Vulnerability Analysis of Security ICs Against Laser Fault Injection

Kyle Mitard
Advisor: Prof Shahin Tajik
April 2023

An MQP Submitted to the Faculty of the Worcester Polytechnic Institute in partial fulfillment of the requirements for the Undergraduate Degree of Engineering In Electrical and Computer Engineering

This report represents the work of one or more WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on the web without editorial or peer review.
Abstract

Lasers offer an unparalleled level of precision when injecting faults into FPGAs and other integrated circuits due to their ability to precisely target specific areas of a chip with minimal effects on everything else. However, the single biggest challenge in conducting a laser fault injection attack is finding where to fire the laser, which we solve using this paper. We first locate LUTs and registers on a Xilinx Kintex-7 FPGA via photon emission, and from there we use an automated "brute force" search that we developed for an AlphaNov laser microscope station to draw a fault sensitivity map for each type of logic element. Using the information we gathered from these maps and our experience developing the procedure, we suggest possible countermeasures and demonstrate a laser fault attack on a simple password lock.
Contents

1 Introduction .................................................. 3

2 Background ...................................................... 3
   2.1 FPGA Architecture ........................................ 3
   2.2 Photon Emission ........................................... 4
   2.3 Laser Fault Injection ....................................... 6

3 Device Under Test and Experimental Setup ....................... 7

4 Locating Logic Elements Via Photon Emission .................... 9
   4.1 LUTs ..................................................... 10
   4.2 Registers ................................................ 10

5 Automatically Finding Fault-Sensitive Locations .............. 13
   5.1 LUTs ..................................................... 14
      5.1.1 DUT Configuration .................................... 14
      5.1.2 Tester Configuration .................................... 14
      5.1.3 Search Parameters ..................................... 14
      5.1.4 Results And Discussion ............................... 15
   5.2 Registers ................................................ 15
      5.2.1 DUT Configuration .................................... 15
      5.2.2 Tester Configuration .................................... 17
      5.2.3 Search Parameters ..................................... 17
      5.2.4 Result And Discussion ............................... 18

6 Laser Fault Attack On A Simple Password Lock ............... 19
   6.1 Target Design And Implementation ........................ 19
   6.2 Attack Vector .............................................. 20
   6.3 Results And Discussion .................................... 21

7 Conclusion ..................................................... 22
1 Introduction

Programmable logic devices, particularly field-programmable gate arrays (FPGAs) can be used throughout industries where it is not practical to develop and fabricate an application-specific IC, mainly due to cost and volume. They achieve this by having thousands of digital circuit elements and the routing capabilities to connect any combination of them, allowing them to implement any digital circuit by loading a design through a “bitstream.” This can be done at any time, hence being “field-programmable.” However, they operate under the assumption that everything operates as designed, and in real-world conditions that sometimes isn’t true. Sometimes devices can experience “faults” where something goes wrong (e.g. a bit in memory is randomly flipped), and although random environmental factors can cause faults, other times they can be deliberately injected by an attacker.

As a means of fault injection, lasers offer perhaps the highest level of precision and control by allowing the attacker to specifically inject faults into one logic element with limited effects on everything else. Other forms of fault injection, such as voltage glitching and electromagnetic pulse attacks, are much harder to control because they target the whole chip.

With this precision, however, does not come less difficulty; by far the biggest hurdle to overcome in a laser fault attack is where to aim the laser. Attempting to overcome this opens up a rabbit hole of reverse engineering – there are techniques used to locate specific circuit elements in integrated circuits, but those have their limitations. Nonetheless, they do help immensely in the next step, which is to take shots and see what works.

This brings us to the primary purpose of this paper: automate that step of trying and seeing what works. By automatically searching for fault-sensitive locations, we draw a heat map of sorts to locate which areas of the chip are sensitive to laser faults. Then using this heat map, we can plan and execute an attack on a target. Furthermore, because we are targeting an FPGA, we show that bespoke bitstreams can be made to streamline all steps of the process.

All of the source code for the work done in this paper is open-source and available at [1].

2 Background

2.1 FPGA Architecture

An FPGA is a programmable logic device that consists of look-up tables (LUTs) for combinational logic and registers for sequential logic, among other things. Generally speaking, these LUTs and registers are dispersed evenly throughout the dye in a grid of “logic cells” with the ability to route their inputs and outputs to other LUTs/registers within the same logic cell or anywhere throughout the chip. In the case of the Kintex-7, these logic cells consist of four 6-input LUTs and eight registers, as shown in Figure 1.
LUTs are implemented as a multiplexer with SRAM cells at the inputs. By writing the respective outputs of a truth table to the SRAM, the LUT can implement any logic function between the select bus and the output of the multiplexer. An example of a three-input AND gate implemented with an 8:1 multiplexer using this method is in Figure 2.

The registers are each single D flip-flops, where the input is latched to the output on the rising edge of a clock signal. According to Xilinx’s documentation [2], the registers are written on rising clock edges and are reset to zero synchronously with an active-high reset signal.

The process of programming an FPGA involves designing modules in a hardware-descriptive language (HDL) such as Verilog, and a series of automated tools are used to configure LUTs/registers, where to place them, and how to route connections between them, with options for manual overrides throughout the process. For Xilinx FPGAs, the official tool to do this is Vivado, which is what we used. These automated tools finally create a “bitstream” which is then loaded to the FPGA, thus programming it.

2.2 Photon Emission

Current flowing across a P-N junction emits photons due to the difference in energy between the conduction and valence bands of silicon. This effect is the operating principle behind light-emitting diodes (LEDs). Within the context
of integrated circuits, it has long been used for failure analysis by looking at a silicon dye under a sensitive infrared camera through a microscope [3]. This is because an unintended short circuit caused by manufacturing defects emits an unusually high amount of photons. An example of this is shown in Figure 3.
By overlaying an image of the dye taken with the same lens in the same
location, we can somewhat precisely determine the location of the circuit element
emitting the photons. For example, Figure 4 shows the same defective circuit
in Figure 3 but with an image of the dye at that location overlayed.

![Figure 4: Photon emission of a defective integrated circuit overlayed over an
image of the dye.](image)

On working circuits, this sort of "photon emission" also occurs, albeit with
much dimmer spots, when a transistor switches, and we can use it similarly to
locate LUTs and registers. Although this phenomenon can be used on its own
as a side channel to extract cryptographic information on its own [4], in this
paper, we only use it to locate LUTs and registers so we know where to search
for fault-sensitive spots with the laser.

### 2.3 Laser Fault Injection

In the same way that current flowing in a P-N junction emits photons, it is
possible to induce current across a P-N junction by striking it with photons. It is
the operating principle behind photodiodes, and within the context of integrated
circuits, it allows us to switch transistors, allowing us to flip SRAM cells and flip-
flops. Although this has long been known as a cause of random faulty behavior
in ICs deployed spacecraft due to cosmic rays [5], security researchers have also
shown this could work as a deliberate attack using a laser [6], defeating things
like secure boot [7] and physical security measures [8].

These attacks are especially possible on devices with "flip-chip" designs,
with the substrate on top and the metal layers on the bottom. It is upside
down compared to a more traditional IC design and primarily exists for perfor-
ance reasons. It allows the pads on top of the metal layers to directly contact
the PCB, which reduces the parasitic effects on signal integrity caused by this connection. Although it is possible to inject faults with lasers on non-flip-chip designs, the process is much more invasive due to the polishing required to bypass the metal layers.

Flip-chip or not, one of the main challenges is finding where to aim the laser, since out of the millions of transistors on an integrated circuit, only a handful of transistors will produce the desired faults when hit with a laser. Furthermore, the high density of integrated circuits makes reverse engineering at a transistor level near impossible. Although photon emission greatly helps narrow down the location, the only way to search from there is to guess randomly. Thus this paper describes a brute-force search routine to automate this process.

When performing laser fault attacks, there is also the danger of permanently damaging the device under test due to either creating a short circuit between power and ground or simply burning the dye from the laser radiation. In fact, the photon emission shown in Figures 3 and 4 actually come from a region of our FPGA that was damaged this way. We produced the results shown in this paper using the same FPGA but in an area physically far away in order to make photon emission easier and mitigate the effects of the damage.

3 Device Under Test and Experimental Setup

The device under test is a Digilent Genesys 2 board featuring a Xilinx Kintex-7 FPGA with part number XC7K325T-2FFG900C. We exposed the silicon dye by removing the fan and cleaning the surface with isopropyl alcohol, leaving us with the backside of the silicon dye exposed, as shown in Figure 5. The Kintex-7 is a "flip-chip" design, which means it is soldered upside down, allowing us to bypass the metal layers. Interestingly, this makes fault injection much easier and less invasive since it removes the need to shave down the dye.

We then placed the board with the exposed dye under an AlphaNov laser microscope station (LMS) equipped with a laser source and an IR camera with three zoom lenses - 2.5X, 20X, and 50X. The laser source is the High Pulse Performance variant of AlphaNov’s Pulse On-Demand Module (PDM HPP), capable of up to 4 A pulses at a wavelength of 1064 nm. However, we rarely exceeded 1.2A, as we found more can permanently damage the FPGA. Both the LMS and PDM HPP are controlled using a Windows computer with software and C/C++ libraries provided by AlphaNov.

As a tester for the DUT, we used a modified Arduino Uno clone, pictured in Figure 7. Because the Genesys 2 board uses 3.3V logic levels at the PMOD headers, we had to modify the Arduino to run on 3.3V. We did this by replacing the 5V regulator with a 3.3V regulator and replacing the fuse with a diode as described in a guide from Adafruit [9]. We connected the digital I/O pins to the PMOD headers of the Genesys2 board, and it communicates with the lab PC over UART, as shown in Figure 6.

The tester sends inputs to and reads outputs from the board and reports any faults to the lab PC over a serial port. This allows us to control as much as
possible without touching the DUT, which is enclosed by the LMS housing to shield us from dangerous laser radiation or the DUT from outside light rendering photon emission impossible to detect. In addition, the lab PC can reprogram the DUT over JTAG, although this has to be done by SSHing into another PC with the enterprise Vivado license required for the Genesys 2 board. A general block diagram for all of these pieces is shown in Figure 8.
4 Locating Logic Elements Via Photon Emission

The first step in preparing for a laser fault injection attack is to locate the desired logic element to attack, and this is done via photon emission. For this step, the DUT is programmed with two modules that can be enabled using the tester: the logic element in question and a ring oscillator. The logic element is set up such that when enabled, it has an output that rapidly flips between 0 and 1 as fast as reasonably possible, which creates a photon emission spot. The ring oscillator then exists as a bright "beacon" to spot the general area and
approximate the boundaries of the logic cell that has the element. The steps as they are implemented in the LMS camera software provided by AlphaNov are as follows...

1. Aim the camera at the general location of the dye using the navigation settings and take a "static image."

2. Without moving the camera, switch to photon emission settings.

3. With everything disabled, take a "dark image."

4. Turn on the desired elements and subtract the dark image from the current image. This will ideally subtract the noise and leave behind the bright spots in the photon emission.

5. Overlay the photon emission image over the static image to see the location on the navigation camera.

Using the photon emission images, we can roughly approximate where the element is, and based on that, we can determine a search zone for fault-sensitive locations.

4.1 LUTs

To locate LUTs, we implemented a 6-input XOR gate where one of the inputs is a 100MHz MMCM clock at the location shown in Figure 9. With this, we found a single bright spot in photon emission, as shown in Figure 10. This spot presumably corresponds to the location of the LUT, and we decided to search the general area it takes up.

4.2 Registers

In each logic cell of the Kintex-7, there are eight registers. To locate them, we created a "where is this register?" Verilog module, whose block diagram is shown in Figure 11. In this module, there are eight parallel registers whose input is the output fed through an inverter, which causes them constantly toggle between 1 and 0. In implementation, we placed the cells that implement the inverters far from the registers such that their photon emission does not intersect. In addition, there is an 11-stage ring oscillator with four stages within the same logic cell as the registers, and the rest of them are in the adjacent cells, as shown in Figure 12. The registers and ring oscillator can be individually enabled over serial through the tester. We overclocked the DUT to 800 MHz, which caused Vivado to report timing violations, but it worked nonetheless, ultimately making the photon emission brighter.

When taking photon emission images, we first took one image with all of the registers enabled. Then we disabled every register and enabled them one at a time, taking an image for every register. By comparing the images, we annotated the image with all registers with which spot came from which register. The
Figure 9: Device-view implementation of the 6-input XOR gate.

Figure 10: The dye of the FPGA, with the photon emission of a single lookup table overlayed

results of this process are shown in Figure 13, and based on them, we decided to search the location of the top two rows of spots since they had just about every register in a line roughly within the boundaries of the logic cell.
Figure 11: Block diagram of the "where is this register?" verilog module.

Figure 12: Device-view implementation of "where is this register?" verilog module, with locations of each register, their respective inverters, and the ring oscillator annotated.
5 Automatically Finding Fault-Sensitive Locations

Once we locate the LUTs/registers using photon emission, the next step is to find the fault-sensitive locations. Using the C/C++ libraries for the LMS supplied by AlphaNov, the tester, openCV-python [10], and pySerial [11], we developed an automated brute-force routine to draw a map of fault-sensitive locations.

Firstly, we take a picture of the location on the DUT to search (the "map image") manually using the LMS camera software. To draw on the map image, we need to know how to translate the coordinate system of the LMS controls to pixels on the map image. After taking the map image, we manually aim the laser at a reference point \((x_{\text{refLMS}}, y_{\text{refLMS}})\) within the map image, and using an image editing software the location of the same reference point \((x_{\text{refIMG}}, y_{\text{refIMG}})\) is manually identified on the image. With this reference point, any point identified on the LMS can be identified on the map image using

\[
x_{\text{IMG}} = -A (x_{\text{LMS}} - x_{\text{refLMS}}) + x_{\text{refIMG}}
\]

\[
y_{\text{IMG}} = -A (y_{\text{LMS}} - y_{\text{refLMS}}) + y_{\text{refIMG}}
\]

where \(A\) is the conversion factor from millimeters to pixels. We manually measured this in advance using the scale given in the LMS camera software, assuming negligible distortion in the camera. There is a minus sign in (1) and (2) because on the LMS, the positive x and y directions are left and up, respectively, which is the opposite of the convention used in computer images.

After identifying a reference point, a bounding box for the search is determined by manually aiming the laser at two corners. The bounding box is then drawn on the map image. Finally, the automatic search starts. The LMS moves in a snake pattern across the entire bounding box, repeating the following process for each point...

Figure 13: The photon emission for all eight registers, annotated with each individual register.
1. Run the PDM HPP for a few seconds

2. Get a report of test results from the Arduino

3. If a fault is detected...
   (a) Mark the fault spot on the map image
   (b) Take a screenshot of the camera feed
   (c) Reprogram the DUT
   (d) Rerun the test to verify the DUT was not permanently damaged

Taking a screenshot on top of the camera feed is necessary in the case of faults because the method of drawing on the map image is somewhat inaccurate. The screenshots serve as both a simple workaround issue and an easier reference to follow when attempting to manually reproduce our automated results.

5.1 LUTs

5.1.1 DUT Configuration

The DUT was configured as a 6-input XOR gate in the same location as the one shown in Figure 9. We chose an XOR gate because the number of inputs that produce an output of 1 and an output of 0 is equal and somewhat evenly distributed from inputs 000000 to 111111. This means we can compare the relative sensitivity of zero-to-one faults (expected 0, received 1) and one-to-zero faults (expected 1, received 0).

5.1.2 Tester Configuration

The inputs and output of the XOR gate are on PMOD header JA connected to the digital I/O pins of the Arduino. When it receives the command to start over serial, it cycles through all 64 possible inputs sequentially, starting at 000000 and ending at 111111. It compares the actual output to the expected output, and for every input, it prints a character over serial, signifying whether they match. If they don’t match (i.e. a there is a fault), it writes the ASCII character for the expected output (0 or 1). If not, it writes a tilde (‘~’). A newline signals the end of a test. An example test is shown in Figure 14.

5.1.3 Search Parameters

The laser was set to output pulses at a frequency of 1 kHz at a peak current of 1.000 A, which we found to be the lowest current that can induce faults in LUTs based on trial and error. The photon emission results shown in Figure 10 show one large spot and a smaller spot right above it, so the search area was immediate vicinity of these spots.
5.1.4 Results And Discussion

The results - shown in Figure 15 - roughly align with the large photon emission spot shown in Figure 10 and consist of almost all one-to-zero faults. This shows that the DUT is much more sensitive to these faults. As a countermeasure, we suggest considering active-low versus active-high logic in designs. If a security-critical signal controlled by the output of a LUT is high most of the time, it would be easier to fault than if it was low most of the time.

5.2 Registers

5.2.1 DUT Configuration

On the DUT, we used the same eight registers shown in Figure 12 configured as D flip-flops clocked using a 100 MHz MMCM clock. The input of every register is a single external switch with external write-enable and reset signals controlled by pushbuttons. This is to manually verify the DUT is not permanently damaged.

Initially, we connected the outputs of these D flip-flops directly to the tester, and if the tester detects any rising or falling edges, that register is considered faulty. However, we found it is possible to create glitches in the output signal without actually flipping the register, as shown in Figure 16. Our solution was to implement a debouncer on the outputs of the flip-flops.

A debouncer is a circuit used on any digital inputs driven by mechanical switches, such as pushbuttons. Typically these switches are closed by physically touching two metal contacts, and often these contacts tend to bounce off each other. This effectively causes a single rising or falling edge to be interpreted as many edges as the input constantly "bounces" between 1 and 0, as shown in Figure 17.

We implemented the debouncer in the DUT, which operates based on a counter; the debounced outputs are separate registers fed by (but physically...
Figure 15: Fault sensitivity map of an XOR gate, with zero-to-one faults in blue and one-to-zero faults in green.

Figure 16: Oscilloscope graph of a signal that was glitched, but not altered.

located far away from) the target D flip-flops. They are only written when their input doesn’t change for 2000 clock cycles, which corresponds to 2 \( \mu \)s at a clock frequency of 100 MHz.
5.2.2 Tester Configuration

The debounced outputs of the DUT are on PMOD header JA, each connected to digital I/O pins on the tester. Each pin triggers an interrupt on a rising or falling edge, which sets a flag indicating the respective register flipped. The lab PC then reads the flags for all eight registers over serial. However, through the Arduino framework, the Arduino Uno only supports two specific digital I/O pins for interrupts [12], so we had to use the pinChangeInterrupt [13] library to add interrupt functionality to all eight pins.

Another issue is that most Arduino boards reset when their serial port is opened [14]. The problem with this is that in a test, firing the laser comes before opening the serial port for the tester. The solution was to write the flags to the Arduino’s EEPROM, which is preserved between resets, and reset the flags in the test after reading them.

5.2.3 Search Parameters

The laser was set to output pulses at a frequency of 1 kHz at a peak current of 1.200 A. The photon emission results shown in Figure 13 show a line of all eight registers towards the top, so the search area was this line and its immediate vicinity. By narrowing the search area to this single line, we reduced the search time from upwards of 6 hours down to roughly 90 minutes.
At first, we did not reprogram the DUT upon detecting a fault, since all we should be doing is flipping a register and not altering it. However, we found that flipping a register alters it such that it cannot be manually written anymore. That being said, so long as the laser power is not too high, this is not permanent and reverts to normal when reprogramming DUT.

5.2.4 Result And Discussion

The results are shown in Figure 18. Curiously, this fault map does not follow the photon emission pattern, and one register (register 4 in blue) doesn’t show up at all. In addition, when manually flipping registers, the single purple spot for register 5 was just another spot for register 0 (black). In fact, when looking back at the test logs, we found that both registers flipped.

The primary limitation of this fault map is that it doesn’t reflect the possibility of flipping multiple registers at one spot, as happened with not only register 5 but many other spots. When this happens, the map prioritizes the highest numbered register.

![Figure 18: Fault sensitivity map of the registers in a logic cell, color-coded by register with a key to the left.](image)

Although these multiple-register spots may be a result of random vibrations during the testing, particularly due to the elevator in close vicinity to the LMS, it is also possible that one spot can flip multiple registers. In either case, this suggests a possible countermeasure - if one were to know a register that has significant overlap in fault sensitivity with other adjacent registers, then it can be used for tamper detection. That is to say, when some security-critical register(s) is faulted the tamper-evident register is inadvertently faulted simultaneously. Since the tamper-evident register should never change under normal conditions, we can detect an attack by detecting when it changes.

Similarly, glitches can also be detected as a means to detect tampering, particularly since we found glitching registers is significantly easier than flipping them. In manual tests, it took numerous glitches on a register many times before we could flip the same register once.
6 Laser Fault Attack On A Simple Password Lock

Lastly, to demonstrate the possibilities for applying these results, we created a simple finite state machine-based password lock and bypassed it using the knowledge we gathered from the results shown in Figure 18.

6.1 Target Design And Implementation

We implemented the password lock as a finite-state machine that follows the state diagram shown in Figure 19. The password is entered as a seven-bit binary number during the locked state, encoded as 00. When the enter button is depressed, it enters the input wait state, encoded as 01, where it waits for the user to release the button. Upon releasing the enter button, it enters the authenticate state, encoded as 10. From there, it will either enter the unlock state, encoded as 11, if the password is correct or return to the locked state if it is not. From the unlock state, it can be reset back to the locked state using a reset input.

![State machine diagram for the password lock.](image)

The device-level implementation is shown in Figure 20. Two registers encode the state with a binary encoding, with a third register for the open signal, and one LUT used to set each register. These registers are the same as registers 0, 2, and 7, respectively, in the "where is this register?" module shown in Figure 12. We placed the fsmState[1] register specifically for the attack based on our experiences manually verifying the results in Figure 18 (see section 6.2), with the others placed arbitrarily in close proximity. Vivado automatically placed the LUTs that handled next-state logic in implementation.
All of the DUT’s inputs and outputs are on PMOD headers JA and JB, and by connecting them to the tester’s digital I/O pins, we made a serial interface for the password lock. The seven-bit password is entered as an ASCII character through the serial terminal, and the enter button is wired to an external switch. The output is connected to pin 2 and triggers an interrupt on a rising or falling edge. When this happens, a message is shown in the serial terminal. A sample of this output under normal operation is shown in Figure 21.

6.2 Attack Vector

Our attack involves flipping one of the state registers using the laser, which causes an otherwise forbidden state transition, shown in Figure 22. The process is as follows...
1. Start in the locked state (00)
2. Press and hold the enter button to enter the input wait state (01)
3. Use the laser to flip the most significant state register, changing the state to 11

Through this we can effectively bypass the authentication state (10).

![State-machine diagram for the password lock with the attack vector added in red.](image)

Figure 22: State-machine diagram for the password lock with the attack vector added in red.

We used the same laser power as the automated test described in section 5.2 – 1.200 A peak current, though we reduced the pulse width to 200 ns. The pulse frequency was varied but never exceeded 100 Hz because we wanted to do single shots.

### 6.3 Results And Discussion

Our attack was successful, as shown by the serial terminal output in Figure 23, and though we could do it in a reasonable amount of time with enough laser shots, it takes many attempts. Similarly to the automated test described in section 5.2, the DUT could not be reset normally after this attack - the only ways to exit the unlock state are to flip the same register back to zero using the laser or to reprogram the DUT. In either case, the DUT returns to normal behavior and does not appear permanently damaged.

Beyond the fault-sensitivity of the registers, what primarily made this attack possible was the hamming distance of 1 between the security-critical state and the others. Specifically, the difference between the unlock state (11) and the input wait state (01) was only a single bit, meaning a transition between these states, faulty or not, only requires flipping one bit. If it took flipping multiple
bits to get to the unlock state, then this attack would be both much more difficult and much less reliable.

Though there are no ways to rearrange this two-bit encoding such that the hamming distance between the unlock state and all other states is more than one, using more than two state registers to encode the same four states makes this possible. Furthermore, the additional states given by this could represent "forbidden states" as a form of tamper detection. These states have no means of entering them, so when they enter them, there is almost certainly faulty behavior, attack or not. A simple and intuitive way to implement this is one-hot encoding, where there is one register for every state, and the register corresponding to the current state is set to 1 with the rest set to 0 (e.g. 0001, 0010, etc.). Thus, at any given time, exactly one register must be set to 1, and if there's any more or any less, it is an illegal state, and therefore the system was tampered with or damaged.

Glitches offer another possible route for tamper detection, since through the many attempts at flipping registers, there were many glitches in the process. The interface did detect glitches, as shown by the multiple "locked" messages preceding the single "open" message in Figure 23. These glitches were detected because, presumably, the glitches were large enough in amplitude to trigger an interrupt in the tester despite the output not changing.

7 Conclusion

In this paper we showed an automated brute-force search process to search for fault-sensitive spots on an FPGA. Although the search process was lengthy, we showed that photon emission is useful for narrowing down the search, and in the
end, we produced practical results that are applicable in an attack on the same hardware. Although the photon emission images and laser searches involved both dedicated "testing configurations" that are only practical on programmable devices, this technique or some subset thereof could be adapted to ASICs or other non-programmable devices where photon emission can reveal potential locations of interest.

Despite this, there are numerous limitations to our setup, some of which we could address with future work and an improved setup. For one, the LMS was on the second floor of an academic building near an elevator, causing numerous vibrations due to people walking throughout the building or using the elevator. This heavily restricts the times when we can get reliable results.

In addition, the DUT was not secured in the LMS, which had three effects. For one, this meant that the coordinates of the laser aiming was not consistent between trials, so when manually reproducing the faults achieved in the automated tests, we were only going off of images of the location. In our experience, this makes the process more tedious, but not impossible. Another effect was that the DUT was not perfectly level, meaning the laser would drift out of focus when going long distances or sitting for a long time. This, plus the lack of an automatic focus, meant all tests had to be supervised to refocus the laser over time. Finally, the combination of these effects could be a source of the inaccuracies in the fault spots on the map image.

We could have addressed three of these issues by building a plate to secure the DUT under the LMS in the future. Also, despite the lack of automatic focus built into the AlphaNov LMS, we could attempt to implement it in software.

Nonetheless, we advise anybody designing for FPGAs to account for these vulnerabilities. Although nothing can be done at the silicon level for chips already designed and built, we discovered many possible countermeasures that are easy to implement in HDL, including many tamper detection mechanisms. As for those who have control of the device at the silicon level, whether it’s ASICs or the ones designing FPGAs, we suggest running these tests in-house and investigating the effects of changes in layout on fault sensitivity.
References


