WP267 (v1.0) August 15, 2007

Advanced Security Schemes for Spartan-3A/3AN/3A DSP FPGAs

By: Glenn Crow

FPGAs provide the ability to integrate and support new protocols and standards with ease, as well as product customization while still delivering rapid time to market. With the internet and the global market, outsourcing manufacturing has become more popular making security a bigger factor. As stated in articles published by industry leaders, reverse engineering, cloning, overbuilding, and tampering have become major security issues. Experts estimate that each year multiple billions of dollars in revenue are lost due to counterfeiting. These goods threaten the economy and have a significant effect worldwide in the consumer markets according to the Anti-counterfeiting Coalition. This white paper identifies the top design security threats, explores the advanced security options, and describes how new, low-cost Spartan™-3A, Spartan-3AN, and Spartan-3A DSP FPGAs from Xilinx can help protect your products and profits.

© 2007 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at http://www.xilinx.com/legal.htm. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

What are the Top Security Breaches?

The top security breaches that designs face today are *reverse engineering*, *overbuilding*, *cloning*, and *tampering*.

Reverse engineering occurs when a thief takes your design with the intent of recreating or rebuilding a competitive product and selling it on the open market. The effects of reverse engineering are that the perpetrator can build the design much faster, and minimize Research and Development costs. This has been the most common threat since the genesis of the electronics industry.

Overbuilding is a potential concern in an outsourcing business model. In this situation, what can occur is unauthorized overbuilding of product that is then sold through other channels without the permission of the original equipment manufacturer. The obvious challenge here is that this can have very adverse ramifications once this product hits the market. Usually, the "overbuilt" products are sold at a lower cost with a much faster time-to-market.

Cloning is when a thief creates a duplicate of your design, Intellectual Property (IP), or product under the same or different label. The obvious benefit to the cloner is that they incur no Research and Development costs and have a drastically reduced time-to-market for the cloned product.

Tampering is the modification and/or replacement of the original design to gain access to unauthorized services, to steal sensitive data, or to sabotage an application. Tampering is a huge concern for financial, defense, and premium service audio/video media providers.

Security Schemes in Spartan-3A/3AN/3A DSP FPGAs

The amount and type of security used to protect FPGAs is related to cost. First, it is important to realize that there is no such thing as unbreakable security. Ultimately, there is nothing you can do to completely stop a determined attacker from breaking a system. If someone wants your data or design, they can use brute force to get what they want. This is not the casual hacker, but possibly a well-funded government or a well-funded competitor, so with that in mind you will not be creating a solution that can never be broken but rather one that adequately protects you from the threats commonly encountered from cloners, overbuilders, tamperers, and reverse engineers. When you think about security, you need to consider what is appropriate for your needs. If your product cost is $10, there is a certain amount of security that you can afford for a system in this price range versus a system that might cost $10,000. This is an evaluation that you will need to do. Once you have gone through that evaluation, you can determine which set of products and which pieces of the security you might wish to implement based on that evaluation. There are a variety of solutions available from Xilinx that you can explore to solve your security issues. These solutions range from very simple to more complex. Solutions that are considered to be more basic for security implementation within the Spartan-3 Generation are addressed in the "Security Solutions Using Spartan-3 Generation FPGAs" white paper (WP266).

This white paper explores more advanced techniques, such as:

- Bitstream Generation Security Level
- Active Defense (JTAG boundary scan)
- Bitstream Validation (Cyclic Redundancy Checking (CRC))
- Advanced Data Manipulation

Beyond the Spartan product, Xilinx offers an even more advanced solution with our Virtex™ FPGA products.

Bitstream Generation Security Level

During the test and debug phase of a design, you can decide to leave the Internal Configuration Access Port (ICAP) or the ChipScope™ Pro Analyzer Cores in the design for possible maintenance or for random check-ups after the design goes into production. Some of the software utilities, such as the ChipScope Pro Analyzer, require these macros for reading the state of internal logic. While this is handy for the designer, it can leave a potential security hole.

The Bitstream Generator creates the configuration .bit file based on the contents of a physical implementation file called the NCD file. The .bit file defines the behavior of the programmed FPGA. The Bitstream Generator includes many options, some of which are not commonly used. One of these options is the Security Level settings. The Bitstream Generator has four settings; the first one is the default, and the remaining three are optional and provide additional security. As shown in Table 1, Readback can be optionally disabled completely or disabled except for internal access from the FPGA application via the ICAP.

Table 1: **Bitstream Generator Security Level Settings**

Security Level	Description
None	Default. Unrestricted access to all configuration and Readback functions.
Level1	Disable all Readback functions from both the configuration or JTAG ports (external pins). Readback via the ICAP allowed.
Level2	Disable all Readback operations on all ports.
Level3	Disable all configuration and Readback functions from all configuration and JTAG ports. The only command (in terms of Readback and configuration) that can be issued and executed in Level3 is reboot. This erases the configuration of the device. This has the same function as enabling the PROG_B pin on the device, except it is done from within the device.

For a detailed explanation of all the Bitstream Generator options; refer to the *Spartan-3 Generation Configuration User Guide* (UG332). Using the above security settings, Level 1, 2, and 3, will inhibit any of the following solutions that require the ICAP primitive.

Active Defense (JTAG)

A common concern is that any device with a JTAG interface is vulnerable to reverse engineering. JTAG can also be used to reverse engineer a system, device, IP, or a standard product by using the boundary scan chain. These actions require a well-funded, knowledgeable, and skilled attacker who has the equipment and the time. This organization, competitor, or government is trying to learn how a product works and will most likely try to cost-reduce it or add features. This section discusses methods for incorporating features into your designs to detect and prevent JTAG reverse engineering.

JTAG boundary scan was initially designed to help test and debug I/O connectivity on a PCB and was later adopted to include the logic inside of a chip. By using the INTEST command with boundary scan, you can shift data into a block or IC and then clock the IC to read back the resultant data. This operation can provide a skilled user with the architecture or logic in an IC or a block. As shown in Figure 1, this is also one way of

reverse engineering a design or a system. For this reason, unauthorized use of the JTAG port is a concern for some users and the security of their products.

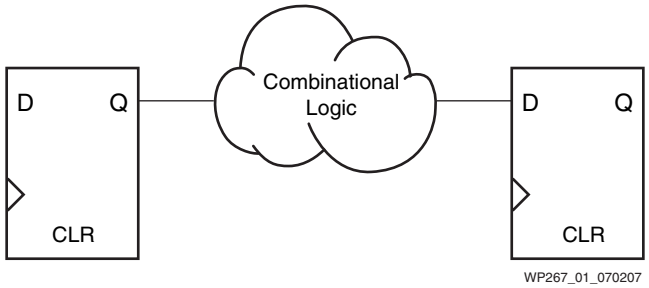

Figure 1: **Standard Boundary Scan Chain**

Spartan-3A/3AN/3A DSP devices are JTAG compliant, which allows configuration and readback of the FPGA. JTAG compliancy also means that JTAG pins cannot be inhibited. However, by using the Boundary Scan Block, a designer can design security to detect and inhibit unauthorized use of the JTAG port.

Boundary Scan Block

The BSCAN_SPARTAN3A macro block (see Figure 2) gives designers access to the boundary-scan signals. By simple instantiation of this block, the designer can monitor the activity on the JTAG pins from inside the FPGA.

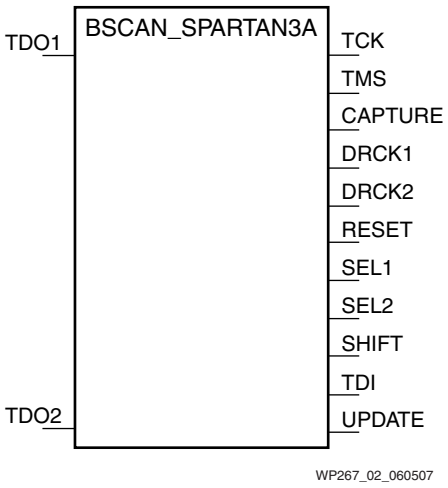

Figure 2: **BSCAN_SPARTAN3A**

```
-- BSCAN_SPARTAN3A: Boundary Scan primitive for connecting internal
-- logic to JTAG interface
-- Spartan-3A
-- Xilinx HDL Libraries Guide, version 9.1i
BSCAN_SPARTAN3A_inst : BSCAN_SPARTAN3A
port map (
TCK => TCK,
TMS => TMS,
CAPTURE => CAPTURE,      -- CAPTURE output from TAP controller
DRCK1 => DRCK1,          -- Data register output for USER1 functions
DRCK2 => DRCK2,          -- Data register output for USER2 functions
RESET => RESET,          -- Reset output from TAP controller
SEL1 => SEL1,            -- USER1 active output
SEL2 => SEL2,            -- USER2 active output
SHIFT => SHIFT,          -- SHIFT output from TAP controller
TDI => TDI,              -- TDI output from TAP controller
UPDATE => UPDATE,        -- UPDATE output from TAP controller
TDO1 => TDO1,            -- Data input for USER1 function
TDO2 => TDO2,            -- Data input for USER2 function
);
-- End of BSCAN_SPARTAN3A_inst instantiation
```

How Does the Boundary Scan Block Increase Security?

As stated previously, the block enables the JTAG port to be monitored internally for activity. If activity is detected on the port, you can design the logic to completely erase the FPGA configuration or bypass/inhibit selected functions. The ICAP can be used to erase the configuration of the Spartan-3A/3AN/3A DSP device. For a detailed explanation about ICAP, refer to the *Spartan-3 Generation Configuration User Guide* (UG332).

Figure 3 shows an example of bypassing key logic and functionality. The design incorporates a bypass MUX into the key input functions that is controlled by the output of the Detection Logic. During normal operation the signals go into the logic but when JTAG activities are detected the signals bypass are disconnected and a set value is placed through the logic. This makes the INTEST output completely useless for reverse engineering the internal logic.

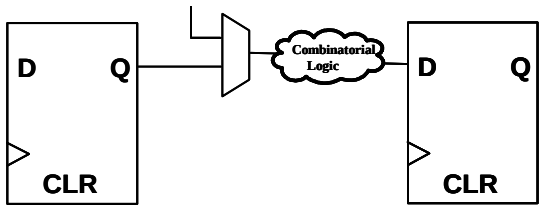

Figure 3: **User Defined Boundary Scan Chain**

In Figure 4, the "Detection Logic" can be as simple as a gate or your application might require more sophisticated logic.

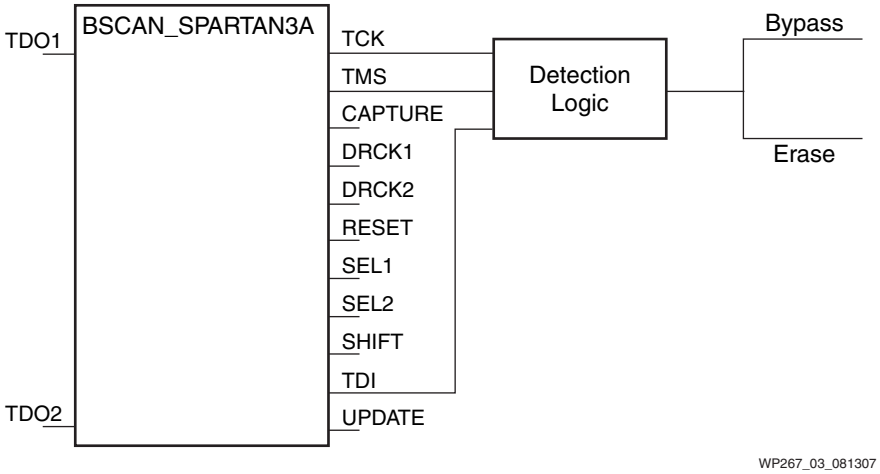

Figure 4: **Detection Logic**

JTAG Field Updates and Diagnostics With Security

In most cases, once a system or device is deployed and in operation, the JTAG interface is not accessed or used. However, there is always an exception to the rule; for example, when a system needs field updates or diagnostics, the JTAG port is required. If detection security is implemented to protect the device from unauthorized access through the JTAG port, this can inhibit authorized access. Several possibilities are available for the design to implement. The first is to design the Detection Logic so it only activates on INTEST test instructions leaving the JTAG to operate normally in all other modes such as BYPASS, IDCODE, USERCODE, and EXTEST. This makes it simple for field access to the JTAG port for upgrades and diagnostics.

For more complex security the Detection Logic could be designed to watch for a specific access routine or code sequence to allow access to the JTAG normal operation modes. This is useful when the field teams need access to the INTEST instruction for internal test and verification of the system functionality. This can unlock the JTAG instructions until the diagnostic testing and upgrades are complete. Once the upgrades are complete the rebooting of the upgraded FPGA can reset the Detection Logic. For systems only undergoing diagnostics, a code sequence can be issued which will restart the Detection Logic monitoring.

Both methods allow necessary field service tasks to be performed without compromising security via the JTAG Port. If the sequence detected is incorrect, then the ICAP can be used to reset (erase).

Active Defense Logic Resource Requirements

The Spartan-3 generation has many embedded features and functions built into the silicon. The JTAG state machine and interface logic to the ICAP are among the embedded functions. The BSCAN_SPARTAN3A block does not require logic resources since this function is embedded. However, user logic connected to the instantiated JTAG block does consume logic and interconnect resources. This logic can be as little as one logic cell or 10's of logic cells depending on the complexity of the user logic/function.

Active Defense Conclusion

With a little extra logic, you can detect and increase the security against reverse engineering by instantiating the Boundary Scan Block and simple Detection Logic when designing with a Spartan-3A/3AN/3A DSP FPGA.

Bitstream Validation

This section is focused on how to deter tampering of the configuration bitstream. A person who is interested in tampering a design may try to modify the original design to gain access to unauthorized services, steal sensitive data, or sabotage an application. By validating the device configuration during normal operation, an altered configuration can be detected and the design can decide how to handle the tampering. There are many ways to implement a validation circuit. One simple example is illustrated in Figure 5 using ICAP and CRC.

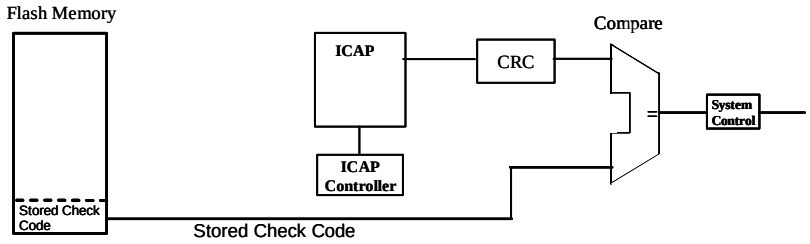

Figure 5: **Bitstream Validation**

ICAP Block

The ICAP block enables interface between the fabric and the FPGA configuration controller. This block primitive is like the Boundary Scan Block primitive in that its instantiation does not require extra logic cells because the ports are embedded in the FPGA. To read the configuration bitstream after the device is configured, the ICAP macro must be instantiated. The ICAP block is also commonly used for MultiBoot capability in the Spartan-3A/3AN/3A DSP platforms. If the ICAP is being used for more than one function, such as MultiBoot and bitstream validation then signal priority and control will need to be taken into consideration when connecting to ICAP. This can be as simple as a multiplexer or more complex arbitration logic.

Figure 6 shows the schematic symbol of the ICAP primitive followed by the VHDL instantiation template.

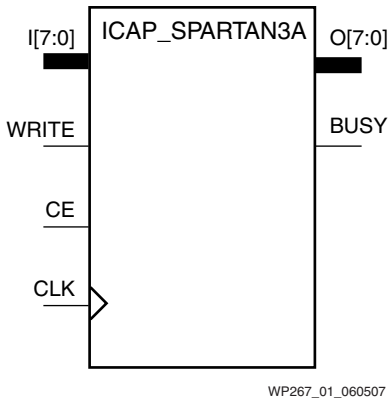

Figure 6: **ICAP_SPARTAN3A**

```
-- ICAP_SPARTAN3A: Internal Configuration Access Port
--                 Spartan-3A
-- Xilinx HDL Libraries Guide, Version 9.1.3i

ICAP_SPARTAN3A_inst : ICAP_SPARTAN3A
port map (
   BUSY => BUSY,      -- Busy output
   O => O,            -- 8-bit data output
   CE => CE,          -- Clock enable input
   CLK => CLK,        -- Clock input
   I => I,            -- 8-bit data input
   WRITE => WRITE     -- Write input
);

-- End of ICAP_SPARTAN3A_inst instantiation
```

Cyclic Redundancy Checking (CRC)

CRC is a type of check sum that is used to detect errors most commonly in data transmission and reception. It is incorporated in Bluetooth, Ethernet, USB, and satellite communication, as well as in the configuration of the FPGA. Xilinx FPGAs have a self check capability to verify the bitstream as the device loads the configuration. The CRC is calculated and compared to the Stored Value in the generated bitstream; if the two values are equal, the “Done” pin goes high indicating a successful configuration.

CRC algorithms are simple yet a highly effective way to check the integrity of the data. Hashing algorithms can also be used to validate the FPGA configuration. The choice of CRC or hashing algorithms is completely up to the designer.

Simple Bitstream Validation

The ICAP block is used to read the device configuration, which is then sent to a CRC that generates an active resulting value. The Active Value is then compared to the stored CRC Stored Value. In this example, the Stored Value is in an empty configuration memory location. If the two values are equal, then the configuration is correct. If the values are different then the device has been tampered with and the designer can determine the action. Some common actions that can be taken are as follows:

- Reload Configuration

  By using the ICAP block the FPGA can be erased and reconfigured. If the main configuration has been tampered with this will result in the FPGA continuously reconfiguring

- No Functionality

  The design completely stops functioning. This can be easily implemented in a Spartan FPGA by using global control signals like 3-state, Gated Clocks, Flip-flop clock enable and so on.

Additional action can be taken based on the design need.

Logic Resource Requirements

Using the embedded ICAP block uses no logic resource in the FPGA. There are a variety of CRC and hashing algorithms to choose from which are as simple as a few logic cells to hundreds of logic cells for the more complex algorithms.

Bitstream Validation Conclusion

Protecting data and access is more important to some designs than the design functionality. Simple bitstream validation can aid in the protection of the data, access, and design functionality being attacked by tamperers.

Advanced Data Manipulation

The Device DNA and the Stored Check Code are not a secret to the outside world; anyone can access this information. For more information on Device DNA security, see the “Security Solutions Using Spartan-3 Generation FPGAs” white paper (WP266). The real secret of the Device DNA design level security is the “security algorithm.” For some designs the security requirements need more than the default 57-bit Device DNA to increase protection from brute force attacks. The Device DNA was designed with the ability to add additional bits for increased security. The more Device DNA bits used, the longer it takes to complete a brute force attack. A brute force attack is when a cloner or overbuilder attempts to discover your security algorithm with the goal of generating the Stored Check Code. At some point, it becomes outrageously long, somewhat impossible, or not worthwhile to attempt a brute force attack. The total time for a brute force attack is a combination of the number of bits in the Device DNA and the Stored Check Code.

Additional security to aid against brute force attacks by using data manipulation of the Device DNA is shown in Figure 7. In this example, the design is constructed to add 64,000 bits to the Device DNA, which is stored in the Spartan-3AN user flash memory. This can just as easily be stored in the configuration memory or system memory. After the Device DNA, a sorter is inserted into the design. The sorter is simply a de-multiplexer, and a counter that is decoded to control the de-multiplexer's select lines. The first output of the de-multiplexer is sending data to the security algorithm and the second output dumps bits into the proverbial bit bucket. This simple circuit has now altered the Device DNA relationship to the Stored Check Code, making a brute force attack or reverse engineering of the security algorithm even more difficult.

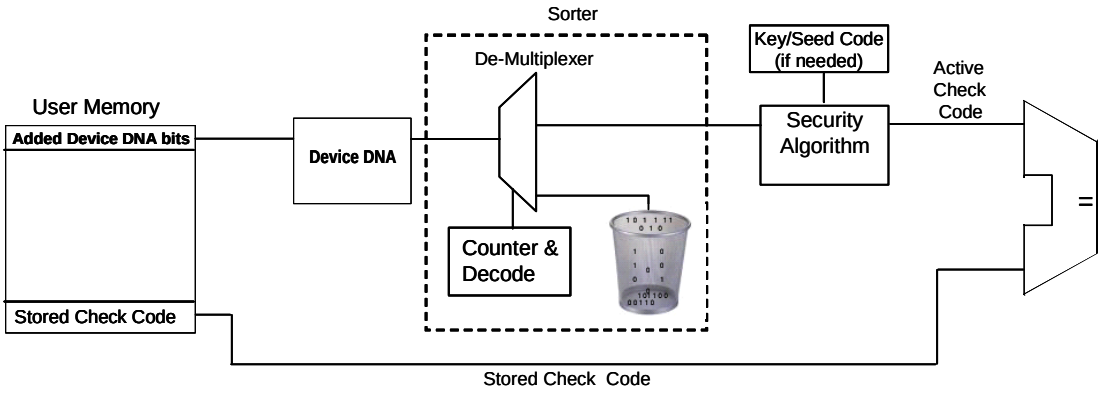

Figure 7: **Data Manipulation of Device DNA**

Advanced Data Manipulation on the Stored Check Code and Algorithm Control

Further expansion of the data manipulation technique can be used to incorporate the Stored Check Code. Figure 8 shows an example in which the data manipulation sorter has been expanded to combine the additional Device DNA bits with the Stored Check Code. Now, the cloner or overbuilder only sees the Device DNA being read into the FPGA. This makes it very difficult for a cloner or overbuilder to first reverse engineer the Device DNA, the Stored Check Code and the trash bits, and then continue to reverse engineer the security algorithm. In this example, a third de-multiplexer output has been added to separate out the Stored Check Code and send it to the comparator.

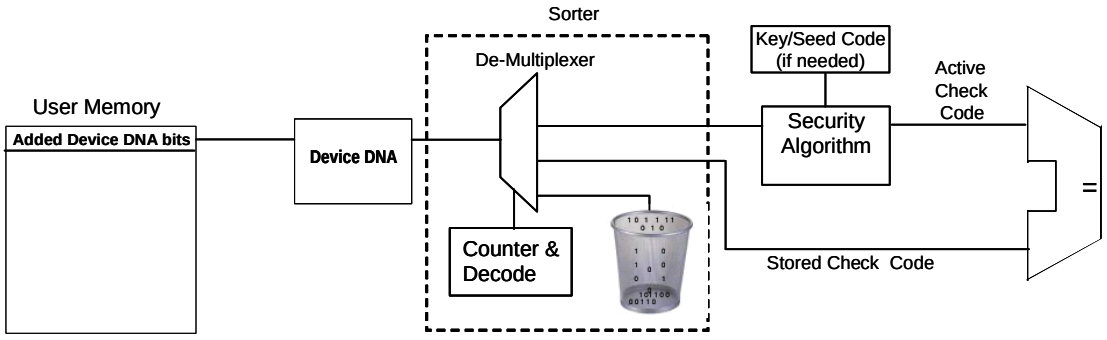

Figure 8: **Data Manipulation on Stored Check Code**

This data manipulation can be taken even further by adding a fourth output to the De-multiplexer and connecting it directly to the security algorithm as seen in Figure 9. Based on the security algorithm selected, this can allow the designer to alter his seed values, security key, or even the algorithm itself resulting in another layer of security to prevent cloning or overbuilding. Also, with this data manipulation the hardware design in the FPGA remains 100% the same, but the security algorithm is altered. This altering of the added security algorithm allows for easy upgrade of the design security in the manufacturing flow or even in the field.

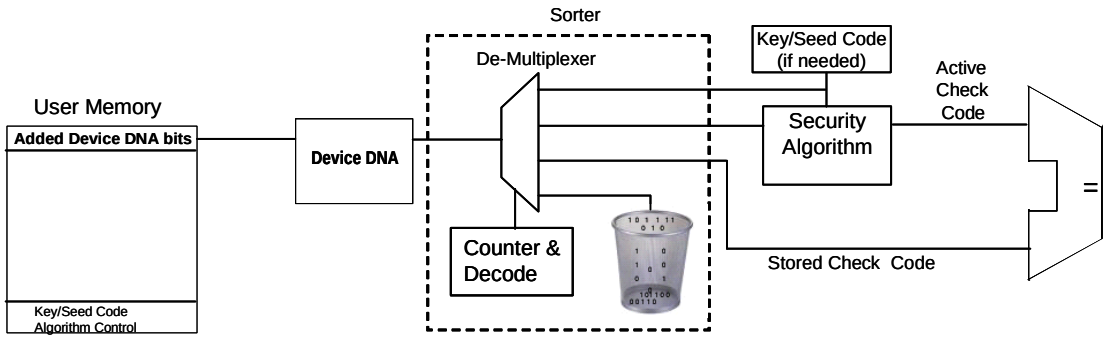

Figure 9: **Adding Fourth Output to De-Multiplexer**

Logic Resource Requirements

The data manipulation sorter is a de-multiplexer, and a counter that is decoded to control the de-multiplexer's select lines that can be implemented in as few as 10's of logic cells.

Advanced Data Manipulation Conclusion

Advanced data manipulation helps protect FPGA designs from cloners and overbuilders attempting brute force attacks while also providing a simple and quick way to upgrade the security.

Conclusion

This white paper presents multiple advanced security schemes that can be incorporated by a designer to help defend against cloning, unauthorized overbuilding, reverse engineering, and tampering of a design or system. Part of the advanced methods described are also layering techniques. This technique incorporates various schemes to reduce multiple vulnerabilities simultaneously.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
08/15/07	1.0	Initial Xilinx release.