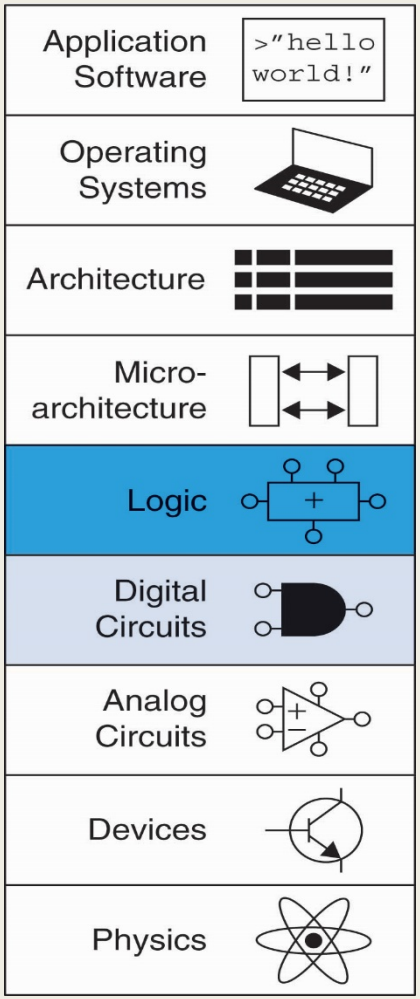
Κεφάλαιο 4

Γλώσσες περιγραφής υλικού

Γιώργος Παπαδημητρίου, Αντώνης Πασχάλης,
Βασιλόπουλος Διονύσης

Περιεχόμενα κεφαλαίου 4

- Τεχνολογία FPGA
- Φάσεις σχεδίασης υλικού
- Τα θεμέλια της γλώσσας VHDL
  - *Σήματα και μεταβλητές τύπου std_logic*
  - *Περιγραφή δομής*
  - *Περιγραφή dataflow*
  - *Περιγραφή συμπεριφοράς*
  - *Η εντολή IF*
    - Συνδυαστική λογική
    - Ακολουθιακή λογική
      - *Ασύγχρονη*
      - *Σύγχρονη*



Προγραμματιζόμενες από τον χρήστη διατάξεις πυλών (FPGA)

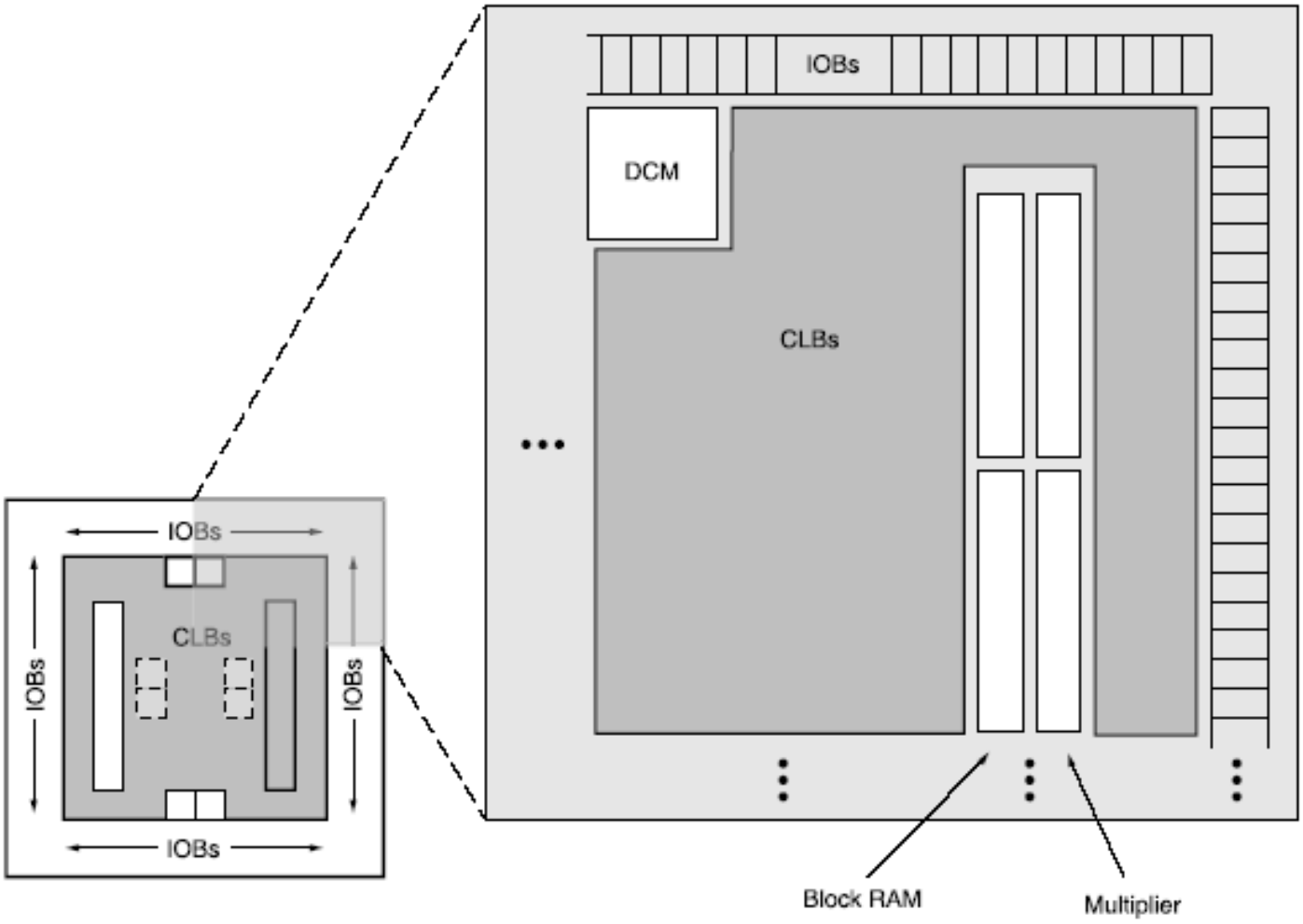
- Παρέχουν δυνατότητα σχεδίασης **VLSI κυκλωμάτων χαμηλού κόστους** στο πεδίο, με τη χρήση σχετικά φθηνών εργαλείων λογισμικού (CAD)
  - *κόστος ανεκτό από μικρές εταιρείες που δραστηριοποιούνται στην ανάπτυξη επιταχυντών υλικού και γενικότερα πυρήνων IP*
- Είναι **επαναπρογραμματιζόμενες** και **επαναδιατάξιμες** (ολικώς ή μερικώς) ακόμα και κατά τη διάρκεια της κανονικής λειτουργίας
  - *Παρέχουν **μεγάλη ευελιξία** στη σχεδίαση ψηφιακών συστημάτων*
- Διαθέτουν **ενσωματωμένες μνήμες, πολλαπλασιαστές, ειδικές μονάδες για ψηφιακή επεξεργασία σήματος, εισόδους/εξόδους υψηλών ταχυτήτων, μετατροπείς δεδομένων (όπως ADC, DAC, RF),** ακόμα και **πυρήνες επεξεργαστών** σε κάποιες περιπτώσεις
- Η γενικότερη τεχνολογική εξέλιξη σε θέματα κόστους, αποδόσεων, κατανάλωσης ισχύος και αξιοπιστίας έχει σαν αποτέλεσμα οι σύγχρονες διατάξεις FPGA να χρησιμοποιούνται ευρέως σε **εμπορικές, βιομηχανικές, αμυντικές και διαστημικές εφαρμογές**

Η αρχιτεκτονική των FPGAs της XILINX

- Παράδειγμα: Η αρχιτεκτονική του Spartan III
  - *Configurable Logic Blocks (CLBs)*
    που περιέχουν πίνακες αναζήτησης (LUT), που υλοποιούν συνδυαστική λογική, και στοιχεία αποθήκευσης που μπορούν να χρησιμοποιηθούν ως D Flip-flops ή Latches
  - *Input/Output Blocks (IOBs)*
    που ελέγχουν την ροή δεδομένων μεταξύ των I/O pins και της εσωτερικής λογικής
  - *Block RAMs*
    που παρέχουν αποθηκευτικό χώρο μνήμης, μεγέθους για παράδειγμα 18Kbit (το μέγεθος εξαρτάται από την τεχνολογία)
  - *Multiplier Blocks*
    σε πολλές οικογένειες οι πολλαπλασιαστές έχουν εξελιχθεί σε ειδικές μονάδες ψηφιακής επεξεργασίας σήματος
  - *Digital Clock Manager (DCM) Blocks*
    που παρέχουν αυτορρυθμιζόμενες πλήρως ψηφιακές λύσεις για κατανομή, καθυστέρηση, διαίρεση και ρύθμιση της φάσης των ρολογιών
  - *Τα blocks διασυνδέονται μέσω προγραμματιζόμενων πινάκων διακοπτών (switch matrices)*

Η αρχιτεκτονική των FPGAs της XILINX

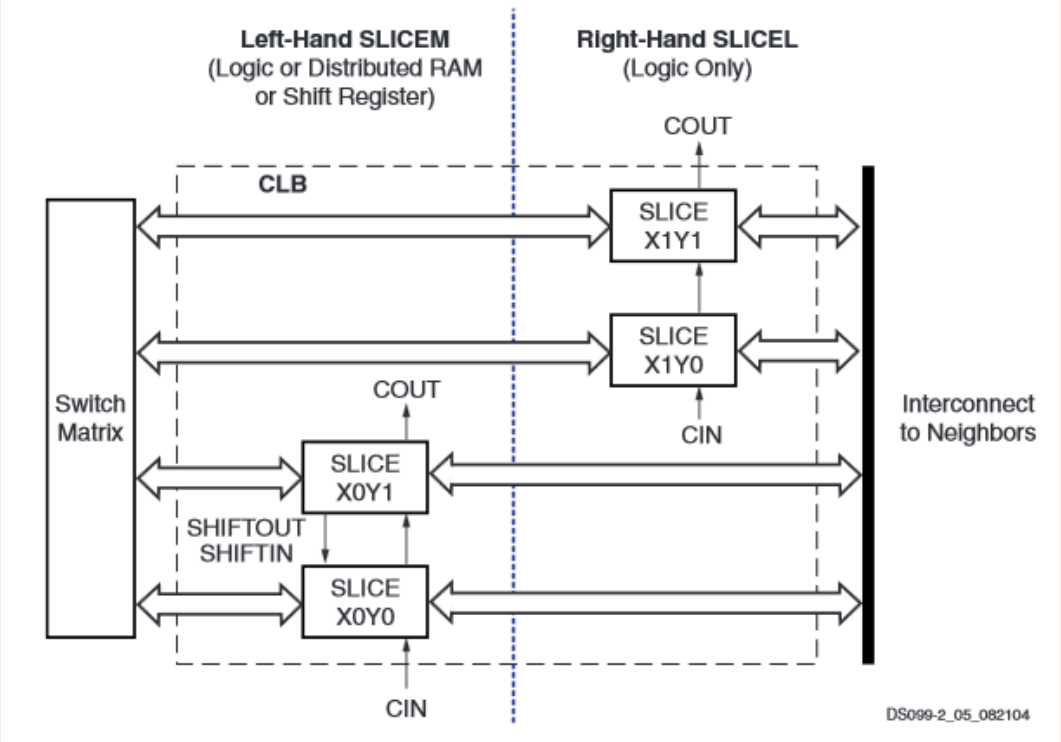
- Παράδειγμα: Η αρχιτεκτονική του Spartan III



Spartan - 3 Architecture

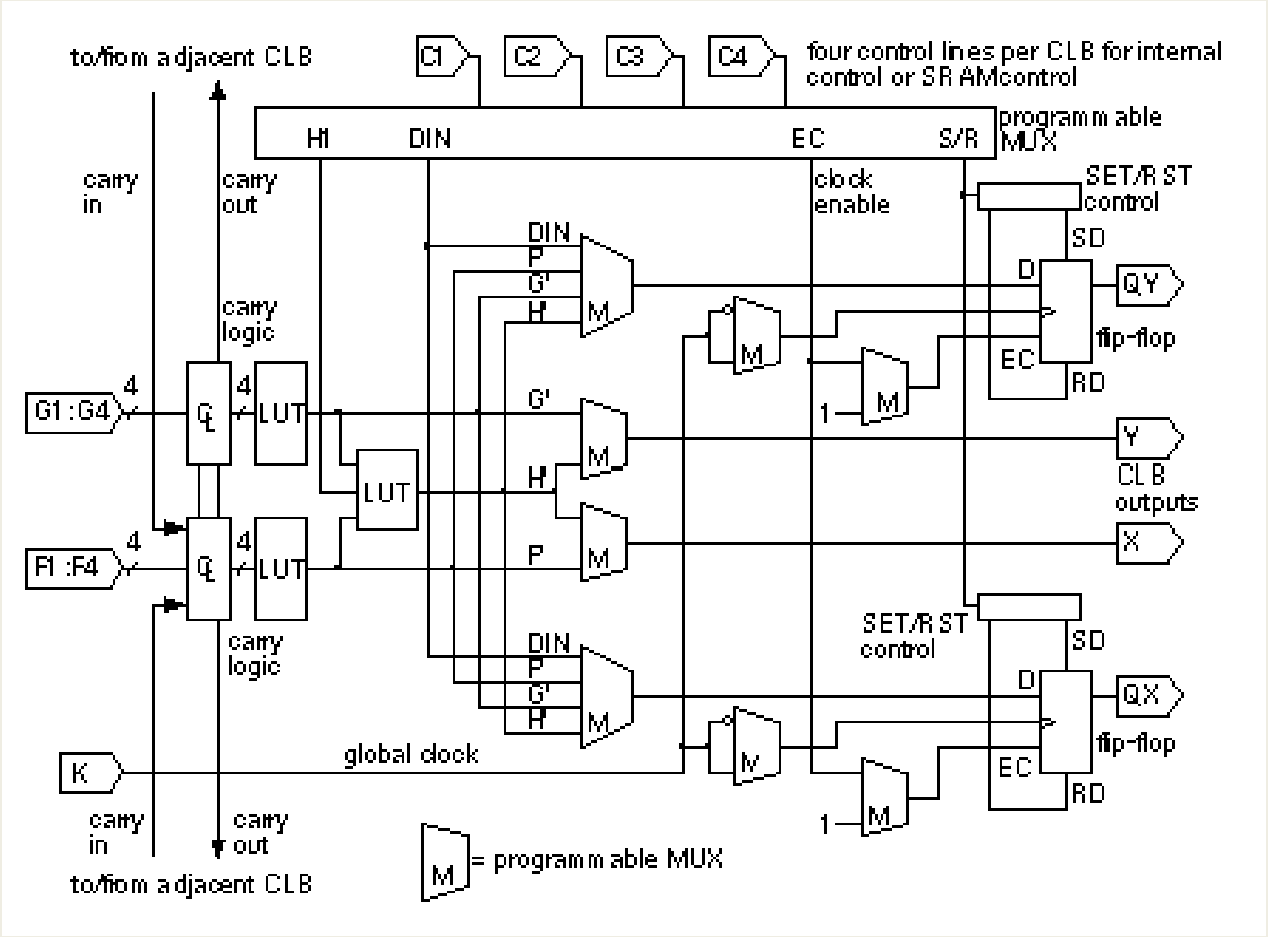
Η αρχιτεκτονική των FPGAs της XILINX

- Παράδειγμα: Η αρχιτεκτονική του Spartan III
  - *Κάθε CLB αποτελείται από τέσσερα συνδεδεμένα **slices** τα οποία ομαδοποιούνται σε ζευγάρια και κάθε ζευγάρι σχηματίζει μια ανεξάρτητη αλυσίδα διάδοσης κρατουμένου.*
    - Το αριστερό ζευγάρι υλοποιεί λογική, κατανεμημένη μνήμη ή καταχωρητή ολίσθησης
    - Το δεξιό ζευγάρι υλοποιεί αποκλειστικά λογική



Η αρχιτεκτονική των FPGAs της XILINX

- Παράδειγμα: Η αρχιτεκτονική του Spartan III
  - *Κάθε slice του CLB έχει : 2 πίνακες αναζήτησης (LUT) των 4 εισόδων, 2 στοιχεία αποθήκευσης, πολυπλέκτες, αλυσίδα διάδοσης κρατούμενου και πύλες για αριθμητική λογική*

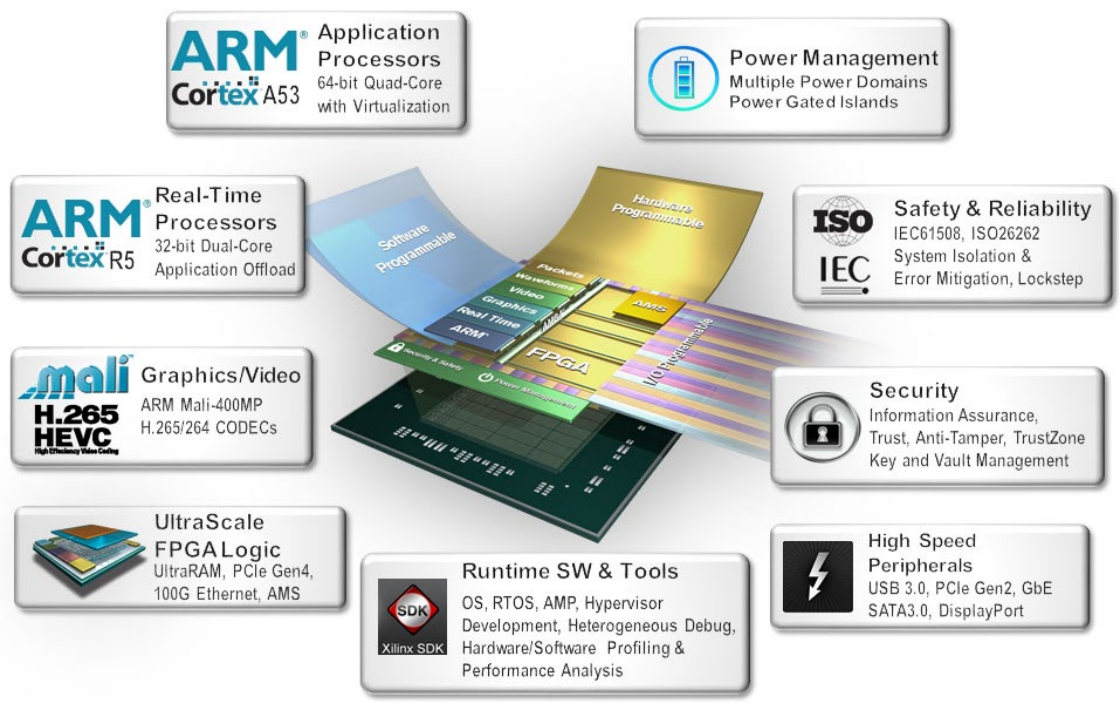


Επιταχυντές Υλικού για ψηφιακά συστήματα υψηλής απόδοσης σε τεχνολογία FPGA

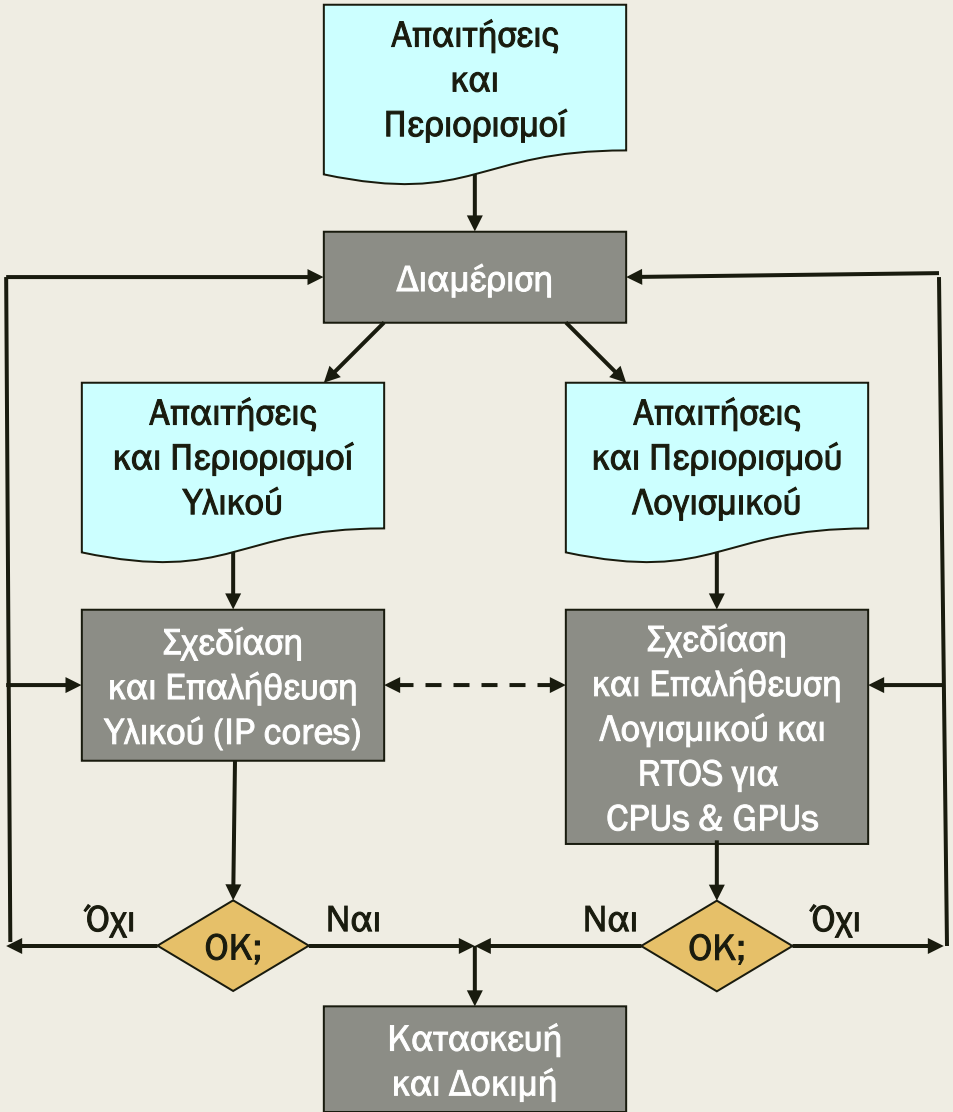
- Κανόνας 90/10
  - *Συχνά, το **90% του χρόνου εκτέλεσης** και της κατανάλωσης ισχύος ενός προγράμματος δαπανάται **από το 10% του κώδικα***
  - *Μικρά κομμάτια μιας εφαρμογής αποτελούν το **bottleneck της απόδοσης***
    - Αφορούν κυρίως επεξεργασία δεδομένων χωρίς πολύπλοκο έλεγχο (dataflow processing), όπως επεξεργασία και συμπίεση εικόνας 3D και βίντεο, κρυπτογραφία, μηχανική μάθηση, κλπ.
  - *Οι **επεξεργαστές γραφικών (GPUs)** επιταχύνουν σημαντικά το κρίσιμο μέρος της εφαρμογής που υλοποιεί αλγορίθμους με διανυσματικές πράξεις που εκτελούνται παράλληλα χωρίς εξαρτήσεις δεδομένων*
  - *Οι **επιταχυντές υλικού** (ως **πυρήνες IP σε προγραμματιζόμενη λογική**) επιταχύνουν σημαντικά το κρίσιμο μέρος της εφαρμογής που υλοποιεί σύνθετους αλγορίθμους με εξαρτήσεις δεδομένων*
  - *Οι **πυρήνες επεξεργαστών (CPUs)** υλοποιούν τα λιγότερο κρίσιμα μέρη με τεχνικές παράλληλης επεξεργασίας*

Επιταχυντές Υλικού για ψηφιακά συστήματα υψηλής απόδοσης σε τεχνολογία FPGA

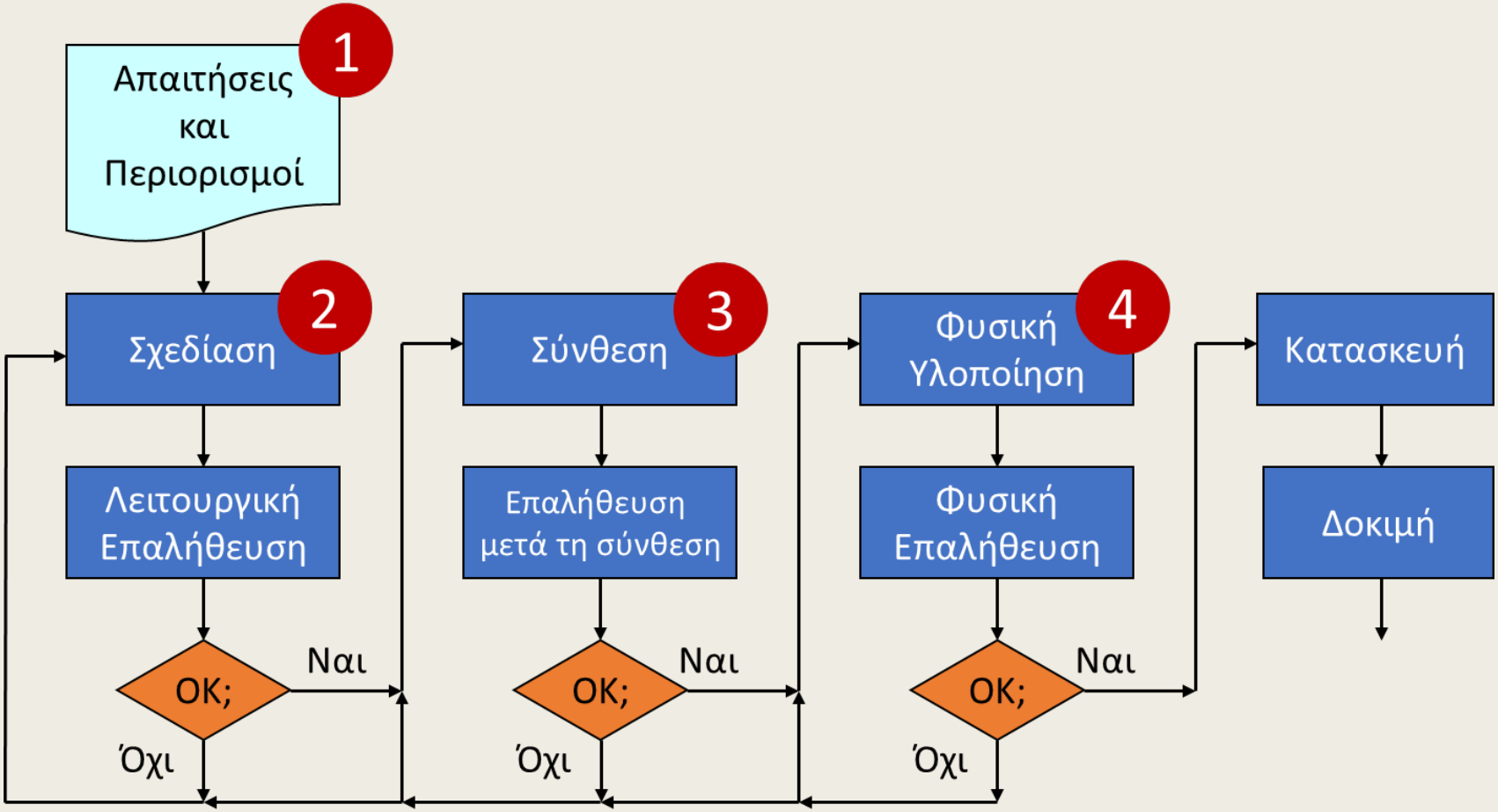
- Η τεχνολογική εξέλιξη που αλλάζει την κλασσική προσέγγιση του **υλικού** και τα όρια της **συ-σχεδίασης υλικού-λογισμικού**
  - *προγραμματιζόμενη λογική (για πυρήνες IP) + μνήμες + επεξεργαστές ARM + επεξεργαστές γραφικών + έτοιμοι επιταχυντές υλικού + συνδεσιμότητα υψηλής ταχύτητας σε ένα τσιπ (Multi-Processing System on Chip)!*
  - *Το ενσωματωμένο λογισμικό μπορεί να εκτελεστεί από την SRAM στο FPGA*
  - *Λύση ενός τσιπ μειωμένου κόστους που αποφεύγει το υψηλό NRE του ASIC*



Συ-σχεδίαση υλικού-λογισμικού στα σύγχρονα ψηφιακά συστήματα



Φάσεις σχεδίασης υλικού



Φάσεις σχεδίασης υλικού

- Σχεδίαση επιταχυντών υλικού σε τεχνολογία FPGA με χρήση εργαλείων CAD (computer-aided design)
  - *Προσδιορισμός απαιτήσεων και περιορισμοί*
  - *Εισαγωγή σχεδίασης (design entry)*
    - Προγραμματισμός σε γλώσσα περιγραφής υλικού (HDL), αντί για σχηματικά διαγράμματα
    - Λειτουργική επαλήθευση της σχεδίασης με προσομοίωση (simulation) και τυπικές μεθόδους (formal methods)
  - *Σύνθεση (synthesis)*
    - Αυτόματη παραγωγή ιεραρχικού σχηματικού διαγράμματος και gate-level netlist
    - Επαλήθευση μετά τη σύνθεση με προσομοίωση (λειτουργική και χρονική)
  - *Φυσική υλοποίηση (implementation)*
    - Υλοποίηση στην τεχνολογία FPGA (διαδικασίες map, place και route)
    - Φυσική επαλήθευση (λειτουργική και χρονική)
    - Ικανοποίηση απαιτήσεων και περιορισμών
  - *Προγραμματισμός FPGA και δοκιμή (κατασκευή)*

Προσδιορισμός απαιτήσεων και περιορισμοί

- **Ανάλυση των απαιτήσεων και καθορισμός των προδιαγραφών**
  - *λειτουργία, εσωτερικές και εξωτερικές διασυνδέσεις*
  - *περιβαλλοντολογικές απαιτήσεις (θερμοκρασία, ακτινοβολία, κλπ.)*
  - *απαιτήσεις επίδοσης και λειτουργίας σε πραγματικό χρόνο*
  - *απαιτήσεις κατανάλωσης ισχύος και ενέργειας*
  - *απαιτήσεις ποιότητας προϊόντος (φερεγγυότητα, ασφάλεια, QA)*
- **Προκαταρτική σχεδίαση σε υψηλό επίπεδο**
  - *περιγραφή σε υψηλό επίπεδο διαγραμμάτων με μπλοκ*
  - *προσδιορισμός λειτουργικών μονάδων και ιεραρχίας*
  - *προσδιορισμός διασυνδέσεων (είσοδοι, έξοδοι)*
  - *αρχιτεκτονική περιγραφή συνήθως σε γλώσσα υψηλού επιπέδου (π.χ. C-like, SystemC) και δημιουργία ενός golden model*

Εισαγωγή σχεδίασης

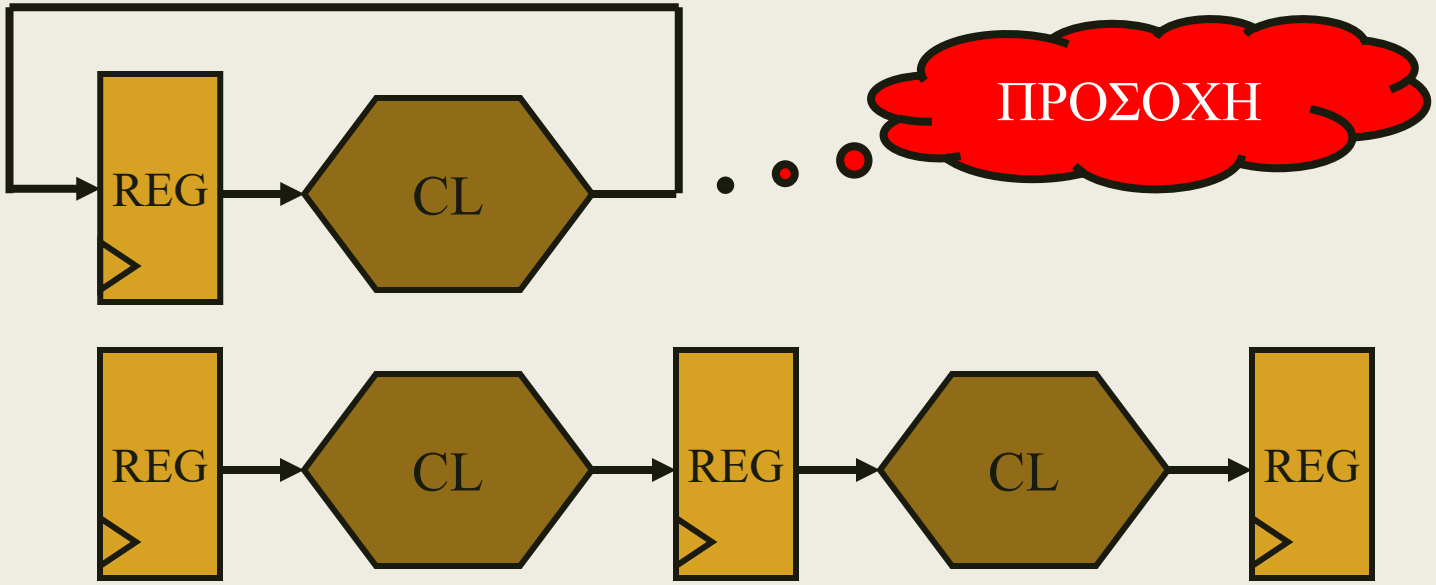
- Προγραμματισμός σε **γλώσσα περιγραφής υλικού (Hardware Description Language – HDL)**, αντί για σχηματικά διαγράμματα
  - *Υπερτερεί σε σύγκριση με τα σχηματικά διαγράμματα*
    - Η περιγραφή σε HDL γίνεται πιο κατανοητή από ένα σχηματικό διάγραμμα, λόγω καλλίτερης διαχείρισης της πολυπλοκότητας
    - Η μοντελοποίηση του συστήματος μπορεί να γίνει σε όλα τα επίπεδα (από τα υψηλότερα ως τα χαμηλότερα) με κατάλληλη αφαίρεση από επίπεδο σε επίπεδο
    - Η περιγραφή σε HDL είναι ανεξάρτητη (?) από τις βιβλιοθήκες σχεδίασης (design libraries) και τα εργαλεία CAD
  - *Υπερτερεί σε σύγκριση με τις γλώσσες προγραμματισμού (SW)*
    - Παρέχει δομές που είναι συνθέσιμες και περιγράφουν καλύτερα το υλικό, όπως τις μνήμες
    - Υποστηρίζει την παράλληλη εκτέλεση εντολών (δεν περιορίζεται στην ακολουθιακή εκτέλεση εντολών)
    - Παρέχει τη δυνατότητα για περιγραφή χρονισμών
  - *Αλλά θέλει προσοχή!*

Εισαγωγή σχεδίασης

- Προγραμματισμός σε **γλώσσα περιγραφής υλικού (Hardware Description Language – HDL)**, αντί για σχηματικά διαγράμματα
  - *Αλλά θέλει προσοχή!*
    - Οι γλώσσες HDL μαθαίνονται εύκολα, αλλά εφαρμόζονται σωστά δύσκολα
    - Οι προγραμματιστές τείνουν να γράφουν κώδικα HDL που μοιάζει με τα προγράμματα λογισμικού με χρήση πολλών μεταβλητών και πολλών βρόχων που μπορεί να μην είναι συνθέσιμα ή να συντίθενται με μη ικανοποιητικό αποτέλεσμα
  - *Πάντα έχουμε στο μυαλό μας το ψηφιακό κύκλωμα που αντιστοιχεί στον κώδικα HDL που γράφουμε*
    - Διαχειριζόμαστε την πολυπλοκότητα με κατάλληλη αφαίρεση και εφαρμόζοντας την ιεραρχία, την τμηματικότητα και την κανονικότητα
    - Η περιγραφή γίνεται πάντα στο επίπεδο μεταφοράς καταχωρητή, (register transfer level – RTL), ώστε να είναι συνθέσιμη
      - *Υψηλότερο επίπεδο αφαίρεσης από τις πύλες*

Περιγραφή Ψηφιακού Συστήματος σε Επίπεδο Μεταφοράς Καταχωρητή - RTL

- Περιγράφεται κάθε **καταχωρητής (REG)** του συστήματος, καθώς και η **συνδυαστική λογική (CL)** ανάμεσα στους καταχωρητές



Γλώσσες περιγραφής υλικού (HDL)

- Κατά τη δεκαετία του 1990 οι σχεδιαστές ανακάλυψαν ότι ήταν πολύ πιο παραγωγικό να δουλεύουν σε ένα υψηλότερο επίπεδο αφαίρεσης από τις πύλες αφήνοντας το έργο της ελαχιστοποίησης των πυλών σε ένα εργαλείο CAD
- Οι δύο κορυφαίες γλώσσες περιγραφής υλικού είναι:
  - ***SystemVerilog, για εμπορικές εφαρμογές (C-like)***
  - ***VHDL, για στρατιωτικές και διαστημικές εφαρμογές (ADA-like)***

  που βασίζονται σε παρόμοιες αρχές, αλλά έχουν διαφορετική σύνταξη
- Η VHDL είναι περισσότερο αναλυτική (απαιτεί περισσότερο κώδικα) και είναι πιο πολύπλοκη, αλλά και πιο ακριβής από τη SystemVerilog,
  - *όπως θα περιμένατε ίσως από μια γλώσσα που έχει αναπτυχθεί από κάποια ειδική επιτροπή για αμυντικές και διαστημικές εφαρμογές*

SystemVerilog

- Η Verilog αναπτύχθηκε το 1984 από την εταιρεία Gateway Design Automation, αρχικά, για την επαλήθευση λογικής με προσομοίωση
- Το 1989 η εταιρεία Gateway αγοράστηκε από την εταιρεία Cadence, και το 1990 η Verilog μετατράπηκε σε ανοικτό πρότυπο υπό την εποπτεία του οργανισμού Open Verilog International
- Το 1995 η γλώσσα έγινε πρότυπο (standard) του Ινστιτούτου IEEE
- Το 2005 η γλώσσα επεκτάθηκε, με απώτερο σκοπό τη βελτιστοποίηση κάποιων εκκεντρικών χαρακτηριστικών και την καλύτερη υποστήριξη για τη μοντελοποίηση, σύνθεση και επαλήθευση
- Οι συγκεκριμένες επεκτάσεις έχουν συγχωνευθεί σε ένα γλωσσικό πρότυπο, το οποίο πλέον ονομάζεται SystemVerilog (IEEE STD 1800-2009).
- Τα ονόματα αρχείων της SystemVerilog συνήθως έχουν προέκταση .sv

VHDL

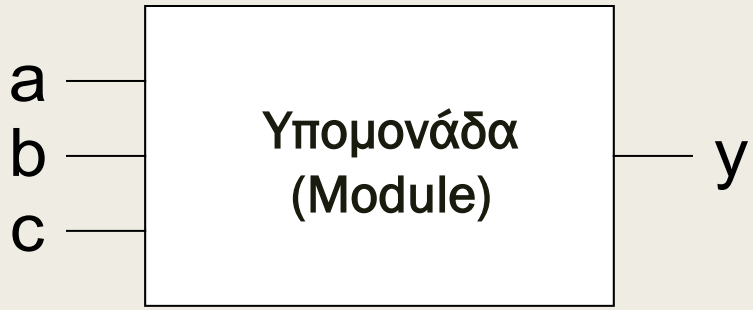
- Το αρκτικόλεξο VHDL προέρχεται από τα αρχικά της έκφρασης:

Very High Speed Integrated Circuits
Hardware Description Language

- Δημιουργήθηκε αρχικά στα πλαίσια του ερευνητικού προγράμματος Very High Speed Integrated Circuits, που χρηματοδότησε το Υπουργείο Αμύνης των ΗΠΑ (στις αρχές του 1980) με σκοπό την περιγραφή της δομής και της λειτουργίας του υλικού, που παραλαμβάνεται από πολλούς διαφορετικούς κατασκευαστές
- Αν και αρχικά επινοήθηκε για σκοπούς τεκμηρίωσης, γρήγορα χρησιμοποιήθηκε για την **περιγραφή**, τη **μοντελοποίηση,** την **επαλήθευση λογικής με προσομοίωση** και τη **σύνθεση** των ψηφιακών συστημάτων
- Τυποποιήθηκε από το Ινστιτούτου IEEE, αρχική έκδοση 1987, τελική έκδοση 1993, επεκτάσεις στην έκδοση 2008 (IEEE STD 1076-2008)
  - *Ελέγχουμε, εάν υποστηρίζεται από το εργαλείο CAD η έκδοση 2008*
- Τα ονόματα αρχείων της VHDL συνήθως έχουν προέκταση.vhd

Υπομονάδα (Module)

- Ένα τμήμα υλικού με εισόδους και εξόδους ονομάζεται **υπομονάδα**

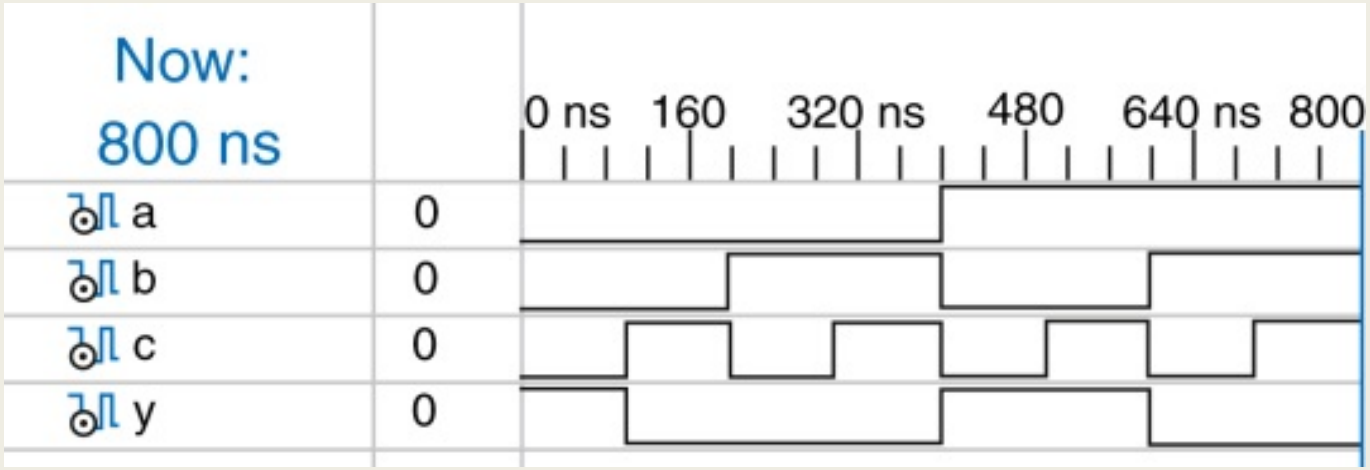


- Η λειτουργικότητα των υπομονάδων περιγράφεται με δύο τρόπους:
  - *ο πρώτος τρόπος βασίζεται στην **περιγραφή της συμπεριφοράς** της υπομονάδας*
  - *ο άλλος τρόπος βασίζεται στην **περιγραφή της δομής** της υπομονάδας.*
- Αντίστοιχα, προκύπτουν δύο διακριτά **μοντέλα περιγραφής της λειτουργικότητας** των υπομονάδων:
  - *τα μοντέλα περιγραφής συμπεριφοράς (behavioral models) που περιγράφουν τι κάνει μια υπομονάδα,*
  - *και τα μοντέλα περιγραφής δομής (structural models), που περιγράφουν πώς κατασκευάζεται η υπομονάδα από απλούστερα στοιχεία*

Προσομοίωση

- Παρακάτω φαίνεται η κυματομορφή που προέκυψε από μία προσομοίωση της υπομονάδας που υλοποιεί την εξίσωση Boole:

$$Y = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}C$$



- Εξετάζοντας όλες τις κυματορμοφές καταλήγουμε ότι η υπομονάδα όντως λειτουργεί σωστά
- Η έξοδος Y έχει την τιμή TRUE όταν οι είσοδοι A, B και C είναι 000, 100 ή 101, όπως δηλαδή ορίζει η εξίσωση Boole

Σφάλματα

- Τα λάθη κατά τη σχεδίαση υλικού ονομάζονται **σφάλματα (bugs)**
- Η διαδικασία της **δοκιμής (testing)** που χρησιμοποιείται για τον έλεγχο της ορθής λειτουργίας ενός συστήματος είναι χρονοβόρα
- Ο **εντοπισμός της αιτίας των λαθών** κατά τη δοκιμή ενός συστήματος μπορεί να αποδειχθεί **εξαιρετικά δύσκολος**, επειδή μπορούν να παρατηρηθούν μόνο σήματα που δρομολογούνται στους ακροδέκτες του τσιπ
  - *Για να παρατηρήσει κανείς απευθείας **τι συμβαίνει μέσα στο τσιπ** πρέπει να ενσωματώσει ένα **Logic Analysis Core***
- Η **προσομοίωση της λογικής** είναι απολύτως απαραίτητη για την **επαλήθευση της ορθής σχεδίασης και την αποσφαλμάτωση** ενός ψηφιακού συστήματος προτού αυτό κατασκευαστεί
  - *Στη συνέχεια το ψηφιακό σύστημα **υλοποιείται σε τεχνολογία FPGA***

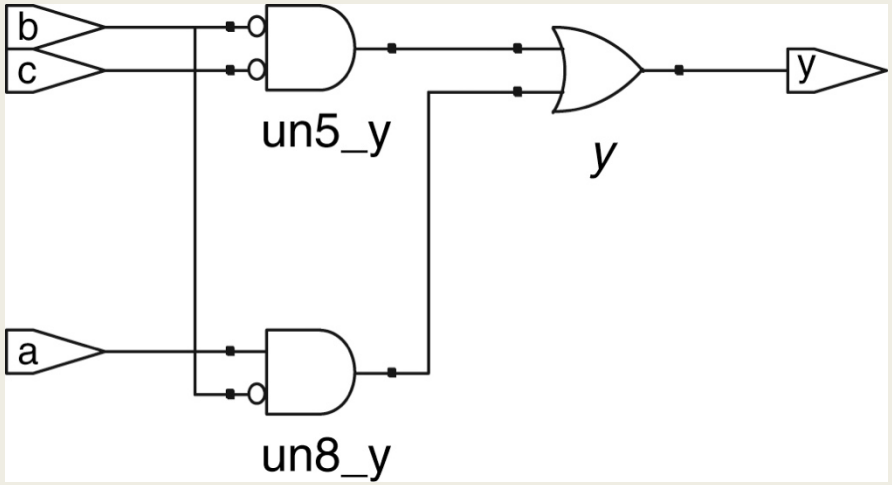
Σύνθεση

- Η σύνθεση της λογικής μετασχηματίζει τον κώδικα HDL σε μια **λίστα συνδέσεων στο επίπεδο της λογικής πύλης** (gate-level netlist)
  - *Περιγράφει το υλικό (τις λογικές πύλες και τα σύρματα που τις συνδέουν)*
- Το εργαλείο σύνθεσης λογικής (logic synthesizer) μπορεί να προχωρήσει σε **βελτιστοποιήσεις** για
  - *Να μειώσει την ποσότητα του απαιτούμενου υλικού*
  - *Να αυξήσει τη συχνότητα λειτουργίας*
- Η λίστα συνδέσεων μπορεί να έχει τη μορφή αρχείου κειμένου ή μπορεί να σχεδιάζεται σαν ένα **σχηματικό διάγραμμα** ώστε να διευκολύνεται η οπτικοποίηση του ψηφιακού κυκλώματος

Σύνθεση

- Στο παρακάτω σχήμα φαίνεται το **αποτελέσμα της σύνθεσης** της λογικής για την υπομονάδα που υλοποιεί την εξίσωση Boole:

$$\mathbf{Y} = \overline{\mathbf{A}}\overline{\mathbf{B}}\overline{\mathbf{C}} + A\overline{B}\overline{C} + A\overline{B}\mathbf{C}$$



- Οι τρεις πύλες AND, με τρεις εισόδους η καθεμία, ελαχιστοποιούνται σε δύο πύλες AND, με δύο εισόδους η καθεμία
- $\mathbf{Y} = \overline{\mathbf{A}}\overline{\mathbf{B}}\overline{\mathbf{C}} + A\overline{B}\overline{C} + A\overline{B}\mathbf{C} = \overline{\mathbf{B}}\overline{\mathbf{C}} + A\overline{B}$

Πρόγραμμα Δοκιμών

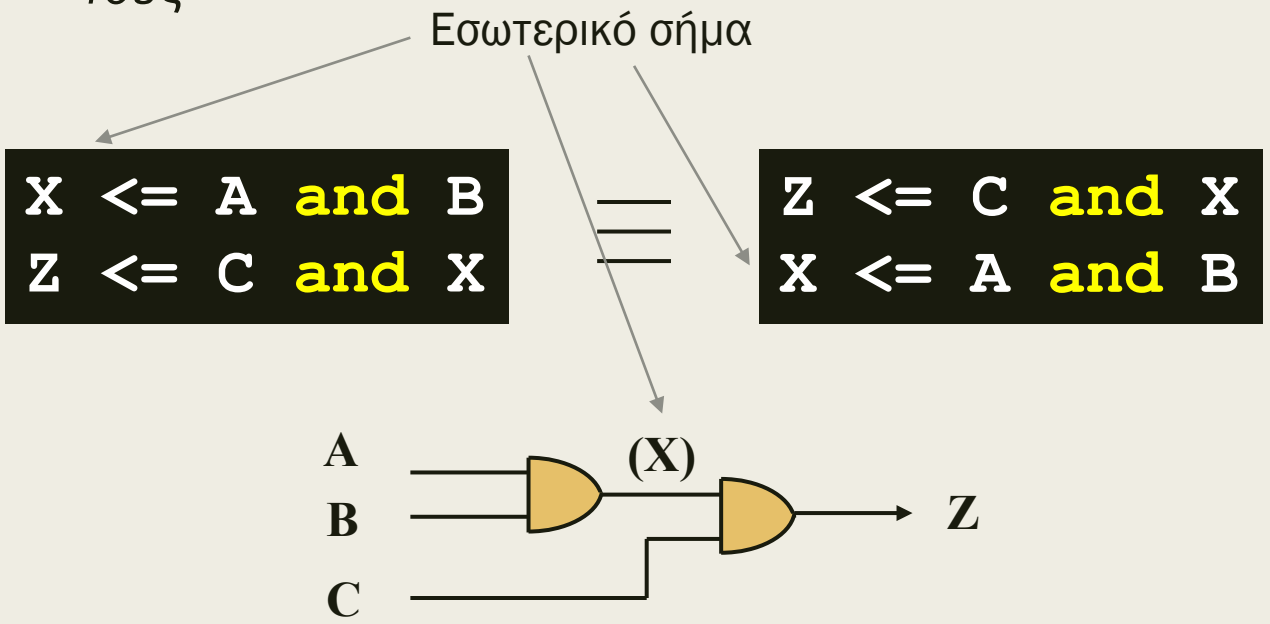
- Οι περιγραφές κυκλωμάτων σε γλώσσες περιγραφής υλικού HDL **μοιάζουν** με τον κώδικα των γλωσσών προγραμματισμού
- Δεν είναι όμως εφικτή η σύνθεση όλων των εντολών των γλωσσών HDL σε υλικό
  - *Για παράδειγμα, μια εντολή που τυπώνει αποτελέσματα στην οθόνη κατά τη διάρκεια της προσομοίωσης δεν «μεταφράζεται» σε υλικό*
- Για την προσομοίωση των υπομονάδων χρησιμοποιούμε τα **προγράμματα δοκιμών** (testbench)
  - *Περιέχουν τον κώδικα σε HDL με τον οποίο τροφοδοτούμε τις εισόδους μιας υπομονάδας, ώστε να ελέγξουμε αν τα αποτελέσματα στις εξόδους είναι σωστά, και θα τυπώσουμε τυχόν εμφανιζόμενες διαφορές στις κυματομορφές (αναμενόμενες έναντι πραγματικές) που προκύπτουν κατά την προσομοίωση*
  - *Ο κώδικας ενός προγράμματος δοκιμών **προορίζεται μόνο για προσομοίωση** και **δεν είναι συνθέσιμος***

Ιδιωματισμοί

- Ο καλύτερος τρόπος για να μάθετε μια γλώσσα περιγραφής υλικού είναι μέσα από **παραδείγματα κωδίκων HDL**
- Οι γλώσσες περιγραφής υλικού διαθέτουν συγκεκριμένους τρόπους με τους οποίους περιγράφουν διάφορα είδη λογικής που ονομάζονται **ιδιωματισμοί** (idioms)
- Όταν πρέπει να περιγράψετε ένα συγκεκριμένο είδος λογικής, αναζητήστε κάποιο παρόμοιο παράδειγμα και προσαρμόστε το στις δικές σας ανάγκες
- Οι ιδιωματισμοί που θα παραθέσουμε στη συνέχεια είναι σε γλώσσα VHDL και στοχεύουν στη **σωστή σύνθεση**, είναι συνθέσιμοι **σε όλα τα εργαλεία CAD** και είναι **ανεξάρτητοι της τεχνολογίας υλοποίησης** (technology agnostic)
  - *Χρησιμοποιούμε ένα **μικρό υποσύνολο** των εντολών της γλώσσας VHDL*

Ταυτόχρονες εντολές στη γλώσσα VHDL

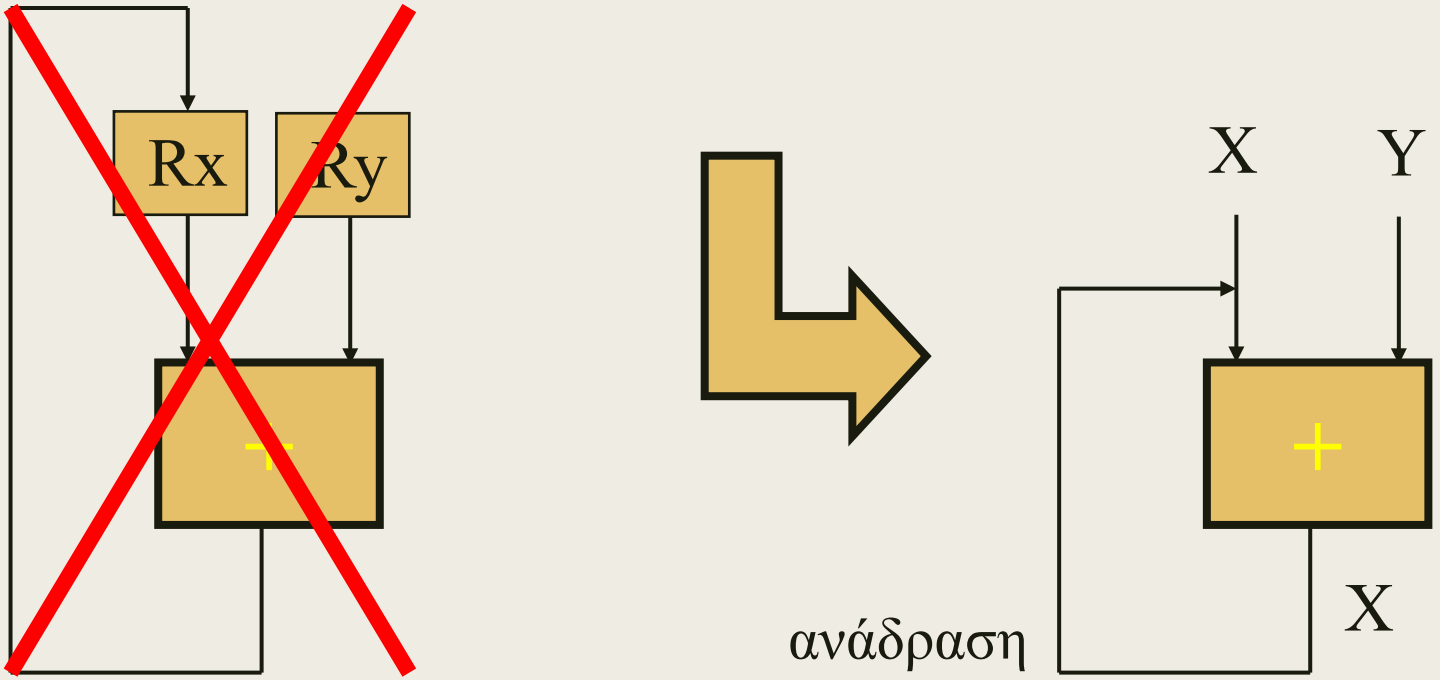
- **Ταυτόχρονες** εντολές (concurrent statements)
  - *εκτελούνται στον ίδιο χρόνο παράλληλα*
  - *η συμπεριφορά τους είναι **ανεξάρτητη** από τη σειρά εμφάνισής τους*



Η φύση του υλικού (hardware) απαιτεί την υποστήριξη ταυτόχρονων εντολών

Ταυτόχρονες εντολές – Προσοχή!

```
X <= X + Y
```



Δεν είναι όπως στο λογισμικό (software)

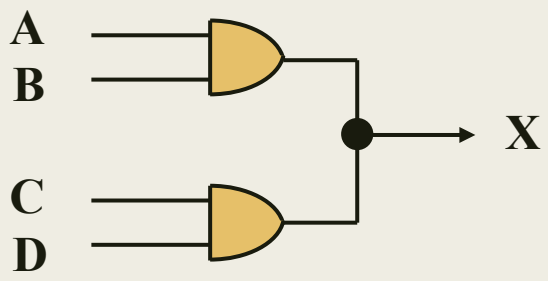
Ακολουθιακές εντολές στη γλώσσα VHDL

- **Ακολουθιακές** εντολές (sequential statements)
  - *εμφανίζονται μέσα σε **δομή διεργασίας (process)** για να ξεχωρίζουν από τις ταυτόχρονες εντολές*
  - *εκτελούνται **μόνο μία φορά** στη σειρά*
  - *η συμπεριφορά τους εξαρτάται από τη σειρά εμφάνισής τους*
  - *αλγοριθμική περιγραφή, όπως στο λογισμικό*

Ταυτόχρονες έναντι ακολουθιακών εντολών

Ταυτόχρονες εντολές

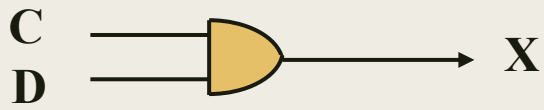
```
X <= A and B
X <= C and D
```



Ακολουθιακές εντολές μέσα σε δομή process

```
X <= A and B
X <= C and D
```

αγνοείται



Αν A=B=C=D='0' =>X='0'
Αν A=B=C=D='1' =>X='1'
Αν A=B='1', C=D='0' =>X='X'

Η οντότητα (entity) στη γλώσσα VHDL

- Περιγράφει τη διασύνδεση μίας ιεραρχικής υπομονάδας, **χωρίς να προσδιορίζει τη συμπεριφορά της** σαν μαύρο κουτί (black box)
- Η διασύνδεση της υπομονάδας περιγράφεται με μία δήλωση των **διαύλων (ports - signals)**

```
entity entity_name is -- σχόλια
    port (
        signal_name: mode signal_type;
        signal_name: mode signal_type;
        ...
        signal_name: mode signal_type);
end entity_name;
```

Η οντότητα (entity) στη γλώσσα VHDL

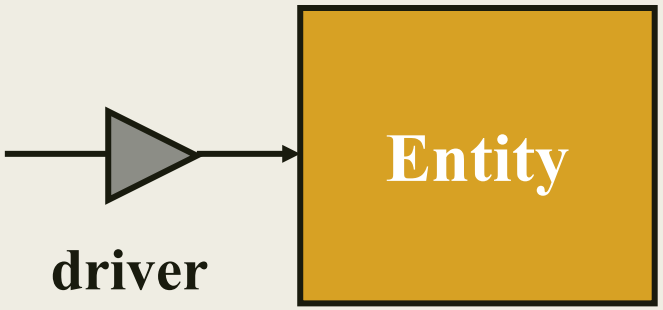
- entity_name: το όνομα της οντότητας
- signal_name: το όνομα του σήματος
  (εάν είναι πολλά σήματα χωρίζονται με κόμμα)
- mode: η κατεύθυνση του driver του σήματος
  - *in: είσοδος της οντότητας*
  - *out: έξοδος της οντότητας*
  - *inout: είσοδος ή έξοδος της οντότητας (bidirectional),*
- signal_type: ο τύπος του σήματος (STD_LOGIC)

Ονόματα και ετικέτες στη γλώσσα VHDL

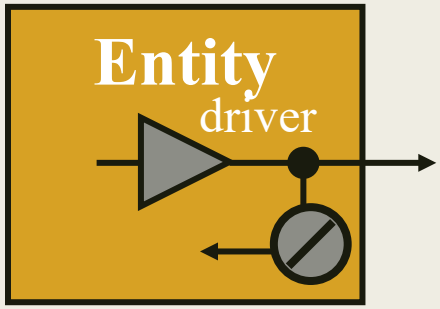
- Είναι **μοναδικά** μέσα σε μία συγκεκριμένη οντότητα (και αρχιτεκτονική)
- Χρησιμοποιούνται οι χαρακτήρες: **a-z, A-Z, 0-9, "_"**
- Δεν χρησιμοποιούνται οι χαρακτήρες, όπως: **+, -, !, &**
- Δεν χρησιμοποιούνται ούτε **σημεία στίξης** στα ονόματα και τις ετικέτες, ούτε διπλό "_", δηλαδή **"__"**
- Δεν διαχωρίζονται κεφαλαία γράμματα από μικρά
- Ο πρώτος χαρακτήρας είναι **αλφαβητικός**
- Το μέγεθος περιορίζεται συνήθως στους **32 χαρακτήρες**
- **Προσοχή στις δεσμευμένες λέξεις!**
- **Δεν παίζουν ρόλο τα κενά και τα carriage returns**
  - ***Η εντολή τελειώνει με “;”***
- **Τα σχόλια σε μία γραμμή έπονται του διπλού "-"**

Σημασία του mode στο port signal

Mode **in**

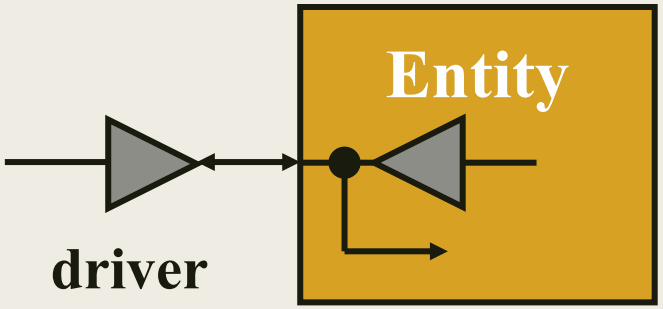


Mode **out**

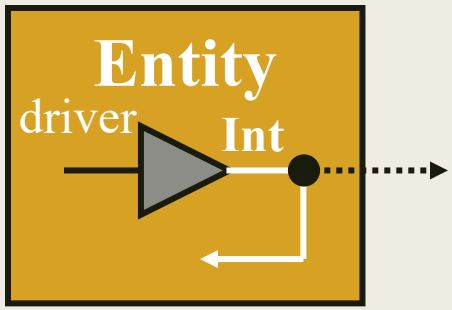


Προσοχή! Τα σήματα εξόδου δεν χρησιμοποιούνται σαν είσοδοι εντός της οντότητας

Mode **inout**



Mode **out**



Απαιτείται χρήση εσωτερικού σήματος (int) που χρησιμοποιείται σαν είσοδος εντός της οντότητας και συνδέεται με σήμα εξόδου

Σημασία του signal_type στο port signal

	Σημασία τιμών στα σήματα τύπου std_logic	
τιμή	Modeling for simulation	Synthesis
U	Unitialized	Unitialized
X	Strong driven unknown	Don't care
0	Strong driven 0	0
1	Strong driven 1	1
Z	High impedance	High impedance
W	Weakly driven unknown	Don't care
L	Weakly driven 0	0
H	Weakly driven 1	1
-	Don't care	Don't care

Ο τύπος του σήματος STD_LOGIC είναι μέρος του πακέτου IEEE.std_logic_1164 της βιβλιοθήκης IEEE. Για να χρησιμοποιηθεί όλο το πακέτο δηλώνουμε:

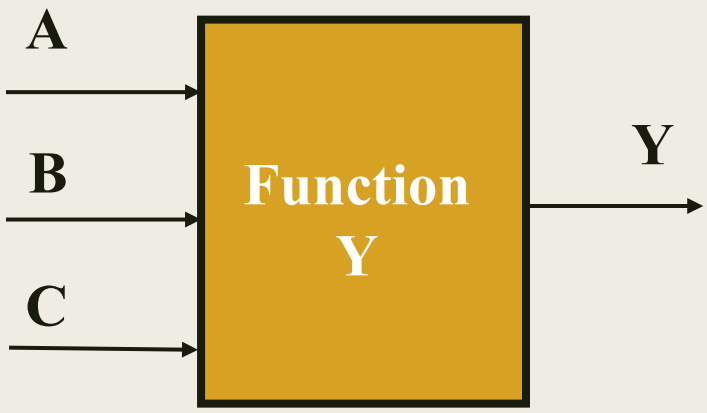
```
library IEEE;
use IEEE.std_logic_1164.all;
```

Οντότητα μίας υπομονάδας στη γλώσσα VHDL

- Παρακάτω φαίνεται η οντότητα της υπομονάδας με όνομα Function_Y που υλοποιεί την εξίσωση Boole:

$$Y = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}C$$

```
entity Function_Y is
    port (
        A: in STD_LOGIC;
        B: in STD_LOGIC;
        C: in STD_LOGIC;
        Y: out STD_LOGIC);
end Function_Y;
```



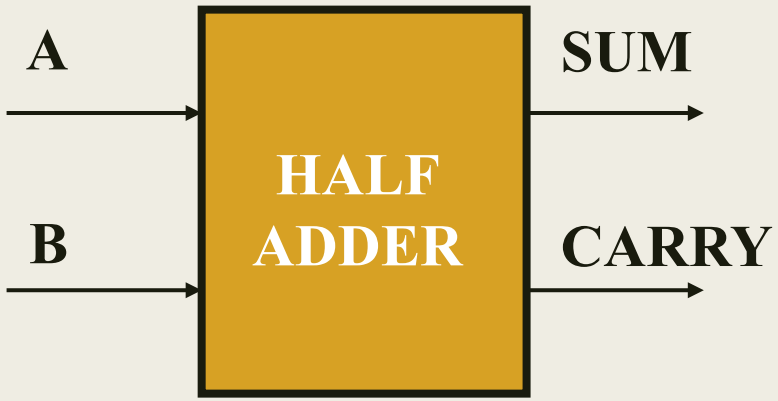
Η οντότητα του ημιαθροιστή στη VHDL

- Παρακάτω φαίνεται η οντότητα του ημιαθροιστή που υλοποιεί τις εξισώσεις Boole:

$$\mathbf{SUM = A \oplus B}$$

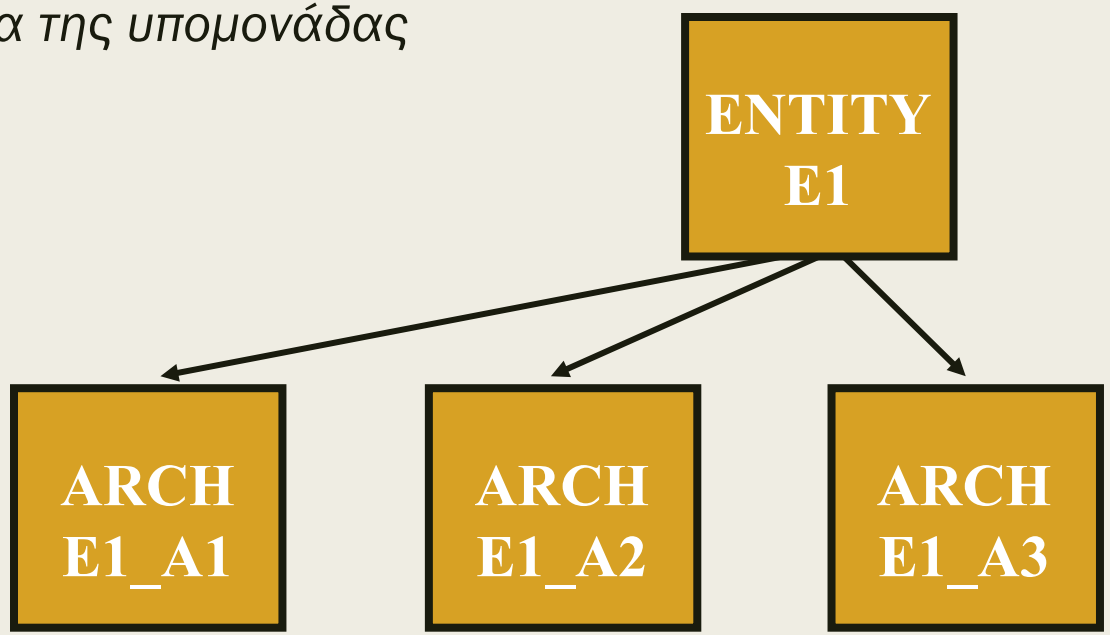
$$\mathbf{CARRY = AB}$$

```
entity HALF_ADDER is
    port (
        A: in STD_LOGIC;
        B: in STD_LOGIC;
        SUM: out STD_LOGIC;
        CARRY: out STD_LOGIC);
end HALF_ADDER;
```



Οντότητα και αρχιτεκτονική στη γλώσσα VHDL

- Στη γλώσσα VHDL υπάρχουν **δύο κύρια στοιχεία** που περιγράφουν μια υπομοναδα (module)
  - ***Η οντότητα****, η οποία περιγράφει τη διασύνδεση της υπομονάδας (είσοδοι, έξοδοι), και*
  - ***Η αρχιτεκτονική****, η οποία περιγράφει τη λειτουργία της υπομονάδας*



Η αρχιτεκτονική (architecture) στη VHDL

- Περιγράφει τη λειτουργία μίας υπομονάδας με έναν από τους ακόλουθους τρόπους:
  - *ένα σύνολο από **ταυτόχρονες εντολές ανάθεσης** (concurrent assignment statements) για **περιγραφή dataflow***
  - *ένα σύνολο από **ακολουθιακές εντολές ανάθεσης** (sequential assignment statements) μέσα σε **δομές διεργασίας (process)** για **περιγραφή συμπεριφοράς (behavioral)***
  - *ένα σύνολο από **διασυνδεδεμένα στοιχεία (components)** για **περιγραφή δομής (structural)**, όπως γίνεται σε μία σχεδίαση με σχηματικό διάγραμμα*
  - *κάθε συνδυασμός από τα πιο πάνω*

Περιγραφή dataflow στη VHDL

```
architecture arch_name of entity_name is
    signal signal_name: signal_type;
begin
    concurrent_statement;
    ...
    concurrent_statement;
end arch_name;
```

- **arch_name:** το όνομα της αρχιτεκτονικής
- **entity_name:** το όνομα της οντότητας
- **signal_name:** το όνομα του σήματος
  (εάν είναι πολλά σήματα χωρίζονται με κόμμα)
  - *στις δηλώσεις σημάτων (μετά το signal) το σήμα είναι μία **εσωτερική διασύνδεση** της υπομονάδας*
- **signal_type:** ο τύπος του σήματος (STD_LOGIC)

Περιγραφή dataflow στη VHDL

- **Ταυτόχρονες εντολές ανάθεσης σήματος** (concurrent_signal_assignment_statements)

```
signal_name <= expression;
```

- **<=**: τελεστής ανάθεσης τιμής σε σήμα
- **expression**: έκφραση με σήματα και τελεστές
- **signal_name**: το όνομα του σήματος
  - *στις ταυτόχρονες εντολές ανάθεσης σήματος :*
    - στην έκφραση προσδιορίζονται σήματα που είναι **είσοδοι** στην υπομονάδα και δηλώνονται κατά τη δήλωση των διαύλων της οντότητας, και **εσωτερικές διασυνδέσεις** της υπομονάδας που δηλώνονται κατά τη δήλωση σημάτων
    - στο αριστερό μέρος της εντολής προσδιορίζεται σήμα που είναι **έξοδος** της υπομονάδας και δηλώνεται κατά τη δήλωση των διαύλων της οντότητας, ή **εσωτερική διασύνδεση** της υπομονάδας που δηλώνεται κατά τη δήλωση σημάτων

Περιγραφή dataflow στη VHDL

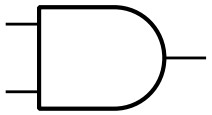
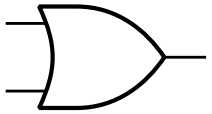
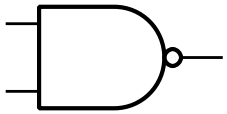
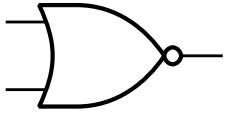
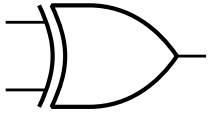
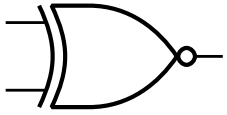
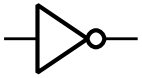
- **Εκτέλεση** ταυτόχρονων εντολών ανάθεσης σήματος (concurrent_signal_assignment_statements)

```
signal_name <= expression;
```

  - *Οι ταυτόχρονες εντολές ανάθεσης σήματος **εκτελούνται μόνο όταν υπάρξει αλλαγή τιμής στις εισόδους** (στα σήματα της δεξιάς πλευράς της ταυτόχρονης εντολής ανάθεσης σήματος).*
  - *Δεν προσδιορίζεται καθυστέρηση διάδοσης άλλη, εκτός από μία απειροελάχιστη καθυστέρηση διάδοσης, την **καθυστέρηση δέλτα** δ_{delay}, που δεν επηρεάζει τον χρονισμό του κυκλώματος*
  - *Η πραγματική καθυστέρηση διάδοσης θα προσδιορισθεί με την υλοποίηση σε μία συγκεκριμένη τεχνολογία*

Η καθυστέρηση δέλτα δ_{delay} δεν είναι πραγματική καθυστέρηση που επηρεάζει την προσομοίωση, αλλά απλώς ιεραρχεί τις μεταβάσεις που συμβαίνουν στα σήματα την ίδια χρονική στιγμή.

Λογικοί τελεστές και προτεραιότητα λογικών πράξεων στη VHDL

a **and** b	$a \cdot b$	
a **or** b	$a + b$	
a **nand** b	$\overline{a \cdot b}$	
a **nor** b	$\overline{a + b}$	
a **xor** b	$a \oplus b$	
a **xnor** b	$\overline{a \oplus b}$	
not a	$\overline{a}$	

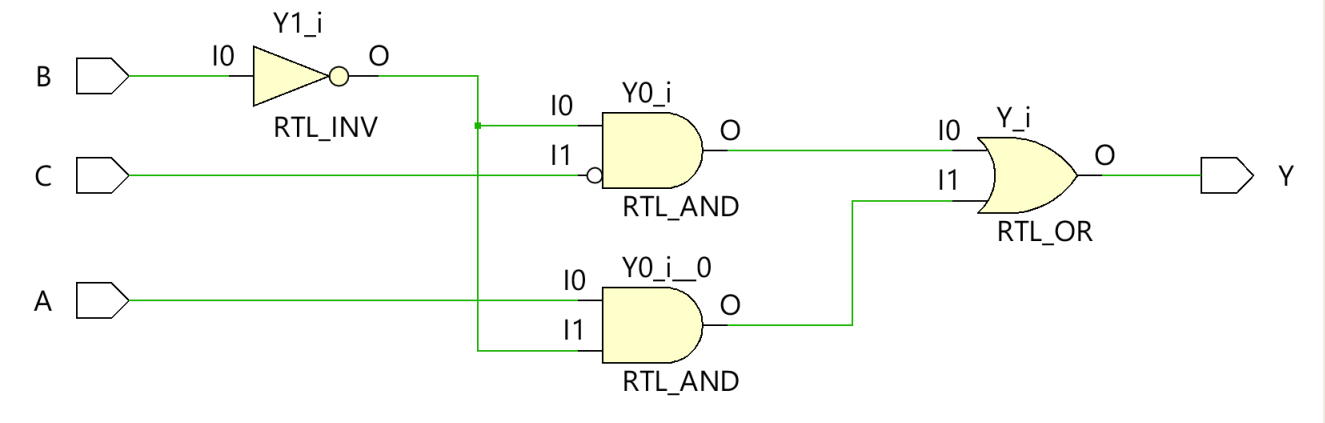
- Προτεραιότητα
  - *το **not** έχει την υψηλότερη προτεραιότητα*
  - *οι υπόλοιποι τελεστές έχουν **ίση προτεραιότητα***
  - *οι λογικές πράξεις εκτελούνται **από αριστερά προς τα δεξιά***
  - *χρησιμοποιούμε **παρενθέσεις** για να ξεκαθαρίσουμε τη σειρά εκτέλεσης των λογικών πράξεων*
- Τιμές bit στην VHDL
  - *'0' και '1'*

Η αρχιτεκτονική μίας υπομονάδας στη VHDL
Περιγραφή dataflow

- Παρακάτω φαίνεται η αρχιτεκτονική της υπομονάδας σε περιγραφή dataflow που υλοποιεί την εξίσωση Boole:
  - $\mathbf{Y = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}C = \bar{B}\bar{C} + A\bar{B}}$

```
architecture Y_DATAFLOW of Function_Y is
begin
    Y <= ((not B) and (not C)) or
         (A and (not B));
end Y_ DATAFLOW;
```

- Σχηματικό διάγραμμα RTL

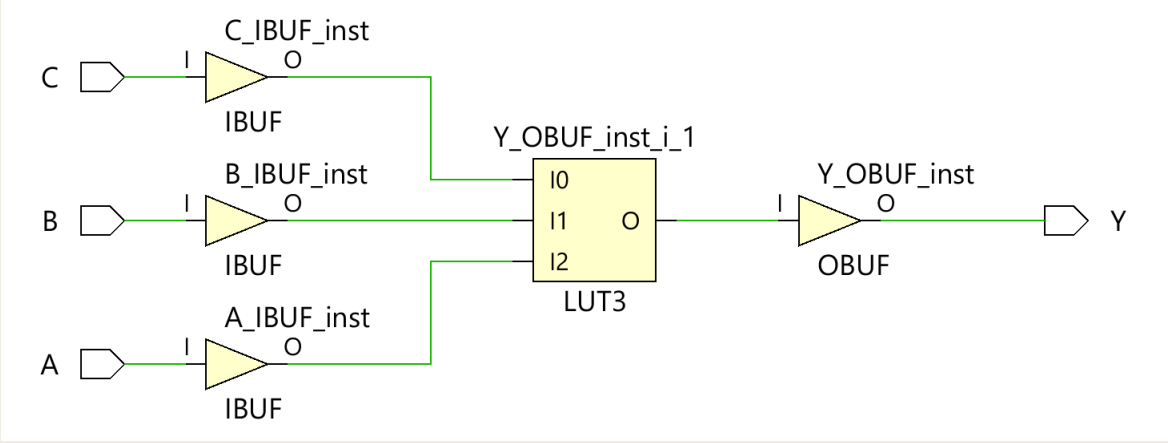


Η αρχιτεκτονική μίας υπομονάδας στη VHDL
Περιγραφή dataflow

- Παρακάτω φαίνεται η αρχιτεκτονική της υπομονάδας σε περιγραφή dataflow που υλοποιεί την εξίσωση Boole:
  - $\mathbf{Y = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}C = \bar{B}\bar{C} + A\bar{B}}$

```
architecture Y_DATAFLOW of Function_Y is
begin
    Y <= ((not B) and (not C)) or
         (A and (not B));
end Y_ DATAFLOW;
```

- Σχηματικό διάγραμμα σε τεχνολογία FPGA

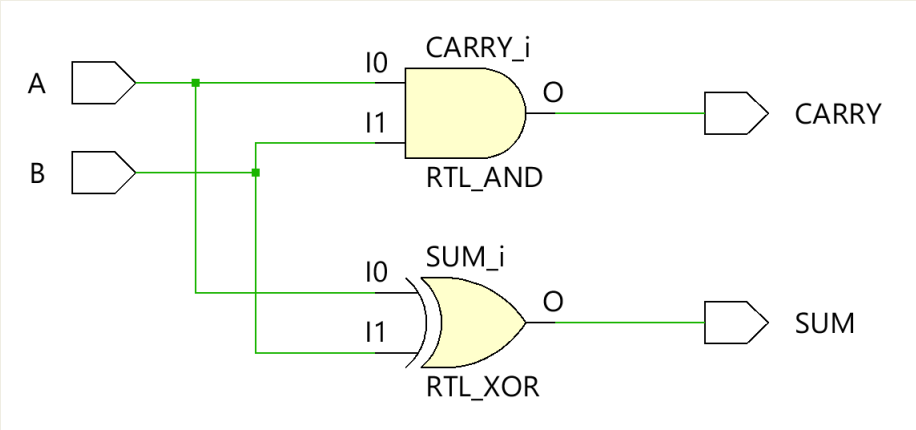


Η αρχιτεκτονική του ημιαθροιστή στη VHDL
Περιγραφή dataflow

- Παρακάτω φαίνεται η αρχιτεκτονική του ημιαθροιστή σε περιγραφή dataflow που υλοποιεί τις εξισώσεις Boole:
  - $\mathbf{SUM = A \oplus B, CARRY = AB}$

```
architecture HA_DATAFLOW of HALF_ADDER is
begin
    SUM <= A xor B;
    CARRY <= A and B;
end HA_ DATAFLOW;
```

- Σχηματικό διάγραμμα RTL

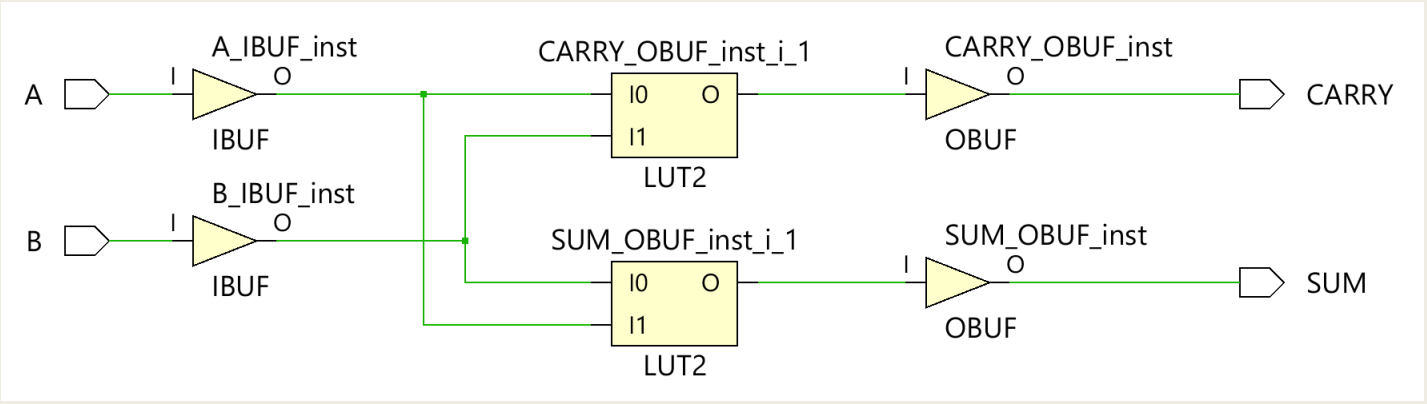


Η αρχιτεκτονική του ημιαθροιστή στη VHDL
Περιγραφή dataflow

- Παρακάτω φαίνεται η αρχιτεκτονική του ημιαθροιστή σε περιγραφή dataflow που υλοποιεί τις εξισώσεις Boole:
  - $\mathbf{SUM = A \oplus B, CARRY = AB}$

```
architecture HA_DATAFLOW of HALF_ADDER is
begin
    SUM <= A xor B;
    CARRY <= A and B;
end HA_ DATAFLOW;
```

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Η αρχιτεκτονική του ημιαθροιστή στη VHDL
Περιγραφή dataflow για προγράμματα δοκιμής

```
architecture HA_DATAFLOW of HALF_ADDER is
begin
    SUM <= A xor B after 10 ns;
    CARRY <= A and B after 5 ns;
end HA_ DATAFLOW;
```

Στα προγράμματα δοκιμής μπορεί να προσδιορισθεί για τις ανάγκες τις προσομοίωσης και καθυστέρηση διάδοσης με τη φράση after. Ο χρόνος αρχίζει να μετράει με την αλλαγή της τιμής σε μία από τις εισόδους. Το after αγνοείται κατά τη σύνθεση.

Περιγραφή συμπεριφοράς (behavioral) στη VHDL

```
architecture arch_name of entity_name is
begin
  label: process (signal_name, ..., signal_name)
    variable variable_name: variable_type;
  begin
    sequential_statement;
    ...
    sequential_statement;
  end process;
end arch_name;
```

Με την περιγραφή συμπεριφοράς προσδιορίζουμε τη λειτουργικότητα και όχι τη δομή της υπομονάδας.
Αν και οι εντολές μέσα σε μία διεργασία (process) αξιολογούνται ακολουθιακά, μετά τη σύνθεση προκύπτει ταυτόχρονη λογική

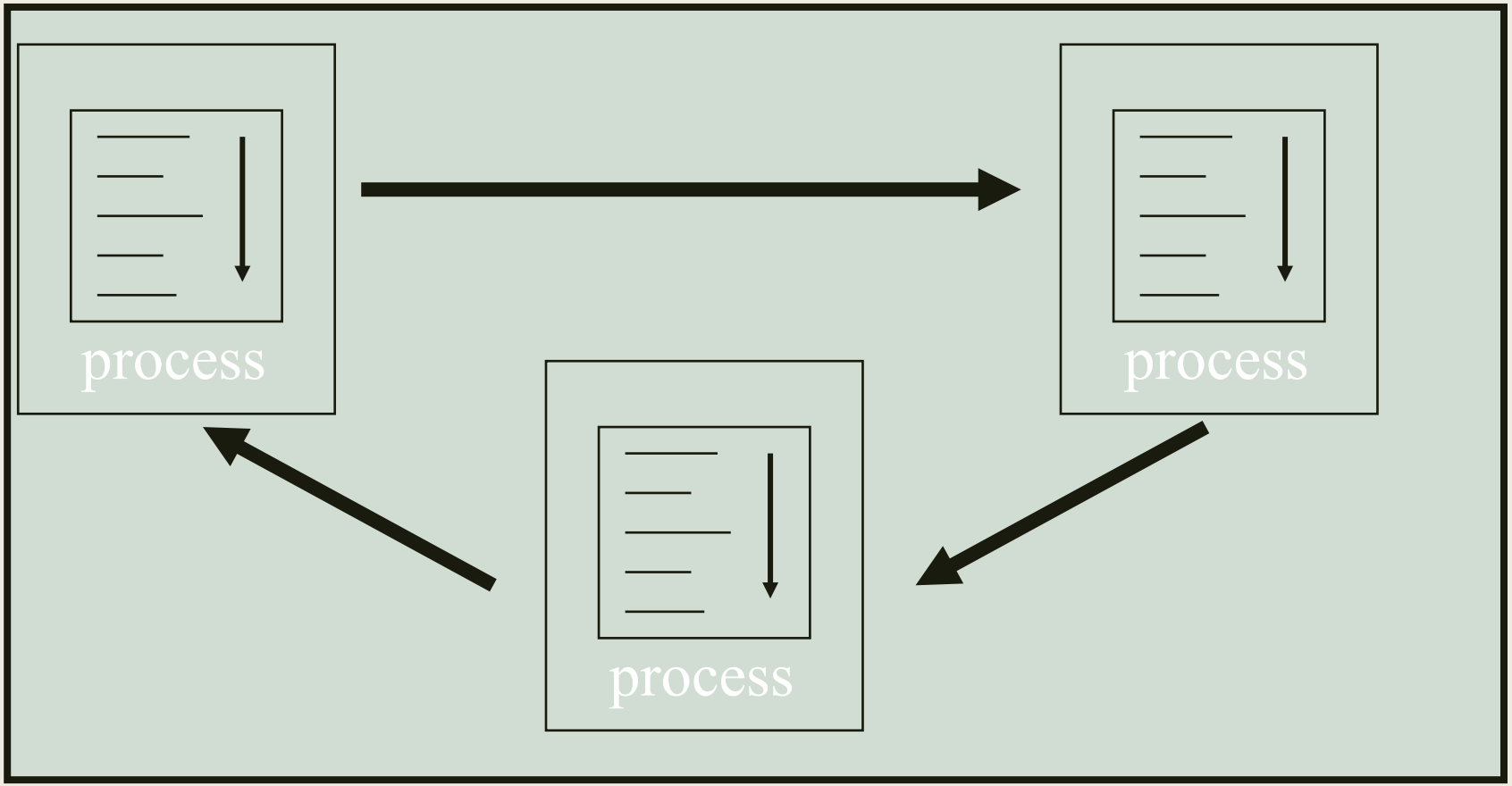
Περιγραφή συμπεριφοράς (behavioral) στη VHDL

- arch_name: το όνομα της αρχιτεκτονικής
- entity_name: το όνομα της οντότητας
- label: οι μοναδικές ετικέτες των διεργασιών
  - *μη υποχρεωτικό, αλλά χρήσιμο κατά τη σύνθεση γιατί τα παραγόμενα σήματα συνήθως συμπεριλαμβάνουν στο όνομά τους και την ετικέτα της διεργασίας από την οποία προέρχονται*
- process: η διεργασία περιέχει μία ομάδα από εντολές που εκτελούνται ακολουθιακά
  - *Οι ακολουθιακές εντολές εμπεριέχουν σήματα και μεταβλητές*

Περιγραφή συμπεριφοράς (behavioral) στη VHDL

- signal_name: το όνομα του σήματος
  (εάν είναι πολλά σήματα χωρίζονται με κόμμα)
  - *στις δηλώσεις των σημάτων (μετά το process) που απαρτίζουν τη **λίστα ευαισθησίας** (sensitivity list) το σήμα είναι **είσοδος** της υπομονάδας που δηλώνεται κατά τη δήλωση των διαύλων της οντότητας ή **εσωτερική διασύνδεση** της υπομονάδας*
  - *κάθε αλλαγή τιμής σήματος εισόδου που ανήκει στη **λίστα ευαισθησίας** οδηγεί στην ακολουθιακή εκτέλεση των εντολών της διεργασίας **μία φορά***
  - *εάν περισσότερες από μία εντολές αναθέτουν τιμή σε κάποιο σήμα λαμβάνεται υπόψη μόνο η **τελευταία ακολουθιακή εντολή***
  - ***Προσοχή στη δήλωση των σημάτων στη λίστα ευαισθησίας***
    - **μία ελλιπής δήλωση σημάτων στη λίστα ευαισθησίας θα οδηγήσει είτε σε μη σωστή σύνθεση, είτε σε ασυμφωνία της προσομοίωσης πριν και μετά τη σύνθεση και την υλοποίηση**

Περιγραφή συμπεριφοράς (behavioral) στη VHDL



Κάθε διεργασία (process) εκτελεί τις εντολές της ακολουθιακά, ενώ πολλές διεργασίες μαζί αλληλεπιδρούν ταυτόχρονα. Επίσης, ταυτόχρονα αλληλεπιδρούν εντολές ταυτόχρονης ανάθεσης και διεργασίες.

Περιγραφή συμπεριφοράς (behavioral) στη VHDL

- variable_name: το όνομα της μεταβλητής
  (εάν είναι πολλές μεταβλητές χωρίζονται με κόμμα)
  - *μέσα στις διεργασίες ορίζονται **τοπικές μεταβλητές** και **ΟΧΙ εσωτερικά σήματα***
  - *στις δηλώσεις μεταβλητών (μετά το **variable**) προσδιορίζονται μεταβλητές που μπορεί να μην έχουν τη φυσική σημασία του σήματος*
- variable_type: ο τύπος της μεταβλητής (STD_LOGIC)

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Σήματα vs μεταβλητές

- Διαφορά στην εφαρμογή της τιμής μίας ακολουθιακής εντολής ανάθεσης μεταβλητής ή σήματος μέσα σε μία διεργασία κατά την προσομοίωση:
  - *η μεταβλητή παίρνει νέα τιμή **άμεσα** με τον τελεστή ανάθεσης **:=**, αμέσως μόλις εκτελεστεί η αντίστοιχη εντολή μέσα στη διεργασία*
    - Δεν χρησιμοποιείται στην υλοποίηση ακολουθιακής λογικής
  - *σε αντίθεση, το σήμα παίρνει νέα τιμή **με καθυστέρηση δέλτα** δ_{delay}, με τον τελεστή ανάθεσης **<=**, στο **τέλος** της εκτέλεσης της διεργασίας*
  - *το σήμα **θυμάται την τιμή του** μέχρι να φτάσει το τέλος της εκτέλεσης της διεργασίας και να λάβει μία νέα τιμή*
    - Χρησιμοποιείται στην υλοποίηση ακολουθιακής λογικής

Η χρήση των μεταβλητών μειώνει σημαντικά το χρόνο της προσομοίωσης

Περιγραφή συμπεριφοράς (behavioral) στη VHDL

- Ακολουθιακές εντολές ανάθεσης σήματος (με μη άμεση εφαρμογή τιμής) (sequential_signal_assignment_statements)

```
signal_name <= expression;
```

- signal_name: το όνομα του σήματος
- expression: έκφραση με σήματα, μεταβλητές και τελεστές
  - *στις ακολουθιακές εντολές ανάθεσης σήματος :*
    - στην έκφραση προσδιορίζονται **σήματα**, που ανήκουν ή δεν ανήκουν στη λίστα ευαισθησίας, και **μεταβλητές** που δηλώνονται κατά τη δήλωση μεταβλητών
    - στο αριστερό μέρος της εντολής προσδιορίζεται σήμα που είναι **έξοδος** της υπομονάδας ή **εσωτερική διασύνδεση** της υπομονάδας

Περιγραφή συμπεριφοράς (behavioral) στη VHDL

- Ακολουθιακές εντολές ανάθεσης μεταβλητής (με άμεση εφαρμογή τιμής) (sequential_variable_assignment_statements)

```
variable_name := expression;
```

- variable_name: το όνομα της μεταβλητής
- expression: έκφραση με σήματα, μεταβλητές και τελεστές
  - *στις ακολουθιακές εντολές ανάθεσης μεταβλητής :*
    - στην έκφραση προσδιορίζονται σήματα, που ανήκουν ή δεν ανήκουν στη λίστα ευαισθησίας, και μεταβλητές που δηλώνονται κατά τη δήλωση μεταβλητών
    - στο αριστερό μέρος της εντολής προσδιορίζεται μεταβλητή που δηλώνεται κατά τη δήλωση των μεταβλητών
    - οι μεταβλητές επιδρούν τοπικά εντός της διεργασίας

Δεν χρησιμοποιείται στην υλοποίηση ακολουθιακής λογικής

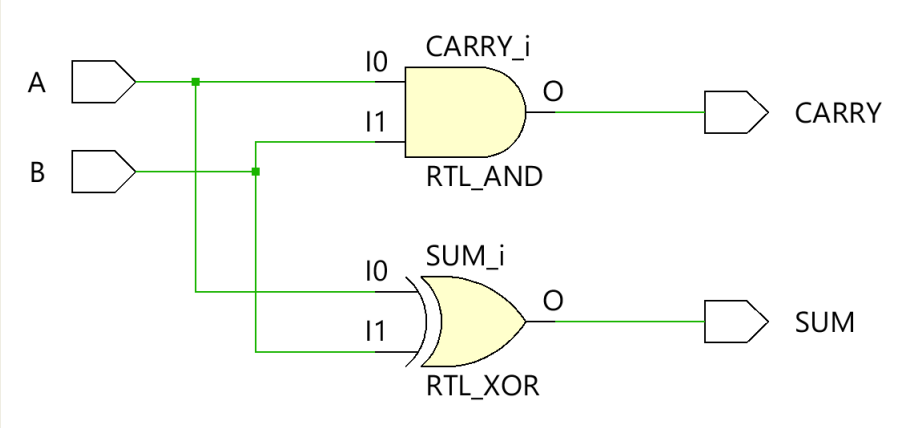
Η αρχιτεκτονική του ημιαθροιστή στη VHDL
Περιγραφή συμπεριφοράς

```
architecture HA_BEHAVIORAL of HALF_ADDER is
begin
  process (A, B)
  begin
    SUM <= A xor B;
    CARRY <= A and B;
  end process;
end HA_BEHAVIORAL;
```

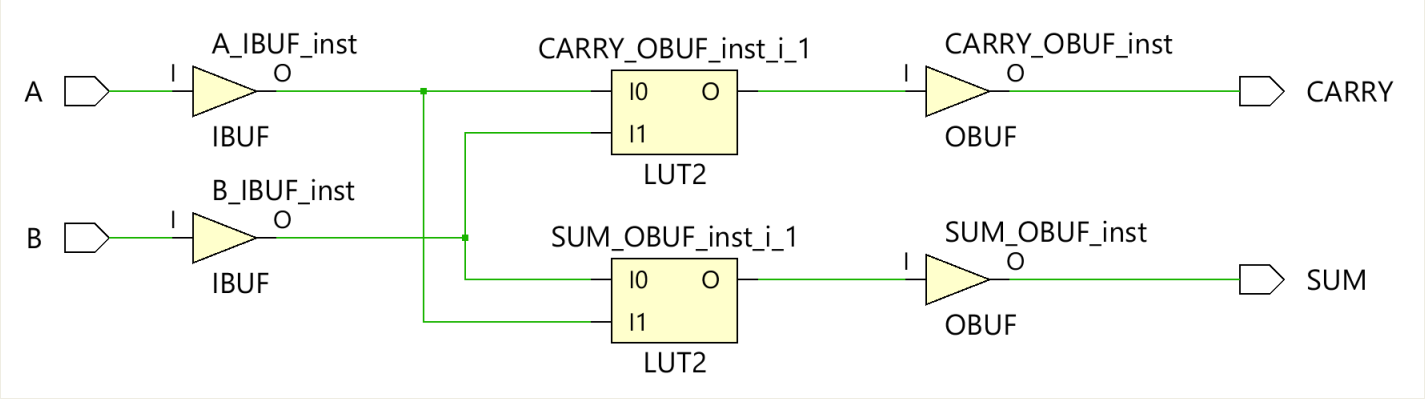
Οι ακολουθιακές εντολές ανάθεσης σήματος εκτελούνται η μία μετά την άλλη, μόνο όταν υπάρξει αλλαγή τιμής στις εισόδους που δηλώνονται στη λίστα ευαισθησίας (στα σήματα της δεξιάς πλευράς)

Η αρχιτεκτονική του ημιαθροιστή στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Περιγραφή δομής στη VHDL

```
architecture arch_name of entity_name is
    signal signal_name: signal_type;
    component comp_name
        port (
            signal_name: mode signal_type;
            ...
            signal_name: mode signal_type);
    end component;
    ...
begin
    concurrent_component_statement;
    ...
    concurrent_component_statement;
end arch_name;
```

Περιγραφή δομής στη VHDL

- arch_name: το όνομα της αρχιτεκτονικής
- entity_name: το όνομα της οντότητας
- comp_name: το όνομα του στοιχείου (component) που χρησιμοποιείται στην αρχιτεκτονική της οντότητας
  - *Το στοιχείο είναι μία ήδη **προκαθορισμένη οντότητα***
- signal_name: το όνομα του σήματος
  (εάν είναι πολλά σήματα χωρίζονται με κόμμα)
  - *στις δηλώσεις σημάτων (μετά το signal) το σήμα είναι μία **εσωτερική διασύνδεση** της υπομονάδας*
  - *στις δηλώσεις των διαύλων του στοιχείου (component) το σήμα είναι **είσοδος ή έξοδος του στοιχείου**, όπως προκύπτει από τη δήλωση των διαύλων της οντότητας του συγκεκριμένου στοιχείου*
- signal_type: ο τύπος του σήματος (STD_LOGIC)

Περιγραφή δομής στη VHDL

- Ταυτόχρονες εντολές στοιχείων (concurrent_component_statements)

```
label: comp_name port map (signal_name, ..);
```

- label: οι μοναδικές ετικέτες των στοιχείων
- comp_name: το όνομα του στοιχείου που χρησιμοποιείται στην αρχιτεκτονική της οντότητας
- signal_name: το όνομα του σήματος
  (εάν είναι πολλά σήματα χωρίζονται με κόμμα)
  - *το σήμα είναι μία διασύνδεση που αφορά τη συγκεκριμένη αρχιτεκτονική της οντότητας που χρησιμοποιεί το στοιχείο*
  - *αντιστοιχεί **αμφιμονοσήμαντα** στο αντίστοιχο σήμα της δήλωσης των διαύλων του στοιχείου*
    - Προσοχή στη διατήρηση της σειράς **των σημάτων**
    - Εναλλακτικά περιγράφουμε την αμφιμονοσήμαντη αντιστοιχία, ώστε ο κώδικας να διαβάζεται πιο εύκολα
      component_signal_name => entity_signal_name

Η αρχιτεκτονική του ημιαθροιστή στη VHDL
Περιγραφή δομής

- Παρακάτω φαίνεται η αρχιτεκτονική του ημιαθροιστή σε περιγραφή δομής (structural) που υλοποιεί τις εξισώσεις Boole:

$$\mathbf{SUM = A \oplus B}$$

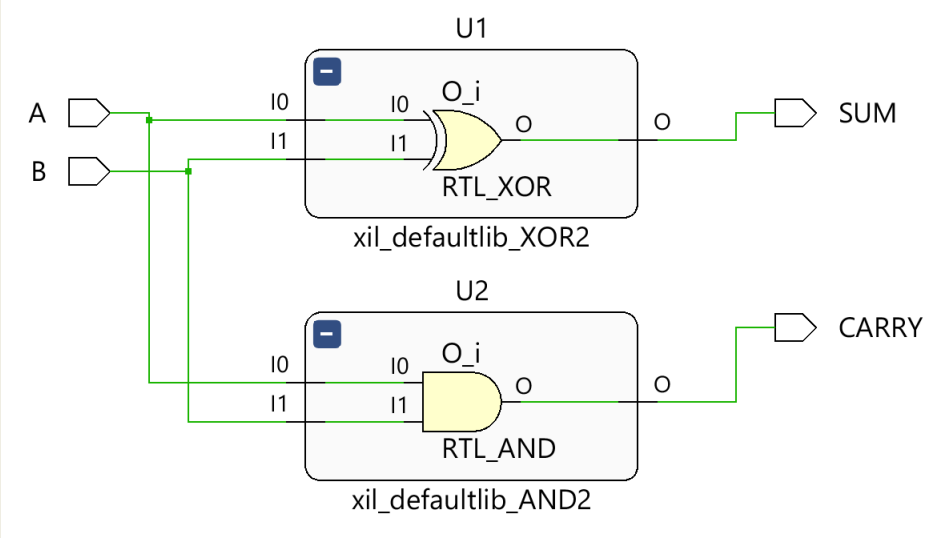
$$\mathbf{CARRY = AB}$$

```
architecture HA_STRUCTURAL of HALF_ADDER is
component XOR2
  port (O : out STD_LOGIC; I0 : in STD_LOGIC; I1 : in STD_LOGIC);
end component;
component AND2
  port (O : out STD_LOGIC; I0 : in STD_LOGIC; I1 : in STD_LOGIC);
end component;
begin
  U1: XOR2 port map (O => SUM, I0 => A, I1 => B);
  U2: AND2 port map (O => CARRY, I0 => A, I1 => B);
end HA_STRUCTURAL ;
```

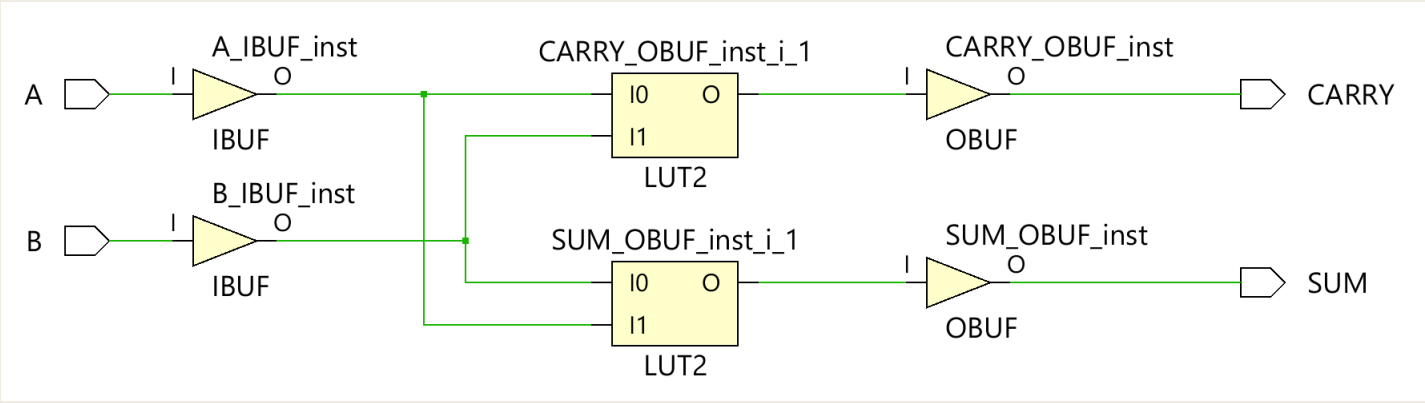
Τα στοιχεία XOR2 και AND2 έχουν ήδη προκαθοριστεί σαν οντότητες του project

Η αρχιτεκτονική του ημιαθροιστή στη VHDL
Σύνθεση περιγραφής δομής

- Σχηματικό διάγραμμα RTL (φαίνεται και η ιεραρχία)



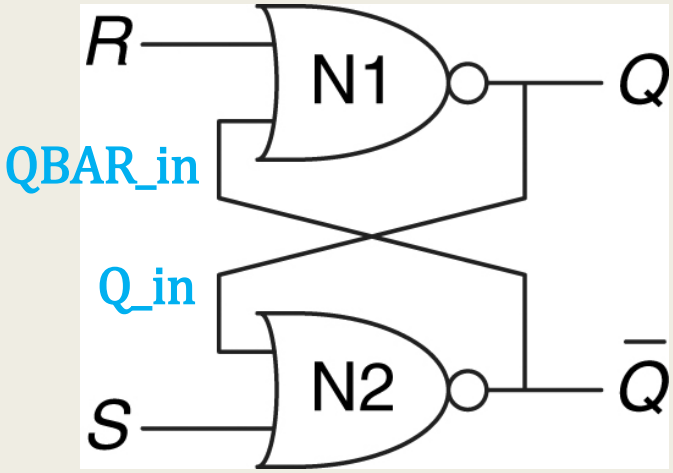
- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Το S-R Latch (Active High) στη VHDL

Περιγραφή Dataflow

```
entity SRL is port (
  S, R: in STD_LOGIC;
  Q, QBAR: out STD_LOGIC);
end SRL;
architecture SRL_DF of SRL is
signal Q_in, QBAR_in: STD_LOGIC;
begin
  QBAR_in <= S nor Q_in;
  Q_in <= R nor QBAR_in;
  QBAR <= QBAR_in;
  Q <= Q_in;
end SRL_DF;
```

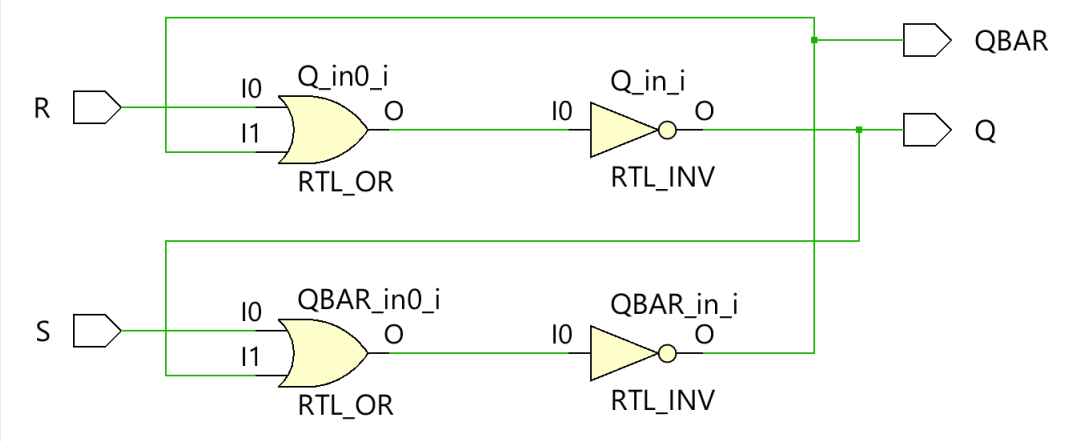


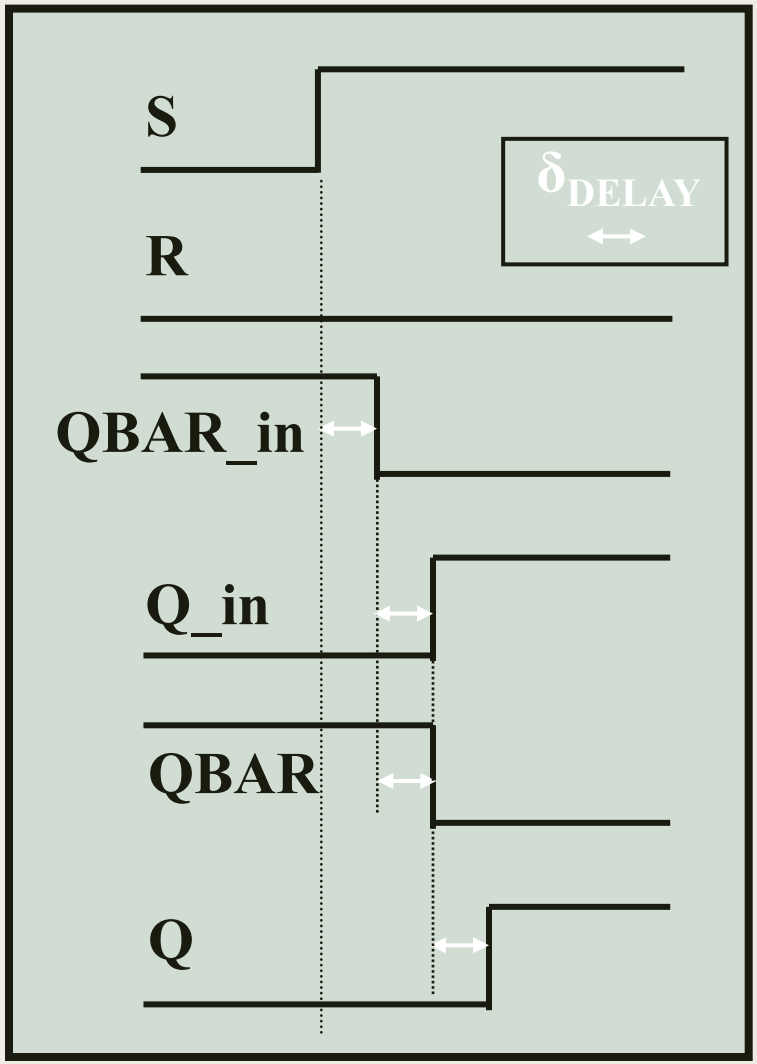
Q_in, QBAR_in:
Εσωτερικά σήματα (signals)

Το S-R Latch (Active High) στη VHDL

Περιγραφή Dataflow

```
entity SRL is port (
   S, R: in STD_LOGIC;
   Q, QBAR: out STD_LOGIC);
end SRL;
architecture SRL_DF of SRL is
signal Q_in, QBAR_in: STD_LOGIC;
begin
   QBAR_in <= S nor Q_in;
   Q_in <= R nor QBAR_in;
   QBAR <= QBAR_in;
   Q <= Q_in;
end SRL_DF;
```

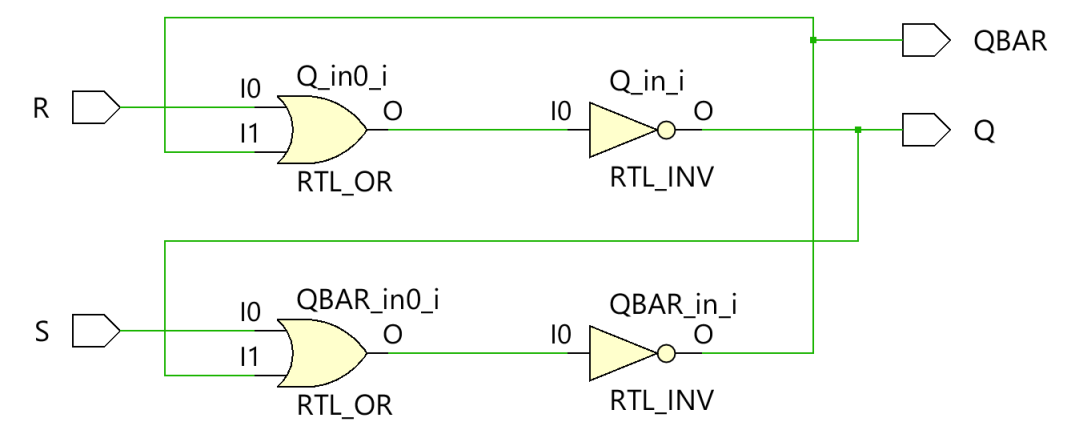




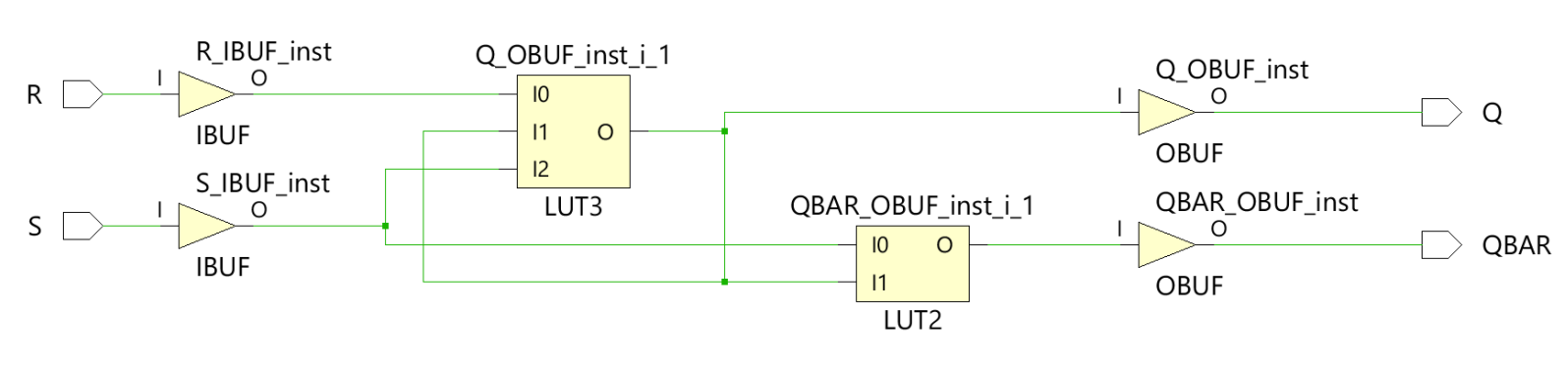
Η ενημέρωση των σημάτων γίνεται με καθυστέρηση δέλτα δ_{delay} αμέσως μόλις εκτελεστεί η αντίστοιχη εντολή

Το SR Latch στη VHDL
Σύνθεση περιγραφής Dataflow

- Σχηματικό διάγραμμα RTL

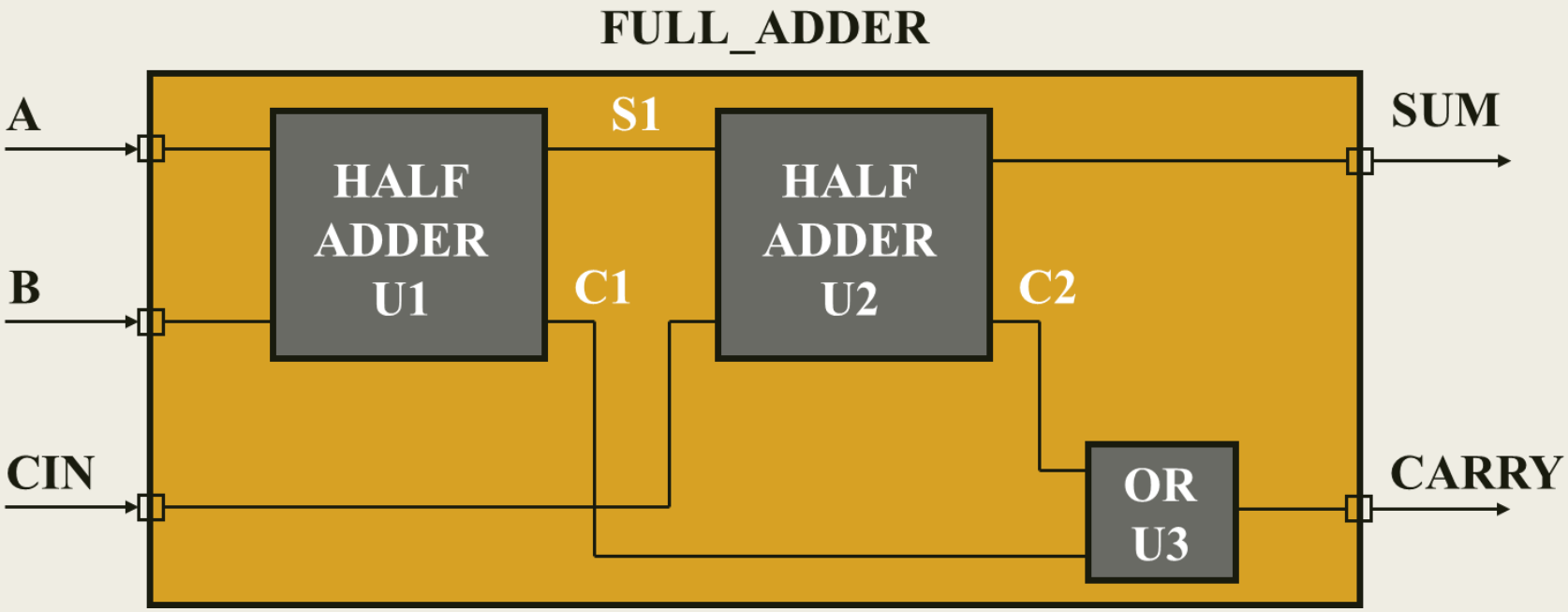


- Σχηματικό διάγραμμα σε τεχνολογία FPGA



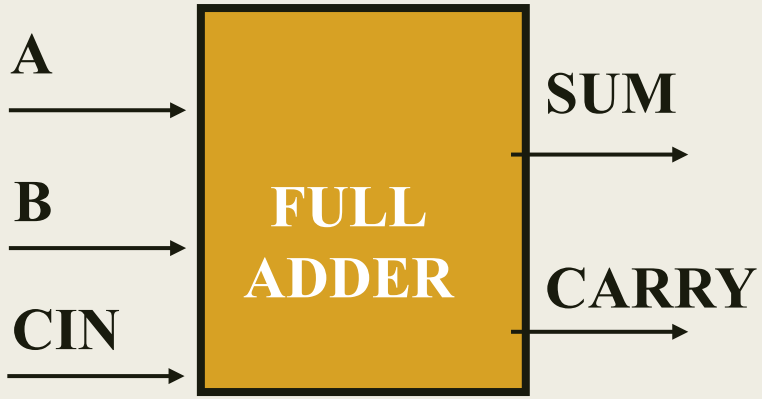
Δημιουργία ιεραρχικής δομής στη VHDL
Περιγραφή δομής

- Παράδειγμα: Πλήρης αθροιστής που αποτελείται από δύο ημιαθροιστές και μία πύλη OR

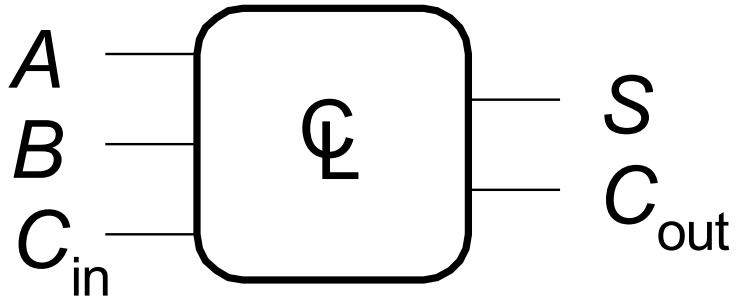


Η οντότητα του πλήρους αθροιστή στη VHDL

```
entity FULL_ADDER is
    port (
       A: in STD_LOGIC;
       B: in STD_LOGIC;
       CIN: in STD_LOGIC;
       SUM: out STD_LOGIC;
       CARRY: out STD_LOGIC);
end FULL_ADDER;
```



C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AB + AC_{in} + BC_{in}$$

Η αρχιτεκτονική του πλήρους αθροιστή στη VHDL
Περιγραφή dataflow

- Με εσωτερικά σήματα (δηλώνονται στο signal)

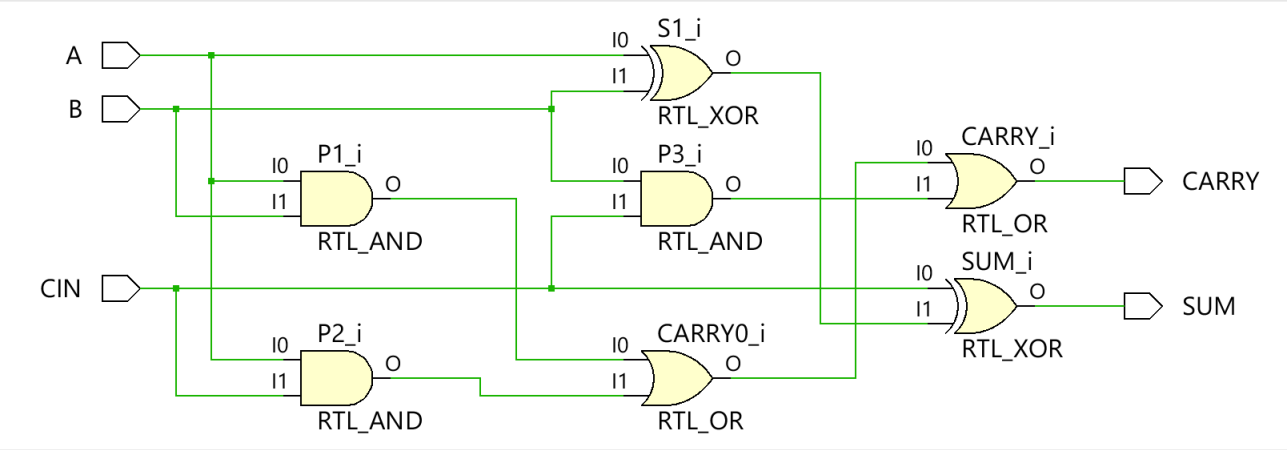
```
architecture FA_DATAFLOW1 of FULL_ADDER is
    signal S1, P1, P2, P3: STD_LOGIC;
begin
    S1 <= A xor B;
    SUM <= CIN xor S1;
    P1 <= A and B;
    P2 <= A and CIN;
    P3 <= B and CIN;
    CARRY <= P1 or P2 or P3;
end FA_DATAFLOW1;
```

- Χωρίς εσωτερικά σήματα

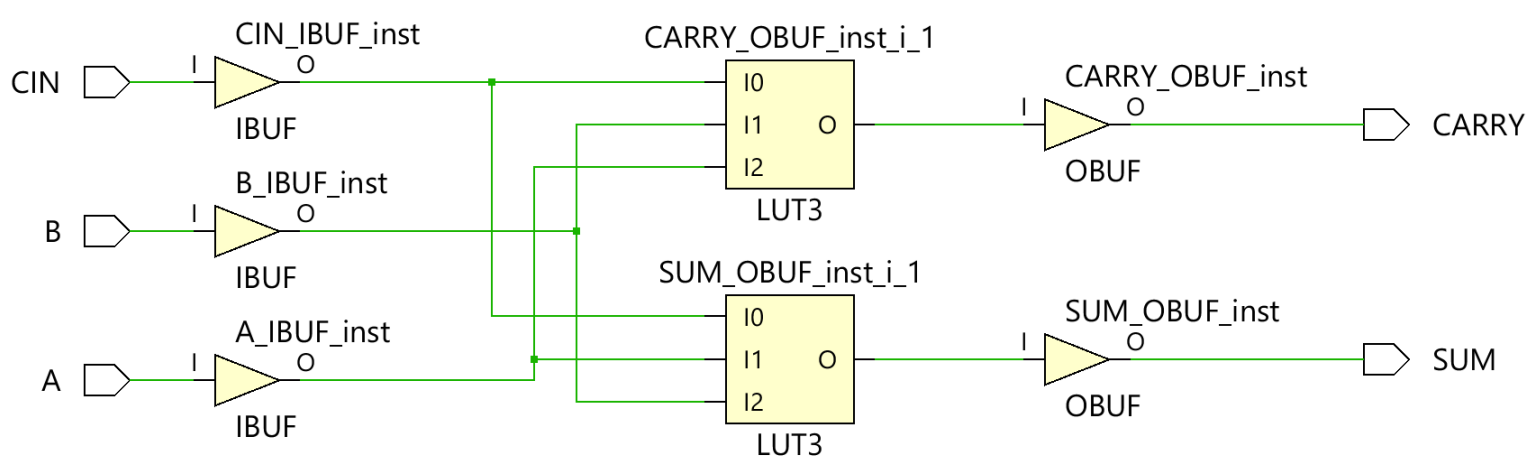
```
architecture FA_DATAFLOW2 of FULL_ADDER is
begin
    SUM <= A xor B xor CIN;
    CARRY <= (A and B) or (A and CIN) or (B and CIN);
end FA_DATAFLOW2;
```

Η αρχιτεκτονική του πλήρους αθροιστή στη VHDL
Περιγραφή dataflow

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



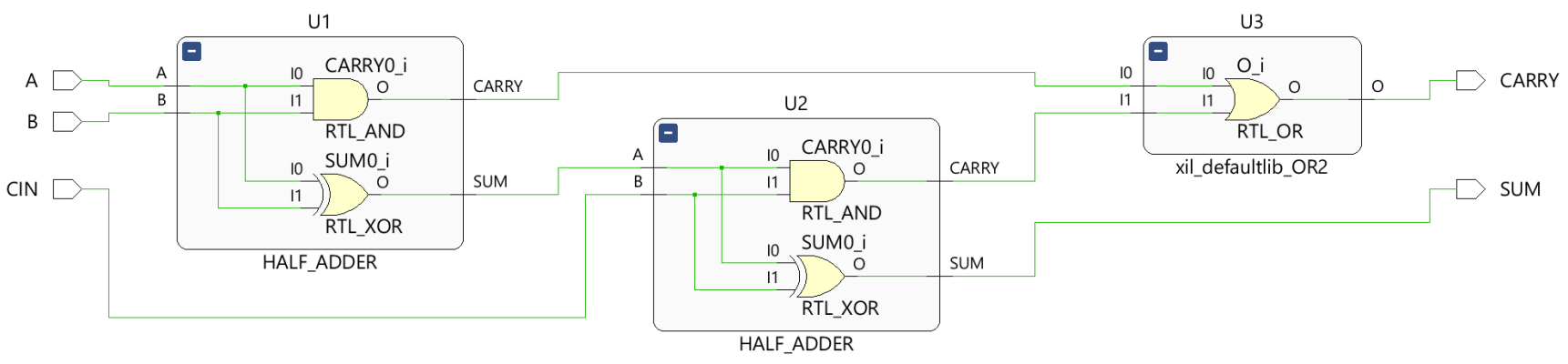
Η αρχιτεκτονική του πλήρους αθροιστή στη VHDL
Περιγραφή δομής

```
architecture FA_STRUCTURAL of FULL_ADDER is
  signal S1, C1, C2: STD_LOGIC;
  component HALF_ADDER
    port (A : in STD_LOGIC; B : in STD_LOGIC;
          SUM : out STD_LOGIC; CARRY : out STD_LOGIC);
  end component;
  component OR2
    port (O : out STD_LOGIC; I0 : in STD_LOGIC; I1 : in STD_LOGIC);
    end component;
begin
    U1: HALF_ADDER port map (A => A, B => B, SUM => S1, CARRY => C1);
    U2: HALF_ADDER port map (A => S1, B => CIN, SUM => SUM, CARRY => C2);
    U3: OR2 port map (O => CARRY, I0 => C1, I1 => C2);
end FA_STRUCTURAL;
```

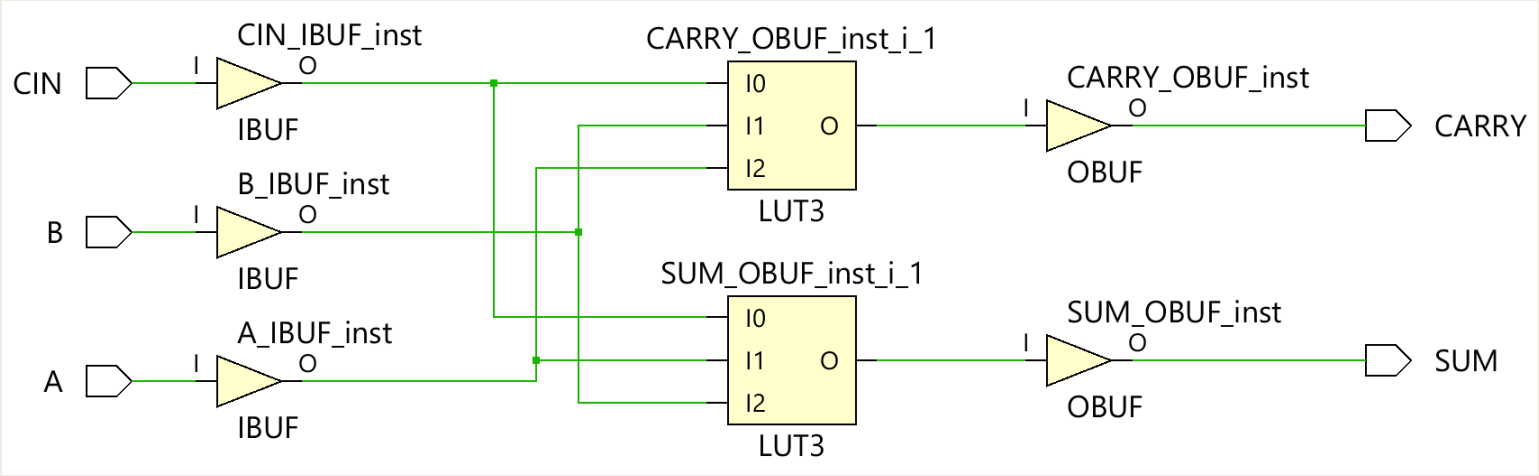
Τα στοιχεία HALF_ADDER και OR2 έχουν ήδη προκαθοριστεί
σαν οντότητες (προσέγγιση σχεδίασης **bottom-up**) του project

Η αρχιτεκτονική του πλήρους αθροιστή στη VHDL
Περιγραφή δομής

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA

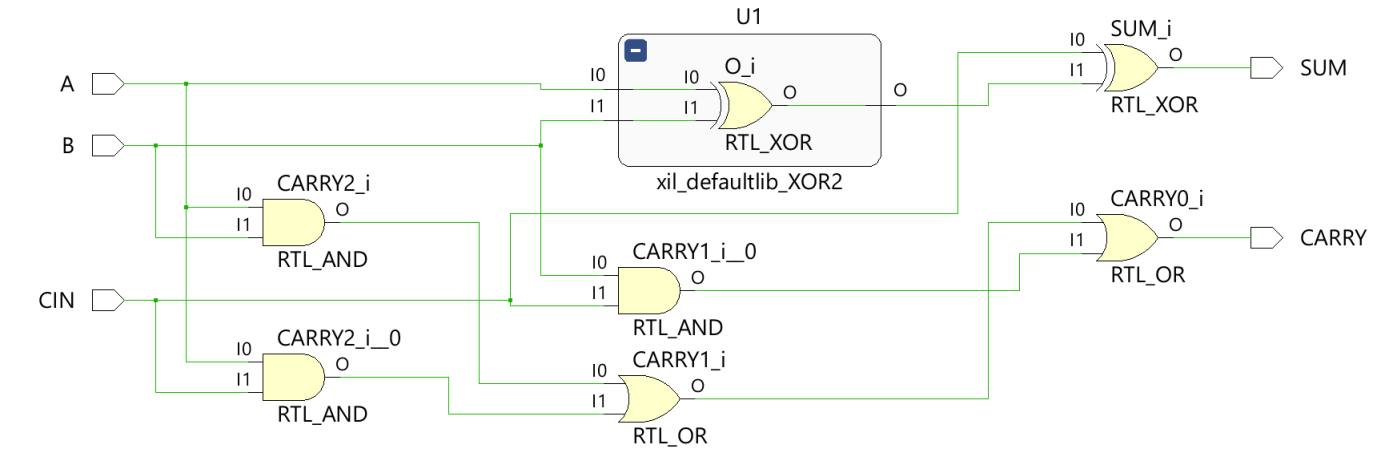


Η αρχιτεκτονική του πλήρους αθροιστή στη VHDL
Μικτή περιγραφή (δομής, dataflow, συμπεριφοράς)

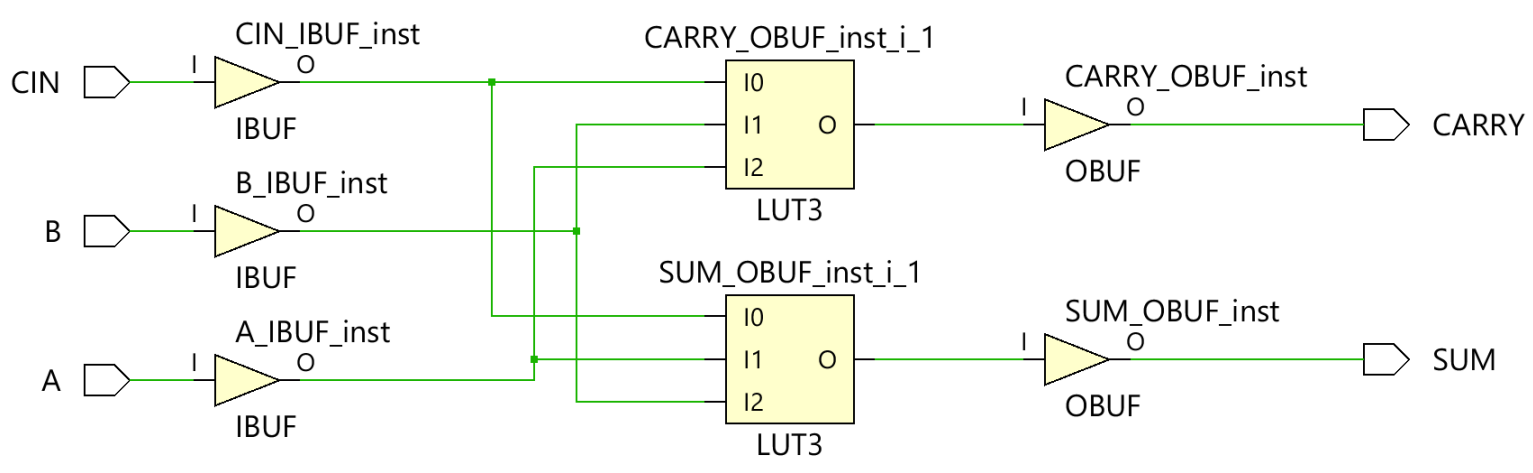
```
architecture FA_MIXED of FULL_ADDER is
    signal S1: STD_LOGIC;
    component XOR2
        port (O : out STD_LOGIC;
              I0 : in STD_LOGIC; I1 : in STD_LOGIC);
    end component;
begin
    U1: XOR2 port map (O => S1, I0 => A, I1 => B);
    SUM <= CIN xor S1;
    process (A, B, CIN)
        variable V1, V2, V3: STD_LOGIC;
    begin
        V1 := A and B;
        V2 := A and CIN;
        V3 := B and CIN;
        CARRY <= V1 or V2 or V3;
    end process;
end FA_MIXED;
```

Η αρχιτεκτονική του πλήρους αθροιστή στη VHDL
Μικτή περιγραφή (δομής, dataflow, συμπεριφοράς)

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Εντολές με συνθήκη

```
architecture arch_name of entity_name is
begin
  process (signal_name, ..., signal_name)
    variable variable_name: variable_type;
  begin
    sequential_signal_assignment_statement;
    sequential_variable_assignment_statement;
    conditional_signal_assignment_statement;
    conditional_variable_assignment_statement;
  end process;
end arch_name;
```

Με την περιγραφή συμπεριφοράς προσδιορίζουμε τη λειτουργικότητα και όχι τη δομή της υπομονάδας.
Αν και οι εντολές μέσα σε μία διεργασία (process) αξιολογούνται ακολουθιακά, μετά τη σύνθεση προκύπτει ταυτόχρονη λογική

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Η εντολή IF

- Από τις πιο σημαντικές ακολουθιακές εντολές ανάθεσης σήματος (ή μεταβλητής) **με συνθήκη** που χρησιμοποιούνται μέσα σε **process**
  - *Η εντολή IF εξετάζει μία συνθήκη και, εάν αληθεύει, εκτελεί την ακολουθιακή εντολή 1, αλλιώς, εκτελεί την ακολουθιακή εντολή 2 (εάν ορίζεται)*
- Δομή εντολής IF

```
if boolean_expression (condition) then
    sequential_statement_1;
end if;
```

```
if boolean_expression (condition) then
    sequential_statement_1;
else
    sequential_statement_2;
end if;
```

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Η εντολή IF

- Η εντολή IF επιτρέπει να εξετασθεί μία **διατεταγμένη σειρά από συνθήκες** με τη χρήση της φράσης **elsif**
  - *Εκτελείται μόνο η ακολουθιακή εντολή για την οποία αληθεύει η συνθήκη (Στη γενική περίπτωση μπορούν να εκτελεστούν και 2 ή περισσότερες εντολές)*
  - *Η σειρά με την οποία γράφονται οι εντολές είναι σημαντική*
    - στην περίπτωση που περισσότερες από μία συνθήκες **αληθεύουν ταυτόχρονα**, θα εκτελεσθεί εκείνη η εντολή για την οποία η συνθήκη αληθεύει **πρώτη**
- Δομή εντολής IF

```
if boolean_expression_1 (condition_1) then
    sequential_statement_1;
elsif boolean_expression_2 (condition_2) then
    sequential_statement_2;
else
    sequential_statement_3;
end if;
```

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Η εντολή IF

- Boolean Expression
  - *υλοποιεί μία συνθήκη*

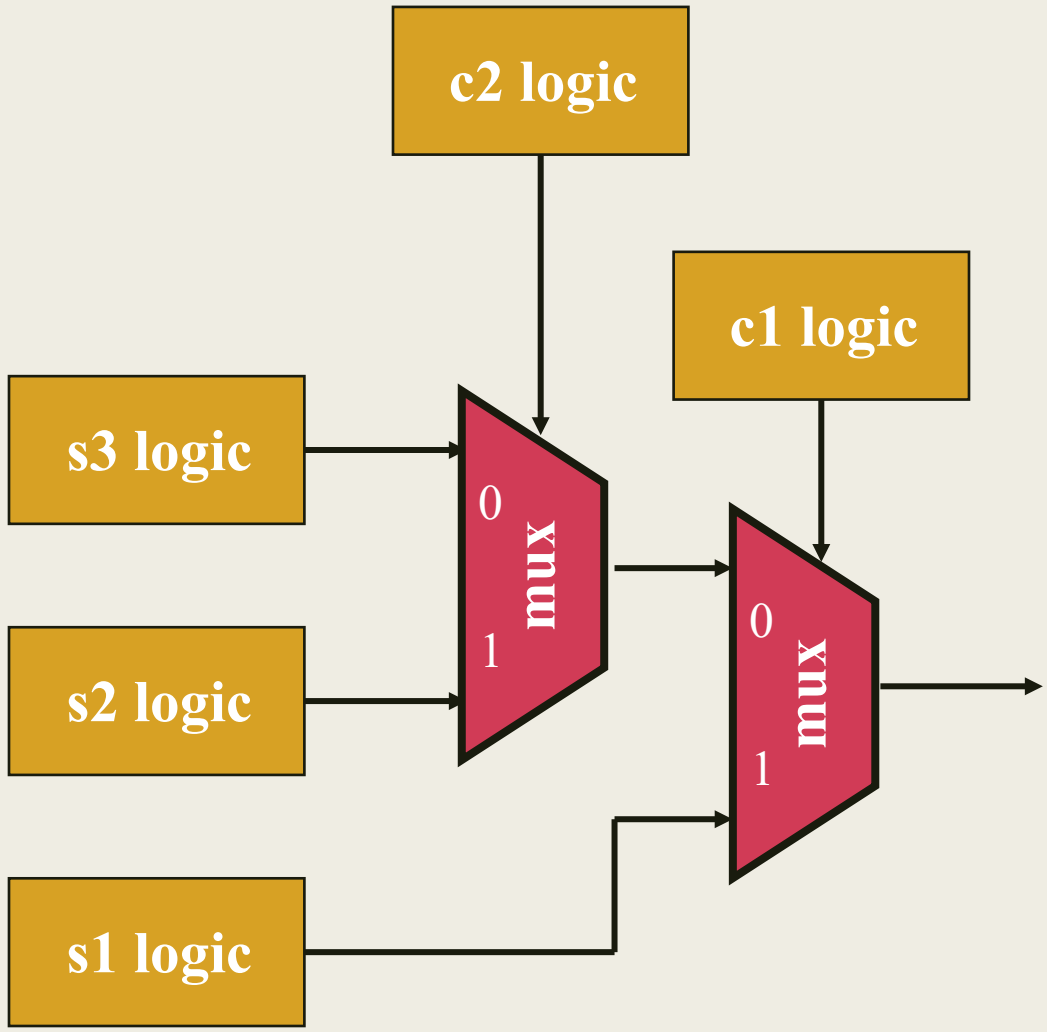
    (A = B)
  - *ή μια σειρά από συνθήκες που συνδέονται μεταξύ τους με λογικούς τελεστές (**and**, **or**, **not**, **nand**, **nor**, **xor**, **xnor**)*

    (A = "000" or RESET = '1')
- Συνθήκη (condition)
  - *επιστρέφει μία boolean τιμή (TRUE, FALSE)*
  - *περιγράφεται με δύο τελεστέους του **ιδίου τύπου**, που συγκρίνονται με τη χρήση ενός τελεστή σύγκρισης (<, <=, >, >=, =, /=)*

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Η εντολή IF

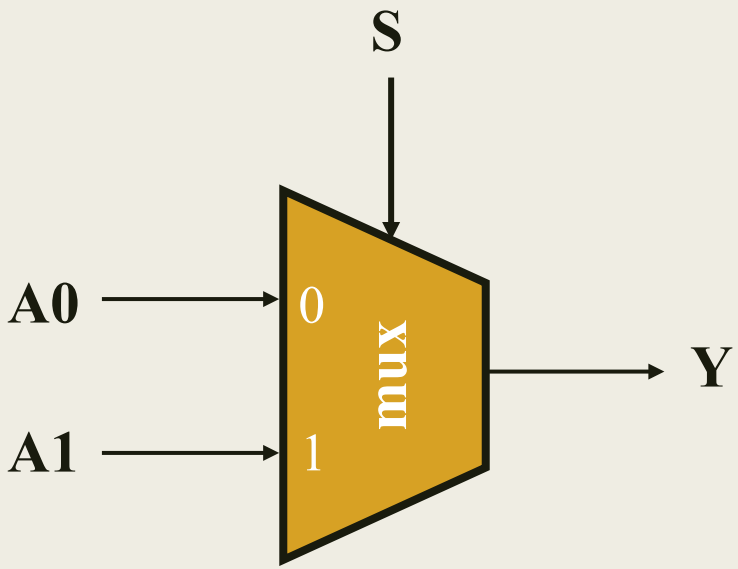
- Υλοποίηση εντολής IF με τη χρήση **πολυπλεκτών 2 σε 1**

```
if c1 then
    s1;
elsif c2 then
    s2;
else
    s3;
end if;
```



Η οντότητα του πολυπλέκτη 2 σε 1 στη VHDL

```
entity MUX_2_to_1 is
    port (
       A0: in STD_LOGIC;
       A1: in STD_LOGIC;
       S: in STD_LOGIC;
       Y: out STD_LOGIC);
end MUX_2_to_1;
```



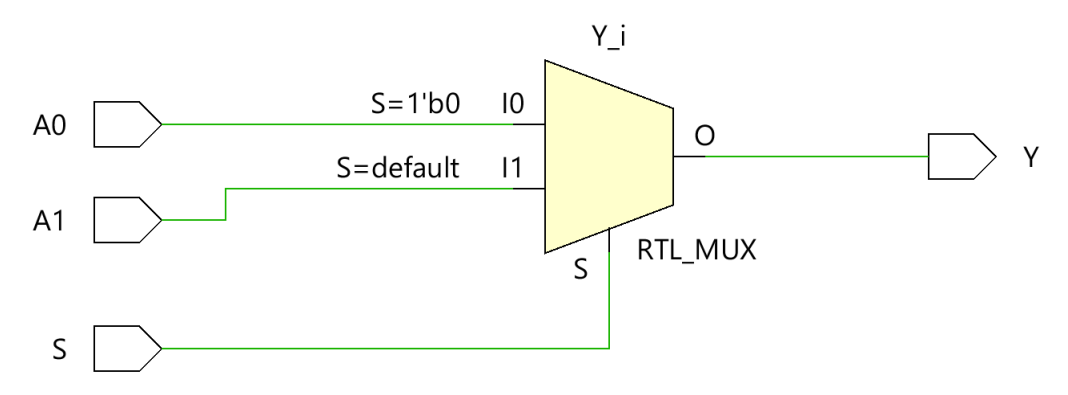
Η αρχιτεκτονική του πολυπλέκτη 2 σε 1 στη VHDL
Περιγραφή συμπεριφοράς

```
architecture BEHAVIORAL of MUX_2_to_1 is
begin
  process (A0, A1, S)
  begin
    if (S = '0') then
      Y <= A0;
    else
      Y <= A1;
    end if;
  end process;
end BEHAVIORAL;
```

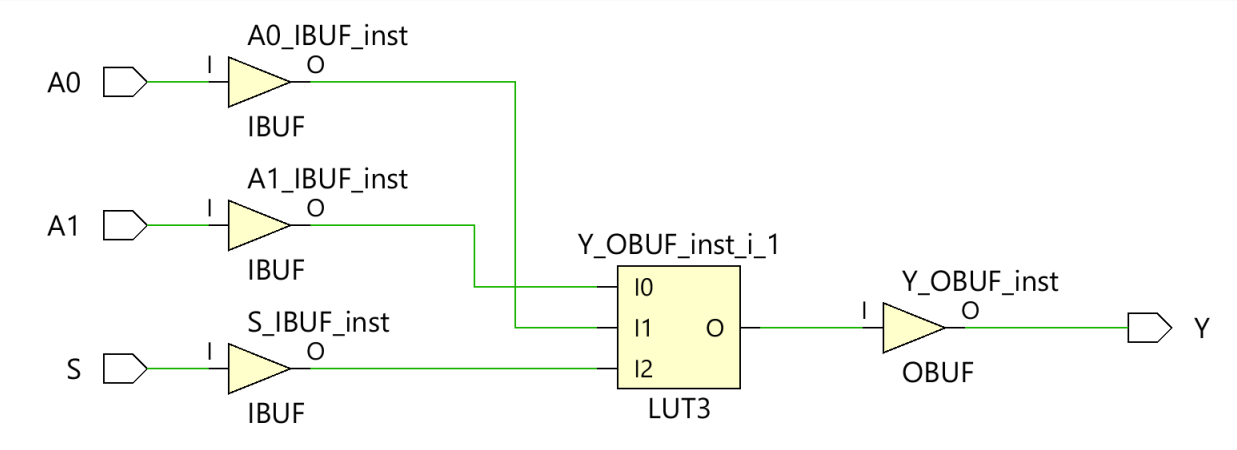
Στη λίστα ευαισθησίας συμπεριλαμβάνονται όλες
οι είσοδοι του συνδυαστικού κυκλώματος

Η αρχιτεκτονική του πολυπλέκτη 2 σε 1 στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL

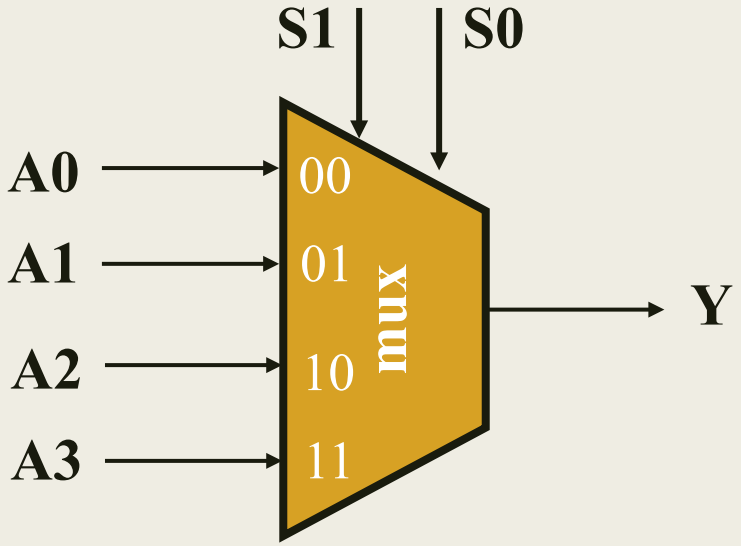


- Σχηματικό διάγραμμα σε τεχνολογία FPGA

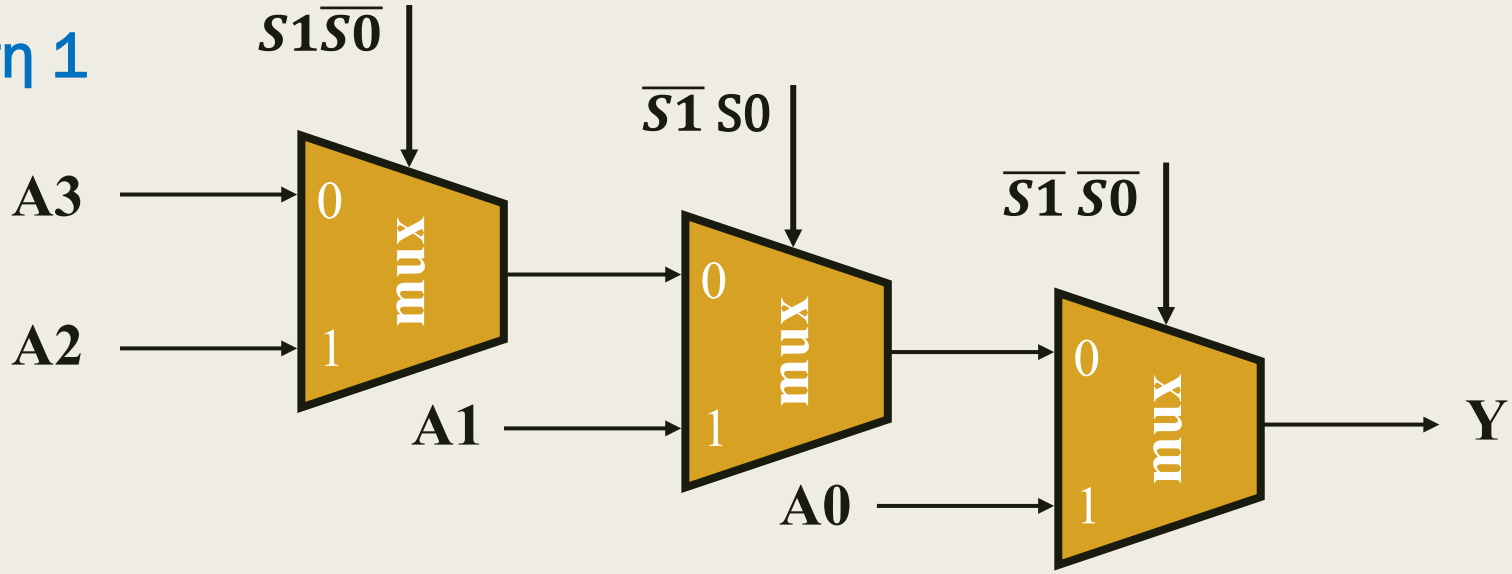


Η οντότητα του πολυπλέκτη 4 σε 1 στη VHDL

```
entity MUX_4_to_1 is
    port (
      A0: in STD_LOGIC;
      A1: in STD_LOGIC;
      A2: in STD_LOGIC;
      A3: in STD_LOGIC;
      S0: in STD_LOGIC;
      S1: in STD_LOGIC;
      Y: out STD_LOGIC);
end MUX_4_to_1;
```



Λύση 1



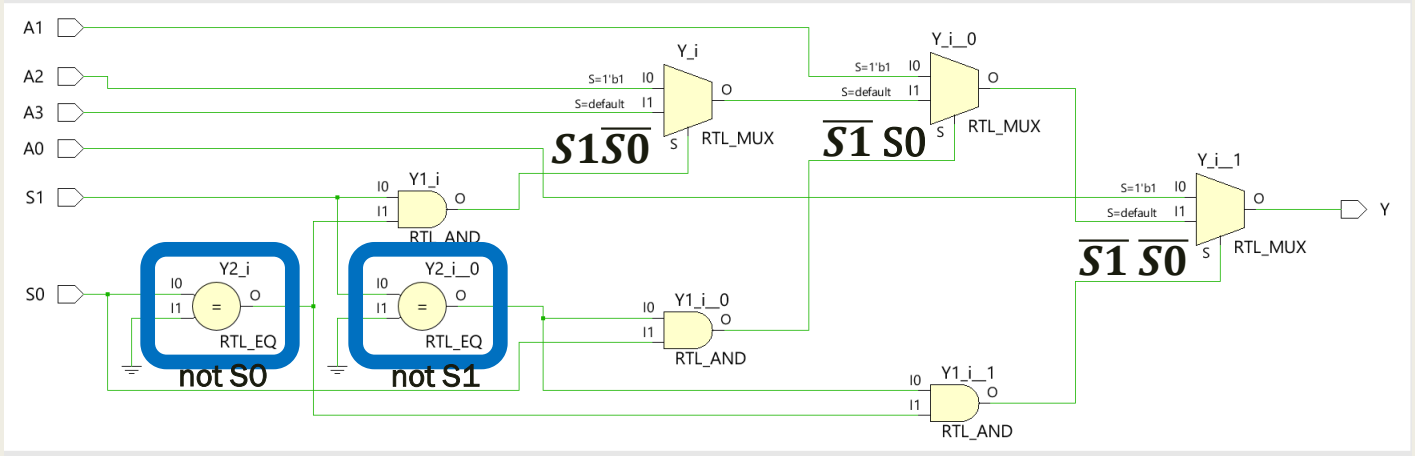
Η αρχιτεκτονική του πολυπλέκτη 4 σε 1 στη VHDL
Περιγραφή συμπεριφοράς – Λύση 1

```
architecture BEHAVIORAL of MUX_4_to_1 is
begin
  process (A0, A1, A2, A3, S0, S1)
  begin
    if    (S1 = '0' and S0 = '0') then Y <= A0;
    elsif (S1 = '0' and S0 = '1') then Y <= A1;
    elsif (S1 = '1' and S0 = '0') then Y <= A2;
    else                               Y <= A3;
    end if;
  end process;
end BEHAVIORAL;
```

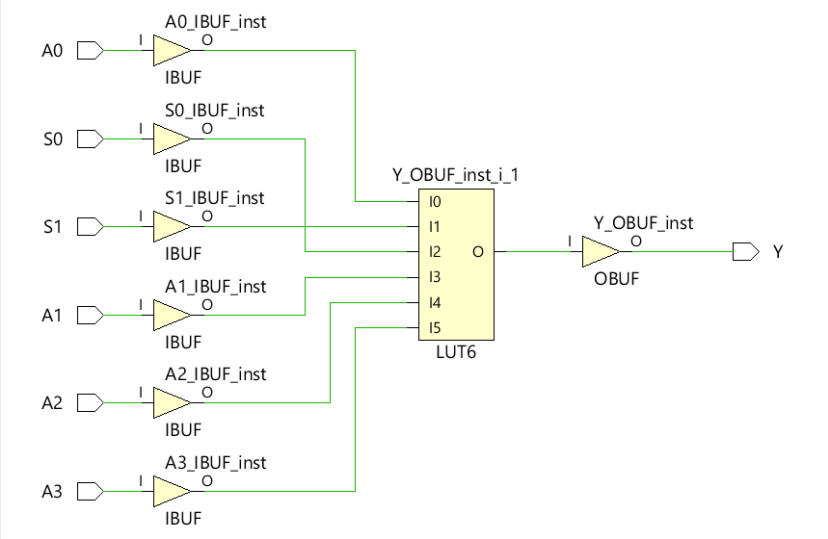
Στη λίστα ευαισθησίας συμπεριλαμβάνονται όλες
οι είσοδοι του συνδυαστικού κυκλώματος

Η αρχιτεκτονική του πολυπλέκτη 4 σε 1 στη VHDL
Περιγραφή συμπεριφοράς – Λύση 1

- Σχηματικό διάγραμμα RTL

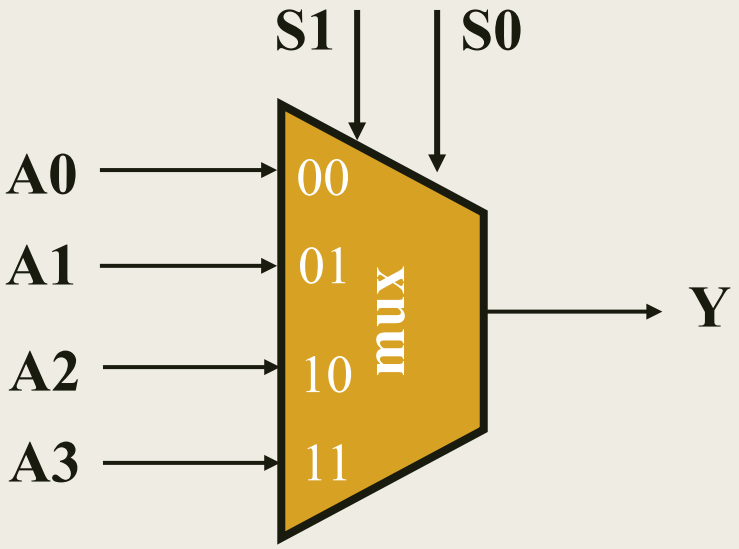


- Σχηματικό διάγραμμα σε τεχνολογία FPGA

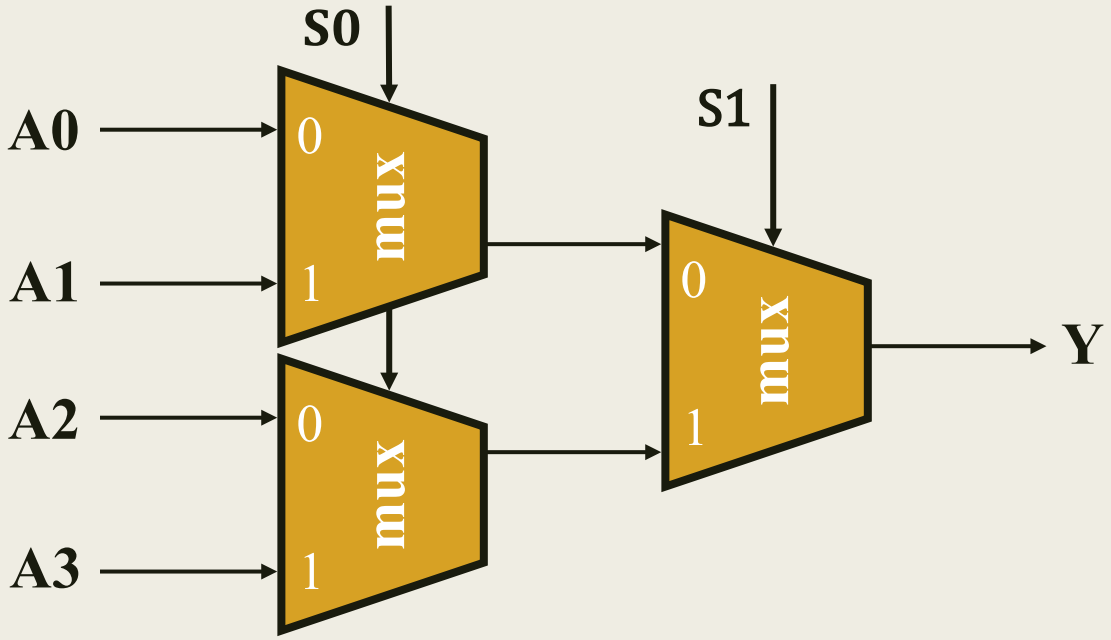


Η οντότητα του πολυπλέκτη 4 σε 1 στη VHDL

```
entity MUX_4_to_1 is
    port (
      A0: in STD_LOGIC;
      A1: in STD_LOGIC;
      A2: in STD_LOGIC;
      A3: in STD_LOGIC;
      S0: in STD_LOGIC;
      S1: in STD_LOGIC;
      Y: out STD_LOGIC);
end MUX_4_to_1;
```



Λύση 2



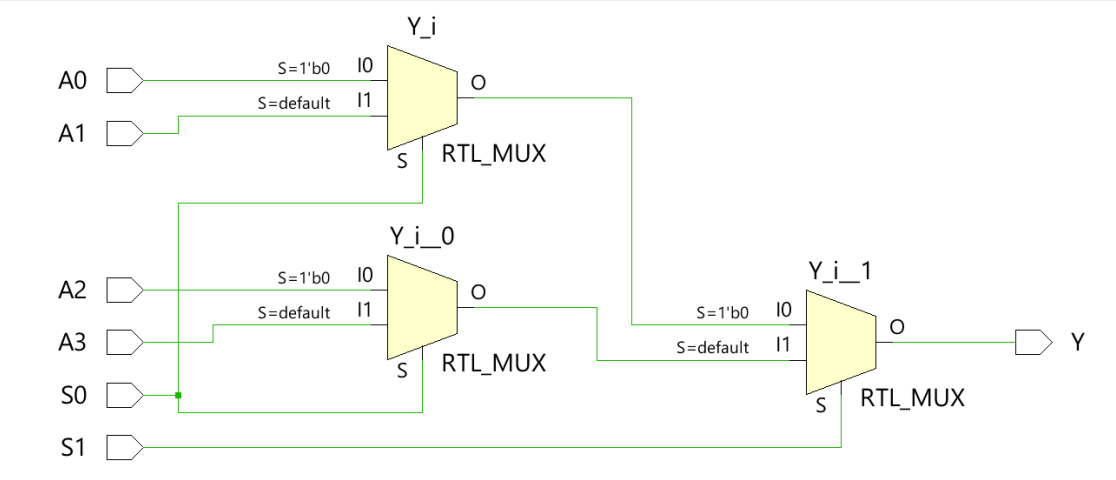
Η αρχιτεκτονική του πολυπλέκτη 4 σε 1 στη VHDL
Περιγραφή συμπεριφοράς – Λύση 2

```
architecture BEHAVIORAL of MUX_4_to_1 is
begin
  process (A0, A1, A2, A3, S0, S1)
  begin
    if (S1 = '0') then
      if (S0 = '0') then Y <= A0;
      else Y <= A1;
      end if;
    else
      if (S0 = '0') then Y <= A2;
      else Y <= A3;
      end if;
    end if;
  end process;
end BEHAVIORAL;
```

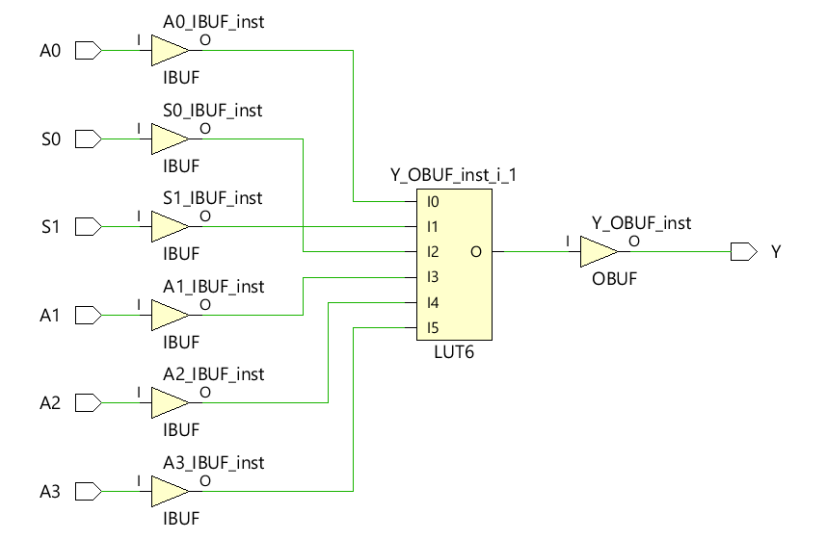
Στη λίστα ευαισθησίας συμπεριλαμβάνονται όλες οι είσοδοι του συνδυαστικού κυκλώματος

Η αρχιτεκτονική του πολυπλέκτη 4 σε 1 στη VHDL
Περιγραφή συμπεριφοράς – Λύση 2

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Και οι 2 λύσεις έχουν την ίδια υλοποίηση σε τεχνολογία FPGA με LUT6

Επιλεγμένη άσκηση: συνδυαστική λογική σε περιγραφή συμπεριφοράς στη VHDL

- Ποιος είναι ο **πίνακας αλήθειας** και η **εξίσωση Boole** του συνδυαστικού κυκλώματος, του οποίου η συμπεριφορά περιγράφεται στη VHDL ως εξης;

```
entity Exercise is port (
  A,B,C,D: in STD_LOGIC;
  Y: out STD_LOGIC);
end Exercise;
architecture BEHAVIORAL of Exercise is
begin
  process (A,B,C,D) begin
    if ((A = '0') and (B = '0')) then Y <= '1';
    elsif (C = D) then Y <= '1';
    else Y <= '0';
    end if;
  end process;
end BEHAVIORAL;
```

Επιλεγμένη άσκηση: συνδυαστική λογική σε περιγραφή συμπεριφοράς στη VHDL

- Πίνακας αλήθειας

```
if ((A = '0') and (B = '0')) then Y <= '1';
```

ABCD	Y
0000	1
0001	1
0010	1
0011	1
0100	
0101	
0110	
0111	

ABCD	Y
1000	
1001	
1010	
1011	
1100	
1101	
1110	
1111	

ABCD	Y
0000	1
0001	
0010	
0011	1
0100	1
0101	
0110	
0111	1

ABCD	Y
1000	1
1001	
1010	
1011	1
1100	1
1101	
1110	
1111	1

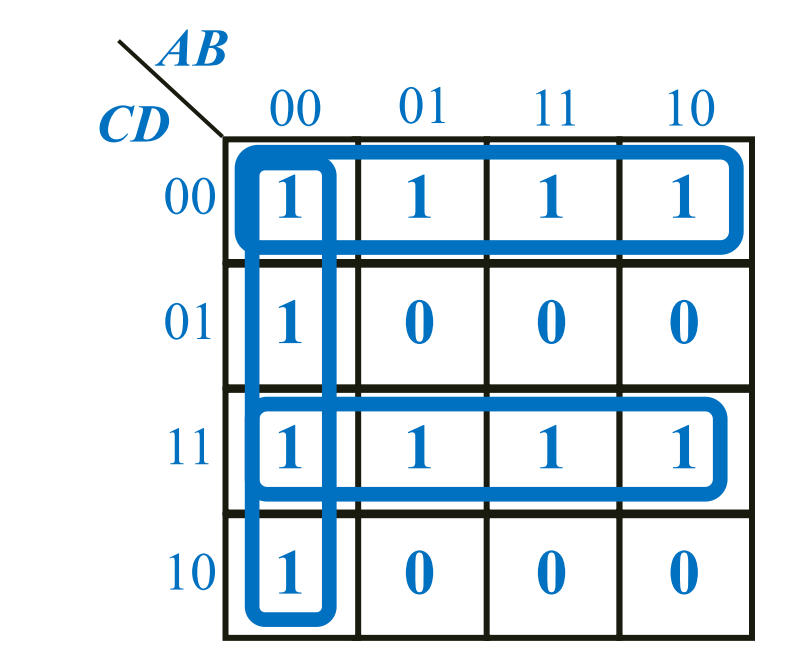
```
elsif (C = D) then Y <= '1';
```

Επιλεγμένη άσκηση: συνδυαστική λογική σε περιγραφή συμπεριφοράς στη VHDL

- Πίνακας αλήθειας

A B C D	Y
0 0 0 0	1
0 0 0 1	1
0 0 1 0	1
0 0 1 1	1
0 1 0 0	1
0 1 0 1	0
0 1 1 0	0
0 1 1 1	1

A B C D	Y
1 0 0 0	1
1 0 0 1	0
1 0 1 0	0
1 0 1 1	1
1 1 0 0	1
1 1 0 1	0
1 1 1 0	0
1 1 1 1	1



- Εξίσωση Boole

$$Y = \overline{A}\,\overline{B} + \overline{C}\,\overline{D} + CD$$

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση σημάτων

- Η ενημέρωση των σημάτων εξόδου (ή εσωτερικών σημάτων) με τη νέα τους τιμή γίνεται στο **τέλος του process** με **καθυστέρηση δέλτα δ_{delay}**
- Μέχρι το τέλος του process τα σήματα «θυμούνται» την τρέχουσα τιμή τους, δηλαδή τα **σήματα έχουν μνήμη μέσα στο process**
- Εάν μέσα σε ένα process γίνεται ανάθεση τιμών με πολλές εντολές στο ίδιο σήμα εξόδου (ή εσωτερικό σήμα), **μόνο η τελευταία εντολή ανάθεσης τιμής** λαμβάνεται υπόψη
- Εάν μέσα σε ένα process γίνεται ανάθεση τιμής σε ένα **εσωτερικό σήμα**, που στη συνέχεια χρησιμοποιείται και ως **είσοδος στο ίδιο process**, τότε αυτό το **εσωτερικό σήμα** πρέπει να δηλώνεται και στη **λίστα ευαισθησίας του process**
  - *για να συμφωνούν οι προσομοιώσεις πριν και μετά τη σύνθεση*

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση εσωτερικών σημάτων

```
entity XOR_single is port (
  A, B, C : in STD_LOGIC;
  X, Y : out STD_LOGIC);
end XOR_single;
architecture beh of XOR_sig is
signal D : STD_LOGIC;
begin
  process (A,B,C,D)
  begin
    D <= A; -- ignored
    X <= C xor D;
    D <= B; -- considered
    Y <= C xor D;
  end process;
end beh;
```

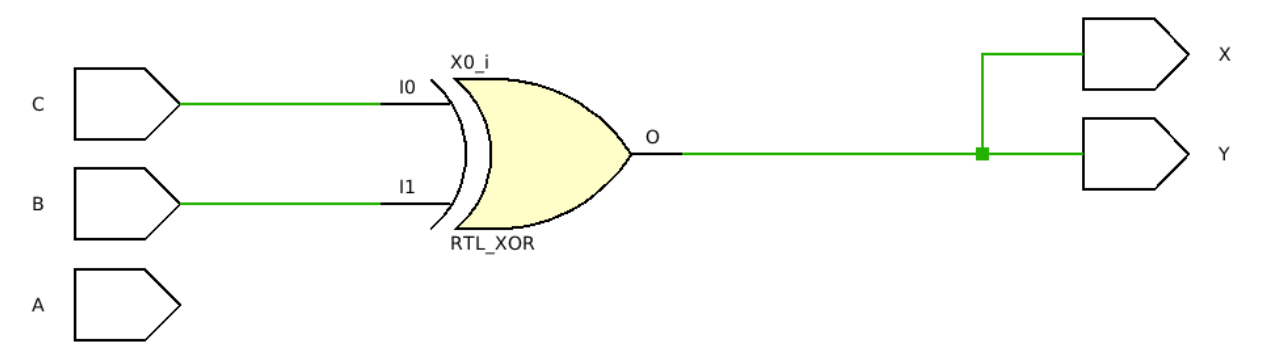
Ανάθεση τιμών στο εσωτερικό σήμα D με δύο εντολές

Το εσωτερικό σήμα D δηλώνεται στη λίστα ευαισθησίας

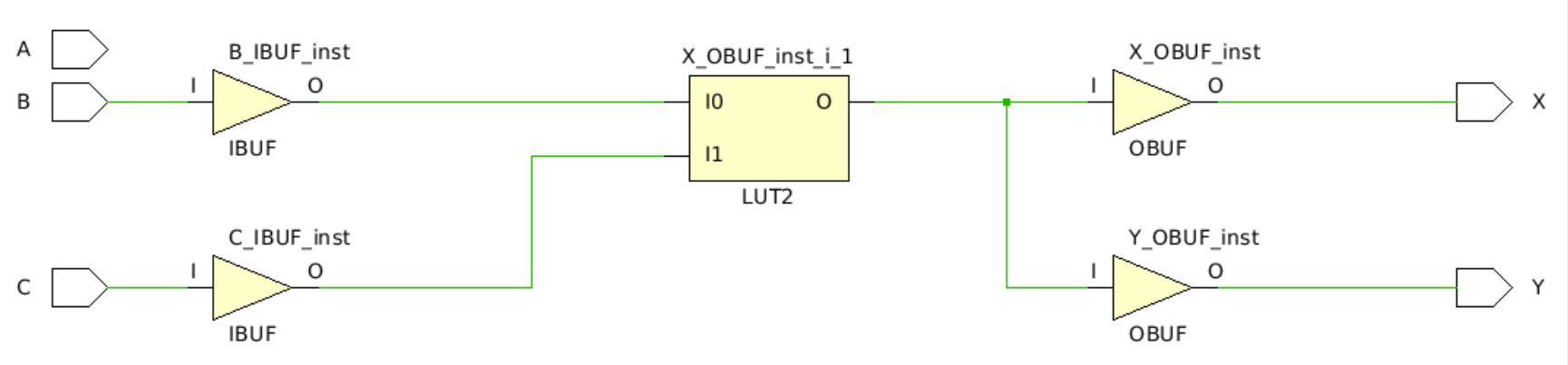
- Κατά τη σύνθεση υλοποιείται η εξίσωση Boole:
  - *X = Y = C xor B*

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση εσωτερικών σημάτων

- Σχηματικό διάγραμμα RTL

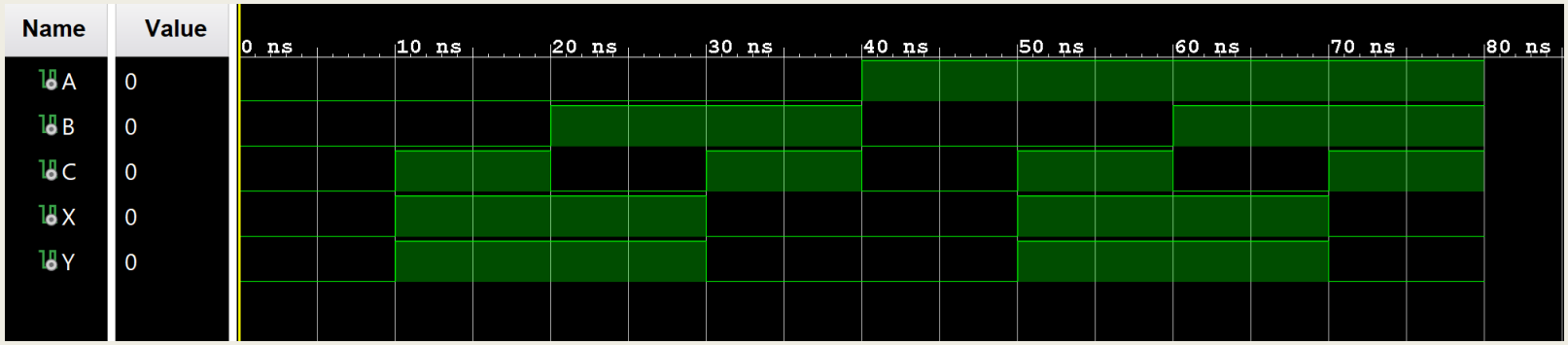


- Σχηματικό διάγραμμα σε τεχνολογία FPGA

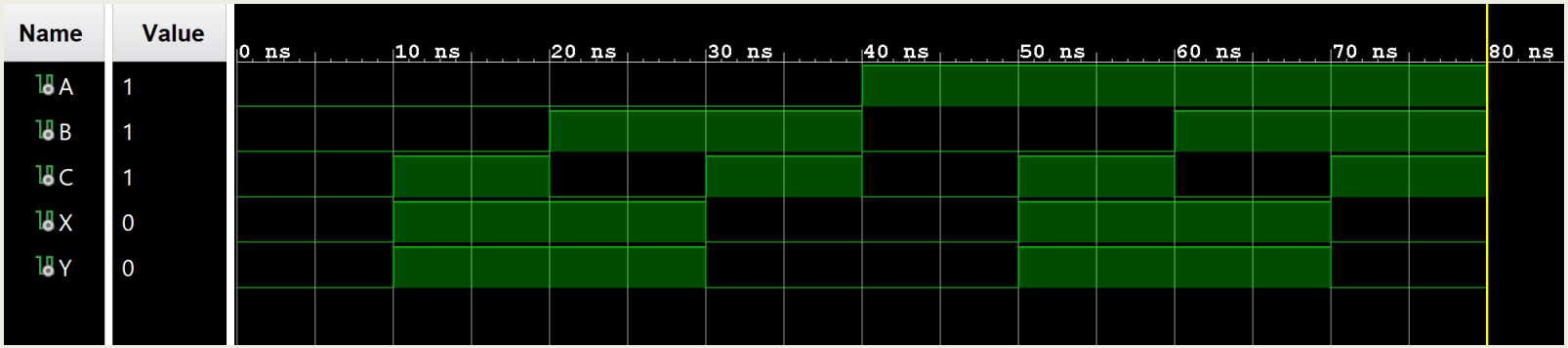


Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση εσωτερικών σημάτων

- Behavioral simulation (του VHDL κώδικα, πριν τη σύνθεση)



- Post-implementation functional simulation (μετά τη σύνθεση και την υλοποίηση)



Προσοχή: Συμφωνία στην προσομοίωση! Λόγω ορθής δήλωσης του εσωτερικού σήματος D στη λίστα ευαισθησίας

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση εσωτερικών σημάτων

```
entity XOR_single is port (
  A, B, C : in STD_LOGIC;
  X, Y : out STD_LOGIC);
end XOR_single;
architecture beh of XOR_sig is
signal D : STD_LOGIC;
begin
  process (A,B,C)
  begin
    D <= A; -- ignored
    X <= C xor D;
    D <= B; -- considered
    Y <= C xor D;
  end process;
end beh;
```

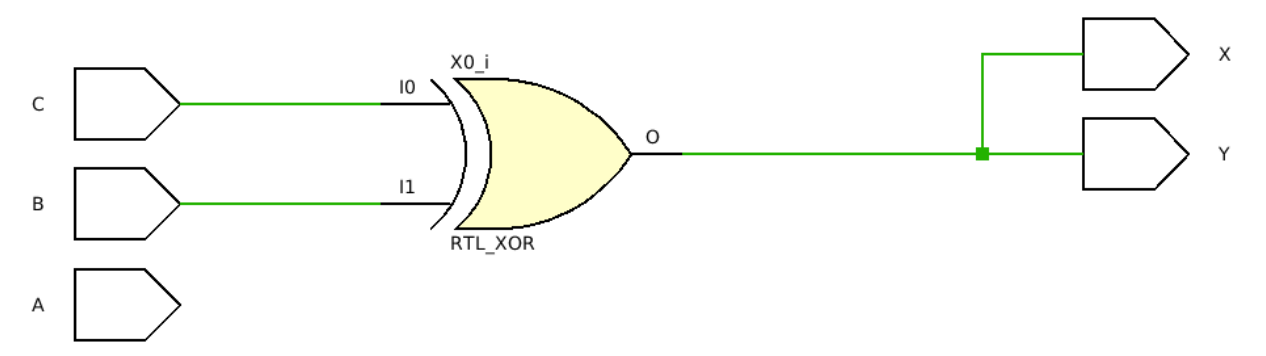
Ανάθεση τιμών στο εσωτερικό σήμα D με δύο εντολές

Το εσωτερικό σήμα D **δεν** δηλώνεται στη λίστα ευαισθησίας

- Κατά τη σύνθεση υλοποιείται η εξίσωση Boole:
  - *X = Y = C xor B*

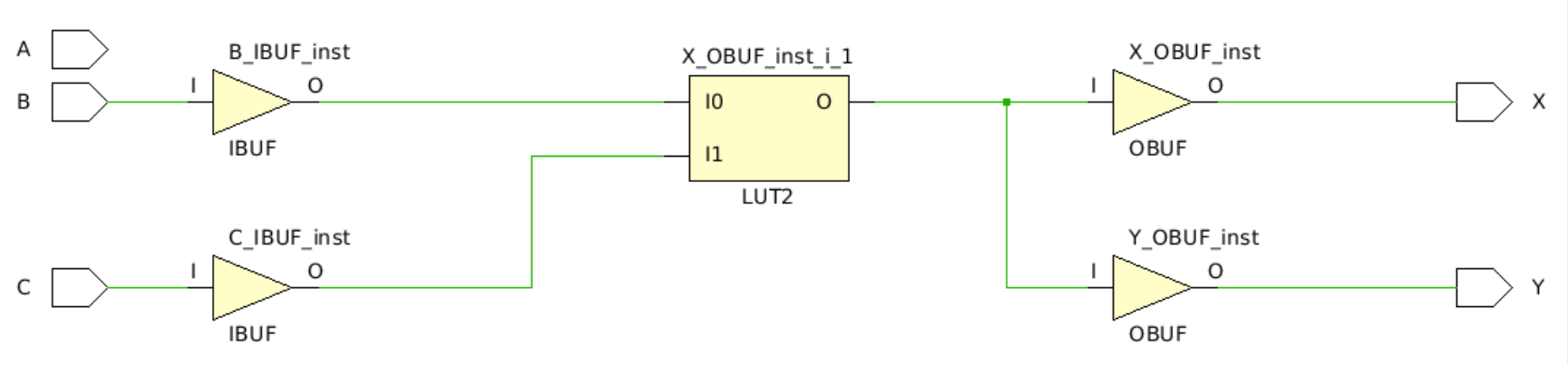
Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση εσωτερικών σημάτων

- Σχηματικό διάγραμμα RTL



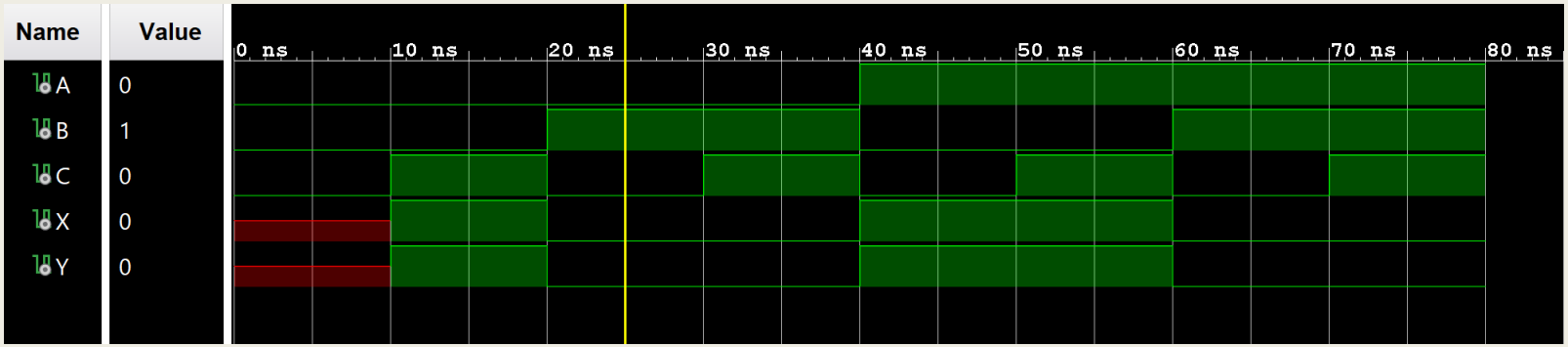
Η σύνθεση και η υλοποίηση γίνονται σωστά

- Σχηματικό διάγραμμα σε τεχνολογία FPGA

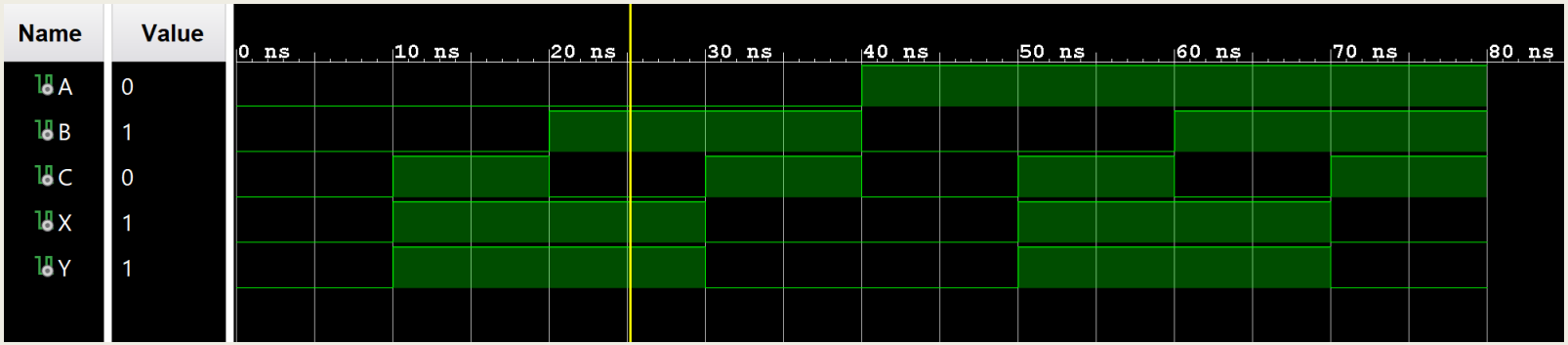


Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση εσωτερικών σημάτων

- Behavioral simulation (του VHDL κώδικα, πριν τη σύνθεση) – **Λάθος!**



- Post-implementation functional simulation (μετά τη σύνθεση και την υλοποίηση) – Σωστό



Προσοχή: Ασυμφωνία στην προσομοίωση! Λόγω **μη δήλωσης** του εσωτερικού σήματος D στη λίστα ευαισθησίας

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση μεταβλητών

- Η ενημέρωση των μεταβλητών γίνεται αμέσως μόλις εκτελεσθεί η αντίστοιχη εντολή ανάθεσης τιμής στη μεταβλητή εντός του process
- Η μεταβλητή διατηρεί την τιμή της μέχρι να προσδιορισθεί άλλη τιμή σε μία επόμενη εντολή ανάθεσης τιμής στην ίδια μεταβλητή εντός του process

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση μεταβλητών

```
entity XOR_var is port (
  A, B, C: in STD_LOGIC;
  X, Y: out STD_LOGIC);
end XOR_var;
architecture beh of XOR_var is
begin
  process (A,B,C)
  variable D: STD_LOGIC;
  begin
    D := A; -- considered
    X <= C xor D;
    D := B; -- considered
    Y <= C xor D;
  end process;
end beh;
```

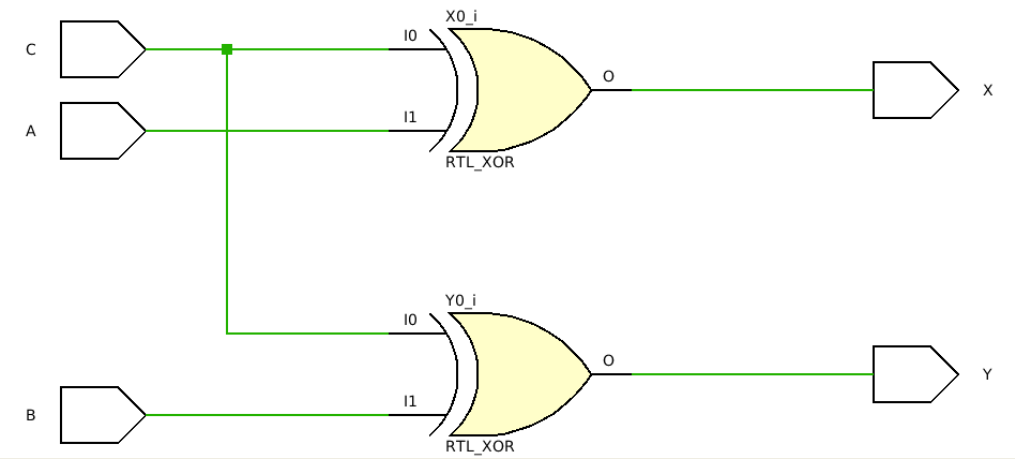
Η μεταβλητή D λαμβάνει άμεσα την τιμή της εισόδου A

Η μεταβλητή D λαμβάνει άμεσα την τιμή της εισόδου B

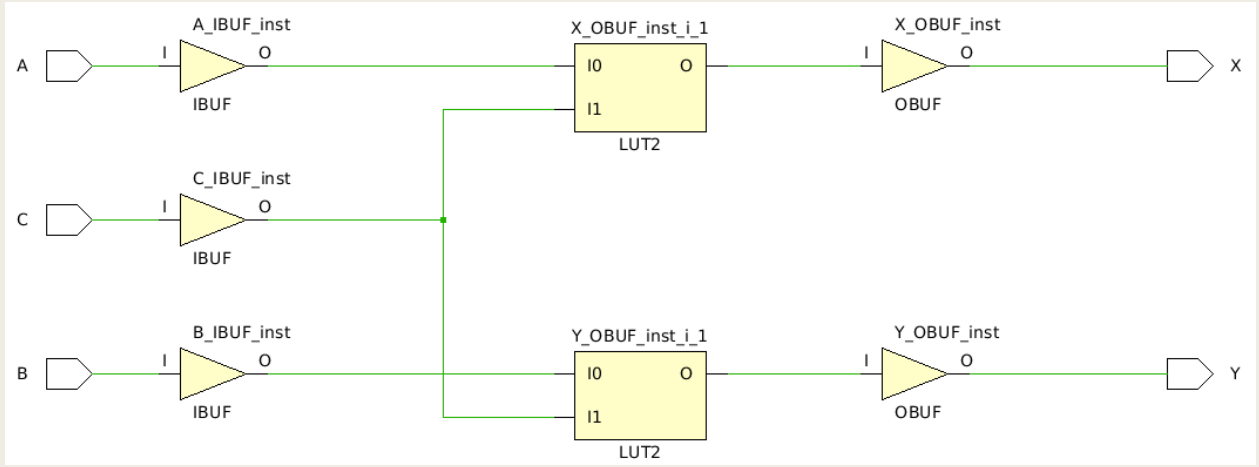
- Κατά τη σύνθεση υλοποιούνται οι εξισώσεις Boole:
  - *X = C xor A* και *Y = C xor B*

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση μεταβλητών

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση σημάτων

```
entity LAST is
  port (
    A, B, selA, selB : in STD_LOGIC;
    C : out STD_LOGIC);
end LAST;
architecture LAST_BEH of LAST is
begin
  process (A, B, selA, selB) begin

    if (selA = '1') then
      C <= A;
    else
      C <= '0';
    end if;

    if (selB = '1') then
      C <= B;
    else
      C <= '0';
    end if;

  end process;
end LAST_BEH;
```

Ανάθεση τιμών στο σήμα εξόδου C με δύο εντολές IF

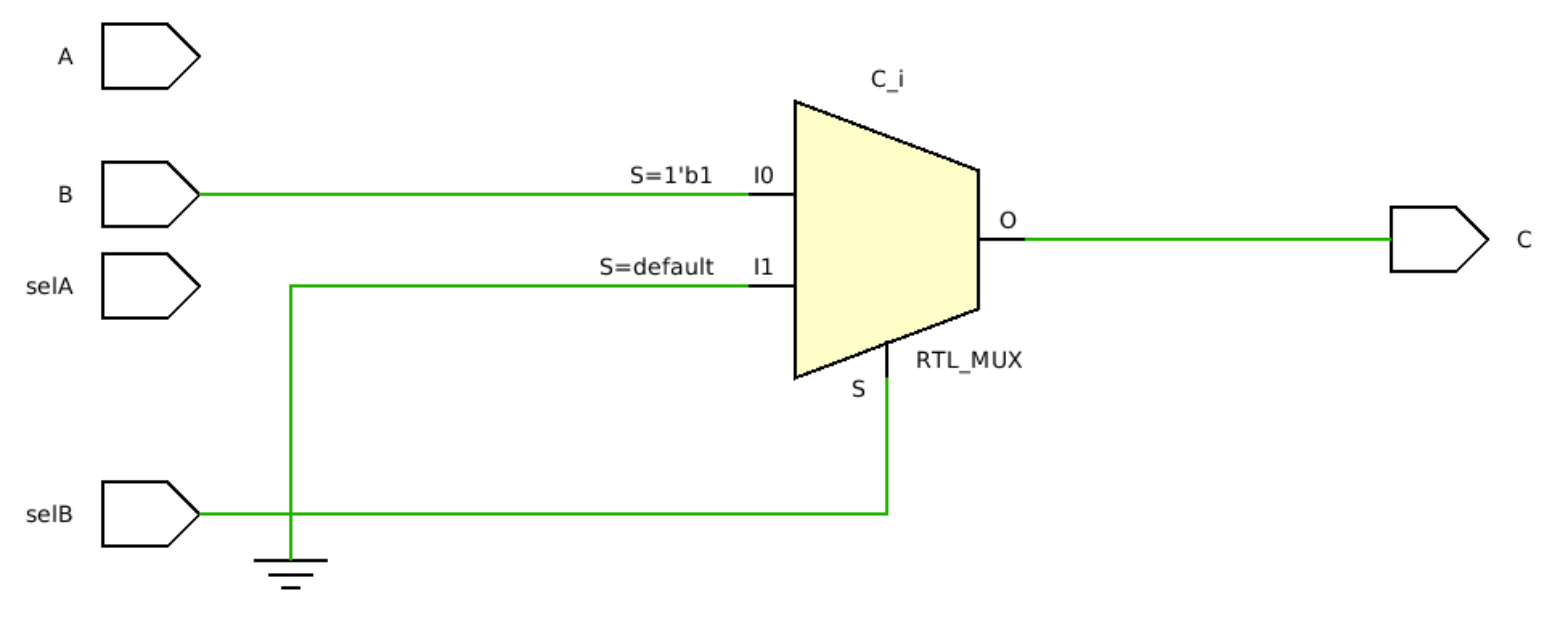
Δεν λαμβάνεται υπόψη

Λαμβάνεται υπόψη

- Ποιο είναι το αποτέλεσμα της σύνθεσης;

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση σημάτων

- Σχηματικό διάγραμμα RTL



- Εξίσωση Boole που υλοποιείται

$$C = B \text{ and } selB$$

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση σημάτων

- Τροποποιούμε τον κώδικα με τη χρήση του **elsif**

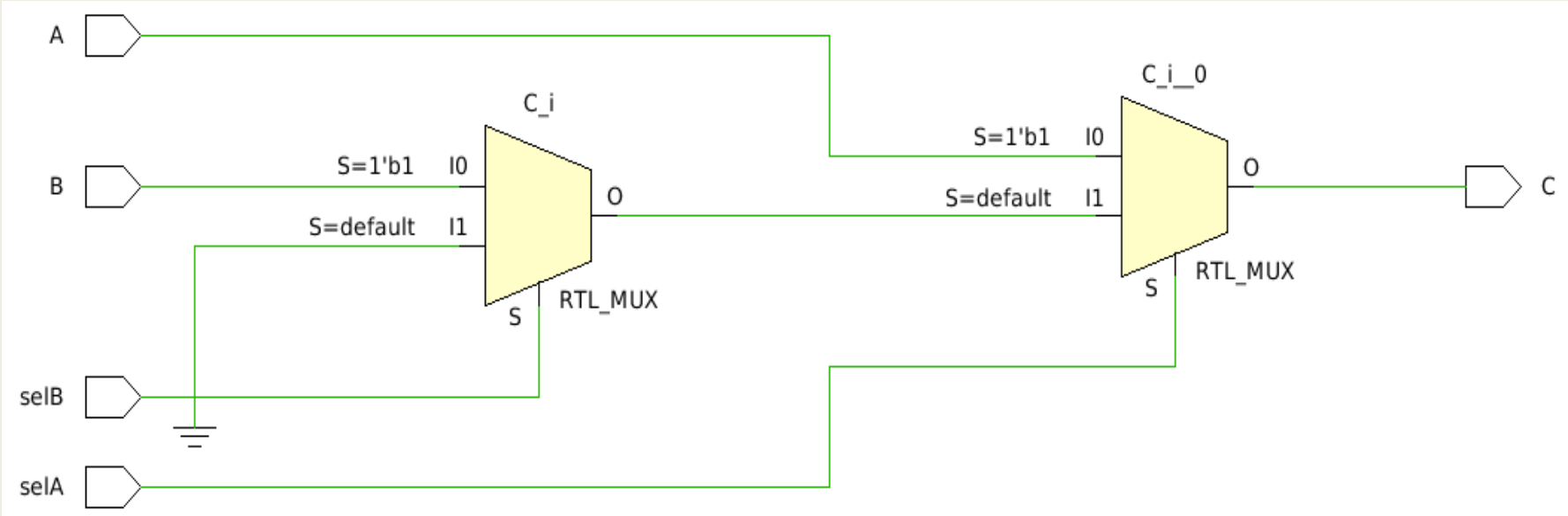
```
entity LAST is
  port (
    A, B, selA, selB: in STD_LOGIC;
    C: out STD_LOGIC);
end LAST;
architecture LAST_BEH of LAST is
begin
  process (A, B, selA, selB)
  begin
    if (selA = '1') then
      C <= A;
    elsif (selB = '1') then
      C <= B;
    else
      C <= '0';
    end if;
  end process;
end LAST_BEH;
```

Η χρήση του elsif επιτρέπει την ανάθεση τιμών στο σήμα εξόδου C με μία μόνο εντολή IF

- Ποιο είναι το αποτέλεσμα της σύνθεσης μετά την τροποποίηση;

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση σημάτων

- Σχηματικό διάγραμμα RTL



- Εξίσωση Boole που υλοποιείται

$$C = (B \text{ and } selB \text{ and } \overline{selA}) \text{ or } (A \text{ and } selA)$$

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Συμπεράσματα στη χρήση σημάτων και μεταβλητών

- **Προσοχή!** Να μην γίνεται ανάθεση τιμών με πολλές εντολές στο ίδιο σήμα εξόδου (ή εσωτερικό σήμα) μέσα σε ένα process
- **Προσοχή!** Στην ανάθεση τιμής εσωτερικού σήματος μέσα σε ένα process, όταν αυτό επαναχρησιμοποιείται στην έκφραση μίας εντολής ανάθεσης τιμής στο ίδιο process
  - *Το εσωτερικό σήμα πρέπει να συμπεριλαμβάνεται στη **λίστα ευαισθησίας**, ώστε το process να εκτελεσθεί ξανά για να διορθωθεί το λανθασμένο αποτέλεσμα του behavioral simulation*
- Προτιμούμε τη χρήση μεταβλητής, αντί για εσωτερικού σήματος, όταν αυτή επαναχρησιμοποιείται στην έκφραση μίας εντολής ανάθεσης τιμής στο ίδιο process
  - *Δεν απαιτείται η διόρθωση στη λίστα ευαισθησίας*
  - *Ο χρόνος προσομοίωσης μειώνεται σημαντικά γιατί το process εκτελείται μόνο μια φορά*

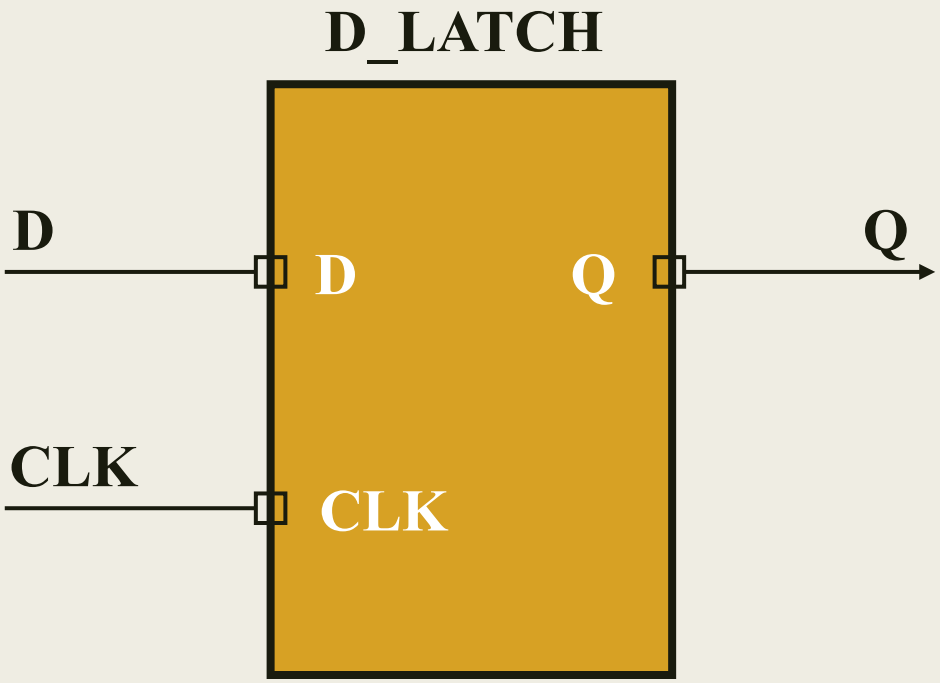
Περιγραφή ακολουθιακής λογικής στη VHDL

- Στην περιγραφή συμπεριφοράς οι ακολουθιακές εντολές ανάθεσης τιμής
  - *ενημερώνουν τα σήματα εξόδου (ή τα εσωτερικά σήματα) με τη νέα τους τιμή στο **τέλος του process** με **καθυστέρηση δέλτα** δ_{delay}*
  - *μέχρι το τέλος της διεργασίας τα σήματα «θυμούνται» την τρέχουσα τιμή τους, δηλαδή τα **σήματα έχουν μνήμη μέσα στο process***
- Η δυνατότητα των σημάτων να «θυμούνται» την τρέχουσα τιμή τους μας επιτρέπει να περιγράψουμε **ακολουθιακή λογική** με τη χρήση μίας **εντολής IF** στην οποία υπάρχει **ελλιπής ανάθεση τιμής** σε ένα σήμα εξόδου (ή εσωτερικό σήμα)
  - *στην εντολή IF δεν ορίζεται η τιμή του σήματος με εντολή ανάθεσης τιμής, όταν δεν ικανοποιείται η συνθήκη*

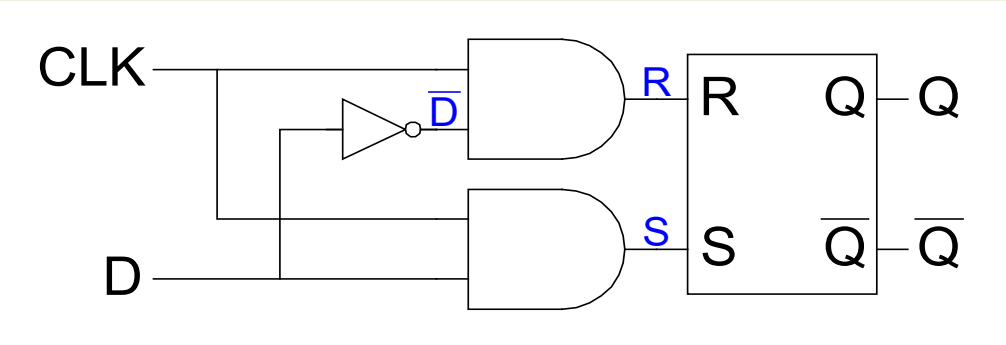
```
if boolean_expression (condition) then
    sequential_statement_1;
end if;
```

Η οντότητα του D Latch στη VHDL

```
entity D_LATCH is
  port (
    CLK, D: in STD_LOGIC;
    Q: out STD_LOGIC);
end D_LATCH;
```



Η αρχιτεκτονική του D Latch στη VHDL
Περιγραφή συμπεριφοράς



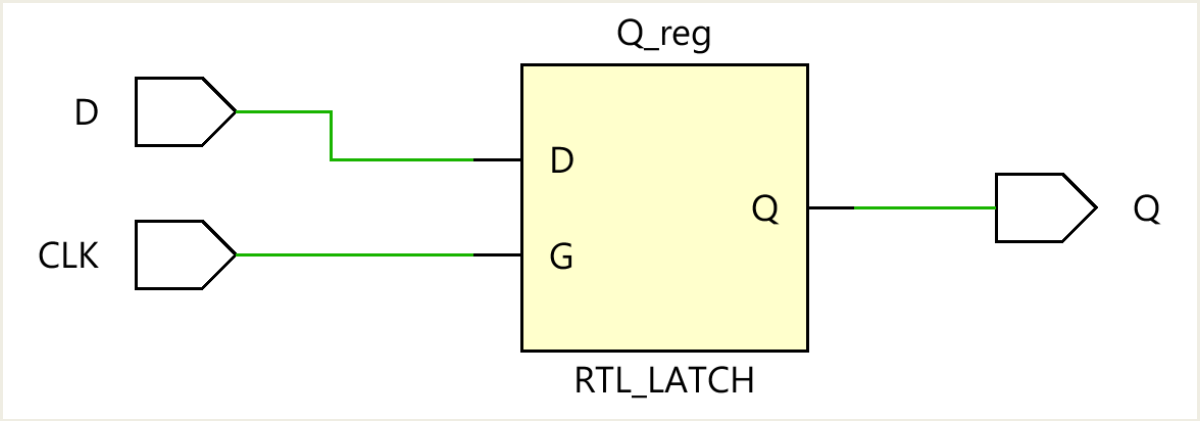
CLK	D	Q	$\overline{Q}$
0	X	Q_{prev}	$\overline{Q}_{prev}$
1	0	0	1
1	1	1	0

```
architecture BEHABIORAL of D_LATCH is
begin
  process (CLK, D)
  begin
    if (CLK = '1') then
      Q <= D;
    end if;          --Το Q δεν ορίζεται για όλες τις τιμές του CLK
  end process;       --Ελλιπής ανάθεση του Q (incomplete assignment)
end BEHABIORAL;
```

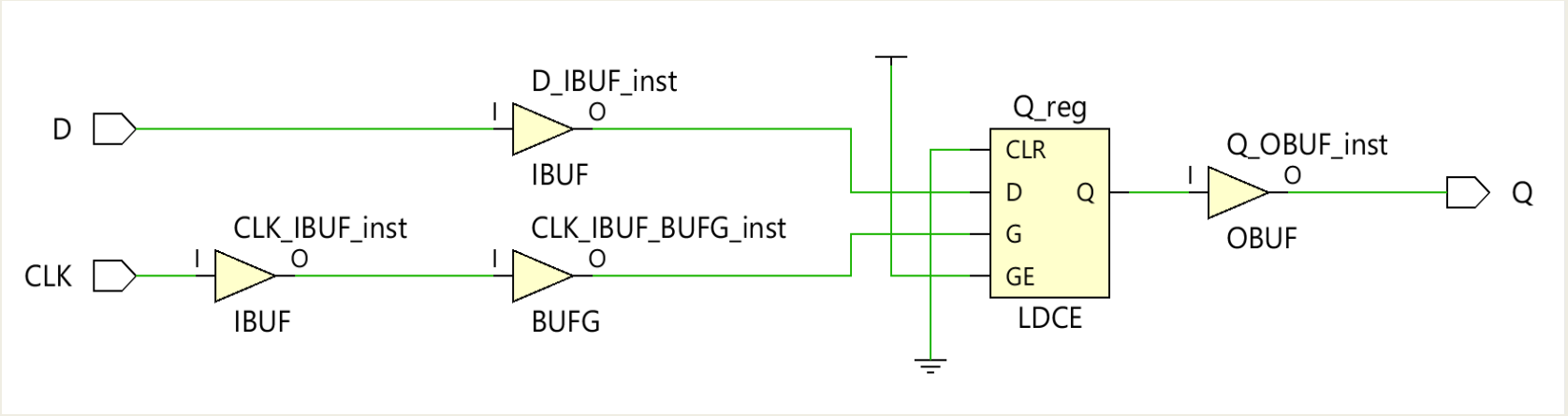
Εκμεταλλευόμαστε την εσωτερική μνήμη των σημάτων

Η αρχιτεκτονική του D Latch στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



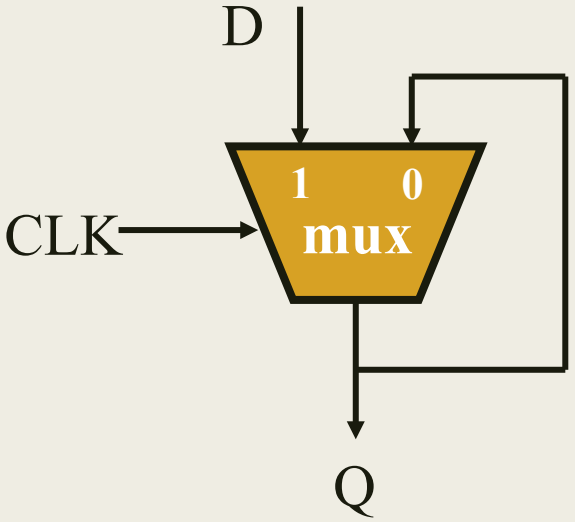
Το πρόβλημα της ελλιπούς ανάθεσης τιμής στη VHDL

- Ανεπιθύμητη εμφάνιση ενός **D-Latch** λόγω **ελλιπούς ανάθεσης τιμής**
  - *όταν για ένα σήμα **δεν ορίζεται μία εντολή ανάθεσης τιμής σε όλες τις διακλαδώσεις μίας εντολής IF**, υπάρχει το ενδεχόμενο να εμφανισθεί μετά τη σύνθεση ένα **ανεπιθύμητο D-Latch** για το συγκεκριμένο σήμα*
  - *η ελλιπής ανάθεση οδηγεί στην υλοποίηση επιπλέον λογικής που συνήθως είναι **πλεονάζουσα***
- Για να **αποφευχθεί η ελλιπής ανάθεση τιμής** κατά τη σύνθεση **συνδυαστικής λογικής**, για κάθε σήμα:
  - *βάζουμε μία **αρχική τιμή**, και*
  - *ορίζουμε μία **εντολή ανάθεσης τιμής** ή την εντολή **null**, (που σημαίνει «μην κάνεις τίποτα – διατήρησε την τρέχουσα τιμή των σημάτων»), σε **όλες τις διακλαδώσεις** μίας εντολής IF*

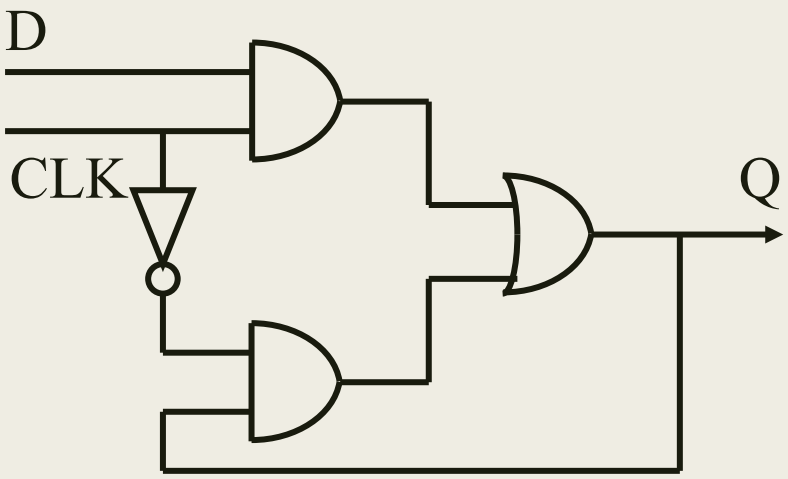
Υλοποίηση ελλιπούς ανάθεσης τιμής
Ακολουθιακή λογική

D-Latch

```
process (CLK, D)
begin
   if (CLK = '1') then
      Q <= D;
   end if;
end process;
```



Σε επίπεδο πολυπλέκτη



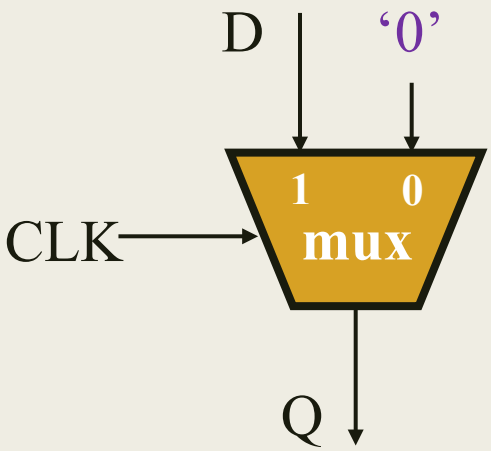
Σε επίπεδο πύλης

Υλοποίηση μη ελλιπούς ανάθεσης τιμής

Συνδυαστική λογική

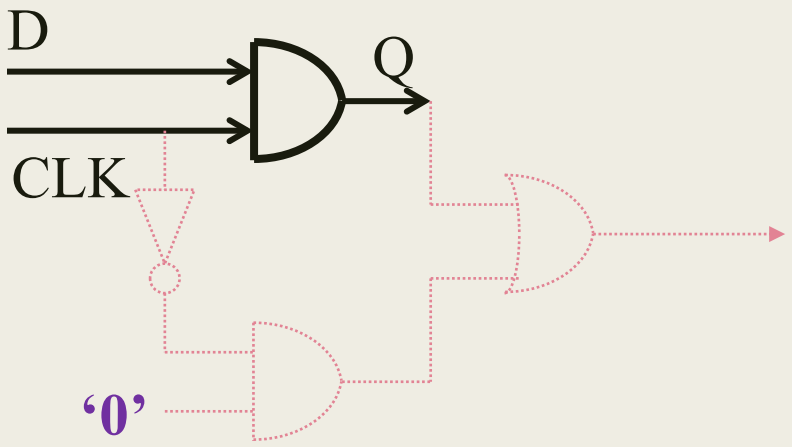
```
process (CLK, D)
begin
  if (CLK = '1') then
    Q <= D;
  else
    Q <= '0';
end if;
end process;
```

```
process (CLK, D)
begin
  Q <= '0';
  if (CLK = '1') then
    Q <= D;
  else
    null;
  end if;
end process;
```



Σε επίπεδο πολυπλέκτη

AND-2



Σε επίπεδο πύλης

Υλοποίηση μη ελλιπούς ανάθεσης τιμής

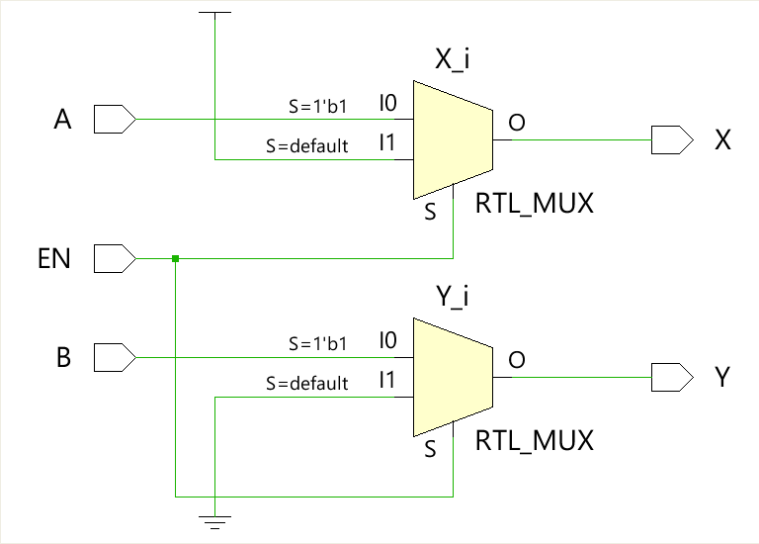
- Παράδειγμα αποφυγής εμφάνισης ελλιπούς ανάθεσης τιμής

```
architecture EXAMPLE_BEH of EXAMPLE is
begin
  process (EN, A, B)
  begin
    X <= '0';   -- αρχικές τιμές
    Y <= '0';
    if (EN = '1') then
      X <= A;
      Y <= B;
    else
      X <= '1';
    end if;
  end process;
end EXAMPLE_BEH;
```

Στην περίπτωση που δεν ικανοποιείται η συνθήκη EN = 1, η έξοδος X λαμβάνει την τιμή 1, ενώ η έξοδος Y διατηρεί την τιμή 0 που έλαβε κατά την αρχικοποίηση των εξόδων.
Λόγω της αρχικοποίησης των εξόδων, δεν υλοποιείται ακολουθιακή, αλλά συνδυαστική λογική.
Εάν δεν υπήρχε η αρχικοποίηση των εξόδων, θα υλοποιείτο ένα D Latch για την έξοδο Y.

Υλοποίηση μη ελλιπούς ανάθεσης τιμής

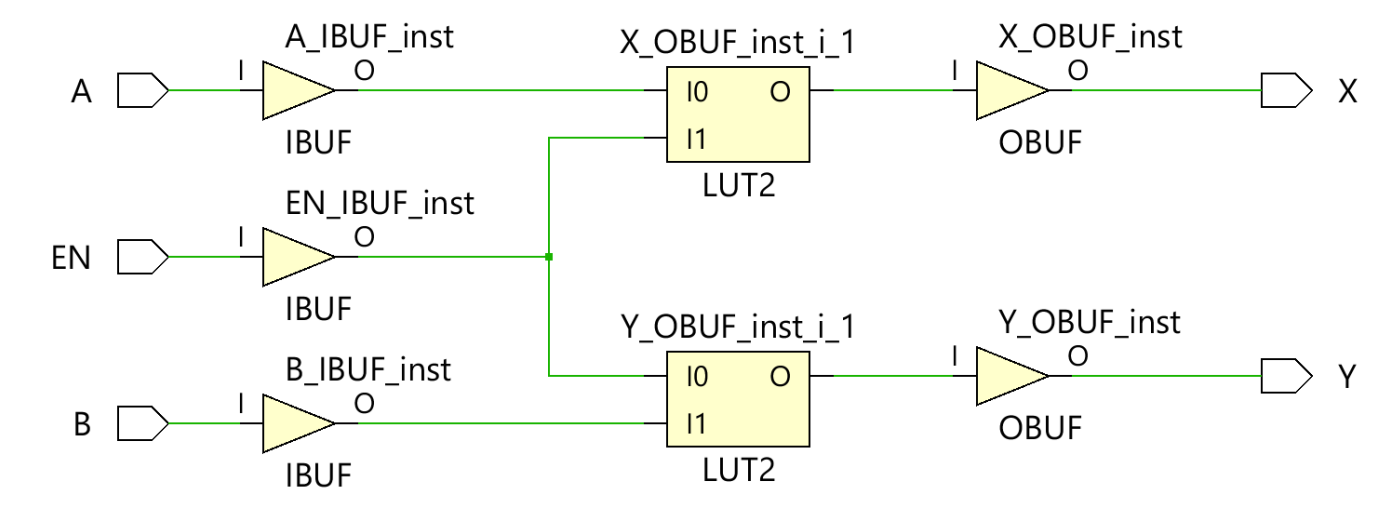
- Σχηματικό διάγραμμα RTL



$$X = ENA + \overline{EN}\,1 = A + \overline{EN}$$

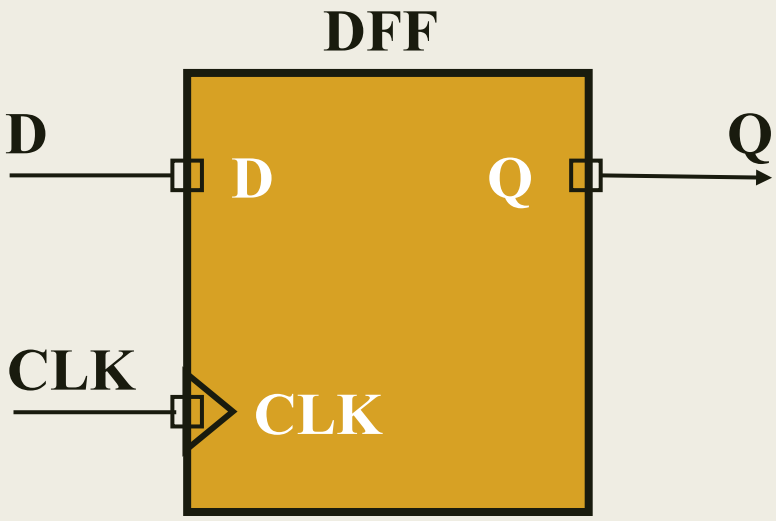
$$Y = ENB + \overline{EN}\,0 = ENB$$

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Η οντότητα του D Flip-Flop στη VHDL

```
entity DFF is
  port (
    CLK, D: in STD_LOGIC;
    Q: out STD_LOGIC);
end DFF;
```



- Ο κώδικας του D Flip-Flop είναι σχεδόν ίδιος με τον κώδικα του D Latch
- Διαφοροποιούνται μόνο στη συνθήκη του CLK
  - *Στον κώδικα του D Flip-Flop χρησιμοποιείται επιπλέον το event attribute (CLK'event), που λαμβάνει την τιμή TRUE όταν το σήμα CLK αλλάζει τιμή (0 → 1 ή 1 → 0)*
  - *Η ανερχόμενη ακμή του CLK περιγράφεται ως*
    - **CLK = '1' and CLK'event**
  - *Η κατερχόμενη ακμή του CLK περιγράφεται ως*
    - **CLK = '0' and CLK'event**

Η αρχιτεκτονική του D Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

```
architecture BEHAVIORAL of DFF is
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      Q <= D;
    end if;
  end process;
end DFF_BEH;
```

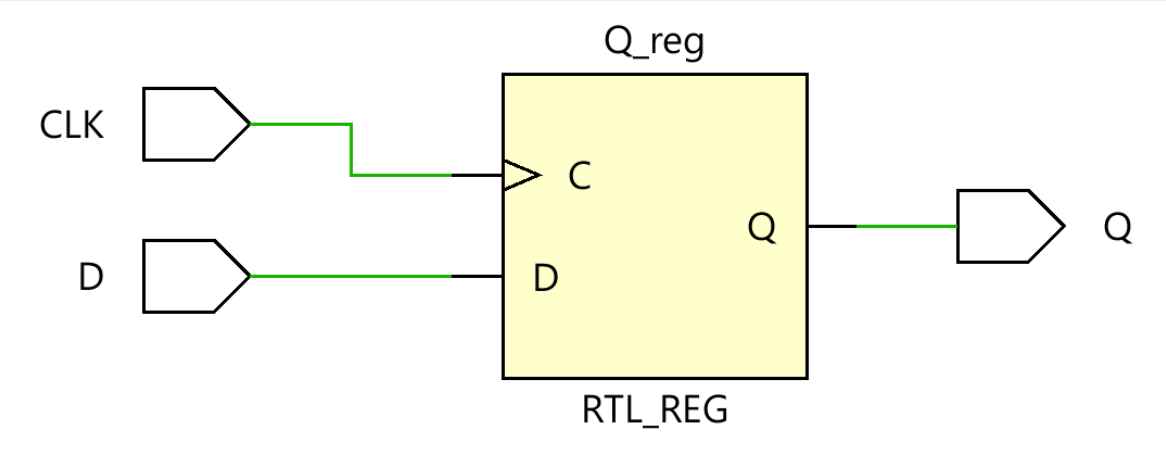
Το D δεν τοποθετείται στη λίστα ευαισθησίας

Η συνθήκη του σύγχρονου D εξετάζεται μετά τη συνθήκη του CLK

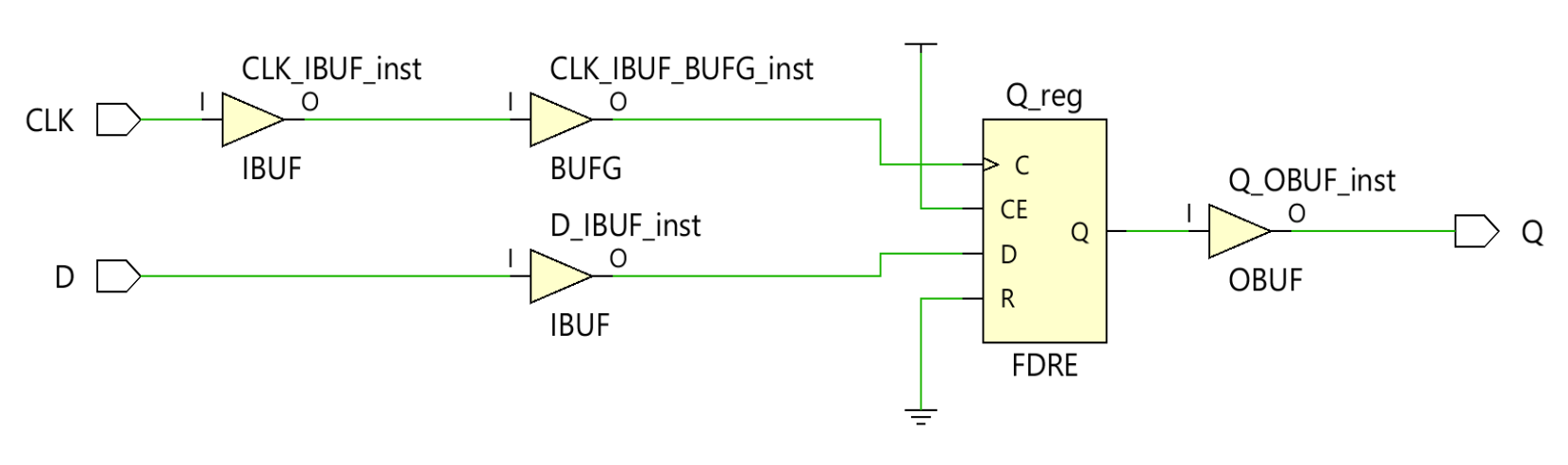
Προσοχή! Η απουσία του CLK'event οδηγεί στην υλοποίηση ενός D-Latch

Η αρχιτεκτονική του D Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



D Latch vs D Flip-Flop στη VHDL

```
architecture BEHAVIORAL of D_LATCH is
begin
  process (CLK, D)
  begin
    if (CLK = '1') then
      Q <= D;
    end if;
  end process;
end BEHABIORAL;
```

```
architecture BEHAVIORAL of DFF is
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      Q <= D;
    end if;
  end process;
end DFF_BEH;
```

D Flip-Flop με 2 εξόδους στη VHDL
Περιγραφή συμπεριφοράς

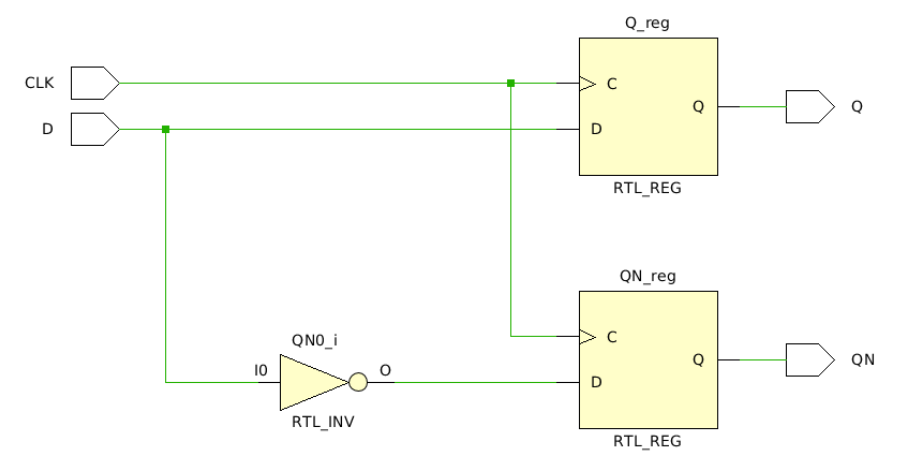
- Υλοποίηση δύο εξόδων (Q και QN) με δύο D Flip-Flop

```
entity DFF is
  port (
    CLK, D: in STD_LOGIC;
    Q, QN: out STD_LOGIC);
end DFF;
architecture DFF_BEH of DFF is
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      Q <= D; QN <= not D;
    end if;
  end process;
end DFF_BEH;
```

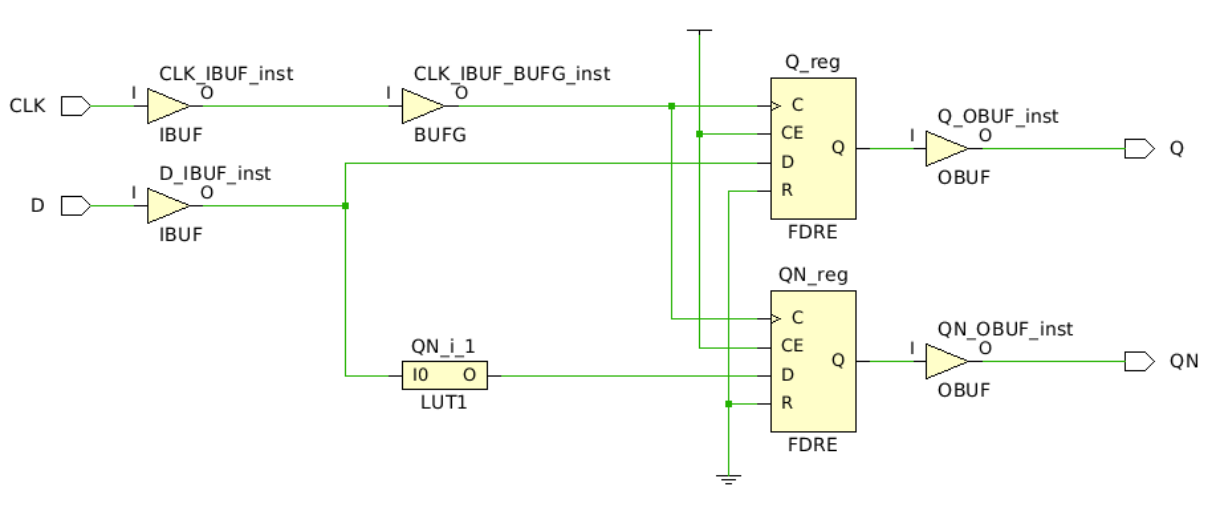
Ανάθεση τιμής στα Q και QN εντός process

D Flip-Flop με 2 εξόδους στη VHDL
Περιγραφή συμπεριφοράς με 2 D F/F

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



D Flip-Flop με 2 εξόδους στη VHDL
Περιγραφή συμπεριφοράς

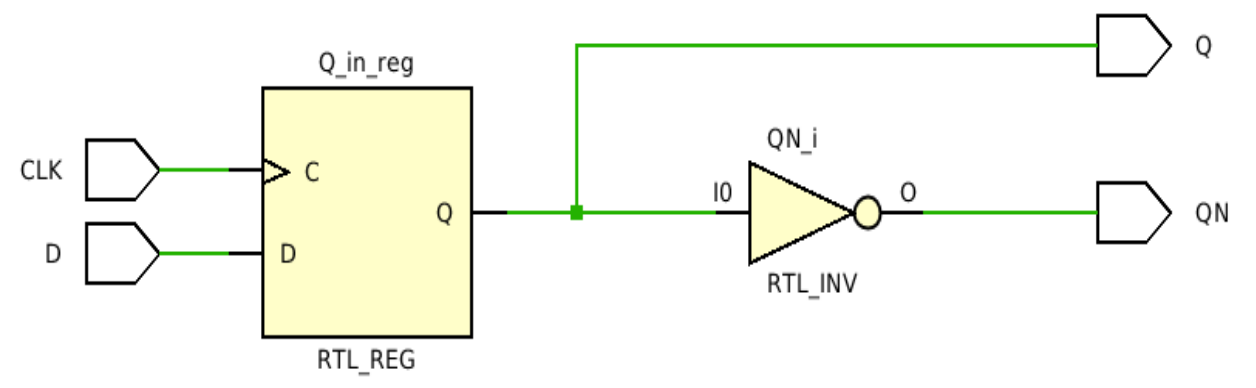
- Υλοποίηση δύο εξόδων (Q και QN) με ένα D Flip-Flop

```
entity DFFwQN is
  port (
    CLK, D: in STD_LOGIC;
    Q, QN: out STD_LOGIC);
end DFFwQN;
architecture DFFwQN_BEH of DFFwQN is
  signal Q_in: STD_LOGIC;
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      Q_in <= D;
    end if;
  end process;
  Q <= Q_in; QN <= not Q_in;
end DFFwQN_BEH;
```

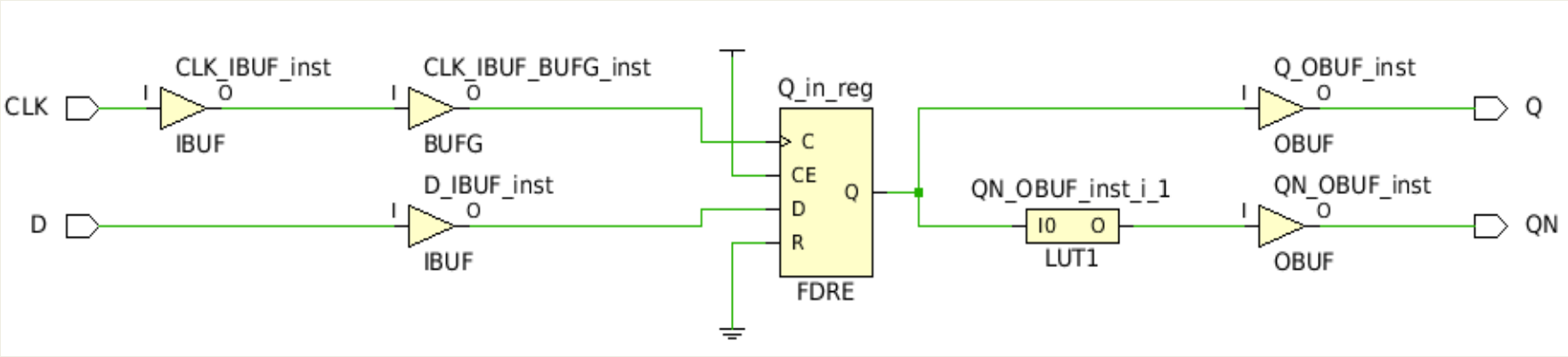
Ανάθεση τιμής στα Q και QN **εκτός** process

D Flip-Flop με 2 εξόδους στη VHDL
Περιγραφή συμπεριφοράς με 1 D F/F

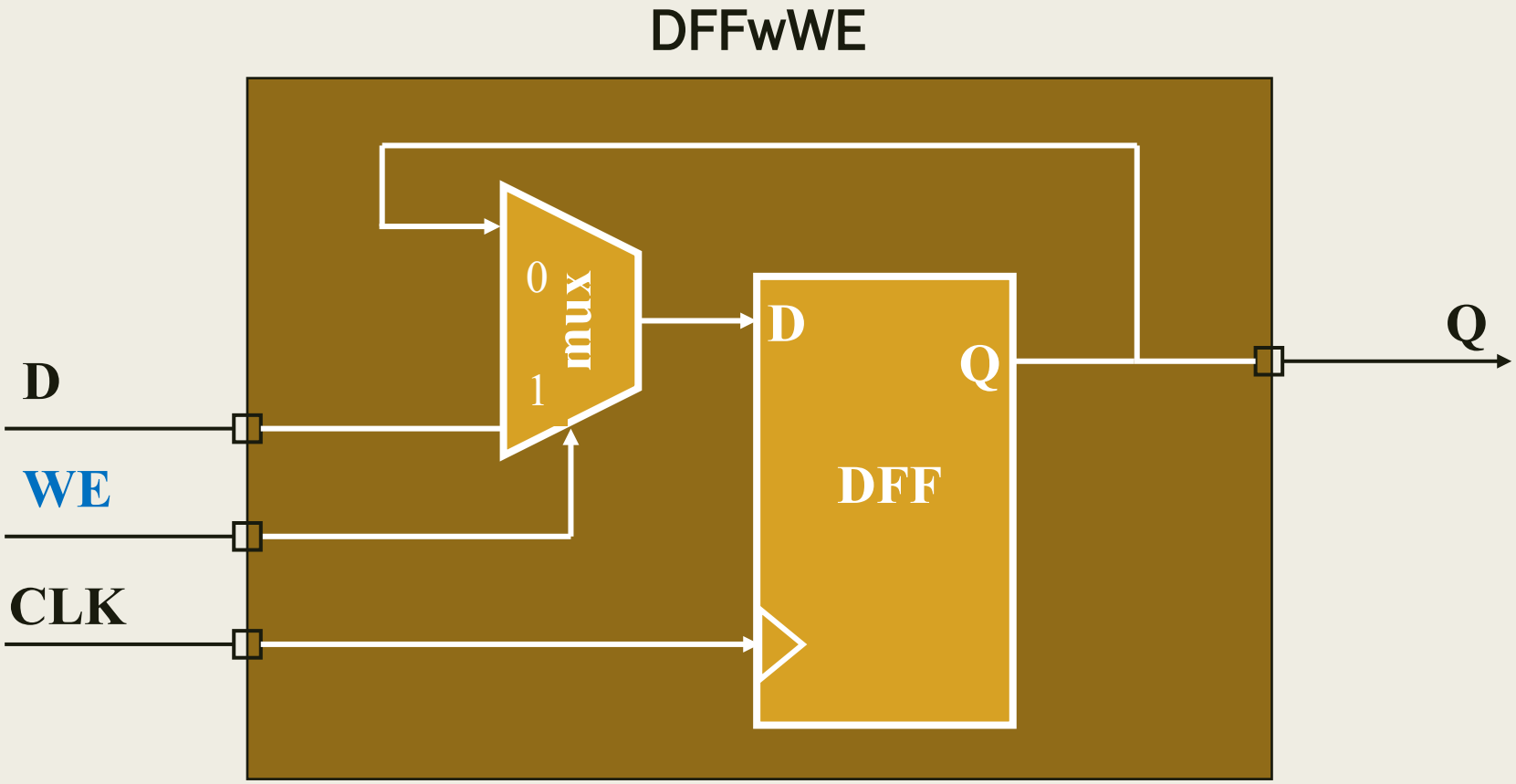
- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



D Flip-Flop with Write Enable στη VHDL
Περιγραφή συμπεριφοράς



Το σήμα Write Enable (WE = 1) είναι σύγχρονο και εγκρίνει την εγγραφή του D F/F στην επόμενη ακμή του CLK

D Flip-Flop with Write Enable στη VHDL
Περιγραφή συμπεριφοράς

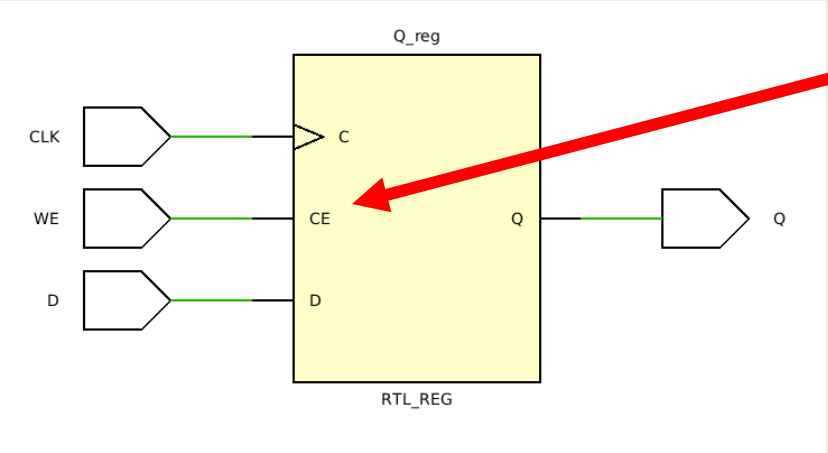
```
entity DFFwWE is
  port (
    CLK, D, WE: in STD_LOGIC;
    Q: out STD_LOGIC);
end DFFwWE;
architecture DFFwWE_BEH of DFFwWE is
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      if (WE = '1') then
        Q <= D;
      end if;
    end if;
  end process;
end DFFwWE_BEH;
```

Το WE δεν τοποθετείται στη λίστα ευαισθησίας

Η συνθήκη του σύγχρονου WE εξετάζεται μετά τη συνθήκη του CLK

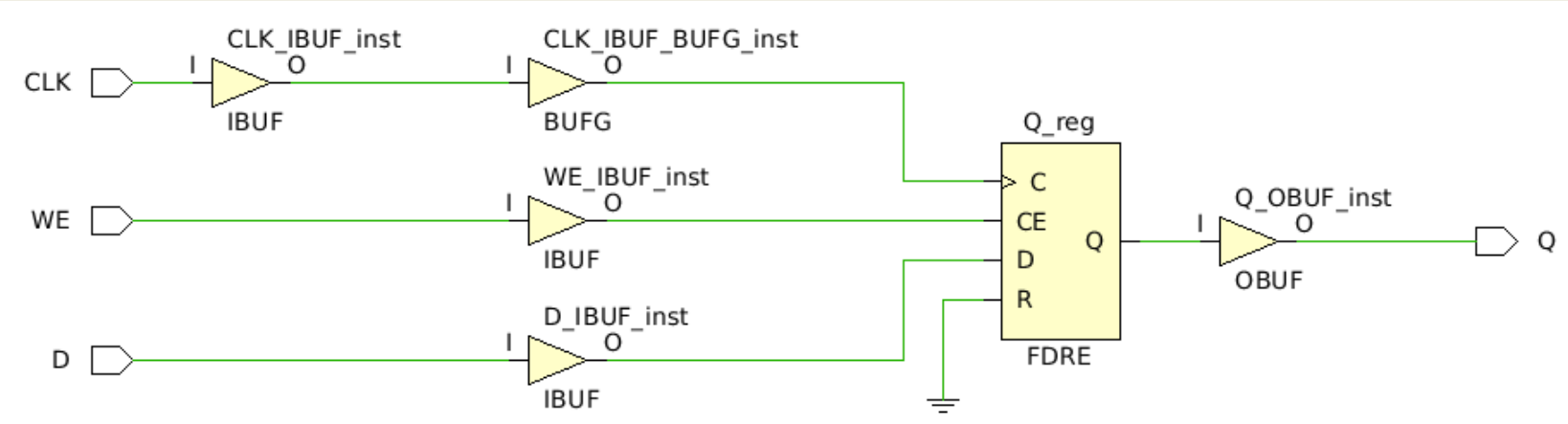
D Flip-Flop with Write Enable στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL

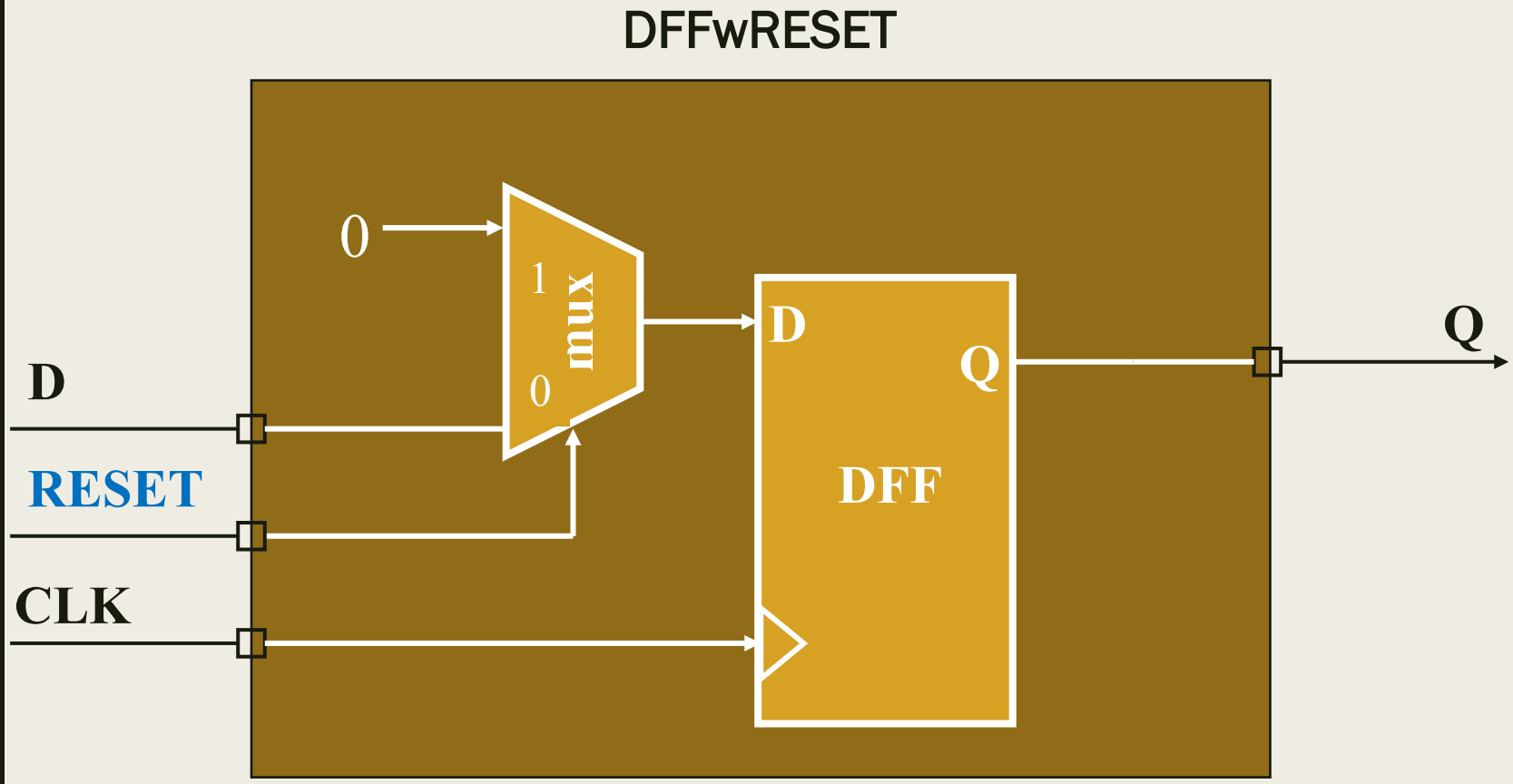


Χρησιμοποιείται η είσοδος Clock Enable (CE) του D Flip-Flop

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



D Flip-Flop with Reset στη VHDL
Περιγραφή συμπεριφοράς



Το σήμα Reset Active High (RESET = 1) είναι σύγχρονο και επαναφέρει το D F/F στην κατάσταση στο 0 στην επόμενη ακμή του CLK

D Flip-Flop with Reset στη VHDL
Περιγραφή συμπεριφοράς

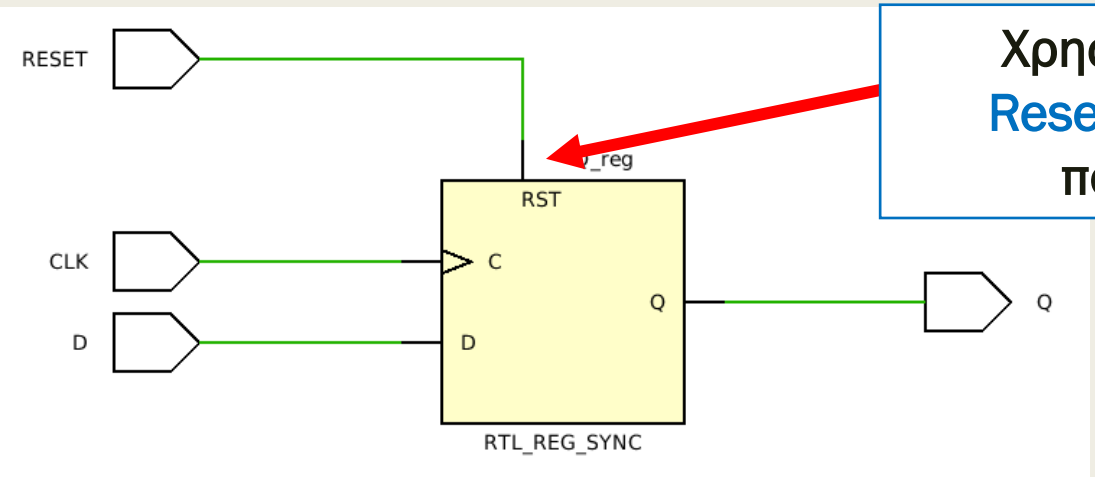
```
entity DFFwRESET is
  port (
    CLK, D, RESET: in STD_LOGIC;
    Q: out STD_LOGIC);
end DFFwRESET;
architecture DFFwRESET_BEH of DFFwRESET is
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      if (RESET = '1') then
      Q <= '0';
      else
        Q <= D;
      end if;
    end if;
  end process;
end DFFwRESET_BEH;
```

Το RESET δεν τοποθετείται στη λίστα ευαισθησίας

Η συνθήκη του σύγχρονου RESET εξετάζεται μετά τη συνθήκη του CLK

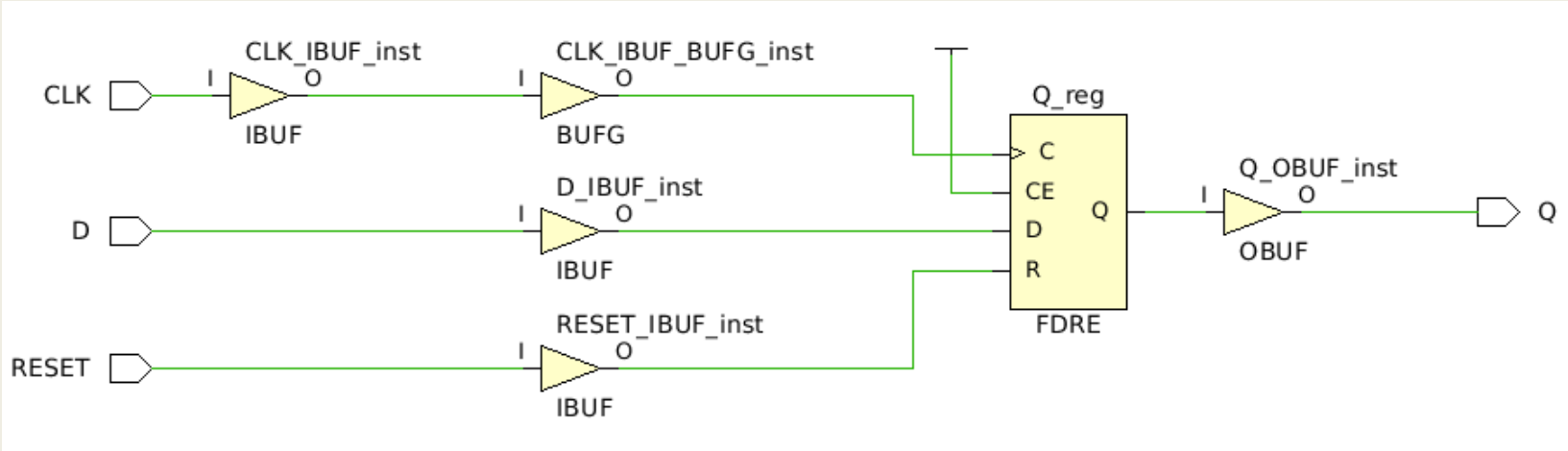
D Flip-Flop with Reset στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL

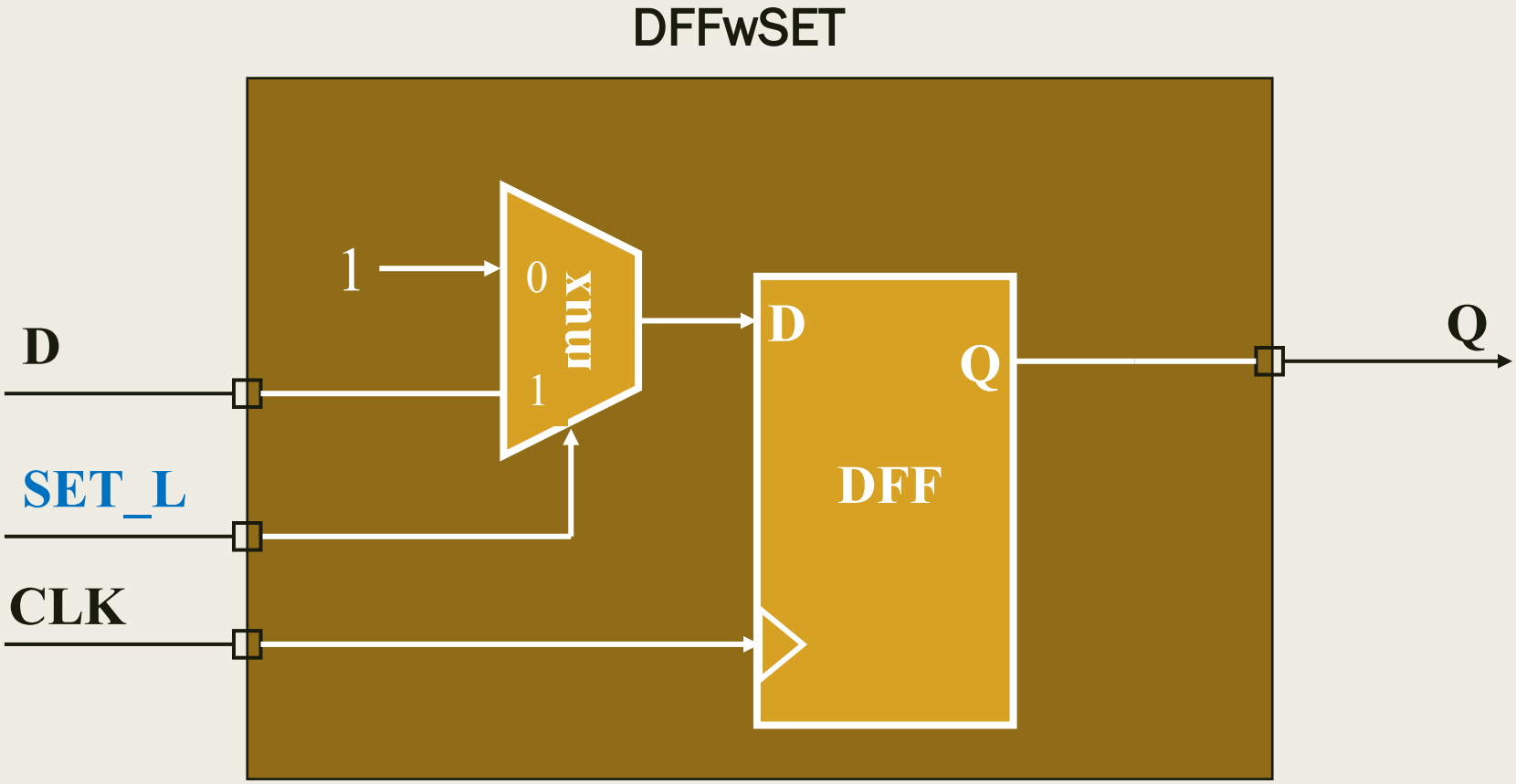


Χρησιμοποιείται η είσοδος Reset (RST) του D Flip-Flop που είναι active high

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



D Flip-Flop with Set στη VHDL
Περιγραφή συμπεριφοράς



Το σήμα Set Active Low (SET_L = 0) είναι σύγχρονο και τοποθετεί το D F/F στην κατάσταση 1 στην επόμενη ακμή του CLK

D Flip-Flop with Set στη VHDL
Περιγραφή συμπεριφοράς

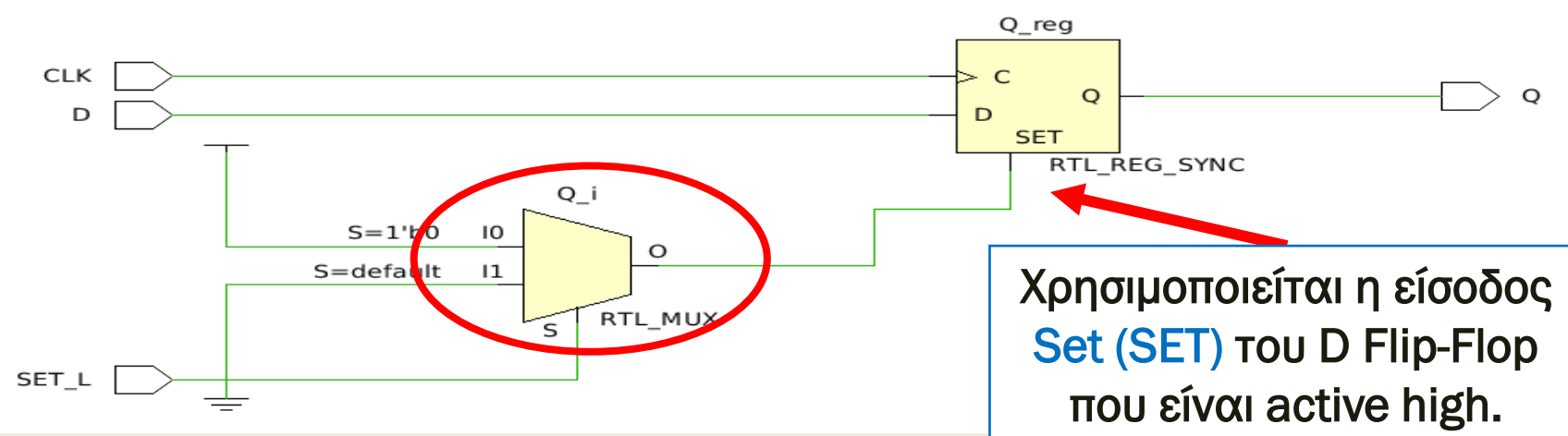
```
entity DFFwSET is
  port (
    CLK, D, SET_L: in STD_LOGIC;
    Q: out STD_LOGIC);
end DFFwSET;
architecture DFFwSET_BEH of DFFwSET is
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      if (SET_L = '0') then
        Q <= '1';
      else
        Q <= D;
      end if;
    end if;
  end process;
end DFFwSET_BEH;
```

Το SET_L δεν τοποθετείται στη λίστα ευαισθησίας

Η συνθήκη του σύγχρονου SET εξετάζεται μετά τη συνθήκη του CLK

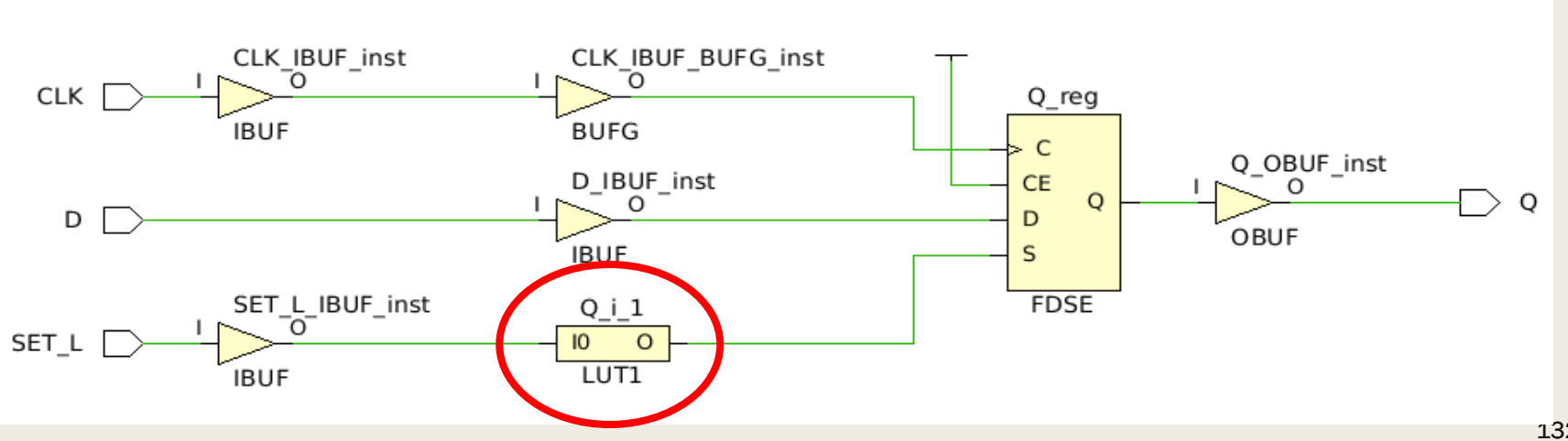
D Flip-Flop with Set στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL

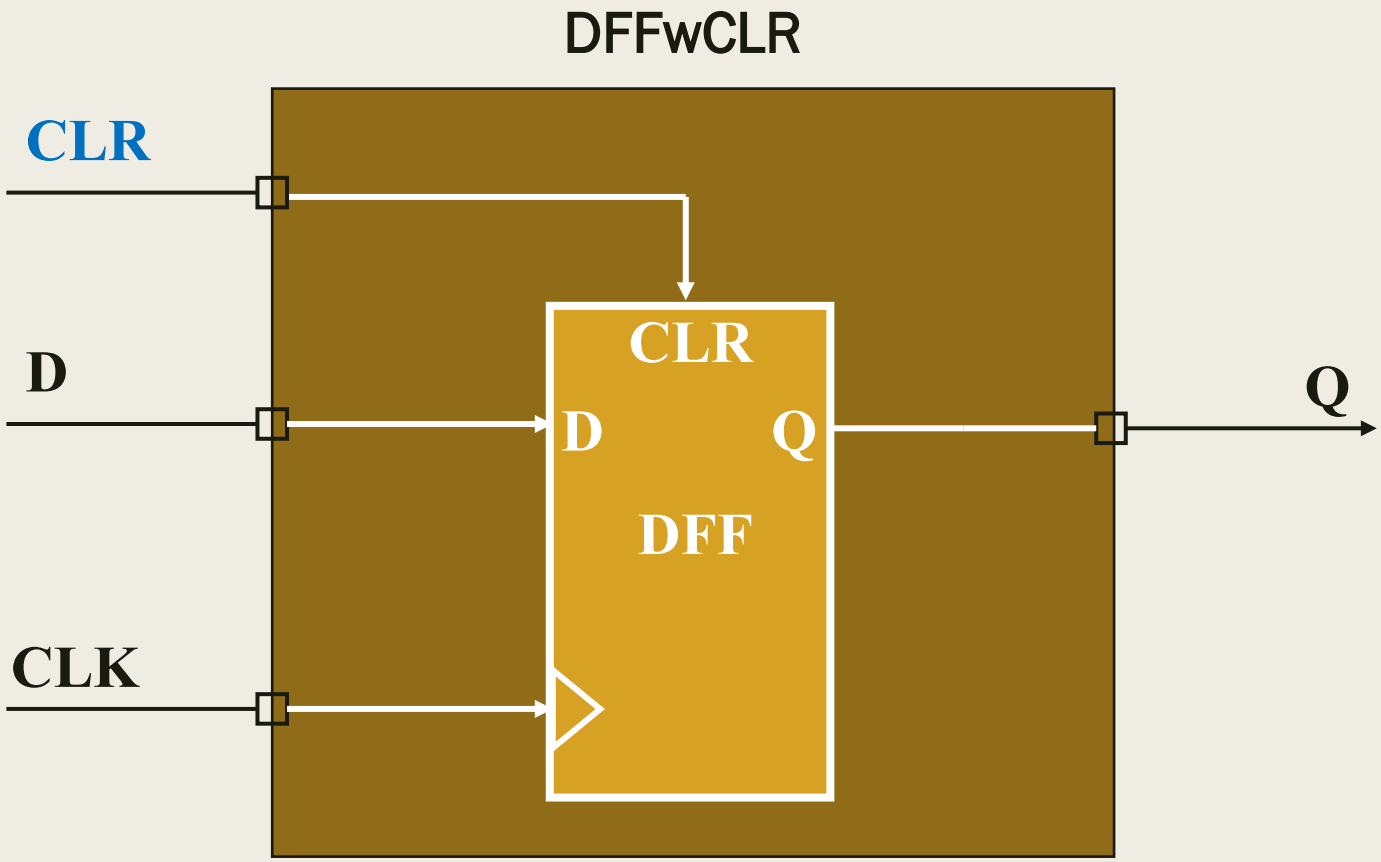


Χρησιμοποιείται η είσοδος Set (SET) του D Flip-Flop που είναι active high. Απαιτείται η χρήση NOT

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



D Flip-Flop with Clear στη VHDL
Περιγραφή συμπεριφοράς



Το σήμα **Clear Active High (CLR = 1)** είναι **ασύγχρονο** και επαναφέρει το D F/F στην κατάσταση 0 άμεσα, ανεξάρτητα από το CLK

D Flip-Flop with Clear στη VHDL
Περιγραφή συμπεριφοράς

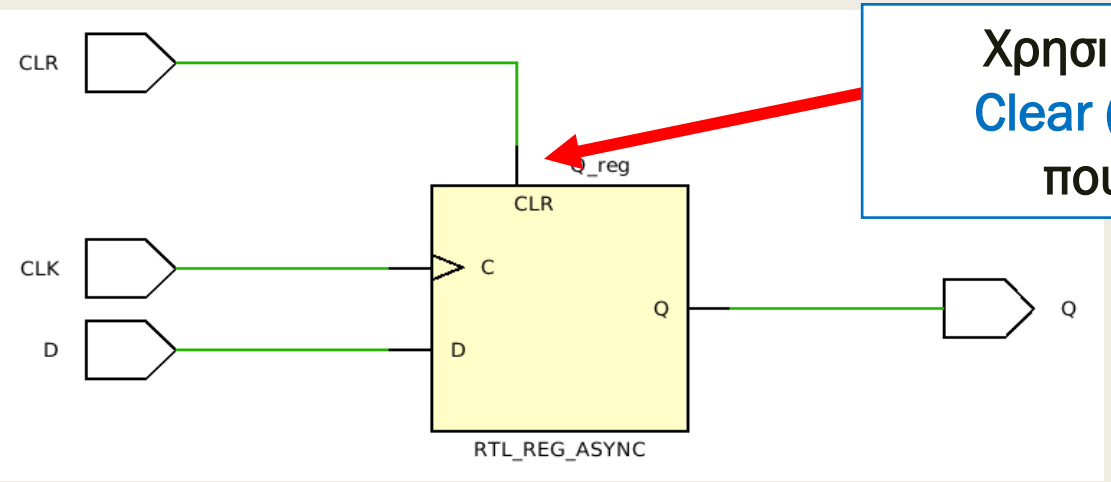
```
entity DFFwCLR is
  port (
    D, CLK, CLR: in STD_LOGIC;
    Q: out STD_LOGIC);
end DFFwCLR;
architecture DFFwCLR_BEH of DFFwCLR is
begin
  process (CLK, CLR)
  begin
    if (CLR = '1') then
      Q <= '0';
    elsif (CLK = '1' and CLK'event) then
      Q <= D;
    end if;
  end process;
end DFFwCLR_BEH;
```

Το CLR τοποθετείται στη λίστα ευαισθησίας

Η συνθήκη του ασύγχρονου CLR εξετάζεται πριν τη συνθήκη του CLK

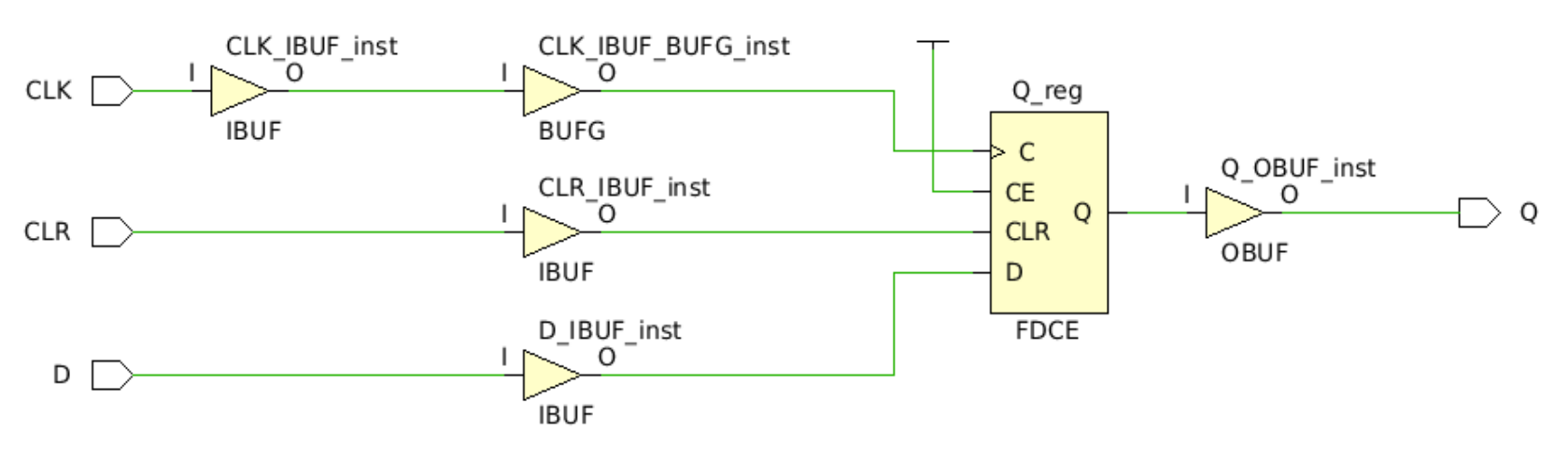
D Flip-Flop with Clear στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



Χρησιμοποιείται η είσοδος Clear (CLR) του D Flip-Flop που είναι active high

- Σχηματικό διάγραμμα σε τεχνολογία FPGA

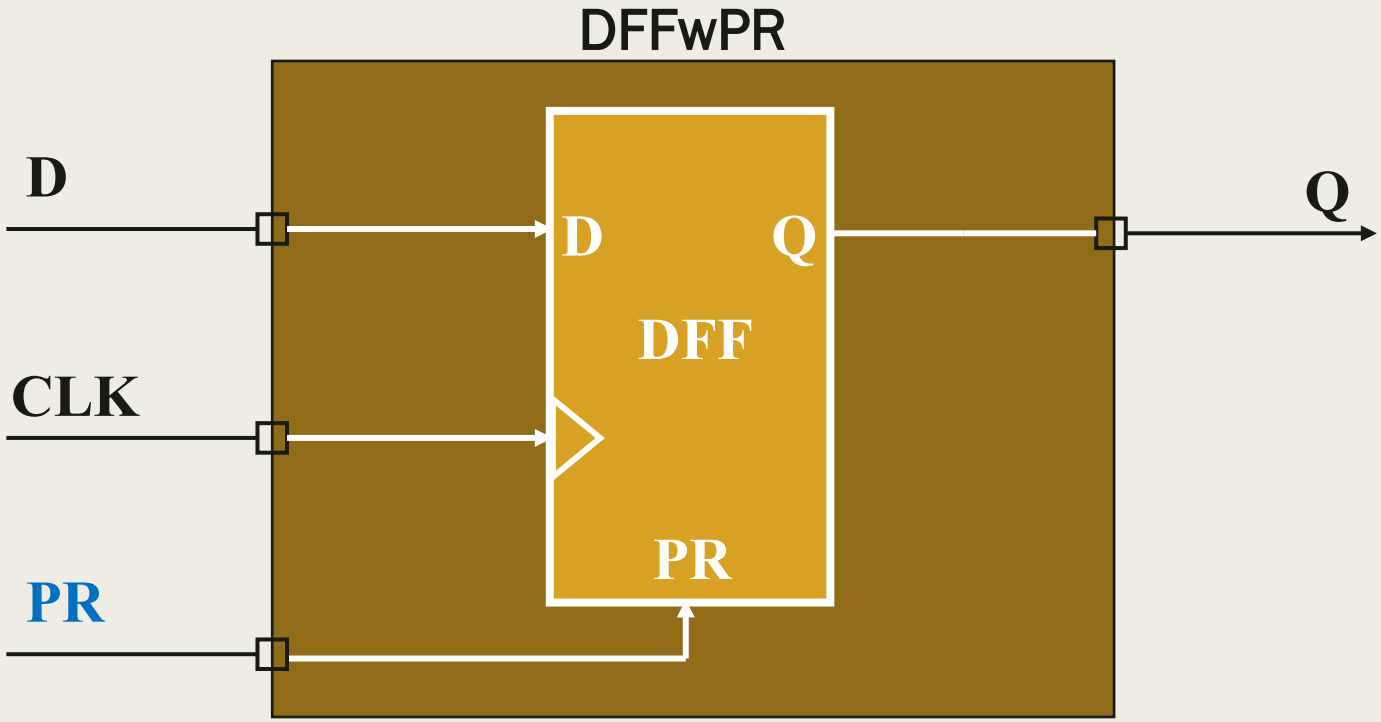


Σύγχρονο RESET vs ασύγχρονου CLEAR στη VHDL

```
architecture DFFwRESET_BEH of DFFwRESET is
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
       if (RESET = '1') then
       Q <= '0';
      else
        Q <= D;
      end if;
    end if;
  end process;
end DFFwRESET_BEH;
```

```
architecture DFFwCLR_BEH of DFFwCLR is
begin
  process (CLK, CLR)
  begin
    if (CLR = '1') then
      Q <= '0';
    elsif (CLK = '1' and CLK'event) then
      Q <= D;
    end if;
  end process;
end DFFwCLR_BEH;
```

D Flip-Flop with Preset στη VHDL
Περιγραφή συμπεριφοράς



Το σήμα **Preset Active High (PR = 1)** είναι **ασύγχρονο** και θέτει το D F/F στην κατάσταση 1 άμεσα, ανεξάρτητα από το CLK

D Flip-Flop with Preset στη VHDL
Περιγραφή συμπεριφοράς

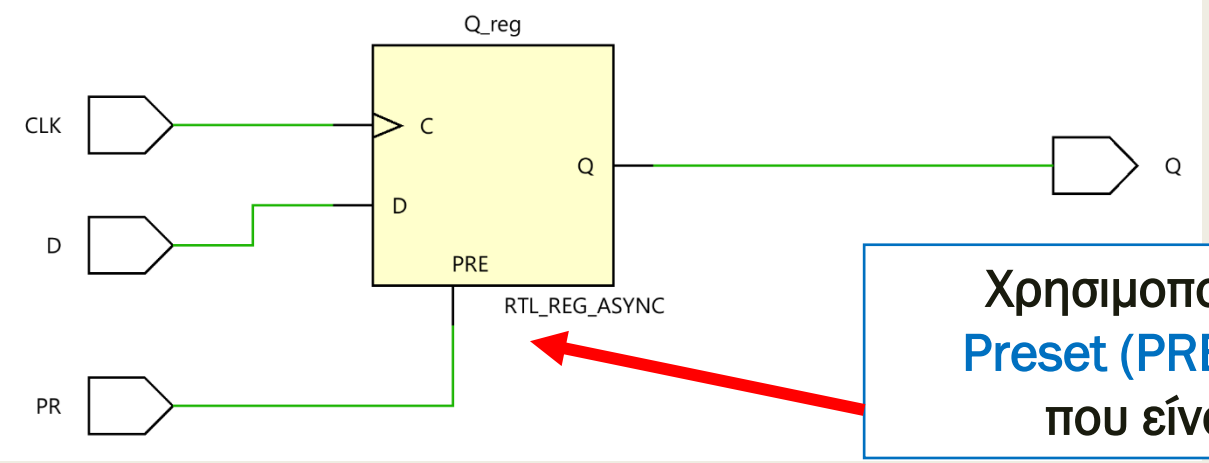
```
entity DFFwCLR is
  port (
    D, CLK, CLR: in STD_LOGIC;
    Q: out STD_LOGIC);
end DFFwCLR;
architecture DFFwPR_BEH of DFFwPR is
begin
  process (CLK, PR)
  begin
    if (PR = '1') then
      Q <= '1';
    elsif (CLK = '1' and CLK'event) then
      Q <= D;
    end if;
  end process;
end DFFwPR_BEH;
```

Το PR τοποθετείται στη λίστα ευαισθησίας

Η συνθήκη του ασύγχρονου PR εξετάζεται πριν τη συνθήκη του CLK

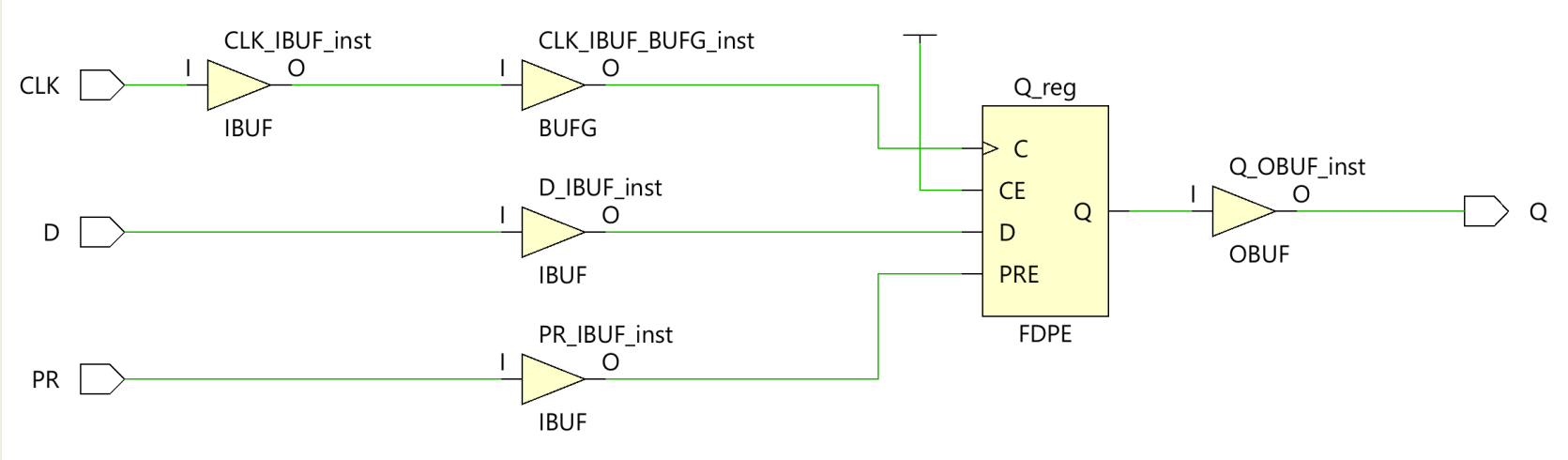
D Flip-Flop with Preset στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL

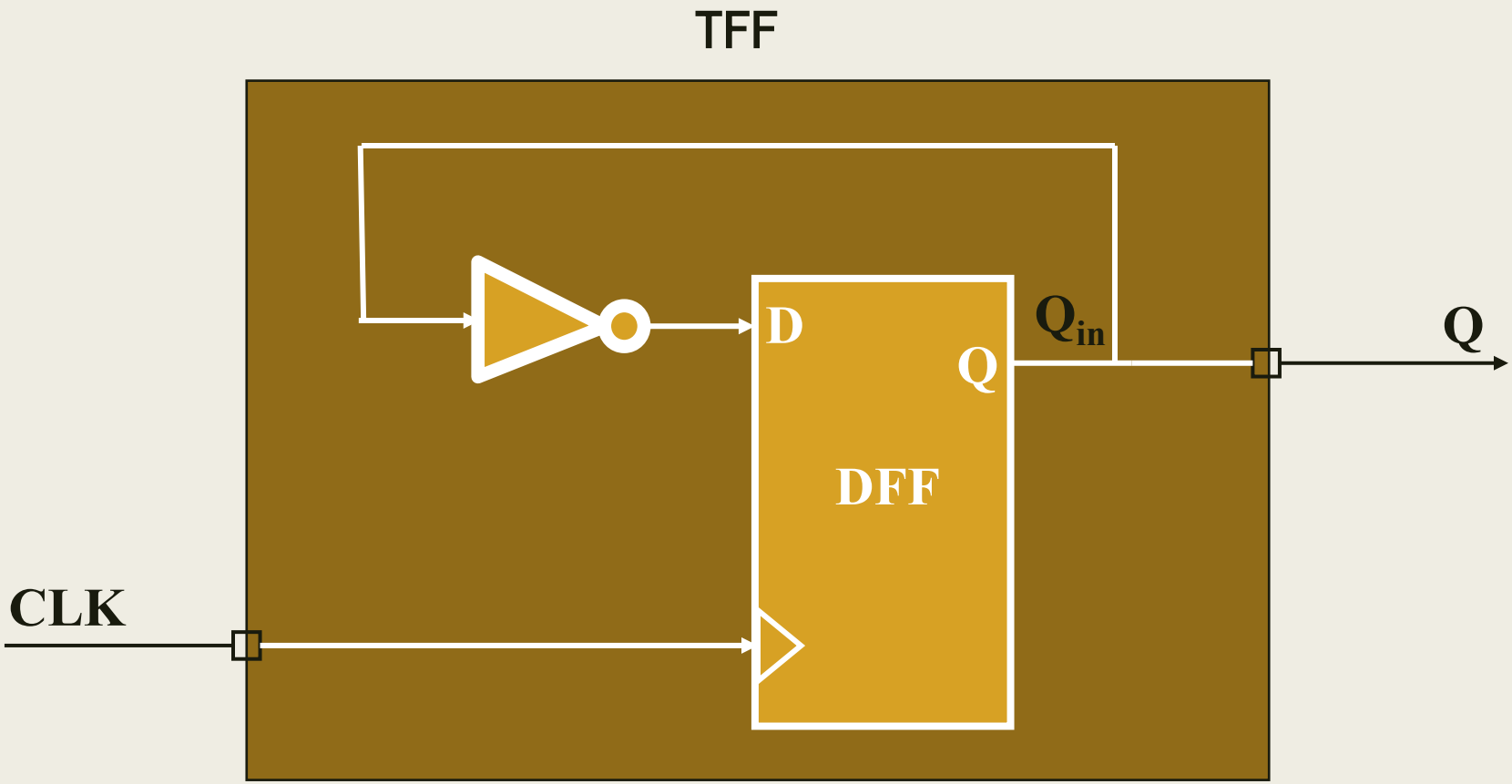


Χρησιμοποιείται η είσοδος **Preset (PRE)** του D Flip-Flop που είναι active high

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



T (Toggle) Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς



Σε κάθε ανερχόμενη ακμή του CLK αλλάζει κατάσταση (0 → 1 → 0 ...)

T Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

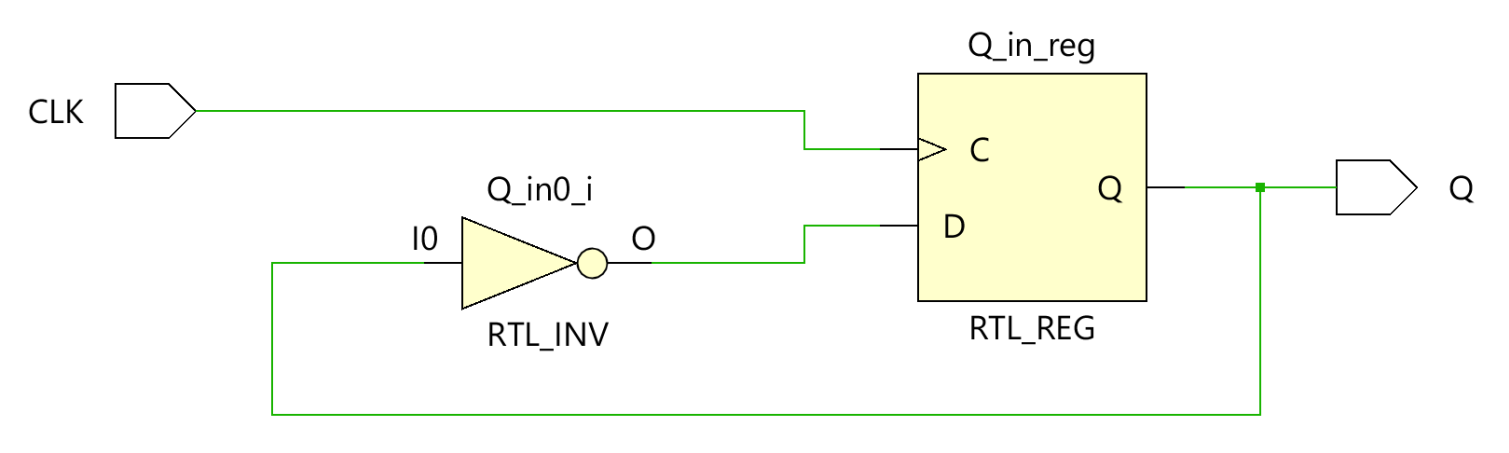
```
entity TFF is
  port (
    CLK : in STD_LOGIC;
    Q : out STD_LOGIC);
end TFF;
architecture BEHAVIORAL of TFF is
  signal Q_in: STD_LOGIC;
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      Q_in <= not Q_in;
    end if;
  end process;
  Q <= Q_in;
end BEHAVIORAL;
```

Απαίτηση για εσωτερικό σήμα Q_in

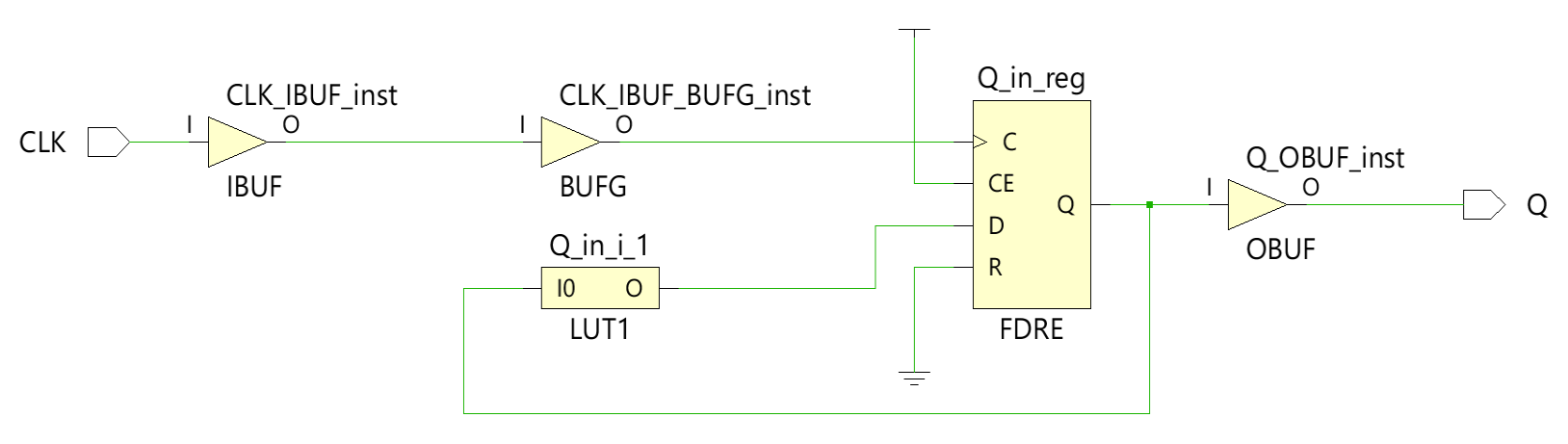
Ανάθεση τιμής στο Q εκτός process

T Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

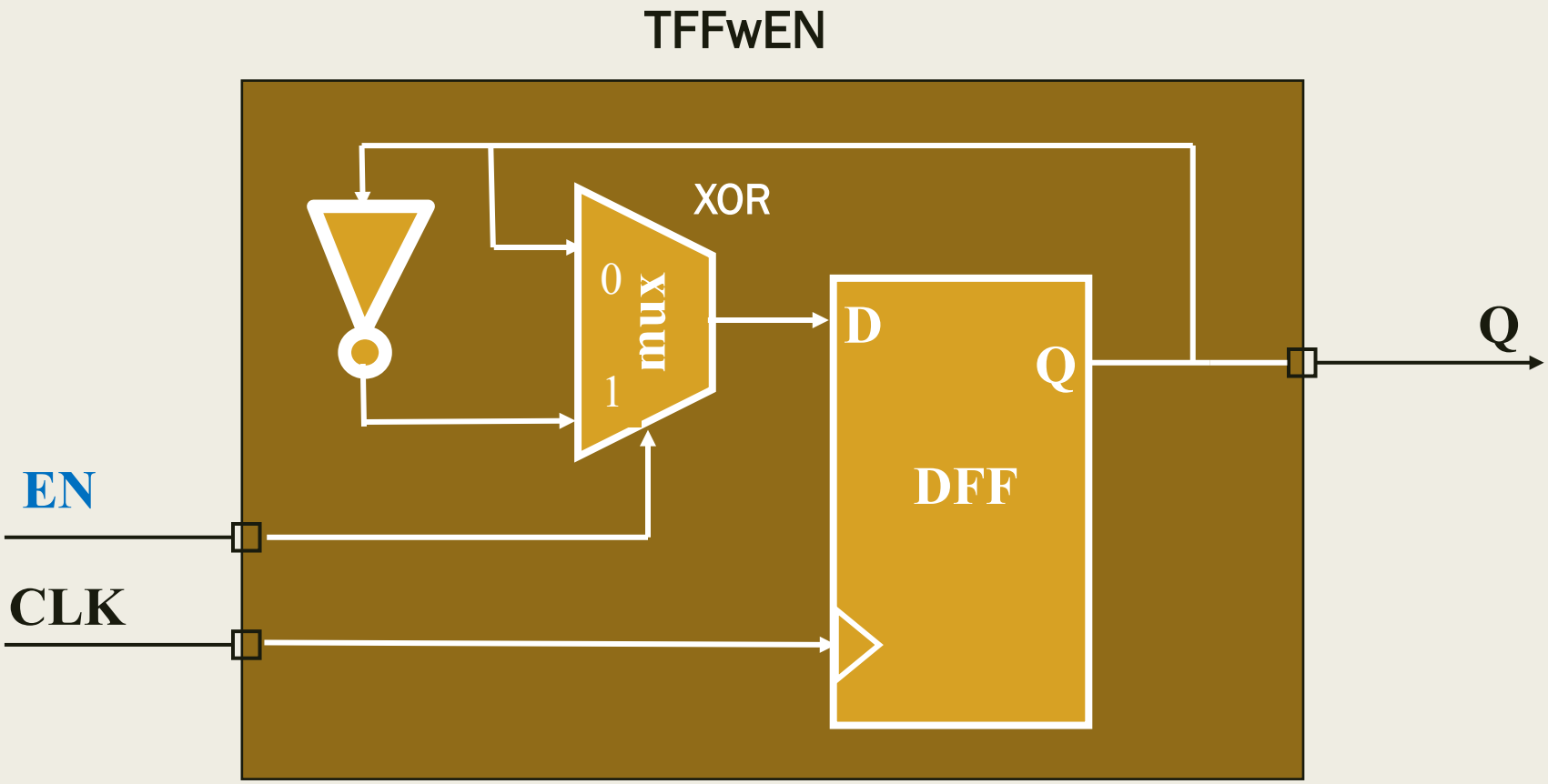
- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



T Flip-Flop with Enable στη VHDL
Περιγραφή συμπεριφοράς



Το σήμα Enable (EN = 1) είναι σύγχρονο και εγκρίνει την αλλαγή κατάστασης του T F/F στην επόμενη ακμή του CLK

T Flip-Flop with Enable στη VHDL

Περιγραφή συμπεριφοράς

```
entity TFFwEN is
  port (
    CLK, EN : in STD_LOGIC;
    Q : out STD_LOGIC);
end TFFwEN;
architecture BEHAVIORAL of TFFwEN is
  signal Q_in: STD_LOGIC;
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      if (EN = '1') then
      Q_in <= not Q_in;
      end if;
    end if;
  end process;
  Q <= Q_in;
end BEHAVIORAL;
```

Απαίτηση για εσωτερικό σήμα Q_in

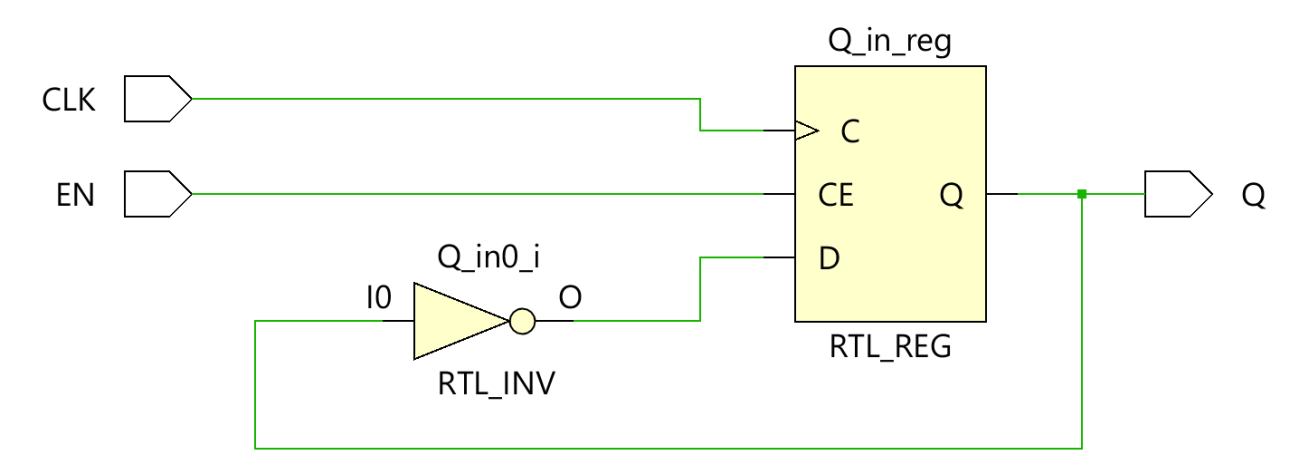
Το EN δεν τοποθετείται στη λίστα ευαισθησίας

Η συνθήκη του σύγχρονου EN εξετάζεται μετά τη συνθήκη του CLK

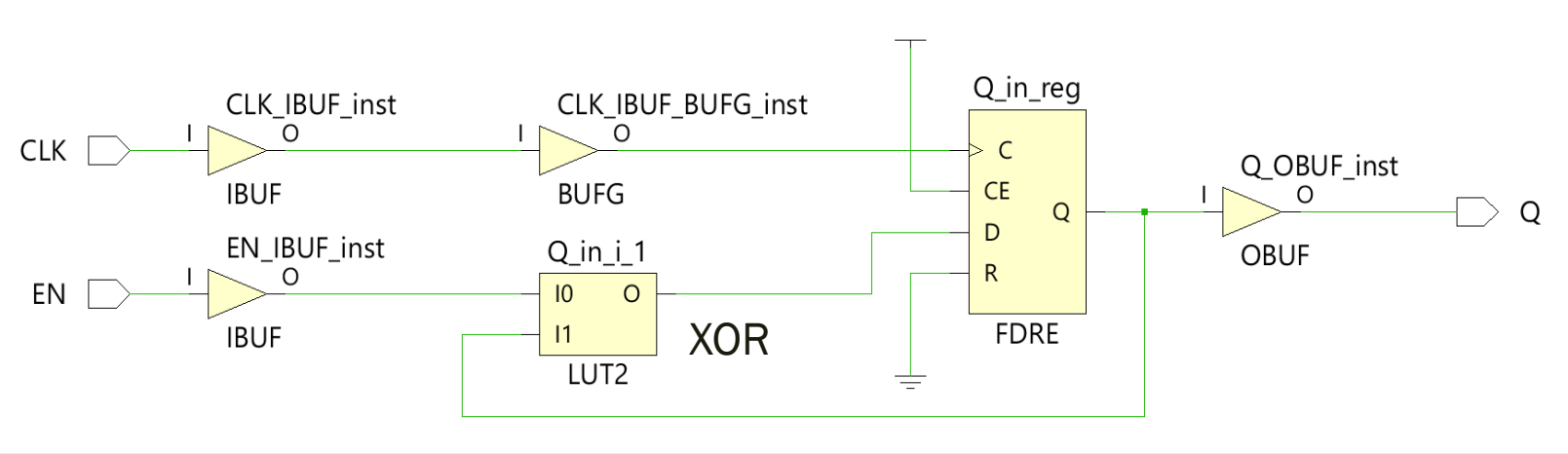
Ανάθεση τιμής στο Q εκτός process

T Flip-Flop with Enable στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



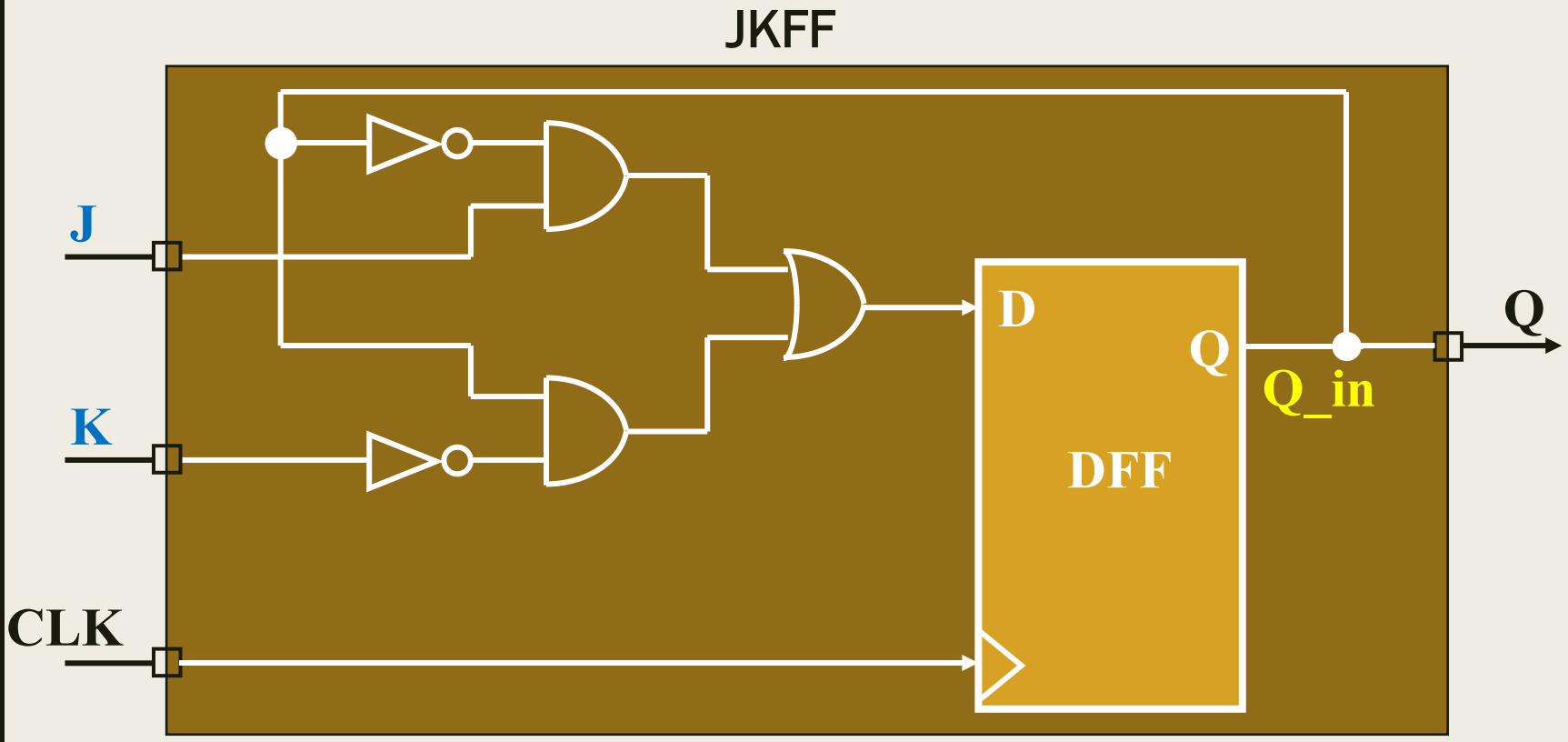
JK Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

- Το J-K Flip-Flop δέχεται ένα σήμα CLK και δύο σύγχρονες εισόδους (το *J* και το *K*).
- Κατά την ανερχόμενη ακμή του CLK ενημερώνει την έξοδο *Q*, *σύμφωνα με τις τιμές που έχουν οι είσοδοι J και K, ως εξής:*
  - *Όταν οι είσοδοι J και K έχουν και οι δύο την τιμή 0, η έξοδος Q διατηρεί την προηγούμενη τιμή της (**hold**)*
  - *Όταν η είσοδος J έχει την τιμή 0 και η είσοδος K έχει την τιμή 1, η έξοδος Q παίρνει την τιμή 0 (**reset**)*
  - *Όταν η είσοδος J έχει την τιμή 1 και η είσοδος K έχει την τιμή 0, η έξοδος Q παίρνει την τιμή 1 (**set**)*
  - *Όταν οι είσοδοι J και K έχουν και οι δύο την τιμή 1, η έξοδος Q εναλλάσσει την τιμή της με το συμπλήρωμα της προηγούμενης τιμής της (**toggle**)*
- Το J-K Flip-Flop υλοποιεί την εξίσωση Boole:

$$Q(t+1) = \overline{Q(t)}\ J + Q(t)\ \overline{K}$$

JK Flip-Flop στη VHDL

Περιγραφή συμπεριφοράς



Τα σήματα J και K είναι σύγχρονα

Υλοποιείται η εξίσωση Boole:

$$Q(t+1) = \overline{Q(t)}\ J + Q(t)\ \overline{K}$$

JK Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

```
entity JKFF is
  port (
    CLK, J, K: in STD_LOGIC;
    Q: out STD_LOGIC);
end JKFF;
architecture JKFF_BEH of JKFF is
  signal Q_in: STD_LOGIC;
begin
  process (CLK)
    begin
      if (CLK = '1' and CLK'event) then
        Q_in <= (J and (not Q_in)) or ((not K) and Q_in);
      end if;
  end process;
  Q <= Q_in;
end JKFF_BEH;
```

Απαίτηση για εσωτερικό σήμα Q_in

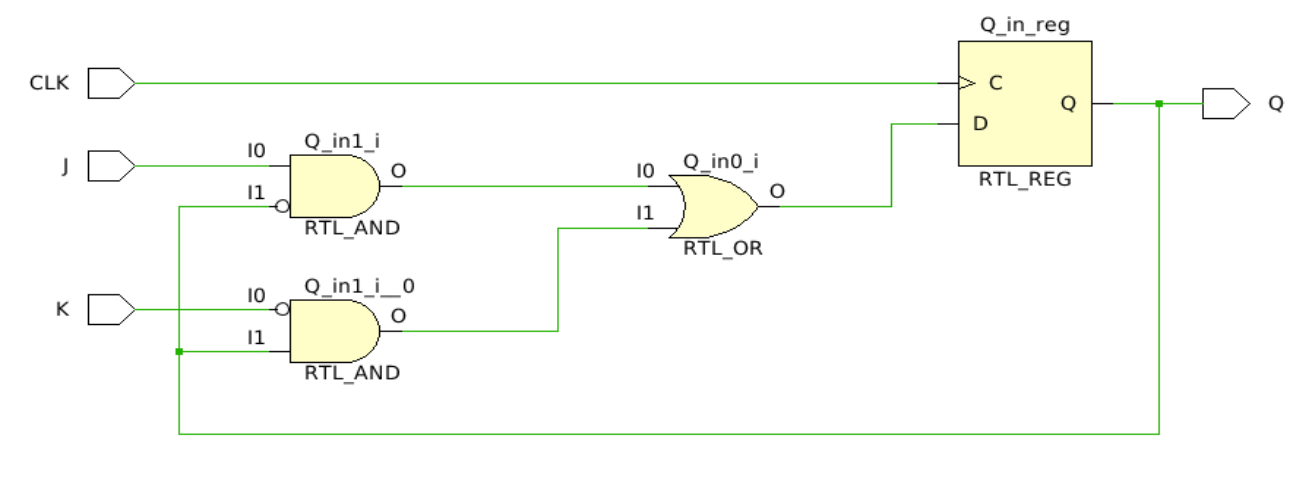
Τα J, K δεν τοποθετούνται στη λίστα ευαισθησίας

Η εξίσωση Boole του Flip-Flop περιγράφεται μετά τη συνθήκη του CLK

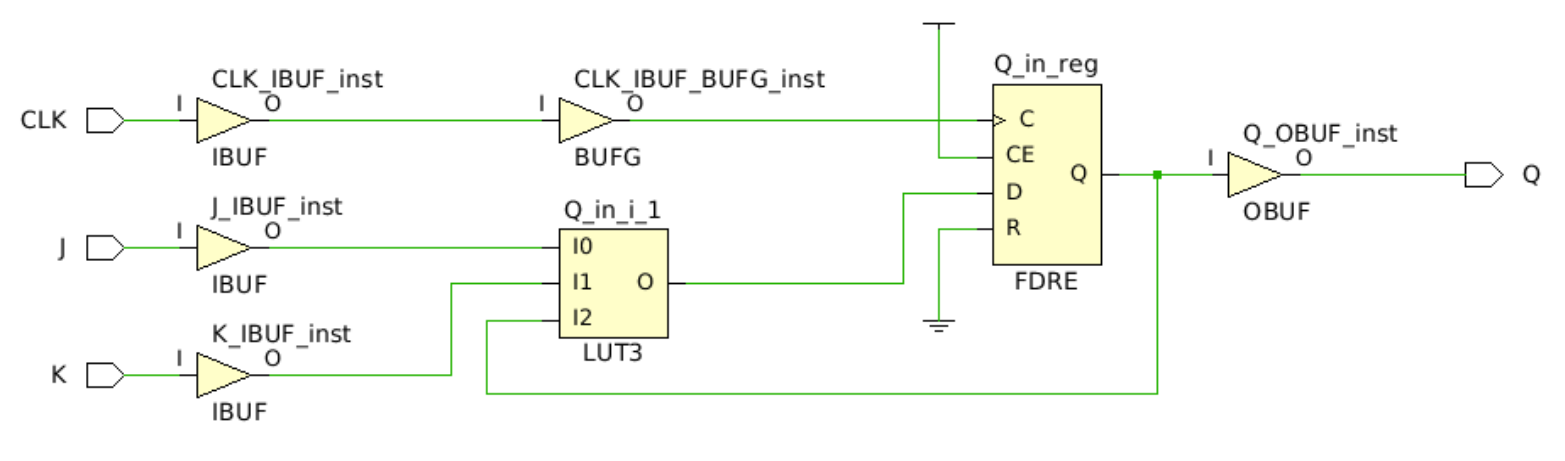
Ανάθεση τιμής στο Q εκτός process

JK Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL – Υλοποιείται η εξίσωση Boole



- Σχηματικό διάγραμμα σε τεχνολογία FPGA – Το LUT3 υλοποιεί την εξίσωση Boole



Επιλεγμένη άσκηση: το AB Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

- Να περιγράψετε στη γλώσσα VHDL την αρχιτεκτονική του **AB Flip-Flop**

AB	Λειτουργία
00	HOLD
01	TOGGLE
10	RESET
11	LOAD

Επιλεγμένη άσκηση: το AB Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

- Να περιγράψετε στη γλώσσα VHDL την αρχιτεκτονική του **AB Flip-Flop**

AB	Λειτουργία
00	HOLD
01	TOGGLE
10	RESET
11	LOAD

AB	Λειτουργία	Q(t+1)
00	HOLD	Q(t)
01	TOGGLE	$\overline{Q(t)}$
10	RESET	0
11	LOAD	D

Επιλεγμένη άσκηση: το AB Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

- Να περιγράψετε στη γλώσσα VHDL την αρχιτεκτονική του **AB Flip-Flop**

AB	Λειτουργία
00	HOLD
01	TOGGLE
10	RESET
11	LOAD

AB	Λειτουργία	Q(t+1)
00	HOLD	Q(t)
01	TOGGLE	$\overline{Q(t)}$
10	RESET	0
11	LOAD	D

$$Q(t+1)=\bar{A}\bar{B}Q(t)+\bar{A}B\overline{Q(t)}+ABD$$
$$=\bar{A}(\bar{B}Q(t)+B\overline{Q(t)})+ABD$$
$$=\bar{A}(Q(t)\,xor\,B)+ABD$$

Επιλεγμένη άσκηση: το AB Flip-Flop στη VHDL

Περιγραφή συμπεριφοράς

- Να περιγράψετε στη γλώσσα VHDL την αρχιτεκτονική του AB Flip-Flop με εξίσωση Boole:

$$Q(t+1) = \overline{A}\left(B \oplus Q(t)\right) + A(BD)$$

AB	Λειτουργία
00	HOLD
01	TOGGLE
10	RESET
11	LOAD

```
architecture BEHAVIORAL of ABFF is
  signal Q_in: STD_LOGIC;
begin
  process (CLK)
    begin
      if (CLK = '1' and CLK'event) then
      Q_in <= ((not A) and (B xor Q_in)) or ((A and (B and D)));
      end if;
  end process;
  Q <= Q_in;
end BEHAVIORAL;
```

Απαίτηση για εσωτερικό σήμα Q_in

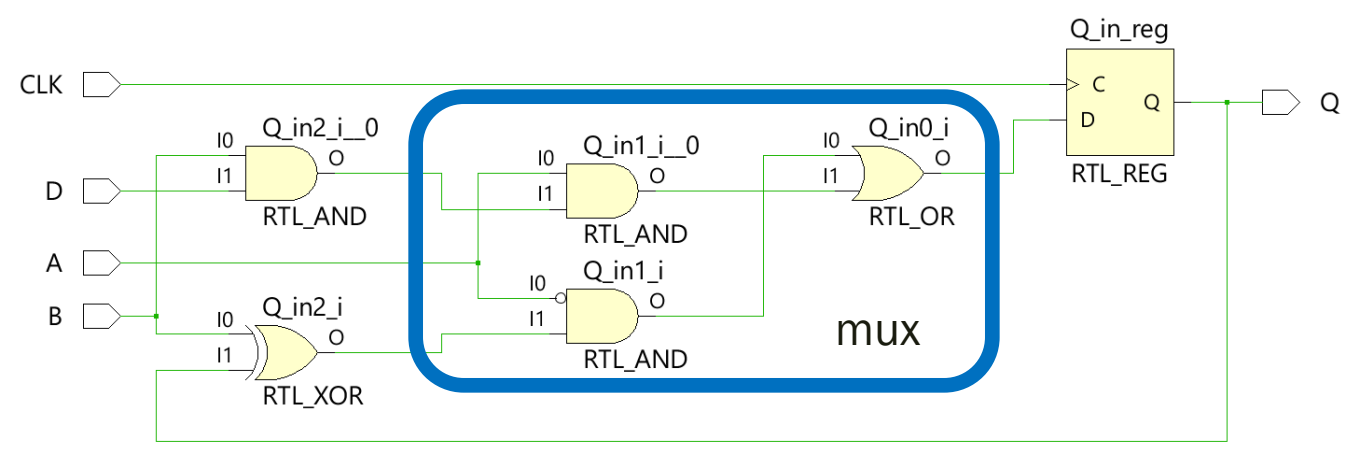
Τα A, B δεν τοποθετούνται στη λίστα ευαισθησίας

Η εξίσωση Boole του Flip-Flop περιγράφεται μετά τη συνθήκη του CLK

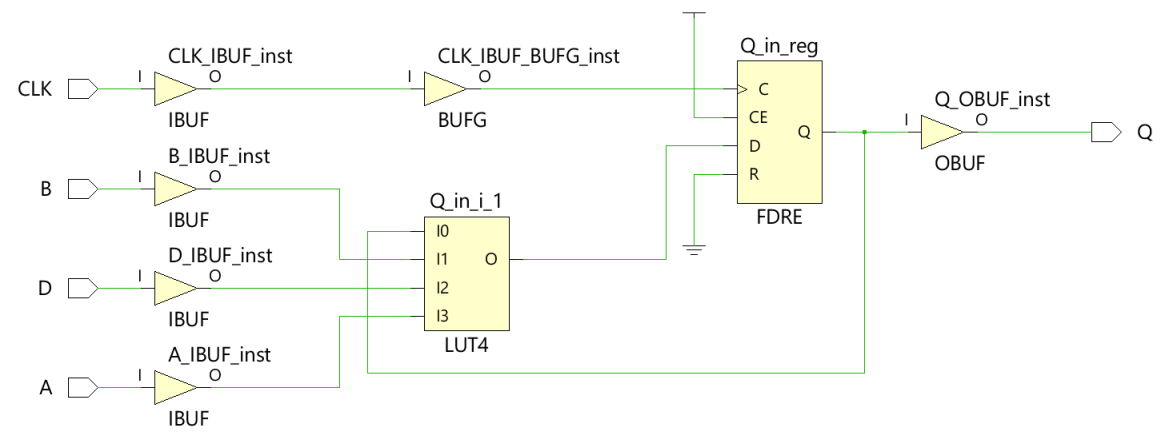
Ανάθεση τιμής στο Q εκτός process

AB Flip-Flop στη VHDL
Περιγραφή Συμπεριφοράς

- Σχηματικό διάγραμμα RTL – Υλοποιείται η εξίσωση Boole



- Σχηματικό διάγραμμα σε τεχνολογία FPGA – Το LUT4 υλοποιεί την εξίσωση Boole



Ασκήσεις Επανάληψης -1

```
entity LD_VHDL is
  Port (
        A, B: in STD_LOGIC;
        Y:    out STD_LOGIC);
end LD_VHDL;

architecture Behavioral of LD_VHDL is

signal Z: std_logic;

begin

B<=A;
Z:=A+Y;

end Behavioral;
```

Εντοπίστε τα λάθη (3)

Ασκήσεις Επανάληψης -1

```
entity LD_VHDL is
  Port (
        A, B: in STD_LOGIC;
        Y:    out STD_LOGIC);
end LD_VHDL;

architecture Behavioral of LD_VHDL is

signal Z: std_logic;

begin

B<=A;
Z:=A+Y;

end Behavioral;
```

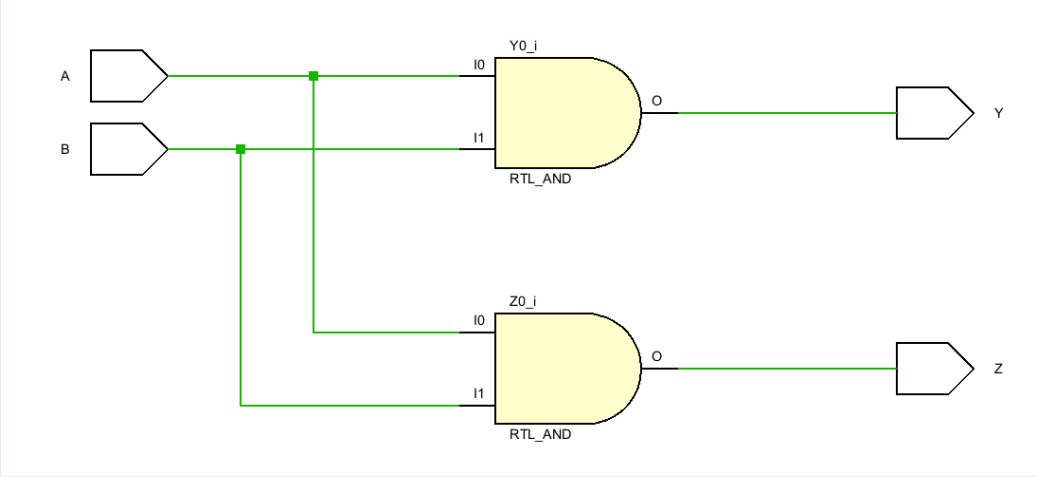
Εντοπίστε τα λάθη (3)

Ασκήσεις Επανάληψης -2

```
entity LD_VHDL is
  Port (
        A, B: in STD_LOGIC;
        Y, Z:    out STD_LOGIC);
end LD_VHDL;

architecture Behavioral of LD_VHDL is

signal C: std_logic;

begin

process (A) is
begin

Y<=A AND B;
C<=B;
Z<=A AND C;
end process;

end Behavioral;
```

Στο κύκλωμα εισέρχονται 3 ζεύγη των A, B ανά 20ns
A=‘0’, B=‘0’
A=‘1’, B=‘1’
A=‘1’, B=‘0’

Βρείτε πως μεταβάλλονται οι έξοδοι Y, Z και το σήμα C
Θεωρείστε ότι αρχικά ΟΛΑ τα σήματα είναι στην τιμή ‘0’



Ασκήσεις Επανάληψης -2

```
entity LD_VHDL is
  Port (
        A, B: in STD_LOGIC;
        Y, Z:    out STD_LOGIC);
end LD_VHDL;

architecture Behavioral of LD_VHDL is

signal C: std_logic;

begin

process (A) is
begin

Y<=A AND B;
C<=B;
Z<=A AND C;
end process;

end Behavioral;
```

Στο κύκλωμα εισέρχονται 3 ζεύγη των A, B ανά 20ns
A=‘0’, B=‘0’
A=‘1’, B=‘1’
A=‘1’, B=‘0’

Βρείτε πως μεταβάλλονται οι έξοδοι Y, Z και το σήμα C

A=‘0’, B=‘0’ => C=‘0’, Y=‘0’, Z=‘0’

Ασκήσεις Επανάληψης -2

```
entity LD_VHDL is
  Port (
        A, B: in STD_LOGIC;
        Y, Z:    out STD_LOGIC);
end LD_VHDL;

architecture Behavioral of LD_VHDL is

signal C: std_logic;

begin

process (A) is
begin

Y<=A AND B;
C<=B;
Z<=A AND C;
end process;

end Behavioral;
```

Στο κύκλωμα εισέρχονται 3 ζεύγη των A, B ανά 20ns
A=‘0’, B=‘0’
A=‘1’, B=‘1’
A=‘1’, B=‘0’

Βρείτε πως μεταβάλλονται οι έξοδοι Y, Z και το σήμα C

A=‘0’, B=‘0’ => C=‘0’, Y=‘0’, Z=‘0’
A=‘1’, B=‘1’ => C=‘1’, Y=‘1’, Z=‘0’

Ασκήσεις Επανάληψης -2

```
entity LD_VHDL is
  Port (
        A, B: in STD_LOGIC;
        Y, Z:    out STD_LOGIC);
end LD_VHDL;

architecture Behavioral of LD_VHDL is

signal C: std_logic;

begin

process (A) is
begin

Y<=A AND B;
C<=B;
Z<=A AND C;
end process;

end Behavioral;
```

Στο κύκλωμα εισέρχονται 3 ζεύγη των A, B ανά 20ns
A=‘0’, B=‘0’
A=‘1’, B=‘1’
A=‘1’, B=‘0’

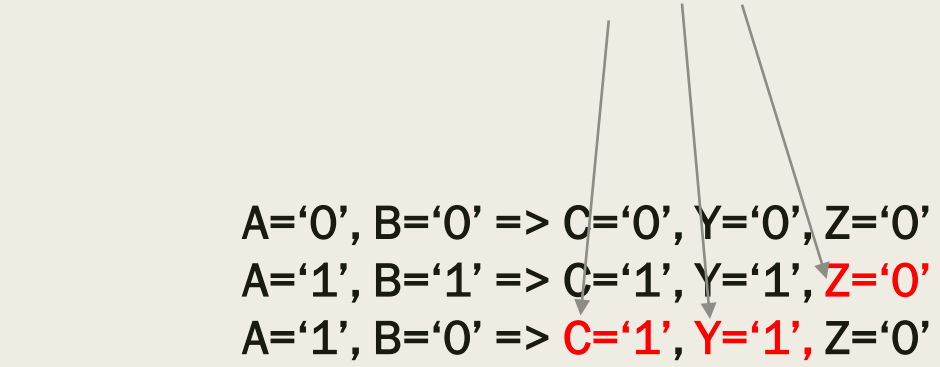
Βρείτε πως μεταβάλλονται οι έξοδοι Y, Z και το σήμα C

A=‘0’, B=‘0’ => C=‘0’, Y=‘0’, Z=‘0’
A=‘1’, B=‘1’ => C=‘1’, Y=‘1’, Z=‘0’
A=‘1’, B=‘0’ => C=‘1’, Y=‘1’, Z=‘0’

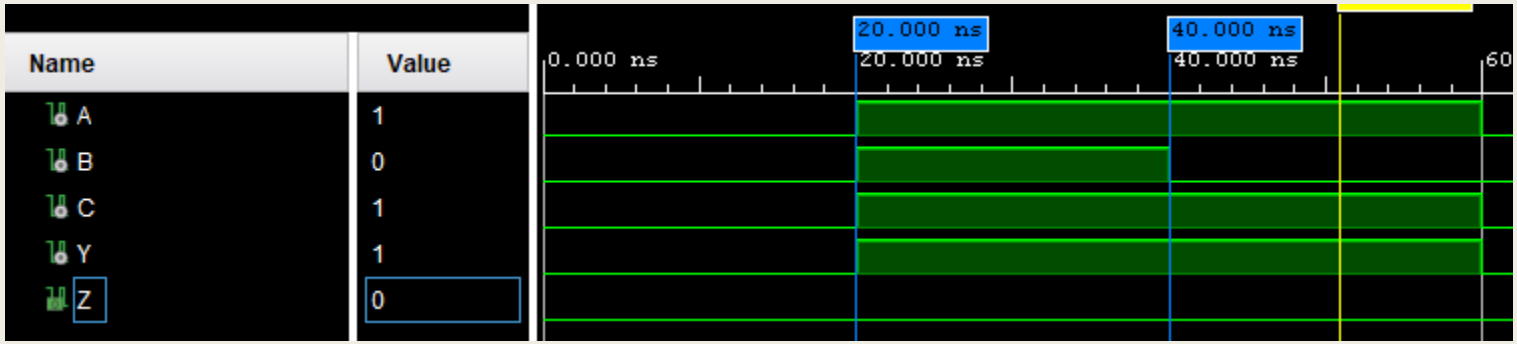
Ασκήσεις Επανάληψης -2

```
entity LD_VHDL is
  Port (
        A, B: in STD_LOGIC;
        Y, Z:    out STD_LOGIC);
end LD_VHDL;

architecture Behavioral of LD_VHDL is

signal C: std_logic;

begin

process (A) is
begin

Y<=A AND B;
C<=B;
Z<=A AND C;
end process;

end Behavioral;
```

Ήθελα αυτές τις τιμές?
Μάλλον όχι



Behavioral Simulation



Ασκήσεις Επανάληψης -2

```
entity LD_VHDL is
  Port (
        A, B: in STD_LOGIC;
        Y, Z:    out STD_LOGIC);
end LD_VHDL;

architecture Behavioral of LD_VHDL is

signal C: std_logic;

begin

process (A) is
begin

Y<=A AND B;
C<=B;
Z<=A AND C;
end process;

end Behavioral;
```

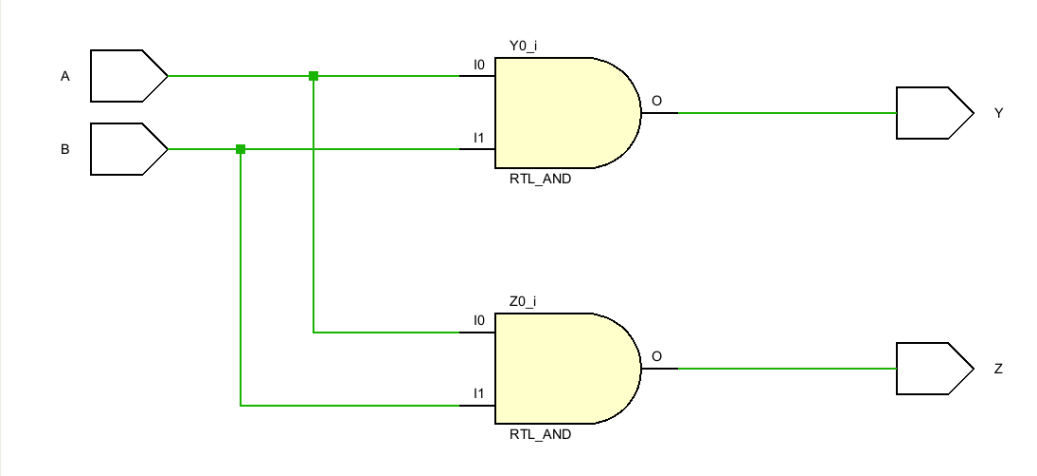
Τιμές που μάλλον ήθελα

A=‘0’, B=‘0’ => C=‘0’, Y=‘0’, Z=‘0’
A=‘1’, B=‘1’ => C=‘1’, Y=‘1’, Z=‘1’
A=‘1’, B=‘0’ => C=‘0’, Y=‘0’, Z=‘0’

Post Synthesis Functional Simulation

Ασκήσεις Επανάληψης -2

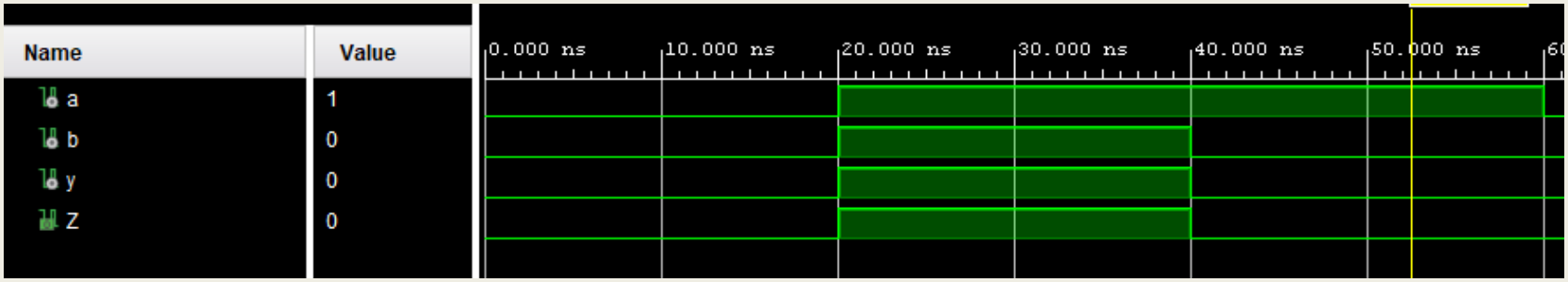
```
entity LD_VHDL is
  Port (
        A, B: in STD_LOGIC;
        Y, Z:    out STD_LOGIC);
end LD_VHDL;

architecture Behavioral of LD_VHDL is

signal C: std_logic;

begin

process (A,B,C) is
begin

Y<=A AND B;
C<=B;
Z<=A AND C;
end process;

end Behavioral;
```



Ασκήσεις Επανάληψης -2

```
entity LD_VHDL is
  Port (
        A, B: in STD_LOGIC;
        Y, Z:    out STD_LOGIC);
end LD_VHDL;

architecture Behavioral of LD_VHDL is

signal C: std_logic;

begin

process (A, B, C) is
begin

Y<=A AND B;
C<=B;
Z<=A AND C;
end process;

end Behavioral;
```

All Simulations



Ασκήσεις Επανάληψης -3

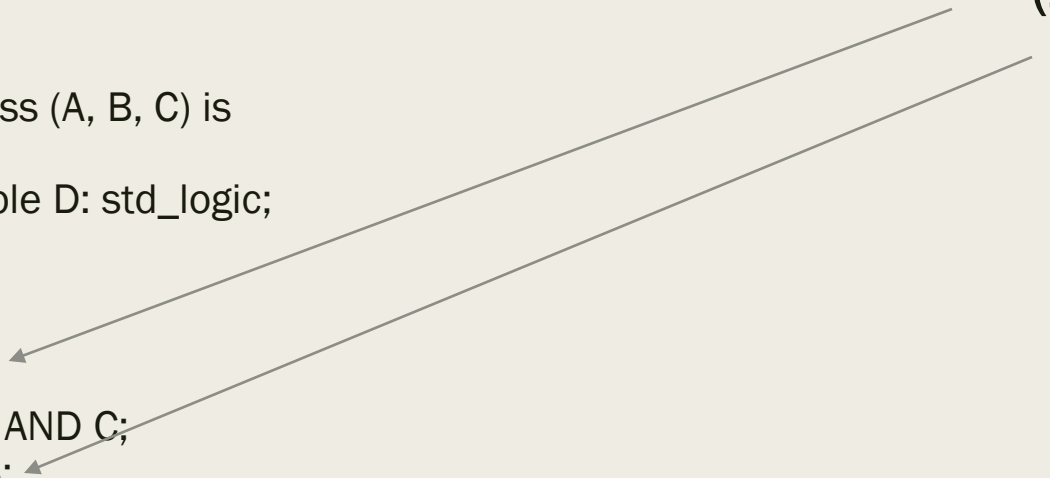
```
entity LD_VHDL is
  Port (
        A, B: in STD_LOGIC;
        Y, Z:    out STD_LOGIC);
end LD_VHDL;

architecture Behavioral of LD_VHDL is

signal C: std_logic;

begin

process (A, B, C) is

variable D: std_logic;
begin

C:=B;
Y<=A AND C;
D<=A;
Z<=A AND C;
end process;

end Behavioral;
```

Εντοπίστε τα λάθη

Ασκήσεις Επανάληψης -3

```
entity LD_VHDL is
  Port (
        A, B: in STD_LOGIC;
        Y, Z:    out STD_LOGIC);
end LD_VHDL;

architecture Behavioral of LD_VHDL is

signal C: std_logic;

begin

process (A, B, C) is

variable D: std_logic;

begin

C:=B;
Y<=A AND C;
D<=B;
Z<=A AND D;
end process;

end Behavioral;
```

Εντοπίστε τα λάθη (2)

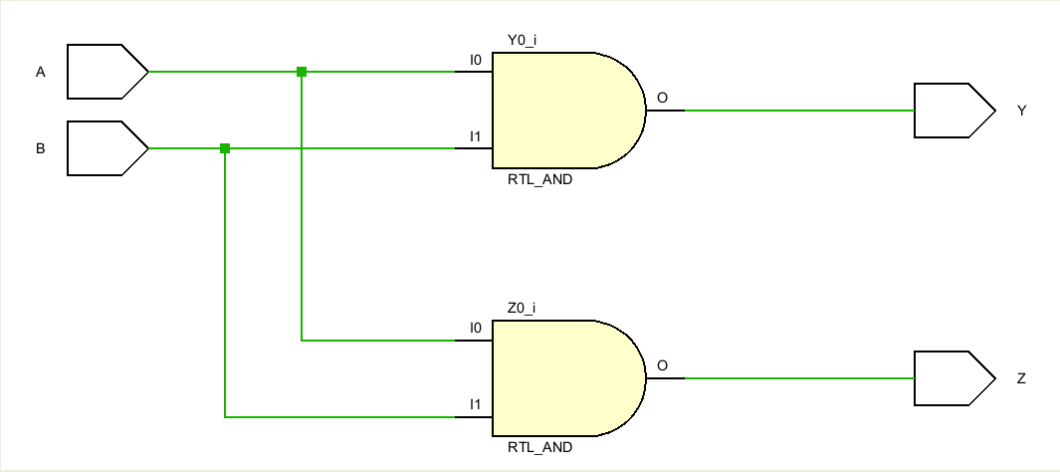


Ασκήσεις Επανάληψης -3

```
entity LD_VHDL is
  Port (
        A, B: in STD_LOGIC;
        Y, Z:    out STD_LOGIC);
end LD_VHDL;

architecture Behavioral of LD_VHDL is

signal C: std_logic;

begin

process (A, B, C) is

variable D: std_logic;

begin

C<=B;
Y<=A AND C;
D:=B;
Z<=A AND D;
end process;

end Behavioral;
```

Στο κύκλωμα εισέρχονται 3 ζεύγη των A, B ανά 20ns
A=‘1’, B=‘1’
A=‘0’, B=‘0’
A=‘1’, B=‘0’

Βρείτε πως μεταβάλλονται οι έξοδοι Y, Z, το σήμα C και η μεταβλητή D
Θεωρείστε ότι αρχικά ΟΛΑ τα σήματα είναι στην τιμή ‘0’



Ασκήσεις Επανάληψης -3

```
entity LD_VHDL is
  Port (
       A, B: in STD_LOGIC;
       Y, Z:    out STD_LOGIC);
end LD_VHDL;

architecture Behavioral of LD_VHDL is

signal C: std_logic;

begin

process (A, B, C) is

variable D: std_logic;

begin

C<=B;
Y<=A AND C;
D:=B;
Z<=A AND D;
end process;

end Behavioral;
```

Στο κύκλωμα εισέρχονται 3 ζεύγη των A, B ανά 20ns
A=‘1’, B=‘1’
A=‘0’, B=‘0’
A=‘1’, B=‘0’

Βρείτε πως μεταβάλλονται οι έξοδοι Y, Z, το σήμα C και η μεταβλητή D
Αρχικά A=B=C=D=‘0’

A=‘1’, B=‘1’ => C=‘1’, D=‘1’,Y=‘0’, Z=‘1’
Επειδή όμως άλλαξε τιμή το C (από ‘0’->’1’) το process εκτελείται ξανά και έχουμε:
A=‘1’, B=‘1’ => C=‘1’, D=‘1’,Y=‘1’, Z=‘1’

Ασκήσεις Επανάληψης -3

```
entity LD_VHDL is
  Port (
       A, B: in STD_LOGIC;
       Y, Z:    out STD_LOGIC);
end LD_VHDL;

architecture Behavioral of LD_VHDL is

signal C: std_logic;

begin

process (A, B, C) is

variable D: std_logic;

begin

C<=B;
Y<=A AND C;
D:=B;
Z<=A AND D;
end process;

end Behavioral;
```

Στο κύκλωμα εισέρχονται 3 ζεύγη των A, B ανά 20ns
A=‘1’, B=‘1’
A=‘0’, B=‘0’
A=‘1’, B=‘0’

Βρείτε πως μεταβάλλονται οι έξοδοι Y, Z, το σήμα C και η μεταβλητή D
Αρχικά A=B=C=D=‘0’

A=‘1’, B=‘1’ => C=‘1’, D=‘1’,Y=‘1’, Z=‘1’
A=‘0’, B=‘0’ => C=‘0’, D=‘0’,Y=‘0’, Z=‘0’
(διαδικασία όπως προηγουμένως)

Ασκήσεις Επανάληψης -3

```
entity LD_VHDL is
  Port (
       A, B: in STD_LOGIC;
       Y, Z:    out STD_LOGIC);
end LD_VHDL;

architecture Behavioral of LD_VHDL is

signal C: std_logic;

begin

process (A, B, C) is

variable D: std_logic;

begin

C<=B;
Y<=A AND C;
D:=B;
Z<=A AND D;
end process;

end Behavioral;
```

Στο κύκλωμα εισέρχονται 3 ζεύγη των A, B ανά 20ns
A=‘1’, B=‘1’
A=‘0’, B=‘0’
A=‘1’, B=‘0’

Βρείτε πως μεταβάλλονται οι έξοδοι Y, Z, το σήμα C και η μεταβλητή D
Αρχικά A=B=C=D=‘0’

A=‘1’, B=‘1’ => C=‘1’, D=‘1’,Y=‘1’, Z=‘1’
A=‘0’, B=‘0’ => C=‘0’, D=‘0’,Y=‘0’, Z=‘0’
A=‘1’, B=‘0’ => C=‘0’, D=‘0’,Y=‘0’, Z=‘0’