

## Introduction

All Lattice FPGAs provide configuration data read security, meaning that a fuse can be set so that when the device is read all zeros will be output instead of the actual configuration data. This kind of protection is common in the industry and provides very good security if the configuration data storage is on-chip, such as with the LatticeXP™ and MachXO™ device families. However, if the configuration bitstream comes from an external boot device it is quite easy to read the configuration data, allowing access to the FPGA design.

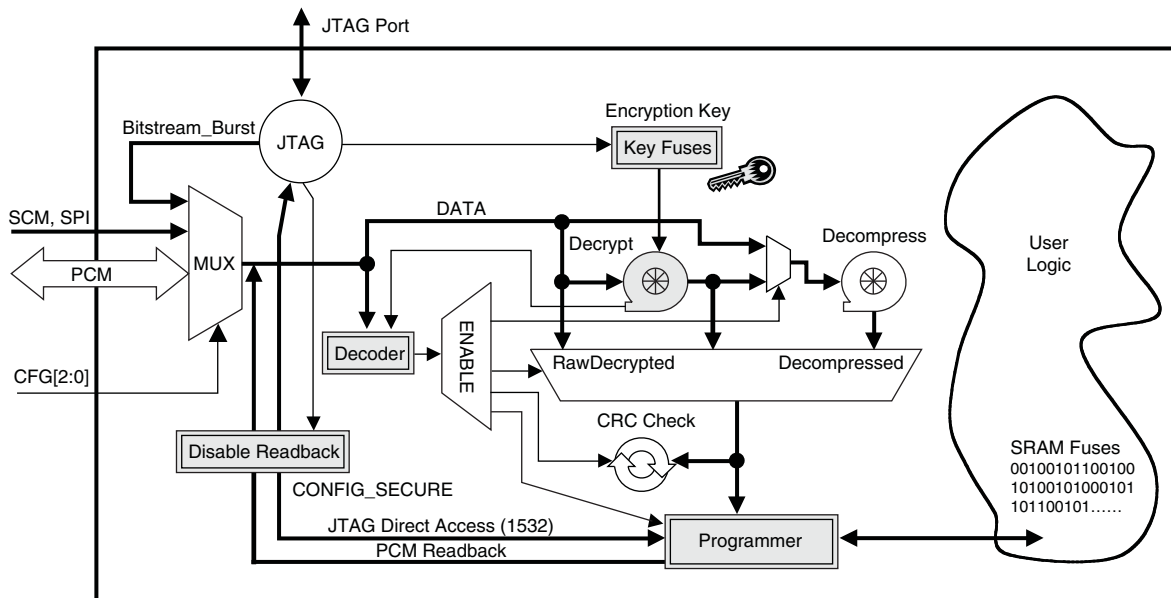
For this reason the “S” versions of LatticeECP2™ and LatticeECP2M™ offer the 128-bit Advanced Encryption Standard (AES) to protect the bitstream. The user selects and has total control over the 128-bit key and no special voltages are required to maintain the key within the FPGA.

This document explains the capabilities of this new security feature and how to take advantage of it.

## General Configuration Process

Figure 16-1 is a block diagram describing the LatticeECP2/M “S” version bitstream encryption data paths. Refer to this figure as you read the following sections.

**Figure 16-1. LatticeECP2/M “S” Version Bitstream Encryption Block Diagram**



Lattice FPGAs are configured by using the sysCONFIG™ interface or the JTAG interface (see Table 16-1).

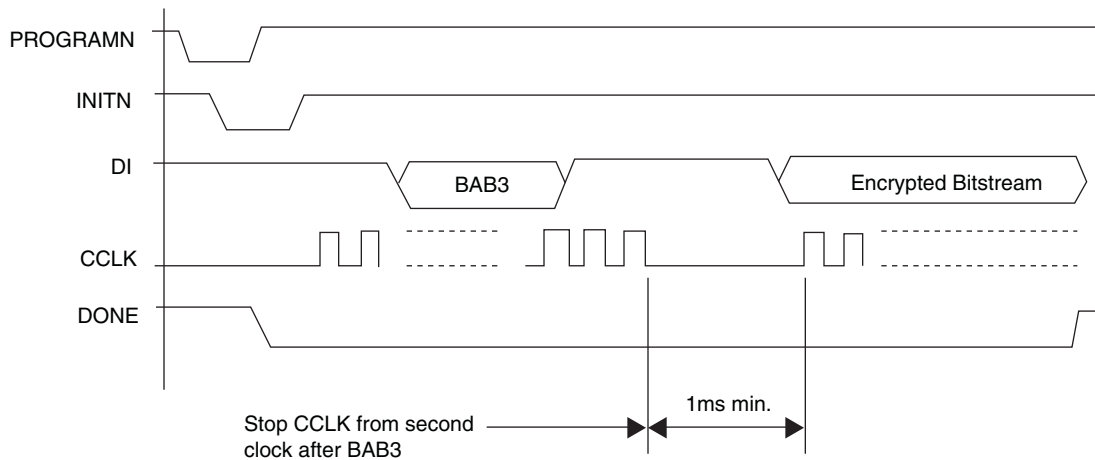
Table 16-1. Configuration Ports

Interface	Port
sysCONFIG	SPI <sup>1</sup>
	SPI <sub>m</sub> <sup>1</sup>
	Slave Serial (SCM) <sup>2</sup>
	Slave Parallel (PCM) <sup>2</sup>
ispJTAG™	IEEE 1532 (Erase, Program, Verify) <sup>3</sup>
	Bitstream Burst (Fast Program)

1. Supports subset of the CCLK frequencies specified in the LatticeECP2/M Family Data Sheet.
2. Users must adhere to the appropriate conditions for the CCLK signal as described below.
3. Does not support encrypted bitstreams.

The sysCONFIG interface allows the user to input data serially, using serial configuration mode (SCM) or SPI Serial Flash, or in parallel, using the parallel configuration mode (PCM). In general, the connection between the FPGA and the configuration device consists of a clock, chip select(s), a write signal (in PCM), and data. During configuration all data written to the FPGA is ignored until a special preamble is detected in the bitstream. Everything after the preamble is configuration data. The normal preamble is BAB3 (hex), however encrypted bitstreams contain a different preamble. When using SCM mode, any CCLK frequencies from 2.5MHz to 45MHz are supported but it is required to stop the CCLK after loading the BAB3 (hex) encryption preamble as shown below.

Figure 16-2. CCLK Timing



To ensure proper programming when using encrypted bitstreams in SPI mode only a subset of CCLK frequencies are supported. The supported frequencies are shown in the Lattice ECP2/M data sheet, available on the Lattice web site at [www.latticesemi.com](http://www.latticesemi.com).

The JTAG port, which conforms to IEEE 1149.1 and IEEE 1532 standards, can input data in Bitstream-Burst mode (Fast Program) or 1532 mode. Configuration bitstreams created for Bitstream-Burst mode (Fast Program) are identical to the configuration bitstreams created for sysCONFIG mode. The bitstream contains a header, a preamble, configuration data, and frame data CRC. However, 1532 mode makes use of standard JTAG instructions to configure the device. In other words, the configuration data file contains configuration data only. Because 1532 mode data files do not contain a preamble, they cannot be used to input encrypted configuration files.

In addition to being a configuration interface, JTAG also allows the user to program the 128-bit encryption key. In fact, JTAG is the only way to program the key.

For detailed information on LatticeECP2/M configuration including bitstream file sizes, refer to Lattice technical note TN1108, *LatticeECP2/M sysCONFIG Usage Guide*, available on the Lattice web site at [www.latticesemi.com](http://www.latticesemi.com). Note that bitstream sizes vary depending on the configuration mode.

## Bitstream Encryption/Decryption Flow

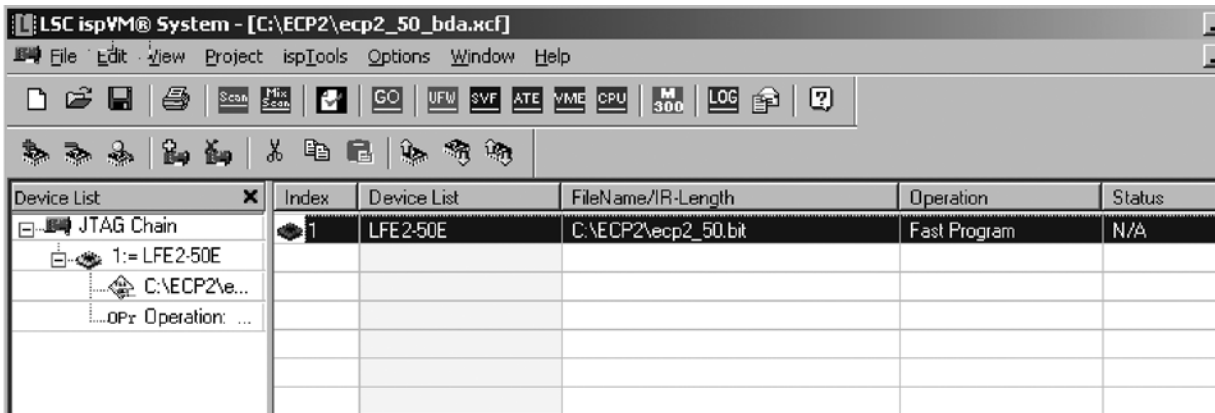
The LatticeECP2/M “S” versions support both encrypted and non-encrypted bitstreams. Since the non-encrypted flow is covered in technical note TN1108, this document will concentrate on the additional steps needed for the encrypted flow. The encrypted flow adds only two steps to the normal FPGA design flow, encryption of the configuration bitstream and programming the encryption key into the LatticeECP2/M “S” version devices.

### Encrypting the Bitstream

As with any other Lattice FPGA design flow, the engineer must first create the design using a version of ispLEVER® which supports the encryption feature. The design is synthesized, mapped, placed and routed, and verified. Once the engineer is satisfied with the design a bitstream is created and loaded into the FPGA for final debug. After the design has been debugged it is time to secure the design.

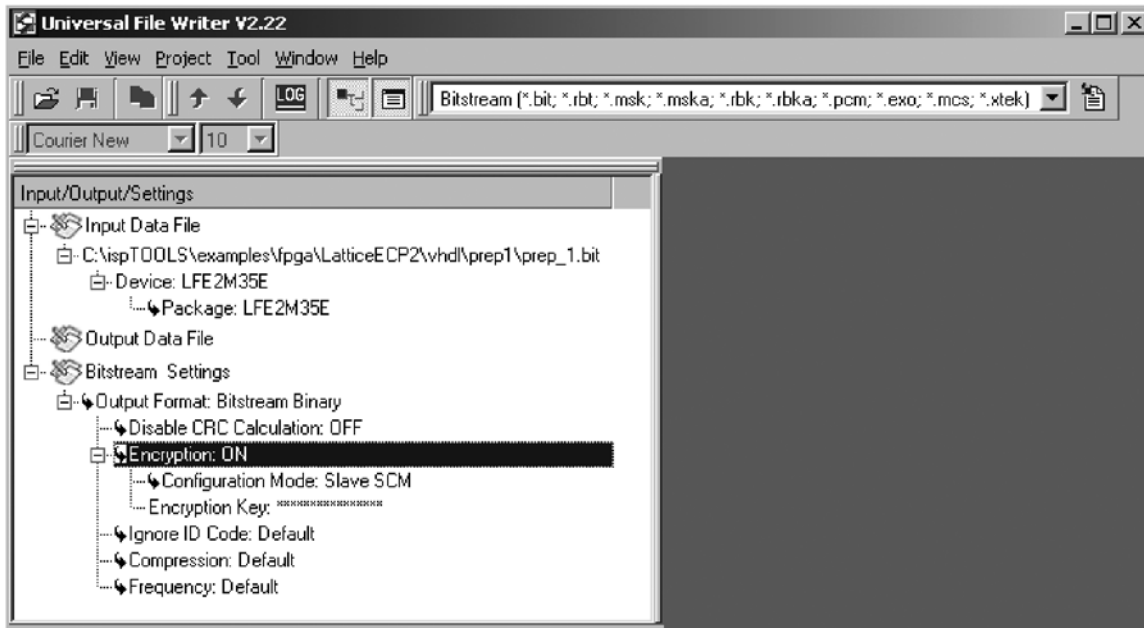
The bitstream can be encrypted using an appropriate version of ispLEVER by going to the Tools pull-down menu and selecting Security features or by using the Universal File Writer (UFW), which is part of the Lattice ispVM® System tool suite. The file is encrypted using ispVM as follows.

**Figure 16-3. ispVM Main Window**



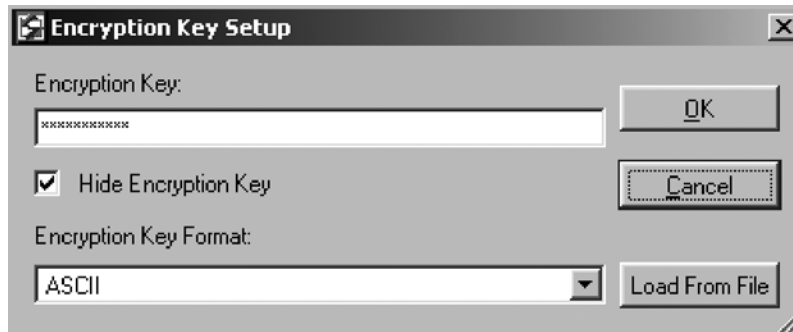
1. Start ispVM. You can start ispVM from within ispLEVER or from the **Start -> Programs** menu in Windows. You should see a window that looks similar to Figure 16-3. Click on the **UFW** button on the toolbar. You will see a window similar to Figure 16-4.

Figure 16-4. Universal File Writer (Encryption Option)



2. Double click on **Input Data File** and browse to the non-encrypted bitstream created using ispLEVER. Double click on **Output Data File** and select an output file name. Right-click on **Encryption** and select **ON**. Right-click on **Configuration Mode** and select the type of device the FPGA will be configuring from, such as SPI Serial Flash. Right-click on **Encryption Key** and select **Edit Encryption Key**. You will see a window that looks similar to Figure 16-5.

Figure 16-5. Encryption Key Dialog Window



3. Enter the desired 128-bit key. The key can be entered in Hexadecimal or ASCII. Hex supports 0 through f and is not case sensitive. ASCII supports all alphanumeric characters, as well as spaces, and is case sensitive. Note: be sure to remember this key. Lattice cannot recover an encrypted file if the key is lost. Click on **OK** to go back to the main UFW window.
4. From the menu bar, click on **Project -> Generate** to create the encrypted bitstream file.
5. The bitstream can now be loaded directly into non-volatile configuration storage (such as SPI Serial Flash) using a Lattice ispDOWNLOAD® Cable, a third-party programmer, or any other method normally used to program a non-encrypted bitstream. However, before the LatticeECP2/M can configure from the encrypted file the 128-bit key used to encrypt the file must be programmed into the one-time programmable fuses on the FPGA.

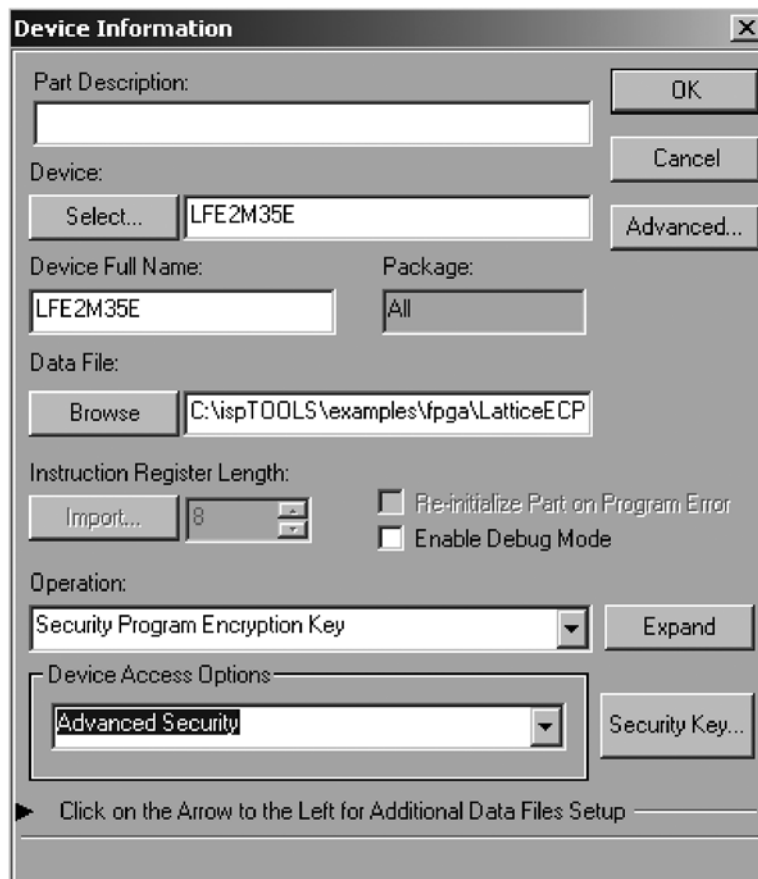
### Programming the 128-bit Key

The next step is to program the 128-bit encryption key into the one-time programmable fuses on the LatticeECP2/M. Note that this step is separated from file encryption to allow flexibility in the manufacturing flow. For instance, the board manufacturer might program the encrypted file into the SPI Serial Flash, but the key might be programmed at the user's facility. This flow adds to design security and it allows the user to control over-building of a design. Over-building occurs when a third party builds more boards than are authorized and sells them to grey market customers. If the key is programmed at the factory, then the factory controls the number of working boards that enter the market. The LatticeECP2/M "S" version will only configure from a file that has been encrypted with the same 128-bit key that is programmed into the FPGA.

To program the key into the LatticeECP2/M "S" version, proceed as follows.

1. Attach a Lattice ispDOWNLOAD cable from a PC to the JTAG connector wired to the LatticeECP2/M (note that the 128-bit key can only be programmed into the LatticeECP2/M using the JTAG port). Apply power to the board.
2. Start the ispVM System software. ispVM can be started from within the ispLEVER design tool or from the **Start -> Programs** menu in Windows. You should see a window that looks similar to Figure 16-3. If the window does not show the board's JTAG chain then proceed as follows. Otherwise, proceed to step 3.
  - a. Click the **SCAN** button in the toolbar to find all Lattice devices in the JTAG chain. The chain shown in Figure 16-3 has only one device, the LatticeECP2.

**Figure 16-6. Device Information Window (Encryption Option)**



3. Double-click on the line in the chain containing the LatticeECP2. This will open the Device Information window (see Figure 5). From the Device Access Options drop-down box select **Security Mode**, then click on the **Security Key** button to the right. The window will look similar to Figure 16-7.

**Figure 16-7. Enter the Encryption Key**

4. Enter the desired 128-bit key. The key can be entered in Hexadecimal or ASCII. Hex supports 0 through f and is not case sensitive. ASCII supports all alphanumeric characters, as well as spaces, and is case sensitive. This key must be the same as the key used to encrypt the bitstream. The LatticeECP2/M will only configure from an encrypted file whose encryption key matches the one loaded into the FPGA's one-time programmable fuses. *Note: be sure to remember this key. Once the Key Lock is programmed, Lattice Semiconductor cannot read back the one-time programmable key.*
  - a. The key can be saved to a file using the **Save to File** button. The key will be encrypted using an 8-character password that the user selects. The name of the file will be <project\_name>.bek. In the future, instead of entering the 128-bit key, simply click on **Load from File** and provide the password.
5. Programming the Key Lock secures the 128-bit encryption key. Once the Key Lock is programmed and the device is power cycled, the 128-bit encryption key cannot be read out of the device. When satisfied, type **Yes** to confirm, then click **Apply**.
6. From the main ispVM window (Figure 16-3) click on the green **GO** button on the toolbar to program the key into the LatticeECP2/M one-time programmable fuses. When complete, the LatticeECP2/M will only configure from a bitstream encrypted with a key that exactly matches the one just programmed.

## Verifying a Configuration

As an additional security step when an encrypted bitstream is used, the readback path from the SRAM fabric is automatically blocked. In this case, for all ports, a read operation will produce all zeros. However, even when the configuration bitstream has been encrypted and readback disabled, there are still ways to verify that the bitstream was successfully downloaded into the FPGA.

If the SRAM fabric is programmed directly, the data is first decrypted and then the FPGA performs a CRC on the data. If all CRCs pass, configuration was successful. If a CRC does not pass, the DONE pin will stay low and INITN will go from high to low (for more information on this type of error, refer to Lattice technical note TN1108, *LatticeECP2/M sysCONFIG Usage Guide*).

If the encrypted data is stored in non-volatile configuration memory, such as SPI Serial Flash, the data is stored encrypted. A bit-for-bit verify can be performed between the encrypted configuration file and the stored data.

## File Formats

The base binary file format is the same for all non-encrypted, non-1532 configuration modes. Different file types (hex, binary, ASCII, etc.) may ultimately be used to configure the device, but the data in the file is the same. Table 16-2 shows the format of a non-encrypted bitstream. The bitstream consists of a comment field, a header, the preamble, and the configuration setup and data.

**Table 16-2. Non-Encrypted Configuration Data**

Frame	Contents	Description
Comments	(Comment String)	ASCII Comment (Argument) String and Terminator
Header	1111...1111	16 Dummy bits
	1011110110110011	16-bit Standard Bitstream Preamble (0xBDB3)
Verify ID		64 bits of command and data
Control Register 0		64 bits of command and data
Reset Address		32 bits of command and data
Write Increment		32 bits of command and data
Data 0		Data, 16-bit CRC, and Stop bits
Data 1		Data, 16-bit CRC, and Stop bits
.	.	.
.	.	.
.	.	.
Data n-1		Data, 16-bit CRC, and Stop bits
End	1111...1111	Terminator bits and 16-bit CRC
Usercode		64 bits of command and data
SED CRC		64 bits of command and data
Program Security		32 bits of command and data
Program Done		32 bits of command and data, 16-bit CRC
NOOP	1111...1111	64 bits of NOOP data
End	1111...1111	32-bit Terminator (all ones)

Note: The data in this table is intended for reference only.

Table 16-3 shows a bitstream that is built for encryption but has not yet been encrypted. The highlighted areas will be encrypted. The changes between Table 16-2 and Table 16-3 include the following:

- The Program Security frame (readback disable) has been moved to the beginning of the file so that readback is turned off at the very beginning of configuration. This is an important security feature that prevents someone from interrupting the configuration before completion and reading back unsecured data.
- A copy of the usercode is placed in the non-encrypted comment string. This has been done to allow the user a method to identify an encrypted file. For example, the usercode could be used as a file index or a “hint”. Note that the usercode itself, while encrypted in the configuration data file, is not encrypted on the device. At configuration the usercode is decrypted and placed in the JTAG Usercode register. This allows the user a method to identify the data in the device. The JTAG Usercode register can be read back at any time, even when all SRAM readback paths have been turned off. The usercode can be set to any 32-bit value. For information on how to set usercode, see ispLEVER’s help facility.
- A copy of CONFIG\_MODE, one of the global ispLEVER preferences, is placed in the non-encrypted comment string. CONFIG\_MODE can be SPI/SPIIm, Slave SCM, or Slave PCM.

Note that if the global COMPRESS\_CONFIG option is turned ON using ispLEVER's Design Planner or UFW, data compression will be performed before encryption.

**Table 16-3. Configuration File Just Before Encryption**

Frame	Contents	Description
Comments	(Comment String)	ASCII Comment (Argument) String and Terminator
Header	1111...1111	16 Dummy Bits
		16-bit Standard Bitstream Preamble
Verify ID		64 bits of Command and Data
Control Register 0		64 bits of Command and Data
Program Security		32 bits of Command and Data
Reset Address		32 bits of Command and Data
Write Increment		32 bits of Command and Data
Data 0		Data, 16-bit CRC, and Stop Bits
Data 1		Data, 16-bit CRC, and Stop Bits
.	.	.
.	.	.
.	.	.
Data n-1		Data, 16-bit CRC and Stop Bits
End	1111...1111	Terminator Bits and 16-bit CRC
Usercode		64 Bits of Command and Data
SED CRC		64 Bits of Command and Data
Program Done		32 Bits of Command and Data, 16-bit CRC
NOOP	1111...1111	64 bits of NOOP data
End	1111...1111	32-bit Terminator (All Ones).

Note: The data in this table is intended for reference only. The shaded areas will be encrypted.

Once encrypted, besides the obvious encryption of the data itself, the file will have additional differences from a non-encrypted file (refer to Tables 16-4, 16-5, and 16-6).

- There are three preambles, the encryption preamble, alignment preamble, and the bitstream preamble. The alignment preamble marks the beginning of the encrypted data. The entire original bitstream, including the bitstream preamble are all encrypted, per Table 16-3. The comment string, the encryption preamble, dummy data, and alignment preamble are not encrypted.
- The decryption engine within the FPGA takes some time to perform its task; extra time is provided in one of two ways. For master configuration modes (SPI and SPIm) the FPGA drives the configuration clock, so when extra time is needed the FPGA stops sending configuration clocks. For slave configuration modes (Bitstream-Burst, Slave Serial, and Slave Parallel) the data must be padded to create the extra time. Because of this there are several different file formats for encrypted data (see Tables 16-4, 16-5, and 16-6). Note that because of the time needed to decrypt the bitstream it takes longer to configure from an encrypted data file than it does from a non-encrypted file. The bitstream sizes may vary depending on the configuration mode. For exact file sizes, refer to Lattice technical note TN1108, *LatticeECP2/M sysCONFIG Usage Guide*, available on the Lattice website at [www.latticesemi.com](http://www.latticesemi.com).

**Table 16-4. Encrypted File Format for a Master Mode**

Frame	Contents	Description
Comments	(Comment String)	ASCII Comment (Argument) String and Terminator.
Header	1111...1111	16 Dummy bits.
		16-bit Encryption Preamble.
30,000 Filler Bits		This allows time for the device to load and hash the 128-bit encryption key.
Alignment Preamble		16-bit Alignment Preamble.
	1	1-bit Dummy Data.
Data		There are no dummy filler bits when the bitstream is generated for master programming modes. The CCLK of the master device stops the clock when it needs time to decrypt the data. It resumes the clock when ready for new data - Encrypted.
Program Done		32-bit Program Done Command - Encrypted.
End	1111...1111	32-bit Terminator (all ones) - Encrypted.
Filler Bits		Filler to meet the bound requirement.
Dummy Data	1111...1111	200 bits of Dummy Data (all ones). Provides a delay to turn off the decryption engine.

Note: The data in this table is intended for reference only. The shaded area is encrypted data.

**Table 16-5. Encrypted File Format for a Slave Serial Mode**

Frame	Contents	Description
Comments	(Comment String)	ASCII Comment (Argument) String and Terminator.
Header	1111...1111	2 Dummy Bytes.
		16-bit Encryption Preamble
30,000 Filler Bits		This allows time for the device to load and hash the 128-bit encryption key.
Alignment Preamble		16-bit Alignment Preamble.
	1	1-bit Dummy Data.
Data		128 bits of Configuration Data.
		64 bits of all ones data. Provides a delay for the decryption engine to decrypt the 128 bits of data just received. If the peripheral device can provide the needed 64 clocks while pausing data, then the 64 bits of dummy data are not required, saving file size.
	...	
		Last 128 bits of the last Frame of Configuration Data.
		64 bits of all ones data. Provides a delay for the decryption engine to decrypt the 128 bits of data just received. If the peripheral device can provide the needed 64 clocks while pausing data, then the 64 bits of dummy data are not required, saving file size.
Program Done		32-bit Program Done Command - Encrypted.
End		32-bit Terminator (all ones) - Encrypted.
Filler Bits		Filler to meet the bound requirement.
Delay		64 bits of all ones data. Delay to decrypt the Program Done command and the filler.
Dummy Data	1111...1111	200 bits of Dummy Data (all ones), to provide delay to turn off the decryption engine.

Note: The data in this table is intended for reference only. The shaded area is encrypted data.

**Table 16-6. Encrypted File Format for a Slave Parallel Mode**

Frame	Contents	Description
Comments	(Comment String)	ASCII Comment (Argument) String and Terminator.
Header	1111...1111	2 Dummy Bytes.
		2-byte Encryption Preamble.
30,000 Filler Bytes		This allows time for the device to load and hash the 128-bit encryption key.
Alignment Preamble		2-byte Alignment Preamble.
	11111111	1-byte Dummy Data.
Data		16 bytes of Configuration Data.
		64 bytes (clocks) of all ones data. Provides a delay for the decryption engine to decrypt the 16 bytes of data just received. If the peripheral device can provide the needed 64 clocks while pausing data, then the 64 bytes of dummy data are not required, saving file size.
	...	
		16 bytes of Configuration Data.
		64 bytes (clocks) of all ones data. Provides a delay for the decryption engine to decrypt the 16 bytes of data just received. If the peripheral device can provide the needed 64 clocks while pausing data, then the 64 bytes of dummy data are not required, saving file size.
Program Done		4-byte Program Done Command - Encrypted.
End		4-byte Terminator (all ones) - Encrypted.
Filler Bits		Filler to meet the bound requirement.
Delay		64 bytes of all ones data. Delay to decrypt the Program Done command and the filler.
Dummy Data	1111...1111	200 bytes of Dummy Data (all ones), to provide delay to turn off the decryption engine.

Note: The data in this table is intended for reference only. The shaded area is encrypted data.

## Decryption Flow

From the user's point of view, as compared to the encryption flow just discussed, the decryption flow is much simpler.

When data comes into the FPGA the decoder starts looking for the preamble (see Figure 16-1) and all information before the preamble is ignored. The preamble, along with the compression bit in Control Register 0, determines the path of the configuration data.

If the decoder detects a standard bitstream preamble in the bitstream it knows that this is a non-encrypted data file. The decoder then examines Control Register 0 in the bitstream to determine if the file has been compressed. If the file has not been compressed then the Raw data path is selected (see Figure 16-1). If the file has been compressed then the Decompressed path is selected; CRC is then checked and the SRAM fuses programmed.

If the decoder detects an encryption preamble in the bitstream it knows that this is an encrypted data file. If an encryption key has not been programmed, the encrypted data is blocked and configuration fails (the DONE pin stays low), if the proper key has been programmed then configuration can continue. The next block read contains 30,000 clocks of filler data. This delay allows time for the FPGA to read the key fuses and prepare the decryption engine. The decoder keeps reading the filler data looking for the alignment preamble. Once found, it knows that the following data needs to go through the decryption engine. It first looks for the standard preamble. Once found, then it reads the Control Register 0 frame. The decoder then examines the decrypted Control Register 0 contents to determine if the file has been compressed. If the file has not been compressed then the Decrypted data path is used, if the file has been compressed then the decrypted data is passed through the decompression engine and the Decompressed path is selected (refer to the block diagram, Figure 16-1). CRC is then checked and the SRAM fuses programmed once the bitstream preamble is read. The decryption and decompression engines are turned off

when the internal Done bit is set at the end of configuration. This is done so that if there is any data overflow (to other devices in a chain) the downstream devices will receive raw data from configuration storage.

But what happens if the key in the FPGA does not match the key used to encrypt the file? Once the data is decrypted, the FPGA expects to find a valid standard bitstream preamble (BDB3), along with proper commands and data that pass CRC checks. If the keys do not match then the decryption engine will not produce a proper configuration bitstream; either configuration will not start because the preamble was not found (the INITN pin stays high and the DONE pin stays low) or CRC errors will occur, causing the INITN pin to go low to indicate the error (see Lattice technical note TN1108, *LatticeECP2/M sysCONFIG Usage Guide*, for more information on INITN and DONE).

## Reference Documents

- Lattice Technical Note TN1108, *LatticeECP2/M sysCONFIG Usage Guide*
- Federal Information Processing Standard Publication 197, Nov. 26, 2001. Advanced Encryption Standard (AES)

## Technical Support Assistance

Hotline: 1-800-LATTICE (North America)  
+1-503-268-8001 (Outside North America)  
e-mail: [techsupport@latticesemi.com](mailto:techsupport@latticesemi.com)  
Internet: [www.latticesemi.com](http://www.latticesemi.com)

## Revision History

Date	Version	Change Summary
April 2006	01.0	Initial release.
September 2006	01.1	Added information throughout for LatticeECP2M support.
		Updated screen shots based on the latest software version.
		Provided clarification in Table 1.
		Changed Bitstream Preamble to Alignment Preamble through out the document.
		Reworded sections of the document to provide additional information/clarification.
March 2007	01.2	Added "S" series encryption information throughout.
August 2007	01.3	Updated for "S" series reduced frequency support and special requirement for CCLK and TCK on LatticeSCM, and PCM and JTAG configuration modes.