CASA
Cyber Security in the Age of Large-Scale Adversaries

Ruhr Universität Bochum
RUB

Max Planck Institute
for Security and Privacy

Unlocking Hardware Security

Red Team, Blue Team, and Trojan Tales

Steffen Becker, e7p, René Walendy

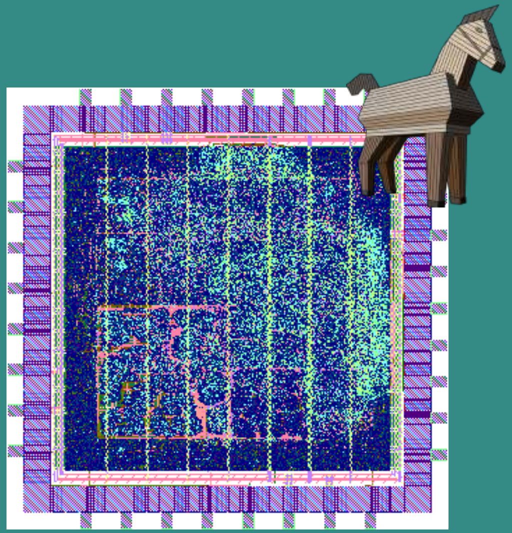
↘ 37th Chaos Communication Congress

↘ December 29th, 2023

↘ Hamburg, Germany

AGENDA

Hardware Trojan Basics



The Over-View

Red Team vs. Blue Team

Red Team vs. Blue Team: A Real-World Hardware Trojan Detection Case Study Across Four Modern CMOS Technology Generations

Endres Puschner*, Thorben Moos†, Steffen Becker‡*, Christian Kison§, Amir Moradi‡, and Christof Paar*
*Max Planck Institute for Security and Privacy, Germany †Université catholique de Louvain, Belgium
‡Ruhr University Bochum, Germany §Bundeskriminalamt, Germany
Email: {endres.puschner, christof.paar}@mpi-sp.org, thorben.moos@uclouvain.be,
{steffen.becker, christian.kison, amir.moradi}@rub.de

Abstract—Verifying the absence of maliciously inserted Trojans in Integrated Circuits (ICs) is a crucial task – especially for security-enabled products. Depending on the concrete threat model, different techniques can be applied for this purpose. Assuming that the original IC layout is benign and free of backdoors, the primary security threats are usually identified as the outsourced manufacturing and transportation. To ensure the absence of Trojans in commissioned chips, one straightforward solution is to compare the received semiconductor devices to the design files that were initially submitted to the foundry. Clearly, conducting such a comparison requires advanced laboratory equipment and qualified experts. Nevertheless, the fundamental techniques to detect Trojans which require evident changes to the silicon layout are nowadays well-understood. Despite this, there is a glaring lack of public case studies describing the process in its entirety while making the underlying datasets publicly available. In this work, we aim to improve upon this state of the art by presenting a public and open hardware Trojan detection case study based on four different digital ICs using a Red Team vs. Blue Team approach. Hereby, the Red Team creates small changes acting as surrogates for inserted Trojans in the layouts of 90 nm, 65 nm, 40 nm, and 28 nm ICs. The quest of the Blue Team is to detect all differences between digital layout and manufactured device by means of a GDSII–vs–SEM-image comparison. Can the Blue Team perform this task efficiently? Our results spark optimism for the Trojan seekers and answer common questions about the efficiency of such techniques for relevant IC sizes. Further, they allow to draw conclusions about the impact of technology scaling on the detection performance.

Index Terms—Hardware Trojans, Very Large Scale Integration, GDSII, Integrated Circuits Verification

1. Introduction

Hardware in the form of digital Integrated Circuits (ICs)
the basis of all IT systems and frequently serves as
ot of trust for security-critical applications. Modern
ies spend billions of dollars in investments to facil-
he rapid advances in semiconductor manufacturing
logy seen in recent decades [1], [2]. Consequently, many hardware design houses cannot afford to keep pace and decide to operate fabless instead, i.e., without their own manufacturing facilities. IC production is then outsourced to contract manufacturers (foundries) that offer to fabricate commissioned chips in a portfolio of process technologies. However, these contract manufacturers cannot always be trusted, as they are in the optimal position to intentionally perform stealthy manipulations – i.e., implement hardware Trojans – in the IC designs of their customers [3]. The design houses and foundries involved in the chip making process may be located in places of the world with vastly different cultural, legal and political structures. Thus, it is only reasonable to consider the possibility of adversarial motivations and to be wary of the integrity of critical devices fabricated by untrusted entities. The transport of digital data or manufactured devices between parties is another vulnerable part of the supply chain, as malicious manipulations may also be performed during transit [4]. The most basic example of a malicious hardware Trojan is a kill switch that can disable (parts of) an IC's functionality on demand [5]. Such Trojans can be implemented with a very low overhead [6]. Beyond such comparably simple constructions, many different Trojan designs with varying degrees of sophistication have been proposed in literature – and the possibilities seem almost endless. We review the relevant state of the art in Section 5. We conclude that
fabless design
in dire need of techniques to
verify that co
duced and delivered
exactly as orde
unintentional
modifications. Si
interest of
trustworthy found
crease
their customer
ness
model. In this p
ng
research question in

How efficient
Trojans[1] *in fa*
sively smaller

To answer this res
ity of tiny and h

[1] That is, Troja
ly sma
ying
gates according to
Karri et

1

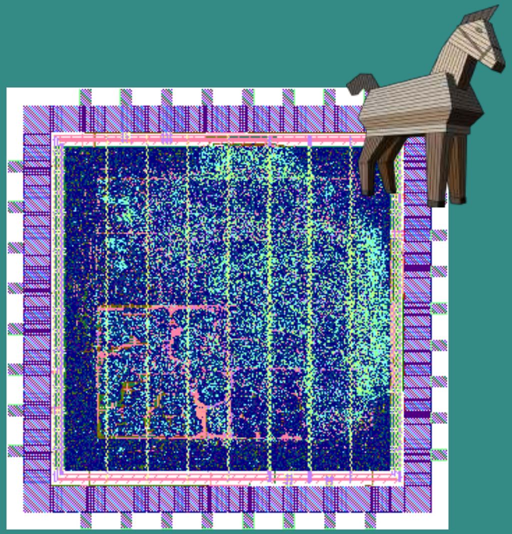
IEEE S&P 2023

The Technical View

Reverse Engineering: A Human-Centred Perspective

The Community View

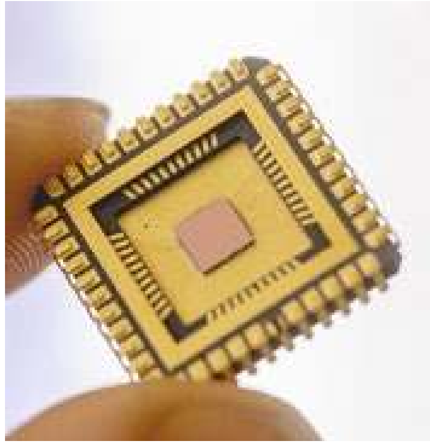
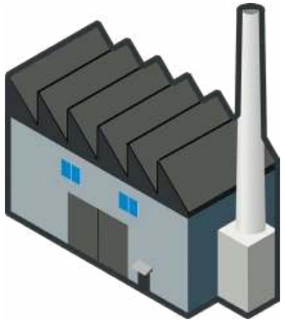
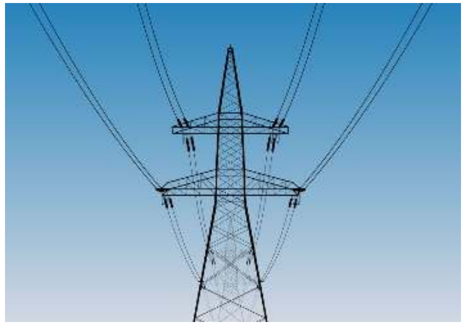
AGENDA



Hardware Trojan Basics

The Over-View

INTEGRATED CIRCUITS FULFILL MISSION-CRITICAL OBJECTIVES IN MANY SYSTEMS

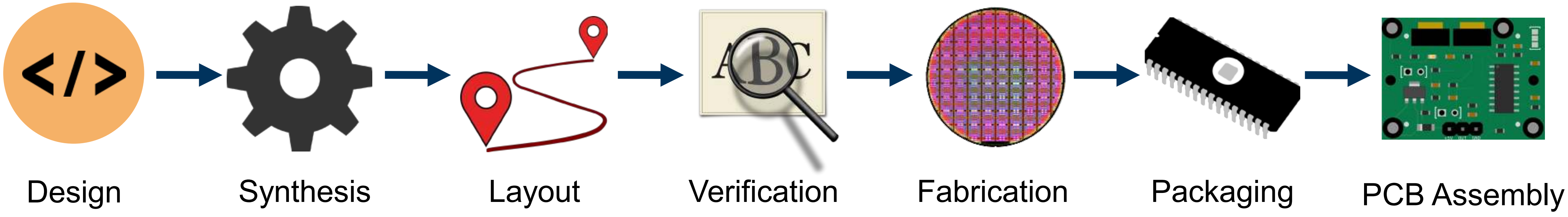
Mission-critical objectives include:

- cryptography & encryption
- location services
- wireless communication
- real-time monitoring
- fault detection & prevention
- power management
- signal processing
- …

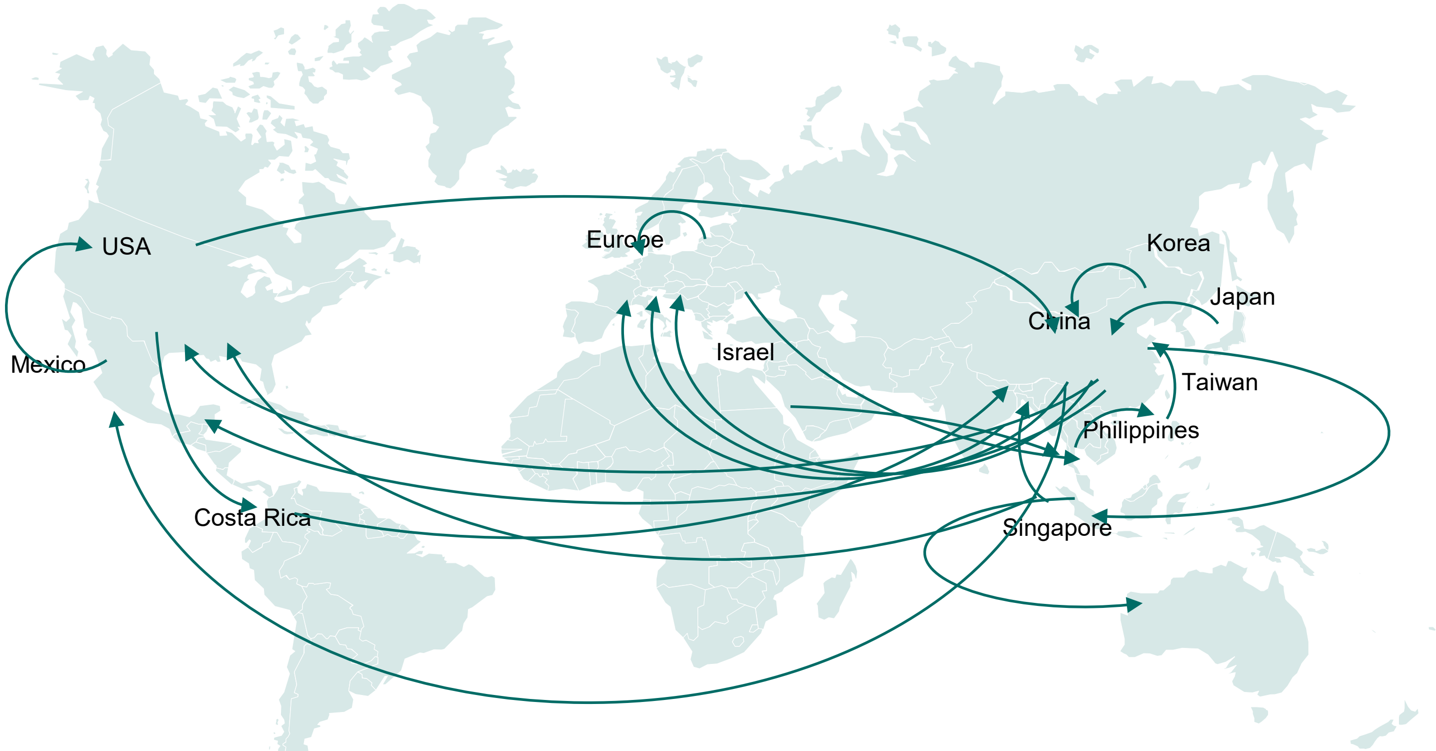
INTEGRATED CIRCUITS ARE TINY AND HIGHLY COMPLEX

Zoom Into a Microchip, NISENet, CC BY 3.0, via YouTube

IC DESIGN FLOW DISTRIBUTED ACROSS GLOBAL SUPPLY CHAIN



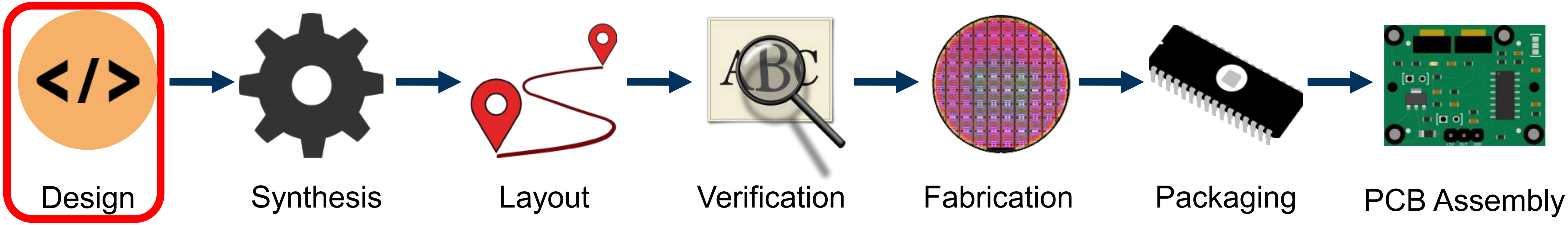
Design → Synthesis → Layout → Verification → Fabrication → Packaging → PCB Assembly



Supply chain threats include:

- Weakening of cryptographic primitives
- Information leakage
- Denial of service (kill switch)
- Safety hazards
- …

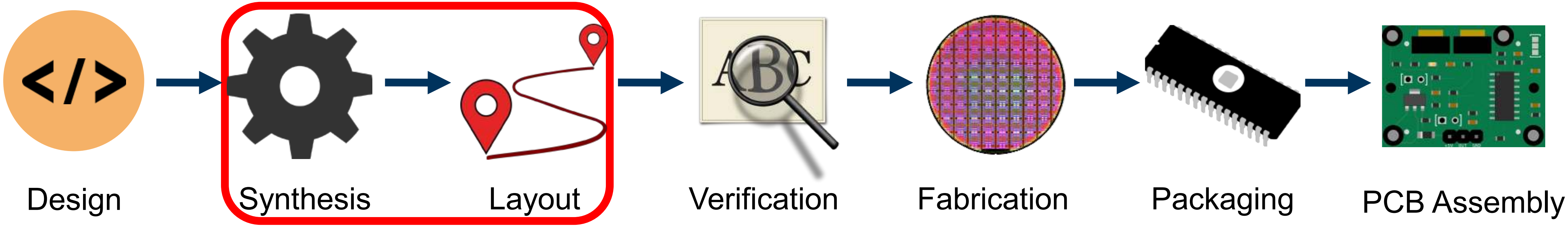
MANY ENTRY POINTS FOR ATTACKERS



Malicious Designer (implants Backdoor)

- Can arbitrarily add or change any function of the chip
- Either during IC design phase or through third-party IP cores

MANY ENTRY POINTS FOR ATTACKERS

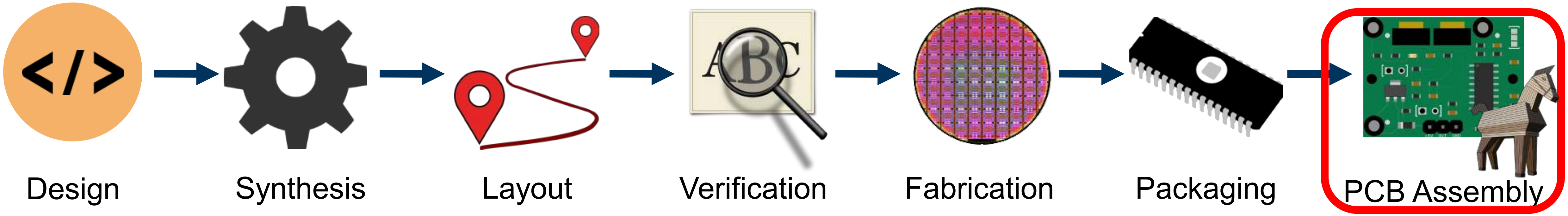


Design → Synthesis → Layout → Verification → Fabrication → Packaging → PCB Assembly

Subverted Design Tool

- (Ex works or after delivery) modified design tool
- Automatically inserts manipulations during synthesis or place & route
- May even tamper with verification tools to prevent detection

MANY ENTRY POINTS FOR ATTACKERS



System-level Hardware Trojan

- Add “spy ICs” to PCB, manipulate PCB routing, …
- For example, during PCB assembly or transportation
- May be detected during optical inspection, product teardown or via XRay

SYSTEM-LEVEL HARDWARE TROJANS – A CASE STUDY

“The Big Hack“ (Bloomberg, October 2018)

- Based on anonymous sources
- In 2015, a security analysis lab presumably found a system-level hardware Trojan
- **Injected during assembly** of specific Supermicro server blades

- **Thousands of these boards were used** for computation-intensive tasks such as
  - Video compression (Apple, Amazon)
  - CIA drone operations, navy warships
  - Secure video conferences

- **High-impact target** for hardware Trojans

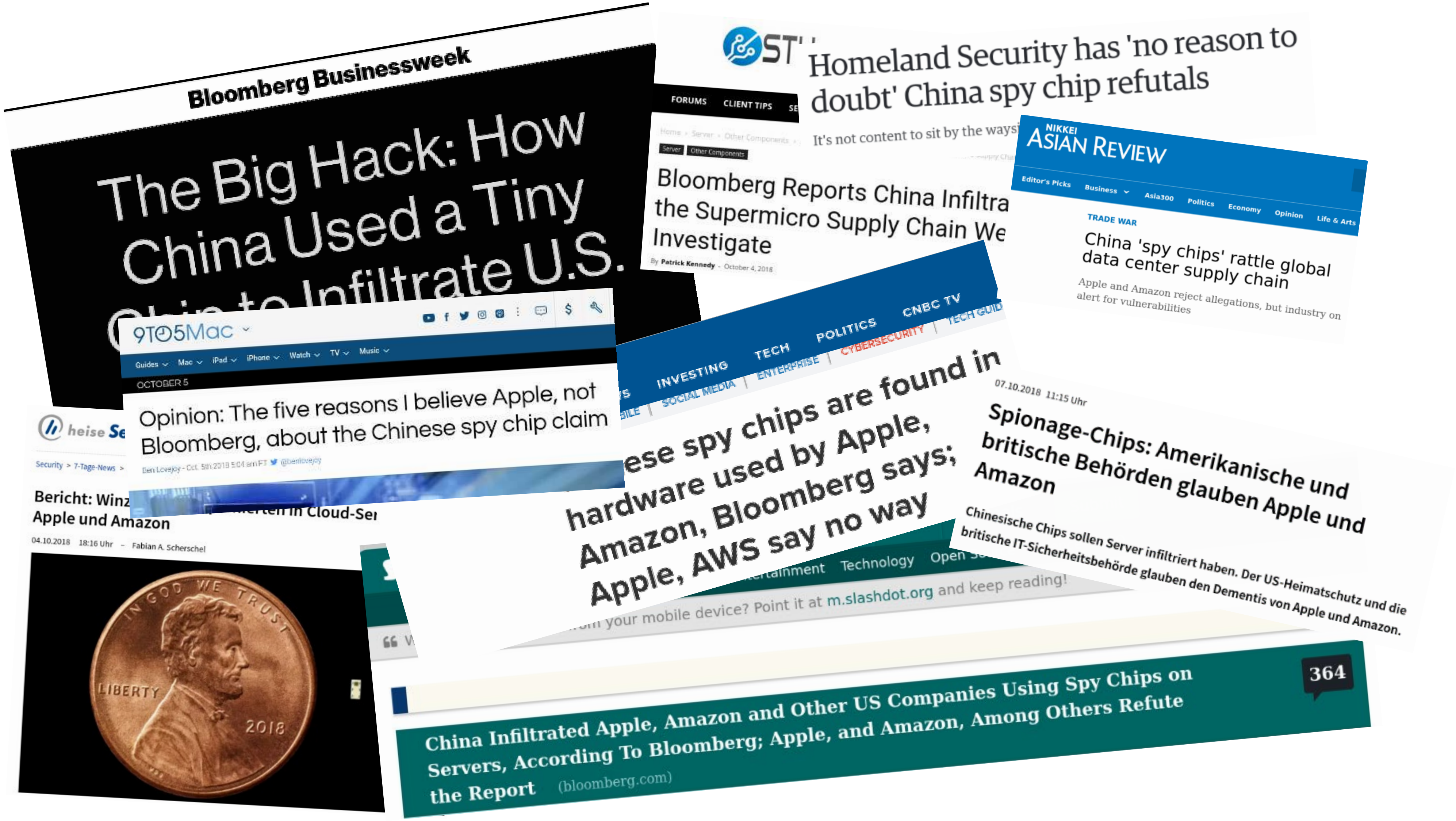
Bloomberg Businessweek

October 8, 2018

The Big Hack

How China used a tiny chip to infiltrate America’s top companies

SYSTEM-LEVEL HARDWARE TROJANS – A CASE STUDY



Bloomberg Businessweek

The Big Hack: How China Used a Tiny Chip to Infiltrate U.S.

Homeland Security has 'no reason to doubt' China spy chip refutals

It's not content to sit by the waysi

FORUMS CLIENT TIPS SE

Home › Server › Other Components ›

Server Other Components

Bloomberg Reports China Infiltra the Supermicro Supply Chain We Investigate

By Patrick Kennedy - October 4, 2018

NIKKEI ASIAN REVIEW

Editor's Picks Business Asia300 Politics Economy Opinion Life & Arts

TRADE WAR

China 'spy chips' rattle global data center supply chain

Apple and Amazon reject allegations, but industry on alert for vulnerabilities

9TO5Mac

Guides Mac iPad iPhone Watch TV Music

OCTOBER 5

Opinion: The five reasons I believe Apple, not Bloomberg, about the Chinese spy chip claim

Ben Lovejoy - Oct. 5th 2018 5:04 am PT @benlovejoy

INVESTING TECH POLITICS CNBC TV

SOCIAL MEDIA ENTERPRISE CYBERSECURITY TECH GUID

ese spy chips are found in hardware used by Apple, Amazon, Bloomberg says; Apple, AWS say no way

heise Security

Security > 7-Tage-News >

Bericht: Winz ... rten in Cloud-Ser Apple und Amazon

04.10.2018 18:16 Uhr – Fabian A. Scherschel

07.10.2018 11:15 Uhr

Spionage-Chips: Amerikanische und britische Behörden glauben Apple und Amazon

Chinesische Chips sollen Server infiltriert haben. Der US-Heimatschutz und die britische IT-Sicherheitsbehörde glauben den Dementis von Apple und Amazon.

rtainment Technology Open S

om your mobile device? Point it at m.slashdot.org and keep reading!

China Infiltrated Apple, Amazon and Other US Companies Using Spy Chips on Servers, According To Bloomberg; Apple, and Amazon, Among Others Refute the Report (bloomberg.com)

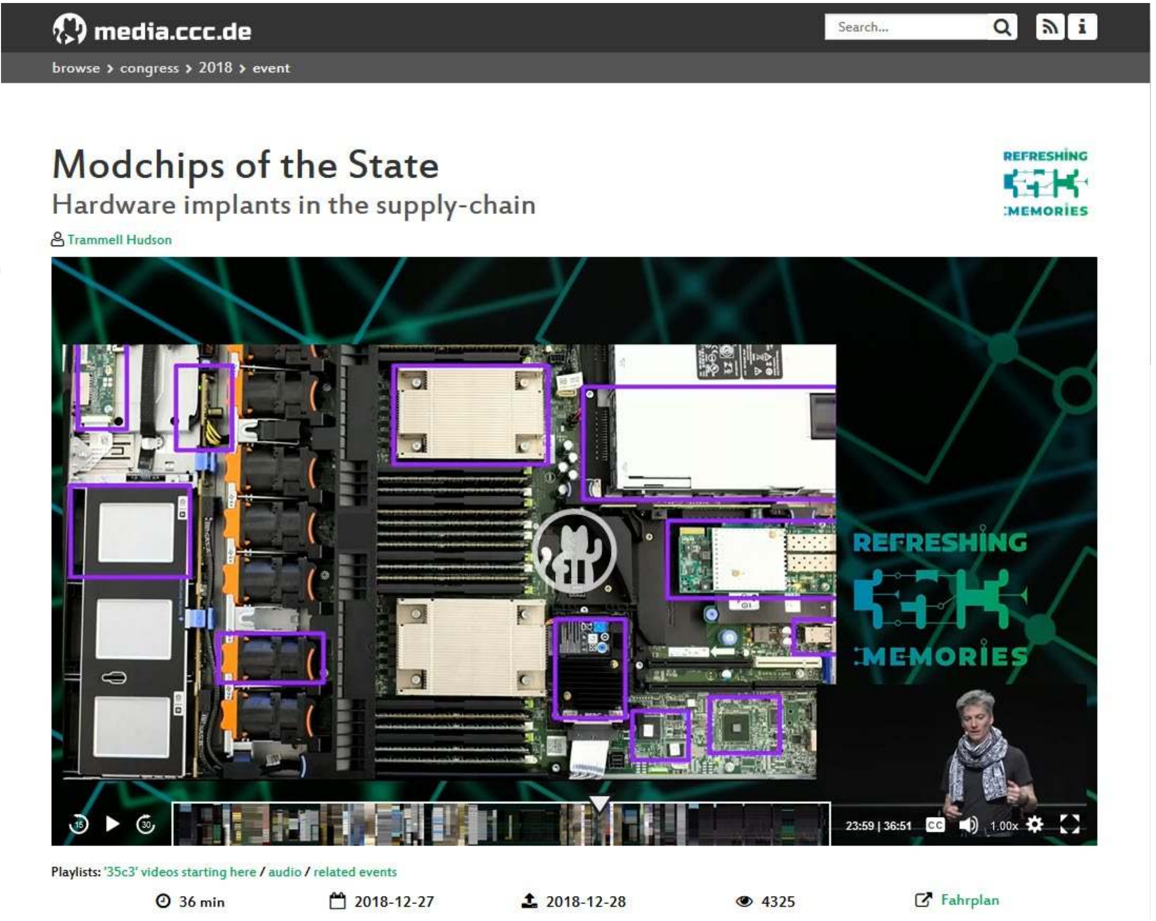
364

TECHNICAL DETAILS OF BLOOMBERG HARDWARE TROJAN

- Detailed technical analysis in Trammell Hudson's 35C3 talk "**Modchips of the State**"

- BMC implant is indeed feasible from a technical perspective
- Hiding the implant is not that easy

- URL: https://media.ccc.de/v/35c3-9597-modchips_of_the_state



media.ccc.de

Search...

browse › congress › 2018 › event

Modchips of the State

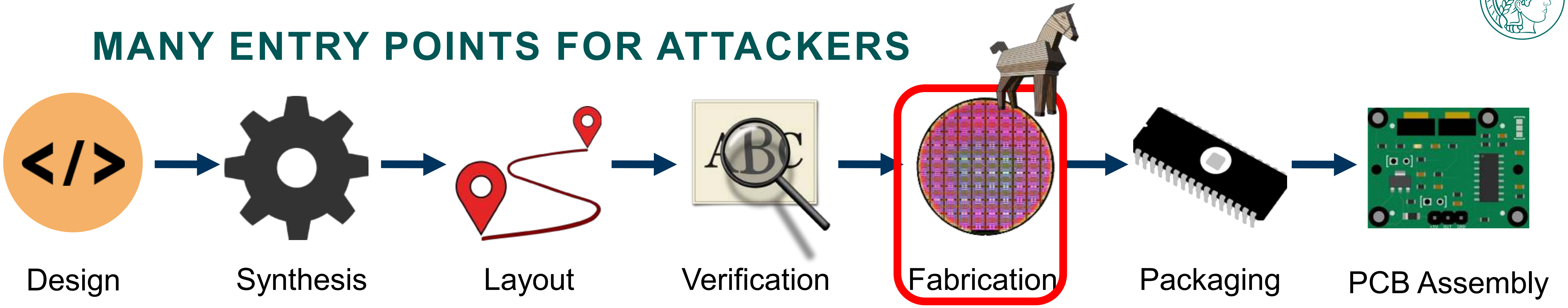
Hardware implants in the supply-chain

Trammell Hudson

Playlists: '35c3' videos starting here / audio / related events

36 min 2018-12-27 2018-12-28 4325 Fahrplan

MANY ENTRY POINTS FOR ATTACKERS



Design → Synthesis → Layout → Verification → Fabrication → Packaging → PCB Assembly

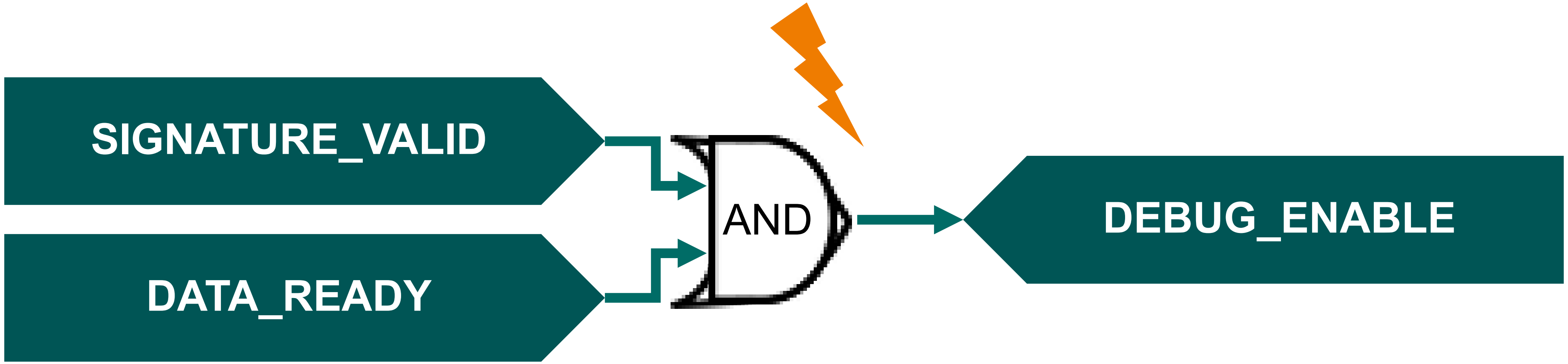
Silicon-level Hardware Trojan

- Arbitrary additions or manipulations of on-chip primitives
- For example, via added or modified logic cells, through routing manipulation, etc.
- Manipulation of design files (pre manufacturing) or edits via focused ion beam (post manufacturing)

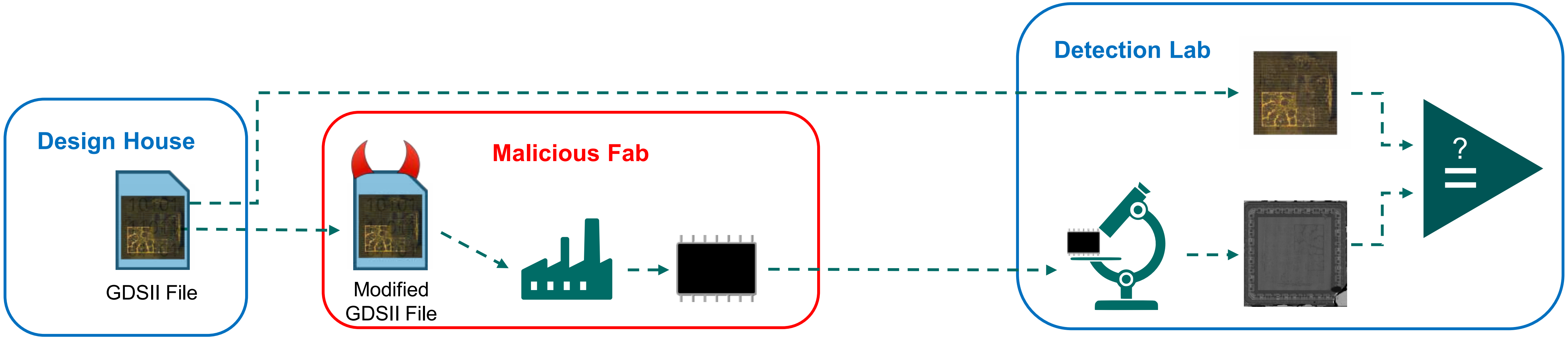
SILICON-LEVEL HARDWARE TROJAN – BASIC EXAMPLE

Modification: Replace AND cell with OR cell

Impact: Debugger authentication bypass



INSERTION & DETECTION OF SILICON-LEVEL HARDWARE TROJANS

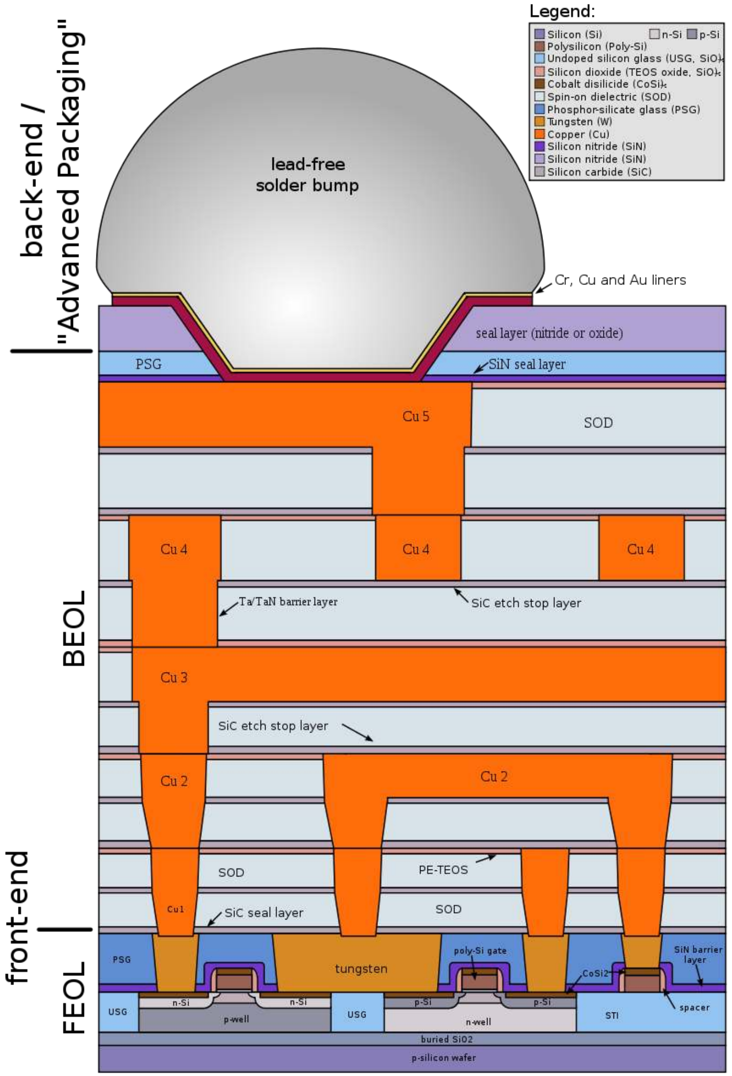


- Fab-less design house sends design file to fab
- Malicious fab edits the design before production (replaces cells)
- Detection lab compares original design file to the manufactured chip

INSIDE CHIPS: STACKED LAYERS

- **Cross-section view**
- Modern **ICs consist of multiple layers**
  - Transistors on the polysilicon layer
  - Connections using metal layers
- A **standard cell** is commonly formed using only the **polysilicon layer & the lowest metal layer (M1)**

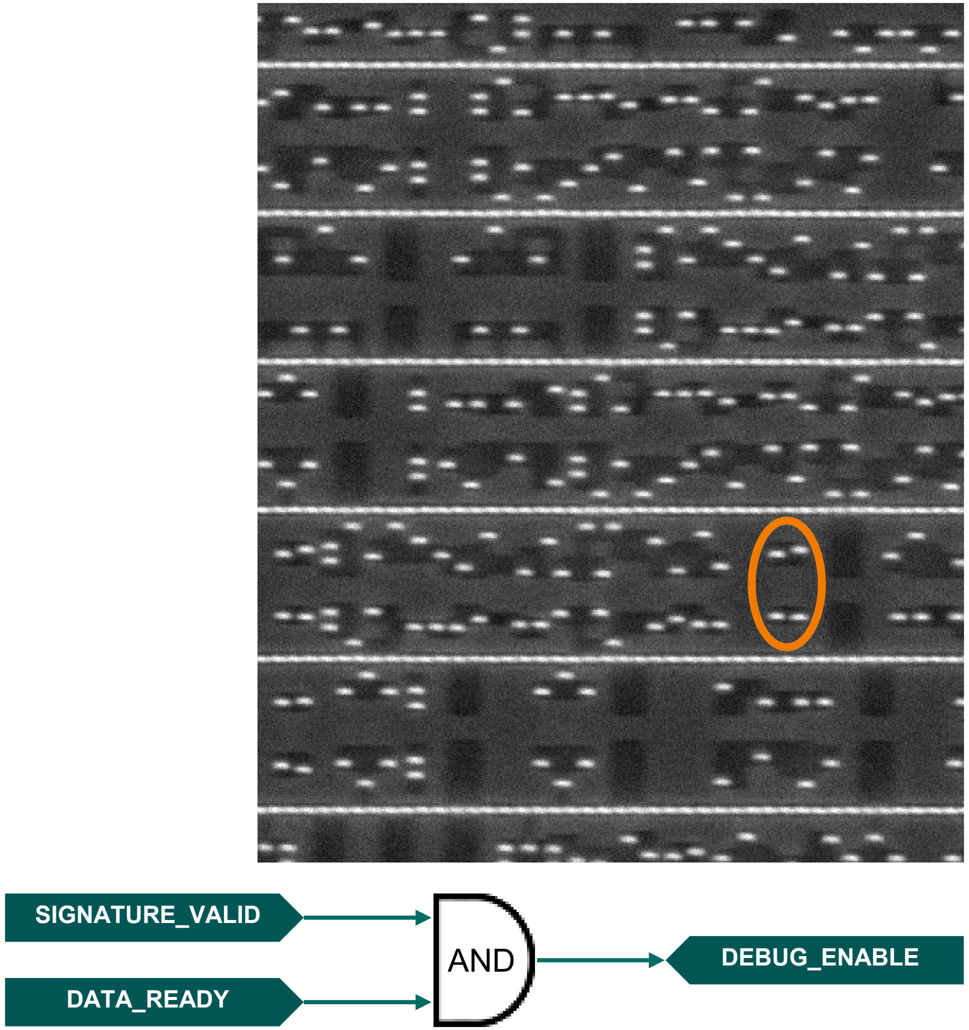
Region of interest



CMOS-chip structure in 2000s (en), Cepheiden, CC BY 2.5, via Wikimedia Commons

- Detection lab thins IC from the backside and takes images with scanning electron microscope

HOW IT SHOULD LOOK LIKE VS. HOW IT ACTUALLY LOOKS

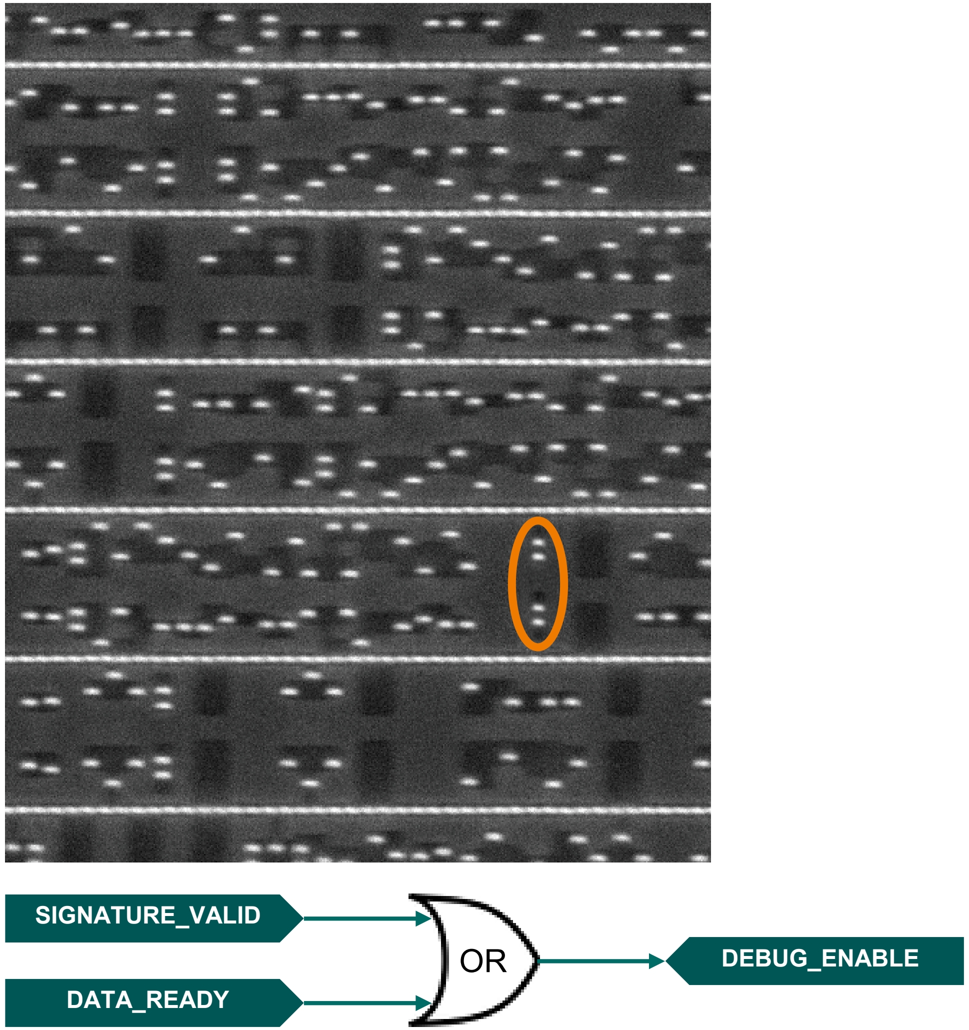


SIGNATURE_VALID

DATA_READY

AND

DEBUG_ENABLE



SIGNATURE_VALID

DATA_READY

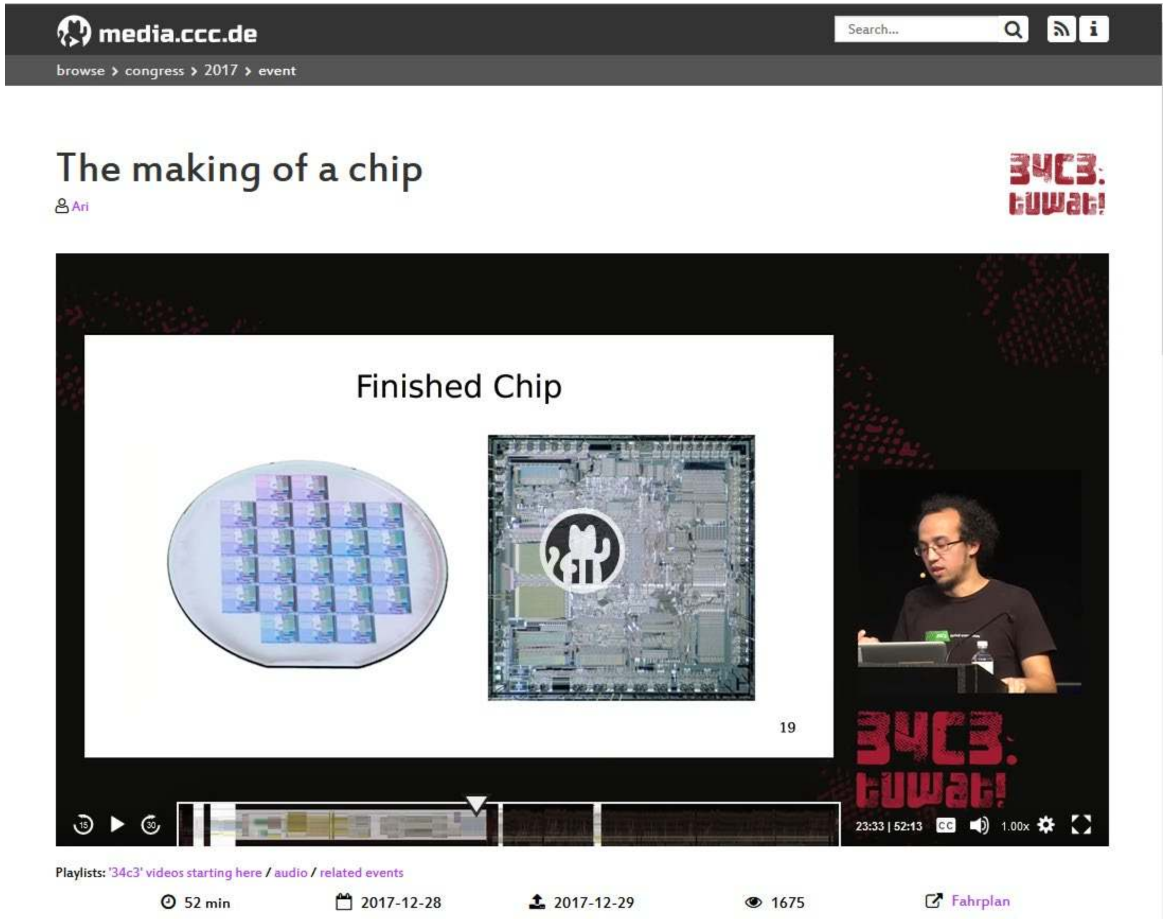
OR

DEBUG_ENABLE

REFERENCE: HOW ARE CHIPS MADE?

- Many more details in Ari's 34C3 talk **"The making of a chip"**

- URL: https://media.ccc.de/v/34c3-9250-the_making_of_a_chip



The making of a chip

Ari

Playlists: '34c3' videos starting here / audio / related events

52 min 2017-12-28 2017-12-29 1675 Fahrplan

AGENDA

Red Team vs. Blue Team: A Real-World Hardware Trojan Detection Case Study Across Four Modern CMOS Technology Generations

Endres Puschner*, Thorben Moos†, Steffen Becker‡*, Christian Kison§, Amir Moradi‡, and Christof Paar*
*Max Planck Institute for Security and Privacy, Germany †Université catholique de Louvain, Belgium
‡Ruhr University Bochum, Germany §Bundeskriminalamt, Germany
Email: {endres.puschner, christof.paar}@mpi-sp.org, thorben.moos@uclouvain.be,
{steffen.becker, christian.kison, amir.moradi}@rub.de

Abstract—Verifying the absence of maliciously inserted Trojans in Integrated Circuits (ICs) is a crucial task – especially for security-enabled products. Depending on the concrete threat model, different techniques can be applied for this purpose. Assuming that the original IC layout is benign and free of backdoors, the primary security threats are usually identified as the outsourced manufacturing and transportation. To ensure the absence of Trojans in commissioned chips, one straightforward solution is to compare the received semiconductor devices to the design files that were initially submitted to the foundry. Clearly, conducting such a comparison requires advanced laboratory equipment and qualified experts. Nevertheless, the fundamental techniques to detect Trojans which require evident changes to the silicon layout are nowadays well-understood. Despite this, there is a glaring lack of public case studies describing the process in its entirety while making the underlying datasets publicly available. In this work, we aim to improve upon this state of the art by presenting a public and open hardware Trojan detection case study based on four different digital ICs using a Red Team vs. Blue Team approach. Hereby, the Red Team creates small changes acting as surrogates for inserted Trojans in the layouts of 90 nm, 65 nm, 40 nm, and 28 nm ICs. The quest of the Blue Team is to detect all differences between digital layout and manufactured device by means of a GDSII–vs–SEM-image comparison. Can the Blue Team perform this task efficiently? Our results spark optimism for the Trojan seekers and answer common questions about the efficiency of such techniques for relevant IC sizes. Further, they allow to draw conclusions about the impact of technology scaling on the detection performance.

Index Terms—Hardware Trojans, Very Large Scale Integration, GDSII, Integrated Circuits Verification

1. Introduction

Hardware in the form of digital Integrated Circuits (ICs) orms the basis of all IT systems and frequently serves as he root of trust for security-critical applications. Modern oundries spend billions of dollars in investments to facilate the rapid advances in semiconductor manufacturing chnology seen in recent decades [1], [2]. Consequently, many hardware design houses cannot afford to keep pace and decide to operate fabless instead, i.e., without their own manufacturing facilities. IC production is then outsourced to contract manufacturers (foundries) that offer to fabricate commissioned chips in a portfolio of process technologies. However, these contract manufacturers cannot always be trusted, as they are in the optimal position to intentionally perform stealthy manipulations – i.e., implement hardware Trojans – in the IC designs of their customers [3]. The design houses and foundries involved in the chip making process may be located in places of the world with vastly different cultural, legal and political structures. Thus, it is only reasonable to consider the possibility of adversarial motivations and to be wary of the integrity of critical devices fabricated by untrusted entities. The transport of digital data or manufactured devices between parties is another vulnerable part of the supply chain, as malicious manipulations may also be performed during transit [4]. The most basic example of a malicious hardware Trojan is a kill switch that can disable (parts of) an IC's functionality on demand [5]. Such Trojans can be implemented with a very low overhead [6]. Beyond such comparably simple constructions, many different Trojan designs with varying degrees of sophistication have been proposed in literature – and the possibilities seem almost endless. We review the relevant state of the art in Section 5. We conclude that fabless design houses are in dire need of techniques to verify that commissioned chips are produced and delivered exactly as ordered, without any intentional or unintentional modifications. Such means should also be in the interest of trustworthy foundries and shipping companies to increase their customer's confidence in the honesty of their business model. In this paper, we therefore address the following research question in a holistic manner:

> *How efficiently can we detect functional hardware Trojans[1] in full-sized ICs manufactured in progressively smaller CMOS technologies?*

To answer this research question, we analyze the detectability of tiny and hidden silicon modifications in four differ-

[1] That is, Trojans that are physically realized by adding or modifying gates according to the taxonomy of Karri et al. [7].

1

The Technical View

Red Team vs. Blue Team

A Real-World Hardware Trojan Detection Case Study Across Four Modern CMOS Technology Generations

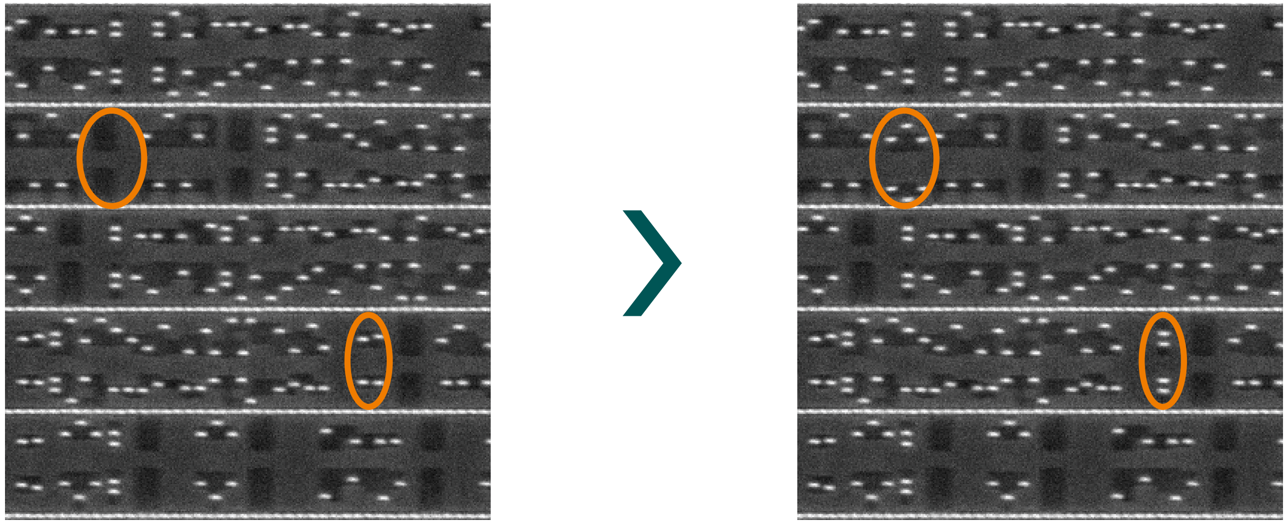
IEEE Symposium on Security and Privacy ’23

Distinguished Paper Award

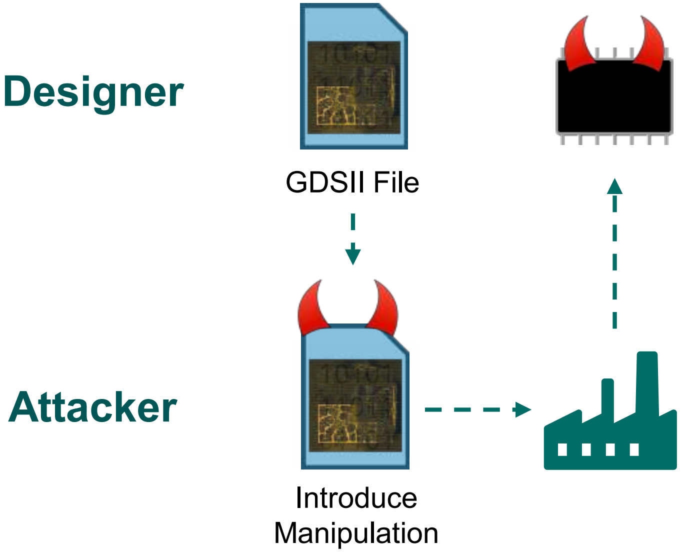
IEEE S&P 2023

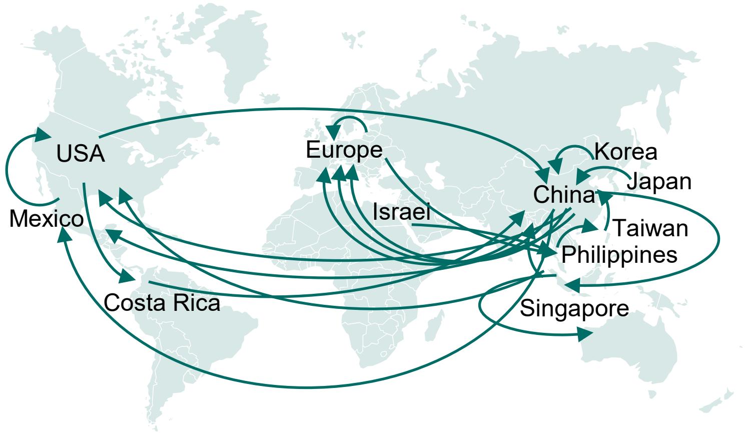
REMINDER: HARDWARE TROJANS

- Malicious modifications of integrated circuits (ICs)
- Example payloads: Kill switch; information leakage; …
- Possible realization: **Alter chip behavior by adding or replacing logic cells**



THREAT MODEL





- Distributed manufacturing
- Relevant scenario: malicious fab / transport
- Other steps are "Trojan free"

Main Research Question:

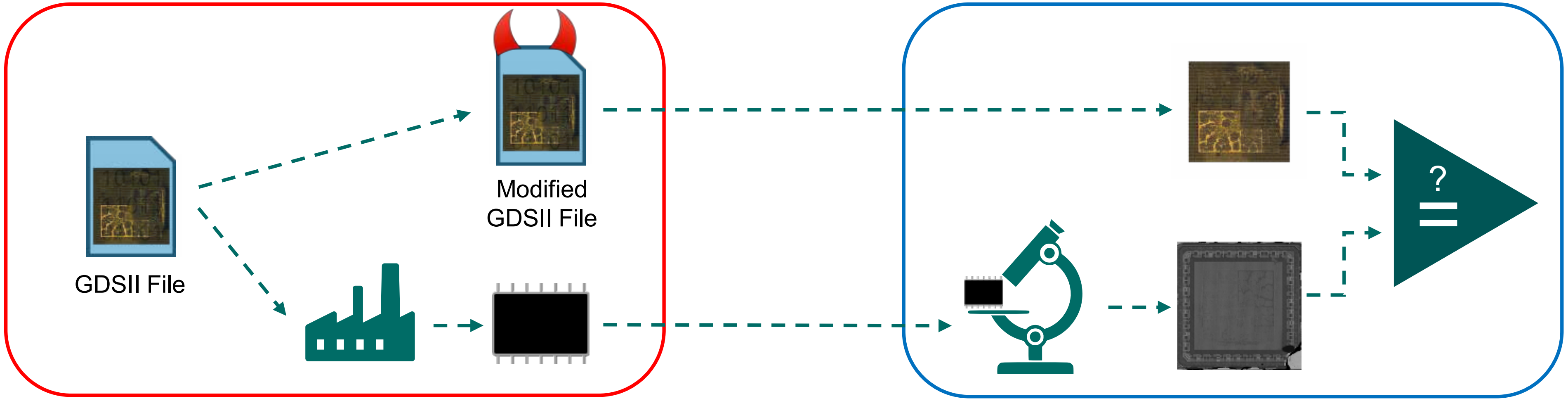
How efficiently can we detect functional hardware Trojans in full-sized ICs manufactured in progressively smaller CMOS technologies?

OUR APPROACH: RED TEAM VS. BLUE TEAM

Purpose: minimize research bias

RED TEAM (resembles malicious third party)

BLUE TEAM (resembles analysis lab)



→ creates delta between chips and design files

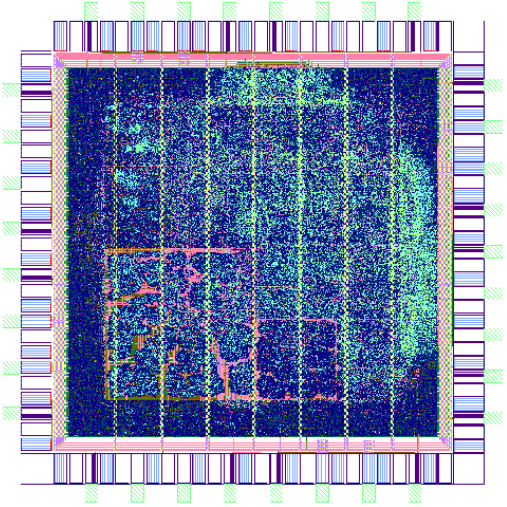
→ receives chips & design files from RED TEAM

→ tries to find the differences

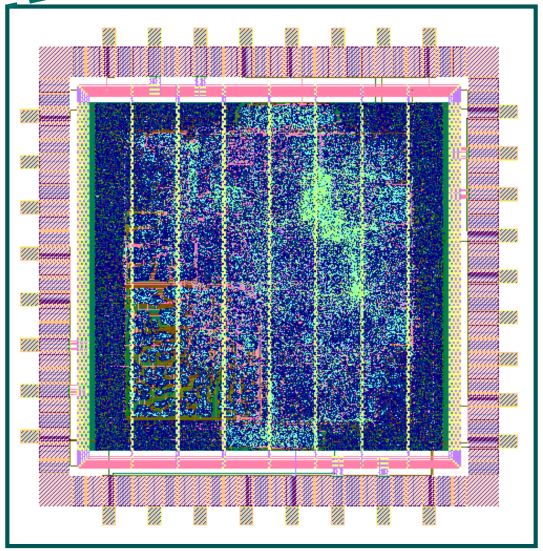
1

Red Team: INTRODUCE CELL MODIFICATIONS

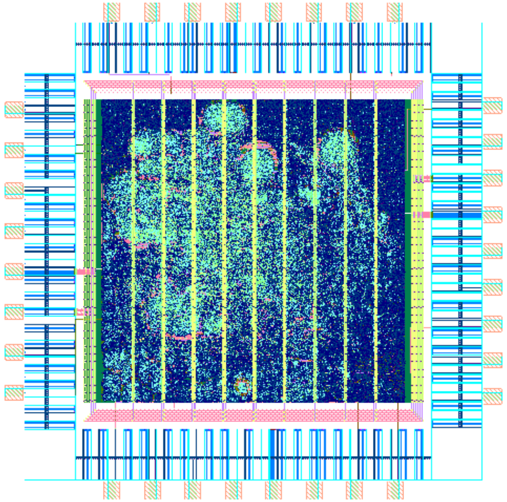
FOUR TARGET CHIPS



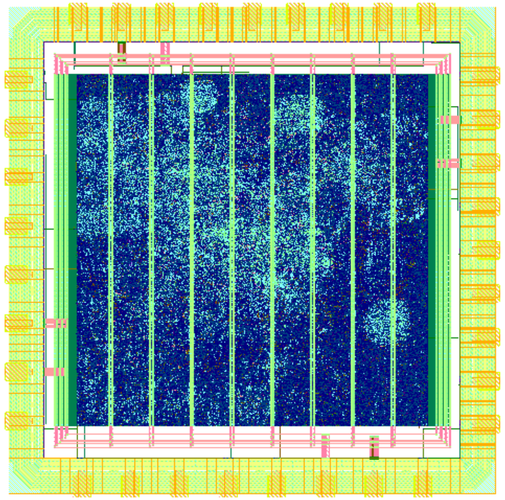
a) 90nm IC



b) 65nm IC

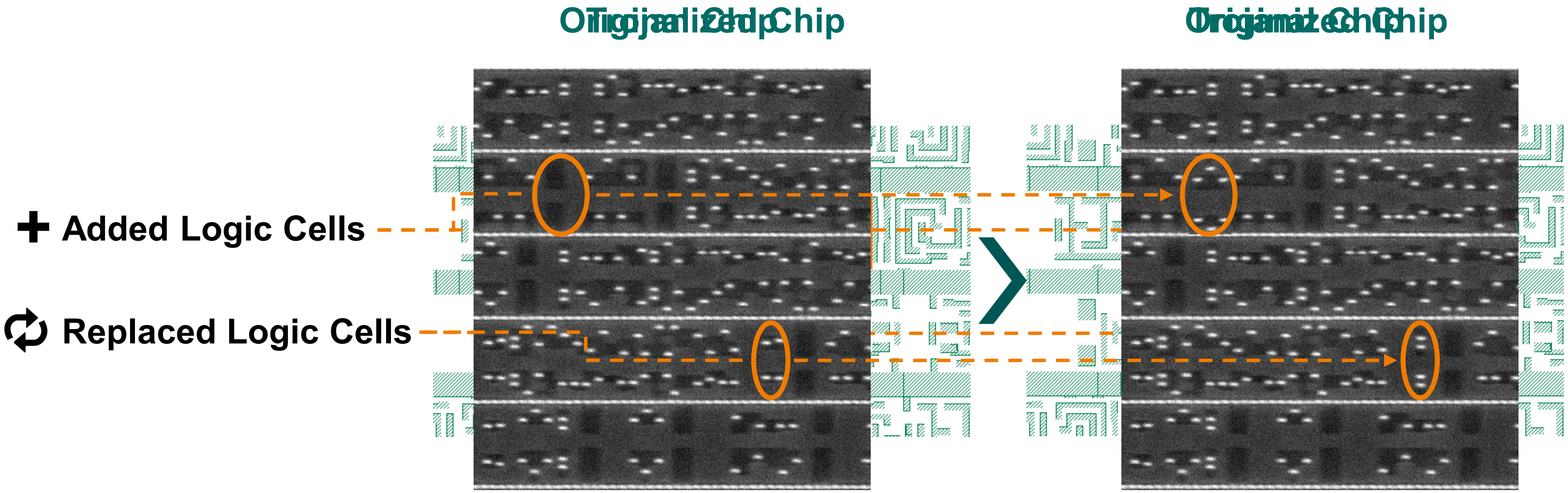


c) 40nm IC



d) 28nm IC

EMULATE INSERTED TROJAN

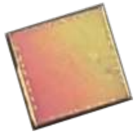


- RED TEAM introduces ten modifications per chip at random positions:
  - 6 x ✚ Added logic cells
  - 4 x 🔁 Replaced logic cells
- 40 of 3,410,580 total cells → 0.001 % modified

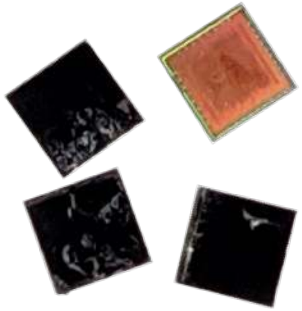
Blue Team: IMAGE THE CHIP

2

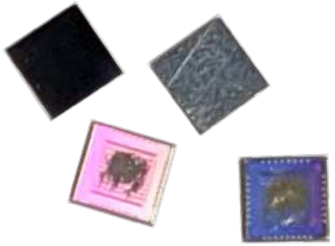
Blue Team receives...



90 nm



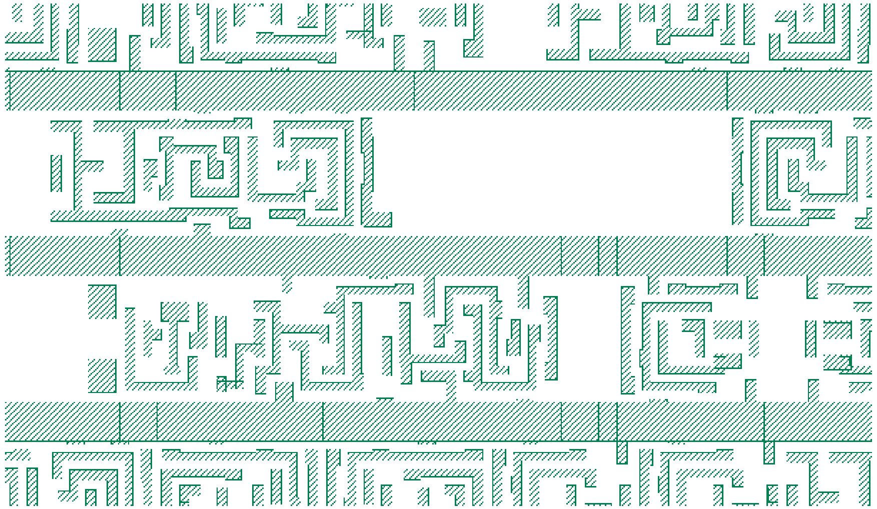
65 nm



40 nm

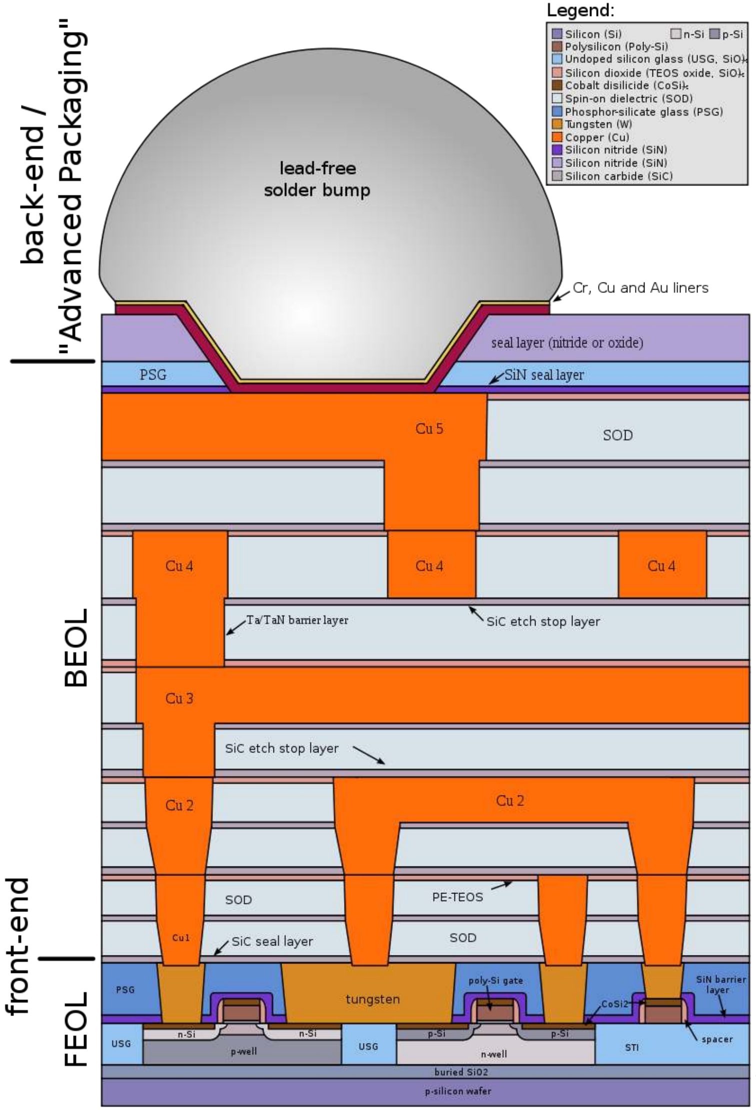
28 nm

a) Physical samples



b) GDS-II Design

INSIDE CHIPS: STACKED LAYERS

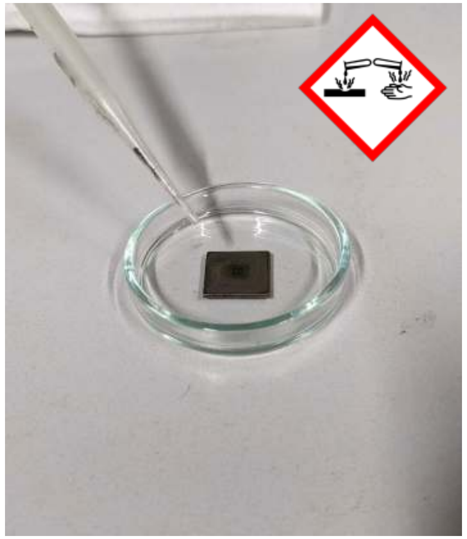


CMOS-chip structure in 2000s (en), Cepheiden, CC BY 2.5, via Wikimedia Commons

Region of interest

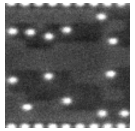
SAMPLE PREPARATION & IMAGING

1. CNC milling



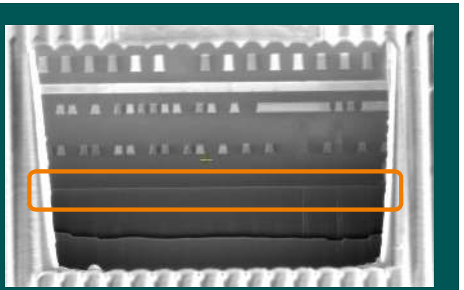
2. Choline hydroxide etching

3. Scanning electron microscope (SEM)

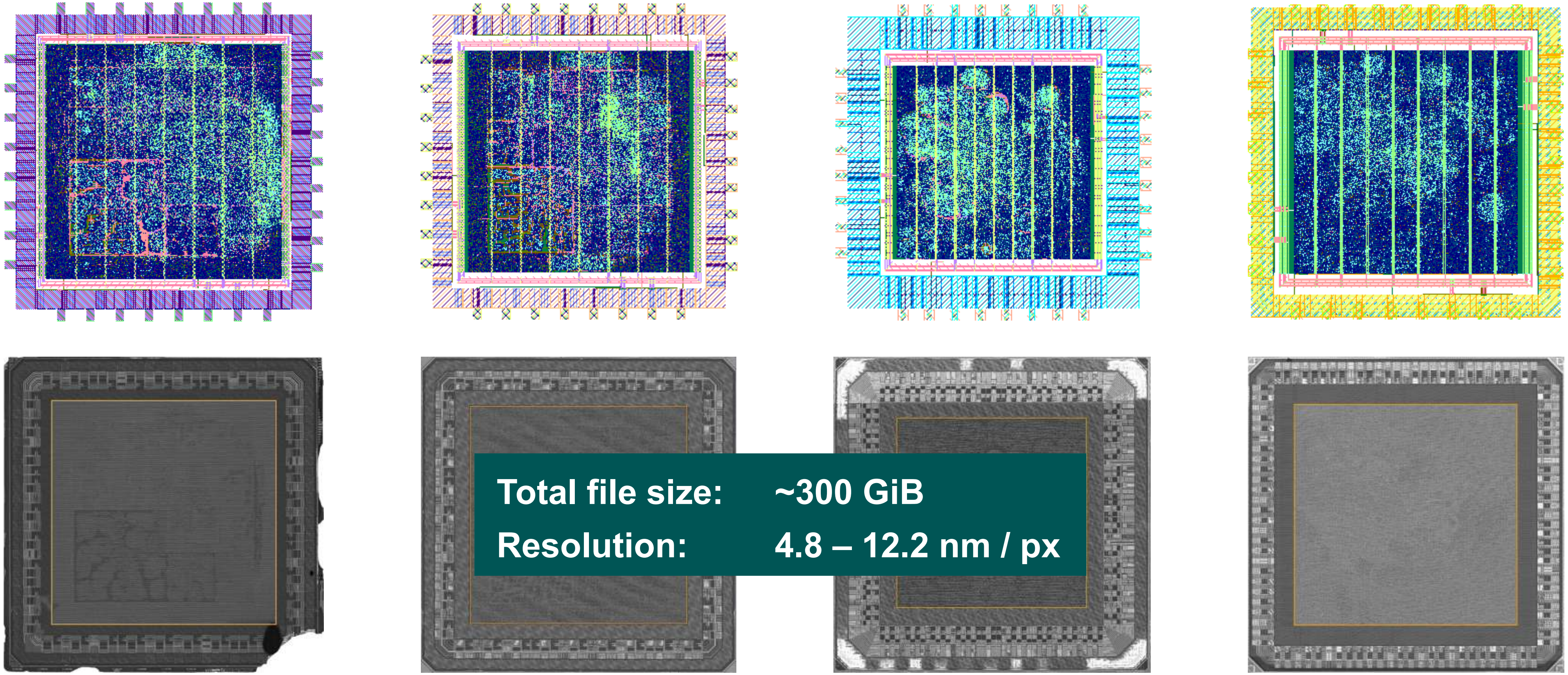


Cross-section Side View

- Cell layout: Bottommost layer
- Removing silicon from the bottom
- Imaging the back side



LAYOUT VS. SEM BACKSIDE IMAGE



Total file size: ~300 GiB

Resolution: 4.8 – 12.2 nm / px

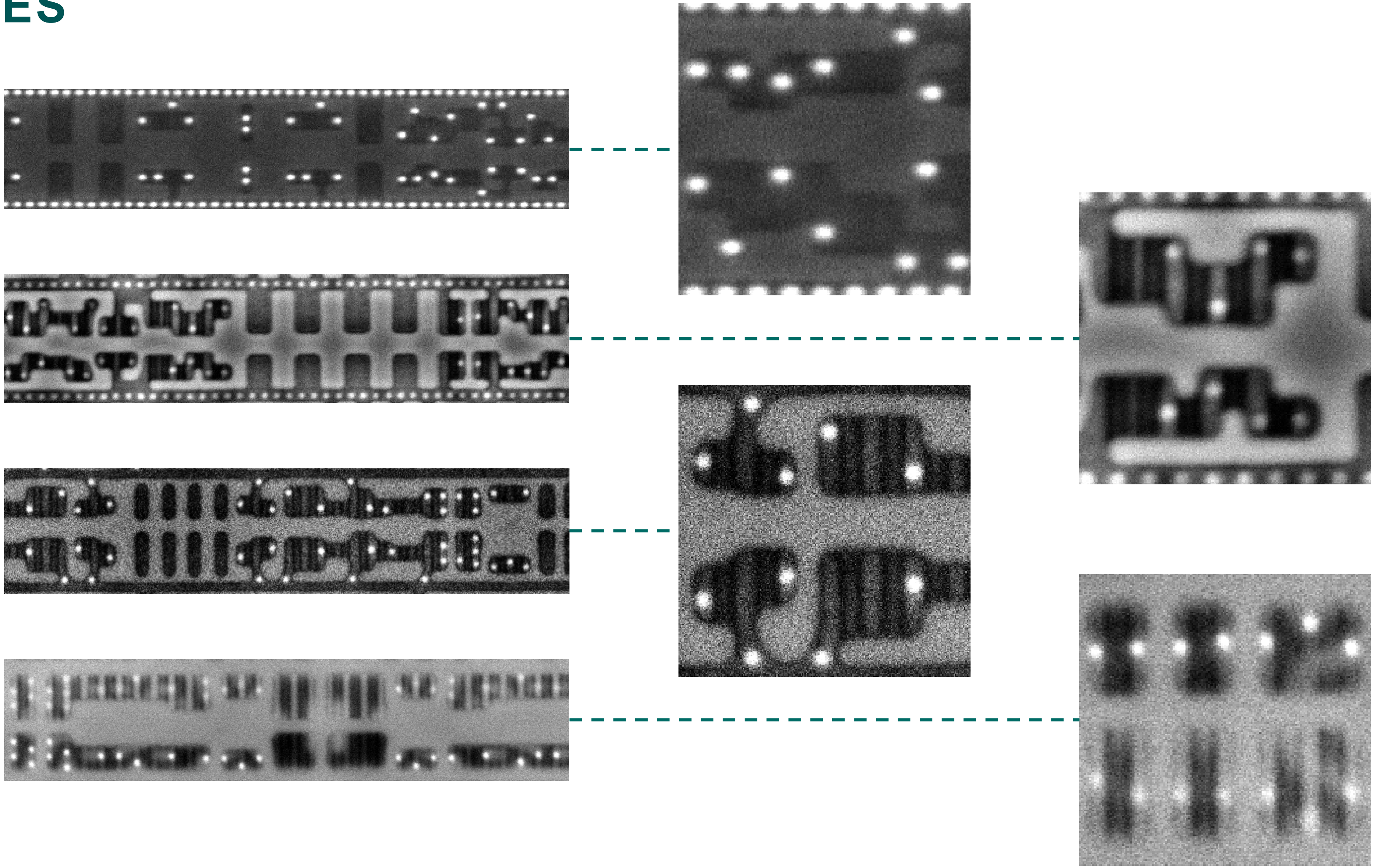
a) 90nm IC

b) 65nm IC

c) 40nm IC

d) 28nm IC

SEM IMAGES



a) 90nm IC

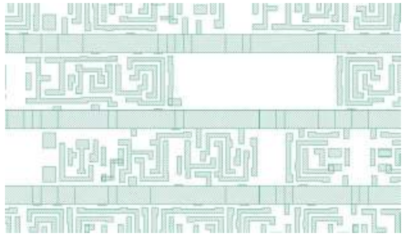
b) 65nm IC

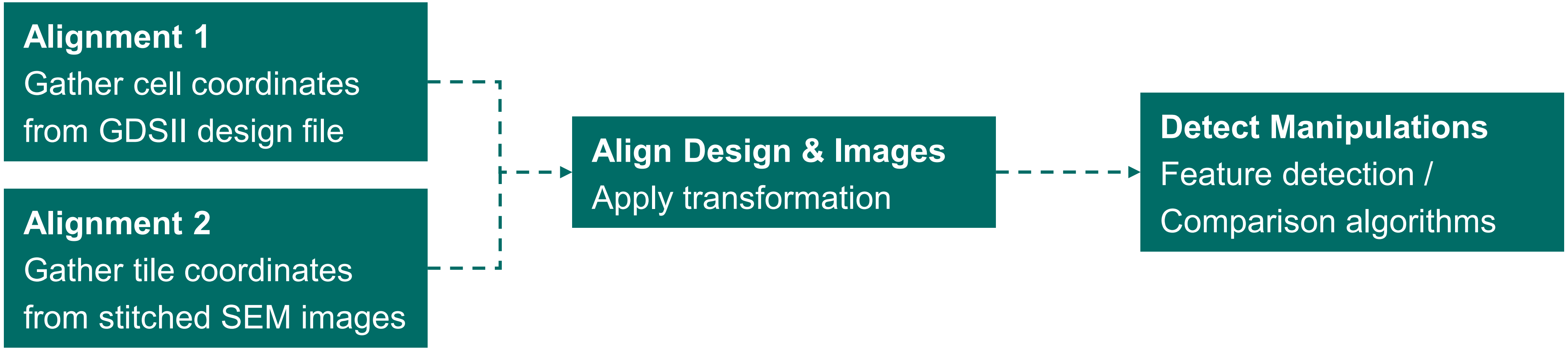
c) 40nm IC

d) 28nm IC

BLUE TEAM: DETECT MANIPULATIONS

ROADMAP



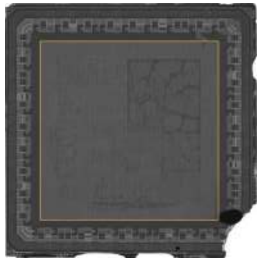


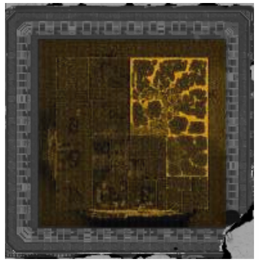
Alignment 1
Gather cell coordinates from GDSII design file

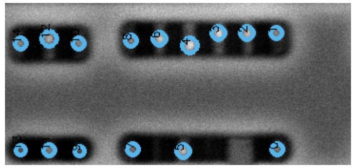
Alignment 2
Gather tile coordinates from stitched SEM images

Align Design & Images
Apply transformation

Detect Manipulations
Feature detection / Comparison algorithms

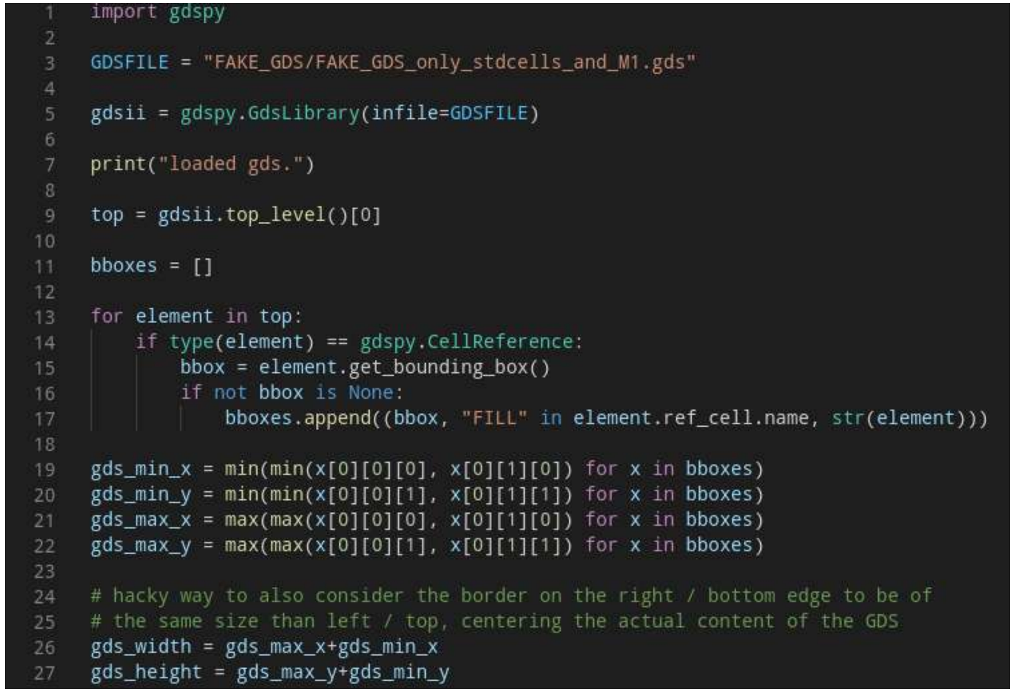




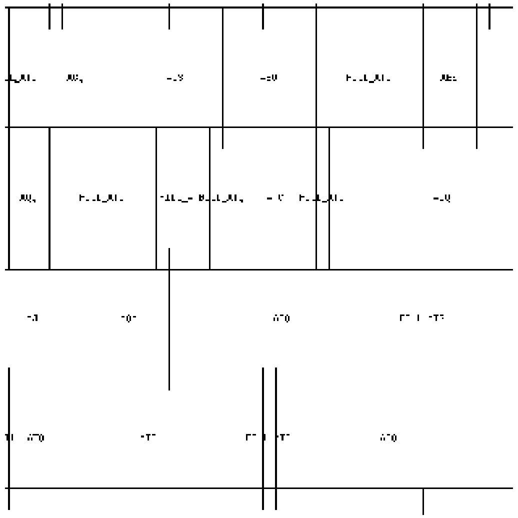


GDSII COORDINATES

- Open Source Library “gdspy” [4]
- Take all “Cell References” that have a bounding box, differentiate if label contains “FILL”



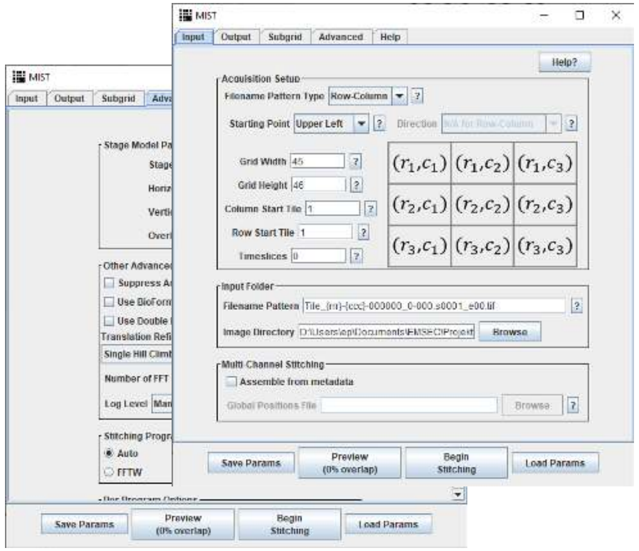
```python
import gdspy

GDSFILE = "FAKE_GDS/FAKE_GDS_only_stdcells_and_M1.gds"

gdsii = gdspy.GdsLibrary(infile=GDSFILE)

print("loaded gds.")

top = gdsii.top_level()[0]

bboxes = []

for element in top:
    if type(element) == gdspy.CellReference:
        bbox = element.get_bounding_box()
        if not bbox is None:
            bboxes.append((bbox, "FILL" in element.ref_cell.name, str(element)))

gds_min_x = min(min(x[0][0][0], x[0][1][0]) for x in bboxes)
gds_min_y = min(min(x[0][0][1], x[0][1][1]) for x in bboxes)
gds_max_x = max(max(x[0][0][0], x[0][1][0]) for x in bboxes)
gds_max_y = max(max(x[0][0][1], x[0][1][1]) for x in bboxes)

# hacky way to also consider the border on the right / bottom edge to be of
# the same size than left / top, centering the actual content of the GDS
gds_width = gds_max_x+gds_min_x
gds_height = gds_max_y+gds_min_y
```



[4] https://github.com/heitzmann/gdspy

TILE IMAGE COORDINATES

Stitching done with e.g. MIST [5]



Filename Pattern Type: Row-Column

Starting Point: Upper Left

Grid Width: 45

Grid Height: 46

Column Start Tile: 1

Row Start Tile: 1

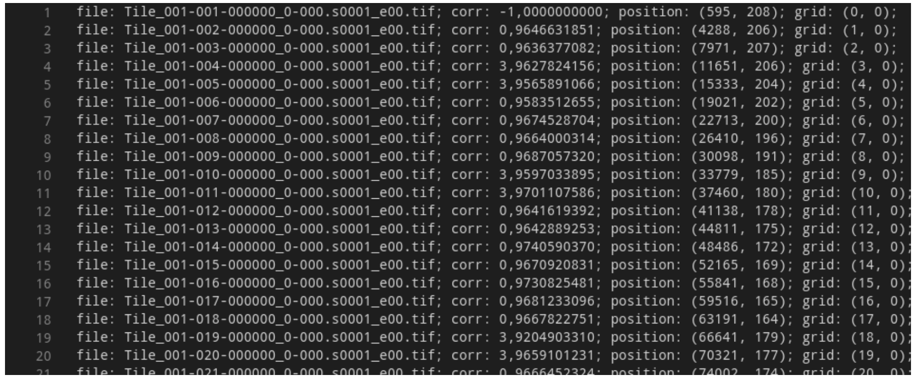
Timeslices: 0

Filename Pattern: Tile_{rrr}-{ccc}-000000_0-000.s0001_e00.tif

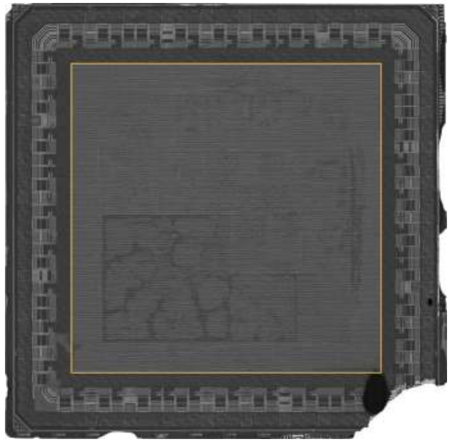
Image Directory: D:\Users\ep\Documents\EMSEC\Project

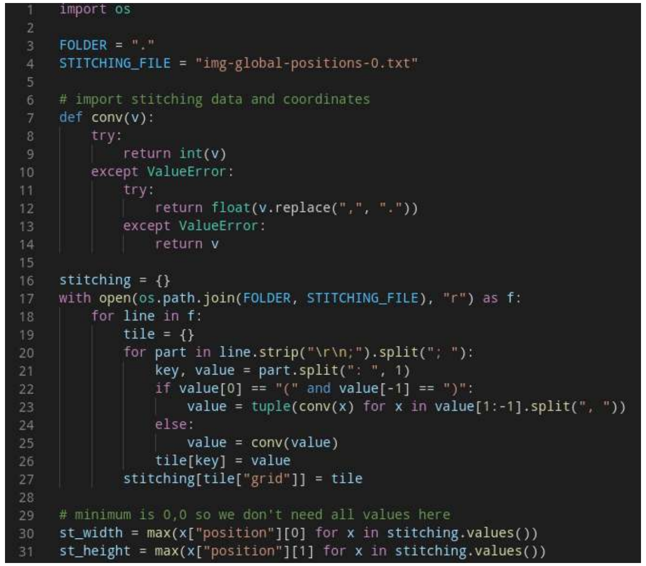
○ Auto

○ FFTW



```
file: Tile_001-001-000000_0-000.s0001_e00.tif; corr: -1,0000000000; position: (595, 208); grid: (0, 0);
file: Tile_001-002-000000_0-000.s0001_e00.tif; corr: 0,9646631851; position: (4288, 206); grid: (1, 0);
file: Tile_001-003-000000_0-000.s0001_e00.tif; corr: 0,9636377082; position: (7971, 207); grid: (2, 0);
file: Tile_001-004-000000_0-000.s0001_e00.tif; corr: 3,9627824156; position: (11651, 206); grid: (3, 0);
file: Tile_001-005-000000_0-000.s0001_e00.tif; corr: 3,9565891066; position: (15333, 204); grid: (4, 0);
file: Tile_001-006-000000_0-000.s0001_e00.tif; corr: 0,9583512655; position: (19021, 202); grid: (5, 0);
file: Tile_001-007-000000_0-000.s0001_e00.tif; corr: 0,9674528704; position: (22713, 200); grid: (6, 0);
file: Tile_001-008-000000_0-000.s0001_e00.tif; corr: 0,9664000314; position: (26410, 196); grid: (7, 0);
file: Tile_001-009-000000_0-000.s0001_e00.tif; corr: 0,9687057320; position: (30098, 191); grid: (8, 0);
file: Tile_001-010-000000_0-000.s0001_e00.tif; corr: 3,9597033895; position: (33779, 185); grid: (9, 0);
file: Tile_001-011-000000_0-000.s0001_e00.tif; corr: 3,9701107586; position: (37460, 180); grid: (10, 0);
file: Tile_001-012-000000_0-000.s0001_e00.tif; corr: 0,9641619392; position: (41138, 178); grid: (11, 0);
file: Tile_001-013-000000_0-000.s0001_e00.tif; corr: 0,9642889253; position: (44811, 175); grid: (12, 0);
file: Tile_001-014-000000_0-000.s0001_e00.tif; corr: 0,9740590370; position: (48486, 172); grid: (13, 0);
file: Tile_001-015-000000_0-000.s0001_e00.tif; corr: 0,9670920831; position: (52165, 169); grid: (14, 0);
file: Tile_001-016-000000_0-000.s0001_e00.tif; corr: 0,9730825481; position: (55841, 168); grid: (15, 0);
file: Tile_001-017-000000_0-000.s0001_e00.tif; corr: 0,9681233096; position: (59516, 165); grid: (16, 0);
file: Tile_001-018-000000_0-000.s0001_e00.tif; corr: 0,9667822751; position: (63191, 164); grid: (17, 0);
file: Tile_001-019-000000_0-000.s0001_e00.tif; corr: 3,9204903310; position: (66641, 179); grid: (18, 0);
file: Tile_001-020-000000_0-000.s0001_e00.tif; corr: 3,9659101231; position: (70321, 177); grid: (19, 0);
file: Tile_001-021-000000_0-000.s0001_e00.tif; corr: 0,9666452324; position: (74002, 174); grid: (20, 0);
```





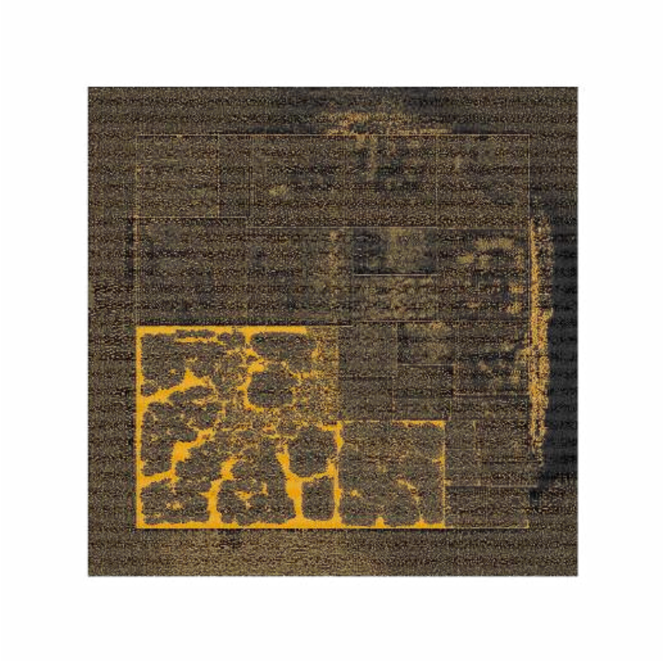
```python
import os

FOLDER = "."
STITCHING_FILE = "img-global-positions-0.txt"

# import stitching data and coordinates
def conv(v):
    try:
        return int(v)
    except ValueError:
        try:
            return float(v.replace(",", "."))
        except ValueError:
            return v

stitching = {}
with open(os.path.join(FOLDER, STITCHING_FILE), "r") as f:
    for line in f:
        tile = {}
        for part in line.strip("\r\n;").split("; "):
            key, value = part.split(": ", 1)
            if value[0] == "(" and value[-1] == ")":
                value = tuple(conv(x) for x in value[1:-1].split(", "))
            else:
                value = conv(value)
            tile[key] = value
        stitching[tile["grid"]] = tile

# minimum is 0,0 so we don't need all values here
st_width = max(x["position"][0] for x in stitching.values())
st_height = max(x["position"][1] for x in stitching.values())
```

[5] J. Chalfoun, M. Majurski, T. Blattner, W. Keyrouz, P. Bajcsy, and M. C. Brady, “MIST: Accurate and Scalable Microscopy Image Stitching Method with Stage Modeling and Error Minimization”, Scientific Reports, vol. 7, no. 1, 2017; see also https://pages.nist.gov/MIST/

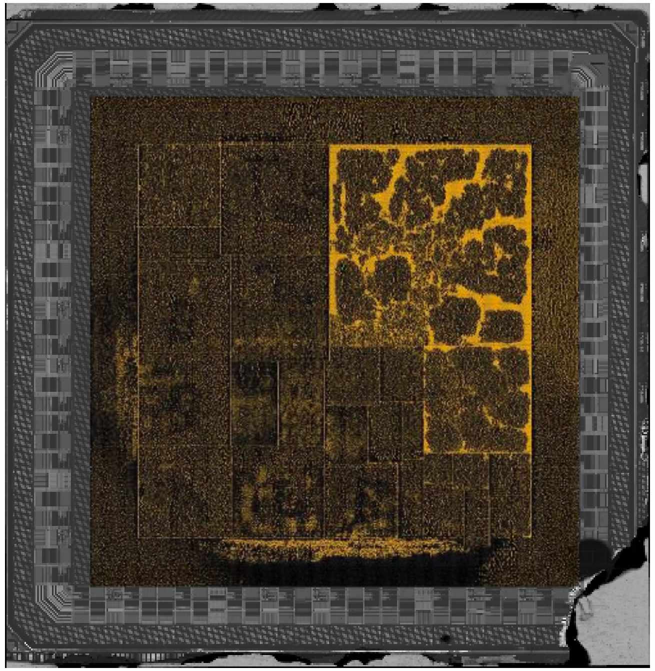
CONVERTING COORDINATES

- First find out about correct rotation / flipping of coordinates (achieved by inverting / swapping axes)
- Hint: Images from the backside are flipped

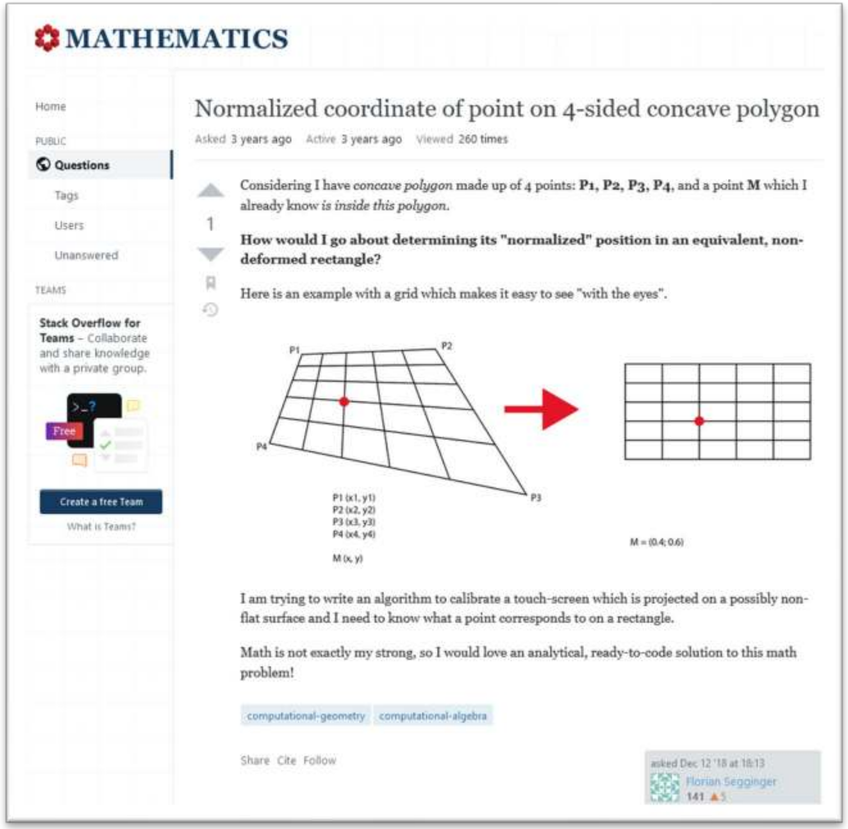
GDSII (orange = filler cell, black = other cell)



Stitched Image (filler cells darker, flipped horizontally and rotated by 90 degrees CCW)



BACK TO SCHOOL MATHS



MATHEMATICS

Home
PUBLIC
Questions
Tags
Users
Unanswered
TEAMS
Stack Overflow for Teams – Collaborate and share knowledge with a private group.
Free
Create a free Team
What is Teams?

Normalized coordinate of point on 4-sided concave polygon

Asked 3 years ago Active 3 years ago Viewed 260 times

1

Considering I have *concave polygon* made up of 4 points: **P1, P2, P3, P4**, and a point **M** which I already know *is inside this polygon.*

How would I go about determining its "normalized" position in an equivalent, non-deformed rectangle?

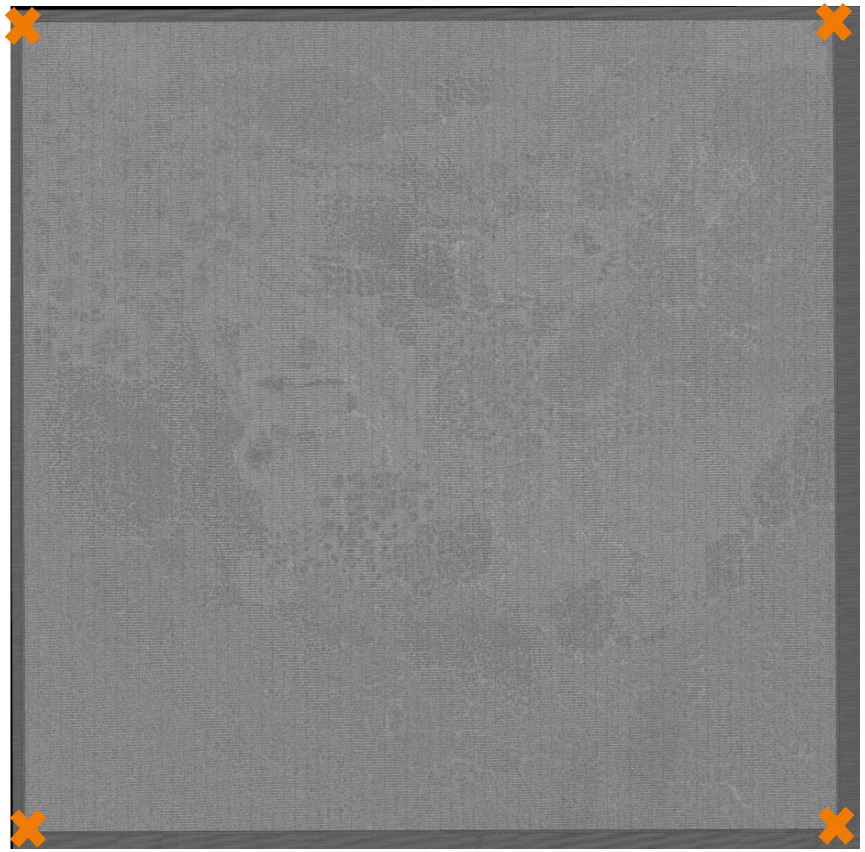
Here is an example with a grid which makes it easy to see "with the eyes".

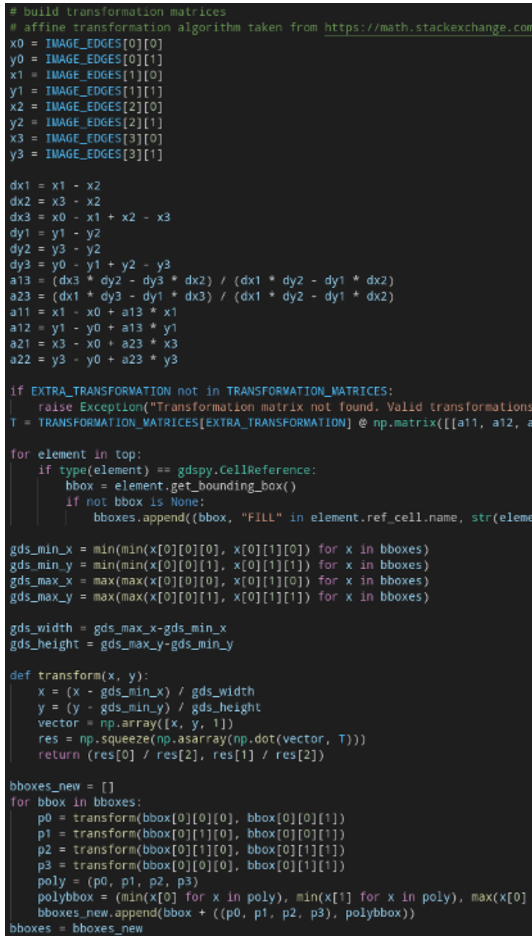
I am trying to write an algorithm to calibrate a touch-screen which is projected on a possibly non-flat surface and I need to know what a point corresponds to on a rectangle.

Math is not exactly my strong, so I would love an analytical, ready-to-code solution to this math problem!

asked Dec 12 '18 at 18:13
Florian Segginger
141 5

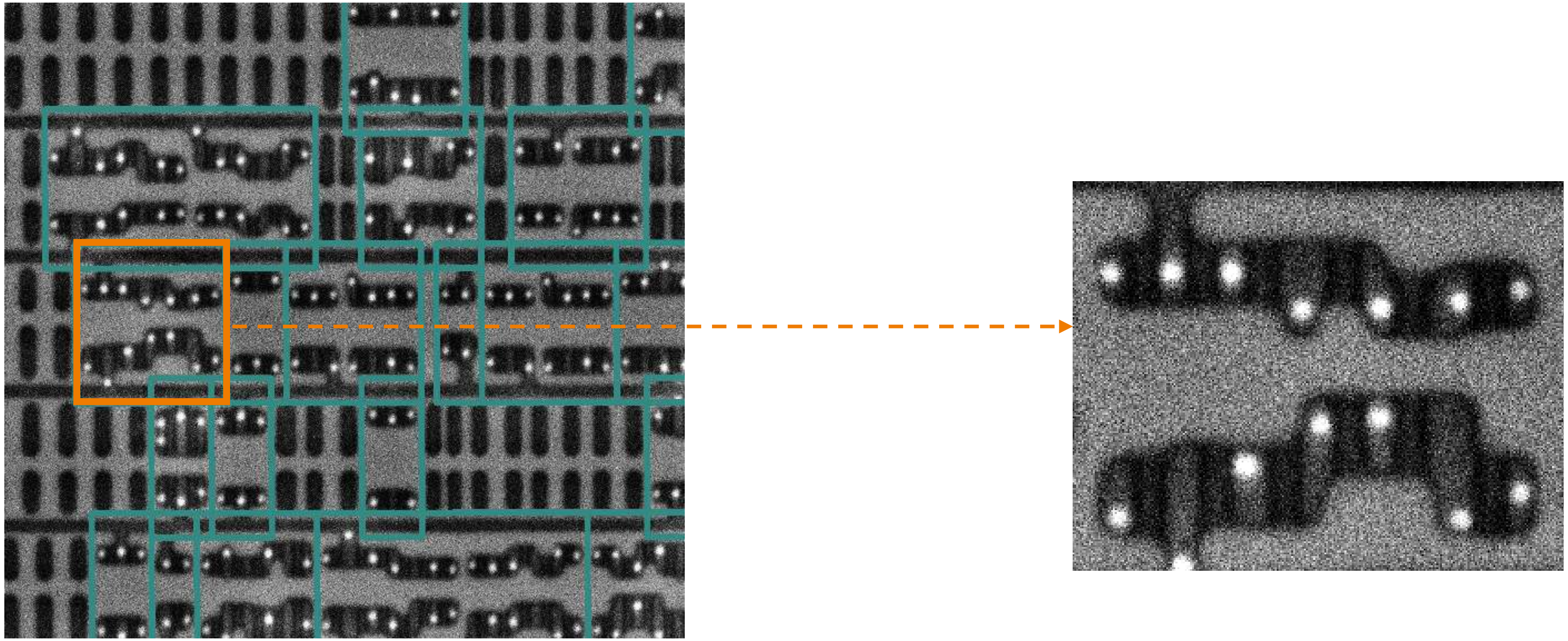
- Static scaling, offset and rotation is not sufficient
- Slight trapezoid (stitching error), thus perspective transformation





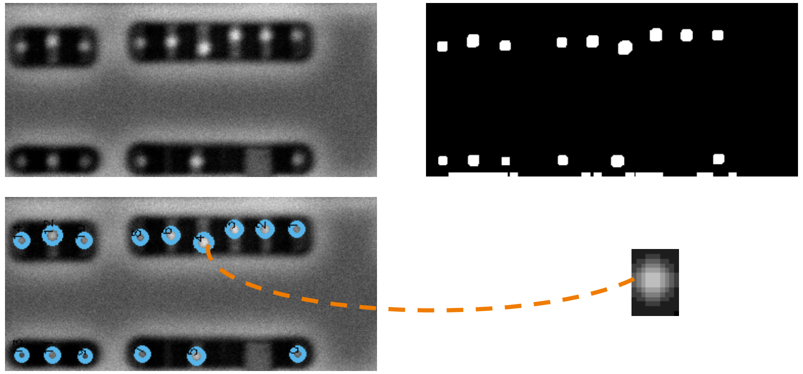
```
# build transformation matrices
# affine transformation algorithm taken from https://math.stackexchange.com
x0 = IMAGE_EDGES[0][0]
y0 = IMAGE_EDGES[0][1]
x1 = IMAGE_EDGES[1][0]
y1 = IMAGE_EDGES[1][1]
x2 = IMAGE_EDGES[2][0]
y2 = IMAGE_EDGES[2][1]
x3 = IMAGE_EDGES[3][0]
y3 = IMAGE_EDGES[3][1]

dx1 = x1 - x2
dx2 = x3 - x2
dx3 = x0 - x1 + x2 - x3
dy1 = y1 - y2
dy2 = y3 - y2
dy3 = y0 - y1 + y2 - y3
a13 = (dx3 * dy2 - dy3 * dx2) / (dx1 * dy2 - dy1 * dx2)
a23 = (dx1 * dy3 - dy1 * dx3) / (dx1 * dy2 - dy1 * dx2)
a11 = x1 - x0 + a13 * x1
a12 = y1 - y0 + a13 * y1
a21 = x3 - x0 + a23 * x3
a22 = y3 - y0 + a23 * y3

if EXTRA_TRANSFORMATION not in TRANSFORMATION_MATRICES:
    raise Exception("Transformation matrix not found. Valid transformations
T = TRANSFORMATION_MATRICES[EXTRA_TRANSFORMATION] @ np.matrix([[a11, a12, a

for element in top:
    if type(element) == gdspy.CellReference:
        bbox = element.get_bounding_box()
        if not bbox is None:
            bboxes.append((bbox, "FILL" in element.ref_cell.name, str(eleme

gds_min_x = min(min(x[0][0][0], x[0][1][0]) for x in bboxes)
gds_min_y = min(min(x[0][0][1], x[0][1][1]) for x in bboxes)
gds_max_x = max(max(x[0][0][0], x[0][1][0]) for x in bboxes)
gds_max_y = max(max(x[0][0][1], x[0][1][1]) for x in bboxes)

gds_width = gds_max_x-gds_min_x
gds_height = gds_max_y-gds_min_y

def transform(x, y):
    x = (x - gds_min_x) / gds_width
    y = (y - gds_min_y) / gds_height
    vector = np.array([x, y, 1])
    res = np.squeeze(np.asarray(np.dot(vector, T)))
    return (res[0] / res[2], res[1] / res[2])

bboxes_new = []
for bbox in bboxes:
    p0 = transform(bbox[0][0][0], bbox[0][0][1])
    p1 = transform(bbox[0][1][0], bbox[0][0][1])
    p2 = transform(bbox[0][1][0], bbox[0][1][1])
    p3 = transform(bbox[0][0][0], bbox[0][1][1])
    poly = (p0, p1, p2, p3)
    polybbox = (min(x[0] for x in poly), min(x[1] for x in poly), max(x[0]
    bboxes_new.append(bbox + ((p0, p1, p2, p3), polybbox))
bboxes = bboxes_new
```

CELLS ARE CUT-OUT

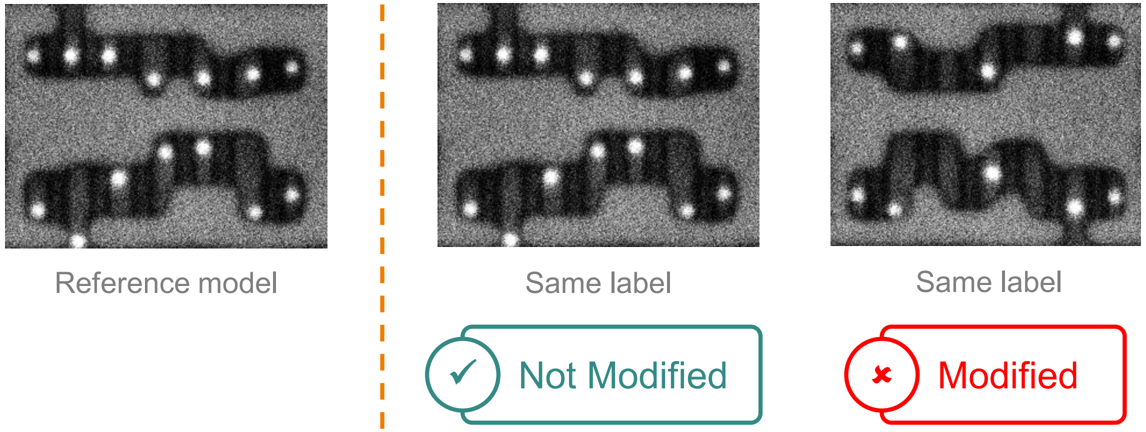


DECISION ALGORITHMS

✚ Additional Cells: Via Detection



Replaced Cells: Template Matching



Reference model

Same label

✓ Not Modified

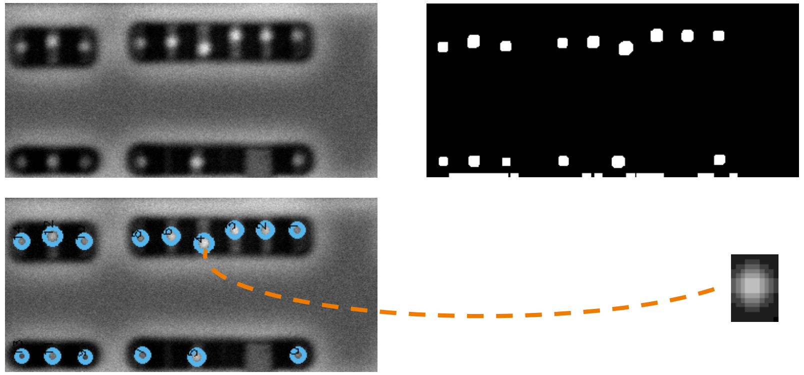
Same label

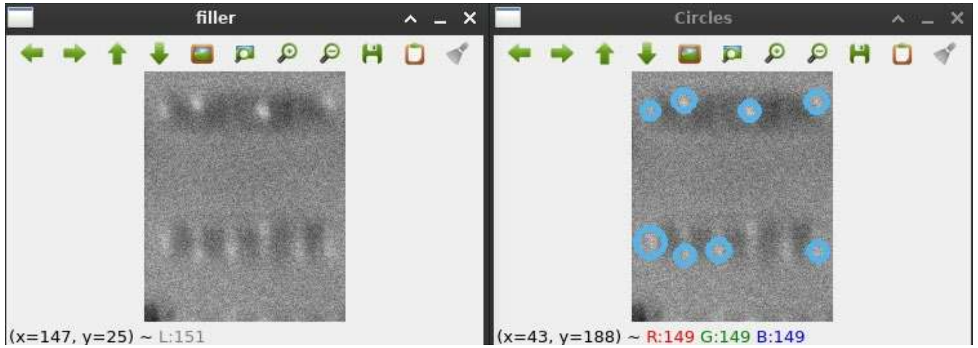
✖ Modified

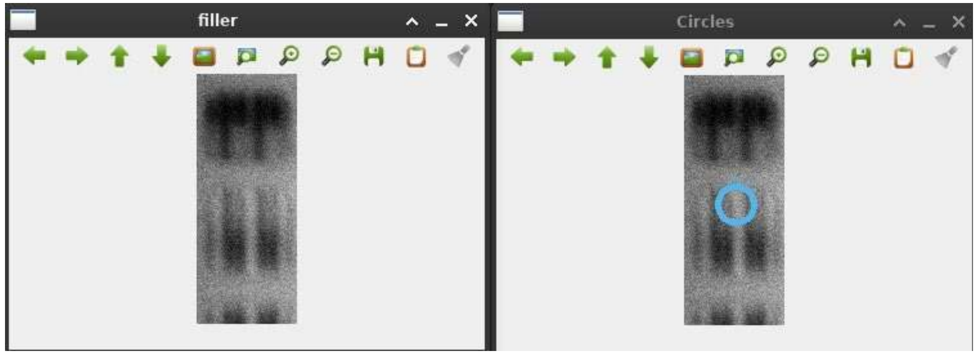
FINDING LOGIC CELLS WHERE FILLER CELLS ARE EXPECTED

Cells contain vias, let’s detect them

- Approach:
  - Suppress noise, threshold, find spots of defined size
  - Verify that they have enough variance (=contrast)
  - Also build a gradient and correlate (corr > x = via)





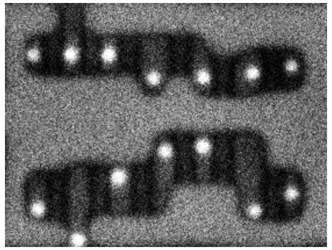


In the end, depends on image quality and parameters

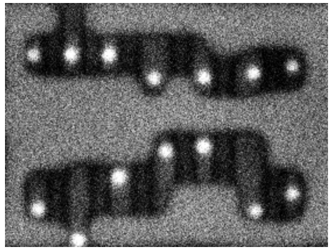
→ some false positives

FINDING CELL REPLACEMENTS

- Q: “Is the cell in question the one it is labeled, or was it replaced by another cell?” (e.g., NAND → NOR)

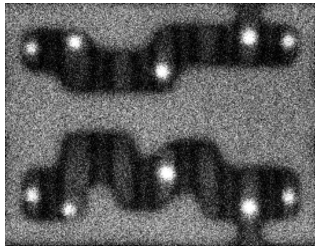


Golden Model / Template



Other Cell labeled same

✓ Not Modified



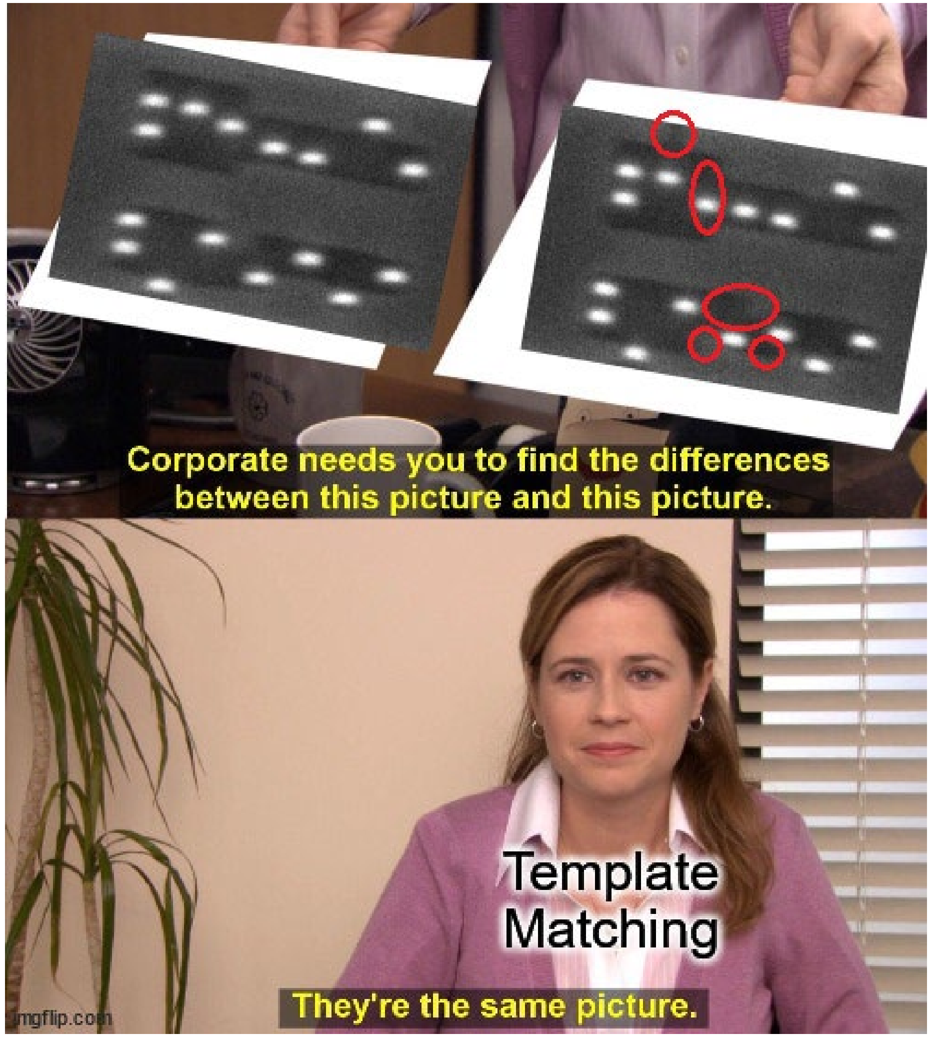
Other Cell labeled same

✗ Modified

→ Template Matching

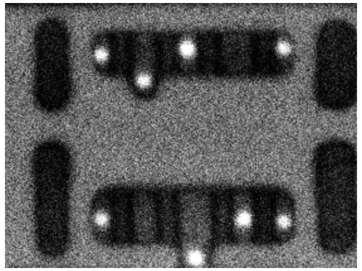
- Use a golden reference model to compare each cell of the same type (= label) against
- If different, count as candidate for modification

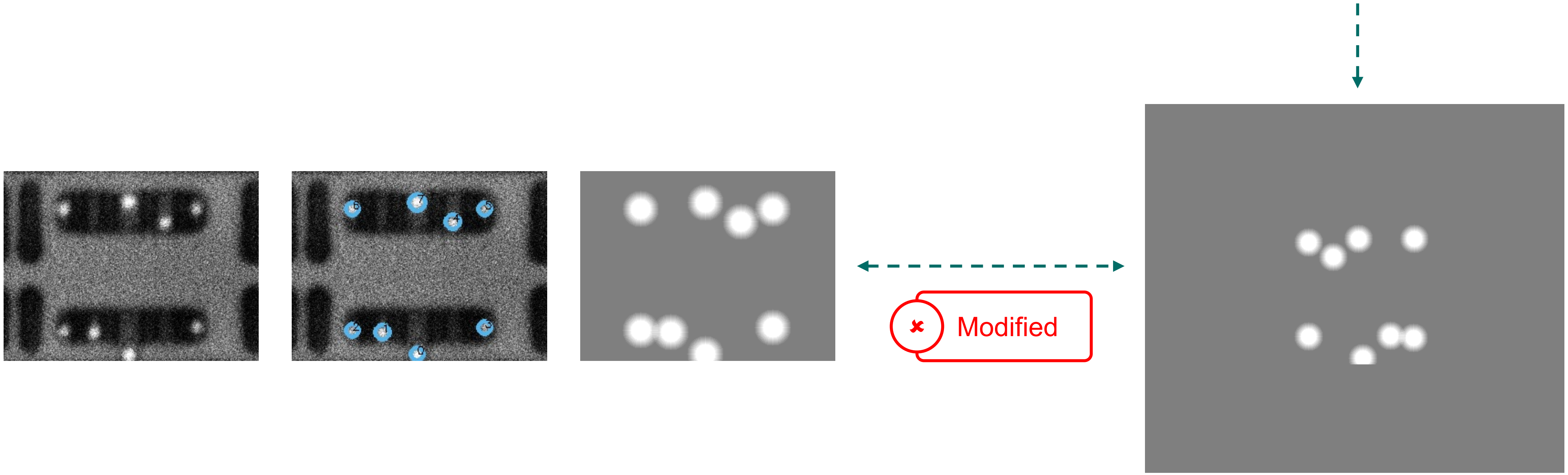
TEMPLATE MATCHING VS. SIMILAR CELLS


Corporate needs you to find the differences between this picture and this picture.
Template Matching
They're the same picture.
imgflip.com

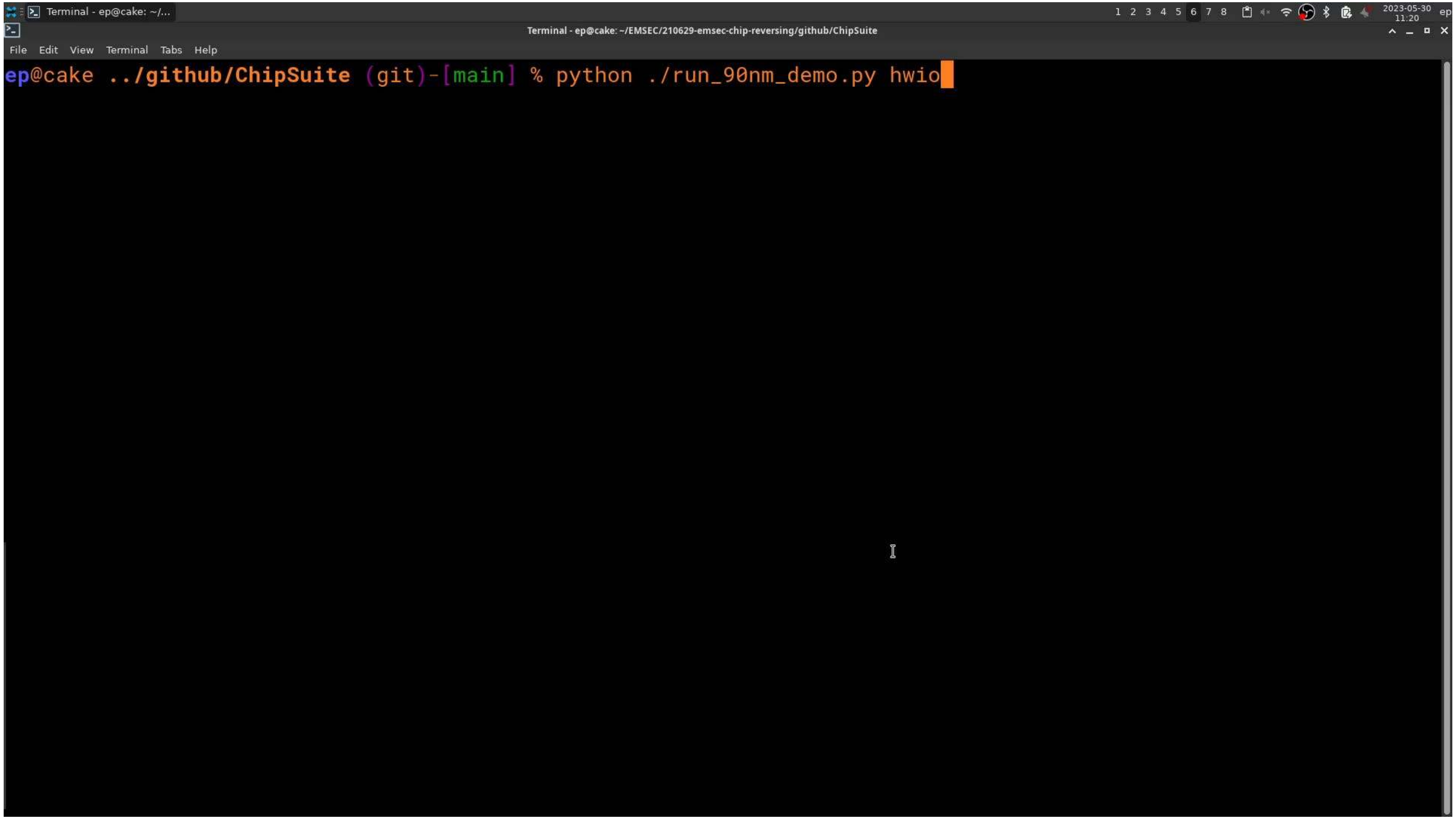
EXTENDED TEMPLATE MATCHING

- Use the via detector to generate a mask out of all via on the cell
- Then do template matching with “golden reference” via mask





LIVE DEMO



Terminal - ep@cake: ~/...
1 2 3 4 5 6 7 8
2023-05-30 11:20
ep
Terminal - ep@cake: ~/EMSEC/210629-emsec-chip-reversing/github/ChipSuite
File Edit View Terminal Tabs Help

```
ep@cake ../github/ChipSuite (git)-[main] % python ./run_90nm_demo.py hwio
```

RESULTS

4

DETECTION RESULTS

Chip Statistics

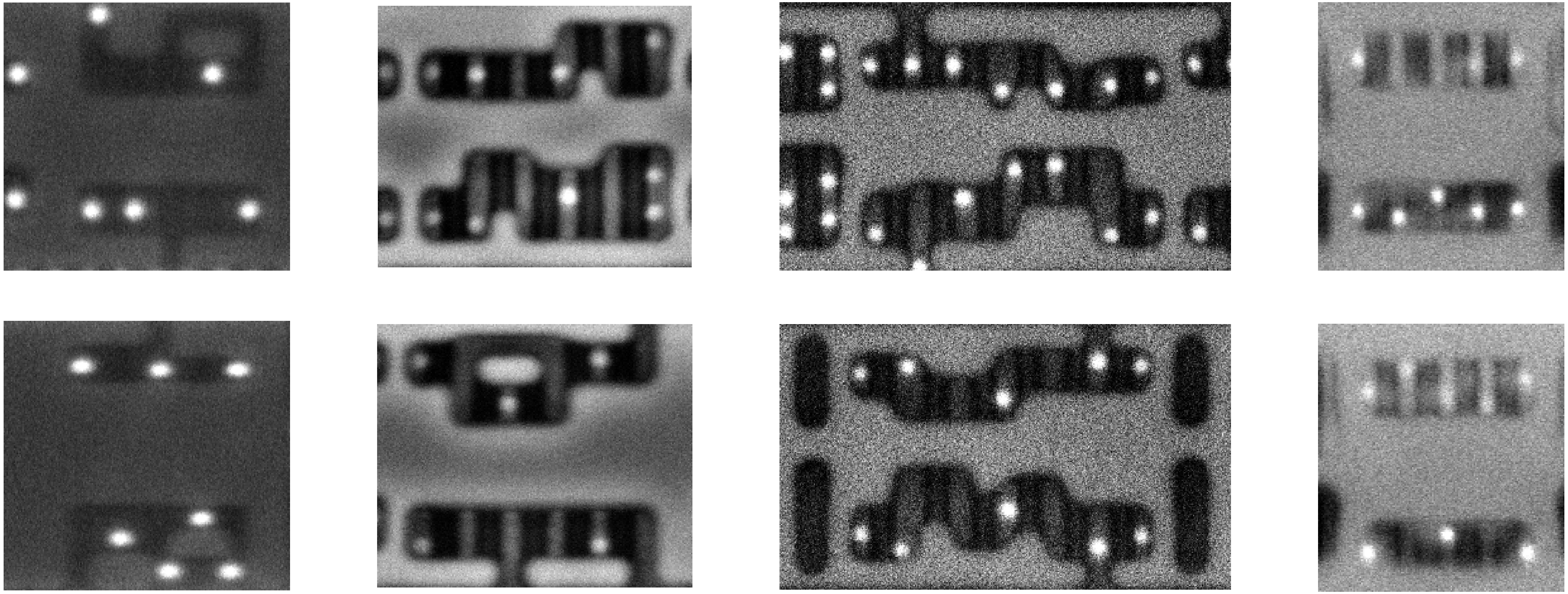
	90 nm	65 nm	40 nm	28 nm
Total Number of Cells	453,850	571,060	917,819	1,467,851
Region of Interest Area	2.089 mm²	1.848 mm²	1.052 mm²	0.962 mm²

+ Additional Cells	4 / 4 ✓	4 / 4 ✓	4 / 4 ✓	4 / 4 ✓
False Positives	0	0	4	17
↻ Replaced Cells	6 / 6 ✓	6 / 6 ✓	6 / 6 ✓	3 / 6 ✗
False Positives	136	6	11	343

Runtime Effort

Image Acquisition (SEM)	~34 h	~23 h	~53 h	~36 h
Detection Algorithms	~2 h	~3 h	~5 h	~4 h

SELECTED TRUE POSITIVES (🔄 REPLACEMENTS)



a) 90nm

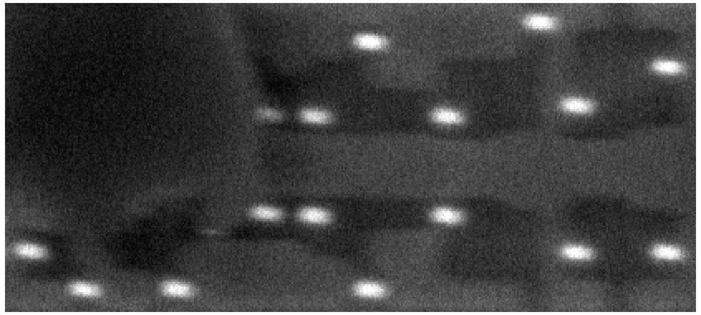
b) 65nm

c) 40nm

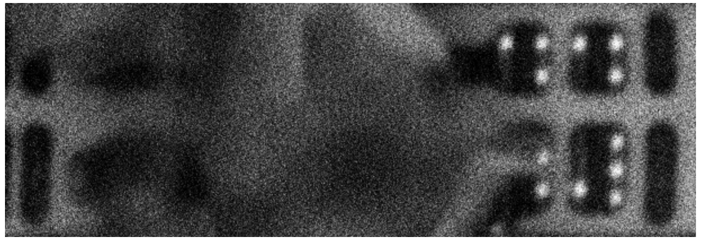
d) 28nm

SELECTED FALSE POSITIVES CAUSED BY DEBRIS OR DUST

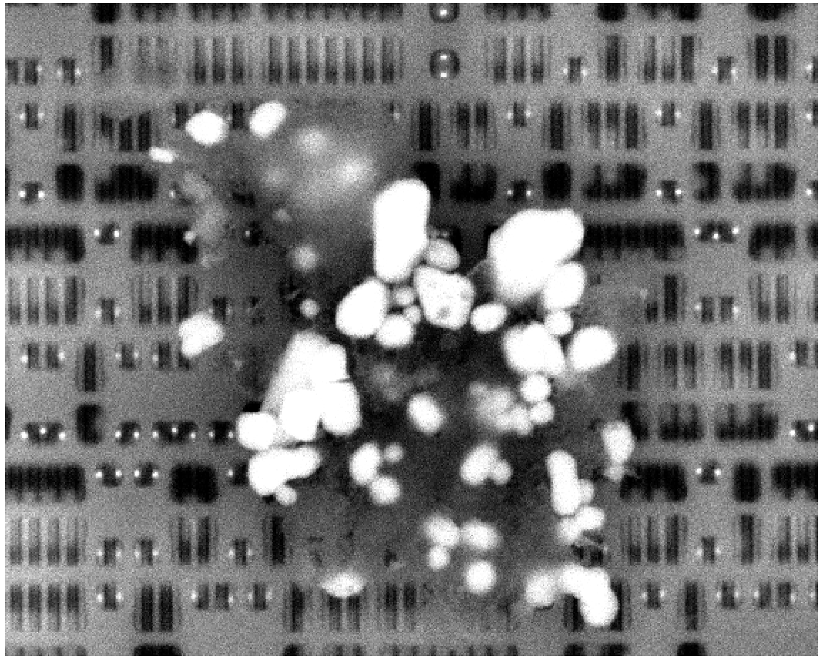
- This affects 161 out of 3.4 million cells in total (0.005%)



a) 90nm example

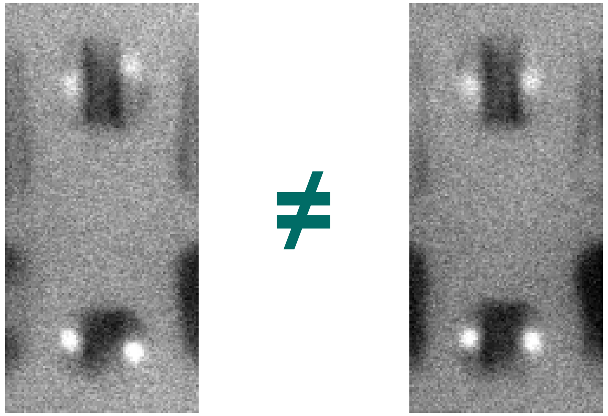


b) 40nm example



c) 28nm example

EXAMPLE OF FALSE NEGATIVE ON 28NM CHIP



- Can we do better?
  - Acquire better images (e.g., with less noise) using a more advanced SEM environment
  - Build algorithms (e.g., involving ML) that can deal with low quality images

TAKEAWAYS

- Easier to integrate (i.e., ✚ Additional Cells) hardware Trojans → less difficult to detect
- 🔁 Replaced functional cells more unobtrusive and harder to detect with shrinking technology sizes
- Detection can likely be improved by advanced detection algorithms and SEM setups
- Sufficient image quality → Detection feasible with high accuracy
  → Scalable to large ICs

- Publication of chip images, (reduced) design files, and detection algorithms:
  - ↘ *Images / Design Files:* *https://doi.org/10.17617/3.396Q7I*
  - ↘ *Code:* *https://github.com/emsec/ChipSuite*

AGENDA

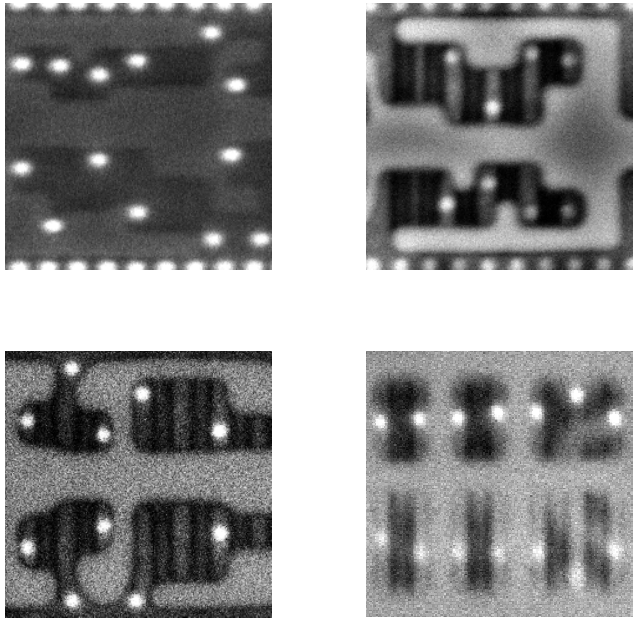
Reverse Engineering: A Human-Centred Perspective

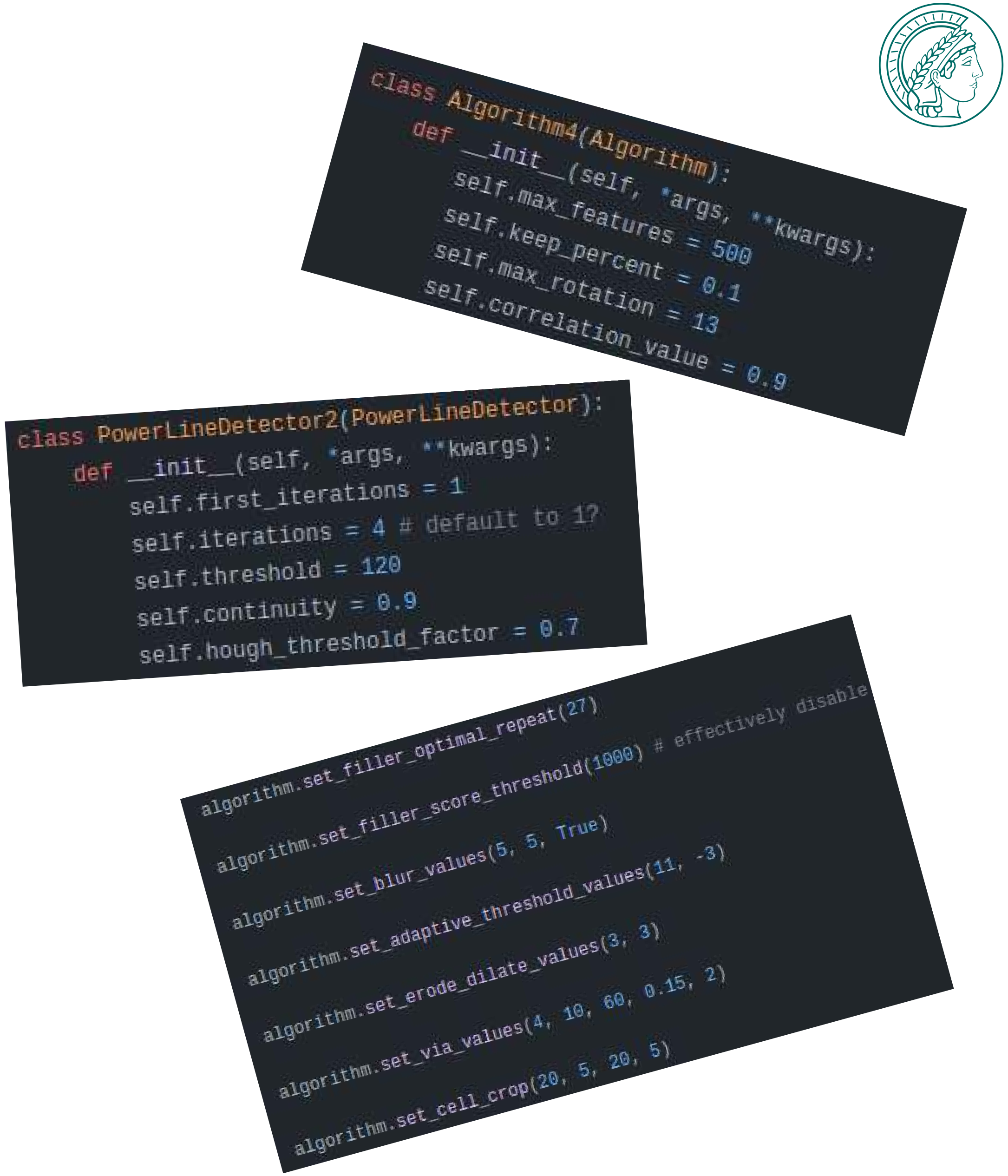
The Community View

LOTS OF PARAMETERS …

… are required because:

- Sample preparation affects image quality
- Imaging conditions affect contrast & brightness





```
class Algorithm4(Algorithm):
    def __init__(self, *args, **kwargs):
        self.max_features = 500
        self.keep_percent = 0.1
        self.max_rotation = 13
        self.correlation_value = 0.9
```

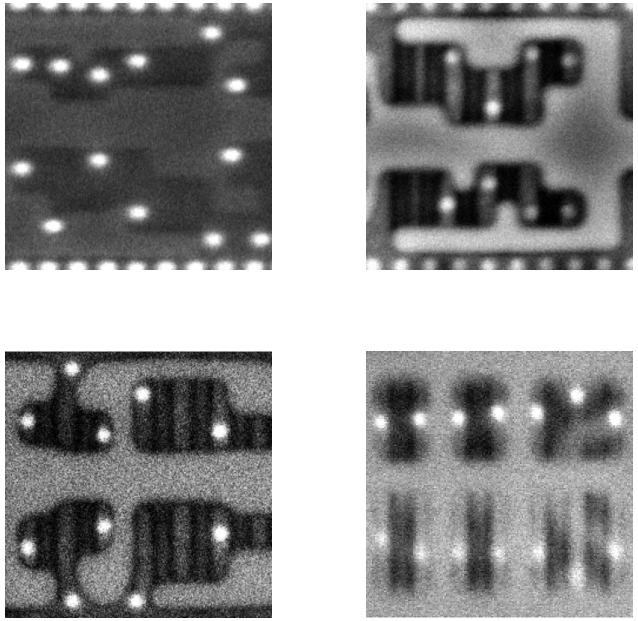
```
class PowerLineDetector2(PowerLineDetector):
    def __init__(self, *args, **kwargs):
        self.first_iterations = 1
        self.iterations = 4 # default to 1?
        self.threshold = 120
        self.continuity = 0.9
        self.hough_threshold_factor = 0.7
```

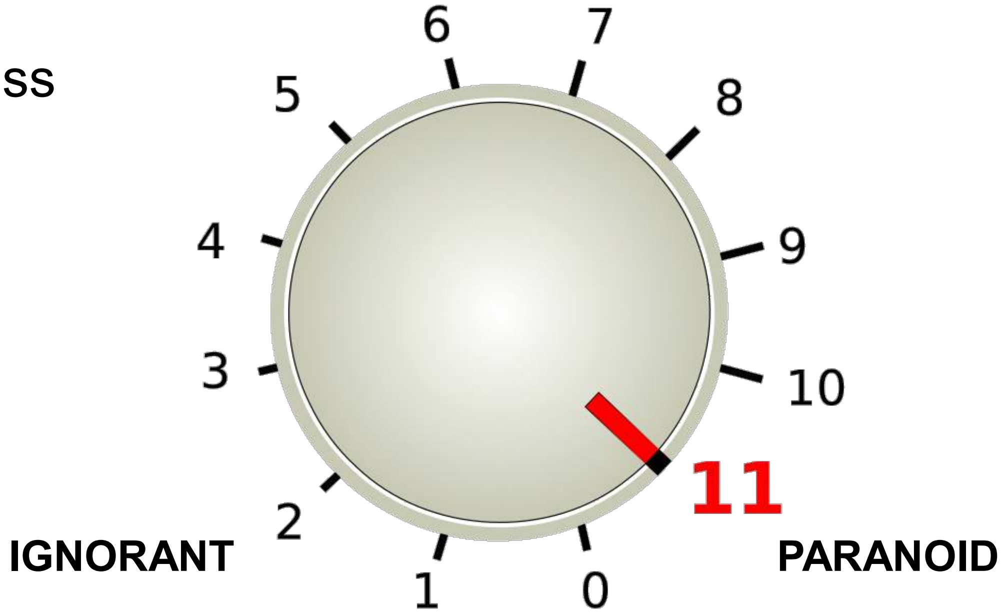
```
algorithm.set_filler_optimal_repeat(27)
algorithm.set_filler_score_threshold(1000) # effectively disable
algorithm.set_blur_values(5, 5, True)
algorithm.set_adaptive_threshold_values(11, -3)
algorithm.set_erode_dilate_values(3, 3)
algorithm.set_via_values(4, 10, 60, 0.15, 2)
algorithm.set_cell_crop(20, 5, 20, 5)
```

LOTS OF PARAMETERS …

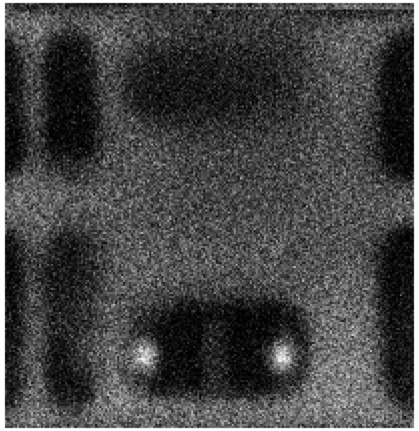
… are required because:

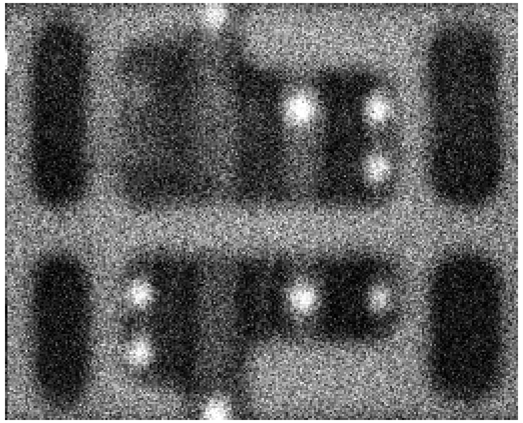
- Sample preparation affects image quality
- Imaging conditions affect contrast & brightness
- Parameters control detection sensitivity





THINK LIKE A TROJAN DESIGNER





Takeaway:

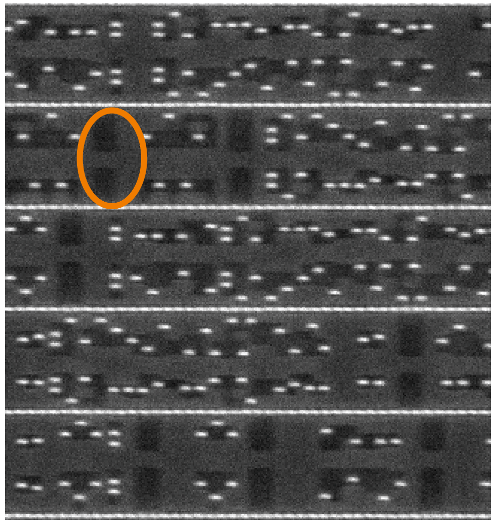
Automation has limits. For a sound result, you *need* experienced analysts & manual investigation.

THINK LIKE A TROJAN DESIGNER

Why would do this?

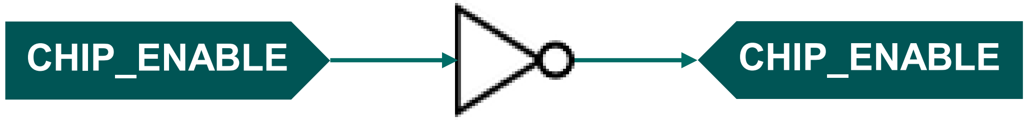
Villain by J.J. at the English-language Wikipedia

THINK LIKE A TROJAN DESIGNER

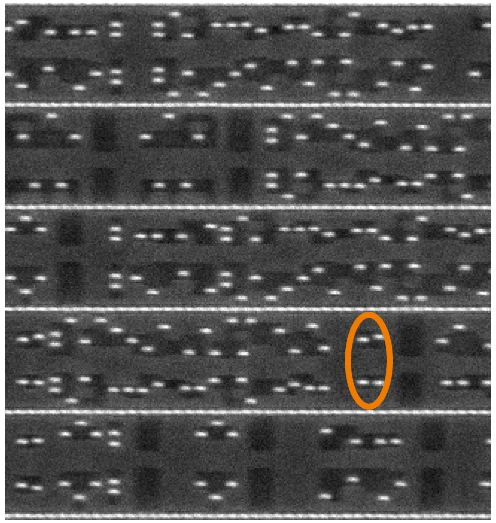


Detected modification: Insertion of NOT cell

Impact: Short Circuit

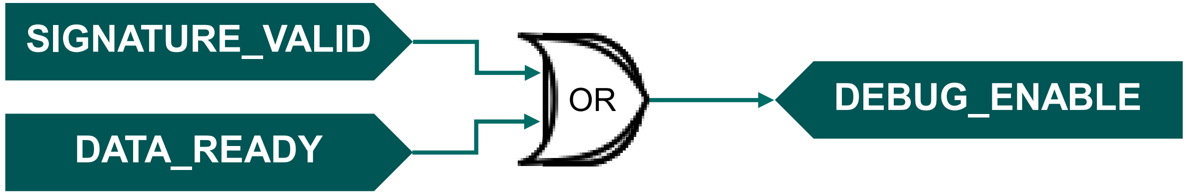


THINK LIKE A TROJAN DESIGNER



Detected modification: Replace AND cell with OR cell

Impact: Debugger authentication bypass

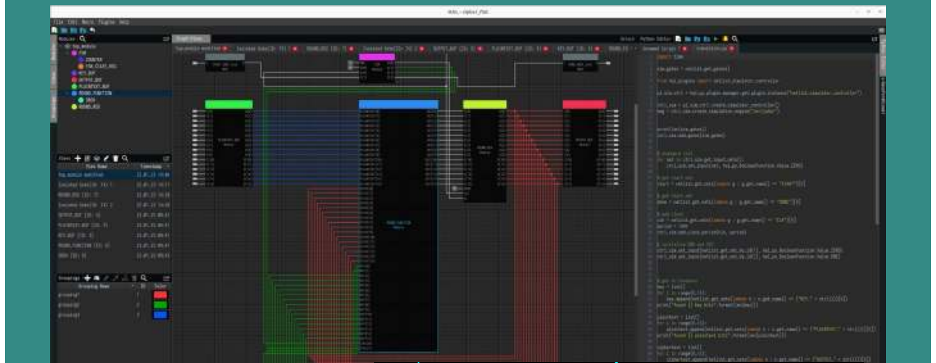


?

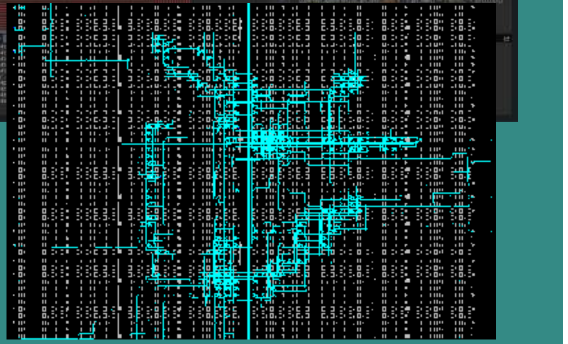
REVERSE ENGINEERING BEYOND IMAGING
From Physical Sample To Logic Diagram

© Steffen Becker 2019

❯



github.com/emsec/hal



© Maik Ender 2020

CHIP-LEVEL & NETLIST REVERSE ENGINEERING
A Short Overview

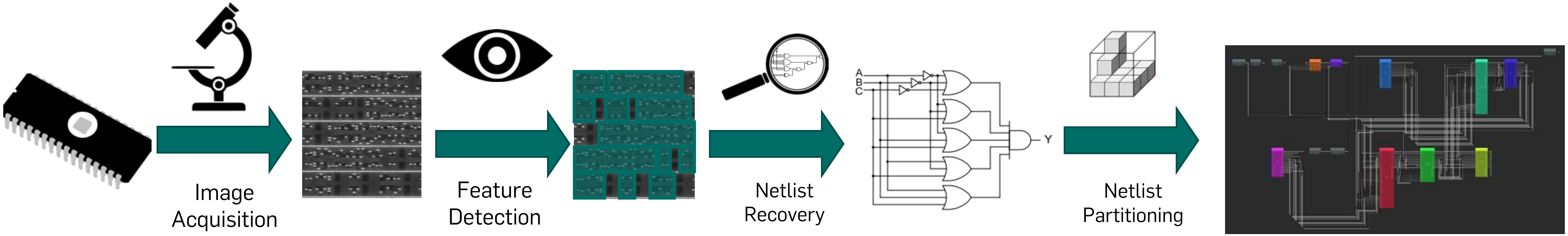
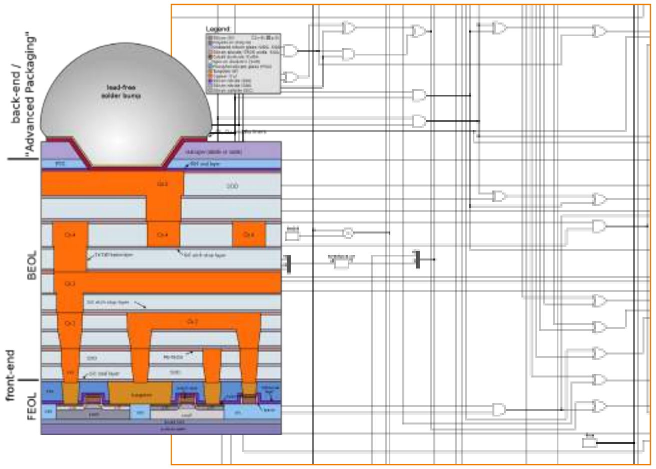


Image Acquisition

Feature Detection

Netlist Recovery

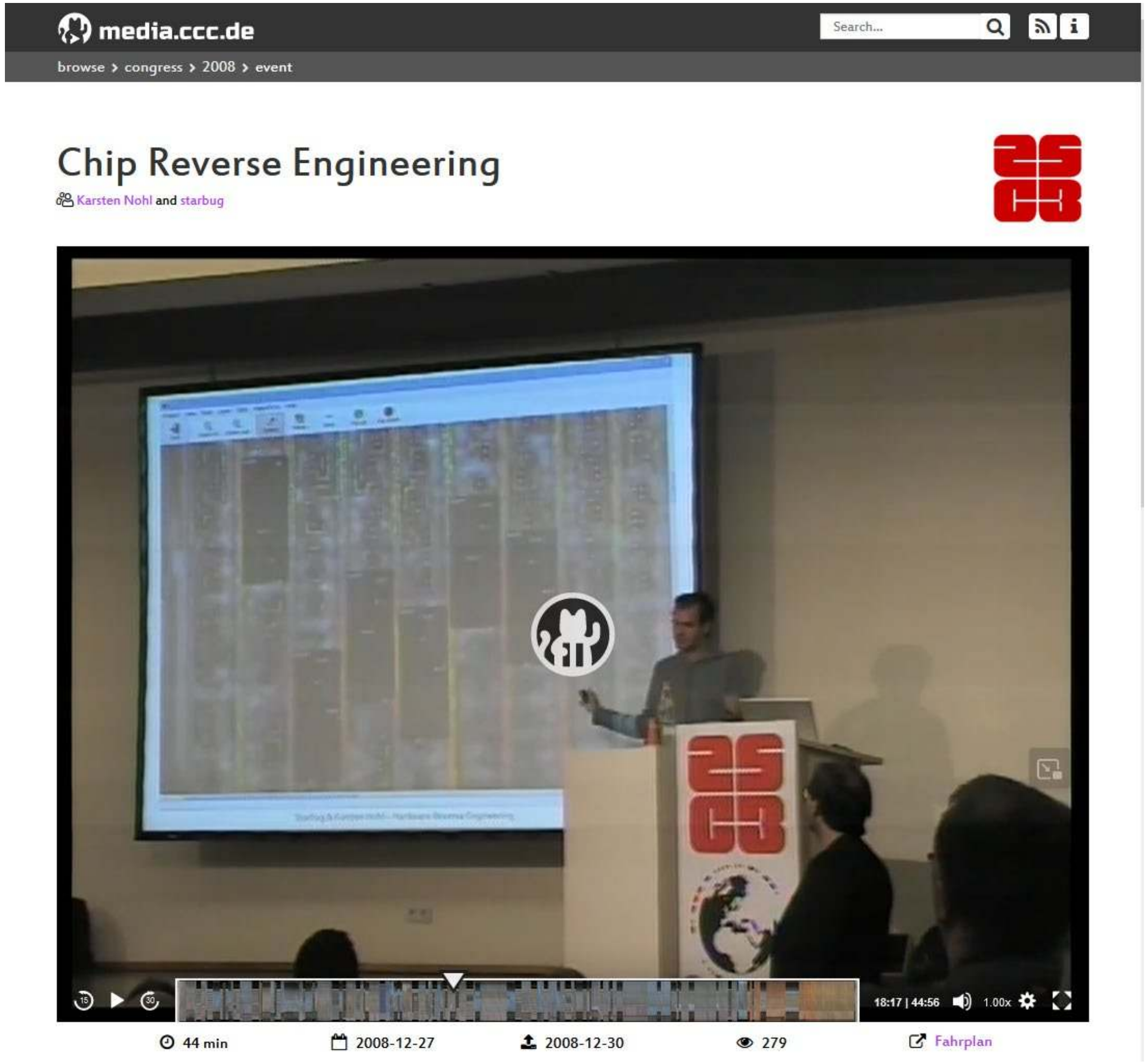
Netlist Partitioning



CHIP REVERSING: OLD BUT GOLD

Reverse Engineering ICs in 28C3 (2008!) talk "Chip Reverse Engineering" by Karsten Nohl and Starbug

URL: https://media.ccc.de/v/25c3-2896-en-chip_reverse_engineering



media.ccc.de

browse › congress › 2008 › event

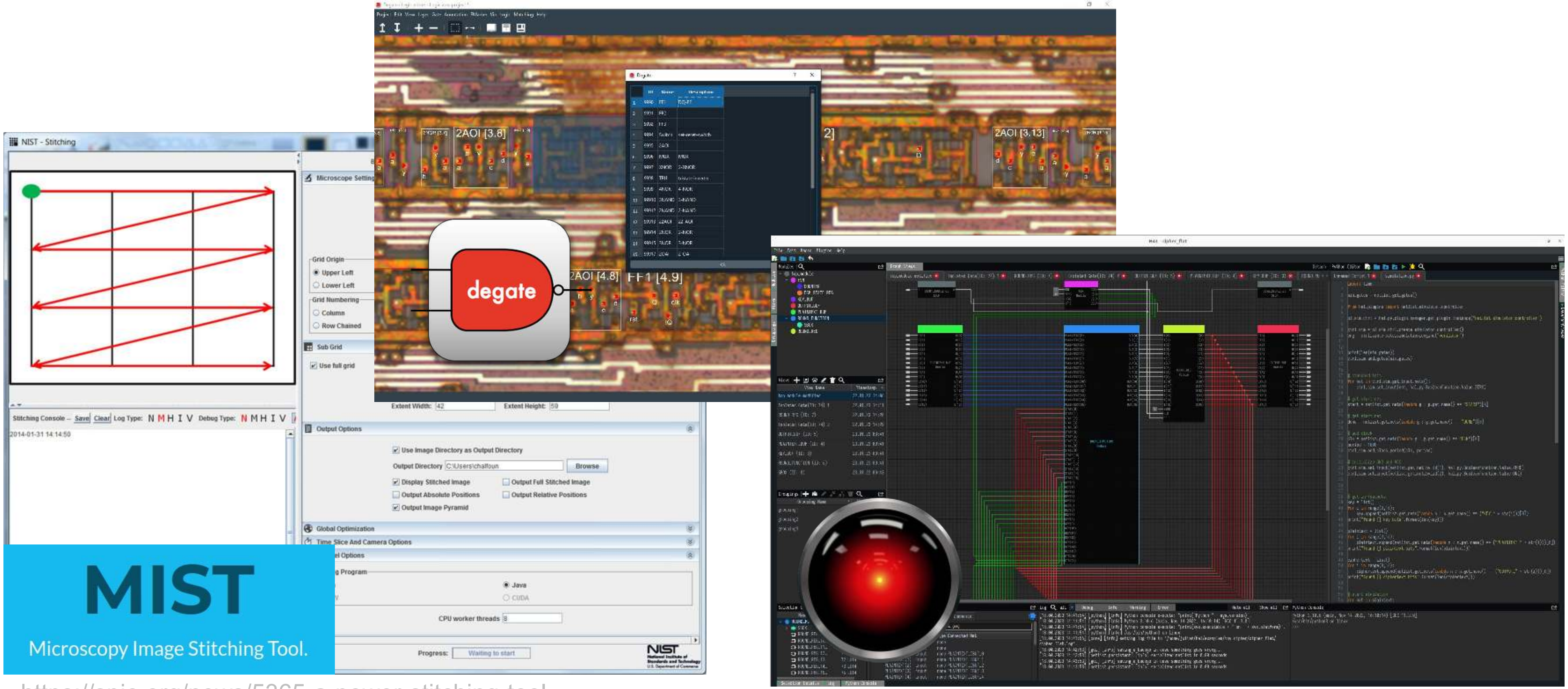
Chip Reverse Engineering

Karsten Nohl and starbug

44 min 2008-12-27 2008-12-30 279 Fahrplan

TOOLS SUPPORT REVERSE ENGINEERING …

https://degate.readthedocs.io


NIST - Stitching
Microscope Setting
2AOI [3.8]
2AOI [3.13]
2AOI [4.8]
FF1 [4.9]
degate
MIST
Microscopy Image Stitching Tool.
NIST National Institute of Standards and Technology U.S. Department of Commerce

Stitching Console – Save Clear Log Type: N M H I V Debug Type: N M H I V

2014-01-31 14:14:50

Grid Origin
● Upper Left
○ Lower Left

Grid Numbering
○ Column
○ Row Chained

Sub Grid
☑ Use full grid

Extent Width: 42 Extent Height: 59

Output Options
☑ Use Image Directory as Output Directory
Output Directory C:\Users\chalfoun Browse
☑ Display Stitched Image □ Output Full Stitched Image
□ Output Absolute Positions □ Output Relative Positions
☑ Output Image Pyramid

Global Optimization

Time Slice And Camera Options

el Options

g Program
● Java
○ CUDA

CPU worker threads 8

Progress: Waiting to start

https://spie.org/news/5365-a-power-stitching-tool

https://github.com/emsec/hal

… BUT HUMANS DRIVE IT

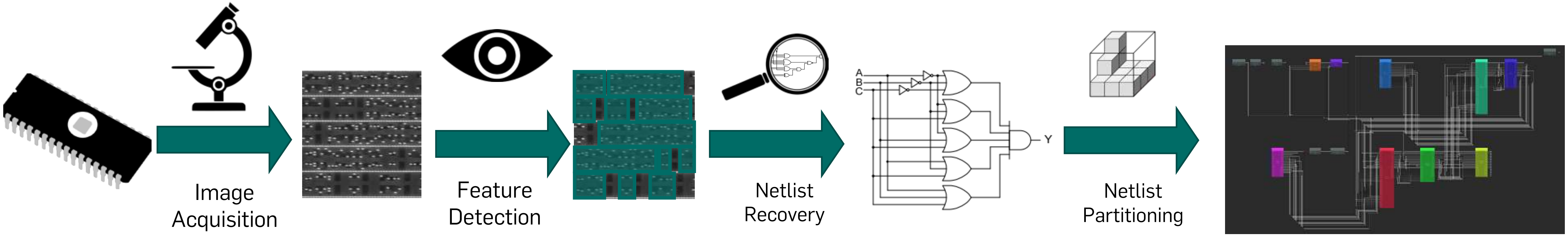


Image Acquisition

Feature Detection

Netlist Recovery

Netlist Partitioning

HARDWARE REVERSE ENGINEERING
A Problem-Solving View

Hardware reverse engineering

=

Manual & semi-automated analyses

+

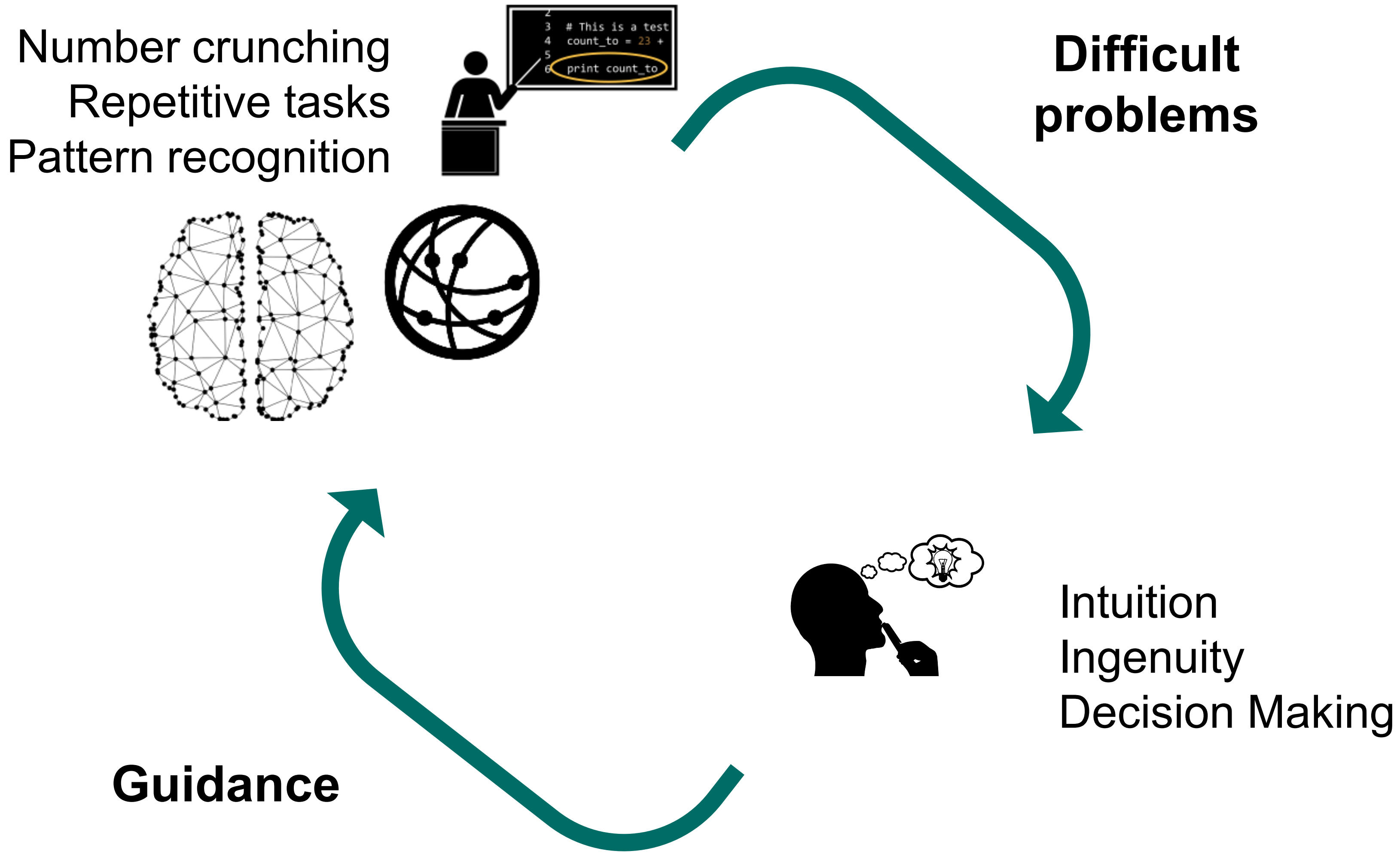
Cognitive factors

Research Gap:

Reverse engineers’ **strategies** and **cognitive factors** remain **poorly understood**.

63

WHAT TO AUTOMATE?



Number crunching
Repetitive tasks
Pattern recognition

```
# This is a test
count_to = 23 +

print count_to
```

Difficult problems

Intuition
Ingenuity
Decision Making

Guidance

THREE PILLARS TO (MORE) OPEN HARDWARE SECURITY

Capable and Open Algorithms

Suitable for real-world data & devices

Accessible & Intuitive Tools

For humans at different levels of experience

Training Resources

Outside of enterprise trainings

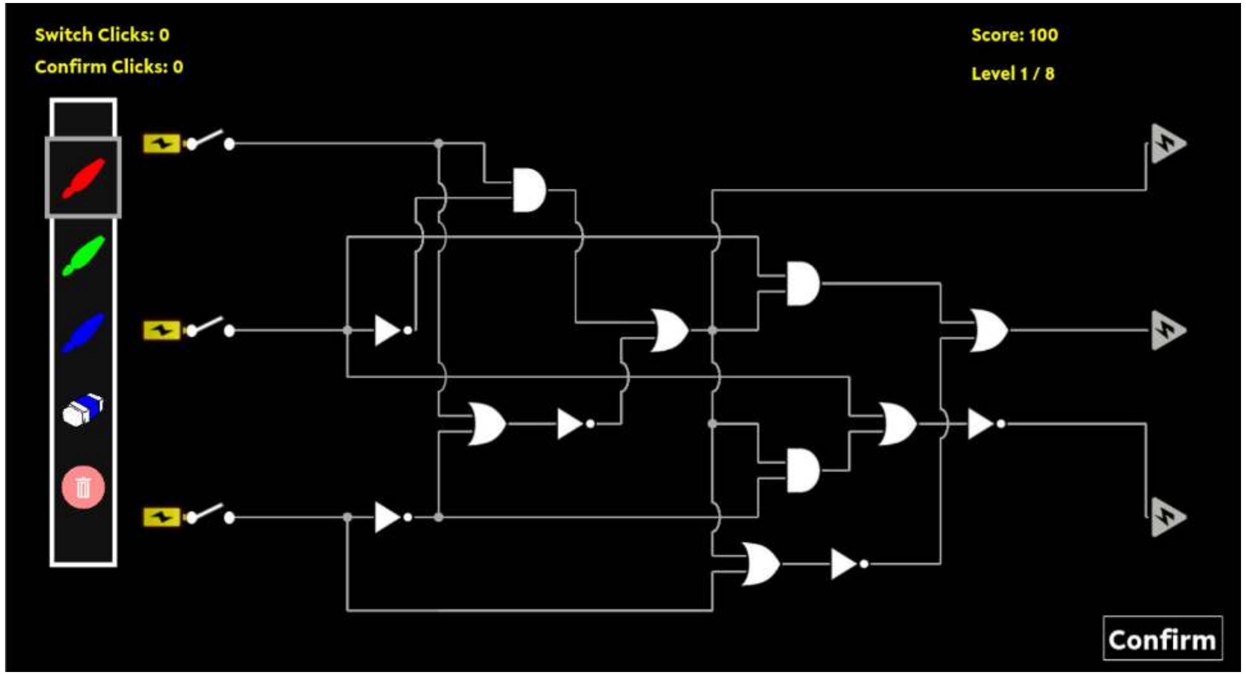
ChipSuite, HAL

Current interdisciplinary research

REVERSE ENGINEERING GAME

We want you …

… to play computer games for science!



Steffen Becker, Carina Wiesen, René Walendy, Nikol Rummel, and Christof Paar. (preprint) Analyzing Human Aspects in Hardware Reverse Engineering with REVERSIM—A Methodological Approach when Experts are Unavailable

PLAY NOW

https://beta.hrestudy.com/37c3

… or find us at the assembly of **LABOR e.V. @ Chaos West**

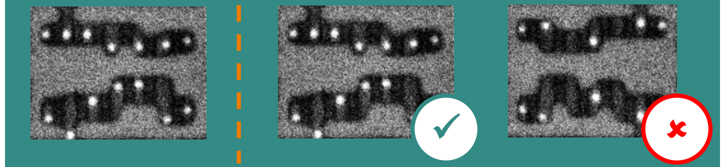
SUMMARY

Hardware Trojan Basics

- Subtle hardware manipulations can maliciously alter behavior
- Complex supply chains provide large attack surfaces
- Detection requires destructive imaging of the IC

The Over-View

Red Team vs. Blue Team



- Computer Vision can help detect potential manipulations
- High accuracy with open algorithms working on consumer devices

The Technical View

Reverse Engineering for Humans

- Reverse Engineering is a complex Human-Computer Interaction challenge
- Hardware Security starts with enabling humans researching hardware

The Community View

HARDWARE & IT SECURITY IN BOCHUM

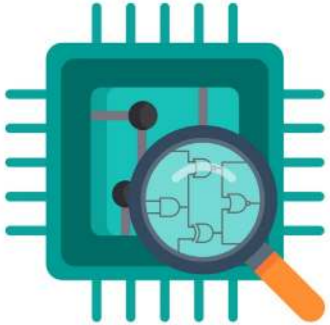
RUHR UNIVERSITÄT BOCHUM RUB

CASA
CYBER SECURITY IN THE AGE OF LARGE-SCALE ADVERSARIES

RUB Faculty of Computer Science
www.informatik.rub.de

CUBE⁵
Creating Security

MAX PLANCK INSTITUTE FOR SECURITY AND PRIVACY



- First **Hardware Reverse Engineering Workshop** (HARRIS) in January 2023
- Save the date for HARRIS 2024: **March 19./20., 2024**
- More info & newsletter at https://harris2023.mpi-sp.org