Γλώσσες Περιγραφής Υλικού I

Μοντελοποίηση συνδυαστικών κυκλωμάτων

Νικόλαος Καββαδίας
nkavv@uop.gr

06 Μαρτίου 2012

Σκιαγράφηση της διάλεξης

- Περισσότερα για τα αρθρώματα
- Αναθέσεις και τελεστές
- Συντρέχων κώδικας
  - Η δήλωση `assign`
- Ακολουθιακός κώδικας
  - Μπλοκ λογικής `initial` και `always`
  - Δομές επιλογής: `if` και `case`
  - Δομές επανάληψης: `for` και `while`
- Παραδείγματα σχεδιασμού κυκλωμάτων: πολυπλέκτης, τρισταθής απομονωτής, αποκωδικοποιητής, κωδικοποιητής προτεραιότητας, αθροιστής απρόσημων ακεραίων, αριθμητική-λογική μονάδα (ALU)

Η οργάνωση ενός αρθρώματος

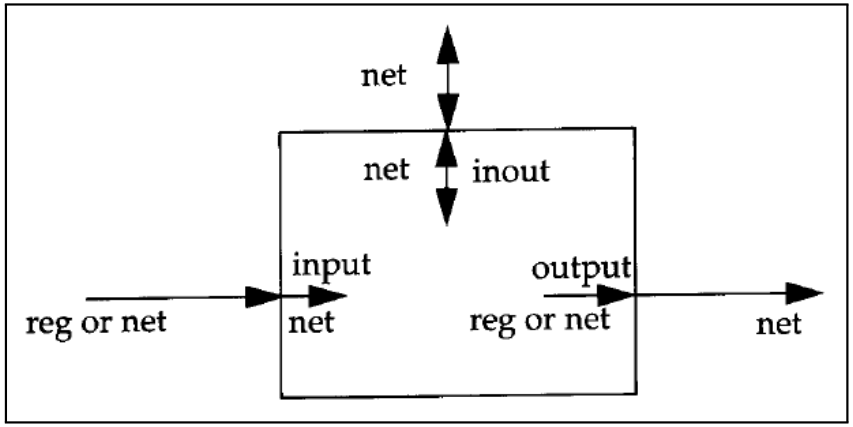
- Το module αποτελεί το βασικό στοιχείο ιεραρχικής σχεδίασης στη Verilog

Οργάνωση ενός module:

```
module <module-name> (<module-terminal-list>);
  <input port list>
  <output port list>
  [<inout port list>]
  [<wire object list>]
  [<reg object list>]
  [<parameter list>]

  // Implementation <module internals>
  [<initial logic blocks>]
  [<always logic blocks>]
  [<concurrent assignment list>]
  [<module instances>]
  [<functions and/or tasks>]
endmodule
```

Τύποι θυρών σε ένα `module`

- Ένα module διαθέτει θύρες για την επικοινωνία του με το περιβάλλον
- Τα αρχεία δοκιμής (testbenches) αποτελούν εικονικά κυκλώματα στα οποία δεν δηλώνονται θύρες
- Οι τύποι θυρών δηλώνονται με τις εξής λέξεις-κλειδιά
  - `input`: Θύρα εισόδου
  - `output`: Θύρα εξόδου
  - `inout`: Δικατευθυντική θύρα (για είσοδο και έξοδο)



- Οι θύρες είναι προκαθορισμένες ως `wire`. Αν η έξοδος αποθηκεύει την τιμή της δηλώνεται ως `reg`

Σύνδεση ενός `module` με εξωτερικά σήματα

- Υπάρχουν δύο τρόποι για τη σύνδεση ενός αντιτύπου module με εξωτερικά σήματα
  - Σύνδεση με διατεταγμένη λίστα (ordered list)
  - Σύνδεση κατ΄ όνομα (by name)

```
module fa4(a, b, ci, s, co);
reg [3:0] sum;
reg cout;
wire [3:0] ain, bin;
wire cin;
```

- Ordered list: τα ονόματα των σημάτων δηλώνονται με την ίδια σειρά με την οποία εμφανίζονται ως θύρες

```
fa4 test(ain, bin, cin, sum, cout);
```

- By name: το συμβολικό όνομα της θύρας αντιστοιχίζεται με το όνομα του συγκεκριμένου σήματος

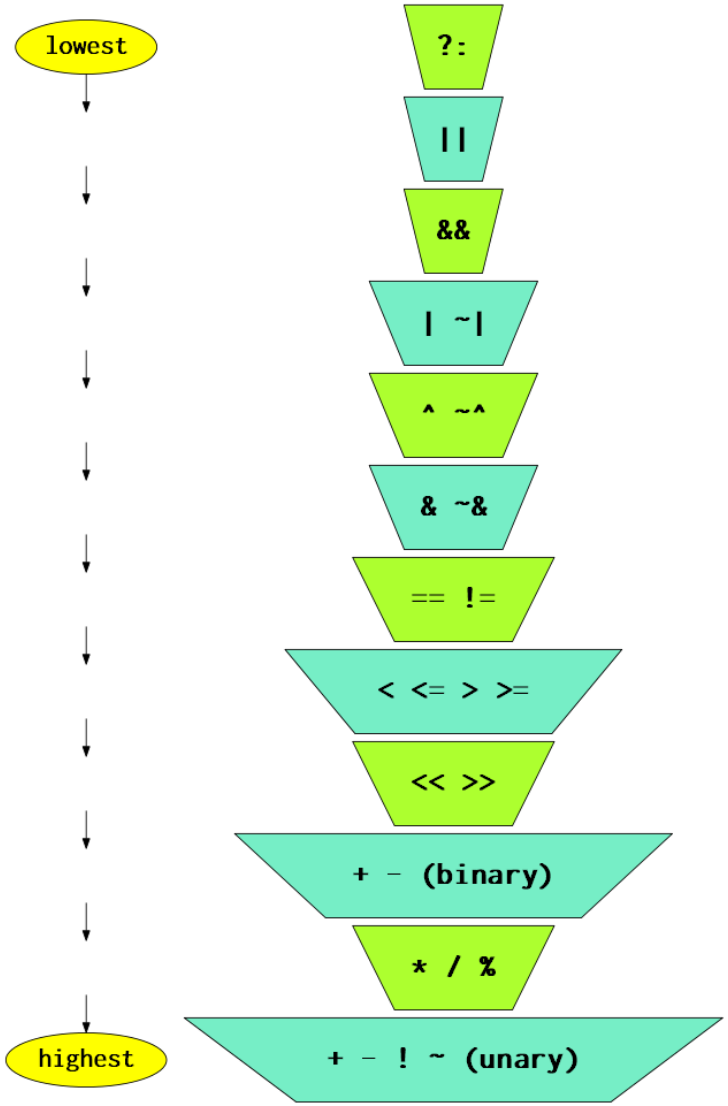
```
fa4 test(.a(ain), .b(bin), .ci(cin), .s(sum), .co(cout));
```

Τελεστές της Verilog (Verilog operators)

Συνοπτικός πίνακας των τελεστών

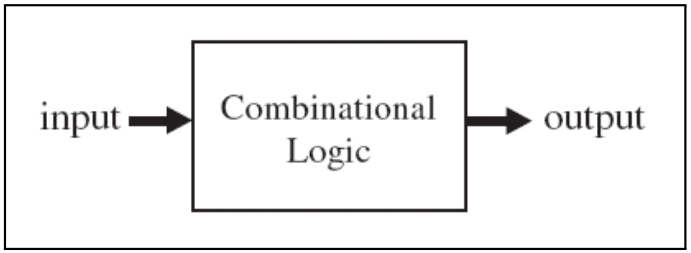
λογικοί τελεστές τύπου bitwise τύπου wordwise	and & &&	or | ||	nand ~& ! (negation)	nor ~|	xor ^	xnor ~^	not ~
αριθμητικοί	+	-	*	/	%		
σύγκρισης	==	!=	<	<=	>	>=	
ολίσθησης	<<	>>					
μοναδιαίοι	+	-					
άλλοι	επιλογή ?:	συνένωση {}	κλωνοποίηση {{}}	event or or			

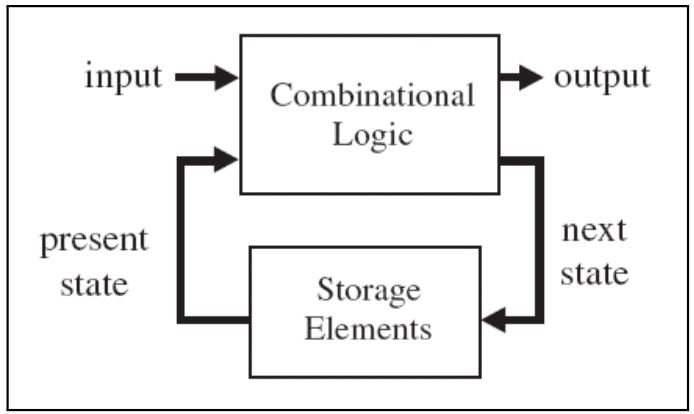
Προτεραιότητα τελεστών



Συνδυαστική και ακολουθιακή λογική

- Θεμελιώδεις τρόποι οργάνωσης των ψηφιακών κυκλωμάτων: συνδυαστική (combinational) και ακολουθιακή (sequential) λογική
- **Συνδυαστική λογική**: η έξοδος του κυκλώματος εξαρτάται αποκλειστικά από τις τρέχουσες εισόδους
- **Ακολουθιακή λογική**: η έξοδος του κυκλώματος εξαρτάται από τις τρέχουσες εισόδους και την τρέχουσα κατάσταση





Συντρέχων και ακολουθιακός κώδικας

- Στη Verilog ο κώδικας είναι από τη φύση του παράλληλα εκτελούμενος (συντρέχων)
- Η Verilog διαθέτει ειδικές προγραμματιστικές δομές για την περιγραφή τμημάτων ακολουθιακού κώδικα, προκειμένου την εξασφάλιση της διαδοχικής εκτέλεσης εντολών όταν αυτό είναι επιθυμητό
- Ακολουθιακός κώδικας στη Verilog: μέσα σε μπλοκ `initial` ή `always` (επίπεδο behavioral)
- Συντρέχων κώδικας
  - Σε επίπεδο dataflow με δηλώσεις `assign`
  - Σε επίπεδο structural με διασύνδεση αντιτύπων `module`
- Στον ακολουθιακό κώδικα χρησιμοποιούνται δομές επανάληψης, ελέγχου και επιλογής παρόμοιες με της C
- Ένα μπλοκ λογικής υλοποιεί μία διεργασία η οποία εκτελείται ΠΑΡΑΛΛΗΛΑ σε σχέση με τυχόν άλλες διεργασίες

Συντρέχων κώδικας: Η δήλωση `assign`

- Η δήλωση `assign` αποτελεί τη βασικότερη δήλωση για τη σχεδίαση στο επίπεδο dataflow
- Με τη χρήση της δηλώνεται η οδήγηση ενός σήματος σε έναν κόμβο
- Πηγή προβλημάτων στην κατανόηση της `assign` αποτελεί το γεγονός ότι επιτρέπεται η χρήση της και εντός μπλοκ `initial` και `always`. Την assign αυτού του τύπου την ονομάζουμε procedural (διαδικαστική) `assign` και δεν θα τη χρησιμοποιούμε
- Σύνταξη της `assign`:
  `assign` `<net-name> = [drive-strength] [delay] <expression>;`
- Η ανάθεση γίνεται σε `output`, `inout` ή `wire`
- Η έκφραση στο δεξί μέλος μπορεί να περιλαμβάνει `reg`, δικτυώματα ή και κλήσεις διαδικασιών ή συναρτήσεων

Παράδειγμα χρήσης assign

- Πολυπλέκτης 4-σε-1 με δήλωση assign

```
module mux4_to_1 (outp, i0, i1, i2, i3, s1, s0);
output outp;
input i0, i1, i2, i3;
input s1, s0;

assign outp = (~s1 & ~s0 & i0) |
              (~s1 & s0 & i1)  |
              (s1 & ~s0 & i2)  |
              (s1 & s0 & i3)   ;

endmodule
```

- Τη δήλωση assign θα τη χρησιμοποιούμε μόνο εκτός των μπλοκ λογικής initial και always

Ακολουθιακός κώδικας

- Ο ακολουθιακός κώδικας στη Verilog συντάσσεται εντός μπλοκ λογικής `initial` και `always`
- Τα μπλοκ αυτά αποτελούν διεργασίες, οι οποίες ενεργοποιούνται από γεγονότα (events)
- Η συμπεριφορά μιας διεργασίας προσομοιώνεται βήμα προς βήμα (δήλωση προς δήλωση)
- Στον ίδιο χρόνο προσομοίωσης, επιτρέπεται να είναι ενεργές (active) περισσότερες από μία διεργασίες
- Οι διεργασίες αυτές περιλαμβάνουν
  - Blocking/non-blocking αναθέσεις
  - Δομές ελέγχου: `if-else` και `if-else if-...-else`
  - Δομές επιλογής: `case-endcase`
  - Δομές επανάληψης: `for-begin-end` και `while-begin-end`

Τα μπλοκ λογικής `initial` και `always`

```
initial
  begin
    // statements
  end
```

- Εκτελείται όταν αρχίζει η προσομοίωση (από time unit 0)
- Τερματίζεται όταν ο έλεγχος φτάσει στο τέλος του μπλοκ (στο end)
- Χρησιμοποιείται για την αρχικοποίηση μνημών, και σε αρχεία δοκιμής για τη δημιουργία σειράς τιμών εισόδου

```
always [@(<sensitivity-list>)]
  begin
    // statements
  end
```

- Εκτελείται όταν αρχίζει η προσομοίωση (από time unit 0)
- Επανεκκινεί κάθε φορά που ο έλεγχος φτάσει στο τέλος του μπλοκ
- Η επανεκκίνηση εξαρτάται είτε από το αν έχουν μεταβληθεί οι τιμές των σημάτων ευαισθησίας είτε προγραμματίζεται να συμβεί μετά από κάποιο χρόνο
- Χρησιμοποιείται στο σχεδιασμό τόσο συνδυαστικών όσο και ακολουθιακών κυκλωμάτων

Παραδείγματα χρήσης των `initial` και `always`

- Η παρακάτω `initial` εκτελείται μία φορά πραγματοποιώντας αναθέσεις κατά τις χρονικές μονάδες 10 και 20

```
initial
  begin
    #10 a = 1; b = 0;
    #10 a = 0; b = 1;
  end
```

- Η παρακάτω `always` περιμένει κάποια αλλαγή στη τιμή σήματος που περιλαμβάνεται στη λίστα ευαισθησίας

```
always @(a or b)
  begin
    c = a + b;
  end
```

- Χρήση `always` για ακολουθιακά κυκλώματα (δηλώνοντας π.χ. θετική ακμοπυροδότηση)

```
always @(posedge clk)
  q = d;              // for a single assignment, begin-end are optional
```

Δομές ελέγχου σε ακολουθιακό κώδικα: Η δήλωση `if` (1)

- Η δήλωση IF αποτελεί τη θεμελιώδη δομή για την εκτέλεση κώδικα υπό συνθήκη
- Σύνταξη της IF:

```
if (<condition>)          // Type 1
  true_statements;

if (<condition>)          // Type 2
  true_statements;
else
  false_statements;

if (<condition1>)         // Type 3
  true_statements_for_condition1;
else if (<condition2>)
  true_statements_for_condition2;
...
else
  default_statements;
```

Δομές ελέγχου σε ακολουθιακό κώδικα: Η δήλωση `if` (2)

- Παράδειγμα για δήλωση IF τύπου 1

```
if (!lock) buffer = data;
if (enable)
  out = in;
```

- Παράδειγμα για δήλωση IF τύπου 2

```
if (number_queued < MAX_Q_DEPTH)
  begin
    data_queue = data;
    number_queued = number_queued + 1;
  end
else
  $display("Queue full. Try again.");
```

- Παράδειγμα για δήλωση IF τύπου 3

```
if (opcode == 0)
  y = x + z;
else if (opcode == 1)
  y = x - z;
else if (opcode == 2)
  y = x * z;
else
  $display("Invalid operation code.");
```

Δομές ελέγχου σε ακολουθιακό κώδικα: Η δήλωση case (1)

- Η δήλωση CASE χρησιμοποιείται για την πραγματοποίηση μίας επιλογής μέσα από πολλές περιπτώσεις
- Η δήλωση CASE αποτελεί μία χρήσιμη εντολή για την περιγραφή δομών αποκωδικοποίησης (decoding) μέσα σε τμήματα ακολουθιακού κώδικα
- Η γενική σύνταξη της CASE:

```
case (<expression>)
  alternative1: statements1;
  alternative2: statements2;
  ...
  default: default_statements;   // optional
endcase
```

- Χρησιμοποιείται αντί της if-else if-... με πολλαπλές διακλαδώσεις
- Αντιστοιχεί στην switch-case της ANSI C

Δομές ελέγχου σε ακολουθιακό κώδικα: Η δήλωση case (2)

- Παράδειγμα 1: Υλοποίηση της IF τύπου 3 με CASE (επιτρέπεται για επιλογές χωρίς προτεραιότητα)

```
reg [1:0] opcode;
...
case (opcode)
  2'd0   : y = x + z;
  2'd1   : y = x - z;
  2'd2   : y = x * z;
  default: $display("Invalid operation code.");
endcase
```

- Παράδειγμα 2: Πολυπλέκτης 4-σε-1 με CASE

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0)
  output out;
  input i0, i1, i2, i3, s1, s0;
  reg out;

  always @(s1 or s0 or i0 or i1 or i2 or i3)
    case ({s1, s0}) // concatenation of s1, s0 to a 2-bit signal
      2'd0: out = i0;
      2'd1: out = i1;
      2'd2: out = i2;
      default: out = i3;
    endcase
endmodule
```

Δομές επανάληψης: Η δήλωση `for` (1)

- Η Verilog διαθέτει τέσσερις δομές επανάληψης, τις δηλώσεις FOR, WHILE, REPEAT, και FOREVER
- ☞ Θα χρησιμοποιούμε μόνο τις δηλώσεις FOR και WHILE
- Η δήλωση FOR αποτελείται από τρία μέρη:
  1. Μία αρχική συνθήκη
  2. Μία συνθήκη για τον έλεγχο τερματισμού
  3. Τουλάχιστον μία διαδικαστική ανάθεση για τη μεταβολή της τιμής της μεταβλητής ελέγχου (π.χ. δείκτης βρόχου)
- Η γενική σύνταξη της FOR:

```
for (<initial-condition>; <terminating-condition>; <index-assignment>)
  <statements>
```

Δομές επανάληψης: Η δήλωση for (2)

- Παράδειγμα 1: Δημιουργία ακολουθίας ακεραίων

```
reg [3:0] i, out1;
...
for (i = 0; i <= 15; i = i + 1)
begin
  out1 = i;
  #10;
end
```

- Παράδειγμα 2: Αρχικοποίηση θέσεων σε μία μνήμη

```
integer state[0:31];
integer i;
...
initial
  begin
    for (i = 0; i < 32; i = i + 2)    // even addresses
      state[i] = 0;
    for (i = 1; i < 32; i = i + 2)    // odd addresses
      state[i] = 1;
  end
```

Δομές επανάληψης: Η δήλωση `while` (1)

- Η δήλωση WHILE εκτελείται μέχρις ότου η έκφραση ελέγχου γίνει ψευδής
- Αν η έκφραση ελέγχου είναι ψευδής εξαρχής (δηλ. κατά την πρώτη χρήση της) τότε το σώμα του βρόχου WHILE δεν εκτελείται καμία φορά
- Η γενική σύνταξη της WHILE:

```
while (<test-condition>)
  <statements>
```

- Η μεταβλητή ελέγχου (π.χ. δείκτης βρόχου) πρέπει να έχει αρχικοποιηθεί πριν την είσοδο στη WHILE

Δομές επανάληψης: Η δήλωση `while` (2)

- Παράδειγμα 1: Δημιουργία ακολουθίας ακεραίων

```
reg [3:0] i, out1;
...
i = 0;
while (i <= 15)
begin
  out1 = i;
  #10 i = i + 1;
end
```

- Παράδειγμα 2: Εύρεση του πρώτου bit με την τιμή 1

```
reg [15:0] flag;
integer i;
reg continue;
initial begin
  flag = 16'b0010_0000_0000_0000;
  i = 0;
  continue = 1;
  while ((i < 16) && continue) begin
    if (flag[i]) begin
      $display("Encountered a '1' at position %d: i);
      continue = 0;
    end
    i = i + 1;
  end
end
```

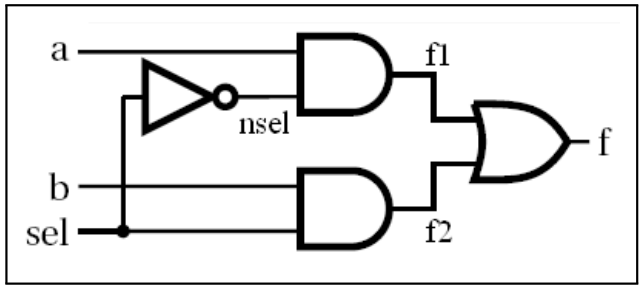
Απλά συνδυαστικά κυκλώματα: Πολυπλέκτης 2-σε-1

- Τρόποι περιγραφής ενός απλού συνδυαστικού κυκλώματος
  - Με προκαθορισμένα module
  - Με δήλωση `assign`
  - Με μπλοκ `always` και διαδικαστικό ακολουθιακό κώδικα
- Παράδειγμα: Πολυπλέκτης 2-σε-1 για σήματα του 1-bit

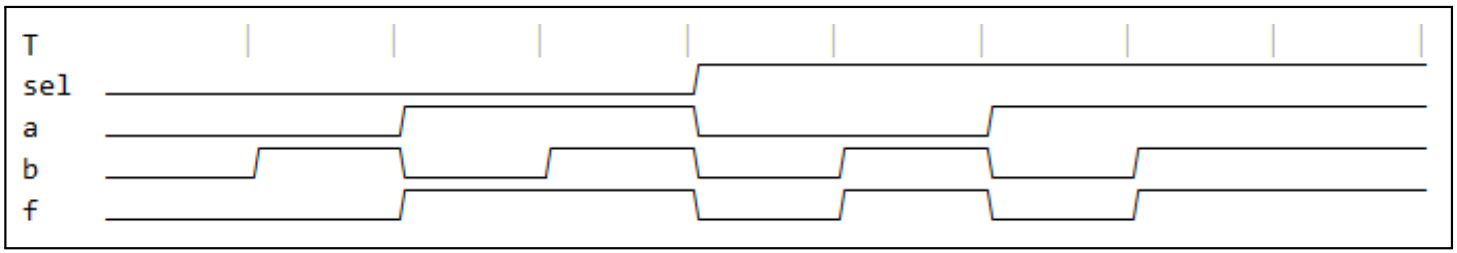
- Πίνακας αληθείας

sel	f
0	a
1	b

- Σχηματικό διάγραμμα



- Διάγραμμα χρονισμού του κυκλώματος



Πολυπλέκτης 2-σε-1 με προκαθορισμένα module

```
// Verilog programs built from modules
module mux2to1(f, a, b, sel);
// Each module has an interface
  output f;
  input a, b, sel;
// Internal signals (even though it is optional,
// do declare them)
  wire f1, f2, nsel;

// Module may contain structure: instances of primitives
// and other modules
  and g1(f1, a, nsel);
  and g2(f2, b, sel);
  or  g3(f, f1, f2);
  not g4(nsel, sel);

endmodule
```

Πολυπλέκτης 2-σε-1 με συντρέχουσα δήλωση assign

```
module mux2to1(f, a, b, sel);
  output f;
  input a, b, sel;

// The concurrent assignment to output f
  assign f = sel ? b : a;

endmodule
```

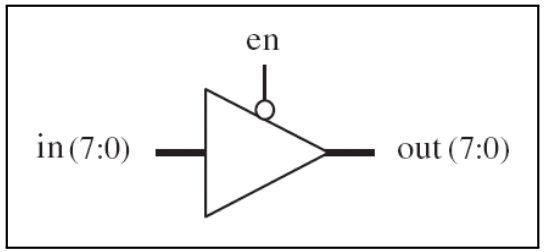
Πολυπλέκτης 2-σε-1 με μπλοκ always

```
module mux2to1(f, a, b, sel);
  output f;
  input a, b, sel;
// Needed when assigned in a sequential logic block
// A reg behaves like memory: holds its value until
// imperatively assigned otherwise
  reg f;

// Modules may contain one or more always blocks
// Sensitivity list contains signals whose change makes the
// block execute
  always @(a or b or sel)
// Body of an always block contains traditional imperative code
  begin
    if (sel)
      f = b;
    else
      f = a;
  end
endmodule
```

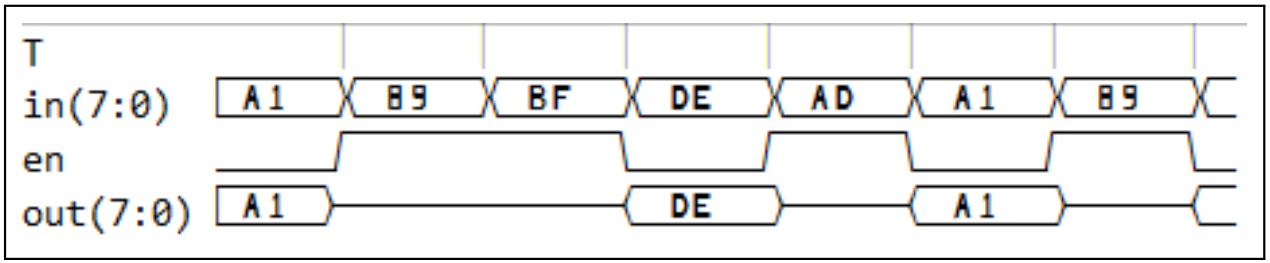
Τρισταθής απομονωτής (tristate buffer)

- Στον τρισταθή απομονωτή η έξοδος ισούται με την είσοδο όταν το σήμα επίτρεψης είναι `en = 0` αλλιώς η έξοδος οδηγείται σε κατάσταση υψηλής αντίστασης (high impedance state) λόγω της μη οδήγησής της

```
module tristate(in, en, out);
  input [7:0] in;
  input en;
  output [7:0] out;

  assign out = (en == 1'b0) ? in : {8{1'bZ}};

endmodule
```



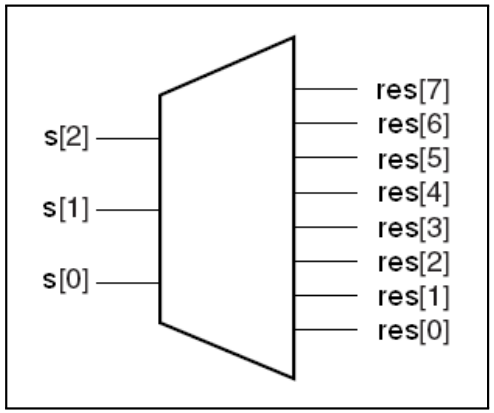
- Διάγραμμα χρονισμού του κυκλώματος



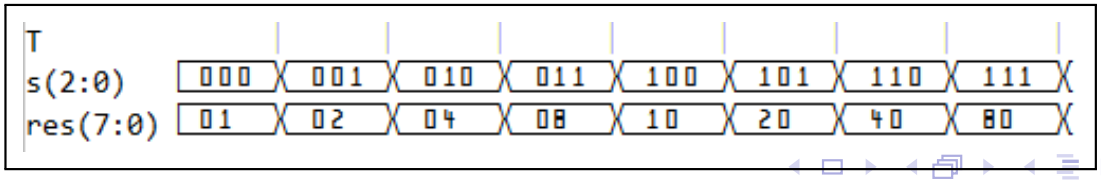
Αποκωδικοποιητής 3-σε-8 (3-to-8 decoder)

- Αποκωδικοποίηση εισόδου από το δυαδικό στο σύστημα one-hot (μόνο ένα ψηφίο είναι ΄1΄)

```
module decoder3to8 (s, res);
  input [2:0] s;
  output [7:0] res;
  reg [7:0] res;
  always @(s)
  begin
    case (s)
      3'b000  : res = 8'b00000001;
      3'b001  : res = 8'b00000010;
      3'b010  : res = 8'b00000100;
      3'b011  : res = 8'b00001000;
      3'b100  : res = 8'b00010000;
      3'b101  : res = 8'b00100000;
      3'b110  : res = 8'b01000000;
      default : res = 8'b10000000;
    endcase
  end
endmodule
```



- Διάγραμμα χρονισμού του κυκλώματος



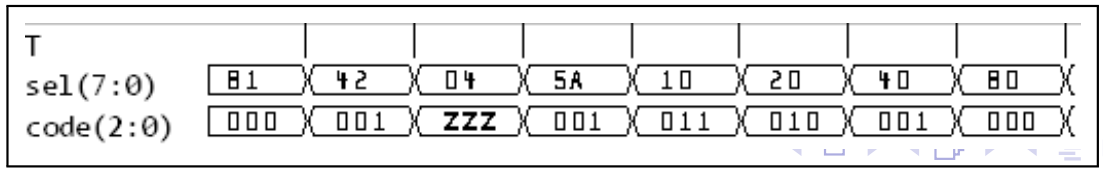
Κωδικοποιητής προτεραιότητας 8-σε-3 (8-to-3 priority encoder)

- Κύκλωμα για επιλογή κατά σειρά προτεραιότητας

sel	code
1-------	000
01------	001
001-----	010
0001----	011
00001---	100
00000---	ZZZ

```
module priority_encoder(sel, code);
  input  [7:0] sel;
  output [2:0] code;
  reg    [2:0] code;
  always @(sel)
  begin
    if (sel[7])
      code = 3'b000;
    else if (sel[6])
      code = 3'b001;
    else if (sel[5])
      code = 3'b010;
    else if (sel[4])
      code = 3'b011;
    else if (sel[3])
      code = 3'b100;
    else
      code = 3'bZZZ;
  end
endmodule
```

- Διάγραμμα χρονισμού του κυκλώματος

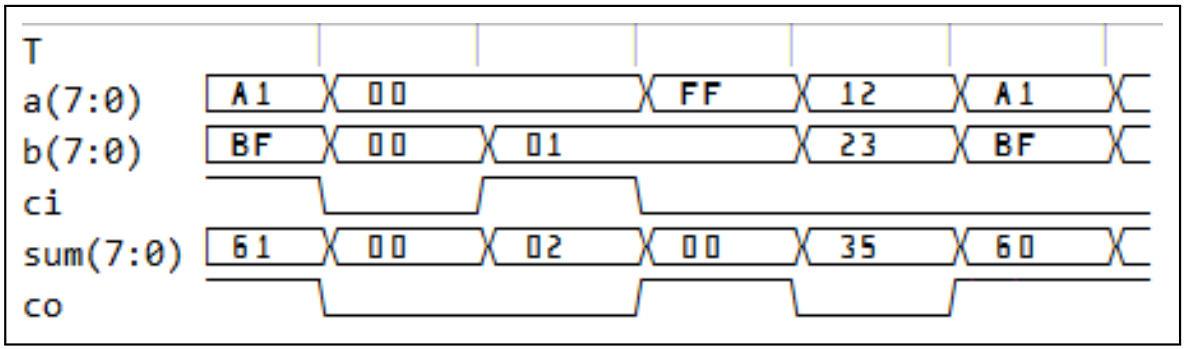


Αθροιστής απρόσημων ακεραίων με κρατούμενο εισόδου και εξόδου

- Περιγραφή

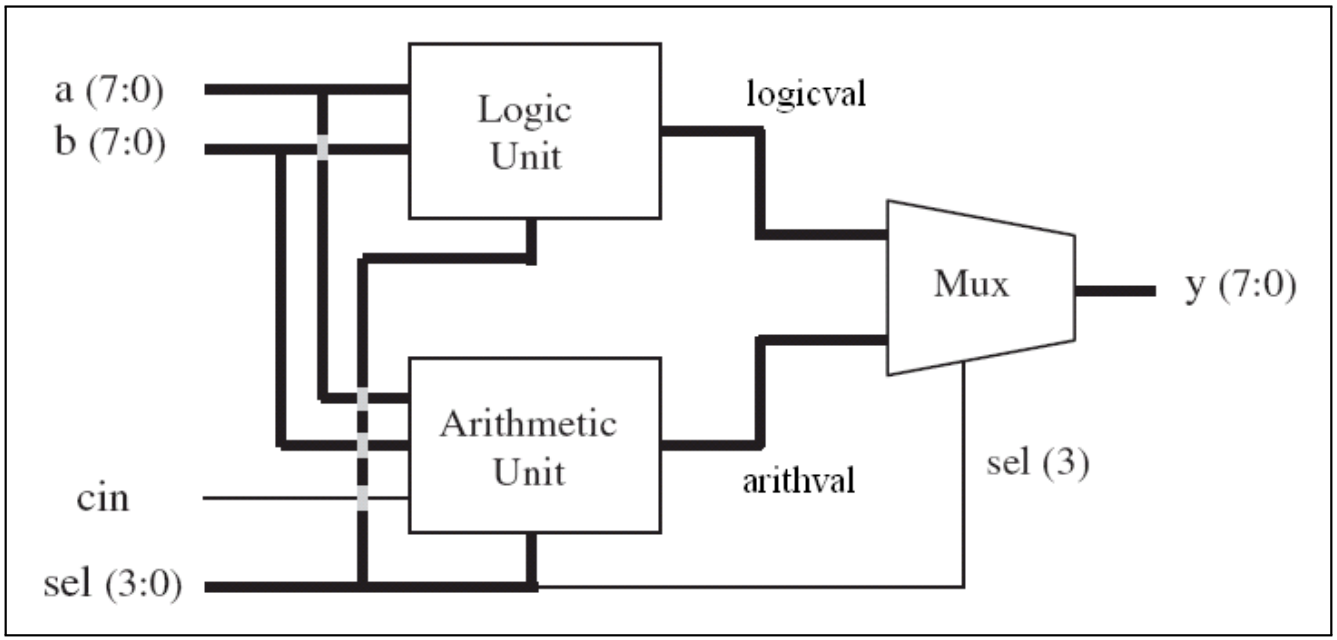
```
module adder(a, b, ci, sum, co);
  input         ci;
  input  [7:0] a;
  input  [7:0] b;
  output [7:0] sum;
  output        co;
  wire   [8:0] tmp;

  assign tmp = a + b + ci;
  assign sum = tmp [7:0];
  assign co  = tmp [8];
endmodule
```

- Διάγραμμα χρονισμού



Αριθμητική-λογική μονάδα (ALU) (1)

Σχηματικό διάγραμμα μιας ALU για έναν υποθετικό 8-bit επεξεργαστή



Αριθμητική-λογική μονάδα (ALU) (2)

- Προδιαγραφές μιας ALU για έναν υποθετικό 8-bit επεξεργαστή
- Ρεπερτόριο εντολών

Opcode	Κωδικοπ.	Πράξη	Λειτουργία
Αριθμητική μονάδα			
`MOVA`	0000	`y <= a`	Μεταφορά του a
`INCA`	0001	`y <= a + 1`	Αύξηση κατά 1 του a
`DECA`	0010	`y <= a - 1`	Μείωση κατά 1 του a
`MOVB`	0011	`y <= b`	Μεταφορά του b
`INCB`	0100	`y <= b + 1`	Αύξηση κατά 1 του b
`DECB`	0101	`y <= b - 1`	Μείωση κατά 1 του b
`ADD`	0110	`y <= a + b`	Άθροιση των a,b
`ADC`	0111	`y <= a + b + cin`	Άθροιση των a,b με κρατούμενο
Λογική μονάδα			
`NOTA`	1000	`y <= not a`	Αντιστροφή του a
`NOTB`	1001	`y <= not b`	Αντιστροφή του b
`AND`	1010	`y <= a and b`	Λογική πράξη AND
`IOR`	1011	`y <= a or b`	Λογική πράξη OR
`NAND`	1100	`y <= a nand b`	Λογική πράξη NAND
`NOR`	1101	`y <= a nor b`	Λογική πράξη NOR
`XOR`	1110	`y <= a xor b`	Λογική πράξη XOR
`XNOR`	1111	`y <= a xnor b`	Λογική πράξη XNOR

Αριθμητική-λογική μονάδα (ALU): Κώδικας (3)

```
module alu(a, b, cin, sel, y);
  input [7:0] a, b;
  input cin;
  input [3:0] sel;
  output [7:0] y;
  reg [7:0] y;
  reg [7:0] arithval;
  reg [7:0] logicval;

  // Arithmetic unit
  always @(a or b or cin or sel)
  begin
    case (sel[2:0])
      3'b000  : arithval = a;
      3'b001  : arithval = a + 1;
      3'b010  : arithval = a - 1;
      3'b011  : arithval = b;
      3'b100  : arithval = b + 1;
      3'b101  : arithval = b - 1;
      3'b110  : arithval = a + b;
      default : arithval = a + b + cin;
    endcase
  end
```

```
  // Logic unit
  always @(a or b or sel)
  begin
    case (sel[2:0])
      3'b000  : logicval = ~a;
      3'b001  : logicval = ~b;
      3'b010  : logicval = a & b;
      3'b011  : logicval = a | b;
      3'b100  : logicval = ~((a & b));
      3'b101  : logicval = ~((a | b));
      3'b110  : logicval = a ^ b;
      default : logicval = ~(a ^ b);
    endcase
  end

  // Multiplexer
  always @(arithval or logicval or sel)
  begin
    case (sel[3])
      1'b0    : y = arithval;
      default : y = logicval;
    endcase
  end
endmodule
```

Αριθμητική-λογική μονάδα (ALU) (4)

Διάγραμμα χρονισμού για την ALU