XAPP1084 (v1.4) June 13, 2017

Developing Tamper Resistant Designs with Xilinx Virtex-6 and 7 Series FPGAs

Author: Ed Peterson

Summary

This application note provides anti-tamper (AT) guidance and practical examples to help the FPGA designer protect the intellectual property (IP) and sensitive data that might exist in an FPGA-enabled system. This protection (in the form of tamper resistance) needs to be effective before, during, and after the FPGA has been configured by a bitstream. Sensitive data can include the configuration data that sets up the functionality of the FPGA logic, critical data and/or parameters that might be included in the bitstream (e.g., initial block RAM contents, initial state of flip-flops, etc.), and external data that is dynamically brought in and out of the FPGA during post-configuration normal operation.

This document summarizes the silicon AT features available in the Virtex®-6 family, 7 series (Spartan®-7[1], Artix®-7, Kintex®-7, and Virtex-7) FPGAs, and the Zynq®-7000 All Programmable SoCs, explains why these features exist, and provides use cases and implementation details for each feature. This document also provides guidance on various methods that can be employed to provide additional tamper resistance.

With this application note, engineers can ensure that they are following AT best practices to provide the highest level of protection of their FPGA designs. These best practices broadly apply whether the reason for requiring tamper resistance is to prevent cloning/overbuilding of a commercial design or to protect valuable military critical technology (CT) from reverse engineering efforts by an adversary.

It is assumed that the reader is somewhat knowledgeable and proficient in Xilinx FPGA architecture [Ref 1] and design, as well as in using the ISE® tools flow [Ref 2] methodology. *Solving Today's Design Security Concerns* [Ref 3] provides a good background on the various security threats and solutions for FPGAs.

Introduction

Xilinx has been at the forefront of providing FPGA AT solutions to their customers for many years. The Virtex-II device was the first FPGA with bitstream encryption. Virtex-6 and 7 series FPGAs extend these solutions to include bitstream encryption and authentication, as well as other silicon AT features. Xilinx also offers a soft IP core, Security Monitor, that provides certain tamper protections after configuration [Ref 4]. Due to certain restrictions, Security Monitor's availability is limited. Contact your local Xilinx FAE for details.

Keeping one step ahead of the adversary is a continuous process that involves understanding the potential vulnerabilities and attacks and then developing new mitigation techniques (countermeasures) to combat those attacks. Xilinx has a multi-generational commitment to secure FPGA technology for the AT-conscious community.

By taking advantage of various Xilinx FPGA AT features, a systems engineer can choose how much AT to include with the FPGA design. AT can be in the form of enabling individual silicon AT features or a combination of these AT features (perhaps tied together by the developer in the FPGA design and following certain guidance).

1. This application note does not apply to the smallest Spartan-7 devices: XC7S6 and XC7S15.

The decision as to how much AT to include primarily depends on three factors:

- Value: The perceived value of the intellectual property and the damage it might cause either financially and/or to national security if it were to become compromised. Certain AT features can be expensive to implement and that cost must be weighed against the value of the technology being protected.
- Adversary: Access to the system and the sophistication level/resources available to carry out the attack. For example, will access to the system be prevented by “guns, gates, and guards” or will it be easily obtained in the open market? Is the adversary a garage-based hacker or a nation-state? The adversary's capabilities could be at these extremes or anywhere in-between.
- Design Stage: At what point in the system development cycle is the decision made to enable AT for the FPGA design? Xilinx highly recommends that the decision to utilize FPGA AT features is made very early on (i.e., after CT is defined in a system) to help address both schedule and cost concerns. It is always more costly and time consuming to insert AT features later on.

Another factor that needs to be considered is the extra FPGA logic resources that might be consumed by enabling certain AT features. The overall resource penalty is usually rather minimal. However, it does depend on how these features are implemented and the size of the FPGA device (i.e., the larger devices experience less of an impact).

Xilinx classifies the built-in silicon AT features as either *passive* security or *active* security. Passive security features are those that are built into the FPGA and do not require the user to do anything extra in their FPGA logic design. Additionally, passive security features are temporal in nature—they come into effect at different times during the normal life cycle of the FPGA:

- Pre-configuration (e.g., decryption key protection in battery-backed RAM (BBRAM))
- During-configuration (e.g., user design protection via bitstream authentication and decryption)
- Post-configuration (e.g., user data protection via disabling of readback)

Active security features require the user design to do something in the FPGA design. These features only come into effect after the FPGA has been configured via the user bitstream and the user design becomes active. Examples are asserting KEYCLEARB to erase the battery-backed AES key or handle a PROGRAM_B intercept.

At a bare minimum, the system designer should always plan on including the appropriate passive security features into their design (e.g., bitstream encryption and authentication). These features do not affect the function of the user design. However, they might create logistical challenges (e.g., key management), system challenges (e.g., might need a battery if using BBRAM for key storage), and increase the configuration time (encrypted configuration is limited to x8 data width). Otherwise, these features come for *free* and can provide a fair amount of tamper protection. For an already fielded system or a design late in the development stage, these AT features are great candidates for enabling (if they are not already).

Additionally, the AT features and guidance presented in this document fall into three main AT categories:

1. Prevention (e.g., bitstream encryption/authentication)
2. Detection (e.g., voltage and temperature monitoring)
3. Response (e.g., bitstream BBRAM decryption key erasure penalty)

Table 1 summarizes and classifies the built-in silicon AT features of the Virtex-6 and 7 series FPGAs:

Table 1: **AT Features Classification and Summary**

Virtex-6, 7 Series, and Zynq Devices Silicon AT Features	Type	Category
Volatile AES-256 BBRAM Key Storage	Passive(1)	Prevention
Non-volatile AES-256 eFUSE Key Storage	Passive(1)	Prevention
256-bit AES Bitstream Decryption	Passive(2)	Prevention
HMAC SHA-256 Bitstream Authentication	Passive(2)	Prevention
RSA Asymmetric Bitstream Authentication (Zynq devices only)	Passive(1)(2)	Prevention
Hardened Readback Disabling Circuitry	Passive(3)	Prevention
Robust Key Load Finite State Machine (FSM) Circuitry	Passive(3)	Prevention
JTAG Disable	Active	Prevention
JTAG Monitor	Active	Detection
Internal Configuration Memory Integrity	Active	Detection
On-chip Temperature and Voltage Monitor/Alarms (SYSMON/XADC)	Active	Detection
PROG Intercept (PREQ/PACK)	Active	Detection
Unique Identifiers (Device DNA and User eFUSE)	Active	Detection
Internal Configuration Memory Clear (IPROG)	Active	Response
Internal AES-256 BBRAM Key Erase (KEYCLEARB)	Active	Response
Global 3-State (GTS)	Active	Response
Global Set-Reset (GSR)	Active	Response

Notes:
1. Pre-configuration.
2. During configuration.
3. Post-configuration.

The following sections explore the above features in depth, providing detailed explanations on what they are, why they exist, and give specific examples on how to properly use them (either by themselves, in conjunction with other built-in features and user logic, or both). Additionally, specific guidance is given on certain methods and techniques that can be employed to increase the tamper resistance of the FPGA design and overall system.

Any AT features enabled at the FPGA level should always be part of an overall system-level AT solution. The features and techniques outlined in this document provide for a very good AT umbrella for the FPGA itself. However, AT is most effective when it is part of a multi-layer approach developed with the entire system in mind.

Passive AT Silicon Features

Bitstream Encryption/Decryption

Storing an encrypted[1] bitstream in external flash (or other means) and then decrypting it dynamically during FPGA configuration (inside the FPGA's decryption engine) provides for a high level of confidentiality. This ensures that information contained in the bitstream is only accessible to those who share the same secret key. Bitstream encryption provides

1. Xilinx uses the Advanced Encryption Standard (AES) in Cipher Block Chaining (CBC) mode with a 256-bit key.

confidentiality while the system is off and during configuration. It protects the FPGA design contents, including block RAM and flip-flop initialization data. Xilinx highly recommends that the externally stored bitstream is always in encrypted form.

To take advantage of this security feature, the configuration bitstream must first be encrypted by the BitGen software [Ref 2] that uses a key supplied by the user to perform the encryption. If an AES key is not supplied, BitGen generates one automatically. However, the user should keep in mind that BitGen generated keys are only pseudorandom and are not as strong as truly random keys. This same key is then loaded into the FPGA via the JTAG port using the iMPACT software [Ref 2]. The key can be loaded into either volatile BBRAM or non-volatile eFUSE storage locations within the FPGA. To decide which storage location to use for the key, the system designer must understand the advantages and disadvantages of BBRAM (Table 2) and eFUSE (Table 3).

Table 2: **BBRAM Storage Location Advantages and Disadvantages**

Advantages	Disadvantages
• Volatile and reprogrammable. • Passive and active key clearing (i.e., the evidence can be removed). • Tamper resistant.(1)	• Requires an external battery. • Many battery vendors do not specify operation at high temperature and/or long lifetimes (although some vendors are now starting to offer *betavoltaic* type batteries to help address these issues).

Notes:

1. Any attempted read or write access to the BBRAM via JTAG causes the BBRAM contents to be cleared and the entire configuration of the FPGA to be erased prior to access being enabled (i.e., being able to enter “key access mode”).

Table 3: **eFUSE Storage Location Advantages and Disadvantages**

Advantages	Disadvantages
• No external battery required. • Makes spoofing impossible. • Only a bitstream encrypted with the eFUSE key can get loaded into the FPGA. All others are rejected if the `cfg_aes_only`(1)(2) eFUSE bit is also blown.	• Permanent; the key cannot be cleared. • Less secure than BBRAM solution (i.e., the evidence remains). • Xilinx does not *scramble* the eFUSE key because the only way to securely scramble data is to encrypt it using an approved algorithm and a secret key.

Notes:

When using the `cfg_aes_only` option (located in the eFUSE Control Register - FUSE_CNTL) there are two important points to consider:

1. If using the indirect flash programming method [Ref 5] for the bitstream, ensure that this option is enabled after the on-board flash has been loaded with the encrypted bitstream. This is because the indirect programming core bitstream from Xilinx is unencrypted. Otherwise, the FPGA will attempt to decrypt the indirect programming bitstream using the eFUSE key and it will fail configuration.
2. In addition to spoof protection, it is important to enable the `cfg_aes_only` option to prevent an adversary from attempting to recover portions of the bitstream encrypted with the eFUSE key. In this attack scenario, the user bitstream is encrypted with a key that is stored in eFUSE, the adversary's design is encrypted with the BBRAM key, and the `cfg_aes_only` option is not set.
   In this scenario, it might be possible for the adversary to read out portions of the user bitstream using partial reconfiguration via the internal configuration access port (ICAP). This attack does not work if the user bitstream is encrypted with the BBRAM key and the adversary’s design is encrypted with the eFUSE key.

Additional details concerning bitstream encryption and key storage can be found in the Virtex-6 and 7 series FPGAs configuration user guides [Ref 6], [Ref 7].

Bitstream Authentication

When a user enables bitstream encryption/decryption on Virtex-6 or 7 series FPGAs, authentication is automatically turned on (i.e., both are enabled/disabled in tandem). The authentication method used is the keyed-hashed message authentication code (HMAC) [Ref 8] using the SHA-256 hash algorithm (FIPS180-2) [Ref 9].

Ciphertext modification attacks are infeasible with this type of authentication. One such attack is the AES-CBC attack, summarized in Figure 1.

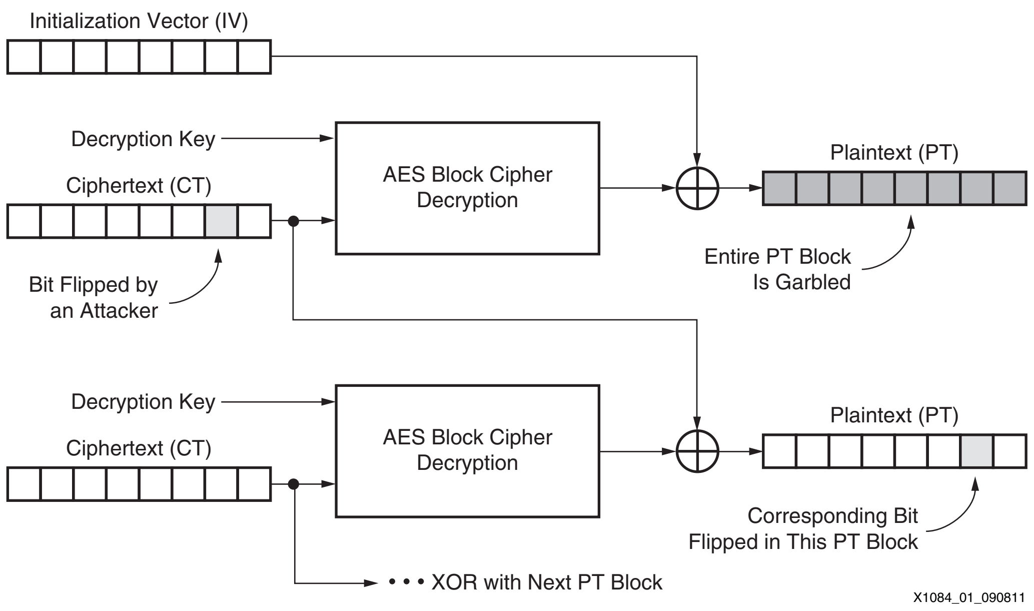
Initialization Vector (IV)
Decryption Key
Ciphertext (CT)
AES Block Cipher Decryption
Plaintext (PT)
Bit Flipped by an Attacker
Entire PT Block Is Garbled
Decryption Key
Ciphertext (CT)
AES Block Cipher Decryption
Plaintext (PT)
Corresponding Bit Flipped in This PT Block
• • • XOR with Next PT Block
X1084_01_090811

Figure 1: **AES-CBC Bit-Flip Attack**

CBC is a *non-error propagating* mode of AES and is the mode used in Virtex-6 and 7 series FPGAs (see note below). When a bit is flipped during transmission, it does not garble everything that comes afterwards. A bit flip garbles the current cipher block plus a corresponding bit in the next block (which is a desirable property when transmitting on a noisy channel). However, it is possible for an attacker to make a small deterministic change in the bitstream. This attack causes 128 other bits to be scrambled, thereby limiting its usefulness. The normal CRC32 integrity check on the entire bitstream also detects the attacker's change, unless it was disabled in the bitstream.

> ***Note:*** Xilinx did not use another encryption algorithm in light of this attack because at the time of the design of the FPGA devices, there were no other NIST-approved modes of AES (such as AES Galois/Counter Mode (GCM)). The authentication used on Virtex-6 and 7 series FPGAs prevents this type of attack.

Because this authentication method uses a key, an HMAC key must also be supplied to the BitGen software. (The HMAC key is contained in the same NKY key file as the AES encryption key. If an HMAC key is not supplied, BitGen generates one automatically.) However, this key is not loaded into the FPGA device directly via JTAG like the AES key. It is contained and protected (wrapped) by the encrypted bitstream. During the on-chip decryption process, this HMAC key is extracted from the bitstream and used by the authentication algorithm (i.e., no on-chip key storage for the HMAC key is required). Additional details concerning bitstream authentication can be found in the Virtex-6 and 7 series FPGAs configuration user guides [Ref 6], [Ref 7].

Figure 2 summarizes which portions of the bitstream are unencrypted (plain text), encrypted (ciphertext), and authenticated when the above passive security measures (bitstream encryption and authentication) are enabled.

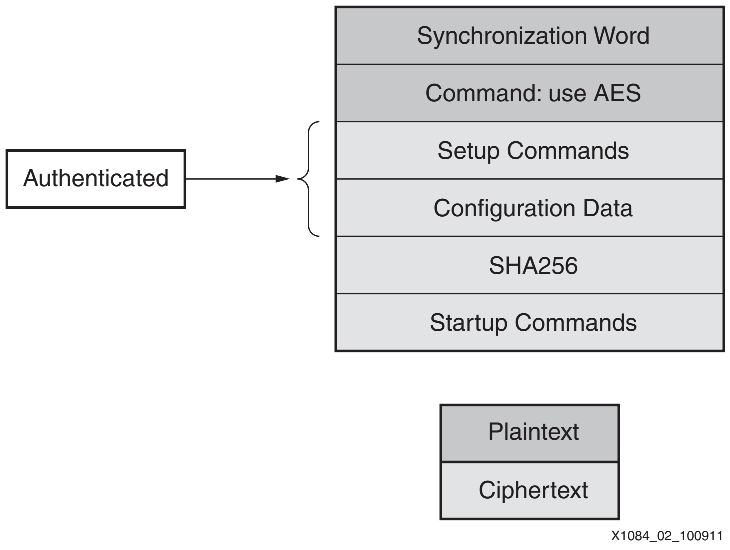

Figure 2: **Secure Bitstream Format**

If the authentication passes, the device then begins normal operation (i.e., the startup commands take place). Evidence of the authentication step failing is the absence of the DONE output signal asserting High after the bitstream is loaded, the INIT_B signal asserting Low, and the HMAC_ERROR_0 bit being set in the bootsts register (bit location = [7] HMAC_ERROR_0). This register can be read via JTAG using the iMPACT software and issuing the Read Device Status command. An authentication failure might indicate that the bitstream has been tampered with. It could also indicate that the channel used for bitstream loading is noisy and bit corruption(s) are taking place during the configuration process.

Figure 3 illustrates how the authentication codes are created (in software) and then verified (in hardware) before allowing the device to run.

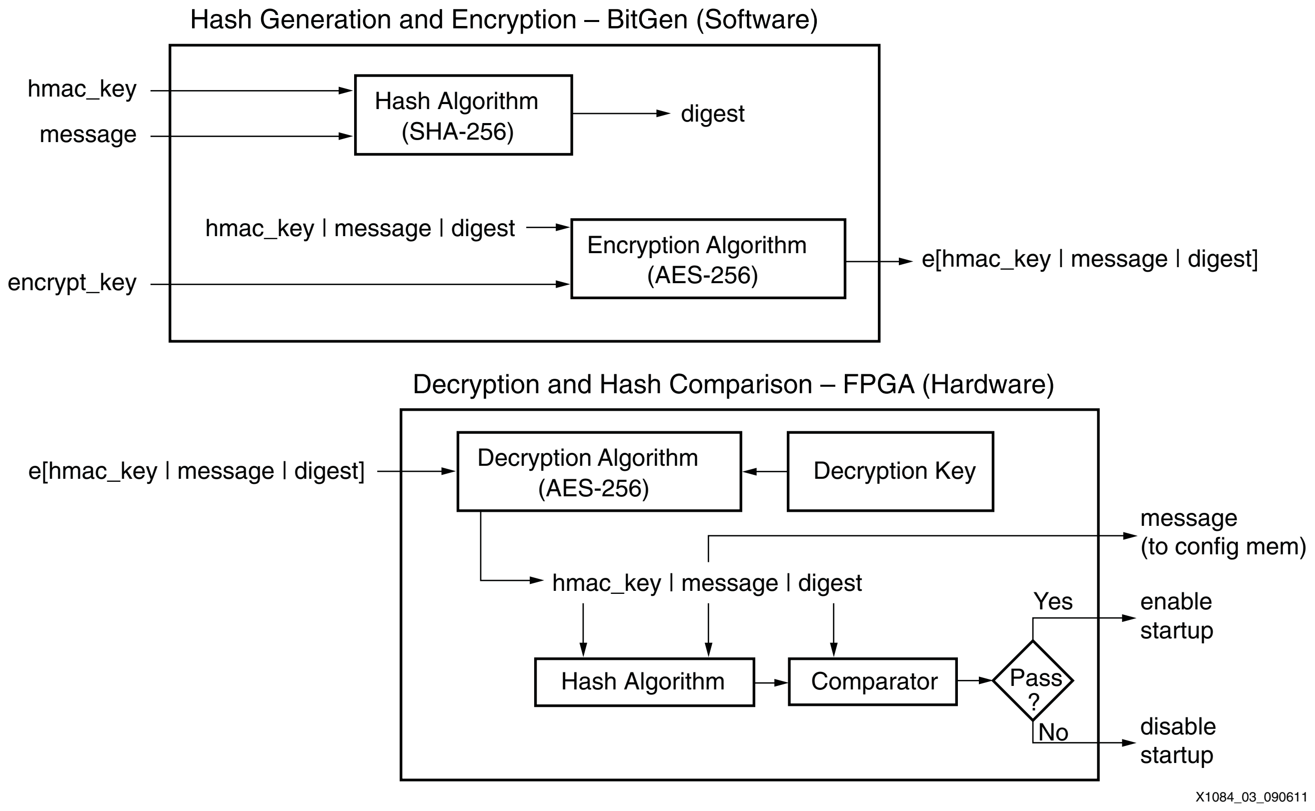
Hash Generation and Encryption – BitGen (Software)
hmac_key
message
Hash Algorithm (SHA-256)
digest
hmac_key | message | digest
encrypt_key
Encryption Algorithm (AES-256)
e[hmac_key | message | digest]
Decryption and Hash Comparison – FPGA (Hardware)
e[hmac_key | message | digest]
Decryption Algorithm (AES-256)
Decryption Key
message (to config mem)
hmac_key | message | digest
Yes
enable startup
Hash Algorithm
Comparator
Pass ?
No
disable startup
X1084_03_090611

Figure 3: **Hashed Message Authentication Operation**

RSA Asymmetric Bitstream Authentication (Zynq Devices Only)

The BootROM can authenticate an encrypted or unencrypted first stage boot loader (FSBL) prior to decryption (if the FSBL is encrypted) and execution using the RSA-2048 public key authentication protocol. This feature is enabled by triggering the RSA Authentication Enable fuse in the processor system (PS) eFUSE array. For additional details on this security feature, refer to *Secure Boot of Zynq-7000 All Programmable SoC* [Ref 10] or contact your local Xilinx FAE.

Hardened Readback Disabling Circuitry

Whenever an encrypted bitstream is loaded into the FPGA, readback of the internal configuration memory cannot be performed by any of the external interfaces (including JTAG). All external readback is automatically blocked (disabled) by hardened triple-redundant logic. The only readback access to the configuration memory after an encrypted bitstream load is via the ICAP. Since the bitstream was authenticated during the loading process, the ICAP is considered a trusted channel because it can only be used via a direct connection to the user's design within the FPGA logic. If the user design does not instantiate the ICAP, it cannot be used at all.

Note: A BitGen security option via a particular control bit in the bitstream provides a means of enabling/disabling readback. This bit can be changed during configuration. Therefore, readback disable is easy to defeat for devices that are not using an encrypted/authenticated bitstream. Hardened readback disabling has no such weakness and always overrides the security option when using an encrypted/authenticated bitstream.

Robust FSM Circuitry

The FSMs for the key loading circuitry employ large hamming distances between states to help guard against a glitch attack that might try to circumvent the normal state machine operation and place the FPGA in an unsecure state.

Active AT Silicon Features

As mentioned in the Introduction, the active AT features require the user to do something in the design to take advantage of the AT features. For example, the user must instantiate the STARTUP_VIRTEX6 in their design to drive the KEYCLEARB input in response to some tamper event. Table 4 summarizes each of these active features, their use cases, and how the user can implement the feature. Each feature is described in greater detail after Table 4.

Table 4: **Active Security Features Use Cases**

Feature	Use Case	User How-To
JTAG Disable and Activity Monitor	Prevent and detect unauthorized JTAG access.	Instantiate BSCAN primitive in a special way and add a monitoring/response function in the FPGA logic.
Internal Configuration Memory Integrity	Background check of configuration memory integrity (non-interfering run-time check).	Enable the POST_CRC configuration user constraint and instantiate the FRAME_ECC primitive. Develop FPGA logic to check/respond to configuration memory integrity.
On-chip Temperature & Voltage Monitor/Alarms (SYSMON)	Ensure device is operating within normal environmental limits.	Instantiate the system monitor primitive and develop FPGA logic to check/respond to environment status.
PROG Intercept (PREQ/PACK)	Hold off device configuration to allow non-resettable data elements to be cleared.	Instantiate STARTUP primitive and develop FPGA logic to determine the proper conditions for PROG_ACK assertion after receiving a PROG_REQ. This feature can be used only with encrypted bitstreams.
Unique Identifiers (Device DNA and User eFUSE)	Prevent the design from operating (or operate in a limited manner) if unique identifier is not recognized.	Develop FPGA logic to be able to read/process the unique identifier(s) and determine if they are valid.
Internal Configuration Memory Clearing	Erase the configuration memory in response to a tamper event.	Instantiate ICAP primitive and develop FPGA logic to determine the proper conditions for sending an IPROG command.
Internal AES-256 Key Erase (KEYCLEARB)	Erase the battery-backed key in response to a tamper event.	Instantiate STARTUP primitive and develop FPGA logic to determine the proper conditions for KEYCLEARB assertion. This feature can be used only with encrypted bitstreams.

Table 4: **Active Security Features Use Cases *(Cont'd)***

Feature	Use Case	User How-To
Global 3-State (GTS)	Shut off outputs in response to a tamper event.	Instantiate STARTUP primitive and develop FPGA logic to determine the proper conditions for GTS assertion.
Global Set-Reset (GSR)	Restore user flip-flop states to initial conditions in response to a tamper event.	Instantiate STARTUP primitive and develop FPGA logic to determine the proper conditions for GSR assertion.

JTAG Disabling (Prevention)

An attacker often starts at the JTAG port when trying to break into a system. The built-in JTAG primitive (BSCAN) can be instantiated in a special way to break the JTAG chain and to monitor for any unauthorized activity. For Virtex-6 FPGAs the primitive is named BSCAN_VIRTEX6; for 7 series FPGAs the primitive is named BSCANE2.

To break the JTAG chain for the FPGA (and any other devices in the same chain as the FPGA device, either upstream or downstream), the user can employ two different methods.

The first method is to instantiate a BSCAN primitive in the user design and attach the DISABLE_JTAG => TRUE attribute to it. Here is an example VHDL instantiation for a Virtex-6 FPGA design:

```
BSCAN_VIRTEX6_U0 : BSCAN_VIRTEX6
  generic map (
    DISABLE_JTAG => TRUE,
    JTAG_CHAIN => 1 -- can be 1, 2, 3, or 4 depending on the
                    -- desired physical location of the primitive
  )
  port map (
    CAPTURE => open,
    DRCK    => open,
    RESET   => open,
    RUNTEST => open,
    SEL     => open,
    SHIFT   => open,
    TCK     => tck_signal,
    TDI     => tdi_signal,
    TMS     => tms_signal,
    UPDATE  => open,
    TDO     => '1'
  );
```

Note: The TCK, TDI, and TMS signals are the only ports connected above because they can be used to monitor for any JTAG activity (this is explained in JTAG Monitoring (Detection), page 10).

The second method to break the JTAG chain is to add the -g DISABLE_JTAG option when running BitGen. Either method can be used. The first method ensures that the chain is broken by the user design. The second method defers this decision to break the chain to later on in the tool flow. It is up to the system designer to determine where to make this decision.

Note: If the BitGen option is used and an attacker tries to turn off this option later on, this bit-flip attack is detected by the authentication step and the device does not start up.

If the user has instantiated a soft ChipScope™ Pro logic analyzer core [Ref 11] in their FPGA logic design, breaking the JTAG chain causes the ChipScope Pro logic analyzer core to become inoperative because the core depends on the JTAG port for the communications path back to the host (this also includes any other type of processor debugger that depends on the JTAG port for communication). During the FPGA debug phase, the JTAG chain can be left intact and only broken later on in the development cycle when the ChipScope Pro analyzer's functionality (or processor debugger) is no longer required.

JTAG configuration of the device cannot be used when the option to break the JTAG chain is enabled. The user must choose one of the other configuration interfaces, such as serial, serial parallel interface (SPI), byte parallel interface (BPI), or SelectMAP [Ref 6], [Ref 7]). JTAG boundary-scan board-level tests operate normally as long as the actual configuration of the FPGA device is delayed.

JTAG Monitoring (Detection)

To detect any JTAG activity from within the device, the user needs to monitor any combination of the JTAG TCK, TDI, or TMS line(s) on a BSCAN primitive. Because any external JTAG command requires these lines to toggle, activity detectors on any or all of these lines catch this. For example, to monitor any rising edges on the TDI line, the circuit in Figure 4 could be used.

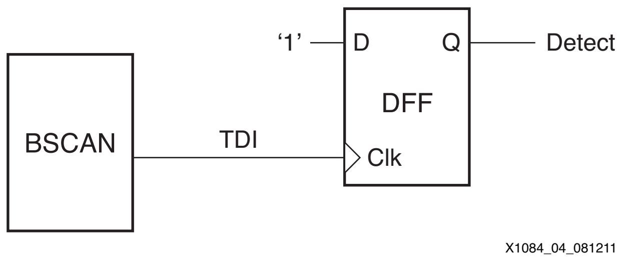

Figure 4: **Example JTAG Activity Detector**

Any rising edge on the TDI line gets latched and the detect output of the DFF remains at 1 until the part is reconfigured (i.e., until the PROGRAM_B input is asserted). This method can be extended to monitor rising/falling edges on any of the TCK, TDI, and/or TMS signal lines. When any of the outputs of the JTAG detector DFFs are set, they can be used to initiate a tamper penalty.

Internal Configuration Memory Integrity (Detection)

Corruption of any of the internal configuration memory cells (which are configured by the decrypted bitstream) could cause the FPGA to operate in an unknown or undesired manner. The corruption could occur by an intentional tamper attack or by an unintentional event such as a single-event upset (SEU). By using the readback cyclic redundancy check (CRC) feature, continuous readback of configuration data in the background of a user design is performed to detect any bit flips. This capability can be used in conjunction with the FRAME_ECC primitive for advanced operations such as SEU corrections.

Implementation details can be found in the Virtex-6 and 7 series FPGAs configuration user guides [Ref 6], [Ref 7] in the Readback CRC chapter. Additionally, Xilinx offers a Soft Error Mitigation (SEM) core [Ref 12] for the Virtex-6 device that has additional detection, correction, and classification features (including error injection so all aspects of a system can be evaluated) contained in a single IP core.

For a more minimal approach, just the POST_CRC check could be used to see if there was a bit flip (corruption) somewhere in the FPGA's configuration memory (overall CRC check on the configuration memory) and then respond at the system level. However, this minimal approach does not allow the user to pinpoint the bit error(s) and perform correction(s).

Temperature/Voltage Monitoring and Alarms (Detection)

By modifying the normal operating voltages and/or temperature of an FPGA, an attacker might attempt to cause the device to behave in an unintended way, perhaps to either extract data from it or cause it to bypass certain security features. For example, FIPS 140-2,*Security*

Requirements For Cryptographic Modules [Ref 13] states: "In particular, the cryptographic module shall monitor and correctly respond to fluctuations in the operating temperature and voltage outside of the specified normal operating ranges."

To help meet this type of requirement, on-chip hard IP blocks can be used, namely, the Virtex-6 FPGA System Monitor (SYSMON) [Ref 14] or the 7 series FPGAs Xilinx analog-to-digital converter (XADC) [Ref 15]. Both are multi-channel ADCs within the FPGA and can be used to monitor on-chip power supply voltages and on-chip die temperature as well as external analog voltages fed into the FPGA. They can also be easily instantiated in the user's design. This type of on-chip monitoring is more secure than off-chip because it is harder to tamper with.

Upper and lower alarm limits can be programmed directly into SYSMON and XADC for the on-chip parameters. Additional FPGA logic can be used to create alarm limits for external voltage inputs (e.g., an external analog voltage tamper loop or the output of a pressure sensor). The status of the alarm signals can be used by the user design/system to determine the appropriate course of action in case they become active (i.e., determine the appropriate tamper penalty). The analog inputs are bandwidth-limited (see *Virtex-6 FPGA System Monitor User Guide* [Ref 14] and *7 Series FPGAs XADC Dual 12-Bit 1MSPS Analog-to-Digital Converter User Guide* [Ref 15] for the maximum input frequencies).

If detection of very fast changes in temperature or voltages is required, an off-chip solution might be required. It is up to the system designer to define what the required detection bandwidth needs to be. *The Sorcerer's Apprentice Guide to Fault Attacks* [Ref 16] describes a number of methods to mount an attack on a chip, one of them being variations in the power supply voltage.

Unique Identifiers (Detection)

Two types of unique identifiers (UIs) are available for use: Device DNA and User eFUSE. They can be used as anti-cloning security measures (i.e., someone steals the user's bitstream and uses it to program their own devices) or for enabling/disabling certain features (upgrade/downgrade) depending on the value of the UI.

Device DNA consists of a 57-bit device-specific serial number and is set by Xilinx in one-time programmable (OTP) fuses on the FPGA during the manufacturing flow (FPGA logic read access to the value is via the DNA_PORT primitive, or it can be read externally via JTAG). User eFUSE provides 32 bits of user read/write OTP area and is set by the user via JTAG (FPGA logic read access to the value is via the EFUSE_USR primitive). Both of these UIs can be used separately or in conjunction for security purposes.

Note: Use of Device DNA and/or User eFUSE provides for unique IDs, but does not provide cryptographically strong confidentiality or authentication (such as AES-256/HMAC). AES-256 encryption is the preferred method of providing anti-cloning protection. However, by taking advantage of these UIs, the designer can add another layer to the overall AT scheme.

These UIs can be used to link the bitstream to one particular device (in the case of Device DNA or multiple devices in the case of User eFUSE). The UI comparison is put into the FPGA design by the user and the results of this comparison can be used to gate FPGA activity. For example, if the UI comparison fails, the design can refuse to function or function with limited capability. An example use case of the UIs is as follows:

1. Setup: Read the UI value(s) from the FPGA via JTAG, generate a checksum from the UI value(s), and store in a flash device accessible to the FPGA (using a robust one-way function—a keyed type is the most secure) as shown in Figure 5.

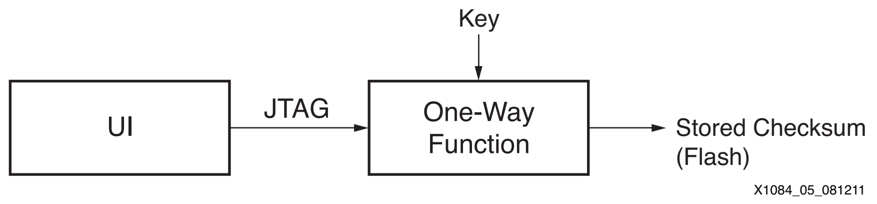

Figure 5: **Encrypt the DNA Value With a Confidential Key**

The key for the one-way shown in Figure 5 could be stored within the encrypted bitstream. If bitstream encryption is not used, this approach relies on the complexity of the bitstream to keep the key confidential.

2. Configure the FPGA.
3. Compare: The FPGA user design reads the UI value from either or both the DNA_PORT and EFUSE_USR primitive(s) and then calculates the checksum using the same algorithm. The user design then compares the calculated checksum with the checksum read from Flash. If the checksum passes, the design is allowed to become active.

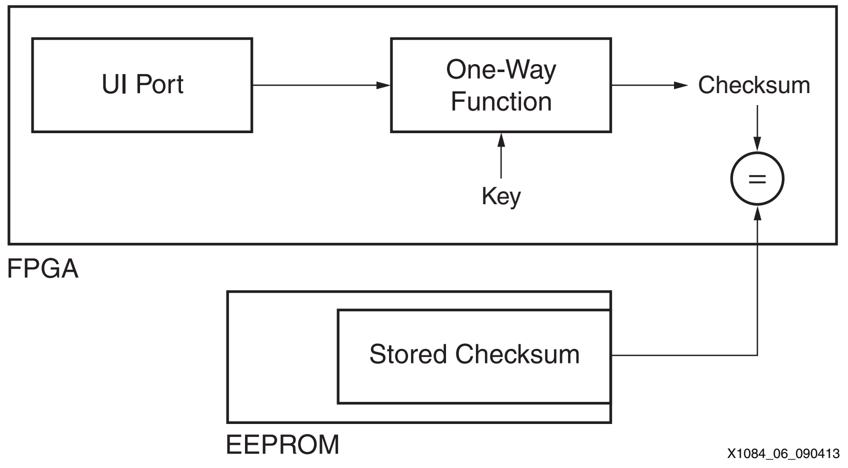

Figure 6: **Checksum Comparison**

For additional information on these UIs, consult the Virtex-6 FPGA and 7 series FPGAs configuration user guides [Ref 6], [Ref 7]. *Security Solutions Using Spartan-3 Generation FPGAs* [Ref 17] also talks about Device DNA operation for certain Spartan class FPGA devices.

Critical User Clock/Signal Monitoring (Detection)

When using bitstream encryption, the user can take advantage of an uninterruptible clock source named CFGMCLK (configuration internal oscillator clock output), located as an output on the STARTUP block primitive (refer to the example VHDL instantiation of the STARTUP block in PROG Intercept (Prevention and Detection). Because this clock is always active, it can be used as the basis for a user clock (or other critical user signal) monitoring function. Even though CFGMCLK can vary quite a bit from its nominal value of 50 MHz (±55% in Virtex-6 devices and ±50% in 7 series and Zynq devices), it can still be quite useful in a monitoring function to make sure a critical user clock or signal is still “alive” and toggling between a lower and upper frequency range (which takes into account the CFGMCLK variation). If the critical user clock or signal falls out of this range, it can indicate either that the user design has malfunctioned or is being tampered with, and the appropriate penalty could be asserted.

PROG Intercept (Prevention and Detection)

Not all memory elements in the FPGA are cleared upon configuration. For example, there might be microprocessor caches or gigabit speed serial I/Os (GTX and GTH transceivers) with FIFOs in use that retain state even after the external PROGRAM_B pin is asserted (assertion of PROGRAM_B causes the FPGA to reset and become reconfigured via the bitstream). An attacker could potentially assert the PROGRAM_B pin and replace the user bitstream with their own bitstream (one that is designed to dump out the contents of the uncleared memory elements after the FPGA is configured). By using the PROG intercept feature (there is a PROG request/acknowledge pair on the STARTUP block PREQ/PACK), the user can indefinitely delay the reconfiguration of the FPGA so that these memory elements can first be cleared by the user design (or any other housekeeping tasks that might need to be performed before allowing a PROGRAM_B to happen).

Another use case for PROG intercept could be a system (when fielded) that should never see PROGRAM_B assertion while active—its mere presence signifies a tamper event. One of the first things an attacker might do is to assert the PROGRAM_B to observe the start-up behavior of the FPGA. If the user design sees the PREQ go active, it could invoke a penalty and then allow the PROGRAM_B to occur (by asserting PACK).

Whenever an encrypted bitstream has been loaded (with the PROGRAM_B intercept security feature enabled) and the PROGRAM_B pin is asserted externally (or an internal IPROG command is sent to the ICAP), the FPGA logic sees the PREQ output assert High on the STARTUP block. Configuration is held off indefinitely until the user drives the PACK input High (rising-edge sensitive) or until the device is power cycled.

The following is an example VHDL instantiation of the STARTUP block with the correct security generic set and the PREQ/PACK signal connections:

```
STARTUP_U0 : STARTUP_VIRTEX6
  generic map (
    PROG_USR => TRUE ) -- turn on PROGRAM_B intercept security feature
  port map (
    CFGCLK     => open,
    CFGMCLK    => cfgmclk_signal,
    DINSPI     => open,
    EOS        => open,
    PREQ       => preq_signal, -- PROGRAM request to FPGA logic output
    TCKSPI     => open,
    CLK        => '0',
    GSR        => gsr_signal,
    GTS        => gts_signal,
    KEYCLEARB  => keyclearb_signal,
    PACK       => pack_signal, -- PROGRAM acknowledge input (rising edge)
    USRCCLKO   => '0',
    USRCCLKTS  => '0',
    USRDONEO   => '0',
    USRDONETS  => '0'
);
```

Key Clear (Response/Penalty)

The crown jewel of the FPGA is the AES key used to decrypt the bitstream. After an attacker has access to the key, the contents of the original bitstream can easily be determined. By connecting the user design to the KEYCLEARB input (located on the STARTUP block primitive, in PROG Intercept (Prevention and Detection), page 13), the system designer can choose to assert this signal as a penalty in response to internal (or external) tamper events. By erasing both the 256-bit key in BBRAM and the 1920-bit expanded key in the decryption block, the encrypted bitstream stored in off-chip non-volatile memory becomes useless to the attacker. The KEYCLEARB signal has no effect on the non-volatile eFUSE key.

The decision for the KEYCLEARB assertion (as well as all other tamper responses) does not necessarily need to originate from within the FPGA itself. It could be due to a tamper event somewhere else in the system (e.g., a breach of a system-level or module-level tamper boundary).

After the key is cleared, the FPGA device is useless until reprogrammed with the same key again or reconfigured using IPROG [Ref 6], [Ref 7] to a different bitstream (perhaps an unencrypted bitstream with reduced functionality). Ensure that the KEYCLEARB input is only asserted under the proper conditions. In many cases, equipment must be taken out of the field and sent back to a central depot or manufacturing facility to be re-enabled with a key load operation.

IPROG (Response/Penalty)

IPROG is an internal command sent through the ICAP interface that clears the FPGA configuration memory, all flip-flop contents, and key expansion memory, but not the key itself. It is equivalent to the assertion of the external PROGRAM_B pin. This command effectively clears configuration memory (configuration data, block RAMs, and flip-flop state) and can be combined with the KEYCLEARB signal in response to tampering. The KEYCLEARB assertion must occur before an IPROG command is sent (the IPROG command can be sent to the ICAP immediately following the assertion of KEYCLEARB).

After both of these penalties are invoked, the FPGA becomes inoperable because the existing bitstream can no longer be decrypted by the FPGA. The fact that device configuration is no longer possible (with the encrypted bitstream) would be an indication that a tamper event has occurred. At this point, the user design might choose to load in an unencrypted bitstream after the KEYCLEARB/IPROG penalties so that there is some basic/reduced functionality without exposing any of the CT.

To send an IPROG command to the configuration engine, the ICAP primitive must be instantiated in the user design and the appropriate sequence of commands must be written to it. For more information, refer to the IPROG Reconfiguration sections in the Virtex-6 FPGA or 7 series FPGAs configuration user guides [Ref 6], [Ref 7].

Global 3-State (Response/Penalty)

Depending on the system design, critical (red) information might flow out of the external FPGA pins (e.g., a crypto module). Asserting the global 3-state (GTS) input on the STARTUP block (see the sample code in PROG Intercept (Prevention and Detection), page 13 and section Global Reset (Response/Penalty) in response to a tamper event causes all FPGA outputs to immediately enter a high-Z state and prevent any more data from flowing outside the FPGA. This could be an immediate step to take just prior to an IPROG and/or KEYCLEARB to make sure red data flow is halted as soon as possible.

Global Reset (Response/Penalty)

Critical data or sensitive parameters can be stored in FPGA logic registers such as user key (not the AES BBRAM key). Asserting the global reset (GSR) input on the STARTUP block (see PROG Intercept (Prevention and Detection), page 13) in response to a tamper event causes all FPGA registers (i.e., flip-flops) to be restored to their default state. This could be an immediate step to take just prior to an IPROG and/or KEYCLEARB to make sure all sensitive data in the FPGA is erased as soon as possible. The GSR does not impact shift register look-up table (SRL) or block RAM contents; these must be cleared by the user design or by an IPROG command.

Tamper Resistance Guidance

This section provides guidance and technical tips that can be used in conjunction with the previously addressed built-in silicon AT features to create tamper-resistant designs using Virtex-6 FPGAs and 7 series FPGAs.

Load CT Only When Needed (Prevention)

If the user design can be partitioned into sections that contain non-critical and critical technology (CT) blocks, it might be possible to only have the non-CT portion of the user design resident at all times and use partial reconfiguration (PR) features [Ref 18] of the FPGA to allow the CT to be loaded *only when needed*. The CT can then be erased (by loading in a *black box* version of the PR region) when it has completed its tasks. The partial bitstream for the CT can be decrypted by the AES key or in the FPGA logic by the system designer's algorithm of choice. In response to a tamper event, both the PR region and the key for the CT (perhaps stored in block RAM or FPGA logic registers) could be erased.

For example, Figure 7, page 15 and Figure 8, page 16 illustrate a general system with an FPGA, CPU, and external memory devices (for FPGA configuration, PR, CPU code, and CPU boot code). In Figure 7, the PR region (named User CT Logic Region) is empty.

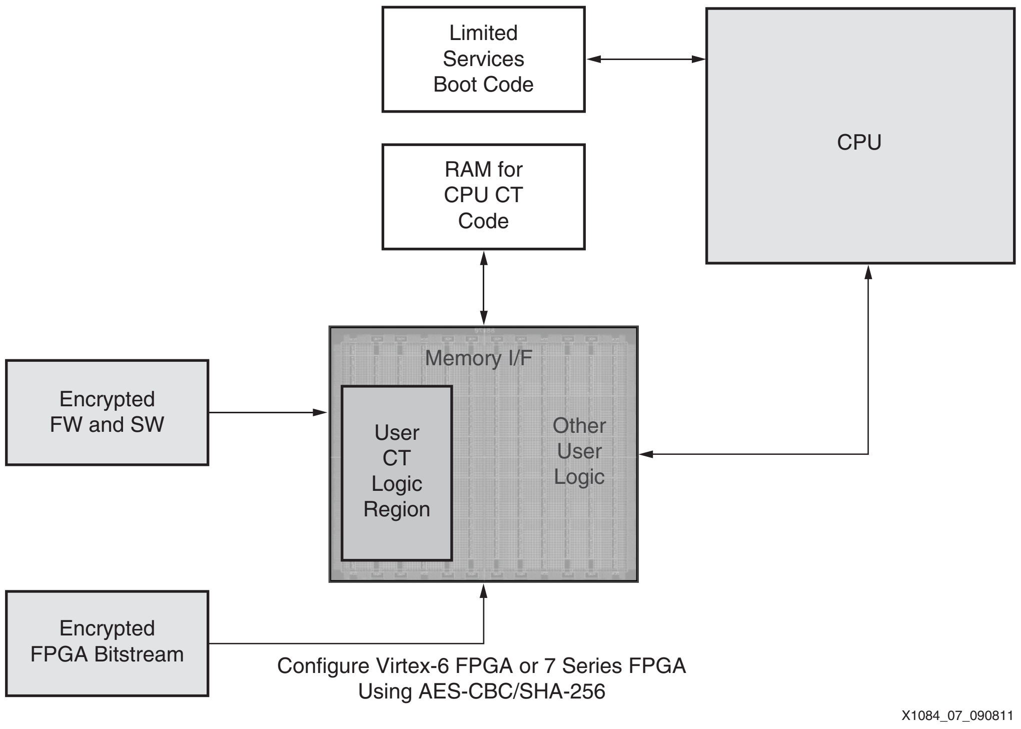

Figure 7: **Use Model: Protecting System CT**

In Figure 8, this region on the FPGA has been dynamically loaded with CT logic via PR (named User CT Logic). When the CT function is completed (or a tamper event occurs), the PR region can be returned to the state shown in Figure 7 by loading in a black box version of the PR module.

Note: If a tamper event was to occur, the sensitive CPU code memory could also be erased so that only encrypted, cleared, or non-sensitive external memory contents remain.

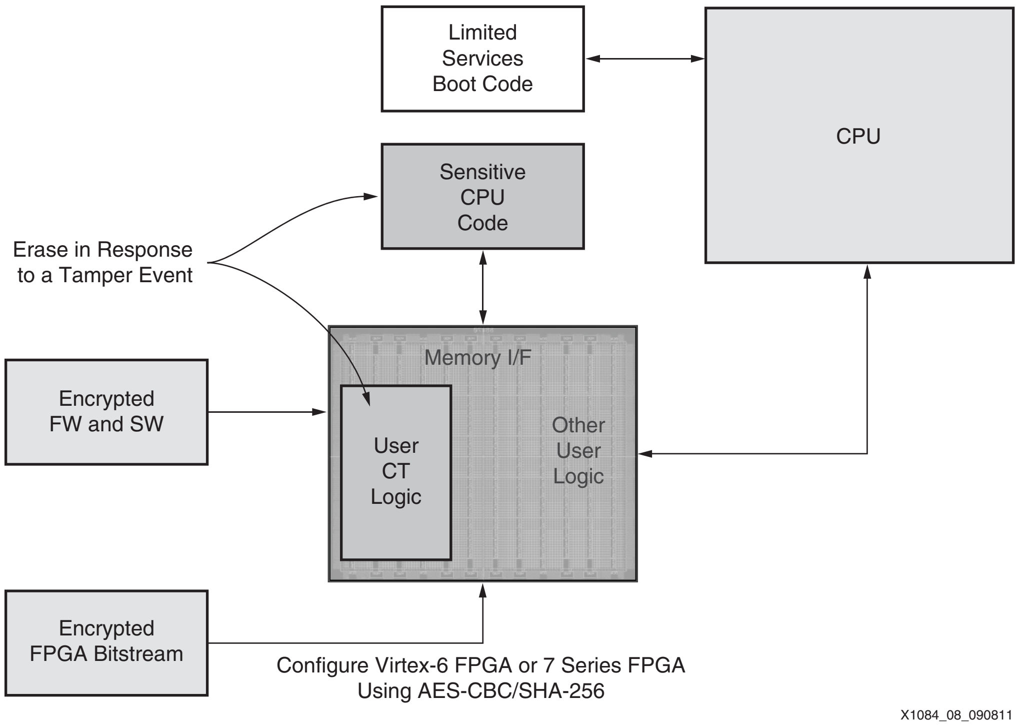
Limited Services Boot Code
CPU
Sensitive CPU Code
Erase in Response to a Tamper Event
Memory I/F
Encrypted FW and SW
User CT Logic
Other User Logic
Encrypted FPGA Bitstream
Configure Virtex-6 FPGA or 7 Series FPGA Using AES-CBC/SHA-256
X1084_08_090811

Figure 8: **Use Model: Protecting System CT—Tamper Response**

In these examples, the ICAP is used as the means of performing the PR. Because the ICAP is a trusted channel, it allows either encrypted or unencrypted PR bitstreams (even if an eFUSE key is being used with the `cfg_aes_only` eFUSE bit blown). An encrypted PR bitstream is always recommended (whether the decryption takes place in the dedicated decryption logic in the FPGA's configuration engine or with the user's decryptor of choice in the FPGA logic). A useful reference on secure PR is *PRC/EPRC: Data Integrity and Security Controller for Partial Reconfiguration* [Ref 19].

Key Erase via External Shunt

Another method to erase the BBRAM key is via an external shunt to ground on the V_{BATT} line. This method can be used to erase the key when main power (V_{CCINT} and V_{CCAUX}) to the FPGA is not applied because active features, such as KEYCLEARB, can be used only after the FPGA is powered up and configured. For instance, if a system-level tamper event is detected prior to the FPGA having its main power applied, the external battery power line to the FPGA's V_{BATT} pin can be opened and the V_{BATT} pin driven to ground with some sort of transistor shunt. Care must be taken that the circuit is designed to open the battery connection before shunting the V_{BATT} pin to ground. Another option is to connect the battery to the V_{BATT} pin via a resistor (V_{BATT} pin maximum input current is 150 nA). By choosing the appropriate resistance value, the V_{BATT} pin can be shunted to ground directly without causing excessive current flow out of the battery.

If an FPGA is not powered up (no V_{CCINT}, V_{CCAUX}, etc., except for V_{BATT}), it would take a worst-case maximum time of 50 ms for the AES key (stored in BBRAM) to become erased if a user was to properly shunt the V_{BATT} pin to ground (at –55°C). Additionally, with only V_{BATT}

powered, the BBRAM key can be retained in the storage temperature range of –65°C and 150°C.

Preemptive Key Erase

Another use case for key erase could be a preemptive measure.

After loading and decrypting the bitstream, the BBRAM key can be purposely erased before sending the system out into the field. Of course, this would only work for a system that is not intended to be power-cycled after it becomes deployed (for example, a missile after being launched). This could also be used for an eFUSE key by writing over it with all ones via JTAG before deployment (as long as eFUSE key reading/writing has not been disabled yet).

Avoiding Weak or Duplicate Keys

All zeroes, all ones, or repetitive patterns should never be used in user keys. Keys should not be reused (or the same key used for AES and HMAC) if at all possible. Personnel access to the key values should be tightly controlled (i.e., only those with a need to know have access to key data). Ideally, a random source should be used to create the keys (avoiding weak keys; for example, an all zeroes random key is theoretically possible). The BitGen software can generate AES and HMAC keys. However, it uses a pseudorandom process seeded with the current date and time. The most secure keying material comes from a truly random process.

Key management is a very important element in any cryptographic system (it is probably the most complex element). For additional help on this subject, the NIST Key Management Guideline [Ref 20] is a useful reference.

Updating Keys in the Field

One obvious approach is using the iMPACT software to load the key in the FPGA that corresponds to the updated bitstream encrypted with a new key (this assumes the user has a portable computer with JTAG programming cable access). Another option is to use *Xilinx In-System Programming Using an Embedded Microcontroller* [Ref 21] or *Embedded JTAG ACE Player* [Ref 22]. This allows the user to play a JTAG sequence from an adjacent processor based on a generated -keyonly option from iMPACT, which then allows the new key to be updated in BBRAM. It is important for the user to remember that because this is a *red* key load (i.e., the key is not encrypted), the appropriate safeguards must be taken to prevent unauthorized access during transfer of the red key data to the FPGA's JTAG port.

Sending Tamper Status Outputs to System

Upon a tamper event (in addition to asserting a penalty) the user design could send tamper status information back to the system (or write it to a non-volatile memory). The system could then store this information away for future auditing purposes. Because the FPGA is volatile, it would have to be designed to transmit the data before an IPROG command (tamper penalty) is given.

Restricting Access to FPGA Probe Points

Making it difficult for an attacker to get near the FPGA is a good example of a layered approach. A robust tamper boundary (perhaps with tamper detect switches) can be used around any device(s) that might contain CT (for example, an activated tamper switch could cause a shunt to go active on the FPGA’s V_{BATT} line). Buried vias and routing can be used on the printed circuit board for FPGA signals, power supply routing can be kept within buried layers (and difficult to get to), and adequate decoupling can be used (buried capacitance technology should be used, if possible). JTAG boundary-scan techniques can be relied on for board-level factory testing and test points should be removed from production boards.

Differential Power Analysis (DPA)

DPA attacks on semiconductor devices have been known and understood for a number of years [Ref 23]. These attacks can be targeted at many different devices including processors, microcontrollers, payment cards (smart cards), as well as FPGAs. Academic papers have been published that claim to have successfully performed a DPA attack upon an FPGA to extract its bitstream decryption key (when using boards specially designed to enable such an attack).

While technically feasible, such attacks are not trivial to execute without specialized knowledge and equipment, as well as close physical access to the device. Systems engineers with concerns about the security of their design should implement a comprehensive (multi-layered) security strategy to protect their architecture and design files, access to physical equipment, and any other measures they deem appropriate. Xilinx has ongoing research and development in this area. Contact your local FAE for more information.

Conclusion

This application note summarizes the AT features in Virtex-6 and 7 series FPGAs currently available to the FPGA designer and gives practical examples of how to use them effectively. By taking advantage of these features and following the AT guidance presented (early on in the design cycle), a tamper-resistant FPGA design can be realized.

No single AT feature or technique is 100% effective all of the time or can meet all the AT needs for the entire system. However, making the adversary's job as difficult (and expensive) as possible and following a layered approach almost always yields very good (if not excellent) results.

The tools and technologies for the development and testing of new integrated circuits (including FPGAs) are always evolving and improving. In parallel, the tools used by the adversary also evolve and improve, so it is important for the FPGA designer to be aware of what AT features and techniques are available for use. Additionally, Xilinx is committed to staying abreast of these developments to enhance and/or develop new features to protect customer IP now and into the future.

References

The following are referenced in this document:

1. Xilinx Getting Started
   http://www.xilinx.com/company/gettingstarted/index.htm
2. Xilinx ISE Design Suite
   https://www.xilinx.com/products/design-tools/ise-design-suite.html
3. WP365, *Solving Today's Design Security Concerns*
4. Security Monitor Product Brief
   http://www.xilinx.com/publications/prod_mktg/CS1140_AD_SecMonIP_ProdBrf_Update_June_2012.pdf
5. Introduction to Indirect Programming – SPI or BPI Flash Memory
   http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/pim_c_introduction_indirect_programming.htm
6. UG360, *Virtex-6 FPGA Configuration User Guide*
7. UG470, *7 Series FPGAs Configuration User Guide*
8. *The Keyed-Hash Message Authentication Code (HMAC)*, FIPS PUB 198a
   http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf
9. *Secure Hash Standard*, FIPS PUB 180-2
   http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf
10. XAPP1175, *Secure Boot of Zynq-7000 All Programmable SoC*
11. Xilinx ChipScope Pro and the Serial I/O Toolkit
    http://www.xilinx.com/tools/cspro.htm

12. Xilinx Soft Error Mitigation (SEM) Core
http://www.xilinx.com/products/intellectual-property/SEM.htm
13. *Security Requirements for Cryptographic Modules*, FIPS PUB 140-2
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf
14. UG370, *Virtex-6 FPGA System Monitor User Guide*
15. UG480, *7 Series FPGAs XADC Dual 12-Bit 1MSPS Analog-to-Digital Converter User Guide*
16. Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. *The Sorcerer's Apprentice Guide to Fault Attacks.*
http://eprint.iacr.org/2004/100.pdf
17. WP266, *Security Solutions Using Spartan-3 Generation FPGAs*
18. Xilinx Partial Reconfiguration
http://www.xilinx.com/tools/partial-reconfiguration.htm
19. XAPP887, *PRC/EPRC: Data Integrity and Security Controller for Partial Reconfiguration*
20. NIST Key Management Guideline
http://csrc.nist.gov/groups/ST/toolkit/key_management.html
21. XAPP058, *Xilinx In-System Programming Using an Embedded Microcontroller*
22. XAPP424, *Embedded JTAG ACE Player*
23. Google differential+power+analysis search results
http://scholar.google.com/scholar?q=Differential+power+analysis&hl=en&as_sdt=0&as_vis=1&oi=scholart

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
09/21/11	1.0	Initial Xilinx release.
12/01/11	1.1	In Introduction, clarified availability of Security Monitor. In Bitstream Encryption/Decryption, clarified description of BitGen generated keys.In the Note in Bitstream Authentication, clarified authentication. In Figure 2, changed background of Authenticated box to white. In Hardened Readback Disabling Circuitry, revised the Note regarding the BitGen security option. In Table 4, updated User How-To for PROG Intercept (PREQ/PACK) and Internal AES-256 Key Erase (KEYCLEARB). In JTAG Disabling (Prevention), revised format of the VHDL source code. Added Global 3-State (Response/Penalty). Revised first paragraph of Key Erase via External Shunt. In Updating Keys in the Field, added red key load clarification at the end of the section. In Differential Power Analysis (DPA), revised the last sentence of the two paragraphs.
08/10/12	1.2	Added *Security Monitor Product Brief* and *Introduction to Indirect Programming – SPI or BPI Flash Memory* to References. Updated descriptions of Value and Adversary in Introduction. Added notes to Table 2 and Table 3. Added Critical User Clock/Signal Monitoring (Detection). Updated IPROG (Response/Penalty).
10/15/13	1.3	Updated Zynq devices throughout. Added RSA Asymmetric Bitstream Authentication to Table 1. In note 1 of Table 3, replaced "core" with "bitstream." Added RSA Asymmetric Bitstream Authentication (Zynq Devices Only). Updated Figure 6. Updated third paragraph in PROG Intercept (Prevention and Detection).
06/13/17	1.4	Added Spartan-7 FPGAs to Summary.

Notice of Disclaimer