

# ENS L3 : "Systèmes numériques : de l'algorithme aux circuits"

Leçon : langages de description de matériel

Sumanta Chaudhuri  
19 Decembre 2023

# Programme

- Verilog
  - Synthetisability
- EDA (Electronic Design Automation)
  - Standard Cell Design
  - Qflow Example
- FPGA Architecture
  - EDA for FPGA
  - VPR example
- Exercices : TD en ligne

# Niveaux de description

- Transaction-Level Model
- Register-Transfert Level :
  - Comportemental
  - dataflow
- Netlist (structurel)

# Langages

- System-C (IEEE 1666)
  - Testbench
  - Dimensionnement de performances
- Verilog (IEEE 1076), VHDL (IEEE 1364)
  - Simulation + synthèse
  - Netlist

# SystemC

```
#include "systemc.h"

SC_MODULE(adder)           // module (class) declaration
{
    sc_in<int> a, b;       // ports
    sc_out<int> sum;

    void do_add()          // process
    {
        sum.write(a.read() + b.read()); //or just sum = a + b
    }

    SC_CTOR(adder)          // constructor
    {
        SC_METHOD(do_add);   // register do_add to kernel
        sensitive << a << b; // sensitivity list of do_add
    }
};
```

# Syntaxe Verilog

```
// Here is a module definition, called FRED
module FRED(q, a, b);

    input a, b;      // These are the inputs
    output q;        // Make sure your comments are helpful

    // ..... Guts of module go here ......

endmodule

// Here is a module definition with two input busses
module FRED(q, d, e);
    input [4:0] d, e;    // Ports d and e are each five bit busses
    output q;           // q is still one bit wide
endmodule
```

# Syntaxe Verilog

```
// Here are some local nets being defined
module FRED(q, a);
    input a;
    output q;
    wire [2:0] xbus, ybus;      // Two local busses of three bits each
    wire parity_x, parity_y;  // Two simple local signals.
    ...

```

```
// Here is a module definition with submodule instances.
module FRED(q, a, b);
    input a, b;    // These are the inputs
    output q;      // Make sure your comments are helpful
    wire qbar;    // Local, unported net
    NOR2 mygate(q, a, qbar), myothergate(qbar, b, q);
endmodule
```

# Syntaxe Verilog

```
// Here is a module definition with named port mapping.
module FRED(q, a, b);

    input a, b;      // These are the inputs
    output q;        // Make sure your comments are helpful
    wire qbar;       // Local, unported net

    NOR2 mygate(.x1(a), .y(q), .x2(qbar)),
               myothergate(.y(qbar), .x2(q), .x1(b));
endmodule
```

# Syntaxe Verilog

```
assign <signal> = <signal-expression> ;
```

For example

```
wire p;  
assign p = (q & r) | (~r & ~s);
```

Il y a 123 mots réservés  
dans verilog !

Syntaxic sugar :

```
wire p = (q & r) | (~r & ~s);
```

# Syntaxe Verilog

```
wire [7:0] bus1, clipbus;  
assign clipbus = (bus1 > 8'd127) ? 8'd127 : bus1;
```

```
wire [23:0] mybus = yourbus & 24'b11110000_11110000_00001111;
```

```
wire [31:0] dbus;  
wire bit7 = dbus[7];  
wire [7:0] lowsevenbits = dbus[7:0];  
wire [31:0] swapbus = dbus[0:31];
```

```
wire [11:0] boo;  
input [3:0] src;  
assign boo[11:8] = src;  
assign boo[7:4] = 4'b0;  
assign boo[3:0] = src + 4'd1;
```

```
assign boo = { src, 4'b0, src + 4'd1 };
```

# Syntaxe Verilog

Symbol	Function	Resultant width
<code>~</code>	monadic negate	as input width
<code>-</code>	monadic complement (*)	as input width
<code>!</code>	monadic logic not	unit
<code>*</code>	unsigned binary multiply (*)	sum of arg widths
<code>/</code>	unsigned binary division (*)	difference of arg widths
<code>%</code>	unsigned binary modulus (*)	width of rhs arg
<code>+</code>	unsigned binary addition	maximum of input widths
<code>-</code>	unsigned binary subtraction	maximum of input widths
<code>&gt;&gt;</code>	right shift operator	as left argument
<code>&lt;&lt;</code>	left shift operator	as left argument
<code>==</code>	net/bus comparison	unit
<code>!=</code>	inverted net/bus compare operator	unit
<code>&lt;</code>	bus compare operator	unit
<code>&gt;</code>	bus compare operator	unit
<code>&gt;=</code>	bus compare operator	unit
<code>&lt;=</code>	bus compare operator	unit
<code>&amp;</code>	diadic bitwise and	maximum of both inputs
<code>^</code>	diadic bitwise xor	maximum of both inputs
<code>~~</code>	diadic bitwise xnor (*)	maximum of both inputs
<code> </code>	diadic bitwise or	maximum of both inputs
<code>&amp;&amp;</code>	diadic logical and	unit
<code>  </code>	diadic logical or	unit
<code>? :</code>	conditional expression	maximum of data inputs

Symbol	Function
<code>&amp;</code>	and
<code> </code>	or
<code>^</code>	xor
<code>~&amp;</code>	and with final invert
<code>~ </code>	or with final invert
<code>~~</code>	xor with final invert

```
wire [9:0] mybus;
wire x = (& (mybus));
```

```
wire yes = p > r;
```

Ternaires :

```
wire [7:0] p, q, r;
wire s0, s1;

wire [7:0] bus1 = (s0) ? p : q;
wire [7:0] bus2 = (s0) ? p : (s1) ? q : r;
```

# Syntaxe Verilog : combinatoire

```
always @(aal_counter) case (aal_counter) // full_case
    0: aal_header <= 8'h00;
    1: aal_header <= 8'h17;
    2: aal_header <= 8'h2D;
    3: aal_header <= 8'h3A;
    default: aal_header <= 8'h74;
endcase
```

# Syntaxe Verilog : sequentiel

```
module EXAMPLE(ck, ex);
    input ck;          // This is the clock input
    output ex;

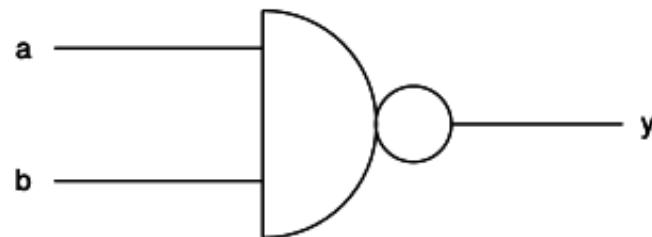
    reg ex, ez;        // ex is both a register and an output
    reg [2:0] q;       // q is a local register for counting.

    always @(posedge ck)
        begin
            if (ez) begin
                if (q == 2) ex <= 1; else ex <= ~ex;
                q <= q + 3'd1;
            end
            ez <= ~ez;
        end
    endmodule
```

- **ez** : compteur modulo 2
- **q** : compteur modulo  $2^3$ , avec enable sur ez
- **ex** : est affecté 2x, mais sous des conditions différentes

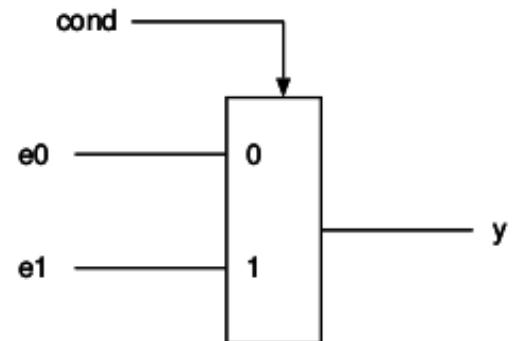
# Petits exemples de codes en Verilog

Nand Gate



```
assign y = ~(a & b);
```

2 input Mux

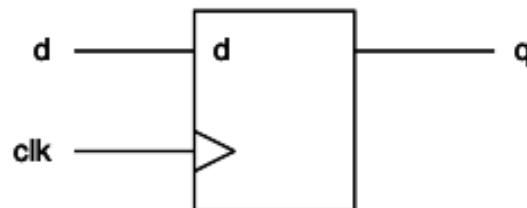


```
if (cond) y = e1;  
else y = e0;
```

```
y = (cond) ? e1 : e0;
```

# Petits exemples de codes en Verilog

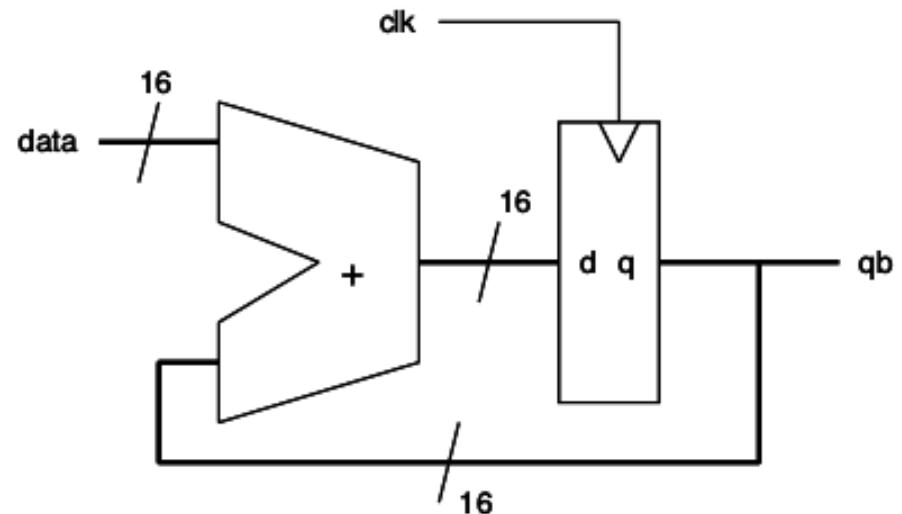
D-type FF



"On the positive edge of the clock  
the value on the d input is copied to  
the q output"

```
always @(posedge clk) q <= d;
```

Accumulator



```
wire [15..0] qb;  
reg [15..0] data  
always @(posedge clk) qb <= qb+data;
```

# Petits exemples de codes en Verilog

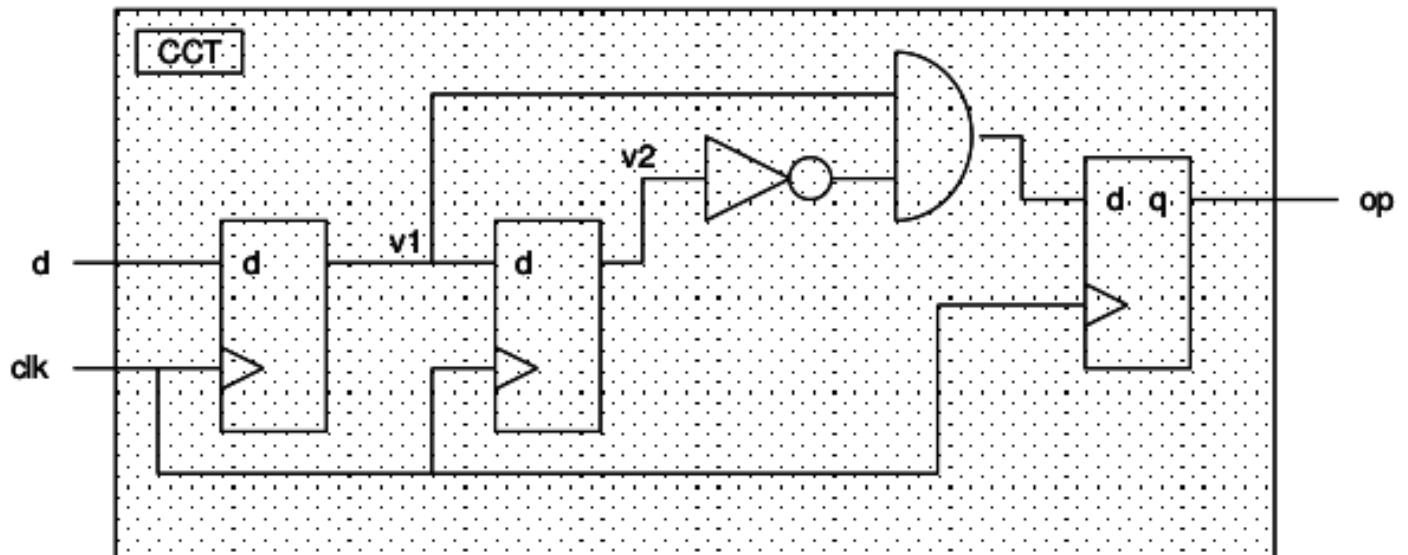
Little Circuit (pulse generator).

```
module |CCT(d, clk, op);

  input d, clk;
  output op;
  reg op;
  reg v1, v2;

  always @(posedge clk)
    begin
      v1 <= d;
      v2 <= v1;
      op <= v1 & ~v2;
    end

endmodule
```



# Powerful Critical Path Synthesis

- Performs aggressive timing driven re-structuring, mapping and gate-level optimization.
- Logic duplication for reducing the load seen by the critical path.
- Buffer high fan out nets to improve total negative slack.

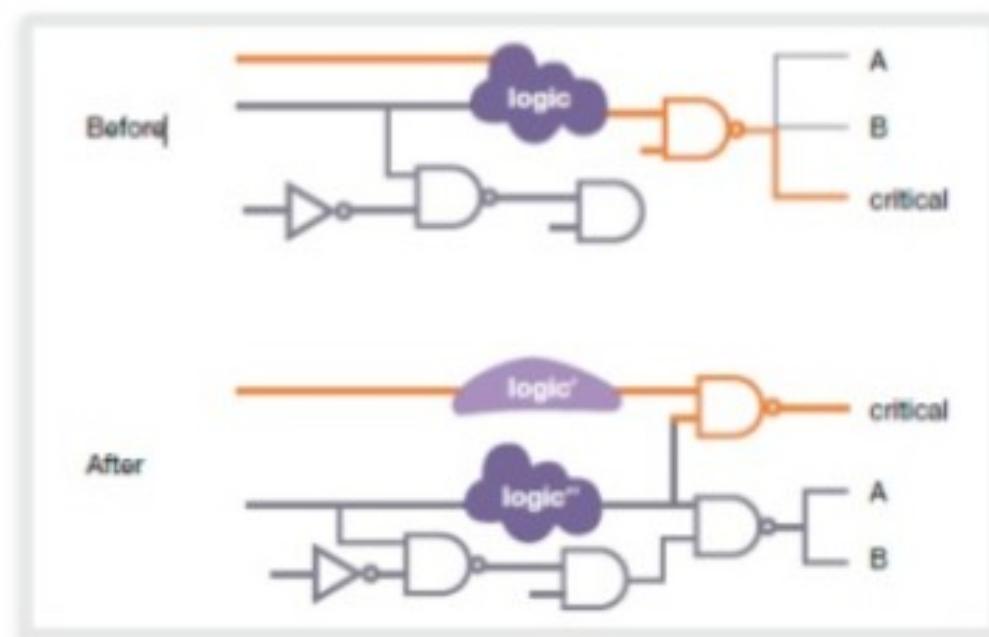
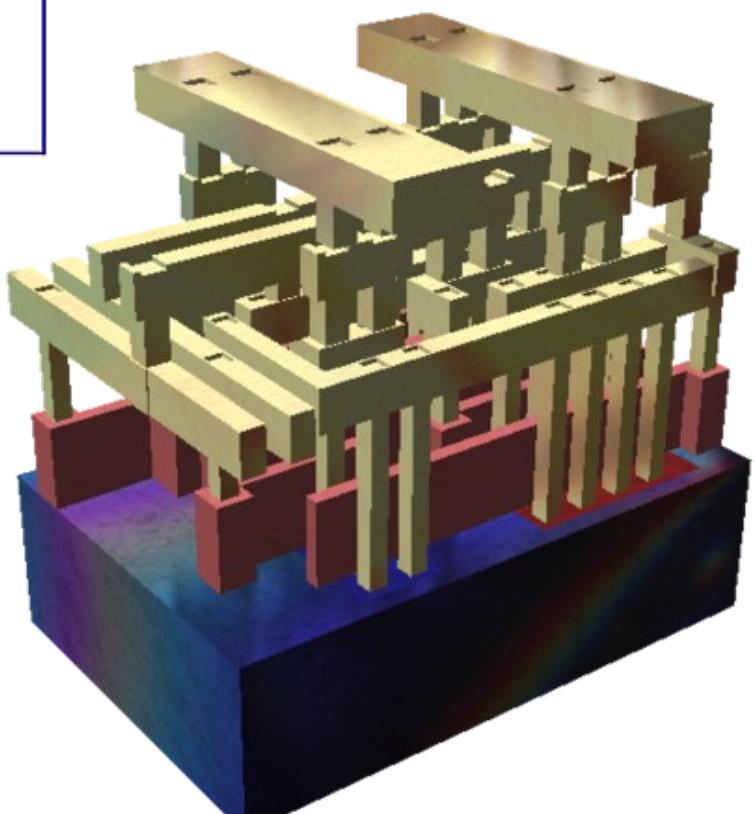
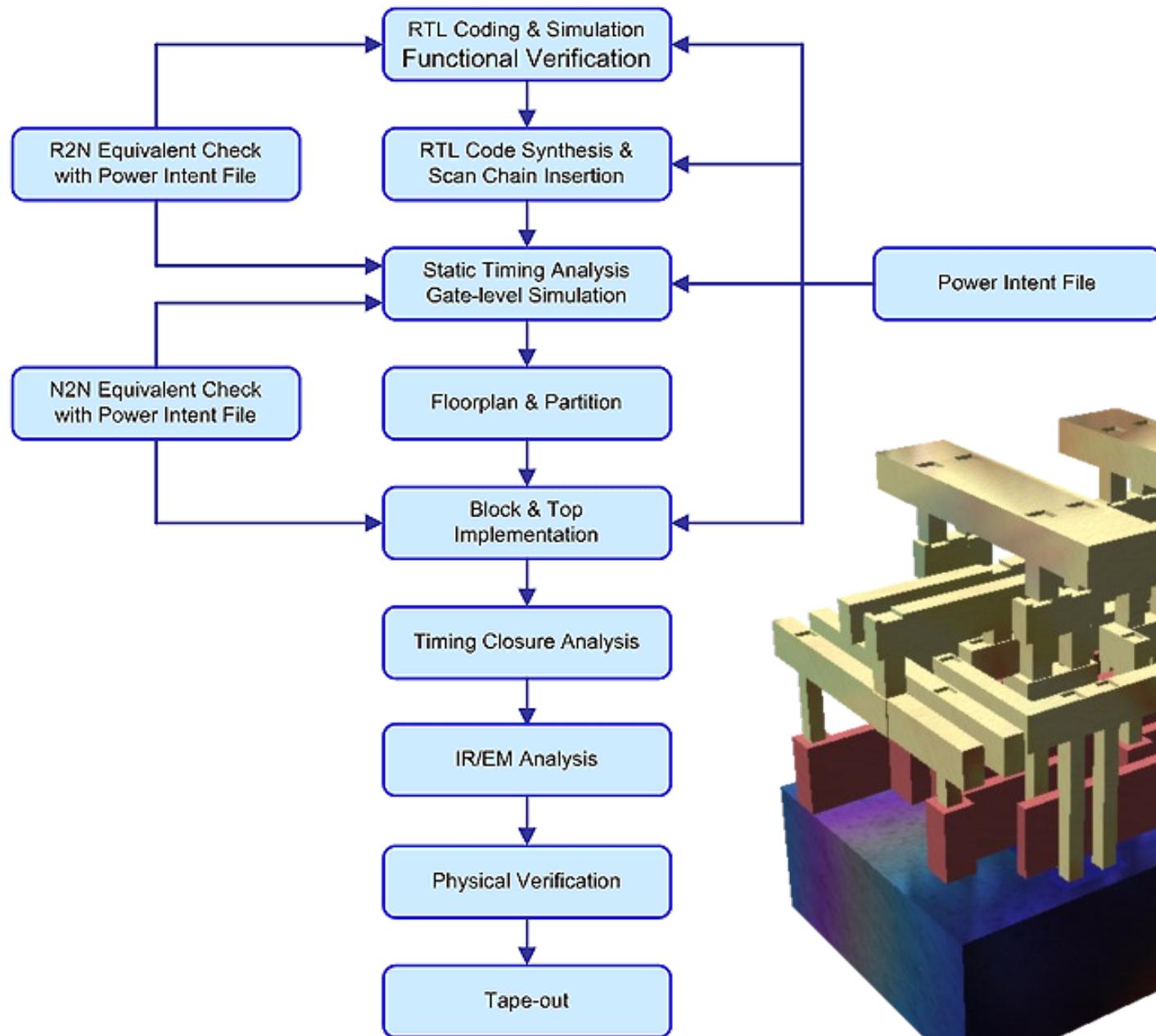


Figure 5. Register duplication [1]

# Semiconductor Manufacturing

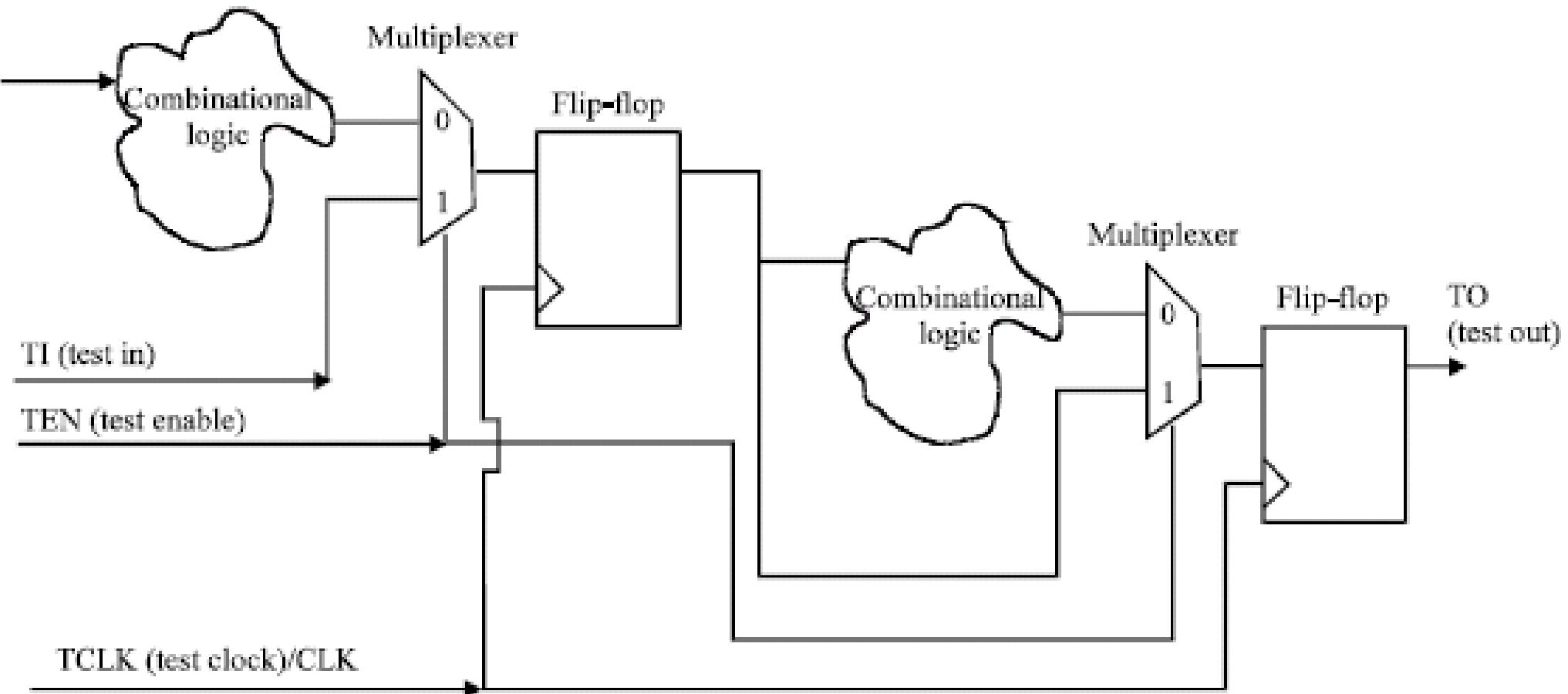
- <https://bit.ly/3qJVrfL>

# EDA



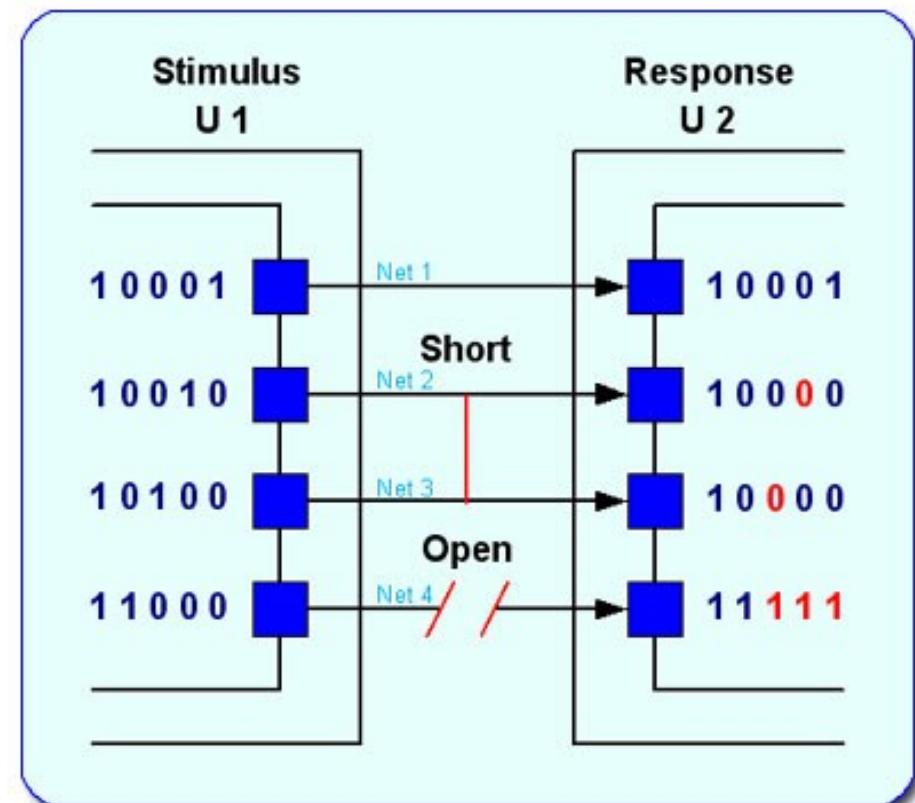
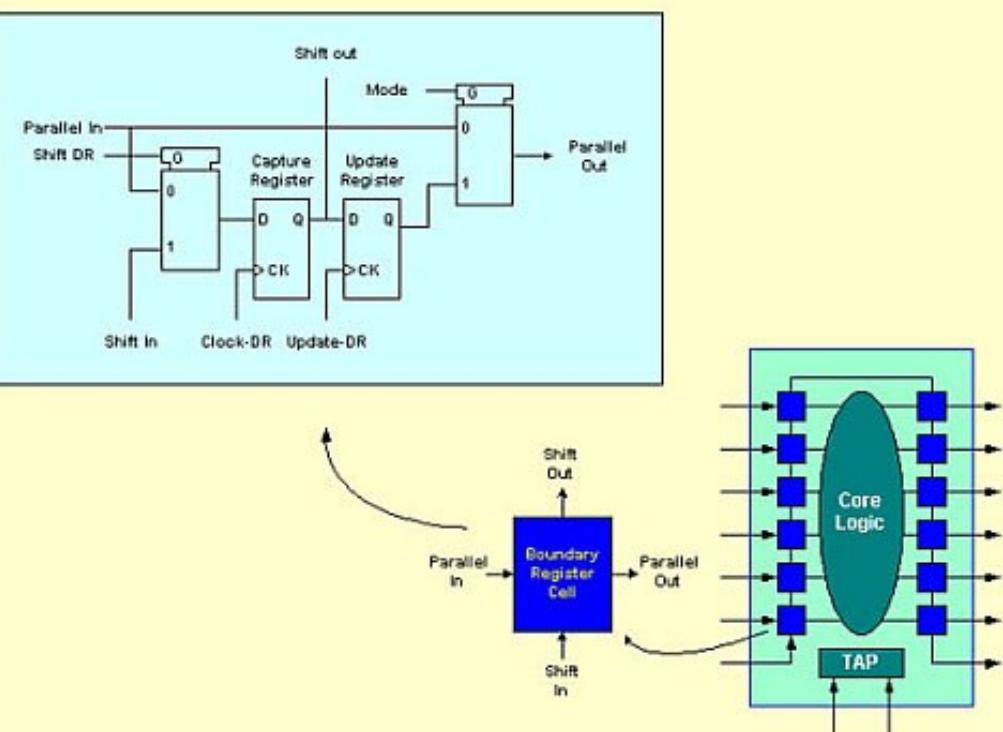
# Test des circuits intégrés

Deux modes, en fonction de TEN

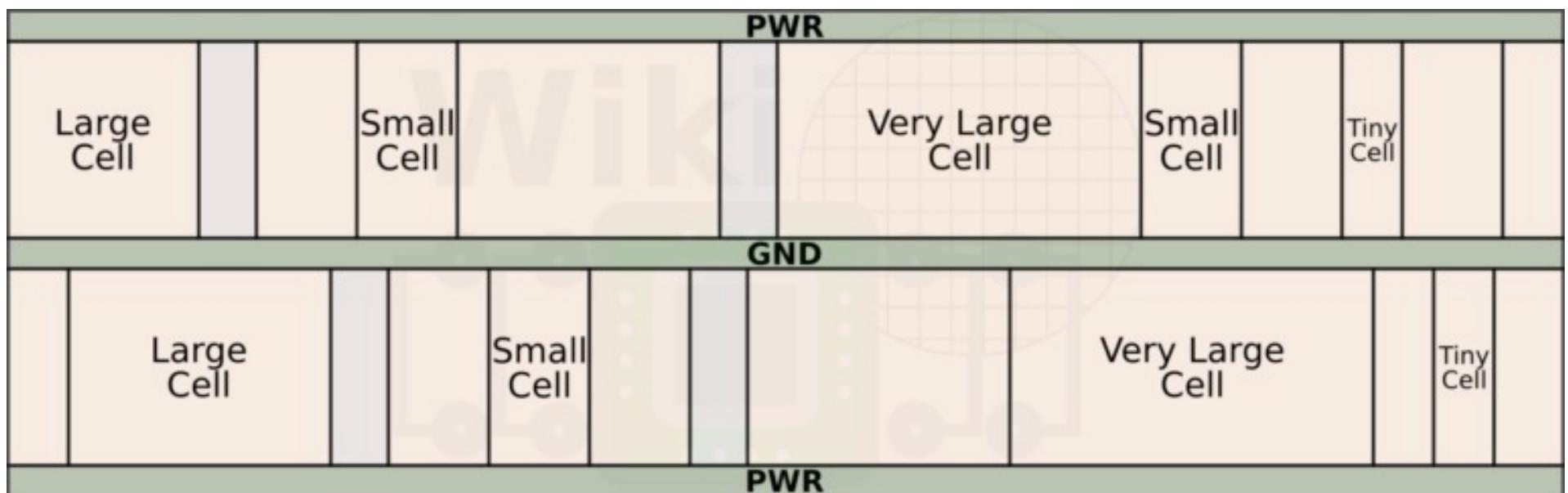


# Test des cartes électroniques

Boundary-scan, défini dans le standard IEEE Std.-1149.1

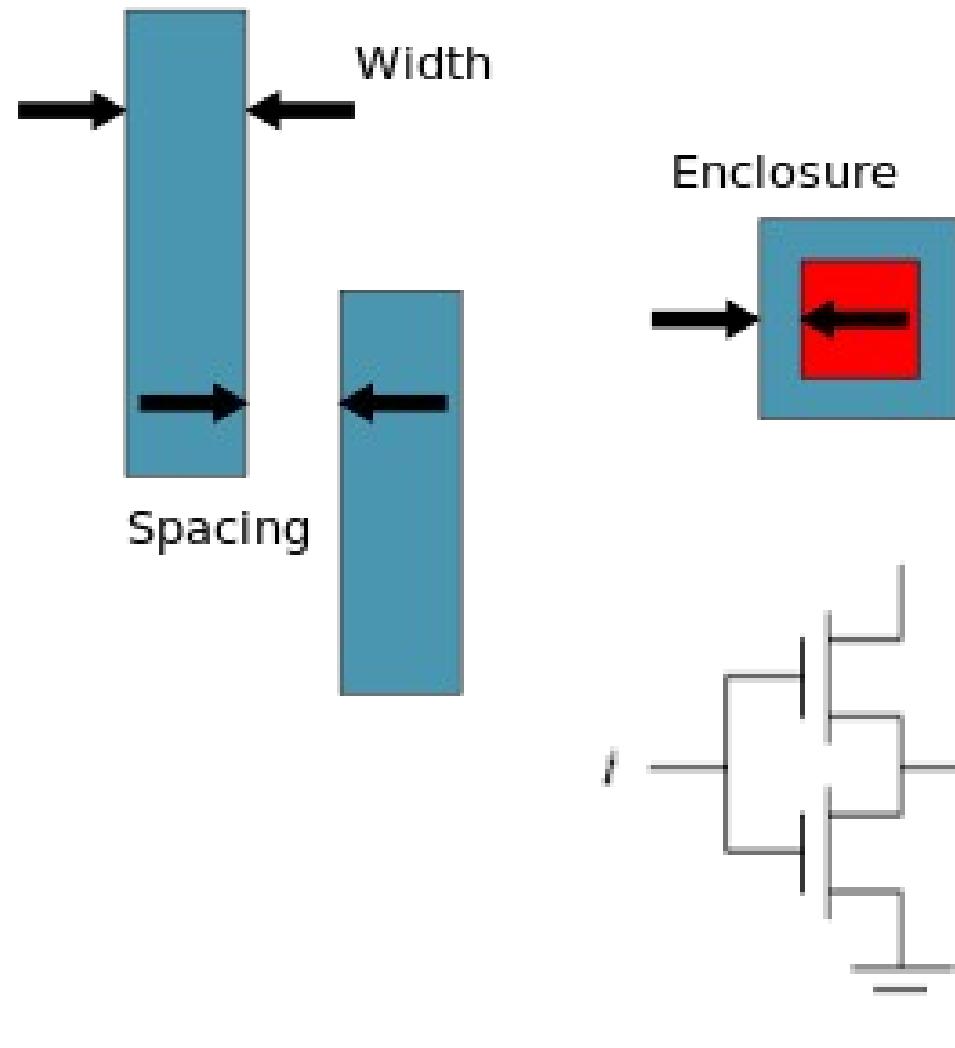


# Standard Cells



# Vérifications finales

## The three basic DRC checks



DRC : Design Rules Check  
LVS : Layout Versus Schematic

# QFLOW Example

- sudo apt-get install qflow

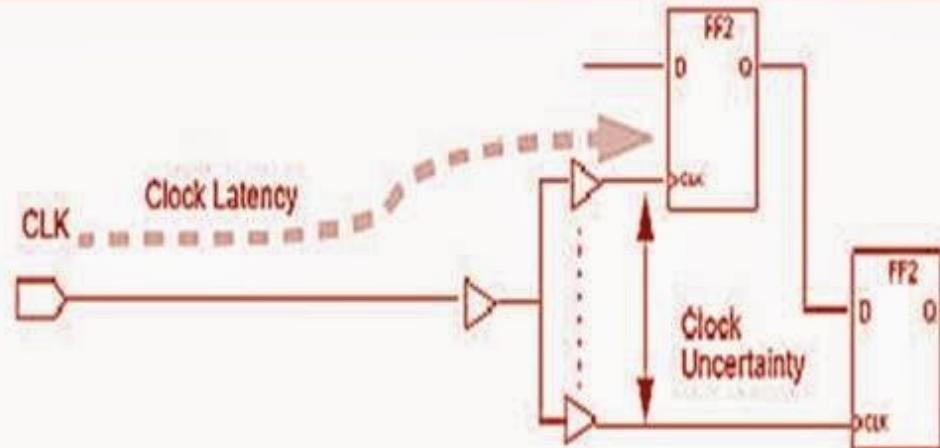
# Standard Cells : Interesting bits

- Clock Tree Synthesis
- Cell Dimension (Rent's rule)
  - $T=t G^P$
  - T : No of output wires
  - T : average number of pins
  - G : No. Of gates
  - P : Rent's parameter
- How many tracks can run over each cell ?

# Commands

## set\_clock\_uncertainty

- ❑ Models clock skew affects on the clock.
- ❑ After CTS real propagated skew is considered.



- ❑ **set\_clock\_uncertainty [-from *from\_clock*] [-rise\_from *rise\_from\_clock*] [-fall\_from *fall\_from\_clock*] [-to *to\_clock*] [-rise\_to *rise\_to\_clock*] [-fall\_to *fall\_to\_clock*] [-rise] [-fall] [-setup] [-hold] *uncertainty* [*object\_list*]**
- ❑ # Specifies the clock uncertainty for clocks or for clock-to-clock transfers.

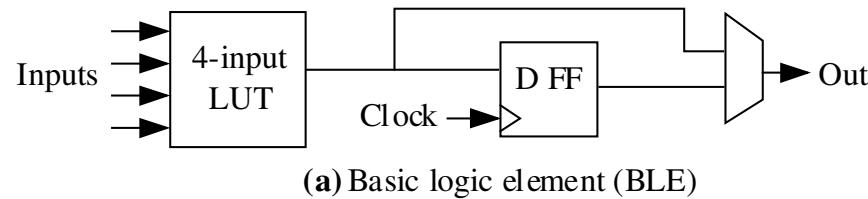
*Examples:*

- ❑ **set\_clock\_uncertainty -setup -rise -fall 0.2 [get\_clocks CLK2]**
- ❑ **set\_clock\_uncertainty -from [get\_clocks HSCLK] -to [get\_clocks SYSCLK] -hold 0.35**
- ❑ **set\_clock\_uncertainty -setup 0.475 clock**
- ❑ **set\_clock\_uncertainty -hold 0.27 clock**

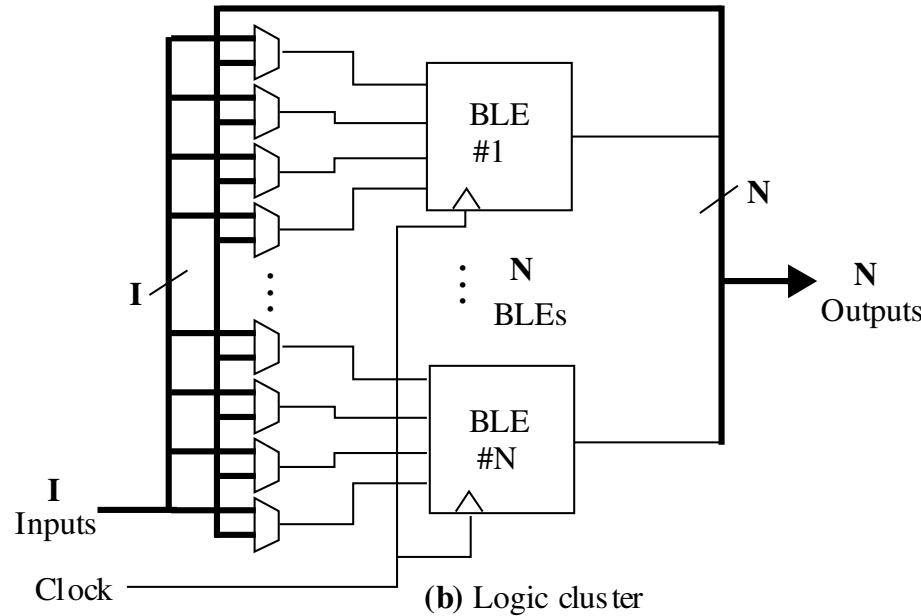
**SDC**

!

# FPGA Architecture



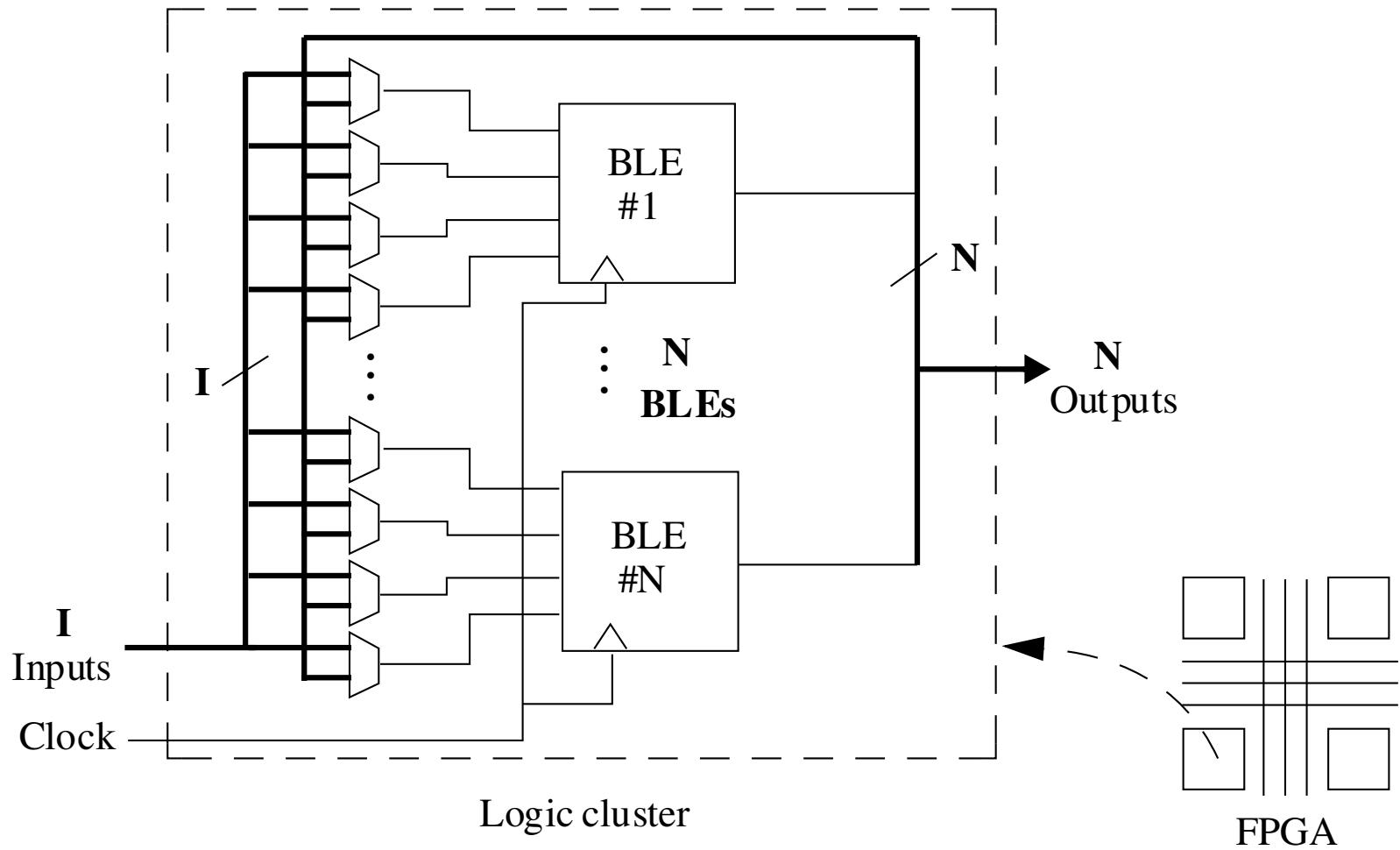
(a) Basic logic element (BLE)



(b) Logic cluster

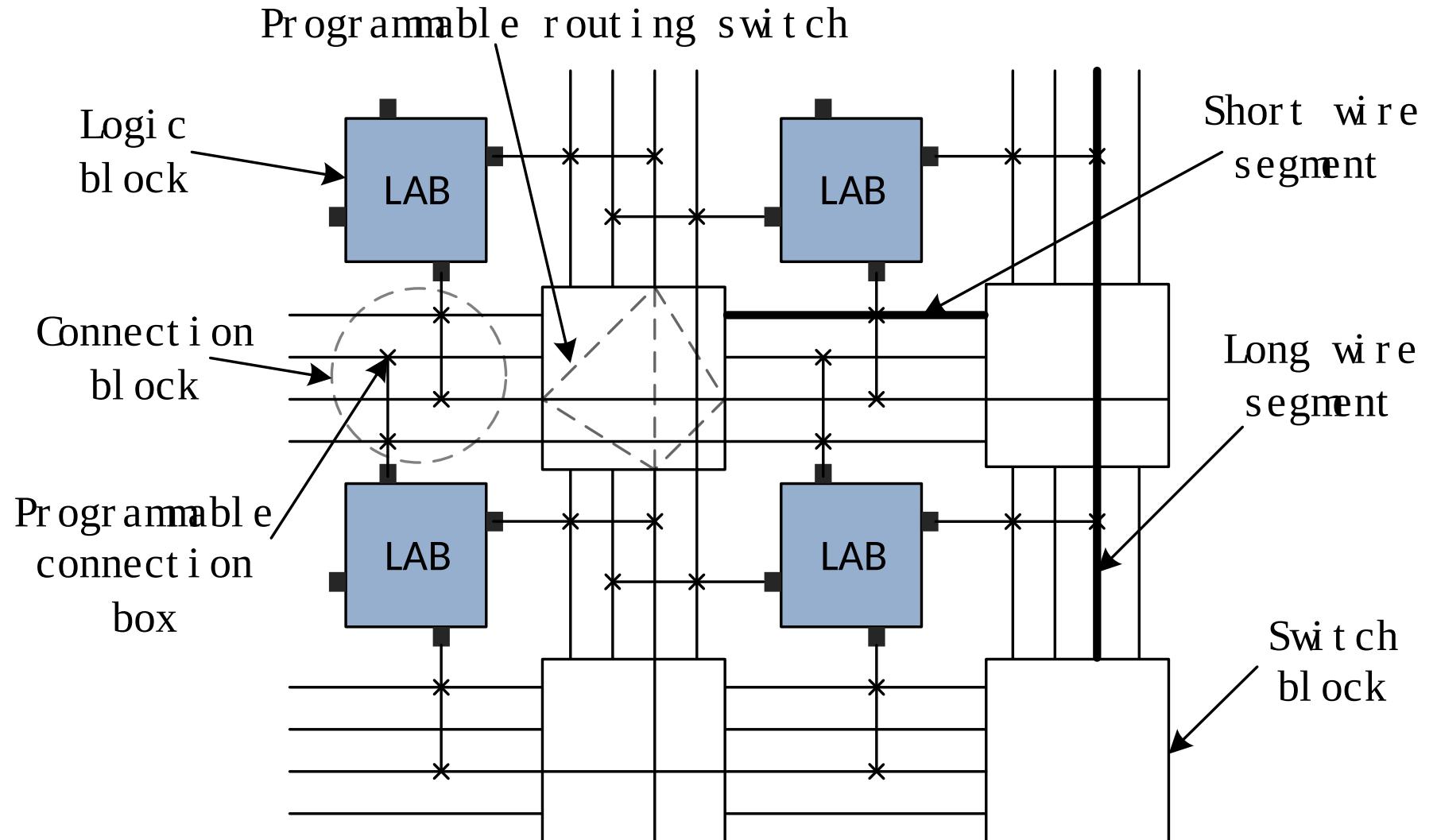
Fig. 1. Structure of basic logic element (BLE) and logic cluster.

# FPGA Architecture

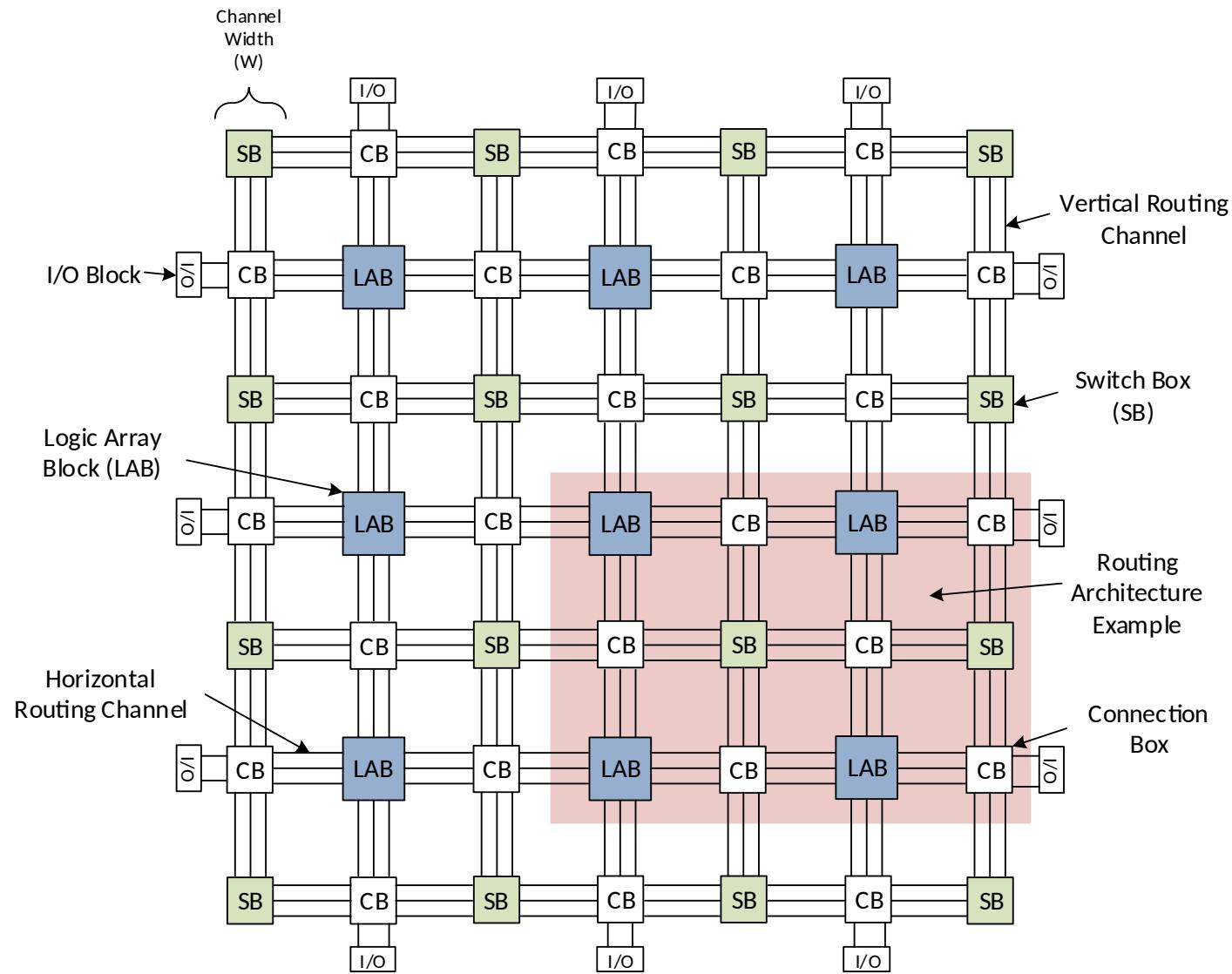


**Figure 3:** Logic cluster structure.

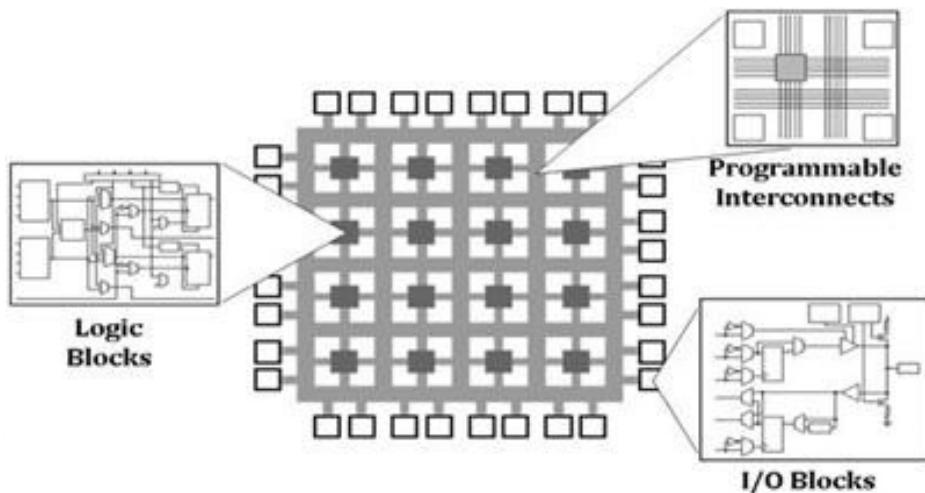
# FPGA Architecture



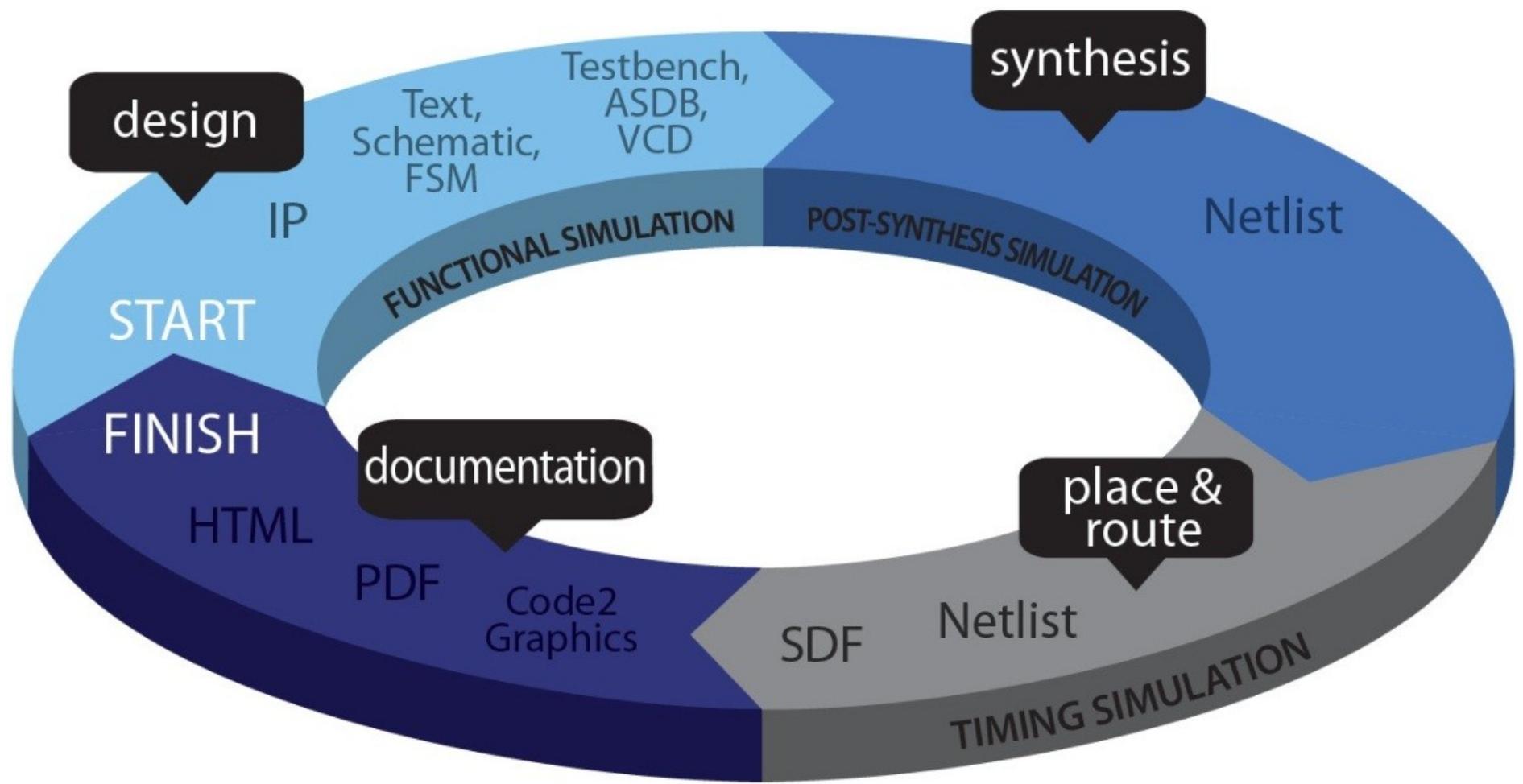
# FPGA Architecture



# EDA



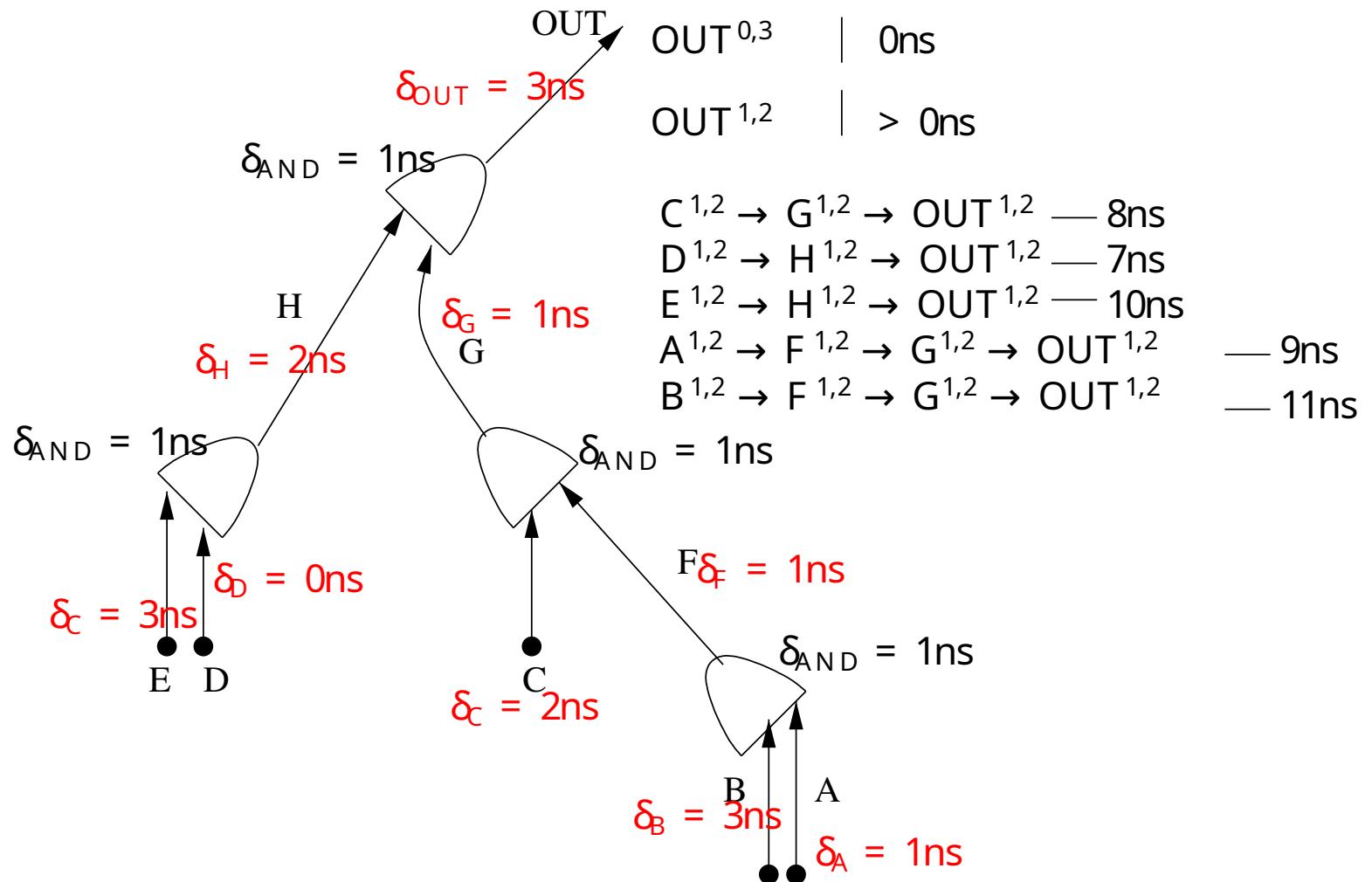
## FPGA Design



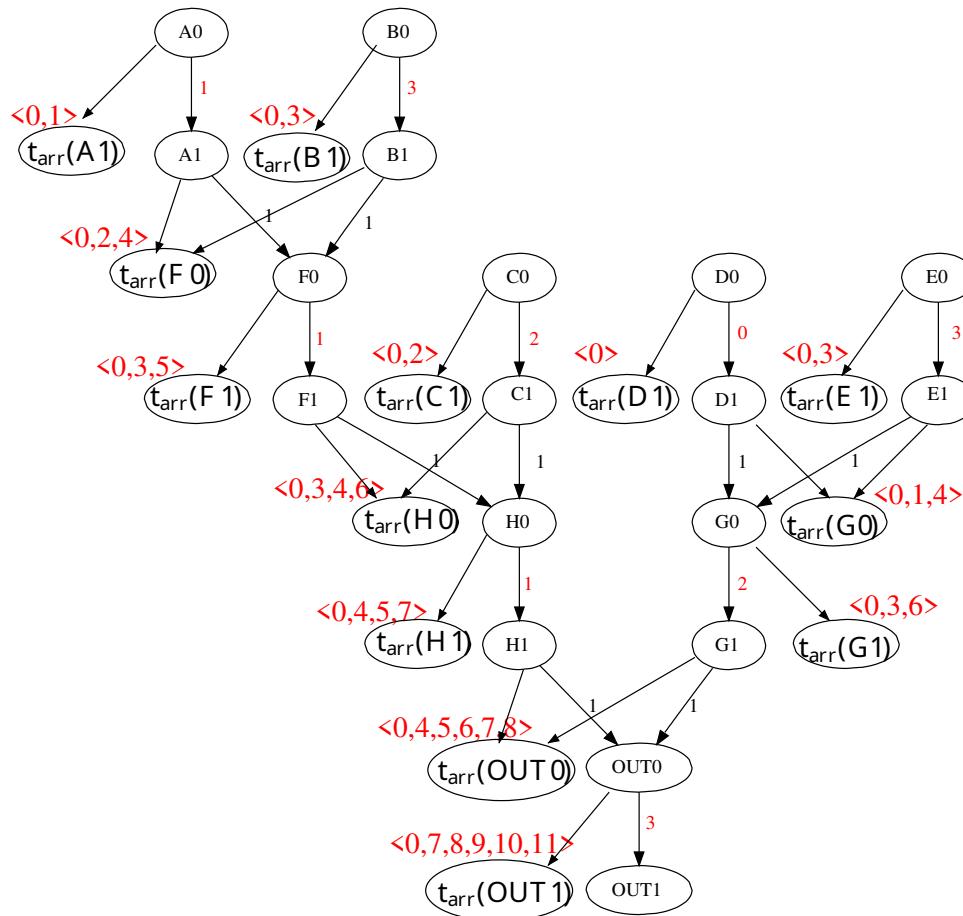
# VPR Example

- git clone  
<https://github.com/verilog-to-routing/vtr-verilog-to-routing.git>

# EDA Algorithms

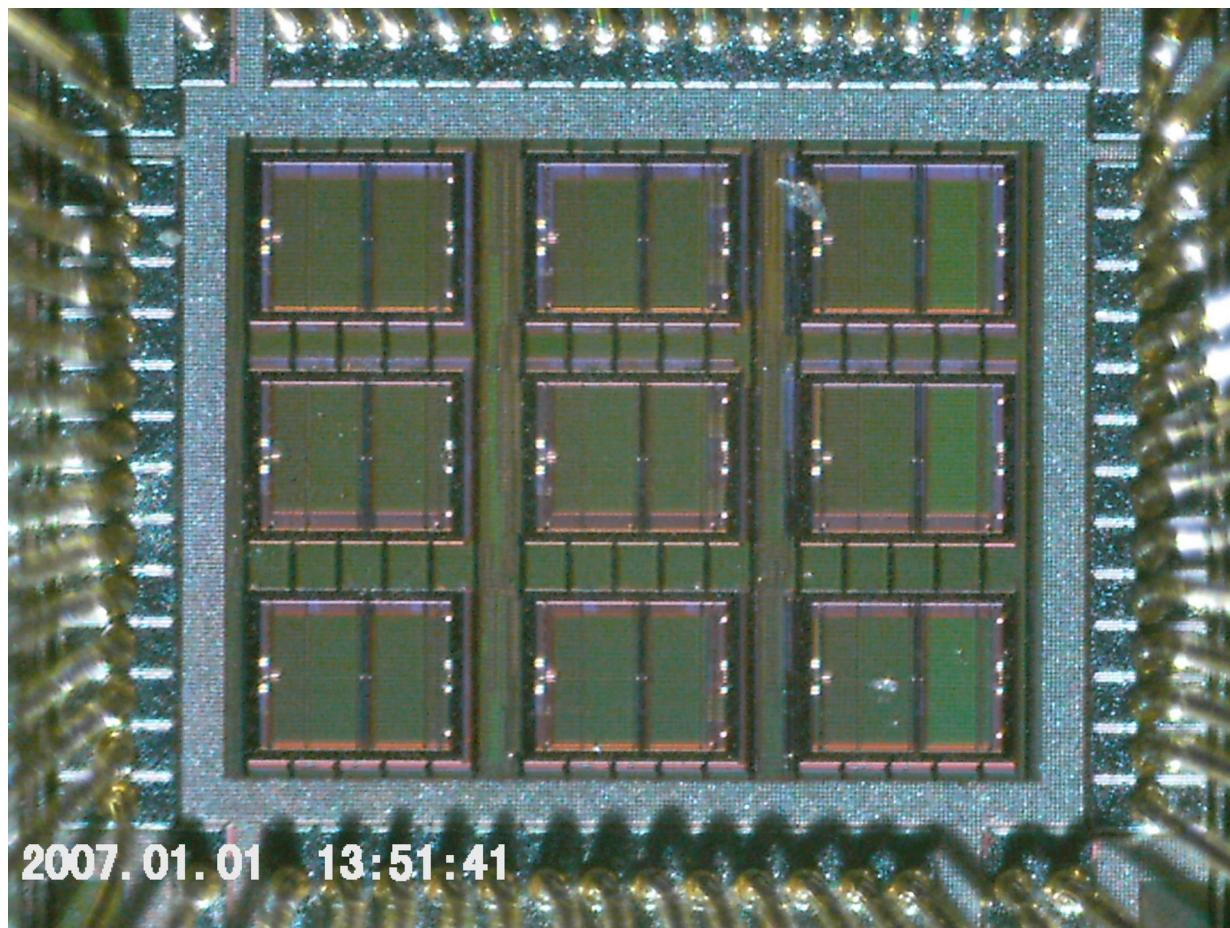


# Converting to Graph

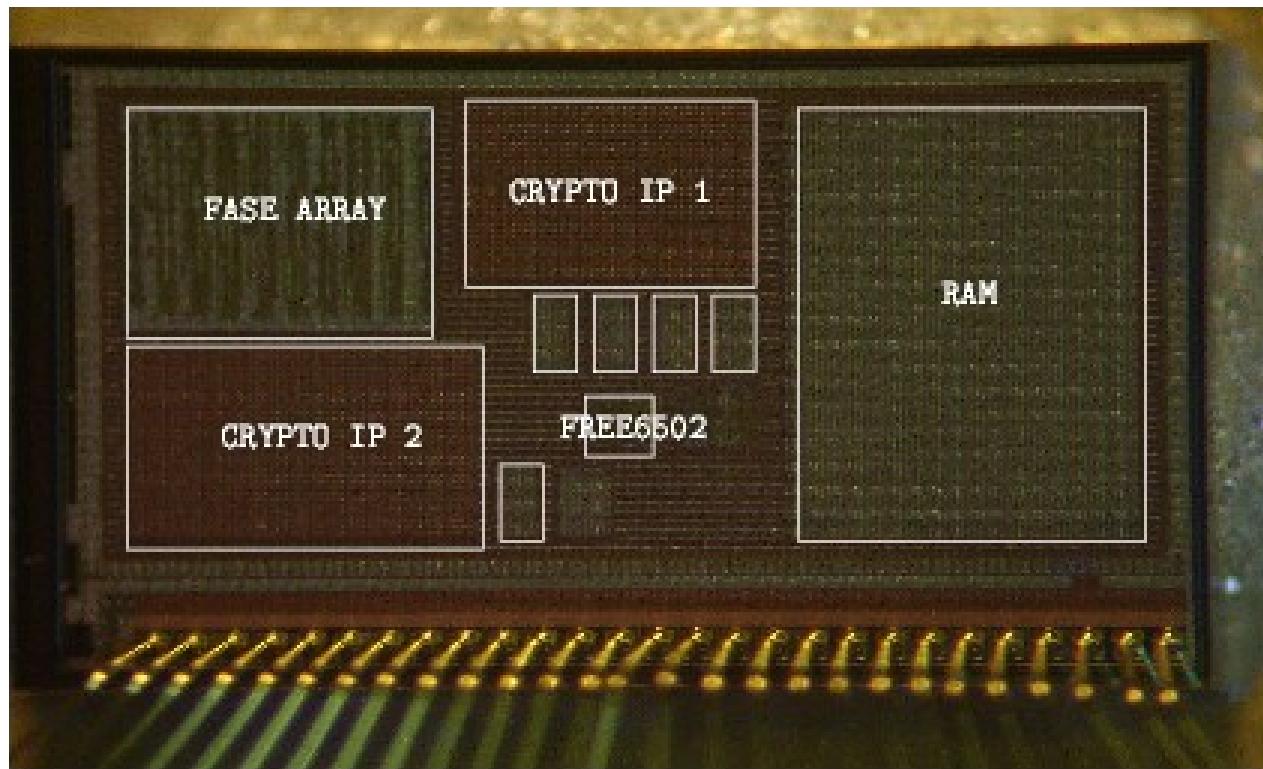


# EDA Algorithms

- Placement
  - MinCut Partition
  - Simulated Annealing
- Routing
  - Dijkstra's Algorithm
  - Maze router
- Static Timing Analysis
- LVS
  - Graph Isomorphism



2007.01.01 13:51:41



# TD (Travaux Dirigés)

- Exercice :
  - <https://perso.telecom-paristech.fr/guilley/ENS/20171205/TP/>
- Entrainement de type MOOC :
  - <http://hdl.telecom-paristech.fr/>

# Exercice 1 : Incrémenteur

# Exercice 2 : FSM

# Exercice 3 : Synthétisabilité

# Synthétisabilité

## Unsupported Verilog Language Constructs

Synthesis tools are not so intelligent. They try to infer hardware from HDL description.

Most synthesis tools does not support the following Verilog constructs:

- Unsupported definitions and declarations

- primitive definition
- time declaration
- event declaration
- triand, trior, tri1, tri0, andtrioregnet types
- Ranges and arrays for integers

- Unsupported statements

- defparam statement
- initial statement
- repeat statement
- delay control
- event control
- wait statement
- fork-join statements
- deassign statement
- force statement
- release statement
- Assignment statement with a variable used as a bit-select on the left side of the equal sign

- Unsupported operators

- Case equality and inequality operators (== and !=)
- Division and modulus operators for variables

- Unsupported gate-level constructs

- nmos, pmos, cmos, rnmos, rpmos, rcmos, pullup, pulldown, tranif0, tranif1, rtran, rtrainf0, andrtrainf1gate types

- Unsupported miscellaneous constructs

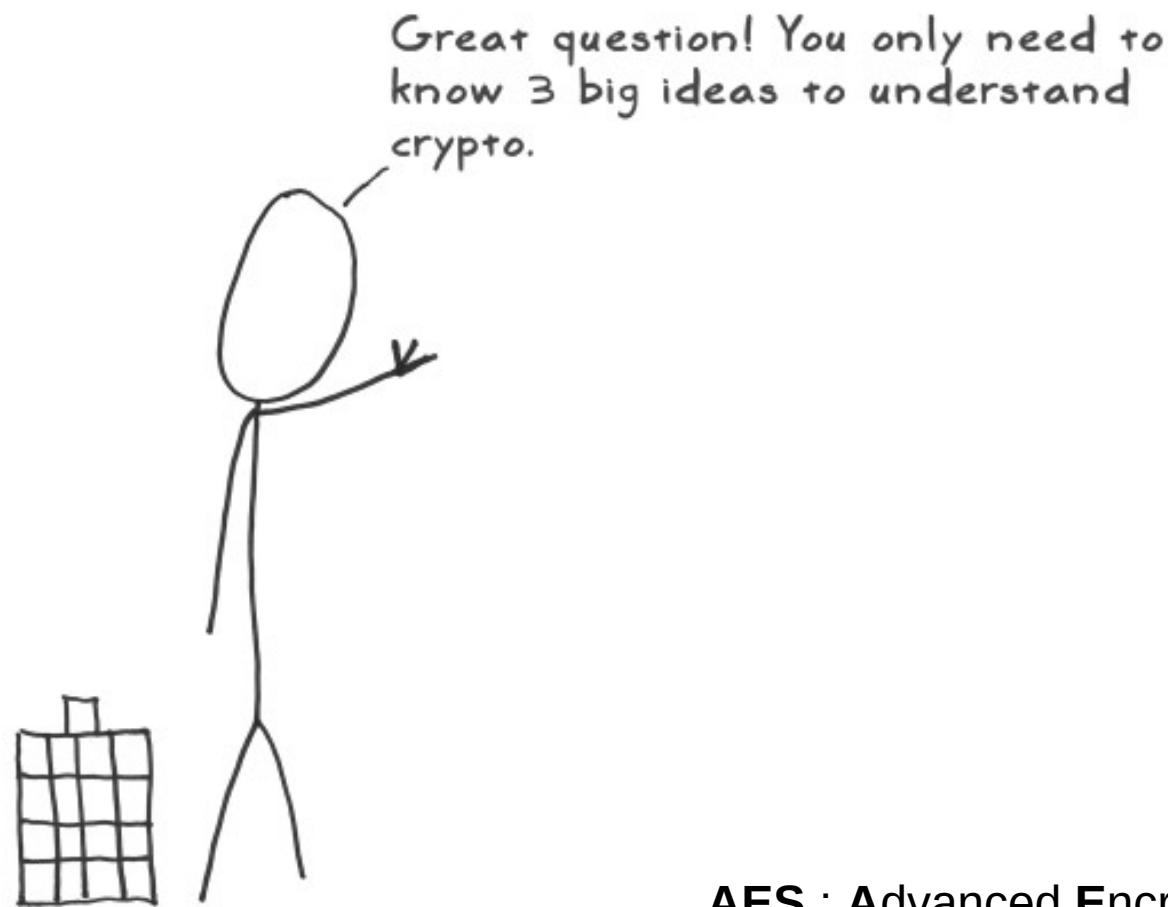
- Hierarchical names within a module

# Exercice 4 : SWAP

# Exercice 5 : BitSlicing (MAX)

# Exercice 6 : AES

# Sboxes en cryptographie (sym.)



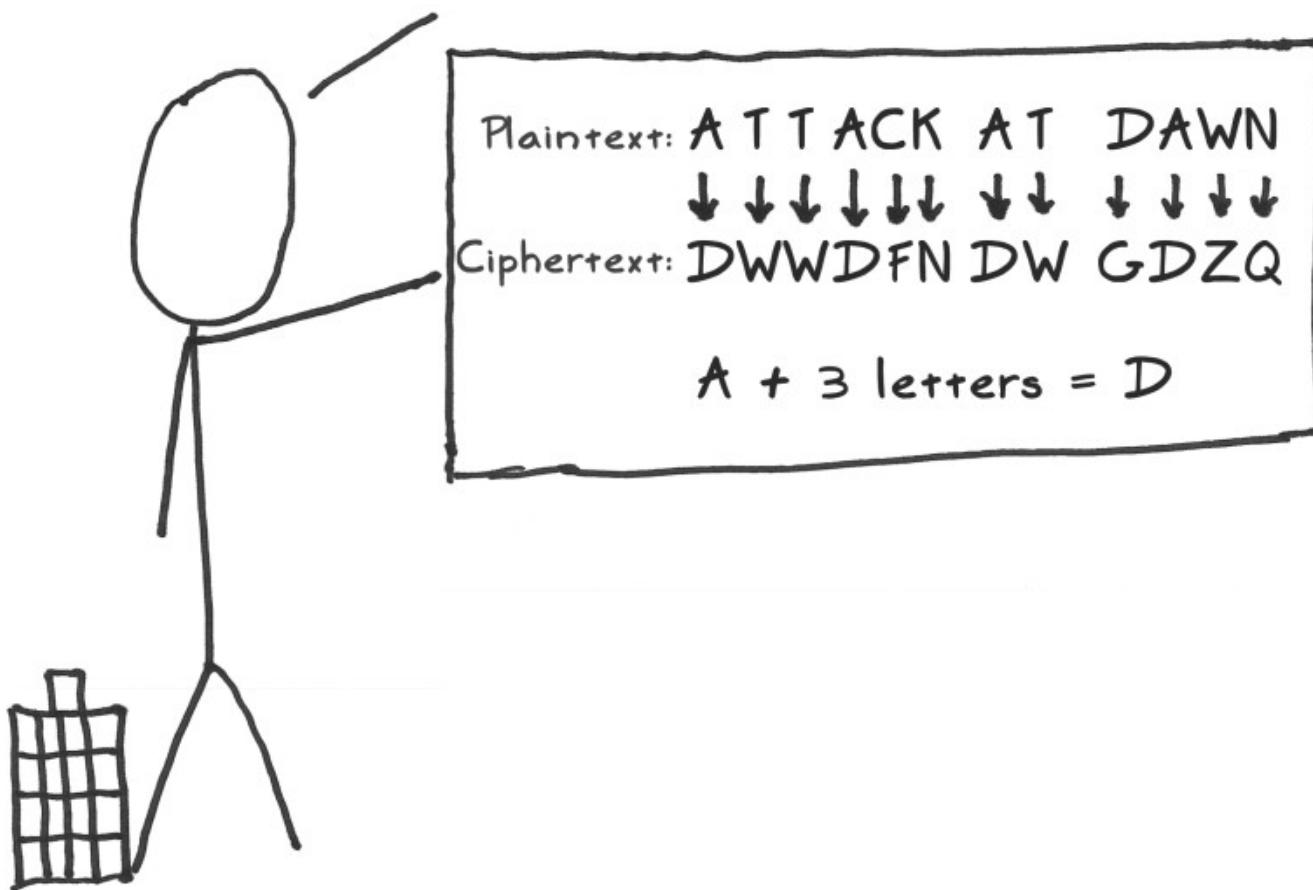
**AES** : Advanced Encryption Standard

NIST FIPS 197

Included in ISO/IEC 18033-3 standard

## Big Idea #1: Confusion

It's a good idea to obscure the relationship between your real message and your "encrypted" message. An example of this "confusion" is the trusty ol' Caesar Cipher:



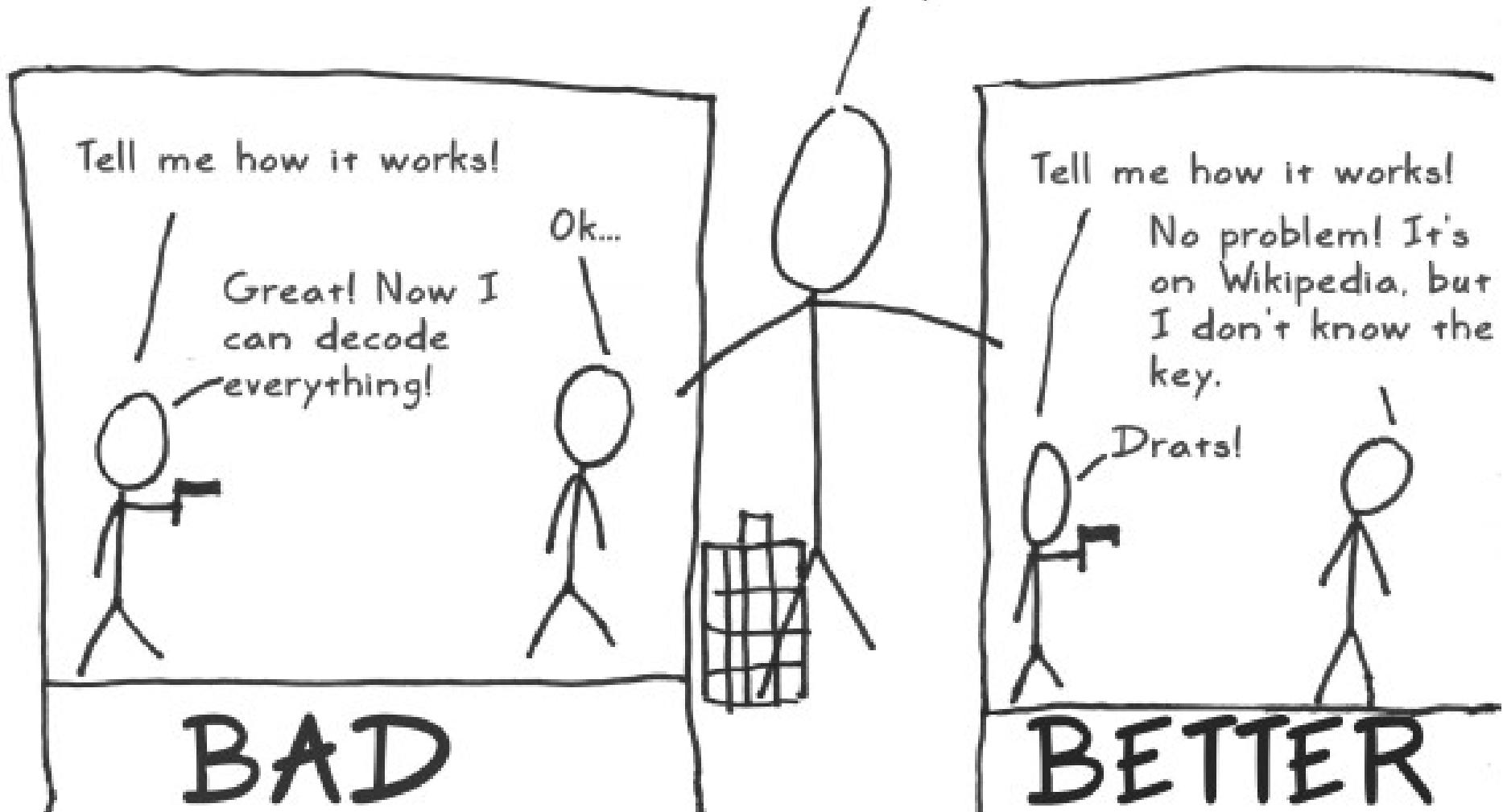
## Big Idea #2: Diffusion

It's also a good idea to spread out the message. An example of this 'diffusion' is a simple column transposition:



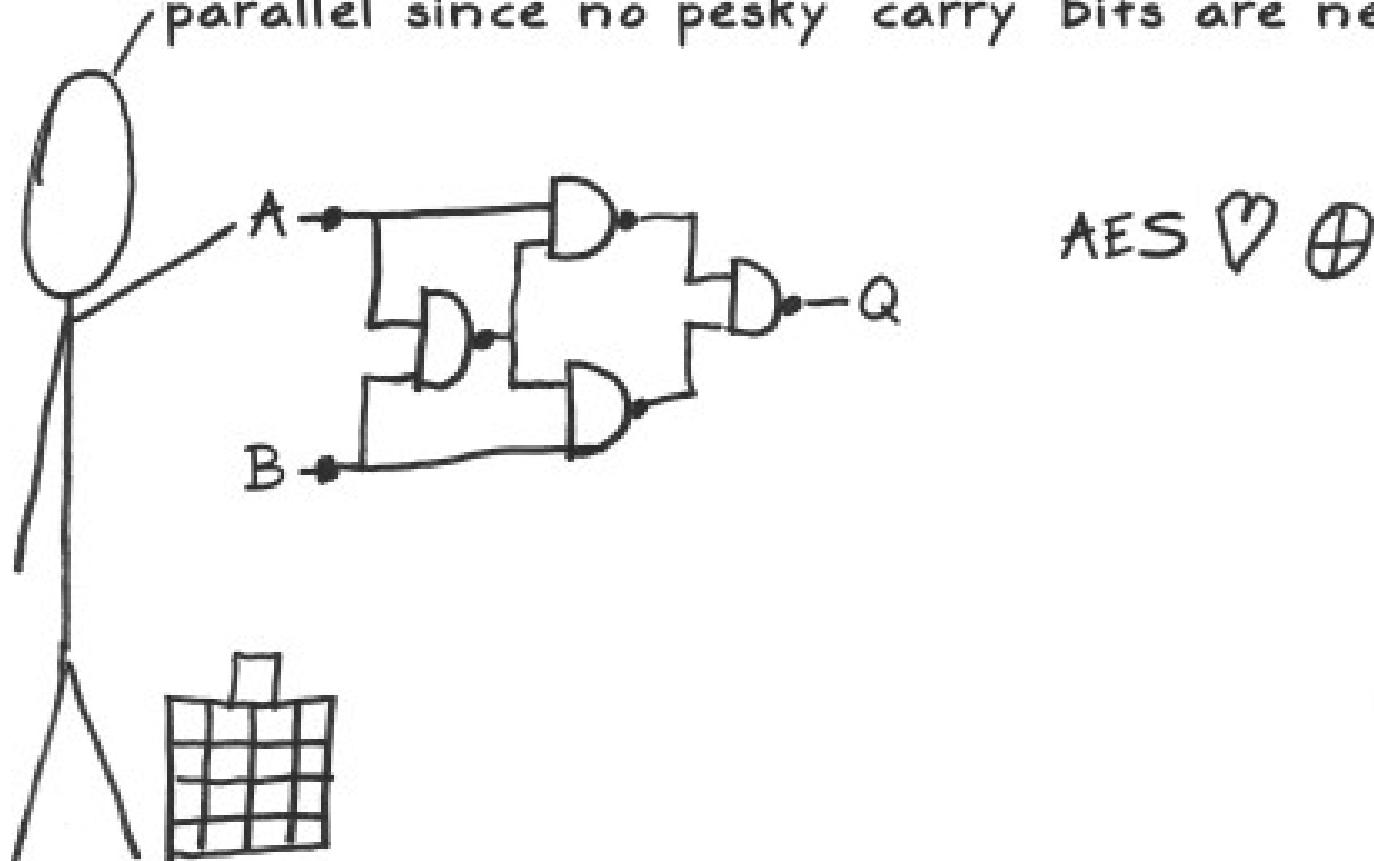
## Big Idea #3: Secrecy Only in the Key

After thousands of years, we learned that it's a bad idea to assume that no one knows how your method works. Someone will eventually find that out.



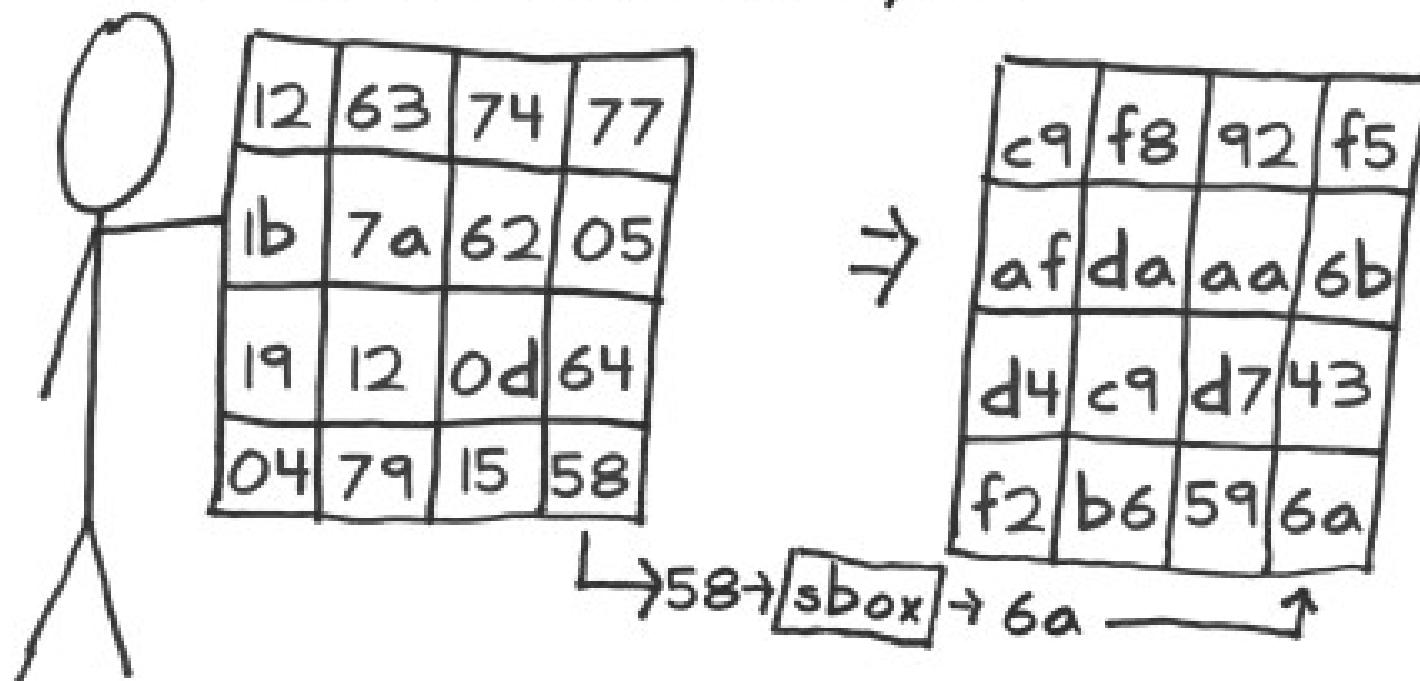
## A Tribute to XOR

There's a simple reason why I use xor to apply the key and in other spots: it's fast and cheap - a quick bit flipper. It uses minimal hardware and can be done in parallel since no pesky 'carry' bits are needed.

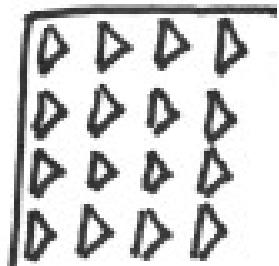


## Applying Confusion: Substitute Bytes

I use confusion (Big Idea #1) to obscure the relationship of each byte. I put each byte into a substitution box (sbox), which will map it to a different byte:



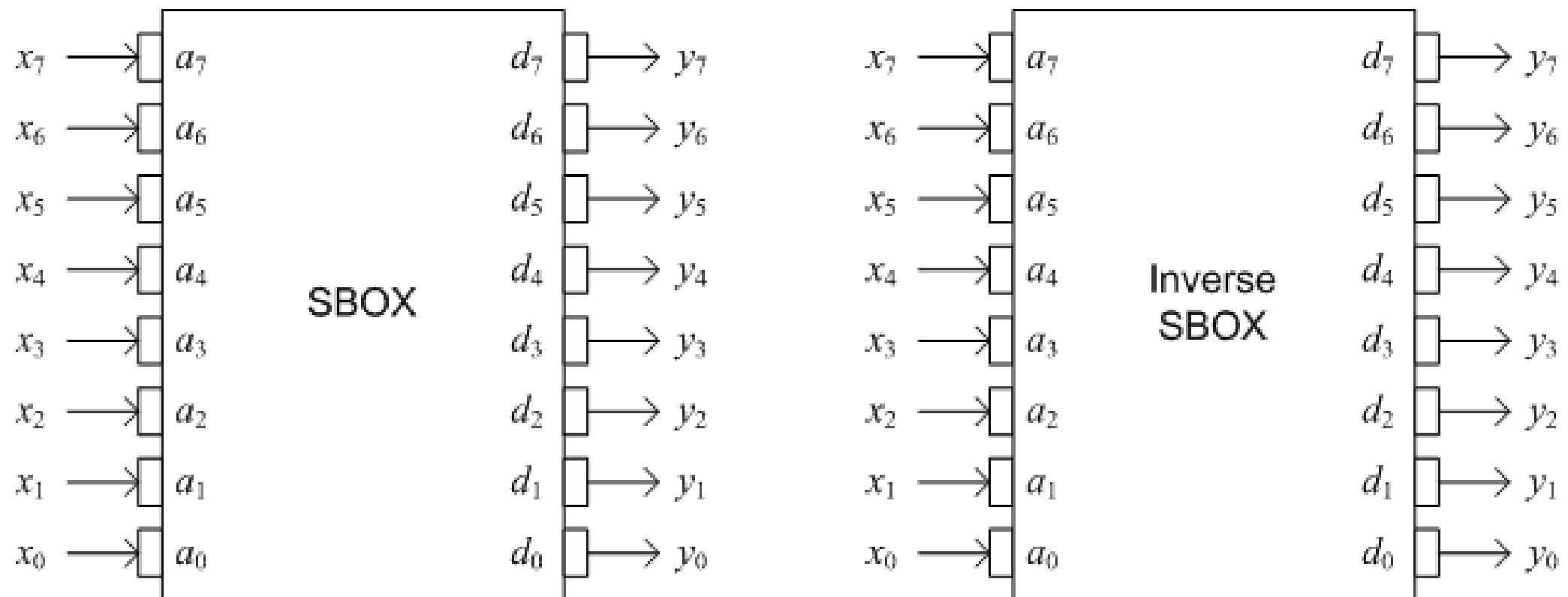
Y Denotes  
'confusion'



# SubBytes = Substitute Bytes

	<b>x0</b>	<b>x1</b>	<b>x2</b>	<b>x3</b>	<b>x4</b>	<b>x5</b>	<b>x6</b>	<b>x7</b>	<b>x8</b>	<b>x9</b>	<b>xa</b>	<b>xb</b>	<b>xc</b>	<b>xd</b>	<b>xe</b>	<b>xf</b>
<b>0x</b>	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
<b>1x</b>	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
<b>2x</b>	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
<b>3x</b>	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
<b>4x</b>	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
<b>5x</b>	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
<b>6x</b>	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
<b>7x</b>	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
<b>8x</b>	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
<b>9x</b>	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
<b>ax</b>	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
<b>bx</b>	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
<b>cx</b>	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
<b>dx</b>	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
<b>ex</b>	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
<b>fx</b>	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

# Hardware view



```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.matrix_pkg.all;

entity sbox is
  generic ( sbox_type : string );
  port( clk : in std_ulogic
        ; state_i : in byte
        ; state_o : out byte
        );
end entity;

architecture rtl of sbox is

  type rom_x8 is array(natural range <>) of byte;
  constant rom256x8 : rom_x8 ( 0 to 255 ) :=
  ( x"63", x"7C", x"77", x"7b", x"F2", x"6B", x"6F", x"C5", x"30", x"01", x"67", x"2B", x"FE", x"D7", x"AB", x"76",
    x"CA", x"82", x"C9", x"7D", x"FA", x"59", x"47", x"F0", x"AD", x"D4", x"A2", x"AF", x"9C", x"A4", x"72", x"C0",
    x"B7", x"FD", x"93", x"26", x"36", x"3F", x"F7", x"CC", x"34", x"A5", x"E5", x"F1", x"71", x"D8", x"31", x"15",
    x"04", x"C7", x"23", x"C3", x"18", x"96", x"05", x"9A", x"07", x"12", x"80", x"E2", x"EB", x"27", x"B2", x"75",
    x"09", x"83", x"2C", x"1A", x"1B", x"6E", x"5A", x"A0", x"52", x"3B", x"D6", x"B3", x"29", x"E3", x"2F", x"84",
    x"53", x"D1", x"00", x"ED", x"20", x"FC", x"B1", x"5B", x"6A", x"CB", x"BE", x"39", x"4A", x"4C", x"58", x"CF",
    x"D0", x"EF", x"AA", x"FB", x"43", x"4D", x"33", x"85", x"45", x"F9", x"02", x"7F", x"50", x"3C", x"9F", x"A8",
    x"51", x"A3", x"40", x"8F", x"92", x"9D", x"38", x"F5", x"BC", x"B6", x"DA", x"21", x"10", x"FF", x"F3", x"D2",
    x"CD", x"0C", x"13", x"EC", x"5F", x"97", x"44", x"17", x"C4", x"A7", x"7E", x"3D", x"64", x"5D", x"19", x"73",
    x"60", x"81", x"4F", x"DC", x"22", x"2A", x"90", x"88", x"46", x"EE", x"B8", x"14", x"DE", x"5E", x"0B", x"DB",
    x"E0", x"32", x"3A", x"0A", x"49", x"06", x"24", x"5C", x"C2", x"D3", x"AC", x"62", x"91", x"95", x"E4", x"79",
    x"E7", x"C8", x"37", x"6D", x"8D", x"D5", x"4E", x"A9", x"6C", x"56", x"F4", x"EA", x"65", x"7A", x"AE", x"08",
    x"BA", x"78", x"25", x"2E", x"1C", x"A6", x"B4", x"C6", x"E8", x"DD", x"74", x"1F", x"4B", x"BD", x"8B", x"8A",
    x"70", x"3E", x"B5", x"66", x"48", x"03", x"F6", x"0E", x"61", x"35", x"57", x"B9", x"86", x"C1", x"1D", x"9E",
    x"E1", x"F8", x"98", x"11", x"69", x"D9", x"8E", x"94", x"9B", x"1E", x"87", x"E9", x"CE", x"55", x"28", x"DF",
    x"8C", x"A1", x"89", x"0D", x"BF", x"E6", x"42", x"68", x"41", x"99", x"2D", x"0F", x"B0", x"54", x"BB", x"16");
begin

  sbox_ram_gen: if sbox_type = "RAM" generate
    process(clk)
    begin
      if falling_edge(clk) then
        state_o <= rom256x8( to_integer(unsigned(state_i)) );
      end if;
    end process;
  end generate;

end architecture;

```

# Synthèse logique

```
# Synthesis script.
# Launch with the command:
# >>> nc -gui -files syn.tcl

set_attribute library "/comelec/softs/opt/opus_kits/HCMOS9GP_REVG/CORE9GPLL_SNPS_AVT_4.1/SIGNOFF/bc_1.
32V_0C_wc_1.08V_105C/PT_LIB/CORE9GPLL_Worst.lib"
read_hdl -vhdl ".../src/aes/matrix_pkg.vhd" ".../src/aes/mixcolumns.vhd" ".../src/aes/sbox.v
hd ".../src/aes/shiftrows.vhd" ".../src/aes/state.vhd" ".../src/aes/subbyt
es.vhd ".../src/aes/addroundkey.vhd" ".../src/aes/cipher.vhd" ".../src/aes/fsm.vh
d ".../src/aes/keyexpansion.vhd" ".../src/aes/aes.vhd" ".../src/aes/aes_25
6_fast.vhd ".../src/sensor/sensor.vhd" ".../src/top/top.vhd"

elaborate
synthesize -to_mapped

# Netlist, to be used in encounter
write_hdl -v2001 > top.vm
report power > rpt_power.txt
report area > rpt_area.txt
report memory > rpt_memory.txt
report gates > rpt_gates.txt
report timing > rpt_timing.txt
report timing -lint > rpt_timing.txt

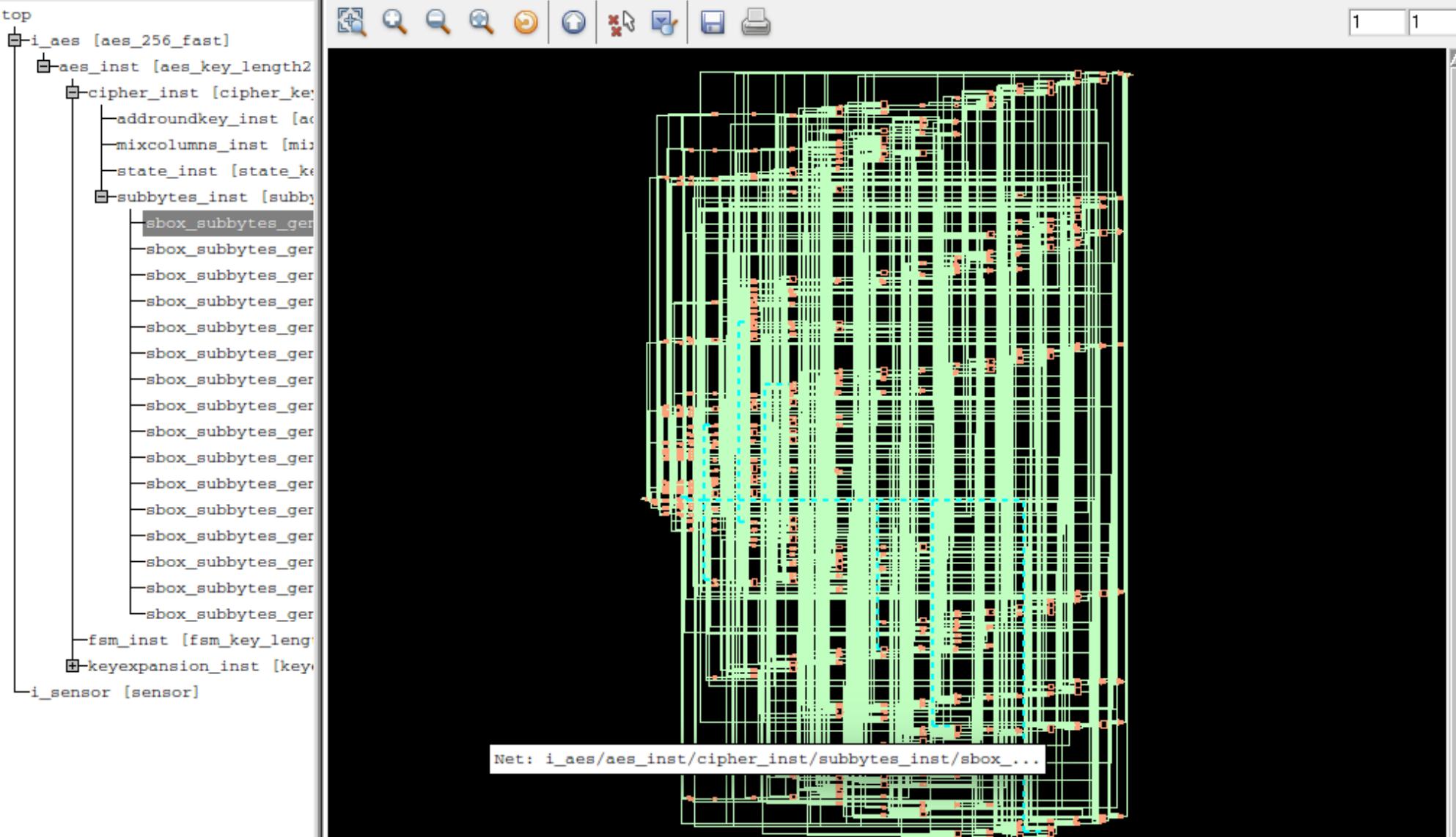
quit
```

## Hierarchy

HDL

Schematic

1 1

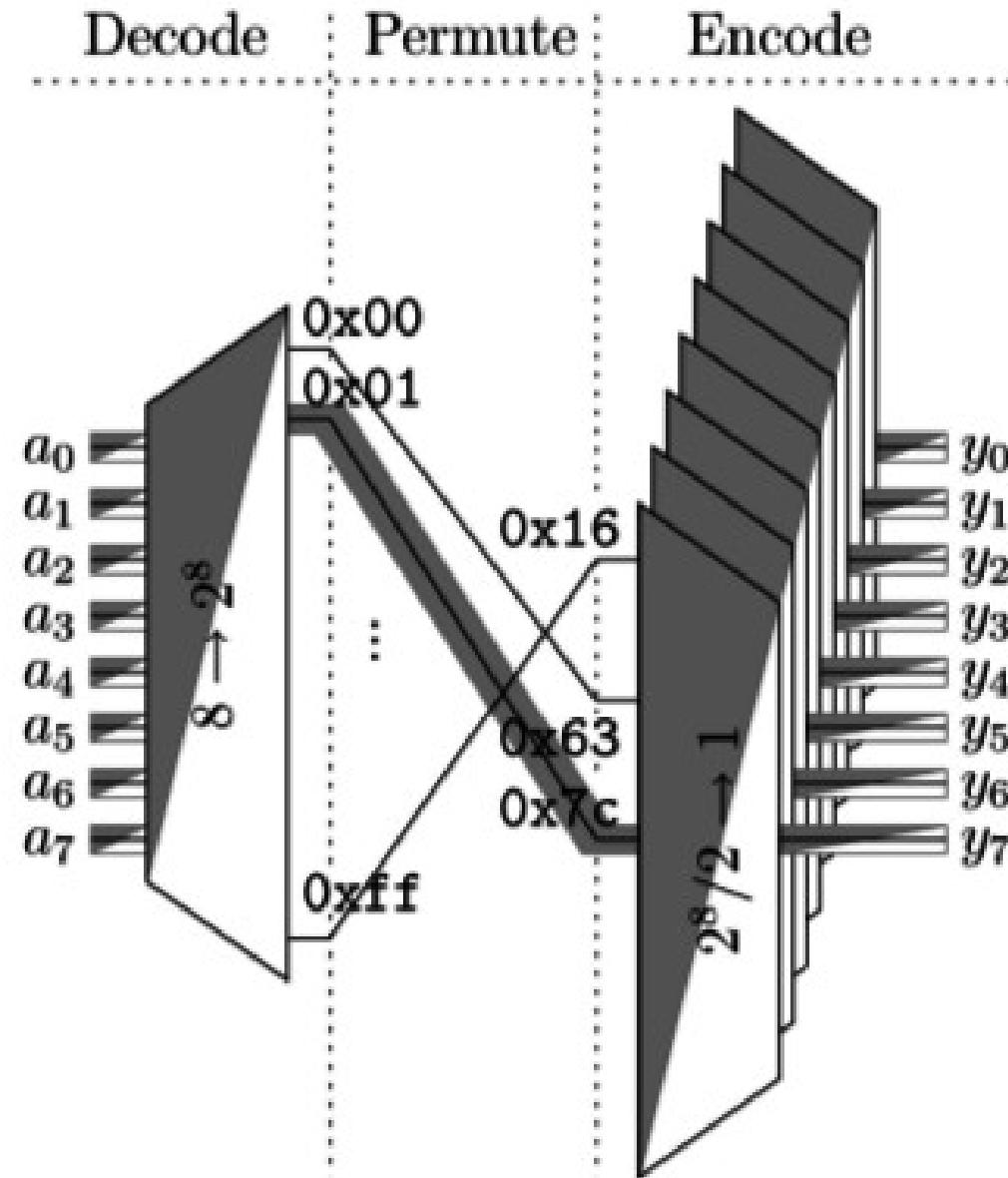


top+i\_aes+aes\_inst+cipher\_inst+subbytes\_inst+sbox\_subbytes\_gen[1].sbox\_subbytes\_inst 1

auto update: 2000

Design is mapped

# Decode – Permute - Encode



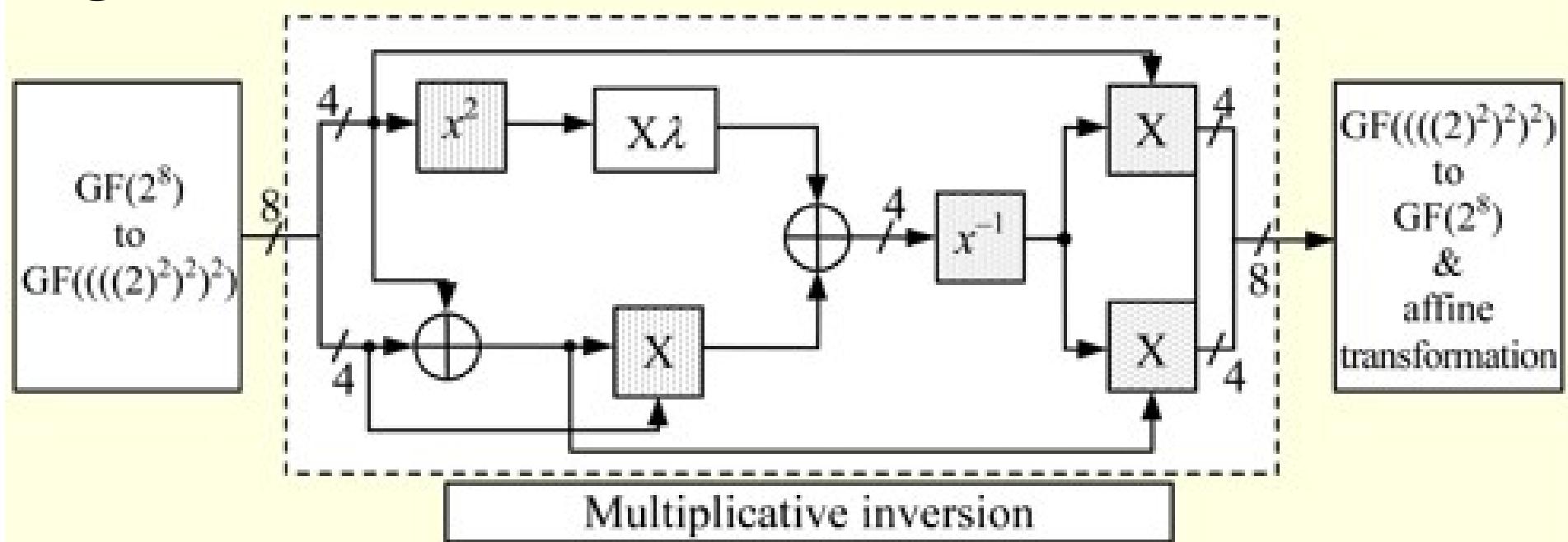
In the polynomial representation, multiplication in  $GF(2^8)$  (denoted by  $\bullet$ ) corresponds with the multiplication of polynomials modulo an **irreducible polynomial** of degree 8. A polynomial is irreducible if its only divisors are one and itself. **For the AES algorithm, this irreducible polynomial is**

$$m(x) = x^8 + x^4 + x^3 + x + 1, \quad (4.1)$$

Sbox :  $a \rightarrow b = a^{-1}$ , then  $b \rightarrow b'$

## Algebraic structure

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$



# AES

- Perform operation AddRoundKey, which XORs the round key with the state.
- For each of the N rounds:
  - perform operation ByteSub (a substitution using an S-box)
  - perform operation ShiftRow (a permutation)
  - perform operation MixColumn (unless it is the last round)
  - perform AddRoundKey.

# Conclusion for AES

Architecture \ Critère	Taille	Energie	Temps
Table			x
Decode – Permute – Encode		x	
Algèbre	x		

# Pipeline de processeurs

- Example on 6502 processor (8 bit) :
  - download here : <https://github.com/chenxiao07/vhdl-nes/tree/master>
  - read in file vhdl-nes-master/src/free6502.vhd lines 765 to 772

```
765      -- The ALU adder
          alu_add <= alu_in1 + alu_add_in2 + alu_add_cin;

          -- The ALU itself  This is purely combinatorial logic
process (alu_add, alu_in1, alu_in2, alu_op, c_flag)
begin
    -- default for alu_add inputs
    alu_add_in2 <= alu_in2;
    alu_add_cin <= c_flag;

    case alu_op is
        -- [...]
        when MC_BIT_AND => alu_out <= alu_in1 and alu_in2;
        when MC_BIT_OR  => alu_out <= alu_in1 or  alu_in2;
        when MC_BIT_XOR => alu_out <= alu_in1 xor alu_in2;
        when MC_BIT_ASL => alu_out <= alu_in1(7 downto 0) & "0";
        when MC_BIT_LSR => alu_out <= alu_in1(0) & "0" & alu_in1(7 downto 1);
        when MC_BIT_ROL => alu_out <= alu_in1(7 downto 0) & c_flag;
        when MC_BIT_ROR => alu_out <= alu_in1(0) & c_flag & alu_in1(7 downto 1);
        when others       => alu_out <= alu_in1;
    end case;
end process;
772
```

# Pipeline de processeurs

Example on LEON processor (32 bit) :

see gaisler/leon3v3/iu3.vhd

```
procedure logic_op(r : registers; aluin1, aluin2, mey : word;
                   ymsb : std_ulogic; logicres, y : out word) is
variable logicout : word;
begin
  case r.e.aluop is
    when EXE_AND    => logicout := aluin1 and aluin2;
    when EXE_ANDN   => logicout := aluin1 and not aluin2;
    when EXE_OR     => logicout := aluin1 or aluin2;
    when EXE_ORN    => logicout := aluin1 or not aluin2;
    when EXE_XOR    => logicout := aluin1 xor aluin2;
    when EXE_XNOR   => logicout := aluin1 xor not aluin2;
    when EXE_DIV    =>
      if DIVEN then logicout := aluin2;
      else logicout := (others => '-'); end if;
    when others => logicout := (others => '-');
  end case;
  if (r.e.ctrl.wy and r.e.mulstep) = '1' then
    y := ymsb & r.m.y(31 downto 1);
  elsif r.e.ctrl.wy = '1' then y := logicout;
  elsif r.m.ctrl.wy = '1' then y := mey;
  elsif MACPIPE and (r.x.mac = '1') then y := mulo.result(63 downto 32);
  elsif r.x.ctrl.wy = '1' then y := r.x.y;
  else y := r.w.s.y; end if;
  logicres := logicout;
end;
```

# Assistance à la synthèse

- VHDL
  - attribute keep of clock\_signal\_name: signal is "true";
- Verilog
  - // synthesis attribute keep of clock\_signal\_name is true;
- Comme des scripts :
  - set\_dont\_touch [get\_cells I\_cdnluser\_cts/S\_CDNSunderM\_DLY\_1 ]

À Vous :  
<https://www.edaplayground.com/x/4eFR>

<https://digitaljs.tilk.eu/>

# Fibonacci

- $U_0 = 0$
- $U_1 = 1$
- $U_2 = U_1 + U_0$
- $U_3 = U_2 + U_1$
- ...

THE ON-LINE ENCYCLOPEDIA  
OF INTEGER SEQUENCES®

founded in 1964 by N. J. A. Sloane

**Annual Appeal: Please make a [donation](#) (tax deductible in USA) to keep the OEIS running. Over 5000 articles have referenced us, often saying "we discovered this result with the help of the OEIS".**

[Search](#) [Hints](#)

(Greetings from [The On-Line Encyclopedia of Integer Sequences!](#)!)

Search: **seq:0,1,1,2,3,5,8,13**

Displaying 1-10 of 64 results found. page 1 [2](#) [3](#) [4](#) [5](#) [6](#) [7](#)

Sort: relevance | [references](#) | [number](#) | [modified](#) | [created](#)      Format: long | [short](#) | [data](#)

**A000045**    Fibonacci numbers:  $F(n) = F(n-1) + F(n-2)$  with  $F(0) = 0$  and  $F(1) = 1$ . +20  
3752

(Formerly M0692 N0256)

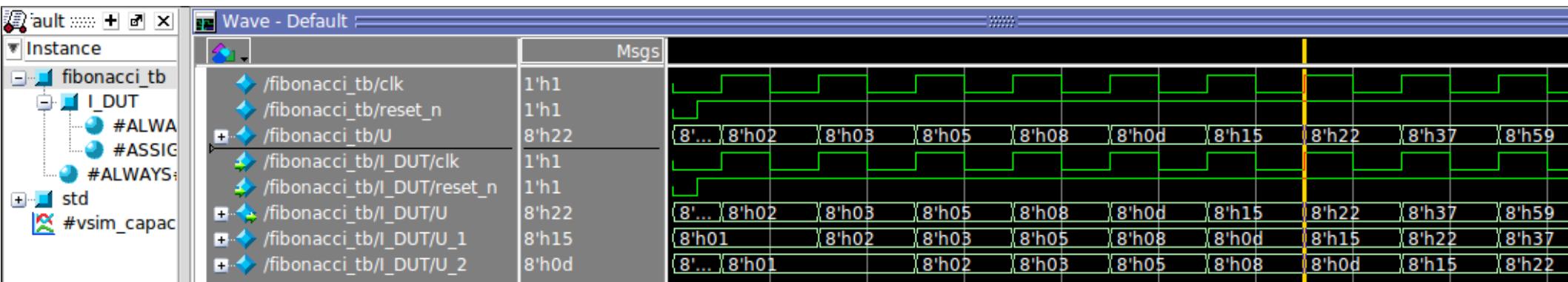
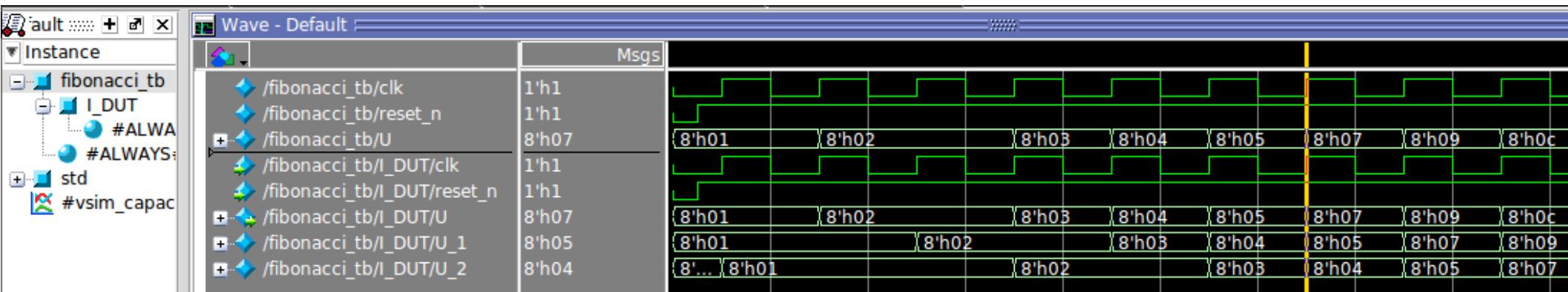
**0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887, 9227465, 14930352, 24157817, 39088169** ([list](#); [graph](#); [refs](#); [listen](#); [history](#); [text](#); [internal format](#))

OFFSET      0,4

COMMENTS     Also sometimes called Lamé's sequence.  
 $F(n+2)$  = number of binary sequences of length  $n$  that have no consecutive 0's.  
 $F(n+2)$  = number of subsets of  $\{1, 2, \dots, n\}$  that contain no consecutive integers.  
 $F(n+1)$  = number of tilings of a  $2 \times n$  rectangle by  $2 \times 1$  dominoes.  
 $F(n+1)$  = number of matchings (i.e., Hosoya index) in a path graph on  $n$  vertices:  
     $F(5)=5$  because the matchings of the path graph on the vertices A, B, C, D are the empty set, {AB}, {BC}, {CD} and {AB, CD}. - [Emeric Deutsch](#), Jun 18 2001  
 $F(n)$  = number of compositions of  $n+1$  with no part equal to 1. [Cayley, Grimaldi]  
Positive terms are the solutions to  $z = 2*x*y^4 + (x^2)*y^3 - 2*(x^3)*y^2 - y^5 - (x^4)*y + 2*y$  for  $x, y \geq 0$  (Ribenboim, page 193). When  $x=F(n)$ ,  $y=F(n+1)$  and  $z>0$  then  $z=F(n+1)$ .

# Simulation

- `vlog fibonacci.sv fibonacci_tb.sv`
- `vsim fibonacci_tb`



# Fibonacci en SystemVerilog

```
module fibonacci(clk,
                  reset_n,
                  U);

  input logic      clk;
  input logic      reset_n;
  output logic [7:0] U;

  // Un-1 (= Un au cycle précédent)
  logic [7:0]      U_1;

  // Un-2 (= Un-1 au cycle précédent)
  logic [7:0]      U_2;

  always @(posedge clk or negedge reset_n)
    if(reset_n==0)
      begin
        U  <= 1;
        U_1 <= 1;
        U_2 <= 0;
      end
    else
      begin
        // Un = Un-1 + Un-2
        U    <= U_1 + U_2;
        // Un-1 = Un retardé d'un cycle
        U_1 <= U;
        // Un-2 = Un-1 retardé d'un cycle
        U_2 <= U_1;
      end
  endmodule
```

# Exercice d'arithmétique

```
#include <stdint.h>

uint32_t guess( uint32_t x, uint32_t y )
{
    return ( x & y ) + (( x ^ y )>>1);
}
```

# Exercice d'arithmétique

## Lemmas :

- $(x+y) = (x \mid y) + (x \And y)$
- $(x \oplus y) = (x \mid y) - (x \And y)$

# Exercice d'arithmétique

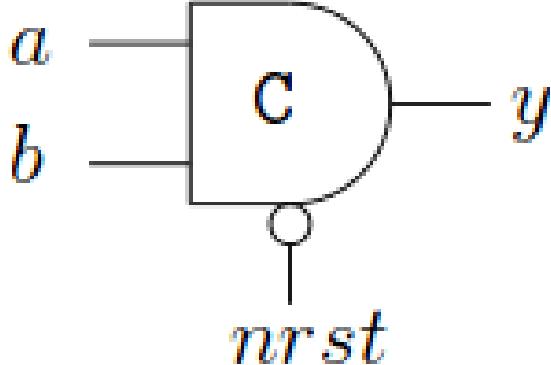
## Lemmas :

Otherwise, consider  
 $(a>>1) + (b>>1) + ((a\%2 + b\%2)>>1)$

- $(x+y) = (x \mid y) + (x \And y)$
- $(x \oplus y) = (x \mid y) - (x \And y)$

# C-Element : comportemental

**TAB. 1. Le C-Element à deux entrées accompagné de son comportement décrit en VHDL.**

	<pre>if (nrst = 0) then     y &lt;= 0 ; elsif (a = b) then     y &lt;= a ; end if ;</pre>
--	---

# C-Element : structures en transistors

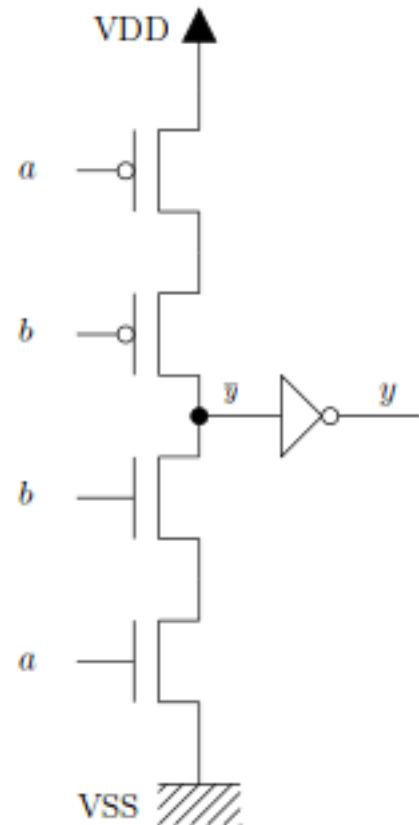


FIG. 1a. C-Element dynamique.

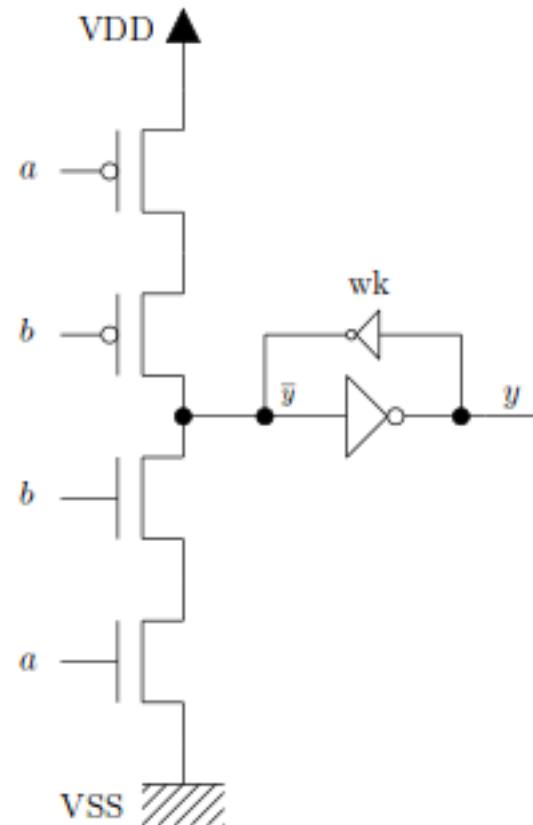
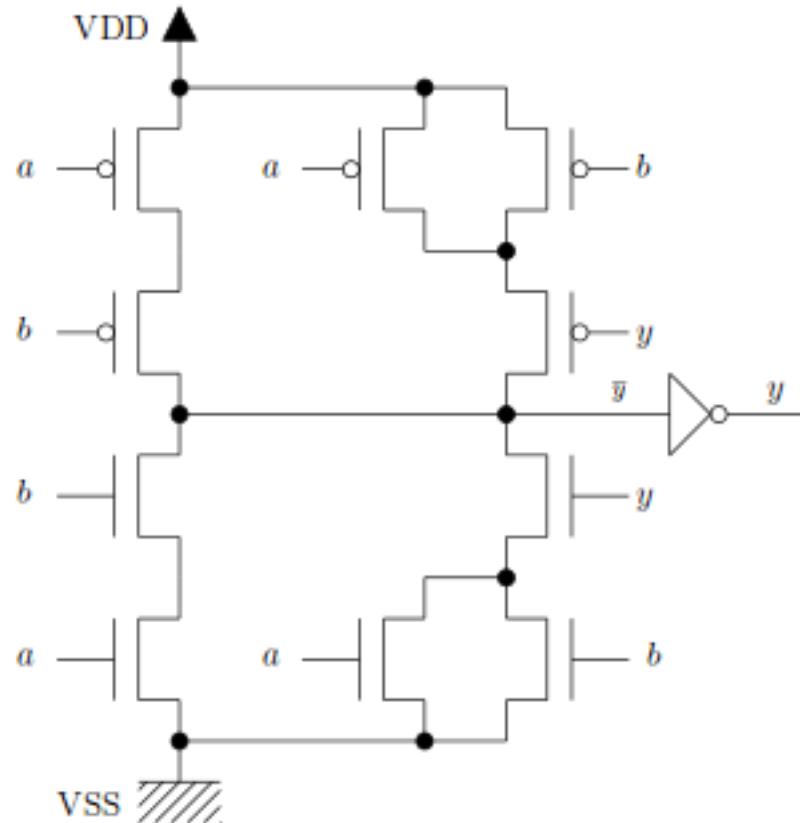
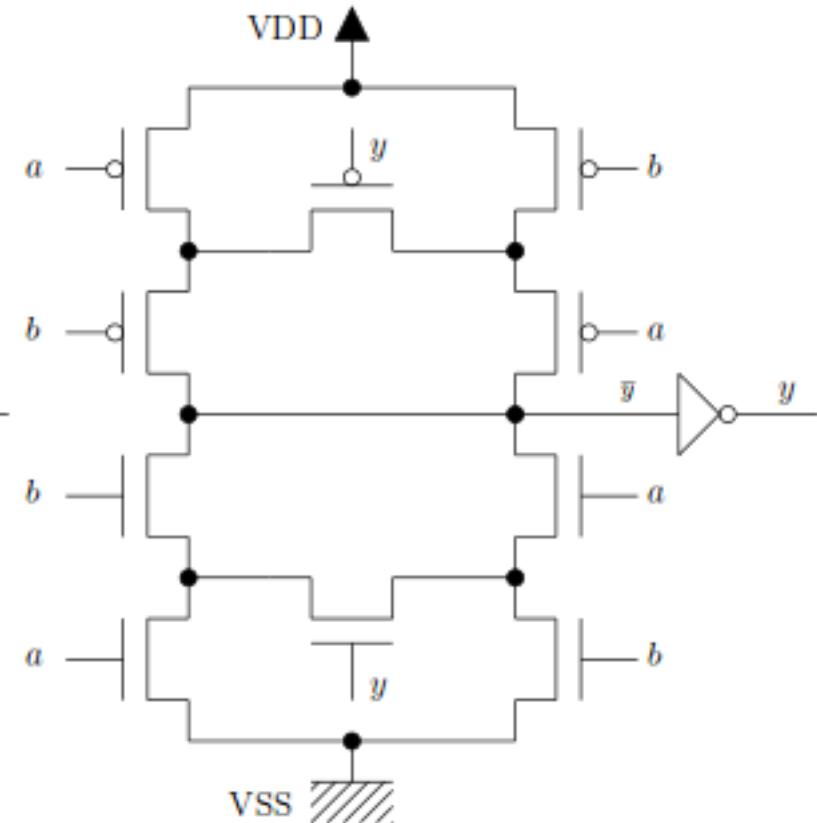


FIG. 1b. C-Element à rebouclage faible.

# C-Element : structures en transistors



**FIG. 1c. C-Element conventionnel.**



**FIG. 1d. C-Element symétrique.**

# C-Element : structures en transistors

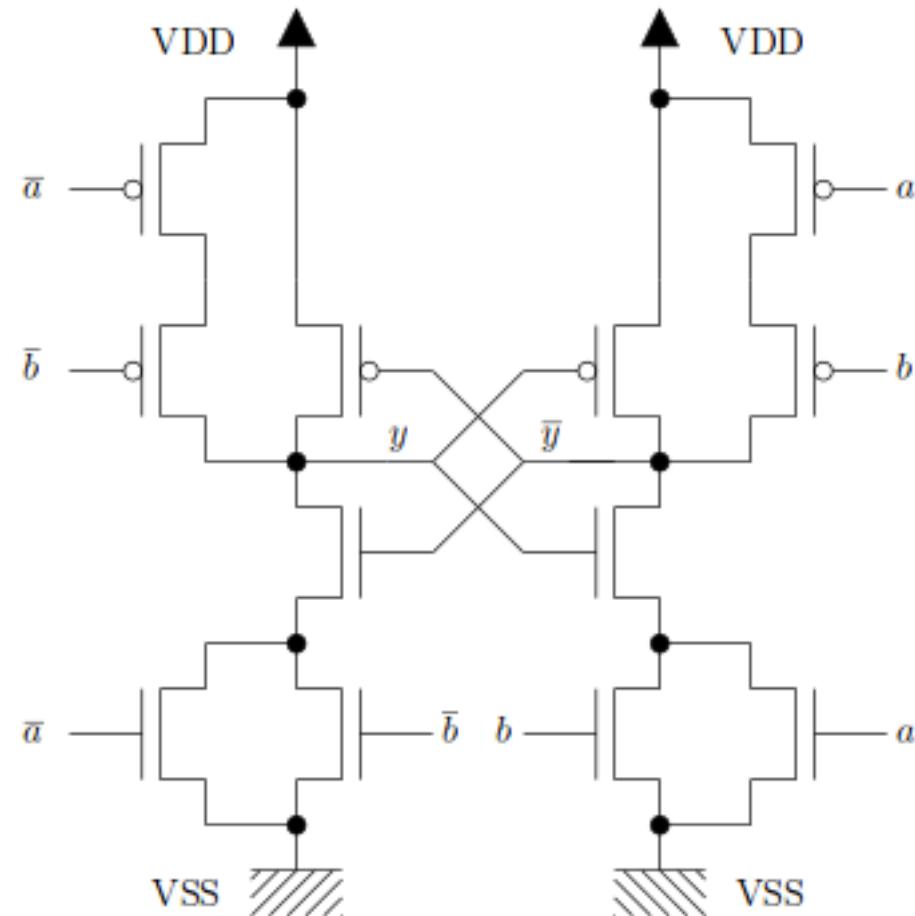
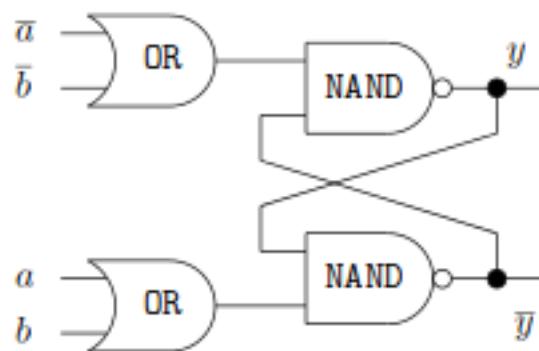


FIG. 2a. Structure en portes du C-Element à bascule RS.

FIG. 2b. Structure en transistors du C-Element à bascule RS de la Fig. 2a.