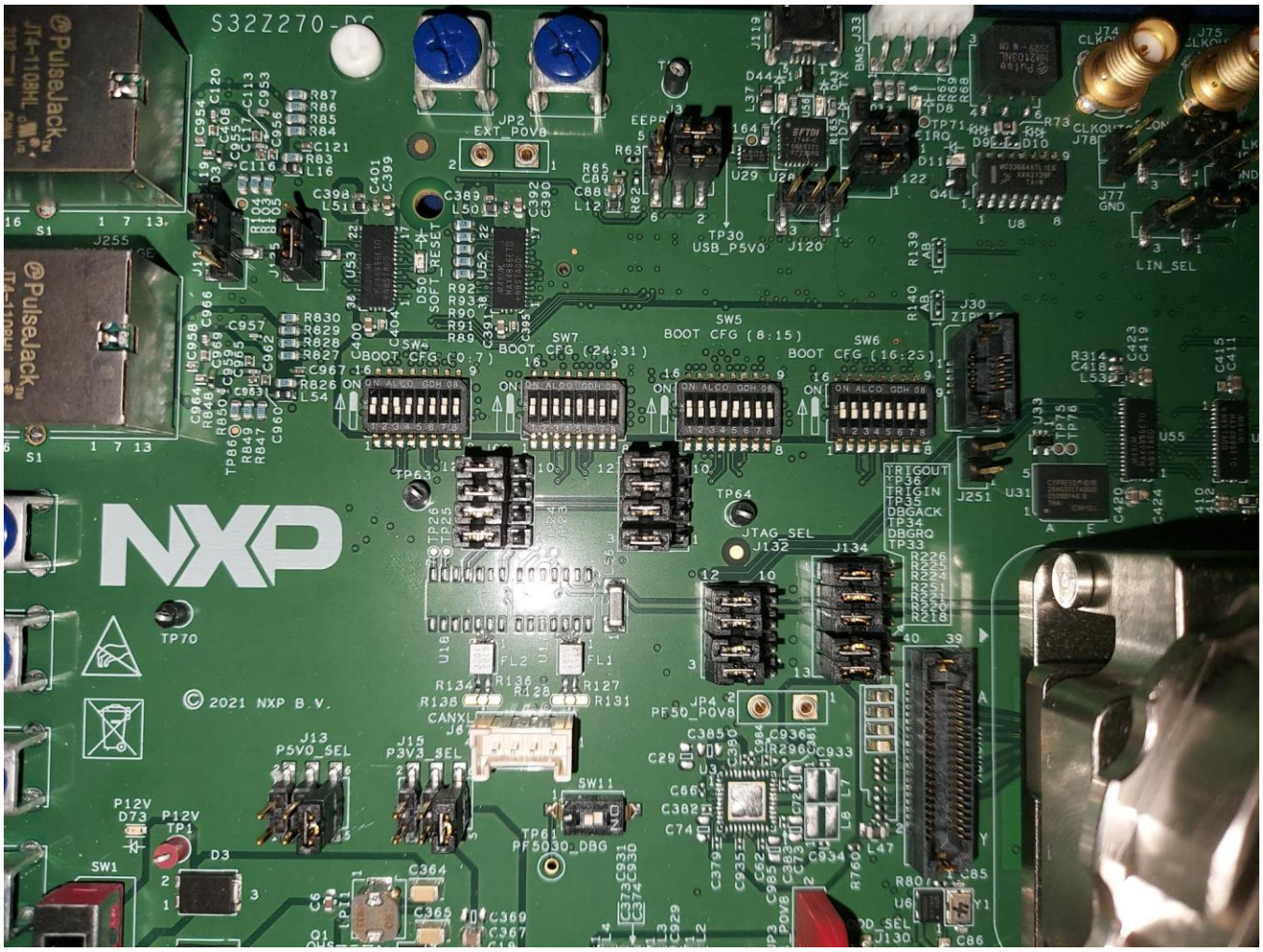
- Left button to position O (up)
- Right button to position 2 (down)

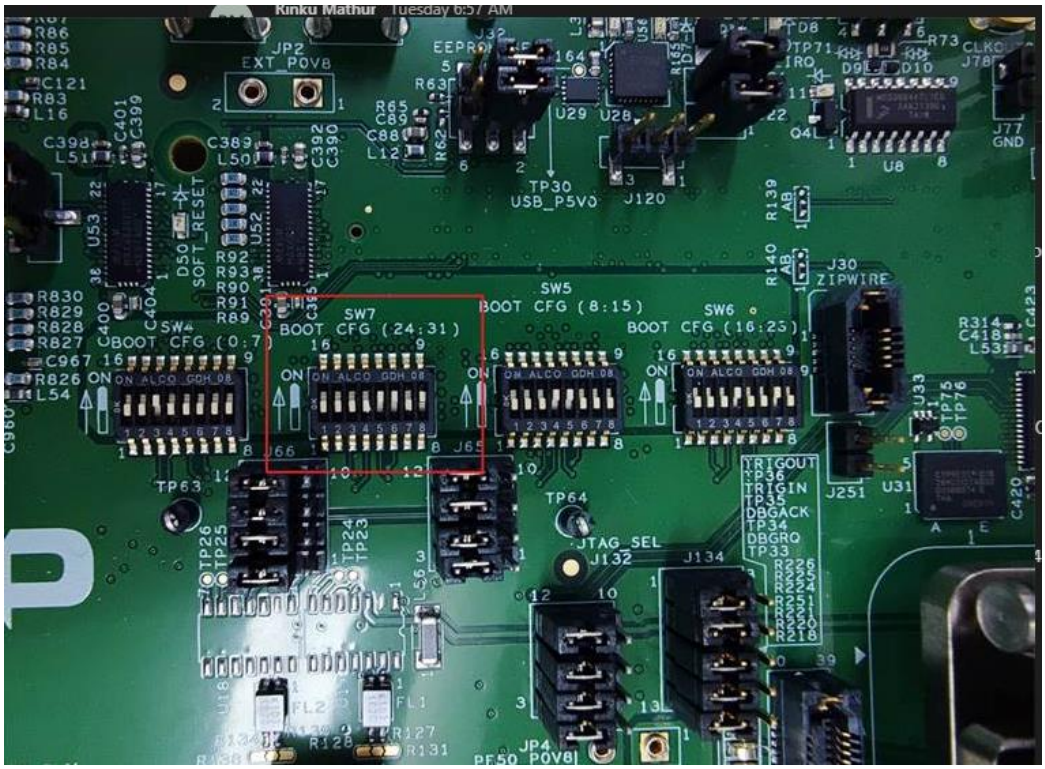


3.5. S32Z270-DC SCH-50921 REVA

Configure the S32ZE board to boot in QSPI mode. In order to do this set the following jumpers:

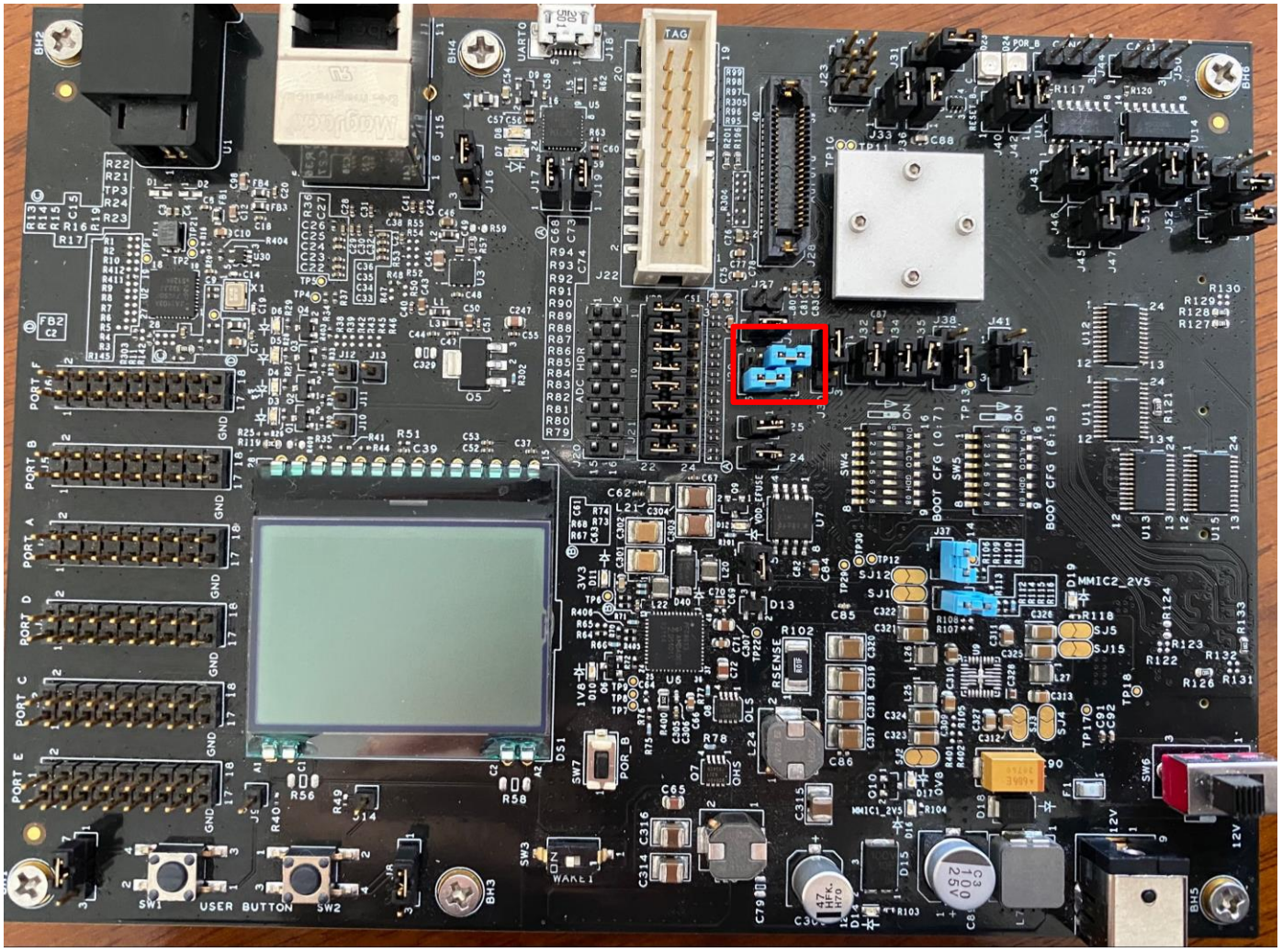
- J124: 1-2 _ J125: 2-3 _ J247: 2-3
- SW7: 5,6,8->ON, all others OFF (doesn't impact serial BOOT)




Kinku Mathur Tuesday 6:57 AM
JP2
EXT P0V8
SW5
BOOT CFG (8:15)
SW6
BOOT CFG (16:23)
SW7
BOOT CFG (24:31)
SW4
CBOOT CFG (0:7)
TP30
USB_P5V0
J120
J30
ZIPWIRE
TP63
TP64
JTAG_SEL
J132
J134
J251
U31
U33
TRIGOUT
TRIGIN
DBGACK
DBGRQ
JP4
P0V8

3.6. S32R41-EVB SCH-48194 REV B

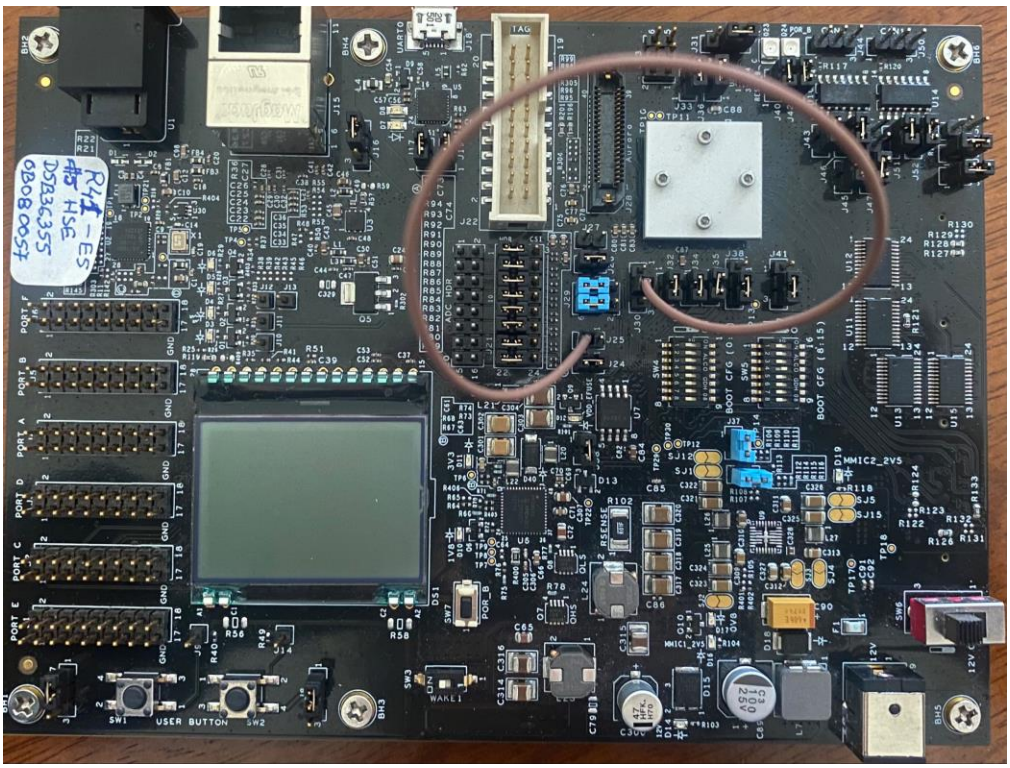
Configure the board in QSPI boot mode (J29 **1-3**, 4-6):



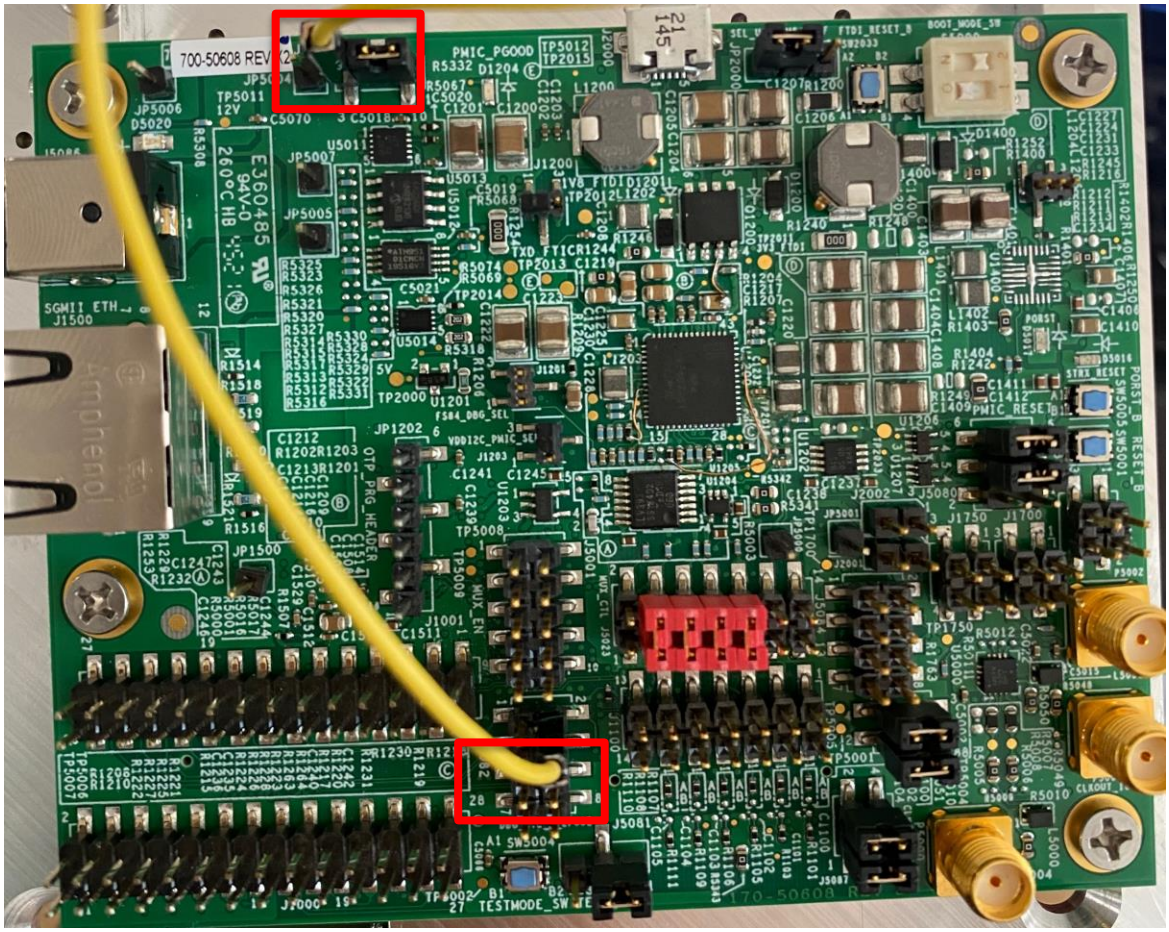
Appendix 4. Connect VDD_EFUSE pin to 1.8V

For the HSE to be able to update fuses, the VDD_EFUSE pin must be tied to 1.8V instead of GND:

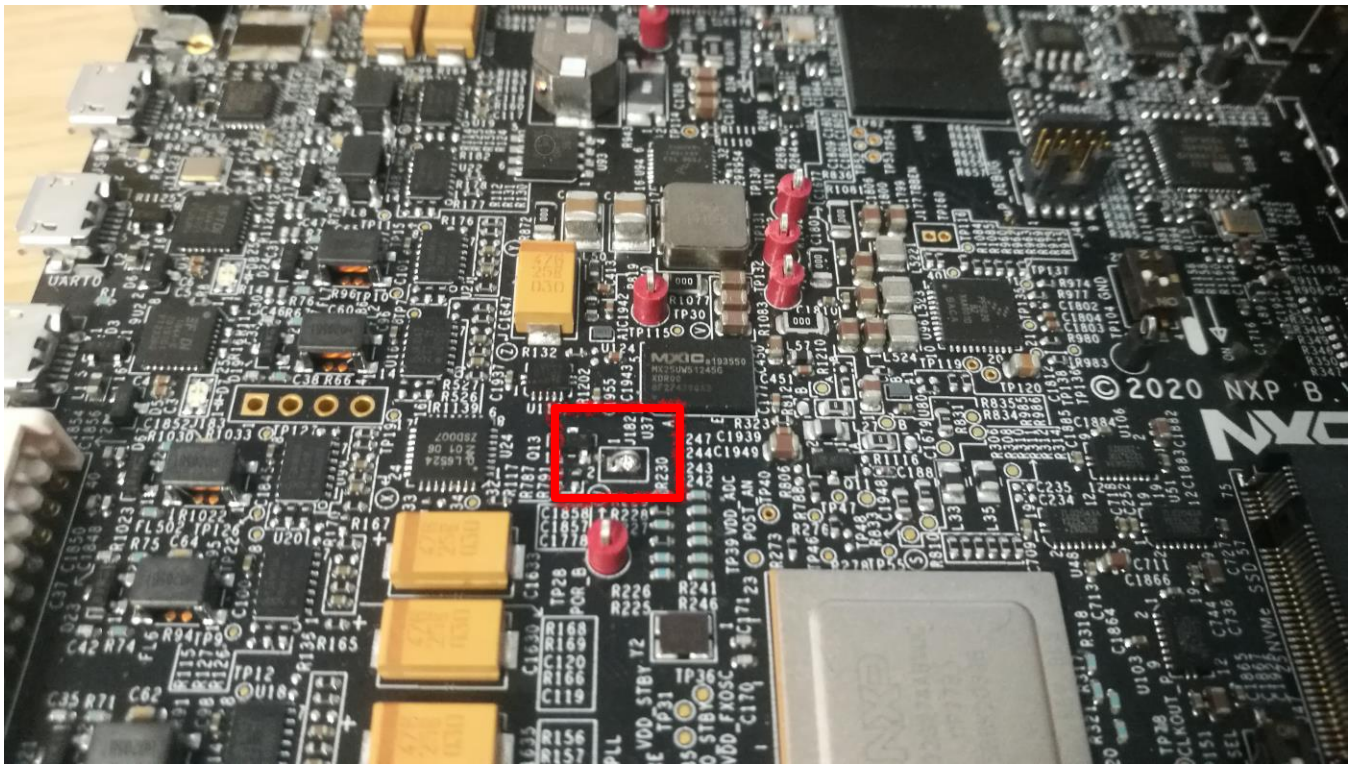
- On S32G-PROCEVB-S boards, J161 must be positioned on 1-2 (default);
- On S32R45-PROC-S boards, J180 must be positioned on 1-2 **(non-default)**.
- On S32R41-EVB boards, J30 should be opened, next from J25 closed to J30-2.



- On X-STRX-SKT_WG_V4 boards, J5082 must be connected to 1.8V:



- On S32G-RDB2 boards this is **non-default**. The alternatives are:
  - SW – either
    1. Configure the VDD_EFUSE configuration word in the IVT. This is used by HSE as start-up, only when required, to connect the VDD_EFUSE to 1.8V. This is the recommended method as most of the time the VDD_EFUSE would be connected to GND.
    2. Use *images/dcd_set_gpio25_and_init_sram.bin* DCD image to power it on at start-up by programming the GPIO25 pin to 1 (details on configuration in *S32 DS* – Generate a DCD image for powering on the VDD_EFUSE and system RAM initialization).
  - HW - tie the EFUSE_VDD to 1.8V by a strap on **J182**, as shown below:



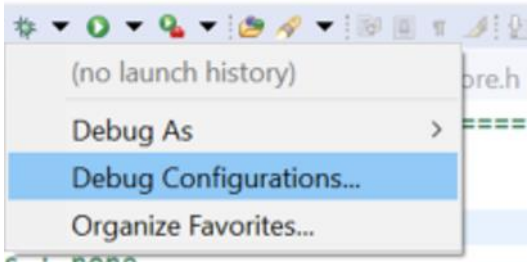
Appendix 5. S32 DS – Editing debug configurations

The S32 Design Studio projects include debug configurations for flashing the blob image, loading the symbols of the flashed application after boot or loading an ELF to RAM, using an S32 Debug Probe. For each of these use cases, two debug configurations are provided: one for each of the two build configurations, Debug_RAM and Release_RAM.

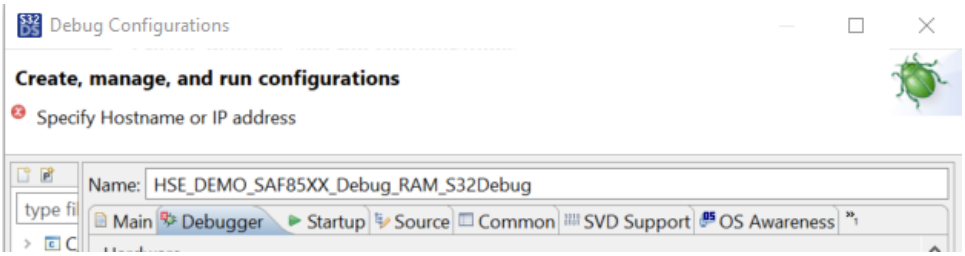
- NOTE: As our scenarios sometimes require debugging another application image than the current project binary, none of the debug configurations will build the project automatically before launch. The Debug_RAM/Release_RAM separation is necessary only to avoid any bias in the default ELF path.

Before using any of these configurations, it needs to be customized for your setup:

1) Connect the S32 Debug Probe to its host through either USB or Ethernet and, in the latter case, perform the necessary network configuration as described in the S32 Debug Probe User Guide;
2) In S32 Design Studio, open the *Debug Configurations* modal dialog, by either of the following methods:
   a. from the *Project Explorer* view, open the project context menu and select *Debug As...* -> *Debug Configurations...*;
   b. from the toolbar, expand the drop-down list of the *Debug As...* button and choose *Debug Configurations...*:



3) In the left pane, expand the *S32 Debugger* interface for the **_Load_symbols* and **_Load_ELF* configurations, or the *S32 Debugger Flash Programmer* interface for the **_Flash_blob_image* configuration. Select the debug configuration which matches the active build configuration and scenario. To be able to start a debug session, any configuration errors must be fixed. In the right pane, these are signaled with an *"X"* icon for the tabs with wrong settings and the error message is shown in the upper part of the window:



4) Go to the *Debugger* tab and update the *Debug Probe Connection* fields accordingly;
5) If necessary, update the C/C++ Application ELF file path in the *Main* tab;
6) Press *Apply* to save your changes. Once all settings are valid and the target is connected via JTAG cable and powered on, you can press the *Debug* button to start the debug session. This is also configured to switch the perspective to *Debug*. For the **_Debug_RAM_** configurations, debug sessions can also be started directly from the *Debug As...* drop-down list.

More information about working with debug configurations can be found in the *"Debugging"* chapter of the S32 Design Studio User Guide, available in *<S32DS_install_dir>\S32DS\help\pdf\S32DS_User_Guide.pdf*.

Appendix 6. GHS - Generating the binary file of the application

NOTE: Both compilation and IVT configuration using the python script are incorporated in *install_hse.bat*.

1) Open the demo_app.gpj GHS project and select the configuration from the right side;
2) Build > Build Top project demo_app.gpj;

The alternative for steps 1 and 2 in a command prompt:

```
set GHS_DIR=<path_to_ghs_dir>
%GHS_DIR%\gbuild.exe –top projects\GHS\demo_app.gpj -cfg=Debug
```

3) Make a copy of the ELF in images (will be used for loading the symbols of the flashed app):

```
cp projects\GHS\output\%PLATFORFM%\bin\demo_app.elf  images\%PLATFORM%\demo_app.elf
```

4) Generate the binary file containing .code, and .data and sections.

```
set GHS_DIR=<path_to_ghs_dir>
%GHS_DIR%\gmemfile.exe images\%PLATFORFM%\demo_app.elf -start ._ram_code_start -end ._ram_code_end –
o images\%PLATFORFM%\demo_app.bin
```

Appendix 7. Python - IVT configuration and blob image generation

NOTE: Both compilation and IVT configuration using the python script are incorporated in *install_hse.bat*.

Generate the blob image using *scripts\ivt_tool\blob.py* script:

```
set PLATFORM=SAF85XX
cd demo_app\scripts\ivt_tool
python3 blob.py -o new -j json\%PLATFORM%\new_blob.json -b ..\images\%PLATFORM%\blob.bin
```

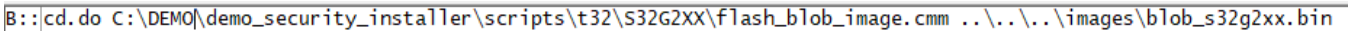
This will generate a new blob named *blob.bin*, based on the configuration specified in *%PLATFORM%\new_blob.json.* For additional details on how to configure the JSON file and supported options checkout *## blob.py* section in **scripts\Readme.md** file.

Appendix 8. TRACE32 - Flashing the blob and booting from QSPI

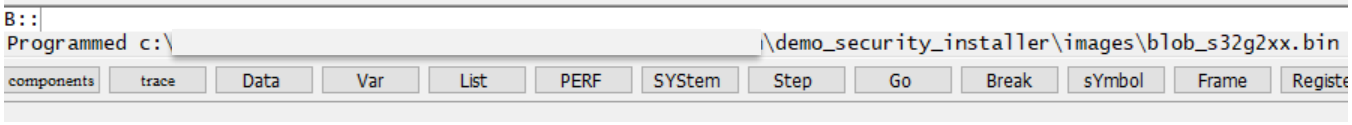
1) Configure the VDD_EFUSE jumper as described in Appendix 4;
2) (Optional) Configure the board to boot from serial interface as described in Appendix 2Appendix 3
3) Open a T32 instance and run *scripts\t32\%PLATFORM%\flash_image.cmm* (with path to blob as parameter):

   NOTE : For SAF85 platform these steps are described in Appendix 9.

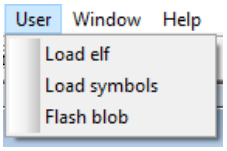
```
cd.do <absolute_path_to_scripts_t32_platform>\flash_blob_image.cmm ..\..\..\images\%PLATFORM%\blob.bin
```



- After it successfully ran, a confirmation message should appear:



- A menu is added, allowing easier running of cmm scripts to flash the blob, load the symbols of flashed application or load an ELF:



4) (optional) With the UART cable connected between board and host, configure a serial terminal: 8N1 without FC (default in PuTTY), baud rate 115200;
5) Issue a **POR**;
6) Configure the board to boot from RCON, QSPI mode as described in Appendix 3
7) (optional) Check the output on the serial terminal to confirm that the application booted correctly and the HSE is up and running:

8) Load the symbols of the flashed application (*images\%PLATFORM%\HSE_DEMO_%PLATFORM%.elf*);
9) If not already done so in step 5) using the terminal emulator, validate that the HSE FW version has been correctly retrieved from the HSE – see **Figure 9**;

```
v.w gGetVersionExample gHseFwVersion
```

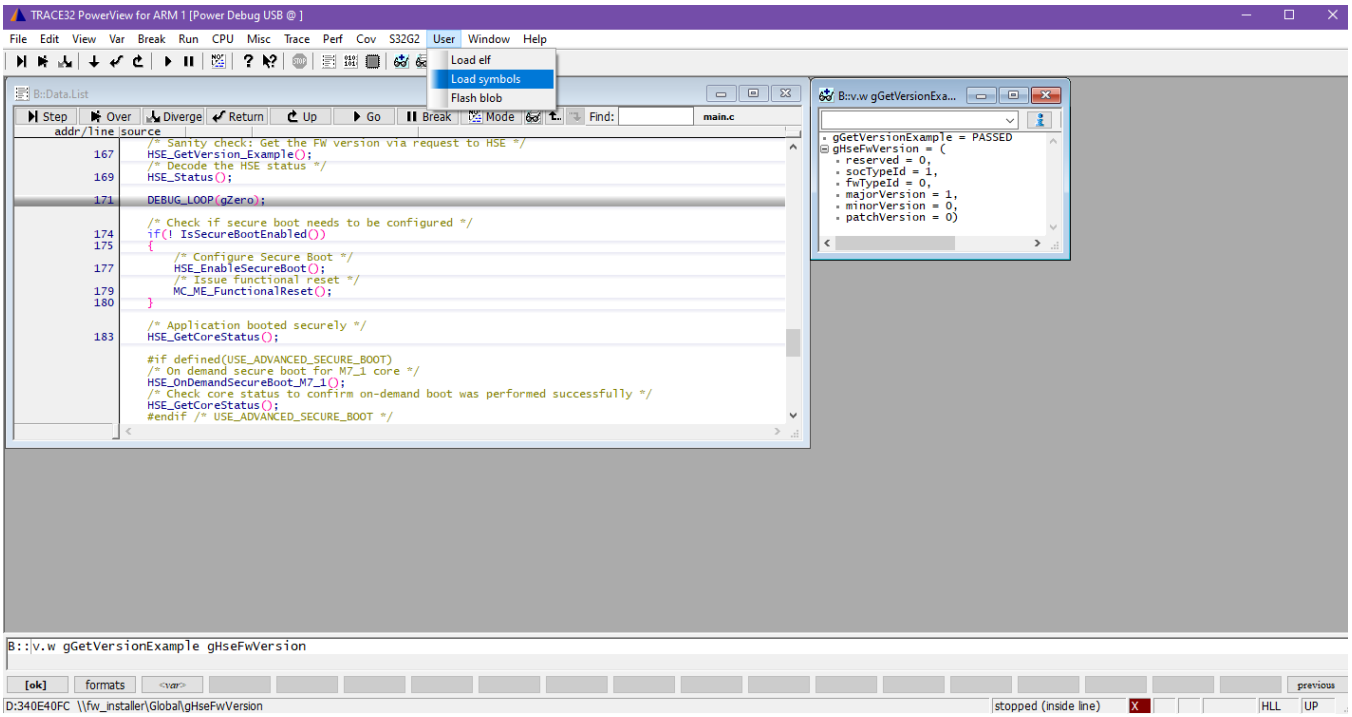


Figure 9 Loading the application symbols and confirming HSE is running

10) Skip the infinite loop by setting *gZero* to 1 and explore the examples.

```
v gZero = 1
```

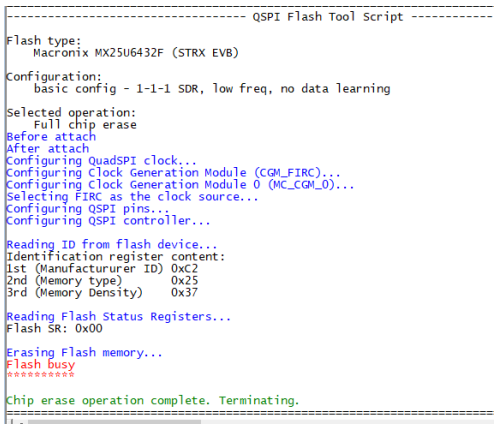
NOTE: For more information about the cmm scripts and how to configure them for easier usage checkout *## LTB scripts* section in **scripts\Readme.md** file.

Appendix 9. SAF85 – Flashing the blob image using T32 scripts

Using the **flash_blob_image.cmm** script to program the blob in the QSPI external flash, the following steps shall be taken:

1. Erase the content of the Flash.

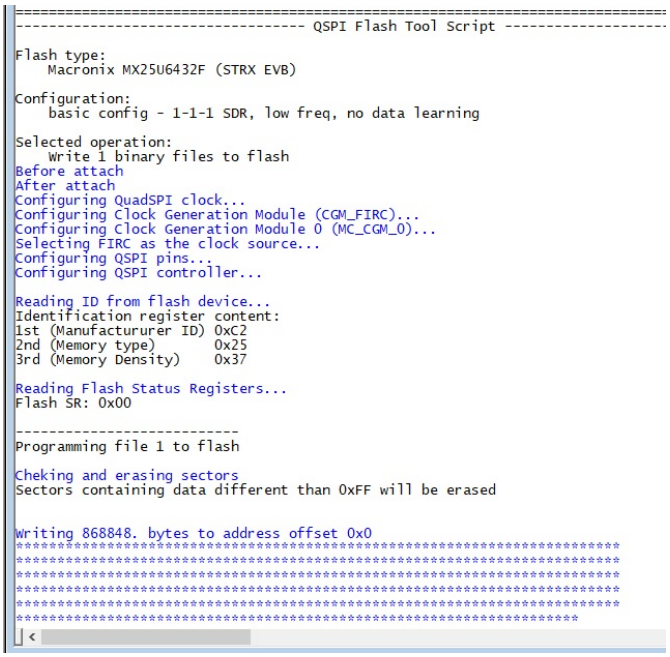
- Using a text editor, edit the line 64 of cmm script **SAF85_QSPI_flash_wrapper.cmm** from **scripts** folder to: **&function=0x4**
- Save the change and run the cmm script **SAF85_QSPI_flash_wrapper.cmm** from **scripts** folder. A window like the one below should be displayed in T32:

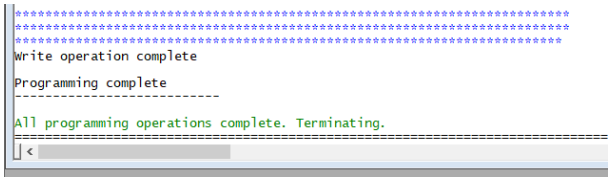


2. Program the blob containing the HSE FW pink image in Flash.

- Using a text editor, edit the line 64 of cmm script **SAF85_QSPI_flash_wrapper.cmm** from **scripts** folder to: **&function=0x1**
- Using a text editor, edit the line 83 of cmm script **SAF85_QSPI_flash_wrapper.cmm** from **scripts** folder to **&filepath1="<BLOB_PATH>"**, where <BLOB_PATH> is the absolute path where the blob resides on your machine, using backslash \ separators.
- Save the changes and run the cmm script **SAF85_QSPI_flash_wrapper.cmm** from **scripts** folder. A window like the one below should be displayed in T32 (first picture below)

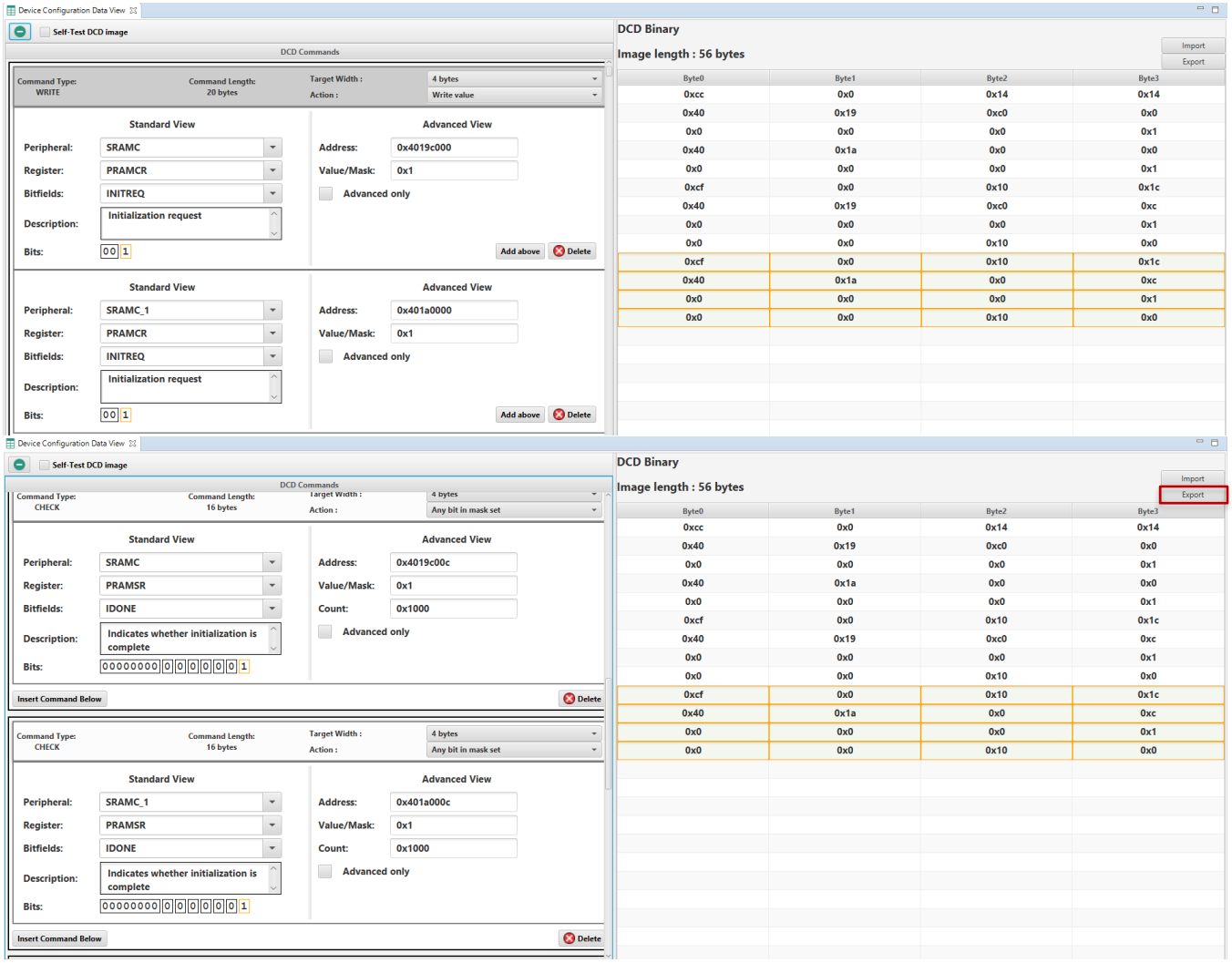
3. When the blob programming in Flash is done, a completion message is displayed (second picture below)


---------- QSPI Flash Tool Script ----------
Flash type:
Macronix MX25U6432F (STRX EVB)
Configuration:
basic config - 1-1-1 SDR, low freq, no data learning
Selected operation:
Write 1 binary files to flash
Before attach
After attach
Configuring QuadSPI clock...
Configuring Clock Generation Module (CGM_FIRC)...
Configuring Clock Generation Module 0 (MC_CGM_0)...
Selecting FIRC as the clock source...
Configuring QSPI pins...
Configuring QSPI controller...
Reading ID from flash device...
Identification register content:
1st (Manufactururer ID) 0xC2
2nd (Memory type) 0x25
3rd (Memory Density) 0x37
Reading Flash Status Registers...
Flash SR: 0x00
Programming file 1 to flash
Cheking and erasing sectors
Sectors containing data different than 0xFF will be erased
Writing 868848. bytes to address offset 0x0


Write operation complete
Programming complete
All programming operations complete. Terminating.

Appendix 10. S32 DS – Generate a DCD image for system RAM initialization

1) In Design Studio > Open the DCD configuration tool
2) Configure two write commands and two check commands:
   a. Write command for SRAMC peripheral, PRAMCR register, INITREQ bitfield
   b. Write command for SRAMC_1 peripheral, PRAMCR register, INITREQ bitfield
   c. Check command for SRAMC peripheral, PRAMSR register, IDONE bitfield, 0x1000 count, Any bit in mask set
   d. Check command for SRAMC_1 peripheral, PRAMSR register, IDONE bitfield, 0x1000 count, Any bit in mask set
3) Export the image in binary format:



Appendix 11. S32 DS – Generate a DCD image for powering on the VDD_EFUSE and system RAM initialization

1) Follow the steps in Appendix 10 for configuring the DCD with System RAM initialization commands
2) Before exporting the image, add two additional write commands:
    a. Write command for SIUL2_0 peripheral, MSCR25 register with value 0x21C000 (OBE set to 1, SRE set to 111)
    b. Write command (**target width 1 byte**) for SIUL2_0 peripheral, GPDO25 register, PDO_n bitfield set
3) Export the image in binary format (an example can be found in *images/dcd_set_gpio25_and_init_sram.bin*: