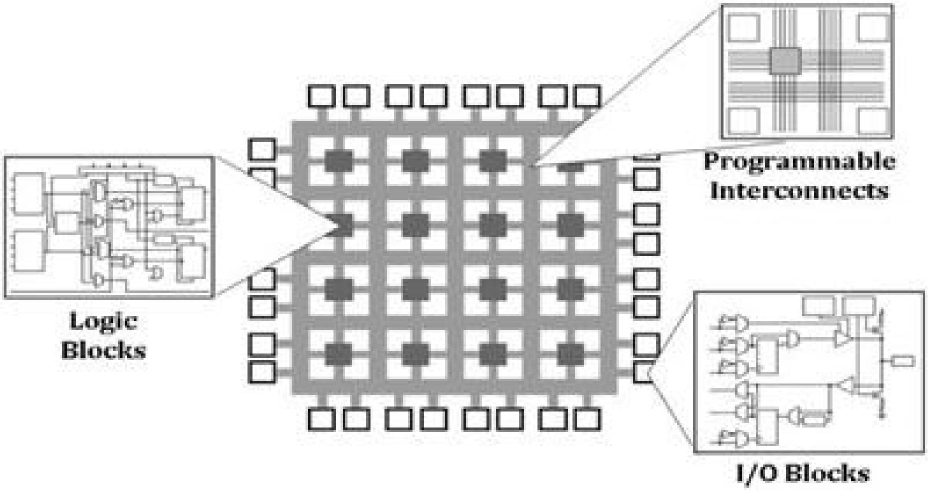
Digital SoC Design

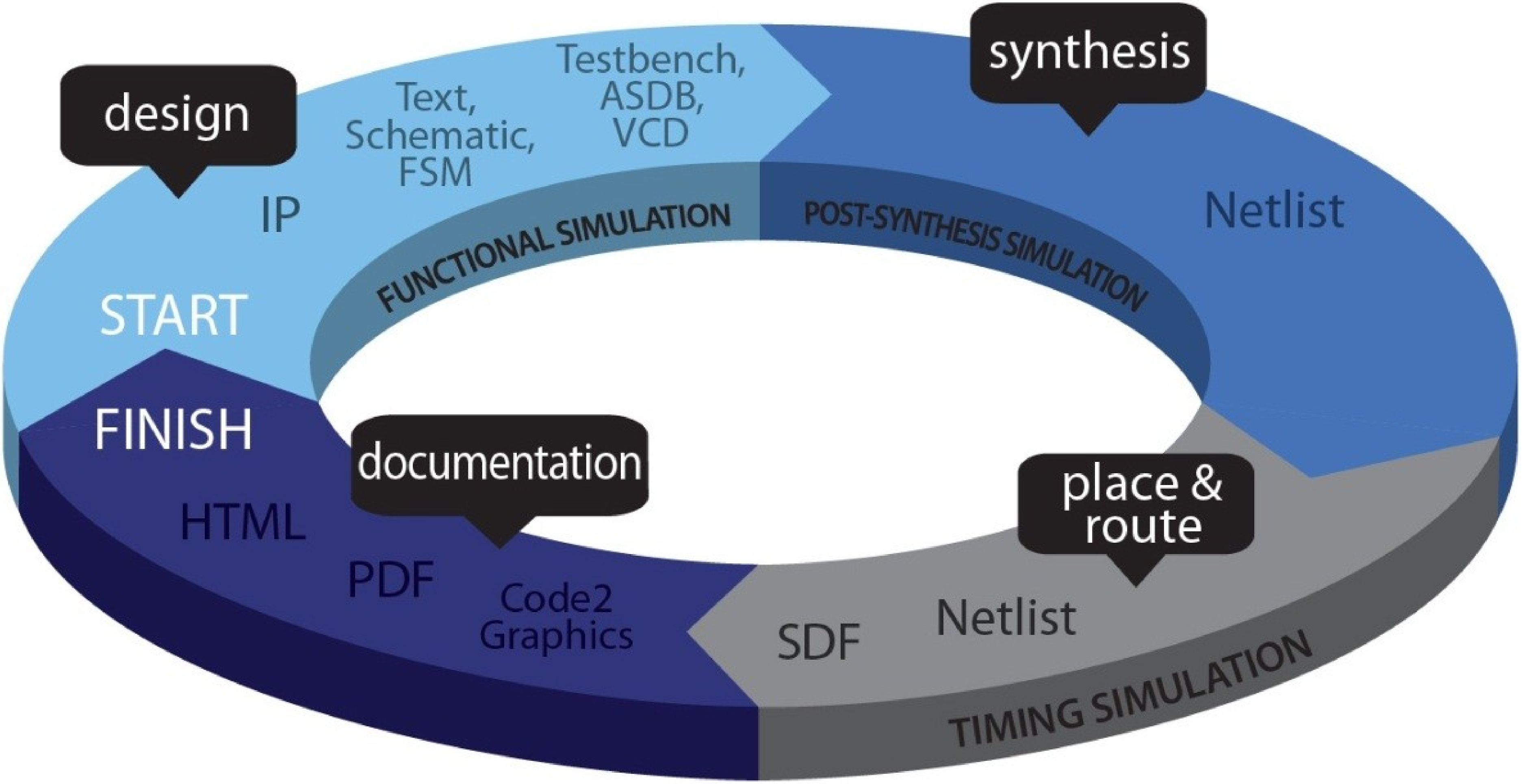
Travaux Pratiques

Sumanta Chaudhuri

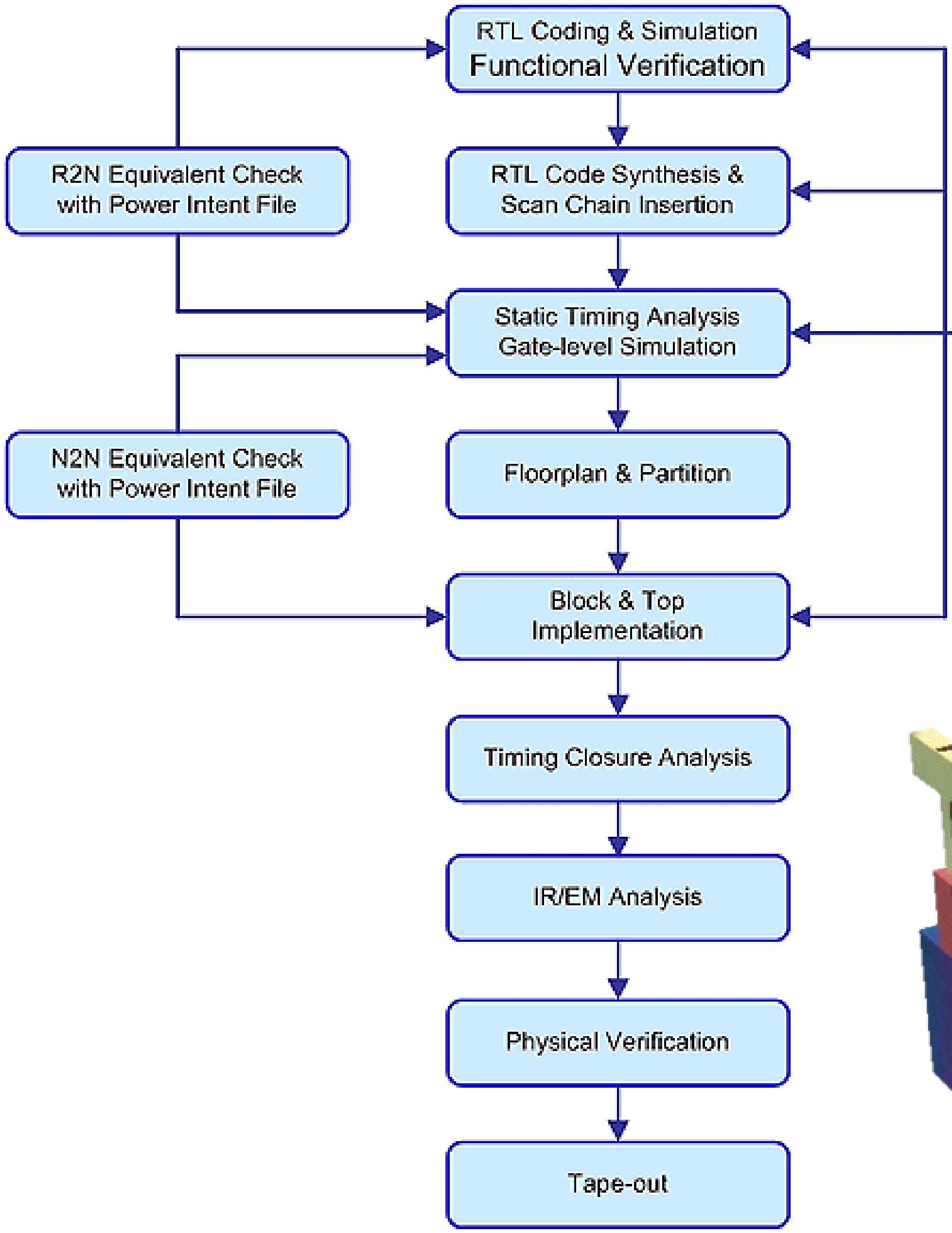
EDA



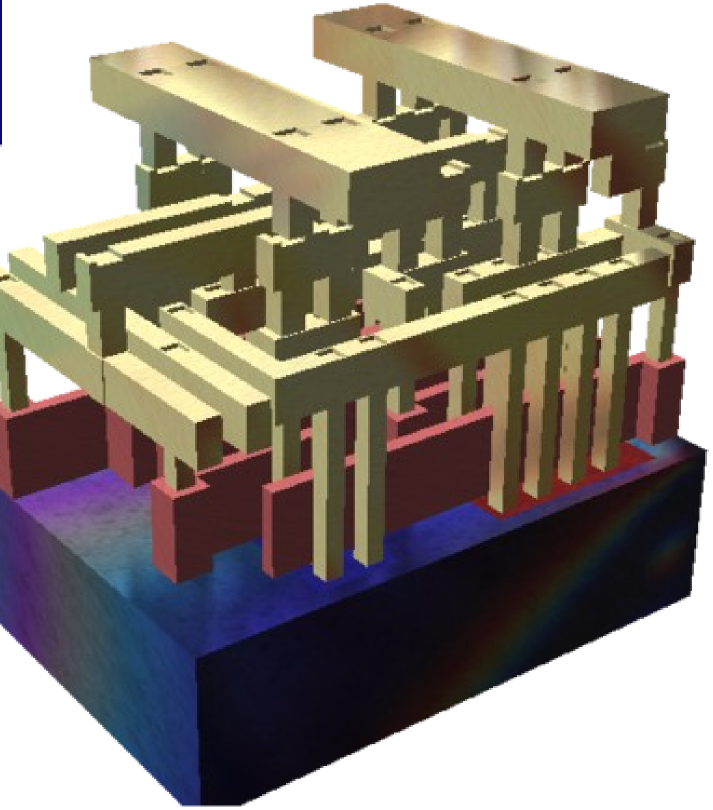
FPGA Design



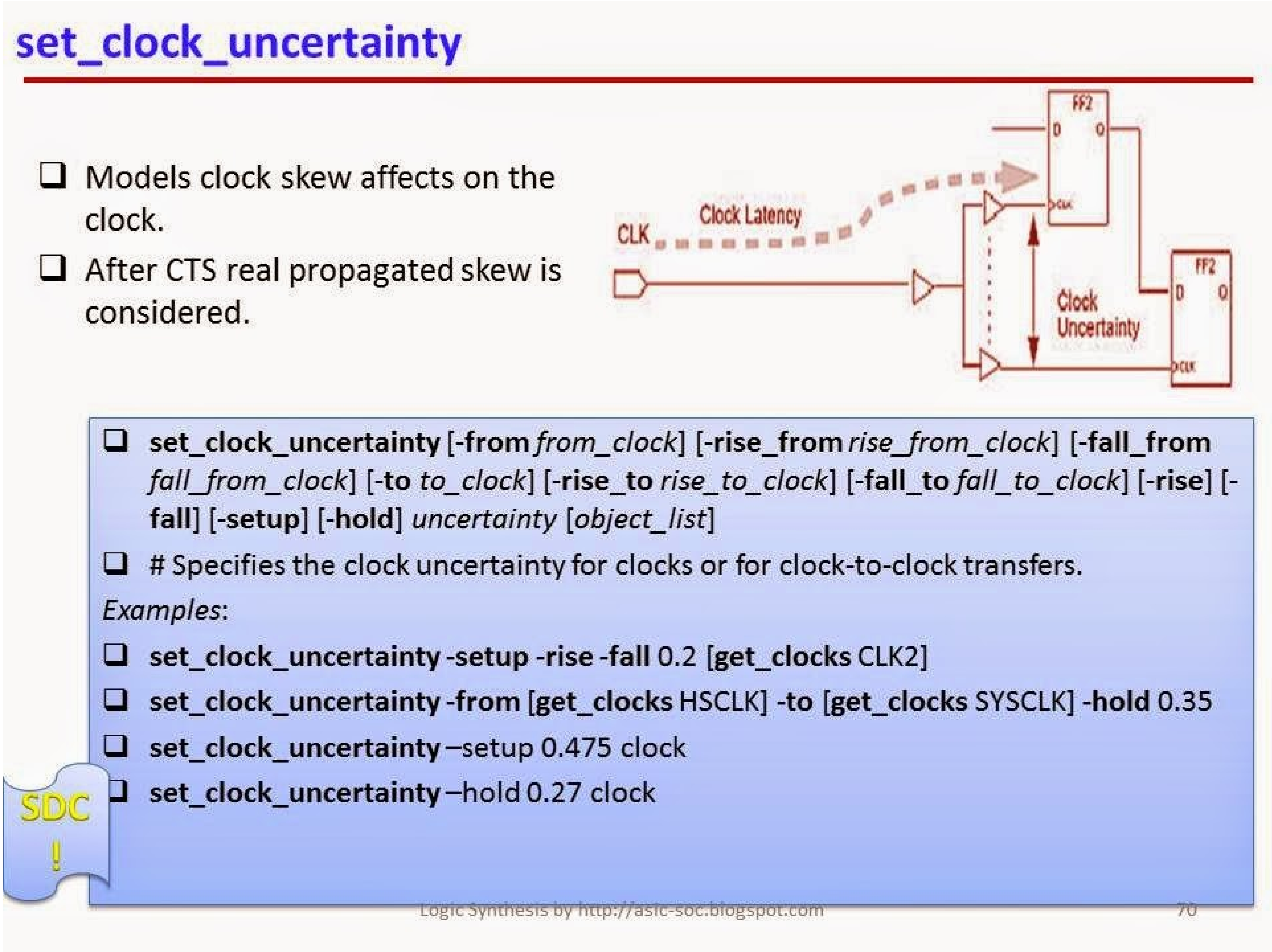
EDA



RTL Coding & Simulation
Functional Verification
R2N Equivalent Check
with Power Intent File
RTL Code Synthesis &
Scan Chain Insertion
Static Timing Analysis
Gate-level Simulation
Power Intent File
N2N Equivalent Check
with Power Intent File
Floorplan & Partition
Block & Top
Implementation
Timing Closure Analysis
IR/EM Analysis
Physical Verification
Tape-out



Commands



set_clock_uncertainty

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

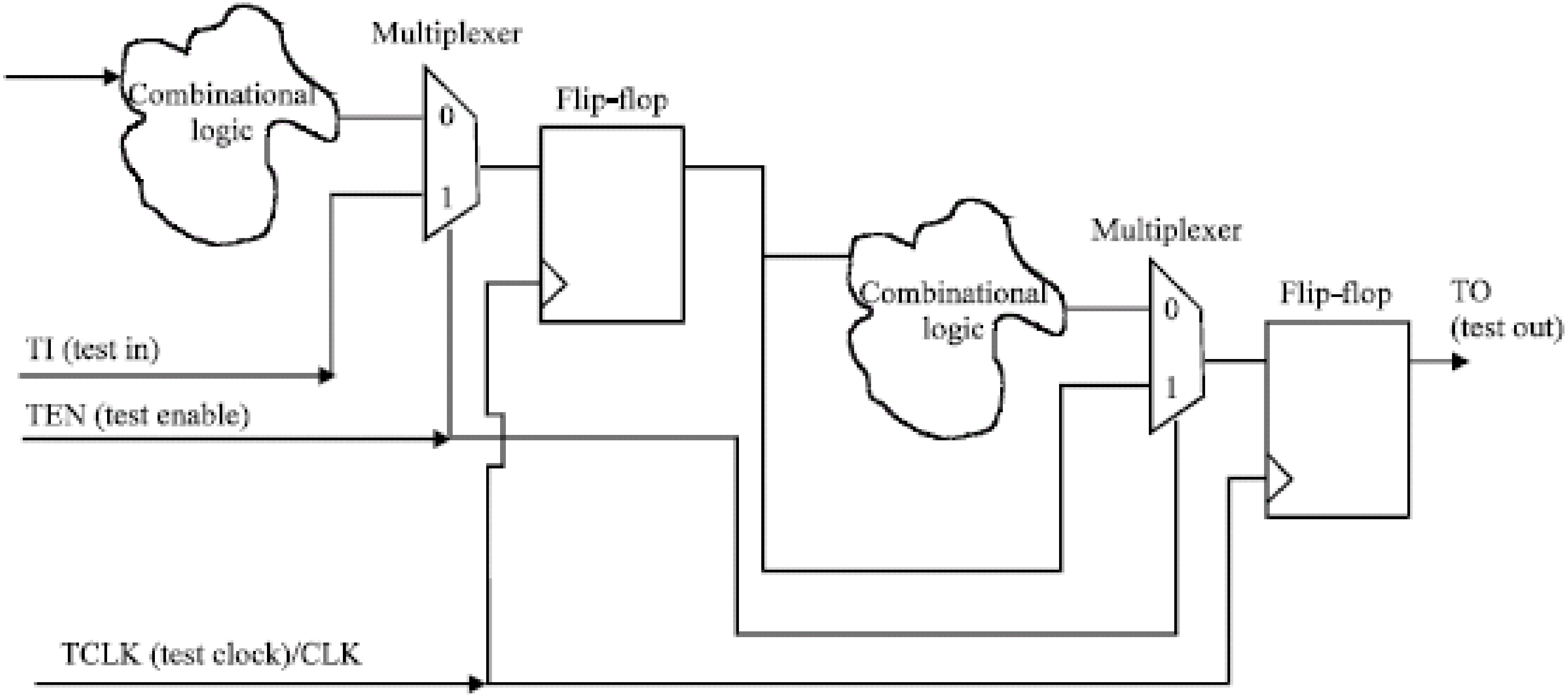
- ❑ **set_clock_uncertainty** [**-from** *from_clock*] [**-rise_from** *rise_from_clock*] [**-fall_from** *fall_from_clock*] [**-to** *to_clock*] [**-rise_to** *rise_to_clock*] [**-fall_to** *fall_to_clock*] [**-rise**] [**-fall**] [**-setup**] [**-hold**] *uncertainty* [*object_list*]
- ❑ # Specifies 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

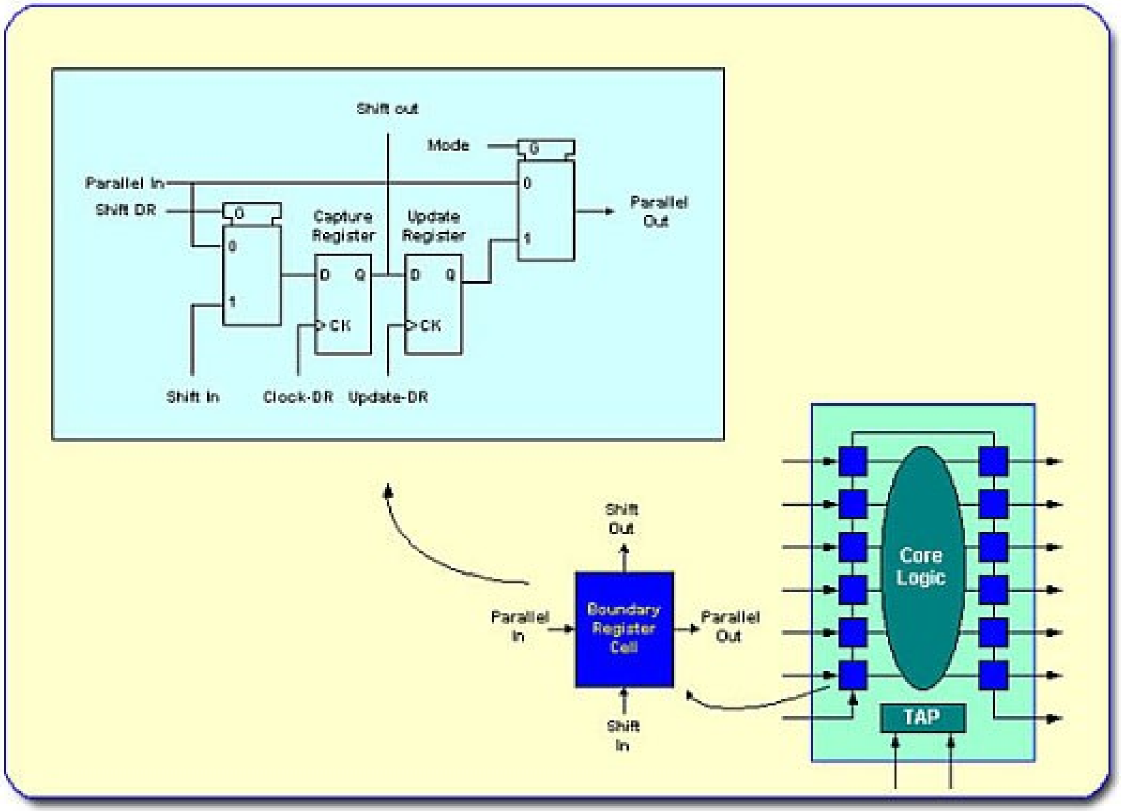
Test des circuits intégrés

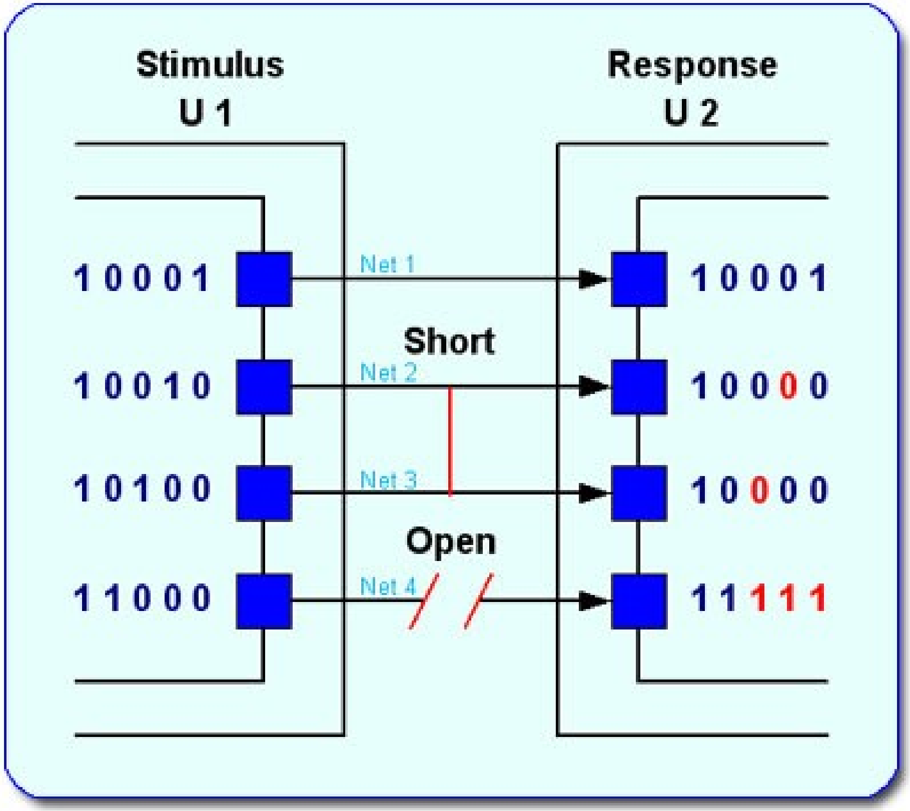
Deux modes, en fonction de TEN



Test des cartes électroniques

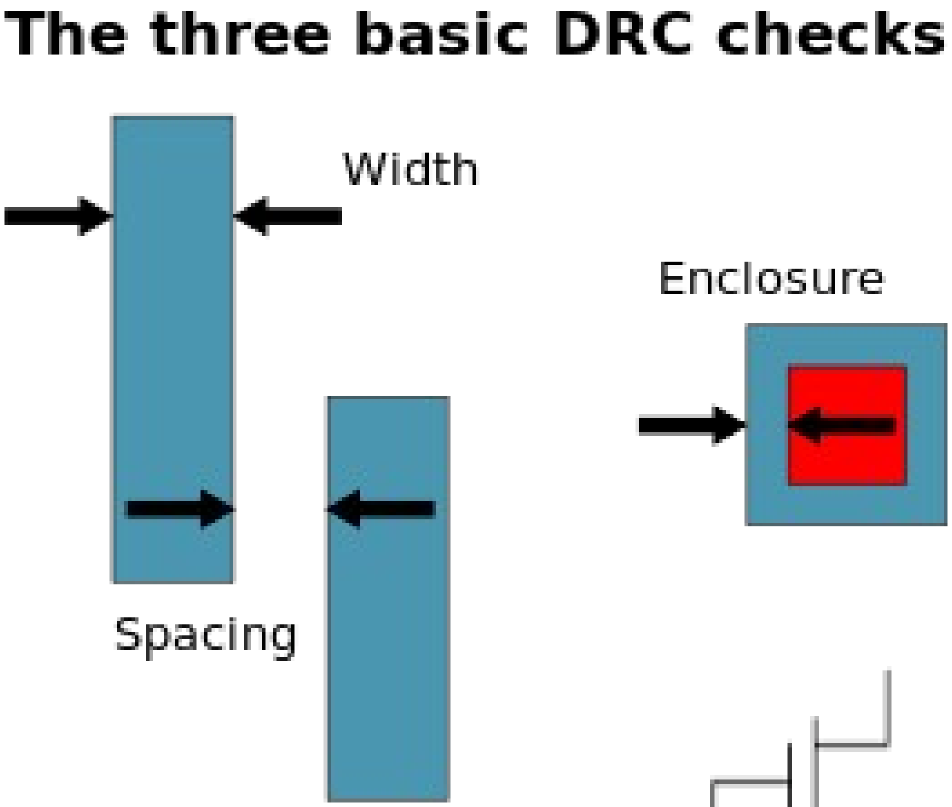
Boundary-scan, défini dans le standard IEEE Std.-1149.1

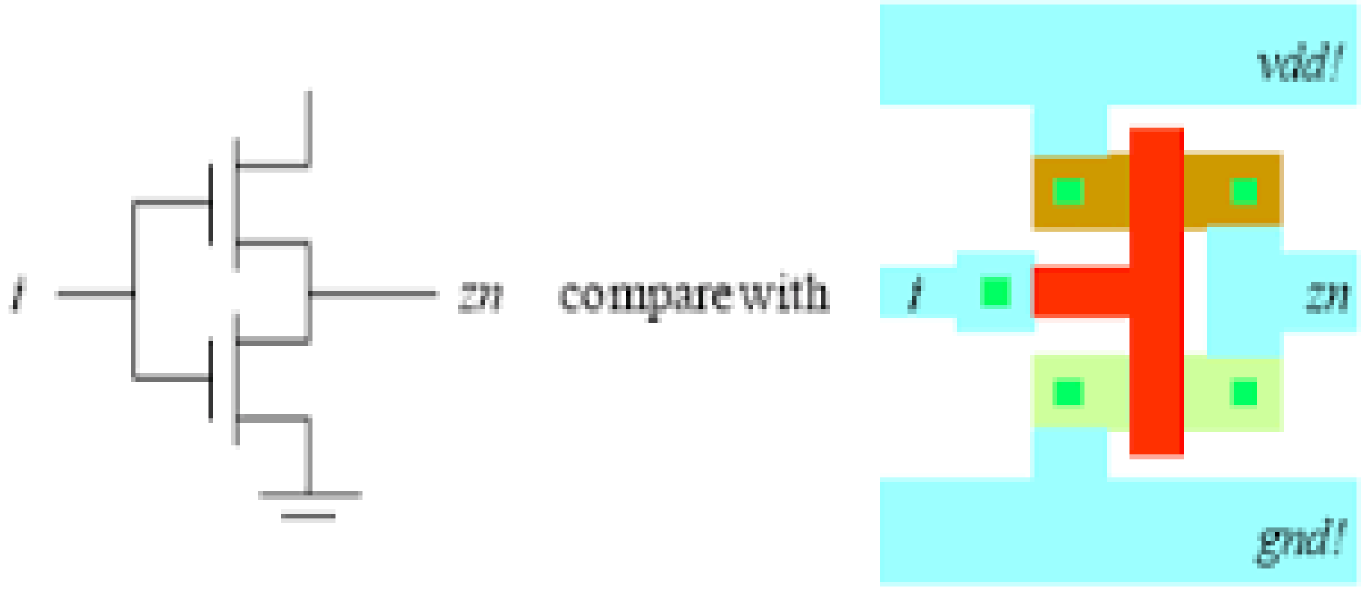




Vérifications finales

DRC : Design Rules Check
LVS : Layout Versus Schematic





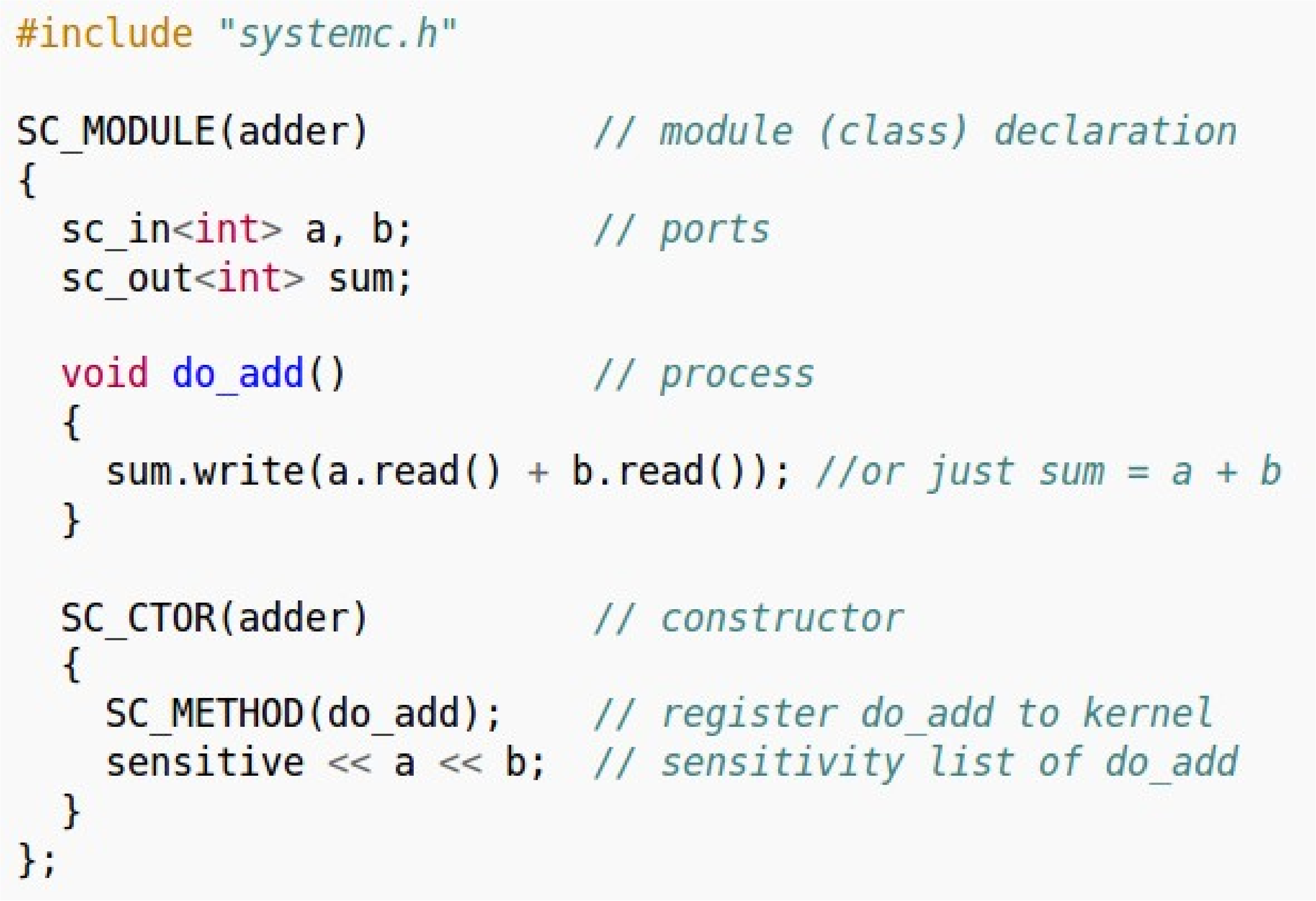
Niveaux de description

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

Langages

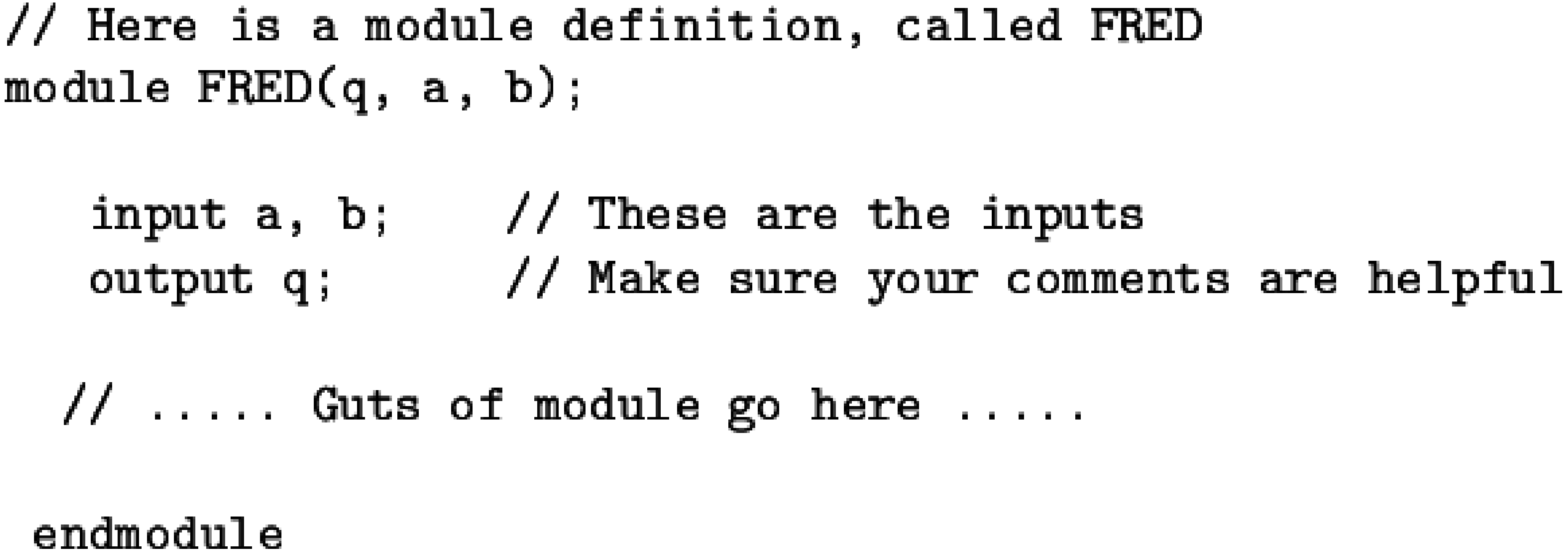
- System-C (IEEE 1666)
  - Testbench
  - Dimensionnement de performances
- Verilog (IEEE 1076), VHDL (IEEE 1364)
  - Simulation + synthèse
  - Netlist

SystemC

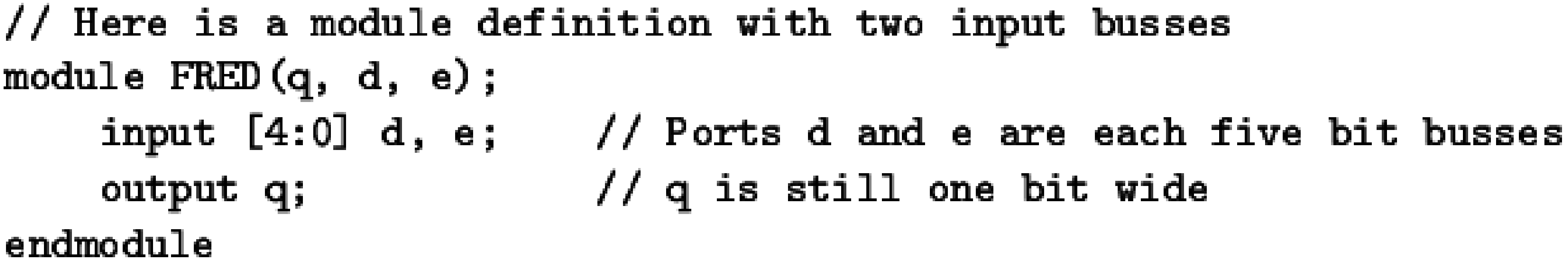


```
#include "systemc.h"

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

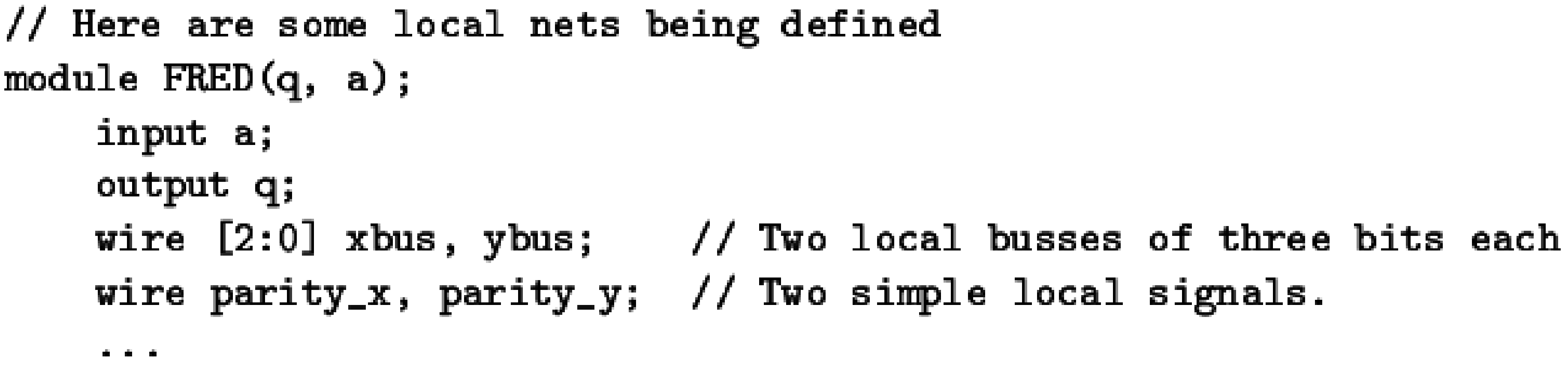


```
// Here is a module definition, called FRED
module FRED(q, a, b);

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

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

 endmodule
```

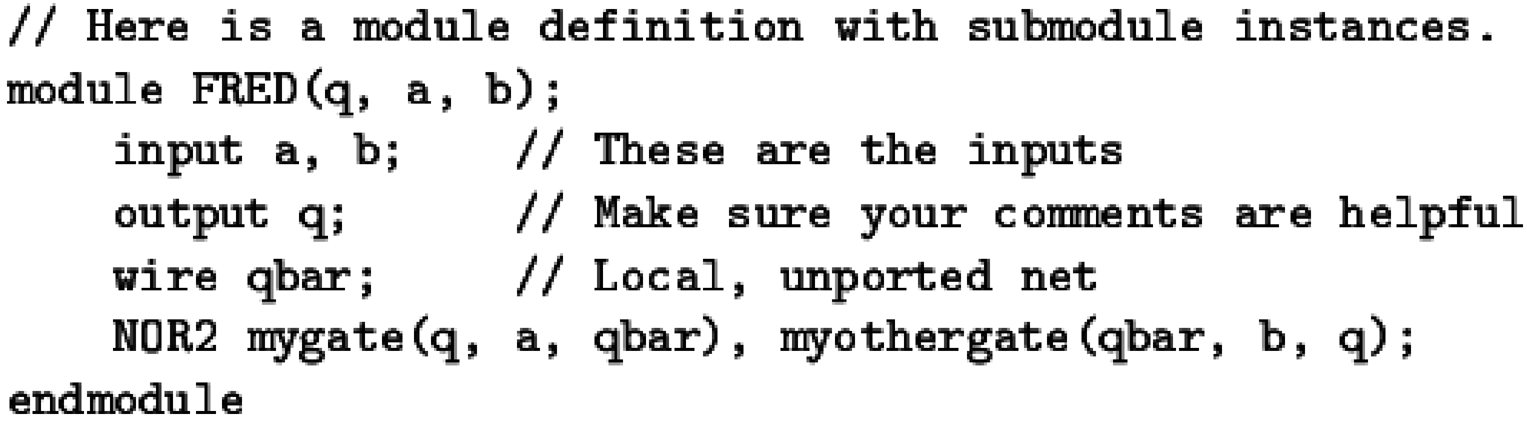


```
// Here is a module definition with two input busses
module FRED(q, d, e);
    input [4:0] d, e;    // Ports d and e are each five bit busses
    output q;            // q is still one bit wide
endmodule
```

Syntaxe Verilog

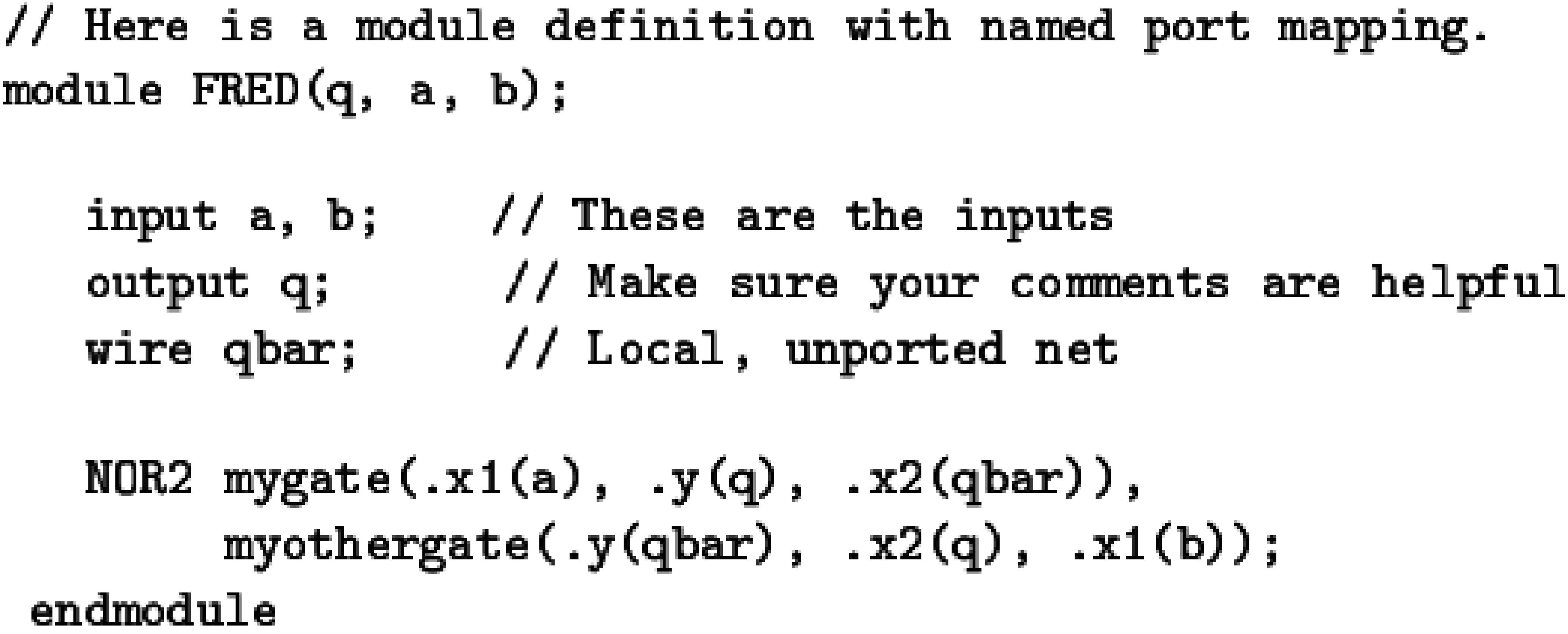


```
// Here are some local nets being defined
module FRED(q, a);
    input a;
    output q;
    wire [2:0] xbus, ybus;    // Two local busses of three bits each
    wire parity_x, parity_y;  // Two simple local signals.
    ...
```



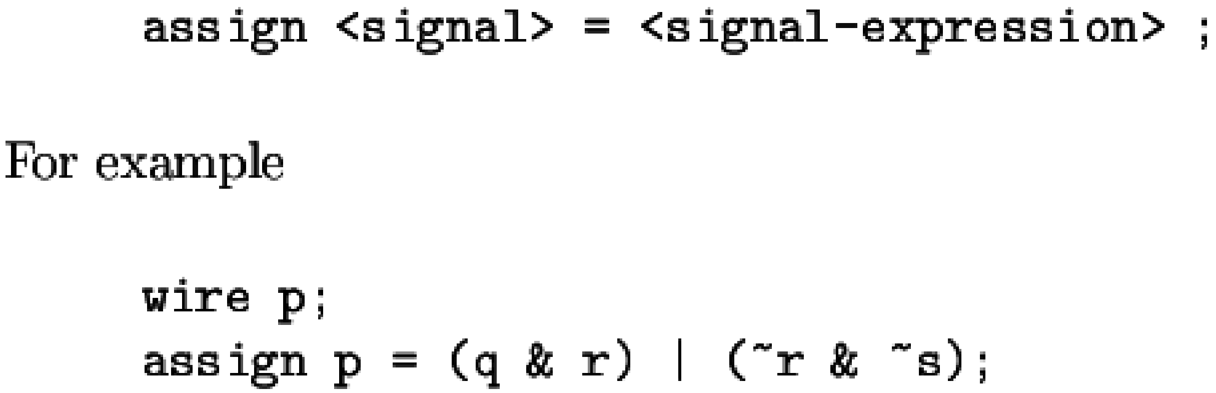
```
// Here is a module definition with submodule instances.
module FRED(q, a, b);
    input a, b;     // These are the inputs
    output q;       // Make sure your comments are helpful
    wire qbar;      // Local, unported net
    NOR2 mygate(q, a, qbar), myothergate(qbar, b, q);
endmodule
```

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



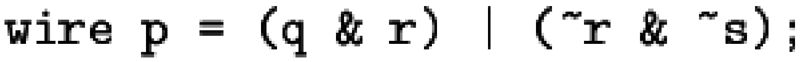
```
assign <signal> = <signal-expression> ;
```

For example

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

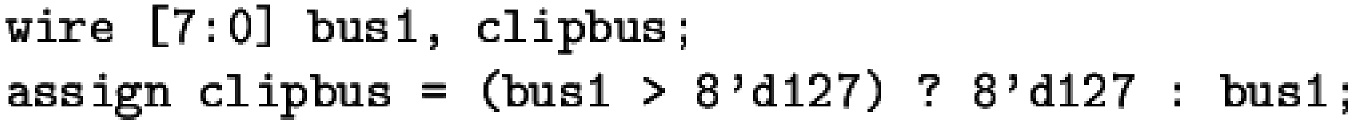
Il y a 123 mots réservés
dans verilog !

Syntaxic sugar :

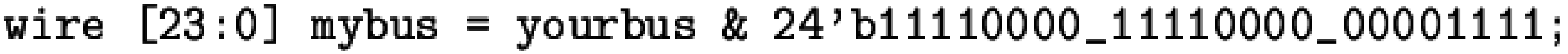


```
wire p = (q & r) | (~r & ~s);
```

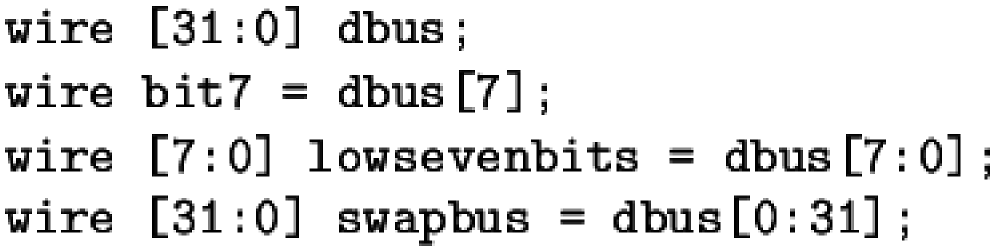
Syntaxe Verilog



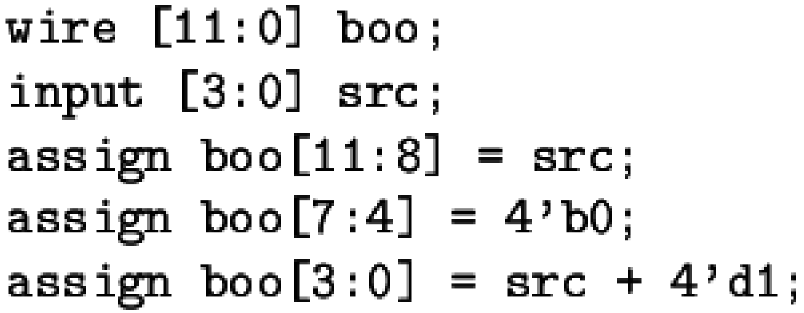
```
wire [7:0] bus1, clipbus;
assign clipbus = (bus1 > 8’d127) ? 8’d127 : bus1;
```



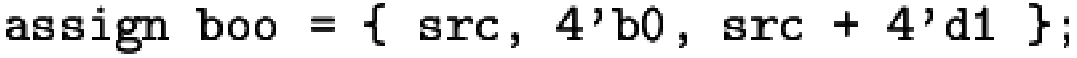
```
wire [23:0] mybus = yourbus & 24’b11110000_11110000_00001111;
```



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

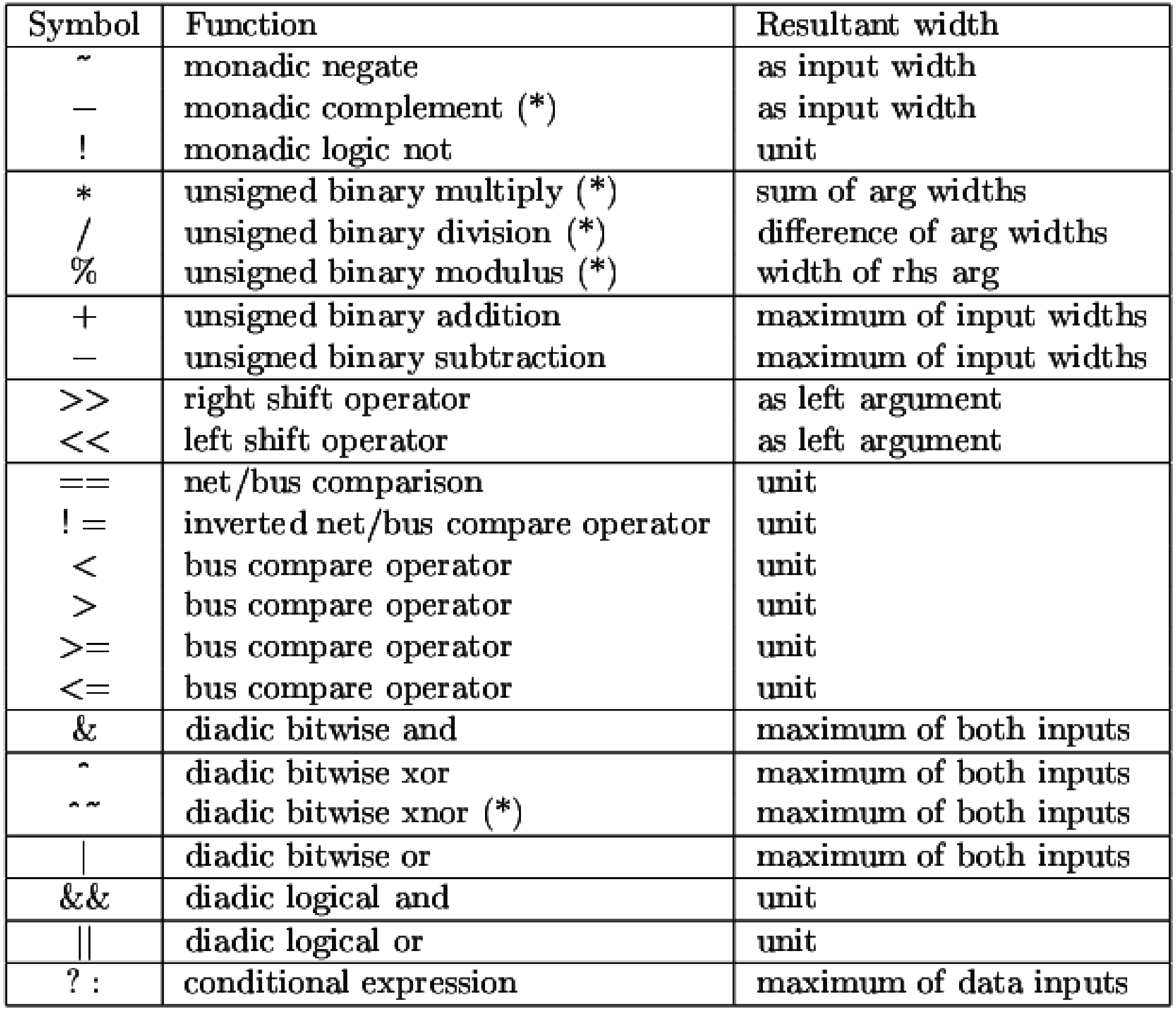


```
wire [11:0] boo;
input [3:0] src;
assign boo[11:8] = src;
assign boo[7:4] = 4’b0;
assign boo[3:0] = src + 4’d1;
```

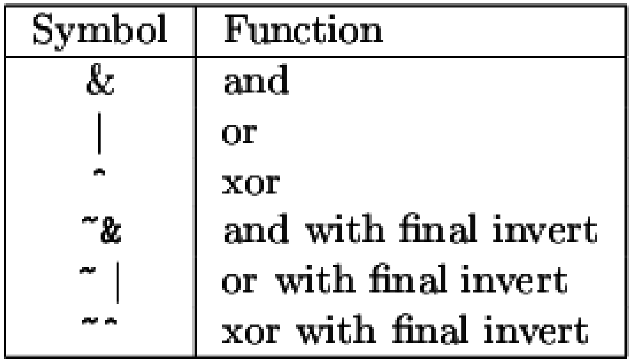


```
assign boo = { src, 4’b0, src + 4’d1 };
```

Syntaxe Verilog



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



Symbol	Function
&	and
|	or
^	xor
~&	and with final invert
~|	or with final invert
~^	xor with final invert

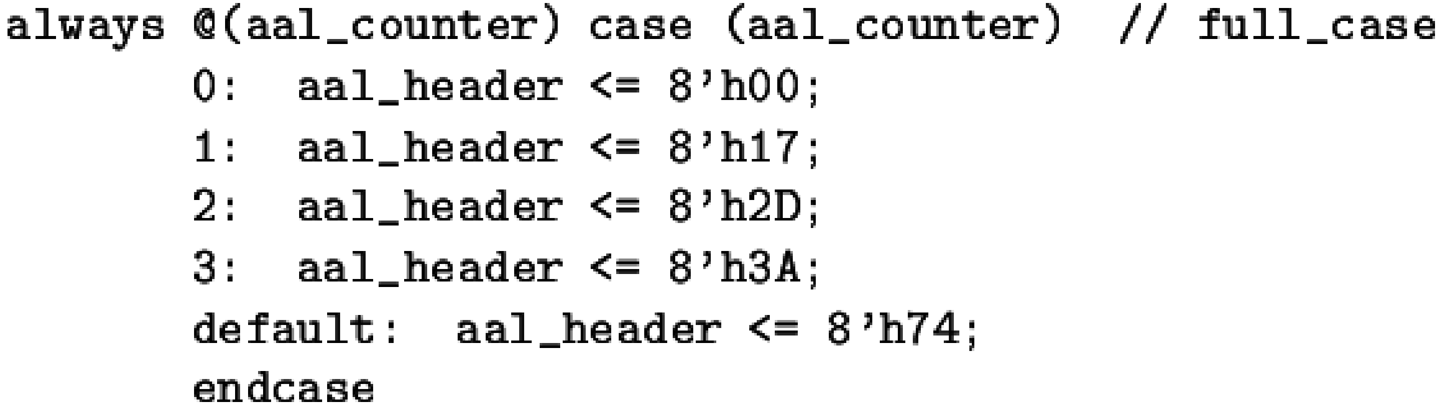
```
wire [9:0] mybus;
wire x = (& (mybus));
```

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

Ternaires :

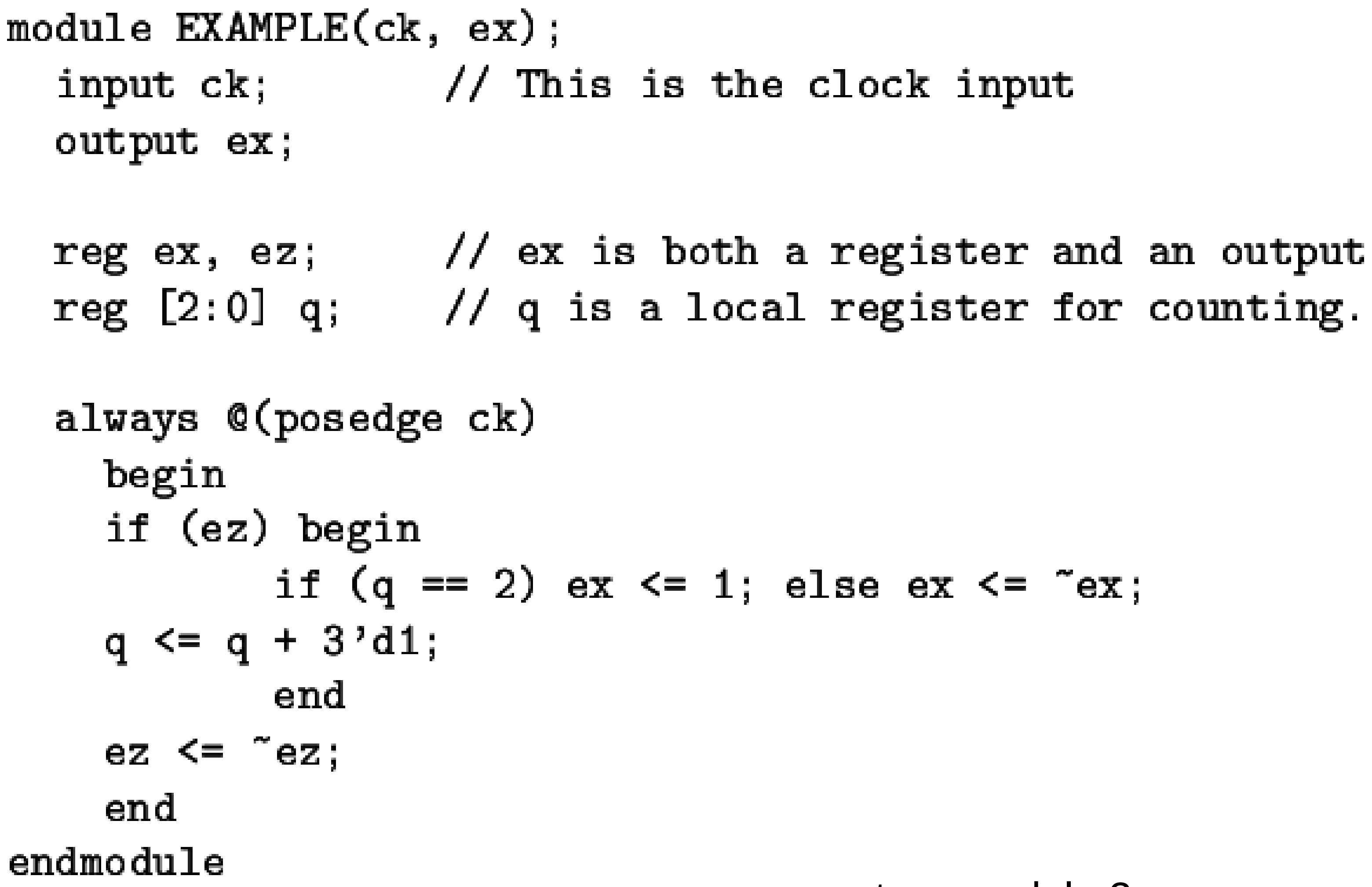
```
wire [7:0] p, q, r;
wire s0, s1;

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
```

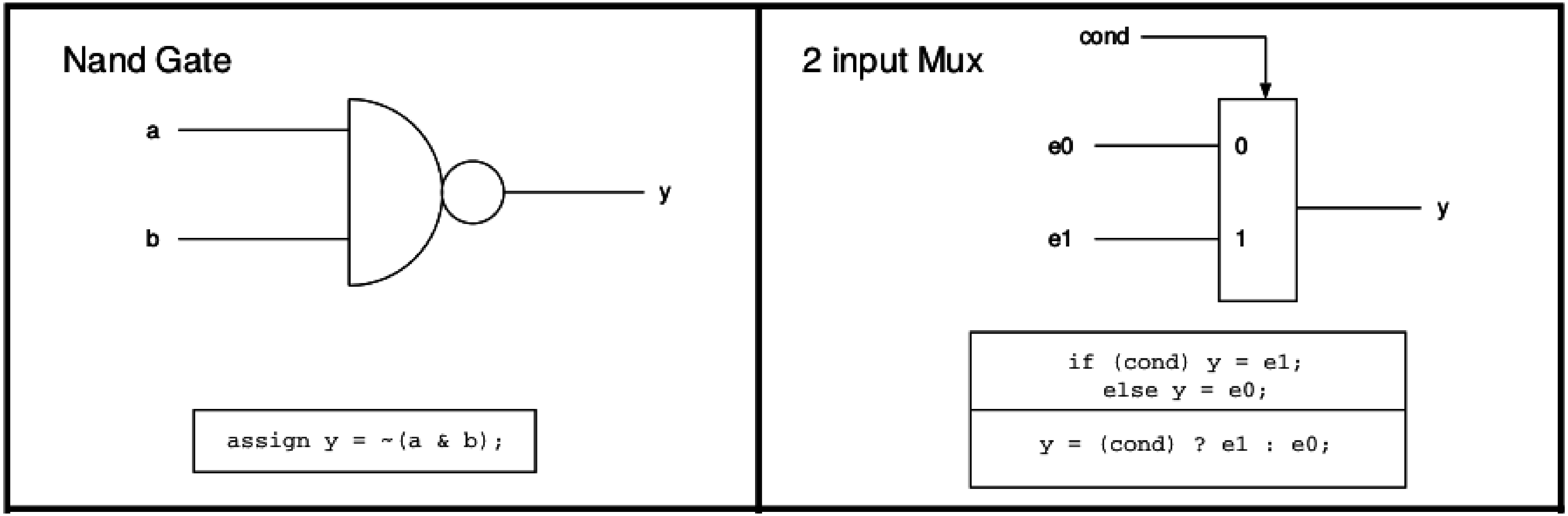
Syntaxe Verilog : sequentiel



```
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

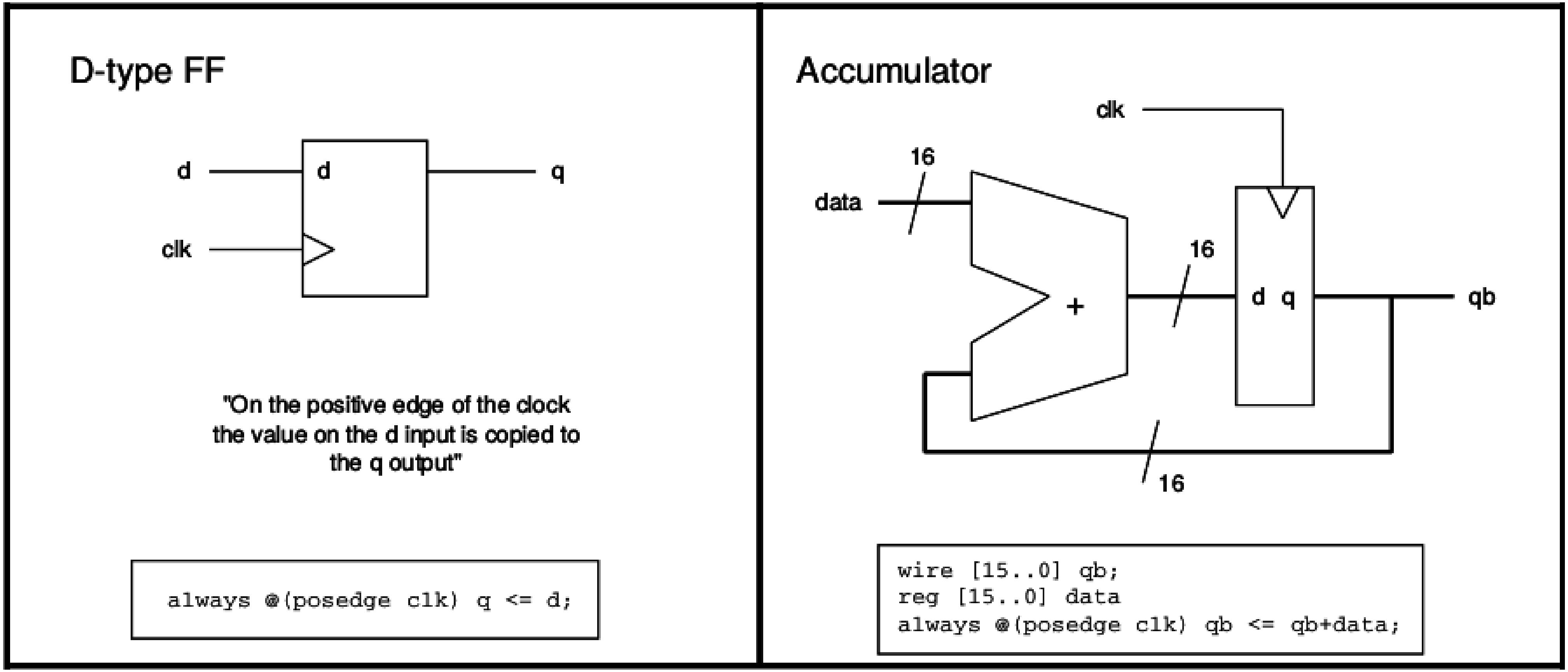


```
assign y = ~(a & b);
```

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

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

Petits exemples de codes en Verilog

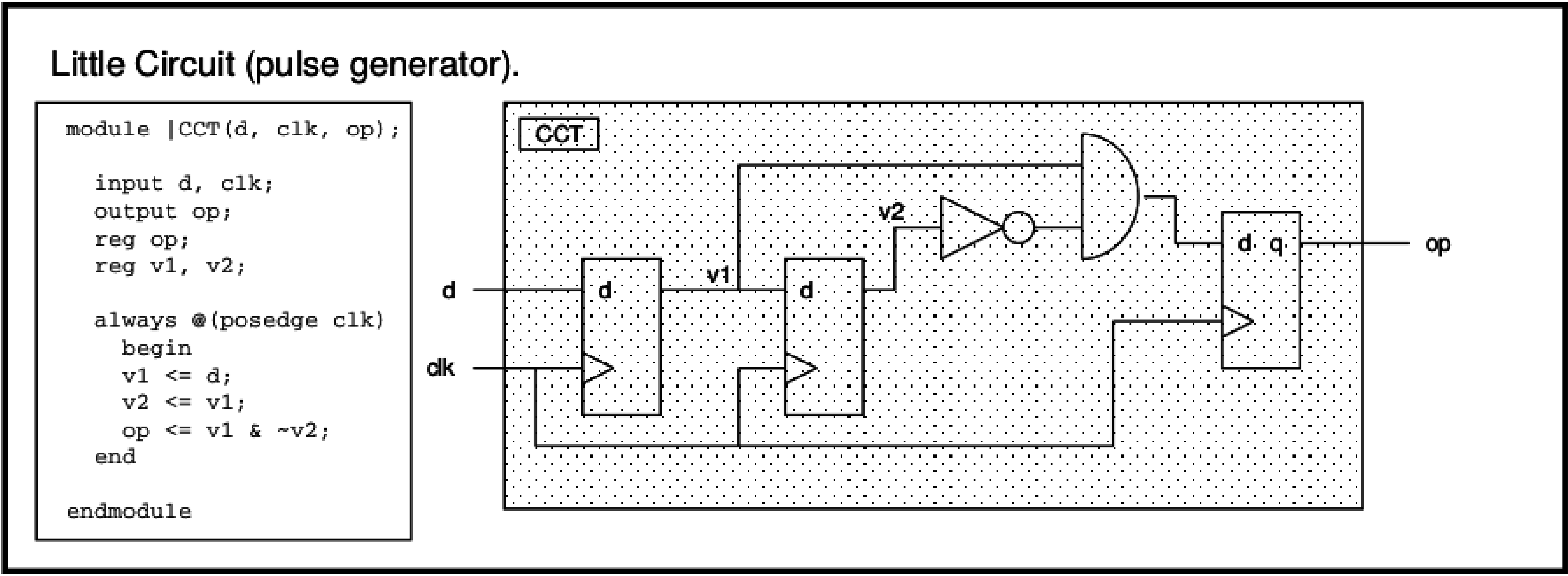


"On the positive edge of the clock the value on the d input is copied to the q output"

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

```
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.

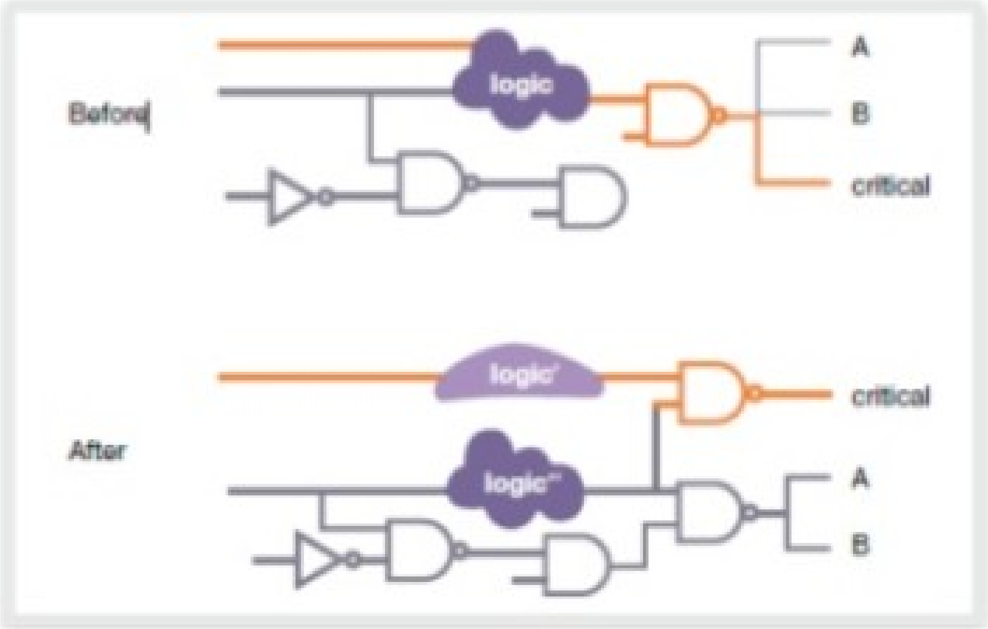


Figure 5. Register duplication [1]

TP : Digital SoC Design

TP: Structure

- RTL
  - adder.v: verilog code for adder
  - adder.io.v: example verilog code with IO Pads
- SYN
  - data : Contains the results of Synthesis Step
  - log: Contains synthesis logs.
  - reports: Contains power/performance/area reports
- PR
  - DBS : Saved Designs
  - LOGS: p&r logs
  - myplugins: Scripts you need to modify
- SIGNOFF
  - To visualize full designs with Virtuoso.
- DOC
  - Innovuse Command Ref.

TP: Tools

- Design Compiler
  - To Synthesize verilog code into a netlist of standard cells.
  - Invoke with $dc_shell
- Innovus
  - Place/route tool
  - Works with abstract standard cells.
- Virtuoso
  - To visualize full design with all layers/ transistors.
  - To perform DRC/LVS

TP:RTL

- Inspect adder.v under rtl.
- To synthesize
  - $ make -C syn synthesis
  - All results are generated in data directory.
  - Check syn/data/adder.synthesis.v ; are all the i/o s same as rtl ? Are there any extra i/o s ?
  - Check the reports under syn/reports. Try to find out performance/power/area values .

TP:Floorplan

- Run $make pr # if all the synthesis files are generated correctly.
- Inspect the file pr/myplugins/pre_place.tcl
  - This file specifies the size.
  - Adds rings around the core area with specified width.
- Specify a floorplan which is in total 1mm2.
  - To do so check the floorplan command in innovus documentation. (under doc)
  - Rerun “make pr”
  - Does the flow finish without errors ?
- Open the design
  - $innovus → choose “Restore Design” → choose DBS/signoff.enc
  - Open tools → Violation Browser
  - Are there any geometry errors ?
- Visualize in Virtuoso
  - $ cd pr/signoff ; launch virtuoso
  - Go to Tools→ Import→ Stream use strmin.template file.
  - A new library adder is created.
  - Open adder

TP:I/O Pads

- **In virtuoso check IO in saed32_io_wb_oa**
- **Check adder.io.v under rtl.**
- **Add all necessary IO pads /corners around adder.**
    - **Modify this new file name under pr/Default.globals.**
- **Specify a floorplan which is in total 1mm2 with IO pads.**
    - **To do so check the floorplan command in innovus documentation. (under doc)**
    - **Rerun “make pr”**
    - **Does the flow finish without errors ?**
- **Open the design**
    - **$innovus → choose “Restore Design” → choose DBS/signoff.enc**
    - **Open tools → Violation Browser**
    - **Are there any geometry errors ?**
- **Visualize in Virtuoso**
    - **$ cd pr/signoff ; launch virtuoso**
    - **Go to File→ Import→ Stream use strmin.template file.**
    - **.Go to File→ Export->Stream**
    - **Submit the gds file generated.**