User's Guide

How to Unlock JTAG and Debug the Hardware Security Module (HSM) on Jacinto7 Security Enabled Devices with Lauterbach

ABSTRACT

This user guide describes the process for unlocking JTAG and accessing the Hardware Security Module (HSM) with Lauterbach's TRACE32™ debugger. The instructions provided in this guide apply to all Jacinto 7 Security Enabled (HS-SE) devices that contain the Security Management Subsystem Generation 2 architecture.

Table of Contents

List of Figures

List of Tables

Trademarks

TRACE32™ is a trademark of Lauterbach GmbH.
Arm® and Cortex® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
All trademarks are the property of their respective owners.

1 Introduction

The Jacinto7 Security Management Subsystem (SMS) is composed of two Arm® Cortex®-M4F cores. The primary core is referred to as the TI Foundational Security Module (TIFS) and executes TI foundational security functions. Furthermore, the secondary core is referred to as the Hardware Security Module (HSM) and is used for running customer or third party security functionality. This user guide describes the process for unlocking JTAG and accessing the HSM (Arm Cortex - M4F Secure Core 1) with Lauterbach's TRACE32 on Jacinto7 security enabled devices.

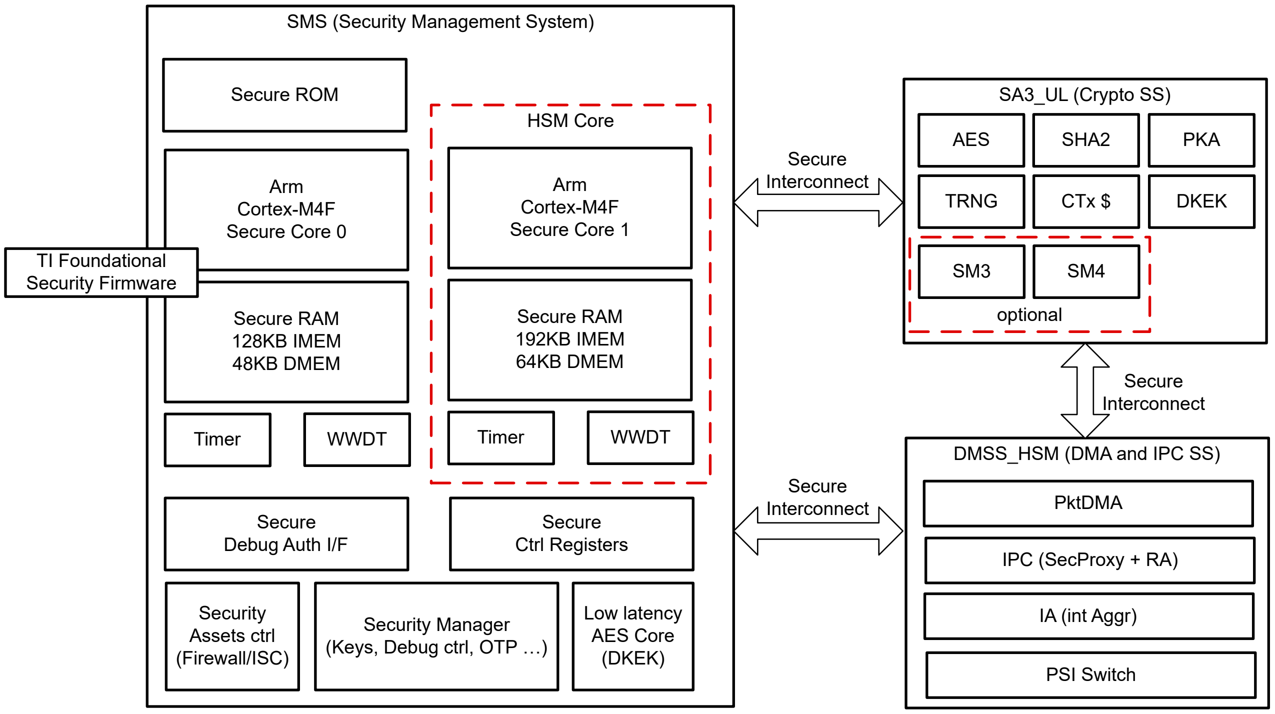

SMS (Security Management System)
Secure ROM
Arm Cortex-M4F Secure Core 0
TI Foundational Security Firmware
Secure RAM 128KB IMEM 48KB DMEM
Timer
WWDT
HSM Core
Arm Cortex-M4F Secure Core 1
Secure RAM 192KB IMEM 64KB DMEM
Timer
WWDT
Secure Debug Auth I/F
Secure Ctrl Registers
Security Assets ctrl (Firewall/ISC)
Security Manager (Keys, Debug ctrl, OTP …)
Low latency AES Core (DKEK)
Secure Interconnect
Secure Interconnect
SA3_UL (Crypto SS)
AES
SHA2
PKA
TRNG
CTx $
DKEK
SM3
SM4
optional
Secure Interconnect
DMSS_HSM (DMA and IPC SS)
PktDMA
IPC (SecProxy + RA)
IA (int Aggr)
PSI Switch

Figure 1-1. Security Management Subsystem Generation 2 Architecture

1.1 Unlocking JTAG With Jacinto7 Security Enabled Devices

Once Jacinto7 silicon transitions to a High Security – Security Enforced device (HS-SE) all security policies are enforced such as secure boot, firewalls engaged, and JTAG is locked. When JTAG access is locked on a HS-SE device there are three ways to unlock JTAG for debugging purposes:

- JTAG Unlock via the Secondary Bootloader (SBL) x509 certificate's Debug Extension
  - TISCI User Guide - Security X509 Documentation - System Firmware Debug Extension
- JTAG Unlock via TI Foundational Security (TIFS) firmware TISCI API
  - TISCI User Guide - Runtime Debug TISCI Description
- JTAG Unlock via JTAG certificate with Lauterbach TRACE32 or TI Code Composer Studio
  - TISCI User Guide - Secure AP Command Interface over JTAG

Note

The Jacinto7 ROM Loader does **not** support the unlocking of JTAG access for the HSM core. This means that is not possible to unlock JTAG with an HS-SE device using the Secondary Bootloader x509 certificate's debug extension as described in the first bullet listed above. This user guide explicitly describes how to unlock JTAG via JTAG certificate with Lauterbach TRACE32, third bullet listed above.

2 Steps to Unlock JTAG for HSM Core With TRACE32

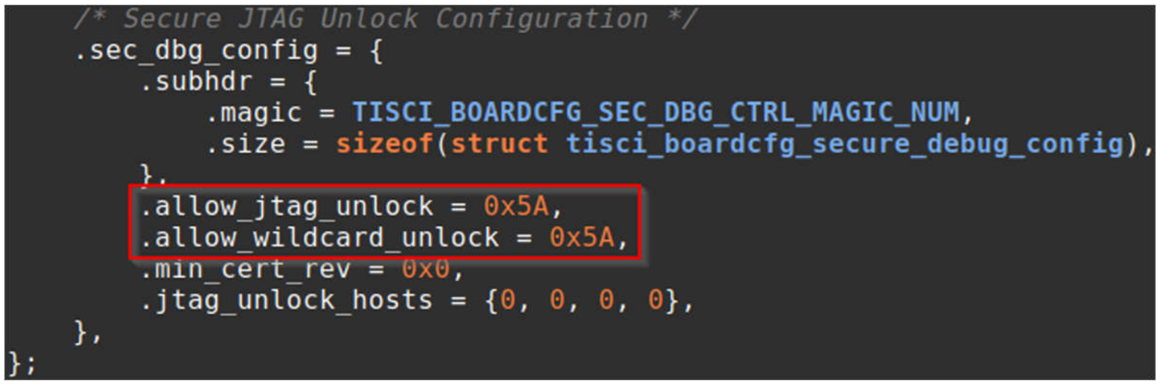
2.1 Modifying the SCI Client Default Security Board Configuration

2.1.1 PROCESSOR-SDK-RTOS

Located within your PROCESSOR-SDK-RTOS directory, *<pdk_path/ti/drv/sciclient/soc/V4/* exists a file named "*sciclient_defaultBoardcfg_security.c*". This file contains two structure elements whose values must be configured correctly in order to permit a runtime JTAG unlock to occur.

Table 2-1. Security Board Configuration Elements for JTAG Unlock

Element	Type	Description
allow_jtag_unlock	uint8_t	Must be set to 0x5A for runtime JTAG unlock to occur
allow_wildcard_unlock	uint8_t	Set to the value of 0 to enforce UID match before JTAG unlock can occur. Hence the x509 certificate must contain the UID of the device being unlocked. Set to a value of 0x5A to bypass UID match before JTAG unlock.



```
    /* Secure JTAG Unlock Configuration */
    .sec_dbg_config = {
        .subhdr = {
            .magic = TISCI_BOARDCFG_SEC_DBG_CTRL_MAGIC_NUM,
            .size = sizeof(struct tisci_boardcfg_secure_debug_config),
        },
        .allow_jtag_unlock = 0x5A,
        .allow_wildcard_unlock = 0x5A,
        .min_cert_rev = 0x0,
        .jtag_unlock_hosts = {0, 0, 0, 0},
    },
};
```

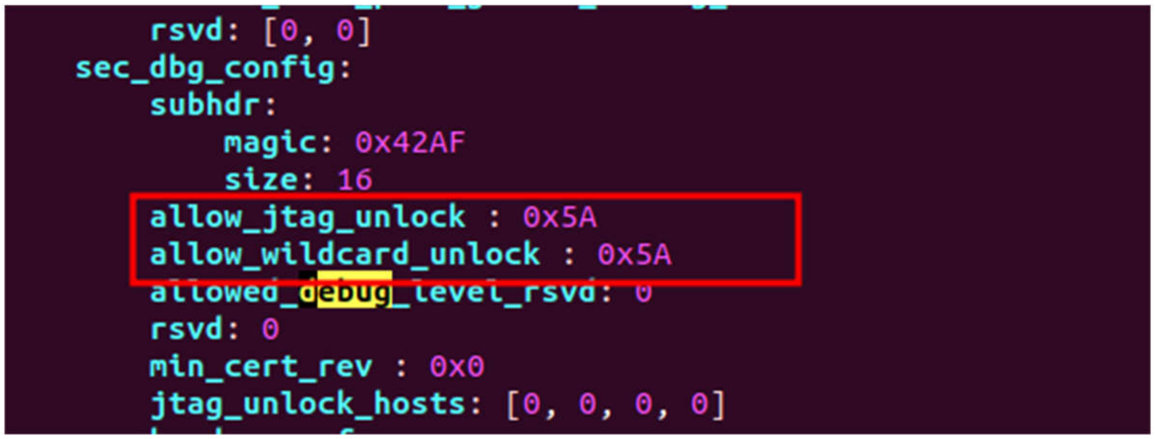
Figure 2-1. Secure Debug Structure – Example Configuration

In Figure 2-1, "*.allow_jtag_unlock = 0x5A*" is set to permit runtime JTAG unlock to occur and "*.allow_wildcard_unlock = 0x5A*", which bypasses any UID check before the JTAG unlock. For production use cases, it is recommended to set "*.allow_wildcard_unlock = 0x0*" in order to enforce the UID check. Enforcing the UID check can prevent the unlocking JTAG of different devices with the same debug certificate.

For a detailed description of all secure debug unlock elements located in the SCI Client Security Board Configuration see TISCI User Guide - Secure Debug Unlock.

2.1.2 PROCESSOR-SDK-LINUX

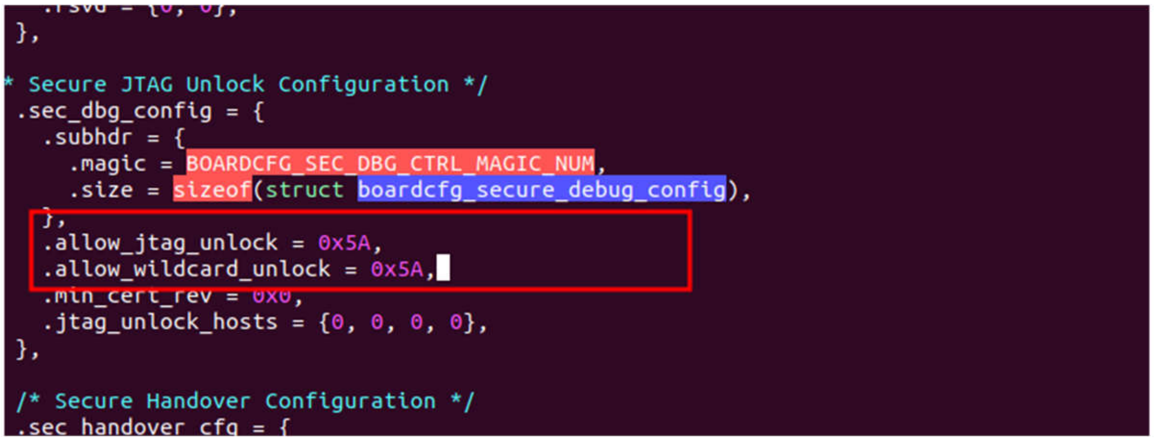
In the PROCESSOR-SDK-LINUX version 9.0 or less you can find the security board configuration file under *<linux sdk>/k3-image-gen-xxxx.xx/soc/<soc_name>/evm/sec-cfg.c.*



```
        rsvd: [0, 0]
    sec_dbg_config:
        subhdr:
            magic: 0x42AF
            size: 16
        allow_jtag_unlock : 0x5A
        allow_wildcard_unlock : 0x5A
        allowed_debug_level_rsvd: 0
        rsvd: 0
        min_cert_rev : 0x0
        jtag_unlock_hosts: [0, 0, 0, 0]
```

Figure 2-2. (SPL < 9.0 SDK) Secure Debug Structure – Example Configuration

In PROCESSOR-SDK-LINUX version 9.0 or greater the security board configuration file can be find under <u-boot-xxxx.xx+x>/board/ti/<soc_name>/sec-cfg.yaml.



```
},

* Secure JTAG Unlock Configuration */
.sec_dbg_config = {
  .subhdr = {
    .magic = BOARDCFG_SEC_DBG_CTRL_MAGIC_NUM,
    .size = sizeof(struct boardcfg_secure_debug_config),
  },
  .allow_jtag_unlock = 0x5A,
  .allow_wildcard_unlock = 0x5A,
  .min_cert_rev = 0x0,
  .jtag_unlock_hosts = {0, 0, 0, 0},
},

/* Secure Handover Configuration */
.sec_handover_cfg = {
```

Figure 2-3. (SPL 9.0+ SDK) Secure Debug Structure – Example Configuration

2.2 Building the SCI Client Security Board Configuration

Once the Security Board Configuration structure has been modified, the next step is to build the SCI Client Board Configuration. If you chose to boot with the Secondary Bootloader (SBL), continue to read section "*PROCESSOR-SDK-RTOS*" that describes how to perform this build. Otherwise, if you choose to boot with the Secondary Program Loader (SPL) continue to read section "*PROCESSOR-SDK-LINUX*".

2.2.1 PROCESSOR-SDK-RTOS

In order to build the SCI Client Board Configuration within the PROCESSOR-SDK-RTOS please navigate to the <pdk_path>/packages/ti/build/ directory and open a command prompt. Next, the following make command should be executed in order to independently build the SCI Client Board Configurations:

```
make sciclient_boardcfg_hs BOARD=j721s2_evm CORE=mcu1_0 OS=linux
```

The make command above will compile, encrypt, and sign the SCI Client Security Board Configuration with the default MEK and MPK development keys located in the PROCESSOR-SDK-RTOS. The final output from this build command will be a C array located within <pdk_path>/packages/ti/drv/sciclient/soc/V4/ directory.

2.2.2 PROCESSOR-SDK-LINUX

In order to build the SCI Client Board Configuration with the SPL, navigate to the root of the U-boot tree within the PROCESSOR-SDK-LINUX. The root of the U-Boot tree is the top-level directory and can be identified by looking for the “MAINTAINTERS” file. For *PROCESSOR-SDK-LINUX version 8.6 and less* the following commands should be executed in a command prompt in order to rebuild the SPL and tiboot3.bin image:

```
export TI_SECURE_DEV_PKG=<path-to-board-support-directory>/core-secdev-k3

export PATH=$HOME/gcc-arm-9.2-2019.12-x86_64-arm-none-linux-gnueabihf/bin:$PATH

export PATH=$HOME/gcc-arm-9.2-2019.12-x86_64-aarch64-none-linux-gnu/bin:$PATH

make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- j721s2_hs_evm_r5_defconfig O=<output
directory>/r5

make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- O=<output directory>/r5

cd ../k3-image-gen-<version>

make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- SOC=j721s2 HS=1 SW_REV=1 SBL=<output
directory>/r5/spl/u-boot-spl.bin SYSFW_HS_PATH=<path to tisdk>/board-support/prebuilt-images/ti-
fs-firmware-j721s2-hs-enc.bin SYSFW_HS_INNER_CERT_PATH=<path to tisdk>/board-support/prebuilt-
images/ti-fs-firmware-j721s2-hs-cert.bin
```

As of *PROCESSOR-SDK-LINUX version 9.0* and greater, the compilation of bootloader images will no longer require different defconfigs for GP and HS devices. Hence, the same build command will generate images of GP, HS-SE, and HS-FS devices. The following commands below should be executed in a command prompt when building the SPL and tiboot3.bin image.

```
export PATH=$HOME/gcc-arm-9.2-2019.12-x86_64-arm-none-linux-gnueabihf/bin:$PATH

export PATH=$HOME/gcc-arm-9.2-2019.12-x86_64-aarch64-none-linux-gnu/bin:$PATH

make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- j721s2_evm_r5_defconfig O=<output
directory>/r5

make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- O=<output directory>/r5 BINMAN_INDIRS=<path
to tisdk>/board-support/prebuilt-images
```

Finally, the tiboot3-j721s2-hs-evm.bin file can be copied from <output directory/r5> directory to the boot partition of your SD card or programmed to a supported non-volatile boot media.

```
cp <output directory>/r5/ tiboot3-j721s2-hs-evm.bin /media/<xyz>/boot/tiboot3.bin
```

Note

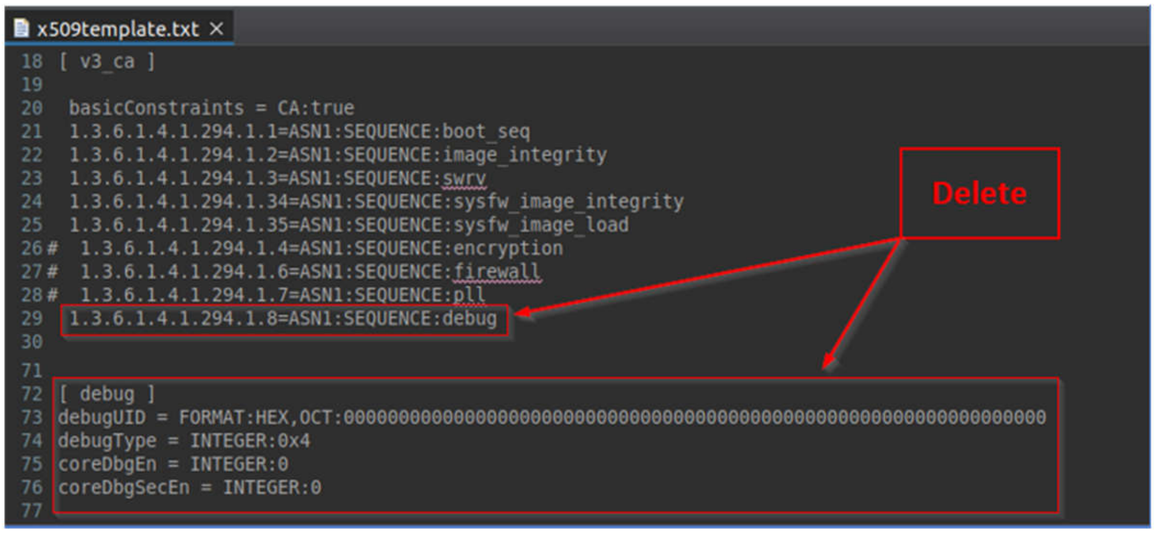
The following link below contains a detailed description of the Security Board Configuration signing procedure: TISCI User Guide - Signing Board Configuration on HS devices.

2.3 Modifying the Secondary Bootloader’s x509 Certificate

JTAG access is controlled by using a debug extension field located in the x509 certificate. By default, for an easy out of the box user experience, the PROCESSOR-SDK-RTOS enables debug via JTAG on HS-SE devices in the Secondary Bootloader’s (SBL) x509 certificate. In order to disable or change the level of JTAG access on HS-SE devices, the user must manually change the debug extension that is used in the PROCESSOR-SDK-RTOS x509 signing scripts and templates when building the SBL. Since the ROM Loader does not support the unlocking of JTAG for debugging the HSM, it is necessary for the user to delete the debug extension that is located in the SBL’s x509 certificate. The following steps describe how to delete the SBL’s debug extension in the PROCESSOR-SDK-RTOS when performing a build within a Windows or Ubuntu environment.

2.3.1 Windows Build Environment

If a user is building the SBL within a Windows environment they must navigate to the "*x509template.txt*" file that is located in the *<pdk_path>/packages/ti/build/makerules* directory and manually delete the debug extension field (see Figure 2-4.

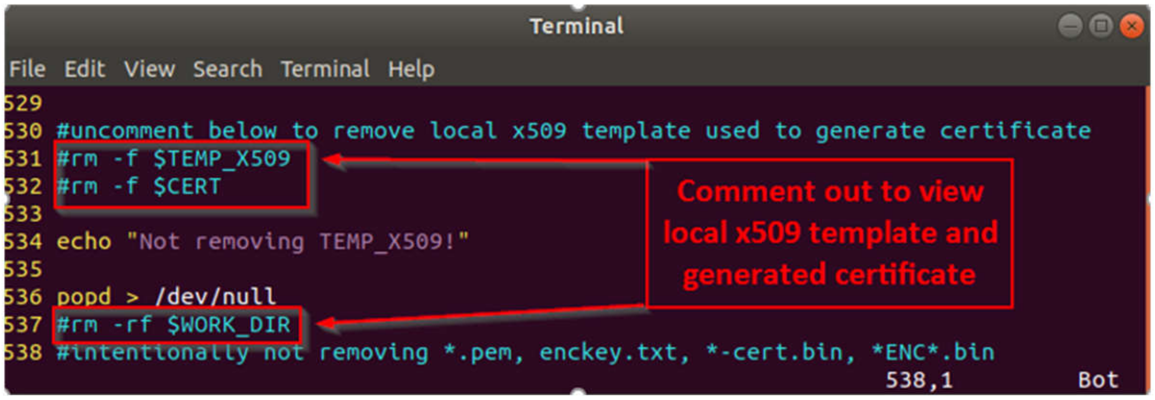


```
[ v3_ca ]

 basicConstraints = CA:true
 1.3.6.1.4.1.294.1.1=ASN1:SEQUENCE:boot_seq
 1.3.6.1.4.1.294.1.2=ASN1:SEQUENCE:image_integrity
 1.3.6.1.4.1.294.1.3=ASN1:SEQUENCE:swrv
 1.3.6.1.4.1.294.1.34=ASN1:SEQUENCE:sysfw_image_integrity
 1.3.6.1.4.1.294.1.35=ASN1:SEQUENCE:sysfw_image_load
#  1.3.6.1.4.1.294.1.4=ASN1:SEQUENCE:encryption
#  1.3.6.1.4.1.294.1.6=ASN1:SEQUENCE:firewall
#  1.3.6.1.4.1.294.1.7=ASN1:SEQUENCE:pll
 1.3.6.1.4.1.294.1.8=ASN1:SEQUENCE:debug


[ debug ]
debugUID = FORMAT:HEX,OCT:0000000000000000000000000000000000000000000000000000000000000000
debugType = INTEGER:0x4
coreDbgEn = INTEGER:0
coreDbgSecEn = INTEGER:0
```

Figure 2-4. Windows Build – x509 Template

2.3.2 Ubuntu Build Environment

If a user is building the SBL within an Ubuntu environment they must navigate to the *<pdk_path>/packages/ti/build/makerules* directory and manually edit the "*x509CertificateGen.sh"* script to remove the debug extension field from being inserted into the SBL's x509 certificate. The "*x509CertificateGen.sh"* script is automatically invoked during the make file build process and contains fields which are inserted into the SBL's x509 certificate. In order to view and prevent the local x509 template from being deleted after the build process, navigate to the bottom of the "*x509CertificateGen.sh"* script and comment out the following script removal statements (see Figure 2-5). This allows you to verify whether the modifications you made to "*x509CertificateGen.sh"* correctly deleted the debug extension.


Terminal
File Edit View Search Terminal Help
Comment out to view local x509 template and generated certificate

```
529
530 #uncomment below to remove local x509 template used to generate certificate
531 #rm -f $TEMP_X509
532 #rm -f $CERT
533
534 echo "Not removing TEMP_X509!"
535
536 popd > /dev/null
537 #rm -rf $WORK_DIR
538 #intentionally not removing *.pem, enckey.txt, *-cert.bin, *ENC*.bin
                                                          538,1        Bot
```

Figure 2-5. Ubuntu Build – x509 Certificate Generation Script

2.4 Building the Secondary Bootloader

After confirming the SBL's x509 certificate no longer contains the debug extension for JTAG unlock, the next step is to build the SBL within the PROCESSOR-SDK-RTOS. In order to do so please navigate to the *<pdk_path>/packages/ti/build/* directory and open a command prompt. Next, the following make commands should be executed in sequential order without any build errors:

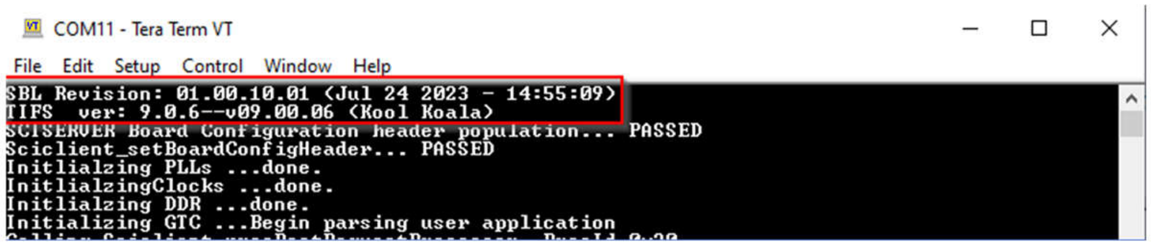
```
make sciclient_boardcfg_hs BOARD=j721s2_evm CORE=mcu1_0 OS=linux

make pdk_libs BOARD=j721s2_evm CORE=mcu1_0 OS=linux

make sbl_mmcsd_img_hs BOARD=j721s2_evm CORE=mcu1_0 OS=linux

cp packages/ti/boot/sbl/binary/j721s2_evm_hs/mmcsd/bin/sbl_mmcsd_img_mcu1_0_release.tiimage /media/
<xyz>/boot/tiboot3.bin
```

Note

The make command to build the sbl_*_img_hs will change pending the boot media you are using. The SBL make command example above is using MMC SD Card as the boot media with Ubuntu as the build environment.

2.5 Verifying Secondary Bootloader and TIFS is Executing

Since the ROM Loader is not capable of unlocking JTAG for the HSM it is necessary to confirm the SBL is successfully booting and the TI Foundational Secure (TIFS) firmware is running. In order to validate this, copy or flash the newly built SBL image and TIFS binary to your non-volatile boot media. Next, connect to the MCU UART port on your board, power on the device, and verify with a terminal window that TIFS is running successfully (see Figure 2-6).



```
SBL Revision: 01.00.10.01 (Jul 24 2023 - 14:55:09)
TIFS  ver: 9.0.6--v09.00.06 (Kool Koala)
SCISERVER Board Configuration header population... PASSED
Sciclient_setBoardConfigHeader... PASSED
Initlialzing PLLs ...done.
InitlialzingClocks ...done.
Initlialzing DDR ...done.
Initializing GTC ...Begin parsing user application
```

Figure 2-6. Verifying SBL and TIFS Boot

2.6 Creating a Downloadable x509 Certificate With a Debug Extension

Now it is necessary to create a downloadable x509 certificate which contains an appropriately configured debug extension. An x509 configuration template does exist online and it's corresponding hyperlink is located at the end of this section. A x509 configuration template does exist online located in the following link: TISCI User Guide - X509 Configuration Template.

For reference, an x509 certificate's fields have been explicitly configured for the unlocking of the HSM below:

```
[ req ]
distinguished_name    = req_distinguished_name
x509_extensions       = v3_ca
prompt                = no
dirstring_type        = nobmp

[ req_distinguished_name ]
C                     = US
ST                    = SC
L                     = Dallas
O                     = Texas Instruments., Inc.
OU                    = PBU
CN                    = Albert
emailAddress          = Albert@ti.com

[ v3_ca ]
basicConstraints = CA:true
```

```
1.3.6.1.4.1.294.1.3=ASN1:SEQUENCE:swrv
1.3.6.1.4.1.294.1.8=ASN1:SEQUENCE:debug

[ swrv ]
swrv= INTEGER:0

[ debug ]
debugUID  = FORMAT:HEX,OCT:0000
debugType = INTEGER:5
coreDbgEn = INTEGER:0x010206070809
coreDbgSecEn = INTEGER:0x202180
```

After creating the “*cert.txt”* file with the appropriate x509 field for JTAG unlock as described above it is now necessary to execute the following OpenSSL signing script:

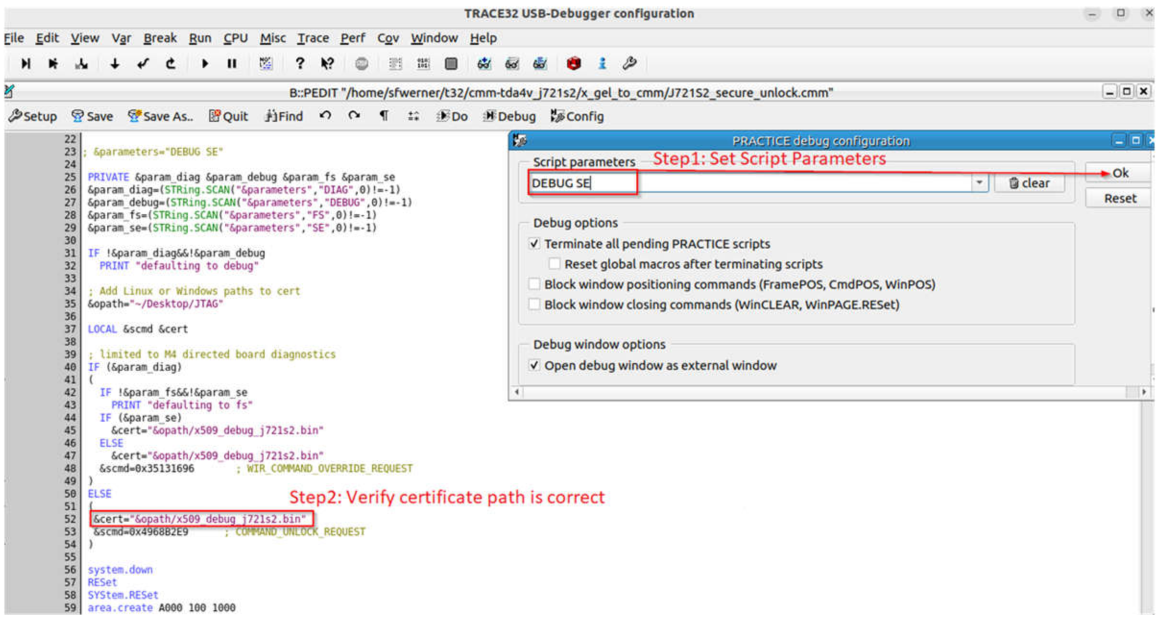
```
openssl req -new -x509 -key  k3_dev_mpk.pem -nodes -outform der -out cert.bin -config cert.txt
-sha512
```

Note

Signing the x509 certificate with OpenSSL must be done with “k3_dev_mpk.pem” key for J7 HS-DK devices. The “k3_dev_mpk.pem” key is located in the following directory: *<pdk_path>/packages/ti/build/makerules*.

2.7 Execution of TRACE32 Unlock Script

After the x509 certificate binary has been successfully created and signed with OpenSSL, a TRACE32 script can be used to perform the HSM JTAG unlock procedure. First, please contact your local TI FAE for the custom Jacinto7 TRACE32 script package. After obtaining the TI script package, navigate to the following folder location *~/t32/cmm-ti/cmm-dra/cmm-tda4v_j721s2/x_gel_to_cmm/* and set the following parameters located in the *“J721S2_secure_unlock.cmm”* script.


TRACE32 USB-Debugger configuration
File Edit View Var Break Run CPU Misc Trace Perf Cov Window Help
Setup Save Save As.. Quit Find Do Debug Config
PRACTICE debug configuration
Step1: Set Script Parameters
Step2: Verify certificate path is correct
Ok
Reset
clear

B::PEDIT "/home/sfwerner/t32/cmm-tda4v_j721s2/x_gel_to_cmm/J721S2_secure_unlock.cmm"

```
; &parameters="DEBUG SE"

PRIVATE &param_diag &param_debug &param_fs &param_se
&param_diag=(STRing.SCAN("&parameters","DIAG",0)!=-1)
&param_debug=(STRing.SCAN("&parameters","DEBUG",0)!=-1)
&param_fs=(STRing.SCAN("&parameters","FS",0)!=-1)
&param_se=(STRing.SCAN("&parameters","SE",0)!=-1)

IF !&param_diag&&!&param_debug
  PRINT "defaulting to debug"

; Add Linux or Windows paths to cert
&opath="~/Desktop/JTAG"

LOCAL &scmd &cert

; limited to M4 directed board diagnostics
IF (&param_diag)
(
  IF !&param_fs&&!&param_se
    PRINT "defaulting to fs"
  IF (&param_se)
    &cert="&opath/x509_debug_j721s2.bin"
  ELSE
    &cert="&opath/x509_debug_j721s2.bin"
  &scmd=0x35131696        ; WIR_COMMAND_OVERRIDE_REQUEST
)
ELSE
(
 &cert="&opath/x509_debug_j721s2.bin"
 &scmd=0x4968B2E9        ; COMMAND_UNLOCK_REQUEST
)

system.down
RESet
SYStem.RESet
area.create A000 100 1000
```

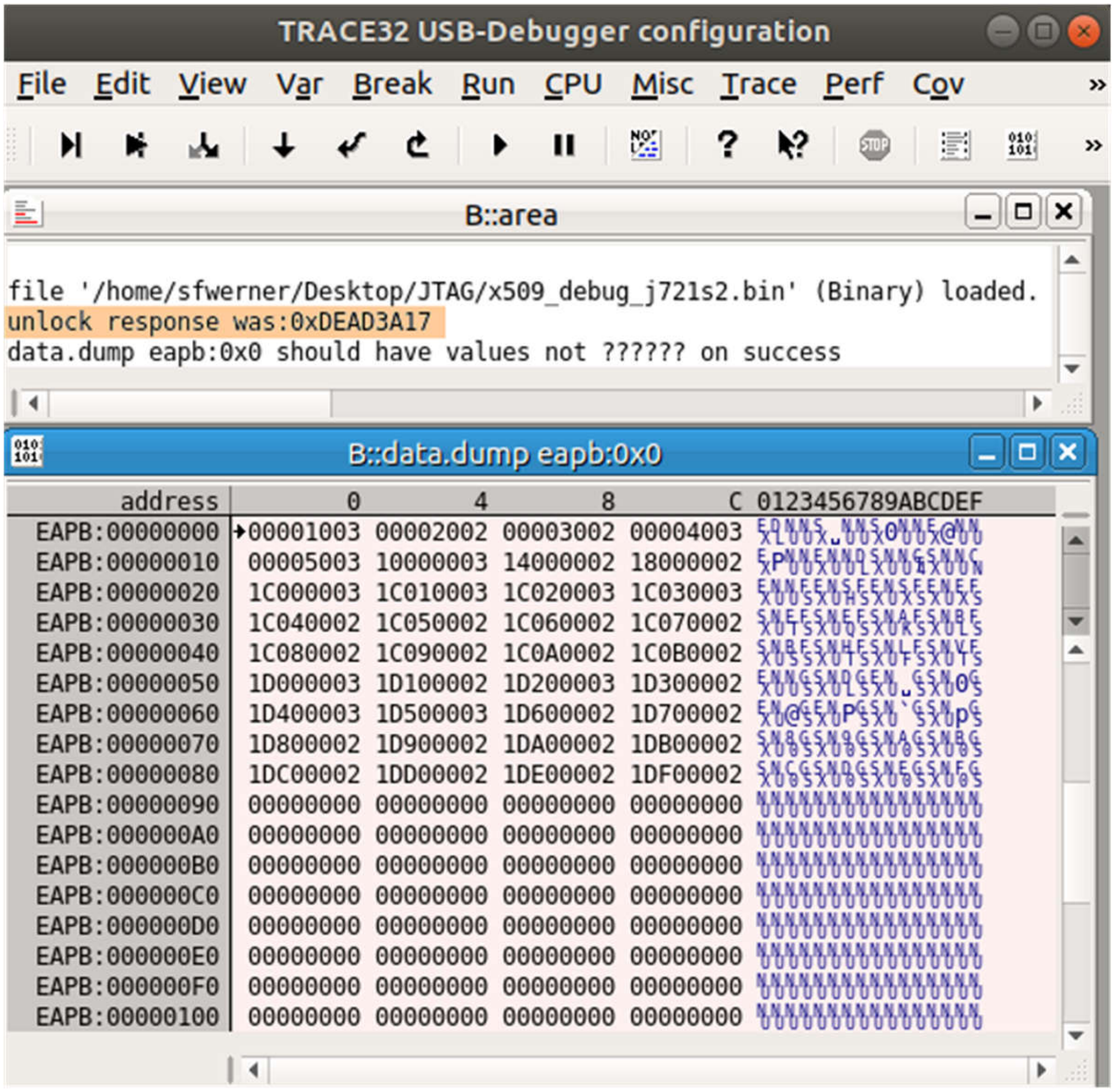
Script parameters: DEBUG SE

Debug options
- [✓] Terminate all pending PRACTICE scripts
- [ ] Reset global macros after terminating scripts
- [ ] Block window positioning commands (FramePOS, CmdPOS, WinPOS)
- [ ] Block window closing commands (WinCLEAR, WinPAGE.RESet)

Debug window options
- [✓] Open debug window as external window

Figure 2-7. TRACE32 – Secure JTAG Unlock Script

After modifying the *“J721S2_secure_unlock.cmm”* script, save your changes, power on your Jacinto7 device, and then execute the script with TRACE32 using the “Do” command. After successfully executing the script and JTAG unlock sequence, you should see the following string appear within the TRACE32 area window: “***unlock response was : 0xDEAD3A17***” (see Figure 2-8).


TRACE32 USB-Debugger configuration
File Edit View Var Break Run CPU Misc Trace Perf Cov
B::area
B::data.dump eapb:0x0

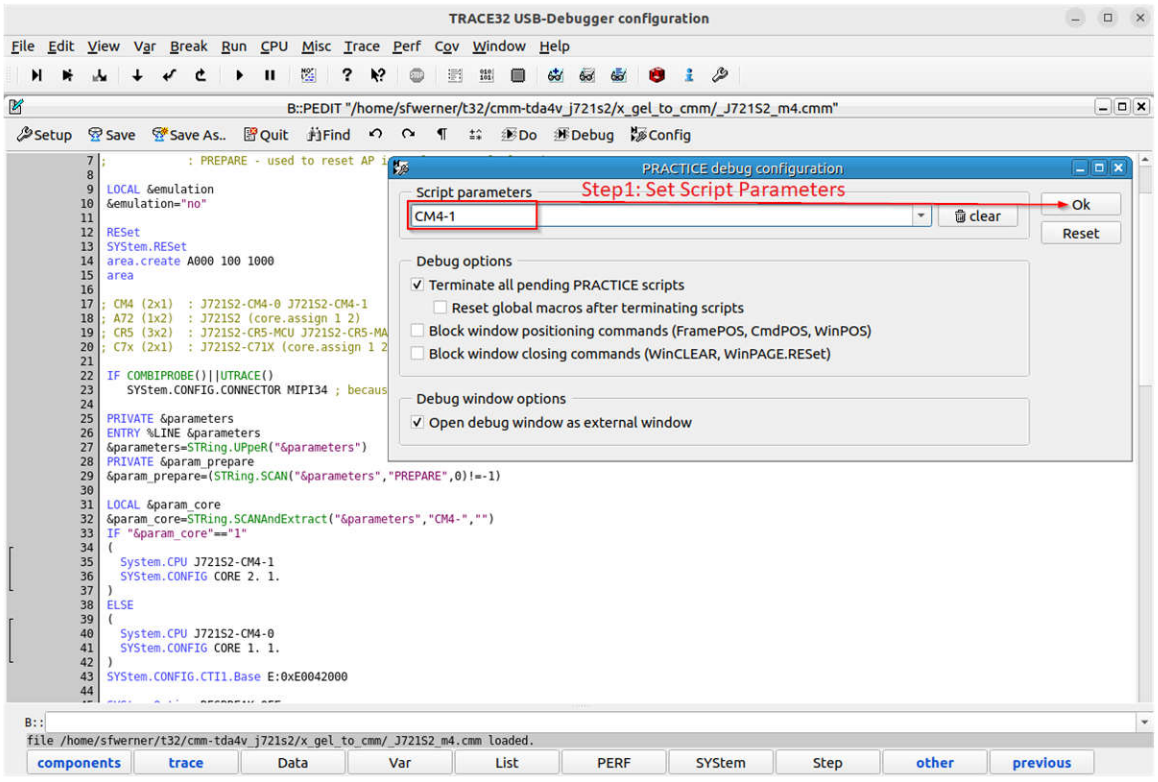
```
file '/home/sfwerner/Desktop/JTAG/x509_debug_j721s2.bin' (Binary) loaded.
unlock response was:0xDEAD3A17
data.dump eapb:0x0 should have values not ?????? on success
```

```
        address          0        4        8        C 0123456789ABCDEF
EAPB:00000000 ►00001003 00002002 00003002 00004003
EAPB:00000010  00005003 10000003 14000002 18000002
EAPB:00000020  1C000003 1C010003 1C020003 1C030003
EAPB:00000030  1C040002 1C050002 1C060002 1C070002
EAPB:00000040  1C080002 1C090002 1C0A0002 1C0B0002
EAPB:00000050  1D000003 1D100002 1D200003 1D300002
EAPB:00000060  1D400003 1D500003 1D600002 1D700002
EAPB:00000070  1D800002 1D900002 1DA00002 1DB00002
EAPB:00000080  1DC00002 1DD00002 1DE00002 1DF00002
EAPB:00000090  00000000 00000000 00000000 00000000
EAPB:000000A0  00000000 00000000 00000000 00000000
EAPB:000000B0  00000000 00000000 00000000 00000000
EAPB:000000C0  00000000 00000000 00000000 00000000
EAPB:000000D0  00000000 00000000 00000000 00000000
EAPB:000000E0  00000000 00000000 00000000 00000000
EAPB:000000F0  00000000 00000000 00000000 00000000
EAPB:00000100  00000000 00000000 00000000 00000000
```

Figure 2-8. TRACE32 – Successful Secure JTAG Unlock Response

2.8 Attaching to HSM Core With TRACE32

After successfully executing and unlocking JTAG access for the HSM with the TRACE32 script *"J721S2_secure_unlock.cmm"*, it is now possible to attach and load HSM symbolic information within TRACE32 for debugging purposes. Please navigate to the following folder location *~/t32/cmm-ti/cmm-dra/cmm-tda4v_j721s2/x_gel_to_cmm/* and edit the following script parameters located in the *"_J721S2_m4.cmm"* script in order to attach to the HSM (Cortex M4F).


TRACE32 USB-Debugger configuration
File Edit View Var Break Run CPU Misc Trace Perf Cov Window Help
B::PEDIT "/home/sfwerner/t32/cmm-tda4v_j721s2/x_gel_to_cmm/_J721S2_m4.cmm"
Setup Save Save As.. Quit Find Do Debug Config
PRACTICE debug configuration
Step1: Set Script Parameters
clear
Ok
Reset
components trace Data Var List PERF SYStem Step other previous

```
;              : PREPARE - used to reset AP i

LOCAL &emulation
&emulation="no"

RESet
SYStem.RESet
area.create A000 100 1000
area

; CM4 (2x1)  : J721S2-CM4-0 J721S2-CM4-1
; A72 (1x2)  : J721S2 (core.assign 1 2)
; CR5 (3x2)  : J721S2-CR5-MCU J721S2-CR5-MA
; C7x (2x1)  : J721S2-C71X (core.assign 1 2

IF COMBIPROBE()||UTRACE()
   SYStem.CONFIG.CONNECTOR MIPI34 ; becaus

PRIVATE &parameters
ENTRY %LINE &parameters
&parameters=STRing.UPpeR("&parameters")
PRIVATE &param_prepare
&param_prepare=(STRing.SCAN("&parameters","PREPARE",0)!=-1)

LOCAL &param_core
&param_core=STRing.SCANAndExtract("&parameters","CM4-","")
IF "&param_core"=="1"
(
  System.CPU J721S2-CM4-1
  SYStem.CONFIG CORE 2. 1.
)
ELSE
(
  System.CPU J721S2-CM4-0
  SYStem.CONFIG CORE 1. 1.
)
SYStem.CONFIG.CTI1.Base E:0xE0042000
```

Script parameters: CM4-1

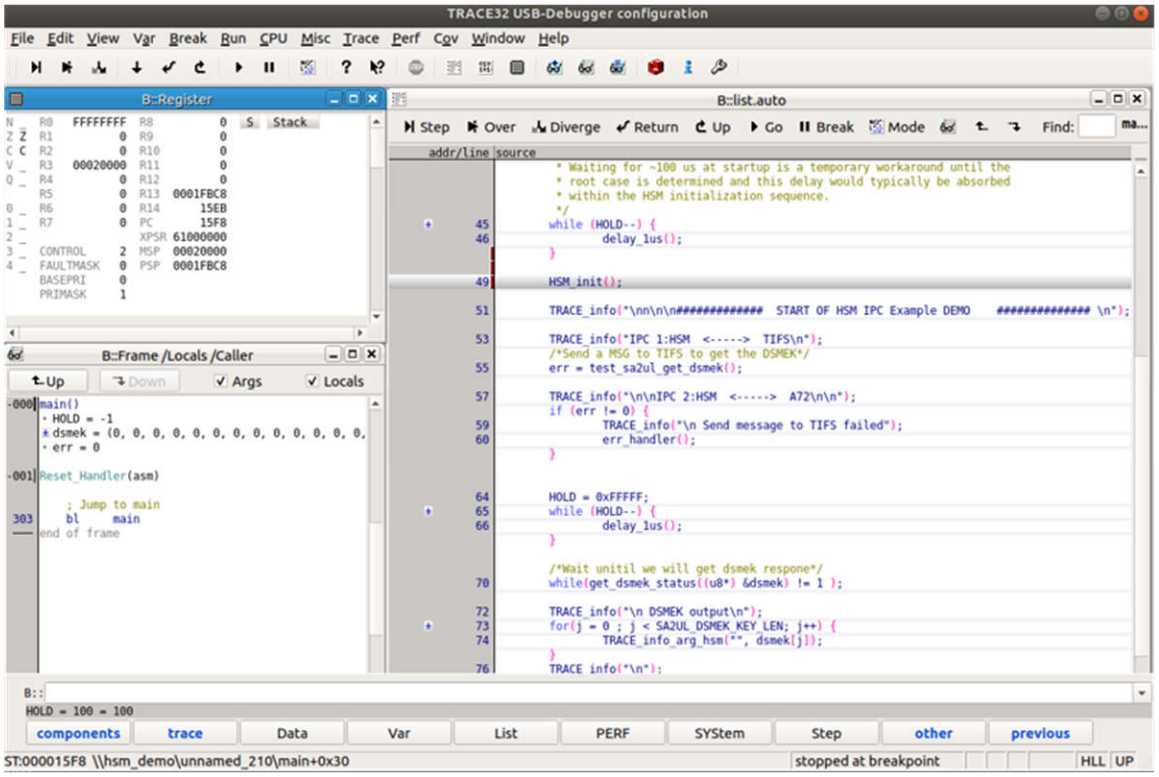
Debug options
- [✓] Terminate all pending PRACTICE scripts
- [ ] Reset global macros after terminating scripts
- [ ] Block window positioning commands (FramePOS, CmdPOS, WinPOS)
- [ ] Block window closing commands (WinCLEAR, WinPAGE.RESet)

Debug window options
- [✓] Open debug window as external window

```
B::
file /home/sfwerner/t32/cmm-tda4v_j721s2/x_gel_to_cmm/_J721S2_m4.cmm loaded.
```

Figure 2-9. TRACE32 – HSM Attach Script

After modifying "_J721S2_m4.cmm" script and saving the file, please once again click on the "Do" button in order to execute the script in TRACE32. Finally, you should now be attached to the HSM and able to view debugging windows in TRACE32 as depicted in the Figure 2-10.


TRACE32 USB-Debugger configuration
File Edit View Var Break Run CPU Misc Trace Perf Cov Window Help
B::Register
B::Frame /Locals /Caller
Up Down Args Locals
B::list.auto
Step Over Diverge Return Up Go Break Mode Find:
addr/line source
components trace Data Var List PERF SYStem Step other previous

```
N _  R0   FFFFFFFF  R8          0   S  Stack
Z Z  R1          0  R9          0
C C  R2          0  R10         0
V _  R3   00020000  R11         0
Q _  R4          0  R12         0
     R5          0  R13  0001FBC8
0 _  R6          0  R14      15EB
1 _  R7          0  PC       15F8
2 _                 XPSR 61000000
3 _  CONTROL     2  MSP  00020000
4 _  FAULTMASK   0  PSP  0001FBC8
     BASEPRI     0
     PRIMASK     1
```

```
-000 main()
      · HOLD = -1
      · dsmek = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      · err = 0

-001 Reset_Handler(asm)

         ; Jump to main
 303     bl      main
 ——  end of frame
```

```
         * Waiting for ~100 us at startup is a temporary workaround until the
         * root case is determined and this delay would typically be absorbed
         * within the HSM initialization sequence.
         */
        while (HOLD--) {
                delay_1us();
        }

        HSM_init();

        TRACE_info("\nn\n\n#############  START OF HSM IPC Example DEMO    ############## \n");

        TRACE_info("IPC 1:HSM  <----->  TIFS\n");
        /*Send a MSG to TIFS to get the DSMEK*/
        err = test_sa2ul_get_dsmek();

        TRACE_info("\n\nIPC 2:HSM  <----->  A72\n\n");
        if (err != 0) {
                TRACE_info("\n Send message to TIFS failed");
                err_handler();
        }


        HOLD = 0xFFFFF;
        while (HOLD--) {
                delay_1us();
        }

        /*Wait unitil we will get dsmek respone*/
        while(get_dsmek_status((u8*) &dsmek) != 1 );

        TRACE_info("\n DSMEK output\n");
        for(j = 0 ; j < SA2UL_DSMEK_KEY_LEN; j++) {
                TRACE_info_arg_hsm("", dsmek[j]);
        }
        TRACE_info("\n");
```

```
B::
HOLD = 100 = 100
ST:000015F8 \\hsm_demo\unnamed_210\main+0x30        stopped at breakpoint    HLL UP
```

Figure 2-10. TRACE32 – HSM Debugging