EMBEDDED SYSTEM CHALLENGE

IMPLEMENTATION OF HARDWARE TROJANS

HARDWARE DESIGN DOCUMENT

BY
KARTHIK GULURSHIVARAM
RUFAEL HAILEMICHAE
JORGE JIMENEZ
AJU RAJU

POLYTECHNIC INSTITUTE OF NEW YORK UNIVERSITY
CONTENTS

1. INTRODUCTION .................................................................1
2. CLASSIFICATION OF TROJAN? ..........................................2
3. OUR PROJECT .......................................................................3
   a. The SKL (Secret-Key Leaker)
      i. Motivation
      ii. Implementation
      iii. Block Diagram
      iv. Application
   b. Two-Face (VGA Killer)
      i. Motivation
      ii. Block Diagram
      iii. Implementation
      iv. Application
   c. Serial-Data-Fiddler
      i. Motivation
      ii. Block Diagram
      iii. Implementation
      iv. VHDL Code
      v. Application
4. RESULTS ..............................................................................12
   a. Synthesis Results/ Map Report (Device-Utilization)
   b. Power Report
   c. Speed Report
   d. The 3-T’s
5. CONCLUSION .....................................................................17
6. REFERENCES ........................................................................18
**LIST OF FIGURES**

*Figure 1* Classification of Trojans
*Figure 2* State Machine of the SKL Trojan
*Figure 3a* LEDs ON
*Figure 3b* LEDs OFF
*Figure 4* RTL Schematic of the SKL Trojan
*Figure 5* Simplified block diagram for VGA hardware Trojan
*Figure 6a* Alpha Logo
*Figure 6b* Trojan Image
*Figure 7* Block Diagram illustrating the Fiddler Trojan
1. INTRODUCTION

Today, with technology all around us and unavoidable, society highly relies on it to keep families connected, to shop online, to send and receive information at lightning speed, watch movies, listen to music, and much more. With all of these devices in constant use, and the enormous amount of information being transferred back and forth at any given time, at any given place, chances are high that somewhere there is someone trying to figure out how to tap into some device to steal sensitive information. In fact, people have already found many ways to hack into computers, networks, mobile devices, security systems through the use of malicious software such as Trojans. Software Trojans have been around for many years, hackers have found clever ways to secretly insert them into electronic systems, and likewise, there are people who have written many clever programs to be able to spot, quarantine and remove these high risk pieces of code. But now, there is a new threat, a threat in the form of Hardware Trojans.

More and more frequently, integrated circuits are being sent out to foreign countries to be manufactured due to the lower costs in doing so. The Defense Advanced Research Projects Agency, or DARPA, is concerned about the vulnerability of some of these ICs to deliberate malicious hardware manipulation. Many of these ICs, whether in the form of ASIC’s or FPGA’s, are designed specifically for U.S military applications. For example, certain micro-processors can be designed specifically for military satellite communications, Navy destroyers’ anti-submarine defense systems or for missile guidance systems. If a person or a group of people intentionally insert Trojans hidden deep within these ICs down to the transistor or logic gate level, and inserted in some military system with out being detected, the results could be catastrophic for our national security.

It is the objective of this project to show how easy it would be, given a design, to insert malicious hardware that could cause the device or system to behave in a way other than the
original design specifies. For example, a hardware Trojan could be inserted to leak out information, or to cause the device to fail at certain times during operation. It is also one of our goals to implement these Trojans with as little extra hardware as possible and also minimal increase in power consumption. We want to match the original Alpha design as close as possible.

2. CLASSIFICATION OF TROJANS

As introduced in the previous section, Trojans are being inserted into the Hardware description models used for FPGA’s and ASIC’s. Regularly, any technology today is researched and designed in one country and manufactured in another. Hence, it simplifies a hacker’s task who intends to insert a Trojan! It is thus a very important task for us as designers to understand the possible techniques of inserting Trojans so that we can be better equipped to detect them.

Trojans can be classified into three categories:

![Figure 1 Classification of Trojans](image)

As seen in the figure above, Trojans can be broadly classified into three categories:

1. This type of Trojan can modify the functionality of an ASIC; a simple example of this would be an encryptor which may perform encryption using a mysterious key (inserted by a hacker) known only to the person who inserted it.

2. This next type is the most obvious function of a Trojan, to reveal classified data.
3. A Trojan of this type must be given serious attention because the magnitude of the effect it can cause.

3. OUR PROJECT

A. The SKL (Secret Key Leaker)

I. Motivation

As is the case with displays being all around us, input devices are also very common in today’s high tech world. These input devices can be anything from pushbuttons, switches, touch sensors, keypads and keyboards to name a few. Electronic devices execute commands issued by us, the users, and these commands come from input devices like those just mentioned. Desktop and laptop computers are an obvious and prime example because all of them, no matter what type, manufacturer, color or size, use a keyboard as the primary input device. Given that the Alpha system also has a keyboard as the primary device used to input data, why not use the keyboard as the device that also leaks out information? We know that the keyboard uses a PS/2 interface to communicate with the FPGA, and this interface is bi-directional, meaning it can send and receive data on the same transmission line. But, the Alpha system only sends information one way, from keyboard to FPGA. With a little modification, we implemented a Trojan that leaks out the master key used in the AES encryption through the PS/2 interface, to the keyboard where the data is displayed by turning on and off the Caps Lock, Num Lock and Scroll Lock LED’s which represent a binary sequence (The Secret Key).

II. Implementation

In order to light up the LED’s of the keyboard, we needed to know the specific hexadecimal values that the keyboard controller uses as commands to turn on and off the corresponding LED. We had some trouble finding those specific values for our keyboard, so what we did instead was to use the hex value of 0xFF, which resets the keyboard controller, and in
doing so, flashes the three LED’s on then off at the same time. So to represent a binary 1, the three LED’s would turn on, and to represent a binary 0, they would turn off. Each time the LED’s turn on or off, they would stay on or off for one second. So to leak out the entire master key, which is 128 bits in length; it would take a total of 128 seconds (or 2 minutes and 8 seconds).

The reason we decided on this frequency was so that visually, we have enough time to interpret the binary sequence; too fast a frequency and our eyes would not be able to distinguish from the transitions, and too slow a frequency would be undesirably long. Being that our system clock frequency is 50 MHz, we used a counter, lets call it i_cnt, to count from 0 to 49,999,999. So every time this counter reaches 49,999,999, a complete second has gone by. Also, when i_cnt reaches 49,999,999, it increments another counter, lets call it j_cnt, which counts from 0 to 127. j_cnt is used as the index for the master key vector. For instance, MasterKey[ j_cnt], where MasterKey is a 128-bit vector that holds the master key. So, every second, MasterKey[ j_cnt] is evaluated, and if it equals a logic ‘1’, then the LED’s will turn on by sending the hex value 0xFF to the keyboard controller through the PS/2 interface. If MasterKey[ j_cnt] = ‘0’, then the LED’s will turn off. This continues until j_cnt = 127, at which point all of the bits have been evaluated and leaked out to the keyboard. This leakage will occur automatically at the end of every single data transmission, it is not a triggered Trojan. Every time the Alpha system has sent out an encrypted message, the master key will begin leaking out to the keyboard.

We used a state machine to implement and control this Trojan. It is composed of four states, see figure 2. Initially it will be in the idle state, where it waits until Alpha has finished encrypting the data and then enter the encrypt_done state. Then, it waits until the system has finished transmitting the encrypted data and enters the master key leak state. In the MK leak state, the counters start incrementing and at every second, the master key is evaluated and the command is sent to the keyboard to either turn on the LED’s or not. To turn on the LED’s, the hex value
0xFF is placed on the TX input data line of the PS/2 interface (see figure 2), then if the current bit position of the master key vector has a logic ‘1’, the write input signal of the PS/2 interface is asserted high, and the 0xFF is sent out to the keyboard, which then turns on the LED’s. If the current bit position has a logic ‘0’, then the write signal is not asserted and the LED’s stay off. When the entire master key has been leaked out, the state machine goes into the idle state, where it waits again for another transmission to be initiated.

III. Block Diagram

![Figure 2 State Machine of the SKL Trojan](image)

Figures 3a and 3b show the LEDs on the keyboard when they are ON and OFF. And Figure 4 shows the RTL schematic generated by Xilinx ISE 10.1, for the KbTop module of the Alpha design (U2), which includes three sub-modules, the PS/2 interface on the bottom left hand corner, the keyboard code to ASCII code converter on the top right corner,
IV. Applications

With the master key used for the encryption known, it is a matter of just taking the encrypted message and decrypting it using the same master key. All the data that goes through the system is now vulnerable and extremely easy to decipher. Take for instance some one entering their credit card information using a keyboard while shopping online. If there is a Trojan of this type somewhere in the computer system, and I know about it, I can wait until that person has finished entering their information, stand close to the keyboard and using a pen and pad, write down the leaked out credit card number. Another example might be in a high security government building where employees might enter some kind of access number into a door opening security system to enter specific rooms with in the building. If these security units are infected, a terrorist group might have an inside man who can now have access to highly sensitive information, weapons, equipment etc., by simply writing down the leaked out access codes.
B. Two-Face (VGA Killer)

I. Motivation

Video displays are an integral part of our daily lives. They are all around us, and come in many shapes and sizes in devices such as; PDAs, smart phones, ATM machines, cars, laptop computers, and the list goes on and on. They are vital in displaying to us, the users, the information we are sending as well as information we are requesting to see and collect. It is of great inconvenience when we are using such devices and the display for one reason or another fails to show the information. When such an event happens, most users grow frustrated and probably will not be able to finish what they have started without this critical device functioning correctly. It serves as one of the most important and used interfaces between man and machine.

With this in mind, we decided to insert a Trojan into the Alpha system, that when triggered, it would do two things; first, it renders the system inoperable and useless, or in other words it freezes the system and the user would have to reset it to be able to use it again. Second, similar to the dreaded “Blue Screen of Death” that Microsoft Windows users have come to hate, when triggered, the VGA Trojan displays on the monitor what we call the “Red Face of Death” along with a warning to the user that states that there is a system failure.

II. Implementation

In order to display any kind of predefined image on a monitor, that image needs to be stored in a ROM, and accessed when triggered. This is what happens when our Alpha system is first powered up. At power up or after a system reset, the video controller accesses the ROM where the alpha logo information has been stored (stored in the form of a coefficient file, or .coe file in one of the FPGA block RAMs) and displays the alpha logo on the monitor. Obviously we needed to store the Trojan image in an available ROM. Since the unmodified design only uses 8
out of 12 available block RAMs, we took advantage of this fact and used two of the unused block RAMs to store the coefficient file for our Trojan image. Figure 5 below shows a simplified block diagram of the video controller module with the VGA Trojan inserted.

III. Block Diagram

![Block Diagram Image]

Figure 5 Simplified block diagram for VGA hardware Trojan

The Trojan is triggered when the user types in a specific character on the keyboard, which we have assigned it to be the “A” character (Note, it is capitalized). When the Trojan has not been triggered, the video controller works as follows; the output of the comparator is logic 0, meaning that the user has not typed in the trigger character, this logic 0 serves as the select input to Mux1, which selects the output of the Text RAM block (Text Ram block stores input data from the entered by the user) to pass to Mux 2. If the system has just reset or powered up, then the show_logo select bit is a logic 1, which passes the logo RAM data to the VGA80x40 block, which displays the alpha logo on the monitor. If the system has not been reset or power up, then data from the Text RAM block is passed to the VGA 80x40 blocks. If the user enters the
trigger character “A” on the keyboard, then the output of the comparator is a logic 1, which then causes Mux 1 to select data from the Trojan ROM block to pass through to Mux 2. Assuming the system has not been reset or powered up, the data from the Trojan ROM is passed to the VGA 80x40 blocks and the Trojan image is displayed on the monitor. Figures 6a and 6b show pictures of the Alpha logo and the Trojan image.

IV. Applications

Trojans like this one can have a wide range of applications in areas ranging from consumer electronics, military and industrial systems. For example, say one of these Trojans has been inserted into an ASIC which was designed specifically for the weapons control system used by the U.S Air Force in their new line of fighter jets. If thousands of these ASICS are infected and have not been detected, and inserted into these fighter jets, then every time the fighter pilot enters a specific sequence of commands to engage an enemy jet, the Trojan is triggered and is displayed confusing the pilot while at the same time temporarily disabling the system. This can be the difference between life and death for thousands of pilots.

C. Serial Data Fiddler

I. Motivation

Communication with the outside world is critical in this day and age. We need to be able to send information back and forth. However, we must make sure that the information that is sent out is secured so that no one has hacked it and that only the intended receiver knows how
to decrypt it. With this in mind, we decided to insert a Trojan to send the plaintext instead of the encrypted text not only to confuse the intended receiver but to leak out the original message to those hackers who are looking to steal the information. It confuses the intended receiver because it will decrypt the incoming data, if the incoming data is not encrypted in the first place, then decrypting this data will produce “gibberish” information. This would help if someone is trying to learn a strategy that is being relayed from one ally to another so that the enemy can be successful.

II. Implementation

The goal of the serial data fiddler Trojan is to send the plaintext to the RS232 if a lower-case “f” was entered. To implement this, a 128 bit 2:1 multiplexer and an 8-bit comparator are required. The next section illustrates the flow path of the Trojan with the help of a block diagram.

III. Block Diagram

![Block Diagram Illustrating the Fiddler Trojan](image)

*Figure 7 Block Diagram Illustrating the Fiddler Trojan*
The ASCII value will be provided from the kbtop component. The plaintext will be provided from the pt_exp component, while the encrypted will be from the aes128_fast component. The comparator would compare the current ASCII value in binary with the binary value of f. If it matches, we will send the plaintext. It will remain that way until the component is reset. The following is the code used to implement the algorithm.

IV. VHDL Code

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY aes_mux IS PORT(
    reset, clk : IN  STD_LOGIC;
    plaintext  : IN  STD_LOGIC_VECTOR(127 DOWNTO 0);
    encrypted  : IN  STD_LOGIC_VECTOR(127 DOWNTO 0);
    ascii_in   : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
    text_out   : OUT STD_LOGIC_VECTOR(127 DOWNTO 0)
); END aes_mux;

ARCHITECTURE rtl OF aes_mux IS

    SIGNAL ascii_equal : STD_LOGIC;
    SIGNAL select_mux  : STD_LOGIC;

    BEGIN

    PROCESS(reset, clk)  BEGIN
        IF(reset=1') THEN
            select_mux<= '0';
        ELSIF (rising_edge(clk)) THEN
            IF(ascii_in = "01100110") THEN
                select_mux <= '1';
            --ELSE
            --select_mux <= '0';
            END IF;
        END IF;
    END PROCESS;

    WITH select_mux SELECT
        text_out <= plaintext WHEN '1',
                    encrypted WHEN OTHERS;

    END rtl;
```

Originally, the plan was to look for the letter f in the plaintext at the Least Significant Byte. However, we would only be looking at the first character. Further more, if the else had not been
commented out, we would be leaking only f and not the whole plaintext. This is what led to the development of this code. As soon as user enters a plaintext message, if he uses a lowercase f, the RS232 will send the plaintext instead of the encrypted text, causing a modification of the output.

V. Application

This Trojan can be used with respects to PDA’s, computers or anything that sends out encrypted data. It will leak out the original data causing incorrect processing and also revealing sensitive information.

4. RESULTS

The main design constraint when we developed these Trojans was to keep the device utilization as close as possible to the reference design. The utilization factor was another issue that we had to deal with, as we see from the reference design that the device utilizes 98% of the resources hence the Trojan that we develop has to consume minimal hardware resources yet be very dangerous. Keeping all this in mind the results section is structured to give a detailed report of our Trojans. We have listed the synthesis, map, power and speed reports and finally a comparison of everything with respect to the reference design.

A. Map Report

The SKL Trojan

Design Summary
-------------
Number of errors:  0
Number of warnings:  19
Logic Utilization:
Number of Slice Flip Flops:  1,556 out of  4,896  31%
Number of 4 input LUTs:  4,152 out of  4,896  84%
Logic Distribution:
Number of occupied Slices:  2,432 out of  2,448  99%
Number of Slices containing only related logic:  2,432 out of  2,432  100%
Number of Slices containing unrelated logic:  0 out of  2,432  0%
Total Number of 4 input LUTs:  4,352 out of  4,896  88%
  Number used as logic:  4,152
  Number used as a route-thru:  200
Number of bonded IOBs:  46 out of  108  42%
IOB Flip Flops:  6
The above summary clearly indicates that the SKL Trojan exceeds the reference design by 1\% with respect to the Slices and 2\% with respect to the 4 input Look up Tables.

**Two-Face Trojan**

Design Summary  
--------------
Number of errors: 0  
Number of warnings: 2  
Logic Utilization:  
Number of Slice Flip Flops: 1,479 out of 4,896 30\%  
Number of 4 input LUTs: 4,176 out of 4,896 85\%  
Logic Distribution:  
Number of occupied Slices: 2,406 out of 2,448 98\%  
Number of Slices containing only related logic: 2,406 out of 2,406 100\%  
Number of Slices containing unrelated logic: 0 out of 2,406 0\%  
Total Number of 4 input LUTs: 4,329 out of 4,896 88\%  
Number used as logic: 4,176  
Number used as a route-thru: 153  
Number of bonded IOBs: 46 out of 108 42\%  
IOB Flip Flops: 6  
Number of RAMB16s: 10 out of 12 83\%  
Number of BUF\textsubscript{MUX}s: 4 out of 24 16\%  
Number of DCMs: 3 out of 4 75\%  
Number of MULT\textsubscript{18X18}IOs: 2 out of 12 16\%

The device utilization remains the same even after implementing the Trojan, only the logic distribution increases by 1\%.

**Serial Data Fiddler Trojan**

Design Summary  
--------------
Number of errors: 0  
Number of warnings: 17  
Logic Utilization:  
Number of Slice Flip Flops: 1,470 out of 4,896 30\%  
Number of 4 input LUTs: 4,176 out of 4,896 85\%  
Logic Distribution:  
Number of occupied Slices: 2,406 out of 2,448 98\%  
Number of Slices containing only related logic: 2,406 out of 2,406 100\%  
Number of Slices containing unrelated logic: 0 out of 2,406 0\%  
Total Number of 4 input LUTs: 4,329 out of 4,896 88\%  
Number used as logic: 4,176  
Number used as a route-thru: 153  
Number of bonded IOBs: 46 out of 108 42\%  
IOB Flip Flops: 5  
Number of RAMB16s: 8 out of 12 66\%  
Number of BUF\textsubscript{MUX}s: 4 out of 24 16\%  
Number of DCMs: 3 out of 4 75\%  
Number of MULT\textsubscript{18X18}IOs: 2 out of 12 16\%
Again, this Trojan has been implemented by maintaining the utilization to a minimum. The logic of the Trojan has been distributed among 4 input LUTs.

B. Power Report

The SKL Trojan

<table>
<thead>
<tr>
<th>Power summary</th>
<th>I (mA)</th>
<th>P (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total estimated power consumption</td>
<td></td>
<td>95</td>
</tr>
<tr>
<td>Total Vccint 1.20V</td>
<td>51</td>
<td>62</td>
</tr>
<tr>
<td>Total Vccaux 2.50V</td>
<td>12</td>
<td>30</td>
</tr>
<tr>
<td>Total Vcco25 2.50V</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clocks</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IOs</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Inputs</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Logic</td>
<td>36</td>
<td>43</td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vcco25</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Signals</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quiescent Vccint 1.20V</td>
<td>16</td>
<td>19</td>
</tr>
<tr>
<td>Quiescent Vccaux 2.50V</td>
<td>12</td>
<td>30</td>
</tr>
<tr>
<td>Quiescent Vcco25 2.50V</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Thermal summary

<table>
<thead>
<tr>
<th>Estimated junction temperature</th>
<th>29C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ambient temp</td>
<td>25C</td>
</tr>
<tr>
<td>Case temp</td>
<td>28C</td>
</tr>
<tr>
<td>Theta J-A</td>
<td>38C/W</td>
</tr>
</tbody>
</table>

Two-Face Trojan

<table>
<thead>
<tr>
<th>Power summary</th>
<th>I (mA)</th>
<th>P (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total estimated power consumption</td>
<td></td>
<td>95</td>
</tr>
<tr>
<td>Total Vccint 1.20V</td>
<td>51</td>
<td>62</td>
</tr>
<tr>
<td>Total Vccaux 2.50V</td>
<td>12</td>
<td>30</td>
</tr>
<tr>
<td>Total Vcco25 2.50V</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clocks</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IOs</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Inputs</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Logic</td>
<td>36</td>
<td>43</td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vcco25</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Signals</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quiescent Vccint 1.20V</td>
<td>16</td>
<td>19</td>
</tr>
<tr>
<td>Quiescent Vccaux 2.50V</td>
<td>12</td>
<td>30</td>
</tr>
<tr>
<td>Quiescent Vcco25 2.50V</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
Thermal summary

<table>
<thead>
<tr>
<th>Estimated junction temperature</th>
<th>29C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ambient temp</td>
<td>25C</td>
</tr>
<tr>
<td>Case temp</td>
<td>28C</td>
</tr>
<tr>
<td>Theta J-A</td>
<td>38C/W</td>
</tr>
</tbody>
</table>

Serial Data Fiddler Trojan

<table>
<thead>
<tr>
<th>Power summary</th>
<th>I(mA)</th>
<th>P(mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total estimated power consumption</td>
<td></td>
<td>95</td>
</tr>
<tr>
<td>Total Vccint 1.20V</td>
<td>51</td>
<td>62</td>
</tr>
<tr>
<td>Total Vccaux 2.50V</td>
<td>12</td>
<td>30</td>
</tr>
<tr>
<td>Total Vcco25 2.50V</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Clocks</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IOs</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Inputs</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Logic</td>
<td>36</td>
<td>43</td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vcco25</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Signals</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Quiescent Vccint 1.20V</td>
<td>16</td>
<td>19</td>
</tr>
<tr>
<td>Quiescent Vccaux 2.50V</td>
<td>12</td>
<td>30</td>
</tr>
<tr>
<td>Quiescent Vcco25 2.50V</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

The power results were obtained by running the XPower tool in Xilinx ISE Suite.

C. Speed Report

The SKL Trojan

Speed Grade: -4

Minimum period: 12.263ns (Maximum Frequency: 81.546MHz)
Minimum input arrival time before clock: 4.635ns
Maximum output required time after clock: 8.635ns
Maximum combinational path delay: No path found

Two-Face Trojan

Speed Grade: -4

Minimum period: 12.263ns (Maximum Frequency: 81.546MHz)
Minimum input arrival time before clock: 4.635ns
Maximum output required time after clock: 8.604ns
Maximum combinational path delay: No path found
Serial Data Fiddler Trojan

Speed Grade: -4

Minimum period: 12.263ns (Maximum Frequency: 81.546MHz)  
Minimum input arrival time before clock: 4.731ns  
Maximum output required time after clock: 8.604ns  
Maximum combinational path delay: No path found

Of the results listed above, the synthesis report is most significant with respect to this project. This is due to the fact that it should not be obvious that the Trojans that we have designed and inserted increase the total device utilization by a significant factor, or then it would become obvious that somebody has modified the design. Therefore, we gave utmost importance to this fact and aimed at achieving a low utilization (with respect to the reference design). Similarly, a timing analysis and a power analysis of the design are equally important, since we don’t want to fry the FPGA by inserting resource demanding Trojans. Neither do we want the FPGA to run slower (output slower than expected). All these facts would clearly indicate that there has been an intrusion.

D. The 3 T’s

To summarize we have made this table which compares the various characteristics with respect to the reference design.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Reference Design</th>
<th>Trojan#1 The SKL</th>
<th>Trojan#2 Two-Face</th>
<th>Trojan#3 Serial-Data Fiddler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slice FF’s (Out of 4896)</td>
<td>1469 (30%)</td>
<td>1,556 (31%)</td>
<td>1479 (30%)</td>
<td>1470 (30%)</td>
</tr>
<tr>
<td>Number of occupied Slices (Out of 2448)</td>
<td>2401 (98%)</td>
<td>2432 (99%)</td>
<td>2446 (99%)</td>
<td>2406 (98%)</td>
</tr>
<tr>
<td>Total number of 4 input LUTs (Out of 4896)</td>
<td>4046 (85%)</td>
<td>4152 (84%)</td>
<td>4057 (82%)</td>
<td>4176 (85%)</td>
</tr>
<tr>
<td>Offset In (ns)</td>
<td>4.731</td>
<td>4.635</td>
<td>4.635</td>
<td>4.731</td>
</tr>
<tr>
<td>Offset Out (ns)</td>
<td>8.604</td>
<td>8.635</td>
<td>8.604</td>
<td>8.604</td>
</tr>
<tr>
<td>Power in mW*</td>
<td>95</td>
<td>95</td>
<td>95</td>
<td>95</td>
</tr>
</tbody>
</table>

*XPower analysis without .vcd file.
All the numbers are close to the reference design which indicates that the Trojans are difficult to identify and are embedded perfectly into the design.

5. CONCLUSION

The objective of this project was to implement Trojans into the given Alpha system with as minimal extra hardware as possible as well as to successfully either modify device functionality, leak out information or deny the services to the user. In this project we have implemented three different Trojans that do not disrupt the normal device functionality when not triggered, without increasing the total percent of logic or distribution utilization by more than 1% and also without increasing the total power consumption. These are just some of the types of Trojans and techniques that can be used to insert malicious hardware into today’s and tomorrow’s designs. Without more knowledge of how and who can insert hardware Trojans, designs that are manufactured offshore are more vulnerable to these threats. We believe that through our project, we have successfully aided to this cause.
REFERENCES


