FPGA’s: a new attack surface for embedded adversaries.

John Dunlap

About Me

- @JohnDunlap2
- Security Researcher
- Reverse Engineer
- Avid collector of bad software
- Work for GDSSecurity doing code review / RE / Research

What’s this talk about?

- FPGA’s – What are the security concerns?
- Also it’s about moving fast, because this is a deep deep topic in a relatively short talk.

Who is the intended audience?

- Security Engineers / Static Analysis People who want to get into FPGAs
- Not deep FPGA SME’s, you may not get as much out of this due to time constraints.

What are FPGAs?

Field Programmable

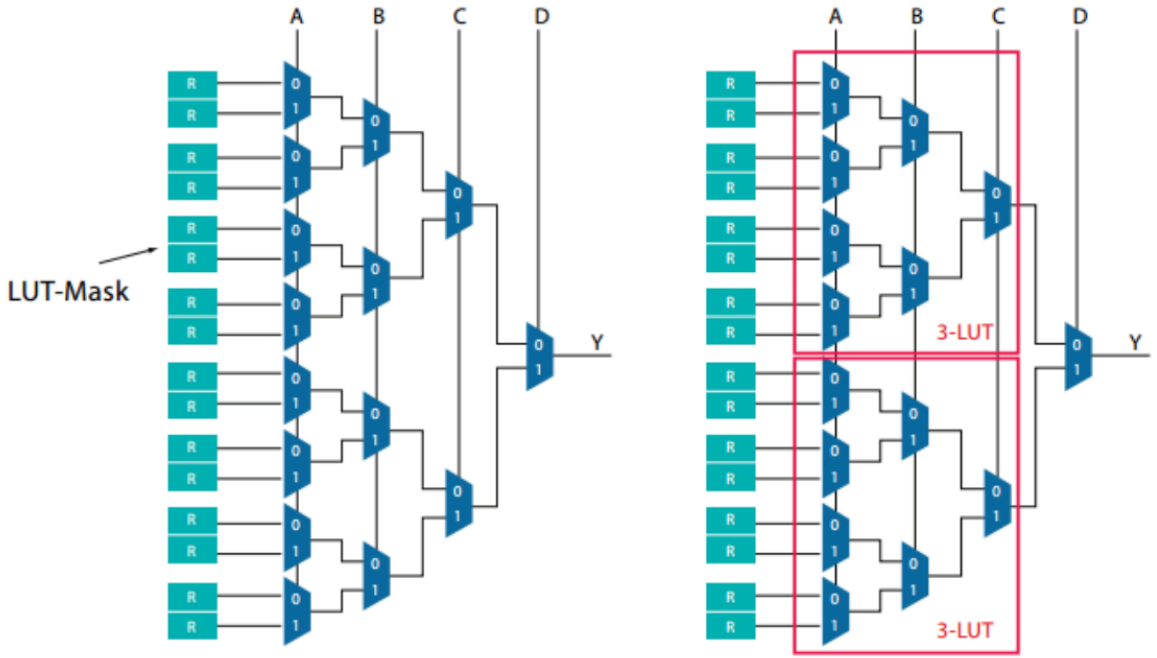
- You can think of it as a CPU you can reprogram yourself
- All of the behaviors of a CPU are yours to define – registers, memory etc..

Gate Array

- Ostensibly an array of Gates.
- Not actually, but you are meant to think that way

In Reality: LUTs (https://electronics.stackexchange.com/questions/169532/what-is-an-lut-in-fpga)

Figure 2. Building a LUT



a'b'c'd' + abcd + abc'd' = 1000 0000 0000 1001 = 0x8009

share edit flag

answered May 8 '15 at 8:04

Most of the time. There are likely deafening sounds of “well actually” in the audience at the moment.

In reality there are many types of hybrid devices that behave differently.

FPGA Flavors (This Relevant to Security!)

- FPGA’s come in a dizzying variety of types that all behave differently
- Different power consumption levels
- Different sizes
- Different storage methods
- Hybrid FPGA/CPLD setups

FPGA Storage Types

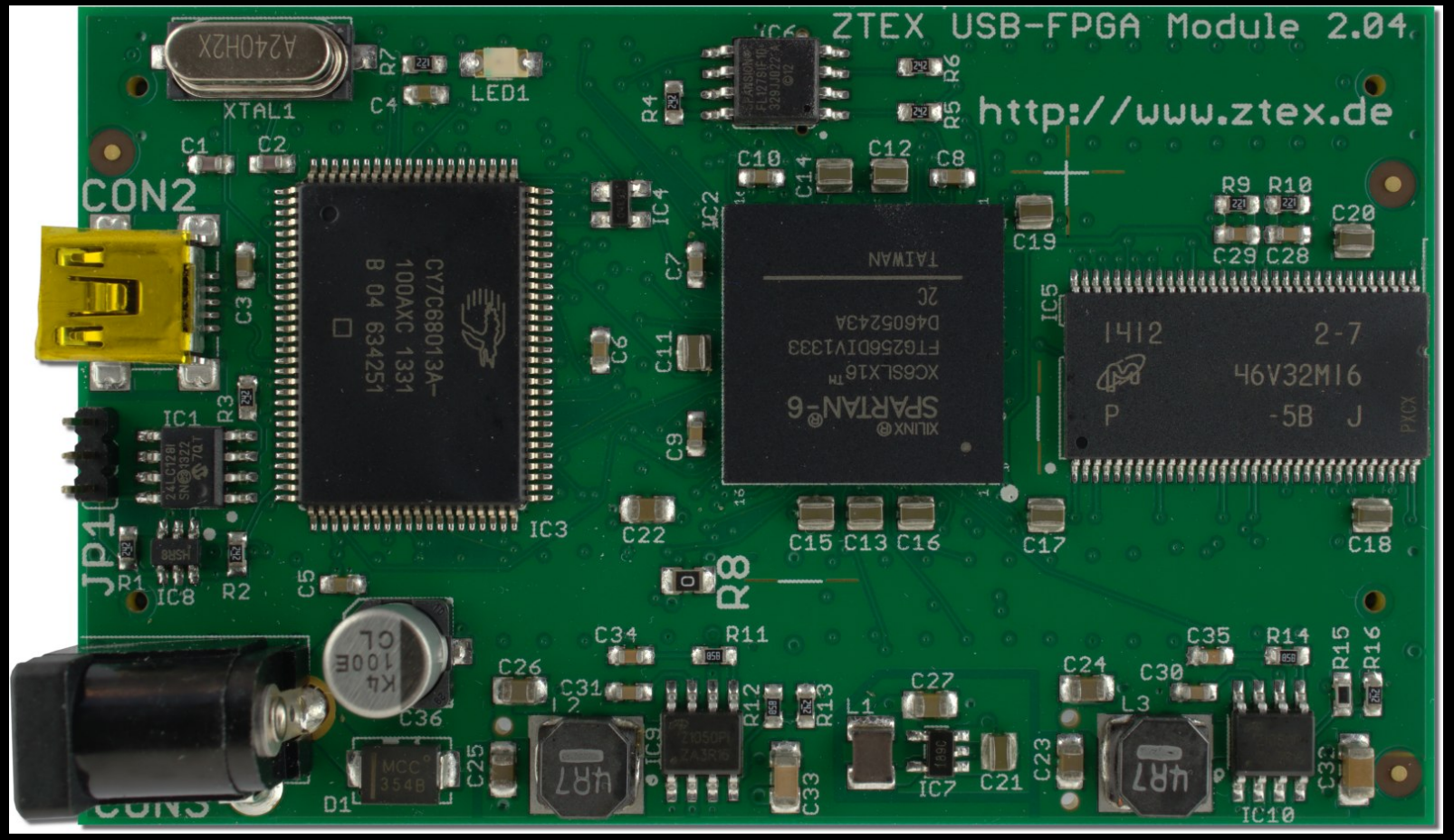
- SRAM
- Antifuse
- PROM
- EEPROM
- EPROM
- Flash

SRAM

Static Ram

- Boots from a CMOS
- Can be easily reconfigured on the fly

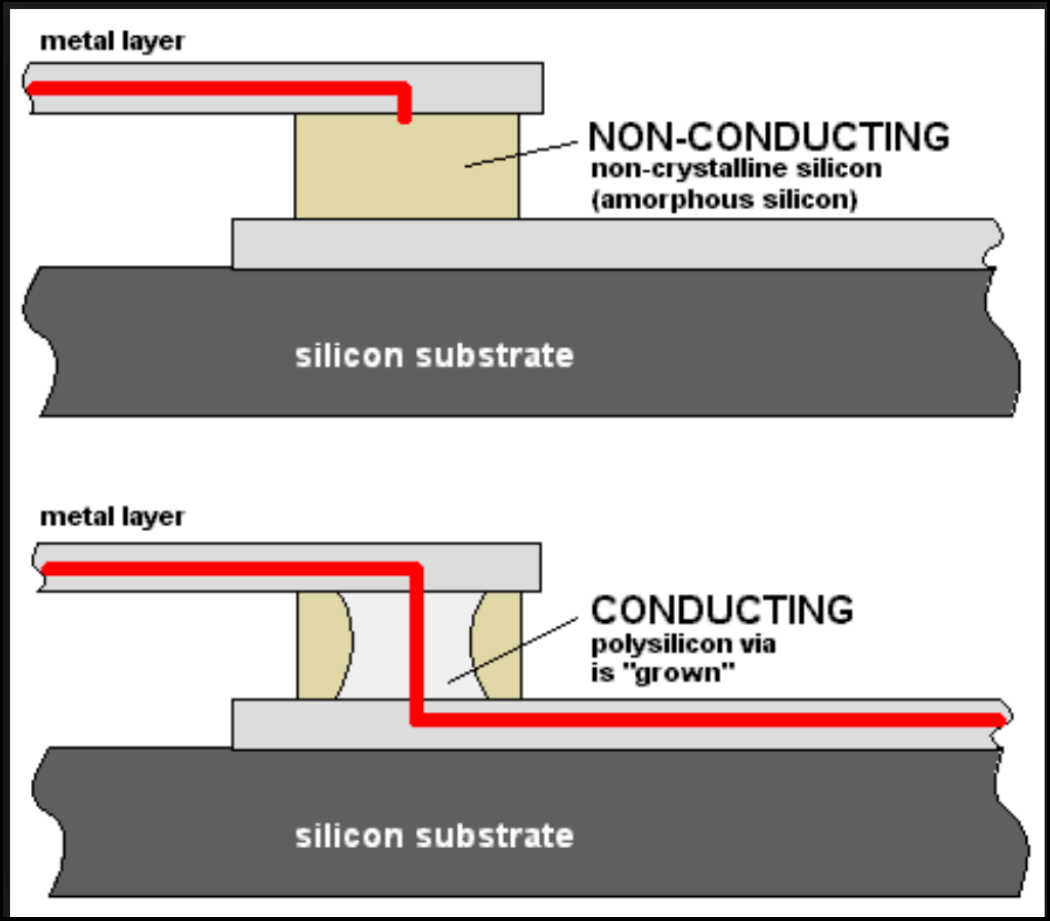
EEPROM


ZTEX USB-FPGA Module 2.04
http://www.ztex.de
XTAL1
LED1
CON2
IC1
IC2
IC3
IC4
IC5
IC6
IC7
IC8
IC9
IC10
JP1
R8
D1
SPARTAN-6
XILINX
TAIWAN
CY7C68013A-
100AXC 1331
B 04 634251
1412
2-7
46V32M16
P
-5B J

Flash


Core3S500E
JTAG
PWR LED
FLAG LED
nCONFIG
RESET
12
13
1
6
8
10
7
9

Antifuse FPGA



But even *some* antifuse FPGA's have readback

Trimberger (https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6849432)

It is important to note that the readback function has, and continues to be, a valuable feature for both the FPGA manufacturer and the user. Whether the manufacturer uses it for device test, or the user employs readback for in-system data integrity checks, it is a feature, much like JTAG, that is useful but needs to be adequately protected to avoid vulnerabilities.

Readback continues to be a concern, and as late as 2012, Skorobogatov and Woods [38] discovered a keyed back-door/test mechanism that enabled the readback feature of a Microsemi antifuse FPGA that was assumed to be protected by the FuseLock protection mechanism [27].

What are people using FPGA’s for?

- Everything

Vehicles without Drivers

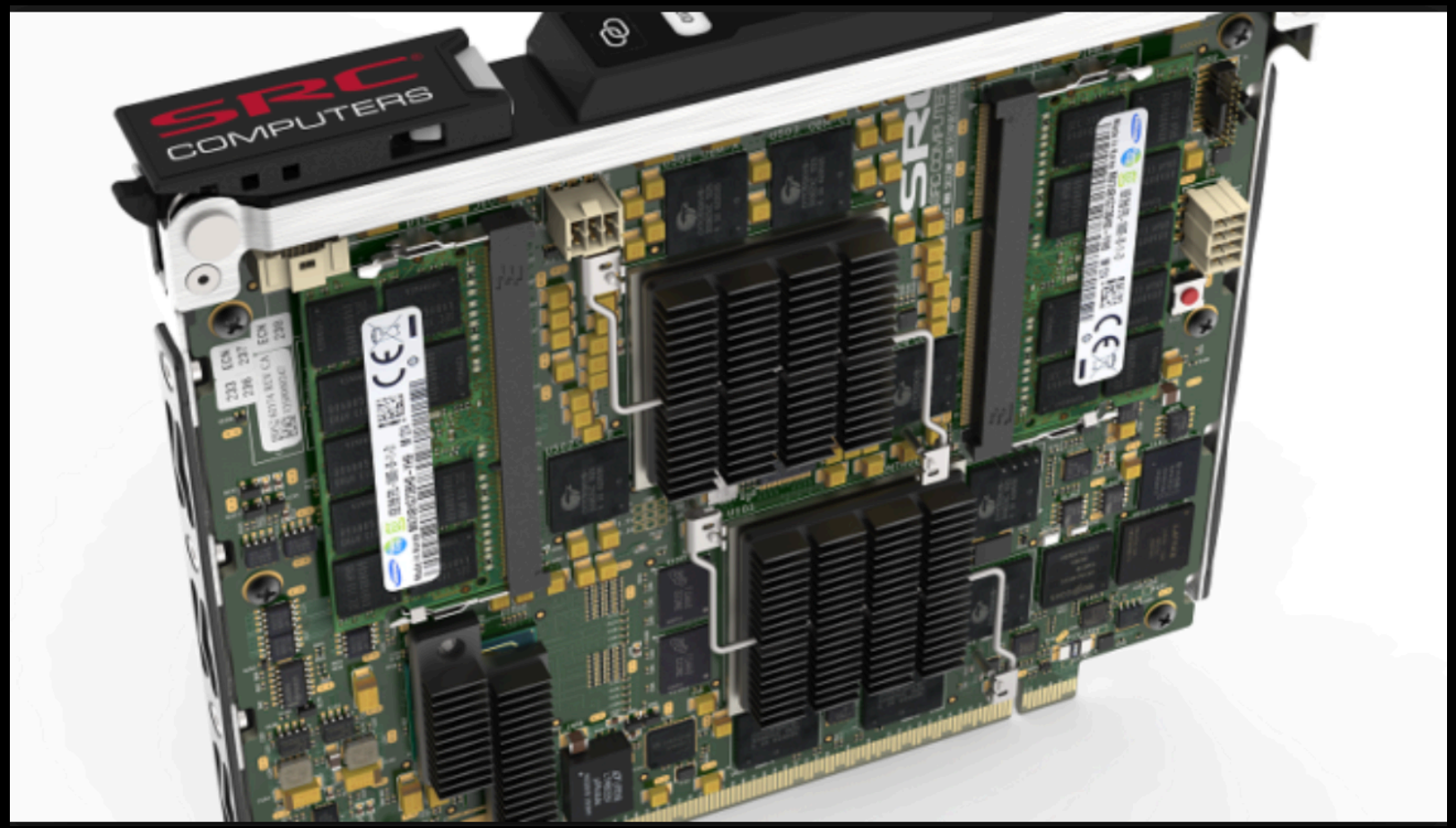
Military Stuff (http://archive.cotsjournalonline.com/articles/view/102372)

Figure 1

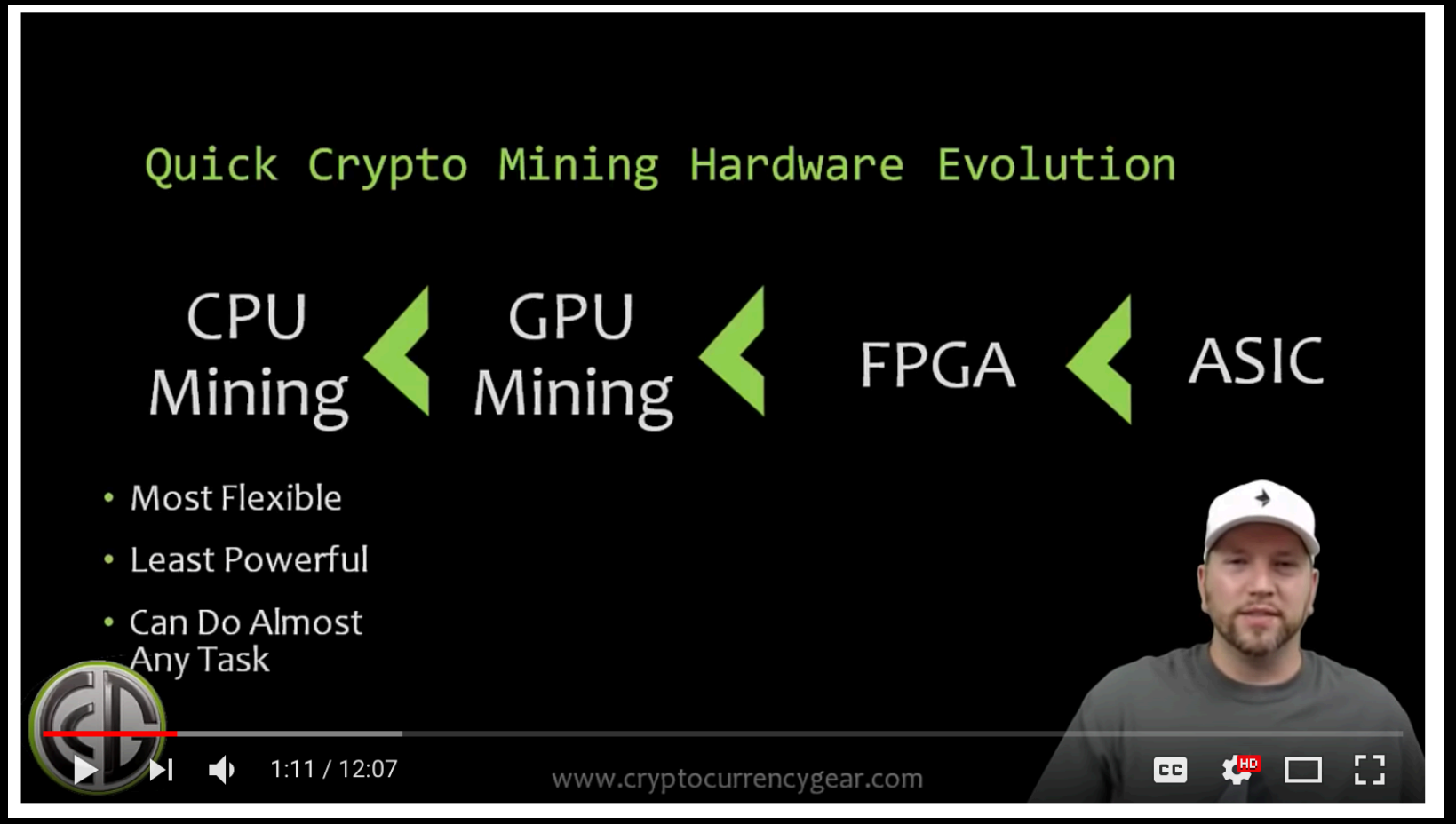
AESA radar systems create highly adaptive steerable beams able to track multiple targets or focus electromagnetic energy in one location. They're used on the F/A-18F Super Hornet to broadcast strong radar signals while remaining undetected.

Routers (http://linuxgizmos.com/openwrt-router-sbc-mixes-cortex-a5-and-fpga/)

Servers



Of Course


Quick Crypto Mining Hardware Evolution
CPU Mining
GPU Mining
FPGA
ASIC
• Most Flexible
• Least Powerful
• Can Do Almost Any Task
1:11 / 12:07
www.cryptocurrencygear.com

General Threat Model

- Attacks against the hardware itself
- Attacks misusing the HDL implementation
- Attacks from outside the electronics environment
- Attacks against the Synthesis Pipeline

Hardware is Hard – the intermediate level is even harder

- FPGA's encourage hardware engineers to recapitulate the sins of the last 40 years of computing
- FPGA's are often placed in situations where the data fed to them is intended to be trusted
- Security Controls aren't always the first thought when the device in intended for performance

Hardware Definition Language

- Allows you to program hardware with code
- Verilog, VHDL and other proprietary variants
- Wires, busses, clocks can be defined dynamically

HDL Language – Verilog CPU

(https://github.com/ejrh/cpu/blob/master/testbench/alu_tb.v)

```
module alu(clk, op, in1, in2, alu_enable, out);

    `include "parameters.vh"

    input wire [2:0] op;
    input wire [WORD_SIZE-1:0] in1, in2;
    input wire clk, alu_enable;
    output reg [WORD_SIZE-1:0] out;

    always @(posedge clk) begin
        if (alu_enable) begin
            case (op)
                `ALU_ADD:
                    out <= in1 + in2;
                `ALU_SUB:
                    out <= in1 - in2;
                `ALU_MUL:
                    out <= in1 * in2;
                `ALU_SLT:
                    out <= in1 < in2;
                `ALU_AND:
                    out <= in1 & in2;
                `ALU_OR:
                    out <= in1 | in2;
                `ALU_XOR:
                    out <= in1 ^ in2;
                `ALU_SHIFT:
                    out <= in1 << in2;
            endcase
        end
    end
```

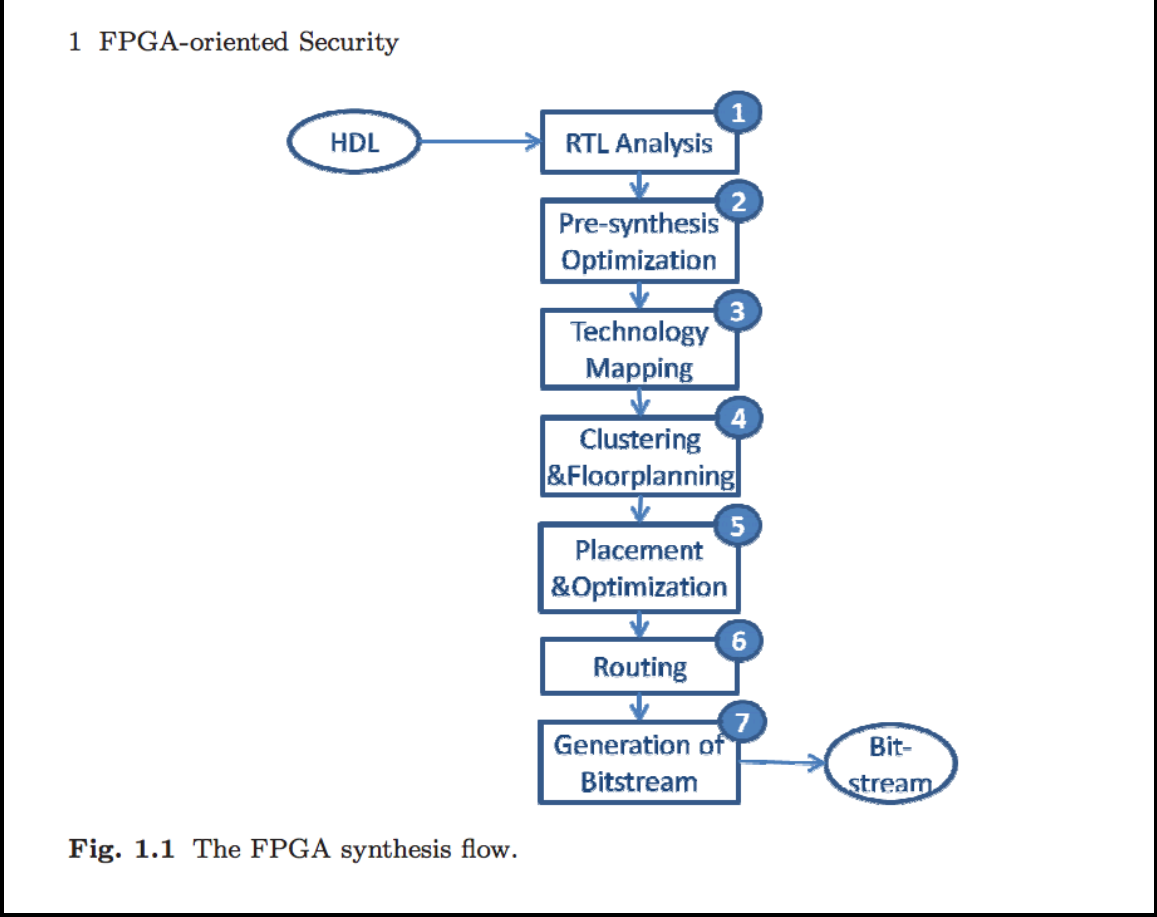
Counting Clock Cycles

(https://www.edaboard.com/showthread.php?45866-How-to-write-code-in-Verilog-for-skipping-two-clock-cycles-at-a-time)

```
module div3 (clk, reset_n, out);
   input clk, reset_n;
   output      out;
   reg [2:0]   count;
   assign      out = count[1];

   always @(posedge clk or negedge reset_n)
     if (!reset_n)
       count <= 0;
     else
       if (count == 2'b10)
         count <= 0;
       else
         count <= count + 1;
endmodule // div3
```

The Synthesis Pipeline(http://web.cs.ucla.edu/~miodrag/papers/Majzoobi_2011.pdf)



Insider Threats May Be an Issue

The common methods for addressing the attacks against stealing of the soft IP cores include watermarking of the soft IP, license agreement, and encryption of the cores that are transferred between parties. Since the soft IP by itself is just a datafile, any other method that is applied to transferring and storage of data files can be used for protecting the transfer and safeguarding of this kind of information. The Trojans/spyware inserted at the HDL level code are either trivial or very hard to detect, based on the availability of designer's information and trust in the designer. It is worth noting that the designer inserted Trojans are very hard to detect in very complex codes, and even the best verification tools may not be able to detect the additional states and functions added by a designer [4, 5]. Often times, the designer does not provide the full specification of the IPs, and therefore, there may not be a basis for comparing the soft IP at hand to a Trojan-free (golden) model. If the designer is trusted, standard cryptographic protocols for integrity checking (e.g., digital signatures) can be applied for ensuring that the original designer's code is not modified. In the final section, we discuss the recent efforts for creation of provably trusted IP.

Physically Uncloneable Functions

- Tie encryption or authentication to device specific factors

Physically Unclonable Functions

tion onto SRAM-based devices. PUFs use the inherent and embedded nano- and micro-scale randomness in silicon device physics to establish and define a secret which is physically tied to the hardware. The randomness is introduced by the existing uncertainty and lack of precise control during the fabrication process that lead to variations in device dimensions, doping, and

More

IC and device on each IC. PUFs typically accepts a set of input challenges and map them to a set of output responses. The mapping is a function of the unique device-dependent characteristics. Therefore, the responses two PUFs on two different chips produce to the same set of inputs are different. A com-

How would you implement that?

- Measuring propagation delays is the typical method
- You can also measure voltages, capacitance and other physical factors
- Learning Parity with Noise
- There are whole books dedicated to getting this right.
- These are not perfect, there are attacks.

These can be defeated by machine learning

Abstract:
Physical Unclonable Function (PUF) has now become a core lightweight hardware-intrinsic cryptographic primitive for device identification and authentication to secure edge computing in Internet of Things (IoT). The main challenge in most delay-based PUF implementations is the rival of response uniqueness and reliability. Due to routing constraint, implementation of delay-based strong PUF on FPGA tends to have either poorer reliability under varying operational conditions or vulnerably high predictability. Therefore, the design of high quality strong PUF often entails tradeoff between reliability and unpredictability (including uniqueness and randomness). Arbiter PUF is one of the most popular structures for FPGA implementation. It suffers from relatively low reliability and high susceptibility to machine learning attacks due to the linearity of its cascaded switch mode delay representation model. To overcome both problems simultaneously, we dichotomize the challenges to winnow out the unreliably weak challenges and obfuscate the remaining reliable strong challenges to increase its unpredictability against machine learning attacks. The security A-PUF is hardened at the expense of small hardware and latency overhead in preprocessing the challenges.

This thesis shows you how (https://github.com/maoswald/cs_masters_thesis)

Machine Learning Attacks on Majority Based Arbiter Physical Unclonable Functions

Master's Thesis

Since secret keys are used as basis for several security features, e.g. encryption or authen- tication, there is a need to protect them from unauthorized access. Physical unclonable functions (PUFs) ought to fulfill this need with the possibility to derive secret keys from their intrinsic hardware imperfections These imperfections are linked inseparable with the device and can be easily used to derive secret keys from but are hard to read out. In addition, there is a high chance that they are unique to every instance. If that is true, they can be used to securely identify devices.

Nevertheless, certain PUFs can be successfully attacked by machine learning attacks. Machine learning attacks in combination with a feasible amount of challenge response pairs make it possible to create models of Arbiter PUFs and XOR Arbiter PUFs, which can be used to predict responses for unknown challenges. Large XOR Arbiter PUFs would prevent these attacks, but suffer instability.

To counteract these instabilities, the principle of majority vote is applied to Arbiter PUFs, which are used in XOR Arbiter PUFs. Its impacts on the stability of Arbiter PUFs and XOR Arbiter PUFs, and relevant attacks are verified by simulations. It is shown that majority vote enables large Majority XOR Arbiter PUFs. Furthermore, this work verifies that large Majority XOR Arbiter PUFs increase the complexity of non-invasive attacks exponentially by linear growth of the PUF. Thus, large Majority XOR Arbiter PUFs constitute a foundation for machine learning attack resistant PUFs. Since they are vulnerable to invasive attacks, they do not protect secret keys completely.

https://zzyue.github.io/PUF/ - Great Further Reading List!

PUF Quick Start

[1] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, "Physical unclonable functions and applications: A tutorial," Proc. IEEE, vol. 102, no. 8, pp. 1126–1141, May 2014.

[2] Potkonjak M, Goudar V. "Public physical unclonable functions[J]". Proc. IEEE, 2014, 102(8): 1142-1156.

[3] Kursawe K, Sadeghi A R, Schellekens D, et al. "Reconfigurable physical unclonable functions-enabling technology for tamper-resistant storage[C]", Hardware-Oriented Security and Trust, 2009. HOST'09. IEEE International Workshop on. IEEE, 2009: 22-29.

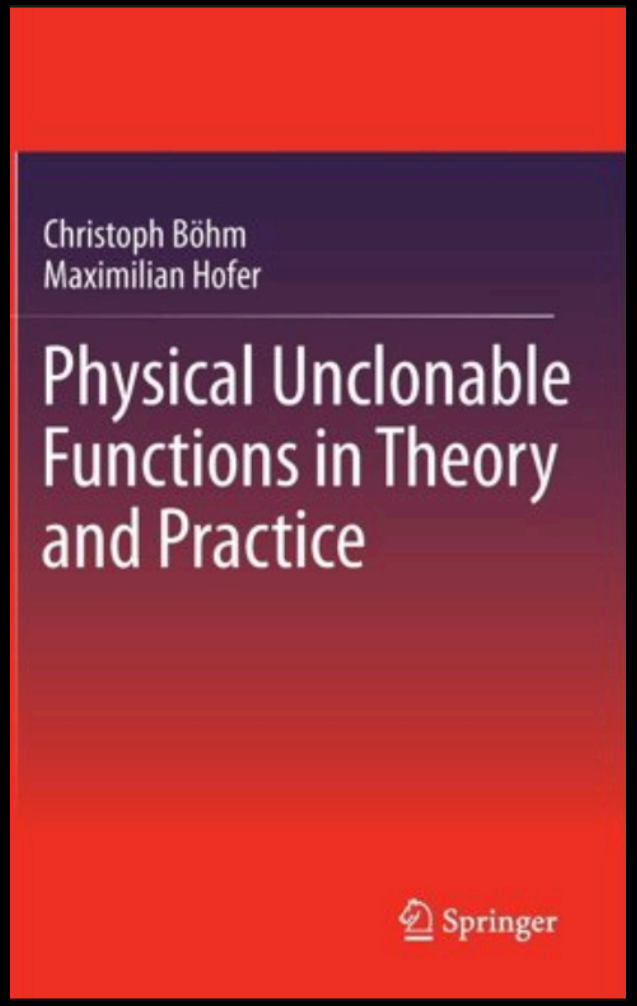
[4] S.S.Zalivaka, L.Zhang, V.P.Klybik, A.A.Ivaniuk,and C.H.Chang, "Design and implementation of high-quality physical unclonable functions for hardware-oriented cryptography," in Secure Syst. Design and Trustable Computing, C. H. Chang and M. Potkonjak, Eds. Springer, 2016, pp. 39–81.

[5] U. Ru¨hrmair and D. E. Holcomb, "PUFs at a glance," in Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, Mar. 2014, pp. 1–6.

[6] Rührmair U, Busch H, Katzenbeisser S. Strong PUFs: models, constructions, and security proofs[M], Towards hardware-intrinsic security. Springer Berlin Heidelberg, 2010: 79-96.

[7] Kursawe K, Sadeghi A R, Schellekens D, et al. Reconfigurable physical unclonable functions-enabling technology for tamper-resistant storage[C]//Hardware-Oriented Security and Trust, 2009. HOST'09. IEEE International Workshop on. IEEE, 2009: 22-29.

Christoph Boehm is the man when it comes to this topic



Threat 1: Disclosure of IP

- Adversaries will attempt to pull your IP off the board
- It has to exist in a form the board will understand and needs to be reloaded on each power on in many situations
- The time when you power on the device is most vulnerable

Methods for Disclosing FPGA IP

- Grabbing IP during configuration off of the device
- Grabbing IP during boot
- Grabbing IP during configuration
- Grabbing IP directly off storage

Bitstream Cloning Attacks

- The easiest thing is to abuse the readback functionality of a FPGA where it exists.
- Usually a bit can be set to prevent read back, but it’s possible to forget to do this.
- Failing this you may have to try and middle the FPGA’s bitstream during boot time. This may be non-trivial.
- Encrypted Bitsreams
- Integrated Devices

The Easy Way

iMPACT

Performing Readback

To Perform Readback

1. Select **Operations** > **Readback**, or right-click on the device icon and select **Readback**.
2. In the Save Readback File dialog box, select a directory and file name.

The programmed contents of the selected device are read back and a JEDEC file is created for CPLDs, an MCS or EXO file for PROMs.

Note The JEDEC, MCS or EXO file can be used to configure other devices. A device that was read protected cannot be read back.

Bitstream Formats

- Typically vary from manufacturer to manufacturer
- May vary device to device
- Almost always proprietary

Majzoobi puts it well

Another potential form of tampering with the bitstream is *reverse-engineering*. The detailed format of the bitstream for a specific FPGA family is typically considered proprietary to the vendor. Even though the bitstream generation or device configuration details are not commonly published and the complexity of the designs often deters a full reversal of the bitstream, the bitstream alone does not provide any provable security. In some sense, vendor specific bitstream generation only provides a level of obscurity, that is not sufficient for providing protection against reverse-engineering. Given enough time and learning algorithms, bitstream reverse engineering is computationally feasible. Therefore, hiding data and information in the bitstream (i.e., security by obscurity) does not yield a strong protection guarantee.

Bitstream Encryption

An application developer prepares a secured FPGA application with the same tools and processes used for any other application. At the end of the design process, when the bitstream is generated, Xilinx proprietary software encrypts the bitstream. The Xilinx software can supply a randomly generated key and initialization vector or the application developer may supply those values. The Xilinx software produces the encrypted bitstream and a key-insertion file.

Some FPGA's use CBC…

1) Tampering With Encrypted Bitstreams: Xilinx FPGAs use 256 b AES encryption [11] in cipher block chaining (CBC) mode of operation [33] to produce a stream cipher. In CBC encryption, each block of data is first XORed with the ciphertext of the previous encryption before being encrypted. In decryption, the decrypted plaintext of each block is XORed with the ciphertext of the previous block (Fig. 4). CBC causes blocks with identical plaintext (for example, all zero) to encrypt to different ciphertext,

Bistream Encryption

- Key generated by synthesis software
- Encrypted at synthesis time
- Key later implanted on the device via jtag

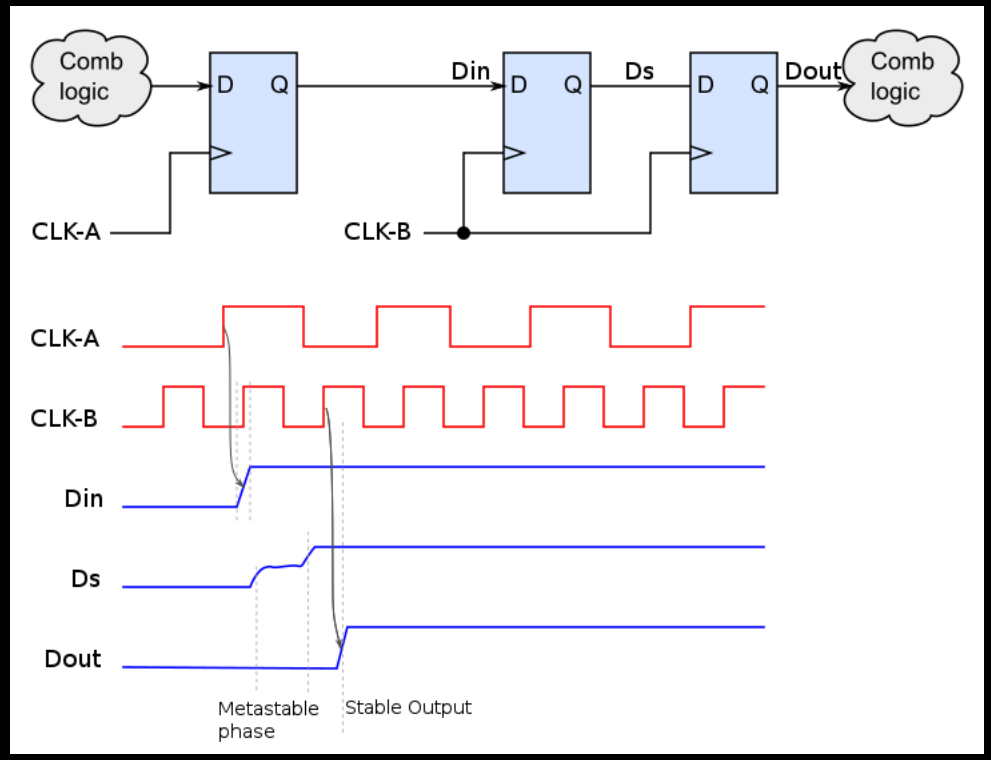
Implementing True Random Numbers In FPGA's

- There are several ways to get true random numbers out of FPGAs
- As a security person it may be your job to prove they are in fact random

FPGA Entropy Sources

- These actually end up being similar to the places you get randomness for PUFs
- Metastability
- Propagation Delay
- Oscillator Jitter
- Oscillator frequency
- Phase Locked Loops
- Dedicated hardware peripherals

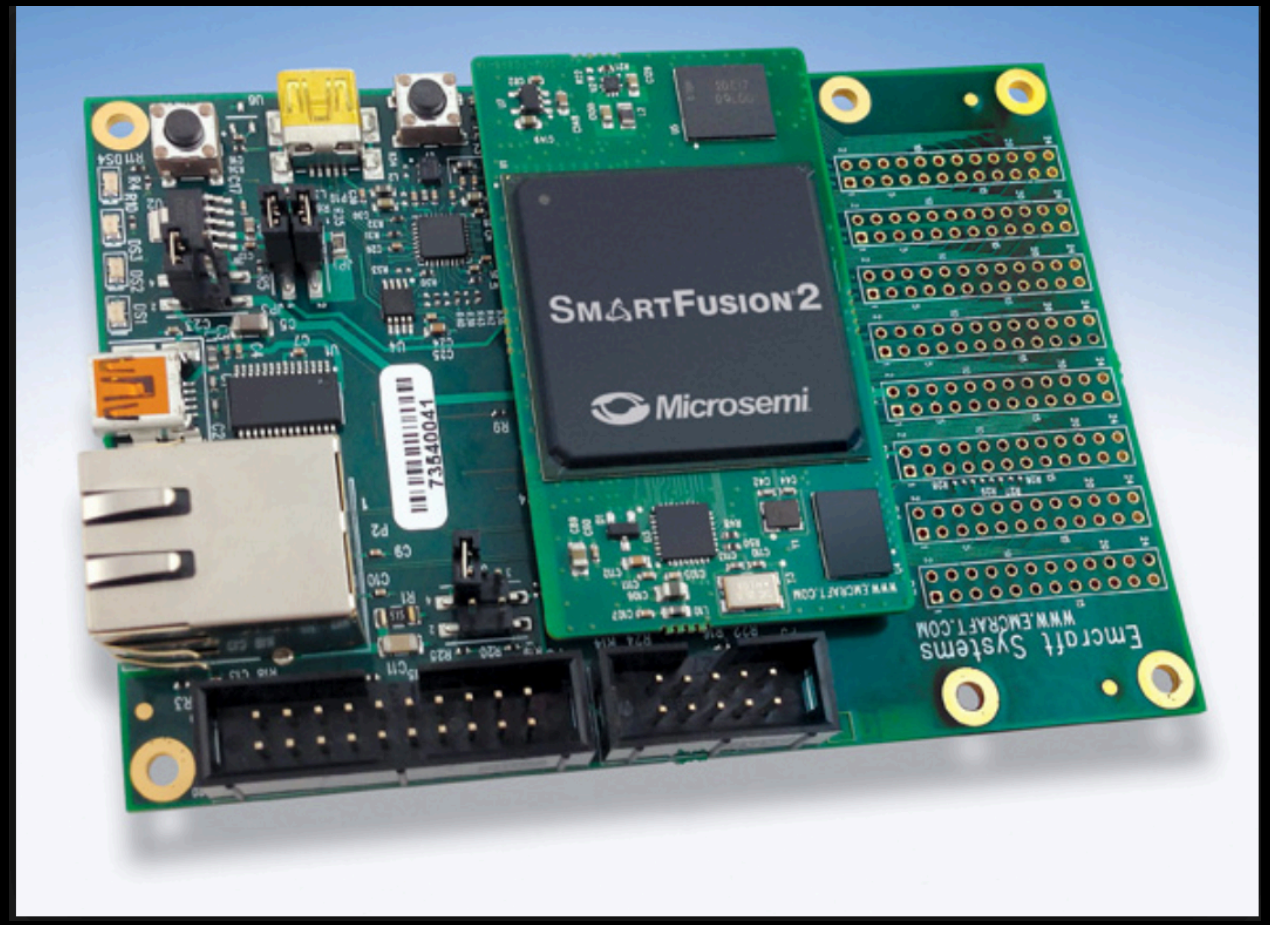
Metastability?



Anti-Tamper Gotchas

- Fuses!
- Tamper resistant Flash memory cells
- Designs which make specific attacks difficult by placement of logic

Microsemi Smart Fusion – Distributed Storage



Microsemi FUSELock (https://www.microsemi.com/document-portal/doc_view/129847-ac168-implementation-of-security-in-actel-antifuse-fpgas-app-note)

Fuse Technology on Microsemi Antifuse FPGAs

Depending upon the architecture selected, Microsemi antifuse FPGAs utilize a number of different fuse elements. Table 1 provides an overview of the different fuses used for each antifuse family.

Table 1 • **Fuse Types of Microsemi Antifuse FPGAs**

Fuse Types for all Microsemi Devices (Except ACT 1 and 40MX)	Fuse Types for ACT 1 and 40MX Devices
Array	Array
Security	Program
	Probe

Described below are the differences between these fuse types:

- Array Fuse: Used to build the metal-to-metal interconnect that creates nonvolatile, low power, high performance paths in Microsemi antifuse architecture
- Security Fuse: Used to prevent unauthorized probing of Microsemi FPGA. Also prevents further programming of the device
- Program Fuse: Prevents additional data from being programmed into the device (ACT 1 and 40MX only)
- Probe Fuse: Used to prevent probing of Microsemi FPGA (ACT 1 and 40MX only)

Microsemi Flashlock (https://www.microsemi.com/document-portal/doc_view/129931-ac185-implementation-of-security-in-microsemi-proasic-and-proasicplus-flash-based-fpgas-application-note)

Types of Security

Microsemi offers two types of security:

- FlashLock®

The FlashLock feature in ProASIC and ProASIC PLUS works through a key mechanism, where the user locks or unlocks the device with a user-defined key. When the device is locked, functions such as device read, write, verify, and erase are disabled. Without the correct key, no one can copy or reverse engineer the design in the FPGA. First, the device must be unlocked using the correct key in order to gain access to the FPGA.

- Permanent FlashLock

The purpose of the permanent lock feature is to provide the highest level of security to the ProASIC PLUS family of devices. The permanent FlashLock feature creates a permanent barrier preventing any access to the contents of the device. This barrier is created by breaking the key after the device is secured. After permanently locking the device, access to the device is not possible even with the proper key. The device is effectively rendered as one-time programmable and therefore is very secure.

Are the keys secure?

Are the Keys Secure

To unlock, the correct key must be loaded through the JTAG programming port. The maximum clock frequency of the JTAG port is 20 MHz. An exhaustive search would take at least

2ks/20 x106 seconds, where ks = key size

Table 3 and Table 4 on page 3 lists how many years are needed to uncover the key for Microsemi Flash devices.

Even using parallel test setups, exhaustive testing of keys would take prohibitively long. Note that care must be taken to use nontrivial keys during key selection.

Table 3 • **Years Needed to Uncover the Key ProASIC Devices**

Device	Years to Uncover the Key
A500K050	57
A500K130	1.57×10^{13}
A500K180	5.27×10^{20}
A500K270	1.77×10^{28}

Xilinx Features

Virtex-6, 7 Series, and Zynq Devices Silicon AT Features	Type	Category
Volatile AES-256 BBRAM Key Storage	Passive(1)	Prevention
Non-volatile AES-256 eFUSE Key Storage	Passive(1)	Prevention
256-bit AES Bitstream Decryption	Passive(2)	Prevention
HMAC SHA-256 Bitstream Authentication	Passive(2)	Prevention
RSA Asymmetric Bitstream Authentication (Zynq devices only)	Passive(1)(2)	Prevention
Hardened Readback Disabling Circuitry	Passive(3)	Prevention
Robust Key Load Finite State Machine (FSM) Circuitry	Passive(3)	Prevention
JTAG Disable	Active	Prevention
JTAG Monitor	Active	Detection
Internal Configuration Memory Integrity	Active	Detection
On-chip Temperature and Voltage Monitor/Alarms (SYSMON/XADC)	Active	Detection
PROG Intercept (PREQ/PACK)	Active	Detection
Unique Identifiers (Device DNA and User eFUSE)	Active	Detection
Internal Configuration Memory Clear (IPROG)	Active	Response
Internal AES-256 BBRAM Key Erase (KEYCLEARB)	Active	Response
Global 3-State (GTS)	Active	Response
Global Set-Reset (GSR)	Active	Response

Notes:

Xilinx - Antireadback

Hardened Readback Disabling Circuitry

Whenever an encrypted bitstream is loaded into the FPGA, readback of the internal configuration memory cannot be performed by any of the external interfaces (including JTAG). All external readback is automatically blocked (disabled) by hardened triple-redundant logic. The only readback access to the configuration memory after an encrypted bitstream load is via the ICAP. Since the bitstream was authenticated during the loading process, the ICAP is considered a trusted channel because it can only be used via a direct connection to the user's design within the FPGA logic. If the user design does not instantiate the ICAP, it cannot be used at all.

> ***Note:*** A BitGen security option via a particular control bit in the bitstream provides a means of enabling/disabling readback. This bit can be changed during configuration. Therefore, readback disable is easy to defeat for devices that are not using an encrypted/authenticated bitstream. Hardened readback disabling has no such weakness and always overrides the security option when using an encrypted/authenticated bitstream.

ICAP (https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6849432)

4) The Value of ICAP and Checking Designs in the Field: ICAP permits logic inside the FPGA to read and write its own bitstream, providing a wide range of powerful use cases. These include:

- internal readback of the device configuration for in-system integrity checks;
- configuration clearing and zeroization;
- algorithm agility for those applications that need to change algorithms without a complete reconfiguration of the device;
- self-test;
- use of user-specific decryption and authentication algorithms with custom protections against attacks such as DPA or other side-channel attacks;
- configuration repair: random single-event upsets (SEU) [23], [52] or intentional tampering may cause configuration bits inside the FPGA to change. Jones [19] describes the SEU controller, an application in which the FPGA logic reads its own bitstream internally through ICAP, checks the stored bitstream with previously computed ECC data, and corrects configuration errors. The SEU controller is intended to detect and correct errors in a high-reliability environment, but it can be used to detect tampering with the FPGA in the field if individual bits are flipped. More recent FPGAs include the SEU detection and scrubbing feature in dedicated hardware [35].

ICAP

- Internal configuration port
- Makes it possible to rewrite the configuration of the FPGA in the field
- Often configured such that the FPGA can rewrite it’s own memory(!)

Threat 2: Damage to Critical Systems

- If a FPGA is connected to a critical system abuse of system logic may be used to cause *physical* damage to that system
- Inappropriate Voltage may be induced to produce a short or an overheating situation

Existing Research on this topic:

http://www.cis.upenn.edu/~jms/papers/fpgavirus.pdf

- *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.

They called it MELT!

destroy system components. We call this class of threat a Malicious Electrical Level Threat (*MELT*).

What should you look for in your own systems

- In much the original research MELT vulnerabilities are created by modifying the bitstream pre-configuration.
- This is because in the majority of cases the compiler wont allow the conflicts that are necessary for MELT to occur on the device itself
- However, that doesn’t protect peripherals connected to the FPGA. Make sure output voltages are constrained in your designs.

Threat 3: Attacks Against Hardware of the FPGA itself

- You might try to reverse engineer it physically
- Focused Ion Beam Measurement ($$$)
- Scanning Electron Microscope
- Analysis with X-Rays
- Thermal analysis

Threat 3A: Attacks against the FPGA which place it in an unexpected environment

- Out of range temperature
- Clock Tampering
- Voltage tampering
- Ionizing Radiation
- FIPS140-2, level 4

Solutions

- “Robust Circuitry”
- Voltage Regulation
- Secure cryptographic implementations
- Redundant storage
- Use a DAC to implement Shutoffs when temperature is wrong
- Physical Isolation of FPGA regions

Xilinx – Isolation Design Flow

Fault Tolerance in Safety Critical Applications

The ability to control system failure modes through fault-tolerant design requires an implementation methodology that ensures fault propagation can be controlled. Xilinx Isolation Design Flow (IDF) provides fault containment at the FPGA module level, enabling single-chip fault tolerance by various techniques including:

- Modular redundancy
- Watchdog alarms
- Segregation by safety level
- Isolation of test logic for safe removal

IDF, pioneered for government cryptographic systems, is also appropriate for avionics, functional safety-related electronics, industrial robotics, critical infrastructure, financial systems, and other high-assurance, high-availability, and high-reliability systems. The IDF is part of a spectrum of reliability technologies that when appropriately combined provide unmatched reliability, performance, and cost effectiveness.

IDF Visualized (Trimberger/Moore https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6849432)

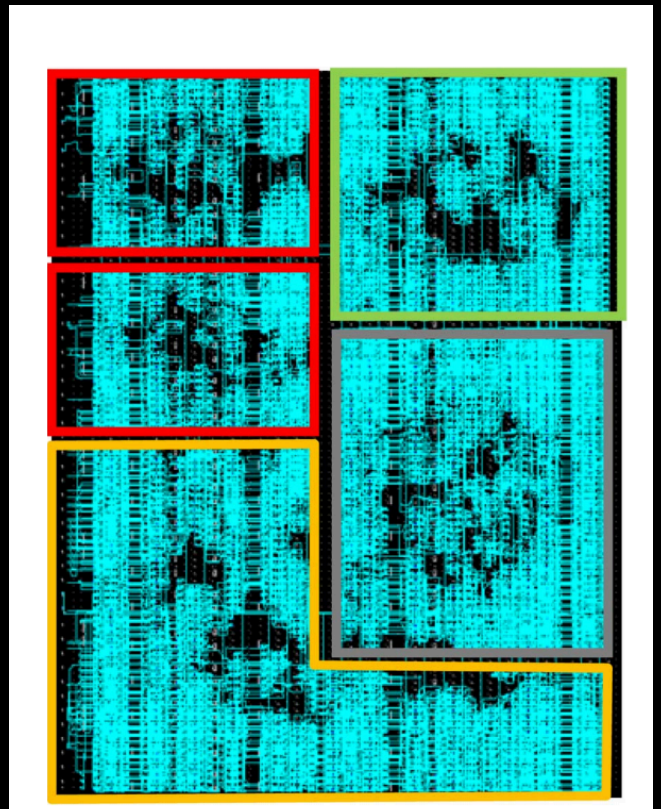


Fig. 9. *FPGA editor view of a design implemented using the IDF methodology.*

“Single Events Upset”

- Radiation causes a bit to flip more or less
- Happens a lot more in outer space.

No Really. Space.

FPGA Design Strategies for the Space Radiation Environment

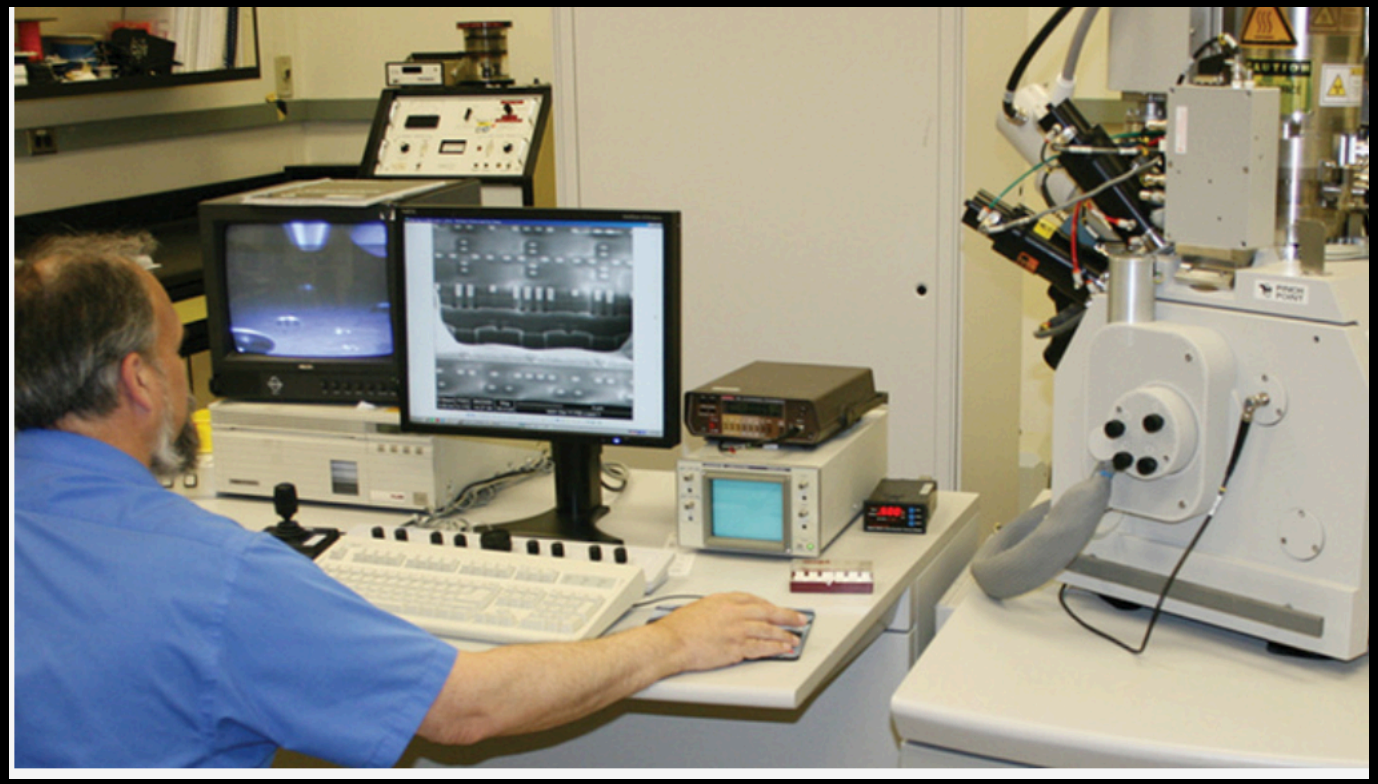
Melanie Berg

Radiation Effects and Analysis Group
NASA Goddard Space Flight Center –
Muniz Engineering and Technologies

High Level Ideas for Radiation Proofing Your FPGA (https://radhome.gsfc.nasa.gov/radhome/papers/seesym06_berg.pdf)

- Good Silicon
- “Ultra Low Alpha” materials
- “Soft Error Mitigation”
- Error Correcting Codes
- Design around it
- Shield
- SRAM

A Guy At BAE Systems Doing Radiation Hardening Testing



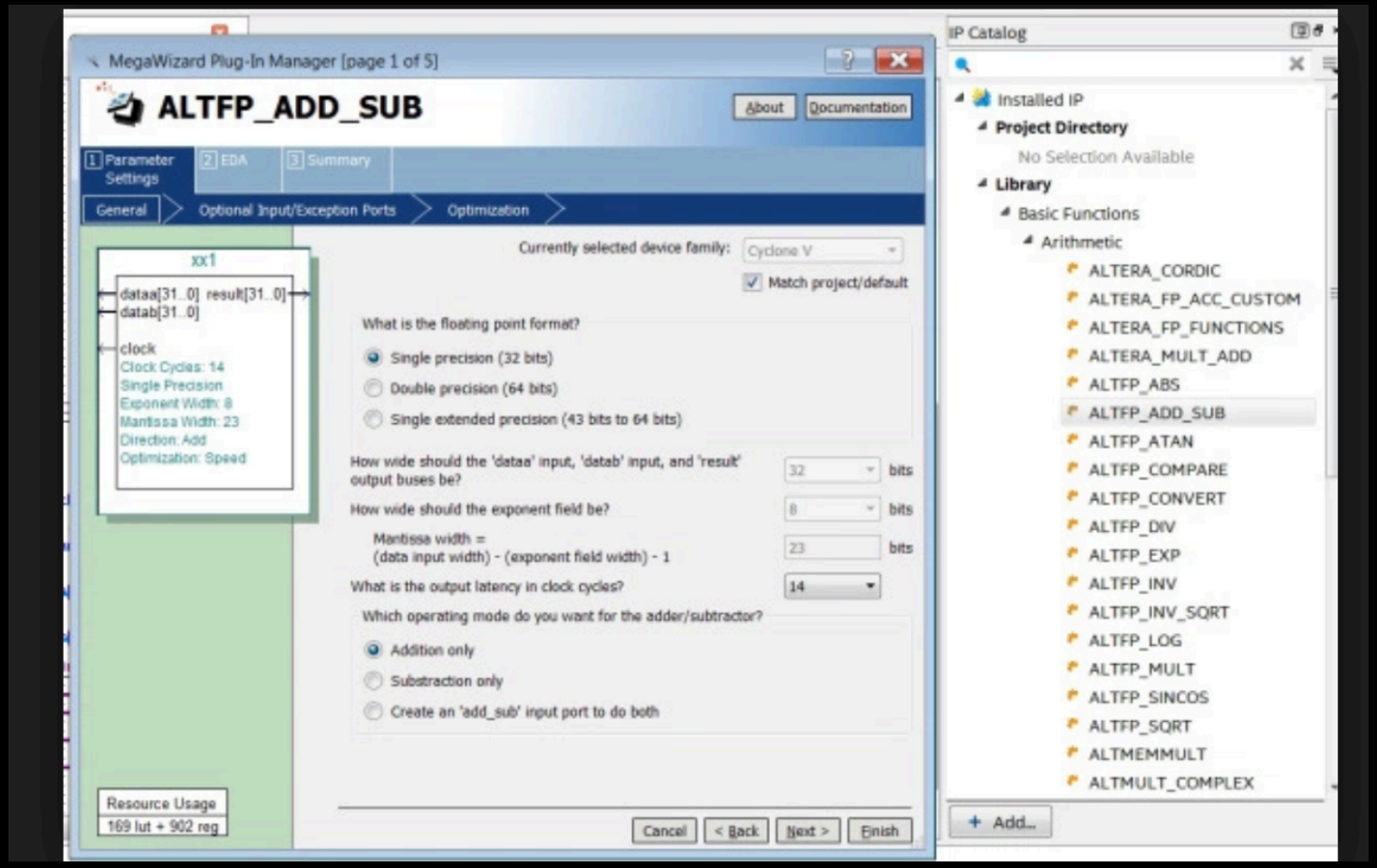
Threat 4: Shared IP based issues

- FPGA’s have libraries too.
- Sort of.

What are IP Blocks?

- HDL for your designs provided by 3rd parties.
- Often sold by the device manufacturer
- Can be as simple as a register and as complex as an entire CPU

The Intel/Altera IP Blocks Menu



Threat 5: Cryptographic Attacks

- Timing based attacks
- Side channel / power analysis
- EMF analysis
- Fault Injection / Glitch Analysis

DPA Attacks

- Practical for recovering keys and decrypting encrypted bitstreams
- May be thwarted by running the circuit in parallel with other power consumers
- Still Effective, active area of research

Timing Analysis

- Timing side channels can be used just like a traditional CPU
- Constant time algorithms work for this just like on a traditional CPU

Fault Injection

- Voltage starvation
- Over Voltage
- EMF field
- Radiation
- Can be counteracted if all states can be accounted for
- Or Tamper detection
- Not too different from traditional CPUs in all this

Epilogue: Security Tools For FPGAs

Static Analysis is Great on FPGA's!

- FPGA's had cool satisfiability solver technology long before the software industry tried it.
- Hardware design was a proving ground for theorem proving based software assurance.
- Sadly, most of it is strictly concerned with electrical engineering properties and doesn't *automatically* extend to more logical properties of the devices.

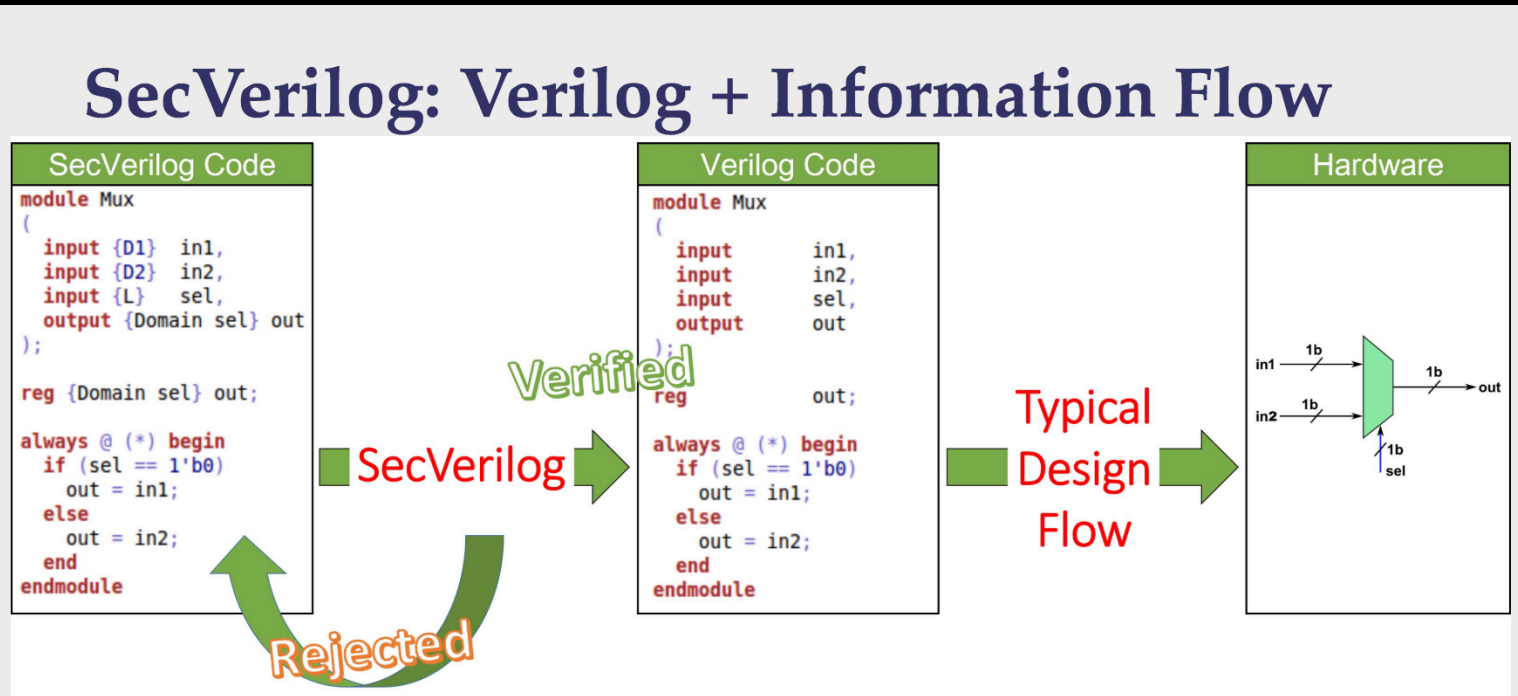
Static Timing Analysis

- All toolsets provide a suite to do this prior to configuring the design.
- Proves soundness in terms of clock and expected propagation of signals through the design.
- Without STA the design might include elements that cause it behave non-deterministicly. Deterministic operations may return differing values.
- Each component in the design induces a delay that must be accounted for.

Issues Finding Security Bugs at the Intermediate Level

- STA doesn’t tell you about logical issues in your design.
- Questions like “can the user write to arbitrary memory?” are not answered by static timing analysis.
- Without your help it cannot harden your crypto against DPA attacks.

This tool from Cornell is awesome



SecVerilog is a new hardware description language (HDL) with fine-grained information flow control. SecVerilog extends Verilog with annotations that support comprehensive, precise reasoning about information flows, including flows via timing, at compile time. The benefit is that hardware designs can be formally verified mostly as-is, with little run-time overhead.

Issues a Human Would Find

- Bad state transitions
- Data flow to unintended areas
- “Timing Sensitive Information Flow”
- Do critical functions check on the state of the clock?
- Race conditions
- Synchronous vs asynchronous
- (logical) Security modes
- Security bit
- Aliasing
- Memory map issues

That’s It!

- Hope you liked it
- I’m @JohnDunlap2 on twitter

Thanks!