Κεφάλαιο 4

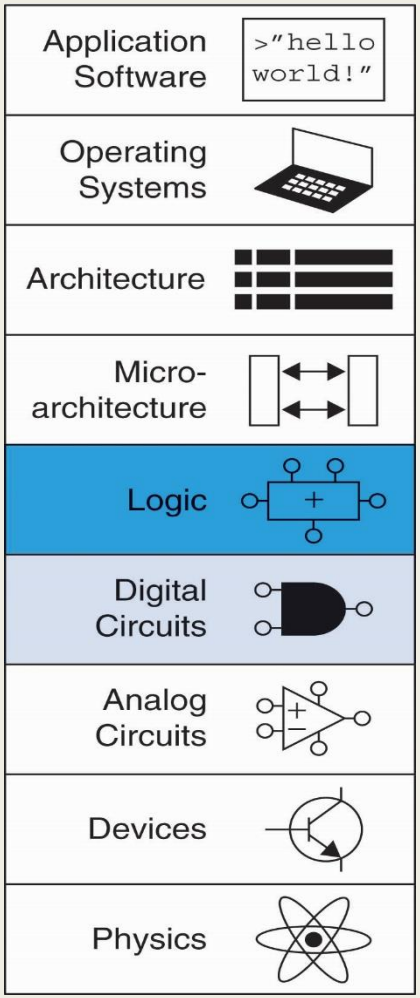
Γλώσσες περιγραφής υλικού

Αντώνης Πασχάλης, Νεκτάριος Κρανίτης

Πλήρης έκδοση

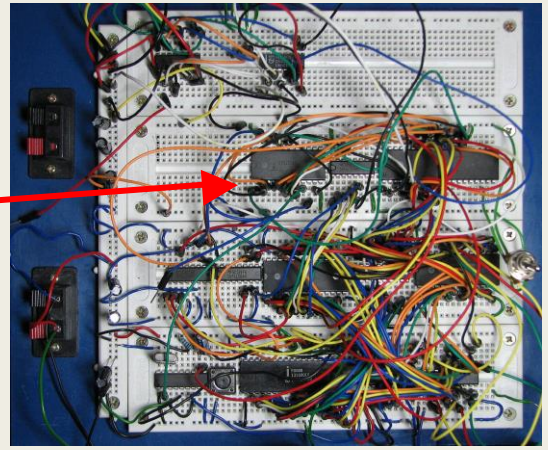
Περιεχόμενα κεφαλαίου 4

- Εισαγωγή
- Τεχνολογία FPGA
- Φάσεις σχεδίασης υλικού
- Η γλώσσα VHDL για σωστή σύνθεση και προσομοίωση
- Πλήρης λίστα κωδίκων (ιδιωματισμών) για υλοποίηση:
  - *συνδυαστικής λογικής*
  - *ακολουθιακής λογική*
    - ασύγχρονη
    - σύγχρονη
  - *μνημών RAM, ROM*
  - *στο επίπεδο RTL*
- Προγράμματα δοκιμής

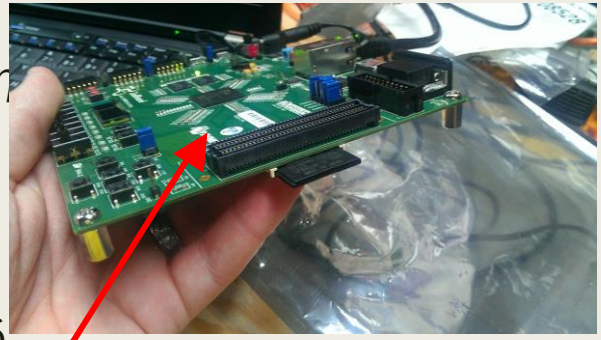


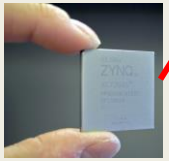
Η εξέλιξη της ψηφιακής σχεδίασης στο χρόνο…

- Παλαιότερα: Χρήση bread-boards
  - *Καλωδίωση κυκλωμάτων SSI-MSI (DIP) με το χέρι*
  - *Περιορισμός σε λογικές πύλες (≈100)*
  - *Χωρίς χρήση εργαλείων λογισμικού*
  - *Χρήση παλμογράφου για verification και debug*



- Σήμερα: Χρήση development boards με FPGAs
  - *Μεγάλη πυκνότητα ολοκλήρωσης (≈100K Logic Cells)*
  - *Σχεδίαση με γλώσσα περιγραφής υλικού*
  - *Χρήση εργαλείων λογισμικού για λογική σύνθεση (synthesis) και υλοποίηση (implementation) για συγκεκριμένο FPGA*
  - *Verification με εργαλεία προσομοίωσης*
  - *Debug με χρήση ειδικών κυκλωμάτων στο υλικό (on-chip debug)*





Τα ψηφιακά συστήματα αλλάζουν!

Μικρά

Μικρά + μεγάλα

Μικρά + μεγάλα και ετερογενή !

Τα ψηφιακά συστήματα είναι επαναδιαμορφώσιμα

Η επαναδιαμόρφωση (reconfiguration) είναι παντού στην καθημερινότητά μας!

Full+static

Partial+static

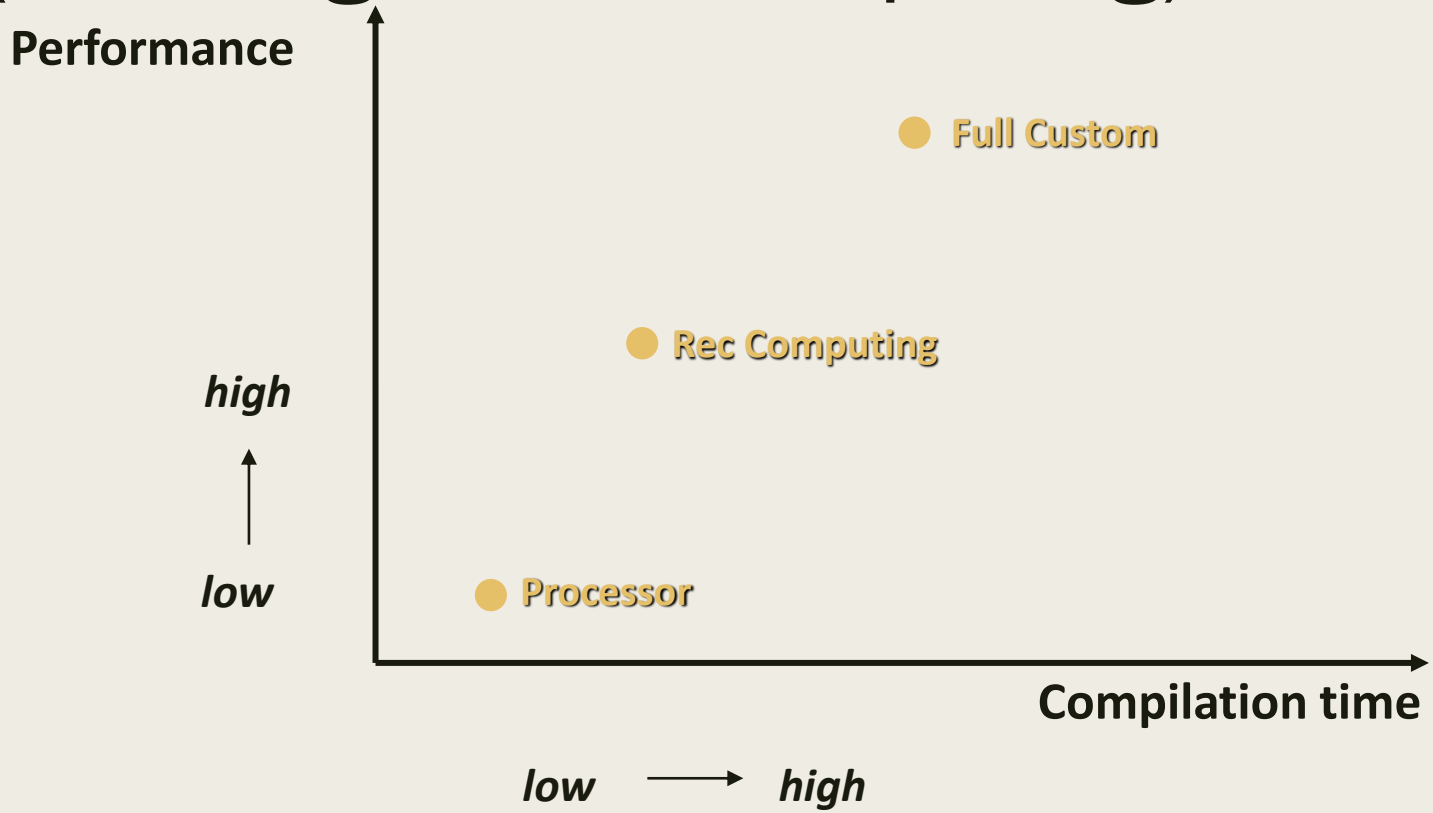
Partial+dynamic

Επαναδιαμόρφωση (reconfiguration)

The process of physically altering the location or functionality of network or system elements. Automatic configuration describes the way sophisticated networks can readjust themselves in the event of a link or device failing, enabling the network to continue operation.

Gerald Estrin, 1960

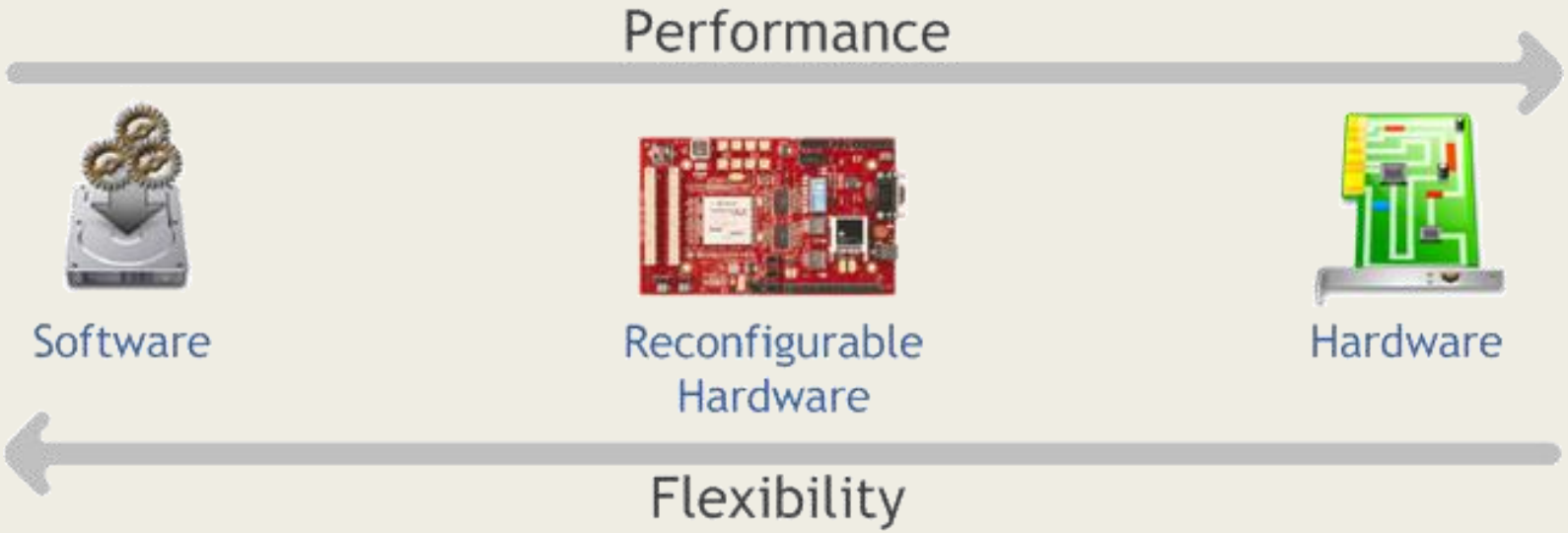
Επαναδιαμορφώσιμη υπολογιστική (reconfigurable computing)



Reconfigurable computing is defined as the study of computation using reconfigurable devices

Christophe Bobda, 2007

Επαναδιαμορφώσιμα συστήματα υλικού



"Reconfigurable computing is intended to fill the gap between hardware and software, achieving potentially much higher performance than software, while maintaining a higher level of flexibility than hardware"

(K. Compton and S. Hauck, *Reconfigurable Computing: a Survey of Systems and Software*, 2002)

Επαναδιαμορφώσιμα συστήματα στο διάστημα

- Adaptable instrument
  - *Προσαρμόζεται σε απρόσμενα συμβάντα*
  - *Αλλαγή ή επέκταση των επιστημονικών στόχων*
  - *Αυξημένο yield στα επιστημονικά δεδομένα*
  - *Μειωμένος κίνδυνος για ολική απώλεια του οργάνου*

- Dynamic Adaptability
  - *Διαστημική τεχνολογία αιχμής*
  - *Time-Space Partitioning*
    - Όταν δεν χρειάζονται ταυτόχρονα όλες οι λειτουργίες
  - *Σημαντικά οφέλη στο size, weight, power and cost (SWaP-C)*

Προγραμματιζόμενες από τον χρήστη διατάξεις πυλών (FPGA)

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

Η αρχιτεκτονική των FPGAs της XILINX

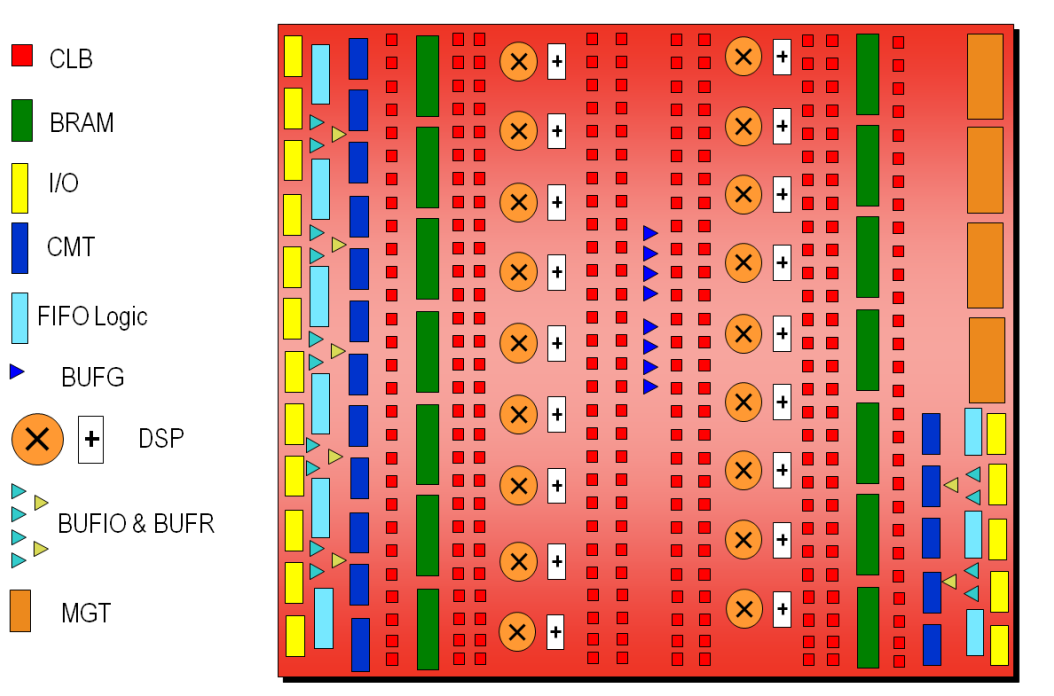
- **Configurable Logic Blocks (CLBs)**
  που περιέχουν πίνακες αναζήτησης (LUT), που υλοποιούν συνδυαστική λογική, και στοιχεία αποθήκευσης που μπορούν να χρησιμοποιηθούν ως D Flip-flops ή Latches
- **Input/Output Blocks (IOBs)**
  που ελέγχουν την ροή δεδομένων μεταξύ των I/O pins και της εσωτερικής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

Οικογένειες 7-Series FPGA + SoC της Xilinx

Maximum Capability	ARTIX-7 Lowest Power and Cost	KINTEX-7 Industry's Best Price/Performance	VIRTEX-7 Industry's Highest Performance	ZYNQ All Programmable SoC
Logic Cells in K	33 – 215	66 – 478	583 – 1,139	28 – 444
Block RAM in Mb	2 – 12	4 – 34	28 – 68	2 - 27
DSP Slices	90 – 740	240 – 1,920	1,260 – 3,360	80–2,020
Peak DSP Perf. (GMACs)	929	2,845	5,335	2,622
Transceivers	Up to 16	Up to 32	Up to 88	Up to 16
Transceiver Perf. (Gbps)	6.6	12.5	12.5, 13.1 and 28	6.6, 12.5
Memory Perf. (Mbps)	1066	1866	1866	1333
User I/O Pins	106 – 500	285 – 500	350 – 1,100	54 – 400
I/O Voltages	3.3V and below	3.3V and below 1.8V and below	3.3V and below 1.8V and below	3.3V and below 1.8V and below

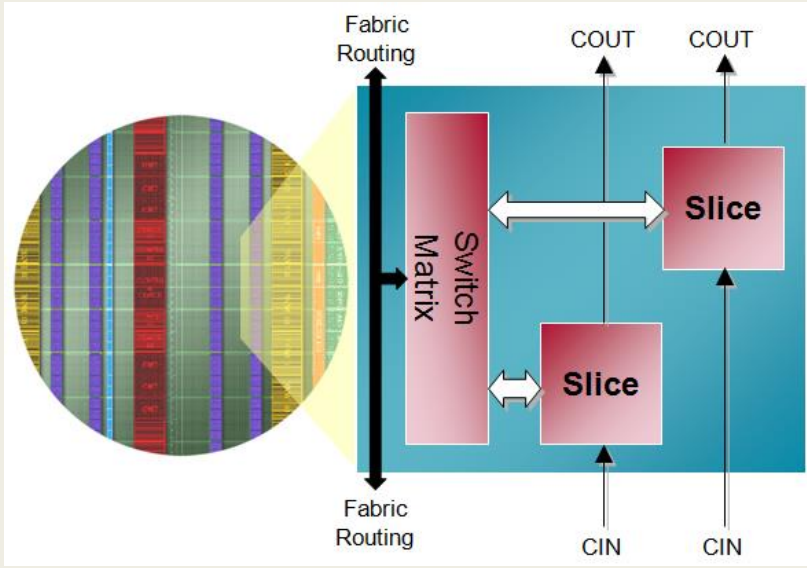
Αρχιτεκτονική 7-Series της Xilinx

- Η χρήση κοινών στοιχείων επιτρέπει την επαναχρησιμοποίηση IP για όλες τις οικογένειες της 7-Series της Xilinx
  - *Κλιμάκωση της σχεδίασης από χαμηλού κόστους έως υψηλής απόδοσης*
  - *Εκτεταμένη υποστήριξη οικοσυστήματος*
  - *Ταχύτατο time-to-market*



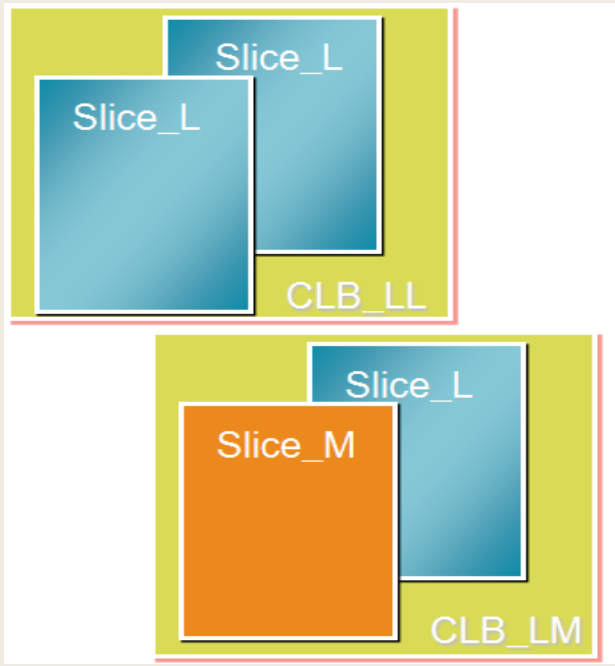
Configurable Logic Block (CLB) in 7-Series FPGAs

- Βασικός πόρος για τη σχεδίαση με Xilinx FPGAs
  - *Συνδυαστική λογική*
  - *Flip-flops*
- Το CLB περιέχει δύο slices
- Συνδέονται με ένα switch matrix για το routing σε άλλους πόρους του FPGA
  - *Η αλυσίδα κρατουμένου (carry chain) διατρέχει κάθετα το CLB σε μια στήλη από το ένα slice στο από πάνω του*

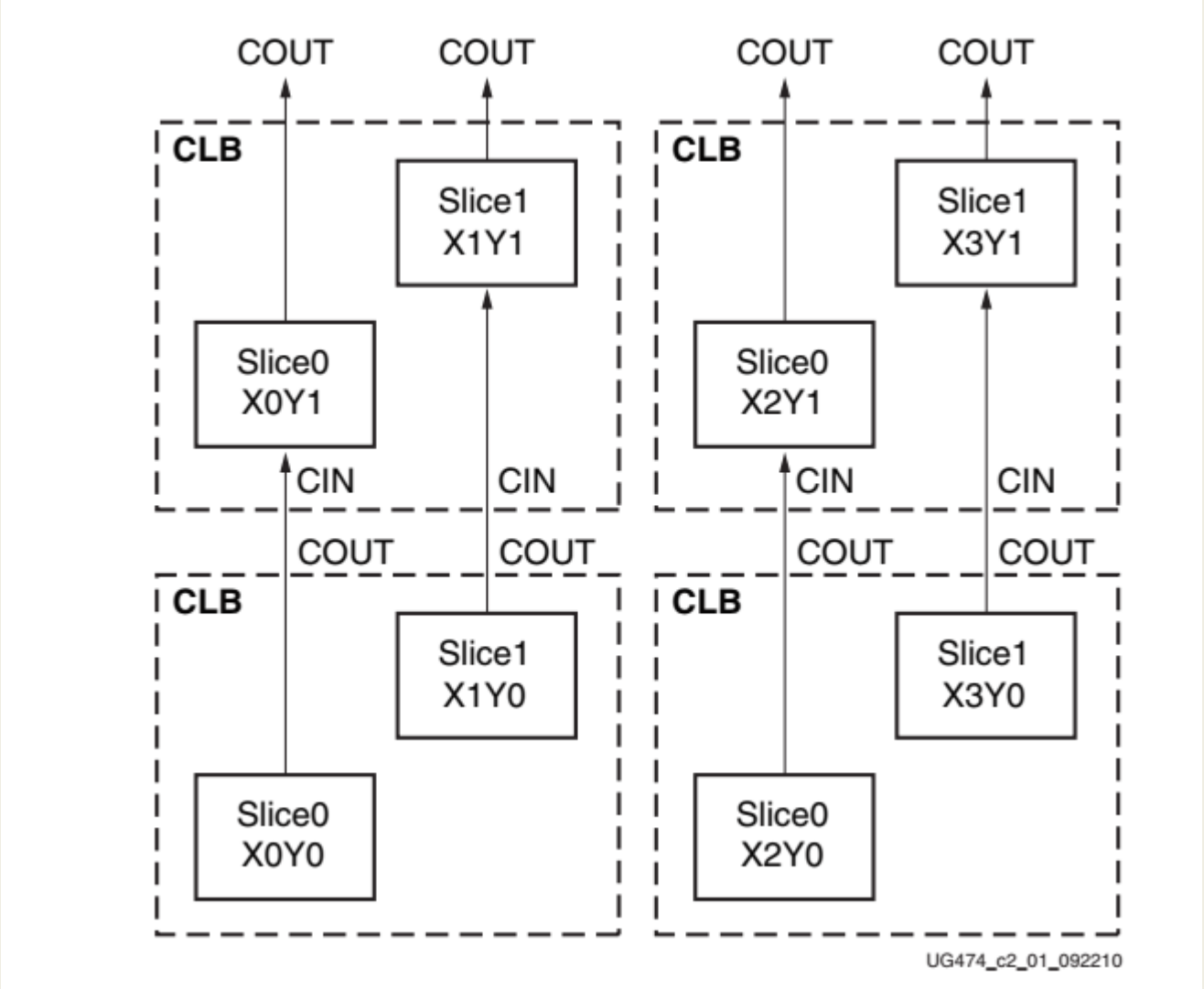


Δύο τύποι CLB Slices

- Δύο τύποι CLB Slices
  - *SLICEM: Πλήρες slice*
    - Το LUT μπορεί να χρησιμοποιηθεί για λογική και μνήμη / SRL
    - Διαθέτει πολυπλέκτες μεγάλου εύρους και αλυσίδα κρατουμένου
    - Το 1/3 του συνόλου των slices
  - *SLICEL: Λογική και αριθμητική μόνο*
    - Το LUT μπορεί να χρησιμοποιηθεί **μόνο για λογική** (όχι μνήμη)
    - Διαθέτει πολυπλέκτες μεγάλου εύρους και αλυσίδα κρατουμένου
    - Τα 2/3 του συνόλου των slices



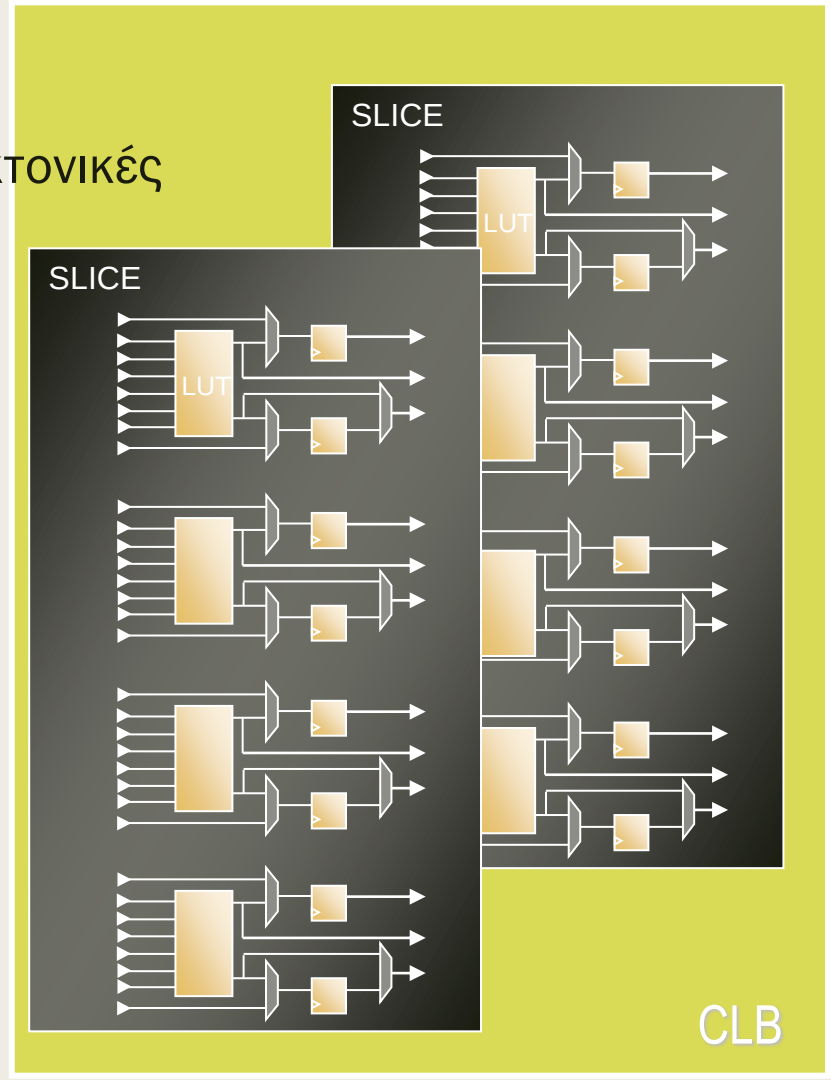
Διάταξη CLB Slices



Το σχήμα δείχνει 4 CLBs που βρίσκονται στην κάτω – αριστερή γωνία του die

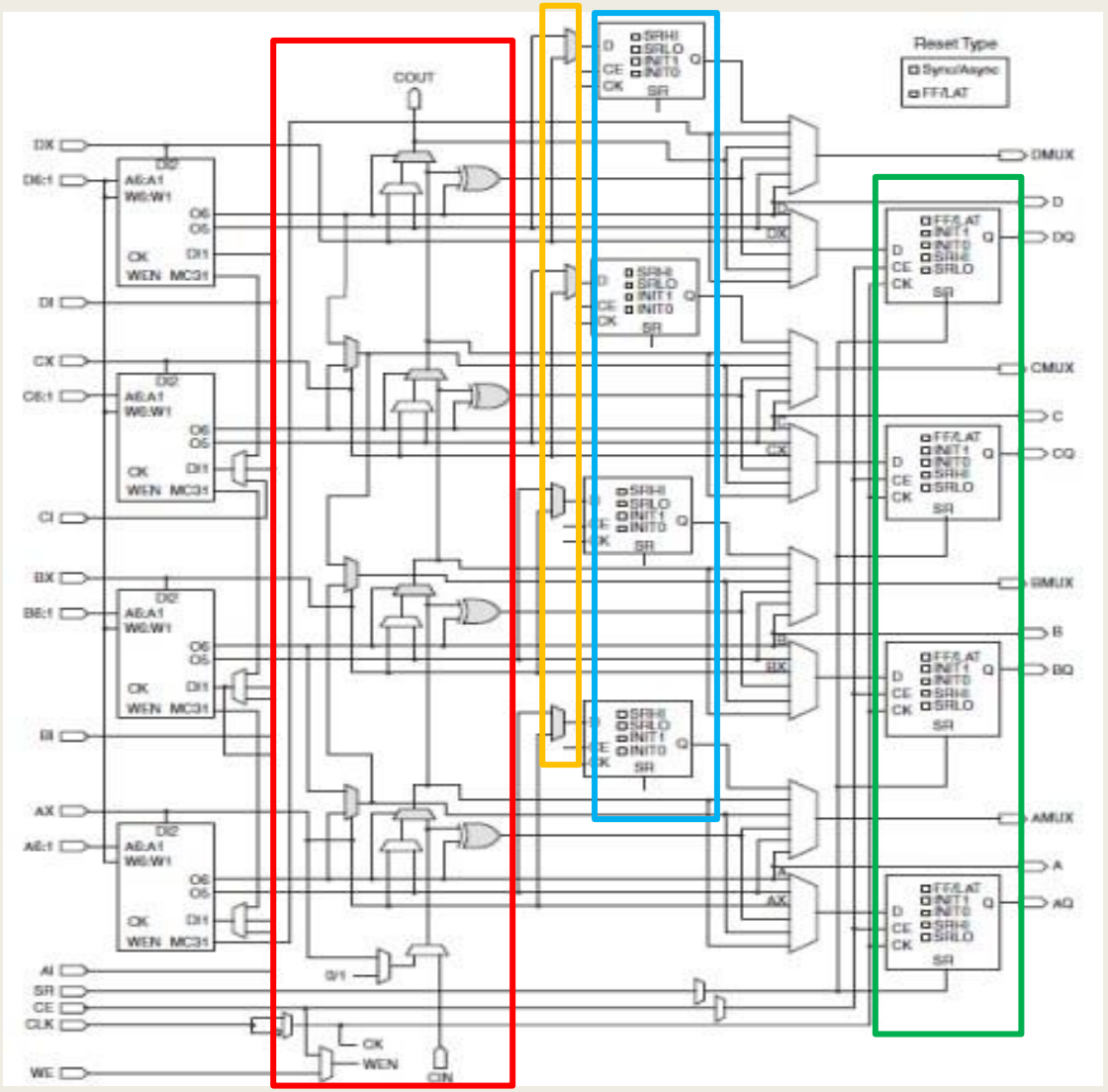
Πόροι ενός Slice

- Τέσσερα 6-input LUTs ανά slice
  - Συμβατό με προηγούμενες αρχιτεκτονικές
- Δύο flip-flops ανά LUT
  - Εξαιρετικό για σχέδια με βαθύ pipeline

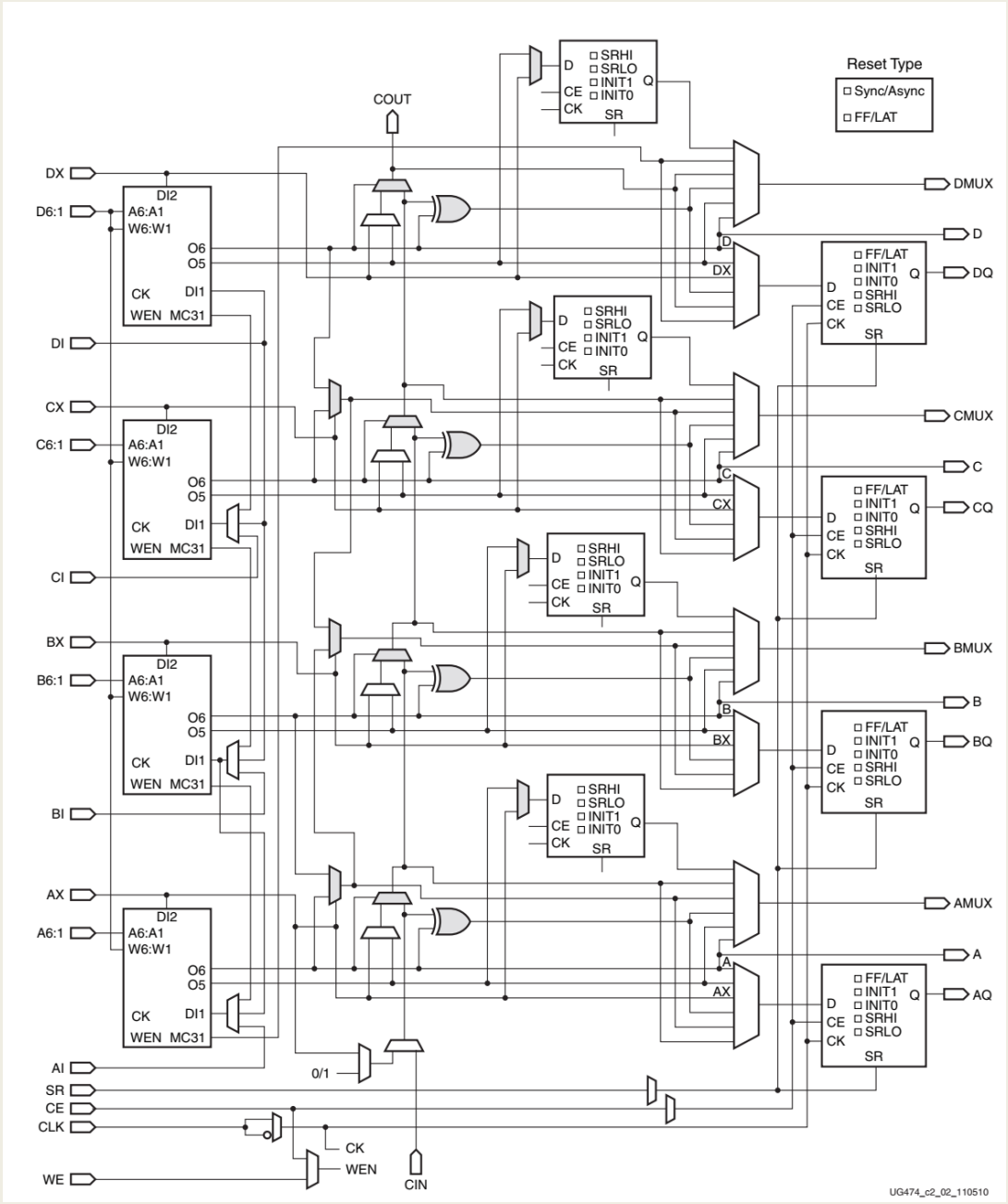


Πόροι ενός Slice

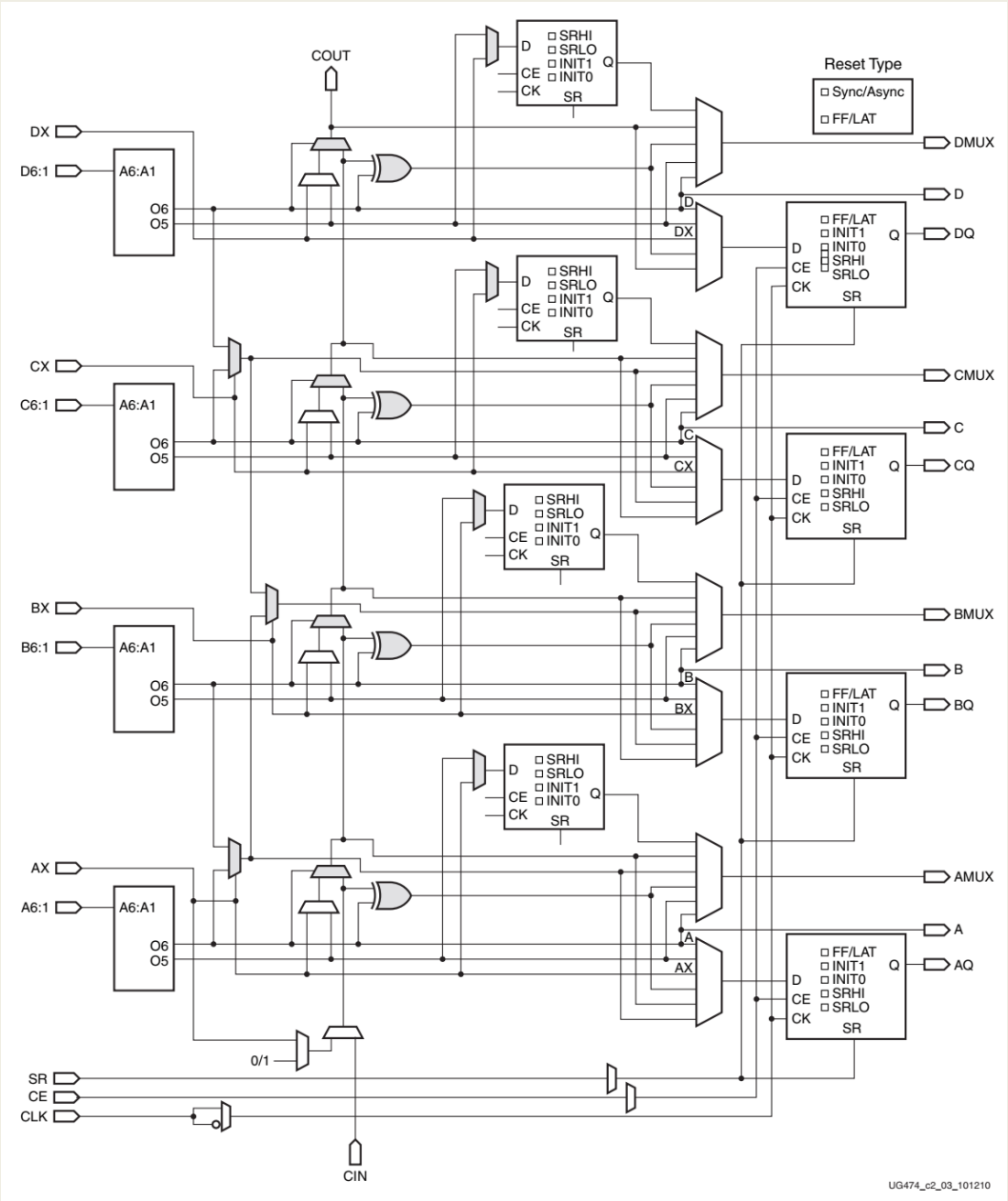
- Τέσσερα Look-Up Tables (LUT) των 6 εισόδων
- Πολυπλέκτες
- Αλυσίδες κρατουμένου (carry chains)
- Καταχωρητές Ολίσθησης (SRL)
- Τέσσερα flip-flops/latches
  - *Τέσσερα επιπλέον FFs*
- Το εργαλείο υλοποίησης θα στοιβάξει πολλαπλά slices στο ίδιο CLB εάν πληρούνται κάποιοι κανόνες



Διάγραμμα ενός SLICEM

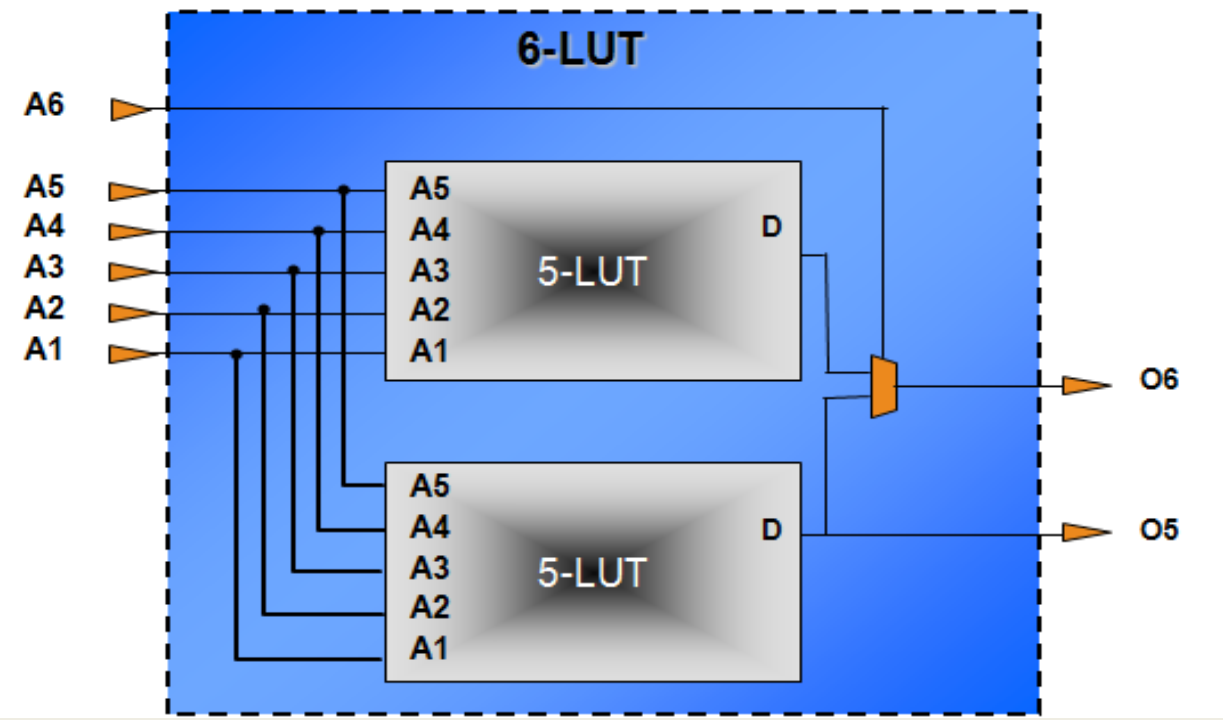


Διάγραμμα ενός SLICEL



LUT 6-εισόδων με Διπλή Έξοδο

- Τα LUTs μπορούν να είναι δύο LUTs 5-εισόδων με κοινή είσοδο
  - *Ελάχιστη επιβάρυνση στην ταχύτητα*
  - *Μία ή δύο έξοδοι*
- Οποιαδήποτε λογική συνάρτηση των 6 μεταβλητών ή δύο συναρτήσεις των 5 μεταβλητών



LUT 6-εισόδων με Διπλή Έξοδο

The function generators in 7 series FPGAs are implemented as six-input look-up tables (LUTs). There are six independent inputs (A inputs - A1 to A6) and two independent outputs (O5 and O6) for each of the four function generators in a slice (A, B, C, and D). The function generators can implement:

- Any arbitrarily defined six-input Boolean function
- Two arbitrarily defined five-input Boolean functions, as long as these two functions share common inputs
- Two arbitrarily defined Boolean functions of 3 and 2 inputs or less

A six-input function uses:

- A1-A6 inputs
- O6 output

Two five-input or less functions use:

- A1–A5 inputs
- A6 driven High
- O5 and O6 outputs

The propagation delay through a LUT is independent of the function implemented. Signals from the function generators can:

- Exit the slice (through A, B, C, D output for O6 or AMUX, BMUX, CMUX, DMUX output for O5)
- Enter the XOR dedicated gate from an O6 output
- Enter the carry-logic chain from an O5 output
- Enter the select line of the carry-logic multiplexer from O6 output
- Feed the D input of the storage element
- Go to F7AMUX/F7BMUX wide multiplexers from O6 output

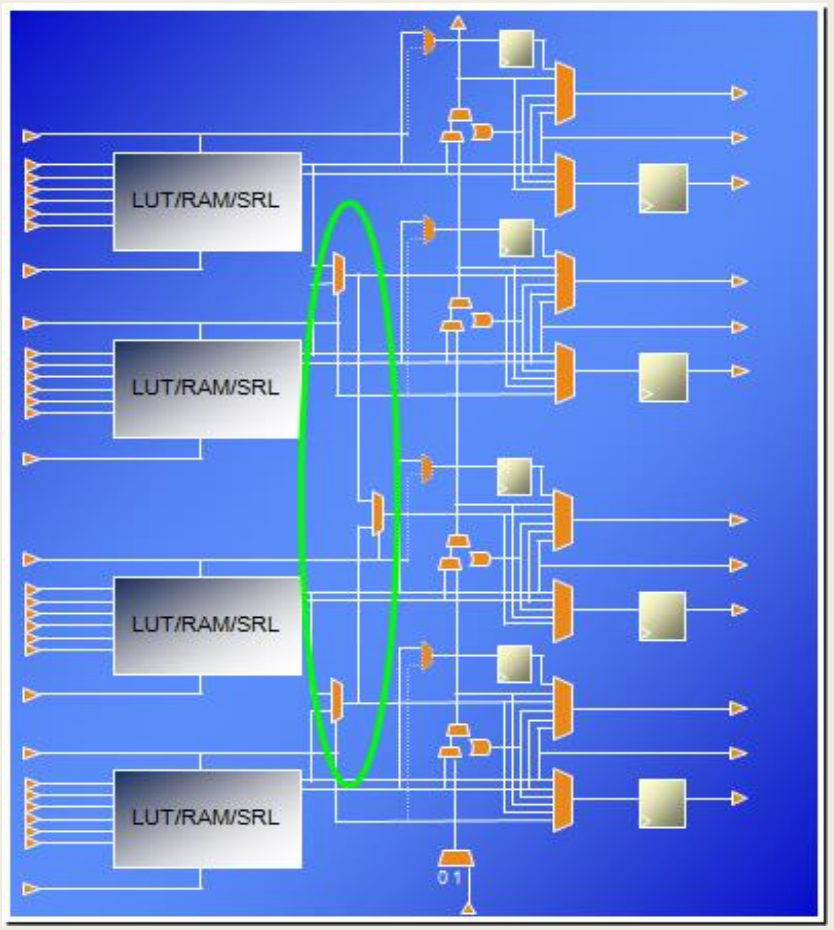
In addition to the basic LUTs, slices contain three multiplexers (F7AMUX, F7BMUX, and F8MUX). These multiplexers are used to combine up to four function generators to provide any function of seven or eight inputs in a slice.

- F7AMUX: Used to generate seven input functions from LUTs A and B
- F7BMUX: Used to generate seven input functions from LUTs C and D
- F8MUX: Used to combine all LUTs to generate eight input functions.

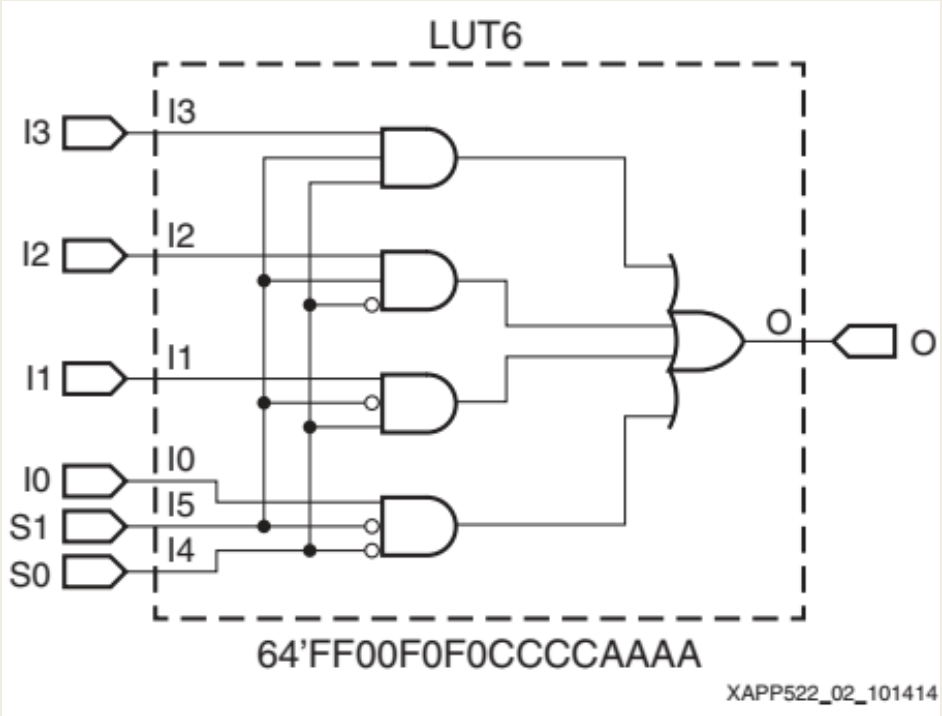
Functions with more than eight inputs can be implemented using multiple slices. There are no direct connections between slices to form function generators greater than eight inputs within a CLB.

Πολυπλέκτες μεγάλου εύρους

- Κάθε F7MUX συνδυάζει τις εξόδους από δύο LUTs
  - *Υλοποιεί μια τυχαία συνάρτηση 7-εισόδων*
  - *Υλοποιεί έναν πολυπλέκτη 8-1*
- Ο F8MUX συνδυάζει τις εξόδους των δύο F7MUXes
  - *Υλοποιεί μια τυχαία συνάρτηση 8-εισόδων*
  - *Υλοποιεί έναν πολυπλέκτη 16-1*
- Ο MUX ελέγχεται από τις εισόδους BX/CX/DX του slice
- Η έξοδος του MUX μπορεί να οδηγήσει την συνδυαστική έξοδο ή το flip-flop/latch

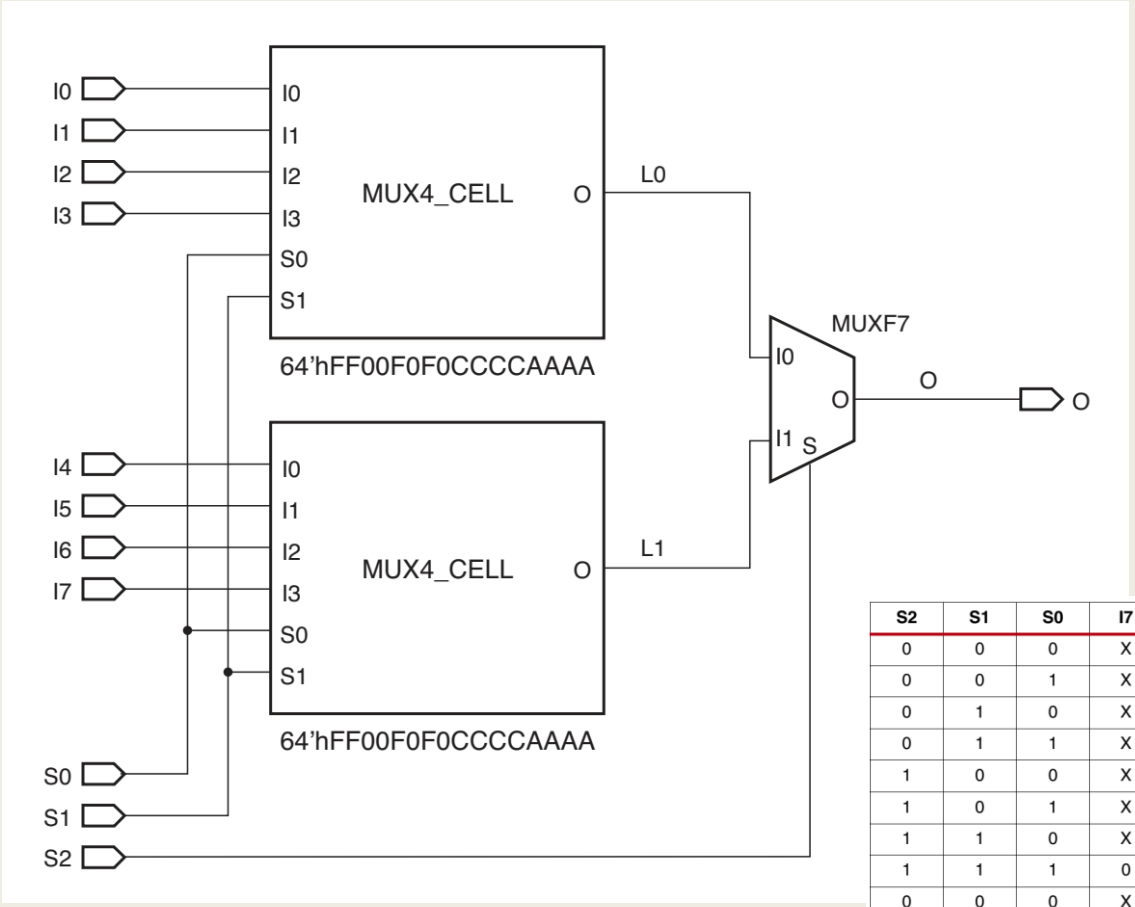


Πολυπλέκτης 4-σε-1



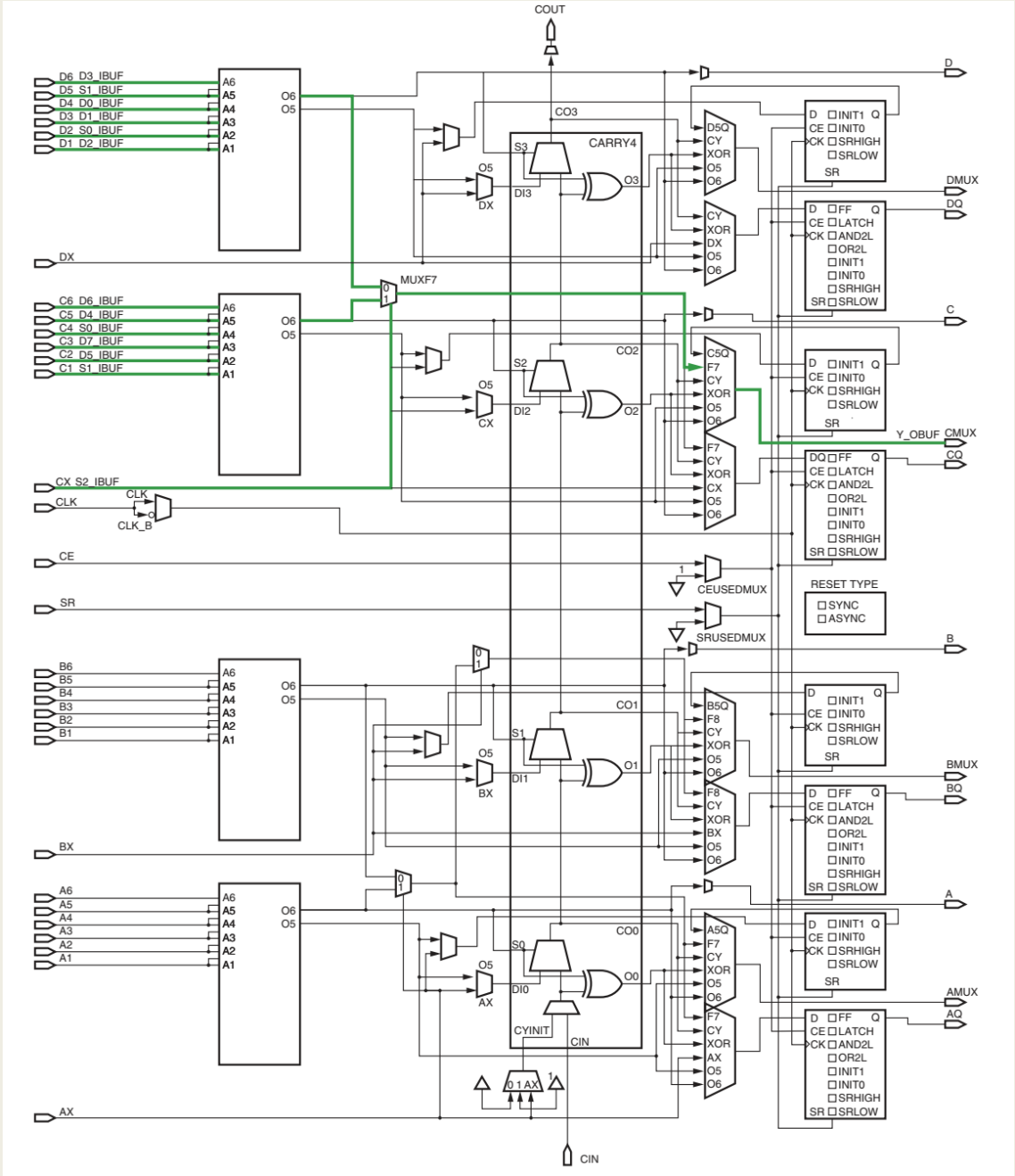
S1	S0	I3	I2	I1	I0	O
0	0	X	X	X	0	0
0	1	X	X	0	X	0
1	0	X	0	X	X	0
1	1	0	X	X	X	0
0	0	X	X	X	1	1
0	1	X	X	1	X	1
1	0	X	1	X	X	1
1	1	1	X	X	X	1

Πολυπλέκτης 8-σε-1

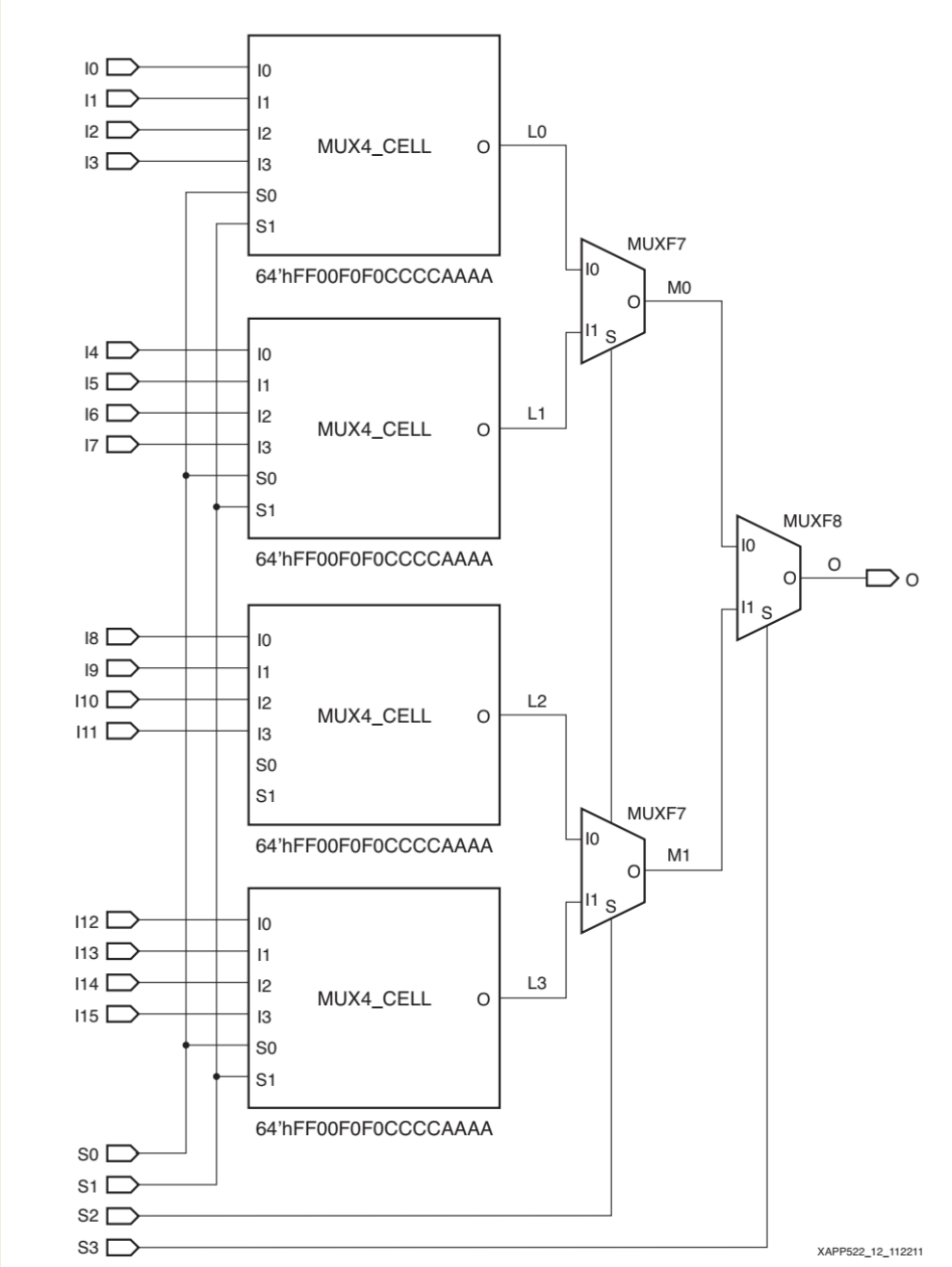


S2	S1	S0	I7	I6	I5	I4	I3	I2	I1	I0	O
0	0	0	X	X	X	X	X	X	X	0	0
0	0	1	X	X	X	X	X	X	0	X	0
0	1	0	X	X	X	X	X	0	X	X	0
0	1	1	X	X	X	X	0	X	X	X	0
1	0	0	X	X	X	0	X	X	X	X	0
1	0	1	X	X	0	X	X	X	X	X	0
1	1	0	X	0	X	X	X	X	X	X	0
1	1	1	0	X	X	X	X	X	X	X	0
0	0	0	X	X	X	X	X	X	X	1	1
0	0	1	X	X	X	X	X	X	1	X	1
0	1	0	X	X	X	X	X	1	X	X	1
0	1	1	X	X	X	X	1	X	X	X	1
1	0	0	X	X	X	1	X	X	X	X	1
1	0	1	X	X	1	X	X	X	X	X	1
1	1	0	X	1	X	X	X	X	X	X	1
1	1	1	1	X	X	X	X	X	X	X	1

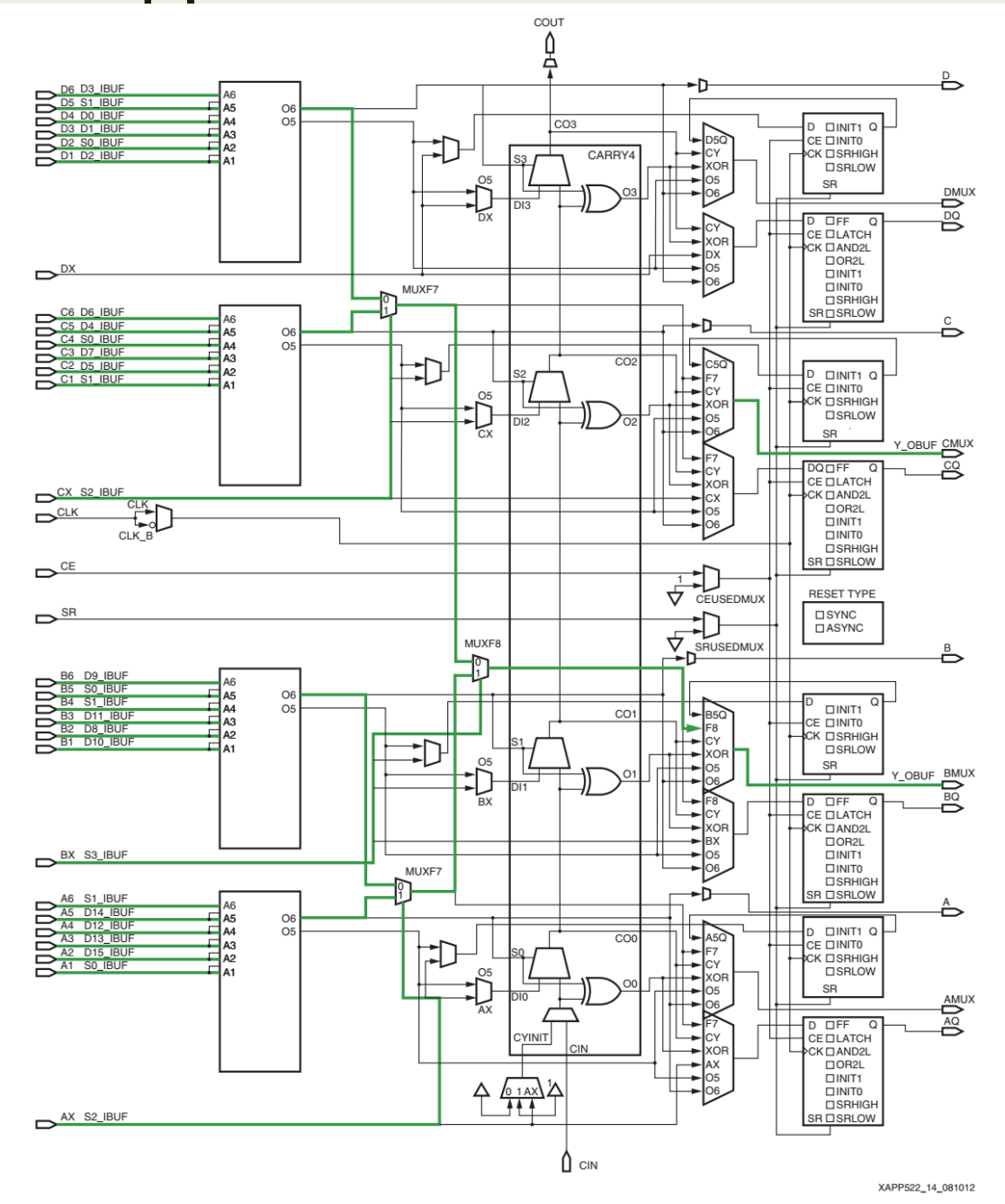
Πολυπλέκτης 8-σε-1



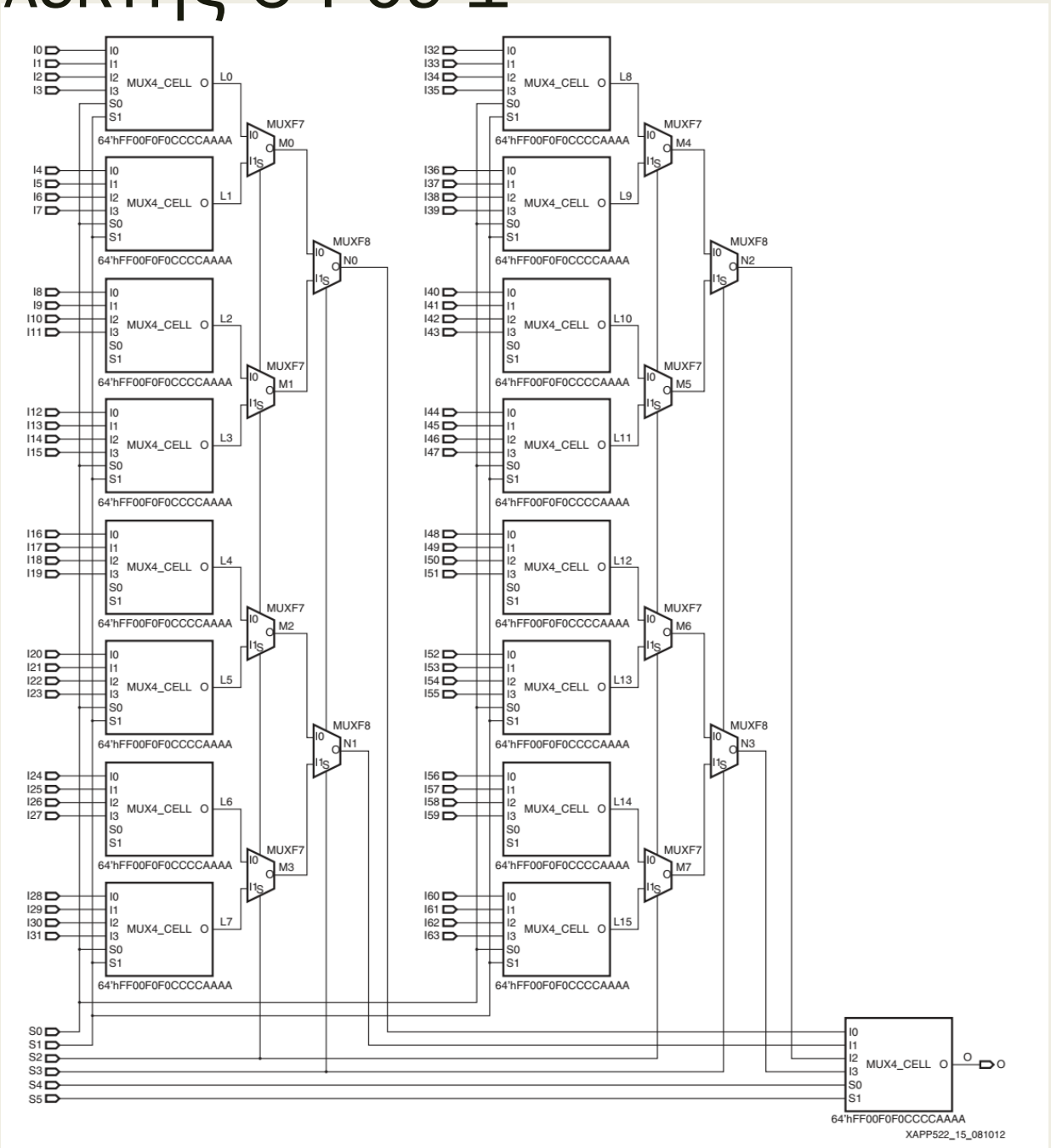
Πολυπλέκτης 16-σε-1



Πολυπλέκτης 16-σε-1



Πολυπλέκτης 64-σε-1



Χρήση soft processor cores σε τεχνολογία FPGA

- Οι **hard processor cores** πλεονεκτούν σε **μέγεθος, απόδοση και κατανάλωση ισχύος** και παρέχουν μία σχετικά φθηνή λύση, αλλά …
  - *Περιορίζουν τον αριθμό των επεξεργαστών, που μπορούν να χρησιμοποιηθούν*
    - Μπορεί να μην καλύπτονται οι συγκεκριμένες ανάγκες μίας εφαρμογής
  - *Δεν προσαρμόζονται σε χαμηλότερες απαιτήσεις για απόδοση, που μπορεί να έχει μία εφαρμογή,*
    - Δεν αλλάζει η πολυπλοκότητά τους
  - *Δημιουργούν προβλήματα στο rooting του FPGA*
    - Είναι τοποθετημένοι σε συγκεκριμένη θέση στο FPGA
  - *Μειώνουν το πλήθος των ορθά παραγόμενων FPGAs*
    - Αυξάνουν το κόστος του FPGA
    - Μειώνουν την πελατειακή βάση

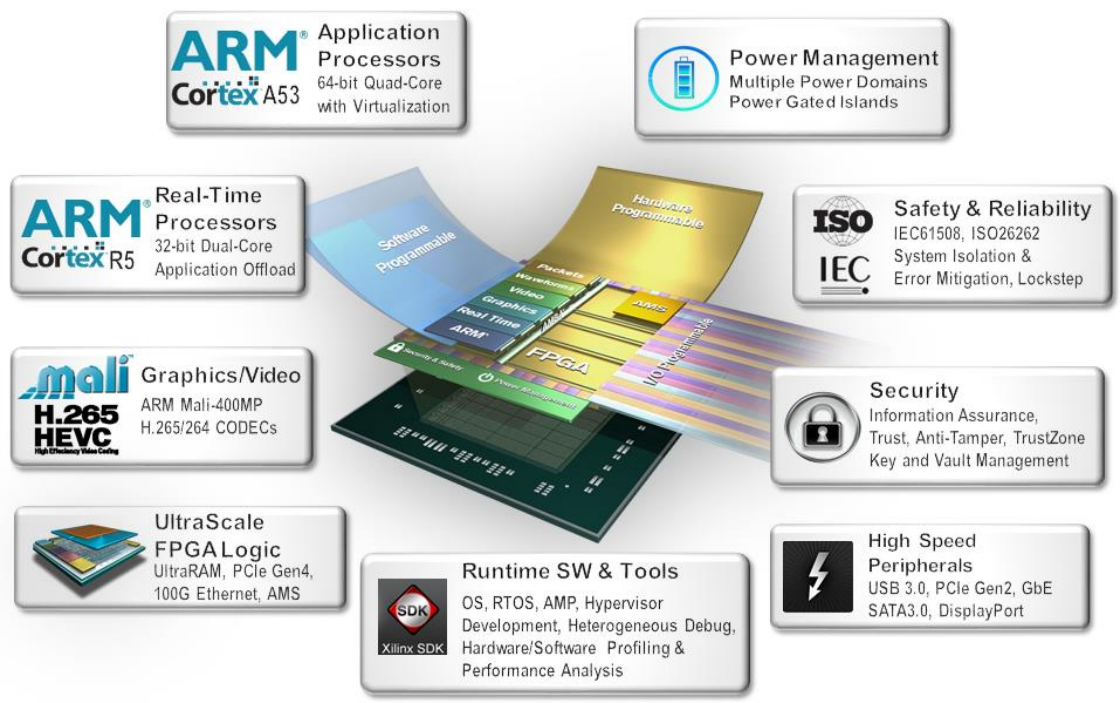
Επιταχυντές υλικού για ψηφιακά συστήματα υψηλής απόδοσης σε τεχνολογία FPGA

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

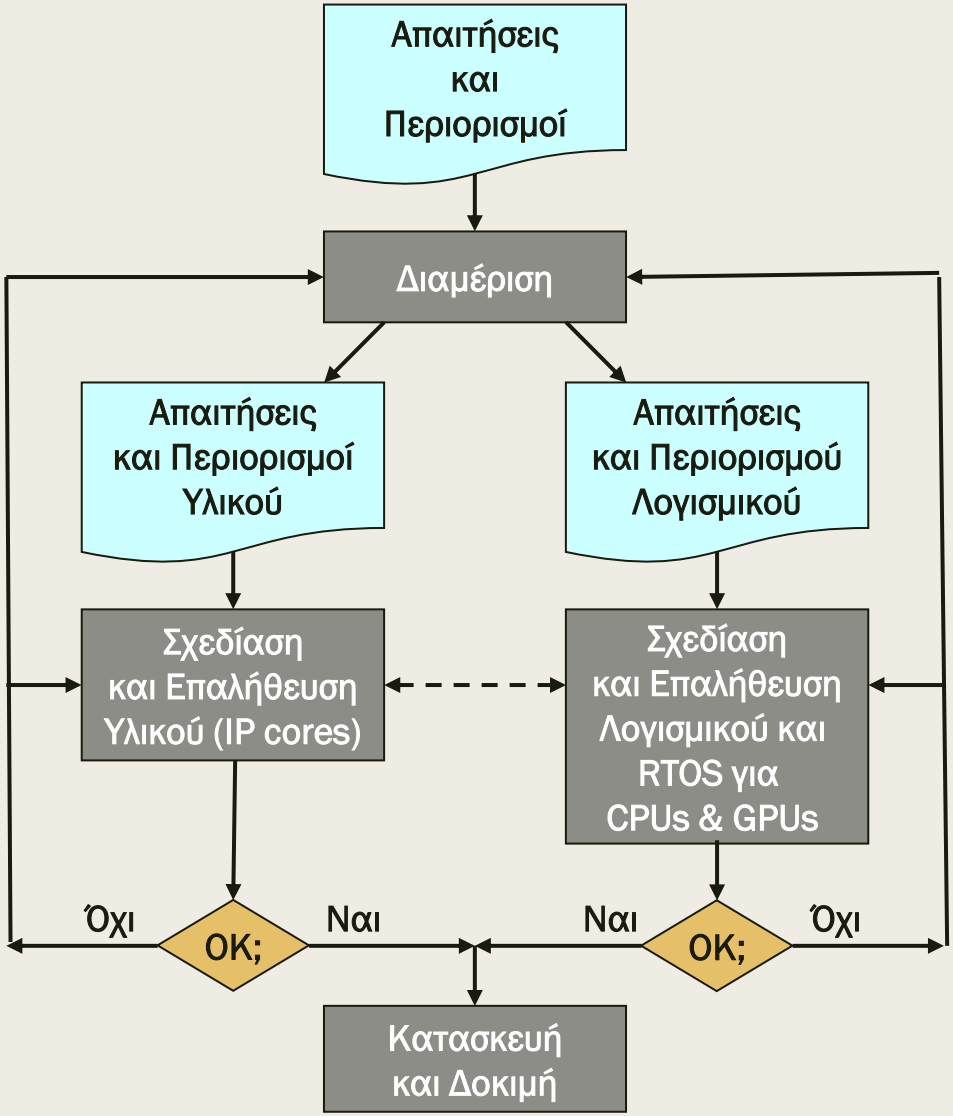
31

Επιταχυντές υλικού για ψηφιακά συστήματα υψηλής απόδοσης σε τεχνολογία FPGA

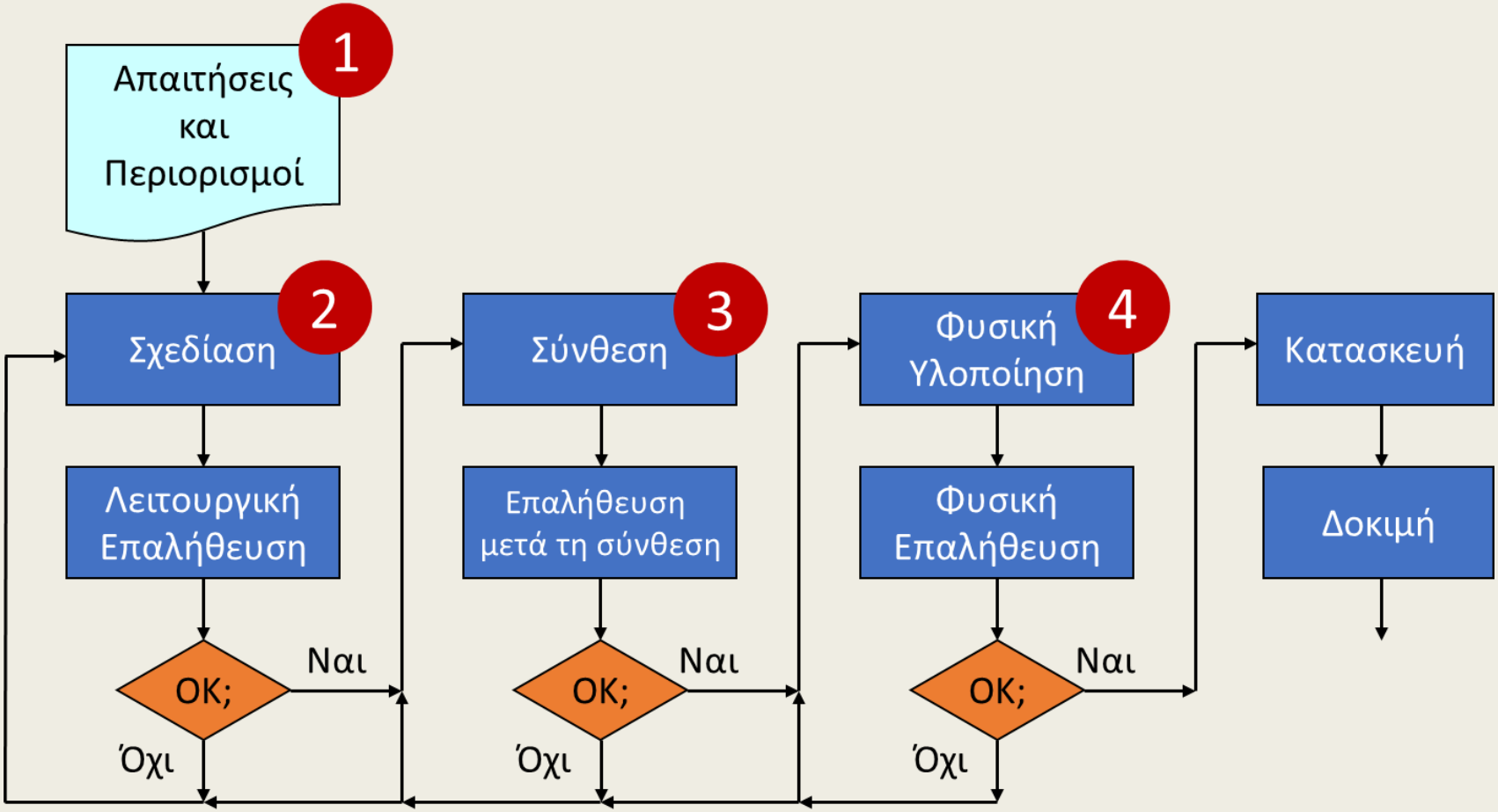
- Η τεχνολογική εξέλιξη που αλλάζει την κλασσική προσέγγιση του **υλικού** και τα όρια της **συ-σχεδίασης υλικού-λογισμικού**
  - *προγραμματιζόμενη λογική (για πυρήνες IP) + μνήμες + επεξεργαστές ARM + επεξεργαστές γραφικών + έτοιμοι επιταχυντές υλικού + συνδεσιμότητα υψηλής ταχύτητας σε ένα τσιπ (**Multi-Processing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



Συ-σχεδίαση υλικού-λογισμικού στα σύγχρονα ψηφιακά συστήματα



Φάσεις σχεδίασης υλικού



Φάσεις σχεδίασης υλικού

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

Προσδιορισμός απαιτήσεων και περιορισμοί

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

Εισαγωγή σχεδίασης

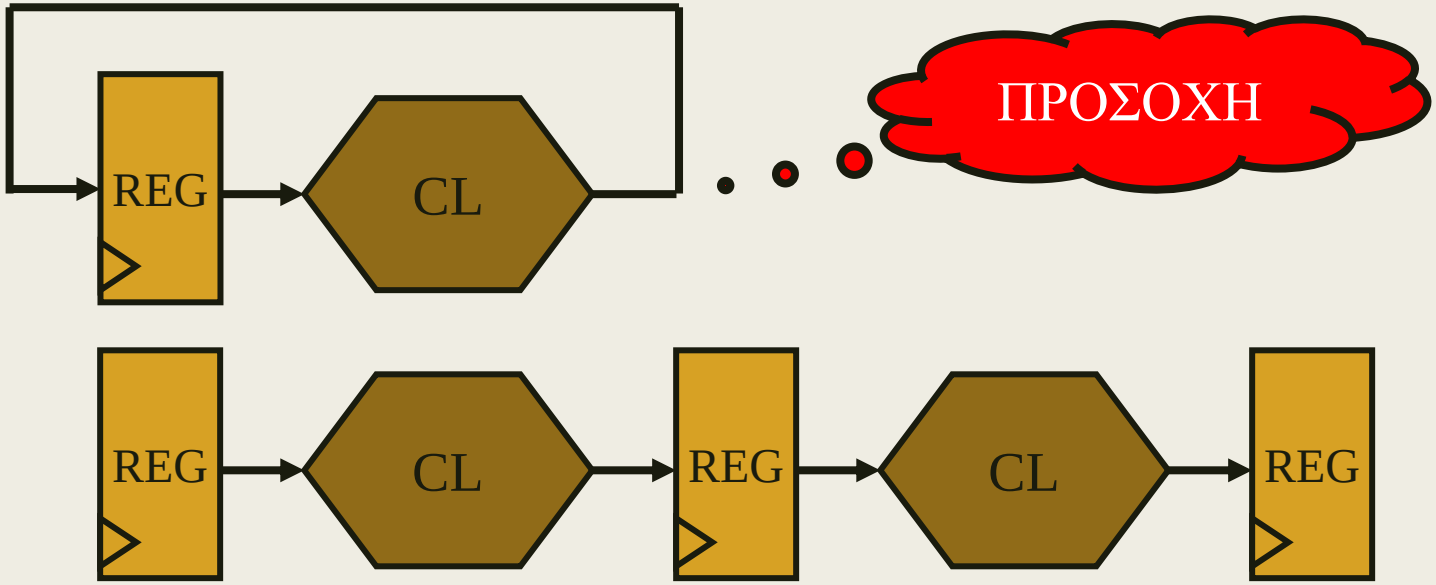
- Προγραμματισμός σε **γλώσσα περιγραφής υλικού (Hardware Description Language – HDL)**, αντί για σχηματικά διαγράμματα
  - *Υπερτερεί σε σύγκριση με τα σχηματικά διαγράμματα*
    - Η περιγραφή σε HDL γίνεται πιο κατανοητή από ένα σχηματικό διάγραμμα, λόγω καλλίτερης διαχείρισης της πολυπλοκότητας
 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

Εισαγωγή σχεδίασης

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

Περιγραφή Ψηφιακού Συστήματος σε Επίπεδο Μεταφοράς Καταχωρητή - RTL

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



Γλώσσες περιγραφής υλικού (HDL)

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

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

SystemVerilog

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

VHDL

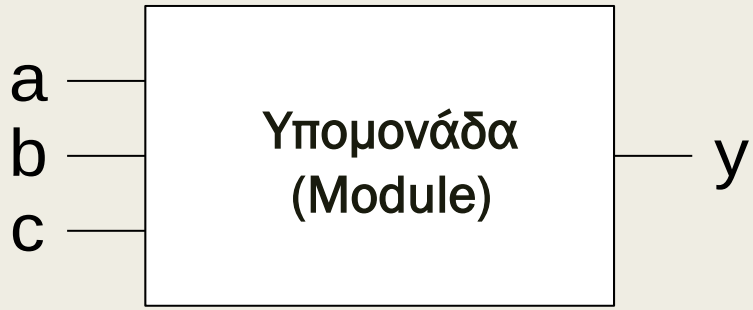
- Το αρκτικόλεξο VHDL προέρχεται από τα αρχικά της έκφρασης:

Very High Speed Integrated Circuits
Hardware Description Language

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

Υπομονάδα (Module)

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

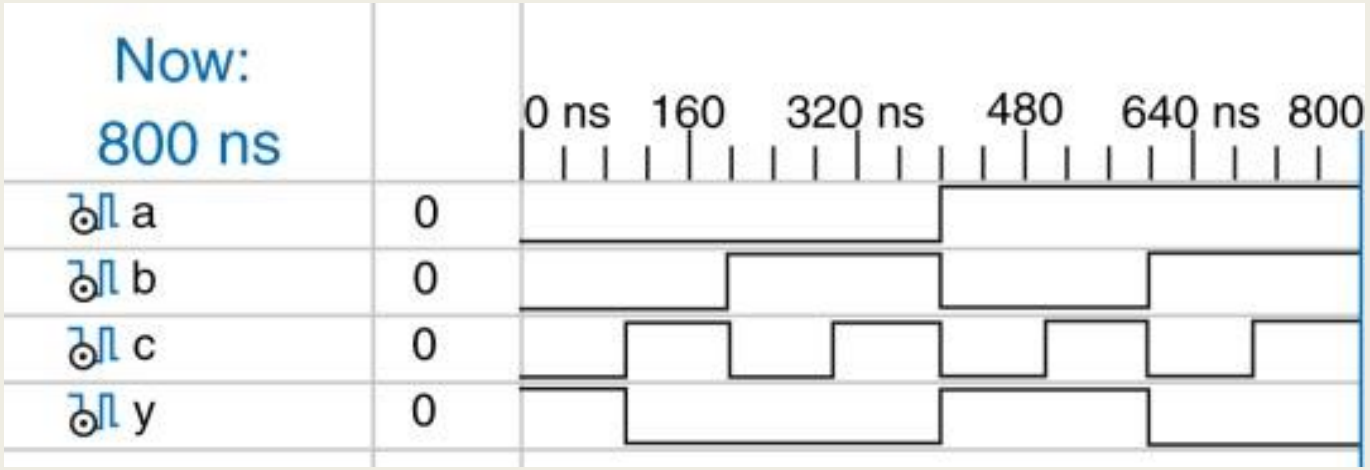


- Η λειτουργικότητα των υπομονάδων περιγράφεται με δύο τρόπους:
  - *ο πρώτος τρόπος βασίζεται στην **περιγραφή της συμπεριφοράς** της υπομονάδας*
  - *ο άλλος τρόπος βασίζεται στην **περιγραφή της δομής** της υπομονάδας.*
- Αντίστοιχα, προκύπτουν δύο διακριτά **μοντέλα περιγραφής της λειτουργικότητας**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

Προσομοίωση

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

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



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

Σφάλματα

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

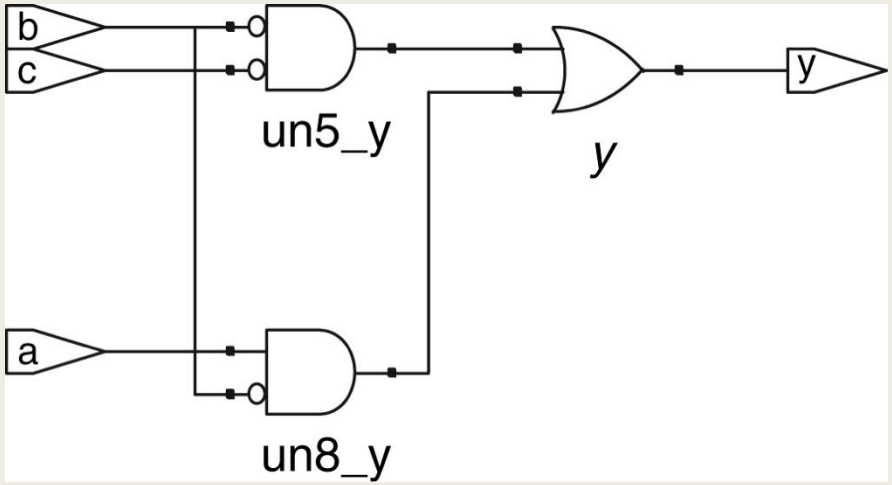
Σύνθεση

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

Σύνθεση

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

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



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

Πρόγραμμα Δοκιμών

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

Ιδιωματισμοί

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

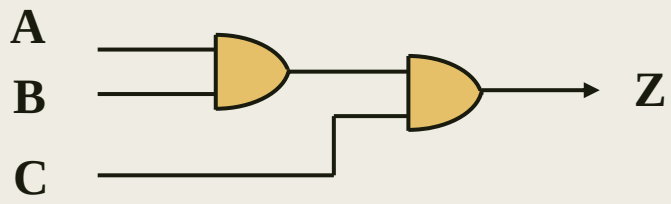
Ταυτόχρονες εντολές στη γλώσσα VHDL

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

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

≡

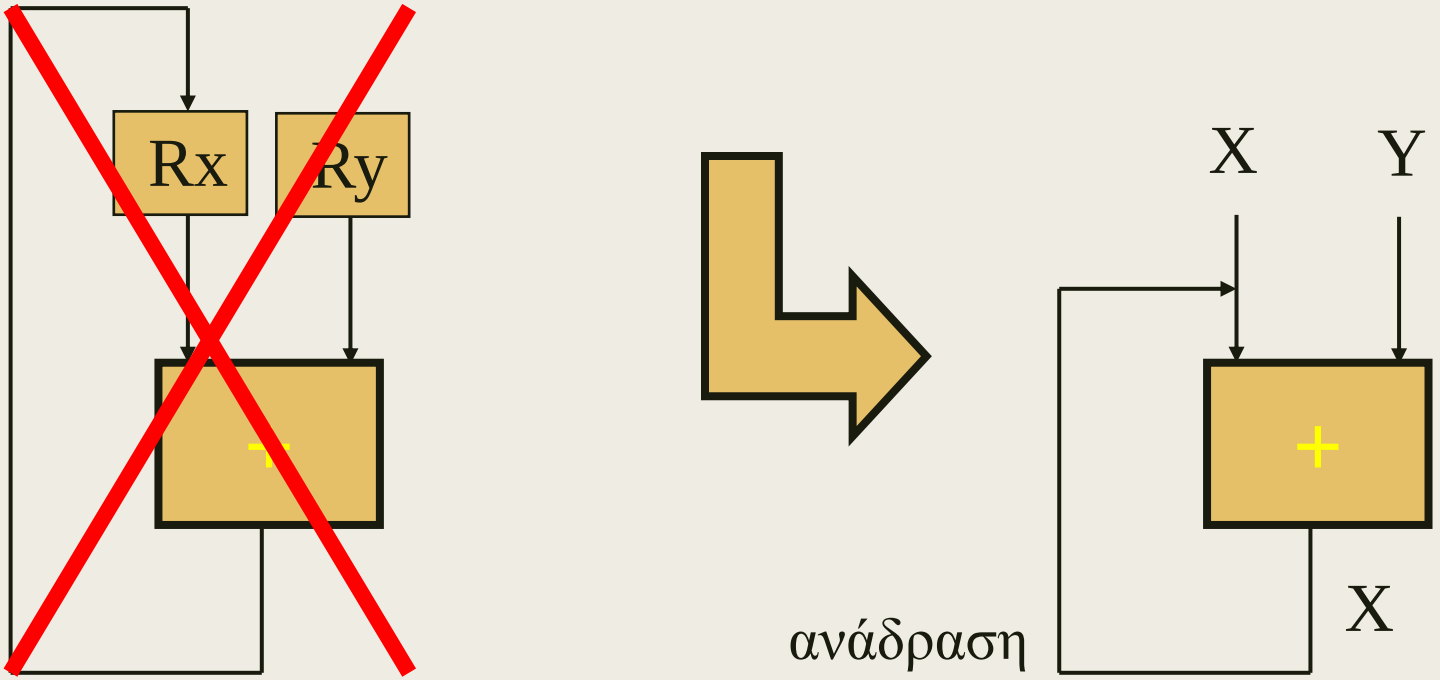
```
Z <= C and X
X <= A and B
```



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

Ταυτόχρονες εντολές – Προσοχή!

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



Δεν είναι όπως στο λογισμικό (software)

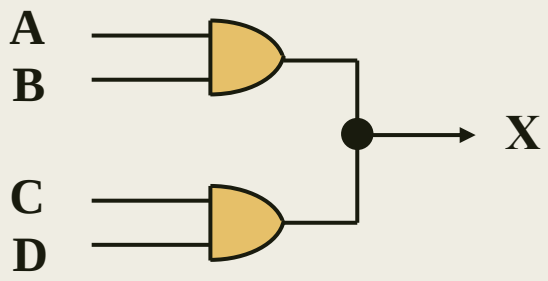
Ακολουθιακές εντολές στη γλώσσα VHDL

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

Ταυτόχρονες έναντι ακολουθιακών εντολών

Ταυτόχρονες εντολές

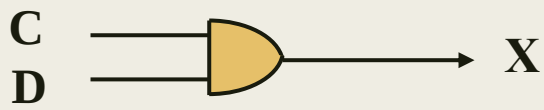
```
X <= A and B
X <= C and D
```



Ακολουθιακές εντολές μέσα σε δομή **process**

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

αγνοείται



Η οντότητα (entity) στη γλώσσα VHDL

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

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

Η οντότητα (entity) στη γλώσσα VHDL

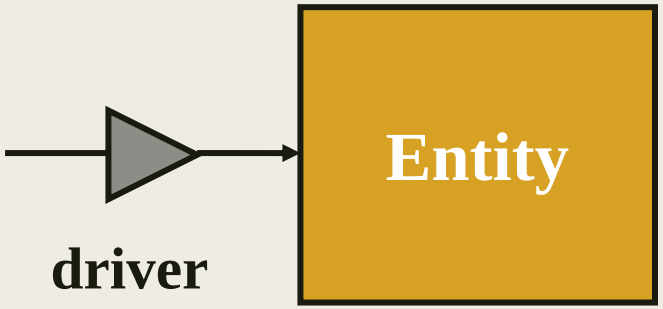
- entity_name: το όνομα της οντότητας
- signal_name: το όνομα του σήματος
  (εάν είναι πολλά σήματα χωρίζονται με κόμμα)
- mode: η κατεύθυνση του driver του σήματος
  - *in: είσοδος της οντότητας*
  - *out: έξοδος της οντότητας*
  - *inout: είσοδος ή έξοδος της οντότητας (bidirectional),*
- signal_type: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

Ονόματα και ετικέτες στη γλώσσα VHDL

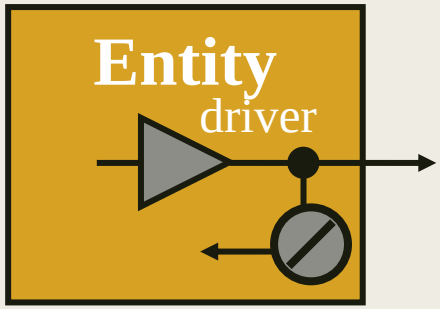
- Είναι **μοναδικά** μέσα σε μία συγκεκριμένη οντότητα (και αρχιτεκτονική)
- Χρησιμοποιούνται οι χαρακτήρες: **a-z, A-Z, 0-9, "_"**
- Δεν χρησιμοποιούνται οι χαρακτήρες, όπως: **+, -, !, &**
- Δεν χρησιμοποιούνται ούτε **σημεία στίξης** στα ονόματα και τις ετικέτες, ούτε διπλό "_", δηλαδή **"__"**
-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

Σημασία του mode στο port signal

Mode **in**

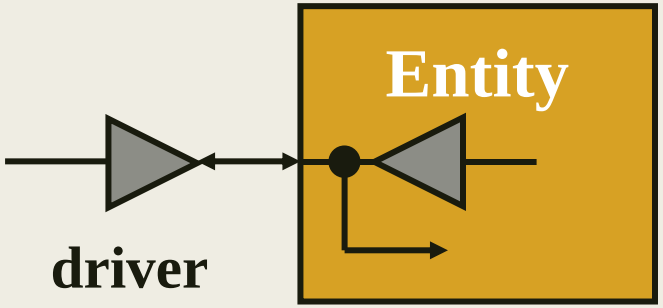


Mode **out**

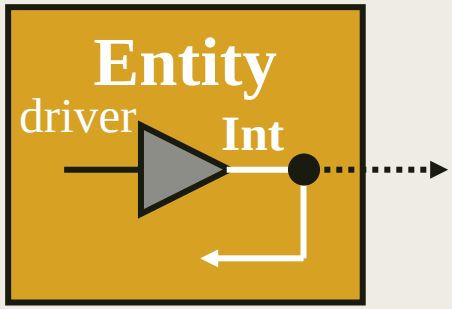


Προσοχή! Τα σήματα εξόδου δεν χρησιμοποιούνται σαν είσοδοι εντός της οντότητας

Mode **inout**



Mode **out**



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

Σημασία του signal_type στο port signal

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

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

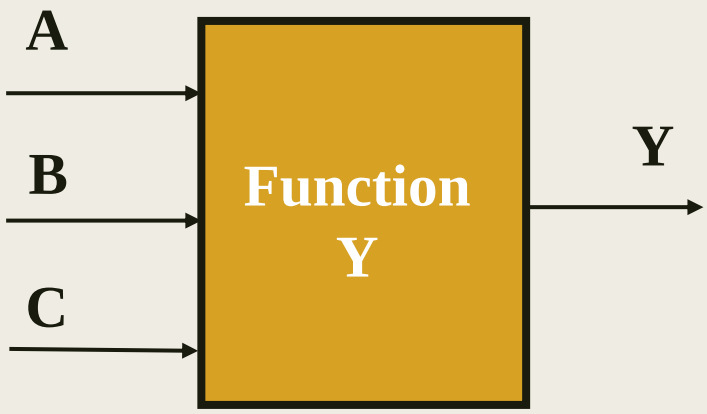
```
library IEEE;
use IEEE.std_logic_1164.all;
```

Οντότητα μίας υπομονάδας στη γλώσσα VHDL

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

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

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



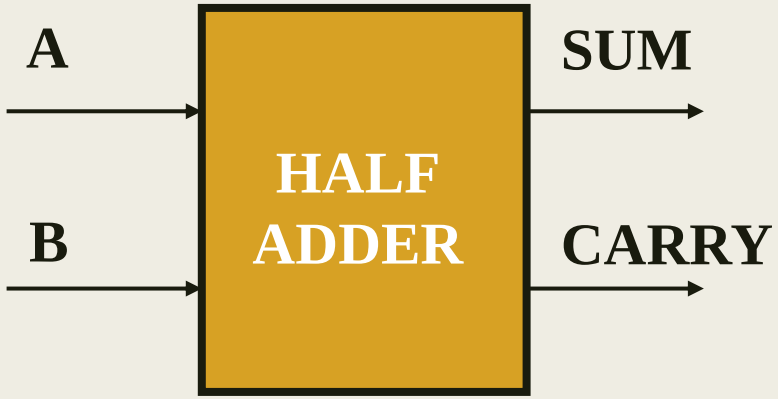
Η οντότητα του ημιαθροιστή στη VHDL

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

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

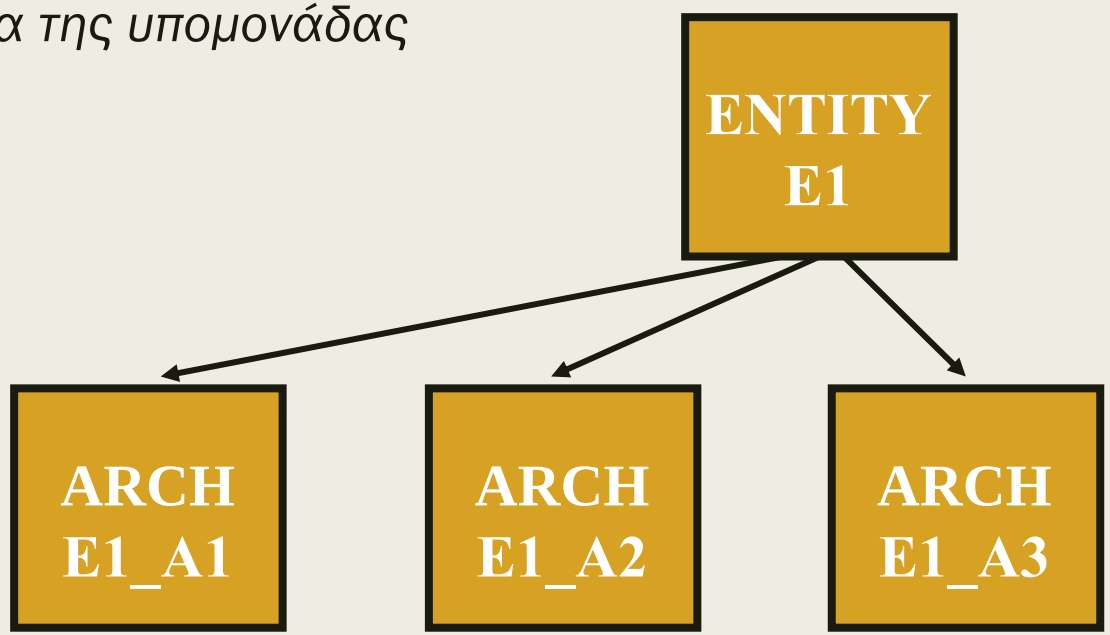
$$\mathbf{CARRY = AB}$$

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



Οντότητα και αρχιτεκτονική στη γλώσσα VHDL

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



Η αρχιτεκτονική (architecture) στη VHDL

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

Περιγραφή δομής στη VHDL

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

Περιγραφή δομής στη VHDL

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

Περιγραφή δομής στη VHDL

- Ταυτόχρονες εντολές στοιχείων (concurrent_component_statements)

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

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

Η αρχιτεκτονική του ημιαθροιστή στη VHDL
Περιγραφή δομής

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

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

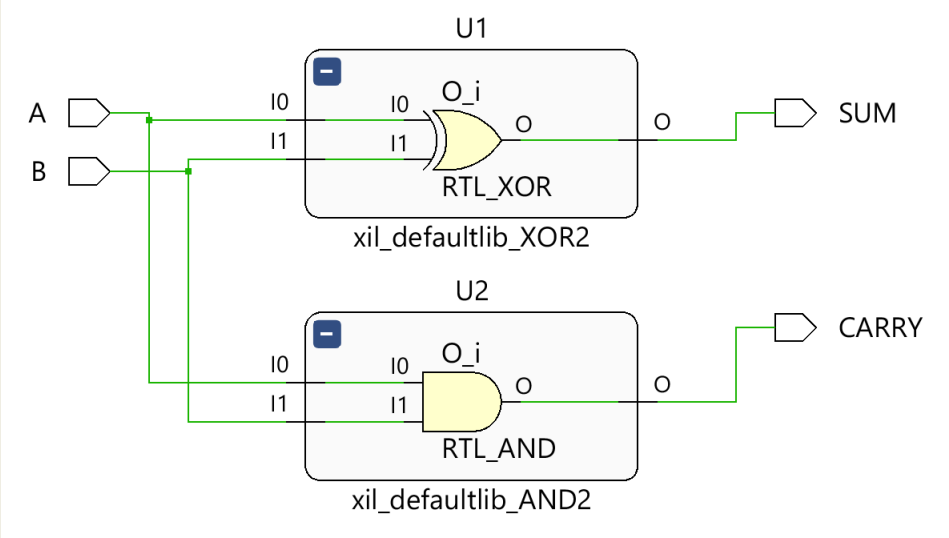
$$\mathbf{CARRY = AB}$$

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

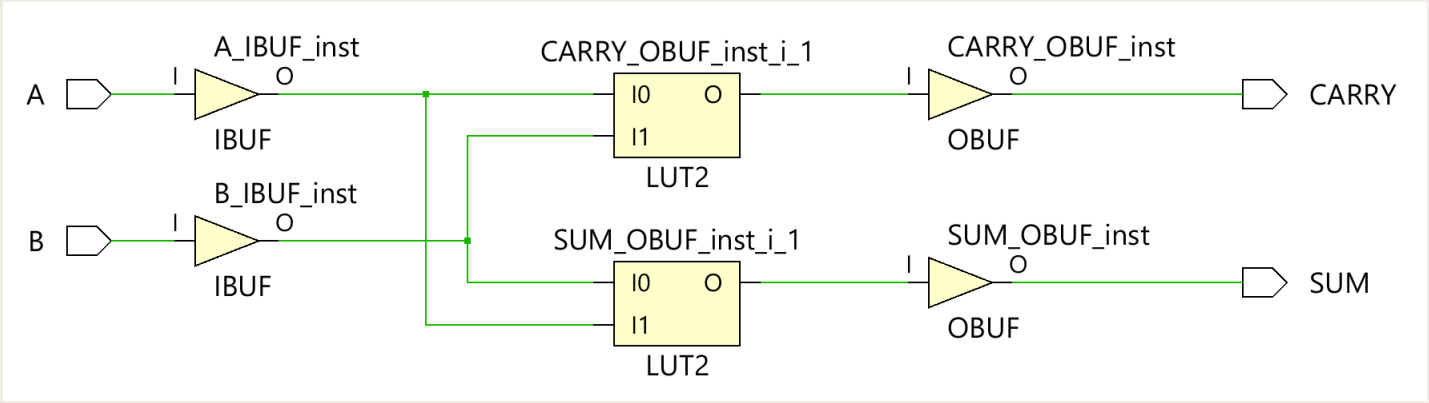
Τα στοιχεία XOR2 και AND2 έχουν ήδη προκαθοριστεί σαν οντότητες του project

Η αρχιτεκτονική του ημιαθροιστή στη VHDL
Σύνθεση περιγραφής δομής

- Σχηματικό διάγραμμα RTL (φαίνεται και η ιεραρχία)



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Περιγραφή dataflow στη VHDL

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

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

Περιγραφή dataflow στη VHDL

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

```
signal_name <= expression;
```

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

Περιγραφή dataflow στη VHDL

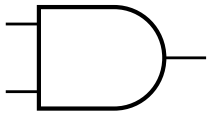
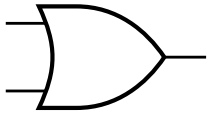
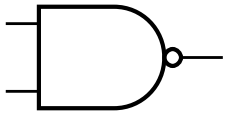
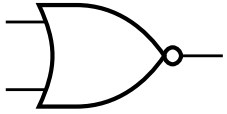
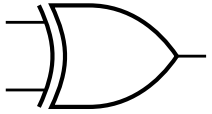
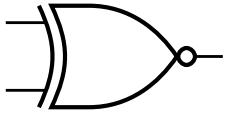
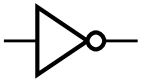
- **Εκτέλεση** ταυτόχρονων εντολών ανάθεσης σήματος (concurrent_signal_assignment_statements)

```
signal_name <= expression;
```

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

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

Λογικοί τελεστές και προτεραιότητα λογικών πράξεων στη VHDL

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

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

Προτεραιότητα Πράξεων στη VHDL

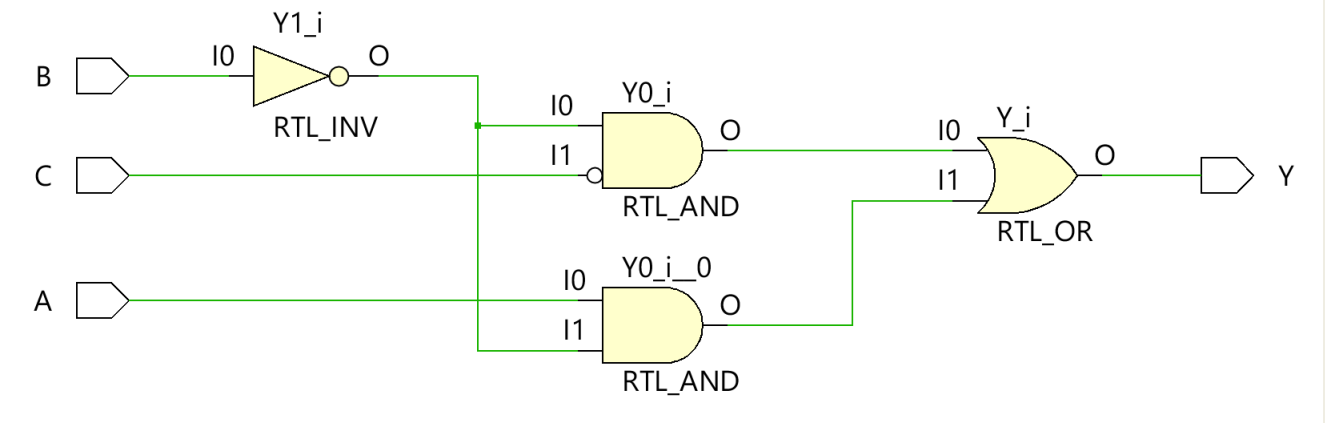
	Τελεστής	Σημασία
Υψηλότερη	not	NOT
	* / mod rem	MUL, DIV, MOD, REM
	+ –	PLUS, MINUS
	rol ror srl sll	Περιστροφή, λογική ολίσθηση
	< <= > >=	Σχετική σύγκριση
	= /=	Σύγκριση ισότητας
Χαμηλότερη	and or nand nor xor xnor	Λογικές πράξεις (εκτελούνται από αριστερά προς τα δεξιά)

Η αρχιτεκτονική μίας υπομονάδας στη VHDL
Περιγραφή dataflow

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

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

- Σχηματικό διάγραμμα RTL

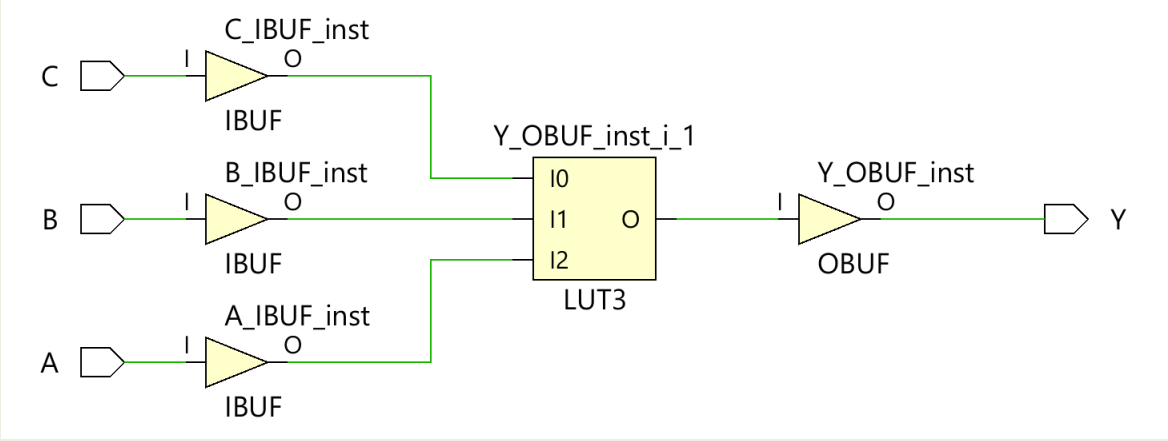


Η αρχιτεκτονική μίας υπομονάδας στη VHDL
Περιγραφή dataflow

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

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

- Σχηματικό διάγραμμα σε τεχνολογία FPGA

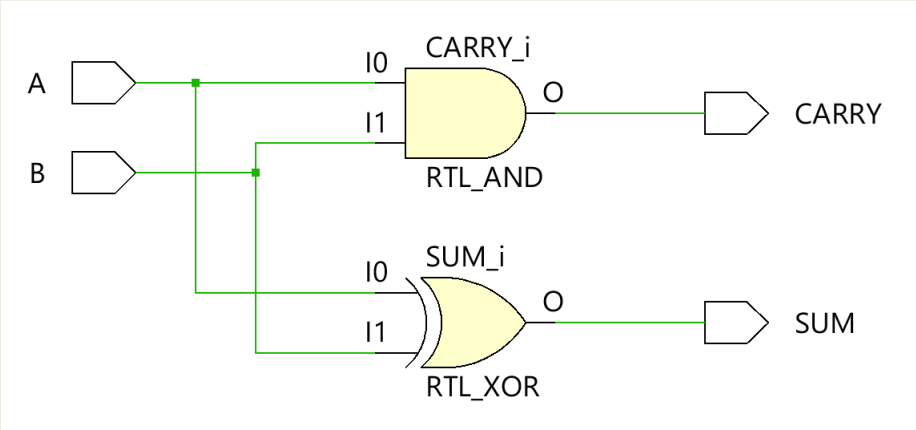


Η αρχιτεκτονική του ημιαθροιστή στη VHDL
Περιγραφή dataflow

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

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

- Σχηματικό διάγραμμα RTL

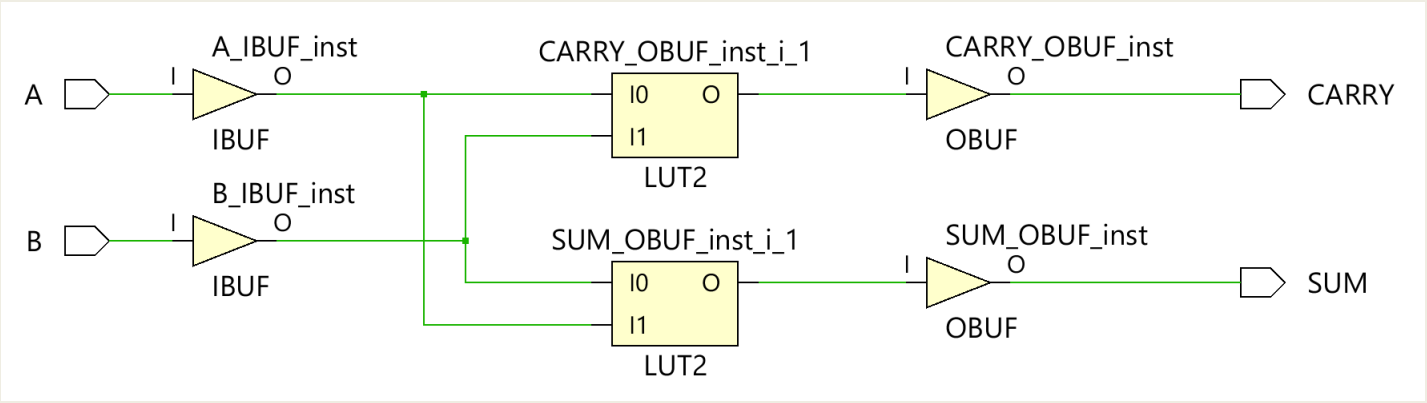


Η αρχιτεκτονική του ημιαθροιστή στη VHDL
Περιγραφή dataflow

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

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

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Η αρχιτεκτονική του ημιαθροιστή στη VHDL
Περιγραφή dataflow για προγράμματα δοκιμής

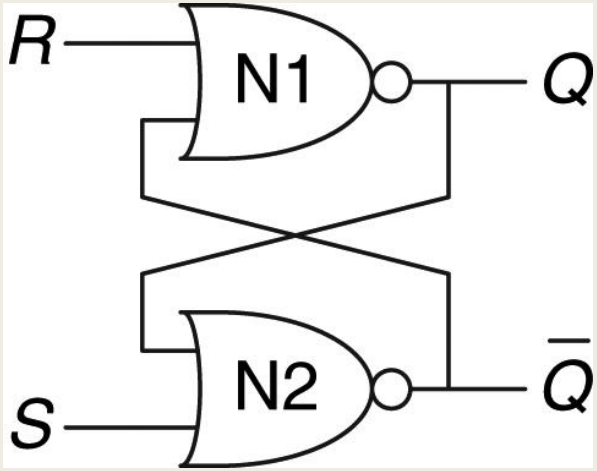
```
architecture HA_DATAFLOW of HALF_ADDER is
begin
    SUM <= A xor B after 10 ns;
    CARRY <= A and B after 5 ns;
end HA_ DATAFLOW;
```

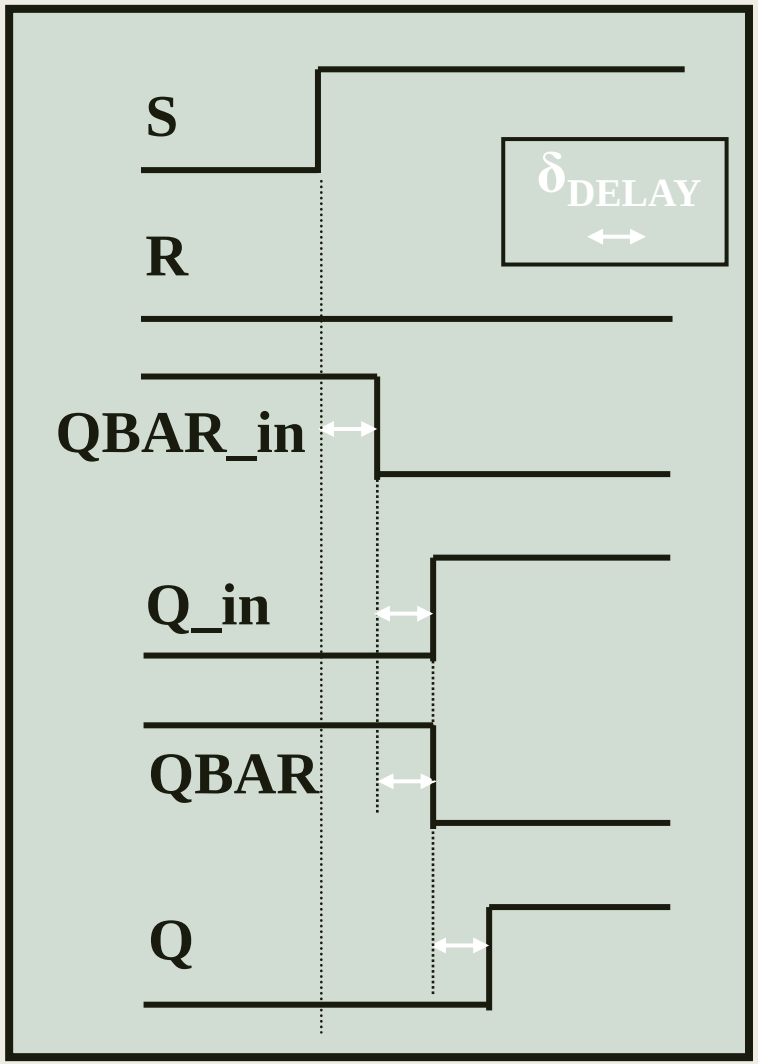
Στα προγράμματα δοκιμής μπορεί να προσδιορισθεί για τις ανάγκες τις προσομοίωσης και καθυστέρηση διάδοσης με τη φράση after. Ο χρόνος αρχίζει να μετράει με την αλλαγή της τιμής σε μία από τις εισόδους. Το after αγνοείται κατά τη σύνθεση.

Το S-R Latch (Active High) στη VHDL

Περιγραφή Dataflow

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

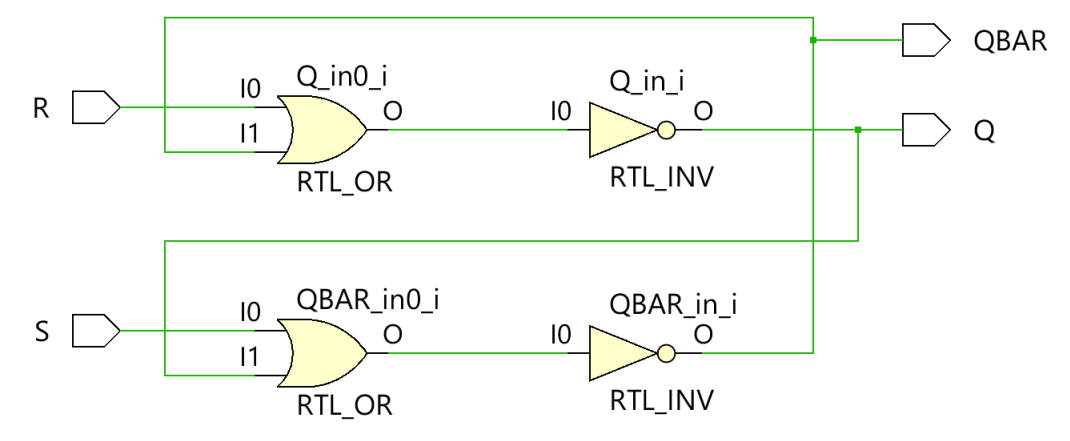




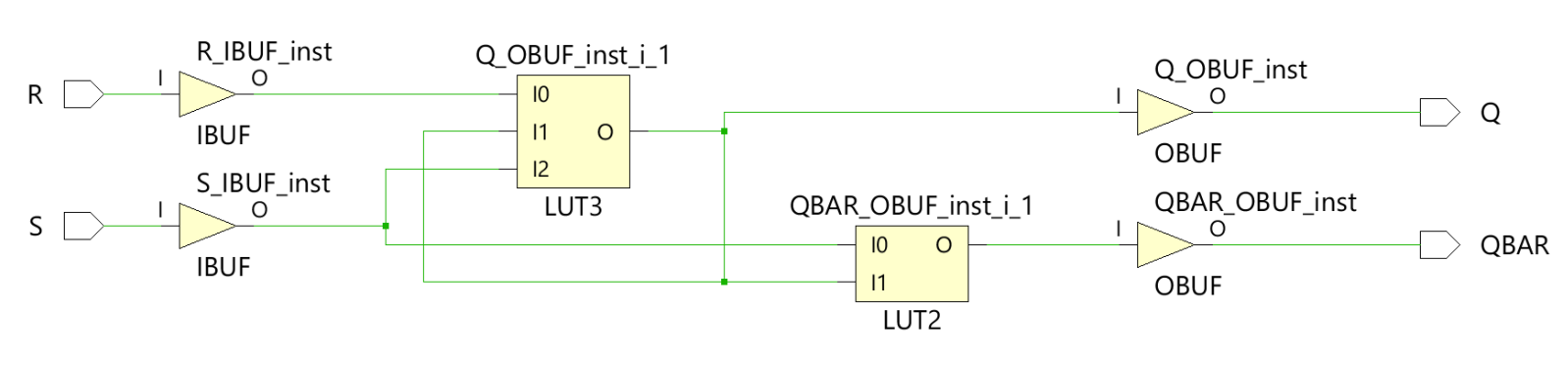
Η ενημέρωση των σημάτων γίνεται με καθυστέρηση δέλτα δ_{delay} αμέσως μόλις εκτελεστεί η αντίστοιχη εντολή

Το SR Latch στη VHDL
Σύνθεση περιγραφής δομής

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Περιγραφή συμπεριφοράς (behavioral) στη VHDL

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

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

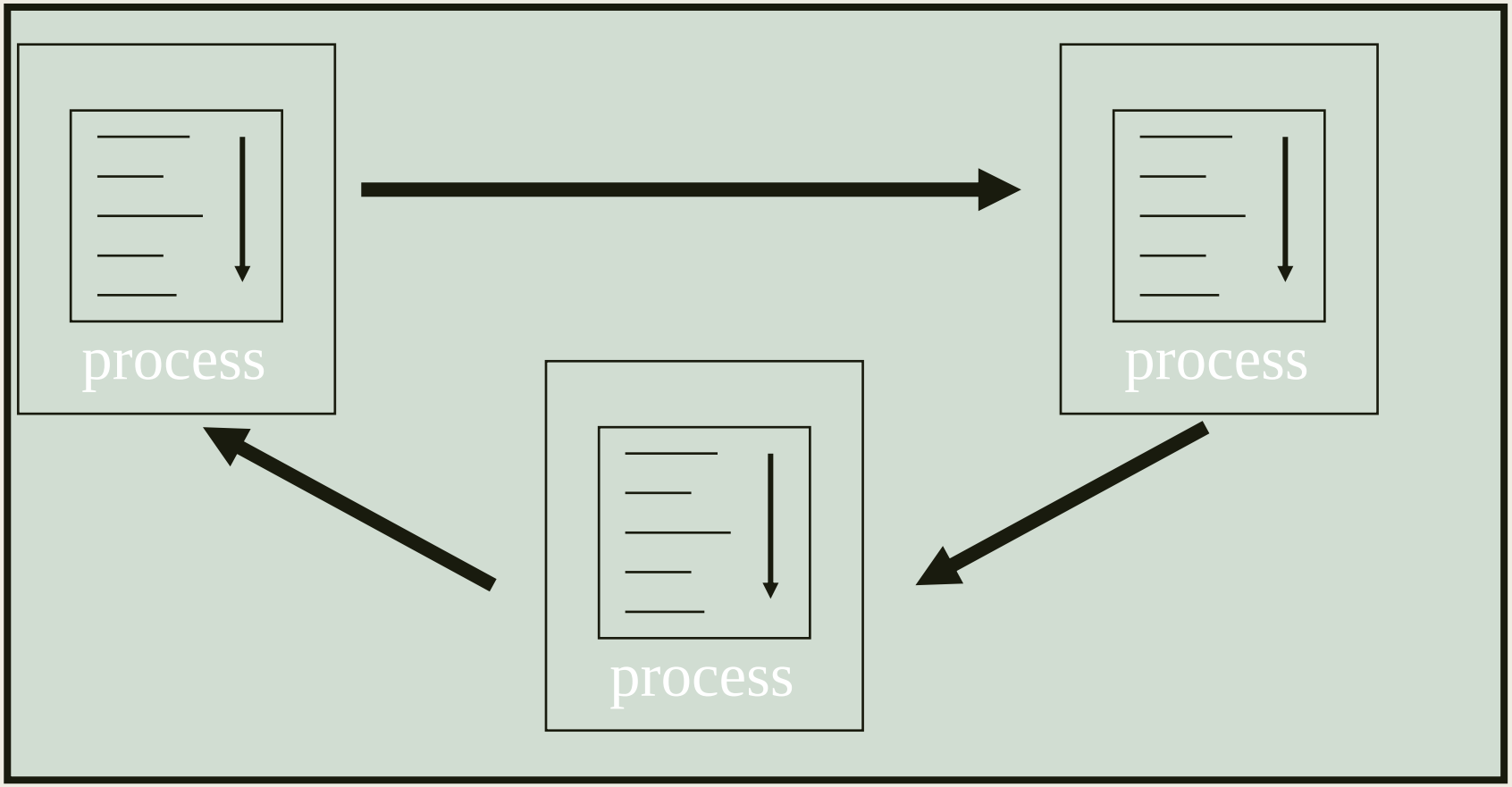
Περιγραφή συμπεριφοράς (behavioral) στη VHDL

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

Περιγραφή συμπεριφοράς (behavioral) στη VHDL

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

Περιγραφή συμπεριφοράς (behavioral) στη VHDL



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

Περιγραφή συμπεριφοράς (behavioral) στη VHDL

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

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Σήματα vs μεταβλητές

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

Η χρήση των μεταβλητών μειώνει σημαντικά το χρόνο της προσομοίωσης

Περιγραφή συμπεριφοράς (behavioral) στη VHDL

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

```
signal_name <= expression;
```

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

Περιγραφή συμπεριφοράς (behavioral) στη VHDL

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

```
variable_name := expression;
```

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

Δεν χρησιμοποιείται στην υλοποίηση ακολουθιακής λογικής

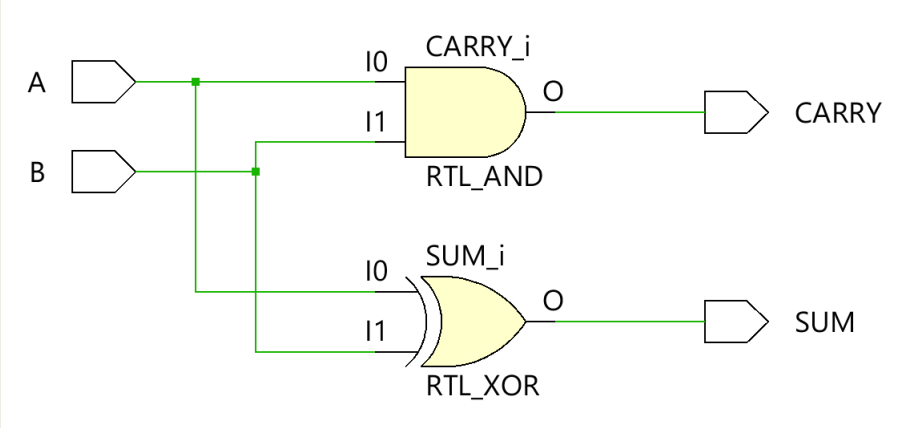
Η αρχιτεκτονική του ημιαθροιστή στη VHDL
Περιγραφή συμπεριφοράς

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

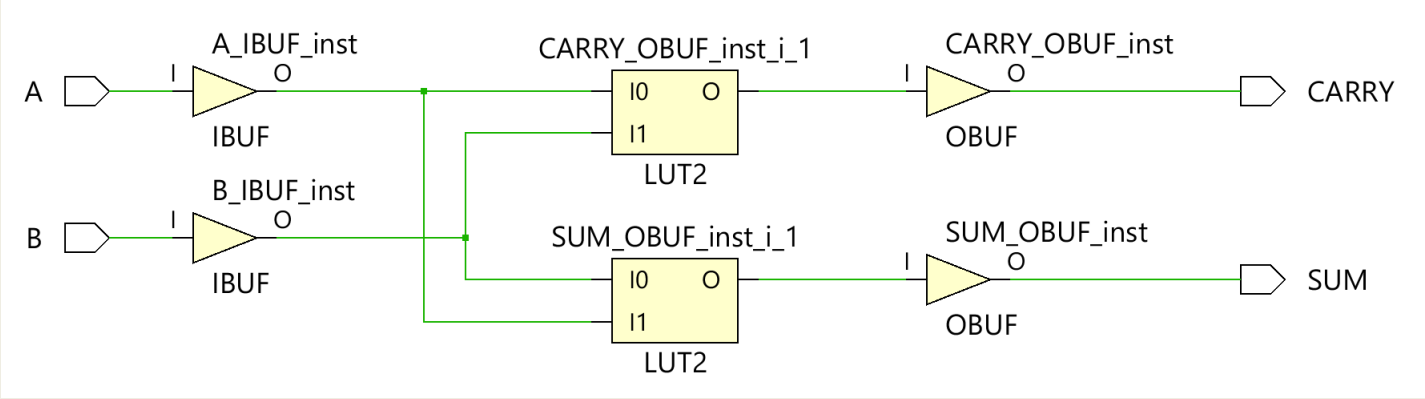
Οι ακολουθιακές εντολές ανάθεσης σήματος εκτελούνται η μία μετά την άλλη, μόνο όταν υπάρξει αλλαγή τιμής στις εισόδους που δηλώνονται στη λίστα ευαισθησίας (στα σήματα της δεξιάς πλευράς)

Η αρχιτεκτονική του ημιαθροιστή στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL

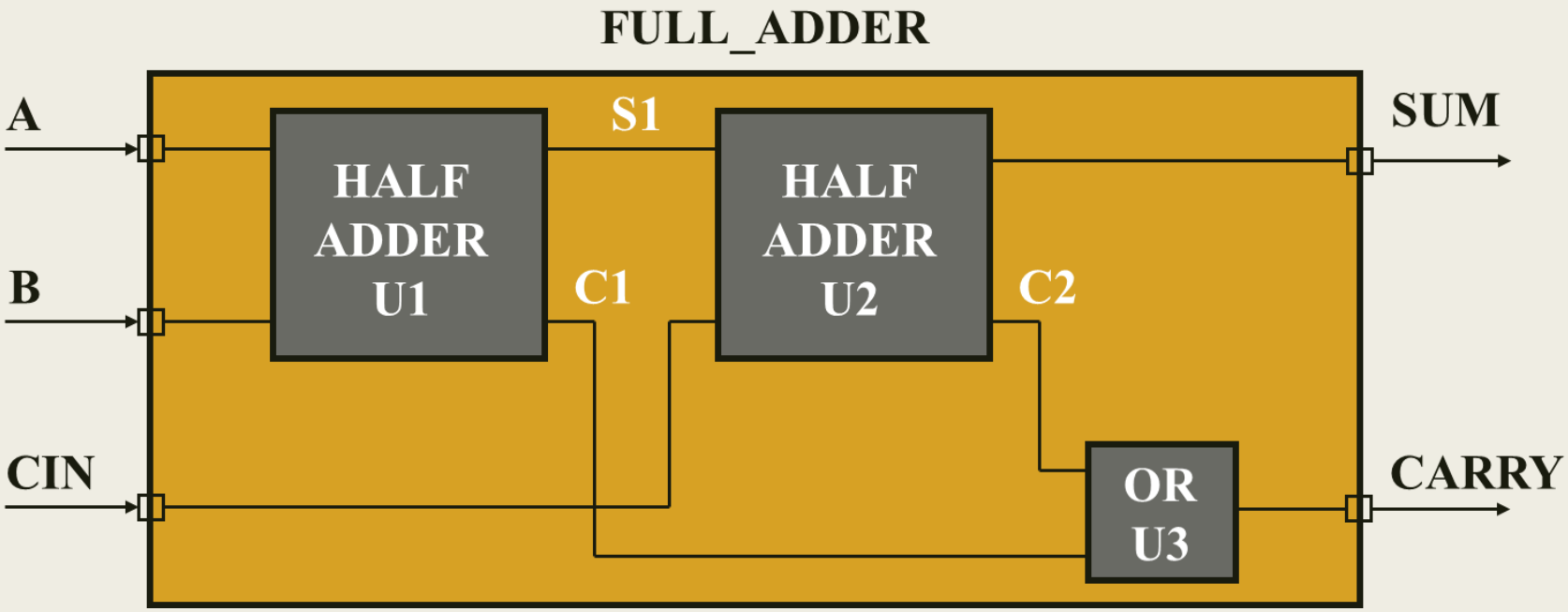


- Σχηματικό διάγραμμα σε τεχνολογία FPGA



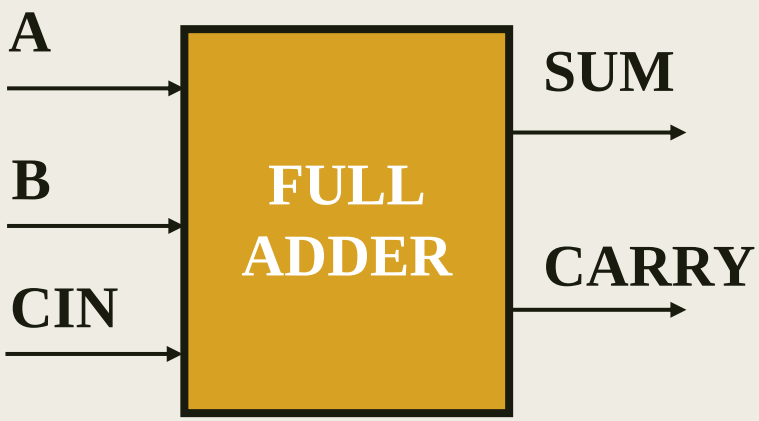
Δημιουργία ιεραρχικής δομής στη VHDL
Περιγραφή δομής

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



Η οντότητα του πλήρους αθροιστή στη VHDL

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



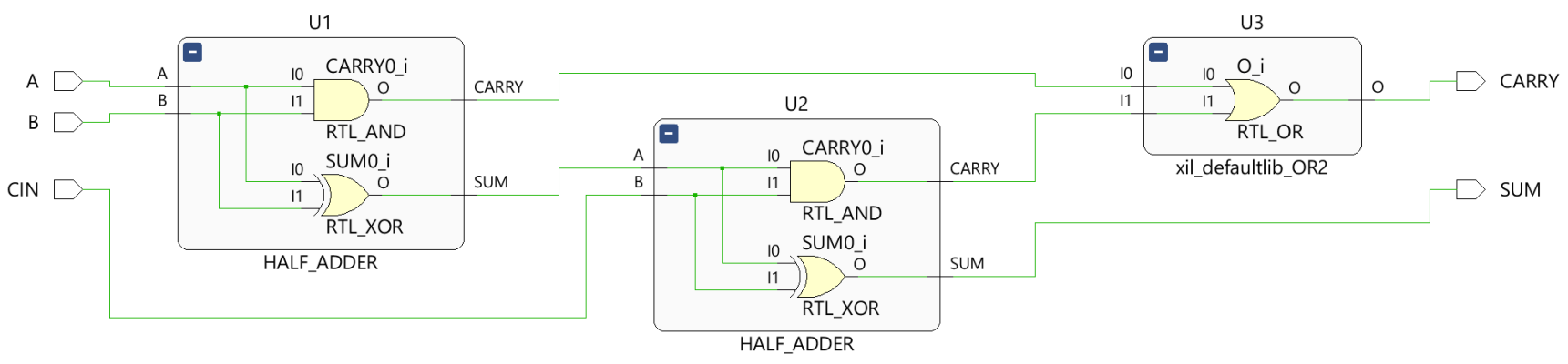
Η αρχιτεκτονική του πλήρους αθροιστή στη VHDL
Περιγραφή δομής

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

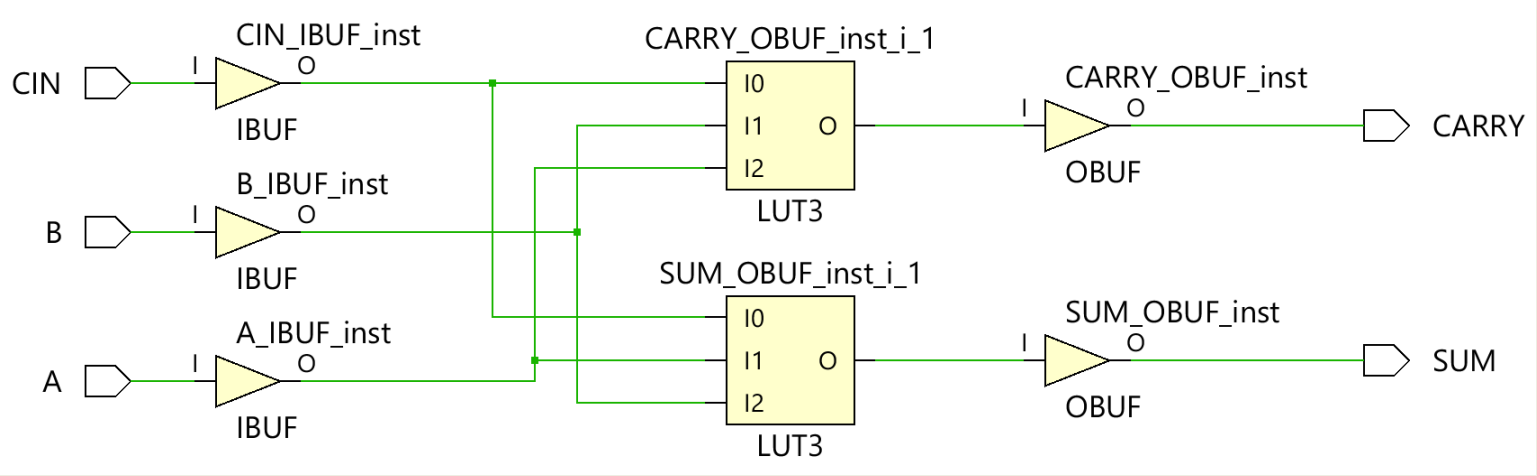
Τα στοιχεία HALF_ADDER και OR2 έχουν ήδη προκαθοριστεί σαν οντότητες (προσέγγιση σχεδίασης **bottom-up**) του project

Η αρχιτεκτονική του πλήρους αθροιστή στη VHDL
Περιγραφή δομής

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Η αρχιτεκτονική του πλήρους αθροιστή στη VHDL
Περιγραφή dataflow

- Με εσωτερικά σήματα (δηλώνονται στο signal)

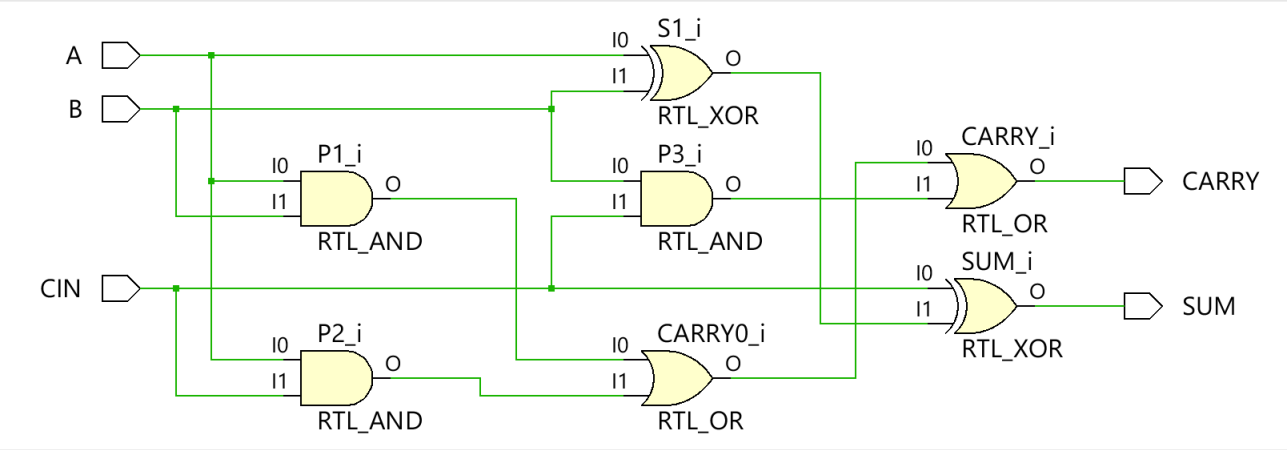
```
architecture FA_DATAFLOW1 of FULL_ADDER is
    signal S1, P1, P2, P3: STD_LOGIC;
begin
    S1 <= A xor B;
    SUM <= CIN xor S1;
    P1 <= A and B;
    P2 <= A and CIN;
    P3 <= B and CIN;
    CARRY <= P1 or P2 or P3;
end FA_DATAFLOW1;
```

- Χωρίς εσωτερικά σήματα

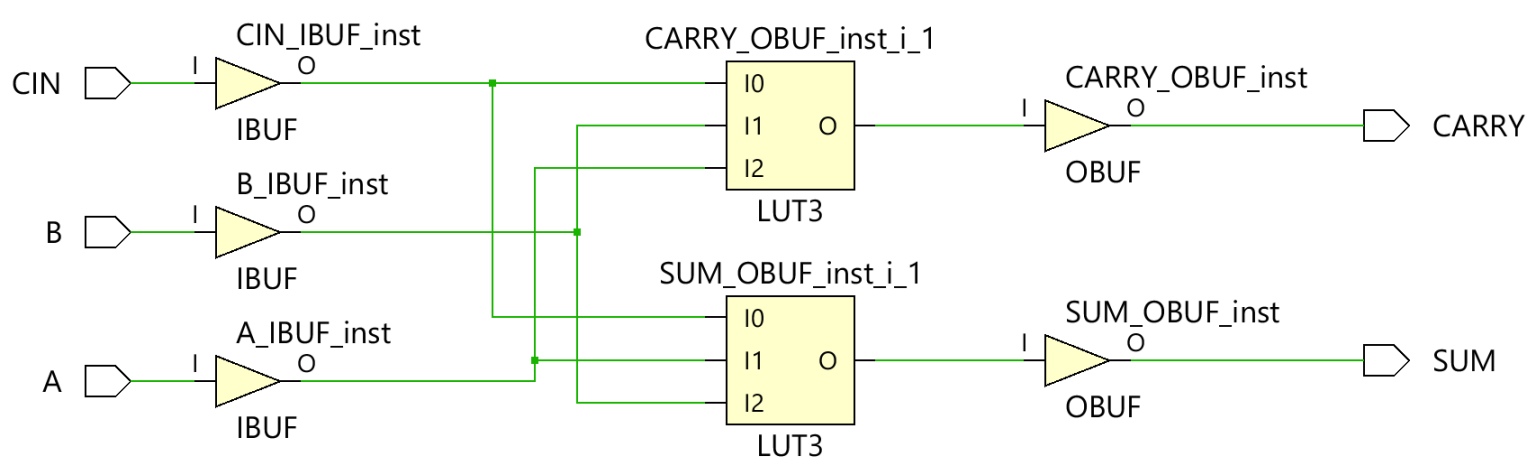
```
architecture FA_DATAFLOW2 of FULL_ADDER is
begin
    SUM <= A xor B xor CIN;
    CARRY <= (A and B) or (A and CIN) or (B and CIN);
end FA_DATAFLOW2;
```

Η αρχιτεκτονική του πλήρους αθροιστή στη VHDL
Περιγραφή dataflow

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA

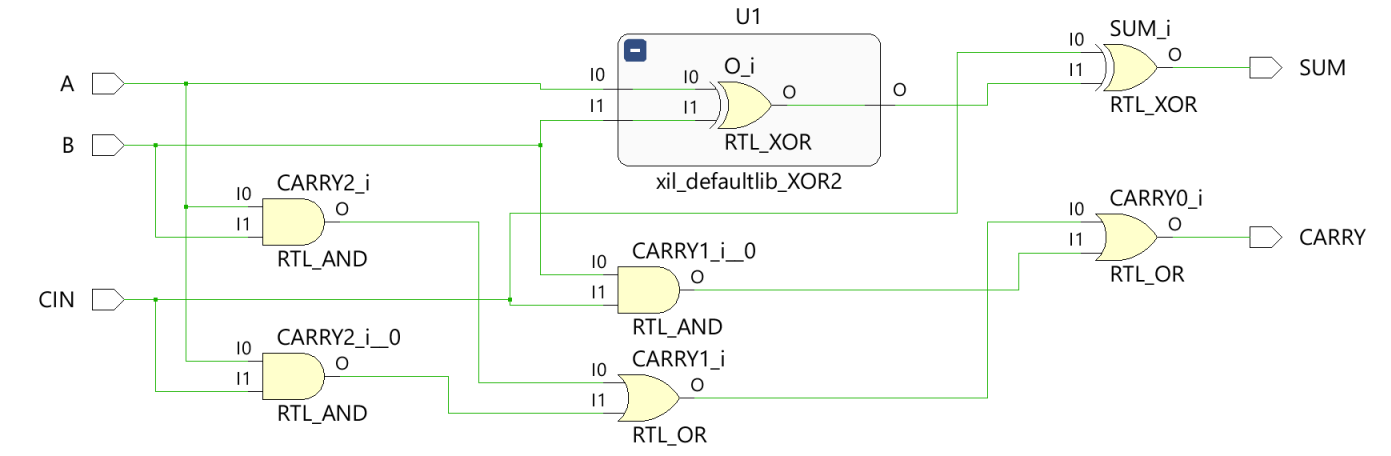


Η αρχιτεκτονική του πλήρους αθροιστή στη VHDL
Μικτή περιγραφή (δομής, dataflow, συμπεριφοράς)

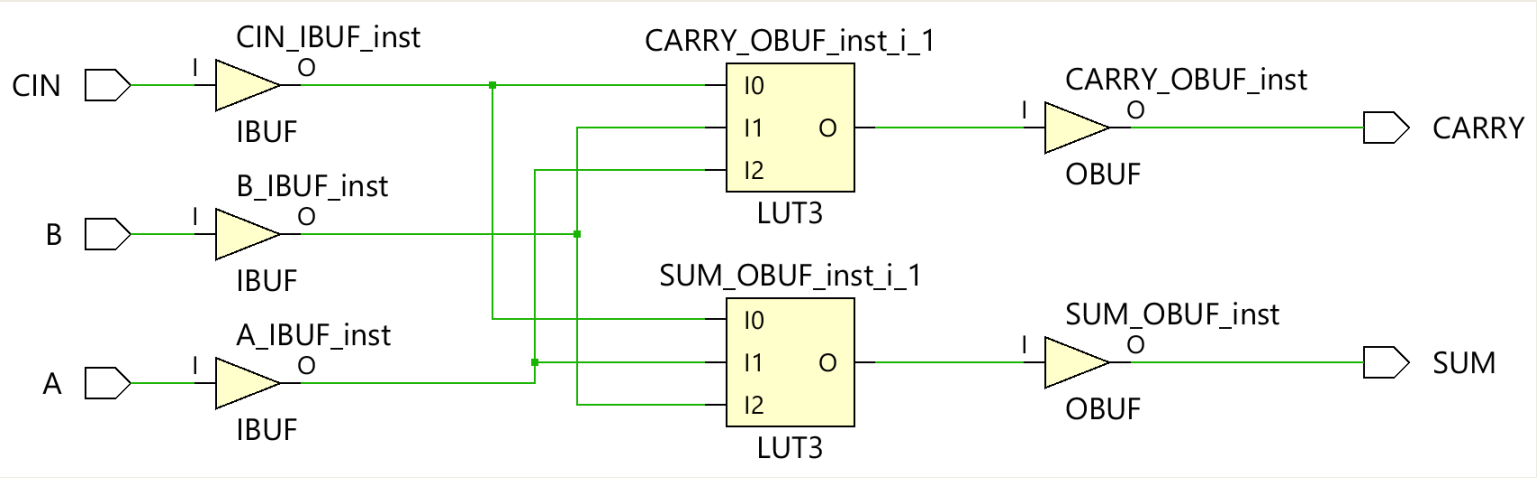
```
architecture FA_MIXED of FULL_ADDER is
    signal S1: STD_LOGIC;
    component XOR2
        port (O : out STD_LOGIC;
              I0 : in STD_LOGIC; I1 : in STD_LOGIC);
    end component;
begin
    U1: XOR2 port map (O => S1, I0 => A, I1 => B);
    SUM <= CIN xor S1;
    process (A, B, CIN)
   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

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

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Εντολές με συνθήκη

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

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

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Εντολές με συνθήκη

- Στις πιο σημαντικές ακολουθιακές εντολές ανάθεσης σήματος (ή μεταβλητής) με συνθήκη συμπεριλαμβάνονται:
  - *η εντολή **IF** (IF statement)*
  - *η εντολή **CASE** (CASE statement)*
  - *η εντολή **FOR LOOP** (FOR LOOP statement)*
  - *η εντολή **WHILE LOOP** (WHILE LOOP statement)*
- Στις πιο σημαντικές ταυτόχρονες εντολές ανάθεσης σήματος με συνθήκη συμπεριλαμβάνεται
  - *η εντολή **FOR GENERATE** (FOR GENERATE statement)*

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Η εντολή IF

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

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

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

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Η εντολή IF

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

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

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Η εντολή IF

- Boolean Expression
  - *υλοποιεί μία συνθήκη*

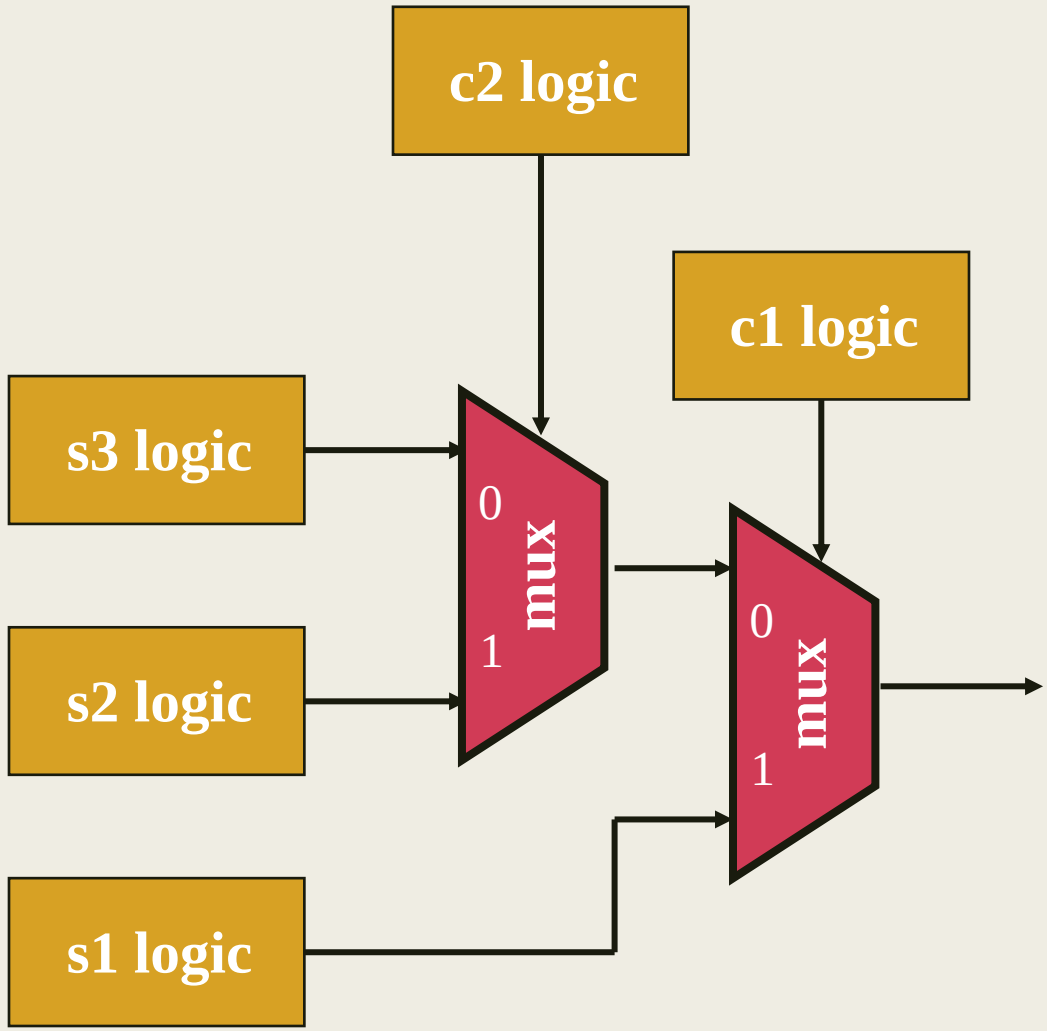
    (A = B)
  - *ή μια σειρά από συνθήκες που συνδέονται μεταξύ τους με λογικούς τελεστές (**and**, **or**, **not**, **nand**, **nor**, **xor**, **xnor**)*

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

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Η εντολή IF

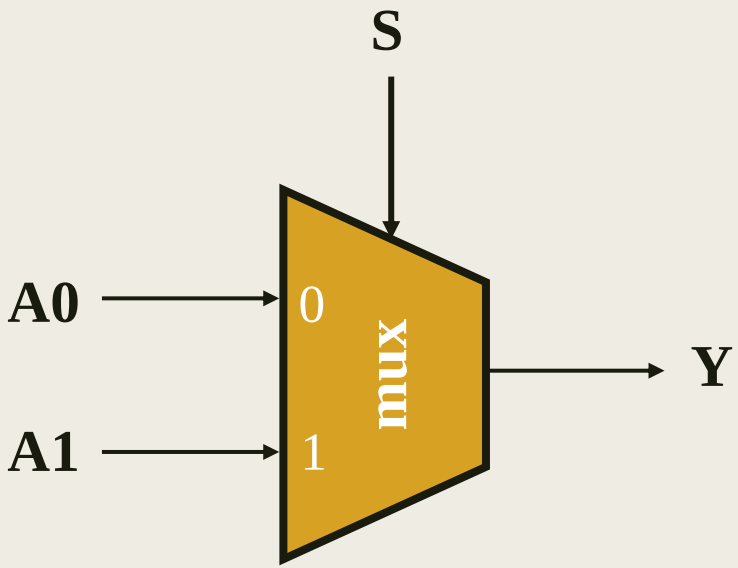
- Υλοποίηση εντολής IF με τη χρήση **πολυπλεκτών 2 σε 1**

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



Η οντότητα του πολυπλέκτη 2 σε 1 στη VHDL

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



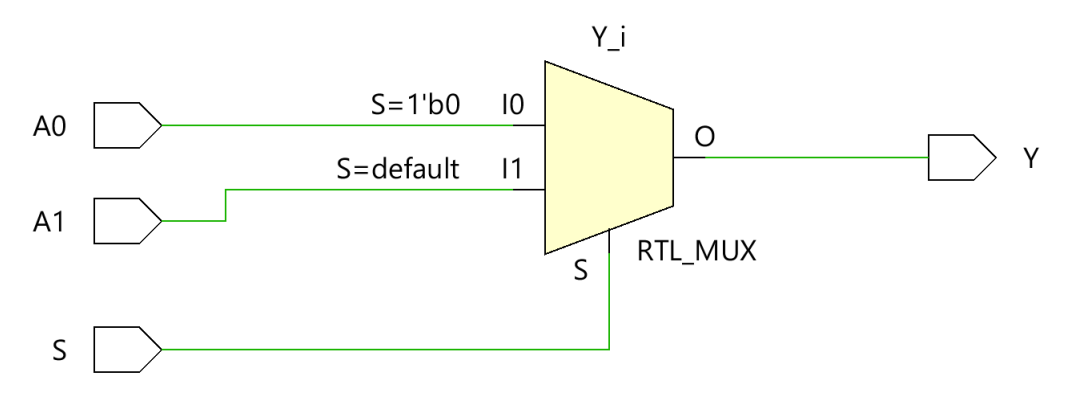
Η αρχιτεκτονική του πολυπλέκτη 2 σε 1 στη VHDL
Περιγραφή συμπεριφοράς

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

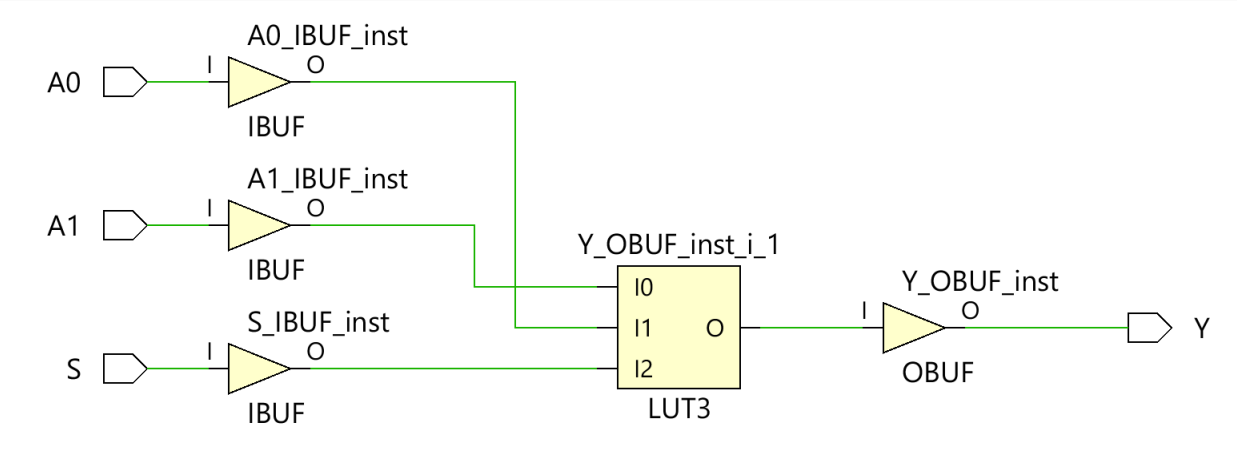
Στη λίστα ευαισθησίας συμπεριλαμβάνονται όλες
οι είσοδοι του συνδυαστικού κυκλώματος

Η αρχιτεκτονική του πολυπλέκτη 2 σε 1 στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL

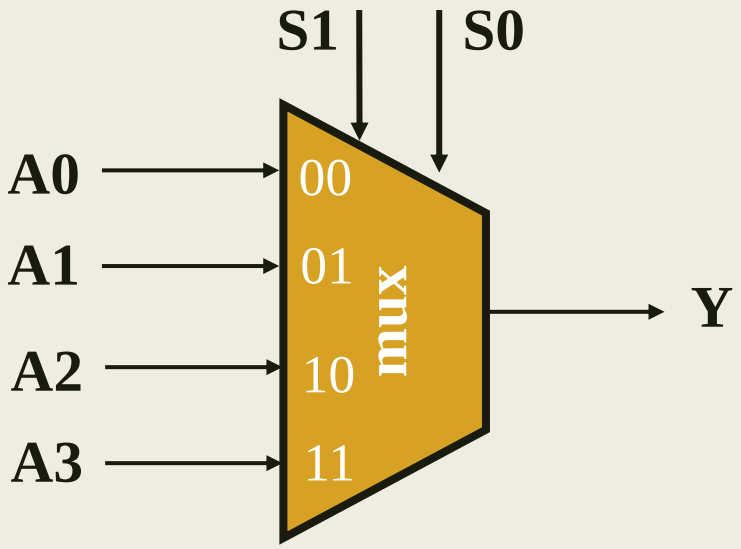


- Σχηματικό διάγραμμα σε τεχνολογία FPGA

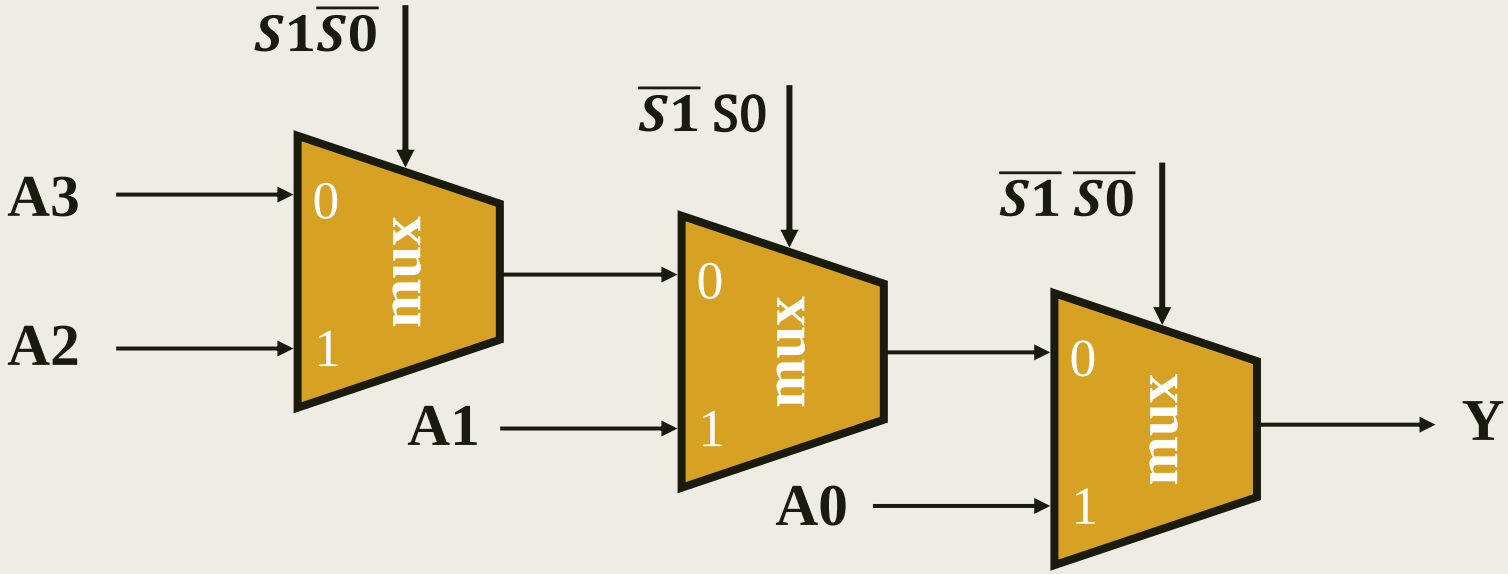


Η οντότητα του πολυπλέκτη 4 σε 1 στη VHDL

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



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



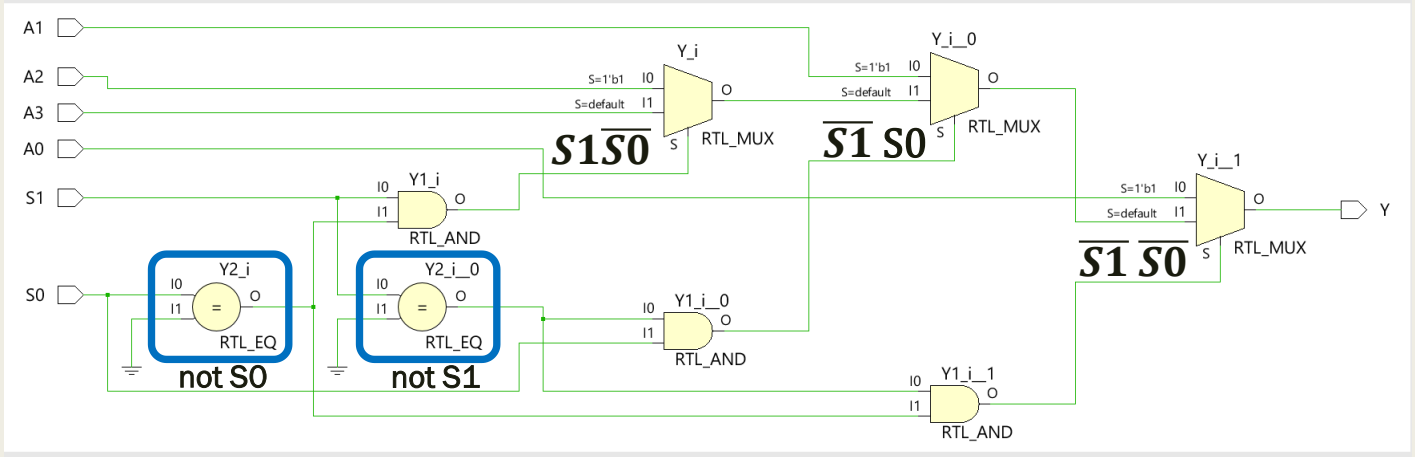
Η αρχιτεκτονική του πολυπλέκτη 4 σε 1 στη VHDL
Περιγραφή συμπεριφοράς – Λύση 1

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

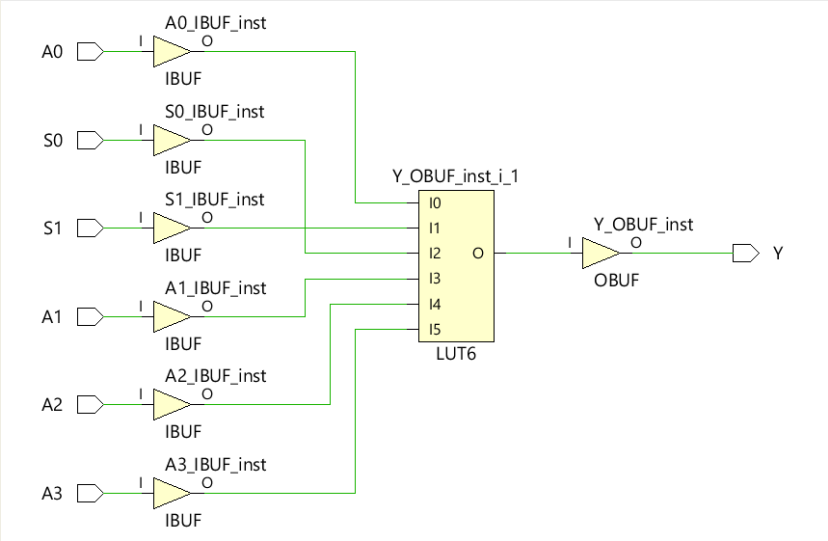
Στη λίστα ευαισθησίας συμπεριλαμβάνονται όλες
οι είσοδοι του συνδυαστικού κυκλώματος

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

- Σχηματικό διάγραμμα RTL

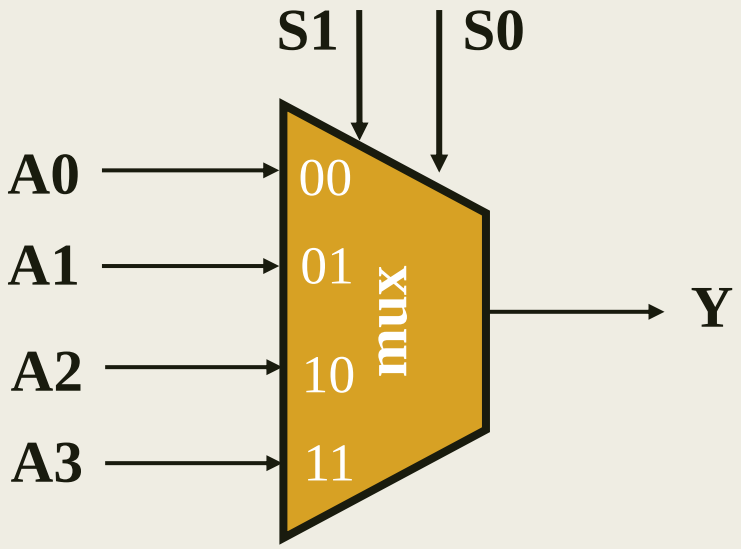


- Σχηματικό διάγραμμα σε τεχνολογία FPGA

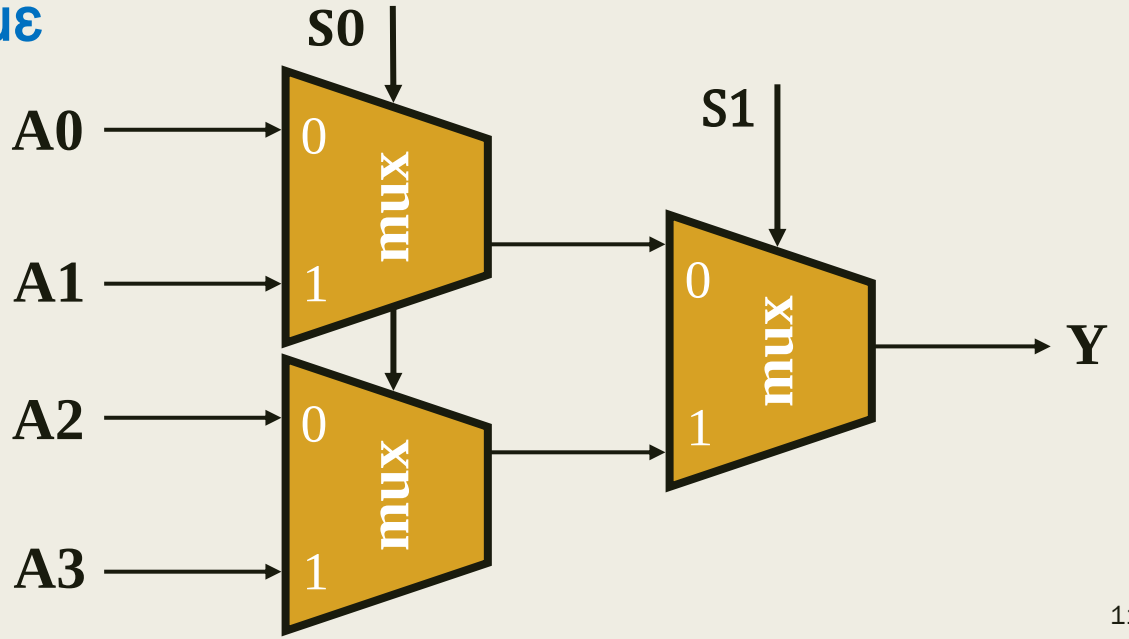


Η οντότητα του πολυπλέκτη 4 σε 1 στη VHDL

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



Λύση 2: Υλοποίηση με πολυπλέκτες 2 σε 1 σε δομή δένδρου



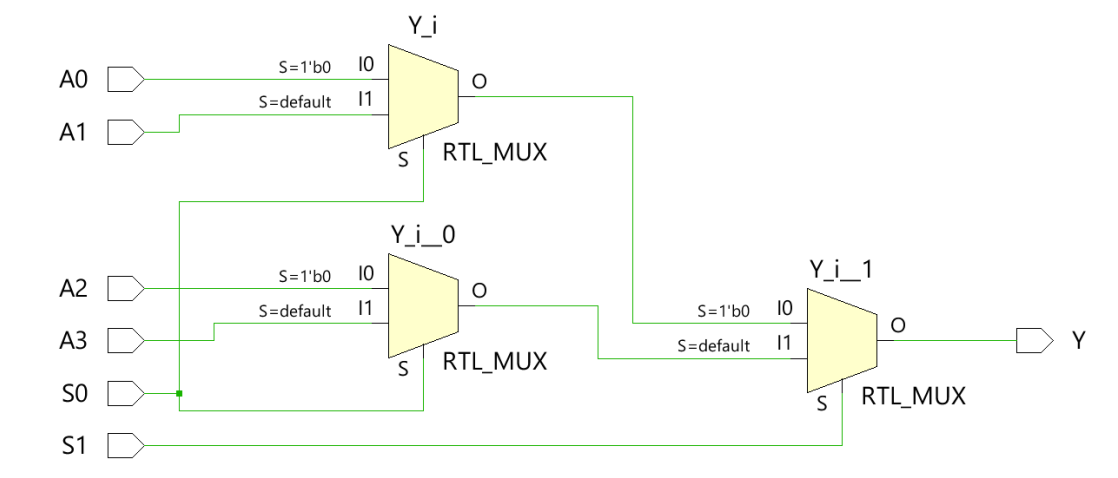
Η αρχιτεκτονική του πολυπλέκτη 4 σε 1 στη VHDL
Περιγραφή συμπεριφοράς – Λύση 2

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

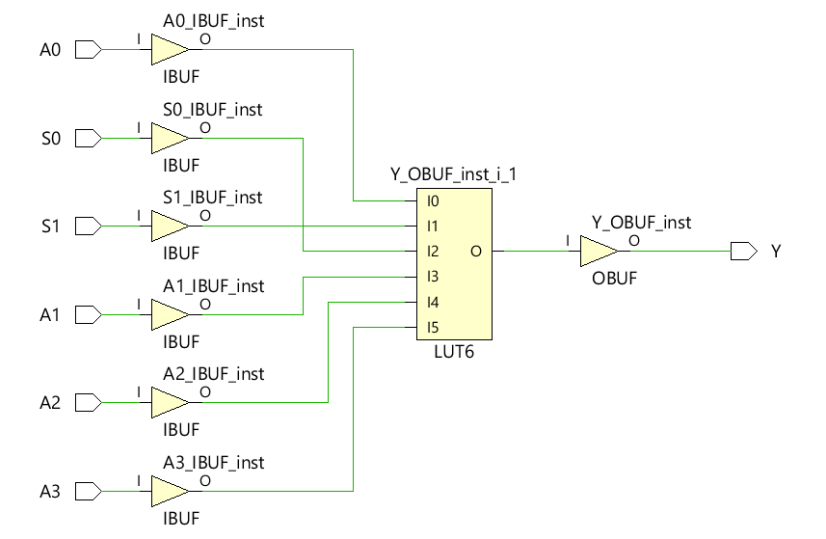
Στη λίστα ευαισθησίας συμπεριλαμβάνονται όλες οι είσοδοι του συνδυαστικού κυκλώματος

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

- Σχηματικό διάγραμμα RTL



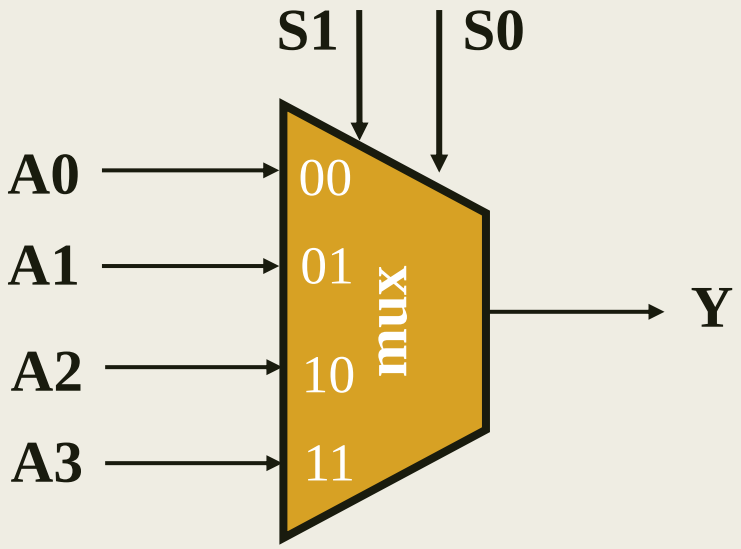
- Σχηματικό διάγραμμα σε τεχνολογία FPGA



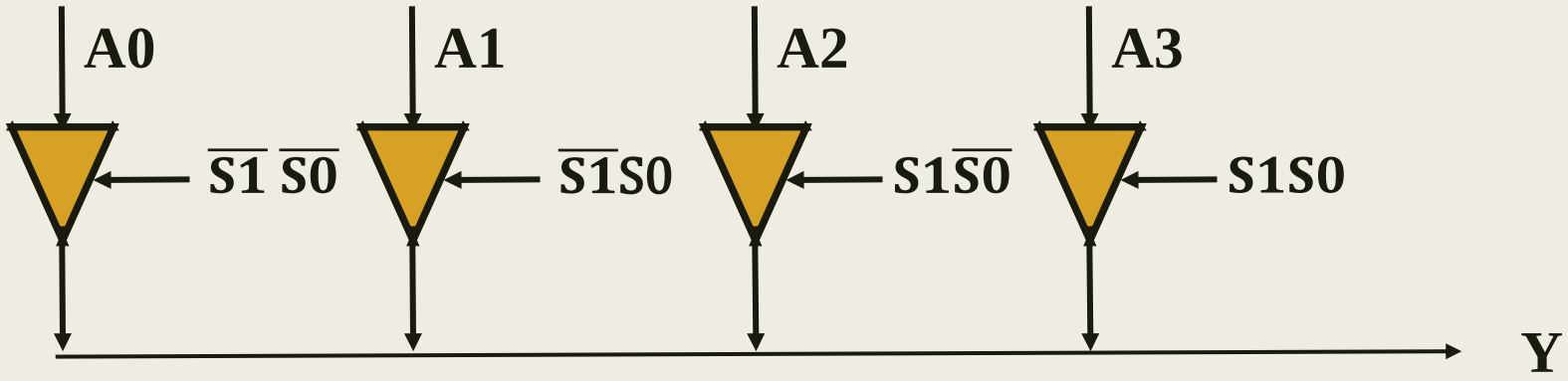
Και οι 2 λύσεις έχουν την ίδια υλοποίηση σε τεχνολογία FPGA με LUT6

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 3: Υλοποίηση με tri-state buffers



Η αρχιτεκτονική του πολυπλέκτη 4 σε 1 στη VHDL

Περιγραφή συμπεριφοράς – Λύση 3

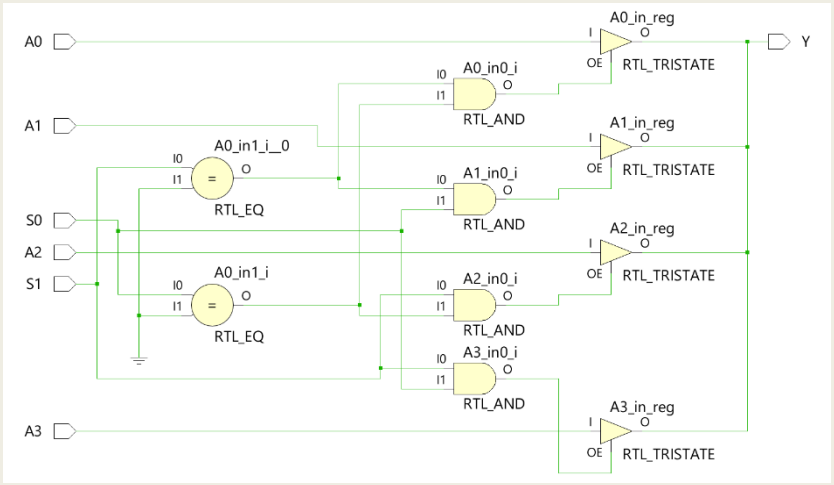
```
architecture BEHAVIORAL of MUX_4_to_1 is
signal A0_in, A1_in, A2_in, A3_in: STD_LOGIC;
begin
  BUFFERS0: process (A0, A1, A2, A3, S0, S1) begin
    if ((S1 = '0') and (S0 = '0'))
    then A0_in <= A0;
    else A0_in <= 'Z'; end if;
  end process;
  BUFFERS1: process (A0, A1, A2, A3, S0, S1) begin
    if ((S1 = '0') and (S0 = '1'))
    then A1_in <= A1;
    else A1_in <= 'Z'; end if;
  end process;
  BUFFERS2: process (A0, A1, A2, A3, S0, S1) begin
    if ((S1 = '1') and (S0 = '0'))
    then A2_in <= A2;
    else A2_in <= 'Z'; end if;
  end process;
  BUFFERS3: process (A0, A1, A2, A3, S0, S1) begin
    if ((S1 = '1') and (S0 = '1'))
    then A3_in <= A3;
    else A3_in <= 'Z'; end if;
  end process;
  Y <= A0_in; Y <= A1_in; Y <= A2_in; Y <= A3_in;
end BEHAVIORAL;
```

Ο tri-state buffer **πάντα** σε χωριστό process

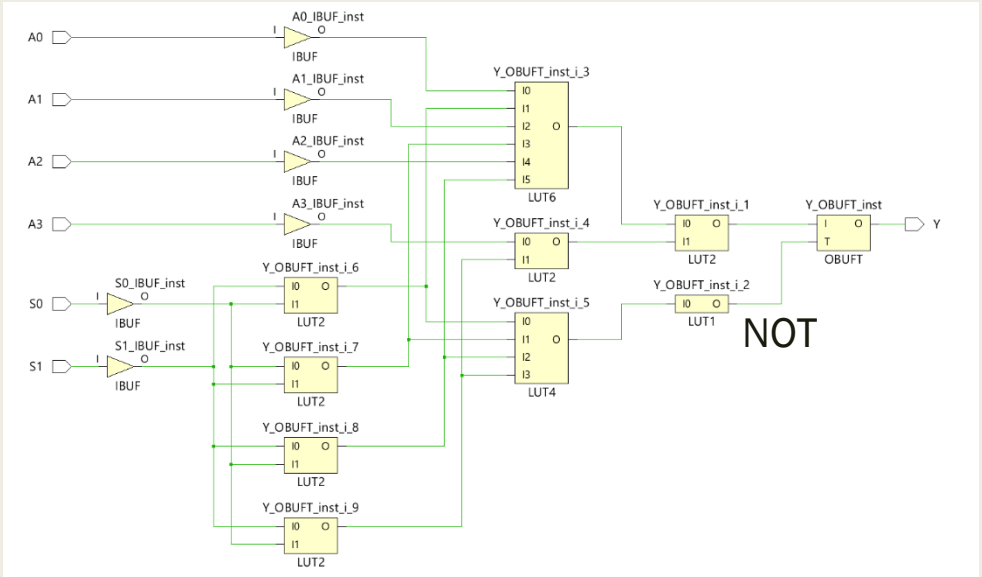
4 multi-source signals

Η αρχιτεκτονική του πολυπλέκτη 4 σε 1 στη VHDL
Περιγραφή συμπεριφοράς – Λύση 3

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA

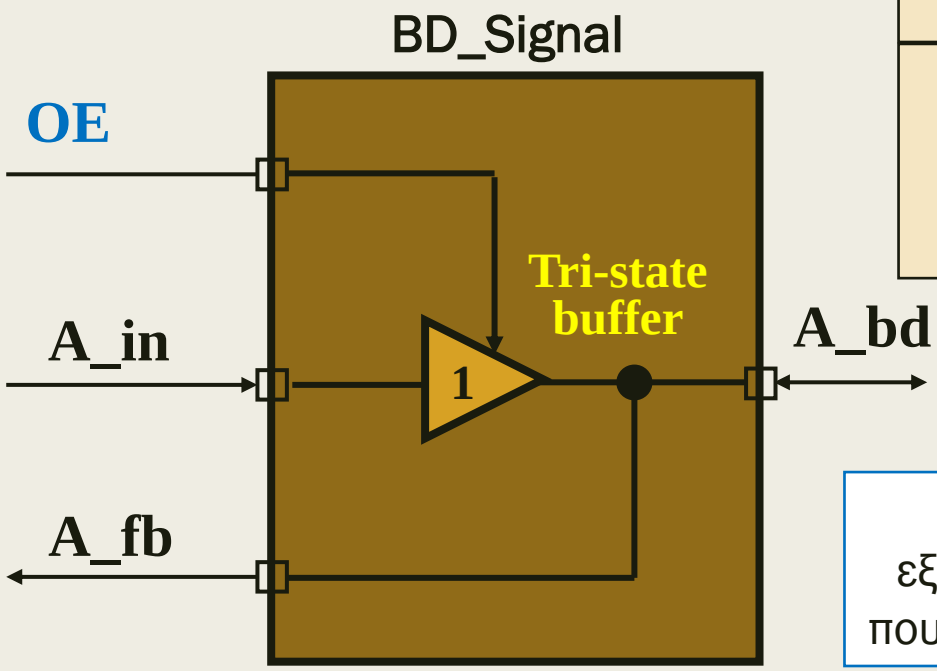


To **Tri-State Buffer (O)BUFT** υποστηρίζεται μόνο στα **OPADs** και **IOPADs** για σήματα που εγκαταλείπουν το chip

Οι **Tri-State Buffer** υλοποιούνται με λογική, αλλά παραμένει ένας **Tri-State Buffer (O)BUFT**, ώστε η έξοδος Y να μπορεί να γίνει **'Z'**

Το σήμα **Output Enable (T)** είναι **Active Low**

Αμφίδρομα (Bi-Directional) σήματα



OE	A_in	A_fb	A_bd
1	0	0	0
1	1	1	1
0	X	Z*	Z*

* Η τιμή αυτού του σήματος (0 ή 1) εξαρτάται από τον άλλο tri-state buffer που είναι συνδεδεμένος στο ίδιο port A_bd

Χρησιμοποιούνται σε ζεύγη

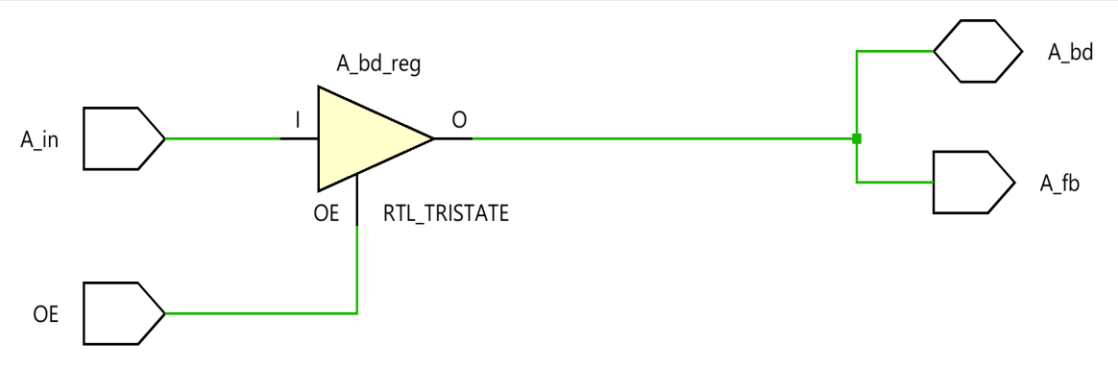
Υλοποίηση όταν υποστηρίζονται οι tri-state buffers στα IOB των FPGA

Αμφίδρομα (Bi-Directional) σήματα
Περιγραφή Συμπεριφοράς

```
entity BD_Signal is
  port (
    A_bd: inout STD_LOGIC;
    OE:   in    STD_LOGIC;
    A_in: in    STD_LOGIC;
    A_fb: out   STD_LOGIC);
end BD_Signal;
architecture BEHAVIORAL of BD_Signal is
begin
  BD_BUFFER: process (OE, A_in) begin
    if (OE = '1')
    then A_bd <= A_in;
    else A_bd <= 'Z';
    end if;
  end process;
  A_fb <= A_bd;
end BEHAVIORAL;
```

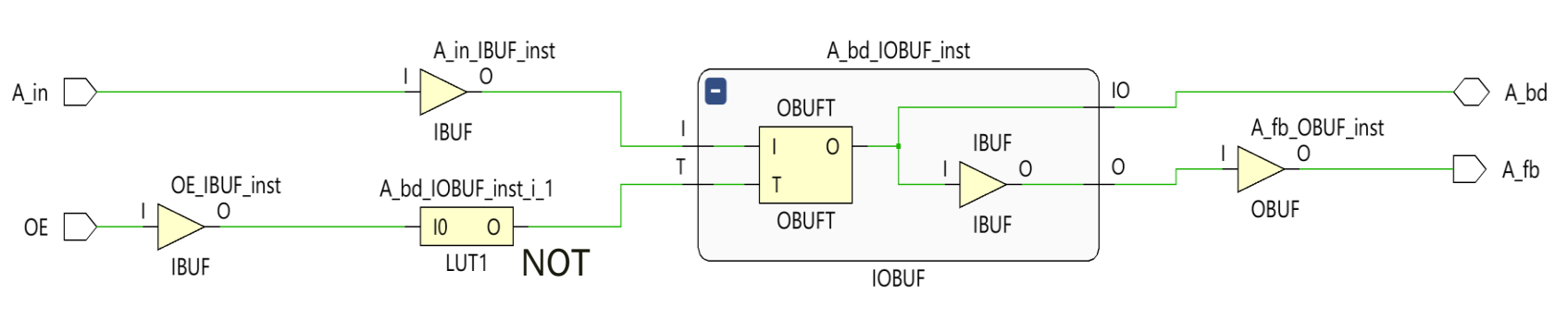
Αμφίδρομα (Bi-Directional) σήματα
Περιγραφή Συμπεριφοράς

- Σχηματικό διάγραμμα RTL



Το Tri-State Buffer (O)BUFT υποστηρίζεται μόνο στα OPADs και IOPADs για σήματα που εγκαταλείπουν το chip

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Το σήμα Output Enable (T) είναι Active Low

Επιλεγμένη άσκηση: συνδυαστική λογική σε περιγραφή συμπεριφοράς στη VHDL

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

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

Επιλεγμένη άσκηση: συνδυαστική λογική σε περιγραφή συμπεριφοράς στη VHDL

- Πίνακας αλήθειας

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

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

ABCD	Y
1000	
1001	
1010	
1011	
1100	
1101	
1110	
1111	

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

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

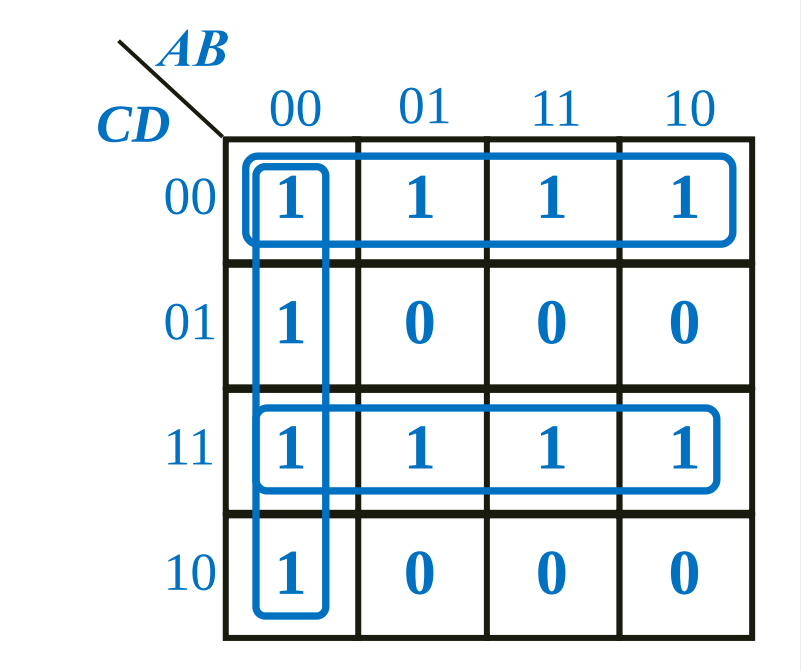
```
elsif (C = D) then Y <= '1';
```

Επιλεγμένη άσκηση: συνδυαστική λογική σε περιγραφή συμπεριφοράς στη VHDL

- Πίνακας αλήθειας

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

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



- Εξίσωση Boole

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

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση σημάτων

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

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση εσωτερικών σημάτων

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

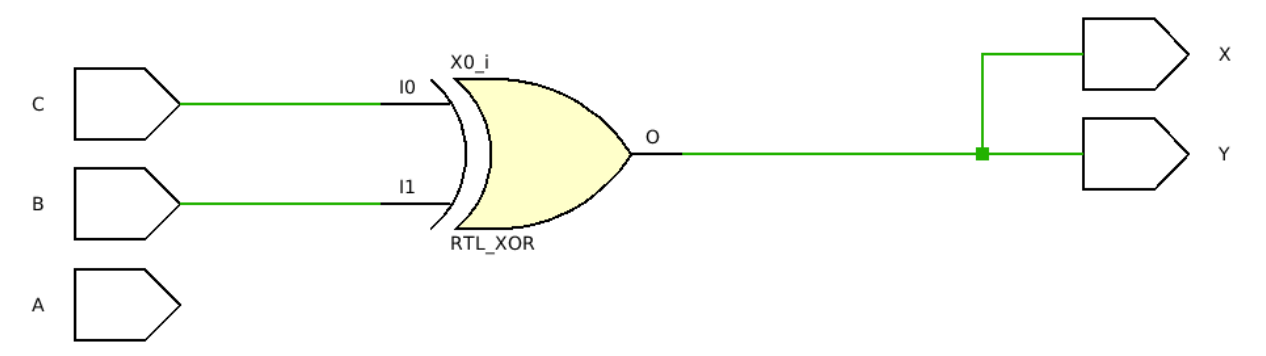
Ανάθεση τιμών στο εσωτερικό σήμα D με δύο εντολές

Το εσωτερικό σήμα D δηλώνεται στη λίστα ευαισθησίας

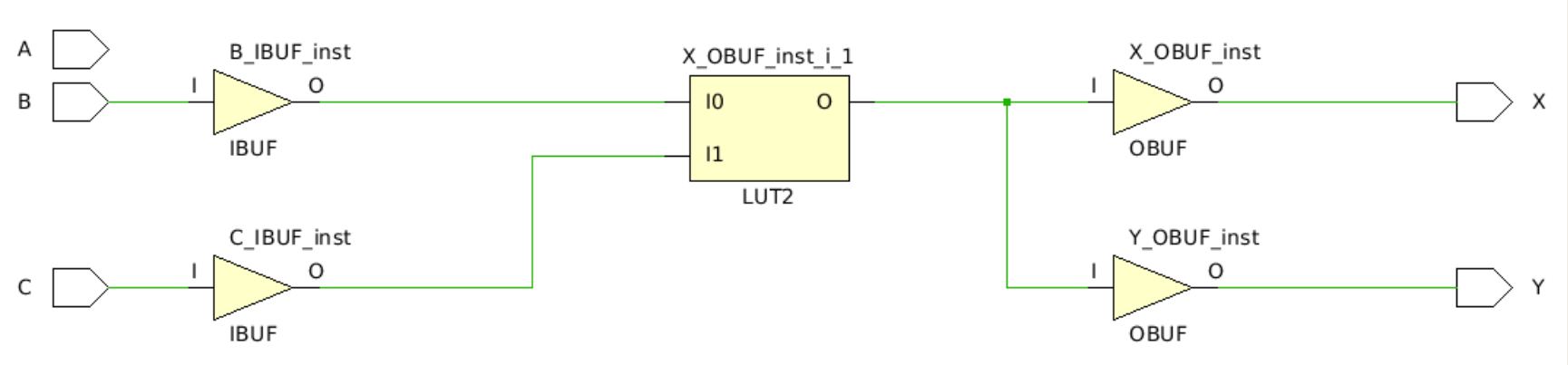
- Κατά τη σύνθεση υλοποιείται η εξίσωση Boole:
  - *X = Y = C xor B*

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση εσωτερικών σημάτων

- Σχηματικό διάγραμμα RTL

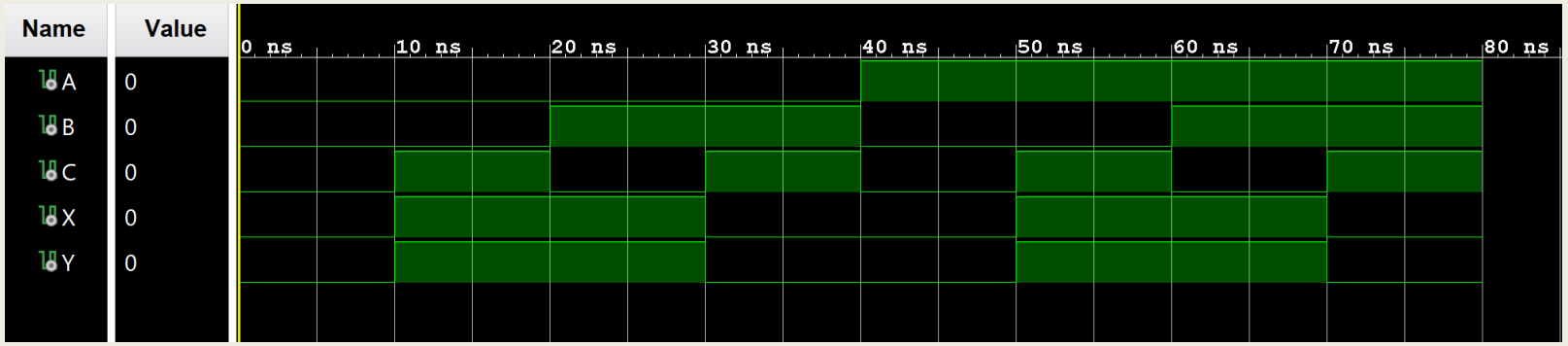


- Σχηματικό διάγραμμα σε τεχνολογία FPGA

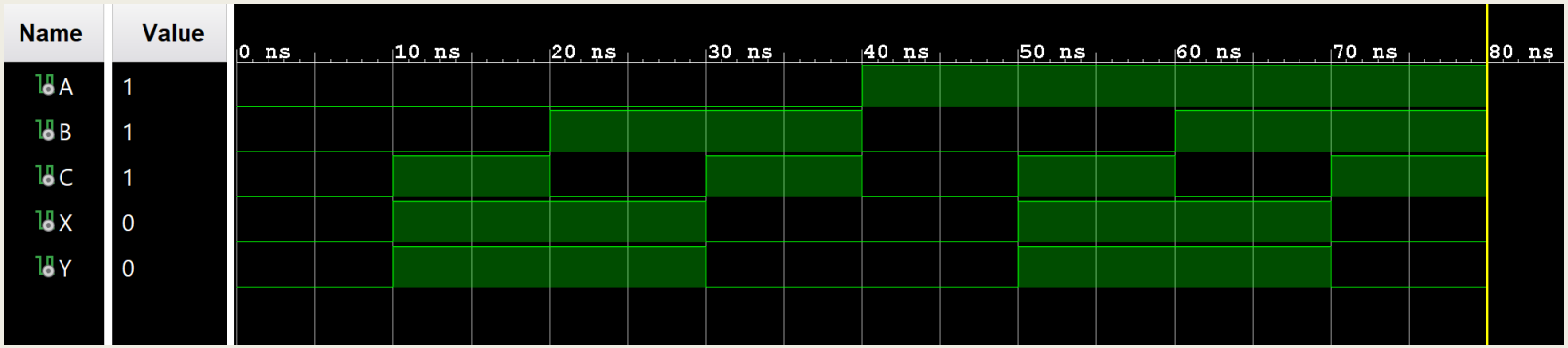


Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση εσωτερικών σημάτων

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



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



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

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση εσωτερικών σημάτων

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

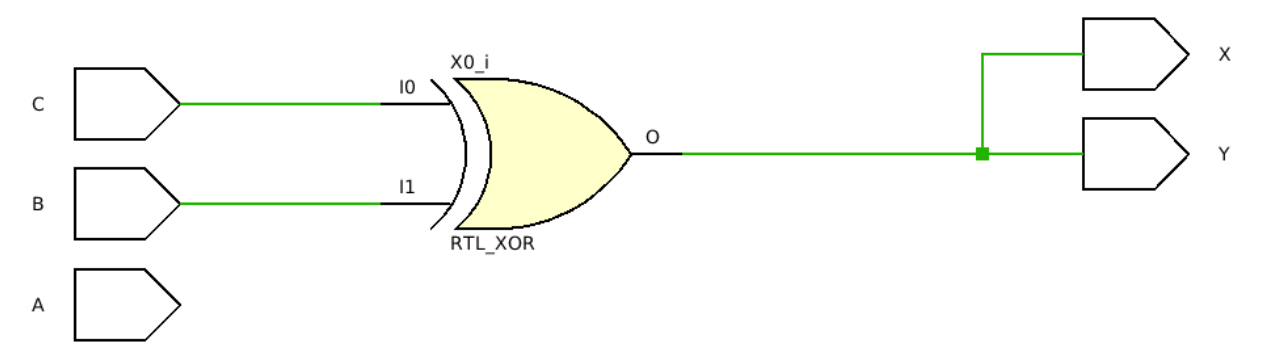
Ανάθεση τιμών στο εσωτερικό σήμα D με δύο εντολές

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

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

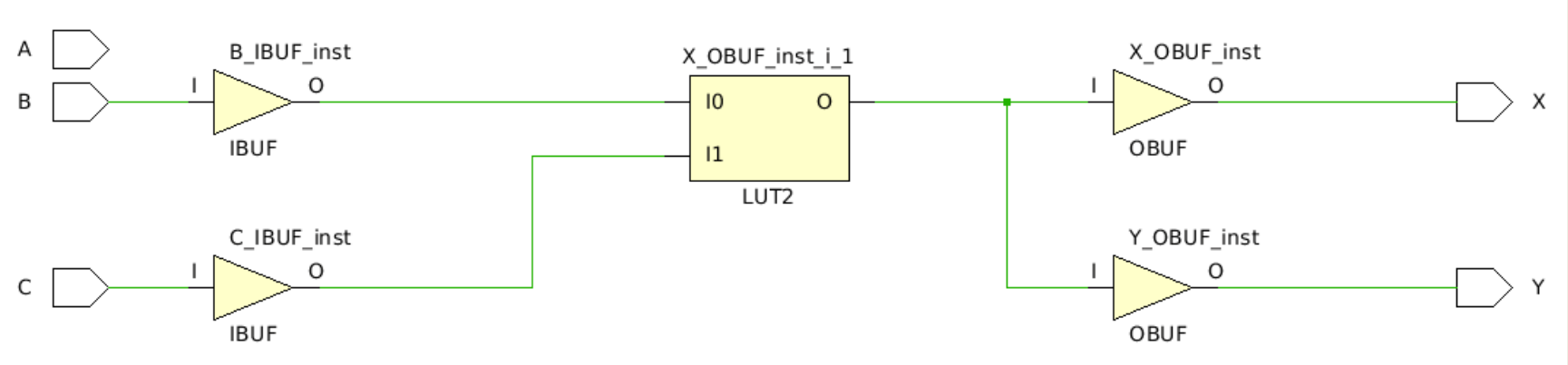
Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση εσωτερικών σημάτων

- Σχηματικό διάγραμμα RTL



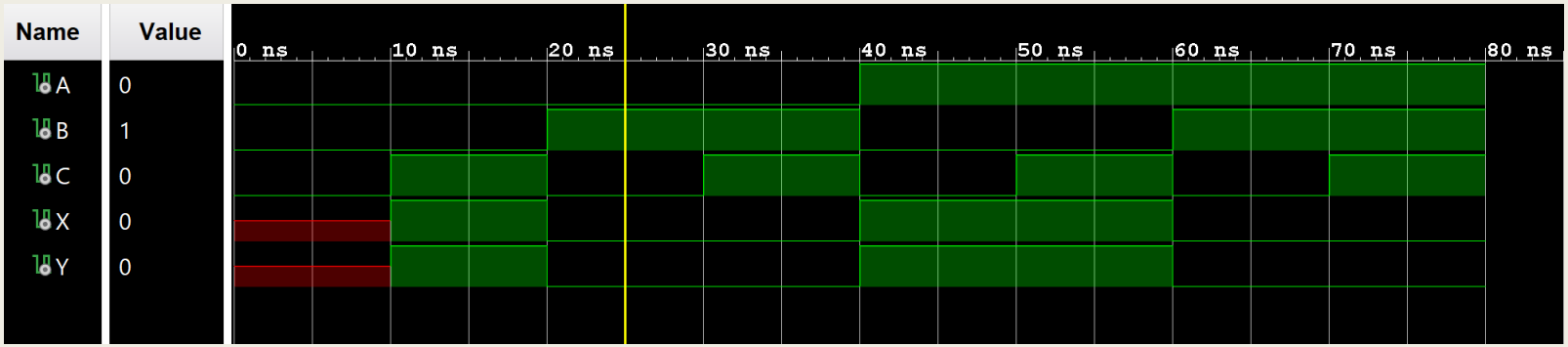
Η σύνθεση και η υλοποίηση γίνονται σωστά

- Σχηματικό διάγραμμα σε τεχνολογία FPGA

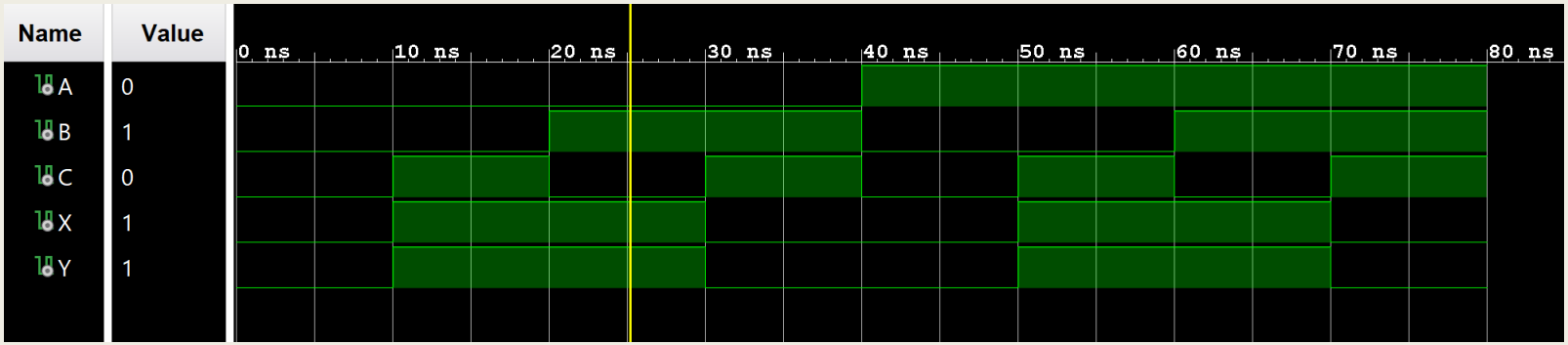


Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση εσωτερικών σημάτων

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



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



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

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση μεταβλητών

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

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση μεταβλητών

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

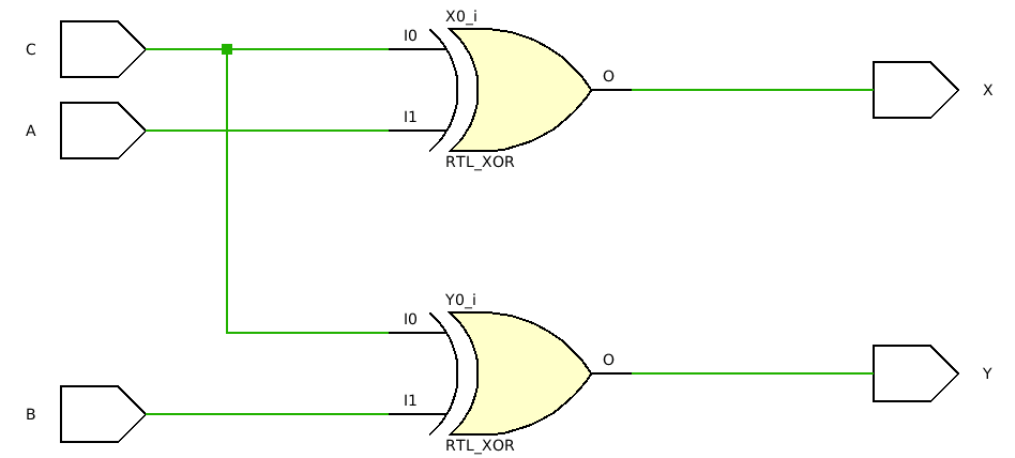
Η μεταβλητή D λαμβάνει άμεσα την τιμή της εισόδου A

Η μεταβλητή D λαμβάνει άμεσα την τιμή της εισόδου B

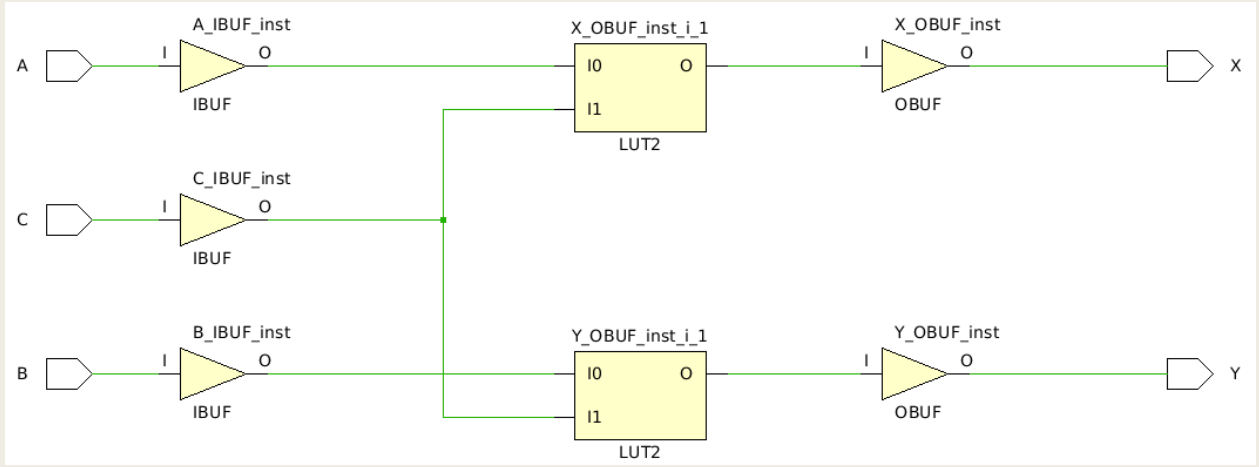
- Κατά τη σύνθεση υλοποιούνται οι εξισώσεις Boole:
  - *X = C xor A* και *Y = C xor B*

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση μεταβλητών

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση σημάτων

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

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

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

  end process;
end LAST_BEH;
```

Ανάθεση τιμών στο σήμα εξόδου C με δύο εντολές IF

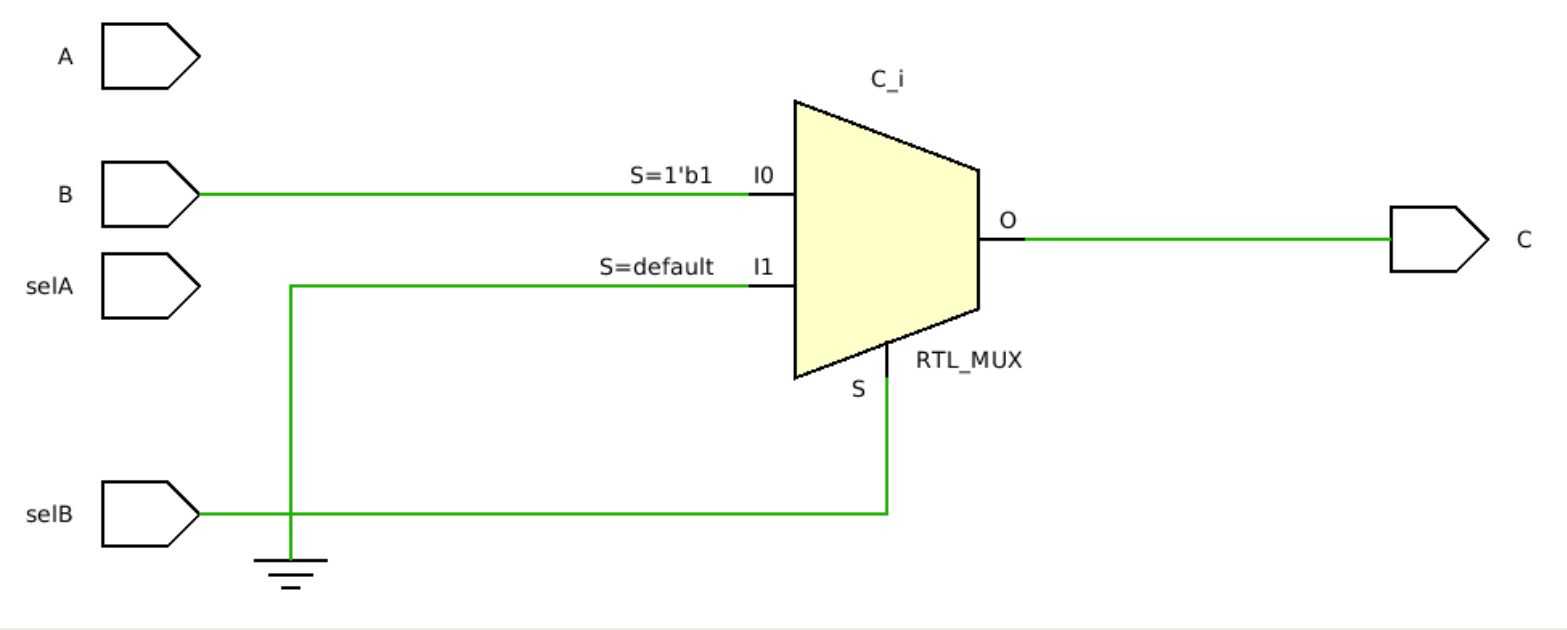
Δεν λαμβάνεται υπόψη

Λαμβάνεται υπόψη

- Ποιο είναι το αποτέλεσμα της σύνθεσης;

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση σημάτων

- Σχηματικό διάγραμμα RTL



- Εξίσωση Boole που υλοποιείται

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

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση σημάτων

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

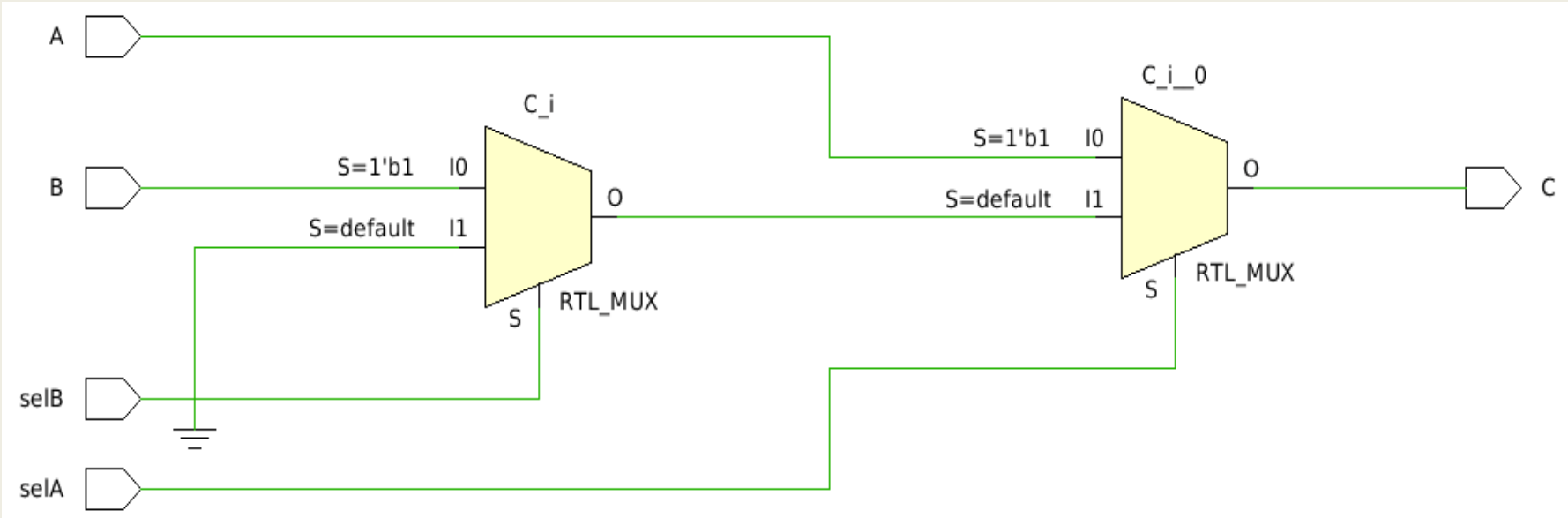
```
entity LAST is
  port (
    A, B, selA, selB: in STD_LOGIC;
    C: out STD_LOGIC);
end LAST;
architecture LAST_BEH of LAST is
begin
  process (A, B, selA, selB)
  begin
    if (selA = '1') then
      C <= A;
    elsif (selB = '1') then
      C <= B;
    else
      C <= '0';
    end if;
  end process;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

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

- Ποιο είναι το αποτέλεσμα της σύνθεσης μετά την τροποποίηση;

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Χρήση σημάτων

- Σχηματικό διάγραμμα RTL



- Εξίσωση Boole που υλοποιείται

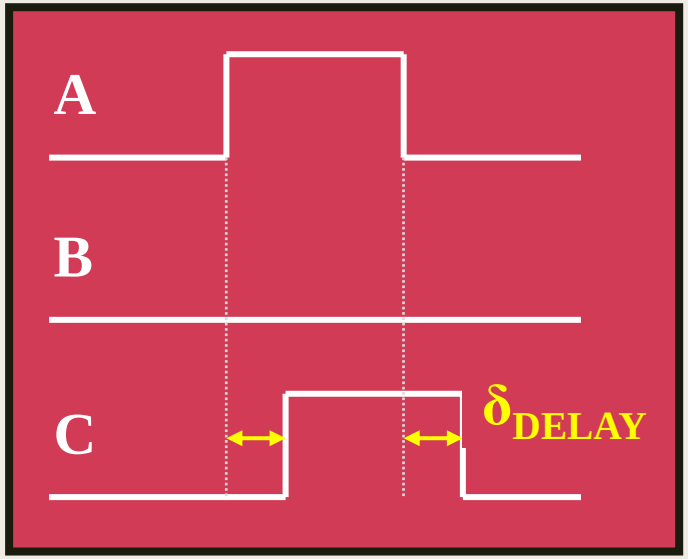
$$C = (B \text{ and } selB \text{ and } \overline{selA}) \text{ or } (A \text{ and } selA)$$

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

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

Επιλεγμένες ασκήσεις

- Ποιο είναι η εξίσωση Boole που υλοποιείται πριν και μετά τη σύνθεση;
- Ποιο είναι το αποτέλεσμα της προσομοίωσης πριν (behavioral simulation) και μετά τη σύνθεση (post simulation) της οντότητας SD1;
  - *Να εξετάσετε εάν συμφωνούν οι δύο προσομοιώσεις και τί πρέπει να αλλάξει στον κώδικά, ώστε να συμφωνήσουν*

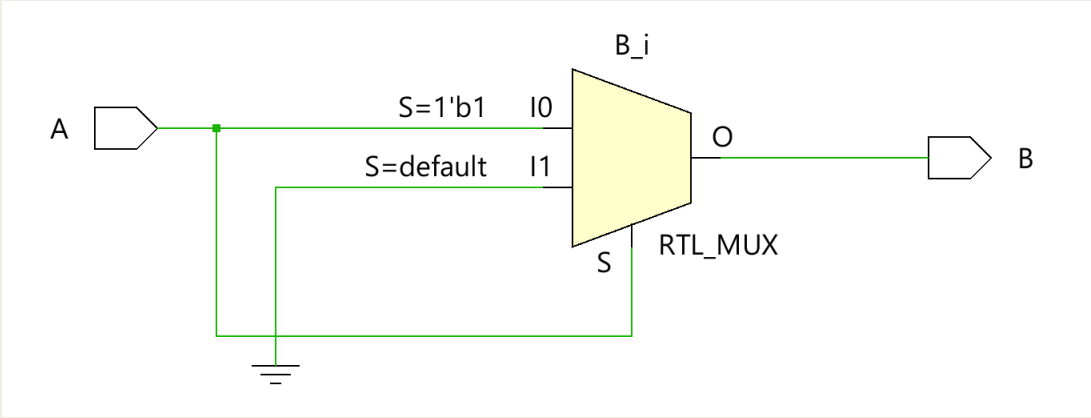



```
entity SD1 is port (
  A: in STD_LOGIC;
  B: out STD_LOGIC);
end SD1;
architecture SD1_BEH of SD1 is
signal C: STD_LOGIC;
begin
  process (A)
  begin
    C <= A;
    if (C = '1') then B <= A;
    else B <= '0';
    end if;
  end process;
end SD1_BEH;
```

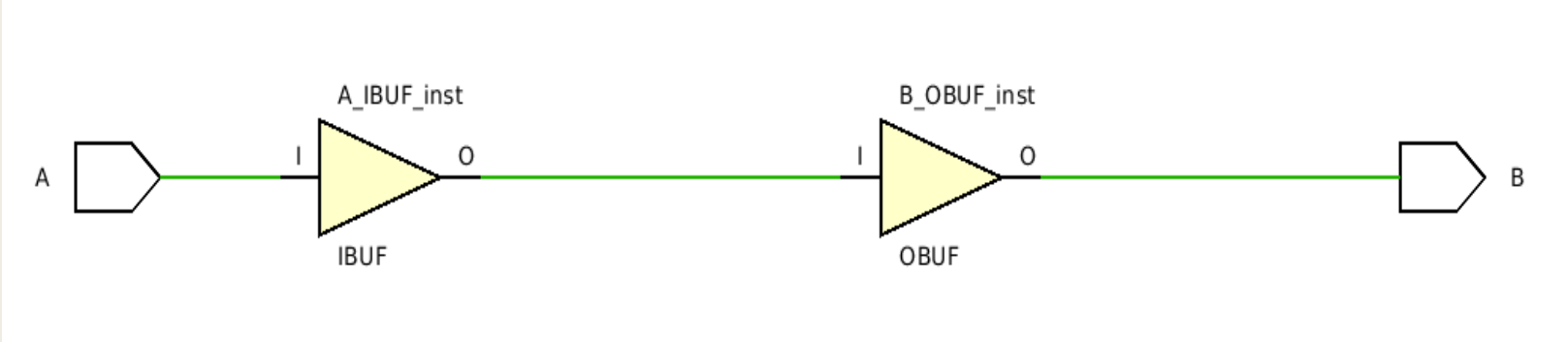
Η ενημέρωση των σημάτων B και C γίνεται στο τέλος της διεργασίας με δ_{delay}.

Επιλεγμένες ασκήσεις

- Πριν τη σύνθεση υλοποιείται η εξίσωση Boole $\mathbf{B = A\,A + \overline{A}\,0 = A}$
  - *Σχηματικό διάγραμμα RTL*

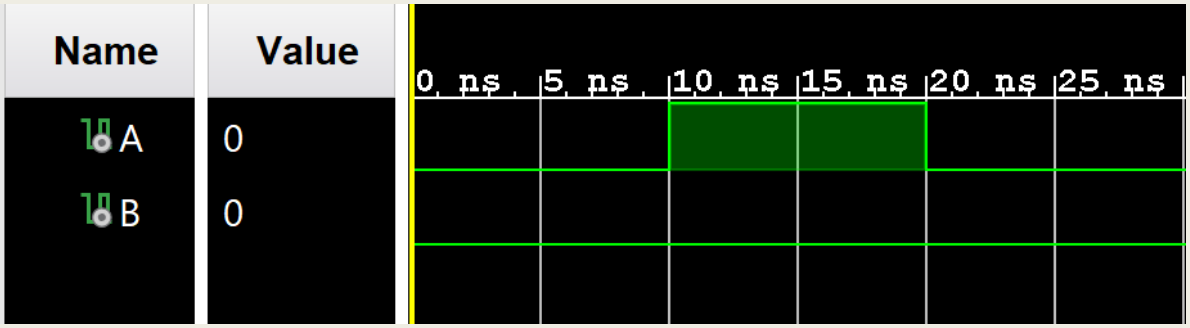


- Μετά τη σύνθεση υλοποιείται η εξίσωση Boole $\mathbf{B = A}$
  - *Σχηματικό διάγραμμα σε τεχνολογία FPGA*

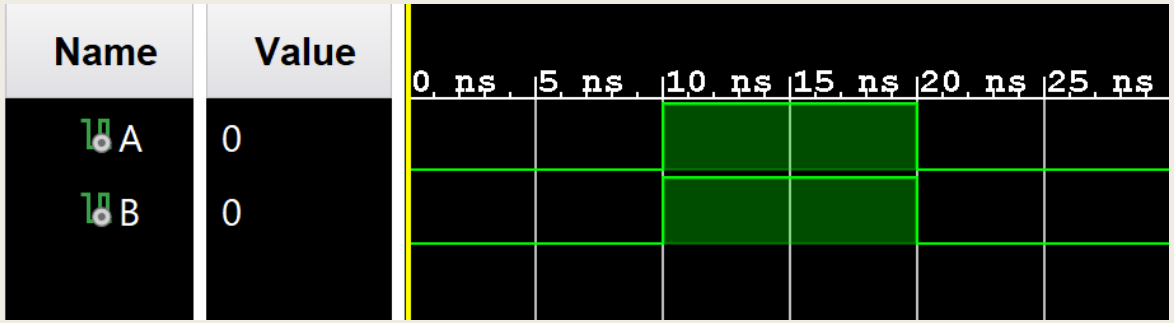


Επιλεγμένες ασκήσεις

- Πριν τη σύνθεση υλοποιείται η εξίσωση Boole $\mathbf{B = 0}$
  - *Προσομοίωση behavioral model*



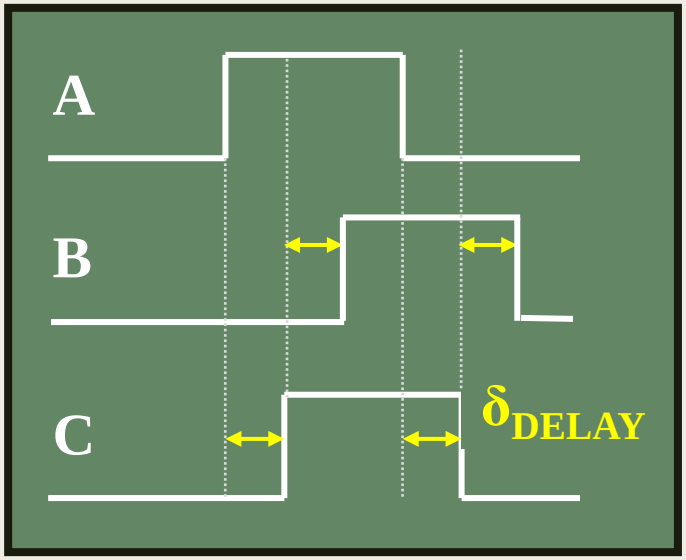
- Μετά τη σύνθεση υλοποιείται η εξίσωση Boole $\mathbf{B = A}$
  - *Προσομοίωση post functional simulation model*



Η ασυμφωνία των προσομοιώσεων πριν και μετά τη σύνθεση οφείλεται στην ελλιπή δήλωση του εσωτερικού σήματος C στη λίστα ευαισθησίας

Επιλεγμένες ασκήσεις

- Ποιο είναι η εξίσωση Boole που υλοποιείται πριν και μετά τη σύνθεση;
- Ποιο είναι το αποτέλεσμα της προσομοίωσης πριν (behavioral simulation) και μετά τη σύνθεση (post simulation) της οντότητας SD2;
  - *Να εξετάσετε εάν συμφωνούν οι δύο προσομοιώσεις και τί πρέπει να αλλάξει στον κώδικά, ώστε να συμφωνήσουν*

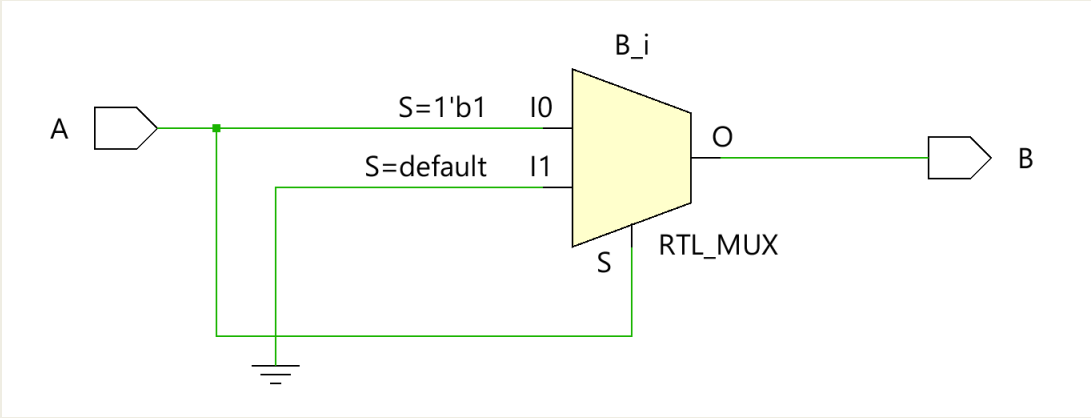



```
entity SD2 is port (
  A: in STD_LOGIC;
  B: out STD_LOGIC);
end SD2;
architecture SD2_BEH of SD2 is
signal C: STD_LOGIC;
begin
  process (A, C)
  begin
    C <= A;
    if (C = '1') then B <= A;
    else B <= '0';
    end if;
  end process;
end SD2_BEH;
```

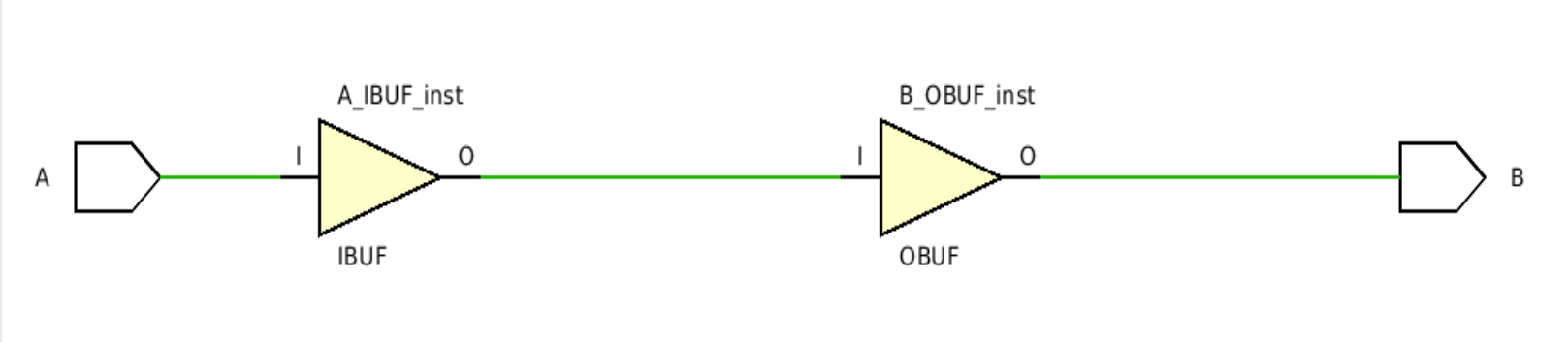
Η ενημέρωση των σημάτων B και C γίνεται στο τέλος της διεργασίας με δ_{delay}.

Επιλεγμένες ασκήσεις

- Πριν τη σύνθεση υλοποιείται η εξίσωση Boole $\mathbf{B = A\,A + \overline{A}\,0 = A}$
  - *Σχηματικό διάγραμμα RTL*

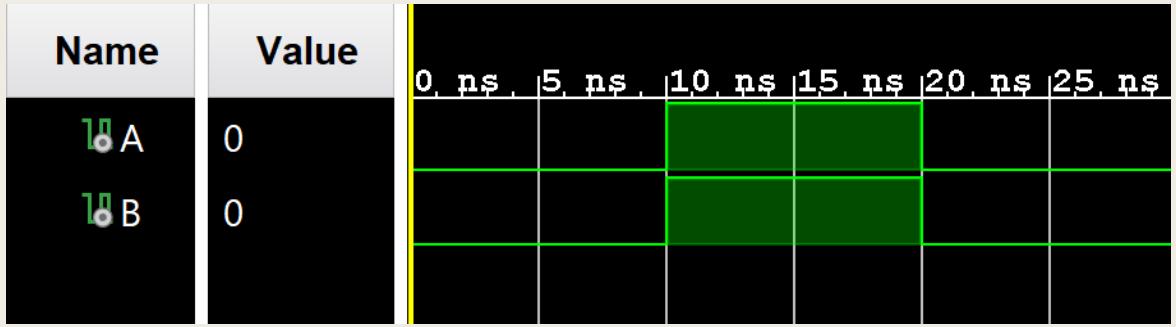


- Μετά τη σύνθεση υλοποιείται η εξίσωση Boole $\mathbf{B = A}$
  - *Σχηματικό διάγραμμα σε τεχνολογία FPGA*

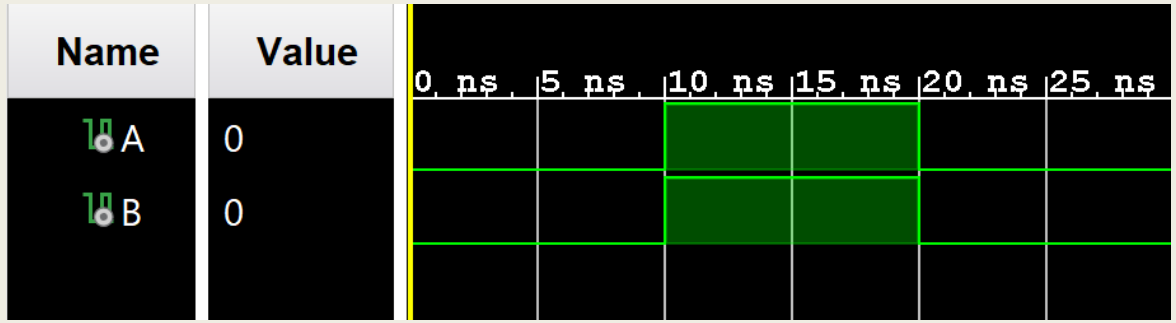


Επιλεγμένες ασκήσεις

- Πριν τη σύνθεση υλοποιείται η εξίσωση Boole $\mathbf{B = A}$
  - *Προσομοίωση behavioral model*



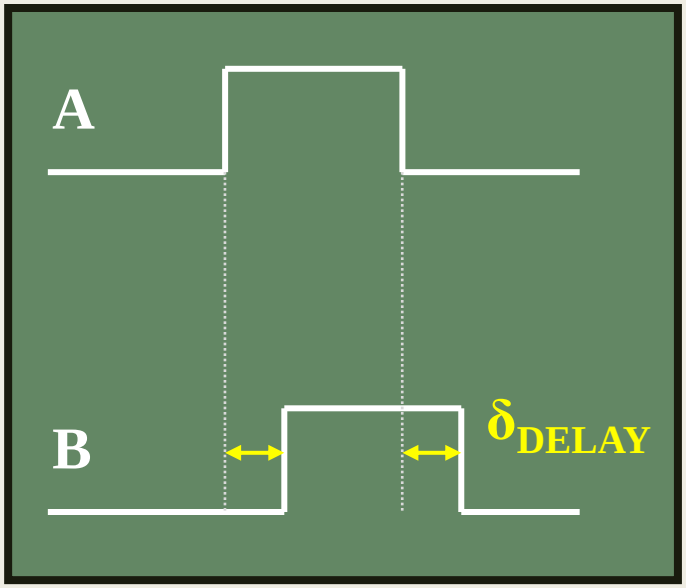
- Μετά τη σύνθεση υλοποιείται η εξίσωση Boole $\mathbf{B = A}$
  - *Προσομοίωση post functional simulation model*



Οι προσομοιώσεις πλέον συμφωνούν

Επιλεγμένες ασκήσεις

- Ποιο είναι η εξίσωση Boole που υλοποιείται πριν και μετά τη σύνθεση;
- Ποιο είναι το αποτέλεσμα της προσομοίωσης πριν (behavioral simulation) και μετά τη σύνθεση (post simulation) της οντότητας E1;
  - *Να εξετάσετε εάν συμφωνούν οι δύο προσομοιώσεις και τί πρέπει να αλλάξει στον κώδικά, ώστε να συμφωνήσουν*

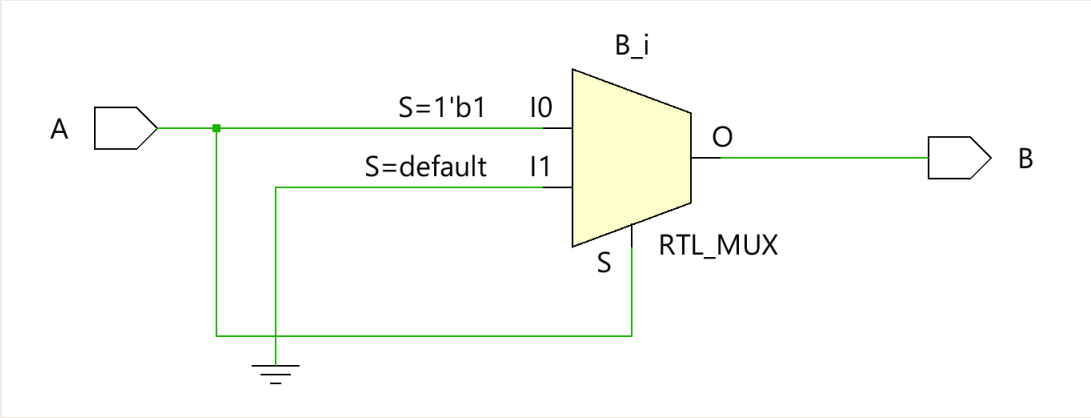



```
entity E1 is port (
  A: in STD_LOGIC;
  B: out STD_LOGIC);
end E1;
architecture E1_BEH of E1 is
begin
  process (A)
  variable C: STD_LOGIC;
  begin
    C := A;
    if (C = '1') then B <= A;
    else B <= '0';
    end if;
  end process;
end E1_BEH;
```

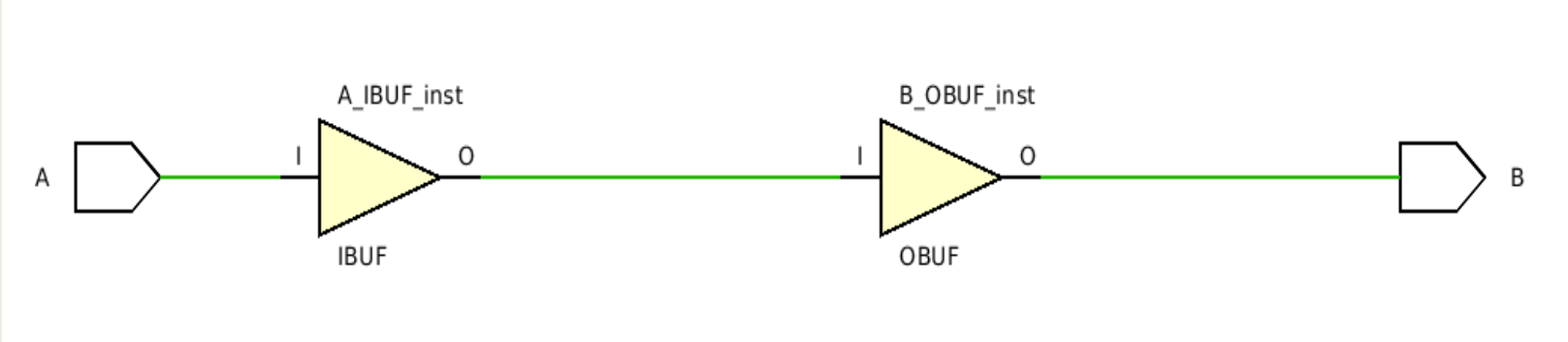
Η ενημέρωση της μεταβλητής C γίνεται αμέσως μόλις εκτελεσθεί η εντολή ανάθεσης τιμής

Επιλεγμένες ασκήσεις

- Πριν τη σύνθεση υλοποιείται η εξίσωση Boole $\mathbf{B = A\,A + \overline{A}\,0 = A}$
  - *Σχηματικό διάγραμμα RTL*

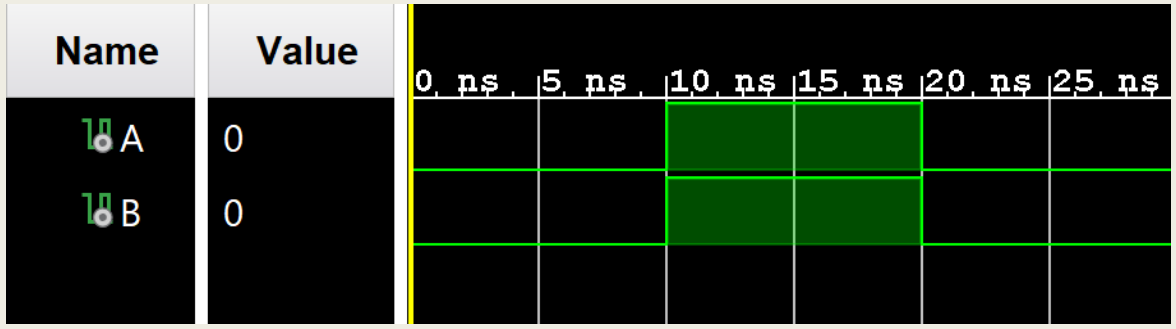


- Μετά τη σύνθεση υλοποιείται η εξίσωση Boole $\mathbf{B = A}$
  - *Σχηματικό διάγραμμα σε τεχνολογία FPGA*

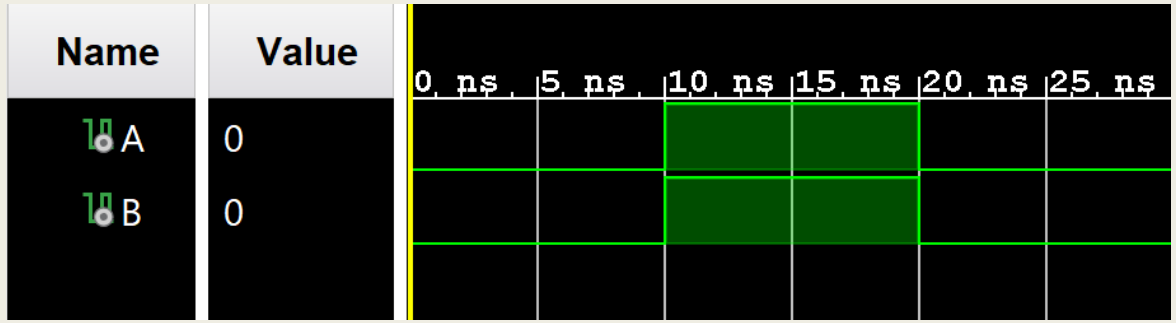


Επιλεγμένες ασκήσεις

- Πριν τη σύνθεση υλοποιείται η εξίσωση Boole $\mathbf{B = A}$
  - *Προσομοίωση behavioral model*



- Μετά τη σύνθεση υλοποιείται η εξίσωση Boole $\mathbf{B = A}$
  - *Προσομοίωση post functional simulation model*



Οι προσομοιώσεις συμφωνούν

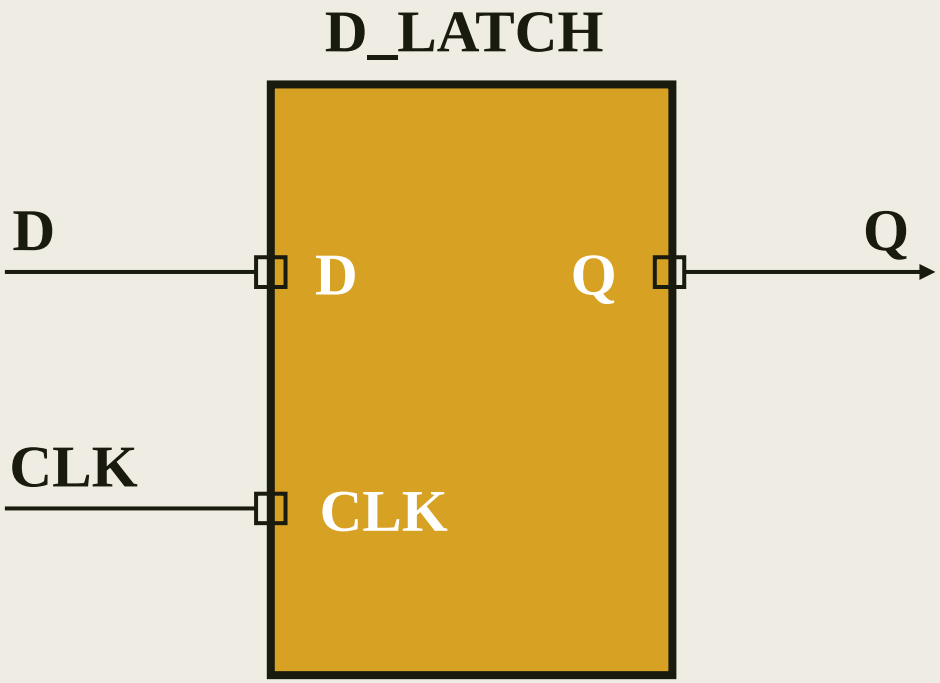
Περιγραφή ακολουθιακής λογικής στη VHDL

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

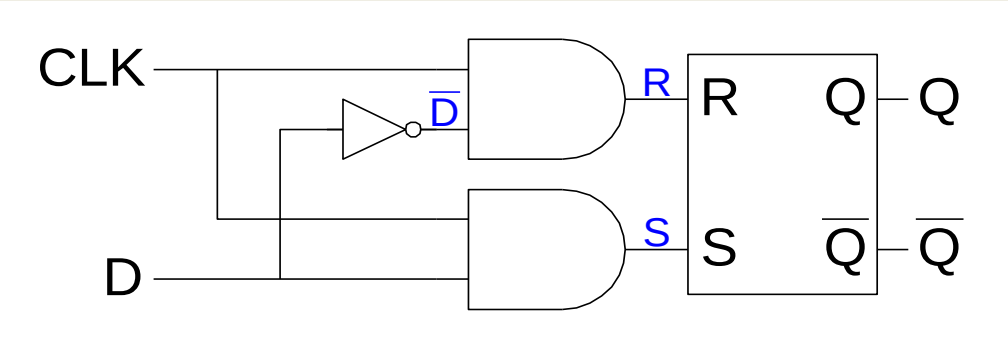
```
if boolean_expression (condition) then
    sequential_statement_1;
end if;
```

Η οντότητα του D Latch στη VHDL

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



Η αρχιτεκτονική του D Latch στη VHDL
Περιγραφή συμπεριφοράς



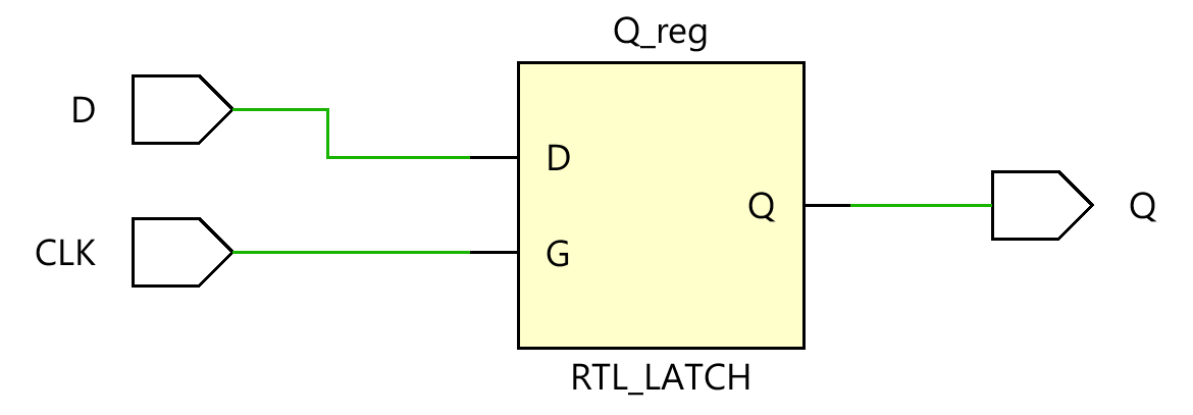
CLK	D	Q	$\overline{Q}$
0	X	Q_{prev}	$\overline{Q}_{prev}$
1	0	0	1
1	1	1	0

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

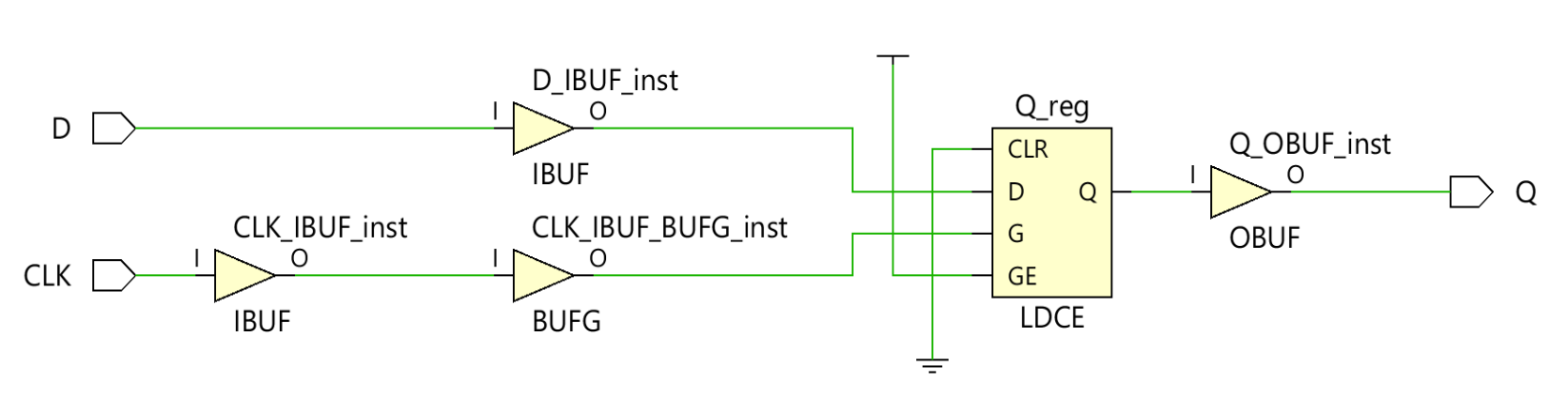
Εκμεταλλευόμαστε την εσωτερική μνήμη των σημάτων

Η αρχιτεκτονική του D Latch στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



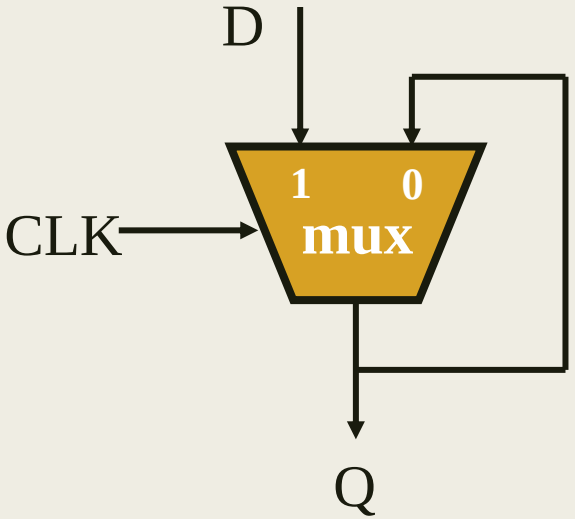
Το πρόβλημα της ελλιπούς ανάθεσης τιμής στη VHDL

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

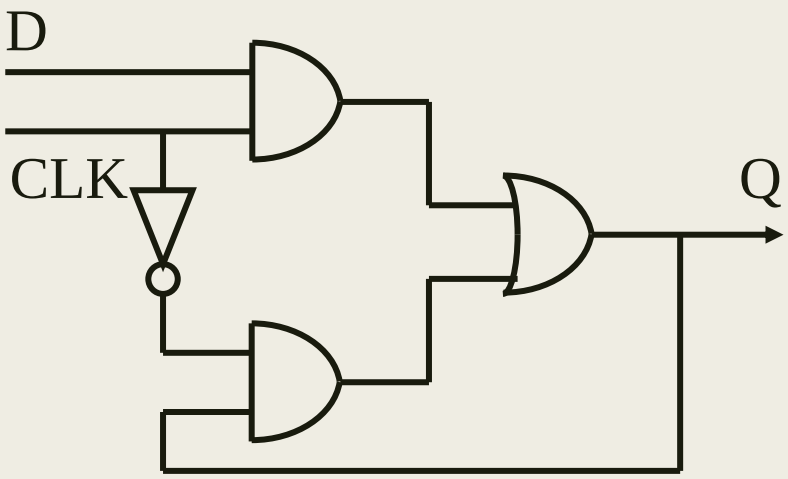
Υλοποίηση ελλιπούς ανάθεσης τιμής
Ακολουθιακή λογική

D-Latch

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



Σε επίπεδο πολυπλέκτη



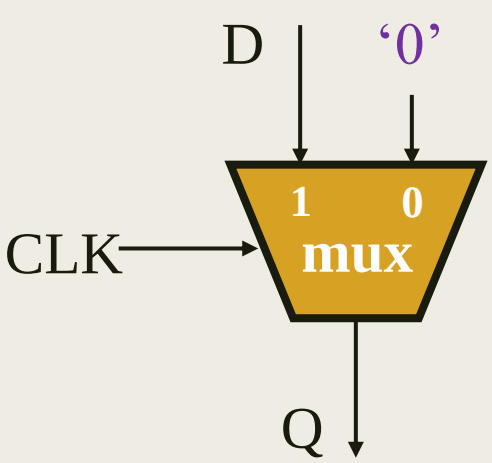
Σε επίπεδο πύλης

Υλοποίηση μη ελλιπούς ανάθεσης τιμής

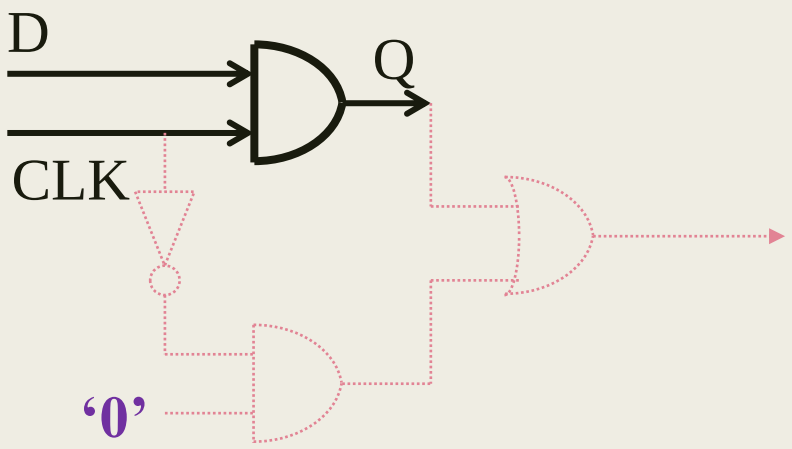
Συνδυαστική λογική

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

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



Σε επίπεδο πολυπλέκτη



Σε επίπεδο πύλης

Υλοποίηση μη ελλιπούς ανάθεσης τιμής

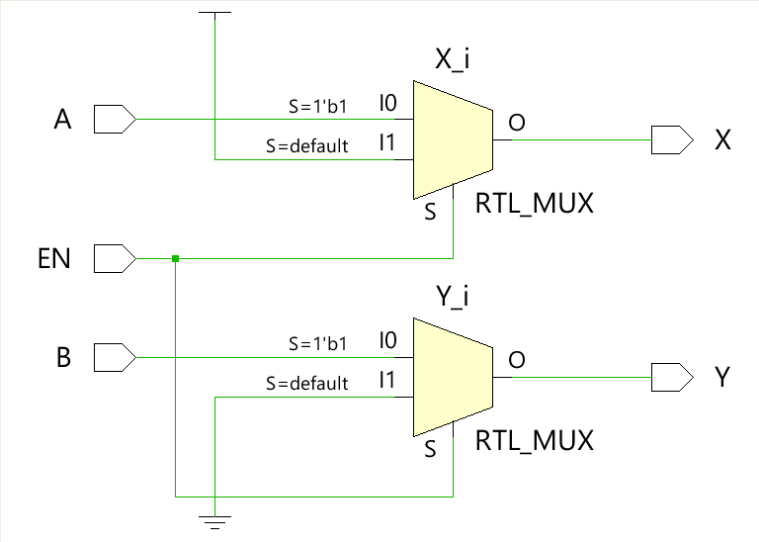
- Παράδειγμα αποφυγής εμφάνισης ελλιπούς ανάθεσης τιμής

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

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

Υλοποίηση μη ελλιπούς ανάθεσης τιμής

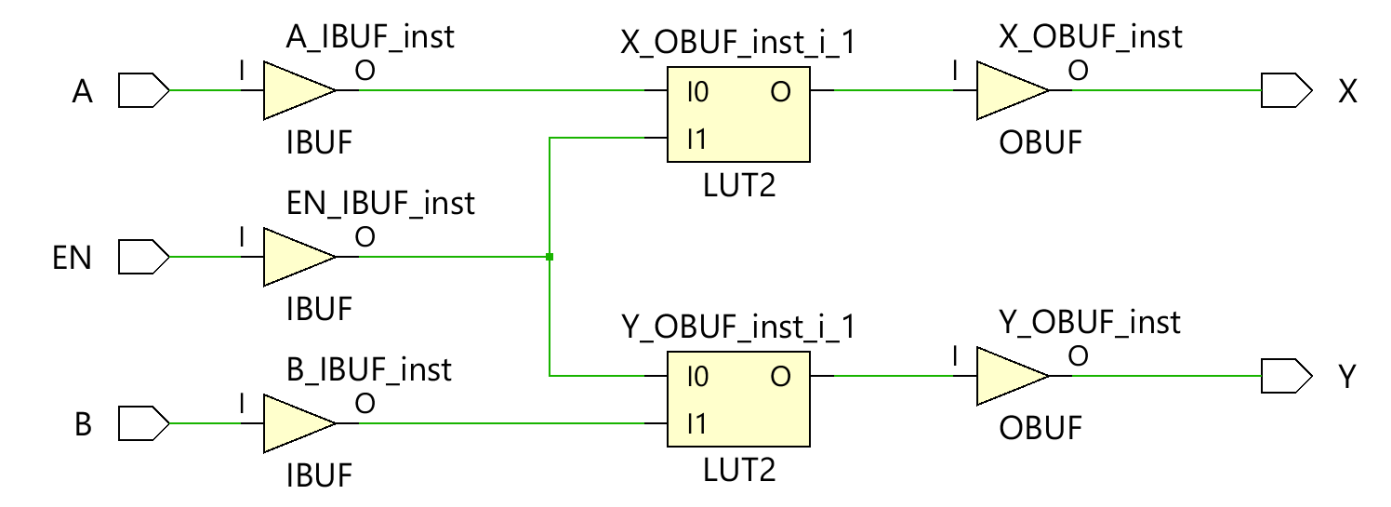
- Σχηματικό διάγραμμα RTL



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

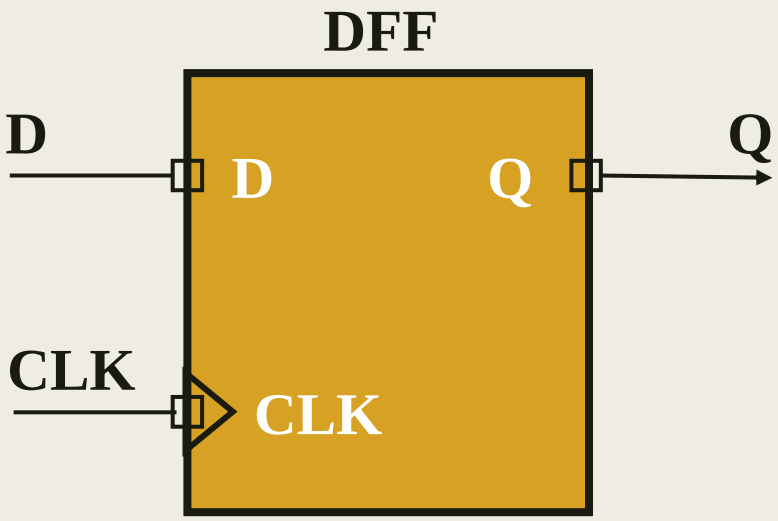
$$Y = ENB + \overline{EN}\,0 = ENB$$

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Η οντότητα του D Flip-Flop στη VHDL

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



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

Η αρχιτεκτονική του D Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

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

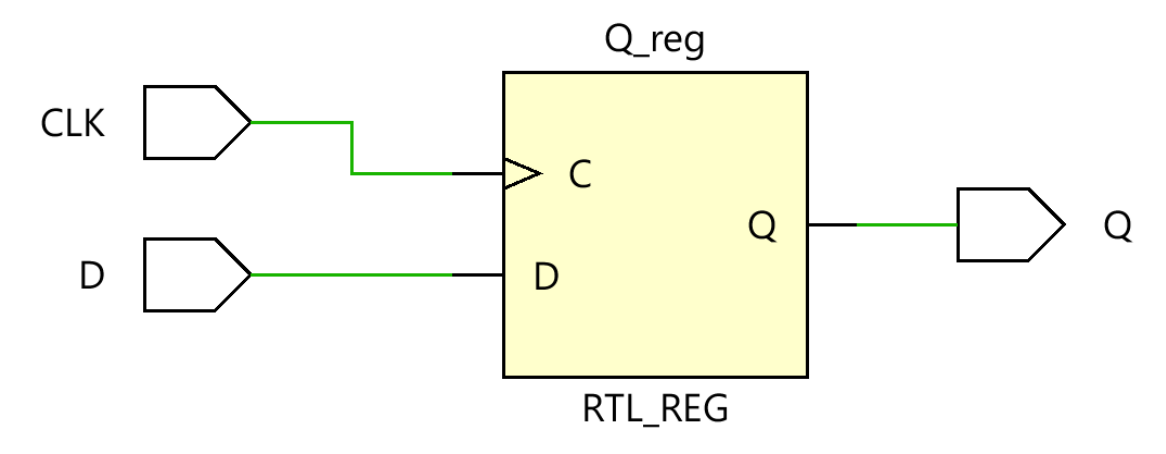
Το D δεν τοποθετείται στη λίστα ευαισθησίας

Η συνθήκη του σύγχρονου D εξετάζεται μετά τη συνθήκη του CLK

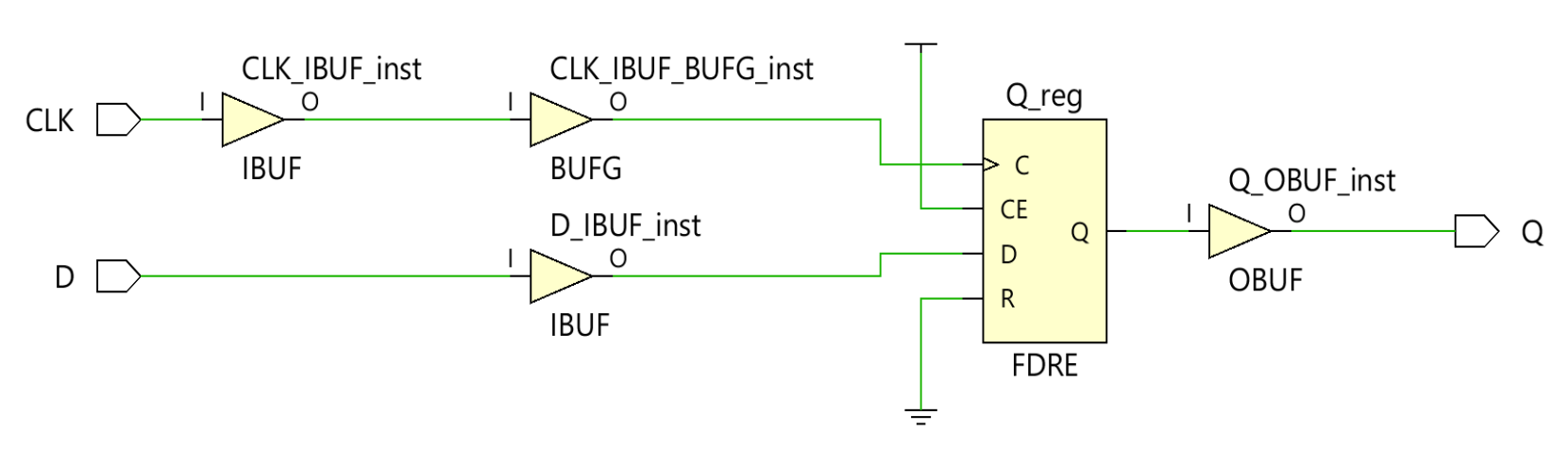
Προσοχή! Η απουσία του CLK'event οδηγεί στην υλοποίηση ενός D-Latch

Η αρχιτεκτονική του D Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



D Latch vs D Flip-Flop στη VHDL

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

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

D Flip-Flop με 2 εξόδους στη VHDL
Περιγραφή συμπεριφοράς

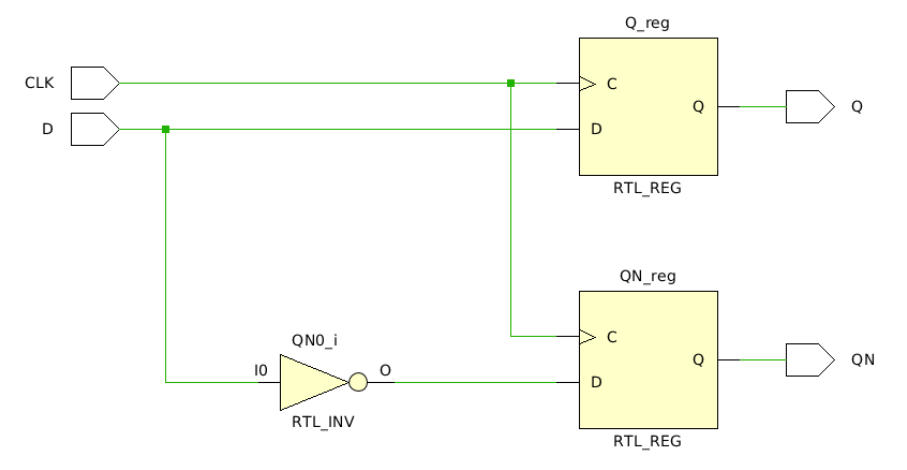
- Υλοποίηση δύο εξόδων (Q και QN) με δύο D Flip-Flop

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

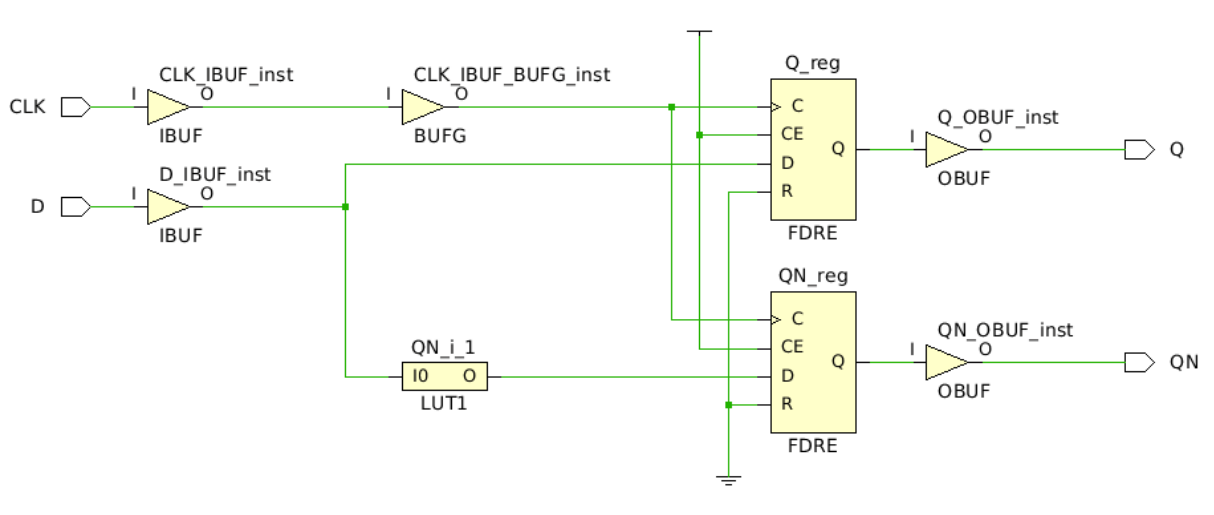
Ανάθεση τιμής στα Q και QN εντός process

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

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



D Flip-Flop με 2 εξόδους στη VHDL
Περιγραφή συμπεριφοράς

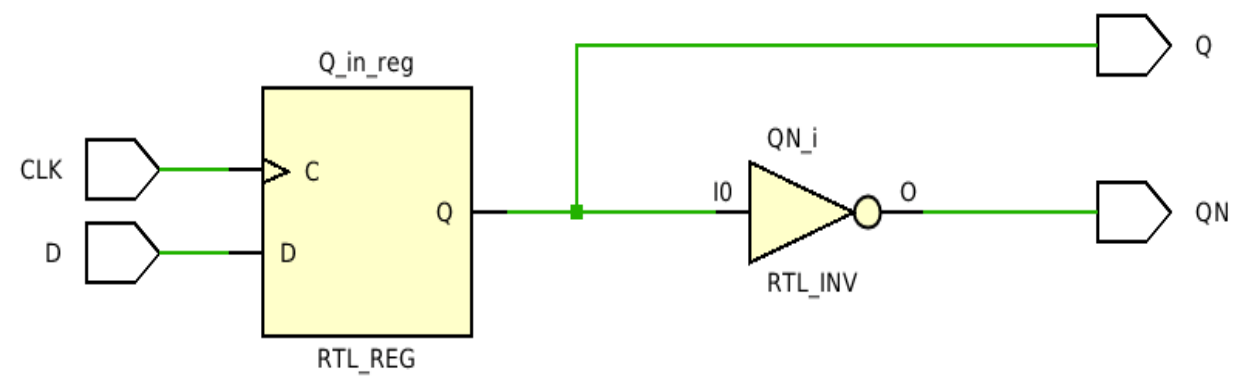
- Υλοποίηση δύο εξόδων (Q και QN) με ένα D Flip-Flop

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

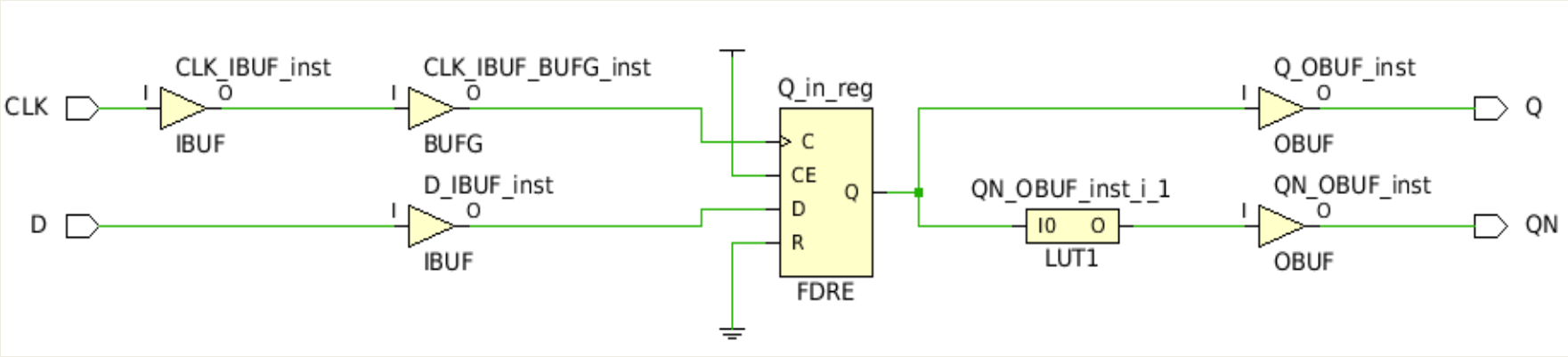
Ανάθεση τιμής στα Q και QN **εκτός** process

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

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA

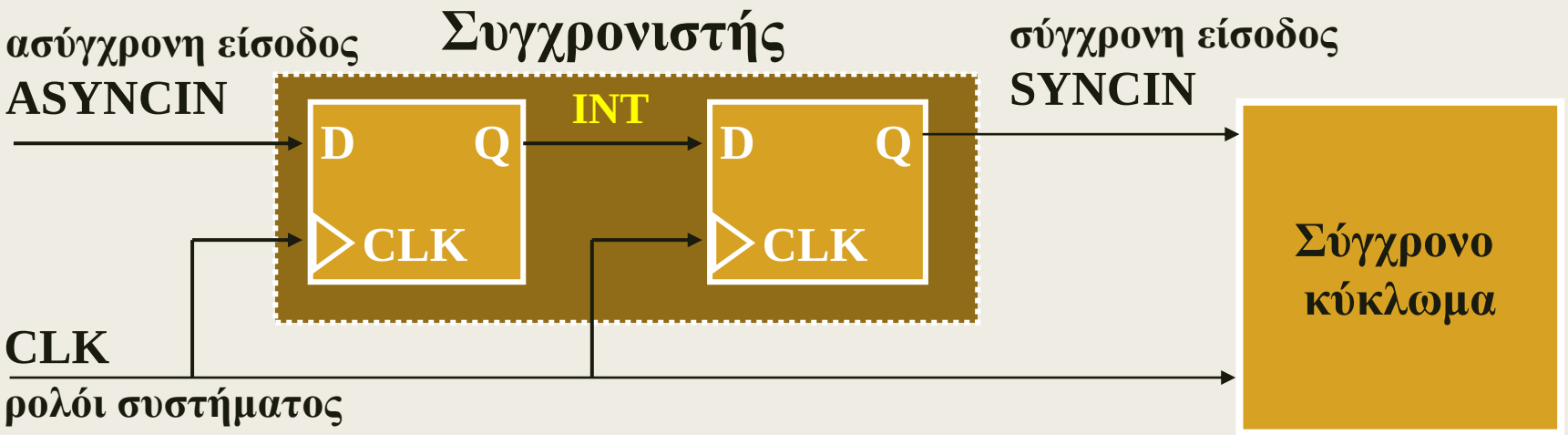


Ασύγχρονοι είσοδοι σε σύγχρονα κυκλώματα

- Οι **ασύγχρονες είσοδοι** δεν είναι συγχρονισμένες με το σήμα **CLK** ενός σύγχρονου κυκλώματος
- Όταν ένα D Flip-Flop δειγματοληπτεί μια ασύγχρονη είσοδο **ASYNCIN**, η οποία μεταβάλλεται κατά τη διάρκεια του **χρόνου ανοίγματος**, τότε η έξοδος Q ενδέχεται στιγμιαία να βρεθεί στη **μετασταθερή κατάσταση** (σε μια τάση που ανήκει στη μη αποδεκτή ζώνη)
  - *τελικά το D Flip-Flop θα καταλήξει σε μια σταθερή κατάσταση με τιμή είτε 0 είτε 1 για την έξοδο*
  - *ο **χρόνος κατάληξης** T_{res} (metastability resolution time) που απαιτείται για να καταλήξει η έξοδος σε μια σταθερή κατάσταση δεν είναι φραγμένος*
    - αλλά, η πιθανότητα να εξακολουθεί να βρίσκεται στη μετασταθερή κατάσταση μειώνεται εκθετικά με το χρόνο
- Τα ασύγχρονα σήματα για να εισέλθουν σε ένα σύγχρονο κύκλωμα περνούν μέσα από τον **συγχρονιστή**

Συγχρονιστής στη VHDL
Περιγραφή συμπεριφοράς

- Ο **συγχρονιστής** παράγει το συγχρονισμένο αντίγραφο **SYNCIN** της ασύγχρονης εισόδου **ASYNCIN**, όταν ισχύει η σχέση:
  - *Cycle-time* > T_{res} + *Set-up-time* + *Skew-time*



Συγχρονιστής στη VHDL
Περιγραφή συμπεριφοράς

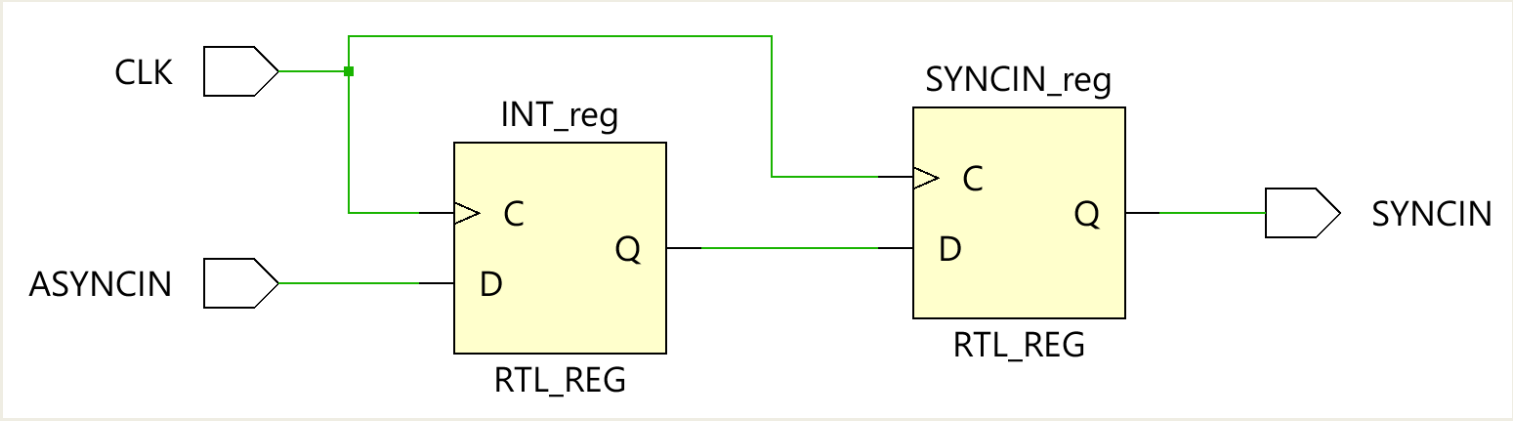
```
entity SYNC is
  port (
    CLK: in STD_LOGIC;
    ASYNCIN: in STD_LOGIC;
    SYNCIN: out STD_LOGIC);
end SYNC;
architecture BEHAVIORAL of SYNC is
signal INT : STD_LOGIC;
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      INT <= ASYNCIN;
      SYNCIN <= INT;
    end if;
  end process;
end DFF_BEH;
```

Απαίτηση για εσωτερικό σήμα INT

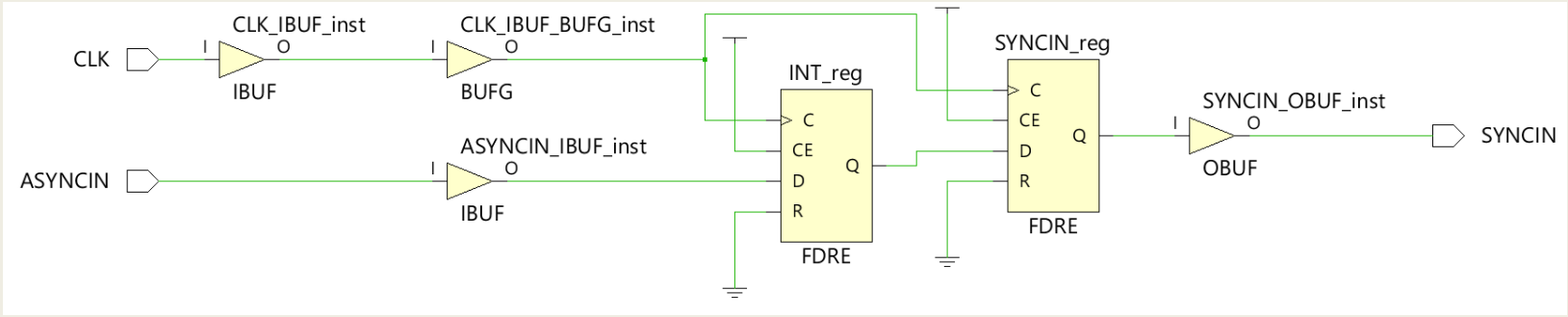
Το SYNCIN εμφανίζεται μετά από ένα κύκλο του CLK

Συγχρονιστής στη VHDL
Περιγραφή συμπεριφοράς

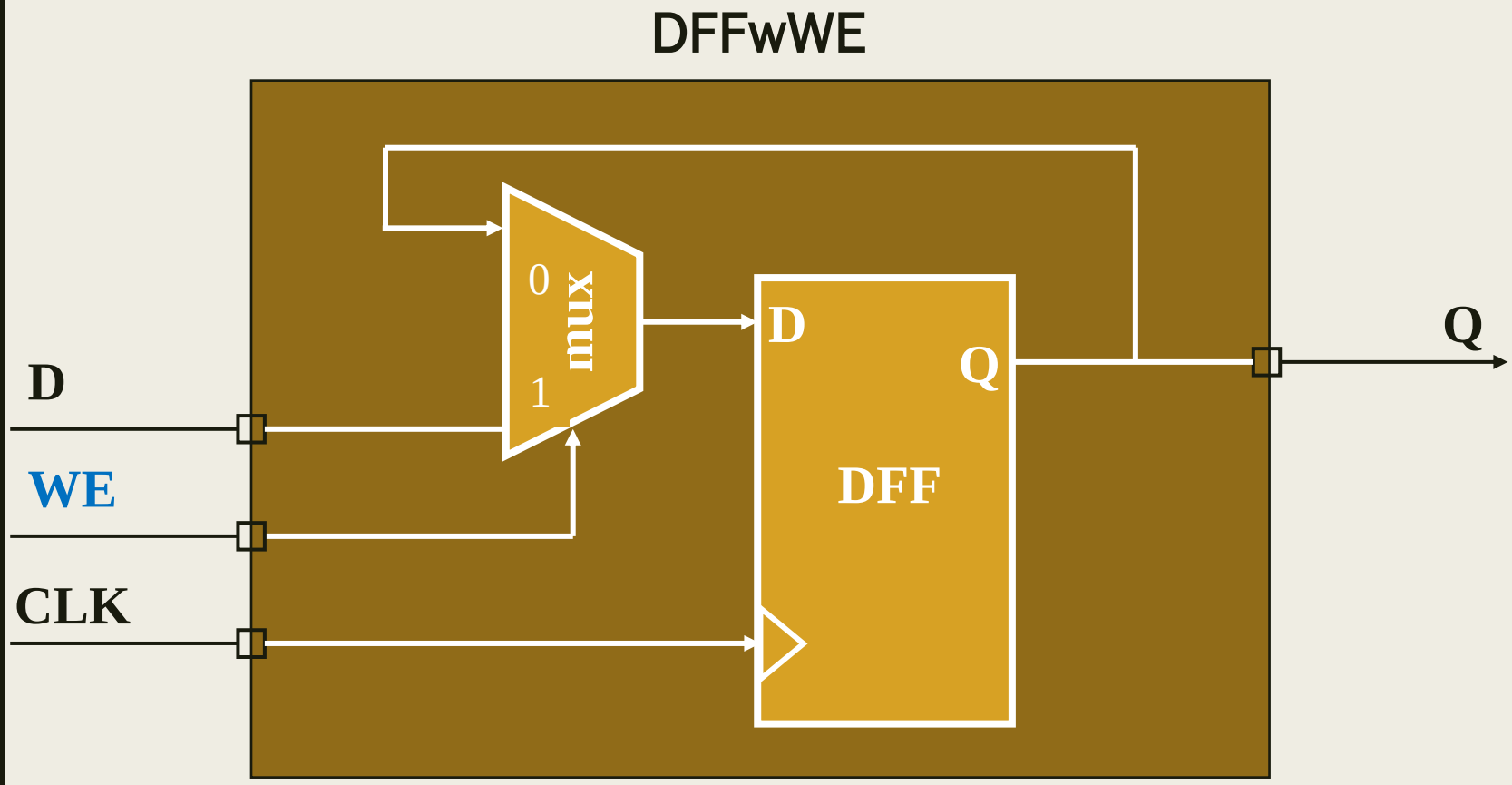
- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



D Flip-Flop with Write Enable στη VHDL
Περιγραφή συμπεριφοράς



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

D Flip-Flop with Write Enable στη VHDL
Περιγραφή συμπεριφοράς

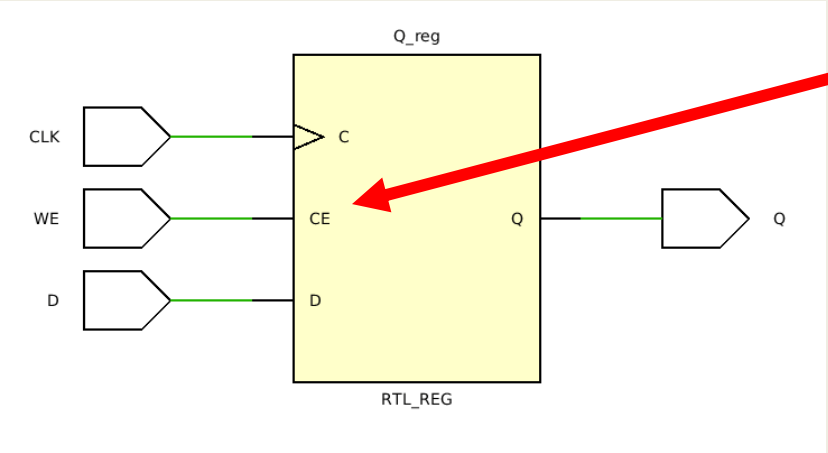
```
entity DFFwWE is
  port (
    CLK, D, WE: in STD_LOGIC;
    Q: out STD_LOGIC);
end DFFwWE;
architecture DFFwWE_BEH of DFFwWE is
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      if (WE = '1') then
        Q <= D;
      end if;
    end if;
  end process;
end DFFwWE_BEH;
```

Το WE δεν τοποθετείται στη λίστα ευαισθησίας

Η συνθήκη του σύγχρονου WE εξετάζεται μετά τη συνθήκη του CLK

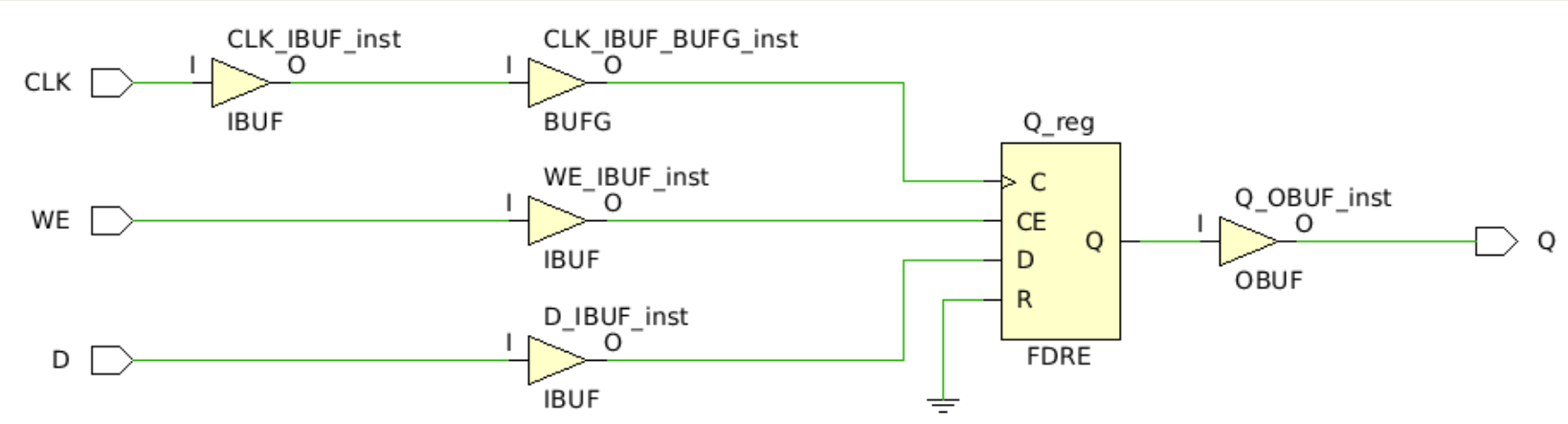
D Flip-Flop with Write Enable στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL

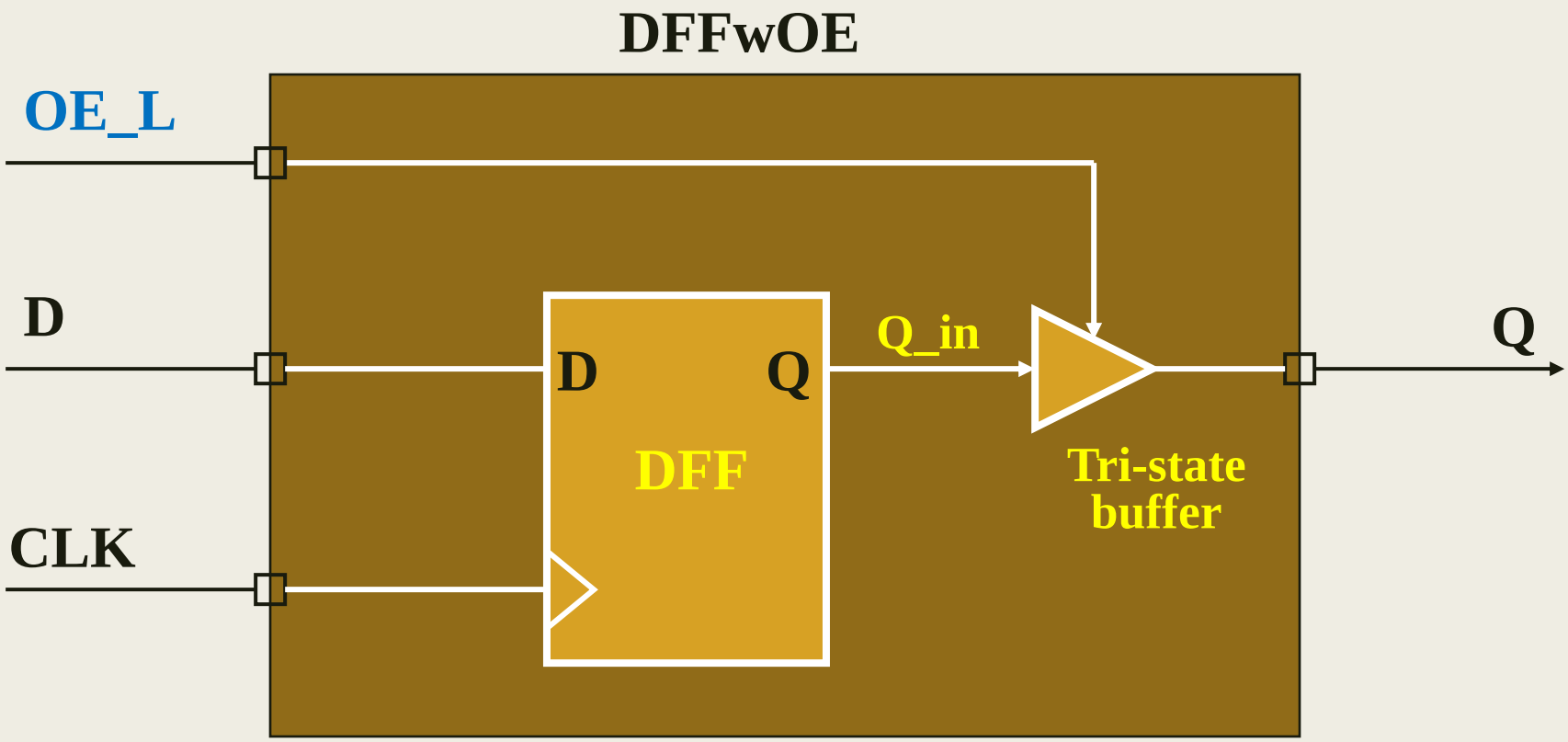


Χρησιμοποιείται η είσοδος Clock Enable (CE) του D Flip-Flop

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



D Flip-Flop with Output (Read) Enable στη VHDL
Περιγραφή συμπεριφοράς



Το σήμα Output Enable Active Low (OE_L = 0) δεν επηρεάζει τη λειτουργία του DFF. Ο tri-state buffer συνδέεται με την έξοδο Q του DFF μέσω του εσωτερικού σήματος Q_in. Χρησιμοποιείται μόνο να για εξόδους σε ακροδέκτες.

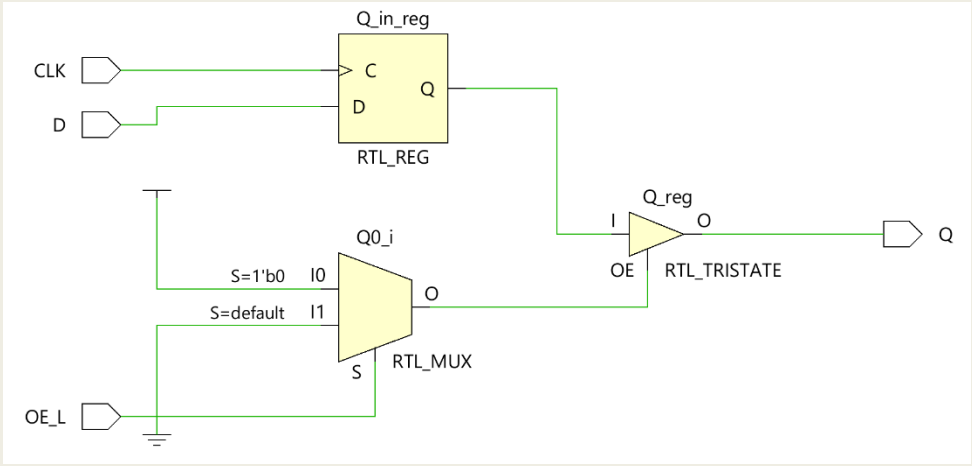
D Flip-Flop with Output (Read) Enable στη VHDL
Περιγραφή συμπεριφοράς

```
entity DFFwOE is
  port (
    CLK, D, OE_L: in STD_LOGIC;
    Q: out STD_LOGIC);
end DFFwOE;
architecture DFFwOE_BEH of DFFwOE is
signal Q_in: STD_LOGIC;
begin
  DFF: process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      Q_in <= D;
    end if;
  end process
  BUFFER1: process (OE_L, Q_in)
  begin
    if (OE_L = '0') then Q <= Q_in;
    else Q <= 'Z';
    end if;
  end process;
end DFFwOE_BEH;
```

Ο tri-state buffer **πάντα** σε χωριστό process

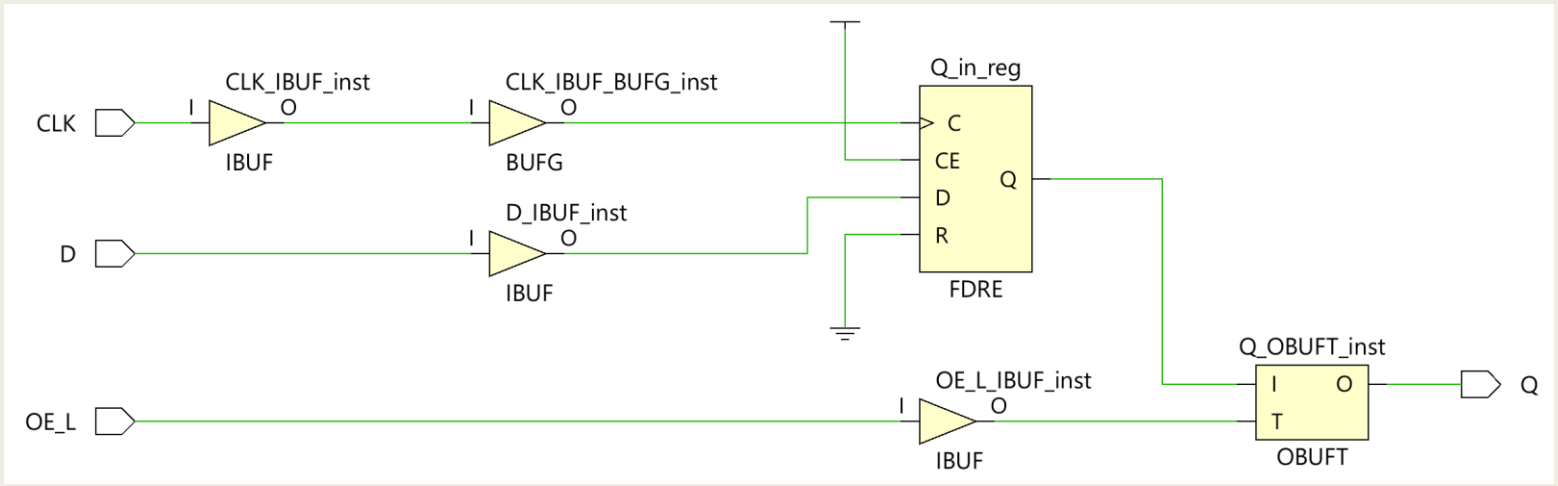
D Flip-Flop with Output (Read) Enable στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL (Το OE ορίζεται ως active high)

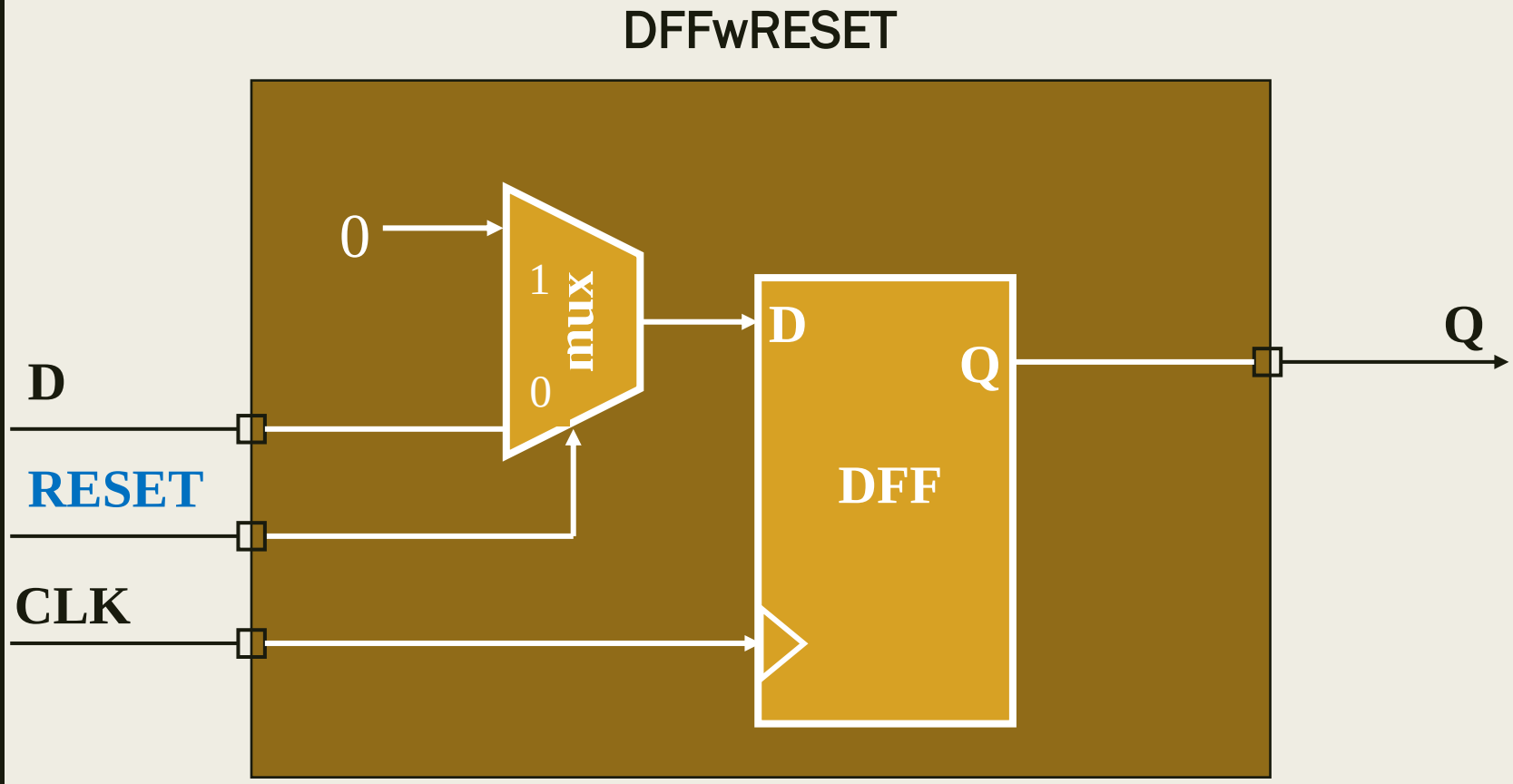


To Tri-State Buffer (O)BUFT υποστηρίζεται μόνο στα OPADs και IOPADs για σήματα που εγκαταλείπουν το chip

- Σχηματικό διάγραμμα σε τεχνολογία FPGA (Το OE είναι active low!)



D Flip-Flop with Reset στη VHDL
Περιγραφή συμπεριφοράς



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

D Flip-Flop with Reset στη VHDL
Περιγραφή συμπεριφοράς

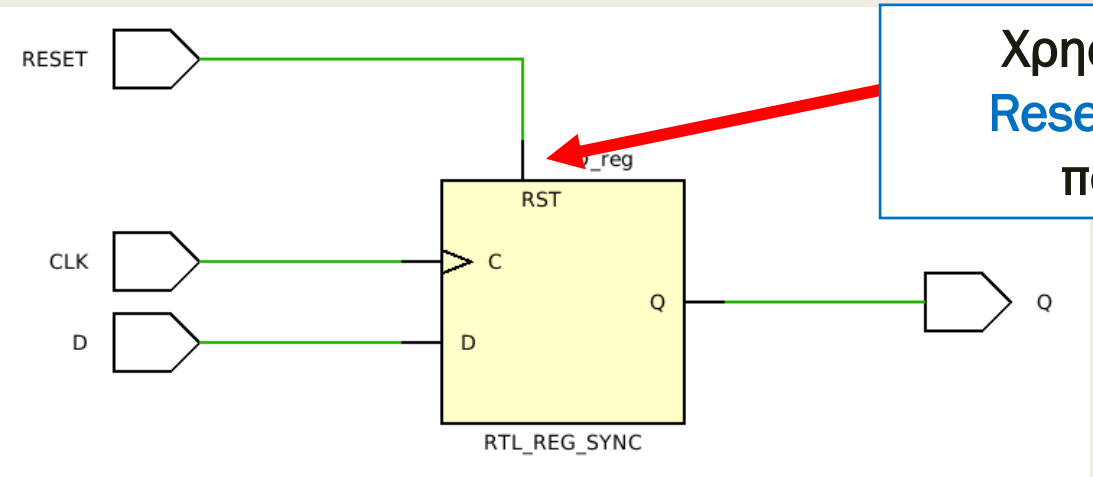
```
entity DFFwRESET is
  port (
    CLK, D, RESET: in STD_LOGIC;
    Q: out STD_LOGIC);
end DFFwRESET;
architecture DFFwRESET_BEH of DFFwRESET is
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      if (RESET = '1') then
      Q <= '0';
      else
        Q <= D;
      end if;
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

Το RESET δεν τοποθετείται στη λίστα ευαισθησίας

Η συνθήκη του σύγχρονου RESET εξετάζεται μετά τη συνθήκη του CLK

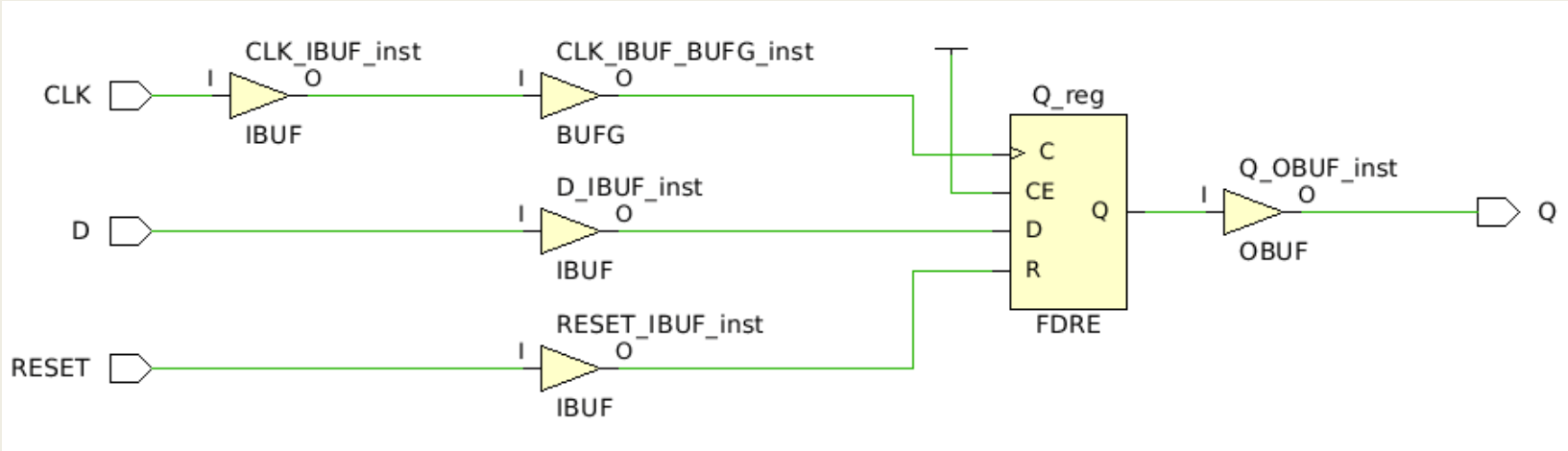
D Flip-Flop with Reset στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL

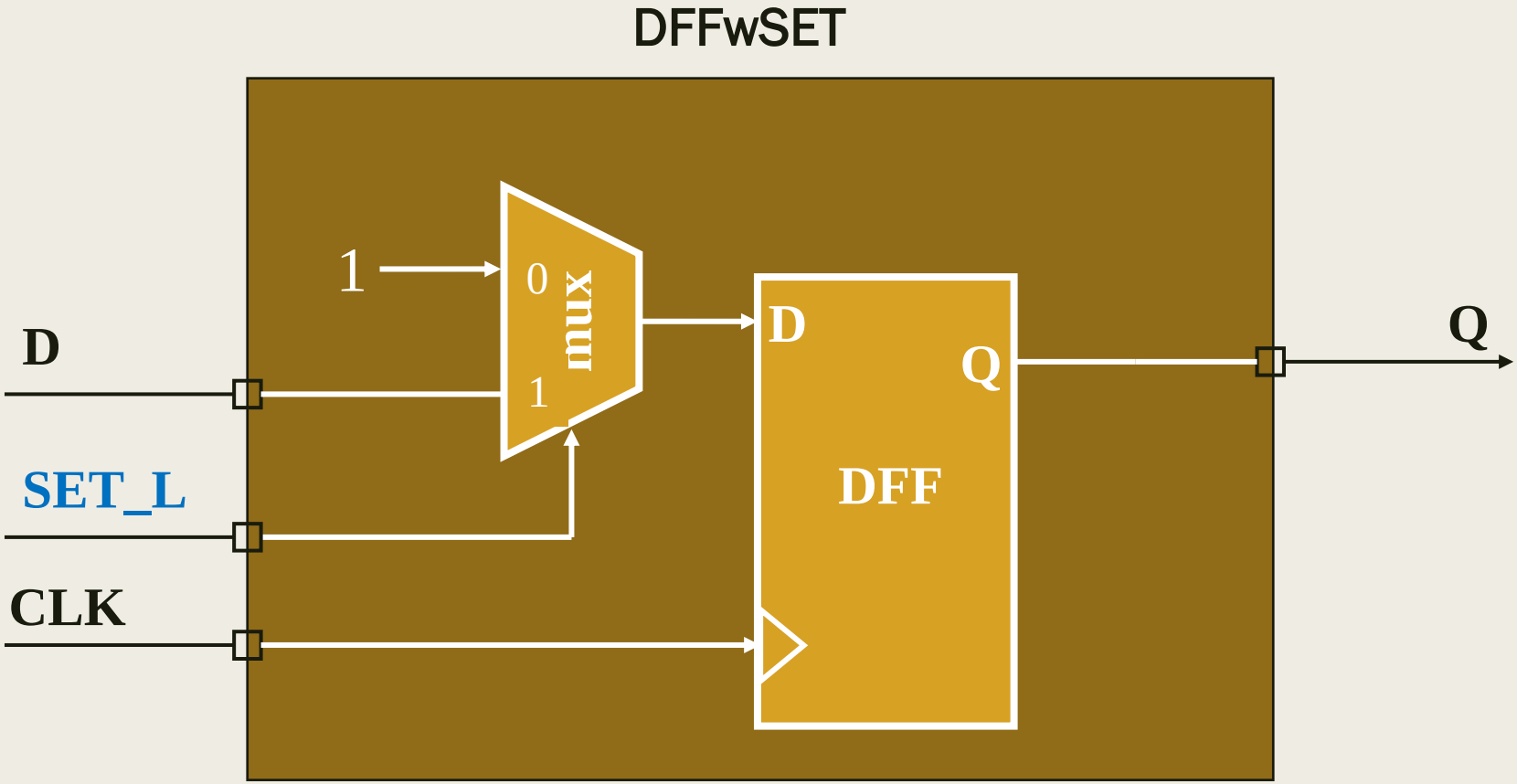


Χρησιμοποιείται η είσοδος Reset (RST) του D Flip-Flop που είναι active high

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



D Flip-Flop with Set στη VHDL
Περιγραφή συμπεριφοράς



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

D Flip-Flop with Set στη VHDL
Περιγραφή συμπεριφοράς

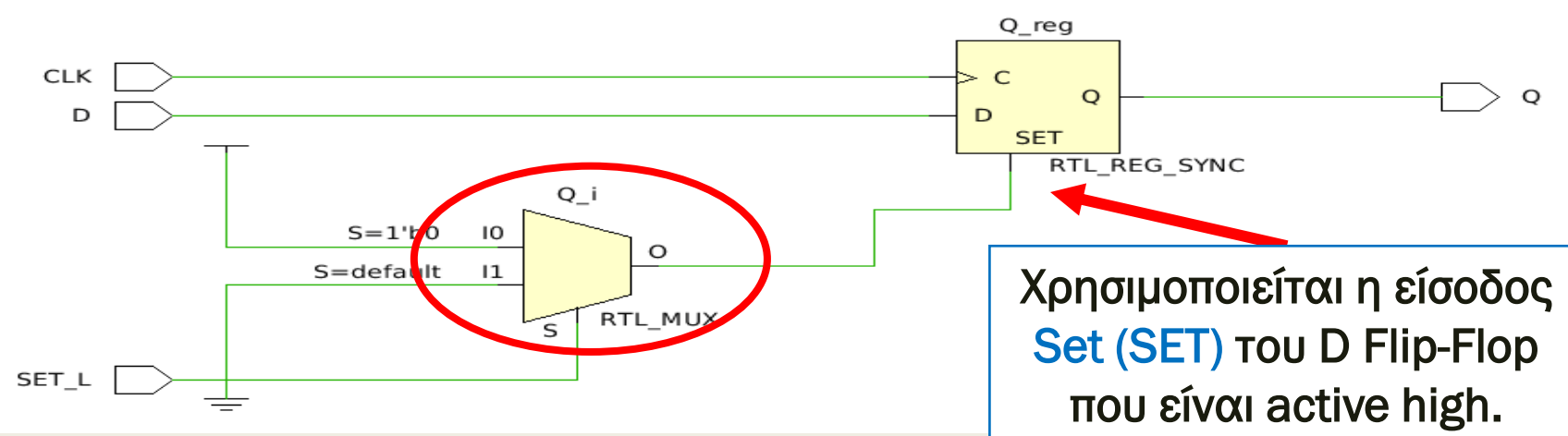
```
entity DFFwSET is
  port (
    CLK, D, SET_L: in STD_LOGIC;
    Q: out STD_LOGIC);
end DFFwSET;
architecture DFFwSET_BEH of DFFwSET is
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      if (SET_L = '0') then
        Q <= '1';
      else
        Q <= D;
      end if;
    end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

Το SET_L δεν τοποθετείται στη λίστα ευαισθησίας

Η συνθήκη του σύγχρονου SET εξετάζεται μετά τη συνθήκη του CLK

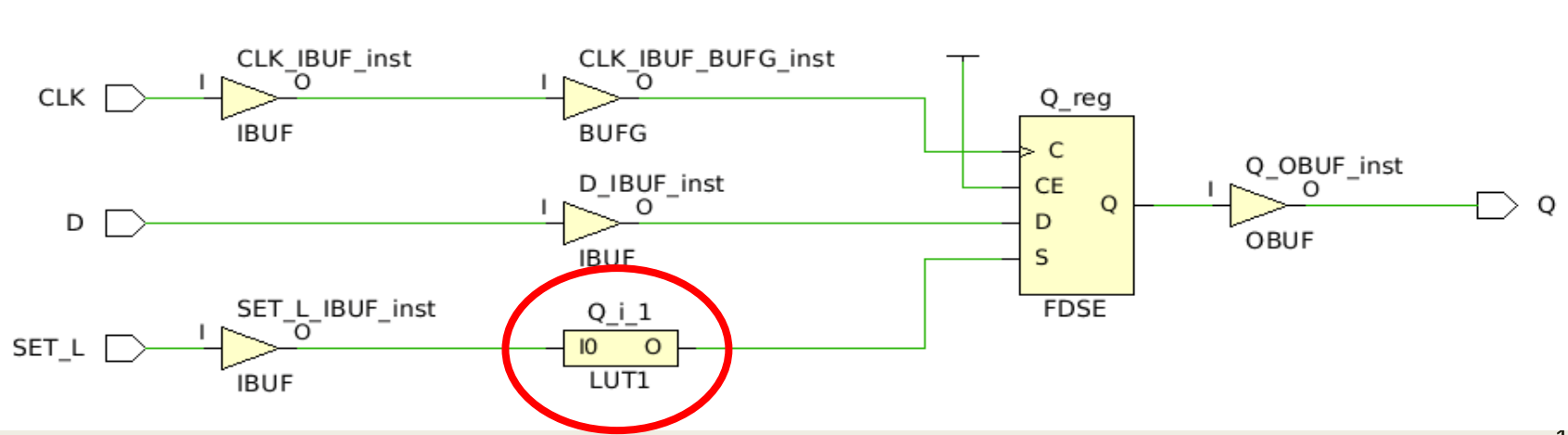
D Flip-Flop with Set στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL

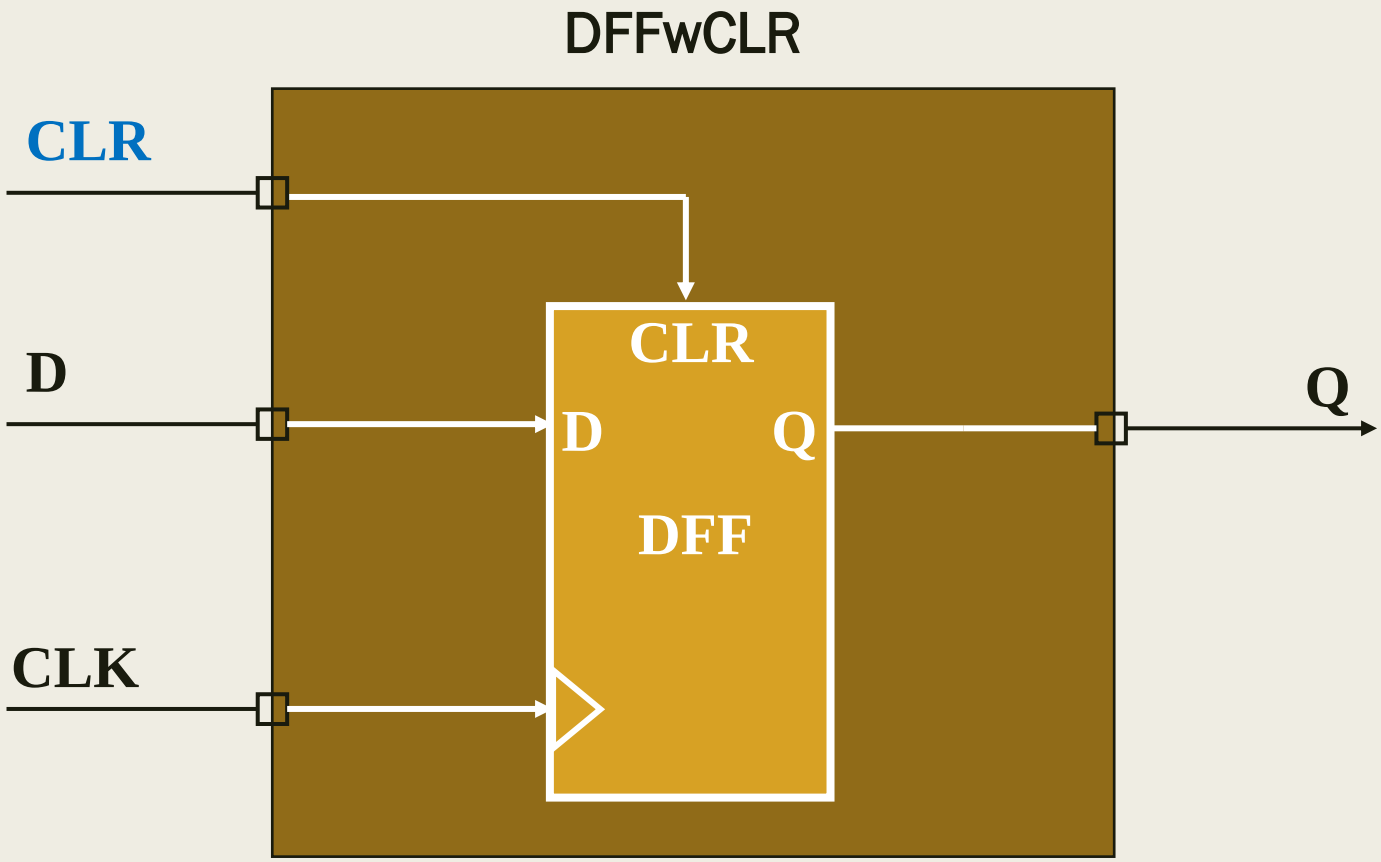


Χρησιμοποιείται η είσοδος Set (SET) του D Flip-Flop που είναι active high. Απαιτείται η χρήση NOT

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



D Flip-Flop with Clear στη VHDL
Περιγραφή συμπεριφοράς



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

D Flip-Flop with Clear στη VHDL
Περιγραφή συμπεριφοράς

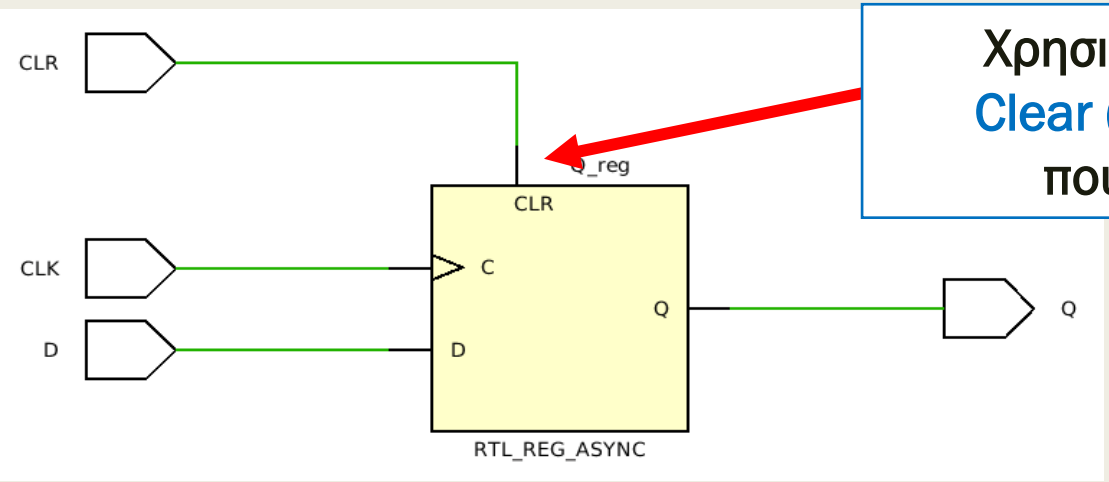
```
entity DFFwCLR is
  port (
    D, CLK, CLR: in STD_LOGIC;
    Q: out STD_LOGIC);
end DFFwCLR;
architecture DFFwCLR_BEH of DFFwCLR is
begin
  process (CLK, CLR)
  begin
    if (CLR = '1') then
      Q <= '0';
    elsif (CLK = '1' and CLK'event) then
      Q <= D;
    end if;
  end process;
end DFFwCLR_BEH;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

Το CLR τοποθετείται στη λίστα ευαισθησίας

Η συνθήκη του ασύγχρονου CLR εξετάζεται πριν τη συνθήκη του CLK

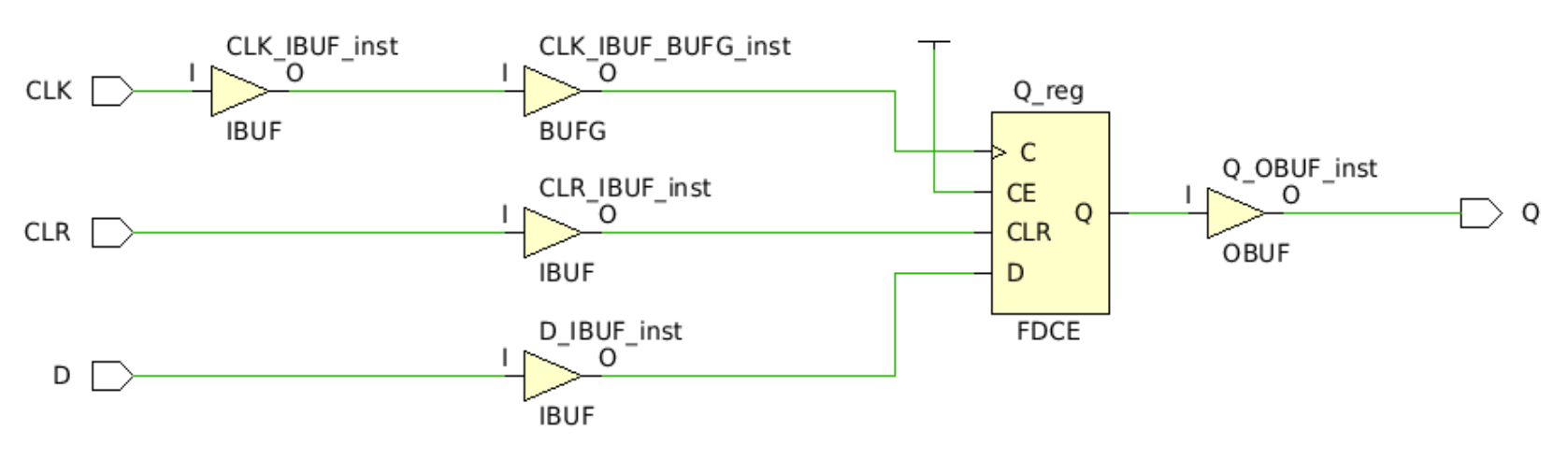
D Flip-Flop with Clear στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



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

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Σύγχρονο RESET vs ασύγχρονου CLEAR στη VHDL

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

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

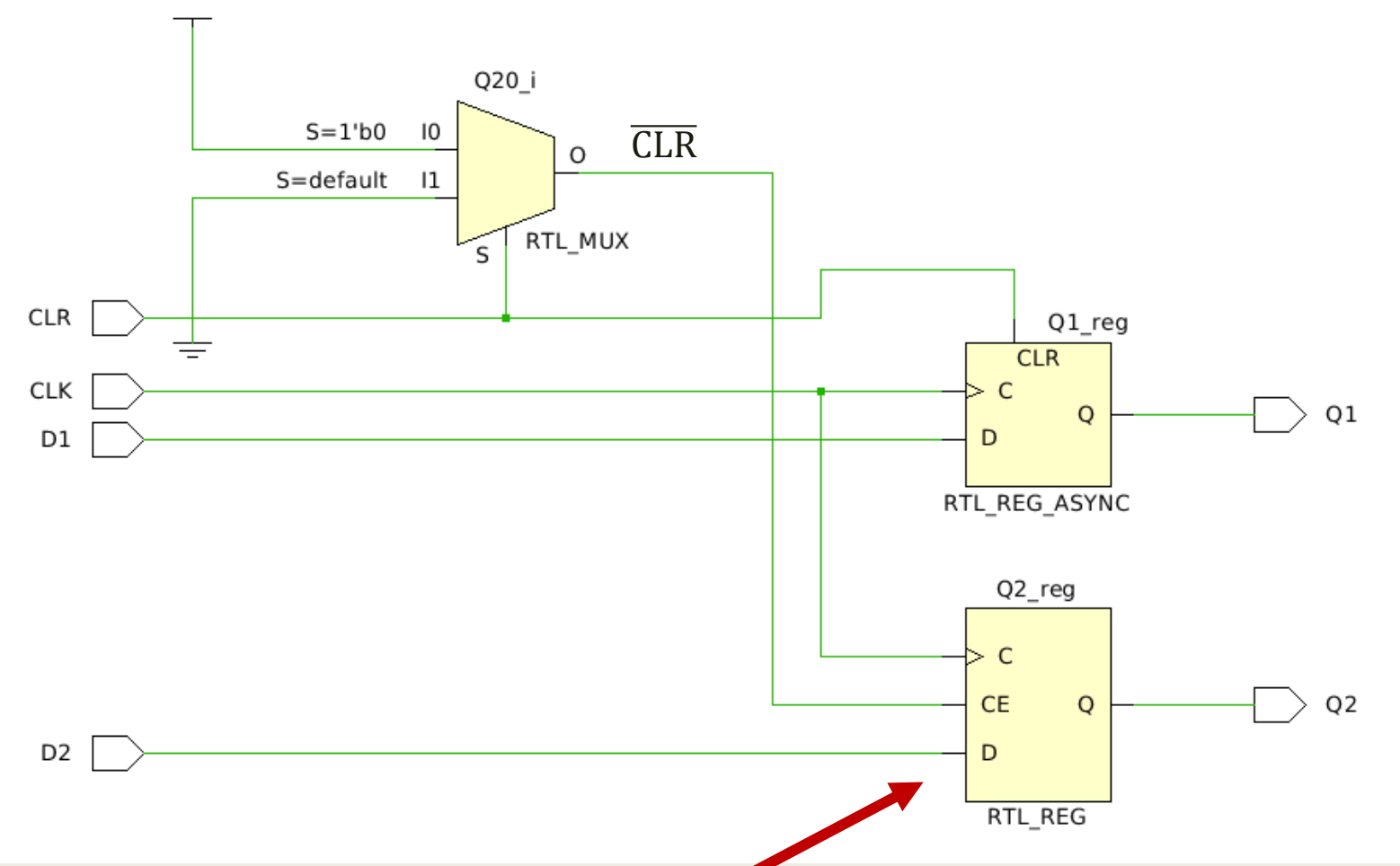
D Flip-Flop with Clear στη VHDL
Περιγραφή συμπεριφοράς

```
entity DFF2wACLR is
  port (
    CLK, D1, D2, CLR: in STD_LOGIC;
    Q1, Q2: out STD_LOGIC);
end DFF2wACLR;
architecture DFF2wACLR_BEH of DFF2wACLR is
begin
  process (CLK, CLR)
  begin
    if (CLR = '1') then
      Q1 <= '0';
    elsif (CLK = '1' and CLK'event) then
      Q1 <= D1; Q2 <= D2;
    end if;
  end process;
end DFF2wACLR_BEH;
```

Ελλιπής ανάθεση του Q2 κατά την περιγραφή της λειτουργίας του CLR

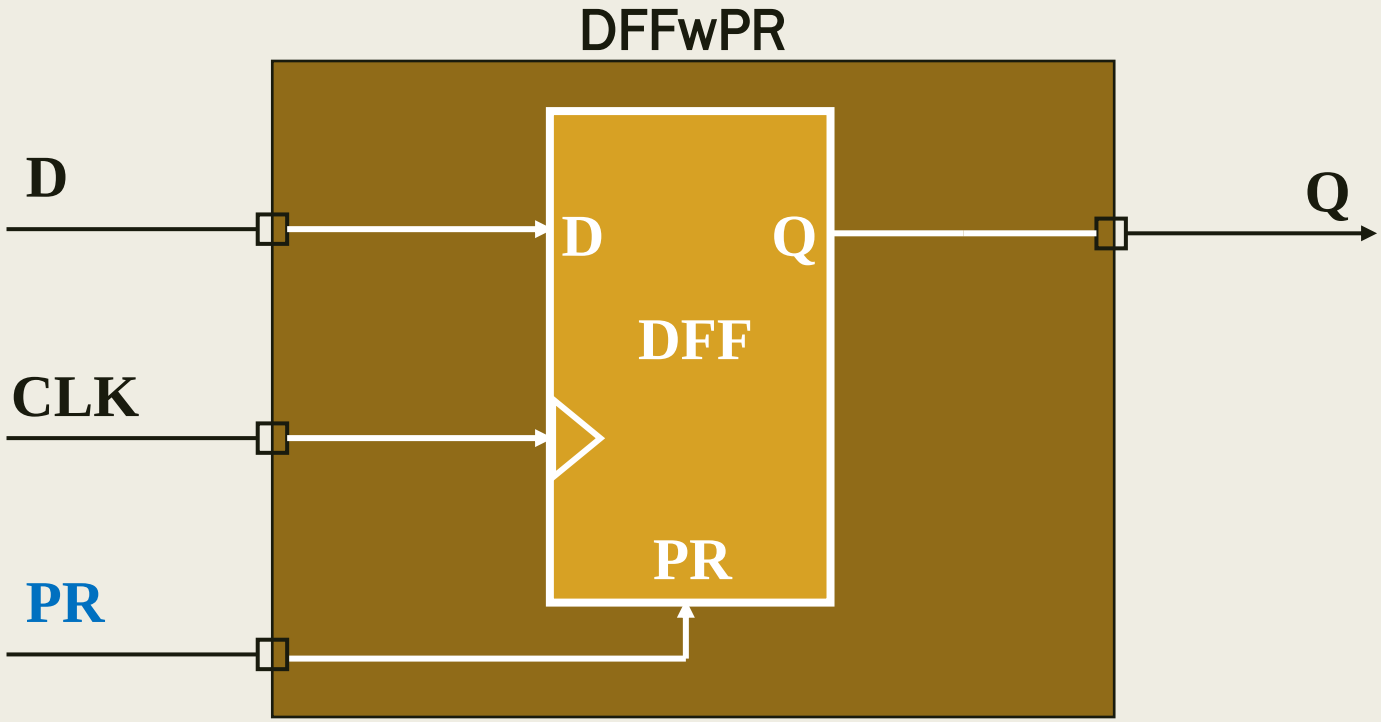
Ποιο είναι το αποτέλεσμα της σύνθεσης;

D Flip-Flop with Clear στη VHDL
Περιγραφή συμπεριφοράς



Το D Flip-Flop with Clock Enable διατηρεί την κατάστασή του όταν CLR = 1, γιατί $\mathbf{CE = \overline{CLR}}$

D Flip-Flop with Preset στη VHDL
Περιγραφή συμπεριφοράς



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

D Flip-Flop with Preset στη VHDL
Περιγραφή συμπεριφοράς

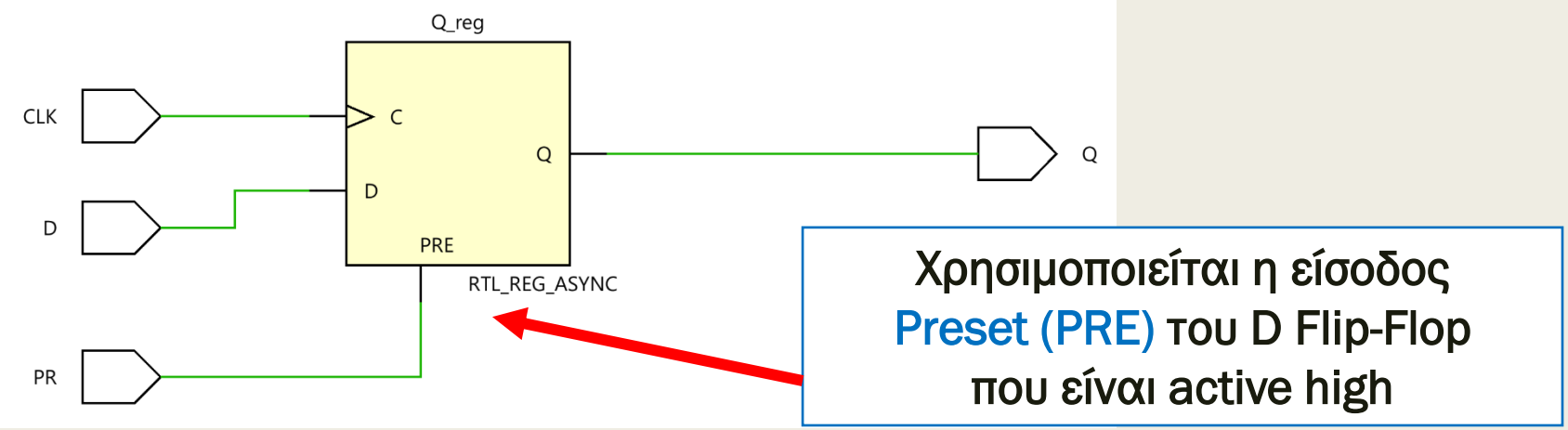
```
entity DFFwCLR is
  port (
    D, CLK, CLR: in STD_LOGIC;
    Q: out STD_LOGIC);
end DFFwCLR;
architecture DFFwPR_BEH of DFFwPR is
begin
  process (CLK, PR)
  begin
    if (PR = '1') then
      Q <= '1';
    elsif (CLK = '1' and CLK'event) then
      Q <= D;
    end if;
  end process;
end DFFwPR_BEH;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

Το PR τοποθετείται στη λίστα ευαισθησίας

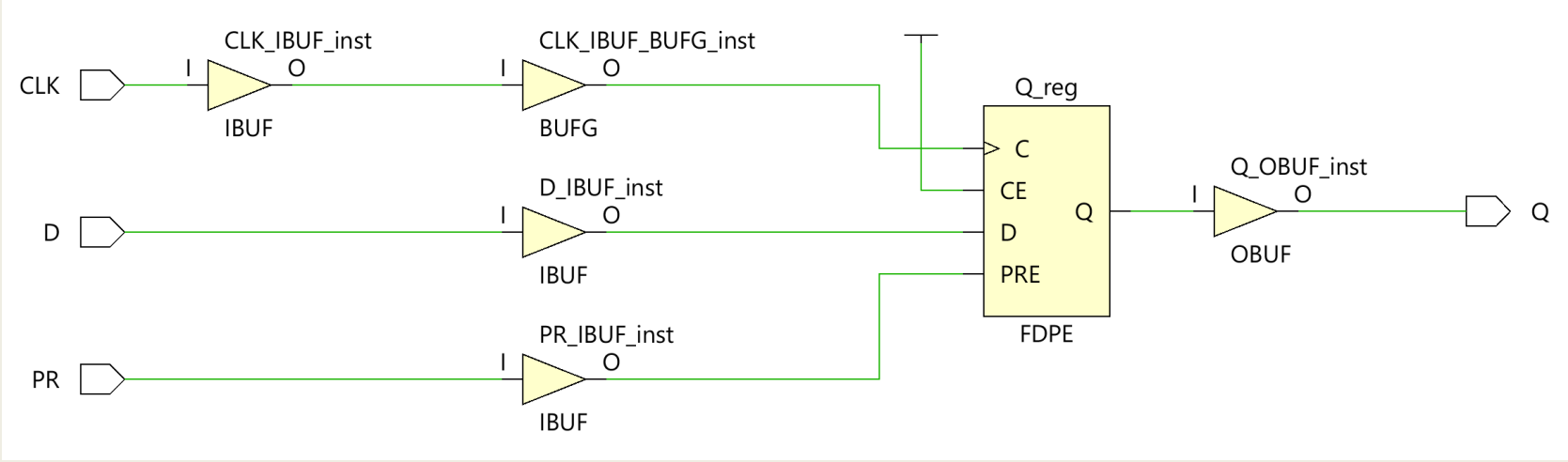
Η συνθήκη του ασύγχρονου PR εξετάζεται πριν τη συνθήκη του CLK

D Flip-Flop with Preset στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL

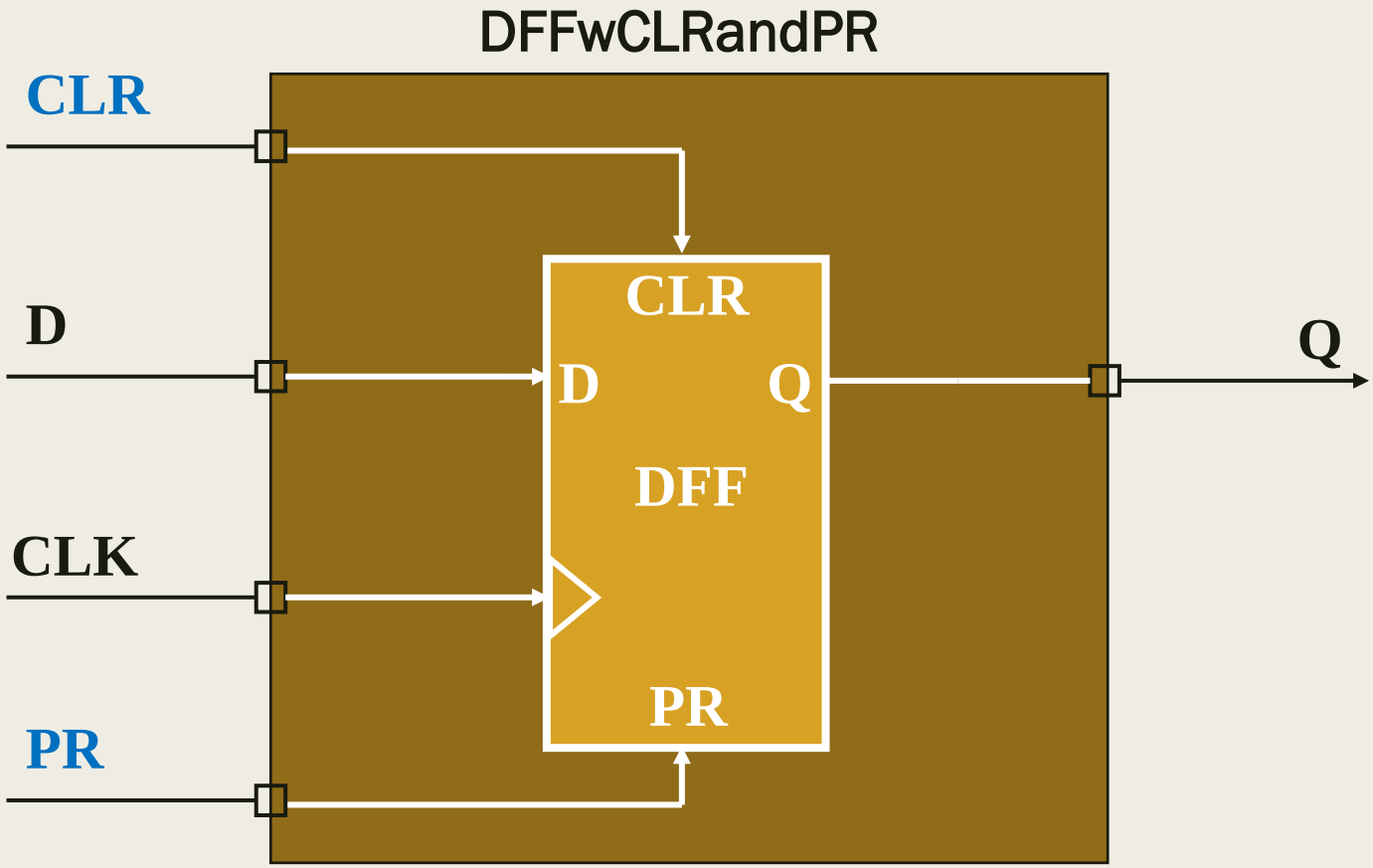


- Σχηματικό διάγραμμα σε τεχνολογία FPGA



D Flip-Flop with Clear και Preset στη VHDL

Περιγραφή συμπεριφοράς



Τα σήματα Clear Active High (CLR = 1) και Preset Active High (CLR = 1) είναι ασύγχρονα

D Flip-Flop with Clear και Preset στη VHDL

Περιγραφή συμπεριφοράς

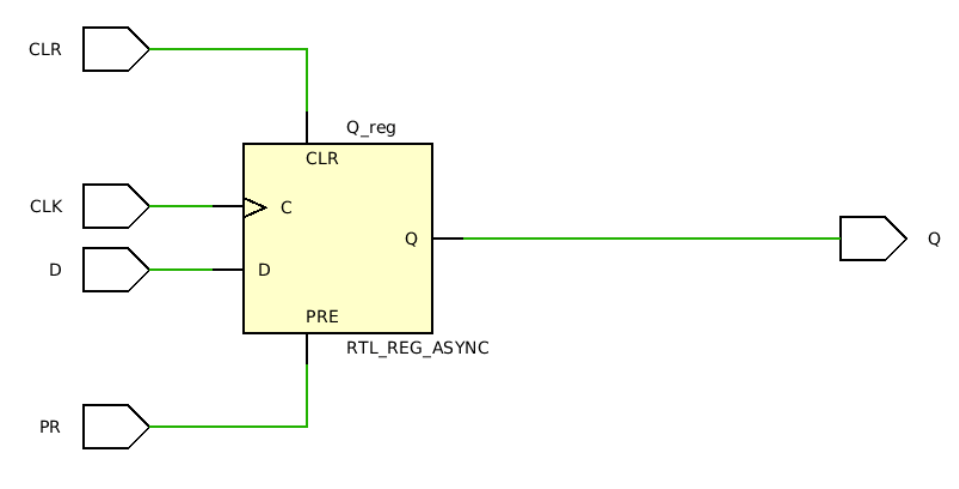
```
entity DFFwCLRandPR2 is
    port (
        D, CLK, CLR, PR : in STD_LOGIC;
        Q : out STD_LOGIC);
end DFFwCLRandPR2;
architecture BEH of DFFwCLRandPR2 is
begin
  process (CLK, CLR, PR)
  begin
    if     (CLR = '1')                then Q <= '0';
    elsif  (PR  = '1')                then Q <= '1';
    elsif  (CLK = '1' and CLK'event)  then Q <= D;
    end if;
  end process;
end BEH;
```

Η ασύγχρονη είσοδος CLR έχει τη μεγαλύτερη προτεραιότητα και επαναφέρει την κατάσταση στο 0, όταν CLR = 1

D Flip-Flop with Clear και Preset στη VHDL

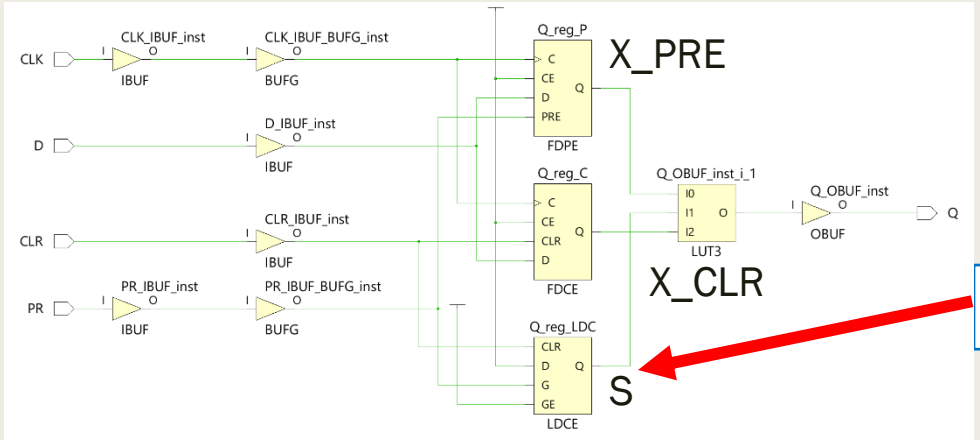
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



Το CLR υπερέχει του PR, όταν CLR = PR = 1

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



$$Q = S\,X\_PRE + \bar{S}\,X\_CLR$$

Latch

D Flip-Flop with Clear και Preset στη VHDL

Περιγραφή συμπεριφοράς

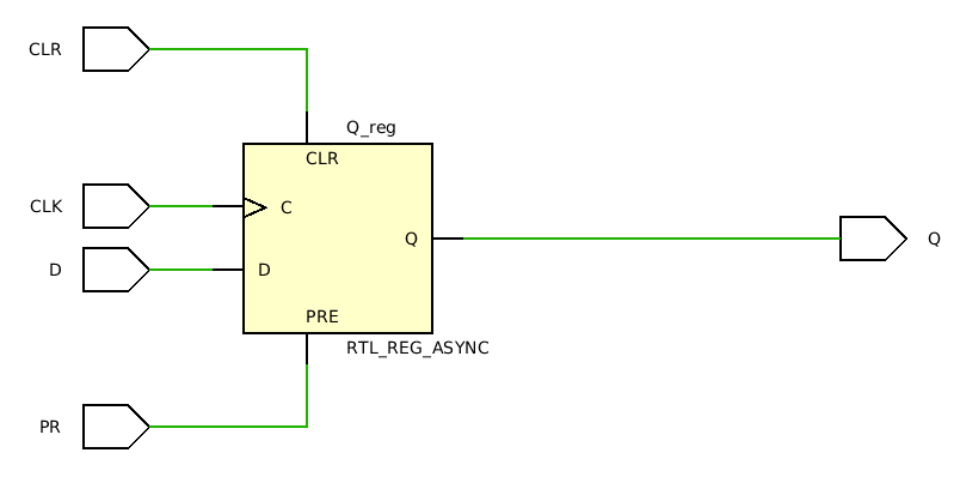
```
entity DFFwCLRandPR3 is
    port (
        D, CLK, CLR, PR : in STD_LOGIC;
        Q : out STD_LOGIC);
end DFFwCLRandPR3;
architecture BEH of DFFwCLRandPR3 is
begin
  process (CLK, CLR, PR)
  begin
    if     (PR  = '1')                then Q <= '1';
    elsif (CLR = '1')                 then Q <= '0';
    elsif (CLK = '1' and CLK'event)   then Q <= D;
    end if;
  end process;
end BEH ;
```

Η ασύγχρονη είσοδος PR έχει τη μεγαλύτερη προτεραιότητα και θέτει την κατάσταση στο 1, όταν PR = 1

D Flip-Flop with Clear και Preset στη VHDL

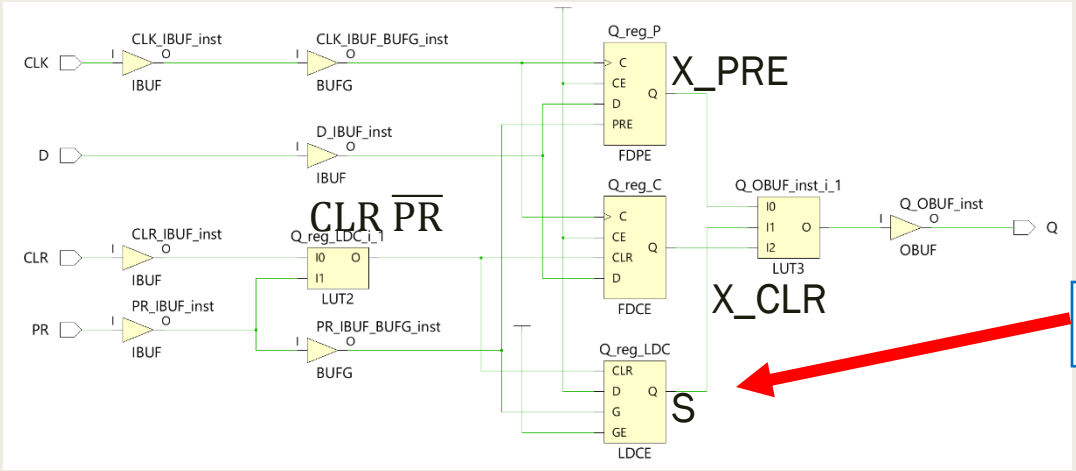
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



Το PR υπερέχει του CLR, όταν CLR = PR = 1

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



$$Q = S\,X\_PRE + \overline{S}\,X\_CLR$$

Latch

D Flip-Flop with Clear και Preset στη VHDL

Περιγραφή συμπεριφοράς

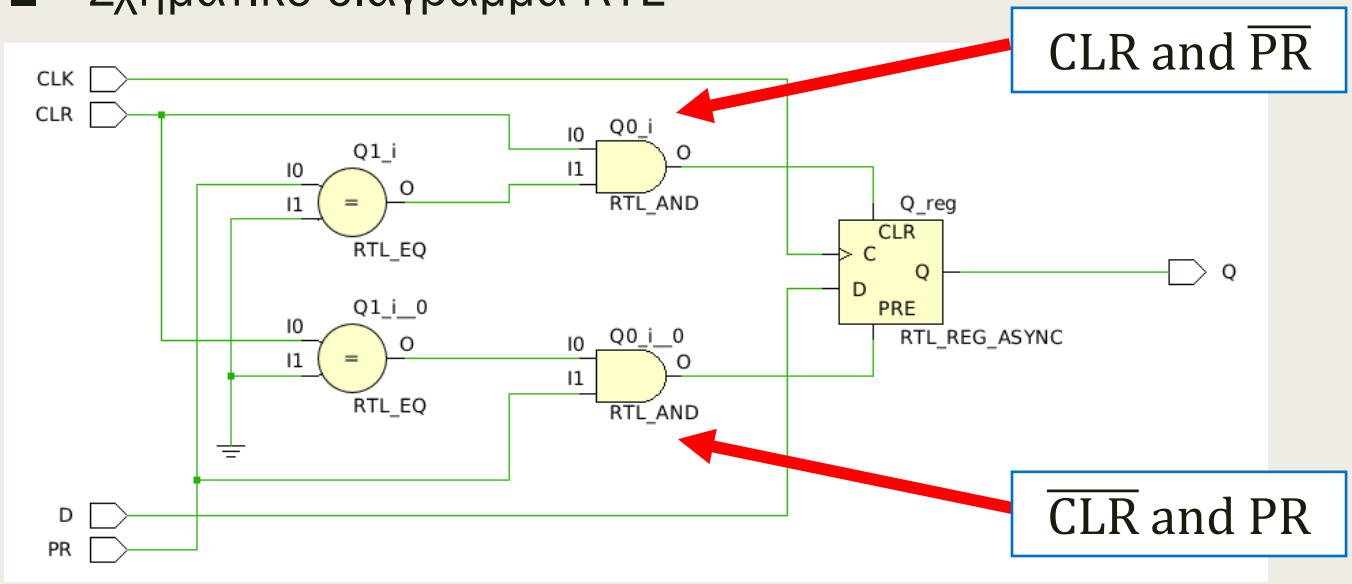
```
entity DFFwCLRandPR is
    port (
        D, CLK, CLR, PR : in STD_LOGIC;
        Q : out STD_LOGIC);
end DFFwCLRandPR;
architecture BEH of DFFwCLRandPR is
begin
  process (CLK, CLR, PR)
  begin
    if    (CLR = '1' and PR = '0')  then Q <= '0';
    elsif (CLR = '0' and PR = '1')  then Q <= '1';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

Στην περίπτωση που CLR = PR = 0 (ή CLR = PR = 1) λειτουργεί σαν κανονικό D-Flip-Flop

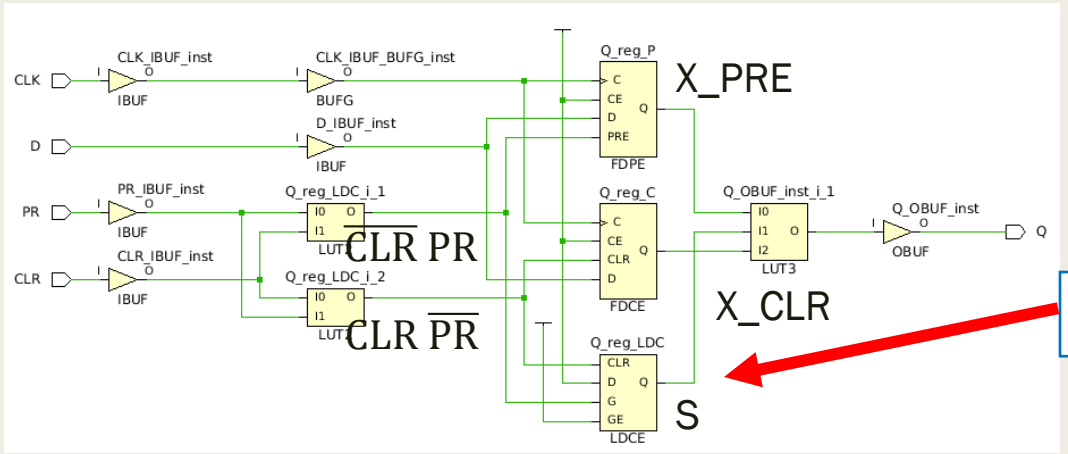
D Flip-Flop with Clear και Preset στη VHDL

Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



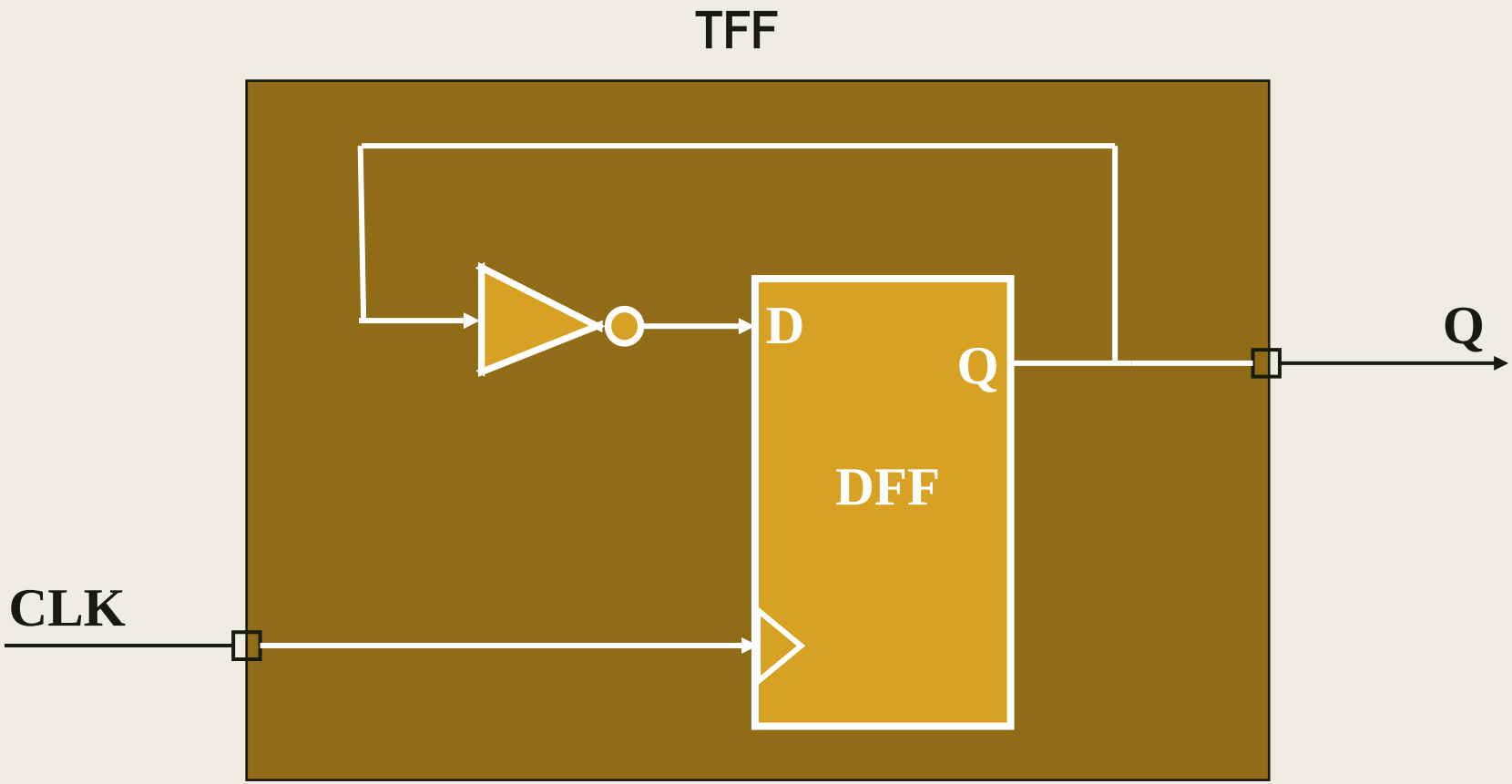
- Σχηματικό διάγραμμα σε τεχνολογία FPGA



$$Q = S\,X\_PRE + \bar{S}\,X\_CLR$$

Latch

T (Toggle) Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς



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

T Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

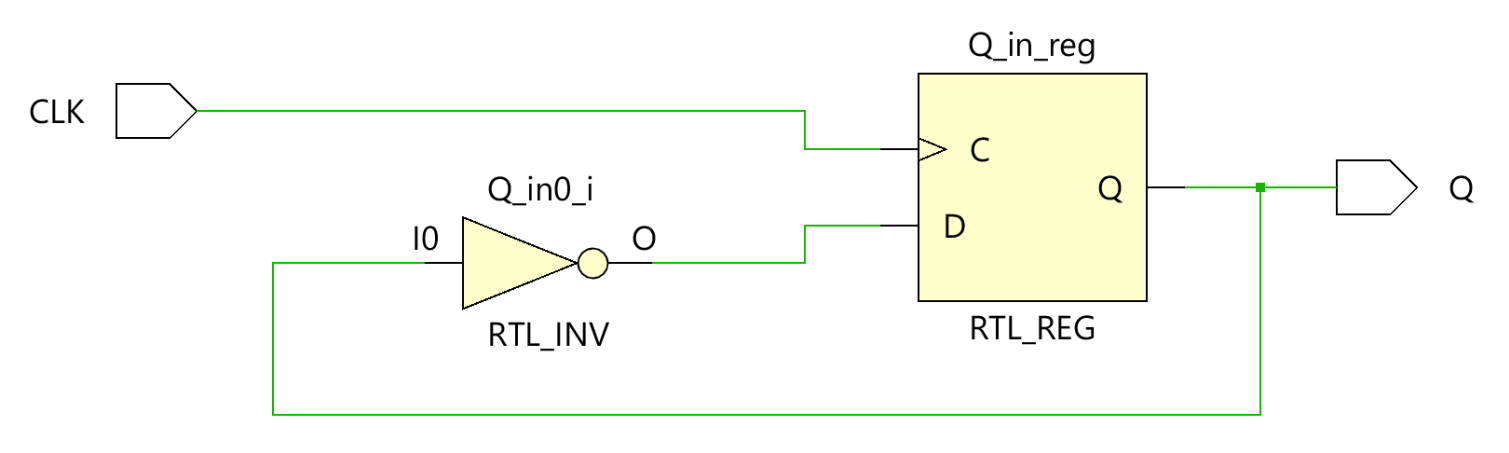
```
entity TFF is
  port (
    CLK : in STD_LOGIC;
    Q : out STD_LOGIC);
end TFF;
architecture BEHAVIORAL of TFF is
  signal Q_in: STD_LOGIC;
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      Q_in <= not Q_in;
    end if;
  end process;
  Q <= Q_in;
end BEHAVIORAL;
```

Απαίτηση για εσωτερικό σήμα Q_in

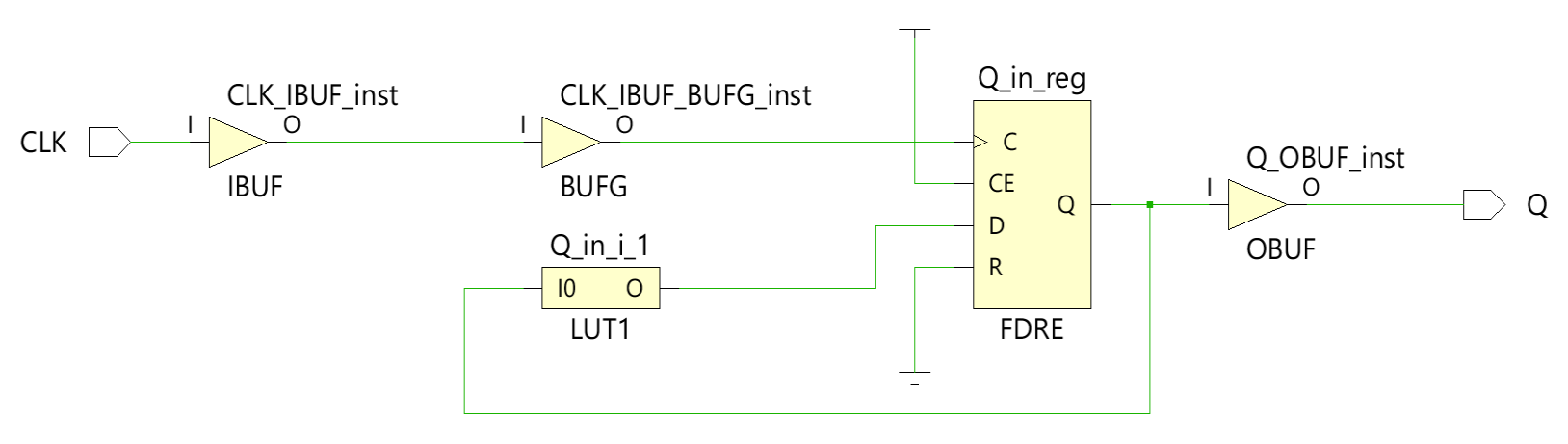
Ανάθεση τιμής στο Q εκτός process

T Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

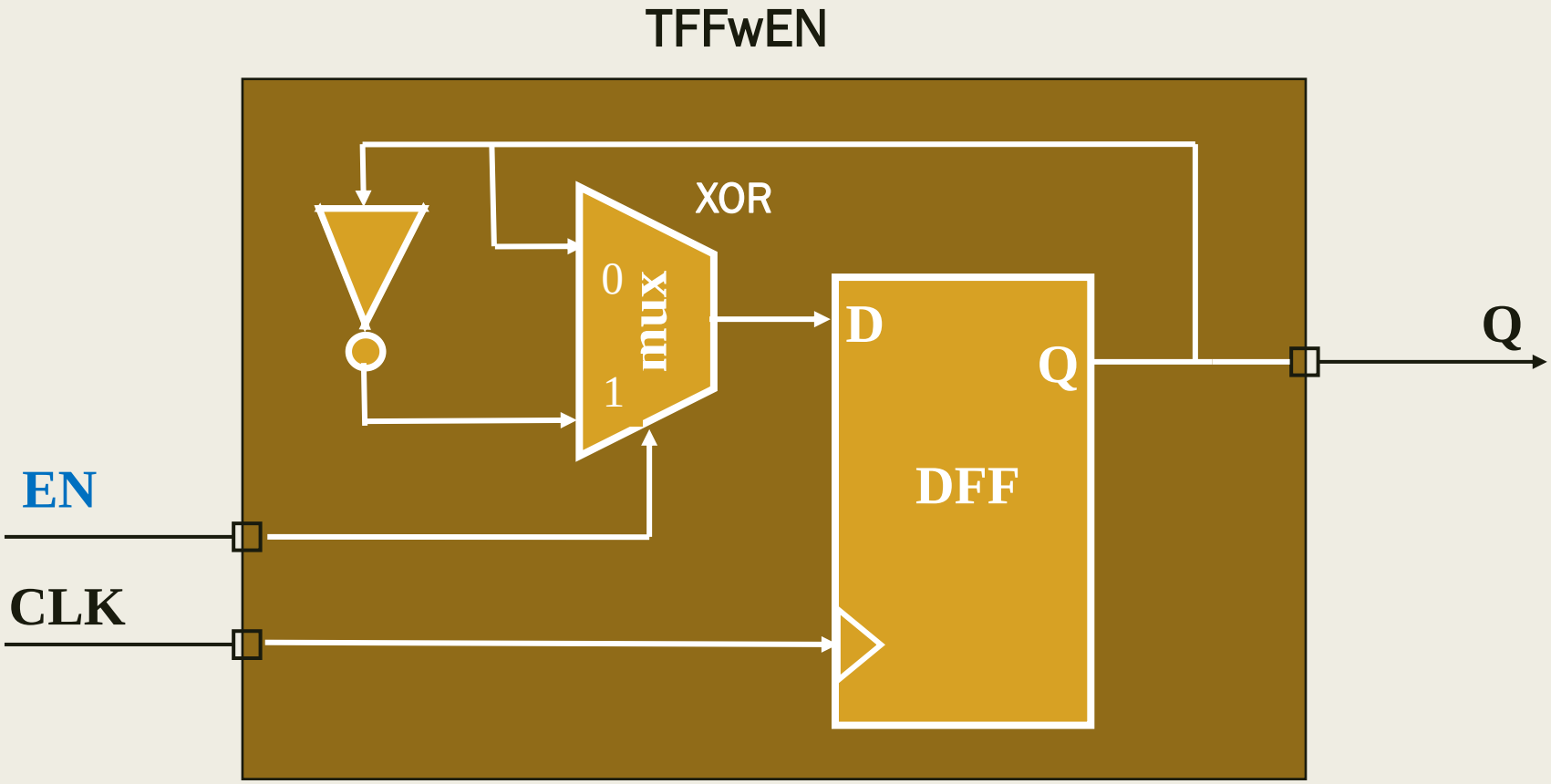
- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



T Flip-Flop with Enable στη VHDL
Περιγραφή συμπεριφοράς



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

T Flip-Flop with Enable στη VHDL

Περιγραφή συμπεριφοράς

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

Απαίτηση για εσωτερικό σήμα Q_in

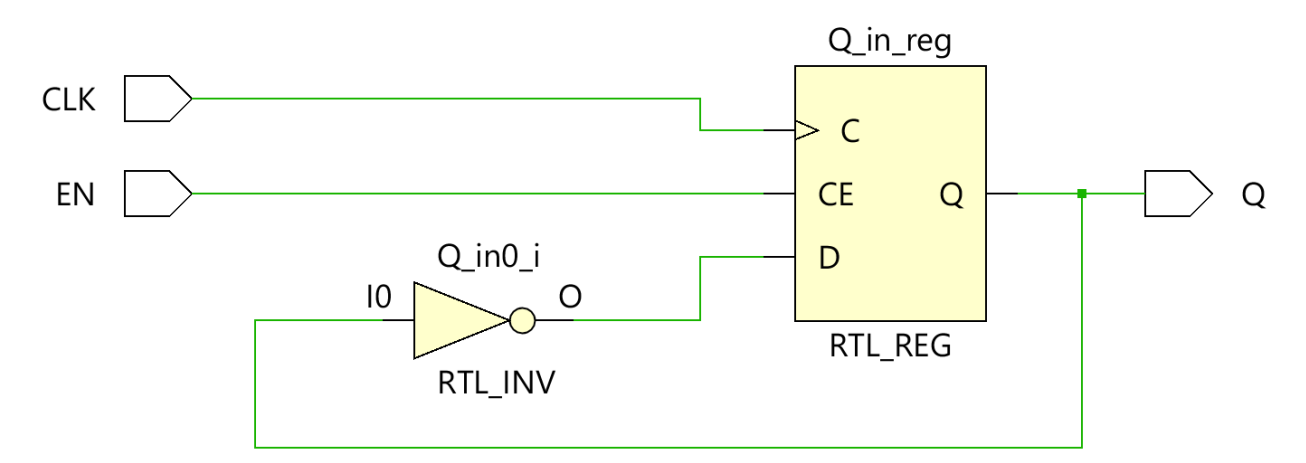
Το EN δεν τοποθετείται στη λίστα ευαισθησίας

Η συνθήκη του σύγχρονου EN εξετάζεται μετά τη συνθήκη του CLK

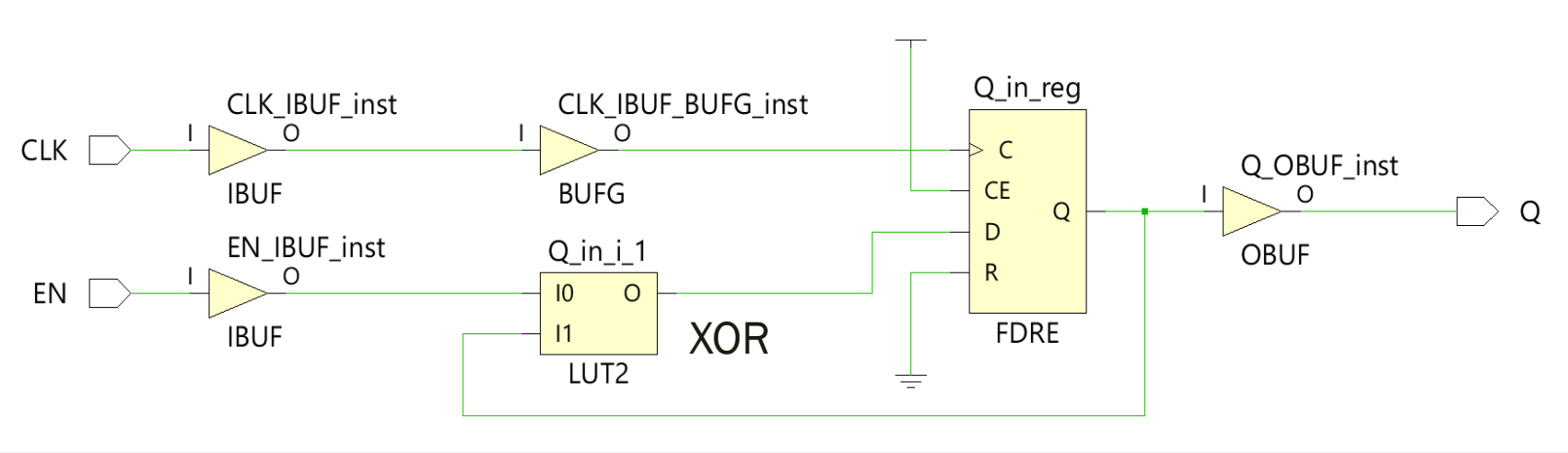
Ανάθεση τιμής στο Q εκτός process

T Flip-Flop with Enable στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



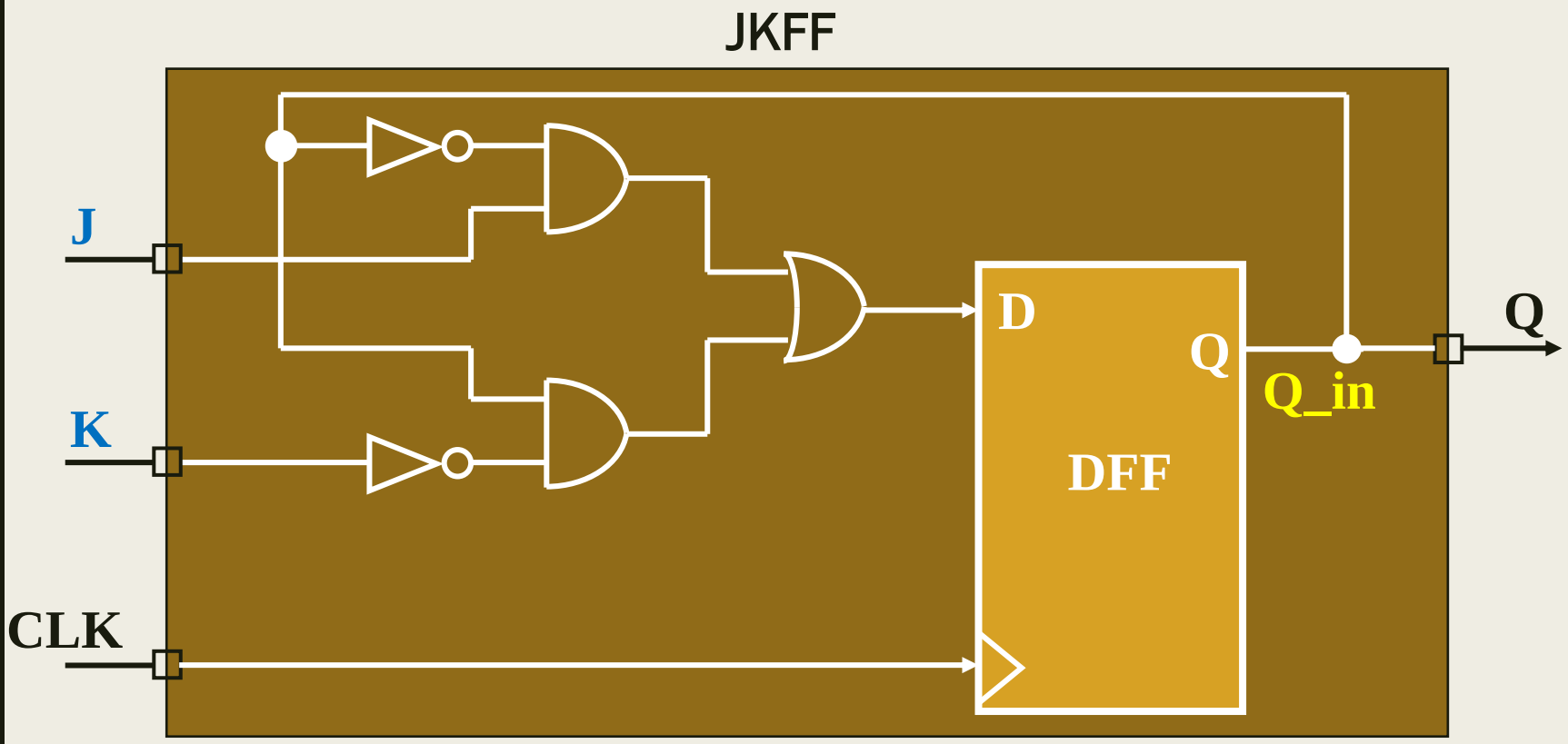
JK Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

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

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

JK Flip-Flop στη VHDL

Περιγραφή συμπεριφοράς



Τα σήματα J και K είναι σύγχρονα

Υλοποιείται η εξίσωση Boole:

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

JK Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

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

Απαίτηση για εσωτερικό σήμα Q_in

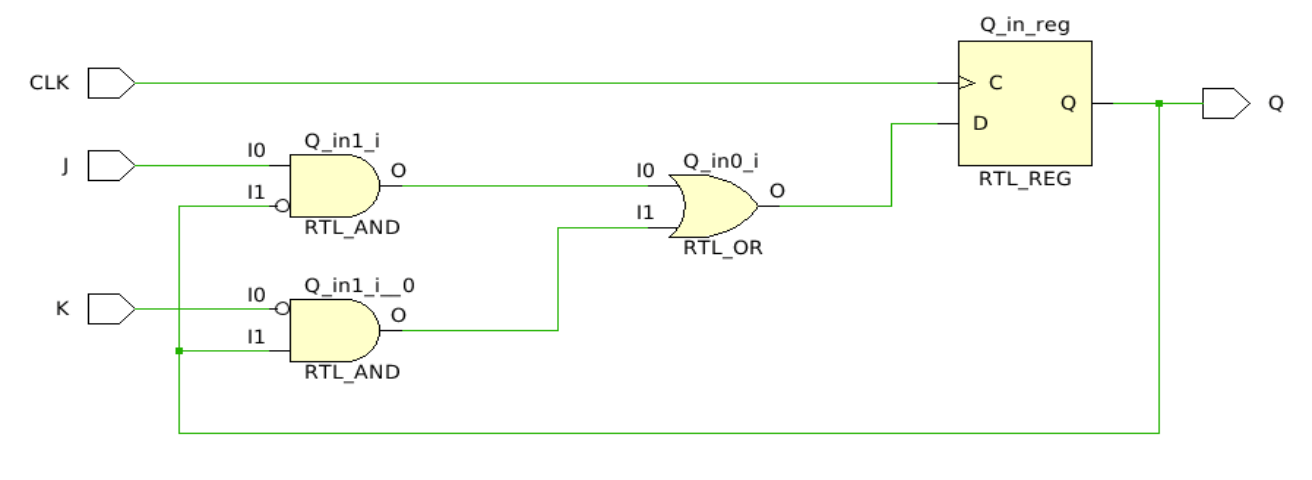
Τα J, K δεν τοποθετούνται στη λίστα ευαισθησίας

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

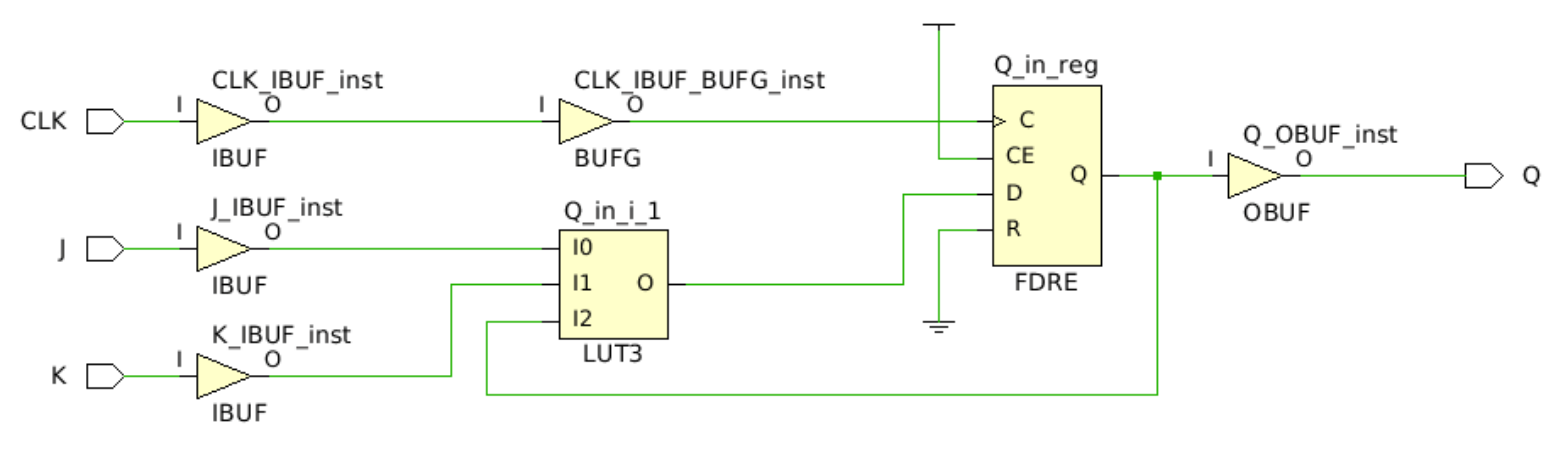
Ανάθεση τιμής στο Q εκτός process

JK Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL – Υλοποιείται η εξίσωση Boole



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



Επιλεγμένη άσκηση: το AB Flip-Flop στη VHDL

Περιγραφή συμπεριφοράς

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

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

AB	Λειτουργία
00	HOLD
01	TOGGLE
10	RESET
11	LOAD

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

Απαίτηση για εσωτερικό σήμα Q_in

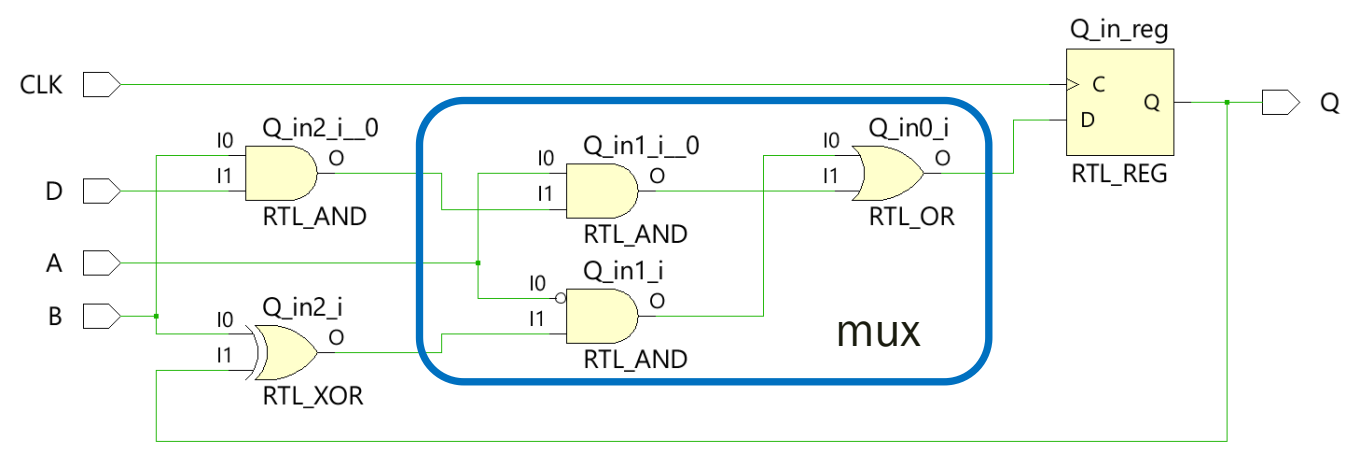
Τα A, B δεν τοποθετούνται στη λίστα ευαισθησίας

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

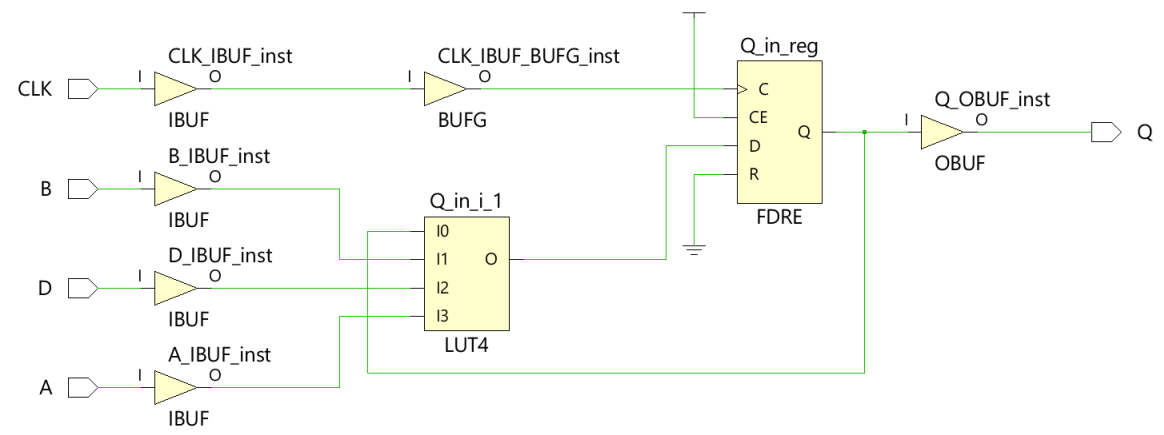
Ανάθεση τιμής στο Q εκτός process

AB Flip-Flop στη VHDL
Περιγραφή Συμπεριφοράς

- Σχηματικό διάγραμμα RTL – Υλοποιείται η εξίσωση Boole



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



Αριθμοί στη VHDL

- Στη VHDL, οι αριθμοί τύπου STD_LOGIC γράφονται στο **δυαδικό σύστημα** και **περικλείονται από μονά εισαγωγικά**: '0' και '1'
- Η μορφή για τη δήλωση σταθερών τύπου STD_LOGIC_VECTOR είναι **NB"value"**, όπου
  - *το **N** είναι το μέγεθος σε bit,*
    - Αν δεν ορίζεται το μέγεθος, θεωρείται ότι ο αριθμός έχει μέγεθος που ταιριάζει με το πλήθος των bit που καθορίζεται στην τιμή
  - *το **B** είναι ένα γράμμα που αναπαριστά τη βάση,*
    - Η VHDL υποστηρίζει το δυαδικό (B), το οκταδικό (O), το δεκαδικό (D) και το δεκαεξαδικό (X) σύστημα
    - Αν παραλειφθεί η βάση, η προεπιλογή είναι το **δυαδικό σύστημα**
  - *και το **value** μέσα σε **διπλά εισαγωγικά** ορίζει την τιμή*
- Οι **χαρακτήρες υπογράμμισης (_)** αγνοούνται ανάμεσα στους αριθμούς
  - *χωρίζουν **πολύ μεγάλους αριθμούς** σε πιο ευανάγνωστα μέρη*

Αριθμοί στη VHDL

Αριθμοί	Bit	Βάση	Τιμή	Αποθηκεύεται ως
3B"101"	3	2	5	101
B"11"	2	2	3	11
8B"11"	8	2	3	00000011
8B"1010_1011	8	2	171	10101011
3D"6"	3	10	6	110
6O"42"	6	8	34	100010
8X"AB"	8	16	171	10101011
"101"	3	2	5	101
B"101	3	2	5	101
X"AB"	8	16	171	10101011

Ο τύπος 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 είτε **αύξουσα (μεγάλου άκρου)***
    STD_LOGIC_VECTOR (0 to 7)
    *είτε **φθίνουσα (μικρού άκρου)***
    STD_LOGIC_VECTOR (7 downto 0)
- Οι δείκτες των στοιχείων του array είναι τύπου natural
- Προσοχή, **δεν είναι ακέραιος δυαδικός αριθμός**
- Για να χρησιμοποιηθεί δηλώνουμε:

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

Ο τύπος std_logic_vector

- Δήλωση τιμών για το 8-ψήφιο λογικό διάνυσμα V
  - *V <= "ZZZZ0000" ή V <= (V(3 downto 0) => '0', others => 'Z')*
  - *V <= (others => '0') -- όλα-0*
  - *V <= (others => ‘1') -- όλα-1*
- 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, αλλά ως μία από τις εννέα τιμές του std_logic

Λογικές πράξεις με λογικά διανύσματα
Περιγραφή Dataflow

```
entity INV4 is port (
    X: in STD_LOGIC_VECTOR (0 to 3);
    Y: out STD_LOGIC_VECTOR (0 to 3));
end INV4;
architecture DATAFLOW of INV4 is
begin
    Y <= not X; --Λογικές πράξεις με λογικά διανύσματα
end DATAFLOW;
```

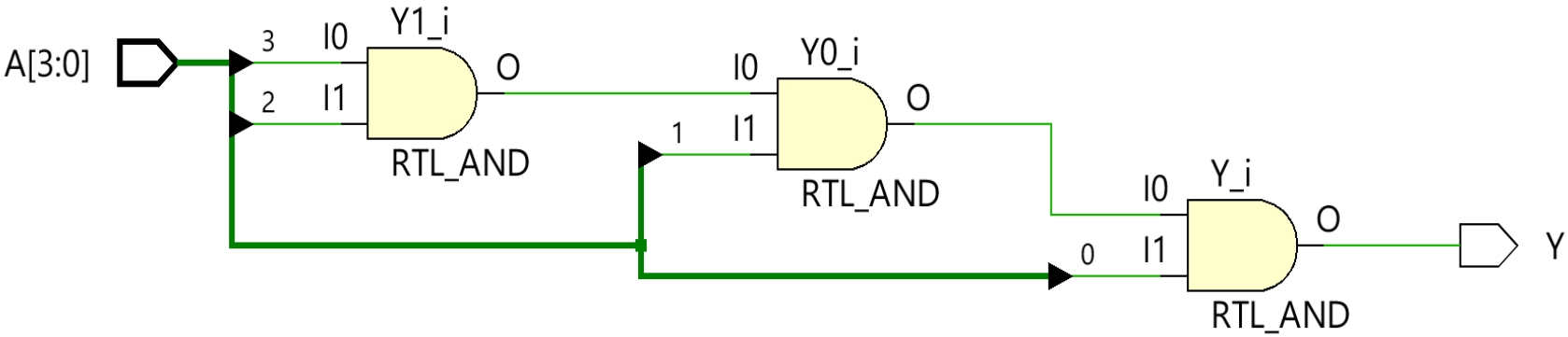
Η αρχιτεκτονική ισοδυναμεί με :

```
architecture INV4_DFL of INV4 is
begin
    Y(0) <= not X(0);
    Y(1) <= not X(1);
    Y(2) <= not X(2);
    Y(3) <= not X(3);
end INV4_DFL;
```

Λογικές πράξεις με λογικά διανύσματα
Περιγραφή Dataflow

```
entity AND4 is port (
    A: in STD_LOGIC_VECTOR (0 to 3);
    Y: out STD_LOGIC;
end AND4;
architecture DATAFLOW of AND4 is
begin
    Y <= A(3) and A(2) and A(1) and A(0);
end DATAFLOW;
```

Σχηματικό διάγραμμα RTL :



Προσοχή! Στο Vivado δεν υποστηρίζεται η αντίστοιχη εντολή `Y <= and A` της VHDL 2008

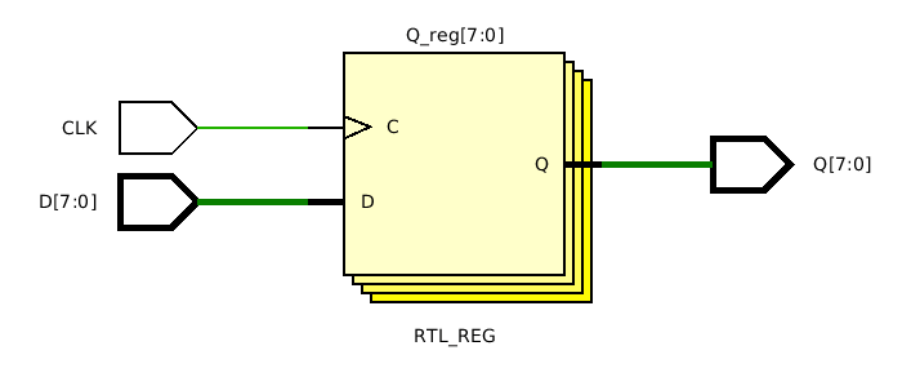
Καταχωρητής των 8 bit στη VHDL
Περιγραφή συμπεριφοράς

```
entity REG1_8 is
    port (
        CLK: in STD_LOGIC;
        D: in  STD_LOGIC_VECTOR (7 downto 0);
        Q: out STD_LOGIC_VECTOR (7 downto 0));
end REG1_8;
architecture REG1_8_BEH of REG1_8 is
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      Q <= D;
    end if;
  end process;
end REG1_8_BEH;
```

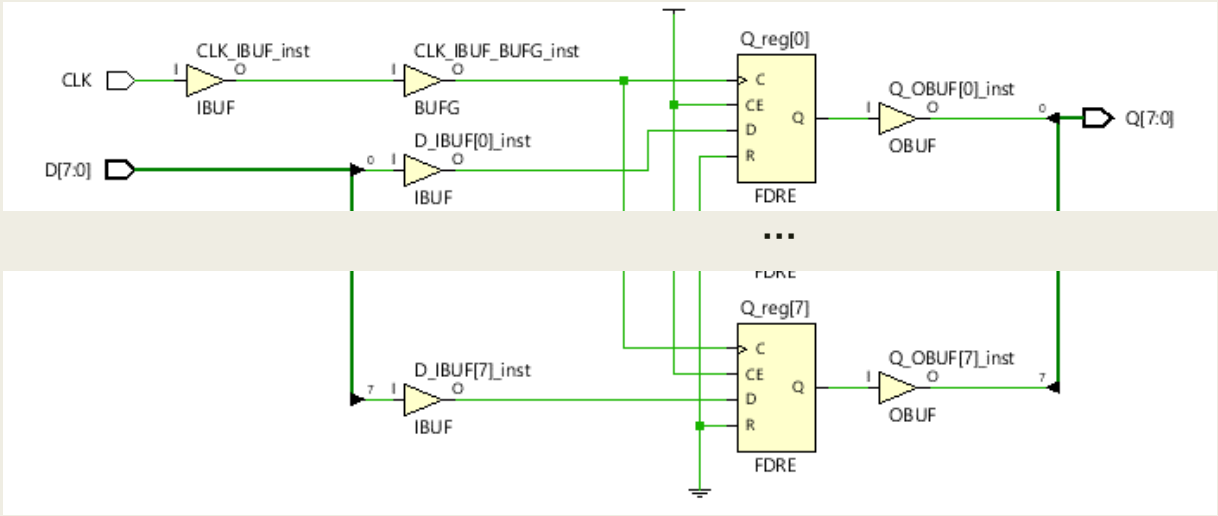
Διαφέρει από το D Flip-Flop στον τύπο των σημάτων D και Q

Καταχωρητής των 8 bit στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



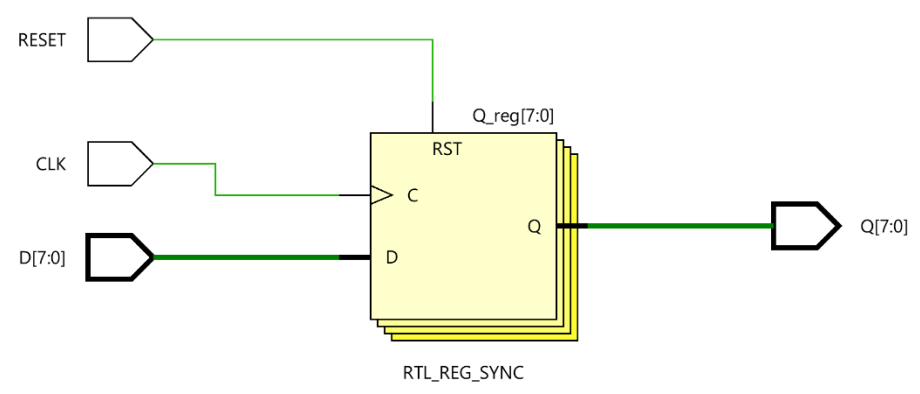
Καταχωρητής των 8 bit με Reset στη VHDL
Περιγραφή συμπεριφοράς

```
entity REG8r is
    port (
        CLK, RESET: in STD_LOGIC;
        D: in  STD_LOGIC_VECTOR (7 downto 0);
        Q: out STD_LOGIC_VECTOR (7 downto 0));
end REG8r;
architecture REG8r_BEH of REG8r is
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      if (RESET = '1') then
       Q <= (others => '0');
      else Q <= D;
      end if;
    end if;
  end process;
end REG8r_BEH;
```

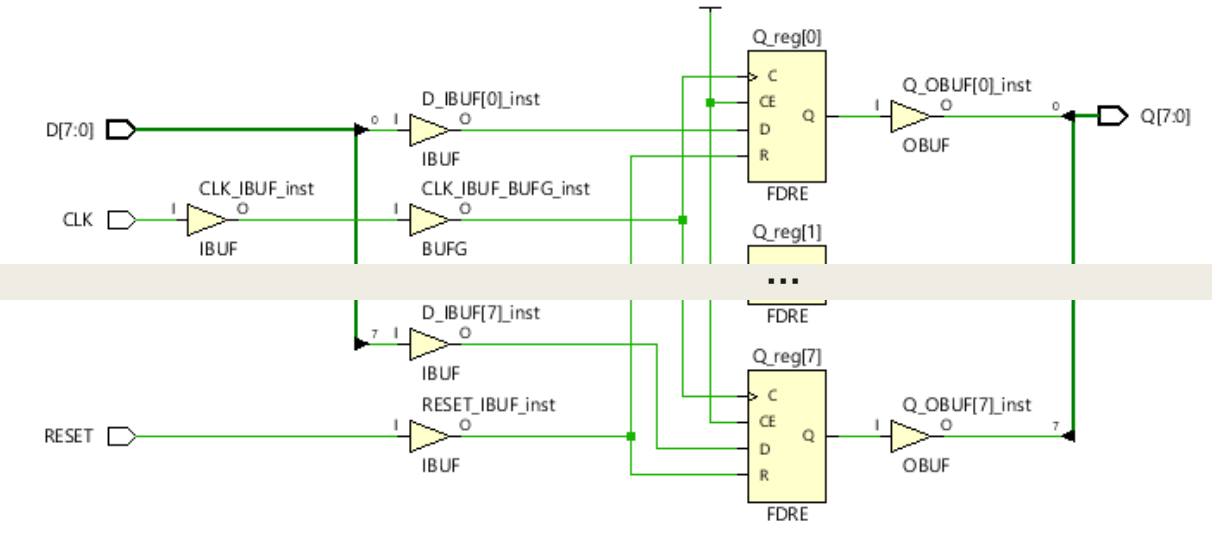
Συντομογραφία του όλα-0

Καταχωρητής των 8 bit με Reset στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Καταχωρητής των 8 bit με Reset και WE στη VHDL

Περιγραφή συμπεριφοράς

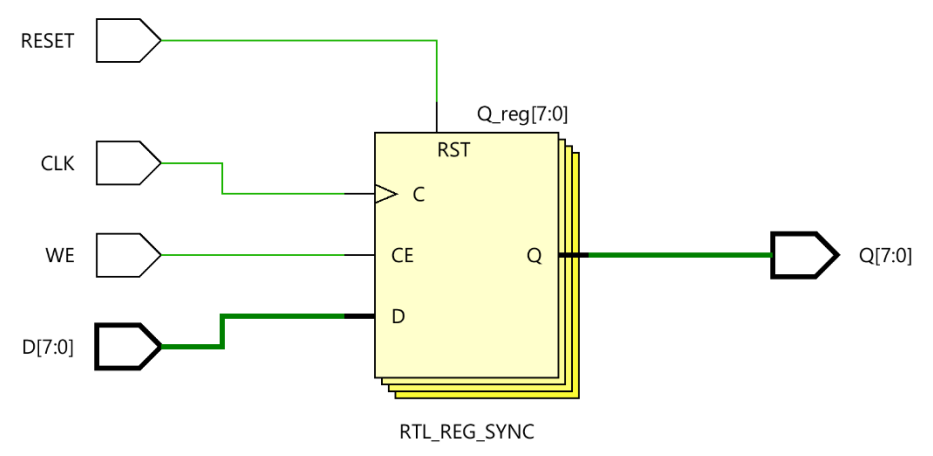
```
entity REG8rwe is
    port (
        CLK, RESET, WE: in STD_LOGIC;
        D: in  STD_LOGIC_VECTOR (7 downto 0);
        Q: out STD_LOGIC_VECTOR (7 downto 0));
end REG8rwe;
architecture REG8rwe_BEH of REG8rwe is
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      if (RESET = '1') then
      Q <= (others => '0');
      elsif (WE = '1') then
      Q <= D;
      end if;
    end if;
  end process;
end REG8rwe_BEH;
```

Συντομογραφία του όλα-0

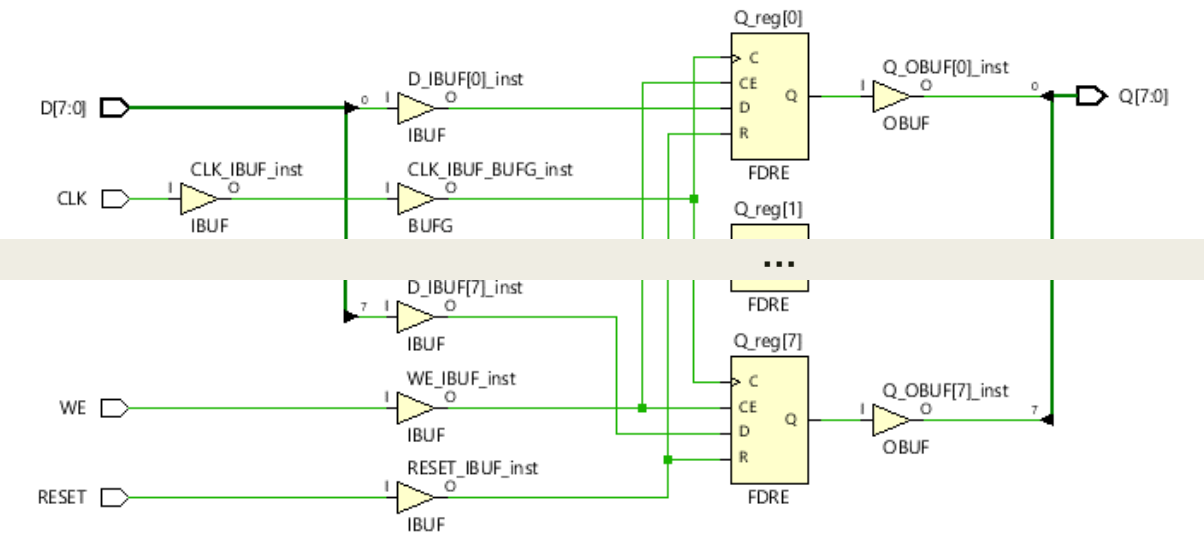
Καταχωρητής των 8 bit με Reset και WE στη VHDL

Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Συνένωση Σημάτων

- **Τελεστής συνένωσης σημάτων** (concatenation) **&**
- Εάν τα διανύσματα **`A(1 downto 0)`** και **`B(0 to 2)`** συνενωθούν στο διάνυσμα **`Y(0 to 4)`** με την εντολή **`Y <= A & B`**, τότε ισχύει:
  - ```
    Y(0) <= A(1)
    Y(1) <= A(0)
    Y(2) <= B(0)
    Y(3) <= B(1)
    Y(4) <= B(2)
    ```
    ```

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Η εντολή CASE

```
case signal (vector, expression) is
    when VALUE_1 =>
        sequential_statement_1;
    ...
    when VALUE_k =>
        sequential_statement_k;
    when others =>
        sequential_statements_k+1;
end case;
```

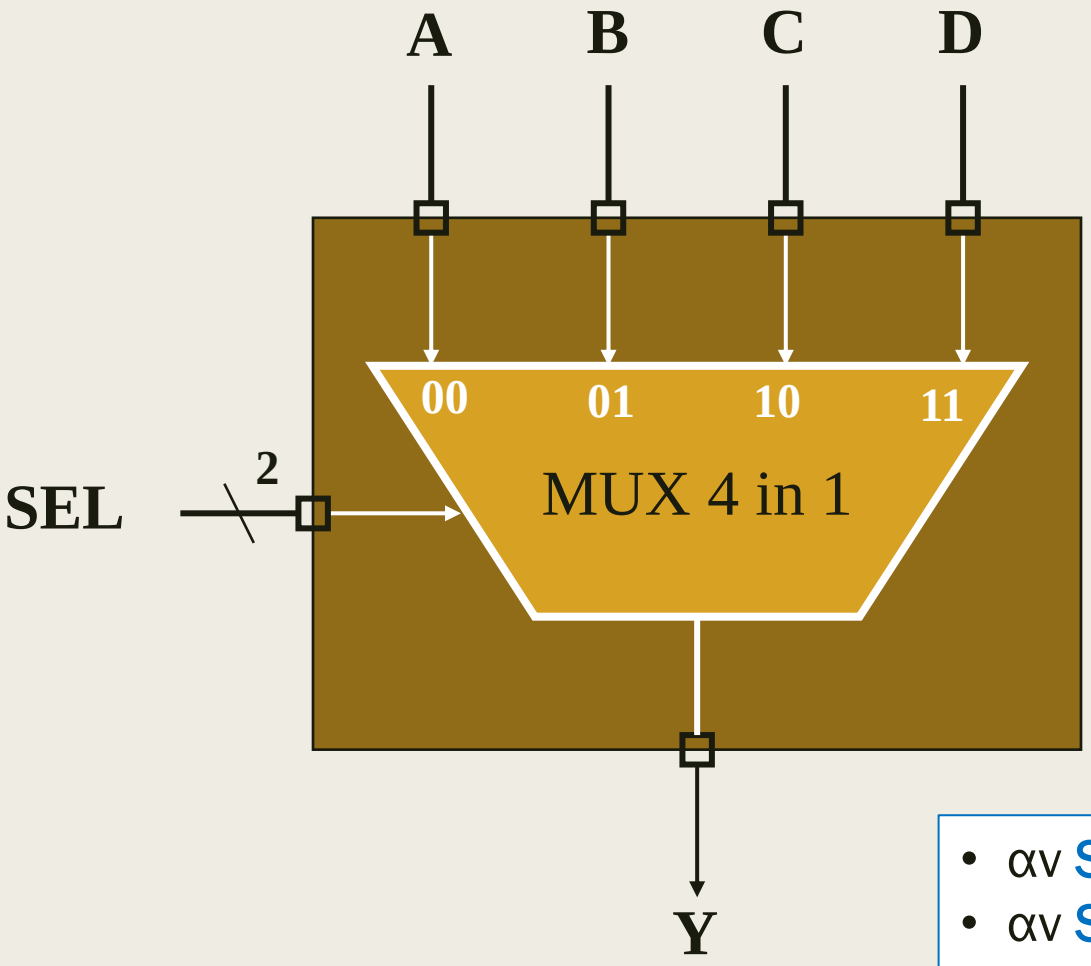
Η εντολή CASE, στην απλή της μορφή, εξετάζει παράλληλα την τιμή που έχει ένα σήμα, διάνυσμα ή μεταβλητή και, για κάθε διαφορετική τιμή value_i, εκτελεί την αντίστοιχη ακολουθιακή εντολή i.
Κατά τη σύνθεση χρησιμοποιείται ένας πολυπλέκτης.

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Η εντολή CASE

- Για την ορθή προσομοίωση και η σύνθεση μίας εντολή CASE απαιτείται:
  - *να εξετάζονται **όλες** οι τιμές*
  - *κάθε τιμή να αναφέρεται **μόνο μία φορά***
  - *η τιμή να παραμένει **τοπικά στατική** (locally static)*
- Η φράση **when others** χρησιμοποιείται πάντα σαν τελευταία συνθήκη, ώστε να καλύπτονται **όλες οι δυνατές τιμές** του std_logic
  - *συνδυάζεται με την εντολή **null***
  - *συνδυάζεται με την **ανάθεση αδιάφορων τιμών** (don't care) '-' ή 'X', που είναι το ίδιο για τη σύνθεση*
    - συνήθως οδηγεί σε **μεγαλύτερη απλοποίηση του κυκλώματος**
- Στη συνδυαστική λογική, για να αποφεύγεται η **ελλιπής ανάθεση τιμών** πριν την εντολή CASE **βάζουμε αρχικές τιμές σε όλα τα σήματα που μετέχουν σε αυτή**

Πολυπλέκτης 4 σε 1 στη VHDL

Περιγραφή συμπεριφοράς



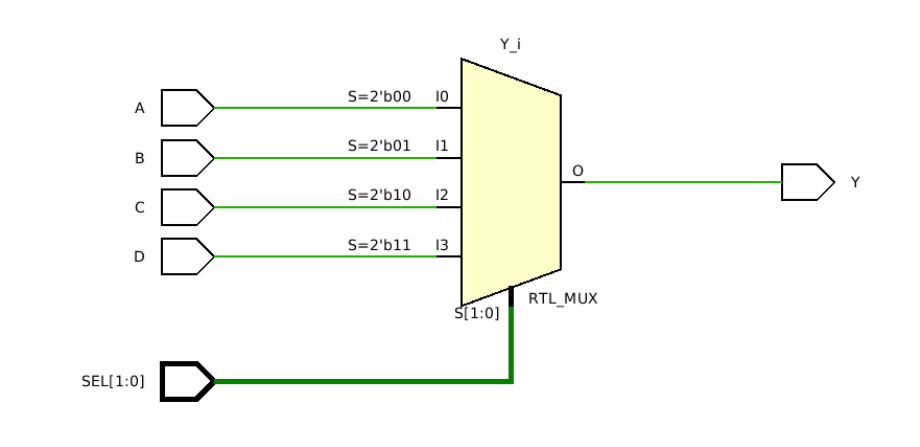
- αν $SEL_{1:0} = 00$, τότε $Y = A$
- αν $SEL_{1:0} = 01$, τότε $Y = B$
- αν $SEL_{1:0} = 10$, τότε $Y = C$
- αν $SEL_{1:0} = 11$, τότε $Y = D$

Πολυπλέκτης 4 σε 1 στη VHDL
Περιγραφή συμπεριφοράς

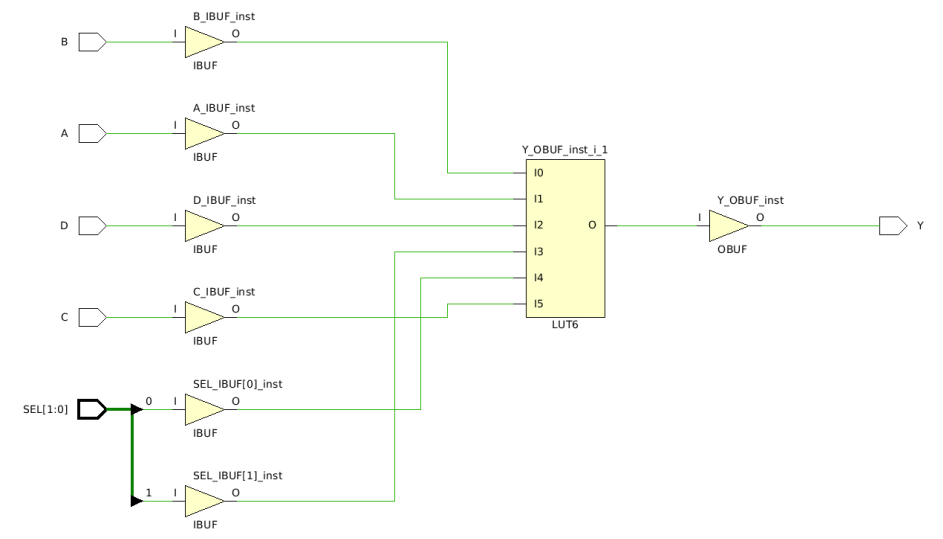
```
entity MUX_4in1 is
  port (
    SEL: in STD_LOGIC_VECTOR (1 downto 0);
    A, B, C, D: in STD_LOGIC;
    Y: out STD_LOGIC);
end MUX_4in1;
architecture MUX_4in1_BEH of MUX_4in1 is
begin
  process (SEL, A, B, C, D)
    begin
      case SEL is
        when  "00" => Y <= A;
        when  "01" => Y <= B;
        when  "10" => Y <= C;
        when  "11" => Y <= D;
        when  others => Y <= '-'; -- don't care
      end case;
  end process;
end MUX_4in1_BEH;
```

Πολυπλέκτης 4 σε 1 στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL

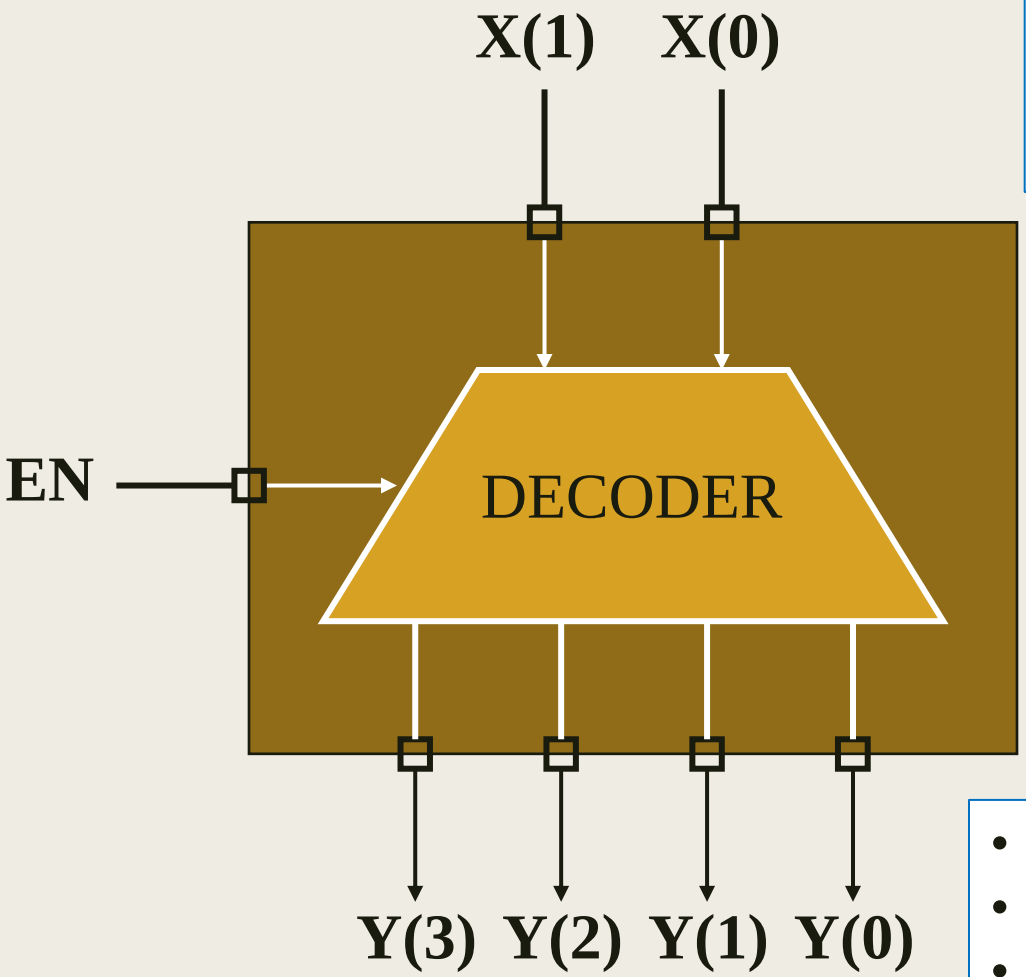


- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Δυαδικός αποκωδικοποιητής 2 σε 4 στη VHDL

Περιγραφή συμπεριφοράς



Σήμα έγκρισης EN (enable)

- αν EN = 1, κανονική λειτουργία
- αν EN = 0, τότε $Y_{3:0} = 0000$

- αν $X_{1:0} = 00$, τότε $Y_{3:0} = 0001$
- αν $X_{1:0} = 01$, τότε $Y_{3:0} = 0010$
- αν $X_{1:0} = 10$, τότε $Y_{3:0} = 0100$
- αν $X_{1:0} = 11$, τότε $Y_{3:0} = 1000$

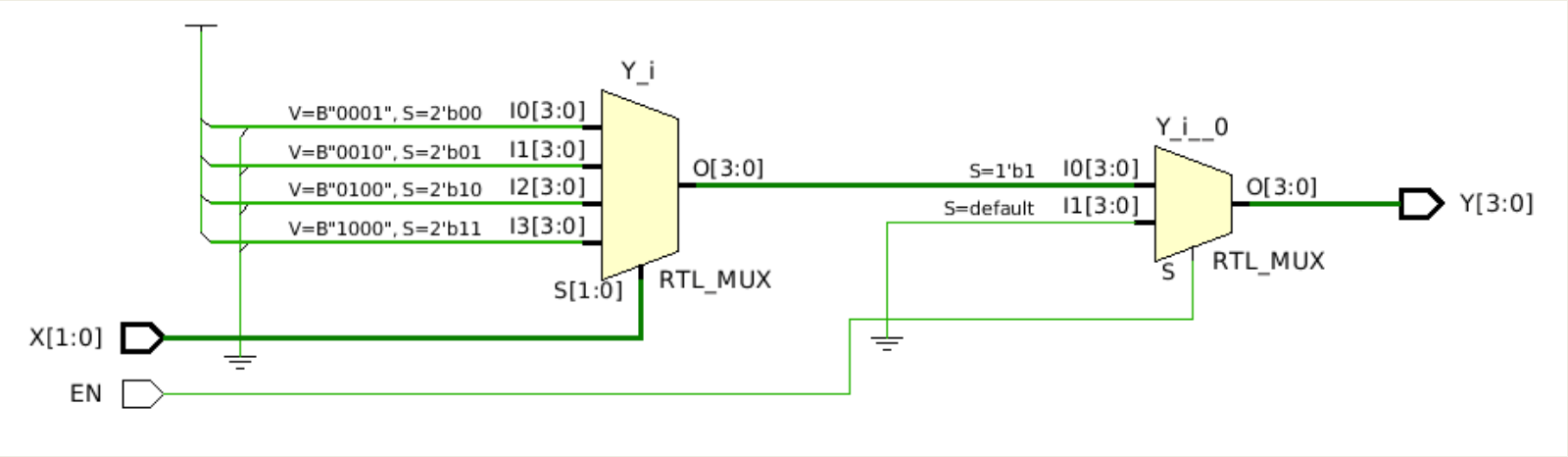
Δυαδικός αποκωδικοποιητής 2 σε 4 στη VHDL

Περιγραφή συμπεριφοράς

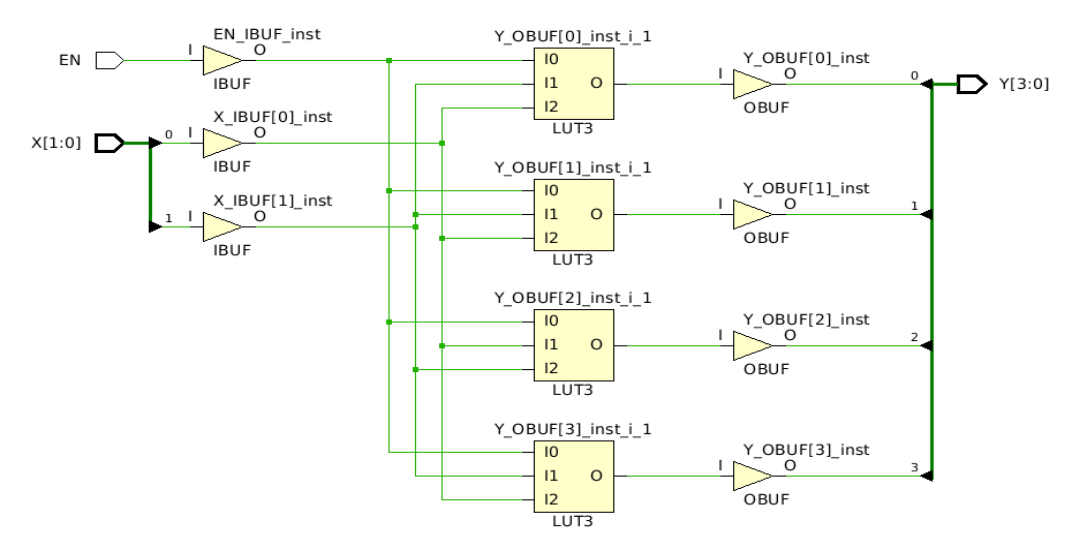
```
entity DEC_2in4 is
  port (
    X: in STD_LOGIC_VECTOR (1 downto 0);
    Y: out STD_LOGIC_VECTOR (3 downto 0);
    EN: in STD_LOGIC);
end DEC_2in4;
architecture DEC_2in4_BEH of DEC_2in4 is
begin
  process (EN, X)
    begin
      Y <= "0000";
      if (EN = '1') then
        case X is
          when  "00" => Y <= "0001";
          when  "01" => Y <= "0010";
          when  "10" => Y <= "0100";
          when  "11" => Y <= "1000";
          when  others => null;
        end case;
      else
        null;
      end if;
  end process;
end DEC_2in4_BEH;
```

Δυαδικός αποκωδικοποιητής 2 σε 4 στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL

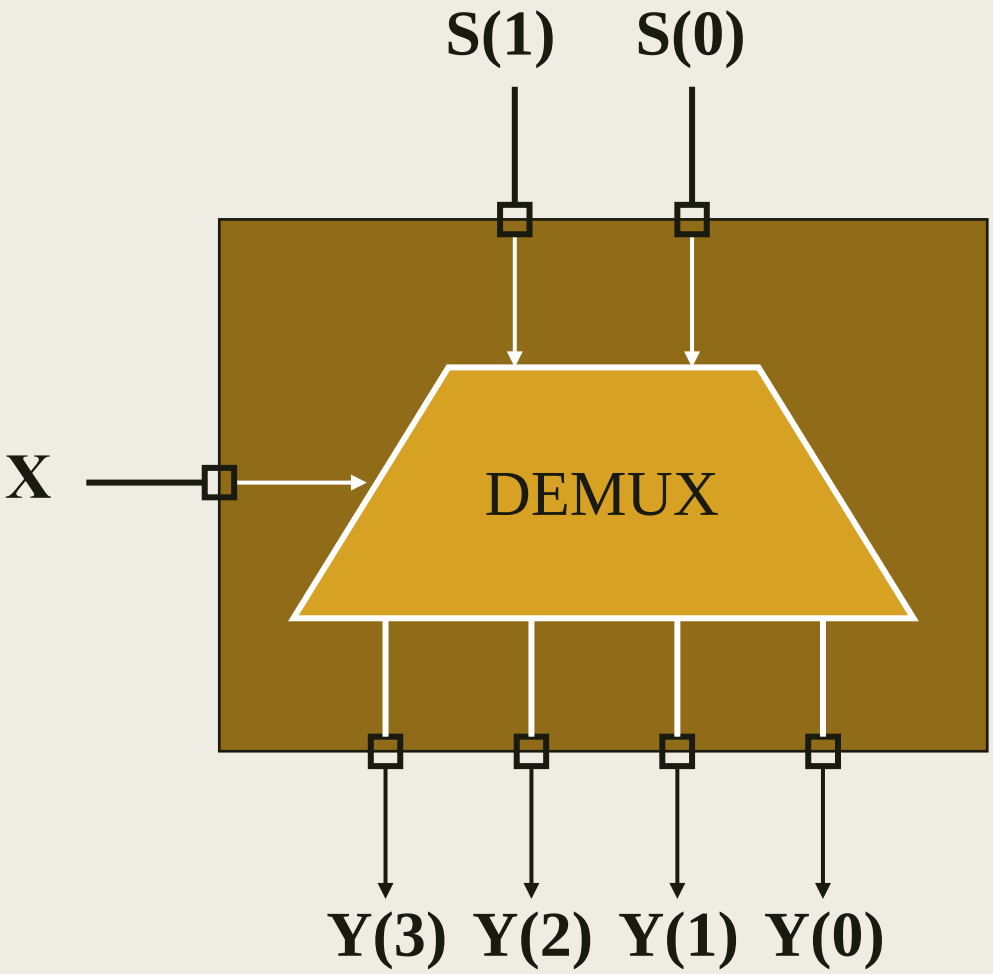


- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Υλοποίηση πυλών AND με LUT3

Αποπλέκτης 1 σε 4 στη VHDL
Περιγραφή συμπεριφοράς



- Ο **δυαδικός αποκωδικοποιητής 2 σε 4 με έγκριση** χρησιμοποιείται και ως **αποπλέκτης 1 σε 4**
  - *Το **σήμα EN** του αποκωδικοποιητή είναι η **είσοδος δεδομένων X** του αποπλέκτη*
  - *Οι **2 είσοδοι δεδομένων** του αποκωδικοποιητή είναι τα **2 σήματα επιλογής** του αποπλέκτη*
  - *Οι **4 έξοδοι «μοναδικού σημαντικού»** του αποκωδικοποιητή είναι οι **4 έξοδοι δεδομένων** του αποπλέκτη*

Αποπλέκτης 1 σε 4 στη VHDL
Περιγραφή συμπεριφοράς

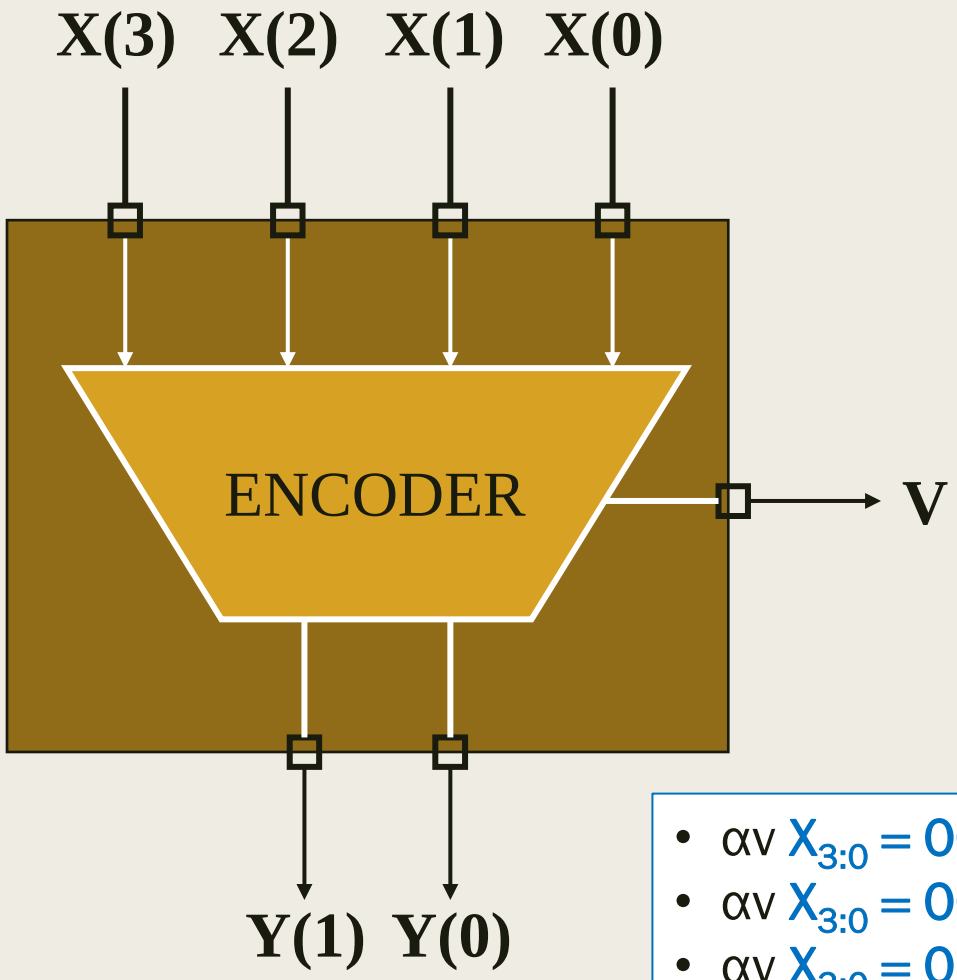
```
entity DEMUX_1in4 is
  port (
    S: in STD_LOGIC_VECTOR (1 downto 0);
    Y: out STD_LOGIC_VECTOR (3 downto 0);
    X: in STD_LOGIC);
end DEMUX_1in4;
architecture DEMUX_1in4_BEH of DEMUX_1in4 is
begin
  process (X, S)
    begin
      Y <= "0000";
      if (X = '1') then
        case S is
          when  "00" => Y <= "0001";
          when  "01" => Y <= "0010";
          when  "10" => Y <= "0100";
          when  "11" => Y <= "1000";
          when  others => null;
        end case;
      else
        null;
      end if;
  end process;
end DEMUX_1in4_BEH;
```

Δυαδική κωδικοποίηση

- Ένας δυαδικός κώδικας των *n* bit έχει 2^n κωδικές λέξεις
- Για να αναπαραστήσουμε ***N*** πιθανές τιμές σε έναν δυαδικό κώδικα
  - *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)

Δυαδικός κωδικοποιητής 4 σε 2 στη VHDL

Περιγραφή συμπεριφοράς



Η έξοδος εγκυρότητας V (valid) ξεχωρίζει τις κωδικές εισόδους από τις μη κωδικές εισόδους

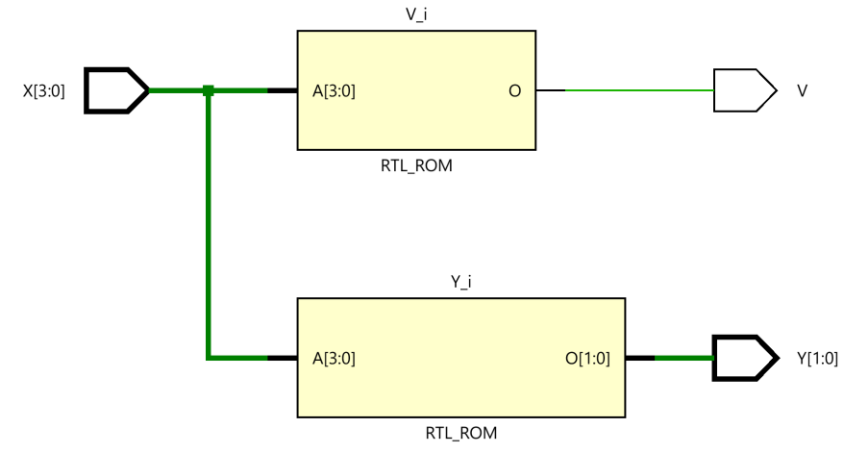
- αν $X_{3:0} = 0001$, τότε $Y_{1:0} = 00$ και $V = 1$
- αν $X_{3:0} = 0010$, τότε $Y_{1:0} = 01$ και $V = 1$
- αν $X_{3:0} = 0100$, τότε $Y_{1:0} = 10$ και $V = 1$
- αν $X_{3:0} = 1000$, τότε $Y_{1:0} = 11$ και $V = 1$
- αλλιώς $Y_{1:0} = 00$ και $V = 0$

Δυαδικός κωδικοποιητής 4 σε 2 στη VHDL
Περιγραφή συμπεριφοράς

```
entity ENC_4in2 is
  port (
    X: in STD_LOGIC_VECTOR (3 downto 0);
    Y: out STD_LOGIC_VECTOR (1 downto 0);
    V: out STD_LOGIC);
end ENC_4in2;
architecture ENC_4in2_BEH of ENC_4in2 is
begin
  process (X)
    begin
      Y <= "00"; V <='0';
      case X is
        when  "0001" => Y <= "00"; V <= '1';
        when  "0010" => Y <= "01"; V <= '1';
        when  "0100" => Y <= "10"; V <= '1';
        when  "1000" => Y <= "11"; V <= '1';
        when  others => null;
      end case;
    end process;
end ENC_4in2_BEH;
```

Δυαδικός κωδικοποιητής 4 σε 2 στη VHDL
Περιγραφή συμπεριφοράς

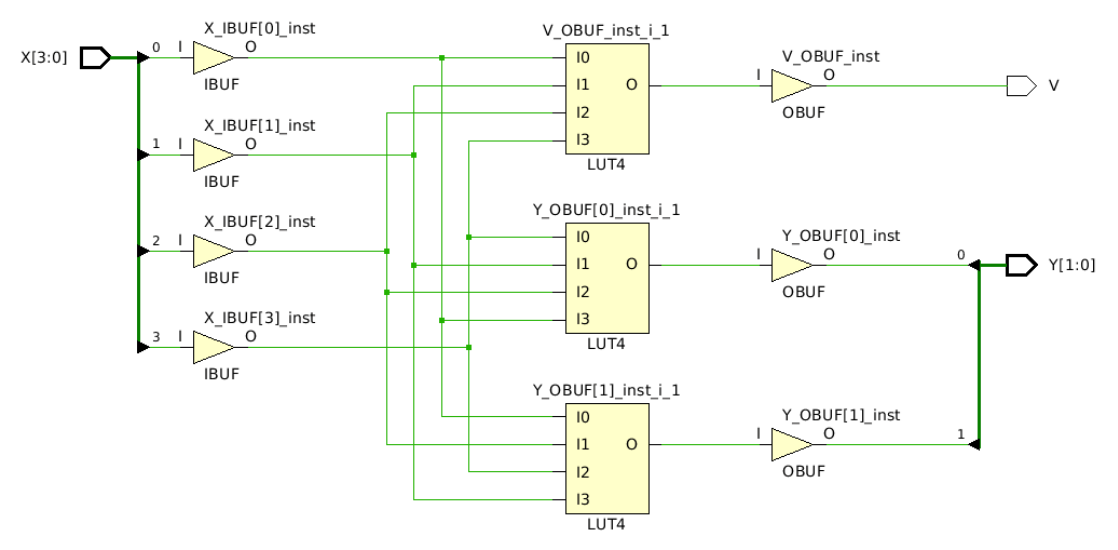
- Σχηματικό διάγραμμα RTL και ROM values



INIT	Value
INIT_DEFAULT	2'b00
INIT_1	2'b00
INIT_2	2'b01
INIT_4	2'b10
INIT_8	2'b11

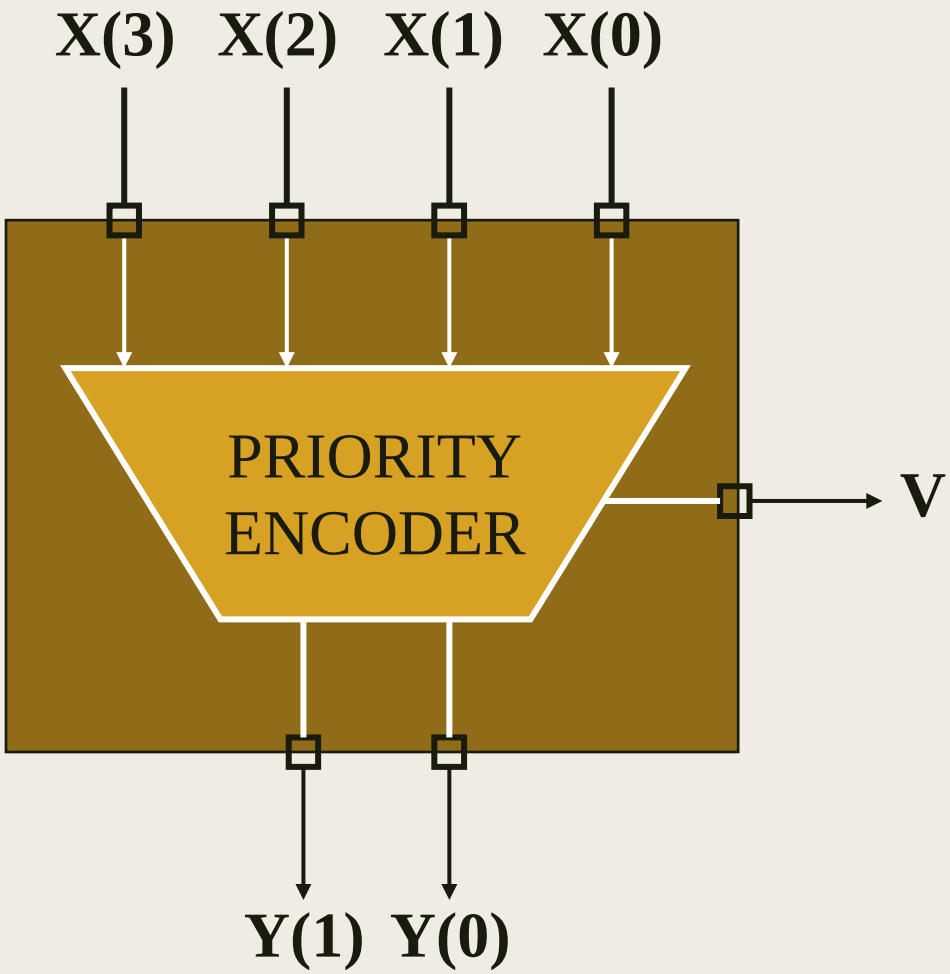
INIT	Value
INIT_DEFAULT	1'b0
INIT_1	1'b1
INIT_2	1'b1
INIT_4	1'b1
INIT_8	1'b1

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Κωδικοποιητής προτεραιότητας 4 σε 2 στη VHDL

Περιγραφή συμπεριφοράς



Οι είσοδοι είναι διατεταγμένες. Το X(3) έχει τη μεγαλύτερη προτεραιότητα και το X(0) τη μικρότερη προτεραιότητα

Η έξοδος εγκυρότητας V ξεχωρίζει την είσοδο όλα-0 από την κωδική είσοδο 0001

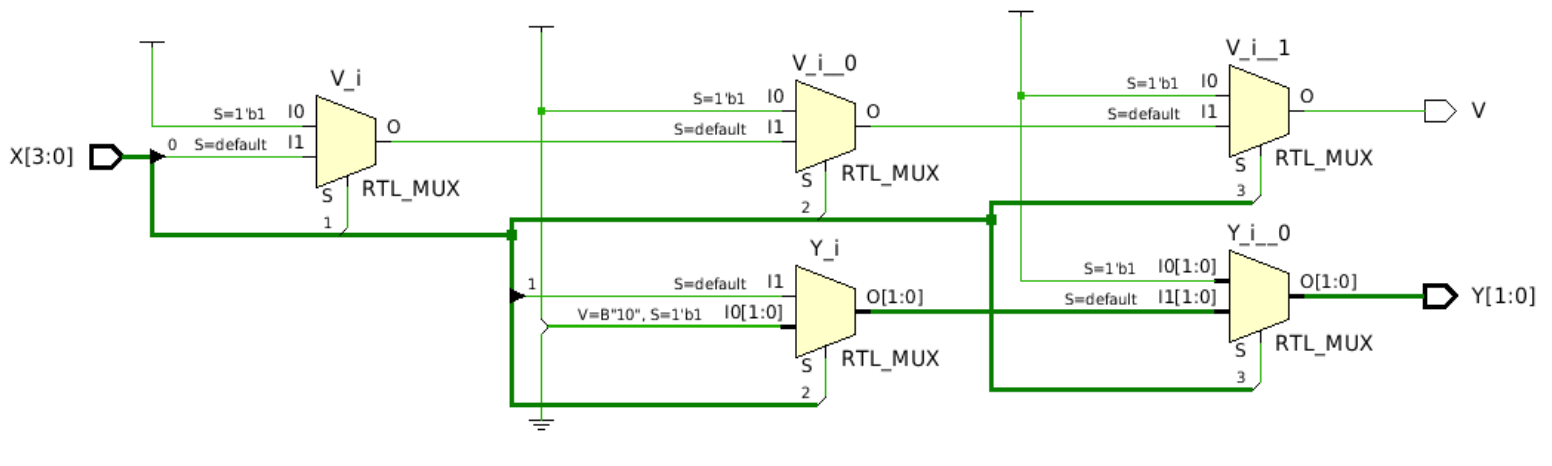
Κωδικοποιητής προτεραιότητας 4 σε 2 στη VHDL
Περιγραφή συμπεριφοράς

```
entity PENC_4in2 is
  port (
    X: in STD_LOGIC_VECTOR (3 downto 0);
    Y: out STD_LOGIC_VECTOR (1 downto 0);
    V: out STD_LOGIC);
end PENC_4in2;
architecture PENC_4in2_BEH of PENC_4in2 is
begin
  process (X)
    begin
      Y <= "00"; V <='0';
      if     (X(3) = '1') then Y <= "11"; V <= '1';
      elsif (X(2) = '1') then Y <= "10"; V <= '1';
      elsif (X(1) = '1') then Y <= "01"; V <= '1';
      elsif (X(0) = '1') then Y <= "00"; V <= '1';
      else null;
      end if;
    end process;
end PENC_4in2_BEH;
```

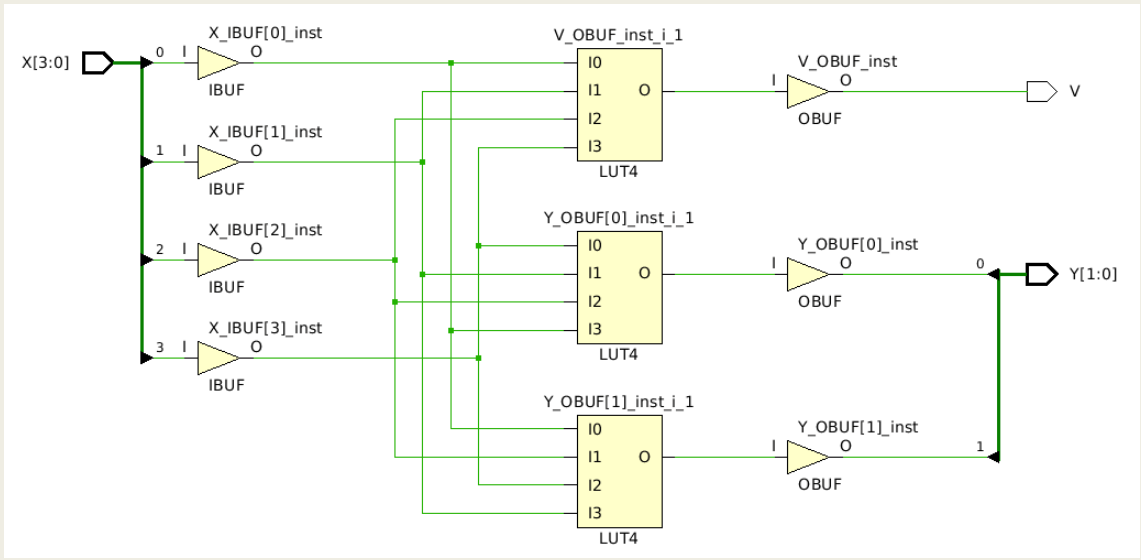
Η εντολή IF επιτρέπει να εξετασθεί μία διατεταγμένη σειρά από συνθήκες.

Κωδικοποιητής προτεραιότητας 4 σε 2 στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



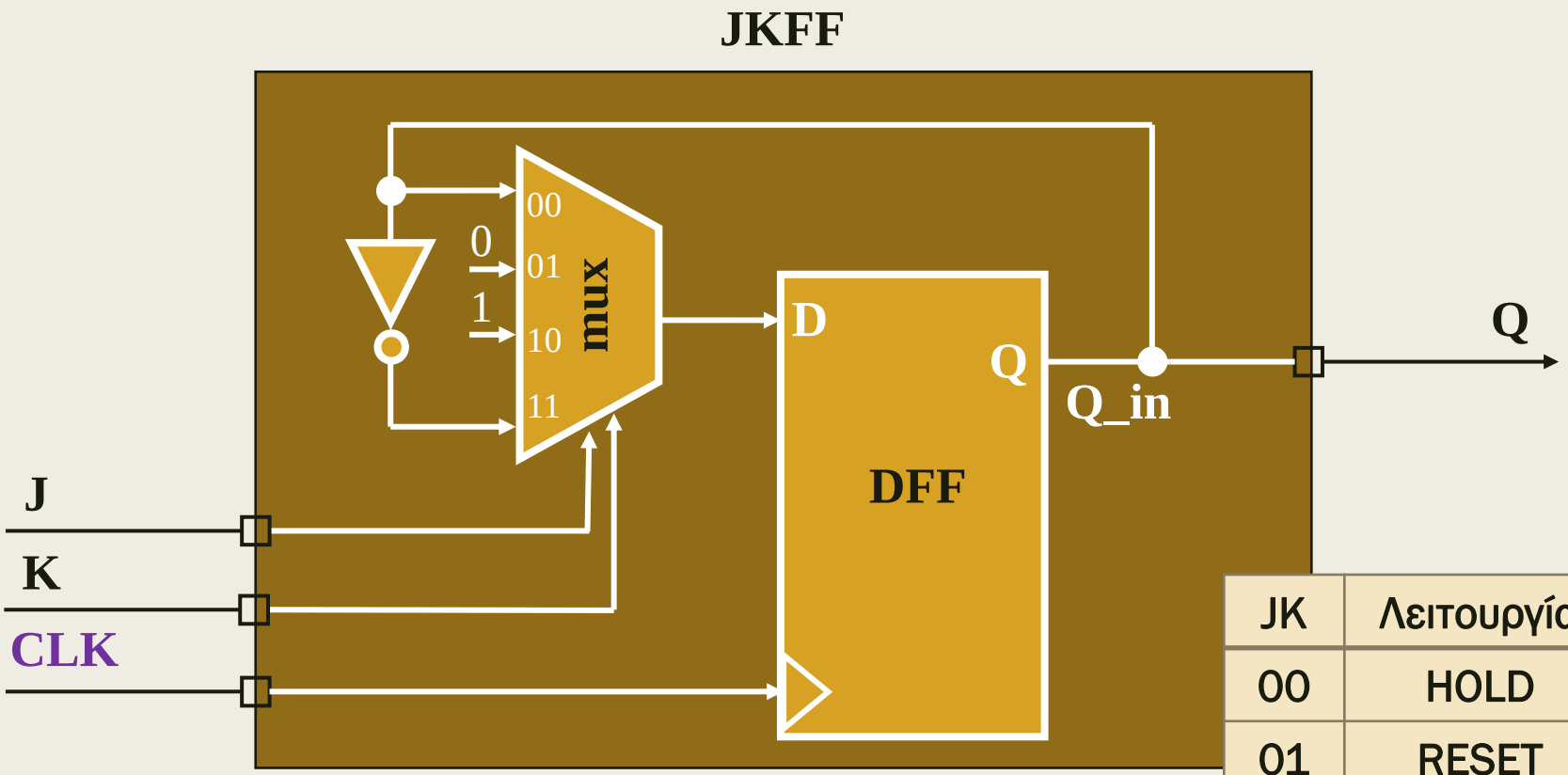
- Σχηματικό διάγραμμα σε τεχνολογία FPGA



JK Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

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

JK Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς



JK	Λειτουργία
00	HOLD
01	RESET
10	SET
11	TOGGLE

Τα σήματα J και K είναι σύγχρονα.
Υλοποιείται ο πίνακας λειτουργίας

JK Flip-Flop στη VHDL
Περιγραφή συμπεριφοράς

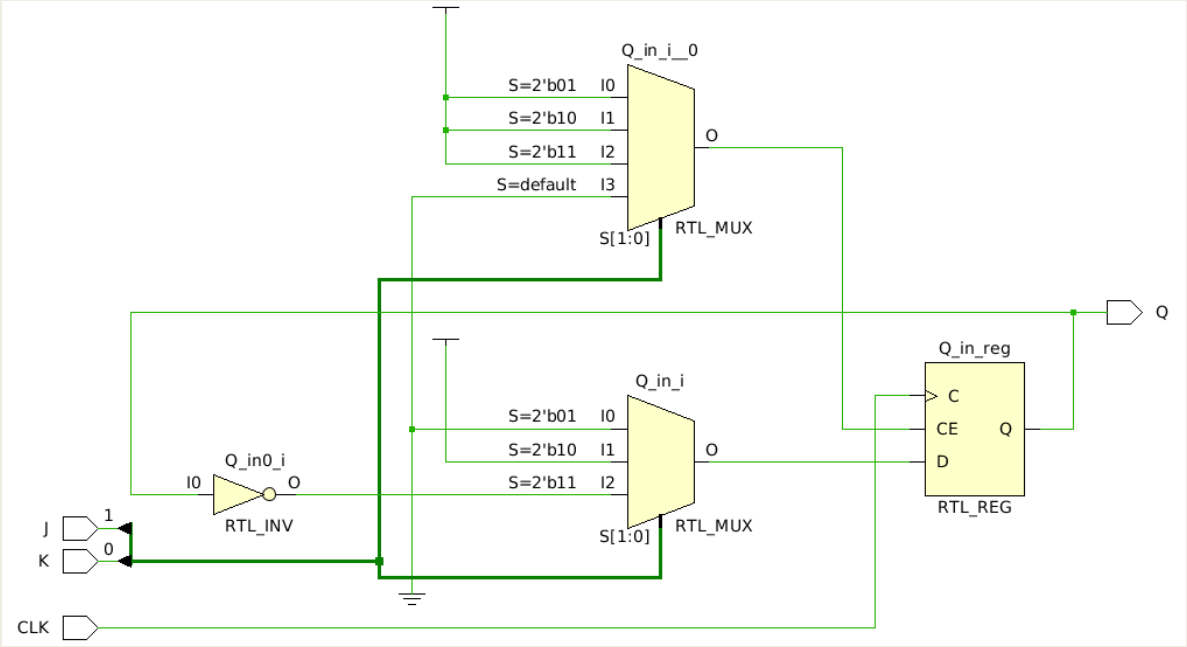
```
entity JKFF is
  port (
    CLK, J, K: in STD_LOGIC;
    Q: out STD_LOGIC);
end JKFF;
architecture JKFF_BEH of JKFF is
  signal Q_in: STD_LOGIC;
begin
  process (CLK)
    variable JK: STD_LOGIC_VECTOR (1 downto 0);
    begin
      if (CLK = '1' and CLK'event) then
        JK := J & K;                          -- concatenation
        case JK is
          when  "01" => Q_in <= '0';          -- reset
          when  "10" => Q_in <= '1';          -- set
          when  "11" => Q_in <= not Q_in;     -- toggle
          when  others => null;               -- hold
        end case;
      end if;
  end process;
  Q <= Q_in;
end JKFF_BEH;
```

Υλοποιείται
ο πίνακας λειτουργίας

JK Flip-Flop στη VHDL

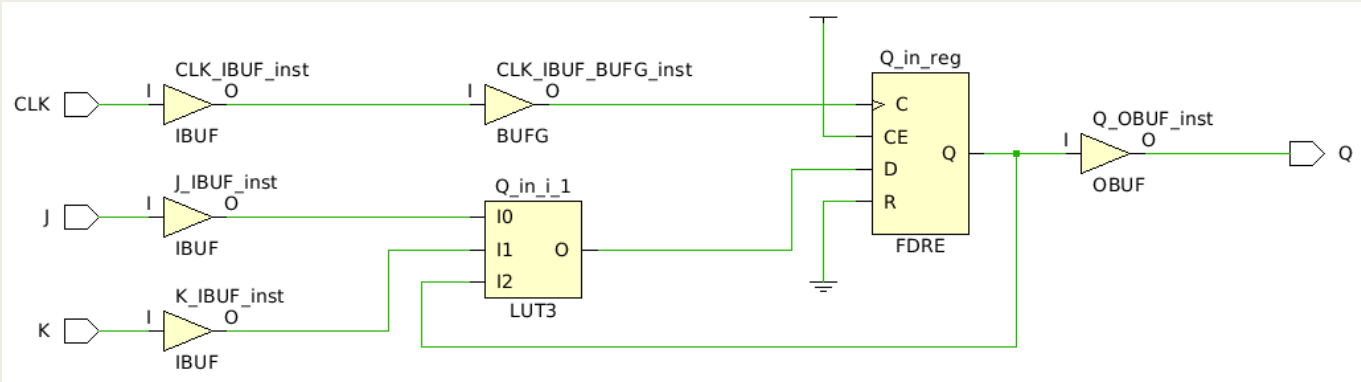
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



JK	Λειτουργία
00	HOLD
01	RESET
10	SET
11	TOGGLE

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Περιγραφή δομής (structural) στη VHDL
Η εντολή FOR GENERATE

```
for I in 0 to 3 generate
    concurrent_statement_with_A(I);
end generate;
```

ισοδυναμεί με :

```
concurrent_statement_with_A(0);
concurrent_statement_with_A(1);
concurrent_statement_with_A(2);
concurrent_statement_with_A(3);
```

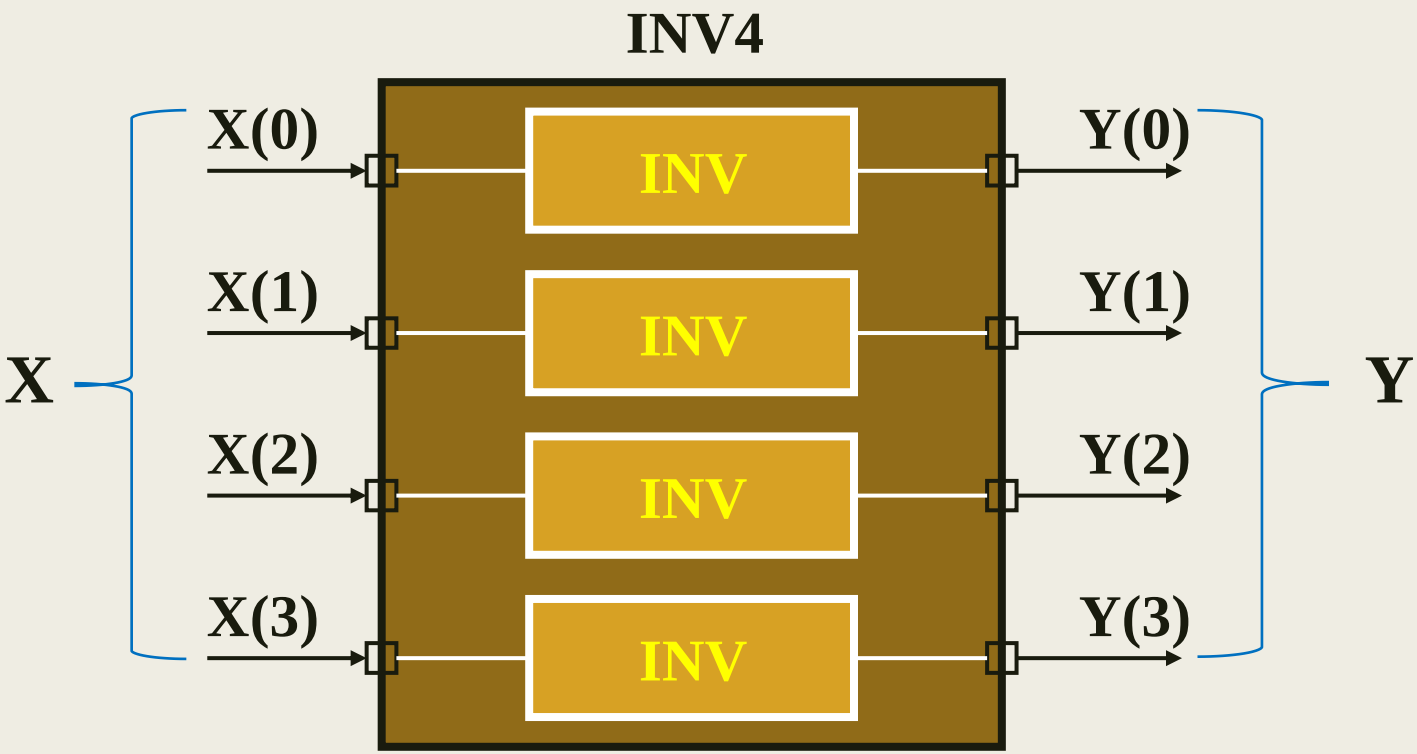
Η εντολή **FOR GENERATE**, εκτελεί την ταυτόχρονη εντολή 4 φορές και κάθε φορά η τιμή του **I** μεταβάλλεται από 0 μέχρι 3

Περιγραφή δομής (structural) στη VHDL
Η εντολή FOR GENERATE

- Χρησιμοποιείται για την επανάληψη **ταυτόχρονων** εντολών
  - *ταυτόχρονες εντολές ανάθεσης σήματος*
  - *ταυτόχρονες εντολές στοιχείων*
- Ο αριθμός των επαναλήψεων **I** είναι μία **μεταβλητή τύπου natural**, η οποία δεν χρειάζεται να δηλωθεί
- Χρησιμοποιείται κυρίως για την περιγραφή **επαναληπτικών διατάξεων λογικής**.
  - ***Προσοχή:*** *Οι **οριζόντιες έξοδοι** μίας επαναλαμβανόμενης κυψελίδας που είναι είσοδοι της αμέσως επόμενης κυψελίδας περιγράφονται σαν **εσωτερικά σήματα** και απαιτούν:*
    - **αρχικοποίηση** πριν την εντολή FOR GENERATE
    - **μετατροπή σε σήματα εξόδου** μετά την εντολή FOR GENERATE
  - *Πρέπει να περιγράφεται επακριβώς η **επαναλαμβανόμενη κυψελίδα** καθώς και ο **αλγόριθμος επανάληψης***

Πολλαπλά αντίγραφα στοιχείων στη VHDL
Περιγραφή δομής

- Θα εξετάσουμε την περιγραφή ενός **4-ψήφιου αντιστροφέα**, που αποτελείται από 4 στοιχεία (component) INV, που είναι ήδη ορισμένα ως entity INV



Πολλαπλά αντίγραφα στοιχείων στη VHDL

Περιγραφή δομής

```
entity INV4 is
  port (
    X: in STD_LOGIC_VECTOR (0 to 3);
    Y: out STD_LOGIC_VECTOR (0 to 3));
end INV4;
architecture INV4_STR1 of INV4 is
  component INV
    port (O : out STD_LOGIC; I : in STD_LOGIC);
  end component;
begin
  U0: INV port map (Y(0), X(0));
  U1: INV port map (Y(1), X(1));
  U2: INV port map (Y(2), X(2));
  U3: INV port map (Y(3), X(3));
end INV4_STR1;
```

```
G1: for I in 0 to 3 generate
    U1: INV port map (Y(I), X(I));
  end generate G1;
```

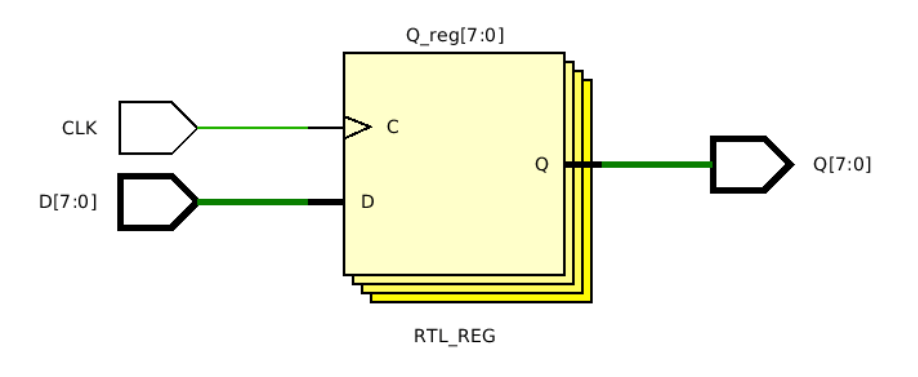
Καταχωρητής των 8 bit στη VHDL
Περιγραφή δομής

```
entity REG4_8 is
    port (
        CLK: in STD_LOGIC;
        D: in  STD_LOGIC_VECTOR (7 downto 0);
        Q: out STD_LOGIC_VECTOR (7 downto 0));
end REG4_8;
architecture REG4_8_STR of REG4_8 is
  component FD
    port (CLK, D : in STD_LOGIC;
          Q : out STD_LOGIC);
  end component;
begin
  G1: for I in 7 downto 0 generate
    U1: FD port map (Q(I), CLK, D(I));
  end generate G1;
end REG4_8_STR;
```

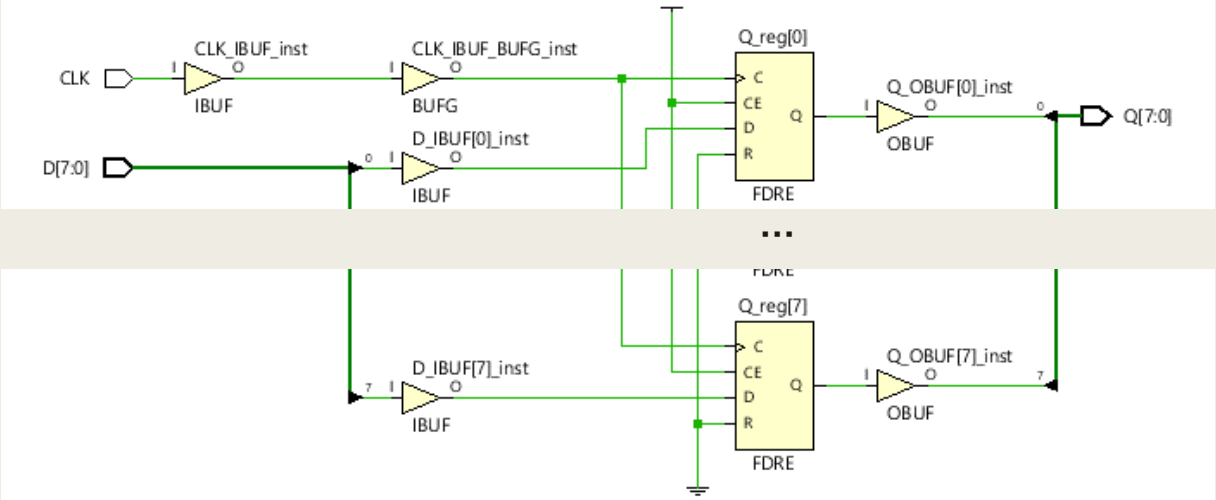
Διαθέσιμο στοιχείο

Καταχωρητής των 8 bit στη VHDL
Περιγραφή δομής

- Σχηματικό διάγραμμα RTL

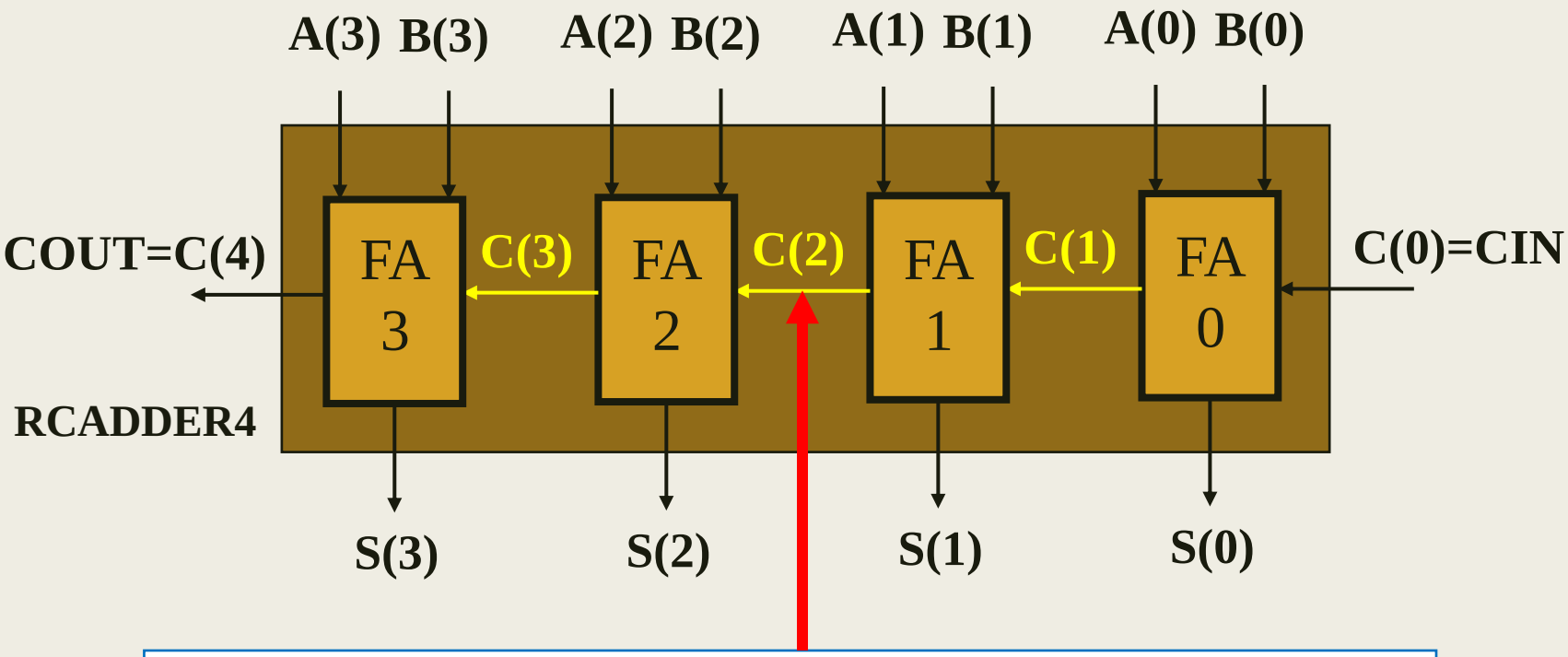


- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Αθροιστής ριπής κρατούμενου των 4 bit στη VHDL

Περιγραφή δομής



Τα οριζόντια σήματα C(i) (i=0, ..,4) ορίζονται σαν εσωτερικά σήματα

Επαναλαμβανόμενη χρήση του στοιχείου FULL_ADDER σε μία περιγραφή δομής

Αθροιστής ριπής κρατούμενου των 4 bit στη VHDL

Περιγραφή δομής

1st Entity - Component

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

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

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

Προσοχή στην υποχρεωτική νέα δήλωση του library που διαχωρίζει τα entities

2nd Entity

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

entity RCADDER4 is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
           B : in  STD_LOGIC_VECTOR (3 downto 0);
           Cin : in  STD_LOGIC;
           S : out  STD_LOGIC_VECTOR (3 downto 0);
           Cout : out  STD_LOGIC);
end RCADDER4;

architecture Behavioral of RCADDER4 is

  signal C: STD_LOGIC_VECTOR (4 downto 0); -- οριζόντια σήματα
  component FULL_ADDER
    port (
    A, B, CIN: in STD_LOGIC;
    SUM, CARRY: out STD_LOGIC);
  end component;
begin
    C(0) <= CIN;     -- αρχικοποίηση
    G1: for I in 0 to 3 generate
        U1: FULL_ADDER port map (A(I),B(I),C(I),S(I),C(I+1));
        end generate G1;
    COUT <= C(4);    -- ανάθεση σε σήμα εξόδου

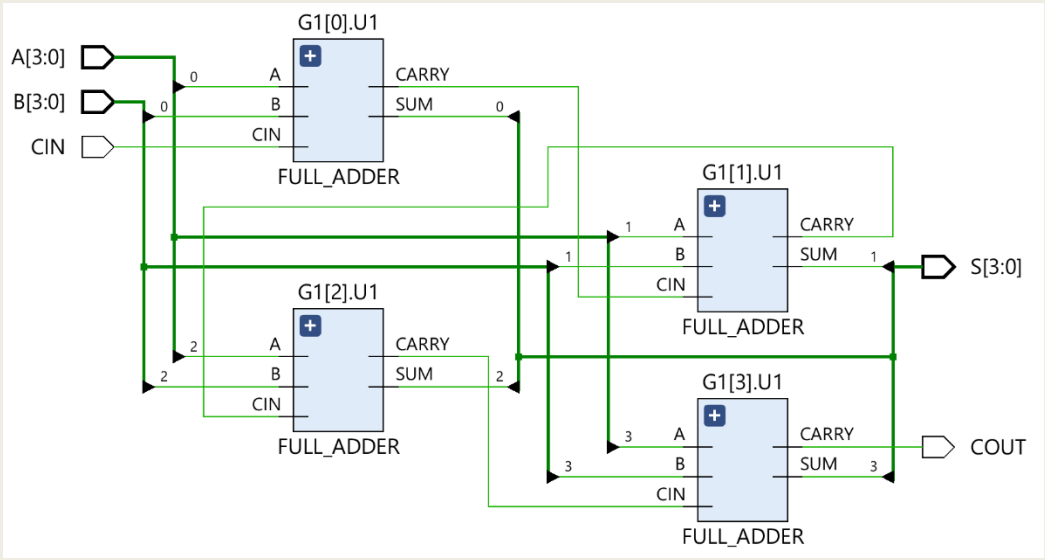
end Behavioral;
```

Περισσότερα από ένα entities ιεραρχικά δομημένα στο ίδιο vhd file

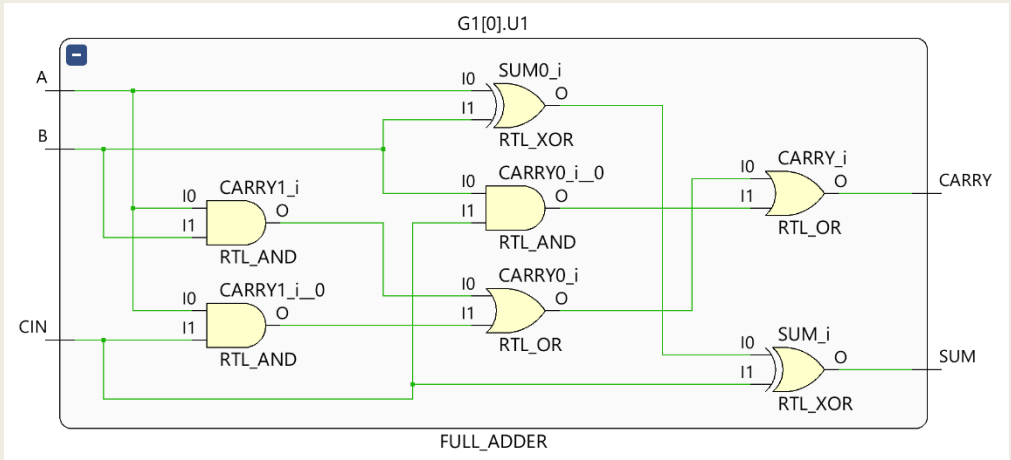
Αθροιστής ριπής κρατούμενου των 4 bit στη VHDL

Περιγραφή δομής

- Ιεραρχικό σχηματικό διάγραμμα RTL (Α' επίπεδο – RCADDER4)



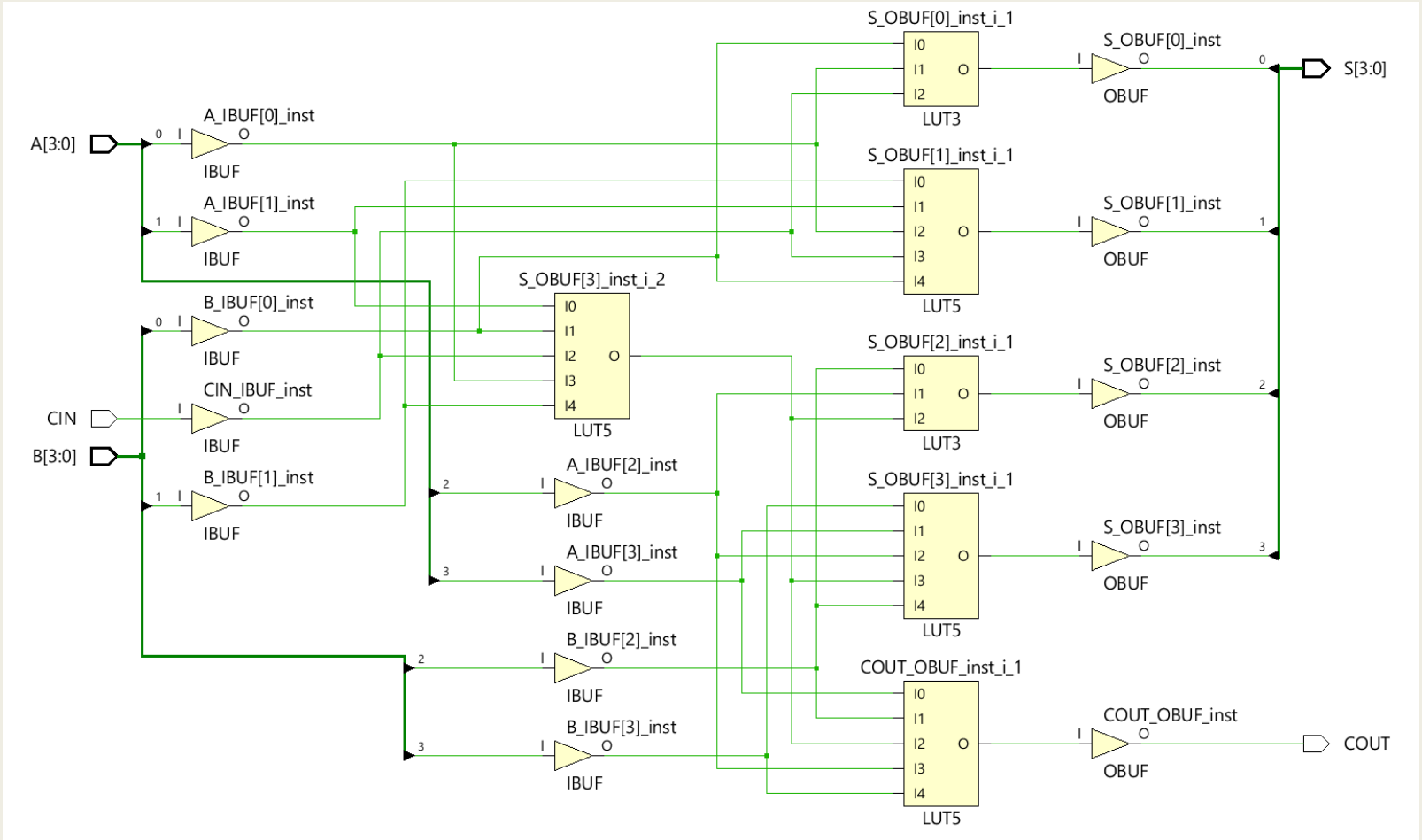
- Ιεραρχικό σχηματικό διάγραμμα RTL (Β' επίπεδο – FULL_ADDER)



Αθροιστής ριπής κρατούμενου των 4 bit στη VHDL

Περιγραφή δομής

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Υλοποίηση με **6 LUTs** σε **2 επίπεδα LUT**

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Η εντολή FOR LOOP

```
for I in 0 to 3 loop
    sequential_statement_with_A(I);
end loop
```

ισοδυναμεί με :

```
sequential_statement_with_A(0);
sequential_statement_with_A(1);
sequential_statement_with_A(2);
sequential_statement_with_A(3);
```

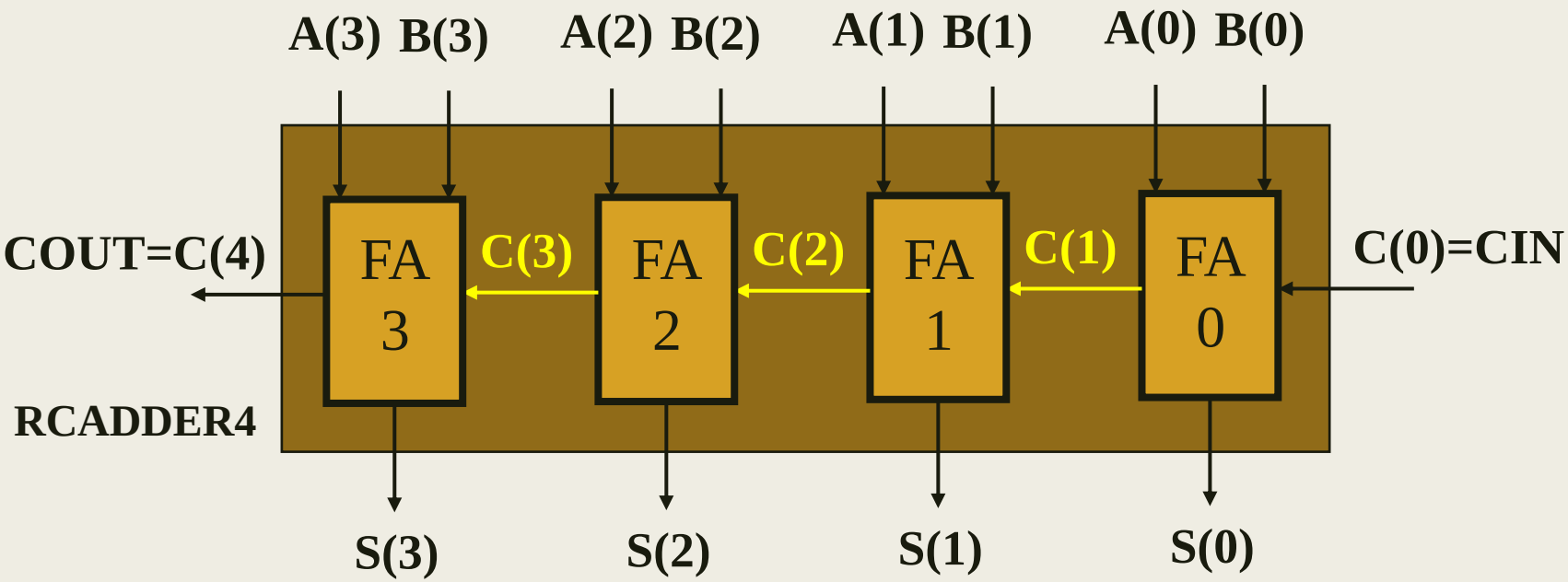
Η εντολή FOR LOOP, εκτελεί την ακολουθιακή εντολή 4 φορές και κάθε φορά η τιμή του I μεταβάλλεται από 0 μέχρι 3

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Η εντολή FOR LOOP

- Χρησιμοποιείται για την **επανάληψη ακολουθιακών εντολών** μέσα σε μία **διεργασία** (process)
- Ο αριθμός των επαναλήψεων **I** είναι μία μεταβλητή **τύπου natural**, η οποία δεν χρειάζεται να δηλωθεί
- Χρησιμοποιείται κυρίως για την περιγραφή **επαναληπτικών διατάξεων λογικής**
  - ***Προσοχή**: Οι οριζόντιες έξοδοι μίας επαναλαμβανόμενης κυψελίδας που είναι είσοδοι της αμέσως επόμενης κυψελίδας περιγράφονται σαν **μεταβλητές** και απαιτούν:*
    - **αρχικοποίηση** πριν την εντολή FOR LOOP
    - **μετατροπή σε σήματα εξόδου** μετά την εντολή FOR LOOP
  - ***Προσοχή**: Οι κατακόρυφες έξοδοι μίας επαναλαμβανόμενης κυψελίδας περιγράφονται σαν **μεταβλητές** και απαιτούν:*
    - **μετατροπή σε σήματα εξόδου** μετά την εντολή FOR LOOP
  - *Πρέπει να περιγράφεται επακριβώς η **επαναλαμβανόμενη κυψελίδα** καθώς και ο **αλγόριθμος επανάληψης***

Αθροιστής ριπής κρατούμενου των 4 bit στη VHDL

Περιγραφή συμπεριφοράς




```
S(I) <= A(I) xor B(I) xor C(I);
C(I+1) = (A(I) and B(I)) or (A(I) and C(I)) or (B(I) and C(I));
```

Αθροιστής ριπής κρατούμενου των 4 bit στη VHDL

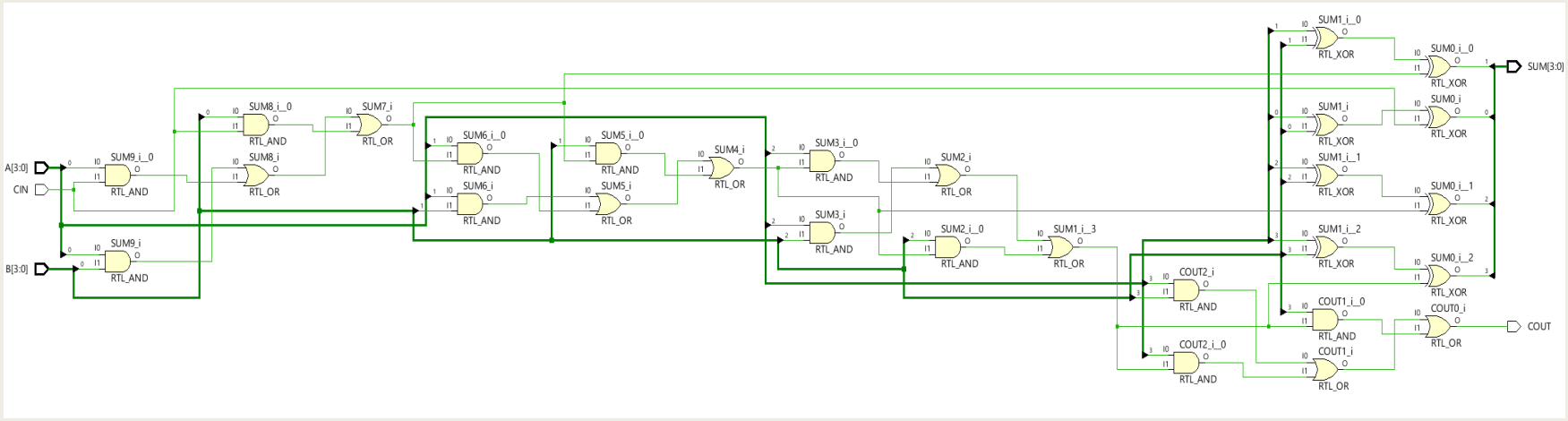
Περιγραφή συμπεριφοράς

```
entity RCADDER4 is
  port (
    A: in STD_LOGIC_VECTOR (3 downto 0);
    B: in STD_LOGIC_VECTOR (3 downto 0);
    SUM: out STD_LOGIC_VECTOR (3 downto 0);
    CIN: in STD_LOGIC;
    COUT: out STD_LOGIC);
end RCADDER4;
architecture BEHAVIORAL of RCADDER4 is
begin
  process (A, B, CIN)
  variable C: STD_LOGIC_VECTOR (4 downto 0); -- επαναλαμβανόμενοι
  variable S: STD_LOGIC_VECTOR (3 downto 0); -- έξοδοι
    begin
    C(0) := CIN;    -- αρχικοποίηση
    for I in 0 to 3 loop
      S(I) := A(I) xor B(I) xor C(I);
      C(I+1)  := (A(I) and B(I)) or (A(I) and C(I)) or (B(I) and C(I));
    end loop;
    SUM <= S;       -- μετατροπή σε σήματα εξόδου
    COUT <= C(4);
  end process;
end BEHAVIORAL;
```

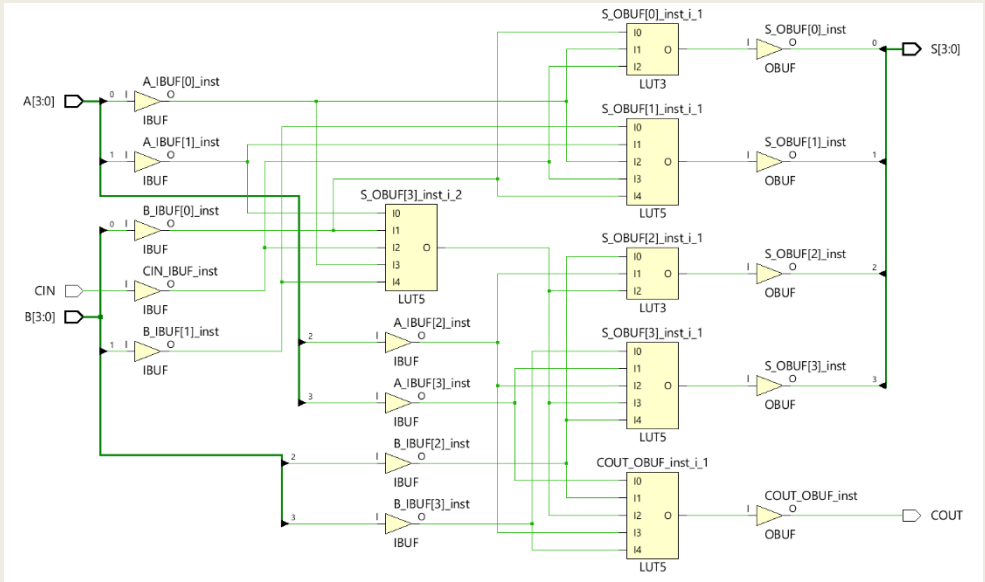
Αθροιστής ριπής κρατούμενου των 4 bit στη VHDL

Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Ίδια υλοποίηση με **6 LUTs** σε **2 επίπεδα LUT**

Χρήσιμοι τύποι στη VHDL

- Type **Integer**
  - *προσδιορίζεται στο πακέτο **STANDARD** που δεν δηλώνεται*
  - *προσημασμένος ακέραιος σε συμπλήρωμα ως προς 2*
  - *το εύρος εξαρτάται από την υλοποίηση, αλλά συνήθως είναι από -2^{31} μέχρι $2^{31}-1$ (32 ψηφία)*
  - *άλλο μικρότερο εύρος προσδιορίζεται με τη χρήση του range,*
    - Παράδειγμα: integer range 0 to 7 (3 ψηφία)
- Subtypes **Natural** και **Positive** του type **Integer**
  - *προσδιορίζονται στο πακέτο **STANDARD** που δεν δηλώνεται*
  - *το εύρος είναι από 0 (ή 1, αντίστοιχα) μέχρι το μέγιστο θετικό ακέραιο*
- Types **Unsigned** και **Signed**
  - *προσδιορίζεται στο πακέτο **NUMERIC_STD** της βιβλιοθήκης IEEE*
    - IEEE στάνταρ 1076.3 του 1997 που υποστηρίζεται από το **VIVADO**
    - για να χρησιμοποιηθεί δηλώνουμε:

```
library IEEE; -- εάν δεν έχει ήδη δηλωθεί
use IEEE.numeric_std.all;
```

Μετατροπές από τον ένα τύπο στον άλλο

Χρήση του τύπου **unsigned**

- LV: std_logic_vector to U: unsigned
  - `U <= unsigned(LV)`
- U: unsigned to N: natural
  - `N <= to_integer(U)`
- LV: std_logic_vector to N: natural
  - `N <= to_integer(unsigned(LV))`
- N: natural(n-bit) to U: unsigned
  - `U <= to_unsigned(N,#bits)`
- U: unsigned to LV: std_logic_vector
  - `LV <= std_logic_vector(U)`
- N: natural(n-bit) to LV: std_logic_vector
  - `LV <= std_logic_vector(to_unsigned(N,#bits))`

Μετατροπές από τον ένα τύπο στον άλλο

Χρήση του τύπου **signed**

- LV: std_logic_vector to S: signed
  - `S <= signed(LV)`
- S: signed to I: integer
  - `I <= to_integer(S)`
- LV: std_logic_vector to I: integer
  - `I <= to_integer(signed(LV))`
- I: integer(n-bit) to U: unsigned
  - `S <= to_signed(I,#bits)`
- S: signed to LV: std_logic_vector
  - `LV <= std_logic_vector(S)`
- I: integer(n-bit) to LV: std_logic_vector
  - `LV <= std_logic_vector(to_signed(I,#bits))`

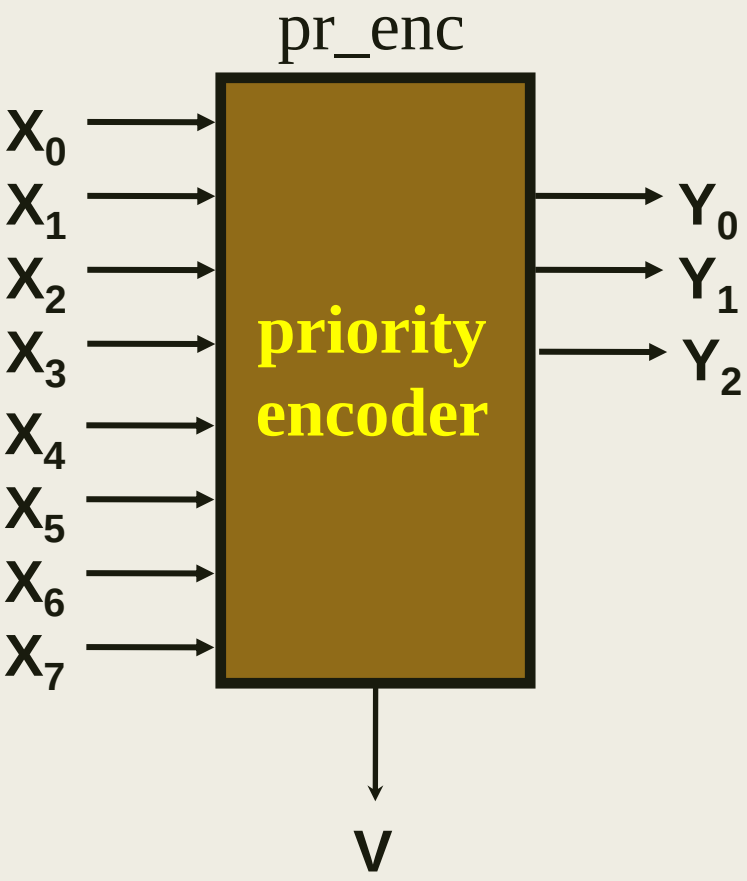
Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Η εντολή WHILE LOOP

```
while boolean_expresion loop
    sequential_statement;
    sequential_statement_affecting_expression;
end loop
```

Η εντολή **WHILE LOOP**, εκτελεί τις ακολουθιακές εντολές μέσα στο loop, όσο ικανοποιείται η συνθήκη (boolean_expression = true). Ο έλεγχος της συνθήκης γίνεται πριν την εκτέλεση των ακολουθιακών εντολών μέσα στο loop. Μία από τις εντολές μέσα στο loop επηρεάζει τη συνθήκη. Χρησιμοποιείται, αντί της εντολής FOR LOOP, στην περίπτωση που θέλουμε να σταματήσει η εκτέλεση του loop όταν ικανοποιηθεί η συνθήκη.

Κωδικοποιητής προτεραιότητας των 8 bit στη VHDL

Περιγραφή συμπεριφοράς



X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	Y_2	Y_1	Y_0	V
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	x	0	0	1	1
0	0	0	0	0	1	x	x	0	1	0	1
0	0	0	0	1	x	x	x	0	1	1	1
0	0	0	1	x	x	x	x	1	0	0	1
0	0	1	x	x	x	x	x	1	0	1	1
0	1	x	x	x	x	x	x	1	1	0	1
1	x	x	x	x	x	x	x	1	1	1	1

Το ψηφίο εγκυρότητας V διαχωρίζει την είσοδο 00000000 από την είσοδο 00000001

Κωδικοποιητής προτεραιότητας των 8 bit στη VHDL

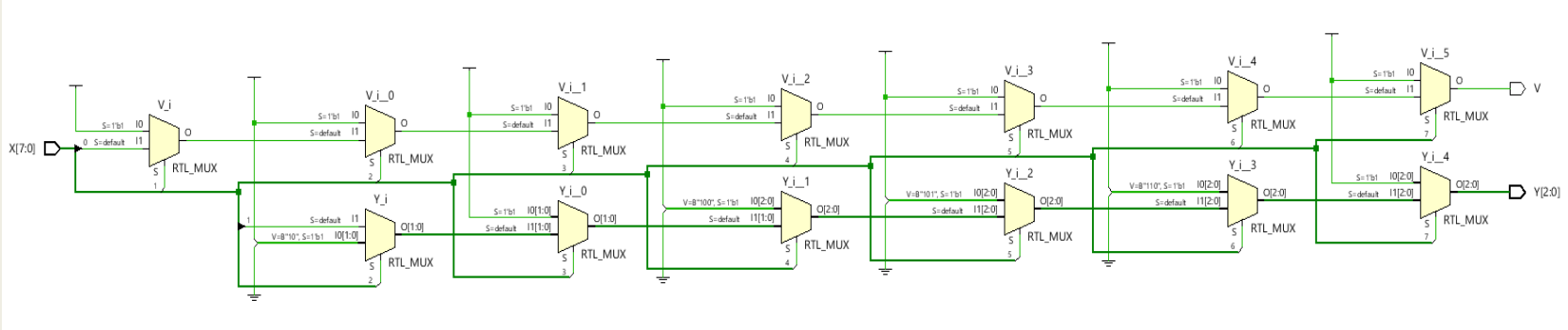
Περιγραφή συμπεριφοράς – Λύση 1 με την εντολή IF

```
entity pr_enc1 is
  port (
    X: in STD_LOGIC_VECTOR (7 downto 0);
    Y: out STD_LOGIC_VECTOR (2 downto 0);
    V: out STD_LOGIC);
end pr_enc1;
architecture BEHAVIORAL of pr_enc1 is
begin
  process (X)
    begin
      Y <= "000"; V <='0';
      if    (X(7) = '1') then Y <= "111"; V <= '1';
      elsif (X(6) = '1') then Y <= "110"; V <= '1';
      elsif (X(5) = '1') then Y <= "101"; V <= '1';
      elsif (X(4) = '1') then Y <= "100"; V <= '1';
      elsif (X(3) = '1') then Y <= "011"; V <= '1';
      elsif (X(2) = '1') then Y <= "010"; V <= '1';
      elsif (X(1) = '1') then Y <= "001"; V <= '1';
      elsif (X(0) = '1') then Y <= "000"; V <= '1';
      else null; end if;
    end process;
end BEHAVIORAL;
```

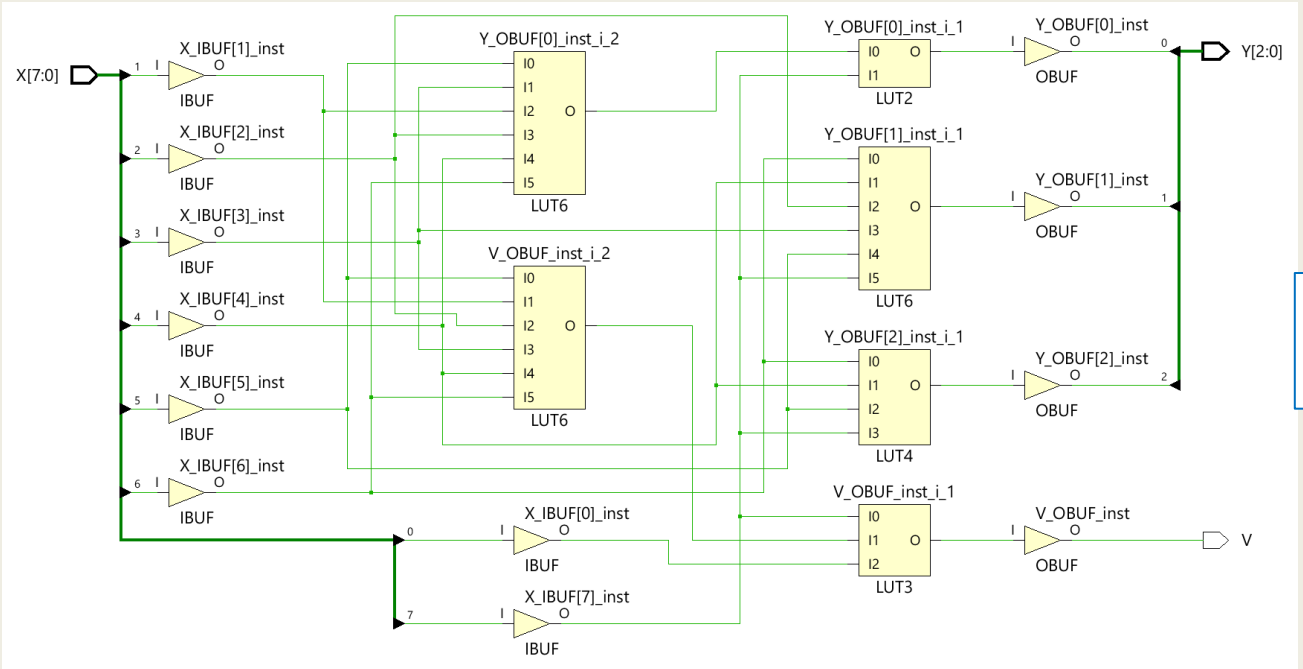
Κωδικοποιητής προτεραιότητας των 8 bit στη VHDL

Περιγραφή συμπεριφοράς – Λύση 1 με την εντολή IF

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Υλοποίηση με **6 LUTs** σε **2 επίπεδα LUT**

Κωδικοποιητής προτεραιότητας των 8 bit στη VHDL

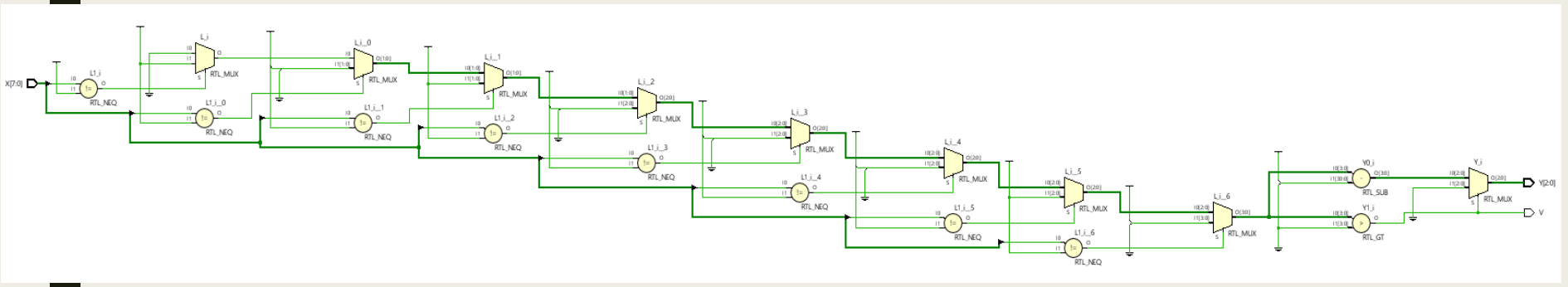
Περιγραφή συμπεριφοράς – Λύση 2 με την εντολή WHILE LOOP

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL; -- ενεργοποίηση του numeric_std
entity pr_enc2 is
  port (
    X: in STD_LOGIC_VECTOR (7 downto 0);
    Y: out STD_LOGIC_VECTOR (2 downto 0);
    V: out STD_LOGIC);
end pr_enc2;
architecture BEHAVIORAL of          is
begin
  process (X)
  variable I : integer range 7 downto 0;
  variable L : integer range 8 downto 0;
  begin
    L := 8;
    while (L > 0) and (X(L-1) /= '1') loop L := L - 1; end loop;
    if (L > 0) then
      I := L - 1;
      Y <= std_logic_vector(to_unsigned(I, 3)); V <= '1';
    else
      Y <= "000"; V <= '0';
    end if;
  end process;
end BEHAVIORAL;
```

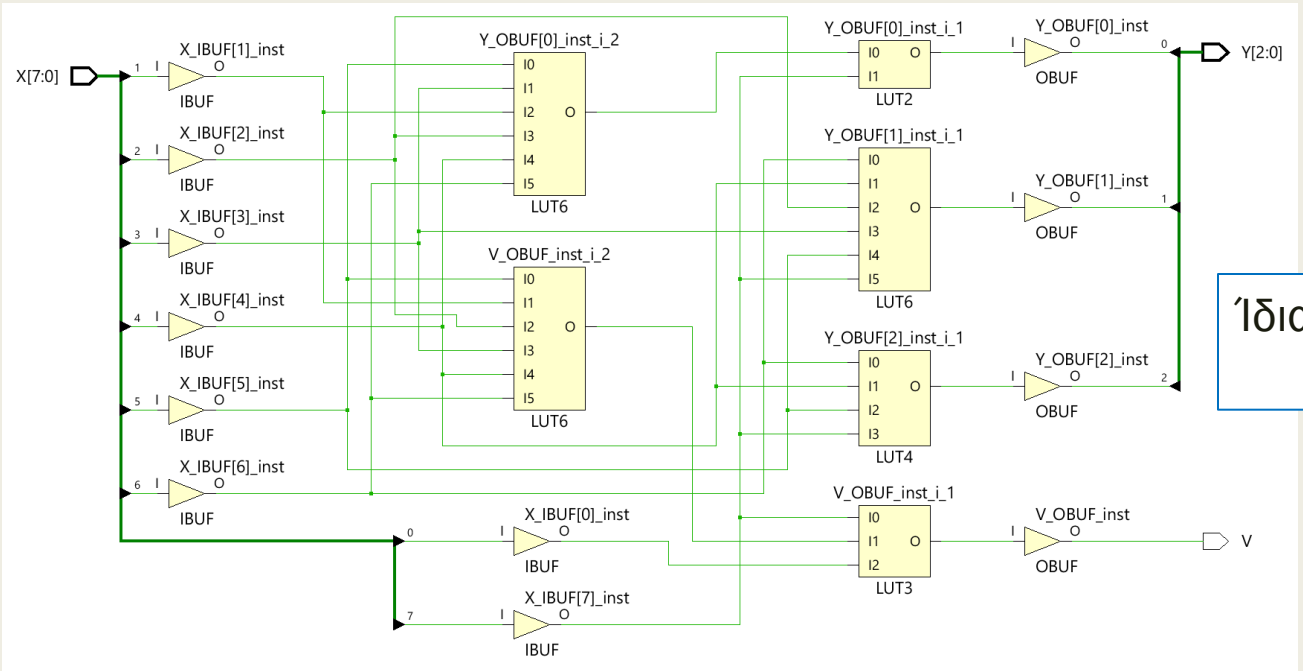
Κωδικοποιητής προτεραιότητας των 8 bit στη VHDL

Περιγραφή συμπεριφοράς – Λύση 2 με την εντολή WHILE LOOP

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Ίδια υλοποίηση με **6 LUTs** σε **2 επίπεδα LUT**

Διευκολύνσεις στη σύνθεση

- Πολλά εργαλεία σύνθεσης υποστηρίζουν **βιβλιοθήκες έτοιμων στοιχείων** για **βέλτιστη υλοποίηση**
- οι βιβλιοθήκες έτοιμων στοιχείων συμπεριλαμβάνουν:
  - *λειτουργικές μονάδες (π.χ. αθροιστές, πολλαπλασιαστές)*
  - *μνήμες (π.χ. ROM, RAM, FIFO)*
  - *πιο σύνθετα υποσυστήματα*
    - Μελετήστε τα application notes της XILINX
- Ο τρόπος χρήσης των έτοιμων στοιχείων μέσα στο VHDL πρόγραμμα είναι συγκεκριμένος και προσδιορίζεται από τον κατασκευαστή των έτοιμων στοιχείων
  - *για αθροιστές:* `S <= A + B`
  - *για πολλαπλασιαστές:* `M <= A * B`
  - *χρησιμοποιούνται είτε σαν **ταυτόχρονες** είτε σαν **ακολουθιακές** εντολές*

Αθροιστές και αφαιρέτες στη VHDL

- Βασίζεται στο πρότυπο πακέτο **numeric_std** της βιβλιοθήκης **IEEE**
  - *Υποστηρίζεται από το **VIVADO** και τη **βιβλιοθήκη έτοιμων στοιχείων** της **XILINX***
- Συνθέτουμε αθροιστές χρησιμοποιώντας τον τελεστή "**+**"
- Συνθέτουμε αφαιρέτες χρησιμοποιώντας τον τελεστή "**-**"
- Όταν οι τελεστέοι είναι τύπου **unsigned**, τότε
  - *ο δεύτερος τελεστέος μπορεί να είναι και τύπου **natural***
  - *εάν δεν έχουν το ίδιο μέγεθος γίνεται **επέκταση μηδενός***
  - *εάν ο τελεστέος **LV** είναι τύπου **std_logic_vector**, πρέπει πρώτα να μετατραπεί σε τελεστέο **U** τύπου **unsigned** χρησιμοποιώντας τη συνάρτηση:*

    ```
    U = unsigned(LV)
    ```

  - *το αποτέλεσμα **U** είναι τύπου **unsigned** και μετατρέπεται σε αποτέλεσμα **LV** τύπου **std_logic_vector** χρησιμοποιώντας τη συνάρτηση:*

    ```
    LV = std_logic_vector(U)
    ```

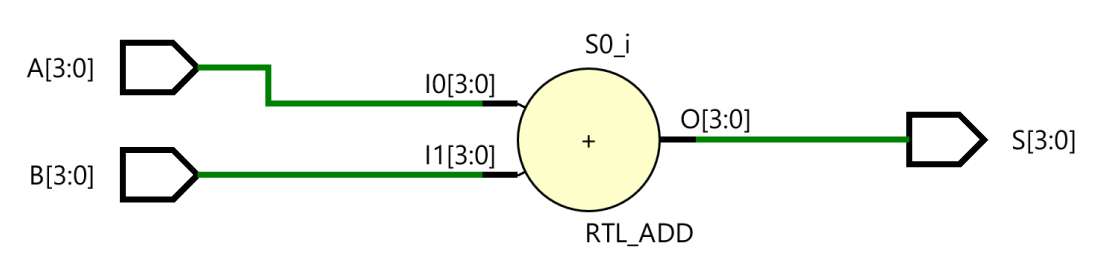
Χρήση του τύπου **unsigned**

Μη προσημασμένος αθροιστής των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

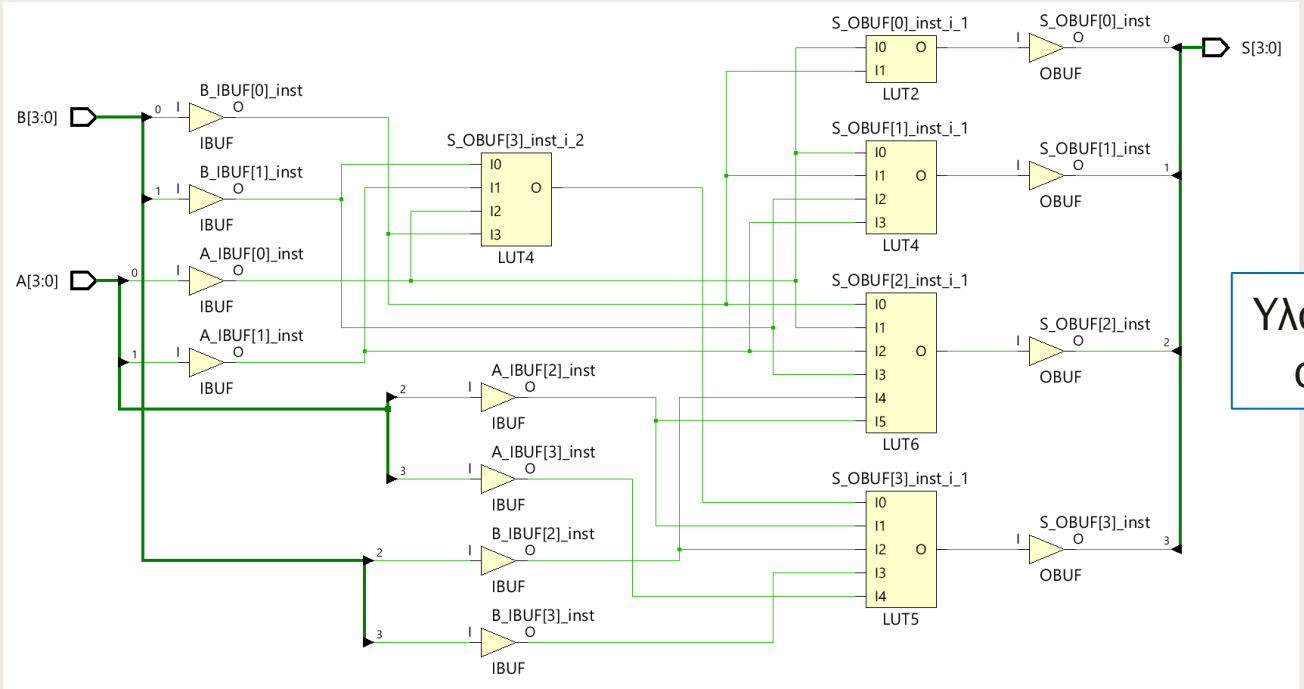
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity ADD4 is
  port (
    A : in  STD_LOGIC_VECTOR (3 downto 0);
    B : in  STD_LOGIC_VECTOR (3 downto 0);
    S : out STD_LOGIC_VECTOR (3 downto 0));
end ADD4;
architecture BEHAVIORAL of ADD4 is
begin
  ADD4: process (A, B)
  variable A_u, B_u, S_u: UNSIGNED (3 downto 0);
  begin
    A_u := unsigned(A);              -- numeric_std
    B_u := unsigned(B);              -- numeric_std
    S_u := A_u + B_u;                -- numeric_std
    S <= std_logic_vector(S_u);      -- numeric_std
  end process;
end BEHAVIORAL;
```

Μη προσημασμένος αθροιστής των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Υλοποίηση με **5 LUTs** σε **2 επίπεδα LUT**

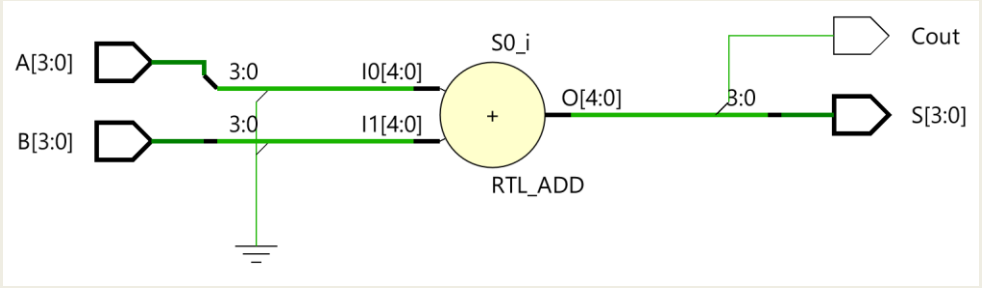
Μη προσημασμένος αθροιστής των 4 bit με Cout στη VHDL – Περιγραφή συμπεριφοράς

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity ADD4co is
  port (
    A : in  STD_LOGIC_VECTOR (3 downto 0);
    B : in  STD_LOGIC_VECTOR (3 downto 0);
    S : out STD_LOGIC_VECTOR (3 downto 0);
    Cout : out STD_LOGIC);
end ADD4co;
architecture BEHAVIORAL of ADD4co is
begin
  ADD4co: process (A, B)
  variable A_u, B_u, S_u: UNSIGNED (4 downto 0);
  begin
    A_u := unsigned('0'&A);                    -- numeric_std
    B_u := unsigned('0'&B);                    -- numeric_std
    S_u := A_u + B_u;                          -- numeric_std
    S <= std_logic_vector(S_u(3 downto 0));    -- numeric_std
    Cout <= S_u(4);
  end process;
end BEHAVIORAL;
```

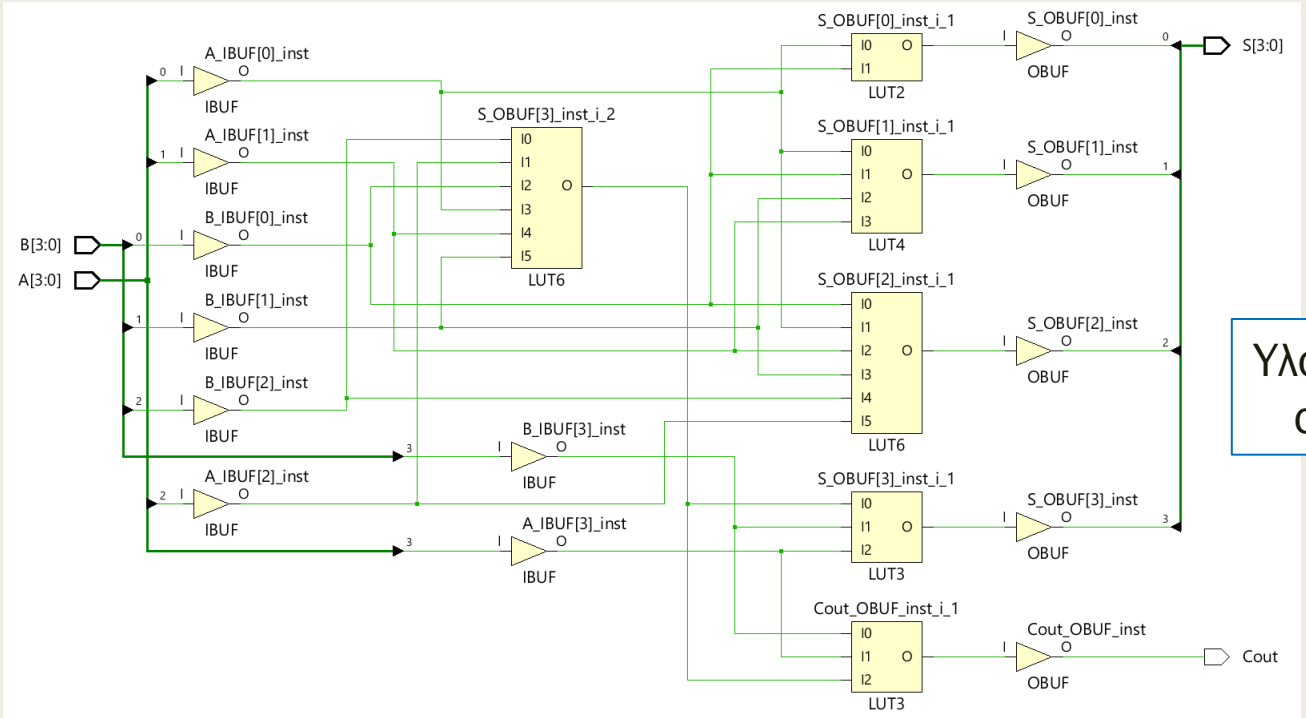
Το Cout = 1 δηλώνει υπερχείλιση στη μη προσημασμένη πρόσθεση

Μη προσημασμένος αθροιστής των 4 bit με Cout στη VHDL – Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Υλοποίηση με **6 LUTs** σε **2 επίπεδα LUT**

Αθροιστές και αφαιρέτες στη VHDL

- Βασίζονται στο πρότυπο πακέτο **numeric_std** της βιβλιοθήκης **IEEE**
  - *Υποστηρίζεται από το **VIVADO** και τη **βιβλιοθήκη έτοιμων στοιχείων** της **XILINX***
- Συνθέτουμε αθροιστές χρησιμοποιώντας τον τελεστή "**+**"
- Συνθέτουμε αφαιρέτες χρησιμοποιώντας τον τελεστή "**-**"
- Όταν οι τελεστέοι είναι τύπου **signed**, τότε
  - *ο δεύτερος τελεστέος μπορεί να είναι και τύπου **integer***
  - *εάν δεν έχουν το ίδιο μέγεθος γίνεται **επέκταση πρόσημου***
  - *εάν ο τελεστέος **LV** είναι τύπου **std_logic_vector**, πρέπει πρώτα να μετατραπεί σε τελεστέο **S** τύπου **signed** χρησιμοποιώντας τη συνάρτηση:*

    ```
    S = signed(LV)
    ```

  - *το αποτέλεσμα **S** είναι τύπου **signed** και μετατρέπεται σε αποτέλεσμα **LV** τύπου **std_logic_vector** χρησιμοποιώντας τη συνάρτηση:*

    ```
    LV = std_logic_vector(S)
    ```

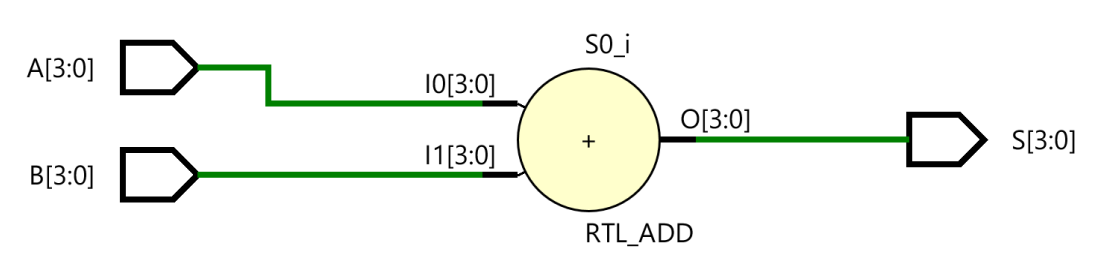
Χρήση του τύπου **signed**

Προσημασμένος αθροιστής των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity ADD4s is
  port (
    A : in  STD_LOGIC_VECTOR (3 downto 0);
    B : in  STD_LOGIC_VECTOR (3 downto 0);
    S : out STD_LOGIC_VECTOR (3 downto 0));
end ADD4s;
architecture BEHAVIORAL of ADD4s is
begin
  ADD4s: process (A, B)
  variable A_s, B_s, S_s: SIGNED (3 downto 0);
  begin
    A_s := signed(A);              -- numeric_std
    B_s := signed(B);              -- numeric_std
    S_s := A_s + B_s;              -- numeric_std
    S <= std_logic_vector(S_s);    -- numeric_std
  end process;
end BEHAVIORAL;
```

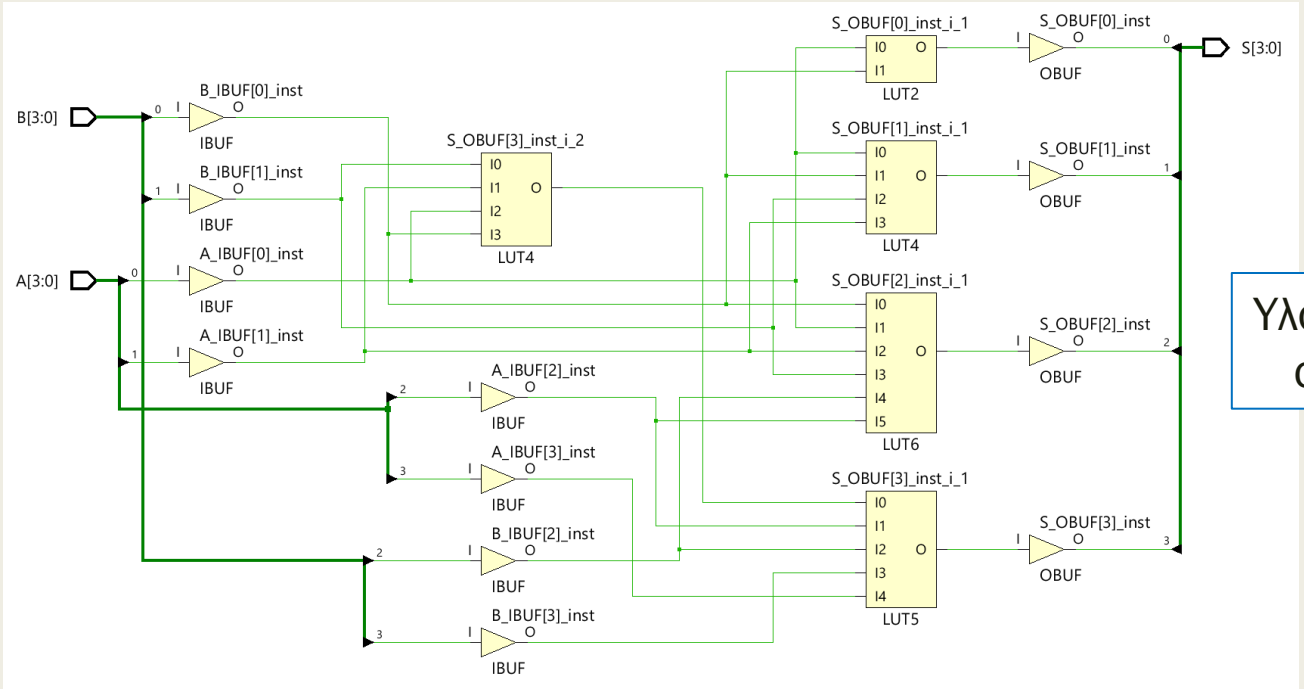
Προσημασμένος αθροιστής των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



Ο προσημασμένος αθροιστής είναι ίδιος με τον μη προσημασμένο αθροιστή

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



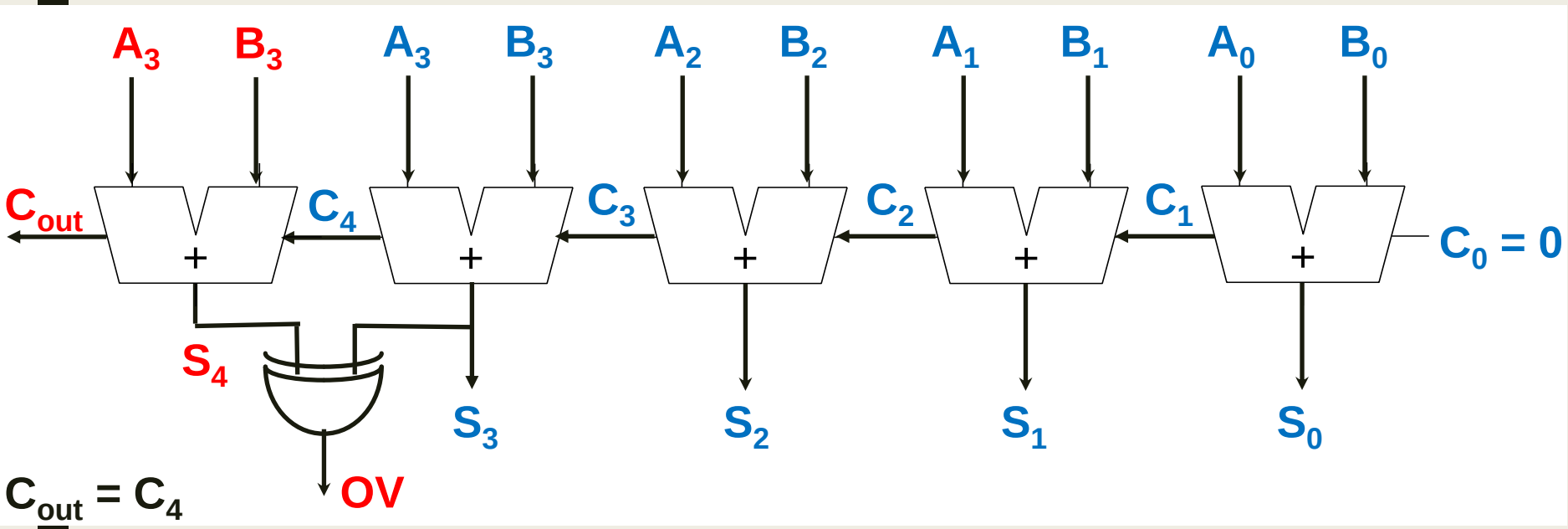
Υλοποίηση με **5 LUTs** σε **2 επίπεδα LUT**

Προσημασμένος αθροιστής των 4 bit με Cout και υπερχείλιση στη VHDL – Περιγραφή συμπεριφοράς

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity ADD4sov is
  port (
    A : in  STD_LOGIC_VECTOR (3 downto 0);
    B : in  STD_LOGIC_VECTOR (3 downto 0);
    S : out STD_LOGIC_VECTOR (3 downto 0);
    Cout : out STD_LOGIC;
    OV :   out STD_LOGIC);
end ADD4sov;
architecture BEHAVIORAL of ADD4sov is
begin
  ADD4sov: process (A, B)
  variable A_s, B_s, S_s: SIGNED (5 downto 0);
  begin
    A_s := signed('0'&A(3)&A);                  -- numeric_std
    B_s := signed('0'&B(3)&B);                  -- numeric_std
    S_s := A_s + B_s;                           -- numeric_std
    S <= std_logic_vector(S_s(3 downto 0));     -- numeric_std
    OV <= S_s(4) xor S_s(3);
    Cout <= S_s(5);
  end process;
end BEHAVIORAL;
```

Το OV = 1 δηλώνει υπερχείλιση στην προσημασμένη πρόσθεση

Υπερχείλιση και Cout στον προσημασμένο αθροιστή



C_3	A_3	B_3	C_4	S_3	C_{out}	S_4	OV
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	1	0	1	0
0	1	1	1	0	1	1	1
1	0	0	0	1	0	0	1
1	0	1	1	0	1	0	0
1	1	0	1	0	1	0	0
1	1	1	1	1	1	1	0

$A_3=B_3=1$ & $S_3=0$

$A_3=B_3=0$ & $S_3=1$

Πρόσθεση

$C_{out}=C_4\ C_3\ C_2\ C_1\ C_0=0$

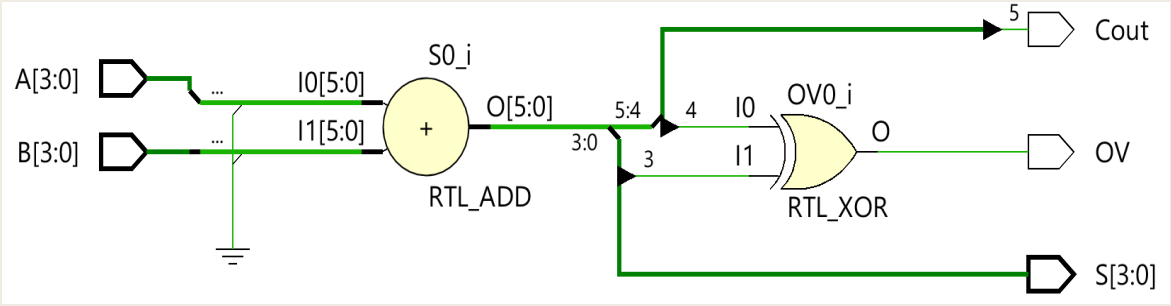
$0\quad A_3\ A_3\ A_2\ A_1\ A_0$

$+\ 0\quad B_3\ B_3\ B_2\ B_1\ B_0$

$C_{out}\ S_4\ S_3\ S_2\ S_1\ S_0$

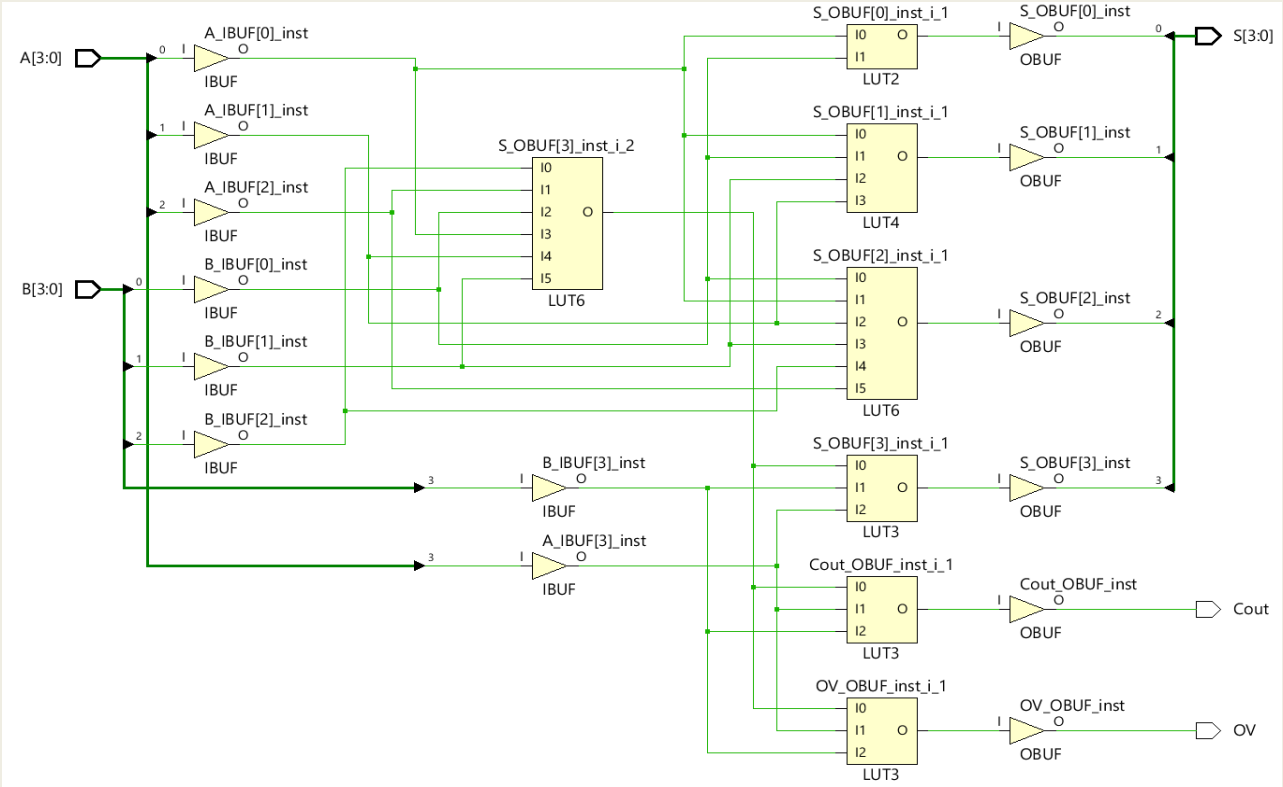
Προσημασμένος αθροιστής των 4 bit με Cout και υπερχείλιση στη VHDL – Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



Ο προσημασμένος αθροιστής διαφέρει από τον μη προσημασμένο αθροιστή στον υπολογισμό της υπερχείλισης και του Cout

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Υλοποίηση με **7 LUTs** σε **2 επίπεδα LUT**

Το ατύχημα του πύραυλου Ariane 5

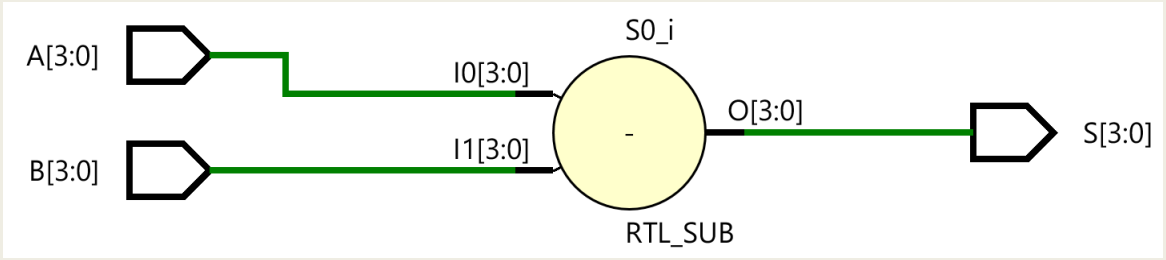
- Ο πύραυλος Ariane 5, που κόστισε 7 δισεκατομμύρια δολάρια και εκτοξεύτηκε στις 4 Ιουνίου 1996, απέκλινε από την πορεία του 40 δευτερόλεπτα μετά από την εκτόξευση, κόπηκε στα δύο και εξερράγη.
- Η αποτυχία προκλήθηκε όταν ο υπολογιστής που έλεγχε τον πύραυλο **υπερχείλισε** το εύρος (16 bit) των προσημασμένων τιμών του και κατέρρευσε.
- Ο εν λόγω κώδικας είχε ελεγχθεί διεξοδικά στον πύραυλο Ariane 4. Όμως, ο Ariane 5 διέθετε πιο γρήγορη μηχανή η οποία παρήγαγε μεγαλύτερες τιμές για τον υπολογιστή ελέγχου, προκαλώντας έτσι την υπερχείλιση

Προσημασμένος αφαιρέτης των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

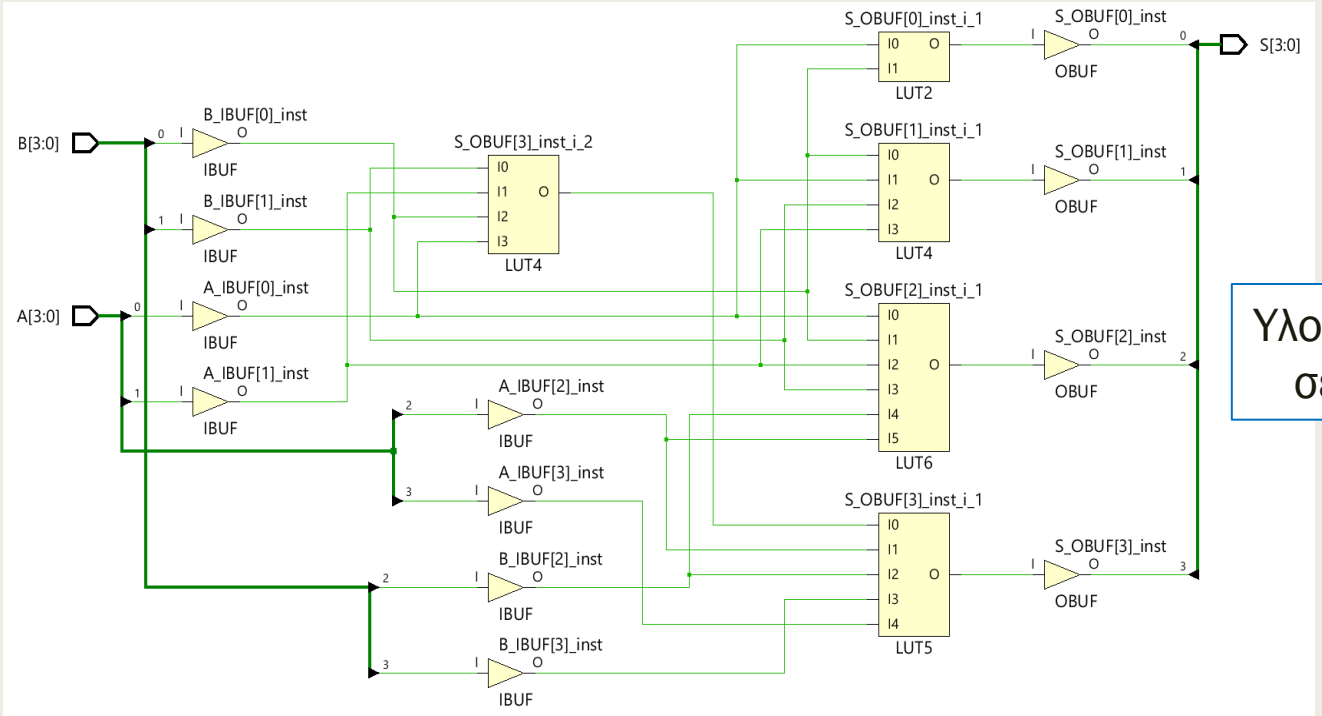
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity SUB4s is
  port (
    A : in  STD_LOGIC_VECTOR (3 downto 0);
    B : in  STD_LOGIC_VECTOR (3 downto 0);
    S : out STD_LOGIC_VECTOR (3 downto 0));
end SUB4s;
architecture BEHAVIORAL of SUB4s is
begin
  SUB4s: process (A, B)
  variable A_s, B_s, S_s: SIGNED (3 downto 0);
  begin
    A_s := signed(A);              -- numeric_std
    B_s := signed(B);              -- numeric_std
    S_s := A_s - B_s;              -- numeric_std
    S <= std_logic_vector(S_s);    -- numeric_std
  end process;
end BEHAVIORAL;
```

Προσημασμένος αφαιρέτης των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Υλοποίηση με **5 LUTs** σε **2 επίπεδα LUT**

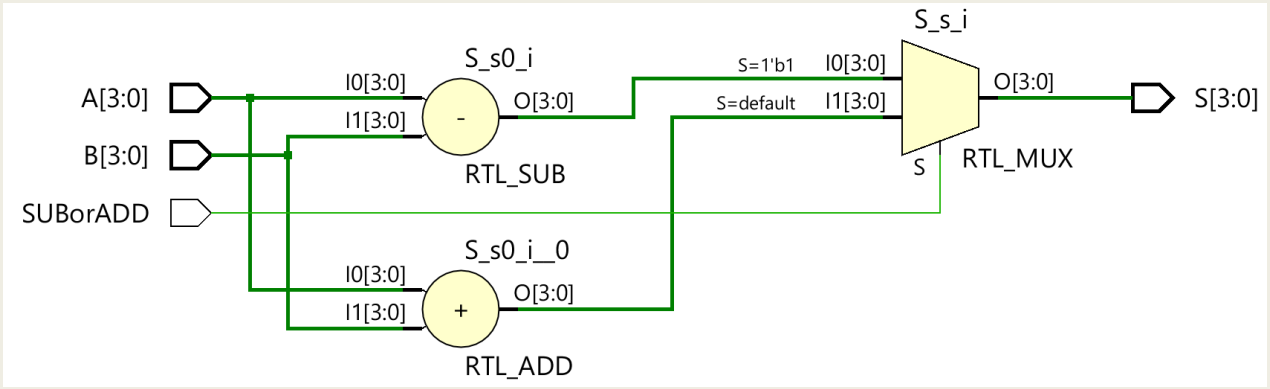
Προσημασμένος αθροιστής/αφαιρέτης των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity ADDSUB4s is
  port (
    SUBorADD : STD_LOGIC;
    A : in  STD_LOGIC_VECTOR (3 downto 0);
    B : in  STD_LOGIC_VECTOR (3 downto 0);
    S : out STD_LOGIC_VECTOR (3 downto 0));
end ADDSUB4s;
architecture BEHAVIORAL of ADDSUB4s is
begin
  ADDSUB4s: process (A, B)
  variable A_s, B_s, S_s: SIGNED (3 downto 0);
  begin
    A_s := signed(A);                              -- numeric_std
    B_s := signed(B);                              -- numeric_std
    if (SUBorADD = '1') then S_s := A_s - B_s;     -- numeric_std
                        else S_s := A_s + B_s;     -- numeric_std
    end if;
    S <= std_logic_vector(S_s);                    -- numeric_std
  end process;
end BEHAVIORAL;
```

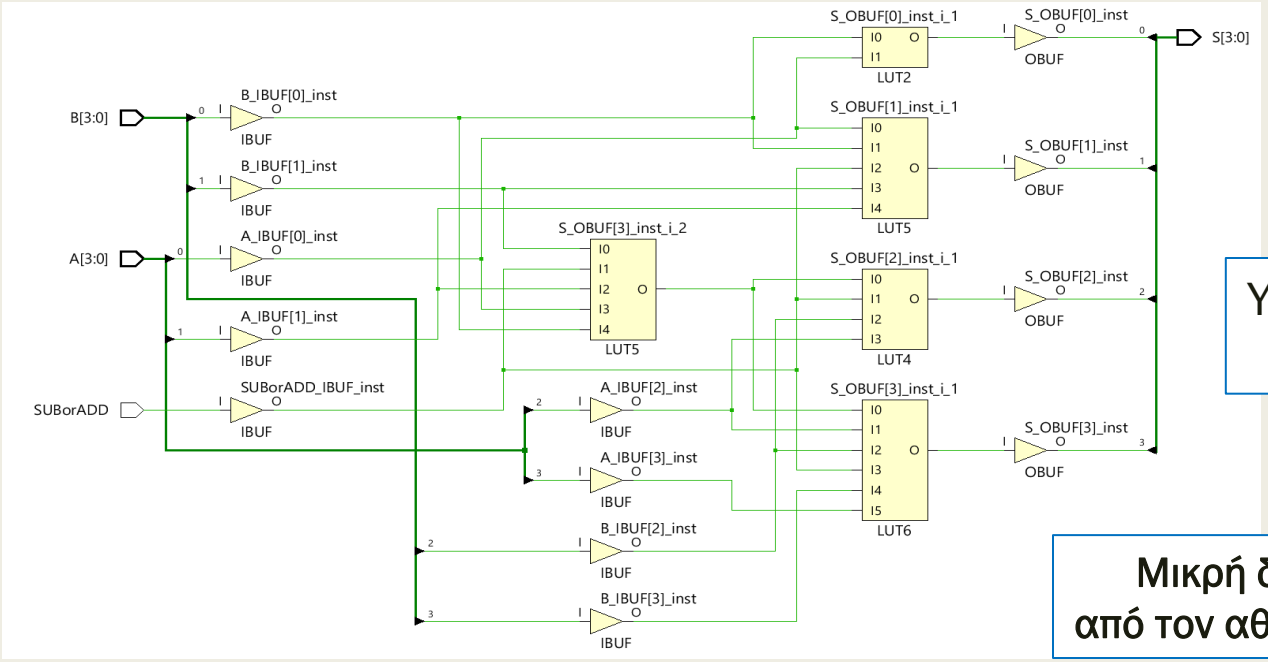
Το σήμα ελέγχου SUBorADD επιλέγει την πράξη που θα εκτελέσει ο αθροιστής/αφαιρέτης (1 = αφαίρεση, 0 = πρόσθεση)

Προσημασμένος αθροιστής/αφαιρέτης των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Υλοποίηση με **5 LUTs** σε **2 επίπεδα LUT**

Μικρή διαφοροποίηση από τον αθροιστή ή αφαιρέτη

Μη προσημασμένοι συγκριτές των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

- Βασίζονται στο πρότυπο πακέτο **numeric_std** της βιβλιοθήκης **IEEE**
  - *Υποστηρίζεται από το **VIVADO** και τη **βιβλιοθήκη έτοιμων στοιχείων** της **XILINX***
- Όταν οι τελεστέοι είναι τύπου **unsigned**, τότε
  - *ο ένας από τους δύο τελεστέους μπορεί να είναι και τύπου **natural***
  - *εάν δεν έχουν το ίδιο μέγεθος γίνεται **επέκταση μηδενός***
  - *εάν ο τελεστέος **LV** είναι τύπου **std_logic_vector**, πρέπει πρώτα να μετατραπεί σε τελεστέο **U** τύπου **unsigned** χρησιμοποιώντας τη συνάρτηση:*

    ```
    U = unsigned(LV)
    ```

  - *το **αποτέλεσμα της σύγκρισης** είναι **Boolean***
- Συνθέτουμε συγκριτές χρησιμοποιώντας τους τελεστές:
  - **Equal – EQ (=), Not equal – NE (/=)**
  - **Higher – HI (>), Lower or same – LS (<=)**
  - **Lower – LO (<), Higher or same – HS (>=)**

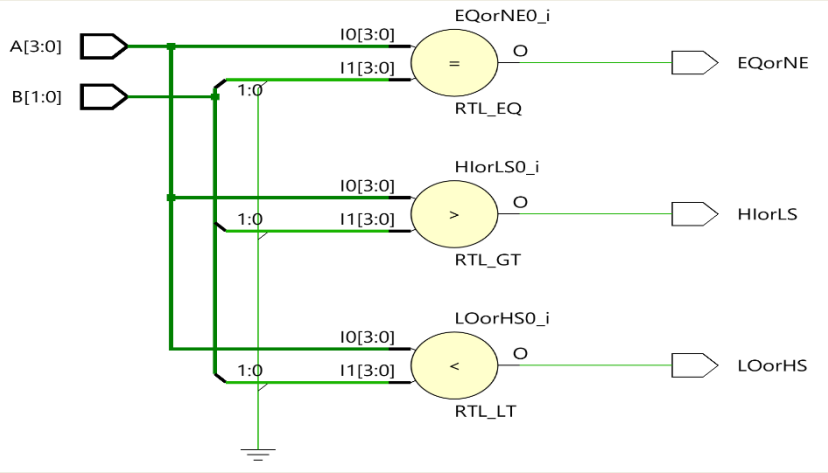
Χρήση του τύπου **unsigned**

Μη προσημασμένοι συγκριτές των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity COMP4u is
  port (
    A : in  STD_LOGIC_VECTOR (3 downto 0);
    B : in  STD_LOGIC_VECTOR (1 downto 0);
    EQorNE, HIorLS, LOorHS : out STD_LOGIC);
end COMP4u;
architecture BEHAVIORAL of COMP4u is
begin
  COMP4u: process (A, B)
  variable A_u : UNSIGNED (3 downto 0); -- 4 bit
  variable B_u : UNSIGNED (1 downto 0); -- 2 bit to be zero_ext
  begin
    A_u := unsigned(A);                          -- numeric_std
    B_u := unsigned(B);                          -- numeric_std
    if (A_u = B_u)  then EQorNE <= '1';          -- numeric_std
                    else EQorNE <= '0'; end if;  -- numeric_std
    if (A_u > B_u)  then HIorLS <= '1';          -- numeric_std
                    else HIorLS <= '0'; end if;  -- numeric_std
    if (A_u < B_u)  then LOorHS <= '1';          -- numeric_std
                    else LOorHS <= '0'; end if;  -- numeric_std
  end process;
end BEHAVIORAL;
```

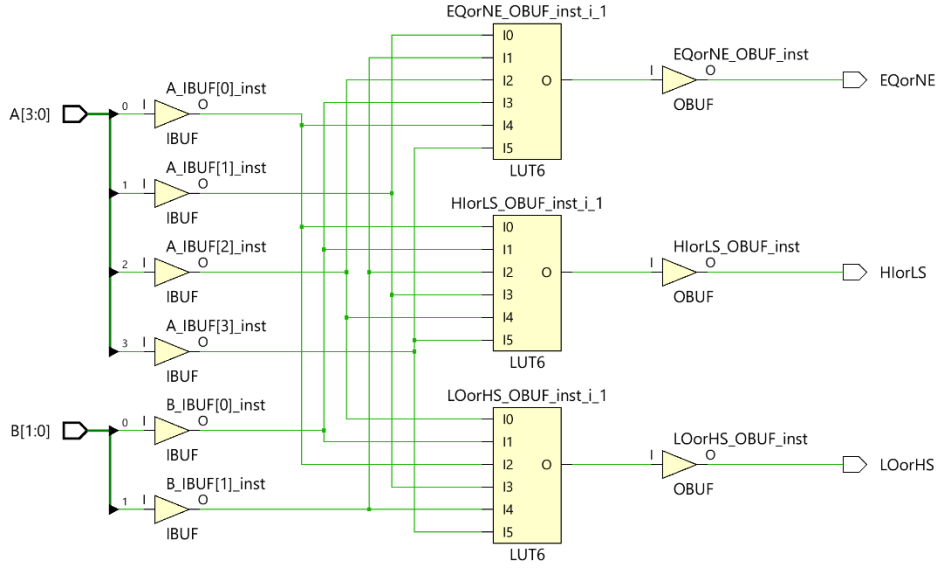
Μη προσημασμένοι συγκριτές των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



Επέκταση μηδενός πριν τη σύγκριση

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Υλοποίηση με **3 LUTs** σε **1 επίπεδο LUT** **λόγω των συνολικά 6 εισόδων**

Προσημασμένοι συγκριτές των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

- Βασίζονται στο πρότυπο πακέτο **numeric_std** της βιβλιοθήκης **IEEE**
  - *Υποστηρίζεται από το **VIVADO** και τη **βιβλιοθήκη έτοιμων στοιχείων** της **XILINX***
- Όταν οι τελεστέοι είναι τύπου **signed**, τότε
  - *ο ένας από τους δύο τελεστέους μπορεί να είναι και τύπου **integer***
  - *εάν δεν έχουν το ίδιο μέγεθος γίνεται **επέκταση πρόσημου***
  - *εάν ο τελεστέος **LV** είναι τύπου **std_logic_vector**, πρέπει πρώτα να μετατραπεί σε τελεστέο **S** τύπου **signed** χρησιμοποιώντας τη συνάρτηση:*

    ```
    U = signed(LV)
    ```

  - *το **αποτέλεσμα της σύγκρισης** είναι **Boolean***
- Συνθέτουμε συγκριτές χρησιμοποιώντας τους τελεστές:
  - Equal – EQ (=), Not equal – NE (/=)
  - Greater than – GT (>), Less than or equal – LE (<=)
  - Less than – LT (<), Greater than or equal– GE (>=)

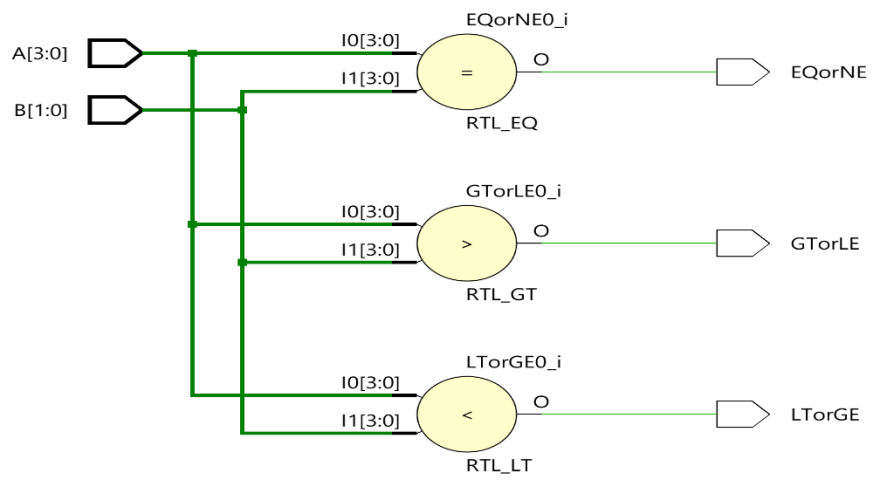
Χρήση του τύπου **signed**

Προσημασμένοι συγκριτές των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity COMP4s is
  port (
    A : in  STD_LOGIC_VECTOR (3 downto 0);
    B : in  STD_LOGIC_VECTOR (1 downto 0);
    EQorNE, GTorLE, LTorGE : out STD_LOGIC);
end COMP4s;
architecture BEHAVIORAL of COMP4s is
begin
  COMP4s: process (A, B)
  variable A_s : SIGNED (3 downto 0);   -- 4 bit
  variable B_s : SIGNED (1 downto 0);   -- 2 bit to be sign_ext
  begin
    A_s := signed(A);                                -- numeric_std
    B_s := signed(B);                                -- numeric_std
    if (A_s = B_s)  then EQorNE <= '1';              -- numeric_std
                    else EQorNE <= '0'; end if;      -- numeric_std
    if (A_s > B_s)  then GTorLE <= '1';              -- numeric_std
                    else GTorLE <= '0'; end if;      -- numeric_std
    if (A_s < B_s)  then LTorGE <= '1';              -- numeric_std
                    else LTorGE <= '0'; end if;      -- numeric_std
  end process;
end BEHAVIORAL;
```

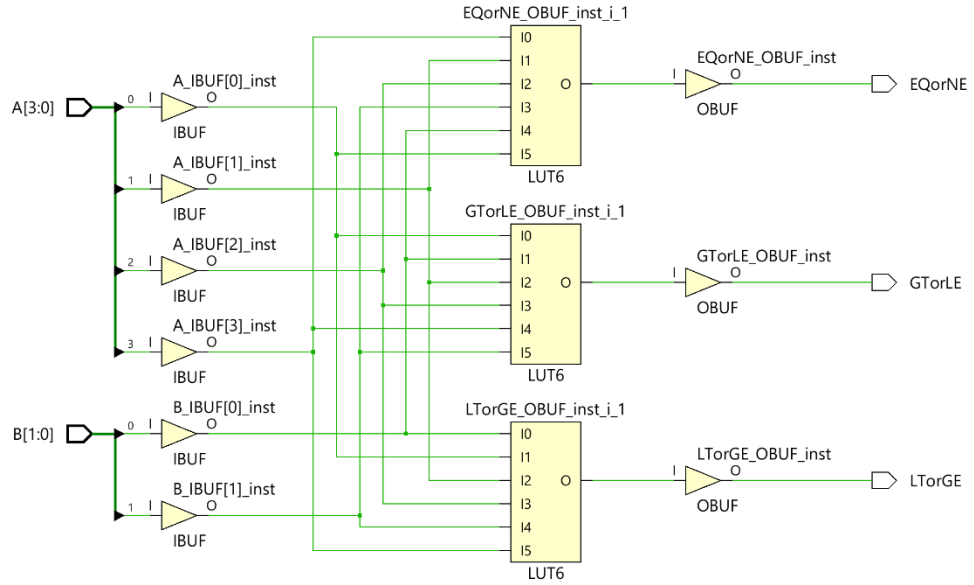
Προσημασμένοι συγκριτές των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



Επέκταση πρόσημου πριν τη σύγκριση

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Υλοποίηση με **3 LUTs** σε **1 επίπεδο LUT** **λόγω των συνολικά 6 εισόδων**

Μη προσημασμένοι ολισθητές των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

- Βασίζονται στο πρότυπο πακέτο **numeric_std** της βιβλιοθήκης **IEEE**
  - *Υποστηρίζεται από το **VIVADO** και τη **βιβλιοθήκη έτοιμων στοιχείων** της **XILINX***

Χρήση του τύπου **unsigned**

- Υλοποιούν τις **συναρτήσεις ολίσθησης**:
  - `vector2 <= SHIFT_LEFT (vector1, count)`
    - εκτελεί **αριστερή λογική ολίσθηση (LSL)** του **vector1** κατά **count** ψηφία
    - **vector1** και **vector2**: unsigned
    - **count**: natural
    - τα **κενά** γεμίζουν με '**0**'
    - τα **count** πιο αριστερά ψηφία του **vector1** χάνονται
  - `vector2 <= SHIFT_RIGHT (vector1, count)`
    - εκτελεί **δεξιά λογική ολίσθηση (LSR)** του **vector1** κατά **count** ψηφία
    - **vector1** και **vector2**: unsigned
    - **count**: natural
    - τα **κενά** γεμίζουν με '**0**'
    - τα **count** πιο δεξιά ψηφία του **vector1** χάνονται

Προσημασμένοι ολισθητές των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

- Βασίζονται στο πρότυπο πακέτο **numeric_std** της βιβλιοθήκης **IEEE**
  - *Υποστηρίζεται από το **VIVADO** και τη **βιβλιοθήκη έτοιμων στοιχείων** της **XILINX***

Χρήση του τύπου **signed**

- Υλοποιούν τις **συναρτήσεις ολίσθησης**:
  - `vector2 <= SHIFT_LEFT (vector1, count)`
    - εκτελεί **αριστερή αριθμητική ολίσθηση (ASL)** του **vector1** κατά **count** ψηφία
    - **vector1** και **vector2**: **signed**
    - **count**: **natural**
    - τα **κενά** γεμίζουν με '0'
    - τα **count** πιο αριστερά ψηφία του **vector1** χάνονται

Η αριστερή λογική ολίσθηση και η αριστερή αριθμητική ολίσθηση έχουν το ίδιο αποτέλεσμα

  - `vector2 <= SHIFT_RIGHT (vector1, count)`
    - εκτελεί **δεξιά αριθμητική ολίσθηση (ASR)** του **vector1** κατά **count** ψηφία
    - **vector1** και **vector2**: **signed**
    - **count**: **natural**
    - τα **κενά** γεμίζουν με **bit πρόσημου** (0: θετικοί αριθμοί, 1: αρνητικοί αριθμοί)
    - τα **count** πιο δεξιά ψηφία του **vector1** χάνονται

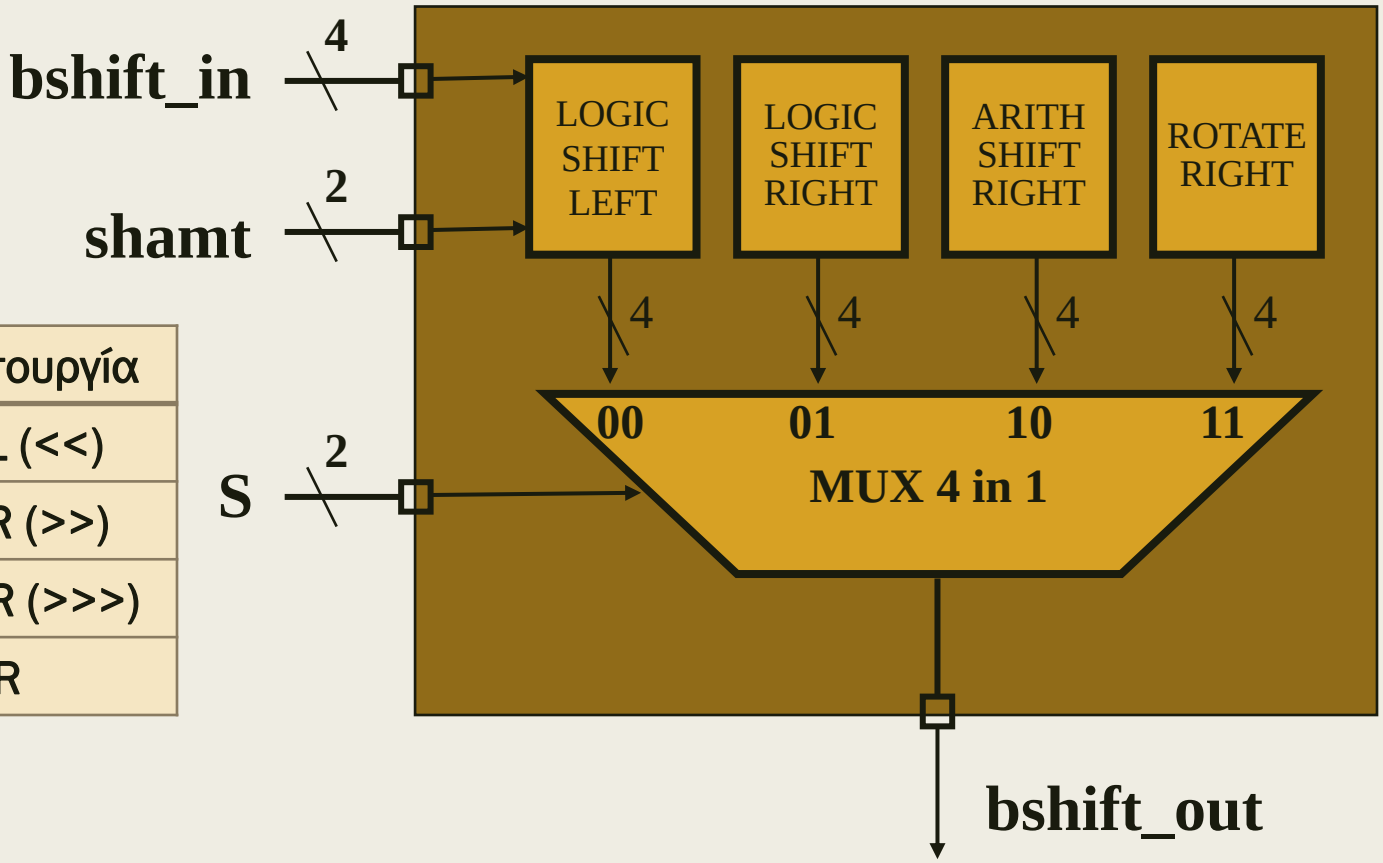
Περιστροφείς των 4 bit στη VHDL
Περιγραφή συμπεριφοράς

- Βασίζονται στο πρότυπο πακέτο **numeric_std** της βιβλιοθήκης **IEEE**
  - *Υποστηρίζεται από το **VIVADO** και τη **βιβλιοθήκη έτοιμων στοιχείων** της **XILINX***
- Υλοποιούν τις **συναρτήσεις περιστροφής**:
  - `vector2 <= ROTATE_LEFT (vector1, count)`
    - εκτελεί **αριστερή περιστροφή** (ROL) του **vector1** κατά **count** ψηφία
    - **vector1** και **vector2**: unsigned ή signed
    - **count**: natural
    - τα **κενά** γεμίζουν με τα **count** πιο αριστερά ψηφία του **vector1**
  - `vector2 <= ROTATE_RIGHT (vector1, count)`
    - εκτελεί **δεξιά περιστροφή** (ROR) του **vector1** κατά **count** ψηφία
    - **vector1** και **vector2**: unsigned ή signed
    - **count**: natural
    - τα **κενά** γεμίζουν με τα **count** πιο δεξιά ψηφία του **vector1**

Χρήση είτε του τύπου unsigned, είτε του τύπου signed.
Το αποτέλεσμα της περιστροφής είναι ίδιο.

Κύκλωμα ολίσθησης των 4 bit στη VHDL

Περιγραφή συμπεριφοράς



$S_{1:0}$	Λειτουργία
00	LSL (<<)
01	LSR (>>)
10	ASR (>>>)
11	ROR

Κύκλωμα ολίσθησης των 4 bit στη VHDL

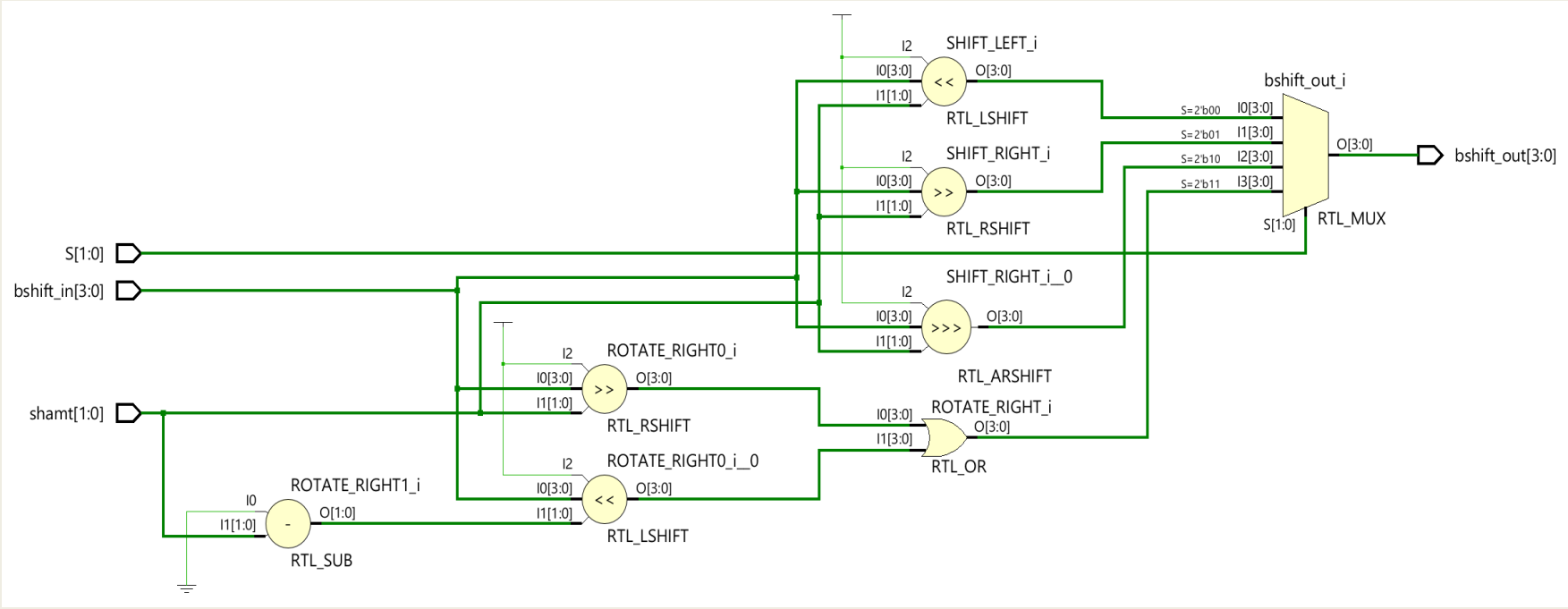
Περιγραφή συμπεριφοράς

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all
entity BSHIFTER4 is                                          -- barrel shifter
  port (S          : in  STD_LOGIC_VECTOR (1 downto 0);      -- επιλογή ολίσθησης
        shamt      : in  STD_LOGIC_VECTOR (1 downto 0);      -- ποσότητα ολίσθησης
        bshift_in  : in  STD_LOGIC_VECTOR (3 downto 0);
        bshift_out : out STD_LOGIC_VECTOR (3 downto 0));
end BSHIFTER4;
architecture BEHAVIORAL of BSHIFTER4 is
begin
  BSHIFTER4: process (S, shamt, bshift_in)
  variable shamt_n : NATURAL range 0 to 3;
  variable X_u : UNSIGNED (3 downto 0);
  variable X_s : SIGNED   (3 downto 0);
  begin
    shamt_n := to_integer(unsigned(shamt));                  -- numeric_std
    X_u := unsigned (bshift_in);                             -- numeric_std
    X_s := signed   (bshift_in);                             -- numeric_std
    case S is
      when "00" =>   bshift_out <= std_logic_vector(SHIFT_LEFT   (X_u, shamt_n));
      when "01" =>   bshift_out <= std_logic_vector(SHIFT_RIGHT  (X_u, shamt_n));
      when "10" =>   bshift_out <= std_logic_vector(SHIFT_RIGHT  (X_s, shamt_n));
      when "11" =>   bshift_out <= std_logic_vector(ROTATE_RIGHT (X_s, shamt_n));
      when others => bshift_out <= "----";
    end case;
  end process;
end BEHAVIORAL;
```

Κύκλωμα ολίσθησης των 4 bit στη VHDL

Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL

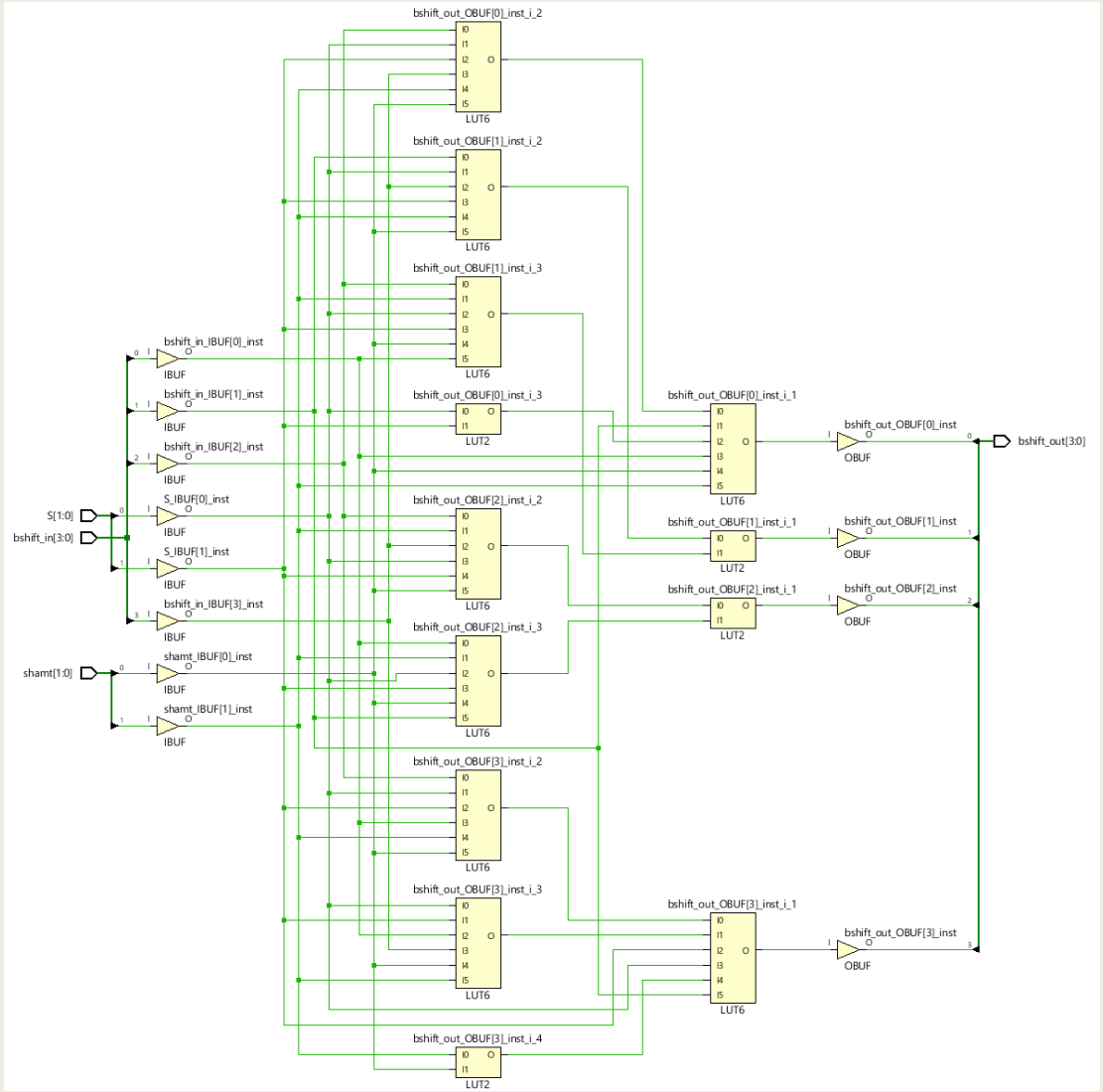


Υλοποίηση του ROR με LSR by n, LSL by (4-n) και OR

Κύκλωμα ολίσθησης των 4 bit στη VHDL

Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Υλοποίηση με **13 LUTs** σε **2 επίπεδα LUT**

Μετρητής με Reset των 4 bit στη VHDL

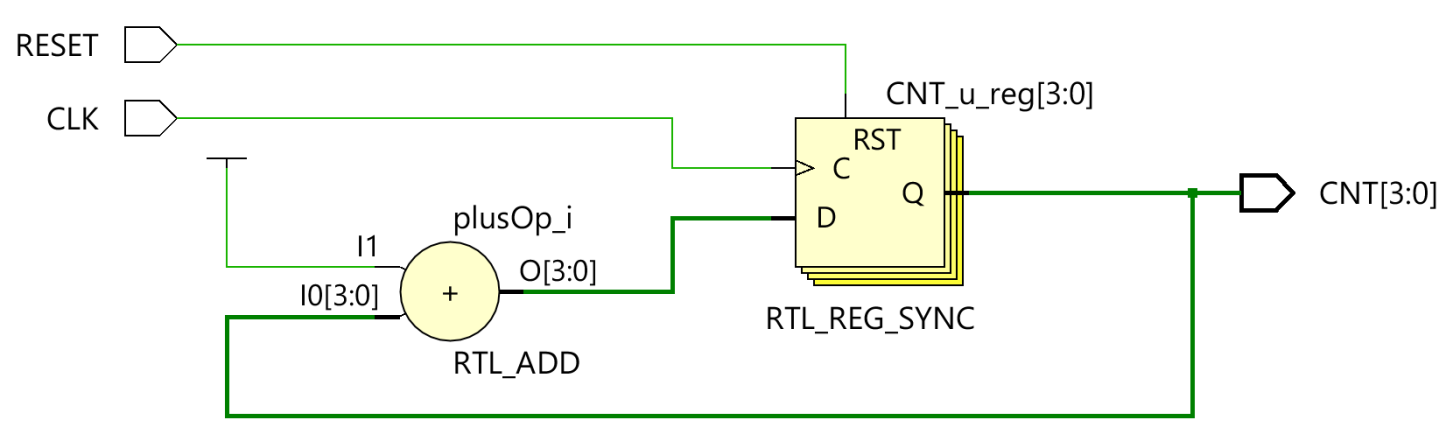
Περιγραφή συμπεριφοράς

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity COUNTERr is
  port (
    CLK :   in STD_LOGIC;
    RESET : in STD_LOGIC;
    CNT :  out STD_LOGIC_VECTOR (3 downto 0));
end COUNTERr;
architecture BEHAVIORAL of COUNTERr is
begin
  COUNTERr : process (CLK)
  variable CNT_u : UNSIGNED (3 downto 0);
  begin
    if (CLK = '1' and CLK'event) then
      if (RESET = '1') then
        CNT_u := (others => '0');
      else
        CNT_u := CNT_u + 1;          -- numeric_std
      end if;
    end if;
    CNT <= std_logic_vector(CNT_u);  -- numeric_std
  end process;
end BEHAVIORAL;
```

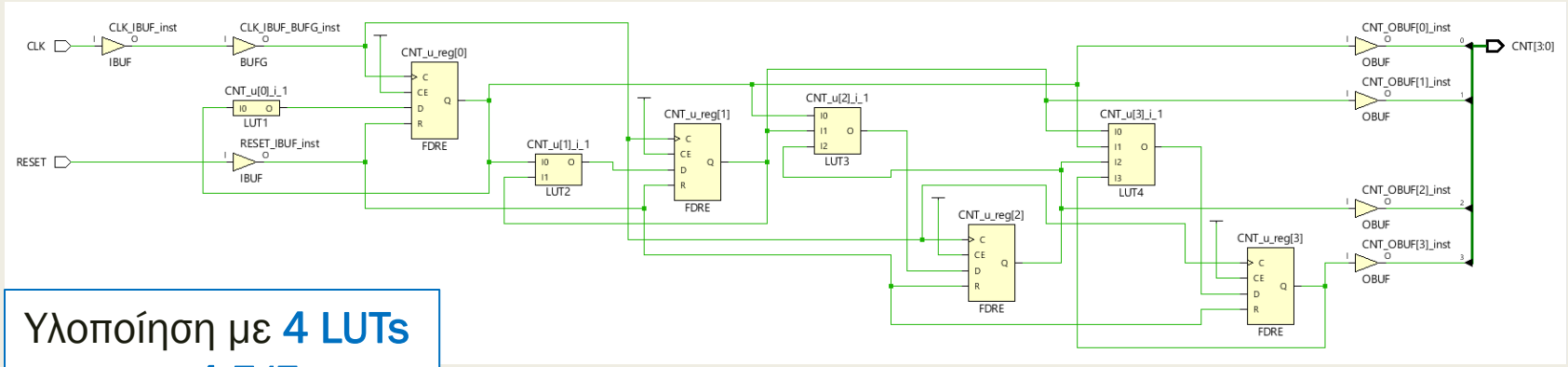
Μετρητής με Reset των 4 bit στη VHDL

Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



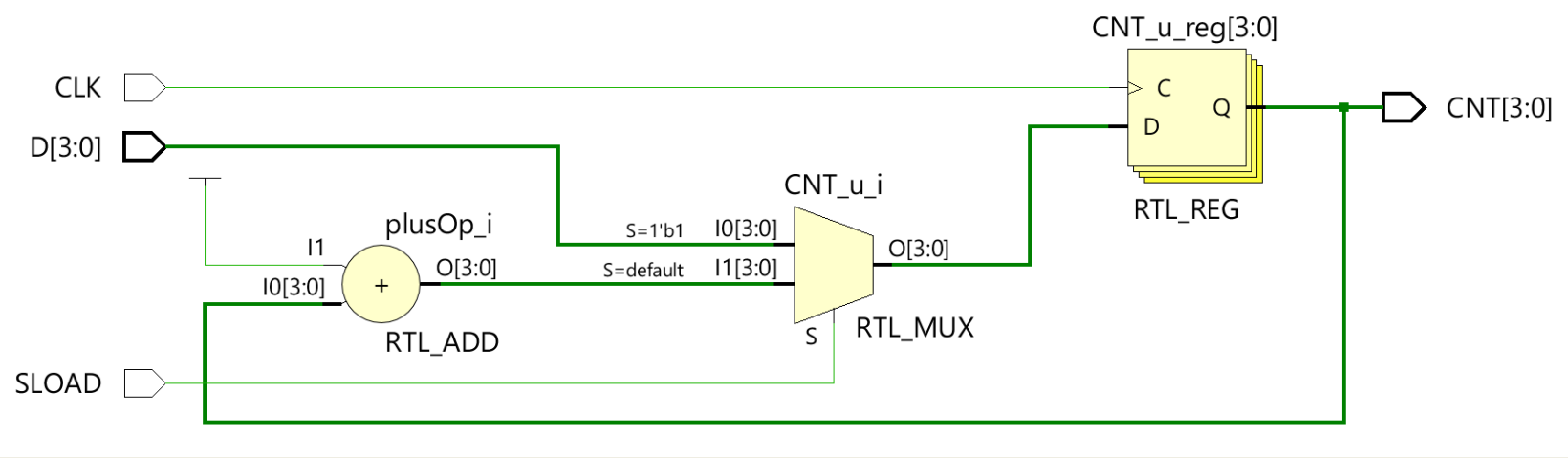
Υλοποίηση με **4 LUTs** και **4 F/Fs**

Μετρητής με σύγχρονη φόρτωση μεταβλητής των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

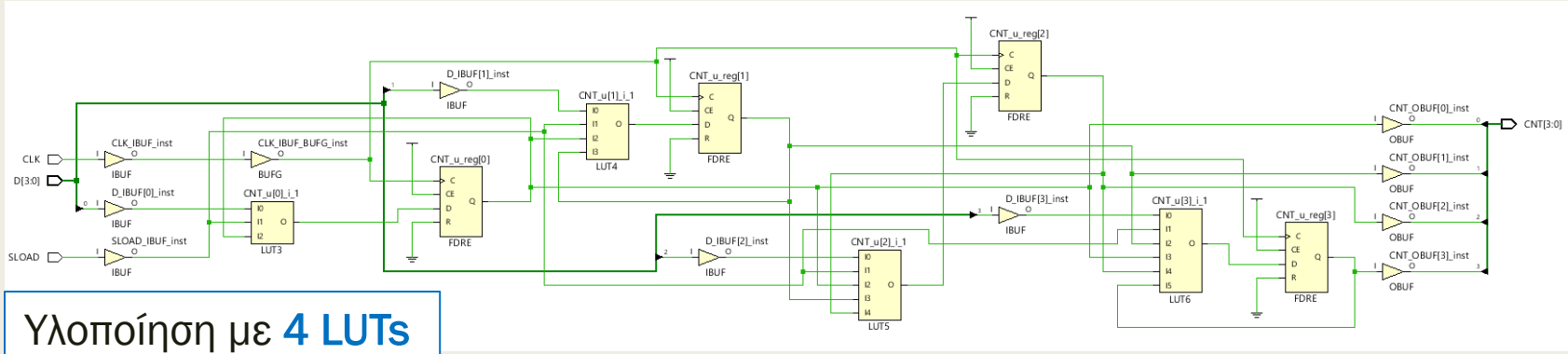
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity COUNTER1 is
  port (
    CLK :   in STD_LOGIC;
    SLOAD : in STD_LOGIC;
    D   :   in STD_LOGIC_VECTOR (3 downto 0);
    CNT :  out STD_LOGIC_VECTOR (3 downto 0));
end COUNTER1;
architecture BEHAVIORAL of COUNTER1 is
begin
  COUNTER1 : process (CLK)
  variable CNT_u : UNSIGNED (3 downto 0);
  begin
    if (CLK = '1' and CLK'event) then
      if (SLOAD = '1') then
        CNT_u := unsigned(D);          -- numeric_std
      else
        CNT_u := CNT_u + 1;            -- numeric_std
      end if;
    end if;
    CNT <= std_logic_vector(CNT_u);    -- numeric_std
  end process;
end BEHAVIORAL;
```

Μετρητής με σύγχρονη φόρτωση μεταβλητής τιμής των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Υλοποίηση με **4 LUTs** και **4 F/Fs**

Μετρητής με σύγχρονη φόρτωση σταθεράς τιμής και σύγχρονο EN των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity COUNTERlen is
  port (
    CLK :   in STD_LOGIC;
    SLOAD : in STD_LOGIC;
    EN :    in STD_LOGIC;
    DONE : out STD_LOGIC;
    CNT :  out STD_LOGIC_VECTOR (3 downto 0));
end COUNTERlen;
architecture BEHAVIORAL of COUNTERlen is
begin
  COUNTERlen : process (CLK)
  variable CNT_u : UNSIGNED (3 downto 0);
  begin
    if (CLK = '1' and CLK'event) then
      if (SLOAD = '1') then CNT_u := to_unsigned(10, 4);  -- numeric_std
       elsif (EN = '1') then CNT_u := CNT_u - 1; end if;   -- numeric_std
    end if;
    CNT <= std_logic_vector(CNT_u);                       -- numeric_std
    if (CNT_u = 0) then DONE <= '1'; else DONE <= '0'; end if;
  end process;
end BEHAVIORAL;
```

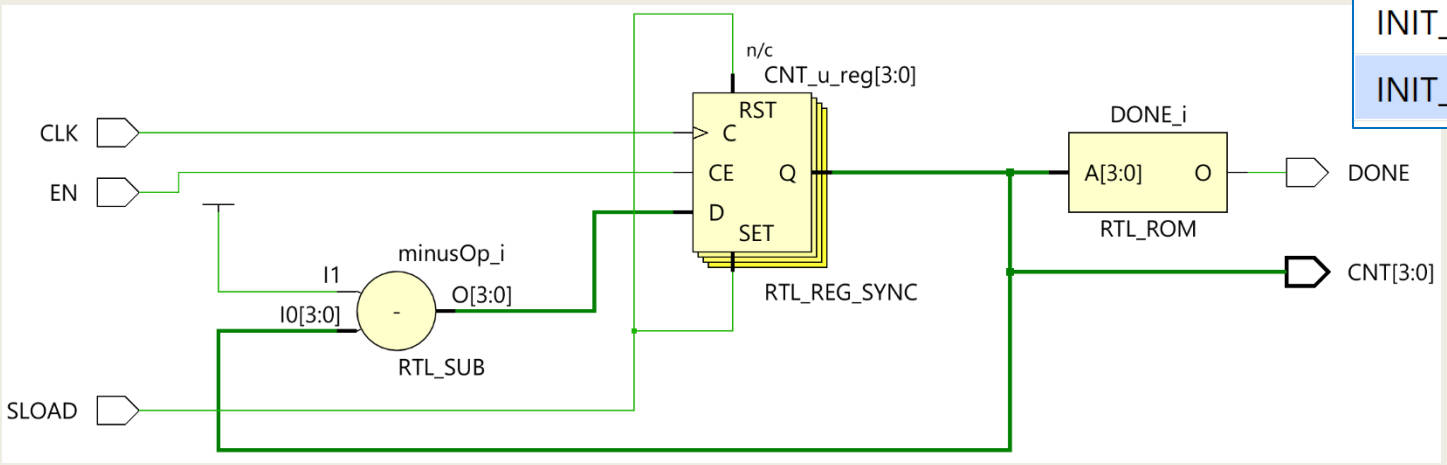
Μετρητής 10 γεγονότων

Ο μετρητής αρχικοποιείται σύγχρονα στην τιμή 10 και μειώνει το περιεχόμενό του κατά 1, όταν το σύστημα αναγνωρίζει κάποιο γεγονός (EN = 1). Μετά το πέρας 10 γεγονότων, το περιεχόμενο του μετρητή γίνεται 0 και ενεργοποιείται το σήμα DONE (DONE =1)

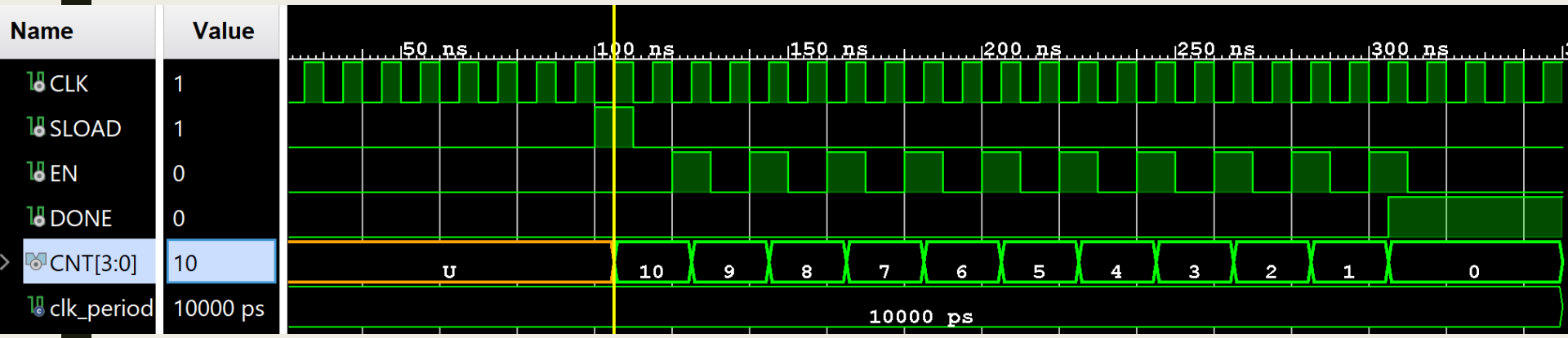
Μετρητής με σύγχρονη φόρτωση σταθεράς τιμής και σύγχρονο EN των 4 bit στη VHDL – Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL

INIT	Value
INIT_DEFAULT	1'b0
INIT_0	1'b1



- Χρονικό διάγραμμα της προσομοίωσης συμπεριφοράς



Προσοχή στο πρόγραμμα δοκιμής: Το σύγχρονο σήμα SLOAD ενεργοποιείται μετά τα πρώτα 100 ns. Η ενεργοποίηση της εξόδου DONE επιβάλει το σήμα EN να παραμένει 0.

Παραμετροποίηση στη VHDL

- Θα εξετάσουμε δύο περιπτώσεις παραμετροποίησης:
  - *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 **σταθερά WIDTH** λαμβάνει μπορεί να παρακάμψει την προεπιλεγμένη τιμή με τη φράση **generic map** σε μία ταυτόχρονη εντολή στοιχείων (concurrent_component_statement)
      - `generic map (WIDTH => 8)`
  - *Παραμετροποίηση του **πλήθους των αντιγράφων μίας κυψελίδας** που παράγονται με την εντολή FOR GENERATE*
    - Η **σταθερά WIDTH** χρησιμοποιείται για τον προσδιορισμό του αριθμού των επαναλήψεων **I**
      - `for I in WIDTH-1 downto 0 generate`

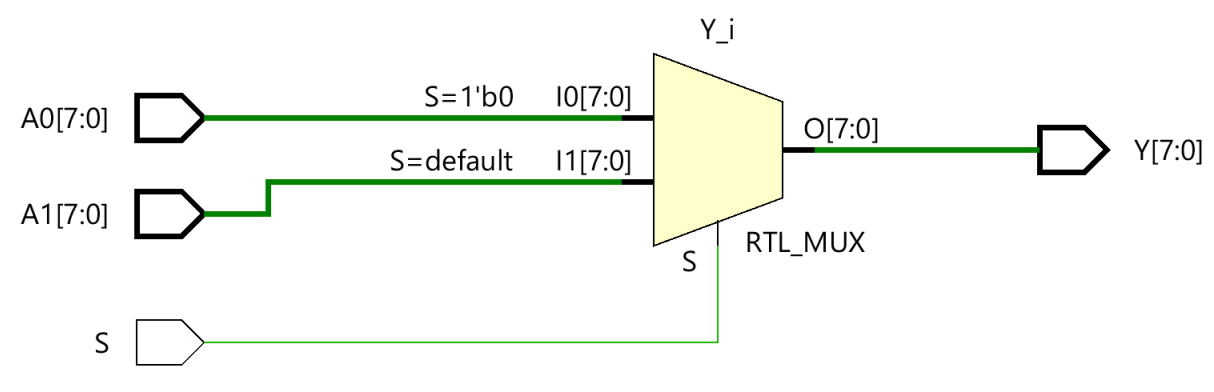
Παραμετροποίηση του μεγέθους μίας αρτηρίας στη VHDL

- Παράδειγμα παραμετροποιημένου πολυπλέκτη 2 σε 1 των N bit

```
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;
```

Παραμετροποίηση του μεγέθους μίας αρτηρίας στη VHDL

- Σχηματικό διάγραμμα RTL του παραμετροποιημένου πολυπλέκτη 2 σε 1 των N bit (λαμβάνεται υπόψη η προεπιλεγμένη τιμή των 8 bit)



Παραμετροποίηση του μεγέθους μίας αρτηρίας στη VHDL

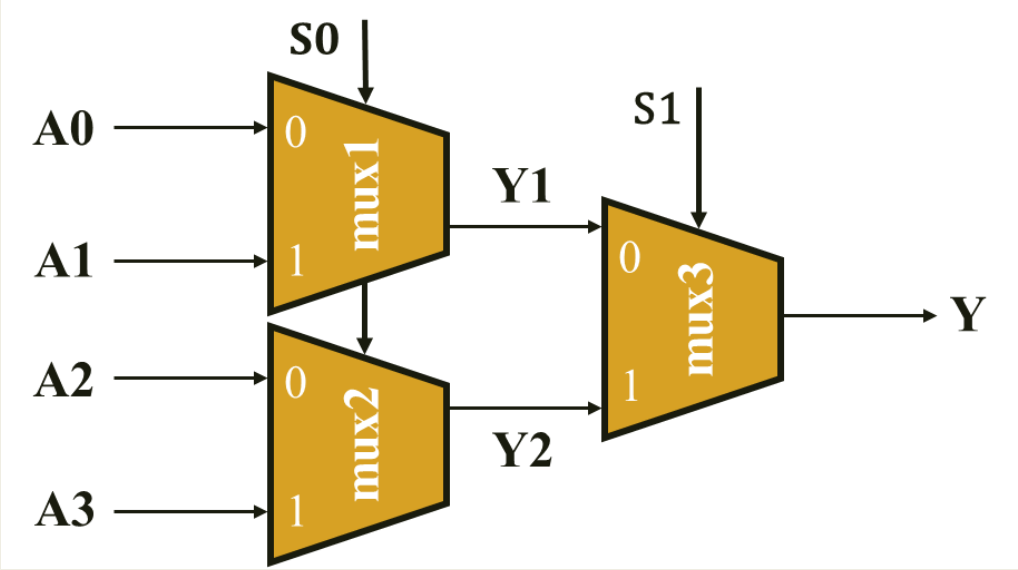
- Παράδειγμα χρήσης παραμετροποιημένου πολυπλέκτη 2 σε 1 των N bit ως στοιχείου (component) σε πολυπλέκτη 4 se1 των 16 bit

```
entity MUX4to1_16 is
  port (
    S:  in STD_LOGIC_VECTOR ( 1 downto 0);
    A0: in STD_LOGIC_VECTOR (15 downto 0);
    A1: in STD_LOGIC_VECTOR (15 downto 0);
    A2: in STD_LOGIC_VECTOR (15 downto 0);
    A3: in STD_LOGIC_VECTOR (15 downto 0);
    Y: out STD_LOGIC_VECTOR (15 downto 0));
end MUX2to1_n;
architecture BEHAVIORAL of MUX4to1_16 is
component MUX2to1_n
  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 component;
signal Y1, Y2 : STD_LOGIC_VECTOR (15 downto 0);
...
```

Παραμετροποίηση του μεγέθους μίας αρτηρίας στη VHDL

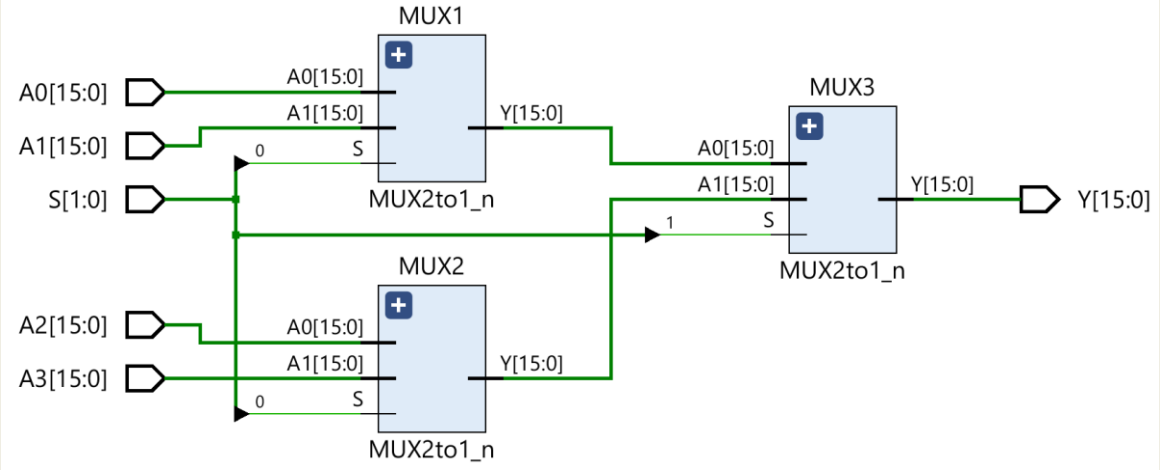
- Παράδειγμα χρήσης του παραμετροποιημένου πολυπλέκτη 2 σε 1 των N bit ως στοιχείου (component) στον πολυπλέκτη 4 σε 1 των 16

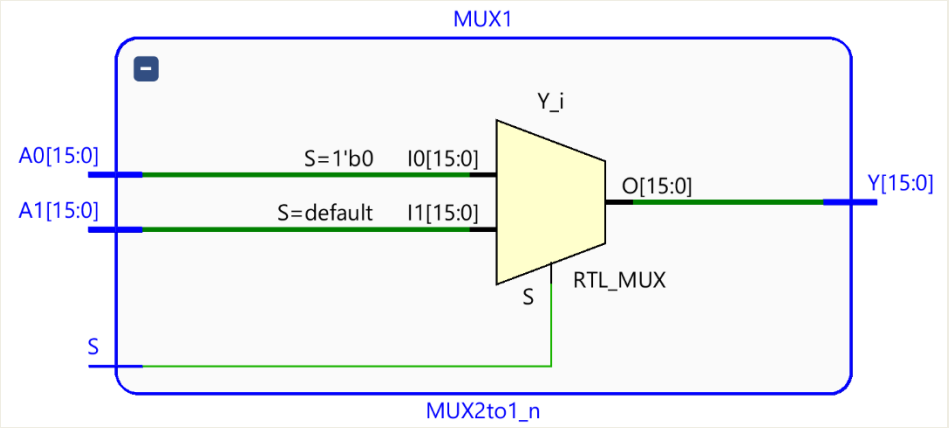
```
...
begin
MUX1: MUX2to1_n generic map (WIDTH => 16) port map
      (S => S(0), A0 => A0, A1 => A1, Y => Y1);
MUX2: MUX2to1_n generic map (WIDTH => 16) port map
      (S => S(0), A0 => A2, A1 => A3, Y => Y2);
MUX3: MUX2to1_n generic map (WIDTH => 16) port map
      (S => S(1), A0 => Y1, A1 => Y2, Y => Y);
end BEHAVIORAL;
```



Παραμετροποίηση του μεγέθους μίας αρτηρίας στη VHDL

- Σχηματικό διάγραμμα RTL του πολυπλέκτη 4 σε 1 των 16 bit
  - *χρησιμοποιεί ως στοιχείο τον παραμετροποιημένο πολυπλέκτη 2 σε 1 των N bit*
  - *δεν λαμβάνεται υπόψη η προεπιλεγμένη τιμή των 8 bit, αλλά η τιμή των 16 bit που δηλώνεται στο generic map*





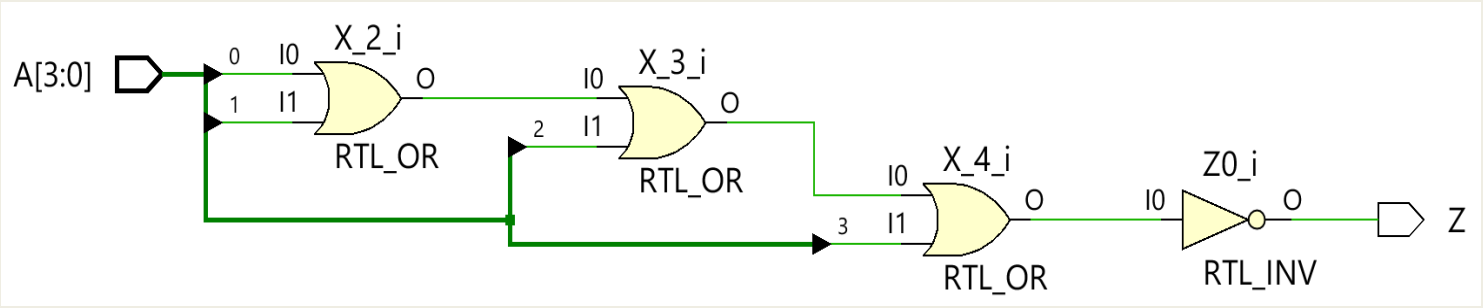
Παραμετροποίηση του πλήθους των αντιγράφων μίας κυψελίδας

- Παράδειγμα παραμετροποιημένης πύλης NOR των N bit που δημιουργείται με τη χρήση της εντολής **FOR GENERATE**
  - *Υπολογίζει την τιμή της σημαίας Z (όταν η είσοδος είναι 0, τότε Z=1)*

```
entity NOR_n is
  generic (WIDTH : positive := 4); -- προεπιλεγμένη τιμή
  port (
    A: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    Z: out STD_LOGIC);
end NOR_n;
architecture STRUCTURAL of NOR_n is
signal X : STD_LOGIC_VECTOR (WIDTH downto 0);
begin
  X(0) <= '0';
  G1: for I in 0 to WIDTH-1 generate
    X(I+1) <= X(I) or A(I);
  Z <= not X(WIDTH);
  end generate G1;
end STRUCTURAL;
```

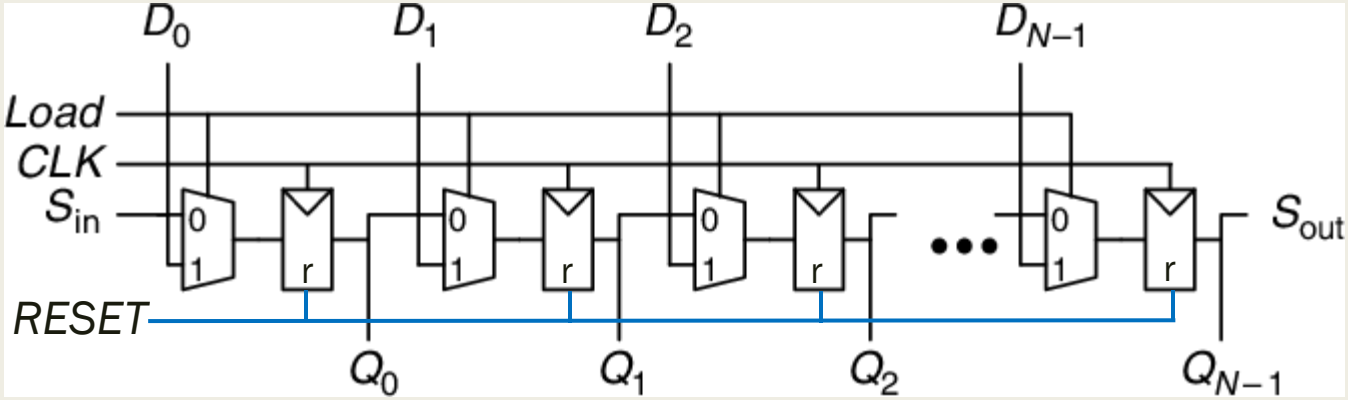
Παραμετροποίηση του πλήθους των αντιγράφων μίας κυψελίδας

- Σχηματικό διάγραμμα RTL της παραμετροποιημένης πύλης NOR των N bit (λαμβάνεται υπόψη η προεπιλεγμένη τιμή των 4 bit)



Καταχωρητής ολίσθησης με παράλληλη φόρτωση και RESET στη VHDL - Περιγραφή συμπεριφοράς

- Λειτουργεί ως **μετατροπέας παράλληλου σε σειριακό** (parallel-to-serial converter) που φορτώνει *N* bit παράλληλα στην είσοδο $D_{N-1:0}$, και κατόπιν τα ολισθαίνει στην σειριακή έξοδο S_{out}, ένα τη φορά
- Λειτουργεί ως **μετατροπέας σειριακού σε παράλληλο** (serial-to-parallel converter) που φορτώνει *N* bit σειριακά από την σειριακή είσοδο S_{in} και κατόπιν τα εμφανίζει παράλληλα στην έξοδο $Q_{N-1:0}$
- Χρειάζεται ένα σήμα ελέγχου ***Load***
  - *Όταν Load = 1, το κύκλωμα λειτουργεί ως **παράλληλος καταχωρητής***
  - *Όταν Load = 0, το κύκλωμα λειτουργεί ως **καταχωρητής ολίσθησης***
- Χρειάζεται ένα σήμα ελέγχου ***RESET*** για επαναφορά στο 0



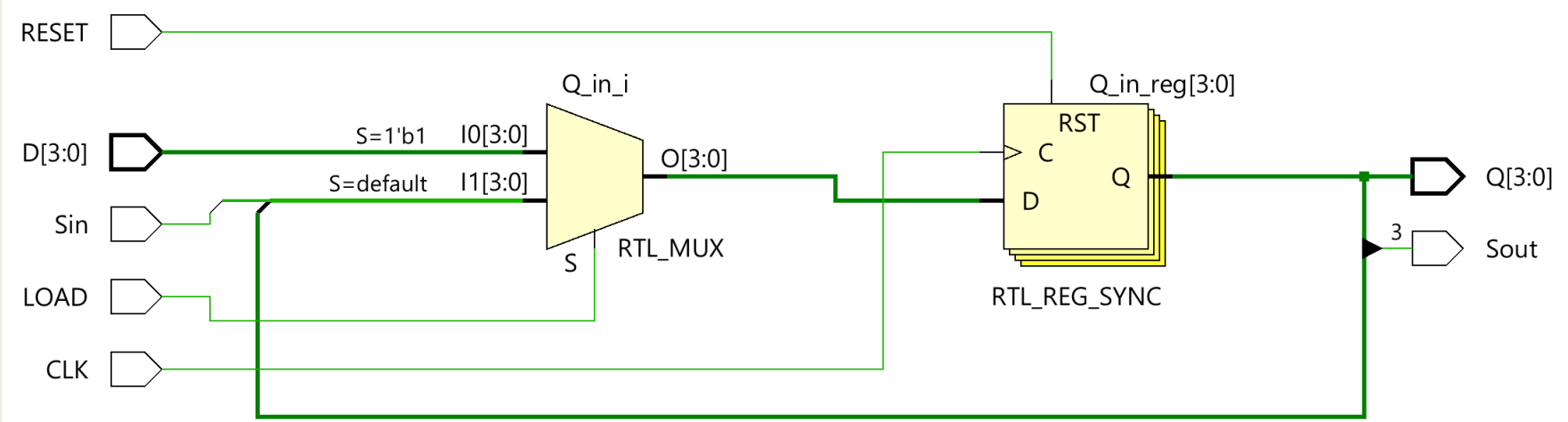
Σειριακή μετάδοση big endian (μεταδίδεται πρώτο το MSB, τελευταίο το LSB)

Καταχωρητής ολίσθησης με παράλληλη φόρτωση και RESET στη VHDL - Περιγραφή συμπεριφοράς

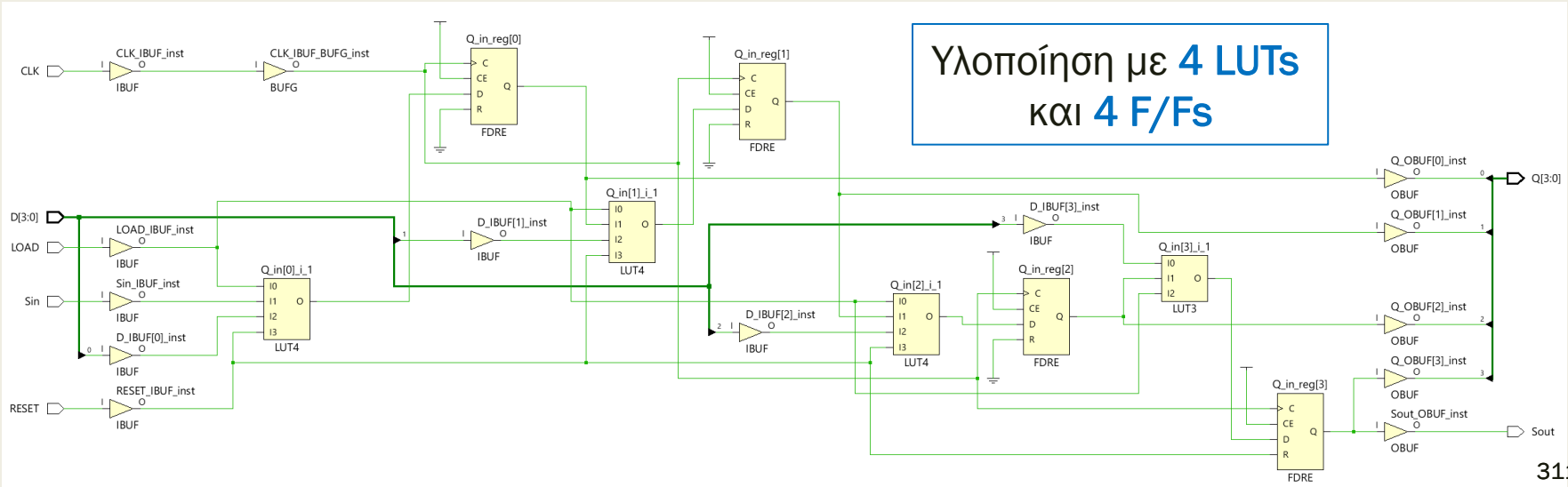
```
entity SHIFTREG_n is
  generic (WIDTH : positive := 4); -- προεπιλεγμένη τιμή
  port (
    CLK:   in STD_LOGIC;
    RESET: in STD_LOGIC; -- αρχικοποίηση στο 0
    LOAD:  in STD_LOGIC;
    Sin:   in STD_LOGIC;
    D:     in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    Sout: out STD_LOGIC;
    Q:    out STD_LOGIC_VECTOR (WIDTH-1 downto 0));
end SHIFTREG_n;
architecture BEHAVIORAL of SHIFTREG_n is
signal Q_in : STD_LOGIC_VECTOR (WIDTH-1 downto 0);
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      if (RESET = '1') then Q_in <= (others => '0');
      elsif (LOAD = '1') then Q_in <= D;
      else Q_in <= Q_in(WIDTH-2 downto 0) & Sin;
      end if;
    end if;
  end process;
  Q <= Q_in;
  Sout <= Q_in(WIDTH-1);
end BEHAVIORAL;
```

Καταχωρητής ολίσθησης με παράλληλη φόρτωση και RESET στη VHDL - Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Αλλαγή μεγέθους στη VHDL
Περιγραφή συμπεριφοράς

- Βασίζεται στο πρότυπο πακέτο **numeric_std** της βιβλιοθήκης **IEEE**
  - *Υποστηρίζεται από το **VIVADO** και τη **βιβλιοθήκη έτοιμων στοιχείων** της **XILINX***
- Επιτυγχάνεται με τις **συναρτήσεις αλλαγής μεγέθους**:
  - `vector2 <= RESIZE (vector1, new_size)`
    - όταν το **new_size** είναι μεγαλύτερο γίνεται **επέκταση μηδενός**
    - όταν το **new_size** είναι μικρότερο γίνεται αποκοπή των πιο **αριστερών ψηφίων** που περισσεύουν
    - **vector1** και **vector2**: unsigned
    - **new_size**: natural

Χρήση του τύπου **unsigned**

  - `vector2 <= RESIZE (vector1, new_size)`
    - όταν το **new_size** είναι μεγαλύτερο γίνεται **επέκταση πρόσημου**
    - όταν το **new_size** είναι μικρότερο γίνεται αποκοπή των πιο **αριστερών ψηφίων** που περισσεύουν (**διατηρείται όμως το πρόσημο**)
    - **vector1** και **vector2**: signed
    - **new_size**: natural

Χρήση του τύπου **signed**

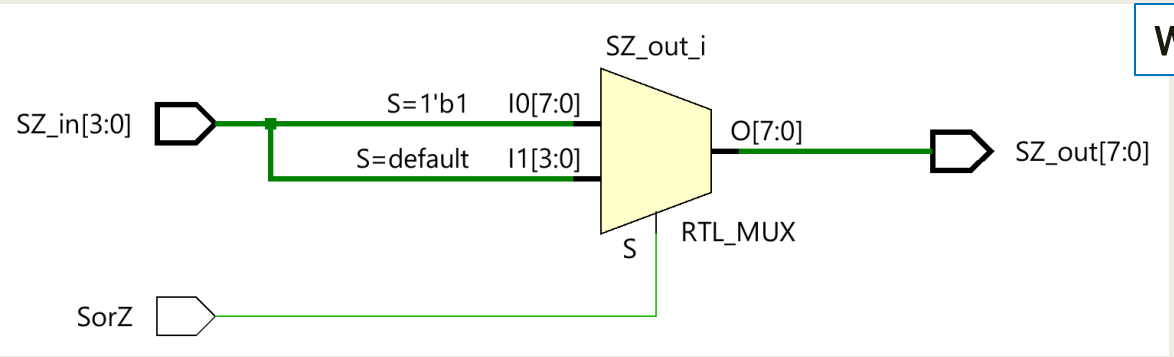
Μονάδα επέκτασης πρόσημου/μηδενός στη VHDL – Περιγραφή συμπεριφοράς

WIDTH_IN < WIDTH_OUT

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity SZEXTEND is
  generic (WIDTH_IN :  positive := 4;  -- προεπιλεγμένη τιμή
           WIDTH_OUT : positive := 8); -- προεπιλεγμένη τιμή
  port (
    SorZ:    in STD_LOGIC;
    SZ_in:   in STD_LOGIC_VECTOR (WIDTH_IN-1  downto 0);
    SZ_out: out STD_LOGIC_VECTOR (WIDTH_OUT-1 downto 0));
end SZEXTEND;
architecture BEHAVIORAL of SZEXTEND is
begin
  SZEXTEND : process (SorZ, SZ_in)
  variable SZ_in_u  : UNSIGNED (WIDTH_IN-1  downto 0);
  variable SZ_in_s  : SIGNED   (WIDTH_IN-1  downto 0);
  variable SZ_out_u : UNSIGNED (WIDTH_OUT-1 downto 0);
  variable SZ_out_s : SIGNED   (WIDTH_OUT-1 downto 0);
  begin
    SZ_in_u := unsigned (SZ_in);  -- numeric_std
    SZ_in_s := signed   (SZ_in);  -- numeric_std
    if (SorZ = '1') then SZ_out_s := RESIZE (SZ_in_s, WIDTH_OUT);
                         SZ_out   <= std_logic_vector(SZ_out_s);
                    else SZ_out_u := RESIZE (SZ_in_u, WIDTH_OUT);
                         SZ_out   <= std_logic_vector(SZ_out_u);
    end if;
  end process;
end BEHAVIORAL;
```

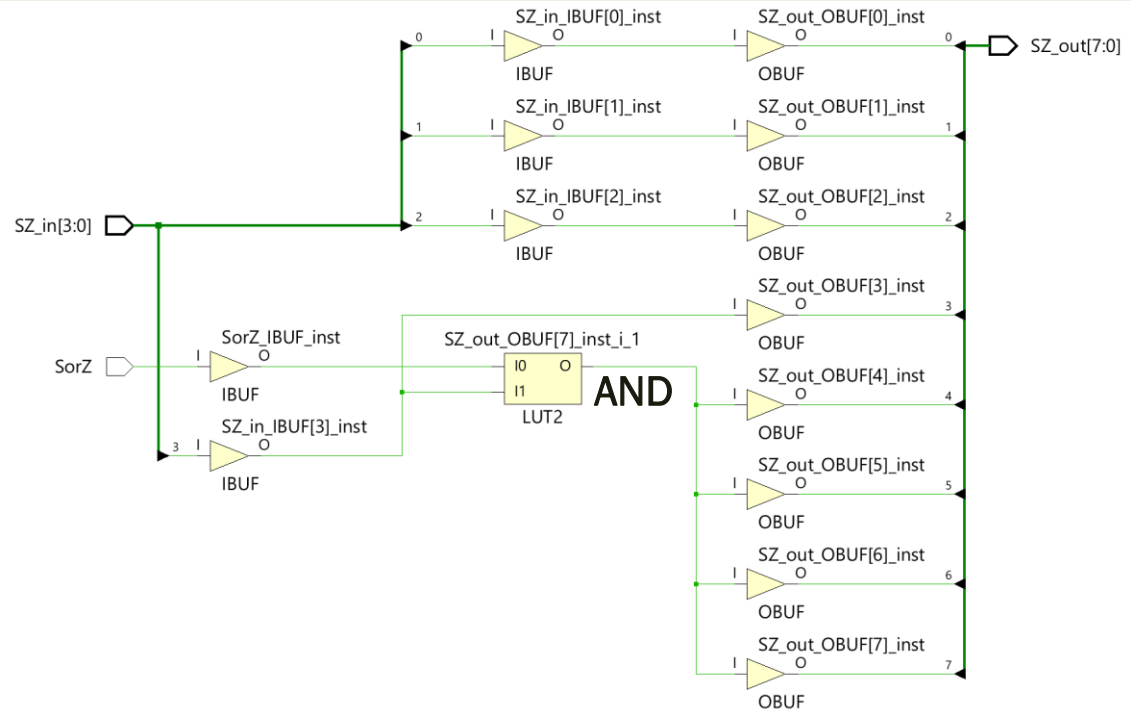
Μονάδα επέκτασης πρόσημου/μηδενός στη VHDL – Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



WIDTH_IN < WIDTH_OUT

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Υλοποίηση με **1 LUT**

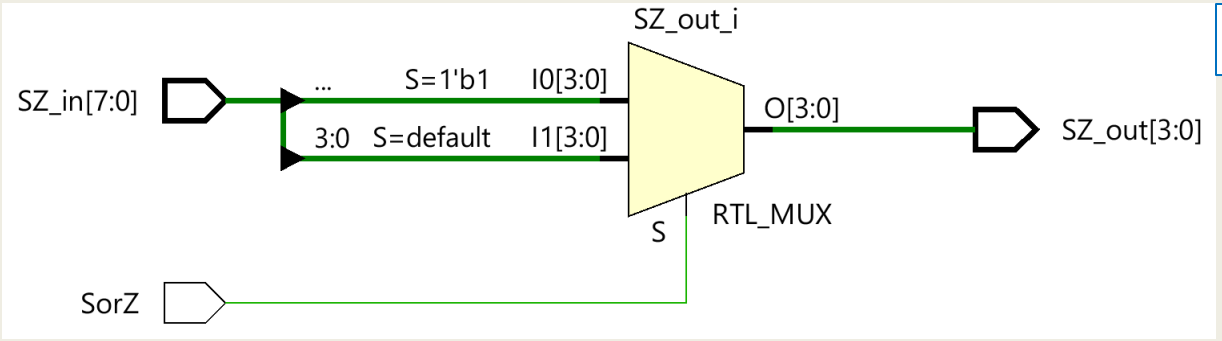
Μονάδα σμίκρυνσης πρόσημου/μηδενός στη VHDL – Περιγραφή συμπεριφοράς

WIDTH_IN > WIDTH_OUT

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity SZEXTEND is
  generic (WIDTH_IN :  positive := 8;  -- προεπιλεγμένη τιμή
           WIDTH_OUT : positive := 4); -- προεπιλεγμένη τιμή
  port (
    SorZ:    in STD_LOGIC;
    SZ_in:   in STD_LOGIC_VECTOR (WIDTH_IN-1  downto 0);
    SZ_out: out STD_LOGIC_VECTOR (WIDTH_OUT-1 downto 0));
end SZEXTEND;
architecture BEHAVIORAL of SZEXTEND is
begin
  SZEXTEND : process (SorZ, SZ_in)
  variable SZ_in_u  : UNSIGNED (WIDTH_IN-1  downto 0);
  variable SZ_in_s  : SIGNED   (WIDTH_IN-1  downto 0);
  variable SZ_out_u : UNSIGNED (WIDTH_OUT-1 downto 0);
  variable SZ_out_s : SIGNED   (WIDTH_OUT-1 downto 0);
  begin
    SZ_in_u := unsigned (SZ_in);  -- numeric_std
    SZ_in_s := signed   (SZ_in);  -- numeric_std
    if (SorZ = '1') then SZ_out_s := RESIZE (SZ_in_s, WIDTH_OUT);
                         SZ_out   <= std_logic_vector(SZ_out_s);
                    else SZ_out_u := RESIZE (SZ_in_u, WIDTH_OUT);
                         SZ_out   <= std_logic_vector(SZ_out_u);
    end if;
  end process;
end BEHAVIORAL;
```

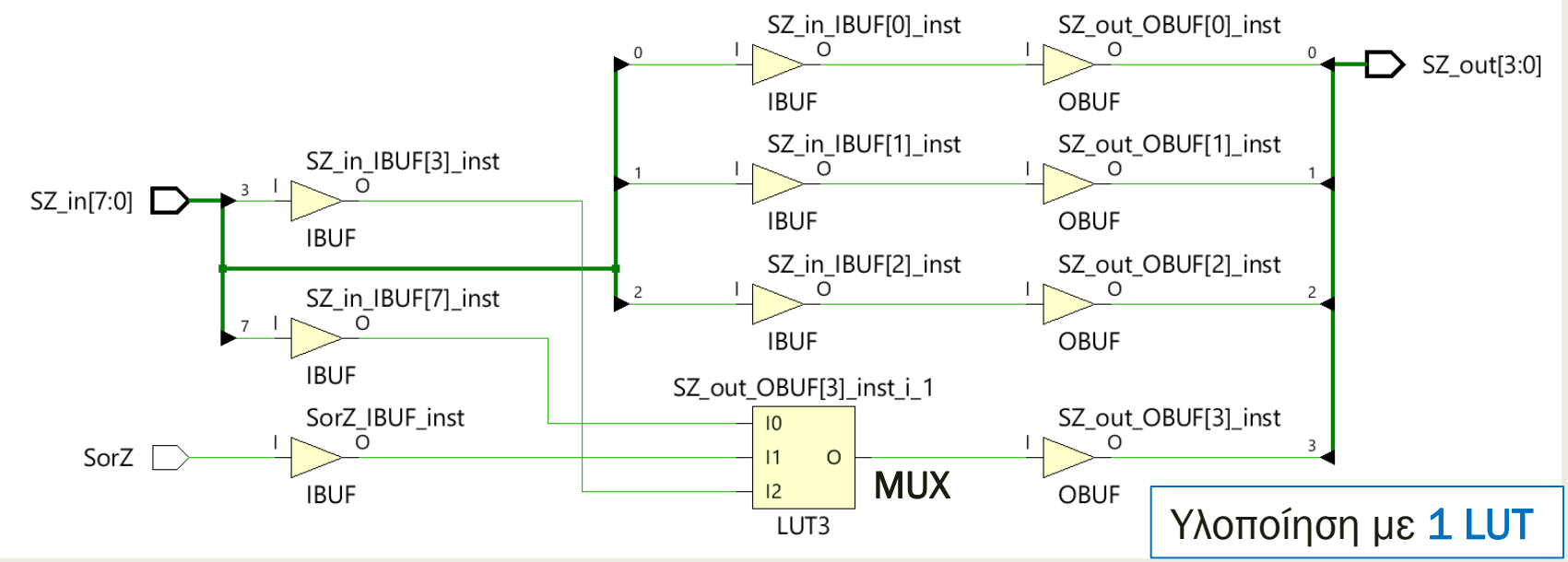
Μονάδα σμίκρυνσης πρόσημου/μηδενός στη VHDL – Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



WIDTH_IN > WIDTH_OUT

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Υλοποίηση με 1 LUT

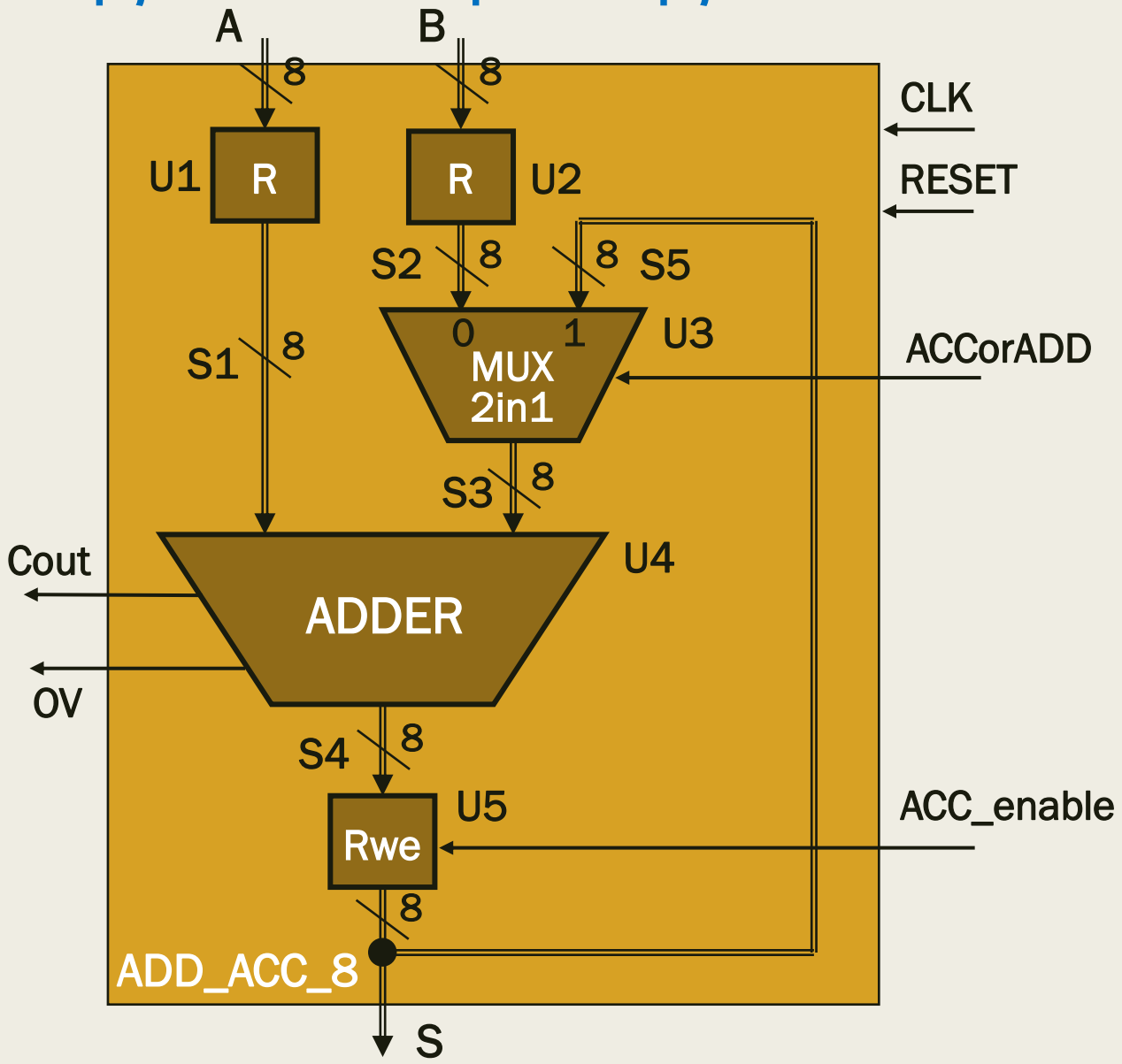
Εάν SorZ = 1 τότε SZ_out[3] = SZ_in[7], αλλιώς SZ_out[3] = SZ_in[3]

Ιεραρχική περιγραφή ψηφιακού συστήματος στο επίπεδο μεταφοράς καταχωρητή - RTL

- Για να επιτευχθεί πιο εύκολα η σύνθεση του ψηφιακού συστήματος συνήθως χρησιμοποιούνται περιγραφές επιπέδου μεταφοράς καταχωρητή (RTL description) για τη **διαδρομή δεδομένων** (datapath)
  - *περιγράφεται χωριστά κάθε καταχωρητής του συστήματος σε διακριτές οντότητες (entities)*
    - διαθέτουν εισόδους CLK, Reset και ενδεχομένως WE
  - *περιγράφεται χωριστά η συνδυαστική λογική ανάμεσα στους καταχωρητές*
  - *περιγράφεται χωριστά η **μονάδα ελέγχου** (control unit) της διαδρομής δεδομένων ως:*
    - συνδυαστική λογική για διαδρομές δεδομένων ενός κύκλου ή με διοχέτευση (pipelining)
    - ακολουθιακή λογική (μηχανή πεπερασμένων καταστάσεων, FSM) για διαδρομές δεδομένων πολλών κύκλων

Διαδρομή δεδομένων στη VHDL

Αθροιστής – Συσσωρευτής των 8 bit



Διαδρομή δεδομένων στη VHDL

Αθροιστής – Συσσωρευτής των 8 bit

- Παραμετροποιημένος καταχωρητής με Reset (R_n)

```
entity R_n is
  generic (WIDTH : positive := 4); -- προεπιλεγμένη τιμή
  port (
    CLK:   in  STD_LOGIC;
    RESET: in  STD_LOGIC;
    D_IN:  in  STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    D_OUT: out STD_LOGIC_VECTOR (WIDTH-1 downto 0));
end R_n;
architecture BEHAVIORAL of R_n is
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      if (RESET = '1') then D_OUT <= (others => '0');
      else D_OUT <= D_IN; end if;
    end if;
  end process;
end BEHAVIORAL;
```

Διαδρομή δεδομένων στη VHDL

Αθροιστής – Συσσωρευτής των 8 bit

- Παραμετροποιημένος καταχωρητής με Reset και WE των N bit (Rwe_n)

```
entity Rwe_n is
  generic (WIDTH : positive := 4); -- προεπιλεγμένη τιμή
  port (
    CLK:   in  STD_LOGIC;
    RESET: in  STD_LOGIC;
    WE:    in  STD_LOGIC;
    D_IN:  in  STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    D_OUT: out STD_LOGIC_VECTOR (WIDTH-1 downto 0));
end Rwe_n;
architecture BEHAVIORAL of Rwe_n is
begin
  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      if (RESET = '1') then D_OUT <= (others => '0');
      elsif (WE = '1') then D_OUT <= D_IN; end if;
    end if;
  end process;
end BEHAVIORAL;
```

Διαδρομή δεδομένων στη VHDL

Αθροιστής – Συσσωρευτής των 8 bit

- Παραμετροποιημένος πολυπλέκτης 2 σε 1 των N bit (MUX2in1_n)

```
entity MUX2in1_n is
  generic (WIDTH : positive := 4); -- 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;
```

Διαδρομή δεδομένων στη VHDL

Αθροιστής – Συσσωρευτής των 8 bit

- Παραμετροποιημένος αθροιστής με Cout και OV των N bit (ADDER_n)

```
entity ADDER_n is
  generic (WIDTH : positive := 4); -- προεπιλεγμένη τιμή
  port (
    A : in  STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    B : in  STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    S : out STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    Cout : out STD_LOGIC;
    OV :   out STD_LOGIC);
end ADDER_n;
architecture BEHAVIORAL of ADDER_n is
begin
  ADDER: process (A, B)
  variable A_s, B_s, S_s: SIGNED (WIDTH+1 downto 0);
  begin
    A_s := signed('0'&A(WIDTH-1)&A);              -- numeric_std
    B_s := signed('0'&B(WIDTH-1)&B);              -- numeric_std
    S_s := A_s + B_s;                             -- numeric_std
    S <= std_logic_vector(S_s(WIDTH-1 downto 0)); -- numeric_std
    OV <= S_s(WIDTH) xor S_s(WIDTH-1);
    Cout <= S_s(WIDTH+1);
  end process;
end BEHAVIORAL;
```

Διαδρομή δεδομένων στη VHDL

Αθροιστής – Συσσωρευτής των 8 bit

- Αθροιστής – Συσσωρευτής των 8 bit (ADD_ACC_8)

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

entity ADD_ACC_8 is
  generic (WIDTH : positive := 8); -- εδώ ορίζεται η τιμή
  port (
    CLK:        in STD_LOGIC;
    RESET:      in STD_LOGIC;
    ACC_enable: in STD_LOGIC;
    ACCorADD:   in STD_LOGIC;

    A:          in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    B:          in STD_LOGIC_VECTOR (WIDTH-1 downto 0);

    S:         out STD_LOGIC_VECTOR (WIDTH-1 downto 0);

    Cout:      out STD_LOGIC;
    OV:        out STD_LOGIC
  );
end ADD_ACC_8;
```

Διαδρομή δεδομένων στη VHDL

Αθροιστής – Συσσωρευτής των 8 bit

- Αθροιστής – Συσσωρευτής των 8 bit (ADD_ACC_8)

```
architecture STRUCTURAL of ADD_ACC_8 is
  signal S1, S2, S3, S4, S5: STD_LOGIC_VECTOR (WIDTH-1 downto 0);

  component R_n
  generic (WIDTH : positive := 4); -- προεπιλεγμένη τιμή
  port (
    CLK:   in  STD_LOGIC;
    RESET: in  STD_LOGIC;
    D_IN:  in  STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    D_OUT: out STD_LOGIC_VECTOR (WIDTH-1 downto 0));
  end component;

  component Rwe_n
  generic (WIDTH : positive := 4); -- προεπιλεγμένη τιμή
  port (
    CLK:   in  STD_LOGIC;
    RESET: in  STD_LOGIC;
    WE:    in  STD_LOGIC;
    D_IN:  in  STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    D_OUT: out STD_LOGIC_VECTOR (WIDTH-1 downto 0));
  end component;
```

Διαδρομή δεδομένων στη VHDL

Αθροιστής – Συσσωρευτής των 8 bit

- Αθροιστής – Συσσωρευτής των 8 bit (ADD_ACC_8)

```
component MUX2in1_n
generic (WIDTH : positive := 4); -- 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 component;

component ADDER_n
generic (WIDTH : positive := 4); -- προεπιλεγμένη τιμή
port (
  A : in  STD_LOGIC_VECTOR (WIDTH-1 downto 0);
  B : in  STD_LOGIC_VECTOR (WIDTH-1 downto 0);
  S : out STD_LOGIC_VECTOR (WIDTH-1 downto 0);
  Cout : out STD_LOGIC;
  OV :   out STD_LOGIC);
end component;
```

Διαδρομή δεδομένων στη VHDL

Αθροιστής – Συσσωρευτής των 8 bit

- Αθροιστής – Συσσωρευτής των 8 bit (ADD_ACC_8)

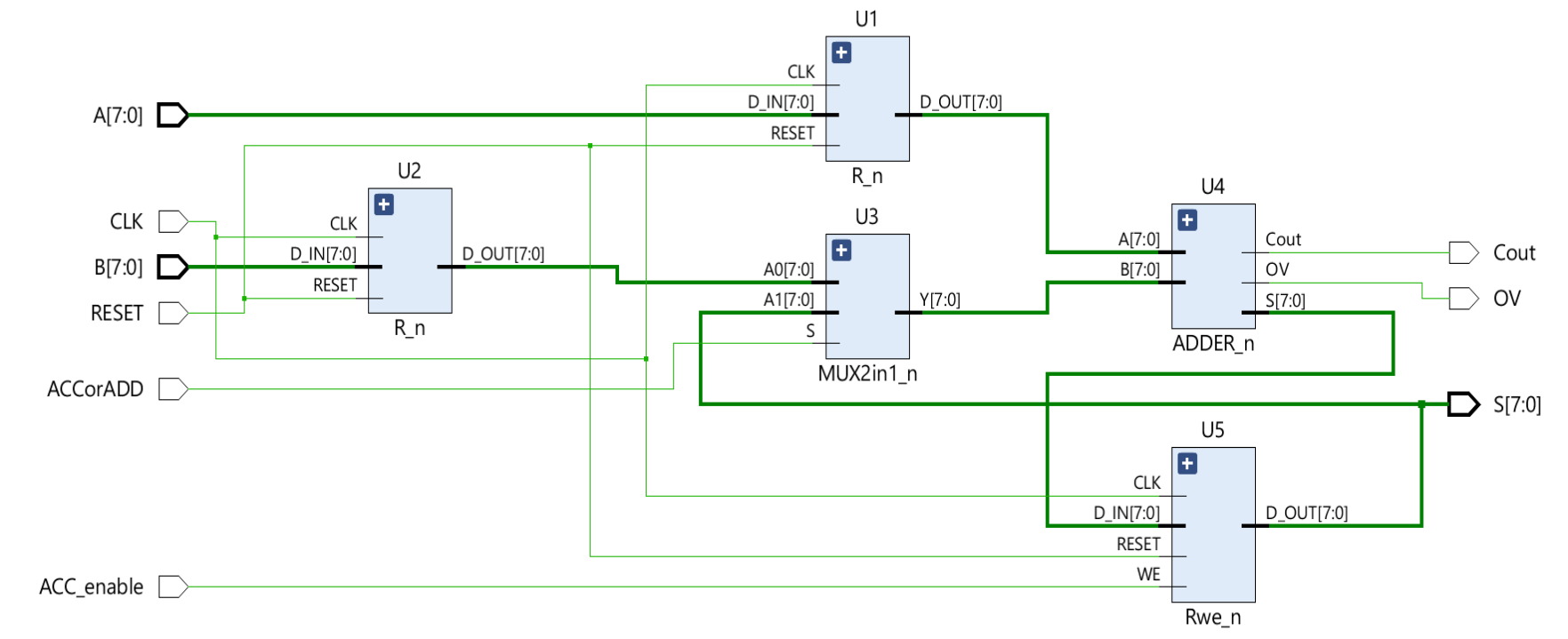
```
begin
  U1: R_n generic map (WIDTH => WIDTH) port map
    (CLK => CLK, RESET => RESET, D_IN => A, D_OUT => S1);
  U2: R_n generic map (WIDTH => WIDTH) port map
    (CLK => CLK, RESET => RESET, D_IN => B, D_OUT => S2);
  U3: MUX2in1_n generic map (WIDTH => WIDTH) port map
    (S => ACCorADD, A0 => S2, A1 => S5, Y => S3);
  U4: ADDER_n generic map (WIDTH => WIDTH) port map
    (A => S1, B => S3, S => S4, Cout => Cout, OV => OV);
  U5: Rwe_n generic map (WIDTH => WIDTH) port map
    (CLK => CLK, RESET => RESET, WE => ACC_enable,
     D_IN => S4, D_OUT => S5);

  S <= S5;
end STRUCTURAL;
```

Διαδρομή δεδομένων στη VHDL

Αθροιστής – Συσσωρευτής των 8 bit

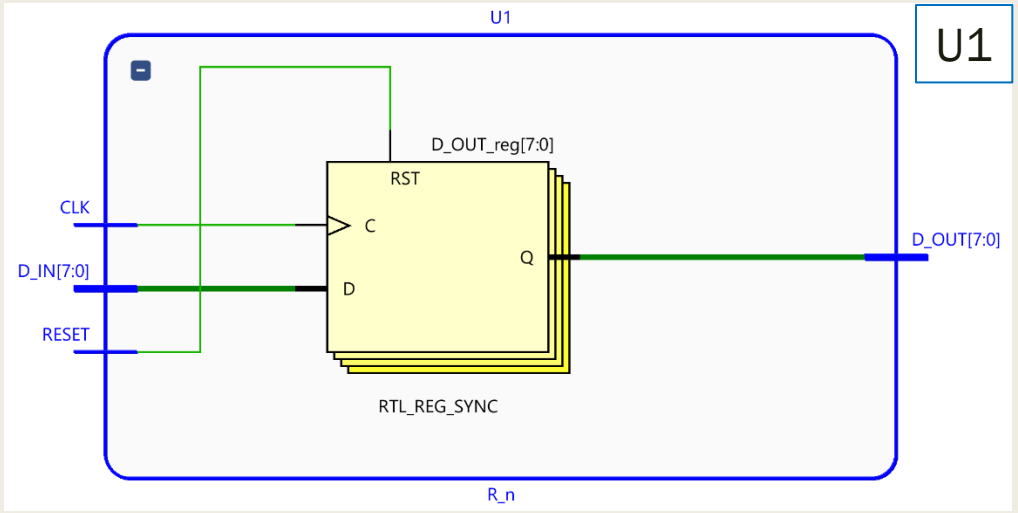
- Σχηματικό διάγραμμα RTL του Αθροιστή – Συσσωρευτή των 8 bit



Διαδρομή δεδομένων στη VHDL

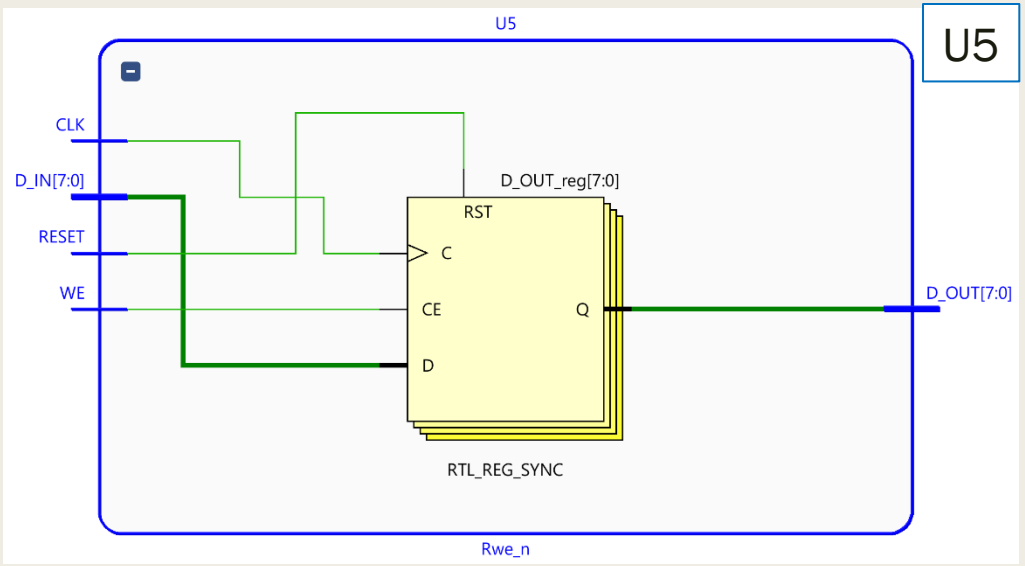
Αθροιστής – Συσσωρευτής των 8 bit

- Σχηματικό διάγραμμα RTL των καταχωρητών των 8 bit



U1

Ομοίως και το U2

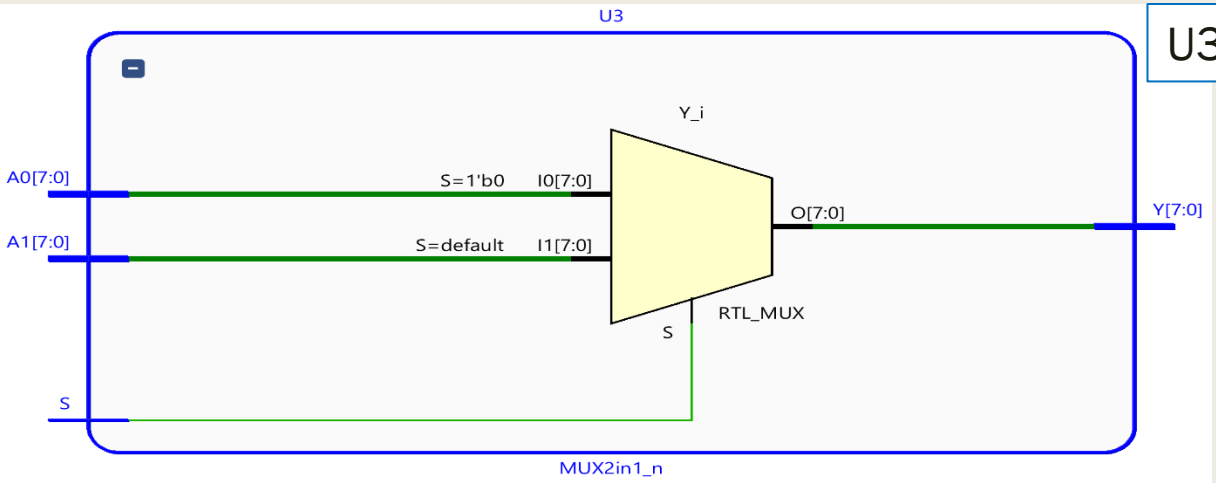


U5

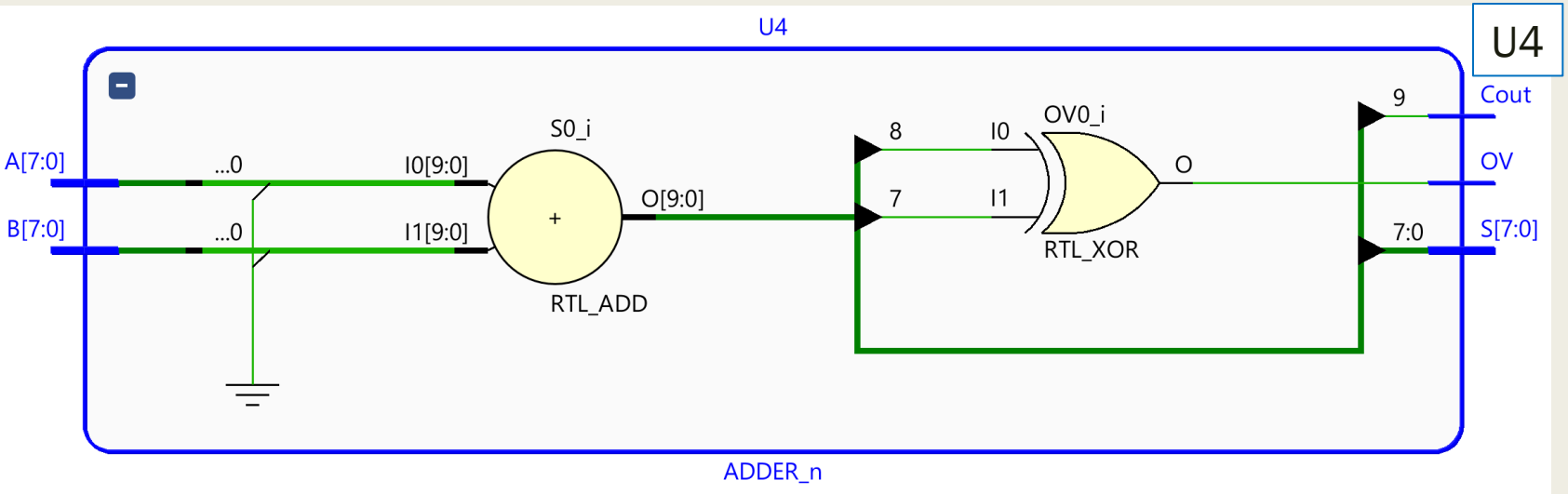
Διαδρομή δεδομένων στη VHDL

Αθροιστής – Συσσωρευτής των 8 bit

- Σχηματικό διάγραμμα RTL του πολυπλέκτη των 8 bit



- Σχηματικό διάγραμμα RTL του αθροιστή των 8 bit

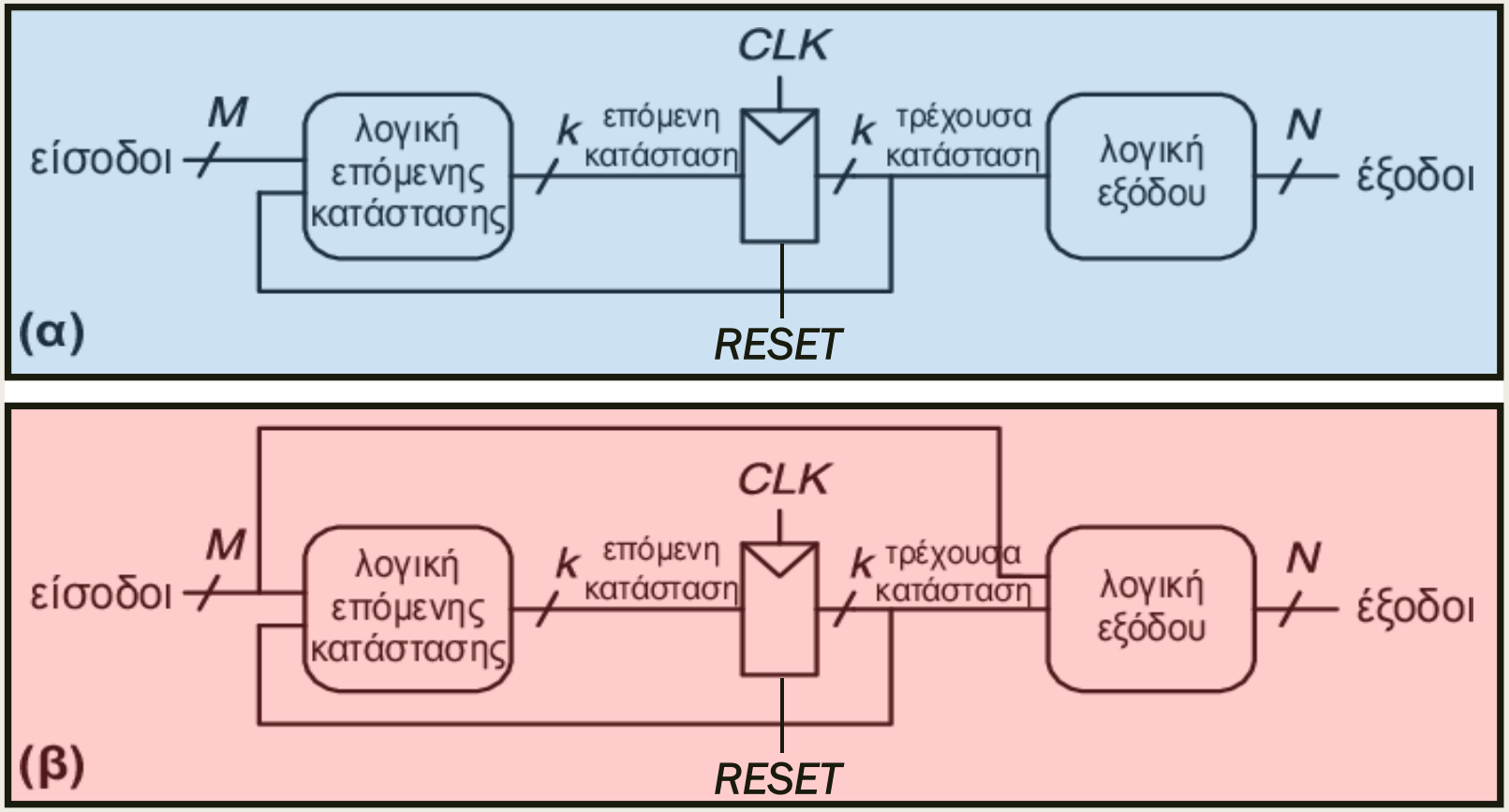


Μηχανές πεπερασμένων καταστάσεων

- Μία **μηχανη πεπερασμένων καταστάσεων** (finite state machines, FSM) είναι ένα σύγχρονο ακολουθιακό κύκλωμα που διαθέτει ***M* εισόδους**, ***N* εξόδους**, ένα σήμα ρολογιού ***CLK*** και ένα σήμα επαναφοράς ***RESET***, *κ*αι συνίσταται από:
  - *έναν **καταχωρητή καταστάσεων** (state register) μεγέθους k bit, που αποθηκεύει από **k μέχρι το πολύ 2^k διακριτές καταστάσεις***
  - *μία **λογική επόμενης κατάστασης (next state logic)** που υπολογίζει την επόμενη κατάσταση Q(t+1) ως συνάρτηση της τρέχουσας κατάστασης Q(t) και των M εισόδων*
  - *μία **λογική εξόδου (output logic)** που υπολογίζει τις τιμές των εξόδων ως συνάρτηση της τρέχουσας κατάστασης Q(t) και προαιρετικά των M εισόδων*
- Το σήμα επαναφοράς ***RESET*** στον καταχωρητή καταστάσεων είναι αναγκαίο ώστε να επιβάλλουμε μια **αρχική κατάσταση** στη μηχανή FSM, όταν την ενεργοποιούμε για πρώτη φορά
- Το προαιρετικό σήμα έγκρισης ***EN*** χρησιμοποιείται για να καθορίζει αν η μηχανή FSM αλλάξει κατάσταση κατά την επόμενη ακμή του CLK

Μηχανές πεπερασμένων καταστάσεων

- Υπάρχουν δύο τύποι μηχανών πεπερασμένης κατάστασης
  - *(α) Μηχανές τύπου Moore (οι έξοδοι εξαρτώνται μόνο από την παρούσα κατάσταση)*
  - *(β) Μηχανές τύπου Mealy (οι έξοδοι εξαρτώνται από την παρούσα κατάσταση **και** τις εισόδους)*



Μηχανές πεπερασμένων καταστάσεων

- Η μηχανή πεπερασμένων καταστάσεων τύπου Mealy είναι πιο γενική από τη μηχανή τύπου Moore
- Οι μηχανές πεπερασμένων καταστάσεων **τύπου Moore**
  - *έχουν **περισσότερες καταστάσεις***
  - *συνήθως πλεονεκτούν σε ταχύτητα και μέγεθος της **λογικής εξόδου***
- Οι μηχανές πεπερασμένων καταστάσεων **τύπου Mealy**
  - *έχουν **λιγότερες καταστάσεις***
  - *συνήθως πλεονεκτούν σε ταχύτητα και μέγεθος της **λογικής επόμενης κατάστασης***
- Μία μηχανή πεπερασμένων καταστάσεων μπορεί να έχει εξόδους και των δύο τύπων

Μηχανές πεπερασμένων καταστάσεων

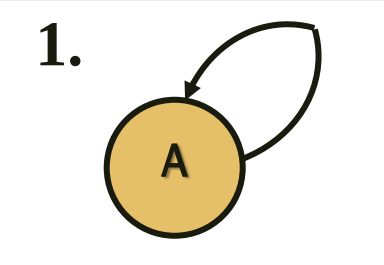
- Η απόδοση και το κόστος της μηχανής εξαρτάται από:
  - *Το **πλήθος των καταστάσεων** (περισσότερες στη μηχανή τύπου Moore)*
  - *Την **πολυπλοκότητα των διακλαδώσεων** ανά κατάσταση (μεγαλύτερη στη μηχανή τύπου Mealy)*
  - *Το **μέγεθος του καταχωρητή καταστάσεων***
  - *Το **πλήθος των ψηφίων που αλλάζουν τιμή** από την τρέχουσα κατάσταση στην επόμενη κατάσταση*
  - *Άρα, από την κωδικοποίηση των καταστάσεων*
    - **Δυαδική**
    - **Gray**
    - **Μονόθερμη (one-hot)**

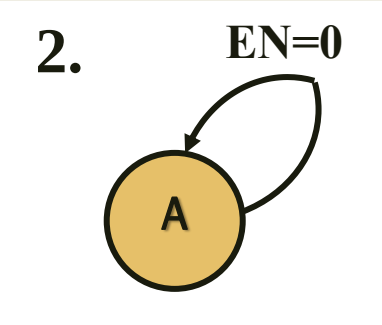
Διάγραμμα μεταβολής κατάστασης

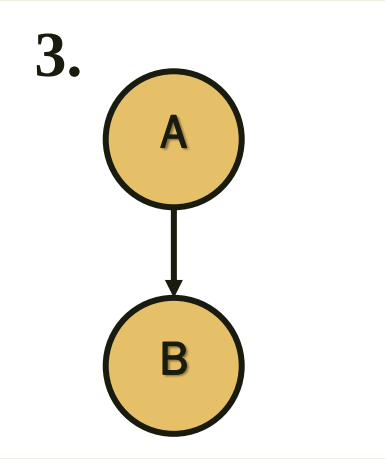
- Το πρώτο στάδιο στη σχεδίαση μίας μηχανής FSM είναι η σχεδίαση του **διαγράμματος μεταβολής κατάστασης** (state transition diagram)
  - *Στη σύγχρονη ψηφιακή σχεδίαση το διάγραμμα μεταβολής κατάστασης κωδικοποιείται σε μία γλώσσα περιγραφής υλικού (π.χ. VHDL) και όλα τα υπόλοιπα στάδια της σχεδίασης τα αναλαμβάνει το εργαλείο λογισμικού*
- Το διάγραμμα μεταβολής κατάστασης περιγράφει:
  - *τις **πιθανές διακριτές καταστάσεις** της μηχανής και πως αυτές μεταβάλλονται σύμφωνα με τις **συνθήκες εισόδου** κατά την ανερχόμενη ακμή του CLK*
  - *τις αντίστοιχες **εξόδους** τύπου Moore ή Mealy*
- Το διάγραμμα μεταβολής κατάστασης απαρτίζεται από:
  - ***Κύκλους** που προσδιορίζουν την τρέχουσα κατάσταση Q(t)*
  - ***Βέλη** που προσδιορίζουν τη μετάβαση από την τρέχουσα κατάσταση Q(t) στην επόμενη κατάσταση Q(t+1)*
    - Όταν υπάρχει μετάβαση με συνθήκη εισόδου, οι τιμές των εισόδων που ικανοποιούν τη συνθήκη γράφονται δίπλα στο βέλος

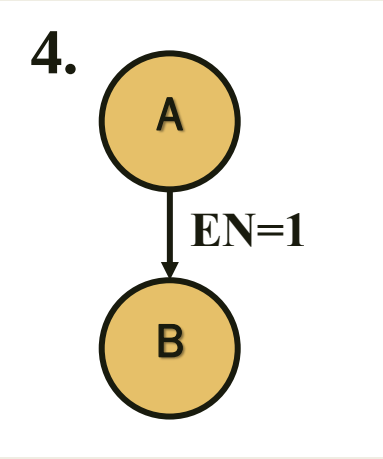
Διάγραμμα μεταβολής κατάστασης

- Παραδείγματα πιθανών **διαγραμμάτων μεταβολής κατάστασης**
  1. Όταν η επόμενη κατάσταση Α είναι ίδια με την τρέχουσα κατάσταση Α, χωρίς συνθήκη εισόδου
  2. Όταν η επόμενη κατάσταση Α είναι ίδια με την τρέχουσα κατάσταση Α με συνθήκη εισόδου (π.χ. EN=0)
  3. Όταν η επόμενη κατάσταση Β είναι διαφορετική από την τρέχουσα κατάσταση Α χωρίς συνθήκη εισόδου
  4. Όταν η επόμενη κατάσταση Β είναι διαφορετική από την τρέχουσα κατάσταση Α με συνθήκη εισόδου (π.χ. EN=1)



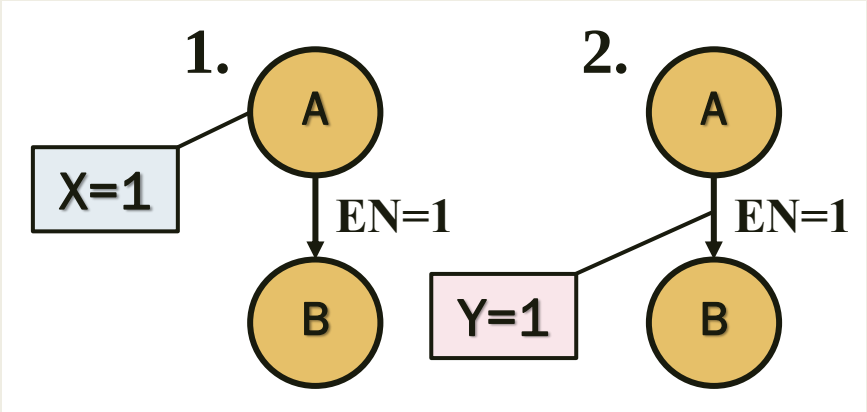






Διάγραμμα μεταβολής κατάστασης

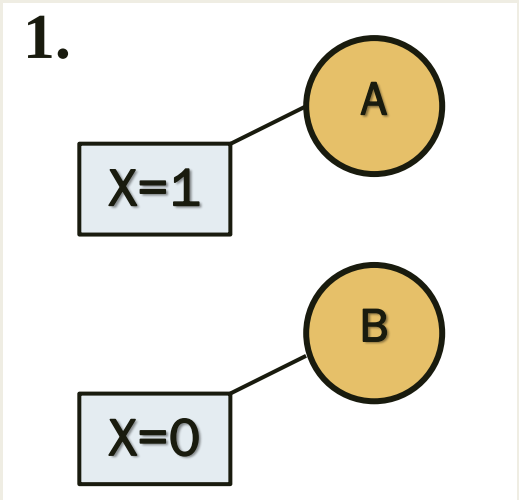
- Επιπλέον, το διάγραμμα μεταβολής κατάστασης απαρτίζεται από:
  - ***Πλαίσια*** *εντός των οποίων γράφονται οι τιμές των εξόδων της μηχανής FSM, που εξαρτώνται αποκλειστικά από την τρέχουσα κατάσταση (****έξοδοι τύπου Moore****)*
    - Τα πλαίσια αυτά συνδέονται με τον κύκλο της σχετικής τρέχουσας κατάστασης
  - ***Πλαίσια*** *εντός των οποίων γράφονται οι τιμές των εξόδων της μηχανής FSM, που εξαρτώνται από την τρέχουσα κατάσταση και την αντίστοιχη συνθήκη εισόδου (****έξοδοι τύπου Mealy****).*
    - Τα πλαίσια αυτά συνδέονται με το αντίστοιχο βέλος, δίπλα στις τιμές των εισόδων που ικανοποιούν τη συνθήκη
  - *Παραδείγματα χρήσης πλαισίων:*
    1. Η έξοδος X είναι 1 στην κατάσταση A (τύπου Moore)
    2. Η έξοδος Y είναι 1 στην κατάσταση A, όταν EN είναι 1 (EN=1) (τύπου Mealy)

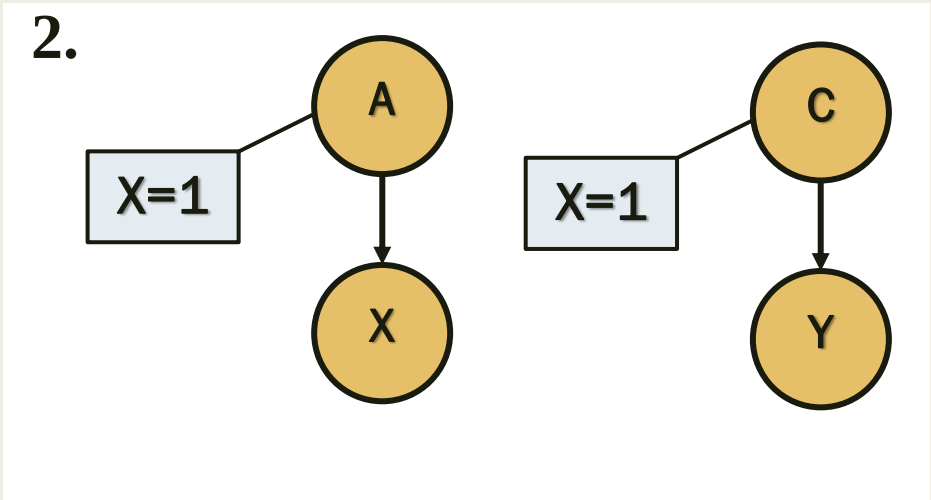


Η χρήση των πλαισίων διευκολύνει πολύ στην κατανόηση ιδίως όταν το πλήθος των σχετικών εξόδων είναι μεγάλο

Σχεδίαση μηχανών FSM στη VHDL

- Βήμα 1: Προσδιορίζουμε τις **εισόδους**, τις **εξόδους**, και τις **διακριτές καταστάσεις**
  - *Δύο τρέχουσες καταστάσεις χαρακτηρίζονται ως διακριτές μεταξύ τους, εάν :*
    - τουλάχιστον μία έξοδος, που εξαρτάται αποκλειστικά από την τρέχουσα κατάσταση, έχει διαφορετική τιμή (παράδειγμα 1), ή/και
    - έχουν τις ίδιες τιμές στις εξόδους, αλλά διαφορετική επόμενη κατάσταση που είναι ανεξάρτητη από τις εισόδους (παράδειγμα 2)
  - *Μη διακριτές καταστάσεις ενοποιούνται σε μία κατάσταση, ώστε να προκύψει ελαχιστοποίηση των καταστάσεων*





Σχεδίαση μηχανών FSM στη VHDL

- Βήμα 1 (συνέχεια): Προσδιορίζουμε τις **μεταβάσεις** ανάμεσα στις διακριτές καταστάσεις με τις αντίστοιχες **συνθήκες εισόδου**
- Προσδιορίζουμε τις τιμές των **εξόδων** που εξαρτώνται**:**
  - *αποκλειστικά από την **τρέχουσα κατάσταση** (**έξοδοι τύπου Moore**)*
  - *από την **τρέχουσα κατάσταση** και την αντίστοιχη **συνθήκη εισόδου** (**έξοδοι τύπου Mealy**)*
- Βήμα 2: Σχεδιάζουμε το **διάγραμμα μεταβολής κατάστασης**
- Βήμα 3: Κωδικοποιούμε το **διάγραμμα μεταβολής κατάστασης** στη VHDL
  - *Επιλέγουμε την **κωδικοποίηση των καταστάσεων** ή αφήνουμε το εργαλείο λογισμικού να την επιλέξει*

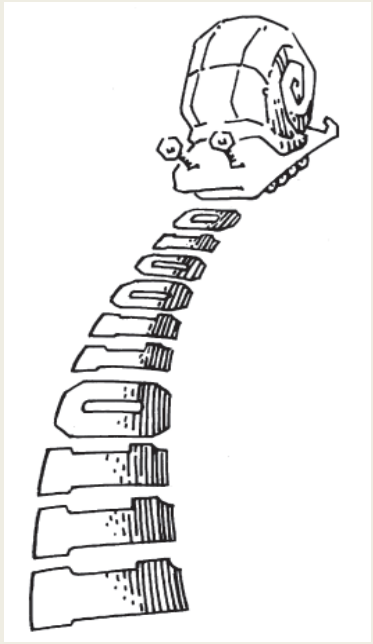
Κωδικοποίηση των καταστάσεων

- Στην πράξη συνήθως χρησιμοποιούνται οι ακόλουθες κωδικοποιήσεις:
- **Binary (δυαδική)**
  - *για δυαδικούς μετρητές, όπου καταστάσεις και έξοδοι ταυτίζονται*
- **Μοναδικού σημαντικού (one-hot) – ένα ξεχωριστό bit ανά κατάσταση**
  - *για υλοποιήσεις σε FPGA και πλήθος καταστάσεων από 10 μέχρι 30*
  - *ελαχιστοποιεί τις εξισώσεις Boole, αλλά αυξάνει το μέγεθος του καταχωρητή*
- **Gray ή τροποποιημένη Gray**
  - *Η πιο διαδεδομένη γιατί συνδυάζει το μικρότερο δυνατό μέγεθος του καταχωρητή καταστάσεων με αρχική τιμή στο όλα-0 και το ελάχιστο πλήθος των ψηφίων που αλλάζουν τιμή από κατάσταση σε κατάσταση*
    - Μόνο ένα ψηφίο στις περισσότερες περιπτώσεις

6 καταστάσεις	One-Hot	Binary	τροπ. Gray
A	000001	000	000
B	000010	001	001
C	000100	010	011
D	001000	011	010
E	010000	100	110
F	100000	101	100 (111)

Σχεδίαση μηχανών FSM στη VHDL

- Έστω ότι έχουμε ένα σαλιγκάρι-ρομπότ ως κατοικίδιο, το οποίο διαθέτει μια μηχανή FSM για εγκέφαλο
  - *Το σαλιγκάρι έρπει κατά μήκος μιας χάρτινης ταινίας που περιέχει μια ακολουθία από 0 και 1*
  - *Σε κάθε κύκλο του ρολογιού, το σαλιγκάρι έρπει έως το επόμενο bit της ακολουθίας*
  - *Το σαλιγκάρι χαμογελάει όταν τα δύο τελευταία **διαδοχικά bit** πάνω από τα οποία έχει περάσει είναι **01***



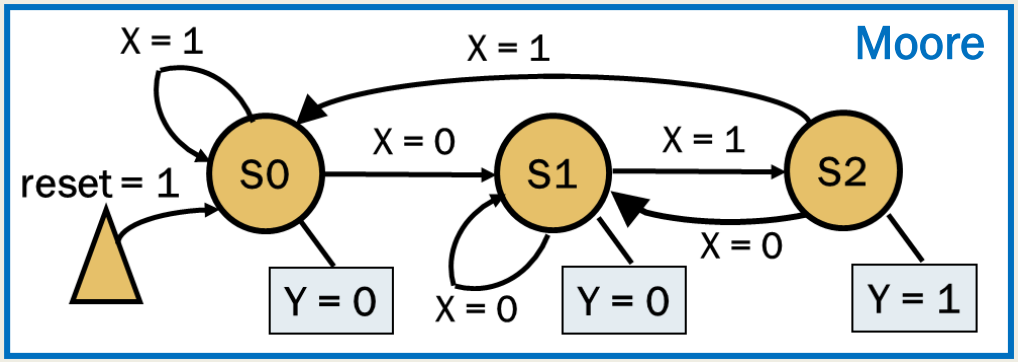
- Σχεδιάστε τη μηχανή FSM στη VHDL έτσι ώστε να υπολογίζει πότε το σαλιγκάρι πρέπει να χαμογελάει
  - *Η μηχανή FSM του σαλιγκαριού-ρομπότ είναι ένας **ανιχνευτής ακολουθίας 2 διαδοχικών bit** που μεταδίδονται σειριακά στην είσοδο X*
  - *Το σαλιγκάρι να χαμογελάει όταν η έξοδος Y γίνεται 1*
- Συγκρίνετε τα **σχηματικά διαγράμματα** των μηχανών FSM **τύπων Moore** και **Mealy**
- Δώστε τις **προσομοιώσεις** για κάθε τύπο της μηχανής FSM
  - *στο οποίο θα φαίνονται η είσοδος, οι καταστάσεις, και η έξοδος καθώς το σαλιγκάρι έρπει κατά μήκος της ακολουθίας **0100110111***

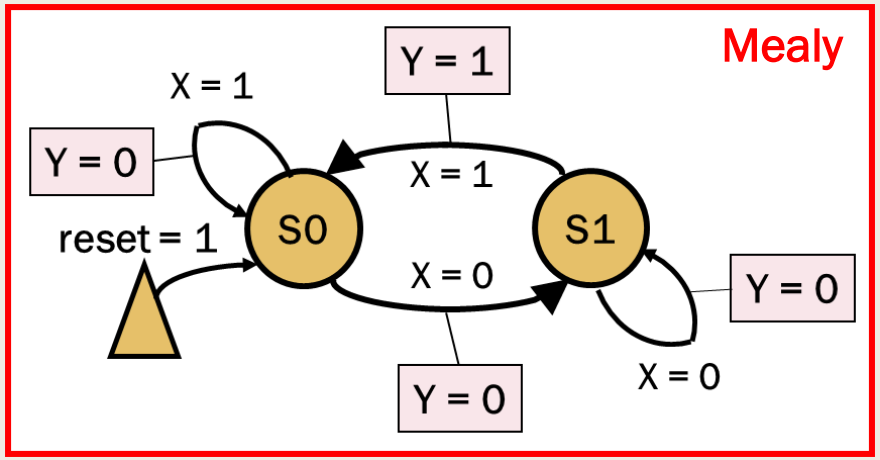
Σχεδίαση μηχανών FSM στη VHDL

- **Βήμα 1: Προσδιορίζουμε τις εισόδους, τις εξόδους, και τις καταστάσεις**
  - *Ο ανιχνευτής έχει μία σειριακή είσοδο X και μία έξοδο Y*
    - Y = 1, όταν τα δύο τελευταία διαδοχικά bit της εισόδου X είναι 01
  - *Καταστάσεις μηχανής **τύπου Moore**:*
    - S0 = αρχική κατάσταση δεν έχει ανιχνευθεί κανένα ψηφίο, Y = 0
    - S1 = έχει ανιχνευθεί στην είσοδο X ένα bit 0, Y = 0
    - S2 = έχουν ανιχνευθεί στην είσοδο X δύο διαδοχικά bit 01, Y = 1
  - *Καταστάσεις μηχανής **τύπου Mealy**:*
    - S0 = αρχική κατάσταση δεν έχει ανιχνευθεί κανένα ψηφίο
      - *εάν X = 1, τότε Y = 0 και παραμένει στην S0*
      - *εάν X = 0, τότε Y = 0 και πηγαίνει στην S1 (ανίχνευση 0)*
    - S1 = έχει ανιχνευθεί στην είσοδο X ένα bit 0
      - *εάν X = 1, τότε Y = 1 και πηγαίνει στην S0 (ανίχνευση 01)*
      - *εάν X = 0, τότε Y = 0 και παραμένει στην S1 (ανίχνευση 0)*
    - Δεν χρειάζεται η κατάσταση S2 για τον προσδιορισμό του Y = 1
      - *Η έξοδος Y = 1 προσδιορίζεται από την είσοδο X = 1, όταν βρίσκεται στην κατάσταση S1*

Σχεδίαση μηχανών FSM στη VHDL

- **Βήμα 2: Σχεδιάζουμε τα διαγράμματα μεταβολής κατάστασης**
  - *Ο μετρητής τύπου Moore έχει **3 καταστάσεις S0, S1 και S2***
  - *Ο μετρητής τύπου Mealy έχει **2 καταστάσεις S0 και S1***
  - *Το σήμα **RESET** αρχικοποιεί τη μηχανή FSM στην κατάσταση **S0***
    - Η αρχικοποίηση γίνεται σύγχρονα ή ασύγχρονα του CLK με κατάλληλη επιλογή των D Flip-Flop





Σχεδίαση μηχανών FSM στη VHDL

- **Κωδικοποιούμε το διάγραμμα μεταβολής κατάστασης στη VHDL**
  - *Αρχικά περιγράφουμε την **οντότητα** της μηχανής FSM προσδιορίζοντας:*
    - τις εισόδους CLK, RESET
    - τις εισόδους του FSM: X
      - *οι είσοδοι του FSM αποθηκεύονται σε καταχωρητή εισόδων για να επιτευχθεί η σύγχρονη λειτουργία όλων των μονάδων*
      - *το σήμα **RESET** αρχικοποιεί και τον καταχωρητή εισόδων*
    - τις εξόδους του FSM: Y
      - *στην κατάσταση **S0** η έξοδος Y είναι **πάντα ανενεργή** ανεξάρτητα της τιμής της εισόδου*

```
entity PATTERN_FSM is
  port (
    CLK:    in STD_LOGIC;
    RESET:  in STD_LOGIC;
-- FSM inputs
    X:      in STD_LOGIC;
-- FSM outputs
    Y:     out STD_LOGIC);
end PATERN_FSM;
```

Σχεδίαση μηχανών FSM στη VHDL
Περιγραφή συμπεριφοράς

- Κωδικοποιούμε το διάγραμμα μεταβολής κατάστασης στη VHDL
  - *Στη συνέχεια περιγράφουμε την αρχιτεκτονική της μηχανής FSM προσδιορίζοντας:*
    - τις καταστάσεις FSM_states με τη χρήση ενός τύπου απαρίθμησης (enumeration type)
      - *αφήνουμε το εργαλείο σύνθεσης να αποφασίσει για την κωδικοποίηση*
    - τα εσωτερικά σήματα (signals) του FSM
      - *current_state και next_state*

```
architecture BEHAVIORAL_MOORE of PATTERN_FSM is
-- state definition
  type FSM_states is
  (S0, S1, S2);
-- internal signals
  signal current_state, next_state: FSM_states;
```

```
architecture BEHAVIORAL_MEALY of PATTERN_FSM is
-- state definition
  type FSM_states is
  (S0, S1); -- Λιγότερες καταστάσεις
-- internal signals
  signal current_state, next_state: FSM_states;
```

Σχεδίαση μηχανών FSM στη VHDL
Περιγραφή συμπεριφοράς

- Κωδικοποιούμε το διάγραμμα μεταβολής κατάστασης στη VHDL
  - *Εάν επιθυμούμε να επιλέξουμε την κωδικοποίηση των καταστάσεων*
    - ορίζουμε έναν υποτύπο **subtype** του FSM_states με τη χρήση σταθεράς **constant**, ως εξής:

```
architecture BEHAVIORAL_MOORE of PATTERN_FSM is
-- state definition
--  type FSM_states is
--  (S0, S1, S2);
-- state encoding in one hot code by the user
  subtype FSM_states is STD_LOGIC_VECTOR (2 downto 0);

  constant S0  : FSM_states := "001";
  constant S1  : FSM_states := "010";
  constant S2  : FSM_states := "100";

-- internal signals
  signal current_state, next_state : FSM_states;

  signal X_in : STD_LOGIC; -- Only when there is an INREG
```

Το X_in απαιτείται μόνο όταν ενσωματώνουμε στον ανιχνευτή τον καταχωρητή εισόδων

Σχεδίαση μηχανών FSM στη VHDL
Περιγραφή συμπεριφοράς

- Κωδικοποιούμε το διάγραμμα μεταβολής κατάστασης στη VHDL
  - *Στη συνέχεια περιγράφουμε την αρχιτεκτονική της μηχανής FSM προσδιορίζοντας:*
    - τον καταχωρητή εισόδων (input register) με τη χρήση της σύγχρονης διεργασίας (process) INREG που εξαρτάται από τα σήματα CLK και RESET
      - *είναι προαιρετική διαδικασία*

```
begin

-- Optional for sychronization
  INREG:  process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      if (RESET = '1') then X_in <= '0';
      else X_in <= X;
      end if;
    end if;
  end process;
```

Ο καταχωρητής εισόδων πηγαίνει σύγχρονα στην κατάσταση 0

Σχεδίαση μηχανών FSM στη VHDL
Περιγραφή συμπεριφοράς

- Κωδικοποιούμε το διάγραμμα μεταβολής κατάστασης στη VHDL
  - *Στη συνέχεια περιγράφουμε την αρχιτεκτονική της μηχανής FSM προσδιορίζοντας:*
    - τον καταχωρητή καταστάσεων (state register) με τη χρήση της σύγχρονης διεργασίας (process) SYNC που εξαρτάται από τα σήματα CLK και RESET
      - *είναι κοινή για όλα τα FSM*

```
-- 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;
```

Ο καταχωρητής καταστάσεων πηγαίνει σύγχρονα στην κατάσταση S0

Σχεδίαση μηχανών FSM στη VHDL
Περιγραφή συμπεριφοράς

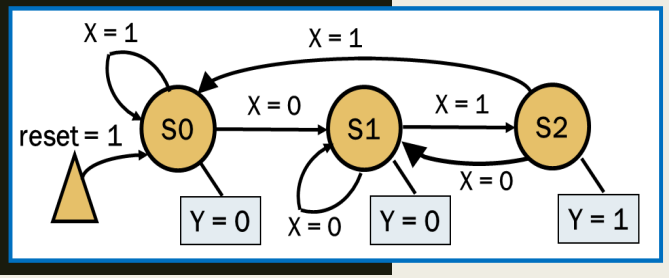
- **Κωδικοποιούμε το διάγραμμα μεταβολής κατάστασης στη VHDL**
  - *Τέλος, περιγράφουμε την αρχιτεκτονική της μηχανής FSM προσδιορίζοντας:*
    - τη **συνδυαστική λογική** που παράγει την **επόμενη κατάσταση** της μηχανής FSM, σαν συνάρτηση της τρέχουσας κατάστασης και των εισόδων της μηχανής του FSM με τη χρήση μίας **ασύγχρονης διεργασίας** (process)
    - τη **συνδυαστική λογική** που παράγει τις **εξόδους** του μονάδας FSM σαν συνάρτηση της τρέχουσας κατάστασης και των εισόδων του FSM (μόνο στις μηχανές τύπου Mealy) με τη χρήση μίας **ασύγχρονης διεργασίας** (process)
  - *Μπορούμε να ενοποιήσουμε τις 2 ασύγχρονες διεργασίες σε ένα process (υλοποίηση μηχανής FSM με 2 process – SYNC & ASYNC)*
    - στην αρχή της διεργασίας **ASYNC** βάζουμε αρχικές τιμές στην επόμενη κατάσταση και σε όλα τα σήματα εξόδου για να αποφύγουμε την **ελλιπή ανάθεση τιμών**
    - η περιγραφή συμπεριφοράς της διεργασίας **ASYNC** βασίζεται στο **διάγραμμα μεταβολής κατάστασης**
    - εξασφαλίζουμε την **ασφαλή συμπεριφορά** της μηχανής FSM
      - *σε περίπτωση βλάβης κλειδώνει στην κατάσταση S0*

Σχεδίαση μηχανών FSM στη VHDL

Περιγραφή συμπεριφοράς

- Κωδικοποιούμε το διάγραμμα μεταβολής κατάστασης στη VHDL
  - *Η ασύγχρονη διεργασία ASYNC που βασίζεται στο διάγραμμα μεταβολής κατάστασης*

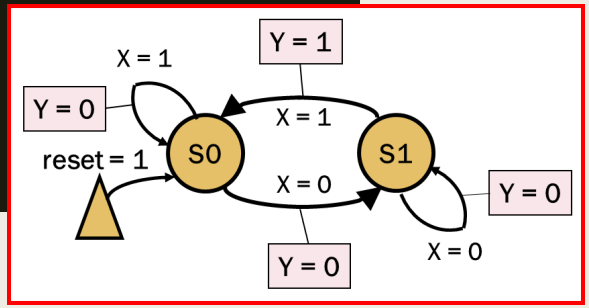
```
-- Process to create next state logic and output logic
  ASYNC: process (current_state, X_in)   -- Moore
  begin
  -- FSM next state and output initialization
    next_state <= S0;
    Y <= '0';
    case current_state is
      when S0 =>
        if (X_in = '0') then next_state <= S1;
        else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



Σχεδίαση μηχανών FSM στη VHDL
Περιγραφή συμπεριφοράς

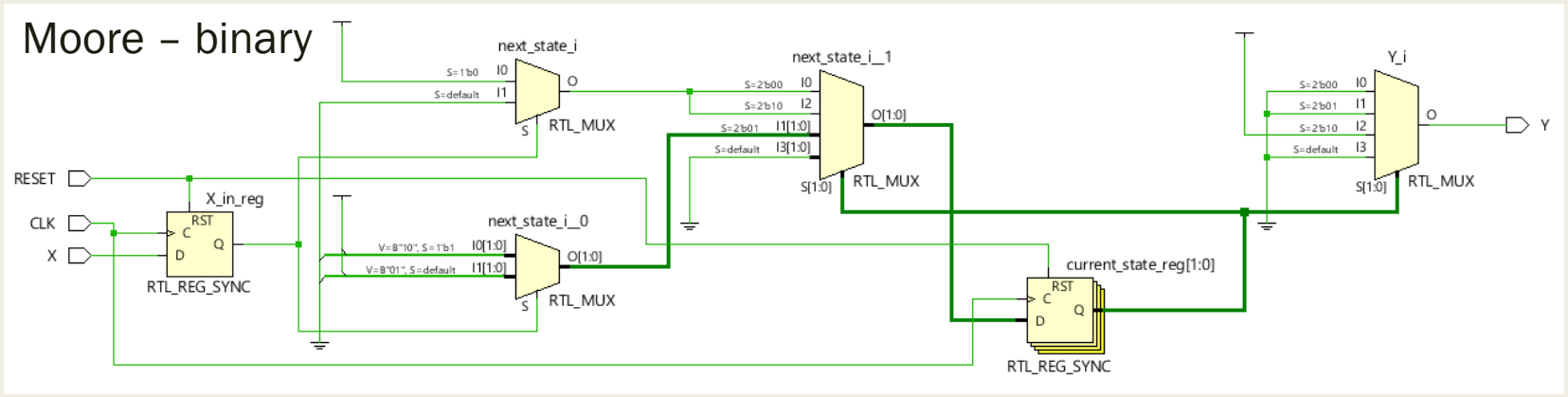
- Κωδικοποιούμε το διάγραμμα μεταβολής κατάστασης στη VHDL
  - *Η ασύγχρονη διεργασία ASYNC που βασίζεται στο διάγραμμα μεταβολής κατάστασης*

```
-- Process to create next state logic and output logic
  ASYNC: process (current_state, X_in)   -- Mealy
  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 Y <= '1'; next_state <= S0;
        else next_state <= S1; end if;
      -- fail-safe behavior
      when others => next_state <= S0;
    end case;
  end process;
end BEHAVIORAL;
```



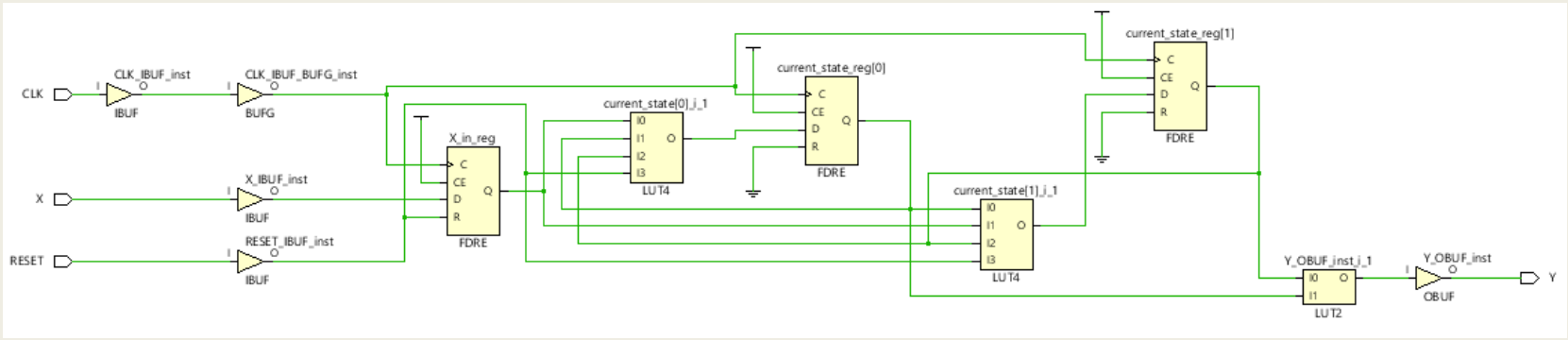
Σχεδίαση μηχανών FSM στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL του ανιχνευτή ακολουθίας 2 διαδοχικών bit



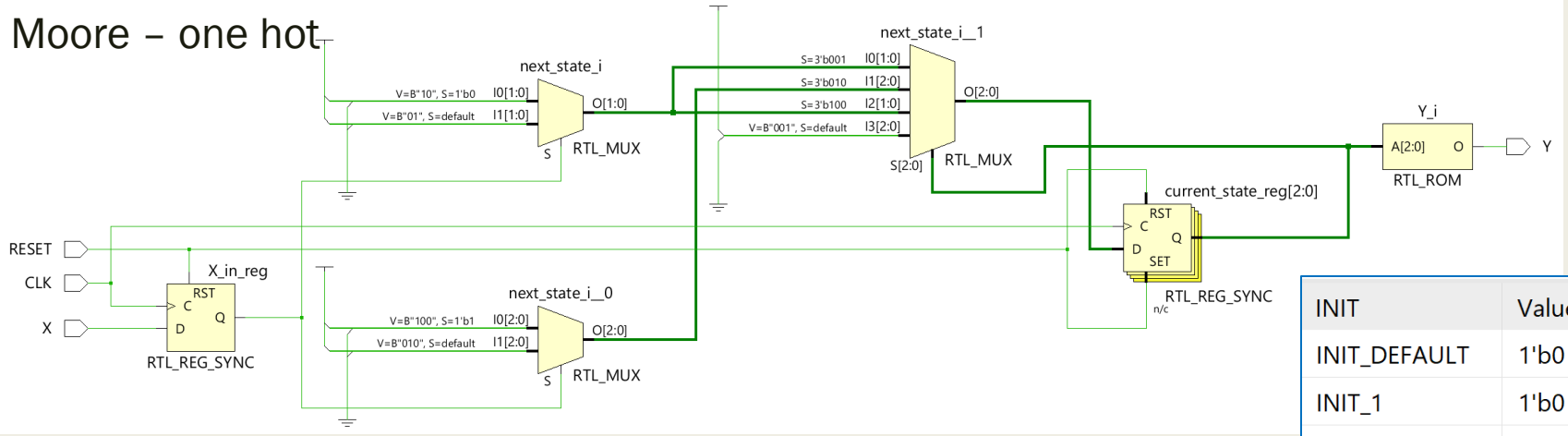
Αυτόματη κωδικοποίηση στις καταστάσεις S0 = 00, S1 = 01, S2 = 10

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Σχεδίαση μηχανών FSM στη VHDL
Περιγραφή συμπεριφοράς

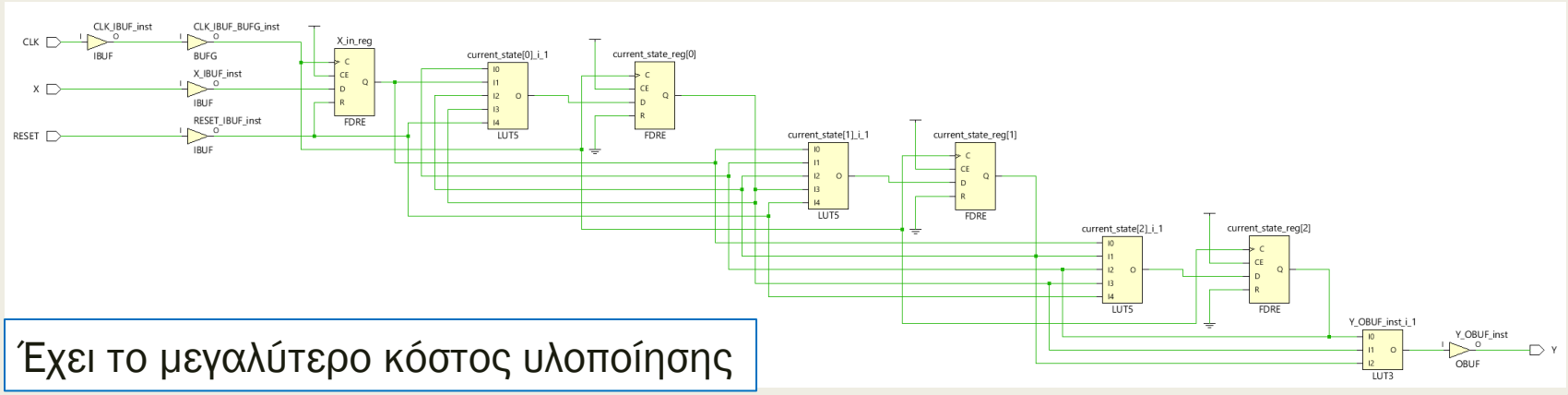
- Σχηματικό διάγραμμα RTL του ανιχνευτή ακολουθίας 2 διαδοχικών bit



INIT	Value
INIT_DEFAULT	1'b0
INIT_1	1'b0
INIT_2	1'b0
INIT_4	1'b1

Κωδικοποίηση στις καταστάσεις S0 = 001, S1 = 010, S2 = 100

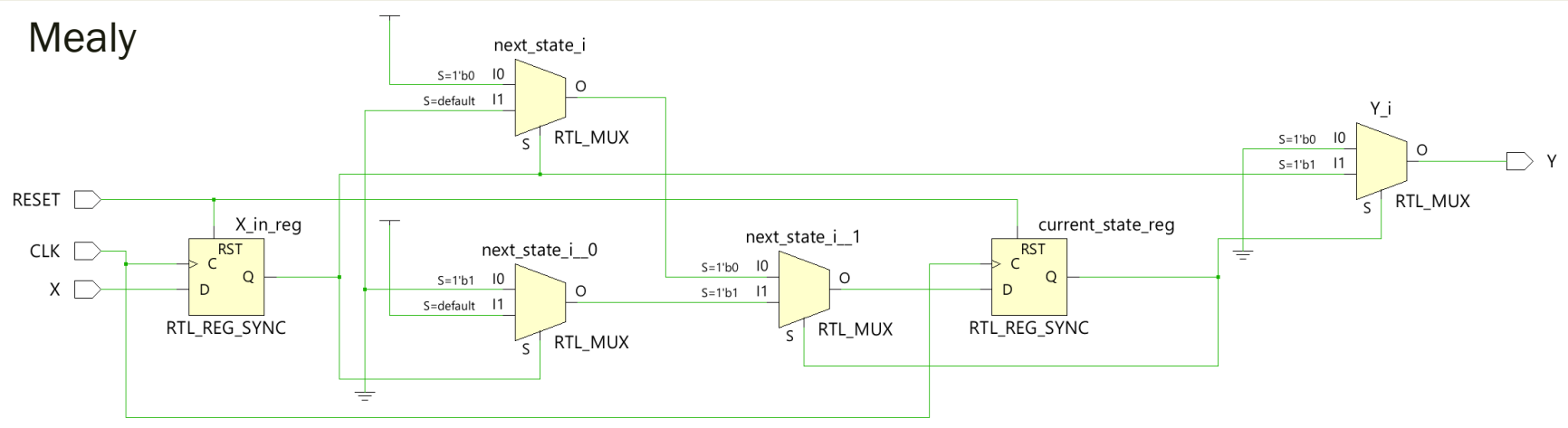
- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Έχει το μεγαλύτερο κόστος υλοποίησης

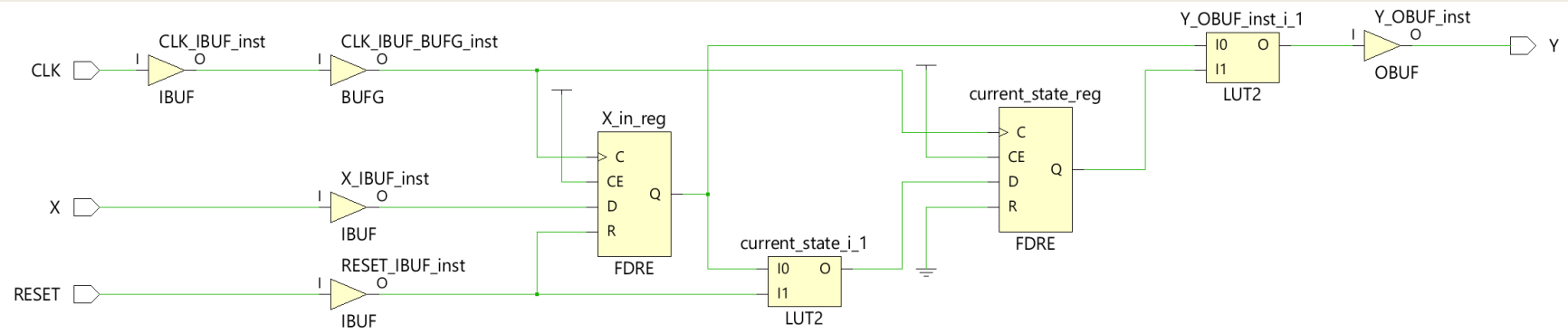
Σχεδίαση μηχανών FSM στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL του ανιχνευτή ακολουθίας 2 διαδοχικών bit



Κωδικοποίηση στις καταστάσεις S0 = 0, S1 = 1

- Σχηματικό διάγραμμα σε τεχνολογία FPGA



Έχει το μικρότερο κόστος υλοποίησης

Προσομοίωση στο VIVADO της XILINX

- **Behavioral simulation**
  - *Εφαρμόζεται στο **behavioral (elaborated design) model** που προκύπτει μετά την ανάλυση στο επίπεδο RTL του κώδικα VHDL*
  - *Λογική προσομοίωση*
- **Post synthesis functional simulation**
  - *Εφαρμόζεται στο **synthesized design model** που προκύπτει μετά τη σύνθεση του κώδικα VHDL*
  - *Λογική προσομοίωση (πρέπει να ταυτίζεται με το **behavioral simulation**)*
- **Post synthesis timing simulation**
  - *Εφαρμόζεται στο **synthesized design model** που προκύπτει μετά τη σύνθεση του κώδικα VHDL*
  - *Χρονική προσομοίωση*
- **Post implementation functional simulation**
  - *Εφαρμόζεται στο **implemented design model** που προκύπτει μετά την υλοποίηση σε συγκεκριμένη τεχνολογία FPGA του **synthesized design model***
  - *Λογική προσομοίωση (πρέπει να ταυτίζεται με το **behavioral simulation**)*
- **Post synthesis timing simulation**
  - *Εφαρμόζεται στο **implemented design model** που προκύπτει μετά την υλοποίηση σε συγκεκριμένη τεχνολογία FPGA του **synthesized design model***
  - *Χρονική προσομοίωση*

Περισσότερα στο UG900: Vivado Design Suite User Guide: Logic Simulation

Προγράμματα δοκιμών στη VHDL
Περιγραφή συμπεριφοράς

- Το **πρόγραμμα δοκιμών** (testbench) είναι μια οντότητα που χρησιμοποιείται για την πραγματοποίηση δοκιμών σε μία άλλη οντότητα, που ορίζεται ως **μονάδα υπό δοκιμή** (unit under test, UUT)
  - *περιέχει εντολές που όταν εκτελούνται κατά τη διάρκεια της **προσομοίωσης** παράγουν κατάλληλους συνδυασμούς τιμών των εισόδων της μονάδας UUT*
  - *μας επιτρέπει να ελέγξουμε αν η μονάδα UUT παράγει τις σωστές εξόδους σύμφωνα με τις προδιαγραφές*
- Οι ακολουθίες (ή μοτίβα) εισόδου με τις αντίστοιχες σωστές ακολουθίες εξόδου ονομάζονται **διανύσματα δοκιμής** (test vectors)
- Ο κώδικας VHDL του προγράμματος δοκιμών **δεν είναι συνθέσιμος**
  - *Χρησιμοποιεί την εντολή **WAIT***

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Η εντολή WAIT

- Η εντολή **WAIT** χρησιμοποιείται σε **διεργασίες** που δεν έχουν λίστα ευαισθησίας ως πρώτη εντολή μετά το process με δύο τρόπους:
  - *Η εντολή **WAIT UNTIL condition** εξετάζει μία συνθήκη που αφορά **μόνο ένα σήμα** και δεν επιτρέπει την εκτέλεση της διεργασίας μέχρι να **ικανοποιηθεί η συνθήκη***
  - *Η εντολή **WAIT ON list of signal** εξετάζει την τιμή **μίας λίστας σημάτων** και δεν επιτρέπει την εκτέλεση της διεργασίας μέχρι να αλλάξει η τιμή ενός από τα σήματα της λίστας*
    - αντικαθιστά τη λίστα ευαισθησίας στην αρχή ενός process

```
wait until single_signal_codition;
wait on signal_1, ..., signal_n;
```

Οι εντολές **WAIT UNTIL** και **WAIT ON** είναι **συνθέσιμες**

Περιγραφή συμπεριφοράς (behavioral) στη VHDL – Η εντολή WAIT

- Η εντολή **WAIT** σημαίνει «περίμενε μη κάνεις τίποτα»
- Η εντολή **WAIT FOR N ns** χρησιμοποιείται σε διεργασίες που δεν έχουν λίστα ευαισθησίας και «παγώνει» για N ns την τιμή που έχουν πάρει τα σήματα με μία εντολή ανάθεσης τιμής
  - *έπεται των εντολών ανάθεσης τιμών σε σήματα*
- Όταν εμφανίζεται μόνο το WAIT «παγώνει» η προσομοίωση γιατί δεν αλλάζουν πλέον οι τιμές των σημάτων που περιγράφονται μέσα σε ένα πρόγραμμα δοκιμών στη VHDL

```
wait;
wait for N ns;
```

Οι εντολές **WAIT** και **WAIT FOR δεν** είναι **συνθέσιμες**

Προγράμματα δοκιμών στη VHDL
Περιγραφή συμπεριφοράς

- **Κωδικοποιούμε το πρόγραμμα δοκιμών στη VHDL**
  - *Αρχικά, περιγράφουμε την **οντότητα Name_TB** που χρησιμοποιείται για την πραγματοποίηση δοκιμών στη **μονάδα Name** που τίθεται **υπό δοκιμή** (unit under test - UUT)*
  - *η οντότητα Name_TB **δεν έχει εισόδους ή εξόδους***

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE std.env.all;

entity Name_TB is
end Name_TB;
```

Προσοχή! Για να εκτελεσθεί η προσομοίωση πρέπει η οντότητα Name_TB να έχει δηλωθεί στα simulation sources του VIVADO ως κορυφαία με το set as top

Προσοχή! Για να εκτελεσθεί ορθά η προσομοίωση των synthesized και implemented design models πρέπει η οντότητα Name να έχει δηλωθεί στα design sources του VIVADO ως κορυφαία με το set as top και αν έχουν εκτελεσθεί πλήρως οι φάσεις RTL analysis, synthesis και implementation

Προγράμματα δοκιμών στη VHDL
Περιγραφή συμπεριφοράς

- **Κωδικοποιούμε το πρόγραμμα δοκιμών στη VHDL**
  - *Στη συνέχεια, περιγράφουμε τις δηλώσεις της αρχιτεκτονικής της οντότητας Name_TB*
    - στην αρχή δηλώνουμε τη μονάδα Name ως στοιχείο Name (component)
      - *Δίδονται παραδείγματα σημάτων CLK, RESET, εισόδου X και εξόδου Y*

```
architecture BEHAVIORAL of Name_TB is

-- component UUT
component Name
port (
    CLK:     in STD_LOGIC;
    RESET:   in STD_LOGIC;
-- UUT inputs
    X:       in STD_LOGIC;
-- UUT outputs
    Y:      out STD_LOGIC);
end component;
```

Τα ports του component Name αντιγράφονται από το entity Name

Προγράμματα δοκιμών στη VHDL
Περιγραφή συμπεριφοράς

- Κωδικοποιούμε το πρόγραμμα δοκιμών στη VHDL
  - *Στη συνέχεια, περιγράφουμε τις δηλώσεις της αρχιτεκτονικής της οντότητας Name_TB*
    - στη συνέχεια δηλώνουμε ως εσωτερικά σήματα, όλα τα σήματα εισόδου και εξόδου που αναφέρονται στο στοιχείο Name
      - *τα σήματα CLK και RESET αρχικοποιούνται στο '0' και '1', αντίστοιχα*
      - *η είσοδος X αρχικοποιείται στην άγνωστη τιμή 'X'*
    - ολοκληρώνουμε τις δηλώσεις με τη δήλωση της περιόδου του CLK ως χρονική σταθερά (π.χ. 10 ns)

```
-- UUT Inputs
signal CLK    : STD_LOGIC := '0';
signal RESET  : STD_LOGIC := '1';

signal X      : STD_LOGIC := 'X';

-- UUT Outputs
signal Y      : STD_LOGIC;

-- Clock period definitions
constant CLK_period : time := 10 ns;
```

Προγράμματα δοκιμών στη VHDL
Περιγραφή συμπεριφοράς

- **Κωδικοποιούμε το πρόγραμμα δοκιμών στη VHDL**
  - *Τέλος, περιγράφουμε το σώμα της αρχιτεκτονικής της οντότητας Name_TB*
    - αρχίζει μετά το begin και ολοκληρώνεται με το end behavioral;
    - στην αρχή διασυνδέουμε την οντότητα Name_TB με το στοιχείο Name με τη χρήση μίας ταυτόχρονης εντολής στοιχείων (UUT instantiation)

```
begin

-- UUT instantiation
  uut: Name
    port map (
         CLK   => CLK,
         RESET => RESET,
         X     => X,
         Y     => Y
          );
```

Προσοχή! Το στοιχείο Name (μετά το UUT:) πρέπει να φαίνεται ότι είναι συνδεδεμένο με το Name_TB στην ιεραρχική δομή των simulation sources του VIVADO

Προγράμματα δοκιμών στη VHDL
Περιγραφή συμπεριφοράς

- **Κωδικοποιούμε το πρόγραμμα δοκιμών στη VHDL**
  - *Τέλος, περιγράφουμε το σώμα της αρχιτεκτονικής της οντότητας Name_TB*
    - στη συνέχεια περιγράφουμε το σήμα CLK σε μία διεργασία (CLK_process)
      - *προαιρετικό στη δοκιμή συνδυαστικής λογικής*

```
-- Clock process definition

CLK_process : process
    begin
      CLK <= '0';
      wait for CLK_period/2;
      CLK <= '1';
      wait for CLK_period/2;
    end process;
```

Προσοχή! Το σήμα CLK ξεκινάει από την τιμή '0'

Προγράμματα δοκιμών στη VHDL
Περιγραφή συμπεριφοράς

- Κωδικοποιούμε το πρόγραμμα δοκιμών στη VHDL
  - *Τέλος, περιγράφουμε το σώμα της αρχιτεκτονικής της οντότητας Name_TB*
    - τέλος περιγράφουμε το σήμα RESET και τις υπόλοιπες εισόδους της οντότητας NAME στην ίδια διεργασία (stimulus_process)
      - *η ανάθεση τιμών γίνεται στην κατερχόμενη ακμή του ρολογιού*

```
-- Stimulus process
stimulus_process: process
    begin
-- RESET deasserted on CLK falling edge
      RESET <= '1';
      wait for 100 ns;
      wait until (CLK = '0' and CLK'event);
      RESET <= '0';
-- συνεχίζεται ...
```

Εδώ ορίζεται η χρονική περίοδος του σήματος GSR

Εδώ εξασφαλίζουμε την ανάθεση τιμών στην κατερχόμενη ακμή του CLK

Προσοχή! Πρέπει να εξασφαλίσουμε ότι RESET = 1 τουλάχιστον για 100 ns που διαρκεί η χρονική περίοδος που το FPGA επαναφέρει στην τιμή '0' όλους τους καταχωρητές και τις εξόδους του με το εσωτερικό σήμα Global Set/Reset (GSR), ώστε να μην χαθεί κάποια από τις εισόδους που θα δημιουργήσει το πρόγραμμα δοκιμών Name_TB κατά τη διάρκεια της προσομοίωσης των synthesized και implemented design models

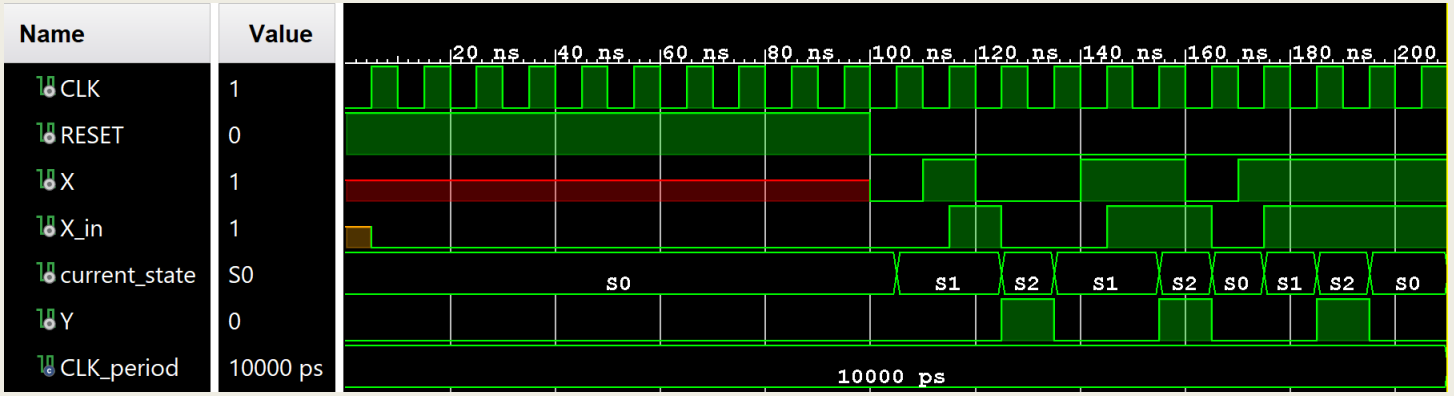
Προγράμματα δοκιμών στη VHDL
Περιγραφή συμπεριφοράς

```
-- συνεχίζεται ...
-- Inputs asserted and deasserted on CLK falling edge
      X <= '0';
      wait for 1*CLK_period;
      X <= '1';
      wait for 1*CLK_period;
      X <= '0';
      wait for 2*CLK_period;
      X <= '1';
      wait for 2*CLK_period;
      X <= '0';
      wait for 1*CLK_period;
      X <= '1';
      wait for 4*CLK_period;
-- Message and simulation end
      report "TESTS COMPLETED";
      stop(2);
   end process;
end behavioral;
```

Προσοχή! Οι αναθέσεις τιμών στα σήματα να γίνονται στην κατερχόμενη ακμή του CLK για να μην δημιουργούνται παραβιάσεις στους χρόνους σταθεροποίησης (set-up) και διατήρησης (hold) των καταχωρητών

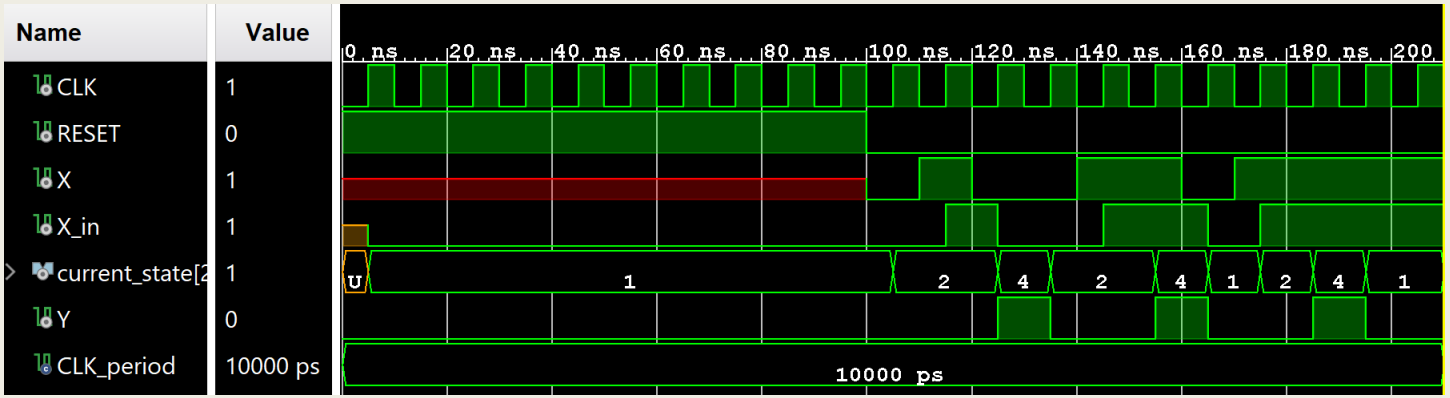
Σχεδίαση μηχανών FSM στη VHDL
Περιγραφή συμπεριφοράς

- Χρονικό διάγραμμα της προσομοίωσης συμπεριφοράς του ανιχνευτή ακολουθίας 2 διαδοχικών bit (NAME = PATTERN_FSM)
  - *Moore με αυτόματη κωδικοποίηση καταστάσεων*



S0 = 00
S1 = 01
S2 = 10

  - *Moore με κωδικοποίηση μοναδικού σημαντικού (one-hot)*

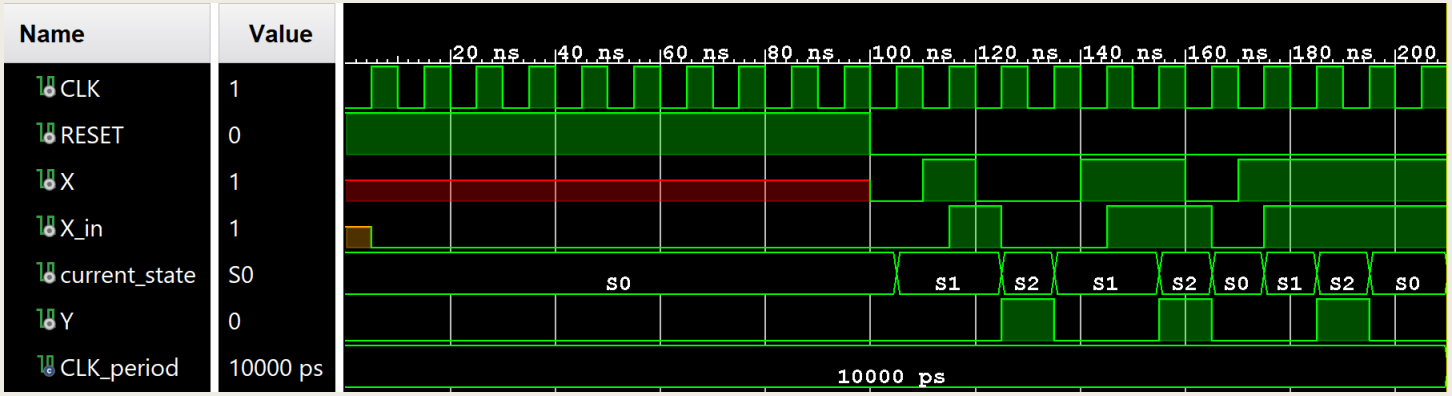


S0 = 001
S1 = 010
S2 = 100

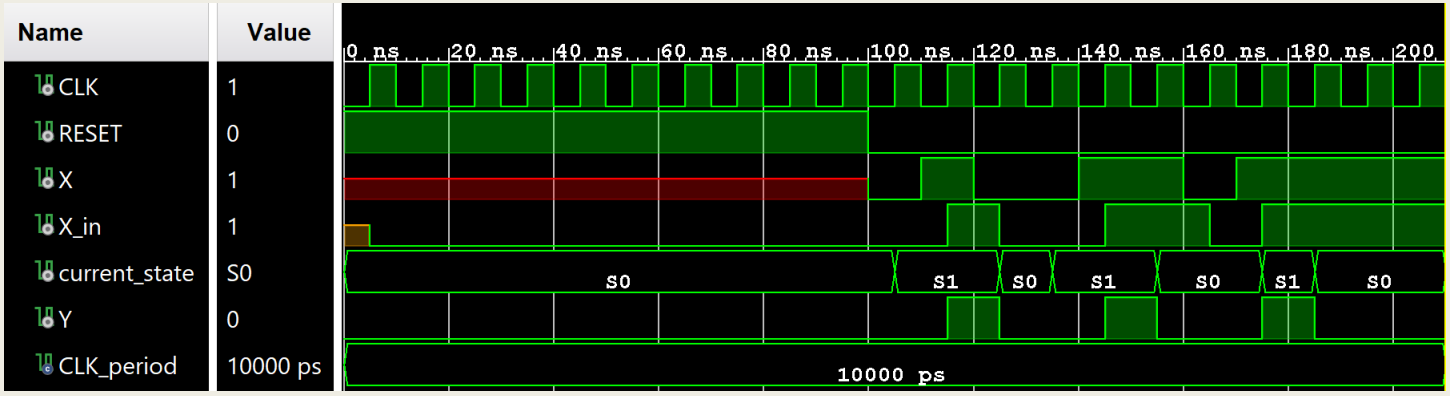
Τα χρονικά διαγράμματα της προσομοίωσης συμπεριφοράς ταυτίζονται

Σχεδίαση μηχανών FSM στη VHDL
Περιγραφή συμπεριφοράς

- Χρονικό διάγραμμα της προσομοίωσης συμπεριφοράς του ανιχνευτή ακολουθίας 2 διαδοχικών bit (NAME = PATTERN_FSM)
  - *Moore με αυτόματη κωδικοποίηση καταστάσεων*



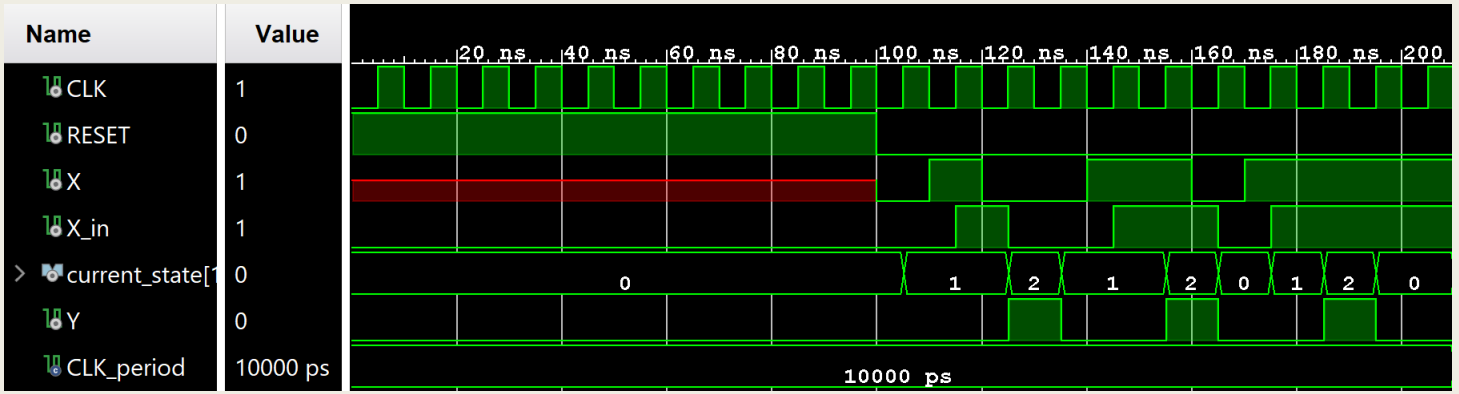
  - *Mealy με αυτόματη κωδικοποίηση καταστάσεων*



Η έξοδος εμφανίζεται ένα κύκλο πριν

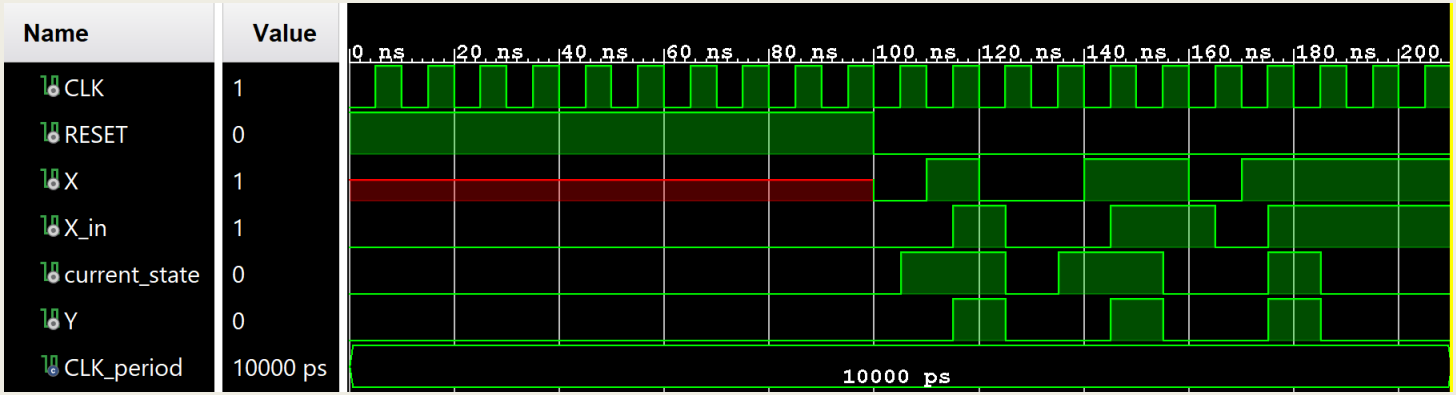
Σχεδίαση μηχανών FSM στη VHDL
Περιγραφή συμπεριφοράς

- Χρονικό διάγραμμα της λογικής προσομοίωσης μετά τη **σύνθεση** του ανιχνευτή ακολουθίας 2 διαδοχικών bit
  - *Moore με αυτόματη κωδικοποίηση καταστάσεων*



S0 = 00
S1 = 01
S2 = 10

  - *Mealy με αυτόματη κωδικοποίηση καταστάσεων*

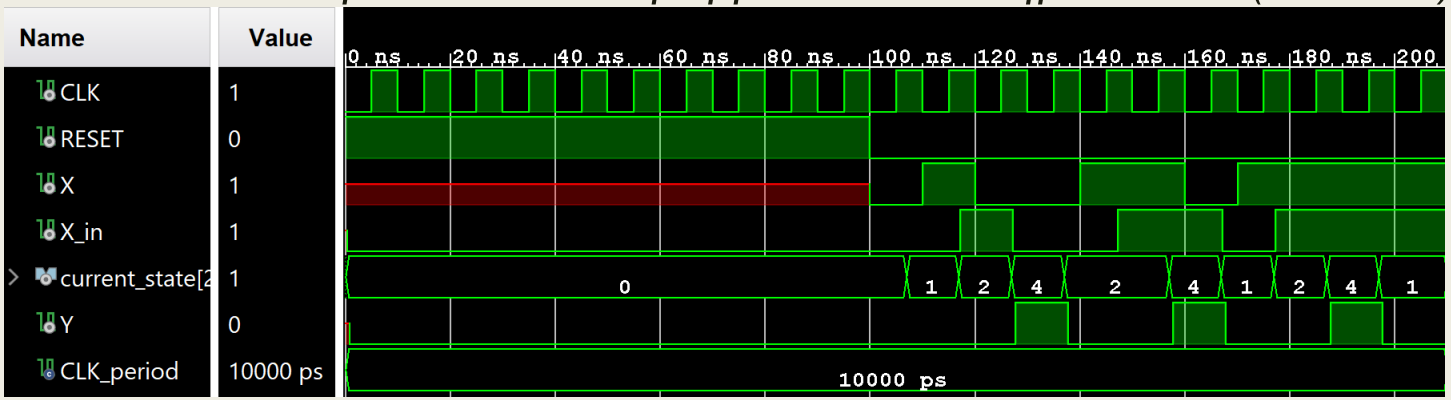


S0 = 0
S1 = 1

Η λογική προσομοίωση πρέπει να ταυτίζεται με την προσομοίωση συμπεριφοράς

Σχεδίαση μηχανών FSM στη VHDL
Περιγραφή συμπεριφοράς

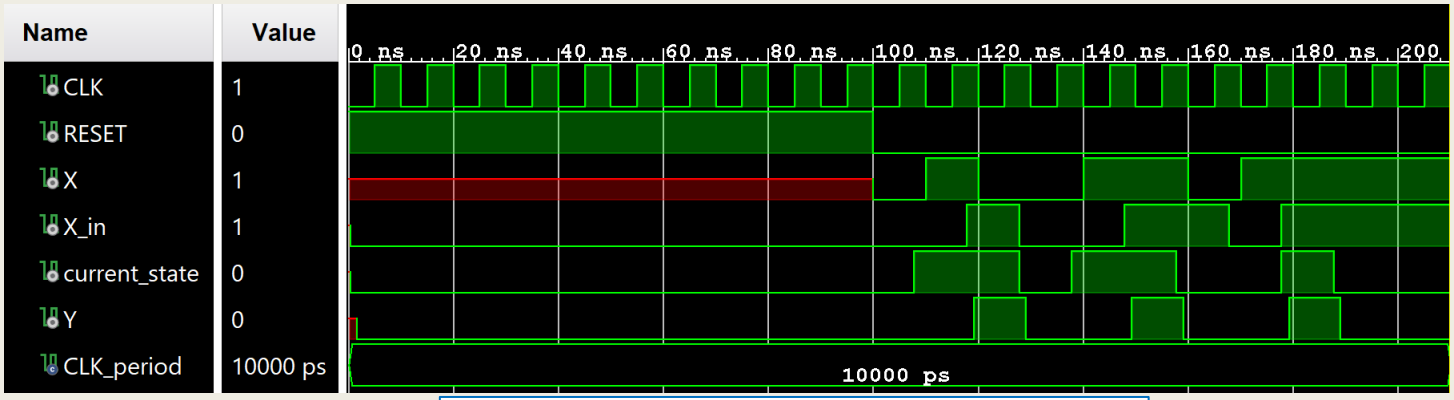
- Χρονικό διάγραμμα της χρονικής προσομοίωσης μετά τη **σύνθεση** του ανιχνευτή ακολουθίας 2 διαδοχικών bit
  - *Moore με κωδικοποίηση μοναδικού σημαντικού (one-hot)*



S0 = 001
S1 = 010
S2 = 100

Προσοχή! X_in = '0' και current_state = "000" για 100 ns λόγω του σήματος GSR του FPGA

  - *Mealy με αυτόματη κωδικοποίηση καταστάσεων*

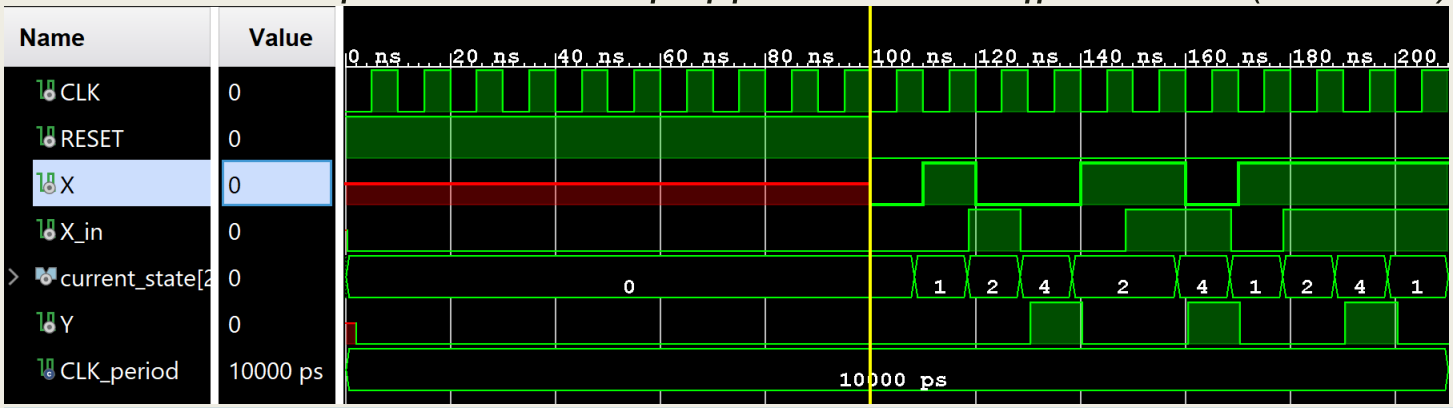


S0 = 0
S1 = 1

Η έξοδος εμφανίζεται ένα κύκλο πριν

Σχεδίαση μηχανών FSM στη VHDL
Περιγραφή συμπεριφοράς

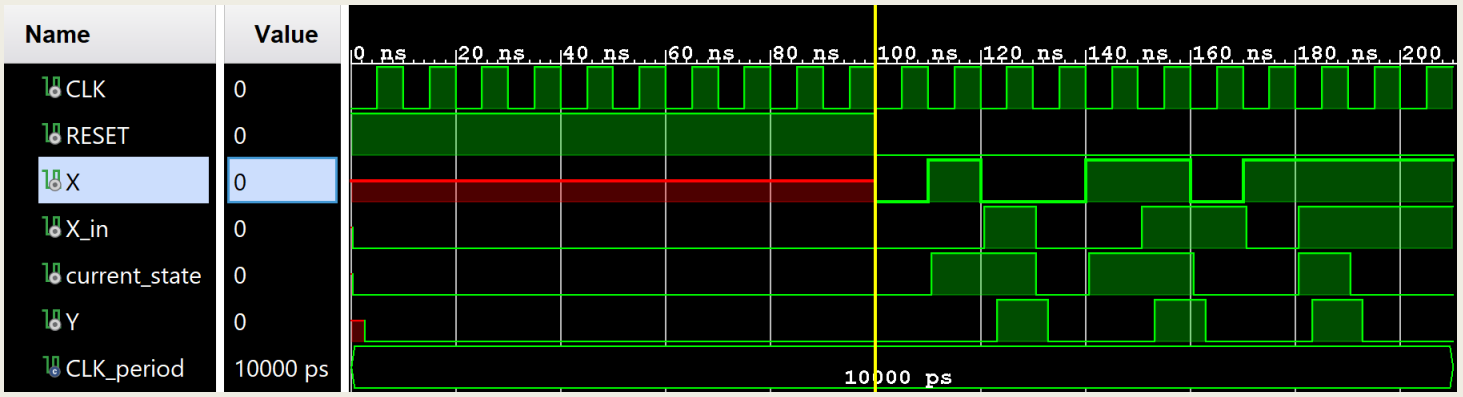
- Χρονικό διάγραμμα της χρονικής προσομοίωσης μετά την **υλοποίηση** του ανιχνευτή ακολουθίας 2 διαδοχικών bit
  - *Moore με κωδικοποίηση μοναδικού σημαντικού (one-hot)*



S0 = 001
S1 = 010
S2 = 100

Προσοχή! X_in = '0' και current_state = "000" για 100 ns λόγω του σήματος GSR του FPGA

  - *Mealy με αυτόματη κωδικοποίηση καταστάσεων*

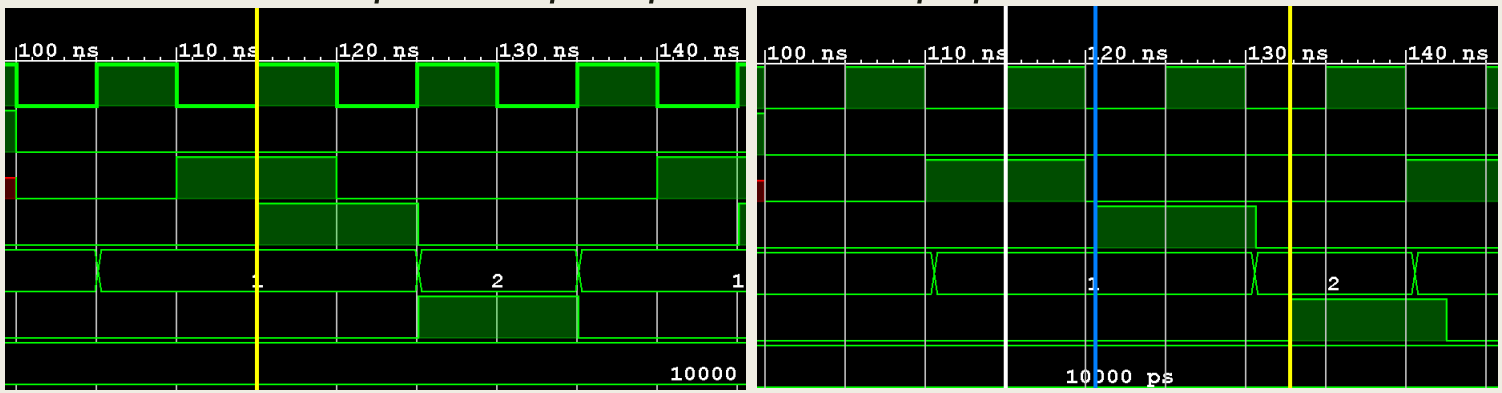


S0 = 0
S1 = 1

Η έξοδος εμφανίζεται ένα κύκλο πριν

Σχεδίαση μηχανών FSM στη VHDL
Περιγραφή συμπεριφοράς

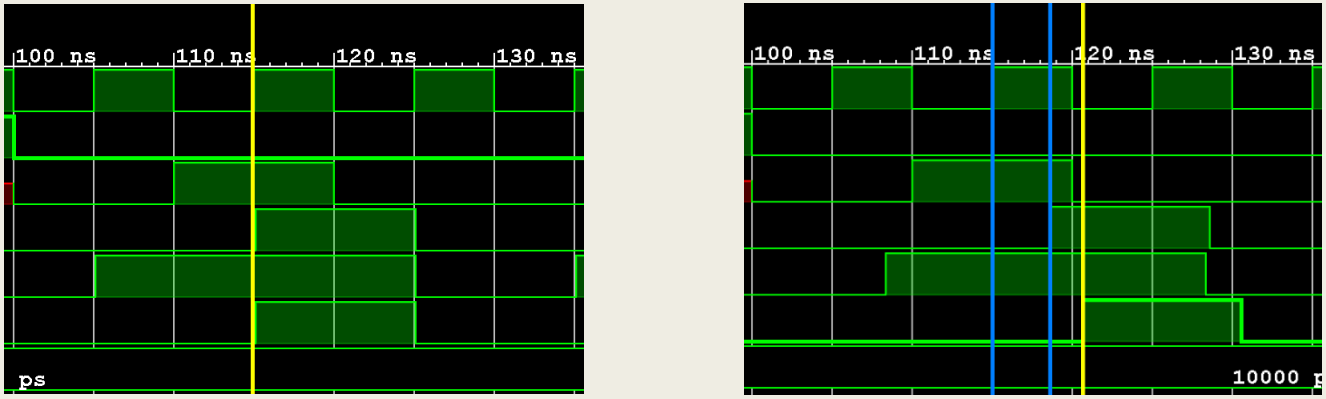
- Χρονικό διάγραμμα της λογικής και της χρονικής προσομοίωσης μετά την **υλοποίηση** του ανιχνευτή ακολουθίας 2 διαδοχικών bit
  - *Moore με αυτόματη κωδικοποίηση καταστάσεων*



S0 = 00
S1 = 01
S2 = 10

Είναι στην S1 και γίνεται X = 1. Στον επόμενο κύκλο πηγαίνει στην S2 και γίνεται το Y = 1

  - *Mealy με αυτόματη κωδικοποίηση καταστάσεων*



S0 = 0
S1 = 1

Είναι στην S1 και γίνεται X = 1. Στον ίδιο κύκλο γίνεται Y = 1. Στον επόμενο κύκλο πηγαίνει στην S0

Προετοιμασία συνδυαστικής λογικής για προσομοίωση στο VIVADO

- Για την ορθή χρονική προσομοίωση τοποθετούμε στον κώδικα της συνδυαστικής λογικής **καταχωρητές** στην είσοδο και στην έξοδο
  - *Παράδειγμα κώδικα συνδυαστικής λογικής*

```
entity CL is port (
  X: in STD_LOGIC;
  Y: out STD_LOGIC);
end CL;
architecture BEHAVIORAL of CL is
signal INT: STD_LOGIC;
begin
  process (X)
  begin
    INT <= X;
    if (INT = '1') then Y <= X;
     else Y <= '0';
    end if;
  end process;
end BEHAVIORAL;
```

Προετοιμασία συνδυαστικής λογικής για προσομοίωση στο VIVADO

- Για την ορθή χρονική προσομοίωση τοποθετούμε στον κώδικα της συνδυαστικής λογικής **καταχωρητές** στην είσοδο και στην έξοδο
  - *Προσθήκη καταχωρητών με σήματα CLK και RESET*

```
entity CLREG is port (
  CLK:    in STD_LOGIC;
  RESET:  in STD_LOGIC;
  X:      in STD_LOGIC;
  Y:     out STD_LOGIC);
end CLREG;
architecture BEHAVIORAL of CLREG is
signal INT:  STD_LOGIC;
signal X_IN: STD_LOGIC;
signal Y_IN: STD_LOGIC;
begin
  process (CLK) -- input register
  begin
    if (CLK = '1' and CLK'event) then
      if (RESET = '1') then X_IN <= '0';
      else X_IN <= X;
      end if;
    end if;
  end process;
```

Απαιτείται προσθήκη των εσωτερικών σημάτων X_IN και Y_IN

Καταχωρητής στην είσοδο

Προετοιμασία συνδυαστικής λογικής για προσομοίωση στο VIVADO

- Για την ορθή χρονική προσομοίωση τοποθετούμε στον κώδικα της συνδυαστικής λογικής **καταχωρητές** στην είσοδο και στην έξοδο
  - *Προσθήκη καταχωρητών με σήματα CLK και RESET*

```
process (X_IN) -- use of internal signals
  begin
    INT <= X_IN;
    if (INT = '1') then Y_IN <= X_IN;
     else Y_IN <= '0';
    end if;
  end process;

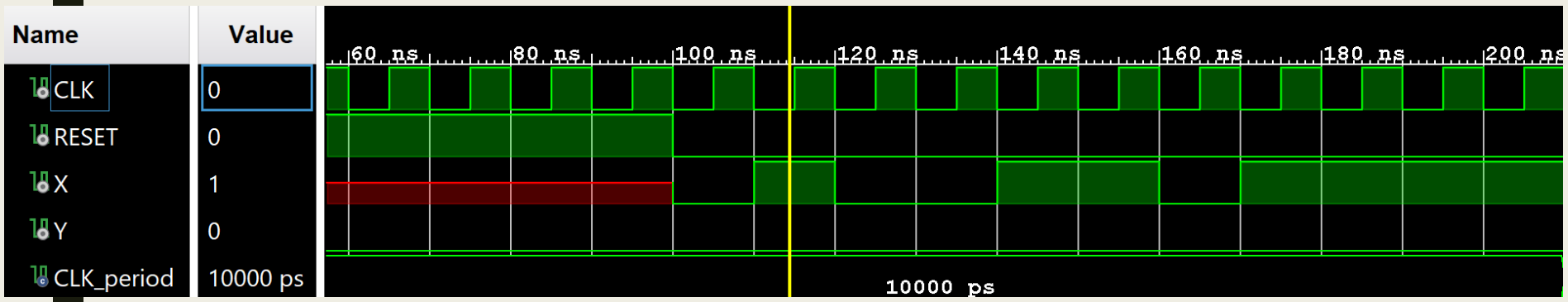
 process (CLK) -- output register
  begin
    if (CLK = '1' and CLK'event) then
      if (RESET = '1') then Y <= '0';
      else Y <= Y_IN;
      end if;
    end if;
  end process;
end BEHAVIORAL;
```

Συνδυαστική λογική

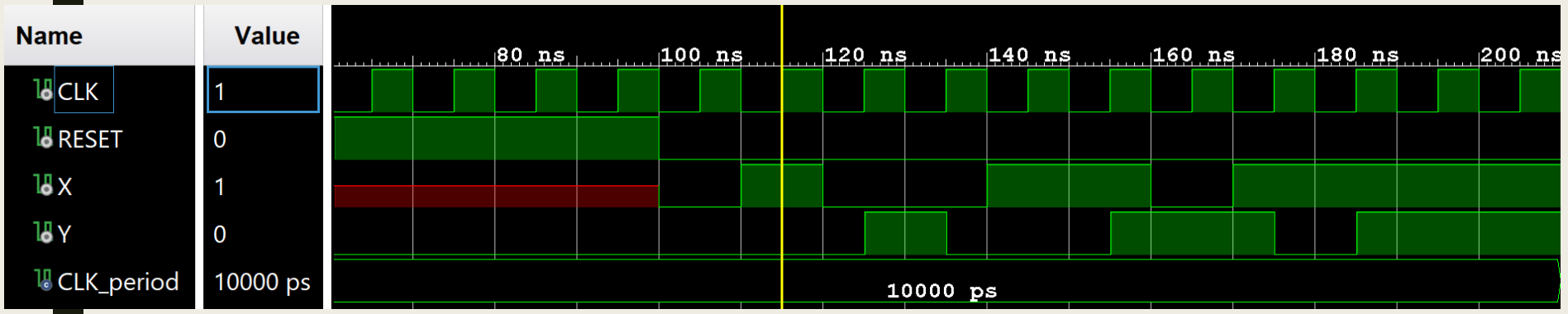
Καταχωρητής στην έξοδο

Προετοιμασία συνδυαστικής λογικής για προσομοίωση στο VIVADO

- Χρονικό διάγραμμα της προσομοίωσης της οντότητας CLREG, με τη χρήση του προγράμματος δοκιμών LCREG_TB που μοιάζει στο PATTERN_FSM_TB
  - *Προσομοίωση συμπεριφοράς (Y = 0)*



  - *Λογική προσομοίωση μετά τη σύνθεση (ή την υλοποίηση) (Y = X)*



Δεν συμφωνούν οι δύο προσομοιώσεις γιατί δεν δηλώθηκε το INT στη λίστα ευαισθησίας

Πρόγραμμα δοκιμής με σύγκριση μοντέλων στη VHDL – Περιγραφή συμπεριφοράς

- Τροποποιούμε κατάλληλα το LCREG_TB σε LCREG_TB1, ώστε να παρέχει σύγκριση των λογικών προσομοιώσεων του **behavioral model (BEH)** με το **synthesized design model** ή το **implemented design model (UUT)**
  - *προσθέτουμε στην αρχή του LCREG_TB1 το **behavioral model CLREG_BEH***

```
-- Behavioral model
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity CLREG_BEH is port (
  CLK:    in STD_LOGIC;
  RESET: in STD_LOGIC;
  X:      in STD_LOGIC;
  Y:     out STD_LOGIC);
end CLREG_BEH;

architecture Behavioral of CLREG_BEH is
signal INT:  STD_LOGIC;
signal X_IN: STD_LOGIC;
signal Y_IN: STD_LOGIC;

begin
  process (CLK) -- input register
  begin
    if (CLK = '1' and CLK'event) then
      if (RESET = '1') then X_IN <= '0';
      else X_IN <= X;
      end if;
    end if;
  end process;
```

```
process (X_IN) -- use of internal signals
  begin
    INT <= X_IN;
    if (INT = '1') then Y_IN <= X_IN;
     else Y_IN <= '0';
    end if;
  end process;

 process (CLK) -- output register
  begin
    if (CLK = '1' and CLK'event) then
      if (RESET = '1') then Y <= '0';
      else Y <= Y_IN;
      end if;
    end if;
  end process;

end Behavioral;
```

Πρόγραμμα δοκιμής με σύγκριση μοντέλων στη VHDL – Περιγραφή συμπεριφοράς

- Τροποποιούμε κατάλληλα το LCREG_TB σε LCREG_TB1, ώστε να παρέχει σύγκριση των λογικών προσομοιώσεων του **behavioral model (BEH)** με το **synthesized design model** ή το **implemented design model (UUT)**
  - *προσθέτουμε στις δηλώσεις της αρχιτεκτονικής του CLREG_TB1 το component CLREG_BEH επιπλέον του CLREG, καθώς και δύο επιπλέον εσωτερικά σήματα: το Y_BEH και το Y_XOR για τη σύγκριση*

```
46  -- TB model
47  LIBRARY ieee;
48  USE ieee.std_logic_1164.ALL;
49  USE ieee.numeric_std.ALL;
50  USE std.env.all;
51
52  entity CLREG_TB1 is
53  end CLREG_TB1;
54
55  architecture Behavioral of CLREG_TB1 is
56  -- component behavioral model
57  -- component CLREG_BEH
58  component CLREG_BEH
59  port (
60    CLK:    in STD_LOGIC;
61    RESET:  in STD_LOGIC;
62    X:      in STD_LOGIC;
63    Y:     out STD_LOGIC);
64  end component;
```

```
66  --component UUT
67  --component CLREG
68  component CLREG
69  port (
70      CLK:    in STD_LOGIC;
71      RESET:  in STD_LOGIC;
72      X:      in STD_LOGIC;
73      Y:     out STD_LOGIC);
74  end component;
75
76  -- Internal signals
77  signal CLK   : STD_LOGIC := '0';
78  signal RESET : STD_LOGIC := '1';
79  signal X     : STD_LOGIC := 'X';
80  signal Y     : STD_LOGIC;
81
82  -- Extra internal signals for comparison
83  signal Y_BEH     : STD_LOGIC;
84  signal Y_XOR     : STD_LOGIC;
85
86  -- Clock period definitions
87  constant CLK_period : time := 10 ns;
```

Πρόγραμμα δοκιμής με σύγκριση μοντέλων στη VHDL – Περιγραφή συμπεριφοράς

- Τροποποιούμε κατάλληλα το LCREG_TB σε LCREG_TB1, ώστε να παρέχει σύγκριση των λογικών προσομοιώσεων του **behavioral model (BEH)** με το **synthesized design model** ή το **implemented design model (UUT)**
  - *διασυνδέουμε στο σώμα της αρχιτεκτονικής του CLREG_TB1 και το component **CLREG_BEH** (επιπλέον του component CLREG) με τη χρήση μίας ταυτόχρονης εντολής στοιχείων (behavioral model instantiation)*
  - *προσθέτουμε την εντολή ανάθεσης τιμής στο εσωτερικό σήμα **Y_XOR***
    - όπου το **Y είναι διάφορο του Y_BEH, το Y_XOR γίνεται 1**

```
 89  begin
 90  -- UUT instantiation
 91  uut: CLREG
 92      port map (
 93          CLK    => CLK,
 94          RESET  => RESET,
 95          X      => X,
 96          Y      => Y);
 97
 98  -- Behavioral model instantiation
 99  BEH: CLREG_BEH
100      port map (
101          CLK    => CLK,
102          RESET  => RESET,
103          X      => X,
104          Y      => Y_BEH);
```

```
106  -- Clock process definition
107  CLK_process : process
108      begin
109          CLK <= '0';
110          wait for CLK_period/2;
111          CLK <= '1';
112          wait for CLK_period/2;
113      end process;
114
115  -- output XOR
116  Y_XOR <= Y xor Y_BEH;
```

Το **UUT** παράγει το **Y**, ενώ το **BEH** παράγει το **Y_BEH**

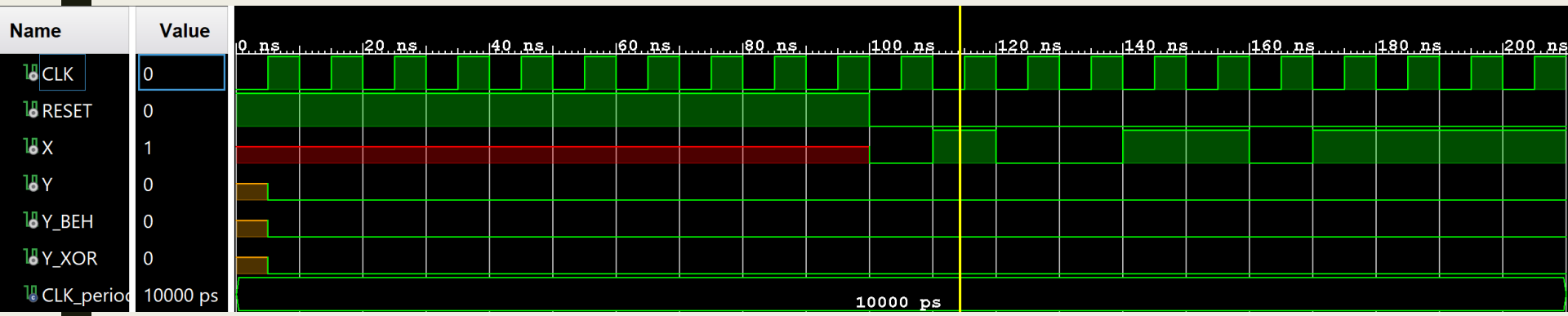
Πρόγραμμα δοκιμής με σύγκριση μοντέλων στη VHDL – Περιγραφή συμπεριφοράς

- Τροποποιούμε κατάλληλα το LCREG_TB σε LCREG_TB1, ώστε να παρέχει σύγκριση των λογικών προσομοιώσεων του **behavioral model (BEH)** με το **synthesized design model** ή το **implemented design model (UUT)**
  - *το **stimulus process** παραμένει ως έχει*
    - παράγει εισόδους και για τα δύο: CLREG (UUT) και CLREG_BEH (BEH)

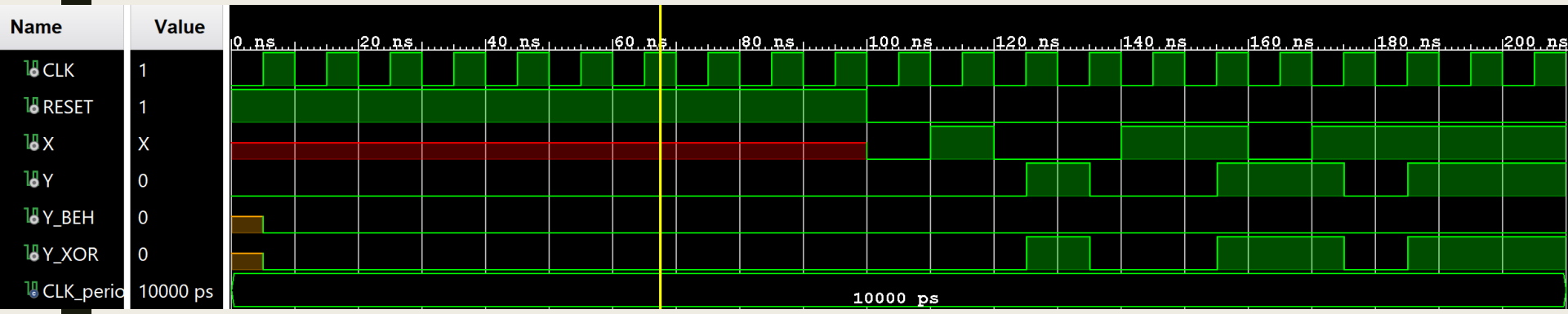
```
118  -- Stimulus process
119  stimulus_process: process
120      begin
121  -- RESET deasserted on CLK falling edge
122        RESET <= '1';
123        wait for 100 ns;
124        wait until (CLK = '0' and CLK'event);
125        RESET <= '0';
126  -- Inputs asserted and deasserted on CLK falling edge
127        X <= '0';
128        wait for 1*CLK_period;
129        X <= '1';
130        wait for 1*CLK_period;
131        X <= '0';
132        wait for 2*CLK_period;
133        X <= '1';
134        wait for 2*CLK_period;
135        X <= '0';
136        wait for 1*CLK_period;
137        X <= '1';
138        wait for 4*CLK_period;
139  -- Message and simulation end
140        report "TESTS COMPLETED";
141        stop(2);
142     end process;
143  end behavioral;
```

Προετοιμασία συνδυαστικής λογικής για προσομοίωση στο VIVADO

- Χρονικό διάγραμμα της λογικής προσομοίωσης της οντότητας CLREG, με τη χρήση του προγράμματος δοκιμών CLREG_TB1 για σύγκριση:
  - *του behavioral model (CLREG_BEH) με το behavioral model (UUT:CLREG)*



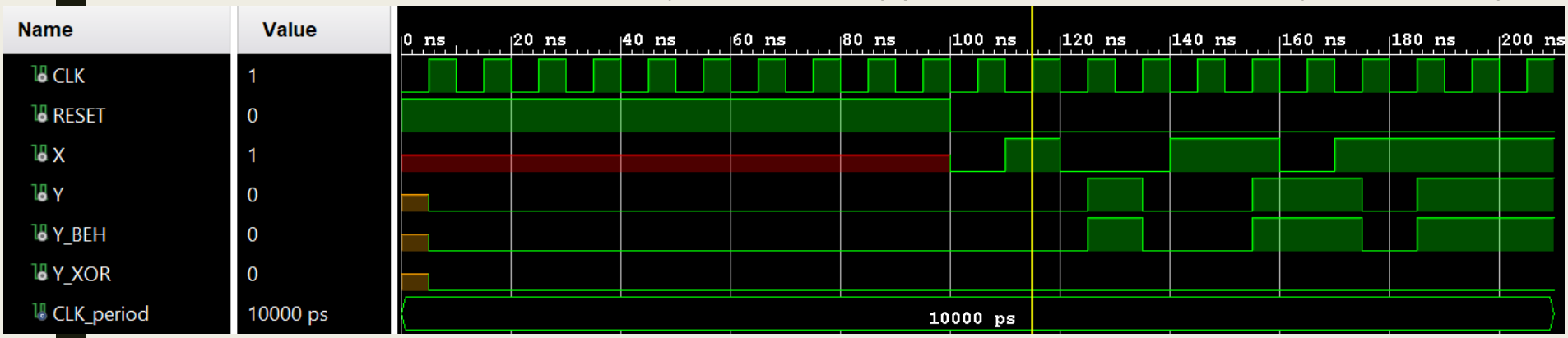
  - *του behavioral model (CLREG_BEH) με το synthesis design model*



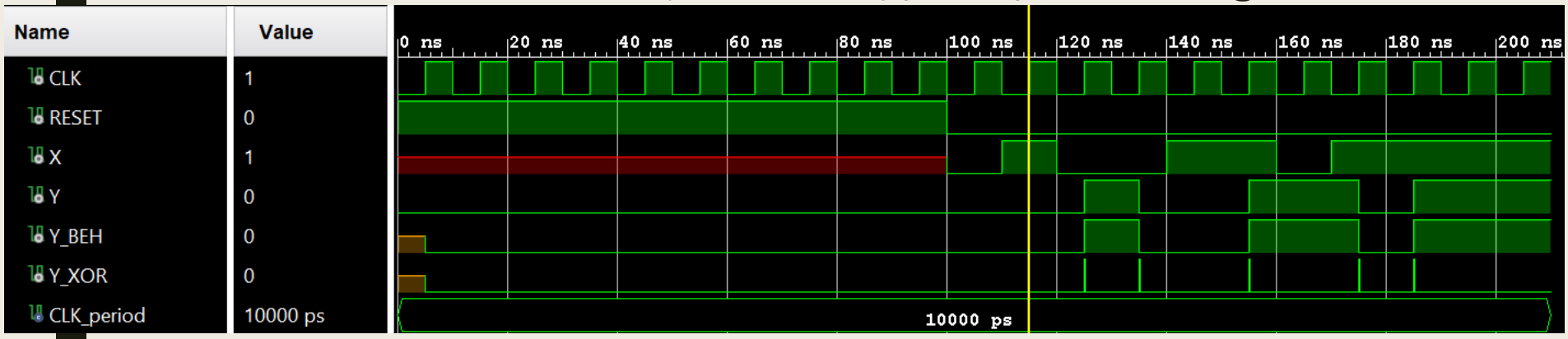
Δεν συμφωνούν οι δύο προσομοιώσεις γιατί δεν δηλώθηκε το INT στη λίστα ευαισθησίας

Προετοιμασία συνδυαστικής λογικής για προσομοίωση στο VIVADO

- Χρονικό διάγραμμα της λογικής προσομοίωσης της οντότητας CLREG, με τη χρήση του προγράμματος δοκιμών CLREG_TB1 για σύγκριση:
  - *του behavioral model (CLREG_BEH) με το behavioral model (UUT:CLREG)*



  - *του behavioral model (CLREG_BEH) με το synthesis design model*



Συμφωνούν πλέον οι δύο προσομοιώσεις γιατί δηλώθηκε το INT στη λίστα ευαισθησίας

Διατάξεις μνήμης RAM στη VHDL
Περιγραφή συμπεριφοράς

- Μια διάταξη μνήμης RAM με **διευθύνσεις** (address) των ***N*** bit και **δεδομένα** (data) των ***M*** bit αποθηκεύει **2^N λέξεις** μεγέθους **M** bit
  - *διαθέτει σύγχρονη εγγραφή με έγκριση (όταν το σήμα WE = 1)*
  - *διαθέτει σύγχρονο διάβασμα, όταν υλοποιείται με Block RAMs*
  - *διαθέτει ασύγχρονο διάβασμα, όταν υλοποιείται με LUTs*
- Η οντότητα **RAM_array** διαθέτει:
  - *είσοδο CLK*
  - *είσοδο WE για έγκριση εγγραφής*
  - *είσοδο ADDR των N bit για διευθυνσιοδότηση*
  - *είσοδο DATA_IN των M bit για εγγραφή δεδομένων*
  - *έξοδο DATA_OUT των M bit για διάβασμα δεδομένων*
    - δεν διαθέτει RESET, αλλά δύναται να μηδενίσει τα περιεχόμενά της μέσω του σήματος GSR

Διατάξεις μνήμης RAM στη VHDL
Περιγραφή συμπεριφοράς

- Κωδικοποίηση διάταξης μνήμης RAM με 2^N **λέξεις** μεγέθους **M** bit
  - *Αρχικά περιγράφουμε την οντότητα **RAM_array***

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

entity RAM_array is
  generic (
    N : positive := 10;    -- address length
    M : positive := 32);   -- data word length
  port (
    CLK:      in STD_LOGIC;
    WE:       in STD_LOGIC;
    ADDR:     in STD_LOGIC_VECTOR (N-1 downto 0);
    DATA_IN:  in STD_LOGIC_VECTOR (M-1 downto 0);
    DATA_OUT: out STD_LOGIC_VECTOR (M-1 downto 0));
end RAM_array;
```

Single port RAM

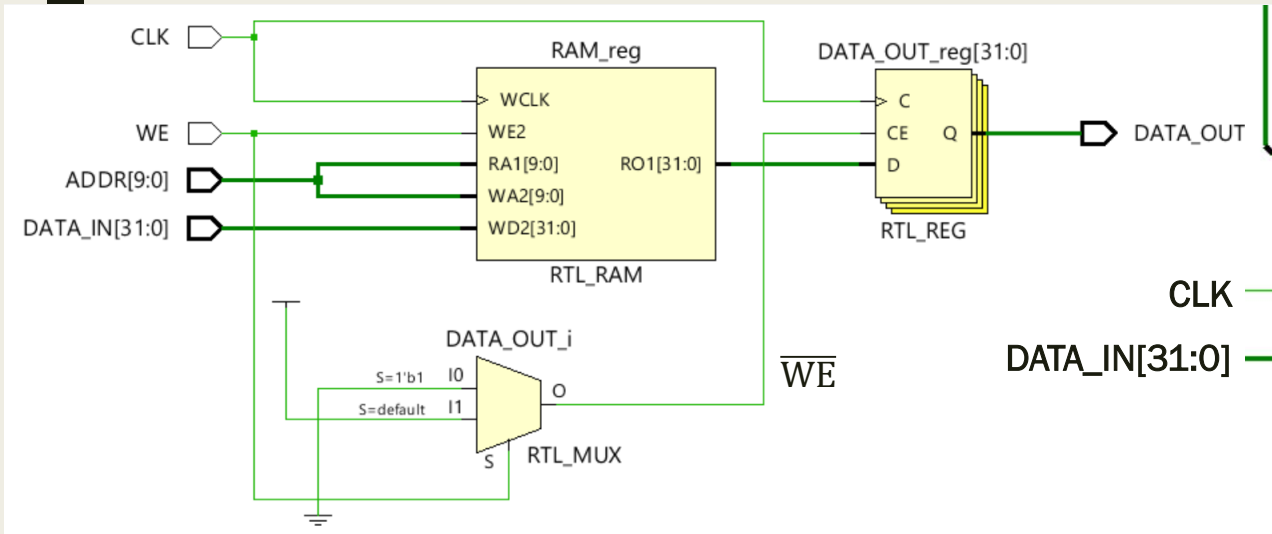
Διατάξεις μνήμης RAM στη VHDL
Περιγραφή συμπεριφοράς

- Κωδικοποίηση διάταξης μνήμης RAM με 2^N **λέξεις** μεγέθους **M** bit
  - *Στη συνέχεια περιγράφουμε την αρχιτεκτονική της οντότητας* ***RAM_array*** *για υλοποίηση με* ***BLOCK_RAM***
    - χρησιμοποιείται ο τύπος **array**
    - **σύγχρονη δήλωση διαβάσματος, όταν WE = 0**

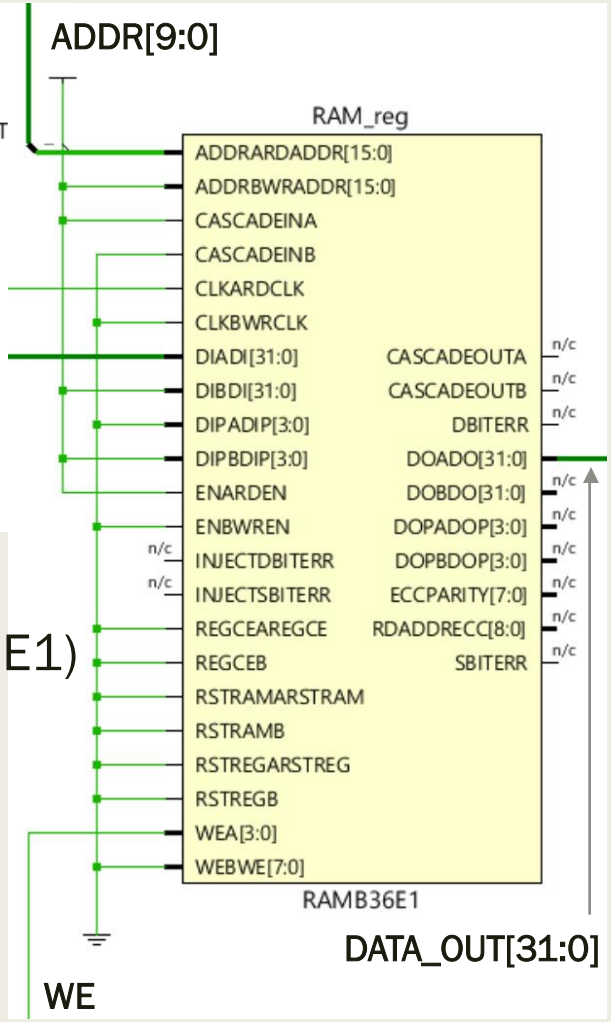
```
architecture BEHAVIORAL of RAM_array is
  type RAM_array is array (2**N-1 downto 0)
    of STD_LOGIC_VECTOR (M-1 downto 0);
  signal RAM : RAM_array;
begin
  Block_RAM: process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      if (WE = '1') then RAM(to_integer(unsigned(ADDR))) <= DATA_IN;
      else   DATA_OUT <= RAM(to_integer(unsigned(ADDR)));
      end if;
    end if;
  end process;
end BEHAVIORAL;
```

Διατάξεις μνήμης RAM στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL της διάταξης μνήμης RAM με Block RAM



- Σχηματικό διάγραμμα σε τεχνολογία FPGA (χρήση μίας Block RAM των 36K bit - RAMB36E1)
  - *single port Block RAM*



Κοινός δίαυλος διεύθυνσης για σύγχρονη εγγραφή και σύγχρονο διάβασμα

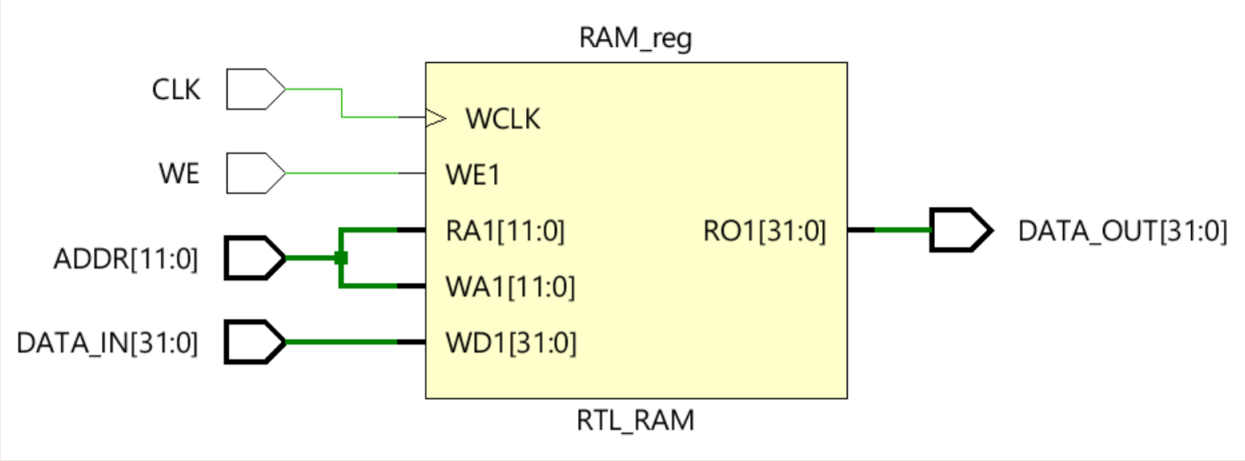
Διατάξεις μνήμης RAM στη VHDL
Περιγραφή συμπεριφοράς

- Κωδικοποίηση διάταξης μνήμης RAM με 2^N **λέξεις** μεγέθους **M** bit
  - *Στη συνέχεια περιγράφουμε την αρχιτεκτονική της οντότητας* ***RAM_array*** *για υλοποίηση με* ***distributed RAM***
    - χρησιμοποιείται ο τύπος **array**
    - **ασύγχρονη δήλωση διαβάσματος**

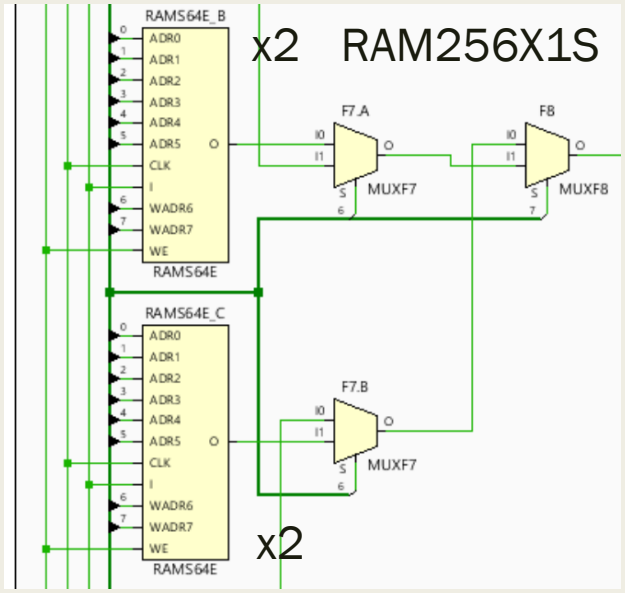
```
architecture BEHAVIORAL of RAM_array is
  type RAM_array is array (2**N-1 downto 0)
    of STD_LOGIC_VECTOR (M-1 downto 0);
  signal RAM : RAM_array;
begin
  Block_RAM: process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      if (WE = '1') then RAM(to_integer(unsigned(ADDR))) <= DATA_IN;
      end if;
    end if;
  end process;
  DATA_OUT <= RAM(to_integer(unsigned(ADDR)));
end BEHAVIORAL;
```

Διατάξεις μνήμης RAM στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL της διάταξης μνήμης RAM με distributed RAM



- Σχηματικό διάγραμμα σε τεχνολογία FPGA
  - *single port distributed RAM των 256x1 bit (RAM256X1S) των 4 LUTs*
    - Απαιτούν 2 MUXF7 και 1 MUXF8
  - *Απαιτούνται συνολικά 548 LUTs*



Κοινός δίαυλος διεύθυνσης για σύγχρονη εγγραφή και ασύγχρονο διάβασμα

Αρχείο καταχωρητών στη VHDL
Περιγραφή συμπεριφοράς

- Το αρχείο καταχωρητών είναι μια διάταξη μνήμης RAM με **διευθύνσεις** (address) των *4/5* bit (για 16/32 καταχωρητές) και **δεδομένα** (data) των *32* bit
  - *διαθέτει σύγχρονη εγγραφή με έγκριση (όταν το σήμα WE = 1)*
  - *διαθέτει ασύγχρονο διάβασμα*
  - *παρέχει τη δυνατότητα διαβάσματος δύο καταχωρητών ταυτόχρονα*
- Η οντότητα **REGFILE** διαθέτει:
  - *είσοδο CLK*
  - *είσοδο WE για έγκριση εγγραφής*
  - *είσοδο ADDR_W των 4/5 bit για διευθυνσιοδότηση ενός καταχωρητή για εγγραφή*
  - *είσοδο ADDR_R1 των 4/5 bit για διευθυνσιοδότηση ενός καταχωρητή για διάβασμα*
  - *είσοδο ADDR_R2 των 4/5 bit για διευθυνσιοδότηση ενός άλλου καταχωρητή για διάβασμα (μπορεί να είναι ίδιος με τον πρώτο)*
  - *είσοδο DATA_IN των 32 bit για εγγραφή δεδομένων*
  - *έξοδο DATA_OUT1 των 32 bit για διάβασμα δεδομένων από τον ένα καταχωρητή*
  - *έξοδο DATA_OUT2 των 32 bit για διάβασμα δεδομένων από τον άλλο καταχωρητή*

Αρχείο καταχωρητών στη VHDL
Περιγραφή συμπεριφοράς

- Κωδικοποίηση του αρχείου καταχωρητών με **16/32 λέξεις** μεγέθους **32** bit
  - *Αρχικά περιγράφουμε την οντότητα **REGFILE***

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

entity REGFILE is
  generic (
    N : positive := 4;    -- address length
    M : positive := 32);  -- data word length
  port (
    CLK:       in STD_LOGIC;
    WE:        in STD_LOGIC;
    ADDR_W:    in STD_LOGIC_VECTOR (N-1 downto 0);
    ADDR_R1:   in STD_LOGIC_VECTOR (N-1 downto 0);
    ADDR_R2:   in STD_LOGIC_VECTOR (N-1 downto 0);
    DATA_IN:   in STD_LOGIC_VECTOR (M-1 downto 0);
    DATA_OUT1: out STD_LOGIC_VECTOR (M-1 downto 0);
    DATA_OUT2: out STD_LOGIC_VECTOR (M-1 downto 0));
end REGFILE;
```

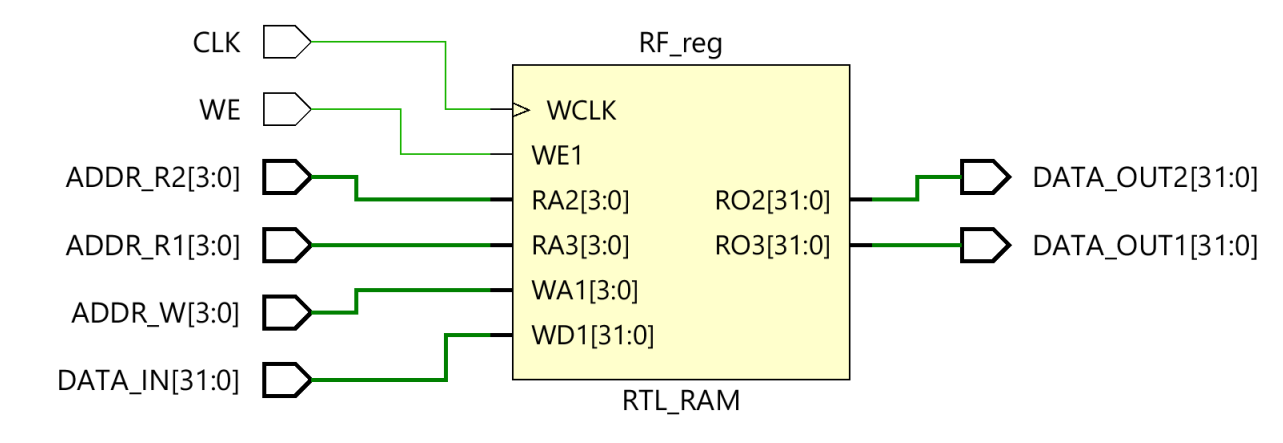
Αρχείο καταχωρητών στη VHDL
Περιγραφή συμπεριφοράς

- Κωδικοποίηση του αρχείου καταχωρητών με **16/32 λέξεις** μεγέθους **32** bit
  - *Στη συνέχεια περιγράφουμε την αρχιτεκτονική της οντότητας **REGFILE** για υλοποίηση με **distributed RAM** (λόγω μικρού μεγέθους)*
    - χρησιμοποιείται ο τύπος **array**
    - **ασύγχρονη δήλωση διπλού διαβάσματος**

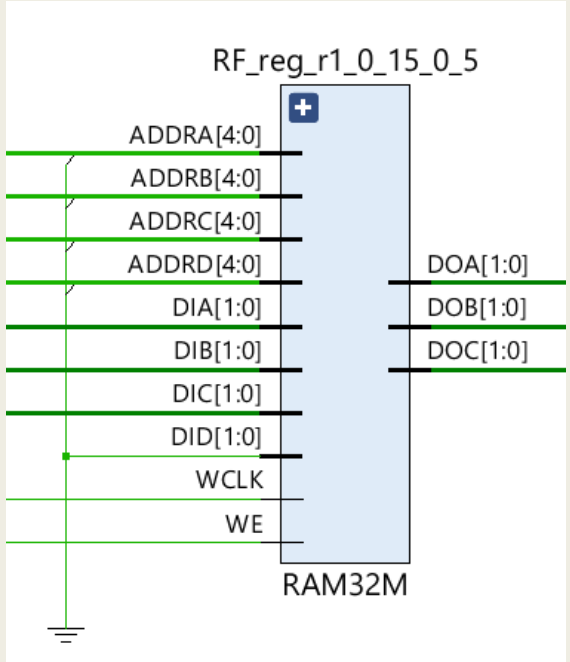
```
architecture BEHAVIORAL of REGFILE is
  type RF_array is array (2**N-1 downto 0)
    of STD_LOGIC_VECTOR (M-1 downto 0);
  signal RF : RF_array;
begin
  REG_FILE: process (CLK)
  begin
    if (CLK = '1' and CLK'event) then
      if (WE = '1') then RF(to_integer(unsigned(ADDR_W))) <= DATA_IN;
      end if;
    end if;
  end process;
  DATA_OUT1 <= RF(to_integer(unsigned(ADDR_R1)));
  DATA_OUT2 <= RF(to_integer(unsigned(ADDR_R2)));
end BEHAVIORAL;
```

Αρχείο καταχωρητών στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL του αρχείου καταχωρητών με distributed RAM



- Σχηματικό διάγραμμα σε τεχνολογία FPGA
  - *12 quad port distributed RAM (RAM32M) των 4 LUTs*
  - *απαιτούνται συνολικά 48 LUTs*



Ανεξάρτητοι δίαυλοι διεύθυνσης για σύγχρονη εγγραφή και διπλό ασύγχρονο διάβασμα

Διατάξεις μνήμης ROM στη VHDL
Περιγραφή συμπεριφοράς

- Μια διάταξη μνήμης ROM με **διευθύνσεις** (address) των ***N*** bit και **δεδομένων** (data) των ***M*** bit αποθηκεύει **2^N λέξεις** μεγέθους **M** bit
  - *υλοποιείται συνδυαστική λογική με LUTs*
- Η οντότητα **ROM_array** διαθέτει:
  - *είσοδο ADDR των N bit για διευθυνσιοδότηση*
  - *έξοδο DATA_OUT των M bit για διάβασμα δεδομένων*
    - τα δεδομένα της διάταξης μνήμης ROM ορίζονται στην αρχιτεκτονική της οντότητας

Διατάξεις μνήμης ROM στη VHDL
Περιγραφή συμπεριφοράς

- Κωδικοποίηση διάταξης μνήμης ROM με 2^N **λέξεις** μεγέθους **M** bit
  - *Αρχικά περιγράφουμε την οντότητα **ROM_array***

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

entity ROM_array is
  generic (
    N : positive := 4;    -- address length
    M : positive := 32);  -- data word length
  port (
    ADDR:       in STD_LOGIC_VECTOR (N-1 downto 0);
    DATA_OUT: out STD_LOGIC_VECTOR (M-1 downto 0));
end RAM_array;
```

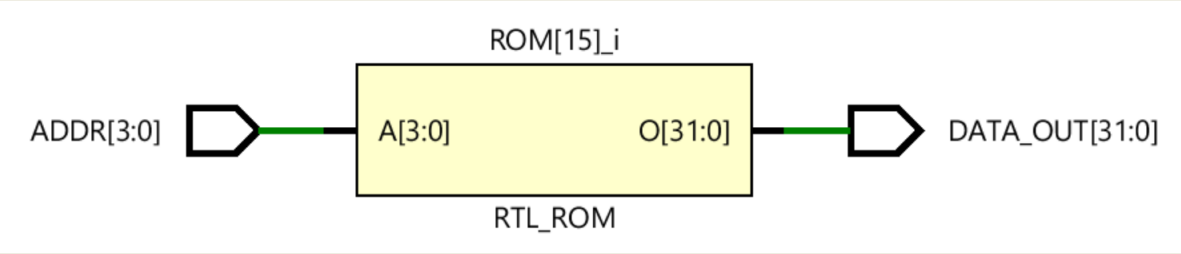
Διατάξεις μνήμης ROM στη VHDL
Περιγραφή συμπεριφοράς

- Κωδικοποίηση διάταξης μνήμης ROM με 2^N **λέξεις** μεγέθους **M** bit
  - *Στη συνέχεια περιγράφουμε την αρχιτεκτονική της οντότητας **ROM_array** για υλοποίηση με **LUTs***
    - χρησιμοποιείται ο τύπος **array**
    - τα **δεδομένα** εισάγονται με τη χρήση **σταθεράς** (constant)
    - **ασύγχρονη δήλωση διαβάσματος**

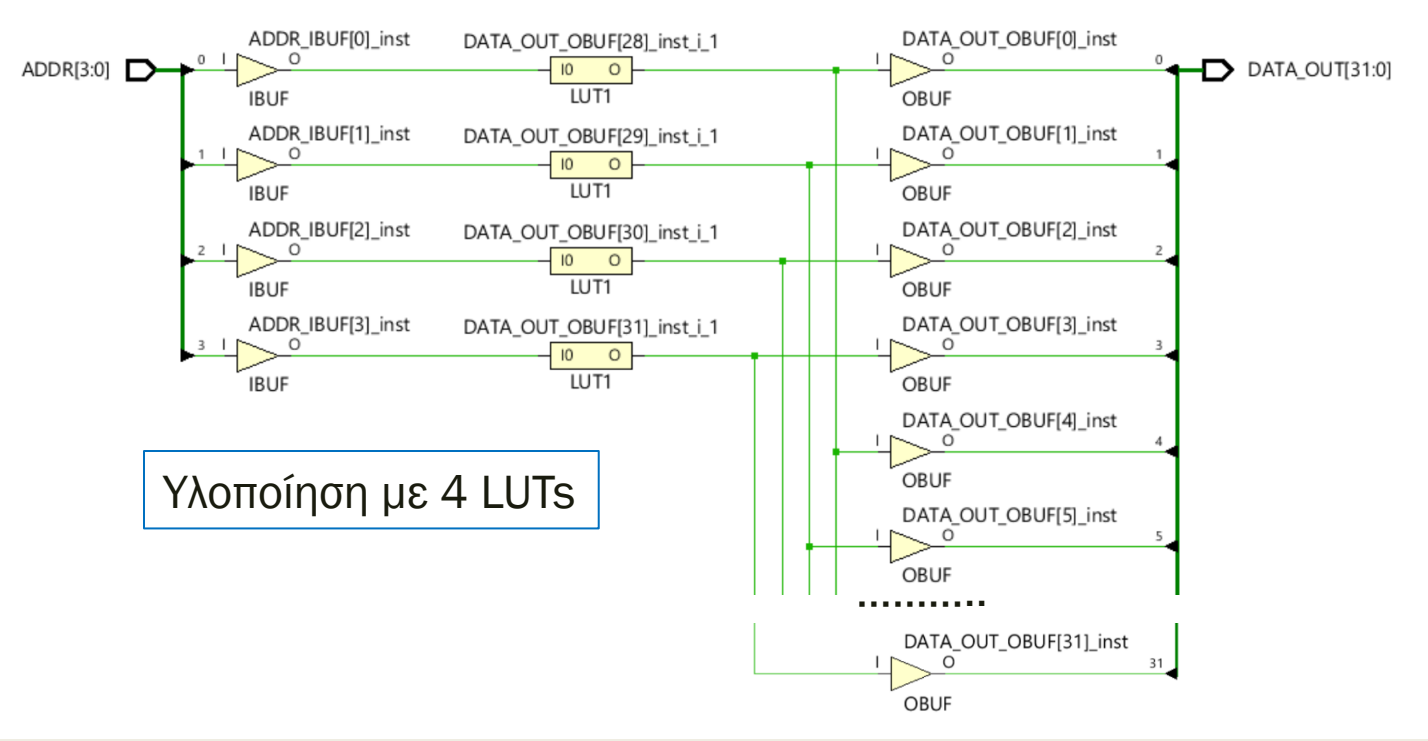
```
architecture BEHAVIORAL of RAM_array is
  type ROM_array is array (2**N-1 downto 0)
    of STD_LOGIC_VECTOR (M-1 downto 0);
  constant ROM : ROM_array := (
    X"00000000", X"11111111", X"22222222", X"33333333",
    X"44444444", X"55555555", X"66666666", X"77777777",
    X"88888888", X"99999999", X"AAAAAAAA", X"BBBBBBBB",
    X"CCCCCCCC", X"DDDDDDDD", X"EEEEEEEE", X"FFFFFFFF");
begin
  DATA_OUT <= ROM(to_integer(unsigned(ADDR)));
end BEHAVIORAL;
```

Διατάξεις μνήμης ROM στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL της διάταξης μνήμης ROM



- Σχηματικό διάγραμμα σε τεχνολογία FPGA (συνδυαστική λογική με LUTs)



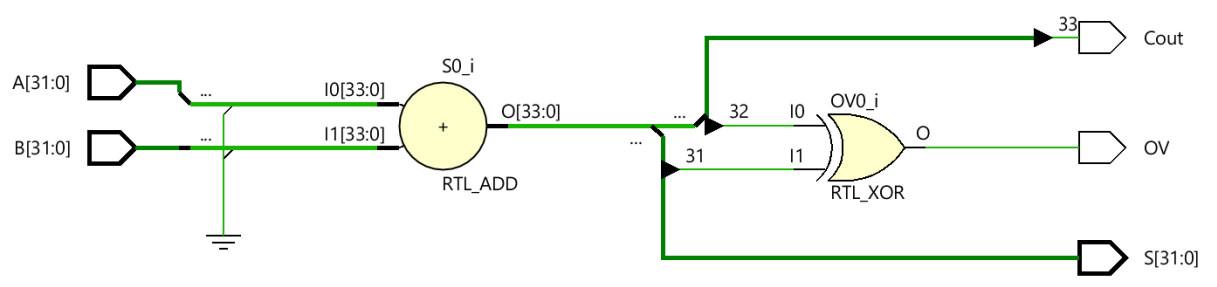
Προσημασμένος αθροιστής των 32 bit με Cout και υπερχείλιση στη VHDL – Περιγραφή συμπεριφοράς

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity ADD32sov is
  generic (WIDTH : positive := 32); -- εδώ ορίζεται η τιμή
  port (
    A : in  STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    B : in  STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    S : out STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    Cout : out STD_LOGIC;
    OV :   out STD_LOGIC);
end ADD32sov;
architecture BEHAVIORAL of ADD4sov is
begin
  ADD32sov: process (A, B)
  variable A_s, B_s, S_s: SIGNED (WIDTH+1 downto 0);
  begin
    A_s := signed('0'&A(WIDTH-1)&A);                -- numeric_std
    B_s := signed('0'&B(WIDTH-1)&B);                -- numeric_std
    S_s := A_s + B_s;                               -- numeric_std
    S <= std_logic_vector(S_s(WIDTH-1 downto 0)); -- numeric_std
    OV <= S_s(WIDTH) xor S_s(WIDTH-1);
    Cout <= S_s(WIDTH+1);
  end process;
end BEHAVIORAL;
```

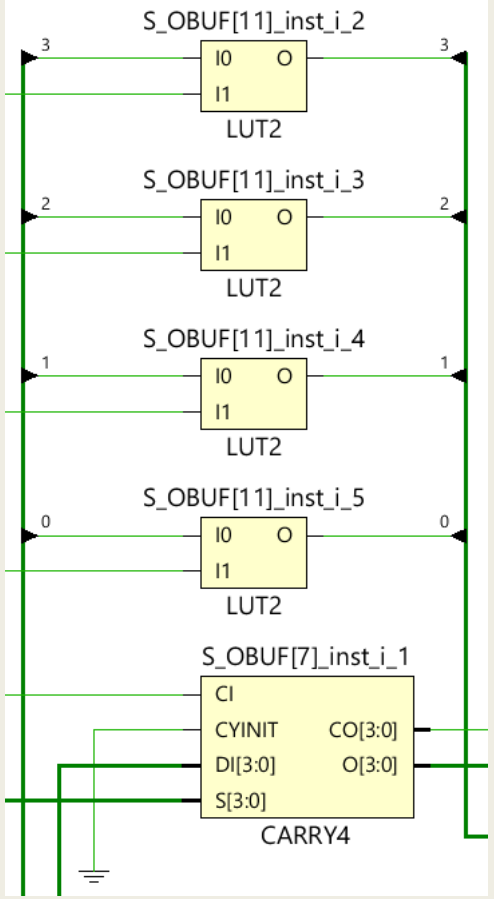
Το OV = 1 δηλώνει υπερχείλιση στην προσημασμένη πρόσθεση

Προσημασμένος αθροιστής των 32 bit με Cout και υπερχείλιση στη VHDL – Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA
  - *Κάθε 4 bit του αθροιστή υλοποιούνται ανεξάρτητα με **4 XOR** (σε LUT2) και μία μονάδα **CARRY4** για γρήγορη διάδοση κρατούμενου από slice σε slice*

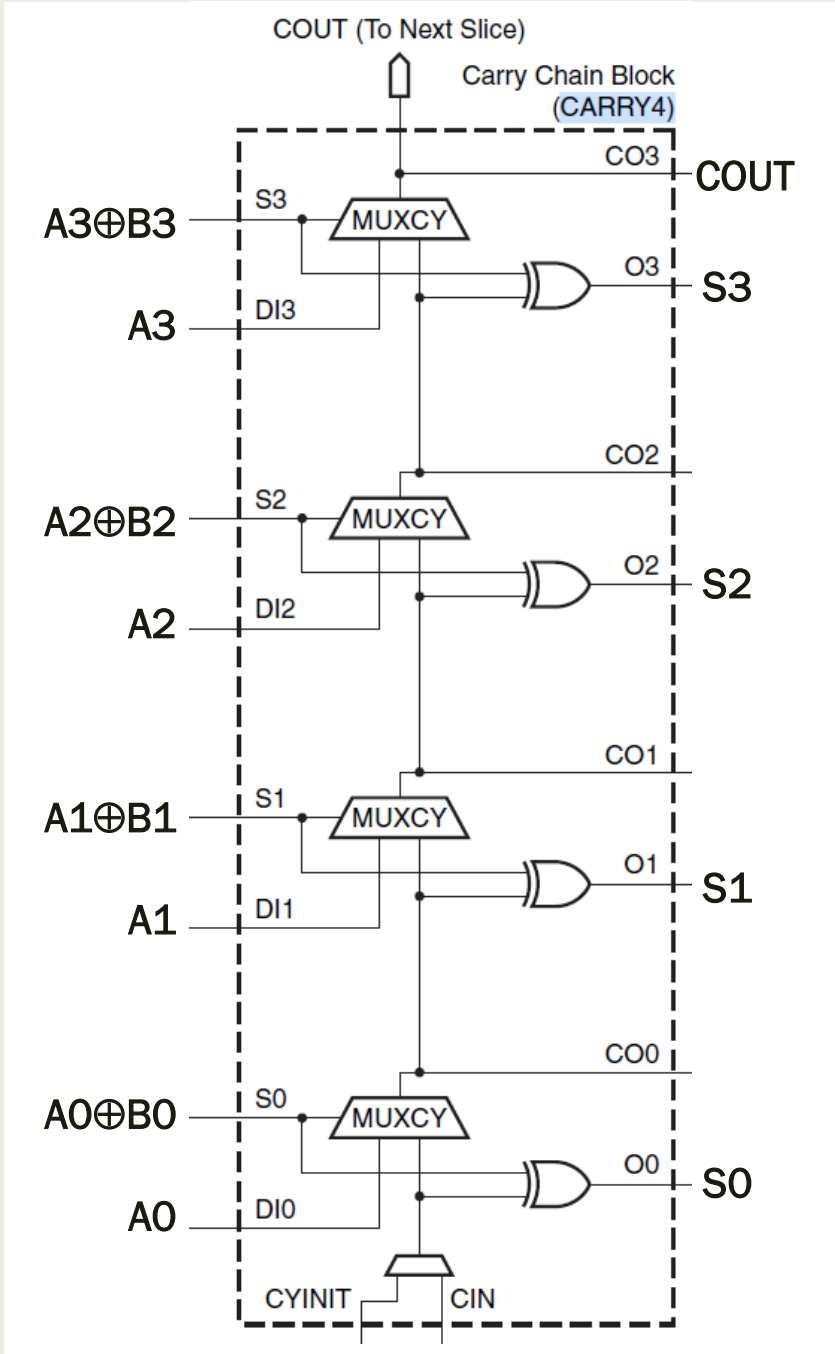


Υλοποιείται με 34 LUTs

Η μονάδα CARRY4

- Η μονάδα **CARRY4** λαμβάνει ως εισόδους:
  - *το κρατούμενο εισόδου CI*
  - *τα σήματα παραγωγής κρατούμενου S[3:0]*
    - παράγονται στα 4 LUT2
  - *τις εισόδους A[3:0] (D[3:0])*

  και παράγει ως εξόδους:
  - *τα αθροίσματα O[3:0]*
  - *τα κρατούμενα εξόδου CO[3:0]*
    - το CO[3] = COUT συνδέεται στο επόμενο CI



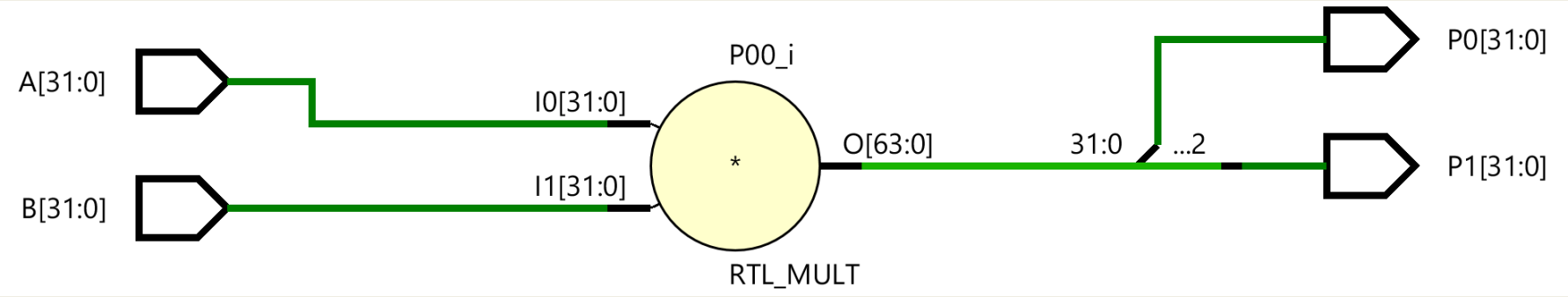
Πολλαπλασιαστής των 32 bit στη VHDL
Περιγραφή συμπεριφοράς

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity MUL32 is
  generic (WIDTH : positive := 32); -- εδώ ορίζεται η τιμή
  port (
    A:   in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    B:   in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    P0: out STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    P1: out STD_LOGIC_VECTOR (WIDTH-1 downto 0));
end MUL32;
architecture BEHAVIORAL of MUL32 is
begin
  MUL32: process (A, B)
  variable A_s: SIGNED (WIDTH-1 downto 0);
  variable B_s: SIGNED (WIDTH-1 downto 0);
  variable P_s: SIGNED (2*WIDTH-1 downto 0);
  begin
    A_s := signed(A);                                    -- numeric_std
    B_s := signed(B);                                    -- numeric_std
    P_s := A_s * B_s;                                    -- numeric_std
    P0 <= std_logic_vector(P_s(WIDTH-1 downto 0));       -- numeric_std
    P1 <= std_logic_vector(P_s(2*WIDTH-1 downto WIDTH)); -- numeric_std
  end process;
end BEHAVIORAL;
```

Προσοχή! Για την ορθή χρήση των στοιχείων DSP48E1 πρέπει να δηλώνονται προσημασμένοι αριθμοί (signed)

Πολλαπλασιαστής των 32 bit στη VHDL
Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL

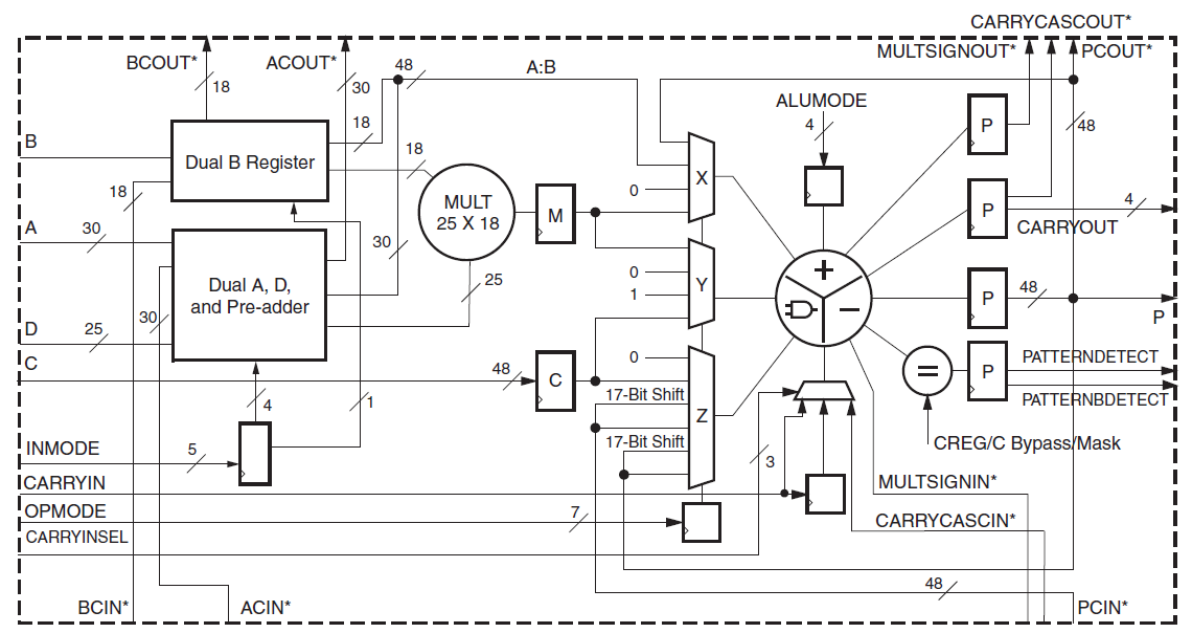


- Σχηματικό διάγραμμα σε τεχνολογία FPGA
  - *4 στοιχεία DSP48E1*
  - *αθροιστής των 48 bit αποτελούμενος από 12 αθροιστές των 4 bit*
    - 12 μονάδες CARRY4
    - 47 XOR (σε LUT2)
- Εκτελείται η πράξη:
  - A[31:0] = A[31:16] * 2^{16} + A[15:0] και B[31:0] = B[31:16] * 2^{16} + B[15:0]
  - P[63:0] = A[31:16] * B[31:16] * 2^{32} +
    (A[31:16] * B[15:0] + A[15:0] * B[31:16]) * 2^{16} +
    A[15:0] * B[15:0]

Το στοιχείο DSP48E1 στη σειρά 7 της XILINX

- Χρησιμοποιείται σε εφαρμογές ψηφιακής επεξεργασίας σήματος
- Εκτελεί πολλαπλασιασμό προσημασμένων αριθμών των 25 x 18 bit
- Εκτελεί συσσώρευση, σύγκριση και λογικές πράξεις των 48 bit
  - *Διπλή πρόσθεση, αφαίρεση, συσσώρευση στα 24 bit*
  - *Τετραπλή πρόσθεση, αφαίρεση, συσσώρευση στα 12 bit*
- Διαθέτει αθροιστή των 25 bit πριν την είσοδο του πολλαπλασιαστή
- Δυνατότητα διοχέτευσης και δεξιάς ολίσθησης των 17 bit
- Διαθέτει pattern detector για ανίχνευση συγκεκριμένης τιμής και υποστήριξη υπερχείλισης, υποχείλισης και στρογγυλοποίησης στις πράξεις πραγματικών αριθμών

UG479



Γενικές συμβουλές για σωστή σύνθεση

- Σηματοδότηση
  - *ένα **tri-state enable** σήμα ανά οντότητα, εκτός από την περίπτωση της οντότητας των αρτηριών*
  - *ένα σήμα **CLK** ανά οντότητα*
  - *ένα σήμα **RESET** ανά οντότητα*
  - *Όλα τα port signals να είναι τύπου **std_logic** και **std_logic_vector***
  - *Όλα τα port signals να είναι **in** ή **out***
  - *η πιο ψηλά στην ιεραρχία οντότητα μπορεί να έχει port signals που να είναι **αμφίδρομα (inout)***
    - Υλοποιείται με τους διαθέσιμους tri-state buffers στα IOB
- Δημιουργήστε ξεχωριστή οντότητα για τις αρτηρίες που ελέγχονται από tri-state enable σήματα
  - *ελέγξτε εάν η προγραμματιζόμενη λογική υποστηρίζει tri-state buffers, αλλιώς χρησιμοποιήστε λογική για muxs*

Σχεδιαστικές επιλογές για υλοποίηση soft processor cores σε τεχνολογία FPGA

- Ιδιαίτερη μελέτη των **κρίσιμων καθυστερήσεων διάδοσης**, που εμφανίζονται κατά τη διασύνδεση των **Block RAMs** και των **ενσωματωμένων multipliers** με τις υπόλοιπες λειτουργικές μονάδες του datapath
- Στην RTL υλοποίηση προσεκτική μελέτη της **καθυστέρησης διάδοσης από καταχωρητή σε καταχωρητή** και αύξηση των ενδιάμεσων προσωρινών καταχωρητών, όπου απαιτείται
- Βέλτιστη εκμετάλλευση των **εισόδων των LUTs**
- Ιδιαίτερη μελέτη των σημάτων που έχουν μεγάλα **fan-outs**
- Οι **Block RAMs** και οι **multipliers** πρέπει να έχουν **καταχωρητές** στην είσοδό τους και στην έξοδό τους
- Οι **multipliers** μπορεί να είναι και **pipelined**

Σχεδιαστικές επιλογές για υλοποίηση soft processor cores σε τεχνολογία FPGA

- Αποφεύγουμε την υλοποίηση **πολύπλοκης μονάδας ελέγχου**
  - *εάν το μέγεθος της μνήμης εντολών, που υλοποιείται με Block RAMs, το επιτρέπει, χρησιμοποιούμε κάποια ψηφία, επιπλέον των 32 ψηφίων της εντολής, σαν σήματα ελέγχου*
- Χρησιμοποιούμε **αθροιστές και αφαιρέτες**, όπου είναι δυνατό, γιατί είναι μία φθηνή και αποδοτική λύση για FPGA
  - *στη σύγκριση χρησιμοποιούμε **αφαιρέτη** αντί για **συγκριτή***
- Χρησιμοποιούμε τους **πολυπλέκτες** με φειδώ, γιατί είναι μία ακριβή και μη αποδοτική λύση για FPGA
  - *στην ολίσθηση για παράδειγμα μπορούμε να χρησιμοποιήσουμε υπάρχοντες **multipliers**, αντί **ολισθητή***
- Χρησιμοποιούμε τα ξεχωριστά **read και write ports** των διαθέσιμών **dual-port Block RAMs**, όπου απαιτείται
  - *υλοποιούμε το αρχείο καταχωρητών χρησιμοποιώντας **μία dual-port Block RAM 32x32**, όπου τα 2 data output ports αντιστοιχούν στα 2 read ports του αρχείου καταχωρητών*