dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

Εργαστήριο Λογικής Σχεδίασης

Εισαγωγή

Βασιλόπουλος Διονύσης

ΕΤΕΠ Τμήματος Πληροφορικής & Τηλεπικοινωνιών

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Αντικείμενο Μαθήματος

Πραγματικές Ανάγκες

Ψηφιακά Συστήματα

Αντικείμενο Μαθήματος

- Πρακτικό μέρος του μαθήματος της Λογικής Σχεδίασης (ΛΣ)
- Συνοπτική επανάληψη θεωρίας ΛΣ
- Προγραμματισμός στη γλώσσα VHDL
- Περιγράφουμε (σε VHDL) ένα Σύστημα Λογικού Κυκλώματος/Ψηφιακό Κύκλωμα
- Με τι προγραμματίζουμε; Με το εργαλείο Vivado
- Τι προγραμματίζουμε; Hardware (κάρτες FPGA)
- Τι κάνει το Λογικό Κύκλωμα; Δίνει λύση σε μία πραγματική ανάγκη

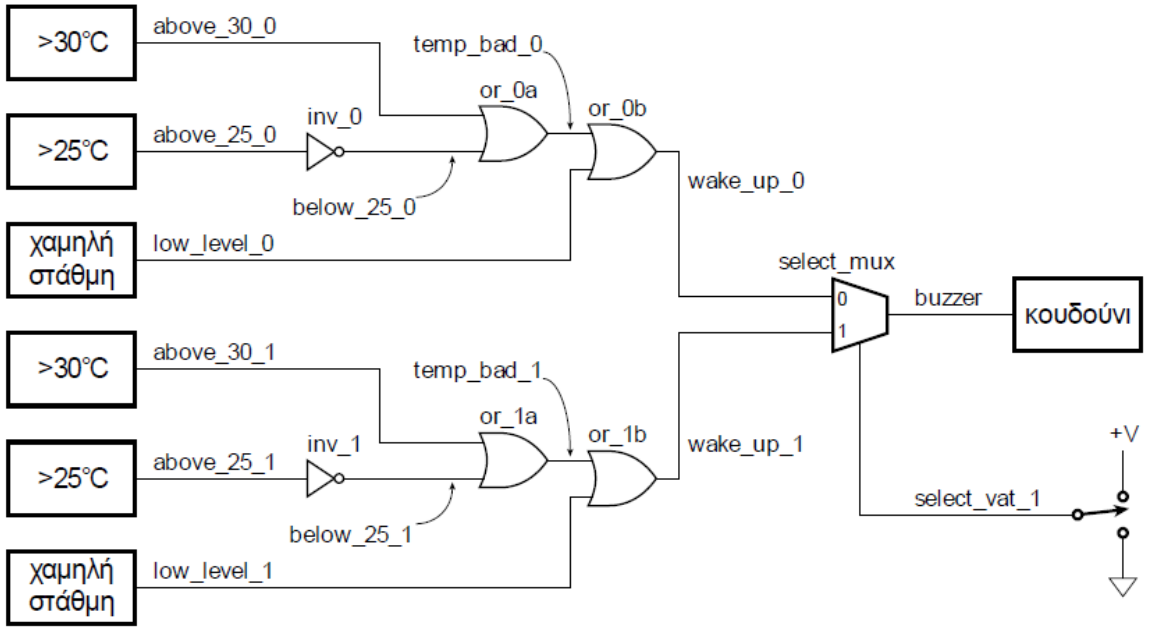
Αντικείμενο Μαθήματος

Εργαστήριο Λογικής Σχεδίασης – Περιεχόμενο μαθήματος

- Βασικές Αρχές της **Γλώσσας Προγραμματισμού VHDL**
- **Ανάπτυξη απλών εφαρμογών** για κατανόηση του μοντέλου της VHDL
- **Χρήση του εργαλείου VIVADO** για την ανάπτυξη εφαρμογών
- **Βασικά στοιχεία της θεωρίας Λογικής Σχεδίασης** (δυαδικό σύστημα αρίθμησης, λογικές πύλες, συνδυαστικά κυκλώματα, ακολουθιακά κυκλώματα)
  - ΔΕΝ θα γίνει ανάπτυξη στα θέματα της θεωρίας της Λογικής Σχεδίασης, παρά μόνο συνοπτικά και ΜΟΝΟ όσα χρειάζονται για την ανάπτυξη των εφαρμογών σε VHDL.
- **Προγραμματισμός κάρτας FPGA** (μέσω VHDL+VIVADO) σε πραγματικές συνθήκες (σε **ΔΙΑ ΖΩΣΗΣ** εκπαίδευση)

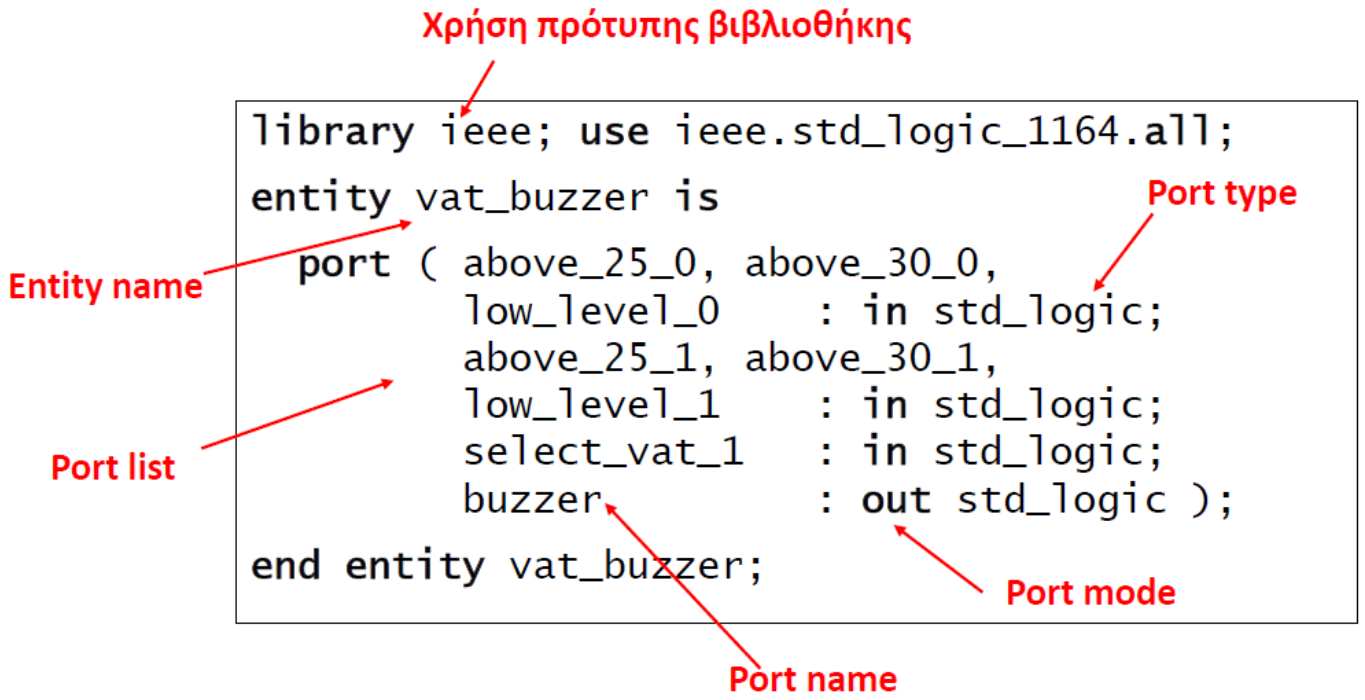
Αντικείμενο Μαθήματος

Λογικά κυκλώματα



Αντικείμενο Μαθήματος

Προγραμματισμός Υλικού σε VHDL



Αντικείμενο Μαθήματος

Εργαστήριο Λογικής Σχεδίασης – Οργάνωση μαθήματος 1/4

- Διδάσκων: Διονύσης Βασιλόπουλος (denis@di.uoa.gr) – Γραφείο: Α33
- 3 διαλέξεις εισαγωγικές και 5 σετ Θεωρίας/εργαστηρίου (θεωρία/παράδοση-χρήση εργαλείου Vivado, επίλυση άσκησης-προγραμματισμός κάρτας) (αρχικός προγραμματισμός μαθήματος).
- Σύνολο 8-9 διαλέξεις στο Αμφιθέατρο Α2 και 4-5 μαθήματα στο Εργαστήριο Ψηφιακής Σχεδίασης & ΗΥ Υψηλών Επιδόσεων (Αναγνωστήριο)

Αντικείμενο Μαθήματος

Εργαστήριο Λογικής Σχεδίασης – Οργάνωση μαθήματος 2/4

Τελικός βαθμός: Ασκήσεις Λογικής Σχεδίασης και VHDL

- **Εξάμηνα φοίτησης: 1, 2, 3, 4 (Α)**
  - ✓ Τρεις (3) ασκήσεις(20%, 30%, 40%) (90% του τελικού βαθμού)
  - ✓ Παρουσίες (τουλάχιστον 6 διαλέξεις και 3 εργαστήρια) (10% του τελικού βαθμού)
- **Εξάμηνα φοίτησης: 5, 6, 7, 8 (Β)**
  - ✓ Τρεις (3) ασκήσεις (20%, 40%, 40%)

  ή
  - ✓ (Α)

  Τελικός βαθμός το μέγιστο εκ των 2
- **Εξάμηνα φοίτησης: Επί πτυχίω (εξάμηνο φοίτησης>=9) (Γ)**
  - ✓ Α ή Β

  ή
  - ✓ Δύο (2) πρώτες ασκήσεις του Α και μία μικρή εργασία

  Τελικός βαθμός το μέγιστο εκ των 3

Η βαθμολογία σας στην 3η άσκηση θα πρέπει να είναι τουλάχιστον 30%

Αντικείμενο Μαθήματος

Εργαστήριο Λογικής Σχεδίασης – Οργάνωση μαθήματος 3/4

https://eclass.uoa.gr/ (eclass)

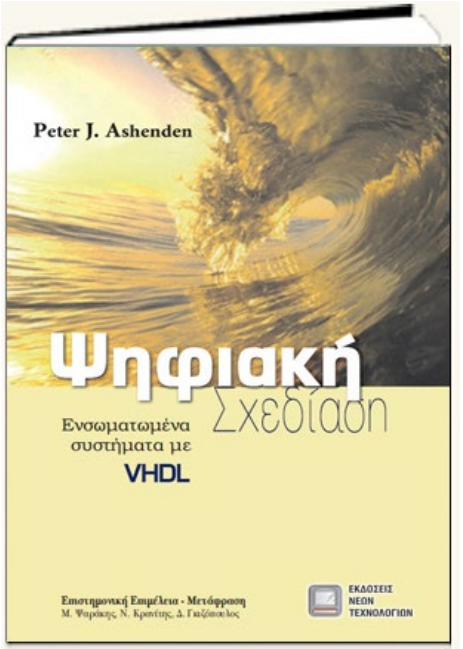
- Ανέβασμα διαφανειών/υλικού και ασκήσεων
- Επικοινωνία μέσω eclass
- Παράδοση εργασιών
- Παρουσίες (**eclass: Ερωτηματολόγιο ή email προς denis@di.uoa.gr**)
- Εργαλείο λογισμικού για το μάθημα: VIVADO
  - Στο eclass θα ανέβει αρχείο που περιγράφει τον τρόπο με τον οποίο θα εγκαταστήσουμε το εργαλείο στον υπολογιστή μας (απαραίτητο).
  - Υπάρχει έκδοση για Windows και Linux (όχι για Mac)

Αντικείμενο Μαθήματος

Εργαστήριο Λογικής Σχεδίασης – Οργάνωση μαθήματος 4/4

Προτεινόμενη Βιβλιογραφία

1. **Ψηφιακή Σχεδίαση. Ενσωματωμένα Συστήματα με VHDL**, Peter J. Ashenden, Επιστημονική Επιμέλεια – Μετάφραση: Μ. Ψαράκης, Ν. Κρανίτης, Δ. Γκιζόπουλος, Εκδόσεις Νέων Τεχνολογιών, 2010
2. Ψηφιακή Σχεδίαση και Αρχιτεκτονική Υπολογιστών, S.L.Harris, D.M.Harris, Εκδόσεις ARM®, 2019



Ψηφιακά Συστήματα

Ψηφιακή Σχεδίαση

- **Ψηφιακή** (*Digital):* κυκλώματα που χρησιμοποιούν δύο επίπεδα τάσης (π.χ. 0V, +5V) για αναπαράσταση της πληροφορίας
  - Λογική *(Logic):* χρήση τιμών αληθείας (0/1, true/false) και κανόνων λογικής (άλγεβρα boole) για την ανάλυση των κυκλωμάτων
- **Σχεδίαση** *(Design):* ανταποκρίνονται σε λειτουργικές απαιτήσεις ενώ ταυτόχρονα ικανοποιούν περιορισμούς
  - Περιορισμοί: απόδοση, μέγεθος (κόστος), ισχύς, κλπ.

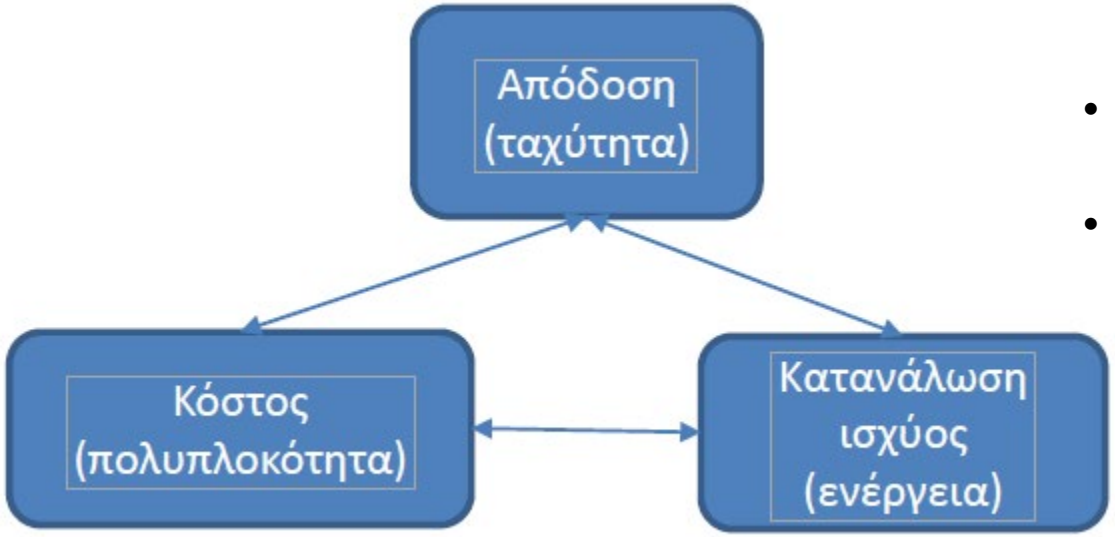
Ψηφιακά Συστήματα

Ψηφιακή Σχεδίαση

- **Δυαδικό Σύστημα Αρίθμησης** (0,1)
- **Λογικοί κανόνες** (Άλγεβρα Boole: ΝΑΙ/ΟΧΙ, ΙΣΧΥΕΙ/ΔΕΝ ΙΣΧΥΕΙ)
- **Πύλες** (Στοιχειώδη κυκλώματα, υλοποιούν την άλγεβρα boole σε επίπεδο υλικού, κάνουμε στοιχειώδεις πράξεις στο δυαδικό σύστημα. Στη βάση τους υπάρχει το Περνάει Ρεύμα/Δεν περνάει ρεύμα ή Υπάρχει Τάση/Γείωση)
- **Συνδυαστικά κυκλώματα** (π.χ. ένα κύκλωμα που προσθέτει δύο δυαδικούς αριθμούς). Έχει εισόδους που παράγουν κάποιες τιμές εξόδων.
- **Ακολουθιακά κυκλώματα**. Η τιμή του εξόδου εξαρτάται από τις εισόδους αλλά και από την προηγούμενη τιμή της εξόδου. Συνήθως υπάρχει συγχρονισμός στο πότε αλλάζουν οι τιμές στην είσοδο και στο πότε ελέγχουμε την τιμή στην έξοδο. Παράδειγμα είναι το χρονόμετρο.

Ψηφιακά Συστήματα

Ψηφιακή **Σχεδίαση**



Συμβιβασμοί

- Μπορούμε να βελτιώσουμε το ένα εις βάρους ενός άλλου ή και των δύο άλλων
- Αυτοί οι συμβιβασμοί υπάρχουν σε κάθε επίπεδο στη σχεδίαση του συστήματος

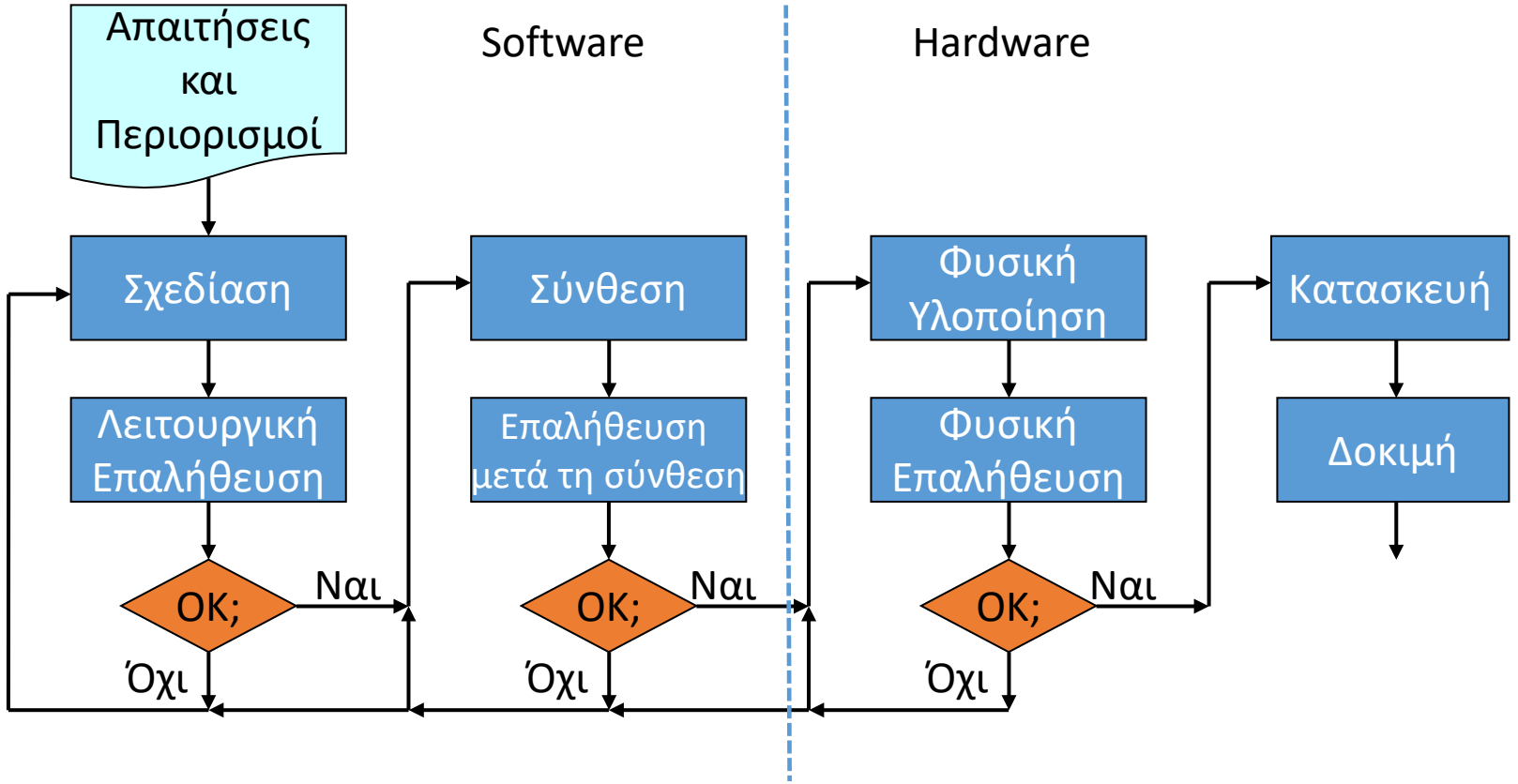
Ψηφιακά Συστήματα

Ψηφιακή Σχεδίαση – Μεθοδολογία Σχεδίασης

- Τα απλά συστήματα μπορούν να σχεδιαστούν από ένα άτομο χρησιμοποιώντας ειδικές μεθόδους
- Τα πραγματικά συστήματα σχεδιάζονται από ομάδες
  - Απαιτούν μια συστηματική μεθοδολογία σχεδίασης
- Καθορίζει
  - Τις εργασίες που αναλαμβάνουμε
  - Την πληροφορία που απαιτείται και παράγεται
  - Τις σχέσεις μεταξύ των εργασιών
    - εξαρτήσεις, καθορισμός των ακολουθιών
  - Τα εργαλεία EDA (Electronic design automation) που χρησιμοποιούνται (αλλιώς και eCAD – electronic Computer Aided Design)

Ψηφιακά Συστήματα

Ψηφιακή Σχεδίαση – Μια απλή μεθοδολογία ανάπτυξης



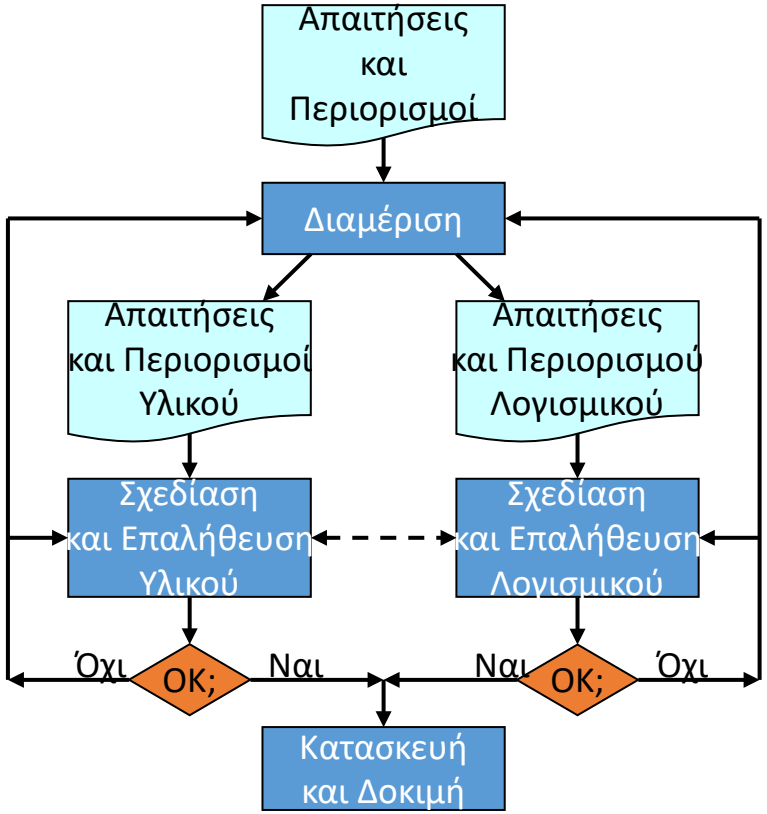
Ψηφιακά Συστήματα

Ψηφιακή Σχεδίαση – Ιεραρχική σχεδίαση

- Τα κυκλώματα είναι αρκετά πολύπλοκα για να σχεδιάσουμε όλες τις λεπτομέρειες με τη μία
- Σχεδιάζουμε υποσυστήματα για απλές λειτουργίες
- Συνθέτουμε το σύστημα από τα υποσυστήματα
  - Αντιμετωπίζουμε τα υποκυκλώματα ως «μαύρα κουτιά»
  - Επαληθεύουμε ανεξάρτητα, και έπειτα επαληθεύουμε τη σύνθεση
- Σχεδίαση top-down (από πάνω προς τα κάτω) ή bottom-up (από κάτω προς τα πάνω)

Ψηφιακά Συστήματα

Ψηφιακή Σχεδίαση – Μεθοδολογία Συσχεδίασης



Κάθε ομάδα υλοποιεί ένα ξεχωριστό Module του συστήματος

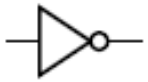
Ψηφιακά Συστήματα

Δυαδική Αναπαράσταση

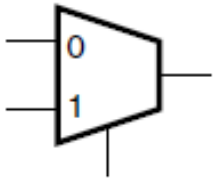
- Συστήματα που αναπαριστούν την πληροφορία με δύο τιμές (0,1 ή True,False)
- Βασικές ψηφιακές λογικές πύλες και πίνακες αληθείας

AND gate

OR gate



inverter



multiplexer

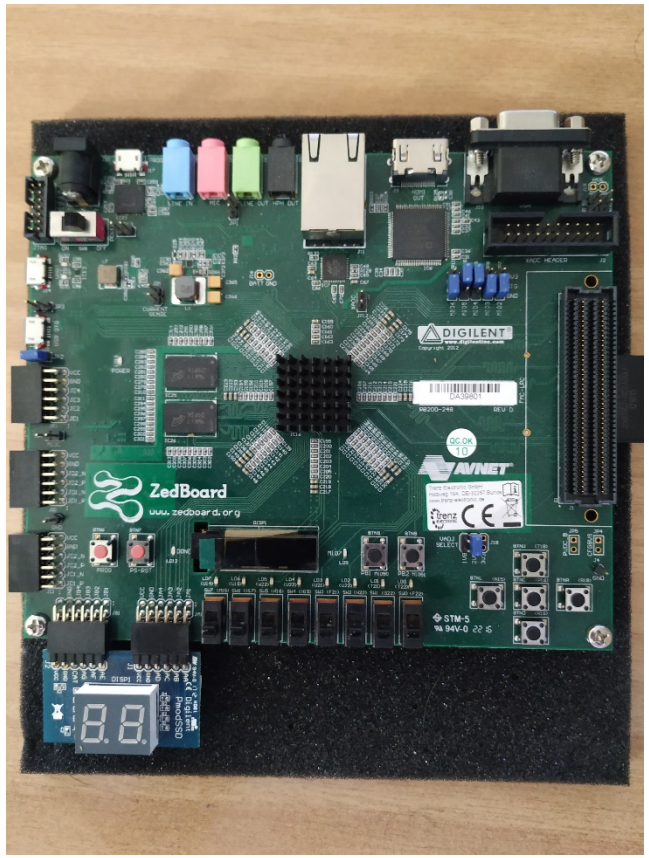
x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	$\overline{x}$
0	1
1	0

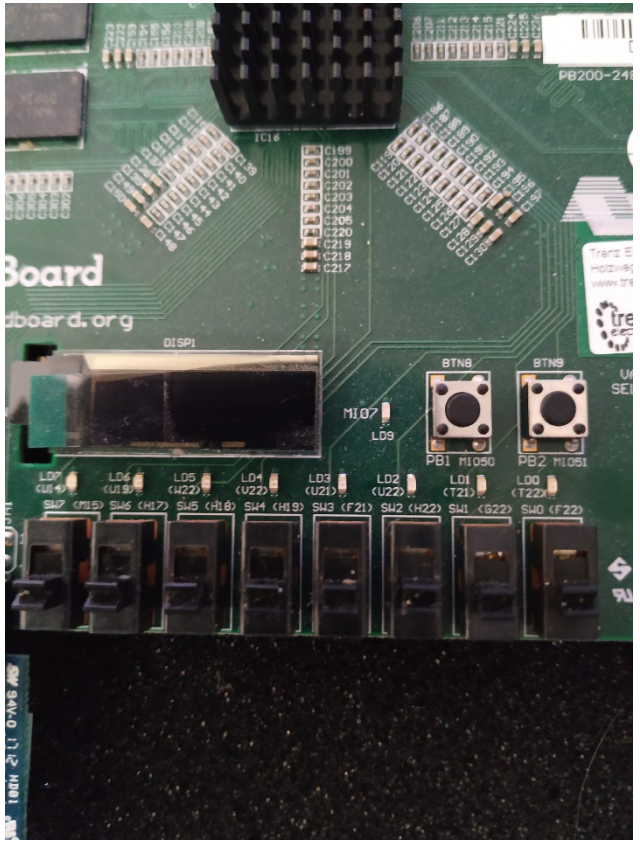
VHDL - VIVADO

Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – FPGA



VHDL - VIVADO

Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – FPGA



Πρωτοετείς

- Εγγραφή στο eclass.uoa.gr
- Χρήση του e-mail σας (sdixx00xxx@di.uoa.gr)
- my-studies (δηλώσεις/βαθμοί)
- Εργαστήρια linux/windows και κωδικοί πρόσβασης

Περίληψη

- Εισαγωγή στο μάθημα «Εργαστήριο Λογικής Σχεδίασης»
- Μεθοδολογία σχεδίασης
- Διαβάζετε το κεφάλαιο 1 (εκτός του 1.3) από το βιβλίο του Ashenden.
- Διαβάζετε τις παραγράφους 1.1, 1.2, 1.3, 1.4.1 – 1.4.5. από το βιβλίο των Harris.

dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

Εργαστήριο Λογικής Σχεδίασης

Εισαγωγή στη VHDL και το εργαλείο Vivado

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Ψηφιακά Συστήματα

Ψηφιακό κύκλωμα – Φυσική υλοποίηση

- Τεχνολογίες υλοποίησης
  - Application-specific ICs (ASICs): Ολοκληρωμένα κυκλώματα εξειδικευμένα για εφαρμογές (δεν προγραμματίζονται)
  - Field-programmable gate arrays (FPGAs): Επιτόπου προγραμματιζόμενοι πίνακες πυλών
- Αντιστοίχιση(mapping): καθορίζει τους πόρους για κάθε υποσύστημα
- Τοποθέτηση (placement): διευθετεί τις πύλες μέσα στα υποσυστήματα
- Δρομολόγηση (routing): ενώνει τις πύλες με αγωγούς
- Φυσική επαλήθευση (physical verification)
  - Το φυσικό κύκλωμα συνεχίζει να ικανοποιεί τους περιορισμούς
  - Χρησιμοποιεί καλύτερες εκτιμήσεις των καθυστερήσεων

Ψηφιακά Συστήματα

Ψηφιακό κύκλωμα – Φυσική υλοποίηση

- Ένας ενσωματωμένος επεξεργαστής είναι ένας επεξεργαστής κρυμμένος σε μία συσκευή, μαζί και με άλλα ηλεκτρονικά ή ηλεκτρομηχανικά μέρη, σχεδιασμένος για να κάνει μια ή και περισσότερες λειτουργίες πραγματικού χρόνου. Προγραμματίζεται σε ορισμένες λειτουργίες.
- Οι ενσωματωμένοι επεξεργαστές εφαρμόζονται σήμερα εκτενώς σε διάφορες συσκευές ήχου, εικόνας, σε κάμερες, σε ηλεκτρονικά παιχνίδια, σε PDAs, σε υπολογιστές τσέπης, σε περιφερειακά συστήματα γενικής χρήσης όπως modem, κάρτες video, στα τηλέφωνα και στα δίκτυα.
- Κάθε μικροεπεξεργαστής τέτοιου είδους έχει ένα σχετικά μικρό σύνολο εντολών, που εκτελούν λειτουργίες όπως αριθμητικές πράξεις και μεταφορά δεδομένων σε καταχωρητές και μνήμη και συναντώνται σε ενσωματωμένα συστήματα

Ψηφιακά Συστήματα

Κατηγορίες

- **Συνδυαστικά** κυκλώματα: Οι τιμές των εξόδων εξαρτώνται μόνο από τις τιμές των εισόδων του συστήματος
- **Ακολουθιακά** κυκλώματα: Οι τιμές των εξόδων εξαρτώνται τόσο από τις τιμές των εισόδων του συστήματος όσο και από τις προηγούμενες τιμές των εξόδων (έχουμε ανάδραση).
  - **Σύγχρονα**: Η συμπεριφορά τους ορίζεται από τις τιμές των εξόδων σε διακριτές στιγμές του χρόνου. Υπάρχει σήμα συγχρονισμού (ρολόι/clock-CLK)
  - **Ασύγχρονα**: Οι τιμές των εξόδων αλλάζουν ανά πάσα στιγμή.

HDL

- Hardware Description Language (HDL)
  - Μια γλώσσα για την μοντελοποίηση της συμπεριφοράς και της δομής των ψηφιακών συστημάτων
- Electronic Design Automation (EDA) using HDL
  - Σχεδίαση ηλεκτρονικών κυκλωμάτων με χρήση εργαλείων CAD (computer-aided design)
  - Εισαγωγή σχεδίασης (design entry)
    - Χρήση κώδικα αντί για σχηματικά διαγράμματα
  - Επαλήθευση(verification)
    - Προσομοίωση (simulation) του κώδικα
  - Σύνθεση (synthesis)
    - Αυτόματη παραγωγή των κυκλωμάτων
  - Φυσική υλοποίηση (implementation)
    - Υλοποίηση του κυκλώματος στην τεχνολογία επιλογής

HDL-VHDL

- VHSIC Hardware Description Language (Γλώσσα Περιγραφής Υλικού)
  - VHSIC: Very High-Speed Integrated Circuits
- Ιστορική αναδρομή:
  - Ξεκίνησε το 1981 από το Υπουργείο Άμυνας των ΗΠΑ ως γλώσσα περιγραφής ολοκληρωμένων κυκλωμάτων
  - Οι εταιρείες IBM, Texas Instruments, Intermetrics ανάπτυξαν και κυκλοφόρησαν την 1η έκδοση το 1985
- Πρότυπο από τον οργανισμό IEEE
  - IEEE Standard 1076-1987 (VHDL-87)
  - IEEE Standard 1076-1993 (VHDL-93)
  - IEEE Standard 1076-2000 and 1076 2002 (VHDL-2000, VHDL-2002)
  - IEEE Standard 1076-2008 (VHDL-2008)
- Πιο διαδεδομένη στην Ευρώπη
  - Στην Αμερική η Verilog είναι πιο διαδεδομένη
- Χρησιμοποιείται για την σχεδίαση συστημάτων για το διάστημα
  - Από την NASA και την ESA

HDL

- Επειδή οι HDLs έχουν τις ρίζες τους σε γλώσσες προγραμματισμού (η VHDL στην Ada και η Verilog στην C) είναι εύκολες στην εκμάθηση
  - .. αλλά δύσκολες στη σωστή χρήση τους!
- Οι αρχάριοι τείνουν να γράφουν κώδικα VHDL που μοιάζει με τα προγράμματα υπολογιστών
  - ... πολλές μεταβλητές και πολλούς βρόχους ...
- Για αυτό:
  - Μη γράφετε VHDL όπως θα γράφατε ένα πρόγραμμα
  - Θυμηθείτε τις δυνατότητες που σας δίνει η VHDL (π.χ. παράλληλη εκτέλεση, περιγραφή χρονισμών, περιγραφή ακολουθίας γεγονότων)
  - Γράφετε VHDL για σύνθεση στο υλικό
  - Να έχετε στο μυαλό σας τι κύκλωμα αντιστοιχεί στον κώδικα VHDL που γράφετε

HDL

Πλεονεκτήματα των HDLs

- Υπερτερούν από τα σχηματικά διαγράμματα
  - Η μοντελοποίηση του συστήματος μπορεί να γίνει σε όλα τα επίπεδα (από τα υψηλότερα ως τα χαμηλότερα)
  - Η περιγραφή σε HDL είναι συνήθως πιο κατανοητή από ένα σχηματικό διάγραμμα
  - Η περιγραφή σε HDL είναι ανεξάρτητη από τις βιβλιοθήκες σχεδίασης (design libraries) και τα εργαλεία CAD
- Υπερτερούν από τις γλώσσες προγραμματισμού
  - Παρέχουν δομές που περιγράφουν καλύτερα το υλικό
  - Παράλληλη εκτέλεση εντολών αντί για ακολουθιακή/σειριακή
  - Παρέχουν δυνατότητα για περιγραφή χρονισμών

HDL

Μοντελοποίηση και προσομοίωση

- Αρχικά οι HDLs σχεδιάστηκαν για τη μοντελοποίηση και τη προσομοίωση των συστημάτων υλικού στα υψηλότερα επίπεδα αφαίρεσης
- Χαρακτηριστικά μοντελοποίησης των HDLs:
  - παράλληλη εκτέλεση
  - ιεραρχική σχεδίαση
  - περιγραφή χρονισμών
  - περιγραφή ακολουθίας γεγονότων
  - περιγραφή σύγχρονης/ασύγχρονης συμπεριφοράς

VHDL - VIVADO

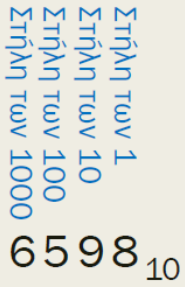
Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – Σύνθεση

- Συνήθως σχεδιάζουμε χρησιμοποιώντας HDL επιπέδου μεταφοράς καταχωρητή (Register Transfer Level - RTL)
  - υψηλότερο επίπεδο αφαίρεσης από τις πύλες
- Το εργαλείο σύνθεσης μεταφράζει το μοντέλο σε ένα κύκλωμα από πύλες που εκτελεί την ίδια λειτουργία
- Προσδιορίζουμε στο εργαλείο
  - την τεχνολογία υλοποίησης που στοχεύουμε
  - περιορισμούς στο χρονισμό, στην επιφάνεια, κλπ.
- Επαλήθευση μετά τη σύνθεση
  - το κύκλωμα που προέκυψε από τη σύνθεση ικανοποιεί τους περιορισμούς

Δυαδικό Σύστημα

Δυαδική Αναπαράσταση

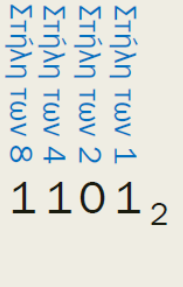
- Δεκαδικό σύστημα αναπαράστασης αριθμών (0-9)



6598_{10}

Κάθε στήλη ενός δεκαδικού αριθμού έχει δεκαπλάσιο βάρος από την προηγούμενη στήλη. Ξεκινώντας από τα δεξιά προς τα αριστερά τα βάρη είναι: 10^0, 10^1, 10^2, 10^3, ...

- Δυαδικό σύστημα αναπαράστασης αριθμών (0-1)



1101_2

Κάθε στήλη ενός δυαδικού αριθμού έχει διπλάσιο βάρος από την προηγούμενη στήλη. Ξεκινώντας από τα δεξιά προς τα αριστερά τα βάρη είναι: 2^0, 2^1, 2^2, 2^3, ...

Δυαδικό Σύστημα

Δυαδική Αναπαράσταση

- Δεκαδικό σύστημα αναπαράστασης αριθμών (0-9)

Στήλη των 1000 Στήλη των 100 Στήλη των 10 Στήλη των 1

$$6598_{10} = 6 \times 10^3 + 5 \times 10^2 + 9 \times 10^1 + 8 \times 10^0$$

Έξι Χιλιάδες — Πέντε Εκατοντάδες — Εννέα Δεκάδες — Οκτώ Μονάδες

- Δυαδικό σύστημα αναπαράστασης αριθμών (0-1)

Στήλη των 8 Στήλη των 4 Στήλη των 2 Στήλη των 1

$$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13_{10}$$

Μια Οκτάδα — Μια Τετράδα — Μηδέν Δυάδες — Μια Μονάδα

Δυαδικό Σύστημα

Αριθμοί χωρίς πρόσημο (Unsigned – Μόνο θετικοί)

Πρόσθεση

```
 0    0    1    1
+0   +1   +0   +1
---  ---  ---  ---
 0    1    1   10
```

Κρατούμενο/(C)arry)

```
 0    0    0    1
```

```
 00   01   11   11
 01  +01  +01  +11
----  ----  -----  -----
 01   10  100  110
```

Κρατούμενο/(C)arry)

```
 0    0    1    1
```

Δυαδικό Σύστημα

Αριθμοί χωρίς πρόσημο (Unsigned – Μόνο θετικοί)

Πολλαπλασιασμός/Διαίρεση με πολλαπλάσια του δύο

10 =2(δεκαδικό) (A)

Εφαρμόζουμε αριστερή ολίσθηση και γεμίζουμε δεξιά με το 0

10**0** =4(δεκαδικό) (B) = 2*(A)

10**00** =8(δεκαδικό) (C) = 2*(B)=4*(A)

1000 =8(δεκαδικό) (A)

Εφαρμόζουμε δεξιά ολίσθηση διαγράφοντας το πιο δεξί ψηφίο

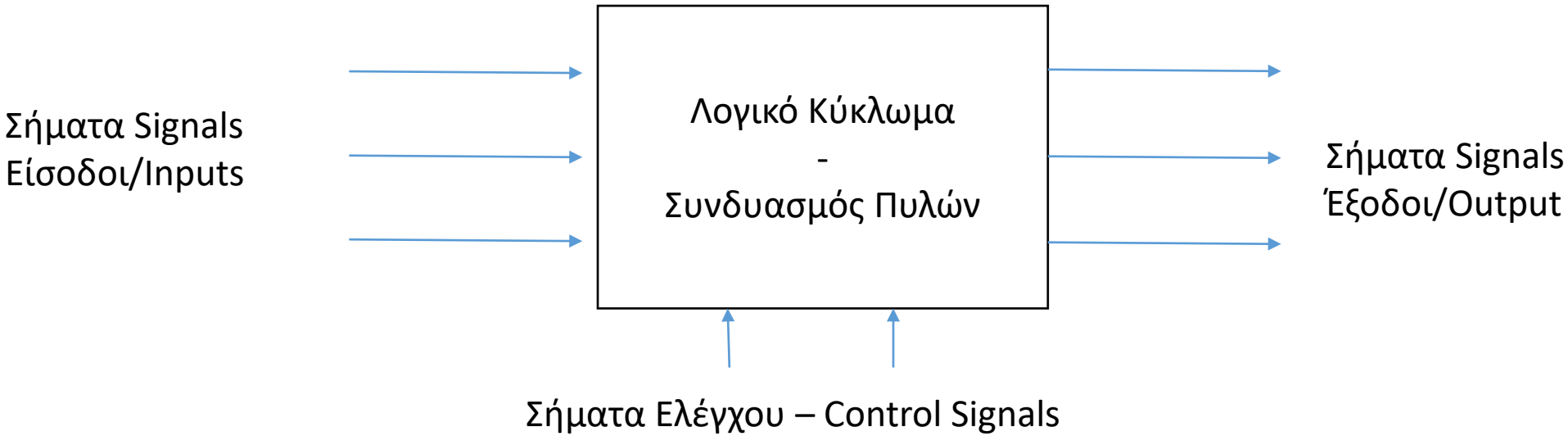
100 =4(δεκαδικό) (B) = (A)/2

10 =2(δεκαδικό) (C)= (B)/2=(A)/4

Λογικές Πύλες

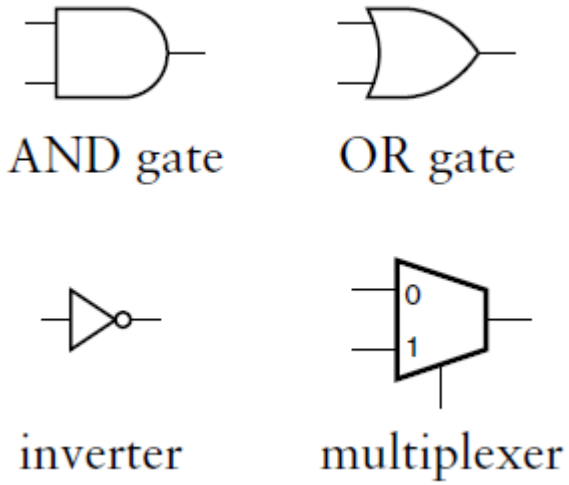
Όλοι οι είσοδοι και έξοδοι είναι σήματα με πιθανές τιμές 0 ή 1

Λογικές Πύλες



Λογικές Πύλες – Πίνακας Αληθείας

- Σε κάθε Πύλη ή Κύκλωμα αντιστοιχεί ένας Πίνακας Αληθείας (Truth Table).
- Ο Πίνακας Αληθείας καθορίζει την τιμή εξόδου για κάθε συνδυασμό εισόδων στην Πύλη ή το Κύκλωμα



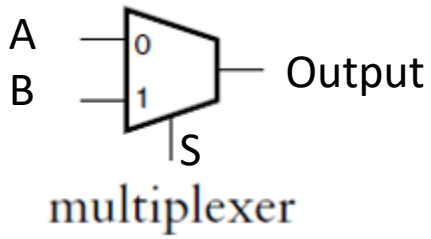
x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	$\bar{x}$
0	1
1	0

Λογικές Πύλες – Πίνακας Αληθείας

- Σε κάθε Πύλη ή Κύκλωμα αντιστοιχεί ένας Πίνακας Αληθείας (Truth Table).
- Ο Πίνακας Αληθείας καθορίζει την τιμή εξόδου για κάθε συνδυασμό εισόδων στην Πύλη ή το Κύκλωμα



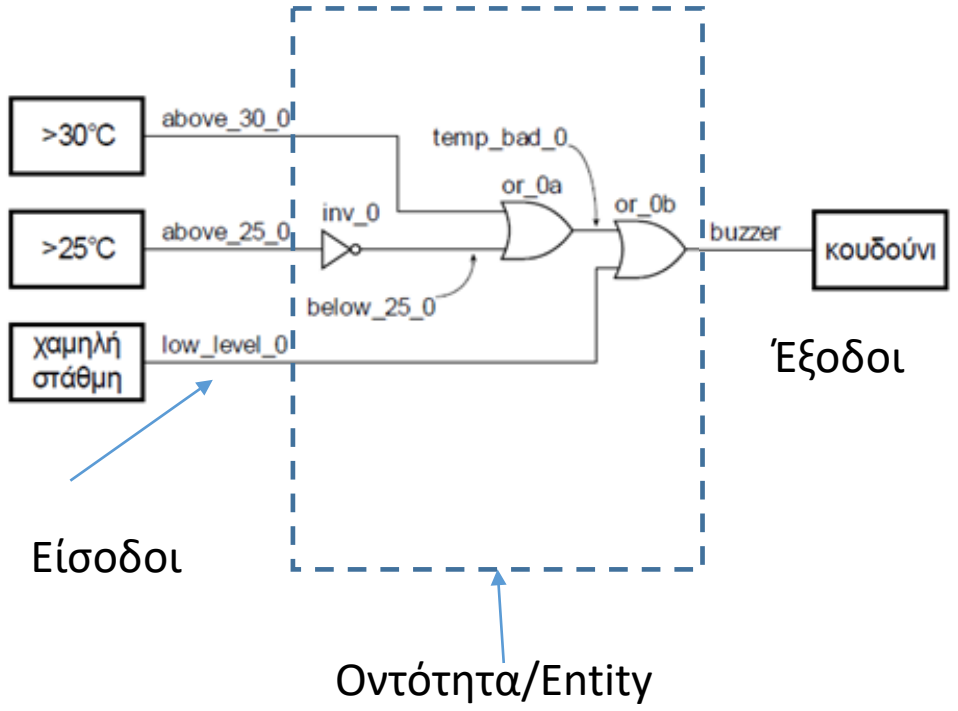
S	Output
0	A
1	B

Αν A=0, B=1, S=1, Output= ;
Αν A=1, B=0, S=1, Output= ;

VHDL - Vivado

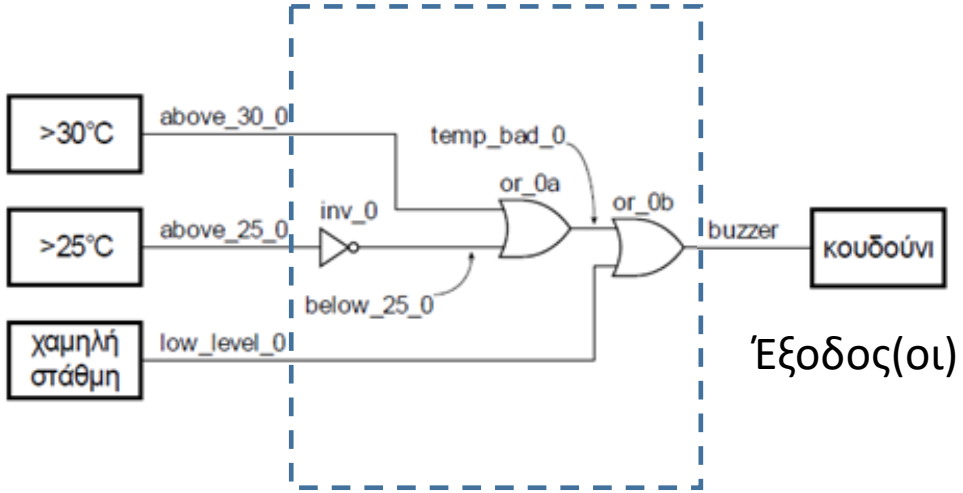
Ψηφιακό κύκλωμα - Παράδειγμα

Παράδειγμα: Εργοστάσιο έχει δοχείο επεξεργασίας υγρών. Το δοχείο πρέπει α)να έχει θερμοκρασία μεταξύ 25 και 30 βαθμών και β) η στάθμη του πρέπει να είναι πάνω από ένα επίπεδο. Σε περίπτωση που το α) ή το β) δεν ικανοποιούνται πρέπει να ενεργοποιηθεί ένα κουδούνι. Στη διάθεσή μας έχουμε αισθητήρες (θερμόμετρα) που δείχνουν αν η θερμοκρασία ξεπερνά ένα όριο (το οποίο ορίζεται από εμάς για κάθε αισθητήρα) και αισθητήρα που μας ενημερώνει εάν η στάθμη του δοχείου είναι κάτω από ένα επίπεδο ασφαλείας. Περιγράψτε το σύστημα.



VHDL - Vivado

Ψηφιακό κύκλωμα – VHDL: Entity (Input/Output PORTS)

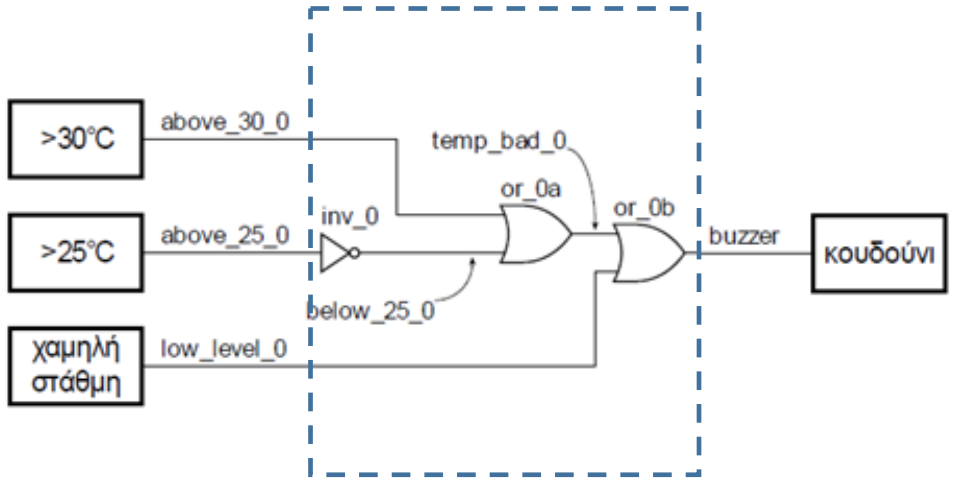



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity buzzer is

port (
above_25_0: in std_logic;
above_30_0: in std_logic;
low_level_0: in std_logic;
buzzer      : out std_logic );

end entity buzzer;
```

Περιγραφή Οντότητας

VHDL - Vivado

Ψηφιακό κύκλωμα – VHDL: Architecture (Behavioral)

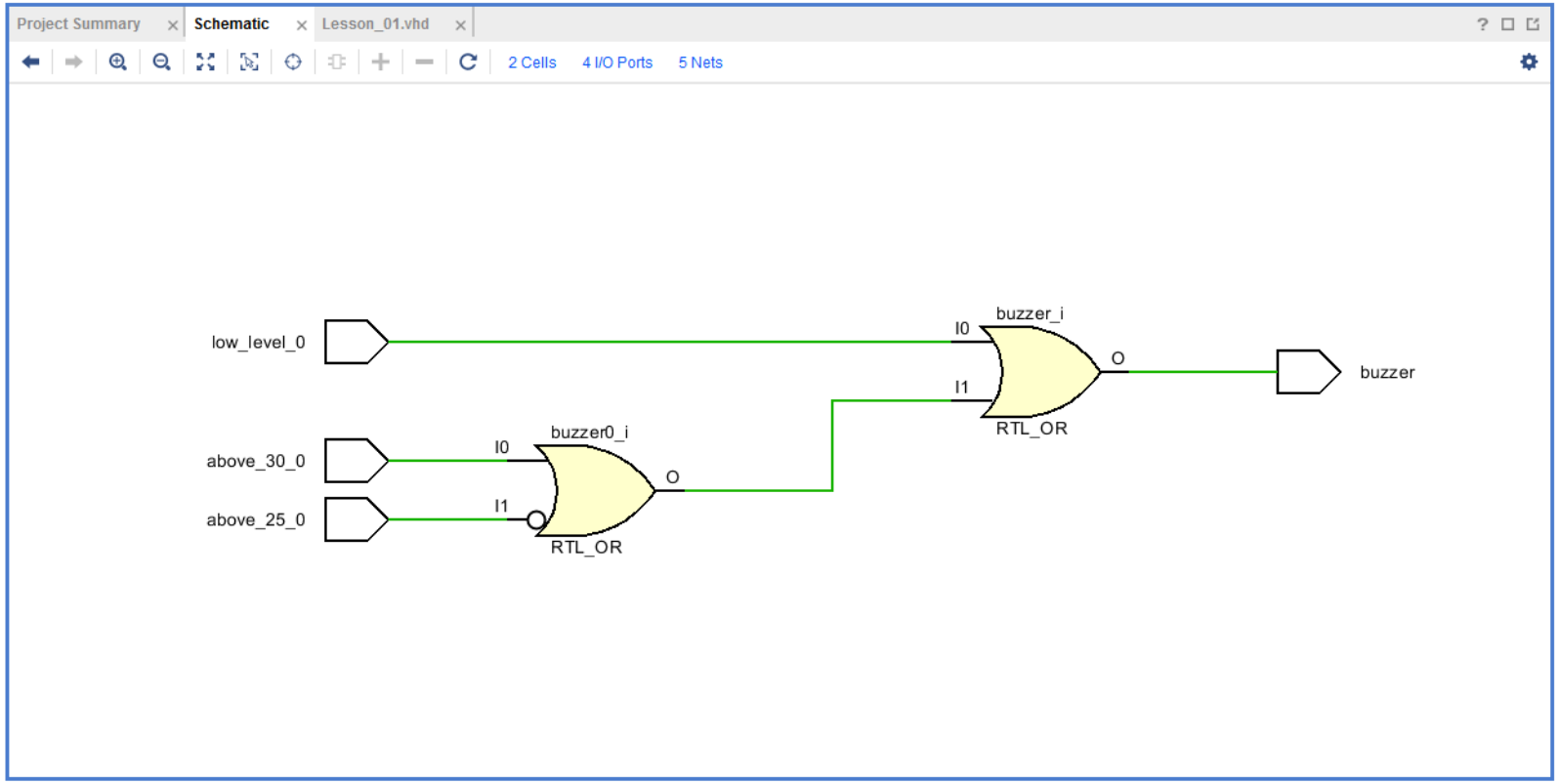


Αρχιτεκτονική
Περιγραφή της λειτουργικότητας που προσφέρει το λογικό κύκλωμα

```
architecture Behavioral of buzzer is

begin

buzzer<= low_level_0 or (above_30_0 or not above_25_0);

end architecture Behavioral;
```

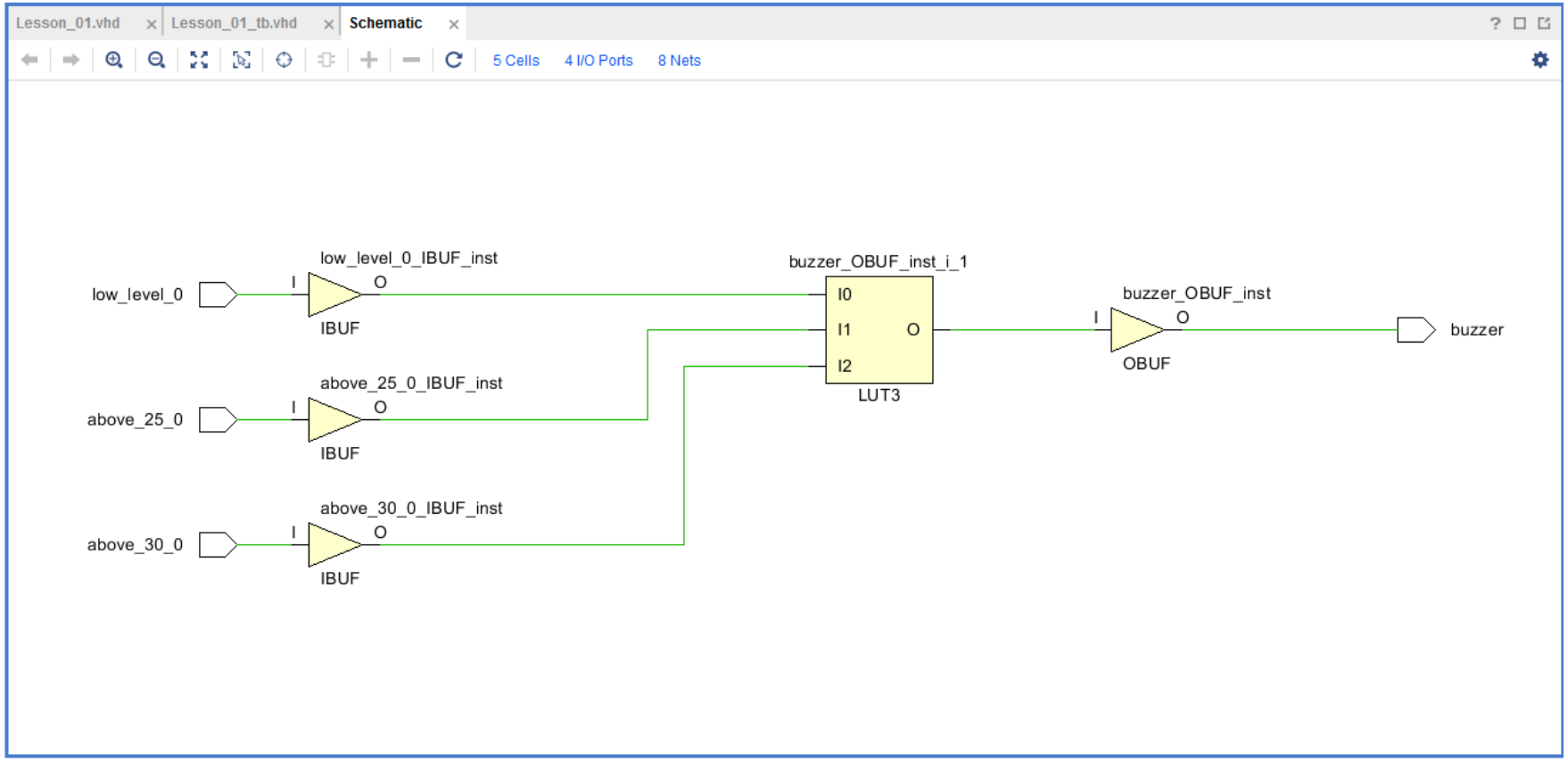
VHDL - Vivado

Ψηφιακό κύκλωμα – VHDL: RTL Αναπαράσταση



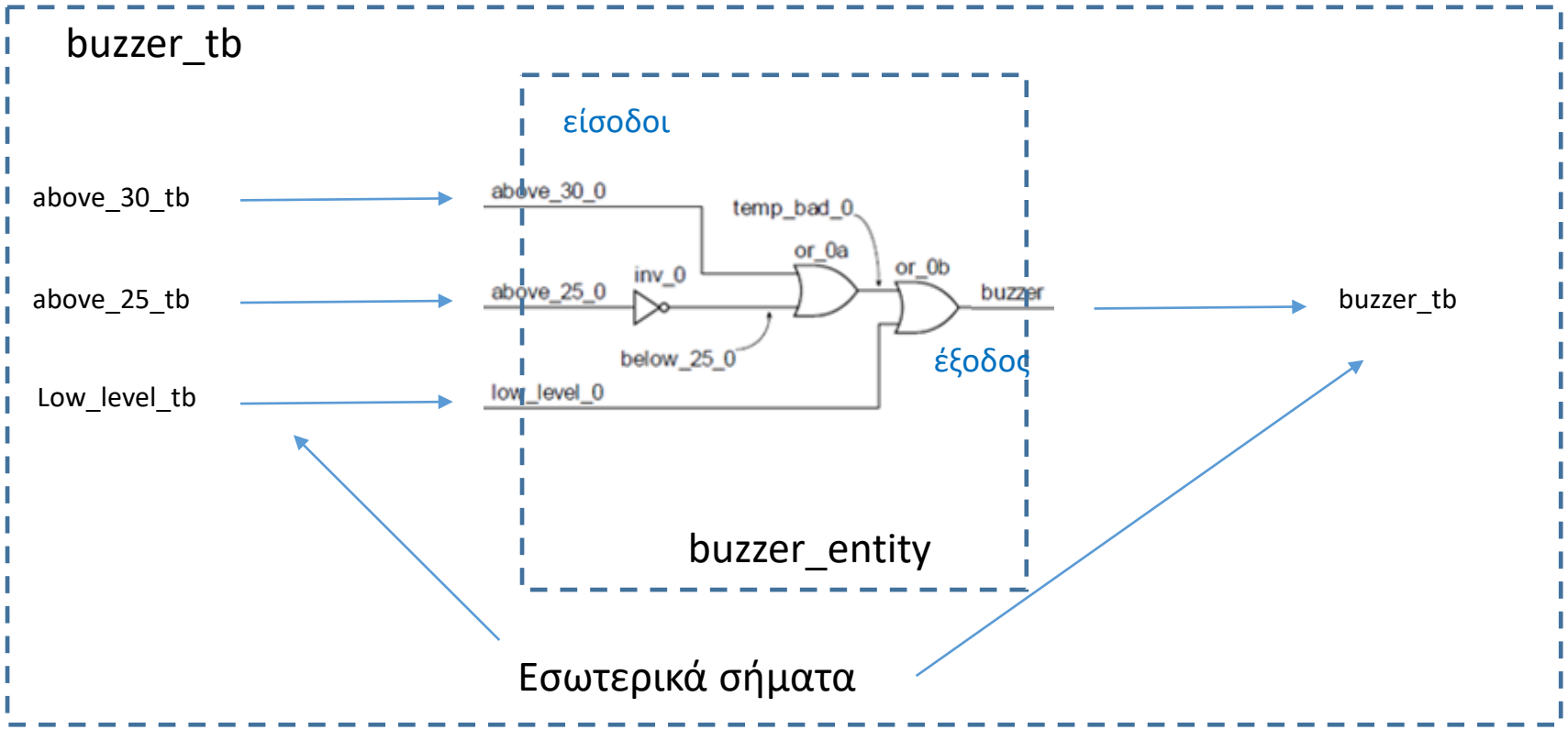
VHDL - Vivado

Ψηφιακό κύκλωμα – VHDL: Σύνθεση



VHDL - Vivado

Ψηφιακό κύκλωμα – VHDL: Προσομοίωση



VHDL - Vivado

Ψηφιακό κύκλωμα – VHDL: Προσομοίωση – Πίνακας Αληθείας

Πίνακας Αληθείας της συνάρτησης που εκφράζει το σήμα buzzer

above_30_0	above_25_0	low_level_0	buzzer
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

VHDL - Vivado

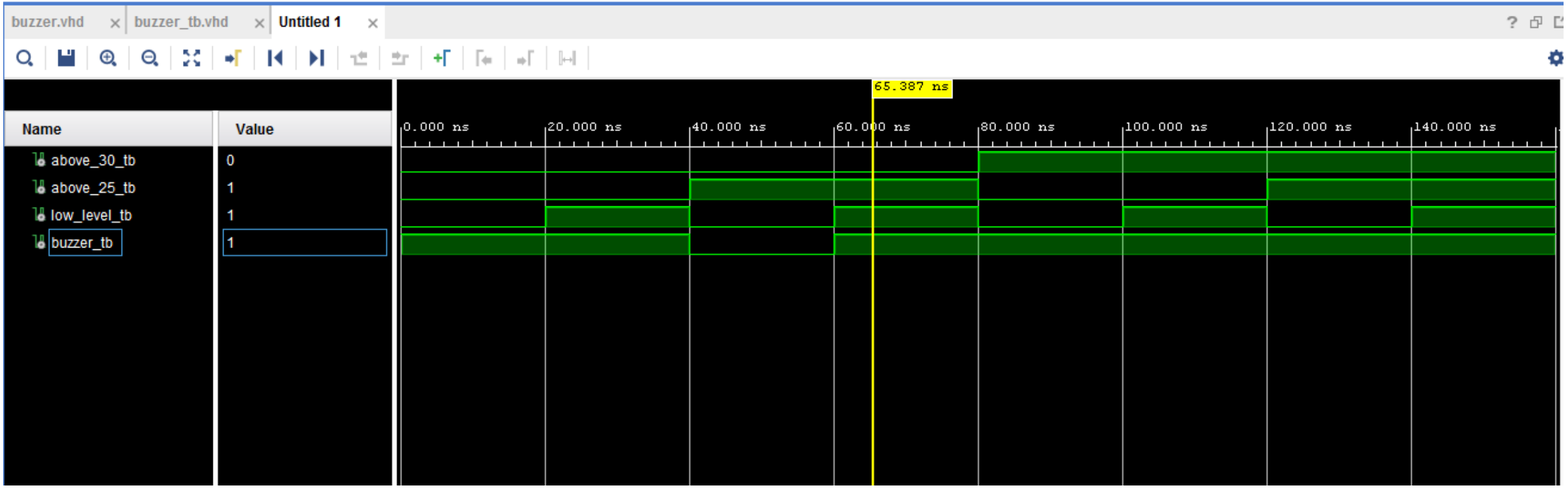
Ψηφιακό κύκλωμα – VHDL: Προσομοίωση - Testbench

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;

entity buzzer_tb is
end buzzer_tb ;

architecture Beh_tb of buzzer_tb is

component buzzer port(
above_30_0: in std_logic;
above_25_0: in std_logic;
low_level_0: in std_logic;
buzzer: out std_logic);

signal     above_25_tb, above_30_tb, low_level_tb    :  std_logic;
signal     buzzer_tb                                 :  std_logic;

begin
uut: buzzer  port map (above_25_0 => above_25_tb,
 above_30_0 => above_30_tb,  low_level_0 => low_level_tb,
          buzzer => buzzer_tb);
```

```
apply_test_cases: process is
begin

   above_30_tb <='0'; above_25_tb <='0'; low_level_tb <='0'; wait for 20 ns;
   above_30_tb <='0'; above_25_tb <='0'; low_level_tb <='1'; wait for 20 ns;
   above_30_tb <='0'; above_25_tb <='1'; low_level_tb <='0'; wait for 20 ns;
   above_30_tb <='0'; above_25_tb <='1'; low_level_tb <='1'; wait for 20 ns;
   above_30_tb <='1'; above_25_tb <='0'; low_level_tb <='0'; wait for 20 ns;
   above_30_tb <='1'; above_25_tb <='0'; low_level_tb <='1'; wait for 20 ns;
   above_30_tb <='1'; above_25_tb <='1'; low_level_tb <='0'; wait for 20 ns;
   above_30_tb <='1'; above_25_tb <='1'; low_level_tb <='1'; wait for 20 ns;

end process apply_test_cases;

end Beh_tb;
```

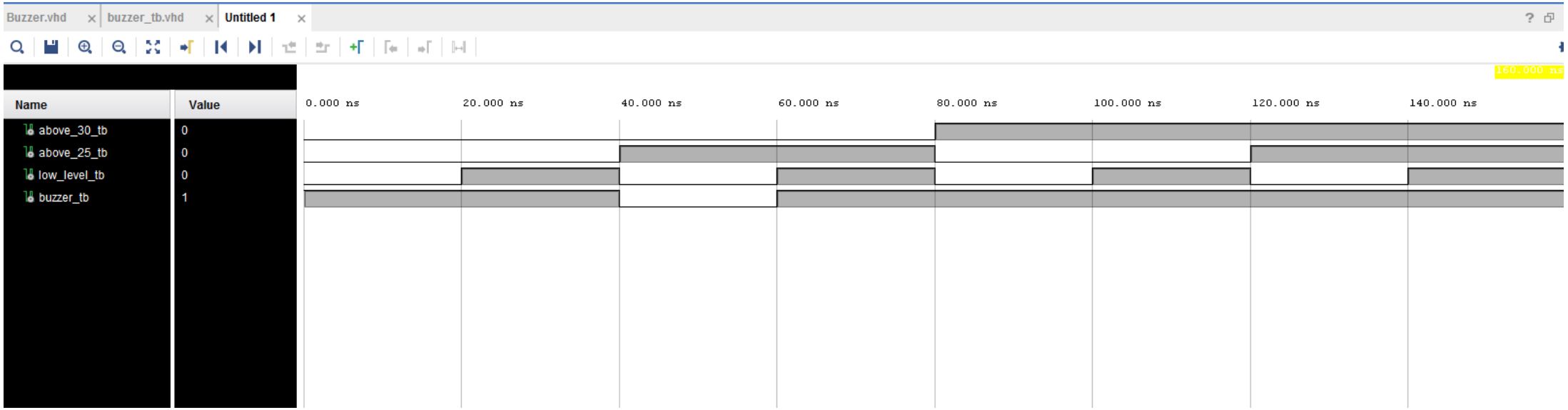
VHDL - Vivado

Ψηφιακό κύκλωμα – VHDL: Προσομοίωση - Χρονοσειρά



VHDL - Vivado

Ψηφιακό κύκλωμα – VHDL: Προσομοίωση - Χρονοσειρά



Πρωτοετείς

- https://delos365.grnet.gr/
  - Microsoft office (Word, Excel, Powerpoint,...) σε online και σε desktop έκδοση. Έχετε χώρο και στο onedrive (1TB). Σύνδεση απλά με τα ακαδημαϊκά σας credentials.
- google.com (ΠΡΟΣΟΧΗ: σύνδεση ως sdixxxxxxx@uoa.gr και μετά θα σας ζητηθούν τα ακαδημαϊκά σας credentials)
  - Εφαρμογές google + 50GB (google drive)
- https://azureforeducation.microsoft.com/devtools
  - Windows 10, 11 και εργαλεία ανάπτυξης λογισμικού της Microsoft (ΠΡΟΣΟΧΗ: σύνδεση ως sdixxxxxxx@o365.uoa.gr και μετά θα σας ζητηθούν τα ακαδημαϊκά σας credentials)

Περίληψη

- Εισαγωγή στη θεωρία γλωσσών HDL
- Εισαγωγή στο δυαδικό σύστημα αρίθμησης και στις λογικές πύλες
- Παράδειγμα ανάπτυξης εφαρμογής σε VHDL
- Διαβάζετε τις παραγράφους 2.1 και 3.1 από το βιβλίο του Ashenden.
- Διαβάζετε τις παραγράφους 1.5.1, 1.5.2, 1.5.3, 1.5.4 από το βιβλίο των Harris.

dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

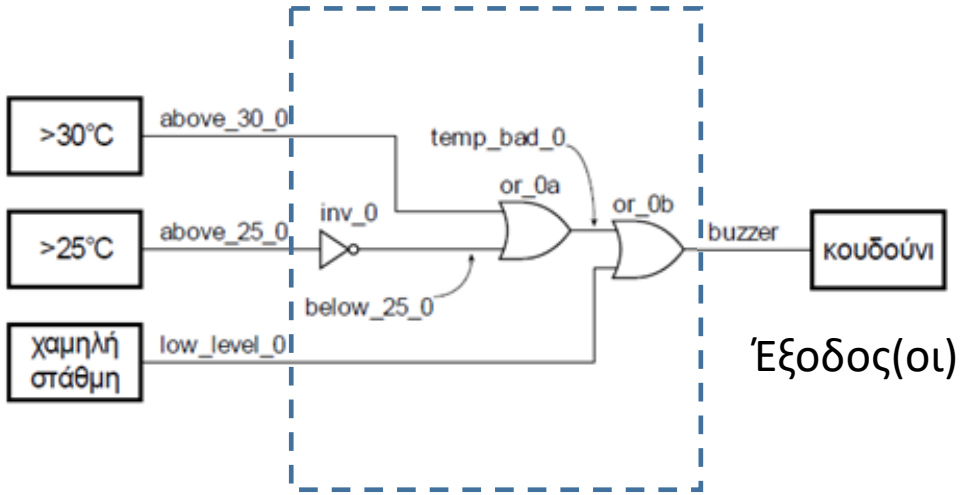
Εργαστήριο Λογικής Σχεδίασης

Εισαγωγή στη VHDL και το εργαλείο Vivado

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

VHDL

Ψηφιακό κύκλωμα – VHDL: Entity (Input/Output PORTS)



Υποχρεωτικές Βιβλιοθήκες

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity buzzer is

port (
above_25_0: in std_logic;
above_30_0: in std_logic;
low_level_0: in std_logic;
buzzer      : out std_logic );

end entity buzzer;
```

Περιγραφή Οντότητας

VHDL

Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Entity: Input/Output

- Περιγράφει τη διασύνδεση μίας λογικής μονάδας, χωρίς να προσδιορίζει τη συμπεριφορά της (μαύρο κουτί - black box)
- Η διασύνδεση της μονάδας περιγράφεται με μία δήλωση των **διαύλων/θυρών επικοινωνίας (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 entity_name;
```

VHDL

Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Entity: Input/Output

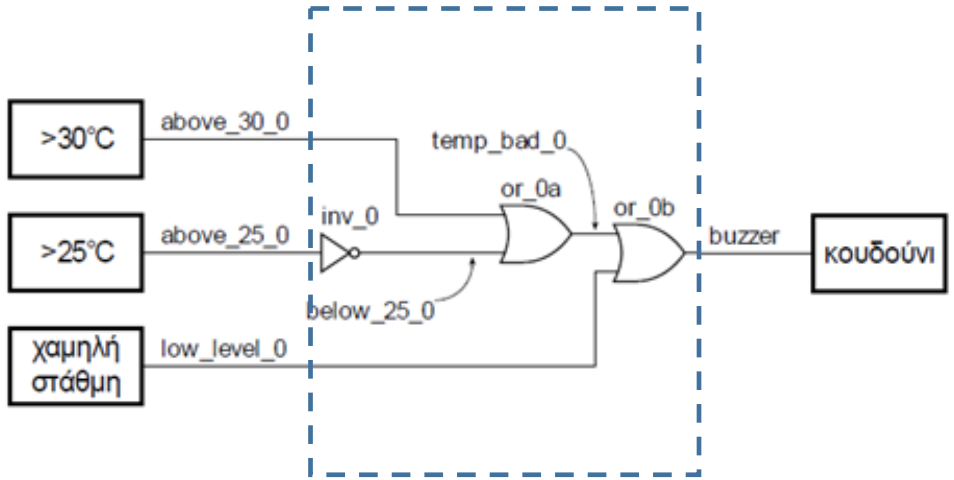
- **entity_name**: το όνομα της οντότητας
- **signal_name**: το όνομα του σήματος
  (εάν είναι πολλά σήματα χωρίζονται με κόμμα)
- **mode**: η κατεύθυνση του σήματος
  - **in**: είσοδος της οντότητας
  - **out**: έξοδος της οντότητας
  - **inout**: είσοδος ή έξοδος της οντότητας (bidirectional),(ΔΕΝ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

VHDL

Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Ονόματα & Ετικέτες

- Είναι **μοναδικά** μέσα σε μία συγκεκριμένη οντότητα (και αρχιτεκτονική)
- Τα σχόλια σε μία γραμμή έπονται του **"--"**
- 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**
- **Προσοχή στις δεσμευμένες λέξεις**

Ψηφιακό κύκλωμα – VHDL: Architecture (Dataflow)

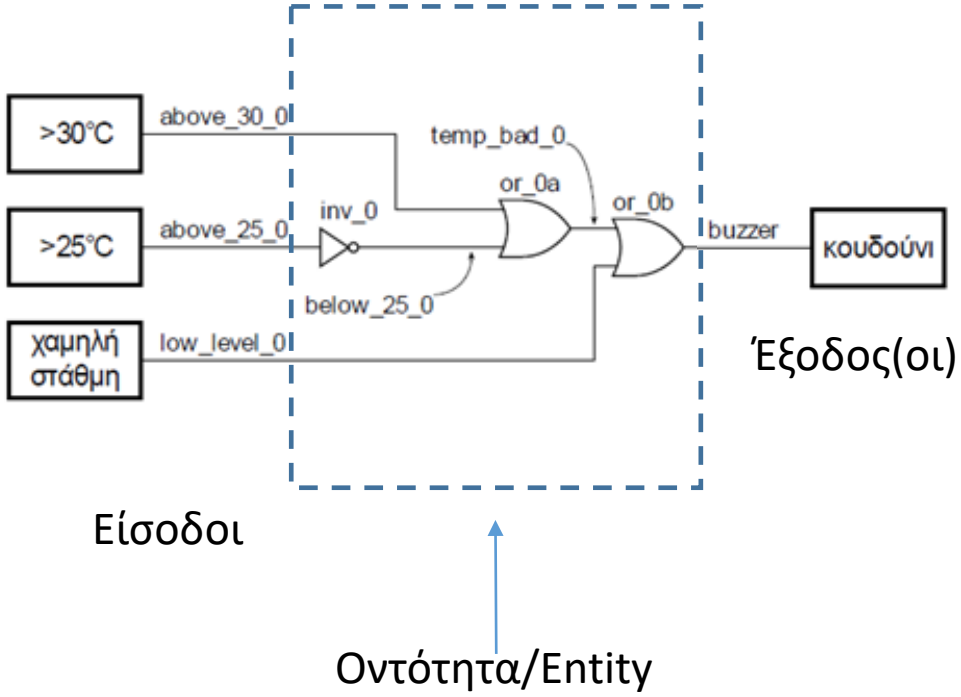



```
architecture Dataflow of buzzer is

begin

buzzer<= low_level_0 or (above_30_0 or not above_25_0);

end architecture buzzer;
```

Αρχιτεκτονική
Περιγραφή της λειτουργικότητας που προσφέρει το λογικό κύκλωμα

Τα σήματα εξόδου (out) ΠΑΝΤΑ αριστερά, τα σήματα εισόδου (in) ΠΑΝΤΑ δεξιά.

Ψηφιακό κύκλωμα – VHDL: Entity (Εσωτερικά σήματα)



Εσωτερικά σήματα

temp_bad_0

-

below_25_0

VHDL

Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Architecture

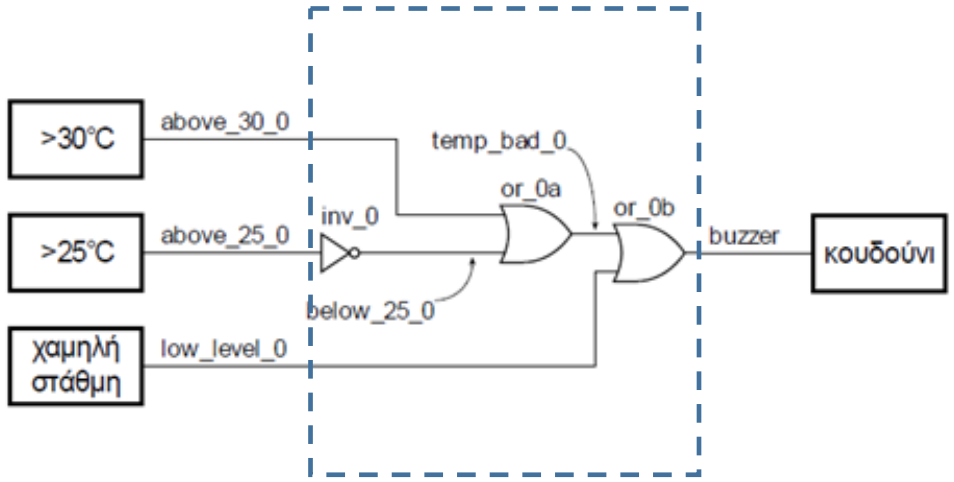
```
architecture arch_name of entity_name is
    signal signal_name: signal_type;
    component comp_name
        port (
            signal_name: mode signal_type;
            ...
            signal_name: mode signal_type);
    end component;
    ...
begin
    concurrent_component_statement;
    ...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

VHDL

Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Architecture

- **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 **is**) το σήμα είναι μία **εσωτερική διασύνδεση** της αρχιτεκτονική της οντότητας
  - στις δηλώσεις των διαύλων του στοιχείου (component) το σήμα είναι είσοδος, έξοδος του στοιχείου, όπως ακριβώς προκύπτει από τη δήλωση των διαύλων της οντότητας του συγκεκριμένου στοιχείου
- **signal_type**: ο τύπος του σήματος (STD_LOGIC ή άλλος)

Ψηφιακό κύκλωμα – VHDL: Architecture (Dataflow)




```
architecture Dataflow of buzzer is
signal temp_bad_0, below_25_0: std_logic;
begin

below_25_0<=not above_25_0;
temp_bad_0<=above_30_0 or below_25_0;
buzzer<= temp_bad_0 or low_level_0;

end architecture buzzer;
```

Περιγραφή της λειτουργικότητας που προσφέρει το λογικό κύκλωμα με χρήση εσωτερικών σημάτων

concurrent statements

Τα εσωτερικά σήματα μπορούν να είναι και αριστερά και δεξιά

Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Architecture

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

```
signal_name <= expression;
```

- **expression**: έκφραση με σήματα και τελεστές
- **signal_name**: το όνομα του σήματος
  - στις ταυτόχρονες εντολές ανάθεσης σήματος :
    - στην **έκφραση (expression)** προσδιορίζονται σήματα που είναι **είσοδοι** στην οντότητα και δηλώνονται κατά τη δήλωση των διαύλων της οντότητας, και **εσωτερικές διασυνδέσεις (εσωτερικά σήματα)** που δηλώνονται κατά τη δήλωση σημάτων
    - στο αριστερό μέρος της εντολής προσδιορίζεται σήμα που είναι **έξοδος** της οντότητας και δηλώνεται κατά τη δήλωση των διαύλων της οντότητας, ή **εσωτερική διασύνδεση (εσωτερικό σήμα)** που δηλώνεται κατά τη δήλωση σημάτων

Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Architecture

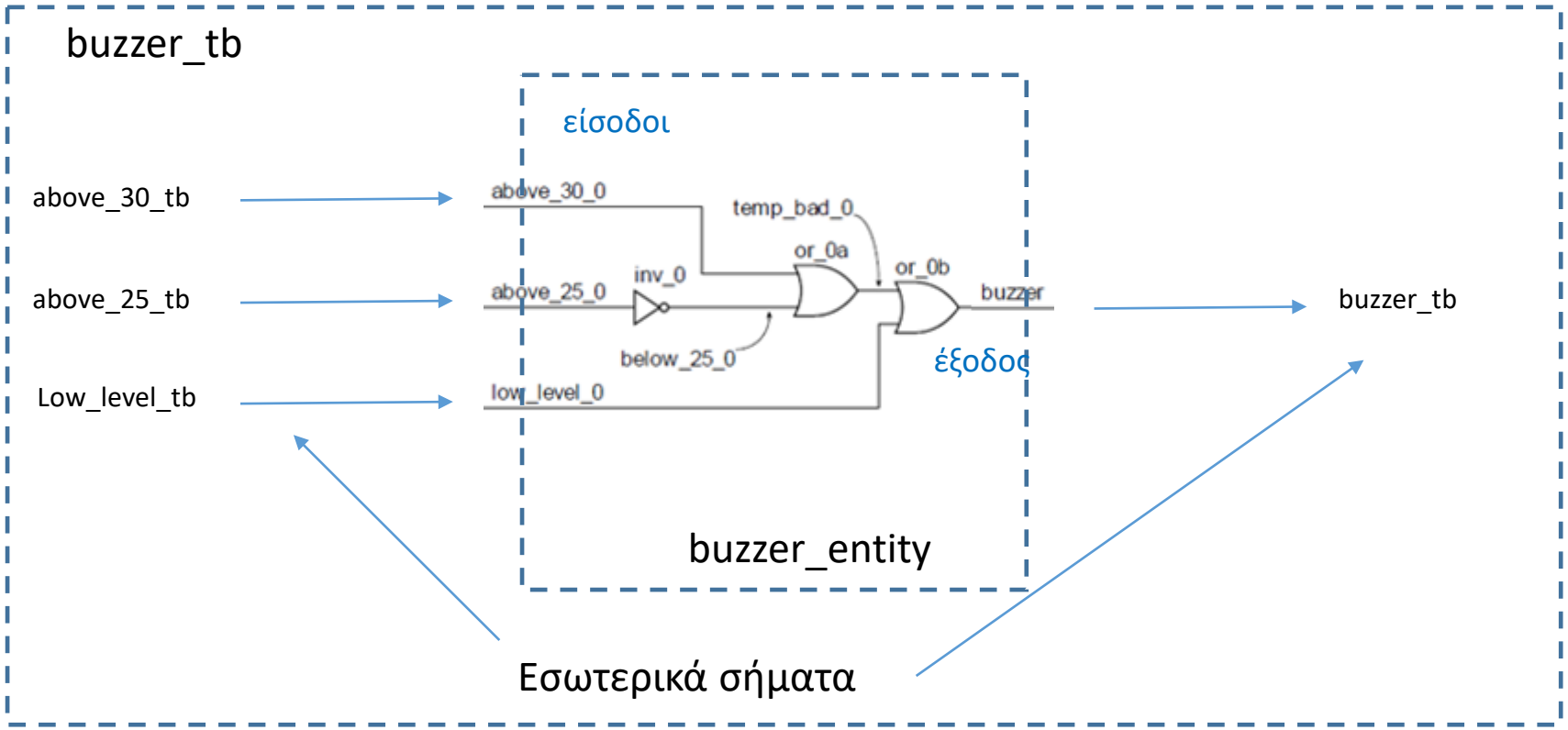
- **Εκτέλεση** ταυτόχρονων εντολών ανάθεσης σήματος (concurrent_signal_assignment_statements)

```
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.

Ψηφιακό κύκλωμα – VHDL: Προσομοίωση



Ψηφιακό κύκλωμα – VHDL: Προσομοίωση – Πίνακας Αληθείας

Πίνακας Αληθείας της συνάρτησης που εκφράζει το σήμα buzzer

above_30_0	above_25_0	low_level_0	buzzer_0
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

VHDL - Vivado

Ψηφιακό κύκλωμα – VHDL: Προσομοίωση - Testbench

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;

entity buzzer_tb is
end buzzer_tb ;

architecture Beh_tb of buzzer_tb is

component buzzer port(
above_30_0: in std_logic;
above_25_0: in std_logic;
low_level_0: in std_logic;
buzzer: out std_logic);

signal     above_25_tb, above_30_tb, low_level_tb    :  std_logic;
signal     buzzer_tb                                 :  std_logic;

begin
uut: buzzer  port map (above_30_tb, above_25_tb, low_level_tb,
buzzer_tb);
```

```
apply_test_cases: process is
begin
   above_30_tb <='0'; above_25_tb <='0'; low_level_tb <='0'; wait for 20 ns;
   above_30_tb <='0'; above_25_tb <='0'; low_level_tb <='1'; wait for 20 ns;
   above_30_tb <='0'; above_25_tb <='1'; low_level_tb <='0'; wait for 20 ns;
   above_30_tb <='0'; above_25_tb <='1'; low_level_tb <='1'; wait for 20 ns;
   above_30_tb <='1'; above_25_tb <='0'; low_level_tb <='0'; wait for 20 ns;
   above_30_tb <='1'; above_25_tb <='0'; low_level_tb <='1'; wait for 20 ns;
   above_30_tb <='1'; above_25_tb <='1'; low_level_tb <='0'; wait for 20 ns;
   above_30_tb <='1'; above_25_tb <='1'; low_level_tb <='1'; wait for 20 ns;
end process apply_test_cases;

end architecture  Beh_tb;
```

Όταν γράφουμε μόνο τα εσωτερικά σήματα, η σειρά τους καθορίζει έμμεσα την αντιστοίχιση με τα port του component

VHDL

Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Αρχιτεκτονική – Components (1/2)

- **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 (υποκύκλωμα) που χρησιμοποιείται στην αρχιτεκτονική της οντότητας
- **signal_name**: το όνομα του σήματος που συνδέεται στο υποκύκλωμα (comp_name) (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 port του υποκυκλώματος (comp_name) (**θέλει προσοχή η σειρά των σημάτων**)

VHDL - Vivado

Ψηφιακό κύκλωμα – VHDL: Προσομοίωση - Testbench

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;

entity buzzer_tb is
end buzzer_tb ;

architecture Beh_tb of buzzer_tb is

component buzzer port(
above_30_0: in std_logic;
above_25_0: in std_logic;
low_level_0: in std_logic;
buzzer: out std_logic);

signal     above_25_tb, above_30_tb, low_level_tb     :  std_logic;
signal     buzzer_tb                                  :  std_logic;

begin
```

```
uut: buzzer  port map (
above_25_0 => above_25_tb,
above_30_0 => above_30_tb,
low_level_0 => low_level_tb,
buzzer => buzzer_tb);
```

Όταν ορίζουμε άμεσα την αντιστοίχιση των εσωτερικών σημάτων με τα port, η σειρά με την οποία το κάνουμε είναι αδιάφορη

```
apply_test_cases: process is
begin
   above_30_tb <='0'; above_25_tb <='0'; low_level_tb <='0'; wait for 20 ns;
   above_30_tb <='0'; above_25_tb <='0'; low_level_tb <='1'; wait for 20 ns;
   above_30_tb <='0'; above_25_tb <='1'; low_level_tb <='0'; wait for 20 ns;
   above_30_tb <='0'; above_25_tb <='1'; low_level_tb <='1'; wait for 20 ns;
   above_30_tb <='1'; above_25_tb <='0'; low_level_tb <='0'; wait for 20 ns;
   above_30_tb <='1'; above_25_tb <='0'; low_level_tb <='1'; wait for 20 ns;
   above_30_tb <='1'; above_25_tb <='1'; low_level_tb <='0'; wait for 20 ns;
   above_30_tb <='1'; above_25_tb <='1'; low_level_tb <='1'; wait for 20 ns;
end process apply_test_cases;

end Beh_tb;
```

VHDL

Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Αρχιτεκτονική – Components (2/2)

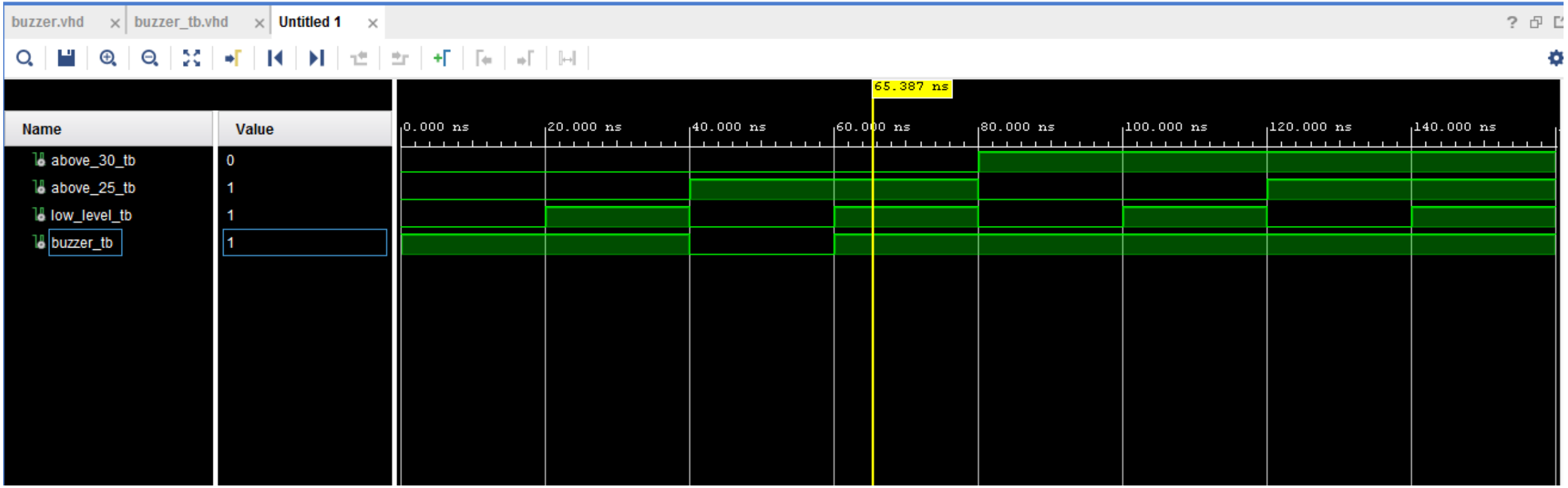
- **Ταυτόχρονες εντολές στοιχείων** (concurrent_component_statements)

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

- **port_name:** το όνομα του port της οντότητας (component) το οποίο καλούμε.
- **signal_name**: το όνομα του σήματος το οποίο συνδέουμε με στο σήμα port_name (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 (**θέλει προσοχή η σειρά των σημάτων**)

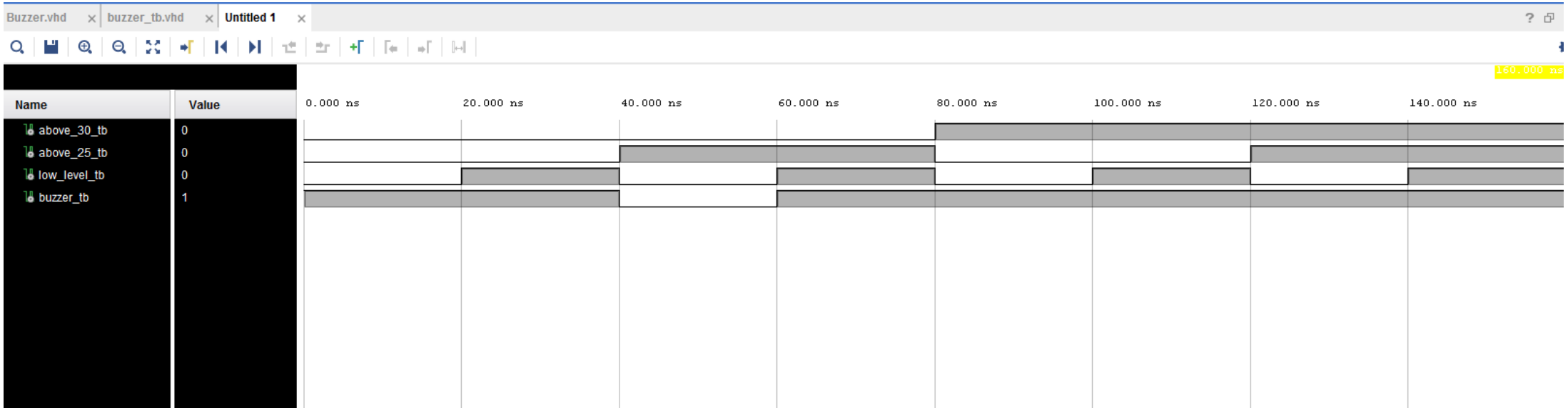
VHDL

Ψηφιακό κύκλωμα – VHDL: Προσομοίωση - Χρονοσειρά



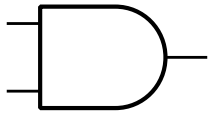
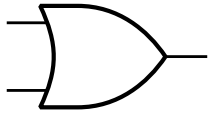
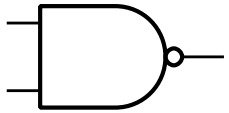
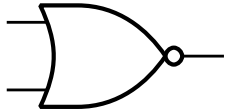
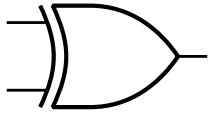
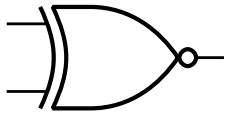
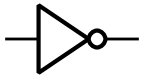
VHDL - Vivado

Ψηφιακό κύκλωμα – VHDL: Προσομοίωση - Χρονοσειρά



VHDL

Λογικοί Τελεστές

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
  - οι υπόλοιποι τελεστές έχουν ίση προτεραιότητα
  - από αριστερά προς τα δεξιά
  - χρησιμοποιούμε παρενθέσεις για να διακρίνουμε τη σειρά υπολογισμού
- Τιμές bit στην VHDL
  - '0' και '1'

VHDL – Τύποι σημάτων

Std_logic

Τιμή	Modeling for simulation	Synthesis
U	Uninitialized	Uninitialized
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

VHDL – Τύποι σημάτων

Std_logic_vector

```
signal_in_0: in std_logic;
signal_in_1: in std_logic;
signal_in_2: in std_logic;
signal_in_3: in std_logic;
```

Εναλλακτικά

```
signal_in : in std_logic_vector (3 downto 0);
```

```
signal_in <="0101";
```

```
signal_in(0) <='1'
signal_in(1) <='0'
signal_in(2) <='1'
signal_in(3) <='0'
```

Προσοχή σε " και '

VHDL – Τύποι σημάτων

Std_logic_vector

- Ο τύπος του λογικού διανύσματος (μονοδιάστατου array) **STD_LOGIC_VECTOR** είναι μέρος του πακέτου **IEEE.std_logic_1164** της βιβλιοθήκης **IEEE**
- Προσδιορίζει ένα διατεταγμένο σύνολο από σήματα (μεταβλητές) τύπου **STD_LOGIC**.
- Η διάταξη μπορεί να είναι αύξουσα

  STD_LOGIC_VECTOR (0 to 7)

  ή φθίνουσα

  STD_LOGIC_VECTOR (7 downto 0)
- Οι δείκτες των στοιχείων του array είναι τύπου **natural**
- Προσοχή, **δεν είναι ακέραιος δυαδικός αριθμός**

VHDL – Τύποι σημάτων

Std_logic_vector

- Δήλωση τιμών για το 8-ψήφιο λογικό διάνυσμα V
  - V <= “11110000"
  - V <= (others => '0') -- όλα-0
- Συγκρίσεις:
  - V = "00000000" για σύγκριση **ολόκληρου** του διανύσματος
  - V(3 downto 0) = "0000" για σύγκριση **μέρους** του διανύσματος
  - Προσοχή. **Μη επιτρεπτή σύγκριση**: V = "----0000"
    - το '-' δεν εκλαμβάνεται σαν don't care κατά τη σύγκριση

Προσοχή. **Σε όλα τα προγράμματα τα PORT στον ορισμό της Οντότητας θα είναι MONO STD_LOGIC ή STD_LOGIC_VECTOR**

Εγγραφές σε Τμήματα Εργαστηρίων

- Όσοι είστε με μεταγραφή και δεν έχετε κωδικούς ως φοιτητές του τμήματος προσπαθήστε να κάνετε εγγραφή ως επισκέπτες.
- Μετά τις 24/10 θα ενταχθείτε σε ομάδες για παρακολούθηση του αντίστοιχου εργαστηρίου (δίπλα στο αναγνωστήριο).
- Την εβδομάδα 31/10 – 4/11 θα γίνει το πρώτο ΕΡΓΑΣΤΗΡΙΟ (Δεν θα γίνει διάλεξη στην Α2).

Περίληψη

- Παράδειγμα ανάπτυξης εφαρμογής σε VHDL
- Δηλώσεις Οντότητας (entity), Αρχιτεκτονικής (architecture).
- Ports και Εσωτερικά σήματα
- Ταυτόχρονες εντολές
- Components
- Προτεραιότητες πράξεων
- Τύποι σημάτων (std_logic, std_logic_vector)
- Διαβάζετε τις παραγράφους 3.2 από Ashenden και 2.1, 2.6, 4.1-4.6 (ΟΧΙ το κομμάτι της VERILOG) από το βιβλίο των Harris.

dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

Εργαστήριο Λογικής Σχεδίασης

Παράδειγμα ανάπτυξης Project στο Vivado

Βασιλόπουλος Διονύσης

ΕΤΕΠ Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

VHDL - Παράδειγμα

Πραγματικό πρόβλημα

Σε ένα Computer Room υπάρχουν δύο (2) αισθητήρες θερμοκρασίας (**Sensor_1** και **Sensor_2**) και δύο (2) κλιματιστικά (**AirCond_1** και **AirCond_2**). Το πρώτο κλιματιστικό (AirCond_1) λειτουργεί εάν τουλάχιστον ένας από τους δύο αισθητήρες ανιχνεύσουν θερμοκρασία άνω των 35 βαθμών στο computer room. Το δεύτερο κλιματιστικό (AirCond_2) λειτουργεί εάν και οι δύο αισθητήρες ανιχνεύσουν θερμοκρασία άνω των 35 βαθμών στο computer room. Θεωρείστε ότι κάθε αισθητήρας δίνει σήμα (‘1’) μόνο όταν η θερμοκρασία που ανιχνεύει γίνει μεγαλύτερη των 35 βαθμών (>35). Σε άλλη περίπτωση ο αισθητήρας στέλνεις την τιμή ‘0’. Σχεδιάστε και υλοποιήστε το λογικό κύκλωμα που περιγράφει το ανωτέρω πρόβλημα. Το όνομα του Vivado Project θα είναι Lab_1, της οντότητας θα είναι CR_AC ενώ το όνομα της αρχιτεκτονικής Dataflow.

Σχεδιάστε και υλοποιήστε το λογικό κύκλωμα που περιγράφει το ανωτέρω πρόβλημα.

VHDL - Παράδειγμα

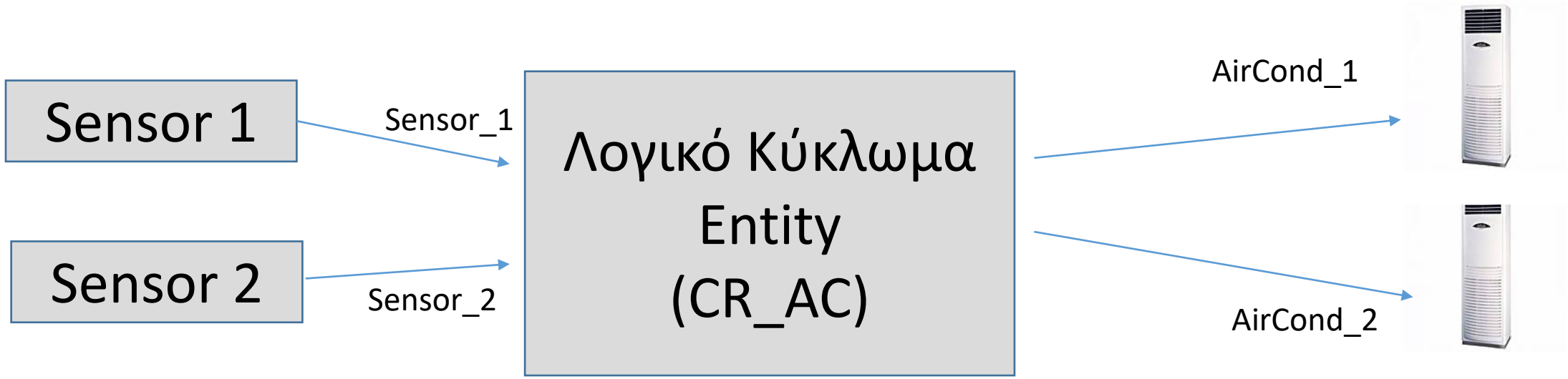
Πραγματικό πρόβλημα – Βήματα επίλυσης

1.Δημιουργία νέου project
2.Δημιουργία Entity – Εντοπισμός Input/Output του συστήματος
3.Δημιουργία Architecture – Θα έχετε τόσες εντολές όσες είναι και οι έξοδοι του συστήματος. Κάθε μία εντολή αντιστοιχεί σε μία έξοδο.
4.Δημιουργία RTL αναπαράστασης
5.Προσομοίωση

(Αναλυτικές οδηγίες στον Οδηγό Vivado στο eclass)

VHDL - Παράδειγμα

Απλοποιημένη μορφή κυκλώματος



VHDL - Παράδειγμα

Περιγραφή Οντότητας (entity)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity CR_AC is
port (
        Sensor_1        :in STD_LOGIC;
        Sensor_2        :in STD_LOGIC;
        AirCond_1       :out STD_LOGIC;
        AirCond_2       :out STD_LOGIC
);
end entity CR_AC ;
```

VHDL - Παράδειγμα

Λογική κατανόηση της άσκησης

AirCond_1: Ανάβει όταν δίνει σήμα **τουλάχιστον ένας** αισθητήρας =>
Ανάβει όταν δίνει σήμα **οποιοσδήποτε** αισθητήρας

AirCond_2: Ανάβει όταν δίνουν σήμα **και** οι δύο αισθητήρες

VHDL - Παράδειγμα

Πίνακας Αληθείας

Είσοδοι (Inputs)		Έξοδοι (outputs)	
Sensor_1	Sensor_2	AirCond_1	AirCond_2
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

VHDL - Παράδειγμα

Πίνακας Αληθείας - Ελαχιστόροι

Είσοδοι (Inputs)		Έξοδοι (outputs)	
Sensor_1	Sensor_2	AirCond_1	AirCond_2
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

AirCond_1= Sensor_1'*Sensor_2+Sensor1*Sensor_2'+
Sensor_1*Sensor_2=
Sensor_2+Sensor1*Sensor_2'=
Sensor_2+Sensor1

AirCond_2= **Sensor1*Sensor_2**

***=AND**
+=OR

VHDL - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Dataflow)

```
architecture Dataflow of CR_AC is
begin

        AirCond_1<=Sensor_1 or Sensor_2;
        AirCond_2<=Sensor_1 and Sensor_2;

end architecture Dataflow;
```

VHDL - Παράδειγμα

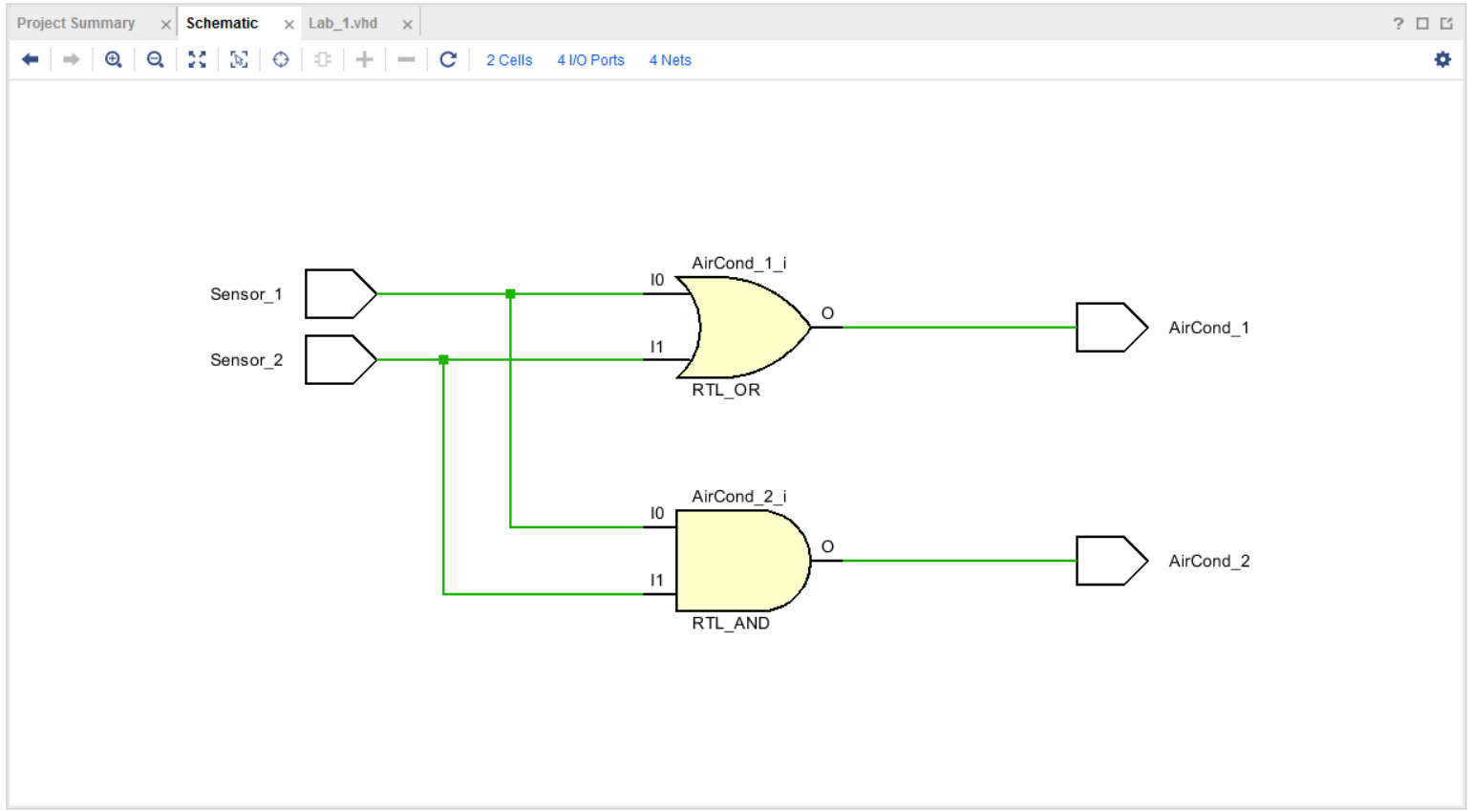
Τελική έκδοση προγράμματος

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

Entity CR_AC is
Port (
Sensor_1    :in std_logic;
Sensor_2    :in std_logic;
AirCond_1   :out std_logic;
AirCond_2   :out std_logic
);

end entity CR_AC;
```

```
architecture Dataflow of CR_AC is

begin

AirCond_1<=Sensor_1 or Sensor_2;
AirCond_2<=Sensor_1 and Sensor_2;

end architecture Dataflow;
```

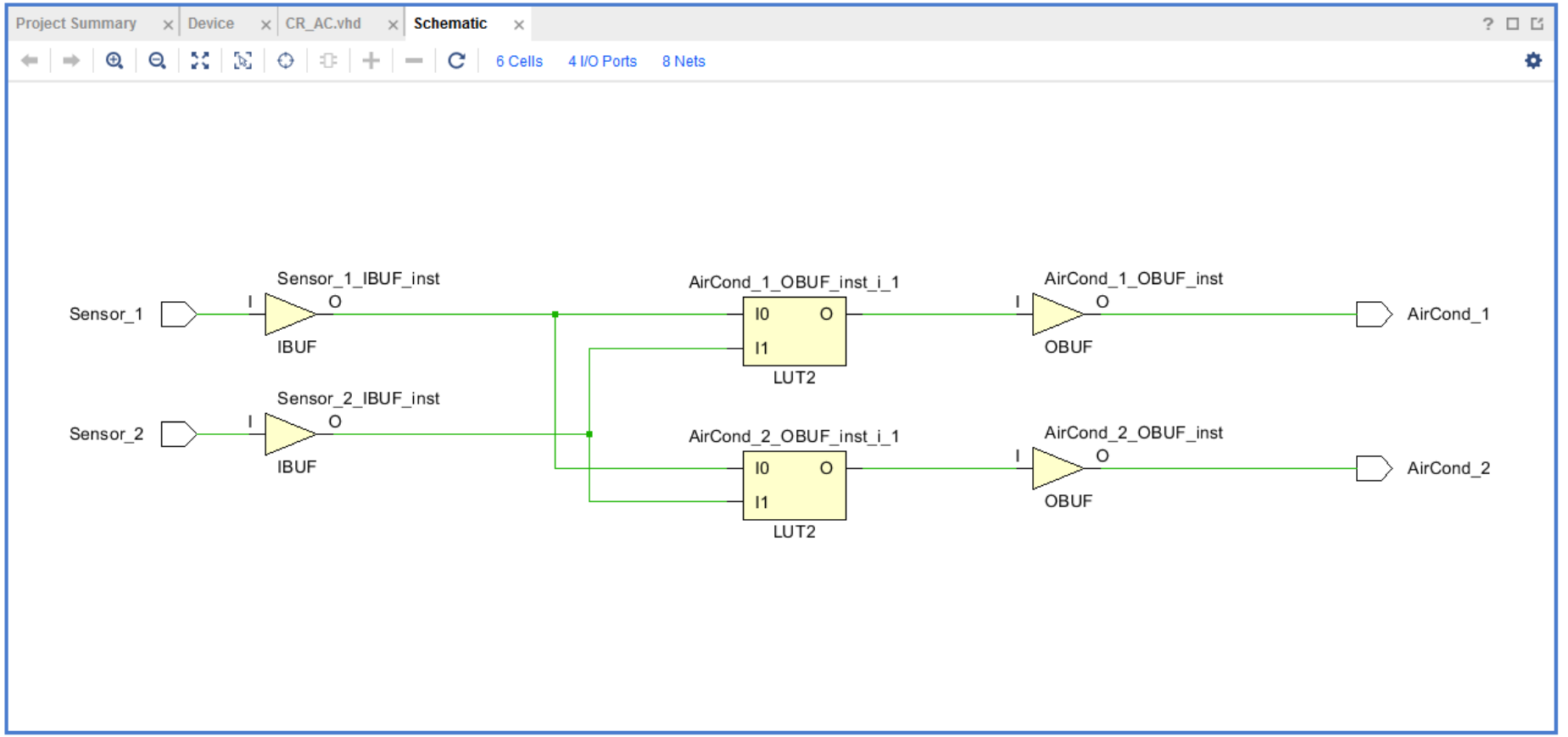
VHDL - Παράδειγμα

Λογικό κύκλωμα: Αναπαράσταση RTL



VHDL - Παράδειγμα

Λογικό κύκλωμα: Φάση Σύνθεσης – LUT(LookUp Table), υλοποιούν Πίνακες Αληθείας (προγραμματιζόμενα μέρη της κάρτας FPGA)

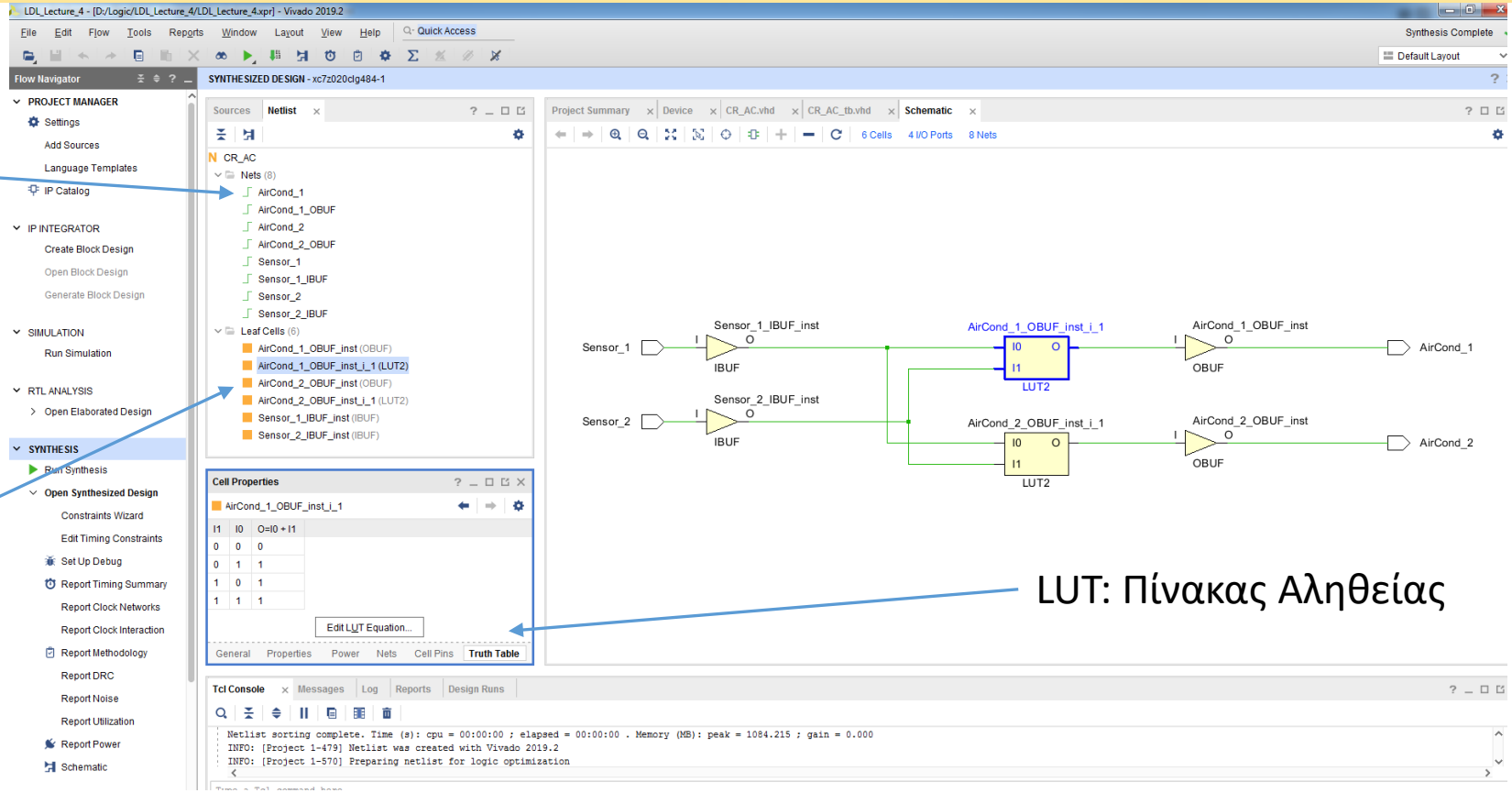


VHDL - Παράδειγμα

Λογικό κύκλωμα: Φάση Σύνθεσης – LUT(LookUp Table), υλοποιούν Πίνακες Αληθείας (προγραμματιζόμενα μέρη της κάρτας FPGA)

Nets->Σύρματα

Cells->Ψηφιακά στοιχεία



LUT: Πίνακας Αληθείας

VHDL - Παράδειγμα

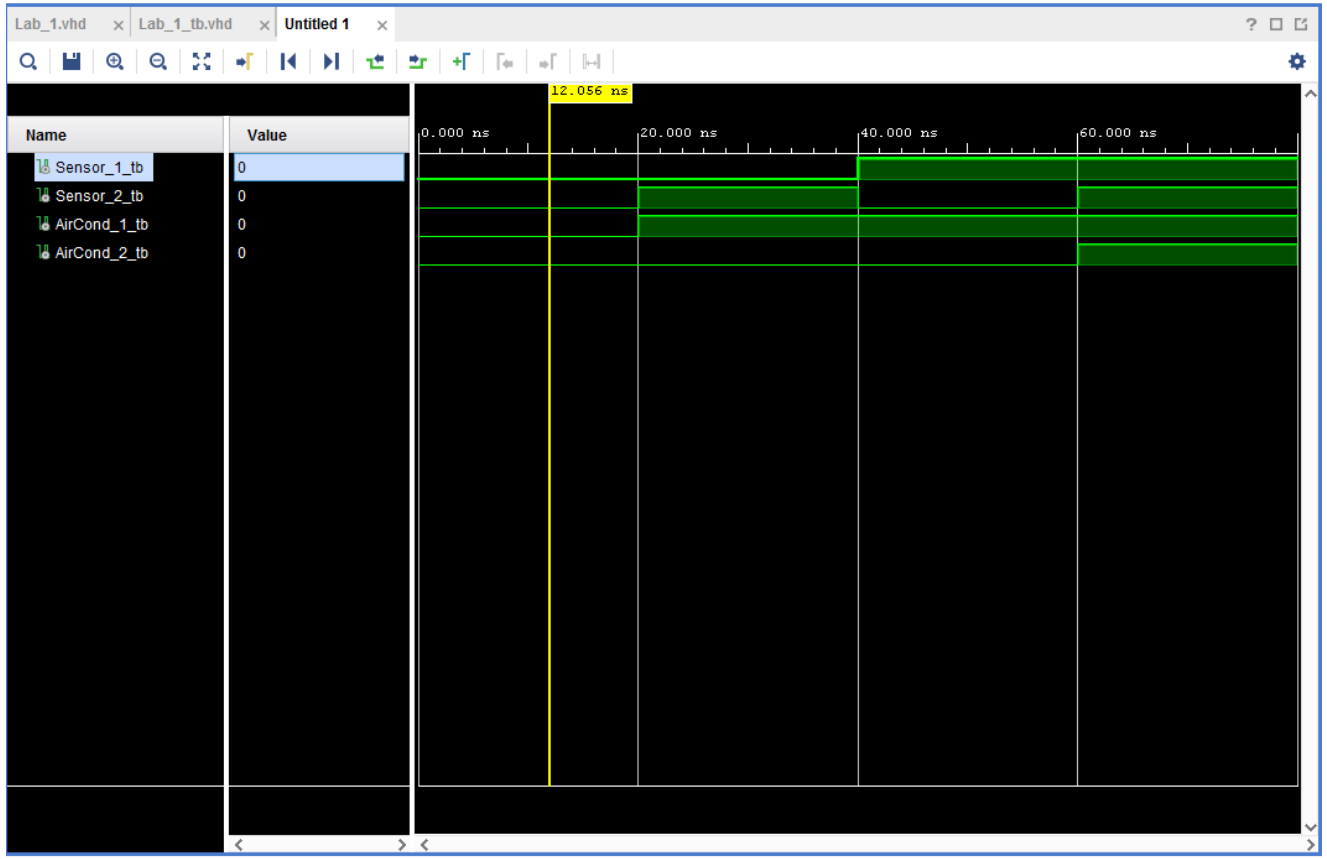
Προσομοίωση (Πρόγραμμα testbench)

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;

entity CR_AC_tb is
end CR_AC_tb;

architecture Behavioral of CR_AC_tb is

component CR_AC is
Port (
Sensor_1    :in std_logic;
Sensor_2    :in std_logic;
AirCond_1  :out std_logic;
AirCond_2  :out std_logic);
end component CR_AC;

-- Internal Signals: One for each port
signal Sensor_1_tb    :std_logic; signal Sensor_2_tb    :std_logic;
signal AirCond_1_tb   :std_logic; signal AirCond_2_tb   :std_logic;
```

```
begin

-- create entity into testbench entity
test_entity: CR_AC port map (Sensor_1=>Sensor_1_tb,
Sensor_2=>Sensor_2_tb, AirCond_1=>AirCond_1_tb,
AirCond_2=>AirCond_2_tb);

enter_test_values: process is
begin

Sensor_1_tb<='0';Sensor_2_tb<='0';wait for 20 ns;
Sensor_1_tb<='0';Sensor_2_tb<='1';wait for 20 ns;
Sensor_1_tb<='1';Sensor_2_tb<='0';wait for 20 ns;
Sensor_1_tb<='1';Sensor_2_tb<='1';wait for 20 ns;

end process enter_test_values;

end  architecture Behavioral;
```

VHDL - Παράδειγμα

Προσομοίωση Behavioral (Χρονοσειρά)



Περίληψη

- Ανάπτυξη βήμα-βήμα μιας απλής εφαρμογής στο Vivado
- RTL->Synthesis
- Προσομοίωση
- LUT

dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

Εργαστήριο Λογικής Σχεδίασης

1ο Εργαστηριακό Μάθημα

Βασιλόπουλος Διονύσης

ΕΤΕΠ Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

VHDL – Παράδειγμα

Άσκηση

Να σχεδιάσετε και να προσομοιώσετε στο Vivado ένα απλό κύκλωμα ηλεκτρονικής κλειδαριάς που θα δέχεται ως είσοδο/κωδικό κλειδαριάς έναν ακέραιο αριθμό 4-bits (στο δυαδικό) και θα ενεργοποιεί (λογικό 1) την έξοδο της κλειδαριάς (lock_out) μόνο όταν ο αριθμός αυτός ταυτίζεται με το τελευταίο ψηφίο του ΑΜ σας. Για παράδειγμα, για τον ΑΜ 1115201900205, ο κωδικός έχει την τιμή 5 (στο δυαδικό 0101).
Το όνομα του project θα είναι Lab1, το όνομα του αρχείου (design source) αλλά και η οντότητα σας θα λέγεται locker, ενώ η αρχιτεκτονική Dataflow. Τα αντίστοιχα ονόματα για την προσομοίωση θα είναι locker_tb, και Dataflow_tb.
Σας δίνεται ο ορισμός της οντότητας

```
entity locker is
port(
  digit3, digit2, digit1, digit0   : in std_logic;
  lock_out : out std_logic);
end locker;
```

Το digit0 αντιστοιχεί στο λιγότερο σημαντικό bit του κωδικού ενώ το digit3 στο πιο σημαντικό bit (στο παράδειγμά μας digit0=’1’ και digit3=’0’). Γράψτε την αρχιτεκτονική που αντιστοιχεί στον ΑΜ σας. Εμφανίστε το RTL διάγραμμα, κάντε τη σύνθεση, εμφανίστε το διάγραμμά της (Schematic), κάντε το ίδιο για την υλοποίηση, προγραμματίστε την κάρτα fpga.

VHDL – Παράδειγμα

Άσκηση – Συσχέτιση port με FPGA-1

Είσοδοι	DIP Switch
digit3	SW3
digit2	SW2
digit1	SW1
digit0	SW0

Έξοδοι	LED
lock_out	LD0

VHDL – Παράδειγμα

Άσκηση – Συσχέτιση port με FPGA-2 (Αρχείο constraints: locker.xdc)

```
# ZedBoard Pin Assignments
##############################
# On-board Slide Switches  #
##############################

set_property -dict { PACKAGE_PIN F21   IOSTANDARD LVCMOS33 } [get_ports { digit3 }];
set_property -dict { PACKAGE_PIN H22   IOSTANDARD LVCMOS33 } [get_ports { digit2 }];
set_property -dict { PACKAGE_PIN G22   IOSTANDARD LVCMOS33 } [get_ports { digit1 }];
set_property -dict { PACKAGE_PIN F22   IOSTANDARD LVCMOS33 } [get_ports { digit0 }];

##############################
# On-board led           #
##############################
set_property -dict { PACKAGE_PIN T22   IOSTANDARD LVCMOS33 } [get_ports { lock_out }];
```

VHDL – Παράδειγμα

Πραγματικό πρόβλημα – Βήματα επίλυσης

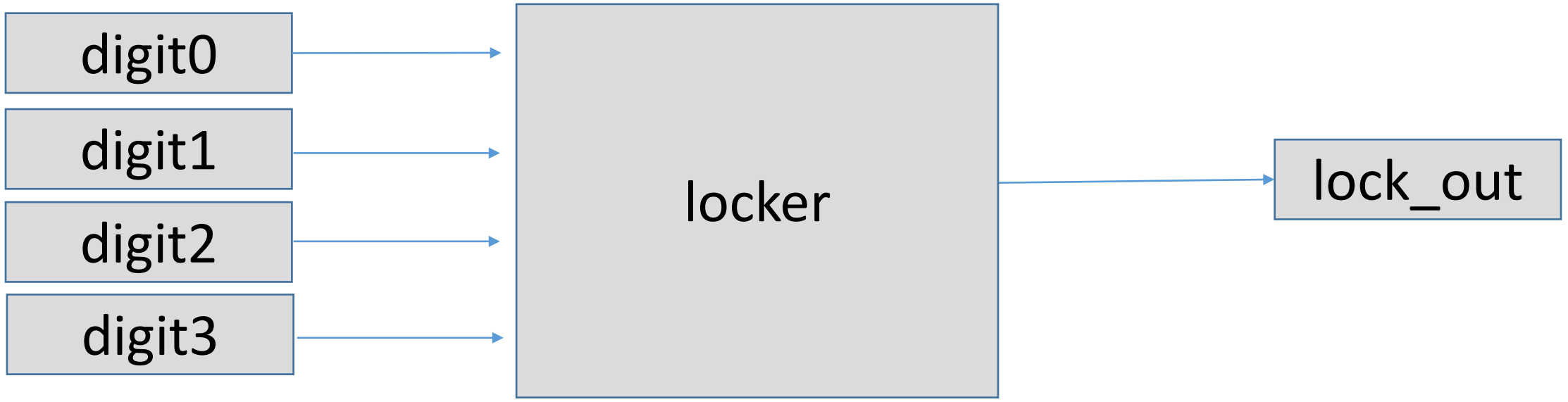
1. Δημιουργία νέου project
2. Δημιουργία Entity – Εντοπισμός Input/Output του συστήματος
3. Εύρεση πίνακα αληθείας για κάθε έξοδο του συστήματος (**αν χρειάζεται**)
4. Δημιουργία Architecture – Θα έχετε τουλάχιστον τόσες εντολές όσες είναι και οι έξοδοι του συστήματος. Κάθε μία εντολή αντιστοιχεί σε μία έξοδο.
5. Δημιουργία RTL αναπαράστασης
6. Σύνθεση
7. Υλοποίηση

Προγραμματισμός κάρτας (Έγινε μόνο στο Εργαστήριο)

8. **Προσομοίωση (Παρουσιάζεται μόνο στις διαφάνειες)**

VHDL – Παράδειγμα

Απλοποιημένη μορφή κυκλώματος – Είσοδοι/Έξοδοι



VHDL – Παράδειγμα

Βήμα 2: Περιγραφή Οντότητας

```
entity locker is

port(
   digit3, digit2, digit1, digit0   : in std_logic;
   lock_out : out std_logic);

end locker;
```

VHDL – Παράδειγμα

Βήμα 3: Πίνακας αληθείας κυκλώματος

Πίνακας Αληθείας
True Table
για ΑΜ που λήγει σε
5 =>0101

Είσοδοι				Έξοδοι
Digit3	Digit2	Digit1	Digit0	lock_out
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	**1**	**0**	**1**	**1**
0	1	1	0	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Υπάρχει μία μόνο γραμμή με ‘1’ και άρα η συνάρτηση του lock_out αντιστοιχεί σε ένα μόνο ελαχιστόρο (γινόμενο)
lockout= !Digit3*Digit2*!Digit1*Digit0

Η ανωτέρω παράσταση σε VHDL είναι:
lockout<= not Digit3 and Digit2 and not Digit1 and Digit0;

VHDL – Παράδειγμα

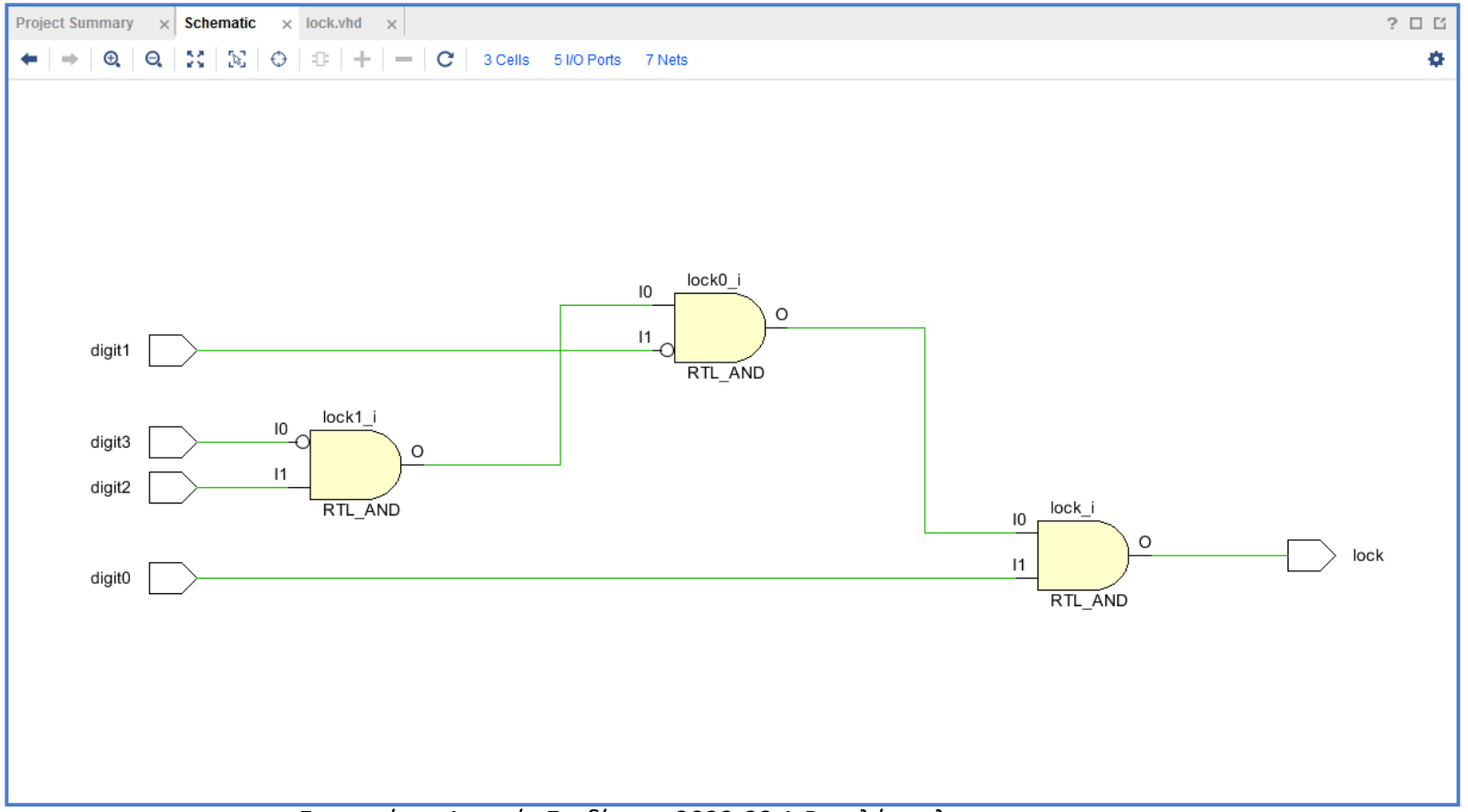
Βήμα 4: Περιγραφή Αρχιτεκτονικής

Αρχιτεκτονική για ΑΜ που λήγει σε 5

```
lock_out<=not digit3 and digit2 and not digit1 and digit0;
ή
lock<_out=(not digit3) and digit2 and (not digit1) and digit0;
```

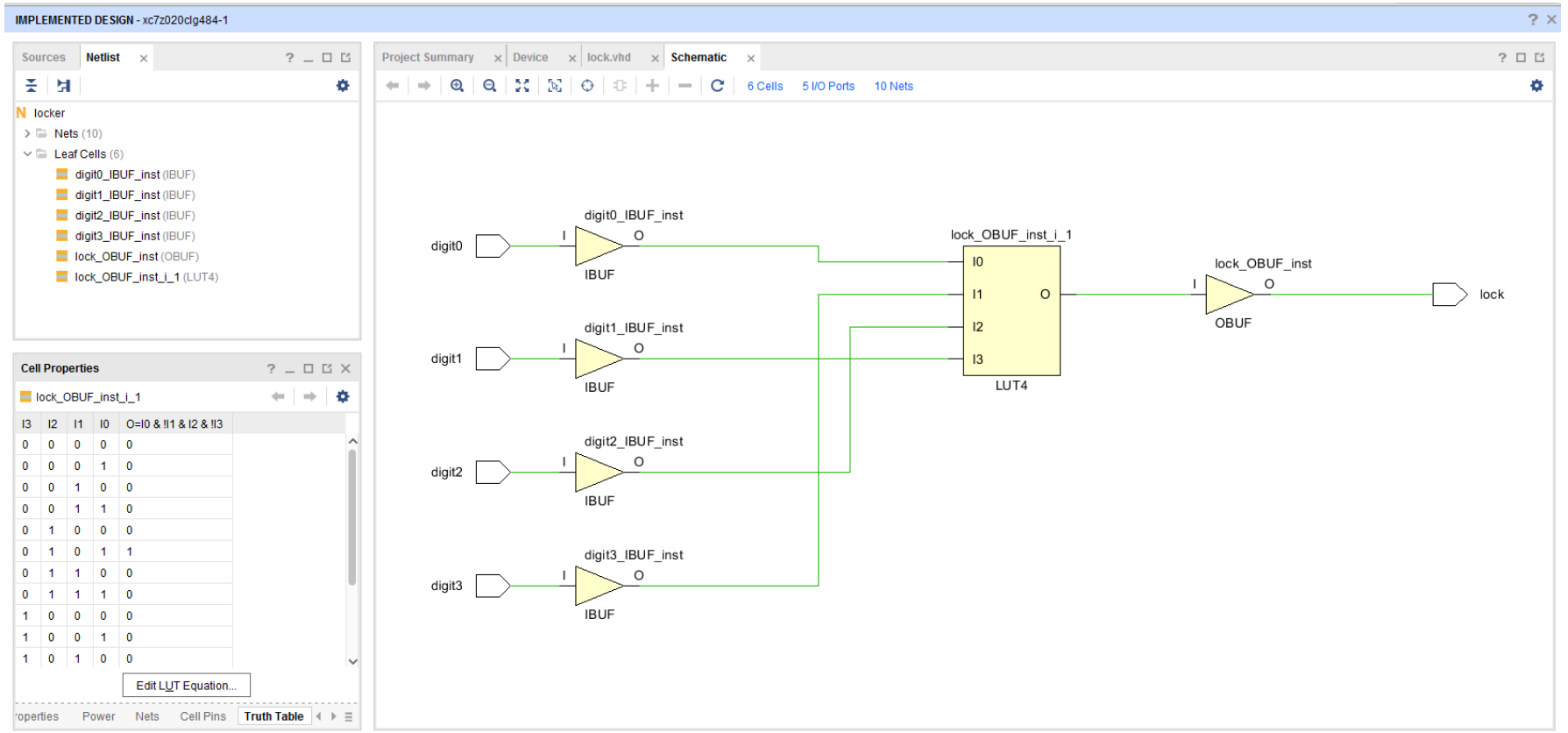
VHDL – Παράδειγμα

Βήμα 5: RTL Analysis



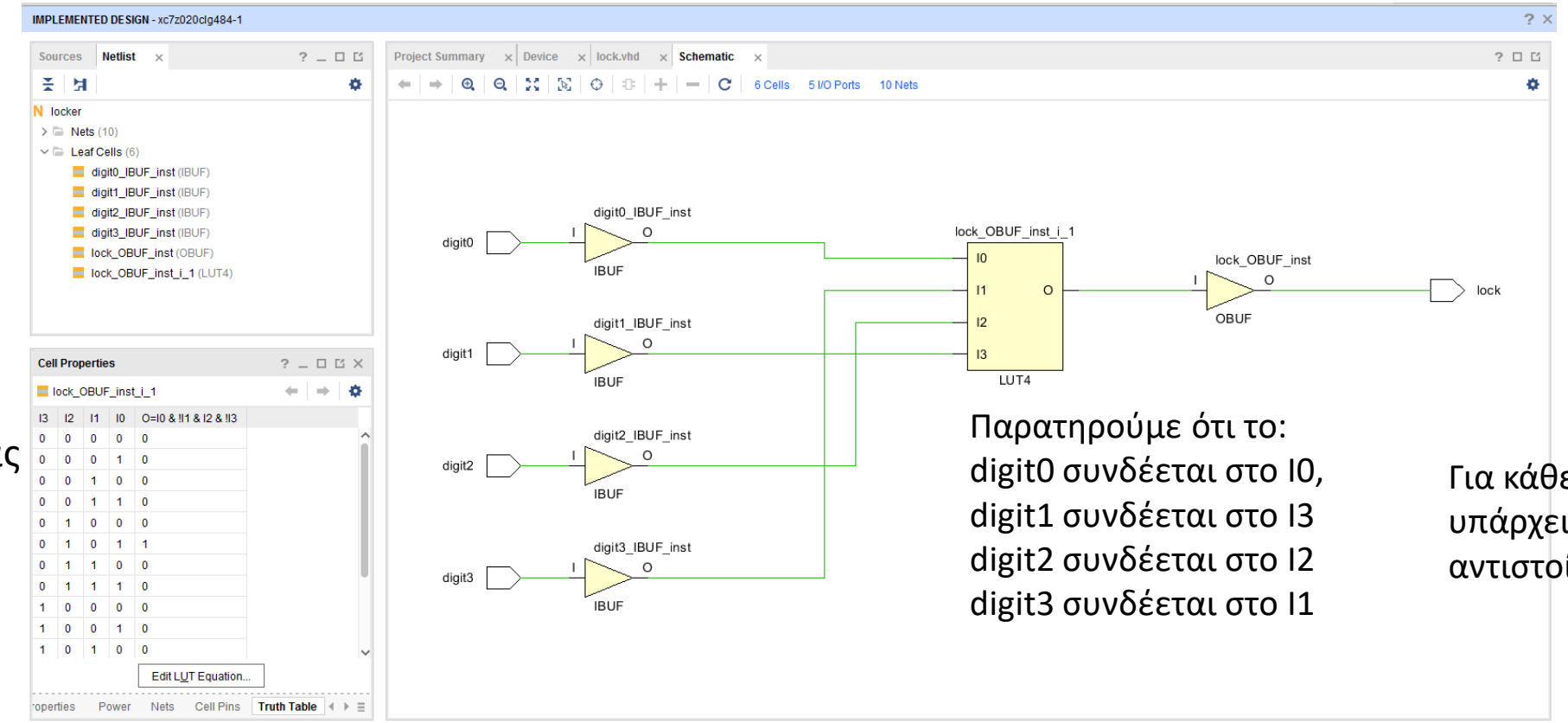
VHDL – Παράδειγμα

Βήμα 6: Synthesis



VHDL – Παράδειγμα

Βήμα 7a: Implementation



Πίνακας Αληθείας

Για κάθε ψηφίο 0-9 θα υπάρχει άλλη αντιστοίχιση

VHDL – Παράδειγμα

Βήμα 7b: Implementation: Modified Truth Table (αντιστοιχίες Digit <=> I στο LUT)

Για κάθε Digit υπάρχει το αντίστοιχο I του LUT

Πίνακας Αληθείας True Table

για ΑΜ που λήγει σε 5 =>0101

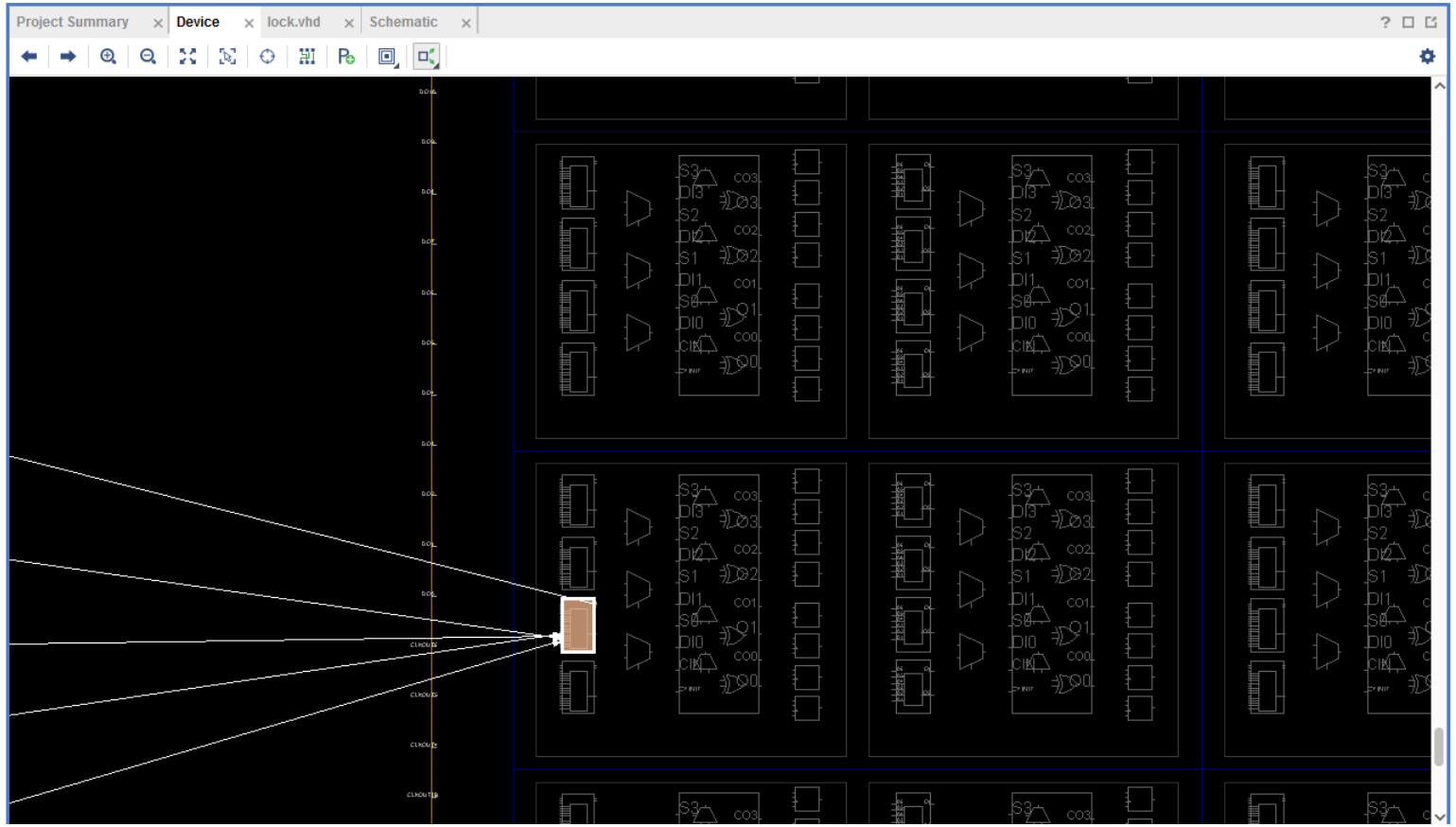
Είσοδοι				Έξοδοι
Digit3/I1	Digit2/I2	Digit1/I3	Digit0/I0	lock_out
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	**1**	**0**	**1**	**1**
0	1	1	0	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Ο αρχικός Πίνακας Αληθείας και ο αντίστοιχος του LUT είναι ίδιοι

lock_out= !Digit3*Digit2*!Digit1*Digit0
ή στο LUT
O=!I1*I2*!I3*I0

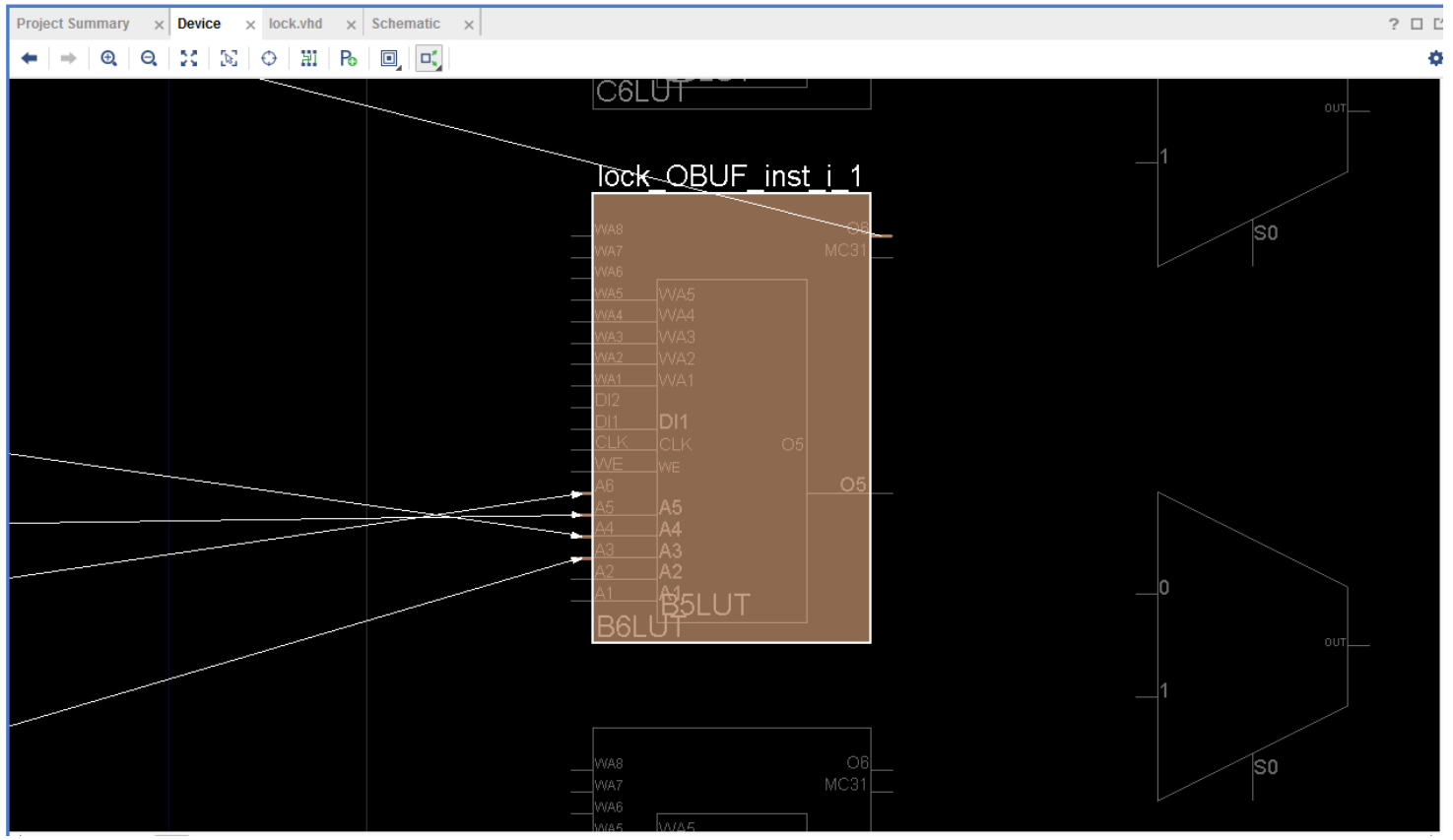
VHDL – Παράδειγμα

Βήμα 7c: Implementation – Design: LUT on Board



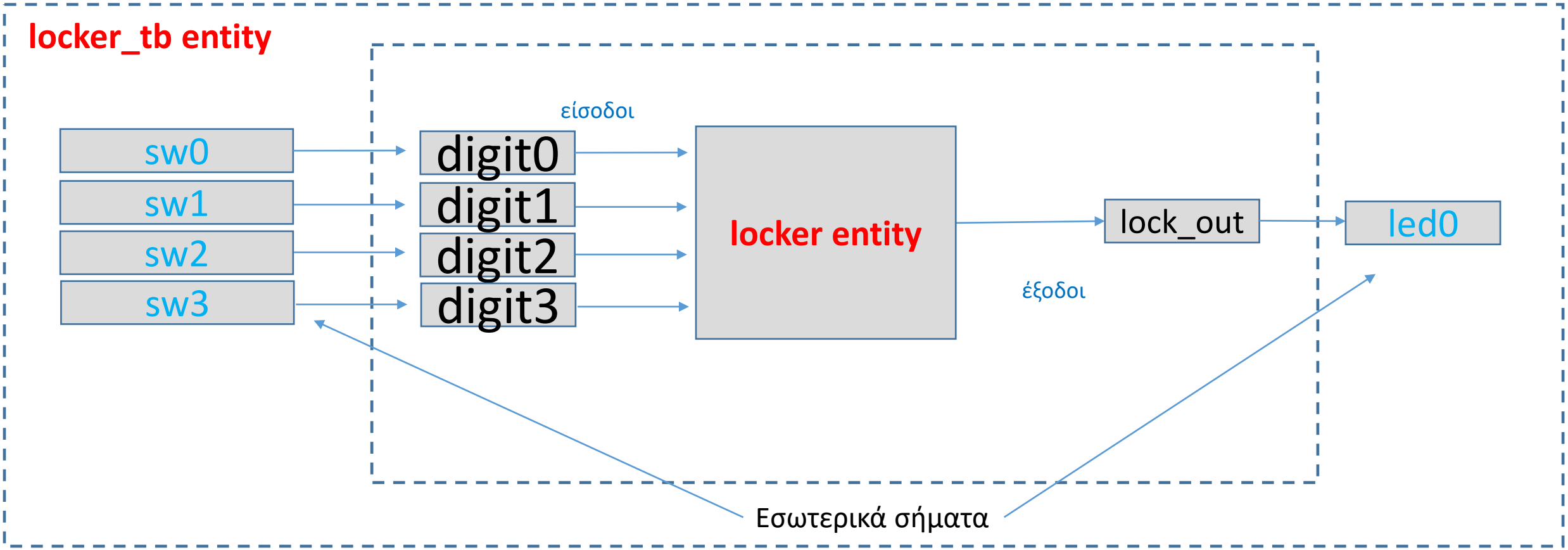
VHDL – Παράδειγμα

Βήμα 7d: Implementation – Design: LUT on Board



VHDL – Παράδειγμα

Βήμα 8a: Simulation



VHDL – Παράδειγμα

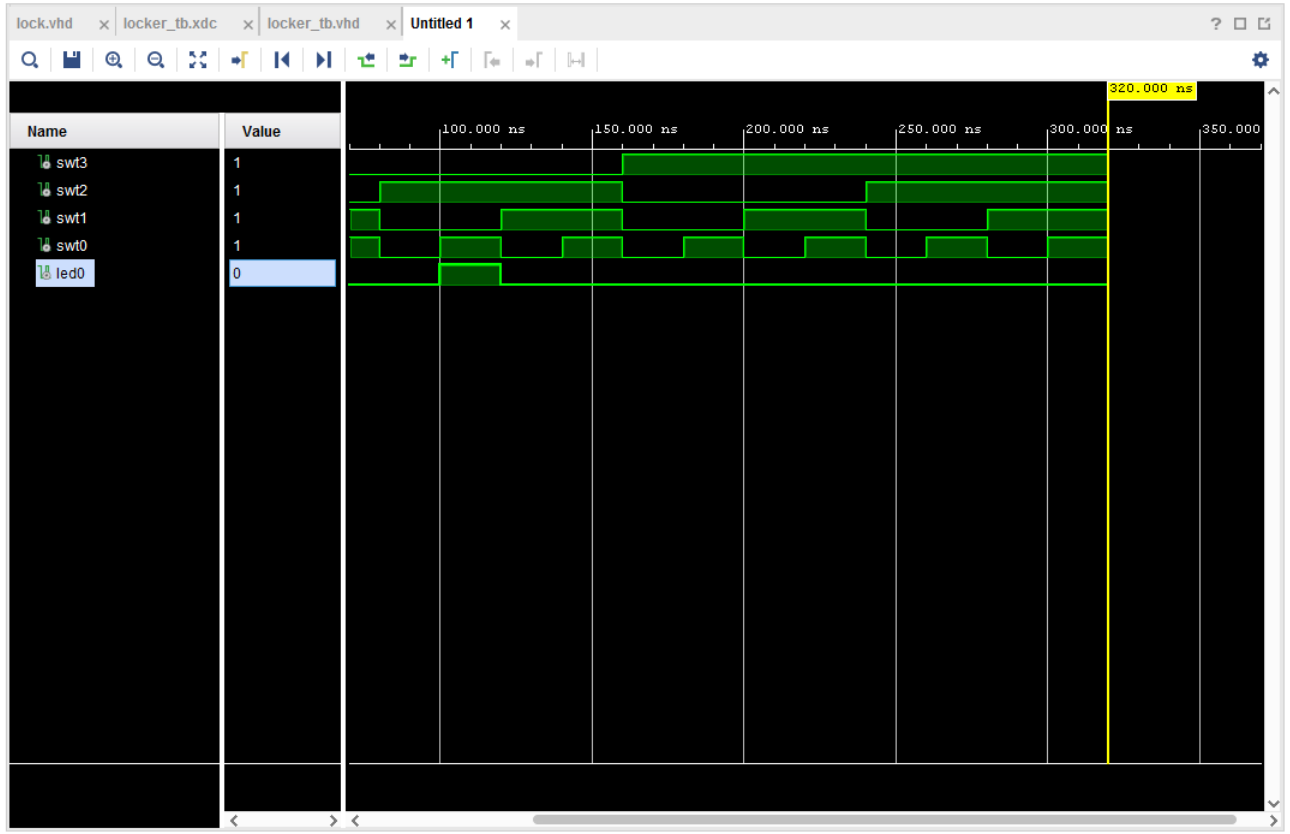
Βήμα 8b: Simulation

```
LIBRARY ieee; USE ieee.std_logic_1164.ALL;

entity locker_tb IS
end locker_tb;

architecture behavior OF locker_tb IS
    -- Component Declaration for the Unit Under Test (UUT)
    component locker
   port(
       digit3, digit2, digit1, digit0   : in std_logic;
                                lock_out : out std_logic);
 end component;

signal sw3, sw2, sw1, sw0 : std_logic; -- Input
signal led_0 : std_logic;                    --Output

begin
-- Instantiate the Unit Under Test (UUT)
  uut: locker PORT MAP (digit0 => sw0,digit1 => sw,digit2 => sw2,digit3 => sw3,
                                      lock_out => led_0);
```

```
 -- Test process
test_proc: process
begin
--
sw3<='0';sw2<='0';sw1<='0';sw0<='0';wait for 20 ns;
sw3<='0';sw2<='0';sw1<='0';sw0<='1';wait for 20 ns;
sw3<='0';sw2<='0';sw1<='1';sw0<='0';wait for 20 ns;
sw3<='0';sw2<='0';sw1<='1';sw0<='1';wait for 20 ns;
sw3<='0';sw2<='1';sw1<='0';sw0<='0';wait for 20 ns;
sw3<='0';sw2<='1';sw1<='0';sw0<='1';wait for 20 ns;
sw3<='0';sw2<='1';sw1<='1';sw0<='0';wait for 20 ns;
sw3<='0';sw2<='1';sw1<='1';sw0<='1';wait for 20 ns;
sw3<='1';sw2<='0';sw1<='0';sw0<='0';wait for 20 ns;
sw3<='1';sw2<='0';sw1<='0';sw0<='1';wait for 20 ns;
sw3<='1';sw2<='0';sw1<='1';sw0<='0';wait for 20 ns;
sw3<='1';sw2<='0';sw1<='1';sw0<='1';wait for 20 ns;
sw3<='1';sw2<='1';sw1<='0';sw0<='0';wait for 20 ns;
sw3<='1';sw2<='1';sw1<='0';sw0<='1';wait for 20 ns;
sw3<='1';sw2<='1';sw1<='1';sw0<='0';wait for 20 ns;
sw3<='1';sw2<='1';sw1<='1';sw0<='1';wait for 20 ns;
end process;
end behavior;
```

VHDL – Παράδειγμα

Βήμα 8c: Simulation



dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

Εργαστήριο Λογικής Σχεδίασης

Τύποι Σημάτων – Δομή Process – Τύποι Αρχιτεκτονικής

Βασιλόπουλος Διονύσης

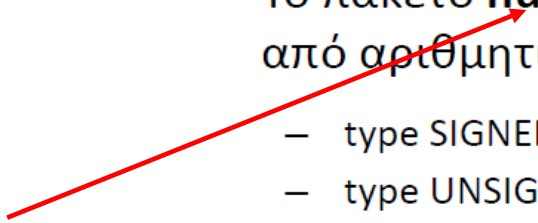
ΕΤΕΠ Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

VHDL – Τύποι σημάτων

Signed-Unsigned αριθμοί

- Το πακέτο **numeric_std** παρέχει τους τύπους signed και unsigned και μια σειρά από αριθμητικές λειτουργίες καθώς και συναρτήσεις μετατροπής
  - type SIGNED is array (NATURAL range <>) of STD_LOGIC;
  - type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;



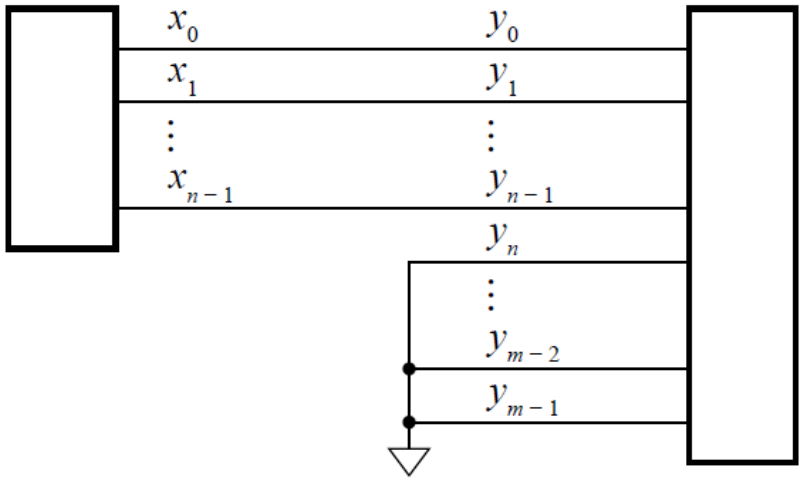
- **signal** x: **signed** (7 **downto** 0);
  - Αναπαριστά τους προσημασμένους αριθμούς (2's complement) με 8 bit
- **signal** y: **unsigned** (3 **downto** 0);
  - Αναπαριστά τους μη-προσημασμένους (θετικούς) αριθμούς με 4 bit
- Δηλώνονται σαν διανύσματα όπως ο τύπος std_logic_vector
- Επιτρέπουν την εκτέλεση αριθμητικών λειτουργιών
  - Σε αντίθεση με τα διανύσματα τύπου std_logic_vector

VHDL – Unsigned

Επέκταση μηδενός

- Για επέκταση αριθμού από *n* bit σε *m* bit
  - Πρόσθεση αρχικών bit 0
  - π.χ., 72_{10} = 1001000 = 000001001000

Σε όλες τις εντολές όσα bit είναι στο δεξί μέλος τόσο ακριβώς πρέπει να είναι και στο αριστερό




```
signal x : unsigned(3 downto 0);
signal y : unsigned(7 downto 0);

y <= "0000" & x;
```

concatenation

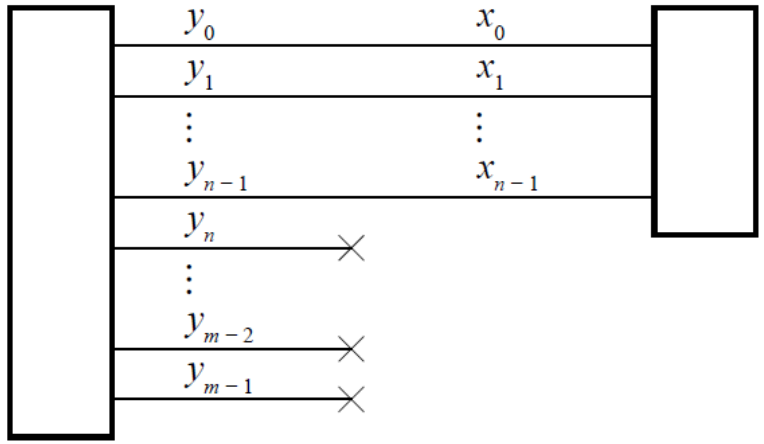
```
y <= resize(x, 8);
```

VHDL – Unisigned

Αποκοπή

Για αποκοπή από *m* bit σε *n* bit

- Απορρίπτουμε τα αριστερότερα bit
- Η τιμή διατηρείται εφόσον τα bit που απορρίπτονται είναι 0
- Το αποτέλεσμα είναι το *x* mod 2^n



Έστω: signal y : unsigned(7 downto 0);
signal x : unsigned(3 downto 0);

```
x <= y(3 downto 0);
```

```
x <= resize(y, 4);
```

Ποια είναι πιο ευέλικτη;

x <= resize(y, x'length)

Και για επέκταση και για αποκοπή.

VHDL – Unsigned

Πρόσθεση

```
library ieee; use ieee.numeric_std.all;
...
signal a, b, s: unsigned(7 downto 0); …
s <= a + b;
```

+: Παράγει αποτέλεσμα ίδιου μήκους.
Δεν ανιχνεύει υπερχείλιση
Θα μπορούσα με προφανώς να έχουμε και - .

Επέκταση με μηδενικά στα a και b κατά 1-bit
Το c είναι το MSB του αποτελέσματος

```
signal tmp_result : unsigned(8 downto 0);
signal c : std_logic;
…
tmp_result <= ('0' & a) + ('0' & b);
c <= tmp_result(8);
s <= tmp_result(7 downto 0);
```

Λύση
Χρειαζόμαστε ένα επιπλέον bit για το κρατούμενο (**C**arry)

VHDL – Unsigned

Αφαίρεση

```
library ieee; use ieee.std_logic_1164.all, ieee.numeric_std.all;
entity adder_subtracter is
  port ( x, y          : in unsigned(11 downto 0);
         s             : out unsigned(11 downto 0);
         mode          : in std_logic;
         ovf_unf       : out std_logic );
end entity adder_subtracter;

architecture behavior of adder_subtracter is
signal s_tmp : unsigned(12 downto 0);
begin
  s_tmp <= ('0' & x) + ('0' & y) when mode = '0' else
           ('0' & x) - ('0' & y);
  s <= s_tmp(11 downto 0);
  ovf_unf <= s_tmp(12);
end architecture behavior;
```

Υπό συνθήκη

VHDL – Unsigned

Increments

- Απλά, πρόσθεση ή αφαίρεση του 1

```
signal x, s: unsigned(15 downto 0);
...

s <= x + 1;  -- increment x

s <= x - 1;  -- decrement x
```

- Σημείωση: 1 (ακέραιος), όχι '1' (bit)

VHDL – Unsigned

Γινόμενο

- Μεγαλύτερο αποτέλεσμα για τελεστέους των n bit :

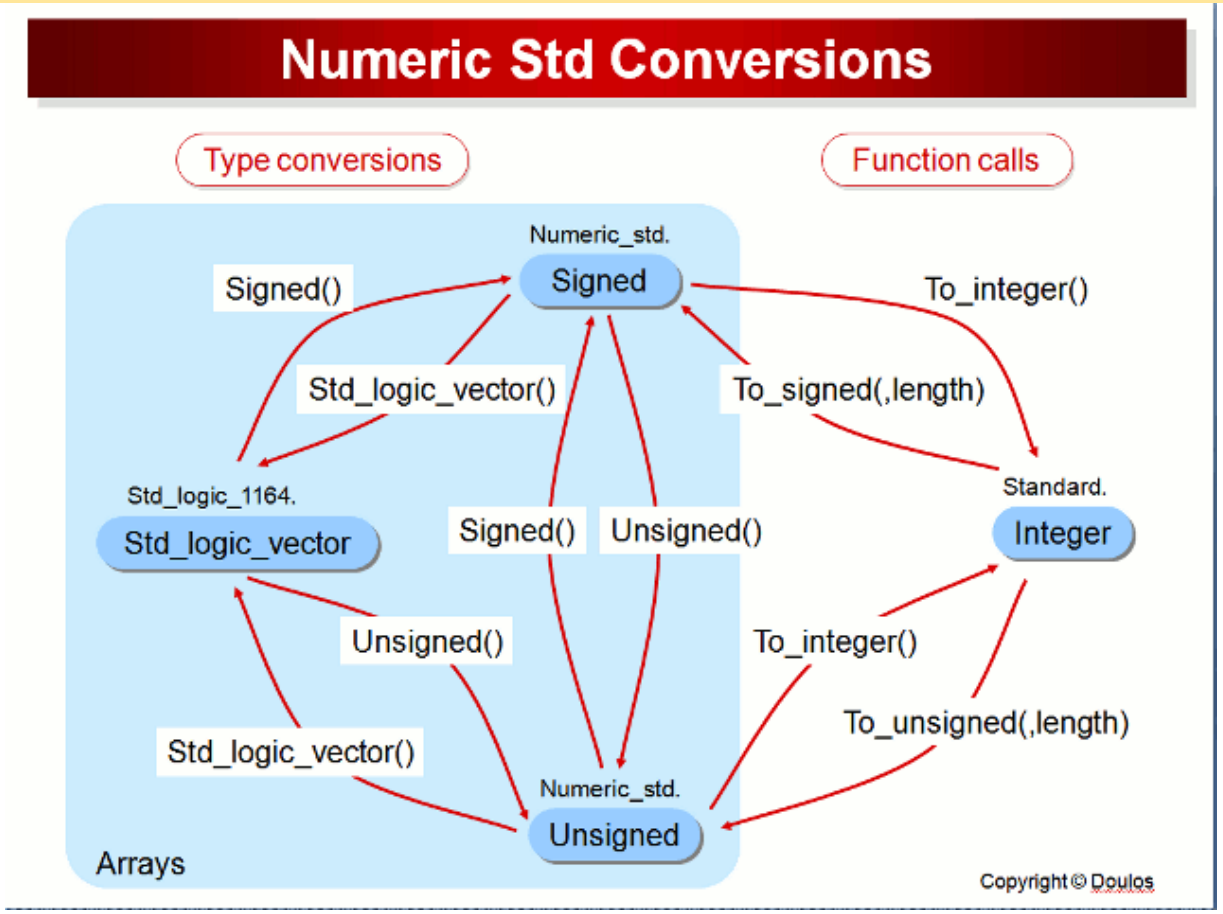
$$(2^n-1)(2^n-1)=2^{2n}-2^n-2^n+1=2^{2n}-\left(2^{n+1}-1\right)$$

  - Απαιτεί 2^{2n} bit για αποφυγή υπερχείλισης
  - Γινόμενο τελεστέων των n bit και m bit
    - Απαιτεί $n+m$ bit

```
signal x : unsigned(7 downto 0);
signal y : unsigned(13 downto 0);
signal p : unsigned(21 downto 0);
…
p <= x * y;
```

VHDL – Τύποι σημάτων

Μετατροπές



VHDL – Τύποι σημάτων

Μετατροπές

Έστω οι δηλώσεις:
signal x: std_logic_vector(3 downto 0);
signal y: unsigned(3 downto 0);

Για να μετατρέψουμε τον ένα τύπο στον άλλο:
x<=std_logic_vector(y);
ή
y<=unsigned(x);

Έστω οι δηλώσεις:
signal x: std_logic_vector(5 downto 0);
signal y: unsigned(3 downto 0);

Για να μετατρέψουμε τον ένα τύπο στον άλλο:
x<=std_logic_vector(resize(y,x'length));
ή
y<=unsigned(x(3 downto 0));

σωστό π.χ. και το y<=unsigned(x(4 downto 1));

Όσα bit είναι αριστερά μιας εντολής, τόσα πρέπει να είναι και δεξιά

VHDL – Process

- **process**: η διεργασία είναι μία ομάδα από εντολές που εκτελούνται ακολουθιακά

Sensitivity list

```
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;
```

VHDL – Process

Variables (Μεταβλητές)

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)

VHDL – Process

Variables (Μεταβλητές)

```
variable my_number: natural;

variable my_logic: std_logic:='1';

variable my_vector :  std_logic_vector(3 downto 0):="1110";
```

Αρχικοποίηση τιμών. Γίνεται σε ΟΛΑ τα σήματα.

Σε process, function, procedure …

VHDL – Process

Variables (Μεταβλητές)

```
variable_name := expression;
```

ΠΡΟΣΟΧΗ
Είναι **:=** και όχι **<=**

- **variable_name**: το όνομα της μεταβλητής
- **expression**: έκφραση με σήματα, μεταβλητές και τελεστές
  - στις ακολουθιακές (sequential) 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

VHDL – Process

Variables (Μεταβλητές)

- Διαφορά μεταβλητής και σήματος μέσα σε μία διεργασία
  - η μεταβλητή παίρνει νέα τιμή **στιγμιαία** με τον τελεστή ανάθεσης **:=**, αμέσως μόλις εκτελεστεί η αντίστοιχη εντολή μέσα στη διεργασία
  - σε αντίθεση, το σήμα παίρνει νέα τιμή **με καθυστέρηση δέλτα δ_{delay}**, με τον τελεστή ανάθεσης **<=**, στο **τέλος** της εκτέλεσης της διεργασίας

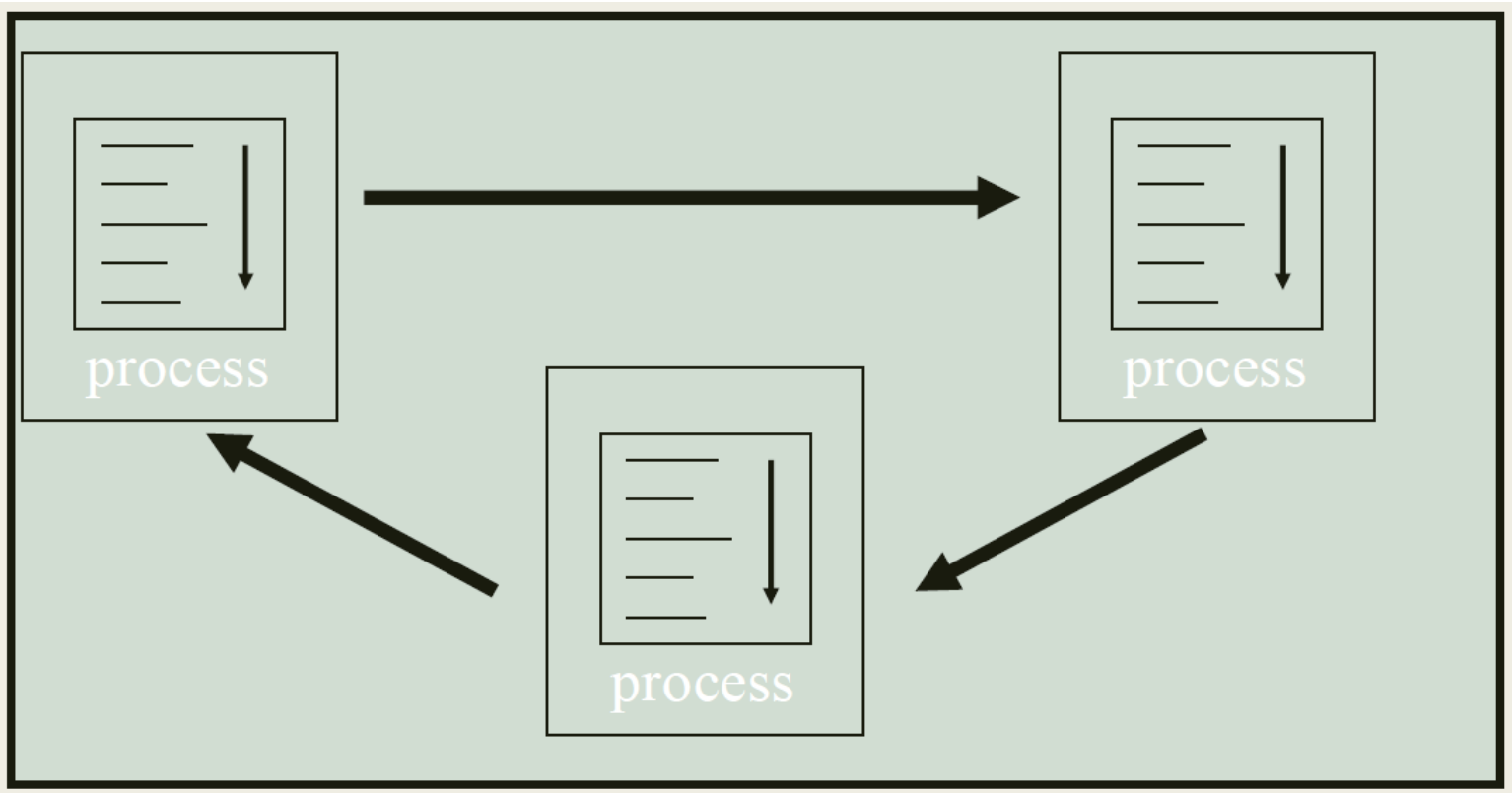
Η χρήση των μεταβλητών μειώνει σημαντικά το χρόνο της προσομοίωσης

VHDL – Process

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

VHDL – Process



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

VHDL – Process

Διαφορά στην εφαρμογή της τιμής μίας sequential εντολής ανάθεσης μεταβλητής ή σήματος μέσα σε μία διεργασία :

– η **μεταβλητή παίρνει νέα τιμή άμεσα** με τον τελεστή ανάθεσης :=, αμέσως μόλις εκτελεστεί η αντίστοιχη εντολή μέσα στη διεργασία
    * Η variable δεν χρησιμοποιείται στην υλοποίηση της 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
    * Το εσωτερικό σήμα χρησιμοποιείται στην υλοποίηση της λογικής

VHDL – Process

- Όταν **δεν υπάρχει sensitivity list** οι εντολές μέσα στο process **εκτελούνται συνέχεια**. Θα πρέπει να υπάρχει **εντολή wait** ώστε να αποδοθούν τιμές στα σήματα (όπως κάνουμε στο simulation).
- Οι εντολές μέσα στο process **εκτελούνται τουλάχιστον 1 φορά**, ακόμα και εάν υπάρχει sensitivity list.
- Οι τιμές αποδίδονται στα σήματα είτε κάθε φορά που συναντάται η εντολή **wait** μέσα στο process είτε όταν φτάνουμε στο **end** του process.
- Εάν υπάρχει wait σε σήμα που δεν αλλάζει τιμή (εκτός του process) τότε η εκτέλεση σταματάει στο συγκεκριμένο wait.
- Όλο το **process θεωρείται ΜΙΑ εντολή μέσα σε μια αρχιτεκτονική** και εκτελείται παράλληλα με πιθανές άλλες εντολές της.

VHDL – Process

```
test:process (a,b) is
begin
....
end process test;
```

```
test:process (a,b) is
begin
....
wait on a,b;
end process test;
```

Δεν το γράφουμε αλλά ισχύει

```
test:process (a,b) is
begin
....
end process test;
```

```
test:process (a,b) is
begin
....
wait on c;
end process test;
```

Το process «κολλάει» σε αυτή την εντολή

VHDL - Παράδειγμα

Πραγματικό πρόβλημα

Σε ένα Computer Room υπάρχουν δύο (2) αισθητήρες θερμοκρασίας (**Sensor_1** και **Sensor_2**) και δύο (2) κλιματιστικά (**AirCond_1** και **AirCond_2**). Το πρώτο κλιματιστικό (AirCond_1) λειτουργεί εάν τουλάχιστον ένας από τους δύο αισθητήρες ανιχνεύσουν θερμοκρασία άνω των 35 βαθμών στο computer room. Το δεύτερο κλιματιστικό (AirCond_2) λειτουργεί εάν και οι δύο αισθητήρες ανιχνεύσουν θερμοκρασία άνω των 35 βαθμών στο computer room. Θεωρείστε ότι κάθε αισθητήρας δίνει σήμα (‘1’) μόνο όταν η θερμοκρασία που ανιχνεύει γίνει μεγαλύτερη των 35 βαθμών (>35). Σε άλλη περίπτωση ο αισθητήρας στέλνεις την τιμή ‘0’. Σχεδιάστε και υλοποιήστε το λογικό κύκλωμα που περιγράφει το ανωτέρω πρόβλημα. Το όνομα του Vivado Project θα είναι Lab_1, της οντότητας θα είναι επίσης Lab_1 ενώ το όνομα της αρχιτεκτονικής Lab1_Beh.

Σχεδιάστε και υλοποιήστε το λογικό κύκλωμα που περιγράφει το ανωτέρω πρόβλημα.

VHDL - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Dataflow)

```
architecture Dataflow of CR_AC is
begin

        AirCond_1<=Sensor_1 or Sensor_2;
        AirCond_2<=Sensor_2 and Sensor_3;

end architecture Dataflow;
```

VHDL - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Dataflow)

1. Αποτελείται από απλές εντολές ανάθεσης τιμών σε σήματα
2. Κάθε εντολή εκτελείται όταν μεταβληθεί η τιμή ενός σήματος στο αριστερό μέρος.
3. Όλες οι εντολές ανάθεσης εκτελούνται ταυτόχρονα (παράλληλα)
4. Οι εντολές ανάθεσης αντιστοιχούν σε λογικές πράξεις της άλγεβρας Boole.
5. Το RTL διάγραμμα που προκύπτει είναι μία απεικόνιση της άλγεβρας Boole που εκφράζουν οι εντολές ανάθεσης σε στοιχειώδεις λογικές πύλες (AND, OR, NOT) και πολυπλέκτες.

VHDL - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Behavioral)

```
architecture behavioral of CR_AC is
begin

room:  process (Sensor_1, Sensor_2) is
          begin
               AirCond_1<=Sensor_1 or Sensor_2;
               AirCond_2<=Sensor_1 and Sensor_2;
          end process room;

end architecture Behavioral;
```

VHDL

Υλοποίηση Αρχιτεκτονικής (Behavioral)

1. Όμοιο RTL Design με το Dataflow
2. ΔΕΝ χρειάζεται να αλλάξουμε τίποτα στην προσομοίωση
3. Εισαγωγή της δομής process (με λίστα ευαισθησίας)
4. Σε περίπτωση που υπάρχει μόνο η εντολή process, τότε ουσιαστικά οι εντολές μας εκτελούνται σειριακά
5. Σε περίπτωση που υπάρχουν και απλές εντολές ανάθεσης στην αρχιτεκτονική, τότε μπορείτε να θεωρήσετε όλη τη δομή process σαν μία εντολή που εκτελείται παράλληλα με τις εντολές ανάθεσης (που είναι εκτός process)

VHDL - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Structural – Δήλωση Οντοτήτων)

```
entity OR_gate is
port(A  : in  std_logic;
     B  : in  std_logic;
     O  : out std_logic);
end entity OR_gate;

architecture Dataflow of OR_gate is
begin
   O<=A or B;
end Dataflow;
```

VHDL - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Structural – Δήλωση Οντοτήτων)

```
entity AND_gate is
port(A  : in  std_logic;
     B  : in  std_logic;
     O  : out std_logic);
end entity AND_gate;

architecture Dataflow of AND_gate is
begin
   O<=A AND B;
end Dataflow;
```

VHDL - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Structural – Δήλωση Αρχιτεκτονικής Κύριας Οντότητας)

```
architecture Structural of CR_AC is

component AND_gate is
port(A  : in  std_logic;
     B  : in  std_logic;
     O  : out std_logic);
end component AND_gate;

component OR_gate is
port(A  : in  std_logic;
     B  : in  std_logic;
     O  : out std_logic);
end component OR_gate;

begin

or_comp : OR_gate  port map (A=>Sensor_1,
B=>Sensor_2, O=>AirCond_1);
and_comp: AND_gate port map (A=>Sensor_1,
B=>Sensor_2, O=>AirCond_2);

end architecture Structural;
```

VHDL

Υλοποίηση Αρχιτεκτονικής (Structural)

1. Για την υλοποίηση της αρχιτεκτονικής χρησιμοποιούμε τη λειτουργικότητα άλλων Οντοτήτων που έχουμε ήδη υλοποιήσει.
2. Τις οντότητες που θα χρησιμοποιήσουμε τις δηλώνουμε (όνομα και port) ανάμεσα στο is και το begin της αρχιτεκτονικής. Όμως πλέον έχουν την έννοια της συνιστώσας (component).
3. Η αρχιτεκτονική πλέον αποτελείτε μόνο από εντολές που καλούν(δημιουργούν) τα components, τα οποία μπορεί και να επικοινωνούν μεταξύ τους (συνηθέστερη περίπτωση).
4. Άρα μια οντότητα μπορεί να χρησιμοποιεί άλλες οντότητες (με τη μορφή component), οι οποίες μπορεί να είναι υλοποιημένες με οποιοδήποτε από τις 3 αρχιτεκτονικές. Στη περίπτωση της structural δομής βλέπουμε ότι σχηματίζεται ένα δέντρο, τα φύλλα του οποίου είναι οντότητες. Τα φύλλα που είναι τερματικά έχουν δομή Dataflow ή Behavioral.
5. ΔΕΝ αλλάζει τίποτα στην προσομοίωση

Μπορείτε να δείτε και το: https://buzztech.in/vhdl-modelling-styles-behavioral-dataflow-structural/

VHDL - Περίληψη

- Τύπος Unsigned
- Process
- Variable
- Παράδειγμα των 3 ειδών αρχιτεκτονικής (Dataflow, Behavioral, Structural)
- Διαβάζετε τις παραγράφους 2.3, 2.4, 3.1 (θεωρία και VHDL) από Ashenden και 2.2 - 2.5, 4.2.5, 4.3, 4.4.1 (ΟΧΙ το κομμάτι της VERILOG) από το βιβλίο των Harris.
- Συνοπτικός οδηγός VHDL: https://redirect.cs.umbc.edu/portal/help/VHDL/summary_one.html

dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

Εργαστήριο Λογικής Σχεδίασης

VHDL - Εισαγωγή

Βασιλόπουλος Διονύσης

ΕΤΕΠ Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

VHDL – Signed

- Τύπος signed από το numeric_std

Απαραίτητη η χρήση της βιβλιοθήκης numeric_std

```
library ieee; use ieee.numeric_std.all;
...
s : signed(15 downto 0);
```

- Οι τύποι signed και unsigned είναι ξεχωριστοί

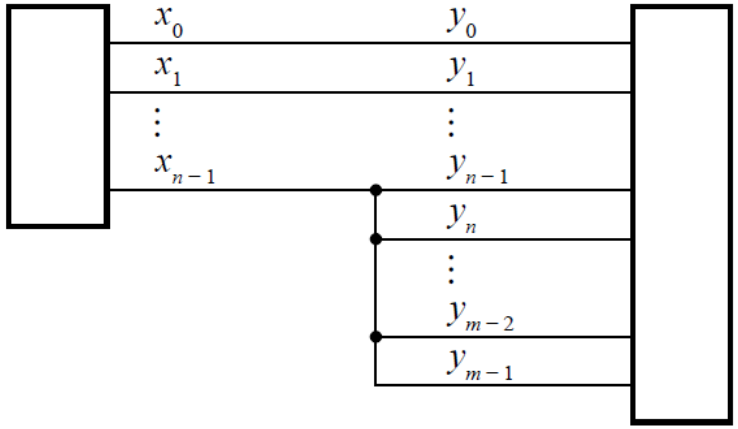
```
signal s1 : unsigned(11 downto 0);
signal s2 : signed(11 downto 0);
...
s1 <= s2; -- illegal
```

```
s1 <= unsigned(s2); -- s2 is known to be non-negative
...
s2 <= signed(s1); -- s1 is known to be less than 2**11
```

VHDL – Signed

Αλλαγή μεγέθους

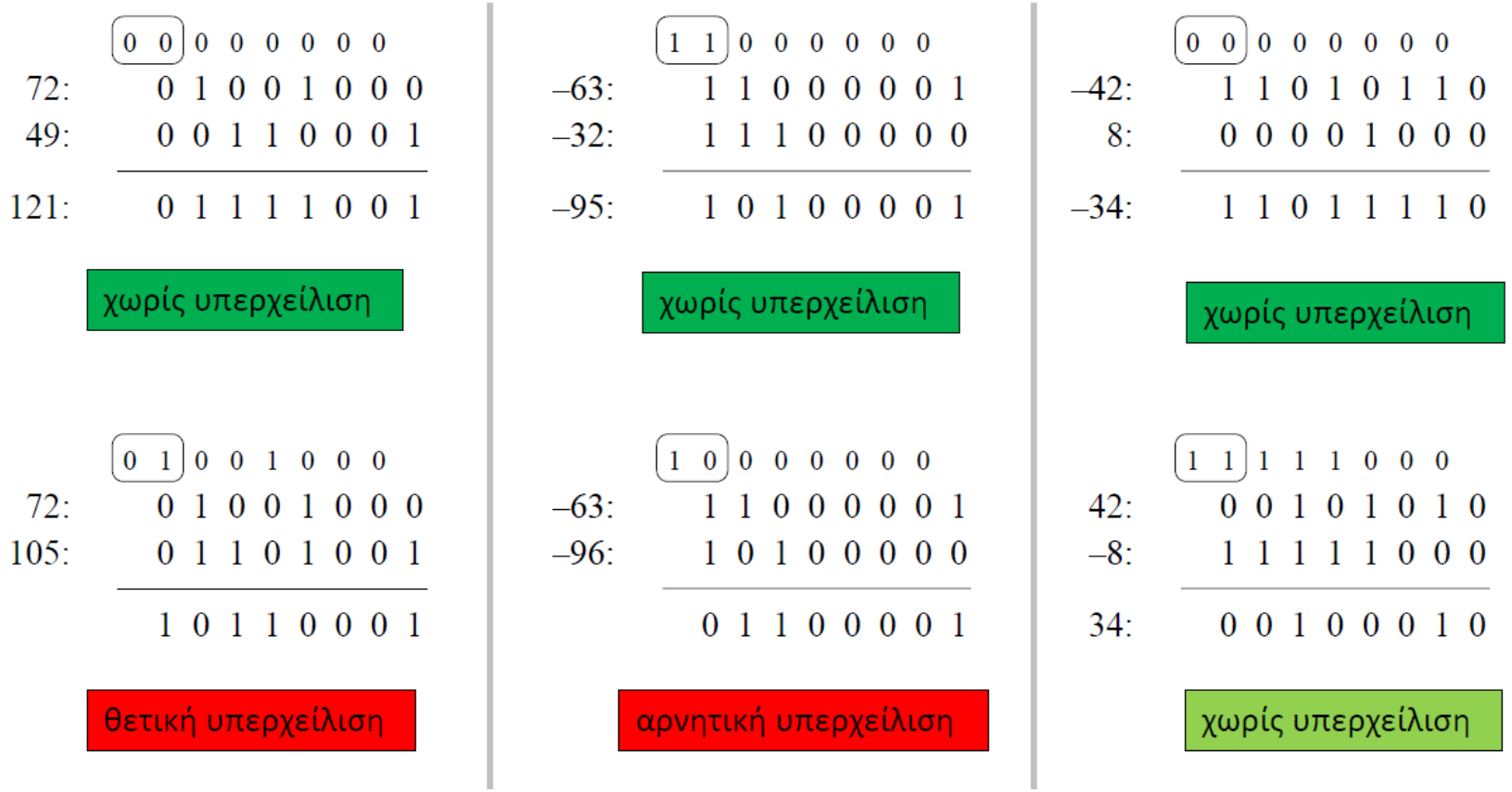
- Γενικά, για ακεραίους σε συμπλήρωμα ως προς 2
  - Επέκταση με επανάληψη του bit προσήμου
    - *Επέκταση προσήμου*
  - Αποκοπή με απόρριψη αρχικών bit
    - Τα bit που απορρίπτονται πρέπει όλα να είναι τα ίδια, και ίδια με το bit προσήμου του αποτελέσματος




```
signal x : signed (7 downto 0);
signal y : signed (15 downto 0);
...
y <= resize(x, y'length);
...
x <= resize(y, x'length);
```

VHDL – Signed

Πρόσθεση



VHDL – Signed

Πρόσθεση

- Το αποτέλεσμα του + έχει ίδιο μέγεθος με τους τελεστέους

ισχύει και για unsigned

```
signal v1, v2 : signed(11 downto 0);
signal sum : signed(12 downto 0);
...
sum <= resize(v1, sum'length) + resize(v2,sum'length);
```

- Για έλεγχο υπερχείλισης, σύγκριση προσήμων

```
signal x, y, z: signed(7 downto 0);
signal ovf : std_logic;
...
z <= x + y;
ovf <= (not x(7) and not y(7) and z(7))
       or (x(7) and y(7) and not z(7));
```

VHDL – Τελεστές (operators)

Signed: Πολλαπλασιασμός

- Πολλαπλασιασμός με 2^k
  - Αριστερή λογική ολίσθηση (όπως για απρόσημους)

Το πρόσημο πρέπει να παραμείνει

- Διαίρεση με 2^k
  - *Δεξιά αριθμητική ολίσθηση*
  - Απόρριψη των k λιγότερο σημαντικών bit, και εισαγωγή k αντιγράφων του bit προσήμου στο περισσότερο σημαντικό άκρο
  - π.χ., s = "11110011" -- –13
    shift_right(s, 2) = "11111100" -- –13 / 2^2

VHDL – Τύποι σημάτων

Οι ακόλουθοι τύποι υποστηρίζονται εξ' ορισμού στην VHDL.

- bit : τιμές MONO '0' και '1' {0,1}
- Boolean: τιμές Αληθής/Ψευδής {true, false}
- Character: ASCII character (είναι 256 σε πλήθος}

Enumerated type: Συγκεκριμένες τιμές

- bit_vector: κατ' αναλογία με std_logic_vector
- integer: -2^{31} έως 2^{31} -1 (signed binary number range: $[-2^{n-1}, 2^{n-1}-1]$, 32bit word arch.)
  - integer range 0 to 1000
- natural: 0 έως 2^{31} -1
- positive: 1 έως 2^{31} -1

Subtypes: Υποτύποι του integer

Υπάρχουν και άλλα Type/Subtypes είτε στη standard βιβλιοθήκη είτε σε άλλες (ieee.numeric_std.all: signed/unsigned, ieee.std_logic_1164: std_logic, std_logic_vector)

VHDL – Τελεστές (operators)

Λογικοί Τελεστές

Operator	Operation
not	Logical negation
and	Logical AND
nand	Logical NAND
or	Logical OR
nor	Logical NOR
xor	Logical Exclusive-OR
xnor	Logical Exclusive-NOR

Εφαρμόζονται σε bit

VHDL – Τελεστές (operators)

Αριθμητικοί Τελεστές

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
mod	Modulus
rem	Remainder
abs	Absolute value
**	Exponential

Εφαρμόζονται σε σήματα τύπου integer/signed/unsigned

VHDL – Τελεστές (operators)

Τελεστές σύγκρισης (Relational Operators)

Operator	Returns true if the comparison is:
=	Equal
/=	Not equal
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal

VHDL – Τελεστές (operators)

Τελεστές Ολίσθησης (Shift/Rotate Operators)

Operator	Operation
sll	Shift left logical
srl	Shift right logical
sla	Shift left arithmetic
sra	Shift right arithmetic
rol	Rotate left
ror	Rotate right

Εφαρμόζονται σε vector

VHDL – Τελεστές (operators)

Συναρτήσεις Ολίσθησης (Πακέτο numeric_std)

shift_left()
shift_right()
-

Αριθμητική Ολίσθηση

rotate_left()
rotate_right()

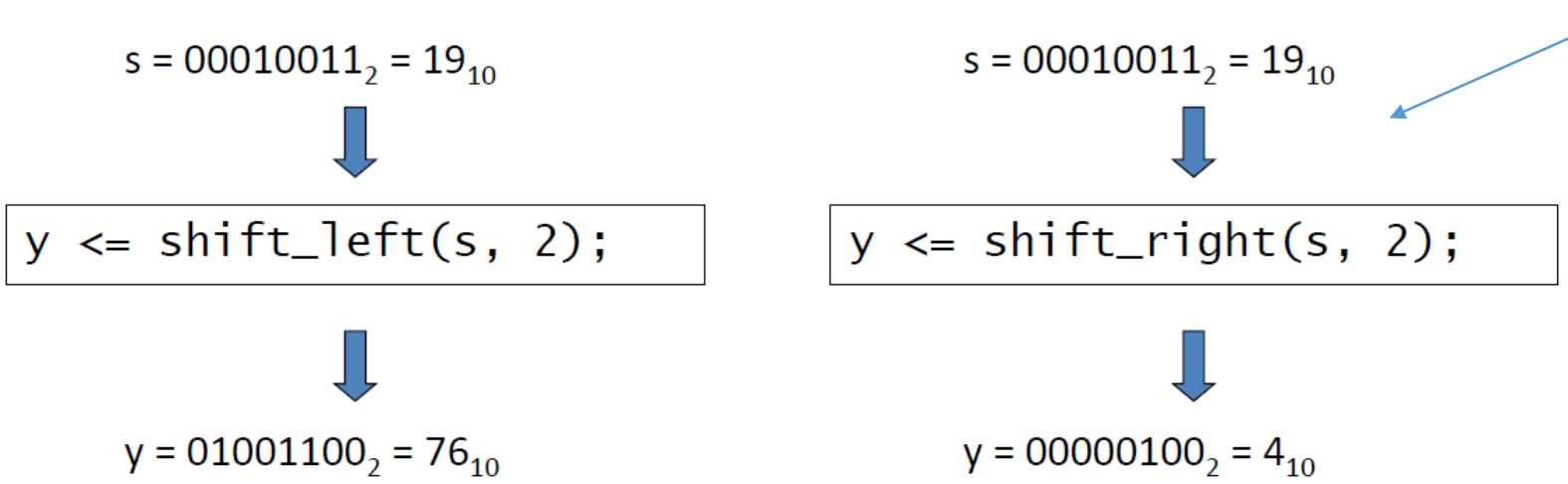
Εφαρμόζονται σε signed/unsigned

VHDL – Τελεστές (operators)

Unsigned: Ολίσθηση

- Λειτουργίες shift_left και shift_right
  - Αποτέλεσμα ίδιου μεγέθους με τον τελεστέο

Αποκοπή προς τα κάτω



VHDL – Τελεστές (operators)

VHDL'93 Vivado 2019.2

Για το std_logic_vector
ΔΕΝ υπάρχει κάποια συνάρτηση.

Εάν όμως ενεργοποιήσουμε τη VHDL 2008 τότε μπορούμε να χρησιμοποιήσουμε τα sll, srl, rol, ror.

signed/unsigned

```
x<=a sll 2;
y<=a srl 2;
-----------------
r<=shift_left(a,2);
t<=shift_right(a,2);
-----------------
q<=a rol 2;
u<=a ror 2;
i<=rotate_left(a,2);
o<=rotate_right(a,2)
```

Εάν όμως ενεργοποιήσουμε τη VHDL 2008 τότε μπορούμε να χρησιμοποιήσουμε και τα sla, sra.

VHDL – Τελεστές (operators)

VHDL'93 Vivado 2022.1

std_logic_vector

```
x<=a sll 2;
y<=a srl 2;
-----------------
q<=a rol 2;
u<=a ror 2;
```

signed/unsigned

```
x<=a sll 2;
y<=a srl 2;
-----------------
z<=a sla 2;
w<=a sra 2;
r<=shift_left(a,2);
t<=shift_right(a,2);
-----------------
q<=a rol 2;
u<=a ror 2;
i<=rotate_left(a,2);
o<=rotate_right(a,2)
```

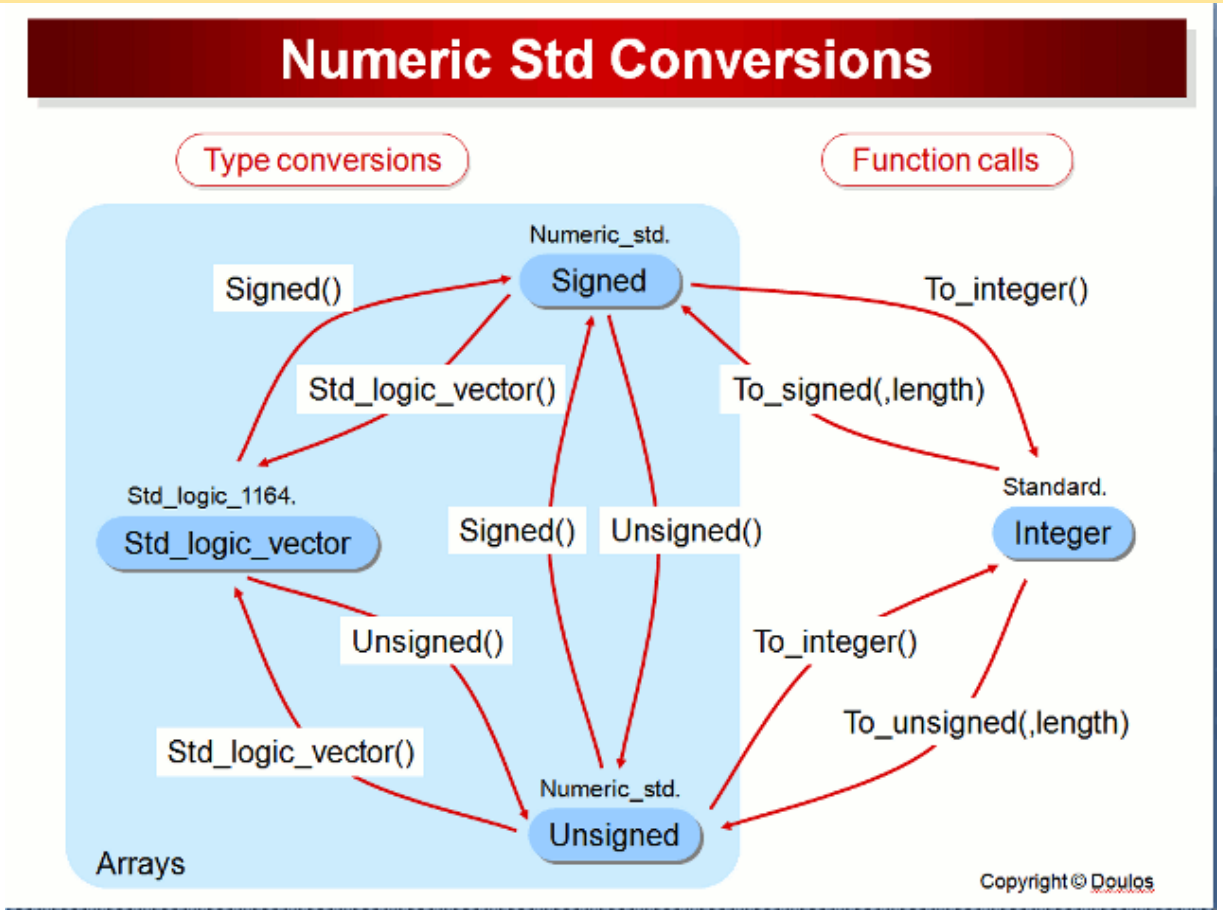
VHDL – Τελεστές (operators)

Προτεραιότητα

	Τελεστής	Σημασία
Υψηλότερη	`not`	NOT
	`*` `/` `mod` `rem`	MUL, DIV, MOD, REM
	`+` `-`	PLUS, MINUS
	`rol` `ror` `srl` `sll`	Περιστροφή, λογική ολίσθηση
	`<` `<=` `>` `>=`	Σχετική σύγκριση
	`=` `/=`	Σύγκριση ισότητας
Χαμηλότερη	`and` `or` `nand` `nor` `xor` `xnor`	Λογικές πράξεις (εκτελούνται από αριστερά προς τα δεξιά)

VHDL – Τύποι σημάτων

Μετατροπές



VHDL – Τελεστές (operators)

Μπορείτε να διαβάσετε και:
https://www.nandland.com/vhdl/examples/example-shifts.html

Μπορείτε να δείτε επίσης και τη συμπεριφορά τους σε bit_vector:
https://hdlworks.com/hdl_corner/vhdl_ref/VHDLContents/BitVector.htm

Συνοπτικός οδηγός VHDL
https://www.ics.uci.edu/~jmoorkan/vhdlref/vhdl.html

Για διαφορές mod/rem:
https://stackoverflow.com/questions/25848879/difference-between-mod-and-rem-operators-in-vhdl

VHDL – Conditional & Selected assignment

Πρόβλημα

Σε ένα Computer Room υπάρχουν 3 κλιματιστικά και ένας αισθητήρας θερμοκρασίας (θερμόμετρο). Εάν το θερμόμετρο δείξει θερμοκρασία κάτω των 25 βαθμών τότε δεν λειτουργεί το κλιματιστικό. Αν η θερμοκρασία είναι μέχρι 30 βαθμούς λειτουργεί το 1ο. Αν η θερμοκρασία είναι μέχρι 45 βαθμούς λειτουργεί και το 2ο. Εάν είναι πάνω από 45 λειτουργεί και το 3ο. Θεωρείστε ότι η σειρά λειτουργίας των κλιματιστικών είναι σαφώς ορισμένη και δεν μας απασχολεί για το πρόβλημά μας. Είσοδος του συστήματος το σήμα temperature και έξοδος το σήμα action (έχει 4 δυνατές τιμές).

VHDL – Selected assignment (Dataflow)

When

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;USE ieee.numeric_std.ALL;

entity testing is
port (temperature:  in std_logic_vector(5 downto 0);
        action          :  out std_logic_vector(1 downto 0)
);
end entity testing;

architecture Dataflow of testing is
signal temp :integer;
begin

temp<= to_integer(unsigned(temperature));
action<="00" when temp<25 else
      "01" when (temp>=25 and temp<30) else
      "10"  when (temp>=30 and temp<45) else
      "11";


end architecture Dataflow ;
```

max=2^5-1=63

action:
00: Δεν δουλεύει τίποτα
01: Δουλεύει 1 AC
10: Δουλεύουν 2 AC
11: Δουλεύουν και τα 3 AC

Λογική Συνθήκη

Μία εντολή: Ανάθεση τιμής σε ένα σήμα

VHDL – Selected assignment (Dataflow)

With Select

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;USE ieee.numeric_std.ALL;



entity testing is
port (temperature:  in std_logic_vector(5 downto 0);
      action       :  out std_logic_vector(1 downto 0)
);

end entity testing;
```

```
architecture Dataflow of testing is

signal temp, temp_action :integer;

begin

temp<= to_integer(unsigned(temperature));
temp_action<=0 when temp<25 else
             1 when (temp>=25 and temp<30) else
             2 when (temp>=30 and temp<45) else
             3;


with temp_action select
action<="00" when 0,
         "01" when 1,
         "10" when 2,
         "11" when 3,
         "XX" when others;


end architecture Dataflow ;
```

Λογική Συνθήκη

Ανάθεση με επιλογή
(διακριτές τιμές συγκεκριμένου σήματος)

Μία εντολή: Ανάθεση τιμής σε ένα σήμα

Πρέπει να ελέγχω ΟΛΕΣ τις τιμές

VHDL – Conditional assignment (Behavioral)

If

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;USE ieee.numeric_std.ALL;

entity testing is
port (temperature:  in std_logic_vector(5 downto 0);
        action          :   out std_logic_vector(1 downto 0)
);
end entity testing;

architecture Behavioral of testing is
signal temp :integer;
```

```
begin
temp<= to_integer(unsigned(temperature));

action_temp: process(temp) is
begin
if (temp<25) then
   action<="00";
elsif (temp<30) then
   action<="01";
elsif (temp<45) then
   action<="10";
else
   action<="11";
end if;
end process action_temp;

end architecture Behavioral;
```

Μπορεί και πολλαπλές εντολές ανά περίπτωση (if/elsif/else)

if, **elsif**, else
Μόνο μέσα
σε process

Πρέπει να
ελέγχω ΟΛΕΣ
τις τιμές

VHDL – Conditional assignment (Behavioral)

Case

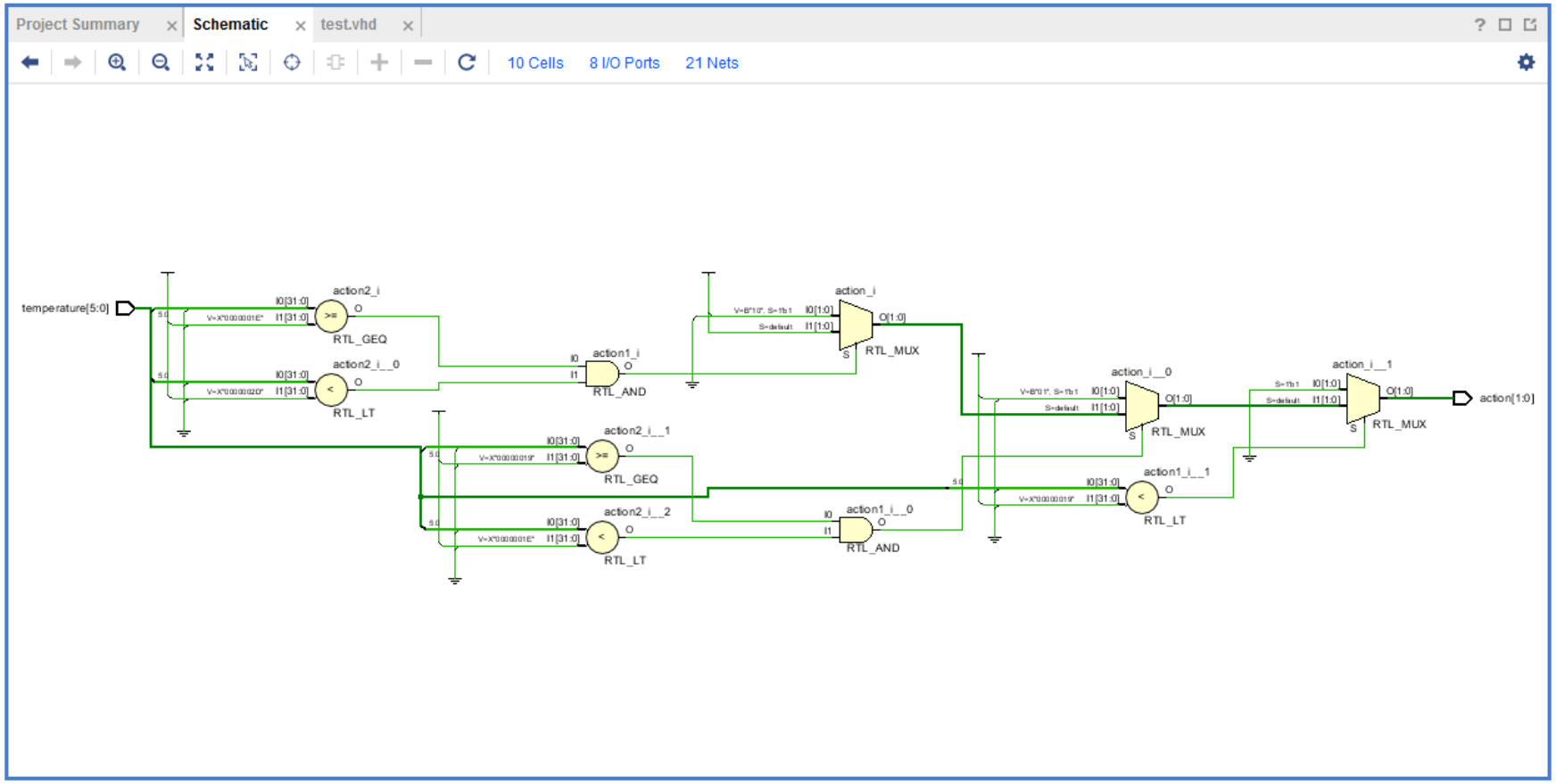
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;USE ieee.numeric_std.ALL;

entity testing is
port (temperature:  in std_logic_vector(5 downto 0);
        action          :  out std_logic_vector(1 downto 0)
);
end entity testing;

architecture Behavioral of testing is
signal temp :integer;

begin
temp<= to_integer(unsigned(temperature));

action_temp: process(temp) is
begin
if (temp<25) then       temp_action<=0;
elsif (temp<30) then    temp_action<=1;
elsif (temp<45) then    temp_action<=2;
else                    temp_action<=3;
end if;

case temp_action is
            when 0 => action<="00";
            when 1 => action<="01";
            when 2 => action<="10";
            when 3 => action<="11";
            when others => action<="XX";
end case;
end process action_temp;
end architecture Behavioral;
```

Μπορεί και πολλαπλές εντολές ανά περίπτωση (when)

case
Διακριτές τιμές

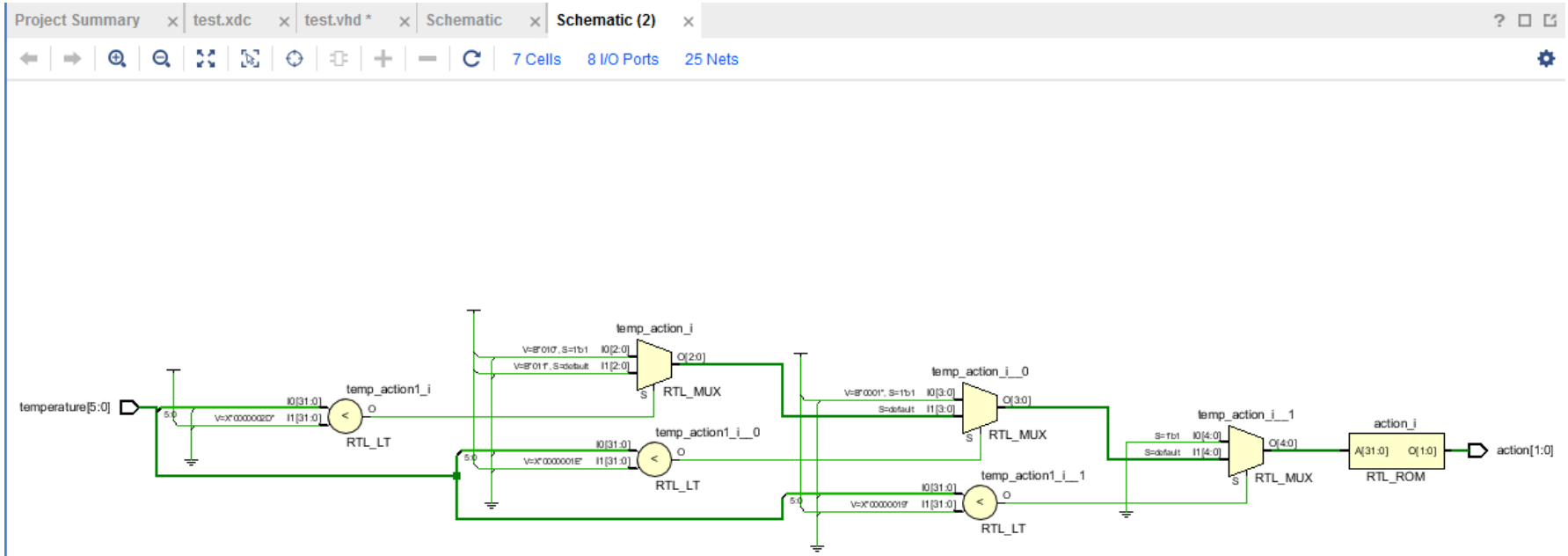
VHDL – Selected assignment

RTL Analysis



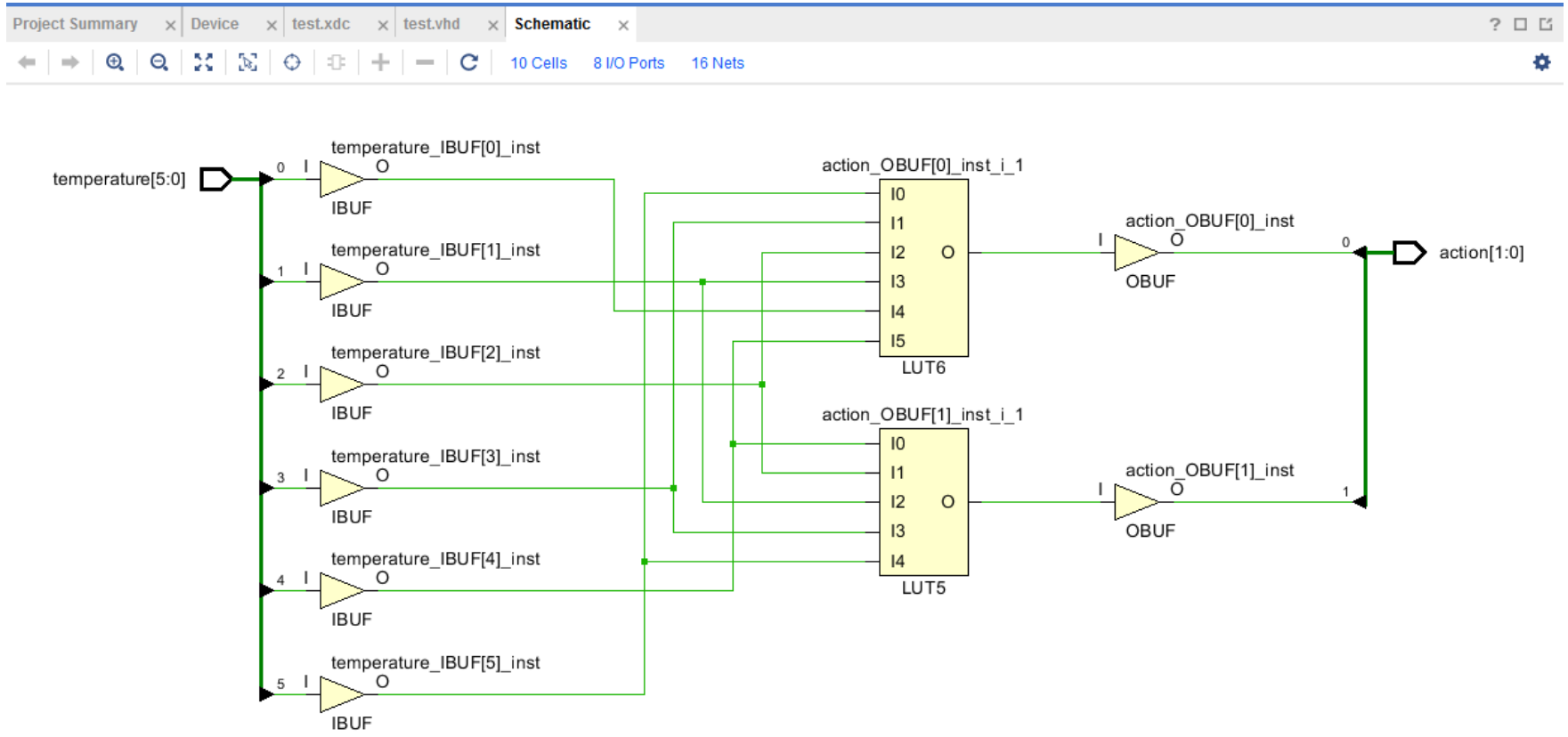
VHDL – Conditional assignment

RTL Analysis



VHDL – Selected & Conditional assignment

Synthesis



VHDL – Selected & Conditional assignment

Synthesis

1.
signal <= value when condition else …

2.
with signal_1 select
signal_2<=value when (discrete signal_1 value),
…

3.
If condition then action;elsif … else end if

4.
case signal is
when value => assignment (discrete signal value),
…

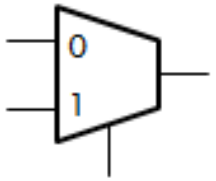
Και στις 2 περιπτώσεις είμαστε απευθείας στο σώμα της αρχιτεκτονικής και η εντολή αφορά απόδοσης τιμής σε ένα συγκεκριμένο σήμα

Και στις 2 περιπτώσεις είμαστε στο σώμα process (ή procedure) και αφορά την εκτέλεση μίας η περισσότερων εντολών ανάλογα τη συνθήκη, και μπορεί να αφορά πάνω από ένα σήματα.

VHDL – Multiplexer

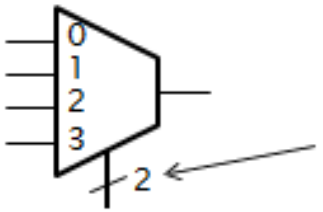
- Επιλέγει μεταξύ εισόδων δεδομένων με βάση μια είσοδο επιλογής

2-σε-1 πολυπλέκτης



sel	z
0	a_0
1	a_1

4-σε-1 πολυπλέκτης



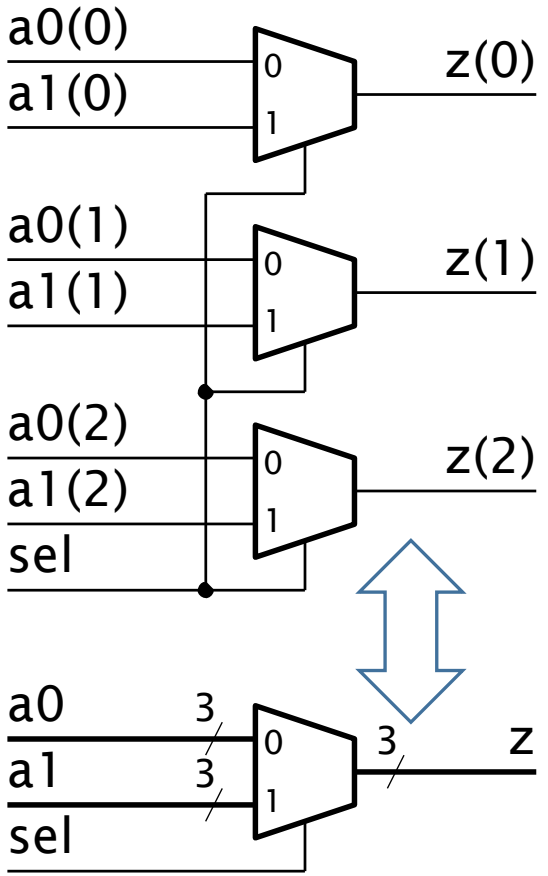
δύο bit επιλογής

sel	z
00	a_0
01	a_1
10	a_2
11	a_3

- Ένας N-σε-1 πολυπλέκτης απαιτεί $\lceil \log_2 N \rceil$ bit επιλογής

VHDL – Multiplexer

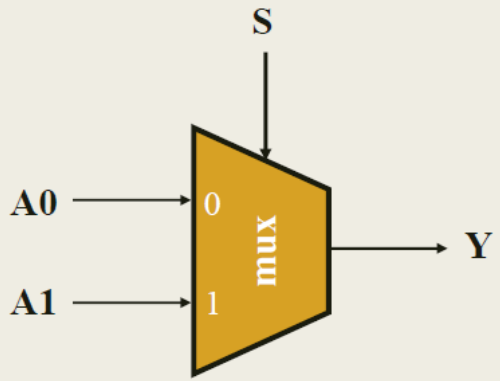
- Για να επιλέξουμε μεταξύ *N* κωδικών λέξεων των *m* bit
  - Συνδέουμε παράλληλα *m* πολυπλέκτες των *N* εισόδων



VHDL – Multiplexer

Entity (2to1)

```
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;
```



VHDL – Multiplexer

Architecture (2to1 - behavioral)

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

y<=A0 when s='0' else A1;

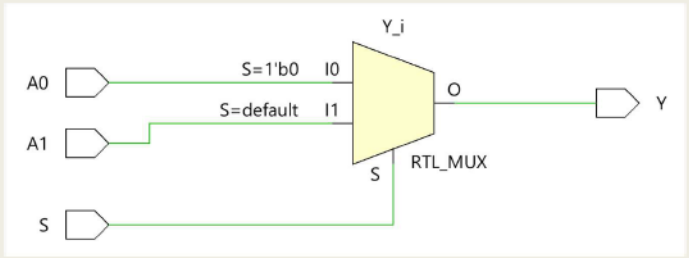
VHDL – Multiplexer

Architecture (2to1 - behavioral)

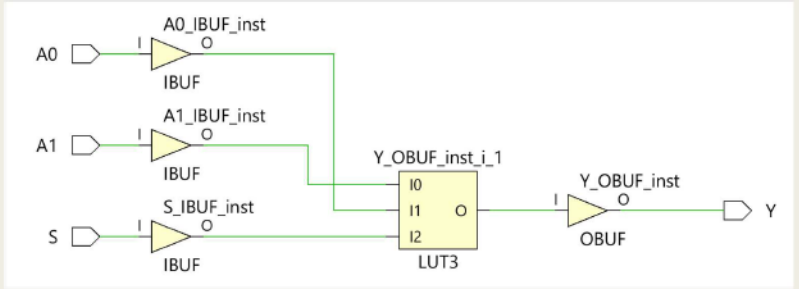
Η αρχιτεκτονική του πολυπλέκτη 2 σε 1 στη VHDL

Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA

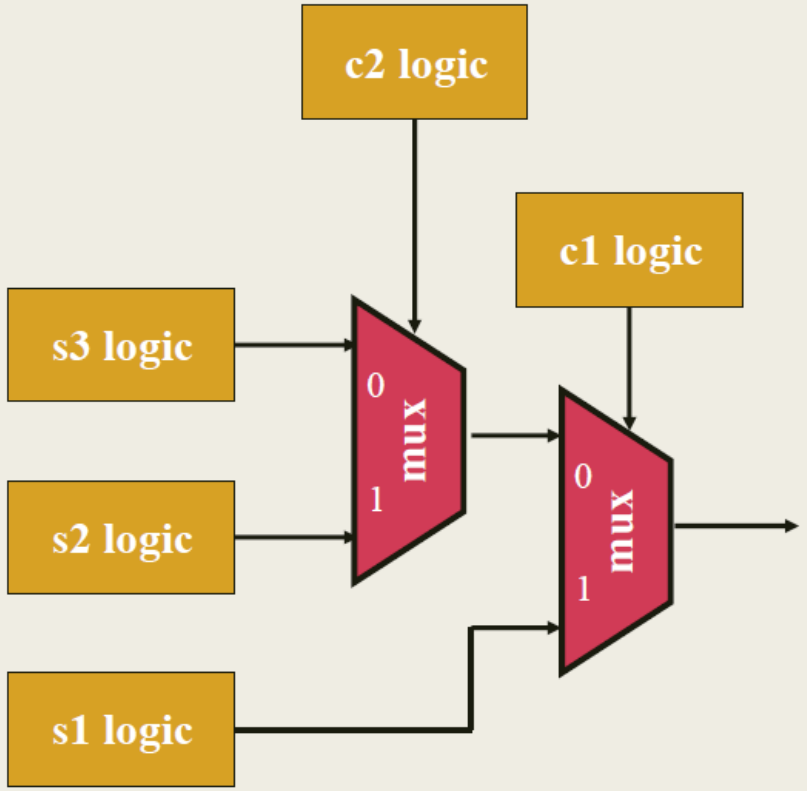


VHDL – Multiplexer

Υλοποίηση If

- Υλοποίηση εντολής IF με τη χρήση πολυπλεκτών 2 σε 1

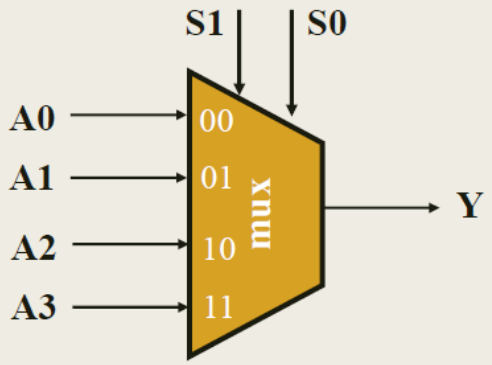
```
if c1 then
    s1;
elsif c2 then
    s2;
else
    s3;
end if;
```



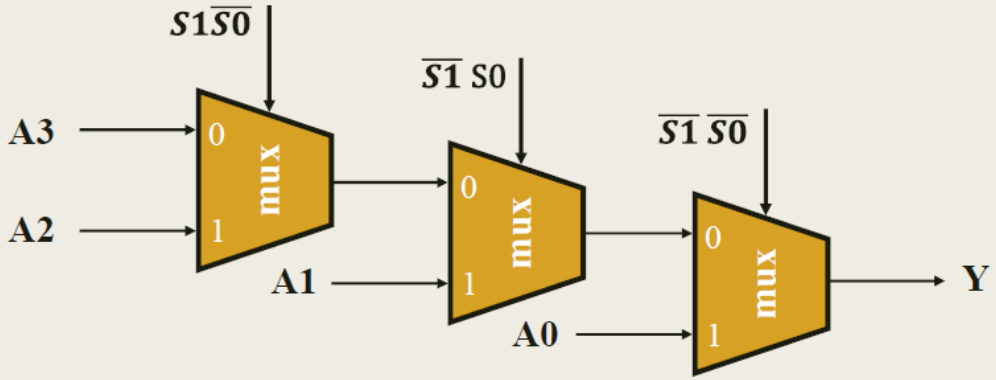
VHDL – Multiplexer

Entity (4to1)

```
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: Υλοποίηση με πολυπλέκτες 2 σε 1 σε δομή αλυσίδας



VHDL – Multiplexer

Architecture (4to1)

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;
```

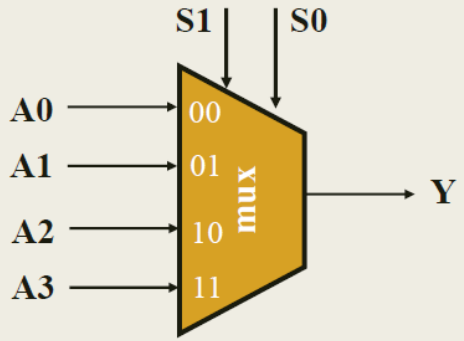
Στη λίστα ευαισθησίας συμπεριλαμβάνονται όλες οι είσοδοι του συνδυαστικού κυκλώματος

VHDL – Multiplexer

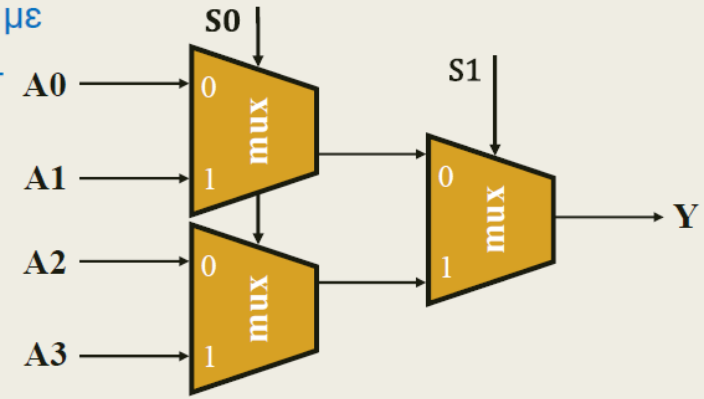
Entity (4to1)

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: Υλοποίηση με πολυπλέκτες 2 σε 1 σε δομή δένδρου



VHDL – Multiplexer

Architecture (4to1)

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

VHDL - Περίληψη

- Τύπος Signed
- Άλλοι τύπο σημάτων
- Τελεστές
- Ολισθήσεις
- Selected Assignment
- Conditional Statements
- Διαβάζετε τις παραγράφους 2.4, 3.2 (θεωρία και VHDL) από Ashenden και 2.7, 2.8, 4.2.4, 4.2.6, 4.3, 4.5.1, 4.5.2, 4.5.3 (ΟΧΙ το κομμάτι της VERILOG) από το βιβλίο των Harris.

dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

Εργαστήριο Λογικής Σχεδίασης

VHDL - Εισαγωγή

Βασιλόπουλος Διονύσης

ΕΤΕΠ Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

VHDL – Επαναλήψεις

Εντολή FOR

```
for variable in range loop
        sequential statements;
end for loop;
```

θα το χρησιμοποιείτε μόνο για simulation

Μόνο μέσα σε Process

```
for i in 0  to 2 loop
   a_tb<=std_logic_vector(to_signed(i,a_tb'length));
   for j in 0 to 2 loop
      b_tb<=std_logic_vector(to_signed(j,a_tb'length));wait for 10ns;
   end loop j;
end loop i;
```

VHDL – Procedure

```
Ctr_tb<='0';
for i in 0  to 2 loop
a_tb<=std_logic_vector(to_signed(i,a_tb'length));
   for j in 0 to 2 loop
      b_tb<=std_logic_vector(to_signed(j,a_tb'length));wait for 10ns;
   end loop j;--end loop i;

Ctr_tb<='1';
for i in 0  to 2 loop
   a_tb<=std_logic_vector(to_signed(i,a_tb'length));
   for j in 0 to 2 loop
      b_tb<=std_logic_vector(to_signed(j,a_tb'length));wait for 10ns;
   end loop j;
end loop i;
```

```
procedure sim_test is
begin

for i in 0  to 2 loop

a_tb<=std_logic_vector(to_signed(i,a_tb'length));
   for j in 0 to 2 loop
      b_tb<=std_logic_vector(to_signed(j,a_tb'length));
      wait for 10ns;
end loop j;

end loop i;

end procedure;

Ctr_tb<='0';sim_test;
Ctr_tb<='1';sim_test;
```

VHDL – Procedure

procedure procedure_name(input and output parameters) **is**

declarations

begin

sequential statement1;
sequential statement2;
.....

end procedure;

VHDL – Procedure

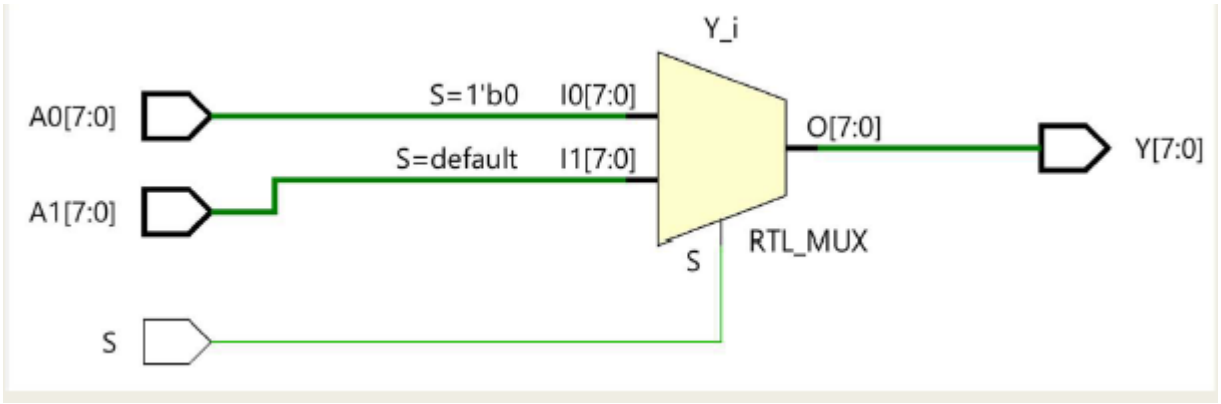
```
procedure sim_test (min, max: in integer; step: in integer) is
variable a2: integer;
begin
for i in min  to max loop
        a<=std_logic_vector(to_signed(i*step,a'length));
        for j in min to max loop
                b<=std_logic_vector(to_signed(j*step,a'length));wait for 10 ns;
        end loop j;
end loop i;

end procedure;

Ctr<='0';sim_test(0,2,5);
Ctr<='1';sim_test(0,2,5);
```

Όταν δεν αναφέρεται τύπος εννοείται variable. Σε αυτή την περίπτωση τα ορίσματα πρέπει να είναι σταθερές ή variables

Όταν ο τύπος στα ορίσματα είναι signal τότε και στα ορίσματα στην κλήση της procedure θα πρέπει να υπάρχουν signal

VHDL – Generic

```
entity MUX2in1_n is
generic (WIDTH : positive := 8); -- προεπιλεγμένη τιμή
port (      S: in STD_LOGIC;
            A0: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
            A1: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
            Y: out STD_LOGIC_VECTOR (WIDTH-1 downto 0));
end MUX2in1_n;
architecture BEHAVIORAL of MUX2in1_n is
begin
process (A0, A1, S)
begin
            if (S = '0') then
                        Y <= A0;
            else
                        Y <= A1;
            end if;
end process;
end BEHAVIORAL;
```



VHDL – Generic

Παραμετροποίηση του μεγέθους μίας αρτηρίας σε μία οντότητα με τη δήλωση της εντολής generic που ορίζει την σταθερά WIDTH

- Η δήλωση της εντολής generic γίνεται πριν από τη δήλωση των ports στην αρχή της οντότητας
- Η σταθερά WIDTH είναι θετικός ακέραιος (positive) και μπορεί να έχει προεπιλεγμένη τιμή
  - generic (WIDTH: positive := 8);
- Η σταθερά WIDTH χρησιμοποιείται κατά τη δήλωση των ports
  - STD_LOGIC_VECTOR (WIDTH-1 downto 0);
- Η τιμή της σταθεράς WIDTH μπορεί να παρακάμψει την προεπιλεγμένη τιμή με τη φράση generic map (συνδυάζεται με το port map)
  - generic map (WIDTH => 8)

VHDL – Structural architecture

Αθροιστής

Πρόσθεση

4 bit

```
  1001
 +1101
 -------
  0110   Carry=1
```

8 bit

1=Carry in στη 2η ALU

```
 0010 1001
+0101 1101
---------------
1000 0110   Carry_out=1 από 1η ALU
```

Carry_in=‘0’ 1η ALU

Carry_out=0 2η ALU

Διπλασιασμός

4 bit

```
 1001 & ‘0’
 --------
  0010   Carry=1
```

8 bit

1=Carry in στη 2η ALU

```
0010 1001 & ‘0’
---------------
0101 0010     Carry_out=1 από 1η ALU
```

Carry_in=0 1η ALU

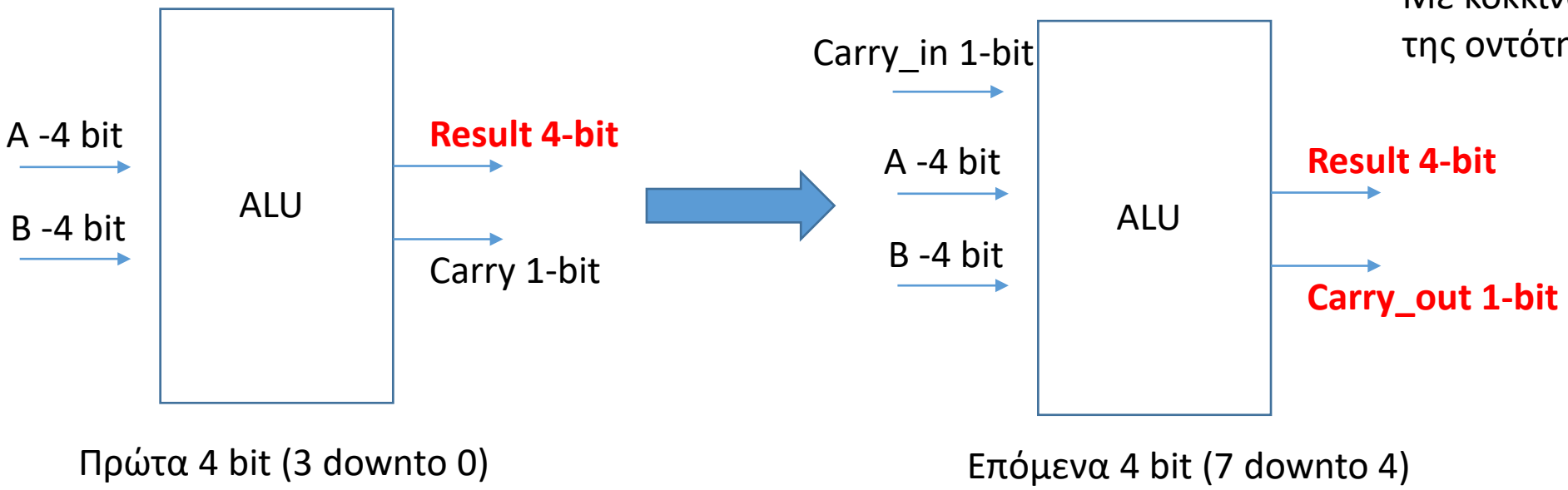
Carry_out=0 2η ALU

VHDL – Structural architecture

Αθροιστής

Για να μπορέσουμε να συνδυάσουμε 2 alu 4-bit θα πρέπει να μπορέσουμε να χειριστούμε το carry

Με κόκκινο οι έξοδοι της οντότητας

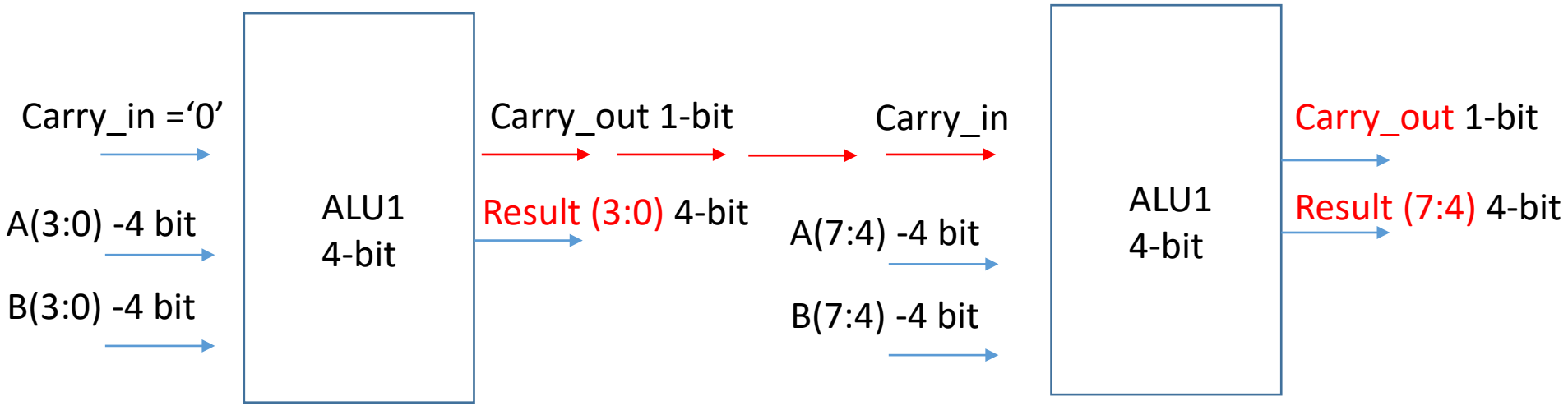


Πρώτα 4 bit (3 downto 0)

Επόμενα 4 bit (7 downto 4)

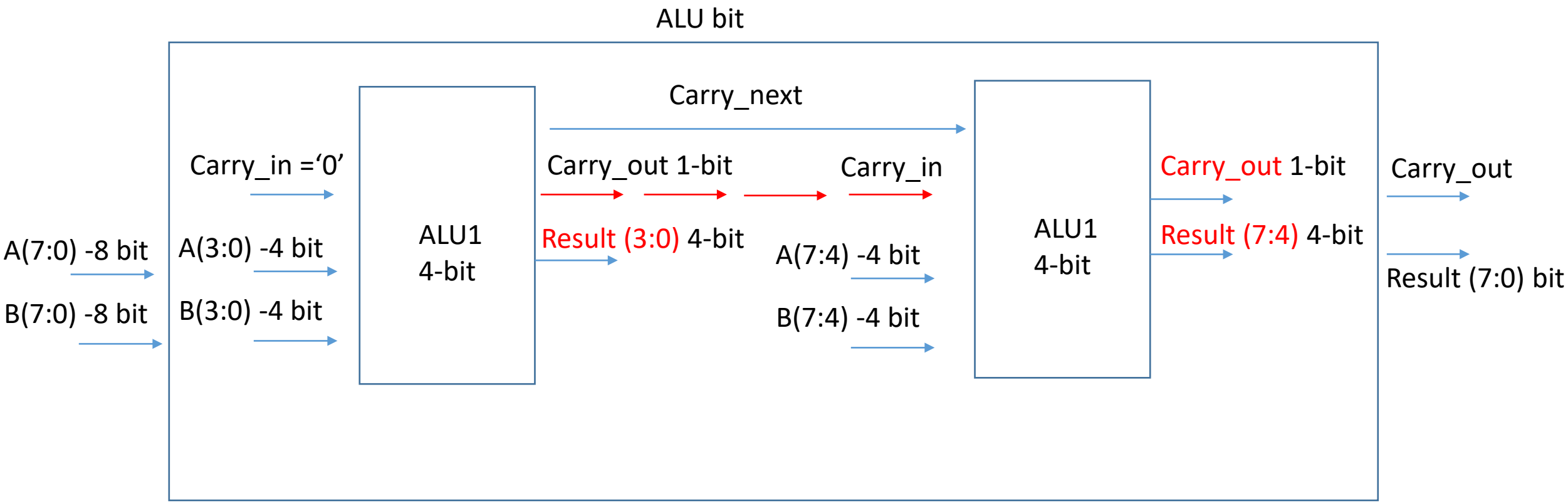
VHDL – Structural architecture

Αθροιστής



VHDL – Structural architecture

Αθροιστής



VHDL – Κωδικοποίηση

- Πώς αναπαριστούμε πληροφορία με περισσότερες από δύο πιθανές τιμές;
  - Πολλαπλά δυαδικά σήματα (πολλαπλά bit)
- (a_1, a_0): (0, 0), (0, 1), (1, 0), (1, 1)
  - Αυτός είναι ένας *δυαδικός κώδικας*
  - Κάθε ζεύγος τιμών είναι μια *κωδική λέξη*

VHDL – Κωδικοποίηση

Code Length

- Ένας κώδικας των n bit έχει 2^n κωδικές λέξεις
- Για να αναπαραστήσουμε N πιθανές τιμές
  - χρειαζόμαστε τουλάχιστον $\lceil \log_2 N \rceil$ bit για τις λέξεις
  - περισσότερα bit μπορούν να είναι χρήσιμα σε κάποιες περιπτώσεις
- Παράδειγμα: κώδικας εκτυπωτή ψεκασμού
  - Black, cyan, magenta, yellow, light cyan, light magenta
  - έξι τιμές, $\lceil \log_2 6 \rceil = 3$
  - Black : (0, 0, 1), cyan : (0, 1, 0), magenta : (0, 1, 1),
    yellow : (1, 0, 0), light cyan : (1, 0, 1), light magenta : (1, 1, 0)

VHDL – Κωδικοποίηση

One Hot

- Κάθε κωδική λέξη έχει ακριβώς ένα bit με την τιμή ‘1'
- Φωτεινός σηματοδότης:
  - κόκκινο: (1,0,0), πορτοκαλί: (0,1,0), πράσινο: (0,0,1)
  - τρεις αγωγοί σημάτων: κόκκινο, πορτοκαλί, πράσινο
- Κάθε bit ενός κώδικα one-hot αντιστοιχεί σε μια κωδικοποιημένη τιμή
  - Μήκος κώδικα ίσο με πλήθος των προς κωδικοποίηση τιμών.
  - Όχι ελάχιστο μήκος

VHDL – Κωδικοποίηση

Παράδειγμα (1/2)

- Ελεγκτής φωτεινού σηματοδότη με κώδικα 1-hot
  - enable = 1: lights_out = lights_in
  - enable = 0: lights_out = (0, 0, 0)

```
library ieee; use       ieee.std_logic_1164.all;
entity   light_controller is
   port   ( lights_in  :      in   std_logic_vector(1 to 3);
            enable     :      in   std_logic;
            lights_out :      out  std_logic_vector(1 to 3) );
end entity light_controller;
```

VHDL – Κωδικοποίηση

Παράδειγμα (2/2)

```
architecture and_enable of light_controller is
begin
   lights_out(1) <= lights_in(1) and enable;
   lights_out(2) <= lights_in(2) and enable;
   lights_out(3) <= lights_in(3) and enable;
end architecture and_enable;
```

```
architecture conditional_enable of light_controller is
begin
  lights_out <= lights_in when enable = '1' else
                "000";
end architecture conditional_enable;
```

or just **when enable**

VHDL – Κωδικοποίηση

Παράδειγμα (1/3) – 1η Υλοποίηση

- Παράδειγμα: Αντικλεπτικό σύστημα συναγερμού - κωδικοποιεί ποια ζώνη έχει ενεργοποιηθεί
  - 8 bit εισόδου: ένας αισθητήρας για κάθε ζώνη
  - 3 bit εξόδου για την κωδικοποίηση των ζωνών

```
library ieee;
use ieee.std_logic_1164.all;
entity alarm is
  port ( zone           : in std_logic_vector (1 to 8);
         intruder_zone  : out std_logic_vector(2 downto 0);
         valid   : out std_logic );
end entity alarm;
architecture eqn of alarm is
begin
  intruder_zone(2) <= zone(5) or zone(6) or zone(7) or zone(8);
  intruder_zone(1) <= zone(3) or zone(4) or zone(7) or zone(8);
  intruder_zone(0) <= zone(2) or zone(4) or zone(6) or zone(8);
  valid <= zone(1) or zone(2) or zone(3) or zone(4) or zone(5)
           or zone(6) or zone(7) or zone(8);
end architecture eqn;
```

Ζώνη	Κωδική λέξη
Zone 1	0, 0, 0
Zone 2	0, 0, 1
Zone 3	0, 1, 0
Zone 4	0, 1, 1
Zone 5	1, 0, 0
Zone 6	1, 0, 1
Zone 7	1, 1, 0
Zone 8	1, 1, 1

VHDL – Κωδικοποίηση

Παράδειγμα (2/3) – 2η Υλοποίηση

- Εάν περισσότερες από μία είσοδοι μπορεί να είναι 1
  - Κωδικοποιούμε την είσοδο που είναι 1 με την υψηλότερη προτεραιότητα

zone								intruder_zone			valid
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(2)	(1)	(0)	
1	–	–	–	–	–	–	–	0	0	0	1
0	1	–	–	–	–	–	–	0	0	1	1
0	0	1	–	–	–	–	–	0	1	0	1
0	0	0	1	–	–	–	–	0	1	1	1
0	0	0	0	1	–	–	–	1	0	0	1
0	0	0	0	0	1	–	–	1	0	1	1
0	0	0	0	0	0	1	–	1	1	0	1
0	0	0	0	0	0	0	1	1	1	1	1
0	0	0	0	0	0	0	0	–	–	–	0

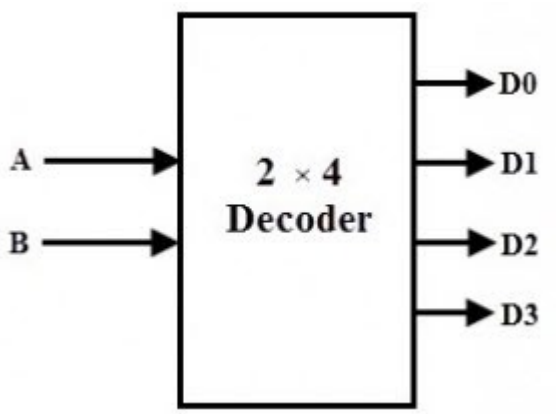
VHDL – Κωδικοποίηση

Παράδειγμα (3/3)

Conditional signal assignment

```
architecture priority_1 of alarm is
begin
  intruder_zone <= "000" when zone(1) = '1' else
                   "001" when zone(2) = '1' else
                   "010" when zone(3) = '1' else
                   "011" when zone(4) = '1' else
                   "100" when zone(5) = '1' else
                   "101" when zone(6) = '1' else
                   "110" when zone(7) = '1' else
                   "111" when zone(8) = '1' else
                   "000";
  valid <= zone(1) or zone(2) or zone(3) or zone(4)
           or zone(5) or zone(6) or zone(7) or zone(8);
end architecture priority_1;
```

VHDL – Αποκωδικοποίηση



- Ο αποκωδικοποιητής εξάγει σήματα ελέγχου από ένα δυαδικά κωδικοποιημένο σήμα
  - Ένα σήμα ανά κωδική λέξη
  - Το σήμα ελέγχου είναι 1 όταν η είσοδος έχει την αντίστοιχη κωδική λέξη, διαφορετικά 0

VHDL – Αποκωδικοποίηση

```
library ieee; use ieee.std_logic_1164.all;
entity decoder4 is
port (a: in std_logic_vector(1 downto 0);
      y: out std_logic_vector(3 downto 0));
end entity decoder4 ;
architecture sel_arch of decoder4 is
begin
   with a select y <=
    "0001" when "00",
    "0010" when "01",
    "0100" when "10",
    "1000" when "11",
    "0000" when others;
end sel_arch ;
```

For simulation, you don't want to enumerate all 77 (81-4) meta-values of std_logic..
For synthesis, it is ignored...

Selected signal assignment

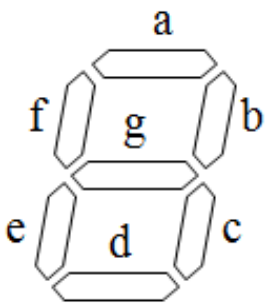
```
library ieee; use ieee.std_logic_1164.all;
entity decoder4 is
port (a: in std_logic_vector(1 downto 0);
      y: out std_logic_vector(3 downto 0));
end entity decoder4 ;
architecture case_arch of decoder4 is
begin
    process (a)
    begin
      case a is
         when "00" =>   y<= "0001";
         when "01" =>   y<= "0010";
         when "10" =>   y<= "0100";
         when "11" =>   y<= "1000";
         when others => y<= "0000";
      end case;
    end process;
end case_arch;
```

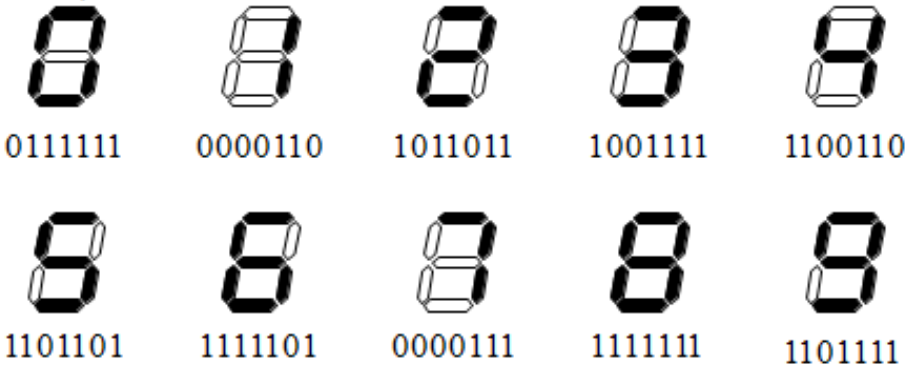
Combinational Process with Case Statement

VHDL – Αποκωδικοποίηση

Παράδειγμα (1/2)

- Αποκωδικοποιεί τον κώδικα BCD (binary coded decimal) για να οδηγήσει μια οθόνη (LED ή LCD) 7 τμημάτων
  - Τμήματα: (g, f, e, d, c, b, a)





- Κώδικας BCD: Δυαδικά κωδικοποιημένοι δεκαδικοί
  - Κώδικας 4 bit για τα δεκαδικά ψηφία

0: 0000	1: 0001	2: 0010	3: 0011	4: 0100
5: 0101	6: 0110	7: 0111	8: 1000	9: 1001

VHDL – Αποκωδικοποίηση

Παράδειγμα (2/2)

```
library ieee; use ieee.std_logic_1164.all;
entity seven_seg_decoder is
  port ( bcd   : in std_logic_vector (3 downto 0);
         blank : in std_logic;
         seg   : out std_logic_vector (7 downto 1) );
end entity seven_seg_decoder;
```

```
architecture behavior of seven_seg_decoder is
  signal seg_tmp : std_logic_vector (7 downto 1);
begin
  with bcd select
    seg_tmp <= "0111111" when "0000", -- 0
               "0000110" when "0001", -- 1
               "1011011" when "0010", -- 2
               "1001111" when "0011", -- 3
               ...
               "1101111" when "1001", -- 9
               "1000000" when others; -- "-"
seg <= "0000000" when blank = '1' else seg_tmp;
end architecture behavior;
```

Selected signal assignment

dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

Εργαστήριο Λογικής Σχεδίασης

3ο Εργαστηριακό Μάθημα

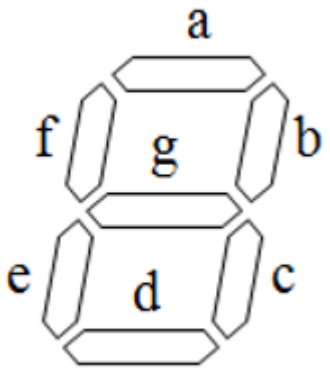
Βασιλόπουλος Διονύσης

ΕΤΕΠ Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

3η Εργαστηριακή Άσκηση

VHDL – 7 segment led



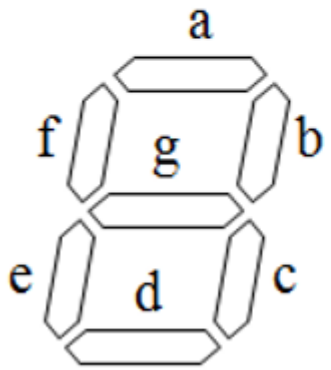
Αναπαράσταση με 7-bit
MSB->g - LSB->a

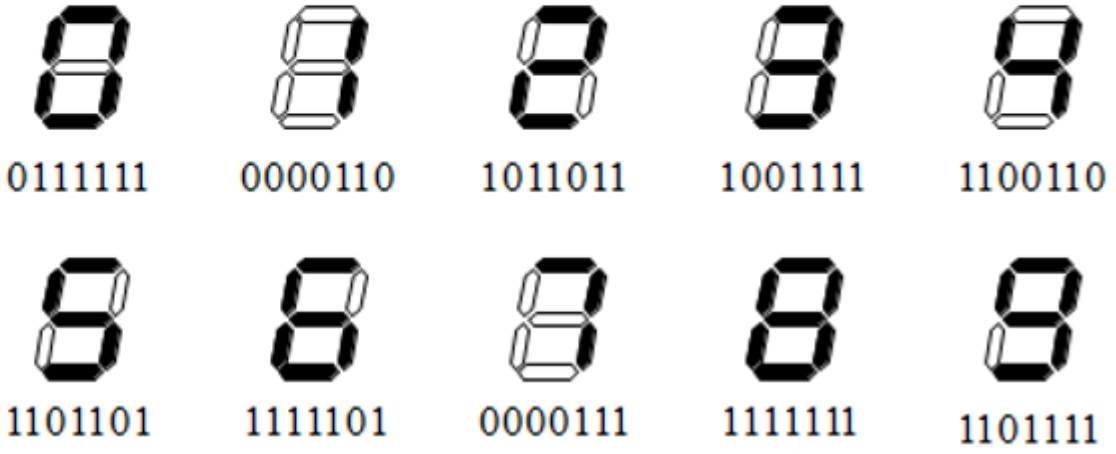
g-f-e-d-c-b-a

3η Εργαστηριακή Άσκηση

VHDL – 7 segment led

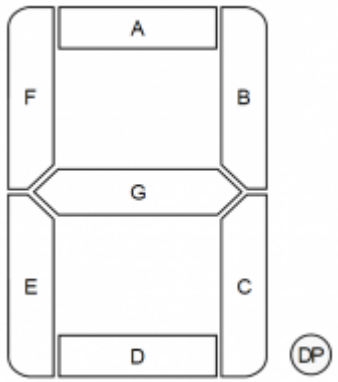
- Αποκωδικοποιεί τον κώδικα BCD (binary coded decimal) για να οδηγήσει μια οθόνη (LED ή LCD) 7 τμημάτων
  - Τμήματα: (g, f, e, d, c, b, a)





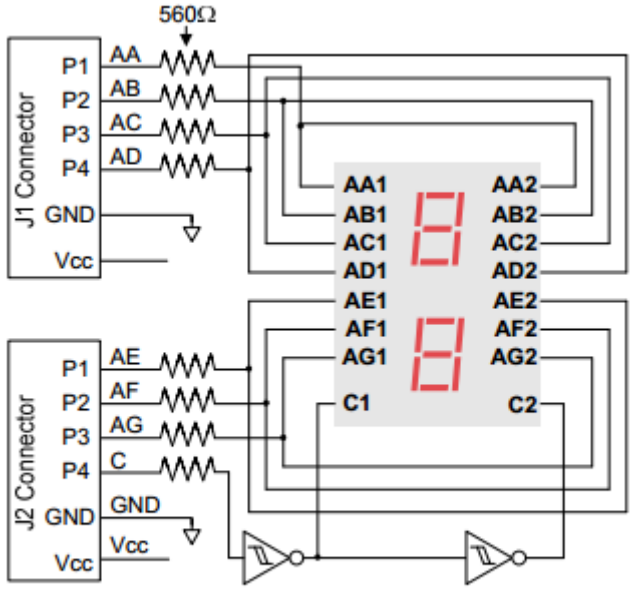
3η Εργαστηριακή Άσκηση

7 segment led - Pmod



Pinout Description Table

Header J1			Header J2		
Pin	Signal	Description	Pin	Signal	Description
1	AA	Segment A	1	AE	Segment E
2	AB	Segment B	2	AF	Segment F
3	AC	Segment C	3	AG	Segment G
4	AD	Segment D	4	C	Digit Selection pin
5	GND	Power Supply Ground	5	GND	Power Supply Ground
6	VCC	Positive Power Supply	6	VCC	Positive Power Supply



Seven-Segment Display Connection Diagram

ΜΟΝΟ ΤΟ ΈΝΑ ΑΠΌ ΤΑ 2 LED ΜΠΟΡΕΙ ΝΑ ΕΊΝΑΙ ΑΝΑΜΕΝΟ ΣΕ ΚΆΘΕ ΧΡΟΝΙΚΗ ΣΤΙΓΜΗ

Pmod: Peripheral Module interface

Εικόνες από το https://reference.digilentinc.com/reference/pmod/pmodssd/reference-manual

3η Εργαστηριακή Άσκηση

7 segment led - Pmod

Table 16 - Pmod Connections

Pmod	Signal Name	Zynq pin	Pmod	Signal Name	Zynq pin
JA1	JA1	Y11	JB1	JB1	W12
	JA2	AA11		JB2	W11
	JA3	Y10		JB3	V10
	JA4	AA9		JB4	W8
	JA7	AB11		JB7	V12
	JA8	AB10		JB8	W10
	JA9	AB9		JB9	V9
	JA10	AA8		JB10	V8

Θα βρείτε σε ποια Signal της FPGA αντιστοιχούν τα signal του Pmod.
Κατόπιν θα βρείτε σε ποια pin της FPGA αντιστοιχούν τα signal του Pmod.

3η Εργαστηριακή Άσκηση

ΠΡΟΧΩΡΗΣΤΕ ΣΤΗΝ ΑΣΚΗΣΗ

3η Εργαστηριακή Άσκηση

Αρχιτεκτονική

```
result_temp<=unsigned('0'&a)+ unsigned('0'&b) when ctr='1' else
             unsigned(a&'0') when ctr='0' else
             (others=>'0');
```

Αρχιτεκτονική →

```
result<=std_logic_vector(result_temp);

with result_temp selectseven_segment<="0111111" when "000",
                                      "0000110" when "001",
                                      "1011011" when "010",
                                      "1001111" when "011",
                                      "1100110" when "100",
                                      "1101101" when "101",
                                      "1111101" when "110",
                                      "1000000" when others;

digit_selection_out<=digit_selection_in;
```

3η Εργαστηριακή Άσκηση

Constraints (1/2) – Switches + Leds

Constraints →

```
# ZedBoard Pin Assignments
##############################
# On-board Slide Switches  #
##############################

set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { digit_selection_in }];
set_property -dict { PACKAGE_PIN H19   IOSTANDARD LVCMOS33 } [get_ports { ctr }];
set_property -dict { PACKAGE_PIN F21   IOSTANDARD LVCMOS33 } [get_ports { b[1] }];
set_property -dict { PACKAGE_PIN H22   IOSTANDARD LVCMOS33 } [get_ports { b[0] }];
set_property -dict { PACKAGE_PIN G22   IOSTANDARD LVCMOS33 } [get_ports { a[1] }];
set_property -dict { PACKAGE_PIN F22   IOSTANDARD LVCMOS33 } [get_ports { a[0] }];

##############################
# On-board Leds  #
##############################

set_property -dict { PACKAGE_PIN T22   IOSTANDARD LVCMOS33 } [get_ports { result[0] }];
set_property -dict { PACKAGE_PIN T21   IOSTANDARD LVCMOS33 } [get_ports { result[1] }];
set_property -dict { PACKAGE_PIN U22   IOSTANDARD LVCMOS33 } [get_ports { result[2] }];
```

3η Εργαστηριακή Άσκηση

Constraints (1/2) – Pmod

Constraints →

```
##############################
# PmodSSO                    #
##############################

set_property -dict { PACKAGE_PIN Y11   IOSTANDARD LVCMOS33 } [get_ports { seven_segment[0] }];
set_property -dict { PACKAGE_PIN AA11  IOSTANDARD LVCMOS33 } [get_ports { seven_segment[1] }];
set_property -dict { PACKAGE_PIN Y10   IOSTANDARD LVCMOS33 } [get_ports { seven_segment[2] }];
set_property -dict { PACKAGE_PIN AA9   IOSTANDARD LVCMOS33 } [get_ports { seven_segment[3] }];
set_property -dict { PACKAGE_PIN W12   IOSTANDARD LVCMOS33 } [get_ports { seven_segment[4] }];
set_property -dict { PACKAGE_PIN W11   IOSTANDARD LVCMOS33 } [get_ports { seven_segment[5] }];
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { seven_segment[6] }];

set_property -dict { PACKAGE_PIN W8   IOSTANDARD LVCMOS33 } [get_ports { digit_selection_out }];
```

3η Εργαστηριακή Άσκηση

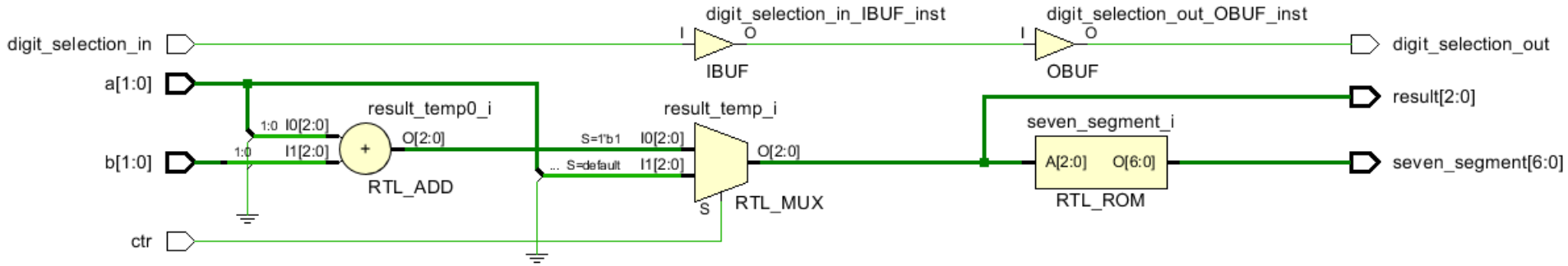
Pmod Manual

Manual Pmod

https://reference.digilentinc.com/reference/pmod/pmodssd/reference-manual

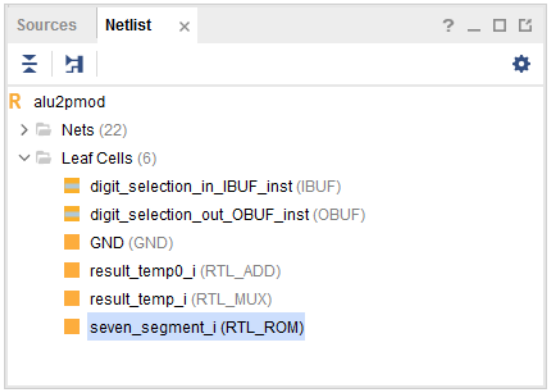
3η Εργαστηριακή Άσκηση

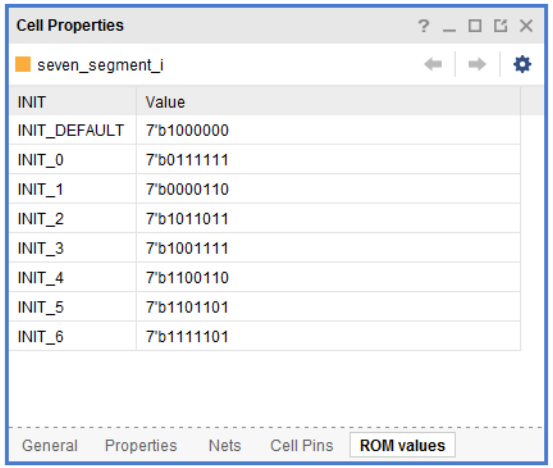
RTL Design

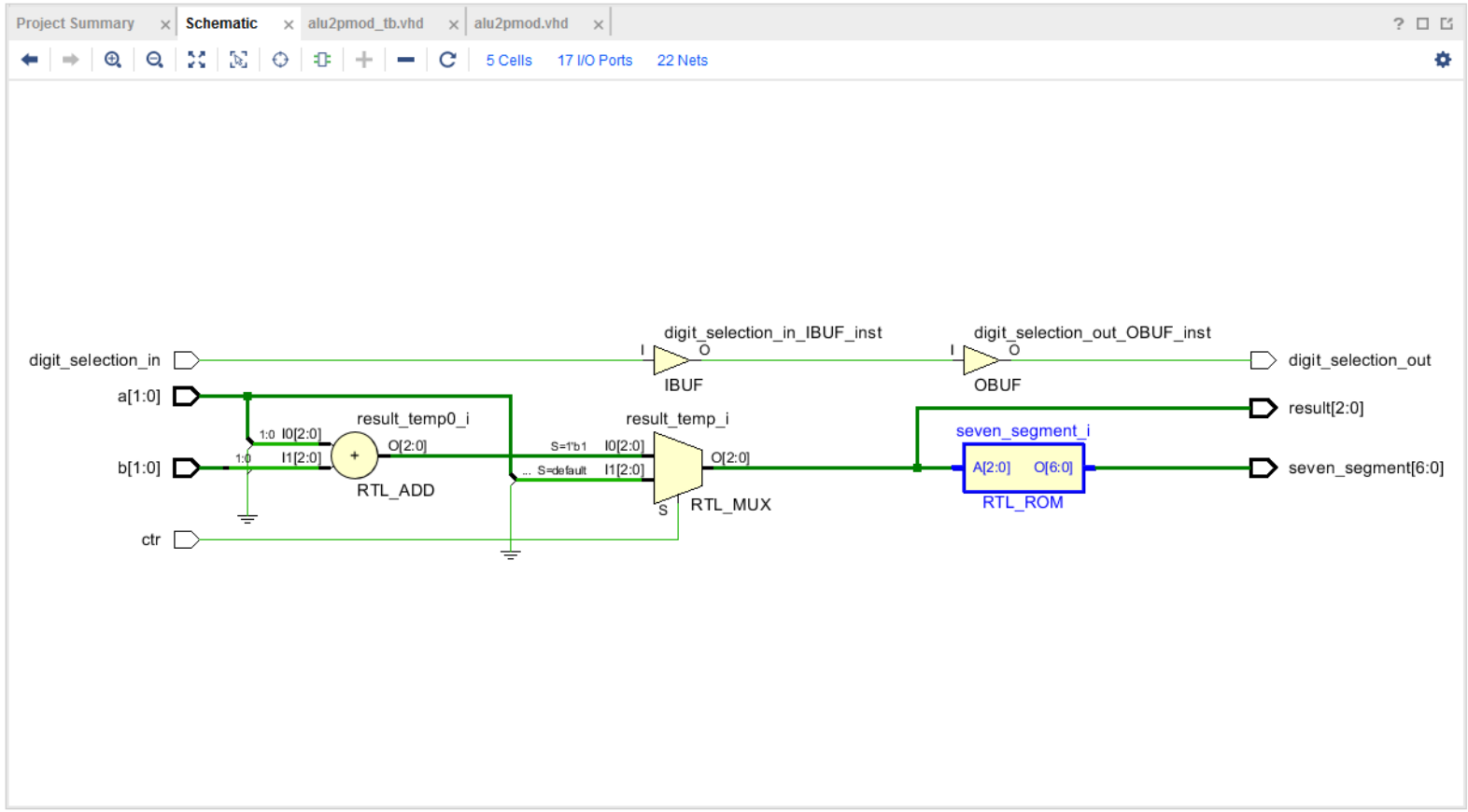


3η Εργαστηριακή Άσκηση

RTL Design – ROM Module/ROM Values

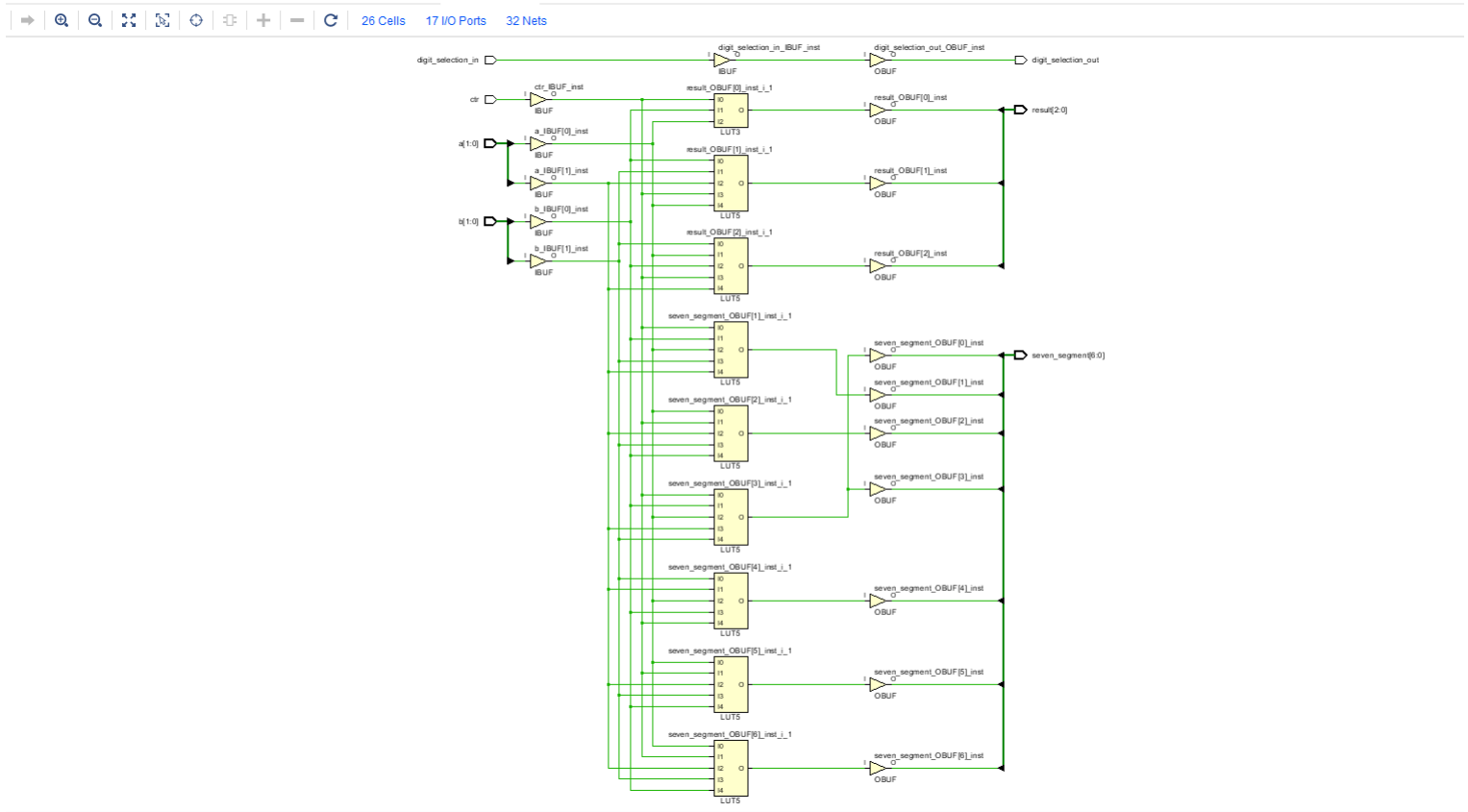






3η Εργαστηριακή Άσκηση

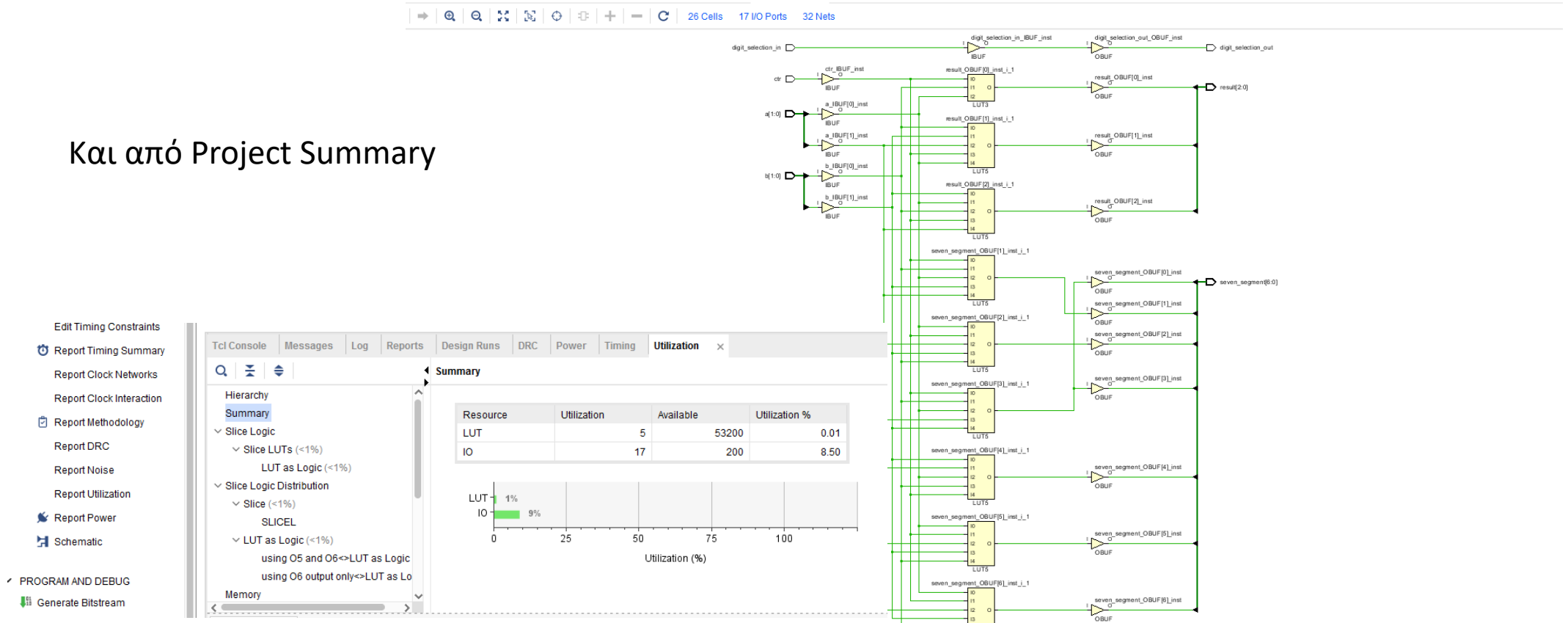
Synthesis/Implementation – Schematic



3η Εργαστηριακή Άσκηση

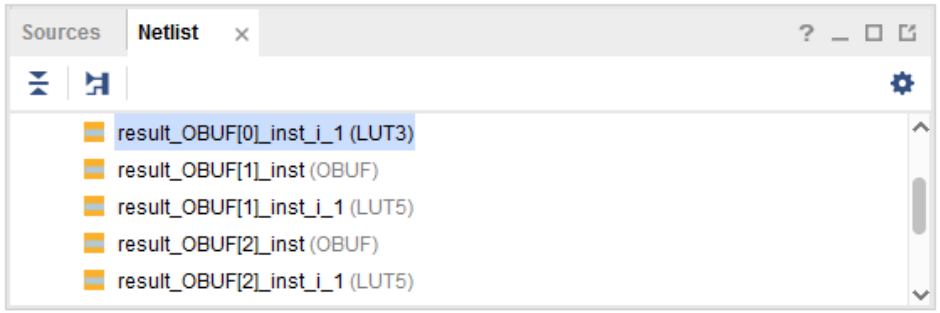
Implementation – Report Utilization (1/3)

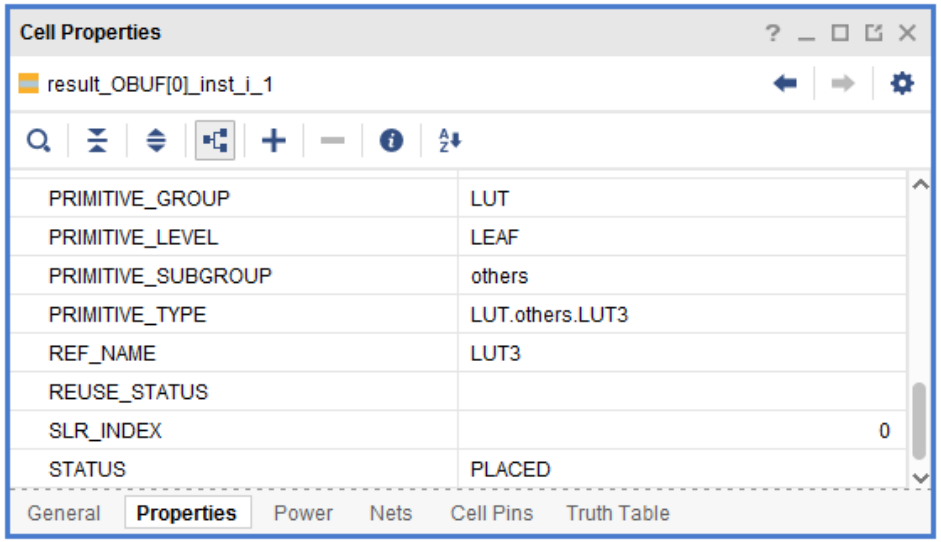
Και από Project Summary

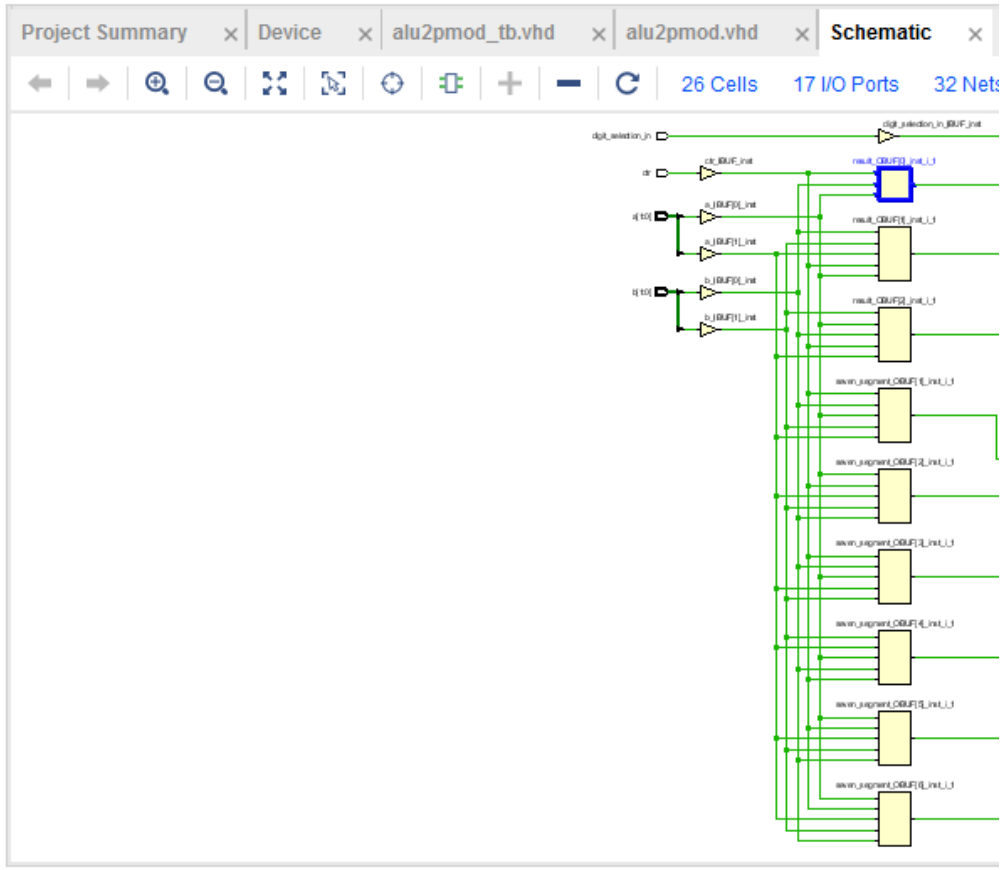


3η Εργαστηριακή Άσκηση

Implementation – Report Utilization (2/3)

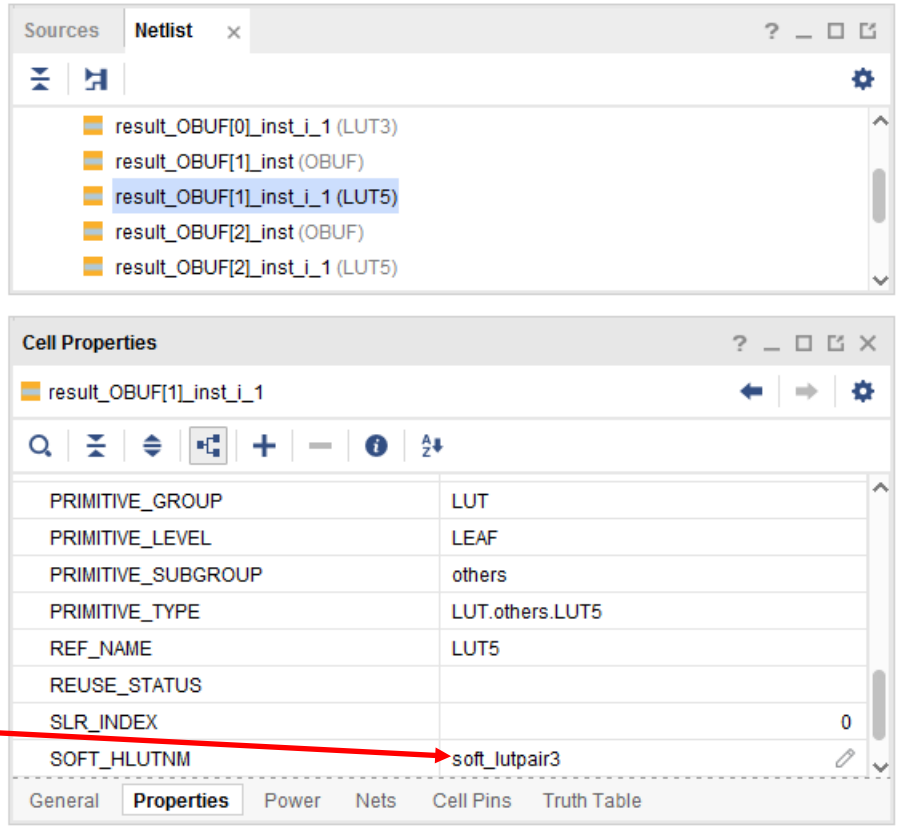


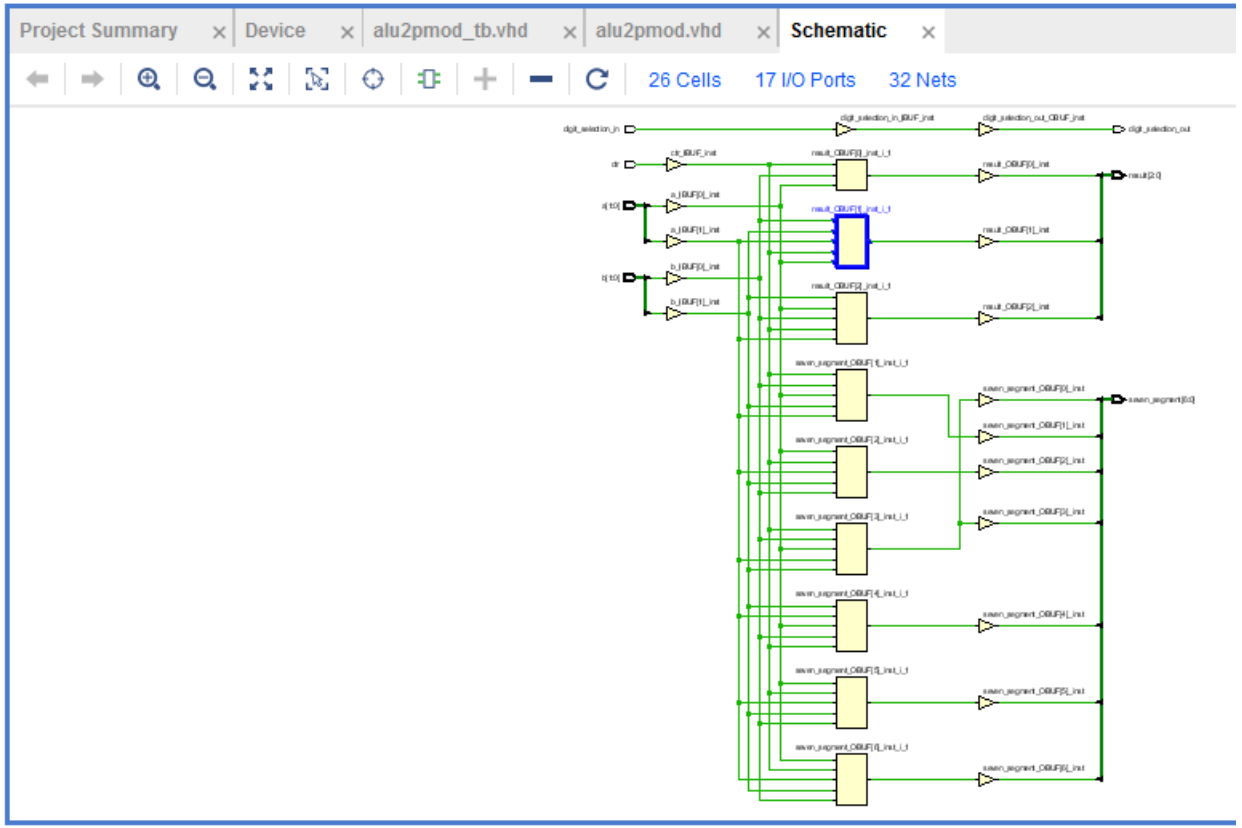




3η Εργαστηριακή Άσκηση

Implementation – Report Utilization (3/3)

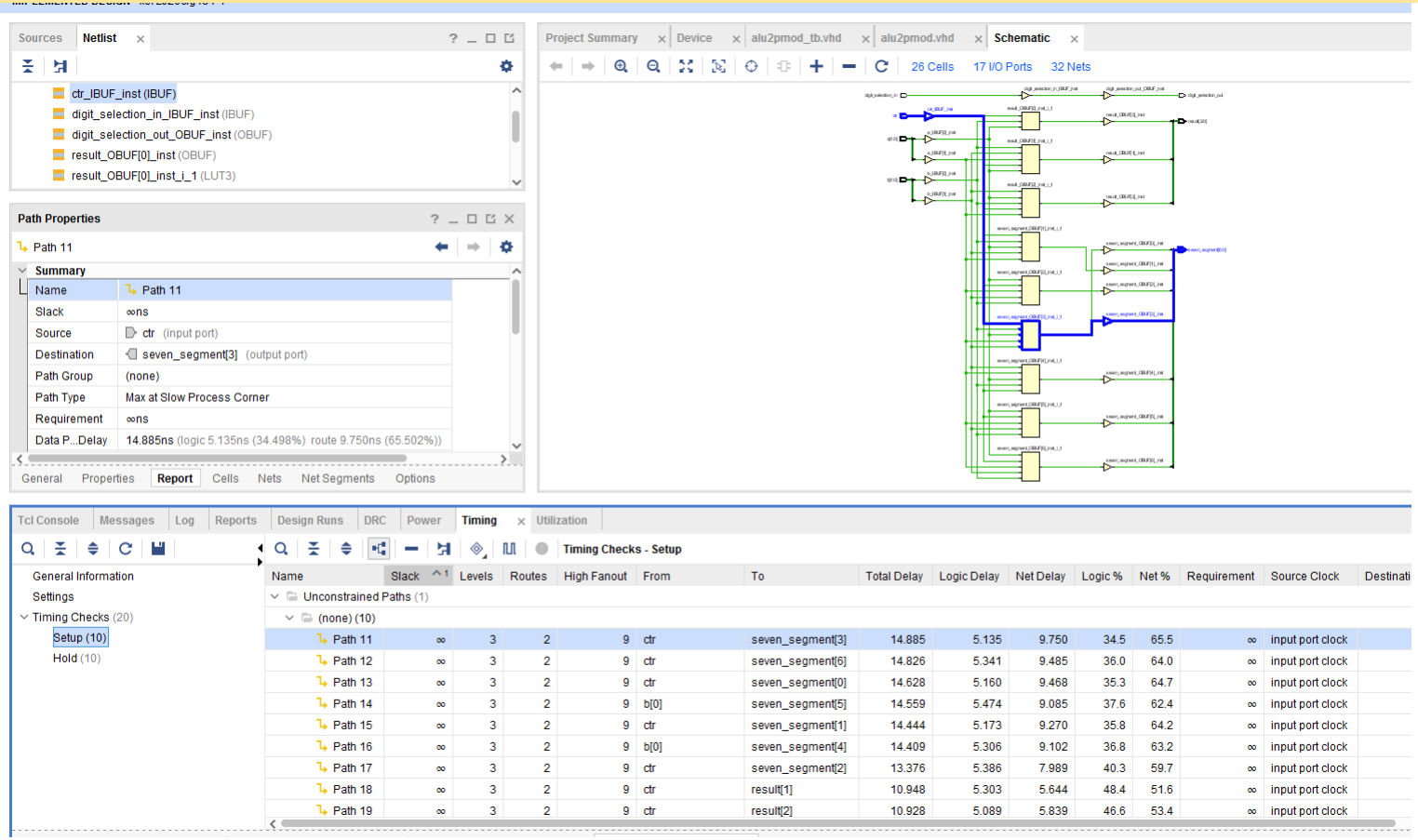




Λεπτομέρειες: https://support.xilinx.com/s/article/63514?language=en_US

3η Εργαστηριακή Άσκηση

Implementation – Timing Reports (1/2)

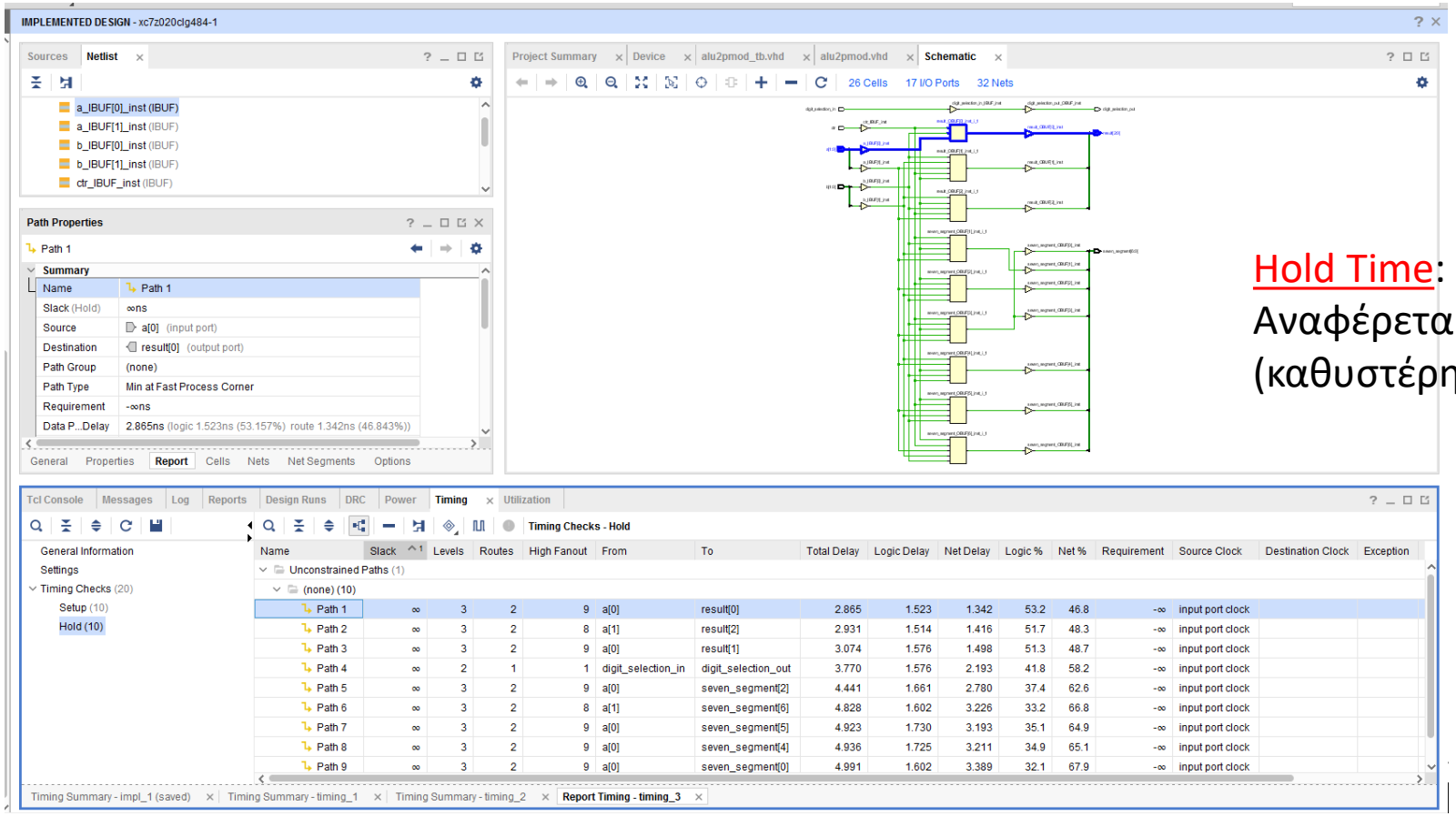


Menu Reports->Timing-Timing Reports

Setup Time:
Αναφέρεται στις αργές διαδρομές (καθυστέρηση διάδοσης)

3η Εργαστηριακή Άσκηση

Implementation – Timing Reports (2/2)

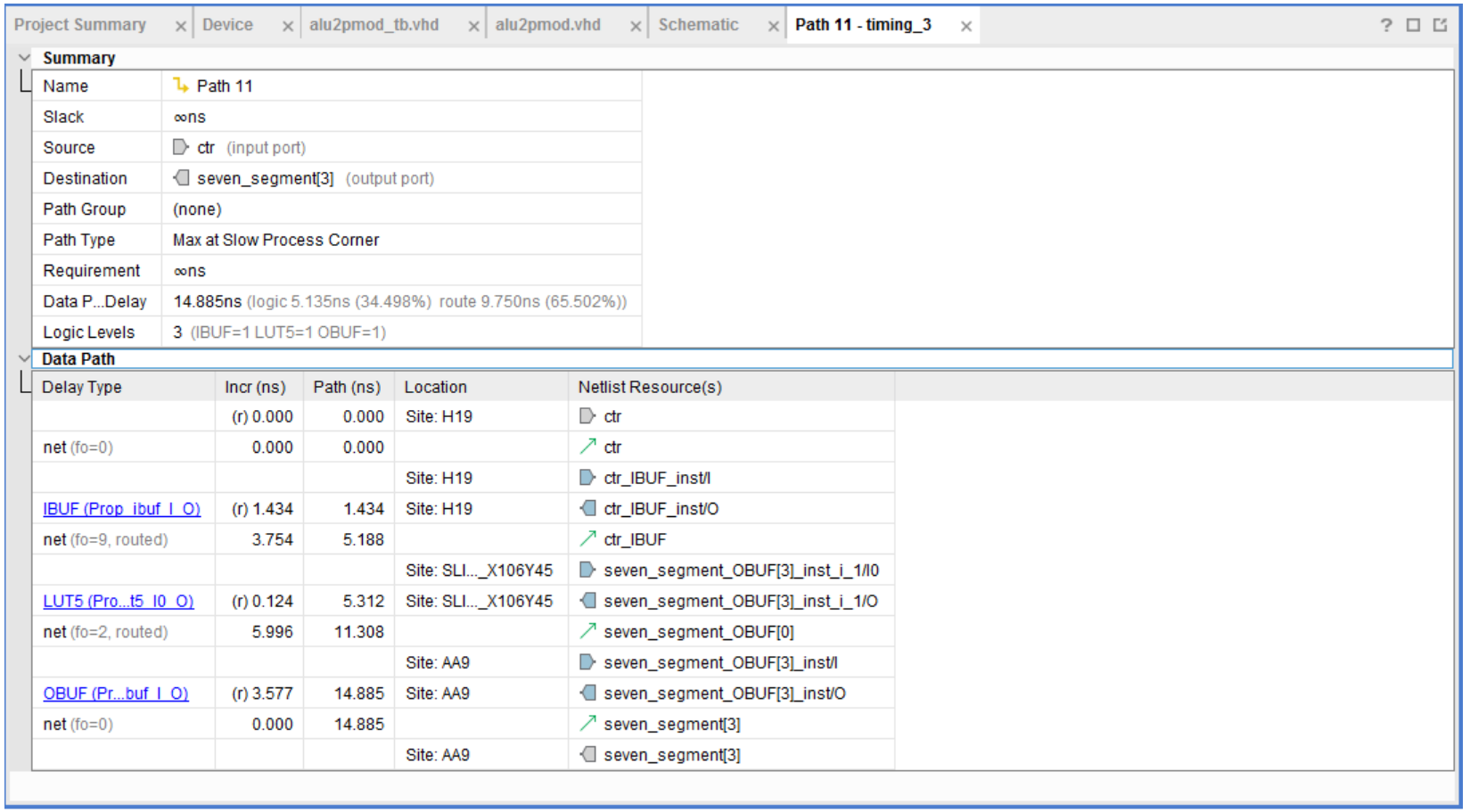


Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Logic %	Net %	Requirement	Source Clock	Destination Clock	Exception
Unconstrained Paths (1)															
(none) (10)															
Path 1	∞	3	2	9	a[0]	result[0]	2.865	1.523	1.342	53.2	46.8	-∞	input port clock		
Path 2	∞	3	2	8	a[1]	result[2]	2.931	1.514	1.416	51.7	48.3	-∞	input port clock		
Path 3	∞	3	2	9	a[0]	result[1]	3.074	1.576	1.498	51.3	48.7	-∞	input port clock		
Path 4	∞	2	1	1	digit_selection_in	digit_selection_out	3.770	1.576	2.193	41.8	58.2	-∞	input port clock		
Path 5	∞	3	2	9	a[0]	seven_segment[2]	4.441	1.661	2.780	37.4	62.6	-∞	input port clock		
Path 6	∞	3	2	8	a[1]	seven_segment[6]	4.828	1.602	3.226	33.2	66.8	-∞	input port clock		
Path 7	∞	3	2	9	a[0]	seven_segment[5]	4.923	1.730	3.193	35.1	64.9	-∞	input port clock		
Path 8	∞	3	2	9	a[0]	seven_segment[4]	4.936	1.725	3.211	34.9	65.1	-∞	input port clock		
Path 9	∞	3	2	9	a[0]	seven_segment[0]	4.991	1.602	3.389	32.1	67.9	-∞	input port clock		

Hold Time:
Αναφέρεται στις γρήγορες διαδρομές (καθυστέρηση μόλυνσης)

3η Εργαστηριακή Άσκηση

Implementation – Path Analysis



Project Summary | Device | alu2pmod_tb.vhd | alu2pmod.vhd | Schematic | **Path 11 - timing_3**

Summary

Name	Path 11
Slack	∞ns
Source	ctr (input port)
Destination	seven_segment[3] (output port)
Path Group	(none)
Path Type	Max at Slow Process Corner
Requirement	∞ns
Data P...Delay	14.885ns (logic 5.135ns (34.498%) route 9.750ns (65.502%))
Logic Levels	3 (IBUF=1 LUT5=1 OBUF=1)

Data Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
	(r) 0.000	0.000	Site: H19	ctr
net (fo=0)	0.000	0.000		ctr
			Site: H19	ctr_IBUF_inst/I
IBUF (Prop_ibuf_I_O)	(r) 1.434	1.434	Site: H19	ctr_IBUF_inst/O
net (fo=9, routed)	3.754	5.188		ctr_IBUF
			Site: SLI..._X106Y45	seven_segment_OBUF[3]_inst_i_1/I0
LUT5 (Pro...t5_I0_O)	(r) 0.124	5.312	Site: SLI..._X106Y45	seven_segment_OBUF[3]_inst_i_1/O
net (fo=2, routed)	5.996	11.308		seven_segment_OBUF[0]
			Site: AA9	seven_segment_OBUF[3]_inst/I
OBUF (Pr...buf_I_O)	(r) 3.577	14.885	Site: AA9	seven_segment_OBUF[3]_inst/O
net (fo=0)	0.000	14.885		seven_segment[3]
			Site: AA9	seven_segment[3]

3η Εργαστηριακή Άσκηση

Implementation – Simulation (1/3)

```
test_1: process is
begin
ctr_tb<= not ctr_tb;
for i in 0  to 3 loop
        a_tb<=std_logic_vector(to_unsigned(i,a_tb'length));
        for j in 0 to 3 loop
                b_tb<=std_logic_vector(to_unsigned(j,a_tb'length));
                wait for 10ns;
        end loop j;
end loop i;
end process test_1;
```

3η Εργαστηριακή Άσκηση

Implementation – Simulation (2/3)

```
test_2: process is
procedure sim_test (a: in integer; b: in integer; ctr: in std_logic)  is
begin
for i in 0 to a loop
        a_tb<=std_logic_vector(to_unsigned(i,a_tb'length));
        for j in 0 to b loop
                b_tb<=std_logic_vector(to_unsigned(j,a_tb'length));
                wait for 10ns;
        end loop j;
end loop i;
end procedure sim_test;
begin
        sim_test(a_tb'length**2-1, b_tb'length**2-1,'0');
        sim_test(a_tb'length**2-1, b_tb'length**2-1,'0');
end process test_2;
```

3η Εργαστηριακή Άσκηση

Implementation – Simulation (3/3)

```
test_2: process is
procedure sim_test (a: in integer; b: in integer; ctr: in std_logic)  is
begin
for i in 0 to a loop
        a_tb<=std_logic_vector(to_unsigned(i,a_tb'length));
        for j in 0 to b loop
                b_tb<=std_logic_vector(to_unsigned(j,a_tb'length));
                wait for 10ns;
        end loop j;
end loop i;
end procedure sim_test;
begin
        sim_test(a_tb'length**2-1, b_tb'length**2-1,'0');
        sim_test(a_tb'length**2-1, b_tb'length**2-1,'0');
end process test_2;
```

dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

Εργαστήριο Λογικής Σχεδίασης

VHDL - Εισαγωγή

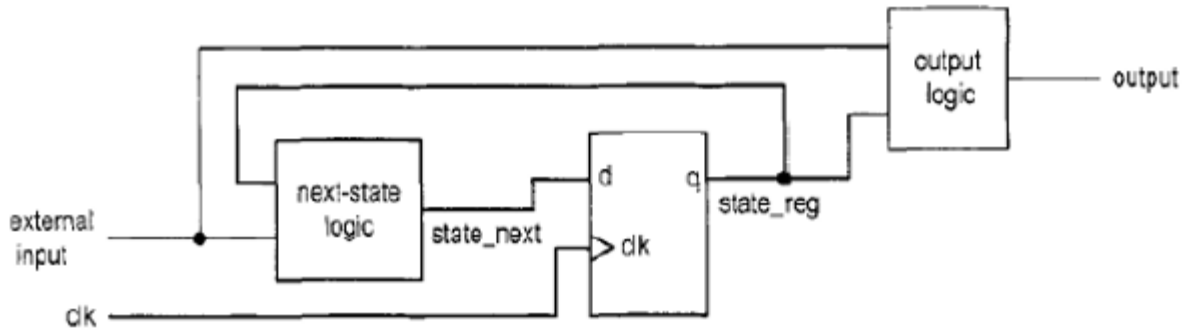
Βασιλόπουλος Διονύσης

ΕΤΕΠ Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

VHDL – Ακολουθιακά Κυκλώματα

- Οι έξοδοι Q_t (τιμή εξόδου τη χρονική στιγμή t) των ακολουθιακών κυκλωμάτων εξαρτώνται όχι μόνο από τις τρέχουσες τιμές των εισόδων, αλλά και από τις προηγούμενες τιμές των εξόδων $Q_{t-1.}$ Στο κύκλωμα αυτό εμφανίζετε ως ανάδραση
- 'Εχουν μνήμη (κατάσταση-state). Για N αποθηκευμένες καταστάσεις το κύκλωμα χρειάζεται από $\log_2 N$ έως και N bit.
- Οι έξοδοι είναι συνάρτηση των εισόδων και της αποθηκευμένης κατάστασης.



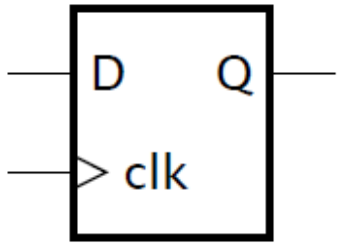
VHDL – Ακολουθιακά Κυκλώματα

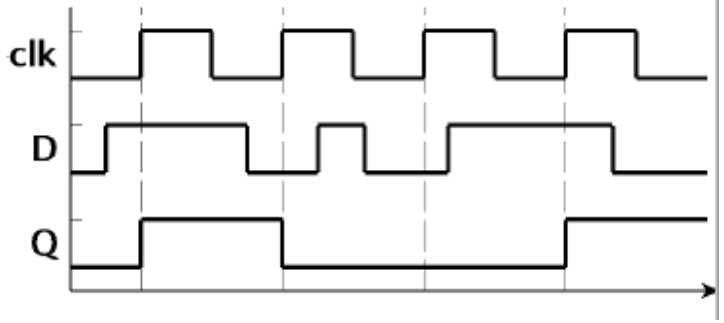
D Flip Flop

- Στοιχείο αποθήκευσης του 1 bit
- Άλλοι τύποι flip-flop
  - JK, T (toggle)

```
entity dff is
   port ( clk,d : in std_logic;
          q     : out std_logic);
end entity;
architecture beh of dff is
begin
   process (clk)
   begin
    if clk'event and clk = '1' then
       q <= d;
    end if;
   end process;
end beh;
```

Μόνο το clk στο sensitivity list





VHDL – Ακολουθιακά Κυκλώματα

D Flip Flop with Enable

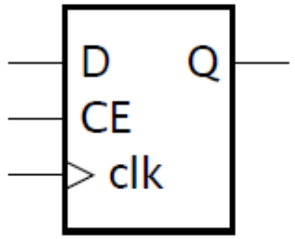
Η αποθήκευση της τιμής D εξαρτάται από το σήμα έγκρισης CE (en)

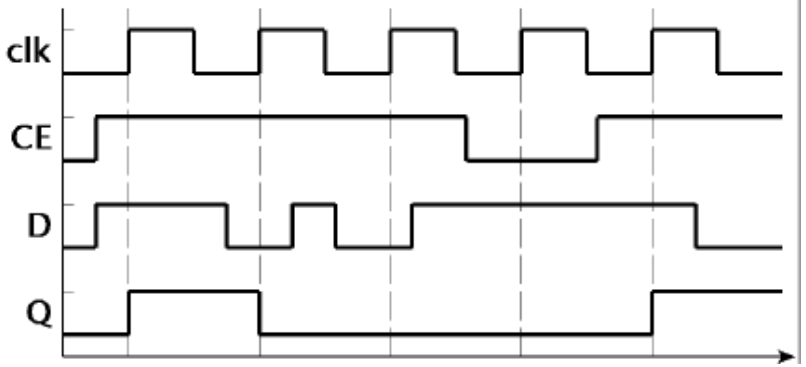
- Αποθηκεύει μόνο όταν CE = 1 σε μια ανοδική ακμή ρολογιού

• Το CE είναι μια σύγχρονη είσοδος ελέγχου

```
entity dff_en is
   port ( clk   : in std_logic;
          ce    : in std_logic;
          d     : in std_logic;
          q     : out std_logic);
end dff_en ;
architecture beh of dff_en is
begin
   process (clk)
   begin
      if clk'event and clk='1' then
         if ce='1' then
            q <= d;
         end if;
      end if;
   end process;
end beh;
```

Σύγχρονο: Πρώτα έλεγχος για το clock



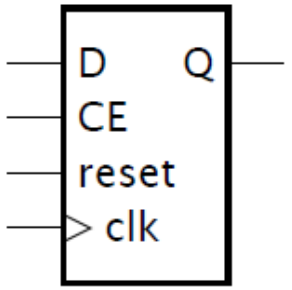


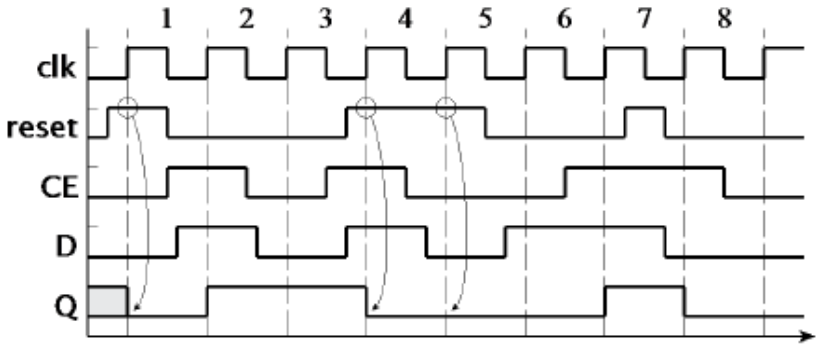
VHDL – Ακολουθιακά Κυκλώματα

D Flip Flop with Reset (σύγχρονο)

- Η είσοδος μηδενισμού (reset) θέτει την αποθηκευμένη τιμή στο 0
  - η είσοδος reset πρέπει να είναι σταθερή γύρω από την ανοδική ακμή του clk

```
entity dff_en_reset is
   port ( clk      : in std_logic;
          ce,reset : in std_logic;
          d        : in std_logic;
          q        : out std_logic);
end dff_en ;
architecture beh of dff_en_reset is
begin
   process (clk)
   begin
      if clk'event and clk='1' then
         if reset = '1' then
           q <= '0';
         elsif ce = '1' then
           q <= d;
         end if;
      end if;
   end process;
end beh;
```



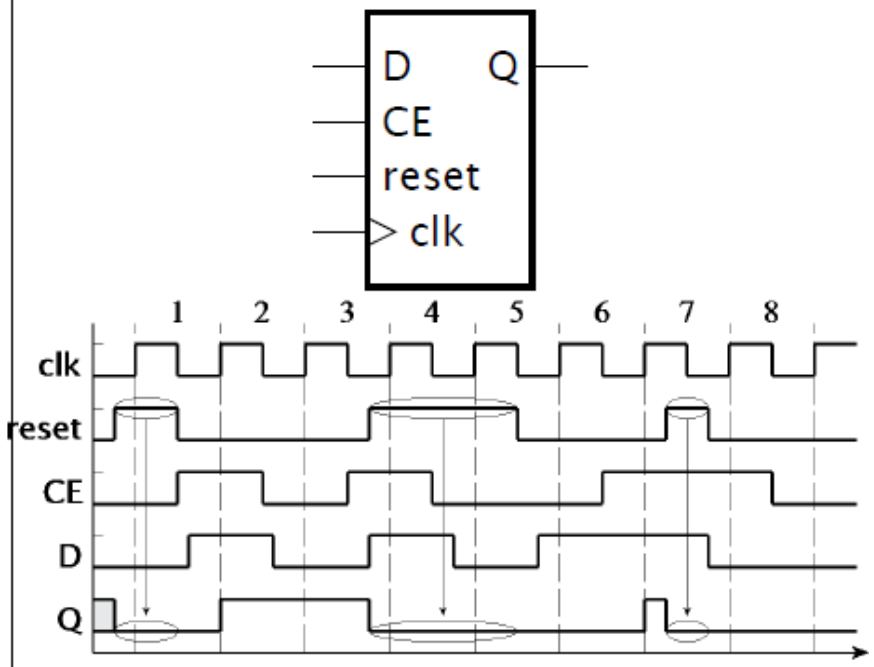


VHDL – Ακολουθιακά Κυκλώματα

D Flip Flop with Reset (ασύγχρονο)

- Η είσοδος μηδενισμού (reset) θέτει την αποθηκευμένη τιμή στο 0
  - το reset μπορεί να γίνει 1 οποιαδήποτε στιγμή, και το αποτέλεσμα είναι άμεσο
  - το συμπεριλαμβάνουμε στη λίστα ευαισθησίας (η διεργασία ανταποκρίνεται άμεσα σε αλλαγή)

```
entity dff_en_areset is
   port ( clk      : in std_logic;
          ce,reset : in std_logic;
          d        : in std_logic;
          q        : out std_logic);
end dff_en ;
architecture beh of dff_en_areset is
begin
   process (clk, reset)
   begin
     if reset = '1' then
        q <= '0';
     elsif clk'event and clk='1' then
       if ce = '1' then
         q <= d;
       end if;
     end if;
   end process;
end beh;
```

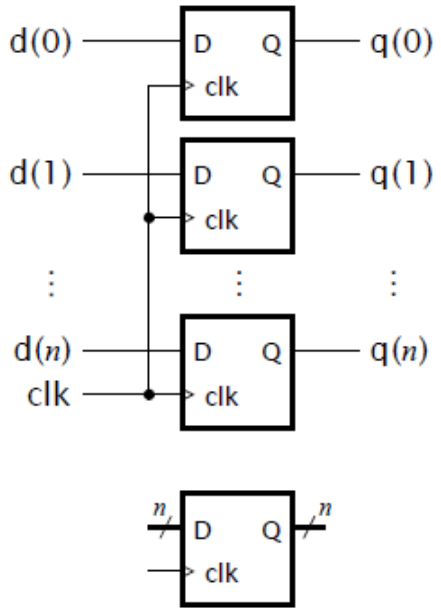


VHDL – Ακολουθιακά Κυκλώματα

Καταχωρητές (D Flip Flop με ασύγχρονο reset)

Αποθηκεύουν μια τιμή πολλαπλών bit

- Ένα flip-flop D ανά bit
- Απαιτεί αλλαγή στο array data type
  - std_logic_vector




```
entity reg_reset is
   port (clk, reset: in std_logic;
      d: in std_logic_vector(7 downto 0);
      q: out std_logic_vector(7 downto 0)
);
end reg_reset;

architecture beh of reg_reset is
begin
   process (clk,reset)
   begin
      if (reset='1') then
         q <= (others=>'0');
      elsif (clk'event and clk='1') then
         q <= d;
      end if;
   end process;
end beh;
```

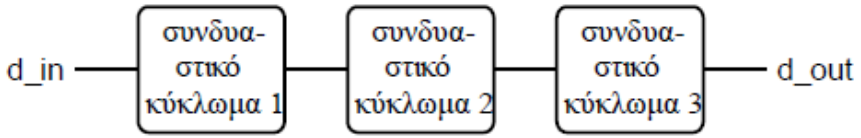
Συντομογραφία του όλα στο 0

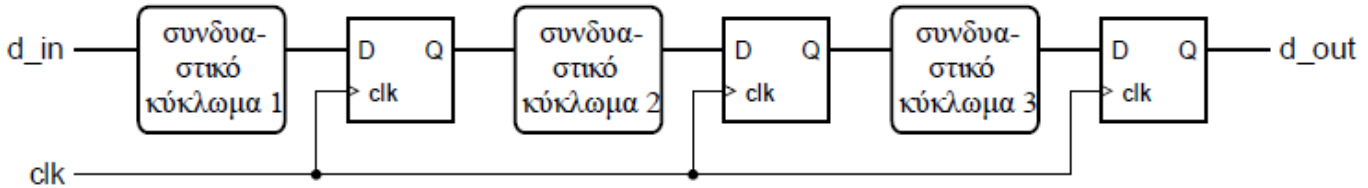
VHDL – Ακολουθιακά Κυκλώματα

Pipelines (διοχέτευση)

Συνολική καθυστέρηση = $Delay_1 + Delay_2 + Delay_3$

Διάστημα μεταξύ των εξόδων > Συνολική καθυστέρηση





Περίοδος ρολογιού = $\max(Delay_1, Delay_2, Delay_3)$

Συνολική καθυστέρηση = 3 × περίοδος ρολογιού

Διάστημα μεταξύ των εξόδων = 1 περίοδος ρολογιού

VHDL – Ακολουθιακά Κυκλώματα

Συσσωρευτής (accumulator 1/2)

- Αθροίστε μια ακολουθία προσημασμένων αριθμών
  - Ένας νέος αριθμός φθάνει όταν data_en = 1
  - Μηδενίστε το άθροισμα με σύγχρονο reset

```
library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;

entity accumulator is
  port ( clk, reset, data_en     : in std_logic;
         data_in                 : in signed(15 downto 0);
         data_out                : out signed(19 downto 0) );
end entity accumulator;
```

VHDL – Ακολουθιακά Κυκλώματα

Συσσωρευτής (accumulator 2/2)

```
architecture rtl of accumulator is
  signal sum, new_sum : signed(19 downto 0);
begin
  new_sum <= sum + resize(data_in, sum'length);
  reg: process (clk) is
  begin
    if rising_edge(clk) then
      if reset = '1' then
        sum <= (others => '0');
      elsif data_en = '1' then
        sum <= new_sum;
      end if;
    end if;
  end process reg;
  data_out <= sum;
end architecture rtl;
```

Εναλλακτικός τρόπος για την ανοδική ακμή του clk. Το rising_edge είναι συνάρτηση για οποιοδήποτε σήμα.

VHDL – Ακολουθιακά Κυκλώματα

Ολισθητές (shift registers 1/3)

- Εκτελούν ολίσθηση (shift) στα αποθηκευμένα δεδομένα
  - Σειριακή μεταφορά δεδομένων
- Καταχωρητής ολίσθησης των 8 bit με ασύγχρονη είσοδο reset, σειριακή είσοδο και παράλληλη έξοδο

```
entity sreg8 is
  port (clk, reset: in bit;
        sin       : in bit;
        dout      : out bit_vector(7 downto 0));
end entity;
architecture bef of sreg8 is
 signal shift_reg : bit_vector(7 downto 0);
begin
        dout <= shift_reg;
        process (clk, reset)
        ...
        end process;
end architecture;
```

Χρήση bit/bit_vector
Προφανώς το std_logic/std_logic_vector είναι προτιμότερο

VHDL – Ακολουθιακά Κυκλώματα

Ολισθητές (shift registers 2/3)

- Υλοποίηση με array slices

```
process (clk, reset)
begin
   if reset = '1' then
      shift_reg <= (others => '0');
   elsif clk'event and clk = '1' then
      shift_reg(7 downto 1) <= shift_reg(6 downto 0);
      shift_reg(0) <= sin;
   end if;
end process;
```

- Υλοποίηση με array concatenation

```
process (clk, reset)
begin
   if reset = '1' then
      shift_reg <= (others => '0');
   elsif clk'event and clk = '1' then
      shift_reg <= shift_reg(6 downto 0) & sin;
   end if;
end process;
```

VHDL – Ακολουθιακά Κυκλώματα

Ολισθητές (shift registers 3/3)

- Υλοποίηση με for-loop

```
process (clk, reset)
begin
   if reset = '1' then
     shift_reg <= (others => '0');
   elsif clk'event and clk = '1' then
     for i in 7 downto 1 loop
     shift_reg(i) <= shift_reg(i-1);
   end loop;
   shift_reg(0) <= sin;
  end if;
end process;
```

VHDL – Ακολουθιακά Κυκλώματα

Μετρητές (counters)

- Αποθηκεύουν την τιμή ενός απρόσημου ακεραίου
  - αυξάνουν ή μειώνουν την τιμή
- Χρησιμοποιούνται για να μετράνε πόσες φορές:
  - έχουν συμβεί κάποια γεγονότα
  - έχει επαναληφθεί ένα βήμα επεξεργασίας
- Χρησιμοποιούνται ως χρονομετρητές (timers)
  - μετράνε πόσα χρονικά διαστήματα έχουν περάσει καθώς αυξάνονται περιοδικά

VHDL – Ακολουθιακά Κυκλώματα

Μετρητές ασύγχρονοι

```
library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;
entity counter4 is
       port (clk, reset : in std_logic;
       count            : out std_logic(3 downto 0);
end entity;

architecture beh of counter4 is
    signal counter : unsigned(3 downto 0);
begin
count <= std_logic_vector(counter);
   process (clk, reset)
   begin
     if reset = '1' then
        counter <= (others => '0');
      elsif clk'event and clk = '1' then
        if counter = 15 then
          counter <= (others => '0');
        else
          counter <= counter + 1;
        end if;
     end if;
   end process;
end architecture;
```

VHDL – Ακολουθιακά Κυκλώματα

Δεκαδικός Μετρητής

```
library ieee; use ieee.std_logic_1164.all, ieee.numeric_std.all;
entity decade_counter is
  port ( clk : in std_logic;  q : out std_logic_vector(3 downto 0) );
end entity decade_counter;

architecture rtl of decade_counter is
  signal count_value : unsigned(3 downto 0);
begin
  count : process (clk) is
  begin
    if rising_edge(clk) then
      count_value <= (count_value + 1) mod 10;
    end if;
  end process count;
  q <= std_logic_vector(count_value);
end architecture rtl;
```

VHDL – Περιοδικό σήμα ελέγχου

```
library ieee; use ieee.std_logic_1164.all, ieee.numeric_std.all;
entity decoded_counter is
  port   ( clk : in std_logic; ctrl : out std_logic );
end entity decoded_counter;

architecture rtl of decoded_counter is
  signal count_value : unsigned(3 downto 0);
begin
  counter : process (clk) is
  begin
    if rising_edge(clk) then
      count_value <= count_value + 1;
    end if;
  end process counter;
  ctrl <= '1' when count_value = "0111" or count_value = "1011" else '0';
end architecture rtl;
```

VHDL – Ακολουθιακά Κυκλώματα

Παράδειγμα

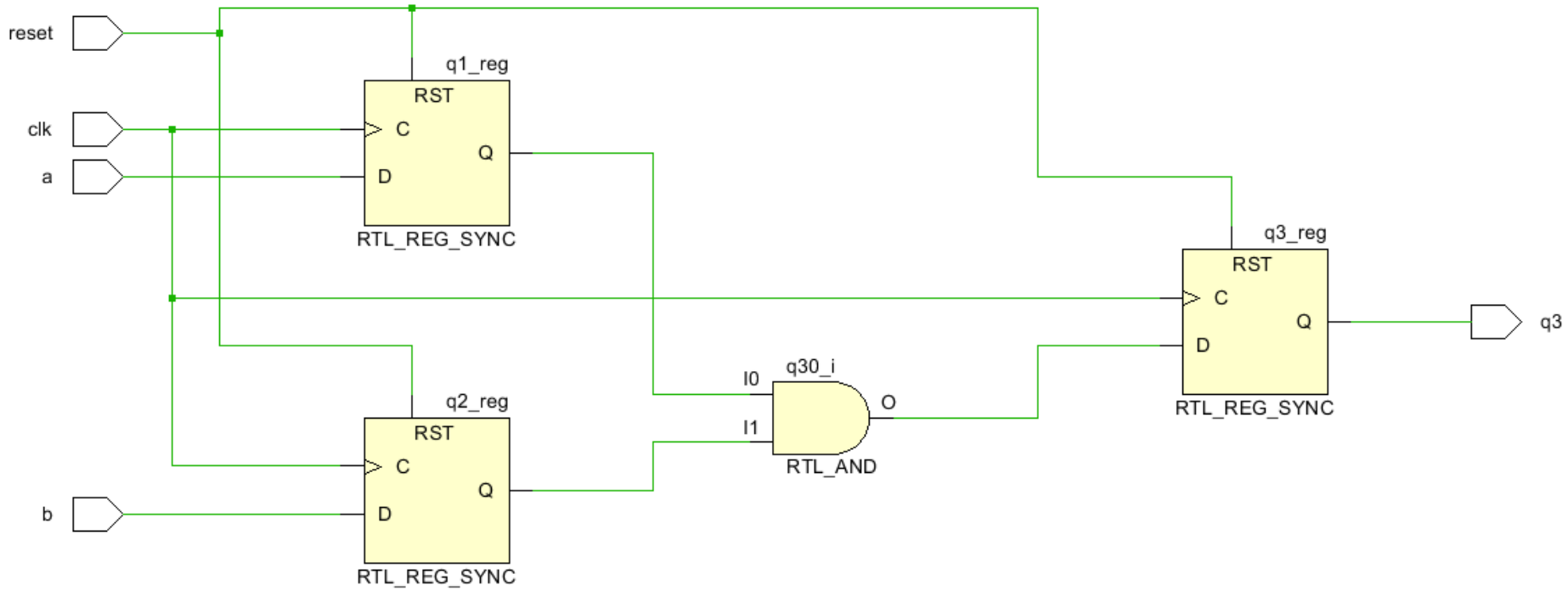
Τι κύκλωμα μοντελοποιεί ο κώδικας;

Πως υπολογίζεται η συχνότητα λειτουργίας του κυκλώματος μετά τη σύνθεση;

```
process (clk) is
begin
   if clk'event and clk='1' then
      if reset = '1' then
         q3<='0';q1<='0';q2<='0';
      else
         q1<=a;
         q2<=b;
         q3<=q1 and q2;
      end if;
   end if;
end process;
```

VHDL – Ακολουθιακά Κυκλώματα

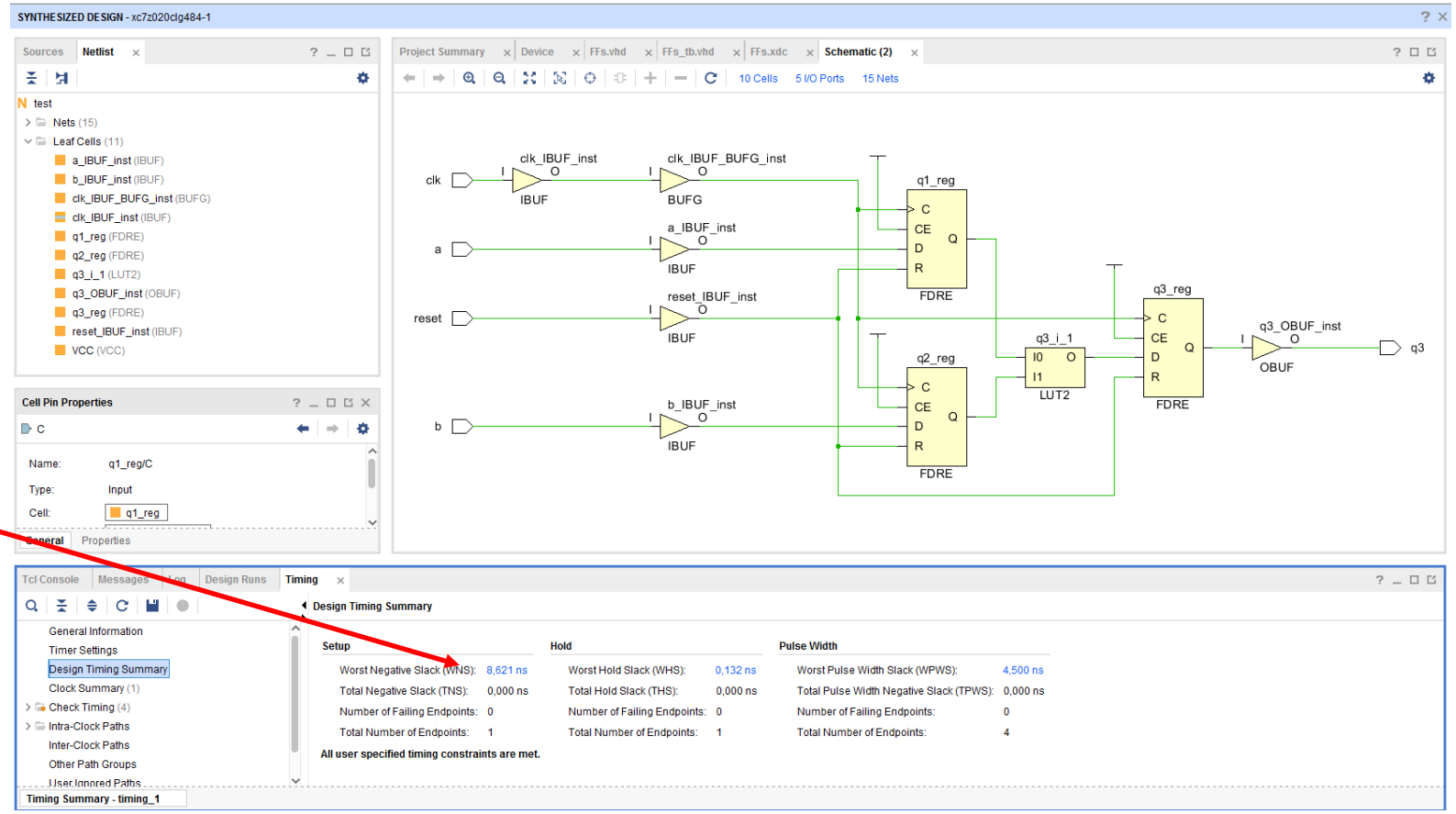
Παράδειγμα



VHDL – Ακολουθιακά Κυκλώματα

Παράδειγμα

Tc=10ns-Slack



dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

Εργαστήριο Λογικής Σχεδίασης

VHDL

Βασιλόπουλος Διονύσης

ΕΤΕΠ Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ

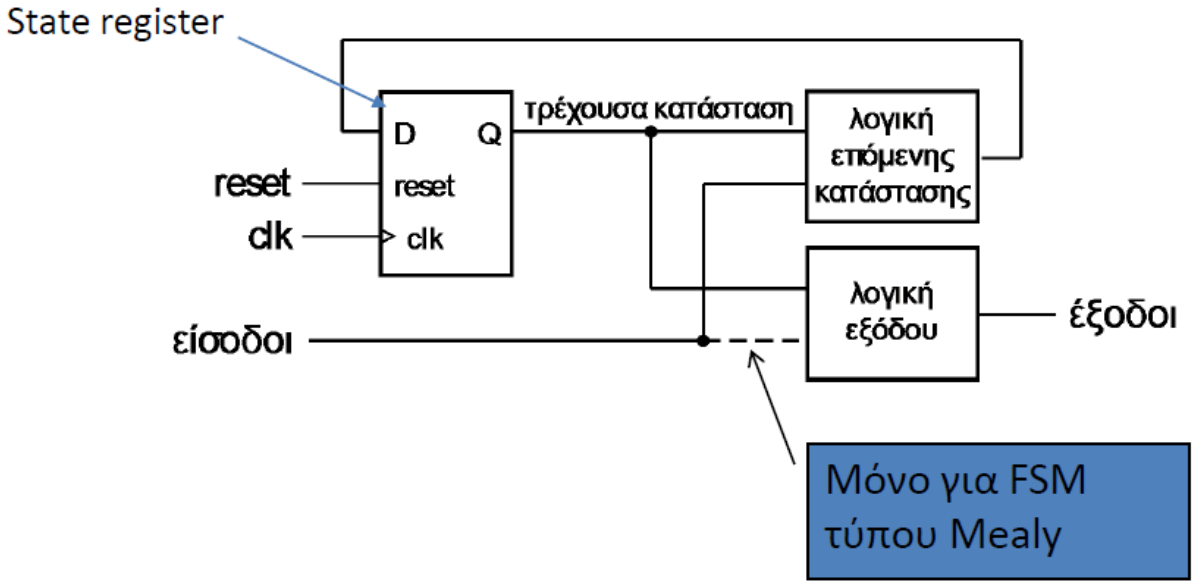
ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

VHDL – Finite State Machines

- Χρησιμοποιούνται για να υλοποιήσουν μια ακολουθία ελέγχου
- Μία FSM (Finite-State Machine – Μηχανή Πεπερασμένων καταστάσεων) ορίζεται από
  - σύνολο εισόδων: I
  - σύνολο εξόδων: O
  - σύνολο καταστάσεων: S
  - αρχική κατάσταση: s0 ανήκει στο S
  - συνάρτηση μετάβασης από μία κατάσταση στην επόμενη
  - συνάρτηση εξόδου

VHDL – Finite State Machines

Γενικό διάγραμμα



VHDL – Finite State Machines

Κωδικοποίηση Καταστάσεων

- Κωδικοποιούνται στο δυαδικό
  - Ν καταστάσεις: χρειάζονται τουλάχιστον log2*N* bit
- Η κωδικοποιημένη τιμή χρησιμοποιείται στα κυκλώματα για τις συναρτήσεις μετάβασης και εξόδου
  - η κωδικοποίηση επηρεάζει την πολυπλοκότητα του κυκλώματος
- Είναι δύσκολο να βρεθεί βέλτιστη κωδικοποίηση
  - τα εργαλεία CAD συνήθως κάνουν την κωδικοποίηση
- Η κωδικοποίηση one-hot δουλεύει καλά στα FPGA
- Συχνά χρησιμοποιείται το 000...0 για την κατάσταση αδράνειας
  - μηδενίζει τον καταχωρητή στην κατάσταση αδράνειας

VHDL – Finite State Machines

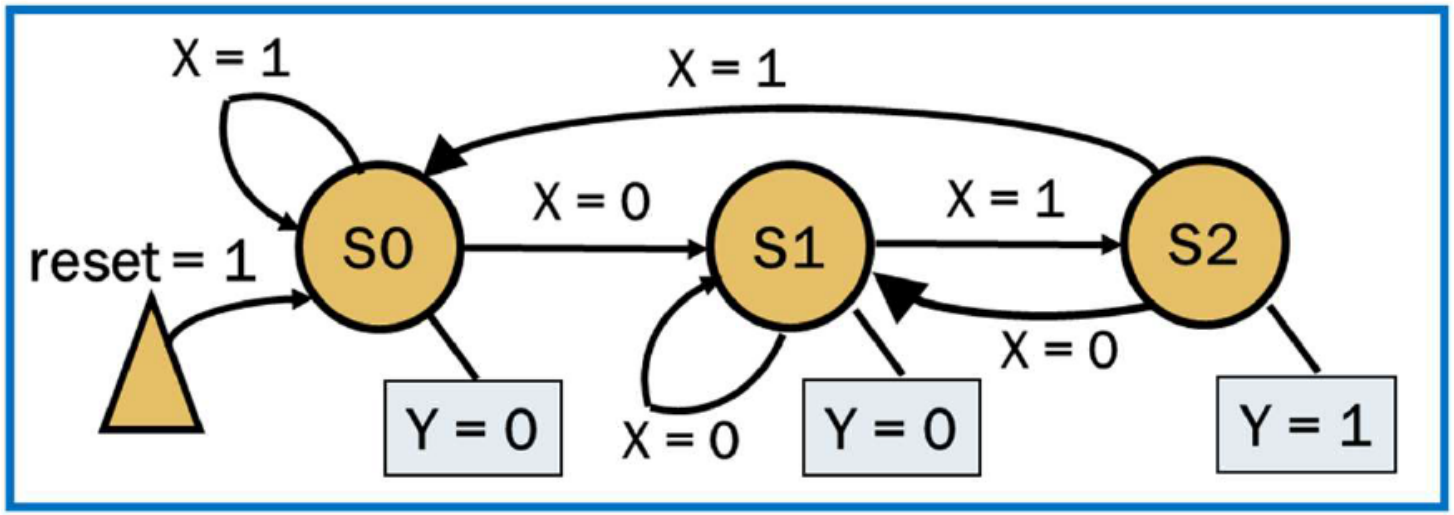
Κωδικοποίηση Καταστάσεων - Type

- Χρησιμοποιούμε έναν τύπο απαρίθμησης για τις τιμές των καταστάσεων
  - αφηρημένο, έτσι αποφεύγουμε να προδιαγράψουμε την κωδικοποίηση

```
type state_type is (zero, edge, one);
signal state_reg, state_next: state_type;
...
```

VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 - Moore



VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 - Moore

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PATTERN_FSM is
   Port ( CLK : in STD_LOGIC;
          RESET : in STD_LOGIC;
          X : in STD_LOGIC;
          Y : out STD_LOGIC);
end PATTERN_FSM;

architecture Behavioral of PATTERN_FSM is

-- state definition
   type FSM_states is (S0, S1, S2);

-- internal signals
   signal current_state, next_state: FSM_states;
   signal X_in : STD_LOGIC; -- Only when there is an INREG
```

ΛΟΓΙΚΗ για RESET

```
begin
-- Optional for synchronization. RESET case
   INREG:  process (CLK)
   begin
      if (CLK = '1' and CLK'event) then
         if (RESET = '1') then X_in <= '1'; -- to trap state S0 during reset
         else X_in <= X;
         end if;
      end if;
   end process;

-- Common process for all FSMs to create state register
   SYNC:  process (CLK)
   begin
      if (CLK = '1' and CLK'event) then
         if (RESET = '1') then current_state <= S0;
         else current_state <= next_state;
         end if;
      end if;
   end process;
```

VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 - Moore

```
-- Process to create next state logic and output logic
   ASYNC: process (current_state, X_in)  -- Moore
   begin
   -- FSM next state and output initialization
      next_state <= S0;
      Y <= '0';
      case current_state is
         when S0 =>
            if (X_in = '0') then next_state <= S1;
            else next_state <= S0;
            end if;
         when S1 =>
            if (X_in = '1') then next_state <= S2;
            else next_state <= S1;
            end if;
         when S2 => Y <= '1';
            if (X_in = '0') then next_state <= S1;
            else next_state <= S0;
            end if;
         -- fail-safe behavior
         when others => next_state <= S0;
      end case;
   end process;

end Behavioral;
```

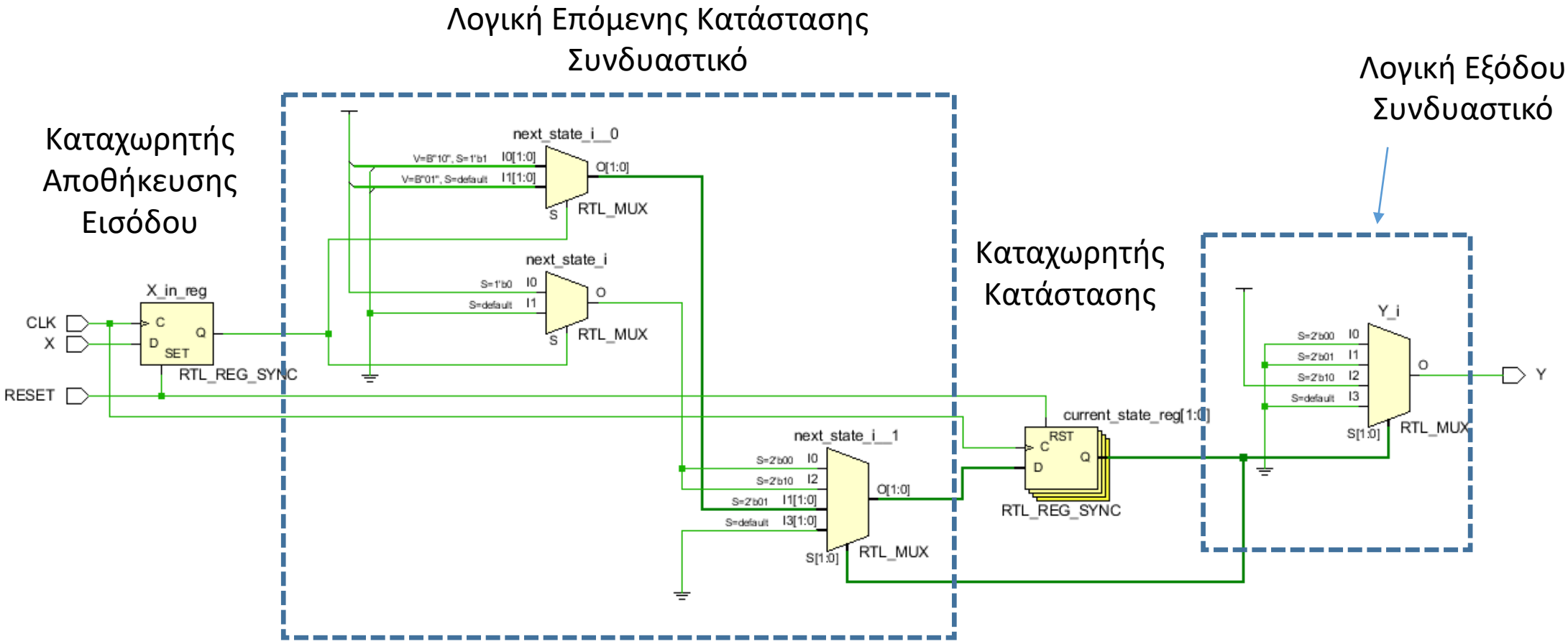
Αρχική τιμή οι τιμές Reset

ΛΟΓΙΚΗ για ΕΠΟΜΕΝΗ ΚΑΤΑΣΤΑΣΗ

ΛΟΓΙΚΗ για ΕΞΟΔΟ

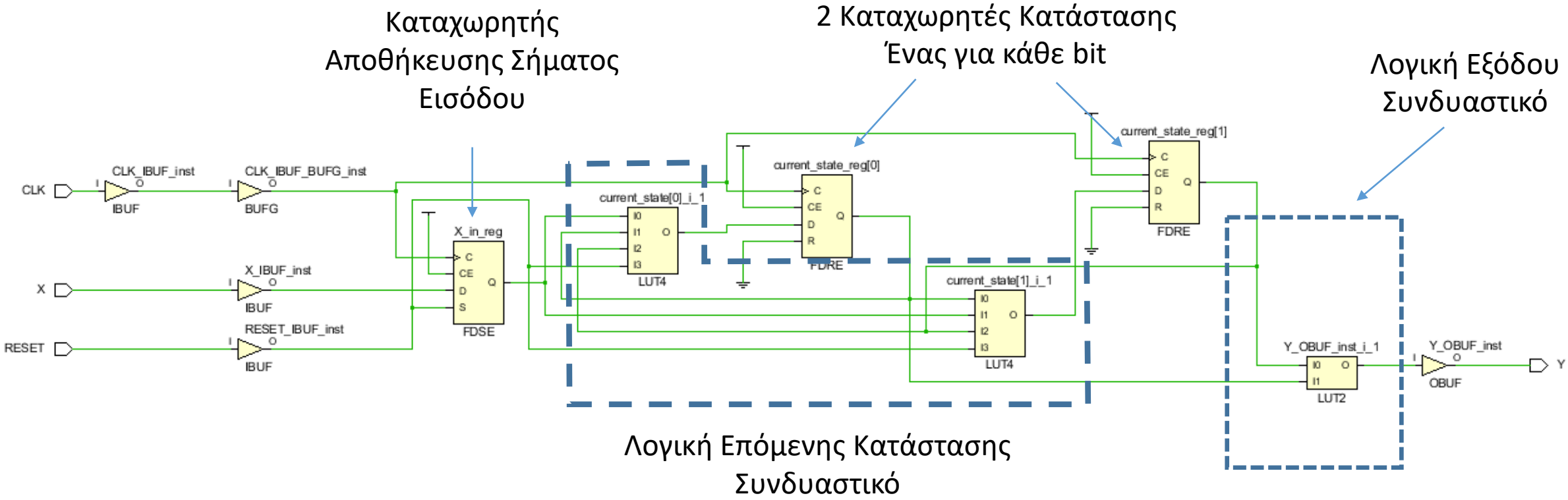
VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 – Moore - RTL



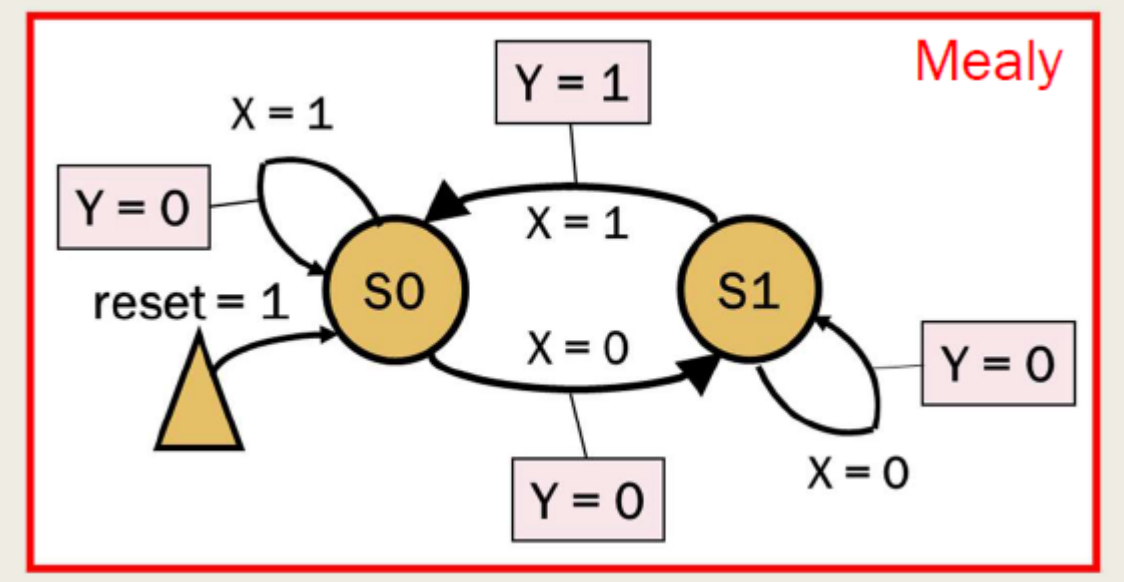
VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 – Moore - Synthesis



VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 - Mealy



VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 - Moore

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PATTERN_FSM is
   Port ( CLK : in STD_LOGIC;
          RESET : in STD_LOGIC;
          X : in STD_LOGIC;
          Y : out STD_LOGIC);
end PATTERN_FSM;

architecture Behavioral of PATTERN_FSM is

-- state definition
   type FSM_states is (S0, S1);

-- internal signals
   signal current_state, next_state: FSM_states;
   signal X_in : STD_LOGIC; -- Only when there is an INREG
```

```
begin
-- Optional for sychronization
   INREG:  process (CLK)
   begin
      if (CLK = '1' and CLK'event) then
         if (RESET = '1') then X_in <= '1'; -- to trap state S0 during reset
         else X_in <= X;
         end if;
      end if;
   end process;

-- Common process for all FSMs to create state register
   SYNC:  process (CLK)
   begin
      if (CLK = '1' and CLK'event) then
         if (RESET = '1') then current_state <= S0;
         else current_state <= next_state;
         end if;
      end if;
   end process;
```

VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 - Mealy

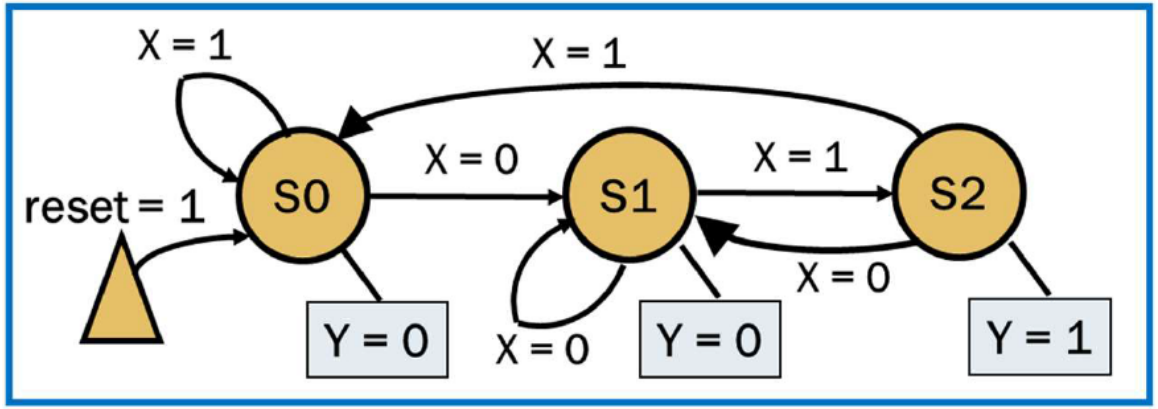
```
-- Process to create next state logic and output logic
   ASYNC: process (current_state, X_in)  -- Moore
   begin
   -- FSM next state and output initialization
      next_state <= S0;
      Y <= '0';
      case current_state is
         when S0 =>
            if (X_in = '0') then next_state <= S1;
            else next_state <= S0;
            end if;
         when S1 =>
            if (X_in = '1') then
                  next_state <= S0;
                  Y<='1';
            else next_state <= S1;
            end if;
-- fail-safe behavior
            when others => next_state <= S0;
      end case;
   end process;
end Behavioral;
```

ΛΟΓΙΚΗ για ΕΞΟΔΟ

VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 – Moore – Πίνακας Αληθείας

S_{old}	$S_{old}(1)$	$S_{old}(0)$	Xin	$S_{new}(1)$	$S_{new}(0)$	S_{new}	Y
S_0	0	0	0	0	1	S_1	0
S_1	0	1	0	0	1	S_1	0
S_2	1	0	0	0	1	S_1	0
S_3	1	1	0	0	0	S_0	0
S_0	0	0	1	0	0	S_0	0
S_1	0	1	1	1	0	S_2	1
S_2	1	0	1	0	0	S_0	0
S_3	1	1	1	0	0	S_0	0



VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 – Moore – Πίνακας Αληθείας

Πίνακες KARNAUGH

$S_{new}(1)$

S_{old}\Xin	0	1
00	0	0
01	0	1
11	0	1
10	0	0

$S_{new}(0)$

S_{old}\Xin	0	1
00	1	0
01	1	0
11	0	0
10	1	0

Y

S_{old}\Xin	0	1
00	0	0
01	0	1
11	0	0
10	0	0

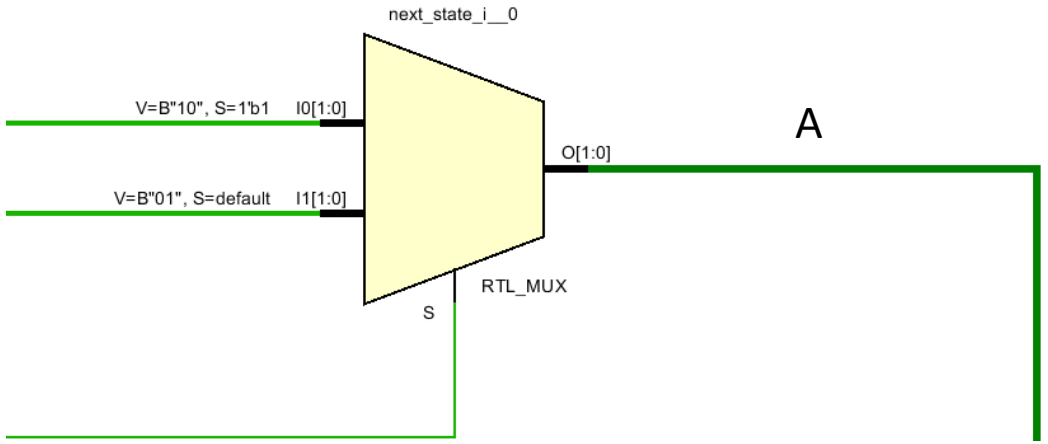
$S_{new}(1)=S_{old}(0)Xin$
$S_{new}(0)=S_{old}(1)'Xin'+S_{old}(0)'Xin'$
$Y = S_{old}(1)S_{old}(0)' Xin$

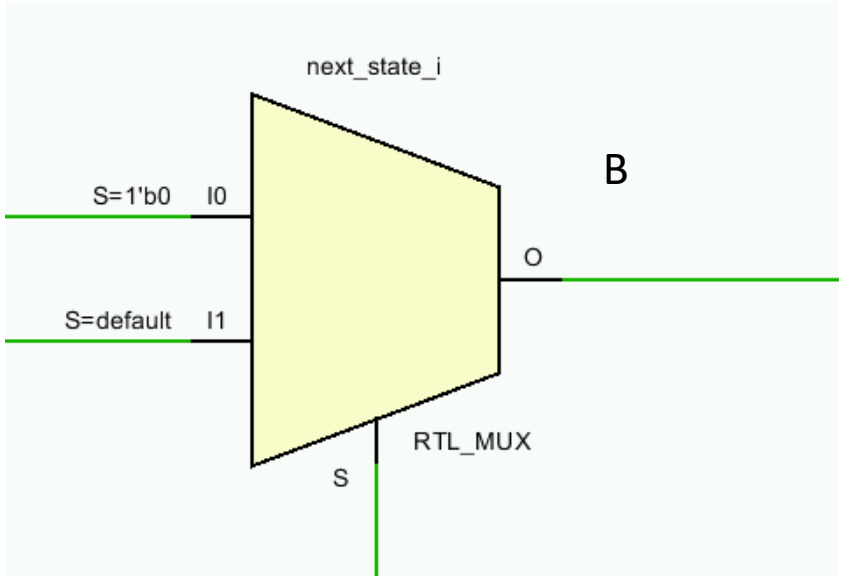
VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 – Moore – Πίνακας Αληθείας

X_{in}=S	Out	$S1_{new}$
1	10	S_2
0	01	S_1

X_{in}=S	Out	$S1_{new}$
0	1	S_1
1	0	S_0





VHDL – Finite State Machines

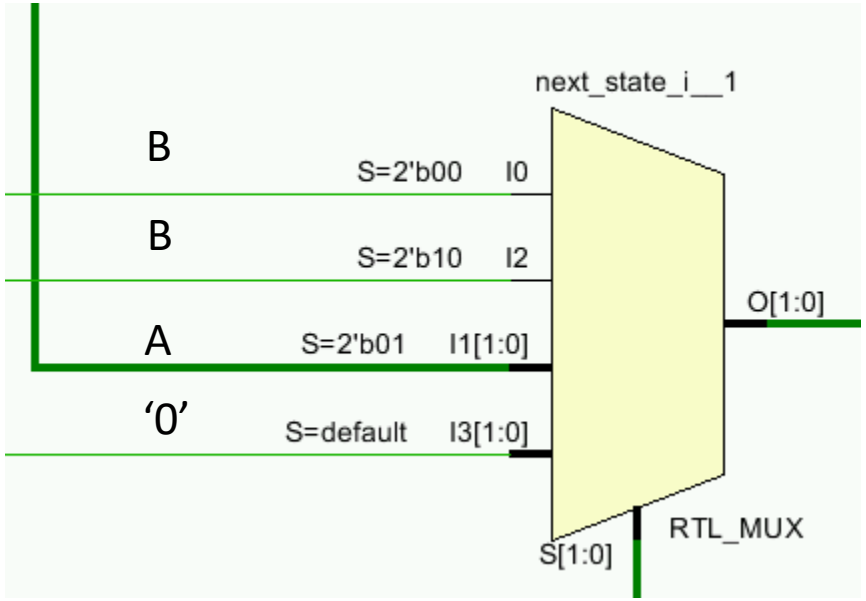
Παράδειγμα – Ανίχνευση ακολουθίας 01 – Moore – Πίνακας Αληθείας

A

X_{in}=S	Out	S_{new}
1	10	S_2
0	01	S_1

B

X_{in}=S	Out	S_{new}
0	01	S_1
1	00	S_0



S_{old}	X_{in}	S_{new}
00(0)	0	S_1
01(1)	0	S_1
10(2)	0	S_1
11(X)	0	S_0
00(0)	1	S_0
01(1)	1	S_2
10(2)	1	S_0
11(X)	1	S_0

dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

Εργαστήριο Λογικής Σχεδίασης

4ο Εργαστηριακό Μάθημα

Βασιλόπουλος Διονύσης

ΕΤΕΠ Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

4η Εργαστηριακή Άσκηση

Ανάλυση άσκησης

Δημιουργία νέου Ρολογιού
=
100 εκ. χτύποι του CLK της κάρτας

Υπολογισμός επόμενης κατάστασης

Ενημέρωση Τρέχουσας Κατάστασης

Υπολογισμός εξόδου

4η Εργαστηριακή Άσκηση

ΠΡΟΧΩΡΗΣΤΕ ΣΤΗΝ ΑΣΚΗΣΗ

4η Εργαστηριακή Άσκηση

Αρχιτεκτονική (1/3) – Νέο Ρολόι (Περίοδος = 1 sec)

Αρχιτεκτονική →

```
One_sec_clk : process (clk, reset) is
variable clk_ticks : integer;

begin
        if reset = '1' then
          clk_100MHz<='0';
          clk_ticks :=0;
        elsif rising_edge(clk) then
                if clk_ticks = 99999999 then
                        clk_ticks := 0;
                        clk_100MHz<= '1';
                else
                        clk_ticks := clk_ticks + 1;
                        clk_100MHz <= '0';
                end if; -- clk_ticks
        end if; --reset
end process One_sec_clk;
```

4η Εργαστηριακή Άσκηση

Αρχιτεκτονική (2/3) – Tick Counter

Αρχιτεκτονική →

```
count: process (clk_100MHz, reset) is
begin
if (reset='1') then
   counts<=(others=>'0');
elsif rising_edge(clk_100MHz) then
   if direction='1' then
      if counts<9 then
        counts<=counts+1;
      else
         counts<=(others=>'0');
      end if; -- counts<99
   else
      if counts>0 then
         counts<=counts-1;
      else
         counts<="1001";
      end if; -- counts>0
   end if; --direction
end if; --reset
end process count;
```

4η Εργαστηριακή Άσκηση

Αρχιτεκτονική (3/3) – Output signals

Αρχιτεκτονική →

```
led_result<=std_logic_vector(counts);

counts_digit_values: process (counts) begin
case counts is
   when X"0" => seven_segment   <="0111111";    -- 0
   when X"1" => seven_segment   <="0000110";    -- 1
   when X"2" => seven_segment   <="1011011";    -- 2
   when X"3" => seven_segment   <="1001111";    -- 3
   when X"4" => seven_segment   <="1100110";    -- 4
   when X"5" => seven_segment   <="1101101";    -- 5
   when X"6" => seven_segment   <="1111101";    -- 6
   when X"7" => seven_segment   <="0000111";    -- 7
   when X"8" => seven_segment   <="1111111";    -- 8
   when X"9" => seven_segment   <="1101111";    -- 9
   when others => seven_segment <="0000000";
end case;
end process counts_digit_values;

digit_selection_out<=digit_selection_in;
```

4η Εργαστηριακή Άσκηση

Constraints (1/2) – CLK + Switches + Leds

Constraints →

```
# CLK - Zedboard 100MHz oscillator
set_property -dict { PACKAGE_PIN Y9 IOSTANDARD LVCMOS33 } [get_ports {clk}]

##############################
# On-board Slide Switches  #
##############################
set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { digit_selection_in }];
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { reset }];
set_property -dict { PACKAGE_PIN F22   IOSTANDARD LVCMOS33 } [get_ports { direction }];

##############################
# On-board LEDS  #
##############################

set_property -dict { PACKAGE_PIN T22   IOSTANDARD LVCMOS33 } [get_ports { led_result[0] }];
set_property -dict { PACKAGE_PIN T21   IOSTANDARD LVCMOS33 } [get_ports { led_result[1] }];
set_property -dict { PACKAGE_PIN U22   IOSTANDARD LVCMOS33 } [get_ports { led_result[2] }];
set_property -dict { PACKAGE_PIN U21   IOSTANDARD LVCMOS33 } [get_ports { led_result[3] }];
```

4η Εργαστηριακή Άσκηση

Constraints (1/2) – Pmod

Constraints →

```
##############################
# PmodSSO                    #
##############################

set_property -dict { PACKAGE_PIN Y11   IOSTANDARD LVCMOS33 } [get_ports { seven_segment[0] }];
set_property -dict { PACKAGE_PIN AA11  IOSTANDARD LVCMOS33 } [get_ports { seven_segment[1] }];
set_property -dict { PACKAGE_PIN Y10   IOSTANDARD LVCMOS33 } [get_ports { seven_segment[2] }];
set_property -dict { PACKAGE_PIN AA9   IOSTANDARD LVCMOS33 } [get_ports { seven_segment[3] }];
set_property -dict { PACKAGE_PIN W12   IOSTANDARD LVCMOS33 } [get_ports { seven_segment[4] }];
set_property -dict { PACKAGE_PIN W11   IOSTANDARD LVCMOS33 } [get_ports { seven_segment[5] }];
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { seven_segment[6] }];

set_property -dict { PACKAGE_PIN W8   IOSTANDARD LVCMOS33 } [get_ports { digit_selection_out }];
```

4η Εργαστηριακή Άσκηση

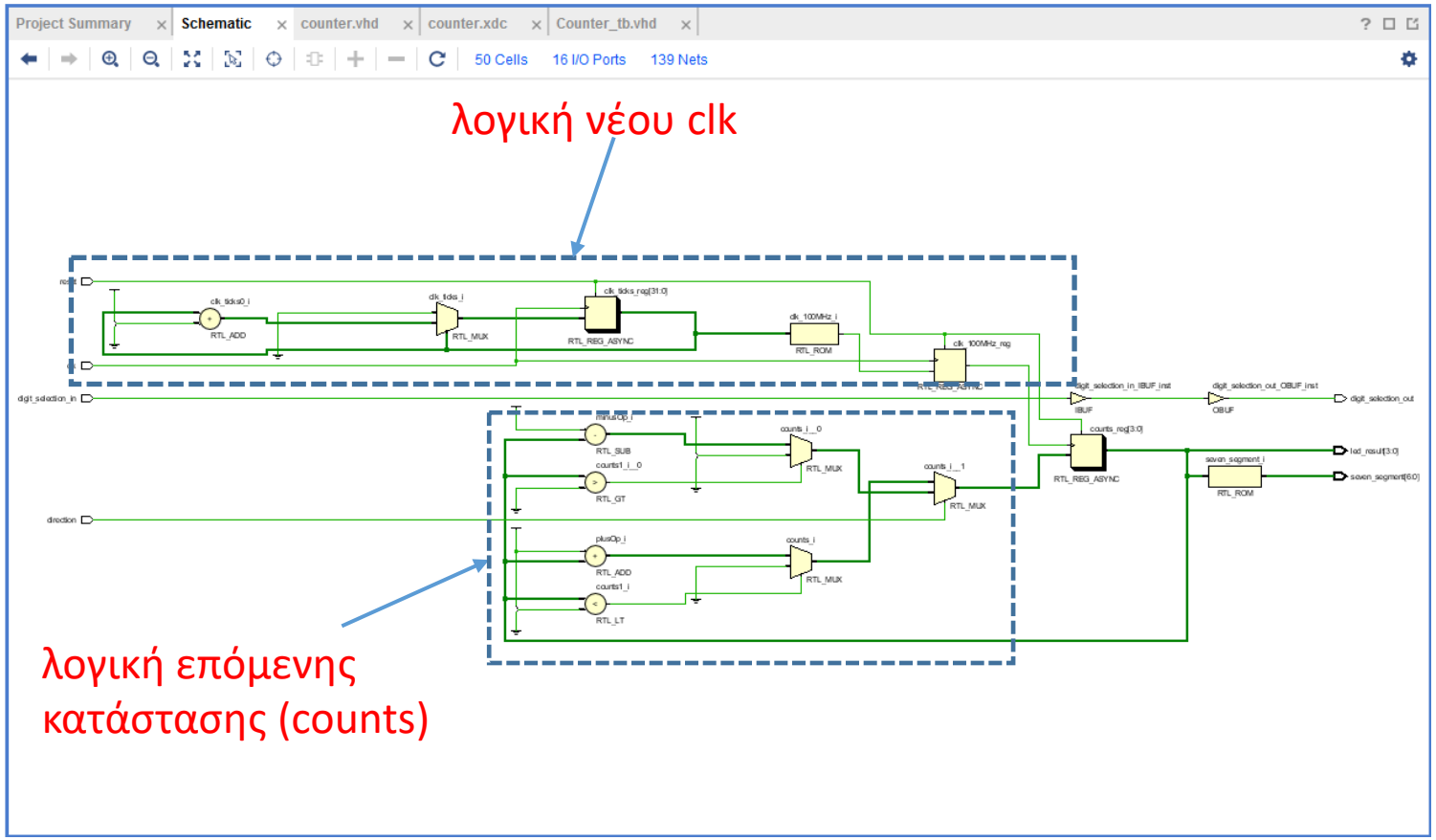
Pmod Manual

Manual Pmod

https://reference.digilentinc.com/reference/pmod/pmodssd/reference-manual

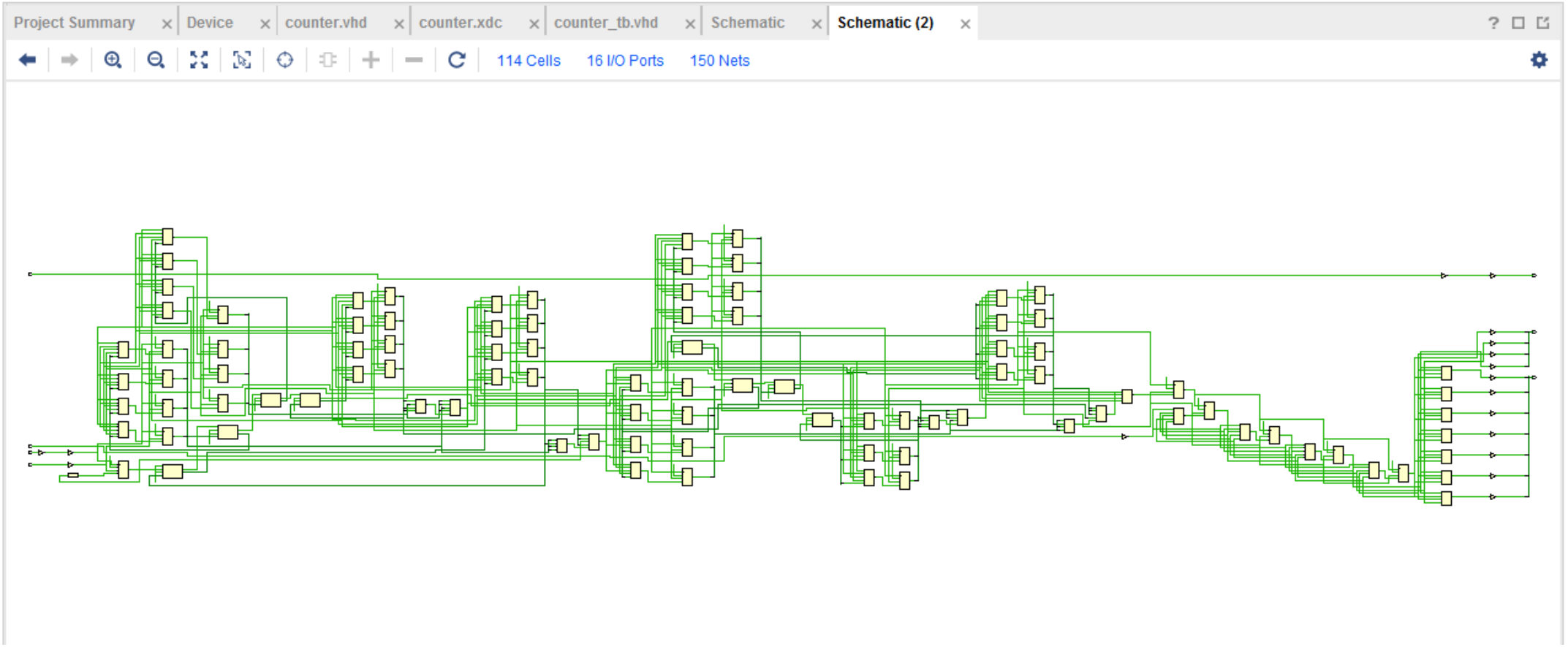
4η Εργαστηριακή Άσκηση

RTL Design



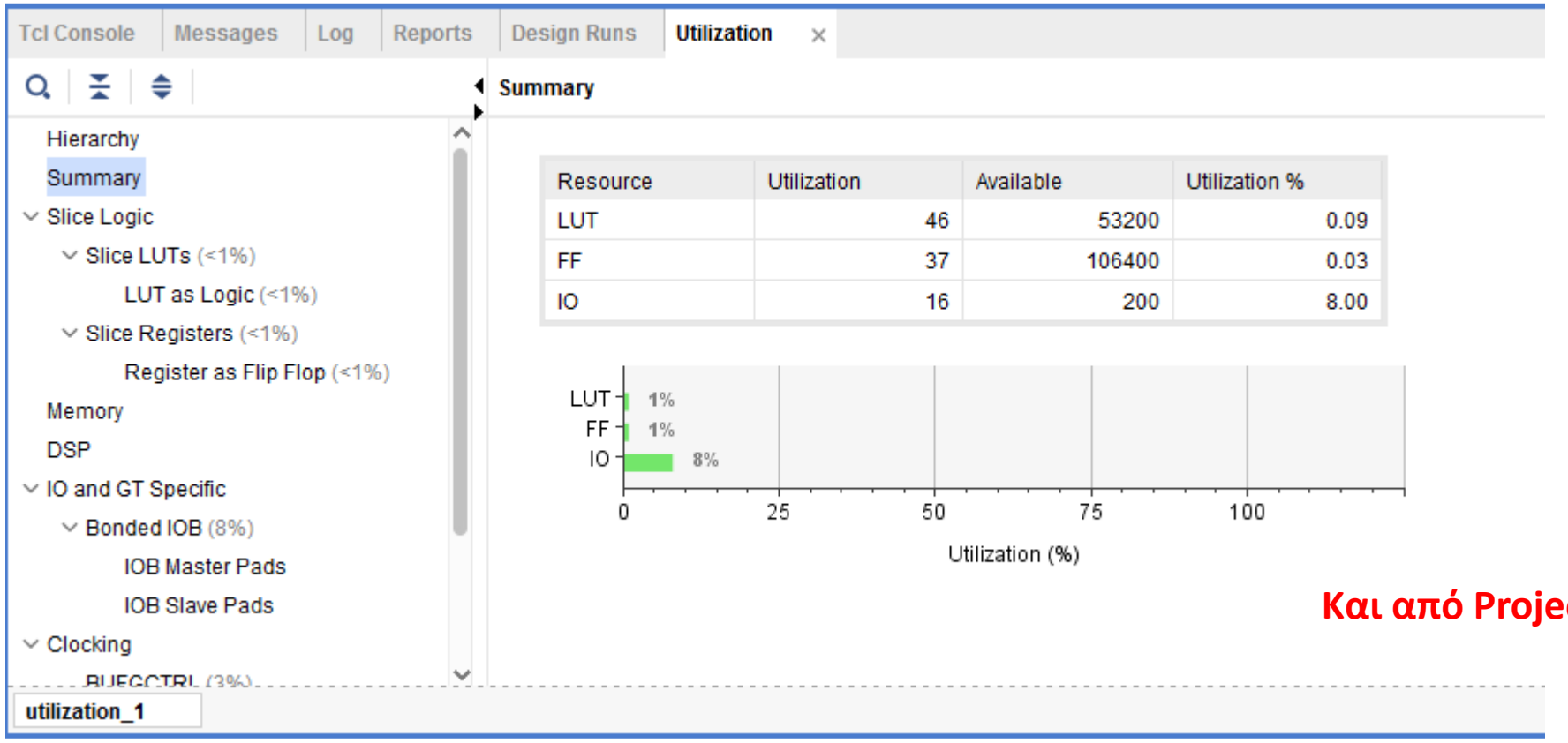
4η Εργαστηριακή Άσκηση

Synthesis/Implementation – Schematic



4η Εργαστηριακή Άσκηση

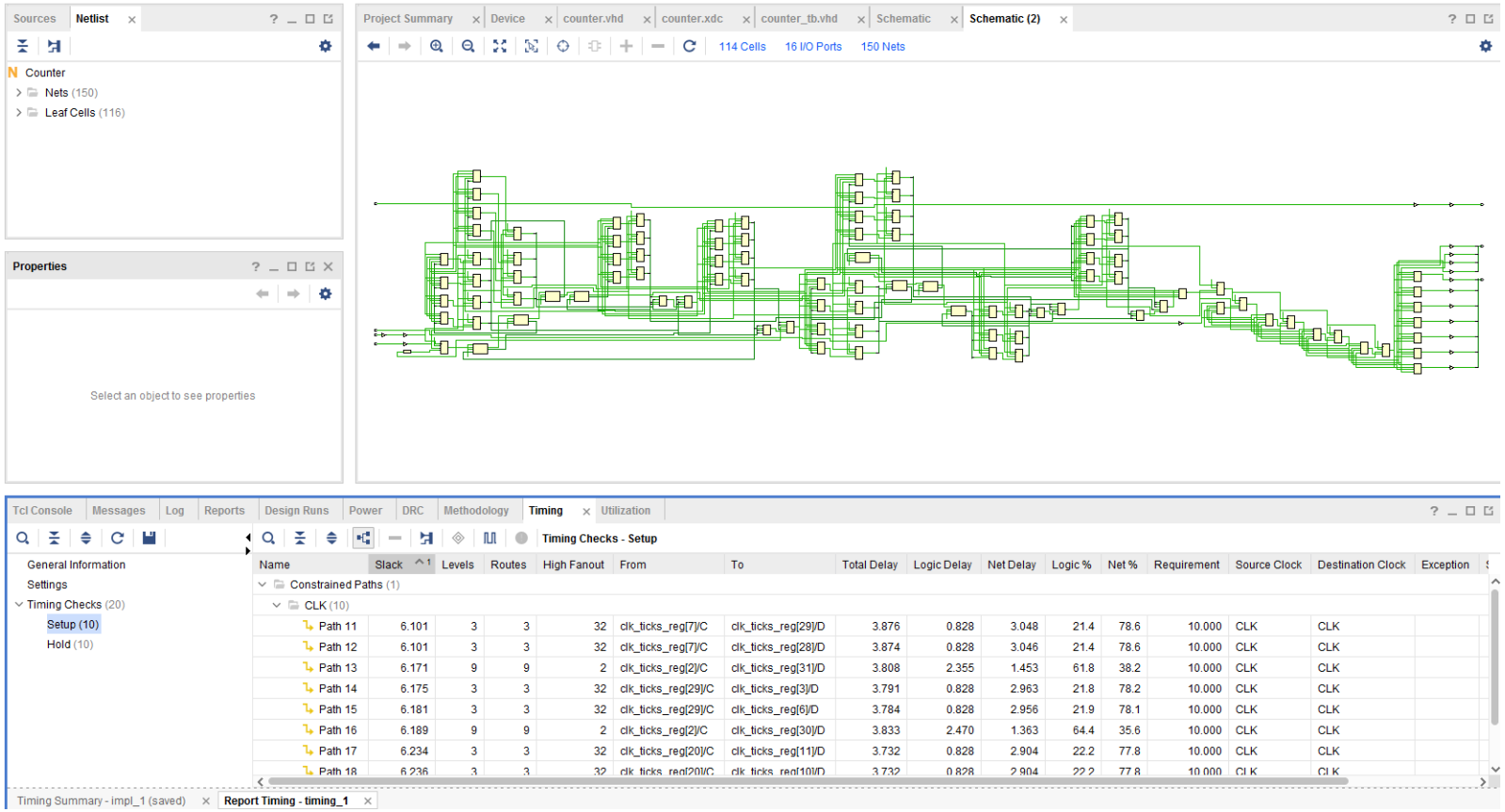
Implementation – Report Utilization



Και από Project Summary

4η Εργαστηριακή Άσκηση

Implementation – Timing Reports (1/5)

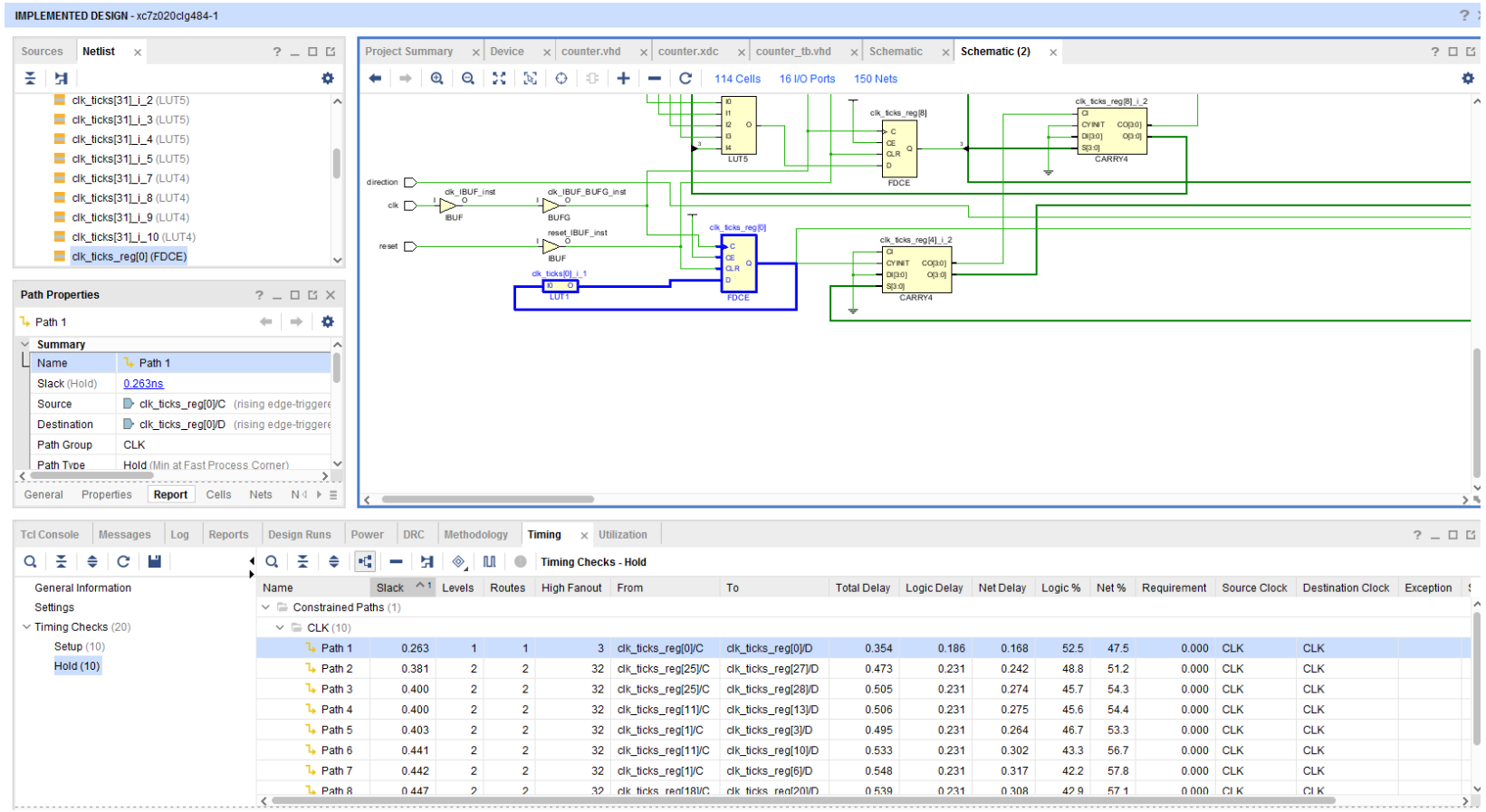


Menu Reports->Timing->Timing Reports

Setup Time:
Αναφέρεται στις αργές διαδρομές (καθυστέρηση διάδοσης)

4η Εργαστηριακή Άσκηση

Implementation – Timing Reports (2/5)

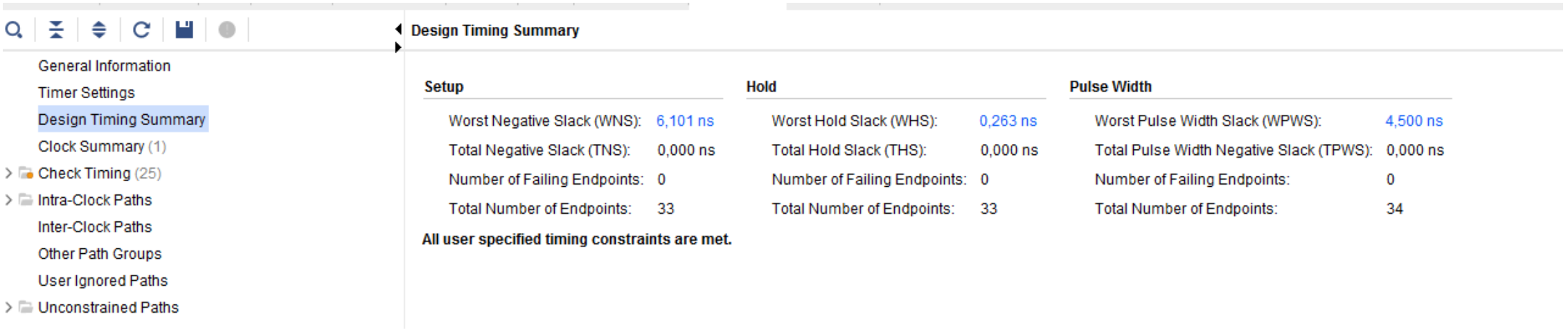


Hold Time:
Αναφέρεται στις γρήγορες διαδρομές (καθυστέρηση μόλυνσης)

4η Εργαστηριακή Άσκηση

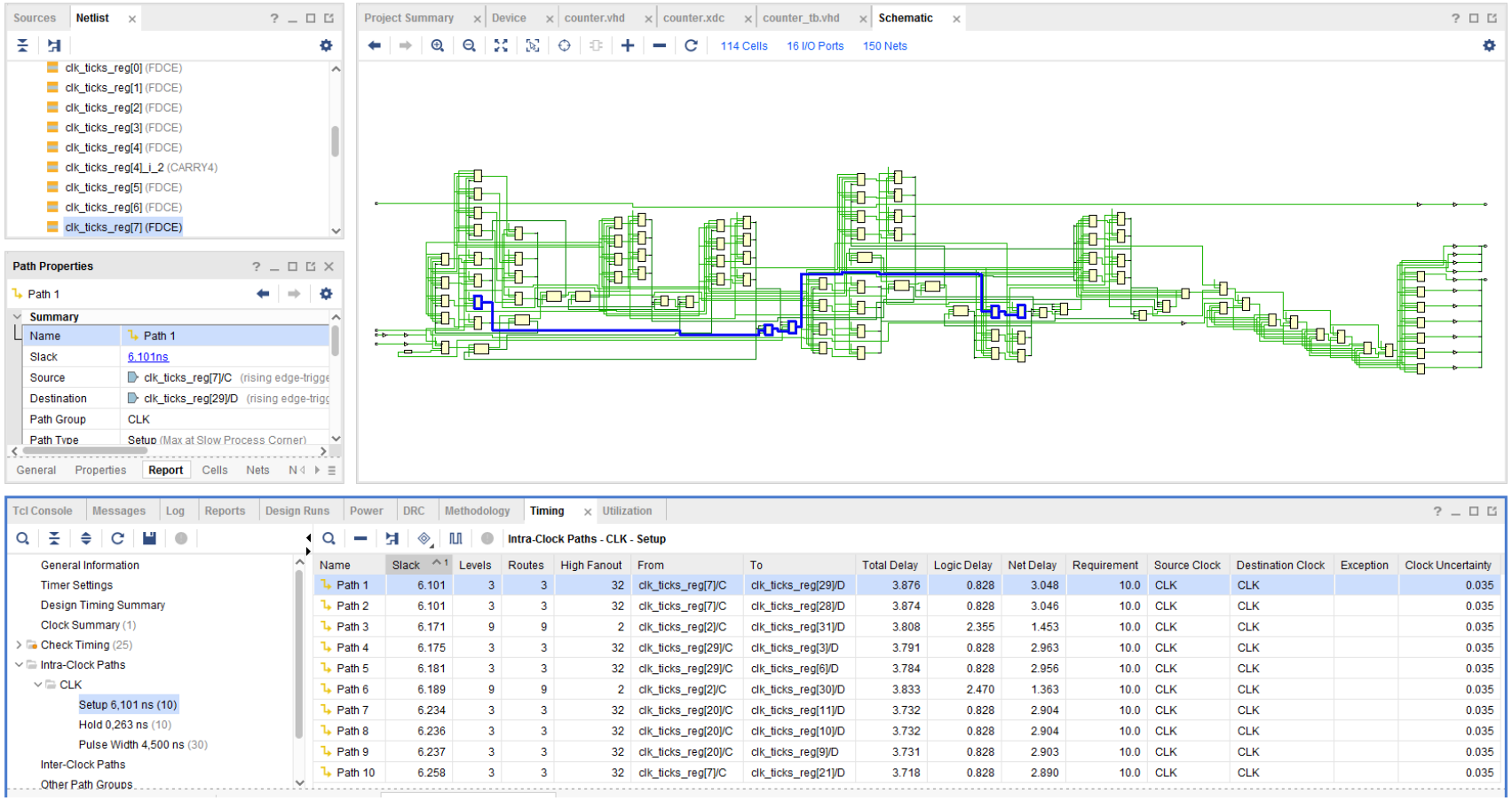
Implementation – Timing Reports (3/5)

Menu Reports->Timing-> Report Timing Summary



4η Εργαστηριακή Άσκηση

Implementation – Timing Reports (4/5)

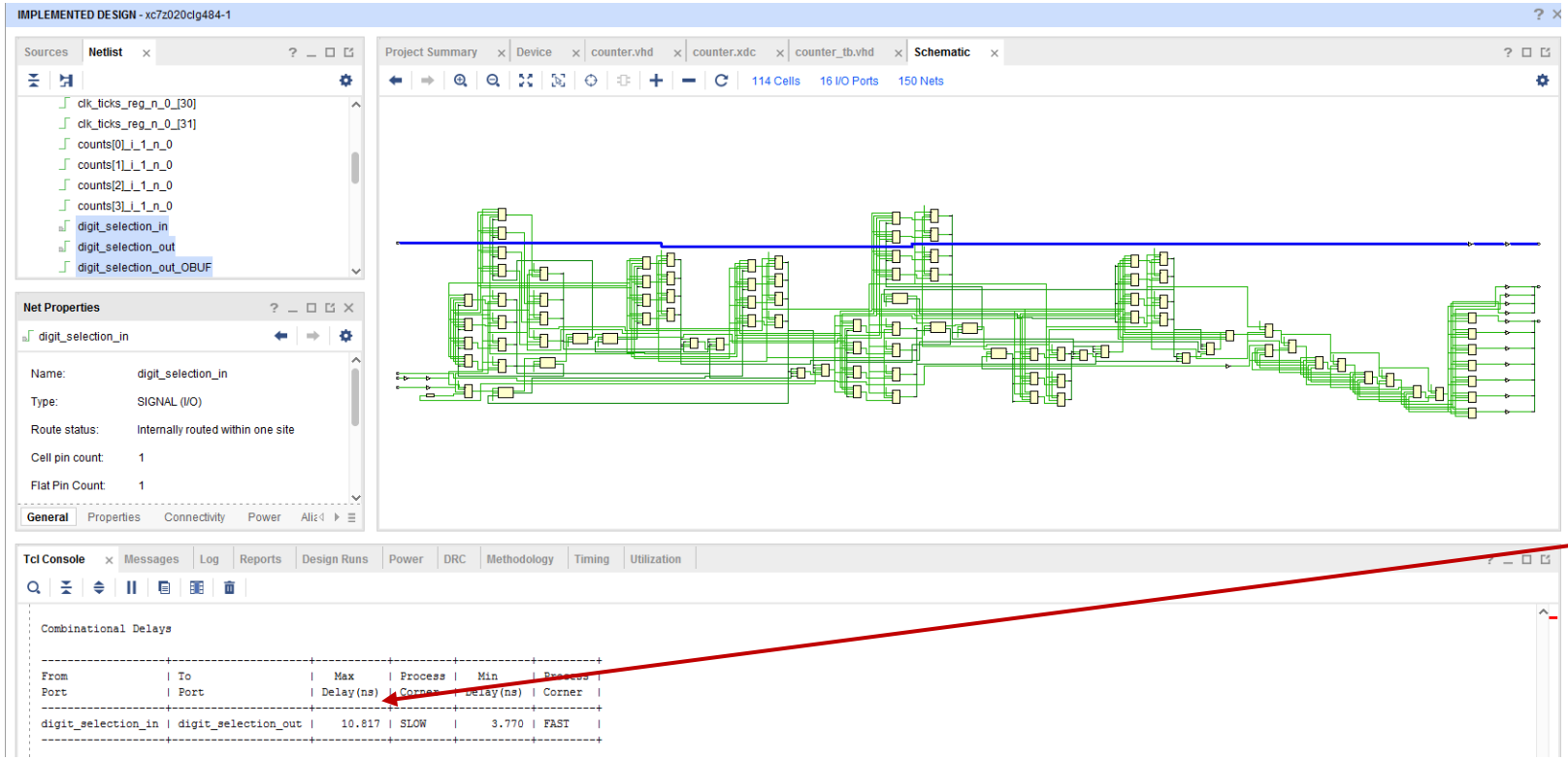


Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	6.101	3	3	32	clk_ticks_reg[7]/C	clk_ticks_reg[29]/D	3.876	0.828	3.048	10.0	CLK	CLK		0.035
Path 2	6.101	3	3	32	clk_ticks_reg[7]/C	clk_ticks_reg[28]/D	3.874	0.828	3.046	10.0	CLK	CLK		0.035
Path 3	6.171	9	9	2	clk_ticks_reg[2]/C	clk_ticks_reg[31]/D	3.808	2.355	1.453	10.0	CLK	CLK		0.035
Path 4	6.175	3	3	32	clk_ticks_reg[29]/C	clk_ticks_reg[3]/D	3.791	0.828	2.963	10.0	CLK	CLK		0.035
Path 5	6.181	3	3	32	clk_ticks_reg[29]/C	clk_ticks_reg[6]/D	3.784	0.828	2.956	10.0	CLK	CLK		0.035
Path 6	6.189	9	9	2	clk_ticks_reg[2]/C	clk_ticks_reg[30]/D	3.833	2.470	1.363	10.0	CLK	CLK		0.035
Path 7	6.234	3	3	32	clk_ticks_reg[20]/C	clk_ticks_reg[11]/D	3.732	0.828	2.904	10.0	CLK	CLK		0.035
Path 8	6.236	3	3	32	clk_ticks_reg[20]/C	clk_ticks_reg[10]/D	3.732	0.828	2.904	10.0	CLK	CLK		0.035
Path 9	6.237	3	3	32	clk_ticks_reg[20]/C	clk_ticks_reg[9]/D	3.731	0.828	2.903	10.0	CLK	CLK		0.035
Path 10	6.258	3	3	32	clk_ticks_reg[7]/C	clk_ticks_reg[21]/D	3.718	0.828	2.890	10.0	CLK	CLK		0.035

4η Εργαστηριακή Άσκηση

Implementation – Timing Reports (5/5)

TCL Console>*report_timing_summary –datasheet*



Προσοχή