Dynamically Configurable Security for SRAM FPGA Bitstreams

Lilian Bossuet[1], Guy Gogniat[1], Wayne Burleson[2],

[1] LESTER Lab
Université de Bretagne Sud,
Lorient, France
{lilian.bossuet, guy.gogniat}@univ-ubs.fr

[2] Department of Electrical and Computer Engineering
University of Massachusetts,
Amherst, USA
burleson@ecs.umass.edu

Abstract

This paper proposes a solution to improve the security of SRAM FPGAs through bitstream encryption. This proposition is distinct from other works because it uses the latest capabilities of SRAM FPGAs like partial and dynamic reconfiguration. It doesn't need any external battery to store the secret key. It opens a new way of application partitioning according to the security policy.

1. Introduction

The FPGA (Field Programmable Gate Array) concept is born during the 80's, at that time the configuration point size (transistors or fuses) was too large in comparison with the chip size to have an interesting FPGA density. So these devices were just used to do prototyping or glue logic. If during a long time the FPGAs have not taken benefit from the best deep-submicron technology, today the more advanced FPGAs use 90 nanometer technology with copper metallization (best actual accessible technology). With the improvement of technological processes and since the FPGAs structure is very regular, it is possible to build some FPGAs with more than one million transistors. Thanks to these evolutions, FPGAs are more and more attractive for numerous systems and to build efficient SoC (System on a Chip). The FPGAs market increases and FPGAs are capturing classical market share of ASIC (Application Specific Integrated Circuit) market. The cost cross-over point, that permits to know the necessary number of systems built to choose an efficient ASIC solution, is more and more far [1]. It is possible even for an important number of systems built to choose an economical-efficient FPGA solution.

Since FPGAs are becoming so important for electronic industry it is necessary to think about the security of FPGA-based systems. It is possible to consider the FPGA-based systems security problem in three ways:

System security using FPGA: in this case the FPGA is used as one part of the security system such as network isolations (firewalls).

Protecting FPGA data: in this case it is necessary to protect the FPGA application. The data inside the circuit and the data transferred to/from the peripheral circuits during the communication must be protected. The main solution is to integrate data encryption scheme inside the FPGA.

FPGA design security: in this last case the protection concerns the design against cloning and reverse engineering. It is intellectual property protection. Concerning the SRAM FPGAs it corresponds to the way to protect the bitstream so the FPGA configuration.

This article focuses on this last way to deal with FPGA and the security problem. If the FPGA design itself is not secure the other security problems can not be efficiently treated. Many works already propose solutions to protect the bitstream. However the contribution of this paper relies on the utilization of the latest improvements of SRAM FPGAs configuration techniques to answer the security problem.

This paper is organized as follows. Section 2 describes some aspects of the design security problem. Section 3 presents several works dealing with the protection of SRAM FPGA configuration. In Section 4 a new solution is proposed. Finally, the section 5 concludes this paper and exposes several future directions.

2. Design Security

It is interesting before investigating the different solutions to secure the configuration of SRAM FPGAs to list what are the different attacks against an integrated circuit today, what is the protection level of some current circuits and why do they have this level of protection?

2.1 Need for Design Security

The problem of design security is simple, the designer doesn't want that a competitor could be able to pirate his design. There are two sorts of piracy:

The Cloning is when a competitor makes a copy of the design, and when he is able to create a copy of the pirated system.

The Reverse Engineering is when a competitor copies a design by reconstructing a "schematic" or netlist level representation; in this process, he understands how the design works and how to improve it, or to modify it with malicious intents.

So reverse engineering is more serious than cloning. These two aspects correspond to different attacks, and the design security must protect the system against both attacks. There are two types of attack; the non-invasive and the invasive attacks:

The non-invasive attacks gather all the methods that use external means. For example the attackers can use all the possibilities of the circuit inputs in order to obtain all the different outputs and draw the system truth table, this method is called "Black Box Attack".
In the case of SRAM FPGA a simple attack method consist in intercepting the bitstream between the root ROM and the FPGA when the system power is switched on. More complex attacks can bring into play time, power and electromagnetic changes and measures like the simple or differential power analysis, interested readers can refer to the work on power analysis of FPGA in [2].

The invasive attacks (or **physical attacks**) are characterized by the necessity to destroy the integrated circuit (component package) to study the chip (design inside the component) with some complex methods. For example it is possible to use laser cutter microscope in order to split the chip in several slices and understand the chip layout. These attacks can use sophisticated tools like optical microscope, mechanical probes and even Focused Ion Beam (FIB). As these attacks use the weakness of the silicon technology, when they are possible, it is very hard to secure the system against them.

The papers [3] and [4] give some information about these different attacks. It is possible to classify the integrated circuits according to their protection against the different types of attacks.

2.2 Protection level of some circuits

The level of protection offered by actual integrated circuits is an interesting metric to be thinking of works to improve the security level of one particular type of integrated circuit. In the IBM Systems Journal a paper [5] defines the various security levels for modern electronic systems and the corresponding taxonomy of attackers:

Level 0 (ZERO): No special security features added to the system.

Level 1 (LOW): Some security features in place. They are relatively easily defeated with common laboratory or shop tools.

Level 2 (MODLOW): More expensive tools are required, as well as specialized knowledge.

Level 3 (MOD): Special tools and equipment are required, as well as some special skills and knowledge. The attack may become time-consuming but will eventually be successful.

Level 4 (MODH): Equipment is available but is expensive to buy and operate. Special skills and knowledge are required to utilize the equipment for an attack. More than one operation may be required so that several adversaries with complementary skills would have to work on the attack sequence. The attack could be unsuccessful.

Level 5 (HIGH): All known attacks have been unsuccessful. Some research by a team of specialists is necessary. Highly specialized equipment is necessary, some of which might have to be designed and built. The success of the attack is uncertain.

According to this classification it is possible to give a general security level for the current integrated circuits. Of course these different levels are not fixed, and depend of the factory, the type of circuit (in a same factory there are several families and some of them can be especially security-efficient like some military families). The authors have tried to give one level by classical integrated circuit and explain the reason of their choices. The security level of the classical integrated circuits is given in the table 1.

Conventional SRAM FPGAs have the lowest security level. These circuits need a bitstream transfer from the root ROM at power up (because the memory of configuration is a SRAM volatile memory). So it is easy for the pirate to read with a simple probe the bitstream during the transfer. The conventional SRAM FPGAs are inefficient for safe design. But with a bitstream encryption it is possible to clearly improve the security level since the security weakness is secure. SRAM FPGAs have a good resistance against some attacks like power analysis [2] even if today few works present the results of attacks against SRAM FPGA [6].

Integrated Circuit	**Security Level**
Conventional SRAM FPGA	1
ASIC Gate Array	3
Cell-based ASIC	3
SRAM FPGA with bitstream encryption	4
Flash FPGA	5
Antifuse FPGA	5

Table 1. Security level of classical integrated circuits Actel - Security [7]

Often considered like a secure technology, ASICs are actually relatively easy to reverse engineer. Because unlike FPGAs, ASICs have no switch. So it is possible to strip the chip to copy with certitude the complete layout in order to understand how it works. It is not a simple process so the security level is 3 for such devices.

Contrary to the ASICs, the FPGAs like antifuse or flash are really security-efficient since they are based on switches. With these FPGAs no bitstream can be intercepted in the field (no bitstream transfer, no external configuration device). In the case of antifuse FPGAs the attacker needs a Scanning Electron Microscope (SEM) in order to know the state of each antifuse. But the difference between a programming and a non-programming antifuse is very difficult to see. Moreover such analysis is intractable in a device like Actel AX2000 that contains 53 million of antifuses and according to Actel [7] only 2-5 % (average) of these antifuses are programmed. For Flash FPGA there is no optical difference after configuration, so the invasive attacks are very complex.

If the antifuse and the flash FPGAs are very security-efficient they are just one time configurable, so they are not really reconfigurable devices. If the designer wants a reconfigurable device he must target a SRAM FPGA. Moreover the capacity of the SRAM FPGAs are the highest for FPGAs devices. Actually the SRAM FPGAs have a market share higher than 60 % (just with the two leaders companies Xilinx [8] and Altera [9]). So the research to improve the security level of such FPGAs and particularly the improvement of bitstream encryption are today necessary.

Some works give efficient solutions to encrypt the SRAM FPGA bitstream. But there are some drawbacks and it is possible to improve them taking into account the latest innovations of these FPGAs. The following section presents some works about the bitstream encryption.

3. Related work

Two approaches are generally possible to address the design security problem. The first one considers that the best solutions to protect the devices against piracy are legal solutions. The definition of efficient laws, the regulation and the management of intellectual properties are parts of this solution.

The second one according to the last section, proposes to improve the security level of actual SRAM FPGAs by configuration protection (bitstream encryption). In the following we only address this last approach.

Xilinx proposes security system [8] based on a triple DES encryption scheme to protect the bitstream of the Virtex-II and Virtex-II Pro family device [10].

Xilinx CAD software tool encrypts the bitstream using the powerful Triple Data Encryption (DES) algorithm before downloading the configuration inside the FPGA. Triple DES is the standard used by many governments for safe communication and by banks around the world for money transfers. This algorithm uses three 56-bits public keys. The designer can use random keys or choose their own-keys.

The figure 1 shows the encryption/decryption system used by Xilinx to protect the configuration of Virtex-II devices.

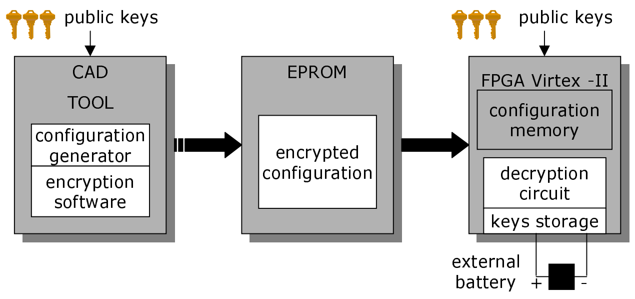

Figure 1. Xilinx Virtex-II triple DES encryption scheme

This system is relatively simple; it is just necessary to choose one option during the last step of the CAD process, the bitstream generation. Firstly a key file that describes the configuration of the three keys, is programmed inside the FPGA. Of course it is not necessary to store the key file inside the configuration memory. The keys are stored in a dedicated SRAM memory inside the FPGA that can be backed up with a small battery (like a watch battery).

After, the configuration step is performed like a classical configuration without bitstream encryption. In fact, the configuration stored in the external EPROM is encrypted. The FPGA contains a decryption circuit that automatically detects when the bitstream is encrypted and decrypts the configuration before the SRAM bits are programmed. Xilinx does not give information about the necessary extra-time to decrypt the configuration.

The Xilinx bitstream encryption scheme is efficient because without the correct key it is not possible to configure other chips with the encrypted bitstream. Nevertheless when the device is configured, it is not possible to use partial reconfiguration and to do read-back.

If the designer does not need security, he can configure the device with non-encrypted bitstream and the on-chip keys are simply ignored.

Nevertheless this method has a strong drawback; it uses an external battery to save the key. It is poor for several reasons. This solution costs a lot of area on the board and even if the used battery is small it is necessary to add it a socket, and the board area is a critical issue for embedded application. Moreover this solution increases the board cost and reduces the system lifetime.

It is necessary to improve the Xilinx solution by proposing a solution without additional battery.

Tom Kean of the Algotronix society proposes an attractive solution to answer this problem [11] [12]. The first idea of Kean is to use a secret cryptographic key stored on an FPGA. He gives some ways to store this key like using a laser to program a set of links during manufacture.

As the secret key is only known by the FPGA, it must contains an encryption and a decryption circuit. But contrary to Xilinx method, the CAD doesn't change and just generates a classical bitstream.

The figure 2 shows the encryption/decryption system used by Kean to protect the configuration of SRAM FPGA. The figure 2(a) shows the initial configuration of secure FPGA and the figure 2(b) shows the normal configuration of secure FPGA.

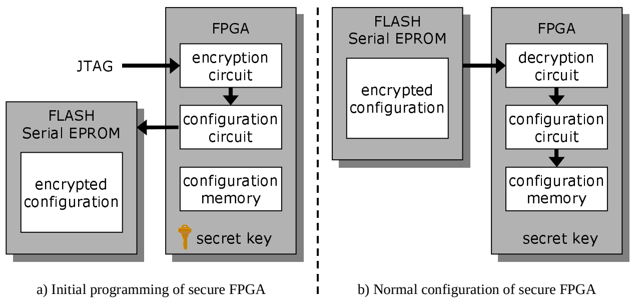

Figure 2. Kean proposed encryption/decryption scheme [11] & [12]

This solution has many advantages, it does not affect system reliability, it does not require additional components and it does not require support from CAD software. In this system nobody (the designer or the CAD tool) needs knowledge on the key.

If Kean's solution overcomes the battery limitation of the Xilinx solution, both solutions have the same important disadvantages. In the two cases the encryption and the decryption circuits are embedded inside the FPGA. These circuits take FPGA silicon area normally reserved for the developed application. So the total application dedicated-area is reduced by these solutions. Moreover in these solutions the encryption and the decryption circuits are fixed, so it is not possible to upgrade them or to choose the encryption/decryption algorithm and architecture.

In both solutions, the entire design is encrypted with the same encryption algorithm. However such approach is very restrictive since it doesn't consider any security policy. Actual designs (due to the high degree of application complexity) are based on numerous heterogeneous parts that do not present the same "security sensitivity". Hence, the designer may want to partition his application in several parts and use different encryption/decryption algorithms to encrypt/decrypt these parts. For example if the designer uses some free or very easy to find IPs (Intellectual Property) it may be not necessary to encrypt these parts of the application. Other parts like interfaces for example don't need a high security level. On the other hand the real designer's IPs need a high security level.

Finally the two proposed solutions give only one fixe answer to the bitstream security problem and they have a lake of flexibility.

Other solutions are proposed, most of them can be found in recent US Patents for example [13], [14], [15] and [16]. But these solutions are not very different from Xilinx [10] or Kean [11, 12] solutions.

If existing solutions are not very different one from another it is mainly due to the fact that they don't use the new features of SRAM FPGAs like partial reconfiguration or dynamic reconfiguration.

In the following section we present a new solution to address the bitstream security problem that takes advantage of this new features.

4. A new solution to protect the SRAM FPGA bitstream.

4.1 New possibility of SRAM FPGA

According to previous sections actual solutions to secure the SRAM FPGA bitstream are efficient but they have a lake of flexibility. Although the main advantage of the reconfigurable devices like SRAM FPGAs (particularly in comparison with other FPGAs or ASIC) is the flexibility given by the reconfiguration capabilities. This advantage is more and more important with the new capabilities of SRAM FPGAs like partial reconfiguration, dynamical reconfiguration or auto-reconfiguration.

In [17] Xilinx presents a self-reconfiguring platform (SRP) for Xilinx Virtex-II and Xilinx Virtex-II Pro devices. Self-reconfiguration extends the concept of dynamic reconfiguration. It assumes that dedicated circuits of the FPGA are used to control the configuration of the other parts of the FPGA. In this case the FPGA is able to dynamically reconfigure itself under the control of an embedded microprocessor. This microprocessor can be a soft core like Xilinx MicroBlaze (32-bit RISC) or a hardcore IBM PowerPC (32-bit RISC) embedded on the Xilinx Virtex-II Pro. To perform the dynamical reconfiguration, the microprocessor uses a specific interface called ICAD (Internal Configuration Access Port) and a small configuration cache that uses the embedded RAM (called BlockRAM in Xilinx Virtex devices) of the FPGA.

Xilinx proposes a tool to manage these new FPGA capabilities called XPART for Xilinx Partial Reconfiguration Toolkit.

4.2 Proposed solution

This solution takes benefit of the new possibilities of reconfiguration of SRAM FPGAs to improve their security level without the drawbacks highlighted previously: The encryption and the decryption circuit must leave all the silicon area free for the developed application. Of course the solution must use an embedded key in order to work without extra-battery. To store the key a solution close to Kean's solution [11, 12] can be chosen. It is possible to use a laser to engrave the key or use some antifuse elements to do a non-volatile key programming.

A very important feature is also to give the designer the opportunity to choose the encryption/decryption algorithms and architectures. In this way it is possible to adapt the encryption/decryption scheme according to the requested security level for the developed application. Furthermore, the feature enables to easily upgrade the system if a new efficient encryption/decryption algorithm is available.

Finally we address the sensitivity policy problem by allowing the designer to use different encryption algorithms for a single application. The Security-Critical Parts (SCP) of the application will only be encrypted.

As the encryption/decryption scheme has a cost because this scheme spends time, consumes power and takes silicon area, it is very interesting to adapt it according to the required security level of the application.

To understand our approach an example is given during the initial configuration step and the normal configuration step. In this example the application is partitioned into three different parts; two SCPs that need high security level (but will be encrypted with two different encryption algorithms) and one other part that doesn't need encryption. The case with 2 SCPs is just an example other configurations can be considered.

The figure 3 shows the encryption system when the FPGA is initially configured and the root configuration memory is programmed (initial configuration).

During the initial FPGA configuration the first step consists in programming the root configuration memory with the non-encrypted parts. There are two decryption circuits that will be used to decrypt the encrypted bitstreams, and the non-encrypted part of the application.

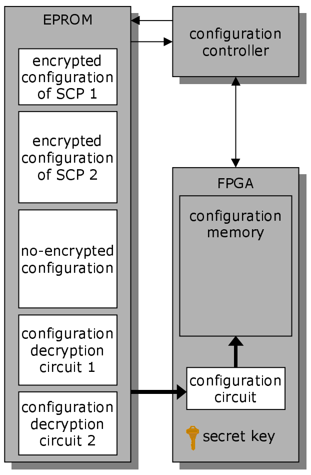
EPROM
encrypted configuration of SCP 1
encrypted configuration of SCP 2
no-encrypted configuration
configuration decryption circuit 1
configuration decryption circuit 2
configuration controller
FPGA
configuration memory
configuration circuit
secret key

Figure 3. Encryption scheme during the initial FPGA configuration

After it is necessary to configure the FPGA with the *encryption circuit 1* in order to encrypt the configuration of the application *SCP 1*. Once the *SCP 1* configuration is encrypted, it is stored in the root external EPROM configuration memory. Since the *SCP 2* needs other encryption circuit, the FPGA configuration is cleared in order to be programmed with the *encryption circuit 2*.

Of course it is necessary for the CAD to manage partial reconfiguration like in Xilinx proposition [17].

At the end of the initial configuration step the root configuration memory contains the encrypted configuration of *SCP 1* and *SCP 2*, the non-encrypted configuration of the other application parts and the configuration of the decryption circuits required to decrypt *SCP 1* and *SCP 2* (*decryption circuit 1* and *decryption circuit 2*). The FPGA configuration is erased.

The figure 4 shows the decryption-configuration system when the FPGA is configured with an extra EPROM memory that stores the configuration (normal configuration).

The FPGA configuration process works as follows: First the FPGA is configured with the *decryption circuit1*. Then the FPGA uses it to decrypt the encrypted configuration of *SCP 1* and auto-configures the *SCP 1* inside the FPGA. Since the *SCP 1* is decrypted it is not necessary to keep the *decryption circuit 1*. The decryption circuit 2 replaces it in order to decrypt the encrypted configuration of *SCP 2*. In the same way after the configuration it is not necessary to keep the *decryption circuit 2*. After this first phase the FPGA is configured with the *SCP 1* and the *SCP 2* parts. The last step consists in configuring the FPGA free area with the other application parts that have not an encrypted configuration.

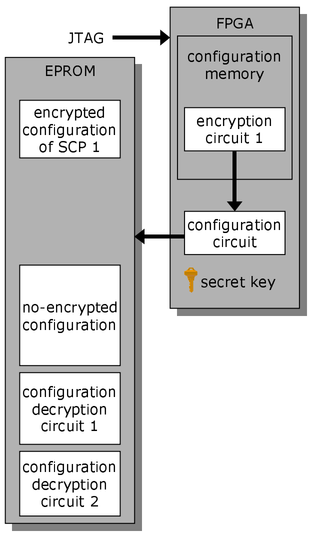
JTAG
FPGA
configuration memory
encryption circuit 1
configuration circuit
secret key
EPROM
encrypted configuration of SCP 1
no-encrypted configuration
configuration decryption circuit 1
configuration decryption circuit 2

Figure 4. Decryption-configuration scheme during the normal FPGA configuration

Finally the FPGA is configured with all the application parts.

As the partial configuration is a specific task, it is necessary to use a dedicated configuration controller. This controller can be external like a dedicated CPLD or a microprocessor. But it is also possible that this controller is embedded inside the FPGA, like in the case of Xilinx auto-configuration system [17]. In this last case the configuration controller can be a soft-core microprocessor (like Xilinx MicroBlaze) or, a hardcore microprocessor (like IBM PowerPC for Xilinx VirtexII-Pro devices).

The security configuration controller (SCC) is based on a finite state machine to perform the configuration management. To manage the configuration sequence the SCC needs the external EPROM memory partitioning (the memory mapping). We can notice that this mapping can be complex in order to still improve the system security. For example the designer can interleave the data stored in the memory and mixes the several encrypted and no-encrypted configurations. A configuration address register stores the memory mapping.

With the knowledge of the memory mapping, the configuration management finite state machine is simple. The figure 5 shows the 3-states used by the SCC. The table 2 describes the actions associated to the states of the SCC. The first state of this 3-states FSM is an idle state. To change of state the SCC waits for a start signal. This signal is the begin-signal of the configuration. Once in the second sate, the loading state, the SCC changes of state according to the type of configuration file. If the configuration file is not encrypted the current state is the second state. In this state the normal configuration of the FPGA is performed. If the configuration file is encrypted the current state is the “loading and decrypting” state. In this state the SCC loads first the configuration of the decryption circuit inside the FPGA before loading the encrypted configuration of a SCP.

The machine returns in the idle state when all the application is loaded inside the FPGA.

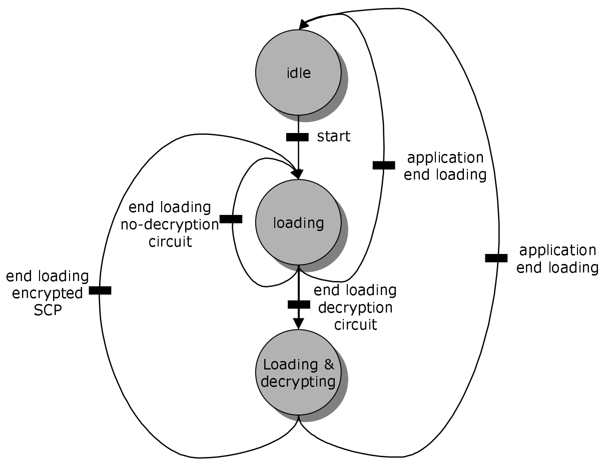
idle
start
application end loading
end loading no-decryption circuit
loading
application end loading
end loading encrypted SCP
end loading decryption circuit
Loading & decrypting

Figure 5. Configuration controller FSM

State name	actions
idle	Wait start
loading	Load selected configuration* on the FPGA. Update the configuration address register. **the selected configuration can be the non-encrypted configuration of a decryption circuit or of a non-SCP application part.*
Loading & decrypting	Start the decryption algorithm and load the corresponding SCP configuration on the FPGA. Update the configuration address register.

Table 2. States description of the configuration controller FSM

One feature is very important in this solution; the key management. It is mandatory that a pirate can’t access to the keys used by the different decryption/encryption circuits. Moreover, as in this solution the decryption/encryption algorithm is not fixed, it is necessary to store a large key. Indeed different algorithms don’t use the same key size (for example the AES algorithm uses a 128 bits key, and the triple DES uses three 56 bits keys so 168 bits key). In fact among the *n* key bits the encryption/decryption circuits select *m* necessary bits. Since only the designer knows the *m* chosen bits it is a supplementary security barrier.

4.3 Drawbacks and advantages of the proposed solution

If this method permits to overcome the limitation of other proposed solutions, it has some drawbacks.

The first drawback is the relative complexity of the method, since it is necessary to manage the partial reconfiguration and auto-configuration. Most of the FPGA manufacturers don't have CAD tool to manage these types of configuration but Xilinx proposes an efficient tool for such needs.

The decryption circuit can have several sizes according to the algorithm and the implementation. Several works give comparisons of the hardware performance of the different AES final candidates (MARS, RC6, Rijndael, Serpent or Twofish for example) using FPGA [18][19][20][21]. All these works use the Xilinx Virtex as reconfigurable target. The table 3 and the table 4 compare the results of these studies for the area requirement (one Virtex slice corresponds to two 4 inputs LUTs, two flip-flop and one carry chain) and time performance (throughput).

Algorithm	# slices of the cryptographic core		
	[18]	[19]	[20]
Rijndael	4312	5302	2902
Serpent	1250	7964	4438
RC6	1749	3189	1139
Twofish	2809	3053	1076
MARS	4621	- -	2737

Table 3. Area requirement of FPGA implementations of AES final candidates

Algorithm	Throughput (Mbit/s)		
	[18]	[19]	[20]
Rinjdael	353.0	300.1	331.5
Serpent	148.9	444.2	339.4
RC6	112.9	126.5	103.9
Twofish	173.1	119.6	177.3
MARS	101.9	- -	39.8

Table 4. Time performance of FPGA implementations of AES final candidates

The performances (time and area) showed in the two tables are different for each work. Because the architectures chosen, for the different studies, have different structures (loop unroll, pipeline and sub-pipeline). All these results are given only for an encryption core without the key-setup circuit. But this circuit must be considered because it can take area (slices). The table 5 shows the number of slices for key-setup circuit of the five AES final candidates and relative area percentage of the total area requirement (encryption core and key-setup circuits).

Algorithm	# slices of the key-setup circuit		% of the total area	
	[18]	[21]	[18]	[21]
Rijndael	1361	128	24 %	14 %
Serpent	1300	2060	51 %	35%
RC6	901	290	34 %	15 %
Twofish	6554	1260	70 %	48 %
MARS	2275	50	33 %	3 %

Table 5. Area requirement of FPGA implementations of AES final candidates

According to these results, it is significant to consider the key-setup circuit in the area requirement. Finally the three tables show that a same decryption standard (AES in this example) can be performed with several algorithms and each algorithm can have different implementations. So it is necessary to give all the possibilities to the designer, and our solution gives all this flexibility.

The configuration controller can be complex. Its complexity depends on the number of SCPs in the application. This number is correlated to the application security partitioning. The costs of a larger root memory and a complex configuration controller are the hardware overhead costs of this method but they represent the origin of its flexibility. The system security has always costs that are necessary to evaluate in order to choose the best solution according to the required security level.

Since it is necessary to first configure the decryption circuit before the real configuration of each SCP, this method can spend time when the system is powered up. But today the SRAM FPGA configuration is more and more fast (about 10 millisecond for a partial reconfiguration for a Xilinx Virtex 1000-E device [22]).

This method has many very interesting advantages. First the encryption/decryption circuits don't take FPGA application-dedicated resources, since when a decryption circuit has been used it is removed from the FPGA. The FPGA resources initially used to perform the decryption circuit are free for other uses.

We choose, like Kean [11, 12], to embed the key inside the FPGA in order to have not an external extra-battery.

One of the main advantages of this method is the increase of flexibility. The designer can partition the application according to the required security level. So if just a small part of the application needs a strong security, the system can be very simple (just one small SCP). The designer has all the possibilities to choose the suitable algorithms and architectures for the encryption/decryption circuits. It is possible to adjust the security level according to the application constraints.

Moreover the designer can upgrade its application and the security scheme with the same reconfigurable hardware. In this way it is possible to take advantage of the latest improvements of the security field.

5. Conclusions and future work

Since the SRAM FPGAs are more and more important for the electronic industry it is necessary to improve the security level of such devices. Although some works have already proposed solutions to improve this security level, we think that is it possible to investigate more this domain.

In this article we propose a new solution to prevent piracy against SRAM FPGAs bitstream. Our contribution is to use the latest developments of configuration technique in order to improve the security system flexibility.

This work is not yet practical or experimental but it corresponds to a proposal to the community in order to become more efficient.

For the near future we want to study the system feasibility, and particularly the management of partial reconfiguration and auto-reconfiguration.

We think that the security problem is a very important issue for FPGAs and for all the reconfigurable systems on chip. Probably in a near future there will be more and more works about this subject.

6. References

[1] N. Tredennick, B. Shimamoto. The Rise of Reconfigurable Systems. *In proceeding of Engineering of Reconfigurable Systems and Application, ERSA'2003. June 23-26, 2003, Las Vegas, Nevada, USA.*

[2] F.X. Standaert, L. van Oldeneel tot Oldenzeel, D. Samyde, J.J. Quisquater. Power Analysis of FPGAs: How Practical Is the Attack. *In proceeding of 13th International Conference on Field-Programmable Logic and Applications, FPL'2003, pp. 707-711. September 2003, Lisbon, Portugal.*

[3] R. Anderson, M. Kuhn. Tamper Resistance – a cautionary Note. *In Proceeding of the Second USENIX Workshop on Electronic Commerce, pp. 1-11. November 18-21, 1996, Oakland, California, USA.*

[4] R. Anderson, M. Kuhn. Low Cost Attack on Tamper Resistant Devices. *In Proceeding of the 5th Workshop of Security Protocols, pp. 125-136. April 7-9, 1997, Paris, France.*

[5] D.G. Abraham, G.M. Dolan, G.P. Double, J.V. Stevens. Transaction Security System. *In IBM Systems Journal, vol. 30, no 2, pp. 206-229, 1991.*

[6] T. Wollinger, C. Paar. How Secure Are FPGAs in Cryptographic Applications. *In proceeding of 13th International Conference on Field-Programmable Logic and Applications, FPL'2003, pp. 707-711. September 2003, Lisbon, Portugal.*

[7] Actel Coporation. Resource Center: Security. www.actel.com/products/rescenter/security/index.html

[8] Xilinx Coporation www.xilinx.com

[9] Altera Coporation www.altera.com

[10] Xilinx Coporation. Virtex-II platform FPGA Handbook. Available on www.xilinx.com

[11] T. Kean. Secure Configuration of Field Programmable Gate Arrays. *In proceeding of 11th International Conference on Field-Programmable Logic and Applications, FPL'2001. Belfast, United Kingdom.*

[12] T. Kean. Secure Configuration of Field Programmable Gate Array. *Proceedings IEEE Symposium on Field Programmable Custom Computing Machines (FCCM), Rohnert Park CA , 2001.*

[13] Kelen, et al. System and Method for PLD Bitstream Encryption. *US Patent 6 118 869, September 12, 2000.*

[14] Erickson, et al. Encryption of Configuration Stream. *US Patent 6 212 639, April 3, 2001.*

[15] Mason, et al. Secure Programmable Logic Device. *US Patent 6 331 784, December 18, 2001.*

[16] Pang, et al. Non Volatile/Battery-Backed Key in PLD. *US Patent 6 336 117, April 2, 2002.*

[17] B. Blodget, P. James-Roxby, E. Keller, S. McMillan and P. Sundararajan. A Self-reconfiguration Platform. *In proceeding of 13th International Conference on Field-Programmable Logic and Applications, FPL'2003, pp. 565-574. September 2003, Lisbon, Portugal.*

[18] A. Dandalis, K. Prasanna, J. D. P. Rolim. A Compartive Study of Performances of the AES Final Candidates Using FPGA. *Workshop on Cryptographic Hardware and Embedded Systems, August 2000.*

[19] A. J. Elbirt, W. Yip, B. Chetwynd, C. Paar. An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists. *In proceeding of the third Advanced Encryption Standard candidate conference, AES3. April 12-14, 2000, New York, New York, USA.*

[20] K. Gaj, P. Chodowiec. Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware. *In proceeding of the third Advanced Encryption Standard candidate conference, AES3. April 12-14, 2000, New York, New York, USA.*

[21] N. Weaver, J. Wawrzynek. A Comparison of the AES Candidates Amenability to FPGA Implementation. *In proceeding of the third Advanced Encryption Standard candidate conference, AES3. April 12-14, 2000, New York, New York, USA.*

[22] J-P Delahaye. Systèmes Radio Dynamiquement Reconfigurables sur des Architectures Hétérogènes. *Master Thesis, Université de Paris XI, Faculté d'Orsay. Spetember 2003.*