Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS and with all faults, and all risk associated with such information is entirely with Buyer. Buyer shall not rely on any data and performance specifications or parameters provided herein. Products sold by Lattice have been subject to limited testing and it is the Buyer’s responsibility to independently determine the suitability of any products and to test and verify the same. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice’s product could create a situation where personal injury, death, severe property or environmental damage may occur. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.
Contents

Acronyms in This Document ........................................................................................................................................... 5
1. Introduction .................................................................................................................................................................... 6
2. Overview ...................................................................................................................................................................... 7
3. Overview of the Lattice Encryption Feature ............................................................................................................... 9
4. Overview of the Encryption Key Programming Algorithm .......................................................................................... 10
   4.1. Encryption Key Programming Flow .................................................................................................................. 10
   4.2. CrossLink-NX Bitstream Encryption Format ..................................................................................................... 12
   4.3. Method for Creating an Encrypted Bitstream File and Key File ........................................................................ 12
   4.4. Setting Security and Encryption for FPGA Devices ............................................................................................ 12
5. Creating Encryption Key and Encrypted Bitstream with Lattice Radiant Security Setting Tool and Process Flow ....... 13
6. Setting Security and Encryption Using the Deployment Tool ..................................................................................... 16
7. Programming Encryption Key .......................................................................................................................................... 21
8. Programming Encrypted Bitstream ................................................................................................................................ 24
Glossary .............................................................................................................................................................................. 27
References ........................................................................................................................................................................... 28
Technical Support Assistance .............................................................................................................................................. 29
Revision History ................................................................................................................................................................. 30
Figures
Figure 3.1. Encryption Key Fuse Block Diagram .................................................................9
Figure 4.1. Encryption Key Fuse JTAG Programming Flow ..................................................10
Figure 5.1. Bitstream Setup .................................................................................................13
Figure 5.2. Key File Password Setup ...................................................................................13
Figure 5.3. Change Key File Password .................................................................................14
Figure 5.4. Setup AES Key and Optional Password ............................................................14
Figure 6.1. Deployment Tool Project Setup .........................................................................16
Figure 6.2. Deployment Tool Input File Setup ....................................................................17
Figure 6.3. Deployment Tool Output Setup ..........................................................................17
Figure 6.4. Password Input for Encrypt Key File .................................................................18
Figure 6.5. Deployment Tool Output Setup with Encryption Key .........................................19
Figure 6.6. Lattice Radiant Deployment Tool Summary .......................................................20
Figure 7.1. Invoke Programmer from Lattice Radiant Software ..........................................21
Figure 7.2. Lattice Radiant Programmer Device Property Setup ........................................22
Figure 7.3. Input the Password for the Key File ................................................................23
Figure 8.1. Invoke Programmer from Lattice Radiant Software ..........................................24
Figure 8.2. Lattice Radiant Programmer Start Up .................................................................25
Figure 8.3. Lattice Radiant Programmer Operation Setup ....................................................25
Figure 8.4. Lattice Radiant Programmer Device Properties Setup ........................................26
Figure 8.5. Lattice Radiant Programmer Activity Start .......................................................26

Tables
Table 4.1. Encryption Key Fuse JTAG Programming Flow ..................................................11
Table 4.2. Encrypted Bitstream Format ..............................................................................12
Acronyms in This Document

A list of acronyms used in this document.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>ECC</td>
<td>Error Correction Code</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>OTP</td>
<td>One-Time Programmable</td>
</tr>
<tr>
<td>TMR</td>
<td>Triple Modular Redundancy</td>
</tr>
</tbody>
</table>
1. Introduction

Lattice Crosslink-NX devices offer 256-bit Advanced Encryption Standard (AES) to protect the bitstream. The 256-bit key is user specified, and no special voltages are required to maintain the key within the FPGA. AES bitstream protection is available in all versions of the CrossLink-NX™ device.

AES encryption and decryption is enhanced bitstream protection. A lessor protection common in the industry is SRAM read-back protection, which blocks the read-back of the configuration image from external ports. However, if the configuration bitstream is sourced from an external boot device, it is quite easy to read and intercept the configuration data, allowing access to the FPGA design. Without encryption, bitstreams are not protected or secure. Anyone can copy the design simply by reading the bitstream out of the boot PROM.

AES Encryption offers a mechanism to prevent reverse engineering of intellectual property. By encrypting the bitstream, it cannot be used in any scenario without the decryption key. Encrypted bitstreams are secure against snooping, copying, or direct reading of the image. Encrypted bitstreams can also be utilized to prevent its use in unsanctioned devices, limiting supply of the device for inventory control purposes.

In terms of design security, an encrypted bitstream is known as protected and an unencrypted bitstream is unprotected. The AES security system is the method that provides a high level of protection to the bitstream of volatile FPGA devices. This is demonstrated by the fact that some FPGA vendors started with different encryption schemes and all eventually standardized on AES. A detailed description of AES can be found in the U.S. government publication of the standard.

AES uses the symmetric Decipher Key format (such as encryption key = decryption key). Therefore, the terms decipher key, encryption key, and decryption key are synonymous. AES defines the decipher key size to be 64, 128, or 256 bits in length.

CrossLink-NX FPGA supports:
- The optimal 256-bit AES key
- The reliable One-Time Programmable (OTP) non-volatile fuses for the AES encryption key
- Option of unencrypted bitstream can be or cannot be programmed into the device when security key is set

This document describes the security key programming operations at various stages of the implementation:
- Board prototyping using Lattice Radiant® Programmer software
- Manufacturing using Lattice Radiant Programmer
- Large volume production programming using desktop programmers
- Advanced security key programming using Lattice Radiant Programmer
  - Decryption key serialization
  - Bitstream encryption with serialized decryption key

The encryption scheme detail implemented in CrossLink-NX FPGA families is provided in this document for those interested in the implementation details.
2. Overview

Lattice Radiant Programmer takes care of all the technical details of programming the device, saving users from having to understand all the underlying programming complexities.

The technical information about the key code programming provided here is for the benefit of encryption users as a reference. Due to the complexity of programming the key code into the device, only the high-level flow is presented. It will not be enough detail for users to craft their own programming code (driver).

In CrossLink-NX devices, the OTP fuses for the key code can be programmed in-system using Lattice Radiant Programmer in one of the following ways:

- On-board programming using a Lattice parallel port cable with the 8-pin AMP connector or 10-pin JEDEC connector connected between a PC or Linux system and a board test system.
- On-board programming using a Lattice USB port cable connected between a PC or Linux system and a board test system.
- Embedded on-board programming using FTDI single chip USB converter devices connected between a PC or Linux system and an outfitted PCB host.

Lattice provides users the following methods to program the key code:

- Off-board programming using key files for third-party programmer vendors (BPM Microsystems, System General).
- On-board programming using STAPL files for third-party BSCAN tools.

The weakest link of the encryption flow is the transmission path of the key code from the source to the programming area. There are two methods for transmitting the key code:

- Verbal
- Written (such as encrypted key files)

The best deterrence against hackers is to transmit the key code in the form of encrypted key files to the trusted area. The password for opening the encrypted key files can be transmitted when the programming operation begins, and controlled with either time or quantity expiration.

Lattice Radiant Programmer supports the following method of key code transmission:

- Importing the key file after entering the password for the key file.

To maintain the confidentiality and security of the key code, an industry standard requires all software and programming tools for key code programming to observe the following rule.

**Note:** It is strongly recommended you do not make a hard copy of the key code in the form of a data record (file) on a non-volatile media, such as hard disks, or as a data image into non-volatile memory such as Flash devices.

Software and programming tools, therefore, must only copy the key code into the volatile RAM of the PC during run time in order to adhere to this rule.

The Lattice Radiant software follows the rule except when the key code is entered directly into the Lattice Radiant Security Setting Tool GUI. The key code is written into a key (.bek) file. The user is informed and can make the appropriate arrangement to protect the files.

The true security of encryption depends entirely on the security (protection) of the key code. Once the key code is compromised, the corresponding encrypted bitstream is also compromised. This is because the decryption algorithm is available to decrypt the bitstream with a good key.

**Note:** A good key is the key code that is used to encrypt the bitstream. The bitstream will not work if a different key code is used to decrypt the encrypted bitstream. Therefore, a different key code is a bad key. A working bitstream means the device receiving the bitstream is configured successfully, resulting in both DONE and INITN pins going high. By simply viewing a decrypted bitstream, one cannot tell if it will successfully configure a device.

A specialist skilled in the art of encryption can extract the key code if both the unencrypted bitstream and the encrypted version of the same bitstream are available. Thus, for security reasons, it is standard practice in the advanced security industry to demand traceability of the unencrypted bitstream. Once the unencrypted bitstream is encrypted, the unencrypted bitstream should be destroyed (deleted) to protect it.
The discussion above concentrates on standard practices of the security industry for protecting key codes from leaking out through paperwork control. For hackers, instead of hunting the key code through paperwork leaks, their first line of attack is to read from FPGA devices.

- Open the package and inspect OTP fuses on the die visually.
- Open the package and micro-probe the die to read the OTP fuses electronically.

In addition to the standard silicon layout defensive practices (such as hiding fuses under several layers of metal and scrambling their physical locations), Lattice has implemented the hashing code by the mesh circuit in CrossLink-NX devices. This is a reliable defense against the two hacking tactics mentioned above. Even in the event that the hacker succeeds in obtaining a key code, the key code is invalid (useless).

Lattice devices are uniquely designed to accept a regular bitstream whether or not the encryption key is programmed into the device. This allows for system debugging even if the key code is lost or unknown. This is especially handy when each device has a unique key code.

Some designers believe that if a device accepts both an encrypted and unencrypted bitstream, hackers can easily create an unencrypted bitstream for the device to plant the probing agent (hacker probe) to compromise the security of the system, and thus the device would have a security hole. This conclusion results from the assumption that the unencrypted bitstream for the hacker probe can be superimposed on the encrypted bitstream. This cannot happen because prior to configuring the device with an unencrypted bitstream, the Lattice devices will perform a mandatory clear-all operation, which erases the SRAM fuses prior to configuration. Thus, the hacker probe planted into a cleared device is useless and the key code and the encrypted bitstream remain protected. On top of that, CrossLink-NX devices also provide an Encryption Only option, which allows you to set the device to accept encrypted bitstream only.
3. Overview of the Lattice Encryption Feature

Figure 3.1 illustrates the logical functional blocks supporting the encryption feature. Due to the nature of the OTP fuses, some fuses might not respond to the program command. Thus, the ECC (Error Correction Code) is deployed for the EFUSE memory to serve the following purposes:

- Enhance the encryption key programming yield
- Enhance encryption key reliability

The function of the key lock fuse is to protect the programmed encryption key from being read. Therefore, programming the key lock fuse does not affect the encryption feature. If the device key is programmed, before shipping the boards out to end-customers, the key lock fuse must be programmed to protect the encryption key from falling into the unauthorized hands.

The purpose of the hardware encryption/decryption is to destroy the visual correlation between the physical location of a fuse and its logical function (such as the decipher key bit number). In other words, the key code is still protected even if the die is open and the fuse pattern is read by visual inspection. This circuit causes the key code encrypted for programming to be different from the one shifted out for verification.

The task of the decryption engine is to decrypt the incoming encrypted bitstream, turning it into an unencrypted bitstream while writing into the SRAM fuses. After the unencrypted bitstream is written into the SRAM fuses, setting the security fuse is the only method that can be used to protect the bitstream. The security fuse is set automatically when encrypting the bitstream.
4. **Overview of the Encryption Key Programming Algorithm**

The E-fuses in Lattice devices do not need a super-voltage or high current to be programmed. This allows the encryption key to be programmed in-system. However, the nature of E-fuses does not allow for any errors. Once an E-fuse that is not supposed to be programmed is programmed, the device will become a reject. The standard precautionary measures are deployed to minimize the possibility of rejects:

- Use a qualification technique to verify first, then program
- Use a double shifting-in technique to screen out double clocking
- Use a double verify technique to protect against data corruption

4.1. **Encryption Key Programming Flow**

**Figure 4.1** provides a high-level view of the encryption key programming flow implemented in Lattice Radiant Programmer. The programming algorithm for the OTP fuses is complex, and it is impractical for users to craft their own programming code. It is also beyond the scope of this document to describe the programming flow in detail.
Table 4.1. Encryption Key Fuse JTAG Programming Flow

<table>
<thead>
<tr>
<th>Note</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Check device ID code</td>
</tr>
<tr>
<td>2</td>
<td>Set the device into programmable mode. The device may require a special password to arm the E-fuse programming engine to prevent accidental or unauthorized access to the E-fuse block if the password protection feature is enabled.</td>
</tr>
<tr>
<td>3</td>
<td>This is the step to check the device status to make sure the efuse user sector of the device is programmable. If the device efuse user sector has been programmed and security protection is set, the programming flow declares failure and terminated.</td>
</tr>
<tr>
<td>4</td>
<td>Programming time or burn time. This is the most critical moment of the programming process. The power supply must not be interrupted. If a power disruption happens (such as a ground surge), the E-fuse state will not be undefined.</td>
</tr>
<tr>
<td>5</td>
<td>Check BUSY flag until the programming activity is finished, or wait for 2 seconds.</td>
</tr>
<tr>
<td>6</td>
<td>Verify the key programming. If the read back data does not match the input data, the device must be rejected. For large volume key code programming, Lattice recommends using third-party desktop programmers (BPM Microsystems or System General).</td>
</tr>
<tr>
<td>7</td>
<td>Optionally program the Decrypt Only option into the EFUSE user feature sector, if it is checked in the Lattice Radiant Programmer user interface. Once programmed, the device will only accept the encrypted bitstream. Unencrypted bitstream is rejected by device.</td>
</tr>
<tr>
<td>8</td>
<td>Verify the programming result for the Decrypt Only option. If the read back data does not match the input data, the device must be rejected.</td>
</tr>
<tr>
<td>9</td>
<td>Optionally program the Key Lock option into the EFUSE user security row, if it is checked in the Lattice Radiant Programmer user interface. Once programmed, the device will no longer allow the cypher key read back or incremental programming depends on the user selection in the Lattice Radiant Programmer user interface.</td>
</tr>
<tr>
<td>10</td>
<td>Verify the programming result for the Key Lock option. If the read back data does not match the input data, the device must be rejected.</td>
</tr>
<tr>
<td>11</td>
<td>Exit the device from programmable mode. If the DONE bit of the device is set, the device goes to user functional mode, otherwise it goes to Un-programmed mode.</td>
</tr>
</tbody>
</table>

Lattice works closely with third-party programmer vendors to develop off-board programmer support. Lattice provides detailed programming algorithm specifications to the Lattice-approved third-party programmer vendors. Lattice engineers perform a full qualification on the programming code developed by these vendors to ensure the fuses are programmed to meet Lattice’s quality and reliability requirements.

Lattice also can pre-program a user’s encryption key into a device before shipment. Contact Lattice Sales for details.
4.2. CrossLink-NX Bitstream Encryption Format

Table 4.2. Encrypted Bitstream Format

<table>
<thead>
<tr>
<th>Frame</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSCC signature</td>
<td>32 bits of the LSCC Signature (LSCC = 0x4C534343)</td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Can be replaced by &gt;= 128 bits of dummy.</td>
</tr>
<tr>
<td>Header</td>
<td>First 16 bits of the 32-bit Bitstream Preamble.</td>
</tr>
<tr>
<td></td>
<td>Second 16 bits of the 32-bit Encryption Preamble</td>
</tr>
<tr>
<td></td>
<td>512-bit Dummy (Required by HW to expand the key)</td>
</tr>
<tr>
<td>Encrypted Configuration Data</td>
<td>The decryption packet size = 128 bits per packet.</td>
</tr>
<tr>
<td>Packet Bound Filler Bits</td>
<td>Insert filler to meet the 128-bit bound requirement (all 0’s).</td>
</tr>
</tbody>
</table>

Notes:
1. Dual Boot and multiple boot with encryption are available with CrossLink-NX devices.
2. The direction of shifting the bitstream is from left to right (such as bit 0, 1, 2...n).
3. When the devices read configuration data out of the SPI Flash devices, the first 128 bits of data are ignored since most SPI Flash devices provide random data on the first eight clocks in a fast read. The header string, which usually stores the user entry, is a don’t care for configuration purpose. Deployment Tool uses the header as the filler to meet the 128 bits dummy requirement.

4.3. Method for Creating an Encrypted Bitstream File and Key File

The Encryption feature must be enabled to create a key file and encrypt a Lattice bitstream file. The encrypted key file could be generated from the Lattice Radiant design software. There are two ways to create an encrypted bitstream file:

- In the Lattice Radiant design software, set security settings using the Security Setting tool and then generate an encrypted bitstream from within the Process pane.
- In the Deployment tool, specify security settings and convert an unencrypted bitstream file to an encrypted bitstream file.

4.4. Setting Security and Encryption for FPGA Devices

There is an important difference between using the Deployment Tool and using the Security Setting tool and the Lattice Radiant Process flow. When the Security Setting tool is not used, an unencrypted bitstream is created in the Lattice Radiant Process flow. When using the Security Setting tool to specify encryption settings and the Lattice Radiant Process flow to create a bitstream, Lattice Radiant only creates an encrypted bitstream. When using the Deployment tool, an unencrypted bitstream file is used to create an encrypted bitstream file. Therefore, using the Deployment tool is considered less secure unless the tools and the unencrypted bitstream file are in a trusted secure area. The unencrypted bitstream could be protected by using a security control such as PKZIP along with a password to protect the file.
5. Creating Encryption Key and Encrypted Bitstream with Lattice Radiant Security Setting Tool and Process Flow

To create the Encryption Key and Encrypted Bitstream using Lattice Radiant Security Setting Tool and Process Flow:

1. In the Lattice Radiant File List Window pane, double-click **Strategy**.
2. The **Strategies** option window opens. Select **Bitstream** and set the required Bitstream options.
3. Click **OK**.

![Bitstream Setup](image)

**Figure 5.1. Bitstream Setup**

4. In Lattice Radiant, select **Tools > Bitstream Security Settings**. The password dialog box appears with LATTICESEMI as the default password.

![Key File Password Setup](image)

**Figure 5.2. Key File Password Setup**
5. Click **Change Password** to create a unique password for the encryption key file.
6. In the **Change Password** dialog box, enter the password, which must be between eight and 16 characters.
7. Verify the password and click **OK**.

![Figure 5.3. Change Key File Password](image)

8. The bitstream **Security Settings** window appears. Select the **AES Encryption** check box to enable entering the Encryption key. Select the Key Format:
   a. **ASCII** – Alphanumeric values, using up to 32 characters
   b. **Hex** – Values of 0 through F, using up to 64 characters
   c. **Binary** – Values of 0 and 1, using up to 256 characters

   For the scenario, an extra password protection is needed to prevent unauthorized access to the AES key stored inside the non-volatile memory, select the **Password Protection** check box to enable entering the Password. Select the password format:
   a. **ASCII** – Alphanumeric values, using up to 16 characters
   b. **Hex** – Values of 0 through F, using up to 32 characters
   c. **Binary** – Values of 0 and 1, using up to 128 characters

9. Click **OK** to create the bitstream encryption key file (.bek).

   The default location of the key file is at: `<Project Name>/security_setting/<Project Name>.bek`.

![Figure 5.4. Setup AES Key and Optional Password](image)
10. To create the encrypted bitstream, double-click on **Bitstream File** under **Export Files** in the Lattice Radiant **Process** pane. An encrypted bitstream file (.bit) is created. You can now use this file to program the device using Lattice Radiant Programmer.

The encrypted bit stream generation is dependent on the selected configuration mode in the Lattice Radiant Device Constraint Editor. This step is necessary to pad the correct number of program mode dependent filler bits when encrypting the bitstream. The filler bits are described later in this document.
6. Setting Security and Encryption Using the Deployment Tool

The Deployment Tool is a standalone tool that allows you to generate files for deployment for single devices or a chain of devices. It is also used to convert data files to other formats. The Deployment Tool is installed along with the Lattice Radiant Programmer, which could be as part of the Lattice Radiant Software installation. The tool can also be installed separately. The Deployment Tool is launched from the Lattice Radiant Programmer.

The Lattice Radiant Programmer could be launched from:
- Microsoft Windows: Start > Programs > Lattice Radiant Software > Accessories > Lattice Radiant Programmer
- Lattice Radiant Software: Tools > Programmer

To set security and encryption using the Deployment Tool:
1. Open the Deployment Tool from Lattice Radiant Programmer by selecting Tools > Deployment Tool.

![Figure 6.1. Deployment Tool Project Setup](image)

3. In Function Type, select File Conversion.
4. In Output File Type, select Bitstream.
5. Click OK.
6. The File Conversion (Step 1) window is displayed. Click on the blank File Name column.
7. Click on the browse (...) button in the File Name column and select the unencrypted .bit file created by double-clicking on the Bitstream file in the Lattice Radiant Process window.
8. Select **Next** to go to the Step 2 window. Select the **Output Format**.
9. To burn the security fuses to disable the ability to read back the bitstream from the SRAM in the FPGA, set the **Program Security Bit to On**.

10. To store device data such as firmware version number, manufacturer ID, programming date, programmer make, pattern code, etc., set a USERCODE. Select the **USERCODE Format** and then enter a **USERCODE**:
   a. **ASCII** – Alphanumeric values, using up to 16 characters
   b. **Hex** – Values of 0 through F, using up to 32 characters
   c. **Binary** – Values of 0 and 1, using up to 128 characters

11. To create an encrypted bitstream from an unencrypted bitstream, select **Encryption**. Since the CrossLink-NX devices do not have different padding bits for different modes, the **Configuration Mode** option is no longer valid nor necessary.

12. Other file conversion options to choose from are:
   a. **Verify ID Code** – For bitstream debugging, inserts the verify device 32-bit JTAG IDCODE frame into the bitstream. The setting Default means do not override the bitstream. ON (insert) or OFF (omit) overrides the current setting in the bitstream. It is recommended to leave this as default.
   b. **Frequency** – Used to adjust the master clock configuration frequency in the bitstream for the two master modes, SPI and SPI mode. The setting has no effect on the Slave modes. The setting of Default means keep the bitstream setting. Selections other than Default overrides the current setting in the bitstream.
   c. **Compression** – Used to compress the bitstream. Default means do not change the bitstream. ON (compress) or OFF (no compress) overrides the bitstream. It is recommended not to use compression with encryption. Compression not available on LatticeECP3 devices.
   d. **CRC Calculation** – Disables the frame-by-frame CRC for bitstream debugging. It is recommended to keep the default of the bitstream for maximum configuration reliability.

13. To enter the encryption key, click **Edit Key**. A Windows Explorer window opens and prompts you to load the Encryption file (.bek). Select the Encryption Key File (<Project Name>.bek) generated from the Lattice Radiant Bitstream Security Setting as described in Section 5. The dialog box will appear as asking for Encryption File Password:

![Figure 6.4. Password Input for Encrypt Key File](image)

14. Enter the Encryption File Password into the prompt, to view what is being typed, un-check the Hide Password check box, then select **OK**. The Encryption Key field is auto filled with the key string from the Encryption Key file.

15. Click **Next**.
16. Select or enter an Output file name for the encrypted bitstream and click **Next** to proceed to Step 4.

17. A Deployment Tool Summary appears. Click **Generate** to create the encrypted bitstream. An example of the summary and messages is shown below:
You are now ready to download the encrypted bitstream using Lattice Radiant Programmer.
7. Programming Encryption Key

To program the key code onto the OTP fuses in the FPGA, use Lattice Radiant Programmer. Since Programmer is used primarily in the board design prototype development phase, choosing to program the key lock fuse while programming the encryption key code is left to the users. The purpose of the key lock fuse is to disable the reading of the fuse state of all the OTP fuses, which provides the key code as the first line of defense. Therefore, the only method to verify that the key code is correct is to configure the device with an encrypted bitstream.

To program the encryption key using Lattice Radiant Programmer:
1. Be sure the board with the Lattice FPGA is properly connected to your computer and turned on.
2. Open Lattice Radiant Programmer:
   - In the Lattice Radiant window, select Tools > Programmer or click the Programmer icon from the Lattice Radiant toolbar.

   ![Figure 7.1. Invoke Programmer from Lattice Radiant Software](image)

   - In Windows, go to the Windows Start menu and choose Programs > Lattice Radiant Software > Accessories > Lattice Radiant Programmer.
   - In Linux, from the `<install_path>/bin/lin` directory, enter the following on a command line:
     `<Install_path>/programmer`
3. In the Programmer window, double-click in the Operation column for the FPGA device to program. The Device Properties window appears.
4. Select **Non Volatile Configuration Memory** as the **Target Memory**.
5. Select **Port Interface** using JTAG, Slave SPI or I2C.
6. Select **Advanced Security Key Programming** as **Access Mode**.
7. Select **Program Encryption Key** as the **Operation**.

Other options include:

a. **Program Password Key** – Program 128-bit password protection key into the device. The key is from the .key file which is generated from the Lattice Radiant **Tools > Bitstream Security Settings**.

b. **Read Password Key** – Read the contents of password protection key from the device.

c. **Program Encryption Key** – Program the 256-bit encryption key into the device. The key is from the .bek file which is generated from the Lattice Radiant **Tools > Bitstream Security Settings**.

d. **Read Encryption Key** – Read the contents of encryption key from the device.

e. **Program TraceID** – Program 8-bit user’s Unique ID header into the device.

f. **Read TraceID** – Read the contents of Unique ID.

g. **Program Locks Policies** – Program sector’s OTP locking bits into the device. The locks policies included Lock the Read, Lock the Write, Lock Erase and Lock Wishbone Access (Hard Lock).

h. **Read Locks Policies** – Reads the sector’s OTP locking bits from the device.

i. **Update Locks Policies** - Reads the sector’s OTP locking bits from the device, modifies and programs sector’s OTP locking bits back into the device.

j. **Lock Ports Interface** – Program the port interface OTP locking bit into the device. The locks policies included Lock Port Access and Lock Wishbone Access (Hard Lock). The Port Interfaces included JTAG, Slave SPI and I2C.
8. Check the **Password Protection Options** if the password protection is enabled. Click **Load Key**. A Window Explorer window opens and prompts you to load Encryption file (.key).

9. Select the Password Key File generated from the Lattice Radiant Bitstream Security Setting.

10. Enter the Encryption File Password into the prompt.

11. Click **OK**.

![Figure 7.3. Input the Password for the Key File](image)

The **Password Key** field is auto filled from the Key File (.key).

**Note:** Check this option only after programming the password protection key into the device.

12. In the **Encryption Key Options** field, click **Load Key**. A Window Explorer window opens and prompts you to load Encryption file (.bek).

13. Select the Encryption Key File generated from the Lattice Radiant Bitstream Security Setting.

14. Enter the Encryption File password into the prompt.

15. Click **OK**.

The **Encryption Key** field is auto filled from the Encryption Key File (.bek).

16. Check the **Encrypted Bitstream Only** option to allow the device to reject un-encrypted bitstream. Click **Yes** to confirm the selection.

17. Check the **Encryption Row Lock Options**, and check the lock selections
   - Read Lock
   - Write Lock
   - Lock Wishbone Access (Hard Lock)

18. Click **OK** to complete the setup

19. Select the **Program** button in the **Programmer** toolbar to proceed with the programming operation.

![Figure 7.4. Non-Volatile Memory Programming](image)
8. **Programming Encrypted Bitstream**

The encrypted bitstream programming procedure is identical to the programming procedure for plain (unencrypted) bitstreams. An example of SPI Flash Memory programming procedure is demonstrated below.

To program the encryption key:

1. Be sure the board with the Lattice FPGA is properly connect to your computer and turned on.
2. Open Lattice Radiant Programmer:
   - In the **Lattice Radiant** window, select **Tools > Programmer** or click the **Programmer** icon from the **Lattice Radiant** toolbar.
   - In Windows, go to the Windows Start menu and choose Programs > Lattice Radiant Software > Accessories > Lattice Radiant Programmer.
   - In Linux, from the `<install_path>/bin/lin directory`, enter the following on a command line:

   ```
   <Install_path>/.programmer
   ```

3. In the **Lattice Radiant Programmer** window, select **Create a new project from a scan**.
4. Click **OK**.

---

**Figure 8.1.** Invoke Programmer from Lattice Radiant Software
5. In the **Lattice Radiant Programmer** window, select **Edit > Device Properties** to open the **Device Properties Editor**.

6. In the **Device Properties Editor** window, select **External SPI Flash Memory (SPI FLASH)** as **Target Memory**.
7. Load the encrypted programming file. Make sure **Device Operation** and **SPI Flash Operation** selections are correct.
8. Click **OK**.
9. Click **Program Device**.

10. Successful programming is confirmed with the status **Operation: Successful**.

---

Figure 8.4. Lattice Radiant Programmer Device Properties Setup

Figure 8.5. Lattice Radiant Programmer Activity Start
## Glossary

A glossary of terms used in this document.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>The algorithmic validation process to determine pass/fail results.</td>
</tr>
<tr>
<td>Bitstream</td>
<td>A binary file that contains commands and data that can configure FPGA devices.</td>
</tr>
<tr>
<td>Configuration</td>
<td>Program (write to the fuses of) a volatile device.</td>
</tr>
<tr>
<td>Decipher Key</td>
<td>The key code used for encryption or decryption.</td>
</tr>
<tr>
<td>Decrypt</td>
<td>Apply the reverse encryption process on an encrypted file.</td>
</tr>
<tr>
<td>Encrypt</td>
<td>The encryption action has taken place.</td>
</tr>
<tr>
<td>Encryption</td>
<td>Uses a password (key) and an algorithm to scramble a file.</td>
</tr>
<tr>
<td>Key Code</td>
<td>The binary key pattern for encryption or decryption.</td>
</tr>
<tr>
<td>Key Code Size</td>
<td>The fixed length of the key code in bits. For CrossLink-NX devices, it is 256 bits.</td>
</tr>
<tr>
<td>Key Lock Fuse</td>
<td>When programmed, the key lock fuse prevents the encryption key code from being read out.</td>
</tr>
<tr>
<td>Decrypt Only Fuse</td>
<td>When programmed, the device will only accept encrypted bitstream.</td>
</tr>
<tr>
<td>Non-Volatile Fuse</td>
<td>Fuses that keep the fuse state when power is turned off.</td>
</tr>
<tr>
<td>OTP Fuse</td>
<td>One-time programmable, non-volatile fuse that can only be programmed once, and cannot be erased.</td>
</tr>
<tr>
<td>Over Program</td>
<td>An error caused by blowing a fuse that was supposed to remain un-programmed.</td>
</tr>
<tr>
<td>Private Key</td>
<td>The key code that is confined to the Trusted Area.</td>
</tr>
<tr>
<td>Program Fuse</td>
<td>An OTP fuse that is blown to produce a logical state 1.</td>
</tr>
<tr>
<td>Public Key</td>
<td>The key code that is not confined to the Trusted Area.</td>
</tr>
<tr>
<td>Un-program Fuse</td>
<td>An OTP fuse that is not blown (left intact) to produce a logical state 0.</td>
</tr>
<tr>
<td>Under Program</td>
<td>An error caused by leaving intact (un-blown) a fuse that was supposed to be programmed.</td>
</tr>
<tr>
<td>Unencrypt</td>
<td>No encryption action has taken place.</td>
</tr>
<tr>
<td>Trusted Area</td>
<td>The real or virtual space that houses all confidential and high-security material.</td>
</tr>
</tbody>
</table>
References

- CrossLink-NX sysCONFIG Usage Guide (FPGA-TN-02099)
Technical Support Assistance
Submit a technical support case through www.latticesemi.com/techsupport.
## Revision History

**Revision 1.0, March 2020**

<table>
<thead>
<tr>
<th>Section</th>
<th>Change Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>