An FPGA Configuration Scheme for Bitstream Protection

Masaki Nakanishi

Graduate School of Information Science, Nara Institute of Science and Technology
Takayama, Ikoma, Nara 630-0101, Japan
m-naka@is.naist.jp

Abstract. FPGAs are widely used recently, and security on configuration bitstreams is of concern to both users and suppliers of configuration bitstreams (e.g., intellectual property vendors). In order to protect configuration bitstreams against the threats such as FPGA viruses, piracy and reverse engineering, configuration bitstreams need to be encrypted and authenticated before loaded into FPGAs. In this paper, we propose a new FPGA configuration scheme that can authenticate and/or decrypt a bitstream. The proposed scheme has flexibility in choosing authentication and/or decryption algorithms and causes only a small area overhead since it utilizes programmable logic blocks to implement authentication and/or decryption circuits.

Keywords: FPGA configuration, bitstream protection, bitstream encryption, bitstream authentication.

1 Introduction

FPGAs are widely used recently, and security on configuration bitstreams is of concern to both users and suppliers of configuration bitstreams (e.g., intellectual property vendors). Configuration bitstreams are exposed to the threats of piracy, reverse engineering and code theft. Moreover, a tampered configuration bitstream can be an FPGA virus [4], which is a hardware analogue of a computer virus. In order to defend against such threats, we need encryption/decryption and authentication of bitstreams at configuration. Some commercial FPGAs have a cryptographic circuit in it [1,8]. Also various methods that protect bitstream security have been proposed [2,3,5,6,7]. Most of them equip dedicated decryption and/or authentication circuits in an FPGA. This causes a hardware overhead. Note that circuits for asymmetric cryptography need large area. Because of the overhead, dedicated decryption and/or authentication circuits equipped in an FPGA are limited to symmetric key cryptography. On the other hand, a method that utilizes reconfigurable logic blocks to implement a decryption circuit was proposed [2]. This might solve the area overhead problem. However, the whole circuit cannot be encrypted because at the end of a configuration, the decryption circuit is replaced with a part of the target circuit whose bitstream is a plain text. This limits the size of the decryption circuit to be implemented since a large part of the target circuit cannot be encrypted if we implement a large decryption circuit. In addition, it needs a dedicated authentication circuit in order to authenticate the bitstreams of the decryption circuit, causing an area overhead.

In this paper, we propose a new configuration scheme such that the whole circuit can be encrypted while the proposed scheme causes only a small area overhead. In our scheme, a decryption circuit is implemented using reconfigurable logic blocks, and a bitstream of (a part of) the target circuit is decrypted using the decryption circuit. At the end of a configuration, the decryption circuit is replaced with the remaining part of the target circuit whose bitstream is one-time padded. Thus, the whole circuit can be securely downloaded into an FPGA. To do this, we use a part of the target circuit as a one-time pad for the bitstream to be loaded in the last phase of the configuration.

Since the proposed scheme can use a large reconfigurable area to implement a decryption circuit, it can handle computationally demanding cryptography such as public key cryptography.

This paper is organized as follows. In Sect. 2, we describe the proposed scheme. In Sect. 3, we discuss the security of the proposed scheme. In Sect. 4, we show how to enhance security and flexibility of the proposed scheme. In Sect. 5, we describe architecture requirements for realization of the proposed scheme. And Section 6 concludes the paper.

2 The Proposed Scheme

Our concern is to protect designs from piracy, reverse engineering and code theft. So we focus on encryption/decryption of bitstreams in this section, although our enhanced scheme can be used to authenticate bitstreams as we will describe later. That is, a bitstream supplier encrypts the bitstream of a circuit, then send it to a user. The user's FPGA (not the user!) decrypts the bitstream and the decrypted bitstream is stored in the configuration memory. The important point is decryption is completed within an FPGA, i.e., a user cannot access to the decrypted bitstream. To do this securely, only an authenticated (or built-in) decryption circuit should be implemented in an FPGA. We describe the proposed scheme in the following.

We divide Configurable Logic Blocks (CLBs) into two sets; One is the set that consists of CLBs chosen regularly skipping several blocks, and the other consists of the remaining CLBs. (See Fig. 1.) We also divide a bitstream into three bitstreams. The first one is a bitstream for configuring CLBs in set A, the second one is for CLBs in set B, and the third one is for the routing information.

Our main idea is as follows. We first implement a decryption circuit using CLBs in set A, decrypt a bitstream for set B and configure CLBs in set B by loading the decrypted bitstream. Then the decryption circuit is replaced with the remaining part of the target circuit by reconfiguring CLBs in set A. The bitstream used in the reconfiguration of CLBs in set A is one-time padded, and the decrypted bitstream for set B is used as the one-time pad. So we need

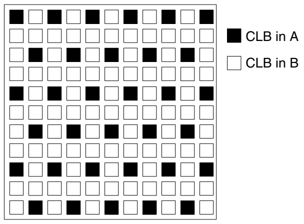

Fig. 1. Grouping CLBs

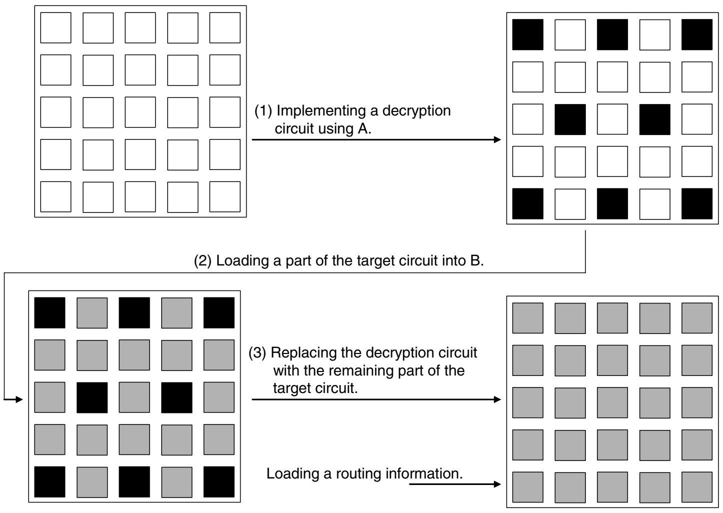
(1) Implementing a decryption circuit using A.
(2) Loading a part of the target circuit into B.
(3) Replacing the decryption circuit with the remaining part of the target circuit.
Loading a routing information.

Fig. 2. Configuration flow

no decryption circuit for reconfiguration of CLBs in set A; just XORing the encrypted bitstream and the one-time pad is enough to decrypt it. After finishing configuration of CLBs, routing information is loaded. The configuration flow is illustrated in Fig. 2.

In the following, we describe the configuration scheme in detail.

Phase 1: Implementation of a decryption circuit - Configuration of A
The configuration memory of each CLB in set A has initial values, which are fixed during manufacturing. It is initialized by the initial values at start-up of a configuration. This initial configuration implements the decryption circuit. Similarly, the routing configuration memory has initial values, and is also configured by initializing them.

Phase 2: Decryption of a bitstream for B - Configuration of B
An encrypted bitstream is passed to the decryption circuit. And then, the decrypted bitstream is loaded into CLBs in B, which implements (a part of) the target circuit.

Phase 3: Implementation of the remaining part of the circuit - Reconfiguration of A and reconfiguration of routing information
A configuration bitstream, which is encrypted by one-time pad, is loaded into CLBs in set A. Each CLB loads the configuration data by XORing them with the configuration data of the neighboring CLB, i.e., using the values stored in the configuration memory of the neighboring CLB as a one-time pad. And then, routing information, which is a plain text, is loaded.

Note that we can use large reconfigurable area in an FPGA to implement a decryption circuit, so that we can implement a large circuit such as public key cryptography.

3 Security of the Proposed Scheme

In this section, we discuss the security of the proposed scheme.

Confidentiality of the target circuit
The confidentiality of part B of the target circuit is guaranteed since its bitstream is encrypted. As for the confidentiality of part A of the target circuit, the one-time pad might be weak since it is not a random bit string but the configuration data of the neighboring CLBs. However, we can enhance the randomness of the one-time pad as we will describe later. The routing information is not encrypted in the above scheme. This is because for an evil user, obtaining the routing information alone makes no sense. However, for higher security, it can be encrypted as we will describe later.

Security of the embedded secret key
Our FPGA architecture has an embedded secret key, which is used by the decryption circuit in Phase 2. Only the decryption circuit in Phase 2 can access to the embedded secret key; After Phase 2, the connection port to the secret key is disabled. Note that the decryption circuit is fixed during manufacturing. Thus, the embedded secret key is securely managed.

4 Enhancing Security and Flexibility

In this section, we describe how the security of the bitstream for part A and the routing information can be enhanced. We also describe an enhanced configuration scheme that can implement an arbitrary decryption and/or authentication circuit instead of the built-in decryption circuit.

4.1 Encryption of the Bitstream for A and the Routing Information

As we described in Sect. 2, the secret key (the one-time pad) for the encryption of the bitstream for A is not a random bit string but a configuration bitstream stored in a neighboring CLB. However, we can enhance the randomness of the key by generating a key by bitwise XORing the bit strings stored in the neighboring four CLBs instead of using a single bit string in a neighboring CLB as a key.

Routing information can also be encrypted similarly by using configuration data stored in CLBs that are placed around the routing configuration memory in order to create a one-time pad.

4.2 Enhancing Flexibility of Decryption Algorithms

In the configuration scheme described in Sect. 2, we can use only the built-in decryption circuit in Phase 2. However, we can enhance the configuration scheme so that the authorized suppliers of configuration bitstreams (e.g., the authorized intellectual property vendors) can distribute arbitrary decryption circuits for the use of the decryption in Phase 2. Moreover, a circuit to be implemented in Phase 2 is not limited to a decryption circuit. We can implement an arbitrary cryptographic circuit such as an authentication circuit. We describe the enhanced configuration scheme below.

We divide CLBs into two sets, A and B, as in Sect. 2. We also divide set B into two sets, A' and B', similarly. Then, we replace Phase 1 with Phase $1'$ below.

Phase $1'$: Implementation of an *authenticated* decryption circuit
First, we implement an authentication circuit using CLBs in A'. Similarly to the case of the decryption circuit described in Sect. 2, the authentication circuit is built-in, and is configured just by initializing the configuration memory. The authentication keys that are generated by the authorized suppliers of configuration bitstreams are also embedded, i.e., they are fixed during manufacturing. Then it loads an arbitrary decryption and/or authentication circuit into A while verifying the MAC for the bitstream of it. Thus only the decryption and/or authentication circuits authorized by the suppliers of configuration bitstreams can be implemented.

The remaining phases are the same as in the configuration scheme described in Sect. 2. The configuration flow is illustrated in Fig. 3.

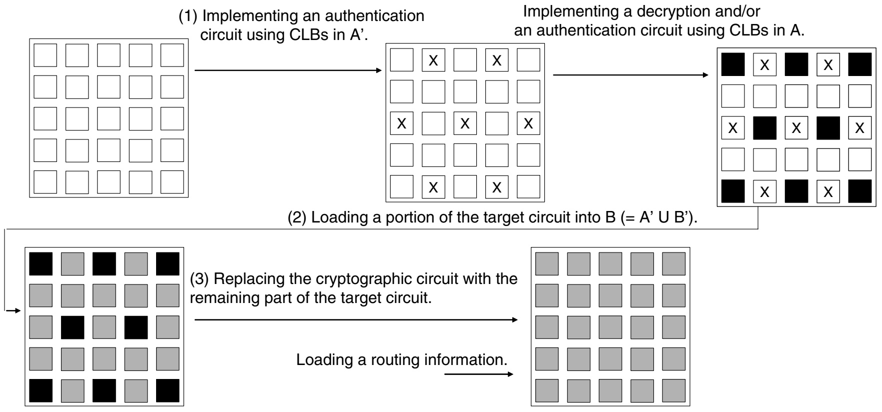
(1) Implementing an authentication circuit using CLBs in A'.
Implementing a decryption and/or an authentication circuit using CLBs in A.
(2) Loading a portion of the target circuit into B (= A' U B').
(3) Replacing the cryptographic circuit with the remaining part of the target circuit.
Loading a routing information.

Fig. 3. Enhanced configuration scheme

5 Architecture Requirements

In this section, we describe architecture requirements for realization of the proposed configuration scheme. In order to configure A, A', B and routing information, four configuration systems (chains of configuration memories) are needed. Each configuration memory of a CLB in A' has its initial value, and can be initialized at the start-up of a configuration. A memory controller for dealing with this initialization is needed. As for the management of the embedded secret keys, a connection port to the embedded secret keys can be disabled after finishing Phase 2.

All the above features together with a controller that manages the whole configuration procedure realizes the proposed configuration scheme. All the above features are easy to implement, and the area overhead is very small.

6 Conclusion

We proposed a configuration scheme that can securely download a bitstream into an FPGA. By using the proposed scheme, we can encrypt the whole circuit with only a small area overhead. Also the proposed scheme has flexibility; An arbitrary cryptographic circuit can be implemented.

References

1. Altera Corp., `http://www.altera.com/`
2. Bossuet, L., Gogniat, G., Burleson, W.: Dynamically configurable security for SRAM FPGA bitstreams. International Journal of Embedded Systems 2(1/2), 73–85 (2006)
3. Drimer, S.: Authentication of FPGA Bitstreams: why and how. In: Diniz, P.C., Marques, E., Bertels, K., Fernandes, M.M., Cardoso, J.M.P. (eds.) ARCS 2007. LNCS, vol. 4419, pp. 73–84. Springer, Heidelberg (2007)
4. Hadžić, I., Udani, S., Smith, J.M.: FPGA viruses. In: Lysaght, P., Irvine, J., Hartenstein, R.W. (eds.) FPL 1999. LNCS, vol. 1673, pp. 291–300. Springer, Heidelberg (1999)
5. Kean, T.: Secure configuration of field programmable gate arrays. In: Proc. of 9th IEEE Symposium on Field Programmable Custom Computing Machines (FCCM2001), pp. 259–260 (2001)
6. Kean, T.: Secure configuration of field programmable gate arrays. In: Brebner, G., Woods, R. (eds.) FPL 2001. LNCS, vol. 2147, Springer, Heidelberg (2001)
7. Parelkar, M.M., Gaj, K.: Implementation of EAX mode of operation for FPGA bitstream encryption and authentication. In: Proc. of IEEE International Conference on Field-Programmable Technology, pp. 335–336 (2005)
8. Xilinx Inc., `http://www.xilinx.com/`