HAL
open science

Two IP Protection Schemes for Multi-FPGA Systems

Lubos Gaspar, Viktor Fischer, Tim Guneysu, Zouha Cherif

To cite this version:

Lubos Gaspar, Viktor Fischer, Tim Guneysu, Zouha Cherif. Two IP Protection Schemes for Multi-FPGA Systems. ReConFig' 12, Dec 2012, cancun, Mexico. pp.2568809. ujm-00763142

HAL Id: ujm-00763142

https://ujm.hal.science/ujm-00763142v1

Submitted on 10 Dec 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Two IP Protection Schemes for Multi-FPGA Systems

Lubos GASPAR, Viktor FISCHER
University of Lyon
Laboratoire Hubert Curien, CNRS
42000, Saint-Etienne, France
Email: {lubos.gaspar, fischer}@univ-st-etienne.fr

Tim GÜNEYSU
Horst Görtz Institute for IT Security
Ruhr University Bochum
44780, Bochum, Germany
Email: gueneysu@crypto.rub.de

Zouha CHERIF JOUINI
TELECOM ParisTech
CNRS, LTCI
75634, Paris, France
Email: cherif@telecom-paristech.fr

***Abstract*—This paper proposes two novel protection schemes for multi-FPGA systems providing high security of IP designs licensed by IP vendors to system integrators and installed remotely in a hostile environment. In the first scheme, these useful properties are achieved by storing two different configuration keys inside an FPGA, while in the second scheme, they are obtained using a hardware white-box cipher for creating a trusted environment. Thanks to the proposed principles, FPGA configurations coming from different IP owners cannot be cloned or reverse-engineered by any involved party, including system integrator and other IP owners. The proposed schemes can be directly implemented in recent FPGAs such as Xilinx Spartan 6 and Virtex 6.**

***Keywords*-IP protection schemes; white-box cipher; licensing;**

I. INTRODUCTION

The advent of Field Programmable Gate Arrays (FPGA) started a new era in the domain of integrated circuits. The high demand for common hardware functions (e.g., complex data processing or cryptographic functions) led to creation of a new Intellectual Property (IP) market, where IP Owners (further IPOwner) could distribute and license their IP designs to system integrators. This raised a question about the security of IP designs in SRAM-based FPGAs.

To prevent an adversary from cloning FPGA configurations, many FPGAs embed a circuit to support configuration bitstream encryption. This way, only the system integrator (further Integrator) with access to the encryption key can access/modify the configuration bitstream. However, if an IPOwner licenses an IP to the Integrator, both parties need to protect their secret from each other as well as from external adversaries. Unfortunately, current FPGAs provide no hardware means to protect IP designs from reverse-engineering or duplication by Integrators.

Previous Work: The easiest way for implementing an IP function as a part of a larger FPGA design is by importing it to an EDA tool and synthesizing the project as a whole. The IP can be protected by a private key, but this key must be present in the EDA tool and can be thus recovered if the tool is disassembled [1].

An interesting solution is proposed in the SeReCon system [2], which provides IPs protection, even if the FPGA configuration is stored or changed remotely. However, the SeReCon system requires a trusted authority for certifying its public key and the party that installs the SeReCon system (Integrator) must be trusted. Another protection scheme based on an FPGA personalization module can effectively protect the IP installed in a remote FPGA environment [3], but it necessitates some features that are not available in current FPGAs. Clearly, no satisfactory solution exists to protect IP designs even in the most recent FPGA devices.

Our Contribution: In this paper[1], we propose two novel protection schemes for IP configuration bitstreams implemented on multi-FPGA systems. The scheme provides IP vendors with means to license each IP bitstream on a per-FPGA basis without a need for additional hardware components or modifications of recent FPGA technology. Each FPGA can contain one IP function.

The two schemes are aimed at an automotive market, where car manufacturers (Integrators) are assumed to buy IP bitstreams from third-parties (IPOwners) and install them in car functional units, each containing one FPGA. The ability to upload the system integrator's bitstream prior to uploading the IP bitstreams facilitates remote updates and integrity verifications of the IP bitstreams by IPOwners. This new security structure prevents the Integrator and other IPOwners from reverse-engineering the IP bitstream or installing it on more FPGAs than licensed. Moreover, the protocols support a shared secret to be established in all FPGAs, in order to make further cryptographic services, such as authenticated key exchange protocols and confidential inter-chip communication available. The new protection scheme is suitable for both low-cost and high-end FPGA devices.

Outline: The paper is organized as follows. In Section II we provide some theoretical background. Section III presents two novel protection schemes for IP designs in multi-FPGA systems. The implementation details are provided in Section IV. The results from Section V are discussed in Sec. VI. Finally, Section VII concludes the paper.

[1]The work presented in this paper was realized in the frame of the SecReSoC project number ANR-09-SEGI-013, supported by the French National Research Agency (ANR).

II. Theoretical background

In this section, we introduce the parties participating in the system and explain relevant terms and system components.

A. Participating Parties

The protection scheme involves two parties: the integrator and the group of IPOwners. The integrator designs a multi-FPGA system (the System). The System is composed of *N* application-specific nodes. Each node comprises one volatile FPGA device, a non-volatile configuration memory (ConfMem) and other auxiliary units. We denote the FPGA and ConfMem belonging to an $\mathrm{i-th}$ node as $\mathrm{FPGA_i}$ and $\mathrm{ConfMem_i}$, respectively.

The IPOwners offer their logic designs aimed at specific applications. An $\mathrm{IP_i}$, designed by the $\mathrm{IPOwner_i}$, is synthesized as a configuration bit file for the $\mathrm{FPGA_i}$. The intention of the $\mathrm{IPOwner_i}$ is to distribute its $\mathrm{IP_i}$ configuration file using a node-locked licensing model. This way, each IPOwner can control the number of Systems in which its IP is installed, and thus prevent the Integrator or third parties from cloning or reverse-engineering the IP (see [4]).

B. Required Cryptographic Algorithms

The following components are essential for achieving protected FPGA designs.

1) Symmetric Ciphers: To prevent the IP theft, each IP configuration bitstream needs to be encrypted. The AES standard [5], is a commonly adopted solution for IP bitstream protection.

Another symmetric block cipher that we consider is Noekeon [6], because it can be very efficiently decomposed to small look-up tables (LUTs).

2) Message Authentication Codes: The Message Authentication Code (MAC) is usually based on block cipher algorithms (i. e. CMAC [7]) or cryptographic hash functions (i. e. HMAC [8]). The MAC functions can be used for an IP bitstream authentication.

3) White-Box Cryptography: The goal of the white-box cryptography is to embed and protect secret keys inside a block cipher while all its internal states can be inspected by an attacker. This goal is reached by introducing a high level obscurity, which is achieved by representing the cipher as a very complex network of LUTs. These LUTs incorporate external encoding bijections to hide the structure of the cipher and the secret key. The white-box cipher with an embedded key represents a one-way function, which transforms input plain-text data into output cipher-text data. White-box implementations together with security analyses can be found in [9]. Even though some theoretical attacks were proposed, they require a lot of computational power and time. In our solution, the white-box cryptography is used to protect IPOwners from cloning and reverse engineering by Integrator and not by external thefts. This lowers considerably security requirements. Unfortunately, white-box AES implementations are too expensive on FPGAs [10]. Unlike AES, the straightforward decomposition of the Noekeon cipher to small LUTs facilitate the implementation of external encoding bijections. The mapping of these small LUTs into an FPGA results in a very compact white-box solution [11].

C. FPGA Features

1) Bitstream Protection: Effective protection of private designs can be achieved only if their bitstreams are encrypted when stored in a Configuration Memory. During configuration of the FPGA, bitstreams are decrypted by a dedicated hardwired decipher unit using secret keys, which are stored in the device in a volatile or non-volatile key memory (KeyRAM or KeyROM, respectively).

Some high-end FPGAs also support the bitstream authentication for tamper detection. But this cannot prevent attackers from cloning a device configuration. To solve this issue, it is essential to bind an IP design to a unique FPGA identifier (ID).

2) Multi-Boot Feature: A multi-boot feature allows FPGAs to switch between several configuration bitstreams stored in the configuration memory. After power-up, the FPGA loads an initial configuration (e.g., provided by the Integrator) stored in the beginning of the configuration memory. This initial configuration can select other configurations (IP modules) stored in the memory and start FPGA reconfiguration with the selected bitstream. If reconfiguration fails, a fall-back bitstream can be loaded instead.

3) Partial Reconfiguration: This technology allows to modify a part of the FPGA fabric while the rest (a static part) stays intact and continues working. If a partial bitstream is supplied using an internal reconfiguration port, it can be protected by hardwired FPGA cryptographic units. Otherwise, a decryption unit aimed at the partial bitstream decryption must be implemented in the static area.

D. Prerequisites and Assumptions

In our protection scheme, we assume the following prerequisites.

P1: Trusted and Untrusted Parties: We assume that the FPGA hardware manufacturer is a trusted party. It is his intention to support protected FPGA configurations and he will not share any secret or other critical information with other parties. However, all other parties are regarded as untrusted and may try to cheat.

P2: Communication Secrecy: The communication between the Integrator and any IPOwner is assumed to be secure against any type of attack. On the contrary, the remote communication between the Integrator and the System is not secured and requires an additional protection layer. The communication buses between different System nodes and each $\mathrm{ConfMem_i}$ and $\mathrm{FPGA_i}$ are unprotected.

P3: Cryptographic Algorithms: We assume that all cryptographic algorithms are computationally secure, such that the secret key cannot be recovered using practical amount of computational power in a reasonable amount of time. Moreover, all implementations used in the protection scheme are assumed to be fault-tolerant and sufficiently secure against side-channel attacks even over multiple executions.

P4: Security Platform: Protection is offered for two different FPGA platforms. For the low-cost FPGAs, system cost and security of third-party IPs are of utmost importance. For the high-end FPGAs, the flexibility of the system solution as well as the protection of third-party IPs is primarily considered.

For our protocols, all FPGAs must feature a bitstream decryption unit and both volatile and non-volatile key memories. It must be also possible to read a unique device ID from inside the FPGA. The multiboot feature is also essential for our system. Moreover, two bitstreams located in the configuration memory can be protected using two different keys. For example, one key may be stored in the KeyRAM and the other in KeyROM. For this reason, it must be possible to specify the key placement in the bitstream header. If the configuration fails, the FPGA must be able to load a fall-back bitstream. Note that all these features are supported by the low-cost Xilinx Spartan-6 family starting from the LX75/T device.

In addition to the requirements mentioned above, high-end FPGAs must support the partial reconfiguration technology and bitstream authentication. The Xilinx Virtex-6 is very convenient for this application.

III. IP Protection Schemes

The goal of the protection scheme is to enable the secure integration of IP designs, potentially from different IPOwners, in different system nodes. Since only two configuration keys can be stored in an FPGA at a time, we assume only one Integrator bitstream and one IPOwner's bitstream to be used per FPGA. Despite this limitation, IPs in different nodes need to be able to communicate with each other in a secure way. This can be possible only if a shared secret (SystemKey) is distributed by the Integrator to all nodes. Moreover, the Integrator should maintain its control over the system and provide remote configuration updates. This can be possible only if he installs first a proprietary configuration (Initial Configuration) in all FPGAs. The Initial Configuration bitstream must be protected from all participating parties as well as external adversaries. At the same time, it does not allow the Integrator to copy or tamper the IP bitstream present in the same configuration memory.

Fig. 1 illustrates the relationship between the participating parties, organization of the multi-FPGA system and the composition of a system node.

Next, we present two different protection schemes to install and maintain the Initial Configuration and the node-specific IP configurations in each system node.

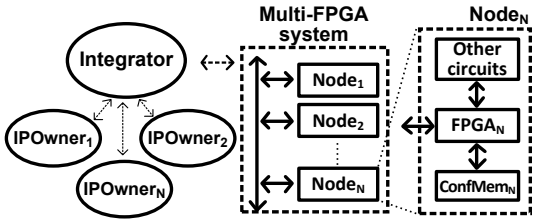

Figure 1. Relationship between parties and structure of the multi-FPGA system

A. IP Protection for Low-Cost Multi-FPGA Systems

The system setup and IP installation protocol are illustrated in Fig. 2. Before the System is assembled, each IPOwner must store its IP bitstream key in the KeyROM and send the FPGA to the Integrator (Steps 1, 2). Alternatively, the trusted FPGA manufacturer can have mutual agreement with an IP Owner and prestore his key in the respective FPGA device. This way the necessary IPOwner secret is present in every FPGA. The Integrator assembles the System, writes his key (IntegratorKey) to every FPGA KeyRAM (Step 3) and stores the Initial Configuration (including the SystemKey) protected by the IntegratorKey to all Configuration Memories (Step 4). All other operations can be performed in an insecure environment.

After power-up, the $\mathrm{FPGA_i}$ loads and decrypts the Initial Configuration (Step 5). This configuration verifies if the $\mathrm{IP_i}$ configuration is present in the $\mathrm{ConfMem_i}$ by searching for the header (SYNC word) and footer (DESYNC word) in the bitstream (Step 6). If confirmed, the Initial Configuration initiates the multiboot and the FPGA loads and deciphers the $\mathrm{IP_i}$ configuration protected by the $\mathrm{IPKey_i}$ (step 12). However, if the $\mathrm{IP_i}$ is not present in the $\mathrm{ConfMem_i}$ or is corrupted, the Initial Configuration establishes a secure communication (protected by the CommunicationKey) with the Integrator and reports the $\mathrm{IP_i}$ issue (Step 7).

The Integrator receives the FPGA $\mathrm{ID_i}$ and SystemKey and can now acquire a license by securely sending both to the $\mathrm{IPOwner_i}$ (step 8). The $\mathrm{IPOwner_i}$ generates an $\mathrm{ID_i}$-locked $\mathrm{IP_i}$ (containing the SystemKey), enciphers it with the $\mathrm{IPKey_i}$ and sends it back to the Integrator (step 9). The inclusion of SystemKey in the $\mathrm{IP_i}$ is very important, because it permits the $\mathrm{IP_i}$ to securely communicate with other IPs in other System nodes. The Initial Configuration receives the encrypted $\mathrm{IP_i}$ from the Integrator and stores it in the $\mathrm{ConfMem_i}$ (Steps 10 and 11). In the last step, the Initial Configuration initiates the multiboot and FPGAs load and decipher the $\mathrm{IP_i}$ (Step 12). If the $\mathrm{IP_i}$ configuration fails, the FPGA loads the Initial Configuration as a fall-back configuration.

B. IP Protection Scheme for High-end Multi-FPGA Systems

In the previous protocol, the IP configuration was regarded as one static design and it was necessary to protect it with

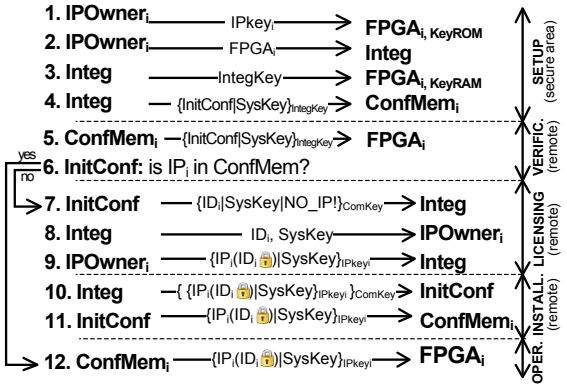

Figure 2. IP protection scheme for FPGAs without partial reconfiguration. The Integrator (Integ) installs the Initial Configuration (InitConf) protected by the IntegratorKey (IntegKey). Communication between the Integrator and the Initial Configuration is protected by the CommunicationKey (ComKey). Afterwards, the IP protected by the IPKey is licensed and installed. IP contains the SystemKey (SysKey) which serves as a shared system secret.

the IPKey. When considering implementation in partially reconfigurable FPGAs, each IP can consist of two parts: 1) a static IP establishment module (StaticModule); 2) reconfigurable IP module (ReconfModule). The static module is an instrument of the IPOwner used for establishing a secure environment inside the FPGA. Interestingly, the static module does not have to be protected by the IPOwner, because it contains the Noekeon white-box cipher. A strong obfuscation achieved by the white-box implementation provides sufficient security for the embedded IPKey. The static module also defines the reconfigurable area borders inside the FPGA. The Noekeon cipher is then used to decipher the reconfigurable module (protected by the IPKey) and to upload it to the reconfigurable area via the reconfiguration port. As an additional layer of protection, the static module is protected by the FPGA hardwired circuitry using the IntegratorKey to thwart all external attacks.

The protocol for high-end partially reconfigurable FPGAs is illustrated in Fig. 3. Unlike in Section III-A, the FPGA does not have to be initialized with the IPKey, but can be directly embedded in the System. When the System is assembled, the Integrator writes the IntegratorKey to KeyRAM of all FPGAs (Step 1) and stores the Initial Configuration (including the shared SystemKey) protected by the IntegratorKey to all ConfMems (Step 2). Only these two steps must be performed in a secure environment.

After power-up, the $\mathrm{FPGA_i}$ loads, decrypts and authenticates the Initial Configuration (Step 3). The Initial Configuration verifies if the $\mathrm{IP_i}$ configuration is present in the $\mathrm{ConfMem_i}$ (Step 4). If confirmed, the Initial Configuration initiates the multiboot and FPGAs load and decipher the $\mathrm{IP_i}$ configurations (Steps 10-12). However, if the $\mathrm{IP_i}$ is not present in $\mathrm{ConfMem_i}$ or is corrupted, the Initial Configuration establishes a secure communication (protected by the CommunicationKey) with the Integrator and reports the $\mathrm{IP_i}$

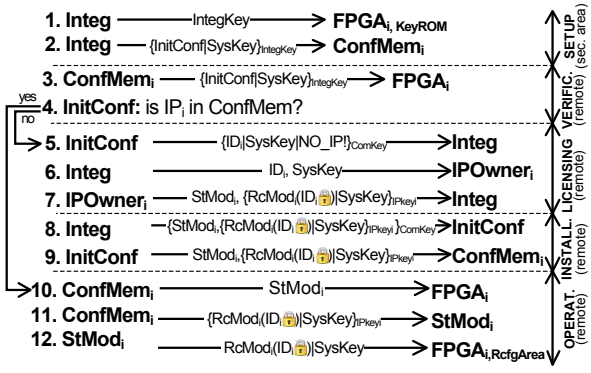

Figure 3. IP protection scheme for partially reconfigurable FPGAs. Static IP establishment module (StMod) is used to configure the IP reconfigurable module (RcMod) into the reconfigurable area (RcfgArea) in a secure way.

absence (Step 5).

The Integrator receives the FPGA $\mathrm{ID_i}$ and SystemKey and initiates the licensing process by sending them securely to $\mathrm{IPO_i}$ (Step 6). The $\mathrm{IPOwner_i}$ generates a $\mathrm{StaticModule_i}$ and $\mathrm{ID_i}$-locked $\mathrm{ReconfModule_i}$ (containing the SystemKey), deciphers the $\mathrm{ReconfModule_i}$ with the $\mathrm{IPKey_i}$ and sends both $\mathrm{IP_i}$ parts to the Integrator (Step 7). The Integrator enciphers the $\mathrm{StaticModule_i}$ with the IntegratorKey to enable its configuration to the $\mathrm{FPGA_i}$ and sends both $\mathrm{IP_i}$ parts to the Initial Configuration. The Initial Configuration receives and stores the $\mathrm{StaticModule_i}$ (encrypted by the IntegratorKey) and $\mathrm{ReconfModule_i}$ (protected by the $\mathrm{IPKey_i}$) in the $\mathrm{ConfMem_i}$ (Steps 8, 9) and initializes the multiboot.

The $\mathrm{FPGA_i}$ loads, deciphers, authenticates and configures the $\mathrm{ReconfModule_i}$ into its static logic area (Step 10). This way, the reconfigurable area is created in the $\mathrm{FPGA_i}$ too. The $\mathrm{StaticModule_i}$ fetches the $\mathrm{ReconfModule_i}$ from the $\mathrm{ConfMem_i}$ (step 11). The $\mathrm{ReconfModule_i}$ is deciphered (in CBC mode) and authenticated (in CMAC mode) by the Noekeon cipher and configured into the reconfigurable area via the partial reconfiguration port.

Readers may have noticed that the Noekeon cipher is used in the opposite order to achieve confidentiality: plaintext is transformed to ciphertext by the decryption ($\mathrm{IPOwner_i}$ domain) and then ciphertext is transformed back to plaintext by encryption ($\mathrm{StaticModule_i}$ FPGA domain). Interestingly, the Noekeon cipher in the $\mathrm{StaticModule_i}$ can execute the CMAC authentication algorithm at the same time. This way, only one shared cipher providing confidentiality and authenticity is necessary. This results in a smaller implementation.

IV. Implementation of a Multi-FPGA System

A. Target FPGA devices

The low-cost Multi-FPGA System can be implemented using Xilinx Spartan-6 FPGAs. Both volatile and non volatile key memories are available. The large FPGA devices contain an AES bitstream decryption circuit with 256-bit keys, i.e., both IntegratorKey and $\mathrm{IPKey_i}$ are 256 bits long.

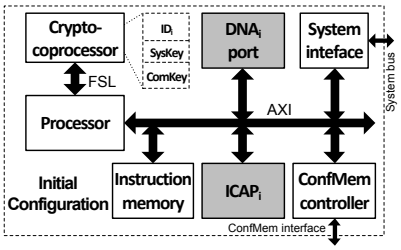

Figure 4. Structure of the system integrator module

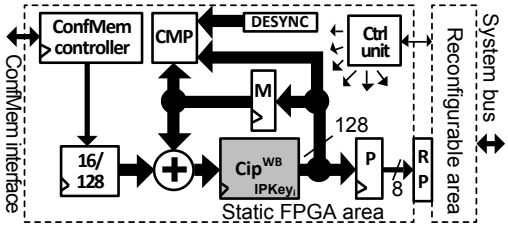

Figure 5. Structure of the IP establishment module (StaticModule)

The 57-bit ID can be read out from the DNA_PORT primitive. Multi-boot and readback can be controlled using the Internal Configuration Access Port (ICAP), which allows to access FPGA configuration registers. Although the bitstream authentication is not supported, its integrity can be verified by the CRC check. A fall-back bitstream can be selected via the ICAP.

The high-end system nodes can consist of Xilinx Virtex-6 FPGAs. In addition to the features available in Spartan-6, Virtex-6 supports partial reconfiguration via the ICAP, and an HMAC unit for bitstream authentication. No extra key register is required for HMAC, because the key is included in the bitstream.

B. Initial Configuration Structure

An example of the Initial Configuration structure is depicted in Fig. 4. The Initial Configuration is based on a MicroBlaze – crypto-coprocessor system presented in [12]. The processor firmware is stored in an instruction memory. All key registers are protected inside the crypto-coprocessor. The Initial Configuration can communicate with other nodes using the system bus and access the configuration memory using the memory controller. ICAP and DNA ports are FPGA-specific primitives.

C. IP Establishment Module - Static Module

The structure of the static module is given in Fig. 5. The most important part is the Noekeon cipher, which involves $\mathrm{IPKey_i}$). The bitstream data are decrypted in CBC mode. For the sake of simplicity, the CBC initialization vector is fetched with the bitstream and deciphered in ECB mode before data decryption. The $\mathrm{ReconfModule_i}$ bitstream is loaded from the $\mathrm{ConfMem_i}$ using the memory controller. The CBC mode provides bitstream confidentiality, but also bitstream authenticity when used as a CBC-MAC. In CBC mode, temporary results are stored in the mode register (M) and plain bitstream register (P). Data from the P register is shifted out to the reconfiguration port RP (ICAP) and used to configure the reconfigurable area. A comparator (CMP) is used to detect the end of the bitstream (desync word) and to compare the last $\mathrm{ReconfModule_i}$ word (fingerprint) with the CBC-MAC result. If both are matching, the $\mathrm{ReconfModule_i}$ is activated. Otherwise, the control unit reports the error and directs the multi-boot to load the fall-back configuration (i.e. the Initial Configuration) via partial reconfiguration port.

Table I
IMPLEMENTATION RESULTS FOR ONE SYSTEM NODE

	Low-cost node				High-end node	
	Spartan-6		Virtex-6		Virtex-6	
	Slices	RAM (kb)	Slices	RAM (kb)	Slices	RAM (kb)
SIM	2062	630	1978	1224	1978	1224
IP (IPE)	1494	603	1438	1206	1740 (294)	1206 (0)

D. Example of an IP Module

To prove the concept, the IP module is based on the HCrypt crypto-processor [13], which uses the SystemKey as a master key. HCrypt is activated only if the pre-stored $\mathrm{ID_i}$ matches the DNA port value.

V. IMPLEMENTATION RESULTS

One low-cost system node and one high-end system node were implemented in VHDL and synthesized using Xilinx ISE 12.4 for Spartan-6 XC6SLX75T and Virtex-6 XC6VLX240T. The latter configuration was successfully tested in the ML605 evaluation board. Since we did not have a card featuring the Spartan-6 device at our disposal, we performed the low-cost system node tests using the ML605 evaluation board, too. A hardware module including the Cypress USB device CY7C68013A was connected to the evaluation board for data transfers from/to the PC. All tests were successful. The results are summarized in Tab. I.

The $\mathrm{ReconfModule_i}$ bitstream was generated, the $\mathrm{ID_i}$ was updated using the data2mem tool and decrypted by Noekeon.

To verify the protection schemes, three applications were developed. The first represented a communication scenario between the FPGA node and the IPOwner on behalf of the Integrator. The second handled IPOwner's tasks: it communicated with the Integrator and generated the ID-locked IPs. In order to test communication between nodes, one node was implemented in the FPGA and the other was emulated by the third application on the PC. All tests were carried out for both low-cost and high-end system nodes.

VI. DISCUSSION

The number of slices required for the Initial Configuration is very similar for both FPGA devices, because the slice structure is almost identical. Since Virtex-6 Block-RAMs are twice the size of those in Spartan FPGAs, they are not efficiently utilized by the Initial Configuration, and so the memory size doubles (Initial Configuration: 1224 vs 630

kb, IP: 1206 vs 603 kb). The IP design for the high-end system is bigger (1740 vs 1438), because it contains also the StaticModule (294 slices).

Our first protocol assumes that the IPKey is written to the KeyROM in the IPOwner's secure environment and the IntegratorKey to the KeyRAM in the Integrator's secure environment. Thus, the total system costs must include FPGA transportation costs from IPOwners to Integrator. Moreover, in case of a battery failure, the IntegratorKey is lost and cannot be restored remotely. Thus, the whole device must be returned to the Integrator, which results in additional transportation overhead. Despite these disadvantages, the first scheme remains the most suitable solution for multi-FPGA systems based on low-cost FPGAs.

The transportation cost issues are solved in the second scheme. The IntegratorKey is stored in the KeyROM and the white-box cryptography provides a unique and secure way to embed IPKeys in FPGAs without any pre-stored IPOwner secret. Furthermore, several ReconfModules can be stored in the ConfMem and the StaticModule can upload the one that is required.

As mentioned in Sec. I, the two schemes are aimed at the use in the automotive market. The car manufacturer (Integrator) assembles a car containing multiple functional units (nodes) and installs his private initialization bitstream in each node. The manufacturer can buy IPs for individual nodes from different IPOwners and install or update them remotely. At the same time, IPOwners would like to license their IPs and protect them against cloning and reverse engineering.

Many other applications of the high-end multi-FPGA systems and the corresponding protection scheme can be found. The simplest one is a cluster of FPGAs, where different customers (IPOwners) would like to rent some computation time without delivering their IP bitstreams to third parties or the cluster provider (in the role of an Integrator). Other applications could be a trusted computer platform offered by a certification authority (Integrator). This platform can contain only certified modules (nodes). Uncertified modules cannot posses the common shared secret and are excluded from any confidential communication with other modules. The system is convenient for all these applications and its protection scheme can provide security for all participating parties and their intellectual properties.

The proposed protection schemes could be significantly improved if the KeyRAM could be programmed from inside the FPGA. Moreover, the SystemKey could be exchanged directly between the Initial Configuration and IP configurations if a secure non-volatile user storage was embedded in the FPGA. Current FPGAs can store only two different configuration keys. However, if more keys could be stored, more IP from different IPOwners could be implemented in a single FPGA.

VII. Conclusion

We proposed two novel protection schemes for IP bitstreams implemented on multi-FPGA systems. The first scheme is targeting low-cost FPGAs and provides a license scheme for IP owners to offer their products to system integrators in a secure way. The scheme uses a volatile and a non-volatile key storage of recent (Xilinx Spartan-6) FPGA devices to store both system integrators' and IP owners' keys. The appropriate key register is selected by the bitstream itself.

The second unique scheme is provided to high-end partially reconfigurable FPGAs and enables IP owners to remotely install their IPs in an untrusted FPGA environment without having any pre-stored secret. These properties are achieved by hardware white-box cryptography.

References

[1] Synplicity Inc., "An Open IP Encryption Flow permits industry-wide interoperability," 2006.

[2] K. Kepa, F. Morgan, K. Kosciuszkiewicz, and T. Surmacz, "SeReCon: a secure reconfiguration controller for self-reconfigurable systems," *International Journal of Critical Computer-Based Systems*, vol. 1, no. 1, pp. 86–103, 2010.

[3] T. Guneysu, B. Moller, and C. Paar, "Dynamic intellectual property protection for reconfigurable devices," in *FPT'07*. IEEE, 2007, pp. 169–176.

[4] S. McNeil, "Solving today's design security concerns," *Xilinx Corporation*, 2012.

[5] FIPS-197, "Advanced Encryption Standard (AES)," 2001.

[6] J. Daemen, M. Peeters, G. Van Assche, and V. Rijmen, "Nessie proposal: Noekeon," in 1st *NESSIE Workshop*, 2000.

[7] NIST800-38B, "The CMAC Mode for Authentication," 2005.

[8] FIPS-198, "The Keyed-Hash Message Authentication Code," 2002.

[9] W. Brecht, "White-box cryptography: hiding keys in software," *NAGRA Kudelski Group*, 2012.

[10] S. Chow, P. Eisen, H. Johnson, and P. Van Oorschot, "White-box cryptography and an AES implementation," in *Selected Areas in Cryptography*. Springer, 2003, pp. 250–270.

[11] Z. Cherif, F. Flament, J. Danger, S. Bhasin, S. Guilley, and H. Chabanne, "Evaluation of white-box and grey-box Noekeon implementations in FPGA," in *ReConFig'10*, 2010, pp. 310–315.

[12] L. Gaspar, V. Fischer, L. Bossuet, and M. Drutarovsky, "Cryptographic extension for soft general-purpose processors with secure key management," in *FPL'11*, 2011, pp. 500–505.

[13] L. Gaspar, V. Fischer, F. Bernard, L. Bossuet, and P. Cotret, "HCrypt: A Novel Concept of Crypto-processor with Secured Key Management," *ReConFig'10*, pp. 280–285, 2010.