# AN4686

## Secure Debug in i.MX 6/7/8M Family of Applications Processors

Rev. 2 — 07/2020 Application Note

Contents



## 1 Introduction

The purpose of this document is to describe how to configure JTAG on the i.MX 6/7/8M family of applications processors.

The i.MX 6/7/8M family's System JTAG Controller (SJC) provides a method of regulating the JTAG access. The three JTAG security modes available on the i.MX 6/7/8M family are:

- Mode #1: No Debug. This mode provides maximum security. All security-sensitive JTAG features are permanently blocked, preventing any debug.
- Mode #2: Secure JTAG. This mode provides high security. JTAG use is regulated by secret key-based challenge/response authentication mechanism.
- Mode #3: JTAG Enabled. This mode provides low security. This is the default mode of operation for the SJC.

In addition to the three security modes, there is an option to disable the SJC functionality. These JTAG modes are configured using One Time Programmable (OTP) eFuses which are burned after packaging. The fuse burning is an irreversible process; once a fuse is burned it is not possible to change the fuse back to the unburned state. This document briefly describes Mode #1 and Mode #3, however, focusses on Mode #2, Secure JTAG. The intent of this mode is to allow return or field testing. When a secured JTAG device is returned for debugging, this mode allows authorized re-activation of the JTAG port. To use this feature the JTAG port must be pinned out and accessible in the application.

## 2 No Debug

The No Debug JTAG mode provides the highest security level. In this mode, all JTAG features are disabled, except for features that do not reduce the security level of the product and allows to perform important tests and board connectivity checks, as Boundary Scan and visibility of the power mode status bits. If SJC is disabled by the `SJC_DISABLE` fuse all JTAG features are disabled including the ones allowed in No Debug mode.

## 3 JTAG Enabled

In the JTAG Enabled mode, all JTAG features are enabled. This mode can be useful during the development process but should not be used in production devices as any access can retrieve critical information about the system.

**NOTE**

JTAG Enabled mode is similar to Field Return mode, which enables NXP to debug return suspected fault parts with debugging restrictions.

## 4 Secure JTAG

The Secure JTAG mode limits the JTAG access by using a challenge/response-based authentication mechanism. Any access to JTAG port is checked. Only authorized debug devices (devices that have the right response) can access the JTAG port; unauthorized JTAG access attempts are denied. This feature requires external debugger tools (such as Lauterbach Trace32,

Arm® RVDS/DS5 Debuggers, etc.) that support the challenge/response-based authentication mechanism. The secure JTAG mode is typically enabled during device manufacture and not on development boards, however NXP leaves it to the user to decide what works best for their environment.

## 4.1 How to put the chip in Secure JTAG mode

The i.MX 6/7/8M family's System JTAG Controller (SJC) can operate in three different security modes. By default the SJC's mode of operation is JTAG enabled (Mode #3). To switch the controller to Secure JTAG mode, the user should program a value `0x1` to the eFuse labeled `JTAG_SMODE`, described in Table 1. This eFuse has a value `0x0` by default, which puts the JTAG controller in low security mode. For further details on eFuse blowing see the Fusemap and On-Chip OTP Controller (OCOTP_CTRL) chapters in the appropriate i.MX 6/7/8M family's reference manual available at www.nxp.com.

In addition to programming the `JTAG_SMODE` eFuse, the user should program the `BOOT_CFG_LOCK` eFuses to lock and prevent further modification to the `JTAG_SMODE` eFuse. Programming these fuses will disable access to functions in addition to the JTAG Security Mode fuse bits, so users should ensure that this is programmed last, once the final fuse configuration has been decided. At a minimum, NXP recommends setting the fuse to Override Protect (OP) mode.

**Table 1. Secure JTAG eFuse configurations**

| Fuse name | Number of fuses | Fuse function | Settings | Locked by |
|---|---|---|---|---|
| `JTAG_SMODE` | 2 | JTAG security mode, controlling the security mode of the JTAG debug interface. | 00: JTAG enable mode (Default)<br>01: Secure JTAG mode<br>11: No debug mode | `BOOT_CFG_LOCK` |
| `SJC_DISABLE` | 1 | Additional JTAG mode with the highest level of JTAG protection, thereby overriding the `JTAG_SMODE` eFuses. In this mode all JTAG features are disabled including Secure JTAG and Boundary Scan. | 0: JTAG is enabled<br>1: JTAG is disabled | `BOOT_CFG_LOCK` |
| `BOOT_CFG_LOCK` | 2 | Perform lock protection on BOOT related fuses. This fuse locks numerous functions including `JTAG_SMODE`. | 00: Unlock<br>1x: Override Protect (OP)<br>x1: Write Protect (WP)<br>11: Both OP and WP | N/A |
| `SJC_RESP` | 56[1] | Response reference value for the secure JTAG controller. | — | `SJC_RESP_LOCK` (locks also for read and explicit sense) |
| `SJC_RESP_LOCK` | 1 | SJC response lock | 0: Unlock<br>1: Lock (WP, OP, RP, sense) | N/A |
| `JTAG_HEO` | 1 | JTAG HAB Enable Override.<br>Disallows HAB JTAG enabling.<br>The HAB may normally enable JTAG debugging by means of the HAB_JDE-bit in the | 0: HAB may enable JTAG debug access<br>1: HAB JTAG enable is overridden (HAB may not enable JTAG debug access) | `BOOT_CFG_LOCK` |

*Table continues on the next page...*

**Table 1. Secure JTAG eFuse configurations (continued)**

| Fuse name | Number of fuses | Fuse function | Settings | Locked by |
|---|---|---|---|---|
| | | OCOTP SCS register. The JTAG_HEO-bit can override this behavior | | |

1. For i.MX 8MPlus, the number of `SJC_RESP` fuses are 128.

### 4.1.1 eFuses used by Secure JTAG

The challenge/response mechanism used to authenticate the JTAG accesses uses a challenge value and the associated secret response key. The keys are stored in eFuses inside the IC. Listed below are the i.MX 6/7/8M family eFuses used to store the challenge value and the secret response key:

- The challenge value is the **Device Unique ID** which is programmed into the eFuses. This Device ID is unique for each IC and can be read from the OCOTP registers. These eFuses will be programmed by NXP during manufacturing.
- The secret response key (56 bits, or 128 bits in case of i.MX 8MPlus) must be programmed by the user into the eFuses marked `SJC_RESP`.

After programming the secret response key, the user must disable the ability of software running on the Arm core to read or overwrite the response key. This is done by programming a `0x1` to the associated lock eFuse `SJC_RESP_LOCK`.

The definition of the response value is left to the user, as once the response fuse field is provisioned and locked it can no longer be read by the Arm core.

### 4.1.2 Debug flow when Secure JTAG mode is enabled

When the SJC is in Secure Debug mode the authentication process is as follows:

1. JTAG shifts the challenge key through the Test Data Output (TDO) chain.
2. On the host side, the debug tool takes the challenge key as an input and generates the expected response key.
3. The associated response key is shifted back through the Test Data Input (TDI) chain.
4. The SJC compares the expected internal fused response key with the one shifted in, and enables the JTAG access only if it matches.

**NOTE**

Any reset after JTAG access authorization will shift the JTAG controller back to its lock state.

Figure 1 shows how the challenge/response mechanism works with the JTAG tools.

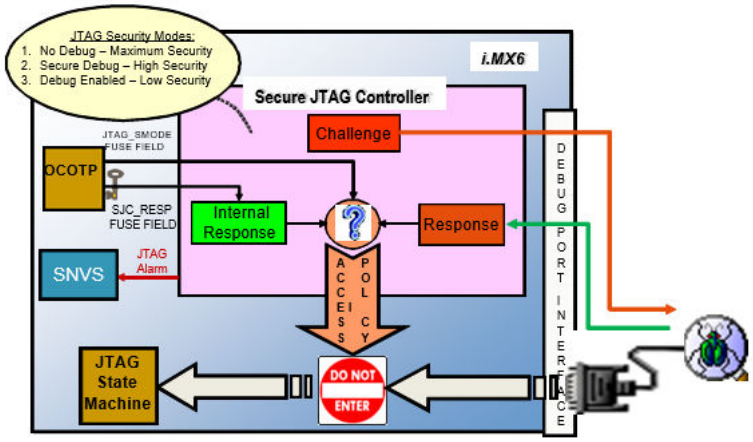


**Figure 1. Secure JTAG operation**

The JTAG debug tool passes the retrieved challenge key to the user's application and gets the associated response key in return. The management of the challenge/response pairs is user-dependent and not handled by NXP or the debug tool vendors. Key management is discussed further in Secret response key management by the user.

## 4.2 SJC disable fuse

In addition to the various JTAG security modes implemented internally in the SJC, there is an option to disable the SJC functionality with the `SJC_DISABLE` eFuse. This eFuse creates an additional JTAG mode, JTAG Disabled with the highest level of JTAG protection, overriding the `JTAG_SMODE` eFuses. In this mode all JTAG features are disabled, including Secure JTAG and Boundary Scan; users must ensure that this fuse is not blown if they wish to use the Secure JTAG functionality.

## 4.3 HAB_JDE

The Secure JTAG authentication may be bypassed in SW by writing **1** to `HAB_JDE` (HAB JTAG DEBUG ENABLE) bit in the eFuse controller module. By this JTAG is opened and it allows enabling JTAG without activating the challenge-response mechanism.

The platform initialization software should set the LOCK bit for JDE bit before transferring control to the application code to ensure that only the trusted SW can set the JDE bit. This is done by the i.MX ROM by default. In order to enable this feature, a special signature should be included in the boot SW. Details in the example below.

The JTAG SW enable does not allow debug in case of boot or memory fault as it requires reset before entering debug.

### 4.3.1 JDE bit control in HAB (High Assurance Boot)

The HAB_JDE bit is set to **1** by ROM boot SW by default. To avoid setting this bit, the initial boot SW should contain special signature. Before generating the signed boot SW, the user must add the `Unlock` command in the CSF file and provide the device specific UID (64/128 bit) in the proper format as a sequence of 8-bytes, see the below example:

```
[Unlock]
        Engine = OCOTP
        Features = JTAG
        UID = 0x01, 0x23x 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef
```

**NOTE**

Refer to the CST User's Guide document in the CST package.

## 4.4 JTAG HAB Enable Override(HEO) fuse

The Unlock JTAG access by HAB_JDE bit can be permanently disabled by burning the JTAG_HEO fuse. The SW enabled JTAG feature reduces the overall security level of the system as it relies on SW protections. If this feature is not required, it is strongly recommended to burn the JTAG_HEO eFuse which disables this feature.

**NOTE**

For the final production hardware, the most secure and preferred method is to ensure that the JTAG interface is not pinned or routed at all. Also the interface is disabled via eFuses. The same applies to any other trace, diagnostic or test ports including SDP.

# 5 Secret response key management by the user

For every challenge value (**Device Unique ID** in i.MX 6/7/8M family) that is retrieved with a JTAG instruction, there is an associated secret response key known only by the user. The JTAG tool vendor only handles the JTAG mechanism used by this authentication process, and does not know the secret response key value programmed into the eFuses. It is left to the user to determine the level of protection that is put in place. The following are policies for secret response key management by the user application.

1. Identical Response Keys—The same response key is used for each chip. The user can choose a response key that will be fused in all chips. This is the simplest, but least sophisticated usage from a security point of view. If an unauthorized user gains access to the fused response key, all the products fused with this response key can be accessed through the JTAG port.
2. Database of Unique Response Keys—The user maintains a database of all generated response keys. The user application can look up the table based on the challenge value. It is possible to implement a secure server holding the challenge/response pairs authenticating the user but this requires an independent implementation effort. The challenge values for all ICs must be read and a database of matching challenge response pairs must be built. Storing and managing numerous response keys is not trivial, but advantageous from a security standpoint, as it does not rely on any breakable algorithms.
3. Algorithmically Generated Response Keys—Response keys are generated based on an algorithm. With this method, there is no large database to manage. For instance, the challenge value can be used by the algorithm to generate a response key. This response key is programmed into SJC_RESP eFuses. Then, every time the challenge value is retrieved through JTAG, it can be processed by the user application and used to generate the expected response key for the JTAG debug tools. Please note that once the algorithm is exposed or reverse engineered, this method is no longer secure.

**NOTE**

NXP does not provide secure response key management or key generation services; these topics are not within the scope of this document.

# 6 Debug tool examples to use Secure JTAG

To use the Secure JTAG feature the JTAG debugger must support it. The first example provided in this section uses the Lauterbach TRACE32 debug tool while the second example uses the Arm DS-5 DSTREAM debug tool, where both tools have been validated by NXP to support this feature.

Although the procedures outlined in the examples below use an i.MX 6Q SoC device on an NXP SABRE SD board, they can also be applied to other i.MX 6/7/8M family ICs. The following steps assume users have experience working with the Lauterbach TRACE32 or Arm DS-5/ARMDS DSTREAM debug tool and the NXP-provided manufacturing tool.

**NOTE**

NXP continuously engages with other Debug tool vendors such as Arm/Lauterbach to incorporate Secure JTAG support in their tools. If such support is not available in the debug tool, please contact the corresponding vendor's support for details.

## 6.1 Steps to program Secure JTAG eFuses using the NXP manufacturing tool (Universal Update Utility)

To program the relevant eFuses needed for Secure JTAG on the chip, the user should first follow the steps outlined below. Information on the Fusemap can be found in the appropriate i.MX 6/7/8M family reference manual available at www.NXP.com . The NXP Universal Update Utility (version 1.3.154) is used in the following steps to program eFuses.

1. Download the latest UUU release from https://github.com/NXPmicro/mfgtools/releases.
2. Download the latest Linux Binary Demo Files for the wanted board from https://www.nxp.com/design/i-mx-developer-resources/i-mx-software-and-development-tools:IMX-SW.
3. Extract the package above to get an U-boot binary. Copy this binary to a new folder, for example, *imx_debug*.
4. Create the *uuu.auto* configuration file in this folder. Edit the content as needed according to the target SoC, fuses banks and words, and user secret key. This file handles the following operations:
   - Program a secret response key in the eFuse SJC_RESP. In the example below, value `0xedcba987654321` is programmed. The user should define their own response key.
   - Program `0x1` in the eFuse SJC_RESP_LOCK to disable read/write access of the secret response key.
   - Program `0x1` in the eFuse JTAG_SMODE to switch the SJC to Secure JTAG mode.

   The `uuu.auto` example for burning Secure JTAG eFuses on i.MX 6/7/8M devices (specific commands for i.MX 8M* families are highlithed):

```
uuu_version 1.3.154

# If needed, modify the binary name below
SDP: boot -f <u-boot_bin>

# The three commands below are only needed if the target board is an i.MX8M*
# If needed, modify the binary name below
SDPV: delay 1000
SDPV: write -f <u-boot_bin> -skipspl
SDPV: jump

# Programming eFuses
# Secret response key example 0x00edcba987654321
# The user should define their own response key and change the lines below accordingly
# If the target is i.MX 8MP the user needs to program the two additional SJC_RESP banks
FB: ucmd fuse prog -y <bank> <word> 0x87654321
FB: ucmd fuse prog -y <bank> <word> 0x00edcba9

# Burn SJC_RESP_LOCK
FB: ucmd fuse prog -y <bank> <word> <SJC_RESP_LOCK_value>

# Burn JTAG_SMODE
FB: ucmd fuse prog -y <bank> <word> <JTAG_SMODE_value>

FB: done
```

   The created folder should now contain the U-boot binary and the `uuu.auto` configuration file.
5. Put the board in serial download mode. The OTG cable must be connected.
6. Turn on the board and run UUU. Be aware that this operation will program fuses on the SoC as specified in the UUU configuration file.

```
$ ./uuu /path/to/folder/imx_debug
```

## 6.2 Steps to connect Lauterbach debug tool via Secure JTAG

The following steps connect the Lauterbach debug tool to the i.MX 6/7/8M SoC when using Secure JTAG:

1. Download the Lauterbach Trace32 scripts for i.MX 6/7/8M family Secure JTAG support from https://www.lauterbach.com/frames.html?scripts.html.

   If you wish to navigate to these scripts from Lauterbach's main page for reference, they are located under **Downloads** -> **Start-Up and other Scripts** at **arm** -> **imx***.

2. In the downloaded package, edit the file named *calculateresponse.cmm*. In this file add the secret response key which was programmed into the `SJC_RESP` eFuse. In the following example the secret response key is *0xedcba987654321*, and matches the response key programmed in the eFuses in How to put the chip in Secure JTAG mode.

   ```
   entry &challenge
   &response=0xedcba987654321
   enddo &response
   ```

3. Run the Lauterbach attach script file named *attachimx.cmm* using the Lauterbach host application. The debug tool should successfully attach to the i.MX 6/7/8M family target over JTAG. Figure 2 shows a successful attach over Secure JTAG.

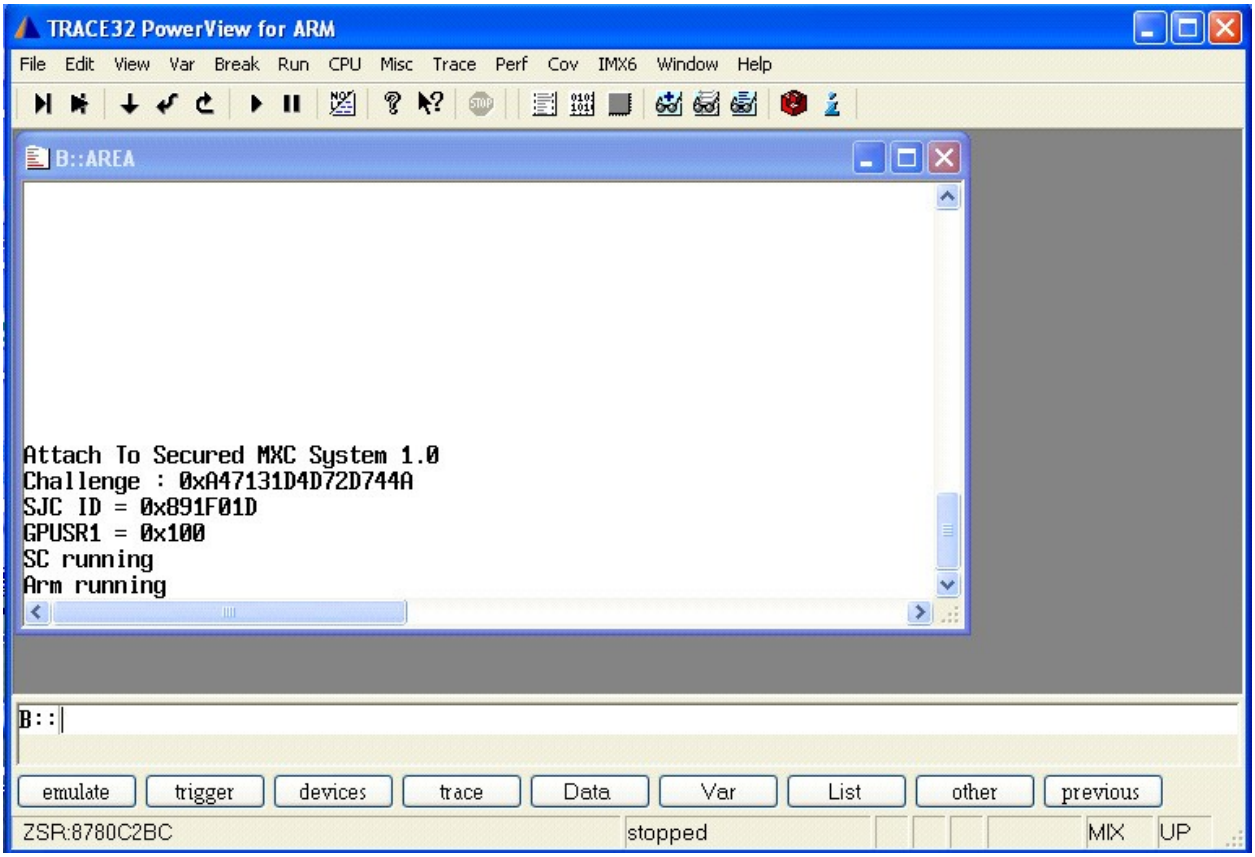


**Figure 2. Successful attach over Secure JTAG**

Users can now perform normal JTAG debugger operations, as the device has been authenticated using the challenge response mechanism.

**NOTE**

Any reset after JTAG access authorization will shift the JTAG controller back to its lock state, requiring that this authentication process be repeated.

4. In order to ensure that i.MX 6/7/8M family SJC is operating in secure mode, edit the *calculateresponse.cmm* file and provide an incorrect response key. Re-run the attach script *attachimx.cmm*. The debug tool should fail to attach to the i.MX 6/7/8M family target over JTAG. Figure 3 shows a failure to attach over Secure JTAG.

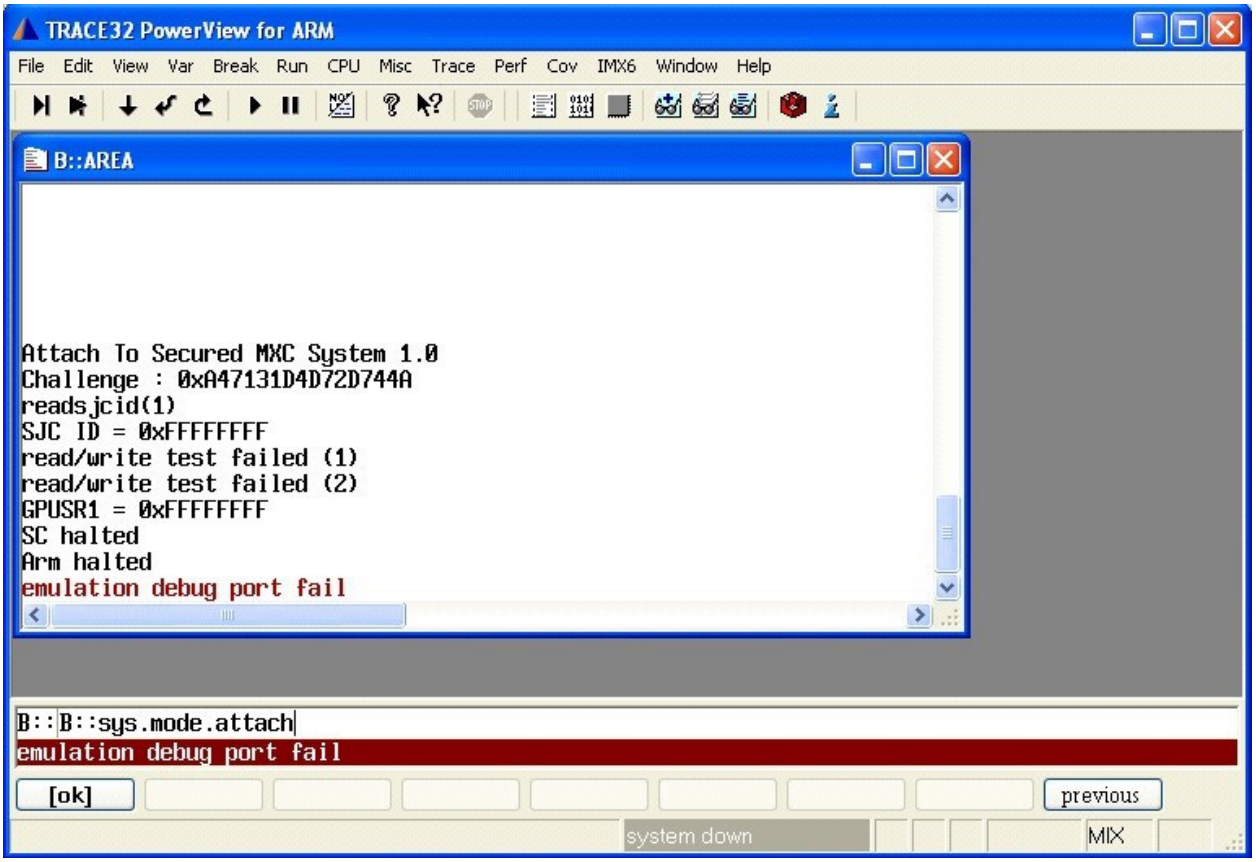


Figure 3. Failure to attach over Secure JTAG

## 6.3 Steps to connect Arm DS-5/ARMDS DSTREAM debug tool via Secure JTAG

The following steps connect the Arm DS-5/ARMDS DSTREAM debug tool to the i.MX SoC when using Secure JTAG:

1. Open the DS-5/ARMDS Eclipse Platform and enter the **Debug Configurations** under the **Run** tab.
2. In the *DS-5/ARMDS Debugger* menu select the configuration according to the target under debug.

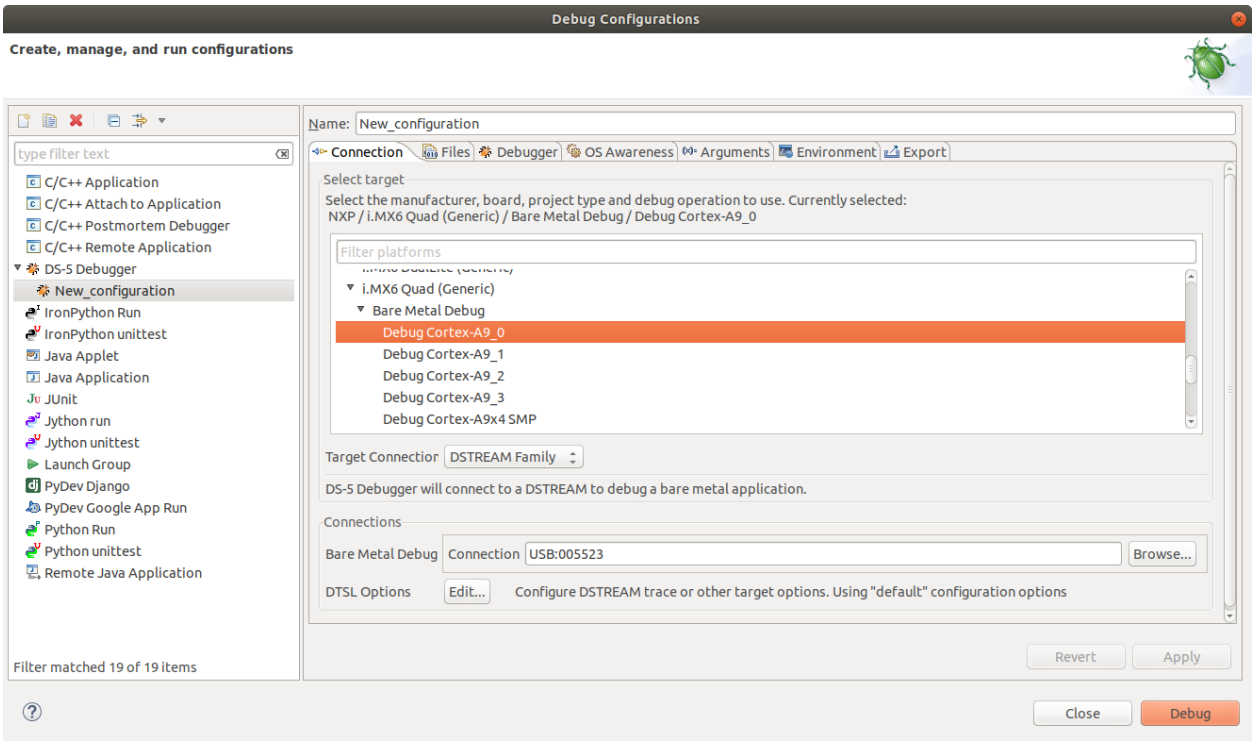


**Figure 4. DS-5 debug configuration**

3. Open the **DTSL Options**. Under the **SJC** tab, select the **Configure the SJC** checkbox and provide the **SJC key**, in this case `0xedbca987654321`.

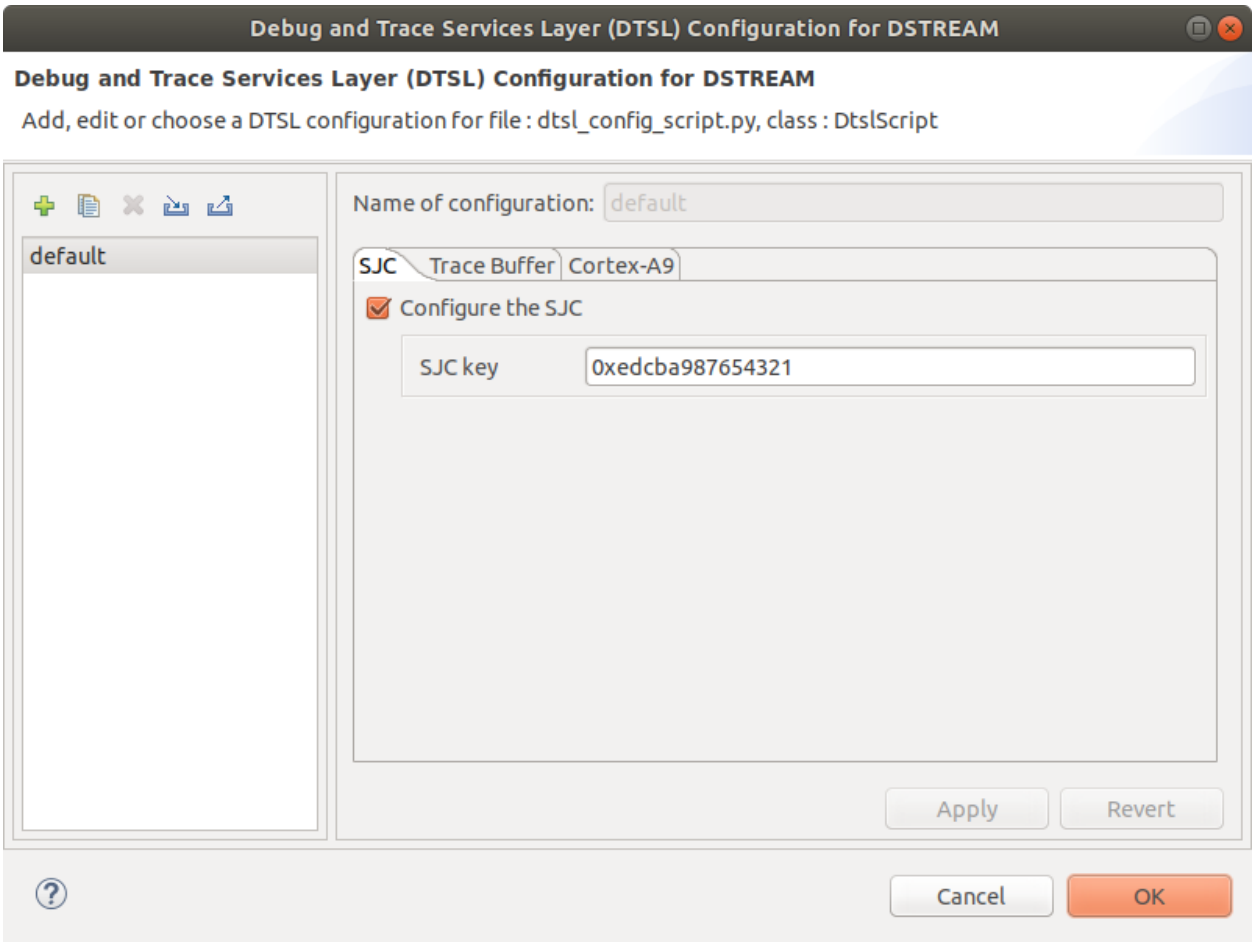


Figure 5. DS-5 SJC key configuration

4. Close this window and start the debug session.

Users can now perform normal JTAG debugger operations, as the device has been authenticated using the challenge response mechanism.

**NOTE**

Any reset after JTAG access authorization will shift the JTAG controller back to its lock state, requiring that this authentication process be repeated.

# 7 Revision history

Table 2 provides a revision history for this document.

**Table 2. Revision history**

| Rev. number | Date | Substantive change(s) |
|---|---|---|
| Rev. 0 | 02/2013 | Initial public release. |
| Rev. 1 | 03/2015 | Removed specific reference manual examples in No Debug and How to put the chip in Secure JTAG mode. |
| Rev. 2 | 07/2020 | • Added information for i.MX 8M family.<br>• Added No Debug and JTAG Enabled related information.<br>• Added JTAG_HEO fuse information.<br>• Added HAB_JDE.<br>• Replaced steps to program Secure JTAG eFuses using the Freescale manufacturing tool chapter with Steps to program Secure JTAG eFuses using the NXP manufacturing tool (Universal Update Utility).<br>• Updated Lauterbach scripts download link.<br>• Added Steps to connect Arm DS-5/ARMDS DSTREAM debug tool via Secure JTAG. |

*How To Reach Us*

**Home Page:**

nxp.com

**Web Support:**

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.



For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

**Date of release: 07/2020**

**Document identifier: AN4686**