Γλώσσες Περιγραφής Υλικού Ι

Εισαγωγή στη Verilog HDL

Νικόλαος Καββαδίας
nkavv@uop.gr

28 Φεβρουαρίου 2012

Αντικείμενο του μαθήματος CST304: Γλώσσες Περιγραφής Υλικού Ι

- Επιμέρους στόχοι του μαθήματος
  - Σχεδιασμός ψηφιακών κυκλωμάτων με τη γλώσσα περιγραφής υλικού Verilog HDL (ή απλά Verilog)
  - Παρουσίαση χαρακτηριστικών συνθέσιμων κυκλωμάτων που συναντώνται στη μοντέρνα ψηφιακή σχεδίαση
  - Εργαστηριακή εξάσκηση στην περιγραφή και προσομοίωση ψηφιακών κυκλωμάτων
- Τρόπος βαθμολόγησης του μαθήματος
  - Το μάθημα θα έχει γραπτή εξέταση, δύο πακέτα ασκήσεων και μία υποχρεωτική εργασία (σε ομάδες των 1-2 ατόμων)
  - Έστω x ο βαθμός στη γραπτή εξέταση και y σε ασκήσεις/εργασία
  - Τελικός βαθμός ($0.0 \leq G \leq 10.0$)
    - $G = x$, αν $x < 5.0$
    - $G = 0.7 \cdot x + 0.3 \cdot y$, αν $x \geq 5.0$
- Ενημέρωση για ανακοινώσεις, διαλέξεις, ύλη, εργασίες: `http://eclass.uop.gr/courses/CST304/`

Οργάνωση των παραδόσεων

Ενδεικτική κατανομή των διαλέξεων

1. Εισαγωγή στη Verilog HDL
2. Μοντελοποίηση συνδυαστικών κυκλωμάτων
3. Μοντελοποίηση ακολουθιακών κυκλωμάτων
4. Προχωρημένα στοιχεία της Verilog HDL
5. Σύνταξη παραμετρικών περιγραφών και σχεδίαση μνημών
6. Μηχανές πεπερασμένων καταστάσεων
7. Μη προγραμματιζόμενοι επεξεργαστές
8. Κυκλώματα για προχωρημένους και στοιχεία λογικής σύνθεσης
9. Προγραμματιζόμενοι επεξεργαστές
10. Θέματα πρακτικής εξάσκησης

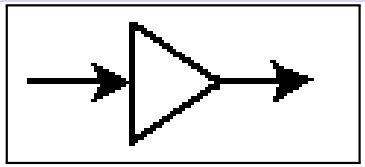
Εισαγωγικά

- Η Verilog αποτελεί μια γλώσσα **καταγραφής δομής** και **περιγραφής λειτουργικής συμπεριφοράς** ψηφιακών κυκλωμάτων/συστημάτων
  - **Δεν** θα την αντιμετωπίσουμε ως άλλη μία γλώσσα διαδικαστικού προγραμματισμού
- Επιτρέπει τη μοντελοποίηση υλικού (hardware)
  - Περιγραφή ψηφιακών κυκλωμάτων από το επίπεδο πύλης μέχρι το αλγοριθμικό επίπεδο
  - Εφικτή η μοντελοποίηση κυκλωμάτων σε επίπεδο τρανζίστορ σε λειτουργία διακοπτών
- Αναπτύχθηκε ως βιομηχανική γλώσσα με βάση την ANSI C από τους Phil Moorby και Prabhu Goel της Gateway D.A. την περίοδο 1983/1984
- Χρησιμοποιείται περισσότερο σε Βόρεια Αμερική, Ιαπωνία και γενικά στη βιομηχανία

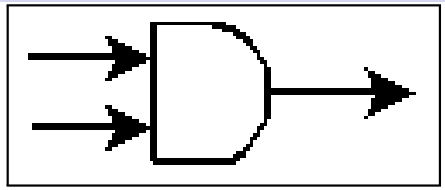
Δυνατότητες και χαρακτηριστικά της Verilog

- Επιτρέπει τη χρήση διαφορετικών μεθοδολογιών σχεδιασμού
- Προσφέρει ανεξαρτησία από την εκάστοτε τεχνολογία υλοποίησης (standard cell VLSI, FPGA)
- Διευκολύνει την επικοινωνία σχεδίων μεταξύ συνεργαζόμενων ομάδων σχεδίασης
- Βοηθά στην καλύτερη διαχείριση του έργου της σχεδίασης
- Στη Verilog HDL μπορεί να περιγραφεί ένα μεγάλο εύρος ψηφιακών κυκλωμάτων
- Χαρακτηριστικά της είναι:
  - Διαθέτει έμφυτους τύπους δεδομένων
  - Τα μοντέλα υλικού σε Verilog είναι γενικά σύντομα σε έκταση και εύκολα κατανοητά
  - Δεν διαθέτει αυστηρή τυποποίηση και χρειάζεται διεξοδική ανάλυση για την αποφυγή σφαλμάτων
  - Για τη σύνταξη μοντέλων επιβεβαίωσης ορθής λειτουργίας, χρησιμοποιούνται κλήσεις σε ρουτίνες συστήματος

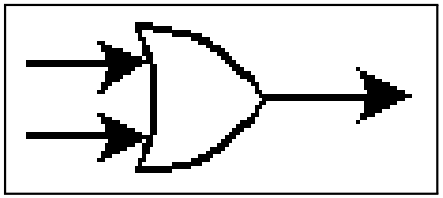
Προκαθορισμένες λογικές πύλες στη Verilog (1)



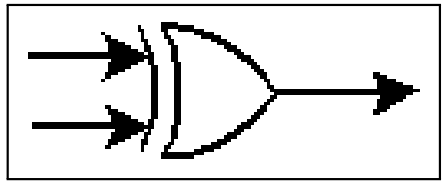
a	buf t(y, a)
0	0
1	1



a	b	and t(y, a, b)
0	0	0
0	1	0
1	0	0
1	1	1

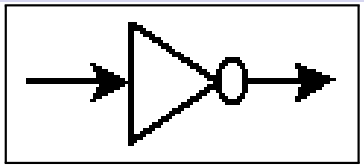


a	b	or t(y, a, b)
0	0	0
0	1	1
1	0	1
1	1	1

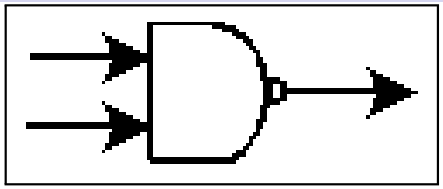


a	b	xor t(y, a, b)
0	0	0
0	1	1
1	0	1
1	1	0

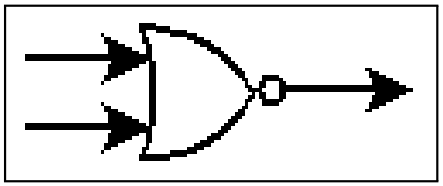
Προκαθορισμένες λογικές πύλες στη Verilog (2)



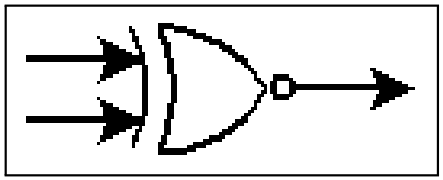
a	not t(y, a)
0	1
1	0



a	b	nand t(y, a, b)
0	0	1
0	1	1
1	0	1
1	1	0



a	b	nor t(y, a, b)
0	0	1
0	1	0
1	0	0
1	1	0



a	b	xnor t(y, a, b)
0	0	1
0	1	0
1	0	0
1	1	1

Ένα τυπικό παράδειγμα: Πολυπλέκτης 2-σε-1

- Το module (άρθρωμα) `mux2_1` υλοποιεί έναν πολυπλέκτη 2-σε-1 για εισόδους των DW bit
- Σχεδιασμός παραμετρικού συνδυαστικού κυκλώματος σε επίπεδο RTL

```
module mux2_1 (in0, in1, sel, mout);
  parameter DW  = 8;
  input[DW-1:0] in0;
  input[DW-1:0] in1;
  input sel;
  output[DW-1:0] mout;
  reg[DW-1:0] mout;

  always @(sel or in0 or in1)
  begin
    case (sel)
      1'b0: begin
        mout <= in0;
      end
      default: begin
        mout <= in1;
      end
    endcase
  end
endmodule
```

Μεθοδολογίες μοντελοποίησης κυκλωμάτων στη Verilog

- Η Verilog προσφέρεται για την περιγραφή κυκλωμάτων και σε χαμηλό αλλά και σε υψηλό επίπεδο αφαίρεσης
- Επίπεδα αφαίρεσης στην ψηφιακή σχεδίαση με Verilog
  - Αλγοριθμικό επίπεδο ή επίπεδο συμπεριφοράς (behavioral/algorithmic level)
    - Καταγράφεται ο επιθυμητός αλγόριθμος χωρίς ακριβή αντιστοίχιση με συγκεκριμένους πόρους υλικού
  - Επίπεδο ροής δεδομένων (dataflow level)
    - Καταγράφεται η ροή επεξεργασίας των δεδομένων εισόδου για την παραγωγή των δεδομένων εξόδου με ενδεχόμενη μεταβολή της κατάστασης του κυκλώματος
    - Ονομάζεται και επίπεδο μεταφοράς καταχωρητή (RTL : Register-Tranfer Level)
  - Επίπεδο πυλών (gate level)
    - Υλοποίηση με πύλες και διασυνδέσεις μεταξύ των πυλών
  - Επίπεδο διακοπτών (switch level)
    - Υλοποίηση με τρανζίστορ τύπου NMOS, PMOS σε διακοπτική λειτουργία

Αναπαράσταση πυλών

- Μοντελοποίηση χαρακτηριστικών μιας πύλης
  - Λειτουργικότητα
  - Χρονική καθυστέρηση (διάδοσης)
- Είδη πυλών
  - `NAND`, `NOR`, `AND`, `OR`, `XOR`, `XNOR`, `BUF`, `NOT`
  - Τρισταθείς πύλες με επίτρεψη στο 0 ή στο 1: `BUFIF0`, `BUFIF1`, `NOTIF0`, `NOTIF1`
- Οι πύλες δέχονται γενικά n εισόδους και παράγουν μία έξοδο
- Γενική μορφή της δήλωσης μίας πύλης Verilog

```
gate-type #delay instance-name(out, in1, in2, in3, ...);
```

Η έννοια της χρονικής καθυστέρησης (delay) βασικών πυλών στη Verilog

- Στη Verilog υπάρχουν τέσσερις θεμελιώδεις τρόποι για τη δήλωση χρονικών καθυστερήσεων σε λογικές πύλες (gate primitives)
  - Μηδενική καθυστέρηση
  - Καθυστέρηση διάδοσης (propagation delay)
  - Καθυστέρηση ανόδου και καθόδου (rise/fall time delay)
  - Ελάχιστη-τυπική-μέγιστη καθυστέρηση

```
// Zero delay
buf           b1(a, b);
// Delay of 3 time units
buf #3        b2(c, d);
// Rise = 4, Fall = 5
buf #(4,5)    b3(e, f);
// Min=3, Typ=4, Max=5
buf #(3:4:5) b4(g, h);
```

Η τετράτιμη λογική `0, 1, X, Z` (1)

- Στη Verilog ένα bit μπορεί κατά τη διάρκεια λειτουργίας του κυκλώματος να λάβει μία από τις εξής τιμές
  - 0: Λογικό μηδέν (forcing 0)
  - 1: Λογικό ένα (forcing 1)
  - Z: Κατάσταση υψηλής εμπέδησης (αντίστασης). Αντιστοιχεί στην έξοδο μιας τρισταθούς πύλης όταν αυτή δεν οδηγείται (high-impedance state)
  - X: Άγνωστη τιμή. Μοντελοποιεί την περίπτωση στην οποία η προσομοίωση δεν καταλήγει σε κάποια τιμή (0, 1 ή Z) για κάποιο κόμβο του κυκλώματος (forcing unknown)
- Περιπτώσεις άγνωστης τιμής
  - Αρχική κατάσταση καταχωρητή
  - Όταν ένα καλώδιο (wire στη Verilog) οδηγείται από το 0 και το 1 ταυτόχρονα
  - Έξοδος μίας πύλης της οποίας η είσοδος είναι Z

Η τετράτιμη λογική `0, 1, X, Z` (2)

- Πως λειτουργούν οι λογικοί τελεστές με την τετράτιμη λογική
- Παράδειγμα: Πύλη AND των 2 εισόδων

AND	0	1	X	Z
0	0	0	0	0
1	0	1	X	X
X	0	X	X	X
Z	0	X	X	X

Επίπεδα ισχύος ενός σήματος στη Verilog

- Τα επίπεδα ισχύος (strength levels) χρησιμοποιούνται για την επίλυση διαμαχών μεταξύ διαφορετικών οδηγών
- Εφαρμόζονται για τα λογικά επίπεδα 0 και 1

Strength level	Type	Degree
supply	Driving	strongest
strong	Driving	
pull	Driving	
large	Driving	
weak	Driving	
medium	Storage	
small	Storage	
highz	High impedance	weakest

- Παραδείγματα
  - Διαμάχη `strong1 - weak0` → `strong1`
  - Διαμάχη `strong1 - strong0` → `X`

ΑΡΘΡΩΜΑ (MODULE)

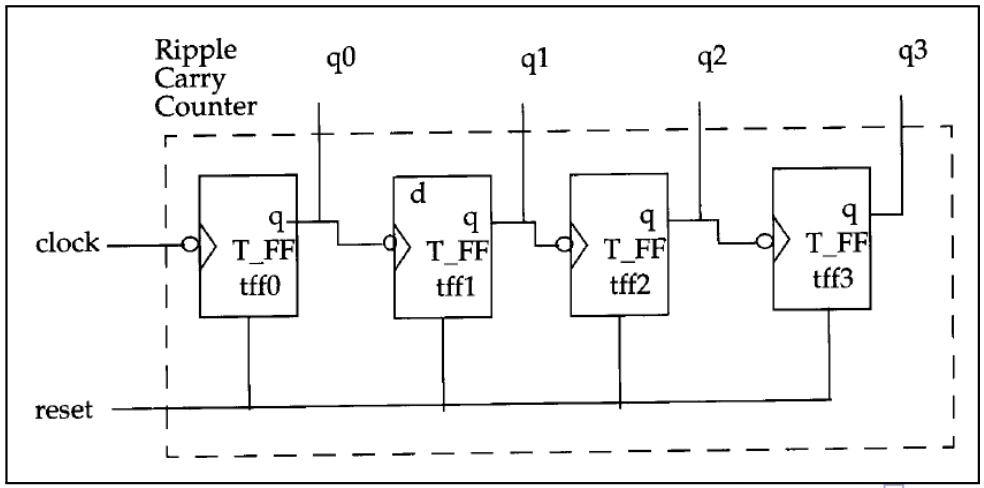
- Περιγράφει τον τρόπο διασύνδεσης με το περιβάλλον του (διεπαφή) καθώς και τη συμπεριφορά του κυκλώματος
- Δήλωση των θυρών εισόδου/εξόδου προς και από το κύκλωμα
- Δήλωση παραμέτρων με εμβέλεια στο άρθρωμα

Σύνταξη ενός module:

```
module <module-name> (<module-terminal-list>);
  ...
  <module internals>
  ...
endmodule
```

Αντίτυπο ενός MODULE (1)

- Το αντίτυπο ενός module αποτελεί ένα τοπικό αντίγραφο ενός module
- Μπορεί να χρησιμοποιηθεί πολλές φορές στην ίδια σχεδίαση
- Χρησιμοποιείται εντός άλλων module για την υλοποίηση μέρους της συμπεριφοράς τους
- Βασικό στοιχείο για την ιεραρχική σχεδίαση
- Παράδειγμα: Απαριθμητής ριπής κρατουμένου των 4-bit



Αντίτυπο ενός MODULE (2)

- Η διεπαφή του T flip-flop

```
module T_FF (q, clock, reset);
```

- Χρήση του T flip-flop στον κώδικα του απαριθμητή

```
module ripple_carry_counter(q, clk, reset);
  output [3:0] q;
  input  clk, reset;

  T_FF  tff0(q[0], clk, reset);
  T_FF  tff1(q[1], q[0], reset);
  T_FF  tff2(q[2], q[1], reset);
  T_FF  tff3(q[3], q[2], reset);
endmodule
```

Οι βασικοί τύποι σημάτων: Δικτυώματα (nets) και καταχωρητές (regs)

- Δικτυώματα
  - Αναπαριστούν διασυνδέσεις μεταξύ αντικειμένων όπως οι θύρες εισόδου και εξόδου σε αντίτυπα κυκλωμάτων
  - Δεν διατηρούν (δεν αποθηκεύουν) την τιμή τους
  - Λαμβάνουν την τιμή τους από κάποιον οδηγό (π.χ. από την έξοδο κάποιας πύλης)
  - Η συντρέχουσα ανάθεση τιμής γίνεται με δήλωση `assign`
  - Δημιουργία συνδυαστικής λογικής
- 'Καταχωρητές'
  - Αναπαριστούν αποθήκευση δεδομένων
  - Διασυνδέονται σε εισόδους ενός αντιτύπου κυκλώματος
  - Δηλώνονται ως έξοδοι στην περιγραφή ενός κυκλώματος
  - Διατηρούν την τιμή τους μέχρι αυτή να λάβει νέα ανάθεση σε ένα ειδικό μπλοκ λογικής `initial` ή `always`
  - Μοντελοποίηση μανδαλωτών, καταχωρητών κ.λ.π.
  - Δημιουργία συνδυαστικής και ακολουθιακής λογικής

Περισσότερα για τα δικτυώματα

- Δηλώνονται κυρίως με τη λέξη-κλειδί `wire`
- Αρχική τιμή τους είναι η `Z` εκτός από το τρισταθές δικτύωμα `trireg` για το οποίο είναι η `X`
- Ένα δικτύωμα που δεν οδηγείται παίρνει την τιμή `Z`
- Παραδείγματα

```
wire a;             // Declaration of wire a
wire b, c;          // Declaration of two wires: b, c
wire d = 1'b0;      // Declaration of wire d with initial
                    // value of 0
```

- Η Verilog διαθέτει πλήθος τύπων δικτυωμάτων όπως είναι τα `wire, wand, wor, tri, triand, trior, trireg`
- Σχεδόν πάντα, θα χρησιμοποιούμε μόνο τον τύπο `wire`

Περισσότερα για τους καταχωρητές

- Δηλώνονται με τη λέξη-κλειδί `reg`
- Οι καταχωρητές διατηρούν την τιμή τους μέχρις ότου γίνει κάποια άλλη ανάθεση τιμής σε αυτούς
- Δεν έχουν άμεση φυσική σημασία: στη Verilog, ένας καταχωρητής (`reg`) δηλώνει απλώς μία ΜΕΤΑΒΛΗΤΗ η οποία μπορεί να συγκρατήσει την τιμή της
- Ένας `reg` δεν χρειάζεται απαραίτητα κάποιον οδηγό
- Δεν χρειάζονται εξ ορισμού σήμα ρολογιού όπως οι φυσικοί καταχωρητές
- Η αρχική τιμή τους είναι η `X`
- Παραδείγματα

```
reg reset;              // Declaration of variable reset
initial begin           // The "initial" block
  reset = 1'b1;         // Initialize reset to 1
  #100 reset = 1'b0;    // After 100 t.u., reset is deasserted.
end
```

Διανύσματα

- Τα δικτυώματα και οι καταχωρητές μπορούν να δηλωθούν ως διανύσματα, με εύρος ψηφίων μεγαλύτερο του 1 bit
- Δηλώνονται με έναν από τους δύο εξής τρόπους:
  `{wire|reg} [LOW:HIGH] <net-name>;`

  `{wire|reg} [HIGH:LOW] <net-name>;`
- Το προκαθορισμένο εύρος ενός wire ή reg είναι 1 bit
- Το αριστερό όριο του εύρους bit θεωρείται ως το MSB (Most Significant Bit) του διανύσματος

```
wire a;
wire [7:0] bus;
wire [31:0] busA, busB, busC;
reg clock;
reg [0:40] virtual_addr;
```

- Αναφορά σε τμήμα ενός διανύσματος

```
busA[7]              // bit #7 of vector busA
bus[2:0]             // Three least significant bits of bus
bus[0:2]             // ILLEGAL!
virtual_addr[0:1]    // The two MSBs of vector virtual_addr
```

Αριθμητικοί τύποι δεδομένων: `integer` και `real`

- Τύπος `integer`
  - Τύπος για τη δήλωση προσημασμένων ακέραιων αριθμών
  - Για τα περισσότερα συστήματα το μήκος του `integer` καθορίζεται στα 32 bits

```
integer counter;
  initial
    counter = -1;
```

- Τύπος `real`
  - Αριθμοί κινητής υποδιαστολής με αρχική τιμή το 0.0
  - Χρήση δεκαδικής (decimal) ή επιστημονικής σημειογραφίας (scientific notation)
  - Αν ανατεθούν σε `integer` γίνεται στρογγυλοποίηση στον πλησιέστερο ακέραιο

```
real delta;
delta = 10e4;
delta = 3.14;
```

Τύποι φυσικών μεγεθών στη Verilog και σημαντικές παρατηρήσεις

- Η Verilog υποστηρίζει τον τύπο `time` για την παρακολούθηση του χρόνου προσομοίωσης
- Το εύρος μιας μεταβλητής `time` πρέπει να είναι τουλάχιστον 64 bits
- Η διεργασία συστήματος `$time` χρησιμοποιείται για τη λήψη του τρέχοντος χρόνου προσομοίωσης

```
time save_sim_time;       // Define a time variable
initial
  save_sim_time = $time; // Save the current simulation $time
```

- Χρήσιμες παρατηρήσεις
  - Επιτρέπεται η δήλωση πινάκων για τους τύπους `reg`, `integer`, `time` αλλά όχι για `real`
  - Δεν επιτρέπονται πολυδιάστατοι πίνακες (Verilog-1995)

```
integer  count[0:7];    // Array of 8 count variables
reg      bool[0:31];    // Array of 32 1-bit boolean register variables
reg[4:0] port_id[0:7]; // Array of 8 port IDs, each one is 5 bits wide
```

Συμβολοσειρές (strings)

- Μια συμβολοσειρά αποτελείται από χαρακτήρες οι οποίοι εσωκλείονται σε διπλά εισαγωγικά (double quotation mark)
- Οι συμβολοσειρές αποθηκεύονται σε μεταβλητές `reg`
- Κάθε χαρακτήρας χρειάζεται 8 bit ή αλλιώς 1 byte
- Αν ανατεθεί ένα string σε μία `reg` με μεγαλύτερο διαθέσιμο εύρος, οι κενές θέσεις συμπληρώνονται με μηδενικά

```
// A variable that is 19 bytes wide
reg [8*19:1] string_value;
initial
  // String is stored in the variable
  string_value = "Hello Verilog world";
```

Ειδικοί χαρακτήρες

- Χρησιμοποιούνται ειδικοί χαρακτήρες όπως νέας σειράς (newline), στηλογνώμονα (tab) κυρίως για την εκτύπωση αποτελεσμάτων από προσομοίωση ενός μοντέλου στην κονσόλα του χρήστη

Escaped characters	Απεικονιζόμενοι χαρακτήρες
`\n`	newline
`\t`	tab
`%%`	%
`\\`	\
`\"`	"
`\ooo`	Χαρακτήρας ASCII με την οκταδική διεύθυνση ooo

Διεργασίες του συστήματος (system tasks)

- Η Verilog παρέχει ορισμένες τυποποιημένες διεργασίες συστήματος τις οποίες ο χρήστης μπορεί να καλέσει στον κώδικά του για την εκτέλεση χρήσιμων διαδικασιών
- Οι διεργασίες συστήματος χρησιμοποιούνται για την απεικόνιση αποτελεσμάτων, την παρακολούθηση εσωτερικών κόμβων σε ένα κύκλωμα και για την έναρξη και παύση της προσομοίωσης
- Απεικόνιση πληροφορίας
  - Με την `$display` (παραπλήσια με την `printf` της C)

```
$display (p1, p2, p3, ..., pn);
```

- Τα p1-pn είναι συμβολοσειρές, μεταβλητές ή εκφράσεις
- Η `$display` εισάγει χαρακτήρα νέας γραμμής στο τέλος

Παραδείγματα χρήσης της $display

```
// Display the string in quotes
$display("Hello Verilog World");
-- Hello Verilog World

// Display value of current simulation time 230
$display($time);
-- 230

// Display value of 32-bit virtual address 1fe0001c and time 200
reg [0:31] virtual_addr;
$display("At time %d virtual address is %h", $time, virtual_addr);
-- At time 200 virtual address is 1fe0001c

// Display value of port_id 5 in binary
reg [4:0] port_id;
reg [0:3] bus;
$display("The value of port ID %b is %b", port_id, bus);
-- The value of port ID 00101 is 10xx

// Display the hierarchical name of instance p1 under the
// top-level module (top)
$display("This string is displayed from %m level of hierarchy.");
-- This string is displayed from top.p1 level of hierarchy.
```

Η διεργασία $monitor

- Χρησιμοποιείται για την παρακολούθηση των μεταβολών της τιμής σε ένα σήμα
- Έχει διαμόρφωση παρόμοια με την $display
- Καλείται μόνο μία φορά για κάθε σήμα και ανά πάσα στιγμή μόνο η τελευταία δηλωμένη $monitor είναι ενεργή
- Παράδειγμα

```
initial
begin
  $monitor($time, "Value of signals clock = %b, reset = %b", clock, reset);
end

-- Example output:
-- 0 Value of signals clock = 0, reset = 1
-- 5 Value of signals clock = 1, reset = 1
-- 10 Value of signals clock = 0, reset = 0
```

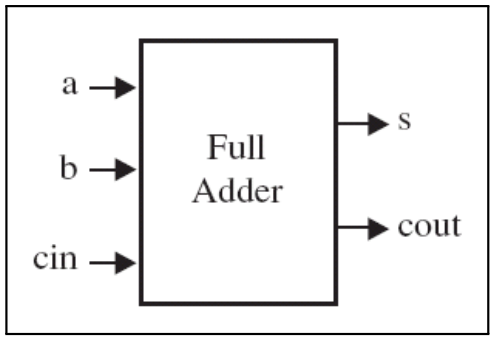
Οι διεργασίες $stop και $finish

- Η διεργασία $stop παύει (προσωρινά) την προσομοίωση
- Η διεργασία $finish τερματίζει την προσομοίωση
- Παραδείγματα

```
initial
begin
  clock = 0;
  reset = 1;
  #100 $stop;     // Suspend the simulation at time = 100
  #900 $finish;   // Terminate the simulation at time = 1000
end
```

Ο πλήρης αθροιστής δυαδικού ψηφίου: Προδιαγραφές

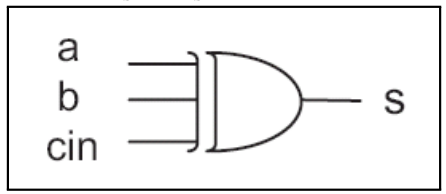
Η διεπαφή του πλήρους αθροιστή (full-adder)



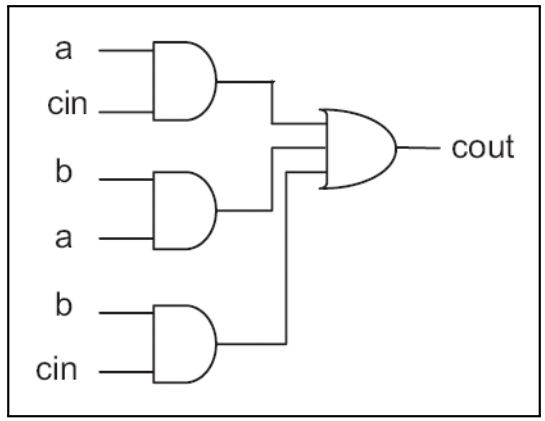
Ο πίνακας αληθείας του full-adder

a	b	cin	s	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Υπολογισμός του ψηφίου s



Υπολογισμός του ψηφίου cout



Ο πλήρης αθροιστής δυαδικού ψηφίου: Υλοποίηση σε Verilog

```
‘timescale 1 ns / 1 ns

module full_adder (a, b, cin, s, cout);
  input a;
  input b;
  input cin;
  output s;
  output cout;
  wire ab, bc, ac;

  xor  xor1(s, a, b, cin);
  or   or1(cout, ab, bc, ac);
  and  and1(ab, a, b);
  and  and2(bc, b, cin);
  and  and3(ac, a, cin);

endmodule
```

Αρχείο ελέγχου/επαλήθευσης (testbench) για τον πλήρη αθροιστή

```
`timescale 1 ns / 1 ns
module main;
  reg  a1, b1, cin1;
  reg[2:0] inputs;
  wire s1, cout1;
  integer idx;
  full_adder test(.a(a1), .b(b1), .cin(cin1), .s(s1), .cout(cout1));
  initial
  begin
    for (idx = 0;  idx <= 7;  idx = idx + 1)
    begin
      inputs = idx;
      a1   = inputs[2];
      b1   = inputs[1];
      cin1 = inputs[0];
      #10                 // 10 time-unit delay
      // Display results after 10 time units
      $display("%t: a=%b, b=%b, cin=%b, s=%b, cout=%b",
        $time, a1, b1, cin1, s1, cout1);
    end
  end
endmodule // main
```

Αποτελέσματα από την προσομοίωση του πλήρους αθροιστή

```
10: a=0, b=0, ci=0, s=0, co=0
20: a=0, b=0, ci=1, s=1, co=0
30: a=0, b=1, ci=0, s=1, co=0
40: a=0, b=1, ci=1, s=0, co=1
50: a=1, b=0, ci=0, s=1, co=0
60: a=1, b=0, ci=1, s=0, co=1
70: a=1, b=1, ci=0, s=0, co=1
80: a=1, b=1, ci=1, s=1, co=1
```