Trusted Design in FPGAs

Steve Trimberger
Xilinx
2100 Logic Dr.
San Jose, CA 95124
1-408-879-5061
steve.trimberger@xilinx.com

ABSTRACT
Using FPGAs, a designer can separate the design process from the manufacturing flow. Therefore, the owner of a sensitive design need not expose the design to possible theft and tampering during its manufacture, dramatically simplifying the process of assuring trust in that design. Modern FPGAs include bitstream security features that turn the fielded design trust problem into an information security problem, with well-known cryptographic information security solutions. The generic nature of the FPGA base array allows the validation expense to be amortized over all designs targeted to that base array. Even the task of checking design tools is simplified by using non-destructive checks of the FPGA design.

Categories and Subject Descriptors
B.7.1. Integrated Circuits, Types and Design Styles. FPGA

General Terms
Design, Security

Keywords
FPGA, Trusted Design, Cryptography.

1. INTRODUCTION
Since their invention, FPGAs have grown in popularity with designers because they eliminate the lengthy IC manufacturing cycle, thereby shrinking up-front NRE costs and manufacturing-cycle delays. FPGA designers have also exploited the reprogrammable capabilities of SRAM FPGAs, delivering high-performance programmable solutions to demanding signal-processing and computing problems, including cryptographic applications [5]. Designs that address these demanding problems are valuable, either intrinsically in their intellectual property, or due to their deployment in sensitive government applications.

Designers of sensitive government systems wish to use Commercial Off-The-Shelf (COTS) devices to gain access to the latest technology and lowest costs. Increasingly, the manufacturing technology for these devices is spread around the world, giving rise to the concern that a design may be stolen or altered by an adversary during the manufacturing process. Designers would like a guarantee that the design that is manufactured performs exactly the intended function. This problem is related to the test problem, though more complicated because traditional testing does not assume an active adversary, nor does it typically detect added functionality.

DAC 2007, June 4–8, 2007, San Diego, California, USA

2. IC MANUFACTURING FLOW
A significant security drawback of the traditional IC manufacturing flow is that the functionality of the design is exposed during the manufacturing process. Although design and final test can be controlled and assumed to be correct, the entire manufacturing process is open to attack. An adversary may exist in the mask-making company, the wafer fabricator, the packaging company or in any of the shipping facilities in between them. An adversary with access to the manufacturing flow has tremendous opportunity to copy, reverse-engineer and tamper with a design. The challenge of trusted design in an IC design flow is to secure the design through all phases of mask making, wafer fabrication, wafer sort and packaging.

3. FPGA MANUFACTURING FLOW
Figure 1 shows a "manufacturing" flow for an FPGA design with two phases of security concern: during manufacture of the base array and during deployment of the fielded system. The base array manufacturing is the same as the IC manufacturing flow outlined above, and all those steps are all collected in the "Non-Secure Manufacturing" box. The FPGA bitstream can be developed and loaded in a "Secure Design Facility" after the base array is manufactured and tested. The secret programming is never exposed to potential adversaries at the various manufacturing sites. Thus, the design in the FPGA need not be protected against theft or tampering until it is released into a possibly-hostile "Non-Secure Environment" in its deployed system. System-level protections can be applied to thwart adversaries in this phase, though such system-level protections are not part of this paper.

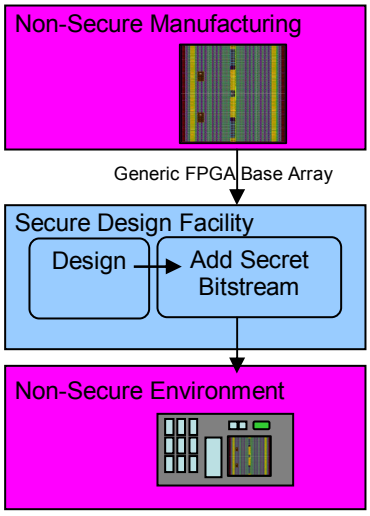

Figure 1. FPGA Component Flow

This separation of manufacturing from

programming is identical to the separation one sees with computers and computer programs. While the computer may be subject to tampering during its manufacture, the program that runs on the computer is not present during the computer's manufacture. Although the computer must still be analyzed for malicious tampering, the program need not be. This tradeoff brings significant advantages. The program is the valuable part of the application. It embodies the critical behavior and may be vastly more complex than the computer on which it executes, so it is important to save time and effort checking the program.

4. SECURING FPGA SYSTEMS IN THE FIELD

Since an SRAM FPGA's bitstream is an electronic message, methods of information security can be applied to assure the integrity and privacy of the bitstream. These methods are well-understood and tested. This section describes methods that have been deployed in FPGAs in sensitive systems.

In this discussion, it is assumed that an adversary has physical access to the FPGA. If the adversary does not have physical access to the FPGA, the problem of FPGA security becomes the solved problem of message authentication and privacy [12] and we can stop now. If the adversary has physical access to the FPGA, denial-of-service attacks on the configuration are irrelevant: a trivial denial-of-service method would be to physically damage the device. Of more concern are unauthorized copy of a design, theft of the design and reverse-engineering.

4.1 Reliance on Bitstream Encoding

FPGA manufacturers do not typically release the *coding* of their bitstream, though they do give a considerable amount of information about the bitstream in tools and in various documents [2]. Despite the availability of this information, there is no report of successful reverse-engineering of an FPGA bitstream. In particular, contrary to one report [5], in the 1990s, according to the principals involved, the NeoCad company reverse-engineered Xilinx's bitstream generation software, not the bitstream itself [4]. Recognizing the complexity of the reverse-engineering task, most FPGA users, like computer programmers, disregard the risk of design theft by reverse-engineering. Of course, both groups suffer from design piracy.

FPGA vendors promote simple, low-cost anti-piracy solutions that may use external devices [6][9]. These solutions increase the difficulty of copying the design but still rely on the difficulty of reverse-engineering the bitstream for their security. These solutions are often considered strong enough for commercial applications, but for security-conscious applications, reliance on the tedium and complexity of bitstream reverse-engineering seems risky.

4.2 Load-Once

A simple method to secure the FPGA bitstream is to program the FPGA once at a secure facility, and retain power to the device constantly in the field [10]. Since all programmable logic devices have privacy settings to prevent readback of the program, and since the bitstream is never exposed outside the device, this method assures that the bitstream running inside the FPGA is both secure and un-modified. This is precisely the same level of security achieved by anti-fuse and other non-volatile FPGAs. The drawback of this method is, of course, the requirement that the system be powered constantly.

4.3 Bitstream Encryption

In 2001, Xilinx introduced bitstream encryption in Virtex-II devices [8][7][11]. Since that time, many FPGA vendors have added the capability to their high-end devices. Although it thwarts reverse-engineering, the goal of bitstream encryption is primarily to prevent unauthorized copy. Large FPGA designs can contain significant intellectual property, and bitstream encryption prevents a competitor from simply copying that intellectual property. The feature also can provide trust assurance by limiting access to the FPGA only to designs constructed with the proper key.

The security implementation and protocols are critical to the security of the design in the FGPA. This section describes the security protocols used in the Virtex family of FPGA.

Virtex FPGAs have a dedicated on-chip decryptor and dedicated key storage memory. The FPGA bitstream contains unencrypted commands for startup and encrypted configuration data [2].

Internal logic in the FPGA restricts access to configuration and key data when bitstream encryption is used in order to eliminate potential security problems. Some of the restrictions are:

- When loading an encrypted bitstream, only a single, full-chip configuration is permitted. Configuration must start at frame 0. Partial configuration and re-configuration are disabled. To re-program, one must shut down the power or issue the JTAG JPROGRAM command, to clear all configuration and data.
- After loading an encrypted bitstream, readback is disabled. This is not a bitstream option, but part of the configuration control.
- An attempt read or write the key data clears all keys and all configuration data.
- The decrypted bitstream must pass data integrity checks before it starts operating.
- The decryptor is not available for encrypting or decrypting user's data after configuration

The security restrictions limit the usable modes of the FPGA, but also eliminate simple attacks on the bitstream. As a result, an adversary may instead attempt to steal a key from the FPGA. The Virtex keys are stored in battery-backed RAM, which vanish when power is cut. To steal the key, an adversary would need to de-cap the FPGA and mill away many levels of metal, then scan the bits with a SEM. This attack must be performed while keeping clean power to the key memory. This is the type of attack required to extract the entire configuration directly from the FPGA SRAM cells, so no bitstream encryption method is qualitatively stronger. This attack is considered to be beyond the capabilities of most adversaries.

4.4 Checking Designs in the Field

Virtex-4 devices introduced the Internal Configuration Access Port (ICAP) (figure 2). ICAP permits logic inside the FPGA to read and write its own bitstream enabling self-reconfiguration for programming and self-test. Potential applications include bitstream decompression, authentication and decryption of bitstreams with a custom encryption algorithm.

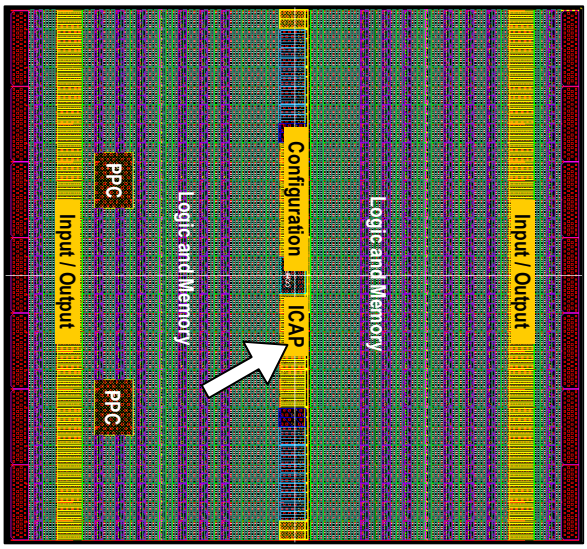

Figure 2. Virtex-4 FPGA with Internal Configuration Access Port

Jones [1] describes the SEU Controller, an application in which the FPGA reads its own bitstream internally to check and correct configuration errors. The SEU Controller is intended to detect and correct errors in a high-reliability environment, but it can be used to detect tampering with the FPGA in the field.

5. SECURING THE BASE ARRAY

In the FPGA flow, the FPGA base array is still manufactured using a standard IC manufacturing flow, and is still subject to attacks during manufacture. Since the sensitive design is not present during manufacture, it cannot be stolen. However, an adversary may attempt to tamper with the base array in a way that will affect the final design after it is loaded into the FPGA. An adversary cannot directly attack the actual design, because he doesn't know where to insert malicious logic, so the attack is a probabilistic one, attempting to improve the chances that the configured design will be affected by the change.

A designer can use clever design to make the adversary's problem arbitrarily difficult with arbitrarily low probability of success. For example, a designer may implement critical parts of the design with triple modular redundancy (TMR). Since each part is placed in a different part of the FPGA, an adversary would need to construct a modification of the base array that would affect two of the three modules implemented in FPGA fabric in the same way.

A potentially more-fruitful attack would be to attack the FPGA's security features, with the expectation that the weakness in the FPGAs security can be exploited by the system in the field. Defeating this potential attack requires that the bitstream security functions be fully verified. This may sound difficult, but the bitstream security functions comprise a very small fraction of the overall device, so that verification task is much simpler than the task of verifying an entire chip. Further, since a single trusted FPGA base array can be used for any number of trusted designs, the cost of verification of a single FPGA base array can be amortized over all trusted designs built on that base array.

COTS FPGAs have further advantages in manufacturing. Since an adversary doesn't know which devices to attack, potential tampering affects all FPGAs. Most FPGAs are used in commercial applications, so there are thousands of non-secure uses that could potentially expose the tampering. IC manufacturers are fastidious about the quality of their product. Any returned devices are carefully inspected for defects. Deliberate tampering could be detected in returned commercial devices.

In addition, since the manufacturing volume of FPGAs is large, large numbers of commercial FPGAs can be (destructively) tested to search for inserted logic or environmental sensitivities. Again, the cost of this testing can be amortized over all designs that use the same base array.

6. DETECTING TAMPERING IN TOOLS

Thompson [3] gives an insightful description of the subtlety with which an expert can tamper with tools. As Thompson showed, an adversary could add additional functionality to expose sensitive information, or provide unauthorized access to stored data. A tool that modifies or eliminates intended functionality will be detected by testing. But, classical testing only verifies the presence of wanted logic. It does not verify the absence of unwanted logic.

In order to properly close the loop on tools and manufacturing, one must formally compare the manufactured design to the original. In an ASIC model, this can be done by reverse-engineering the manufactured design: de-capping the part, and repeatedly photographing the die and stripping away layers until all the physical layers are reproduced. Then one extracts the transistor network from the physical layout and performs a network equivalence check of the extracted network against the original design. Many die can be analyzed to gain (statistical) assurance that the remainder can also be trusted. Not only is this process time-consuming, but the process is necessarily destructive. There is no guarantee that the fielded system is identical to the tested one.

In contrast, FPGA designs can be checked non-destructively, so equivalence between the implemented design and the original design guarantees correctness of the operating design. This check is also relatively simple and quick. The FPGA design system retains logical information through physical design, so the final implementation can be compared to the original design specification without complex reverse-engineering. Layout-versus-schematic (LVS) comparison tools compare two netlists,

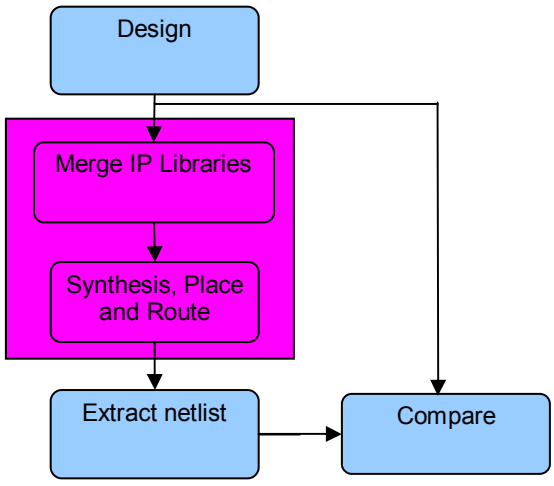

Figure 3. FPGA Design Validation Flow

and are commonly used to compare a schematic with an extracted manually-generated layout. LVS tools not only identify missing logic, but also identify added logic and added connections that would be expected from a clever adversary. They can be employed to compare the original design to the implemented FPGA design to identify unwanted logic added by tools or libraries.

7. Conclusion

The FPGA design and implementation flow allows us to separate the manufacturing process from the design process. In the FPGA flow, the sensitive design is not exposed to theft and tampering through the manufacturing process. Only the base array must be verified through manufacture. That verification can be significantly simplified because the adversary does not know the design to attack. Systemic advantages also accrue from the broad commercial uses of the same base array that may be deployed in sensitive applications. FPGA flows permit non-destructive detection of tampering inserted by tools. The security of the FPGA design during deployment is assured by an application of well-known information-security methods.

8. References

[1] Jones, L., "Single Event Upset (SEU) Detection and Correction Using Virtex-4 Devices", Xilinx Application Note #714, 2007, http://www.xilinx.com/bvdocs/appnotes/xapp714.pdf

[2] Xilinx Virtex-4 Configuration Users Guide, v1.5, UG 071 2007, http://www.xilinx.com/bvdocs/userguides/ug071.pdf

[3] Thompson, K., "Reflections on Trusting Trust", *Communications of the ACM*, Vol. 27, No. 8, August 1984, http://www.acm.org/classics/sep95/

[4] Bennett, D., Private communication.

[5] Wollinger, T. and Parr, C. "How Secure are FPGAs in Cryptographic Applications", *13th International Conference on Field Programmable Logic and Applications, FPL 2003*, P.Y.K. Cheung, G.A. Constantinides, J.T.de Sousa, eds., LNCS 2887, Springer, 2003.

[6] Feng, J. and Seely, J.A., "Design Security with Waveforms", http://www.altera.com/literature/cp/cp_sdr_design_security.pdf

[7] Lesea, A., "IP Security in FPGAs", Xilinx http://direct.xilinx.com/bvdocs/whitepapers/wp261.pdf

[8] Teliknepalli, A., "Is Your FPGA Design Secure?", XCell Journal, 2003, http://www.xilinx.com/publications/xcellonline/xcell_47/xc_pdf/xc_secure47.pdf

[9] Baetoniu, C. and Sheth S., "FPGA IFF Copy Protection Using Dallas Semiconductor/Maxim DS2432 Secure EEPROMs", Xilinx Application Note XAPP 780, Xilinx 2005.

[10] Trimberger, S., *FPGA Technology,* Kluwer Academic Press, 1994.

[11] Trimberger, S. "Method and apparatus for protecting proprietary configuration data for programmable logic devices", US Patent 6654889, 2003.

[12] Schneier, B., *Applied Cryptography Second Edition*, Wiley, 1996.