This application note describes how you can use the design security features in Altera 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**

<table>
<thead>
<tr>
<th>FPGA</th>
<th>Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 nm</td>
<td>Arria® II and Stratix® IV</td>
</tr>
<tr>
<td>28 nm</td>
<td>Arria V, Cyclone® V, and Stratix V</td>
</tr>
<tr>
<td>20 nm</td>
<td>Arria 10</td>
</tr>
</tbody>
</table>

In the highly competitive 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 ever more important to protect the designs from unauthorized copying, reverse engineering, and tampering. FPGAs address concerns with the ability to decrypt a configuration bitstream using the 256-bit Advanced Encryption Standard (AES) algorithm, an industry standard encryption algorithm.

**Table 2: AES Modes in Supported Altera FPGAs**

<table>
<thead>
<tr>
<th>FPGA</th>
<th>AES Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 nm</td>
<td>Counter Mode (CTR)</td>
</tr>
<tr>
<td>28 nm</td>
<td>Cipher-block chaining (CBC)</td>
</tr>
<tr>
<td>20 nm</td>
<td>CTR</td>
</tr>
</tbody>
</table>

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. You can use the intercepted configuration data to configure another FPGA.

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

- 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 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 Arria 10 Devices provides more information about the design security for Arria 10 devices.

Overview of the Design Security Feature

The design security feature for FPGAs protects against unauthorized copying, reverse engineering, and tampering.

The following table lists some of the design approaches to make the solution secure.

Table 3: Design Security Approach for FPGAs

For 28- and 20-nm FPGAs, if you enable the tamper-protection bit, the device is in JTAG secure mode.

<table>
<thead>
<tr>
<th>Design Security Element</th>
<th>40-nm FPGA</th>
<th>28-nm FPGA</th>
<th>20-nm FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Volatile key</td>
<td>The non-volatile key is securely stored in fuses within the device. Proprietary security features make it difficult to determine this key.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Volatile Key</td>
<td>The volatile key is securely stored in battery-backed RAM within the device. Proprietary security features make it difficult to determine this key.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key Generation</td>
<td>Two user provided 256-bit strings are processed to generate a 256-bit key that is programmed into the device.</td>
<td>A user provided 256-bit key is processed by a one-way function before being programmed into the device.</td>
<td></td>
</tr>
<tr>
<td>Design Security Element</td>
<td>40-nm FPGA</td>
<td>28-nm FPGA</td>
<td>20-nm FPGA</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Tamper Protection Mode</td>
<td>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 will result in a configuration failure. You can enable this mode by setting a fuse within the device.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configuration Readback</td>
<td>These devices do not support a configuration readback feature. From a security perspective, this makes readback of your unencrypted configuration data infeasible.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For additional details on these and other security features, contact your Altera sales representative.

**Security Encryption Algorithm**

FPGAs have a dedicated decryption block that uses the AES algorithm to decrypt configuration data using a user-defined 256-bit key. Prior to receiving the encrypted data, you must write the user-defined 256-bit key into the device.

The AES algorithm is a symmetrical block cipher that encrypts and decrypts data in blocks of 256 bits. The encrypted data is subject to a series of transformations that includes byte substitutions, data mixing, data shifting, and key additions.

FPGAs contain an AES decryptor block that uses the AES algorithm to decrypt the configuration data prior to configuring the FPGA device. 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**

  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**

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.

**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 4: Volatile and Non-Volatile Key Comparison

<table>
<thead>
<tr>
<th>Option</th>
<th>Volatile Key</th>
<th>Non-Volatile Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Length</td>
<td>256 bits</td>
<td>256 bits</td>
</tr>
<tr>
<td>Key Programmability</td>
<td>Reprogrammable and erasable key</td>
<td>One-time programmable key</td>
</tr>
<tr>
<td>External Battery</td>
<td>Required</td>
<td>Not required</td>
</tr>
<tr>
<td>Key Programming Method(1)</td>
<td>On-board</td>
<td>Both on-board and off-board (2)</td>
</tr>
<tr>
<td>Design Protection(3)</td>
<td>Secure against copying, reverse engineering, and tampering(4)</td>
<td></td>
</tr>
</tbody>
</table>

**Caution:** Enabling the tamper-protection bit disables the test mode in FPGAs. Disabling the test mode is irreversible and prevents Altera from carrying out failure analysis. To enable the tamper protection bit, contact Altera Technical Support.

**Related Information**
**Altera Technical Support**
To enable tamper-protection bit.

**Key Programming**

Table 5: Key Programming Methods

<table>
<thead>
<tr>
<th>Programming Procedure</th>
<th>Method</th>
<th>Programming Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-Board Programming</td>
<td>Prototyping</td>
<td>EthernetBlaster, JTAG Technologies, ByteBlaster™ II, USB-Blaster™ II, and USB-Blaster II</td>
</tr>
<tr>
<td></td>
<td>Production</td>
<td>JTAG Technologies</td>
</tr>
<tr>
<td>Off-Board Programming</td>
<td>Prototyping</td>
<td>System General</td>
</tr>
<tr>
<td></td>
<td>Production</td>
<td>System General</td>
</tr>
</tbody>
</table>

(1) Key programming is carried out through JTAG interface. You need to use valid MSEL pin settings for the JTAG interface.
(2) 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.
(3) The volatile key tamper-protection is only available for Arria II, Arria V, Cyclone V, and Stratix V devices.
(4) Tampering is prevented only when the volatile key tamper-protection bit is set, thus preventing configuration with unencrypted Programmer Object Files (.pof).
(5) ByteBlaster II and USB-Blaster support only volatile key programming. EthernetBlaster and JTAG Technologies support both volatile and non-volatile key programming. For non-volatile key programming, you must regulate the JTAG TCK pulse width (period) for proper polyfuse 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

### Related Information

**Altera Technical Support**
Provides information about programming support.

## Hardware and Software Requirements

This section provides the hardware and software requirements for the FPGAs design security feature. When using this 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 6: Specifications for Key Programming

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Key Programming Mode</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Non-Volatile Key</td>
<td>Volatile Key</td>
</tr>
<tr>
<td>TCK period</td>
<td>10 µs ± 1 µs</td>
<td>(6)</td>
</tr>
<tr>
<td>Ambient Temperature</td>
<td>25°C ± 5°C</td>
<td>25°C ± 5°C</td>
</tr>
<tr>
<td>Voltage (V\textsubscript{CCBAT})</td>
<td>—</td>
<td>(7)</td>
</tr>
</tbody>
</table>

V\textsubscript{CCBAT} is a dedicated power supply for the volatile key storage and is not shared with other on-chip power supplies, such as V\textsubscript{CCIO} or V\textsubscript{CC}. V\textsubscript{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 (\texttt{ekp}) file if you have the V\textsubscript{CCBAT} pin tied to GND. The V\textsubscript{CCBAT} pin must be tied to the recommended V\textsubscript{CCBAT} voltage provided in the respective device family pin connection guidelines for proper operation.

---

(6) For the volatile key programming TCK period specification, refer to the JTAG Specification section in the respective device datasheet.

(7) If you do not use the volatile key, refer to the respective device family pin connection guidelines for V\textsubscript{CCBAT} connection.
Software Requirements

To enable the design security feature of 40-nm FPGAs, you must use the Quartus® II software version 9.0 or later.

To enable the design security feature of 28-nm FPGAs, you must use the Quartus II software version 11.0 or later.

To enable the design security feature of 20-nm FPGAs, you must use the Quartus II software version 14.0a10s or later.

To enable the design security feature for Quartus II web edition, obtain a license file from Altera Technical Support.

Related Information

Altera 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 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: Generate the .ekp File and Encrypt Configuration File on page 7.

2. Program the user-defined 256-bit key into the FPGAs.
   For details, refer to Step 2a: Program the Volatile Key into the FPGAs on page 11 and Step 2b: Program the Non-Volatile Key into the FPGAs on page 12.

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 FPGAs. The devices use 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: Configure the 40-nm, 28-nm or 20-nm FPGAs with Encrypted Configuration Data on page 16.

**Figure 1: Secure Configuration Flow**

![Secure Configuration Flow Diagram]

---

**Step 1: Generate the .ekp File and Encrypt Configuration File**

To use the design security feature in the FPGAs, you must generate an .ekp file and encrypt your configuration files using the Quartus Prime software. The key is not saved into any 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 Altera 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 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 generated automatically from the Quartus Prime software. You must create the .jam and .svf files using the Quartus Prime software if these files are required in the key programming. The Quartus Prime software generates the JBC format of the .ekp file in the same project directory.

**Note:** Altera recommends that you keep the .ekp file confidential.
Use the .ekp file with the EthernetBlaster communications cable or USB-Blaster download cable and the Quartus Prime software. The EthernetBlaster communications cable can support both volatile and non-volatile key programming whereas the USB-Blaster 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.

Related Information

Altera Technical Support
To enable the design security feature, you must obtain a license file.

How to Generate the Single-Device .ekp File and Encrypt the Configuration File using 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 Altera Technical Support.
2. Start the 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.
   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, click on 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. 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 ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789

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.

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

2. Click OK.

8. In the Convert Programming Files dialog box, click OK. The <filename>.ekp and encrypted configuration file are generated in the same project directory.


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 EthernetBlaster 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.

   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 step 15–step 17 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.

How to Generate the Single-Device .ekp File and Encrypt Configuration File Using Quartus Prime Software with the Command-Line Interface

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 interface uses the 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>
```

You can learn more on the option file from the Quartus Prime software command line help. Run quartus_cpf --help=option to learn more on the available options.

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
```
How to Generate the Multi-Device .ekp File and Encrypt the Configuration File using QuartusPrime Software

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

1. Start the Quartus Prime software.
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 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 Quartus Prime programmer.

4. Repeat step 3 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.

    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 step 7–step 9 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: Program the 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 Quartus Prime software.

You can program the key into the FPGAs with on-board prototyping listed in Key Programming on page 4.
Step 2b: Program the 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 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 4.

Volatile or Non-Volatile Key Programming Using EthernetBlaster and Quartus Prime Software

Connect the EthernetBlaster communications cable to the EthernetBlaster header as shown in the following figure.

Figure 2: EthernetBlaster Header

The EthernetBlaster header and USB-Blaster header are identical for key programming.

Note: For Ethernet Blaster II, please 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 how to change the TCK clock speed for Ethernet Blaster II.
- 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.
How to Perform Single-Device Volatile or Non-Volatile Key Programming Using Quartus Prime Software

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

1. Check the firmware version of the EthernetBlaster. Verify that the JTAG firmware build number is 101 or greater. If the version precedes build number 101, apply the firmware upgrade.
   
   **Note:** Apply the firmware upgrade (EBFW100101.tar.gz) to the EthernetBlaster unit. This updates the JTAG Firmware to Build 101.

2. Start the 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 **EthernetBlaster** 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 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.
**How to Perform Single-Device Volatile or Non-Volatile Key Programming Using Quartus Prime Software with the Command-Line Interface**

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

2. To determine the EthernetBlaster 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: Generate the .ekp File and Encrypt Configuration File on page 7, 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 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`.

**How to Perform Multi-Device Volatile or Non-Volatile Key Programming Using Quartus Prime Software**

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

2. Click Add File. The Select Programmer File dialog box appears.
   a. Programming using single-device .ekp files:
      1. Type `<single_device>.ekp` in the File name field.
      2. Click Open.
      3. Repeat step 2.a.i on page 14–step 2.a.ii on page 14 for the number of devices in the same chain.
      4. 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 Quartus Prime programmer.

b. Programming using a multi-device .jam file:
1. Type `<multi_device>.jam` in the File name field.
2. Click Open.
3. Highlight the .jam file you added and click Program/Configure.

4. Repeat step 8 on page 13–step 10 on page 13 of How to Perform Single-Device Volatile or Non-Volatile Key Programming Using Quartus Prime Software on page 13 to perform volatile or non-volatile key programming.

4. Click Start to program the key.

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

How to Perform Multi-Device Volatile or Non-Volatile Key Programming Using Quartus Prime Software with the Command-Line Interface

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

2. To determine the EthernetBlaster 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: Generate the .ekp File and Encrypt Configuration File on page 7, 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 Quartus Prime software command-line provides information about the success or failure of the key programming operation.

Key Programming 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 How to Generate the Multi-Device .ekp File and Encrypt the Configuration File using Quartus Prime Software on page 11.

Related Information
JTAG Technologies
Provides more information about procedures for JTAG programming.
Step 3: Configure 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.

Supported Configuration Schemes

The design security feature is available in all configuration schemes except JTAG-based configuration. Therefore, you can use the design security feature in FPP, AS, and PS configuration schemes.

Table 7: Design Security Support for Each Configuration Scheme

<table>
<thead>
<tr>
<th>Configuration Scheme</th>
<th>Configuration Method</th>
<th>Design Security</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPP</td>
<td>A MAX II or MAX V device, or a microprocessor and a flash memory</td>
<td>Yes</td>
<td>In this mode, the host system must send a DCLK signal that is 4x the data rate.</td>
</tr>
<tr>
<td>AS</td>
<td>Serial configuration device</td>
<td>Yes</td>
<td>—</td>
</tr>
<tr>
<td>PS</td>
<td>A MAX II or MAX V device, or a microprocessor and a flash memory</td>
<td>Yes</td>
<td>—</td>
</tr>
<tr>
<td>JTAG</td>
<td>Download cable</td>
<td>Yes</td>
<td>The MicroBlaster™ tool is required to execute encrypted PS configuration using a .rbf through ByteBlaster II or ByteBlasterMV™ download cable.</td>
</tr>
</tbody>
</table>

In addition, 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 parallel flash loader (PFL) feature 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 Quartus Prime software. During configuration, the FPGA first decrypts and then uncompresses the configuration file.

You can either perform boundary-scan test (BST) or use the SignalTap™ II 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 SignalTap II 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 SignalTap II logic analyzer. After the FPGA is configured with a SignalTap II logic analyzer instance in the design and after the SignalTap II logic analyzer window is opened in the Quartus Prime software, scan the chain and the SignalTap II logic analyzer is now ready to acquire data over 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 Arria 10 Devices**
  Provides more information about the design security for Arria 10 devices.

## Security Mode Verification

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 8: **KEY_VERIFY** JTAG Instruction

<table>
<thead>
<tr>
<th>JTAG Instruction</th>
<th>Instruction Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_VERIFY</td>
<td>00 0001 0011</td>
<td>Connects the key verification scan register between TDI and TDO.</td>
</tr>
</tbody>
</table>

**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 9: Security Mode Verification for 40-nm FPGAs

<table>
<thead>
<tr>
<th>Security Mode</th>
<th>Supported Device</th>
<th>Bit 0</th>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Bit 3</th>
<th>Bit 4</th>
<th>Bit 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>No key</td>
<td>Arria II GX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>• Arria II GZ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Stratix IV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Volatile key</td>
<td>Arria II GX</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>• Arria II GZ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Stratix IV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Volatile key with tamper protection</td>
<td>Arria II GX</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>• Arria II GZ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Stratix IV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Non-volatile key</td>
<td>Arria II GX</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>• Arria II GZ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Stratix IV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Non-volatile key with tamper protection bit</td>
<td>Arria II GX</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>• Arria II GZ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Stratix IV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### Table 10: Security Mode Verification for 28-nm FPGAs

<table>
<thead>
<tr>
<th>Security Mode</th>
<th>Bit 0</th>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Bit 3</th>
<th>Bit 4</th>
<th>Bit 5</th>
<th>Bit 6</th>
<th>Bit 7</th>
<th>Bit 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>No key</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Volatile key</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Volatile key with tamper protection (8)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Non-volatile key</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Non-volatile key with tamper protection bit(8)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### Table 11: Security Mode Verification for 20-nm FPGAs

<table>
<thead>
<tr>
<th>Security Mode</th>
<th>Bit 0</th>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Bit 3</th>
<th>Bit 4</th>
<th>Bit 5 - Bit 20</th>
</tr>
</thead>
</table>

(8) If the tamper protection is enabled, the device will be in JTAG secure mode after power-up. You need to issue the UNLOCK to disable the JTAG secure mode.
<table>
<thead>
<tr>
<th>Key Type</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>No key</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>Volatile key</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>Volatile key with tamper protection(^{(8)})</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>Non-volatile key</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>Non-volatile key with tamper protection bit(^{(8)})</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

The following examples show the .jam files to verify the FPGAs security modes.

**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;
```
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 Arria 10 Devices**
  Provides more information about the design security for Arria 10 devices.

Verification During JTAG Secure Mode

Non-mandatory JTAG instructions that are disabled when the tamper protection bit is enabled in 28-nm and 20-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 will result 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.

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 FlashLoader Support with Encryption Enabled

Altera provides an in-system programming (ISP) solution for serial configuration devices—Serial FlashLoader (SFL). The SFL megafunction is available in Quartus Prime software version 6.0 SP1 or later. You can instantiate the SFL block to your design and have the flexibility to update your 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 SFL megafunction 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 FlashLoader Support with Encryption Enabled for Single FPGA Device Chain

To use the SFL megafunction with the encryption feature enabled in a single FPGA device chain, follow these steps:

1. Start the Quartus Prime software.
2. Instantiate the SFL megafunction 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:
   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 8 of How to Generate the Single-Device .ekp File and Encrypt the Configuration File using Quartus Prime Software on page 8.
   j. Highlight FlashLoader 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 How to Perform Single-Device Volatile or Non-Volatile Key Programming Using Quartus Prime Software on page 13.

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

- Using the Serial FlashLoader with the Quartus II 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.
- Arria 10 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

When you enable the tamper-protection bit, 28-nm and 20-nm FPGAs are in JTAG secure mode upon power up. 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 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 12: Mandatory and Non-Mandatory IEEE Std. 1149.1 and IEEE Std. 1149.6 BST JTAG Instructions

<table>
<thead>
<tr>
<th>Mandatory IEEE Std. 1149.1 and IEEE Std. 1149.6 BST JTAG Instructions</th>
<th>Non-Mandatory IEEE Std. 1149.1 and IEEE Std. 1149.6 BST JTAG Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>• BYPASS</td>
<td>• CONFIG_IO</td>
</tr>
<tr>
<td>• EXTEST</td>
<td>• CLAMP</td>
</tr>
<tr>
<td>• IDCODE</td>
<td>• EXTEST_PULSE</td>
</tr>
<tr>
<td>• LOCK</td>
<td>• EXTEST_TRAIN</td>
</tr>
<tr>
<td>• UNLOCK</td>
<td>• HIGHZ</td>
</tr>
<tr>
<td>• SAMPLE/PRELOAD</td>
<td>• KEY_CLR_VREG</td>
</tr>
<tr>
<td>• SHIFT_EDERROR_REG</td>
<td>• KEY_VERIFY</td>
</tr>
<tr>
<td></td>
<td>• PULSE_NCONFIG</td>
</tr>
<tr>
<td></td>
<td>• USERCODE</td>
</tr>
</tbody>
</table>
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 do 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 with tamper-protection bit disabled do 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—external JTAG interface and internal JTAG interface.

The external JTAG interface refers to access from the physical JTAG pins—`TCK`, `TDI`, `TDO`, and `TMS`—to the JTAG control block. 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 of `TCK`, `TDI`, `TDO`, and `TMS` signals from the internal FPGA core fabric to the JTAG control block. You can only access the JTAG control block using either external or internal JTAG interface 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 Quartus Prime design.

**Figure 3: Internal and External JTAG Interface Connection**
## Table 13: WYSIWYG Atom for 28-nm and 20-nm FPGAs

<table>
<thead>
<tr>
<th>Device Family</th>
<th>JTAG WYSIWYG Atom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arria V</td>
<td><code>arriv_jtag &lt;jtagblock_name&gt;</code></td>
</tr>
<tr>
<td></td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>.clkdruser(),</td>
</tr>
<tr>
<td></td>
<td>.corectl(),</td>
</tr>
<tr>
<td></td>
<td>.runidleuser(),</td>
</tr>
<tr>
<td></td>
<td>.shiftuser(),</td>
</tr>
<tr>
<td></td>
<td>.tck(),</td>
</tr>
<tr>
<td></td>
<td>.tckcore(),</td>
</tr>
<tr>
<td></td>
<td>.tckutap(),</td>
</tr>
<tr>
<td></td>
<td>.tdi(),</td>
</tr>
<tr>
<td></td>
<td>.tdicore(),</td>
</tr>
<tr>
<td></td>
<td>.tdiutap(),</td>
</tr>
<tr>
<td></td>
<td>.tdo(),</td>
</tr>
<tr>
<td></td>
<td>.tdocore(),</td>
</tr>
<tr>
<td></td>
<td>.tdouser(),</td>
</tr>
<tr>
<td></td>
<td>.tdoutap(),</td>
</tr>
<tr>
<td></td>
<td>.tms(),</td>
</tr>
<tr>
<td></td>
<td>.tmscore(),</td>
</tr>
<tr>
<td></td>
<td>.tmsutap(),</td>
</tr>
<tr>
<td></td>
<td>.updateuser(),</td>
</tr>
<tr>
<td></td>
<td>.usrluser()</td>
</tr>
<tr>
<td></td>
<td>);</td>
</tr>
<tr>
<td>Device Family</td>
<td>JTAG WYSIWYG Atom</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Cyclone V</td>
<td><code>cyclonev_jtag &lt;jtagblock_name&gt;</code></td>
</tr>
<tr>
<td></td>
<td>(.clkdruser(),</td>
</tr>
<tr>
<td></td>
<td>.corectl(),</td>
</tr>
<tr>
<td></td>
<td>.runidleuser(),</td>
</tr>
<tr>
<td></td>
<td>.shiftuser(),</td>
</tr>
<tr>
<td></td>
<td>.tck(),</td>
</tr>
<tr>
<td></td>
<td>.tckcore(),</td>
</tr>
<tr>
<td></td>
<td>.tckutap(),</td>
</tr>
<tr>
<td></td>
<td>.tdi(),</td>
</tr>
<tr>
<td></td>
<td>.tdicore(),</td>
</tr>
<tr>
<td></td>
<td>.tdiutap(),</td>
</tr>
<tr>
<td></td>
<td>.tdo(),</td>
</tr>
<tr>
<td></td>
<td>.tdocore(),</td>
</tr>
<tr>
<td></td>
<td>.tdouser(),</td>
</tr>
<tr>
<td></td>
<td>.tdoutap(),</td>
</tr>
<tr>
<td></td>
<td>.tms(),</td>
</tr>
<tr>
<td></td>
<td>.tmscore(),</td>
</tr>
<tr>
<td></td>
<td>.tmsutap(),</td>
</tr>
<tr>
<td></td>
<td>.updateuser(),</td>
</tr>
<tr>
<td></td>
<td>.usrluser()</td>
</tr>
<tr>
<td></td>
<td>);</td>
</tr>
<tr>
<td>Device Family</td>
<td>JTAG WYSIWYG Atom</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Stratix V</td>
<td><code>stratixv_jtag &lt;jtagblock_name&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>.clkdruser()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.corectl()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.runidleuser()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.shiftuser()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.tck()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.tckcore()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.tckutap()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.tdi()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.tdicore()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.tdiutap()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.tdo()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.tdocore()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.tdouser()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.tdoutap()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.tms()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.tmscore()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.tmsutap()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.updateuser()</code>,</td>
</tr>
<tr>
<td></td>
<td><code>.usr1user()</code></td>
</tr>
<tr>
<td></td>
<td>);</td>
</tr>
</tbody>
</table>
Table 14: Functions of the Ports in WYSIWYG Atom

<table>
<thead>
<tr>
<th>Ports</th>
<th>Input/Output</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;jtagblock_name&gt;</td>
<td></td>
<td>Identifier for the <code>arriaii_jtag</code> WYSIWYG atom and represents any identifier name that is legal for the given description language, such as Verilog HDL, VHDL, and AHDL.</td>
</tr>
</tbody>
</table>
### Internal JTAG Interface

<table>
<thead>
<tr>
<th>Ports</th>
<th>Input/Output</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>.corectl()</td>
<td>Input</td>
<td>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 will enable the internal JTAG interface (with external JTAG interface disabled at the same time) and pulling this port to logic low will disable the internal JTAG interface (with external JTAG interface enabled at the same time).</td>
</tr>
<tr>
<td>.tckcore()</td>
<td>Input</td>
<td>Core TCK signal. (9)</td>
</tr>
<tr>
<td>.tdicore()</td>
<td>Input</td>
<td>Core TDI signal. (9)</td>
</tr>
<tr>
<td>.tdocore()</td>
<td>Output</td>
<td>Core TDO signal. (9)</td>
</tr>
<tr>
<td>.tmscore()</td>
<td>Input</td>
<td>Core TMS signal. (9)</td>
</tr>
</tbody>
</table>

(9) 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.
### Related Information

- **EthernetBlaster Communications Cable User Guide**  
  Provides more information about how to change the TCK clock speed for Ethernet Blaster II.
- **EthernetBlaster II Communications Cable User Guide**  
  Provides more information about the specific voltages required using the JTAG download 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.
- **Arria 10 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**
Design Example for JTAG Secure Mode

This design example demonstrates the instantiation of internal JTAG WYSIWYG atom and shows the example of user logic implementation in the Quartus Prime software to execute the LOCK and UNLOCK JTAG instructions. This reference design is targeted on the Arria V device with the tamper-protection bit enabled.

The reference design is developed for the Arria V device using the following components in the Quartus Prime design.

Related Information
AN 556 Design Files

Design Example Quartus Prime Design Components

Table 15: Quartus Prime Design Components for the Arria V Device

<table>
<thead>
<tr>
<th>Component</th>
<th>Function and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTAG_Lock_Unlock.bdf</td>
<td>The top entity of the reference design.</td>
</tr>
<tr>
<td>JTAG_Lock_Unlock_wysiwyg.v</td>
<td>The Verilog code for the Arria V device WYSIWYG atom instantiation. You need to modify this code according to Table 13 for compliance with other 28-nm FPGAs.</td>
</tr>
<tr>
<td>ALTINT_OSC.v</td>
<td>The megafunction instantiation for 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 external clock source.</td>
</tr>
<tr>
<td>User_logic_control_block.v</td>
<td>The example of user logic developed in Verilog code to facilitate the JTAG instruction execution for the Arria V device WYSIWYG atom. You can modify this code to fit your design requirement and restriction, or replace this code with other similar implementation.</td>
</tr>
<tr>
<td>Pulse_nconfig.jam</td>
<td>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 32. This file is optional and can be replaced with other method to verify the JTAG secure mode.</td>
</tr>
</tbody>
</table>

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 16: Input and Output Port of the User Logic

<table>
<thead>
<tr>
<th>Port</th>
<th>Input/Output</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clk_in</code></td>
<td>Input</td>
<td>Clock source for the user logics. The $f_{\text{MAX}}$ of the user logic depends on the timing closure analysis. You need to apply timing constraint and perform timing analysis on the path to determine the $f_{\text{MAX}}$.</td>
</tr>
<tr>
<td><code>start_lock</code></td>
<td>Input</td>
<td>Logic high to trigger the execution of the <strong>LOCK</strong> JTAG instruction to the internal JTAG interface.</td>
</tr>
<tr>
<td><code>start_unlock</code></td>
<td>Input</td>
<td>Logic high to trigger the execution of the <strong>UNLOCK</strong> JTAG instruction to the internal JTAG interface.</td>
</tr>
<tr>
<td><code>jtag_core_en_out</code></td>
<td>Output</td>
<td>Output to the JTAG WYSIWYG atom. This port is connected to the <code>corectl</code> port of the JTAG WYSIWYG atom to enable the internal JTAG interface.</td>
</tr>
<tr>
<td>Port</td>
<td>Input/Output</td>
<td>Function</td>
</tr>
<tr>
<td>---------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tck_out</td>
<td>Output</td>
<td>Output to the JTAG WYSIWYG atom. This port is connected to the tck_core port of the JTAG WYSIWYG atom.</td>
</tr>
<tr>
<td>tdi_out</td>
<td>Output</td>
<td>Output to the JTAG WYSIWYG atom. This port is connected to the tdi_core port of the JTAG WYSIWYG atom.</td>
</tr>
<tr>
<td>tms_out</td>
<td>Output</td>
<td>Output to the JTAG WYSIWYG atom. This port is connected to the tms_core port of the JTAG WYSIWYG atom.</td>
</tr>
<tr>
<td>indicator</td>
<td>Output</td>
<td>Logic high of this output pin indicates the completion of the LOCK or UNLOCK JTAG instruction execution.</td>
</tr>
</tbody>
</table>

**Related Information**

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

**Verifying JTAG Secure Mode**

Altera 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\(^{(10)}\), 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 on page 7. 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 reconfig-

\(^{(10)}\) You should only apply these steps on an FPGA with the tamper-protection bit enabled.
ration 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 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 STAPI Solution for Device Programming

Provides more information about quartus_jli.

Document Revision History

Table 17: Document Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>November 2015</td>
<td>2015.11.02</td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Changed instances of Quartus II to Quartus Prime.</td>
</tr>
<tr>
<td>June 2015</td>
<td>2015.06.15</td>
<td>Added link to JTAG Secure Mode Design Example.</td>
</tr>
<tr>
<td>May 2015</td>
<td>2015.05.04</td>
<td>Corrected the total number of character in .key file example.</td>
</tr>
<tr>
<td>January 2015</td>
<td>2015.01.23</td>
<td>• Added 20-nm FPGAs (Arria 10) support.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added JAM file example for 20-nm.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Security Mode Verification for 20-nm table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added JTAG WYSIWYG atom for Arria 10.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added AES modes in Altera FPGAs.</td>
</tr>
<tr>
<td>December 2014</td>
<td>2014.12.15</td>
<td>Added USB-Blaster II support for non-volatile security key programming.</td>
</tr>
<tr>
<td>Date</td>
<td>Version</td>
<td>Changes</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>---------</td>
</tr>
</tbody>
</table>
| 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. |
<p>| May 2014      | 2014.05.19 | Updated the Non-Volatile and Volatile Key Storage section to include information on using valid MSEL pin settings. |</p>
<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 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.  
• 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. |
<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 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 Figure 26 on page 29.  
| March 2009 | 1.0     | Initial release.  