MILITARY ANTI-TAMPERING SOLUTIONS USING PROGRAMMABLE LOGIC

Charlie Jenkins (Altera, San Jose, California, chjenkin@altera.com)
Christian Plante (Altera, San Jose, California, cplante@altera.com)

ABSTRACT

Military applications are becoming increasingly complex. Major programs such as Future Combat Systems (FCS), Joint Strike Fighter F-35 (JSF), and the Joint Tactical Radio System (JTRS) are pushing technological capabilities to their limits. Due to the technology requirements and environments to which they are exposed, these military systems rely on programmable logic (FPGAs) to provide extreme flexibility plus protection from tampering. As FPGAs become an integral part of leading-edge architectural design replacing ASICs and ASSPs, the security of the FPGA design and configuration bitstream is of utmost importance. This paper describes two techniques—configuration bitstream encryption and handshaking tokens—for securing designers' intellectual property (IP) within SRAM-based FPGAs.

1. INTRODUCTION

Anti-tampering capabilities are imperative for today's varied military scenarios and applications. The exploitation of system vulnerabilities by enemies, former allies, and counterintelligence personnel can have serious military impacts, including faulty/compromised operations, reverse-engineered technology for superior devices, and premature system obsolescence. Applications areas include:

- *Missiles and munitions:* To meet the long shelf life required by these applications, anti-tampering capabilities require non-volatile encryption key storage that eliminates the need for batteries.
- *Electronic warfare:* Non-volatile encryption key storage is ideal in applications where limited space, low maintenance, and ultimate reliability are required, and that necessitate the elimination of extra components like batteries.
- *Secure communication:* With the need to secure communication on the battlefield, the encryption of FPGA configuration bitstreams provides an additional security level above and beyond current methods used.
- *Remote sensors and surveillance:* The ability of FPGAs to protect critical IP is important for applications that are exposed to harsh environments throughout the battlefield.

2. ISSUE OF DESIGN SECURITY WITH FPGAS

In order for an FPGA to accomplish a certain function, source code must be generated first. For SRAM-based FPGAs, the compiled version of this source code is called a programming file or, more simply, a configuration bitstream. Similar to standard microprocessors, the configuration bitstream, stored in external FLASH or other memory, must be programmed into the FPGA before it can start operating. Unfortunately hackers or rogue entities interested in capturing the source code for reverse-engineering, tampering, or disabling purposes can try to read the content of the configuration memory, intercept the bitstream while it is being transferred to the FPGA, or read back the bitstream from the FPGA.

The solution to this problem is the use of encryption. To protect the intellectual property (IP) implemented in the original code, designers need to encrypt the bitstream before access is granted to external parties, such as external contractors or contract manufacturers. Once the bitstream is encrypted, the target FPGA must implement internal circuitry to reliably and safely decrypt the encoded data, while preventing read-back or tampering.

3. METHODS FOR PROTECTING FPGA IP

Two techniques, configuration bitstream encryption and handshaking tokens, can be used to secure IP within SRAM-based FPGAs. Bitstream encryption is better supported using the Advanced Encryption Standard (AES) (FIPS-197). AES is compatible with key lengths of 128, 192, and 256 bits. AES keys provide more stringent protection than other methods such as the 56-bit key size Data Encryption Standard (DES) and triple DES (112-bit effective key size). To understand the increased security level of AES, studies have shown that if a machine could discover a DES key every second, it would take approximately 149 trillion years to discover a 128-bit AES key.

In order to enable encryption, keys are required to encode the bitstream generated by the design software tool (such as Altera® Quartus® II development software).

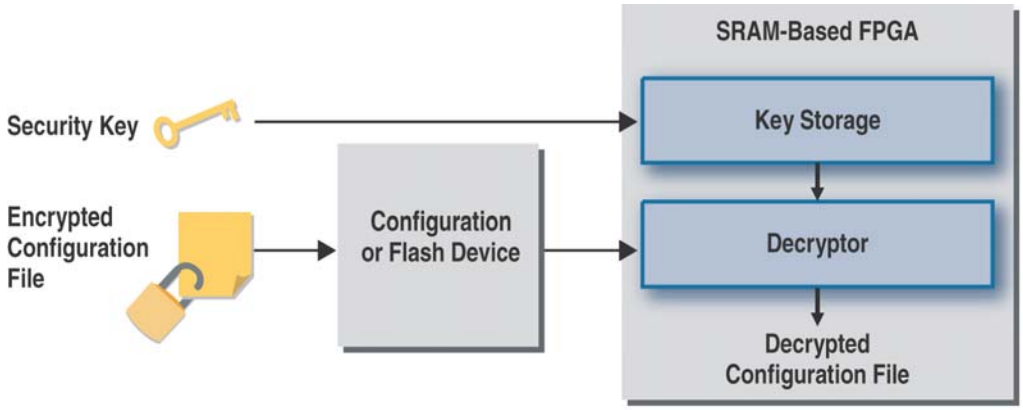

Figure 1. Design Security With FPGA-Based Key

Figure 1 shows a diagram depicting the use of encryption keys. The chosen key must also be known by the AES decryptor inside in the FPGA. This key may be programmed permanently on the FPGA (non-volatile) or can be stored in a special memory location (volatile).

The non-volatile key is stored in the FPGA using one-time programmable polyfuses and retains its information when the power is off, eliminating the need for unreliable battery backup in harsh military environments. The key is programmed into the FPGA during regular manufacturing flow, either before or after assembly onto the printed circuit board. Keys stored in volatile memory require an external backup battery when there is no power to the device. Keys can also be easily reprogrammed on the manufacturing line or in the field.

For systems designed around FPGAs that do not offer on-board AES key support, the handshaking tokens method offers a valid alternative. With this method, the FPGA communicates with an external secure device, such as a CPLD, which includes a non-volatile encrypted token. The FPGA design must read this token and compare it with a matching token, otherwise the design will shut down.

With these methods, the decryption key is securely stored inside the FPGA. Even if the configuration bitstream is captured, it is virtually impossible to decrypt without the appropriate key and therefore cannot be used to configure another FPGA. Further read-back of a decrypted configuration file is not allowed by some FPGA vendors.

Reverse engineering of any FPGA design through the configuration bitstream is very difficult and time-consuming, even without encryption. For high-density devices, the configuration file may contain millions of bits. Some FPGA vendors' configuration file formats are proprietary and confidential, providing another layer of security. With the addition of configuration bitstream encryption, it might be easier and quicker to build a competitive design from scratch rather than trying to reverse engineer such a design.

Tampering cannot be prevented if a volatile key is used because the key is erasable; once the key is erased, the device can be configured with any configuration file. For the non-volatile key solution, the device can be set to only accept configuration files encrypted with the stored key. A configuration failure signals possible tampering with the configuration file, whether in the external memory, during transmission between the external memory and the FPGA, or during remotely communicated system upgrades. This is another advantage of a non-volatile key.

4. TOKEN-BASED ENCRYPTION

Configuration bitstream encryption is only available in high-density, high-performance, SRAM-based FPGAs. The following solution allows any FPGA design to remain secure even if the configuration bitstream is captured. This is accomplished by disabling the functionality of a user design within the FPGA until handshaking tokens are passed to the FPGA from a secure external device. The secure external device generates continuous handshaking tokens to the FPGA to ensure continuous operation. This concept is similar to the software license scheme shown in Figure 2.

Configuring the FPGA is similar to installing software onto a computer; the configuration bitstream is not protected. The external secure device is similar to the license file. The software will only operate when a valid license file is present. Likewise, the user design within the FPGA will only operate when the handshaking tokens sent from the external secure device are valid. Figure 3 shows a simplified hardware implementation for this solution, where a CPLD is used as the secure external device because it is non-volatile and retains its configuration data during power down.

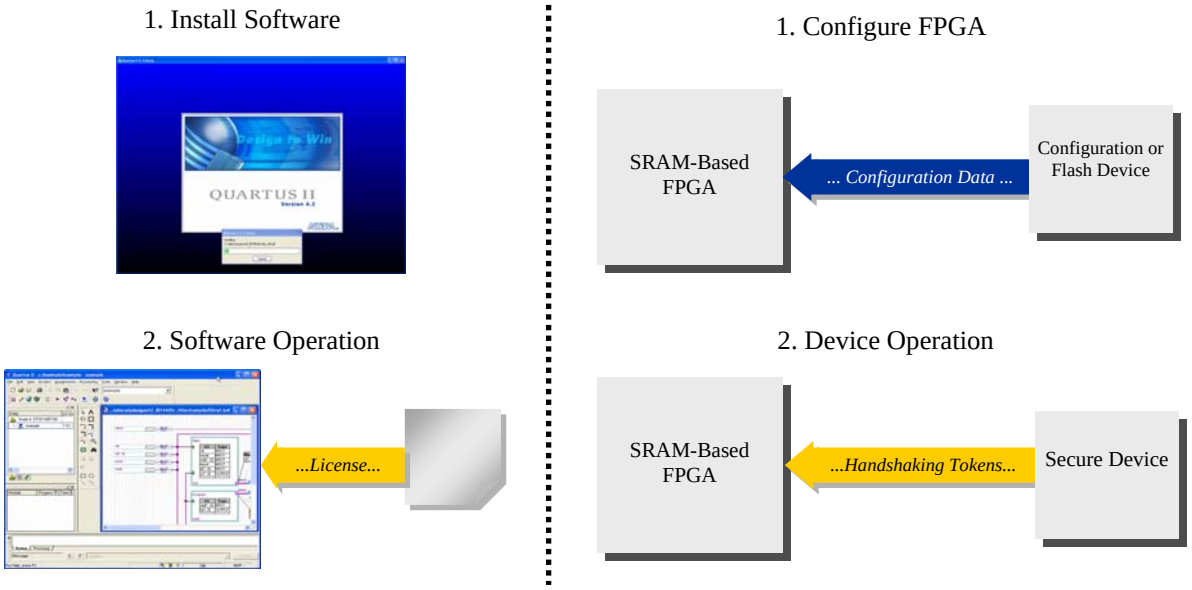

Figure 2. Comparison of Software License Scheme and FPGA Security Scheme

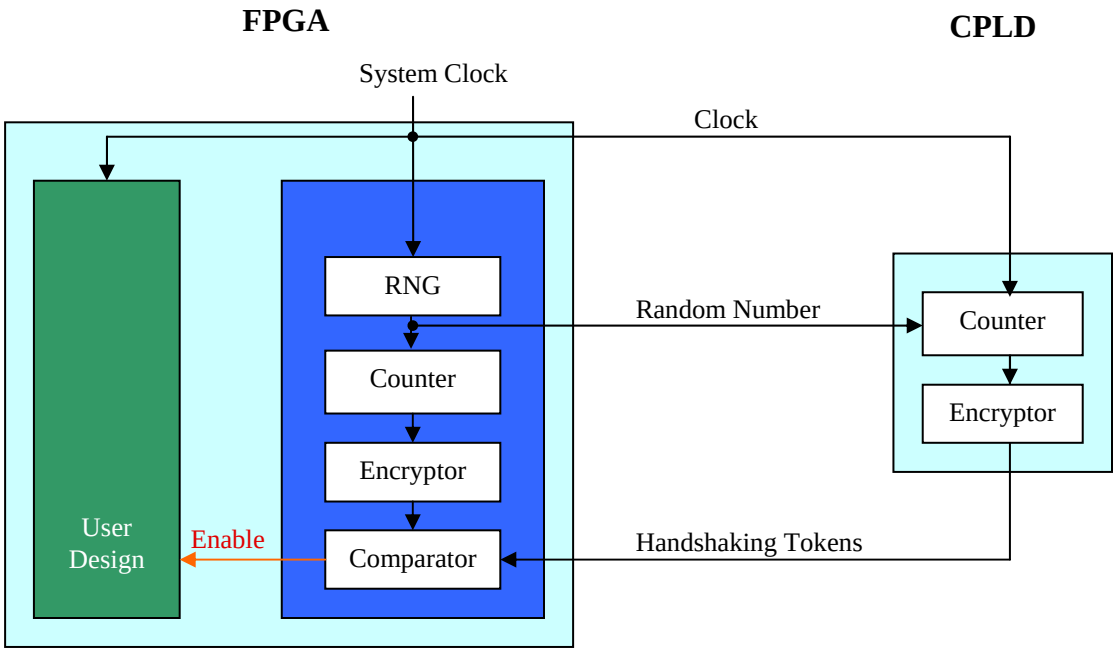

Figure 3. Simplified Hardware Implementation of the FPGA Design Security Solution

Once the FPGA is configured, its user design functionality is disabled. Then, after properly handshaking tokens with the external CPLD, the enable signal is asserted by the security block within the FPGA. The random number generator (RNG) generates and sends the initial counter value to the CPLD, which encrypts the counter value and sends the resulting handshaking token to the FPGA. If the handshaking token matches the data generated inside the FPGA, the enable signal is asserted, and the user design starts functioning. This process continues during the entire operation of the FPGA. A mismatch will cause the enable signal to go low and disable the functionality of the user design. Figure 4 shows an example of how the enable signal is used with a simple AND gate.

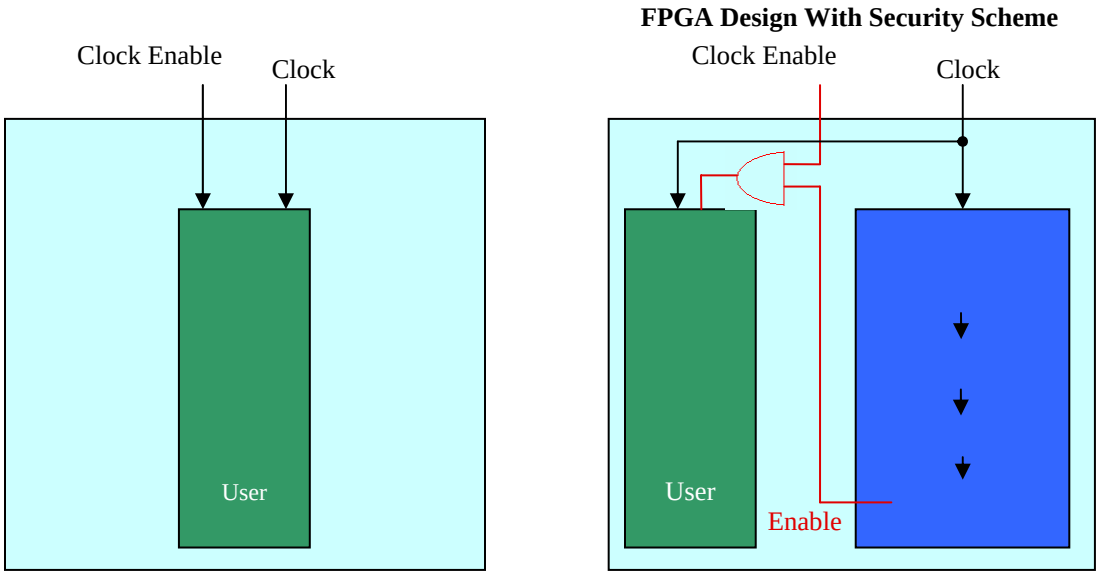

Figure 4. Design With Security Scheme

The FPGA user design works only when the handshaking tokens from the external secure device and the data generated inside the FPGA are identical. Even if the FPGA configuration bitstream is stolen, it is useless, similar to software without a license. Therefore, the FPGA user design is secure from copying. This solution does not provide additional protection against reverse engineering (though difficult) and tampering.

The security of the solution relies on the external device to be secure and the handshaking tokens to be unpredictable. A secure external device needs to be non-volatile and to retain its configuration during power-down (e.g., CPLDs or security processors). The RNG in the solution is critical. It ensures that every time the device starts up, it uses a different initial value. This prevents anyone from storing the handshaking tokens in a storage device. To prevent someone from detecting the pattern in the handshaking tokens, a proven encryption algorithm such as AES should be used.

To ensure the security scheme works properly, the system clock feeding the FPGA user design should be the same as the system clock feeding the security block. This prevents someone from disabling the security block when the enable signal is asserted. To further increase security, the comparator block can be duplicated several times to produce more enable signals to feed different portions of the user designs.

5. DESIGN SECURITY AND OUTSOURCING OF MANUFACTURING

For cost reasons, manufacturers of military equipment are increasingly outsourcing the manufacturing of certain modules to foreign contractors. There is a growing concern that sensitive IP can be leaked to non-authorized entities. Thus, in some cases, FPGA-based IP has to be kept secret from the manufacturing subcontractor.

Therefore, encryption keys should only be known by the original equipment manufacturer (OEM). From a development perspective, this is easily controlled by restricting access to the key inside the OEM's organization. Bitstreams can be encrypted and then easily and safely shared with outside contractors.

On the hardware side of things, design security schemes based on non-volatile keys need to rely on a trusted entity being responsible for programming the key into the FPGA. Once programmed, the FPGA can be provided to the manufacturing company safely. Another approach has the subcontractor build the board and then send it to a trusted party for on-board programming of the non-volatile encryption key. This means that final system testing has to be done by this trusted partner, or requires the system to be shipped back to the original manufacturer. The use of a volatile key also requires the system be shipped to a trusted partner. This partner will have access to the key and the equipment needed to program it. Additionally, volatile keys can be programmed in the field. For such systems, a non-encrypted bitstream that does not include any sensitive IP can be generated to test the hardware system. The final key can then be programmed in the field by the trusted owner of the final equipment.

6. CONCLUSION

Building on the success of the Stratix® II design security implementation, Stratix III devices are the industry's first

FPGAs to support configuration bitstream encryption using AES and a 256-bit key with the option of volatile or non-volatile on-chip storage. Other FPGA vendors only support encryption using a battery to power up or back up a volatile key. Only Stratix III FPGAs provide the choice between the flexibility of a battery-backed-up volatile key or the ultimate security of a non-volatile scrambled key. Moreover, Stratix III devices do not allow read-back of configuration data, thus providing an additional level security against reverse engineering. The following points summarize some of the key features of Altera's latest anti-tampering solutions:

- Both non-volatile encryption key storage (no battery backup required) and volatile (for field programmability) are supported.
- The encryption key is stored securely inside the FPGA, using internal circuit techniques virtually impossible to jeopardize.
- Once inside Altera FPGAs, the configuration file (encrypted or unencrypted) cannot be read back, adding another layer of security.
- Supports 128-bit and 256-bit AES encryption keys.
- FIPS-197 certified.

In an era of ever-increasing security concerns, enhanced anti-tampering capabilities of Altera FPGAs provide military application designers the assurance that their IP within these systems is secure against copying, reverse engineering, and tampering.

7. REFERENCES

[1] Altera, *Design Security Using MAX II CPLDs*, September 2004, http://www.altera.com/literature/wp/wp_m2dsgn.pdf.

[2] Altera, "Design Security in Stratix II Devices", http://www.altera.com/products/devices/stratix2/features/security/st2-security.html