

Cryptographic hardware engineering

Francesco Regazzoni

Why do we need to know this?

Focus on Digital Systems



Formula 1...



Contents

- 1 Hardware Design
- 2 Performance
- 3 Reconfigurable Devices
- 4 Cryptographic Algorithms
- 5 Two interesting approaches
- 6 Side Channel

Top-Down Approach

- From an high level specification (usually abstract) to detailed design by decomposition and successive refinement
- Answers to the question: “What do we build?”
- Handles the complexity

Bottom-Up Approach

- From detailed primitive blocks to a larger and more complex functional block by combining primitives blocks
 - Answers to the question: “How do we build it?”
 - Focuses on the details
-
- Designs usually proceed from both directions simultaneously

Let's focus on the Top-Down part

The dream

- From power Point to GDSII

Let's focus on the Top-Down part

The dream

- From power Point to GDSII

Why not?

- It is too difficult!

From Idea to ASIC: the design flow....

Once upon a time....



System Requirements

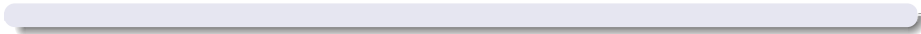
- **Functionality:** what the system will do
- **Performance:** establish the solution space
- **Cost:** set the limits to what is possible

- Function: what is expected from the chip
- Performance: how fast? what area? how much power?
- I/O requirements: how does the chip integrates with the rest of the system?

Front-End Design Flow (ASIC)

- Design: propose an architecture
- Design Entry: Description in HDL
- Verification: Ensuring that is doing what is supposed to do
- Synthesis: Mapping to target technology
- Scan Insertion: adding structure for testing

Design: Main Architectural Transformation



Create the Design Entry

- Have the block diagram
- Have your state diagram
- Define the interface
- Define the block one by one
- Define the states

■ HDL is not C++

- ▶ You are drawing block diagrams
- ▶ Always keep in mind the actual implementation
- ▶ FSM “can” be written behavioral

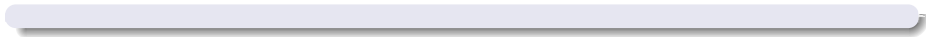
■ Separate Datapath and Control

- ▶ Datapath performs operations on data
- ▶ Multiplexers control alternative pats for operations
- ▶ FSM control multiplexers and enable signals

■ Keep it simple

- ▶ Use simple constructs
- ▶ Do not use a single sequential process
- ▶ Avoid complex sequential codes (nested ifs...)

Front-End Design Flow (ASIC)

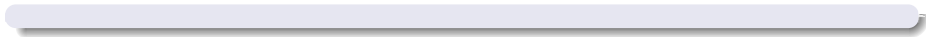


- Depends on the HDL code....
- And on the tool version! (Also Significantly!)

Back-End Design Flow (ASIC)

- Placement: putting gates on a chip
- Clock Tree insertion: Distributing the clock signal
- Routing: Connecting the gates
- Chip finishing: PADs, ...
- DRC and LVS: ensure the physical layout is correct

Back-End Design Flow (ASIC)



Contents

- 1 Hardware Design
- 2 Performance**
- 3 Reconfigurable Devices
- 4 Cryptographic Algorithms
- 5 Two interesting approaches
- 6 Side Channel

Performance Parameters

- Area: total area occupied by the circuit
- Throughput: data processed per time unit
- Power: Peak power for operation
- Energy: Total energy to complete a task
- Latency: time to process a given data item
- Time to Market: total time to complete the design

How to Improve Performance

Use More Advanced Technology

- Easy way
- Not Always Feasible
- Cost Increases
- Other Problems (adapting, missing IPs)

Make a Better Implementation

- Hard (requires brain)
- Not Always Feasible (limit in what can be done)
- Academia Should Concentrate on this

Both

- Have Similar Design Flows
- Use Same Tools
- Define Performance Similarly

...but in Industry

Chip has to work within the specification

- goal is to **sell** a product
- **Cost and Time** to design are important
- **Manufacturability** is crucial
- **Conservative** design approach is followed

EDA Tools are designed for Industry

Industry

- Circuit has to work in the worst case
- Constraints are used to define specification
- EDA tools ensure that the circuit performance meets specification

Academia

- Interested only on limits (smallest, fastest...)
- Performance numbers needed for comparison
- Not easy to get fair numbers from tools
- Constraints are “used” to explore design space

Academia Ignores some Problems

- Overhead is not always included (Large I/O bandwidth, Very fast clocks, Frequency scaling)
- Additional Verification Efforts (Interfaces between different clock domains,...)
- Testing overhead (difficult to test asynchronous designs)

■ Synthesis

- ▶ First level of Physical Properties
- ▶ Routing overhead unknown
- ▶ Timing and Power models are mean values

■ Post Layout

- ▶ Routing is known, more accurate timing
- ▶ Not all post-layout include proper provision for testing and power

■ Actual Measurements

- ▶ Real proof of concept
- ▶ Performance affected by practical problem, worst than expected
- ▶ Require a costly test infrastructure

How much silicon area will be used for the circuit?

■ Why Care

- ▶ Silicon Cost
- ▶ Feasibility (will it fit?)

■ “Should” be easy to determine

- ▶ Does not change during/after fabrication
- ▶ Reliable and accurate post-layout numbers

■ Units

- ▶ mm^2 : correct, technology dependent
- ▶ GE : commonly used, not accurate

- Total Area divided by the 2-input NAND
- Coarse measure: varies **at least** 10% between different technologies

But what is the area?

- Numbers reported after synthesis (routing missing, power missing, clock missing,...)
- Post layout gate counts (still missing power, routing, ...)
- Smallest rectangle where the block fits?
- Total die area (area can be imposed mini@sic, if the chip has more than one design?)

Numbers are approximated

- Synthesis does not tell the whole story (routing overhead can be between 10% and 500% design dependent)
- Post layout numbers are more reliable
- Total chip is easy to determine (not often the whole chip is of interest)

How fast is my circuit?

- Latency: time required to perform some action, measured in units of time (nanoseconds, clock periods, ...).
- Throughput: the number of such actions executed per unit of time, measured in units of whatever is being produced.
- Critical path: the path in the entire design with the maximum delay
- Clock frequency = $\frac{1}{T_{clk}}$ ($t_{clk} \geq$ critical path)

How do I know it?

- Ignore (in modern process it is dominant)
- Use models for synthesis (statistical look up tables function of fan out)
- Extracted (need the place and routed circuit, different level of accuracy)

- Specify Voltage and Frequency
- Specify how circuit activity is determined
- Specify how the power was measured
- Specify the corner

Contents

- 1 Hardware Design
- 2 Performance
- 3 Reconfigurable Devices**
- 4 Cryptographic Algorithms
- 5 Two interesting approaches
- 6 Side Channel

- Field Programmable Gate Arrays (FPGAs)
- Reconfigurable hardware devices
- Trade offs between ASIC and microprocessors
- Current progresses allow to store a complete SoC on FPGA

- Non recurring engineering costs
- Reduced time to market
- Always the latest technology

- Configurable blocks (look-up-tables)
- Configurable routing matrix
- Input/Output blocks
- Memory configuration
- Advanced processing elements (DSP, whole processors)

Generic Architecture



Generic Block



Generic Routing



Specific Block



Contents

- 1 Hardware Design
- 2 Performance
- 3 Reconfigurable Devices
- 4 Cryptographic Algorithms**
- 5 Two interesting approaches
- 6 Side Channel

- Well defined (we have golden model)
- Simple (based on few operations)
- Easy to test (few vectors guarantee a good coverage)

What Cryptographic Algorithm needs?

- Fast Speed (sometimes)
- Low Latency
- Low Area
- Low Power
- Low Energy

What an Hardware Designer wants?

- Not many exceptions
- Scalability (number of rounds, state size, word size power of 2)
- Be careful of auxiliary functions (padding with counters of 64 bits)
- Do not base algorithms on the capabilities of current processors

State Size Determines the Minimum Area

- Flip-Flops (unlimited access, large)
- Shift Registers (small, access limited)
- RAM arrays (small, access by word)

Tricks Example: scan-registers



Tricks Example: S-box



Trick Example FPGA: Xilinx Virtex-5

- Larger and more complex devices
- Embed multipliers, RAM memories, full processors
- Slice:
 - ▶ 4 flip-flops
 - ▶ 4 6-input LUTs
 - ▶ 2 multiplexers (F7MUX and F8MUX)
- Slices can be configured as distributed RAMs
- Very suitable for mapping 8-bit input Look-up-tables

Sbox of Oswald and Schramm

- S-box: inversion over $GF(2^8)$ and affine mapping (easy to mask):
 - ▶ Transform the masked input to the composite field $GF(2^4) \times GF(2^4)$
 - ▶ invert it there efficiently
 - ▶ transform it back to the $GF(2^8)$

Sbox of Oswald and Schramm

- S-box: inversion over $GF(2^8)$ and affine mapping (easy to mask):
 - ▶ Transform the masked input to the composite field $GF(2^4) \times GF(2^4)$
 - ▶ invert it there efficiently
 - ▶ transform it back to the $GF(2^8)$
- Oswald and Schramm approach for software:
 - ▶ perform the inversion in $GF(2^4)$ combining XOR operations with four pre-computed tables: T_{d_1} , T_{d_2} , T_m and T'_{inv} .
 - ▶ Transform the result back to $GF(2^8)$ with two additional tables: T'_{map} (from $GF(2^8)$ to $GF(2^4) \times GF(2^4)$) and $T'_{map^{-1}}$ (from $GF(2^4) \times GF(2^4)$ to $GF(2^8)$)
 - ▶ The affine transformation is integrated with the isomorphic mapping

Why it is suitable?

- Virtex-5 maps well 8-bit input Look-up-tables

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m : input two elements of $GF(2^4)$, output an element of $GF(2^4)$

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{inv} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{inv} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**

- T_{d_1} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{inv} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{map} : input an element of $GF(2^8)$, output an element of $GF(2^4)$

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**

- T_{d_1} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{inv} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{map} : input an element of $GF(2^8)$, output an element of $GF(2^4)$ ✓

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{inv} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{map} : input an element of $GF(2^8)$, output an element of $GF(2^4)$ ✓
- $T'_{map^{-1}}$: input two elements of $GF(2^4)$, output an element of $GF(2^4)$

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**

- T_{d_1} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{inv} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{map} : input an element of $GF(2^8)$, output an element of $GF(2^4)$ ✓
- $T'_{map^{-1}}$: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**

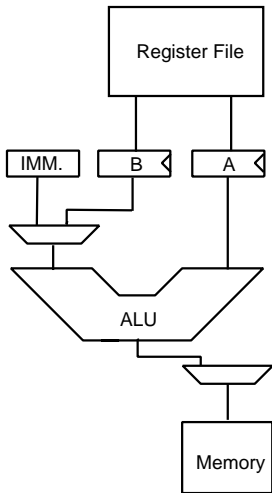
- T_{d_1} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{inv} : input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{map} : input an element of $GF(2^8)$, output an element of $GF(2^4)$ ✓
- $T'_{map^{-1}}$: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓

- All these tables have input size of 8 bits: fit **perfectly** our target FPGA

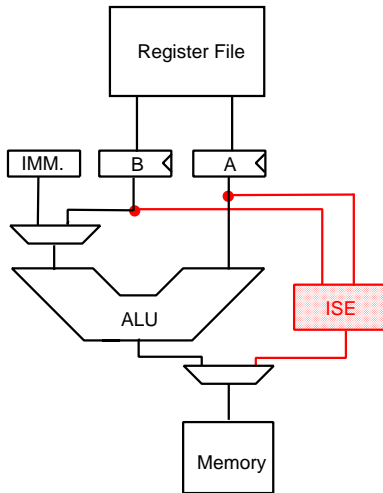
Contents

- 1 Hardware Design
- 2 Performance
- 3 Reconfigurable Devices
- 4 Cryptographic Algorithms
- 5 Two interesting approaches**
- 6 Side Channel

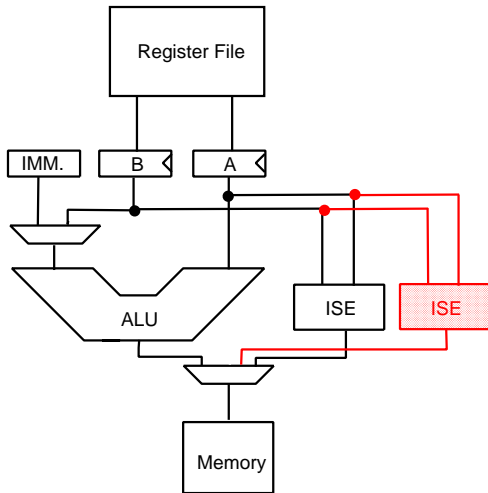
Processor



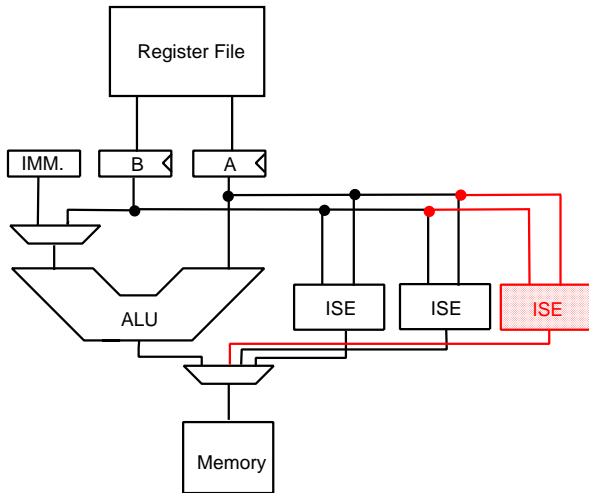
Processor Customization



Processor Customization



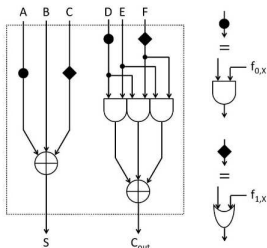
Processor Customization



Symmetric-key cryptography

- bit permutation
- rotation
- addition modulo 2^n (in ARX-based ciphers)
- $S = A \oplus B \oplus C_{in} C_{out} = AB + (A + B)C_{in} = AB \oplus AC_{in} \oplus BC_{in}$
- addition modulo 2, i.e. exclusive OR (XOR)
- substitution box (S-box)
- quadratic functions (for threshold implementations) $f(x, y, z, w) = a_0 \oplus a_1x \oplus a_2y \oplus a_3z \oplus a_4w \oplus a_{12}xy \oplus a_{13}xz \oplus a_{14}xw \oplus a_{23}yz \oplus a_{24}yw \oplus a_{34}zw$


- Fine-grained reconfigurable architecture
- Matrix of configurable Full Adder (cFA) cells
- One cFA (6 inputs and 2 outputs) can be programmed to 8 functions
- 8 functions are in standard cell libraries: re-use ASIC synthesis tools



| $f_{0,A}$ | $f_{1,C}$ | S |
|-----------|-----------|---|
| 0 | 0 | $0 \oplus B \oplus C = B \oplus C$ |
| 0 | 1 | $0 \oplus B \oplus 1 = \overline{B}$ |
| 1 | 0 | $A \oplus B \oplus C$ |
| 1 | 1 | $A \oplus B \oplus 1 = \overline{A \oplus B}$ |

| $f_{0,D}$ | $f_{1,F}$ | C_{out} |
|-----------|-----------|--|
| 0 | 0 | $0 \oplus 0 \oplus EF = EF$ |
| 0 | 1 | $0 \oplus 0 \oplus E = E$ |
| 1 | 0 | $DE \oplus DF \oplus EF = DE + (D + E)F$ |
| 1 | 1 | $DE \oplus D \oplus E = D + E$ |

3x - 9x area decrease



| cipher | Xilinx | | | | | cFA array | | | |
|------------------|--------|--------|-----------|---------------|---------|-----------|---------|---------------|--------|
| | SLICEL | SLICEM | area | critical path | conf | cFA | area | critical path | conf |
| AES-128 | 404 | 0 | 179,053 | 4.95 | 105,040 | 624 | 27,886 | 4.97 | 14,976 |
| PRESENT-80-D3 | 70 | 0 | 31,024 | 1.65 | 18,200 | 190 | 8,491 | 0.95 | 4,560 |
| PRESENT-80-D2 | 74 | 0 | 32,797 | 1.65 | 19,240 | 139 | 6,212 | 1.64 | 3,336 |
| SPECK-128/128 | 130 | 0 | 57,616 | 5.99 | 33,800 | 294 | 13,139 | 9.91 | 7,056 |
| NOEKEON | 168 | 0 | 74,458 | 3.30 | 43,680 | 288 | 12,871 | 2.41 | 6,912 |
| KTANTAN-64 | 80 | 0 | 35,456 | 2.2 | 20,800 | 119 | 5,318 | 1.95 | 2,856 |
| AES-128-TI | 2,076 | 120 | 1,092,679 | 2.2 | 582,640 | 3,058 | 136,656 | 1.54 | 73,392 |
| PRESENT-80-TI | 350 | 0 | 155,120 | 1.65 | 91,000 | 638 | 28,511 | 1.01 | 15,312 |
| SPECK-128/128-TI | 436 | 0 | 193,235 | 1.10 | 113,360 | 1556 | 69,535 | 1.33 | 37,344 |
| NOEKEON-TI | 846 | 0 | 374,947 | 2.75 | 219,960 | 952 | 42,543 | 2.12 | 22,848 |

Contents

- 1 Hardware Design
- 2 Performance
- 3 Reconfigurable Devices
- 4 Cryptographic Algorithms
- 5 Two interesting approaches
- 6 Side Channel**

What are Physical Attacks

- Physical attacks recover secrets by exploiting the implementation

Physical Attacks: the Weakest Point

**Active
Fault Injection**

**Passive
Power Analysis
Timing Analysis**

Why Physical Security is so Important Today?

Long Time Ago Past Present

Why Physical Security is so Important Today?

Long Time Ago Past Present

Mainframes Personal Computer Pervasive

From Axel Poschmann

Power Analysis Attacks

Paul Kocher, Joshua Jaffe, and Benjamin Jun, “**Differential Power Analysis**”, in Proceedings of *Advances in Cryptology-CRYPTO’99*, Santa Barbara, California, USA, August 15-19, 1999.

- Cheap
- Powerful

Examples

- Simple Power Analysis
- Differential Power Analysis

Countermeasures

Power consumption **independent** from processed key dependent data

Intermediate values of the cryptographic algorithm



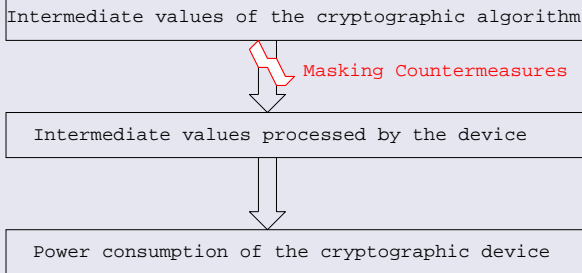
Intermediate values processed by the device



Power consumption of the cryptographic device

Countermeasures

Power consumption **independent** from processed key dependent data



Countermeasures

Power consumption **independent** from processed key dependent data

Intermediate values of the cryptographic algorithm



Masking Countermeasures

Intermediate values processed by the device



Hiding Countermeasures

Power consumption of the cryptographic device

Countermeasures

Power consumption **independent** from processed key dependent data

Intermediate values of the cryptographic algorithm



Masking Countermeasures

Intermediate values processed by the device



Hiding Countermeasures

Power consumption of the cryptographic device

They can be implemented in **Software** or in **Hardware**

INPUT:

- HDL Description
- Technological Library (area, timing, power)
- Synthetic Library (multipliers...)
- Constraints

OUTPUT:

- DPA resistant Gate Level Netlist
- Estimation of area, timing, power (!)
- Timing constraints

Approach Two

INPUT:

- HDL Description
- Technological Library (area, timing, power)
- Synthetic Library (multipliers...)
- Constraints (limit the gates)

OUTPUT:

- Gate Level Netlist

“Cell Substitution”:

- Replace cells
- Reload in the tool for correct area and timing constraints

INPUT:

- Gate level description of the circuit
- Physical view of the library (pin placement, ...)
- Constraints from synthesis

OUTPUT:

- Gate Level Netlist
- Position and interconnection of the gates
- Estimation of area, timing, power (!)

What About Resistance Assessment?

Simulation is an **Early Estimation**

- Yield
- Sawing
- Test

Questions?

Thank you for your attention!

mail: regazzoni@alari.ch