Keywords: FPGA counterfeit protection, DeepCover Secure Authenticator, challenge-and-response, IP Protection

APPLICATION NOTE 5803

SECURE YOUR FPGA SYSTEM USING A DEEPCOVER SECURE AUTHENTICATOR

By: Michael D'Onofrio

Abstract: This application note describes how designers can secure their Xilinx® FPGA implementation, protect IP, and prevent attached peripheral counterfeiting. Designers can achieve this security by using one of the reference designs described in this application note. These designs implement either SHA-256 or ECDSA challenge-and-response secure authentication between the FPGA and a DeepCover® Secure Authenticator.

Introduction

This application note focuses on SHA-256 authentication reference designs (RDs) using MAXREFDES34# for 1-Wire and MAXREFDES43# for I^{2}C. Each of the reference designs (RD) describes either a Verilog® implementation of a state machine or a "C" bare-metal implementation in a microcontroller. Each implementation is tested on the corresponding FPGA demo board.

For 1-Wire® designs, the reference code defines a combined SHA-256 processor and 1-Wire master on the host FPGA. For I^{2}C designs, the reference code defines a SHA-256 processor and utilizes existing FPGA I^{2}C protocol. The RDs use one of the following secure authenticators:

- DS28E15 DeepCover Secure Authenticator with 1-Wire SHA-256 and 512-Bit User EEPROM
- DS28C22 DeepCover Secure Memory with I^{2}C SHA-256 Authentication, Encryption, and 3Kb User EEPROM
- DS28E35 DeepCover Secure Authenticator with 1-Wire ECDSA and 1Kb User EEPROM

To interface the Maxim secure authenticator IC with the FPGA demo board, each RD ships with a compatible plugin module—a Pmod™ port standard developed by Digilent, Inc. The corresponding IC is soldered onto the Pmod board. Regarding the 1-Wire designs, with minor tweaking a SHA-256 authenticator with larger memory density could be used, such as the DS28E22 or DS28E25.

The section **Why Apply Security to an FPGA System?** details important security concerns that designers face. The following section entitled **Securing Your FPGA** demonstrates how authentication actually secures FPGA systems.

Why Apply Security to an FPGA System?

1. **Counterfeit Protection of a Peripheral**
   Systems that use an FPGA to communicate and operate replacement peripherals, consumables, modules, or sensors are commonly targeted by counterfeiters or unauthorized aftermarket companies. These counterfeit versions of a peripheral can introduce safety concerns, reduce quality to the application and, generally, negatively impact the OEM solution. Furthermore, the OEM loses the revenue of peripherals to these counterfeiters. Introducing secure authentication into the solution enables the FPGA to assure peripheral authenticity and to take application-specific action if a counterfeit is detected. As shown in Figure 1, a challenge-and-response sequence between the system and attached peripheral is exercised to confirm authenticity.

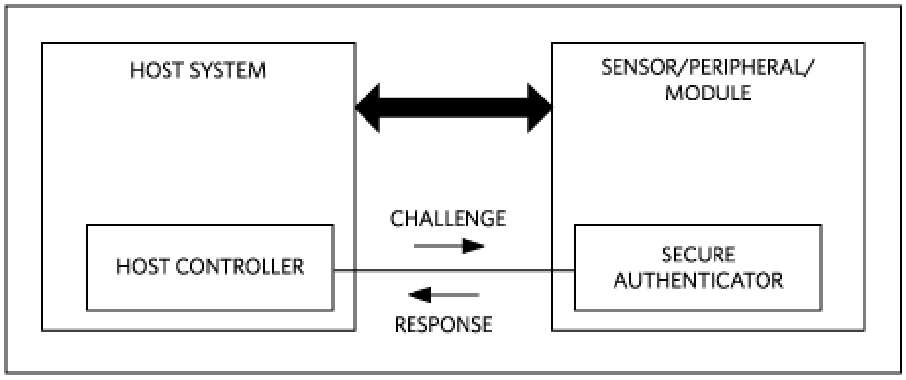

Figure 1. Testing for authenticity with a challenge-and-response sequence.

2. **FPGA IP Protection and Implementation Protection**
   Spartan 6. Static-RAM-based (SRAM) FPGAs have few safeguards to protect that IP (i.e., the configuration data or the FPGA implementation) against illegal copying and theft. The reason is that once the data is loaded, it is held in SRAM memory cells, which can easily be probed to determine their contents. In addition, without some type of security mechanism to protect the configuration data or bit file before it is loaded into the chip, that data is open to snooping. Prowling through that data is possible because the bit stream is usually stored in a separate memory chip read by the FPGA at power-up when it loads its configuration pattern. A cloner could simply copy the configuration file and create clones of the original. These types of FPGAs do not have built-in encryption that would otherwise protect the configuration file from copying.
   One way to make the SRAM-based FPGAs more secure is to leverage multichip packaging and mount the nonvolatile (NV) memory inside a package along with the FPGA. Yet if someone opens the package, the data interface between the memory and the FPGA is exposed and the configuration pattern can be compromised. Multichip packaging can also be an expensive endeavor.

   Xilinx 7-Series. High-end FPGAs protect internal IP by using security keys and bit-stream encryption. These protection mechanisms greatly mitigate the risk of the IP on the bit file being snooped. Furthermore, these FPGAs do not store sensitive data in insecure SRAM. However, implementing these built-in cryptographic security mechanisms can become cumbersome. Added manufacturing steps mean greater cost. But more importantly, if these cryptographic keys are programmed during contract manufacturing, then the subcontractor will know the keys and you can no longer be assured that your IP is safe.

Solution: Regrettably, for IP protection, the FPGA must offer proper bit file protection so the bit file cannot be modified. If the FPGA does have proper bit file protection, then challenge-and-response authentication can provide IP protection. So an authentication sequence can then be added to the FPGA implementation. If there is a Maxim secure authenticator with a valid security key on the bus, then the FPGA knows that the system is authentic. This is specifically only for FPGAs with built-in encryption of its bit file. In the Maxim solution, the RD calls for the authentication key to be embedded securely into the FPGA implementation code so that the subcontractor would never know this key.

3. **Feature Management, License Management, and Overbuild Protection**

   Feature Management. To decrease design time and effort, designers will create fully featured FPGA systems and de-feature certain aspects using firmware to achieve different price points or feature levels. This, however, creates a new problem: a smart customer who needs several fully featured systems could just buy one fully featured unit and several units with reduced features. Then, copying the software, the simpler units behave like the fully featured unit but for a lower price, shortchanging the system vendor.

   License Management. At other times, companies create and sell RDs. These are subsequently bought, licensed to, and manufactured by third parties. The RD vendors require barriers to prevent unauthorized use of the intellectual property. For revenue reasons, it is also necessary to track and confirm the number of reference uses.

   Overbuild Protection. In all cases, designers might wish to build their end product using third-party contract manufacturing. Subcontractors can be a great extension of a supply chain, and they can manufacture embedded systems efficiently and cost effectively. However, less scrupulous contract manufacturers (CMs) have been known to build more widgets than contracted. Then they can produce bootleg products of the same quality and authenticity as the originals. Indeed, by overbuilding, an unscrupulous CM freeloads on all of the R&D and marketing costs that the designer incurred.

   Solution. A practical solution for all three of these security issues is secure authentication. For *Feature Management*, device settings would only be stored in the user EEPROM of the secure authenticator, and a secure challenge and response would be required to read these settings. For *License Management*, a RD vendor would require a secure authenticator with a valid secret to be supplied to the licensee or third-party manufacturer. The RD would be made unable to operate without a secure challenge and response. Licensees would be supplied to the secure authenticator through one of two secure methods: 1) preprogrammed by the company licensing the reference, or 2) preprogrammed by Maxim per the licensing company's input and then delivered to the third-party manufacturer. In either case, the number of devices sent to the licensee or manufacturer is known and can be used to validate license fees. *Overbuild Protection* works just like License Management in that a contract manufacturer would only be able to build as many units as it can procure secure authenticators. A designer would be able to control how many authenticators that the CM can procure by working with Maxim.

For more information on all the potential applications of secure authentication, refer to application note 3675, "Protect Your R&D Investment with Secure Authentication."

How Does the Authentication Work?

To implement the authentication in the most secure manner, the following are used as general guidelines:

- Ensure that the random challenge is a cryptographically secure random number.
- Know a secret key (called "FPGA Secret") that can be used for internal operations, but cannot be discovered from outside.
- Compute a hash that involves the secret key, a random number and additional data, just like the secure memory.
- Compare hash results.

In the context of the FPGA environment, the way that the challenge-and-response authentication works is shown in the following numbered statements. The letters (e.g., A, B, C) correspond to data flows in **Figure 2**. This section describes SHA-256 symmetric authentication.

1. Generate a random number and send it as a challenge (A) to the secure authenticator.
2. Instruct the authenticator to compute a hash based on its secret key, the challenge A, its unique ID, and other fixed data. The hash is the output of the algorithm block (B).
3. Compute a hash (C), based on the same input and constants used by the secure authenticator and the FPGA's secret key.
4. Take the hash computed by the secure authenticator (B) as the response and compare it to the expected response (C).

If the expected response and the actual response are identical, then the FPGA knows that the authenticator is genuine. If genuine, your security goals—IP protection, counterfeit protection, etc.—are achieved.

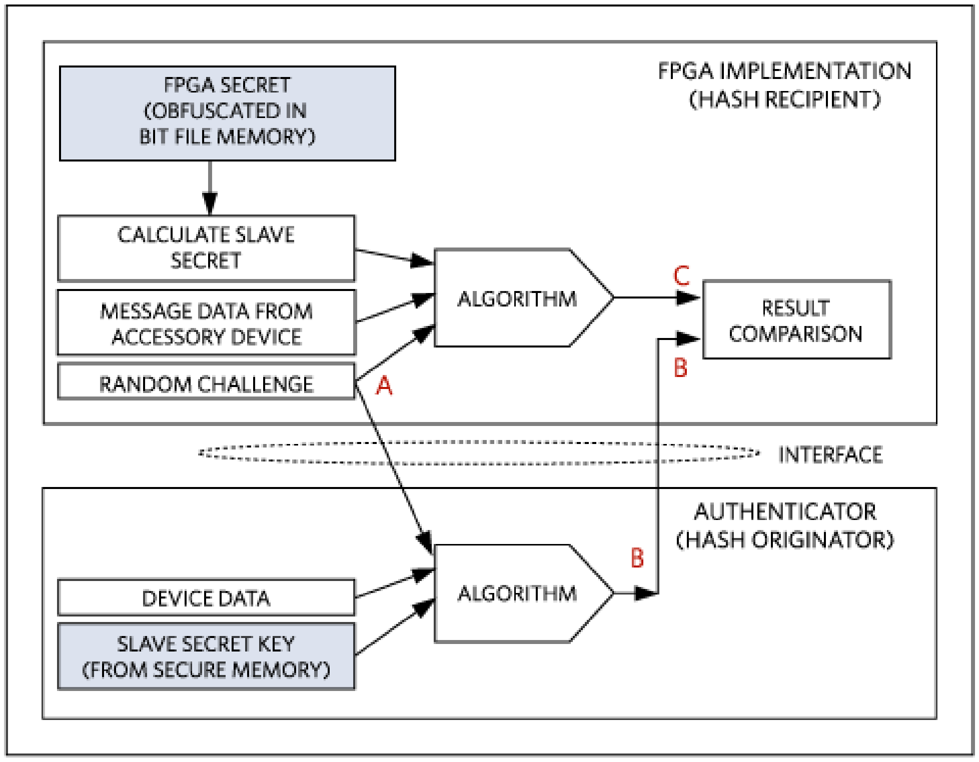
FPGA SECRET (OBFUSCATED IN BIT FILE MEMORY)
FPGA IMPLEMENTATION (HASH RECIPIENT)
CALCULATE SLAVE SECRET
MESSAGE DATA FROM ACCESSORY DEVICE
RANDOM CHALLENGE
ALGORITHM
C
RESULT COMPARISON
B
A
INTERFACE
AUTHENTICATOR (HASH ORIGINATOR)
B
DEVICE DATA
ALGORITHM
SLAVE SECRET KEY (FROM SECURE MEMORY)

Figure 2. Challenge-and-response authentication flows in greater detail. Proves authenticity of hash originator —the secure authenticator.

As previously mentioned, authentication between an FPGA and a secure authenticator achieves the security of points 1 through 3 of the **Why Apply Security to an FPGA?** section. Now, let's observe how to physically set up a security system based on the following block diagrams.

Counterfeit Protection of a Peripheral

To implement counterfeit protection, a system configuration, as shown in **Figure 3**, should be used. The FPGA, and its corresponding security implementation, resides in an embedded system. Some peripheral device—perhaps a sensor, disposable, consumable, or another embedded system—is the object that, as a designer, we wish to secure. The Maxim secure authentication IC resides in this peripheral device.

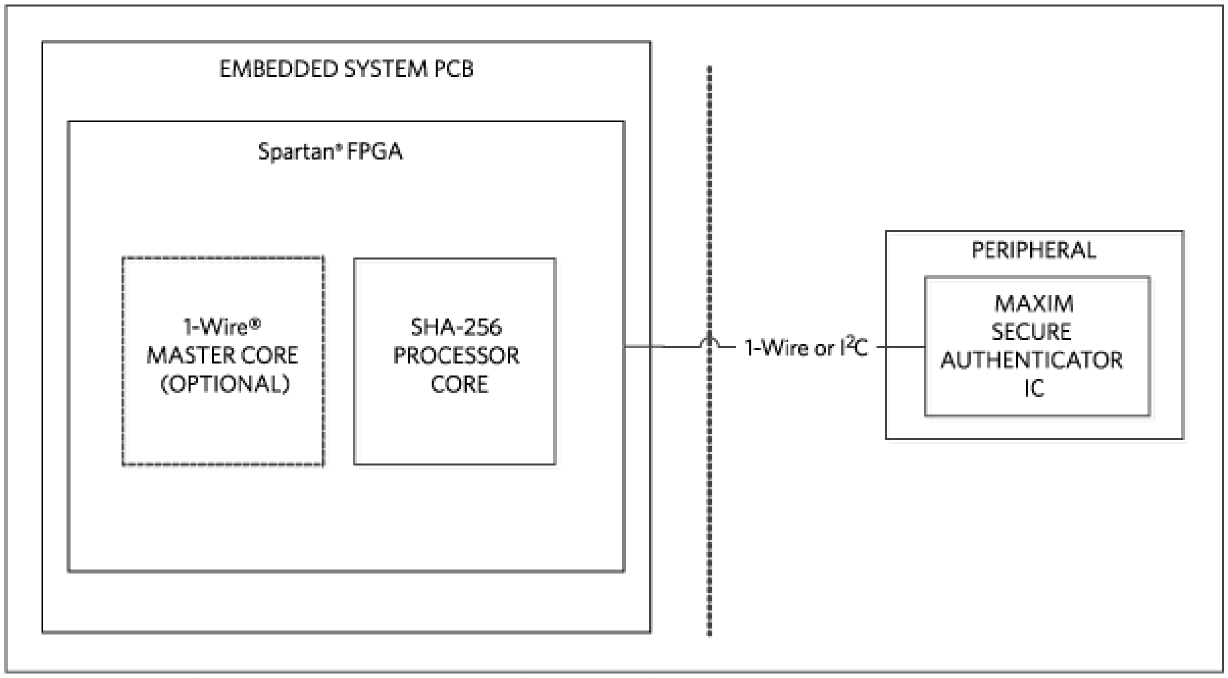

Figure 3. Block diagram for counterfeit protection of peripherals.

IP Protection, Feature Management, License Management, and Overbuild Protection

For all other security needs described in the section entitled, **Why Apply Security to an FPGA?**, the hardware setup shown in **Figure 4** should be used. The FPGA, and its corresponding security implementation, resides in an embedded system along with the Maxim secure authentication IC.

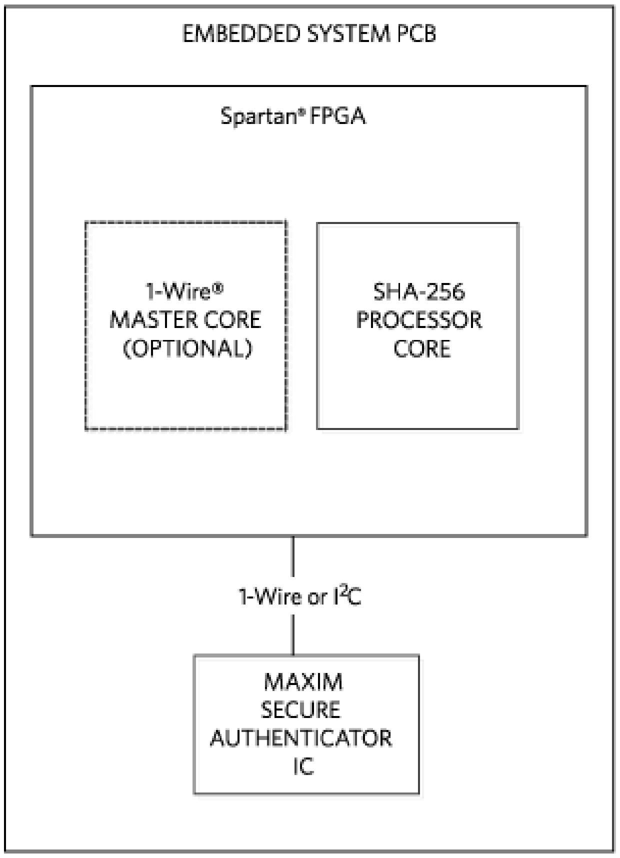

Figure 4. Block diagram for IP protection, and other applications.

Available Reference Designs

Reference Design	FPGA	Demo Board	Implementation	Authentication IC Used	Interface of Authentication IC
MAXREFDES34#	Xilinx Spartan-6	Avnet® Spartan-6 LX9 MicroBoard	Verilog	DS28E15	1-Wire
Xilinx Zynq	MicroZed	'C'			
MAXREFDES43#	Xilinx Zynq	ZedBoard™	'C'	DS28C22	I^2C
MAXREFDES44#	Xilinx Zynq	MicroZed	'C'	DS28E35	1-Wire

Reference Design-Specific Notes

MAXREFDES43# with the DS28C22 is unique because it implements bidirectional, small-message encryption of sensitive data communicated between the FPGA and the DS28C22. This encryption feature is optional; the most common use is when the DS28C22 is inside an attached peripheral subsystem. Or, perhaps it is necessary to encrypt sensitive sensor data, calibration data, feature setting data, or personal data (e.g., patient heart rate or SSN).

For more information on the encryption feature of the DS28C22, refer to application note 5785, "Implement Heightened Security with a SHA-256 Master/Slave Authentication System."

MAXREFDES44# with the DS28E35 utilizes asymmetric ECDSA authentication instead of implementing symmetric SHA-256 authentication. For symmetric authentication, both the FPGA and authentication IC must store the same key, and therefore the sensitive secret key data resides on both sides of the system. However, with asymmetric authentication, the FPGA holds only the public key, and the DS28E35 holds the private key. The only sensitive data is the private key; the public key need not be secured.

Using asymmetric cryptography could be more attractive from two standpoints:

- You are licensing your product to your customers or using multiple contract manufacturers. Asymmetric authentication offers a great management tool for adding new licenses or removing existing ones due to the usage of certificates. Refer to application note 5767, "The Fundamentals of an ECDSA Authentication System," for more information on certificates.
- Implementations with poor security of FPGA configuration/bit file. Using SHA-256 symmetric authentication for these applications could be risky if the FPGA secret key is exposed; however, note that for relevant reference designs (e.g., MAXREFDES34# for Spartan-6 FPGAs), extra steps have been taken to secure the FPGA-side secret key. Indeed if you are not implementing the built-in FPGA encryption feature (on applicable FPGAs like Xilinx Zynq) or if you are using a previous generation FPGA that does not secure the configuration/bit file, then using the DS28E35 is ideal.

MAXREFDES34# with DS28E15 implements symmetric authentication using SHA-256. These authentication schemes require both the FPGA-side secret keys and secure authenticator keys to be secure. For the following reasons, the MAXREFDES34# successfully secures the FPGA-side secret keys:

- The secret key is stored in the FPGA bit file in a FLASH device. It is very difficult to reverse the bit file from bits back to Verilog code (the Spartan-6 reference was written in Verliog). Therefore, the obscurity and unwieldiness of the bit file provide the first layer of security.
- The secret on the bit file is not the exact same secret as that stored in the DS28E15. The FPGA computes the DS28E15 secret based on its own version of the secret.
- There are other proprietary FPGA secret protection techniques that cannot be disclosed in this application note.

Summary

Designers of FPGA embedded systems face many potential security threats including counterfeiting of peripherals and copying of FPGA implementation, among others. Using a Maxim secure authenticator along with either of these reference designs protects FPGA systems from these potential issues.

Related Parts		
DS28C22	DeepCover Secure Memory with I^{2}C SHA-256 and 3Kb User EEPROM	Free Samples
DS28E15	DeepCover Secure Authenticator with 1-Wire SHA-256 and 512-Bit User EEPROM	Free Samples
DS28E22	DeepCover Secure Authenticator with 1-Wire SHA-256 and 2Kb User EEPROM	Free Samples
DS28E25	DeepCover Secure Authenticator with 1-Wire SHA-256 and 4Kb User EEPROM	Free Samples
DS28E35	DeepCover Secure Authenticator with 1-Wire ECDSA and 1Kb User EEPROM	Free Samples
DS28EL15	DeepCover Secure Authenticator with 1-Wire SHA-256 and 512-Bit User EEPROM	Free Samples
DS28EL22	DeepCover Secure Authenticator with 1-Wire SHA-256 and 2Kb User EEPROM	Free Samples
DS28EL25	DeepCover Secure Authenticator with 1-Wire SHA-256 and 4Kb User EEPROM	Free Samples

More Information
For Technical Support: http://www.maximintegrated.com/en/support
For Samples: http://www.maximintegrated.com/en/samples
Other Questions and Comments: http://www.maximintegrated.com/en/contact

Application Note 5803: http://www.maximintegrated.com/en/an5803
APPLICATION NOTE 5803, AN5803, AN 5803, APP5803, Appnote5803, Appnote 5803