Ταχεία Πρωτοτυποποίηση Ψηφιακών Συστημάτων

Workshop #1 – Συνδυαστικά Κυκλώματα & Γνωριμία με τη Verilog

Α.Ο. Φαρμάκης

In association with the ACM Student Chapter University of Patras

Περιεχόμενα του σημερινού workshop

Επισκόπηση στο δυαδικό σύστημα μέτρησης (1/5)

Γενικώς, ένας αριθμός αναπαρίσταται με την εξής μορφή:

$$a_N \ldots a_3 a_2 a_1 a_0, a_{-1} a_{-2} a_{-3} \ldots$$

Όπως για παράδειγμα ο αριθμός $307{,}6501_{10}$ στο δεκαδικό σύστημα αρίθμησης.

Με αντίστοιχο τρόπο ορίζουμε μια οποιαδήποτε *βάση* κάποιου συστήματος αρίθμησης.

Επισκόπηση στο δυαδικό σύστημα μέτρησης (2/5)

Η *βάση* (base ή radix) ενός αριθμητικού συστήματος καθορίζει το πλήθος των διακριτών τιμών που μπορούν να χρησιμοποιηθούν για την αναπαράσταση οποιωνδήποτε αριθμητικών ποσοτήτων, όπως:

- $307{,}6501_{10}$
- $100101011101{,}110101_{2}$
- $13374{,}2173_{8}$
- $B65FCAFE_{16}$

Επισκόπηση στο δυαδικό σύστημα μέτρησης (3/5)

Υπάρχουν, προφανώς, πολλά αριθμητικά συστήματα. Γενικά όμως, ένας αριθμός που εκφράζεται σε σύστημα με βάση r έχει συντελεστές που πολλαπλασιάζονται με δυνάμεις του r:

$$a_N \cdot r^N + a_{N-1} \cdot r^{n-1} + \ldots + a_2 \cdot r^2 + a_1 \cdot r^1 + a_0 + a_{-1} \cdot r^{-1} a_{-2} \cdot r^{n-2} + \ldots + a_{-M} \cdot r^{-M}$$

Βέβαια, στο πλαίσιο αυτού του workshop θα ασχοληθούμε (σχεδόν αποκλειστικά) με το δυαδικό σύστημα.

Επισκόπηση στο δυαδικό σύστημα μέτρησης (4/5)

Οι θετικοί ακέραιοι, μαζί με το 0, μπορούν να αναπαρασταθούν ως μη προσημασμένοι αριθμοί. Τι γίνεται όμως αν χρειαζόμαστε και αρνητικούς αριθμούς;

Τα συμπληρώματα χρησιμοποιούνται στα ψηφιακά κυκλώματα για την απλοποίηση προσημασμένων πράξεων, το οποίο οδηγεί και σε πιο οικονομικά (σε μέγεθος) κυκλώματα.

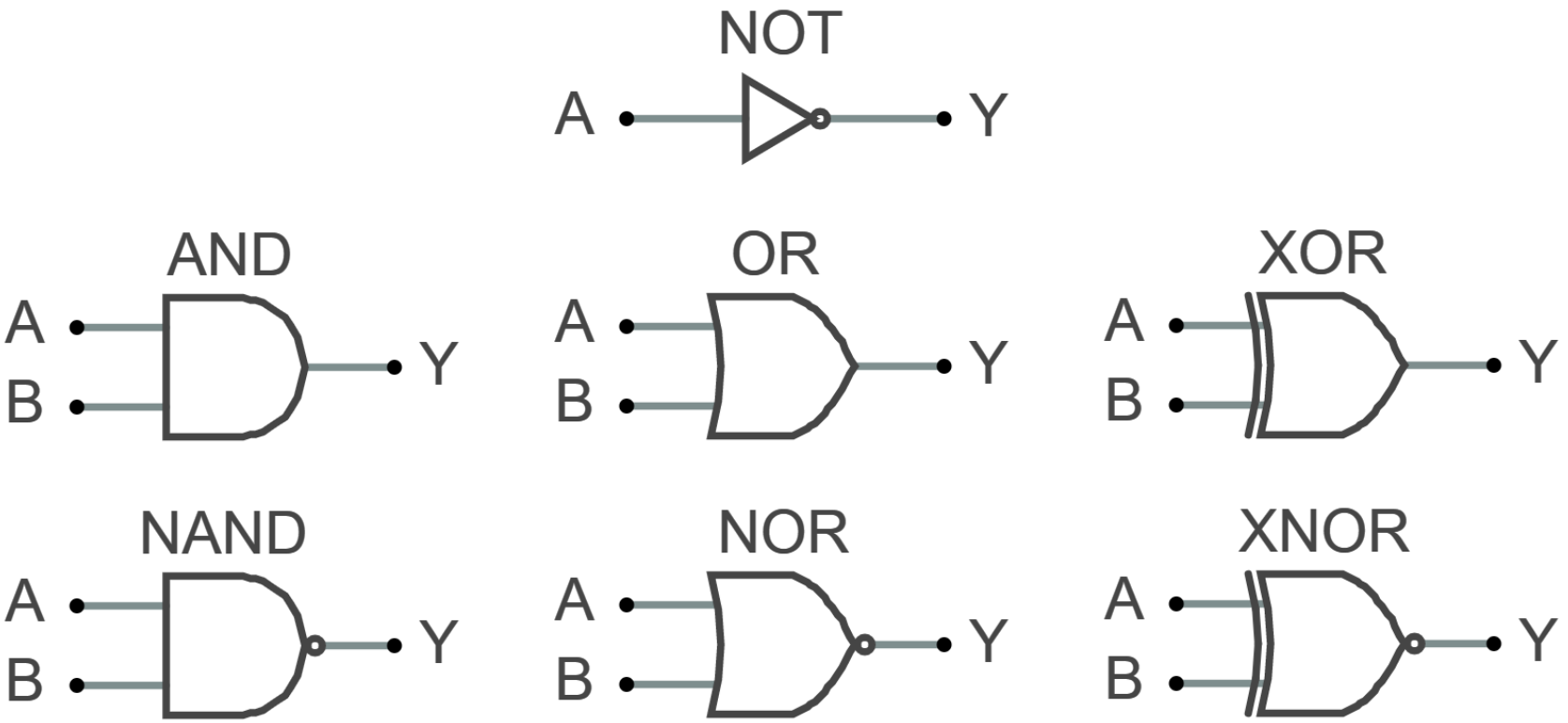
Επισκόπηση στο δυαδικό σύστημα μέτρησης (5/5)

Στο διπλανό πίνακα φαίνεται η αναπαράσταση των ακέραιων αριθμών {-8, +7} σε συμπλήρωμα ως προς 2. Παρατηρήστε το πιο σημαντικό bit (most significant bit, MSB) και τι υποδηλώνει στο κάθε νούμερο.

Δεκαδικός Αριθμός	2's Complement
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

Επισκόπηση στην άλγεβρα Boole (1/11)

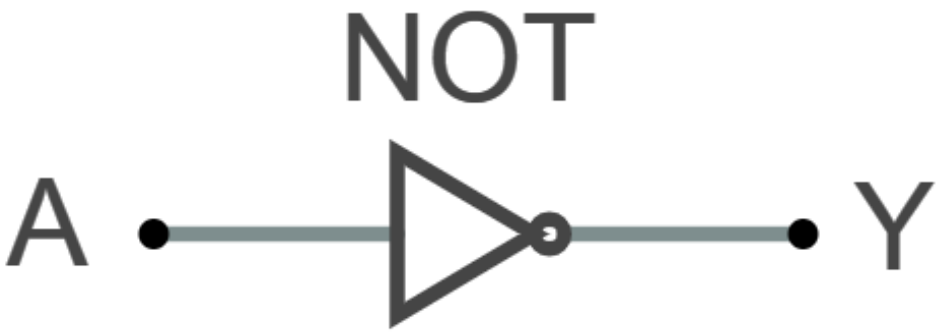
Σύμβολα λογικών πυλών



Επισκόπηση στην άλγεβρα Boole (2/11)

Πύλη “ΟΧΙ” (Αντιστροφέας)
Λογική εξίσωση: Y = ~A

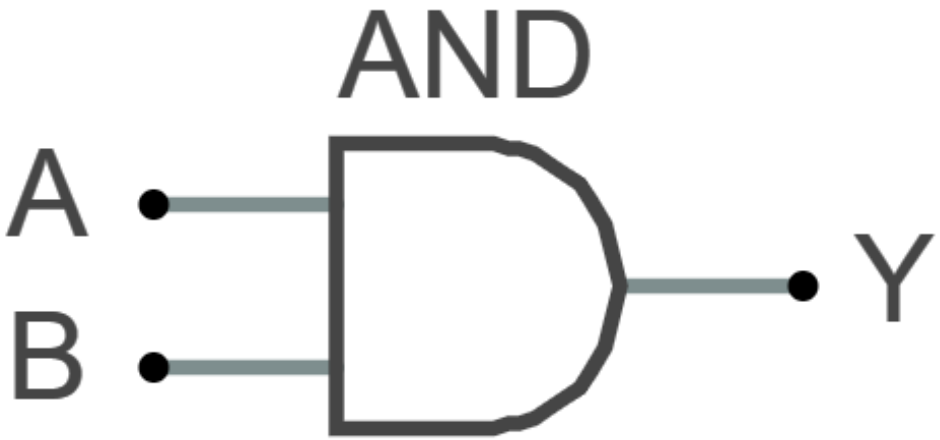
A	Y
0	1
1	0



Επισκόπηση στην άλγεβρα Boole (3/11)

Πύλη “ΚΑΙ”
Λογική εξίσωση: Y = A·B
Έξοδος 1 όταν όλες οι είσοδοι είναι 1

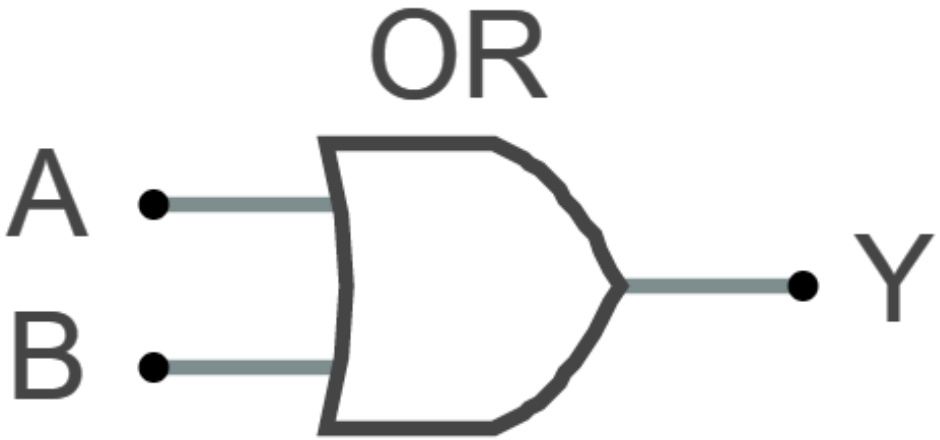
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



Επισκόπηση στην άλγεβρα Boole (4/11)

Πύλη “Ή”
Λογική εξίσωση: Y = A+B
Έξοδος 1 όταν τουλάχιστον μία είσοδος είναι 1

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



Επισκόπηση στην άλγεβρα Boole (5/11)

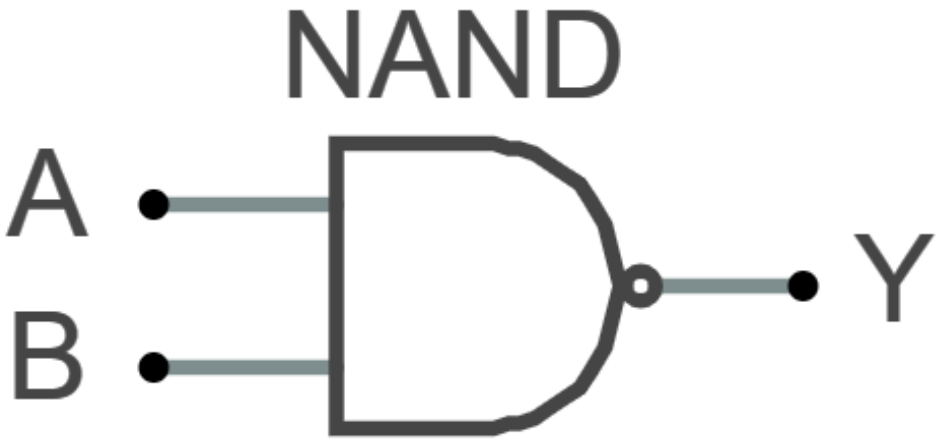
Πύλη “ΑΠΟΚΛΕΙΣΤΙΚΟ Ή”
Λογική εξίσωση: $Y = A \oplus B$
Έξοδος 1 όταν περιττό πλήθος εισόδων είναι 1

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Επισκόπηση στην άλγεβρα Boole (6/11)

Πύλη “ΟΧΙ ΚΑΙ”
Λογική εξίσωση: Y = ~(A·B)
Έξοδος 1 όταν τουλάχιστον μία είσοδος είναι 0

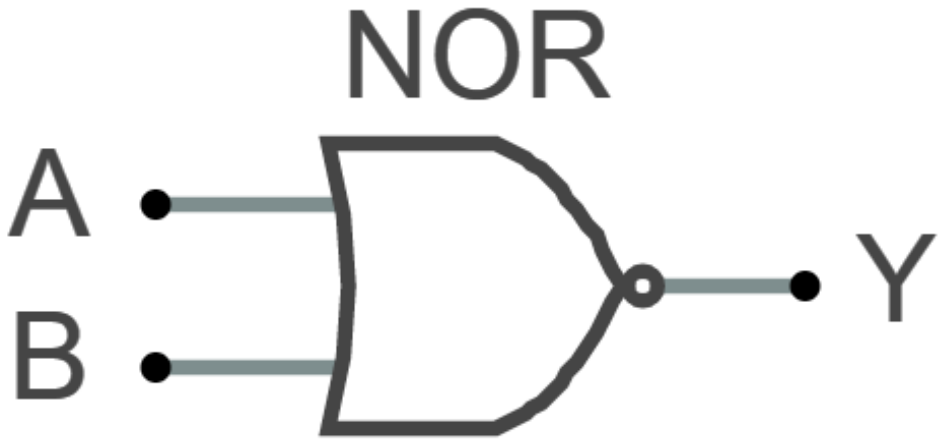
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



Επισκόπηση στην άλγεβρα Boole (7/11)

Πύλη “ΟΧΙ Ή”
Λογική εξίσωση: Y = ~(A+B)
Έξοδος 1 όταν όλες οι είσοδοι είναι 0

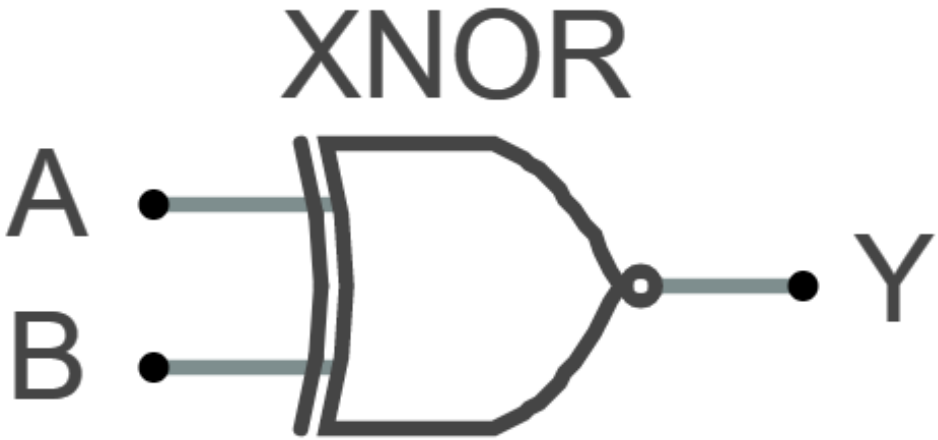
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



Επισκόπηση στην άλγεβρα Boole (8/11)

Πύλη “ΑΠΟΚΛΕΙΣΤΙΚΟ ΟΧΙ Ή”
Λογική εξίσωση: $Y = \sim(A \oplus B)$
Έξοδος 1 όταν άρτιο πλήθος εισόδων είναι 1

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1



Επισκόπηση στην άλγεβρα Boole (9/11)

Κάθε λογική πύλη υλοποιεί κάποια (σε αυτή τη περίπτωση βασική αλλά και συμμετρική) λογική συνάρτηση, και επομένως έναν πίνακα αληθείας, ο οποίος δείχνει τη σχέση των εξόδων σε σχέση με τις μεταθέσεις των εισόδων.

Είναι λογικό από αυτό να συμπεράνει κανείς ότι και οι διάφοροι συνδυασμοί αυτών των πυλών επίσης υλοποιούν λογικές συναρτήσεις, και από αυτές πίνακες αληθείας!

Επισκόπηση στην άλγεβρα Boole (10/11)

Επιπλέον, σημειώνεται ότι κάθε πίνακας αληθείας μπορεί να έχει πολλαπλές υλοποιήσεις της ίδιας συνάρτησης αλλά με διαφορετικό συνδυασμό των πυλών.

Εμείς στα πλαίσια του workshop θα ασχοληθούμε με τη λεγόμενη RTL (Register-Transfer Logic), δηλαδή το ***ποια*** είναι η λογική των σημάτων που έχουμε στη διάθεση και ***όχι*** *στη συγκεκριμένη υλοποίηση τους*. Άλλωστε, όπως σύντομα θα δούμε, αυτή είναι και η τακτική που ακολουθείται στο μεγαλύτερο μέρος παραγωγής VLSI συστημάτων.

Επισκόπηση στην άλγεβρα Boole (11/11)

Συνεπώς, προσπερνάμε για την ώρα την ανάλυση πάνω σε ελαχιστόρους και μεγιστόρους μιας συνάρτησης, όπως και τους τρόπους ελαχιστοποίησης με χάρτες Karnaugh.

Σημειώνεται όμως ότι σε έναν πίνακα αληθείας όταν βλέπουμε “X” σημαίνει αυτή η κατάσταση είναι αδιάφορη, δηλαδή μπορούμε να την κάνουμε “ό,τι μας βολεύει”, δηλαδή μπορούμε να το δούμε ως 1, 0 ή και τίποτα, αναλόγως τη περίπτωση.

Βασικό συντακτικό της Verilog (1/12)

Η Verilog είναι μια από πολλές γλώσσες περιγραφής υλικού (hardware description language, HDL), η οποία χρησιμοποιείται ευρέως από τη βιομηχανία για την ανάπτυξη όλων των ειδών ψηφιακών VLSI κυκλωμάτων.

Μικροεπεξεργαστές, κάρτες γραφικών, ρομπότ, επιταχυντές νευρωνικών δικτύων, μια HDL μπορεί (με τα κατάλληλα ορίσματα) να περιγράψει τα κυκλώματα που υλοποιούν όλες αυτές τις λειτουργίες!

Βασικό συντακτικό της Verilog (2/12)

Το βασικότερο συστατικό στη Verilog είναι το `module`. ΟΛΑ είναι modules. Περιγράφει τη λειτουργικότητα του σχεδιασμού, αναφέρει τις θύρες I/O, περιέχει δηλώσεις μεταβλητών, δηλώσεις και όλη τη λογική που θέλουμε να περιγράψουμε.

Ένα module, σαφώς, μπορεί να περιέχει και άλλα modules μέσα του, το οποίο δημιουργεί και μια ιεραρχία στο σχεδιασμό.

Βασικό συντακτικό της Verilog (3/12)

Ένα παραδειγματικό module στη Verilog:

```
module module_name(a, b, c);
    input a, b;
    output [3:0] c;    // 4-bit vector [(3-0)+1] output
    inout declarations;
    net declarations;
    parameter declarations;
    function declarations;
    task declarations;
    behavior/functionality description;    // e.g. c=b+a
endmodule
```

Με μπλε φαίνονται τα keywords και με πράσινο το σχόλιο.

Βασικό συντακτικό της Verilog (4/12)

Η Verilog έχει δύο (2) κύρια data types.

- **wire**: Αντιπροσωπεύουν συνδέσεις μεταξύ δομικών στοιχείων σαν απλά καλώδια, οπότε δε κρατάνε τη τιμή τους.

- **reg**: Αντιπροσωπεύουν την αποθήκευση δεδομένων, οπότε διατηρούν τη τιμή τους μέχρι να εκχωρηθεί ρητά. Μπορεί να χρησιμοποιηθούν για τη μοντελοποίηση latch, flip-flop κ.λπ., αλλά δεν αντιστοιχούν ακριβώς.

Βασικό συντακτικό της Verilog (5/12)

Η Verilog είναι γλώσσα τεσσάρων (4) τιμών.

- 1 / 0: Λογικές τιμές 1 και 0 αντίστοιχα.
- Z: Κατάσταση υψηλής εμπέδησης. Είναι η περίπτωση όπου τίποτα δεν καθορίζει την τιμή ενός καλωδίου.
- X: Μοντελοποιεί την περίπτωση όπου ο προσομοιωτής δεν μπορεί να αποφασίσει τη τιμή του δικτυώματος. Αποτελεί την αρχική κατάσταση των regs, και συμβαίνει επίσης όταν ένα καλώδιο οδηγείται στο 0 και στο 1 ταυτόχρονα ή η έξοδος έχει Z εισόδους.

Βασικό συντακτικό της Verilog (6/12)

Η Verilog περιέχει και διάφορες άλλες λειτουργίες, όπως parameters (σταθερές δηλωμένες στην αρχή), integer και real αριθμούς, strings και άλλες λειτουργίες.

Ενδεικτικά:

```
parameter a = 3;
parameter var = 5.5;
integer num = 32'b1;    // 32-bit integer written in binary
real var_decimal = 10.24;
reg [8*12:1] str_var = "Hello World!";    // ASCII string
```

Βασικό συντακτικό της Verilog (7/12)

Τελεστές (operators) της Verilog:

Type	Symbol	Description	Note
Arithmetic	+	add	
	-	subtract	
	*	multiply	
	/	divide	May not synthesize
	%	modulus (remainder)	May not synthesize
	**	power	May not synthesize

Βασικό συντακτικό της Verilog (8/12)

Τελεστές (operators) της Verilog:

Type	Symbol	Description	Note
Concatenation	{…, …}	concatenates bits	explicit bit lengths
Bitwise	~	not	
	|	or	
	&	and	
	^	xor	
	~& or &~	nand	mix two operators

Βασικό συντακτικό της Verilog (9/12)

Τελεστές (operators) της Verilog:

Type	Symbol	Description	Note
Relational	>	greater than	
	<	less than	
	>=	greater or equal than	
	<=	less or equal than	
	==	equal	
	!=	not equal	

Βασικό συντακτικό της Verilog (10/12)

Τελεστές (operators) της Verilog:

Type	Symbol	Description	Note
Logical	!	negation	
	||	logical OR	
	&&	logical AND	
	==	logical equality	

Βασικό συντακτικό της Verilog (11/12)

Τελεστές (operators) της Verilog:

Type	Symbol	Description	Note
Shift Operators	>>	right shift	
	<<	left shift	
	>>>	right shift with MSB shifted right (sign)	
	<<<	same as <<	

Βασικό συντακτικό της Verilog (12/12)

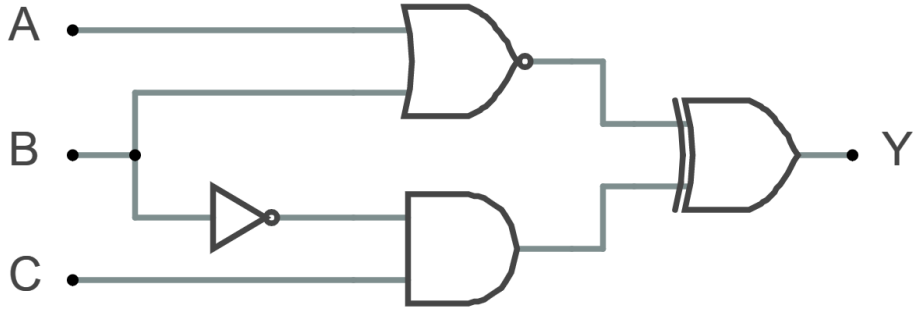
Στη περίπτωση χρήσης παραπάνω από ενός τελεστή στη Verilog, η σειρά εκτέλεσής τους είναι η εξής:

1. Bitwise / Logical πράξεις (~, |, &, ^, ~&, !, ||, &&)
2. Concatenation ({..., ...})
3. Πολλαπλασιασμός, διαίρεση, modulo (*, /, %)
4. Αριθμητική πρόσθεση / αφαίρεση (+, -)
5. Ολισθήσεις (<<, >>, <<<, >>>)
6. Σχεσιακές διαφορών (<, >, <=, >=)
7. Σχεσιακές ισότητας (==, !=)

Συνδυαστικά κυκλώματα στη Verilog (1/14)

Έστω το παρακάτω συνδυαστικό κύκλωμα. Θα το περιγράψουμε με δύο (2) διαφορετικούς τρόπους:

- Στο επίπεδο των πυλών
- Στο επίπεδο μεταφοράς καταχωρητών (RTL)



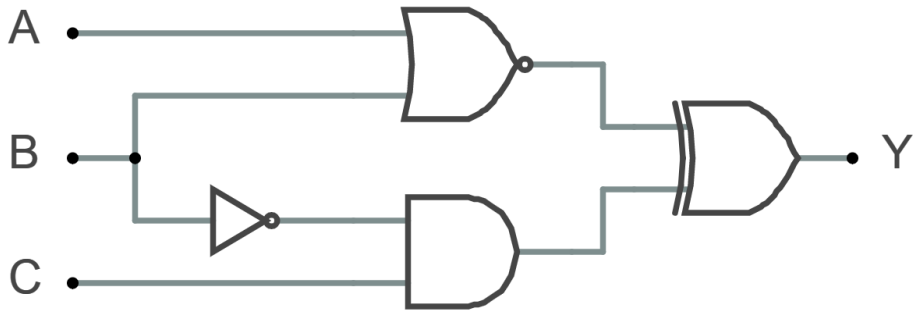
Συνδυαστικά κυκλώματα στη Verilog (2/14)

```
module example_combinational_gate_level (a, b, c, y);
   input a, b, c;   // 1-bit (wire implied) inputs a, b, c
   output y;        // 1-bit (wire implied) output y

   wire g1, g2, g3; // Temporary gate outputs are written as wires

   // The outputs of each gate
   // are always written first
   not (g1, b);
   nor (g2, a, b);
   and (g3, g1, c);
   xor (y, g2, g3);

endmodule
```

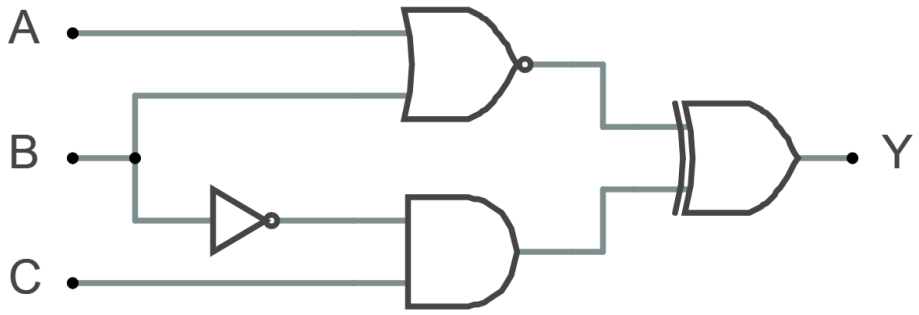


Συνδυαστικά κυκλώματα στη Verilog (3/14)

```
module example_combinational_rtl (a, b, c, y);
    input a, b, c;    // 1-bit (wire implied) inputs a, b, c
    output y;         // 1-bit (wire implied) output y

    // A single "assign" will do the trick!
    assign y = (~(a | b)) ^ ((~b) & c);

endmodule
```



Συνδυαστικά κυκλώματα στη Verilog (4/14)

- Τι άλλα κυκλώματα μπορούμε να περιγράψουμε με assign;
  - *ΟΛΗ τη συνδυαστική λογική ενός κυκλώματος.*

- Και τι περιέχει αυτή η συνδυαστική λογική;
  - Αριθμητικά κυκλώματα.
  - Πολυπλέκτες (multiplexers).
  - Αποκωδικοποιητές (decoders).
  - Ολισθητές (shifters).
  - Ό,τι άλλο κύκλωμα μπορεί να γίνει “με τη μία” η πράξη.

Συνδυαστικά κυκλώματα στη Verilog (5/14)

Ημιαθροιστής

Λειτουργία: Πρόσθεση δύο εισόδων A και B

Είσοδοι: A, B

Έξοδοι: Sum, Carry

```
module half_adder (a, b, carry, sum);
   input a, b;
   output carry, sum;

   assign carry = a & b;
   assign sum = a ^ b;

endmodule
```

Συνδυαστικά κυκλώματα στη Verilog (6/14)

Πλήρης Αθροιστής

Λειτουργία: Πρόσθεση δύο εισόδων A και B, μαζί με τυχόν κρατούμενο

Είσοδοι: A, B, C_{in}

Έξοδοι: Sum, Carry

```
module full_adder (a, b, c_in, carry,
sum);
   input a, b, c_in;
   output carry, sum;

   assign carry = (a & b) | (a & c_in) |
(b & c_in);
   assign sum = a ^ b ^ c_in;

endmodule
```

Συνδυαστικά κυκλώματα στη Verilog (7/14)

Αλλά μπορούμε να τα κάνουμε όλα αυτά και πιο γρήγορα!

```
module half_adder_one_assign (a, b,
carry, sum);
   input a, b;
   output carry, sum;

   // Concatenate to use one assign!
   assign {carry, sum} = a + b;

endmodule
```

```
module full_adder_one_assign (a, b, c_in,
carry, sum);
   input a, b, c_in;
   output carry, sum;

   // Concatenate to use one assign!
   assign {carry, sum} = a + b + c_in;

endmodule
```

Συνδυαστικά κυκλώματα στη Verilog (8/14)

Μάλιστα, την ιδέα πίσω από το πλήρη αθροιστή (full adder) μπορούμε να την επεκτείνουμε στο παρακάτω κύκλωμα:

```
module scalable_size_adder (a, b, c_in, carry, sum);
   parameter N = 8;   // To make a, b and sum the size we want easily (here 8-bits)

   input [N-1:0] a, b;
   input c_in;
   output [N-1:0] sum;
   output carry;

   // Concatenate to use one assign!
   assign {carry, sum} = a + b + c_in;

endmodule
```

Συνδυαστικά κυκλώματα στη Verilog (9/14)

Πολυπλέκτης 2-σε-1

Λειτουργία: Επιλογή μεταξύ δύο εισόδων A και B ποιο θα βγει στην έξοδο
Είσοδοι: A, B, Select
Έξοδοι: Y

Με άλλα λόγια, έχουμε τη συνάρτηση: Αν S = 1, Y = B, αλλιώς αν S = 0, Y = A

```
module mux_2_to_1 (a, b, sel, y);
   parameter N = 8;

   input [N-1:0] a, b;
   input sel;
   output [N-1:0] y;

   // Instead of standard RTL, use
   // a tenary operator!
   assign y = (sel) ? b : a;

endmodule
```

Συνδυαστικά κυκλώματα στη Verilog (10/14)

Πολυπλέκτης 4-σε-1

Λειτουργία: Επιλογή μεταξύ των εισόδων A, B, C, D ποιο θα βγει στην έξοδο

Είσοδοι: A, B, C, D, Select

Έξοδοι: Y

```
module mux_4_to_1 (a, b, c, d, sel, y);
   parameter N = 8;

   input [N-1:0] a, b, c, d;
   input [1:0] sel;
   output [N-1:0] y;

   // Last option is the default option
   // No need to write default explicitly
   assign y = (sel == 2'b00) ? a :
              (sel == 2'b01) ? b :
              (sel == 2'b10) ? c : d;

endmodule
```

Συνδυαστικά κυκλώματα στη Verilog (11/14)

Ένα λίγο πιο σύνθετο παράδειγμα που θα συνδυαστούν όλα όσα έχουμε δει, όπως και καινούργιες λειτουργίες

Έστω συνδυαστικό κύκλωμα με εισόδους a, b, c μήκους 3-bit η καθεμία. Βάσει ενός σήματος sel, θα επιλέγει η έξοδος y να είναι η έξοδος του `example_combinational_rtl.v`, *αλλιώς η έξοδος sum ενός 3-bit αθροιστή με τις ίδιες εισόδους*. Στη περίπτωση του c_in στον αθροιστή, η είσοδος να είναι 1 αν τουλάχιστον τα μισά bit του c είναι 1. Να δημιουργηθεί με instantiations όπου αυτό δυνατό.

Συνδυαστικά κυκλώματα στη Verilog (12/14)

```
module instantiation_circuit (a, b, c, sel, y);
   parameter N = 3;

   input [2:0] a, b, c;
   input sel;
   output [2:0] y;

   wire [2:0] rtl_out, add_out;
   wire c_in;

   // We could use a generate for loop here of course, but let's
   // not complicate things for now. We're instantiating using
   // an ordered list (my personal favorite). Why? Less is more!
   example_combinational_rtl rtl_0 (a[0], b[0], c[0], rtl_out[0]);
   example_combinational_rtl rtl_1 (a[1], b[1], c[1], rtl_out[1]);
   example_combinational_rtl rtl_2 (a[2], b[2], c[2], rtl_out[2]);
```

Ο υπόλοιπος κώδικας φαίνεται και στην επόμενη διαφάνεια

Συνδυαστικά κυκλώματα στη Verilog (13/14)

```
    // Making a majority function that we can call later. Yes, functions are
    // synthesizable (more on that later), and they can be quite helpful,
    // though this use below is trivial
    function MAJ;
       input I1, I2, I3;
          begin
             MAJ = (I1 & I2) | (I1 & I3) | (I2 & I3);
          end
    endfunction

    // Calling the MAJ function we defined previously
    assign c_in = MAJ(c[2], c[1], c[0]);

    // Instantiating our 3-bit adder, and using our new c_in
    // that we made with our function call earlier
    scalable_size_adder #(N) adder (a, b, c_in, add_out);

    // Our final output multiplexed from the two previous sub-circuits that we
    // instantiated above. Keep in mind that ALL OF THIS is parallel, i.e. it all
    // runs at the exact same time!
    assign y = (sel) ? add_out : rtl_out;

endmodule
```

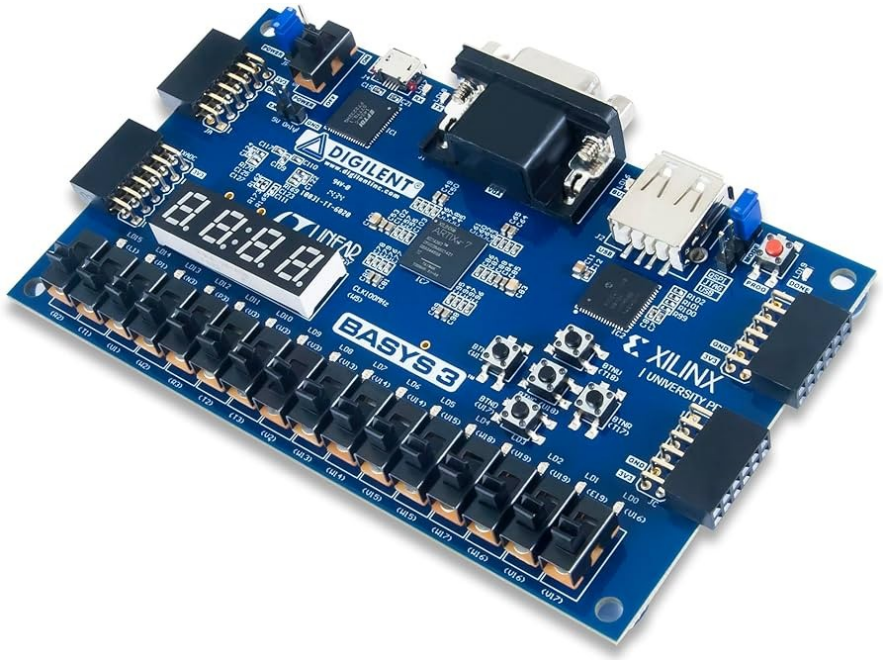
Συνδυαστικά κυκλώματα στη Verilog (14/14)

Στόχος μας στο workshop δεν είναι μόνο να γράψουμε κώδικα Verilog. Άλλωστε, αυτό είναι θέμα συντακτικού, και ένας έμπειρος προγραμματιστής μπορεί να μάθει να γράφει εντός λίγων ημερών σε πολύ καλό επίπεδο.

Όχι, το θέμα είναι να γράφουμε *συνθέσιμο* κώδικα Verilog! Δηλαδή, κώδικας ο οποίος *παράγει κύκλωμα*! Αυτό θα πρέπει να μπορούμε μετά να το στείλουμε σε εταιρεία προς κατασκευή ή να το φορτώσουμε σε ένα FPGA.

Τεχνολογία των FPGA (1/7)

Τα Field-Programmable Gate Arrays είναι ένα είδος προγραμματίσιμης λογικής. Περιέχουν μια σειρά προγραμματιζόμενων λογικών κελιών & διασυνδέσεων που με κατάλληλη χρήση μπορούν να υλοποιήσουν διάφορες λογικές συναρτήσεις.

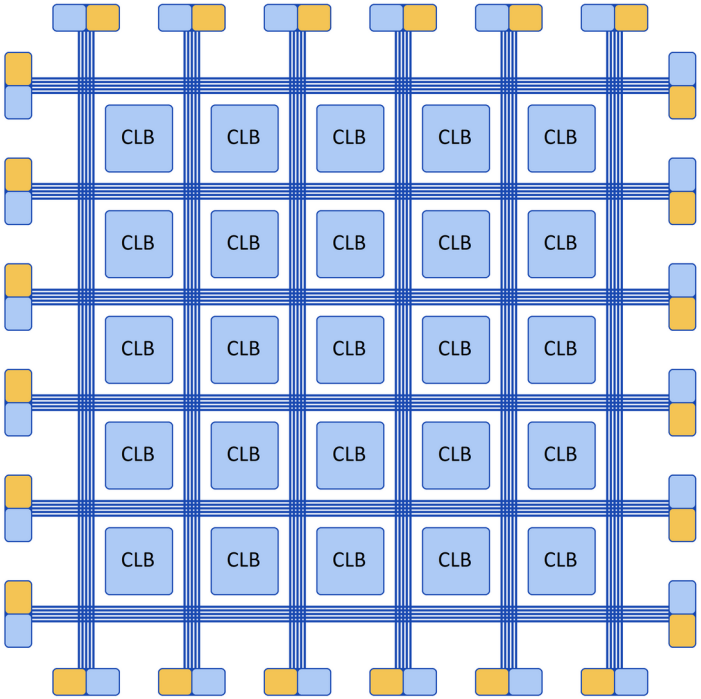


Basys 3 FPGA board της Digilent, με ένα Artix-7 FPGA της Xilinx

Τεχνολογία των FPGA (2/7)

Η γενική αρχιτεκτονική ενός FPGA αποτελείται από τα λεγόμενα Configurable Logic Blocks (CLBs), τα Programmable Interconnects και τα I/O Blocks.

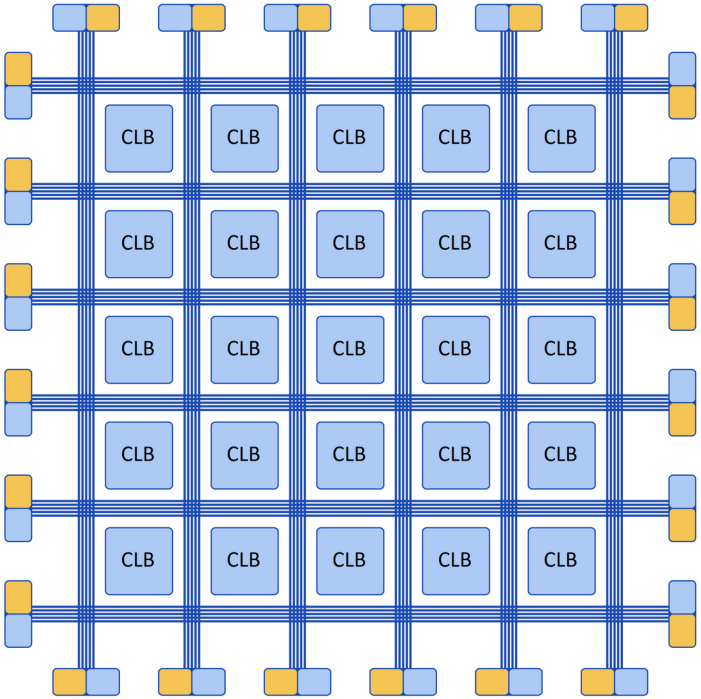
Η ονομασία αλλάζει από κατασκευαστή σε κατασκευαστή.



Γενική αρχιτεκτονική ενός FPGA

Τεχνολογία των FPGA (3/7)

Κάθε CLB παίρνει εισόδους από άλλα CLB ή από τα I/O Blocks, και η έξοδος μπορεί να οδηγηθεί σε καταχωρητή ή tri-state στοιχεία. Σήμερα, τα πιο εξελιγμένα FPGA της αγοράς αποτελούνται από εκατομμύρια τέτοια μεγακύτταρα.



Γενική αρχιτεκτονική ενός FPGA

Τεχνολογία των FPGA (4/7)

Η ακριβής εσωτερική διάρθρωση ενός CLB διαφέρει από οικογένεια σε οικογένεια και από κατασκευαστή σε κατασκευαστή. Αλλά, γενικώς, διαθέτουν όλα λίγο-πολύ τα εξής χαρακτηριστικά:

- Κάποια Look-Up Tables (LUT) συνήθως ως και 8 εισόδων.
- Γρήγορη άθροιση στα ίδια bit (συνήθως ripple carry).
- D Flip-flops για τη δημιουργία ακολουθιακών κυκλωμάτων.
- Πολλά multiplexers για τη κατάλληλη διασύνδεση όλων αυτών με βάση τις απαιτήσεις του χρήστη!

Τεχνολογία των FPGA (5/7)

Ένα Look-Up Table δεν είναι τίποτα άλλο παρά ένας (προγραμματίσιμος) πίνακας αληθείας! Ως εκ τούτου, με κατάλληλο πλήθος εισόδων, μπορεί να περιγράψει οποιαδήποτε λογική συνάρτηση θέλουμε.

Στη πραγματικότητα όμως αποτελεί κύτταρο μνήμης!

Και τι σημαίνει αυτό δηλαδή;

Τεχνολογία των FPGA (6/7)

Ένα Look-Up Table “χωράει” διάφορες συναρτήσεις με όλες ή μερικές από τις γραμμές των αντίστοιχων πινάκων αληθείας που θέλουμε να υλοποιήσουμε, και ασχέτως το πλήθος των μεταβλητών που χρησιμοποιούμε, θα έχει τον ίδια χρόνο υπολογισμού ακριβώς επειδή υλοποιείται με μνήμη.

Με λιγότερα (αλλά πιο τεχνικά λόγια), το κρίσιμο μονοπάτι ΔΕΝ ΑΛΛΑΖΕΙ!

Τεχνολογία των FPGA (7/7)

Ένα ενδιαφέρον αποτέλεσμα αυτού είναι το εξής: Σχεδόν ποτέ δε βγάζει νόημα σε ένα FPGA να υλοποιήσουμε αθροιστές διαφορετικούς της διάδοσης κρατουμένου (ripple carry). Γιατί όμως;

Αφού έχουμε ίδια καθυστέρηση ασχέτως συνάρτησης, άλλες αρχιτεκτονικές αθροιστή (που θα έπρεπε να είναι πιο γρήγοροι!!!) θα απαιτήσουν παραπάνω LUTs από αυτό της διάδοσης κρατουμένου. Αυτό ισχύει προφανώς αν και οι δύο αθροιστές αυτοί υλοποιηθούν σε LUTs του FPGA.

Βιβλιογραφία

- Digital Design: Principles and Practices, John F. Wakerly, 5th Edition
- CMOS VLSI Design: A Circuits and Systems Perspective, Neil H. E. Weste, David M. Harris, 4th Edition
- FPGA Prototyping by Verilog Examples: Xilinx Spartan-3 Version, Pong P. Chu
- Σχεδίαση Συστημάτων με Χρήση Υπολογιστών (E-CAD), Πανεπιστήμιο Πατρών, Χαρίδημος Βέργος
- Introduction to Verilog HDL, Synopsys Courseware, Jorge Ramírez