



# Taking Device DNA Technology to the Next Level

Xilinx and Helion work together to make low-cost FPGA designs more secure.

by Maureen Smerdon  
Strategic Marketing Manager  
Xilinx, Inc.  
[maureen.smerdon@xilinx.com](mailto:maureen.smerdon@xilinx.com)

Graeme Durant  
CEO  
Helion Technology Limited  
[graeme@heliontech.com](mailto:graeme@heliontech.com)

In today's world, device cloning and unauthorized overbuilding have driven design security to a higher level of importance. In consumer electronics applications, design security has become as important as the media encryption flowing across the device. Example applications include broadband access, wireless networking, routers, high-definition DVD, and DVRs.

In 2007, Xilinx introduced its Device DNA technology, offering a range of security features to safeguard against reverse engineering, cloning, and unauthorized overbuilding. Xilinx offers this revolutionary capability in its latest low-cost Spartan™-3 FPGAs, specifically Spartan-3A, Spartan-3AN, and Spartan-3A DSP devices.

To expand the capabilities of this technology, Xilinx worked closely with Helion, a Xilinx Alliance IP partner specializing in data encryption and security solutions, to develop Device DNA Checker, an IP block specifically designed to work in conjunction with Spartan-3 devices to provide designers with an even more solid and robust security solution.

## Device DNA

Before understanding how the Helion Device DNA Checker works together with Spartan-3 devices, let's look at the core of Xilinx Device DNA technology.

Xilinx Device DNA technology is a permanent, factory-set ID code that is different in every device. Starting at 57 bits and programmable up to any bit length to meet user requirements, the Device DNA enables advanced 64-bit, 128-bit, or even more complex 256-bit algorithms. This unique ID can be used to tie a design to a specific FPGA.

Every design can hold a unique authentication algorithm to further increase design security and reduce reverse engineering and potential threats. This allows you to precisely select the complexity of the algorithm and

directly control the amount of security needed. With this flexibility, you can spend as much or as little on security as necessary, depending on the security needs of your specific application.

The authentication algorithm you choose is implemented in the FPGA and takes in the ID value and generates a unique result for that ID. The result is then stored wherever you choose, such as in external memory or internal flash (for Spartan-3AN FPGA devices only). Each time the FPGA is powered, the generated and stored results must match in order to authorize that device. The algorithm is the secret to the security because it is known only to you, and is almost impossible to determine without access to the original design source.

Designers have full flexibility in customizing algorithms for both authentication as well as responses to authentication failures. With its embedded flash, the Spartan-3AN device further enhances security by hiding any configuration communication from the outside, making it extremely difficult to understand the design contained within the FPGA.

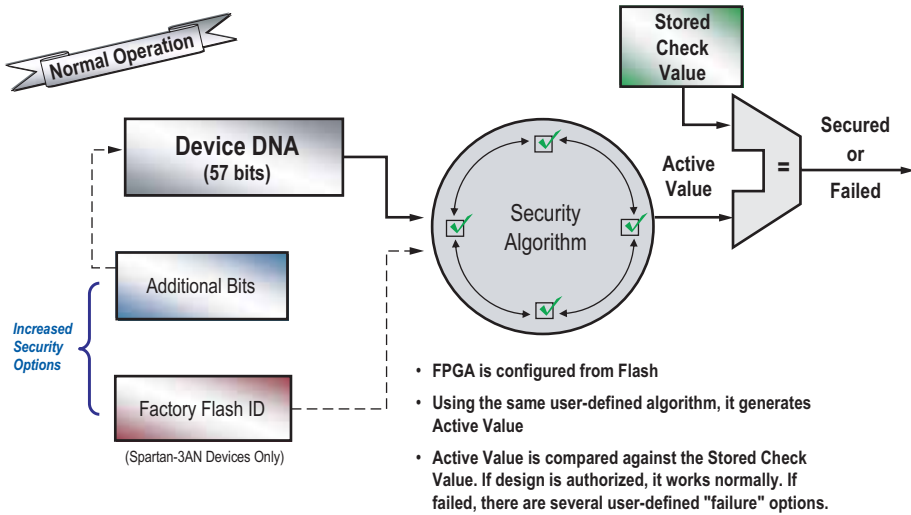


Figure 1 – Device DNA design-level security

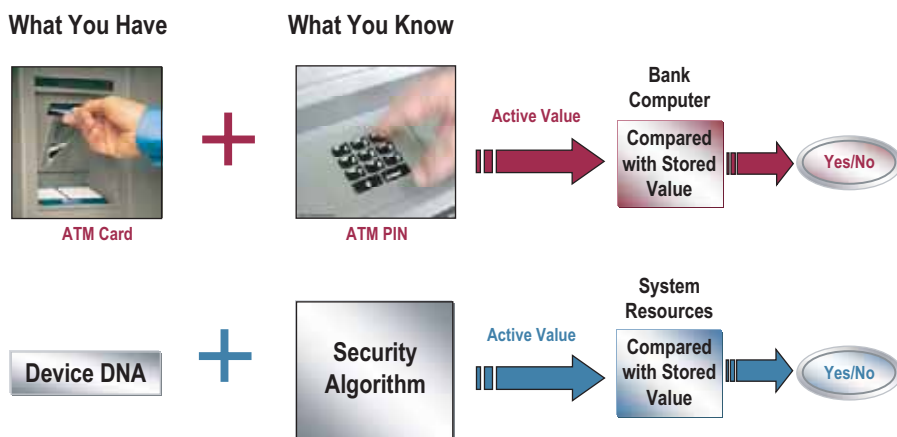


Figure 2 – ATM transactions and Device DNA security flow

This security or licensing process is designed with complete flexibility in mind. You can easily change the security or licensing process from model to model, thus increasing design security. The read-only Device DNA is accessible through either the external JTAG port or the internal DNA port for easy connection to the security algorithm.

If a cloner or overbuilder copies the bitstream and places it into another FPGA, the Device DNA of the new FPGA will be different. After using the algorithm to check the Device DNA, the design will return an unauthorized or a failed result, allowing the user or designer to determine how to respond to the security breach.

Figure 1 shows a graphical representation of the security flow.

The Device DNA security process is like an ATM transaction. To withdraw money from an ATM, you insert your ATM card and enter your PIN on the touch pad. If your card and associated PIN match the ID stored at the bank, your transaction is approved and your money is available. If the match fails, your transaction is rejected and you do not get your money. Figure 2 shows the how the ATM transaction compares to the security flow.

### Unauthorized Operation

Users have more flexibility in the determination of what happens when a design

is non-authorized or illegitimate. The simplest way is to disable functionality. This can easily be accomplished by utilizing global control signals such as resets, 3-states, or clock gating.

During normal operation, the device is powered up and the bitstream is loaded for configuration of the FPGA. The security algorithm reads the Device DNA and generates an active value. It then compares the active value with the check value stored during the initial setup. If the check value is equal to the active value, the device will operate normally.

You can design your product to respond in one of the following ways when the two values do not match:

- No functionality. The design completely ceases to function. This can be easily implemented in a Spartan-3 FPGA by using global control signals like 3-state, gated clocks, or flip-flop clock enable.
- Limited functionality. The design has partial or basic operation but key functionality is disabled or bypassed. This response allows a third-party test house or contract manufacturer to build and test while preventing overbuilding. It also allows the system to be run in evaluation or demo mode.
- Time bomb. The design operates with full functionality for a predetermined amount of time before shutting off. This response allows a third-party test house or contract manufacturer to build and test. It also allows the system to operate in demo mode or for IP evaluation.
- Self-destruction (Spartan-3AN devices only). Uses flash sector erase and lock-down protection to erase all sectors and permanently lock flash memory to all zeros. This response prevents repeated unauthorized access attempts.

### Authentication Algorithms

The techniques necessary to create a solution using Device DNA and protect a product against overbuilding and cloning are described both in this article and in Xilinx white papers such as WP266, “Security

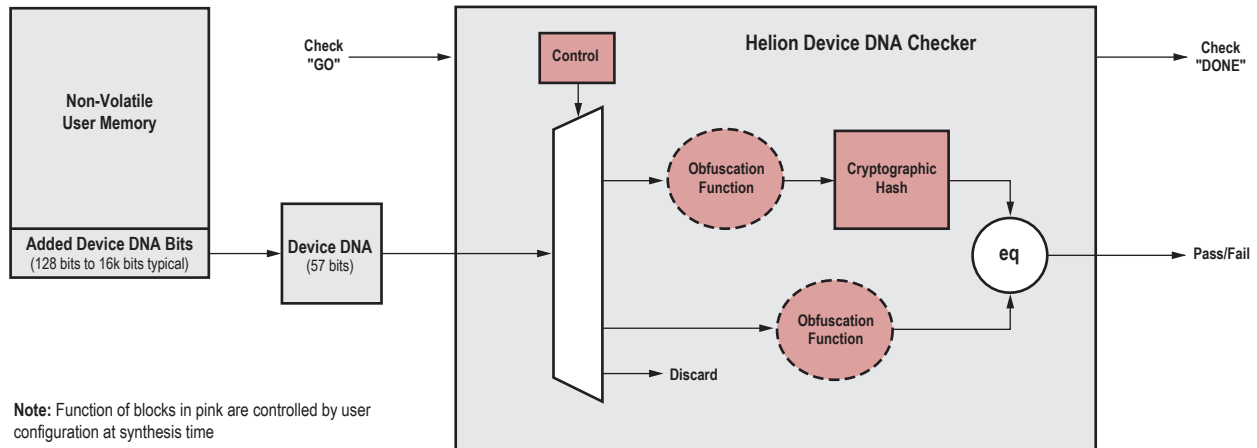


Figure 3 – Device DNA Checker block diagram

Solutions Using Spartan-3 Generation FPGAs” ([www.xilinx.com/documentation/white\\_papers/wp266.pdf](http://www.xilinx.com/documentation/white_papers/wp266.pdf)). However, actually implementing such a solution quickly and securely and ensuring that it does not use excessive logic resources in your device can be time-consuming. Thus, there is a strong case for using a ready-made and tested IP block specifically designed for the purpose.

Helion, using its well-established cryptographic IP and expertise, developed a ready-made solution comprising a pair of IP cores, allowing designers to easily implement Device DNA technology in their designs.

The Helion approach uses a selection of strong cryptographic functions to implement the security algorithm and various obfuscation techniques to ensure that any reverse-engineering attacks are made suitably difficult. Most importantly, the design is highly parameterized so that each user can make their implementation unique to them. Other customers using these IP blocks – even Helion itself – cannot work around the scheme.

### The Helion Device DNA Checker

The Helion Device DNA Checker comprises a ready-made IP building block that is dropped into the user design to sit alongside the main FPGA application. It has relatively low resource requirements and is intended to coexist easily with the main design. Its interface is simple; it requires an input from the Device DNA block in the FPGA, and after processing generates a

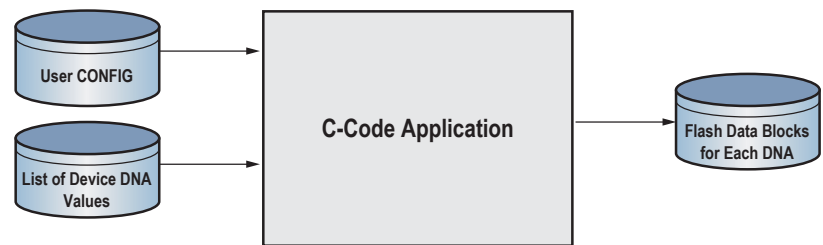


Figure 4 – Using Device DNA in a software application

“pass/fail” flag. This flag can be used to degrade normal operation of the rest of the design if the check does not pass. Simple “go” and “done” signals start the check process. Figure 3 shows a block diagram of the Helion Device DNA Checker.

Helion’s Device DNA Checker uses both the Device DNA code plus additional check bits stored in non-volatile system memory to enhance security. These additional bits are a mixed-up combination of randomly generated bits that form part of the check algorithm; randomly generated bits that are ignored (sometimes called “salt” bits); and the final stored check code itself. This additional data is simply streamed into the Checker after the Device DNA bits via the cascade input to the Device DNA block.

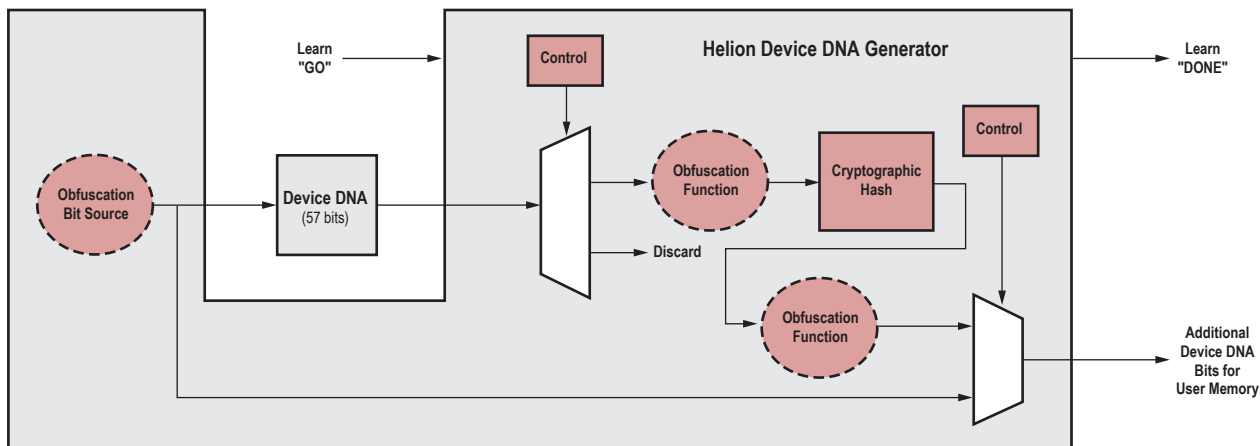
So that each customer using this solution can be confident that their protection is unique to them, the exact operation of the Checker is highly parameterized and not fixed by Helion. You set your own secret “configuration value” when the

Checker block is synthesized. This is effectively a long string of bits that define the low-level operation of each part of the check algorithm used in the block. Because this is set at synthesis time, the configuration value can result in significantly different logic for each user.

Neither Helion nor any other user of this technology can replicate the algorithm without access to the secret configuration, adding another layer of protection into the system. In the diagram, the functions of the blocks shown in pink are all affected by this secret compile-time configuration value.

### Internal Operation

Within the Checker, the input stream goes one of three ways. It is either used in the cryptographic hash, used as the stored check code (to be compared to the generated active code after the hash is complete), or discarded. The choice of which bits go where is determined by the user configuration previously outlined and is different for each user.



Note: Function of blocks in pink are controlled by user configuration at synthesis time

Figure 5 – Using Device DNA as a hardware download

Any bits destined for hashing or for the check code go through “obfuscation” functions before being used. These are intended to further obscure the processing being performed. Their exact definition is again driven by the user configuration outlined above, so each user’s implementation will be different. Even the choice of cryptographic hash function used is determined by your secret configuration. Once the appropriate bits have been hashed to form an active code and the stored check code has been regenerated, the two are compared and the final pass/fail result is indicated.

### Getting Authorized

For each legitimate product that has a unique Device DNA and contains the DNA checker, the product must be activated by generating the additional Device DNA bits to be stored in non-volatile memory. This can be achieved in two ways:

1. Software application. In this case, a software application supplied by Helion models the algorithms used in the Checker to calculate the necessary additional Device DNA data. It takes in a list of authorized Device DNA values and a copy of the customer’s secret compile-time configuration and generates data sets to be placed into the systems’ non-volatile memories (see Figure 4). This application could be used remotely to generate data sets

on demand, perhaps automatically over the Internet. Clearly this software and the user-configuration information must only be distributed to highly trusted parties; otherwise the whole scheme is compromised. However, remotely enabling manufactured units may be a great way to achieve separation between trusted and non-trusted parties in the manufacturing process.

2. Hardware download. In this case, an optional second IP block called the Helion Device DNA Generator implements the same algorithms as the Checker to generate the necessary additional Device DNA data (Figure 5). You would put together a special programming bitstream incorporating this block using the same user configuration as the Checker at synthesis time. This does not include any of your normal design, as it is only required for configuring non-volatile user memory with the appropriate additional

Device DNA data. This FPGA image could be self-contained, taking the data generated by the Helion block and burning it into the appropriate non-volatile memory, thereby enabling that unit.

Again, this special bitstream must only be distributed to highly trusted parties. It would need to be loaded once into the manufactured hardware to authorize the unit, so it cannot be used remotely. This may be more appropriate in some manufacturing processes than the alternative software approach, however.

### Conclusion

Helion’s solution for accessing the benefits of Xilinx Spartan-3 FPGA Device DNA technology is both off-the-shelf yet unique to each user. It is easy to use and requires minimal device resources. It can be deployed in an existing design very quickly, offering the benefits of protection against overbuilding and cloning during outsourced manufacture. 🌈

**TAKE THE NEXT STEP** (Digital Edition: [www.xcellpublications.com/subscribe/](http://www.xcellpublications.com/subscribe/))

- Learn more about the Spartan-3A device family or the Spartan-3AN device family.
- View a security demo on the Spartan-3AN device family.
- Learn more about Helion, [www.heliontech.com](http://www.heliontech.com).