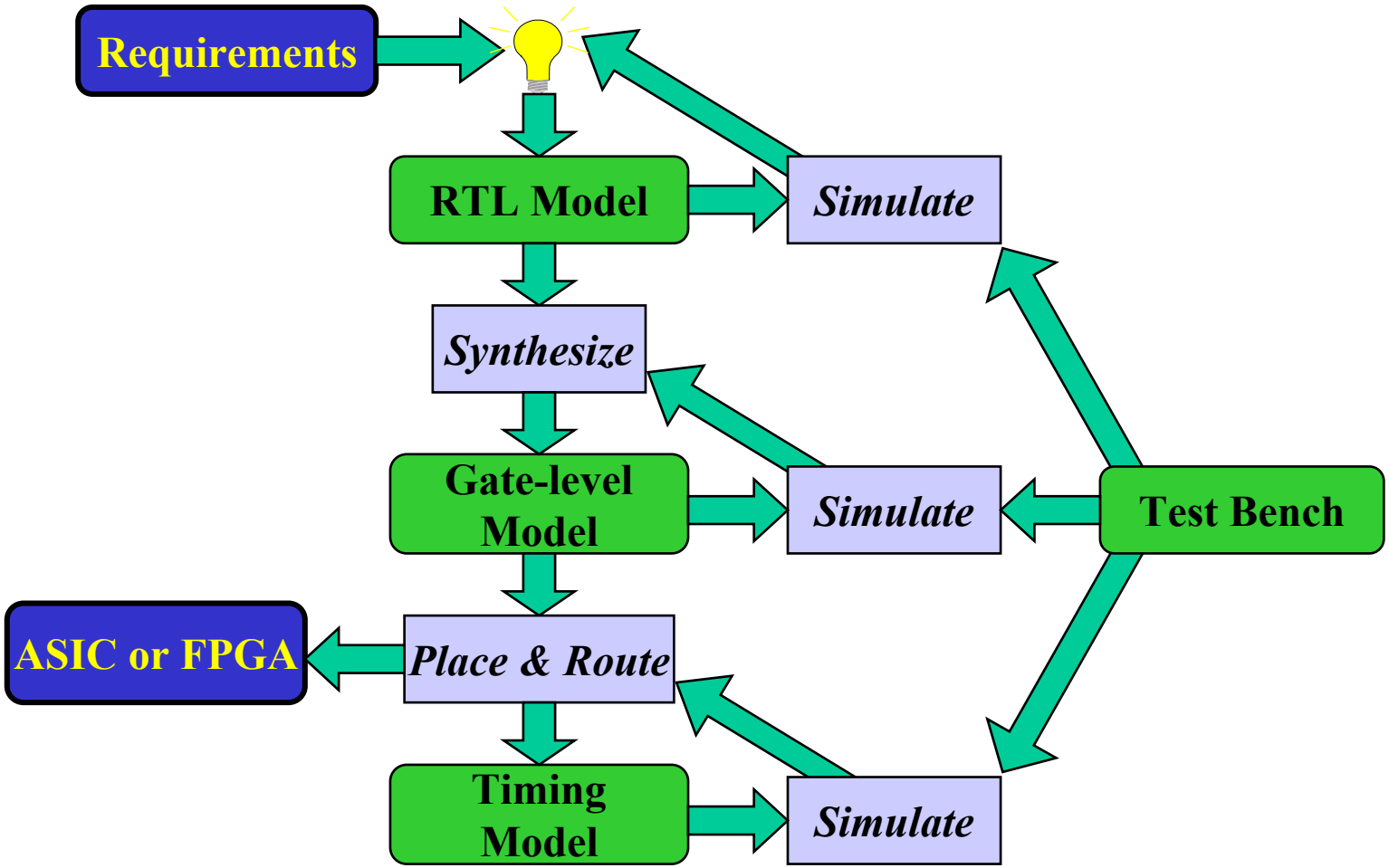
HY-225

Verilog HDL. Τα βασικά...

Βασική Ροή Σχεδίασης



Τι είναι η Verilog;

- Verilog Hardware Description Language(HDL)
  - Μία υψηλού επιπέδου γλώσσα που μπορεί να αναπαραστεί και να προσομοιώνει ψηφιακά κυκλώματα.
    - Hardware concurrency
    - Parallel Activity Flow
    - Semantics for Signal Value and Time
  - Παραδείγματα σχεδίασης με Verilog HDL
    - Intel Pentium, AMD K5, K6, Athlon, ARM7, etc
    - Thousands of ASIC designs using Verilog HDL
- Other HDL : VHDL, SystemC, SystemVerilog

Συμβάσεις στην γλώσσα Verilog

- Η Verilog είναι case sensitive.
  - Λέξεις κλειδιά είναι σε μικρά.
- Σχόλεια
  - Για μία γραμμή είναι //
  - Για πολλές /* */
- Βασικές τιμές 1-bit σημάτων
  - 0: λογική τιμή 0.
  - 1: λογική τιμή 1
  - x: άγνωστη τιμή
  - z: ασύνδετο σήμα. high impedance

Αριθμοί

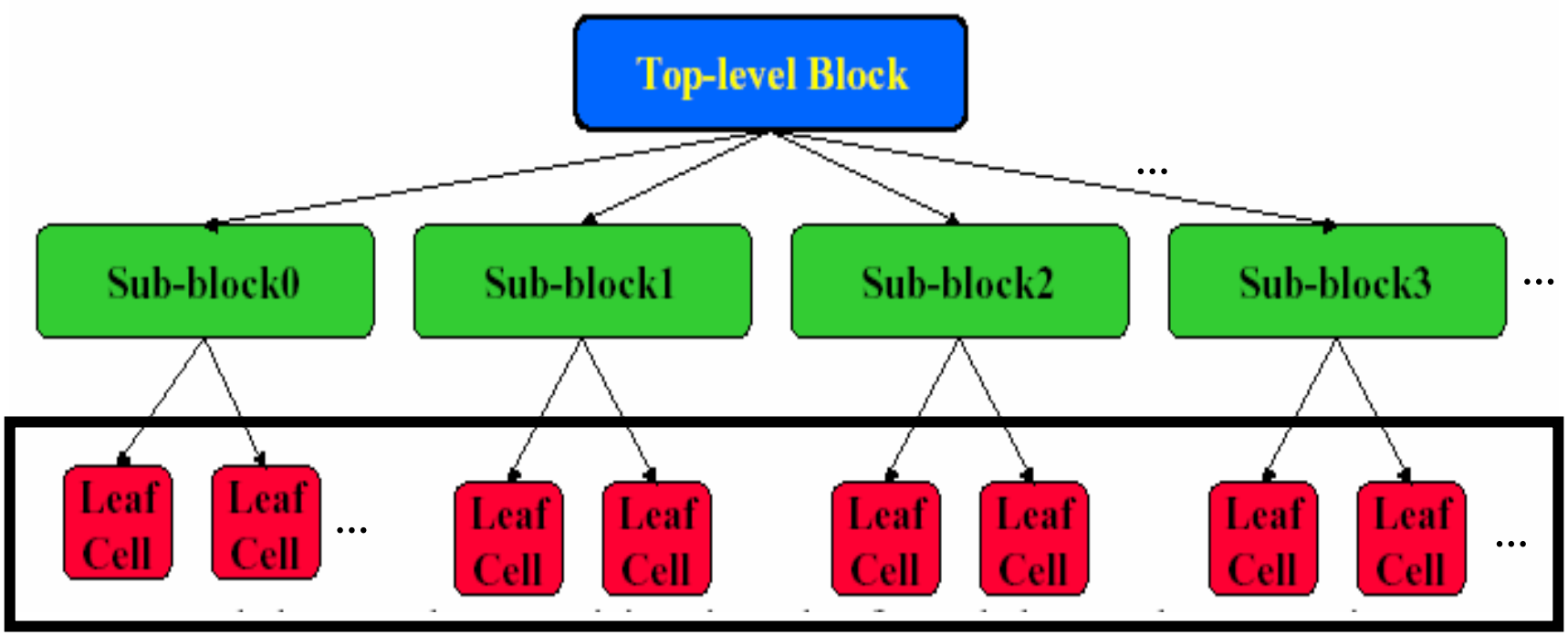
- Αναπαράσταση αριθμών
  - <size>’ <base_format><number>

    <size> δείχνει τον αριθμό απο bits

    <base_format> μπορεί να είναι : d, h, b, o
  - Όταν το <size> λείπει το μέγεθος καθορίζεται από τον compiler.

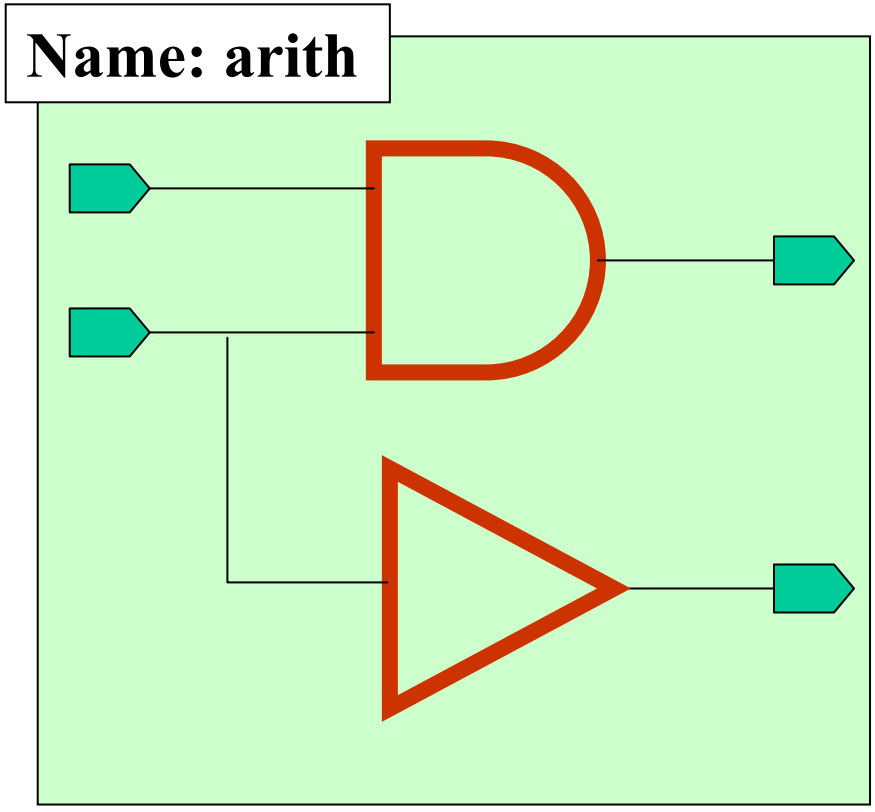
- 100 // 100
- 4’b1111 // 15, 4 bits
- 6’h3a // 58, 6 bits
- 6’b111010 // 58, 6 bits
- 12’h13x // 304+x, 12 bits

Μεθοδολογία Σχεδίασης



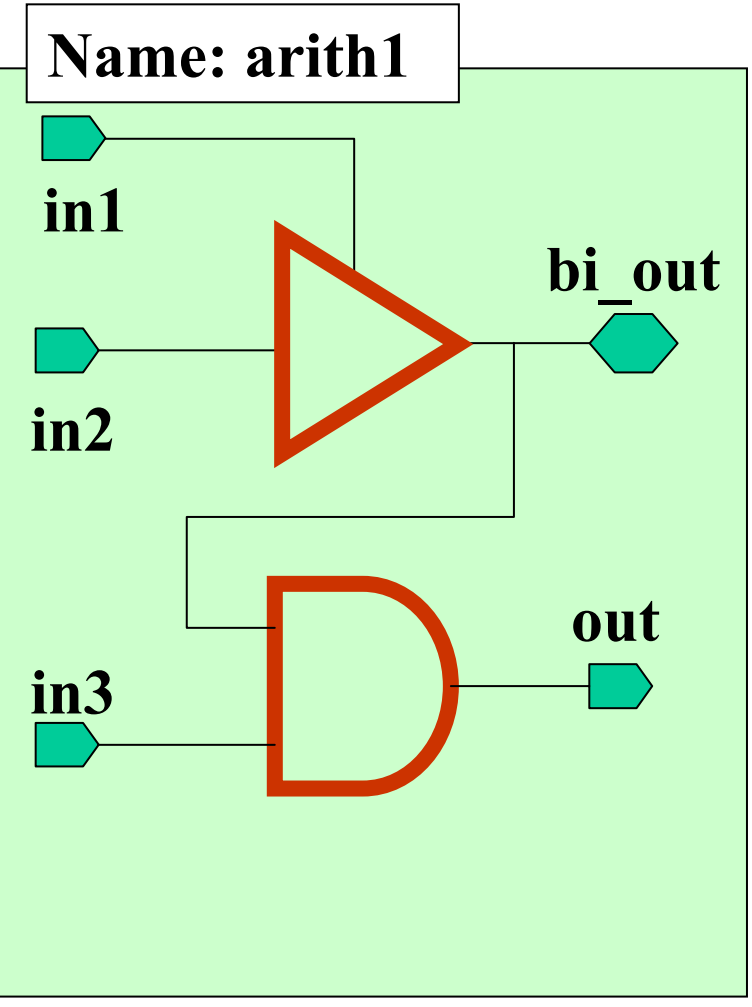
Τελικό σύστημα αποτελείται από τα Leaf blocks που τρέχουν όλα παράλληλα. Δεν υπάρχει program counter στην Verilog.

Βασικό Block: Module




```
module arith (out1, out2,
in1, in2);

output out1, out2;

input in1, in2;

…

…

endmodule
```

Πόρτες ενός Module

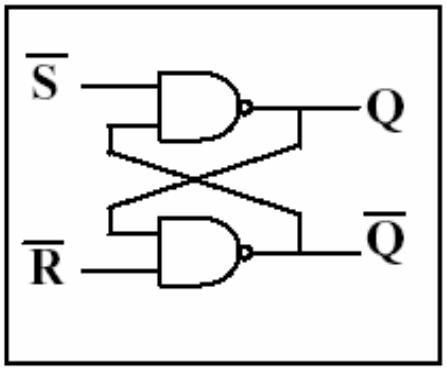



```
module arith1 (bi_out, out, in1,
                in2, in3);
inout bi_out;
output out;
input in1, in2;
input in3;
…
…
endmodule
```

Modules vs Instances

- Instantiation είναι η διαδικασία δημιουργίας αντικειμένου από το module.

Storage Cell



$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
0	0	undef	
0	1	1	0
1	0	0	1
1	1	Q	$\overline{Q}$

```
module nand(out, a, b,);
input a, b;
Output out;
wire out;
  out = ~ (a & b);
endmodule
```

```
module SRLATCH(Q, Qbar, Sbar, Rbar);
input  Sbar, Rbar;
output Q, Qbar;

// Instantiate lower-level modules
nand n1(Q, Sbar, Qbar);
nand n2(Qbar, Rbar, Q);

endmodule
```

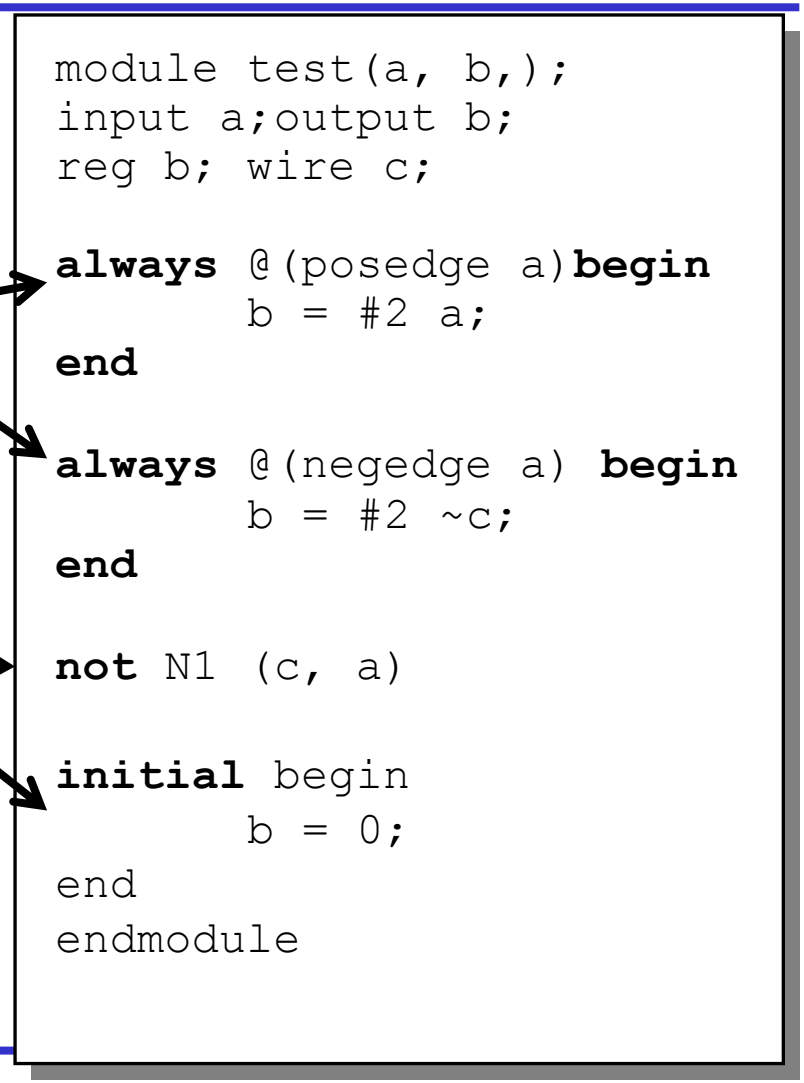
Χρόνος Προσομοίωσης

- `timescale <time_unit>/<time_precision>
  - time_unit: μονάδα μέτρησης χρόνου
  - time_precision: ελάχιστο χρόνο βήματα κατά την προσομοίωση.

  Μονάδες χρόνου : s, ms, us, ns, ps
- #<time> : αναμονή για χρόνο <time>
  - #5 a=8'h1a
- @<σήμα> : αναμονή μέχρι το σήμα να αλλάξει τιμή
  - @ (posedge clk)
  - @ (negedge clk)
  - @ (a)

Module Body

- Declarations
  - always blocks. Μπορεί να περιέχει πάνω από ένα
  - initial. Μπορεί να περιέχει ένα ή κανένα.
  - primitives
- Instantiations



```
module test(a, b,);
input a;output b;
reg b; wire c;

always @(posedge a)begin
        b = #2 a;
end

always @(negedge a) begin
        b = #2 ~c;
end

not N1 (c, a)

initial begin
        b = 0;
end
endmodule
```

Τύποι μεταβλητών στην Verilog

- integer // αριθμός
- wire // καλώδιο – σύρμα
- reg // register
- tri // tristate

Wires

- Συνδυαστική λογική (δεν έχει μνήμη)
- Γράφος εξαρτήσεων
- Μπορεί να περιγράψει και ιδιαίτερα πολύπλοκη λογική…

```
wire sum  = a ^ b;
wire c = sum | b;
wire a = ~d;
```

```
wire sum;
...
assign sum = a ^ b;
```

```
wire muxout = (sel == 1) ? a : b;
wire op = ~(a & ((b) ? ~c : d) ^ (~e));
```

Registers (ακολουθιακή λογική)

- Στοιχεία μνήμης
- … κάτι ανάλογο με μεταβλητές στη C
- Μόνο registers (οχι wires) παίρνουν τιμή σε initial και always blocks.
- Synthesizable code:
  - Αναθέσεις «κοντά»
  - Εξαίρεση: test bench
- Hold time

```
reg a;

initial begin
 a = 0;
 #5;
 a = 1;
end
```

```
reg q;

always @(posedge clk)
begin
 if (load)
  q = #2 d;
end
```

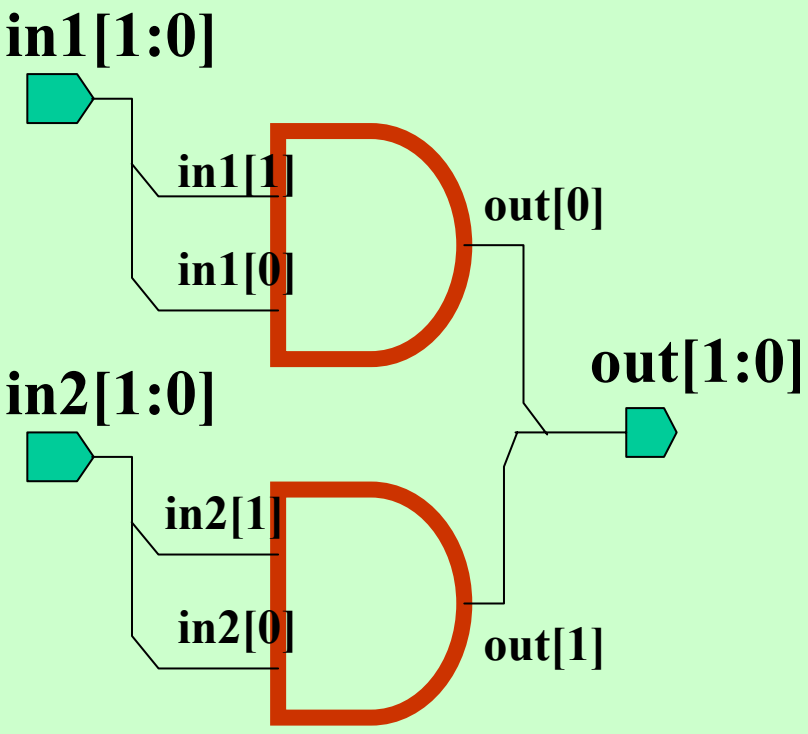
Συνδέσεις μεταξύ Instances

- Με βάση την θέση
  - module adder(Sum, In1, In2)
  - adder (A, B, C) // Sum = A, In1 = B, In2 = C

- Συσχετίζοντας ονόματα
  - module adder(Sum, In1, In2)
  - adder (.In2(B), .In1(A), .Sum(C))
    // Sum = C, In1 = A, In2 = B

Buses

Name: arith2




```
module arith2 (out, in1,
in2);

output [1:0] out;

input [1:0] in1, in2;

…

…

endmodule
```

Buses (cont)

```
module adder(a, b, sum, cout);

input  [7:0] a, b;
output [7:0] sum;
output       cout;

wire [8:0] tmp = a + b;

wire [7:0] sum  = tmp[7:0];
wire       cout = tmp[8];

endmodule
```

- Καμία διαφορά στη συμπεριφορά
- Συμβάσεις:
  - [high : low]
  - [msb : lsb]
- Προσοχή στις συνδέσεις εκτός του module…

Γράψιμο Κώδικα

- Η λειτουργία ένος module μπορεί να οριστεί με διάφορους τρόπους:
  - Behavioral (επίπεδο πιο κοντά στην λογική)

    Παρόμοια με την C – ο κώδικας δεν έχει άμεση σχέση με το hardware.

    π.χ. wire a = b + c

  - Gate level/structural (επίπεδο κοντά στο hardware)

    Ο κώδικας δείχνει πως πραγματικά υλοποιειται σε πύλες η λογική.

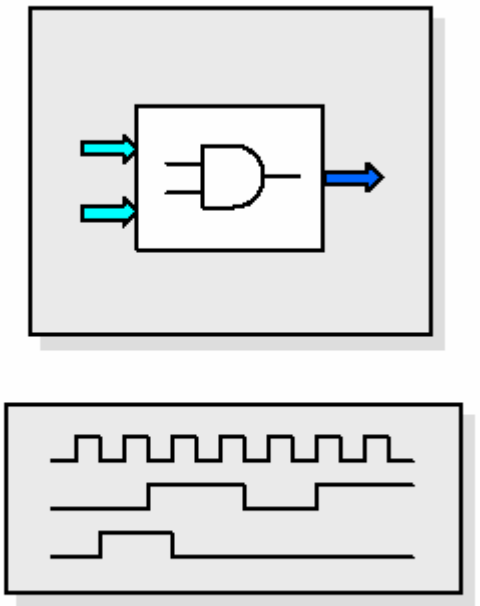
    π.χ.

```
wire sum  = a ^ b;
wire cout = a & b;
```

Testing

Hierarchical Testing

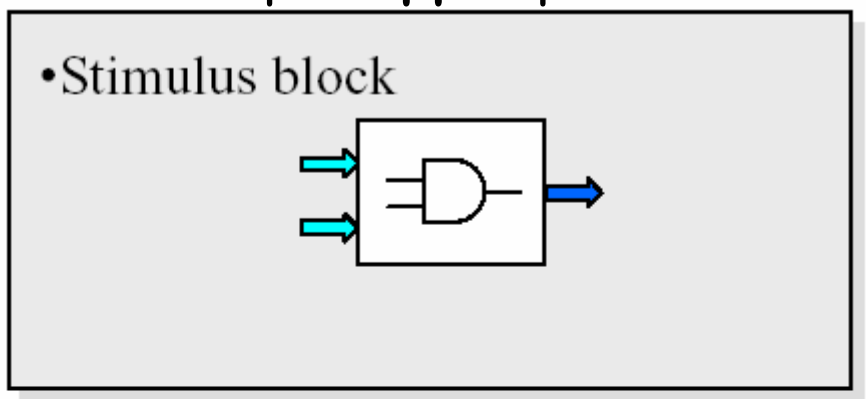
- Modules
  - Individual
  - **- Block-level simulation**
- Design Simulation
  - Top level **/ System-level**
  - Stimulus block (Test bench)
- Evaluations



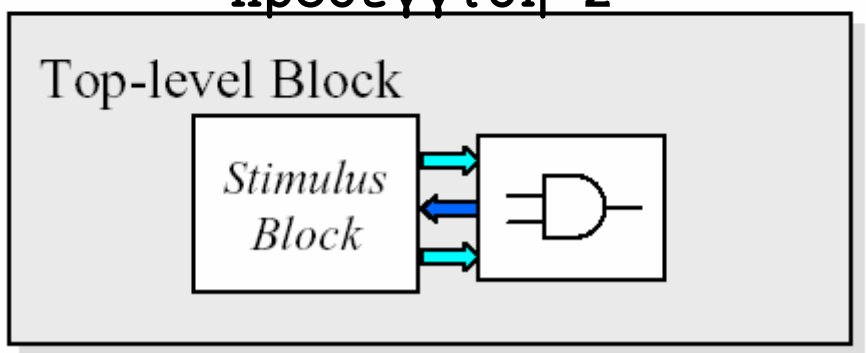
Προσεγγίσεις για έλεγχο σωστής λειτουργίας

- Testbench : top module που κάνει instantiate το module που τεστάρουμε, δημιουργεί τις τιμές των εισόδων του και ελέγχει ότι οι έξοδοί του παίρνουν σωστές τιμές.

Προσέγγιση 1



Προσέγγιση 2



Ένα απλό «test bench»

```
module test;
reg  a, b;
wire s, c;

adder add0(a, b, s, c);

initial begin
 a = 0; b = 0;
 #5 $display("a: %x, b: %x, s: %x, c: %x", a, b, s, c);
 a = 1;
 #5 $display("a: %x, b: %x, s: %x, c: %x", a, b, s, c);
 b = 1;
 #5 $display("a: %x, b: %x, s: %x, c: %x", a, b, s, c);
 a = 0;
 #5 $display("a: %x, b: %x, s: %x, c: %x", a, b, s, c);
end

endmodule
```

```
module adder(a, b, sum, cout);
input  a, b;
output sum, cout;

wire sum  = a ^ b;
wire cout = a & b;

endmodule
```

Verilog Simulator

> rlogin [garbis, kirkios, levantes, apraktias, pounentes, apiliotis]
> source ~hy225/verilog/scripts/cds_ldv.sh
> mkdir test; cd test
> cp ~hy225/verilog/examples/test.v
> verilog test.v
> signalscan

signalscan

- **File > Open Simulation File (Διαλέξτε το test.shm/test.trn)**
- **Πατείστε το DesBrows**
- **Επιλέξτε το «test» στο Instances in Current Context**
- **Πατείστε 2 φορές στο «clk»**
- **Πατείστε το AddToWave**

test.v

```
initial
begin
// Start Tracing (signalscan)
$shm_open("test.shm");
$shm_probe(test, "AS");

// print values of clk at stdout each time
// it changes
$monitor ($time, ":clk=%b", clk);
#200
// Stop Tracing
$shm_close();
// Stop Simulation
$finish;
end
```

Ανακεφαλαίωση!

- Συνδυαστική λογική:
  - `wire`
- Ακολουθιακή λογική:
  - `reg`
- Buses:
  - `[high:low]`
- Καθυστερήσεις:
  - `#t`
  - `@(posedge …)`
- Δύο τύποι statements:
  - `initial`
  - `always`
- Είσοδος - έξοδος:
  - `input, output`
- Αποτελέσματα:
  - `$display`
  - `Cadence waves`