XAPP1260 (v1.1) December 21, 2017

eFUSE Programming on a Device Programmer

Author: Randal Kuramoto

Summary

This application note provides guidelines for programming the eFUSE bits in 7 series and UltraScale™ FPGAs on a device programmer. Device programmers are effective for:

- High-volume production programming
- Preprogramming devices with sensitive data into the eFUSE before they are sent for assembly at a board manufacturing site
- Cataloging the DNA (unique serial numbers) of programmed devices

IMPORTANT: *The eFUSE bits are one-time programmable and cannot be changed after they are programmed.*

Specifying the exact settings and a verification method is important to ensure the correct programming of the eFUSE bits. This application note provides guidelines for the eFUSE bit settings. A reference design is provided for creating an encrypted verification bitstream for a device programmer.

The reference design files for this application note can be downloaded from the Xilinx website. For detailed information about the design files, see Reference Design.

Introduction to Programmable eFUSE

Figure 1 shows the inputs to a device programmer for programming and verifying the eFUSE bits, as well as the uses of the eFUSE bits in the 7 series FPGA.

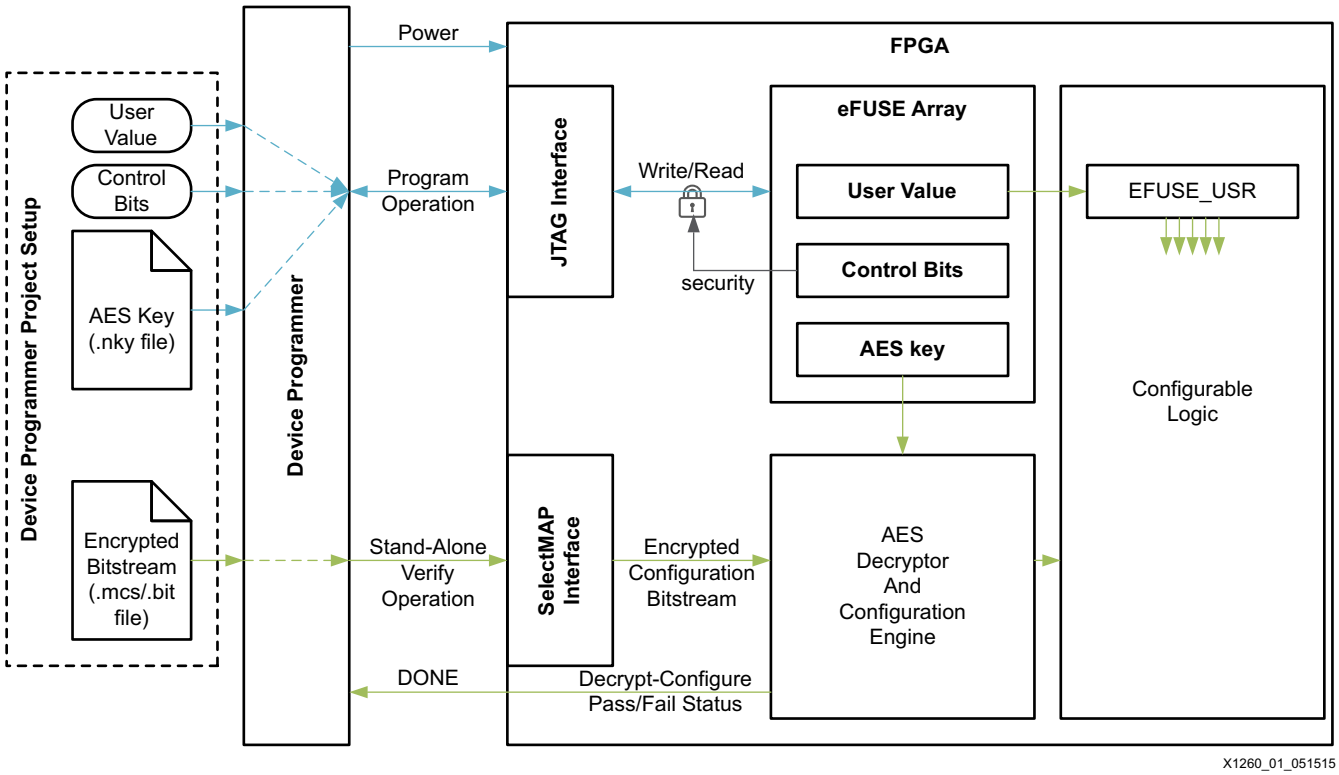

Figure 1: **Block Diagram of 7 Series FPGA eFUSE Array and Associated Functions**

The FPGA bitstream can be encrypted to prevent unauthorized use of a copied design. The FPGAs use the advanced encryption standard (AES) that is issued in the Federal Information Processing Standards Publication 197 (FIPS PUB 197) by the National Institute of Standards and Technology (NIST). The AES is a symmetric key algorithm, which means that the same secret key is used for encrypting and decrypting a bitstream. The Vivado® design tools use the key to encrypt the bitstream when writing the bitstream file. The FPGA uses the key to decrypt a bitstream during configuration. For the FPGA AES decryptor, the AES key can reside in battery-backed RAM (BBRAM) or in nonvolatile, one-time programmable eFUSE bits within the FPGA. Only the eFUSE bits can be programmed on a device programmer. The BBRAM cannot be effectively programmed on a device programmer because the RAM loses the settings when the device is removed from the programmer.

Additionally, the UltraScale FPGAs support authentication via the asymmetric RSA public-private key algorithm. For authentication, the Vivado design tools use the RSA private key when generating a signed bitstream. A digest of the public key is programmed into the FPGA eFUSE to authenticate the signed bitstream. RSA authentication can be used alone or with AES encryption.

The advantages of using the nonvolatile eFUSE bits to store the key include:

- No battery is required to preserve the key value when power is removed from the FPGA
- Lower bill of materials cost (than with a battery)
- The eFUSE can be preprogrammed into the device at a secure location on a device programmer before assembly at a board manufacturing site

The nonvolatile eFUSE bits can also store a user-defined 32-bit *user value* that the FPGA configurable logic design can read from the EFUSE_USR primitive. Also, external devices can read this value through the FPGA JTAG port.

Additional eFUSE bits are used to control FPGA security settings, such as read-security or write-security of the eFUSE against further reading or writing.

IMPORTANT: *The secret AES key must always be read-secured during eFUSE programming.*

All user-programmable eFUSE bits are a "0" value when shipped from Xilinx. Selected eFUSE bits are programmed to "1" through the FPGA JTAG port. The eFUSE programming procedure includes verifying the programmed eFUSE bits and setting the selected security options. After completion of the eFUSE programming procedure, the eFUSE bits are read-secured and cannot be directly inspected.

RECOMMENDED: *For device programming, Xilinx recommends using a sample bitstream designed for the device programmer and encrypted with a matching key to validate the correct eFUSE key programming on a device programmer.*

This sample bitstream can also be used to identify a device that has been programmed with a matching key in situations when the device programmer is used in stand-alone verification mode to recheck an inventory of programmed devices. The sample bitstream is sent to the FPGA SelectMAP configuration interface during the stand-alone verify operation.

See the *7 Series FPGAs Configuration User Guide* (UG470) [Ref 1] or the *UltraScale Architecture Configuration Guide* (UG570) [Ref 2] for more information on eFUSE features and device configuration.

Using the Device Programmer for eFUSE Programming

The device programmer can be used to:

- Program the eFUSE bits in the FPGA and record the FPGA DNA value
- Stand-alone verify that an FPGA has been previously programmed with a specific AES key or RSA public key digest

Programming the eFUSE

This section provides the steps for programming the eFUSE in an FPGA on a device programmer.

1. The programming manager creates a device programmer project that requires:
   a. Specification of input files, including:
      - NKY file, which contains the AES key or RSA public key digest
      - Encrypted/authenticated verification bitstream file
   b. Definition of settings for programmable options, including:
      - User-defined 32-bit value
      - eFUSE control bits for read-protection, write-protection, and other security features
2. The device programmer operator programs devices on the device programmer, and performs these operations:
   a. Identifies the device and records the device DNA
   b. Programs and verifies the eFUSE bits in the device
   c. Secondary verification by downloading the encrypted/authenticated verification bitstream to the device

After eFUSE programming, the eFUSE within the device is typically secured, which means the AES key is secured and cannot be read from the device.

At a later time, the device programmer can perform a stand-alone operation to identify and verify that a previously programmed FPGA contains the correct AES key or RSA public key digest in its eFUSE, even though the eFUSE is secured in the device. For this stand-alone verify operation, the device programmer downloads an encrypted/authenticated verification bitstream to the device. If the device successfully configures from the encrypted/authenticated verification bitstream, the device is verified to contain the matching keys.

Preparing for a Device Programmer

The required preparations for programming on a device programmer are:

- Establish the device programmer site or service
- Decide on the security setting for each eFUSE control bit option
- Build the encrypted/authenticated verification bitstream from the provided reference design using your AES key or RSA key and save the output NKY file
- Deliver device samples and device programmer settings and files to the device programming site

Establish the Device Programmer Site or Service

Contact the local authorized Avnet distributor for device programming services or contact an FAE for in-house programmer options.

IMPORTANT: *It is critical to establish the device programmer site or service as early as possible. It can take many weeks to develop a device programmer socket for a device-package that has not been previously established.*

Define the Device Programmer Settings and Input Files

A complete and clear list of settings with required files must be provided for the device programmer. Use Table 1 or Table 2 for a 7 series or UltraScale FPGA, respectively, as a checklist and definition sheet for the device programmer settings and input files. Supply the completed Table 1 or Table 2 with the required files to the device programming site with device samples for the first few pilot devices.

Table 1: **7 Series Device Programmer Settings**

Description of Function When Programmed	Setting Type	Device Programmer Input File or Setting (Define your settings and file names here)
256-bit AES key This key can decrypt an encrypted bitstream for FPGA configuration.	NKY file name The NKY file contains the 256-bit AES key value	
32-bit user-defined value Optional user-defined device identifier value accessible from the EFUSE_USR primitive in the FPGA design or from JTAG (if R_EN_B_User is not programmed). If not used, then leave blank.	8-character hexadecimal value `[31:0]`	

Table 1: **7 Series Device Programmer Settings *(Cont'd)***

Description of Function When Programmed	Setting Type	Device Programmer Input File or Setting (Define your settings and file names here)
CFG_AES_Only (FUSE_CNTL[0]) Forces the use of the AES key stored in eFUSE, and prevents bitstreams without a matching key from configuring the FPGA. (If this bit is not programmed, the FPGA might configure using an unencrypted bitstream, or a bitstream encrypted with a key value stored in BBRAM.) **CAUTION!** *Return material authorization (RMA) returns cannot be accepted if this bit is programmed because it prevents Xilinx test bitstreams from being loaded into the device for failure analysis.*	Yes=program to '1' No=do not program ('0') Typical setting=No	
AES_Exclusive (FUSE_CNTL[1]) Disables partial reconfiguration from external configuration interfaces. However, partial reconfiguration is allowed via the ICAPE2. **CAUTION!** *If the AES_Exclusive bit is programmed, return material authorization (RMA) returns are limited in device analysis and debug. The recommended alternative, that achieves the same security function, is to set the BITSTREAM.READBACK.SECURITY property to Level2 in the Vivado design tools before generating the bitstream.*	Yes=program to '1' No=do not program ('0') Typical setting=No	
W_EN_B_Key_User (FUSE_CNTL[2]) Disables the programming of the AES key and FUSE_USER.	Yes=program to '1' No=do not program ('0') Typical setting=Yes	

Table 1: **7 Series Device Programmer Settings *(Cont'd)***

Description of Function When Programmed	Setting Type	Device Programmer Input File or Setting (Define your settings and file names here)
R_EN_B_Key (FUSE_CNTL[3]) Disables reading of AES key as well as programming of AES key and 32-bit user-defined value.  **IMPORTANT:** *The R_EN_B_Key bit must be programmed to prevent the secret AES key from being read through the JTAG port.*	Yes=program to '1' No=do not program ('0') REQUIRED setting=Yes, if the AES key is programmed	Yes Do not change unless you are not programming an AES key
R_EN_B_User (FUSE_CNTL[4]) Disables reading of 32-bit user-defined value through JTAG as well as programming of AES key and user-defined value. This does not disable reading the user-defined value through the EFUSE_USR component, although it disables reading the user-defined value through the JTAG port.	Yes=program to '1' No=do not program ('0') Typical setting=No	
W_EN_B_Cntl (FUSE_CNTL[5]) Disables programming of control bits.	Yes=program to '1' No=do not program ('0') Typical setting=Yes	
Encrypted verification bitstream The bitstream is encrypted using the AES key from the NKY file and is supplied in MCS format.  **CAUTION!** *Do NOT use your actual design. See Encrypted Verification Bitstream for bitstream design constraints.*	MCS file name	

Table 2: **UltraScale Device Programmer Settings**

Description of Function When Programmed	Setting Type	Device Programmer Input File or Setting (Define Your Settings and File Names Here)
256-bit AES keys (FUSE_KEY) These keys can decrypt an encrypted bitstream for FPGA configuration. If using the obfuscated key feature, supply the KeyObfuscate values instead of the secret key values from the output NKY file, i.e., provide a copy of the output NKY file with the secret Key0 values deleted. For UltraScale SSI devices, different key values can be specified for each SLR, numbered in the NKY file: 0, 1, ... If not used, leave blank.	Output NKY file name The NKY file contains the 256-bit AES key values	
384-bit RSA public key hash value (FUSE_RSA) This RSAPublicKeyDigest value authenticates a signed bitstream. If not used, leave blank.	Output NKY file name The NKY file contains the 384-bit RSA public key digest value	
32-bit user-defined (FUSE_USER/EFUSE_USR) value Optional user-defined device identifier value accessible from the EFUSE_USR primitive in the FPGA design or from the JTAG FUSE_USER instruction (if R_EN_B_User is not programmed). If not used, leave blank.	8-character hexadecimal value[31:0]	
128-bit user-defined (FUSE_USER_128) value Optional user-defined device identifier value accessible from the JTAG FUSE_USER_128 instruction (if R_EN_B_User is not programmed). If not used, leave blank.	64-character hexadecimal value[127:0]	
R_DIS_KEY (FUSE_CNTL[0]) Disables the reading the FUSE_KEY CRC check result and the programming of the FUSE_KEY. The FUSE_KEY cannot be directly read from the device under any condition. However, the FUSE_KEY value can be tested via a CRC check method when this bit is not set.	Yes=program to '1' No=do not program ('0') Typical setting=Yes is required, if the FUSE_KEY is programmed to secure the AES key	
R_DIS_USER (FUSE_CNTL[1]) Disables the reading of the FUSE_USER value via JTAG and the programming of FUSE_USER. This does not disable access to the user code via the EFUSE_USR component.	Yes=program to '1' No=do not program ('0')	
R_DIS_SEC (FUSE_CNTL[2]) Disables the reading and programming of FUSE_SEC.	Yes=program to '1' No=do not program ('0')	

Table 2: **UltraScale Device Programmer Settings *(Cont'd)***

Description of Function When Programmed	Setting Type	Device Programmer Input File or Setting (Define Your Settings and File Names Here)
W_DIS_CNTL (FUSE_CNTL[5]) Disables the programming of FUSE_CNTL.	Yes=program to '1' No=do not program ('0') Typical setting=Yes to prevent further FUSE_CNTL changes	
R_DIS_RSA (FUSE_CNTL[6]) Disables the reading and programming of the FUSE_RSA public key digest.	Yes=program to '1' No=do not program ('0')	
W_DIS_KEY (FUSE_CNTL[7]) Disables the programming of FUSE_KEY. Redundant to programming disable effect of R_DIS_KEY.	Yes=program to '1' No=do not program ('0') Typical setting=Yes, if the FUSE_KEY is programmed to prevent further changes	
W_DIS_USER (FUSE_CNTL[8]) Disables the programming of FUSE_USER.	Yes=program to '1' No=do not program ('0')	
W_DIS_SEC (FUSE_CNTL[9]) Disables the programming of FUSE_SEC.	Yes=program to '1' No=do not program ('0') Typical setting=Yes to prevent further FUSE_SEC changes	
W_DIS_RSA (FUSE_CNTL[15]) Disables the programming of the FUSE_RSA public key digest.	Yes=program to '1' No=do not program ('0') Typical setting=Yes, if the FUSE_RSA is programmed to prevent further changes	
W_DIS_USER_128 (FUSE_CNTL[16]) Disables the programming of FUSE_USER_128.	Yes=program to '1' No=do not program ('0')	
CFG_AES_Only (FUSE_SEC[0]) Forces the bitstream through the decryptor, i.e., only allows encrypted bitstreams with the matching key to configure the device. **CAUTION!** *Return material authorization (RMA) returns cannot be accepted if this bit is programmed because it prevents Xilinx test bitstreams from being loaded into the device for failure analysis. The Vivado Indirect SPI/BPI flash programming flow cannot be used if this bit is programmed. You must have external configuration memories programmed BEFORE you program this fuse if the Vivado design tools are used for programming.*	Yes=program to '1' No=do not program ('0')	

Table 2: **UltraScale Device Programmer Settings *(Cont'd)***

Description of Function When Programmed	Setting Type	Device Programmer Input File or Setting (Define Your Settings and File Names Here)
FUSE_SEC[1] For encrypted bitstreams, forces the use of the AES key stored in eFUSE. When this bit is *not* programmed, encryption and the key source can be selected via bitstream options. The FPGA can be configured using an unencrypted bitstream or a bitstream encrypted with a key value stored in BBRAM or eFUSE.	Yes=program to '1' No=do not program ('0')	
RSA_AUTH (FUSE_SEC[2]) Forces RSA authentication, i.e., only allows bitstreams signed with the corresponding key to configure the device. **CAUTION!** *RMA returns cannot be accepted if this bit is programmed because it prevents Xilinx test bitstreams from being loaded into the device for failure analysis. The Vivado Indirect SPI/BPI flash programming flow cannot be used if this bit is programmed. You must have external configuration memories programmed BEFORE you program this fuse if the Vivado design tools are used for programming.*	Yes=program to '1' No=do not program ('0')	
FUSE_SEC[3] Disables external JTAG pins. **CAUTION!** *If this bit is programmed, RMA device analysis and debug is limited.*	Yes=program to '1' No=do not program ('0')	
SCAN_DISABLE (FUSE_SEC[4]) Disables Xilinx test access. **CAUTION!** *If this bit is programmed, RMA device analysis and debug is limited.*	Yes=program to '1' No=do not program ('0')	

Table 2: **UltraScale Device Programmer Settings *(Cont'd)***

Description of Function When Programmed	Setting Type	Device Programmer Input File or Setting (Define Your Settings and File Names Here)
CRYPT_DISABLE (FUSE_SEC[5]) Permanently disables the decryptor. **CAUTION!** *CRYPT_DISABLE and CFG_AES_Only must NOT both be programmed to '1'.*	Yes=program to '1' No=do not program ('0')	
FUSE_BKS_ENABLE (FUSE_SEC[6]) Enables obfuscated key when loading encrypted bitstreams.	Yes=program to '1' No=do not program ('0')	
Encrypted/authenticated verification bitstream Bitstream that is encrypted or authenticated to verify the device eFUSE settings. **CAUTION!** *Do NOT use your actual design. See Encrypted Verification Bitstream for bitstream design constraints.*	MCS file name	

AES Key and NKY File

Typically, the 256-bit AES key is defined and provided to the Vivado design tools for encrypting the bitstream. The `write_bitstream` command is run to generate the encrypted bitstream. The Vivado design tools save the AES key value in an NKY file. Keep the NKY file as an input to the device programmer for programming the AES key value into the FPGA eFUSE, and for using the same key to encrypt the bitstream in successive runs of the `write_bitstream` command, including a run of `write_bitstream` to encrypt the verification bitstream.

See "Generating an Encrypted Bitstream" in the *Vivado Design Suite Programming and Debugging User Guide* (UG908) [Ref 3] and *Using Encryption to Secure a 7 Series FPGA Bitstream* (XAPP1239) [Ref 4] for instructions. Ensure that the selected location of the encryption key is "eFUSE" in the Vivado design tools encrypted bitstream flow.

Example 7 series FPGA output NKY file:

```
Device xc7a200tfbg676;
Key 0 0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef;
Key StartCBC 9966d617cc42731291374e58239a823e;
Key HMAC cd552f296733ac7abec573bf72e1b2d91d8c5ee603857a62f743ab8229fcbb25;
```

Example UltraScale FPGA output NKY file:

```
Device xcku040;
EncryptKeySelect EFUSE;
KeyObfuscate 5d349b7f5bfdfd5c4f0cc06802577110cb597129988c7d70256c9bf9cd3e997c;
StartIvObfuscate 15215b01cae66ddeaba1a393122c1cfd;
RsaPublicKeyModulus
00:9e:2b:49:ae:aa:37:0e:92:82:41:b6:fb:d7:df:79:c5:12:f7:40:b2:ed:96:01:8f:77:88:ea:b4
:2d:c5:02:4b:1a:43:bc:31:0b:f3:34:9a:bb:13:10:3a:0d:5d:fb:fd:ff:e7:cb:7a:d7:0f:0d:5b:4
9:fa:d3:e9:2f:6e:a2:12:4c:17:07:d8:72:c6:3e:95:54:50:cc:20:e1:19:a9:8b:3b:32:eb:d4:e5:
51:f7:10:da:10:62:92:17:7b:3f:c9:05:0e:25:4a:d2:db:77:0e:e3:80:76:c3:f9:e2:7b:ad:65:a6
:04:a1:b0:e8:5f:0b:1b:7d:72:73:a8:f3:98:6f:1b:f0:e9:a7:f0:cf:6b:78:09:1b:c8:c0:e9:a1:5
4:f5:00:a1:9d:5d:95:ae:24:a2:bc:15:b9:03:f6:bd:f0:bc:25:cb:f6:0a:c1:6c:c4:aa:39:33:76:
58:c8:33:83:9a:61:15:cd:dd:09:2b:bd:c5:40:d7:bf:b4:45:02:41:c4:36:b9:84:a7:06:5c:a2:38
:9a:6a:db:07:40:4b:aa:2f:3b:62:dd:cb:1e:e5:5b:ba:46:4a:c7:1b:c1:c7:4b:ea:ae:23:a1:ea:5
1:c2:9d:8a:61:1e:9f:69:10:10:69:21:a6:21:bc:3e:07:e1:e1:2d:eb:c7:a9:7e:9f:3a:fb:23;
RsaPublicKeyDigest
B561A9330C2C7DD3BE1464892B0D5ACCB9C1B4EE282F076CEE0CBF81EA9BC2700CCF1F688B8447BE71E01F
2790307DDE;
Key0 0000000000000000000000000000000000000000000000000000000000000000;
StartIV0 0e2004dbaa739e8320f43d1f003d04ca;
```

Example UltraScale SSI FPGA output NKY file with different AES keys for each SLR:

```
Device xcvu190;
EncryptKeySelect EFUSE;
RsaPublicKeyModulus
00:9e:2b:49:ae:aa:37:0e:92:82:41:b6:fb:d7:df:79:c5:12:f7:40:b2:ed:96:01:8f:77:88:ea:b4
:2d:c5:02:4b:1a:43:bc:31:0b:f3:34:9a:bb:13:10:3a:0d:5d:fb:fd:ff:e7:cb:7a:d7:0f:0d:5b:4
9:fa:d3:e9:2f:6e:a2:12:4c:17:07:d8:72:c6:3e:95:54:50:cc:20:e1:19:a9:8b:3b:32:eb:d4:e5:
51:f7:10:da:10:62:92:17:7b:3f:c9:05:0e:25:4a:d2:db:77:0e:e3:80:76:c3:f9:e2:7b:ad:65:a6
:04:a1:b0:e8:5f:0b:1b:7d:72:73:a8:f3:98:6f:1b:f0:e9:a7:f0:cf:6b:78:09:1b:c8:c0:e9:a1:5
4:f5:00:a1:9d:5d:95:ae:24:a2:bc:15:b9:03:f6:bd:f0:bc:25:cb:f6:0a:c1:6c:c4:aa:39:33:76:
58:c8:33:83:9a:61:15:cd:dd:09:2b:bd:c5:40:d7:bf:b4:45:02:41:c4:36:b9:84:a7:06:5c:a2:38
:9a:6a:db:07:40:4b:aa:2f:3b:62:dd:cb:1e:e5:5b:ba:46:4a:c7:1b:c1:c7:4b:ea:ae:23:a1:ea:5
1:c2:9d:8a:61:1e:9f:69:10:10:69:21:a6:21:bc:3e:07:e1:e1:2d:eb:c7:a9:7e:9f:3a:fb:23;
RsaPublicKeyDigest
B561A9330C2C7DD3BE1464892B0D5ACCB9C1B4EE282F076CEE0CBF81EA9BC2700CCF1F688B8447BE71E01F
2790307DDE;
KeyObfuscate 5ce2977020c19f7c8b48bf8ac612d52bf501a88ef56db18db688da6bfb1ce5c9, 0;
StartIvObfuscate f0704cc2a6f3962366f9df51abfce84f, 0;
Key0 0000000000000000000000000000000000000000000000000000000000000000, 0;
StartIV0 e8811c308746a8e8908acc6c005faa3e, 0;
KeyObfuscate 09c0ecfc113a25ef838a9f4f99de19d095978ff651d8078f5d67a0495823746d, 1;
StartIvObfuscate 221b761a307f1c7230b3add60568f0d1, 1;
Key0 0000000000000000000000000000000000000000000000000000000000000000, 1;
StartIV0 7ed58adb564dd284cd3df358005faa3e, 1;
KeyObfuscate a206241f992d9c73a943a98ad93d8f8453b61edeff7919312759f99b84f8e48e, 2;
StartIvObfuscate 7a1780d0318beb27030f7816c8fa36b7, 2;
Key0 0000000000000000000000000000000000000000000000000000000000000000, 2;
StartIV0 df5f5a40008b13907df2a416005faa3e, 2;
```

The write_bitstream output NKY file might not include all values shown in these examples. The values in the output NKY file depend on the bitstream properties. When AES encryption is used, forward the secret 7 series FPGA Key0 or UltraScale FPGA Key0 values for programming into the device eFUSE, unless the UltraScale architecture obfuscate key feature is used. When using the UltraScale architecture FPGA obfuscated key feature, forward the KeyObfuscate values for programming into the device eFUSE with the FUSE_SEC[6] (enable obfuscated key) setting, instead of the secret Key0 values. When using the UltraScale architecture RSA authentication feature, forward the RsaPublicKeyDigest value for programming into the device eFUSE. The StartCBC and HMAC values are used during bitstream generation and are not programmed into the eFUSE.

User-Defined (FUSE_USER/EFUSE_USR) 32-bit Value

A value can optionally be specified as input to the device programmer for programming the 32-bit user-defined value field in the eFUSE.

The 32-bit value can be accessed from:

- JTAG FUSE_USER instruction (if the R_EN_B_User [read-security] eFUSE bit is not programmed)
- FPGA design EFUSE_USR primitive

The value is typically specified to the device programmer as an 8-character hexadecimal value[31:0].

Note: Due to the physical assignment of the user-defined value bits in the 7 series FPGA eFUSE array, some programmer solutions, such as the Vivado hardware manager, divide the user value into the user value[31:8] and user value[7:0] segments. A programmer operator should be able to make this division from the given complete value[31:0].

User-Defined (FUSE_USER_128) 128-bit Value

For UltraScale FPGAs only, a value can optionally be specified as input to the device programmer for programming the 128-bit user-defined value field in the eFUSE. The 128-bit value can be accessed only via the JTAG interface via the JTAG FUSE_USER_128 instruction. The value is typically specified to the device programmer as a 64-character hexadecimal value[127:0].

eFUSE Control (FUSE_CNTL) and Security (FUSE_SEC) Bits

The eFUSE control bits are located in the FUSE_CNTL eFUSE register, and for UltraScale FPGAs in the FUSE_SEC eFUSE register. Each bit permanently enables a specific security setting. In general, the selected eFUSE control/security settings are dependent on the requirements for your application. See Table 1 or Table 2 for descriptions of each control/security setting. The typical application is one that secures the bitstream contents and its use only to FPGAs that have been programmed with the matching AES key or the RSA public key hash value.

Additional security for application requirements beyond the typical application can be achieved through selection of additional eFUSE control bits to program. See the caution statements in Table 1 and Table 2 for how the eFUSE control bits can affect RMA analysis.

Encrypted Verification Bitstream

A specially designed and encrypted/authenticated bitstream can be provided to the device programmer for the verification procedures described in the Using the Device Programmer for eFUSE Programming.

This section provides the reference design and instructions for building an encrypted verification bitstream for use on a device programmer.

CAUTION! *Due to the device programmer socket pinout and limitations, do NOT use your actual design bitstream for verification on a device programmer.*

Reference Design

Due to constraints imposed by the device programmer, which are described in Encrypted Verification Bitstream Design Modification and Limitations, the reference design is a mostly empty design.

The reference design files for this application note can be downloaded from:

https://secure.xilinx.com/webreg/clickthrough.do?cid=393635

Table 3 shows the reference design matrix.

Table 3: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Randal Kuramoto
Target devices	7 series and UltraScale FPGAs
Source code provided	Yes
Source code format	VHDL/Verilog
Design uses code and IP from existing Xilinx application note and reference designs or third party	No
Implementation	
Synthesis software tools/versions used	Vivado Design Suite 2017.3
Implementation software tools/versions used	Vivado Design Suite 2017.3
Static timing analysis performed	N/A
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	AC701 and KCU105 evaluation kits

Building the Encrypted Verification Bitstream Reference Design

After downloading the reference design files, use these instructions to build your encrypted verification bitstream:

1. Unzip the reference design directory tree and files to your computer.
2. If using encryption, copy the NKY file containing your AES secret key to the `verilog` or `vhdl` directory within the reference design directory tree. If using RSA authentication for an UltraScale FPGA, copy the RSA secret key (`.pem`) file to the `verilog` or `vhdl` directory within the reference design directory tree.
3. Edit the `XAPP1260_make_encrypted_verification_bitstream.tcl` file within the `verilog` or `vhdl` directory, and follow the usage instructions within the Tcl script file to customize the script for your device and NKY file. The following is an example of the Tcl script usage instructions for an UltraScale device.

```
#Usage: Perform the following steps 1-7:
# ---START HERE-----------------------------------------------------
# 1. Replace xcku040-fbva676-1-c on the following line with your part type.
set part xcku040-fbva676-1-c

# ---AES ENCRYPTION SETTINGS----------------------------------------
# 2. If you want to verify the eFUSE AES key, then set encrypt to YES
# AND replace XAPP1260_example.nky on the following line with your NKY file name
# that contains your eFUSE AES secret key,
# and ensure that a copy of your NKY file is in the same directory as this script.
# Otherwise, if encrypt is NO, then no NKY file is needed.
set encrypt YES
set nkyFileName XAPP1260_example.nky
# -----FOR OBFUSCATED AES KEY ONLY-----
# If you use an eFUSE obfuscated AES key, then set obfuscate to ENABLE (or DISABLE if not),
# and only for obfuscated keys, you must set the obfuscateFamilyKeyPath setting for the
BITSTREAM.ENCRYPTION.FAMILY_KEY_FILEPATH property.
# Contact Xilinx Support for the BITSTREAM.ENCRYPTION.FAMILY_KEY_FILEPATH setting.
# -----FOR OBFUSCATED AES KEY ONLY-----
set obfuscate DISABLE
set obfuscateFamilyKeyPath ""

# ---RSA AUTHENTICATION SETTINGS------------------------------------
# 3. If you want to verify the eFUSE RSA authentication hash, then set authenticate to YES
# AND replace the rsa.pem on the following line with your RSA .pem file name
# that contains your RSA secret key
# and ensure that a copy of your .pem file is in the same directory as this script.
# Otherwise, if authenticate is NO, then no .pem file is needed.
set authenticate NO
set pemFileName XAPP1260_example.pem

# ---IDENTIFICATION SETTING-----------------------------------------
# 4. Optionally set the user ID (JTAG USERCODE) for the verification bitstream
# Value must be an 8-digit hexadecimal number.
set userIdVerificationBitstream 00000001

# ---FPGA I/O SETTING-----------------------------------------------
# 5. If you know that your FPGA PUDC_B (pull-up during configuration [bar]) pin
# on your application board is tied Low, then change this to PULLUP (from PULLNONE).
# Otherwise, leave the default PULLNONE.
# This only matters if you download the .bit file to the FPGA on your application board
# to test that the verification bitstream decrypts correctly in an FPGA on your board
```

```
# AND if your board depends on internal FPGA I/O pull-ups when the FPGA is NOT configured.
# This setting is intended to result in an FPGA bitstream that results in an FPGA
# that acts like an unconfigured FPGA.
# alue must be PULLNONE or PULLUP.
set pudc PULLNONE

# ---BITSTREAM GENERATION INSTRUCTIONS----------------------------
# 6. Open a terminal or command prompt and change (cd) to the directory containing this script file.
# 7. Start the Vivado design tools
# 8. source this script file into the Vivado design tools:
# source XAPP1260_make_encrypted_verification_bitstream.tcl
# NOTE: View the vivado.log file to confirm details of bitstream properties.
# ---DONE--------------------------------------------------------
```

4. Optionally, modify the `XAPP1260_encrypted_verification_bitstream.v[hd]` file to verify the 32-bit EFUSE_USR value.
5. Start the Vivado design tools, change (cd) to the `verilog` or `vhdl` directory, and source the `XAPP1260_make_encrypted_verification_bitstream.tcl` file or use this command line:

   ```
   vivado -mode batch -source XAPP1260_make_encrypted_verification_bitstream.tcl
   ```

6. Verify that the summary of write_bitstream options in the log includes the settings shown in Encrypted Verification Bitstream Checklist.

The Tcl script builds the encrypted verification bitstream and formats the bitstream in a MCS file. The MCS file can be used on the device programmer. The resulting file is named:

```
XAPP1260_<part>_<options>_verification_bitstream.mcs
```

where <part> is the part name specified in step 3.a. in Building the Encrypted Verification Bitstream Reference Design.

For design validation or debug purposes, the Tcl script also creates an unencrypted version of the bitstream for checking that the design works in the FPGA without encryption. The unencrypted bitstream file name is:

```
XAPP1260_<part>_unencrypted_bitstream.bit
```

where <part> is the part name specified in step 3.a. in Building the Encrypted Verification Bitstream Reference Design and <options> is the descriptive text of the selected options.

Note: The `XAPP1260_example.nky` file contains an all zero key value that matches the key of an FPGA with unprogrammed eFUSE. An encrypted test bitstream built using the `XAPP1260_example.nky` file can be correctly decrypted and used to successfully configure an FPGA whose eFUSE have not been programmed. This is true because the default all zero eFUSE key matches the all zero encryption key. This method can be used to validate the Vivado design tools encryption flow without having to program the eFUSE in an FPGA.

In Case of a Build Error

Check the `vivado.log` file from the Vivado design tools run of the Tcl script. The Tcl script can report an error when an issue with the part name or NKY file is encountered. If this error occurs, check the log files in the project subdirectories for an indication of an error:

```
ERROR: [Common 17-69] Command failed: Run 'impl_1' has not been launched.
```

Specifically, check the `XAPP1260_encrypted_verification_bitstream.runs\synth_1\runme.log` file for errors during synthesis.

Encrypted Verification Bitstream Design Modification and Limitations

The reference design can be modified to alter the encrypted contents from the base reference design as needed to further randomize the encrypted design. However, the resulting FPGA design must still comply with these limitations:

- A verification bitstream must not consume more than the minimum power-on current. Typically, a device programmer is power supply limited. The device programmer only has sufficient power to power on the device and program the eFUSE.
- Do not use any user-configurable I/O. A device programmer has fixed pin connections. The device programmer only properly connects to the dedicated JTAG interface pins and might control the SelectMAP configuration interface pins. A verification design should not interfere with the SelectMAP pins. Otherwise, the connection or power for all user-configurable pins is undefined.
- Do not instantiate or enable the transceivers. The device programmer socket is designed with the assumption that the transceivers are not used or enabled.
- Do not use the UltraScale FPGA key rolling feature (i.e., the KEYLIFE or RSAKEYLIFEFRAMES properties must be 0). Otherwise, the key rolling feature can expand the bitstream beyond the space allocated on a device programmer for the verification bitstream.

Encrypted Verification Bitstream Checklist

If using encryption, these properties must be accurately specified for the encrypted verification bitstream:

```
BITSTREAM.ENCRYPTION.ENCRYPT = YES
BITSTREAM.ENCRYPTION.ENCRYPTKEYSELECT = EFUSE
BITSTREAM.ENCRYPTION.KEYFILE = <NKY file name that contains your AES keys>
```

These UltraScale FPGA properties can optionally be specified with the above encryption settings to enable the obfuscated key option. If enabled, both properties must be specified:

```
BITSTREAM.ENCRYPTION.OBFUSCATEKEY = ENABLE
BITSTREAM.ENCRYPTION.FAMILY_KEY_FILEPATH = <family_key_filepath>
```

Contact Xilinx support for further information about the obfuscated key feature and `<family_key_filepath>`.

This property is recommended to minimize the verification time on the device programmer, except compression is not supported with the UltraScale FPGA RSA authentication feature:

```
BITSTREAM.GENERAL.COMPRESS = TRUE
```

If using UltraScale FPGA RSA authentication, these properties must be specified for authenticating the verification bitstream:

```
BITSTREAM.AUTHENTICATION.AUTHENTICATE = YES
BITSTREAM.AUTHENTICATION.RSAPRIVATEKEYFILE= <PEM file name that contains your RSA secret key>
```

The settings for the properties listed can be verified by running the `write_bitstream` command with the `-verbose` option. The reference design Tcl script includes the `-verbose` option with its `write_bitstream` command. When `write_bitstream` is run with the `-verbose` option, a summary of the property settings when writing the bitstream is printed to the log. An extracted subset of the important fields within the summary is shown here:

```
Summary of write_bitstream Options:
+-------------------------+-------------------------+
| Option Name             | Current Setting         |
+-------------------------+-------------------------+
...
+-------------------------+-------------------------+
| KEYFILE                 | your_key_file_name.nky  |
+-------------------------+-------------------------+
...
+-------------------------+-------------------------+
| ENCRYPTKEYSELECT        | EFUSE                   |
+-------------------------+-------------------------+
...
+-------------------------+-------------------------+
| ENCRYPT                 | YES                     |
+-------------------------+-------------------------+
...
+-------------------------+-------------------------+
| COMPRESS                | TRUE                    |
+-------------------------+-------------------------+
...
+-------------------------+-------------------------+
```

```
| OBFUSCATEKEY             | ENABLE                   |
+--------------------------+--------------------------+
| KEYLIFE                  | 0**                      |
+--------------------------+--------------------------+
| RSAKEYLIFEFRAMES         | 0**                      |
+--------------------------+--------------------------+
| AUTHENTICATE             | YES                      |
+--------------------------+--------------------------+
| RSAPRIVATEKEYFILE        | XAPP1260_example.pem     |
+--------------------------+--------------------------+
```

Actual current settings can vary depending on your chosen write_bitstream properties.

The data beyond the bitstream header (i.e., beyond the first approximately 12 lines) in the resulting encrypted verification bitstream MCS file can be visually inspected for encryption. The unencrypted bitstream data has mostly zero bit values (because most of the resources in a given FPGA design are not used). The encrypted bitstream should appear to be more like a random data pattern. See Figure 2 for a comparison of an unencrypted versus encrypted bitstream data pattern. Bitstream data is highlighted in the yellow boxes.

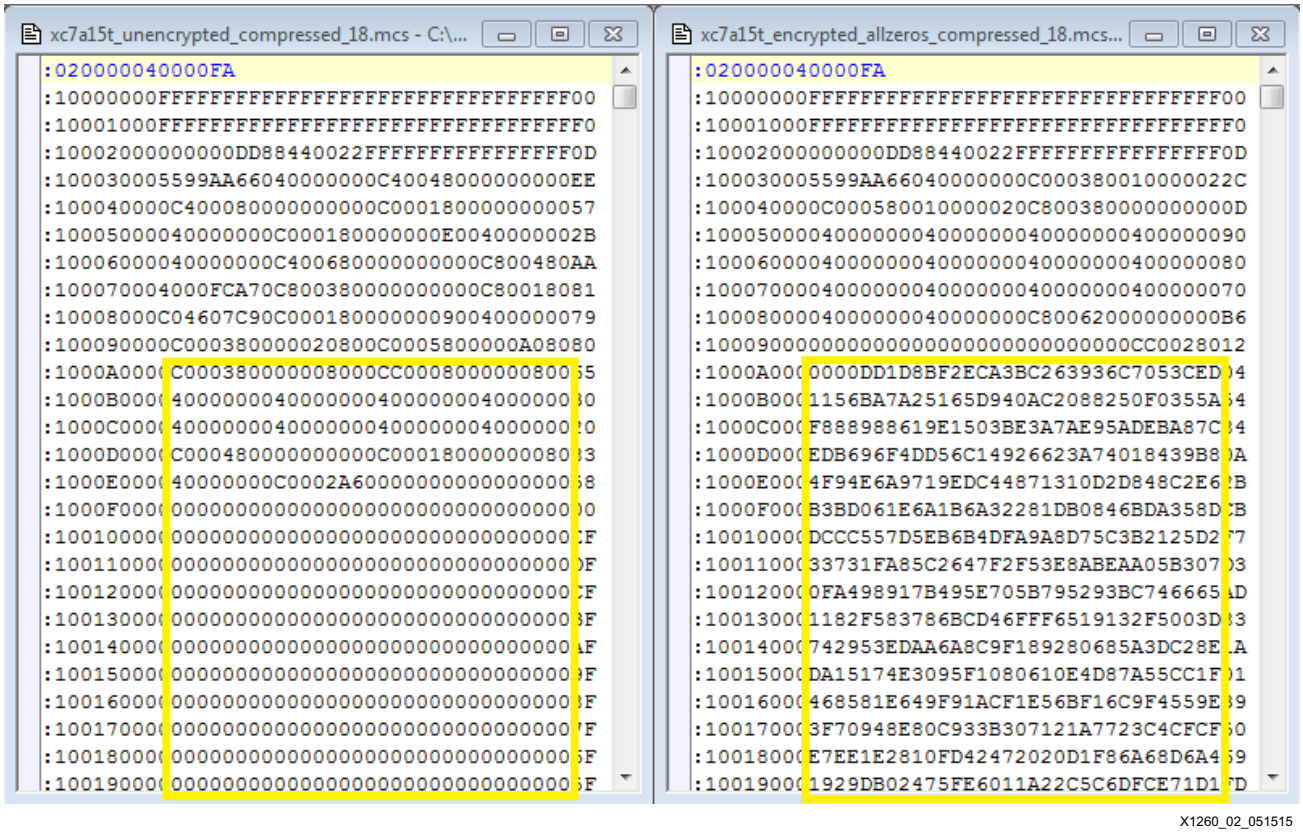

Figure 2: **Unencrypted (On Left) Versus Encrypted (On Right) Bitstreams**

Guidelines for Validating Device Programmer Setup and Operation

The previous sections provide instructions for defining the device programmer settings and building the device programmer input files.

This section provides guidelines for:

- Checking the intended device programmer settings and input files before they are sent to the device programmer
- Checking the device programmer settings and input files
- Checking that the first device programmer programmed device works for the eFUSE settings on an assembled board

Use these guidelines as a checklist to ensure successful FPGA eFUSE programming on a device programmer.

Precheck for Device Programmer Files

The guidelines in this section are for checking the intended device programmer settings and input files before they are sent to the device programmer.

1. Validate that the base reference design works for your device without encryption on a prototype board:

   Use the Vivado hardware manager to download the `XAPP1260_<part>_unencrypted_bitstream.bit` file to the FPGA on your prototype board that has not been programmed with an AES key in its eFUSE. The FPGA should successfully be configured.

   Note: If you modified the `XAPP1260_encrypted_verification_bitstream.v[hd]` file to verify the 32-bit EFUSE_USR value is a non-zero (not blank) value, this test can fail.

2. Validate the security of an encrypted bitstream on a prototype board:

   Use the Vivado hardware manager to download the encrypted verification bitstream to the FPGA on your prototype board that has not been programmed with an AES key in its eFUSE. The FPGA should fail the configuration attempt.

3. Check that the encrypted verification bitstream and actual encrypted design bitstream work in the device on your prototype board:

   a. Use the Vivado hardware manager to program the eFUSE with the settings from your completed Table 1 or Table 2. See *Using Encryption to Secure a 7 Series FPGA Bitstream* (XAPP1239) [Ref 4] or *Using Encryption and Authentication to Secure an UltraScale/UltraScale+ FPGA Bitstream* (XAPP1267) [Ref 5] for instructions on using the Vivado tools to access the device eFUSE.

- Use the Vivado hardware manager to view the device **Properties** -> **Register** -> **eFUSE** -> **FUSE_CNTL** value. Save a copy of the FUSE_CNTL value that results from the Vivado hardware manager programming of the eFUSE. This can be used later to verify that the device programmer results in the same FUSE_CNTL value. Also check the FUSE_USER, FUSE_USER_128 (UltraScale FPGAs only), and FUSE_SEC (UltraScale FPGAs only) values.

Note: When these eFUSE registers are read secured, all bits are '1'.

b. Use the Vivado hardware manager to download the encrypted/authenticated verification bitstream to the device on your prototype board. The FPGA should successfully be configured.

Note: See *UltraScale Architecture Configuration User Guide (*UG570) [Ref 6] for UltraScale FPGAs and configuration modes that support RSA authenticated bitstream download. Some UltraScale devices do not support authenticated bitstream download via JTAG mode.

c. Use the Vivado hardware manager to download the actual encrypted/authenticated design bitstream to the device on your prototype board. The FPGA should successfully be configured.

Device Programmer Setup Verification

The guidelines in this section are for checking the device programmer settings and input files.

4. Check that the device programmer standalone verify operation reports an error when the device is NOT programmed with the matching key:
   a. Put a factory new device in the device programmer socket.
   b. Run the standalone verify operation with the encrypted verification bitstream.
   c. The device programmer should report an error.

      If the standalone verify passes without error, then:

      - Recheck that the device is blank.
      - Recheck the encrypted verification bitstream. Use the Vivado hardware manager to configure a device on a board, which does not have the eFUSE programmed, with the encrypted verification bitstream. The Vivado hardware manager should report a configuration error.
5. Program and verify one or two test devices with the settings, and send the programmed test devices for testing on an assembled board.

   If the verify operation fails, then recheck the encrypted verification bitstream using the guidelines from the Precheck for Device Programmer Files section.

First Programmed Device Verification on Assembled Board

The guidelines in this section are for checking that the first device programmer programmed device works correctly for the eFUSE settings on an assembled board.

6. Check the FUSE_CNTL register for read-security of the AES key and other security settings:
   a. Use the Vivado hardware manager to connect to the device on a board through JTAG.
   b. View the device **Properties** -> **Register** -> **eFUSE** -> **FUSE_CNTL** value. If you saved the FUSE_CNTL value that resulted from the Vivado hardware manager programming of the eFUSE in a prototype board, then compare the FUSE_CNTL values from the Vivado hardware manager programming to the device programmer programmed device. Also check the FUSE_USER, FUSE_USER_128 (UltraScale FPGAs only), and FUSE_SEC (UltraScale FPGAs only) values.

      Note: When these eFUSE registers are read secured, all bits are '1'.

   c. Check that at least the 7 series FPGA R_EN_B_Key (FUSE_CNTL[3]) bit or UltraScale FPGA R_DIS_KEY (FUSE_CNTL[0]) bit is programmed to a "1".
      - If the bit that disables reading of the key is not programmed to "1", recheck all device programmer settings.
7. Load your encrypted bitstream into the FPGA.
   a. If configuration fails, recheck that your design and the device programmer settings used the exact same NKY file (AES key) to encrypt the bitstream.

References

1. *7 Series FPGAs Configuration User Guide* (UG470)
2. *UltraScale Architecture Configuration Guide* (UG570)
3. *Vivado Design Suite Programming and Debugging User Guide* (UG908)
4. *Using Encryption to Secure a 7 Series FPGA Bitstream* (XAPP1239)
5. *Using Encryption and Authentication to Secure an UltraScale/UltraScale+ FPGA Bitstream* (XAPP1267)
6. *UltraScale Architecture Configuration User Guide* (UG570)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/21/2017	1.1	Added UltraScale FPGA information.
11/15/2015	1.0	Initial Xilinx release.

Please Read: Important Legal Notices