Overview of Microsemi Antifuse Device Security

White Paper

November 2013

Introduction

When third parties discuss the design security of Microsemi's antifuse FPGAs, the focus is frequently on the security fuse. It is often incorrectly identified as the primary protection that prohibits device cloning, tampering or reverse engineering. Unlike other PLDs with security fuses, Microsemi's security fuse is only a small part of the security built into every Antifuse FPGA. But what happens if, in the unlikely event, the security fuse is bypassed or is left unprogrammed? What else must a hacker do to extract design information from a Microsemi antifuse device?

An antifuse-based FPGA is the most secure programmable device available. This paper explores various aspects of the Microsemi antifuse security structures, as well as the inherent security of the technology and underlying architecture. The method of programming an antifuse device and the role of the security fuse are also discussed. Various means of attack or "hacking" a secure antifuse FPGA are also described and shown to be ineffective. We will show that the antifuse security system is much more sophisticated than just a simple security fuse, and even if left unprogrammed, Microsemi's antifuse FPGAs are still the most secure programmable technology available today.

Understanding Antifuse Programming

When users think of FPGA programming, they often think in terms of a bitstream being downloaded to and stored in the FPGA. An SRAM-based FPGA is a volatile device that must be initialized or programmed at each power-up cycle. This is not the case for antifuse-based FPGAs which are one-time programmable and non-volatile devices. An antifuse does not use a bitstream. This means that there is never a bitstream that can be intercepted, copied, modified, or corrupted.

Once a designer has completed place-and-route using Microsemi’s Designer software, the program generates an AFM (Microsemi fuse map) programming file. This file's format is unpublished, and is a carefully guarded secret within Microsemi, in contrast to competitors' bitstream formats which have been well documented and are freely discussed in the public domain. The AFM is an encrypted, binary file compressed using a proprietary algorithm. What can be disclosed is that this file contains the addresses of fuses to be programmed, and that the data is stored in records (sometimes of variable length), containing programming information for multiple fuses. No design information is contained within the file, in contrast to the ADB (Microsemi database) file, which does contain design information; but the ADB is not used for programming. Each AFM also includes device and design-specific end-of-programming tests to insure correct device programming.

The AFM file is downloaded to the Sculptor programming station via the PC interface. This file is essentially a script file telling the programmer which fuses to program. The programmer communicates with the device programming interface, issuing the commands necessary to program and verify each antifuse needed for the design. The device-level programming interface only understands how to build a connection based upon the addresses communicated by the programmer. These addresses are not stored within the device. No other programming data is downloaded or stored within the device. All "intelligence" for programming is contained in the programmer, not in the device, so no programming file can be downloaded or read back from the device. This fact alone makes antifuse FPGAs immune to device cloning.

The AFM File: A Secure Medium

The secure nature of Microsemi's AFM file makes it the ideal method for communicating a secure design to a manufacturer for two reasons: the design's netlist cannot be compromised if intercepted by a third party, and the manufacturer has the only means of programming devices so the design data is never resident with the manufacturer. Thus, the design data is never on the outside.

This is in contrast to communicating a design database to a secure ASIC manufacturer. To produce an ASIC, the ASIC manufacturer needs either the source netlist or GDSII layout data (which can be converted within hours to a netlist). This second scenario leaves design data on the outside and vulnerable, both to interception during transmission and to theft once at the manufacturer's site.

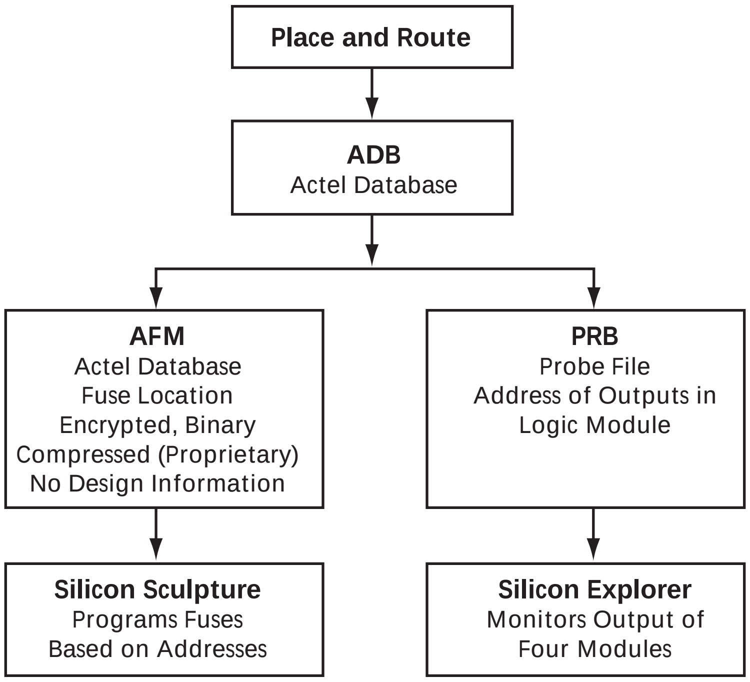

Figure 1: Programming Files

The Role of the Security Fuse

So what is the role of the security fuse if Microsemi's antifuse FPGAs are immune to cloning? The security fuse, once programmed, disables the probe and programming interface, serving merely as additional insurance to thwart hackers—a belt-and-suspenders approach. Even if Microsemi’s antifuse FPGAs did not have a security fuse, they would still be the most secure programmable devices available.

Possible Means of Attack

Next we will explore possible means of attack that a hacker might contemplate in an attempt to recover design data from a Microsemi antifuse FPGA.

Bypassing the Security Fuse

A hacker’s first thought might be to try and bypass the security fuse to gain access to the device. The first problem is that the hacker has to identify which antifuse out of several million is the correct one. How to identify which is the correct fuse? One approach suggested by competitors is the use of thermal imaging during security fuse programming. Antifuse connections are passive low-impedance elements, and although they require high voltage to program, the current and duration of the pulse to build the antifuse connection is very small. The total energy needed is on the order of 40µ joules (an amount of energy sufficient only to raise the temperature of one gram of silicon 0.000056ºC). Any heat generated from a fuse being programmed would be swamped by other heat from

Cloning

Cloning is different from reverse engineering. With cloning, a competitor or hacker simply copies the device contents—he does not know how your design works, nor does he need to.

Cloning requires that something be easily copied; this is the situation with SRAM-based FPGA designs. A competitor either makes a copy of the boot prom, or intercepts the bitstream from the on-board processor and copies the code. He is able to steal the entire design by copying the external bitstream, which is always required for an SRAM FPGA. He can then manufacture exact clones of your product.

CMOS circuitry in the area, as well as the thermal mass of surrounding metal and silicon. Conclusion: simple thermal imaging cannot be used to detect the location of any antifuse, including the security bit.

What if a highly capable hacker has somehow been able to locate and bypass the security fuse? He has now gained access to the programming and probe interface, but since no programming data is stored in the device, there is no bitstream to download. The programming interface has limited capability. For example, this interface cannot be queried for the location of programmed fuses, so there is no way to have the programming I/F reconstruct the AFM file, or even a fuse map. So, as stated before, it is impossible to clone an antifuse device—bypassing the security fuse only gains more access to internal nodes within the device; it does not allow programming additional devices or reverse engineering an existing device.

Next we will look at what might be done with this additional access.

Exploiting the Probe Interface

Bypassing the security fuse (or leaving it unprogrammed) will allow a hacker access to the antifuse probe circuitry. In normal usage, the probe interface allows the designer to have internal access to his design during operation, providing virtual logic probes that operate in real time and do not affect either the performance or loading of the design. To properly probe the design requires a valid probe file (PRB) generated from the same database (ADB) as the AFM programming file.

The probe interface accepts output addresses for logic modules to be probed (basically building a connection to a dedicated output pin). So how could a hacker try an exploit this interface? First, he would need to reverse engineer the probe file structure, then reverse engineer how the addressing scheme works. Then he would have to determine the address for each logic module output in the device. For example, for the AX2000, a hacker has to determine addresses for 32,256 logic module outputs out of more than 50-million antifuses in the device—technically feasible, but a highly daunting task.

Again, for argument's sake, let us assume that a highly capable hacker has spent countless hours, destroyed multiple devices, and has somehow determined the probe interface's addressing scheme. How can he exploit this knowledge to reverse engineer a device?

First, we will assume that the hacker has the board and system from which the FPGA comes, and has the equipment and time to understand how each of the I/O pins of the FPGA are configured and what the signals going into the FPGA look like. We will also assume that he has countless hours of access to a sophisticated (and expensive) component tester. We will also assume that he can generate a stimulus file using data he has gained from the actual system. Where is he now?

He can now see anywhere from two to four outputs at the same time. Using the AX1000 as an example, he can monitor the output of four modules until that output changes state on one of the modules, and note the patterns leading up to that change. Then, using the other three probes, he can check each of the remaining 18,143 to see which clock-cycle they are on and if they change state, to determine the connectivity. Of course since the output of one module can drive multiple outputs, he would have to rerun the same set of patterns 18,143 times for each module. And since each combinatorial module can have as many as five inputs, it would be unwise to try and shortcut the process by eliminating the earlier modules examined. So the hacker would need to run 18,1432 tests to determine module connectivity. Assuming that he runs a minimum of five patterns to "flush out" a connection, he would need to run more than 1.6 billion patterns just to determine module interconnectivity. Of course he would also have to develop a methodology to weed out false paths (did module 1,046 change state because it is driven by module 864, or is there another path that is exercised by the same pattern?).

Once again, let us assume that our highly capable hacker has been able to work out a correct connectivity map. Now he needs to determine from the input patterns the correct configuration for each logic module. However, this is even more challenging than the connectivity problem, as each register module (R-Cell) has to be identified for the clock domain to which it belongs, whether the output change is a result of clocking, setting or clearing, or if the clock polarity is negative (clock polarity can be configured at each individual R-Cell, and may need to double his pattern set to look at both rising and falling edges—another 3-billion patterns). The combinatorial module (C-Cell) is even trickier as it can implement up to 4,000 possible functions with up to five inputs. Looking at only module outputs via the probe interface to construct the entire circuit is a nearly impossible task, so even a highly capable hacker needs to find another way.

Exploiting the Programming Interface

So after having failed or finally given up trying to exploit the probe interface, our hacker now turns to attack the programming interface. As discussed earlier, the programmer addresses each fuse to program and verify each needed for the design. A hacker could attempt to use this verification process to determine the programmed state of each fuse. He would need his fuse-addressing scheme reverse engineered during his probing attack, but now he needs to generate a special (and legal) AFM file to address and verify every fuse. As discussed earlier, the AFM file is a proprietary encrypted and compressed binary file, so our hacker would have to reverse engineer the entire programming process without the help of Microsemi—an impossible task.

Our hacker could try to bypass this step, thinking that all he needs to do is generate a design that utilizes 100% of the logic resources and 100% of all module inputs in the device, then generate an AFM that would address and verify most, if not every, fuse. However, even 100% utilized designs require less than 1.5% of the total of all device fuses be programmed (the vast majority of fuses are used as vias for routing). Our hacker is back to square one.

We will assume that through a combination of genius and persistence, our highly capable hacker has overcome these impossible odds and has developed a special AFM file, and can verify each fuse to determine its programmed state. However, our hacker now hits another wall. During device programming, antifuses are programmed in a specific order, and from the inside out. This is done to minimize the amount of addressing circuitry that would be needed to address every fuse independently. So when our hacker tries to verify a fuse, he may not be able to address the correct one, or he may end up with a false positive (the fuse he thinks he is addressing may also address additional fuses in sympathy). Additionally, there are other buried security structures inside a Microsemi antifuse device to designed to hinder illicit access. As a result, any hacker will end up with a junk fuse map that will be of no value.

Having exhausted both the probe and programming interface without success, our hacker now tries another method to get at the design.

Decompiling the AFM File

Our hacker has given up attacking the device directly in his attempts to steal the design. Now he resorts to more traditional and clandestine methods. Since the AFM file is used to program devices, it is more widely distributed than the design files, which are kept in a more secure environment. Our hacker finds a corruptible employee or an unscrupulous contract manufacturer and obtains a copy of the AFM file. But, as discussed earlier, the AFM file contains no design information, and there are no programs available either inside or outside Microsemi that can decrypt, decompile and regenerate the existing design info. Even if our hacker had "inside" knowledge, he could still not decompile the AFM. All he can do is program additional copies of the original device.

Direct Inspection

So our hacker is left with one last resort, an invasive attack—direct inspection of each fuse to generate a fuse map. The antifuses are located between two layers of metal, so inspection from above won't work. He needs to slice along each metal track in the device. Since the feature size difference between a programmed and unprogrammed antifuse is only several angstroms wide, our hacker will need a scanning electron microscope (SEM) to determine the state of the antifuse (they are not visible with the naked eye). Given the difficulty in sectioning a device without damaging the antifuse location, he would need to slice many, many programmed devices and inspect each slice to find which 1.5% of the fuses out of millions is programmed—a very expensive, time-consuming and error-prone process. Once our very determined hacker has finally created a fuse map, he then would need to reverse engineer the FPGA architecture to know which fuses are used for interconnect, which are for logic module configuration, which are for clock distribution, etc. Only then would he be able to reverse engineer a device, or produce modified copies of the original design.

Summary

- Microsemi antifuse devices cannot be cloned whether or not the security fuse has been programmed. There is no recoverable bitstream or similar mechanism contained in an antifuse part.

- Specific architectural features and aspects of the device hinder or make impossible any attempts to reverse engineer the device.
- Design information is contained in the ADB file, which is not used for programming. The ADB is normally kept in a secure place and is not distributed to third parties for programming.
- The AFM file does not contain any device information, and cannot be reverse engineered to extract design information. The AFM only specifies what fuses should be blown, and in which order.
- The antifuse technology itself provides a high level of security and design protection. Determining the difference between a programmed and unprogrammed fuse requires a scanning electron microscope, and the device must be physically destroyed in the process.
- Attempts to identify fuse locations via thermal imaging of fuses being programmed is ineffective. Insufficient thermal energy is generated during programming, and the thermal signature is swamped by the power generated by the CMOS circuits within the chip itself.
- Probes look at module outputs only. To use them effectively, you need access to the ADB file to generate a valid probe map. Without this information, you are trying to reconstruct an entire design by looking at the outputs of modules without knowing the input stimuli.

It is clear that even the most determined hacker (even a highly capable one) is faced with several significant hurdles, many of which are virtually impossible to overcome when attempting to reverse-engineer a Microsemi antifuse device. A serious attempt would require significant investment in dollars and time—and would most likely end in failure. From a practical perspective, Microsemi antifuse devices are virtually impossible to reverse engineer, and are the most secure silicon devices available.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA92656USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

55800003-0/11.13