Freescale Semiconductor
Application Note

Document Number: AN4686
Rev. 1, 03/2015

Configuring Secure JTAG for the i.MX 6 Series Family of Applications Processors

1 Introduction

The purpose of this document is to describe how to configure Secure JTAG on the i.MX 6 series family of applications processors.

The i.MX 6 series System JTAG Controller (SJC) provides a method of regulating the JTAG access. The three JTAG security modes available on the i.MX 6 series 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

Contents

to change the fuse back to the unburned state. This document focuses 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 Secure JTAG on i.MX 6 series

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 Freescale leaves it to the user to decide what works best for their environment.

2.1 How to put the chip in Secure JTAG mode

The i.MX 6 series 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 series reference manual available at www.freescale.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, Freescale recommends setting the fuse to Override Protect (OP) mode.

Table 1. Secure JTAG eFuse Configuration

Fuse Name	Number of Fuses	Fuse Function	Settings	Locked By
JTAG_SMODE	2	JTAG Security Mode. Controls the security mode of the JTAG debug interface	00 - JTAG enable mode (Default) 01 - Secure JTAG mode 11 - No debug mode	BOOT_CFG_LOCK

Table 1. Secure JTAG eFuse Configuration

Fuse Name	Number of Fuses	Fuse Function	Settings	Locked By
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 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 1x - Override Protect (OP) x1 - Write Protect (WP) 11 - Both OP and WP	N/A

2.1.1 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 series 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 OCOTP_CFG0 and OCOTP_CFG1. These eFuses will be programmed by Freescale during manufacturing.
- The secret response key (56 bits) 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 HW_OCOTP_LOCK_SJC_RESP.

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.

2.1.2 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.

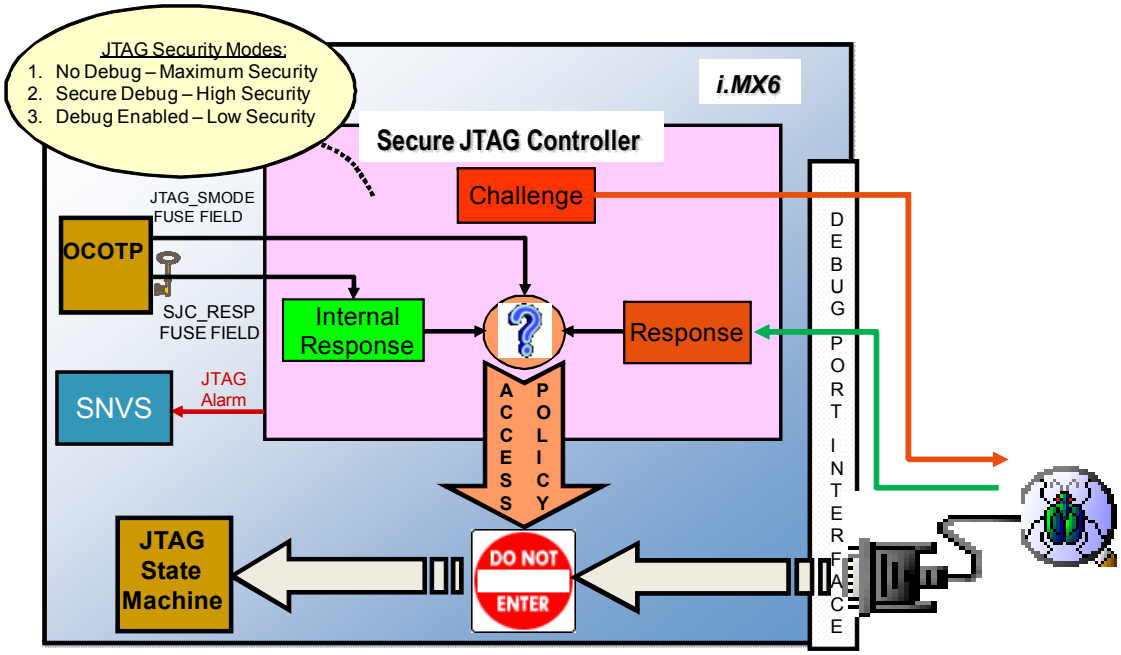

Figure 1. Secure JTAG operation

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 Freescale or the debug tool vendors. Key management is discussed further in Section 3, "Secret response key management by the user".

2.2 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.

3 Secret response key management by the user

For every challenge value ("Device Unique ID" in i.MX 6 series) 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:

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

4 Debug tool example to use Secure JTAG

To use the Secure JTAG feature the JTAG debugger must support it. The example provided in this section uses the Lauterbach TRACE32 debug tool, which has been validated by Freescale to support this feature. Freescale is also working with other Debug tool vendors such as ARM to incorporate Secure JTAG support in future tools.

Although the procedures outlined in the example below use an i.MX 6Dual/6Quad SoC device on a Freescale SABRE SD board, they can also be applied to their i.MX 6 series ICs. The following steps assume users have experience working with the Lauterbach TRACE32 debug tool and the Freescale-provided manufacturing tool.

4.1 Steps to program Secure JTAG eFuses using the Freescale manufacturing tool

To program the relevant eFuses needed for Secure JTAG on the chip, the user should first follow the steps outlined below. Information on the On-Chip OTP Controller (OCOTP_CTRL) and the Fusemap can be found in the appropriate i.MX 6 series reference manual available at www.freescale.com . The Freescale manufacturing tool (version 2.0) is used in the following steps to program eFuses.

1. Download the latest i.MX 6 series manufacturing tool from: http://www.freescale.com/webapp/sps/site/taxonomy.jsp?code=IMX6X_SERIES
2. Open the "*ucl2.xml*" file. For i.MX 6Quad, it is located in the "*Profiles\MX6Q Linux Update\OS Firmware*" directory.
3. Add a new operation list section to this file to program eFuses for secure JTAG. In the example provided, this list is named "*SabreSD-SJC-Fuse*". The user should program the values below to the eFuses needed for secure JTAG:

— 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 HW_OCOTP_LOCK_SJC_RESP 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 following example demonstrates how to add the commands which can be interpreted by the manufacturing tool to program the above eFuses. The example does not program the BOOT_CFG_LOCK[1:0] eFuses to prevent further modifications of the JTAG_SMODE eFuse, as programming these lock eFuses will disable access to functions in addition to the JTAG mode bits. Hence, this should only be performed by the user once the final configuration has been decided.

```
<LIST name="SabreSD-SJC-Fuse" desc="Blow SJC fuses">

<CMD state="BootStrap" type="boot" body="BootStrap" file ="u-boot-mx6q-sabresd.bin"
>Loading U-boot</CMD>

<CMD state="BootStrap" type="load" file="uImage" address="0x10800000"

loadSection="OTH" setSection="OTH" HasFlashHeader="FALSE" >Loading Kernel.</CMD>

<CMD state="BootStrap" type="load" file="initramfs.cpio.gz.uboot"
address="0x10C00000"

loadSection="OTH" setSection="OTH" HasFlashHeader="FALSE" >Loading Initramfs.</CMD>

<CMD state="BootStrap" type="jump" > Jumping to OS image. </CMD>

<CMD state="Updater" type="push" body="$ ls /sys/fsl_otp ">Showing HW_OCOTP fuse
bank</CMD>

<CMD state="Updater" type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_LOCK">Read the
HW_OCOTP_LOCK fuse</CMD>

<CMD state="Updater" type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_CFG5">Read the
JTAG_SMODE fuse</CMD>

<CMD state="Updater" type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_RESP0">Read the
SJC_RESP0 fuse</CMD>

<CMD state="Updater" type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_HSJC_RESP1">Read
the SJC_RESP1 fuse</CMD>

<CMD state="Updater" type="push" body="$ echo 0x87654321 >
/sys/fsl_otp/HW_OCOTP_RESP0"> Burn SJC_RESP0 fuse</CMD>

<CMD state="Updater" type="push" body="$ echo 0x00edcba9 >
/sys/fsl_otp/HW_OCOTP_HSJC_RESP1"> Burn RESP1 fuse</CMD>

<CMD state="Updater" type="push" body="$ echo 0x00000040 >
/sys/fsl_otp/HW_OCOTP_LOCK"> Burn SJC_RESP lock fuse</CMD>

<CMD state="Updater" type="push" body="$ echo 0x00400000 >
/sys/fsl_otp/HW_OCOTP_CFG5"> Burn JTAG_SMODE =01 fuse</CMD>

<CMD state="Updater" type="push" body="$ echo SJC Fuse blow Complete!">Done</CMD>

</LIST>
```

4. Open the "*cfg.ini*" file located in the top-level directory of the manufacturing tool package. Edit the file to run the newly-added "*SabreSD-SJC-Fuse*" operation list, as shown in this example:

```
[profiles]

chip = MX6Q Linux Update
```

```
[platform]
board = SabreSD


[LIST]
name = SabreSD-SJC-Fuse
```

5. Run the Manufacturing tool.

4.2 Steps to connect Lauterbach debug tool via Secure JTAG

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

1. Download the Lauterbach Trace32 scripts for i.MX 6 series Secure JTAG support from: www.lauterbach.com/scripts/arm/imx6/secure-jtag_/arm_imx6_secure-jtag_20130128_all_files.zip.

   If you wish to navigate to these scripts from Lauterbach's main page for reference, they are located under *"Support"* - *"Download Center"* - *"Start-UpScripts"* at *"arm"* - *"imx6"* - *"secure-jtag"*.
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 section 4.1

   ```
   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 series target over JTAG. The screen capture in Figure 2 shows a successful attach over Secure JTAG:

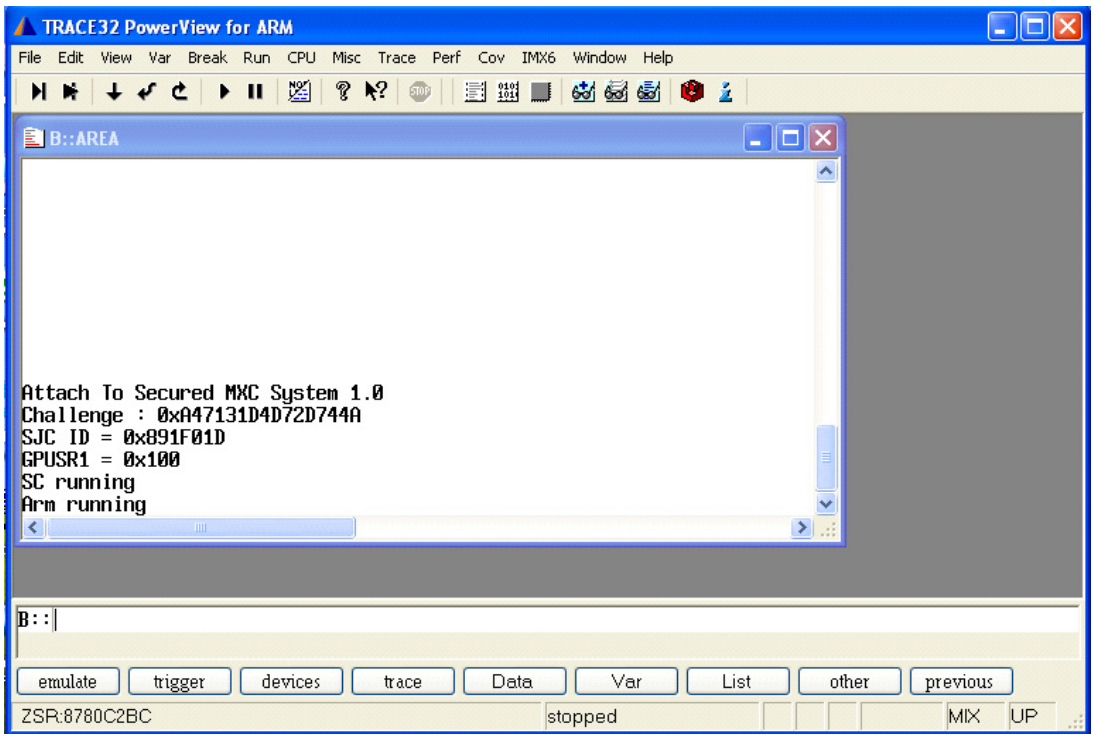

Figure 2. Successful attach over Secure JTAG

Users can now perform normal JTAG debugger operations, as the device has been autheticated 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 series 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 series target over JTAG. The screen shot in Figure 3 shows a failure to attach over Secure JTAG:

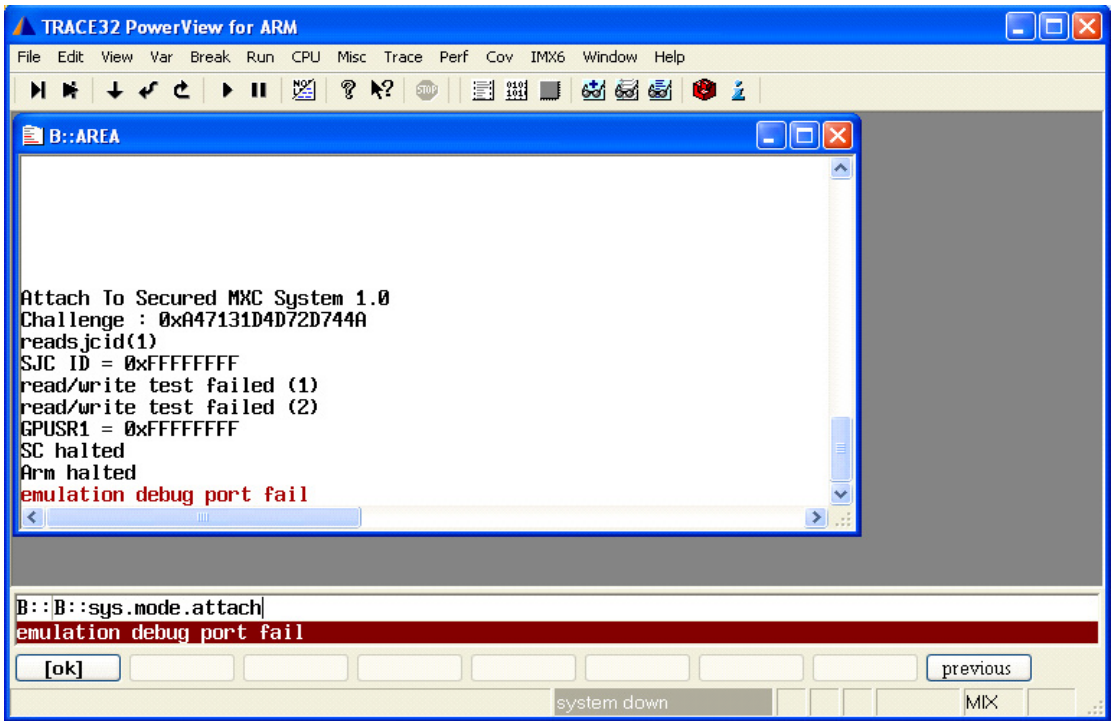

Figure 3. Failure to attach over Secure JTAG

5 Revision History

This table provides a revision history for this document.

Table 2. Document 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 Section 2.1 and 4.1.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4686
Rev. 1
03/2015