AN 556: Using the Design Security Features in Intel FPGAs

Contents

Using the Design Security Features in Intel® FPGAs..... 3
Overview of the Design Security Feature..... 4
Security Encryption Algorithm.....6
Non-Volatile and Volatile Key Storage..... 6
Key Programming..... 7
Intel Arria 10 and Intel Cyclone 10 GX Qcrypt Security Tool..... 8
Hardware and Software Requirements.....11
Hardware Requirements.....11
Software Requirements.....12
Steps for Implementing a Secure Configuration Flow.....12
Step 1: Generating .ekp File and Encrypting Configuration File..... 13
Step 2a: Programming Volatile Key into the FPGAs..... 18
Step 2b: Programming Non-Volatile Key into the FPGAs.....18
Step 3: Configuring the 40-nm, 28-nm, or 20-nm FPGAs with Encrypted Configuration Data..... 22
Steps to Enable Tamper-Protection Bit Programming..... 23
Supported Configuration Schemes..... 23
Security Mode Verification..... 24
Verification During JTAG Secure Mode..... 28
Serial Flash Loader Support with Encryption Enabled.....29
Serial Flash Loader Support with Encryption Enabled for Single FPGA Device Chain..... 29
JTAG Secure Mode for 28-nm and 20-nm FPGAs..... 30
Internal JTAG Interface..... 31
Design Example for JTAG Secure Mode.....36
Document Revision History for AN 556: Using the Design Security Features in Intel FPGAs... 39

683269 | 2021.05.21

Send Feedback

Using the Design Security Features in Intel® FPGAs

This application note describes how you can use the design security features in Intel® 40-, 28- and 20-nm FPGAs to protect your designs against unauthorized copying, reverse engineering, and tampering of your configuration files. This application note provides the hardware and software requirements for the 40-, 28- and 20-nm FPGAs design security features. This application note also provides steps for implementing secure configuration flow.

Note: This application note uses the term "40-nm","28-nm" or "20-nm" FPGAs. The following table lists the supported FPGAs and its applicable devices.

Table 1. Supported FPGAs

FPGA	Devices
40 nm	Arria® II and Stratix® IV
28 nm	Arria V, Cyclone® V, and Stratix V
20 nm	Intel Arria 10 and Intel Cyclone 10 GX

In the commercial and military environments, design security is an important consideration for digital designers. As FPGAs start to play a role in larger and more critical system components, it is important to protect the designs from unauthorized copying, reverse engineering, and tampering. Intel FPGAs address these concerns by encrypting their configuration bitstreams with the 256-bit Advanced Encryption Standard (AES) algorithm.

Table 2. AES Modes in Supported Intel FPGAs

FPGA	AES Mode
40 nm	Counter Mode (CTR)
28 nm	Cipher-block chaining (CBC)
20 nm	CTR and keyed-hash message authentication code (HMAC)

During device operation, FPGAs store configuration data in SRAM configuration cells. Because SRAM memory is volatile, the SRAM cells must be loaded with configuration data each time the device powers up. Configuration data is typically sent from an external memory source, such as a flash memory or a configuration device, to the FPGA. It is possible to intercept the configuration data when it is being sent from the memory source to the FPGA. If the configuration data were not encrypted, you could use the intercepted configuration data to configure another FPGA.

Intel FPGAs offer both volatile and non-volatile key storage. The key is stored in FPGAs when using the design security feature. Depending on the security mode, you can configure the FPGAs with a configuration file that is encrypted with the same key, or for board testing, configure with a normal configuration file.

The design security feature is available when configuring the FPGAs with fast passive parallel (FPP) configuration scheme with an external host (such as a MAX® II or MAX V device or microprocessor) or when using active serial (AS) or passive serial (PS) configuration schemes.

Related Information

- AN 680: Product Security Features for Altera Devices
  Provides more information about the product security features.
- Configuration, Design Security, and Remote System Upgrades in Arria II Devices
  Provides more information about the design security for Arria II devices.
- Configuration, Design Security, and Remote System Upgrades in Stratix IV Devices
  Provides more information about the design security for Stratix IV devices.
- Configuration, Design Security, and Remote System Upgrades in Arria V Devices
  Provides more information about the design security for Arria V devices.
- Configuration, Design Security, and Remote System Upgrades in Cyclone V Devices
  Provides more information about the design security for Cyclone V devices.
- Configuration, Design Security, and Remote System Upgrades in Stratix V Devices
  Provides more information about the design security for Stratix V devices.
- Configuration, Design Security, and Remote System Upgrades in Intel Arria 10 Devices
  Provides more information about the design security for Intel Arria 10 devices.
- Configuration, Design Security, and Remote System Upgrades in Intel Cyclone 10 GX Devices
  Provides more information about the design security for Intel Cyclone 10 devices.
- Configuration Design Security in Intel MAX 10 FPGA

Overview of the Design Security Feature

The design security feature for Intel FPGAs protects against unauthorized copying , reverse engineering, and tampering. The following table lists some of the design approaches to make the solution secure.

The 20-nm FPGAs have additional security features that you can enable by burning a fuse, or by setting an option bit in the configuration bit-stream by using the stand-alone Qcrypt tool or the Intel Quartus® Prime Convert Programming File tool. Tamper-protection bit and JTAG Secure mode can be enabled separately in 20-nm FPGAs only.

Table 3. Design Security Approach for 40-nm and 28-nm FPGAs

Caution: Enabling the tamper-protection bit disables the test mode in 40-nm and 28-nm FPGAs. Disabling the test mode is irreversible and prevents Intel from carrying out failure analysis. To enable the tamper protection bit, refer to the *Steps to Enable Tamper-Protection Bit Programming* section.

Design Security Element	40-nm FPGA	28-nm FPGA (1)
Non-Volatile key	The non-volatile key is securely stored in fuses within the device. Proprietary security features make it difficult to determine this key.	
Volatile Key	The volatile key is securely stored in battery-backed RAM within the device. Proprietary security features make it difficult to determine this key.	
Key Generation	Two user provided 256-bit strings are processed to generate a 256-bit key that is programmed into the device.	A user provided 256-bit key is processed by a one-way function before being programmed into the device.
Key Choice	User only set either 1 security key type (non-volatile key or volatile key) into the device.	
Tamper Protection Mode	Tamper protection mode prevents the FPGA from being loaded with an unencrypted configuration file. When you enable this mode, the FPGA can only be loaded with a configuration that has been encrypted with your key. Unencrypted configurations and configurations encrypted with the wrong key result in a configuration failure. You can enable this mode by setting a fuse within the device.	
Configuration Readback	These devices do not support a configuration readback feature which makes readback of your unencrypted configuration data infeasible.	

Table 4. Design Security Approach for 20-nm FPGAs

Design Security Element	Description
Non-Volatile key	The non-volatile key is securely stored in fuses within the device. Proprietary security features make it difficult to determine this key.
Volatile Key	The volatile key is securely stored in battery-backed RAM within the device. Proprietary security features make it difficult to determine this key.
Key Generation	A user provided 256-bit key is processed by a one-way function before being programmed into the device.
Key Choice	Both volatile and non-volatile key can exist in a device. User can choose which key to use by setting the option bits in encrypted configuration file through the Convert Programming File tool or the Qcrypt tool.
Tamper Protection Mode	Tamper protection mode prevents the FPGA from being loaded with an unencrypted configuration file. When you enable this mode, the FPGA can only be loaded with a configuration that has been encrypted with your key. Unencrypted configurations and configurations encrypted with the wrong key result in a configuration failure. You can enable this mode by setting a fuse within the device.

continued...

(1) When you enable the tamper-protection bit in 28-nm FPGAs, the device is in the JTAG secure mode.

Design Security Element	Description
Configuration Readback	These devices do not support a configuration readback feature. From a security perspective, this makes readback of your unencrypted configuration data infeasible.
Security Key Control	By using different JTAG instructions and the security option in the Qcrypt tool, you have the flexibility to permanently or temporarily disable the use of the non-volatile or volatile key. You can also choose to lock the volatile key to prevent it from being overwritten or reprogrammed.
JTAG Access Control	You can enable various levels of JTAG access control by setting the OTP fuses or option bits in the configuration file using the Qcrypt tool: 1. Force full configuration or partial configuration to be done through HPS only. 2. Bypass external JTAG pin or HPS JTAG. This feature disables external JTAG or HPS JTAG access, but can be unlocked through internal core access. (2) 3. Disable all AES key related JTAG instructions from external JTAG pins. 4. Allows only a limited set of mandatory JTAG instruction to be accessed through external JTAG, similar to JTAG Secure mode.

Note: For additional details on these and other security features, contact Intel FPGA Technical Support.

Related Information

- Intel Arria 10 and Intel Cyclone 10 GX Qcrypt Security Tool on page 8
- Steps to Enable Tamper-Protection Bit Programming on page 23
- Steps for Implementing a Secure Configuration Flow on page 12
- JTAG Secure Mode for 28-nm and 20-nm FPGAs on page 30

Security Encryption Algorithm

Intel FPGAs have a dedicated AES decryptor block than can decrypt configuration bit-streams prior to configuring the FPGA device. The 28-nm FPGAs use the AES block in CBC mode, while the 40-nm and 20-nm FPGAs use the AES block in CTR mode. In addition, the 20nm devices implement techniques to mitigate side-channel attacks against the standard NIST CTR mode of encryption. If the security feature is not used, the AES decryptor is bypassed. The FPGAs AES implementation is validated as conforming to the Federal Information Processing Standards FIPS-197.

Related Information

Computer Security Resource Center (CSRC)
For the AES algorithm, refer to the *Federal Information Processing Standards Publication FIPS-197* or the *AES Algorithm (Rijndael) Information*. For the AES validation for FPGAs, refer to the *Advanced Encryption Standard Algorithm Validation List*.

Non-Volatile and Volatile Key Storage

Intel FPGAs offer both volatile and non-volatile key storage. The volatile key storage registers are reprogrammable and erasable. The contents of the key registers are retained between power-cycles with a coin-cell battery. Non-volatile key registers are fuse-based and are one-time programmable.

(2) Intel Cyclone 10 GX does not support force full configuration or partial configuration through HPS and HPS JTAG Bypass.

Note: Examples of lithium coin-cell type batteries that are used for volatile key storage purposes are BR1220 (–30°C to 80°C) and BR2477A (–40°C to 125°C).

Table 5. Volatile and Non-Volatile Key Comparison

Option	Volatile Key	Non-Volatile Key
Key Length	256 bits	256 bits
Key Programmability	Reprogrammable and erasable key	One-time programmable key
External Battery	Required	Not required
Key Programming Method (3)	On-board	Both on-board and off-board (4)
Design Protection (5)	Secure against copying, reverse engineering, and tampering	

Key Programming

Table 6. Key Programming Methods

Programming Procedure	Method	Programming Tool/Support
On-Board Programming	Prototyping	Intel FPGA Ethernet Cable (6), JTAG Technologies (7), Intel FPGA Parallel Port Cable (8), Intel FPGA Download Cable (9), and Intel FPGA Download Cable II (10).
	Production	JTAG Technologies*
Off-Board Programming	Prototyping	System General*
	Production	System General

(3) Key programming is carried out through JTAG interface. You need to use valid MSEL pin settings for the JTAG interface.

(4) Programming the non-volatile key fuses uses the standard voltage sources used by the FPGA during normal operation. No additional voltage rails are necessary for programming non-volatile key.

(5) Volatile key tamper-protection is only available for Arria II, Arria V, Cyclone V, Stratix V, Intel Arria 10, and Intel Cyclone 10 GX devices.

(6) Intel FPGA Ethernet Cable supports both volatile and non-volatile key programming.

(7) JTAG Technologies* supports both volatile and non-volatile key programming.

(8) Intel FPGA Parallel Port Cable supports only volatile key programming.

(9) Intel FPGA Download Cable support only volatile key programming except in 20-nm FPGAs, where it supports both volatile and non-volatile key programming.

(10) Intel FPGA Download Cable II supports both volatile and non-volatile key programming.

Key programming uses the following definitions:

- **On-board:** procedure in which the device is programmed on your board
- **Off-board:** procedure in which the device is programmed on a separate programming system
- **Prototyping:** method initially used to verify proper operation of a particular method
- **Production:** method used for large-volume production

Note: For other third-party non-volatile key programming, you must regulate the JTAG TCK pulse width (period) for proper polyfuse programming, as listed in Table 10 on page 11.

Related Information

Intel FPGA Technical Support
Provides information about programming support.

Intel Arria 10 and Intel Cyclone 10 GX Qcrypt Security Tool

The Qcrypt tool is a stand-alone encryption tool for encrypting and decrypting Intel Arria 10 and Intel Cyclone 10 GX FPGA configuration bit-stream files. The Qcrypt tool can also be used to encrypt HPS boot images through a script. Different kinds of security settings that are currently not accessible from the Intel Quartus Prime graphical user interface can be set through the Qcrypt tool.

The Qcrypt tool encrypts and decrypts raw binary files (`.rbf`) only and not other configuration files, such as `.sof` and `.pof` files. Throughout the encryption flow, the Qcrypt tool generates an authentication tag while encrypting the `.rbf` file. The authentication tag prevents any modification or tampering of the configuration bit-stream. Besides encryption and decryption, the Qcrypt tool allows you to enable and set various security features and settings. By incorporating security features and settings into the `.rbf` file, you have the flexibility to use different kinds of security features on Intel Arria 10 and Intel Cyclone 10 GX devices without permanently burning the security fuses. To generate the `.ekp` file or encrypted configuration file other than `.rbf`, you have to use the Intel Quartus Prime Convert Programming File tool.

Note: The Qcrypt tool is not license-protected and can be used by all Intel Quartus Prime software user.

Related Information

- Overview of the Design Security Feature on page 4
- Steps to Enable Tamper-Protection Bit Programming on page 23
- Steps for Implementing a Secure Configuration Flow on page 12
- JTAG Secure Mode for 28-nm and 20-nm FPGAs on page 30
- Qcrypt Tool Options on page 9
  Provides more information about Qcrypt tool features.
- AN 759: Intel Arria 10 SoC Secure Boot User Guide
  Provides more information about encrypting HPS boot images.

Using Qcrypt Tool

You can use the following command to encrypt and decrypt `.rbf` files. This command is the only way to set the advanced security options.

```
qcrypt [options] <input_file.rbf> <output_file.rbf>
```

Qcrypt Tool Options

Table 7. Basic Options in Qcrypt Tool

Basic Option	Descriptions
`--encrypt`	Encrypts `input_file.rbf` with default behavior.
`--decrypt`	Decrypts `input_file.rbf` to obtain the original bit-stream. The decrypted `.rbf` is not the same as original bit-stream if you had previously enabled any security options. You must explicitly reset these security options to level 0 if you want the decrypted `.rbf` to match the original pre-encrypted `.rbf`. Note that there are minor differences between the original and decrypted `.rbf` files. The differences can be ignored.
`--keyfile=<KEY_FILE>`	Default name for this key file is `keyfile.key`. This option allows you to specify an alternate name for the `keyfile.key`. The key file is located in the current project directory where the `input_file.rbf` is also stored. Refer example key file in Generating Single-Device .ekp File and Encrypting Configuration File using Intel Quartus Prime Software on page 14.
`--keyname=<KEY_NAME>`	Specify a named key to use from the key file. By default, the tool uses the first key from the key file.
`--keystore=<types of key>`	Specify which security key to be use: • `otp` (non-volatile key) • `battery` (volatile key) One-time programmable (`otp`) is the default value.
`--iv=<HEX_VALUE>`	Optional seed value for creating a non-random initialization vector (IV). By default, an `.rbf` generates a different encrypted `.rbf` every time it is encrypted. This option allows you to specify a seed value to ensure the same encrypted `.rbf` is generated when using same `--iv` value. HEX_VALUE can be any arbitrary 32-bit hexadecimal value.

Table 8. Security Options in Qcrypt Tool

Security Option	Descriptions
`--lockto=<FILE_NAME.qlk>`	Locks authentication to corresponding prior base bitstream. The `.qlk` file is automatically created when a base configuration file, such as a CvP core image bitstream, is encrypted. Use this option when you want a follow-on core CvP or partial reconfiguration image to be usable only with that base configuration. This prevents a follow-on bitstream from being loaded over a wrong (but otherwise authenticated) base bitstream.
`--no-lockto`	Overrides any mandatory `--lockto` requirement
`--epof-only=[0:3]`	Only allow encrypted and authenticated bit-streams to be used for external configuration.
`--no-config=[0:3]`	Disables configuration from external pins. With this option set, configuration can only be controlled by the internal HPS. *Note:* This security option is not supported in Intel Cyclone 10 GX.
`--no-pr=[0:3]`	Disables external partial-configuration.

continued...

Security Option	Descriptions
`--no-jtag-key=[0:3]`	Disables key-related JTAG instructions.
`--no-jtag-ext=[0:3]`	Enables JTAG Secure mode.
`--no-jtag=[0:3]`	Forces the external JTAG pins into BYPASS mode.
`--no-hps-jtag=[0:3]`	Forces the internal HPS JTAG into BYPASS mode. *Note:* This security option is not supported in Intel Cyclone 10 GX.
`--no-otp-key=[0:3]`	Disables use of the non-volatile OTP fuse key.
`--no-battery-key=[0:3]`	Disables use of the battery-backed key.
`--lock-battery-key=[0:3]`	Prevents the battery-backed volatile key from being changed or overwritten.
`--secure=[2:3]`	Disables Test Mode <default=2>.

Related Information

- Intel Arria 10 and Intel Cyclone 10 GX Qcrypt Security Tool on page 8
- AN 759: Intel Arria 10 SoC Secure Boot User Guide
  Provides more information about encrypting HPS boot images.

Security Levels of Qcrypt Tool Security Option

The Qcrypt tool allows the flexibility to determine the security level of the security options in Table 8 on page 9. You can choose the minimum or maximum requirement by specifying the level of security from 0 to 3.

Table 9. Qcrypt Tool Security Option Security Levels

Security Level	Descriptions
0	The security feature is not enabled unless by the corresponding OTP fuse.
1	The security feature is enabled from the start of the current full- or partial-reconfiguration until the start of the next full configuration.
2	The security feature is enabled until the next power-on-reset. Additionally, configuration does not proceed if any action normally prevented by the security feature has taken place since the last power-on-reset.
3	Configuration does not proceed unless the security feature has been permanently enabled by blowing the corresponding fuses in the device.

The security level of 2 provides a level of security almost as powerful as setting the corresponding OTP security fuse, but with some flexibility. For example, the use of JTAG may be required for manufacturing test or debug, but you may want to totally disable JTAG while a secured (encrypted) bit-stream is loaded in the device. Furthermore, you may not want to load a secured bit-stream into a device that had previously been probed with any kind of JTAG command.

Intel recommends that you use the strictest security level for each option that is consistent with your design requirements.

Note: You can find information on the Qcrypt tool by using the `--help` option.

Hardware and Software Requirements

When using the design security feature, a volatile or non-volatile key is stored in the FPGA. The key must be programmed before the FPGA is configured with an encrypted configuration file.

Hardware Requirements

The following table lists the specifications that you must follow for a successful key programming.

Table 10. Specifications for Key Programming

Parameter	Key Programming Mode	
	Non-Volatile Key	Volatile Key
TCK period	10 µs ± 1 µs (11)	—
Ambient Temperature	25°C ± 5°C	25°C ± 5°C
Voltage (VCCBAT)	—	(12)

V_{CCBAT} is a dedicated power supply for the volatile key storage and is not shared with other on-chip power supplies, such as V_{CCIO} or V_{CC}. V_{CCBAT} continuously supplies power to the volatile register regardless of the on-chip supply condition.

Note: After power up, you must wait for the device to exit power-on reset (POR) before beginning the key programming. You may encounter verification error when programming the volatile Encryption Key Programming (**.ekp**) file if you have the VCCBAT pin tied to GND. The VCCBAT pin must be tied to the recommended V_{CCBAT} voltage provided in the respective device family pin connection guidelines for proper operation.

Related Information

- Device Datasheet for Arria II Devices
  Provides more information about JTAG, POR, and voltage specifications.
- DC and Switching Characteristics for Stratix IV Devices
  Provides more information about JTAG, POR, and voltage specifications.
- Arria V Device Datasheet
  Provides more information about JTAG, POR, and voltage specifications.
- Cyclone V Device Datasheet
  Provides more information about JTAG, POR, and voltage specifications.
- Stratix V Device Datasheet
  Provides more information about JTAG, POR, and voltage specifications.
- Intel Arria 10 Device Datasheet
  Provides more information about JTAG, POR, and voltage specifications.

(11) Applies to 40-nm and 28-nm FPGAs only.

(12) If you do not use the volatile key, refer to the respective device family pin connection guidelines for VCCBAT connection.

- Intel Cyclone 10 GX Device Datasheet
  Provides more information about JTAG, POR, and voltage specifications.
- Stratix V E, GS, and GX Device Family Pin Connection Guidelines
- Stratix V GT Device Family Pin Connection Guidelines
- Arria V GT, GX, ST and SX Device Family Pin Connection Guidelines
- Arria V GZ Device Family Pin Connection Guidelines
- Stratix IV GX and E Device Family Pin Connection Guidelines
- Stratix IV GT Device Family Pin Connection Guidelines
- Arria II Device Family Pin Connection Guidelines
- Intel Arria 10 GX, GT, and SX Device Family Pin Connection Guidelines

Software Requirements

Table 11. Supported Quartus Version for Intel FPGAs

You are required to use the supported Quartus software version below to enable the design security feature based on your FPGA type.

Device	Supported Quartus Version
40-nm FPGA	Quartus II software version 9.0 or later.
28-nm FPGA	Quartus II software version 11.0 or later.
20-nm FPGA	Intel Quartus Prime software version 15.1 or later. (13)

Note: To enable the design security feature for Intel Quartus Prime Lite Edition, obtain a license file from Intel FPGA Technical Support.

Related Information

Intel FPGA Technical Support

Steps for Implementing a Secure Configuration Flow

To implement a secure configuration flow, follow these steps:

1. Generate the **.ekp** file and encrypt the configuration data.

   The Intel Quartus Prime configuration software always uses the user-defined 256-bit key to generate a key programming file and an encrypted configuration file. The encrypted configuration file is stored in an external memory, such as a flash memory or a configuration device. For details, refer to Step 1: Generating .ekp File and Encrypting Configuration File on page 13.

   Note: For the 20-nm FPGAs, you can also encrypt an `.rbf` by using the stand-alone Qcrypt tool with extended security options.

2. Program the user-defined 256-bit key into the FPGAs.

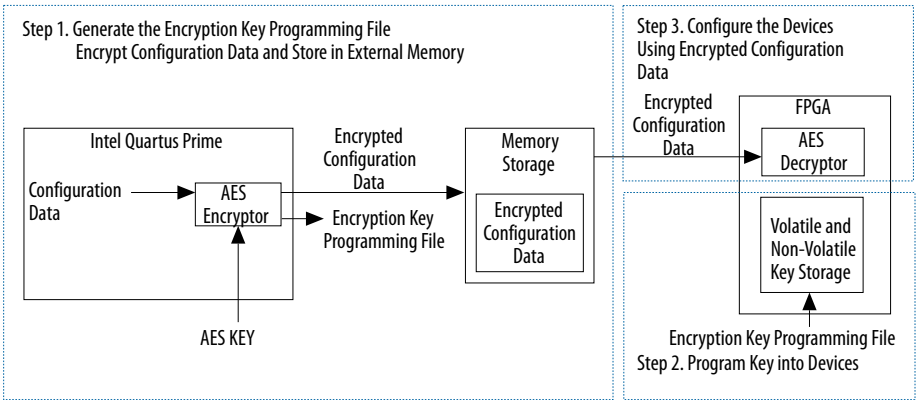
(13) For 20-nm FPGAs, you can also enable the design security features by using the stand-alone Qcrypt tool available in the Intel Quartus Prime software.

For details, refer to Step 2a: Programming Volatile Key into the FPGAs on page 18 and Step 2b: Programming Non-Volatile Key into the FPGAs on page 18.

3. Configure the 40-nm, 28-nm or 20-nm FPGA device.

   At power up, the external memory source sends the encrypted configuration file to the FPGA. The device uses the stored key to decrypt the file and to configure itself. For details about how to configure FPGAs with encrypted configuration data, refer to Step 3: Configuring the 40-nm, 28-nm, or 20-nm FPGAs with Encrypted Configuration Data on page 22.

Figure 1. Secure Configuration Flow



Step 1: Generating .ekp File and Encrypting Configuration File

To use the design security feature in the FPGAs, you must encrypt your 20-nm design using the Qcrypt tool, or generate an **.ekp** file and encrypt your configuration files using the Intel Quartus Prime software. The key is not saved into any Intel Quartus Prime-generated configuration files and the actual 256-bit key is generated from the bit sequences.

To enable the design security feature, you can obtain a license file from Intel FPGA Technical Support.

The **.ekp** file has different formats, depending on the hardware and system used for programming. There are three file formats supported by the Intel Quartus Prime software:

- JAM Byte Code (**.jbc**) file
- JAM™ Standard Test and Programming Language (STAPL) Format (**.jam**) file
- Serial Vector Format (**.svf**) file

Only the **.ekp** file type is generated automatically from the Intel Quartus Prime software. You must create the **.jam** and **.svf** files using the Intel Quartus Prime software if these files are required in the key programming. The Intel Quartus Prime software generates the JBC format of the **.ekp** file in the same project directory.

Note: Intel recommends that you keep the **.ekp** file confidential.

Use the **.ekp** file with the Intel FPGA Ethernet Cable communications cable or Intel FPGA Download Cable and the Intel Quartus Prime software. The Intel FPGA Ethernet Cable communications cable can support both volatile and non-volatile key

programming whereas the Intel FPGA Download Cable is used only for volatile key programming. The **.jam** file format is generally used with third-party programming vendors and JTAG programmer vendors. The **.svf** file format is used with JTAG programmer vendors.

Generating Single-Device .ekp File and Encrypting Configuration File using Intel Quartus Prime Software

To generate a single device **.ekp** file and encrypt your configuration file, follow these steps:

1. Obtain a license file to enable the design security feature from Intel FPGA Technical Support.
2. Start the Intel Quartus Prime software.
3. On the Tools menu, click **License Setup**. The **Options** dialog box displays the **License Setup** options.
4. In the **License file** field, enter the location and name of the license file, or browse to and select the license file.
5. Click **OK**.
6. Compile your design with one of the following options:
   a. On the Processing menu, click **Start Compilation**.
   b. On the Processing menu, point to **Start** and click **Start Assembler**.

   An unencrypted SRAM Object File (**.sof**) is generated.
7. On the File menu, click **Convert Programming Files**. The **Convert Programming Files** dialog box appears.
   a. In the **Convert Programming Files** dialog box, select the programming file type from the **Programming file type** list.
   b. If applicable, select the appropriate configuration device from the **Configuration device** list.
   c. Select the mode from the **Mode** list.
   d. Type the file name in the **File name** field, or browse to and select the file.
   e. Under the **Input files to convert** section, click **SOF Data**.
   f. Click **Add File** to open the **Select Input File** dialog box.
   g. Browse to the unencrypted SOF file and click **Open**.
   h. Under the **Input files to convert** section, select- the SOF file name. The field is highlighted.
   i. Click **Properties**. The **SOF Files Properties: Bitstream Encryption** dialog box appears.
   j. In the **SOF Files Properties: Bitstream Encryption** dialog box, turn on **Generate encrypted bitstream**.
   k. Turn on **Generate key programming file** and type the **.ekp** file path and file name in the text area, or browse to and select `<filename>`**.ekp**.
   l. Additional step for 20-nm FPGAs only: Turn on **Enable volatile security key check** box to encrypt the `.sof` file with volatile security key or turn it off to use non-volatile security key.

m. Additional step for 20-nm FPGAs only: Turn on **Generate encryption lock file** and insert the `.qlk` file path and file name in the text area, or browse to the desired `<filename>.qlk`.

n. Add the keys to the pull-down list either with a **.key** file or the **Add** button. The **Add** and **Edit** buttons bring up the **Key Entry** dialog box. The **Delete** button deletes the currently selected key from the pull-down list.

Note: 40-nm FPGAs require entry of two 256-bit keys. The encryption derived from a combination of the two 256-bit keys. 28-nm and 20-nm FPGAs require entry of a single 256-bit key. The final encryption key is derived using a one-way function.

Using the **.key** file option allows you to specify one or two key files in the corresponding drop-down box. You may use different files for the **Key 1** and **Key 2** fields, or use one **.key** file for both.

The **.key** file is a plain text file in which each line represents a key unless the line starts with "#". The "#" symbol is used to denote comments. Each valid key line has the following format: *<key identity><white space><256-bit hexadecimal key>*.

```
# This is an example key file
key1 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
key2 ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789
```

The key identity is an alphanumeric name that is used to identify the keys (similar to the key file entry). The key is also the text displayed when the **Show entered keys** button is turned off. It is displayed together with the full key when **Show entered keys** is turned on.

You can save the keys in the pull-down list to a **.key** file. You must click the corresponding **Save** button to save a key and to display the standard **File** dialog box. All keys in the pull-down list are saved to the selected or created **.key** file.

Select the **Key Entry Method** to enter the encryption key either with the on-screen keypad or keyboard.

The on-screen keypad allows you to enter the keys using the keypad. Select a key and click on the on-screen keypad to enter values. You have the option of allowing the keys to be shown as they are entered. If you use this option, you do not need to confirm the keys.

While the on-screen keypad is being used, any attempt to use the keyboard to enter the keys generates a pop-up notification and the key press is ignored. Alternatively, you can enter the encryption key from the keyboard.

i. Read the design security feature disclaimer. If you agree to and acknowledge the design security feature disclaimer, turn on the **acknowledgment** box.

ii. Click **OK**.

o. Additional step for 20-nm FPGAs only: Under **Security Options**, select the level from the **Disable external partial reconfiguration** list.

p. Additional step for 20-nm FPGAs only: Under **Security Options**, select the level from the **Disable key-related JTAG instructions** list.

q. Additional step for 20-nm FPGAs only: Under **Security Options**, select the level from the **Disable other extended JTAG instructions** list.

8. In the **Convert Programming Files** dialog box, click **OK**. The <*filename*>**.ekp** and encrypted configuration file are generated in the same project directory.
9. On the Tools menu, click **Programmer**. The **Programmer** dialog box appears.
10. In the **Mode** list, select **JTAG** as the programming mode.
11. Click **Hardware Setup**. The **Hardware Setup** dialog box appears.
    a. In the currently selected hardware list, select **Intel FPGA Ethernet Cable** as the programming hardware.
    b. Click **Done**.
12. Click **Add File**. The **Select Programmer File** dialog box appears.
    a. Type <*filename*>**.ekp** in the **File name** field.
    b. Click **Open**.
13. Highlight the **.ekp** file you added and click **Program/Configure**.
14. On the File menu, point to **Create/Update** and click **Create JAM, SVF, or ISC File**. The **Create JAM, SVF, or ISC File** dialog box appears.
15. Select the file format required (JEDEC STAPL Format [**.jam**]), for the **.ekp** file in the **File format** field.
16. Type the file name in the **File name** field, or browse to and select the file.
17. Click **OK** to generate the **.jam** file.
18. On the Tools menu, click **Programmer Options**. The **Programmer Options** dialog box appears.

    Note: For non-volatile secure design feature, you must turn off the **Configure volatile design security key** option to generate a non-volatile **.svf** file of the **.ekp** file.
19. Click OK.
20. Repeat steps 15 on page 16 to 17 on page 16 to generate a **.svf** file of the **.ekp** file. Use the default setting in the **Create JAM, SVF, or ISC File** dialog box when generating a **.svf** file of the **.ekp** file.

Generating Single-Device .ekp File and Encrypting Configuration File using Command-Line Interface in Intel Quartus Prime Software

There is a command-line interface that allows you to generate a single-device **.ekp** file and encrypt Raw Binary File (**.rbf**). The command-line interfaces uses the Intel Quartus Prime software command-line executable, `quartus_cpf`, and requires the following syntax or options:

- --key/-k <*path to key file*>:<*key identity*>
- A **.sof** file (user design)
- An **.ekp** file (the required encryption key programming file name)

You can create a compressed and uncompressed `.rbf` for configuration by using the following command with an option file which contains the string compression=on.

```
quartus_cpf -c --option=<option file> --key
<keyfile>:<keyid1>:<keyid2> <input_sof_file> <output_rbf_file>
```

Note:

1. Encryption and compression cannot be used simultaneously in 20 nm FPGAs.
2. For 20 nm FPGAs, use `non_volatile_key=off` to control the **Enable volatile key security** option during the **.rbf** file creation.

You can learn more on the option file from the Intel Quartus Prime software command line help. Run `quartus_cpf --help=option` to learn more on the available options. For 20 nm FPGAs, use the Qcrypt tool command line to encrypt or decrypt the `.rbf` file. To generate the **.ekp** or encrypted configuration file other than **.rbf**, you have to go through quartus_cpf.

Example 1.

The following example shows two sets of keys that are stored in two different key files: key1 in **key1.key** and key2 in **key2.key**.

```
quartus_cpf --key D: \SIV_DS\key1.key:key1 --key

D:\SIV_DS\key2.key:key2 D:\SIV_DS\test.sof D:\SIV_DS\test.ekp
```

Example 2.

The following example shows two sets of keys that are stored in the same key file: key1 and key2 in **key12.key**.

```
quartus_cpf --key

D:\SIV_DS\key12.key:key1:key2 D:\SIV_DS\test.sof D:\SIV_DS
\test.ekp
```

Generating Multi-Device .ekp File and Encrypting Configuration File using Intel Quartus Prime Software

To generate a multi-device **.ekp** file and encrypt your configuration file, follow these steps:

1. Start the Intel Quartus Prime software.
2. Repeat step 9 on page 16–step 11 on page 16 in Generating Single-Device .ekp File and Encrypting Configuration File using Intel Quartus Prime Software on page 14.
3. Click **Add File**. The **Select Programmer File** dialog box appears.
   a. Select the single-device **.ekp** file, and type <*single_ekp*>**.ekp** in the **File name** field.
   b. Click **Open**.

   Note: For the correct sequence of devices in the same JTAG chain, you can use the **Auto-Detect** option in the Intel Quartus Prime programmer. If one of the FPGA is not required to be key-programmed, you are not required to replace the device with the <*single_ekp*>**.ekp** file in the Intel Quartus Prime programmer.
4. Repeat step 3 on page 17 for each device in the same chain. Ensure the right device sequence is used when adding the **.ekp** files to the programmer window.
5. Highlight all the **.ekp** files you added and click **Program/Configure**.

6. On the File menu, point to **Create/Update** and click **Create JAM, SVF, or ISC File**. The **Create JAM, SVF, or ISC File** dialog box appears.
7. Select the required file format (**.jam**), for all the **.ekp** files in the **File format** field.
8. Type the file name in the **File name** field, or browse to and select the file.
9. Click **OK** to generate the **.jam** file.
10. On the Tools menu, click **Programmer Options**. The **Programmer Options** dialog box appears.

    Note: You must turn off **Configure volatile design security key** to generate a non-volatile **.svf** file of the **.ekp** file.
11. Click **OK**.
12. Repeat steps 7 on page 18 to 9 on page 18 to generate a **.svf** file for all the **.ekp** files. Use the default setting in the **Create JAM, SVF, or ISC File** dialog box when generating a **.svf** file of the **.ekp** file.

Step 2a: Programming Volatile Key into the FPGAs

Before programming the volatile key into the FPGAs, ensure that you can successfully configure the FPGA with an unencrypted configuration file. The volatile key is a reprogrammable and erasable key. Before you program the FPGAs with the volatile key, you must provide an external battery to retain the volatile key. FPGAs with the volatile key successfully programmed can accept both encrypted and unencrypted configuration bitstreams. This enables the use of unencrypted configuration bitstreams for board-level testing.

Any attempt to configure the FPGAs containing the volatile key with a configuration file encrypted with the wrong key causes the configuration to fail. If this occurs, the nSTATUS signal from the FPGA pulses low and continues to reset itself if you enable the **Auto-restart configuration after error** option in the Intel Quartus Prime software.

You can program the key into the FPGAs with on-board prototyping listed in Key Programming on page 7.

Step 2b: Programming Non-Volatile Key into the FPGAs

Before programming the non-volatile key into the devices, ensure that you can successfully configure the FPGA with an unencrypted configuration file. The non-volatile key is one-time programmable through the JTAG interface. You can program the non-volatile key into the devices without an external battery. Devices with the non-volatile key successfully programmed can accept both encrypted and unencrypted configuration bitstreams. If you set the tamper protection bit, only encrypted configuration bitstreams are accepted. This enables the use of unencrypted configuration bitstreams for board-level testing.

Any attempt to configure the FPGAs containing the volatile key with a configuration file encrypted with the wrong key causes the configuration to fail. If this occurs, the nSTATUS signal from the FPGA pulses low and continues to reset itself if you enable the **Auto-restart configuration after error** option in the Intel Quartus Prime software.

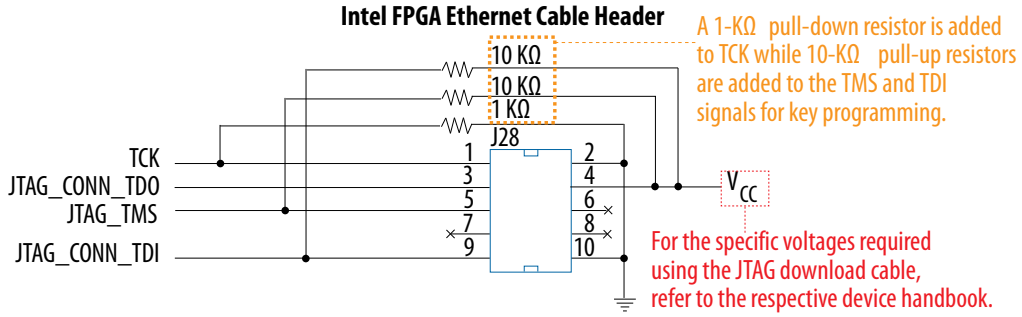
You can program the non-volatile key into the devices using on-board prototyping, volume production, and off-board prototyping and production listed in Key Programming on page 7.

Programming Volatile or Non-Volatile Key using Intel FPGA Ethernet Cable and Intel Quartus Prime Software

Connect the Intel FPGA Ethernet Cable communications cable to the Intel FPGA Ethernet Cable header as shown in the following figure.

Figure 2. Intel FPGA Ethernet Cable Header

The Intel FPGA Ethernet Cable header and Intel FPGA Download Cable header are identical for key programming.



Note: For Intel FPGA Ethernet Cable, set the TCK speed to the required TCK period.

Related Information

- EthernetBlaster Communications Cable User Guide
- EthernetBlaster II Communications Cable User Guide
  Provides more information about changing the TCK clock speed for Intel FPGA Ethernet Cable.
- Device Datasheet for Arria II Devices
  Provides more information about the specific voltages required using the JTAG download cable.
- DC and Switching Characteristics for Stratix IV Devices
  Provides more information about the specific voltages required using the JTAG download cable.
- Arria V Device Datasheet
  Provides more information about the specific voltages required using the JTAG download cable.
- Cyclone V Device Datasheet
  Provides more information about the specific voltages required using the JTAG download cable.
- Stratix V Device Datasheet
  Provides more information about the specific voltages required using the JTAG download cable.
- Intel Arria 10 Device Datasheet
  Provides more information about the specific voltages required using the JTAG download cable.

- Intel Cyclone 10 GX Device Datasheet
  Provides more information about the specific voltages required using the JTAG download cable.

Programming Single-Device Volatile or Non-Volatile Key using Intel Quartus Prime Software

To perform single-device volatile or non-volatile key programming using the Intel Quartus Prime software through the Intel FPGA Ethernet Cable, follow these steps:

1. Check the firmware version of the Intel FPGA Ethernet Cable. Verify that the JTAG firmware build are up-to-date.

   Note: Refer to the Cable and Adapter driver page to find the latest Intel FPGA Ethernet Cable firmware build version.
2. Start the Intel Quartus Prime software.
3. On the Tools menu, click **Programmer**. The **Programmer** dialog box appears.
4. In the **Mode** list, select **JTAG** as the programming mode.
5. Click **Hardware Setup**. The **Hardware Setup** dialog box appears.
   a. In the **Currently selected hardware** list, select **Intel FPGA Ethernet Cable** as the programming hardware.
   b. Click **Done**.
6. Click **Add File**. The **Select Programmer File** dialog box appears.
   a. Type <*filename*>**.ekp** in the **File name** field.
   b. Click **Open**.
7. Highlight the **.ekp** file you added and click **Program/Configure**.
8. On the Tools menu, click **Options**. The **Options** dialog box appears.
9. In the **Category** list, click **Programmer**. You can choose to turn on or turn off the **Configure volatile design security key** option to perform volatile or non-volatile key programming.
10. Click **OK** to close the **Options** dialog box.
11. Click **Start** to program the key.

    Note: The Intel Quartus Prime software message window provides information about the success or failure of the key programming operation.

Related Information

EthernetBlaster Communications Cable User Guide
Provides more information about JTAG firmware upgrade instructions.

Programming Single-Device Volatile or Non-Volatile Key using the Command-Line Interface in Intel Quartus Prime Software

To perform single-device volatile or non-volatile key programming using the Intel Quartus Prime software command-line interface through the Intel FPGA Ethernet Cable, follow these steps:

1. Perform step 1 on page 20 of Programming Single-Device Volatile or Non-Volatile Key using Intel Quartus Prime Software on page 20.
2. To determine the Intel FPGA Ethernet Cable cable port number that is connected to the JTAG server, type `quartus_jli -n` at the command-line prompt.
3. With the **single_ekp.jam** file generated in Step 1: Generating .ekp File and Encrypting Configuration File on page 13, execute volatile or non-volatile key programming to a single FPGA with the following command line:
   - Volatile key programming:

     ```
     quartus_jli -c<n> single_ekp.jam -aKEY_CONFIGURE
     ```
   - Non-volatile key programming:

     ```
     quartus_jli -c<n> single_ekp.jam -aKEY_PROGRAM
     ```

     <n> is the port number returned with the `-n` option.

   Note: The Intel Quartus Prime software command-line provides information about the success or failure of the key programming operation.

Related Information

AN 425: Using the Command-Line Jam STAPL Solution for Device Programming
Provides more information about `quartus_jli` .

Programming Multi-Device Volatile or Non-Volatile Key using Intel Quartus Prime Software

To perform multi-device volatile or non-volatile key programming using the Intel Quartus Prime software through the Intel FPGA Ethernet Cable, follow these steps:

1. Repeat step 1 on page 20–step 5 on page 20 in Programming Single-Device Volatile or Non-Volatile Key using Intel Quartus Prime Software on page 20.
2. Click **Add File**. The **Select Programmer File** dialog box appears.
   a. Programming using single-device **.ekp** files:
      i. Type <*single_device*>**.ekp** in the **File name** field.
      ii. Click **Open**.
      iii. Repeat steps 2.a.i on page 21 to 2.a.ii on page 21 for the number of devices in the same chain.
      iv. Highlight the **.ekp** files you added and click **Program/Configure**.

      Note: For the correct sequence of the devices in the same JTAG chain, you can use the **Auto-Detect** option in the Intel Quartus Prime programmer.
   b. Programming using a multi-device **.jam** file:
      i. Type <*multi_device*>**.jam** in the **File name** field.
      ii. Click **Open**.
      iii. Highlight the **.jam** file you added and click **Program/Configure**.
3. Repeat step 8 on page 20–step 10 on page 20 of Programming Single-Device Volatile or Non-Volatile Key using Intel Quartus Prime Software on page 20 to perform volatile or non-volatile key programming.
4. Click **Start** to program the key.

Note: The Intel Quartus Prime software message window provides information about the success or failure of the key programming operation.

Programming Multi-Device Volatile or Non-Volatile Key using the Command-Line Interface in Intel Quartus Prime Software

To perform multi-device volatile or non-volatile key programming using the Intel Quartus Prime software command-line interface through the Intel FPGA Ethernet Cable, follow these steps:

1. Perform step 1 on page 20 of Programming Single-Device Volatile or Non-Volatile Key using Intel Quartus Prime Software on page 20.
2. To determine the Intel FPGA Ethernet Cable cable port number that is connected to the JTAG server, type `quartus_jli -n` at the command-line prompt.
3. With the **multi_ekp.jam** file generated in Step 1: Generating .ekp File and Encrypting Configuration File on page 13, execute volatile or non-volatile key programming for multiple FPGAs with the following command line:
   - Volatile key programming:

     ```
     quartus_jli -c<n> multi_ekp.jam -aKEY_CONFIGURE
     ```
   - Non-volatile key programming:

     ```
     quartus_jli -c<n> multi_ekp.jam -aKEY_PROGRAM
     ```

     <n> is the port number returned with the `-n` option.

   Note: The Intel Quartus Prime software command-line provides information about the success or failure of the key programming operation.

Programming Key using JTAG Technologies

The key programming for your design is performed using a **.svf** file (**.ekp** file in **.svf** format) and a JT 37xx boundary scan controller in combination with a JT2147 QuadPod system.

Information about creating a .svf file to support multi-device programming is described in Generating Multi-Device .ekp File and Encrypting Configuration File using Intel Quartus Prime Software on page 17.

Related Information

JTAG Technologies
Provides more information about procedures for JTAG programming.

Step 3: Configuring the 40-nm, 28-nm, or 20-nm FPGAs with Encrypted Configuration Data

The final step is to configure the protected 40-nm, 28-nm, or 20-nm FPGAs with the encrypted configuration file.

During configuration, the encrypted configuration data is sent to the 40-nm, 28-nm, or 20-nm FPGAs. Using the previously stored key, the FPGA decrypts the configuration data and uses the unencrypted data to configure itself. Only configuration files encrypted using the correct key are accepted by the FPGA for successful configuration. Without a correct key, a stolen encrypted file is useless.

Steps to Enable Tamper-Protection Bit Programming

The default `.ekp` file generated in the *Steps for Implementing a Secure Configuration Flow* section contains only volatile or non-volatile key programming. To enable the tamper-protection bit programming, follow these steps:

1. Create a `quartus.ini` file using the text editor, with this key-value pair: `PGM_GEN_KEY_SECURE_EKP=ON`.
2. Save the `quartus.ini` in one of the following folders:
   - Project folder
   - <*Quartus installation folder*>\bin64 folder for Windows OS
   - <*Quartus installation folder*>/linux64 folder for Linux OS
3. When the Intel Quartus Prime Convert Programming File tool read the `quartus.ini` during `.ekp` file generation process, the additional tamper-protection bit programming instruction is inserted into the generated `.ekp` file.

   Caution: The `.ekp` file generated with this `quartus.ini` contain tamper-protection bit programming. When the `.ekp` file is used to program into the devices, the tamper-protection bit is programmed, and this programming is not reversible. You need to manage the `.ekp` file to avoid unintentional programming of tamper-protection bit into your device.

   As the `.ekp` file contains the tamper bit programming instruction, therefore if you generate `.jam` or `.svf` files from this `.ekp` file for key programming, the `.jam` or `.svf` files program the tamper-protection bit without the need for the `quartus.ini` with the specified key-value pair.

Supported Configuration Schemes

The design security feature is available in all configuration schemes except JTAG-based configuration.

Table 12. Design Security Support for Each Configuration Scheme

Configuration Scheme	Configuration Method	Design Security	Notes
FPP	A MAX II or MAX V device, or a microprocessor and a flash memory	Yes	In this mode, the host system must send a DCLK signal that is 4x the data rate.
AS	Serial configuration device	Yes	—
PS	A MAX II or MAX V device, or a microprocessor and a flash memory	Yes	—
	Intel FPGA Download Cable and Intel FPGA Download Cable II	Yes	Configure encrypted `.rbf` to FPGA using PS mode in Intel Quartus Prime Programmer.
JTAG	Intel FPGA Download Cable and Intel FPGA Download Cable II	—	For key programming.

If your system contains a common flash interface (CFI) flash memory, you can also use it for the FPGA configuration. The MAX II and MAX V together with the Parallel Flash Loader Intel FPGA IP core provides an efficient method to program CFI flash memory through the JTAG interface.

You can use the design security feature with other configuration features, such as the compression and remote system upgrade features. When compression is used with the design security feature, the configuration file is first compressed and then encrypted in the Intel Quartus Prime software. During configuration, the FPGA first decrypts and then uncompresses the configuration file.

Note: Encryption and compression cannot be used simultaneously in 20-nm FPGAs.

You can either perform boundary-scan test (BST) or use the Signal Tap logic analyzer to analyze functional data within the FPGA. However, you cannot perform JTAG configuration after the key with tamper-protection bit set is programmed into the 40-nm, 28-nm or 20-nm FPGAs.

When using the Signal Tap logic analyzer, you must first configure the device with an encrypted configuration file using PS, FPP, or AS configuration schemes. The design must contain at least one instance of the Signal Tap logic analyzer. After the FPGA is configured with a Signal Tap logic analyzer instance in the design. Open the Signal Tap logic analyzer window in the Intel Quartus Prime software and click **Scan Chain**. Once the scanning is complete, the Signal Tap logic analyzer is ready to acquire data using JTAG interface.

Related Information

- Configuration, Design Security, and Remote System Upgrades in Arria II Devices
  Provides more information about the design security for Arria II devices.
- Configuration, Design Security, and Remote System Upgrades in Stratix IV Devices
  Provides more information about the design security for Stratix IV devices.
- Configuration, Design Security, and Remote System Upgrades in Arria V Devices
  Provides more information about the design security for Arria V devices.
- Configuration, Design Security, and Remote System Upgrades in Cyclone V Devices
  Provides more information about the design security for Cyclone V devices.
- Configuration, Design Security, and Remote System Upgrades in Stratix V Devices
  Provides more information about the design security for Stratix V devices.
- Configuration, Design Security, and Remote System Upgrades in Intel Arria 10 Devices
  Provides more information about the design security for Intel Arria 10 devices.
- Configuration, Design Security, and Remote System Upgrades in Intel Cyclone 10 GX Devices
  Provides more information about the design security for Intel Cyclone 10 devices.

Security Mode Verification

Intel FPGAs support the KEY_VERIFY JTAG instruction that allows you to verify the existing security mode of the device. To check if you have successfully programmed the volatile key, use the `.jam` files to automate the security mode verification steps.

Table 13. KEY_VERIFY JTAG Instruction

JTAG Instruction	Instruction Code	Description
KEY_VERIFY	00 0001 0011	Connects the key verification scan register between TDI and TDO.

The KEY_VERIFY JTAG instruction allows you to read out the information on the security features that are enabled on the chip. This instruction scans out associated bit values.

Table 14. Security Mode Verification for 40-nm FPGAs

Security Mode	Supported Device	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5
No key	Arria II GX	0	0	0	0	0	0
	• Arria II GZ • Stratix IV	0	0	0	0	X	X
Volatile key	Arria II GX	1	0	0	0	0	0
	• Arria II GZ • Stratix IV	1	0	0	0	X	X
Volatile key with tamper protection	Arria II GX	1	0	0	0	1	0
	• Arria II GZ • Stratix IV	X	X	X	X	X	X
Non-volatile key	Arria II GX	0	1	0	1	0	0
	• Arria II GZ • Stratix IV	0	1	0	1	X	X
Non-volatile key with tamper protection bit	Arria II GX	0	1	1	1	0	0
	• Arria II GZ • Stratix IV	0	1	1	1	X	X

Table 15. Security Mode Verification for 28-nm FPGAs

Security Mode	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8
No key	0	0	0	0	0	X	X	X	X
Volatile key	1	0	0	0	0	X	X	X	X
Volatile key with tamper protection (14)	1	0	0	0	1	X	X	X	X
Non-volatile key	0	1	0	1	0	X	X	X	X
Non-volatile key with tamper protection bit (14)	0	1	1	1	0	X	X	X	X

(14) If the tamper protection is enabled, the device is in JTAG secure mode after power-up. You need to issue the UNLOCK to disable the JTAG secure mode.

Table 16. Security Mode Verification for 20-nm FPGAs

Bit	Security Feature or Settings	Description	Active value
0	Volatile Key	This bit is set when a volatile key has been successfully programmed into the device.	1
1	Attempt Non-volatile Key Programming	This bit is set to indicate that someone attempted to burn a non-volatile key in the OTP fused.	1
2	Disable Non-volatile Key	This bit is set to disable use of the volatile key.	1
3	Non-volatile Key	This bit is set to indicate that someone has successfully burned a non-volatile key into the OTP fuses.	1
4	Tamper Protection	This bit is set when FPGA is in Tamper Protection mode with either Non-volatile or Volatile key.	1
5	Don't Care	Don't Care.	X
6	Volatile Key Lock	This bit is set to prevent the volatile key from being reprogrammed from external JTAG.	1
7 - 10	Don't Care	Don't Care.	X
11 (15)	Force Configuration from HPS only	This bit is set when configuration is allowed from HPS only.	1
12	External JTAG Bypass	This bit is set to indicate that external JTAG is disabled.	1
13 (16)	HPS JTAG Bypass	This bit is set to indicate that HPS JTAG is disabled.	1
14 (17)	Disable Partial Reconfiguration and Scrubbing	This bit is set to indicate that external PR and external scrubbing (including HPS PR and HPS scrubbing) are disabled.	1
15	Disable Volatile Key	This bit is set to indicate that the volatile key is disabled.	1
16	Don't Care	Don't Care.	X
17	Disable Key Related JTAG Instructions	This bit is set to indicate that external JTAG access to all key-related JTAG instructions is disabled.	1
18	JTAG Secure Mode	This bit is set to indicate that only mandatory JTAG instructions are allowed to be externally accessed.	1
19	Don't Care	Don't Care.	X
20	Volatile Key Clear	This bit is set when the volatile key is successfully cleared from the device.	1

The following examples show the `.jam` files to verify the FPGAs security modes. The example `.jam` files are only applicable to single FPGA device in a JTAG chain. For SoC devices, add the following statements before the `IRSCAN` command.

```
PREIR 4;
PREDR 1;
```

(15) Bit 11 is not applicable to Intel Cyclone 10 GX devices. In Intel Cyclone 10 GX devices, this bit is in a "don't care" condition.

(16) Bit 13 is not applicable to Intel Cyclone 10 GX devices. In Intel Cyclone 10 GX devices, this bit is in a "don't care" condition.

(17) Bit 14 is not applicable to Intel Cyclone 10 GX devices. In Intel Cyclone 10 GX devices, this bit is in a "don't care" condition.

Example 3. JAM File for 40-nm FPGAs (Arria II GX Devices)

```
STATE RESET;
STATE IDLE;
'Security Mode Identification
BOOLEAN verify_reg[6];
IRSCAN 10, $013;
WAIT 100 USEC;
DRSCAN 6, $0, CAPTURE verify_reg[5..0];
```

Example 4. JAM File for 40-nm FPGAs (Arria II GZ and Stratix IV Devices)

```
STATE RESET;
STATE IDLE;
'Key Verification
BOOLEAN verify_reg[4];
IRSCAN 10, $013;
WAIT 100 USEC;
DRSCAN 4, $0, CAPTURE verify_reg[3..0];
```

Example 5. JAM File for 28-nm FPGAs

```
STATE RESET;
STATE IDLE;
'Key Verification in JAM format
BOOLEAN verify_reg[9];
IRSCAN 10, $013;
WAIT 100 USEC;
DRSCAN 9, $0, CAPTURE verify_reg[8..0];
```

Example 6. JAM File for 20-nm FPGAs

```
STATE RESET;
STATE IDLE;
'Key Verification in JAM format
BOOLEAN verify_reg[21];
```

```
IRSCAN 10, $013;

WAIT 100 USEC;

DRSCAN 21, $0, CAPTURE verify_reg[20..0];
```

Related Information

- Configuration, Design Security, and Remote System Upgrades in Arria II Devices
  Provides more information about the design security for Arria II devices.
- Configuration, Design Security, and Remote System Upgrades in Stratix IV Devices
  Provides more information about the design security for Stratix IV devices.
- Configuration, Design Security, and Remote System Upgrades in Arria V Devices
  Provides more information about the design security for Arria V devices.
- Configuration, Design Security, and Remote System Upgrades in Cyclone V Devices
  Provides more information about the design security for Cyclone V devices.
- Configuration, Design Security, and Remote System Upgrades in Stratix V Devices
  Provides more information about the design security for Stratix V devices.
- Configuration, Design Security, and Remote System Upgrades in Intel Arria 10 Devices
  Provides more information about the design security for Intel Arria 10 devices.
- Configuration, Design Security, and Remote System Upgrades in Intel Cyclone 10 GX Devices
  Provides more information about the design security for Intel Cyclone 10 devices.

Verification During JTAG Secure Mode

Non-mandatory JTAG instructions are disabled when the tamper protection bit is enabled in 28-nm FPGAs. When executing KEY_VERIFY during the tamper protection bit is programmed, TDI points to the BYPASS register. Due to this, executing the KEY_VERIFY instruction when the tamper protection bit has been set results in 0x0 (hex) being returned.

To check if the tamper protection bit has been programmed in a device, shift a user defined pattern in when executing the KEY_VERIFY instruction and check that the TDO pattern received has a 0 shifted in.

In 20-nm FPGAs, KEY_VERIFY instruction can be executed during JTAG Secure mode. To perform verification during JTAG secure mode for 20-nm FPGAs, you can expect 0x0 (hex) value being returned when executing USERCODE instruction.

Example 7. Verification During JTAG Secure Mode Example

Shift in 0x15A (1 0101 1010 in binary). If the tamper protection bit has been programmed, since KEY_VERIFY=BYPASS, you should expect 0 1011 0100 where the last 0 is the content of the BYPASS register.

Serial Flash Loader Support with Encryption Enabled

Intel provides an in-system programming (ISP) solution for serial configuration devices: the Serial Flash Loader Intel FPGA IP core. You can instantiate the serial flash loader (SFL) block in your design to provide the flexibility to update the design stored in the serial configuration device without reprogramming the configuration device through the AS interface.

As long as the JTAG interface of the FPGA is accessible, you can use the SFL solution for your application. If the design security feature with tamper-protection bit is set, the SFL solution does not work. Although the JTAG programming is not supported when the tamper-protection bit is set, you may instantiate the Serial Flash Loader IP core in your design and execute the SFL programming for the first time before non-volatile key programming with the tamper-protection bit is set in the FPGA.

Serial Flash Loader Support with Encryption Enabled for Single FPGA Device Chain

To use the Serial Flash Loader IP core with the encryption feature enabled in a single FPGA device chain, follow these steps:

1. Start the Intel Quartus Prime software.
2. Instantiate the Serial Flash Loader IP core in your FPGA top-level design.
3. Compile your design with one of the following options. An unencrypted **.sof** is generated.
   a. On the Processing menu, click **Start Compilation**; or
   b. On the Processing menu, point **Start** and click **Start Assembler**.
4. Follow these steps to convert a **.sof** to a **.jic** file:
   a. On the File menu, choose **Convert Programming Files**.
   b. In the **Convert Programming Files** dialog box, scroll to the **JTAG Indirect Configuration File (.jic)** from the **Programming file type** field.
   c. In the **Configuration device** field, specify the serial configuration device.
   d. In the **File name** field, browse to the target directory and specify an output file name.
   e. Highlight the **.sof** data in the **Input files to convert** section.
   f. Click **Add File**.
   g. Select the **.sof** file that you want to convert to a **.jic** file.
   h. Click **OK**.
   i. Click on the **.sof** file name to encrypt the **.sof** file.

      Note: To encrypt the .sof file, refer to step 7 on page 14 of Generating Single-Device .ekp File and Encrypting Configuration File using Intel Quartus Prime Software on page 14.
   j. Highlight Flash Loader and click **Add Device**.
   k. Click **OK**. The **Select Devices** page appears.
   l. Select the target FPGA that you are using to program the serial configuration device.

m. Click **OK**.

5. Program the serial configuration device with the encrypted **.jic** file.
6. Program the key into the FPGA device.

   Note: To program the key to a single FPGA device, follow the steps in Programming Single-Device Volatile or Non-Volatile Key using Intel Quartus Prime Software on page 20.
7. The encrypted FPGA is then configured by the programmed serial configuration device.

   Note: To program the key with a **.jam** file, you must convert the **.jic** file to a **.jam** file.

Related Information

- AN 370: Using the Intel FPGA Serial Flash Loader IP Core with the Intel Quartus Prime Software
- Device Datasheet for Arria II Devices
  Provides more information about the timing parameters for PS and FPP configuration schemes.
- DC and Switching Characteristics for Stratix IV Devices
  Provides more information about the timing parameters for PS and FPP configuration schemes.
- Arria V Device Datasheet
  Provides more information about the timing parameters for PS and FPP configuration schemes.
- Cyclone V Device Datasheet
  Provides more information about the timing parameters for PS and FPP configuration schemes.
- Stratix V Device Datasheet
  Provides more information about the timing parameters for PS and FPP configuration schemes.
- Intel Arria 10 Device Datasheet
  Provides more information about the timing parameters for PS and FPP configuration schemes.
- Intel Cyclone 10 GX Device Datasheet
  Provides more information about the timing parameters for PS and FPP configuration schemes.

JTAG Secure Mode for 28-nm and 20-nm FPGAs

FPGAs are in JTAG Secure mode upon power up when you:

- Enable the tamper-protection bit for 28-nm FPGAs
- Enable the JTAG Secure settings for 20-nm FPGAs

Attention: 20-nm FPGAs do not support LOCK and UNLOCK JTAG instructions, you are not able to unlock external JTAG to access non-mandatory JTAG instructions.

During JTAG secure mode, many JTAG instructions are disabled. The 28-nm and 20-nm FPGAs in JTAG secure mode only allow you to exercise mandatory IEEE Std. 1149.1 and IEEE Std. 1149.6 BST JTAG instructions. If you attempt to exercise a non-mandatory JTAG instruction when the FPGA is in the JTAG secure mode, the BYPASS JTAG instruction chain is selected and the instruction is not executed.

Table 17. Mandatory and Non-Mandatory IEEE Std. 1149.1 and IEEE Std. 1149.6 BST JTAG Instructions

Mandatory IEEE Std. 1149.1 and IEEE Std. 1149.6 BST JTAG Instructions	Non-Mandatory IEEE Std. 1149.1 and IEEE Std. 1149.6 BST JTAG Instructions
• BYPASS • EXTEST • IDCODE • LOCK • UNLOCK • SAMPLE/PRELOAD • SHIFT_EDERROR_REG	• CONFIG_IO • CLAMP • EXTEST_PULSE (18) • EXTEST_TRAIN (18) • HIGHZ • KEY_CLR_VREG • KEY_VERIFY (18) • PULSE_NCONFIG • USERCODE

For 28-nm FPGAs, to enable the access of non-mandatory JTAG instructions, you must issue the UNLOCK JTAG instruction to deactivate the JTAG secure mode. You can issue the LOCK instruction to put the device back into JTAG secure mode. You can only issue both the LOCK and UNLOCK JTAG instructions during user mode using internal JTAG interface. Issuing these two instructions using the external JTAG pins does not activate or deactivate the JTAG secure mode.

The LOCK and UNLOCK JTAG instructions only activate or deactivate the JTAG secure mode on an FPGA with tamper-protection bit enabled. Issuing these two instructions on a device that has a tamper-protection bit disabled does not turn on or turn off the JTAG secure mode.

Internal JTAG Interface

There are two interfaces to access the JTAG control block in 28-nm and 20-nm FPGAs: the external JTAG interface and the internal JTAG interface.

The external JTAG interface accesses the JTAG control block through the physical JTAG pins—TCK, TDI, TDO, and TMS. You use the external JTAG interface for FPGA configuration when using JTAG configuration scheme via programming cables or executing JTAG instructions using external player or processor such as JAM player or JTAG chain debugger tool. The internal JTAG interface refers to the connection between TCK, TDI, TDO, and TMS signals from the internal FPGA core fabric and the JTAG control block.

You can only access the JTAG control block using either one of these interfaces one at a time. For example, when you use the internal JTAG interface, the external JTAG interface to the JTAG control block is disabled. To access the internal JTAG interface, you must include the WYSIWYG atom in your Intel Quartus Prime design.

(18) You can execute these JTAG instructions during JTAG Secure mode for 20-nm FPGAs.

Figure 3. Internal and External JTAG Interface Connection

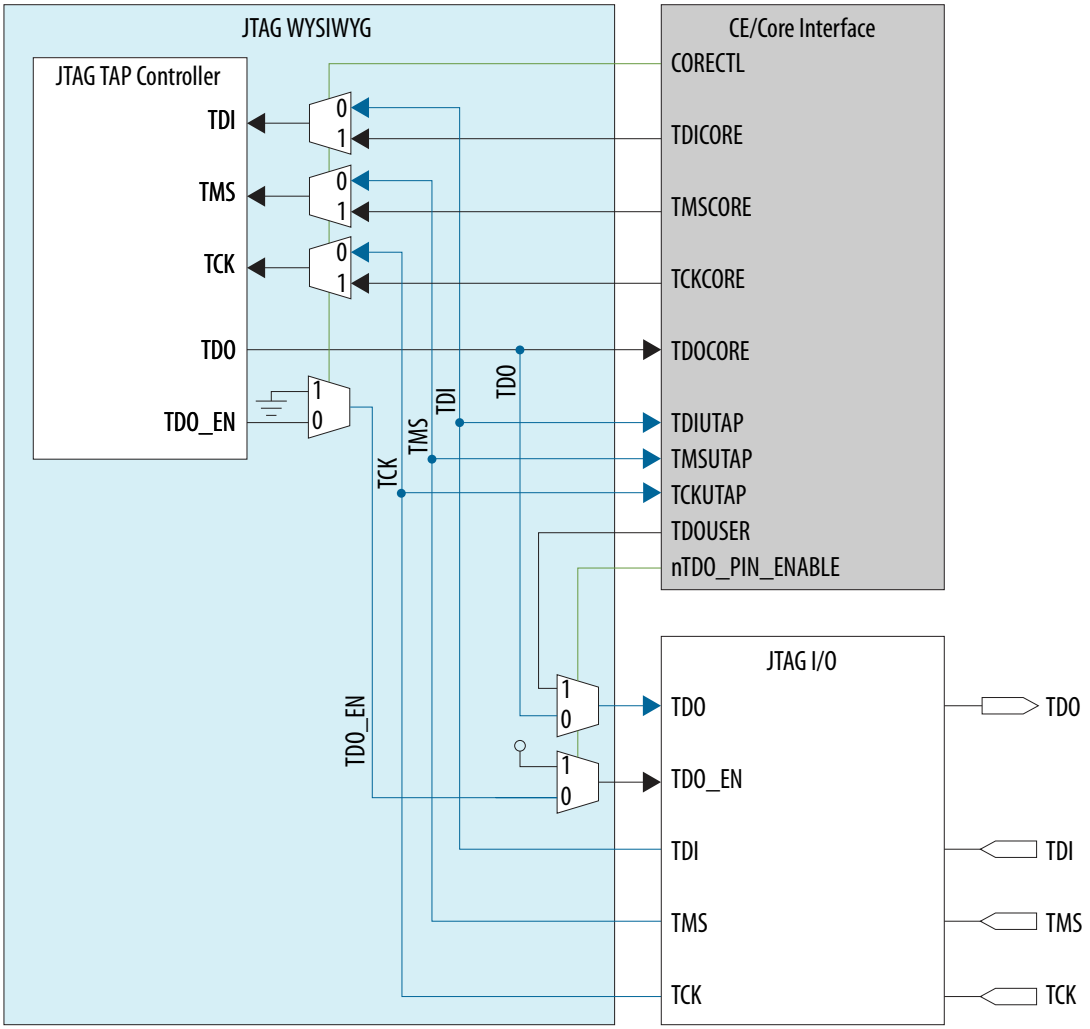


Table 18. WYSIWYG Atom for 28-nm and 20-nm FPGAs

Device Family	JTAG WYSIWYG Atom

Arria V	`arriav_jtag <jtagblock_name>`
	`(`
	`.clkdruser(),`
	`.corectl(),`
	`.runidleuser(),`
	`.shiftuser(),`
	`.tck(),`
	`.tckcore(),`
	`.tckutap(),`
	`.tdi(),`
	`.tdicore(),`
	`.tdiutap(),`
	`.tdo(),`
	`.tdocore(),`
	`.tdouser(),`
	`.tdoutap(),`
	`.tms(),`
	`.tmscore(),`
	`.tmsutap(),`

continued...

Device Family	JTAG WYSIWYG Atom
	.updateuser(), .usr1user());
Cyclone V	cyclonev_jtag <jtagblock_name> (.clkdruser(), .corectl(), .runidleuser(), .shiftuser(), .tck(), .tckcore(), .tckutap(), .tdi(), .tdicore(), .tdiutap(), .tdo(), .tdocore(), .tdouser(), .tdoutap(), .tms(), .tmscore(), .tmsutap(), .updateuser(), .usr1user());
Stratix V	stratixv_jtag <jtagblock_name> (.clkdruser(), .corectl(), .runidleuser(), .shiftuser(), .tck(), .tckcore(), .tckutap(), .tdi(), .tdicore(), .tdiutap(), .tdo(), .tdocore(), .tdouser(), .tdoutap(), .tms(), .tmscore(), .tmsutap(), .updateuser(), .usr1user());
Intel Arria 10	twentynm_jtag <jtagblock_name> (.tms(), .tck(), .tdi(), .ntrst(), .tdoutap(), .tdouser(), .tmscore(), .tckcore(), .tdicore(), .ntrstcore(), .tmscorehps(), .tckcorehps(), .tdicorehps(), .ntrstcorehps(), .tdocorefrwl(), .corectl(), .ntdopinena(), .tdo(), .tmsutap(), .tckutap(), .tdiutap(), .ntrstutap(), .tmsuhps(), .tckuhps(), .tdiuhps(), .ntrstuhps(), .tmscoreout(), .tckcoreout(), .tdocorehps(), .ntrstcoreout(),

continued...

Device Family	JTAG WYSIWYG Atom
	.shiftuser(), .clkdruser(), .updateuser(), .runidleuser(), .usr1user(), .tdocore(),);
Intel Cyclone 10 GX	twentynm_jtag <jtagblock_name> (.tms(), .tck(), .tdi(), .ntrst(), .tdoutap(), .tdouser(), .tmscore(), .tckcore(), .tdicore(), .ntrstcore(), .tdocorefrwl(), .corectl(), .ntdopinena(), .tdo(), .tmsutap(), .tckutap(), .tdiutap(), .ntrstutap(), .tmscoreout(), .tckcoreout(), .tdocorehps(), .ntrstcoreout(), .shiftuser(), .clkdruser(), .updateuser(), .runidleuser(), .usr1user(), .tdocore(),);

Table 19. Functions of the Ports in WYSIWYG Atom

Ports	Input/ Output	Functions
<jtagblock_name>	—	Identifier for the *arriaii_jtag* WYSIWYG atom and represents any identifier name that is legal for the given description language, such as Verilog HDL, VHDL, and AHDL.
.corectl()	Input	Active high input to the JTAG control block to enable the internal JTAG access from core interface. When the FPGA enters user mode after configuration, this port is low by default. Pulling this port to logic high enables the internal JTAG interface (with external JTAG interface disabled at the same time) and pulling this port to logic low disables the internal JTAG interface (with external JTAG interface enabled at the same time).
.tckcore()	Input	Core TCK signal. (19)
.tdicore()	Input	Core TDI signal. (19)
.tdocore()	Output	Core TDO signal. (19)
.tmscore()	Input	Core TMS signal. (19)
.clkdruser()	Input/Output	These ports are not used for enabling the JTAG secure mode using the internal JTAG interface, hence you can leave them unconnected.

continued...

(19) For external JTAG interface, refer to the respective device datasheet for the JTAG configuration timing specification. For internal JTAG interface, you must perform timing constraint and timing closure analysis on these paths to meet the setup or hold time requirement.

Ports	Input/ Output	Functions
.runidleuser()		
.shiftuser()		
.tck()		
.tckutap()		
.tdi()		
.tdiutap()		
.tdo()		
.tdouser()		
.tdoutap()		
.tms()		
.tmsutap()		
.updateuser()		
.usr1user()		

Related Information

- EthernetBlaster Communications Cable User Guide
- EthernetBlaster II Communications Cable User Guide
  Provides more information about changing the TCK clock speed for Intel FPGA Ethernet Cable.
- Device Datasheet for Arria II Devices
  Provides more information about the specific voltages required using the JTAG download cable.
- DC and Switching Characteristics for Stratix IV Devices
  Provides more information about the specific voltages required using the JTAG download cable.
- Arria V Device Datasheet
  Provides more information about the specific voltages required using the JTAG download cable.
- Cyclone V Device Datasheet
  Provides more information about the specific voltages required using the JTAG download cable.
- Stratix V Device Datasheet
  Provides more information about the specific voltages required using the JTAG download cable.
- Intel Arria 10 Device Datasheet
  Provides more information about the specific voltages required using the JTAG download cable.
- Intel Cyclone 10 GX Device Datasheet
  Provides more information about the specific voltages required using the JTAG download cable.
- Stratix V E, GS, and GX Device Family Pin Connection Guidelines

- Stratix V GT Device Family Pin Connection Guidelines
- Arria V GT, GX, ST and SX Device Family Pin Connection Guidelines
- Arria V GZ Device Family Pin Connection Guidelines
- Stratix IV GX and E Device Family Pin Connection Guidelines
- Stratix IV GT Device Family Pin Connection Guidelines
- Arria II Device Family Pin Connection Guidelines
- Intel Arria 10 GX, GT, and SX Device Family Pin Connection Guidelines

Design Example for JTAG Secure Mode

This design example demonstrates

- The instantiation of an internal JTAG WYSIWYG atom.
- The execution of the LOCK and UNLOCK JTAG instructions through user logic implementation in the Intel Quartus Prime software.

This reference design is targeted on the Arria V device with the tamper-protection bit enabled. This design example is applicable to other 28-nm FPGAs.

Related Information

AN 556 Design Files

Design Example Intel Quartus Prime Design Components

Table 20. Intel Quartus Prime Design Components for the Arria V Device

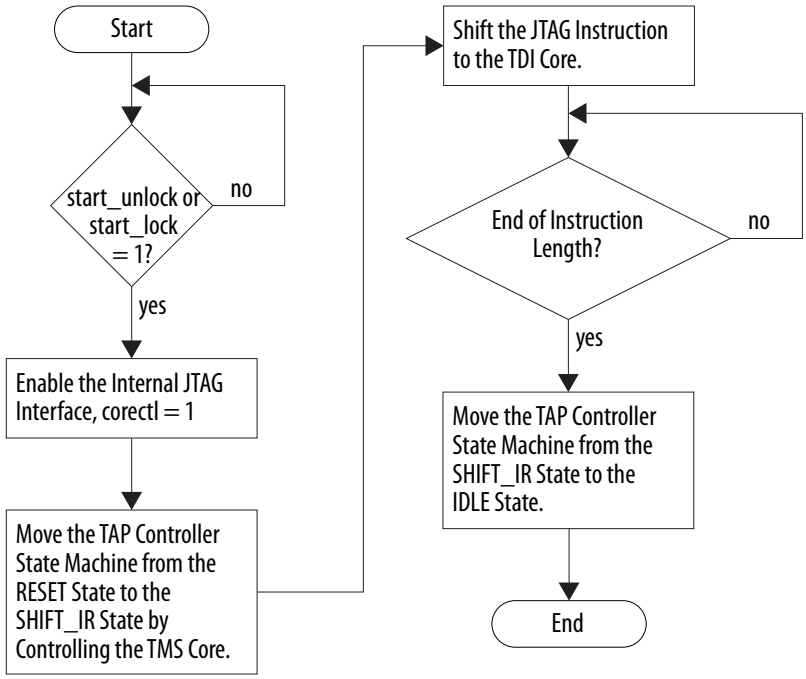
Component	Function and Description
JTAG_Lock_Unlock.bdf	The top entity of the reference design.
JTAG_Lock_Unlock_wysiwyg.v	The Verilog code for the Arria V device WYSIWYG atom instantiation. You need to modify this code according to Table 18 on page 32 for compliance with other 28-nm FPGAs.
ALTINT_OSC.v	A IP core instantiation of an internal oscillator clock source. In this reference design, the clock source from the internal oscillator is used to drive the user logic to eliminate the need of an external clock source.
User_logic_control_block.v	An example Verilog file that executes JTAG instructions using Arria V device WYSIWYG atom. You can modify this code to fit your design requirements and restrictions, or replace this code with another similar implementation.
Pulse_nconfig.jam	Use this JAM file to execute the PULSE_NCONFIG JTAG instruction to verify the JTAG secure mode as shown in Verifying JTAG Secure Mode on page 38. This file is optional and can be replaced with other methods to verify the JTAG secure mode.

LOCK and UNLOCK JTAG Instructions

When you configure this reference design into an Arria V device with the tamper-protection bit enabled, the Arria V device is in JTAG secure mode after power up and configuration, whereby you can only execute mandatory JTAG instructions.

To disable the JTAG secure mode, you can trigger the `start_unlock` port of the user logic to issue the UNLOCK JTAG instruction. After the `start_unlock` port goes high, the UNLOCK JTAG instruction is issued. After the UNLOCK JTAG instruction is issued, the device exits from JTAG secure mode, whereby both mandatory and non-mandatory JTAG instructions are allowed.

Figure 4. LOCK or UNLOCK JTAG Instruction Execution



The `start_lock` port in the user logic triggers the execution of the LOCK JTAG instruction. The function of the LOCK JTAG instruction is to put the device back into JTAG secure mode.

Table 21. Input and Output Port of the User Logic

Port	Input/Output	Function
`clk_in`	Input	Clock source for the user logic. The f_{MAX} of the user logic depends on the timing closure analysis. You need to apply timing constraints and perform timing analysis on the path to determine the f_{MAX}.
`start_lock`	Input	Logic high to trigger the execution of the LOCK JTAG instruction to the internal JTAG interface.
`start_unlock`	Input	Logic high to trigger the execution of the UNLOCK JTAG instruction to the internal JTAG interface.
`jtag_core_en_out`	Output	Output of the `User_logic_control_block`. This port is connected to the `corectl` port of the JTAG WYSIWYG atom to enable the internal JTAG interface.
`tck_out`	Output	Output of the `User_logic_control_block`. This port is connected to the `tck_core` port of the JTAG WYSIWYG atom.

continued...

Port	Input/Output	Function
tdi_out	Output	Output of the User_logic_control_block. This port is connected to the tdi_core port of the JTAG WYSIWYG atom.
tms_out	Output	Output of the User_logic_control_block. This port is connected to the tms_core port of the JTAG WYSIWYG atom.
indicator	Output	Logic high on this output pin indicates the completion of the LOCK or UNLOCK JTAG instruction execution.

Related Information

AN 39: IEEE 1149.1 JTAG Boundary-Scan Testing in Altera Devices

Verifying JTAG Secure Mode

Intel recommends that you verify whether your device has successfully enter or exit JTAG secure mode by executing the non-mandatory JTAG instructions. To validate the JTAG secure mode with the reference design [20], follow these steps:

1. FPGA power up

   After the FPGA is powered up, the FPGA is in the JTAG secure mode because the tamper-protection bit is enabled.

2. FPGA configuration

   Configure the reference design into the FPGA. Since the FPGA is tamper resistant and accepts only encrypted configuration file, you need to configure the reference design in encrypted file as shown in Step 3: Configuring the 40-nm, 28-nm, or 20-nm FPGAs with Encrypted Configuration Data on page 22. To ensure the device enters user mode successfully, you can check the CONFDONE pin or observe the counter_output pin. If the device enters user mode successfully, the CONFDONE pin goes high and the counter_output pin should toggle.

3. Verify the JTAG secure mode

   After the device enters user mode, issue the PULSE_NCONFIG JTAG instruction using the external JTAG pins. You can use the pulse_nconfig.jam file attached in the design example. To execute the pulse_nconfig.jam file, you can use the quartus_jli or the JAM player. The PULSE_NCONFIG JTAG instruction triggers device reconfiguration. If your device is in the JTAG secure mode, reconfiguration is not taking place because the PULSE_NCONFIG JTAG instruction is a non-mandatory JTAG instruction. You can confirm this by observing the CONFDONE pin and the counter_output pin. If reconfiguration did not take place, the CONFDONE pin stays high and the counter_output pin continues to toggle.

4. Execute the UNLOCK JTAG instruction

   Pull the start_unlock port of the user logic to logic high. After the UNLOCK JTAG instruction is complete, the indicator port goes high.

5. Verify the JTAG secure mode

   After the UNLOCK JTAG instruction is completed, issue the PULSE_NCONFIG JTAG instruction again using the external JTAG pins. If your device is not in the JTAG secure mode, the PULSE_NCONFIG JTAG instruction triggers device

[20] You should only apply these steps on an FPGA with the tamper-protection bit enabled.

reconfiguration. You can observe the CONFDONE pin and the `counter_output` pin to monitor the device reconfiguration. The CONFDONE pin goes from high to low and the `counter_output` pin stops toggling during device reconfiguration.

Related Information

AN 425: Using the Command-Line Jam STAPL Solution for Device Programming
Provides more information about `quartus_jli` .

Document Revision History for AN 556: Using the Design Security Features in Intel FPGAs

Document Version	Changes
2021.05.21	Added a note to *Generating Single-Device .ekp File and Encrypting Configuration File using Command-Line Interface in Intel Quartus Prime Software*.
2019.11.12	Updated the *Key Programming* section: • Updated the footnote for Intel FPGA Download Cable II. • Added a note to state that the JTAG TCK pulse width (period) for other third-party non-volatile key programming must be regulated for proper polyfuse programming.
2018.12.11	Updated the *Generating Single-Device .ekp File and Encrypting Configuration File using Intel Quartus Prime Software* section to correct the key file examples.
2018.06.15	• Updated the description in the *Overview of the Design Security Feature* section. • Updated the description in the *Design Security Approach for 40-nm and 28-nm FPGAs* table. • Removed the note to the design protection option in the *Volatile and Non-Volatile Key Comparison* table. • Corrected the note to Intel FPGA Parallel Port Cable in the *Key Programming Methods* table. • Updated the description for `--decrypt` in the *Basic Options in Qcrypt Tool* table. • Updated the information on encrypting an `.rbf` by using the stand-alone Qcrypt tool in the *Steps for Implementing a Secure Configuration Flow* section. • Added steps in the *Generating Single-Device .ekp File and Encrypting Configuration File using Intel Quartus Prime Software* section. • Added the *Steps to Enable Tamper-Protection Bit Programming* section. • Updated the PS and JTAG configuration schemes in the *Design Security Support for Each Configuration Scheme* table. • Added description for the examples in the *Security Mode Verification* section. • Updated the *Internal and External JTAG Interface Connection* diagram. • Corrected the functions for the `jtag_core_en_out`, `tck_out`, `tdi_out`, and `tms_out` ports in the *Input and Output Port of the User Logic* table. • Renamed the following IP cores as per Intel rebranding: — Renamed Intel FPGA Parallel Flash Loader IP core to Parallel Flash Loader Intel FPGA IP core. — Renamed Intel FPGA Serial Flash Loader IP core to Serial Flash Loader Intel FPGA IP core.

Date	Version	Changes
December 2017	2017.12.18	• Added support for Intel Cyclone 10 GX device family. • Updated the "Specifications for Key Programming" table: Updated the Non-Volatile Key and Volatile Key descriptions for TCK period. • Updated the "Security Mode Verification for 20-nm FPGAs" table: Added footnotes for bits 11 and 13 to clarify that these bits are not applicable Intel Cyclone 10 GX devices.

continued...

Date	Version	Changes
		• Updated for latest Intel branding standards. • Updated `--lockto=<FILE_NAME.qlk>` security option description in *Security Options in Qcrypt Tool* table. • Made minor text edits to the document.
June 2016	2016.06.01	• Added *Arria 10 Qcrypt Tool* information. • Added information about tamper-protection bit and JTAG Secure can be enabled separately in 20-nm FPGAs. • Added software requirements for 20-nm FPGAs. • Added two additional steps for 20-nm FPGAs in *Generating Single-Device .ekp File and Encrypting Configuration File using Intel Quartus Prime Software*. • Added 20-nm FPGA JTAG Secure verification methods. • Added note about EXTEST_PULSE, EXTEST_TRAIN, KEY_VERIFY JTAG instructions can be used during JTAG Secure mode. • Updated Arria 10 JTAG atom. • Added separate security approach for 20-nm FPGAs. • Added note about encryption and compression cannot be used simultaneously in 20-nm FPGAs. • Updated TCK period non-volatile key specification for 20-nm FPGAs. • Added note about USB-Blaster supports volatile and non-volatile key for 20-nm FPGAs.
November 2015	2015.11.02	• Added note about user need set the TCK speed to required TCK period for EthernetBlaster II and added link EthernetBlaster II Communications Cable User Guide. • Changed instances of *Quartus II* to *Quartus Prime*.
June 2015	2015.06.15	Added link to JTAG Secure Mode Design Example.
May 2015	2015.05.04	Corrected the total number of character in .key file example.
January 2015	2015.01.23	• Added 20-nm FPGAs (Arria 10) support. • Added JAM file example for 20-nm. • Added Security Mode Verification for 20-nm table. • Added JTAG WYSIWYG atom for Arria 10. • Added AES modes in Altera FPGAs.
December 2014	2014.12.15	Added USB-Blaster II support for non-volatile security key programming.
September 2014	2014.09.30	• Added example .key file in How to Generate the Single-Device .ekp File and Encrypt the Configuration File using Quartus II Software. • Removed V_{CCBAT} voltage guideline and added device family pin connection guidelines links for updated values in Hardware Requirements. • Added note to modes with tamper protection in Security Mode Verification for 28-nm FPGAs. • Added Verification During JTAG Secure Mode subsection to tamper bit protection settings during JTAG Secure mode.
May 2014	2014.05.19	Updated the Non-Volatile and Volatile Key Storage section to include information on using valid MSEL pin settings.
June 2013	2013.06.19	• Updated the Design Security Approach for FPGAs table to include more design security features. • Updated the Non-Volatile and Volatile Key Storage section to include details on both volatile and non-volatile key storage. • Updated the Key Programming section to include support for both 28-nm and 40-nm FPGAs using the System General programming tool. • Updated the Hardware Requirements section to update the Specifications for Key Programming table.

continued...

Date	Version	Changes
		• Updated the Steps for Implementing a Secure Configuration Flow section. • Updated the Step 1: Generate the .ekp File and Encrypt Configuration File, Step 2a: Program the Volatile Key into the FPGAs, and Step 2b: Program the Volatile Key into the FPGAs sections. • Updated the How to Generate the Single-Device .ekp File and Encrypt the Configuration File using Quartus II Software to include information about the encryption key for 28-nm and 40-nm FPGAs. • Updated the Security Mode Verification section to update the security mode and its associated bit values for both 28-nm and 40-nm FPGAs. • Updated the JTAG Secure Mode for 28-nm FPGAs section to include more information about the mandatory and non-mandatory JTAG instructions, internal JTAG interface and external JTAG interface, WYSIWYG atom functions, and design example for JTAG secure mode. • Moved all links in all topics to the Related Information section for easy reference.
June 2012	2.1	• Updated Table 1 and Table 3. • Updated **.ekp** file verification error information. • Updated "Hardware Requirements" section.
June 2011	2.0	• Updated application note for the Quartus II software version 11.0 release. • Changed the specific device names to 40- or 28-nm FPGAs. • Added "Security Mode Verification" and "JTAG Secure Mode for 28-nm FPGAs" sections. • Added Table 1. • Added Table 5. • Added Example 3, Example 4, and Example 5. • Updated Figure 1. • Minor text edits.
June 2009	1.1	• Updated "Introduction" on page 1. • Updated "Overview of the Design Security Feature" on page 2. • Updated "Security Encryption Algorithm" on page 2. • Updated "Non-Volatile and Volatile Key Storage" on page 3. • Updated (Note 3) of Table 2 on page 4. • Updated "Hardware and Software Requirements" on page 4. • Updated (Note 1) of Table 3 on page 5. • Updated "Steps for Implementing a Secure Configuration Flow" on page 5. • Updated "Step 2a: Program the Volatile Key into the Arria II GX or Stratix IV Devices" on page 17. • Updated "Step 2b: Program the Non-Volatile Key into the Arria II GX or Stratix IV Devices" on page 18. • Updated "Step 3: Configure the Arria II GX or Stratix IV Devices with Encrypted Configuration Data" on page 24. • Added Table 3 on page 28. • Updated Figure 1 on page 6 and Figue 26 on page 29.
March 2009	1.0	Initial release.