LABORATOIRE HUBERT CURIEN

UMR • CNRS • 5516 • SAINT-ETIENNE

life.augmented

# Low-Latency Masking with Arbitrary Protection Order Based on Click Elements

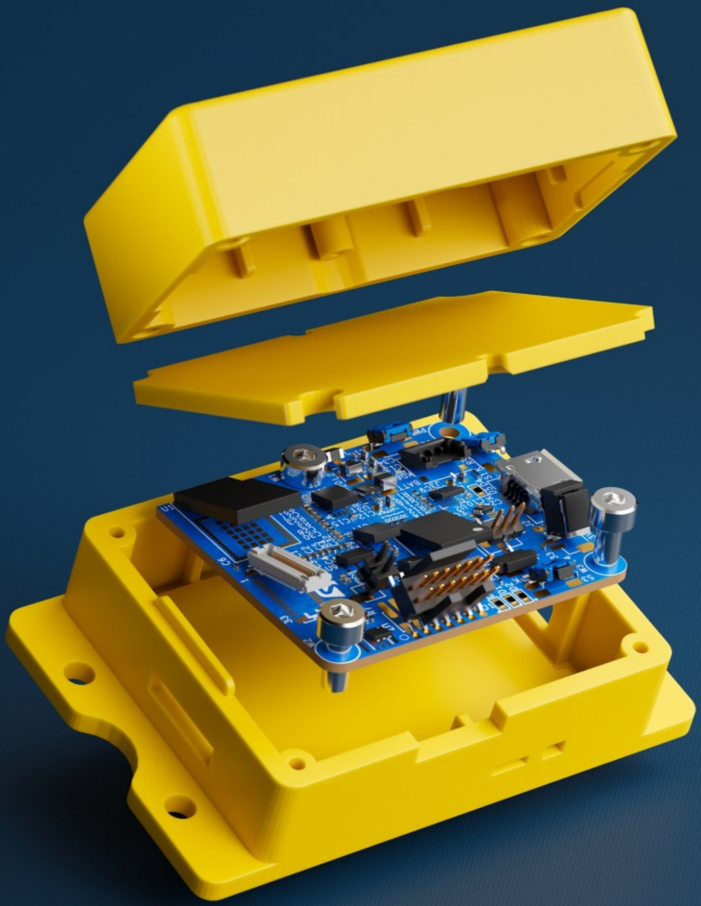
**Mateus Simões**[1, 2], Lilian Bossuet[2], Nicolas Bruneau[1], Vincent Grosso[2], Patrick Haddad[1], Thomas Sarno[1]

[1] STMicroelectronics, 13106 Rousset, France

[2] Laboratoire Hubert Curien, CNRS, 42000 Saint-Étienne, France

March 22, 2024

# Agenda



1. Masking Against Side-Channel Attacks
2. Click-Based Low-Latency Masking
3. Practical Implementation of AES
4. Assessment of Side-Channel Leakages
5. Conclusion and Perspectives

life.augmented

# Masking Against Side-Channel Attacks

life.augmented

# Side-channel attacks [1]

Side-channel attacks (**SCA**) are a type of cyber attack that exploit weaknesses in a system's physical or electromagnetic emissions, power consumption, or other observable characteristics to extract sensitive information.

These attacks can bypass traditional security measures such as encryption and authentication. It can be used to steal sensitive data such as passwords, cryptographic keys, and other confidential information.

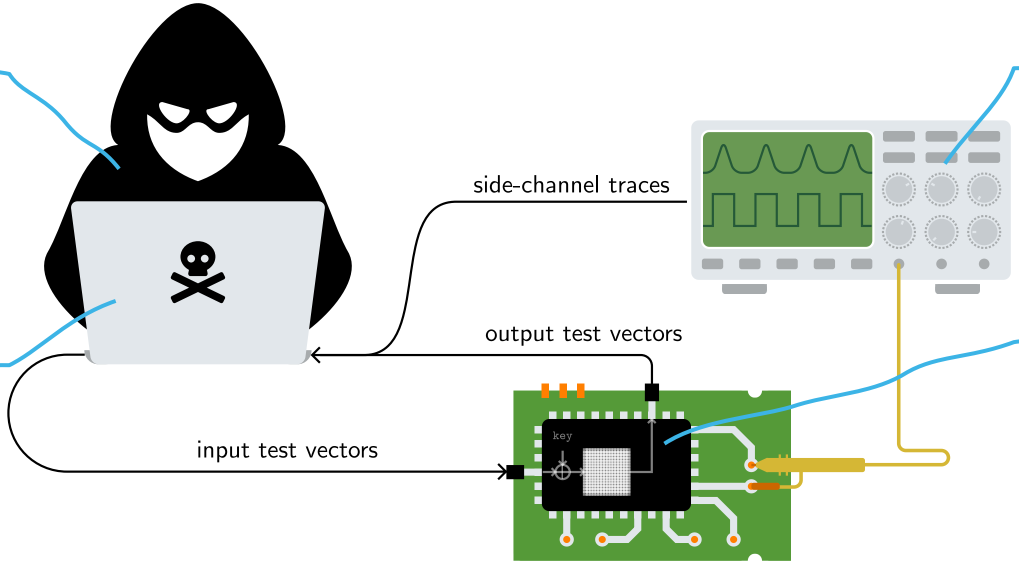
Side-channel attacks can be difficult to detect and prevent and require specialized knowledge and tools to mitigate.

1 The attacker sends messages to the device under attack.

2 The device under attack process the input data.

3 Acquisition of side-channel traces during the processing.

4 The attacker performs statical analysis on the traces.



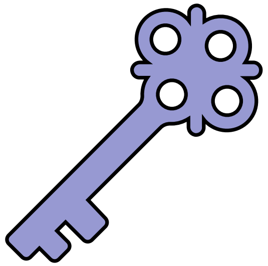

[1] P. Kocher et al. “Differential Power Analysis”. In CRYPTO 1999.

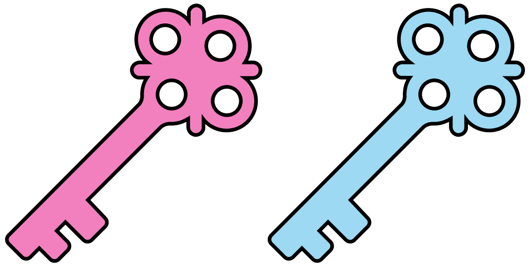
# Hardware masking against side-channel attacks

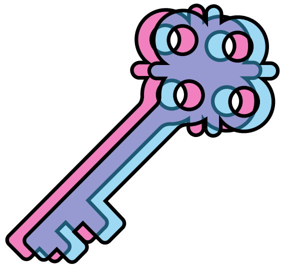
Let us consider secret data, such as the private key.

Masking [1] splits the secret into various random shares.

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







$\texttt{secret} \leftarrow$ **the sensitive data**

$\texttt{share}_1 \leftarrow$ **random mask**

$\texttt{share}_2 \leftarrow \texttt{secret} \oplus \texttt{share}_1$

$\texttt{secret} = \texttt{share}_1 \oplus \texttt{share}_2$

\+ 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 masking design procedure may introduce side-channel exposures.

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

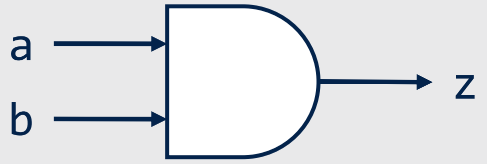
life.augmented

# Gadget-based hardware masking

**A *gadget* is a secure circuit that performs a specific logical function.**

## Operations over variables in $\mathbb{F}_2$

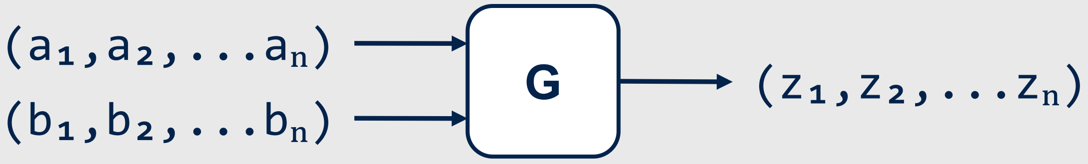
**The logic is composed of atomic gates.**




**The side-channel leakages exhibit a correlation with the secret data.**

## Operations over masked variables in $\mathbb{F}_2^n$

**The logic is composed of $n$-share gadgets.**



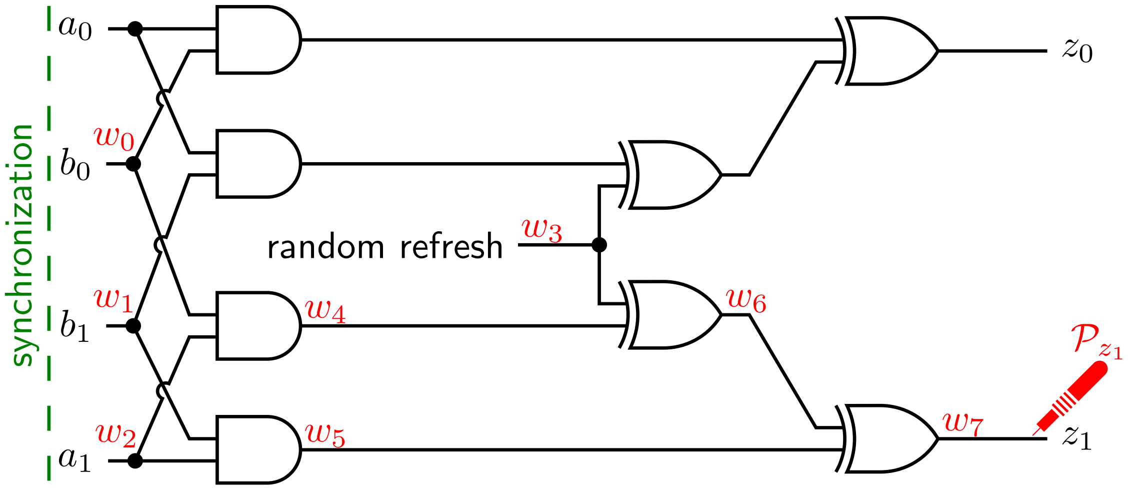

**Theoretically, there is no longer a correlation between the secret data and the side-channel leakages.**

life.augmented

# Notions of security and gadget composability

**The probing [1] $d$-order protection of a masked circuit**

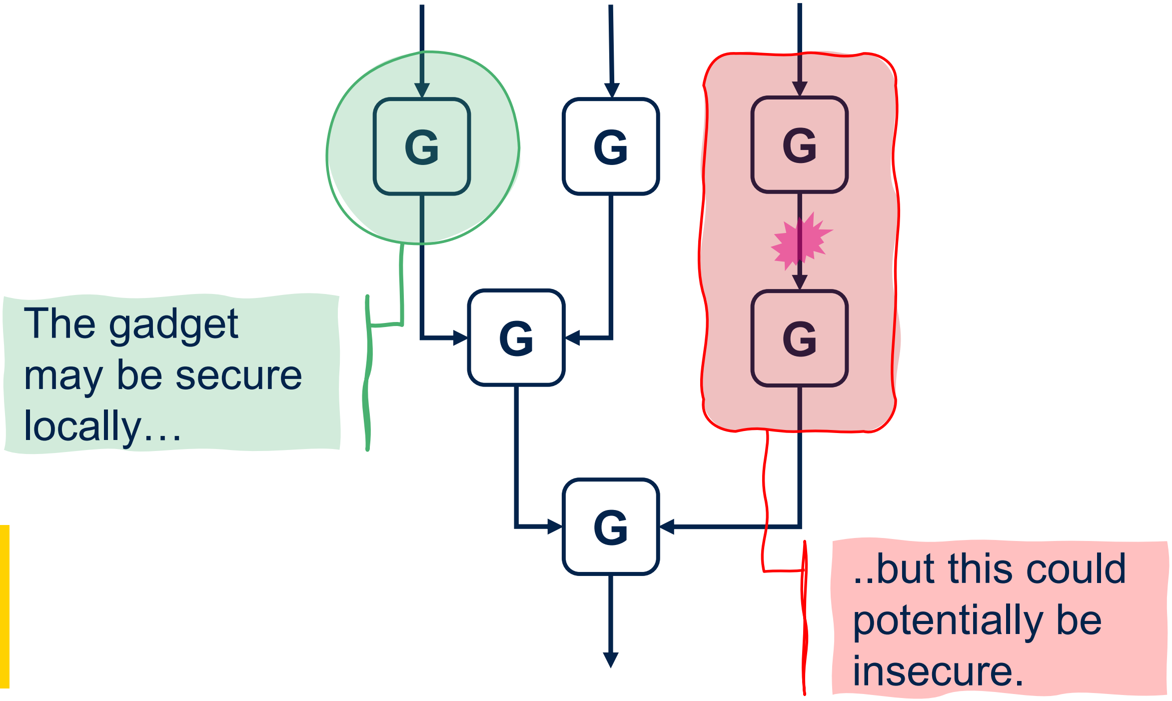
**Any set of at most $d$ probes is independent of the secrets.**




- **Standard probe :** $\mathcal{W}_{z_1} = (w_7)$
- **Robust probe :** $\mathcal{P}_{z_1} = (w_7, w_6, w_5, w_4, w_3, w_2, w_1, w_0)$

**Gadgets composition limitations**

The composition of multiple gadgets that possess $d$-order security may not necessarily lead to the design of a system that is $d$-order secure.




[1] Y. Ishai et al. "Private Circuits: Securing Hardware against Probing Attacks". In CRYPTO 2003.

life.augmented

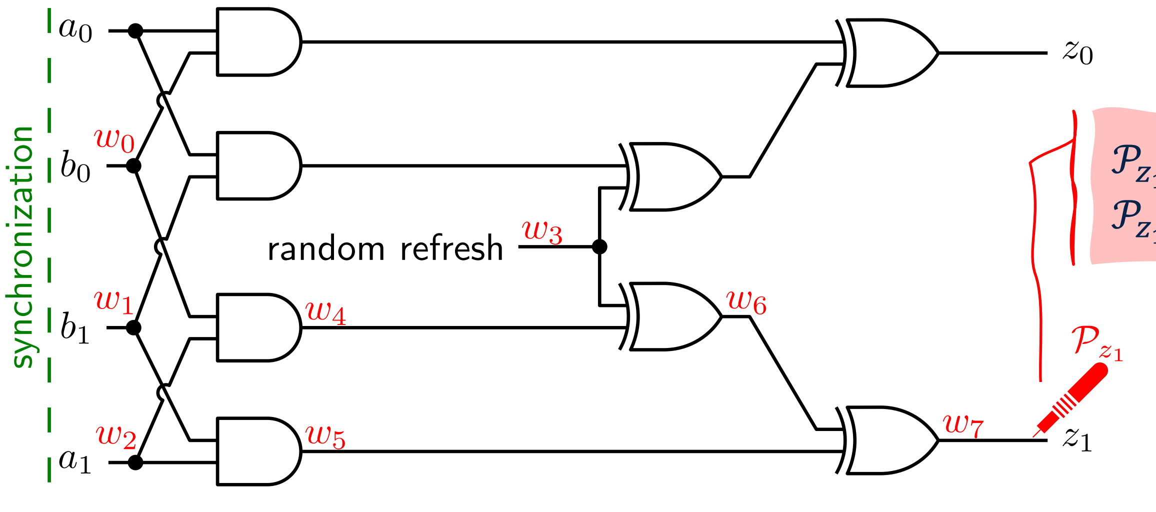
# Masking in the presence of glitches

**Hardware masking in the presence of glitches**

Security flaws caused by glitches, which are the result of unintentional activities arising from signal and switching delays, is a well-known threat [1].

These glitches can compromise the security of the masked circuit by revealing sensitive information. **To avoid glitches, using registers is a common solution.**

Unfortunately, those gadgets introduce a significant amount of additional overhead into the design, in terms of area, latency, and randomness requirements.

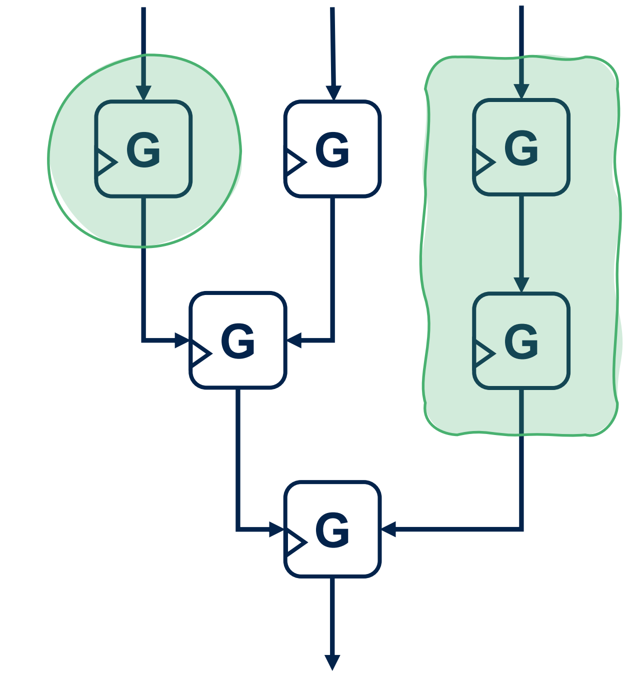



$\mathcal{P}_{z_1}$ depends on the secret $b$

$\mathcal{P}_{z_1} = a_1 b + r$

How to solve it ?

The use of registers is a common solution to achieve $d$-glitch-extended security [2].





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

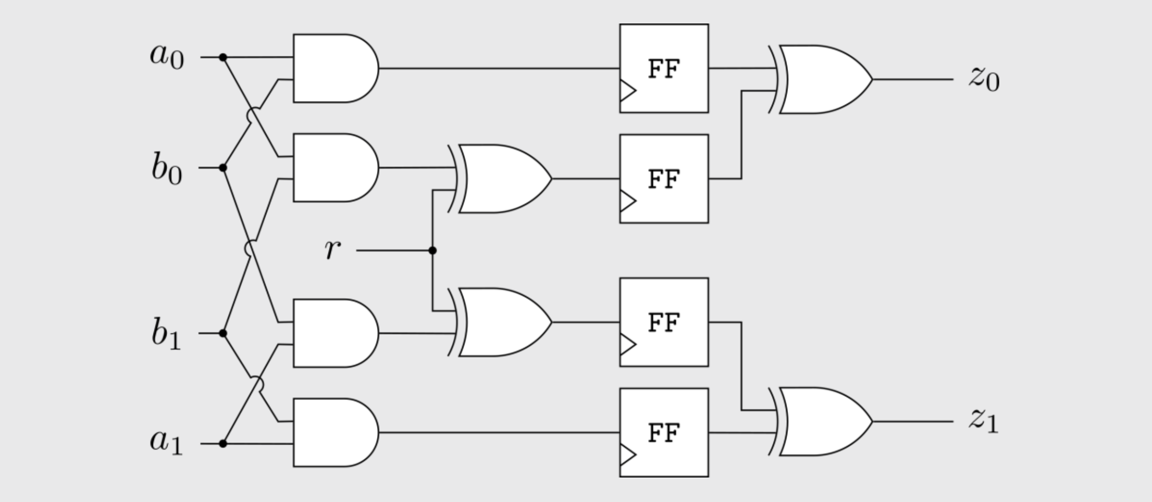
[2] O. Reparaz et al. “Consolidating Masking Schemes”. In CRYPTO 2015.


life.augmented

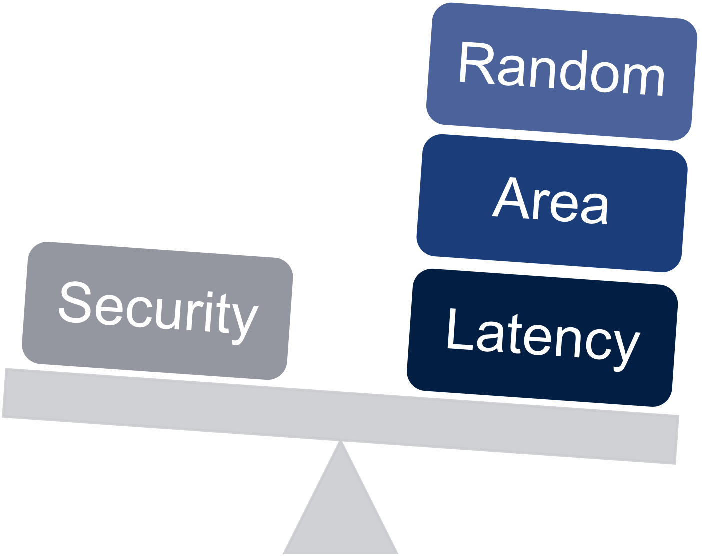
# Why does hardware masking increase latency?

**Benchmark: Domain-Oriented Masking (DOM)**

Masking with 2 shares the bitwise multiplication:




- Physical hazards – e.g., glitches – are source of exploitable side-channel leakages.
- The shares must be statistically independent to compose several masked functions.
- An inaccurate leakage model may result in flawed masking implementations [1].




What are the techniques for achieving secure masking of complex hardware with low latency?

life.augmented

[1] T. Moos et al. "Glitch-Resistant Masking Revisited: or Why Proofs in the Robust Probing Model are Needed". In IACR TCHES 2019.

# Challenges and limitations of low-latency masking

Several low-latency masking solutions exist in the state of the art:

- ❖ Generic Low-Latency Masking (GLM) [1]
- ❖ Low-Latency Threshold Implementations (LLTI) [2]
- ❖ LUT-based Masked Dual-Rail with Pre-charge Logic (LMDPL) [3]
- ❖ Self-Synchronized Masking (SESYM) [4]

However, these solutions are often impractical due to their large silicon area requirements or design flaws caused by physical phenomena such as early propagation and share collisions.

**The state of the art :**

- Requires larger silicon area.
- Relies on dual-rail logic circuits.
- Introduces security vulnerabilities.

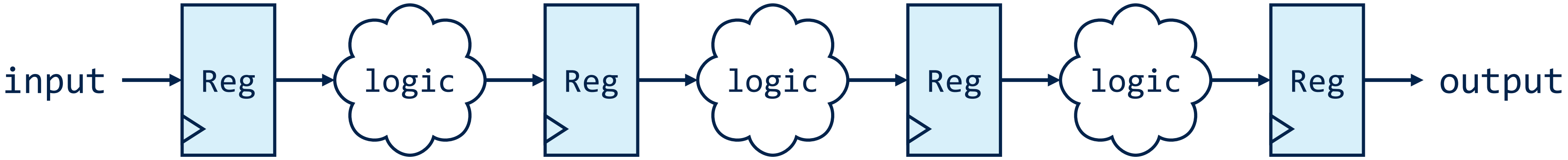
This work proposes a low-latency and area-efficient masking design.

[1] H. Gross et al. “Generic Low-Latency Masking in Hardware”. In TCHES 2018.
[2] V. Arribas et al. “LLTI: Low-Latency Threshold Implementations”. In IEEE TIFS 2022.
[3] P. Sasdrich et al. “Low-Latency Hardware Masking with Application to AES”. In TCHES 2020.
[4] R. Nagpal et al. “Riding the Waves Towards Generic Single-Cycle Masking”. In TCHES 2022.

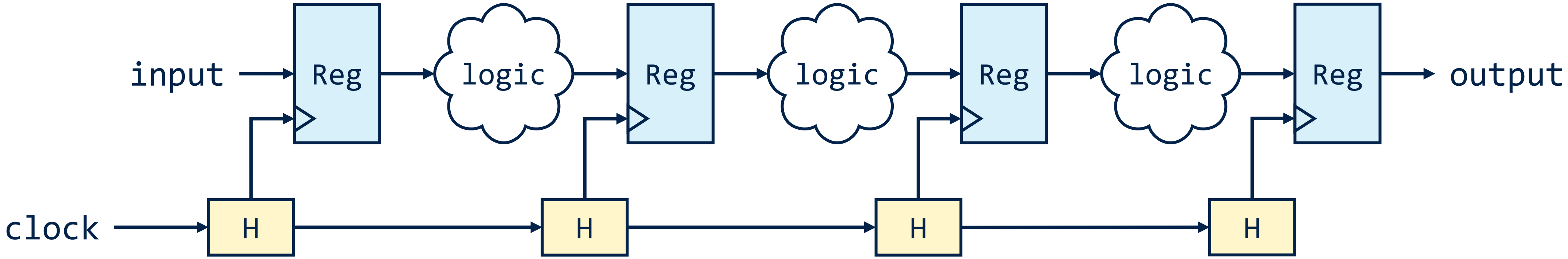
# Click-Based Low-Latency Masking

# Single-cycle masking through logic handshaking

1 Resistance against glitches often necessitates the use of registers, which in turn increases the latency.



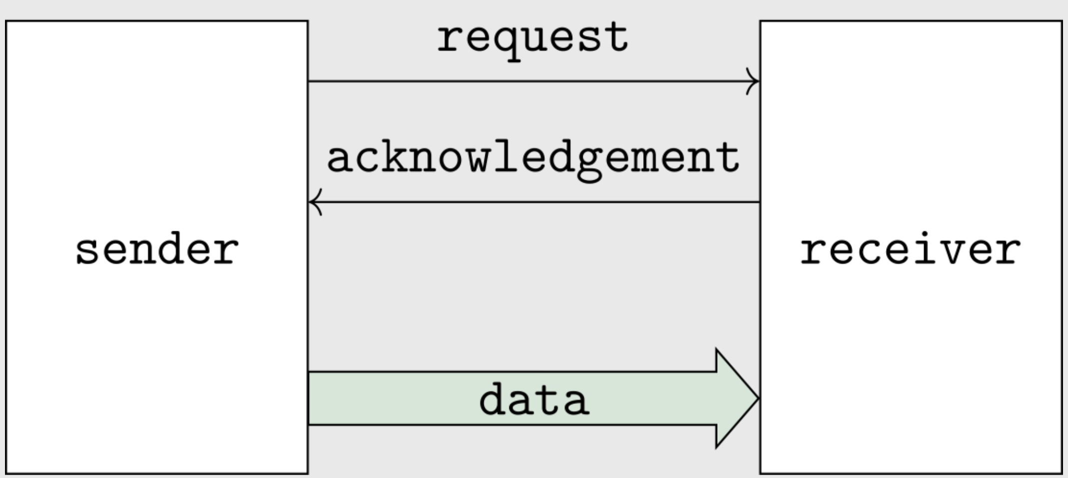

2 To achieve single-cycle masking, synchronization can be accomplished by using handshaking logic.



# Two-phase bundled-data handshake circuits [1]

## The bundled-data circuit

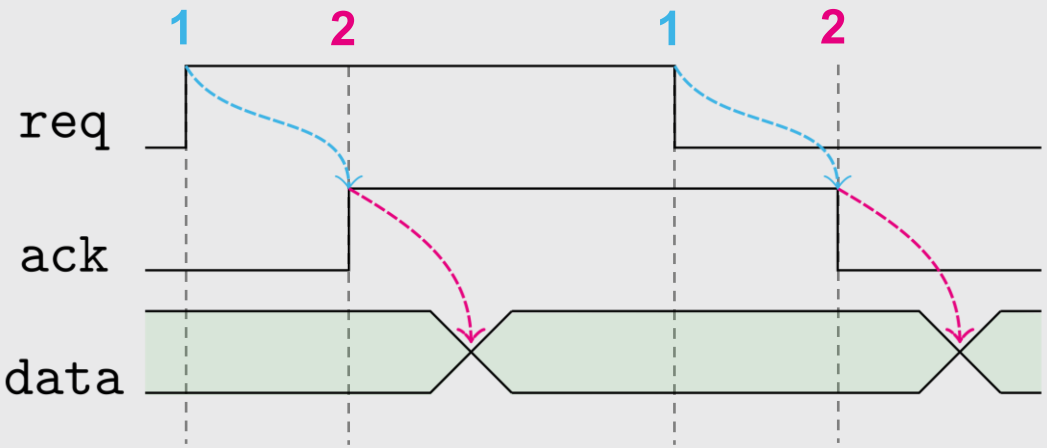
Data are bundled with **handshake** signals.




The **sender** signals data availability with a request bit.
The **receiver** flags evaluation with an acknowledgement.

## The Two-phase protocol

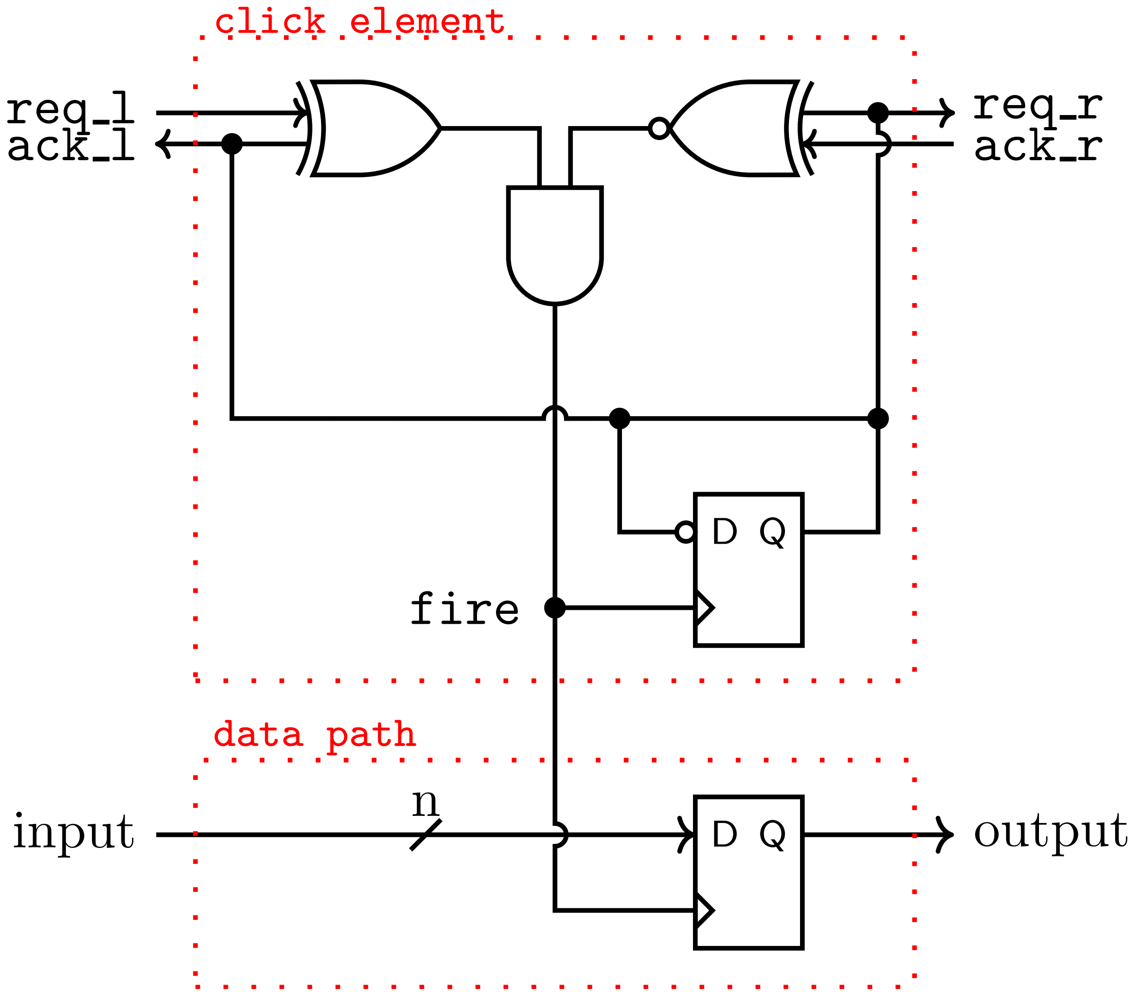
The **handshake** logic is encoded as signal transitions.

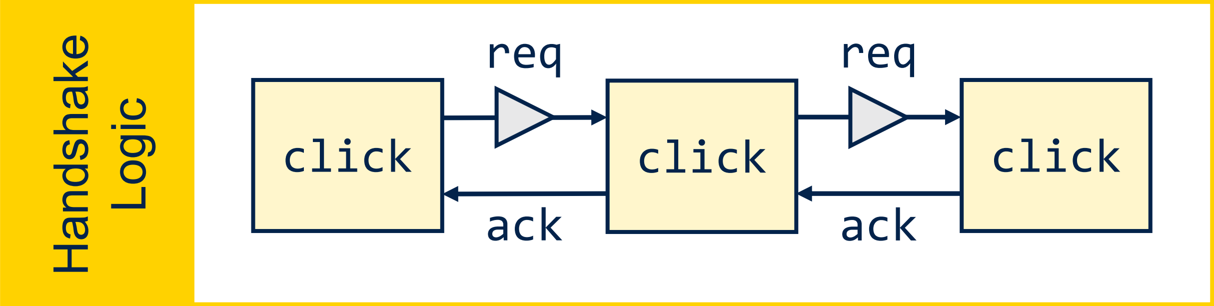



PHASE 1 ↕**req**: The request signal transitions.
PHASE 2 ↕**ack**: The acknowledgement signal transitions.

[1] I. Sutherland. “Micropipelines”. In Commun. ACM 1989.

# Handshake logic with click elements [1]




**The functioning of a click element**

The click element[1] is a handshake controller that relies on edge-triggered flip-flops and regular logic gates.

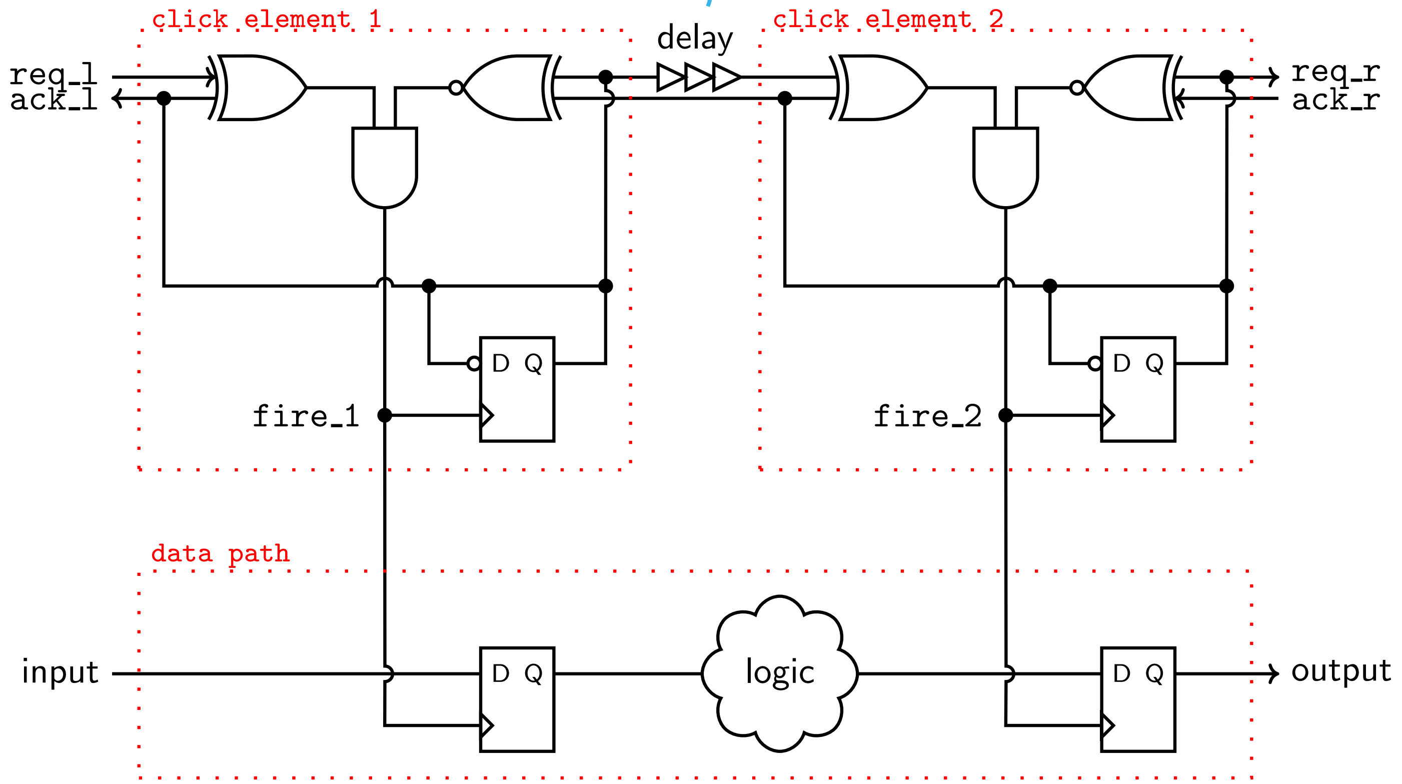
The click element is designed for use in two-phase bundled data circuits, where the signal edge triggers a handshake event.

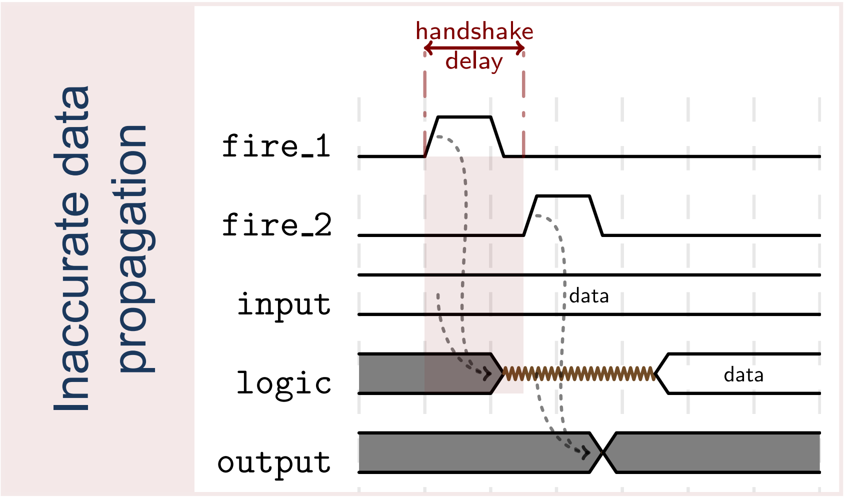
The controller generates a fire pulse when the previous block requests to send new input data (**req_l ≠ ack_l**) and the current data has already been captured by the following block (**req_r = ack_r**).

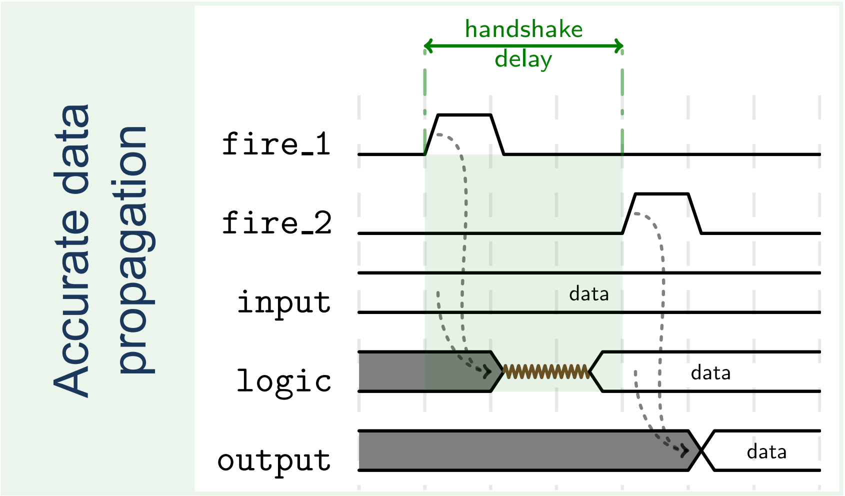
[1] A. Peeters et al. “Click Elements: An Implementation Style for Data-Driven Compilation”. In ASYNC 2010.

# Delay matching requirements

To ensure accurate data propagation, the delay in the handshake channel must match the data path delay.

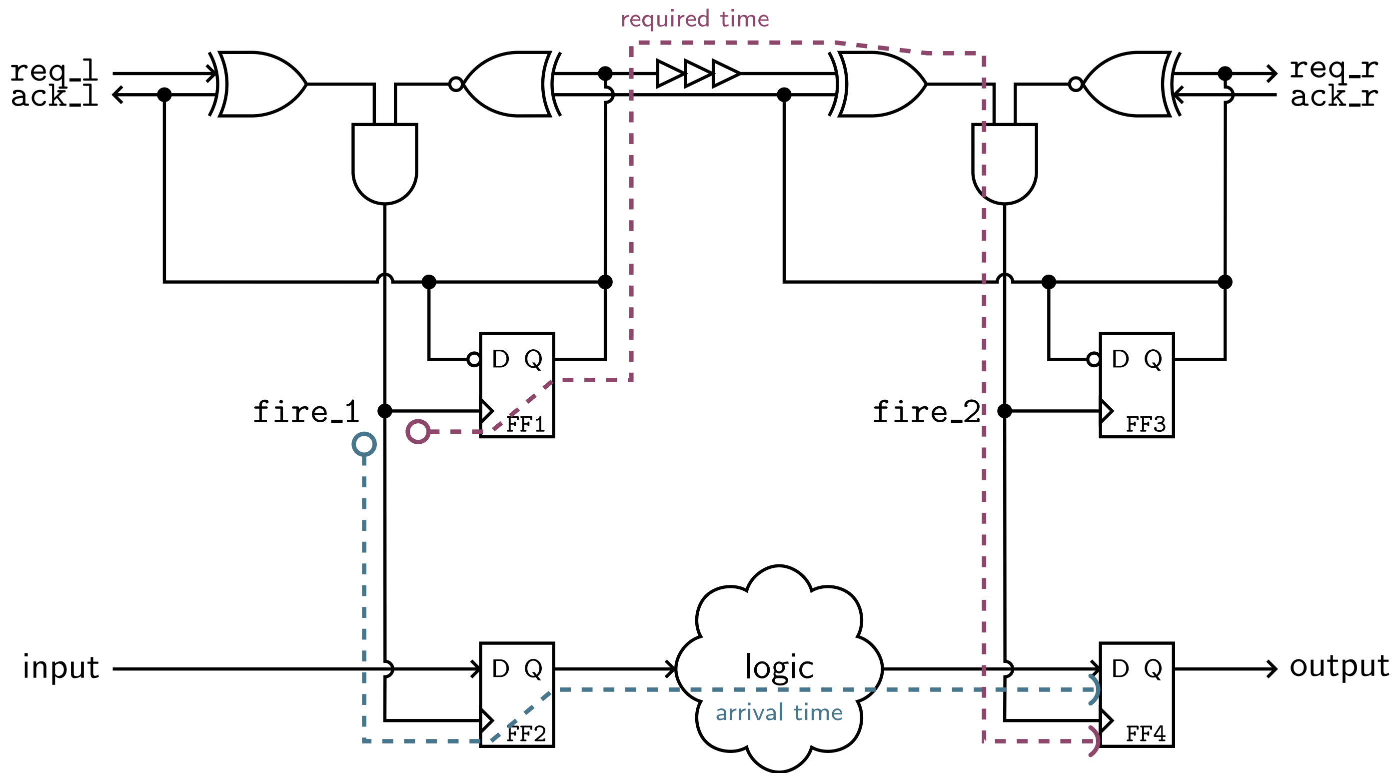


# Static timing analysis of asynchronous circuits

**Timing analysis is needed to satisfy the delay between click elements**




```
required time - arrival time > 0
```

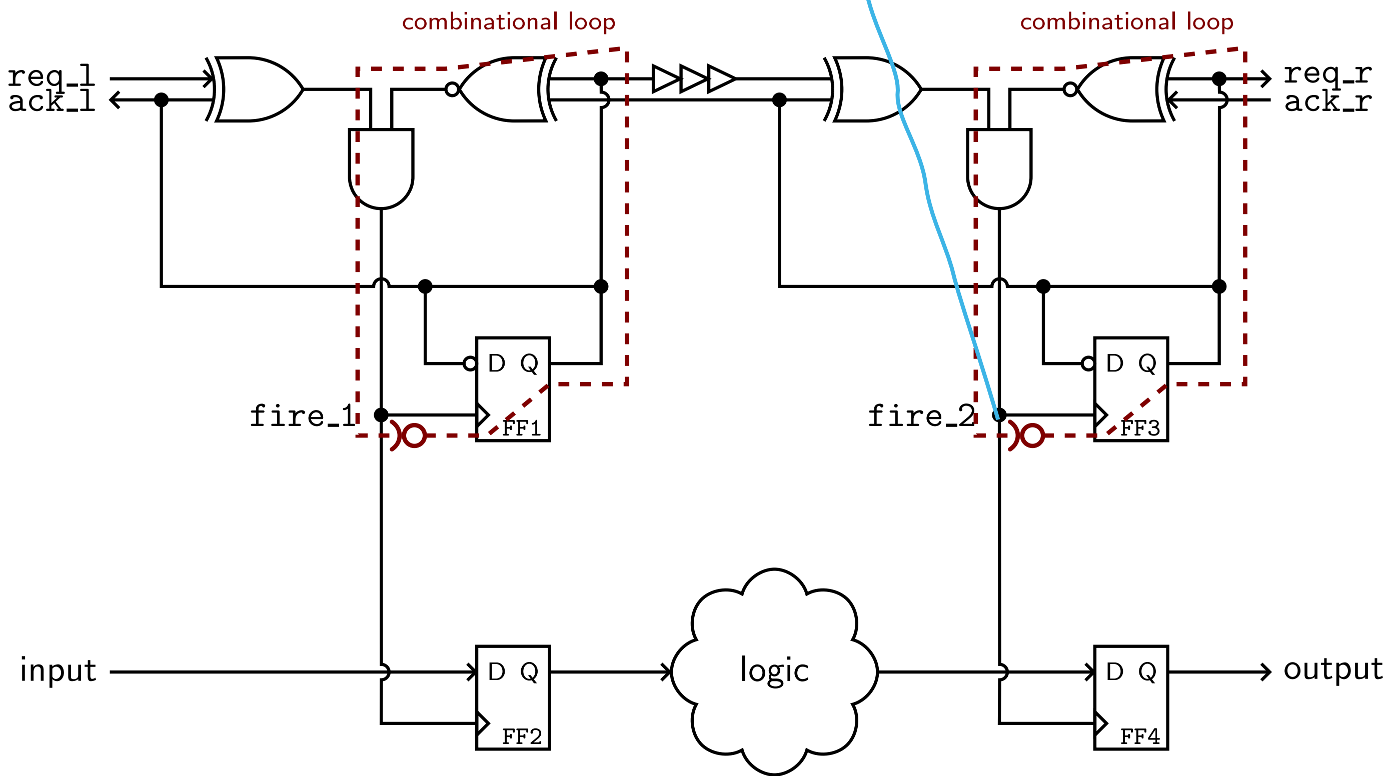
A positive slack indicates that the timing path meets the required timing constraints. Conversely, a negative slack indicates that the timing path violates the setup and hold timing constraints.

To meet the timing requirements, a chain of buffers (▷▷▷) is utilized on the request channel.

# Configuring clock objects

One can use a dummy clock object to break this combinational loop.

To enable the EDA tool to view asynchronous circuits as a composition of locally synchronous circuits and perform static timing analysis step-by-step on each one of them, one can utilize a set of local clocks.



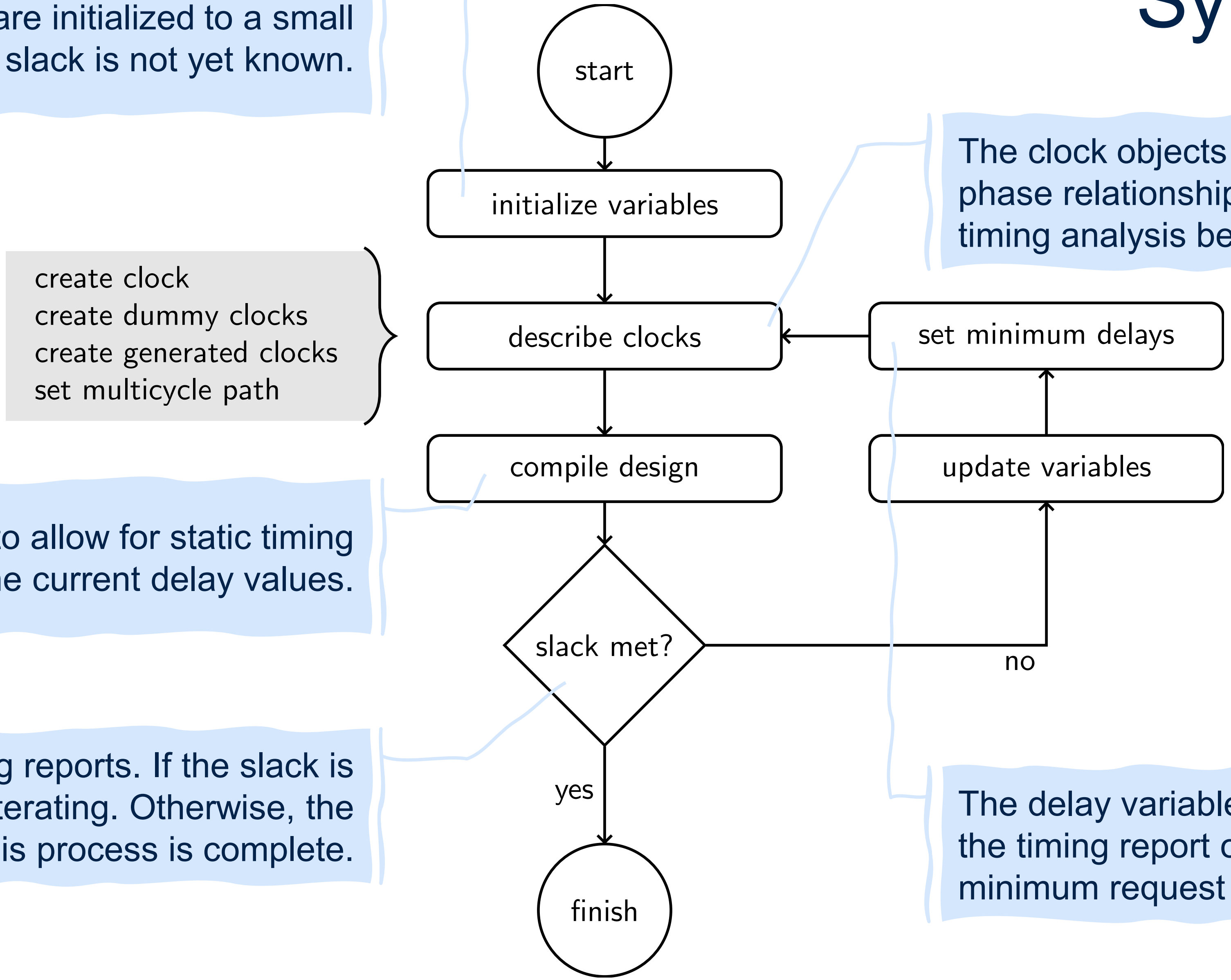

```
# Create clock objects.
create_generate_clock -name -fire_1 ...
create_generate_clock -name -fire_2 ...

# Compile the design to report timing.
compile

# Get slack time requirements.
report_timing -from fire_1 -to fire_2

# Set minimum delay.
set_min_delay -to click_2/req_l $delay

# Re-compile the design to meet timing.
compile
```

life.augmented

# Synthesis flow

The delay variables are initialized to a small value of 0.1 as the slack is not yet known.

The clock objects are set, as well as their phase relationship, to allow for the static timing analysis between click elements.




Compile the design to allow for static timing analysis with the current delay values.

Check the timing reports. If the slack is violated, continue iterating. Otherwise, the synthesis process is complete.

The delay variables are updated based on the timing report outcomes, and then the minimum request channel delays are set.

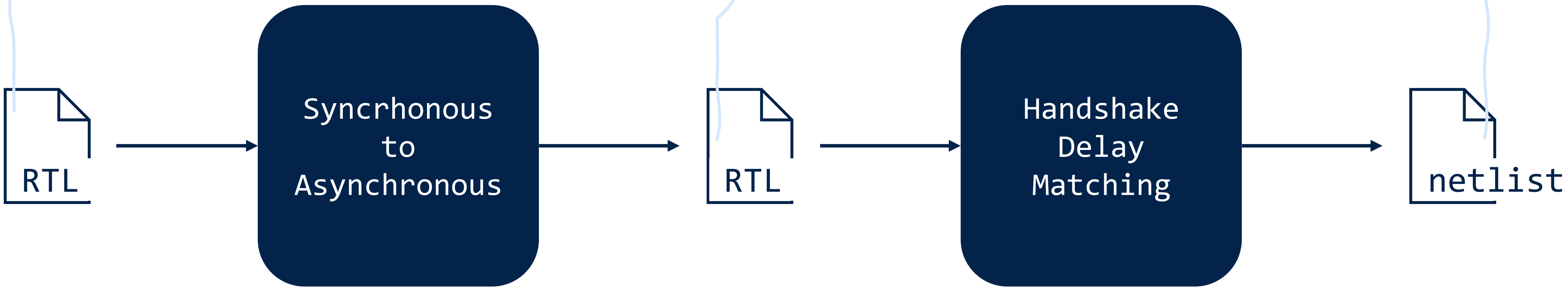
# Asynchronous design using conventional tools

The asynchronous circuit synthesis and verification processes are done in a conventional design flow.

This is a hardware description of a $k$-cycle masked circuit.

Addition of $k$ click elements.

This is a post-synthesis netlist of a 1-cycle masked circuit.

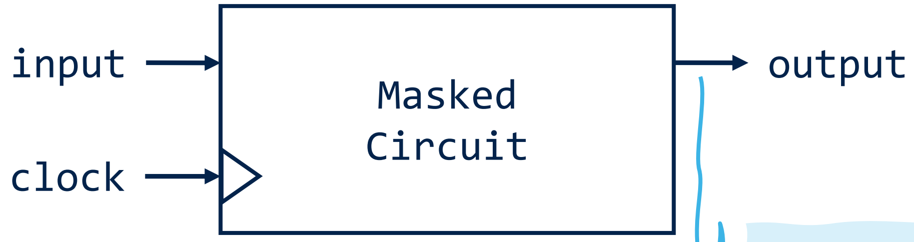



- ❑ Add click elements.
- ❑ Declare fire pulses as clocks.
- ❑ Create buffer lines.

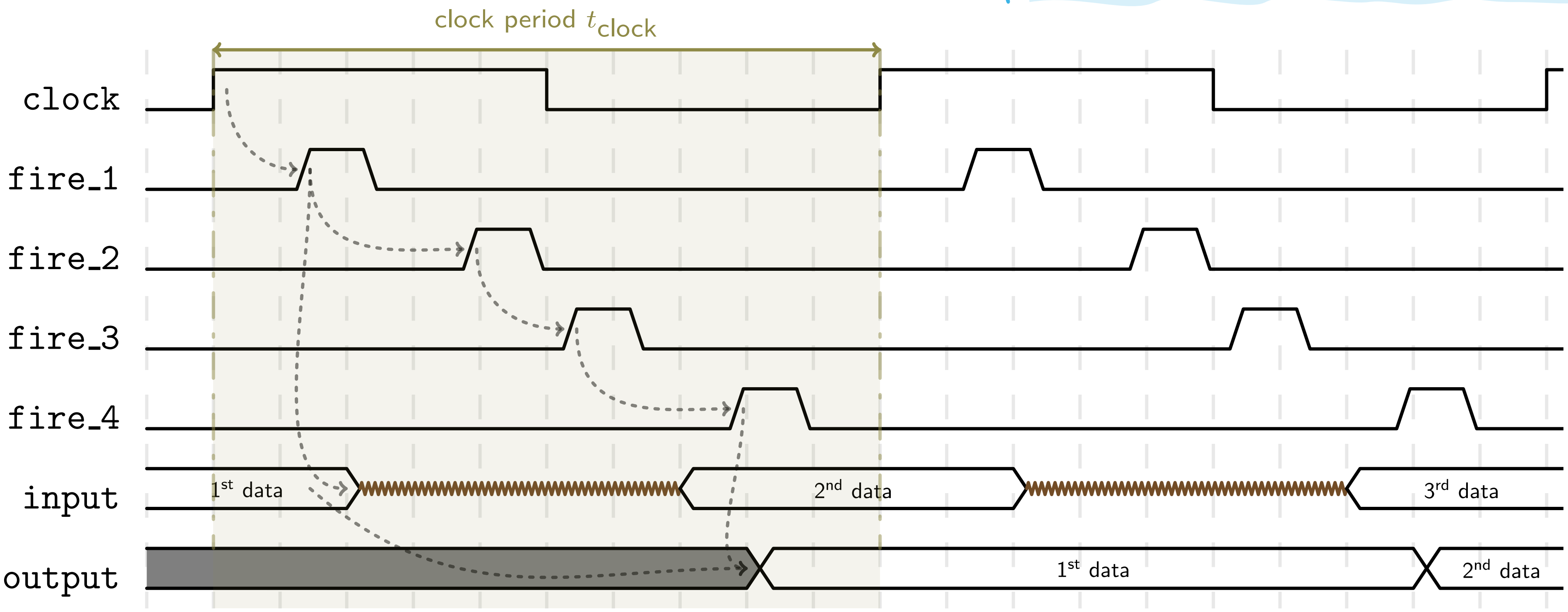
- ❑ Set minimum delay.
- ❑ Run synthesis.
- ❑ Check timing constraints.

life.augmented

# A generic single-cycle design style for masking




The output is available in a clock cycle, regardless of the number of register barriers.




life.augmented

# Practical Implementation of AES

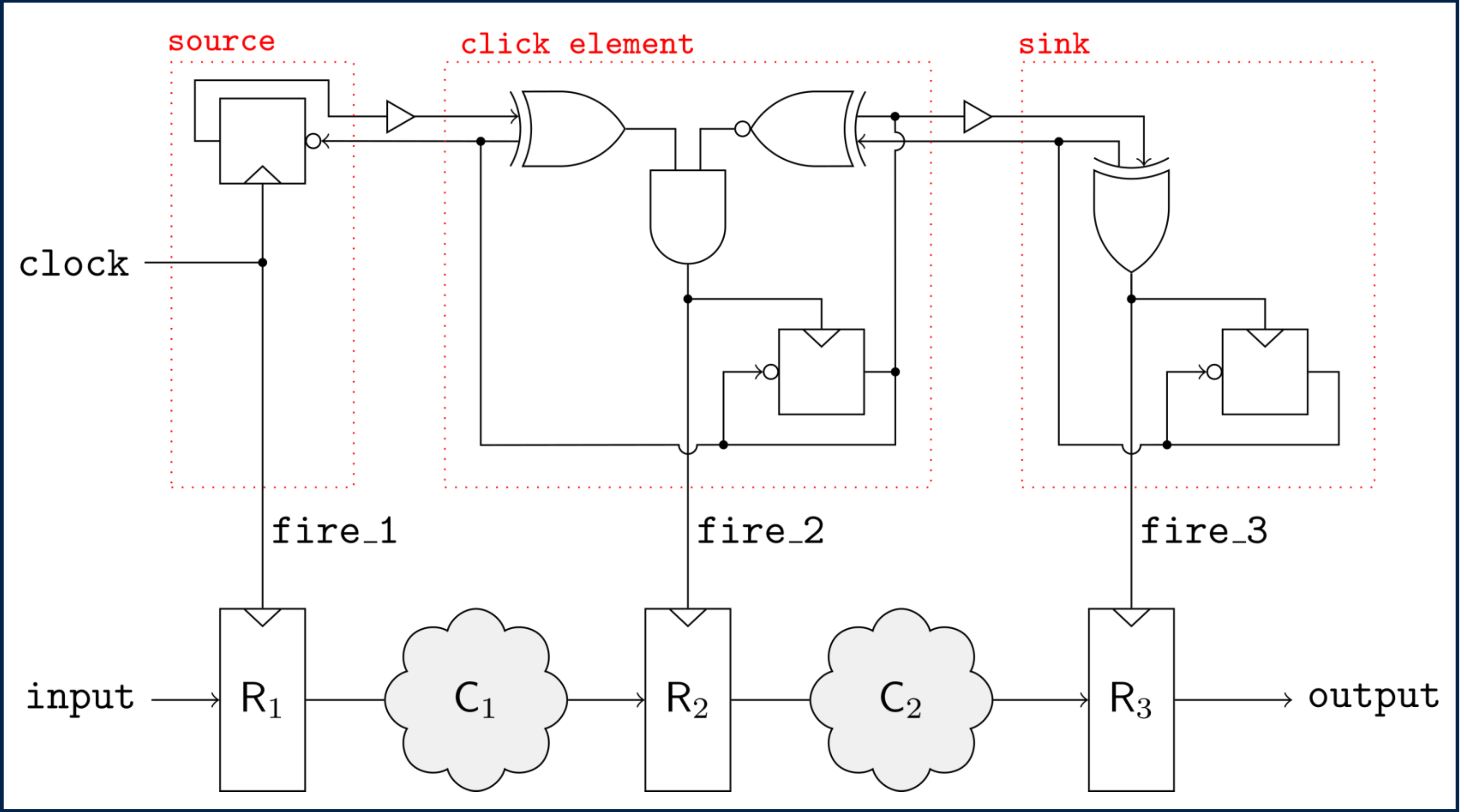
life.augmented

# Locally asynchronous and globally synchronous circuit

The **source** block issues the first request signal.

The **sink** block consumes the last request signal.

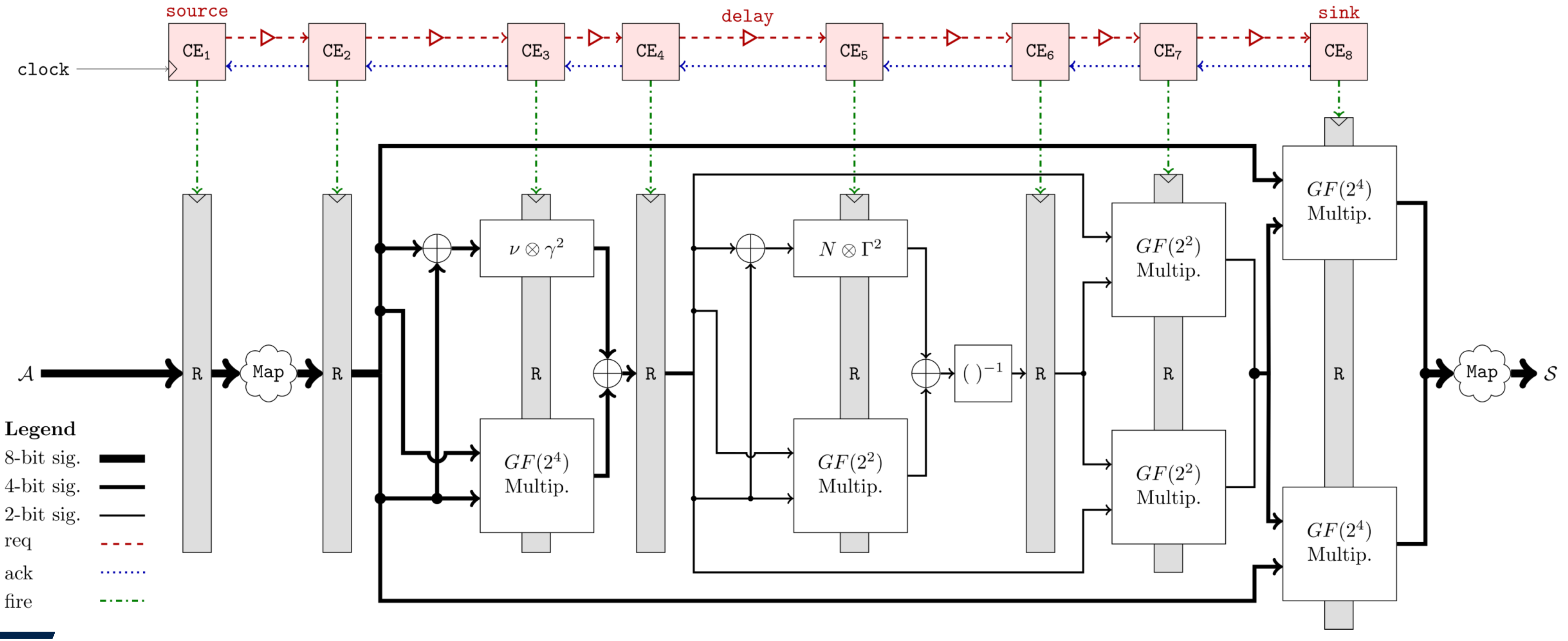
The **clock** triggers an asynchronous pipeline.




life.augmented

# Low-latency masking implementation of the AES S-box

Based on the Canright's [1] design of the AES S-box.




[1] D. Canright. "A Very Compact S-Box for AES". In CHES 2005.

# Low-latency masking implementation of the AES S-box

| Design | Security [order] | Latency [cycles] | Area [kGE] | Random Refresh [bits / cycle] | Frequency [MHz] |
|---|---|---|---|---|---|
| **First-order masking** | | | | | |
| GLM | $d = 1$ | 1 | 60.73 | 2048 | 356 |
| LLTI | 1 | 1 | 25.78 | 0 | 277 |
| SESYM | $d = 1$ | 1 | 7.59 | 18 | 192 |
| LMDPL | 1 | 1 | 3.48 | 36 | 400 |
| **This work[1]** | $\boldsymbol{d = 1}$ | **1** | **1.64** | **36** | **50** |

Our low latency solution is area-efficient and shows lower toggle activity compared to its synchronous version.

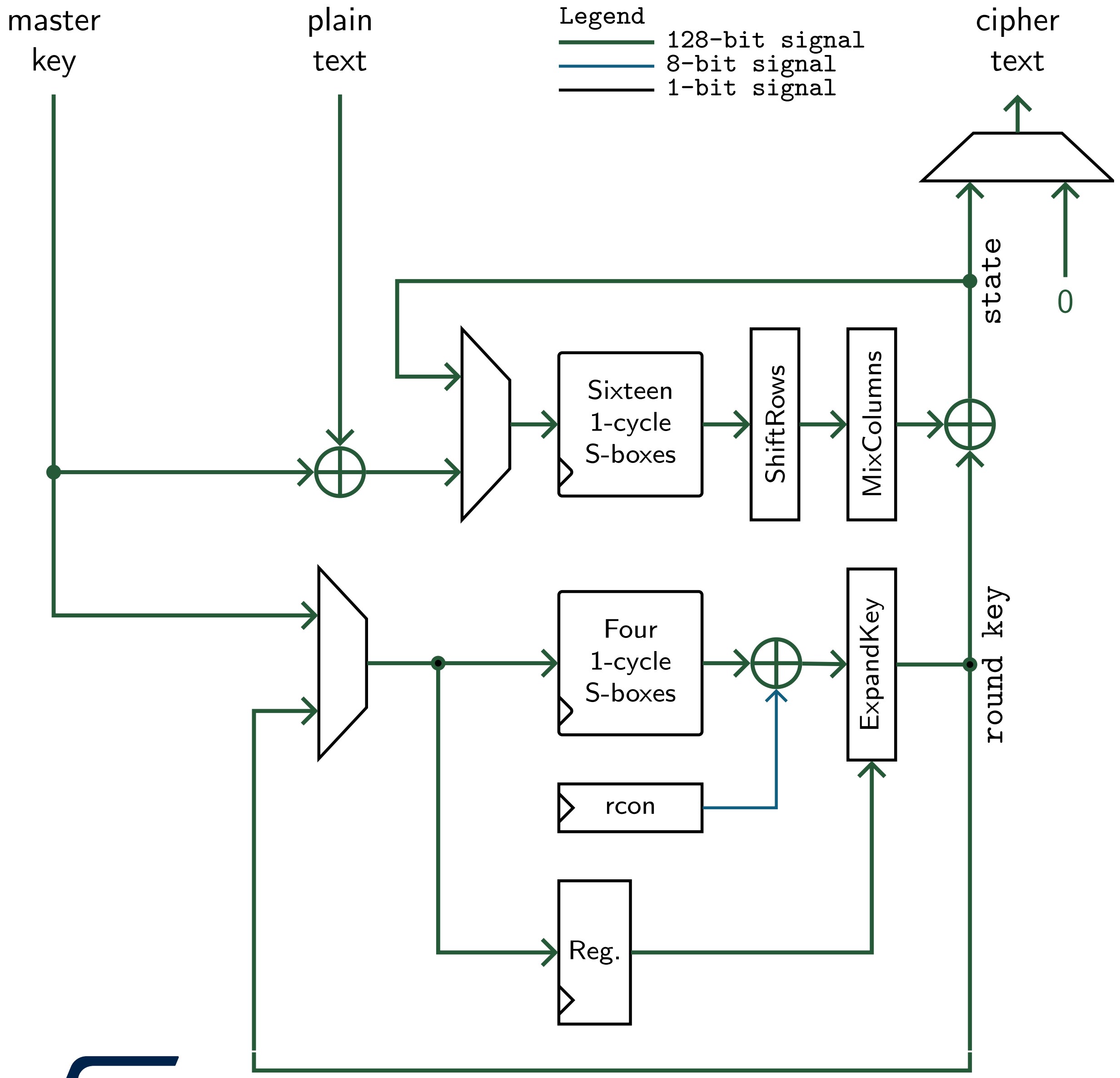
Our AES S-box is the only secure against arbitrary-order protection.

Lower throughput is due to a larger data path and handshake delay margin.

life.augmented

[1] ***Synthesized with "worst-case" PVT for CMOSM40 @ 100MHz.***

# AES architecture[1]




| Design | AES Area [kGE] | S-box Area [kGE] | Refresh [bits/cycle] | Latency [cycles] |
|---|---|---|---|---|
| First-order click-based masking | | | | |
| 8-bit | 6.42 | 1.64 | 36 | 216 |
| 32-bit | 12.73 | 6.56 | 144 | 54 |
| 128-bit | 43.45 | 32.80 | 720 | 11 |
| Second-order click-based masking | | | | |
| 8-bit | 10.64 | 3.43 | 108 | 216 |
| 32-bit | 22.77 | 13.72 | 432 | 54 |
| 128-bit | 83.82 | 68.60 | 2160 | 11 |

[1] ***Synthesized for CMOSM40 @ 100MHz.***

# Assessment of Side-Channel Leakages

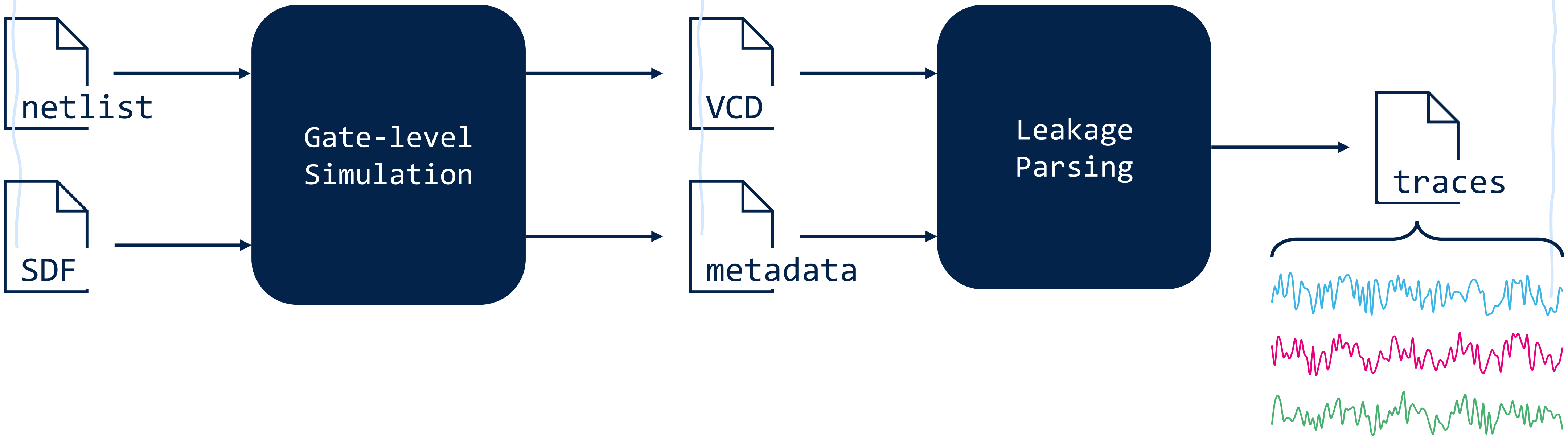
# Overview of the trace generation process

The side-channel traces are generated from toggle counting after post-synthesis gate-level simulation.

The post-synthesis netlist and the gate-level delay information.

The input messages and side-channel data.

For each message, we obtain the side-channel trace from toggle counting.



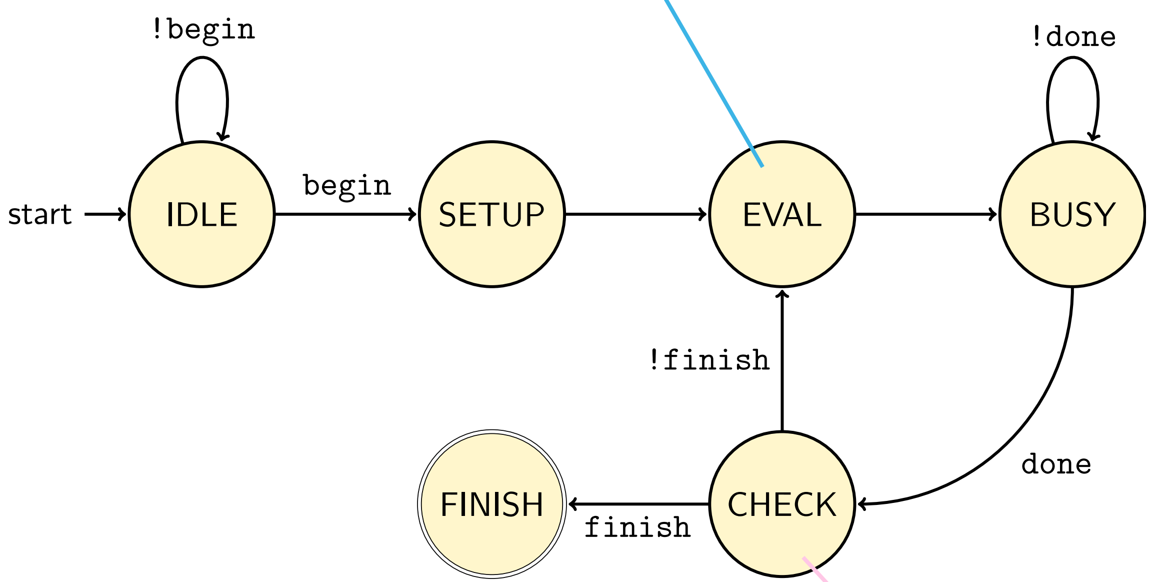

life.augmented

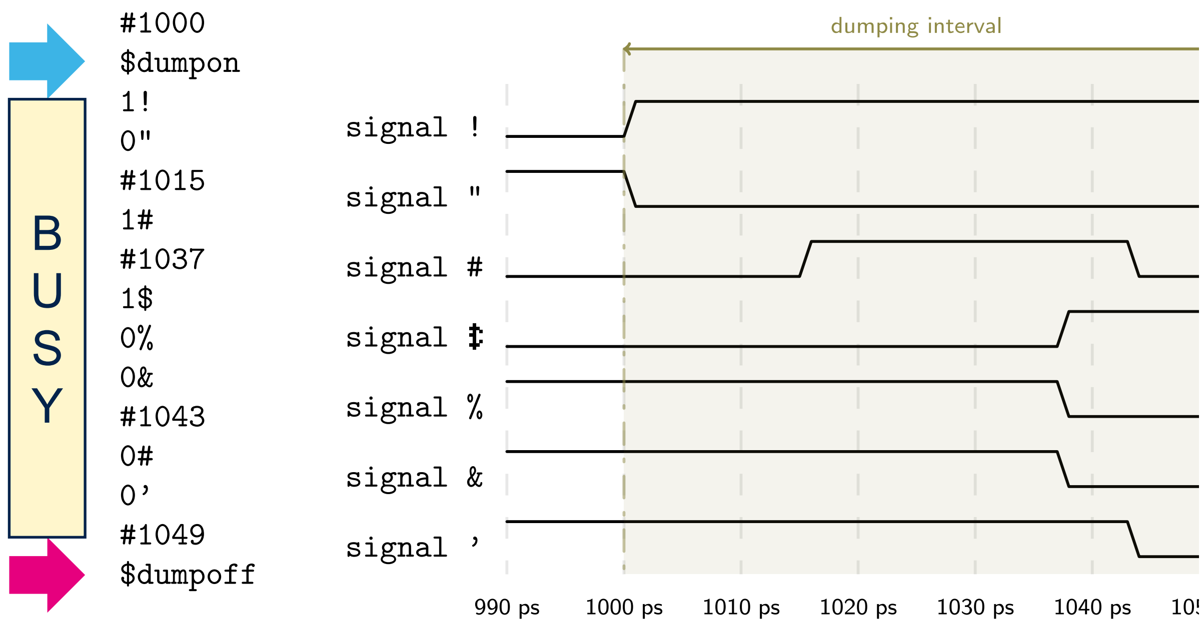
# Extracting side-channel information

**Run simulation**

**Value change dump**

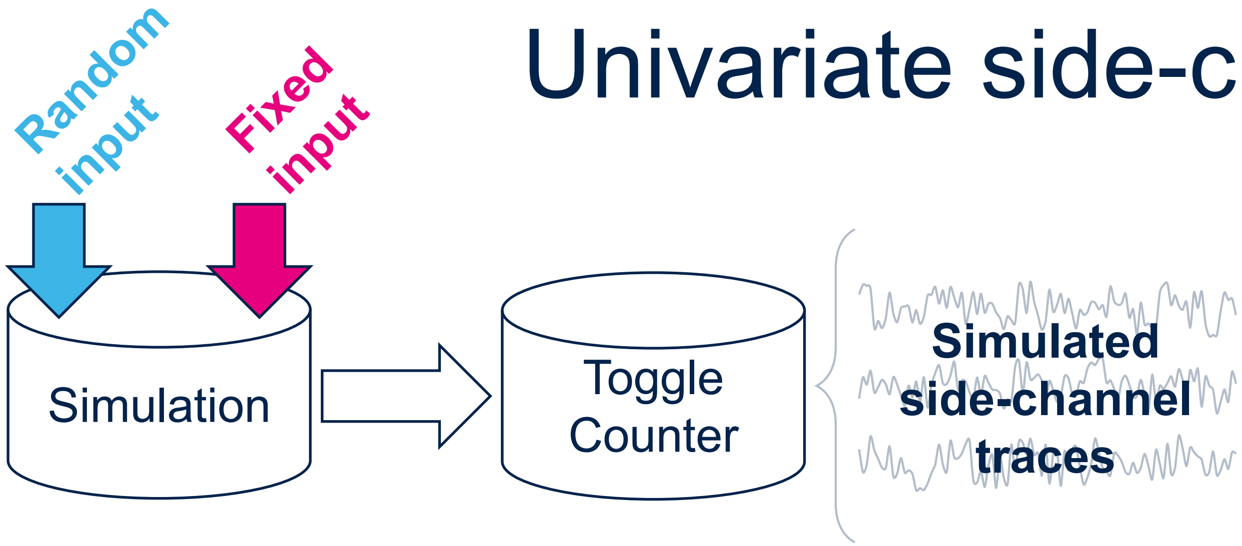
Start value change dump.




Stop value change dump.

# Univariate side-channel leakage assessment

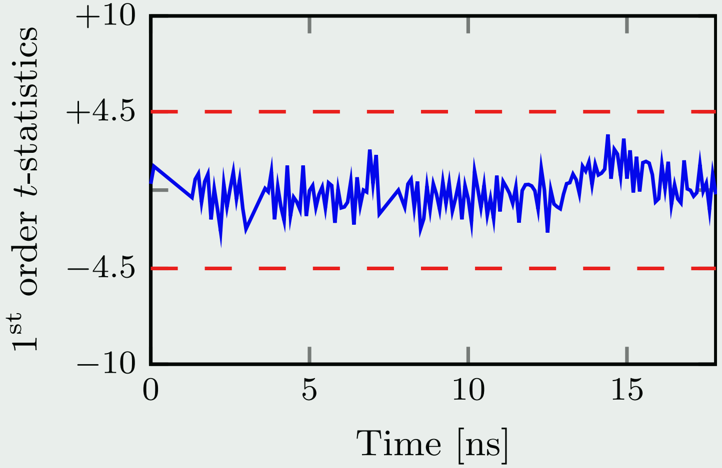



**Test Vector Leakage Assessment (TVLA)**

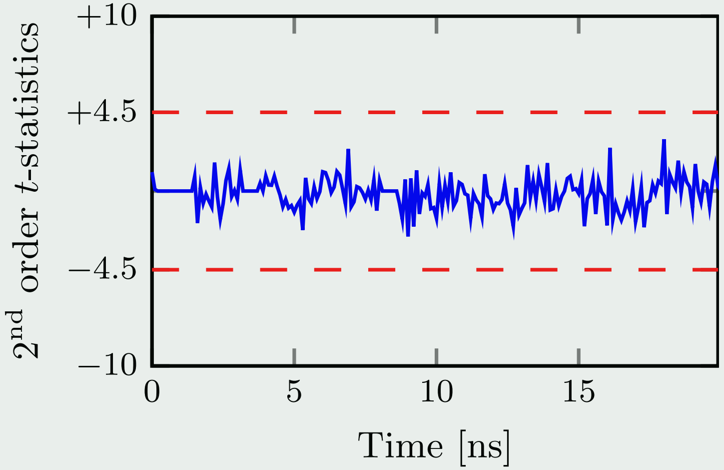
$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{v_0}{n_0} + \frac{v_1}{n_1}}} \quad \begin{cases} \texttt{[PASS]} \text{ if } |t| < 4.5 \\ \texttt{[FAIL]} \text{ if } |t| \geq 4.5 \end{cases}$$

**Our low-latency solution does not weaken a secure masking scheme.[1]**

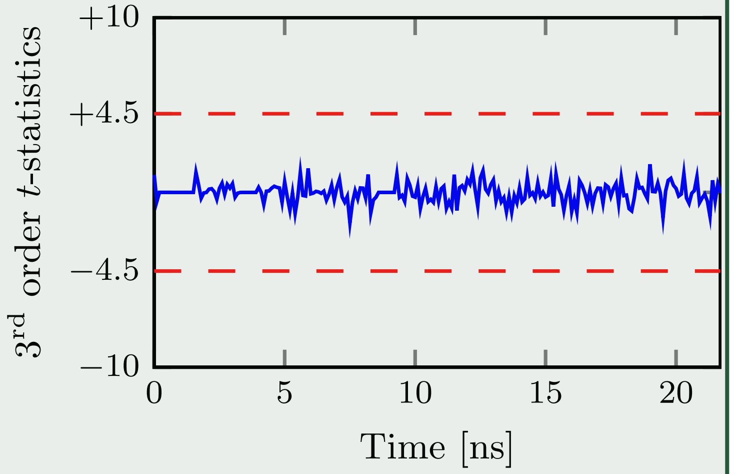
First-order masking **[PASS]**




Second-order masking **[PASS]**




Third-order masking **[PASS]**




[1] *The results are based on 10E6 simulated traces.*

life.augmented

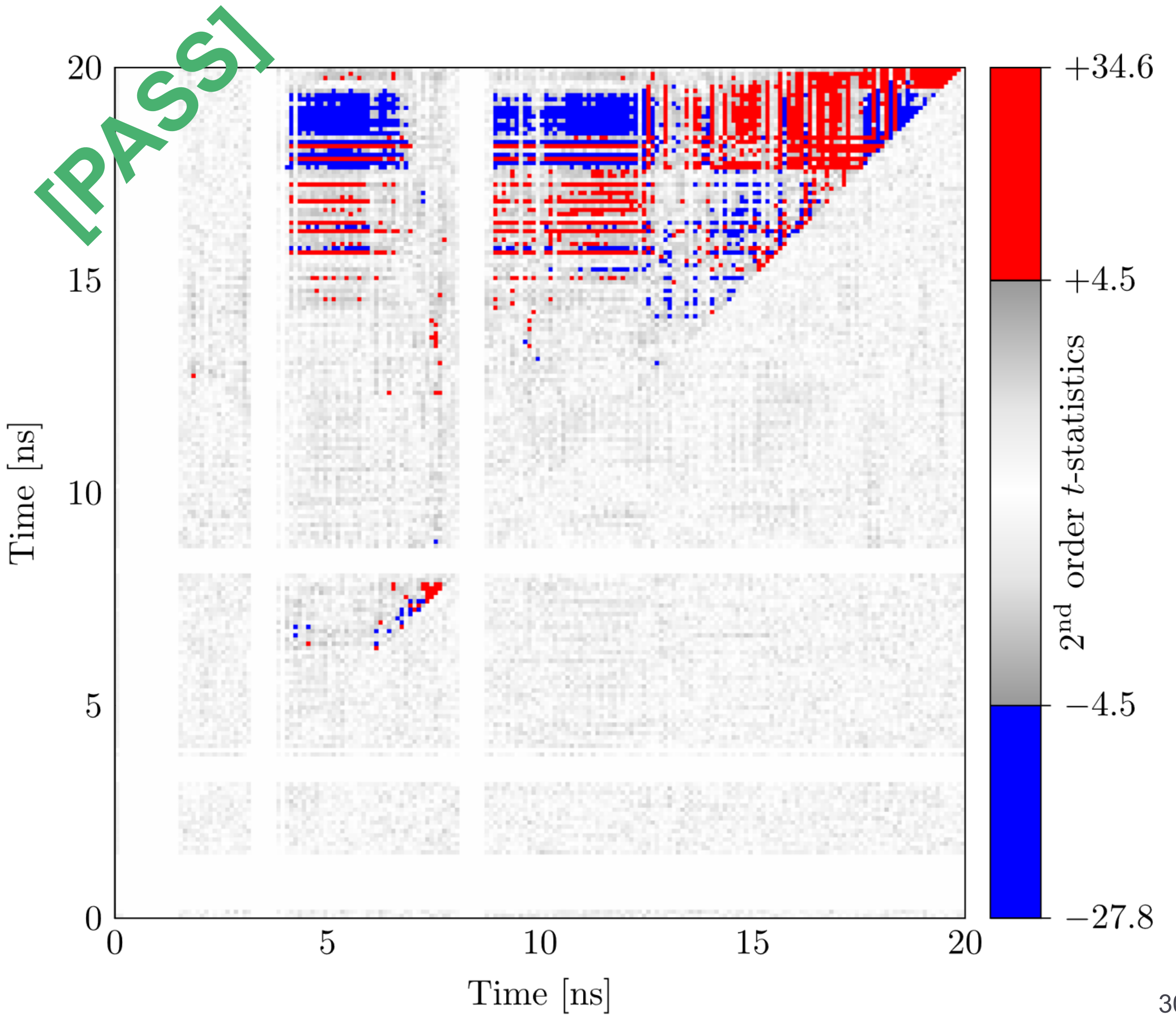
# Bivariate side-channel leakage assessment

Upper triangle: **masks are disabled**

Lower triangle: **masks are enabled**

Second order masking

10E6 simulated traces.



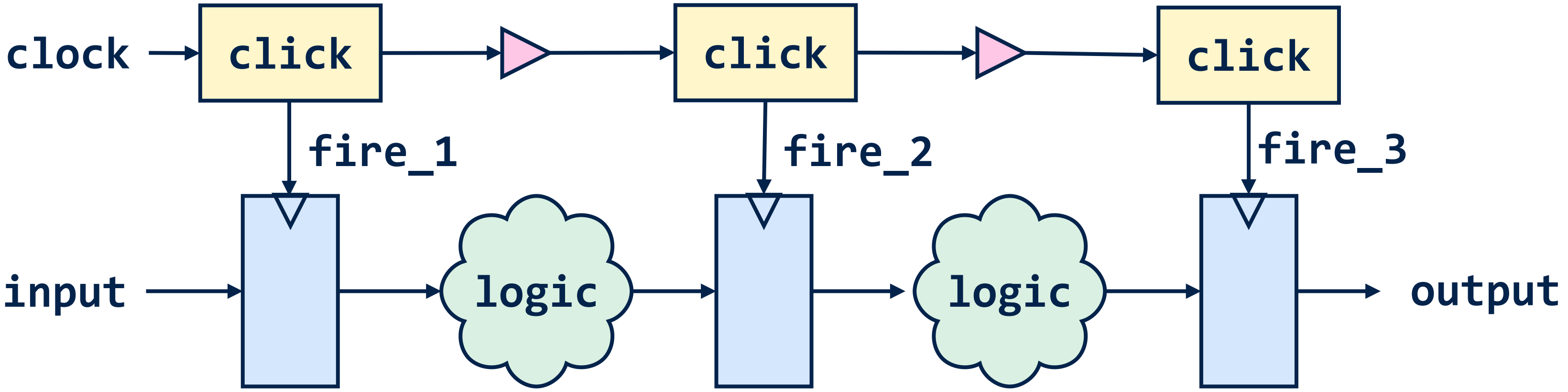

life.augmented

# Conclusion and Perspectives

# Secure, low-latency, and area-efficient masking

**How to mask complex hardware with low latency and secure against glitches?**

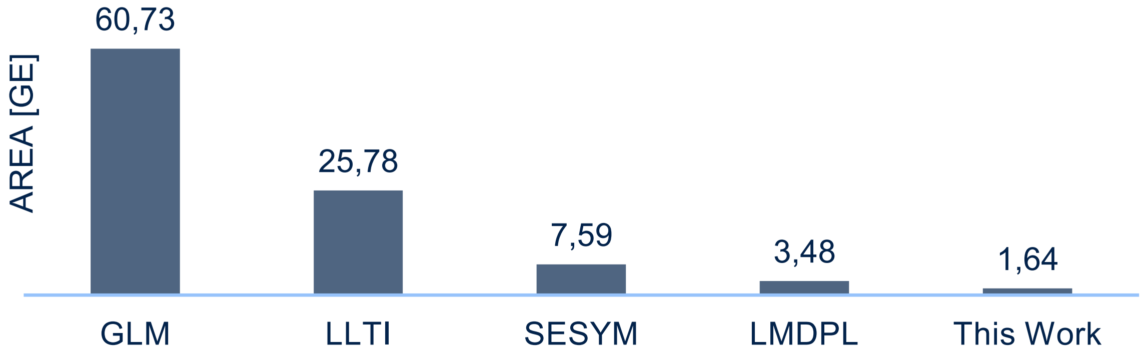
Masking + Click Elements [1]




**The click-based low-latency masked AES implementation is secure against SCA.**

`[PASS]` First-order univariate analysis.
`[PASS]` Second-order bivariate analysis.

A state-of-the-art area comparison

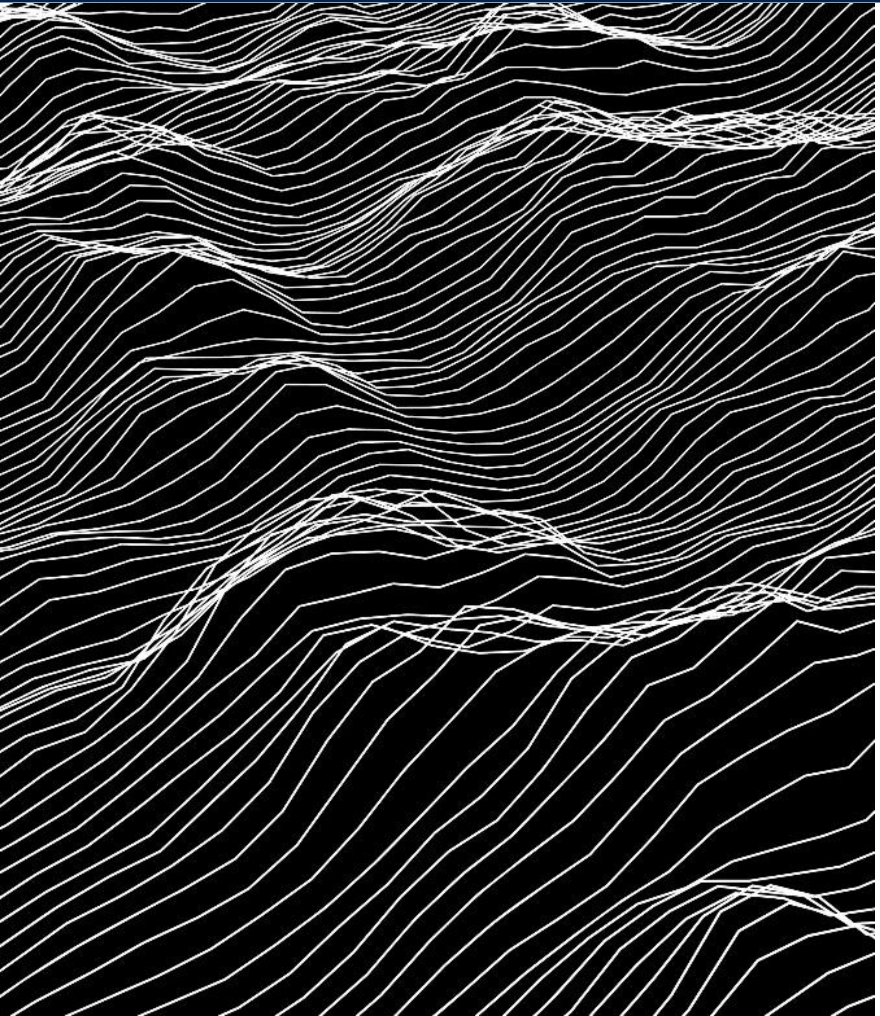



[1] M. Simões et al. “Low-Latency Masking with Arbitrary Protection Order Based on Click Elements”. In HOST 2023.

life.augmented

# Conclusion and perspectives

**Click-based masking is a generic approach to implement $d$-order secure low-latency circuits.**



**Pros**

+ Low-latency masking with low area overhead.
+ Hardware design approach.
+ Potential low power consumption.
+ Asynchronous design using commercial tools.
+ Arbitrary number of shares → arbitrary protection order.

**Cons**

- Use of `set_min_delay` and `dont_touch` constraints.
- Lower operation frequency due to larger data path.

**For potential future work.**

- Enhance frequency.
- Implement click-based masking on a real test chip.

# Our technology starts with You

Find out more at www.st.com



life.augmented