

# ???

# Representations of Numbers and Electrical Activity

Arnaud TISSERAND

CNRS, Lab-STICC

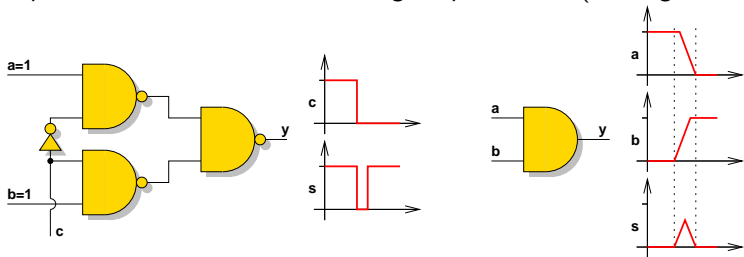
NAC 2024 Paris

## Topics:

- ▶ Computer arithmetic: **representations of numbers** and algorithms
- ▶ **Crypto**: asymmetric (RSA, (H)ECC, lattice based PQC), hash functions, symmetric ciphers, homomorphic encryption
- ▶ **Observation** and perturbation attacks
- ▶ Secure implementations:
  - ▶ **hardware**: accelerators and secure processors (ASIC, FPGA)
  - ▶ **software**: microcontrollers, embedded processors, high-end multicores

Power consumption:

- ▶ Static power due to leakage(s)
- ▶ **Dynamic** power due to transitions
  - ▶ **useful/logical** transitions due to state switching ( $0 \rightarrow 1$  and  $1 \rightarrow 0$ )
  - ▶ parasitic transitions due to timings imperfections (skew, glitches, ...)



In this talk, we only deal with **logical activity**

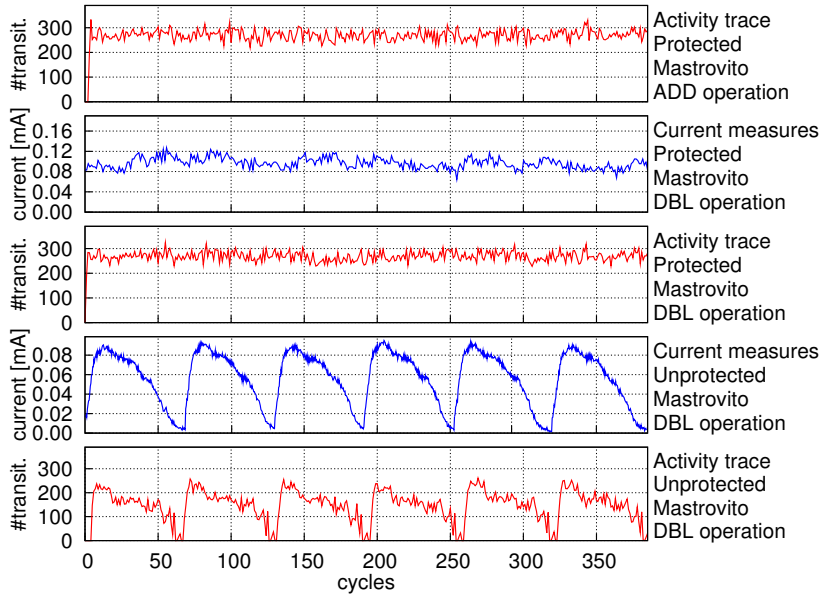
General principle:

1. Measure/observe **external physical parameter(s)** on a running device
2. Deduce **internal (secret) informations**

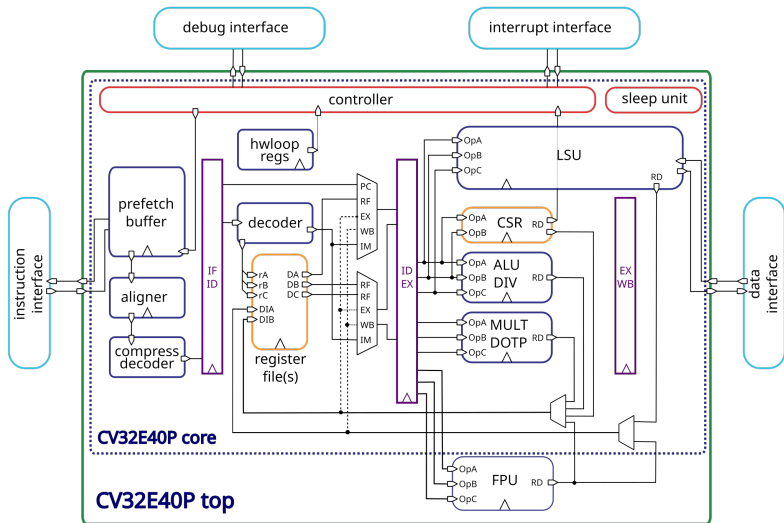
Examples:

- ▶ Timings
- ▶ Power consumption
- ▶ Electromagnetic radiation
- ▶ Temperature
- ▶ Number of cache misses
- ▶ ...

Attacks are always improving (strong statistics, deep learning, ...)

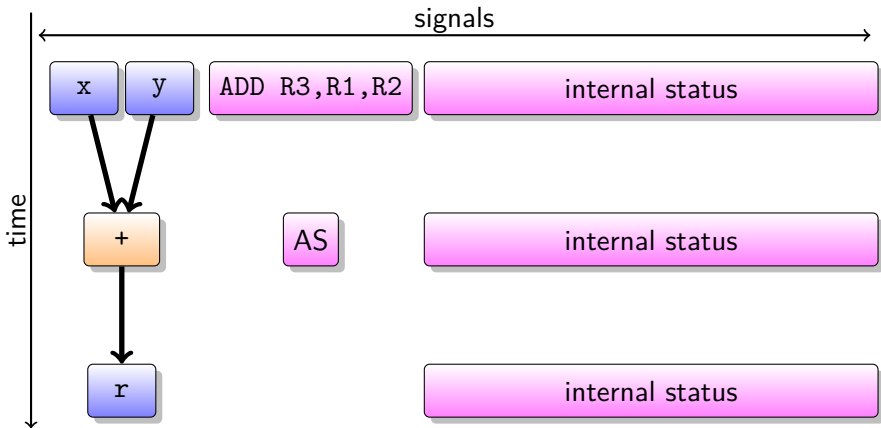


## RISC-V CV32E40P core from OpenHW Group (documentation)



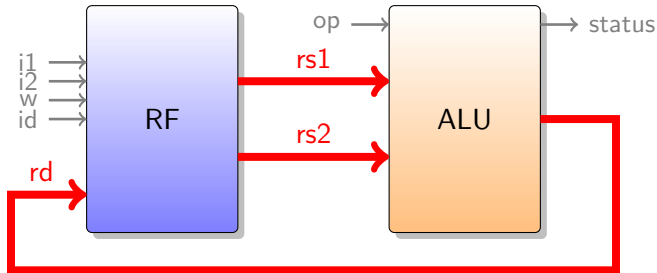
HDL code(s), complete software toolchain, numerous libraries and works

Operation to be executed:  $r \leftarrow x + y$



- ▶ AS: ALU status and internal pipeline
- ▶ Internal status: pipeline management, bypasses, memory hierarchy, branch predictor, monitoring, etc

- ▶ Very (over?) simplified 32-bit processor
- ▶ Register file (RF): 32 registers, 2 read ports (*rs1*, *rs2*) and 1 write port (*rd*) active at each instruction
- ▶ Arithmetic and logic unit (ALU)
- ▶ Basic instruction set (e.g.,  $R1 \leftarrow R2 + R3$ )
- ▶ Simulation: only logical transitions (no glitch), 1-cycle instructions
- ▶ Only (*rs1*, *rs2*, *rd*) are **observable** (not other signals!)
- ▶ Start with random data in registers (crypto context)





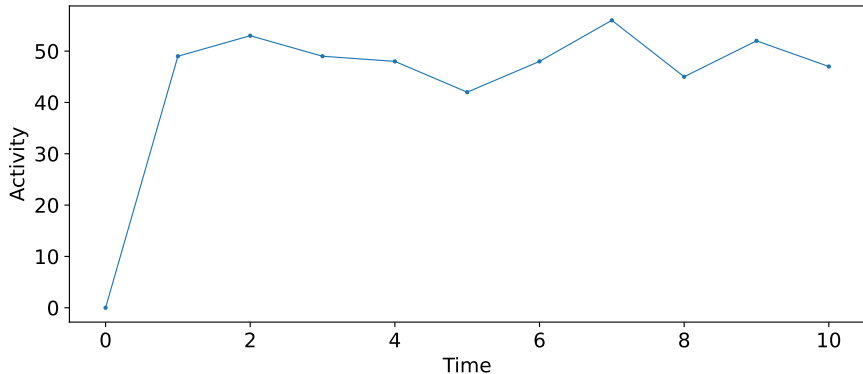
NOP

R0  $\leftarrow$  R1 + R2 // all sources and destinations

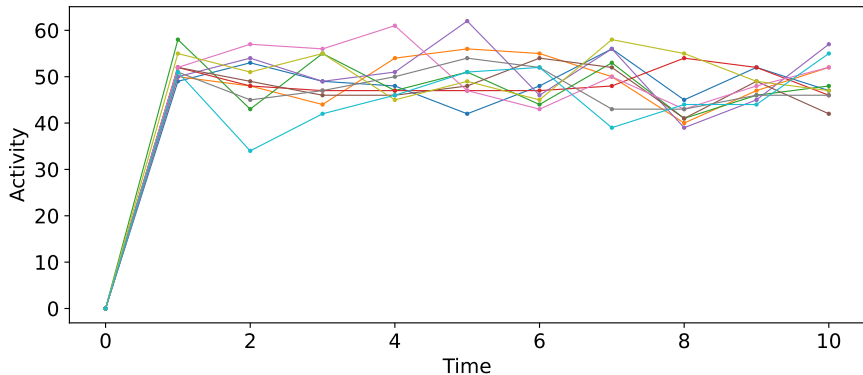
R3  $\leftarrow$  R4 + R5 // are different and random

R6  $\leftarrow$  R7 + R8

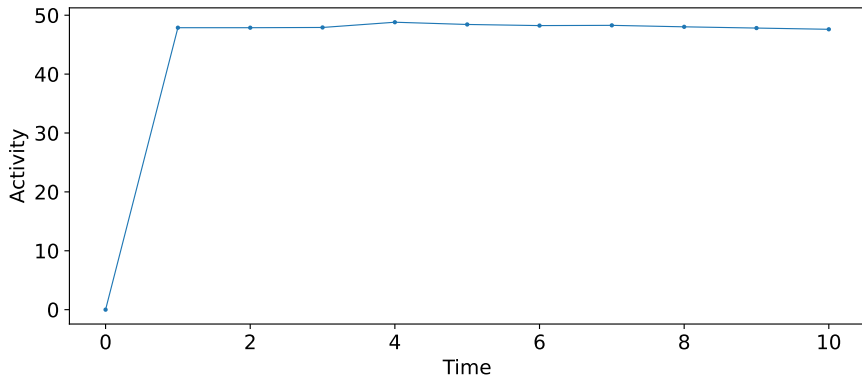
...



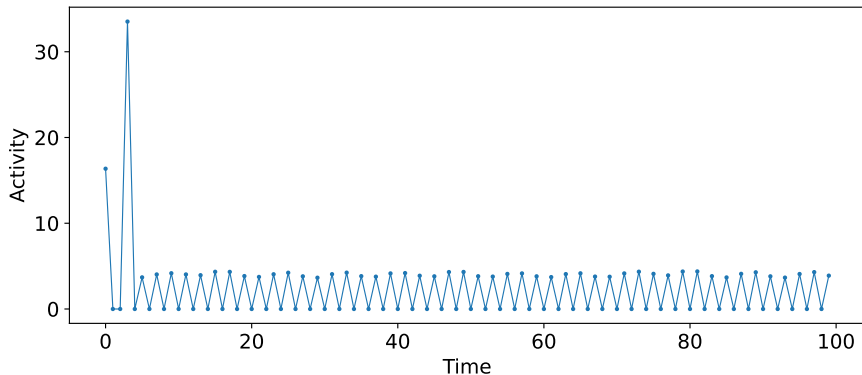
Traces for 10 sets of initial (random) values in the registers:



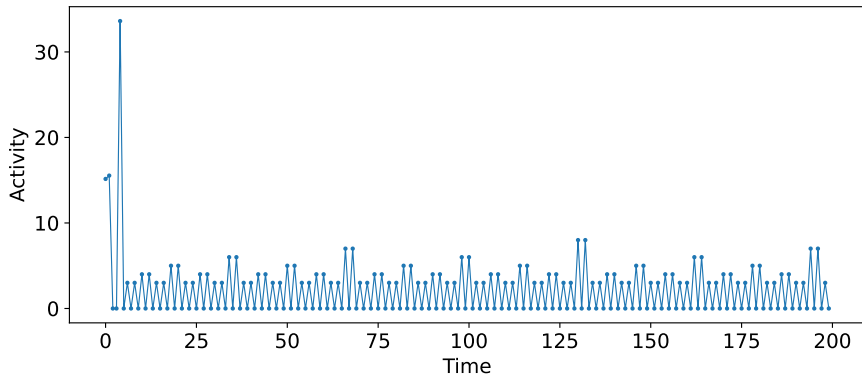
Average trace:



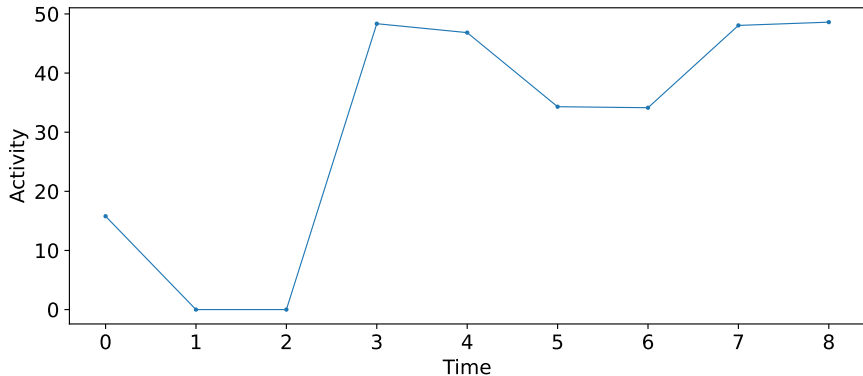
```
R1 <- 1           // +1 for loop index incr.  
NOP  
NOP  
loop: R2 <- R2 + R1 // incr. (R2 random)  
      JMP loop
```



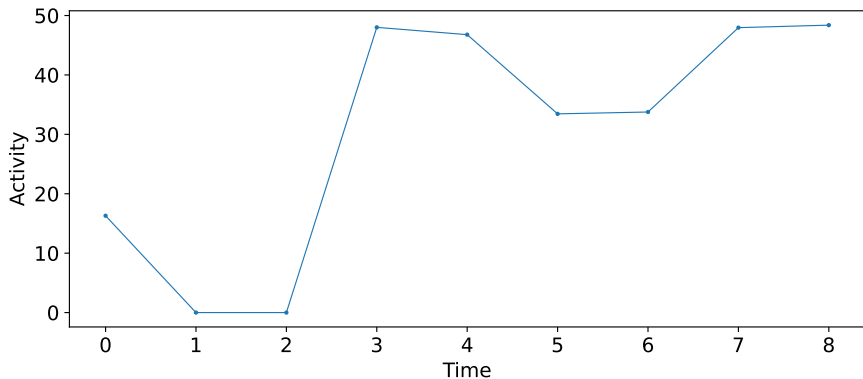
```
R1 <- 1           // +1 for loop index incr.  
R2 <- 0           // reset acc.  
NOP  
NOP  
loop: R2 <- R2 + R1 // incr.  
      JMP loop
```



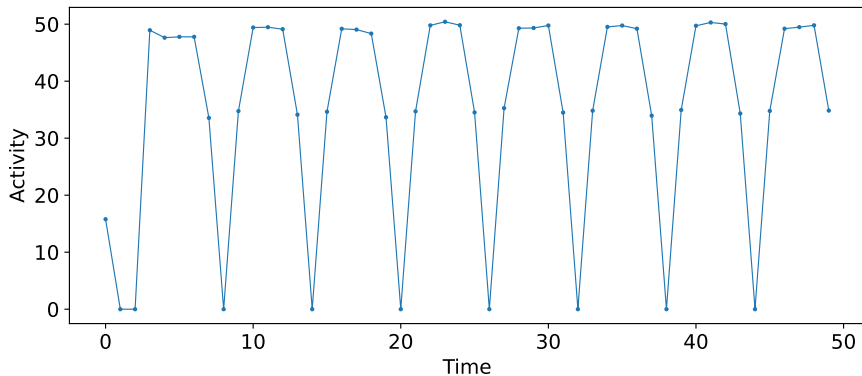
```
R1 <- 1
NOP
NOP
R2 <- R2 + R3      // acc. random
R2 <- R2 + R4      // acc. random
R2 <- R2 + R1      // acc. 1
R2 <- R2 + R5      // acc. random
R2 <- R2 + R6      // acc. random
```



```
R1 ← 0xFFFFFFFF
NOP
NOP
R2 ← R2 + R3
R2 ← R2 + R4
R2 ← R2 + R1
R2 ← R2 + R5
R2 ← R2 + R6
```



```
R0 <- 0           // index
R1 <- 1           // +1 for loop index incr.
NOP
NOP
loop: R2 <- R2 + R3 // acc. random
      R2 <- R2 + R4 // acc. random
      R0 <- R0 + R1 // i <- i + 1
      JMP loop
```

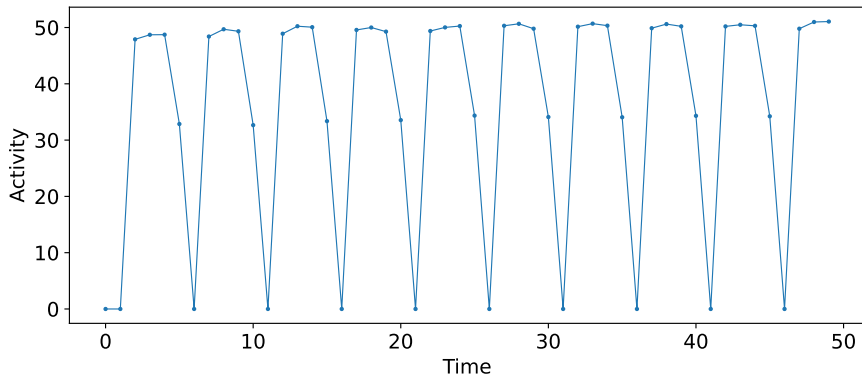




```
NOP
```

```
NOP
```

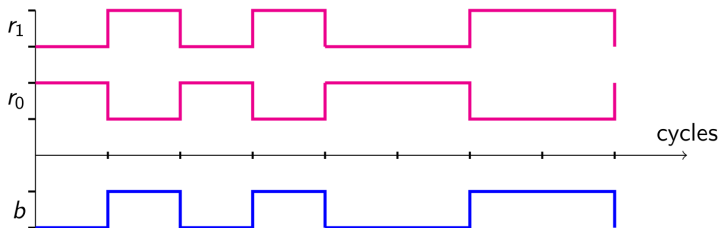
```
loop: R2 <- R2 + R3      // acc. random  
      R2 <- R2 + R4      // acc. random  
      R2 <- R2 + R10     // acc. random  
      R2 <- R2 + R10     // acc. same random  
      JMP loop
```



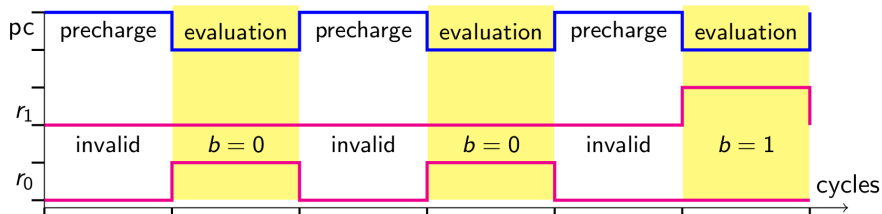
Assumptions:

- ▶  $b$  is a bit (i.e.  $b \in \{0, 1\}$ , logical or mathematical value)
- ▶ electrical states for a wire:  $V_{DD}$  (logical 1) or GND (logical 0)

	$b = 0$	$b = 1$
standard	— GND	— $V_{DD}$
dual rail	$\begin{array}{l} \text{— } r_0 = V_{DD} \\ \text{— } r_1 = \text{GND} \end{array} \Big] (1, 0)_{DR}$	$\begin{array}{l} \text{— } r_0 = \text{GND} \\ \text{— } r_1 = V_{DD} \end{array} \Big] (0, 1)_{DR}$



Precharge style:



Other encodings:

- ▶ different states for odd/even cycles
- ▶ many other solutions

Often lead to important overheads (silicon area in operators & registers)

$$X = \sum_{i=0}^{n-1} x_i \beta^i \quad \text{with} \quad x_i \in \mathcal{D}$$

- ▶ Carry-save (CS):  $\beta = 2$ ,  $\mathcal{D}_{\text{CS}} = \{0, 1, 2\}$
- ▶ Borrow-save (BS):  $\beta = 2$ ,  $\mathcal{D}_{\text{BS}} = \{-1, 0, 1\}$
- ▶ Avizienis:  $\beta > 2$ ,  $\mathcal{D}_\alpha = \{-\alpha, \dots, -1, 0, 1, \dots, \alpha\}$  with  $2\alpha + 1 > \beta$
- ▶ Other solutions

Question: how to select  $\beta$ ,  $\alpha$ , and all digits encodings?

I tested *numerous* solutions, up to now the interest is limited!

- ▶ activity variations are reduced but not enough
- ▶ leads to silicon overheads

- ▶ 2 bits with same weight


 $w=1$        $w=1$

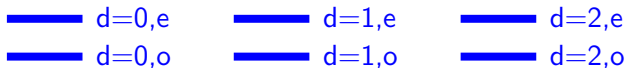
- ▶ one bit for each digit


 $d=0$        $d=1$        $d=2$

- ▶ odd/even cycles and bits of same weight


 $w=1,e$        $w=1,e$   
 $w=1,o$        $w=1,o$

- ▶ odd/even cycles and one bit for each digits


 $d=0,e$        $d=1,e$        $d=2,e$   
 $d=0,o$        $d=1,o$        $d=2,o$

- ▶ 2 bits of same weight and one bit for logical inversion


 $w=1$        $w=1$        $inv$

- ▶ other (silly?) encodings

- ▶ Redundant number systems (CS, BS, Avizienis, variants, other ideas?) help to reduce activity variations, but this is not sufficient
- ▶ Low-level encoding of digits is important
- ▶ Value 0 (as a digit and as a number) is tricky to manage w.r.t. activity variations
- ▶ RNS and variants have potential
- ▶ Combine with other arithmetic level solutions (e.g.  $GF(P)$  with Montgomery domain  $\delta P$ ,  $\delta$  small and add random multiples of  $P$ )
- ▶ Instructions and control flow are **very** important for SCA but *operands* also participate to side-channel leakage
- ▶ Do not overestimate “constant-time” protection
- ▶ Compilers (and CAD tools) optimizations can remove some protection “tricks”

- ▶ Number systems impact electrical activity and lead to side-channel leakage
- ▶ Still need more work:
  - ▶ RNS and variants
  - ▶ randomization
  - ▶ models for links between arithmetic properties and electrical properties
  - ▶ selection/design of appropriate **algorithms**/implementations
  - ▶ take into account parasitic transitions (tricky)
  - ▶ “calibration” of library components from implementation **S** result **S**
- ▶ Countermeasures against observation *and* perturbation attacks

---

**Thank you! Questions?**

arnaud.tisserand@cns.fr / <https://www.arnaud-tisserand.fr>