POLITECNICO
MILANO 1863

Advanced topics in Hardware Security
-
Introduction

Luca Cassano
luca.cassano@polimi.it
cassano.faculty.polimi.it/ds.html

Acknowledgment

Prof. Mark Tehranipoor

University of Florida (US)

Prof. Marco Ottavi

University of Rome Tor Vergata (IT) and University of Twente (NL)

Course organization

Lecturer

Luca Cassano
luca.cassano@polimi.it

Students meeting:

‘‘Officially’’ on Monday 15:00 – 18:00

@DEIB, 1° Floor, Building 20, Campus Leonardo

But you can always send an email to fix an appointment

Lecturer

Associate Professor @DEIB-POLIMI

MSc in “Ingegneria Informatica” (2009) and PhD in “Ingegneria dell’Informazione” (2013) both from the University of Pisa

Research topics:

- Design, verification, test and diagnosis of electronic circuits and systems
- Hardware Security (Hardware Trojans, Microarchitectural Side-channel attacks, Logic Locking, Fault Attacks)

Students introductions…

Additional lecturer

Prof. Christian Pilato
christian.pilato@polimi.it

Course topics

- Introduction to hardware security
  - The VLSI design cycle
  - Cryptography
  - Fault attacks
- Hardware Trojan Horses
  - Taxonomy and examples
  - Countermeasures
- Side-Channel Attacks and Microarchitectural Side-Channel Attacks
  - Flush+Reload, Spectre, Meltdown
  - Countermeasures
- VLSI counterfeiting and intellectual property stealing
  - Countermeasures

Course calendar

Date			Topic
March	6	Mon	Course introduction – perspective
March	10	Fri	Students presentations
March	13	Mon	Hardware Trojans Horses
March	17	Fri	Students presentations
March	20	Mon	Microarchitectural SCAs
March	24	Fri	Students presentations
March	27	Mon	IP Piracy
March	31	Fri	Students presentations
April	3	Mon	Logic locking
April	6	Thu	Students presentations

Luca Cassano

Christian Pilato

Monday, 9:15 – 12:15, room 3.1.8

Friday, 9:15 – 11:15, room 2.2.3

Thursday, April 6th 9:15 – 11:15, room 25.1.4

Reference Material

These slide

Bhunia, Swarup, and Mark Tehranipoor. *Hardware security: a hands-on learning approach*. Morgan Kaufmann, 2018

Reference scientific papers at the end of these slides

Course assessment

- For PhD students:
  - Attending the lectures and carrying out assignments

- For Master students:
  - A final project is required

Non-functional properties

Dependability

Dependability: the level of trust that a system operates as expected, and will not fail for some unintended reason

Dependability

Quality: the level of trust that a system operates correctly when facing the occurrence of random events at manufacturing

Dependability

Reliability: the level of trust that a system will operate correctly when facing the occurrence of random events during its mission

Dependability

Security: the level of trust that a system will operate correctly when facing the occurrence of malicious events both at manufacturing and mission time

Security properties

Confidentiality

No one except the legitimate user(s) is able to **understand** the content of a piece of data

Integrity

No one can **modify** a piece of data without the legitimate user(s) be able to detect the modification

Non repudiation

A piece of data can **univocally** be associated with its legitimate author

Anti-reply

A piece of data cannot be maliciously **replicated** without the legitimate user(s) be detect it

Anti-DOS (continuity)

A system/functionality/data should **continuously** be available for its legitimate user(s)

The role of hardware in security

Hardware Security Problem

Cybersecurity experts have traditionally assumed that the hardware underlying information systems is secure and trusted.

However such assumption is no longer true.

Hardware cannot be considered the root of trust

Hardware Trust and Hardware Vulnerability

Two main problems with the root of trust assumption

- **Hardware Trust**: are we sure that the hardware has been made by trusted sources?
  - Has the intellectual property been protected?
  - Has the right number hardware pieces been produced?
  - Are the hardware pieces we received the expected ones?
  - Are there any unwanted malicious modifications (Trojans)?

Hardware Trust and Hardware Vulnerability

Two main problems with the root of trust assumption

- **Hardware Vulnerability**: are there any vulnerabilities that could be exploited by an attacker?
  - Fault attacks
  - Side Channel Attack attacks
  - Transient execution attacks (example Spectre, Meltdown)

Some attack examples

In 1962 the Xerox 914 copy machine, the world's first, was used in soviet embassies all over the world.

The machine was so complex that the CIA used a tiny camera designed by Zoppoth to capture documents copied on the machine by the soviets.

Pictures were then retrieved by a "Xerox repairman" right under the eyes of soviet security.

Roy Zoppoth stands over a Xerox 914

Some attack examples

The **Outside the Box** Israeli Air Force operation:

- Eight fighter planes from IAF attacked and destroyed a nuclear plant (under construction) in Deir ez-Zor, Syria, in 2007
- All Syrian radars and air defense missile systems switched-off simultaneously
- Syrian defense equipments featured COTS components (probably manufactured by Intel Israel)
- All these information have been confirmed by IAF in 2018

Some attack examples

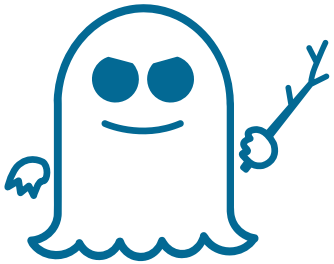
Fake Cisco routers risk "IT subversion”

- An internal Federal Bureau of Investigation presentation states that counterfeit Cisco routers imported from China may cause unexpected failures in American networks. The equipment could also leave secure systems open to attack through hidden backdoors.

Some attack examples

Spectre *poisons* the branch prediction and the speculative execution to force the processor to execute instruction sequences that should not be executed

Then, by exploiting **timing measurements** on the cache accesses, the attacker can retrieve a secret from the cache of the attacked program **without having physical access to it**



Some attack examples

Meltdown exploits out-of-order execution to break the isolation between the memory spaces of user applications and of the operating system

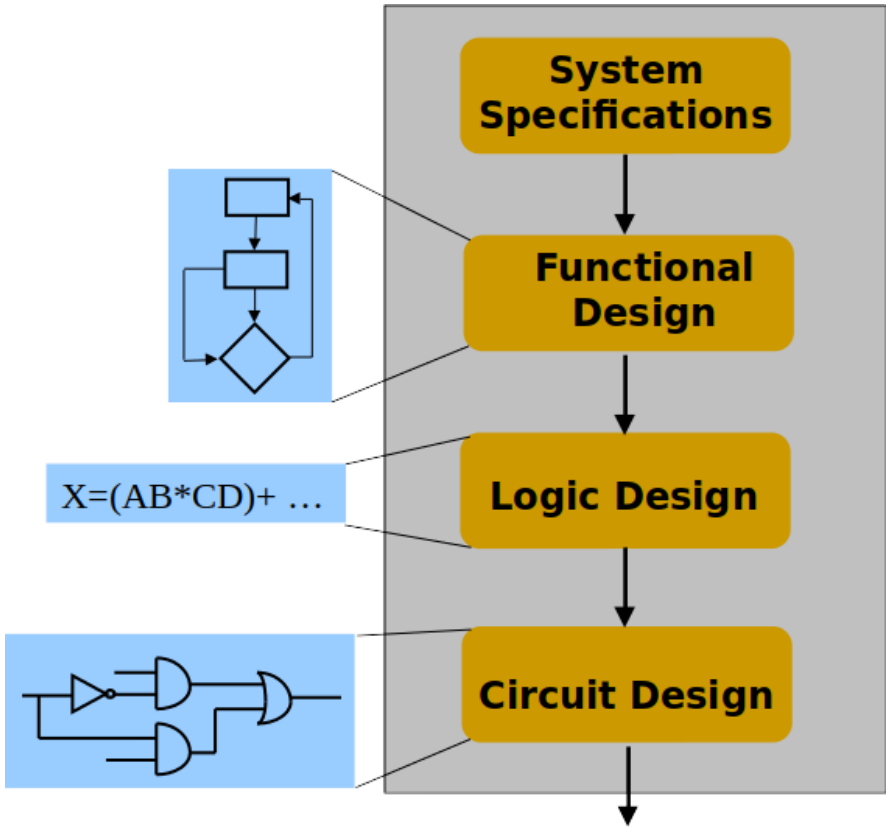
It allows the attacker program to **access any memory space**, thus, stealing secrets from the operating system or other users applications

The lifecycle of modern integrated circuits

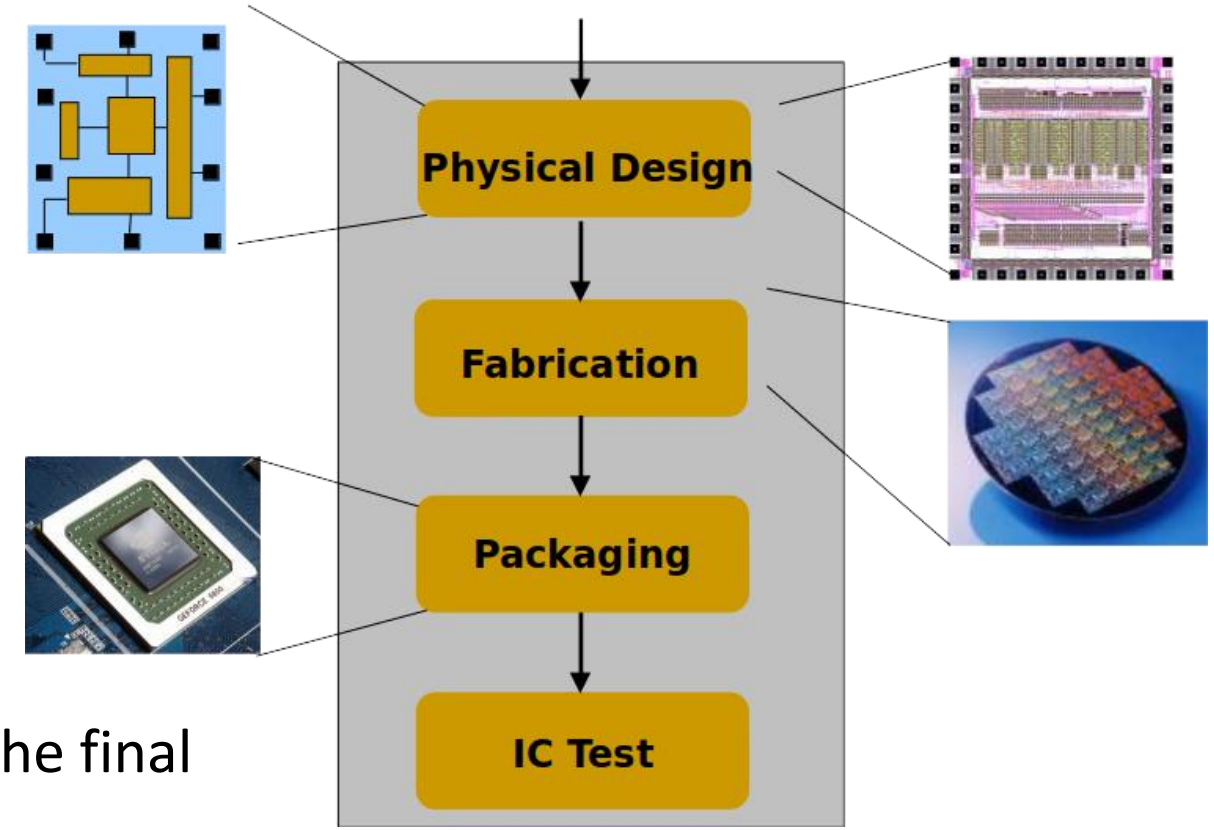
VLSI Design Cycle

Each level is characterized by a specific representation of the design.

- Functional level → flow charts
- Logic Design → Boolean logic
- Circuit Design → logic gates



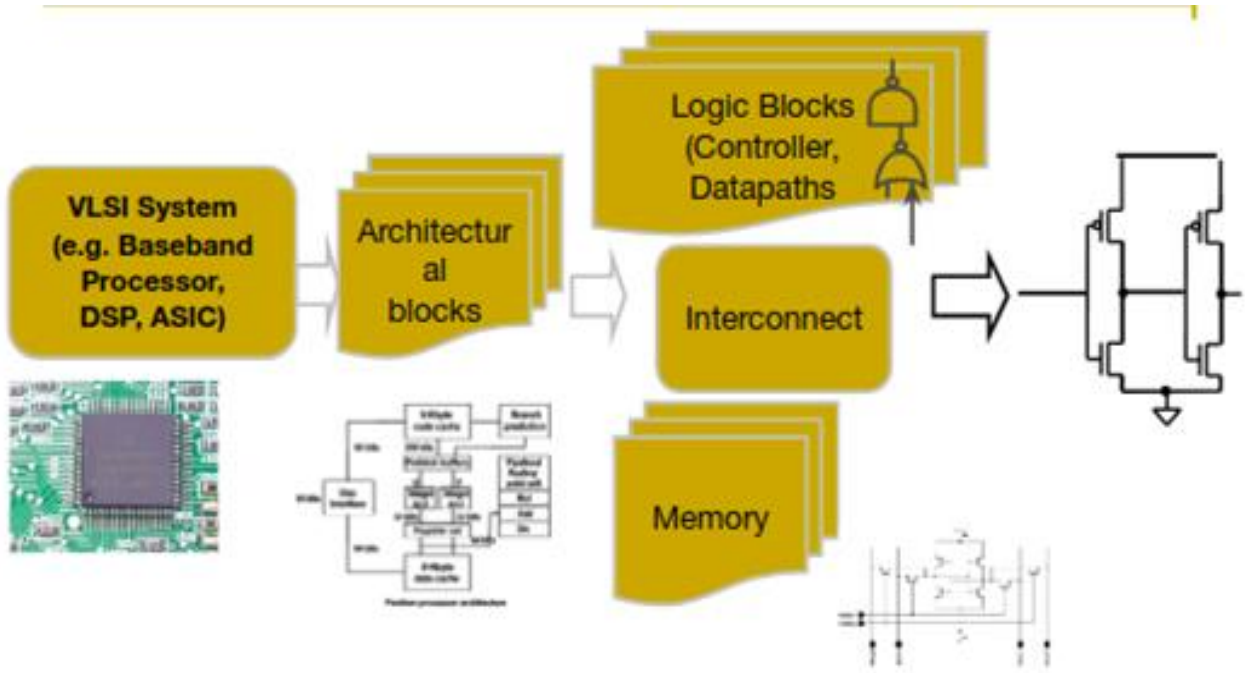
VLSI Design Cycle



At manufacturing level the final representations are

- Physical design → layout and standard cells
- Fabrication → masks for lithography

VLSI Design Cycle

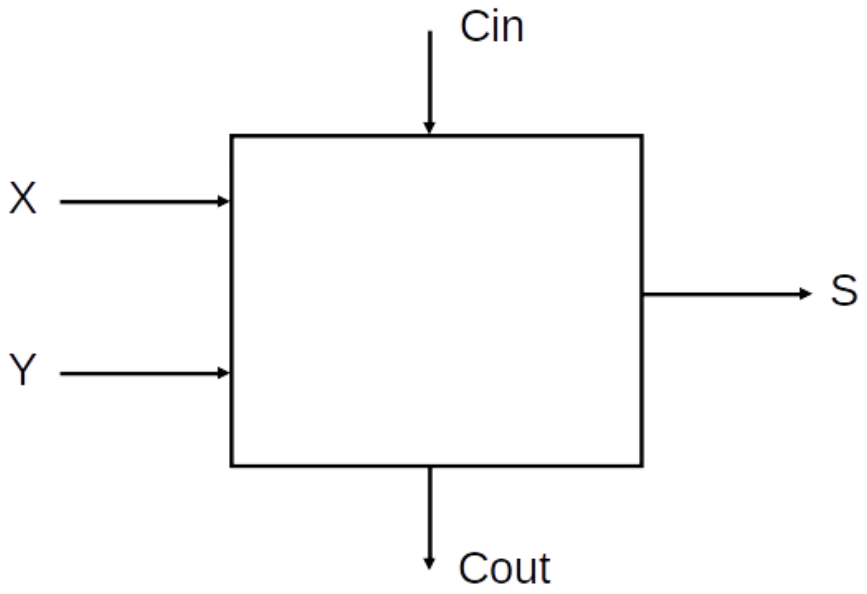


System -> Architecture -> Logic ->Transistor

VLSI Design Cycle

Full Adder, Black Box view

- Inputs X,Y, C_{in}
- Outputs S, C_{out}



VLSI Design Cycle

Full Adder, Behavioral view

- VHDL

```
entity adder is
        -- i0, i1 and the carry-in ci are inputs of the adder.
        -- s is the sum output, co is the carry-out.
        port (i0, i1 : in bit; ci : in bit; s : out bit; co : out bit);
  end adder;
  architecture rtl of adder is
        begin -- This full-adder architecture contains two concurrent assignment.
         -- Compute the sum. s <= i0 xor i1 xor ci;
        -- Compute the carry. co <= (i0 and i1) or (i0 and ci) or (i1 and ci);
end rtl;
```

VLSI Design Cycle

Full Adder, Behavioral view

- Verilog

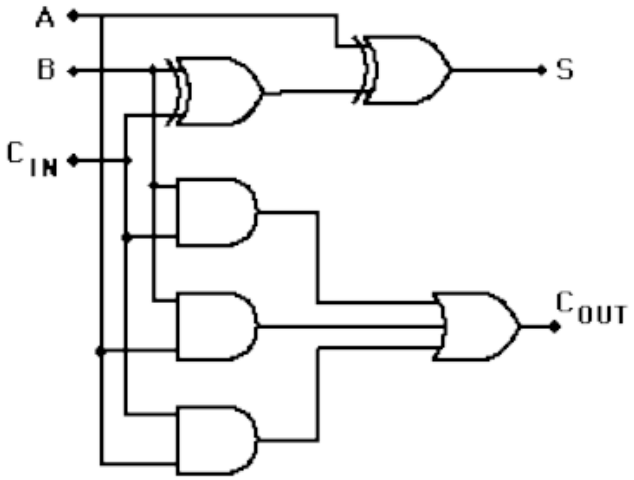
```
module fulladder (a,b,cin,sum,cout);
   input a,b,cin;
   output sum,cout;

   reg sum,cout;
   always @ (a or b or cin)
   begin
     sum <= a ^ b ^ cin;
     cout <= (a & b) | (a & cin) | (b & cin);
   end
endmodule
```

VLSI Design Cycle

Full Adder, Logic view

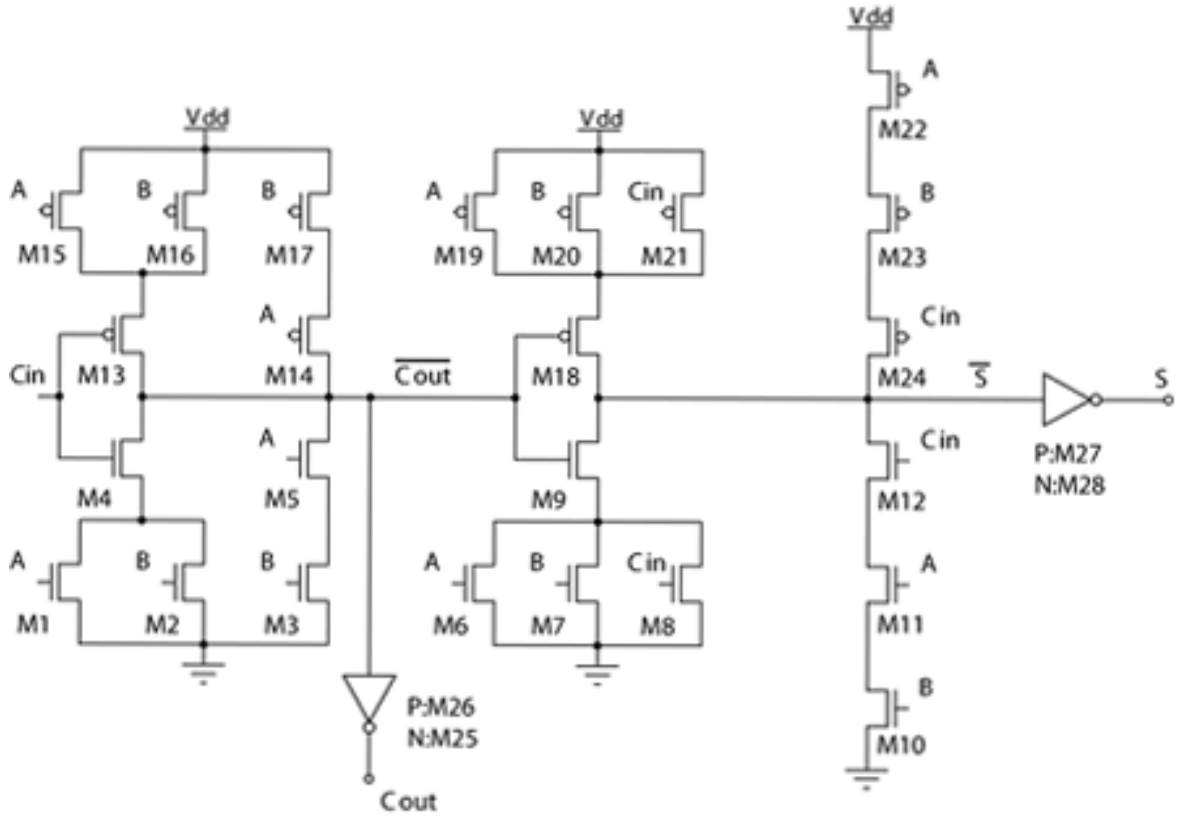
Full Adder Truth Table

CARRY IN	*input B*	*input A*	*CARRY OUT*	*SUM digit*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



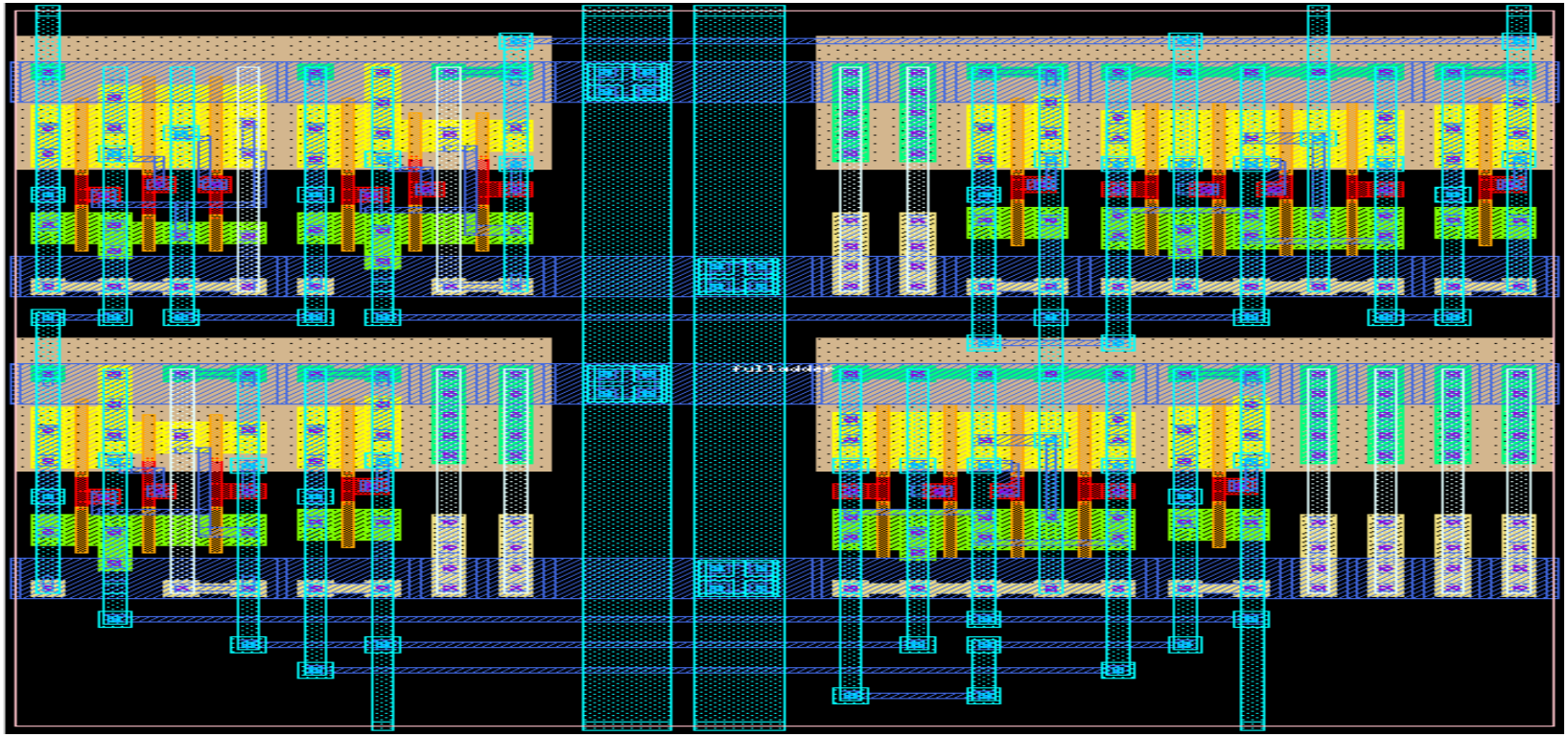
VLSI Design Cycle

Full Adder, Circuit view

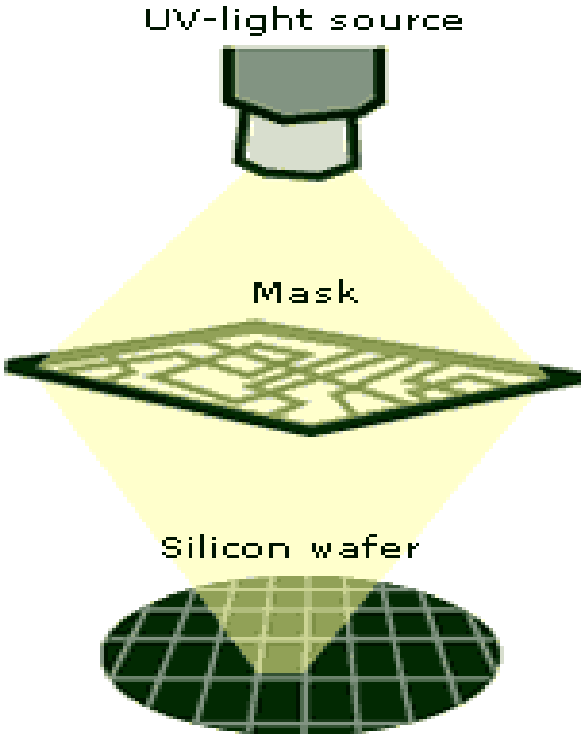


VLSI Design Cycle

Full Adder, Layout view



VLSI Design Cycle



How do we make a transistor?

How do you control where the features get placed?

- Photolithography masks on several layers
- Iterative process with several masks

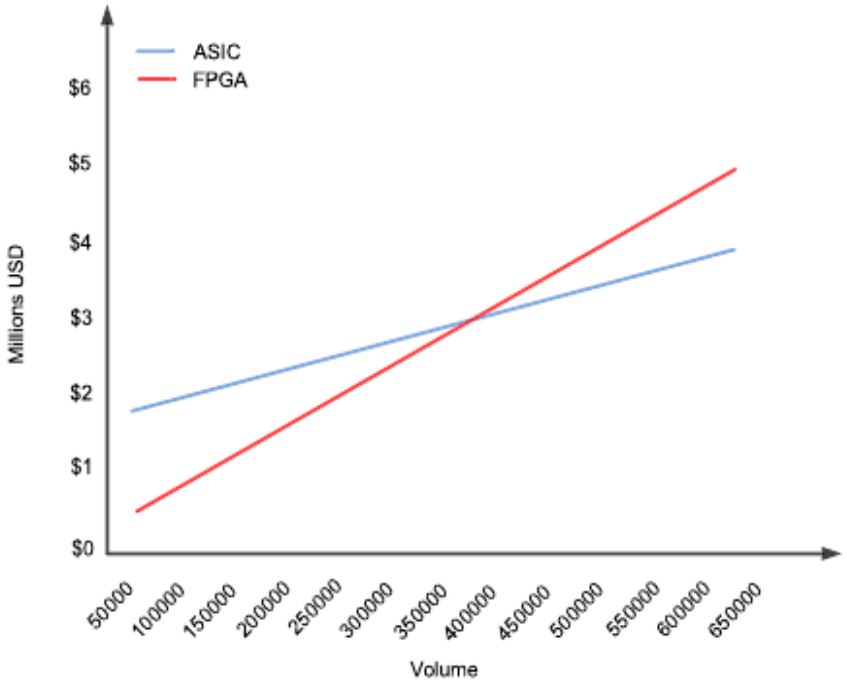
VLSI Design Methodologies

- Full custom (ASIC)
  - Design for performance-critical cells
  - Very expensive
- Standard cell (ASIC)
  - Faster
  - Performance is not as good as full custom
- Gate array
  - Field Programmable Gate Array
  - Lower performances
  - Design errors can be fixed through reprogramming
  - Higher cost on sngle chip with respect to ASICs
  - Good for emulation and for low volume production

VLSI Design Methodologies

	Full Custom	Standard Cell	Gate Array	FPGA
Area	Compact	Moderate	Moderate	Large
Performance	High	Moderate	Moderate	Low
Production Volume:	**Mass Production Volume**	**Medium Production Volume**	**Medium Production Volume**	**Low Production Volume**
Complexity:	**High**			**Low**

VLSI Design Methodologies



ASICs:

- high costs for design and development
- low manufacturing cost of single device when produced in high volumes

FPGAs:

- low cost for design and development
- high costs of individual devices → does not scale with volume

Where hardware security issues come from?

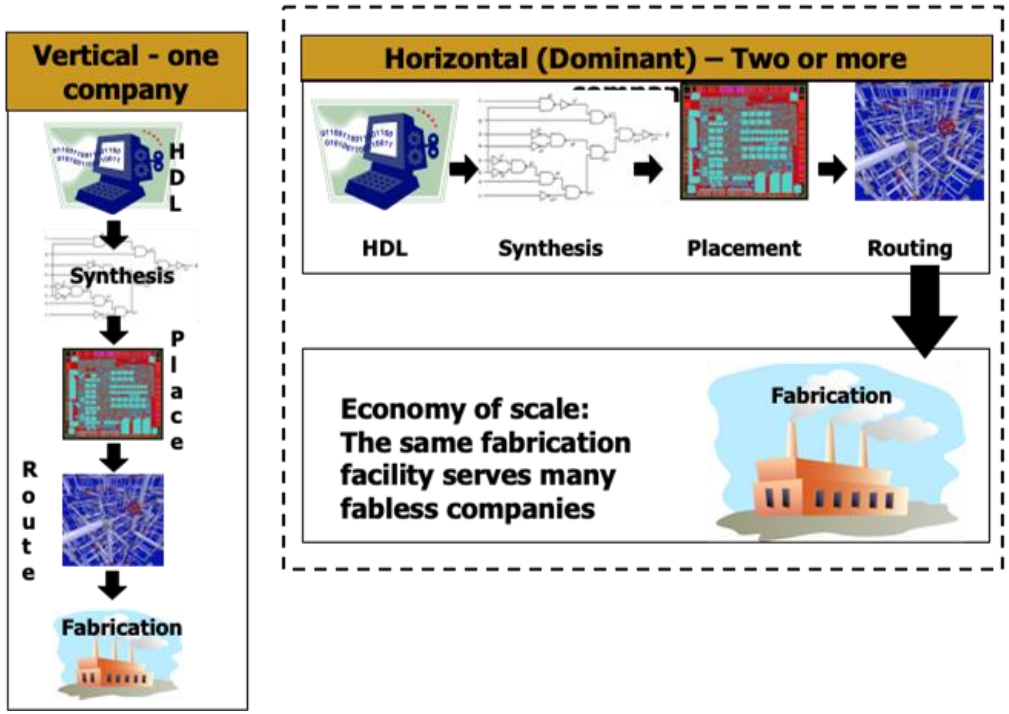
VLSI Industry: Business Model trends

Vertical Model:

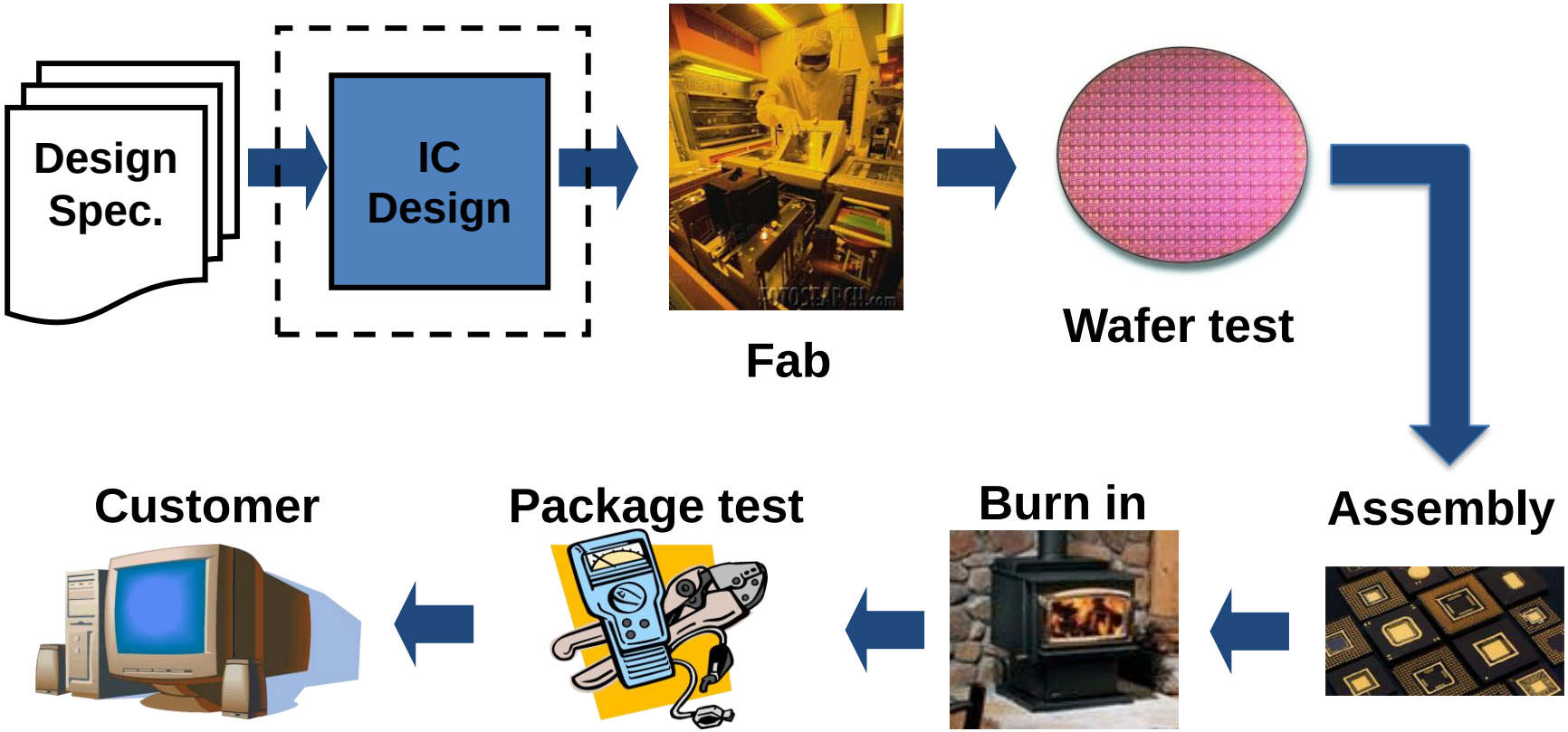
all in-house development, high costs, low economy of scale

Horizontal Model:

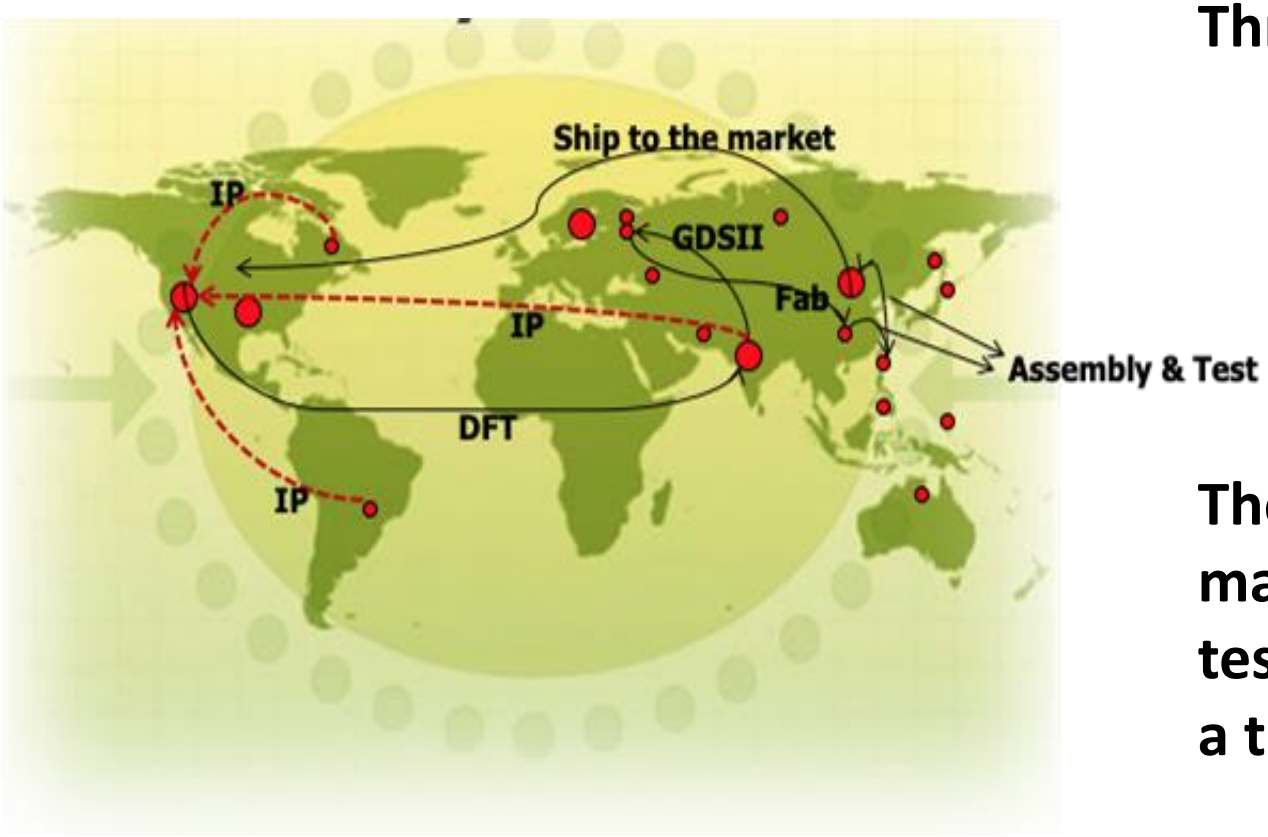
several companies involved, lower costs, economy of scale



IC Design and Test Flow



Where are modern chips developed?

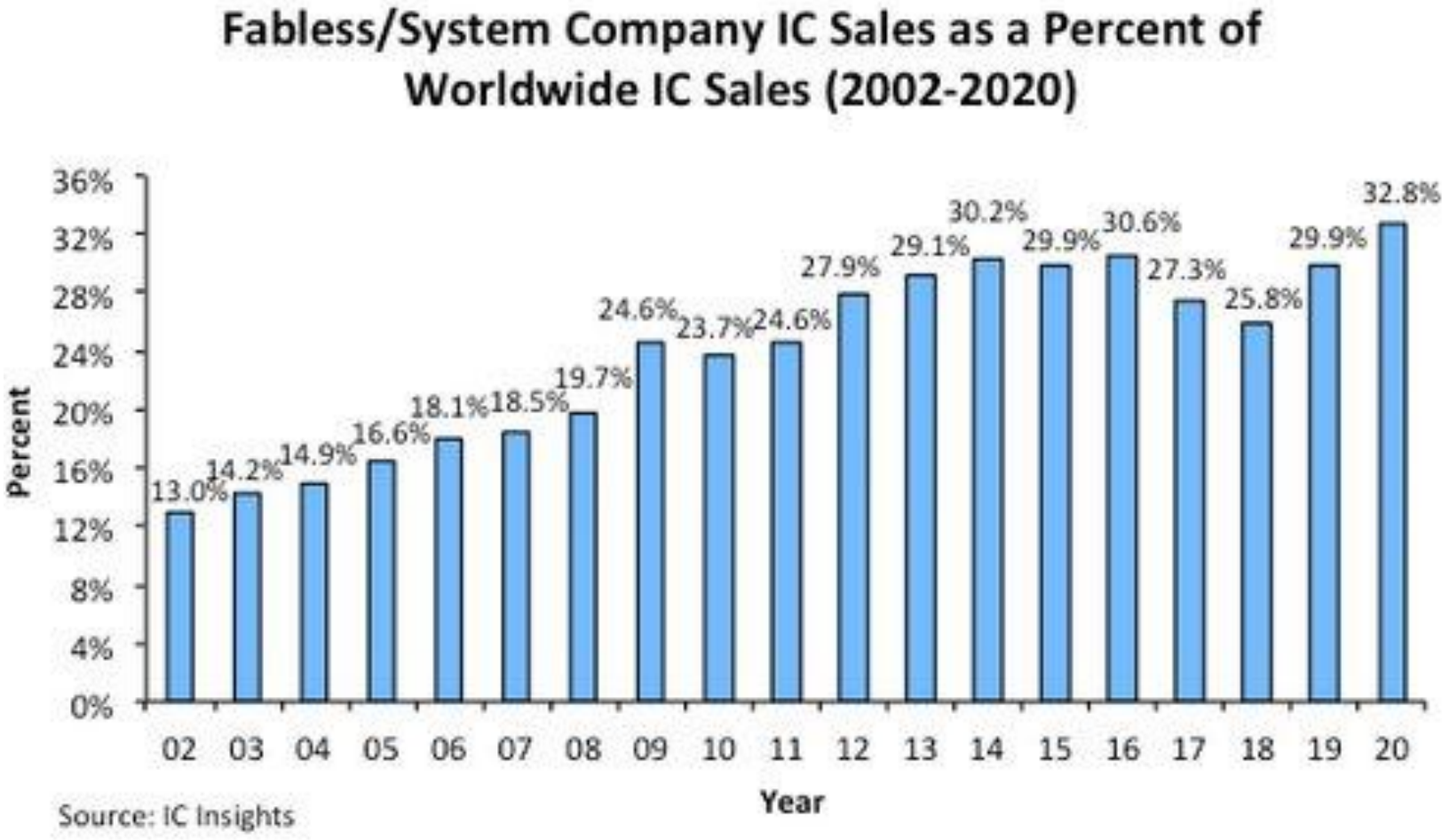


Throughout the globe

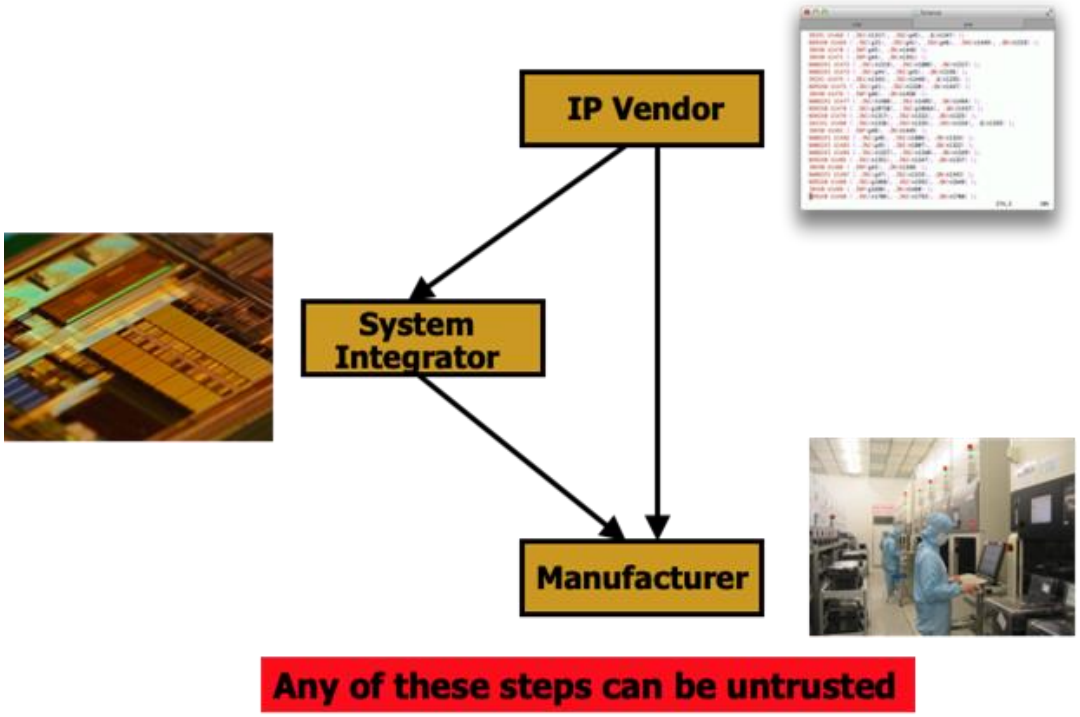
The phases of design, manufacturing, testing, packaging are a truly global activity

This raises potential trust issues

Fabless industry up to 33%



Can we Trust Hardware?

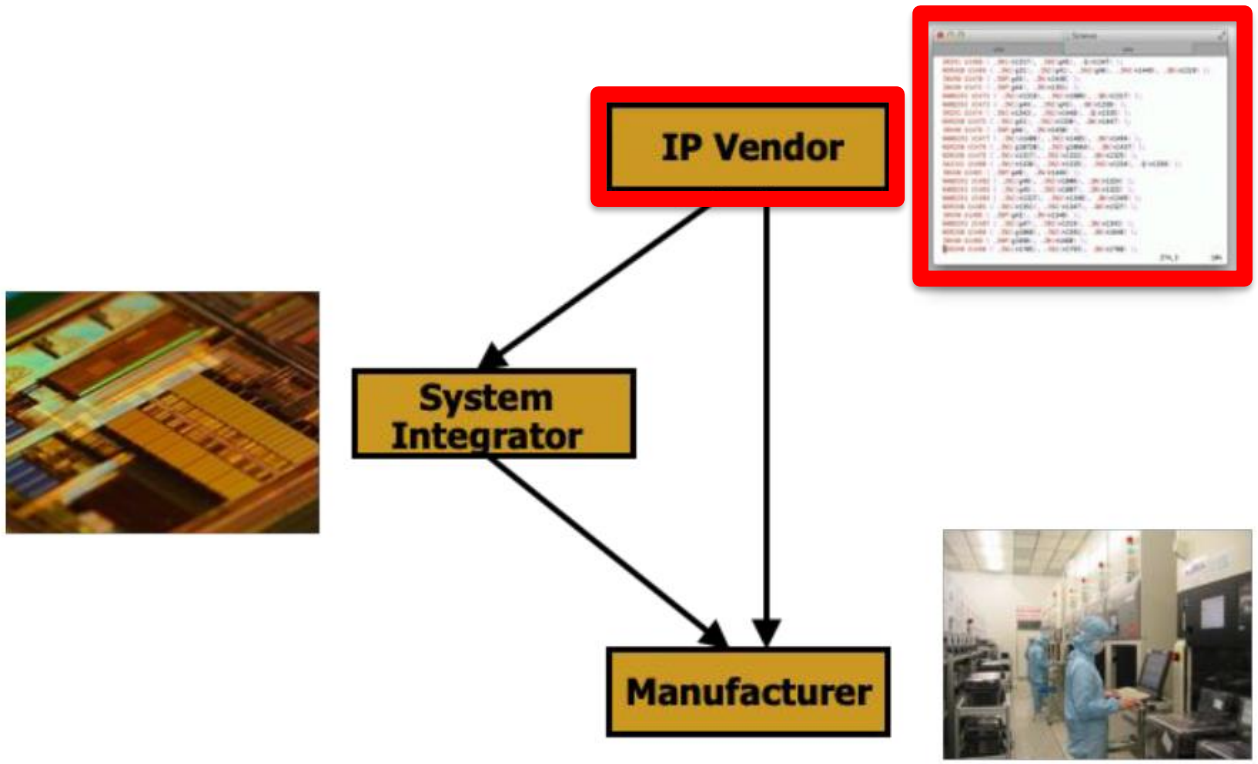


IP: Intellectual properties sometimes **provided by third party vendors**

System Integrator **combines several IPs** into a chip design

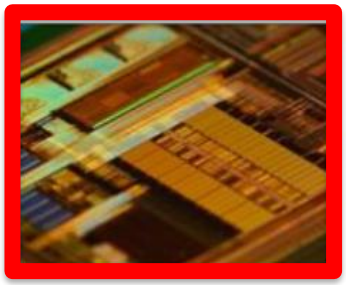
Manufacturer fabricates the chips **based on the received design**

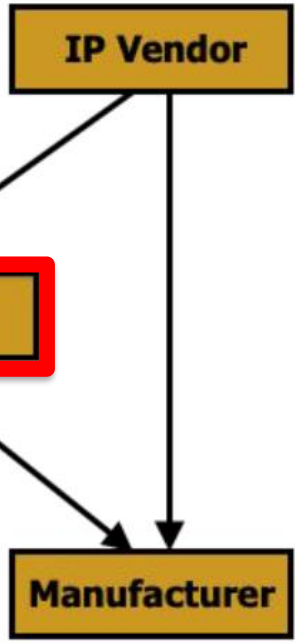
Can we Trust Hardware?

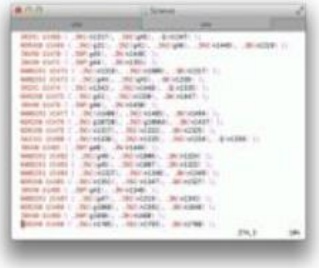


- HW Trojan Horses

Can we Trust Hardware?

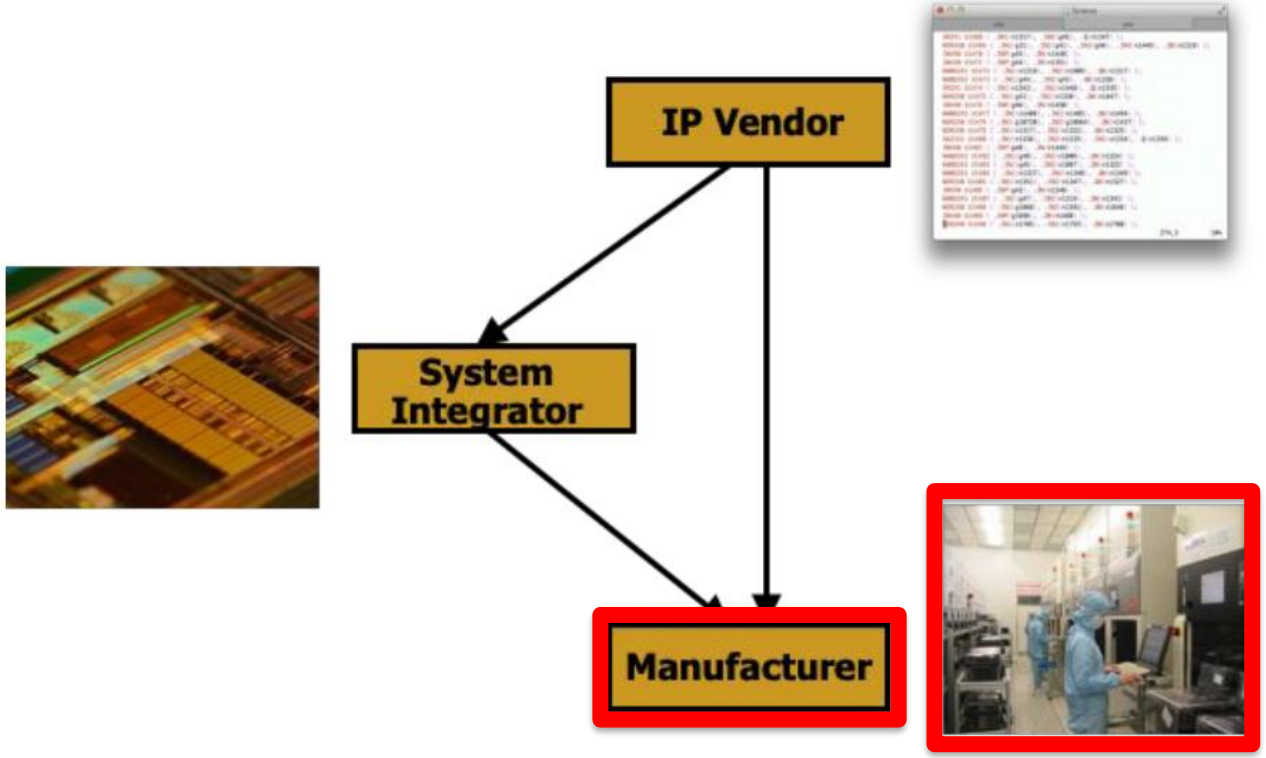






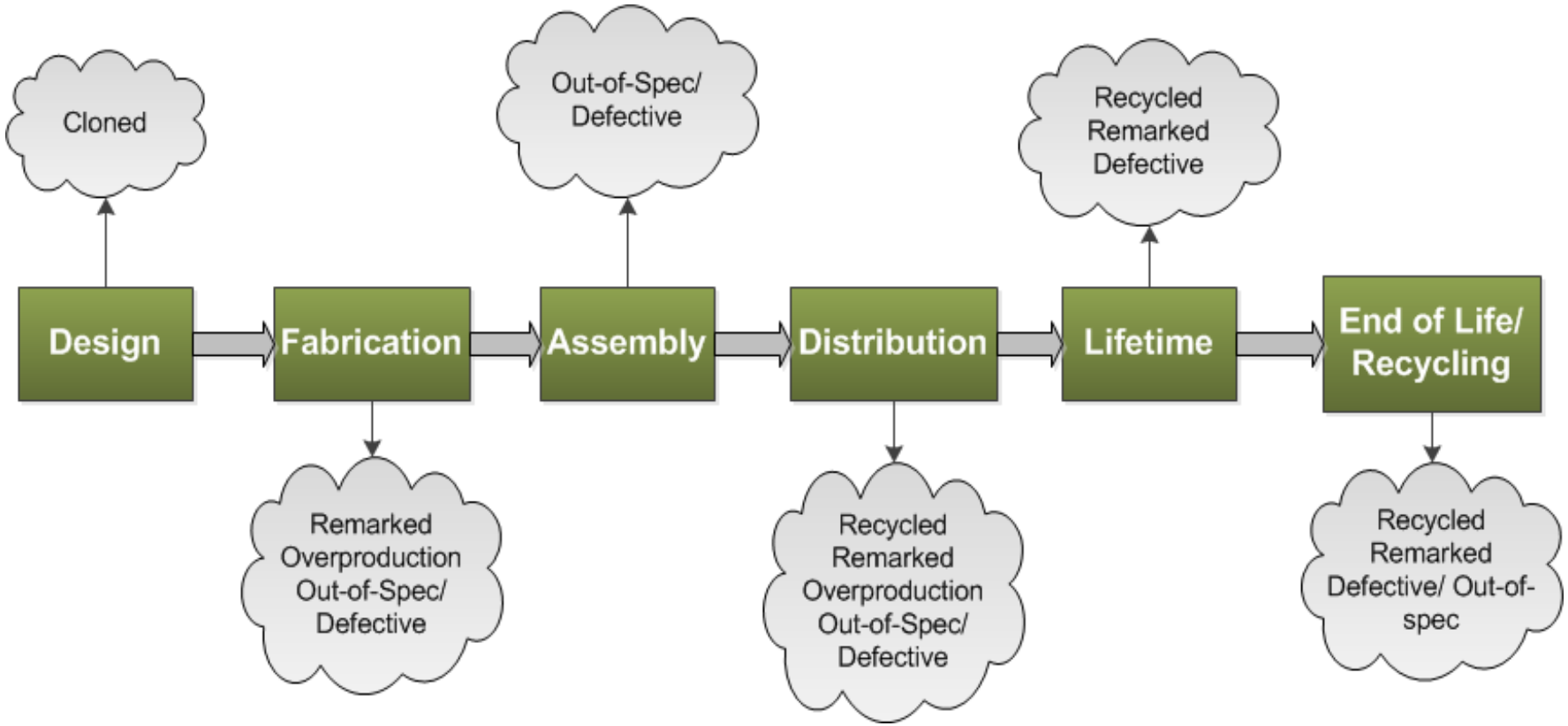
- HW Trojan Horses
- IP theft
- Malicious CAD tools

Can we Trust Hardware?



- HW Trojan Horses
- IP theft
- Off spec. and Defective ICs
- Overproduced ICs
- Recycled ICs

Can we Trust Hardware?



Security principles

What Does Secure Mean?

It has to do with an asset that has some value – think of what can be an asset

There is no static definition for “secure”

Depends on what is that you are protecting your asset from

Typically, breach of one security makes the protection agent aware of its shortcoming

There is no security by obscurity

Typical Cycle in Securing a System

1. Identify the attacker
2. Predict potential breaches and vulnerabilities
3. Consider possible countermeasures, or controls
4. Either actively pursue identifying a new breach, or wait for a breach to happen
5. Identify the breach and work out a protected system again

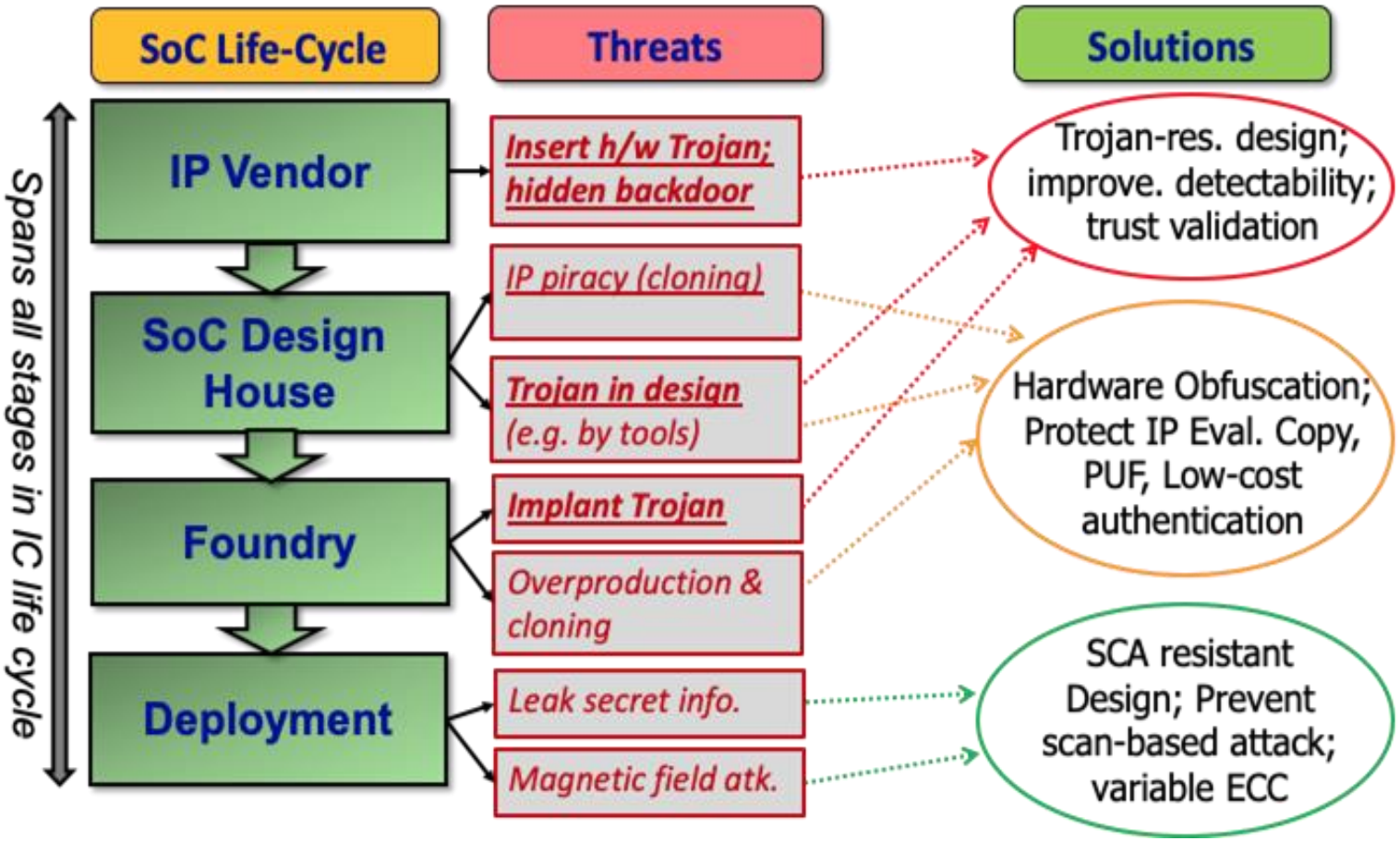
Some definitions

Threat: Set of circumstances that has the potential to cause loss or harm

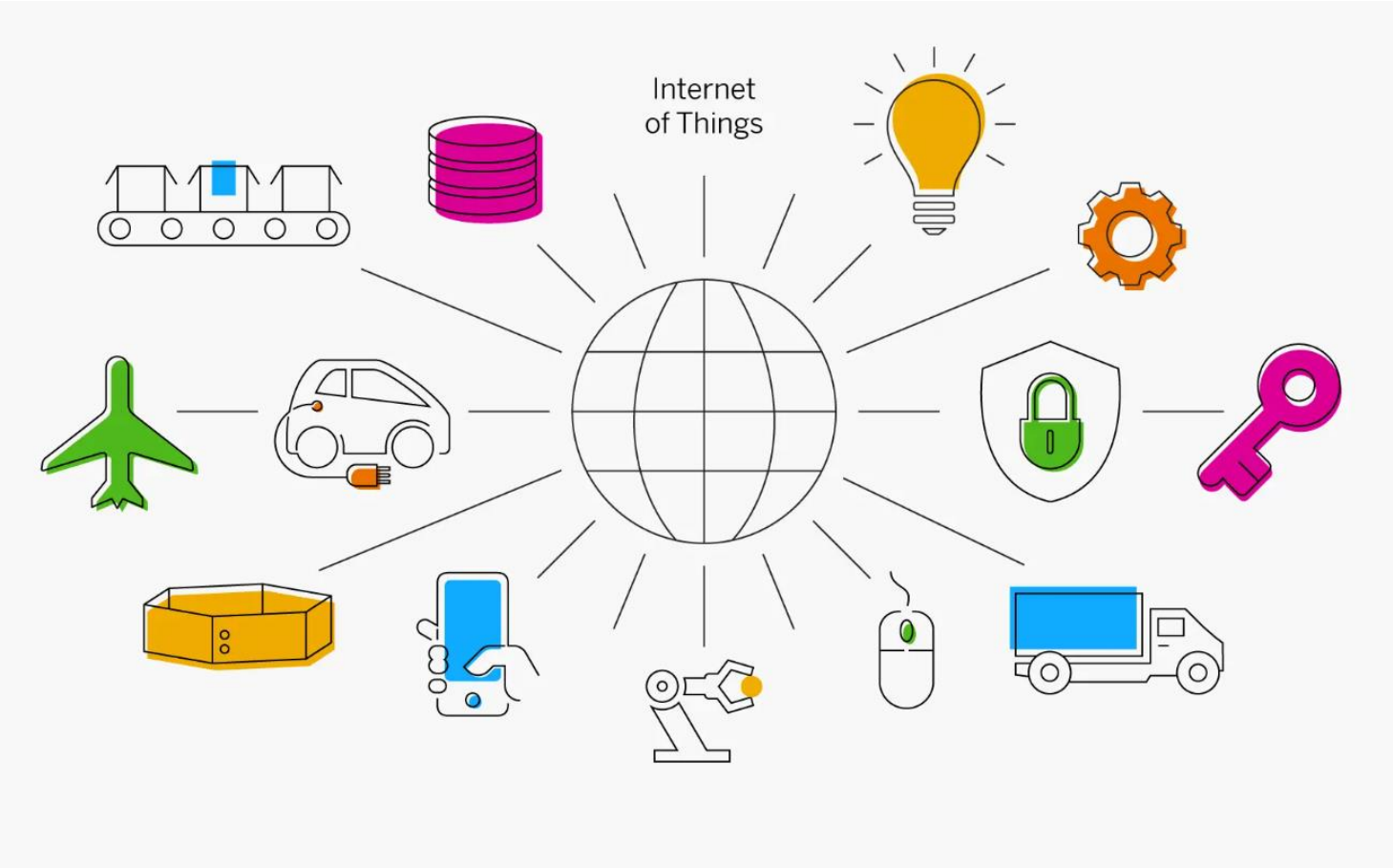
Vulnerability: Weakness in the secure system

Attack: The act of a human exploiting the vulnerability in the system

Hardware Threats



Where does security is required?



In the *cyber-physical world* security is not only related to data anymore

Embedded Systems Security/IoTs

Security processing adds overhead

- Performance and power

Security is challenging in embedded systems/IoTs

- Size and power constraints, and operation in harsh environments

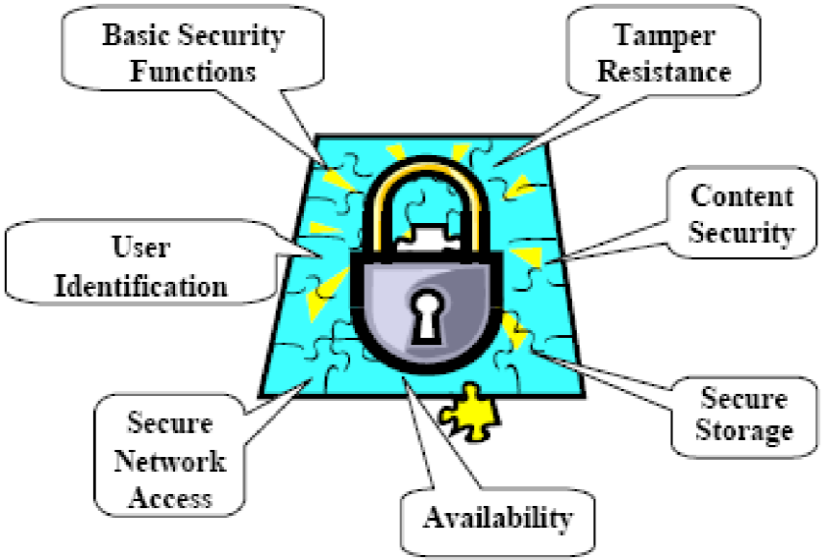
Security processing may easily overwhelm the other aspects of the system

Security has become a new design challenge that must be considered at the design time, along with other metrics, i.e., cost, power, area

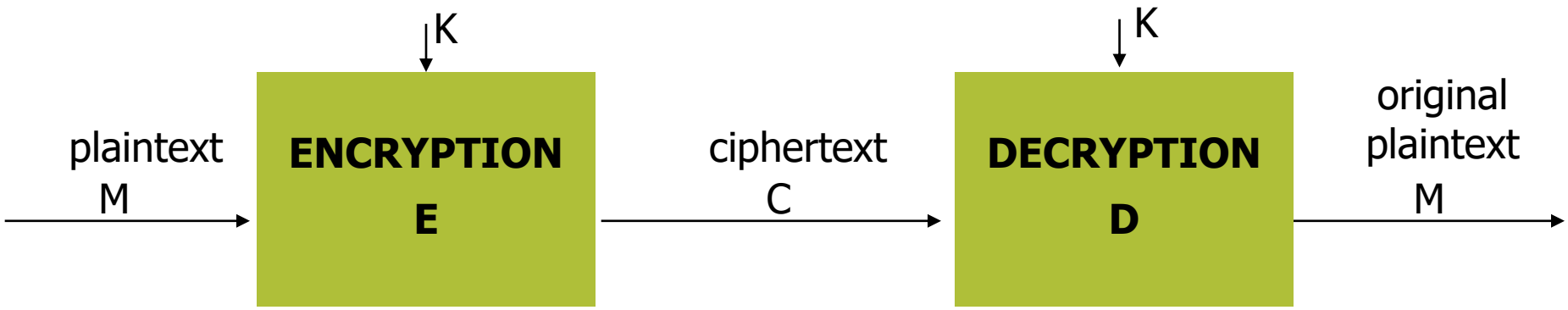
Basic security: cryptography

Crypto-algorithms enable:

- Confidentiality
- Integrity
- Authentication



Basic Cryptographic Scheme



- $M = <m_1, m_2, \ldots, m_n>$ m_i = i-th char of M
  - M = "DO NOT TELL ANYBODY" m_1 ="D", m_2 = "O", etc.

- $C = <c_1, c_2, \ldots, c_n>$ c_i = i-th char of C
  - C = "ep opu ufmm bozcpez" c_1 ="e", c_2 ="p", etc.

Cryptanalysis

- Cryptanalysts goals:
  - Deduce the key
  - Break a single message without deducing the key
  - Recognize patterns in encrypted msgs, to be able to break the subsequent ones

- Find a general weakness in an encryption algorithm

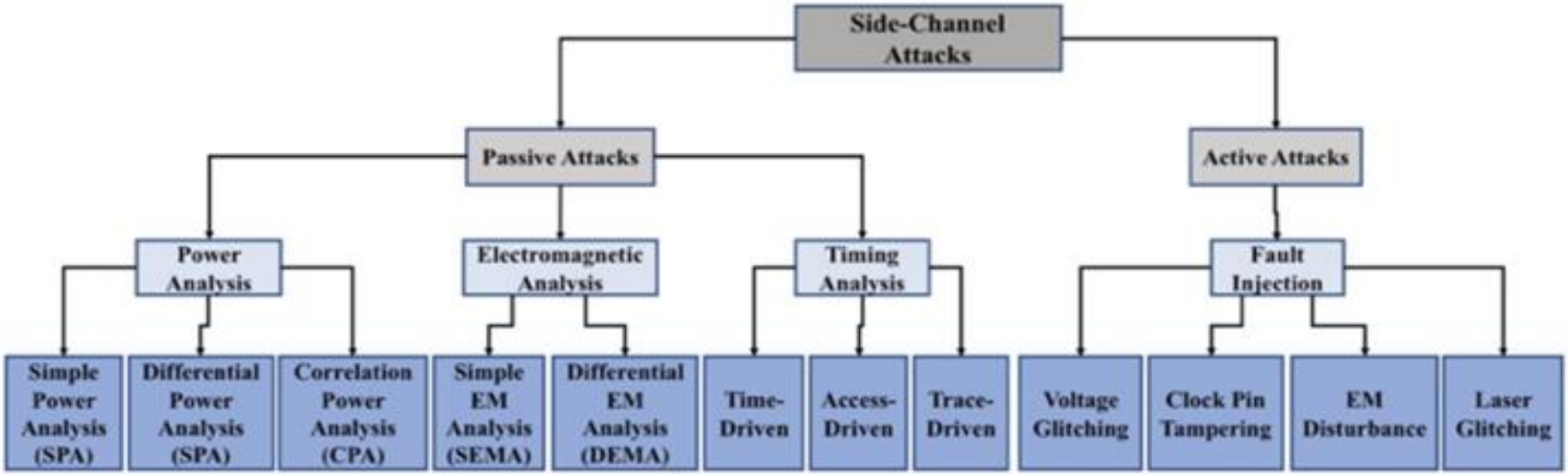
- **Find vulnerabilities in the implementation or the execution environment of an encryption algorithm**

Cryptanalysis

- Traditional attacks:
  - Ciphertext only attack
  - Known plaintext attack
  - Chosen plaintext attack
  - Chosen ciphertext attack

- **Side-channel attacks**

Taxonomy of Side Channel Attacks



Passive attacks: observe the behavior of the device to infer information about the secret

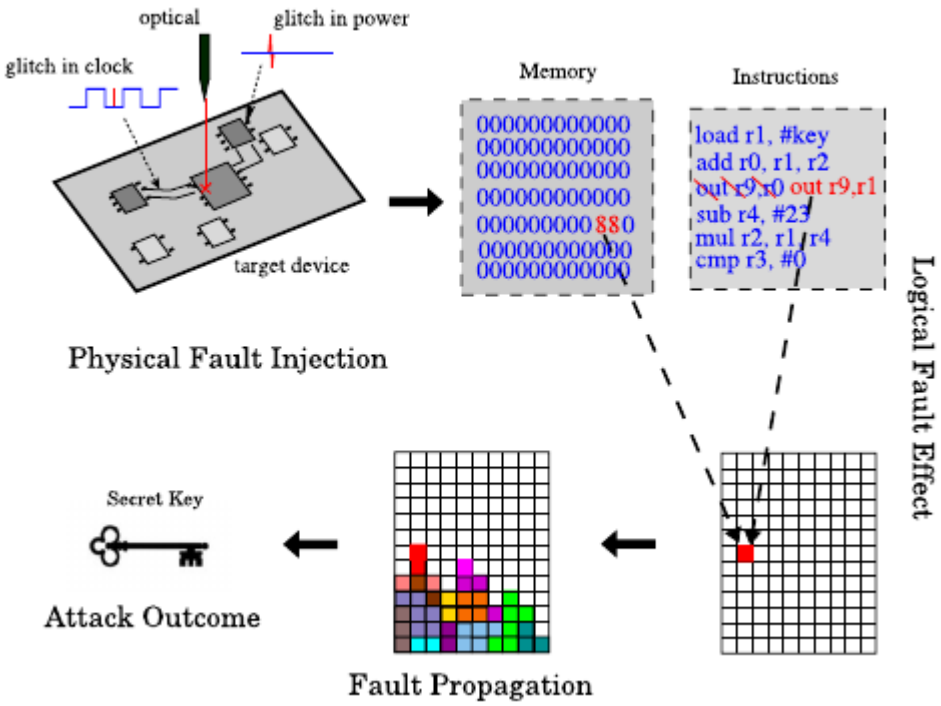
Active Attacks: physically operate on the device to gather information about secret (e.g. fault injection or microprobing)

Fault Injection Attacks

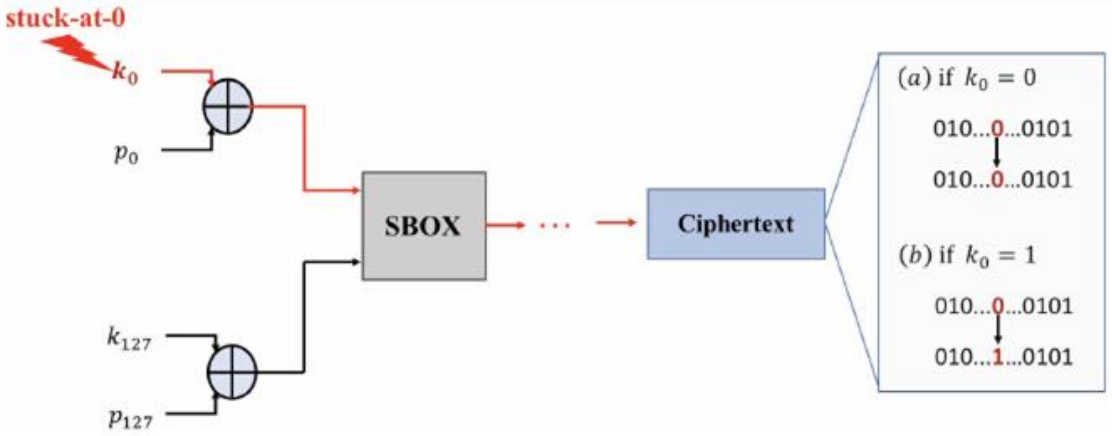
Methodology: the device executing the crypto algorithm is injected with a fault

Objective: break the implementation of the crypto algorithm

The injected fault introduces a temporary malfunction during the device operation



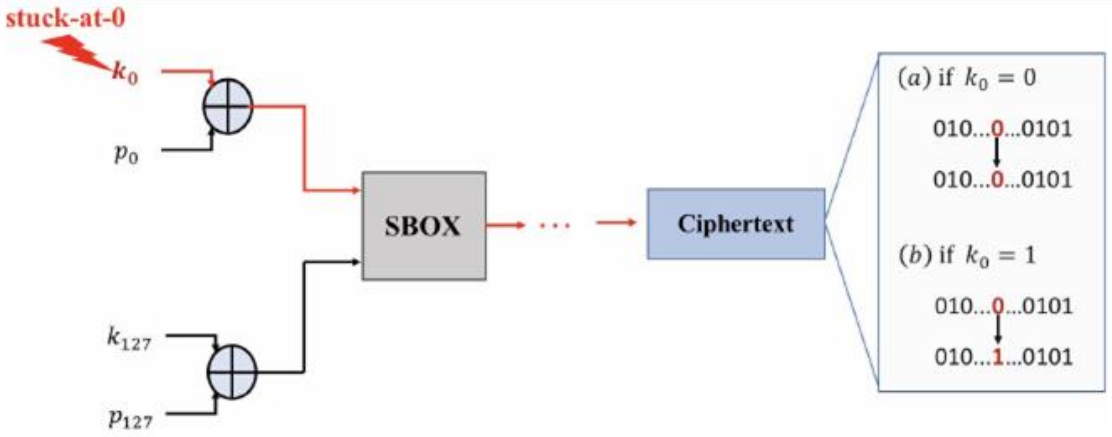
Fault Injection Attacks



Example: fault injection attack on AES

- The attacker is able to chose a plaintext *P* and to encrypt it, thus getting C_g
- The attacker wants to retrieve the secret key

Fault Injection Attacks



The attacker re-encrypts *P*

k_0 bit of the key in injected with a **stuck-at-zero fault**

The attacker gets C_f

Two possible outcomes:

- If $C_g = C_f$ the injected fault had no effect, thus k_0 was 0
- If C_g != C_f the injected fault affected the encryption, thus k_0 was 1

Fault Injection Techniques: low cost

Voltage Glitching: the device is supplied with a lower voltage.

- Not invasive
- High precision if the structure of the circuit is known

Fault Injection Techniques: low cost

Clock tampering: a *faulty* clock signal is fed into the system

- Either a glitch in the clock or the modification of the voltage level of the clock
- Not invasive
- High precision if the structure of the circuit is known

Fault Injection Techniques: low cost

EM Disturbance are generated and directed to the target device

- Not invasive
- Extremely difficult to be precise (the disturbance will affect the entire device)

Fault Injection Techniques: medium to high cost

Laser beams, radiation beams and light pulses attacks

- requires a decapsulated chip (invasive)
- precision depends on the quality (and the cost) of the beam

Fault Injection Attacks: the RSA example

Basics of RSA:

- m is the plaintext and c is the cyphertext
- $n = p * q$ public modulus, with p and q two large prime numbers ($0 <= m <= n$)
- e public exponent (calculated based on p and q)
- d private exponent (calculated based on p and q)
- e, d and n are such that
    - $(m^e)^d = m \bmod n$
    - $(m^d)^e = m \bmod n$
- The **public key** is (n, e)
- The **private key** is (n, d)

Fault Injection Attacks: the RSA example

Basics of RSA:

- when encrypting $c = m^e \bmod n$

Such encryption can be inverted (decrypted) only knowing the d associated with e

- when decrypting $m = c^d \bmod n$

Fault Injection Attacks: the RSA example

Attacks to RSA aim at:

- Factoring *n* thus recovering the prime numbers *p* and *q*
- Recovering *d*

Fault Injection Attacks: the RSA example

The ***Bellcore attack*** (a *chosen-plaintext attack*)*:*

- The plaintext *m* and the associate cyphertext *c* are given
- The attacker injects a fault while the circuit is decrypting *c* such that the produced plaintext is $m' = m + \delta$
- The attacker can now retrieve *p* (or *q*) by calculating the greatest commod divisor between δ and *n*
- It is enough to have *m* and *m'* to *break* RSA

Fault Injection Attacks: the RSA example

The ***safe-error attack****:*

- The attacker must know where the bits of the key are stored and when they are eused
- The attacker is able to flip a bit of the key when it is used
- If the flipped bit was 0 (flipped to 1) the produced output will be the same w.r.t. the expected one
- If the flipped bit was 1 (flipped to 0) the produced output will differ from the expected one

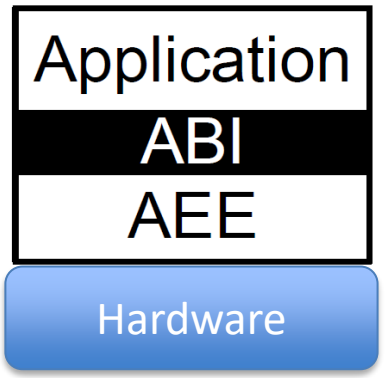
Security features in RISC-V

RISC-V security: sw stack implementations

Several ***software stack implementations*** with different levels of security are available

SINGLE APPLICATION SCENARIO

- No operating System
- The user application interacts with Application Execution Environment (AEE) through the Application Binary Interface (ABI)
- The AEE has direct access to the hardware



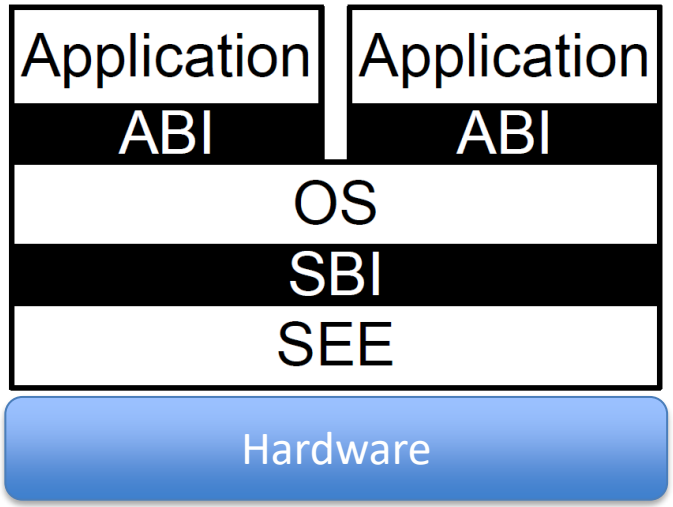
A. Waterman and K. Asanovic, "The risc-v instruction set manual volume ii: Privileged architecture document version 20190608-priv-msuratified," RISC-V Foundation, Tech. Rep., 2019.

RISC-V security: sw stack implementations

Several ***software stack implementations*** with different levels of security are available

MULTIPROGRAMMED SCENARIO

- The Operating System manages multiple applications
- User applications interacts with the OS through the ABIs
- In turn the OS interacts with the Supervisor Execution Environment (SEE) to the Supervisor Binary Interface (SBI)
- Finally the SEE has direct access to the hardware

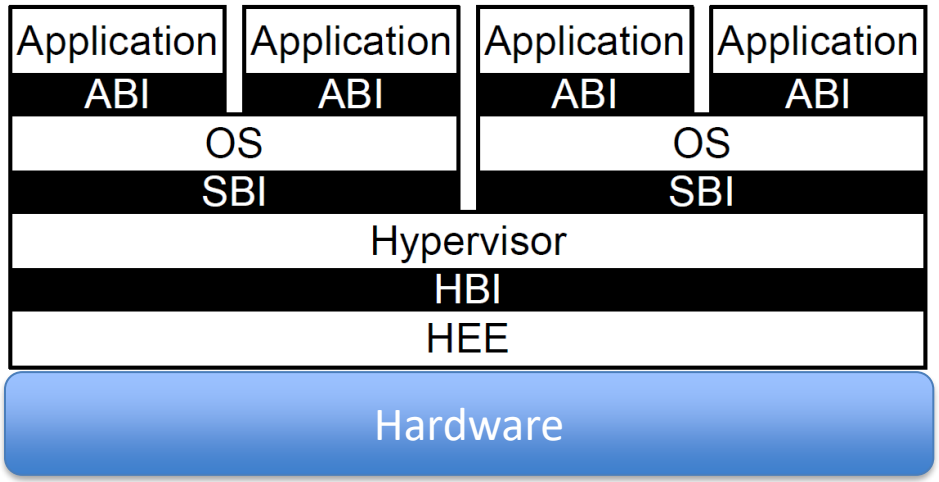


RISC-V security: sw stack implementations

Several ***software stack implementations*** with different levels of security are available

VIRTUALIZED SCENARIO

- Multiple OS are executed, each with its own SBI
- A Hypervisor manages the execution of the multiple OS
- The Hypervisor manages the OS and interact with the Hypervisor Execution Environment (HEE) through the Hypervisor Binary Interface (HBI)
- In turn, the HEE has direct access to the hardware



RISC-V security: instruction privileges

Several ***instruction privilege levels*** are available

- *M* (*Machine mode*): code running in machine mode is totally trusted and has direct access to the HW
    - (AEE, SEE and HEE are executed in M mode)
- *U* (*User mode*): untrusted code with no direct access to the HW
    - (user applications are executed in U mode)
- *S* (*Supervisor mode*): an additional mode introduced to provide isolation between the (possibly untrusted) OS and the Execution Environments

RISC-V security: Additional mechanisms

Physical Memory Protection (***PMP***)

The Machine mode (the highest privileged mode) allows to specify access privileges (read, write, execute) for each physical memory region

Attempting to fetch an instruction from a memory location that does not have execute permissions or to load data from a location without read permissions raise exceptions

RISC-V security: Additional mechanisms

Cryptography Extension

Instructions implementing cryptographic algorithms have been introduced to enable confidentiality of the data exchanged in the system

AES (for symmetric encryption) and SHA (for hashing) as well as lightweight algorithms like PRESENT and GOST are supported

RISC-V security: limitations

All the available security mechanisms have been designed to deal with "*traditional*" attacks

- Attempts to break data confidentiality and integrity
- Attempts to perform unauthorized accesses or executions

No protection is provided against "*novel*" hardware-based menaces like MSCAs and HTHs

Students assignments

- *“Lightweight Protection of Cryptographic Hardware Accelerators against Differential Fault Analysis”*, Ana Lasheras, Ramon Canal, Eva Rodríguez and Luca Cassano, In IOLTS 2020, the 26th IEEE International Symposium on On-Line Testing and Robust System Design, Naples (Italy), July 13-15, 2020
- *"Physical Unclonable Functions and Applications: A Tutorial,"* C. Herder, M. -D. Yu, F. Koushanfar and S. Devadas, in Proceedings of the IEEE, vol. 102, no. 8, pp. 1126-1141, Aug. 2014
- *"Anti-counterfeit Techniques: From Design to Resign,"* U. Guin, D. Forte and M. Tehranipoor, 2013 14th International Workshop on Microprocessor Test and Verification, Austin, TX, USA, 2013, pp. 89-94
- *"Building trusted ICs using split fabrication,"* K. Vaidyanathan, B. P. Das, E. Sumbul, R. Liu and L. Pileggi, 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), Arlington, VA, USA, 2014, pp. 1-6
- *"Robust, low-cost, and accurate detection of recycled ICs using digital signatures,"* M. Alam, S. Chowdhury, M. M. Tehranipoor and U. Guin, 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), Washington, DC, USA, 2018, pp. 209-214,

References

- M. Rostami, F. Koushanfar, J. Rajendran, R. Karri, "*Hardware security: threat models and metrics*", in: Proc. Int. Conf. Computer-Aided Design, 2013, pp. 819–823.
- Mohammad Tehranipoor, Cliff Wang, *"Introduction to Hardware Security and Trust"*, Springer-Verlag New York, 2012
- Potlapally, Nachiketh. "*Hardware security in practice: Challenges and opportunities*." 2011 IEEE International Symposium on Hardware-Oriented Security and Trust. IEEE, 2011.
- Hu, Wei, et al. "*An overview of hardware security and trust: Threats, countermeasures, and design tools*." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 40.6 (2020): 1010-1038.
- U. Guin, K. Huang, D. DiMase, J.M. Carulli, M. Tehranipoor, Y. Makris, *Counterfeit integrated circuits: a rising threat in the global semiconductor supply chain*, Proc. IEEE 102 (8) (2014) 1207–1228
- Guin, U., DiMase, D. & Tehranipoor, M. *"Counterfeit Integrated Circuits: Detection, Avoidance, and the Challenges Ahead*." Journal of Electronic Testing 30, 9–23 (2014).
- A. Carelli, C. A. Cristofanini, A. Vallero, C. Basile, P. Prinetto and S. Di Carlo, "S*ecuring bitstream integrity, confidentiality and authenticity in reconfigurable mobile heterogeneous systems,*" 2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), Cluj-Napoca, Romania, 2018, pp. 1-6, doi: 10.1109/AQTR.2018.8402795
- A. Barenghi, L. Breveglieri, I. Koren, D. Naccache, "*Fault injection attacks on cryptographic devices: theory, practice, and countermeasures*", Proc. IEEE 100 (11) (2012) 3056–3076
- D. Boneh, R.A. DeMillo, R.J. Lipton, "*On the importance of checking cryptographic protocols for faults*", International Conference on the Theory and Applications of Cryptographic Techniques, 1997, pp. 37–51.
- F. Bao, R.H. Deng, Y. Han, A. Jeng, A.D. Narasimhalu, T. Ngair, "*Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults*", International Workshop on Security Protocols (1997) 115–124.
- M. Joye, S.-M. Yen, "*The montgomery powering ladder*," International Workshop on Cryptographic Hardware and Embedded Systems, 2002, pp. 291–302.
- S.-M. Yen, M. Joye, "*Checking before output may not be enough against fault-based cryptanalysis*," IEEE Trans. Comput. 49 (9) (2000) 967–970.
- A. Barenghi, G. Bertoni, E. Parrinello, G. Pelosi, "*Low voltage fault attacks on the rsa cryptosystem*," in: 2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), 2009, pp. 23–31.