FPGA Viruses

Ilija Hadžić, Sanjay Udani and Jonathan M. Smith
{ihadzic, udani, jms}@dsl.cis.upenn.edu

Distributed Systems Laboratory, University of Pennsylvania *

Abstract. Programmable logic is widely used, for applications ranging from field-upgradable subsystems to advanced uses such as reconfigurable computing platforms which are modifiable at run-time. Users can thus implement algorithms which are largely executed by a general-purpose CPU, but may be selectively accelerated with special purpose hardware. In this paper, we show that programmable logic devices unfortunately open another avenue for malicious users to implement the hardware analogue of a computer virus.
We begin the paper with an outline of the general properties of FPGAs that create risks. We then explain how to exploit these risks, and demonstrate through directed experiments that they are exploitable even in the absence of detailed layout information. We prove our point by demonstrating the first known FPGA virus and its effect on the current absorbed by the device, namely that the device is destroyed. We close by outlining possible methods of defense and point out the similarities and differences between FPGA and software viruses.

1 Introduction

SRAM-based programmable logic devices have been widely deployed wherever hardware performance and software flexibility are required concurrently. One of the most ambitious uses of the devices has been in the field of Run-time Reconfigurable Computing [16] where selected portions (or even the entirety) of algorithms are implemented in hardware, offering high performance while maintaining the flexibility of software systems. Various research and commercial platforms that utilize programmable logic as an accelerator or processing engine have been proposed[4, 5, 12, 15].

The majority of research on using the devices for computing has focused on the issues of mapping various well known algorithms to reconfigurable hardware[8], device technology[6], resource management[9], hardware-software co-design[11], and to some extent, programming models[10]. Very little attention, in fact none that we are aware of, has been paid to the security models for these devices. In fact, the connotation of *security* in the FPGA community has been framed

* This work was supported by DARPA under Contracts #DABT63-95-C-0073, #N66001-96-C-852 and #MDA972-95-1-0013, with additional support from the Hewlett-Packard and Intel Corporations.

in terms of protecting the intellectual property contained in a device's configuration, rather than the security and integrity of the system itself. Furthermor, PLD vendors assume that by keeping the architectural details and the format of the configuration data of their devices proprietary, the design contained in the configuration data can be secured[1].

In this paper we show that neither the system nor any associated intellectual property can be protected by practicing security through obscurity. We show by example that it is possible to deduce the architectural details necessary for constructing malicious configurations without knowledge of any proprietary information.

In the next section, we analyze the properties of FPGA devices and show how they can be exploited to create specific forms of attack. We also provide the definition of classes of attack that can be performed in run-time reconfigurable systems. In Section 3 we present an experiment in which an extremely destructive form of FPGA virus attacks a device at the transistor level and attempts to destroy it (in several experiments in our laboratory, the attempt succeeded). In Section 4, we outline the potential spreading mechanisms for FPGA viruses. In Section 5 we discuss possible methods for preventing and detecting attacks. We assess the impact of our results in Section 6, which concludes the paper.

2 Opportunities for Attack

Reconfigurable hardware has the interesting property that it can both change a system's behavior at the logic level as well its electrical properties. In general, no other architectural component has this property.

For example, by executing different programs, a processor changes its logic behavior, but the electrical properties of the system remain unchanged. Similarly, memory can be viewed as a lookup table for which the logic behavior is programmed by changing its content. On the other hand, reprogramming an FPGA device can change its electrical properties (*e.g*, power consumption, pin types, slew rate of output signals, etc.). A malicious user can exploit this property to cause damage (effecting a security attack) at the electrical signal level. This creates an entirely new class of destructive behavior than the attacks used by software computer viruses. Most interestingly, these attacks are centered on the physical destruction of the system (*e.g.* by overheating).

To classify the wide range of potential attacks to a system, we have created three categories based on the type of threat:

- *Level 0* (Electrical Signals): At the lowest level, the attacker creates electrical conflicts either inside the device or at pins connecting the attacked device to other components of the system. The goal of this attack type is to physically destroy system components. We call this class of threat a Malicious Electrical Level Threat (*MELT*).
- *Level 1* (Logic Signals): At this level, the attacker generates signals which are electrically correct, but logically do not make sense to other devices.

For example, an FPGA device attached to a processor bus can generate a sequence of signals which do not represent any meaningful bus cycle causing unpredictable behavior of the system. We call this class of threat a Signal Alteration Logic Threat (*SALT*).
- *Level 2* (Software Attacks): Finally, a virus may generate legitimate cycles which together compose an execution of a malicious task (*e.g.*, deleting data from the disk). This attack level is equivalent to the attacks performed by software viruses. We call this class of threat a Higher Abstraction Level Threat (*HALT*).

MELT represents an interesting, and most destructive, form of attack enabled by the addition of reconfigurable hardware. SALT attacks may cause unpredictable behavior in the system, but cannot directly cause physical damage (although they may indirectly cause damage, such as forcing a disk device or FLASH to operate until failure). It is less destructive than MELT, but its detection and prevention can be very difficult, mainly because any such prevention requires a rather complete model of the system in which the device is embedded. We are, however, thinking about how to address SALT attacks. Finally, HALT attacks should be treated the same as malicious software code (*i.e.* software virus) and are thus not FPGA-specific. They are harder to detect since the device has no model for the valid behavior of systems scaffolded on top of it. In such instances, a defense should be based on establishing a trust relationship between the source of the configuration and the user executing it (*i.e.*, when configuring the FPGA device). The problem is thus a classic security problem and therefore neither novel nor of particular interest to FPGA users.

The attacker's goal at the electrical level is to physically damage the system. To destroy a system component, the attacker must create high currents either inside the device or at its input/output pins. The latter case can be easily realized provided that the attacker is familiar with the board level architecture of the system. It is necessary to know which pins of the attacked device are supposed to be configured as inputs (*i.e.* connected to outputs of external devices) and configure them as outputs.

This will result in a potential conflict in logic levels creating a high current through the output transistors of both the device and whatever is connected to it, for example another device, as shown in Figure 1 (a).

Either device may be destroyed if a high current is applied sufficiently long. In addition to knowing the board level architecture, the attacker must have some insight on the behavior of external signals, as high currents exist only if the attacked device outputs the logic complement of signals applied to it. Since the compiler is not aware of the board level system architecture, such a malicious configuration represents a legitimate design from the device's perspective and a defense using a compiler technique does not appear possible. More appropriate methods of defense are discussed in Section 5.

The second group of electrical level attacks attempts to create high currents inside the device by programming it with a configuration that creates a logic conflict in the internal connections. Most FPGA devices use pass gates to connect

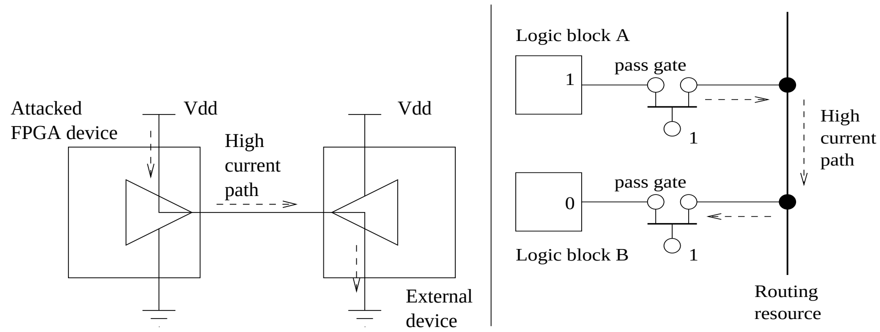

Fig. 1. Electrical conflict at a) I/O pins - left and b) logic elements - right

logic blocks to routing resources. This represents an opportunity for creating an internal conflict if two (or more) logic blocks are configured to drive the same routing resource as shown in Figure 1 (b).

In contrast with the previous example, such a configuration cannot be generated by a compiler since it will not allow internal logic conflicts. However, it is in fact possible to modify the compiler output file (*i.e.*, the device configuration data) and create internal logic conflicts. In the next section we demonstrate the construction of such a configuration, using the Altera EPF8636ALC84-4 device[3] as an experimental platform (any vendor's devices will exhibit the relevant properties), using no Altera proprietary information and rendering the device inoperable.

3 Constructing a Virus

In this section we describe an attack at the electrical level (MELT) that creates internal logic conflicts taking advantage of the fact that the interconnection between routing resources is achieved via pass-gates that connect multiple logic blocks to the same routing resource.

The first step in constructing the internal logic conflict is the identification of vulnerable points. That is, a connection of multiple logic blocks to the same routing resource via a pass gate. In Altera Flex8000 family, the logic elements (LE) from logic array blocks (LAB) in the same column share a column interconnect.

For example LE(1) in LAB(A1) in Figure 2 will share the column interconnect with LE(1) in LAB(B1). Therefore, if we could program the LE(1) in LAB(A1) and LAB(B1) to output complementary signals and connect both of them to the shared column interconnect, an internal conflict would be created. This conflict pattern can be replicated as many times as the device size allows and increases the device power consumption up to a level sufficiently high to overheat and destroy it. Column interconnects are the *only* vulnerable point in Altera's Flex devices, since each logic element has a dedicated row interconnect and internal conflicts among the logic elements in the same row are not possible.

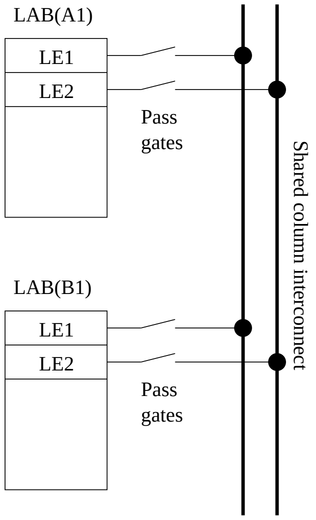

Fig. 2. Shared column interconnect in Altera Flex 8000/10K devices

To construct an internal conflict, we analyzed the configuration files of simple logic designs and compared the differences resulting from changing the logic element assignments and the logic functions. We identified locations in the configuration file which correspond to the logic element configuration and connections to column interconnect. To create a conflict we used two designs, one that utilizes a 4-input NOR gate in row A of the device and one that utilizes a 4-input OR gate in row B[2]. We created a configuration with internal conflicts by mixing the logic elements and column interconnects from two previously described configurations. Since the configuration files for these devices do not use a global checksum or CRC, this cut-and-paste type of attack can be easily realized. Even if such a global checksum or CRC existed, the device would not be secure since the attack would require only slightly more effort.

We have experimentally verified our claim by downloading configurations with internal conflicts into the device and measuring the supply current. No clock has been applied so the measured result represents the quiescent supply current which should be very low given that the device is fabricated using CMOS technology. The results of our measurements are shown in Figure 3. With only one conflict the quiescent current is greater than the maximum of $10mA$ specified by the datasheet[3] and it grows almost linearly with the number of conflicts. A small non-linearity appears due to the fact the the mobility of carriers in silicon drops with the temperature, causing some current flow to be reduced. Typically, the supply current will have an overshoot after the device is configured and the

[2] The logic function used is arbitrary as long as the functions in row A always output the complement of functions in row B an are sufficiently “complex” so as to prevent the compiler from placing the logic in I/O blocks

current will fall as the device heats up until it reaches steady state. Despite this negative temperature feedback, quiescent current can grow arbitrarily and the upper limit is determined only by device size – that is, the number of possible logic conflicts.

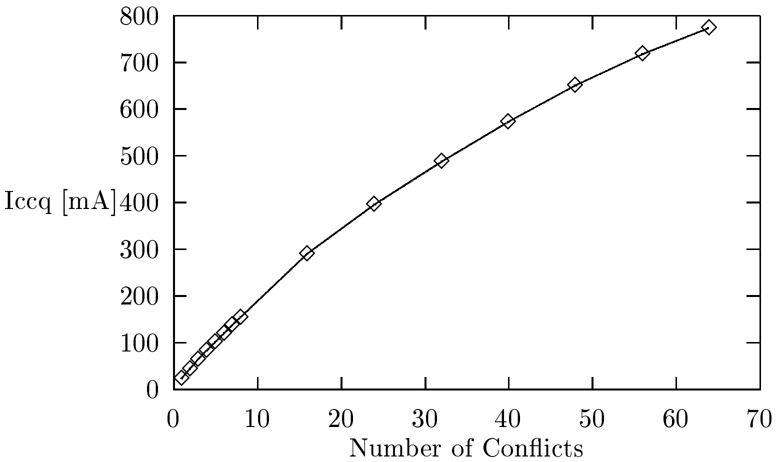

Fig. 3. Quiescent current as the function of number of logic conflicts

Although it is not possible to guarantee the physical destruction of a device, the attacker would normally try to make a device operate in the unsafe region and increase the probability of its desctuction. As an illustration we will determine the critical current at which the junction temperature exceeds the maximum of $T_J = 135^oC$ specified by the datasheet. We will show that this temperature is easily achieved with a relatively modest number of logic conflicts.

The thermal resistance of the PLCC84 package is $\theta_{JA} = 35^oC/W$[2] and assuming the ambient temperature of $T_A = 25^oC$, we can easily calculate the maximum power consumption:

$$P = \frac{T_J - T_A}{\theta_{JA}} = \frac{135 - 25}{35} = 3.143[W] \tag{1}$$

For the power supply voltage of $V_{dd} = 5V$, we calculate that the maximum allowed supply current is $I_{cc}^{max} = 629mA$. From the graph in the Figure 3, it is clear that this limit can be easily exceeded with the quiescent current (*i.e.*, without applying the clock) if the number of logic conflicts is greater than 50. During our experiments, we have observed extensive device heating. In several trials, this resulted in physical destruction of the device, manifested through the device's inability for further reconfiguration after being exposed to the high supply current.

In addition to being a threat to the attacked device, the high current drawn by a infected device/card in a system will potentially reduce the available supply current for other devices in the system. For systems designed to be physically small and densely packed, temperature may be a critical issue and increasing the ambient temperature could make the entire system operate in an unsafe region, leading to larger scale failures. Even if the system power requirements are met, this heat can lead to long term instability.

In the case of cards plugged into a PCI bus, the PCI Bus Specification (version 2.1)[13] says that the *total* power drawn by a PCI card cannot exceed $25W$. If a card with several FPGAs on it were infected with the virus (thus drawing high current), that specification would be violated. Depending on the design of the bus, the system may then crash or it may work intermittently or even work normally. This uncertainity is unacceptable for most systems. As programmable devices are used in more cards, this problem will grow.

4 Spreading a Virus

Replication mechanisms can be classified as either software assisted replication or pure hardware replication.

The first mechanism is equivalent to the replication mechanism of software viruses, with the only difference being in the system component the virus attacks. While most software viruses target the hard disk of a computer system, the FPGA virus would target the programmable logic devices in the system. If run-time reconfigurable computing platforms become more widely adopted, future computer systems will utilize a combination of a general purpose CPU and some amount of reconfigurable hardware. An FPGA virus would then be a piece of code which carries a malicious FPGA configuration and whose replication mechanism is implemented in software. The attack could be performed either by directly programming the FPGA device once it has been found in the system or by replacing the FPGA configurations associated with other programs in the system which utilize the run-time reconfigurable logic. A hardware-library model, such as that proposed in [14], is especially vulnerable to such attacks.

The second replication mechanism provides limited opportunity for the virus to spread without software assistance. An FPGA device could theoretically store a configuration of other smaller device in the user memory and use it to create a reduced version of itself in other devices given sufficiently large difference in device sizes. Although possible, this replication mechanism is difficult to realize in practice and is mentioned here only for completeness.

Logic conflicts inside the device can be generated either immediately upon downloading or some time after device configuration. Postponed logic conflicts can be easily generated by programming the short circuited logic blocks so that a control signal selects if they should output same logic levels (no conflict) or opposite logic levels (logic conflict). The virus with hardware replicating mechanism would typically use postponed logic conflicts to avoid device destruction before it gets the chance to spread to other devices.

Software replication mechanisms have the same properties as classical software viruses and should be studied as such. On the other hand, hardware replication mechanisms are limited to the local system and can be easily disabled if the architecture on the board level is such that FPGA devices do not have access to configuration signals of other devices.

5 Detecting and Preventing the Attack

The design space for an attacker in systems with run-time reconfigurable hardware contains the design space for the attacker in CPU based systems.

Our goal is to study the defense methods only for attacks specific to reconfigurable hardware and to provide a safe environment on the electrical and logic signal levels (*i.e.*, protect from MELT and SALT attack types) so that the attacker's design space is reduced back to its software subset. This approach would then allow us to treat the FPGA viruses the same way we treat software viruses, which have been studied in work by others[7].

We will now present the three main methods of defense against FPGA viruses, and discuss the advantages and disadvantages of each method.

5.1 Configuration File Verification

Before it is downloaded into a device, the configuration file can be analyzed for potential logic conflicts, both internal and external. For a logic conflict to exist either of the following must be true:

- two or more logic blocks are connected to the same routing resource
- an output pin is connected to the output of an external device

The goal of the configuration file analysis is to ensure that none of the above necessary conditions is satisfied. A correct place and route algorithm will ensure that the first condition is not satisfied. However, it is still possible to create a compiler that generates a malicious configuration or directly modify the configuration file as demonstrated in Section 3. The second condition cannot be addressed at compile time since the compiler is not aware of the board level architecture.

It is therefore necessary to analyze the configuration file against both conditions prior to downloading it into the device. Since the system is or can be made aware of the board level architecture, analysis against the second condition is also possible. For successful attack prevention, the system must know the format of the configuration file and search for the binary patterns that correspond to potential conflicts. Since the number of connections to a routing resource is finite, analyzing the configuration file is a viable solution.

The advantage of this method is that the attack can be prevented before the device is exposed to a malicious configuration. The major disadvantage is in the time necessary to perform the analysis, which adds to the download time. Users would also need to know the device configuration file format, something which device vendors are reluctant to make public.

5.2 High Current Detection

Electrical level attacks have the property that they must generate a high current either internally or at the I/O pins. In the case of internal logic conflicts the high current drawn causes a high current at the power supply lines. This high current will be present in quiescent mode and it will be greater than the maximum allowed quiescent current[3] for the device as specified by the vendor.

The attack can easily be detected by measuring the quiescent supply current after the device has been configured but before applying the clock. Power can then be immediately disconnected from the device if the maximum is exceeded. There are many vendors (*e.g.* [17]) who carry current sensors which can be used to provide a digital signal when the current on a circuit exceeds a limit. The response time of these devices is typically on the order of a few microseconds. With the appropriate circuitry, these sensors could be used to guard not only power pins but also I/O pins from high current damage. Once the attack is detected, the device can be immediately cleared to prevent damage.

This detection method has neither of the disadvantages of the analysis method - it does not require additional time to analyze the file, nor does it require knowledge of proprietary file formats. The key disadvantage is in the additional protection circuitry required and the existence of a short current pulse before the attack is detected. In addition, the current detection method could have problems detecting postponed attacks as it relies on measuring the quiescent supply current and (by definition) postponed attacks happen after the clock has been applied and the supply current consists of both quiescent and switching current, which could result in bogus attack detection.

5.3 Avoiding Pass Gates

The final way of preventing electrical level attacks would be to remove pass gates from the architecture of FPGAs. Although it would completely eliminate the opportunity for an electrical level attack on the internal logic, this approach is not feasible for economic reasons. Replacing pass gates with logic (*i.e.* multiplexers and demultiplexers) would dramatically reduce the device density as well as achievable clock rates. Also, removing pass gates would not prevent external (I/O) level attacks.

6 Conclusion

We have presented a threat model and outlined possible defense methods in systems that utilize run-time reconfigurable hardware. This includes line cards and add-in boards with accessible FPGAs. We have developed the conceptual basis for, and experimentally demonstrated, that the threat is realistic.

[3] We have experimentally verified this claim only for Altera Flex8000 devices, but we believe that other device families also have this property.

We attempt an additional constructive contribution by outlining various methods for making reconfigurable hardware safe against some classes of attacks, particularly those we called Malicious Electrical Level Threats (MELTs). For future work, we plan to extend the study to SALTs (that is, the logic signals level), provide demonstrations for all major FPGA families, and provide a more complete study of the threat model.

While research in the FPGA community is currently focused on stimulating adoption of FPGAs by demonstrating the potential of reconfigurable systems, once reconfigurable computing goes mainstream the new threats to system security we have identified will have to be thoroughly understood. The goal of our demostration was not to favor or disfavor and particular device vendor, but to point out the security threat shared by all FPGAs.

References

1. P. Allke. *Configuration Issues: Power-up, Volatility, Security, Battery Back-up.* Xilinx Inc., November 1997. Application Note 092, Version 1.1.
2. Altera, Corporation. *Altera Device Package Information - Data Sheet*, 7 edition, March 1998.
3. Altera, Corporation. *Flex 8000 Programmable Logic Family - Data Sheet*, 9.11 edition, September 1998.
4. Annapolis Micro Systems Inc., http://www.annapmicro.com. *Information on the Web.*
5. J. M. Arnold, D. A. Buell, and E. G. Davis. Splash 2. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 316–324, June 1992.
6. B. Borriello, *et. al.* The Triptych FPGA architecture. *IEEE Transactions on VLSI Systems*, 3(4):491–501, 1995.
7. F. Cohen. Computer Viruses, Theory and Experiments. *Computers and Security*, 6:22–35, 1987.
8. P. Graham and B. Nelson. A Hardware Genetic Algorithm for the Traveling Salesman Problem on SPLASH 2. In *Proceedings of FPL'95*, pages 352–361, September 1995.
9. J. Burns, *et. al.* A Dynamic Reconfiguration Run-Time System. In *Proceedings of FCCM'97*, April 1997.
10. E. Lechner and S. A. Guccione. The Java Environment of Reconfigurable Computing. In *Proceedings of FPL'97*, pages 284–293, September 1997.
11. G. McGregor, D. Robinson, and P. Lysaght. A Hardware/Software Co-design Environment for Reconfigurable Logic Systems. In *Proceedings of FPL'98*, pages 258–267, September 1998.
12. P. I. Mackinlay., *et. al.* Riley-2: A Flexible Platform for Codesign and Dynamic Reconfigurable Computing Research. In *Proceedings FPL'97*, pages 91–100, September 1997.
13. T. Shanley and D. Anderson. *PCI System Architecture.* Addison Wesley, 3rd edition edition, 1995.
14. D. Smith and D. Bhatia. Race: Reconfigurable and adaptive computing environment. In *Proceedings of FPL'96*, September 1996.
15. Virtual Computer Corporation, http://www.vcc.com. *Information on the Web.*

16. W. H. Mangione-Smith, *et al.* Seeking Solutions in Configurable Computing. *IEEE Computer Magazine*, pages 38–43, December 1997.
17. Zetex Semiconductors, http://www.zetex.com/sensors.htm. *Current Sensors.*