Periodic Licensing of FPGA Based Intellectual Property

Nathaniel Couture [1], Kenneth B. Kent [2]

Faculty of Computer Science, University of New Brunswick
Fredericton, New Brunswick, Canada
[1]l394e@unb.ca
[2]ken@unb.ca

***Abstract*— This work describes a method of licensing IP on FPGAs based on techniques derived from software licensing schemes. Current software and hardware licensing techniques are described in detail, including a survey of current research in the fields of FPGA security, secure memory technologies, and cryptography. A licensing architecture for FPGA IP is proposed, and an implementation on a Xilinx Vertex 2 FPGA demonstrates that expiration of FPGA based IP can be achieved. Future work includes the development of a hardware architecture for consumer products that supports licensable IP cores as well as their delivery.**

I. Introduction

Profits generated from the sale of IP targeting FPGAs are currently made from single sales to large manufacturing companies. These sales, typically closed under a legal agreement, make the IP available to the customer in an unrestricted fashion [1]. Under this type of agreement, the customer may reuse the IP as many times as desired for an unspecified amount of time. This prevents any future sale of the IP to this client, with the exception of new product releases. This situation is undesirable for two reasons. The price of the IP must be set prohibitively high, limiting sales, and the IP vendor/creator loses control over the use of its IP.

This research investigates the feasibility of physically enforced periodic licensing of FPGA IP. The goal is to design a hardware architecture to support licensable FPGA IP cores targeting consumer products [2]. This situation would allow the creators of FPGA IP to profit from individual sales of their IP to consumers, rather than limited sales to manufacturers under a Sign Once agreement [1]. The goal of this research is to develop a proof of concept prototype that demonstrates how FPGA IP can be made to expire based on license enforcement techniques taken from software. By demonstrating that custom IP can be made to expire while in use on an FPGA, future work can be done in order to achieve a standardized hardware architecture that will support licensable IP cores. The scope of this research is limited to the creation of a secure architecture for circuit expiration but does not address the delivery of this technology to consumers.

II. Related Work

A license enforcement scheme is a method of ensuring that IP users are adhering to the laws and policies set out in a licensing agreement. A license agreement, when present, is a document that a user must agree to prior to using an IP stating that the user must use a valid, legally obtained copy of the IP and only use it for a specified period of time. However, users do not necessarily obey licensing agreements [3]. Enforcing licensing agreements often requires a combination of techniques. Some software license enforcement techniques include: watermarking and/or fingerprinting for fraud identification, code partitioning, software/hardware tokens and dynamic code decryption [4], [5], [6], [3].

A. Hardware License Enforcement

Current hardware licensing techniques are less varied than those used by the software industry. This is due to the expensive nature of reverse engineering hardware designs in comparison to software designs [7]. Similar to software licensing schemes, the licensing of hardware IP often involves a license agreement combined with a separate component used for physical protection of the design. Using such a combination provides a more robust and secure licensing scheme [8].

Kean suggests a pay-per-use licensing scheme where IP blocks can be purchased for one-time use only [9]. Taking advantage of on-chip cryptography the IP can be transferred to the FPGA chip securely from a trusted third-party for use in a single design without providing the source code for the IP. The majority of the techniques used to protect IP on FPGAs are derived from this technique, and use cryptography in various ways to protect the IP. By encrypting the designs, they can be downloaded and decrypted inside the FPGA with the goal of preventing source code exposure to malicious users [10], [8].

The majority of FPGA IP vendors are preventing illegitimate use of the IP by implementing generic legally binding contracts between the vendors and the customers. An example is the Xilinx SignOnce initiative [1]. Under this agreement the customer pays once for the IP, and can use it with few restrictions. Although this provides a means to profit from FPGA IP, there is nothing in place to physically protect the IP itself. In addition, this licensing scheme makes IP very expensive [9] since an IP vendor can only profit once from each client. To support the SignOnce contract, watermarking and fingerprinting techniques are used in the identification of theft, but can only be loosely enforced worldwide [9].

B. FPGA Security

A licensing scheme enforced within a design is only as secure as its environment. In order for the proposed licensing architecture to be valid it must be shown to be secure. The security attacks on FPGA designs can be grouped into 3 classes of attacks: Class 1, Class 2, and Class 3 [11]. The majority of attacks, Class 1, come from clever outsiders with negligible resources. Class 2 attacks include industry insiders with access to some sophisticated resources, such as university students. The information discovered by this class is very technical and only usable by the same class of attacker or higher. Class 3 attacks are funded organizations with a determined team of experts. These groups can often crack anything, e.g. FBI, CIA, NSA, and any other large commercial or government organization [11].

A secure FPGA design must thwart attacks from Classes 1 and 2 as well as pose a challenge to Class 3. Physical attacks using highly sophisticated probes to dismantle and reverse engineer designs will not be considered for the purpose of this paper since it is far too costly for typical users. The remaining attacks include: Black-Box attacks [12], [13], bitstream reverse engineering [12], [13], read-back attacks [12], physical attacks [12], [14], design cloning[12], [13], power analysis attacks [15], etc.

C. Physical Uncloneable Functions

Physical Uncloneable Functions or Physical Random Functions, known by the acronym PUF, can be used to prevent unauthorized access to a physical device, i.e. memory. A PUF is defined as a function that maps challenges to responses, is embodied by a physical device and has the following two properties[16]: easy evaluation and hard to characterize.

Provided an IC is equipped with a secret key k, a pseudo-random hash function h, and k is impossible to extract from the IC, the function $x \rightarrow h(k,x)$ is a Digital PUF. If control logic is added to disable/erase the IC when tampered with then it is called a Controlled Physical Random Function or CPUF. The security of these CPUFs has been shown to be weak in defending against physical tampering that would reveal the key. This would allow the device to be cloned, which is highly undesirable [16]. The Silicon PUF (SPUF) is a more secure alternative and exploits the statistical variations in the delays of devices within the IC [16].

III. Proposed Solution

The proposed solution is based on the token approach used in software products [3]. For this, a token is used to store the licensing information: life span/expiration data, usage counter and site information. Under this model, if the IP does not have access to a valid token, the product only functions partially or not at all. However, a different environment means the supporting licensing architecture will require different security mechanisms for storing a token, disabling the IP, tracking the usage (life span) and preventing successful attacks on the architecture.

One of the challenges of using a token based licensing scheme is keeping the token secure. The other challenge is disabling the user design when the life of the license has ended. A high level view of the proposed solution is shown in Figure 1. In this figure, the licensed IP is shown as **User Design**. The **Licensing Component** contains the logic to perform the expiration of the connected IP.

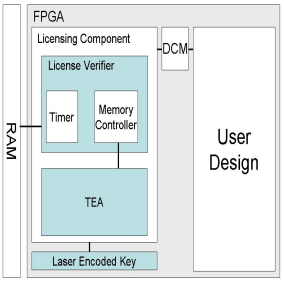

Fig. 1. Licensing Architecture Components

To address the token storage, a secure memory element must be added to store the token. This can be a PUF secured external non-volatile memory, or an internal non-volatile memory. Both will provide the storage required to maintain the licensing information of the token, and keep it inaccessible.

In addition to the memory, an encryption engine must be used to encrypt and decrypt the licensing information prior to storing it. The secure memory allows the device to power off/on as desired without losing the licensing information, while the encryption prevents the data from being compromised in the event that an attack managed to penetrate the secure memory.

Furthermore, an additional feature must be added to the FPGA itself. This feature is a laser encoded key inside the FPGA fabric in order to uniquely identify the chip [9]. This is used to prevent a malicious user from cloning the FPGA configuration in order to use a licensed product on another FPGA.

IV. Licensing Architecture

The expiration of the user design depends on a single component, the licensing component, as shown in Figure 1. This component encompasses several parts: License Controller, Real Time Timer, Non-Volatile Secure Memory, and the Encryption Component.

1) License Controller: The license controller is the engine of the licensing architecture. This component has four primary tasks: i) update the usage parameter in the license token; ii) verify the validity of the license; iii) verify the laser encoded key; and iv) expire the licensed IP. The overall goal of this component is to expire an IP core after a period of time has elapsed. For this, it relies on the timer component to keep track of the elapsed time. At regular intervals, the elapsed time is stored in non-volatile storage. As a result, sudden power outages, which are common in battery powered electronics, will not cause the licensing component to lose track of the

elapsed time. When the device is powered on, the value is retrieved from the non-volatile storage and is used to initialize the timer. The license controller therefore acts as a controller for the various reads/writes to/from memory. Additionally, it uses the encryption engine to encrypt the data prior to performing a write, and decrypt the data after performing a read.

To prevent the use of illegitimate licensing components the licensing architecture is designed to target a single unique FPGA. Therefore, the FPGA must be uniquely identifiable. For this, a laser encoded key is embedded at manufacture time into the FPGA [9]. When the design is synthesized it contains within it the value of the laser encoded key. This value must match the actual laser encoded key throughout the duration of the license. If at any point in time, the keys do not match - indicating IP theft, the design is expired immediately.

This component also ensures that the license is valid by verifying the time elapsed has not exceeded what is specified in the license token. The licensing controller will immediately expire the licensed IP if any of the following conditions is met: the elapsed time exceeds the licensing period, the laser encoded key does not match the design's hard-coded key, or the token is tampered with and has been detected.

2) Timer: For timed expiration to take place, a means to measure the usage of the device must be in place. For this, there are two possibilities: the licensing component can keep track of the real time elapsed since the device was programmed or track another usage parameter, such as actual run time or a usage counter. Applicability of either scheme is dependent upon the IP under license.

The timer must continue counting time using a previously stored value, which will be read in from secure memory. It will need to provide the current time in order for it to be stored continuously in the event of a power failure in which the licensing component would be required to provide up to date current time to the licensing controller upon future power-up.

3) Non-Volatile Memory: Using existing technology we must assume that the only non-volatile storage available to the FPGA must be external to the FPGA fabric. Therefore we must ensure that the memory containing our licensing token can not be duplicated. The data within the component will be encrypted using a secure encryption algorithm thus removing any doubt that the data itself will be modified. Once the data is encrypted, it is also necessary to prevent its duplication, as this would allow a user to bypass the memory with a copy made when the license was new. Silicon PUFs, allow us to uniquely identify our memory such that it cannot be reproduced. Using this type of PUF, the memory controller can run several challenge-response pairs and verify that this memory is indeed the original valid memory [16].

4) Encryption Unit: The encryption component is required unless internal non-volatile storage is provided inside the FPGA, currently unavailable in FPGA technologies. Thus, the ability to protect data stored external to the FPGA is required. A solution to this problem would be to implement an encryption component that would encrypt/decrypt data using a secure cipher for the purpose of protecting sensitive data such as the license token and the elapsed time.

By encrypting the time values prior to storing them, the goal is to prevent theft/tampering of the overall system via token data modification. Using a publicly known algorithm such as AES, RC5 or TEA, should provide adequate security since they have been widely studied. As shown in [13] a version of TEA will provide security at a relatively low cost in terms of FPGA design space. The amount of hardware space or CLBs taken could be adjusted by changing the length of the key, the number of rounds and/or the amount of data being encrypted [13].

A. Additional Design Issues

Keys are required for both the encryption engine and the unique chip identifier. Encryption key management is rather simple. There is no need for the encryption key to be known outside the FPGA. Prior to synthesis, the design will have a random key embedded into the design. During the very first power-on, the token data will be encrypted and stored using this new key. During every subsequent memory action, the same key will be used to encrypt and decrypt the data; however this key will remain as part of the original FPGA bitstream design which will also be encrypted to avoid reverse engineering. [18]

The unique chip identifier is a laser encoded key as presented by [9]. This key is very difficult to modify and duplicate [10]. Once the device is manufactured and is encoded with a unique identifier, this identifier will be shipped with the device. When the "programmer" of the device wants to target a licensable design to this device it simply embeds this key into the design. The design will verify throughout operation that the embedded code matches the chip identifier.

A technique to expire a license is to take over the clock signal and prevent it from powering the user design. A typical design contains one or more DCMs providing clock inputs to various components within a design, therefore, controlling the state of the main DCM of a user design can create the effect of an expired circuit. By putting the DCM in an idle state, thus not providing a main clock signal to the user design, the design will not function. Controlling the expiration through the DCM will allow a proprietary design/core, to which the source code would not be given, to be used.

V. Prototype Implementation

This section describes the implementation of the licensing architecture implemented on a Xilinx Vertex II (XC2V2000) FPGA. The fact that the proposed design relies on new technology, unavailable in commercial format (research stage only), implies that the prototype built as a proof of concept must be modified to work, using commercially available equipment. The resulting adapted prototype demonstrates that the concept works but will not provide all of the security features of the original design. The missing components are

the non-volatile secure memory with built in PUF, and the laser encoded key.

The license controller is the heart of the licensing architecture, and as such acts as the overall system controller for all of the components. It makes use of an instance of the timer component by loading it with the value for elapsed time stored in memory. It also ensures that the license is valid by verifying that the time elapsed has not exceeded what is specified in the license token, and also verifies that the laser encoded key matches the hard coded key. It also ensures that the current time is stored in memory regularly. Finally, it outputs the state of the license which enables or disables the user design.

The timer is responsible for keeping track of real time inside the FPGA. The timer outputs the current time count constantly in order for the licensing controller component to have the exact time at any given moment. This value is used by the licensing component to determine if the license is expired. The timer is also able to start up at a predetermined value to allow stopping the device and restarting it (power off/power on). The current time is stored in memory in short intervals. When the device re-loads, the time stored in memory will be loaded, and used as the starting point.

The memory used in the prototype is a block RAM internal to the FPGA. Ideally, the internal RAM would be non-volatile. However, the design specification also suggests the use of an external non-volatile memory with PUF functionality.

The encryption component is an implementation of the TEA encryption algorithm. An instance of this component is used by the memory controller in order to encrypt and decrypt data before/after reading/writing to/from memory. The design space using 128 bit key and 64 rounds is 5% of the entire chip.

The full licensing component has only two I/O pins, the *clock*, and the *license valid* signal. The valid signal output connects to the DCM, which will toggle the DCM from a working state to a non-working (idle) state in order to expire a user design. Any user design can connect to the outgoing clocks on the DCM in order to be made licensable.

To validate the design, a System-on-Chip implementation of an MP3 decoder is used [19]. The core of this design is the Xilinx Microblaze (MB) soft core processor [18]. The MP3 decoder has multiple components: Discrete Cosine Transform, Fast Fourier, Multipliers and many other small peripherals. Once the licensing core is added to the MP3 player project, it can be connected to the DCMs used in the user design. The MP3 player has two DCMs, a primary DCM to power the MB processor and one to power the Zero Bus Turnaround (ZBT) Ram. The main DCM powers many of the peripherals and cores attached to the user design. By connecting the valid signal from the licensing component to the external reset signal of the main DCM, the licensing component can expire the MP3 player by denying its main clock signals.

VI. Conclusions and Future Work

Periodic licensing of IP targeting FPGAs was achieved and demonstrates the expiration of an mp3 player [19]. The prototype lacks many of the features required in order to provide a secure means of licensing IP. However, by implementing the proposed architecture on a secure FPGA [20], the concept could become a very useful component in the creation of consumer electronics that allow custom hardware to be sold directly to consumers. By providing features such as internal non-volatile storage and laser encoded keys in next generation FPGAs, FPGA vendors would make it possible for this architecture to be developed commercially. Ideally, it would be beneficial to provide this licensing technology as firmware built into the FPGA fabric.

References

[1] Xilinx. Common license consortium for intellectual property. [Online]. Available: www.xilinx.com/ipcenter/signonce.htm

[2] N. Couture, "Self-expiring licensing architecture for intellectual property on fpgas," Master's thesis, University of New Brunswick, New Brunswick, Canada, September 2005.

[3] P. Devanbu and S. Stubblebine, "Software engineering for security: a roadmap," Future of Software Engineering, Special volume published in conjunction with ICSE 2000, Limerick, Ireland, 2002.

[4] J. Lach, W. Mangione-Smith, and M. Potkonjak, "Fpga fingerprinting techniques for protecting intellectual property," in *Custom Integrated Circuits Conference (CICC'98)*, 1998.

[5] A. Corporation, "Protecting your intelectual property from the pirates." [Online]. Available: www.actel.com

[6] A. K. Jain, L. Yuan, P. R. Pari, and G. Qu, "Zero overhead watermarking technique for fpga designs," in *Great Lakes Symposium on VLSI 2003(GLSVLSI'03)*, Washington, DC, April 2003, pp. 147–152.

[7] M. Dalpasso, A. Bogliolo, and L. Benini, "Hardware/software ip protection," in *Design Automation Conference 2000*, Los Angeles, CA, 2000, pp. 593–596.

[8] T. Kean, "Cryptographic rights management of fpga intellectual property cores," *Field Programmable Gate Array 2002*, February 2002.

[9] ——, *Cryptographically Enforced Pay-Per-Use Licensing of FPGA Design Intellectual Property*, Edinburgh, UK, 2002.

[10] ——, "Secure configuration of field programmable gate arrays," in *Lecture Notes in Computer Science. Proceedings of the 11th International Conference on Field-Programmable Logic and Applications*. London, UK: Springer-Verlag, 2001, pp. 142–151.

[11] D. G. Abraham, G. M. Dolan, G. P. Double, and J. V. Stevens, "Transaction security system," *IBM Systems Journal*, vol. 30, no. 2, pp. 206–229, 1991.

[12] T. Wollinger and C. Paar, "How secure are fpgas in cryptographic applications," in *International Association for Cryptologic Research, Cryptology ePrint Archive*, Limerick, Ireland, June 2003.

[13] T. Wollinger, J. Guajardo, and C. Paar, "Cryptography on fpgas: State of the art implementations and attacks," *ACM Transactions in Embedded Computing Systems (TECS)*, 2004, special Issue on Embedded Systems and Security.

[14] Y. Ishai, A. Sahai, and D. Wagner, "Private circuits: Securing hardware against probing attacks," in *Lecture Notes in Computer Science: Advances in Cryptology - CRYPTO 2003*, vol. 2729. Springer Berlin / Heidelberg, 2003, pp. 463–481.

[15] F. Standaert, L. van Oldeneel tot Oldenzeel, D.Samyde, and J. Quisquater, "Differential power analysis of fpgas : How practical is the attack?" in *Lecture Notes in Computer Science: FPL 2003*, vol. 2778.

[16] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Controlled physical random functions," in *18th Annual Computer Security Applications Conference (ACSAC '02)*, Las Vegas, Nevada, December 2002, p. 149.

[17] Y. Ko, S. Hong, W. Lee, S. Lee, and J. Lim, "Related key differential attacks on 26 rounds of xtea and full rounds of gost," in *Fast Software Encryption Workshop '04*. Springer-Verlag, 2004.

[18] Xilinx. Xilinx: The programmable logic company. [Online]. Available: www.xilinx.com/

[19] U. of Toronto, "Soc mp3 decoder source code." [Online]. Available: www.eecg.toronto.edu

[20] L. S. Corporation. Fpga design security issues: Using the ispxpga family of fpgas to achieve high design security. [Online]. Available: www.latticesemi.com