Mitigation of Power-based Side-channel Leakage using custom Firmware and Micro-Architecture

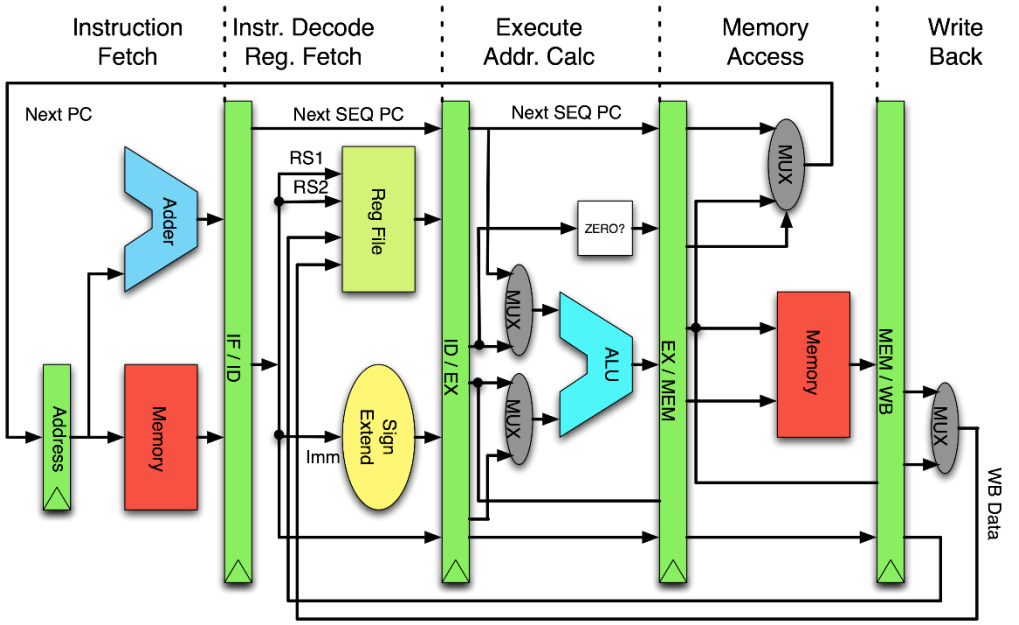
Patrick Schaumont and Pantea Kiaei

ECE Department
Worcester Polytechnic Institute

Power Based SCL in a Microprocessor

```
sw      zero,0(a5)
lw      a5,-20(s0)
andi    a5,a5,255
mv      a0,a5
```

ISA



Power Based SCL in a Microprocessor

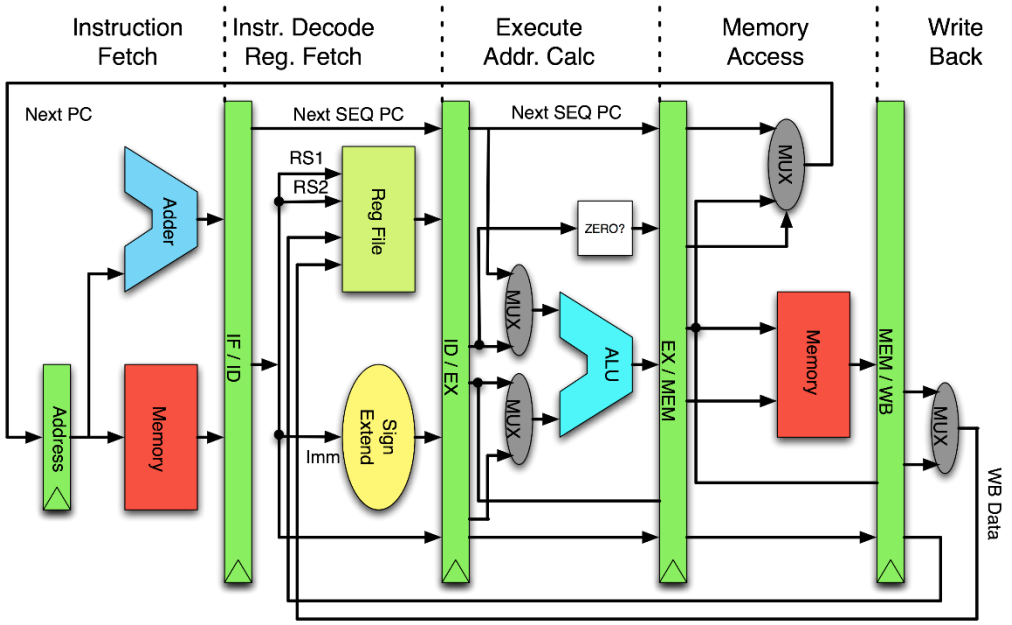
- Secrets injected through **software**
- Propagate through data dependencies

```
sw      zero,0(a5)
lw      a5,-20(s0)
andi    a5,a5,255
mv      a0,a5
```

ISA

- Propagate through *architecture*
- Power effects originate from **hardware**

Challenge:
How to mitigate power-based SCL originating from software-based secrets?



Roadmap

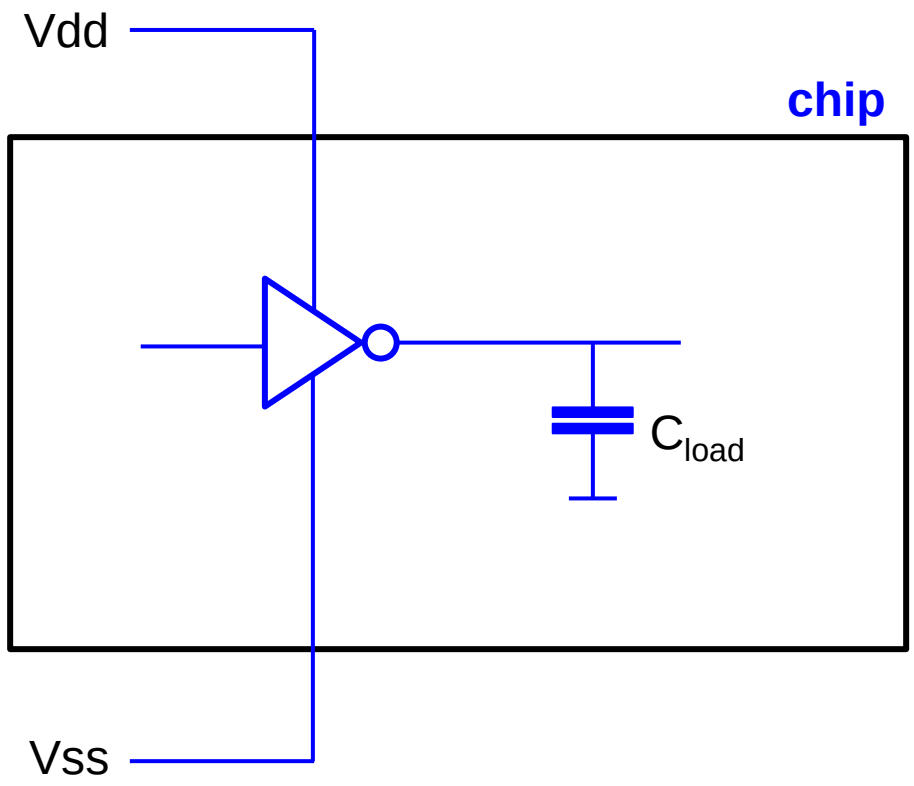
- **Power-based side-channel leakage across the ISA**
- **Secret Sharing as a Countermeasure - SKIVA**
- **Automatic Bitslice Design**
- **DOM-based Instruction-set Design for RISCV**
- **Further Reading**

Roadmap

- **Power-based side-channel leakage across the ISA**
- **Secret Sharing as a Countermeasure - SKIVA**
- **Automatic Bitslice Design**
- **DOM-based Instruction-set Design for RISCV**
- **Further Reading**

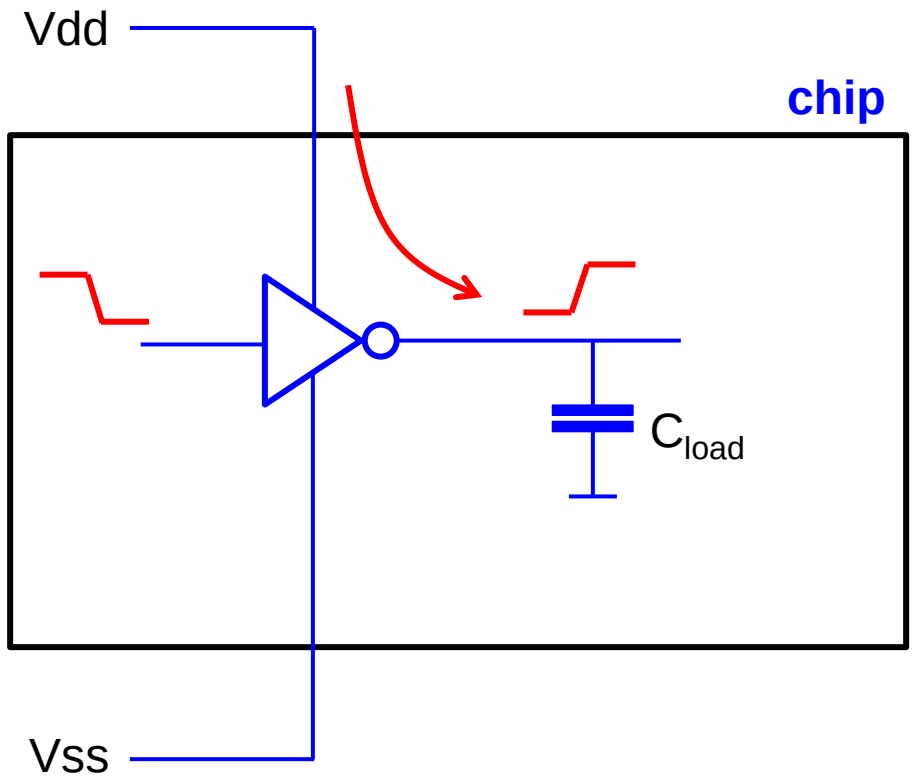
Power-based side-channel leakage

Any measurable data-dependent power dissipation may be a source of SCL:



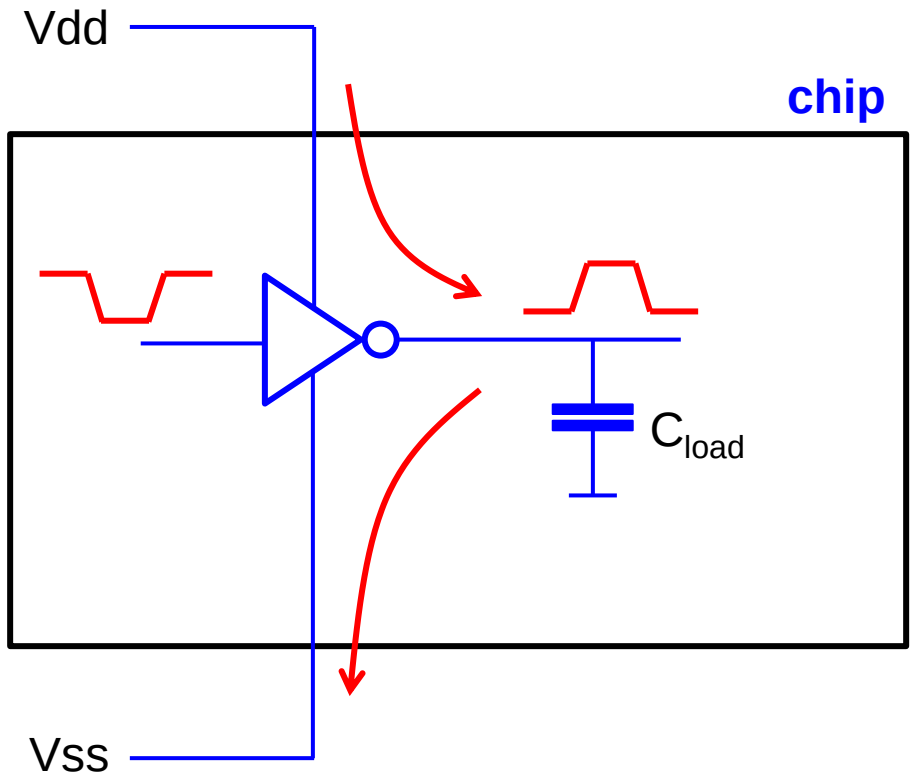
Power-based side-channel leakage

Any measurable data-dependent power dissipation may be a source of SCL:
(a) transitions



Power-based side-channel leakage

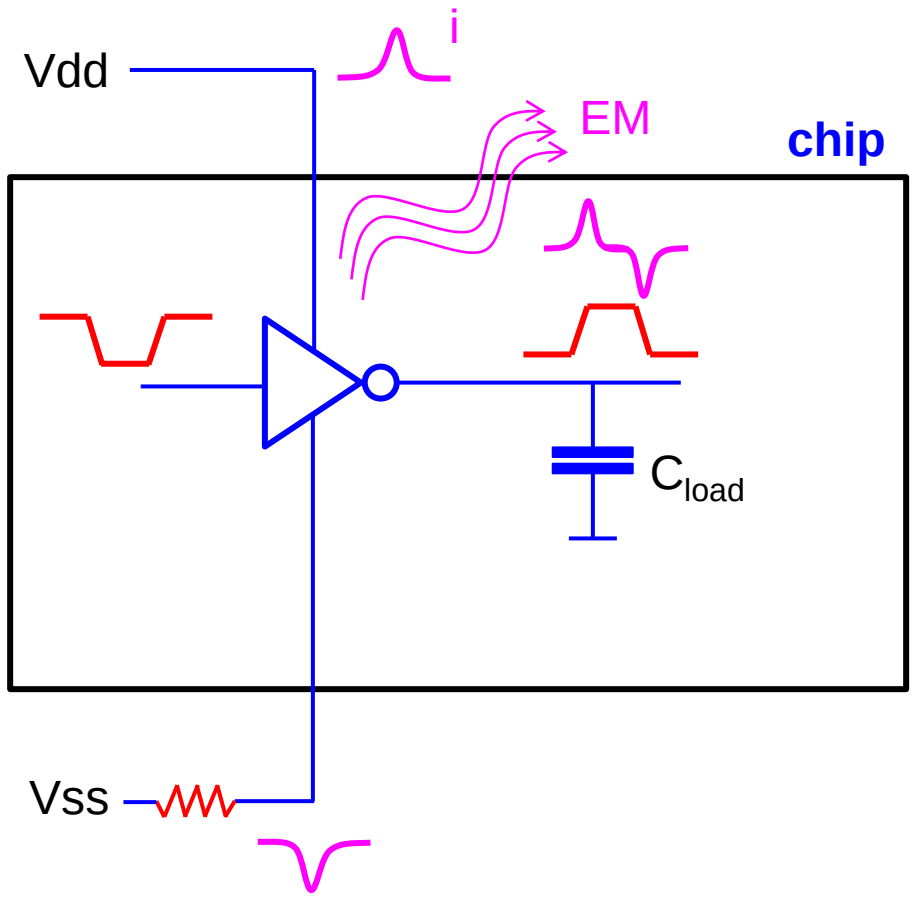
Any measurable data-dependent power dissipation may be a source of SCL:
(a) transitions



Power-based side-channel leakage

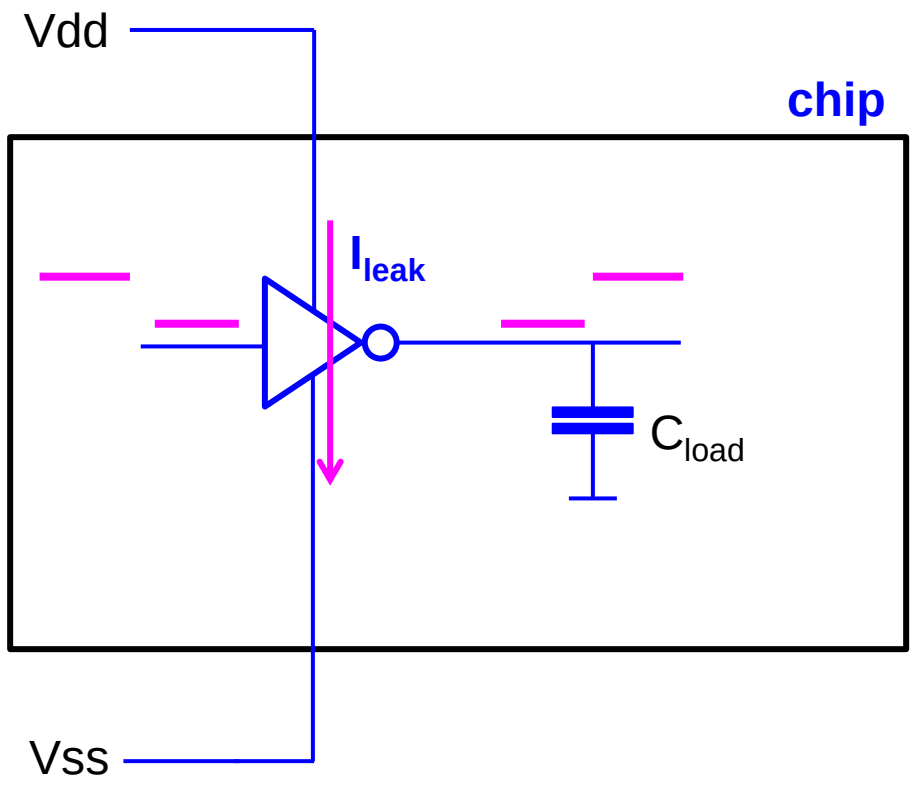
Any measurable data-dependent power dissipation may be a source of SCL:

(a) transitions



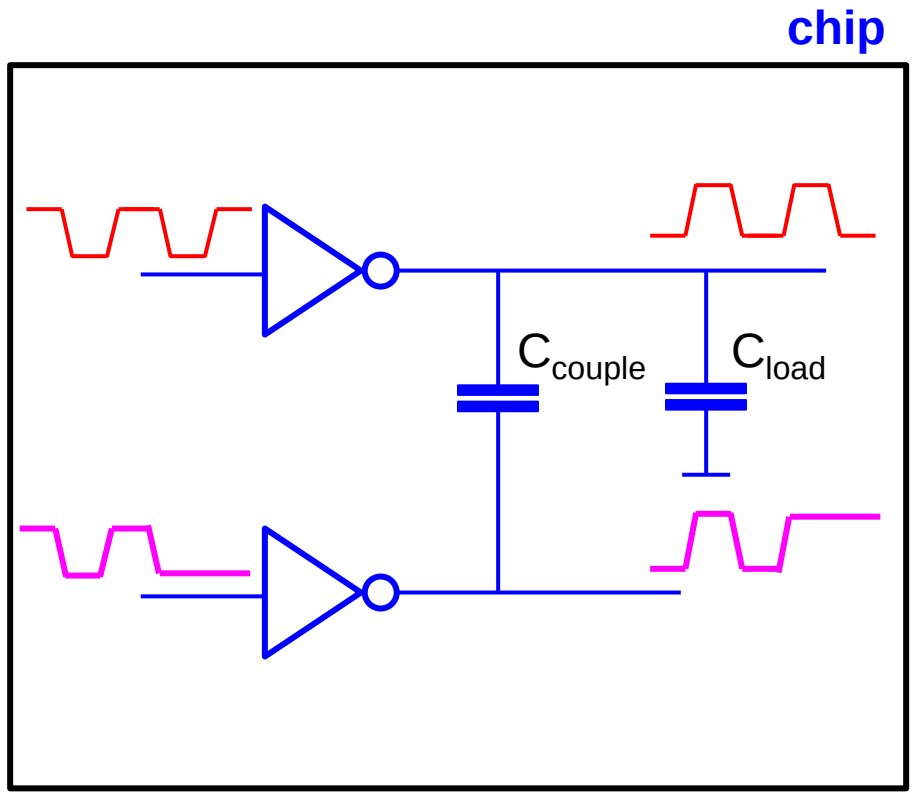
Power-based side-channel leakage

Any measurable data-dependent power dissipation may be a source of SCL:
(a) transitions, **(b) static power**



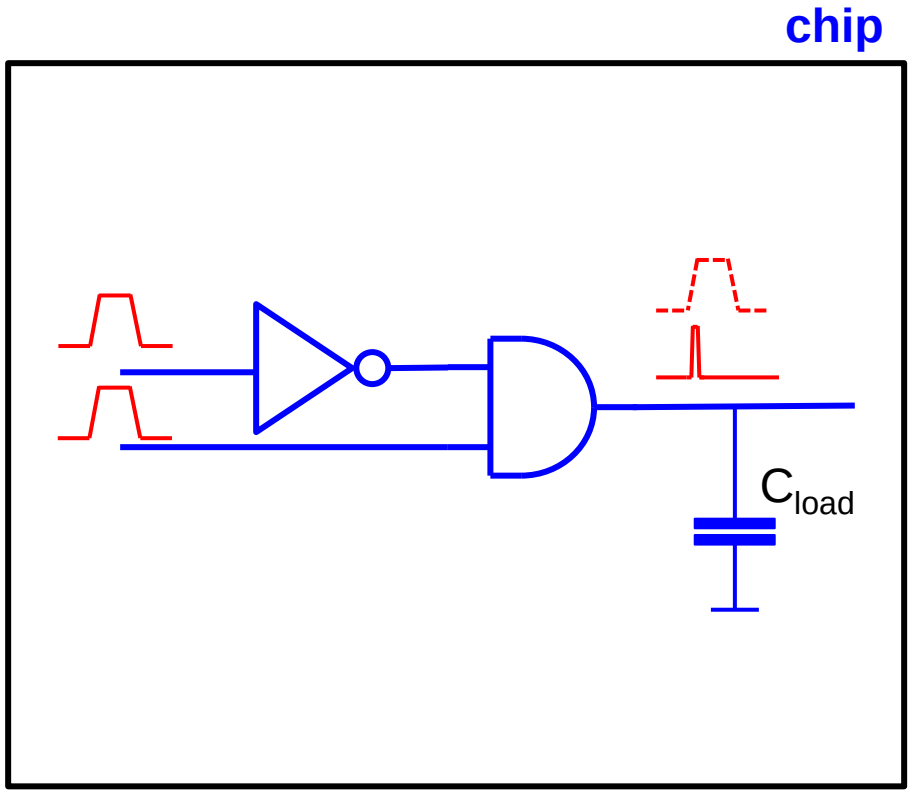
Power-based side-channel leakage

Any measurable data-dependent power dissipation may be a source of SCL:
(a) transitions, (b) static power, **(c) coupling C**



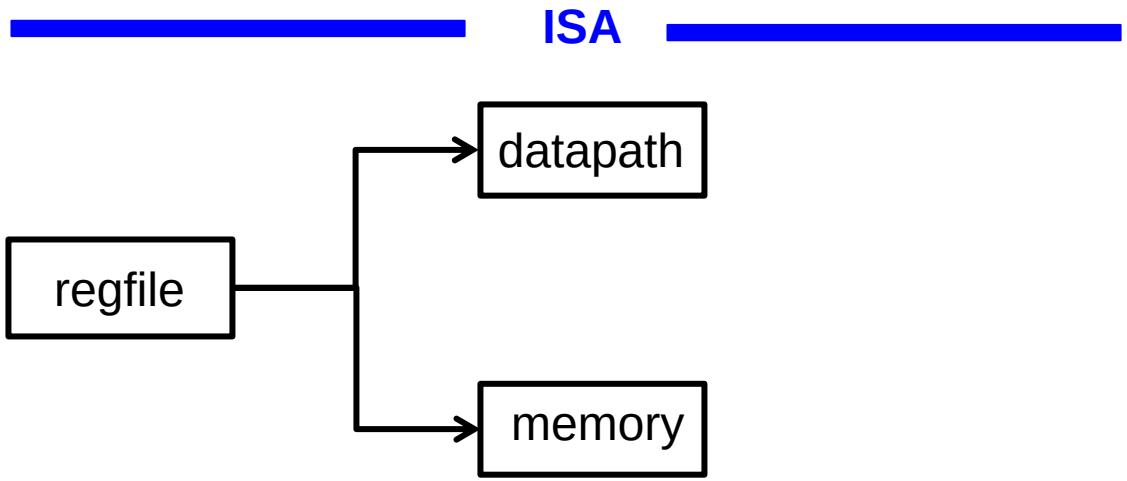
Power-based side-channel leakage

Any measurable data-dependent power dissipation may be a source of SCL:
(a) transitions, (b) static power, (c) coupling C, **(d) glitches**



Power-based SCL in architecture

```
sw      zero,0(a5)
lw      a5,-20(s0)
andi    a5,a5,255
mv      a0,a5
```

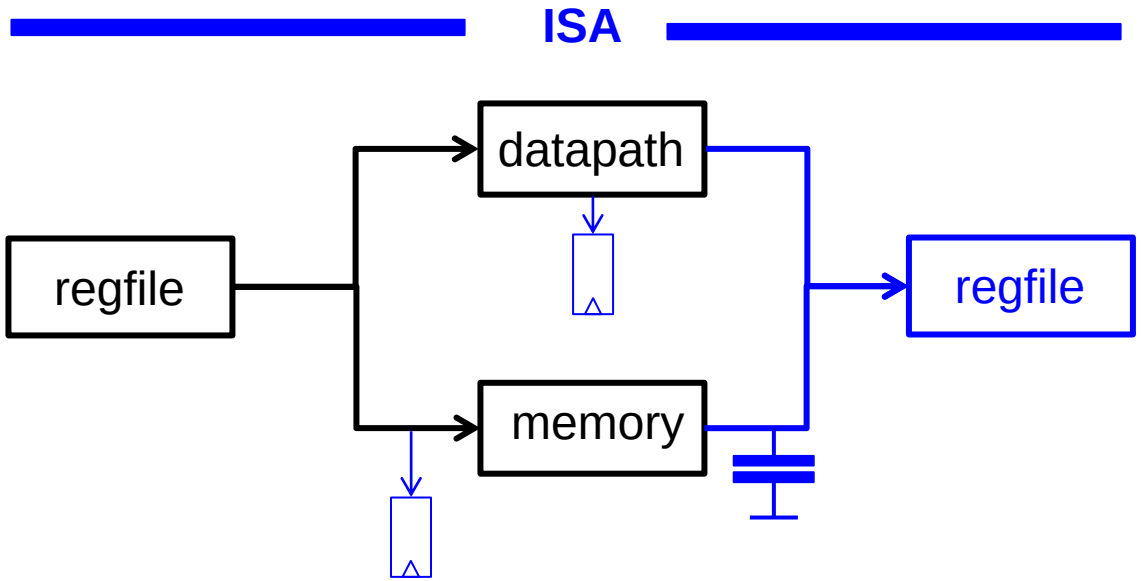


- Power depends on operands [ELMO]

[ELMO] D. McCann, C. Whitnall, E. Oswald: ELMO: Emulating Leaks for the ARM Cortex-M0 without Access to a Side Channel Lab. IACR Cryptol. ePrint Arch. 2016: 517 (2016).
[ROSITA] M. A. Shelton, N. Samwel, L. Batina, F. Regazzoni, M. Wagner, Y. Yarom: Rosita: Towards Automatic Elimination of Power-Analysis Leakage in Ciphers. CoRR abs/1912.05183 (2019).

Power-based SCL in architecture

```
sw      zero,0(a5)
lw      a5,-20(s0)
andi    a5,a5,255
mv      a0,a5
```



- Power depends on operands [ELMO]
- Power also depends on [ROSITA]
  - Data dependencies through registers
  - Write buffers on memory interface
  - Residual data on memory bus
  - Interactions through ISA-invisible state
  - ...

To mitigate power-based SCL, one needs to address architecture-level interactions from code

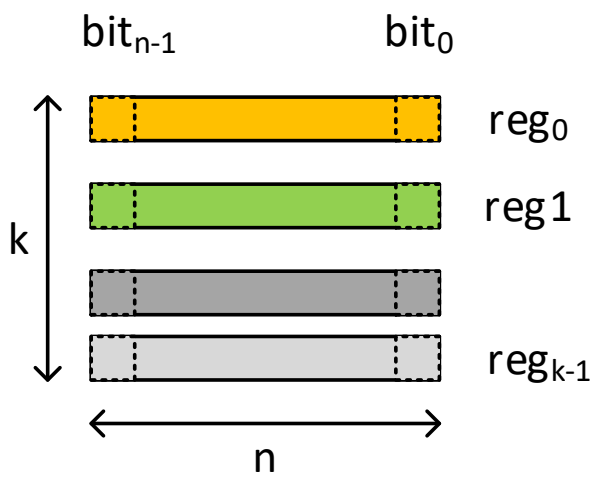
[ELMO] D. McCann, C. Whitnall, E. Oswald: ELMO: Emulating Leaks for the ARM Cortex-M0 without Access to a Side Channel Lab. IACR Cryptol. ePrint Arch. 2016: 517 (2016).
[ROSITA] M. A. Shelton, N. Samwel, L. Batina, F. Regazzoni, M. Wagner, Y. Yarom: Rosita: Towards Automatic Elimination of Power-Analysis Leakage in Ciphers. CoRR abs/1912.05183 (2019).

Roadmap

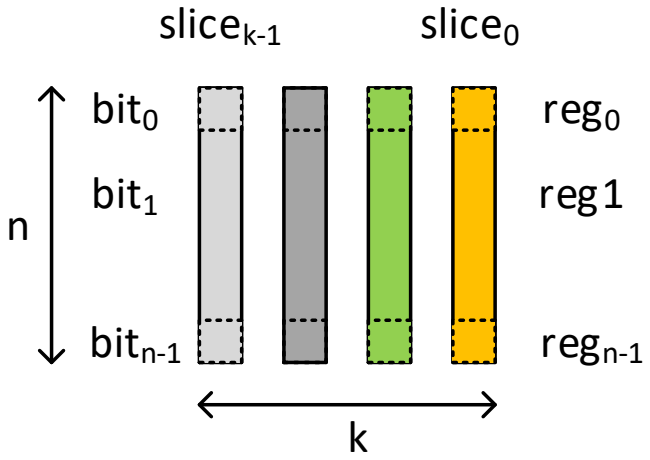
- **Power-based side-channel leakage across the ISA**
- **Secret Sharing as a Countermeasure - SKIVA**
- **Automatic Bitslice Design**
- **DOM-based Instruction-set Design for RISCV**
- **Open Challenges**
- **Further Reading**

Bitslicing as a mechanism to isolate bits

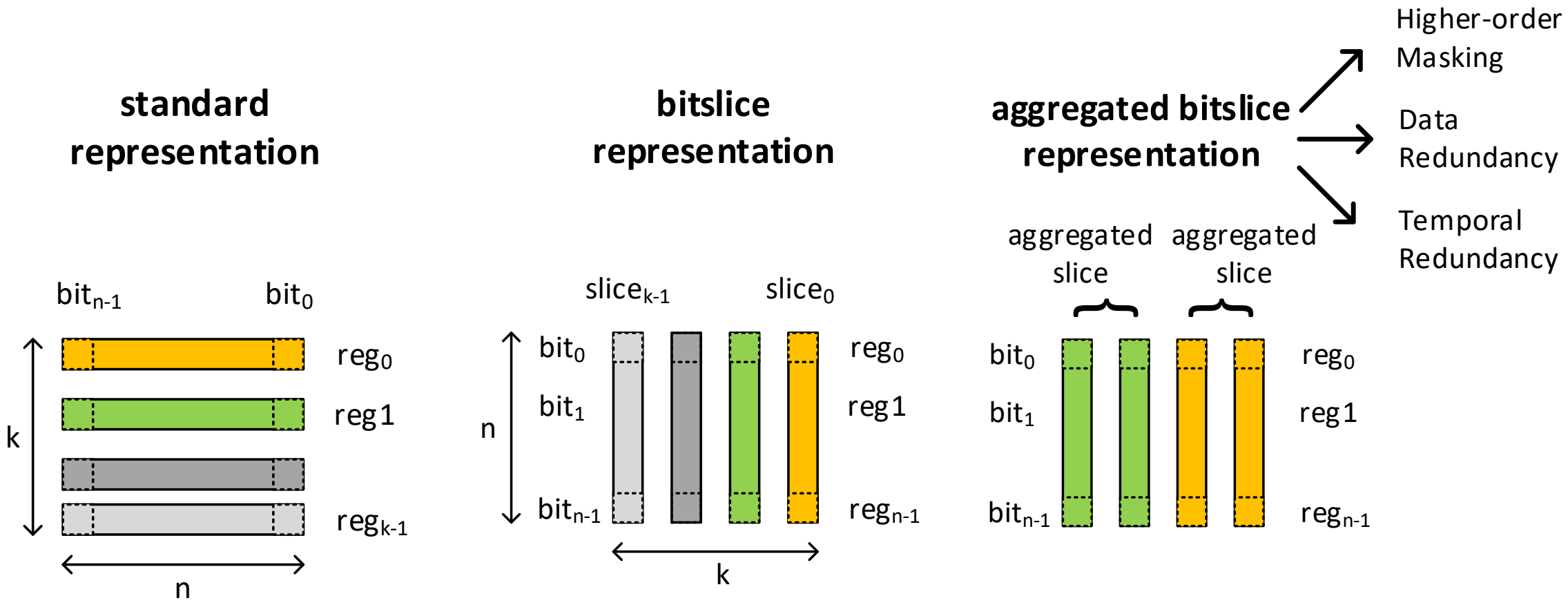
standard representation



bitslice representation

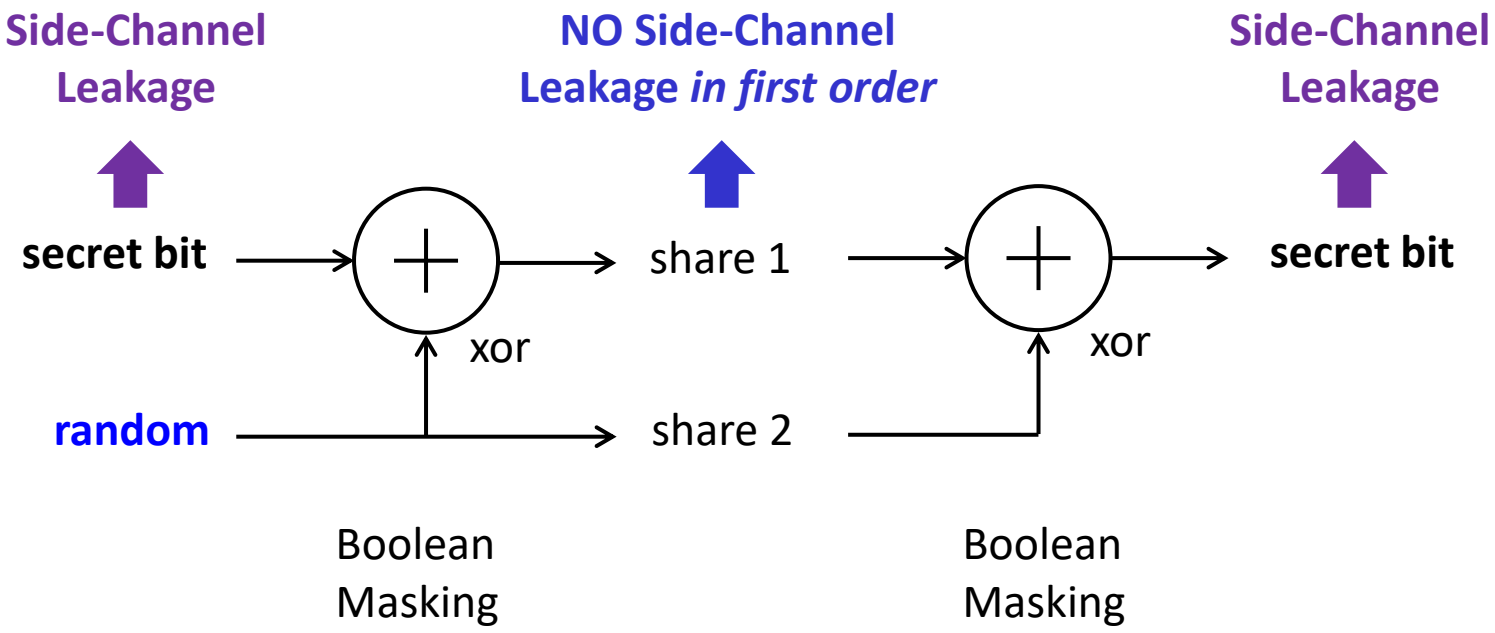


Bitslicing as a mechanism to isolate bits



https://github.com/Secure-Embedded-Systems/Skiva

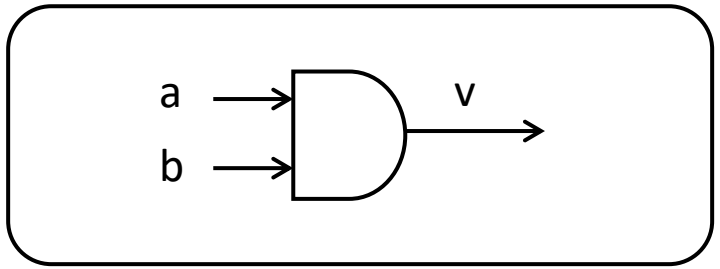
Boolean Masking



Boolean Masked Software

Hardware Description of Crypto (RTL)

Logic Synthesis

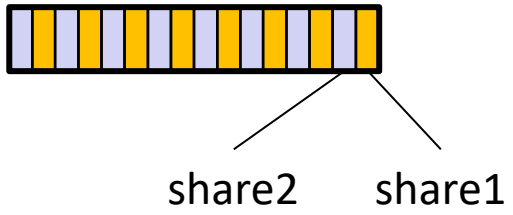


Netlist

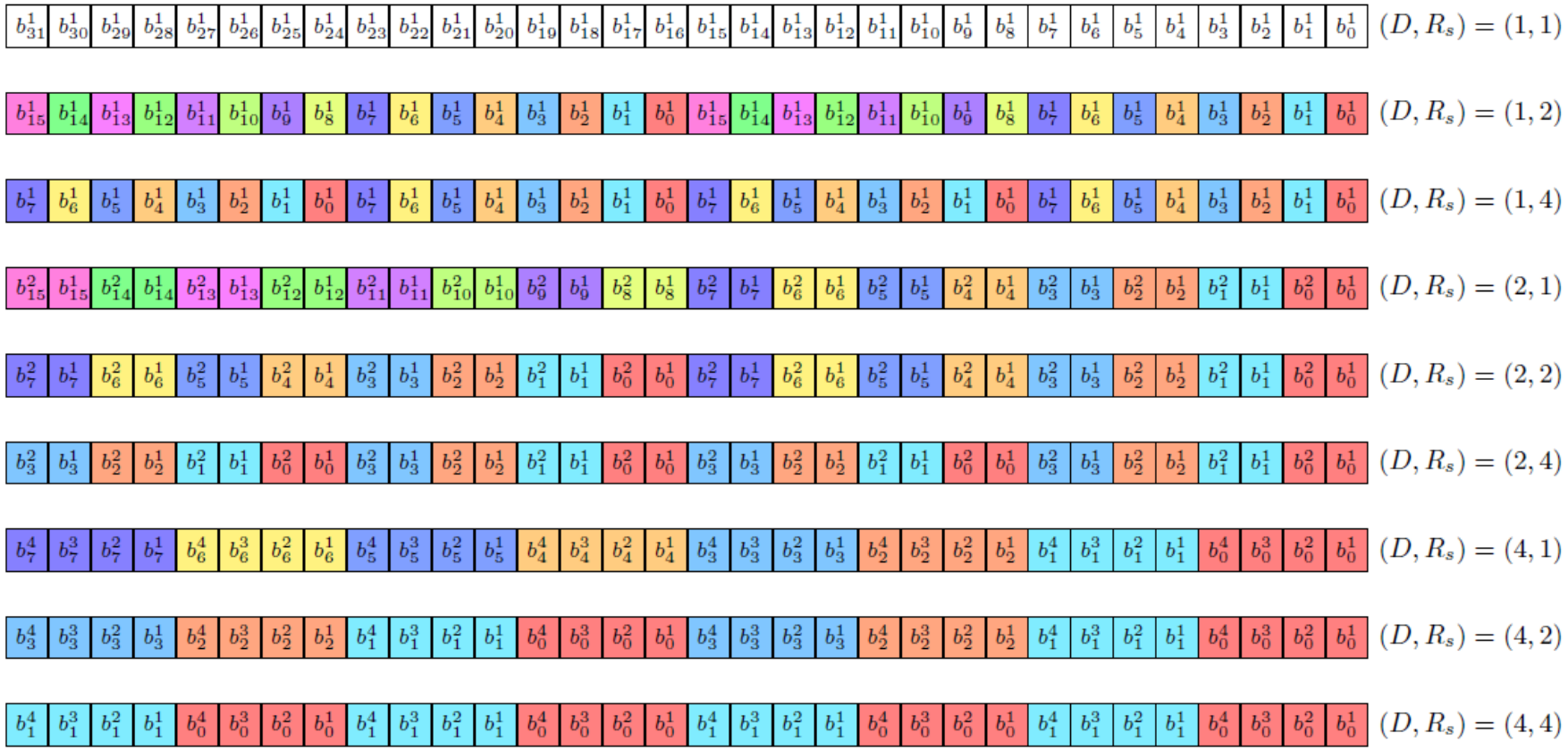
Leveling + Boolean Masking Mapping

```
and    %r1, %r2, %r6
subrot %r2, 2, %l0
and    %r1, %l0, %r5
xor    %rand, %r6, %r6
xor    %r6, %r5, %r6
```

Parallel Masked program



SKIVA: Flexible Redundant Bitslices

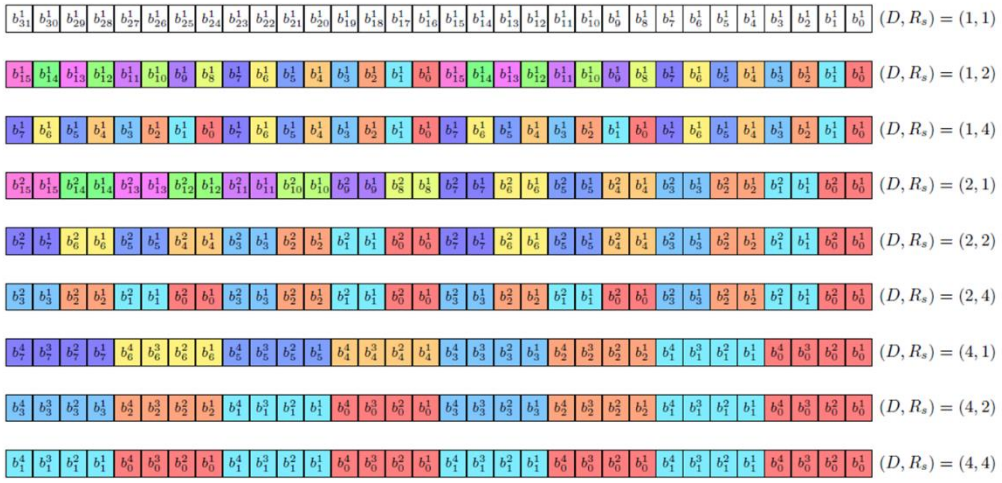


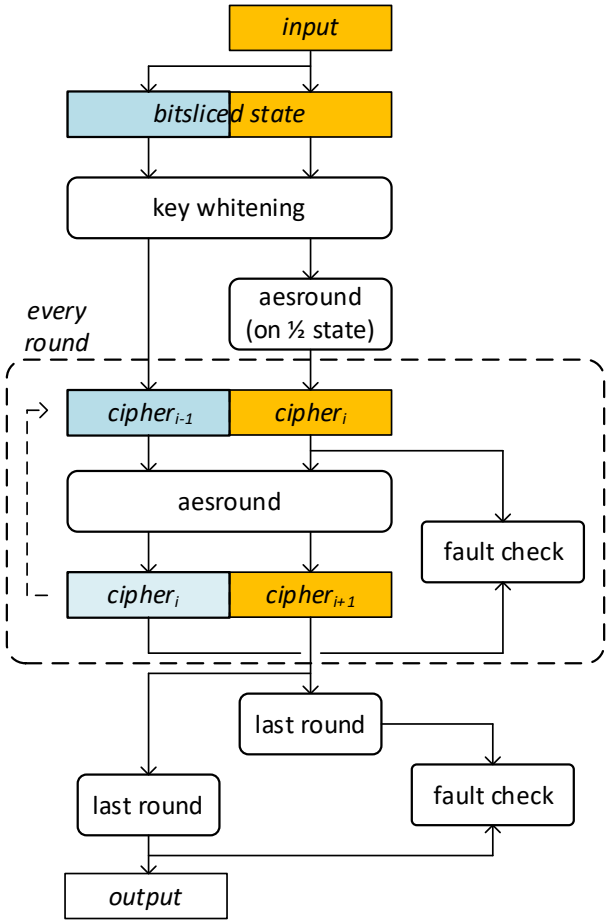
(D, R_s)

Number of shares (Side Channel Leakage)

Level of Spatial Redundancy (Data faults)

SKIVA: Flexible Redundant Bitslices





+ R_t

Temporal Redundancy
(Control faults)

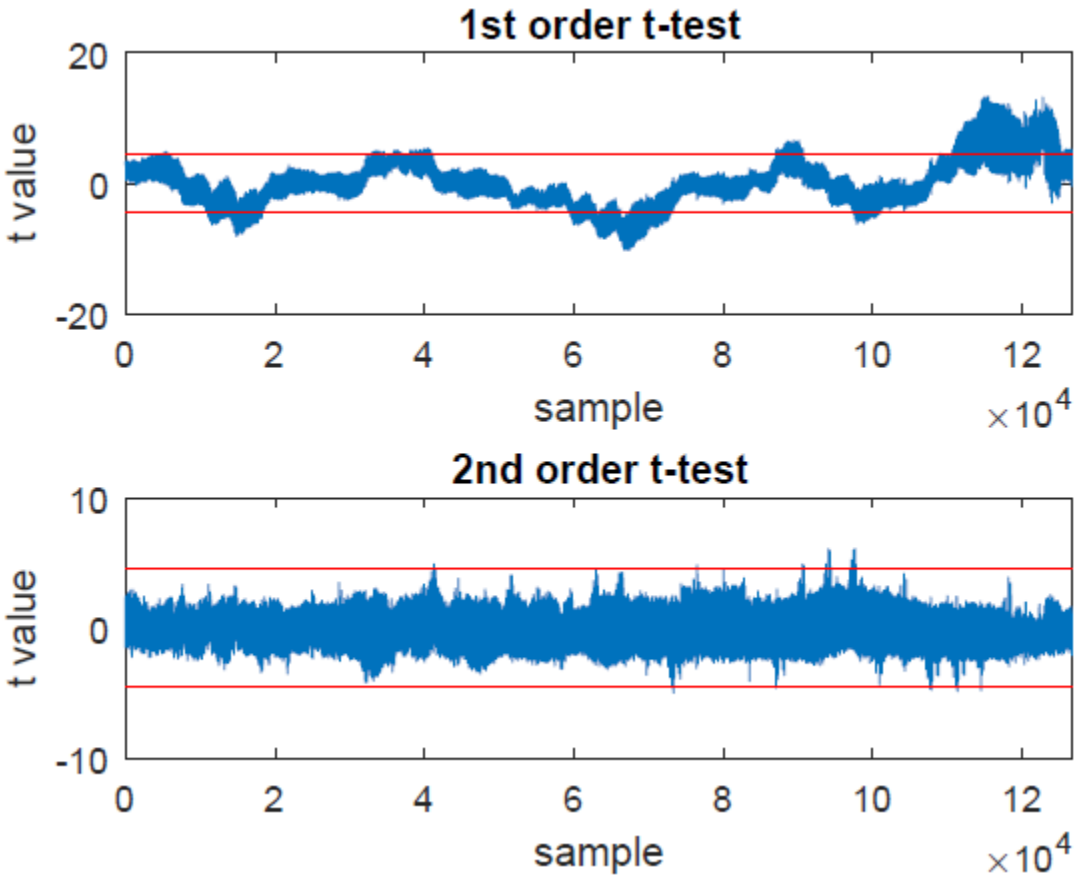
SKIVA Custom Instructions

- **Bitslice Transformation**
- **Higher-order Masking**
- **Redundant & Fault-correcting Computation**
- **Fault Checking**

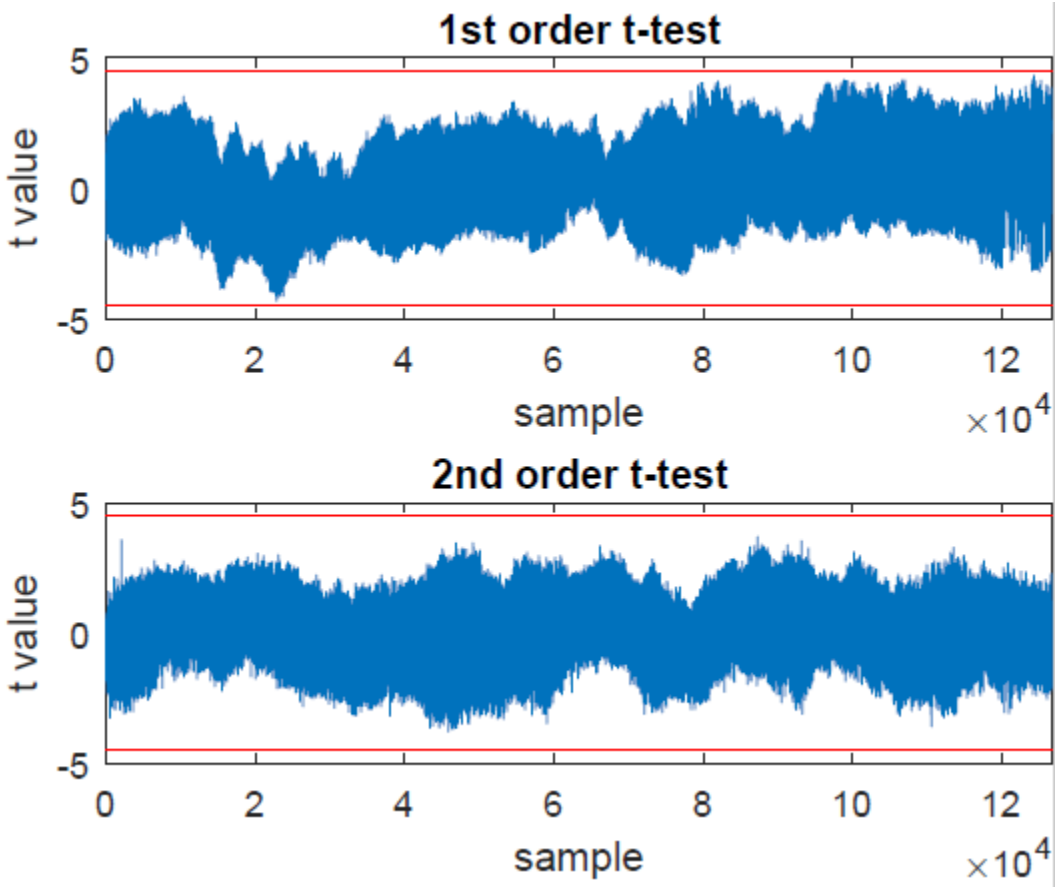
AES Cycles/Byte on SKIVA
(LEON-3/Cyclone IV Integration)

(R_t = 1)	D = 1	D = 2	D = 3
R_s = 1	44	176	579
R_s = 1	89	413	1298
R_s = 1	169	819	2593

SKIVA Side-channel Leakage Assessment



(D,R_s) = (4,1), 35K+35K Traces, PRNG off



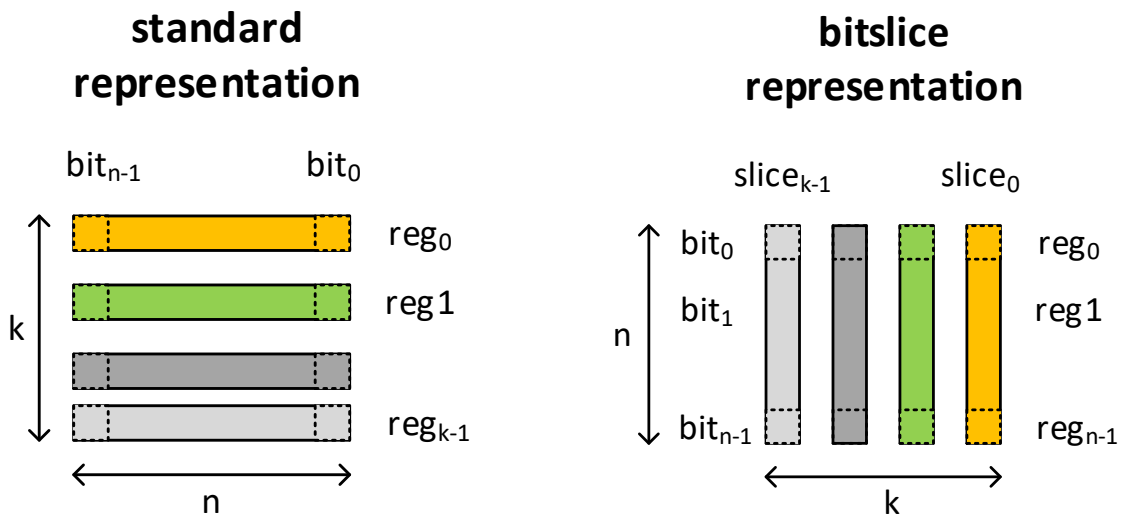
(D,R_s) = (4,1), 500K+500K Traces, PRNG on

Roadmap

- **Power-based side-channel leakage across the ISA**
- **Secret Sharing as a Countermeasure - SKIVA**
- **Automatic Bitslice Design**
- **DOM-based Instruction-set Design for RISCV**
- **Further Reading**

Background: Bitslicing

- Uncertainty in timing
  - Data-dependent process (eg. Loop)
  - Memory hierarchy *out of programmer's control*
  - Resource contention *out of programmer's control*

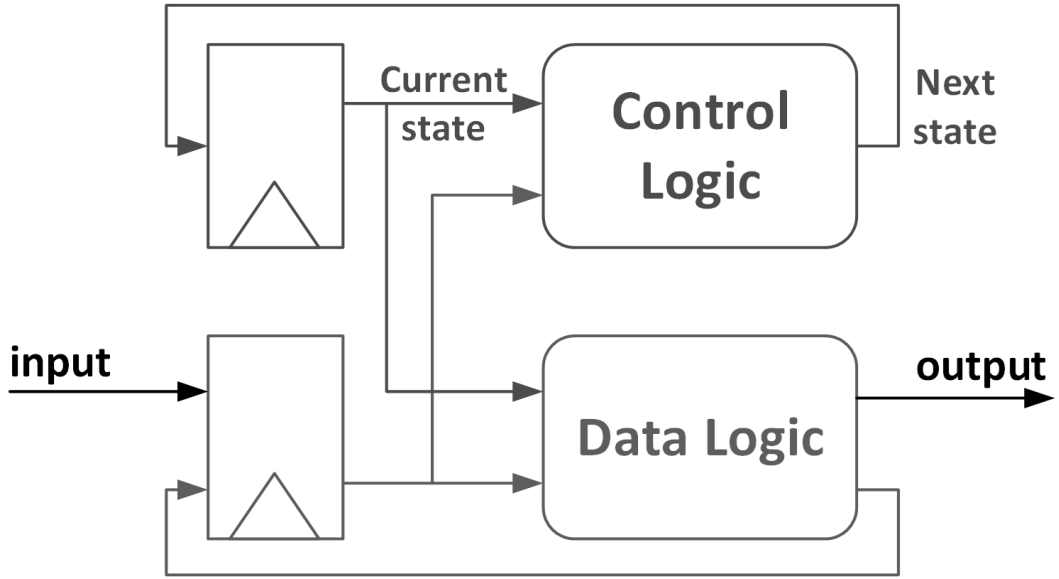


- Full utilization of processor word
- Flat
- Hand-written / special tools and language

Background: Synchronous Program and FSMD

```
while true do
    wait for clock_tick;
    outputs = eval(inputs, current_state);
    next_state = update(inputs, current_state);
    current_state = next_state;
end
```

Common in real-time applications
Repeatable timing



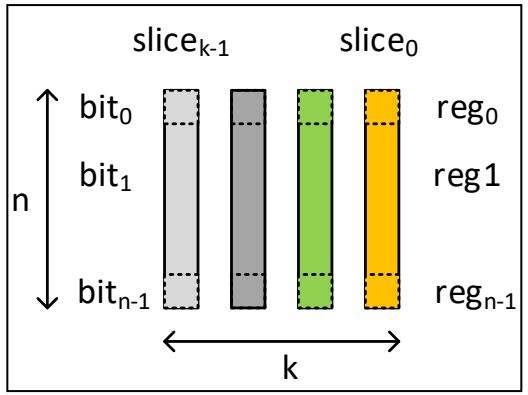
Common in digital design
Control and data

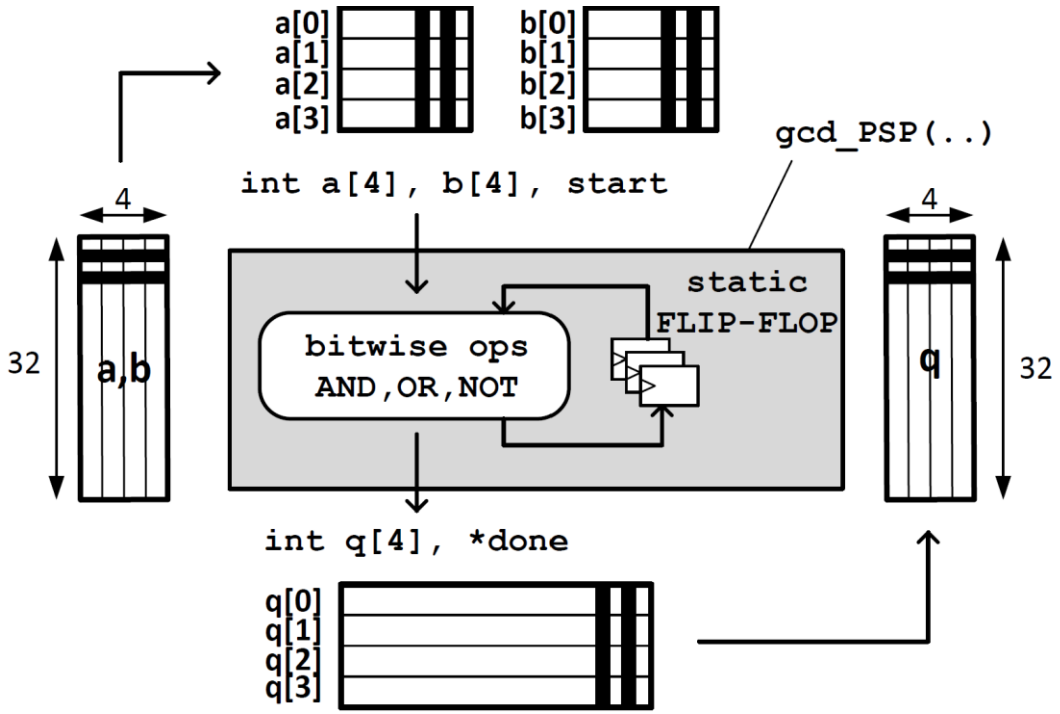
Parallel Synchronous Program (PSP)

- Repeatable
  - Real-time applications
- Data-independent
  - Secure programs

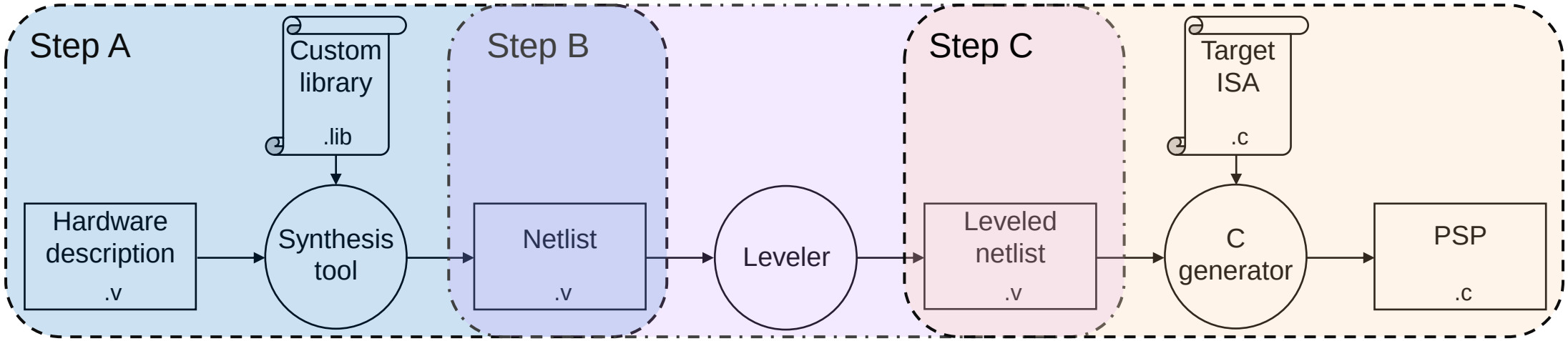
```
while true do
    wait for clock_tick;
    outputs = eval(inputs, current_state);
    next_state = update(inputs, current_state);
    current_state = next_state;
end
```

+

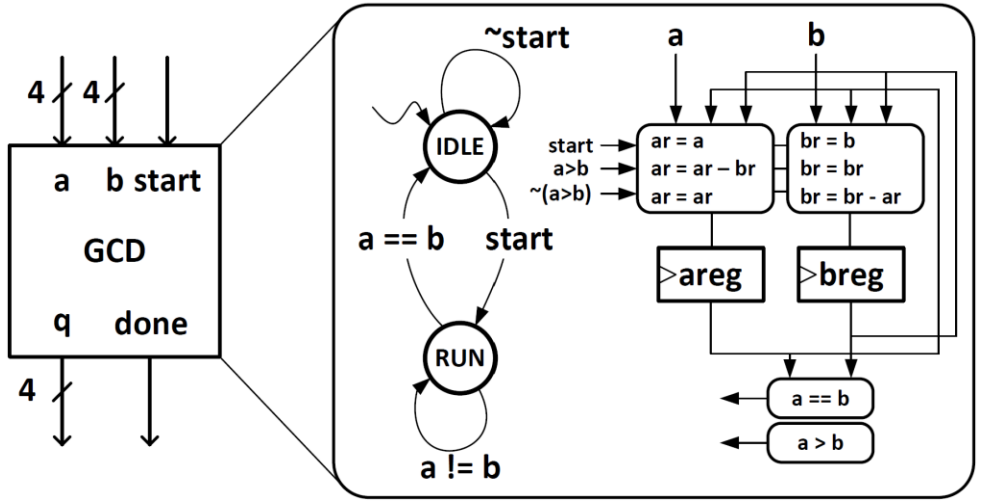


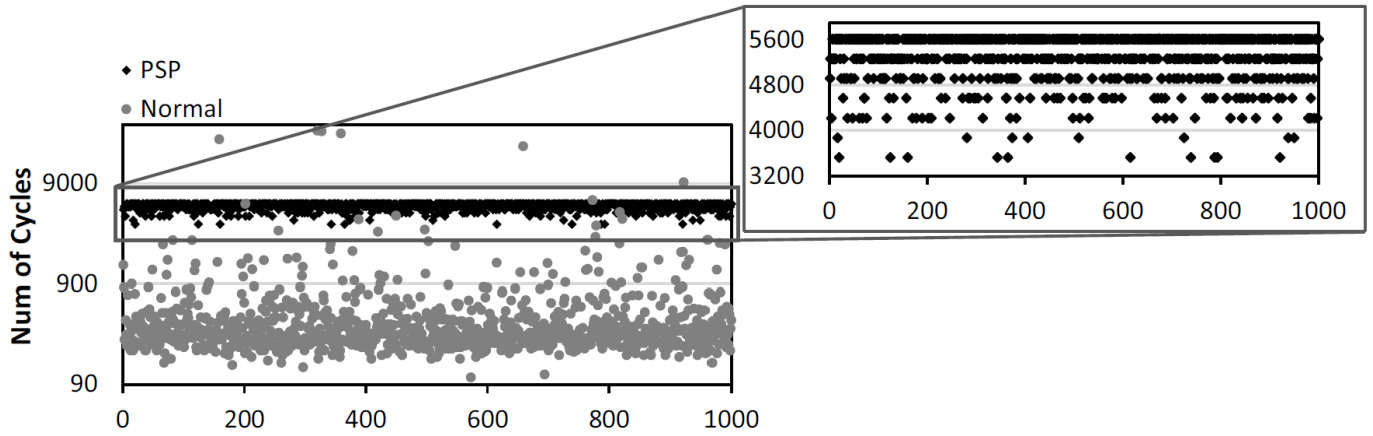


Automated Synthesis of PSP



Experimental Evaluation



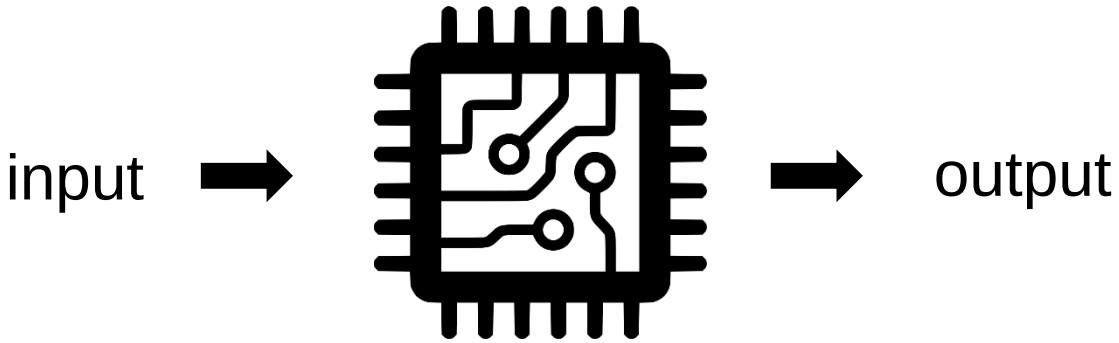


Runtime of normal and PSP implementations of the GCD algorithm on 1000 random inputs

Roadmap

- **Power-based side-channel leakage across the ISA**
- **Secret Sharing as a Countermeasure - SKIVA**
- **Automatic Bitslice Design**
- **DOM-based Instruction-set Design for RISCV**
- **Further Reading**

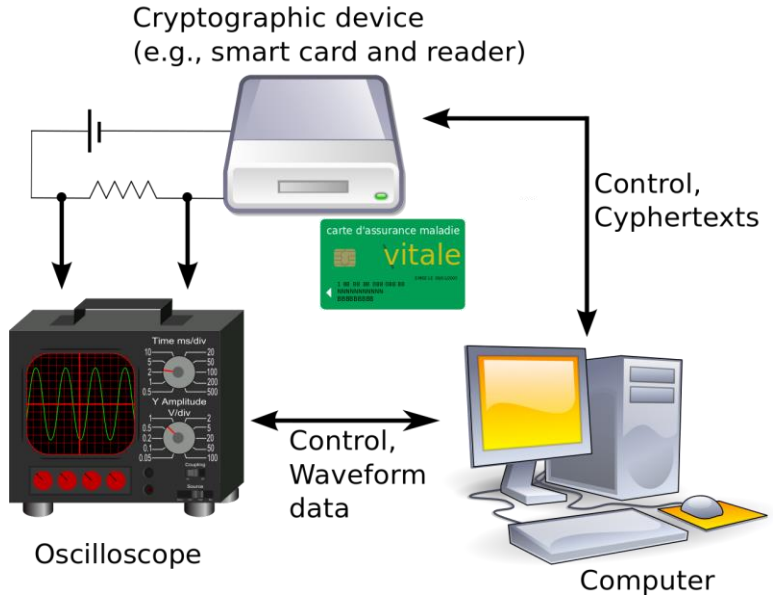
Side-Channel Analysis



Power
Timing
EM emanation

Power Side-Channel Attack

Data processed contributes to the power



Source: Wikipedia

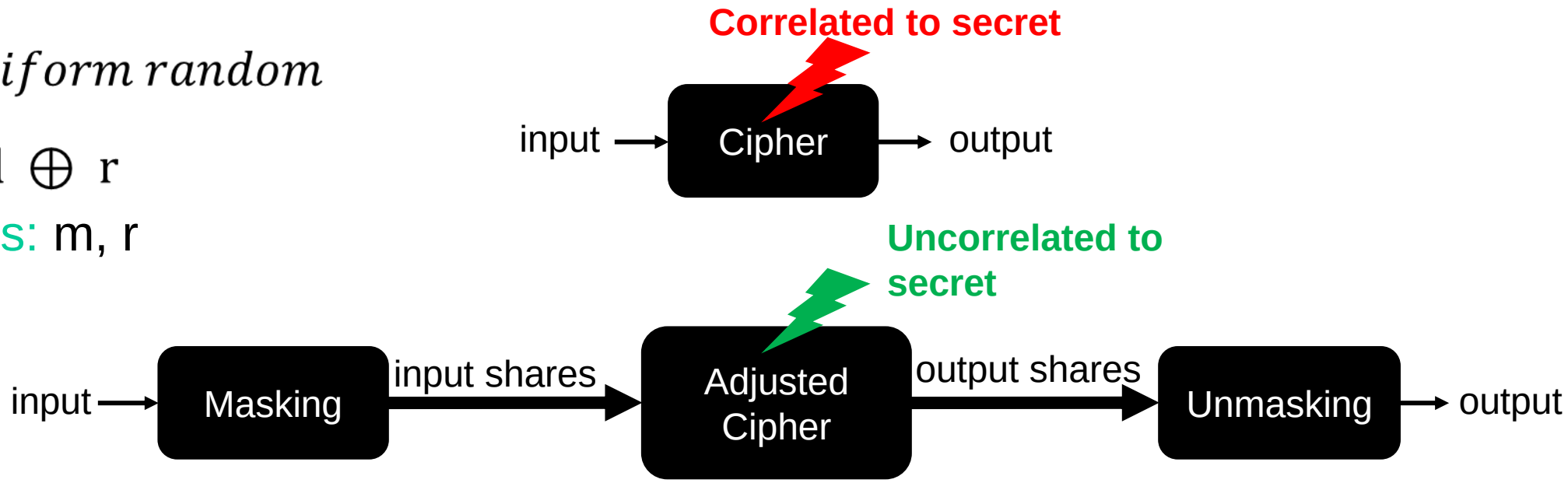
Power SCA Countermeasure masking

d: data to mask

$r \sim uniform\ random$

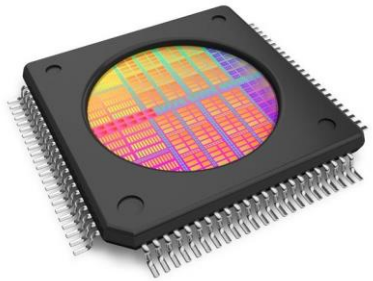
$m = d \oplus r$

Shares: m, r

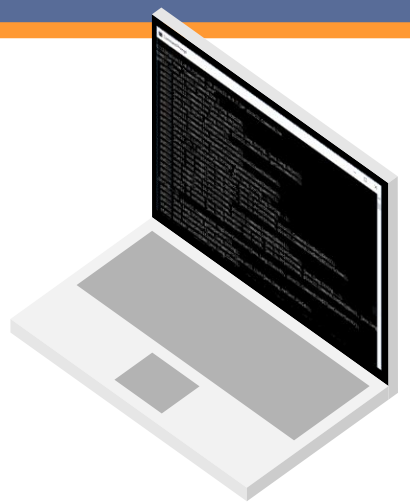


Masking Implementation

- In hardware
  - Threshold implementation (TI)
  - Domain-oriented masking
- In software
  - Masking
    - Additional tweaks
    - Skiva [SKIVA]



Source: regmedia.co.uk



Register allocation

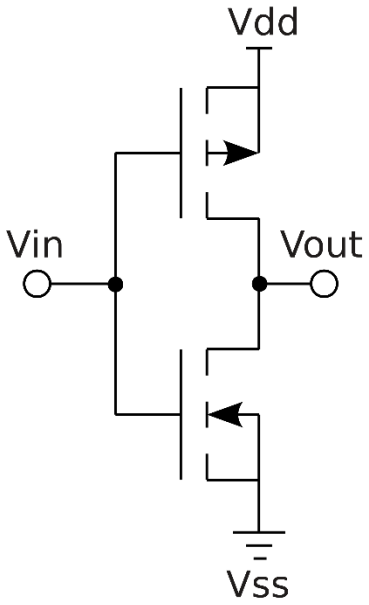
```
# two-share NINA Secure Multiplication
# input:  %i2 (a),
#         %i3 (b),
#         %i4 (random),
#         %i5 (fault flags)
# output: %o0 (a & b)
#         %i5 (accumulated fault)
# step 1: clear input in case of a fault
NOT     %i5, %o4        #
AND     %i2, %o4, %o6   #
# step 2: calculate AND result
AND     %i3, %o6, %o5   # partial product 1
SUBROT %o6,    2, %l0   # share-rotate
AND     %i3, %l0, %o3   # partial product 2
XOR     %l0, %l0, %l0   # clear SUBROT output
XOR     %o5, %i4, %o2   # random + parprod 1
XOR     %o2, %o3, %o1   #        + parprod 2
# step 3: refresh the output
SUBROT %i4,    2, %l1   # parallel refresh
XOR     %o1, %l1, %o0   #           output
# step 4: update fault flags
FTCHK   %o0, imm, %g5   # imm depends on Rs and Rt
OR      %g5, %i5, %i5   #
```

Actual calculation

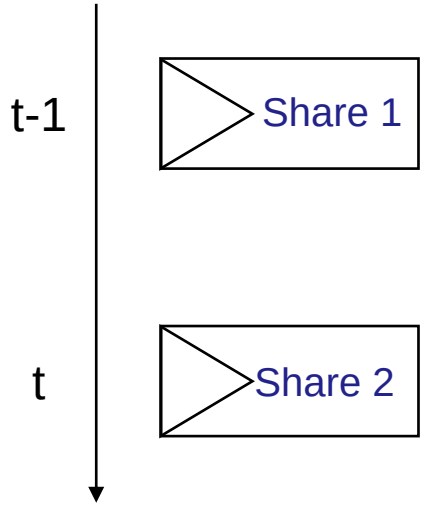
Additional steps

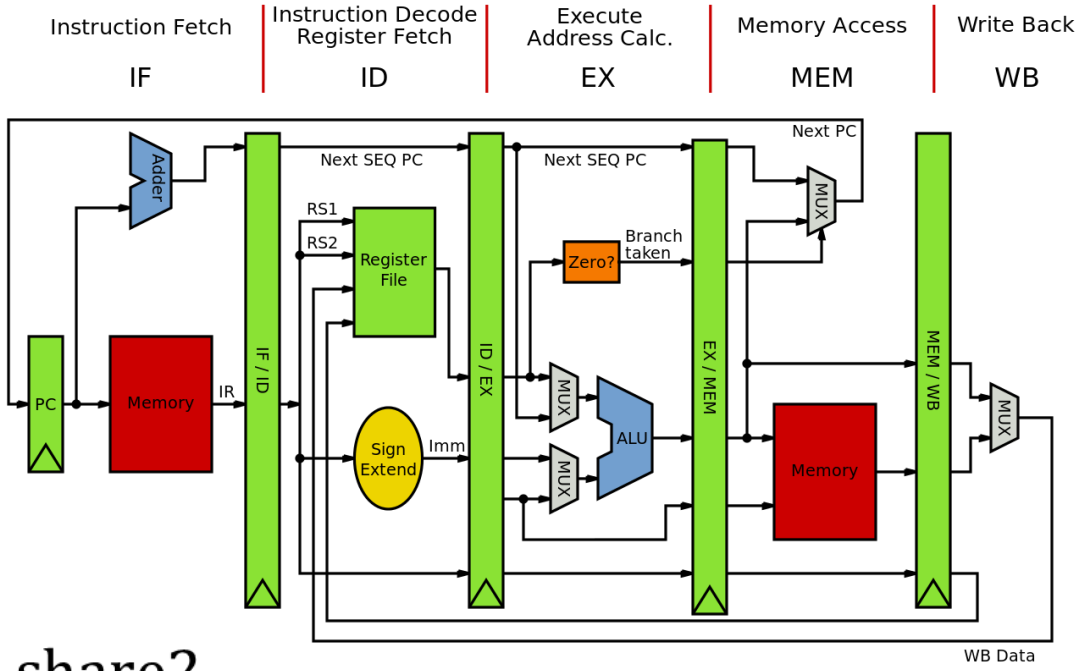
[SKIVA] P. Kiaei, D. Mercadier, PE. Dagand, K. Heydemann, and P. Schaumont “Custom Instruction Support for Modular Defense against Side-channel and Fault Attacks”, COSADE 2020.

Implementation Consideration



Source: Wikipedia



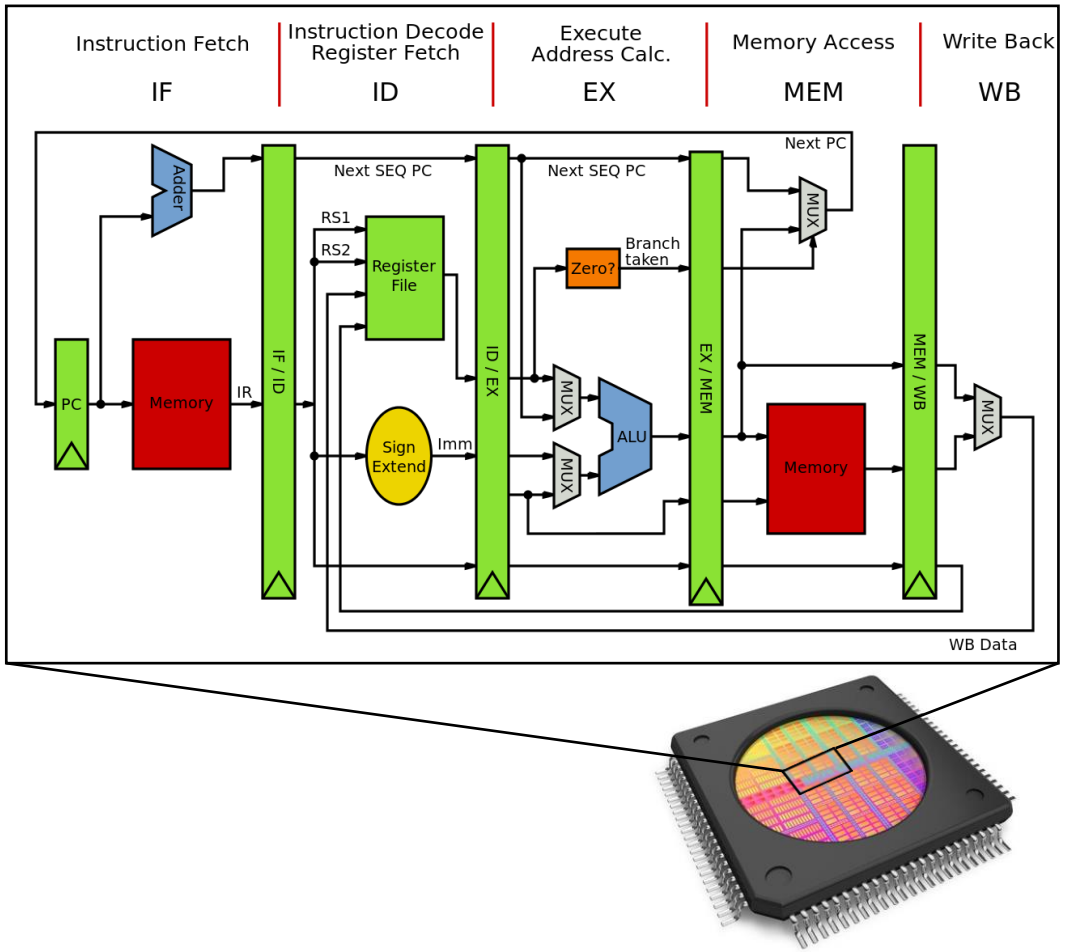


Power consumption ~ share1 $\oplus$ share2

Power consumption ~ data

Hybrid Approach

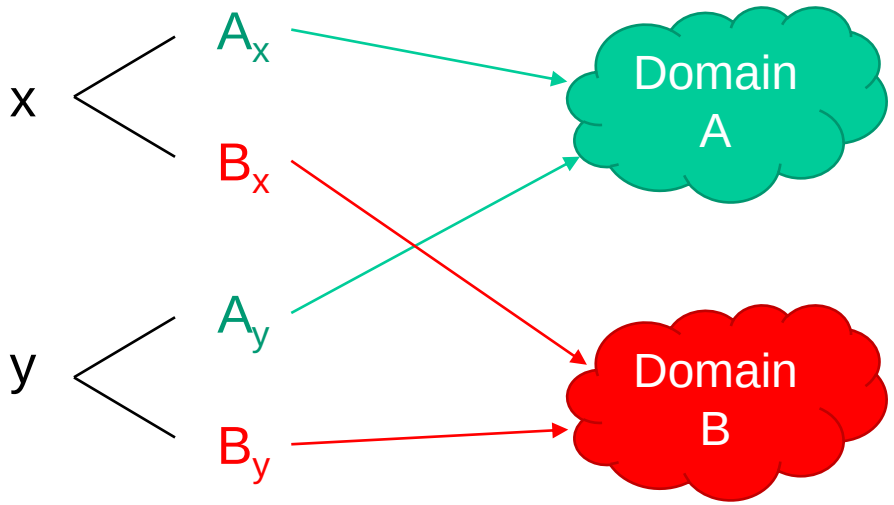
- Goal: protect the software
- Apply countermeasures to the processor hardware
  - Only the pipeline
- Related work:
  - De Mulder et al. [TIRISCV]: TI



[TIRISCV] De Mulder, Elke, Samatha Gummalla, and Michael Hutter. “Protecting RISC-V against side-channel attacks.” *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019.

Domain-Oriented Masking [DOM]

- To each share, its own domain!



L: linear

$$A_L = f(A_x , A_y)$$
$$B_L = f(B_x , B_y)$$

NL: non-linear

$$A_{NL} = f(A_x , B_x , A_y , B_y)$$
$$B_{NL} = f(A_x , B_x , A_y , B_y)$$

resharing

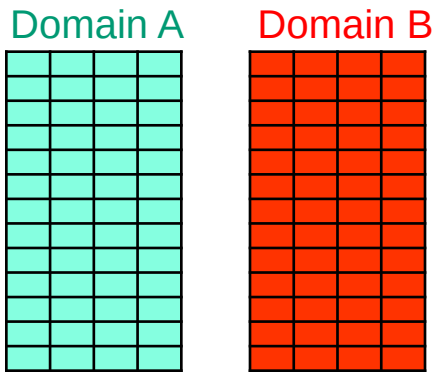
[DOM] Groß, Hannes, Stefan Mangard, and Thomas Korak. "Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order." *TIS@ CCS*. 2016.

DOM in Pipeline Stages

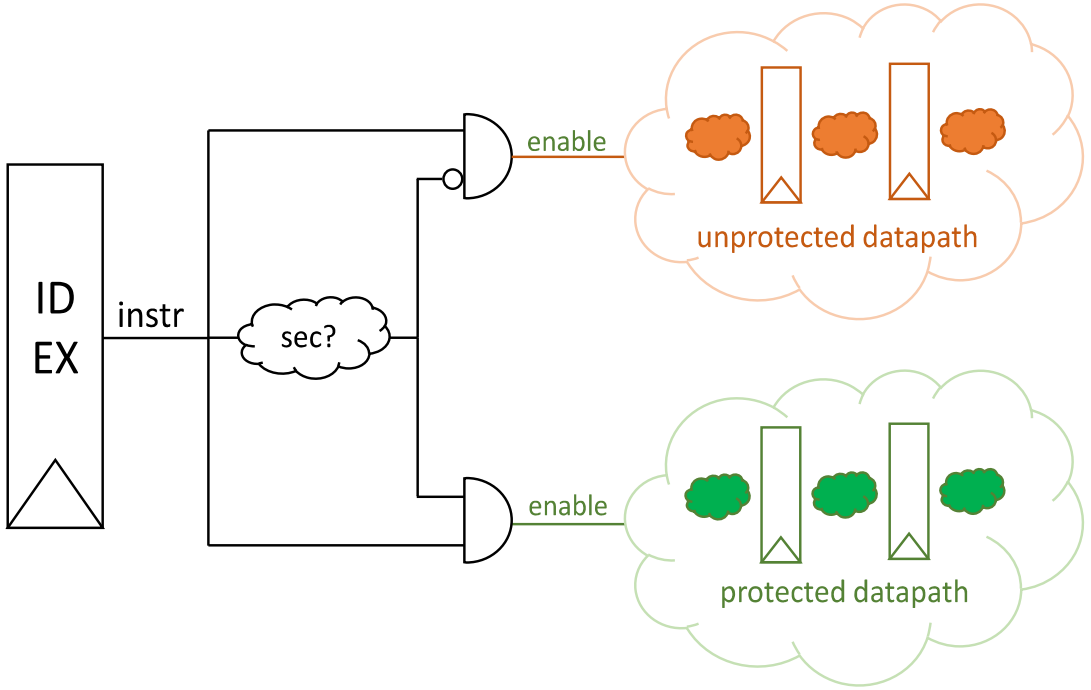
Design principles:

1. Separating the secure and the unprotected parts of the processor
2. Protecting the secure part

   Applying DOM with 2 shares



Duplicated memory elements



DOM ISA

Universal set of instructions:

Linear:

Not: $q = \sim x$ $A_q = \sim A_x$ $B_q = B_x$ `dom.not rd, rs1, rs2`

Xor: $q = x \oplus y$ $A_q = A_x \oplus A_y$ $B_q = B_x \oplus B_y$ `dom.xor rd, rs1, rs2`

Non-linear:

And: $q = x \,.\, y$

`dom.and rd, rs1, rs2`

instruction	$x \cdot y$	
domain	$\mathcal{A}$	$\mathcal{B}$
cycle 1 (Z_0 req'd)	$A_{t1} = A_x \cdot A_y$ $A_{t2} = B_y \oplus Z_0$ $A_{t3} = A_x \cdot Z_0$	$B_{t1} = B_x \cdot B_y$ $B_{t2} = A_y \oplus Z_0$ $B_{t3} = B_x \cdot Z_0$
cycle 2 (Z_1 req'd)	$A_q = A_{t1} \oplus A_x \cdot A_{t2} \oplus A_{t3} \oplus Z_1$	$B_q = B_{t1} \oplus B_x \cdot B_{t2} \oplus B_{t3} \oplus Z_1$

Or: $q = x + y$

`dom.or rd, rs1, rs2`

instruction	$x + y$	
domain	$\mathcal{A}$	$\mathcal{B}$
cycle 1 (Z_0 req'd)	$A_{t1} = A_x \cdot A_y$ $A_{t2} = B_y \oplus Z_0$ $A_{t3} = A_x \cdot Z_0$	$B_{t1} = B_x \cdot B_y$ $B_{t2} = A_y \oplus Z_0$ $B_{t3} = B_x \cdot Z_0$
cycle 2 (Z_1 req'd)	$A_q = A_x \oplus A_y \oplus A_{t1} \oplus A_x \cdot A_{t2} \oplus A_{t3} \oplus Z_1$	$B_q = B_x \oplus B_y \oplus B_{t1} \oplus B_x \cdot B_{t2} \oplus B_{t3} \oplus Z_1$

Special instruction: one-bit *add*

Sum: $S = x \oplus y \oplus \boxed{c_i}$ Carry special register

Carry-out: $\boxed{C_o} = (x \oplus y).c_i + x.y$

```
dom.add rd, rs1, rs2
```

instruction	$(x \oplus y) \cdot c_i + x \cdot y$	
domain	$\mathcal{A}$	$\mathcal{B}$
cycle 1 (Z_0, Z_2 req'd)	$A_{t1} = A_x \oplus A_y$ $A_{t2} = B_{c_i} \oplus Z_0$ $A_{t3} = (A_x \oplus A_y) \cdot Z_0$ $A_{t4} = B_y \oplus Z_2$ $A_{t5} = A_x \cdot Z_2$	$B_{t1} = B_x \oplus B_y$ $B_{t2} = A_{c_i} \oplus Z_0$ $B_{t3} = (B_x \oplus B_y) \cdot Z_0$ $B_{t4} = A_y \oplus Z_2$ $B_{t5} = B_x \cdot Z_2$
cycle 2 (Z_1, Z_3 req'd)	$A_a = A_{t1} \cdot A_{c_i} \oplus A_{t1} \cdot A_{t2} \oplus A_{t3} \oplus Z_1$ $A_b = A_x \cdot A_y \oplus A_x \cdot A_{t4} \oplus A_{t5} \oplus Z_3$	$B_a = B_{t1} \cdot B_{c_i} \oplus B_{t1} \cdot B_{t2} \oplus B_{t3} \oplus Z_1$ $B_b = B_x \cdot B_y \oplus B_x \cdot B_{t4} \oplus B_{t5} \oplus Z_3$
cycle 3 (Z_4 req'd)	$A_{t6} = B_b \oplus Z_4$ $A_{t7} = A_a \cdot Z_4$	$B_{t6} = A_b \oplus Z_4$ $B_{t7} = B_a \cdot Z_4$
cycle 4 (Z_5 req'd)	$A_{C_o} = A_a \oplus A_b \oplus A_a \cdot A_b \oplus A_a \cdot A_{t6} \oplus A_{t7} \oplus Z_5$	$B_{C_o} = B_a \oplus B_b \oplus B_a \cdot B_b \oplus B_a \cdot B_{t6} \oplus B_{t7} \oplus Z_5$

Opcode Mapping and Mnemonics

inst[4:2] / inst[6:5]	000	001	010	011	100	101	110	111 (> 32b)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	48b
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	64b
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	48b
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	≥ 80b

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

Funct7	Funct3	Opcode	Mnemonic
0000000	000	0001011	`dom.not`
0000000	001	0001011	`dom.xor`
0000000	010	0001011	`dom.and`
0000000	011	0001011	`dom.or`
0000001	000	0001011	`dom.add`

Further Reading

SKIVA

P. Kiaei, D. Mercadier, P. E. Dagand, K. Heydemann, P. Schaumont, "Custom Instruction Support for Modular Defense against Side-channel and Fault Attacks," 11th International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE 2020), Lugano, Switzerland, April 2020. IACR ePrint architve 2020/466.

https://github.com/Secure-Embedded-Systems/Skiva

Automatic Bitslicing

P. Kiaei and P. Schaumont, "Synthesis of Parallel Synchronous Software," in IEEE Embedded Systems Letters, doi: 10.1109/LES.2020.2992051.

https://github.com/Secure-Embedded-Systems/psp-esl2020

DOM Instruction set

P. Kiaei, P. Schaumont, "Domain-Oriented Masked Instruction Set Architecture for RISC-V," Workshop on Secure RISC-V Architecture Design (SECRISCV), August 2020. Also as IACR ePrint archive 2020/465 (preprint).