Handcrafting: Improving Automated Masking in Hardware with Manual Optimizations

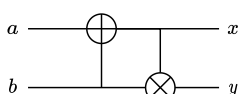
Charles Momin, Cassiers Gaëtan, François-Xavier Standaert

UCLouvain

Masking: top level view

Basic principle of block cipher masking:

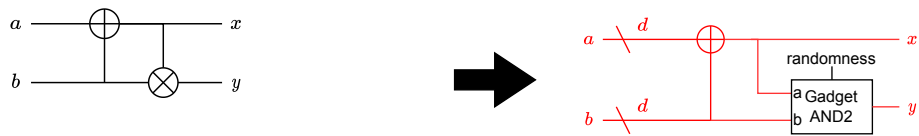
- 1 wire $\rightarrow$ d wires (sharing)
- logical gates $\rightarrow$ gadgets



Masking: top level view

Basic principle of block cipher masking:

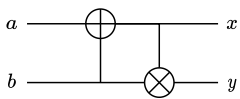
- 1 wire $\rightarrow$ d wires (sharing)
- logical gates $\rightarrow$ gadgets

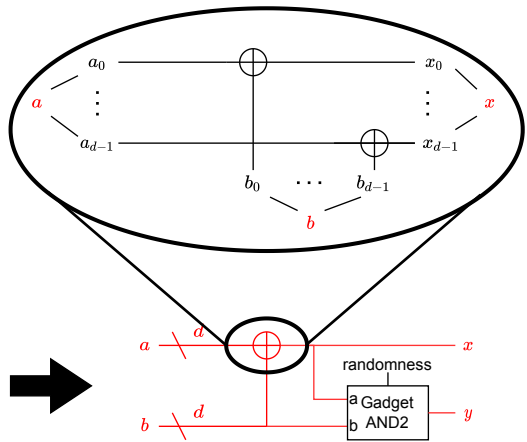


Masking: top level view

Basic principle of block cipher masking:

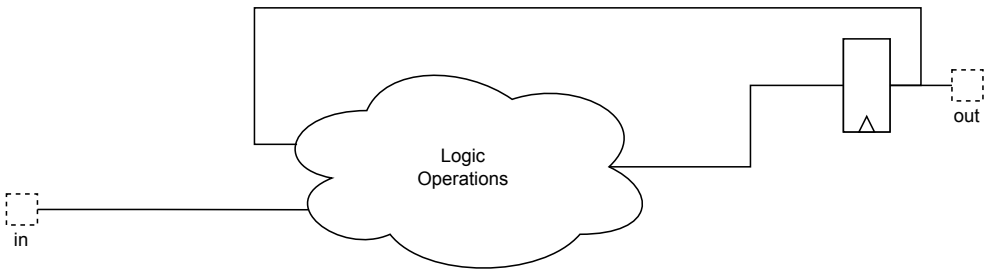
- 1 wire $\rightarrow$ d wires (sharing)
- logical gates $\rightarrow$ gadgets





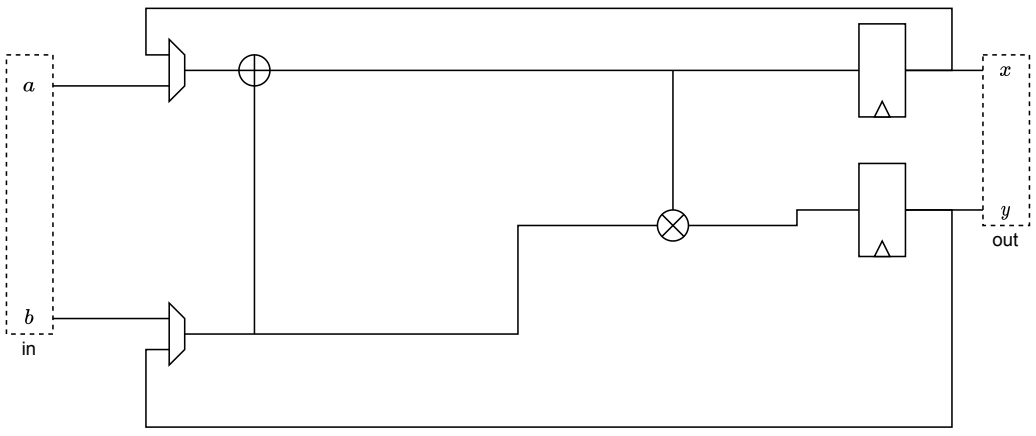
Masking in hardware

A generic hardware implementation



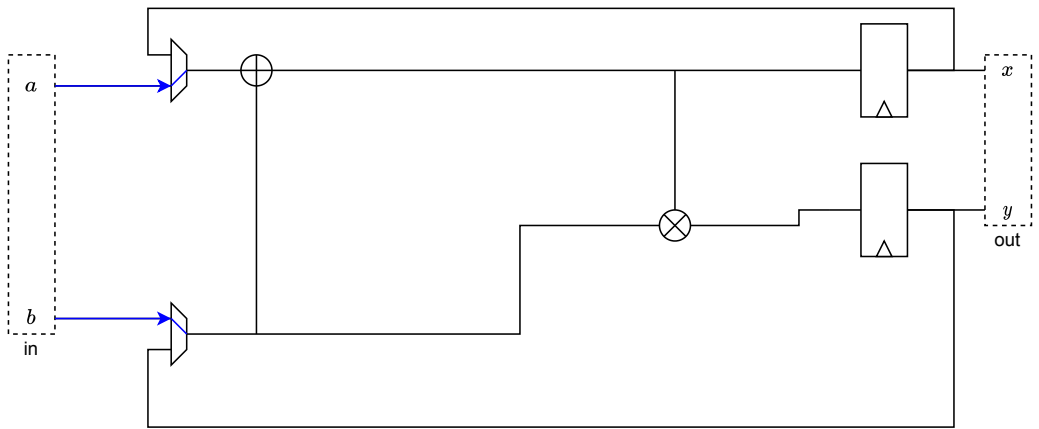
Masking in hardware

More practically



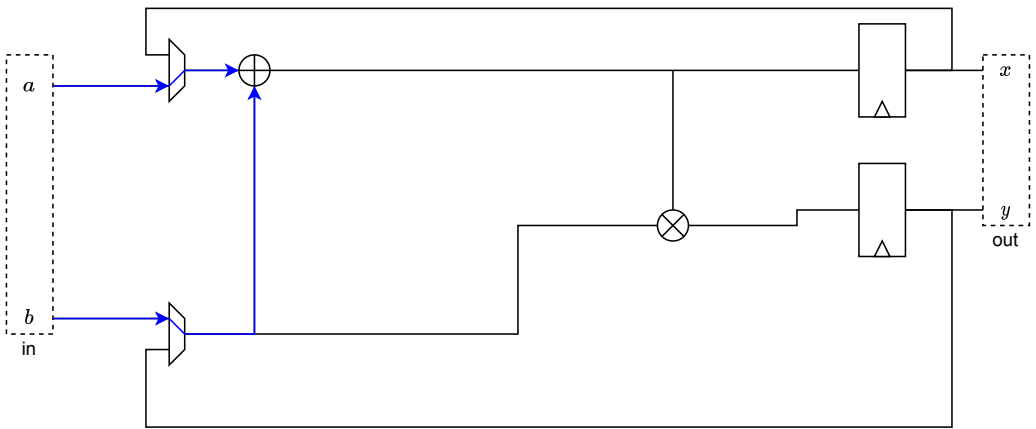
Masking in hardware

How do signals propagate? Cycle 0



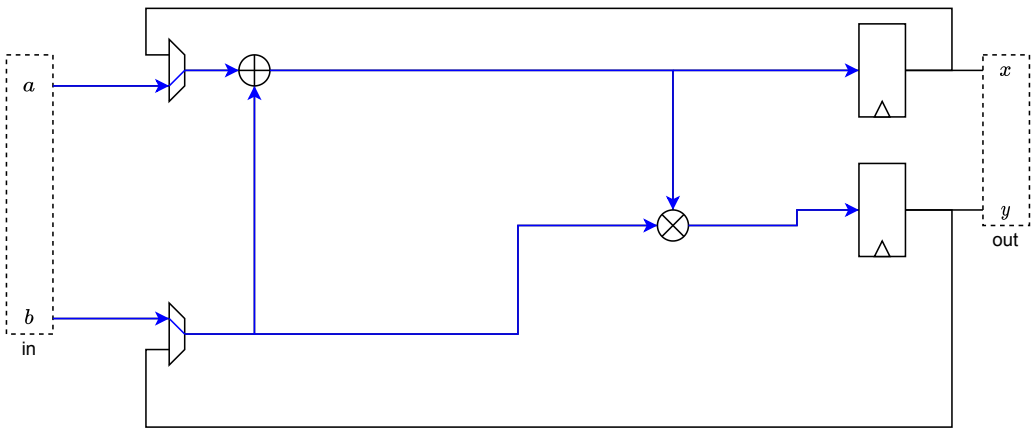
Masking in hardware

How do signals propagate? Cycle 0



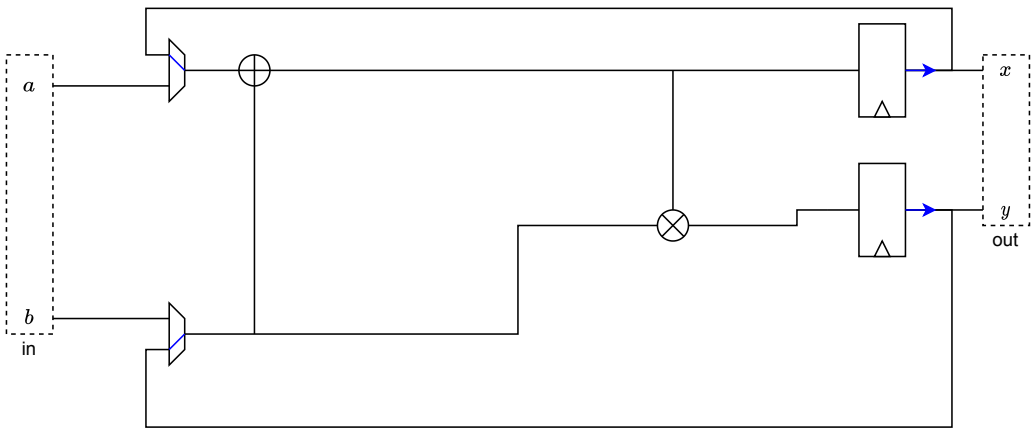
Masking in hardware

How do signals propagate? Cycle 0



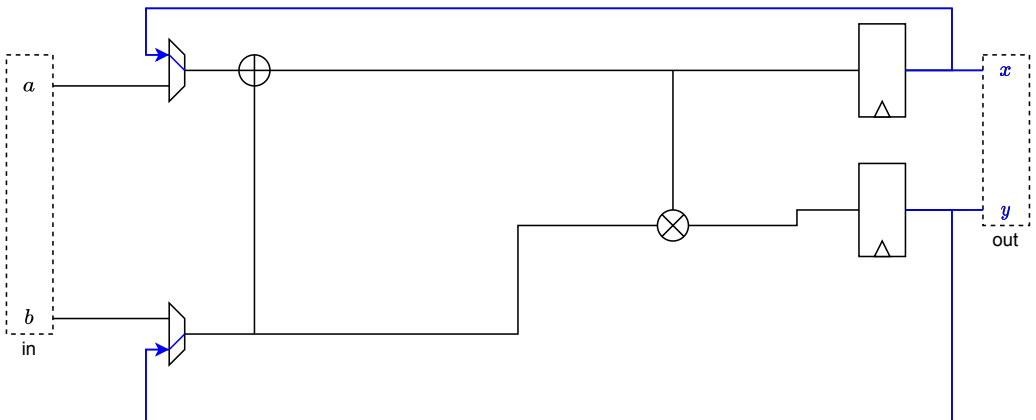
Masking in hardware

How do signals propagate? Cycle 1



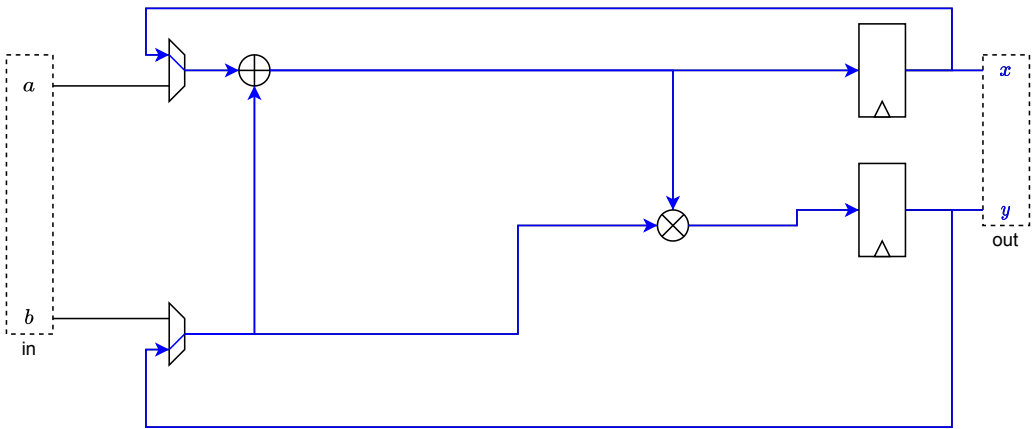
Masking in hardware

How do signals propagate? Cycle 1



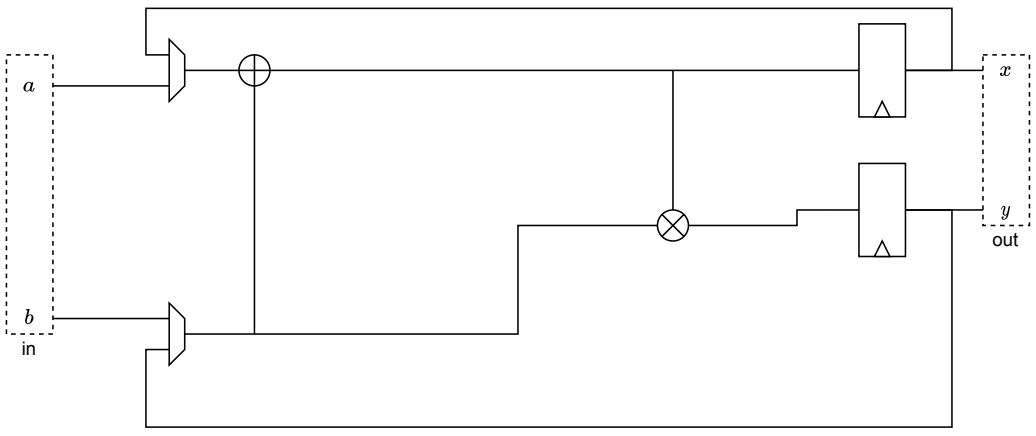
Masking in hardware

How do signals propagate? Cycle 1



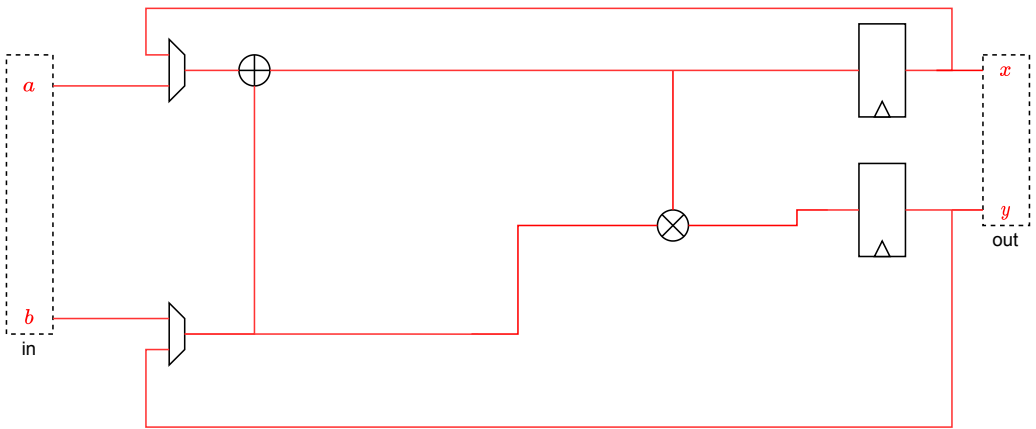
Masking in hardware

Lets convert to a masked implementation...



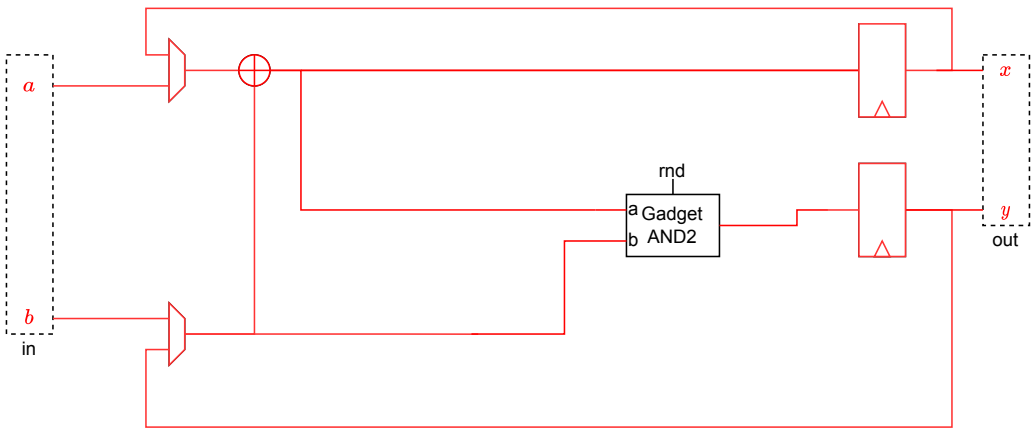
Masking in hardware

Data become sharing



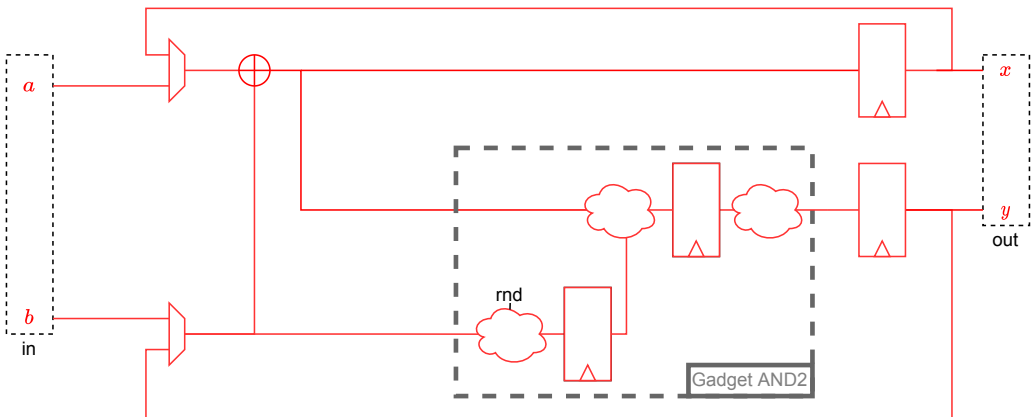
Masking in hardware

Operations become gadgets



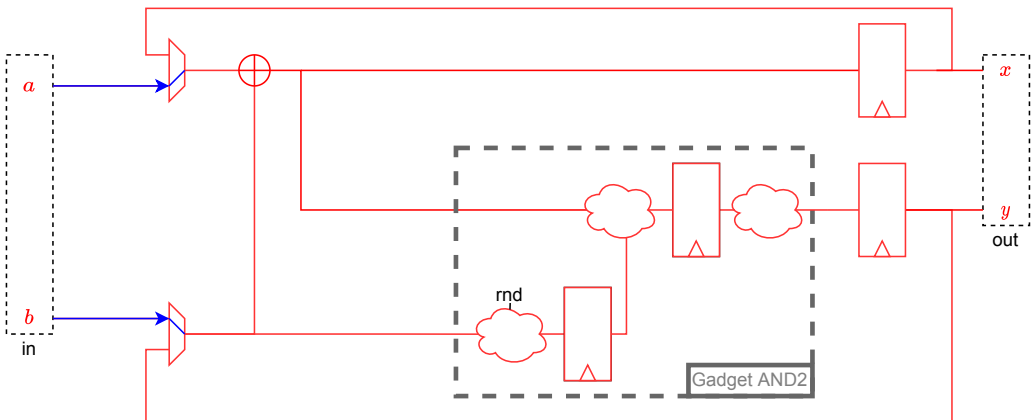
Masking in hardware

Example of AND2 gadget



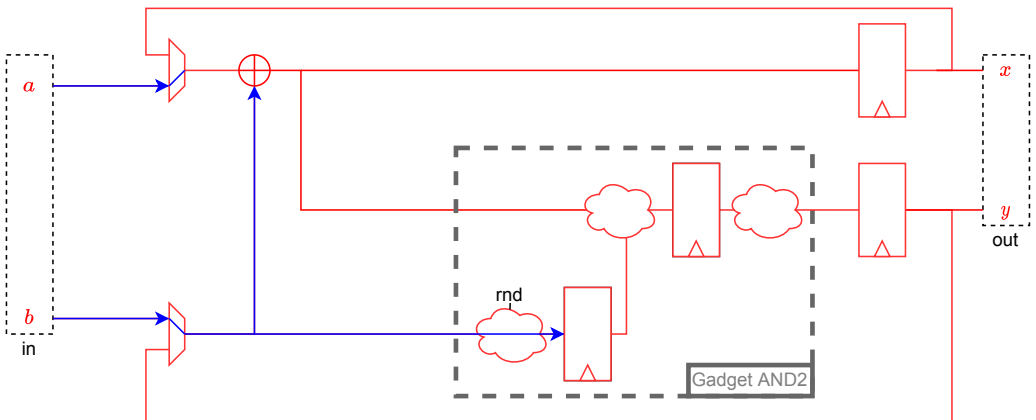
Masking in hardware

Is it still working? Cycle 0



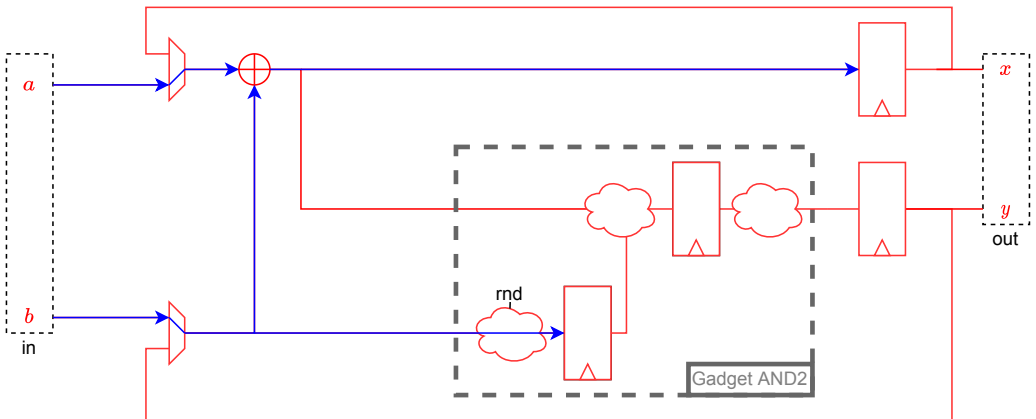
Masking in hardware

Is it still working? Cycle 0



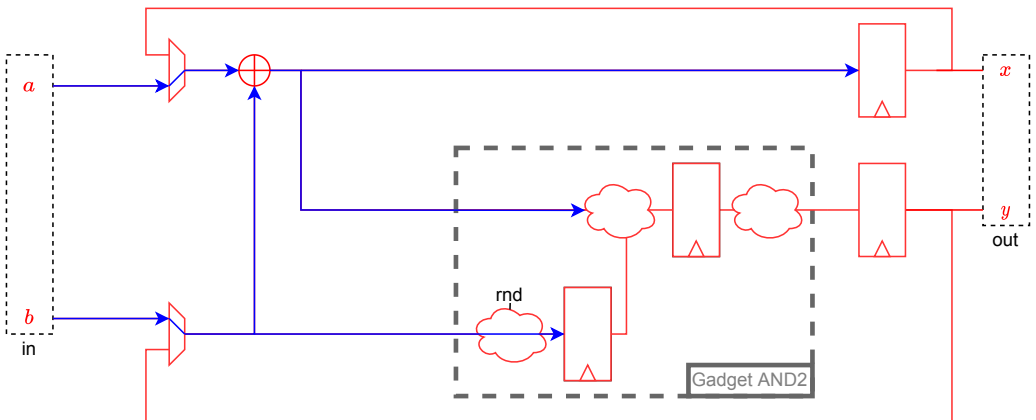
Masking in hardware

Is it still working? Cycle 0



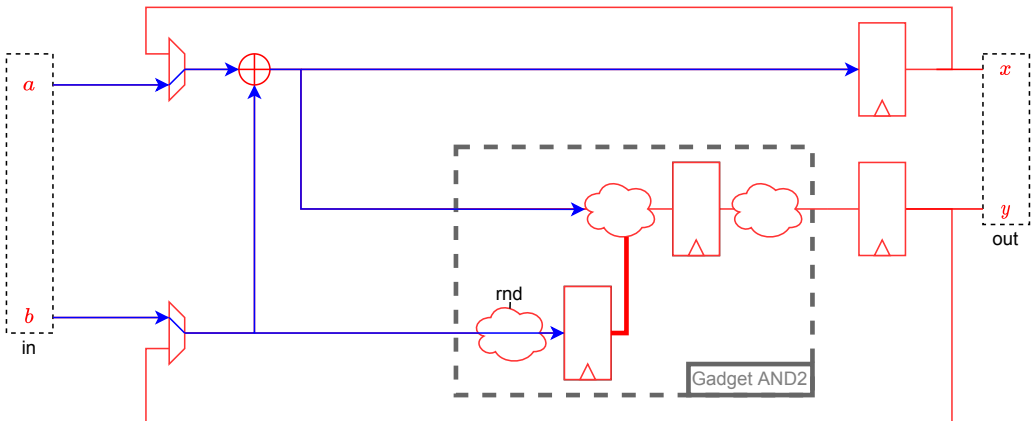
Masking in hardware

Is it still working? Cycle 0



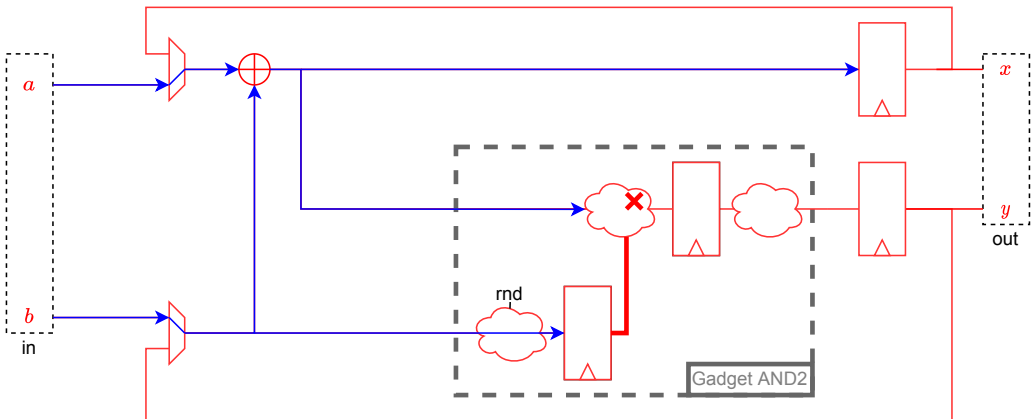
Masking in hardware

Is it still working? Cycle 0



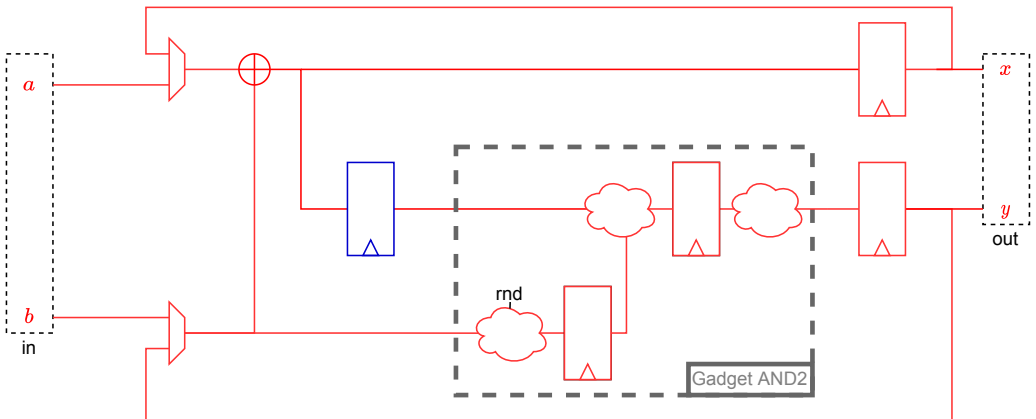
Masking in hardware

Failure due to synchronization mismatch! Cycle 0



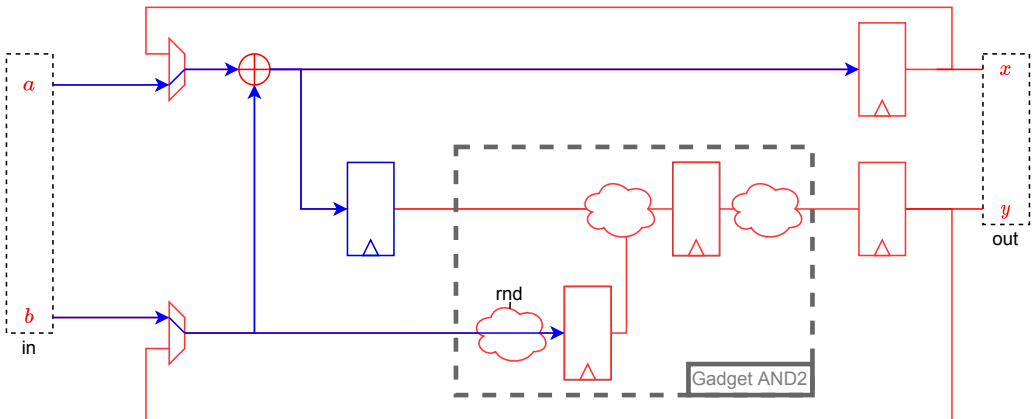
Masking in hardware

Let's try to add latency...



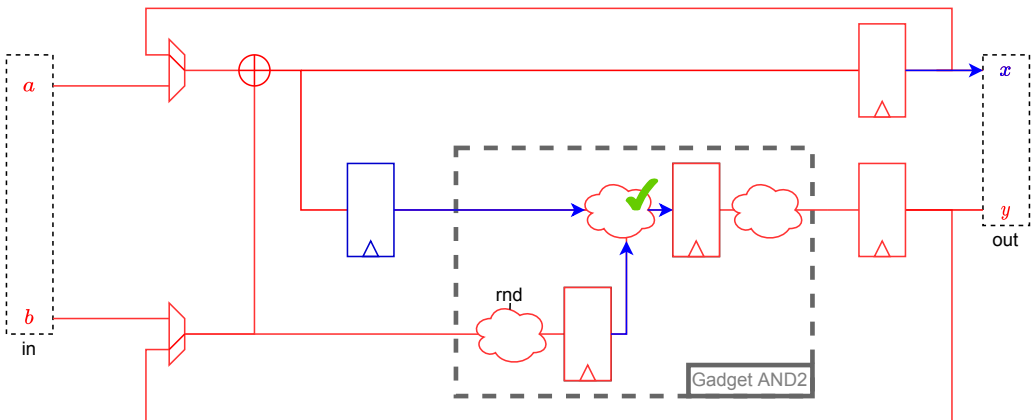
Masking in hardware

Is it fixed? Cycle 0



Masking in hardware

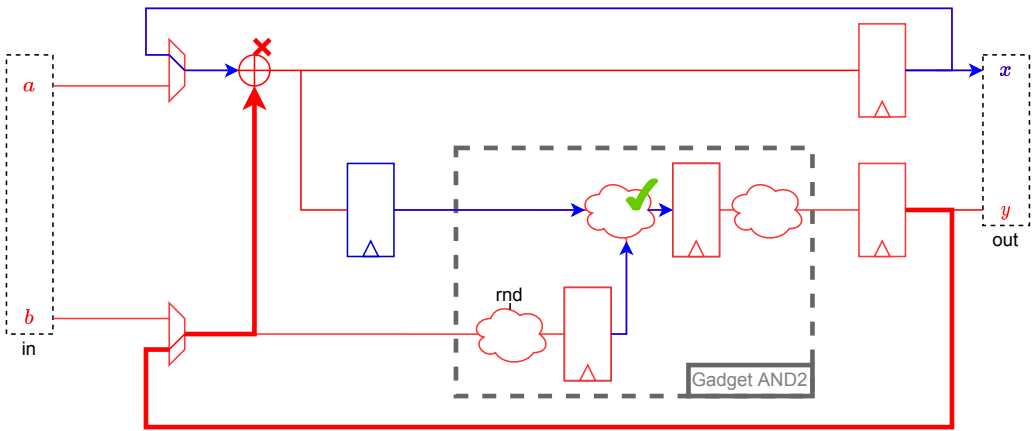
Ok for our previous bug! Cycle 1



Masking in hardware

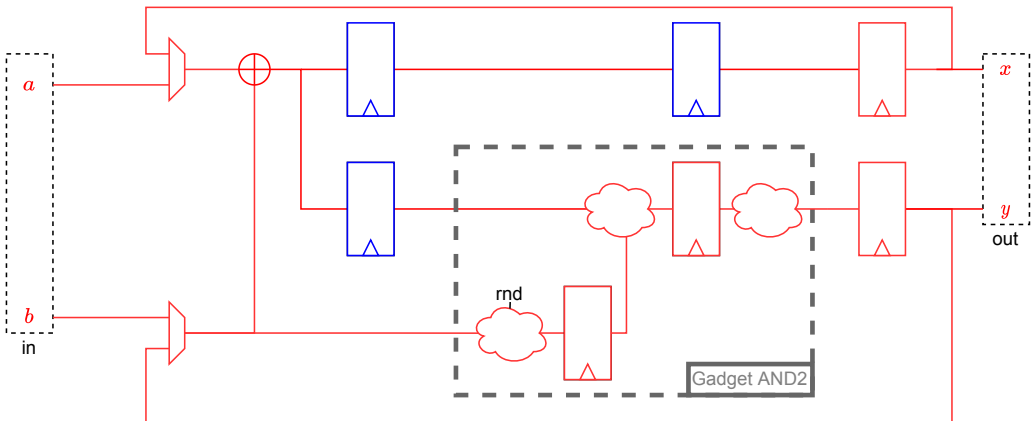
However not all bugs fixed...

Cycle 1



Masking in hardware

All signals must be synchronized!



Hardware masking: challenges

Implementation constraints

- Functionality
- Security
- Performance

Hardware masking: challenges

Implementation constraints

- Functionality
- Security
- Performance

From an implementer point of view

- Requires expertise
- Error prone ($\rightarrow$ verification tool)
- Time consuming

  $\Rightarrow$ Automated implementation tool.

AGEMA tool

Automatic generation of masked hardware circuit (Knichel et al., 2022):

- Functionality: masking + synchronization.
- Security relies on proven gadgets implementations.
- Variants:
  - Pipeline: high throughput.
  - Clock-gating: reduced area.

AGEMA tool

Automatic generation of masked hardware circuit (Knichel et al., 2022):

- Functionality: masking + synchronization.
- Security relies on proven gadgets implementations.
- Variants:
  - Pipeline: high throughput.
  - Clock-gating: reduced area.

$\Rightarrow$ Are masked HW implementers outdated?

Contributions

There is still room for improvement!

In particular for

- latency,
- area.

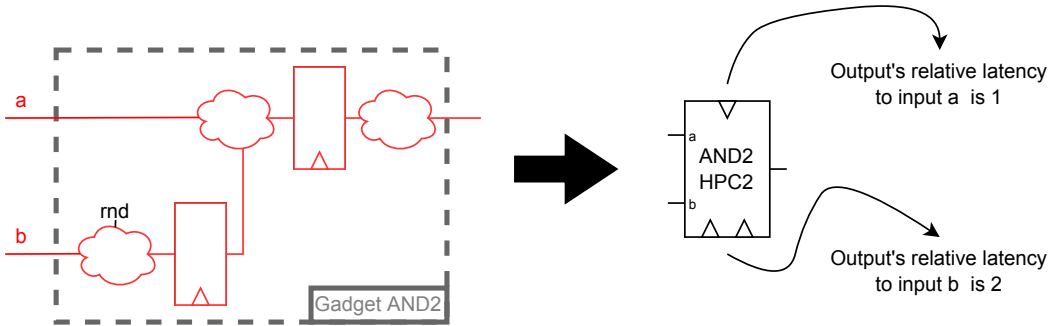
Our contributions:

- Generic netlist optimization strategy.
- New optimized masked AES architecture.

Case study: HPC2 masking scheme

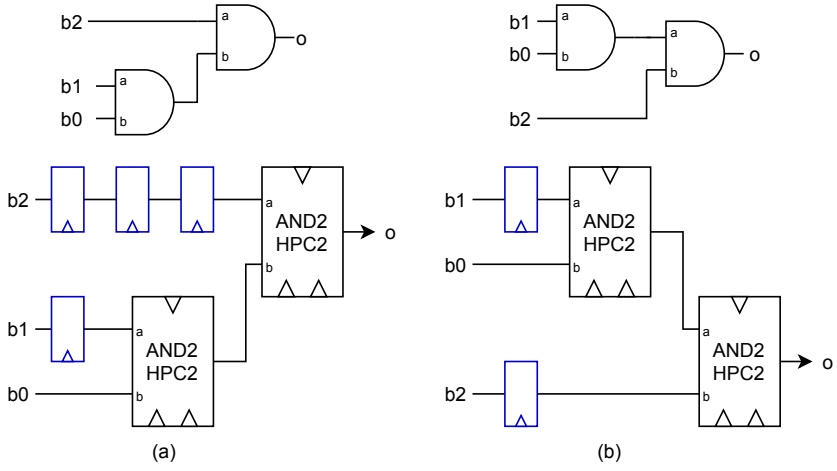
The HPC2 AND gadget has asymmetric latency.

- input b valid at cycle t
- input a valid at cycle $t + 1$
- ouput valid at cycle $t + 2$



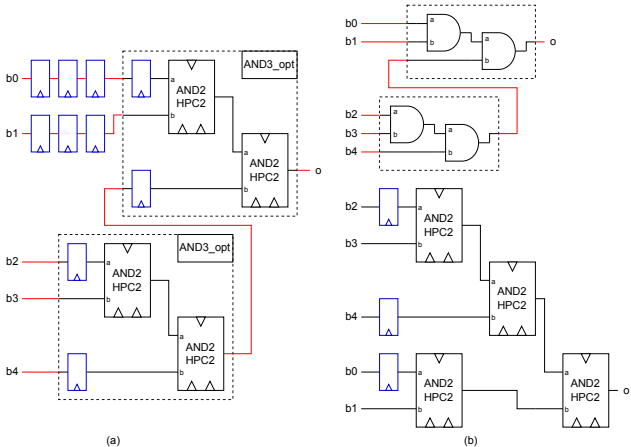
Netlist optimisation strategy (AND3 case)

Let's try to take advantage of the asymmetry



Netlist optimisation strategy (AND5 case)

Global optimisation required.



Is it working? AES Sbox

Bit-level Sbox representation used (Boyar & Peralta, 2012, same as AGEMA).

- 34 AND gates, 93 XOR gates
- 4 levels AND depth

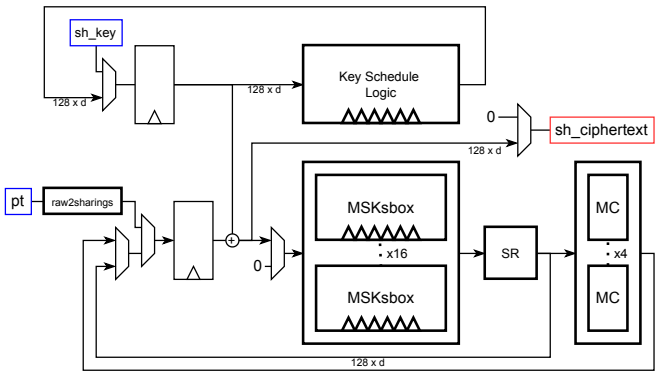
Pipeline AES Sbox optimization:

Shares	Latency [%]	Area [%]
2	−25	−19
3	−25	−20
4	−25	−20

ASIC implementation.

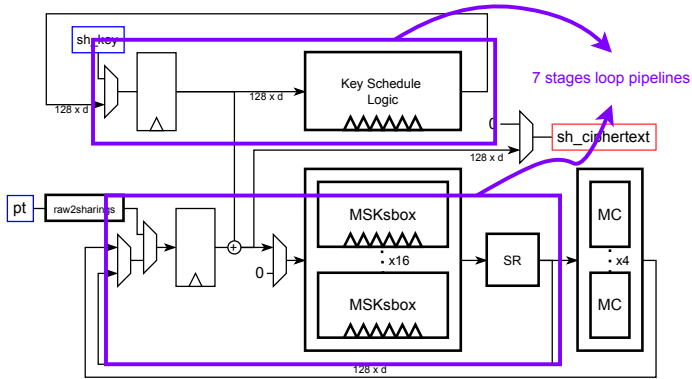
128-bits AES architecture

20 Sboxes (16 for rounds, 4 for key schedule)

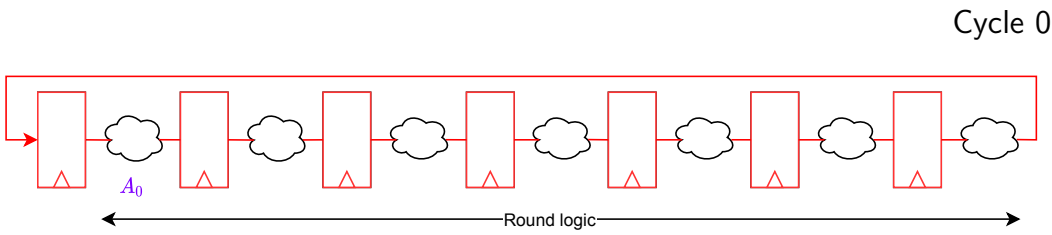


128-bits AES architecture

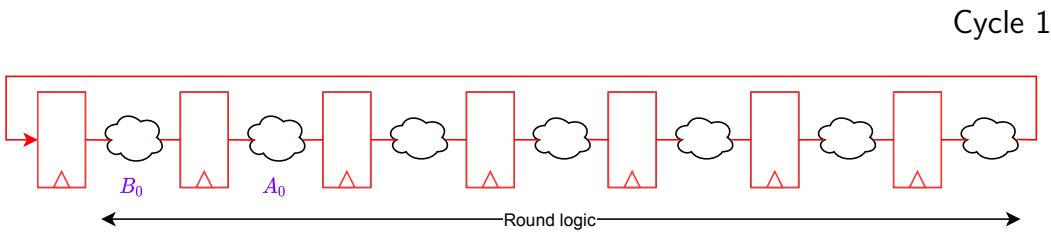
20 Sboxes (16 for rounds, 4 for key schedule)



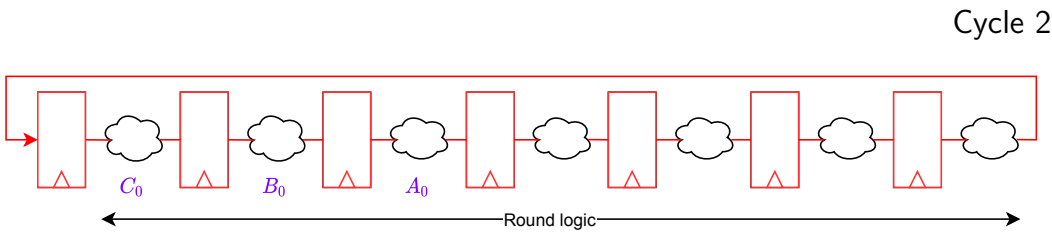
128-bits AES architecture



128-bits AES architecture

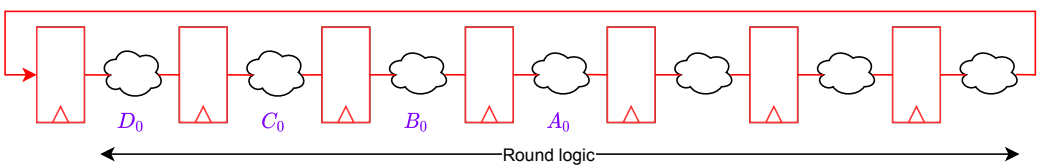


128-bits AES architecture



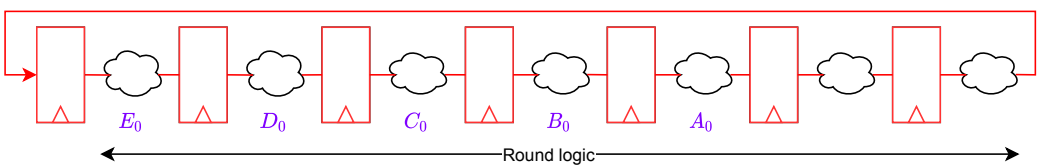
128-bits AES architecture

Cycle 3



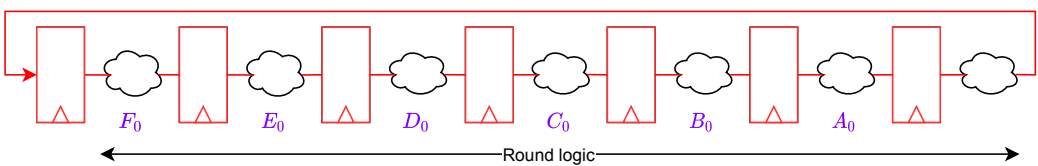
128-bits AES architecture

Cycle 4



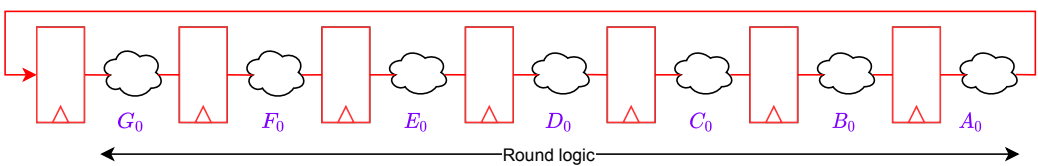
128-bits AES architecture

Cycle 5



128-bits AES architecture

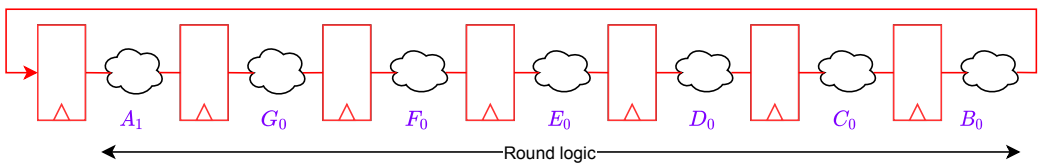
Cycle 6



128-bits AES architecture

1^{st} round finished

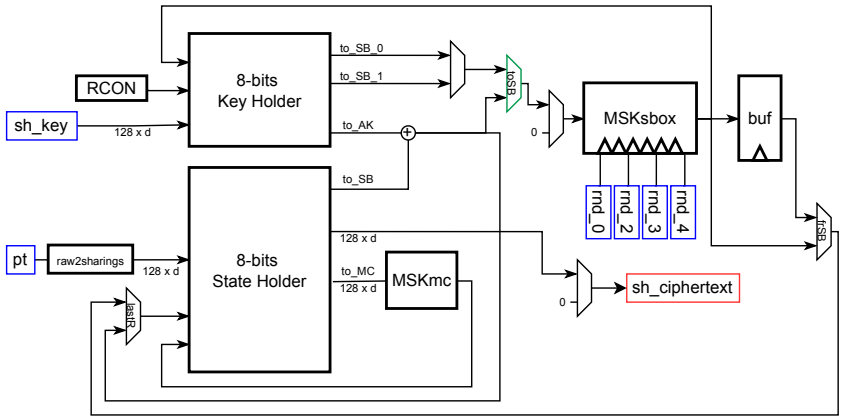
Cycle 7



- 7 parallel executions.
- Similar to AGEMA's, better thanks to new Sbox.

8-bits serial AES implementation

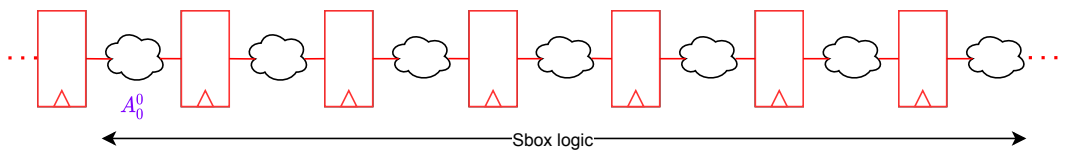
1 Sbox used, shared between rounds and key scheduling.



8-bits serial AES implementation

With the pipeline representation

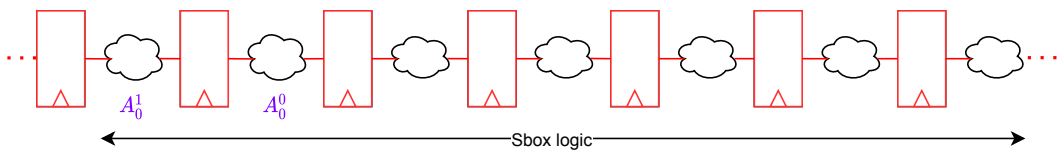
Cycle 0



8-bits serial AES implementation

With the pipeline representation

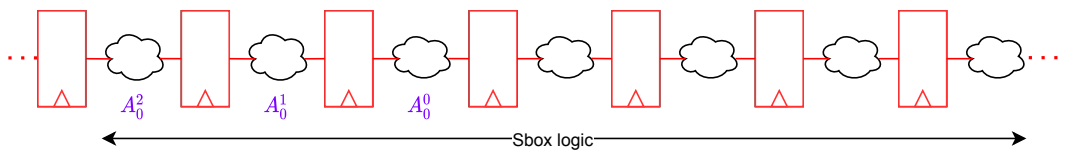
Cycle 1



8-bits serial AES implementation

With the pipeline representation

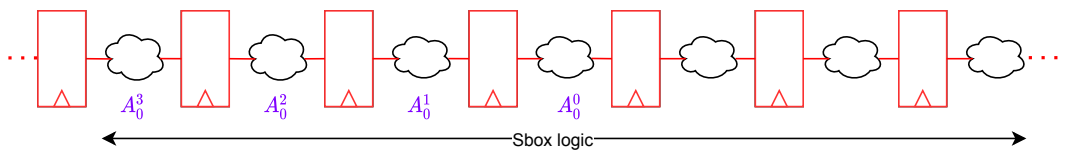
Cycle 2



8-bits serial AES implementation

With the pipeline representation

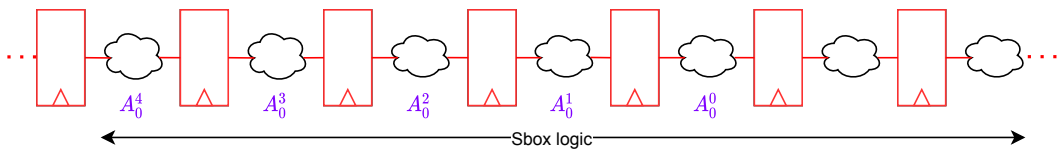
Cycle 3



8-bits serial AES implementation

With the pipeline representation

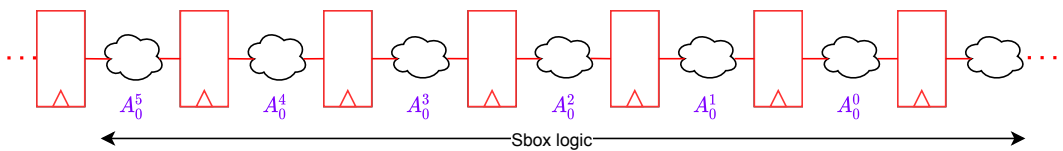
Cycle 4



8-bits serial AES implementation

With the pipeline representation

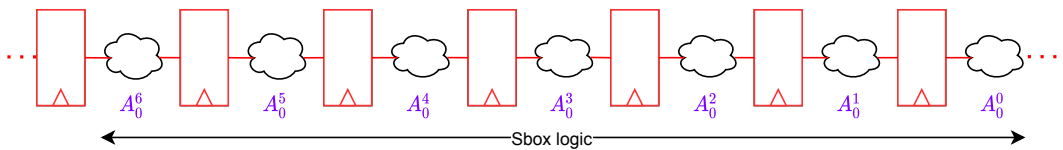
Cycle 5



8-bits serial AES implementation

With the pipeline representation

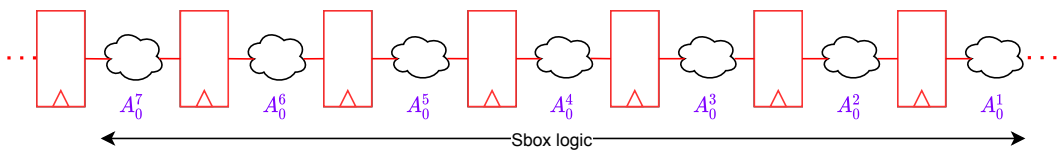
Cycle 6



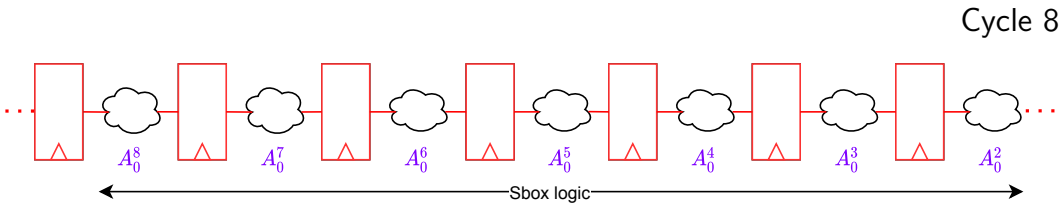
8-bits serial AES implementation

With the pipeline representation

Cycle 7



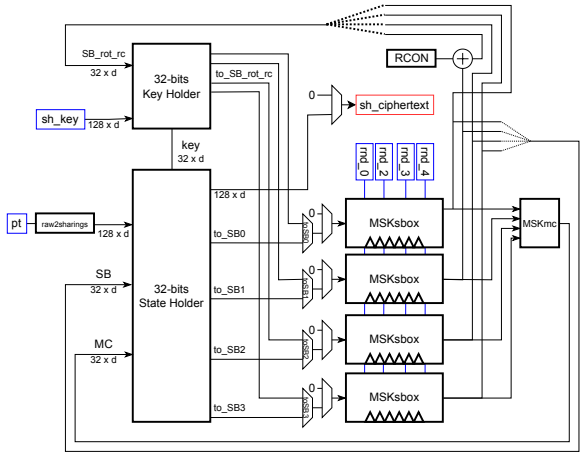
8-bits serial AES implementation



- 1 single execution with continuous feeding.
- Latency reduction (area and/or throughput cost).

32-bits serial AES implementation

4 Sboxes , shared between rounds and key scheduling.



How does it compare?

Instance	Share [count]	Seq. area [GE]	Area [GE]	Latency [cycle]	Throughput [exec/cycle]
AGEMA 8-bit c.g.*	4	12 375	24 919	2043	0.00049
AGEMA 8-bit pipe.†	4	53 382	65 921		0.0044
New 8-bit	4	13 315	25 915	322	0.0031
New 32-bit	4	37 511	59 217	105	0.0095
AGEMA 128-bit c.g.*	4	167 376	254 948	99	0.010
AGEMA 128-bit pipe.†	4	259 307	346 880		0.091
New 128-bit	4	171 274	249 011	70	0.1

* Clock-gating synchronization mechanism.

† Pipeline synchronization mechanism.

ASIC TSMC-N65 AES implementation.

Conclusion

- AGEMA = significant advance to ease masked design implementation...
- ... But there is still room for human designer.
- For how long?
  - Generic optimisation strategy could be easily automated.
  - Less clear for continuous pipeline feeding.

Questions

Questions?