Degate

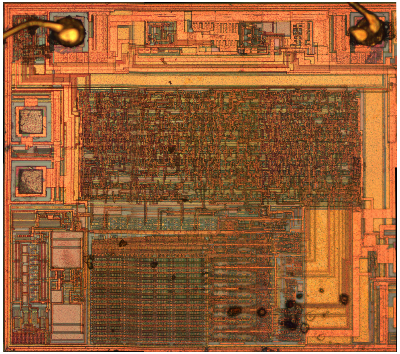
The stakes and challenges of silicon reverse engineering

https://www.degate.org

D. Bachelot

SecSea2k24,
October 11th and 12th, 2024

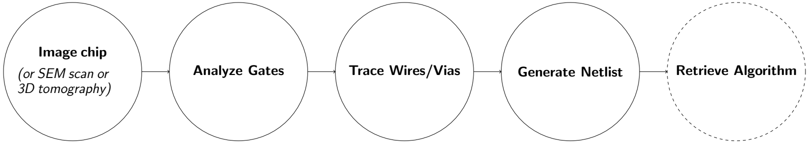
What is Silicon Chips RE?



Same idea than with software RE (from binary, to assembly and to code), silicon chip RE go **from silicon**, **to images**, **to transistors**, **to gates**, **to netlist** and **to algorithm**.

With proper preparation and knowledge, we can go into silicon, **analyze transistors**, **retrieve gates/wires/vias** and **reconstruct implemented algorithms**. This can be used to **analyze old hardware**, build **software emulators**, search for **vulnerabilities** and **backdoors**, **break/test a protection**, **secret extraction** or **check intellectual property**.

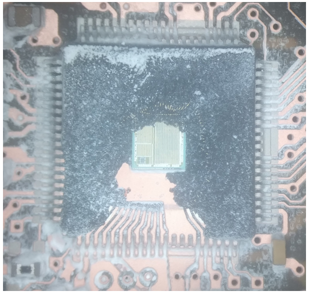
Used in IC industry for **fault/failure detection** & analysis, but **not at the same scale**.



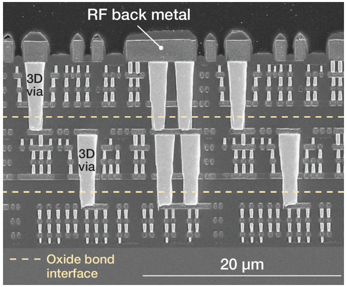
How to Access Silicon?

Can be very costly (plasma & laser) and destructive... But also accessible with simpler methods (like chemical/mechanical). More on [4].

1. **Decapsulation** (heat, acid, mechanical, plasma, laser...)
2. **Delayering** (chemical, abrasive, laser, plasma...)
3. **Cleaning** (ultrasound, acid...)


[1]

[2]

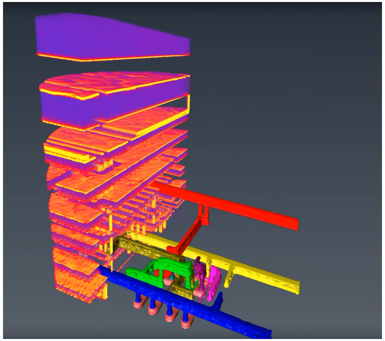


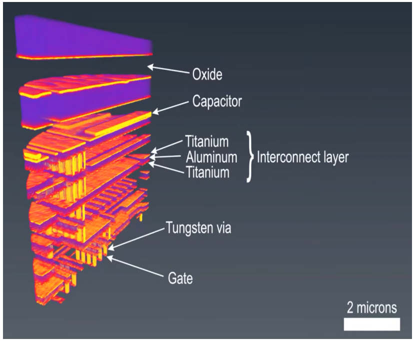
MIT

How to Retrieve Images?

Using each layer (invasive) or directly using the chip (non-invasive):

- Take very-high resolution images from **optical microscope** (basic, confocal) ;
- Scan from an **electron microscope** (SEM, TEM...) ;
- Generate a 3D model using **electron tomography** ;



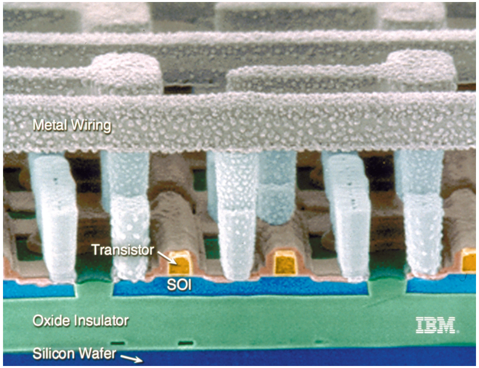


[1]

How to Perform the Analysis?

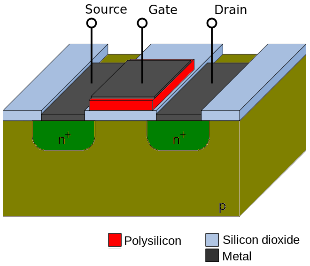
Overview:

1. Choose a **zone of interest**,
2. Identify each **gate type**, annotate, and place in a **"gate library"**,
3. Find other **gates instance** from gate library,
4. Link gates by tracing **wires and vias**,
5. Export to **netlist** (e.g. by translating each gate to VHDL/Verilog code).

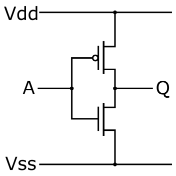


How to identify a transistor?

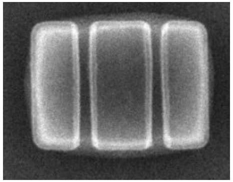
1. Search, at transistor layer, for **doped zones**.
2. Spot the **zebras**.
3. Use logic to identify the **type of each transistor** (e.g. PMOS are bigger to compensate with lower hole mobility).
4. Search for **wires** (to identify inputs and outputs).



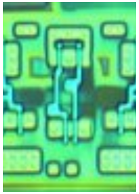
(NMOS, *Wikipedia*)

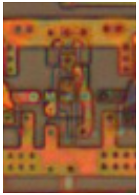


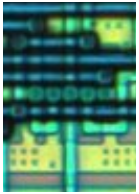
(Inverter, *Wikipedia*)


(PMOS [10])

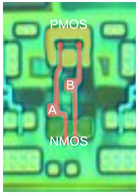
How to Identify a Gate?


Transistor layer

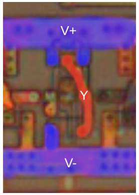

Logic layer


Metal layer

$\Downarrow$

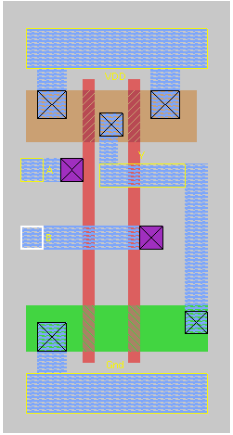


P & N zones and 2 inputs



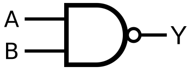
V+ & V-, and output

$\Rightarrow$



[7] $\Rightarrow$

NAND gate!



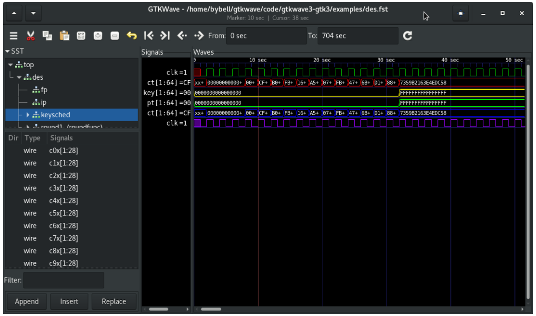
A	B	Y
0	0	1
1	0	1
0	1	1
1	1	0

How to Retrieve the Netlist from Analyzed Gates?

```
module jsrflipflop(q,qbar,clk,rst,sr);
    output reg q;
    output qbar;
    input clk, rst;
    input [1:0] sr;

    assign qbar = ~q;

    always @(posedge clk)
    begin
        if (rst)
            q <= 0;
        else
            case(sr)
                2'b00: q <= q;
                2'b01: q <= 0;
                2'b10: q <= 1;
                2'b11: q <= 1'bx;
            endcase
    end
endmodule
```

- Each gate can be described with **hardware description language (HDL)**, like **Verilog** or **VHDL**.
- **Wires & vias** can also be described.
- That's all we need to **obtain the netlist**!

We can, from HDL, **simulate the extracted netlist** and **find incoherence** (*example with gtkwave below*):


GTKWave - /home/bybell/gtkwave/code/gtkwave3-gtk3/examples/des.fst
Marker: 10 sec | Cursor: 38 sec

From: 0 sec To: 704 sec

```
clk=1
ct[1:64]=CF   xx+ 00000000000+ 00+ CF+ B0+ FB+ 16+ A5+ 07+ FB+ 47+ 6B+ D1+ 88+ 7359B2163E4EDC58
key[1:64]=00  0000000000000000                                         FFFFFFFFFFFFFFFF
pt[1:64]=00   0000000000000000                                         FFFFFFFFFFFFFFFF
ct[1:64]=CF   xx+ 00000000000+ 00+ CF+ B0+ FB+ 16+ A5+ 07+ FB+ 47+ 6B+ D1+ 88+ 7359B2163E4EDC58
clk=1
```

Dir	Type	Signals
	wire	c0x[1:28]
	wire	c1x[1:28]
	wire	c2x[1:28]
	wire	c3x[1:28]
	wire	c4x[1:28]
	wire	c5x[1:28]
	wire	c6x[1:28]
	wire	c7x[1:28]
	wire	c8x[1:28]
	wire	c9x[1:28]

How to Get the Algorithm/Specification from Netlist? [3]

After retrieving the **netlist**, we are left with a **huge and "unorganized" number of gates**. The **specification discovery** phase aims to **retrieve IC's algorithm/functionality** from the extracted netlist.

Using specific algorithms you can **automate some phase**:

- **Partitioning** of the netlist (*to retrieve a semblance of "code" structure*).
- **Recovery** of the registers (*if applicable*).
- **Identification** of the extracted "groups" (*partitions*) of the netlist.
- **Construction** of a library of netlist components from the identified "groups".

These algorithms **need to allow some degrees of error** from the netlist extraction. This phase is ∼analogous with **duplicated, standard & library functions identification** for **software engineering**. A nice open source tool for this is **HAL**[3] (compatible with Degate's outputs!).

[3] https://github.com/emsec/hal

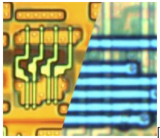
To Summarize

Image chip *(or SEM or 3D tomography)* → **Analyze Gates** → **Trace Wires/Vias** → **Generate Netlist** → **Retrieve Algorithm**

Image chip
- **Decapsulate**,
- Select **area of interest**,
- **Delayer**,
- Clean,
- Stitch.

[2]

Analyze Gates
- Analyze transistors,
- **Identify each type of gates**,
- Retrieve all instances of each gate (**pattern matching**).


[2]

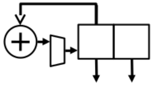
Trace Wires/Vias
- Identify **wires**,
- Identify **vias**,
- **Connect gates**.

[2]

Generate Netlist
- **Export netlist**,
- Gate simulation,
- Try to identify errors.

```
<?xml version="1.0"
<gate-library>
 <gate description=
  <ports>
   <port id="0" nam
   <port id="1" nam
```
[2]

Retrieve Algorithm
- Extract algorithm,
- **Discover specification**,
- Perform functional analysis[3],
- Build a **software emulator**,
- Try to **identify errors**.


[2]

Importance for Cybersecurity

How can we **trust software** if we **can't trust hardware** (e.g. "specialized" ASIC)?

- Is there any **vulnerability in the hardware implementation** of an algorithm (e.g. crypto standard with predictable initialization, bad randomness...)?
- Is there any **hardware Trojan** (e.g. placed by the foundry)?
- If there is a vulnerability/backdoor, **patching is impossible**, far **more impactful** than software vulnerabilities.

Some examples of vulnerabilities found thanks to silicon RE:

- *Legic Prime*, *NXP Hitag2*, *DECT DSC*, *CryptoRF*, *Atmel CryptoMemory* & *NXP Mifare Crypto-1* (∼2008, Nohl et al): **weak (or potentially weak) cryptographic ciphers**.
- Undisclosed ones?

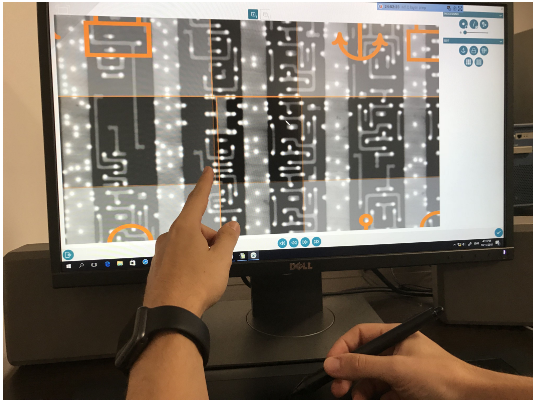
Available Tools & Products

Commercial products:

- **CHIPJUICE**: Extracting Data from Highly Encrypted ICs.
- *Internal tools*: for sure, there is a lot of them.

Open Source tools:

- **Degate**
- **psxrev**: SONY PlayStation PCB/chips reverse engineering.
- **Deroute**: Tool for untangling wires.
- **dietools**: Series of tools for die shot reverse-engineering.

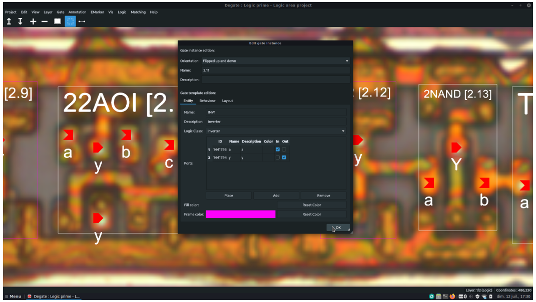


(Texplained)

Introduction

Degate is a multi-platform software for semi-automatic **Very-Large-Scale Integration (VLSI) chips reverse engineering** of digital logic in chips.

- ~70k LoC
- Supports Mac, Linux & Windows,
- Qt based,
- Multi-language support,
- Gate definition,
- Gate template, via & wire matching,
- Rule checks,
- ...


Degate : Logic prime - Logic area project
Project Edit View Layer Gate Annotation EMarker Via Logic Matching Help
[2.9]
22AOI [2.
a
y
b
c
y
[2.12]
2NAND [2.13]
Y
a
b
a
Edit gate instance
Entity Behaviour Layout
Place Add Remove
Reset Color
Reset Color
OK
Menu
Degate : Logic prime - L...

Gate instance edition:

Orientation: Flipped up and down

Name: 2.11

Description:

Gate template edition:

Name: INV1

Description: inverter

Logic Class: Inverter

Ports:

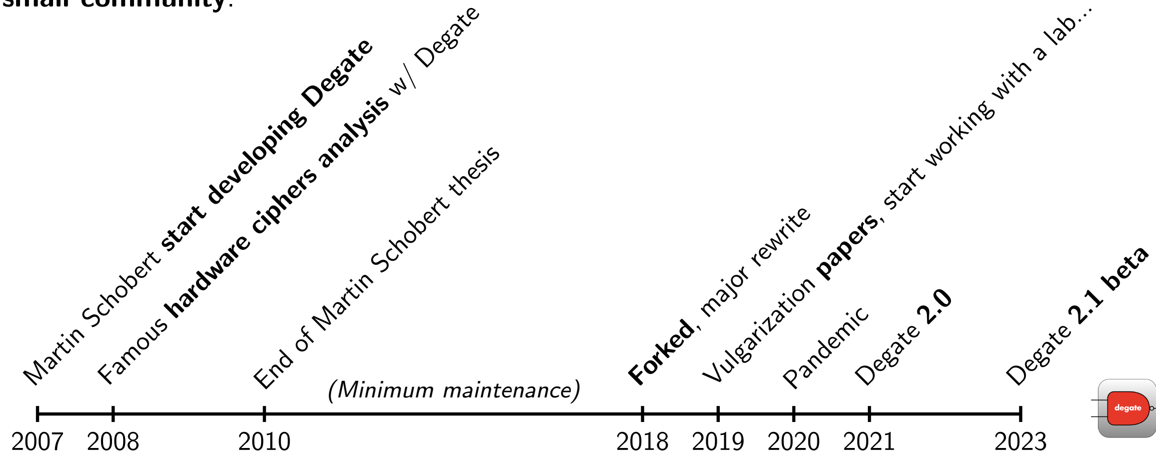
	ID	Name	Description	Color	In	Out
1	1441793	a	a		✓	
2	1441794	y	y			✓

Fill color:

Frame color: 🟪

History

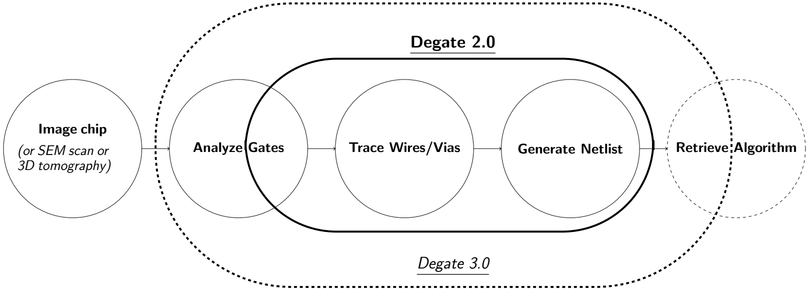
A long story, with **technical debt** and **major IC evolution** (in transistor count), along with a **small community**.


Martin Schobert **start developing Degate**
Famous **hardware ciphers analysis** w/ Degate
End of Martin Schobert thesis
(Minimum maintenance)
Forked, major rewrite
Vulgarization **papers**, start working with a lab...
Pandemic
Degate **2.0**
Degate **2.1 beta**
2007
2008
2010
2018
2019
2020
2021
2023

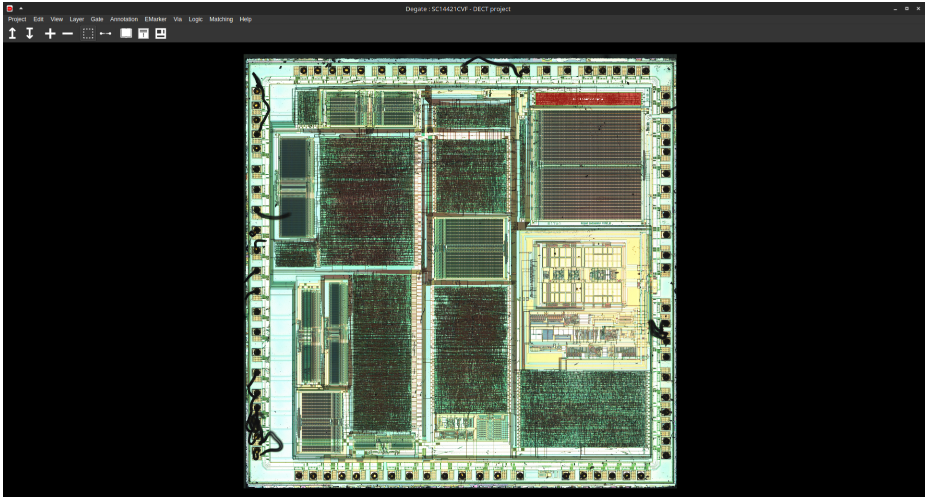
Usage

Degate help to reverse **VLSI chips** by creating an analyzed **gate library**, doing **template matching** to find gates instances from this library, **matching wires & vias**, **exporting netlist** and **navigating really huge images**.

Focus on **modern ICs** with **standard cells**, and supports **any 2D capture/imaging method** (SEM, optical...).



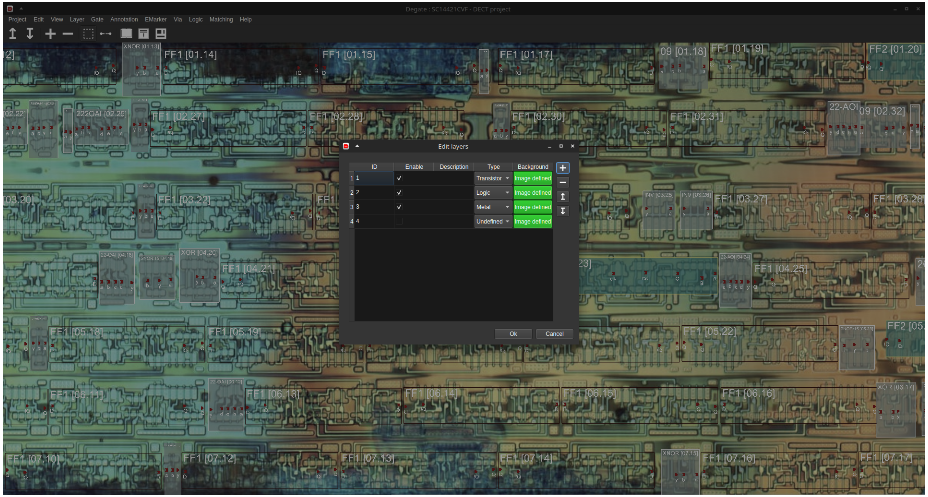
Small Demonstration


Degate : SC14421CVF - DECT project
Project Edit View Layer Gate Annotation EMarker Via Logic Matching Help

Overview of the chip, for zone of interest selection.

A sub-project can then be created on the zone of interest, and specific layers can be added (independent from the rest).

Small Demonstration

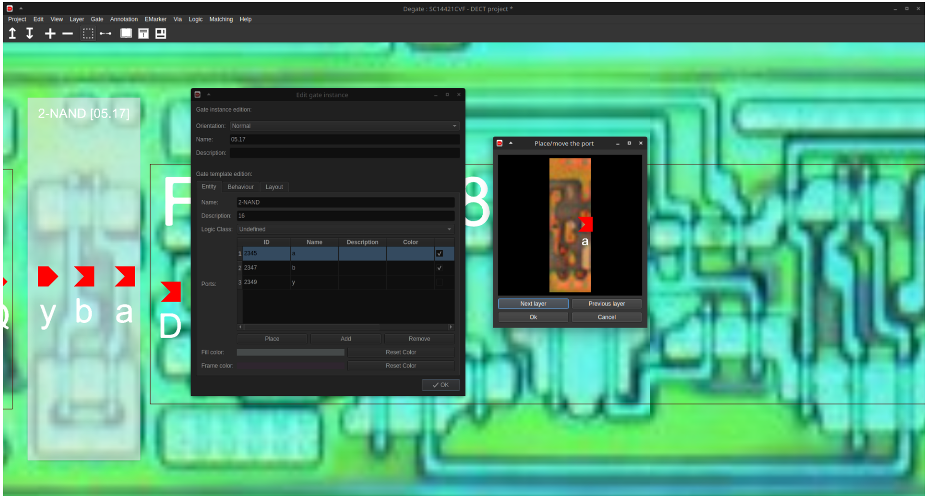

Degate : SC14421CVF - DECT project
Project Edit View Layer Gate Annotation EMarker Via Logic Matching Help
Edit layers
Ok
Cancel

	ID	Enable	Description	Type	Background
1	1	✓		Transistor	Image defined
2	2	✓		Logic	Image defined
3	3	✓		Metal	Image defined
4	4			Undefined	Image defined

Each sub-project can contains multiple layers (pre-aligned images).

Two project mode: 1. For smaller images, will convert each images in Degate's format (for fast access) and 2. New (WIP, beta) mode for huge images (load only partial tiles in RAM, and doesn't change/import initial file).

Small Demonstration



Gate instance edition:

Orientation: Normal
Name: 05.17
Description:

Gate template edition:

Entity | Behaviour | Layout

Name: 2-NAND
Description: 16
Logic Class: Undefined

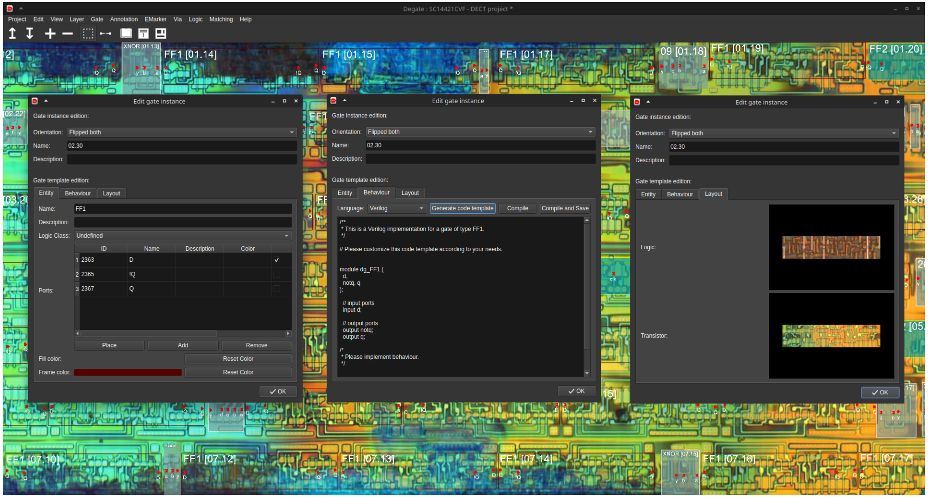
Ports:

	ID	Name	Description	Color	
1	2345	a			✓
2	2347	b			✓
3	2349	y			

Fill color:
Frame color:

Each gate can be described with VHDL/Verilog, have a list of port (placed on image), a type associated etc.

Small Demonstration



Gate instance edition:

Orientation: Flipped both
Name: 02.30
Description:

Gate template edition:

Name: FF1
Description:
Logic Class: Undefined

Ports:

	ID	Name	Description	Color	
1	2363	D			✓
2	2365	!Q			
3	2367	Q			

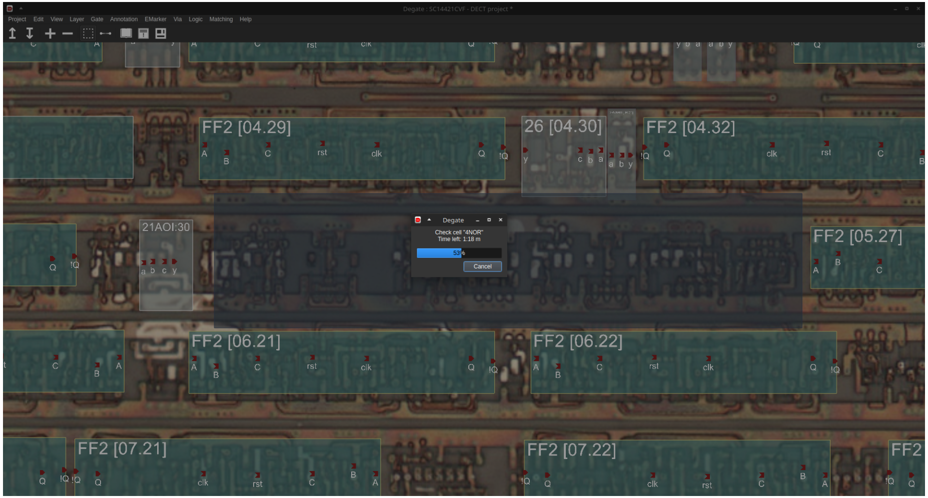
Fill color:
Frame color: 🟥

Language: Verilog · Generate code template · Compile · Compile and Save

```
/**
 * This is a Verilog implementation for a gate of type FF1.
 */

// Please customize this code template according to your needs.


module dg_FF1 (
  d,
  notq, q
);

  // input ports
  input d;

  // output ports
  output notq;
  output q;

/*
 * Please implement behaviour.
 */
```

Each identified gate (from the gate library) can be matched manually or using template matching algorithms.

Small Demonstration

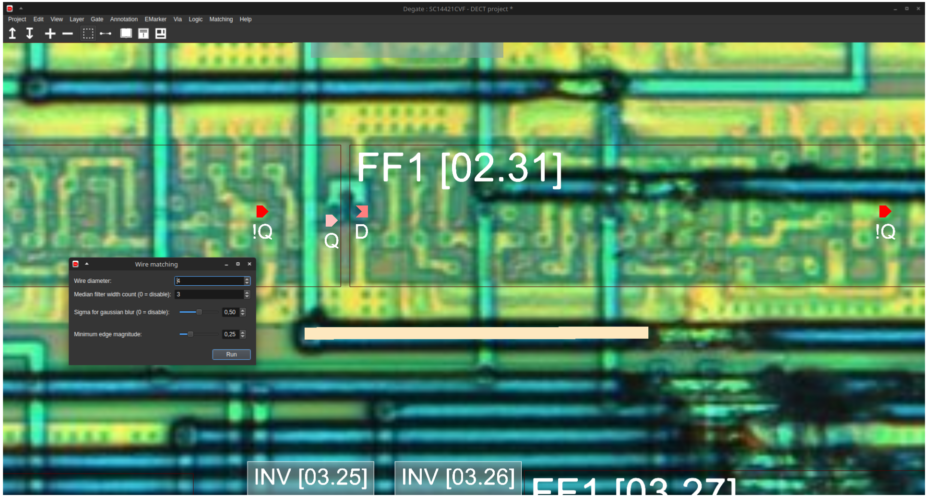


Check cell "4NOR"
Time left: 1:18 m
53%

Template matching (will soon be ported to OpenCV) will use gate library to automate gate identification.

Currently it uses normalized cross-correlation (with some more steps).

Small Demonstration

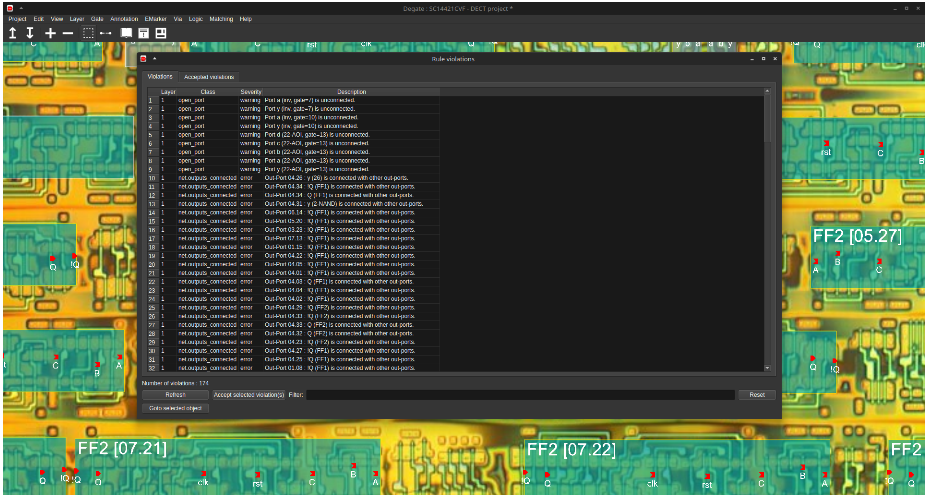

Degate : SC14421CVF - DECT project *
Project Edit View Layer Gate Annotation EMarker Via Logic Matching Help
FF1 [02.31]
!Q
Q
D
!Q
INV [03.25]
INV [03.26]
Wire matching
Run

Wire diameter: 4
Median filter width count (0 = disable): 3
Sigma for gaussian blur (0 = disable): 0,50
Minimum edge magnitude: 0,25

Wire matching, and specifically port inter-connection, is the real challenge (and very error prone).

Currently it uses zero crossing edge detection.

Small Demonstration

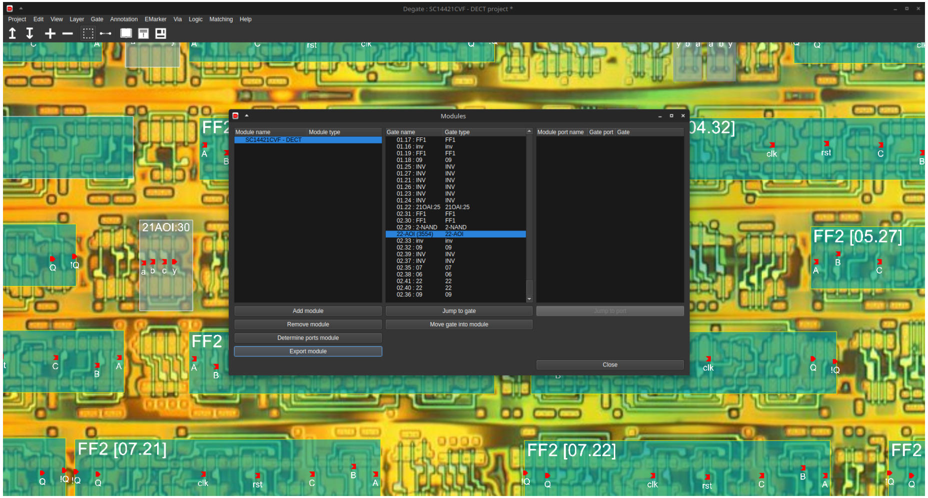

Degate : SC14421CVF - DECT project *
Project Edit View Layer Gate Annotation EMarker Via Logic Matching Help
Rule violations
Violations
Accepted violations
FF2 [05.27]
FF2 [07.21]
FF2 [07.22]
FF2
Refresh
Accept selected violation(s)
Filter:
Reset
Goto selected object

	Layer	Class	Severity	Description
1	1	open_port	warning	Port a (inv, gate=7) is unconnected.
2	1	open_port	warning	Port y (inv, gate=7) is unconnected.
3	1	open_port	warning	Port a (inv, gate=10) is unconnected.
4	1	open_port	warning	Port y (inv, gate=10) is unconnected.
5	1	open_port	warning	Port d (22-AOI, gate=13) is unconnected.
6	1	open_port	warning	Port c (22-AOI, gate=13) is unconnected.
7	1	open_port	warning	Port b (22-AOI, gate=13) is unconnected.
8	1	open_port	warning	Port a (22-AOI, gate=13) is unconnected.
9	1	open_port	warning	Port y (22-AOI, gate=13) is unconnected.
10	1	net.outputs_connected	error	Out-Port 04.26 : y (26) is connected with other out-ports.
11	1	net.outputs_connected	error	Out-Port 04.34 : !Q (FF1) is connected with other out-ports.
12	1	net.outputs_connected	error	Out-Port 04.34 : Q (FF1) is connected with other out-ports.
13	1	net.outputs_connected	error	Out-Port 04.31 : y (2-NAND) is connected with other out-ports.
14	1	net.outputs_connected	error	Out-Port 06.14 : !Q (FF1) is connected with other out-ports.
15	1	net.outputs_connected	error	Out-Port 05.20 : !Q (FF1) is connected with other out-ports.
16	1	net.outputs_connected	error	Out-Port 03.23 : !Q (FF1) is connected with other out-ports.
17	1	net.outputs_connected	error	Out-Port 07.13 : !Q (FF1) is connected with other out-ports.
18	1	net.outputs_connected	error	Out-Port 01.15 : !Q (FF1) is connected with other out-ports.
19	1	net.outputs_connected	error	Out-Port 04.22 : !Q (FF1) is connected with other out-ports.
20	1	net.outputs_connected	error	Out-Port 04.05 : !Q (FF1) is connected with other out-ports.
21	1	net.outputs_connected	error	Out-Port 04.01 : !Q (FF1) is connected with other out-ports.
22	1	net.outputs_connected	error	Out-Port 04.03 : Q (FF1) is connected with other out-ports.
23	1	net.outputs_connected	error	Out-Port 04.04 : !Q (FF1) is connected with other out-ports.
24	1	net.outputs_connected	error	Out-Port 04.02 : !Q (FF1) is connected with other out-ports.
25	1	net.outputs_connected	error	Out-Port 04.29 : !Q (FF2) is connected with other out-ports.
26	1	net.outputs_connected	error	Out-Port 04.33 : !Q (FF2) is connected with other out-ports.
27	1	net.outputs_connected	error	Out-Port 04.33 : Q (FF2) is connected with other out-ports.
28	1	net.outputs_connected	error	Out-Port 04.32 : Q (FF2) is connected with other out-ports.
29	1	net.outputs_connected	error	Out-Port 04.23 : !Q (FF2) is connected with other out-ports.
30	1	net.outputs_connected	error	Out-Port 04.27 : !Q (FF1) is connected with other out-ports.
31	1	net.outputs_connected	error	Out-Port 04.25 : !Q (FF1) is connected with other out-ports.
32	1	net.outputs_connected	error	Out-Port 01.08 : !Q (FF1) is connected with other out-ports.

Number of violations : 174

Helpers are available, like rudimentary (but to be improved) rule checking (e.g. for coherency).

Small Demonstration



Module name	Module type
SC14421CVF - DECT	

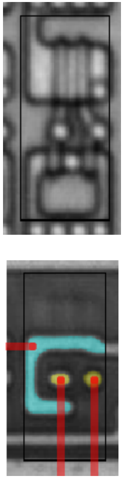
Gate name	Gate type
01.17 : FF1	FF1
01.16 : inv	inv
01.19 : FF1	FF1
01.18 : 09	09
01.25 : INV	INV
01.27 : INV	INV
01.21 : INV	INV
01.26 : INV	INV
01.23 : INV	INV
01.24 : INV	INV
01.22 : 21OAI:25	21OAI:25
02.31 : FF1	FF1
02.30 : FF1	FF1
02.29 : 2-NAND	2-NAND
22-AOI (3554)	22-AOI
02.33 : inv	inv
02.32 : 09	09
02.39 : INV	INV
02.37 : INV	INV
02.35 : 07	07
02.38 : 06	06
02.41 : 22	22
02.40 : 22	22
02.36 : 09	09

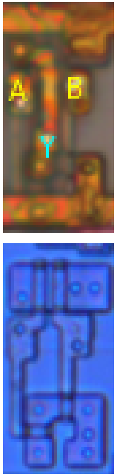
Module port name	Gate port	Gate

Everything can be organized in "module", exported individually (in Verilog/VHDL), etc... "Divide et impera".

Engineering Challenges

- Gate template, wires & vias **matching**.
- Very **huge images** handling.
- **Error** recovery/acceptance/identification.
- Multiple possible **image format** (e.g. .tiff, .png...) & **image source** (e.g. SEM, confocal...).
- 10+ years **old software** (mix of old & new C++).
- **Collaborative** analysis.
- Integrated **gate analyzer**.
- Explicit full **netlist exporter**.


MIFARE NAND[2]



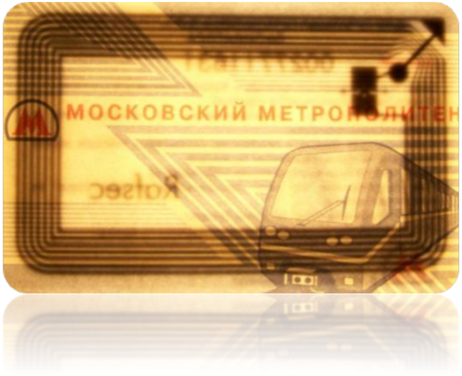
LEGIC NAND[2]

Research Challenges

- **3D capture**, imply rethinking Degate (New 3D mode? New software? Really accessible?), and **new algorithms** (e.g. for matching, tracing and gate identification).
- **Machine learning**/better algorithms for:
  - Auto-**vectorization** ;
  - Gate auto **identification** (from vectorized analysis) ;
  - Gate auto **wiring** ;
  - Auto vias & wires **identification**.
- Take advantage of certain capture methods such as **SEM** which makes **automation easier**.
- Making the **field more accessible** (more automation, new abstractions for analysis, communication...).
- Use Degate for **advanced analysis** and **published results**.

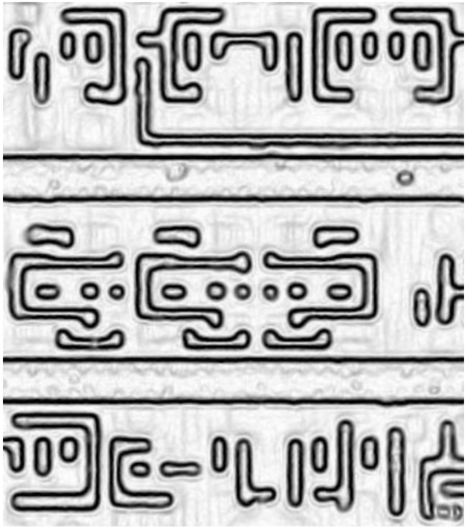
MIFARE Classic Chip [2]

- **RFID card** from NXP launched in 1994.
- Used the **Crypto1 cypher** (until MIFARE Classic EV1, that are using **Hitag2** cipher).
- **Proprietary encryption** algorithm (stream cipher), security by obscurity.
- Cryto1 cipher is only **implemented in hardware**.
- Used (back in 2008) in more than **3.5 billions cards** (including many building access control systems).



A **huge target** with a **suspicious cypher** and **security standards**?

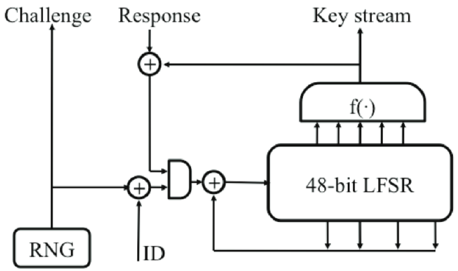
Degate's origins [5]



- K. Nohl & Starbug **reverse-engineered the Crypto1 cypher** from MIFARE Classic chip in 2007.
- Used **acetone** to dissolve the RFID cards.
- Used **manual polishing** for delayering.
- Image a total of **6 layers**.
- Identify zone of interest, **searching for 48-bit register** & group of XOR gates.
- Used **standard optical microscope** (500x) & hugin tool for stitching.
- Identified **around 70 types of gates**.
- Used **home-made scripts** (which became the base of Degate) for **template matching** to identify all gates.
- **Manually** reconstructed **connections** between gates.
- Made a **script** to help detecting **wires & vias**.

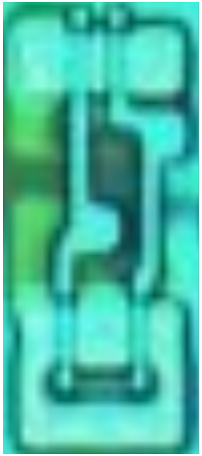
Consequences [5]

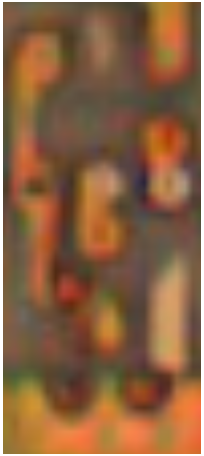
- Using the reverse-engineering results and protocol analysis, authors found **multiple weakness** in the cipher:
    - The cipher is vulnerable to **brute force** attack, key is too small.
    - RNG is predictable, it uses a 16-bit LFSR (linear feedback shift register) **initialized with constant value** and reset at each power-up.
    - There is **only one secret key** for each ID that can result to a specific session key, and all shifts are linear.
- Meaning that just by **sniffing interactions** with the card and the reader, we can compute the key and **retrieve all the data** of the card.
- NXP release a retro-compatible & "hardened" version of the Cipher (Hitag2), which was also weak, MIFARE Classic were **"discontinued" in 2015**.

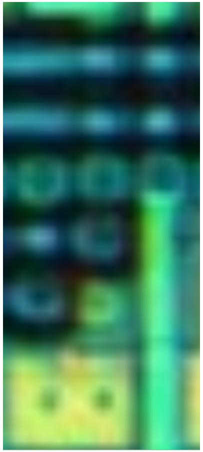


- Authors **analyzed other RFID devices** after.
- **Degate was created** from this analysis, used for other RFID devices reverse-engineering and open-sourced in 2008.

Which gate is this?


Transistor layer


Logic layer


Metal layer

References I

[1] Mirko Holler, Manuel Guizar-Sicairos, Esther H. R. Tsai, Roberto Dinapoli, Elisabeth Müller, Oliver Bunk, Jörg Raabe, and Gabriel Aeppli.
High-resolution non-destructive three-dimensional imaging of integrated circuits.
Nature, 543(7645):402–406, March 2017.

[2] Starbug Karsten Nohl.
Pacsec silicon conference.
2009.

[3] Nils Albartus Ran Ginosara Avi Mendelson Leonid Azriel, Julian Speith and Christof Paar.
Azriel and julian speith and nils albartus and ran ginosara and avi mendelson and christof paar.
Cryptology ePrint Archive, Paper 2021/1278, 2021.

References II

[4] John McMaster.
Siliconpr0n, https://siliconpr0n.org/.

[5] Karsten Nohl, David Evans, and Henryk Plotz.
Reverse-Engineering a Cryptographic RFID Tag.
page 9.

[6] Martin Schobert.
Gnu software degate.
Webpage: http://www.degate.org.

[7] Berlin Security Research Labs.
Siliconzoo, http://siliconzoo.org.

References III

[8] Ken Shirriff.
Ken shirriff's blog, https://www.righto.com/.

[9] Mikhail Svarichevsky.
Zeptobars, https://zeptobars.com/en/.

[10] Zonenberg Andrew Yener Bulent.
Csci 4974/6974 hardware reverse engineering, 2014.