



VERILOG



Γενικά περί γλώσσας



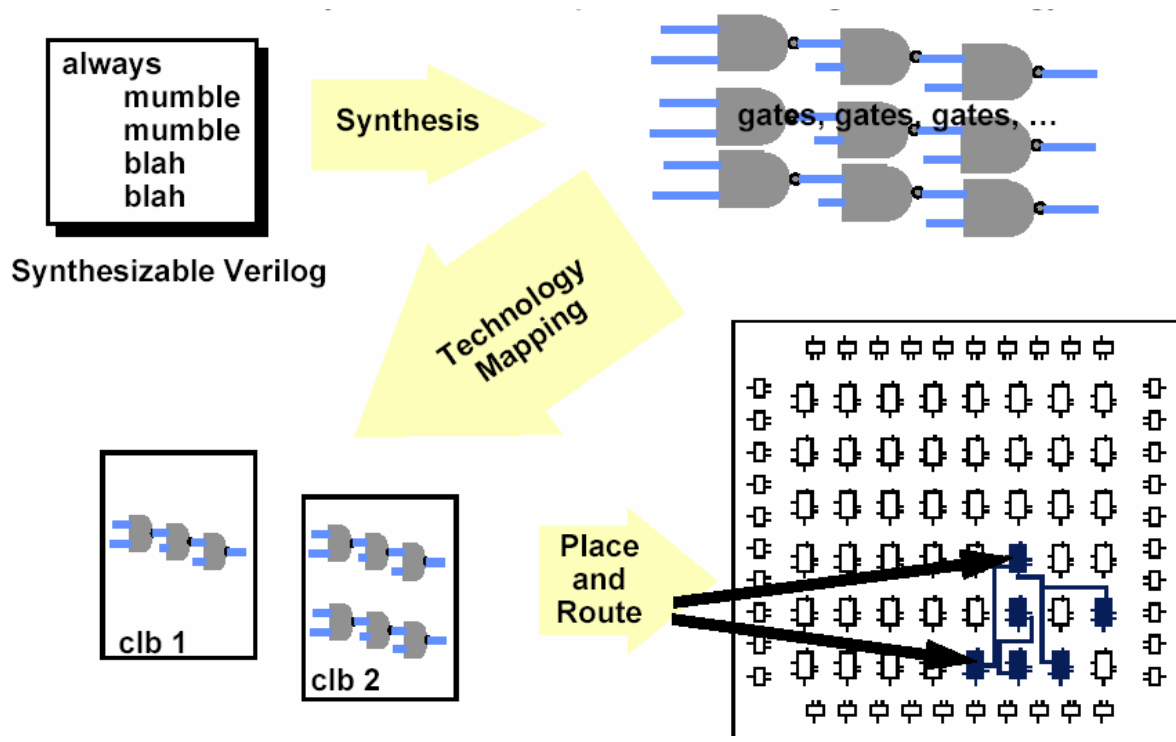
Χρησιμότητα της Verilog

- Υψηλού επιπέδου περιγραφή της συμπεριφοράς του συστήματος με σκοπό την εξομοίωση.
- RTL περιγραφή της λειτουργίας του συστήματος με σκοπό τη σύνθεσή του σε κύκλωμα.



Μοντέρνα διαδικασία ανάπτυξης συστημάτων ASIC

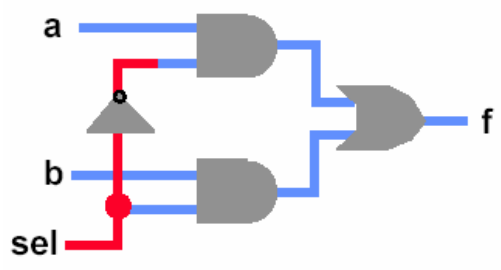
- Δημιουργία συνθέσιμου κώδικα.
- Τροφοδότηση του κώδικα σε εργαλείο σύνθεσης το οποίο θα δημιουργήσει αίσίως δύο εκδόσεις υλικού.
 - Ανεξάρτητο τεχνολογίας κύκλωμα.
 - Εξαρτώμενο από την τεχνολογία κύκλωμα (technology dependent).
- Τοποθέτηση και δρομολόγηση (P&R) λογικής στο ολοκληρωμένο.
- Προγραμματισμός συσκευής FPGA ή υλοποίηση του κυκλώματος.



Μοντέλα αναπαράστασης κυκλώματος (1/2)

Δομικό μοντέλο (structural)

- Αναπαράσταση κυκλώματος σαν δομή αποτελούμενη από πύλες και τις γραμμές διασύνδεσης αυτών.
- Πύλες μπορεί να είναι primitive πύλες και ορισμένες από τον χρήστη μονάδες.



```
module mux (f, a, b, sel);
  output f;
  input a, b, sel;

  and #5 g1 (f1, a, nsel),
  g2 (f2, b, sel);
  or #5 g3 (f, f1, f2);
  not g4 (nsel, sel);
endmodule
```

Μοντέλα αναπαράστασης κυκλώματος (2/2)

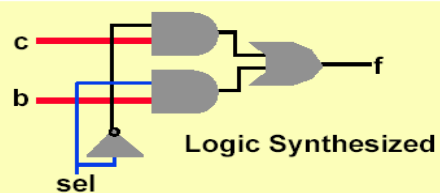
Λειτουργικό μοντέλο (behavioral)

- Το κύκλωμα περιγράφεται με βάση τη λειτουργία που θέλουμε να εκτελεί
- Μοιάζει με συνάρτηση γλώσσας προγραμματισμού

```
module mux (f, sel, b, c);  
  output f;  
  input sel, b, c;  
  reg f;  
  
  always @(sel or b or c)  
    if (sel == 1)  
      f = b;  
    else  
      f = c;  
endmodule
```

Read this as follows:
Wait for any change on a, b, or c,
then execute the begin-end block
containing the if. Then wait for
another change.

This "if" functionally describes the MUX



Γραμματικές δομές



Σχόλια / Αριθμοί

- Σχόλια:
Τα σχόλια μπορούν να δηλωθούν με δύο τρόπους (όπως στην C/C++)
- Αναπαράσταση αριθμών:
Οι αριθμοί μπορούν να αναγραφούν σε οποιοδήποτε συστήματα (δεκαδικό, δεκαεξαδικό, δυαδικό)

Κάθε αριθμός έχει την εξής τυπική μορφή:

Size ' **Base** **Value**
(πχ. `37 b 101`)



Τελεστές

- Λογικοί:
!, &&, ||, ==, !=
- Bitwise:
~, &, |, ^, ~^
- Τριαδικός
(συνθήκη) ? A : b

Τελεστές

- Συναθροιστικοί

<code>-&, - , -^</code>	NAND, NOR, EX-NOR all bits together <code>c = -& b ; d = - a ; e = ^c ;</code>
<code>-, &, , ^</code>	bit-wise NOT, AND, OR, EX-OR <code>b = -a ; // invert a</code> <code>c = b & a ; // bitwise AND a,b</code> <code>e = b a ; // bitwise OR</code> <code>f = b ^ a ; // bitwise EX-OR</code>
<code>-&, - , -^</code>	bit-wise NAND, NOR, EX-NOR <code>c = a -& b ; d = a - b ;</code> <code>e = a -^ b ;</code>

Λογικές τιμές

Λογικές τιμές

- 0 : λογικό μηδέν ή ψευδής συνθήκη
- 1 : λογικό ένα ή αληθής συνθήκη
- x : απροσδιόριστη λογική τιμή
- z : κατάσταση υψηλής εμπέδησης



Τύποι Δεδομένων

Αναπαριστούν αποθηκευτικά στοιχεία και γραμμές μετάδοσης

- Wire (καλώδια):
αναπαριστά φυσικές συνδέσεις μεταξύ δομικών μονάδων, όπως οι πύλες.
- Reg (Καταχωρητές):
αναπαριστά ένα στοιχείο αποθήκευσης.
Μια απλή δήλωση wire ή reg υπονοεί καλώδιο ή καταχωρητή πλάτους ενός bit.
Δήλωση εύρους:
wire [MSB: LSB] net_name;
wire a;
wire [15:0] bus
Reg b;
Reg [3:0] bank;
- Μνήμες:
Οι μνήμες αντιστοιχούν σε μια ακολουθία από n-bit αποθηκευτικά στοιχεία.
reg [7:0] memory [1:256];



Τύποι Δεδομένων

- Parameters
Αντιστοίχιση μιας σταθεράς σε ένα αναγνωριστικό
parameter msb=7;
- Υπάρχουν επίσης τύποι δεδομένων που συναντάμε σε γλώσσες υψηλού επιπέδου όπως integer και time.

Μονάδες λογικής (Modules)

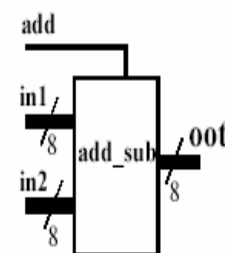
- Τι είναι;
Module είναι μια μονάδα λογισμικού διαθέτει εισόδους, εξόδους και μπορεί να είναι συνδιαστικό ή ακολουθιακό κύκλωμα.
- Δήλωση module

Syntax

```
module module_name (port_list);  
  input [msb:lsb] input_port_list;  
  output [msb:lsb] output_port_list;  
  inout [msb:lsb] inout_port_list;  
  ... statements ...  
endmodule
```

Example 7.1

```
module add_sub(add, in1, in2, oot);  
  input add; // defaults to wire  
  input [7:0] in1, in2; wire in1, in2;  
  output [7:0] oot; reg oot;  
  ... statements ...  
endmodule
```



Μονάδες λογικής (Modules)

Δημιουργία στιγμιοτύπων ενός module

- Η δήλωση στιγμιοτύπων είναι ένα πρότυπο από το οποίο μπορούμε να δημιουργήσουμε πραγματικά αντικείμενα (στιγμιότυπα).
- Οι θύρες του στιγμιοτύπου πρέπει να ταιριαστούν με τις θύρες εκείνες που ορίζονται στο module. Αυτό γίνεται με δύο τεχνικές:
 - μέσω του ονόματος της θύρας του module, χρησιμοποιώντας τελεία (.)
.όνομα_θύρας_module(όνομα_καλωδίου_σύνδεσης_με_θύρα)
 - μέσω θέσης της θύρας του module
and4 instance0 (καλώδιο_σύνδεσης_με_1η_θύρα, καλώδιο_σύνδεσης_με_2η_θύρα);

Syntax for Instantiation

```
module_name  
  instance_name_1 (port_connection_list),  
  instance_name_2 (port_connection_list),  
  .....  
  instance_name_n (port_connection_list);
```

Example 7.3

```
// MODULE DEFINITION  
  
module and4(a, b, c);  
  input [3:0] a, b;  
  output [3:0] c;  
  assign c = a & b;  
endmodule
```

// MODULE INSTANTIATIONS

```
wire [3:0] in1, in2;  
wire [3:0] o1, o2;  
/* C1 is an instance of module and4  
C1 ports referenced by position */  
and4 C1 (in1, in2, o1);  
/* C2 is another instance of and4.  
C2 ports are referenced to the  
declaration by name. */  
and4 C2 (.c(o2), .a(in1), .b(in2));
```



Χρονικός Έλεγχος

- Χρονικός έλεγχος με καθυστέρηση (delay control)
 - Ορίζει τη χρονική καθυστέρηση πριν την εκτέλεση μιας πρότασης.
 - Η προδιαγραφή ξεκινά με το σύμβολο #.`#Δt` πρόταση;
 - Χρονικός έλεγχος με συμβάντα (event control)
 - Προκαλεί την εκτέλεση μιας πρότασης μόνο μετά την πραγματοποίηση ενός συγκεκριμένου συμβάντος.
 - Πολλαπλά γεγονότα μπορούν να συνδυαστούν με τη χρήση του τελεστή or ανάμεσά τους.
 - Η προδιαγραφή ξεκινά με το σύμβολο @.`@(posedge clk or negedge reset)` πρόταση;
- `@(A or B)` πρόταση;



Διαδικασίες: Always μπλοκ

- Πρόκειται για ένα είδος βρόγχου το οποίο εκτελείται συνεχώς.
- Τα always μπλοκ εκτελούνται παράλληλα.
- Οι προτάσεις εντός αυτών εκτελούνται συνήθως σειριακά.
- Η έναρξη εκτέλεσης ενός always μπλοκ ελέγχεται από μια λίστα με χρονικά συμβάντα.

always

`@(συμβαν_1 or συμβαν_2)` πρόταση;

π.χ.

reg A, B, clock;

always @(A or B) πρόταση;

always @(posedge clock) πρόταση;



Διαδικασίες: Initial μπλοκ

Το initial μπλοκ είναι όπως το always μπλοκ, μόνο που εκτελείται μόνο μια φορά στην αρχή της εξομοίωσης.

```
initial προταση;
```

π.χ.

```
reg a;
```

```
initial
```

```
begin
```

```
    a = 2'b00;
```

```
    #50 a = 2'b01;
```

```
    #100 a=2'b11;
```

```
end
```



Διαδικασίες: Functions

- **Functions**

- Δηλώνονται μέσα σε modules και μπορούν να κληθούν από continuous αναθέσεις ή always μπλοκ.
- Περιγράφουν συνδυαστική λογική, όπως ακριβώς και μια continuous ανάθεση.

π.χ.

```
function AND;
```

```
    input IN1, IN2;
```

```
    AND = IN1 & IN2;
```

```
endfunction
```

```
module foo();
```

```
    δηλώσεις μεταβλητών;
```

```
    assign OUT = AND (IN1,IN2);
```

```
endmodule
```

Διαδικασίες: Tasks & System tasks & UDPs

- **Tasks**

Ένα task είναι παρόμοιο με μια function, έχει όμως θύρες εισόδου και εξόδου.

- **System tasks**

Υπάρχουν tasks συστήματος, τα οποία χρησιμοποιούνται κατά την εξομοίωση και εκτελούν λειτουργίες IO μεταξύ αρχείου/οθονής και μεταβλητών.

Τα ονόματά τους αρχίζουν με το σύμβολο \$.

π.χ. \$display(a,b);

\$monitor(a,b);

- **UDPs**

Δίνεται η δυνατότητα στο χρήστη να ορίσει τις δικές του primitive πύλες (συνδυαστικής ή ακολουθιακής λογικής).

primitive AND(OUT, IN1, IN2);

output OUT;

input IN1, IN2;

table

// IN1 IN2 OUT

0 0 : 0

0 1 : 0

1 0 : 0

1 1 : 1

Structural Modeling: Gate & Switch Level

- Αναπαράσταση κυκλώματος σαν δομή αποτελούμενη από primitive πύλες ή τρανζίστορ και τις γραμμές διασύνδεσης αυτών.

- Τέτοια είναι:

and, or, not, buf, bufif0 (tri-state bufs), pullup, pulldown, nmos, ...

- Κατά τη δημιουργία στιγμιοτύπων ορίζεται η:

- καθυστέρηση διάδοσης της πύλης
- δύναμη οδήγησης των γραμμών εξόδου.

π.χ.

and (strong0, strong1) #2.2 instance(OUT, IN2, IN1);

Behavioral περιγραφή κυκλώματος

Στην behavioral περιγραφή ενός κυκλώματος χρησιμοποιούνται δομές ανάθεσης, οι οποίες αφού μετασχηματίσουν κάποιες μεταβλητές, αποθηκεύουν το αποτέλεσμα σε κάποιον καταχωρητή ή οδηγούν κάποιο καλώδιο.

- Υπάρχουν δύο ειδών δομές ανάθεσης, οι procedural και οι continuous
- Οι continuous αναθέσεις οδηγούν μεταβλητές-καλώδια και η τιμή τους ανανεώνεται οποτεδήποτε κάποια από τις εισόδους αλλάζει τιμή.

wire OUT;
assign OUT = IN1 & IN2

- Στις procedural αναθέσεις, τα περιεχόμενα των καταχωρητών ανανεώνονται υπό τον έλεγχο χρονικών δομών, όπως χρονική καθυστέρηση και χρονικά συμβάντα.

always @(IN1 or IN2) OUT = IN1 & IN2

Αναθέσεις με καθυστέρηση

- Στην κανονική ανάθεση με καθυστέρηση, περνούν Δt χρονικές μονάδες και εν συνεχεία η πρόταση στο δεξί μέλος αποτιμάται και ανατίθεται στο αριστερό

Σύνταξη

Δt μεταβλητή = πρόταση

- Στην εσω-ανάθεση (intra-assignment), η πρόταση στο δεξί μέλος αποτιμάται αμέσως, αλλά ανατίθεται στο αριστερό μέλος μετά από χρόνο Δt .

Σύνταξη

μεταβλητή = # Δt πρόταση

Π.χ.

#15 OUT = IN1 & IN2;

OUT = #15 (IN1 & IN2);

Procedural αναθέσεις

Η Verilog περιλαμβάνει δυο τύπους procedural αναθέσεων, blocking και non-blocking.

- Οι blocking αναθέσεις (=) εκτελούνται ακολουθιακά.
Μια δεύτερη ανάθεση δεν εκτελείται εάν δεν ολοκληρωθεί πρώτα η προηγούμενη.
π.χ. `Y = X;`
`Z = Y;`
- Μια ακολουθία από non-blocking αναθέσεις (<=) εκτελείται παράλληλα.
 - Η πρόταση στο δεξί μέρος μιας non-blocking ανάθεσης αποτιμάται όταν ολοκληρωθεί η προηγούμενη blocking ανάθεση, εάν αυτή υπάρχει.
 - Μόνο οι κανονικές αναθέσεις με καθυστέρηση μπορούν να καθυστερήσουν την έναρξη της εκτέλεσης μιας non-blocking ανάθεσηςπ.χ. `Y <= X;`
`Z <= Y; // Στο Z θα αποθηκευτεί η αρχική τιμή του Y (όχι το X)`
`B = C;`
`A <= B; // Η ανάθεση θα πραγματοποιηθεί μετά το τέλος της προηγούμενης`
`// Στο A θα αποθηκευτεί η τελευταία τιμή του B (δηλαδή το C)`

Βασικές προτάσεις

- `Begin...end`
Μπορεί να ονομαστεί με ένα αναγνωριστικό.
`begin: LABEL`
`// προτάσεις`
`end`
- `For` και `While` βρόγχοι, η συντακτική δομή τους είναι ίδια με αυτή της `C`.
- `Disable`
Η εκτέλεση μιας `disable` τερματίζει ένα block και ο έλεγχος περνά στην επόμενη πρόταση μετά από αυτό.
`begin: TERMINATE`
`while (1)`
`begin`
`// προτάσεις`
`if (...) disable TERMINATE;`
`end`
`end`



Βασικές προτάσεις

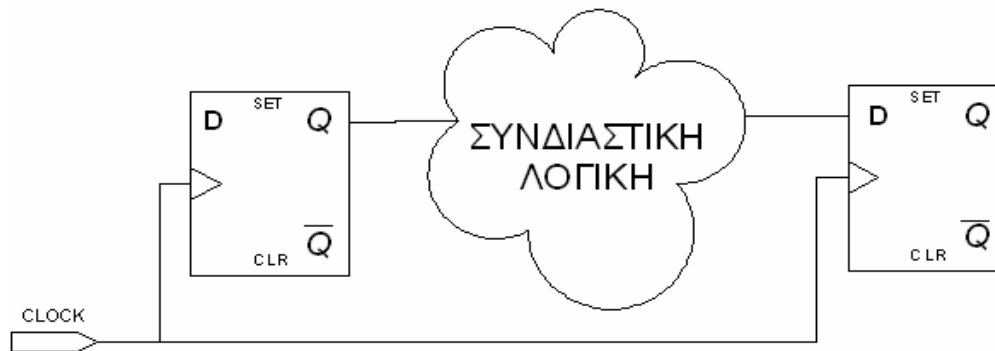
- If...else if...else
- Case
Η χρήση της είναι ίδια με αυτή της C case (μεταβλητή)
4'b0101: προταση 1
...
default: προταση n
endcase
- Casex
Στην casex οι επιλογές της μπορούν να περιέχουν τις τιμές z, x, ?, οι οποίες χρησιμοποιούνται για να δηλώσουν αδιαφορία για κάποιο bit στη σύγκριση.
4'b11xx: προταση;
- Casez
Η casez είναι ίδια με την casex, με τη διαφορά ότι μόνο οι τιμές z και ? χρησιμοποιούνται για να δηλώσουν αδιαφορία.



Verilog Synthesizable code

Γενικά Περί Σύνθεσης

- Περιορισμένο υποσύνολο της Verilog είναι συνθέσιμο.
- Αυτό ονομάζεται κώδικας RTL και αρκετά αφαιρετικά αντιστοιχεί σε κώδικα που περιγράφει τον μετασχηματισμό που υπόκειται ένας καταχωρητής κατά την αντιγραφή του σε κάποιον άλλο.



Μοντελοποιώντας συνδυαστική λογική

- Συνδυαστική λογική μπορεί να δημιουργηθεί κάνοντας χρήση continuous αναθέσεων ή always blocks.
- Στην περίπτωση του always πρέπει:
 - Στην λίστα συμβάντων (sensitivity list) να μην περιέχονται ακρο συμβάντα (posedge/negedge)
 - Εάν μια μεταβλητή reg τίθεται σε κάποιο μονοπάτι εκτέλεσης πρέπει να τίθεται και σε όλα τα υπόλοιπα.
 - Όλες οι μεταβλητές που βρίσκονται στο δεξί μέρος αναθέσεων ή σε συνθήκη προς επαλήθευση σε δομές if, elseif και case πρέπει να περιέχονται στην λίστα συμβάντων.

Π.χ. Σύνθεση πύλης AND
Wire OUT;
Assign OUT = IN1 & IN2

Reg OUT;
Always @(IN1 or IN2)
If (enable) OUT = IN1 & IN2;
Else OUT = z;



Μοντελοποιώντας Latches & Flip-Flops

- Ένα latch συμπεραίνεται όταν:
 - Έχω always και μια μεταβλητή reg που τίθεται σε κάποιο μονοπάτι εκτέλεσης αλλά όχι σε όλα.
 - Στην λίστα συμβάντων δεν υπάρχουν ακμο-συμβάντα

```
always @( D or GATE )  
if (GATE) Q = D;
```
- Ένα Flip-flop συμπεραίνεται κάθε φορά που στην λίστα συμβάντων υπάρχει ακμο συμβάν (posedge, negedge).

```
always @(posedge CLOCK)  
Q <= D;
```



Μοντελοποιώντας Πολυπλέκτες

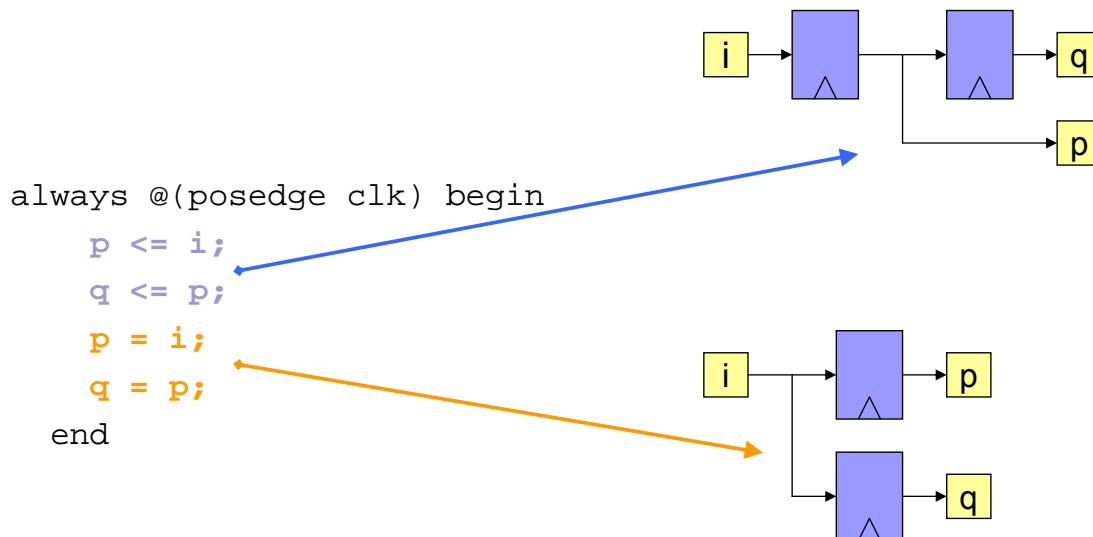
Πολυπλέκτες μπορούν να προκύψουν κάνοντας χρήση δομών if...else_if..else, case και του τριαδικού τελεστή (?:)

```
always @(SELECTION or IN1 or IN2 or IN3 or IN4)  
if (SELECTION==2'b00) OUT = IN1;  
else if (SELECTION==2'B01) OUT = IN2;  
else if (SELECTION==2'B10) OUT = IN3;  
else OUT = IN4;
```

```
always @(SELECTION or IN1 or IN2 or IN3 or IN4)  
case (SELECTION)  
2'b00: OUT = IN1;  
2'B01: OUT = IN2;  
2'B10: OUT = IN3;  
2'b11: OUT = IN4;
```

```
assign OUT = SELECTION[1] ? (SELECTION[0] ? IN4 : IN3) : (SELECTION[0] ? IN2  
: IN1);
```

Σύνθεση “=” και “<=”



Καθοδήγηση για σύνθεση

- Στις continuous αναθέσεις και αναθέσεις εντός always χωρίς posedge/negedge στην λίστα συμβάντων χρησιμοποιώ των τελεστή “=”. Στις άλλες περιπτώσεις τον “<=”
- Η συγγραφή RTL κώδικα πρέπει να γίνεται πάντα έχοντας κατά νου του υλικό που εν δυνάμει θα προκύψει.
- Αν είναι δυνατόν είναι χρήσιμο ο σχεδιασμός να χωρίζεται σε ακολουθιακό και συνδιαστικό.
- Υπάρχουν πολλοί τρόποι να περιγράψει ένα κύκλωμα, προτιμότερος είναι εκείνος που καταλαβαίνουμε καλύτερα
- Προσοχή στα ελλιπή if...elseif...else διότι δημιουργούν latches τα οποία μπορεί να είναι περιττά.
- Προσοχή στα εργαλεία σύνθεσης, συνθέτουν ετερόκλιτα κυκλώματα ή και συνήθως λανθασμένα
- Σε μερικές περιπτώσεις η εταιρία κατασκευής FPGA μπορεί να παρέχει έτοιμες λογικές δομές (macros) κατά πολύ καλύτερες από αυτό που εμείς σχεδιάσαμε.