UCLouvain

CASA
Cyber Security in the Age of Large-Scale Adversaries

Ruhr Universität Bochum

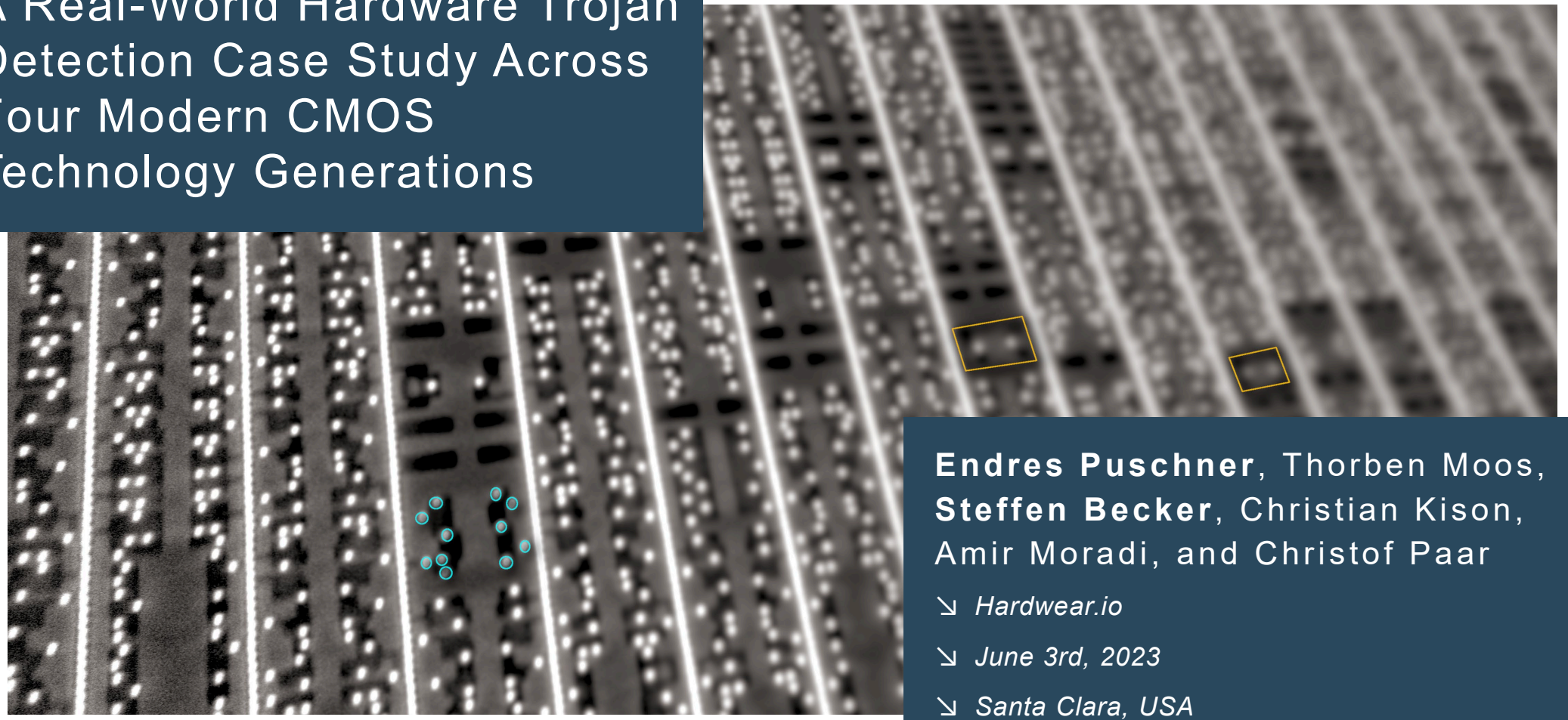
RUB

Bundeskriminalamt

MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

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

↘ *Hardwear.io*

↘ *June 3rd, 2023*

↘ *Santa Clara, USA*

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

CUBE 5
Creating Security

MAX PLANCK INSTITUTE FOR SECURITY AND PRIVACY

HARDWARE & IT SECURITY IN BOCHUM

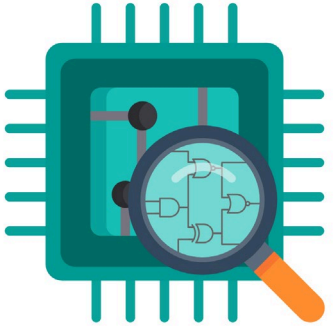
RUHR UNIVERSITÄT BOCHUM
RUB

CASA
CYBER SECURITY IN THE AGE OF LARGE-SCALE ADVERSARIES

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

CUBE 5
Creating Security

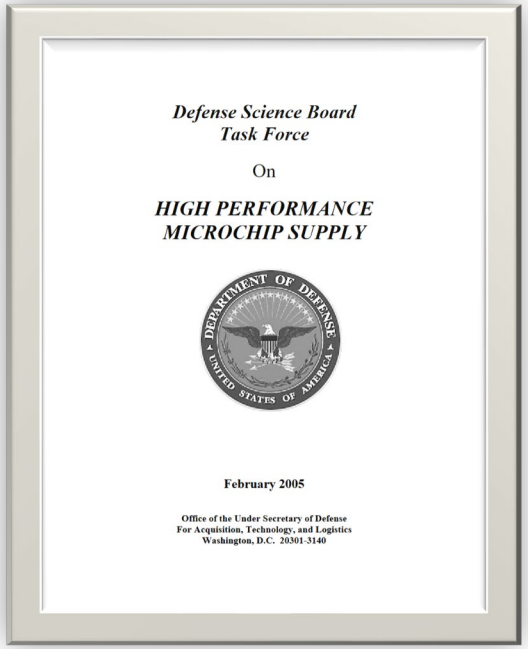
MAX PLANCK INSTITUTE FOR SECURITY AND PRIVACY



- 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

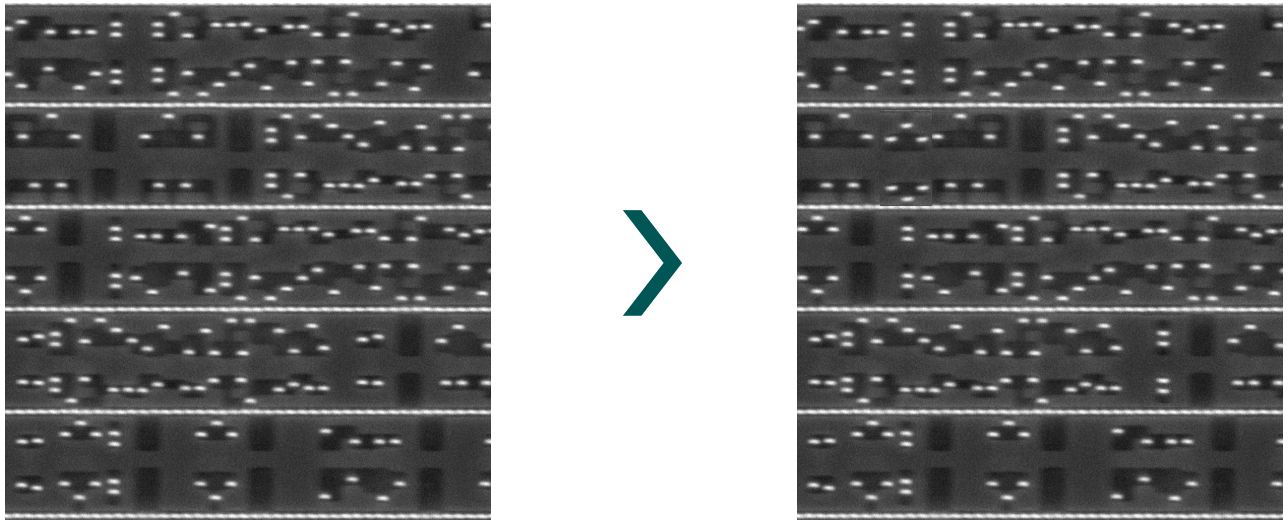
HARDWARE TROJANS

- Malicious modifications of integrated circuits (ICs)
- First publicly discussed by Department of Defense (DoD) in 2005
- Example payloads: Kill switch; information leakage; …
- Possible realization: Alter chip behavior by adding or replacing logic cells



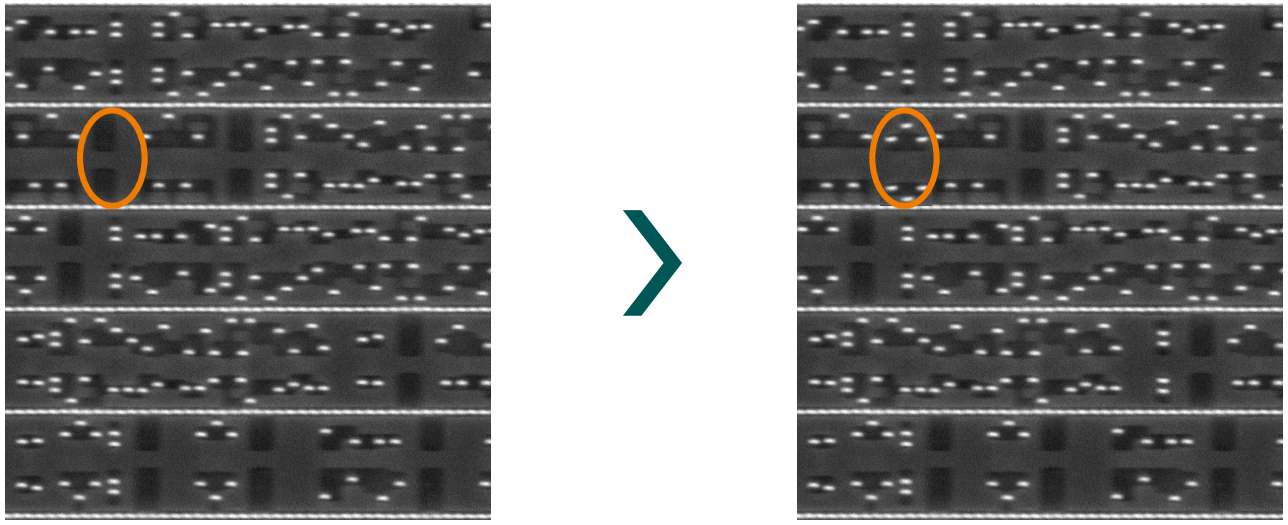
HARDWARE TROJANS

- Malicious modifications of integrated circuits (ICs)
- First publicly discussed by Department of Defense (DoD) in 2005
- Example payloads: Kill switch; information leakage; …
- Possible realization: Alter chip behavior by adding or replacing logic cells



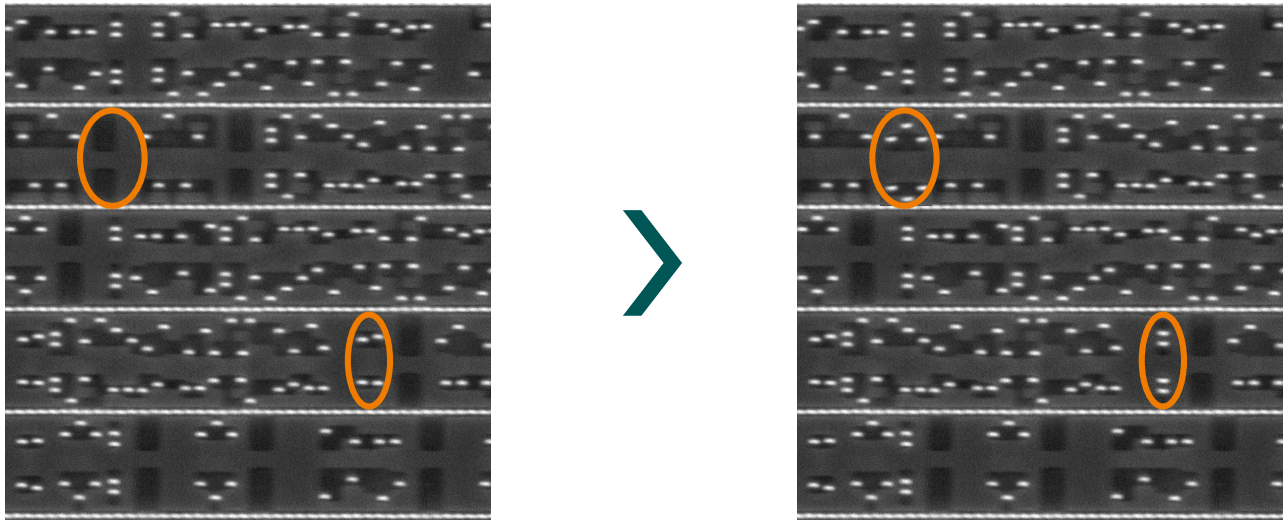
HARDWARE TROJANS

- Malicious modifications of integrated circuits (ICs)
- First publicly discussed by Department of Defense (DoD) in 2005
- Example payloads: Kill switch; information leakage; …
- Possible realization: Alter chip behavior by adding or replacing logic cells

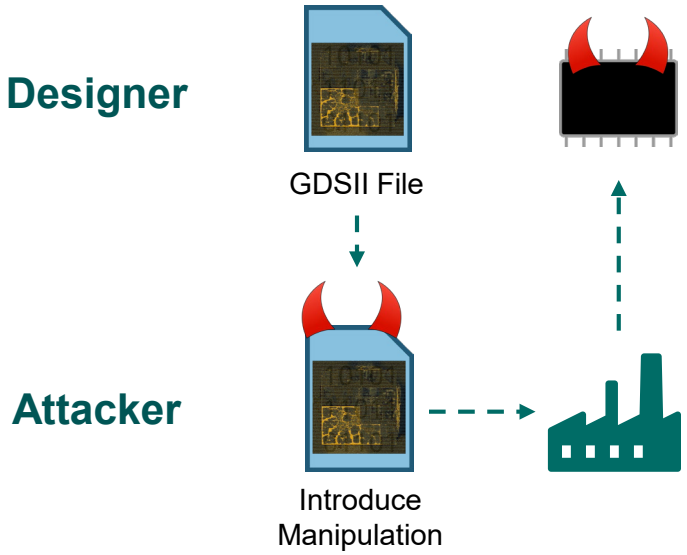


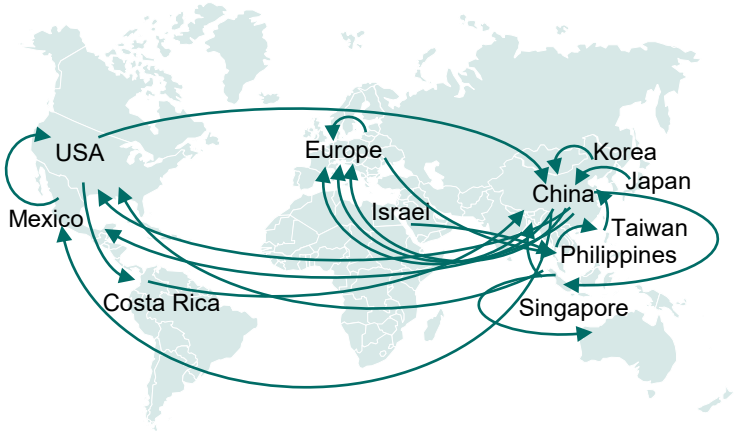
HARDWARE TROJANS

- Malicious modifications of integrated circuits (ICs)
- First publicly discussed by Department of Defense (DoD) in 2005
- Example payloads: Kill switch; information leakage; …
- Possible realization: Alter chip behavior by adding or replacing logic cells



THREAT MODEL





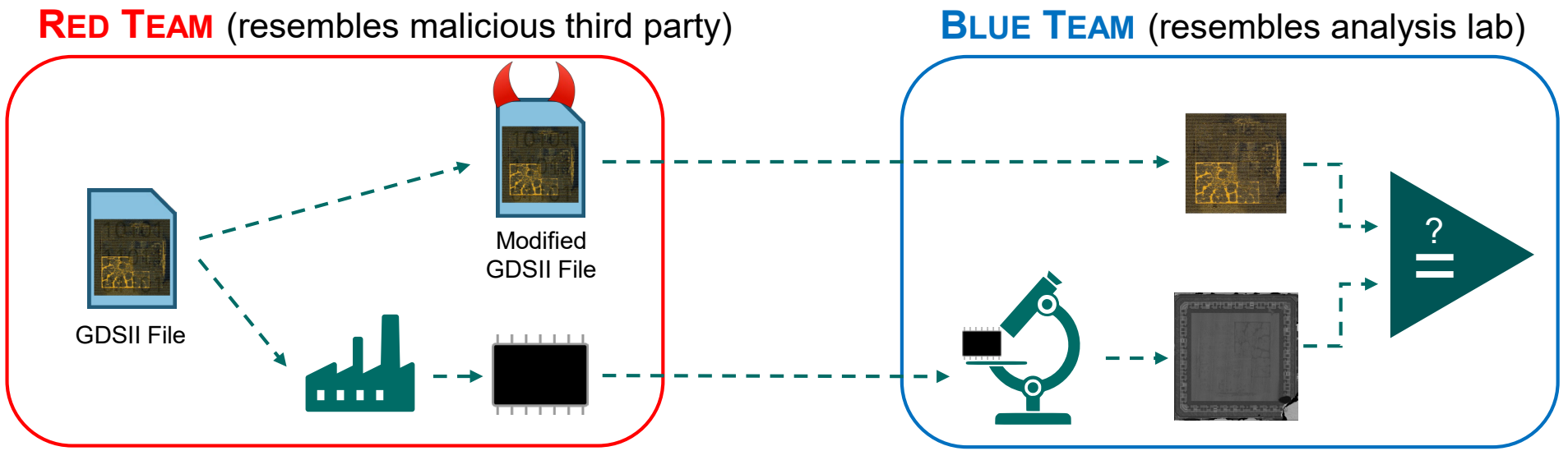
- Distributed manufacturing
- Relevant scenario: malicious fab / transport
- Other steps are “Trojan free”

Main Research Question:

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

RED TEAM VS. BLUE TEAM APPROACH

Purpose: minimize research bias



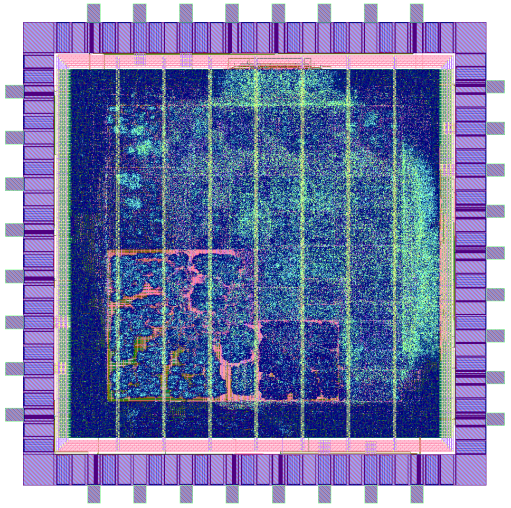
→ creates delta between chips and design files

→ receives chips & design files from RED TEAM

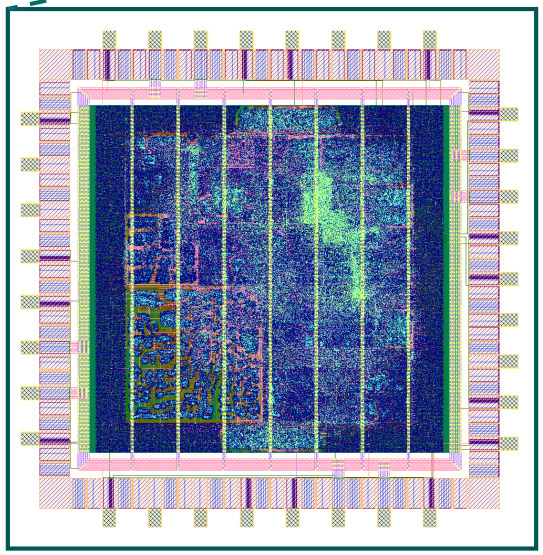
→ tries to find the differences

RED TEAM: INTRODUCE CELL MODIFICATIONS

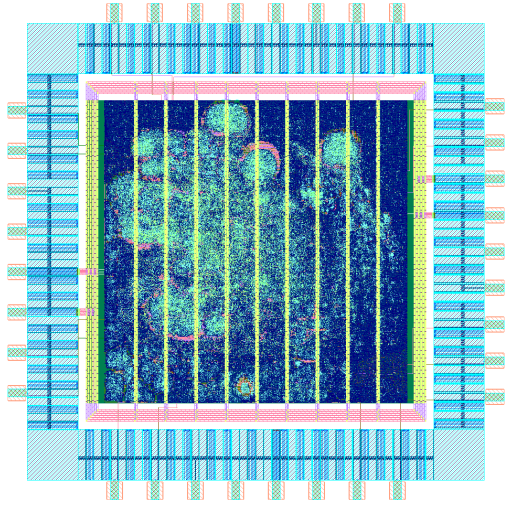
FOUR TARGET CHIPS



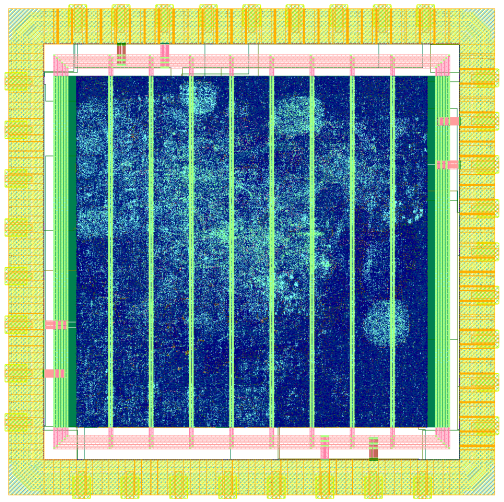
a) 90nm IC



b) 65nm IC

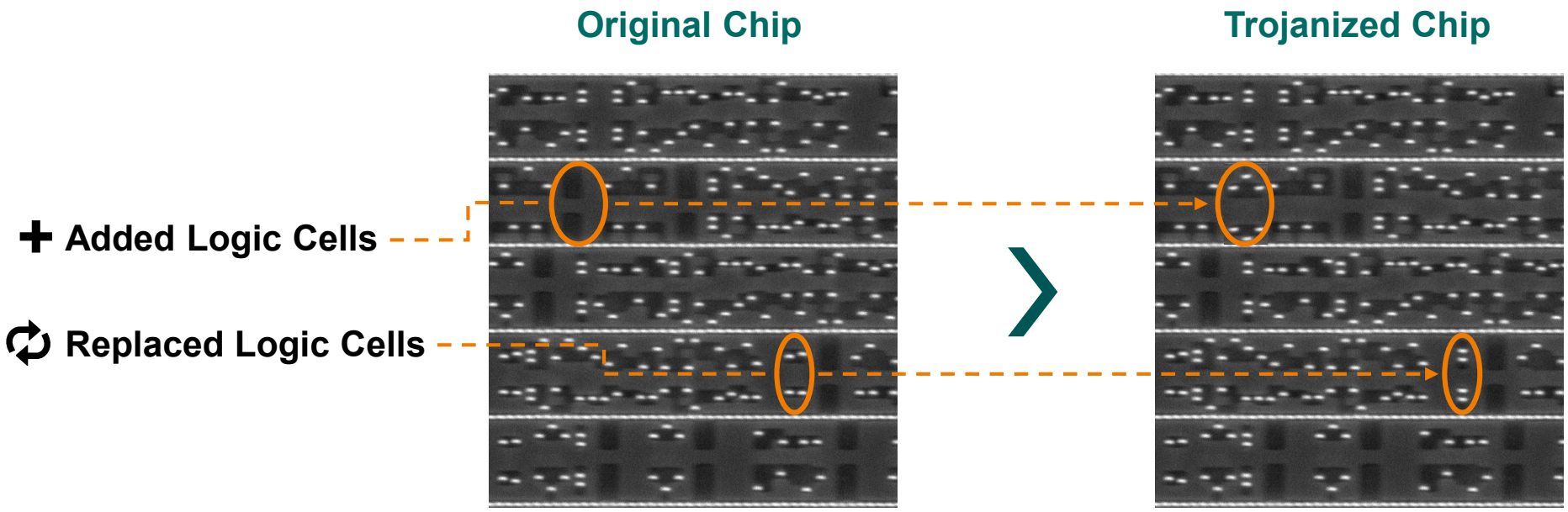


c) 40nm IC



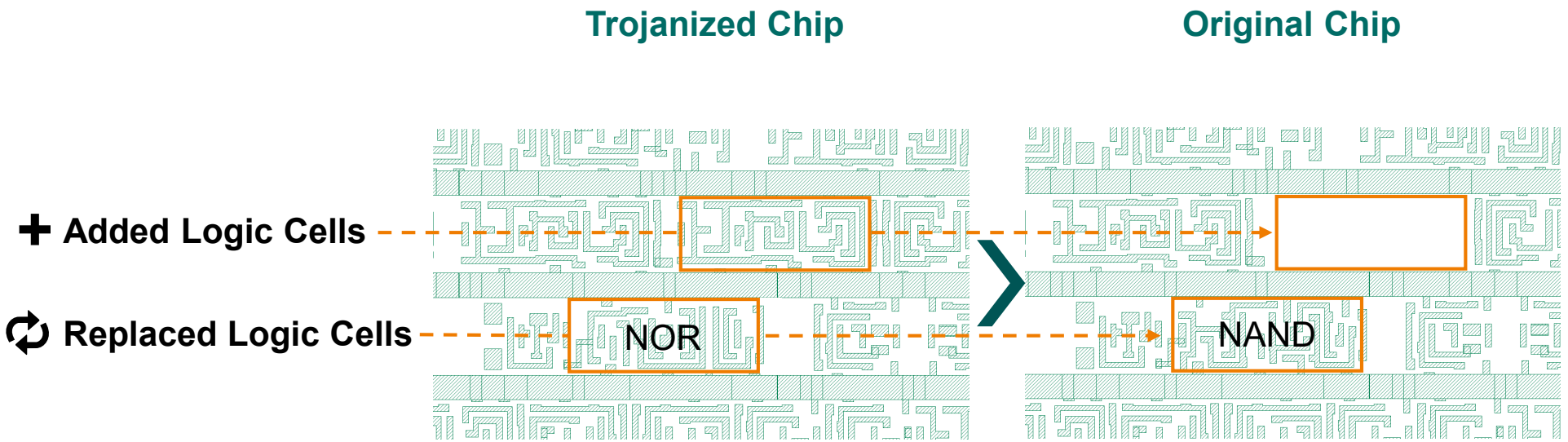
d) 28nm IC

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

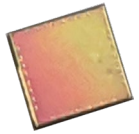
EMULATE INSERTED TROJAN



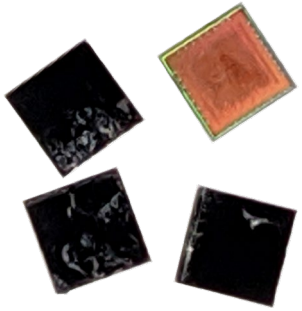
- RED TEAM introduces ten modifications per chip at random positions:
  - 6 x ✚ Added logic cells
  - 4 x 🔁 Replaced logic cells
- 40 of 3,410,580 total cells → 0.001 % modified

BLUE TEAM: IMAGE THE CHIP

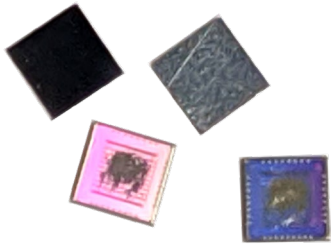
BLUE TEAM RECEIVES…



90 nm



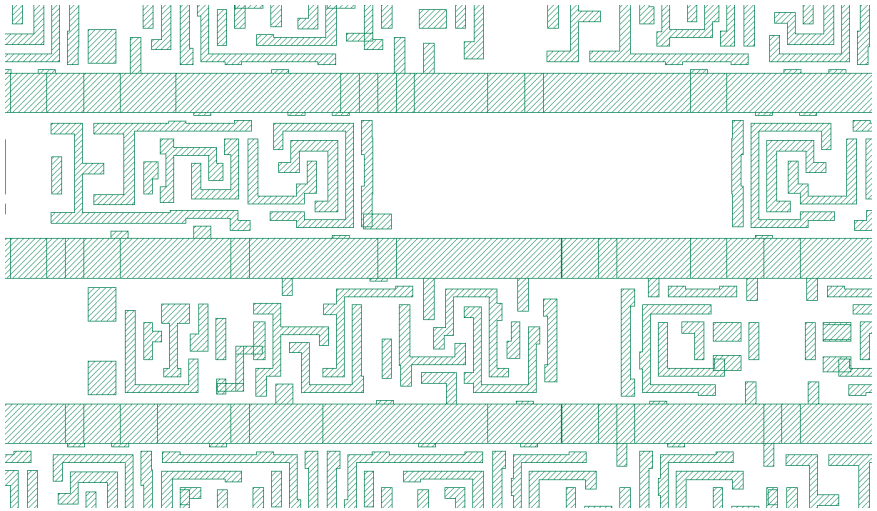
65 nm



40 nm

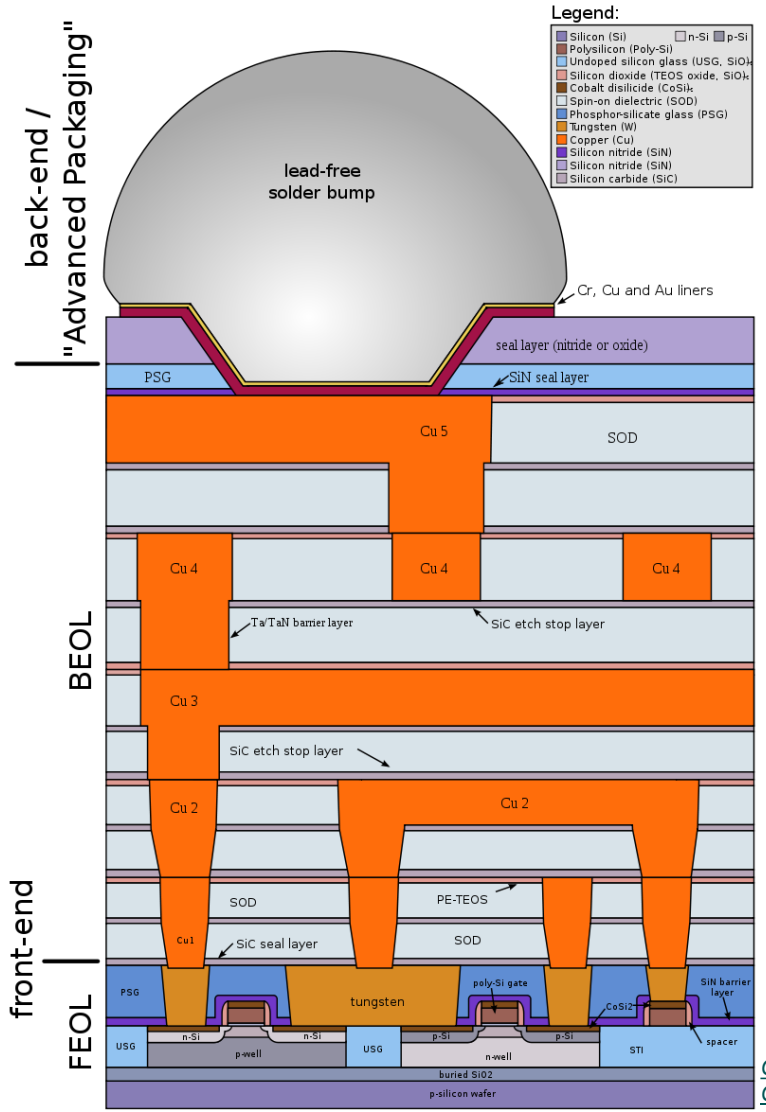
28 nm

a) Physical samples



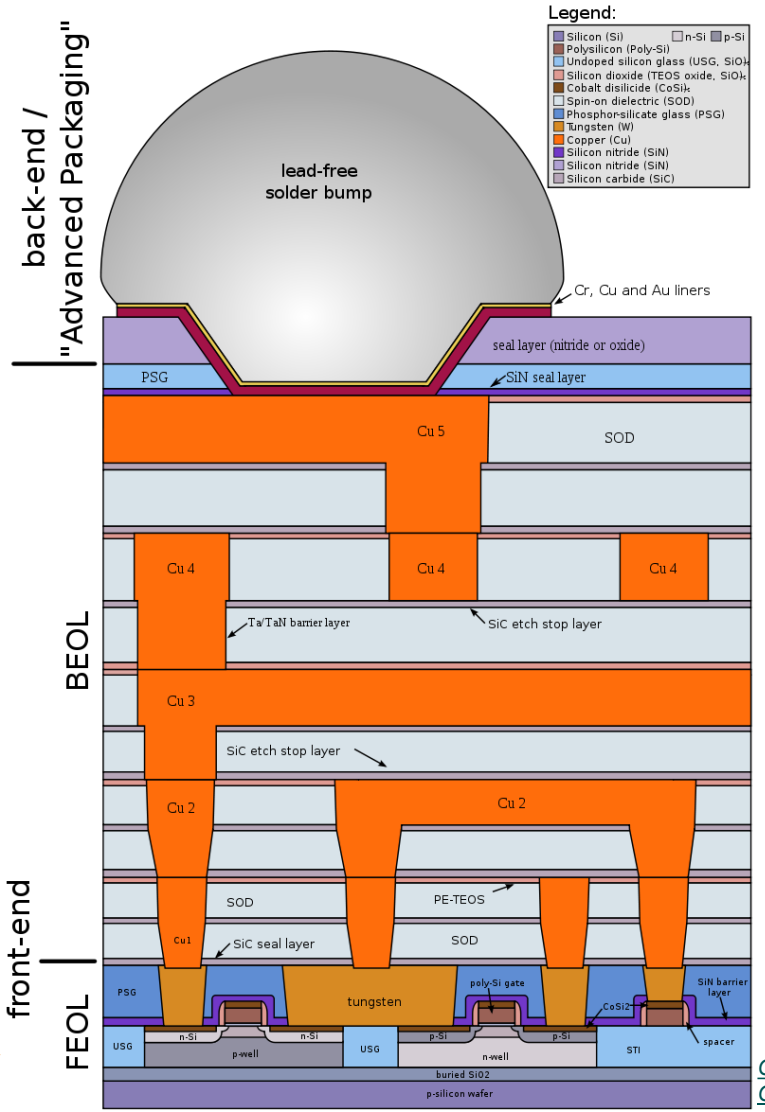
b) GDS-II Design

INSIDE CHIPS: STACKED LAYERS



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

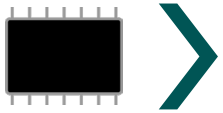
INSIDE CHIPS: STACKED LAYERS



Region of interest

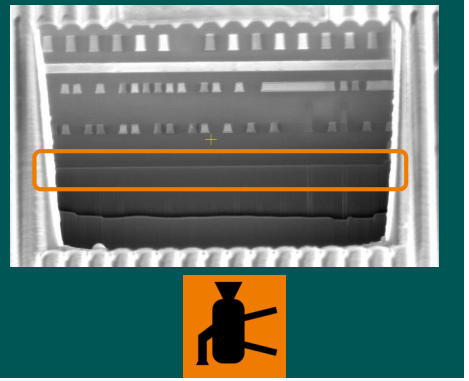
CMOS-chip structure in 2000s (en), Cepheiden, *CC BY 2.5*, via Wikimedia Commons

SAMPLE PREPARATION & IMAGING

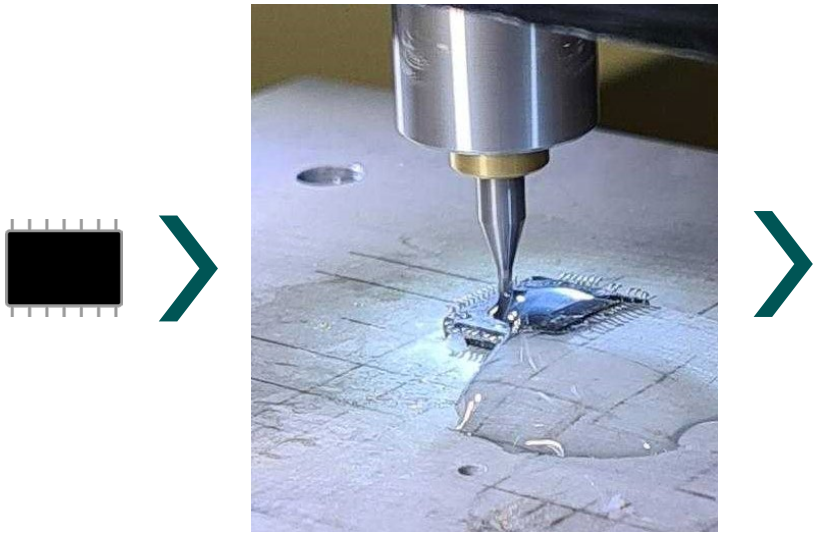


Cross-section Side View

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



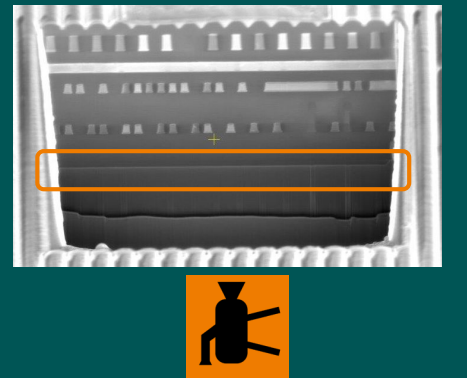
SAMPLE PREPARATION & IMAGING



1. CNC milling

Cross-section Side View

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



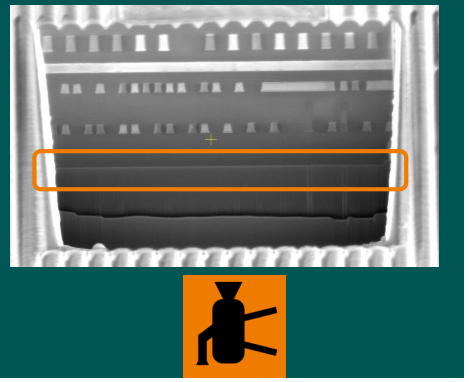
SAMPLE PREPARATION & IMAGING

1. CNC milling

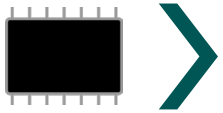
2. Choline hydroxide etching

Cross-section Side View

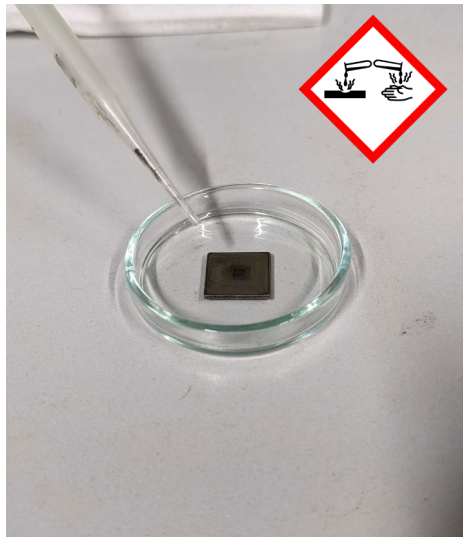
- Cell layout: Bottommost layer
- Removing silicon from the bottom
- Imaging the back side



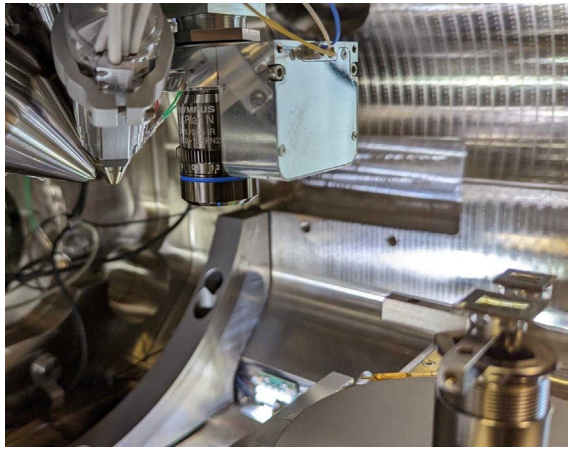
SAMPLE PREPARATION & IMAGING



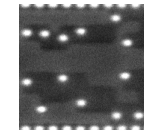
1. CNC milling



2. Choline hydroxide etching

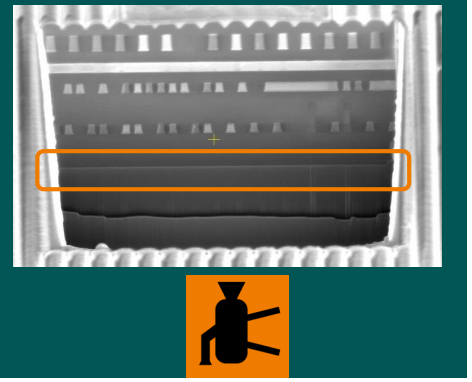


3. Scanning electron microscope (SEM)

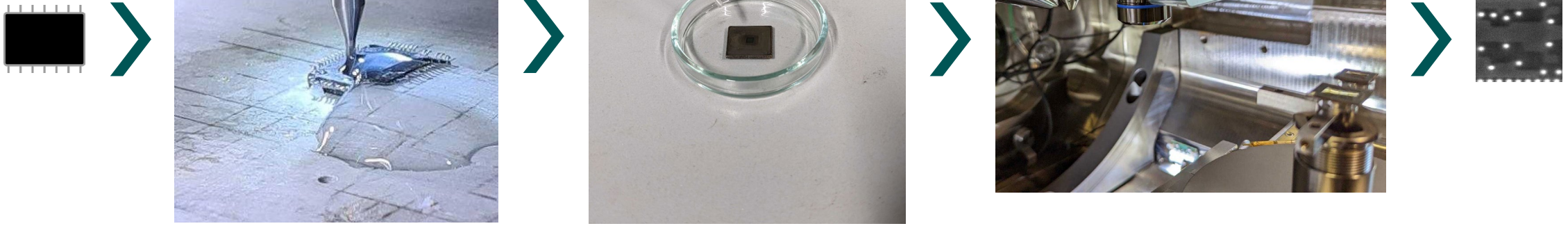


Cross-section Side View

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



SAMPLE PREPARATION & IMAGING



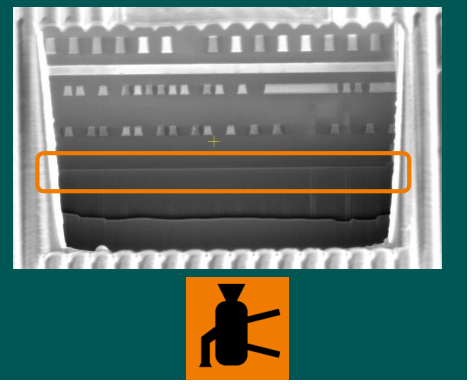
1. CNC milling

2. Choline hydroxide etching

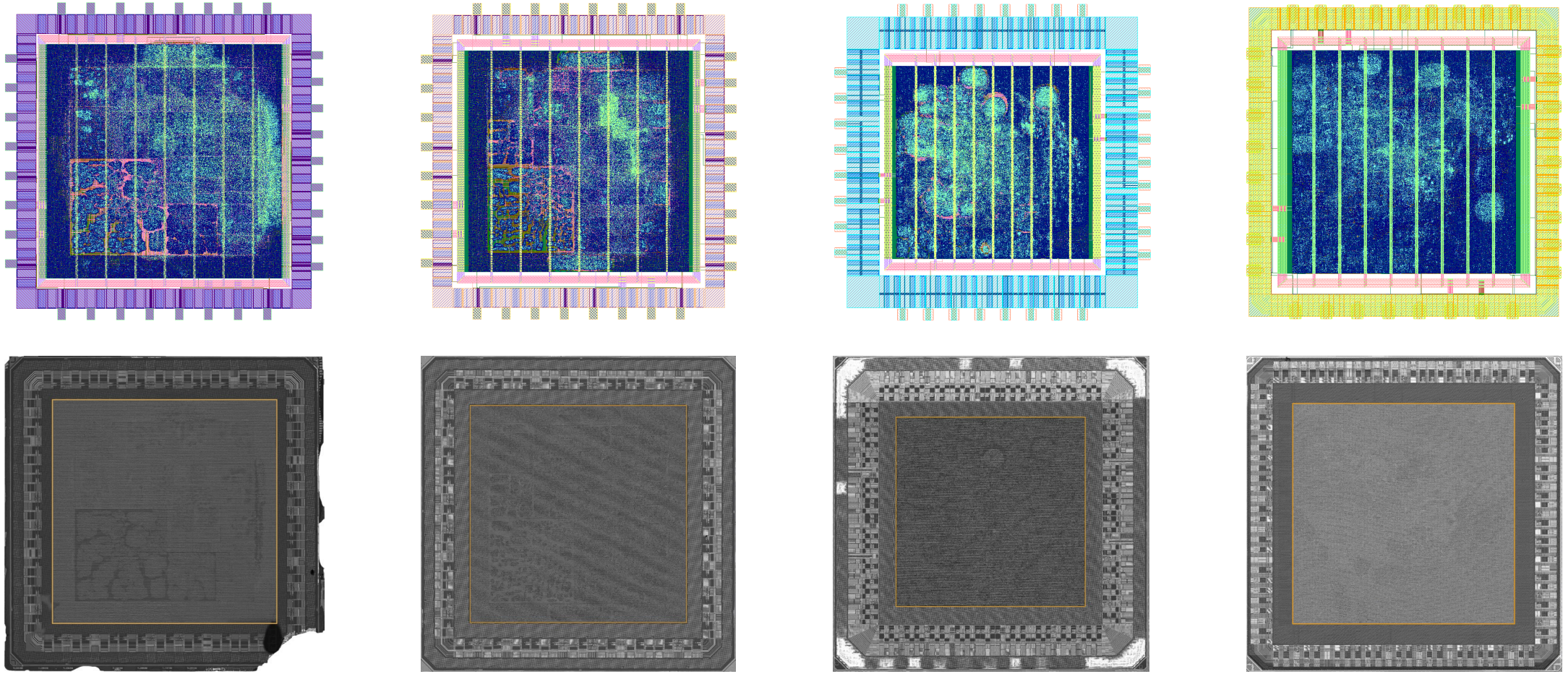
3. Scanning electron microscope (SEM)

Cross-section Side View

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



LAYOUT VS. SEM BACKSIDE IMAGE



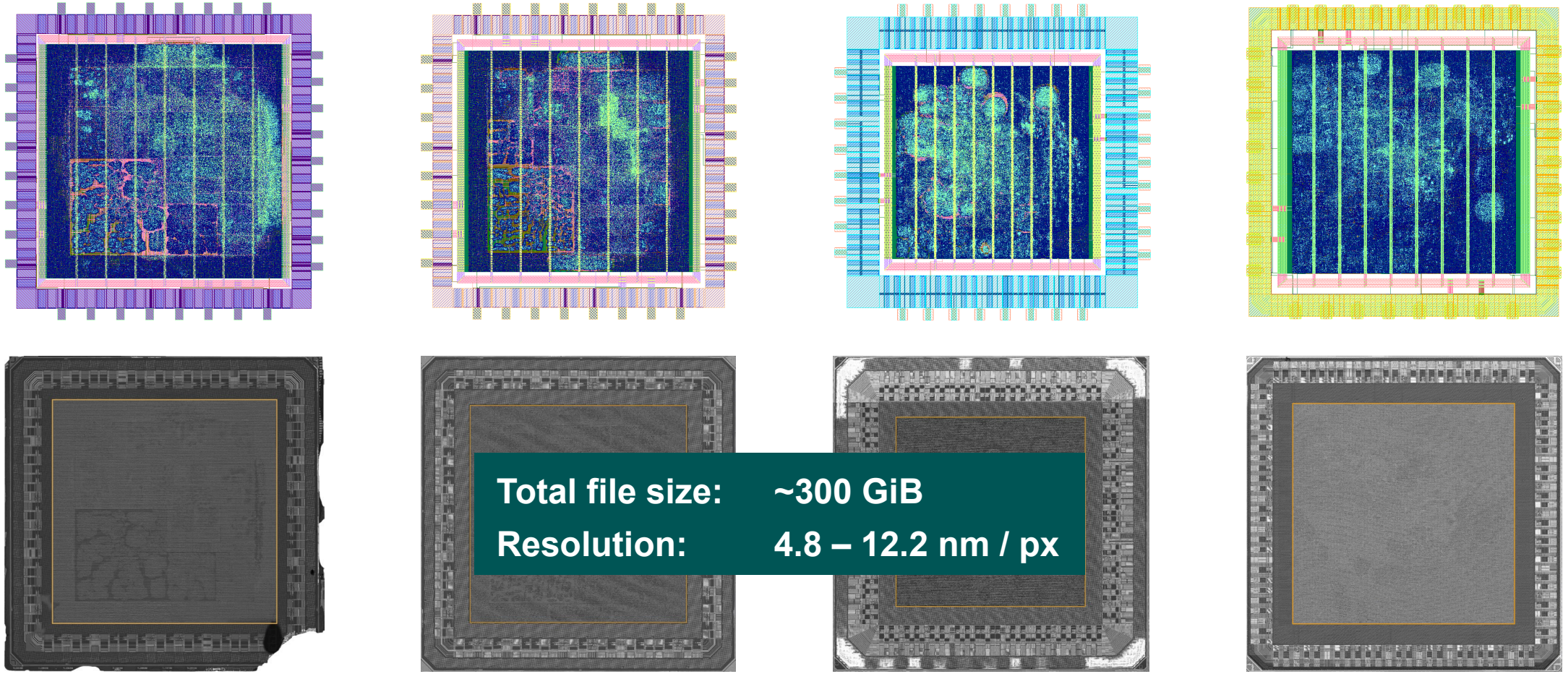
a) 90nm IC

b) 65nm IC

c) 40nm IC

d) 28nm IC

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

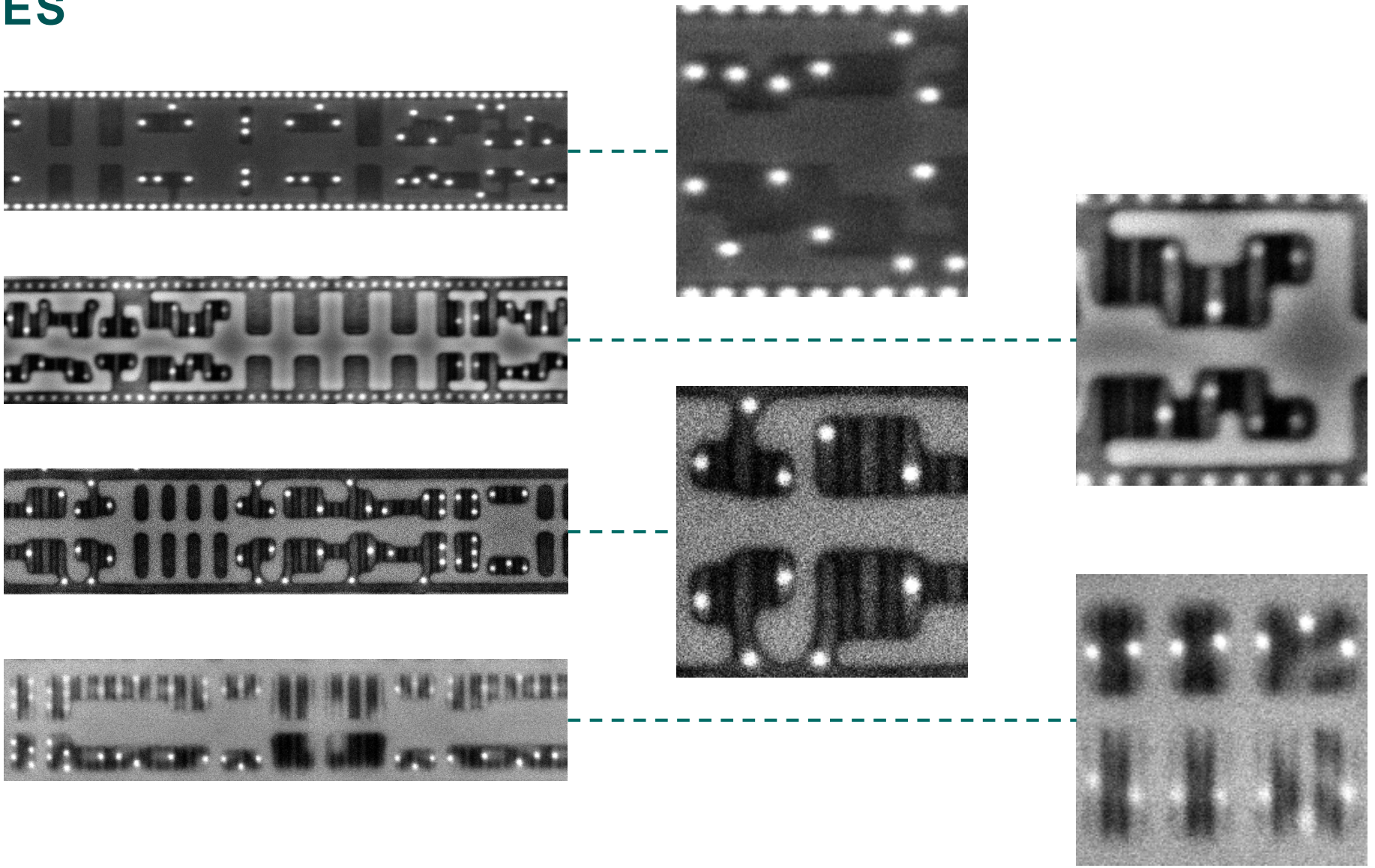
SEM IMAGES

a) 90nm IC

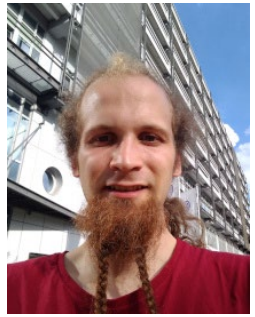
b) 65nm IC

c) 40nm IC

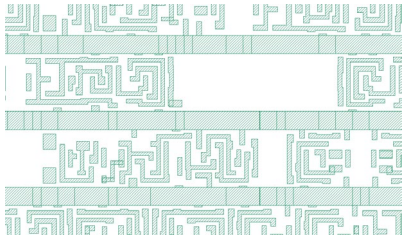
d) 28nm IC



BLUE TEAM: DETECT MANIPULATIONS



ROADMAP

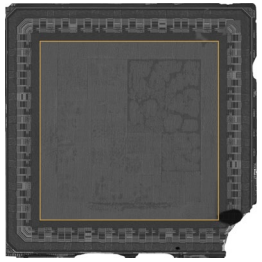


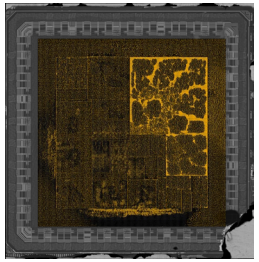
Alignment 1
Gather cell coordinates from GDSII design file

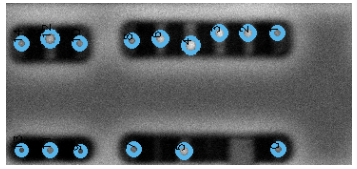
Alignment 2
Gather tile coordinates from stitched SEM images

Align Design & Images
Apply transformation

Detect Manipulations
Feature detection / Comparison algorithms



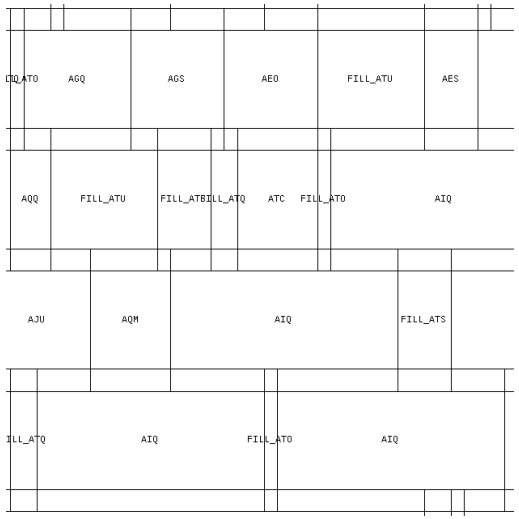




GDSII COORDINATES

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

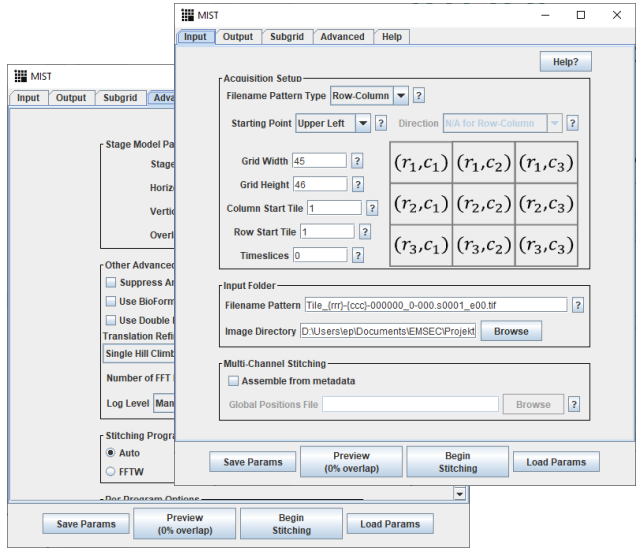
```python
import gdspy

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
```



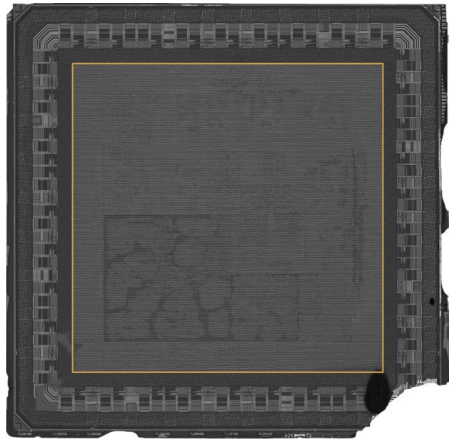
[4] https://github.com/heitzmann/gdspy

TILE IMAGE COORDINATES

Stitching done with e.g. MIST [5]




```
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);
```



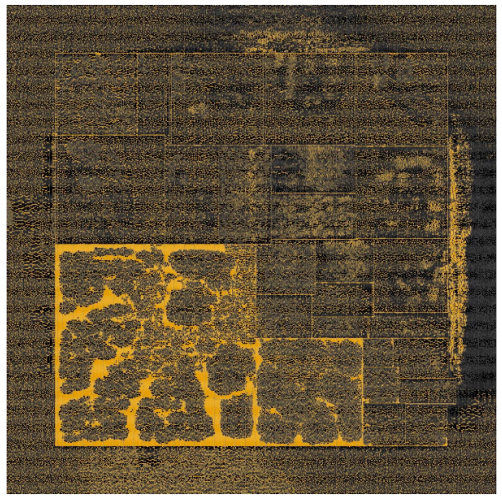
```python
import os

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/

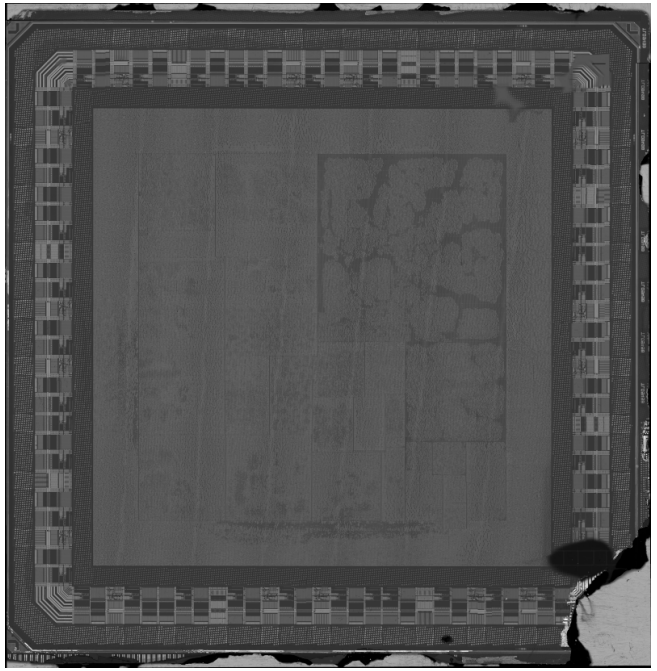
CONVERTING COORDINATES

- 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)



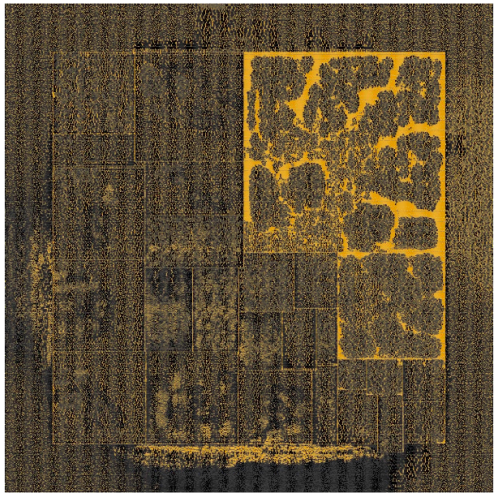
Stitched Image (filler cells darker, flipped horizontally and rotated by 90 degrees CCW)



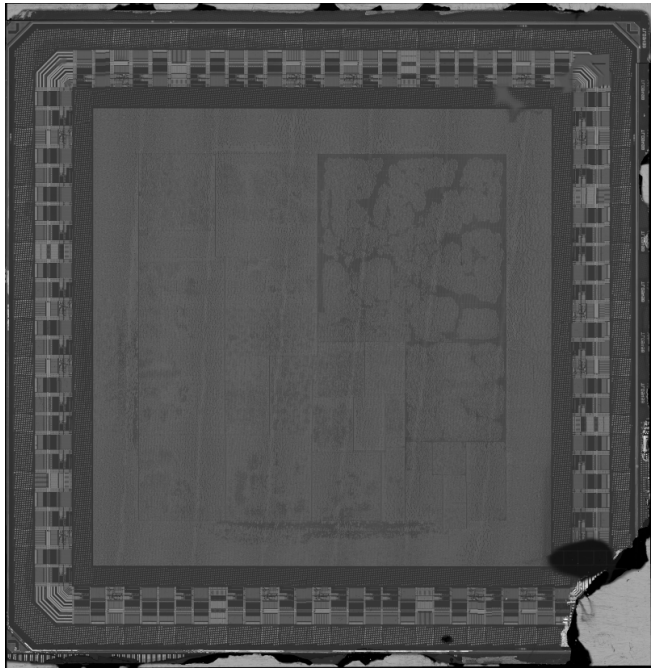
CONVERTING COORDINATES

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

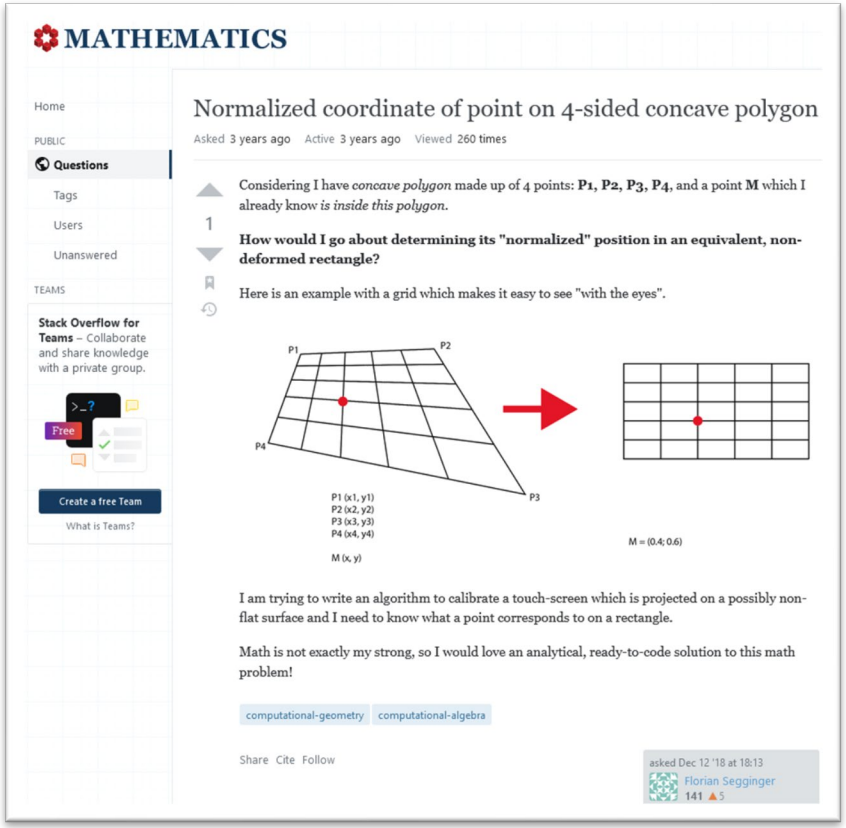
GDSII (orange = filler cell, black = other cell)



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.

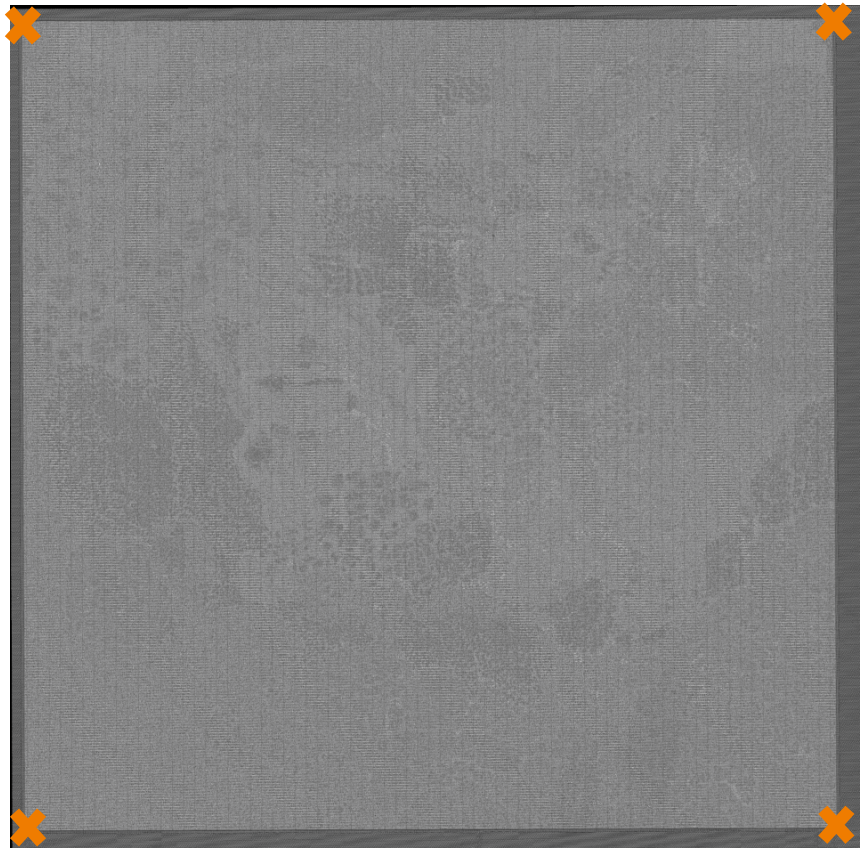
Math is not exactly my strong, so I would love an analytical, ready-to-code solution to this math problem!

computational-geometry computational-algebra

Share Cite Follow

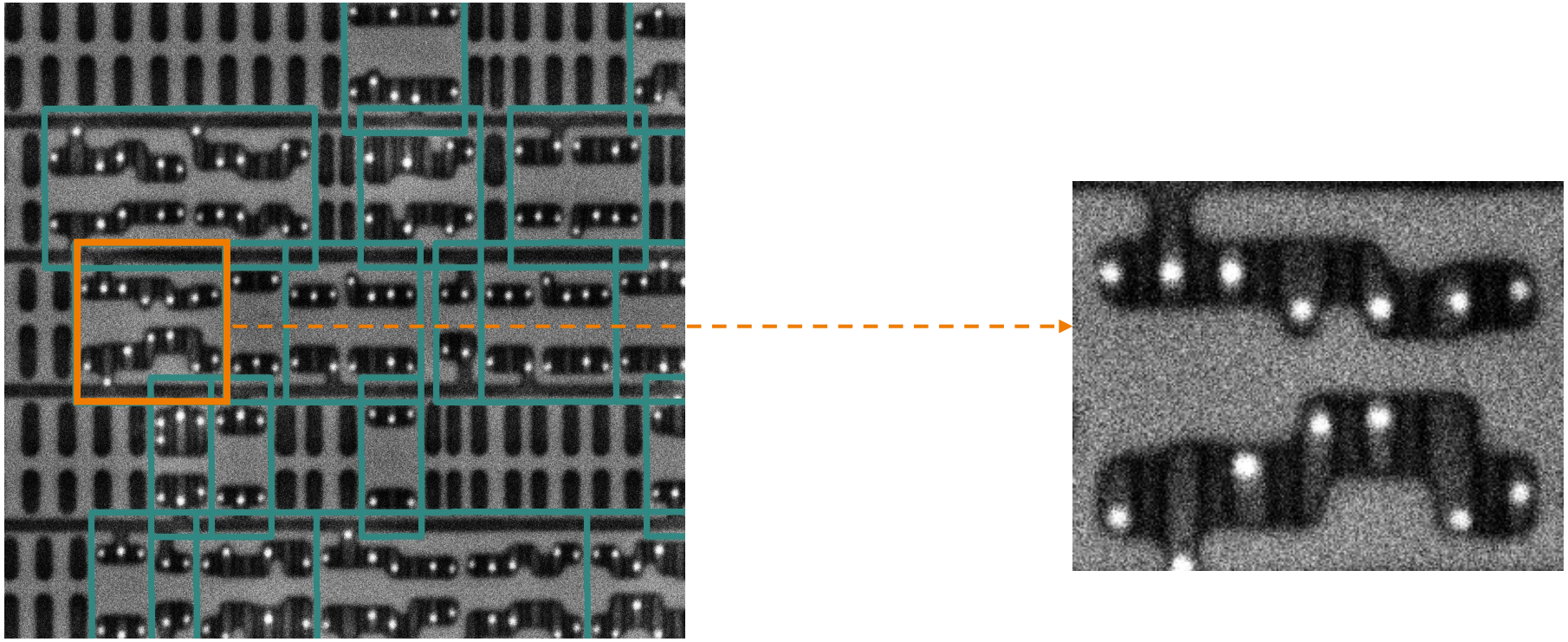
asked Dec 12 '18 at 18:13
Florian Segginger
141 ▲5

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



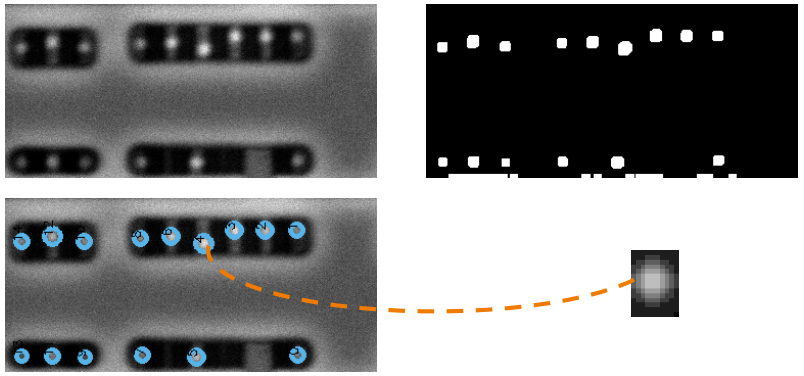
```python
# build transformation matrices
# affine transformation algorithm taken from https://math.stackexchange.com
x0 = IMAGE_EDGES[0][0]
y0 = IMAGE_EDGES[0][1]
x1 = IMAGE_EDGES[1][0]
y1 = IMAGE_EDGES[1][1]
x2 = IMAGE_EDGES[2][0]
y2 = IMAGE_EDGES[2][1]
x3 = IMAGE_EDGES[3][0]
y3 = IMAGE_EDGES[3][1]

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

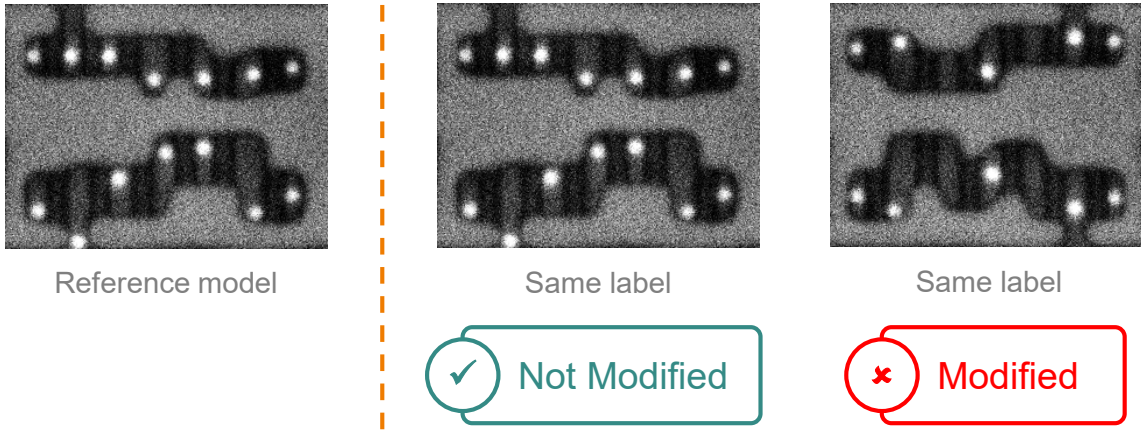


DECISION ALGORITHMS

✚ Additional Cells: Via Detection



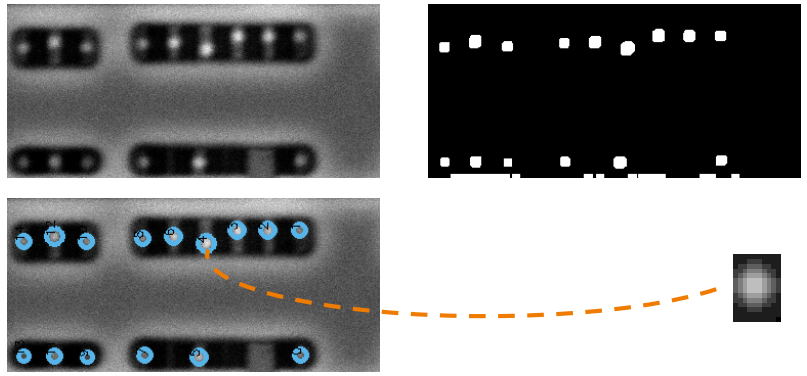
Replaced Cells: Template Matching

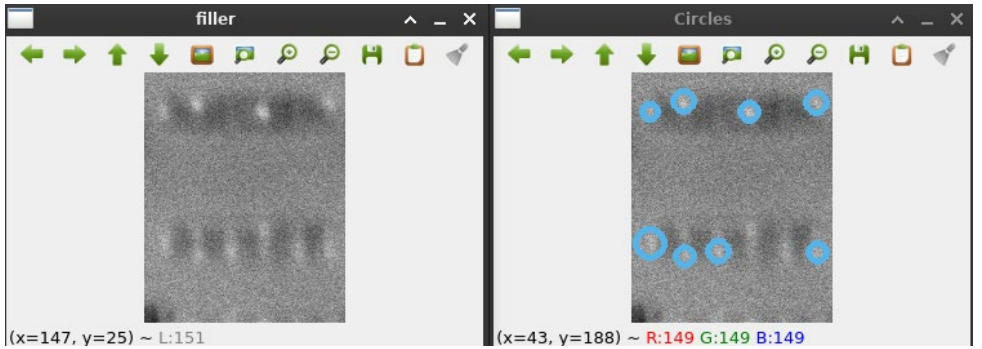


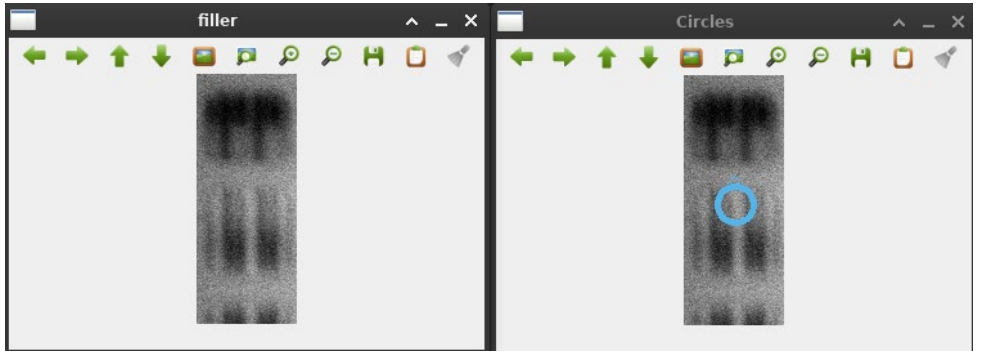
FINDING LOGIC CELLS WHERE FILLER CELLS ARE EXPECTED

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)





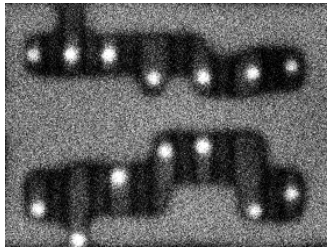


In the end, depends on image quality and parameters

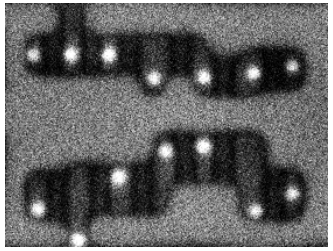
→ some false positives

FINDING CELL REPLACEMENTS

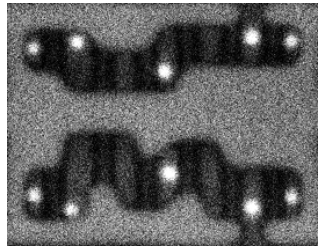
- Q: “Is the cell in question the one it is labeled, or was it replaced by another cell?” (e.g., NAND → NOR)



Golden Model / Template



Other Cell labeled same



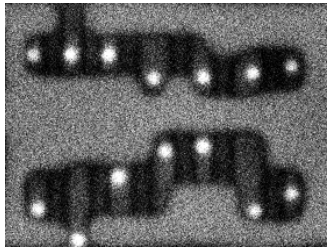
Other Cell labeled same

→ Template Matching

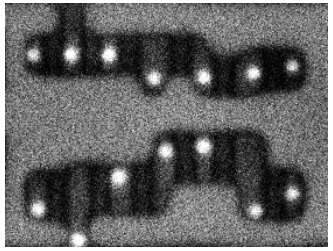
- Use a golden reference model to compare each cell of the same type (= label) against
- If different, count as candidate for modification

FINDING CELL REPLACEMENTS

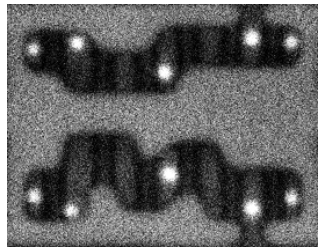
- Q: “Is the cell in question the one it is labeled, or was it replaced by another cell?” (e.g., NAND → NOR)



Golden Model / Template



Other Cell labeled same



Other Cell labeled same

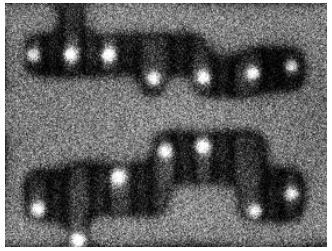
✓ Not Modified

→ Template Matching

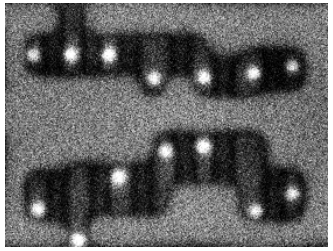
- Use a golden reference model to compare each cell of the same type (= label) against
- If different, count as candidate for modification

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)

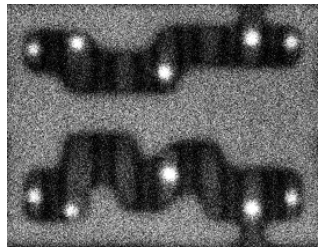


Golden Model / Template



Other Cell labeled same

✓ Not Modified



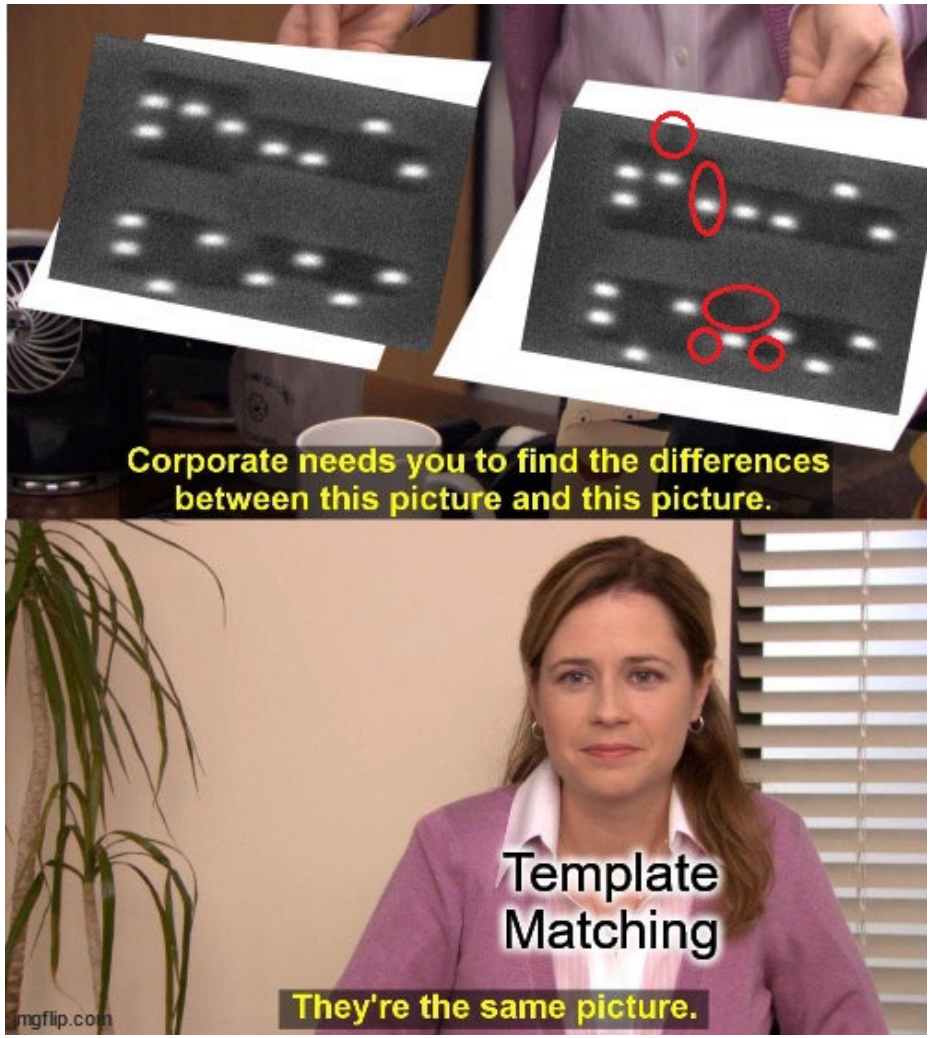
Other Cell labeled same

✘ Modified

→ Template Matching

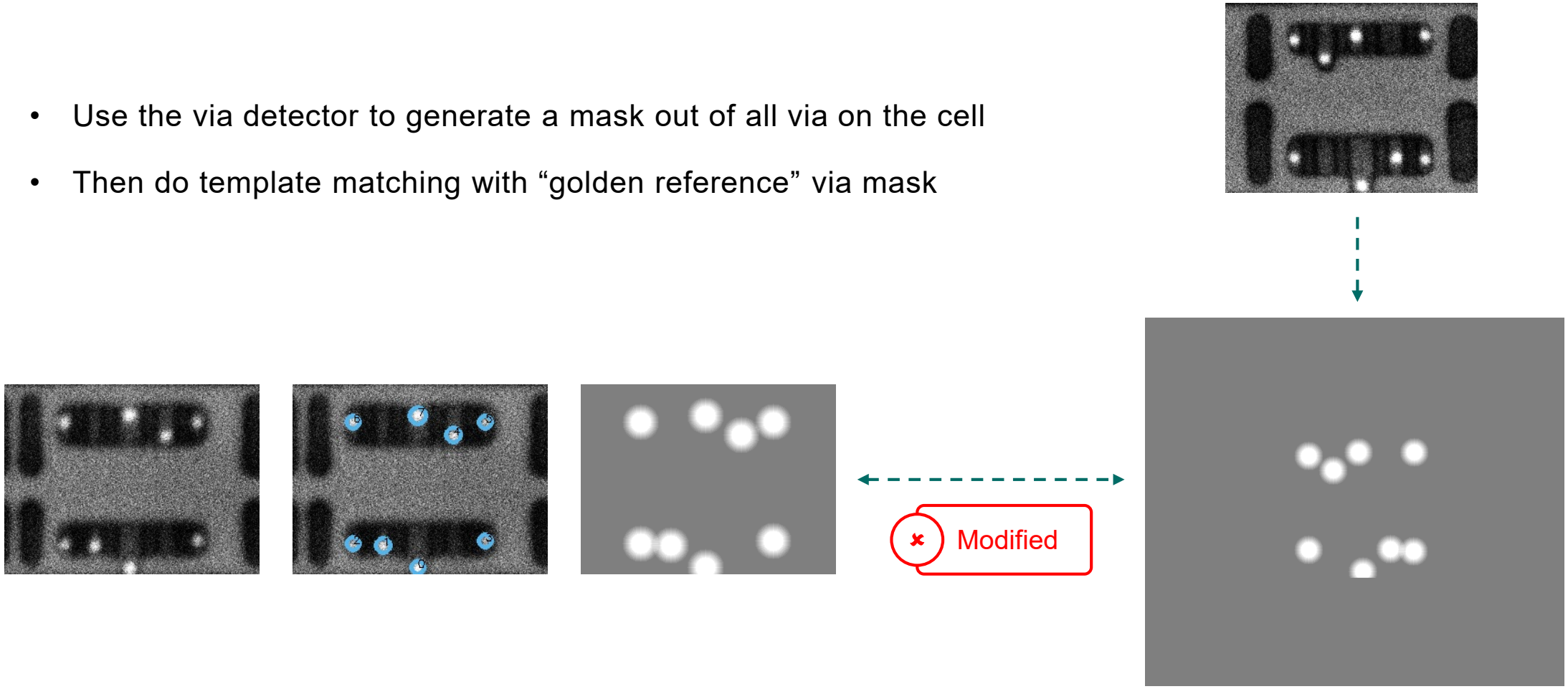
- Use a golden reference model to compare each cell of the same type (= label) against
- If different, count as candidate for modification

TEMPLATE MATCHING VS. SIMILAR CELLS



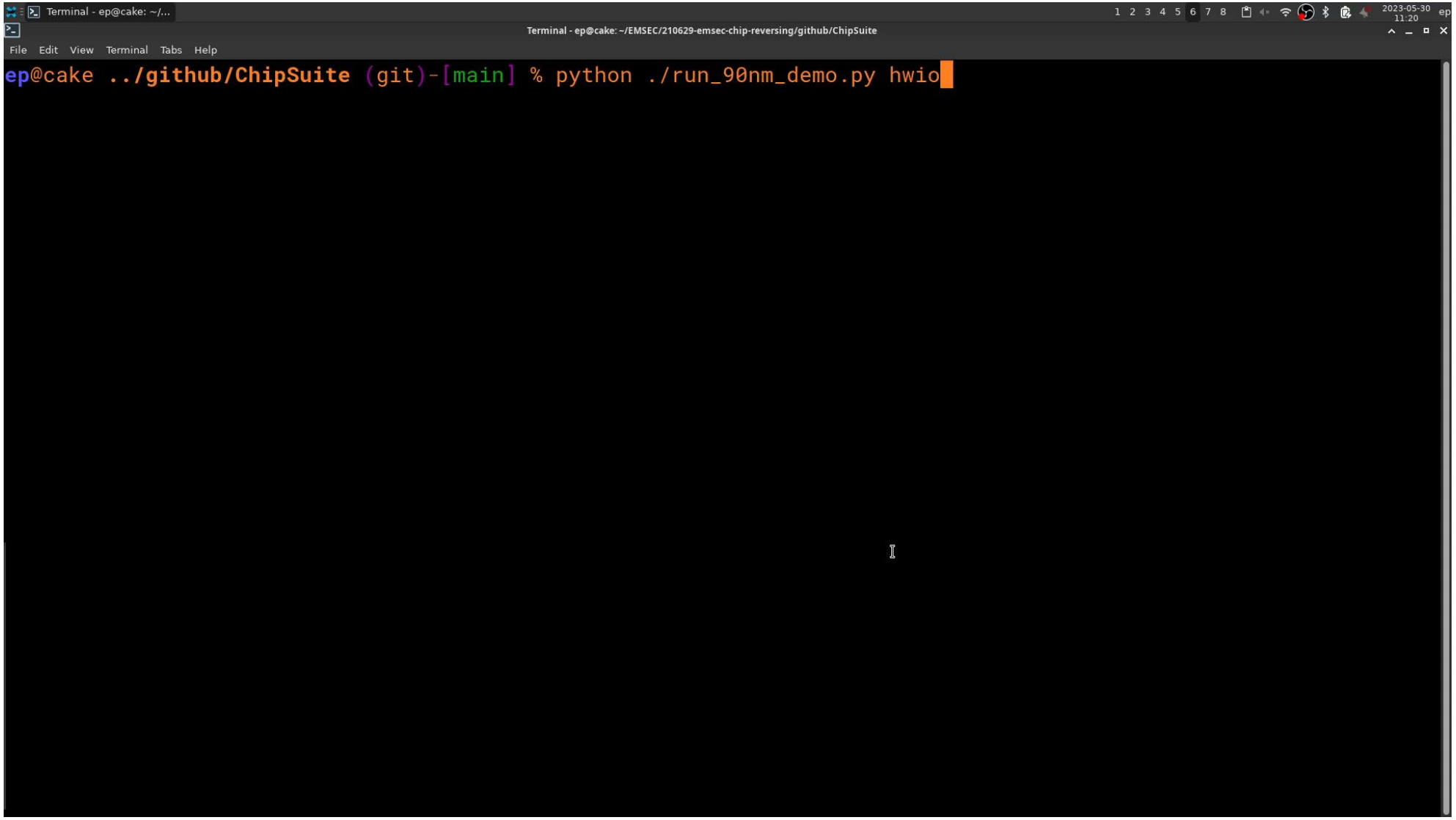
EXTENDED TEMPLATE MATCHING

- 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

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

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

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	

DETECTION RESULTS

Chip Statistics

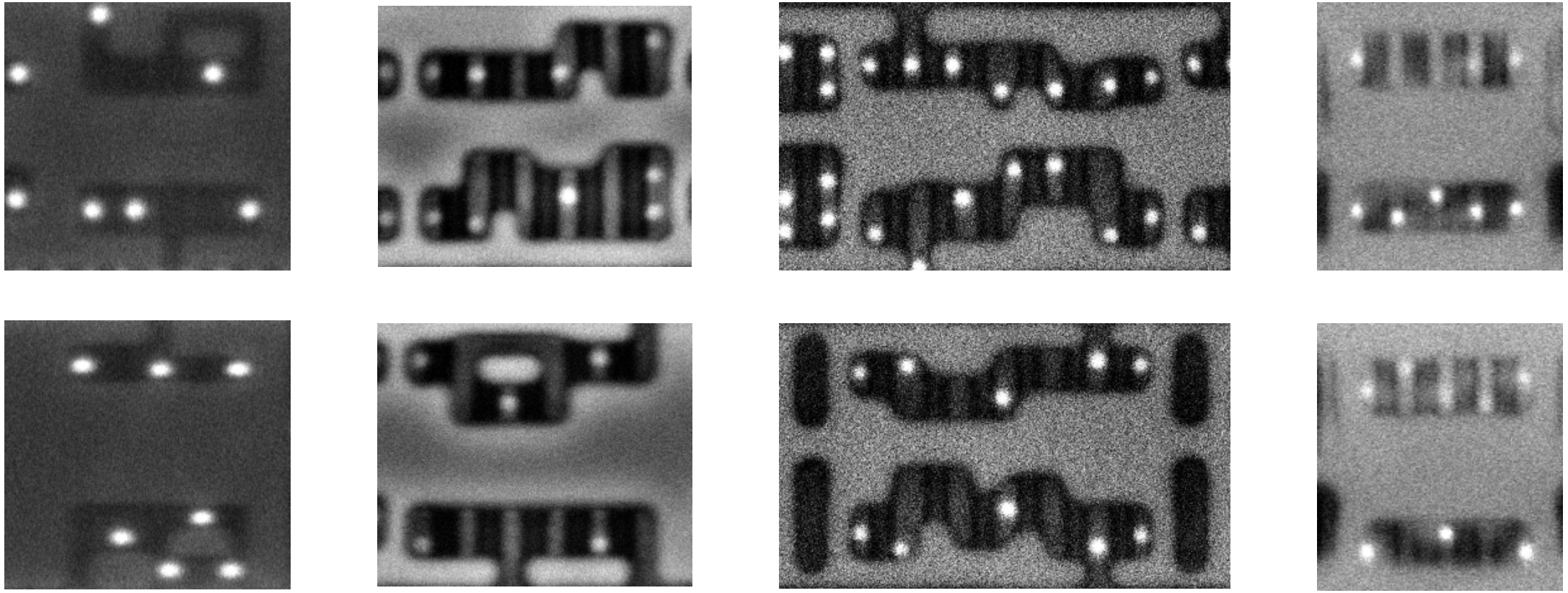
	90 nm	65 nm	40 nm	28 nm
Total Number of Cells	453,850	571,060	917,819	1,467,851
Region of Interest Area	2.089 mm²	1.848 mm²	1.052 mm²	0.962 mm²

✚ 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

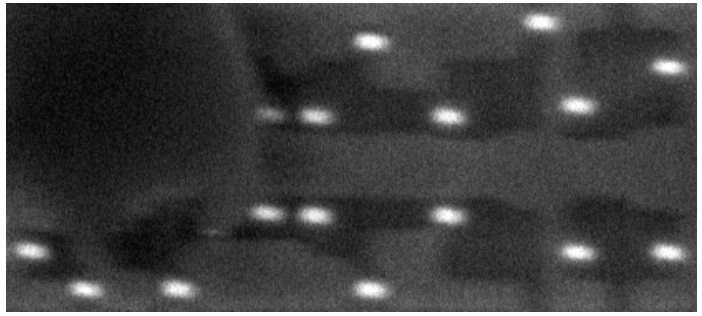
b) 65nm

c) 40nm

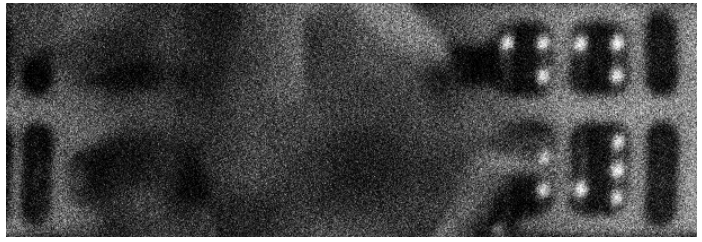
d) 28nm

SELECTED FALSE POSITIVES CAUSED BY DEBRIS OR DUST

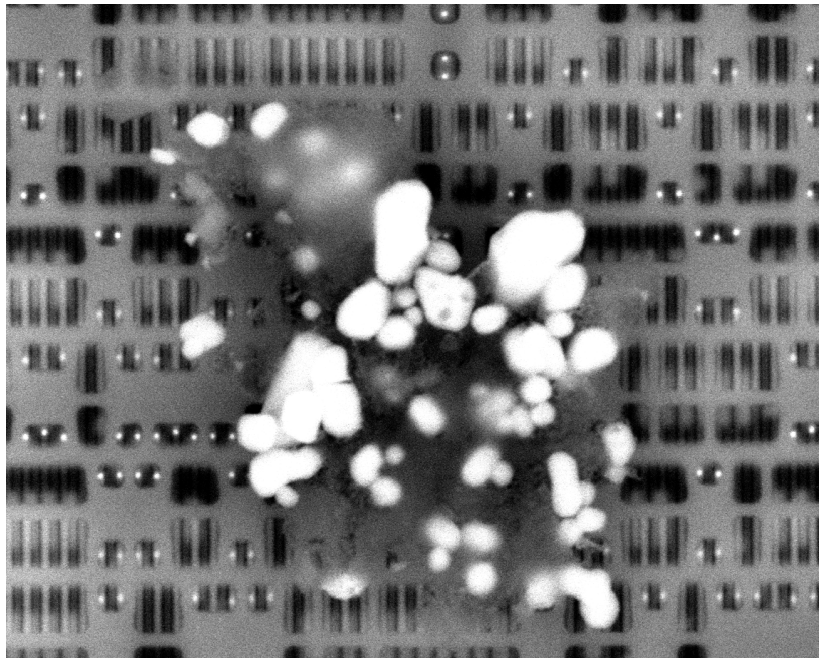
- This affects 161 out of 3.4 million cells in total (0.005%)



a) 90nm example

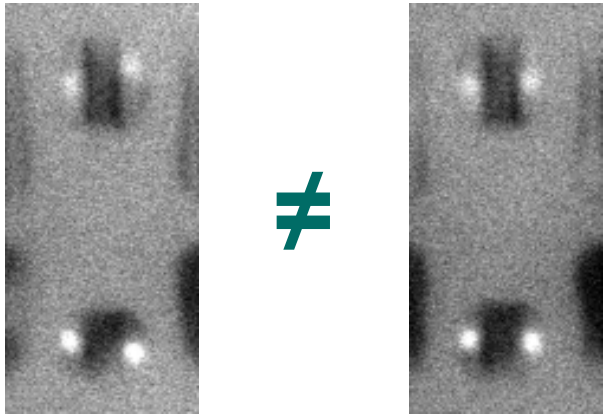


b) 40nm example



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

CONCLUSION

- 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*

SUMMARY

1. RED TEAM produces chips different to the design file
2. BLUE TEAM takes microscope images from chips
3. BLUE TEAM compares the images with labels from design file
4. Efficient detection possible with good image quality
5. Public dataset for further research

LINK TO THE GITHUB REPOSITORY

https://github.com/emsec/chipsuite