LABORATOIRE HUBERT CURIEN
UMR • CNRS • 5516 • SAINT-ETIENNE

life.augmented

Assessing Side-Channel Leakages

Simulating Traces with Open-Source Tools

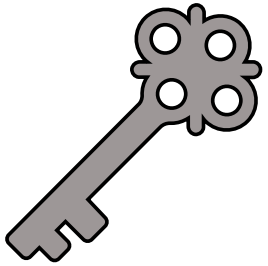
Mateus Simões[1, 2], Lilian Bossuet[2], Nicolas Bruneau[1], Vincent Grosso[2], Patrick Haddad[1], Thomas Sarno[1]

[1] STMicroelectronics, 13106 Rousset, France
[2] Laboratoire Hubert Curien, CNRS, 42000 Saint-Étienne, France

November 13, 2023

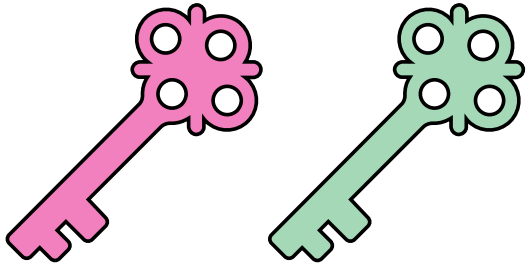
Hardware masking against side-channel attacks

Let us consider `secret` data, such as the private key.



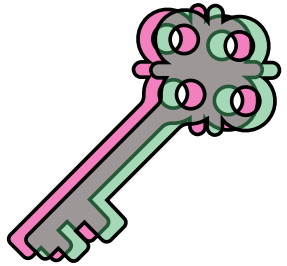
```
secret ← the private key
```

Masking[1] divides the `secret` into various random shares.



```
share_1 ← random
share_2 ← secret ⊕ share_1
```

After computing, we can combine the shares to reveal the `secret`.



```
secret = share_1 ⊕ share_2
```

- \+ Masking is a strong countermeasure against side-channel attacks.
- \+ The security increases exponentially with each additional share.
- \+ To reveal the secret, it is ideal to possess knowledge of all the shares.
- \- While the principle may seem simple and effective, the practical implementation of masking can be complex and challenging.

It is not straightforward.

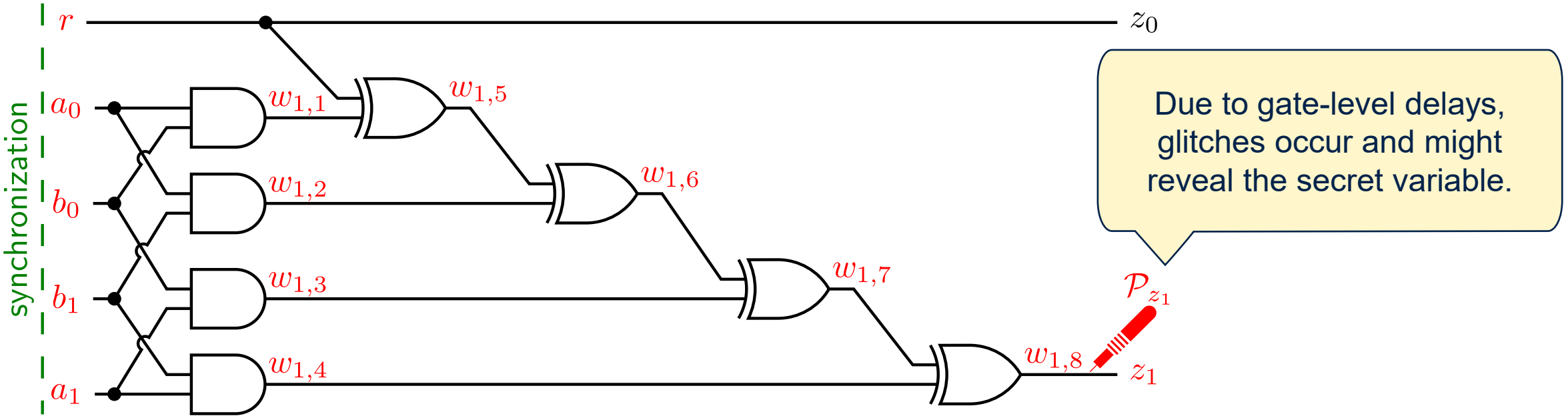
The design process may introduce side-channel vulnerabilities.

life.augmented

[1] L. Goubin et al. “DES and Differential Power Analysis (The "Duplication" Method)”. In CHES 1999.

The Trichina's multiplication gadget[1]

To protect $\boldsymbol{A}$ and $\boldsymbol{B}$, they are each divided into two shares, $(\boldsymbol{a_0}, \boldsymbol{a_1})$ and $(\boldsymbol{b_0}, \boldsymbol{b_1})$, and an additional random bit, $\boldsymbol{r}$, is needed for the process.



Due to gate-level delays, glitches occur and might reveal the secret variable.

Secure under the probing model…

… but vulnerable in the presence of glitches.

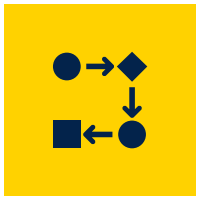
[1] E. Trichina. "Combinational Logic Design for AES SubByte Transformation on Masked". In IACR Cryptol. ePrint 2003.

life.augmented

Masking security verification processes

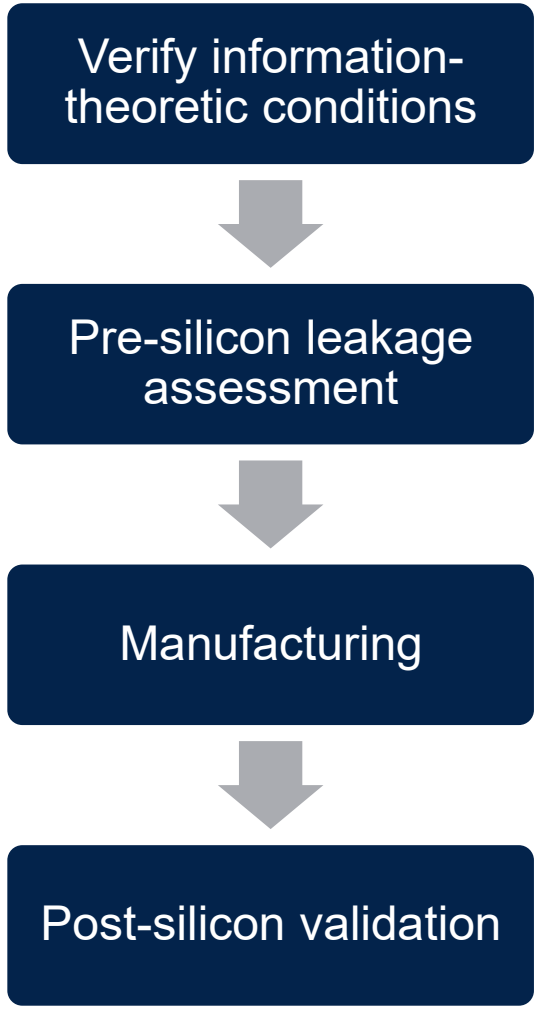
Verification of theoretic security levels may not be feasible in certain cases.

Manual verification is both prone to errors and time-consuming, thereby posing a challenge for scaling up to larger systems.



Gadget-level verification tools have limited capability to assess the security of complex systems, such as algorithmic masking.

EDA tools provide a range of features and functionalities to assess the side-channel leakages of masking implementations.



life.augmented

Summary

Physical hazards – e.g., glitches – are source of exploitable side-channel leakages[1].

The shares must be statistically independent to compose several masked sub-blocks effectively.

An inaccurate leakage model may result in vulnerable designs.

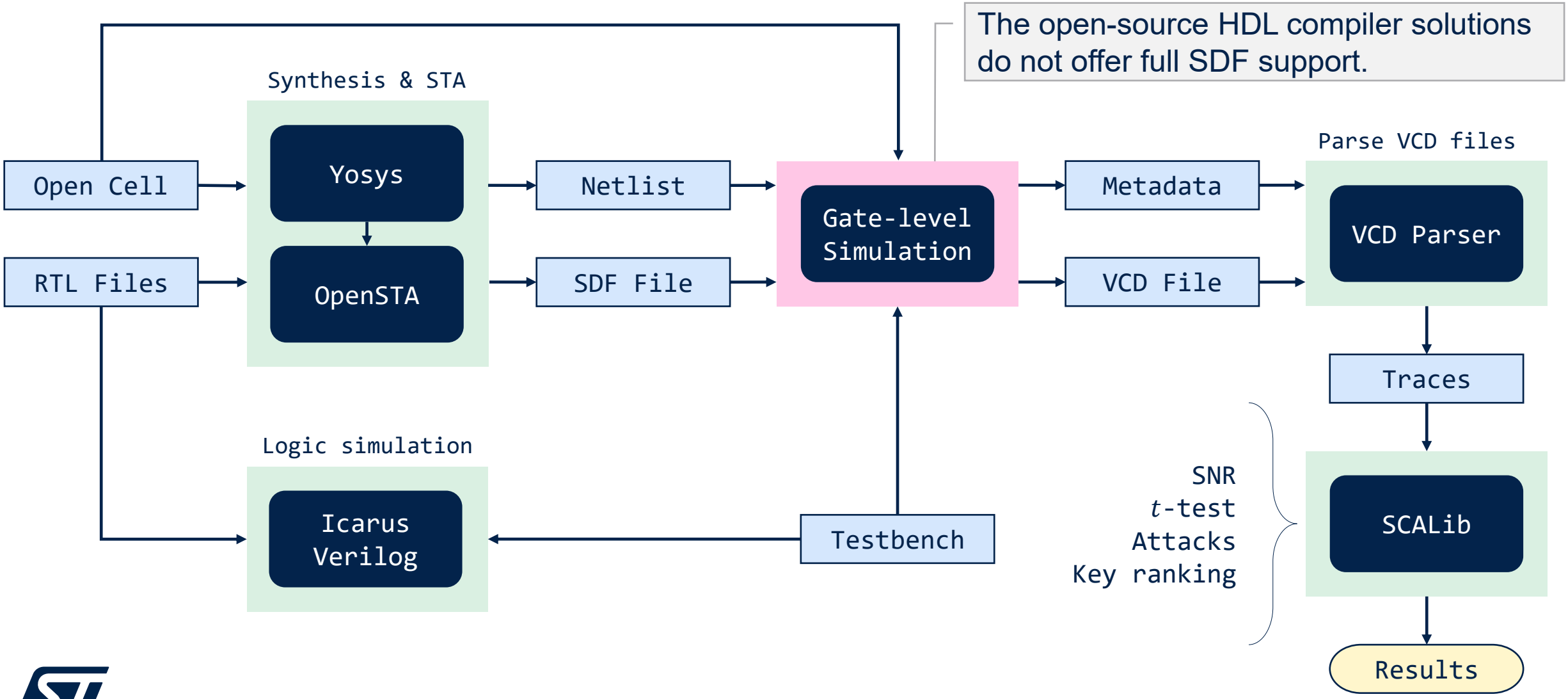
Verifying security on complex systems with algorithmic-level hardware masking can be challenging.

How to evaluate the security of complex masking implementations using (open-source) EDA tools ?

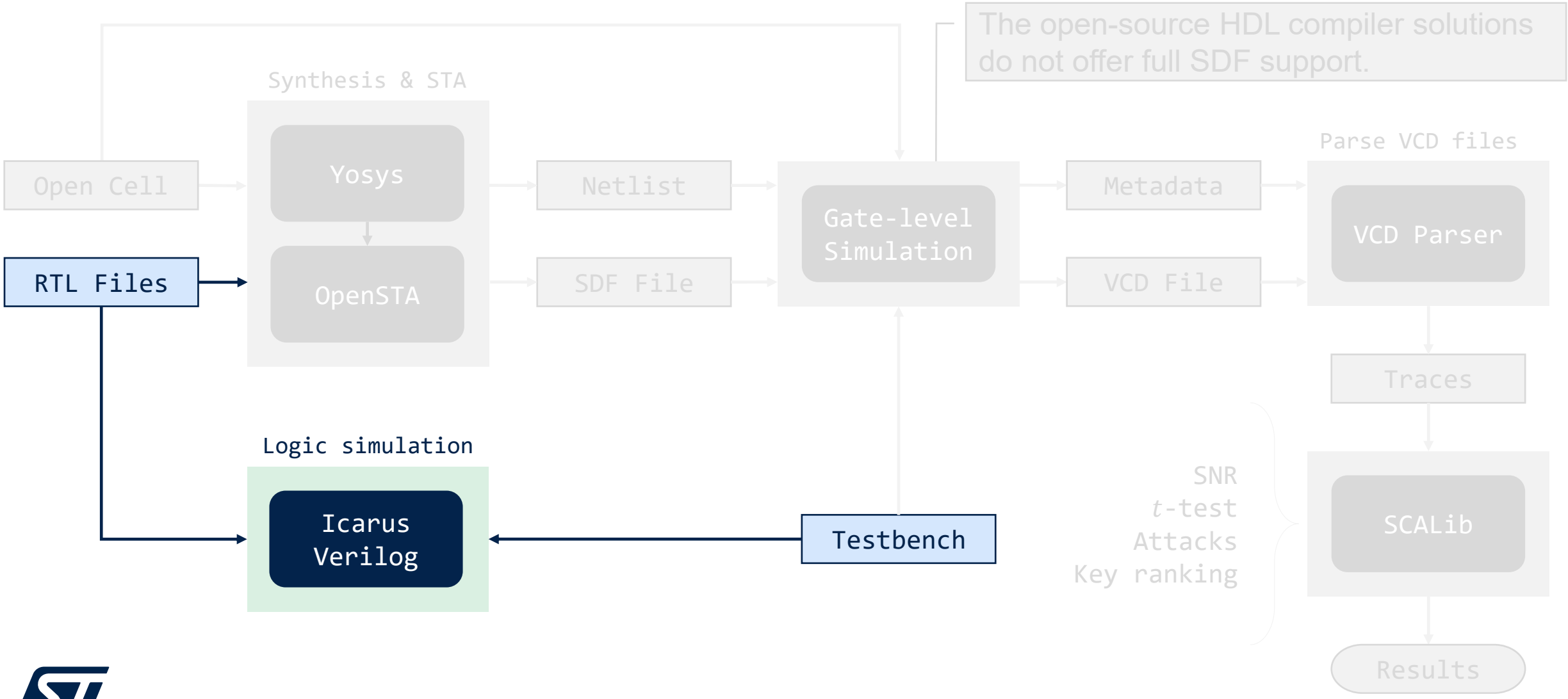
life.augmented

[1] S. Mangard et al. “Side-Channel Leakage of Masked CMOS Gates”. In CR-RSA 2005.

The (quasi-open-source) design flow



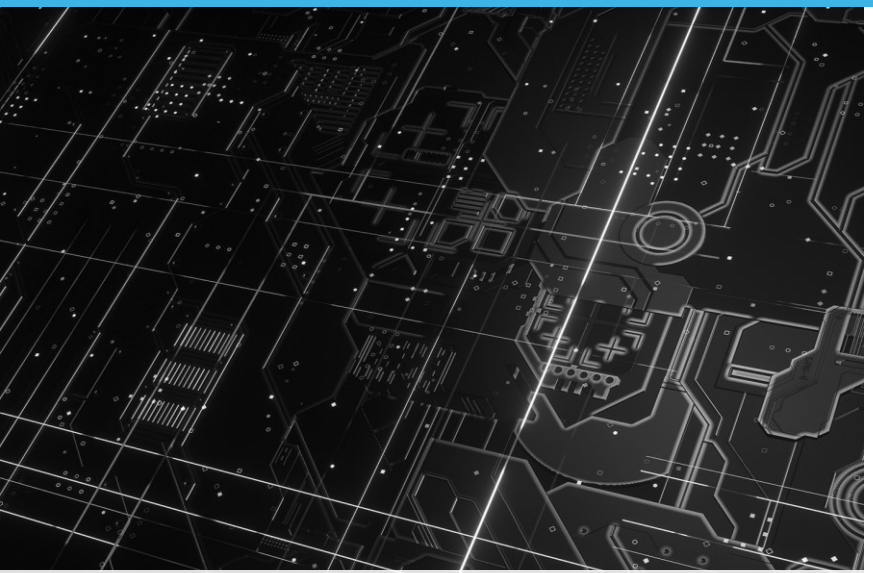
The (quasi-open-source) design flow



Logic simulation

Icarus Verilog

Open Source



Key applications

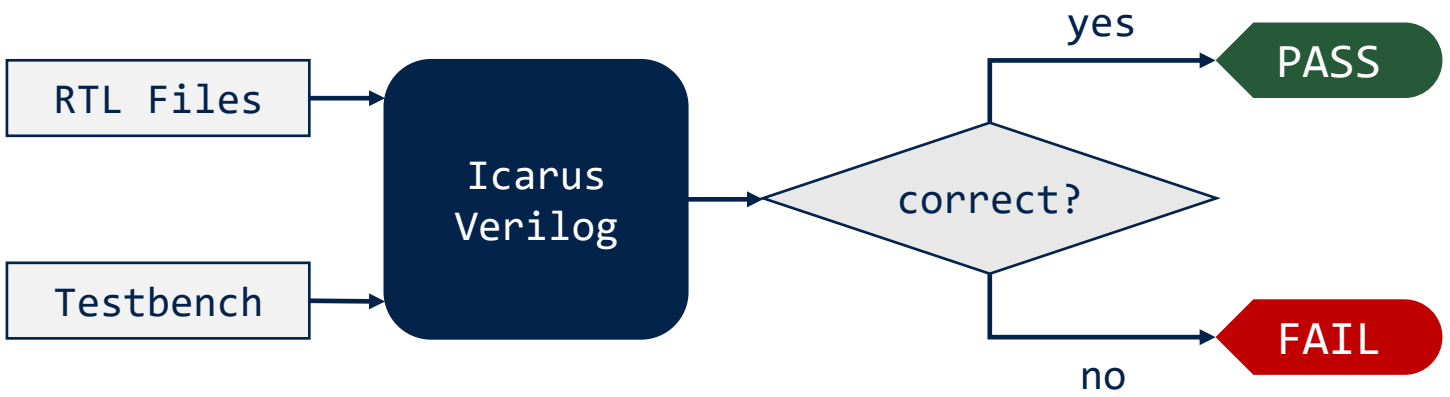
- Hardware description language compiler.
- Logic simulation with VPI.
- Behavioral validation.

hello.v

```
module hello;
 initial begin
  $display("Hello, World");
  $finish;
 end
endmodule
```

Terminal

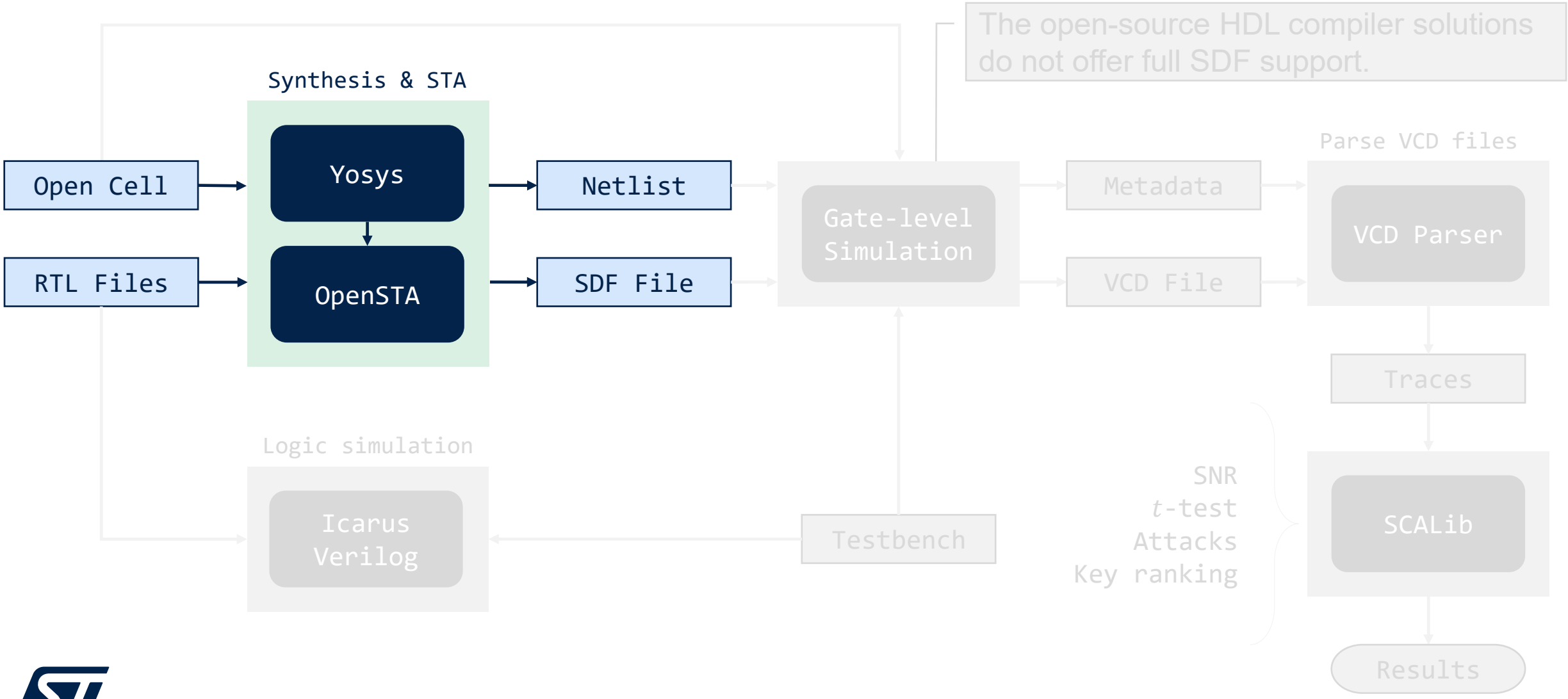
```
> iverilog -o hello hello.v
> vvp hello
Hello, World
```



S. Williams. “The ICARUS Verilog Compilation”. In GitHub: Icarus Verilog

life.augmented

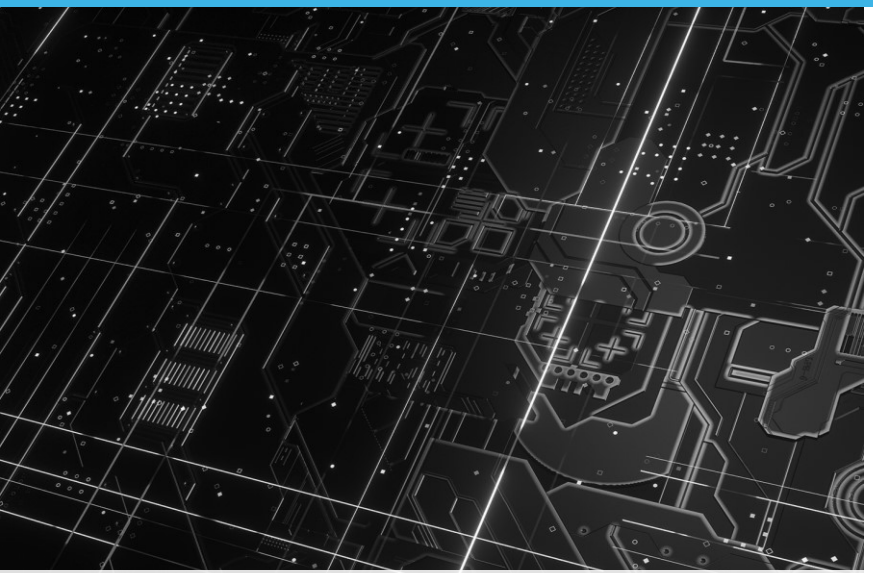
The (quasi-open-source) design flow



Verilog RTL synthesis

Yosys Open Synthesis Suite

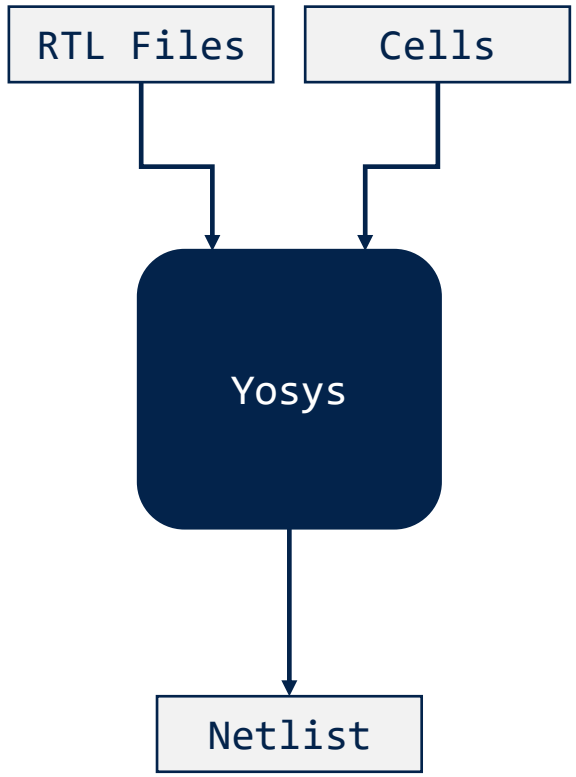
Open Source



Key applications

- Process almost any synthesizable Verilog-2005 design.
- Mapping to ASIC standard cell libraries.
- Design reports.

```
# read design
read_verilog mydesign.v

# generic synthesis
synth -top mytop

# mapping library cells
dfflibmap -liberty mycells.lib
abc -liberty mycells.lib
clean

# write synthesized design
write_verilog synth.v
```

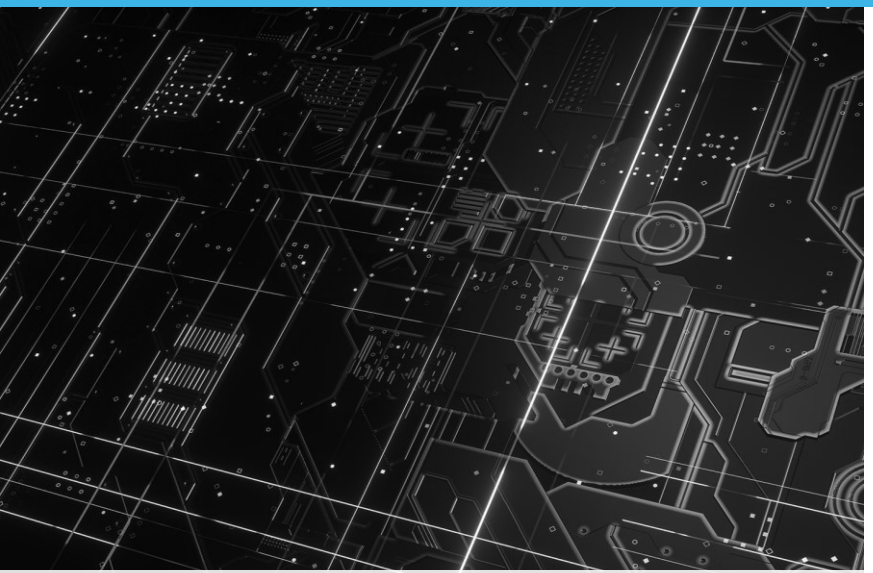


C. Wolf. “Yosys Open SYnthesis Suite”. In https://yosyshq.net/yosys/

Static Timing Analysis (STA)

OpenSTA: Parallax Static Timing Analyzer

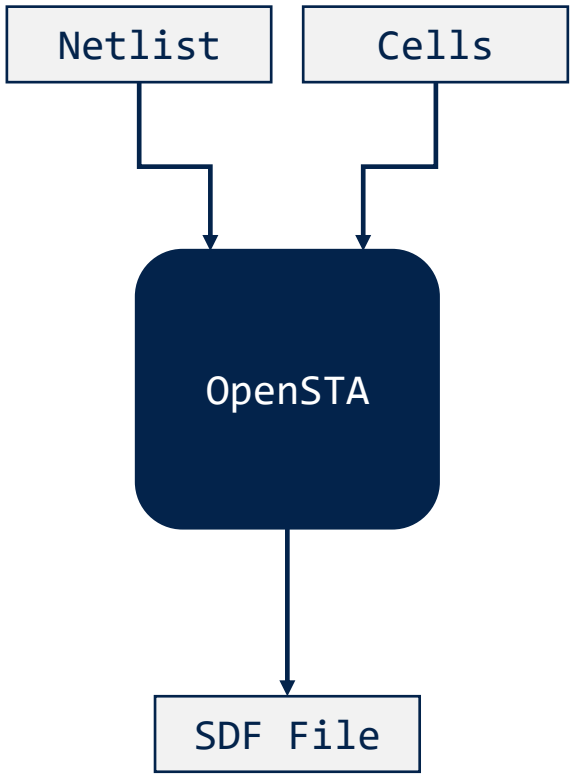
Open Source



Key applications

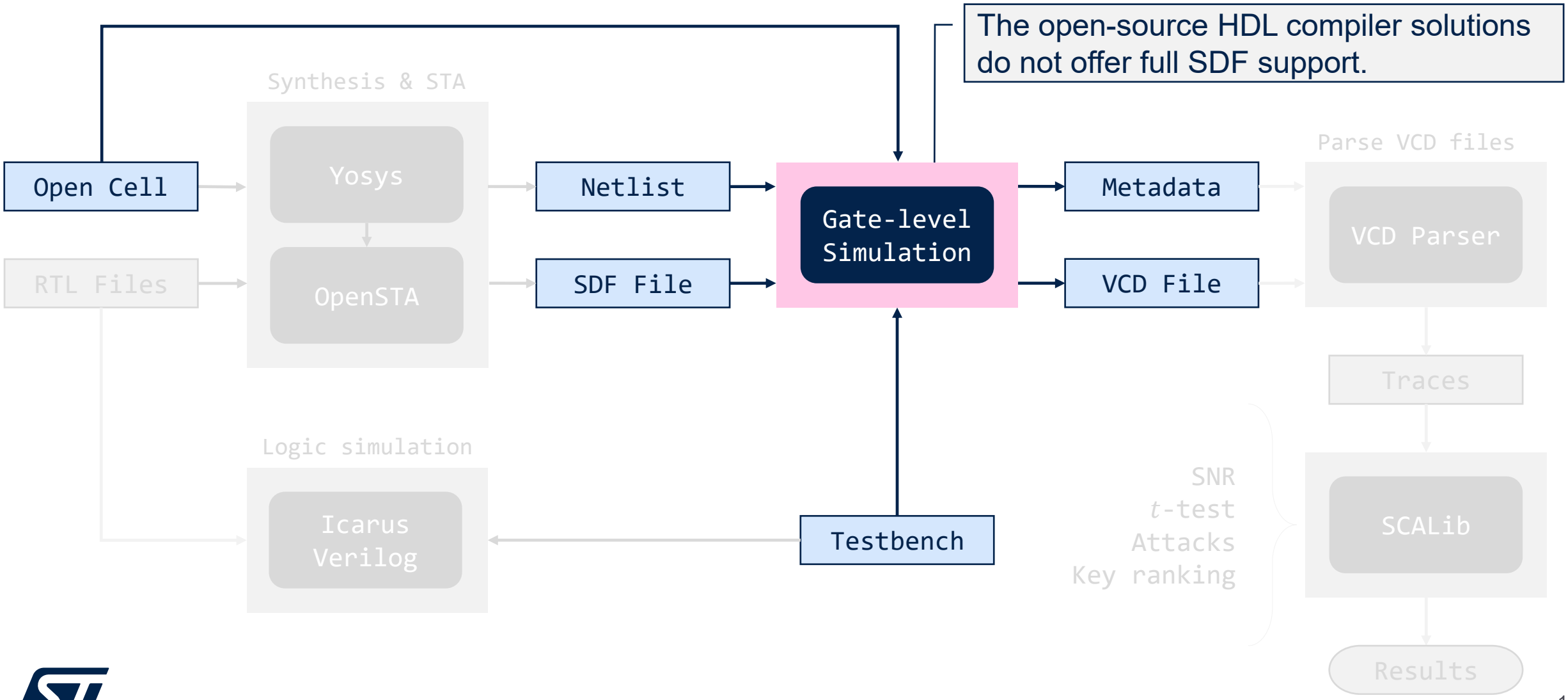
- Verify the timing of a design using standard file formats.
- Generate gate-level delay files (SDF).
- Timing reports.

```
# read library cells
read_liberty mycells.lib

# read design
read_verilog mydesign.v
link_design mydesign
create_clock -period 10 clock_i

# report timing
report_checks > timing.log

# write sdf file
write_sdf mydesign.sdf
```



J. Cherry. “OpenSTA: Parallax Static Timing Analyzer”. In GitHub: OpenSTA

The (quasi-open-source) design flow



Gate-level simulation with open-source tools

SDF back-annotation is necessary to take glitches into account.

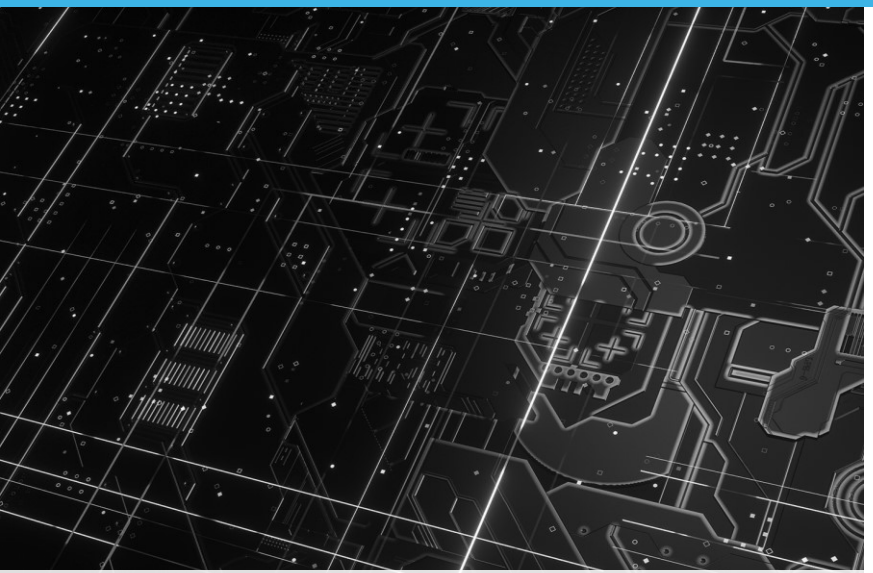
1	**Icarus Verilog**	Poor support for SDF back-annotation.
2	**Tachyon DA's CVC**	There is better support for SDF back-annotation, but it is still limited.
3	**Verilator**	There is a complete lack of support for this feature.

Hence, we rely on commercial tools for this gate-level task.

Gate-level verification

Gate-level simulation with back-annotated delay

Commercial

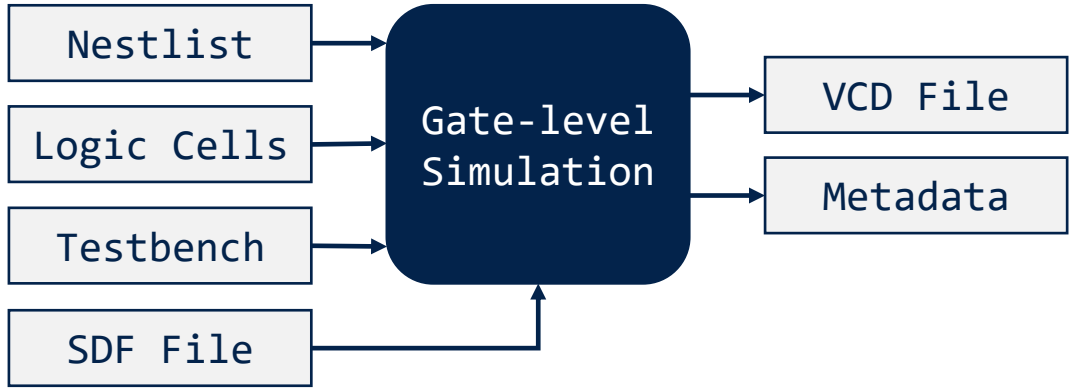


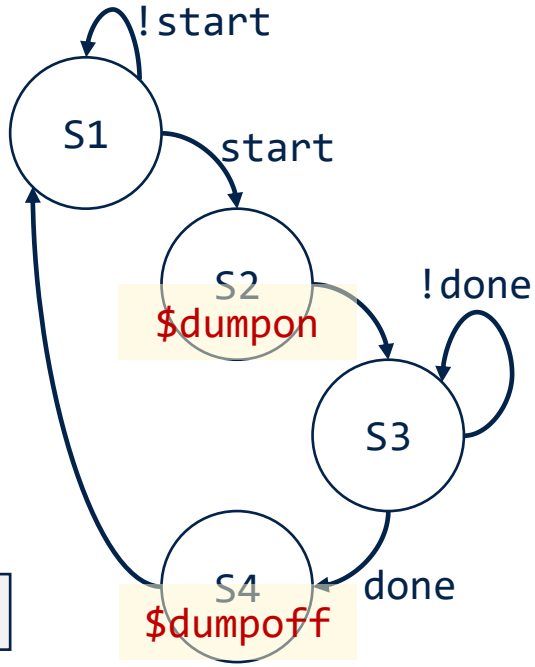
Key applications

- SDF back-annotation.
- Post-synthesis verification.
- VCD generation.

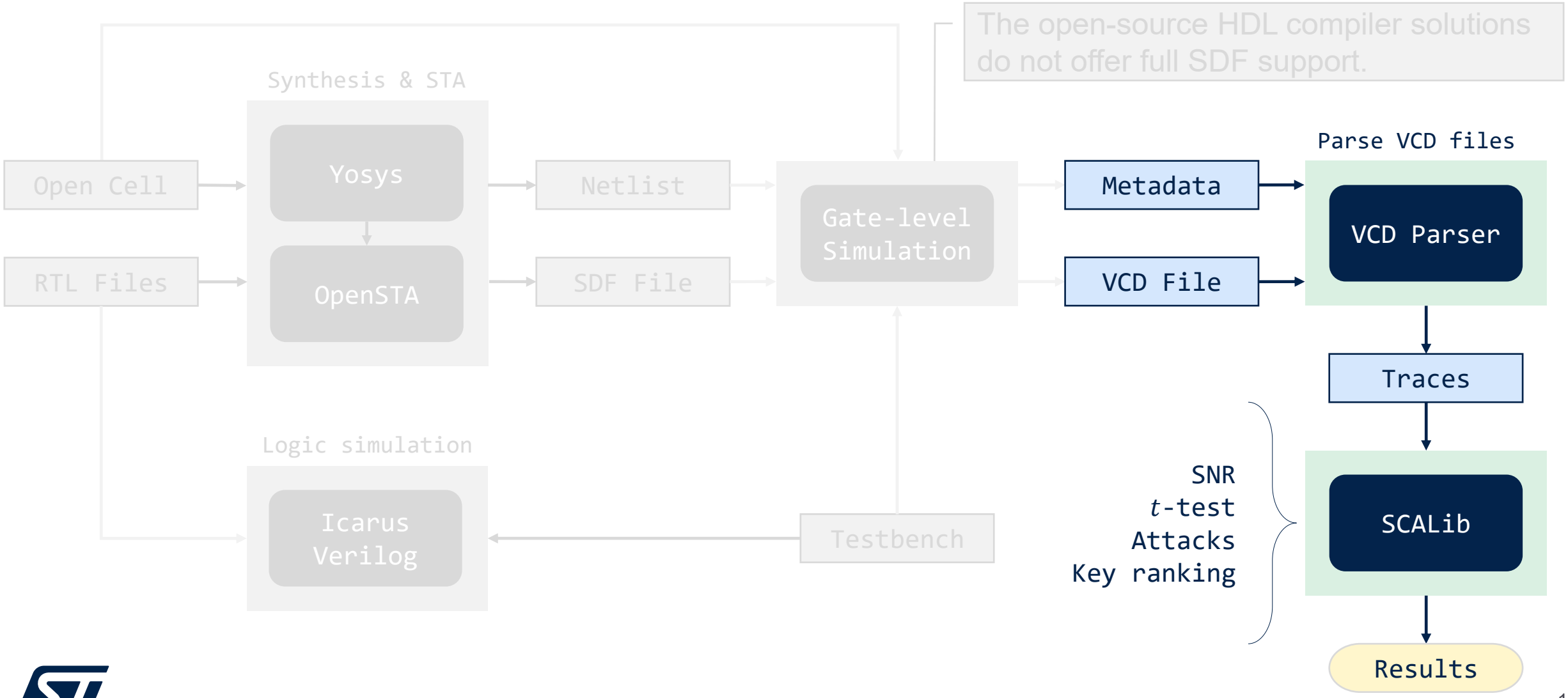
device_tb.v

```
module device_tb;
device DUT(…);
initial begin
  $sdf_annotate(“delay.sdf”, DUT);
 end
endmodule
```





The (quasi-open-source) design flow



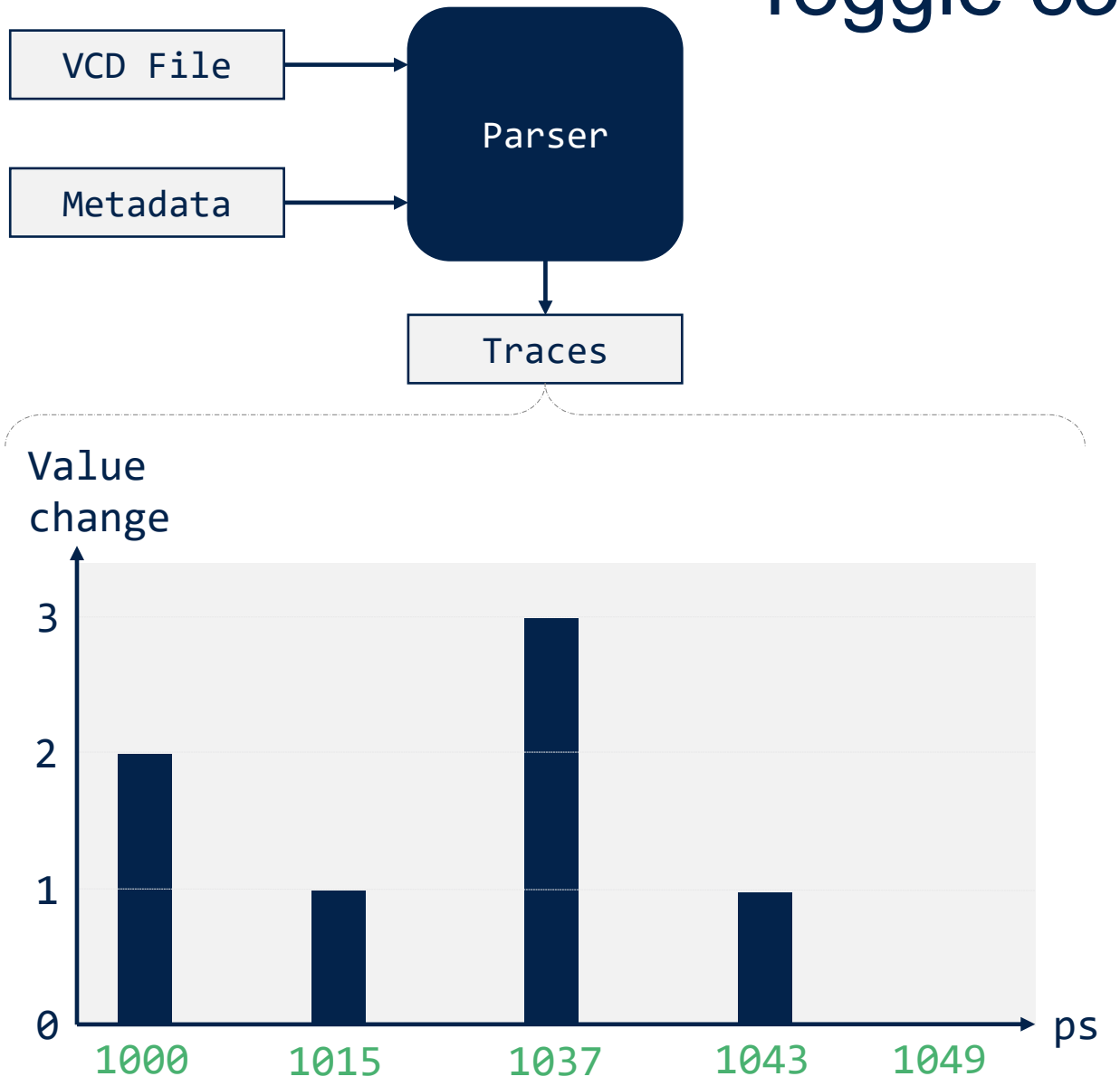
Toggle count

VCD File

```
#1000
$dumpon
1!
0“
#1015
1#
#1037
1$
0%
0&
#1043
0#
#1049
$dumpoff
```

Metadata

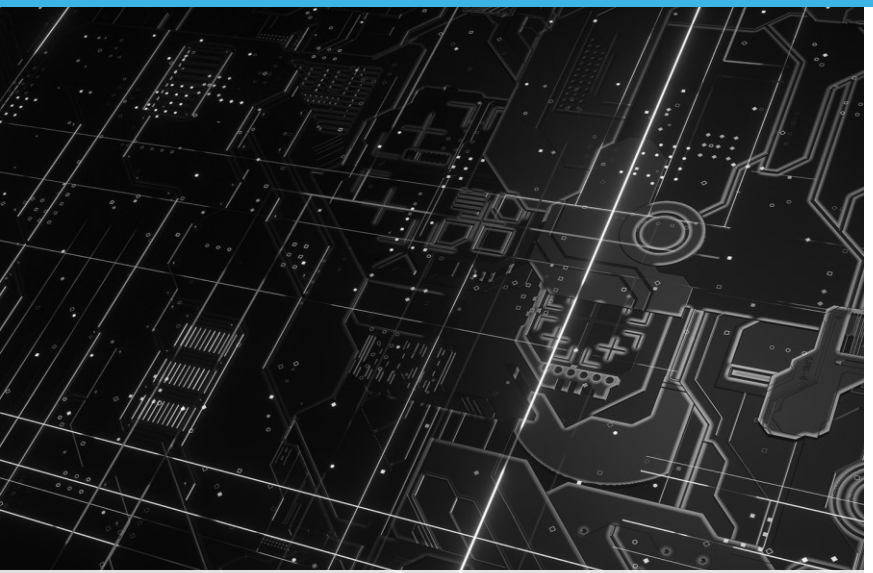
```
tag input
0 random
0 random
0 random
1 fixed
1 fixed
0 random
0 random
1 fixed
...
```



life.augmented

The Side-Channel Analysis Library (SCALib)

SCALib: state-of-the-art tools for side-channel evaluation

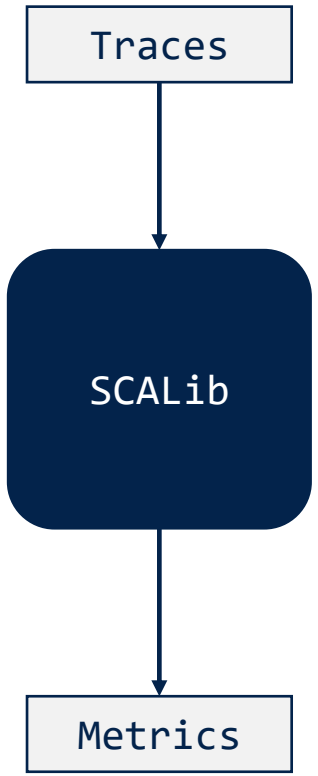


Key applications

- High performance leakage assessment.
- Metrics, modeling and attacks.
- On-the-fly computation with a streaming API.

```
from scalib.metrics import Ttest
import numpy as np
traces = open("simulation.traces")
tag = open("metadata.txt")

ttest = Ttest(1000, d=3)
ttest.fit_u(traces, tag)
t = ttest.get_ttest()

plot(t)
```

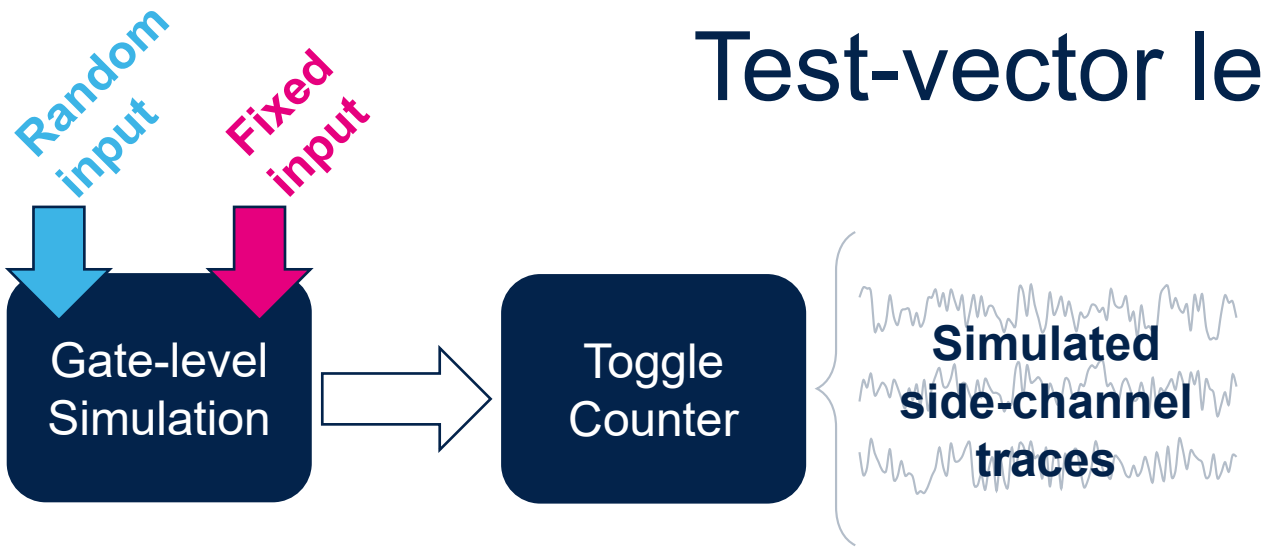
Python Package



G. Cassiers and O. Bronchain. "SCALib: A Side-Channel Analysis Library". In GitHub: SCALib

life.augmented

Test-vector leakage assessment (TVLA[1])



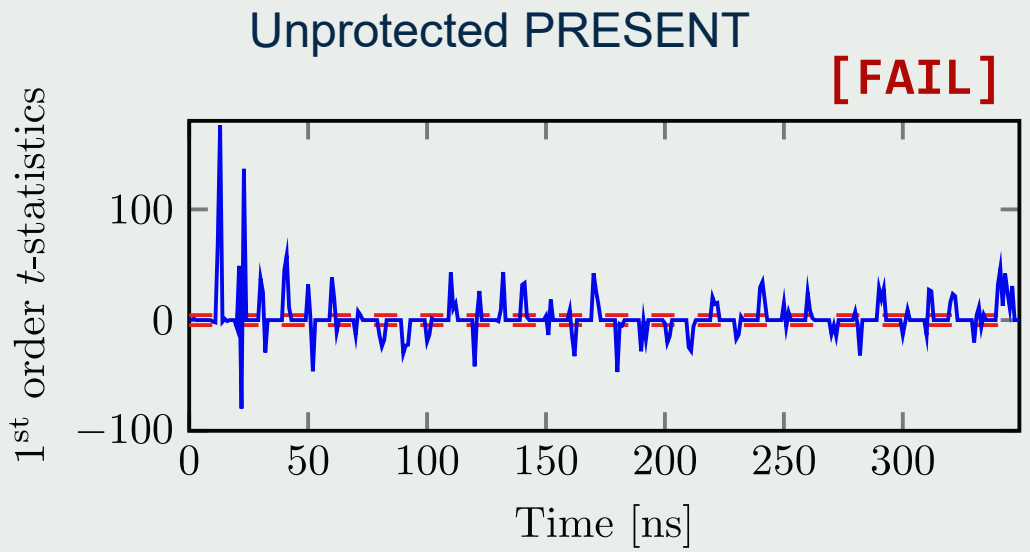
Welch's t-test

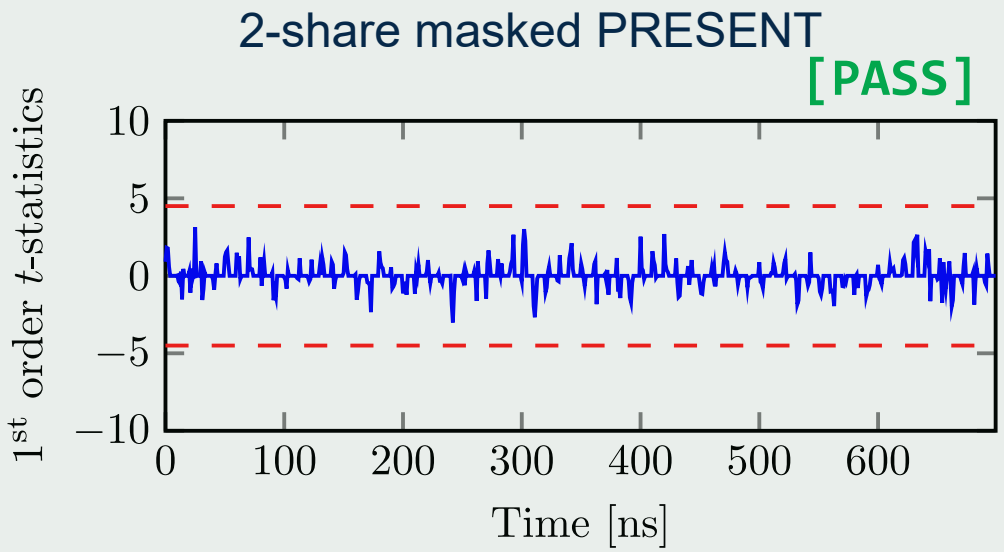
$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{v_0}{n_0} + \frac{v_1}{n_1}}}$$

[PASS] if $|t| < 4.5$

[FAIL] if $|t| \geq 4.5$

The TVLA procedure to identify exploitable side-channel leakages in as masking implementation.





life.augmented

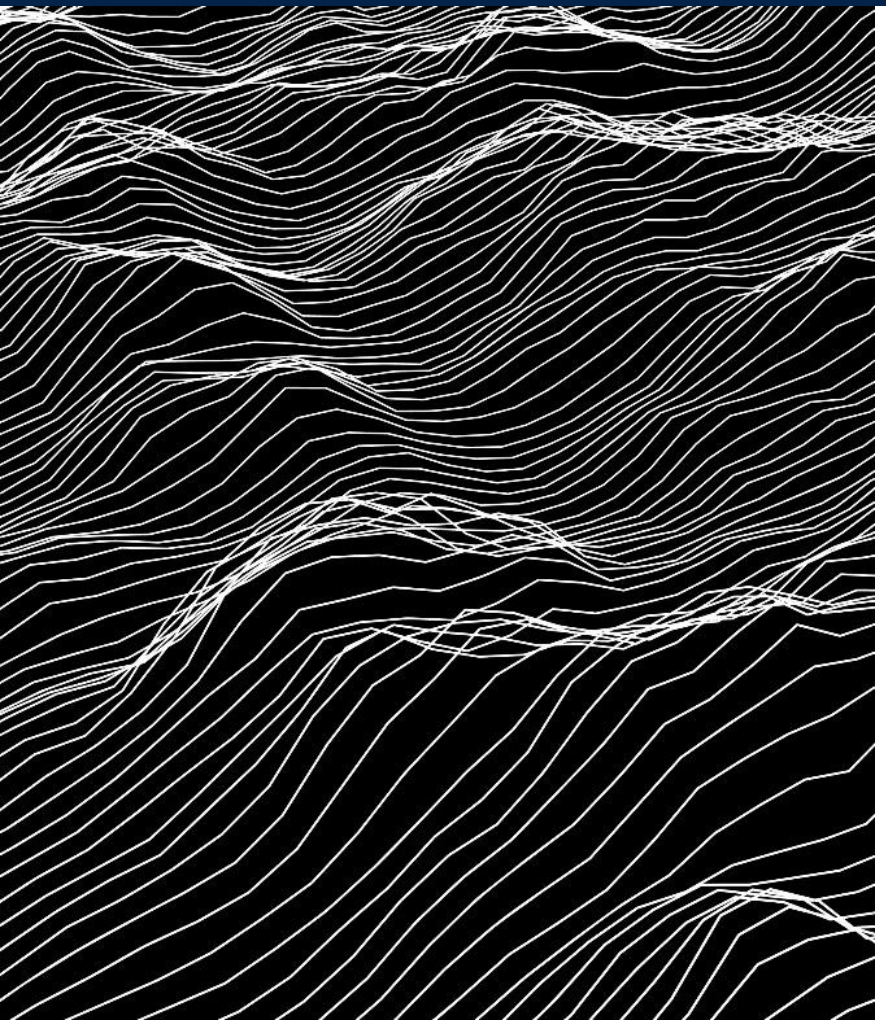
[1] G. Goodwill et al. "A testing methodology for side-channel resistance validation". In NIST Workshop 2011.

Conclusion and perspectives

life.augmented

Conclusion and perspectives

Simulating Traces with Open-Source Tools.



Pros

+ A (close to) open-source design flow.
  + Logic simulation with Icarus Verilog or Verilator.
  + Synthesis with Yosys.
  + Static timing analysis with OpenSTA.
  + Open-source libraries Google Skywater 130 nm, FreePDK 45 nm.

Cons

- The gate-level simulation with open-source tools is currently limited due to some issues with the SDF support.

What next?

- Implement support for SDF files or consider utilizing commercial tools.

Our technology starts with You

Find out more at www.st.com

life.augmented