ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Συγγραφή Εκπαιδευτικού Υλικού για το μάθημα του Τμήματος Πληροφορικής & Τηλεπικοινωνιών «Space Data Systems»

Ιωάννης Χ. Σίδερης

Επιβλέπων: **Αντώνιος Πασχάλης**, Καθηγητής

ΑΘΗΝΑ

Οκτώβριος 2020

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Συγγραφή Εκπαιδευτικού Υλικού για το μάθημα
του Τμήματος Πληροφορικής & Τηλεπικοινωνιών
«Space Data Systems»

Ιωάννης Χ. Σίδερης
Α.Μ.: 1115201200227

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Αντώνιος Πασχάλης,** Καθηγητής

ΠΕΡΙΛΗΨΗ

Ο σκοπός της παρούσας πτυχιακής εργασίας είναι η συγγραφή ενός αναλυτικού εργαστηριακού οδηγού για την σχεδίαση και υλοποίηση ενός πραγματικού επεξεργαστή της οικογένειας ARM σε FPGA χρησιμοποιώντας το σχεδιαστικό εργαλείο Vivado IDE της εταιρίας Xilinx. Η εργασία είναι χωρισμένη σε δύο μέρη:

1. Το πρώτο μέρος, που είναι και με διαφορά το εκτενέστερο, ασχολείται με την εκμάθηση του σχεδιαστικού εργαλείου Vivado μέσα από δύο απλά αλλά αντιπροσωπευτικά ψηφιακά κυκλώματα (ένα κύκλωμα αθροιστή με καταχωρητές και μία FSM). Για κάθε κύκλωμα αναπτύσσεται ο τρόπος περιγραφής του σε VHDL, η διαδικασία σύνθεσης και η υλοποίησής του καθώς και η διαδικασία λογικής και χρονικής προσομοίωσης σε όλα τα στάδια της σχεδίασης.
2. Το δεύτερο μέρος πραγματεύεται τις σχεδιαστικές απαιτήσεις ενός επεξεργαστή ενός κύκλου της οικογένειας ARM. Για εκπαιδευτικούς λόγους, το υποστηριζόμενο ρεπερτόριο εντολών είναι περιορισμένο πλην επαρκές ώστε ο επεξεργαστής να μπορεί να χρησιμοποιηθεί σε μία πληθώρα εφαρμογών.

Ο εργαστηριακός οδηγός συνοδεύεται επίσης από έναν μεγάλο αριθμό screenshots παρμένων από το σχεδιαστικό περιβάλλον που δείχνουν όλα τα επί μέρους βήματα που περιγράφονται στο κείμενο, ώστε ο φοιτητής που το μελετά να είναι πλήρως κατατοπισμένος για όλα τα στάδια της διαδικασίας σχεδίασης.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Ψηφιακή Σχεδίαση, Σχεδίαση Επεξεργαστή

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: VHDL, Xilinx, Vivado, Σχεδίαση Υλικού, Εκπαίδευση

ABSTRACT

The purpose of this thesis is the writing of a comprehensive practical guide concerning the design and implementation of a real processor of the ARM family in FPGA using the Xilinx Vivado IDE design tool. The thesis is split into two parts:

1. The first part, which is by far the most extensive, is a tutorial of the Vivado design tool through two simple but representative digital circuits (one involving an adder and some registers and the other an FSM). For each digital circuit, it is described its coding style in VHDL, its synthesis and implementation processes and also the processes of the logical and timing simulations in every stage of the design.
2. The second part discusses the design requirements of a single cycle processor of the ARM family. For educational reasons, the supported instruction set is limited, but is enough so that the processor can be used in a wide range of applications.

The practical guide also contains a large number of screenshots taken from the design environment that display all the steps described in the text. This way, the student who studies it is fully aware of every stage of the design process.

SUBJECT AREA: Digital Design, Processor Design

KEYWORDS: VHDL, Xilinx, Vivado, Hardware Design, Education

Αυτή η πτυχιακή εργασία αφιερώνεται στην οικογένειά μου.

ΕΥΧΑΡΙΣΤΙΕΣ

Η πτυχιακή εργασία δεν θα ήταν δυνατόν να πραγματοποιηθεί στη παρούσα τελική μορφή της χωρίς την πολύτιμη βοήθεια του επιβλέποντα καθηγητή και άλλων πολύτιμων συνεργατών. Για αυτό, ο συγγραφέας αισθάνεται την ανάγκη να ευχαριστήσει τα ακόλουθα πρόσωπα:

- Τον επιβλέποντα καθηγητή κ. Αντώνιο Πασχάλη για την συνεχή υποστήριξη και άριστη συνεργασία καθ' όλο το χρονικό διάστημα της εργασίας.
- Τον δρ. Νεκτάριο Κρανίτη για τις εποικοδομητικές παρατηρήσεις του.
- Τον δρ. Διονύσιο Βασιλόπουλο για τις χρήσιμες συμβουλές του.

Τέλος, ο συγγραφέας αποδίδει την βαθιά του ευγνωμοσύνη και τις θερμότερες ευχαριστίες προς τον παντοδύναμο Τριαδικό Θεό, χωρίς την βοήθεια του Οποίου κανένα καλό έργο δεν έρχεται εις πέρας.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΠΡΟΛΟΓΟΣ

Η δημιουργία ενός σύγχρονου πανεπιστημιακού υλικού αποτελεί μία δύσκολη διαδικασία. Αφενός μεν απαιτεί μια εις βάθος γνώση του γνωστικού αντικειμένου από τον συγγραφέα και αφετέρου απαιτεί ιδιαίτερη ικανότητα στον τρόπο παρουσίασης ώστε το αποτέλεσμα να είναι εύληπτο και προσιτό από τους φοιτητές. Η δυσκολία αυξάνεται περισσότερο όταν το καθ' εαυτό αντικείμενο είναι σύνθετο και παρουσιάζει αρκετές προκλήσεις. Για αυτούς τους λόγους, η ανάγκη για ένα διεξοδικό υλικό διδασκαλίας καθίσταται αναγκαία, διότι σε συνδυασμό με την σωστή διδασκαλία, συμβάλλει καθοριστικά στο να ελκύσει το ενδιαφέρον του φοιτητή για το γνωστικό αντικείμενο και να βοηθήσει στην αποτελεσματικότερη εκμάθησή του. Ένας ακόμα πιο φιλόδοξος στόχος είναι να κάνει τον φοιτητή να αγαπήσει το αντικείμενο και μελλοντικά να ασχοληθεί με αυτό.

Στην παρούσα πτυχιακή εργασία παρουσιάζεται ένας ολοκληρωμένος εργαστηριακός οδηγός που αφορά το χώρο της ψηφιακής σχεδίασης και συγκεκριμένα την σχεδίαση επεξεργαστών με σύγχρονα σχεδιαστικά εργαλεία λογισμικού. Έχει καταβληθεί κάθε δυνατή προσπάθεια ώστε το παρόν εκπαιδευτικό υλικό να εκπληρώνει τους προαναφερθέντες στόχους. Η επιθυμία του συγγραφέα είναι αφενός μεν να ικανοποιήσεις κατά πάντα την φοιτητική κοινότητα που ενδιαφέρεται για το συγκεκριμένο γνωστικό αντικείμενο και αφετέρου να αποτελέσει ένα ισχυρό έναυσμα γνωριμίας του αντικειμένου από φοιτητές που δεν προσανατολίζονται προς αυτή τη κατεύθυνση.

ΕΙΣΑΓΩΓΗ

Η παρούσα πτυχιακή εργασία αποτελεί τον εργαστηριακό οδηγό του μαθήματος Space Data Systems του τμήματος Πληροφορικής και Τηλεπικοινωνιών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών. Αυτό το μάθημα αφορά τόσο τους προπτυχιακούς φοιτητές του τμήματος όσο και τους μεταπτυχιακούς φοιτητές σε προγράμματα μεταπτυχιακών σπουδών στα οποία συμμετέχει το τμήμα.

Το εργαστηριακό υλικό χωρίζεται σε δύο επί μέρους αυτοτελείς εργαστηριακές ασκήσεις:

1. Η πρώτη εργαστηριακή άσκηση είναι η εκτενέστερη όλων. Πρόκειται για ένα αναλυτικό tutorial του σχεδιαστικού εργαλείου Vivado IDE της εταιρίας Xilinx. Στις σελίδες του tutorial περιγράφεται λεπτομερώς βήμα–προς–βήμα η διαδικασία σχεδίασης ενός ψηφιακού συστήματος ξεκινώντας από την δημιουργία ενός νέου project και καταλήγοντας στην επαλήθευση ορθής λειτουργίας μέσω προσομοίωσης. Το tutorial δίνει ιδιαίτερη έμφαση στη μελέτη των αποτελεσμάτων των διαδικασιών της σύνθεσης (synthesis) και της υλοποίησης (implementation) καθώς και της λογικής και χρονικής προσομοίωσης. Για καλύτερη κατανόηση της ροής σχεδίασης, χρησιμοποιούνται ως παραδείγματα δύο απλά πλην αντιπροσωπευτικά ακολουθιακά ψηφιακά κυκλώματα: ένα κύκλωμα αθροιστή με καταχωρητές και μία μηχανή πεπερασμένων καταστάσεων (FSM). Η διαδικασία της σχεδίασης περιγράφεται ξεχωριστά σε όλα της τα στάδια για κάθε ένα κύκλωμα.
2. Η δεύτερη εργαστηριακή άσκηση πραγματεύεται τη σχεδίαση ενός επεξεργαστή ARM ενός κύκλου. Στις σελίδες της περιγράφονται λεπτομερώς όλες οι σχεδιαστικές απαιτήσεις του επεξεργαστή, που περιλαμβάνουν τη γενική αρχιτεκτονική, τους πίνακες αλήθειας των σημάτων ελέγχου και μία συνοπτική πλην πλήρη παρουσίαση κάθε εντολής του υποστηριζόμενου ρεπερτορίου εντολών που πρέπει να υλοποιηθεί. Επιπλέον, παρέχονται οδηγίες και σχεδιαστικές συμβουλές για το πως θα πρέπει οι επιμέρους υπο–μονάδες του επεξεργαστή να «κουμπώσουν» μεταξύ τους συνθέτοντας έτσι τη διαδρομή δεδομένων καθώς και αντίστοιχες οδηγίες για τη μονάδα ελέγχου. Ο τρόπος που παρουσιάζονται όλες αυτές οι πληροφορίες είναι τέτοιος ώστε άπαξ και ο φοιτητής έχει βασική γνώση της θεωρίας γύρω από τον επεξεργαστή ARM να μπορεί να δουλέψει μόνο με τον εργαστηριακό οδηγό χωρίς να ανατρέχει εκ νέου στη θεωρία. Η άσκηση συνεχίζει με τις οδηγίες για την επαλήθευση της ορθής λειτουργίας τόσο των επί μέρους βασικών μονάδων της όσο και ολόκληρου του επεξεργαστή. Στο τέλος, παρατίθενται οι απαιτήσεις της τεχνικής αναφοράς που πρέπει να συνοδεύουν τον υλοποιημένο πλέον επεξεργαστή.

Όλες οι εργαστηριακές ασκήσεις συνοδεύονται από πληθώρα screenshots που δείχνουν τα περιγραφόμενα βήματα αναλυτικά. Με αυτό τον τρόπο ο φοιτητής ξέρει τι να περιμένει σε κάθε βήμα της σχεδίασης. Έτσι η εκμάθηση της σχεδιαστικής ροής καθίσταται ευκολότερη και επιπλέον μπορεί να μελετηθεί σε ικανοποιητικό βαθμό ακόμα και χωρίς την άμεση επαφή με το σχεδιαστικό εργαλείο.

1. Μέρος Α΄: Οδηγός χρήσης των βασικών λειτουργιών του Vivado IDE της XILINX

1.1 Εισαγωγικά στοιχεία

Στον παρόντα οδηγό χρήσης θα παρουσιάσουμε αναλυτικά τις βασικές λειτουργίες του εργαλείου **Vivado IDE** (Integrated Development Environment) της Xilinx μέσα από τη σχεδίαση απλών ψηφιακών κυκλωμάτων (συνδυαστικών και ακολουθιακών) με τη χρήση της γλώσσας περιγραφής υλικού VHDL και την υλοποίησή τους στην αναπτυξιακή κάρτα **ZedBoard**, που διαθέτει τη διάταξη FPGA της XILINX **Zynq XC7Z020–1CLG484** All Programmable SoC (AP SoC).

Αρχικά, θα δούμε τα βασικά βήματα της εισαγωγής του κώδικα περιγραφής της σχεδίασης (design entry), της εισαγωγής του προγράμματος δοκιμής (testbench entry), της προσομοίωσης (simulation) για την επαλήθευση της ορθής σχεδίασης σύμφωνα με τις προδιαγραφές, της σύνθεσης (synthesis) και της υλοποίησης (implementation) που απαιτούνται για τη σχεδίαση ενός ψηφιακού κυκλώματος, χρησιμοποιώντας τις προκαθορισμένες ρυθμίσεις και αναλύοντας τα αποτελέσματα που προκύπτουν σε κάθε βήμα, χωριστά. Αυτά είναι και τα βήματα που θα ακολουθήσετε κατά τη σχεδίαση του επεξεργαστή στο πλαίσιο του μαθήματος.

Τέλος, θα δούμε πώς παράγεται το bitstream, ώστε να μπορεί να «κατέβει» η σχεδίαση στη διάταξη FPGA της αναπτυξιακής κάρτας Zedboard για την τελική δοκιμή της λειτουργίας του ψηφιακού μας κυκλώματος. Το τελευταίο στάδιο είναι προαιρετικό υπό τις παρούσες συνθήκες.

1.2 Μαθησιακοί στόχοι

Με την εκμάθηση του οδηγού χρήσης θα είστε σε θέση να:

- δημιουργείτε ένα **Project** στο Vivado IDE και να δηλώστε ως target device, τη διάταξη Zynq που βρίσκεται στην αναπτυξιακή κάρτα ZedBoard,
- δημιουργείτε ένα κατάλληλο αρχείο τύπου **Xilinx Design Constraint (XDC)** για τη δήλωση του σήματος του ρολογιού (CLK), του σήματος RESET, των χρονικών περιορισμών (timing constraints), καθώς και φυσικών περιορισμών (physical constraints) που σχετίζονται με τη χρησιμοποιούμενη αναπτυξιακή κάρτα, όπως οι αντιστοιχήσεις των χρησιμοποιούμενων ακροδεκτών,
- εισάγεται τον **κώδικα περιγραφής υλικού** σε γλώσσα VHDL ενός συνδυαστικού ή ακολουθιακού ψηφιακού κυκλώματος (δημιουργία αρχείων τύπου .vhd), ώστε το εργαλείο Vivado IDE να παράγει το **behavioral (elaborated design) model**, που προκύπτει μετά την ανάλυση στο επίπεδο RTL του κώδικα VHDL,
- εισάγετε το κατάλληλο **πρόγραμμα δοκιμής (testbench)** σε γλώσσα VHDL (δημιουργία αρχείων τύπου .vhd), ώστε το εργαλείο Vivado IDE να δύναται να προχωρήσει στα στάδια της προσομοίωσης.
- εκτελείτε **όλα τα στάδια της προσομοίωσης** (αναφέρονται αναλυτικά στη συνέχεια) με τον Simulator XSIM που διατίθεται μέσω του εργαλείου Vivado IDE,

- εκτελείτε τη διαδικασία της **σύνθεσης**, ώστε το εργαλείο Vivado IDE να παράγει το **synthesized design model**, που προκύπτει μετά τη σύνθεση του κώδικα VHDL, και να αναλύεται τα αποτελέσματα που προκύπτουν στο στάδιο της σύνθεσης,
- εκτελείτε τη διαδικασία της **υλοποίησης**, ώστε το εργαλείο Vivado IDE να παράγει το **implemented design model**, που προκύπτει μετά την υλοποίηση σε συγκεκριμένη τεχνολογία FPGA του synthesized design model, και να αναλύεται τα αποτελέσματα που προκύπτουν στο στάδιο της υλοποίησης,
- παράγετε το **αρχείο προγραμματισμού (bitstream)** της διάταξης FPGA και προγραμματίζετε τη **διάταξη Zynq** που βρίσκεται στην αναπτυξιακή κάρτα ZedBoard χρησιμοποιώντας το αρχείο bitstream (απαιτείται πρόσβαση στην κάρτα).

1.3 Πιθανές προσομοιώσεις μέσω του εργαλείου Vivado IDE

Μετά τη δημιουργία του κατάλληλου **προγράμματος δοκιμής (testbench)** στη γλώσσα VHDL δύναστε να εκτελέσετε τις ακόλουθες πιθανές προσομοιώσεις:

Behavioral simulation

Λογική προσομοίωση που εφαρμόζεται στο **behavioral (elaborated design) model** που προκύπτει μετά την ανάλυση στο επίπεδο RTL του κώδικα VHDL που έχετε εισάγει.

Post synthesis functional simulation

Λογική προσομοίωση που εφαρμόζεται στο **synthesized design model** που προκύπτει μετά τη σύνθεση του κώδικα VHDL. (Προσοχή! πρέπει να ταυτίζεται με το behavioral simulation).

Post synthesis timing simulation

Χρονική προσομοίωση που εφαρμόζεται στο **synthesized design model** που προκύπτει μετά τη σύνθεση του κώδικα VHDL.

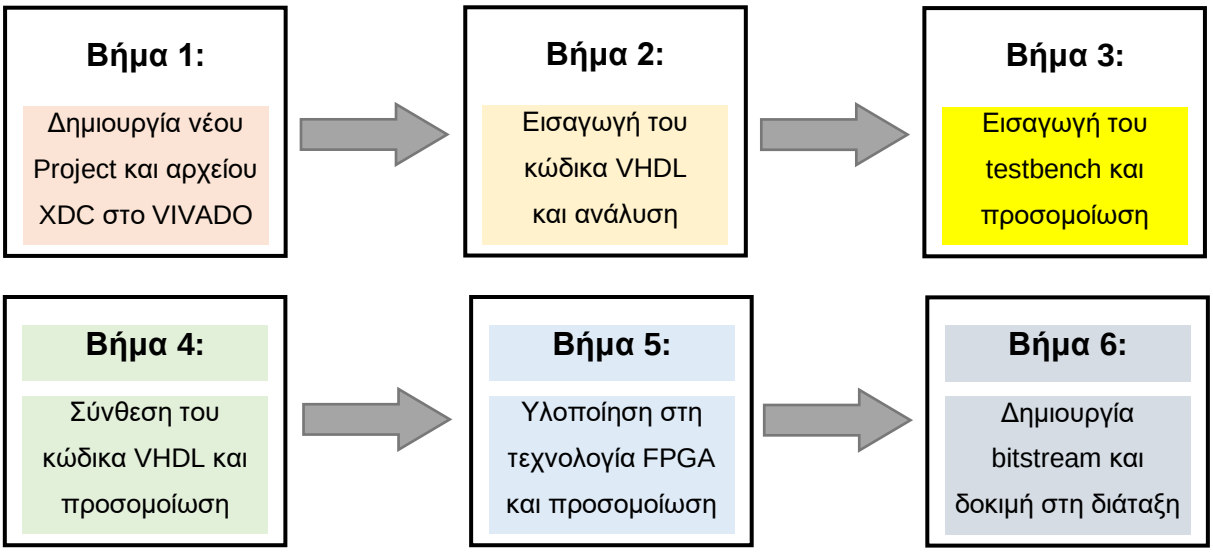
Post implementation functional simulation

Λογική προσομοίωση που εφαρμόζεται στο **implemented design model** που προκύπτει μετά την υλοποίηση σε συγκεκριμένη τεχνολογία FPGA του synthesized design model. (Προσοχή! πρέπει να ταυτίζεται με το behavioral simulation).

Post synthesis timing simulation

Χρονική προσομοίωση που εφαρμόζεται στο **implemented design model** που προκύπτει μετά την υλοποίηση σε συγκεκριμένη τεχνολογία FPGA του synthesized design model.

1.4 Γενική ροή σχεδίασης και υλοποίησης ενός ψηφιακού κυκλώματος



Το Βήμα 6 θα περιγραφθεί σε επόμενη έκδοση του οδηγού.

1.5 Ψηφιακά κυκλώματα που θα σχεδιάσουμε και θα υλοποιήσουμε

Στην παρούσα εργαστηριακή άσκηση θα επικεντρωθούμε στα ακόλουθα χαρακτηριστικά ψηφιακά κυκλώματα:

1. Παραμετροποιημένος καταχωρητής με RESET και WE των N bit.
2. Παραμετροποιημένος αθροιστής με Cout και OV των N bit.
3. Αθροιστής των 8 bit με καταχωρητές εισόδου και εξόδου ως σύγχρονο ακολουθιακό κύκλωμα.
4. Ανιχνευτής ακολουθίας 2 διαδοχικών bit ως μηχανή πεπερασμένων καταστάσεων (FSM) τύπου Moore.

1.6 Βήμα 1: Δημιουργία νέου Project και αρχείου τύπου XDC στο VIVADO IDE

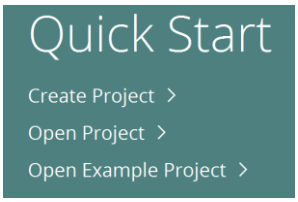
Σημείωση: Οι οδηγίες για τη δημιουργία του νέου project εμφανίζουν μικρές διαφοροποιήσεις ανάλογα με το λειτουργικό σύστημα στο οποίο έχουμε εγκαταστήσει το VIVADO IDE (Linux ή MS Windows).

1.6.1 Εκτέλεση της εφαρμογής VIVADO IDE

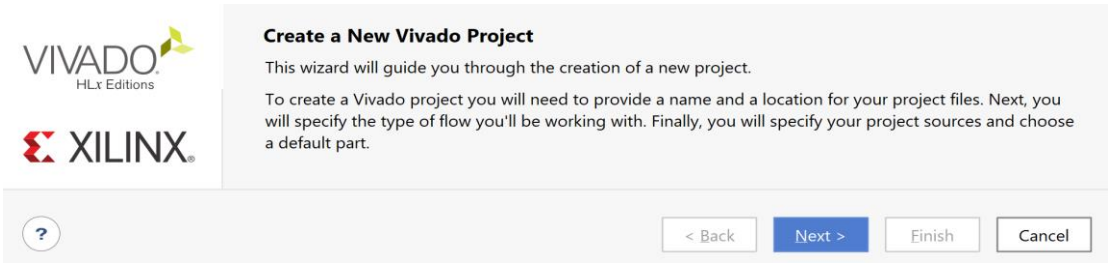
- Ανοίξτε το Vivado πατώντας στο **εικονίδιο του Vivado** που είναι διαθέσιμο στο Desktop ή εμφανίζεται όταν αναζητήσετε την εφαρμογή “Vivado”. Η έκδοση πιθανώς να διαφέρει (π.χ. Vivado 2019.2).

1.6.2 Δημιουργία ενός νέου project στο VIVADO IDE

- Πατήστε **Create Project** στο παράθυρο *Quick Start* για να ξεκινήσετε τον wizard δημιουργίας ενός νέου project. Εάν το project έχει ήδη δημιουργηθεί μπορούμε να το επιλέξουμε με το Project_name είτε από το **Open Project** στο παράθυρο *Quick Start*, είτε από το παράθυρο *Recent Project*.

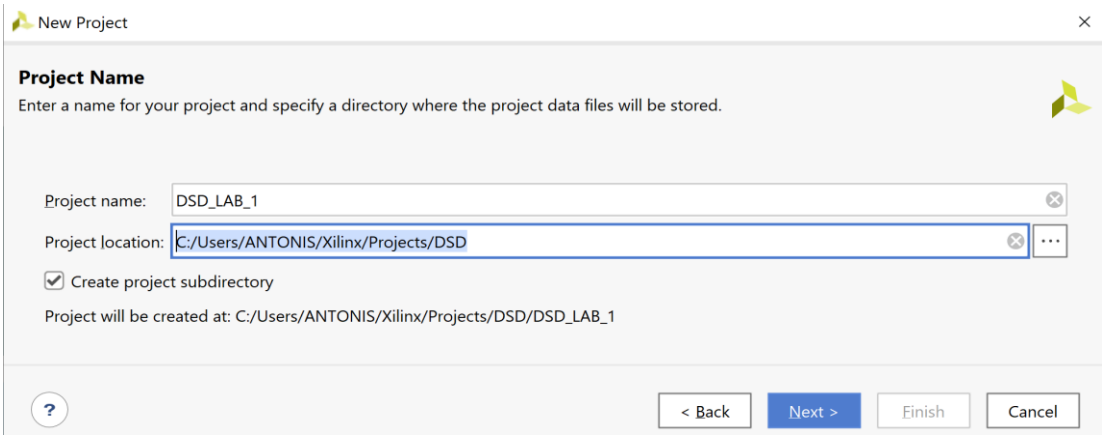


- Θα δείτε ένα παράθυρο διαλόγου *Create a New Vivado Project*. Πατήστε **Next**. Το κυκλάκι με το ερωτηματικό παρέχει τις απαραίτητες οδηγίες (είναι πολύ χρήσιμο εάν έχετε απορίες).



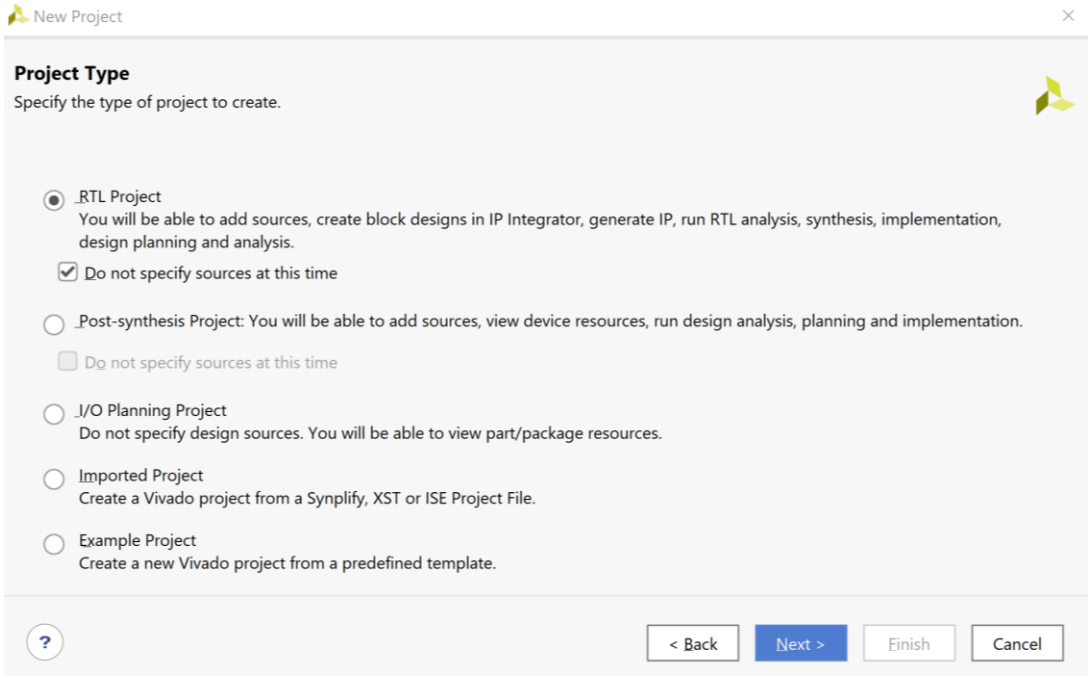
Εικόνα 1 – Αρχικό παράθυρο δημιουργίας νέου project

- Στο παράθυρο διαλόγου *Project name* πατήστε το κουτάκι με τις τελίτσες του πεδίου **Project location**, ώστε να κάνετε browse και να δημιουργήσετε τον φάκελο που θα τοποθετήσετε το project σας (π.χ. C:/Users/ANTONIS/Xilinx/Projects/DSD), και πατήστε **Select**.
- Στη συνέχεια, δηλώστε το Project_name (π.χ. DSD_LAB_1) στο πεδίο **Project name**. Βεβαιωθείτε ότι το κουτί **Create Project Subdirectory box** είναι επιλεγμένο. Πατήστε **Next**.



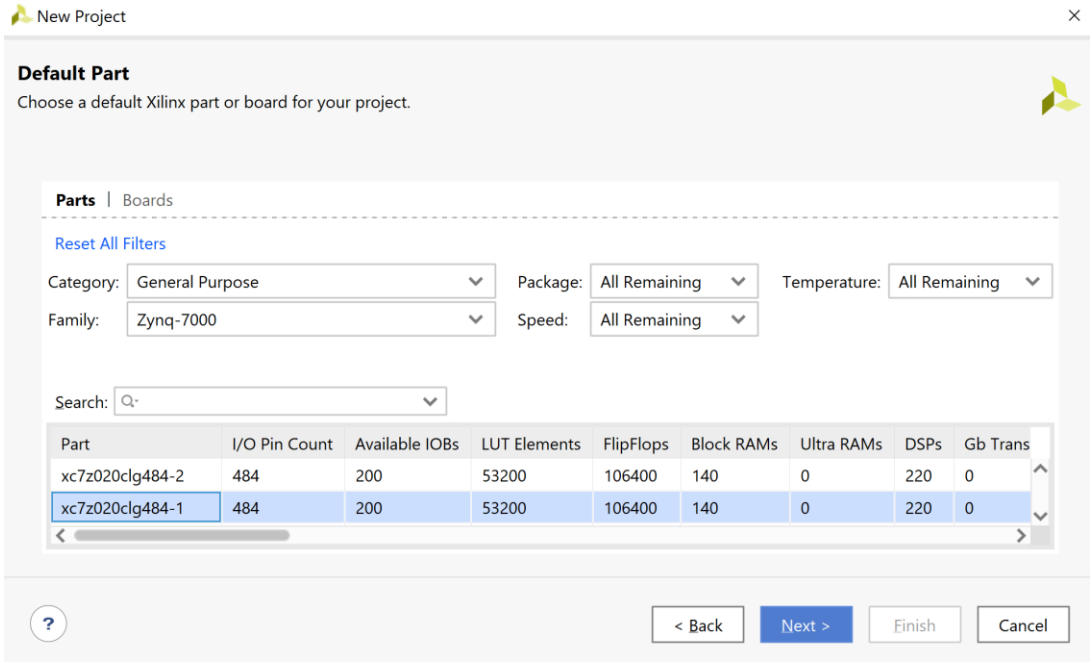
Εικόνα 2 – Ορισμός ονόματος και μονοπατιού για το νέο project

- Στο παράθυρο διαλόγου *Project Type* πατήστε την επιλογή **RTL Project**. Βεβαιωθείτε ότι το κουτί **Do not specify sources at this time** είναι επιλεγμένο, ώστε να μην εισάγουμε πηγαία αρχεία VHDL κατά τη διάρκεια της δημιουργίας του project. Πατήστε **Next**.



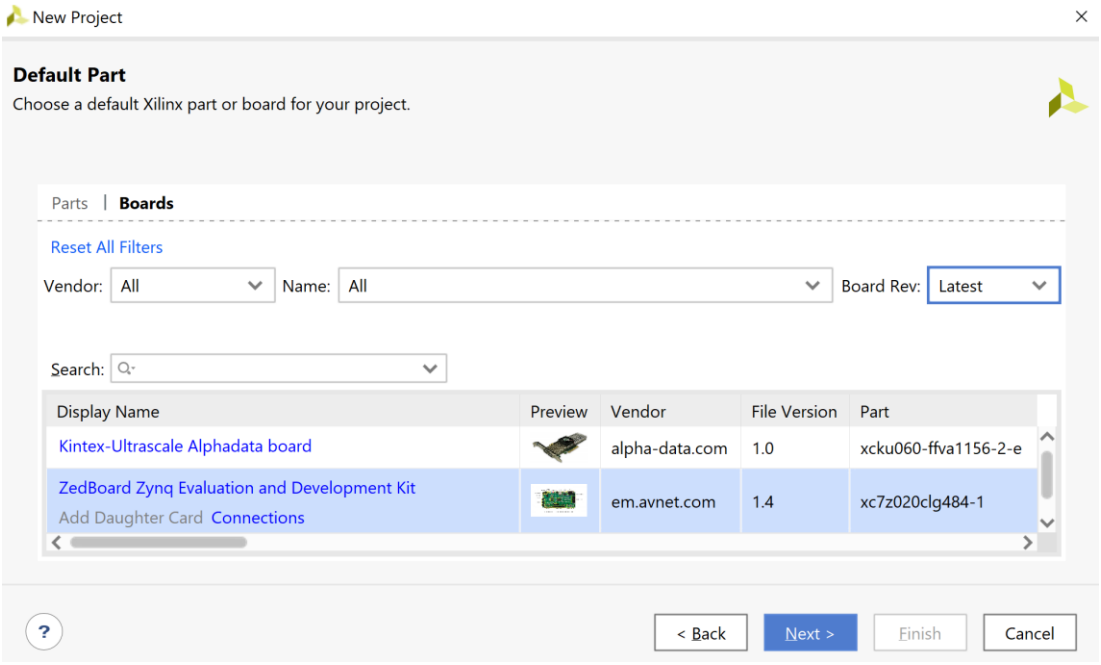
Εικόνα 3 – Ορισμός τύπου για το νέο project

- Στο παράθυρο διαλόγου *Default Part* κρατήστε την επιλογή **Parts** και επιλέξτε στο **Category** (π.χ. General Purpose) και στο **Familly** (π.χ. Zynq–7000). Στη συνέχεια επιλέξτε το Part **xc7z020clg484–1** (Zynq–7020, με 484 ακροδέκτες και ταχύτητα (speed grade) –1 (το πιο αργό της οικογένειας) που βρίσκεται στην αναπτυξιακή κάρτα ZedBoard. Παρατηρήστε τα διαθέσιμα resources του επιλεγμένου part (IOB = 200, LUT Elements = 53.200, Flip–Flops = 106.400 (διπλάσια από τα LUT Elements) Block RAMs = 140 και DSPs = 220). Πατήστε **Next**.



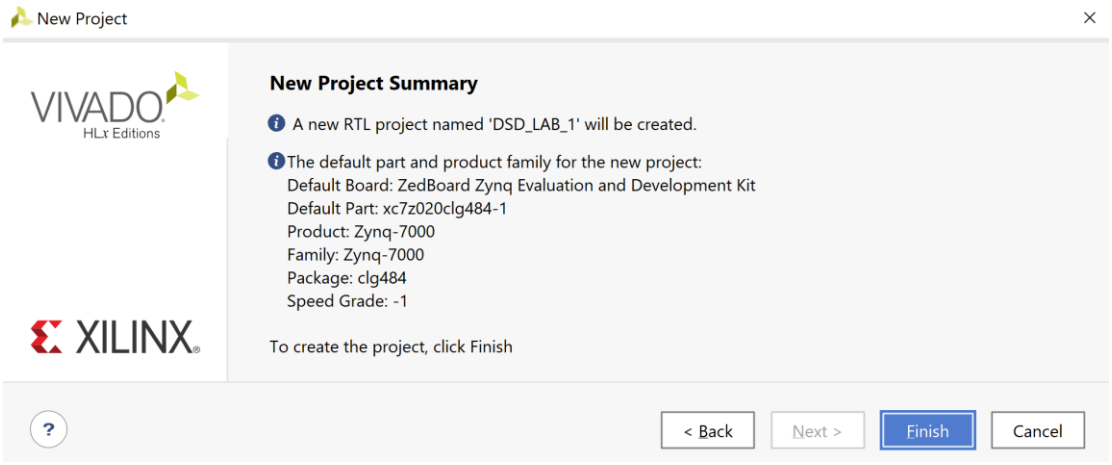
Εικόνα 4 – Ορισμός FPGA για το νέο project

- Εναλλακτικά, στο παράθυρο διαλόγου *Default Part* πατήστε την επιλογή **Boards** και επιλέξτε την αναπτυξιακή κάρτα **ZedBoard**. Πατήστε **Next**.



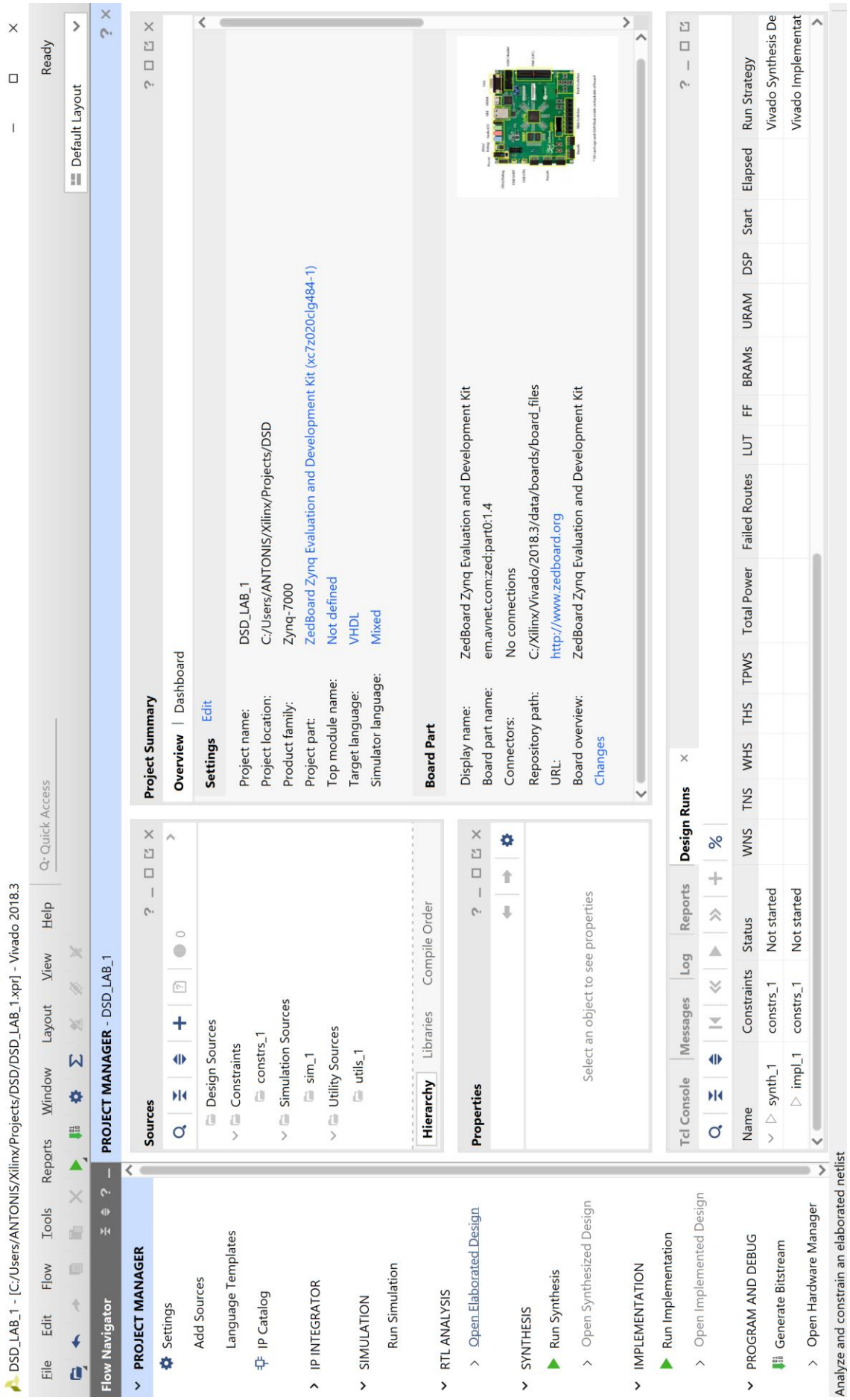
Εικόνα 5 – Ορισμός αναπτυξιακής κάρτας για το νέο project

- Στο παράθυρο διαλόγου *New Project Summary* βλέπετε το project_name και τα χαρακτηριστικά της επιλεγμένης (default) διάταξης FPGA. Πατήστε **Finish** για να δημιουργηθεί το νέο σας project που ονομάζεται **DSD_LAB_1**.



Εικόνα 6 – Σύνοψη επιλογών πριν τη δημιουργία του νέου project

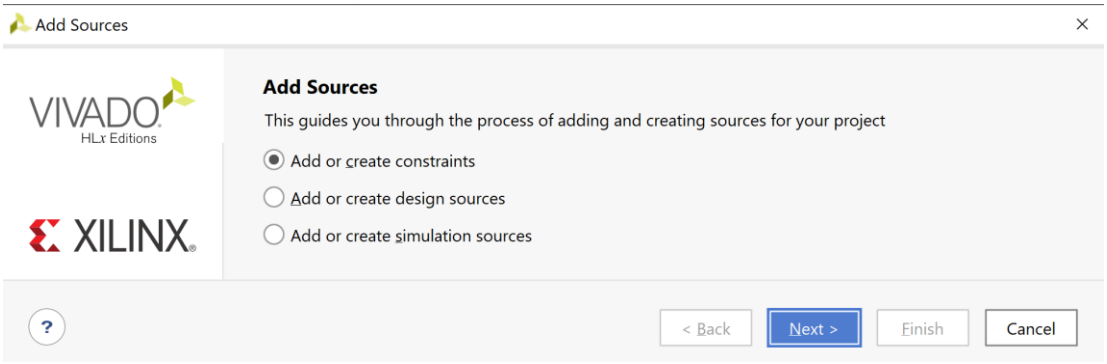
- Στην επόμενη σελίδα φαίνεται το περιβάλλον του **PROJECT MANAGER** πάνω στο οποίο θα σχεδιάσουμε τα ψηφιακά κυκλώματά μας. Αναγνωρίζουμε την οριζόντια μπάρα επάνω με επιλογές File, Edit, Flow, Tools, Reports, Window, Layout, View και Help. Το κατακόρυφο παράθυρο αριστερά του Flow Navigator, όπου με κατάλληλη επιλογή εκτελούνται όλα τα βήματα της ψηφιακής σχεδίασης. Το παράθυρο Sources, όπου φαίνονται όλα τα πηγαία αρχεία του project διαρθρωμένα στα directories: Design Sources, Constraints, Simulation Sources και Utility Sources (αρχικά είναι άδειο). Το παράθυρο Properties. Το παράθυρο Project Summary (σε κάθε βήμα της ψηφιακής σχεδίασης διαφοροποιείται κατάλληλα). Τέλος, το κάτω οριζόντιο παράθυρο πολλαπλών χρήσεων, που χρησιμεύει μεταξύ άλλων ως Tcl Console και για την ανάλυση των αποτελεσμάτων της σύνθεσης και της υλοποίησης.



Εικόνα 7 – Αρχικό παράθυρο του Project Manager

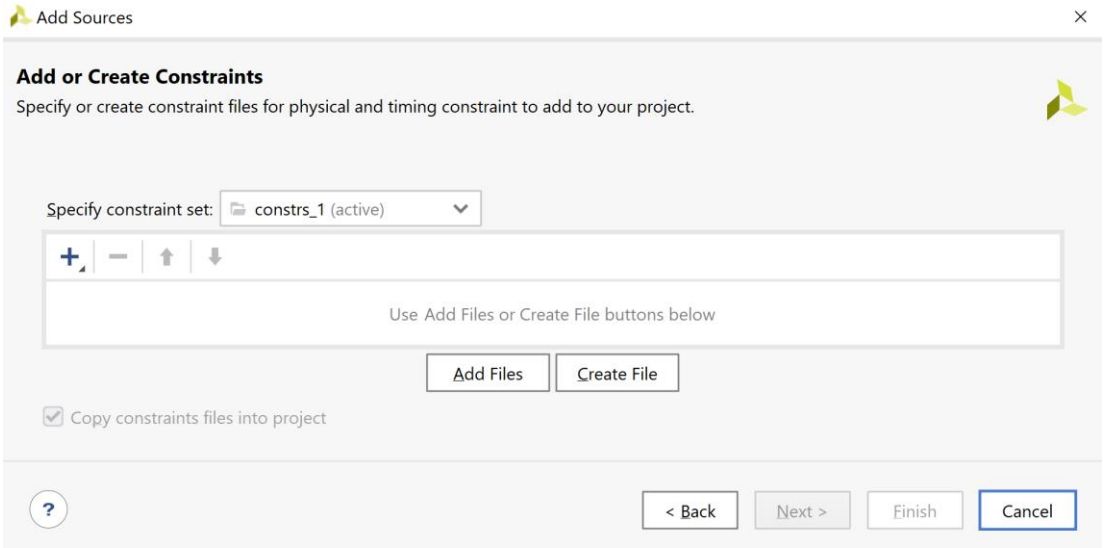
1.6.3 Δημιουργία του αρχείου τύπου XDC στο VIVADO IDE

- Πατήστε το **+** στο παράθυρο *Sources* του PROJECT MANAGER για να ξεκινήσετε τον wizard δημιουργίας ενός νέου αρχείου (source).
- Στο παράθυρο διαλόγου *Add Sources* επιλέξτε το Add or create constraints, ώστε να δημιουργήσετε ένα αρχείο περιγραφής περιορισμών (constraint file) τύπου XDC. Σε αυτό το αρχείο, που αρχικά είναι άδειο, δηλώνουμε το σήμα του ρολογιού (CLK), το σήμα RESET, τους χρονικούς περιορισμούς (timing constraints), καθώς και τους φυσικούς περιορισμούς (physical constraints) που σχετίζονται με τη χρησιμοποιούμενη αναπτυξιακή κάρτα, όπως οι αντιστοιχίσεις των χρησιμοποιούμενων ακροδεκτών. Πατήστε **Next**.



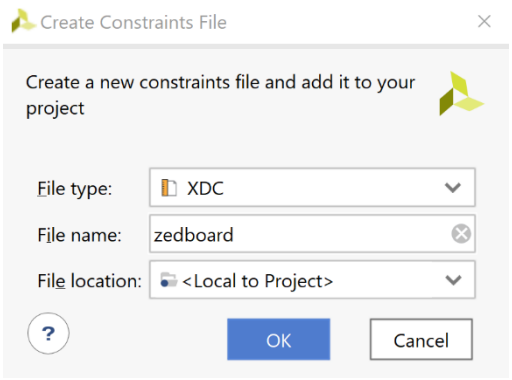
Εικόνα 8 – Πρώτο βήμα για τη δημιουργία constraints

- Στο παράθυρο διαλόγου *Add or Create Constraints* πατήστε την επιλογή **Create File** για τη δημιουργία του άδειου αρχείου τύπου XDC, που θα τοποθετηθεί στο ήδη καθορισμένο constraint set *constrs_1*.



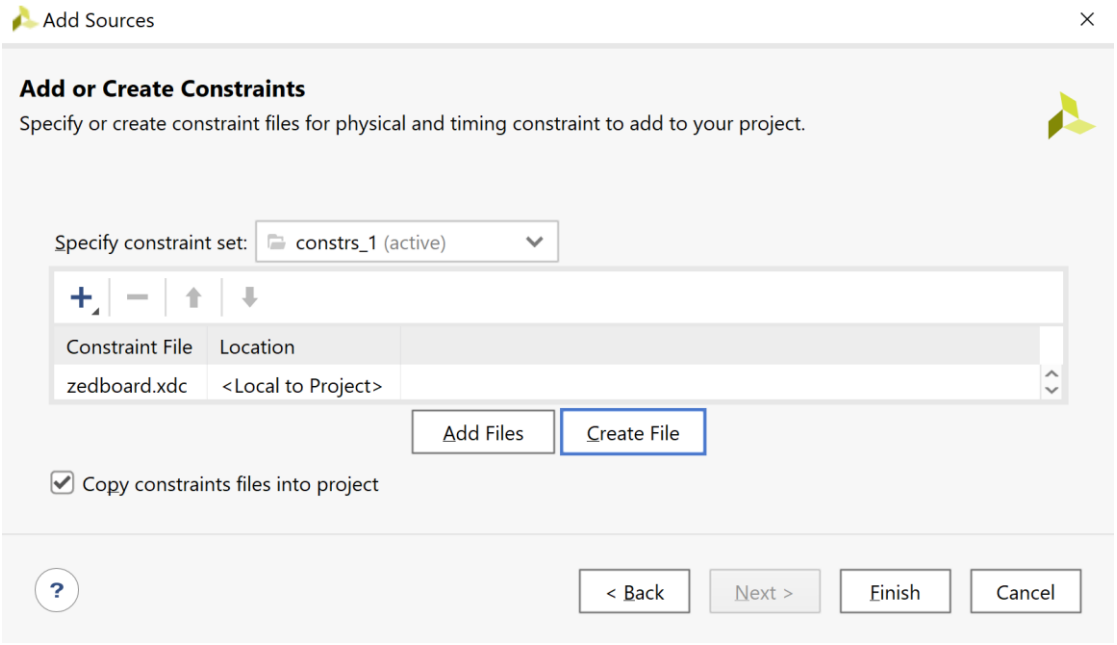
Εικόνα 9 – Επιλογή για δημιουργία νέου άδειου αρχείου .xdc

- Στο παράθυρο διαλόγου *Create Constraints File* δηλώστε το όνομα του αρχείου XDC (π.χ. zedboard), αφού οι περιορισμοί σχετίζονται με την αναπτυξιακή κάρτα Zedboard. Πατήστε **OK**.



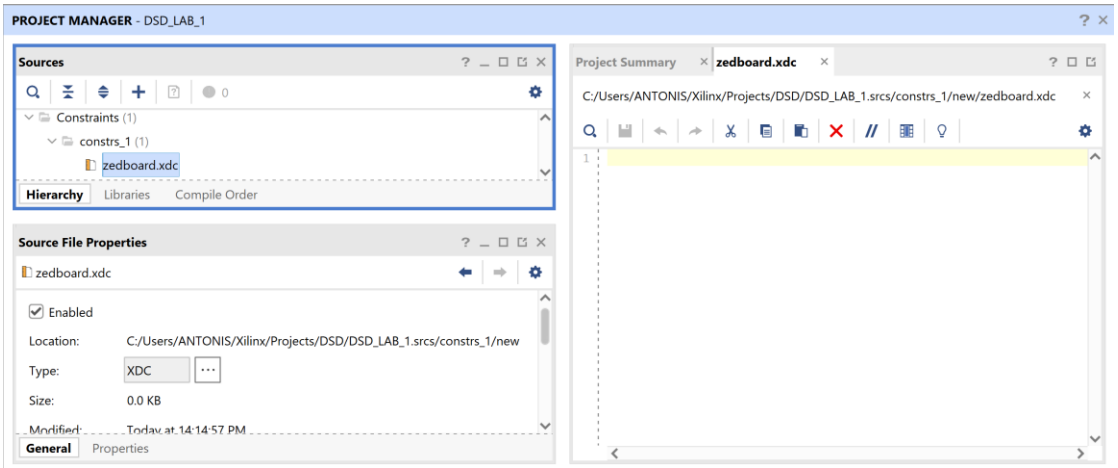
Εικόνα 10 - Δημιουργία αρχείου .xdc

- Επιστρέφετε στο παράθυρο διαλόγου *Add or Create Constraints*, όπου φαίνεται ότι έχει δημιουργηθεί το αρχείο **zedboard.xdc** και έχει συμπεριληφθεί στα αρχεία του project που ονομάζεται **DSD_LAB_1**. Πατήστε **Finish**.



Εικόνα 11 – Ολοκλήρωση διαδικασίας δημιουργίας constraints

- Επιβεβαιώστε τη δημιουργία του αρχείου **zedboard.xdc** στο παράθυρο *Sources* του PROJECT MANAGER και ανοίξτε το με διπλό κλικ. Αρχικά είναι άδειο.



Εικόνα 12 – Επιβεβαίωση δημιουργίας του αρχείου .xdc

- Με copy–paste συμπληρώνουμε το αρχείο **zedboard.xdc** με βάση το **πρότυπο zedboard.xdc**, που παρατίθεται στο Παράρτημα Γ από το κατασκευαστή (AVNET) της αναπτυξιακής κάρτας **Zedboard**. Αφαιρούμε το σύμβολο (#), όπου απαιτείται. Τέλος κάνουμε **Save**.

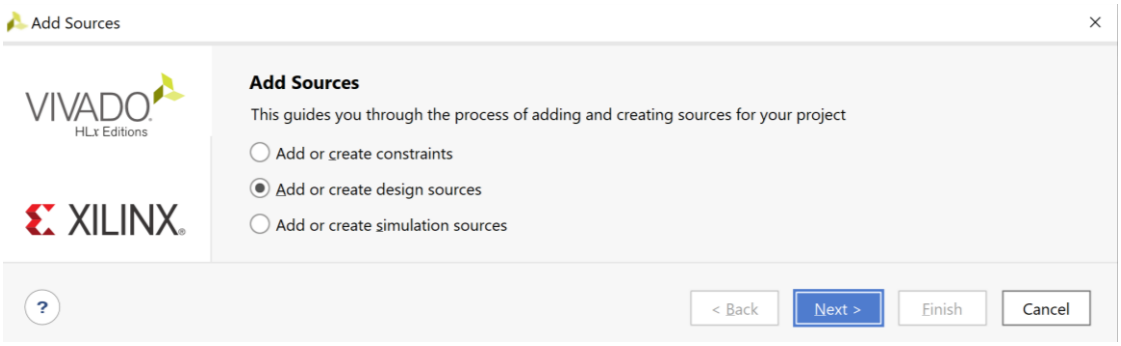
  Στη συνέχεια φαίνεται το περιεχόμενο του **zedboard.xdc**, που καλύπτει τις εργαστηριακές μας ανάγκες (δήλωση και χρονικοί περιορισμοί (περίοδος 10 ns) για το σήμα του ρολογιού CLK, φυσικοί περιορισμοί για σήμα CLK, τα 5 GPIO push buttons, που μας επιτρέπουν να παράγουμε σήματα RESET και EN με το πάτημα του αντίστοιχου κουμπιού, τα 8 DIP Switches για αρχικές σταθερές τιμές σε εισόδους σημάτων και τα 8 LEDs για παρατήρηση τιμών σε εξόδους σημάτων).

```
##############################################################
# ZedBoard Pin Assignments
##############################################################

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

# User GPIO push button for RESET and EN purposes
#set_property PACKAGE_PIN P16 [get_ports {BTNC}];  # "BTNC" central
#set_property PACKAGE_PIN R16 [get_ports {BTND}];  # "BTND" down
#set_property PACKAGE_PIN N15 [get_ports {BTNL}];  # "BTNL" left
#set_property PACKAGE_PIN R18 [get_ports {BTNR}];  # "BTNR" right
#set_property PACKAGE_PIN T18 [get_ports {BTNU}];  # "BTNU" up

# User DIP Switches – 8 bit user input
#set_property PACKAGE_PIN F22 [get_ports {SW0}];  # "SW0"
#set_property PACKAGE_PIN G22 [get_ports {SW1}];  # "SW1"
#set_property PACKAGE_PIN H22 [get_ports {SW2}];  # "SW2"
#set_property PACKAGE_PIN F21 [get_ports {SW3}];  # "SW3"
#set_property PACKAGE_PIN H19 [get_ports {SW4}];  # "SW4"
#set_property PACKAGE_PIN H18 [get_ports {SW5}];  # "SW5"
#set_property PACKAGE_PIN H17 [get_ports {SW6}];  # "SW6"
#set_property PACKAGE_PIN M15 [get_ports {SW7}];  # "SW7"

# User LEDs – 8 bit user output
#set_property PACKAGE_PIN T22 [get_ports {LD0}];  # "LD0"
#set_property PACKAGE_PIN T21 [get_ports {LD1}];  # "LD1"
#set_property PACKAGE_PIN U22 [get_ports {LD2}];  # "LD2"
#set_property PACKAGE_PIN U21 [get_ports {LD3}];  # "LD3"
#set_property PACKAGE_PIN V22 [get_ports {LD4}];  # "LD4"
#set_property PACKAGE_PIN W22 [get_ports {LD5}];  # "LD5"
#set_property PACKAGE_PIN U19 [get_ports {LD6}];  # "LD6"
#set_property PACKAGE_PIN U14 [get_ports {LD7}];  # "LD7"


##########################################################
##ZedBoard Timing Constraints
##########################################################

# define clock and period
create_clock –period 10 –name CLK –waveform {0.000 5.000} [get_ports {CLK}]
```

1.7 Βήμα 2: Εισαγωγή του κώδικα VHDL και ανάλυση στο επίπεδο RTL

1.7.1 Δημιουργία νέου αρχείου τύπου VHD στο VIVADO IDE

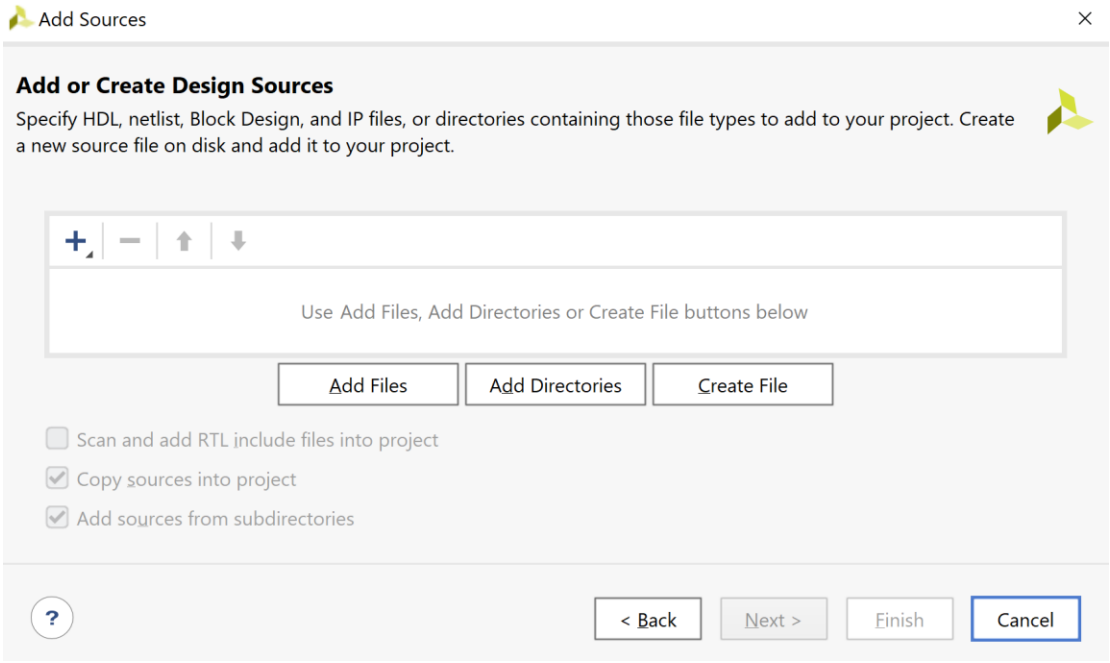
- Πατήστε το **+** στο παράθυρο *Sources* του PROJECT MANAGER για να ξεκινήσετε τον wizard δημιουργίας ενός νέου αρχείου (source).
- Στο παράθυρο διαλόγου *Add Sources* επιλέξτε το Add or create design sources, ώστε να δημιουργήσετε ένα νέο πηγαίο αρχείο περιγραφής στο επίπεδο RTL ενός ψηφιακού κυκλώματος (ή συστήματος, ανάλογα με την πολυπλοκότητα της σχεδίασης) στη γλώσσα VHDL (design source file) τύπου VHD. Σε αυτό το αρχείο εισάγουμε τον κώδικα VHDL. Πατήστε **Next**.



Εικόνα 13 – Πρώτο βήμα για τη δημιουργία πηγαίου αρχείου VHDL

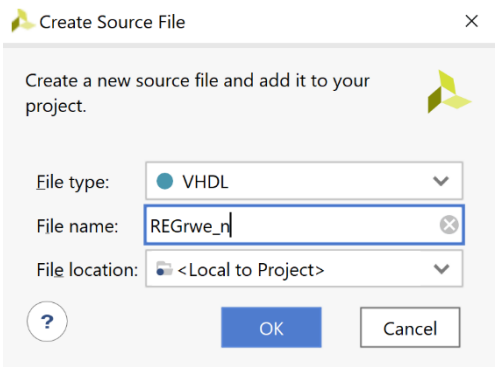
- Στο παράθυρο διαλόγου *Add or Create Design Sources* πατήστε την επιλογή **Create File** για τη δημιουργία του design source file που θα τοποθετηθεί στο ήδη καθορισμένο design source set *sources_1*.

  Εάν απαιτείται επιλέξτε την **VHDL** ως *Target Language* και το **Mixed** ως *Simulator Language*.



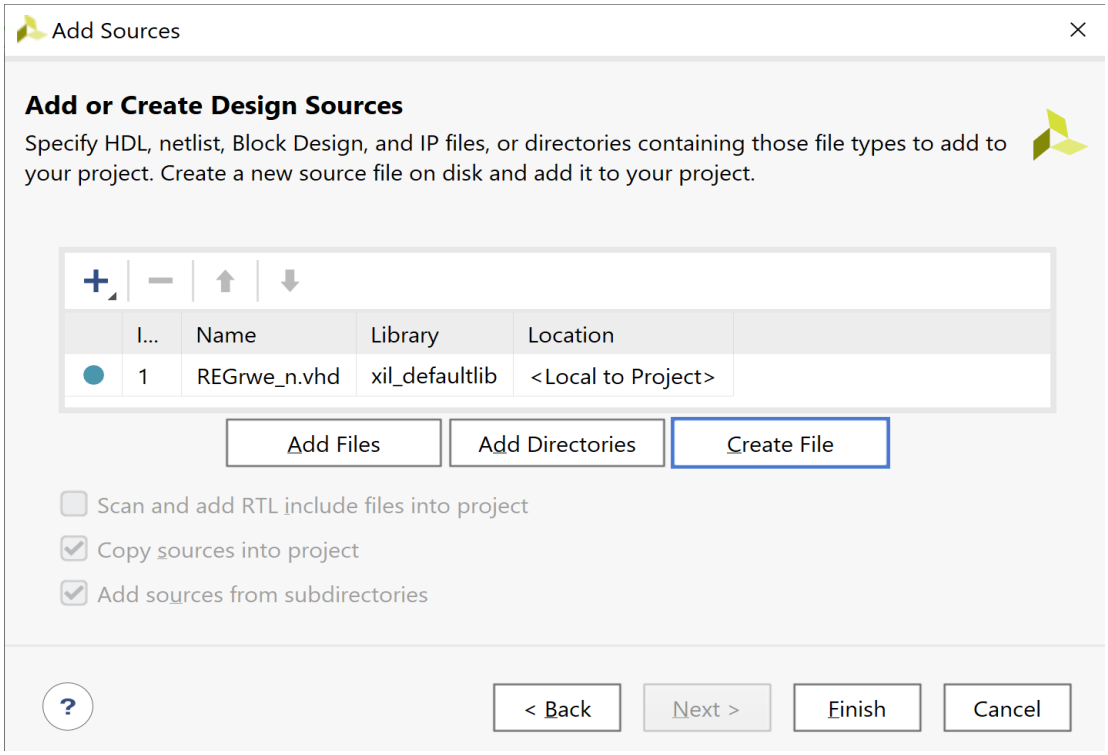
Εικόνα 14 – Επιλογή για δημιουργία νέου άδειου αρχείου .vhd

- Στο παράθυρο διαλόγου *Create Source File* δηλώστε το όνομα του αρχείου VHD (π.χ. **REGrwe_n**), αφού σε αυτό το αρχείο θα περιγράψετε στη γλώσσα VHDL έναν παραμετροποιημένο καταχωρητή των N bit με σύγχρονη επαναφορά στο 0 με την ενεργοποίηση του σήματος RESET (RESET = 1) και με έγκριση εγγραφής με την ενεργοποίηση του σήματος ελέγχου WE (WE = 1). Πατήστε **OK**.



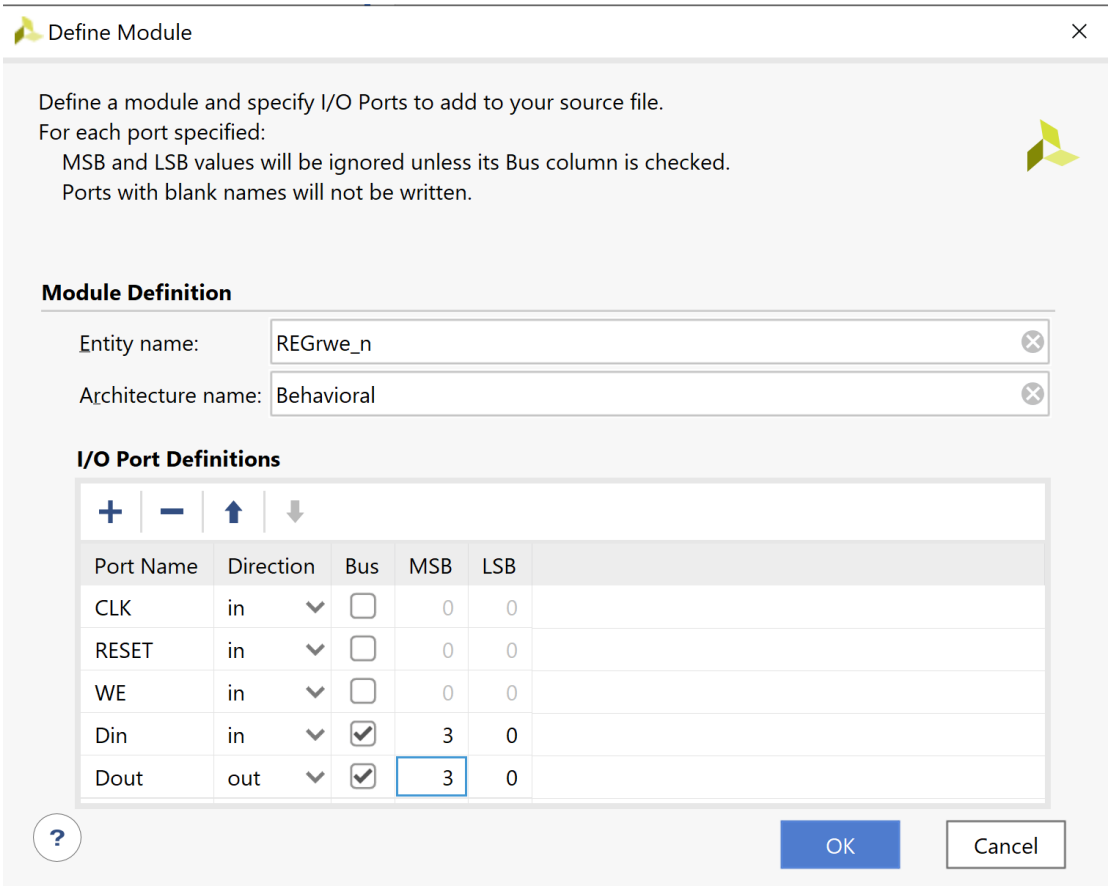
Εικόνα 15 - Δημιουργία αρχείου .vhd

- Επιστρέφετε στο παράθυρο διαλόγου *Add or Create Design Sources*, όπου φαίνεται ότι έχει δημιουργηθεί το αρχείο **REGrwe_n.vhd** και έχει συμπεριληφθεί στα αρχεία του project που ονομάζεται DSD_LAB_1. Πατήστε **Finish**.



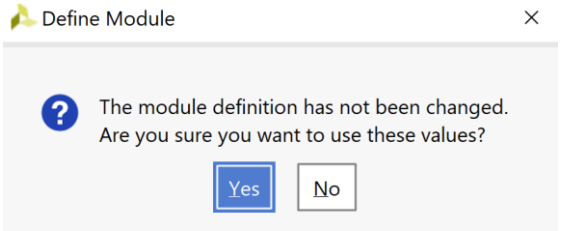
Εικόνα 16 – Ολοκλήρωση διαδικασίας δημιουργίας πηγαίου αρχείου

- Αμέσως μετά εμφανίζεται το παράθυρο διαλόγου *Define Module*, όπου σας παρέχεται η δυνατότητα της δήλωσης του ονόματος της οντότητας, του ονόματος της αρχιτεκτονικής και των ports της οντότητας. Ως όνομα της οντότητας διατηρείτε το ίδιο όνομα που είχατε δώσει στο αρχείο (**REGrwe_n**). Επιλέξτε ως όνομα της αρχιτεκτονικής κάποιο από τα ονόματα *behavioral, structural, dataflow, mixed* ανάλογα με το είδος της περιγραφής που θα κάνετε στη γλώσσα VHDL. Στη συνέχεια, προαιρετικά δηλώστε τα ports στο **I/O Port Definitions**. Στη συγκεκριμένη περίπτωση επιλέξτε το όνομα **behavioral**. Ας υποθέσουμε ότι αρχικά δηλώνουμε τα ports ενός μη–παραμετροποιημένου καταχωρητή των 4 bit με εισόδους **CLK**, **RESET**, **WE**, **Din[3:0]** και έξοδο **Dout[3:0]**. Εισάγετε το όνομα του σήματος ή της αρτηρίας στο port name. Για τις αρτηρίες επιλέξτε επιπλέον το Bus και ορίστε την τιμή του MSB (3) και του LSB (0). Πατήστε το **+** για δήλωση επιπλέον ports με τον ίδιο τρόπο. Τέλος, πατήστε **OK**.



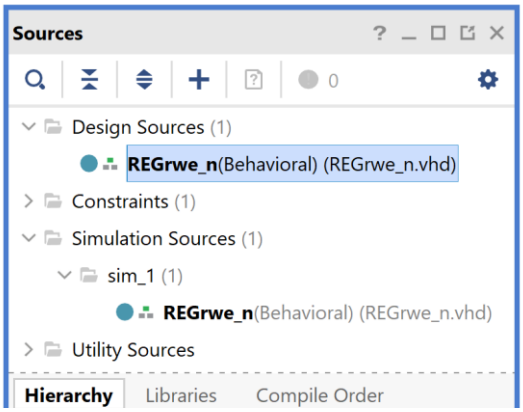
Εικόνα 17 – Δήλωση οντότητας, αρχιτεκτονικής και διεπαφής οντότητας

Μπορούμε να παρακάμψουμε τη δήλωση των ports. Στο παράθυρο προειδοποίησης *Define Module* που θα εμφανισθεί πατήστε **Yes**.



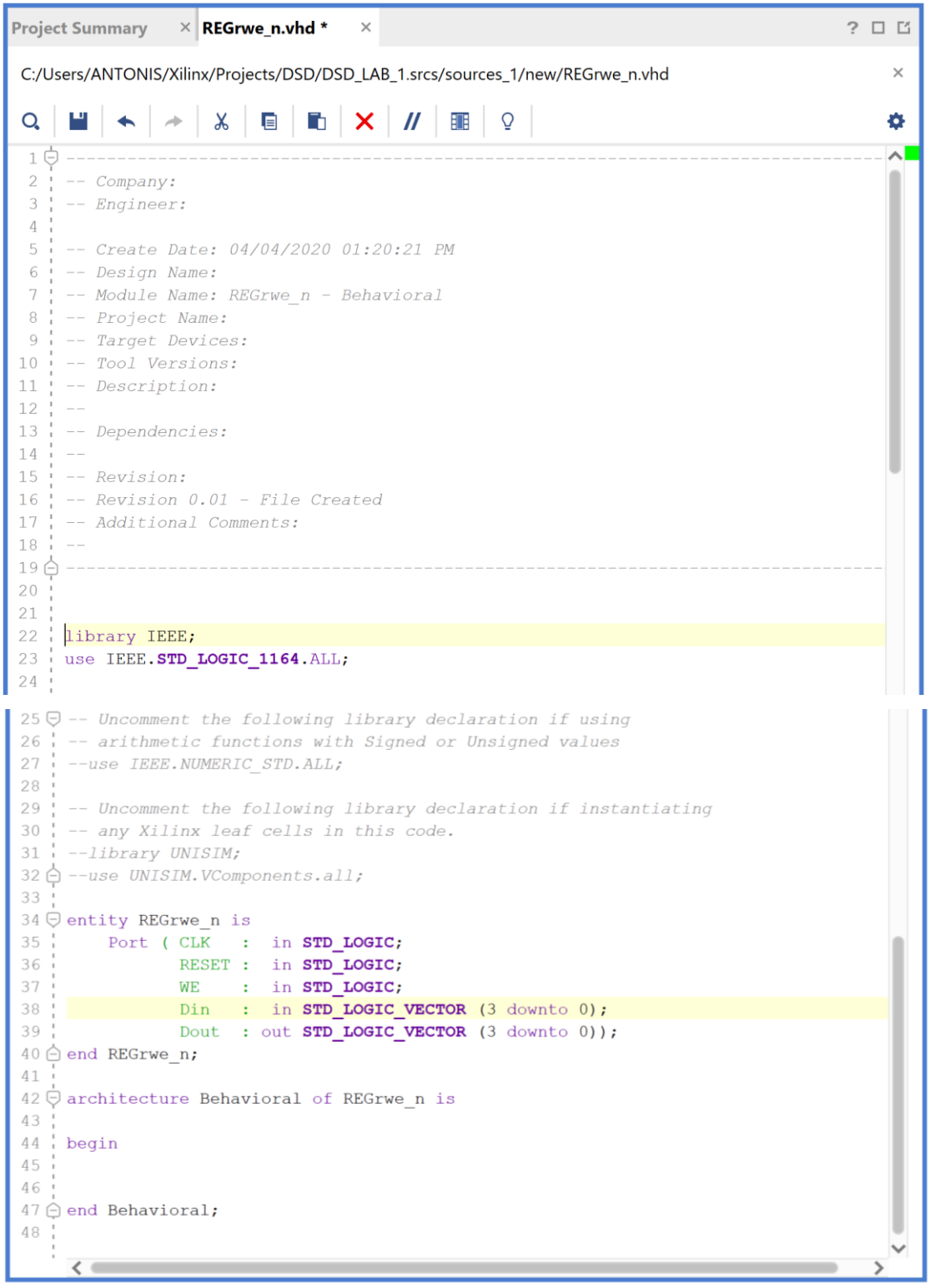
Εικόνα 18 – Παράκαμψη ορισμού ports

- Επιβεβαιώστε τη δημιουργία του αρχείου **REGrwe_n.vhd** στο παράθυρο *Sources* του PROJECT MANAGER (επίσης φαίνεται το όνομα της οντότητας **REGrwe_n** και το όνομα της αρχιτεκτονικής **Behavioral**). Το αρχείο αυτό είναι αποθηκευμένο και στο design source set *sources_1* και στο simulation source set *sim_1* του project DSD_LAB1. Επειδή δεν υπάρχει άλλη οντότητα διαθέσιμη στα design sources του project, η οντότητα **REGrwe** ορίζεται αυτόματα ως η κορυφαία οντότητα της ιεραρχίας (**top**) επί της οποίας θα εκτελεσθούν στη συνέχεια τα επόμενα βήματα της σχεδίασης (RTL analysis, Synthesis, Implementation, Program and debug). Η κορυφαία οντότητα (**top design source)** διαφοροποιείται από τις υπόλοιπες οντότητες του design source set *sources_1* με το σύμβολο και τα **έντονα** (bold) γράμματα στο όνομα της οντότητας.



Εικόνα 19 – Διαθέσιμα sources

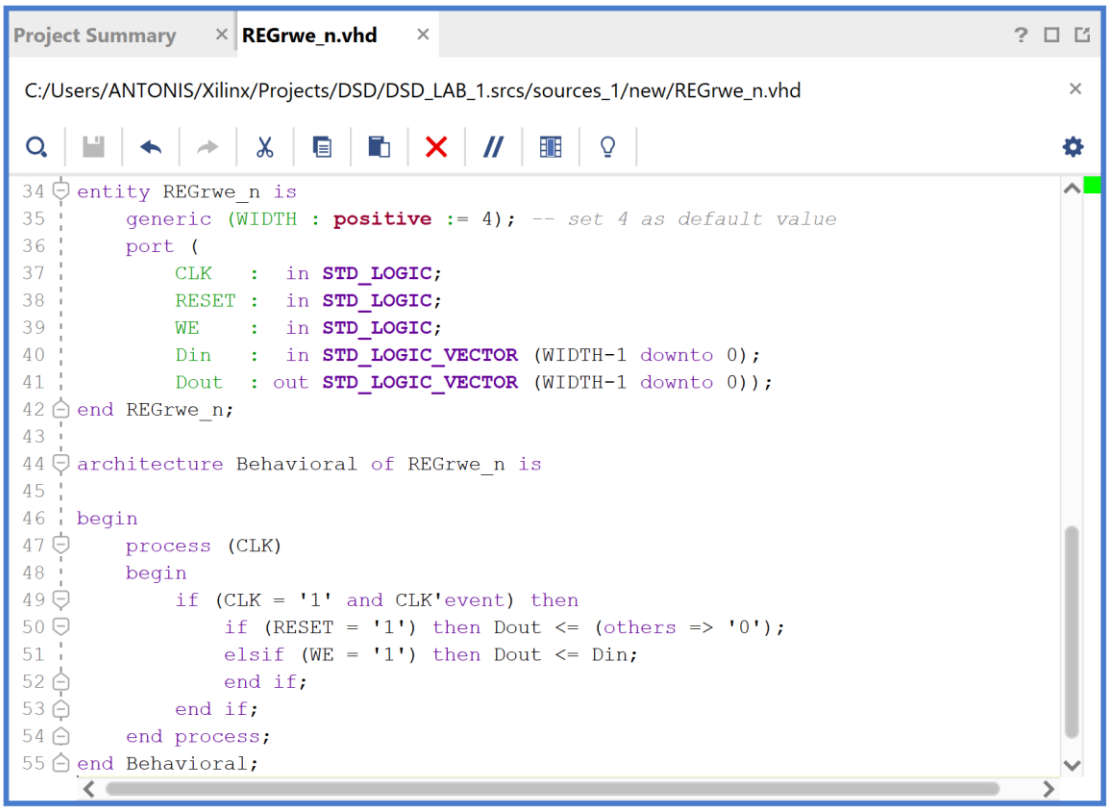
- Ανοίξτε το αρχείο **REGrwe_n.vhd** με διπλό κλικ στο επιλεγμένο αρχείο. Τα ports της οντότητας έχουν ήδη ορισθεί, αλλά λείπει ο ορισμός της αρχιτεκτονικής της οντότητας.



Εικόνα 20 – Περιεχόμενα πηγαίου αρχείου στην αρχική του μορφή

Στις δυνατότητες του VHDL editor να λάβετε υπόψη σας το *Toggle Line Comments* και το *Toggle Column Selection Mode* (ιδιαίτερα χρήσιμο στην αντιγραφή ονομάτων σημάτων που έχετε στοιχίσει εκ των προτέρων) που βρίσκονται μετά το ×.

- Παραμετροποιείστε τις αρτηρίες Din και Dout με τη δήλωση generic πριν τη δήλωση των ports και συμπληρώστε τον ορισμό της αρχιτεκτονικής με βάση τον κώδικα που δίδεται στη σελίδα 301 του Κεφαλαίου 4 «Γλώσσες περιγραφής υλικού – Πλήρης έκδοση» των παραδόσεων του μαθήματος. Τέλος πατήστε **Save File**.

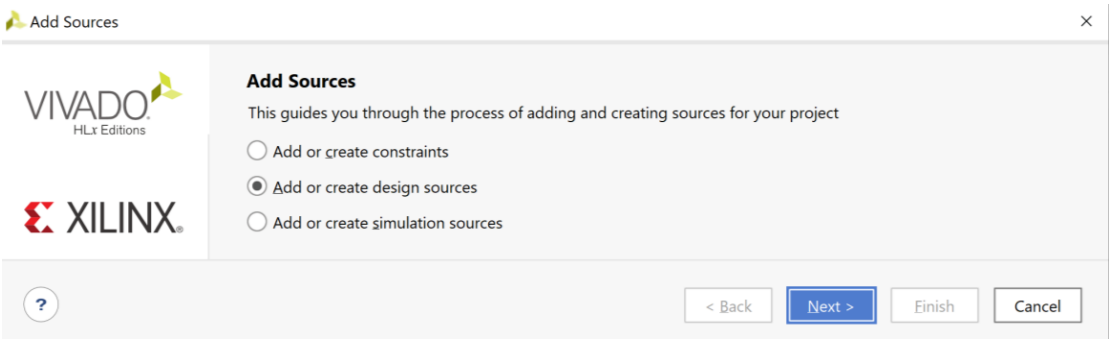


Εικόνα 21 – Περιεχόμενα πηγαίου αρχείου μετά τις τροποποιήσεις/προσθήκες

- Χρησιμοποιήστε τον File Manager και κοιτάξτε στο φάκελο που ορίσατε το project. Θα βρείτε ότι έχει δημιουργηθεί εντός αυτού, μεταξύ άλλων φακέλων, και ο φάκελος **DSD_LAB_1.srcs**, καθώς και το Vivado Project File **DSD_LAB_1**. Μέσα στον φάκελο **DSD_LAB_1.srcs** θα βρείτε τους υποφακέλους **constrs_1** και **sources_1**. Στον υποφάκελο **constrs_1/new** θα βρείτε το **ZedBoard.xdc** (constraints file), ενώ στον υποφάκελο **sources_1/new** θα βρείτε το **REGrwe_n.vhd** (design source file).

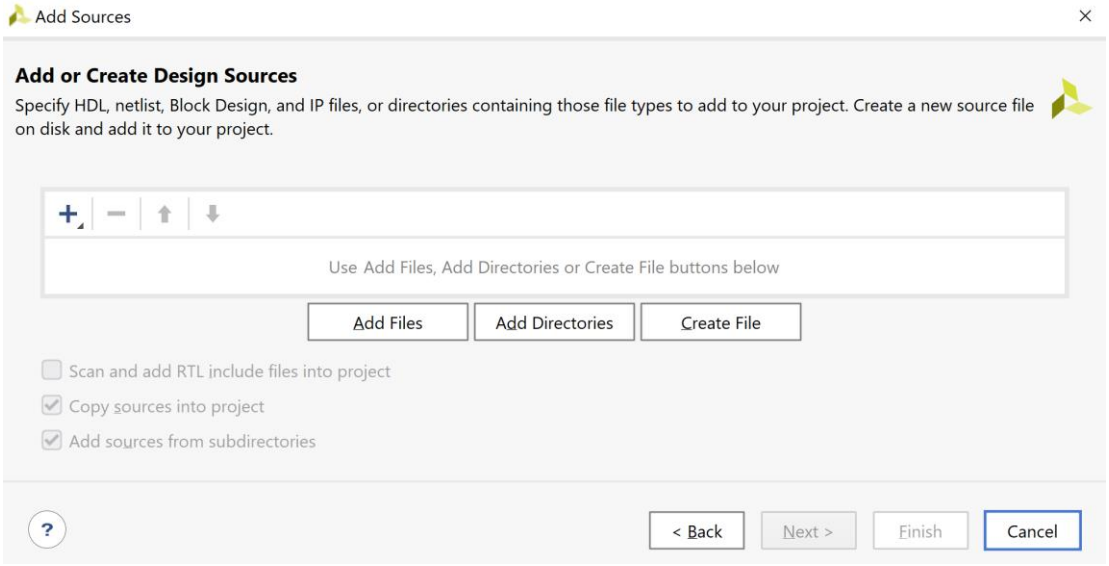
1.7.2 Προσθήκη υπάρχοντος αρχείου τύπου VHD στο VIVADO IDE

- Πατήστε το **+** στο παράθυρο *Sources* του PROJECT MANAGER για να ξεκινήσετε τον wizard δημιουργίας ενός νέου αρχείου (source).
- Στο παράθυρο διαλόγου *Add Sources* επιλέξτε το Add or create design sources, ώστε να προσθέσετε ένα υπάρχον πηγαίο αρχείο περιγραφής στο επίπεδο RTL ενός ψηφιακού κυκλώματος (ή συστήματος, ανάλογα με την πολυπλοκότητα της σχεδίασης) στη γλώσσα VHDL (design source file) τύπου VHD. Πατήστε **Next**.



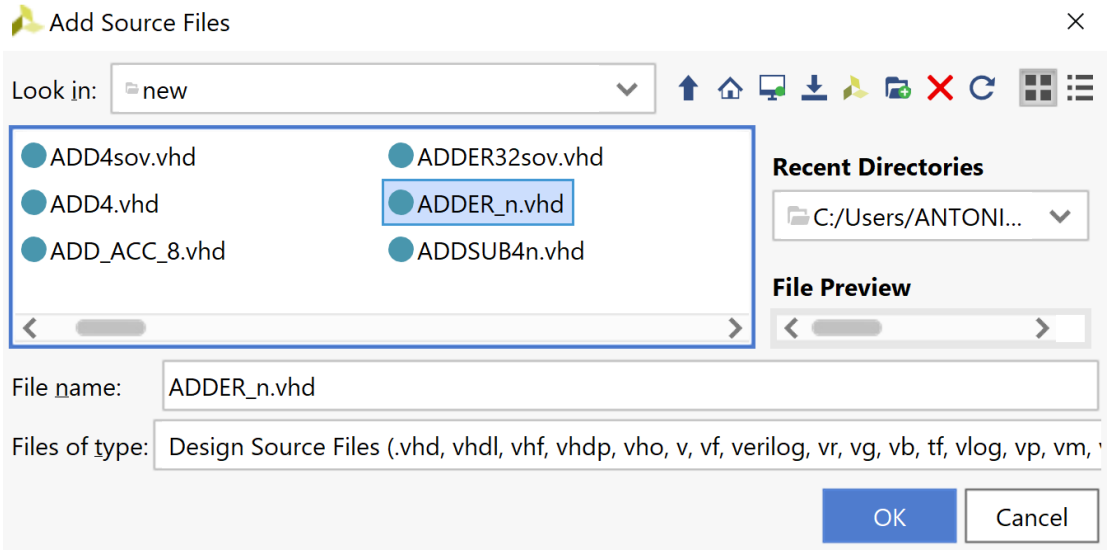
Εικόνα 22 – Πρώτο βήμα για την προσθήκη υπάρχοντος αρχείου .vhd

- Στο παράθυρο διαλόγου *Add or Create Design Sources* πατήστε την επιλογή **Add Files** για την προσθήκη υπαρχόντων design source files που θα τοποθετηθούν στο ήδη καθορισμένο design source set *sources_1*.



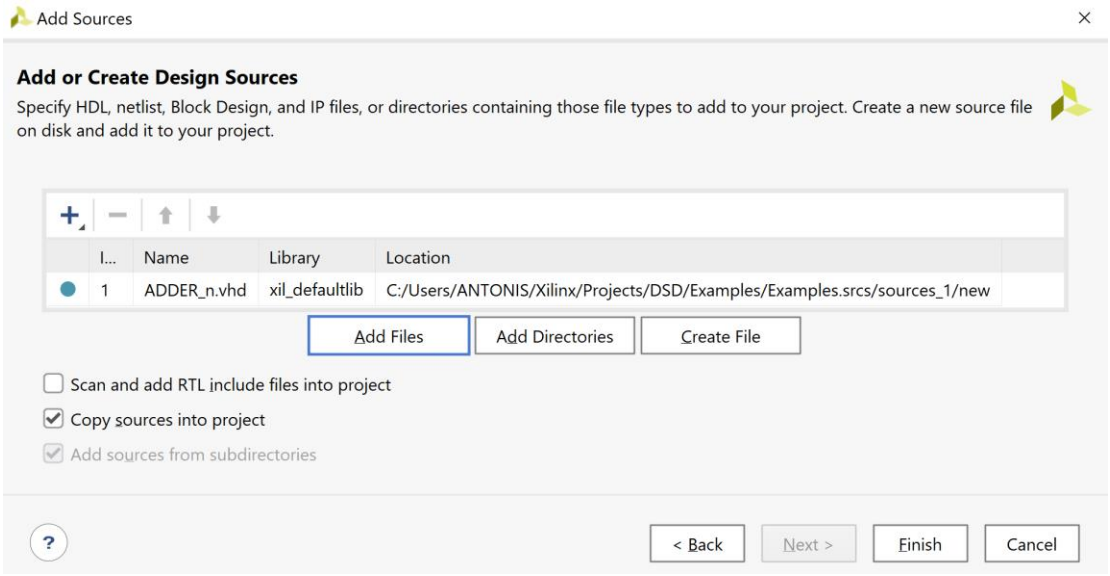
Εικόνα 23 – Επιλογή για προσθήκη υπάρχοντος αρχείου .vhd

- Στο παράθυρο διαλόγου *Add Source File* επιλέξτε το αρχείο **ADDER_n** που περιγράφει έναν παραμετροποιημένο προσημασμένο αθροιστή με κρατούμενο εξόδου Cout και υπερχείλιση OV των N bit με βάση τον κώδικα που δίδεται στη σελίδα 303 του Κεφαλαίου 4 «Γλώσσες περιγραφής υλικού – Πλήρης έκδοση» των παραδόσεων του μαθήματος. Τέλος πατήστε **OK**.



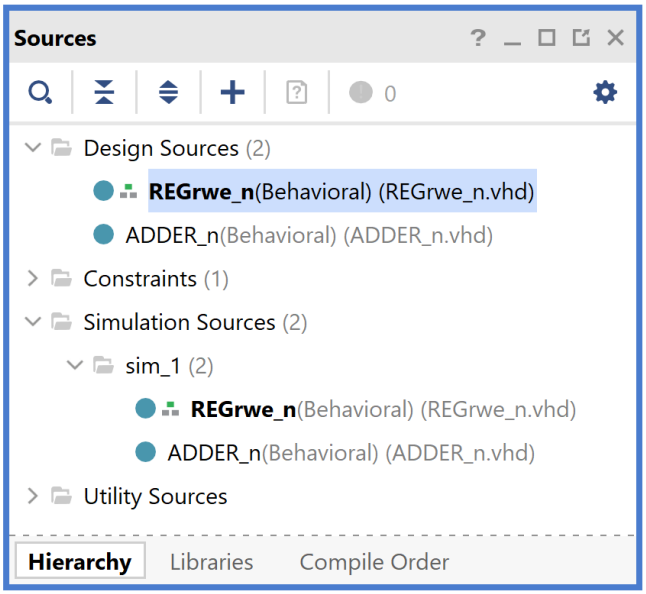
Εικόνα 24 – Προσδιορισμός υπάρχοντος αρχείου .vhd προς χρήση

- Επιστρέφετε στο παράθυρο διαλόγου *Add or Create Design Sources*, όπου φαίνεται ότι έχει προσδιορισθεί η θέση του αρχείου **ADDER_n.vhd** που θα αντιγραφεί στα αρχεία του project που ονομάζεται DSD_LAB_1. Βεβαιωθείτε ότι το κουτί **Copy sources into project** είναι επιλεγμένο, ώστε να επιτευχθεί η αντιγραφή του αρχείου. Πατήστε **Finish**.



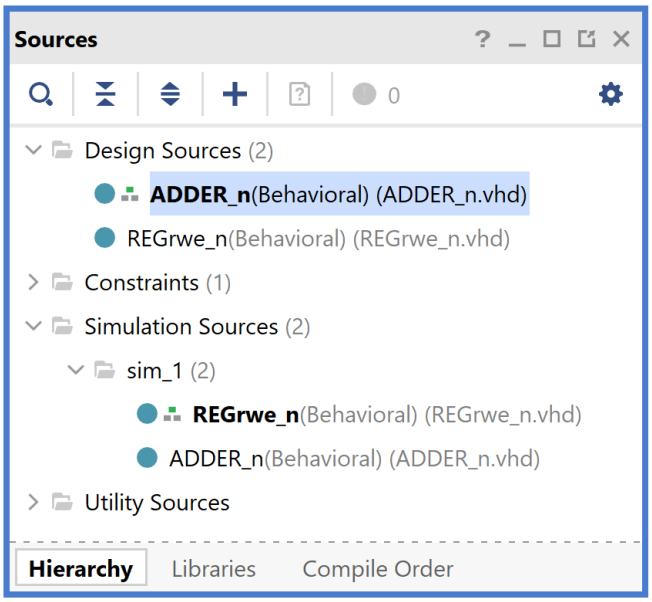
Εικόνα 25 – Ολοκλήρωση διαδικασίας προσθήκης υπάρχοντος αρχείου .vhd

- Επιβεβαιώστε τη δημιουργία του αρχείου **ADDER_n.vhd** στο παράθυρο *Sources* του PROJECT MANAGER (επίσης φαίνεται το όνομα της οντότητας **Adder_n** και το όνομα της αρχιτεκτονικής **Behavioral**). Η οντότητα **Adder_n** είναι αποθηκευμένη και στο design source set sources_1 και στο simulation source set sim_1 του project DSD_LAB1. Επί του παρόντος, η οντότητα **REGrwe_n** είναι ορισμένη ως η κορυφαία οντότητα της ιεραρχίας (**top**) επί της οποίας θα εκτελεσθούν στη συνέχεια τα επόμενα βήματα της σχεδίασης (RTL analysis, Synthesis, Implementation, Program and debug).



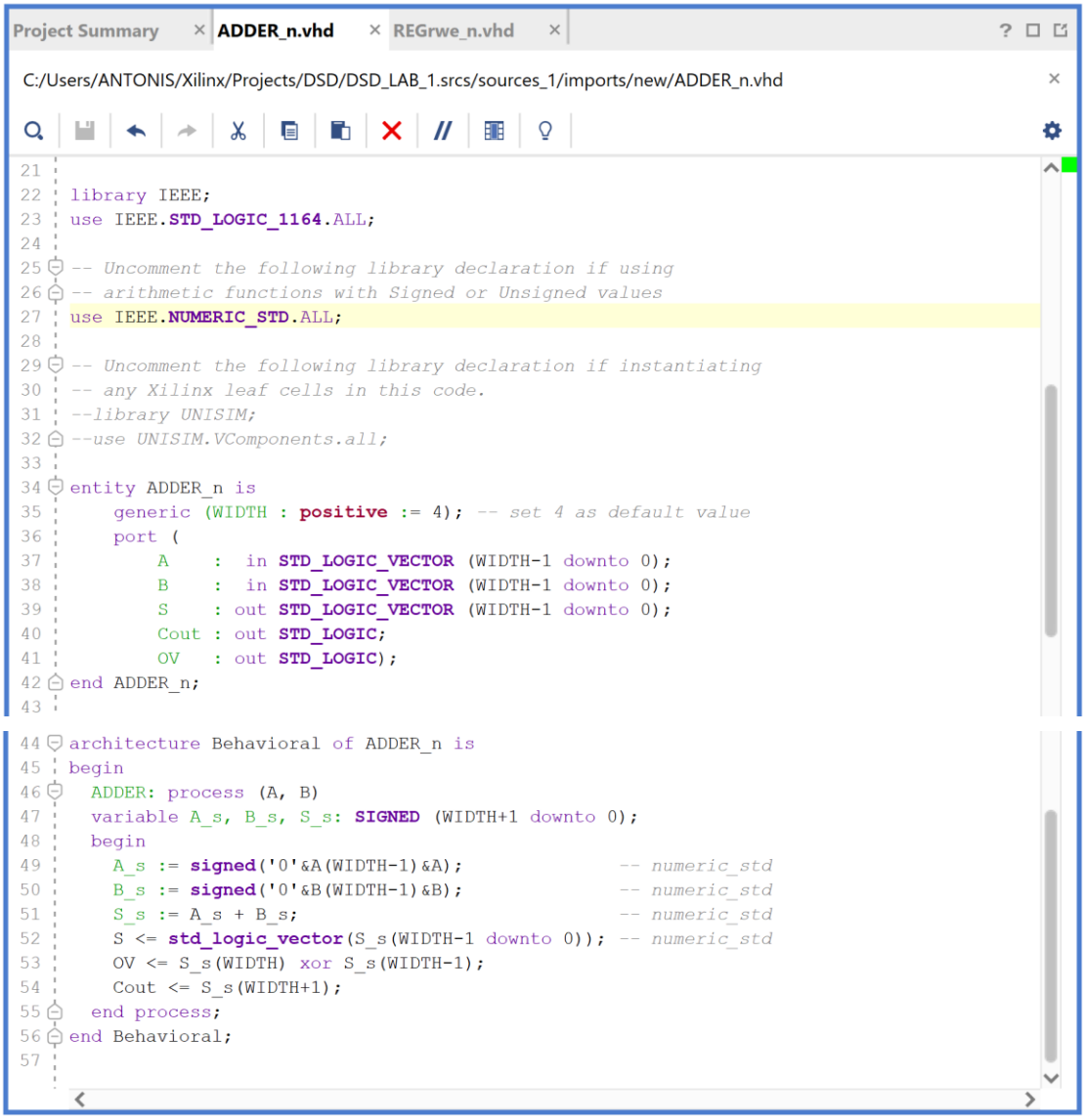
Εικόνα 26 - Διαθέσιμα sources

- Για να ορισθεί η οντότητα **ADDER_n** ως η κορυφαία οντότητα της ιεραρχίας (**top**) των design sources, θα πρέπει να κάνετε δεξί κλικ επί του ονόματος στο παράθυρο *Sources* του PROJECT MANAGER και να επιλέξετε το **Set as Top**. Η κορυφαία οντότητα (**top design source)** διαφοροποιείται από τις υπόλοιπες οντότητες του design source set *sources*_1 με το σύμβολο και τα **έντονα** (bold) γράμματα στο όνομα της οντότητας.



Εικόνα 27 – Ορισμός νέας κορυφαίας οντότητας της ιεραρχίας

- Ανοίξτε το αρχείο **ADDER_n.vhd** με διπλό κλικ στο επιλεγμένο αρχείο.



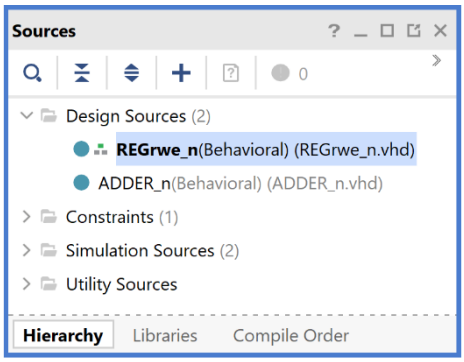
Εικόνα 28 – Περιεχόμενα υπάρχοντος αρχείου Adder_n.vhd

Προσέξτε ιδιαίτερα τη δήλωση του πακέτου **NUMERIC.STD** και τον τρόπο που παράγονται τα σήματα Cout και OV στους προσημασμένους αθροιστές.

1.7.3 Ανάλυση του κώδικα VHDL στο επίπεδο RTL.

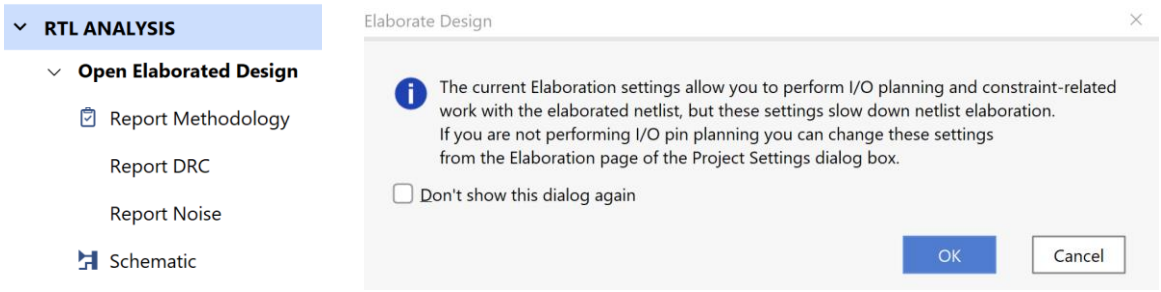
Τα πηγαία αρχεία περιγραφής συμπεριφοράς στη γλώσσα VHDL του καταχωρητή (**REGrwe_n.vhd**) και του αθροιστή (**ADDER_n.vhd**) αναλύονται στη συνέχεια στο **επίπεδο RTL**. Η ανάλυση περιορίζεται στη μελέτη του σχηματικού διαγράμματος στο επίπεδο RTL.

- Αρχικά, στο παράθυρο *Sources* του PROJECT MANAGER επιλέξτε την οντότητα **REGrwe_n** ως την κορυφαία οντότητα της ιεραρχίας (**top**), κάνοντας δεξί κλικ επί του επιλεγμένου ονόματος και επιλέγοντας το **Set as Top**. Το παράθυρο *Sources* διαμορφώνεται όπως το βλέπετε δεξιά.



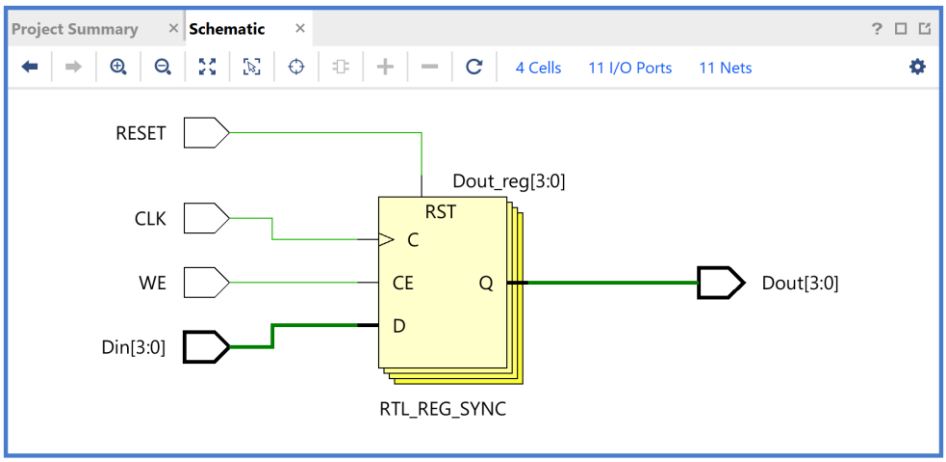
Εικόνα 29 - Επιλογή REGrwe_n

- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Open Elaborated Design**, ώστε να εμφανισθούν οι δυνατότητες που παρέχει το εργαλείο Vivado IDE μετά την ανάλυση στο επίπεδο RTL (RTL ANALYSIS) του **behavioral (elaborated design) model** της κορυφαίας οντότητας **REGrwe_n** της ιεραρχίας (**top**). Εάν εμφανισθεί το παράθυρο προειδοποίησης *Elaborate Design*, πατήστε **OK**, ώστε να εκτελεσθεί η διαδικασία της ανάλυσης στο επίπεδο RTL.



Εικόνα 30 – Δυνατότητες του εργαλείου Vivado μετά την ανάλυση επιπέδου RTL

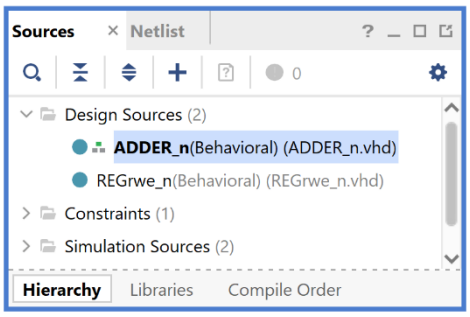
Εμφανίζεται το παράθυρο *Schematic* στη θέση του παραθύρου *Project Summary* με το σχηματικό διάγραμμα στο επίπεδο RTL του καταχωρητή των 4 bit με κοινές εισόδους CLK, RESET και EN. Ένα νέο παράθυρο *Schematic* εμφανίζεται κάθε φορά που επιλέγετε το **Schematic** στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*.



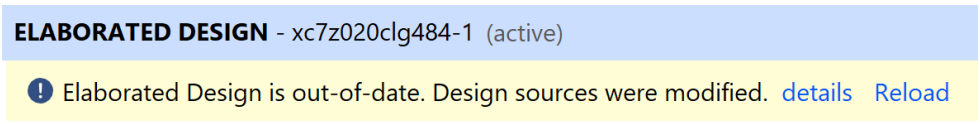
Εικόνα 31 – Σχηματικό διάγραμμα στο επίπεδο RTL του κυκλώματος REGrwe_n

- Στη συνέχεια, στο παράθυρο *Sources* του PROJECT MANAGER επιλέξτε την οντότητα **ADDER_n** ως την κορυφαία οντότητα της ιεραρχίας (**top**), κάνοντας δεξί κλικ επί του επιλεγμένου ονόματος και επιλέγοντας το **Set as Top**. Το παράθυρο *Sources* διαμορφώνεται όπως το βλέπετε δεξιά.

  Προσοχή! Επειδή είναι ενεργή η διαδικασία της ανάλυσης στο επίπεδο RTL, θα εμφανισθεί η προειδοποίηση αλλαγής του **behavioral (elaborated design) model**.



Εικόνα 32 - Επιλογή ADDER_n

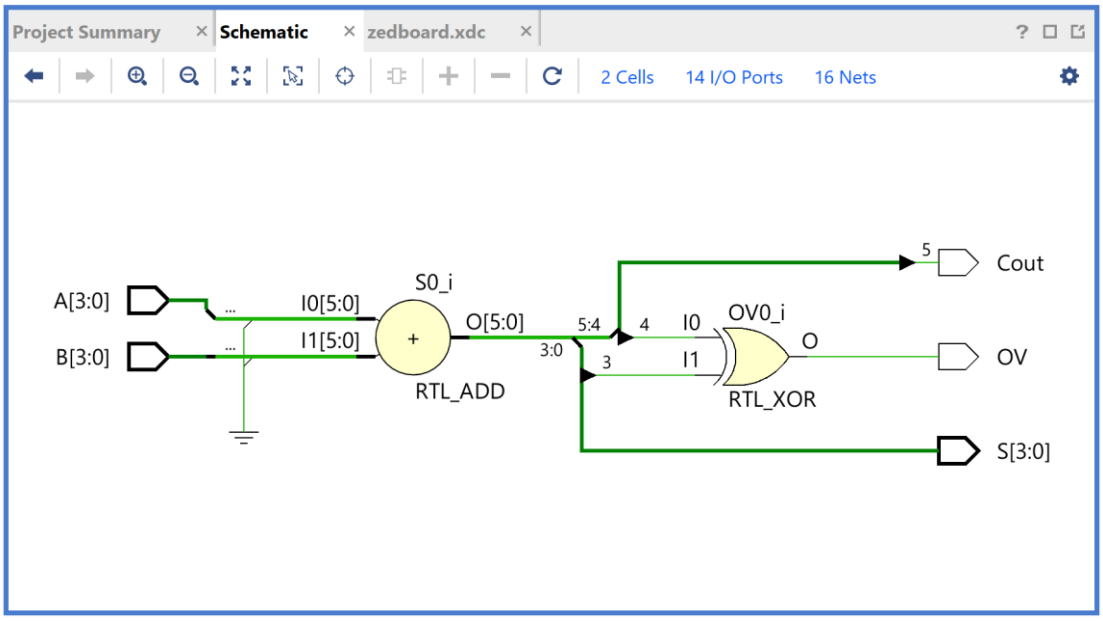


Εικόνα 33 - Προειδοποίηση αλλαγής του behavioral (elaborated design) model

- Με επιλογή του **details** επιβεβαιώστε την έγκριση του αρχείου **ADDER_n.vhd**.

Εικόνα 34 - Λεπτομέρειες προειδοποίησης αλλαγής του behavioral (elaborated design) model

- Με επιλογή του **Reload** εκτελέστε τη διαδικασία της ανάλυσης RTL της οντότητας **ADDER_n** (πατήστε **OK** σε όποιο παράθυρο προειδοποίησης εμφανισθεί). Εμφανίζεται στο νέο παράθυρο *Schematic* το σχηματικό διάγραμμα στο επίπεδο RTL του αθροιστή των 4 bit με εξόδους Cout και OV.

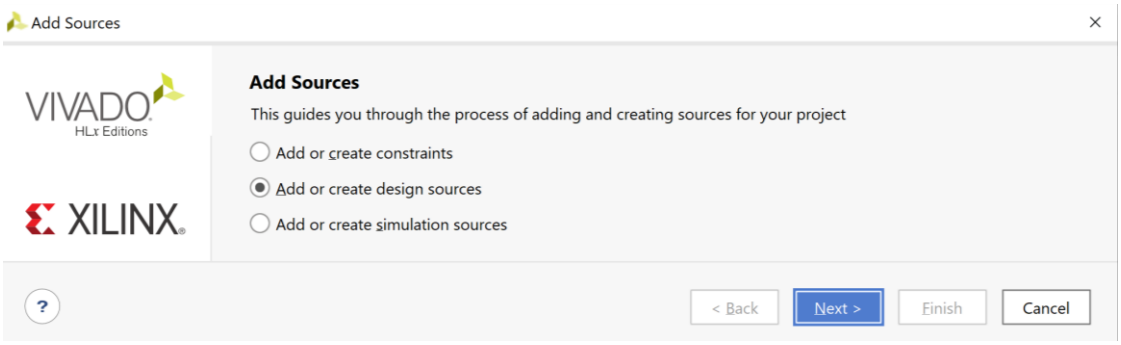


Εικόνα 35 – Σχηματικό διάγραμμα στο επίπεδο RTL του κυκλώματος ADDER_n

Επιβεβαιώστε το elaborated design. Συμπεριλαμβάνει έναν αθροιστή των 6 bit και μια πύλη XOR των 2 bit με έξοδο OV. Η έξοδος Cout είναι η έξοδος O[5] του αθροιστή.

1.7.4 Δημιουργία ιεραρχικής δομής τύπου VHD στο VIVADO IDE

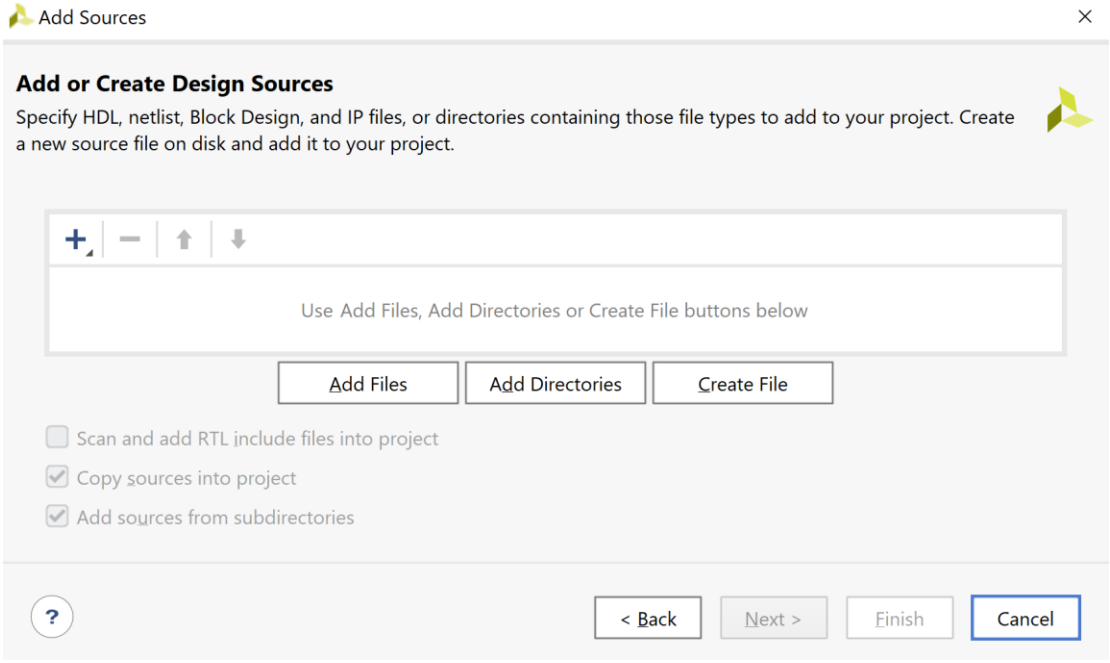
- Πατήστε το **+** στο παράθυρο *Sources* του PROJECT MANAGER για να ξεκινήσετε τον wizard δημιουργίας της ιεραρχικής δομής ως πηγαίου αρχείου (source) του αθροιστή των 8 bit με καταχωρητές εισόδου και εξόδου (**ADDER_REG_8**) που χρησιμοποιεί ως στοιχεία (components) τις οντότητες (entities) **REGrwe_n** και **ADDER_n**, που ήδη έχετε δημιουργήσει.
- Στο παράθυρο διαλόγου *Add Sources* επιλέξτε το Add or create design sources, ώστε να δημιουργήσετε ένα νέο πηγαίο αρχείο περιγραφής στο επίπεδο RTL ενός ψηφιακού κυκλώματος (ή συστήματος, ανάλογα με την πολυπλοκότητα της σχεδίασης) στη γλώσσα VHDL (design source file) τύπου VHD. Σε αυτό το αρχείο εισάγουμε τον κώδικα VHDL. Πατήστε **Next**.



Εικόνα 36 – Πρώτο βήμα για τη δημιουργία ιεραρχικής δομής με νέο αρχείο .vhd

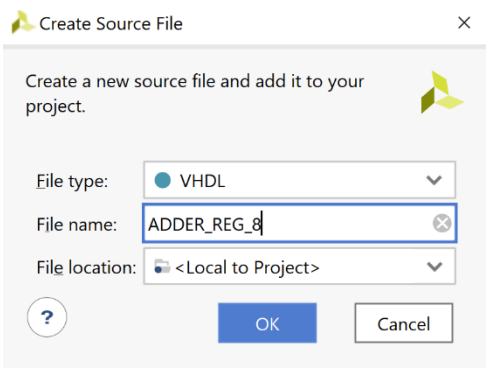
- Στο παράθυρο διαλόγου *Add or Create Design Sources* πατήστε την επιλογή **Create File** για τη δημιουργία του νέου design source file που θα τοποθετηθεί στο ήδη καθορισμένο design source set *sources_1*.

  Εάν απαιτείται επιλέξτε την **VHDL** ως *Target Language* και το **Mixed** ως *Simulator Language*.



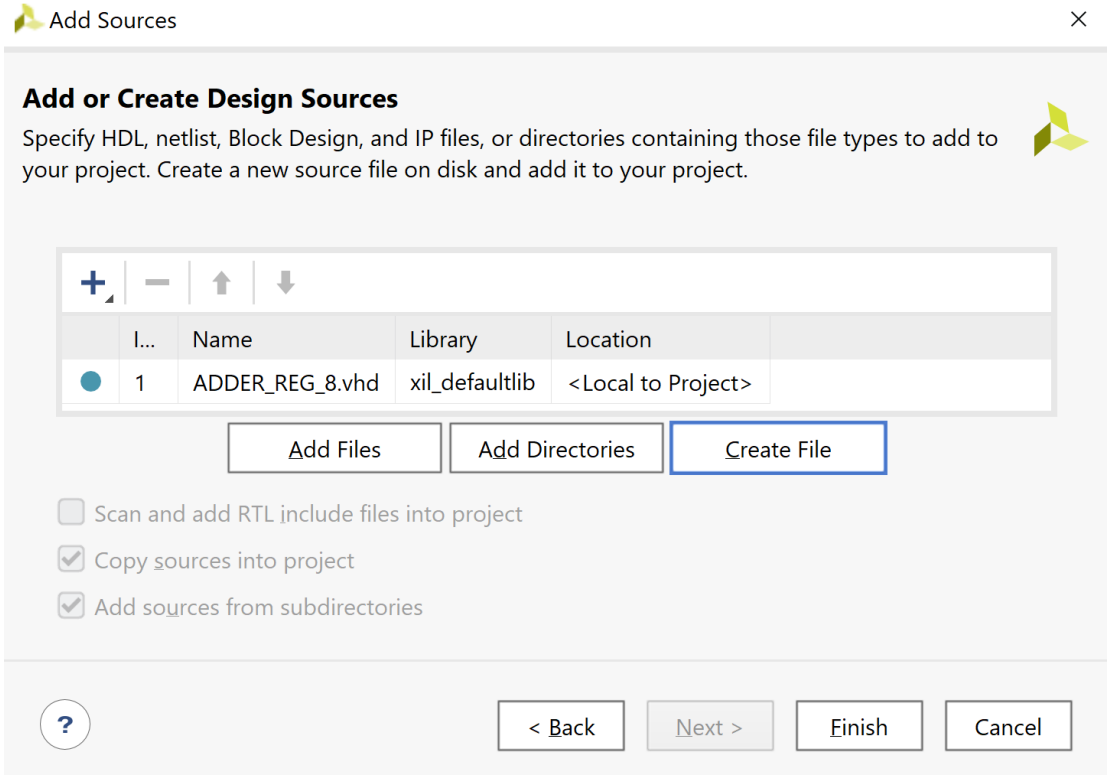
Εικόνα 37 – Επιλογή για δημιουργία νέου αρχείου .vhd

- Στο παράθυρο διαλόγου *Create Source File* δηλώστε το όνομα του αρχείου VHD (π.χ. **ADDER_REG_8**), αφού σε αυτό το αρχείο θα περιγράψετε στη γλώσσα VHDL έναν αθροιστή των 8 bit με καταχωρητές εισόδου και εξόδου, το οποίο θεωρείται σύγχρονο ακολουθιακό κύκλωμα. Πατήστε **OK**.



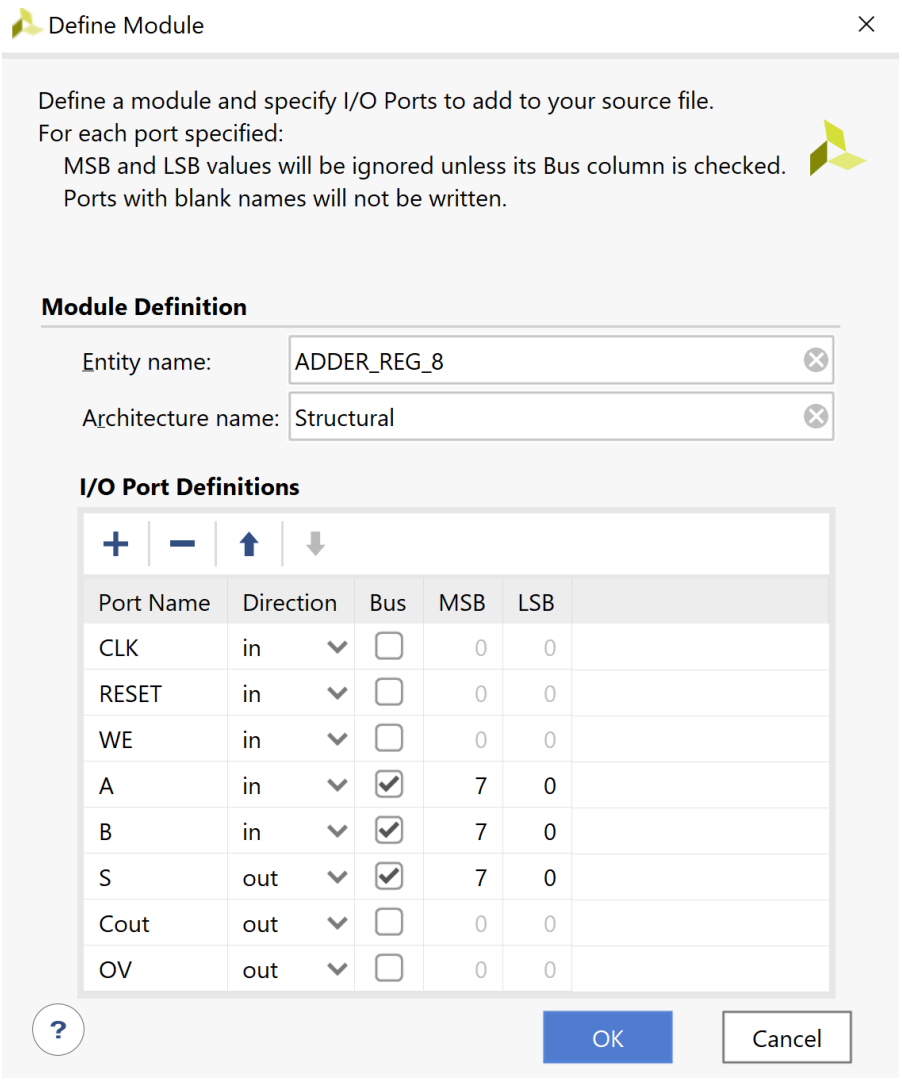
Εικόνα 38 - Δημιουργία αρχείου .vhd

- Επιστρέφετε στο παράθυρο διαλόγου *Add or Create Design Sources*, όπου φαίνεται ότι έχει δημιουργηθεί το αρχείο **ADDER_REG_8.vhd** και έχει συμπεριληφθεί στα αρχεία του project που ονομάζεται DSD_LAB_1. Πατήστε **Finish**.



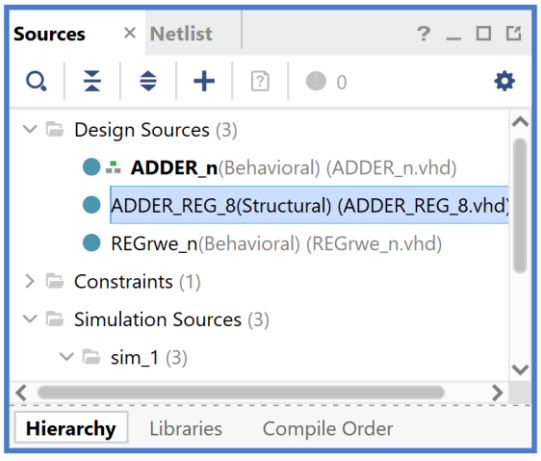
Εικόνα 39 – Ολοκλήρωση διαδικασίας προσθήκης νέου .vhd

- Εμφανίζεται το παράθυρο διαλόγου *Define Module*, όπου σας παρέχεται η δυνατότητα της δήλωσης του ονόματος της οντότητας, του ονόματος της αρχιτεκτονικής και των ports της οντότητας. Ως όνομα της οντότητας διατηρείστε το ίδιο όνομα που είχατε δώσει στο αρχείο (**ADDER_REG_8**). Επιλέξτε ως όνομα της αρχιτεκτονικής το όνομα *structural* αφού θα κάνετε περιγραφή δομής στη γλώσσα VHDL. Στη συνέχεια, προαιρετικά δηλώστε τα ports στο **I/O Port Definitions**. As υποθέσουμε ότι αρχικά δηλώνουμε τα ports του αθροιστή με καταχωρητές εισόδου και εξόδου των 8 bit που έχει εισόδους CLK, RESET, WE (για έγκριση εγγραφής στον καταχωρητή εισόδων του αθροιστή), A[7:0] και B[7:0], καθώς και εξόδους S[7:0], Cout και OV (όλες registered). Εισάγετε το όνομα του σήματος ή της αρτηρίας στο port name. Για τις αρτηρίες επιλέξτε επιπλέον το Bus και ορίστε την τιμή του MSB και του LSB. Πατήστε το **+** για δήλωση επιπλέον ports με τον ίδιο τρόπο. Τέλος, πατήστε **OK**.



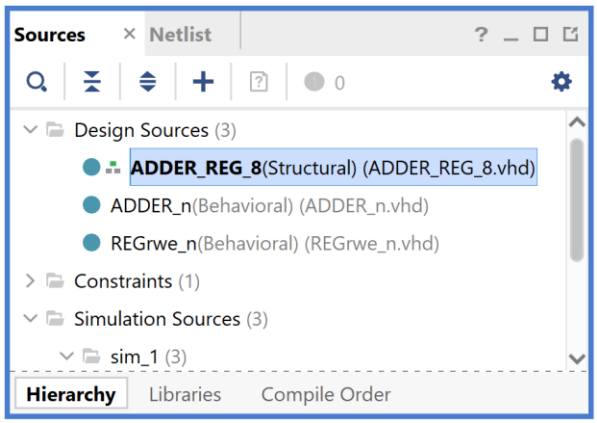
Εικόνα 40 – Δήλωση οντότητας, αρχιτεκτονικής και διεπαφής οντότητας

- Επιβεβαιώστε τη δημιουργία του αρχείου **ADDER_REG_8.vhd** στο παράθυρο *Sources* του PROJECT MANAGER (επίσης φαίνεται το όνομα της οντότητας **ADDER_REG_8** και το όνομα της αρχιτεκτονικής **Structural**). Η οντότητα **ADDER_REG_8** αποθηκεύεται και στο design source set *sources_1* και στο simulation source set *sim_1* του project DSD_LAB1. Παρατηρείστε ότι η οντότητα **ADDER_n** παραμένει ως η κορυφαία οντότητα της ιεραρχίας (**top**) των design sources, επί του παρόντος.



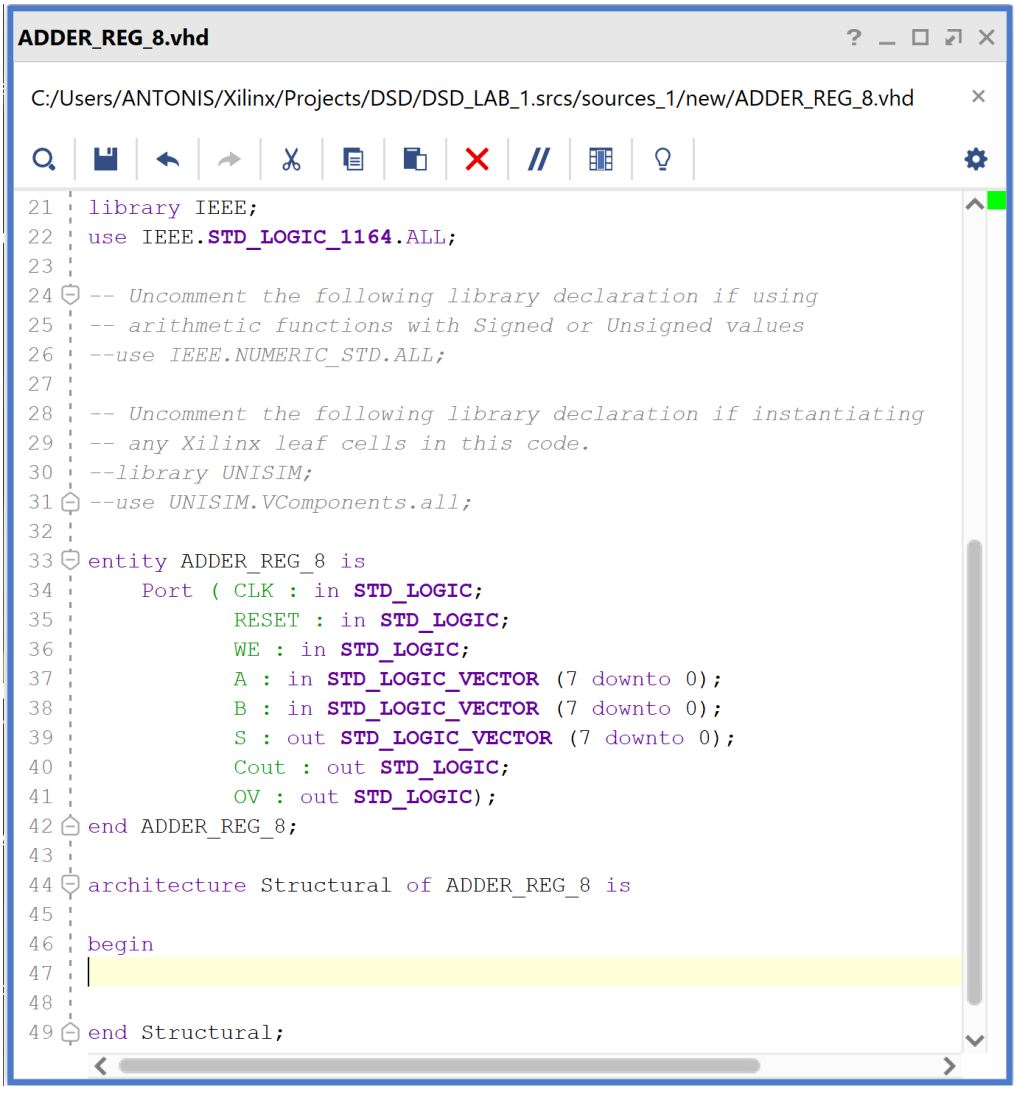
Εικόνα 41 - Διαθέσιμα sources

- Για να ορισθεί η οντότητα **ADDER_REG_8** ως η κορυφαία οντότητα της ιεραρχίας (**top**) των design sources, θα πρέπει να κάνετε δεξί κλικ επί του ονόματος στο παράθυρο *Sources* του PROJECT MANAGER και να επιλέξετε το **Set as Top**. Σε αυτό το σημείο, οι 3 οντότητες (τα 3 design sources) είναι ανεξάρτητα μεταξύ τους.



Εικόνα 42 - Ορισμός top-level entity

- Ανοίξτε το πηγαίο αρχείο **ADDER_REG_8.vhd** με διπλό κλικ στο επιλεγμένο αρχείο. Τα ports της οντότητας έχουν ήδη ορισθεί, αλλά λείπει ο ορισμός της αρχιτεκτονικής της οντότητας. Μπορούμε να αυξήσουμε το μέγεθος του παραθύρου *ADDER_ REG_8.vhd*, είτε σε πλάτος με την επιλογή του **maximize** , είτε με τη δημιουργία νέου παραθύρου με την επιλογή του **float** . Επιλέξτε το **float**.



Εικόνα 43 – Περιεχόμενα πηγαίου αρχείου ADDER_REG_8.vhd στην αρχική του μορφή

Παρατηρείστε ότι το *use IEEE.NUMERIC_STD.ALL* εμφανίζεται σε γραμμή σχολίου, που πρέπει να αφαιρέσετε με την επιλογή του **Toggle Line Comments (//)**.

- Συμπληρώστε την αρχιτεκτονική της οντότητας **ADDER_REG_8**. Στο τμήμα δηλώσεων της αρχιτεκτονικής (πριν το begin): δηλώστε τα components **ADDER_n** και **REGrwe_n** (με αντιγραφή των ports από τα αντίστοιχα entities) και τις εσωτερικές αρτηρίες **A_in[7:0]**, **B_in[7:0]**, **S_in[7:0]** και **FlagsI_in[1:0]**, **FlagsO_in[1:0]**. Στο σώμα της αρχιτεκτονικής (μετά το begin) περιγράψτε τις διασυνδέσεις των 5 components που απαρτίζουν την οντότητα **ADDER_REG_8** με τις **5 αντίστοιχες ταυτόχρονες εντολές στοιχείων**, που διαθέτουν και generic map και port map. Τέλος πατήστε **Save File**.

ADDER_REG_8.vhd

C:/Users/ANTONIS/Xilinx/Projects/DSD/DSD_LAB_1.srcs/sources_1/new/ADDER_REG_8.vhd

```
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx leaf cells in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity ADDER_REG_8 is
34      Port (
35          CLK    :  in STD_LOGIC;
36          RESET :  in STD_LOGIC;
37          WE     :  in STD_LOGIC;
38          A      :  in STD_LOGIC_VECTOR (7 downto 0);
39          B      :  in STD_LOGIC_VECTOR (7 downto 0);
40          S      : out STD_LOGIC_VECTOR (7 downto 0);
41          Cout   : out STD_LOGIC;
42          OV     : out STD_LOGIC);
43  end ADDER_REG_8;
44
45  architecture Structural of ADDER_REG_8 is
46      component ADDER_n
47          generic (WIDTH : positive := 4); -- set 4 as default value
48          port (
49              A      :  in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
50              B      :  in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
51              S      : out STD_LOGIC_VECTOR (WIDTH-1 downto 0);
52              Cout   : out STD_LOGIC;
53              OV     : out STD_LOGIC);
54      end component;
55
56      component REGrwe_n
57          generic (WIDTH : positive := 4); -- set 4 as default value
58          port (
59              CLK    :  in STD_LOGIC;
60              RESET :  in STD_LOGIC;
61              WE     :  in STD_LOGIC;
62              Din    :  in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
63              Dout   : out STD_LOGIC_VECTOR (WIDTH-1 downto 0));
64      end component;
```

Εικόνα 44 – Περιεχόμενα πηγαίου αρχείου ADDER_REG_8.vhd μετά τις συμπληρώσεις

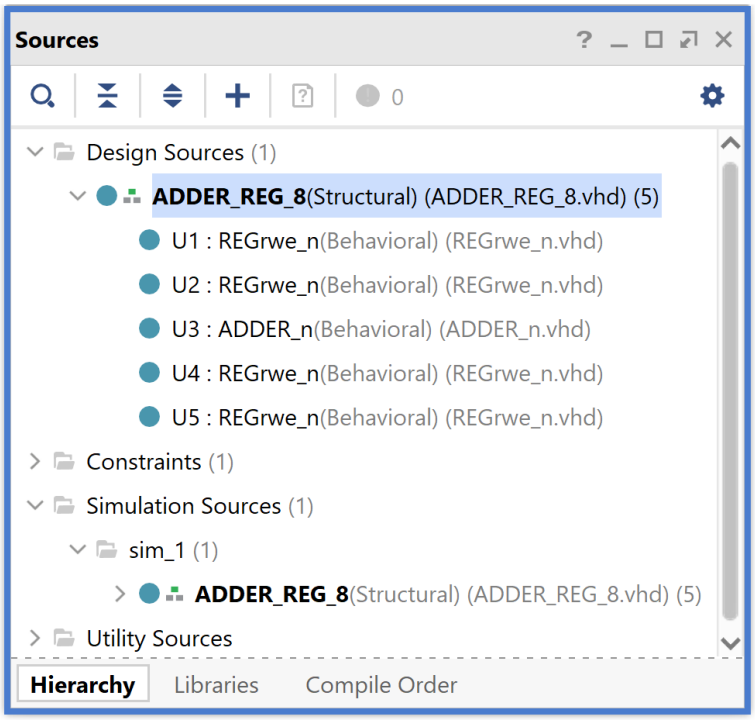
ADDER_REG_8.vhd

C:/Users/ANTONIS/Xilinx/Projects/DSD/DSD_LAB_1.srcs/sources_1/new/ADDER_REG_8.vhd

```
66      signal  A_in      : STD_LOGIC_VECTOR (7 downto 0);
67      signal  B_in      : STD_LOGIC_VECTOR (7 downto 0);
68      signal  S_in      : STD_LOGIC_VECTOR (7 downto 0);
69
70      signal  FlagsI_in : STD_LOGIC_VECTOR (1 downto 0);
71      signal  FlagsO_in : STD_LOGIC_VECTOR (1 downto 0);
72
73  begin
74
75      U1: REGrwe_n
76          generic map (WIDTH => 8)
77          port map (
78              CLK   => CLK,
79              RESET => RESET,
80              WE    => WE,
81              Din   => A,
82              Dout  => A_in);
83
84      U2: REGrwe_n
85          generic map (WIDTH => 8)
86          port map (
87              CLK   => CLK,
88              RESET => RESET,
89              WE    => WE,
90              Din   => B,
91              Dout  => B_in);
92
93      U3: ADDER_n
94          generic map (WIDTH => 8)
95          port map (
96              A     => A_in,
97              B     => B_in,
98              S     => S_in,
99              Cout  => FlagsI_in(0),
100             OV    => FlagsI_in(1));
101
102     U4: REGrwe_n
103         generic map (WIDTH => 8)
104         port map (
105             CLK   => CLK,
106             RESET => RESET,
107             WE    => '1',
108             Din   => S_in,
109             Dout  => S);
110
111     U5: REGrwe_n
112         generic map (WIDTH => 2)
113         port map (
114             CLK   => CLK,
115             RESET => RESET,
116             WE    => '1',
117             Din   => FlagsI_in,
118             Dout  => FlagsO_in);
119
120     Cout <= FlagsO_in(0);
121     OV   <= FlagsO_in(1);
122
123 end Structural;
```

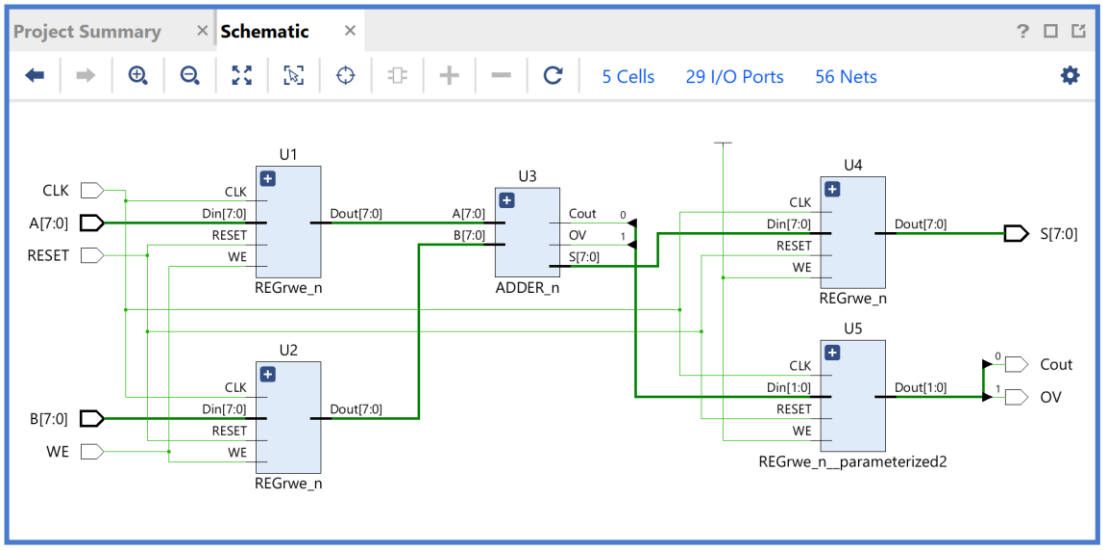
Εικόνα 45 – Περιεχόμενα πηγαίου αρχείου ADDER_REG_8.vhd μετά τις συμπληρώσεις

- Παρατηρείστε το παράθυρο *Sources*, όπου έχει δημιουργηθεί αυτόματα η ιεραρχική δομή της οντότητας **ADDER_REG_8**. Επιλέξτε το επίπεδο της ιεραρχίας που επιθυμείτε να βλέπετε με κατάλληλη επιλογή πάνω στα **>** και ∨ (στην εικόνα φαίνονται δύο διαφορετικές επιλογές). Οι οντότητες **REGrwe_n** και **ADDER_n** έχουν ενταχθεί στην ιεραρχική δομή και δεν είναι πλέον ανεξάρτητες.



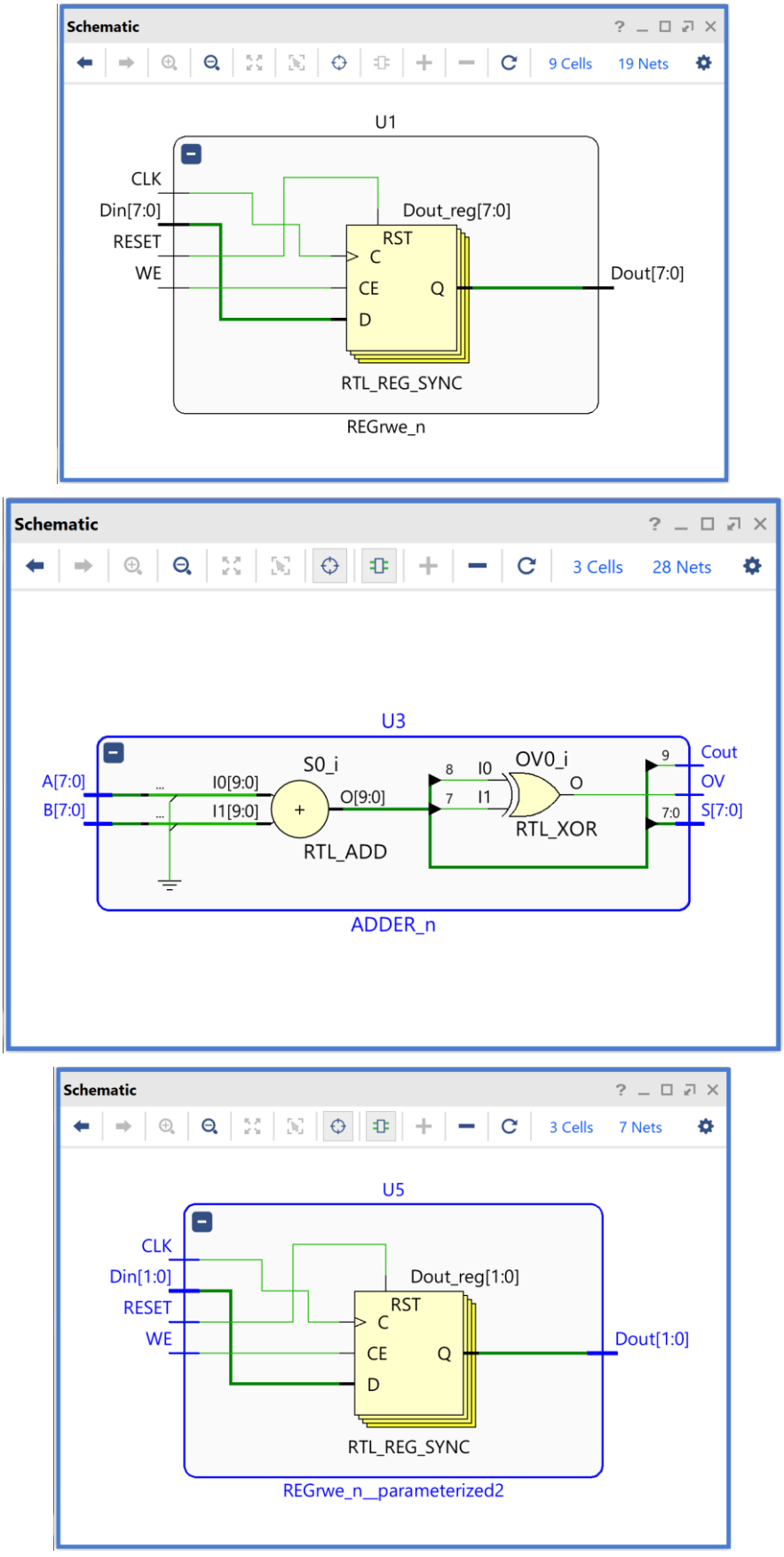
Εικόνα 46 – Η σχηματισμένη πλέον ιεραρχική δομή στο παράθυρο Sources

- Με επιλογή του **Reload** εκτελέστε τη διαδικασία της ανάλυσης RTL της οντότητας **ADDER_REG_8**. Εμφανίζεται στο νέο παράθυρο *Schematic* το σχηματικό διάγραμμα στο επίπεδο RTL του αθροιστή των 8 bit με καταχωρητές εισόδου και εξόδου (ADDER_REG_8) που χρησιμοποιεί ως στοιχεία (components) τις οντότητες (entities) **REGrwe_n (U1, U2, U4, U5)** και **ADDER_n (U3)**, που ήδη έχετε δημιουργήσει.



Εικόνα 47 – Σχηματικό διάγραμμα στο επίπεδο RTL του κυκλώματος ADDER_REG_8

Επιβεβαιώστε το elaborated design στο πιο κάτω ιεραρχικά επίπεδο με διπλό κλικ πάνω στα επιλεγμένα components U1 – U5. Τα components U1, U2 και U4 είναι ίδια.



Εικόνα 48 – Σχηματικά διαγράμματα στο επίπεδο RTL των components του ADDER_REG_8

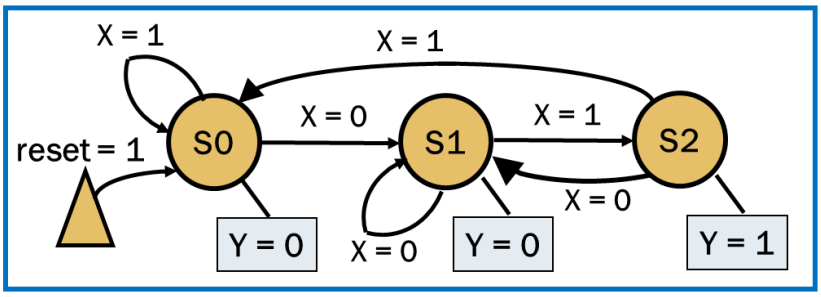
1.7.5 Δημιουργία του πηγαίου αρχείου PATTERN_FSM.VHD στο VIVADO IDE

Αυτό το αρχείο περιγράφει στη γλώσσα VHDL τον ανιχνευτή ακολουθίας 2 διαδοχικών bit ως μηχανή FSM τύπου Moor με βάση τον κώδικα που δίδεται στις σελίδες 324 – 330 του Κεφαλαίου 4 «Γλώσσες περιγραφής υλικού – Πλήρης έκδοση» των παραδόσεων του μαθήματος.

Ο ανιχνευτής ακολουθίας, πέραν των εισόδων CLK και RESET, έχει μία σειριακή είσοδο X και μία σειριακή έξοδο Y, που γίνεται 1 κάθε φορά που τα δύο τελευταία διαδοχικά bit του X είναι 01. Η συγκεκριμένη μηχανή FSM έχει τρεις καταστάσεις τύπου Moore ως εξής:

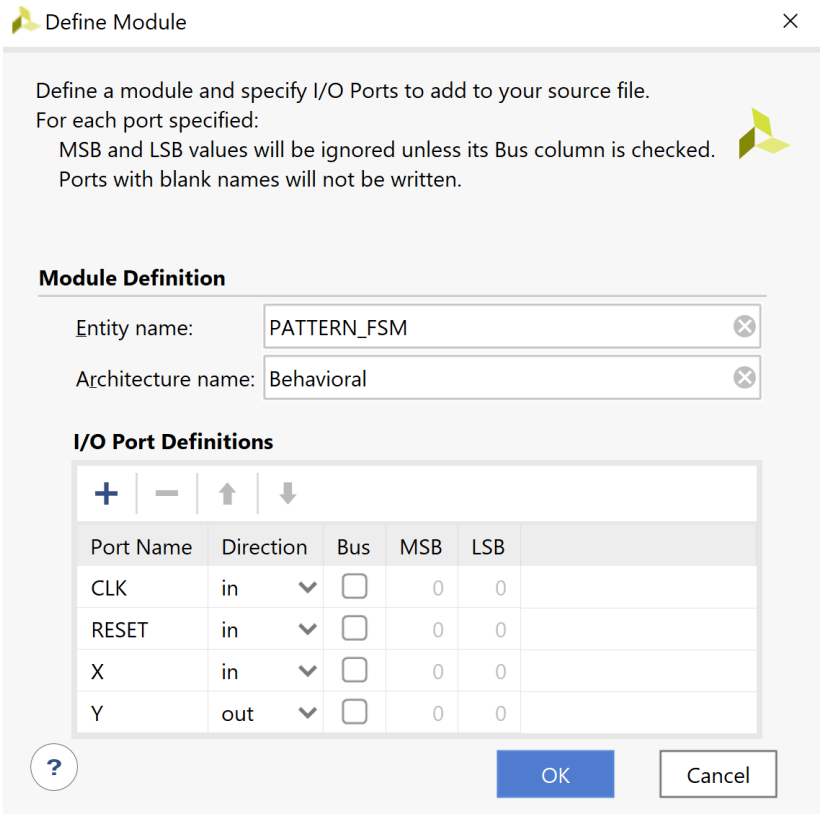
- S0 = αρχική κατάσταση, δεν έχει ανιχνευθεί κανένα ψηφίο, Y = 0,
- S1 = έχει ανιχνευθεί στην είσοδο X ένα bit 0, Y = 0,
- S2 = έχουν ανιχνευθεί στην είσοδο X δύο διαδοχικά bit 01, Y = 1.

Ο ανιχνευτής ακολουθίας υλοποιεί το ακόλουθο διάγραμμα μεταβολής κατάστασης:



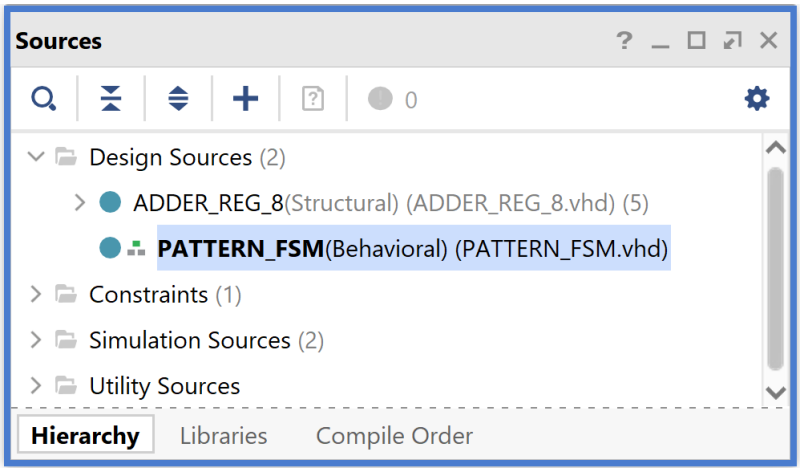
Εικόνα 49 – Διάγραμμα καταστάσεων της FSM

- Πατήστε το **+** στο παράθυρο *Sources* του PROJECT MANAGER για να ξεκινήσετε τον wizard δημιουργίας του πηγαίου αρχείου (source) του ανιχνευτή **PATTERN_FSM**.
- Στο παράθυρο διαλόγου *Add Sources* επιλέξτε το Add or create design sources και πατήστε **Next**.
- Στο παράθυρο διαλόγου *Add or Create Design Sources* πατήστε την επιλογή **Create File** για τη δημιουργία του νέου design source file που θα τοποθετηθεί στο ήδη καθορισμένο design source set *sources_1*.
- Στο παράθυρο διαλόγου *Create Source File* δηλώστε το όνομα του αρχείου VHD (π.χ. **PATTERN_FSM**) και πατήστε **OK**.
- Επιστρέφετε στο παράθυρο διαλόγου *Add or Create Design Sources*, όπου φαίνεται ότι έχει δημιουργηθεί το αρχείο **PATTERN_FSM.vhd** και έχει συμπεριληφθεί στα αρχεία του project που ονομάζεται DSD_LAB_1. Πατήστε **Finish**.
- Εμφανίζεται το παράθυρο διαλόγου *Define Module*, όπου σας παρέχεται η δυνατότητα της δήλωσης του ονόματος της οντότητας, του ονόματος της αρχιτεκτονικής και των ports της οντότητας. Ως όνομα της οντότητας διατηρείστε το ίδιο όνομα που είχατε δώσει στο αρχείο (**PATTERN_FSM**). Επιλέξτε ως όνομα της αρχιτεκτονικής το όνομα *behavioral* αφού θα κάνετε περιγραφή συμπεριφοράς στη γλώσσα VHDL. Στη συνέχεια, προαιρετικά δηλώστε τα ports στο **I/O Port Definitions** (**CLK**, **RESET**, **X**, **Y**). Τέλος, πατήστε **OK**.



Εικόνα 50 – Δήλωση οντότητας, αρχιτεκτονικής και διεπαφής οντότητας

- Επιβεβαιώστε τη δημιουργία του αρχείου **PATTERN_FSM.vhd** στο παράθυρο *Sources* του PROJECT MANAGER (επίσης φαίνεται το όνομα της οντότητας **PATTERN_FSM** και το όνομα της αρχιτεκτονικής **Behavioral**). Η οντότητα αυτή είναι αποθηκευμένη και στο design source set *sources_1* και στο simulation source set *sim_1* του project DSD_LAB1. Παρατηρείστε ότι η οντότητα **ADDER_REG_8** παραμένει ως η κορυφαία οντότητα της ιεραρχίας (**top**) των design sources, επί του παρόντος.
- Για να ορισθεί η οντότητα **PATTERN_FSM** ως η κορυφαία οντότητα της ιεραρχίας (**top**) των design sources, θα πρέπει να κάνετε δεξί κλικ επί του ονόματος στο παράθυρο *Sources* του PROJECT MANAGER και να επιλέξετε το **Set as Top**.



Εικόνα 51 – Ορισμός του PATTERN_FSM ως κορυφαία οντότητα

- Ανοίξτε το πηγαίο αρχείο **PATTERN_FSM.vhd** με διπλό κλικ στο επιλεγμένο αρχείο. Τα ports της οντότητας έχουν ήδη ορισθεί, αλλά λείπει ο ορισμός της αρχιτεκτονικής της οντότητας.
- Συμπληρώστε την αρχιτεκτονική της οντότητας **PATTERN_FSM** με βάση τον κώδικα που δίδεται στις σελίδες 324 – 330 του Κεφαλαίου 4 «Γλώσσες περιγραφής υλικού – Πλήρης έκδοση» των παραδόσεων του μαθήματος. Οι καταστάσεις FSM_states περιγράφονται με τύπο απαρίθμησης ως S0, S1 και S2. Η είσοδος X αρχικά αποθηκεύεται σε D Flip–Flop που **τοποθετείται στο 1** με το σήμα RESET, ώστε να παγιδεύεται η μηχανή στην κατάσταση S0, όσο το σήμα RESET είναι ενεργό και μέχρι να εμφανιστεί η πρώτη είσοδος στην έξοδο του D Flip–Flop (X_in). Η μηχανή FSM περιγράφεται με δύο process: ένα σύγχρονο process (SYNC), που περιγράφει τον καταχωρητή κατάστασης που αρχικοποιείται στην κατάσταση S0 με το σήμα RESET, και ένα ασύγχρονο process (ASYNC), που περιγράφει τη λογική επόμενης κατάστασης και τη λογική εξόδου σύμφωνα με το διάγραμμα μεταβολής κατάστασης και υποστηρίζει ασφαλή λειτουργία στην περίπτωση βλάβης (fail–safe). Τέλος πατήστε **Save File**.

PATTERN_FSM.vhd

C:/Users/ANTONIS/Xilinx/Projects/DSD/DSD_LAB_1.srcs/sources_1/new/PATTERN_FSM.vhd

```
34  entity PATTERN_FSM is
35      Port ( CLK : in STD_LOGIC;
36             RESET : in STD_LOGIC;
37             X : in STD_LOGIC;
38             Y : out STD_LOGIC);
39  end PATTERN_FSM;
40
41  architecture Behavioral of PATTERN_FSM is
42
43  -- state definition
44      type FSM_states is
45      (S0, S1, S2);
46
47  -- internal signals
48      signal current_state, next_state: FSM_states;
49      signal X_in : STD_LOGIC; -- Only when there is an INREG
50
51  begin
52
53  -- Optional for sychronization
54      INREG:  process (CLK)
55      begin
56          if (CLK = '1' and CLK'event) then
57              if (RESET = '1') then X_in <= '1'; -- to trap state S0 during reset
58              else X_in <= X;
59              end if;
60          end if;
61      end process;
62
63  -- Common process for all FSMs to create state register
64      SYNC:  process (CLK)
65      begin
66          if (CLK = '1' and CLK'event) then
67              if (RESET = '1') then current_state <= S0;
68              else current_state <= next_state;
69              end if;
70          end if;
71      end process;
```

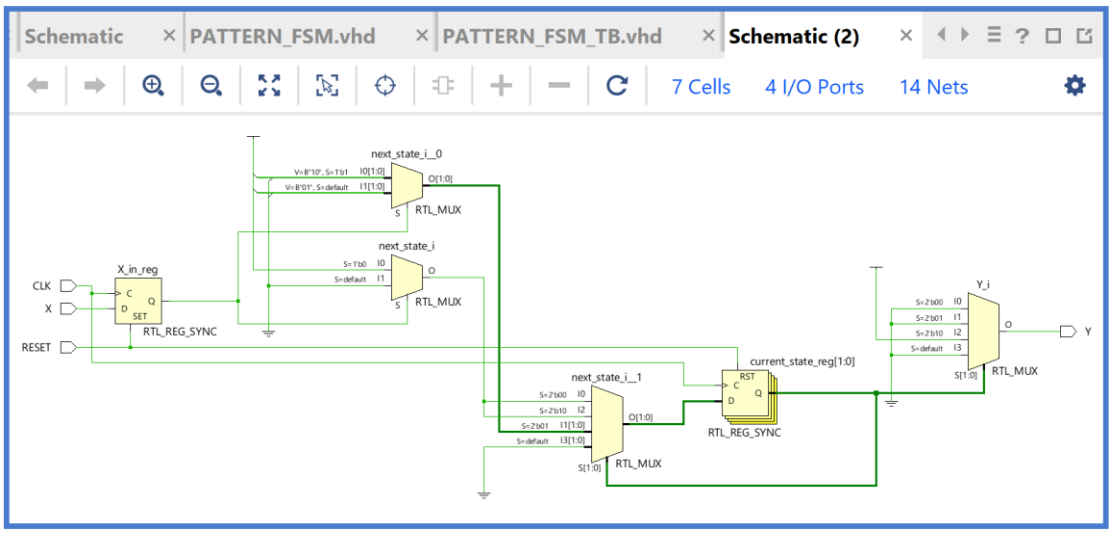
Εικόνα 52 – Περιεχόμενα πηγαίου αρχείου PATTERN_FSM.vhd (συνεχίζεται)

PATTERN_FSM.vhd

C:/Users/ANTONIS/Xilinx/Projects/DSD/DSD_LAB_1.srcs/sources_1/new/PATTERN_FSM.vhd

```
73      -- Process to create next state logic and output logic
74      ASYNC: process (current_state, X_in)   -- Moore
75      begin
76      -- FSM next state and output initialization
77          next_state <= S0;
78          Y <= '0';
79          case current_state is
80              when S0 =>
81                  if (X_in = '0') then next_state <= S1;
82                  else next_state <= S0;
83                  end if;
84              when S1 =>
85                  if (X_in = '1') then next_state <= S2;
86                  else next_state <= S1;
87                  end if;
88              when S2 => Y <= '1';
89                  if (X_in = '0') then next_state <= S1;
90                  else next_state <= S0;
91                  end if;
92              -- fail-safe behavior
93              when others => next_state <= S0;
94          end case;
95      end process;
96
97  end Behavioral;
```

- Με επιλογή του **Reload** εκτελέστε τη διαδικασία της ανάλυσης RTL του νέου πηγαίου αρχείου **PATTERN_FSM.vhd**. Εμφανίζεται στο νέο παράθυρο *Schematic* το σχηματικό διάγραμμα στο επίπεδο RTL του ανιχνευτή ακολουθίας 2 διαδοχικών bit ως μηχανή τύπου Moore.

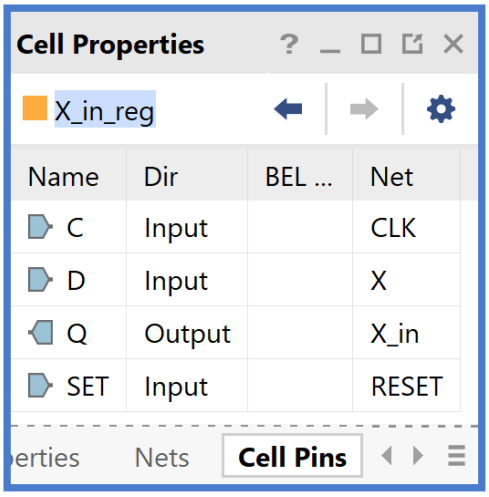


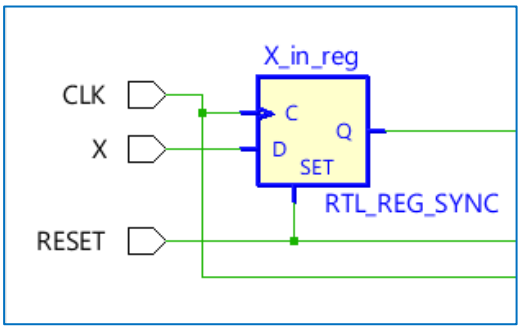
Εικόνα 53 – Σχηματικό διάγραμμα στο επίπεδο RTL του κυκλώματος PATTERN_FSM

Επιβεβαιώστε το elaborated design. Μελετήστε την πληροφορία που παρέχεται στο παράθυρο *Cell Properties*, κάτω από το παράθυρο *Sources*. Αναλύστε χωριστά κάθε ένα από τα cells που φαίνονται παράθυρο schematic, με κατάλληλη επιλογή στα leaf cells είτε του παράθυρου *Netlist*, είτε του παραθύρου schematic. Παρατηρείστε ότι οι εντολές *if* και *case* της γλώσσας VHDL, που περιγράφουν συνδυαστική λογική, υλοποιούνται με πολυπλέκτες στο σχηματικό διάγραμμα στο επίπεδο RTL.

D Flip–Flop εισόδου X (**X_in_reg**):

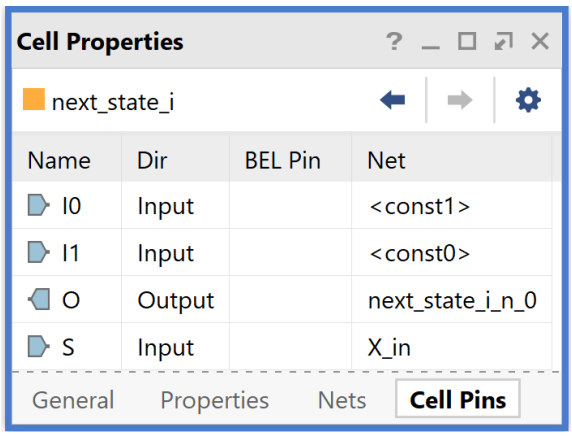
```
if (RESET = '1') then X_in <= '1'; else X_in <= X;
```

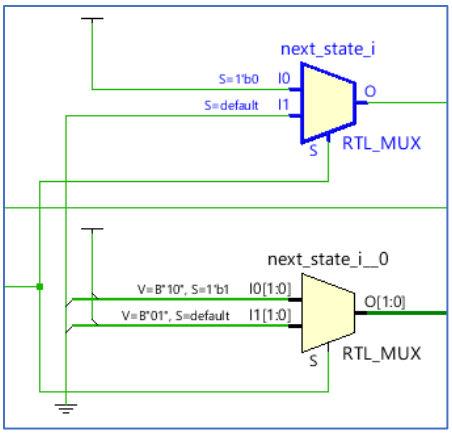




Πολυπλέκτης 2 σε 1 λογικής NOT (**next_state_i**):

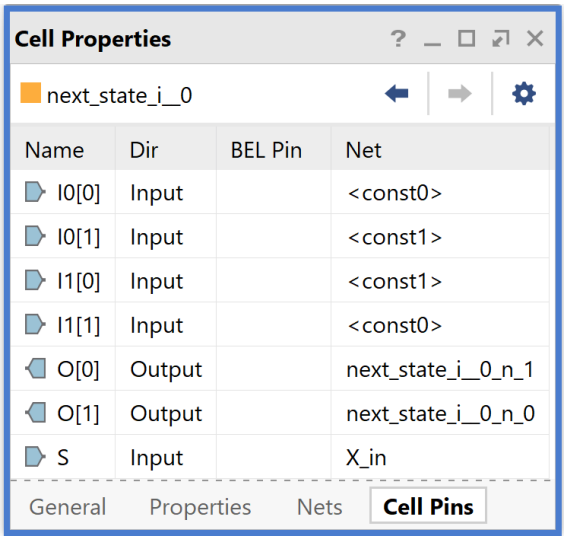
```
next_state_i_n_0 = not X_in
```

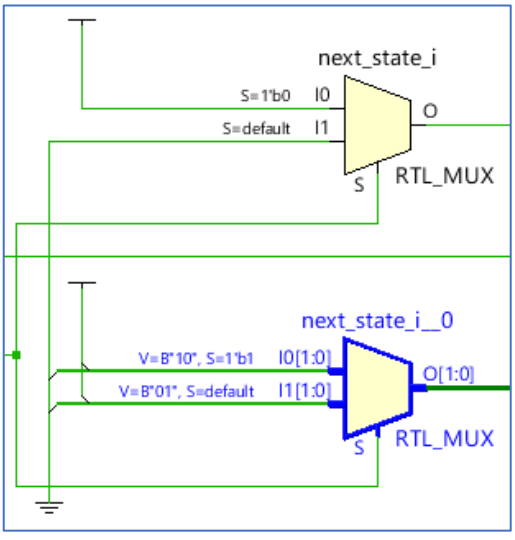




Πολυπλέκτης 2 σε 1 λογικής επόμενης κατάστασης (**next_state_i_0**):

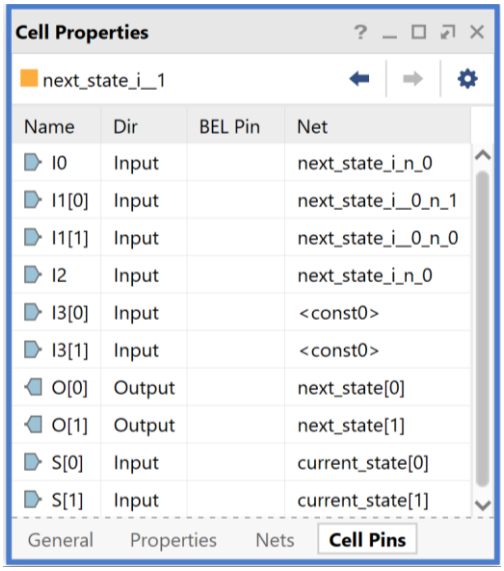
```
If (X_in = '1') then next_state_i_0 <= S2 = "10"
                else next_state_i_0 <= S1 = "01"
```

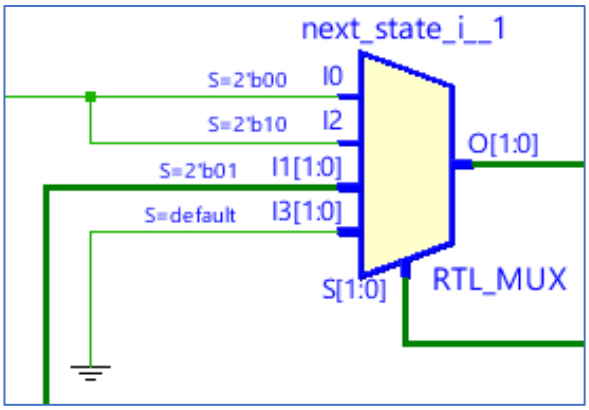




Πολυπλέκτης 4 σε 1 λογικής επόμενης κατάστασης (**next_state_i__1**):

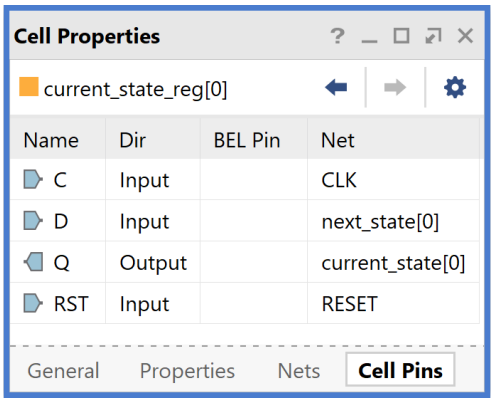
```
when (current_state = S0 = "00") =>
    next_state_i_n_0 = not X_in;
    next_state <= '0'&next_state_ i_n_0;
    If (X_in = '1') then next_state <= S0 = "00"
                    else next_state <= S1 = "01"
when (current_state = S2 = "10") =>
    next_state_i_n_0 = not X_in;
    next_state <= '0'&next_state_ i_n_0;
    If (X_in = '1') then next_state <= S0 = "00"
                    else next_state <= S1 = "01"
when (current_state = S1 = "01") =>
    next_state <= next_state_i_0;
    If (X_in = '1') then next_state <= S2 = "10"
                    else next_state <= S1 = "01"
when (current_state = "11") =>
    next_state <= S0 = "00"; (fail-safe operation)
```

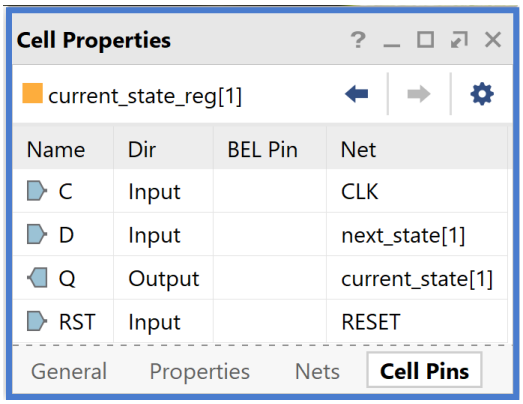


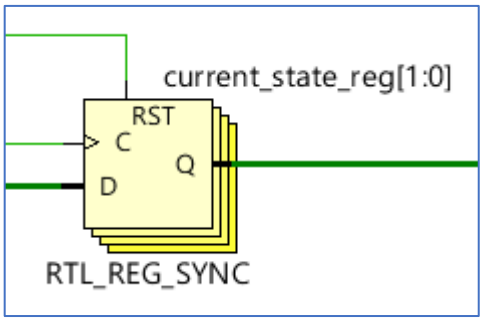


Καταχωρητής κατάστασης των 2 bit (**current_state_reg_0/current_state_reg_1**):

```
if (RESET = '1') then current_state <= S0 = "00";
                 else current_state <= next_state;
```

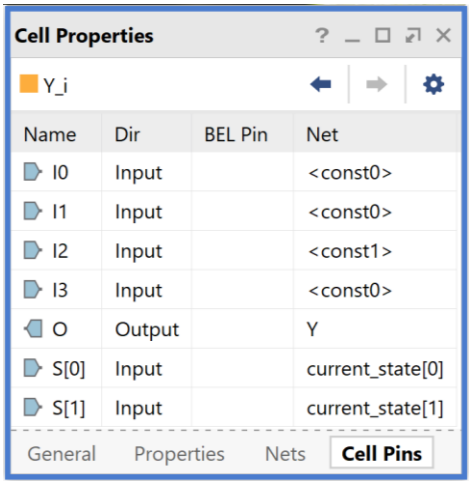


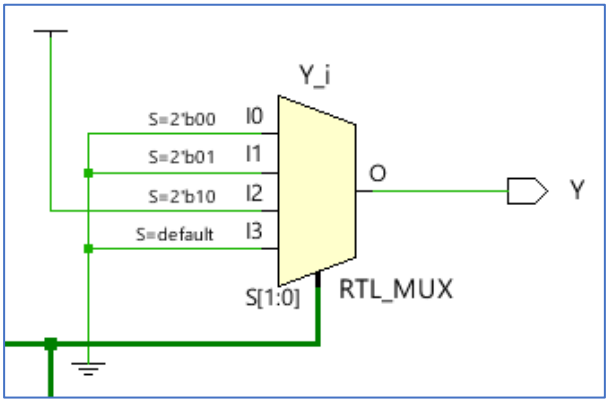




Πολυπλέκτης 2 σε 1 λογικής εξόδου:

```
if (current_state = S2 = "10") then Y <= '1'; else Y <= '0';
```



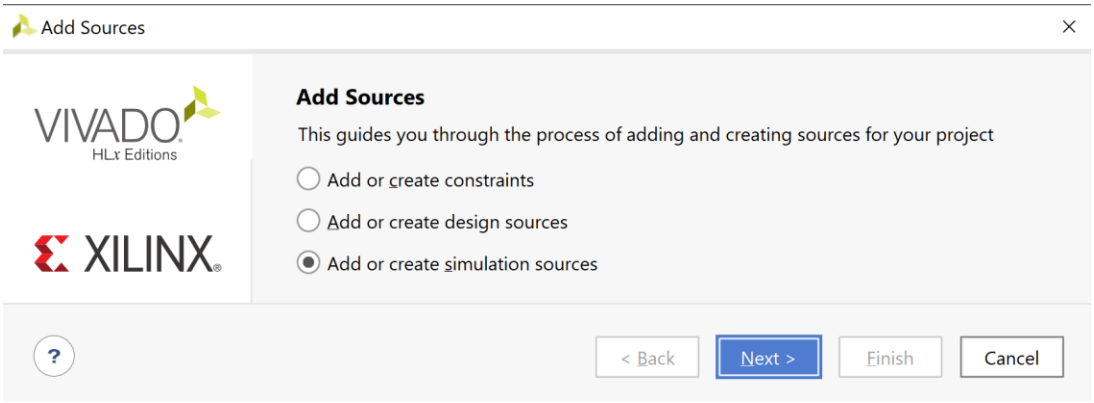


Η διαδικασία που ήδη περιγράψαμε στο Βήμα 2 «**Εισαγωγή του κώδικα VHDL και ανάλυση στο επίπεδο RTL**» εφαρμόζεται παρομοίως σε κάθε πιθανή υπομονάδα συνδυαστικής ή ακολουθιακής λογικής του επεξεργαστή.

1.8 Βήμα 3: Εισαγωγή του VHDL testbench και προσομοίωση συμπεριφοράς

1.8.1 Δημιουργία προγράμματος δοκιμών (testbench) τύπου VHD στο VIVADO IDE για συνδυαστική λογική ανάμεσα σε καταχωρητές

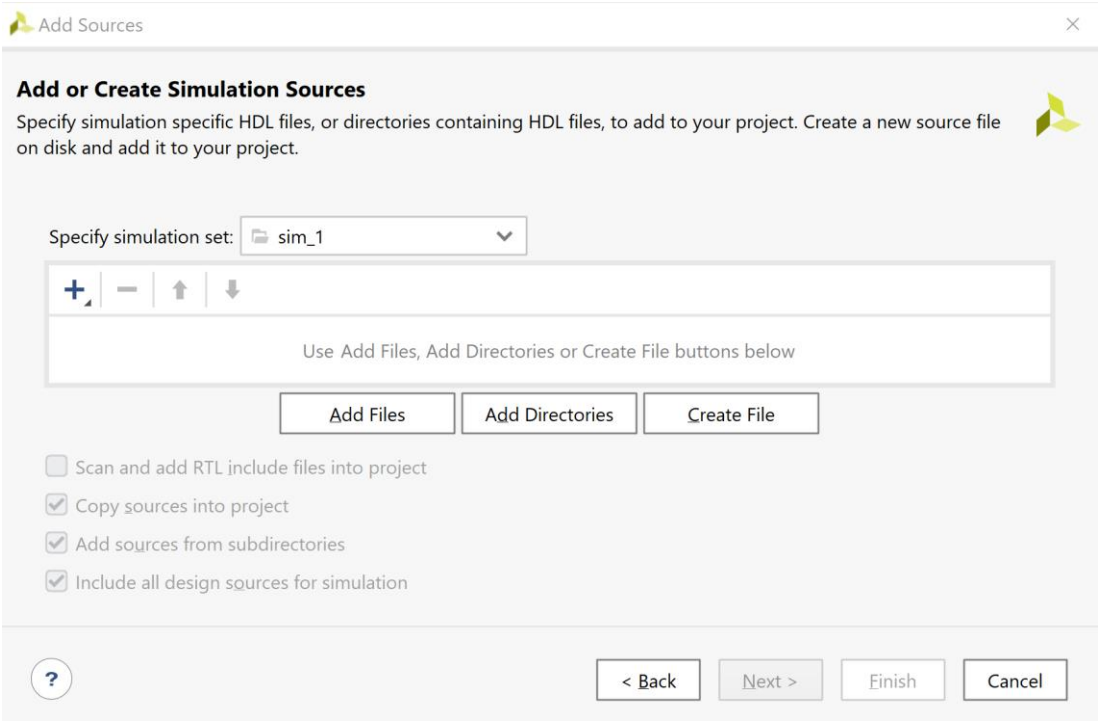
- Πατήστε το **+** στο παράθυρο *Sources* του PROJECT MANAGER για να ξεκινήσετε τον wizard δημιουργίας ενός νέου αρχείου (source).
- Στο παράθυρο διαλόγου *Add Sources* επιλέξτε το Add or create simulation sources, ώστε να δημιουργήσετε ένα νέο αρχείο προσομοίωσης (testbench) στη γλώσσα VHDL (simulation source file) τύπου VHD. Σε αυτό το αρχείο εισάγουμε τον κώδικα VHDL που περιγράφει τη λειτουργία του testbench. Πατήστε **Next**.



Εικόνα 54 – Πρώτο βήμα για την προσθήκη νέου πηγαίου αρχείου προσωμοίωσης

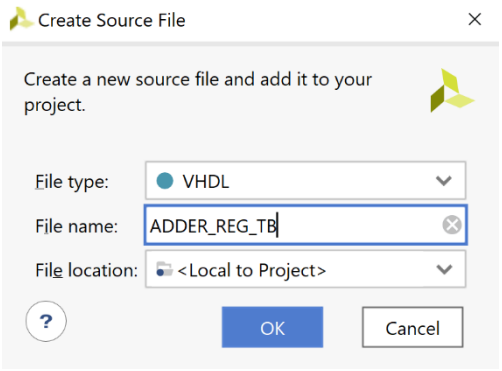
- Στο παράθυρο διαλόγου *Add or Create Simulation Sources* πατήστε την επιλογή **Create File** για τη δημιουργία του design simulation file που θα τοποθετηθεί στο ήδη καθορισμένο design source set *sources_1*.

  Εάν απαιτείται να καθοριστεί νέο simulation set (π.χ. sim_2), αρχικά δημιουργείστε το και στη συνέχεια ενεργοποιήστε το (επιλογή στο make active).

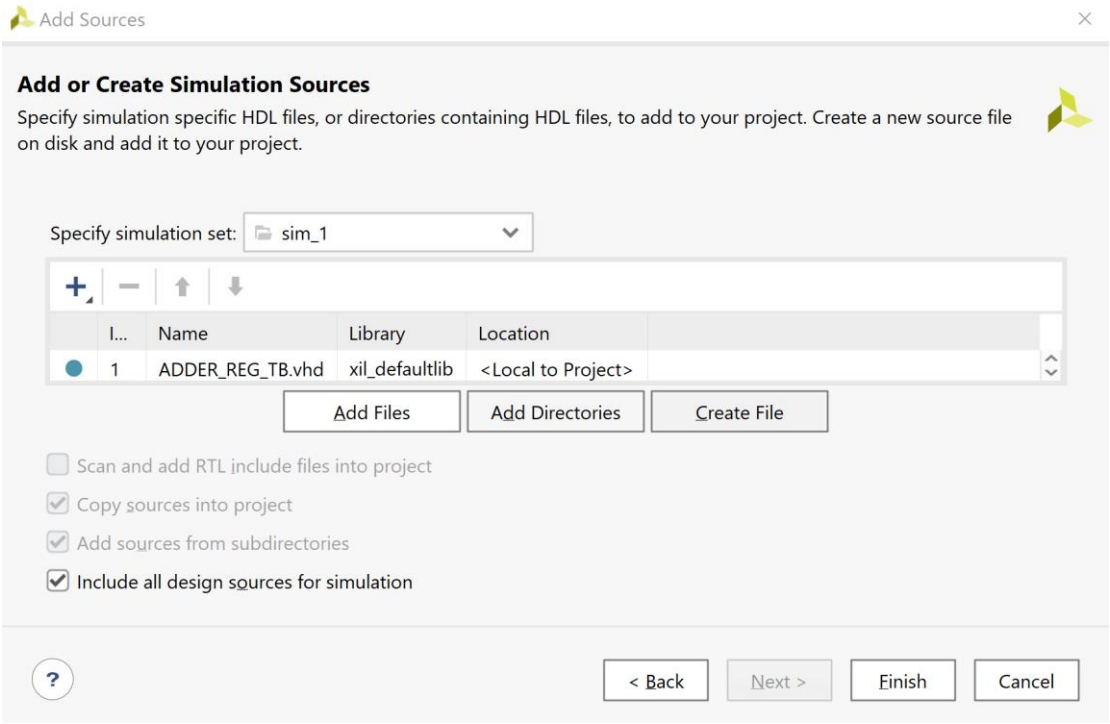


Εικόνα 55 – Επιλογή για προσθήκη νέου πηγαίου αρχείου προσομοίωσης

- Στο παράθυρο διαλόγου *Create Source File* δηλώστε το όνομα του testbench αρχείου (_TB) VHD (π.χ. **ADDER_REG_8_TB**), αφού σε αυτό το αρχείο θα περιγράψετε στη γλώσσα VHDL την οντότητα **ADDER_REG_8_TB** που απαιτείται για τις ανάγκες της προσομοί–ωσης της οντότητας **ADDER_REG_8**. Πατήστε **OK**.
- Επιστρέφετε στο παράθυρο διαλόγου *Add or Create Simulation Sources*, όπου φαίνεται ότι έχει δημιουργηθεί το αρχείο **ADD_REG_8_TB.vhd** και έχει συμπεριληφθεί στα αρχεία του project που ονομάζεται DSD_LAB_1. Διατηρείστε την επιλογή Include all design sources for simulation και πατήστε **Finish**.

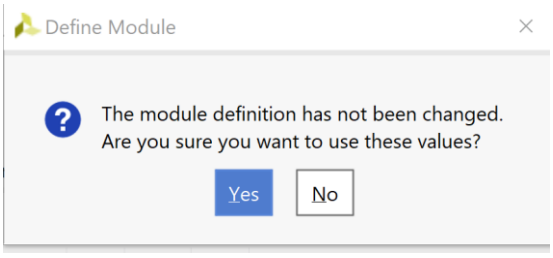


Εικόνα 56 – Δημιουργία αρχείου



Εικόνα 57 – Ολοκλήρωση διαδικασίας δημιουργίας πηγαίου αρχείου προσομοίωσης

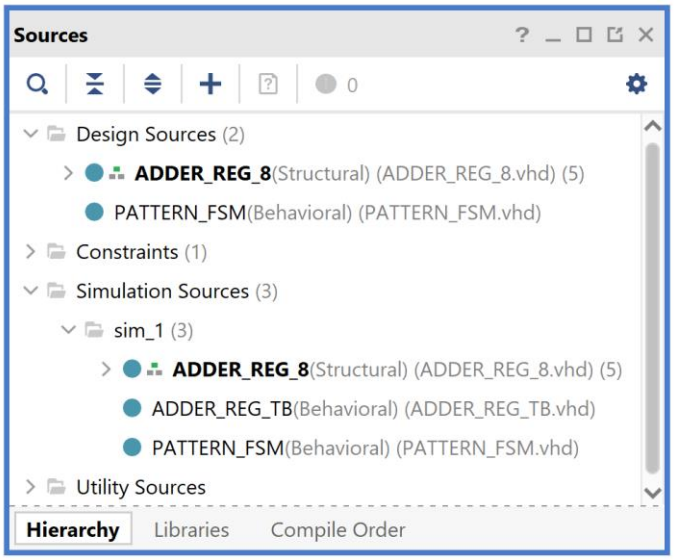
- Εμφανίζεται το παράθυρο διαλόγου *Define Module*, όπου σας παρέχεται η δυνατότητα της δήλωσης του ονόματος της οντότητας (παραμένει **ADDER_REG_8_TB**), του ονόματος της αρχιτεκτονικής (επιλέξτε *behavioral*). Αυτή η οντότητα δεν έχει ports. Τέλος, πατήστε **OK**.
- Εμφανίζεται το παράθυρο προειδο–ποίησης *Define Module* που προειδο–ποιεί για τη μη δήλωση των I/O ports. Πατήστε **Yes**.



Εικόνα 58 - Παράκαμψη δήλωσης ports

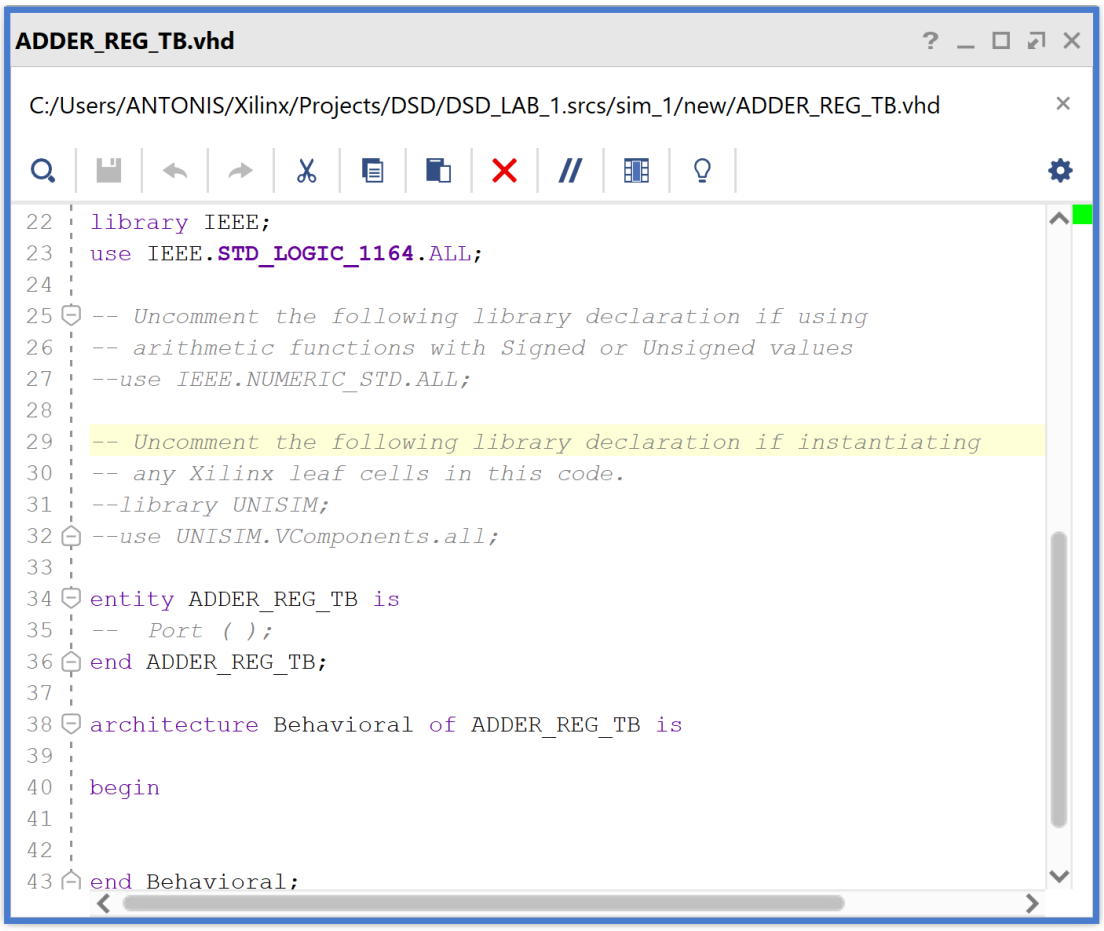
- Επιβεβαιώστε τη δημιουργία του προγράμματος δοκιμών (testbench) **ADDER_REG_8_TB .vhd** στο παράθυρο *Sources* του PROJECT MANAGER (επί–σης φαίνονται τα ονόματα της οντότητας **ADDER_REG_8_TB** και της αρχιτεκτονικής της **Behavioral**). Το αρχείο αυτό είναι αποθηκευμένο μόνο στο simulation source set *sim_1* του project DSD_LAB1.

  Η οντότητα **ADDER_REG_8** παραμένει ως η κορυφαία οντότητα της ιεραρχίας (**top**) των simulation resources επί του παρόντος.



Εικόνα 59 - Διαθέσιμα sources

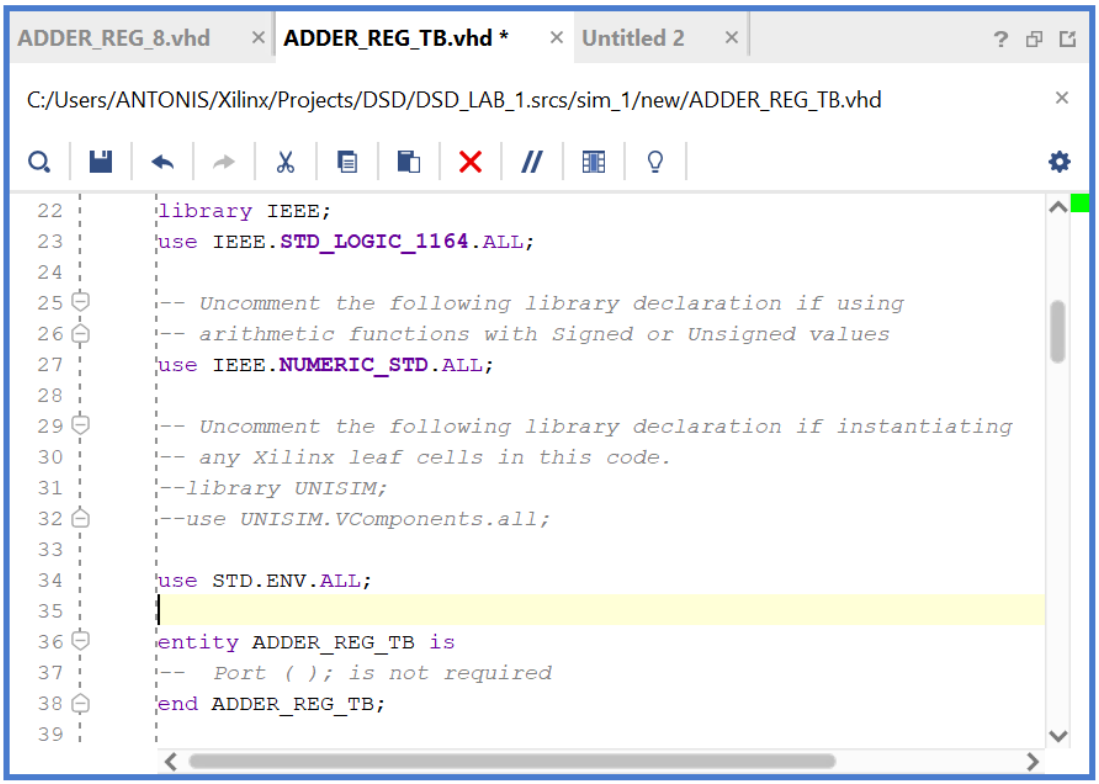
- Ανοίξτε το αρχείο **ADDER_REG_8_TB.vhd** με διπλό κλικ στο επιλεγμένο αρχείο. Λείπει ο ορισμός της αρχιτεκτονικής της οντότητας.



Εικόνα 60 – Αρχικά περιεχόμενα πηγαίου αρχείου PATTERN_FSM.vhd

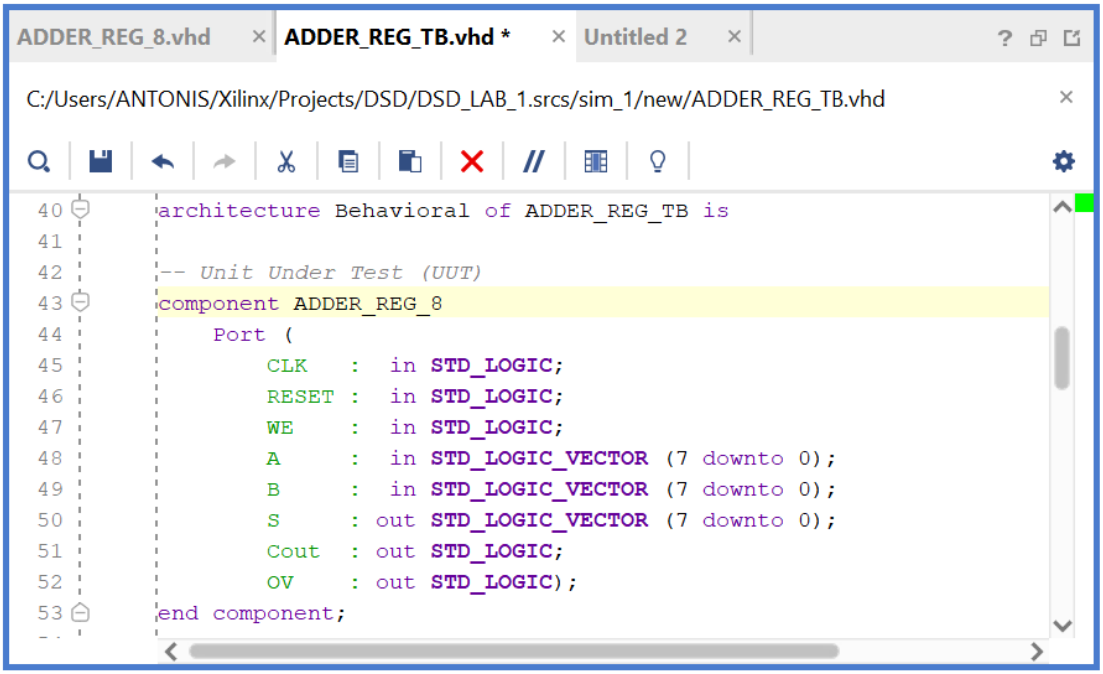
- Συμπληρώστε το πρόγραμμα δοκιμών (testbench) **ADDER_REG_8_TB.vhd** και στο τέλος πατήστε **Save File**, αφού πρώτα ακολουθήσετε τα παρακάτω βήματα:

  Βήμα 1: Ενεργοποιείστε τα πακέτα: **IEEE.NUMERIC_STD.ALL** και **STD.ENV.ALL**.



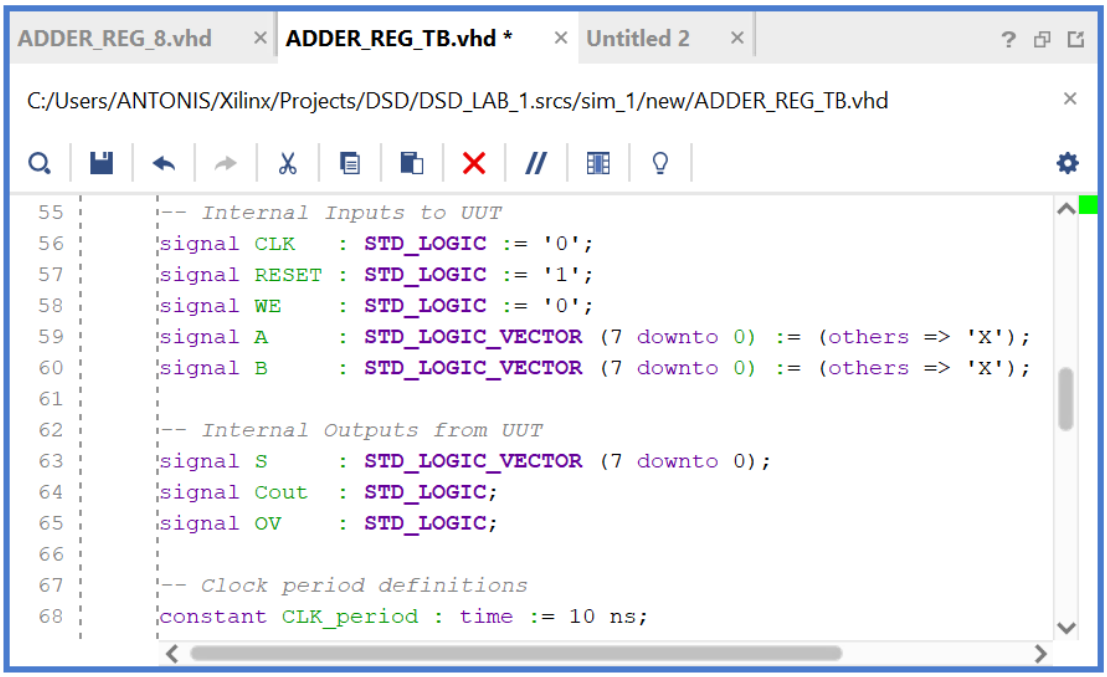
Εικόνα 61 – Ενεργοποίηση των πακέτων NUMERIC_STD και STD.ENV

Βήμα 2: Στις δηλώσεις της αρχιτεκτονικής αρχικά προσθέστε το **component ADDER_REG_8** (προκύπτει από το **entity ADDER_REG_8** με τις κατάλληλες τροποποιήσεις) που θα είναι και το *Unit Under test (UUT)* της οντότητας **ADDER_REG_8_TB** του συγκεκριμένου προγράμματος δοκιμών (testbench).



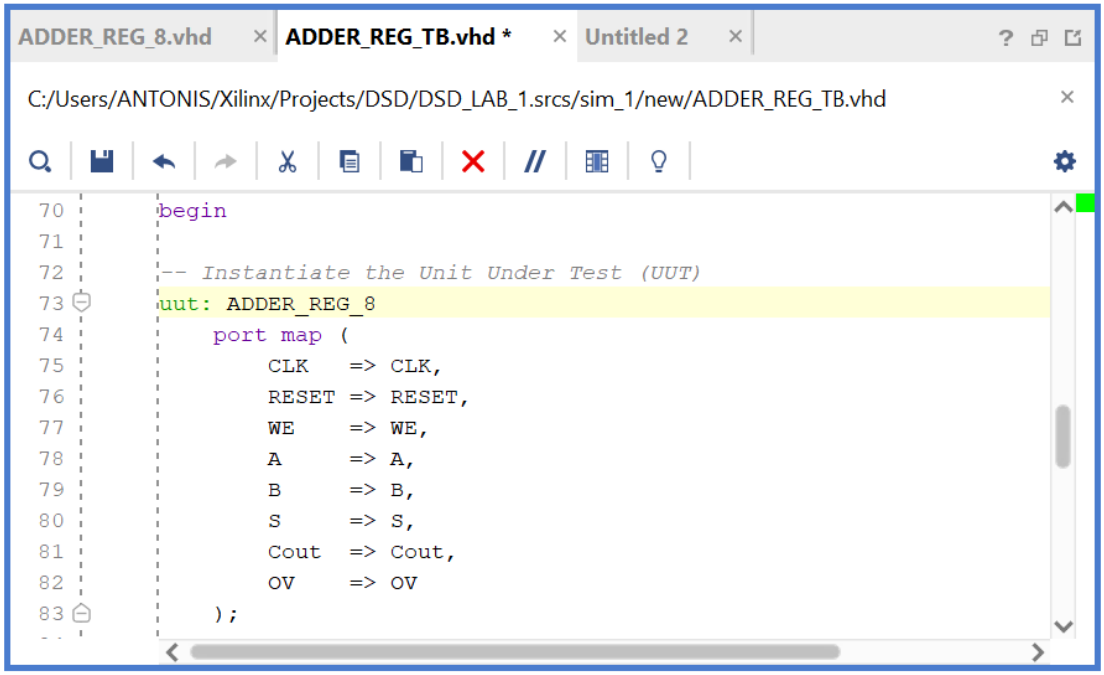
Εικόνα 62 – Προσθήκη του component ADDER_REG_8

Βήμα 3: Στη συνέχεια προσθέστε τα εσωτερικά σήματα που θα χρησιμοποιηθούν ως είσοδοι και έξοδοι του *UUT*. Τέλος, ορίστε την περίοδο του CLK (έχουμε επιλέξει αρχικά τα **10 ns** λαμβάνοντας υπόψη το σήμα του ρολογιού της κάρτας που είναι στα 100 MHz). Μπορείτε να προσαρμόσετε την περίοδο του CLK στις απαιτήσεις της δικής σας υλοποίησης της σχεδίασης, για τις ανάγκες τις προσομοίωσης.



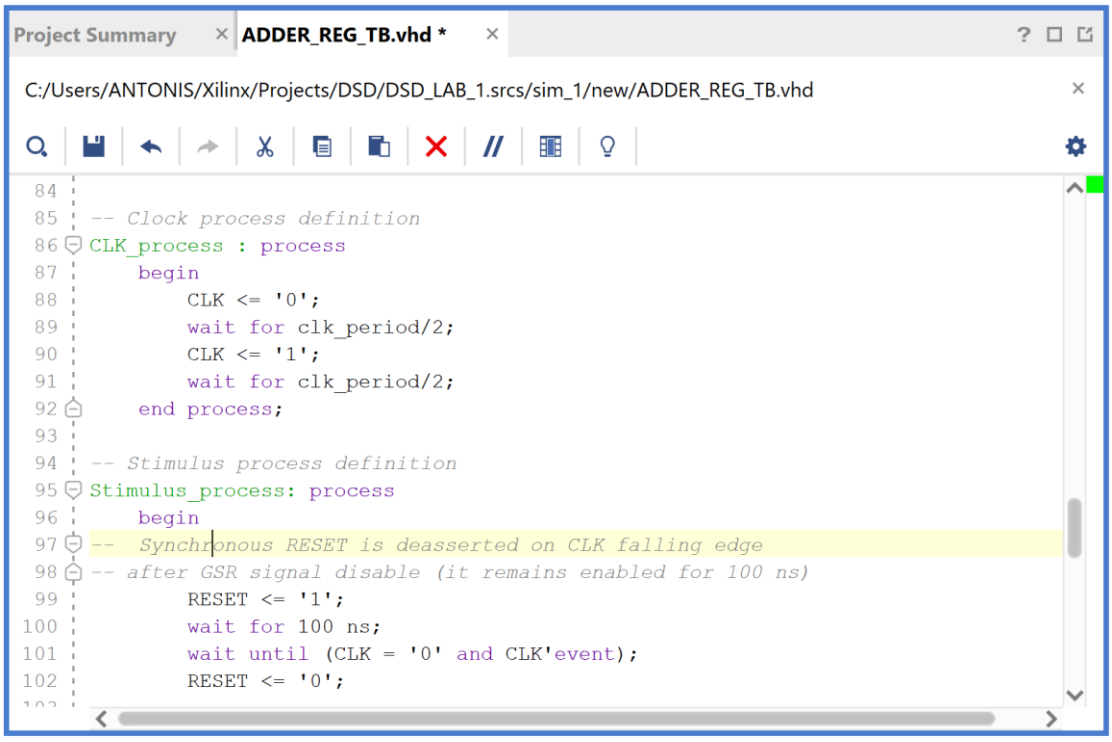
Εικόνα 63 – Προσθήκη εσωτερικών σημάτων στο αρχείο δοκιμών

Βήμα 4: Στο σώμα της αρχιτεκτονικής μετά το begin, αρχικά, συνδέστε το *UUT* (**ADDER_REG_8**) με την οντότητα **ADDER_REG_8_TB** διατηρώντας τα ίδια ονόματα στα σήματα. Ιεραρχικά πλέον το *UUT* θα εμφανίζεται κάτω από τη συγκεκριμένη οντότητα.



Εικόνα 64 – Προσθήκη και σύνδεση του UUT με το υπόλοιπο αρχείο δοκιμών

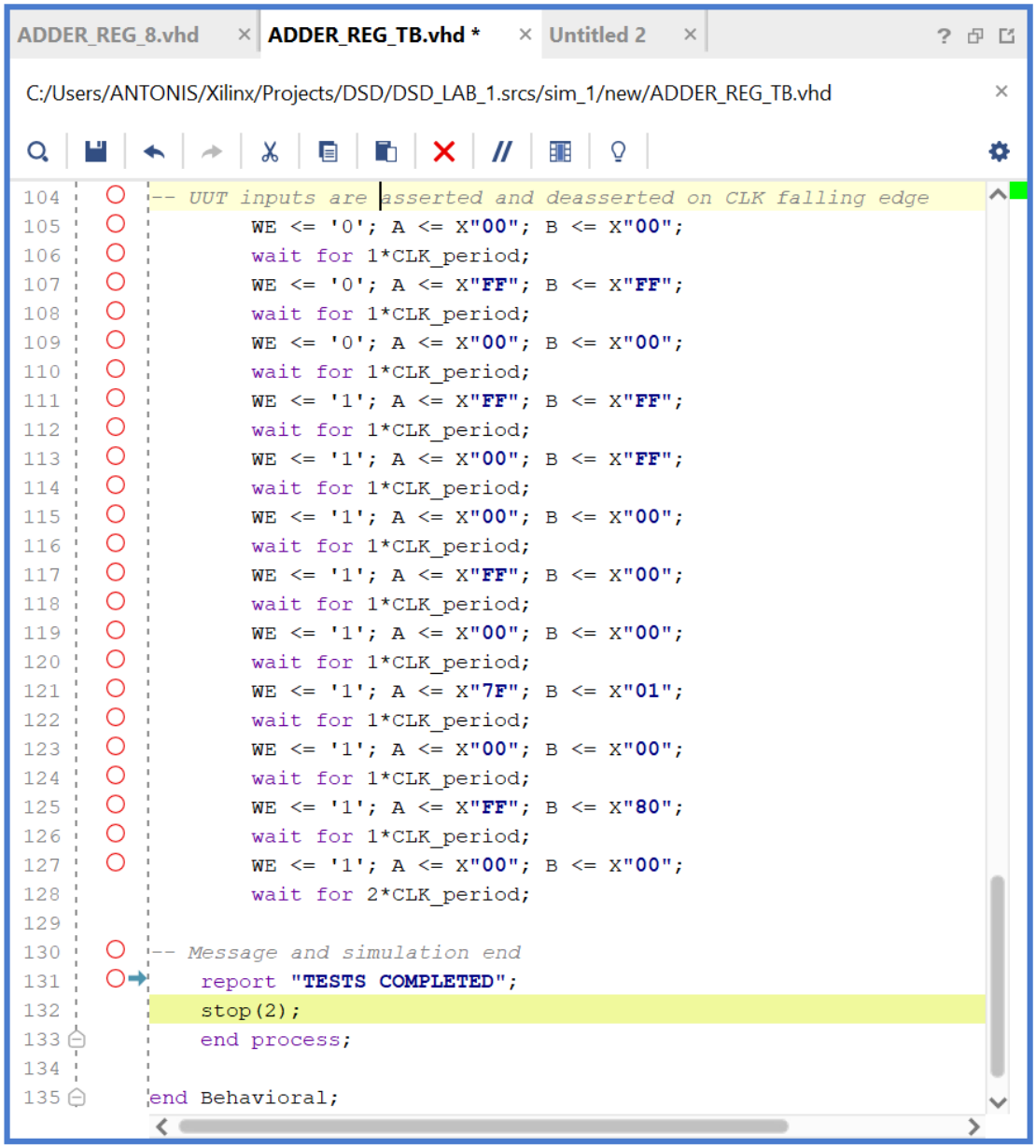
Βήμα 5: Στη συνέχεια περιγράψτε τη συμπεριφορά του CLK (στο *CLK_process*) και τη συμπεριφορά του σήματος RESET (στην αρχή του *Stimulus_process*). Για να υπάρχει συμβατότητα σε όλα τα είδη των προσομοιώσεων έχουμε επιλέξει το σήμα RESET (με τη εντολή *wait for 100 ns*) να παραμένει ενεργό για τουλάχιστον 100 ns, όσο 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 (testbench) κατά τη διάρκεια της προσομοίωσης των synthesized design models και implemented design models. Επιπλέον, έχουμε επιλέξει (με την εντολή *wait until (CLK = '0' and CLK'event)*) η απενεργοποίηση του σήματος RESET να γίνεται στην κατερχόμενη ακμή του CLK, ώστε να μην δημιουργούνται παραβιάσεις στους χρόνους σταθεροποίησης (set–up) και διατήρησης (hold) των καταχωρητών, ανεξάρτητα από την περίοδο του CLK.



Εικόνα 65 – Περιγραφή συμπεριφοράς σημάτων RESET και CLK

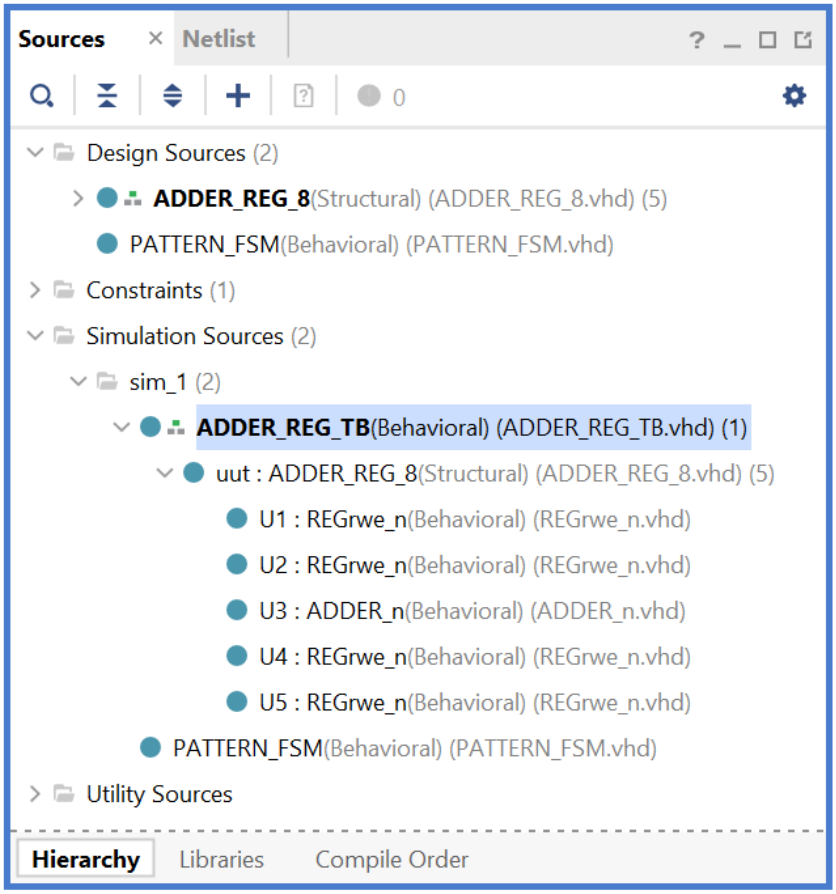
Βήμα 6: Στη συνέχεια, στο ίδιο *Stimulus_process*, περιγράψτε τις εισόδους δοκιμής που θα λάβει το *UUT* κατά τη διάρκεια της προσομοίωσης. Οι αναθέσεις τιμών σε όλες τις εισόδους του *UUT* γίνονται στην κατερχόμενη ακμή του CLK, ώστε να μην δημιουργούνται παραβιάσεις στους χρόνους σταθεροποίησης (set–up) και διατήρησης (hold) των καταχωρητών, ανεξάρτητα από την περίοδο του CLK.

Τέλος, δηλώστε μήνυμα ολοκλήρωσης της δοκιμής και διακοπής της προσομοίωσης με την εντολή *stop(2)*.



Εικόνα 66 – Περιγραφή εισόδων δοκιμής στο UUT

Βήμα 7: Επιβεβαιώστε την ιεραρχία της οντότητας του προγράμματος δοκιμών (testbench) **ADDER_REG_8_TB** σε σχέση με το *UUT* του (που είναι η οντότητα **ADDER_REG_8**) στο παράθυρο *Sources* του PROJECT MANAGER. Το αρχείο αυτό είναι αποθηκευμένο μόνο στο simulation source set *sim_1* του project DSD_LAB1 και πλέον αυτόματα η οντότητα **ADDER_REG_8_TB** έχει ήδη ορισθεί ως η κορυφαία οντότητα της ιεραρχίας (**top**) των simulation resources. Εάν δεν έχει γίνει αυτόματα, ορίστε το, οι ίδιοι. Πριν συνεχίσετε επιβεβαιώστε ότι η κορυφαία οντότητα της ιεραρχίας (**top**) των design resources είναι η οντότητα **ADDER_REG_8**.



Εικόνα 67 – Επιβεβαίωση της ιεραρχίας του προγράμματος δοκιμών

1.8.2 Εκτέλεση προσομοίωσης συμπεριφοράς στο VIVADO IDE για συνδυαστική λογική ανάμεσα σε καταχωρητές

Η προσομοίωση συμπεριφοράς εκτελείται στην οντότητα **ADDER_REG_8_TB** του προγράμματος δοκιμών (testbench) που έχει ορισθεί ως η κορυφαία οντότητα της ιεραρχίας (**top**) των simulation resources. Κατά την προσομοίωση, η οντότητα **ADDER_REG_8_TB** καλεί το *UUT* της (που είναι η οντότητα **ADDER_REG_8**).

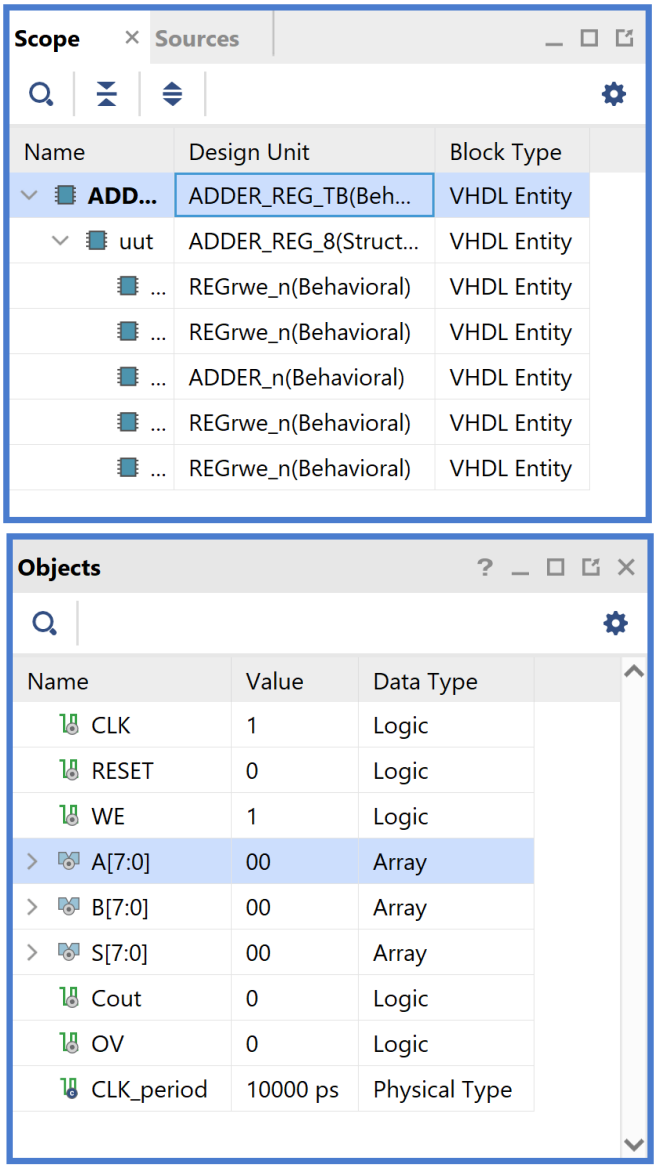
- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Run Simulation**, ώστε να εμφανιστούν όλες οι πιθανές προσομοιώσεις που υποστηρίζει το Vivado IDE. Επιλέξτε **Run Behavioral Simulation**.
- Το πρόγραμμα δοκιμής (testbench) και όλες οι οντότητες του *UUT* θα γίνουν compiled και θα τρέξει το Vivado simulator (εφόσον βέβαια δεν υπάρχουν σφάλματα). Θα εμφανιστεί το παράθυρο *SIMULATION* που απαρτίζεται από 4 παράθυρα:

  Το παράθυρο *Scope*, όπου παρουσιάζεται η οντότητα **ADDER_REG_8_TB**, η οντότητα **ADDER_REG_8** (UUT) καθώς και οι υπόλοιπες οντότητες του *UUT* σύμφωνα με την ιεραρχία που έχουμε ορίσει.

  Το παράθυρο Objects, όπου εμφανίζονται τα σήματα top–level, δηλαδή οι είσοδοι και οι έξοδοι της οντότητας **ADDER_REG_8**, που είναι η κορυφαία οντότητα της ιεραρχίας του *UUT*, καθώς και η περίοδος του CLK (*CLK_period*). Οι αρτηρίες (array) αναλύονται στα σήματα που τις απαρτί–ζουν. Οι τιμές αντιστοιχούν στις τιμές που έχει σταματήσει η προσομοίωση της οντότητας **ADDER_REG_8_TB** (*stop (2)*).

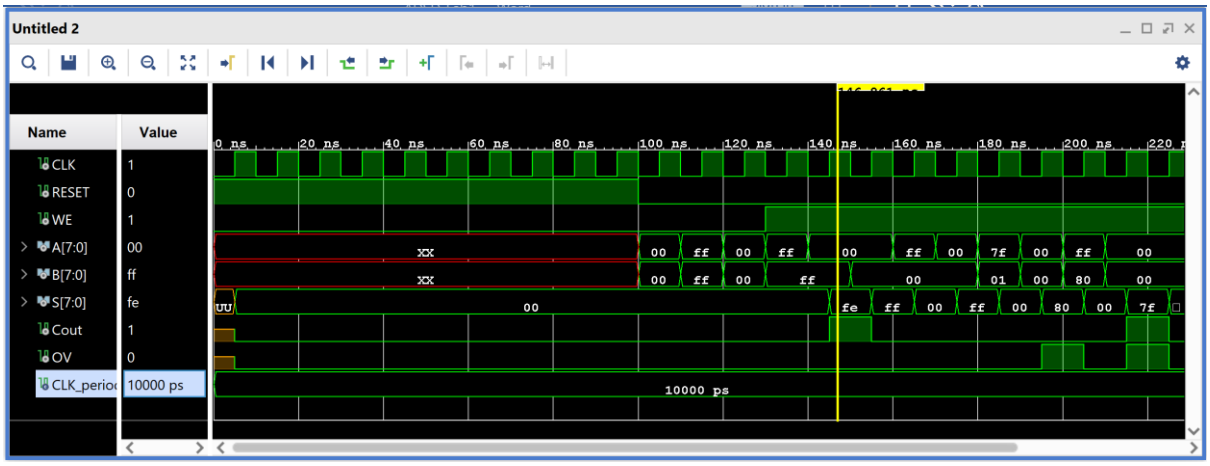
  Το παράθυρο *Tcl Console* με όλες τις διαδικασίες που εκτελούνται στο πλαίσιο της προσομοίωσης. Το Tcl Console καθαρίζει με την επιλογή *Clear*.

  Το παράθυρο με τα διαγράμματα χρονισμού (μέχρι να αποθηκευτεί είναι untitled). Εάν δεν είναι εμφανές το παράθυρο *Untitled*, επιλέξτε το *Untitled*. Βλέπετε ολόκληρο το διάγραμμα χρονισμού με κατάλληλο **zoom out** ή επιλέγοντας το **zoom fit** .



Εικόνα 68 - Παράθυρα scope και Objects

Για να επαναφέρετε το floating παράθυρο πίσω, απλά επιλέξτε το κουμπί Dock Window.



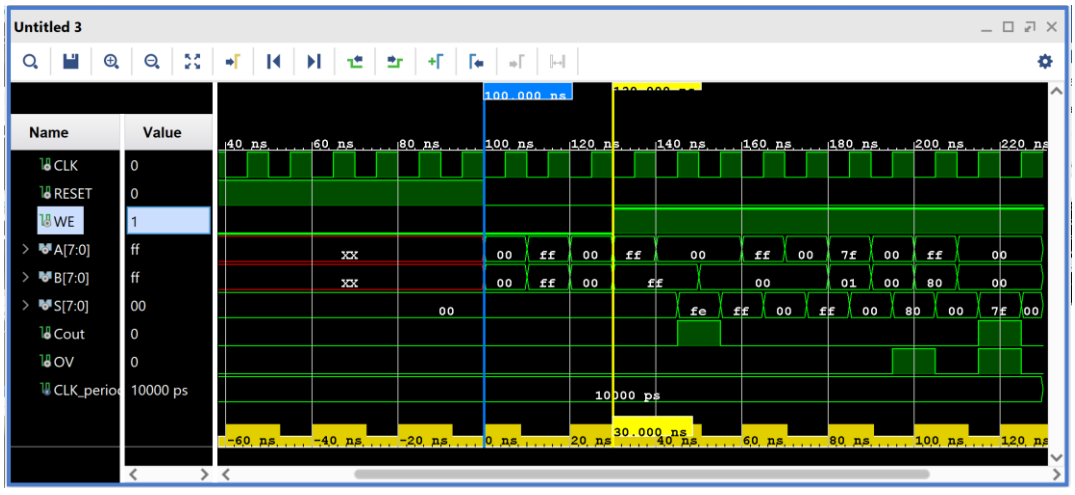
Εικόνα 69 – Παράθυρο με διάγραμμα χρονισμού

Πάνω απ' το παράθυρο με τα διαγράμματα χρονισμού, θα δείτε τα πιο κάτω κουμπιά:

Οι τιμές (στήλη value) που βλέπουμε δίπλα από τα σήματα αντιστοιχούν στη συγκεκριμένη χρονική στιγμή που δείχνει ο κίτρινος marker. Ο κίτρινος marker μετακινείται κάνοντας κλικ σε ένα συγκεκριμένο σήμα και σε μία συγκεκριμένη χρονική στιγμή.

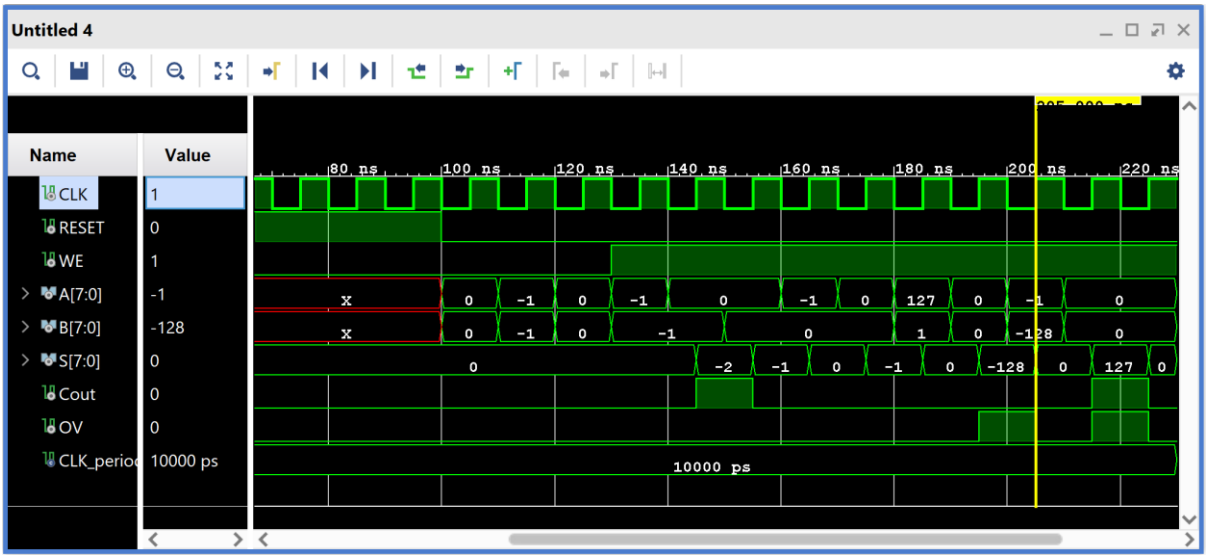
- Επιλέξτε το σήμα που επιθυμείτε να μελετήσετε και μετακινείστε κατάλληλα τον κίτρινο marker, πατώντας το αντίστοιχο κουμπί: α) στην αρχή της κυμματομορφής (0 ns), β) στο τέλος της κυμματομορφής (230 ns), γ) στην προηγούμενη αλλαγή τιμής του επιλεγμένου σήματος, ή δ) στην επόμενη τιμή του επιλεγμένου σήματος, ώστε να μελετήσετε τις χρονικές στιγμές που το επιλεγμένο σήμα αλλάζει τιμή.

  Προσθέστε και έναν δεύτερο marker, πατώντας το αντίστοιχο κουμπί, τον μπλε marker. Μετρήστε χρονικές αποστάσεις ανάμεσα στις αλλαγές τιμών δύο σημάτων. Για παράδειγμα, βάλτε τον μπλε marker στην κατερχόμενη ακμή του σήματος RESET και τον κίτρινο marker στην ανερχόμενη ακμή του σήματος WE. Απέχουν 30 ns.



Εικόνα 70 – Διάγραμμα χρονισμού με επιπλέον marker

- Αλλάξτε την τιμή στις αρτηρίες A[7:0], B[7:0], S[7:0] από δεκαεξαδική σε προσημασμένη δεκαδική. Κάντε δεξί κλικ πάνω σε κάθε αρτηρία και επιλέξετε το Radix. Στο παράθυρο που εμφανίζεται διαλέξτε Signed Decimal.



Εικόνα 71 – Διάγραμμα χρονισμού με διαφορετικό radix στις αρτηρίες

Παρατηρείστε ότι στην ανερχόμενη ακμή του CLK στο 200 ns, A = –1 και B = –128. Στην επόμενη ακμή του CLK εμφανίζεται η λανθασμένη τιμή στην έξοδο S = 127, Cout = 1, και OV = 1. Η καθυστέρηση του ενός κύκλου οφείλεται στην ύπαρξη καταχωρητών στην έξοδο του αθροιστή των 8 bit.

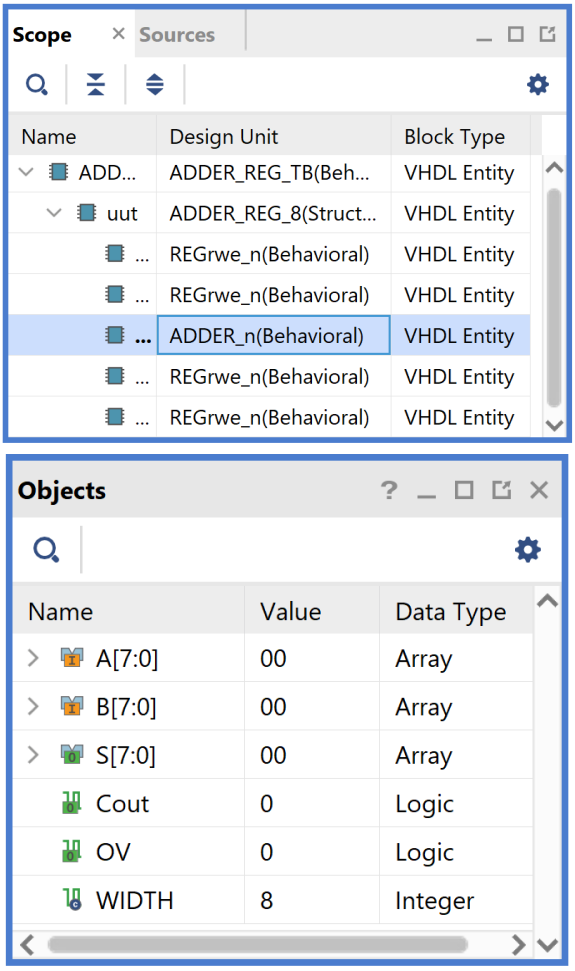
- Προσθέστε περισσότερα **εσωτερικά σήματα** στο διάγραμμα χρονισμού της προσομοίωσης. Για παράδειγμα τις εισόδους και τις εξόδους της συνδυαστικής λογικής (της οντότητας **Adder_n**).

  Στο παράθυρο *Scope* επιλέξτε την οντότητα **Adder_n**.

  Παρατηρείστε ότι στο παράθυρο *Objects* εμφανίζονται τα σήματα της οντότητας **ADDER_n**.

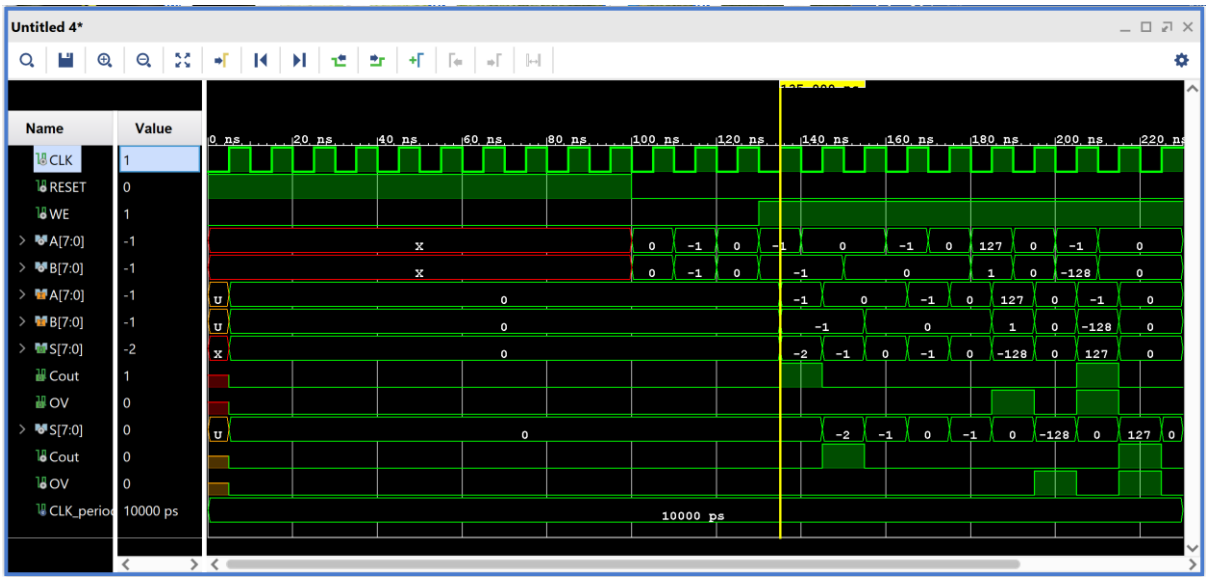
  Επιλέξτε το εσωτερικό σήμα A[7:0] και κάντε drag and drop στο παράθυρο με τα διαγράμματα χρονισμού, τοποθετώντας το κάτω από το υπάρχουν top–level σήμα B[7:0]. Επαναλάβετε την ίδια διαδικασία για τα εσωτερικά σήματα B[7:0], S[7:0] Cout και OV τοποθετώντας τα ακριβώς μετά το εσωτερικό σήμα A[7:0].

  Επαναλάβετε τη διαδικασία της προσομοίωσης από την αρχή με επιλογή του κουμπιού **Restart** και στη συνέχεια του κουμπιού **Run All**. Και τα



Εικόνα 72 - Παράθυρα scope και Objects

δύο κουμπιά βρίσκονται στην οριζόντια μπάρα στο πάνω μέρος.



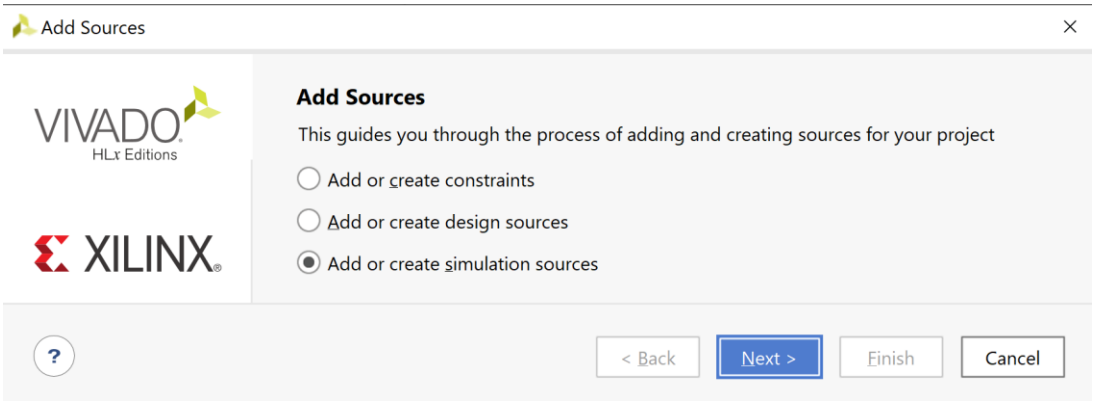
Εικόνα 73 – Διάγραμμα χρονισμού μετά την επανεκκίνηση της προσομοίωσης

- Κλείστε τον simulator επιλέγοντας το κουμπί × πάνω δεξιά στο παράθυρο του *SIMULATION*. Στο παράθυρο *Confirm Close* που εμφανίζεται πατήστε **OK**. Στο παράθυρο *Save Waveform Configuration* πατήστε **Save** για να αποθηκεύσετε το configuration του διαγράμματος χρονισμού στο οποίο έχετε καταλήξει, αλλιώς πατήστε **Discard**. Εάν πατήσετε **Save**, στο παράθυρο *Save Waveform* που εμφανίζεται διατηρείστε το όνομα **ADDER_REG_TB_behav.wcfg** και πατήστε **Save**. Στο παράθυρο *Waveform Configuration File* που εμφανίζεται πατήστε **Yes**.

Η διαδικασία που ήδη περιγράψαμε στα Βήματα 3–1 και 3–2 της «**Εισαγωγής του VHDL testbench και προσομοίωσης συμπεριφοράς**» εφαρμόζεται παρομοίως σε κάθε πιθανή υπομονάδα συνδυαστικής λογικής της διαδρομής δεδομένων και της μονάδας ελέγχου του επεξεργαστή.

1.8.3 Δημιουργία προγράμματος δοκιμών (testbench) τύπου VHD στο VIVADO IDE για μηχανές πεπερασμένων καταστάσεων (FSM)

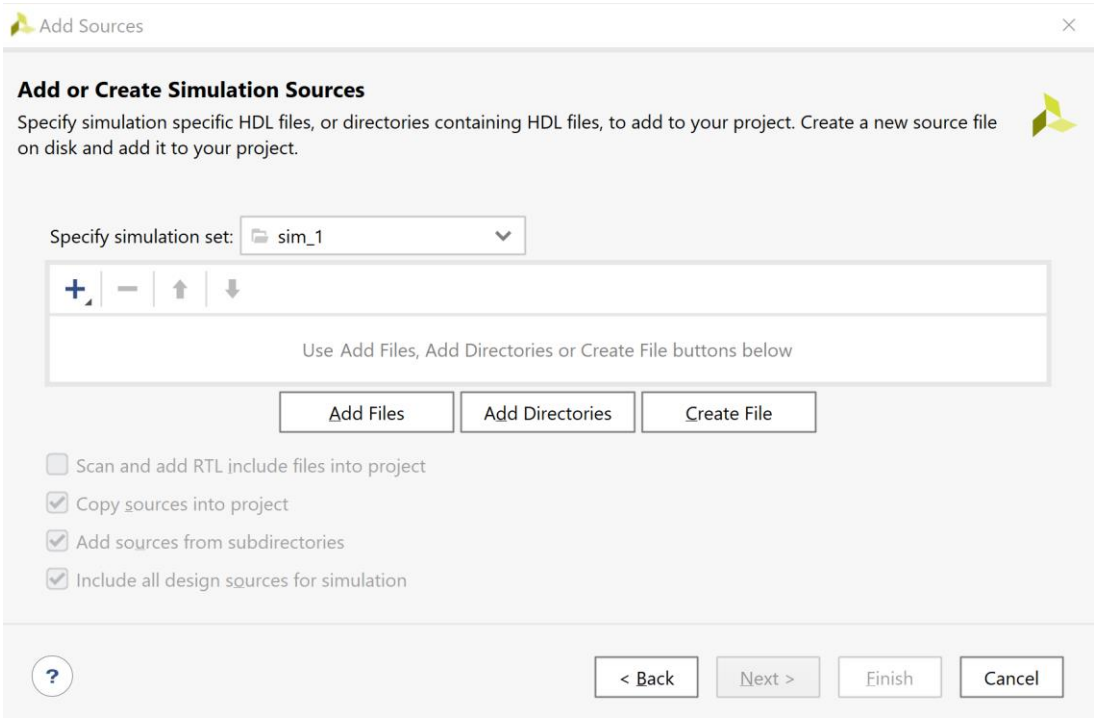
- Πατήστε το **+** στο παράθυρο *Sources* του PROJECT MANAGER για να ξεκινήσετε τον wizard δημιουργίας ενός νέου αρχείου (source).
- Στο παράθυρο διαλόγου *Add Sources* επιλέξτε το Add or create simulation sources, ώστε να δημιουργήσετε ένα νέο αρχείο προσομοίωσης (testbench) στη γλώσσα VHDL (simulation source file) τύπου VHD. Σε αυτό το αρχείο εισάγουμε τον κώδικα VHDL που περιγράφει τη λειτουργία του testbench. Πατήστε **Next**.



Εικόνα 74 – Πρώτο βήμα για τη δημιουργία αρχείου δοκιμών

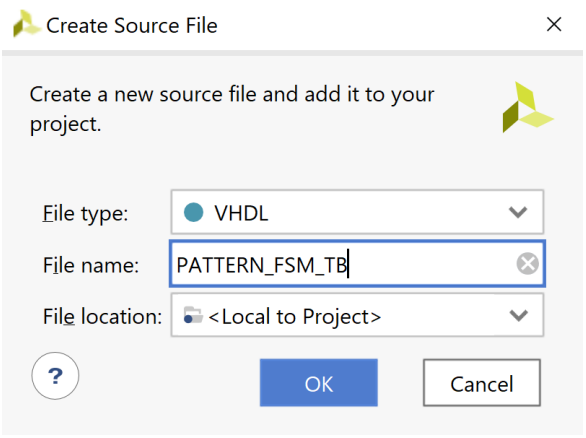
- Στο παράθυρο διαλόγου *Add or Create Simulation Sources* πατήστε την επιλογή **Create File** για τη δημιουργία του design simulation file που θα τοποθετηθεί στο ήδη καθορισμένο design source set *sources_1*.

  Εάν απαιτείται να καθοριστεί νέο simulation set (π.χ. sim_2), αρχικά δημιουργείστε το και στη συνέχεια ενεργοποιήστε το (επιλογή στο make active).



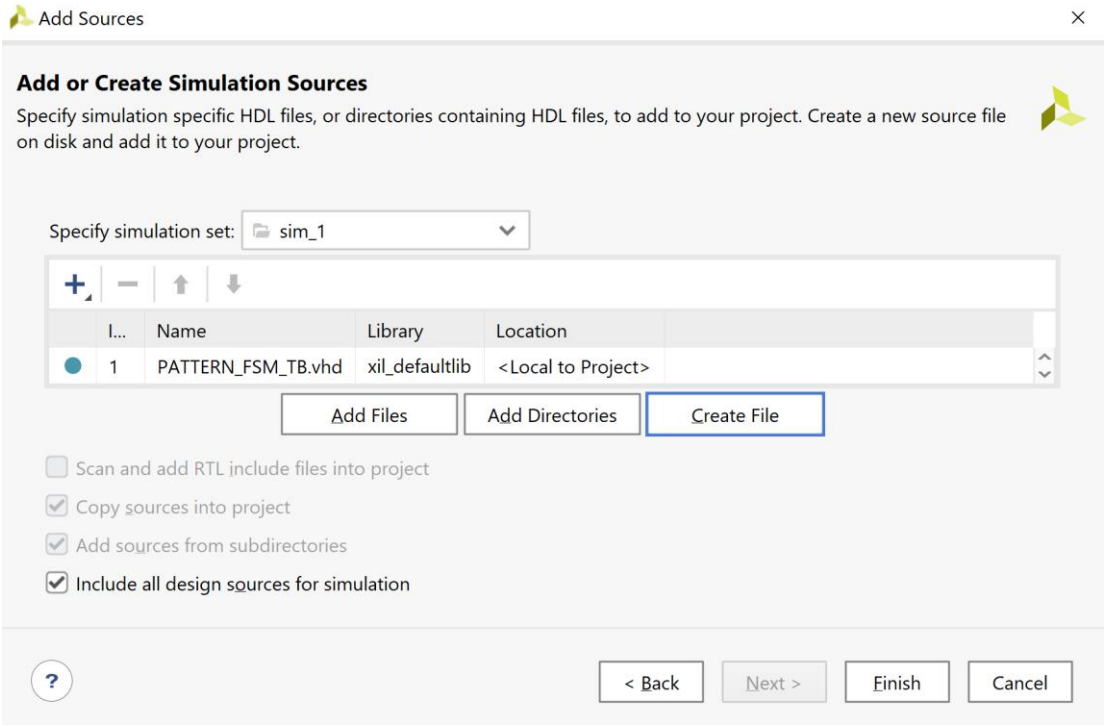
Εικόνα 75 – Επιλογή για προσθήκη νέου πηγαίου αρχείου προσομοίωσης

- Στο παράθυρο διαλόγου *Create Source File* δηλώστε το όνομα του testbench αρχείου (_TB) VHD (π.χ. **PATTERN_FSM_TB**), αφού σε αυτό το αρχείο θα περιγράψετε στη γλώσσα VHDL την οντότητα **PATTERN_FSM _TB** που απαιτείται για τις ανάγκες της προσομοίωσης της οντότητας **PATTERN_FSM**. Πατήστε **OK**.



Εικόνα 76 - Δημιουργία νέου αρχείου

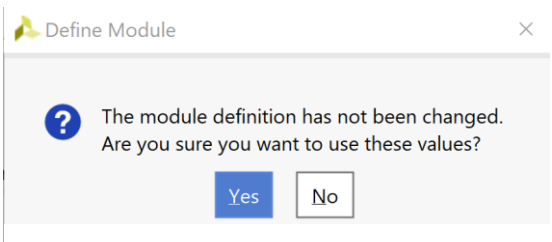
- Επιστρέφετε στο παράθυρο διαλόγου *Add or Create Simulation Sources*, όπου φαίνεται ότι έχει δημιουργηθεί το αρχείο **PATTERN_FSM_TB.vhd** και έχει συμπεριληφθεί στα αρχεία του project που ονομάζεται DSD_LAB_1. Διατηρείστε την επιλογή Include all design sources for simulation και πατήστε **Finish**.



Εικόνα 77 – Ολοκλήρωση διαδικασίας δημιουργίας πηγαίου αρχείου προσομοίωσης

- Εμφανίζεται το παράθυρο διαλόγου *Define Module*, όπου σας παρέχεται η δυνατότητα της δήλωσης του ονόματος της οντότητας (παραμένει **PATTERN_FSM_TB**), του ονόματος της αρχιτεκτονικής (επιλέξτε *behavioral*). Αυτή η οντότητα δεν έχει ports. Τέλος, πατήστε **OK**.

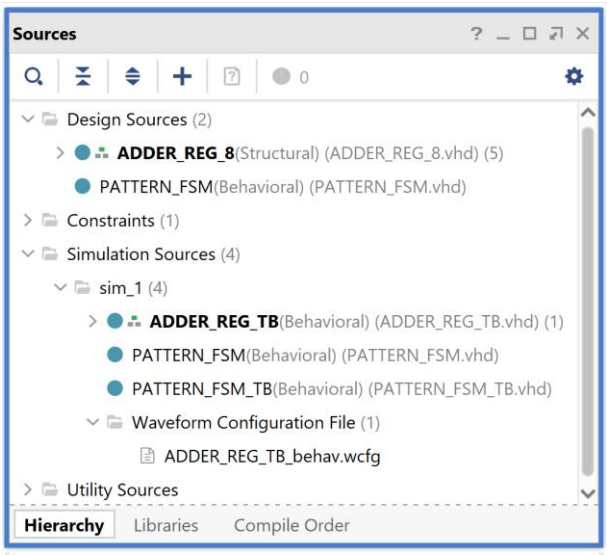
- Εμφανίζεται το παράθυρο προειδο–ποίησης *Define Module* που προειδο–ποιεί για τη μη δήλωση των I/O ports. Πατήστε **Yes**.



Εικόνα 78 - Παράκαμψη ορισμού ports

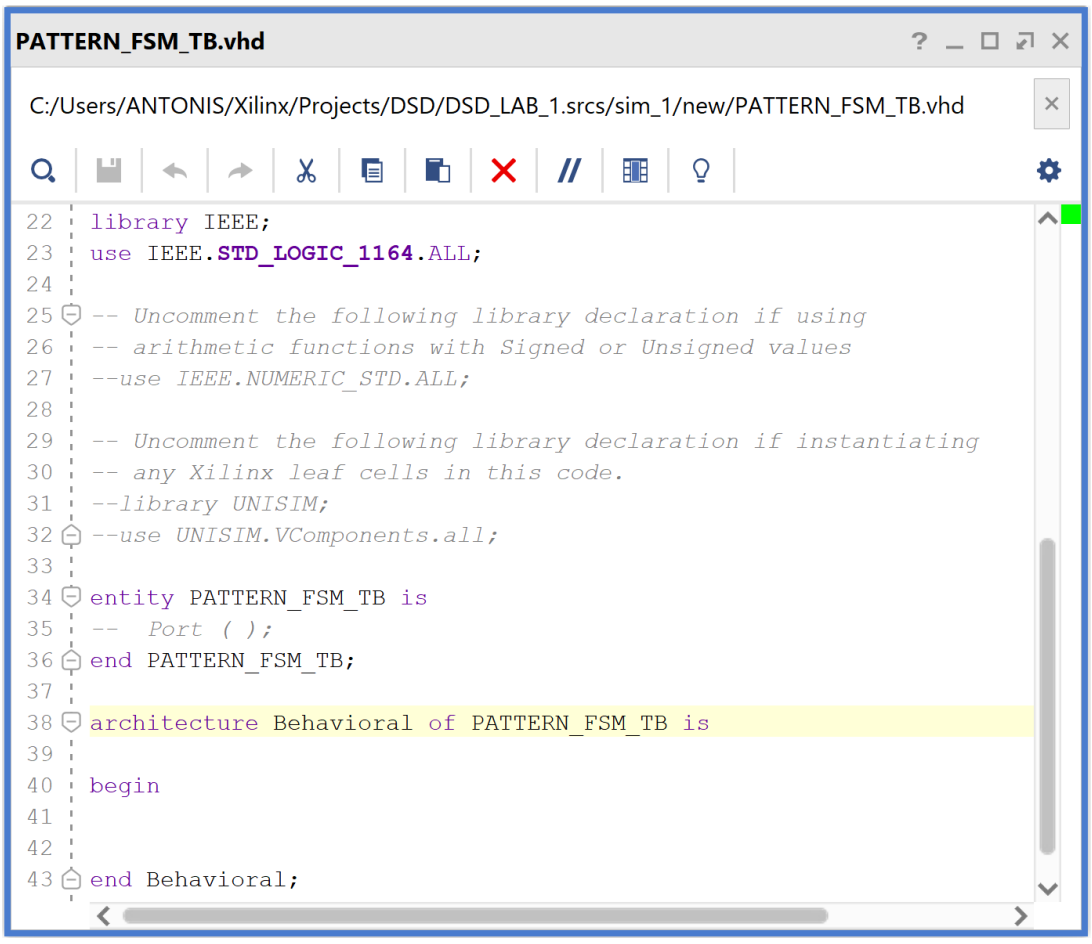
- Επιβεβαιώστε τη δημιουργία του προγράμματος δοκιμών (testbench) **PATTERN_FSM_TB.vhd** στο παράθυρο *Sources* του PROJECT MANAGER (επίσης φαίνονται τα ονόματα της οντότητας **PATTERN_FSM_TB** και της αρχιτεκτονικής της **Behavioral**). Το αρχείο αυτό είναι αποθηκευμένο μόνο στο simulation source set *sim_1* του project DSD_LAB1.

  Η οντότητα **ADDER_REG_8_TB** παραμένει ως η κορυφαία οντότητα της ιεραρχίας (**top**) των simulation resources επί του παρόντος. Επίσης, βλέπουμε όλα τα αρχεία που έχουμε δημιουρ–γήσει μέχρι τώρα.



Εικόνα 79 - Διαθέσιμα sources

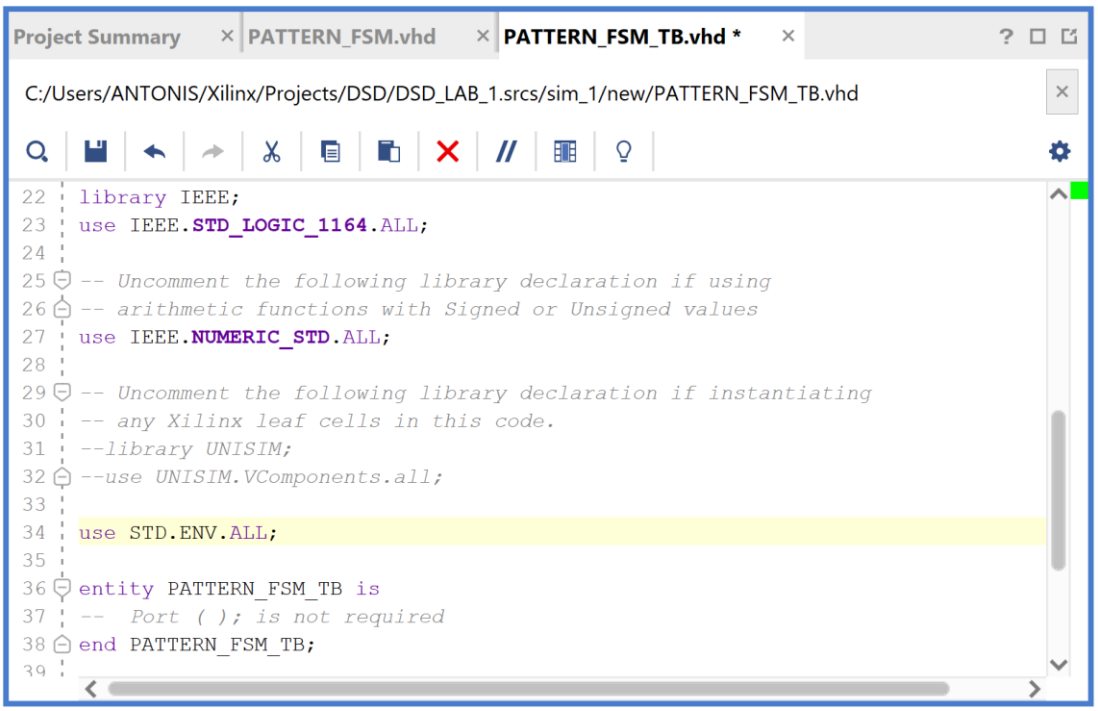
- Ανοίξτε το αρχείο **PATTERN_FSM_TB.vhd** με διπλό κλικ στο επιλεγμένο αρχείο. Λείπει ο ορισμός της αρχιτεκτονικής της οντότητας.



Εικόνα 80 – Αρχικά περιεχόμενα πηγαίου αρχείου PATTERN_FSM_TB.vhd

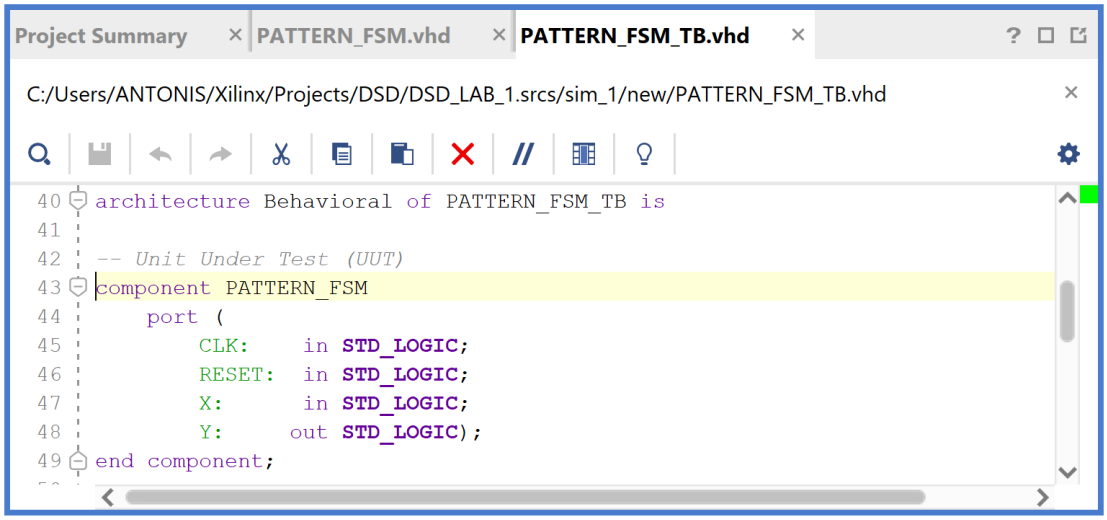
- Συμπληρώστε το πρόγραμμα δοκιμών (testbench) **PATTERN_FSM_TB.vhd** και στο τέλος πατήστε **Save File**, αφού πρώτα ακολουθήσετε τα παρακάτω βήματα:

  Βήμα 1: Ενεργοποιείστε τα πακέτα: **IEEE.NUMERIC_STD.ALL** και **STD.ENV.ALL**.



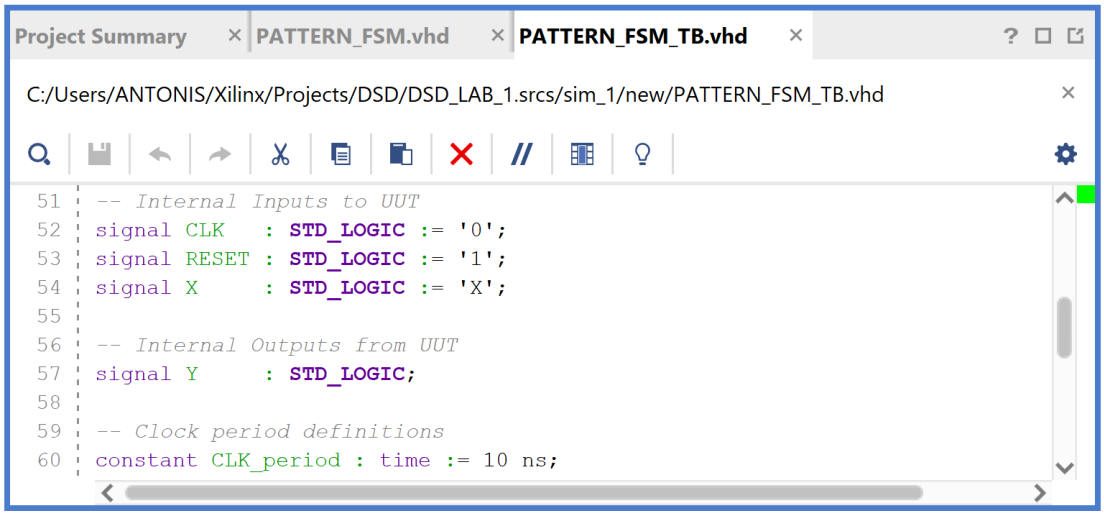
Εικόνα 81 – Ενεργοποίηση των πακέτων NUMERIC_STD και STD.ENV

Βήμα 2: Στις δηλώσεις της αρχιτεκτονικής αρχικά προσθέστε το **component PATTERN_FSM** (προκύπτει από το **entity PATTERN_FSM** με τις κατάλληλες τροποποιήσεις) που θα είναι και το *Unit Under test (UUT)* της οντότητας **PATTERN_FSM _TB** του συγκεκριμένου προγράμματος δοκιμών (testbench).

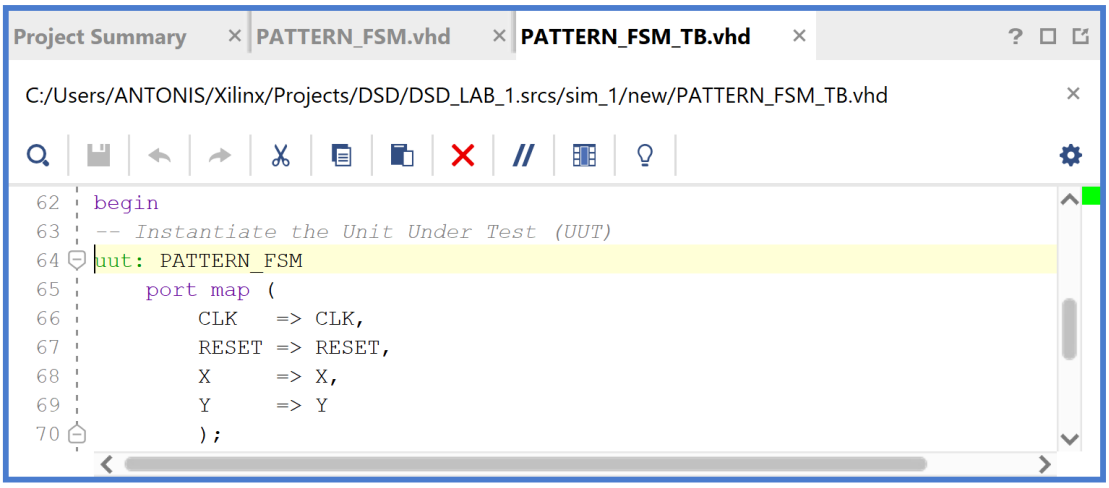


Εικόνα 82 – Προσθήκη του component PATTER_FSM στο αρχείο προσομοίωσης

Βήμα 3: Στη συνέχεια προσθέστε τα εσωτερικά σήματα που θα χρησιμοποιηθούν ως είσοδοι και έξοδοι του *UUT*. Τέλος, ορίστε την περίοδο του CLK (έχουμε επιλέξει αρχικά τα **10 ns** λαμβάνοντας υπόψη το σήμα του ρολογιού της κάρτας που είναι στα 100 MHz). Μπορείτε να προσαρμόσετε την περίοδο του CLK στις απαιτήσεις της δικής σας υλοποίησης της σχεδίασης, για τις ανάγκες τις προσομοίωσης.

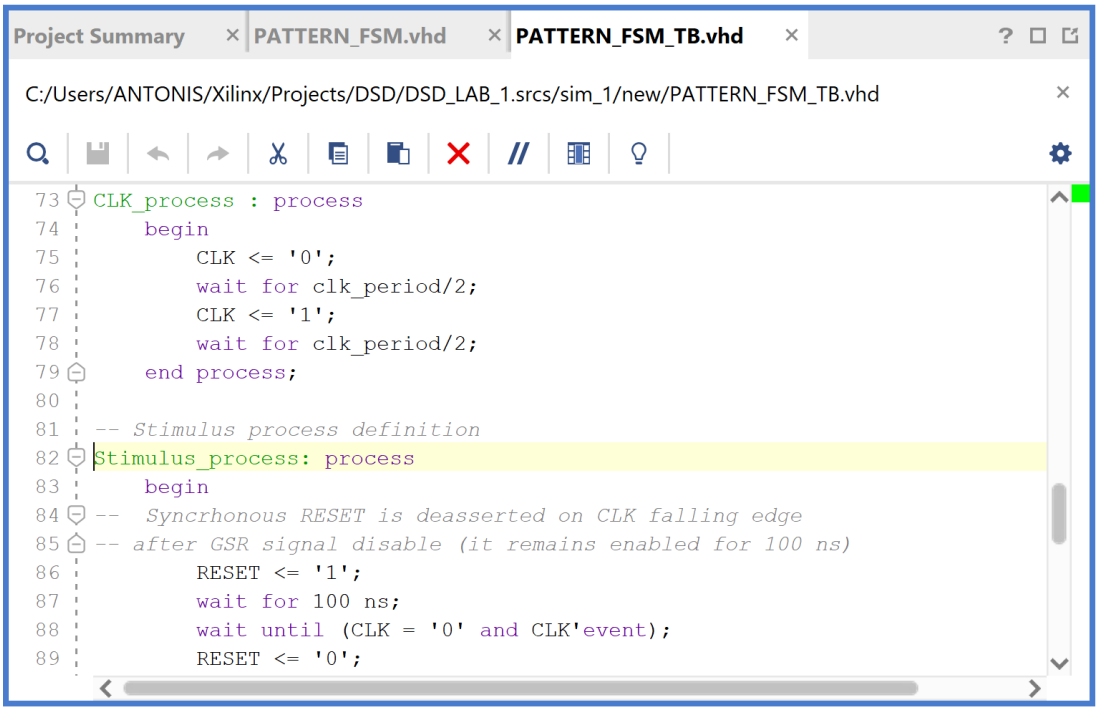


Βήμα 4: Στο σώμα της αρχιτεκτονικής μετά το begin, αρχικά, συνδέστε το *UUT* (**PATTERN_FSM**) με την οντότητα **PATTERN_FSM_TB** διατηρώντας τα ίδια ονόματα στα σήματα. Ιεραρχικά πλέον το *UUT* θα εμφανίζεται κάτω από τη συγκεκριμένη οντότητα.



Εικόνα 83 – Προσθήκη και σύνδεση του UUT με το υπόλοιπο αρχείο δοκιμών

Βήμα 5: Στη συνέχεια περιγράψτε τη συμπεριφορά του CLK (στο *CLK_process*) και τη συμπεριφορά του σήματος RESET (στην αρχή του *Stimulus_process*). Για να υπάρχει συμβατότητα σε όλα τα είδη των προσομοιώσεων έχουμε επιλέξει το σήμα RESET (με τη εντολή *wait for 100 ns*) να παραμένει ενεργό για τουλάχιστον 100 ns, όσο 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 (testbench) κατά τη διάρκεια της προσομοίωσης των synthesized design models και implemented design models. Επιπλέον, έχουμε επιλέξει (με την εντολή *wait until (CLK = '0' and CLK'event)*) η απενεργοποίηση του σήματος RESET να γίνεται στην κατερχόμενη ακμή του CLK, ώστε να μην δημιουργούνται παραβιάσεις στους χρόνους σταθεροποίησης (set–up) και διατήρησης (hold) των καταχωρητών, ανεξάρτητα από την περίοδο του CLK.

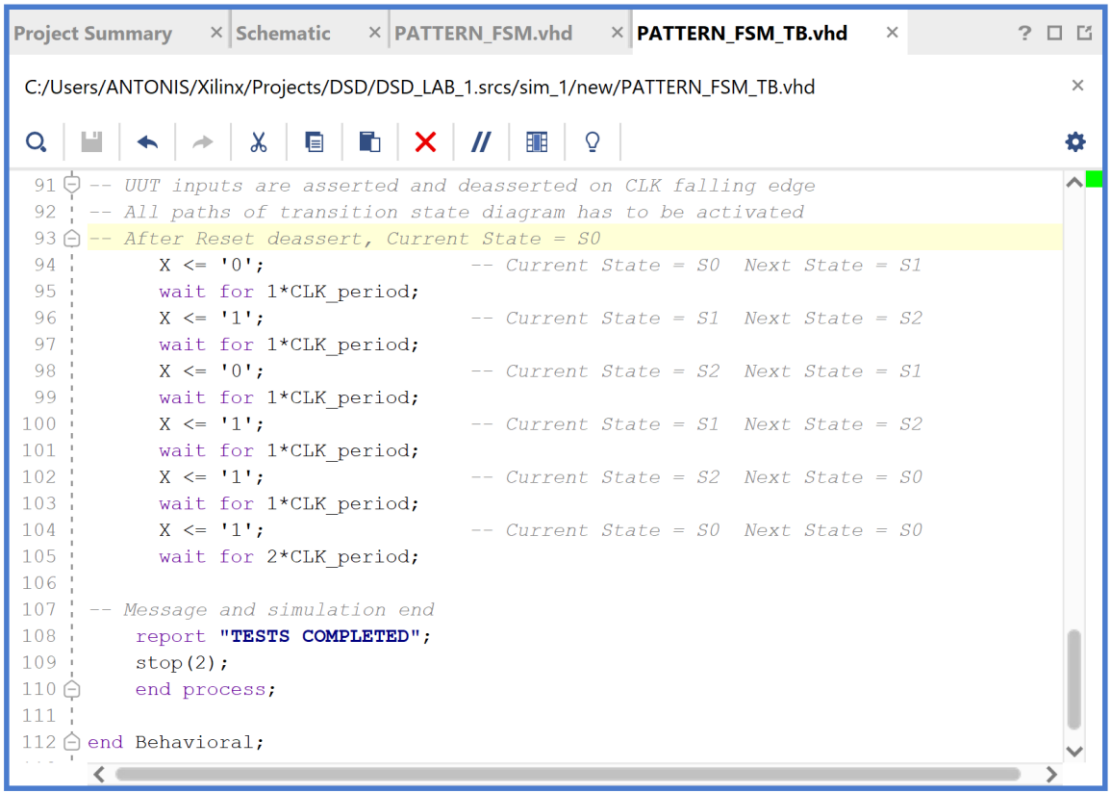


Εικόνα 84 – Περιγραφή συμπεριφοράς σημάτων RESET και CLK

Βήμα 6: Στη συνέχεια, στο ίδιο *Stimulus_process*, περιγράψτε τις εισόδους δοκιμής που θα λάβει το *UUT* κατά τη διάρκεια της προσομοίωσης. Οι αναθέσεις τιμών σε όλες τις εισόδους του *UUT* γίνονται στην κατερχόμενη ακμή του CLK, ώστε να μην δημιουργούνται παραβιάσεις στους χρόνους σταθεροποίησης (set–up) και διατήρησης (hold) των καταχωρητών, ανεξάρτητα από την περίοδο του CLK.

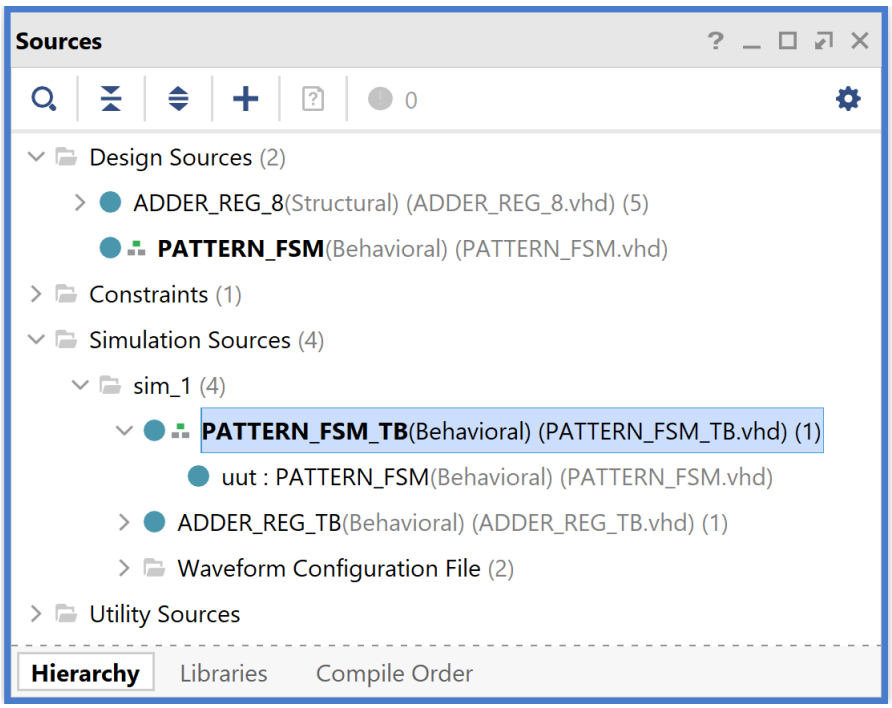
Στην περίπτωση των μηχανών πεπερασμένων καταστάσεων (FSM) για κάθε τρέχουσα κατάσταση ορίζουμε την επόμενη κατάσταση με βάση τις τιμές στην είσοδο, ξεκινώντας από την κατάσταση S0 του FSM που προκύπτει με την ενεργοποίηση του σήματος RESET. Φροντίζουμε να ενεργοποιήσουμε όλες τις διαδρομές του διαγράμματος μεταβολής κατάστασης, που εμφανίζονται κατά την κανονική λειτουργία του FSM.

Τέλος, δηλώστε μήνυμα ολοκλήρωσης της δοκιμής και διακοπής της προσομοίωσης με την εντολή *stop(2)*.



Εικόνα 85 – Περιγραφή εισόδων δοκιμής στο UUT

Βήμα 7: Επιβεβαιώστε την ιεραρχία της οντότητας του προγράμματος δοκιμών (testbench) **PATTERN_FSM_TB** σε σχέση με το *UUT* του (που είναι η οντότητα **PATTERN_FSM**) στο παράθυρο *Sources* του PROJECT MANAGER. Το αρχείο αυτό είναι αποθηκευμένο μόνο στο simulation source set *sim_1* του project DSD_LAB1. Ορίστε την οντότητα **PATTERN_FSM_TB** ως την κορυφαία οντότητα της ιεραρχίας (**top**) των simulation resources. Επίσης, ορίστε την οντότητα **PATTERN_FSM** ως την κορυφαία οντότητα της ιεραρχίας (**top**) των design resources.



Εικόνα 86 - Επιβεβαίωση ιεραρχίας οντότητας προγράμματος δοκιμών

1.8.4 Εκτέλεση προσομοίωσης συμπεριφοράς στο VIVADO IDE για μηχανές πεπερασμένων καταστάσεων (FSM)

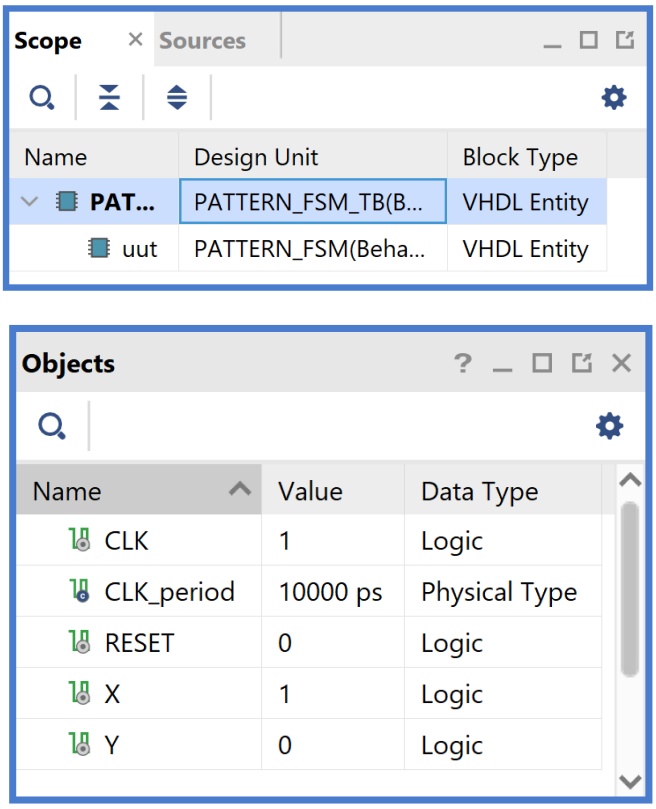
Η προσομοίωση συμπεριφοράς εκτελείται στην οντότητα **PATTERN_FSM_TB** του προγράμματος δοκιμών (testbench) που έχει ορισθεί ως η κορυφαία οντότητα της ιεραρχίας (**top**) των simulation resources. Κατά την προσομοίωση, η οντότητα **PATTERN_FSM_TB** καλεί το *UUT* της (που είναι η οντότητα **PATTERN_FSM**). Θα εμφανιστεί ένα νέο παράθυρο διαγραμμάτων χρονισμού χωρίς όνομα (untitled) και χωρίς σήματα.

- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Run Simulation**, ώστε να εμφανιστούν όλες οι πιθανές προσομοιώσεις που υποστηρίζει το Vivado IDE. Επιλέξτε **Run Behavioral Simulation**.
- Το πρόγραμμα δοκιμής (testbench) και όλες οι οντότητες του *UUT* θα γίνουν compiled και θα τρέξει το Vivado simulator (εφόσον βέβαια δεν υπάρχουν σφάλματα). Θα εμφανιστεί το παράθυρο *SIMULATION* που απαρτίζεται από 4 παράθυρα:

  Το παράθυρο *Scope*, όπου παρουσιάζεται η οντότητα **PATTERN_FSM_TB** και η οντό–τητα **PATTERN_FSM** (*UUT*).

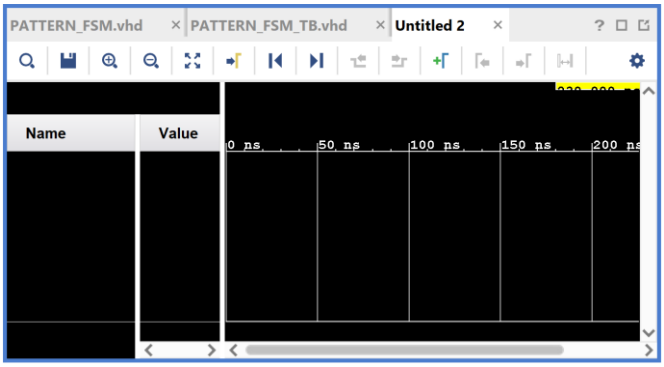
  Το παράθυρο Objects, όπου εμφανίζονται τα σήματα top–level, δηλαδή οι είσοδοι και οι έξοδοι της οντότητας **PATTERN_FSM**, που είναι η κορυφαία οντότητα της ιεραρ–χίας του *UUT*, καθώς και η περίοδος του CLK (*CLK_period*). Οι τιμές αντιστοιχούν στις τιμές που έχει σταματήσει η προσομοίωση της οντότητας **PATTERN_FSM_TB** (*stop (2)*).

  Το παράθυρο *Tcl Console* με όλες τις διαδικασίες που εκτελούνται στο πλαίσιο της προσομοίωσης. Το Tcl Console καθαρίζει με την επιλογή *Clear*.



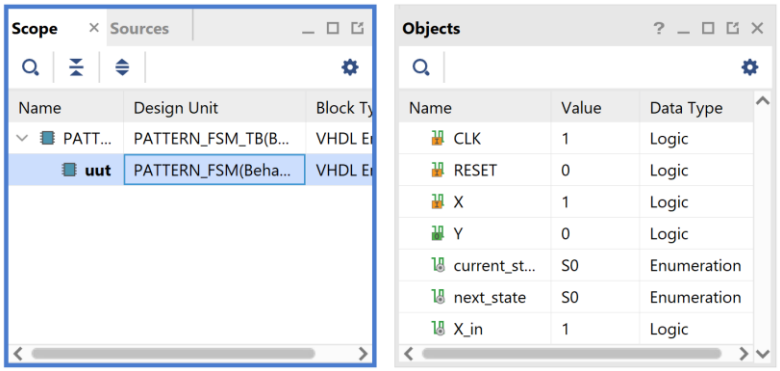
Εικόνα 87 - Παράθυρα Scope και Objects

Το παράθυρο με τα διαγράμματα χρονισμού. **Προσοχή**! Εάν έχετε αποθηκεύσει κάποιο προηγούμενο *waveform configuration*, όπως για παράδειγμα το **ADDER_REG_TB_behav.wcfg**, που αφορά στην προσομοίωση της οντότητας **ADDER_REG_TB** θα πρέπει να το αποσυνδέστε από την εκτέλεση της προσομοίωσης της οντότητας **PATTERN_FSM_TB** επιλέγοντας το **File**, στη συνέχεια το **Simulation Waveform** και τέλος το **New Configuration**. Θα εμφανιστεί ένα κενό παράθυρο διαγράμματος χρονισμού χωρίς όνομα (*untitled*).



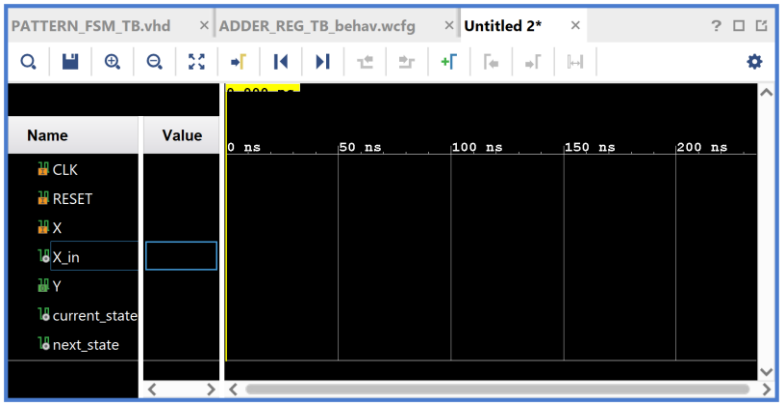
Εικόνα 88 – Κενό παράθυρο διαγράμματος χρονισμού

- Επιλέξτε στο παράθυρο Scope την οντότητα **PATTERN_FSM** (*UUT*). Στο παράθυρο *Objects* θα εμφανιστούν όλα τα σήματα της οντότητας **PATTERN_FSM** (είσοδοι, έξοδοι και εσωτερικά σήματα).



Εικόνα 89 – Σήματα οντότητας PATTERN_FSM

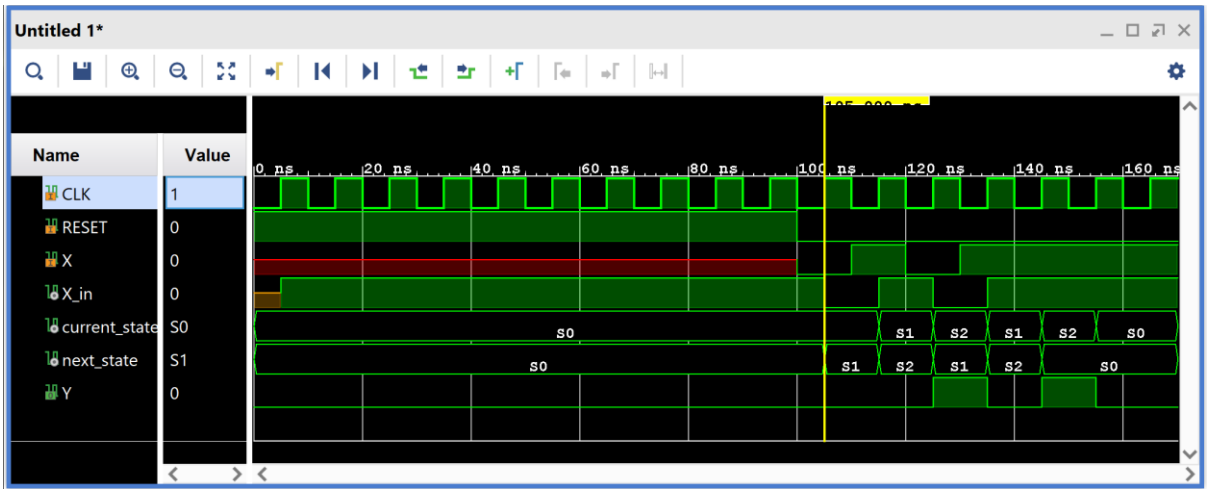
- Επιλέξτε όλα τα διαθέσιμα σήματα στο παράθυρο *Objects* και κάντε drag and drop στο παράθυρο με τα διαγράμματα χρονισμού. Συμπεριλάβετε το εσωτερικό σήμα **X_in** (που είναι συγχρονισμένο στην ανερχόμενη ακμή του CLK ως έξοδος του καταχωρητή εισόδων INREG) κάτω από την είσοδο X, καθώς και τα εσωτερικά σήματα **current_state** και **next_state** που βοηθάνε στην αποσφαλμάτωση.



Εικόνα 90 – Διάγραμμα χρονισμού με τα διαθέσιμα σήματα του UUT

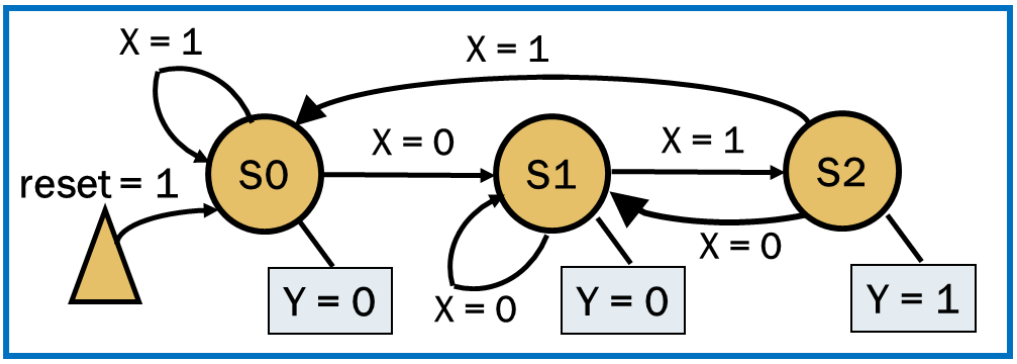
- Επαναλάβετε τη διαδικασία της προσομοίωσης από την αρχή με επιλογή του κουμπιού Restart και στη συνέχεια του κουμπιού Run All. Και τα δύο κουμπιά βρίσκονται στην οριζόντια μπάρα στο πάνω μέρος.

Εάν δεν είναι εμφανές το παράθυρο *Untitled*, επιλέξτε το *Untitled*. Βλέπετε ολόκληρο το διάγραμμα χρονισμού με κατάλληλο **zoom out** ή επιλέγοντας το **zoom fit** . Για να επαναφέρετε το floating παράθυρο πίσω, απλά επιλέξτε το κουμπί Dock Window.



Εικόνα 91 – Παράθυρο διαγράμματος χρονισμού μετά την επανεκκίνηση της προσομοίωσης

- Συγκρίνετε το διάγραμμα χρονισμού με το διάγραμμα μεταβολής κατάστασης.



Εικόνα 92 - Διάγραμμα μεταβολής κατάστασης

- Κλείστε τον simulator επιλέγοντας το κουμπί × πάνω δεξιά στο παράθυρο του *SIMULATION*. Στο παράθυρο *Confirm Close* που εμφανίζεται πατήστε **OK**. Στο παράθυρο *Save Waveform Configuration* πατήστε **Save** για να αποθηκεύσετε το configuration του διαγράμματος χρονισμού στο οποίο έχετε καταλήξει, αλλιώς πατήστε **Discard**. Εάν πατήσετε **Save**, στο παράθυρο *Save Waveform* που εμφανίζεται διατηρείστε το όνομα **PATTERN_FSM_TB_behav.wcfg** και πατήστε **Save**. Στο παράθυρο *Waveform Configuration File* που εμφανίζεται πατήστε **Yes**.

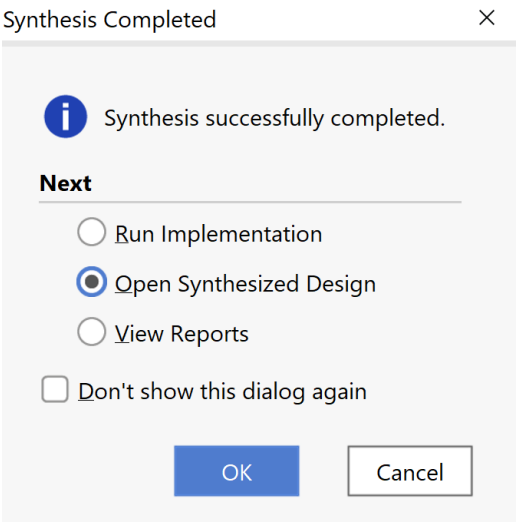
Η διαδικασία που ήδη περιγράψαμε στα Βήματα 3–3 και 3–4 της «**Εισαγωγής του VHDL testbench και προσομοίωσης συμπεριφοράς**» εφαρμόζεται παρομοίως σε κάθε πιθανή μηχανή πεπερασμένων καταστάσεων (FSM).

1.9 Βήμα 4: Σύνθεση του κώδικα VHDL και προσομοίωση (λογική, χρονική)

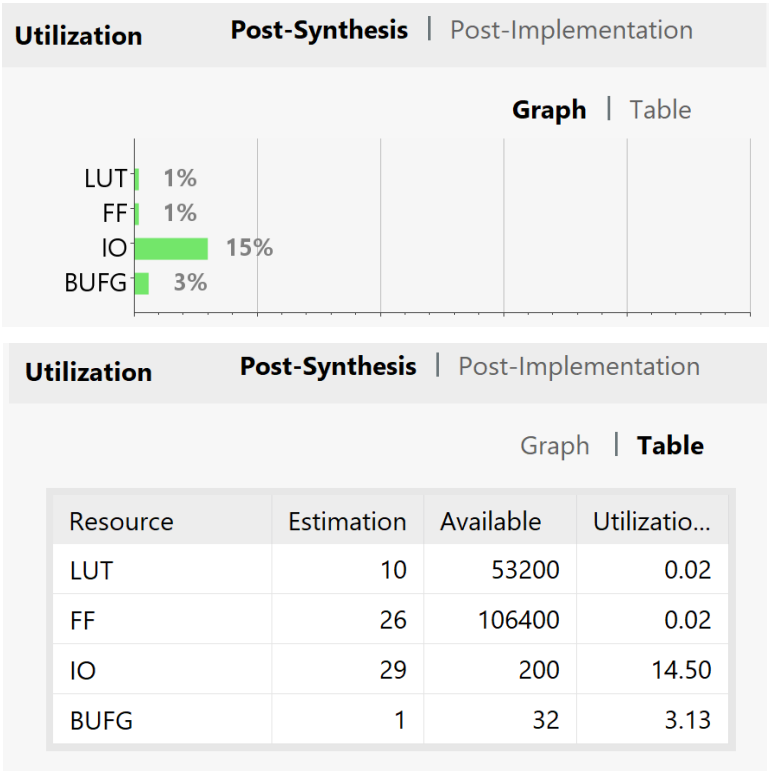
1.9.1 Εκτέλεση της διαδικασίας της σύνθεσης και ανάλυση των αποτελεσμάτων μετά τη σύνθεση στο VIVADO IDE για συνδυαστική λογική ανάμεσα σε καταχωρητές

Η σύνθεση εκτελείται στην οντότητα **ADDER_REG_8** που έχει ορισθεί ως η κορυφαία οντότητα της ιεραρχίας (**top**) των design resources, καθώς και σε όλα τα υπάρχοντα αρχεία της ιεραρχίας. Με τη σύνθεση το εργαλείο Vivado IDE παράγει το **synthesized design model** της οντότητας **ADDER_REG_8**.

- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Run Synthesis**. Πατήστε **OK** στα παράθυρα προειδο–ποίησης που εμφανίζονται.
- Στο παράθυρο *Synthesis Completed* υπάρχουν τρεις επιλογές. Επιλέξτε το **Open Synthesized Design** και πατήστε **OK** για να μελετήσετε το αποτέλεσμα της σύνθεσης πριν προχωρήσετε στο στάδιο της υλοποίησης (implementation). Πατήστε **Yes** για να κλείσετε το elaborated design, εάν εμφανιστεί το παράθυρο *Close Design*.
- Αρχικά, επιλέξτε το παράθυρο *Project Summary* και μελετήστε τα διάφορα υπο–παράθυρα. Εάν δεν το βλέπετε επιλέξτε το εικονίδιο Project Summary . Μελε–τήστε στο υποπαράθυρο Utilization τους πόρους που χρησιμοποιεί η οντότητα **ADDER_REG_8** σε μορφή *Graph* και σε μορφή *Table* (στο κάτω μέρος αριστερά).

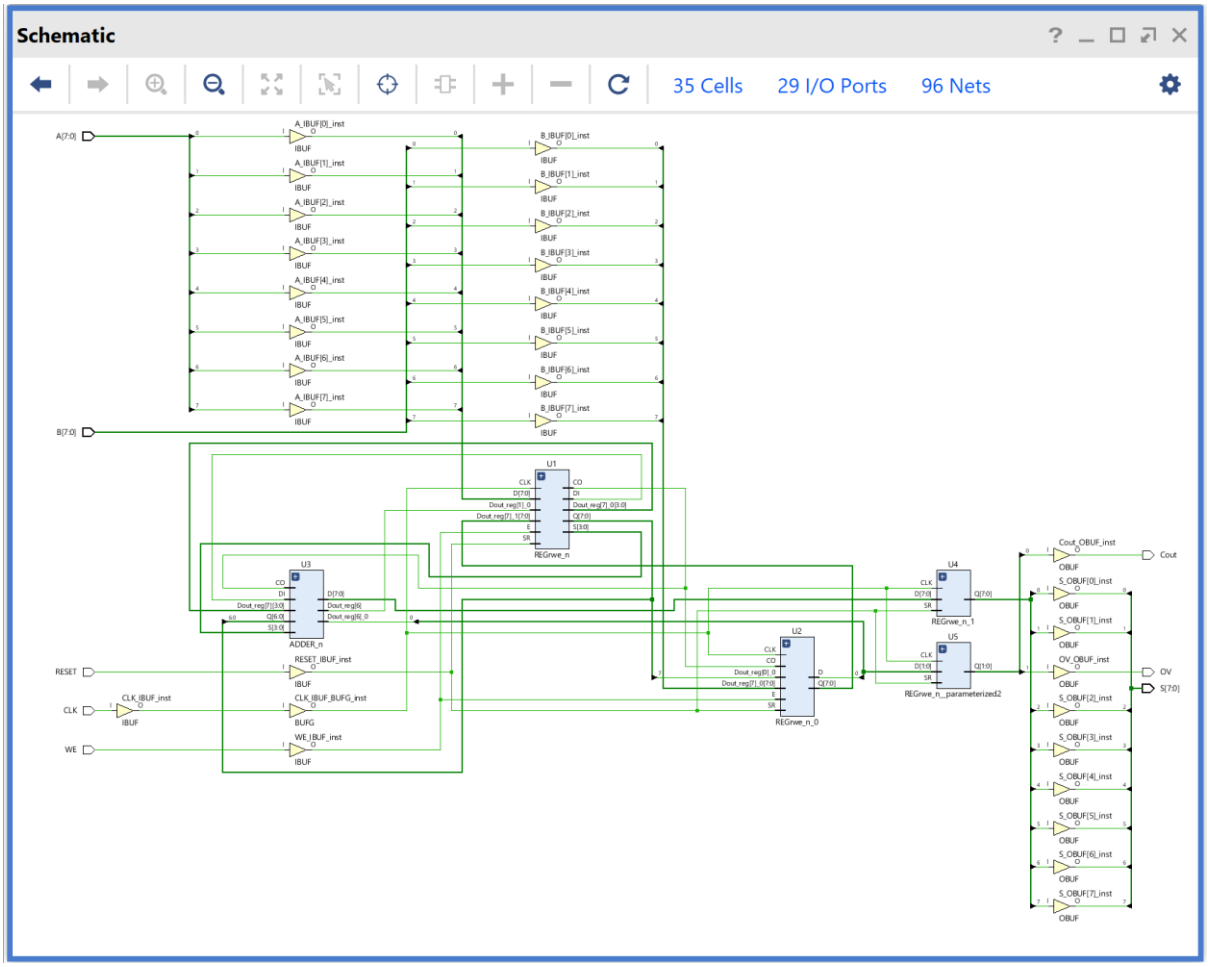


Εικόνα 93 - Επιλογές μετά τη σύνθεση



Εικόνα 94 – Υπο–παράθυρα του παραθύρου Project Summary

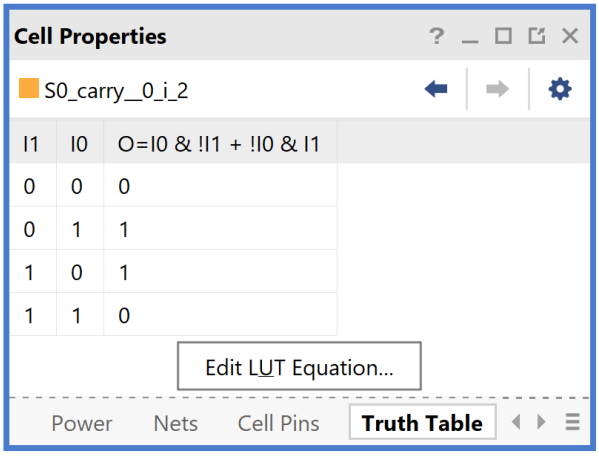
- Στη συνέχεια, στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, μέσα στο *Open Synthesized Design* επιλέξετε το **Schematic** για να δείτε το σχηματικό διάγραμμα του **synthesized design model**.



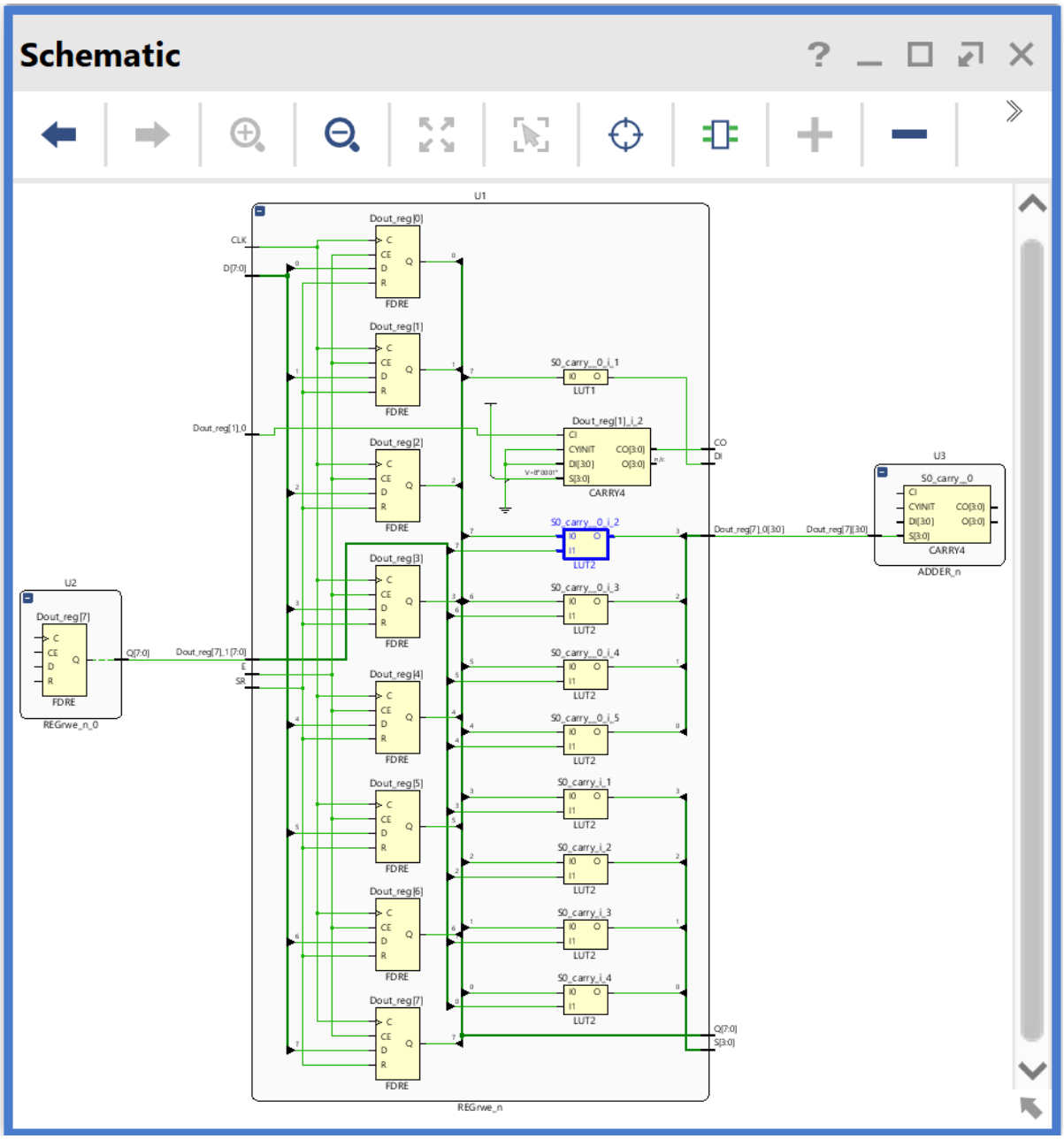
Εικόνα 95 – Σχηματικό διάγραμμα του synthesized design model

Παρατηρείστε ότι έχουν αυτόματα προστεθεί τα απαραίτητα IBUFs, OBUFs και BUFG primitives στο σχηματικό διάγραμμα, καθώς και ότι οι είσοδοι και οι έξοδοι από το FPGA είναι buffered. Επίσης, παρατηρείστε ότι έχει διατηρηθεί εν μέρει η ιεραρχία που έχει ορισθεί στην οντότητα **ADDER_REG_8**. Πατήστε το (+) σε όλες τις υπομονάδες U1–U5 και μελετήστε τις μία προς μία.

Υπομονάδα U1 (REGrwe_n): Συμπεριλαμβάνει τον καταχωρητή εισόδου A των 8 bit, 8 πύλες XOR (LUT2), έναν αντιστροφέα (LUT1) και μία μονάδα CARRY4. Επαληθεύστε τον πίνακα αλήθειας και την εξίσωση Boole ενός LUT, επιλέγοντας το συγκεκριμένο LUT2 (που αποκτά έντονο μπλε περίγραμμα) και ρυθμίζοντας την επιλογή *Truth Table* στο παράθυρο *Cell Properties*. Επίσης, φαίνονται οι είσοδοι και οι έξοδοι του συγκεκριμένου LUT2.



Εικόνα 96 - Παράδειγμα πίνακα αλήθειας LUT



Εικόνα 97 – Σχηματικό διάγραμμα υπομονάδας U1 (REGrwe_n)

Υπομονάδα U2 (REGrwe_n_0): Συμπεριλαμβάνει τον καταχωρητή εισόδου B των 8 bit, 1 πύλη XOR (LUT2) και μία μονάδα CARRY4.

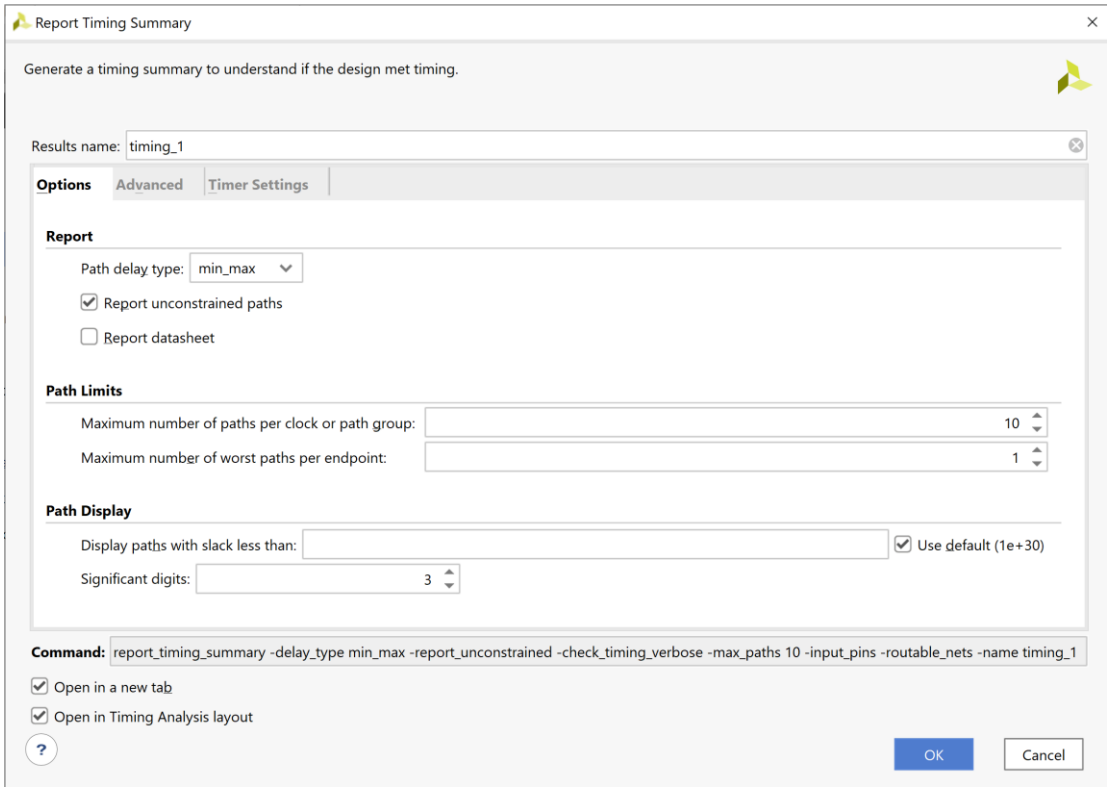
Υπομονάδα U3 (ADDER_n): Συμπεριλαμβάνει 1 πύλη XOR (LUT2) και 2 μονάδες CARRY4.

Υπομονάδα U4 (REGrwe_n_1): Συμπεριλαμβάνει τον καταχωρητή εξόδου S των 8 bit.

Υπομονάδα U5 (REGrwe_n_parameterized2): Συμπεριλαμβάνει τον καταχωρητή εξόδου των 2 bit για τις σημαίες Cout και OV.

Προσοχή! Οι υπομονάδες **U1**, **U2** και **U3** είναι διαφορετικές από πλευράς λογικής στο **synthesized design model**, που προκύπτει μετά τη σύνθεση, σε σχέση με το **elaborated design model**. Αντίθετα, οι υπομονάδες **U4** και **U5** παραμένουν ίδιες από πλευράς λογικής.

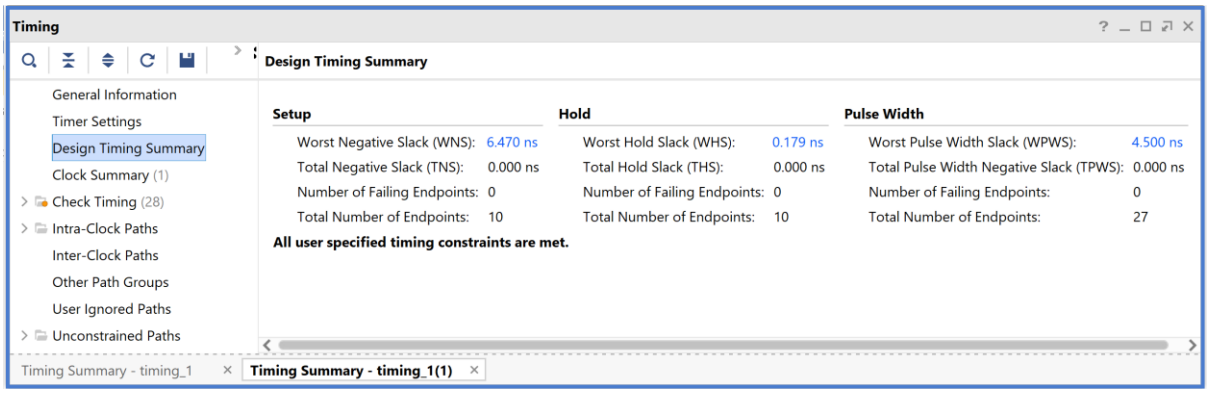
- Στη συνέχεια, στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, μέσα στο *Open Synthesized Design* επιλέξετε το **Report Timing Summary** για να δείτε την ανάλυση χρονισμού που κάνει το εργαλείο Vivado IDE στο **synthesized design model**. Αν και η ανάλυση χρονισμού σε αυτό το επίπεδο δεν είναι ακριβής, χρησιμοποιείται ευρέως στις πολύπλοκες σχεδιάσεις για εξοικονόμηση χρόνου κατά την ανάπτυξη του κώδικα VHDL.



Εικόνα 98 – Ανάλυση χρονισμού για το synthesized design model

Στην επιλογή *Path delay type* ορίζεται ο τύπος της ανάλυσης που θα εκτελεσθεί. Το *max delay analysis* αφορά στην εύρεση της κρίσιμης διαδρομής (με τη μεγαλύτερη καθυστέρηση διάδοσης) και συνεπώς στην εύρεση της μέγιστης συχνότητας λειτουργίας χωρίς την παραβίαση του **χρόνου σταθεροποίησης** (setup time). Το *min delay analysis* αφορά στην εύρεση της σύντομης διαδρομής (με τη μικρότερη καθυστέρηση διάδοσης) χωρίς την παραβίαση του **χρόνου διατήρησης** (hold time).

- Πατήστε **OK** για να παραχθεί το Timing_1 report.



Εικόνα 99 – Timing report για το synthesized design model

Στη στήλη **Setup** παρουσιάζονται τα αποτελέσματα του *max delay analysis*.

- Το *Worst Negative Slack* (**WNS**) είναι μια τιμή (με link) που αντιστοιχεί στο μικρότερο διαθέσιμο περιθώριο (slack) που προκύπτει από την ανάλυση όλων των κρίσιμων διαδρομών του max delay analysis. Μπορεί να είναι θετικό η αρνητικό. Ένα θετικό slack (6.470 ns) δηλώνει ότι η κρίσιμη διαδρομή ικανοποιεί την περίοδο του CLK. Ένα αρνητικό slack δηλώνει ότι παραβιάζεται ο χρόνος σταθεροποίησης (setup time).
- Το *Total Negative Slack* (**TNS**) είναι το άθροισμα όλων των αρνητικών WNS για κάθε timing path endpoint. Η τιμή είναι 0.000 ns όταν δεν υπάρχουν παραβιάσεις του χρόνου σταθεροποίησης. Το ψηφιακό κύκλωμα λειτουργεί στην επιλεγμένη συχνότητα λειτουργίας.
- Το *Number of Failing Endpoints* αφορά στον συνολικό αριθμό των endpoints που έχουν παραβιάσει το χρόνο σταθεροποίησης (αρνητικό WNS).
- Το *Total Number of Endpoints* αφορά στον συνολικό αριθμό των endpoints που έχουν αναλυθεί.

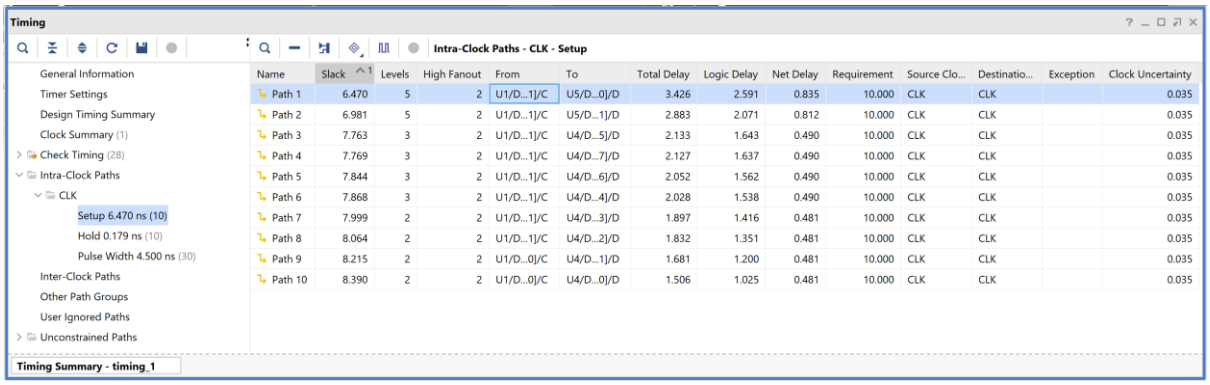
Στη στήλη **Hold** παρουσιάζονται τα αποτελέσματα του min delay analysis.

- Το *Worst Hold Slack* (**WHS**) είναι μια τιμή (με link) που αντιστοιχεί στο μικρότερο διαθέσιμο περιθώριο (slack) που προκύπτει από την ανάλυση όλων των σύντομων διαδρομών του min delay analysis. Μπορεί να είναι θετικό η αρνητικό. Ένα αρνητικό slack δηλώνει ότι παραβιάζεται ο χρόνος διατήρησης (hold time).
- Το *Total Hold Slack* (**THS**) είναι το άθροισμα όλων των αρνητικών WHS για κάθε timing path endpoint. Η τιμή είναι 0.000 ns όταν δεν υπάρχουν παραβιάσεις του χρόνου διατήρησης. Το ψηφιακό κύκλωμα λειτουργεί κανονικά.
- Το *Number of Failing Endpoints* αφορά στον συνολικό αριθμό των endpoints που έχουν παραβιάσει το χρόνο διατήρησης (αρνητικό WHS).
- Το *Total Number of Endpoints* αφορά στον συνολικό αριθμό των endpoints που έχουν αναλυθεί.

Στη στήλη **Pulse Width** παρουσιάζονται τα περιθώρια του σήματος CLK, όταν είναι HIGH ή LOW.

- Επιλέξτε το **WNS link** και δείτε τις 10 χειρότερες κρίσιμες διαδρομές (δηλαδή με το μικρότερο θετικό slack) που δεν παραβιάζουν το χρόνο σταθεροποίησης. Η κρίσιμη διαδρομή έχει καθυστέρηση διάδοσης 3.426 ns, εκ των οποίων τα 2.591 ns αφορούν στη λογική (logic), ενώ τα 0.835 ns αφορούν στη δικτύωση (net). Η

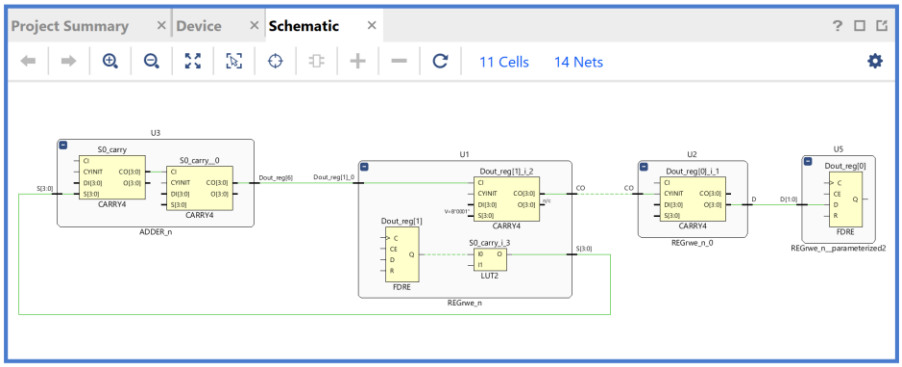
αβεβαιότητα του CLK εκτιμάται στα 0.035 ns. Τα επίπεδα λογικής (logic level) είναι 5.



Εικόνα 100 – Κρίσιμη διαδρομή για το synthesized design model

- Επιλέξτε με διπλό κλικ το **Path 1**, ώστε να εμφανιστεί το παράθυρο *Path 1 – timing_1*. Η κρίσιμη διαδρομή περνάει από 4 μονάδες CARRY4 και 1 LUT2 (πύλη XOR). Υπολογίστε το WNS slack:
- η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Source Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK του καταχωρητή της υπομονάδας U1) είναι 2.975 ns,
- η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Data Path* (από την είσοδο CLK μέχρι την έξοδο Q του καταχωρητή της υπομονάδας U1 και μέσω του LUT2 (U1) και των 4 μονάδων CARRY4 (U3–U1–U2) μέχρι την είσοδο D του καταχωρητή της υπομονάδας U5) είναι 3.426 ns,
- το **Arrival Time**, ως άθροισμα των ανωτέρων χρόνων, είναι 6.401 ns,
- το **Required Time**, ως η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Destination Clock Path* (από την πηγή του CLK τη χρονική στιγμή 10.000 ns μέχρι την είσοδο CLK του καταχωρητή της υπομονάδας U5) συν τον χρόνο σταθεροποίησης είναι 12.871 ns.

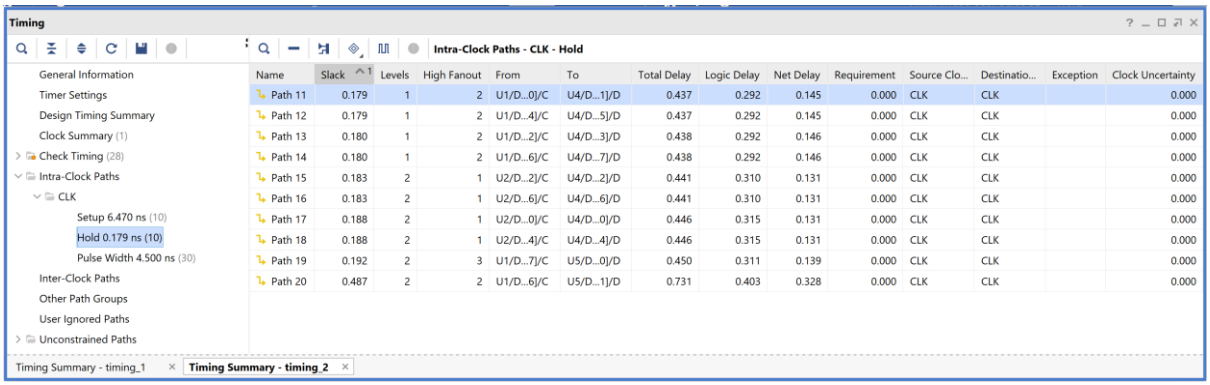
  WNS slack = Required Time – Arrival Time = 12.871 – 6.401 = 6.470 ns
- Επιλέξτε με δεξί κλικ στο **Path 1**, το **Schematic**. Μελετήστε το σχηματικό διάγραμμα της κρίσιμης διαδρομής της οντότητας **ADDER_REG_8**.



Εικόνα 101 – Σχηματικό διάγραμμα για την κρίσιμη διαδρομή του synthesized design model

- Επιλέξτε το **WHS link** και δείτε τις 10 χειρότερες σύντομες διαδρομές (δηλαδή με το μικρότερο θετικό slack) που δεν παραβιάζουν το χρόνο διατήρησης. Η σύντομη διαδρομή έχει καθυστέρηση μόλυνσης 0.437 ns, εκ των οποίων τα 0.292 ns αφορούν στη λογική (logic), ενώ τα 0.145 ns αφορούν στη δικτύωση (net). Η

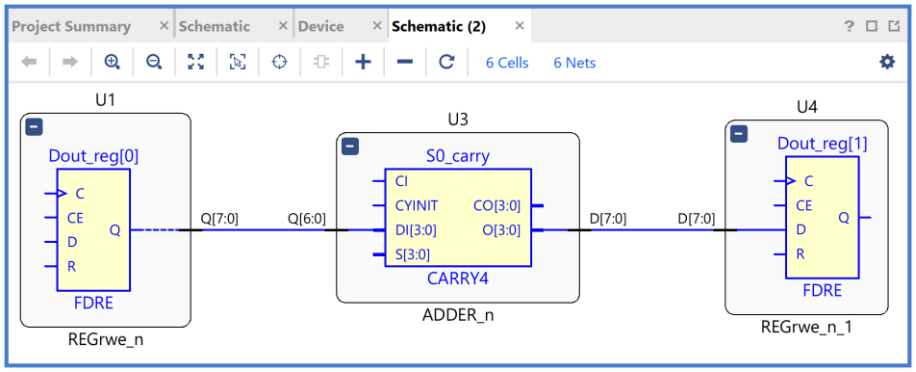
αβεβαιότητα του CLK εκτιμάται στα 0.000 ns. Τα επίπεδα λογικής (logic level) είναι 1.



Εικόνα 102 – Χειρότερες σύντομες διαδρομές (με θετικό slack) του synthesized design model

- Επιλέξτε με διπλό κλικ το **Path 11**, ώστε να εμφανιστεί το παράθυρο *Path 11 – timing_1*. Η σύντομη διαδρομή περνάει από 1 μονάδα CARRY4. Υπολογίστε το WHS slack:
- η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Source Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK του καταχωρητή της υπομονάδας U1) είναι 0.735 ns,
- η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Data Path* (από την είσοδο CLK μέχρι την έξοδο Q του καταχωρητή της υπομονάδας U1 και μέσω της 1 μονάδας CARRY4 (U3) μέχρι την είσοδο D του καταχωρητή της υπομονάδας U4) είναι 0.438 ns,
- το **Arrival Time**, ως άθροισμα των ανωτέρων χρόνων, είναι 1.173 ns,
- το **Required Time**, ως η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Destination Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK του καταχωρητή της υπομονάδας U4) συν τον χρόνο διατήρησης, είναι 0.993 ns.

  WHS slack = Arrival Time – Required Time = 1.173 – 0.993 = 0.180 ns
- Επιλέξτε με δεξί κλικ στο **Path 11**, το **Schematic**. Μελετήστε το σχηματικό διάγραμμα της σύντομης διαδρομής της οντότητας **ADDER_REG_8**.



Εικόνα 103 – Σχηματικό διάγραμμα χειρότερης σύντομης διαδρομής (με θετικό slack)

1.9.2 Εκτέλεση προσομοίωσης μετά τη σύνθεση (λογική και χρονική) στο VIVADO IDE για συνδυαστική λογική ανάμεσα σε καταχωρητές

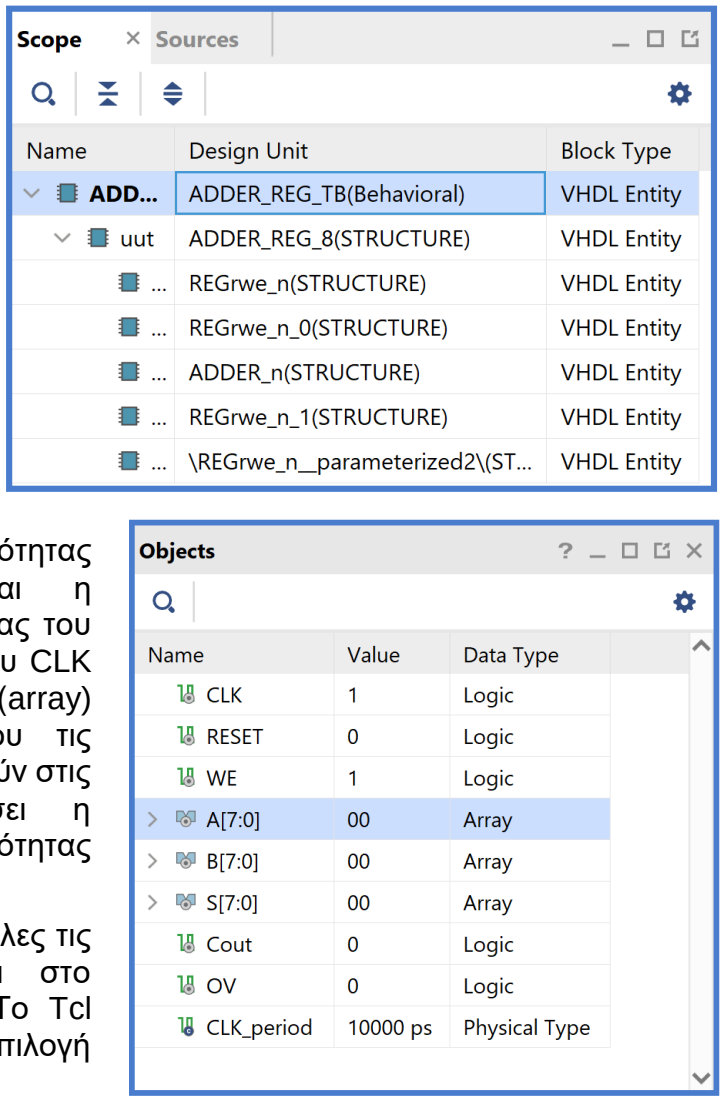
Η προσομοίωση μετά τη σύνθεση (λογική και χρονική) εκτελείται στην οντότητα **ADDER_REG_8_TB** του προγράμματος δοκιμών (testbench) που έχει ορισθεί ως η κορυφαία οντότητα της ιεραρχίας (**top**) των simulation resources. Κατά την προσομοίωση, η οντότητα **ADDER_REG_8_TB** καλεί το *UUT* της (που είναι το **synthesized design model** της οντότητας **ADDER_REG_8**).

- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Run Simulation**, ώστε να εμφανιστούν όλες οι πιθανές προσομοιώσεις που υποστηρίζει το Vivado IDE. Επιλέξτε **Run Post–Synthesis Functional Simulation**.
- Το πρόγραμμα δοκιμής (testbench) και όλες οι οντότητες του *UUT* θα γίνουν compiled και θα τρέξει το Vivado simulator (εφόσον βέβαια δεν υπάρχουν σφάλματα). Θα εμφανιστεί το παράθυρο *SIMULATION* που απαρτίζεται από 4 παράθυρα:

Το παράθυρο *Scope*, όπου παρουσιάζεται η οντότητα **ADDER_REG_8_TB**, το **synthesized design model** της οντότητας **ADDER_REG_8** (*UUT*) καθώς και οι υπόλοιπες νέες οντότητες του *UUT* που προκύπτουν μετά τη σύνθεση. Όλες οι οντότητες που απαρτίζουν πλέον το *UUT* είναι *structural*.

Το παράθυρο *Objects*, όπου εμφανίζονται τα σήματα top–level, δηλαδή οι είσοδοι και οι έξοδοι της οντότητας **ADDER_REG_8**, που είναι η κορυφαία οντότητα της ιεραρχίας του *UUT*, καθώς και η περίοδος του CLK (*CLK_period*). Οι αρτηρίες (array) αναλύονται στα σήματα που τις απαρτίζουν. Οι τιμές αντιστοιχούν στις τιμές που έχει σταματήσει η προσομοίωση της οντότητας **ADDER_REG_8_TB** (*stop (2)*).

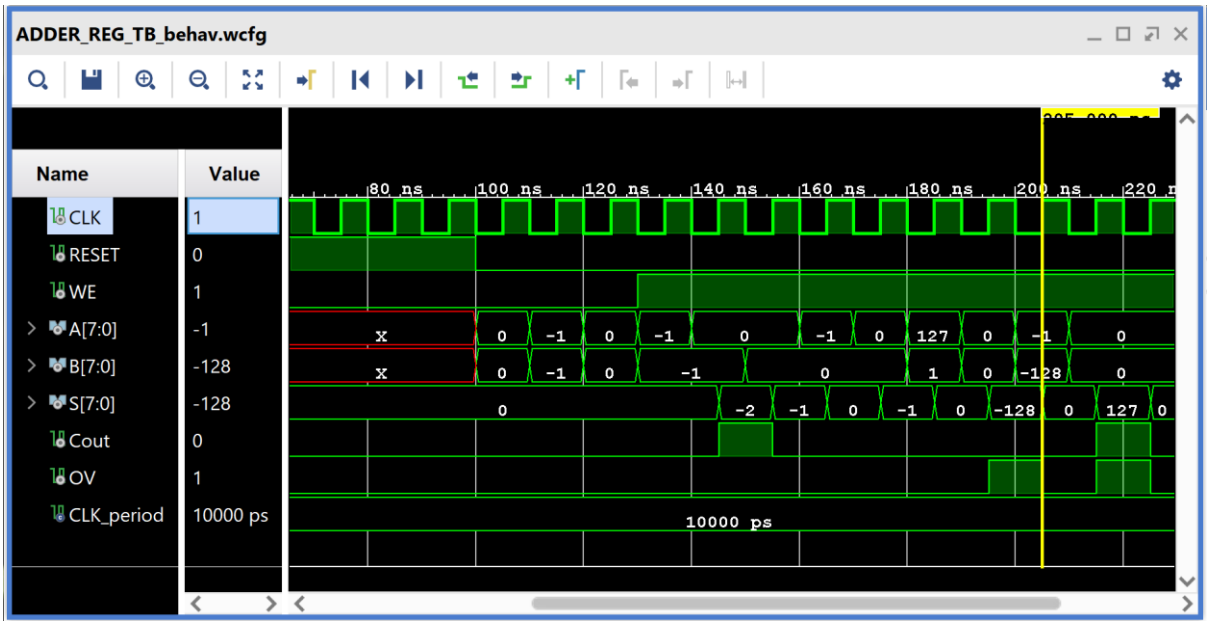
Το παράθυρο *Tcl Console* με όλες τις διαδικασίες που εκτελούνται στο πλαίσιο της προσομοίωσης. Το Tcl Console καθαρίζει με την επιλογή *Clear*.



Εικόνα 104 – Παράθυρα Scope και Objects

Επιλέξτε το παράθυρο *ADDER_REG_TB_behav.wcfg*, που δημιουργήσατε για την προσομοίωση συμπεριφοράς. Βλέπετε ολόκληρο το διάγραμμα χρονισμού της λογικής προσομοίωσης μετά τη σύνθεση με κατάλληλο **zoom out** ή επιλέγοντας το **zoom**

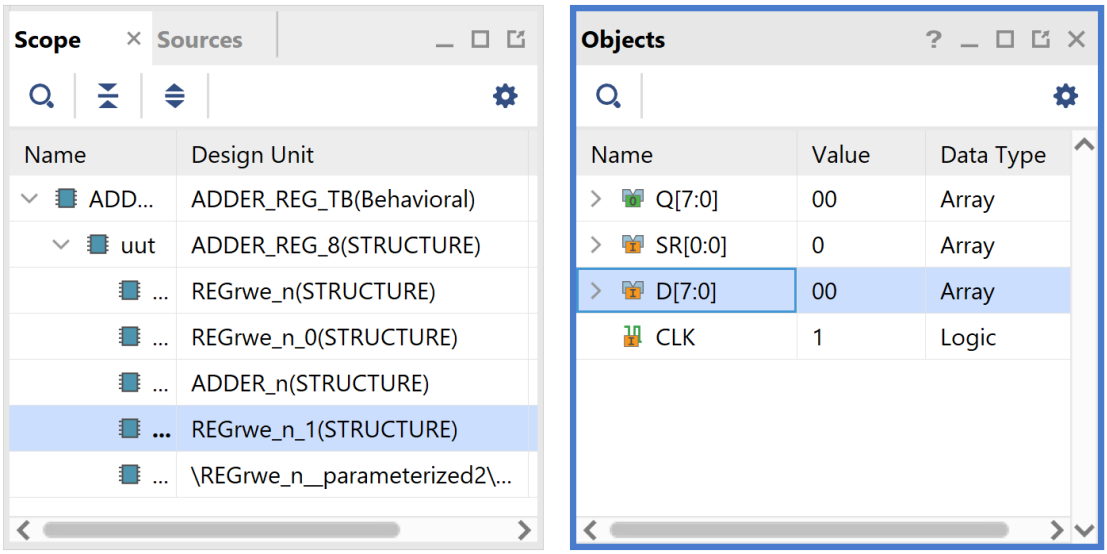
fit . Για να επαναφέρετε το floating παράθυρο πίσω, απλά επιλέξτε το κουμπί Dock Window. Μελετήστε μόνο top–level εισόδους/εξόδους.



Εικόνα 105 – Διάγραμμα χρονισμού λογικής προσομοίωσης μετά τη σύνθεση

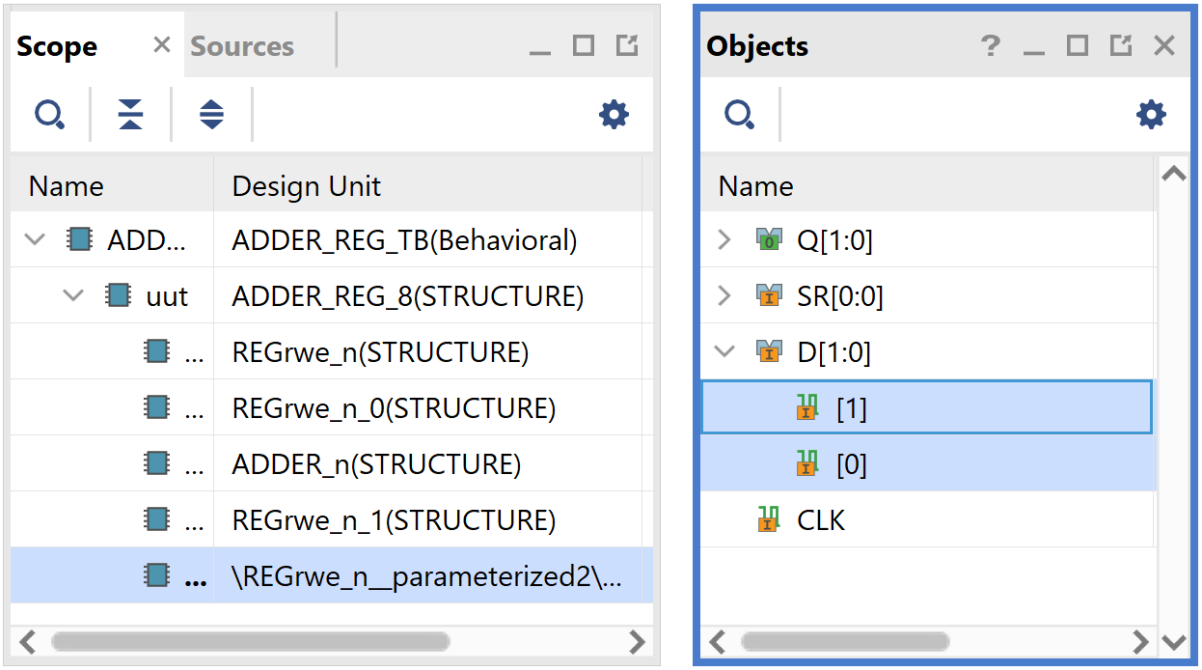
- Συγκρίνετε τα διαγράμματα χρονισμού της λογικής προσομοίωσης μετά τη σύνθεση και της προσομοίωσης συμπεριφοράς (υπάρχει ταύτιση στο επίπεδο των top–level εισόδων/εξόδων).
- Προσθέστε περισσότερα **εσωτερικά σήματα** στο διάγραμμα χρονισμού της προσομοίωσης μετά τη σύνθεση. Μετά από μελέτη του σχηματικού διαγράμματος του **synthesized design model** προκύπτει ότι οι είσοδοι **D[7:0]** και **D[1:0]** των υπομονάδων **U4** και **U5**, αντίστοιχα, αντιστοιχούν στις εξόδους **S[7:0]** και **Cout**, **OV** της οντότητας **Adder_n** (του **elaborated design model** πριν τη σύνθεση) και για αυτό το λόγο τις προσθέτουμε στο διάγραμμα χρονισμού της προσομοίωσης μετά τη σύνθεση.

  Στο παράθυρο *Scope* επιλέξτε την οντότητα **REGrwe_n_1**. Παρατηρείστε ότι στο παράθυρο Objects εμφανίζονται τα σήματα της οντότητας **REGrwe_n_1**. Επιλέξτε το εσωτερικό σήμα **D[7:0]** (που αντιστοιχεί στο S[7:0]) και κάντε drag and drop στο παράθυρο με τα διαγράμματα χρονισμού, τοποθετώντας το κάτω από το υπάρχουν top–level σήμα **B[7:0]**.



Εικόνα 106 - Παράθυρα Scope και Objects

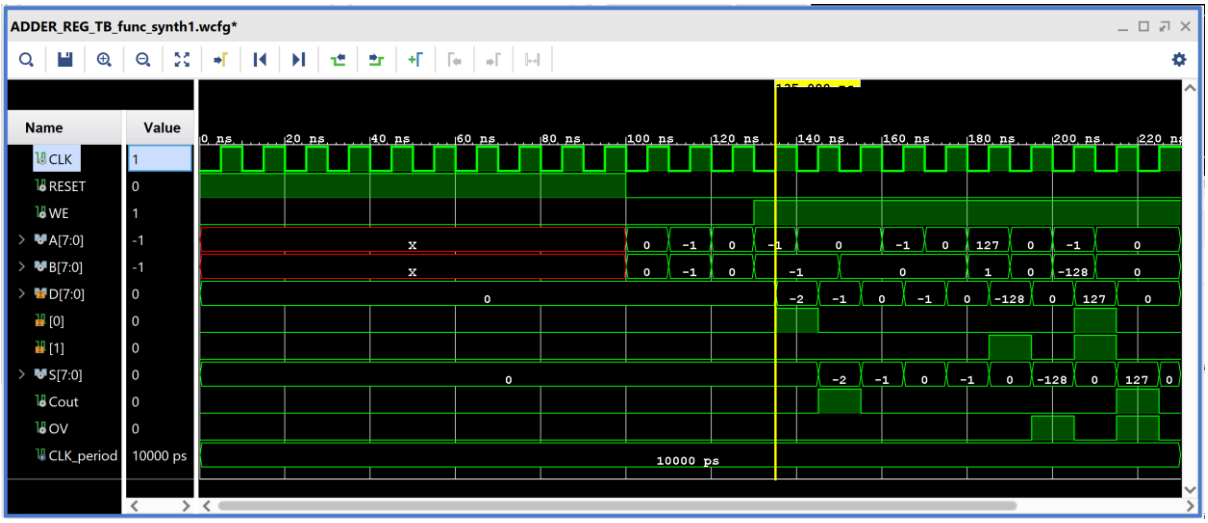
Στο παράθυρο *Scope* επιλέξτε την οντότητα **REGrwe_n_parameterized2**. Παρατηρείστε ότι στο παράθυρο Objects εμφανίζονται τα σήματα αυτής της οντότητας. Επιλέξτε τα εσωτερικά σήματα **D[0]** (που αντιστοιχεί στο Cout) και **D[1]** (που αντιστοιχεί στο OV) και κάντε drag and drop στο παράθυρο με τα διαγράμματα χρονισμού, τοποθετώντας το κάτω από το σήμα **D[7:0]**.



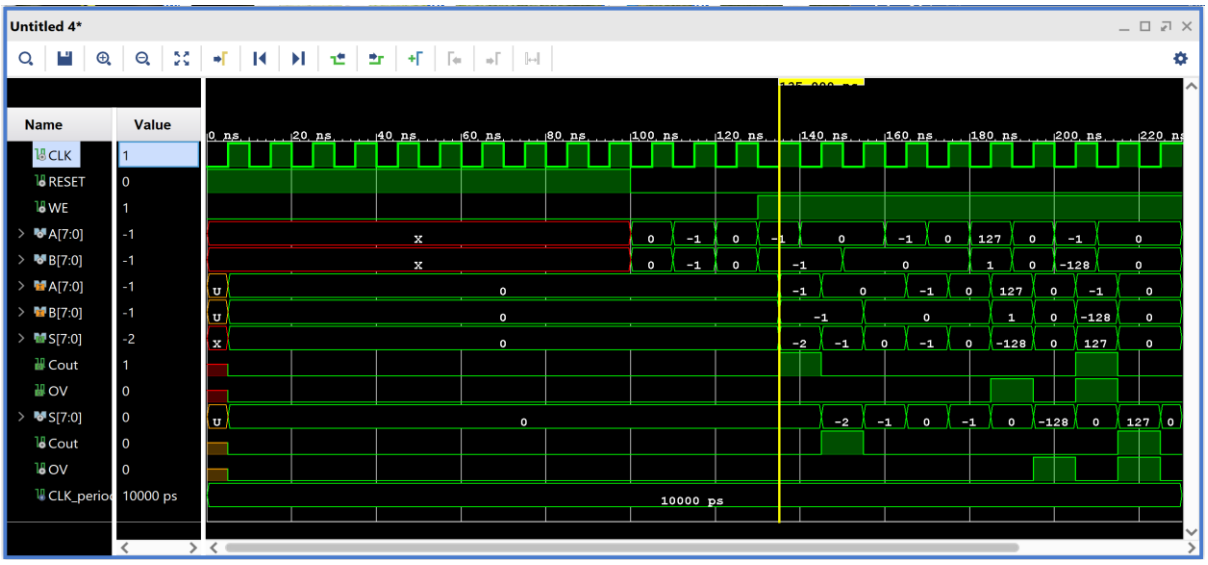
Εικόνα 107 - Παράθυρα Scope και Objects

Επαναλάβετε τη διαδικασία της προσομοίωσης από την αρχή με επιλογή του κουμπιού **Restart** και στη συνέχεια του κουμπιού **Run All**. Και τα δύο κουμπιά βρίσκονται στην οριζόντια μπάρα στο πάνω μέρος.

Το εσωτερικό σήμα **D[7:0]** του διαγράμματος χρονισμού της προσομοίωσης μετά τη σύνθεση αντιστοιχεί στο εσωτερικό σήμα **S[7:0]** του διαγράμματος χρονισμού της προσομοίωσης συμπεριφοράς. Ομοίως, το εσωτερικό σήμα **[0]** αντιστοιχεί στο **Cout** και το εσωτερικό σήμα **[1]** αντιστοιχεί στο **OV**.



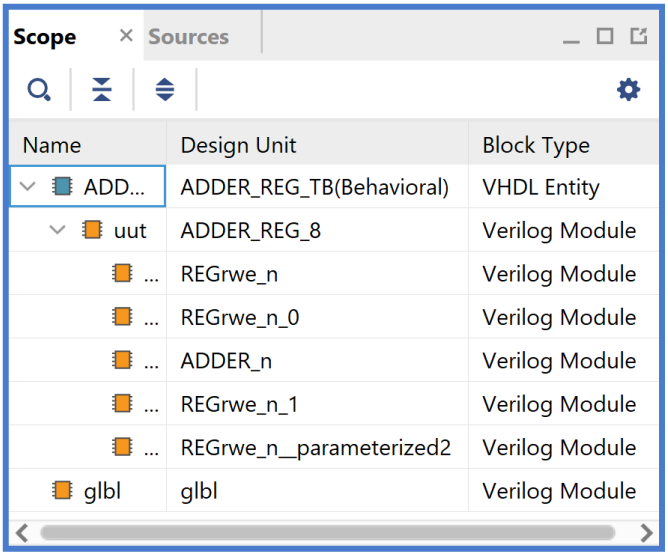
Εικόνα 108 – Διάγραμμα χρονισμού λογικής προσομοίωσης με επιπλέον εσωτερικά σήματα



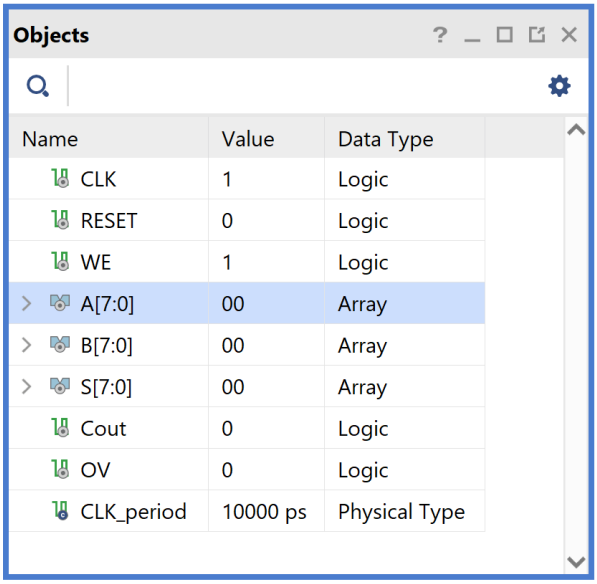
Εικόνα 109 – Διάγραμμα χρονισμού προσομοίωσης συμπεριφοράς με επιπλέον εσωτερικά σήματα

- Συγκρίνετε τη λογική προσομοίωση μετά τη σύνθεση με την προσομοίωση συμπεριφοράς (υπάρχει ταύτιση στο επίπεδο εισόδων/εξόδων και των επιλεγμένων εσωτερικών σημάτων).
- Επιλέξτε αρχικά **File** στη συνέχεια **Simulation Waveform** και τέλος **Save Configuration as ..** για να αποθηκεύσετε το configuration του διαγράμματος χρονισμού στο οποίο έχετε καταλήξει με νέο όνομα (π.χ. **ADDER_REG_TB_func_synth1.wcfg**). Στο παράθυρο *Waveform Configuration File* που εμφανίζεται πατήστε **Yes**.
- Κλείστε τον simulator επιλέγοντας το κουμπί × πάνω δεξιά στο παράθυρο του *SIMULATION*. Στο παράθυρο *Confirm Close* που εμφανίζεται πατήστε **OK**.
- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Run Simulation**, ώστε να εμφανιστούν όλες οι πιθανές προσομοιώσεις που υποστηρίζει το Vivado IDE. Επιλέξτε **Run Post–Synthesis Timing Simulation**.
- Το πρόγραμμα δοκιμής (testbench) και όλες οι οντότητες του *UUT* θα γίνουν compiled και θα τρέξει το Vivado simulator (εφόσον βέβαια δεν υπάρχουν σφάλματα). Θα εμφανιστεί το παράθυρο *SIMULATION* που απαρτίζεται από 4 παράθυρα:

Το παράθυρο *Scope*, όπου παρουσιάζεται η οντότητα **ADDER_REG_8_TB**, το **synthesized design model** της οντότητας **ADDER_REG_8** (*UUT*) καθώς και οι υπόλοιπες νέες οντότητες του *UUT* που προκύπτουν μετά τη σύνθεση για χρονική προσομοίωση. Όλες οι οντότητες που απαρτίζουν πλέον το *UUT* είναι *Verilog Module!*



Το παράθυρο *Objects*, όπου εμφανίζονται τα σήματα top–level, δηλαδή οι είσοδοι και οι έξοδοι της οντότητας **ADDER_REG_8**, που είναι η κορυφαία οντότητα της ιεραρχίας του *UUT*, καθώς και η περίοδος του CLK (*CLK_period*). Οι αρτηρίες (array) αναλύονται στα σήματα που τις απαρτίζουν. Οι τιμές αντιστοιχούν στις τιμές που έχει σταματήσει η προσομοίωση της οντότητας **ADDER_REG_8_TB** (*stop (2)*).

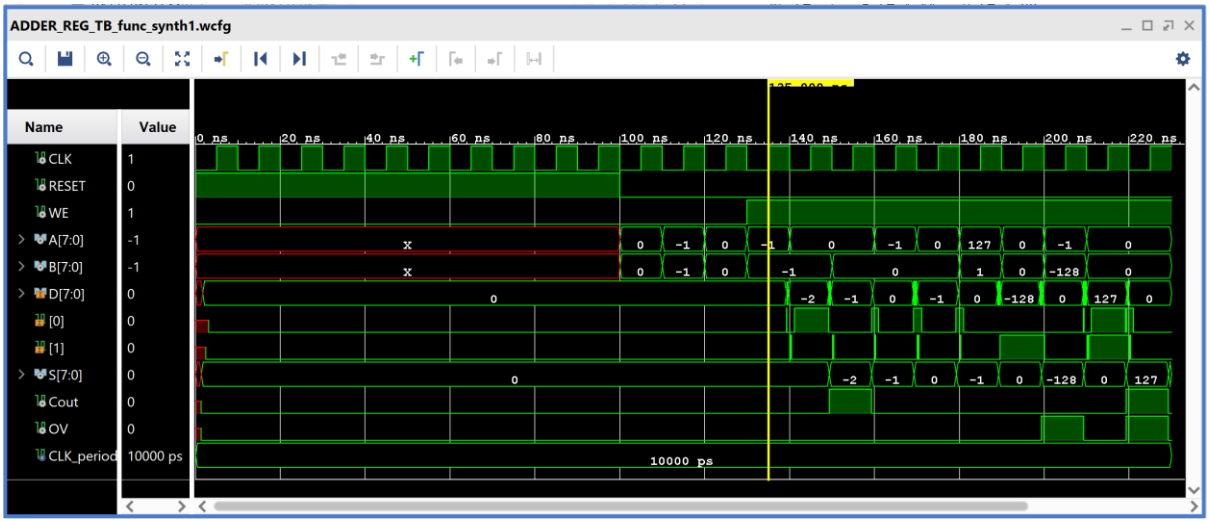


Εικόνα 110 - Παράθυρα Scope και Objects

Το παράθυρο *Tcl Console* με όλες τις διαδικασίες που εκτελούνται στο πλαίσιο της προσομοίωσης. Το Tcl Console καθαρίζει με την επιλογή *Clear*.

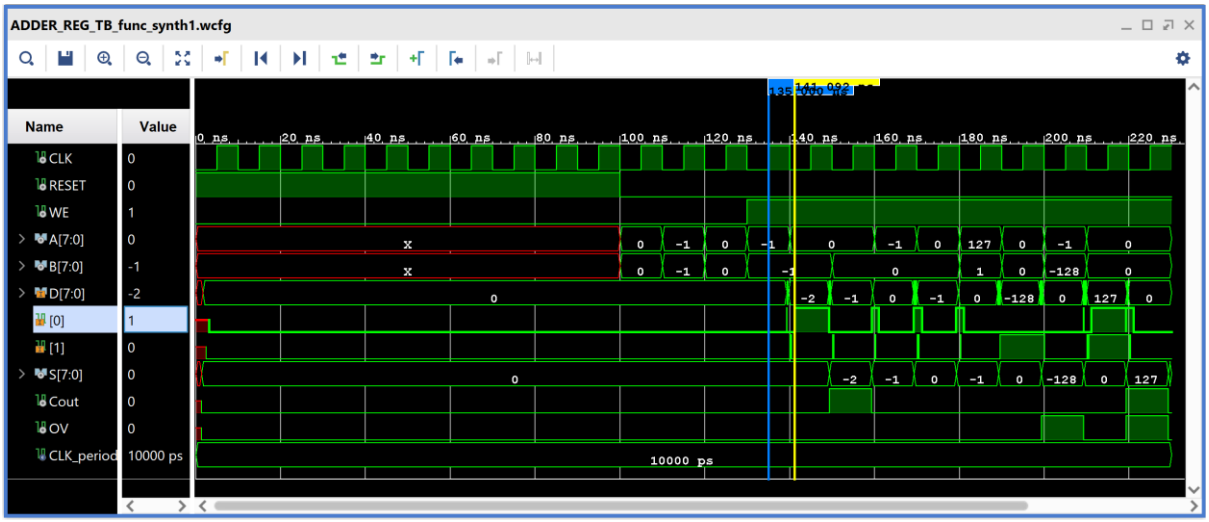
Επιλέξτε το παράθυρο *ADDER_REG_TB_func_synth1.wcfg*. Βλέπετε ολόκληρο το διάγραμμα χρονισμού της χρονικής προσομοίωσης μετά τη σύνθεση με κατάλληλο **zoom out** ή επιλέγοντας το **zoom fit** . Για να επαναφέρετε το floating παράθυρο πίσω, απλά επιλέξτε το κουμπί Dock Window.

Προσοχή! Εάν δεν θέλετε να βλέπετε άλλα ενεργοποιημένα παράθυρα διαγραμμάτων χρονισμού κατά τη διάρκεια της προσομοίωσης μπορείτε να τα απενεργοποιήσετε. Στο παράθυρο *Sources* επιλέξτε με δεξί κλικ το παράθυρο διαγραμμάτων χρονισμού που θέλετε να απενεργοποιήσετε και πατήστε **Disable File**. Τα απενεργοποιημένα παράθυρα διαγραμμάτων χρονισμού μεταφέρονται στα *Disabled Sources*. Για να ενεργοποιήσετε πάλι κάποιο παράθυρο διαγραμμάτων χρονισμού επιλέξτε το με δεξί κλικ και πατήστε **Enable File**.



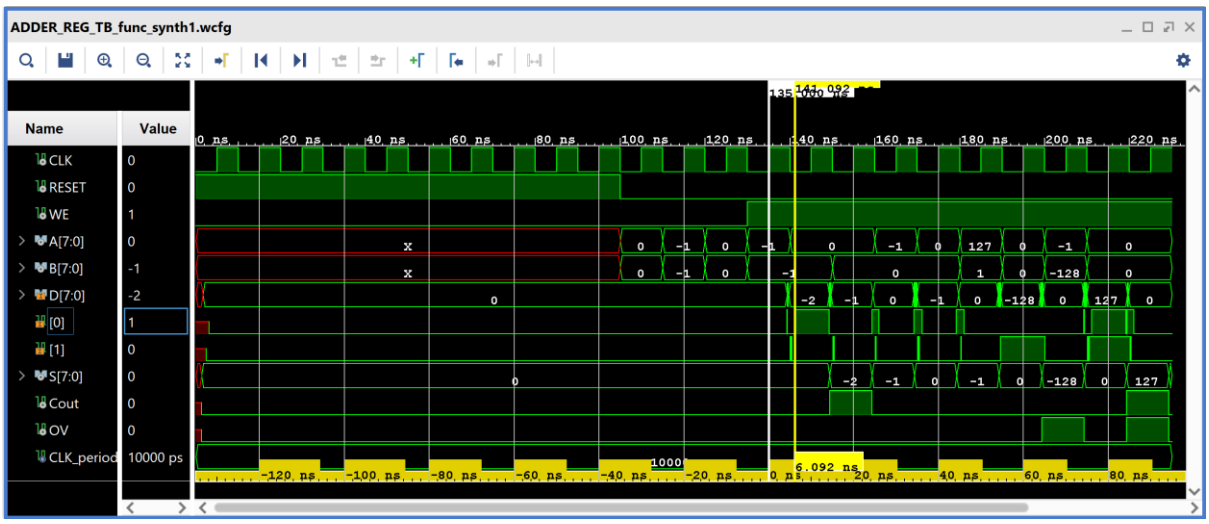
Εικόνα 111 – Διάγραμμα χρονισμού χρονικής προσομοίωσης

- Συγκρίνετε τα διαγράμματα χρονισμού της χρονικής προσομοίωσης μετά τη σύνθεση και της λογικής προσομοίωσης μετά τη σύνθεση. Είναι εμφανείς οι καθυστερήσεις διάδοσης στο πρώτο διάγραμμα χρονισμού.
- Υπολογίστε το **Arrival Time** ενός σήματος στο διάγραμμα χρονισμού της χρονικής προσομοίωσης μετά τη σύνθεση με τη χρήση των marker. Για παράδειγμα, βάλτε τον μπλε marker στην ανερχόμενη ακμή του **CLK** στα 135.000 ns και τον κίτρινο marker στην ανερχόμενη ακμή του εσωτερικού σήματος **[0]** (που αντιστοιχεί στο Cout) στα 141.092 ns. Απέχουν 6.092 ns. Συγκρίνετε με το **Arrival Time** που είχατε βρει κατά τη χρονική ανάλυση για την κρίσιμη διαδρομή, που ήταν 6.401 ns.



Εικόνα 112 – Επαλήθευση του arrival time (βήμα #1)

Με κλικ πάνω στον μπλε marker (γίνεται λευκός), ορίζουμε τη θέση του στα 0.000 ns και είναι πλέον εμφανές ότι ο κίτρινος marker απέχει 6.092 ns.



Εικόνα 113 – Επαλήθευση του arrival time (βήμα #2)

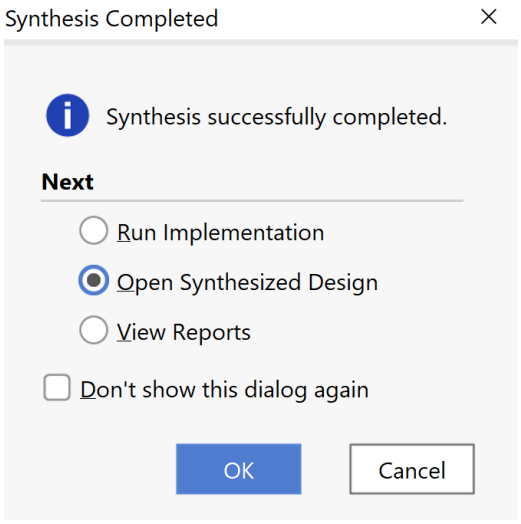
- Κλείστε τον simulator επιλέγοντας το κουμπί × πάνω δεξιά στο παράθυρο του *SIMULATION*. Στο παράθυρο *Confirm Close* που εμφανίζεται πατήστε **OK**. Εάν επιθυμείτε να μη σώσετε ένα επιπλέον waveform configuration, στο παράθυρο *Save Waveform Configuration* επιλέξτε **Discard**.

Η διαδικασία που ήδη περιγράψαμε στα Βήματα 4–1 και 4–2 της «**Σύνθεσης του κώδικα VHDL και προσομοίωση (λογική, χρονική)**» εφαρμόζεται παρομοίως σε κάθε πιθανή υπομονάδα συνδυαστικής λογικής της διαδρομής δεδομένων του επεξεργαστή.

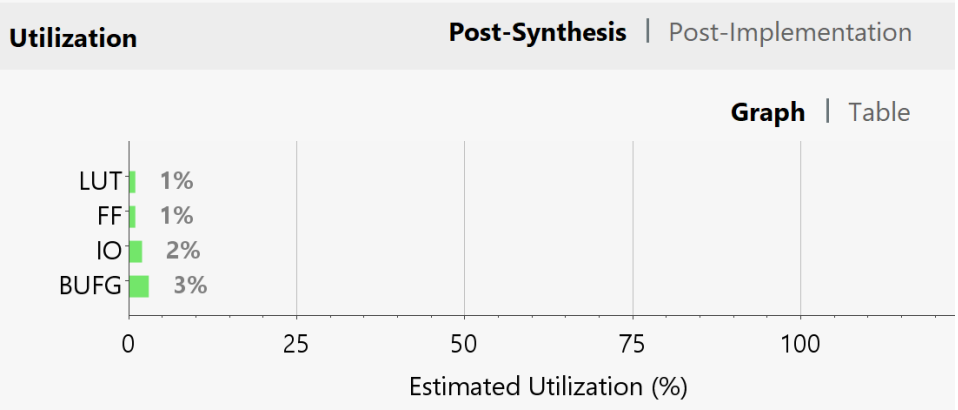
1.9.3 Εκτέλεση της διαδικασίας της σύνθεσης και ανάλυση των αποτελεσμάτων μετά τη σύνθεση στο VIVADO IDE για μηχανή πεπερασμένων καταστάσεων (FSM)

Η σύνθεση εκτελείται στην οντότητα **PATTERN_FSM** που έχει ορισθεί ως η κορυφαία οντότητα της ιεραρχίας (**top**) των design resources, καθώς και σε όλα τα υπάρχοντα αρχεία της ιεραρχίας. Με τη σύνθεση το εργαλείο Vivado IDE παράγει το **synthesized design model** της οντότητας **PATTERN_FSM**.

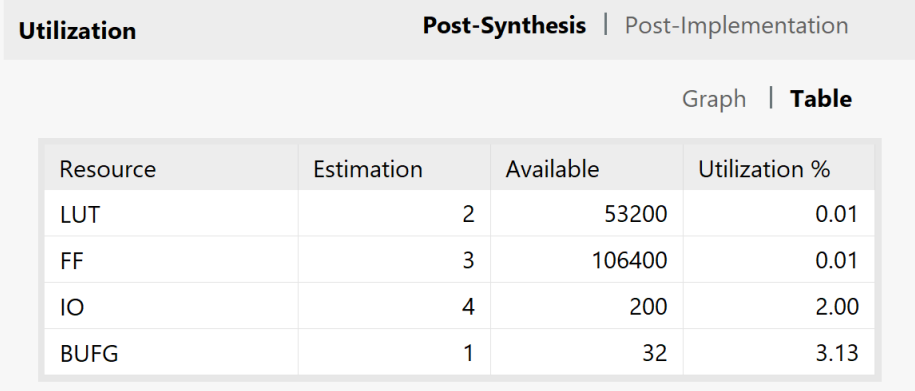
- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Run Synthesis**. Πατήστε **OK** στα παράθυρα προειδο–ποίησης που εμφανίζονται.
- Στο παράθυρο *Synthesis Completed* υπάρχουν τρεις επιλογές. Επιλέξτε το **Open Synthesized Design** και πατήστε **OK** για να μελετήσετε το αποτέλεσμα της σύνθεσης πριν προχωρήσετε στο στάδιο της υλοποίησης (implementation). Πατήστε **Yes** για να κλείσετε το elaborated design, εάν εμφανιστεί το παράθυρο *Close Design*.
- Αρχικά, επιλέξτε το παράθυρο *Project Summary* και μελετήστε τα διάφορα υπο–παράθυρα. Εάν δεν το βλέπετε επιλέξτε το εικονίδιο Project Summary . Μελετήστε στο υποπαράθυρο Utilization τους πόρους που χρησιμοποιεί η οντότητα **PATTERN_FSM** σε μορφή *Graph* και σε μορφή *Table* (στο κάτω μέρος αριστερά).



Εικόνα 114 - Επιλογές μετά τη σύνθεση



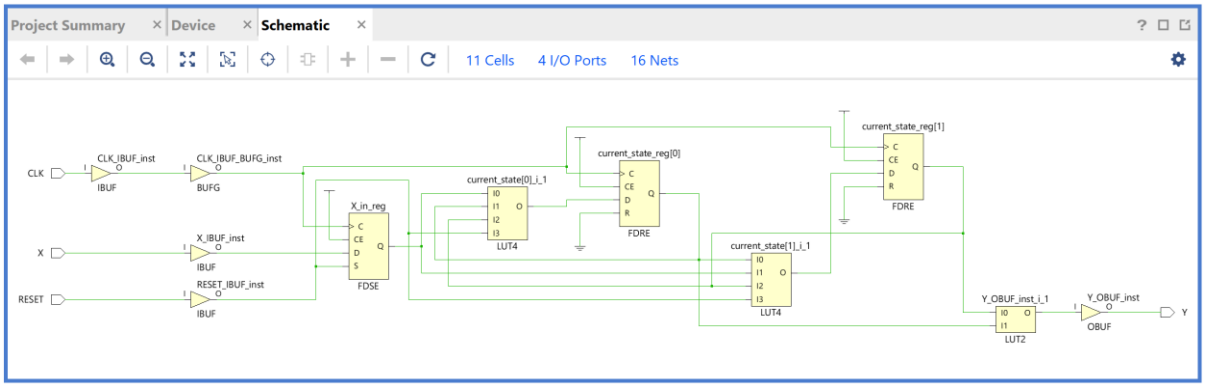
Εικόνα 115 – Πόροι που χρησιμοποιεί η οντότητα PATTERN_FSM (σε μορφή γραφήματος)



Resource	Estimation	Available	Utilization %
LUT	2	53200	0.01
FF	3	106400	0.01
IO	4	200	2.00
BUFG	1	32	3.13

Εικόνα 116 – Πόροι που χρησιμοποιεί η οντότητα PATTERN_FSM (σε μορφή πίνακα)

- Στη συνέχεια, στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, μέσα στο *Open Synthesized Design* επιλέξετε το **Schematic** για να δείτε το σχηματικό διάγραμμα του **synthesized design model**.

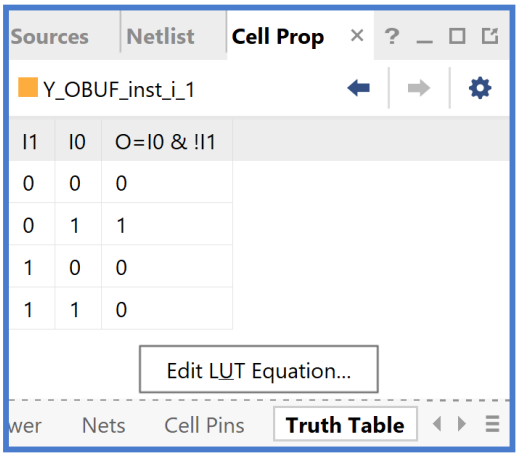


Εικόνα 117 – Σχηματικό διάγραμμα του synthesized design model

Παρατηρείστε ότι έχουν αυτόματα προστεθεί τα απαραίτητα IBUFs, OBUFs και BUFG primitives στο σχηματικό διάγραμμα. Με επιλογή των LUTs μπορείτε να επαληθεύσετε τη λογική που έχετε περιγράψει στον κώδικα VHDL. Για παράδειγμα, η λογική εξόδου έχει περιγραφθεί ως εξής:

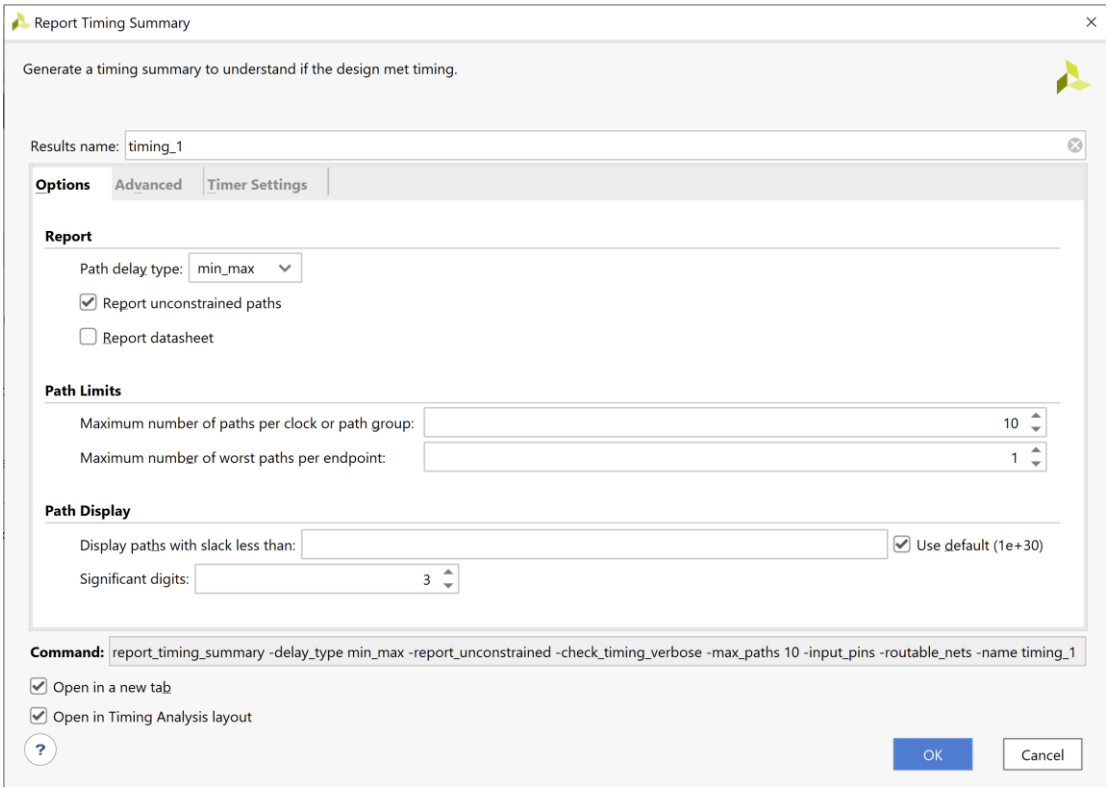
```
if (current_state = S2 = "10") then Y <= '1'; else Y <= '0';
```

Επιλέγοντας το LUT2 με έξοδο Y (που αποκτά μπλε περίγραμμα), επαληθεύουμε τη λογική ρυθμίζοντας την επιλογή *Truth Table* στο παράθυρο *Cell Properties*. Μελετώντας το σχηματικό διάγραμμα παρατηρούμε ότι το I1 είναι η έξοδος του current_state_reg[0], ενώ το I0 είναι η έξοδος του current_state_reg[1].



Εικόνα 118 - Παράδειγμα πίνακα αλήθειας LUT

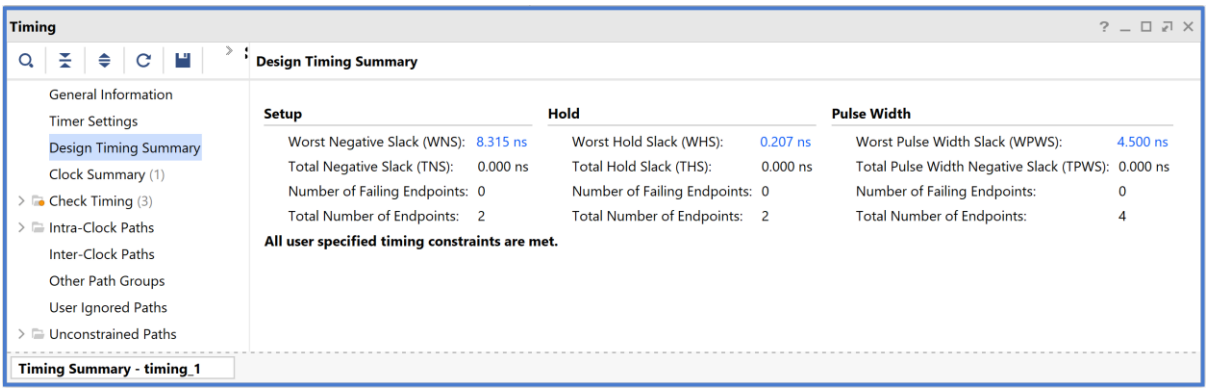
- Στη συνέχεια, στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, μέσα στο *Open Synthesized Design* επιλέξετε το **Report Timing Summary** για να δείτε την ανάλυση χρονισμού που κάνει το εργαλείο Vivado IDE στο **synthesized design model**. Αν και η ανάλυση χρονισμού σε αυτό το επίπεδο δεν είναι ακριβής, χρησιμοποιείται ευρέως στις πολύπλοκες σχεδιάσεις για εξοικονόμηση χρόνου κατά την ανάπτυξη του κώδικα VHDL.



Εικόνα 119 – Επιλογές για την παραγωγή του timing summary για το synthesized design model

Στην επιλογή *Path delay type* ορίζεται ο τύπος της ανάλυσης που θα εκτελεσθεί. Το *max delay analysis* αφορά στην εύρεση της κρίσιμης διαδρομής (με τη μεγαλύτερη καθυστέρηση διάδοσης) και συνεπώς στην εύρεση της μέγιστης συχνότητας λειτουργίας χωρίς την παραβίαση του **χρόνου σταθεροποίησης** (setup time). Το *min delay analysis* αφορά στην εύρεση της σύντομης διαδρομής (με τη μικρότερη καθυστέρηση διάδοσης) χωρίς την παραβίαση του **χρόνου διατήρησης** (hold time).

- Πατήστε **OK** για να παραχθεί το Timing_1 report.



Εικόνα 120 – Timing report summary για το synthesized design model

Στη στήλη **Setup** παρουσιάζονται τα αποτελέσματα του *max delay analysis*.

- Το *Worst Negative Slack* (**WNS**) είναι μια τιμή (με link) που αντιστοιχεί στο μικρότερο διαθέσιμο περιθώριο (slack) που προκύπτει από την ανάλυση όλων των κρίσιμων διαδρομών του max delay analysis. Μπορεί να είναι θετικό η αρνητικό. Ένα θετικό slack (8.315 ns) δηλώνει ότι η κρίσιμη διαδρομή ικανοποιεί την περίοδο του CLK. Ένα αρνητικό slack δηλώνει ότι παραβιάζεται ο χρόνος σταθεροποίησης (setup time).
- Το *Total Negative Slack* (**TNS**) είναι το άθροισμα όλων των αρνητικών WNS για κάθε timing path endpoint. Η τιμή είναι 0.000 ns όταν δεν υπάρχουν παραβιάσεις του χρόνου σταθεροποίησης. Το ψηφιακό κύκλωμα λειτουργεί κανονικά στην επιλεγμένη συχνότητα λειτουργίας.
- Το *Number of Failing Endpoints* αφορά στον συνολικό αριθμό των endpoints που έχουν παραβιάσει το χρόνο σταθεροποίησης (αρνητικό WNS).
- Το *Total Number of Endpoints* αφορά στον συνολικό αριθμό των endpoints που έχουν αναλυθεί.

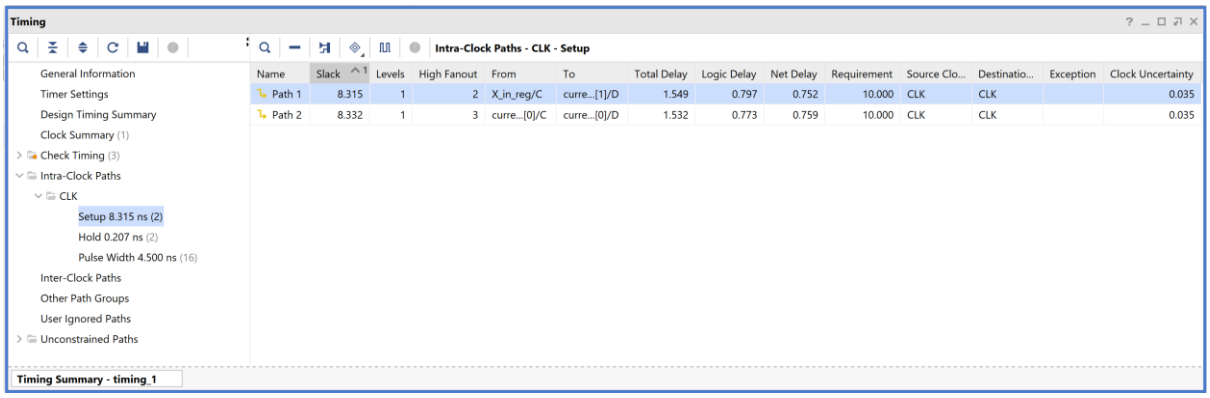
Στη στήλη **Hold** παρουσιάζονται τα αποτελέσματα του min delay analysis.

- Το *Worst Hold Slack* (**WHS**) είναι μια τιμή (με link) που αντιστοιχεί στο μικρότερο διαθέσιμο περιθώριο (slack) που προκύπτει από την ανάλυση όλων των σύντομων διαδρομών του min delay analysis. Μπορεί να είναι θετικό η αρνητικό. Ένα αρνητικό slack δηλώνει ότι παραβιάζεται ο χρόνος διατήρησης (hold time).
- Το *Total Hold Slack* (**THS**) είναι το άθροισμα όλων των αρνητικών WHS για κάθε timing path endpoint. Η τιμή είναι 0.000 ns όταν δεν υπάρχουν παραβιάσεις του χρόνου διατήρησης. Το ψηφιακό κύκλωμα λειτουργεί κανονικά.
- Το *Number of Failing Endpoints* αφορά στον συνολικό αριθμό των endpoints που έχουν παραβιάσει το χρόνο διατήρησης (αρνητικό WHS).
- Το *Total Number of Endpoints* αφορά στον συνολικό αριθμό των endpoints που έχουν αναλυθεί.

Στη στήλη **Pulse Width** παρουσιάζονται τα περιθώρια του σήματος CLK, όταν είναι HIGH ή LOW.

- Επιλέξτε το **WNS link** και δείτε τις 2 διαθέσιμες κρίσιμες διαδρομές (δηλαδή με το μικρότερο θετικό slack) που δεν παραβιάζουν το χρόνο σταθεροποίησης. Η κρίσιμη διαδρομή έχει καθυστέρηση διάδοσης 1.549 ns, εκ των οποίων τα 0.797 ns αφορούν στη λογική (logic), ενώ τα 0.752 ns αφορούν στη δικτύωση (net). Η

αβεβαιότητα του CLK εκτιμάται στα 0.035 ns. Τα επίπεδα λογικής (logic level) είναι 1.

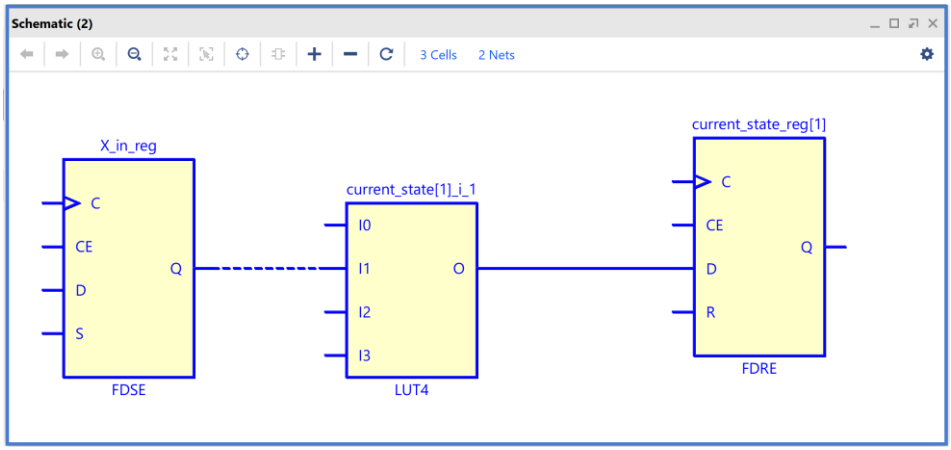


Εικόνα 121 – Μελέτη κρίσιμης διαδρομής στο synthesized design model

- Επιλέξτε με διπλό κλικ το **Path 1**, ώστε να εμφανιστεί το παράθυρο *Path 1 – timing_1*. Η κρίσιμη διαδρομή περνάει από 1 LUT4. Υπολογίστε το WNS slack:
- η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Source Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK του καταχωρητή εισόδων) είναι 2.975 ns,
- η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Data Path* (από την είσοδο CLK μέχρι την έξοδο Q του καταχωρητή εισόδων και μέσω του LUT4 μέχρι την είσοδο D του καταχωρητή κατάστασης 1) είναι 1.549 ns,
- το **Arrival Time**, ως άθροισμα των ανωτέρων χρόνων, είναι 4.524 ns,
- το **Required Time**, ως η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Destination Clock Path* (από την πηγή του CLK τη χρονική στιγμή 10.000 ns μέχρι την είσοδο CLK του καταχωρητή κατάστασης 1) συν τον χρόνο σταθεροποίησης είναι 12.839 ns.

  WNS slack = Required Time – Arrival Time = 12.839 – 4.524 = 8.315 ns

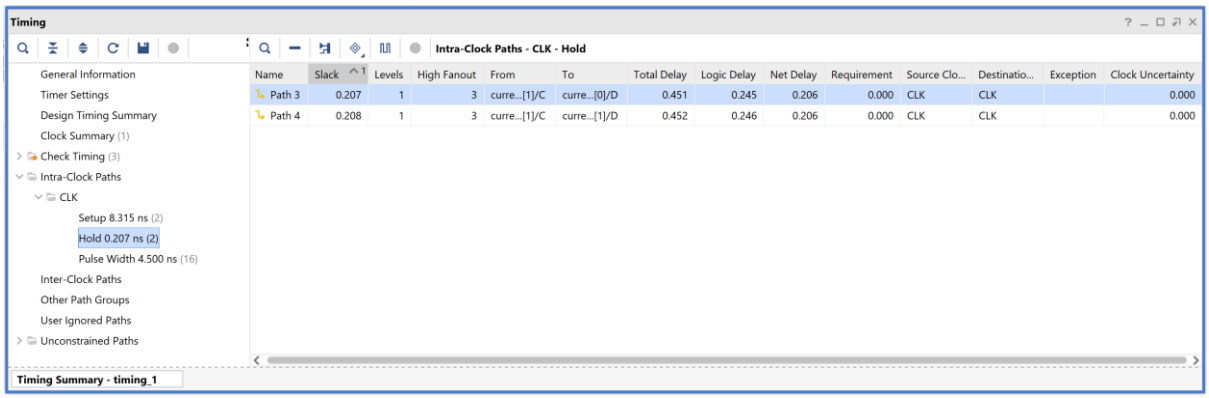
- Επιλέξτε με δεξί κλικ στο **Path 1**, το **Schematic**. Μελετήστε το σχηματικό διάγραμμα της κρίσιμης διαδρομής της οντότητας **PATTERN_FSM**.



Εικόνα 122 – Σχηματικό διάγραμμα κρίσιμης διαδρομής στο synthesized design model

- Επιλέξτε το **WHS link** και δείτε τις 2 σύντομες διαδρομές (δηλαδή με το μικρότερο θετικό slack) που δεν παραβιάζουν το χρόνο διατήρησης. Η σύντομη διαδρομή έχει καθυστέρηση μόλυνσης 0.451 ns, εκ των οποίων τα 0.245 ns αφορούν στη λογική

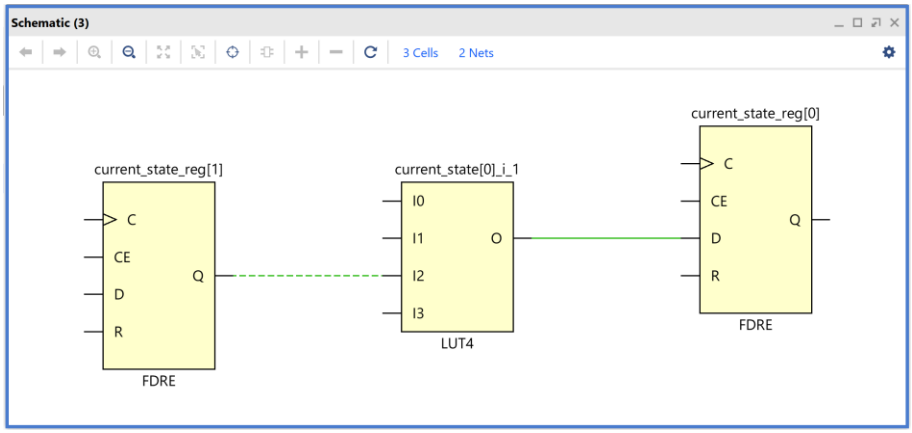
(logic), ενώ τα 0.206 ns αφορούν στη δικτύωση (net). Η αβεβαιότητα του CLK εκτιμάται στα 0.000 ns. Τα επίπεδα λογικής (logic level) είναι 1.



Εικόνα 123 – Μελέτη χειρότερης σύντομη διαδρομής (θετικό slack) του synthesized design model

- Επιλέξτε με διπλό κλικ το **Path 3**, ώστε να εμφανιστεί το παράθυρο *Path 3 – timing_1*. Η σύντομη διαδρομή περνάει από 1 μονάδα LUT4. Υπολογίστε το WHS slack:
- η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Source Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK του καταχωρητή κατάστασης 1) είναι 0.735 ns,
- η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Data Path* (από την είσοδο CLK μέχρι την έξοδο Q του καταχωρητή κατάστασης 1 και μέσω του LUT4 μέχρι την είσοδο D του καταχωρητή κατάστασης 0) είναι 0.451 ns,
- το **Arrival Time**, ως άθροισμα των ανωτέρων χρόνων, είναι 1.186 ns,
- το **Required Time**, ως η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Destination Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK στον καταχωρητή κατάστασης 0) συν τον χρόνο διατήρησης, είναι 0.979 ns.

  WHS slack = Arrival Time – Required Time = 1.186 – 0.979 = 0.207 ns
- Επιλέξτε με δεξί κλικ στο **Path 3**, το **Schematic**. Μελετήστε το σχηματικό διάγραμμα της σύντομης διαδρομής της οντότητας **PATTERN_FSM**.



Εικόνα 124 – Σχηματικό διάγραμμα χειρότερης σύντομης διαδρομής του synthesized design model

1.9.4 Εκτέλεση προσομοίωσης μετά τη σύνθεση (λογική και χρονική) στο VIVADO IDE για μηχανή πεπερασμένων καταστάσεων (FSM)

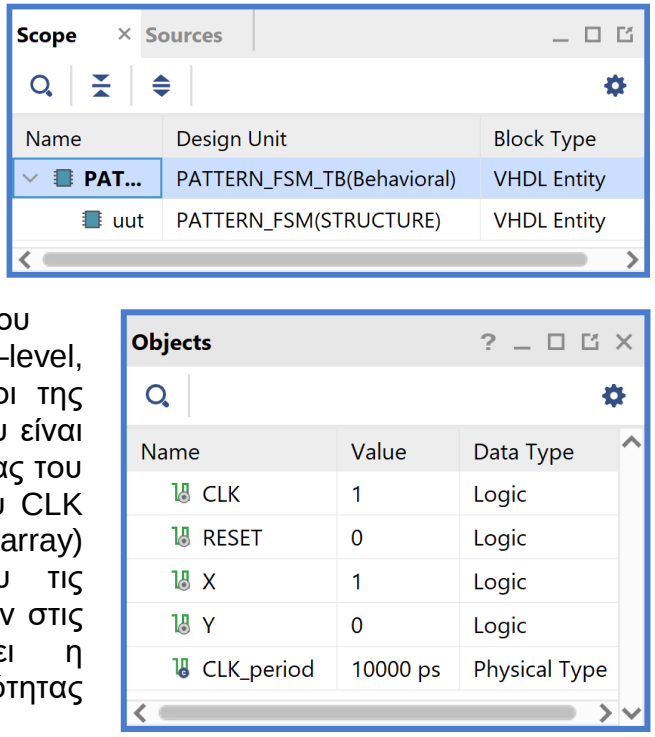
Η προσομοίωση μετά τη σύνθεση (λογική και χρονική) εκτελείται στην οντότητα **PATTERN_FSM_TB** του προγράμματος δοκιμών (testbench) που έχει ορισθεί ως η κορυφαία οντότητα της ιεραρχίας (**top**) των simulation resources. Κατά την προσομοίωση, η οντότητα **PATTERN_FSM_TB** καλεί το *UUT* της (που είναι το **synthesized design model** της οντότητας **PATTERN_FSM**).

- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Run Simulation**, ώστε να εμφανιστούν όλες οι πιθανές προσομοιώσεις που υποστηρίζει το Vivado IDE. Επιλέξτε **Run Post–Synthesis Functional Simulation**.
- Το πρόγραμμα δοκιμής (testbench) και όλες οι οντότητες του *UUT* θα γίνουν compiled και θα τρέξει το Vivado simulator (εφόσον βέβαια δεν υπάρχουν σφάλματα). Θα εμφανιστεί το παράθυρο *SIMULATION* που απαρτίζεται από 4 παράθυρα:

  Το παράθυρο *Scope*, όπου παρουσιάζεται η οντότητα **PATTERN_FSM_TB** και το **synthesized design model** της οντότητας **PATTERN_ FSM** (*UUT*) που προκύπτει μετά τη σύνθεση, η οποία πλέον είναι *structural*.

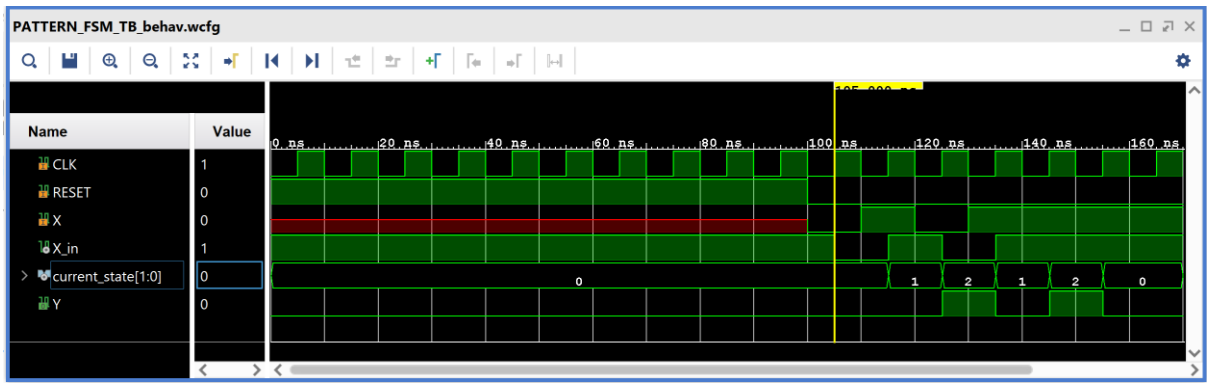
  Το παράθυρο *Objects*, όπου εμφανίζονται τα σήματα top–level, δηλαδή οι είσοδοι και οι έξοδοι της οντότητας **PATTERN_ FSM**, που είναι η κορυφαία οντότητα της ιεραρχίας του *UUT*, καθώς και η περίοδος του CLK (*CLK_period*). Οι αρτηρίες (array) αναλύονται στα σήματα που τις απαρτίζουν. Οι τιμές αντιστοιχούν στις τιμές που έχει σταματήσει η προσομοίωση της οντότητας **PATTERN_ FSM_TB** (*stop (2)*).

  Το παράθυρο *Tcl Console* με όλες τις διαδικασίες που εκτελούνται στο πλαίσιο της προσομοίωσης. Το Tcl Console καθαρίζει με την επιλογή *Clear*.



Εικόνα 125 - Παράθυρα Scope και Objects

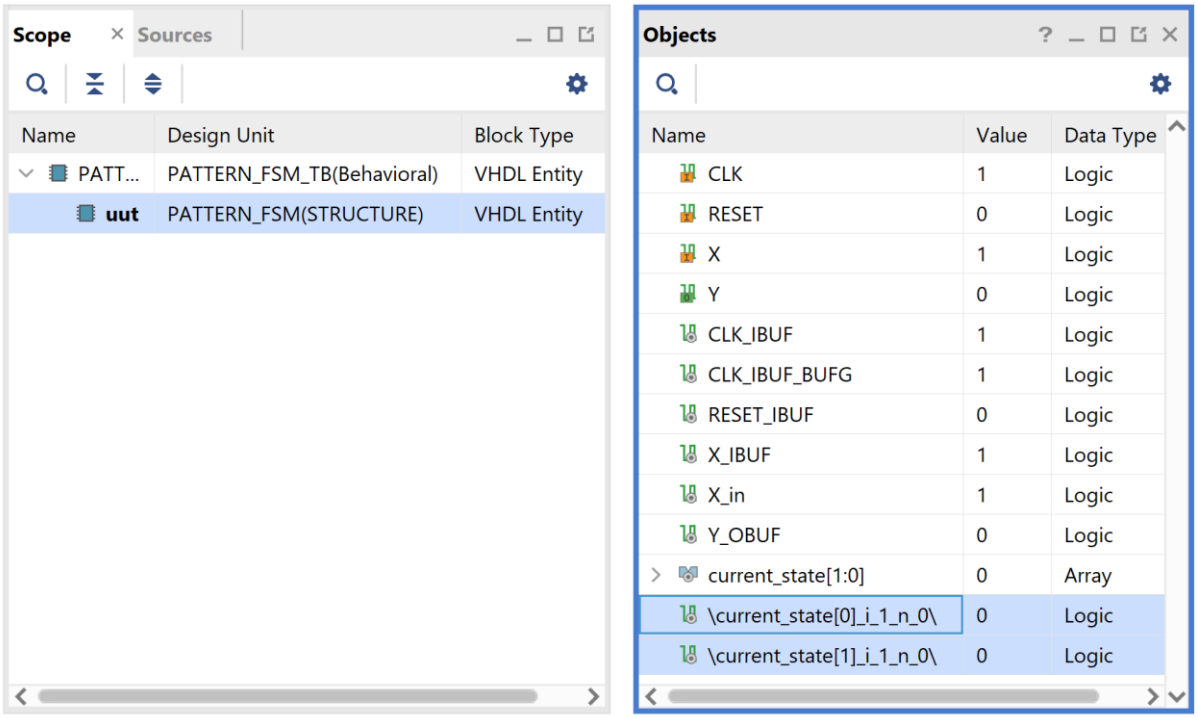
Επιλέξτε το παράθυρο *PATTERN_ FSM_TB_behav.wcfg*, που δημιουργήσατε για την προσομοίωση συμπεριφοράς. Βλέπετε ολόκληρο το διάγραμμα χρονισμού της λογικής προσομοίωσης μετά τη σύνθεση με κατάλληλο **zoom out** ή επιλέγοντας το **zoom fit** . Για να επαναφέρετε το floating παράθυρο πίσω, απλά επιλέξτε το κουμπί Dock Window.



Εικόνα 126 – Διάγραμμα χρονισμού προσομοίωσης μετά τη σύνθεση

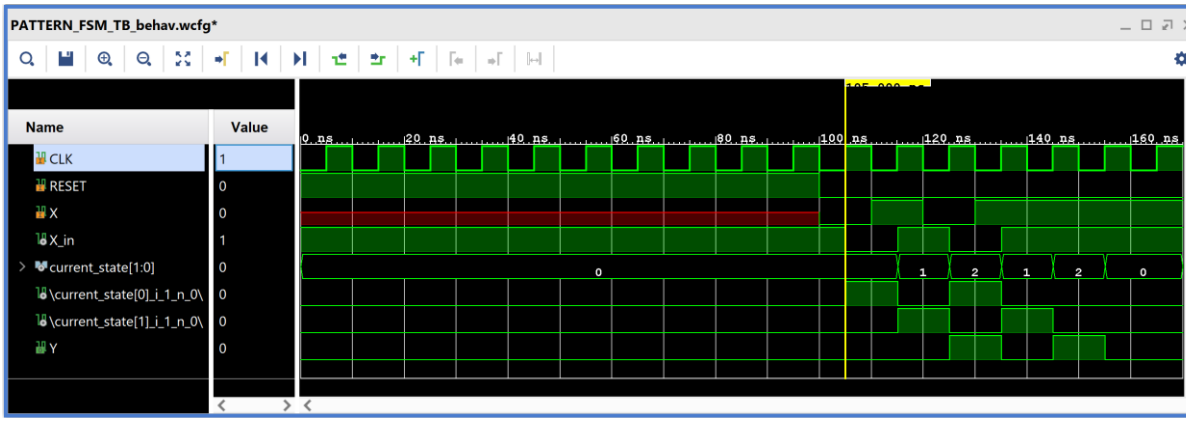
- Συγκρίνετε τα διαγράμματα χρονισμού της λογικής προσομοίωσης μετά τη σύνθεση και της προσομοίωσης συμπεριφοράς (υπάρχει ταύτιση στο επίπεδο εισόδων/ εξόδων και εσωτερικών σημάτων X_in και current_state, όπου S0 = 0, S1 = 1, S2 = 2).
- Προσθέστε περισσότερα **εσωτερικά σήματα** στο διάγραμμα χρονισμού της προσομοίωσης μετά τη σύνθεση. Μετά από μελέτη του σχηματικού διαγράμματος του **synthesized design model** προκύπτει ότι οι είσοδοι **D** των FDRE **current_state _reg[0]** (**current_state[0]_i_1_n_0**) και **current_state _reg[1]** (**current_state[1] _i_1_n_0**) αντιστοιχούν στο **next_state** της οντότητας **PATTERN_ FSM** (του **elaborated design model** πριν τη σύνθεση) και για αυτό το λόγο τις προσθέτουμε στο διάγραμμα χρονισμού της προσομοίωσης μετά τη σύνθεση.

  Στο παράθυρο *Scope* επιλέξτε την οντότητα **PATTERN_ FSM** (*UUT*). Παρατηρείστε ότι στο παράθυρο Objects εμφανίζονται όλα τα σήματα του *UUT*. Επιλέξτε τα εσωτερικά σήματα **current_state[0]_i_1_n_0** (αντιστοιχεί στο next_state[0]) και **current_ state[1]_i_1_n_0** (αντιστοιχεί στο next_state[1]) και κάντε drag and drop στο παράθυρο με τα διαγράμματα χρονισμού, τοποθετώντας τα κάτω από το **current_state**.



Εικόνα 127 - Παράθυρα Scope και Objects

Επαναλάβετε τη διαδικασία της προσομοίωσης από την αρχή με επιλογή του κουμπιού **Restart** και στη συνέχεια του κουμπιού **Run All**. Και τα δύο κουμπιά βρίσκονται στην οριζόντια μπάρα στο πάνω μέρος.



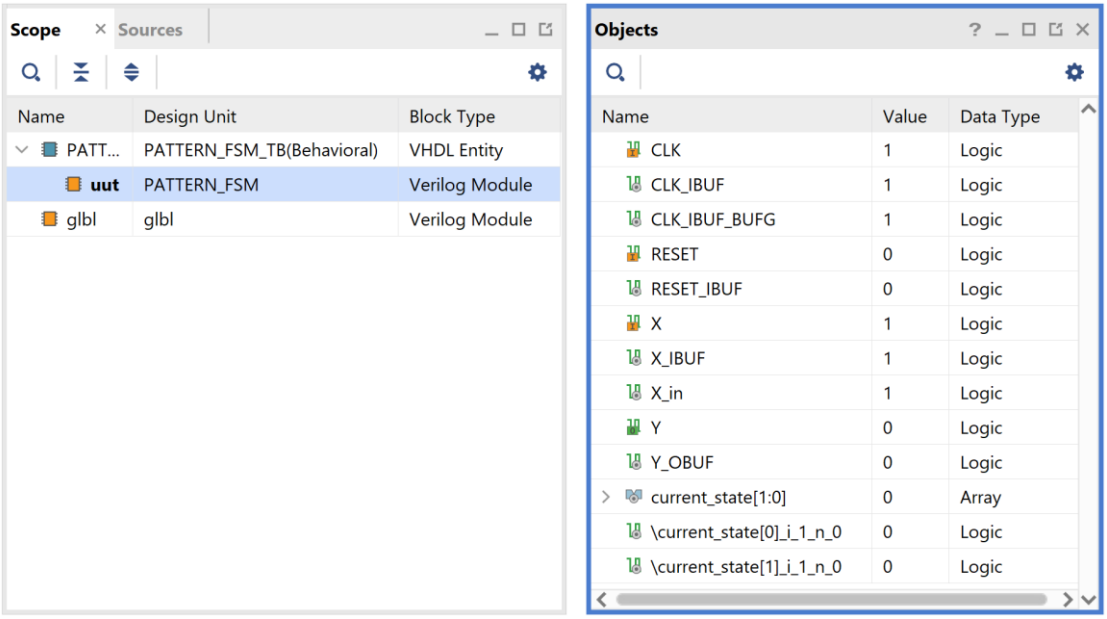
Εικόνα 128 – Διάγραμμα χρονισμού προσομοίωσης μετά τη σύνθεση (με εσωτερικά σήματα)

- Συγκρίνετε τη λογική προσομοίωση μετά τη σύνθεση με την προσομοίωση συμπεριφοράς (υπάρχει ταύτιση στο επίπεδο εισόδων/εξόδων και επιλεγμένων εσωτερικών σημάτων).
- Επιλέξτε αρχικά **File** στη συνέχεια **Simulation Waveform** και τέλος **Save Configuration as ..** για να αποθηκεύσετε το configuration του διαγράμματος χρονισμού στο οποίο έχετε καταλήξει με νέο όνομα (π.χ. **PATTERN_FSM_TB _func_synth.wcfg**). Στο παράθυρο *Waveform Configuration File* που εμφανίζεται πατήστε **Yes**.
- Κλείστε τον simulator επιλέγοντας το κουμπί × πάνω δεξιά στο παράθυρο του *SIMULATION*. Στο παράθυρο *Confirm Close* που εμφανίζεται πατήστε **OK**.

- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Run Simulation**, ώστε να εμφανιστούν όλες οι πιθανές προσομοιώσεις που υποστηρίζει το Vivado IDE. Επιλέξτε **Run Post–Synthesis Timing Simulation**.
- Το πρόγραμμα δοκιμής (testbench) και όλες οι οντότητες του *UUT* θα γίνουν compiled και θα τρέξει το Vivado simulator (εφόσον βέβαια δεν υπάρχουν σφάλματα). Θα εμφανιστεί το παράθυρο *SIMULATION* που απαρτίζεται από 4 παράθυρα:

  Το παράθυρο *Scope*, όπου παρουσιάζεται η οντότητα **PATTERN_ FSM_TB** και το **synthesized design model** της οντότητας **PATTERN_ FSM** (*UUT*) που προκύπτει μετά τη σύνθεση για χρονική προσομοίωση. Όλες οι οντότητες που απαρτίζουν πλέον το *UUT* είναι *Verilog Module!*

  Το παράθυρο *Objects*, όπου εμφανίζονται τα σήματα της οντότητας **PATTERN_ FSM** (*UUT*). Οι αρτηρίες (array) αναλύονται στα σήματα που τις απαρτίζουν. Οι τιμές αντιστοιχούν στις τιμές που έχει σταματήσει η προσομοίωση της οντότητας **PATTERN_ FSM_TB** (st*op (2)*).

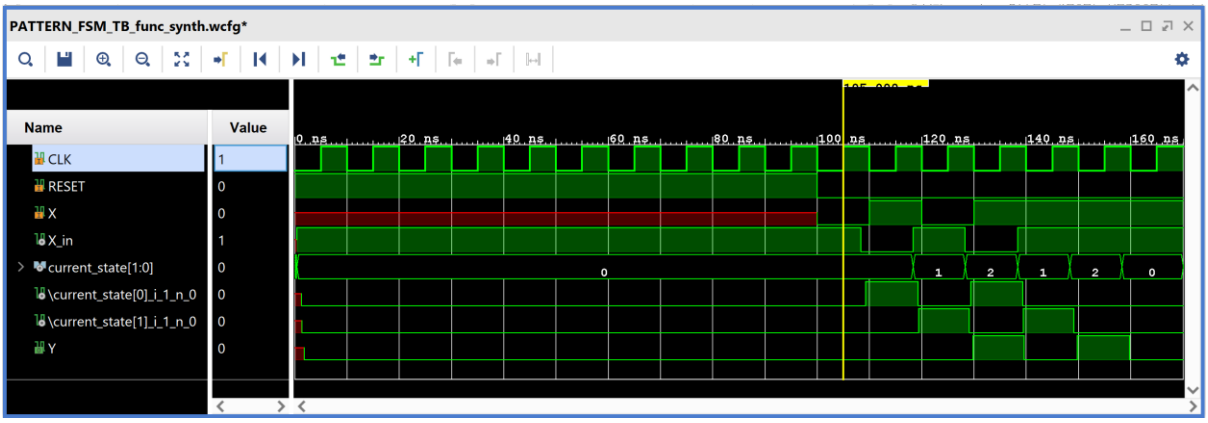


Εικόνα 129 - Παράθυρα Scope και Objects

Το παράθυρο *Tcl Console* με όλες τις διαδικασίες που εκτελούνται στο πλαίσιο της προσομοίωσης. Το Tcl Console καθαρίζει με την επιλογή *Clear*.

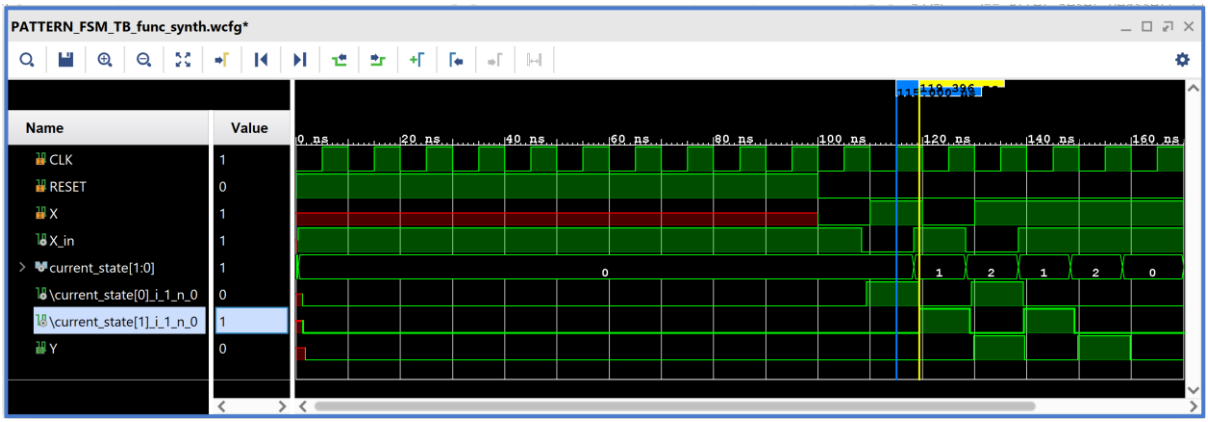
Επιλέξτε το παράθυρο *PATTERN_ FSM_TB_func_synth.wcfg*. Βλέπετε ολόκληρο το διάγραμμα χρονισμού της χρονικής προσομοίωσης μετά τη σύνθεση με κατάλληλο **zoom out** ή επιλέγοντας το **zoom fit**.

Σημείωση: Εάν δεν θέλετε να βλέπετε άλλα ενεργοποιημένα παράθυρα διαγραμμάτων χρονισμού κατά τη διάρκεια της προσομοίωσης μπορείτε να τα απενεργοποιήσετε. Στο παράθυρο *Sources* επιλέξτε με δεξί κλικ το παράθυρο διαγραμμάτων χρονισμού που θέλετε να απενεργοποιήσετε και πατήστε **Disable File**. Τα απενεργοποιημένα παράθυρα διαγραμμάτων χρονισμού μεταφέρονται στα *Disabled Sources*. Για να ενεργοποιήσετε πάλι κάποιο παράθυρο διαγραμμάτων χρονισμού επιλέξτε το με δεξί κλικ και πατήστε **Enable File**.



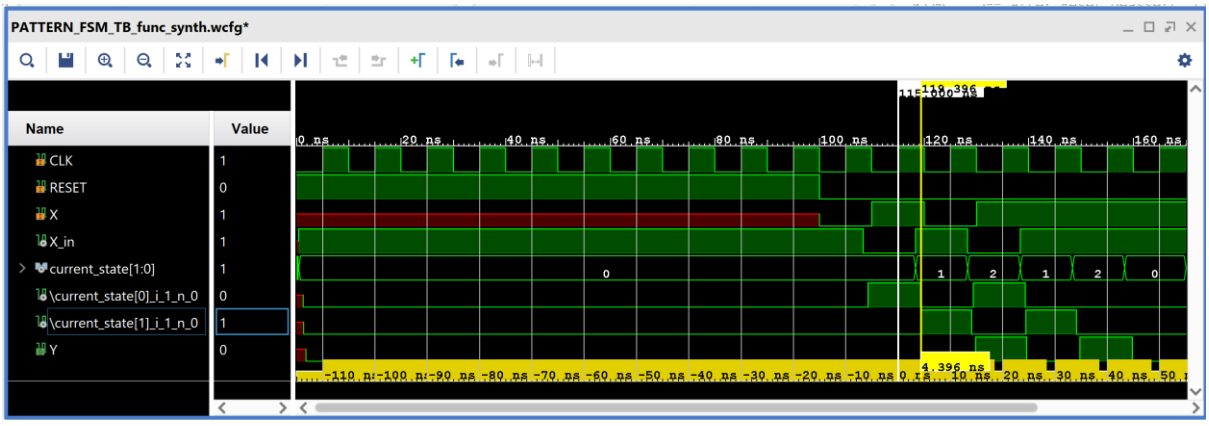
Εικόνα 130 – Διάγραμμα χρονισμού χρονικής προσομοίωσης μετά τη σύνθεση

- Συγκρίνετε τα διαγράμματα χρονισμού της χρονικής και της λογικής προσομοίωσης μετά τη σύνθεση. Είναι εμφανείς οι καθυστερήσεις διάδοσης στο πρώτο διάγραμμα χρονισμού.
- Υπολογίστε το **Arrival Time** ενός σήματος στο διάγραμμα χρονισμού της χρονικής προσομοίωσης μετά τη σύνθεση με τη χρήση των marker. Για παράδειγμα, βάλτε τον μπλε marker στην ανερχόμενη ακμή του **CLK** στα 115.000 ns και τον κίτρινο marker στην ανερχόμενη ακμή του εσωτερικού σήματος **current_state[1]_i_1_n_0** (που αντιστοιχεί στο next_state[1]) στα 119.396 ns. Απέχουν 4.396 ns. Συγκρίνετε με το **Arrival Time** που είχατε βρει κατά τη χρονική ανάλυση για την κρίσιμη διαδρομή, που ήταν 4.524 ns.



Εικόνα 131 – Επαλήθευση του arrival time (βήμα #1)

Με κλικ πάνω στον μπλε marker (γίνεται λευκός), ορίζουμε τη θέση του στα 0.000 ns και είναι πλέον εμφανές ότι ο κίτρινος marker απέχει 4.396 ns.



Εικόνα 132 – Επαλήθευση του arrival time (βήμα #2)

- Κλείστε τον simulator επιλέγοντας το κουμπί × πάνω δεξιά στο παράθυρο του *SIMULATION*. Στο παράθυρο *Confirm Close* που εμφανίζεται πατήστε **OK**. Εάν επιθυμείτε να μη σώσετε ένα επιπλέον waveform configuration, στο παράθυρο *Save Waveform Configuration* επιλέξτε **Discard**.

Η διαδικασία που ήδη περιγράψαμε στα Βήματα 4–3 και 4–4 της «**Σύνθεσης του κώδικα VHDL και προσομοίωση (λογική, χρονική)**» εφαρμόζεται παρομοίως σε κάθε πιθανή μηχανή πεπερασμένων καταστάσεων (FSM).

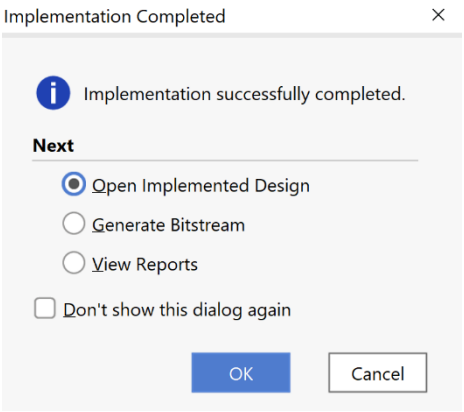
1.10 Βήμα 5: Υλοποίηση στη τεχνολογία FPGA και προσομοίωση (λογική, χρονική)

1.10.1 Εκτέλεση της διαδικασίας της υλοποίησης και ανάλυση των αποτελεσμάτων μετά την υλοποίηση στο VIVADO IDE για συνδυαστική λογική ανάμεσα σε καταχωρητές

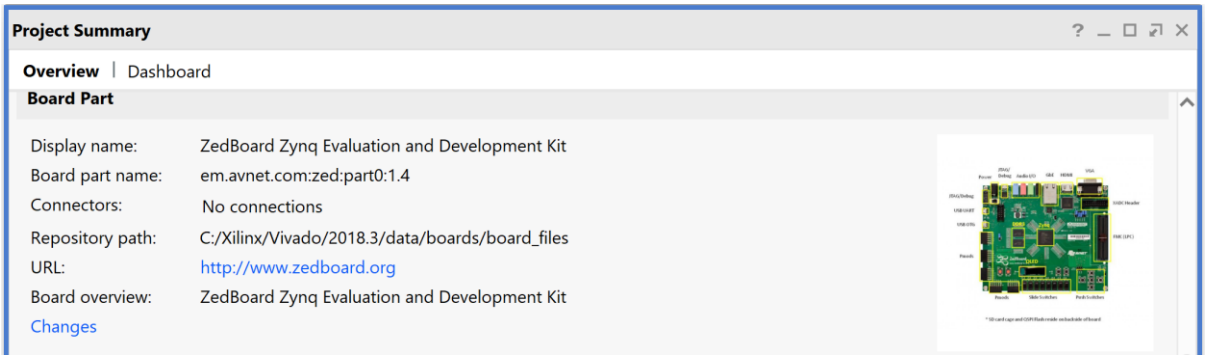
Σε μία διαδικασία bottom–up σχεδίασης και υλοποίησης ενός ψηφιακού συστήματος, η υλοποίηση σε συγκεκριμένη τεχνολογία FPGA εφαρμόζεται κυρίως στα ανώτερα ιεραρχικά επίπεδα, όταν η σχεδίαση έχει πλέον ωριμάσει στο επίπεδο της αποσφαλμάτωσης και της βελτιστοποίησης. Η διαδικασία της υλοποίησης εφαρμόζεται στο **synthesized design model** της οντότητας που έχει ορισθεί ως η κορυφαία οντότητα της ιεραρχίας (**top**) των design resources και μετά από την ολοκλήρωση της υλοποίησης παράγεται το αντίστοιχο **implemented design model**.

Η υλοποίηση στην τεχνολογία **Zynq–7000 (device xc7z020clg484–1)** εκτελείται στο **synthesized design model** της οντότητας **ADDER_REG_8** που έχει ορισθεί ως η κορυφαία οντότητα της ιεραρχίας (**top**) των design resources, καθώς και σε όλα τα υπάρχοντα αρχεία της ιεραρχίας. Με τη σύνθεση το εργαλείο Vivado IDE παράγει το **implemented design model** της οντότητας **ADDER_REG_8**.

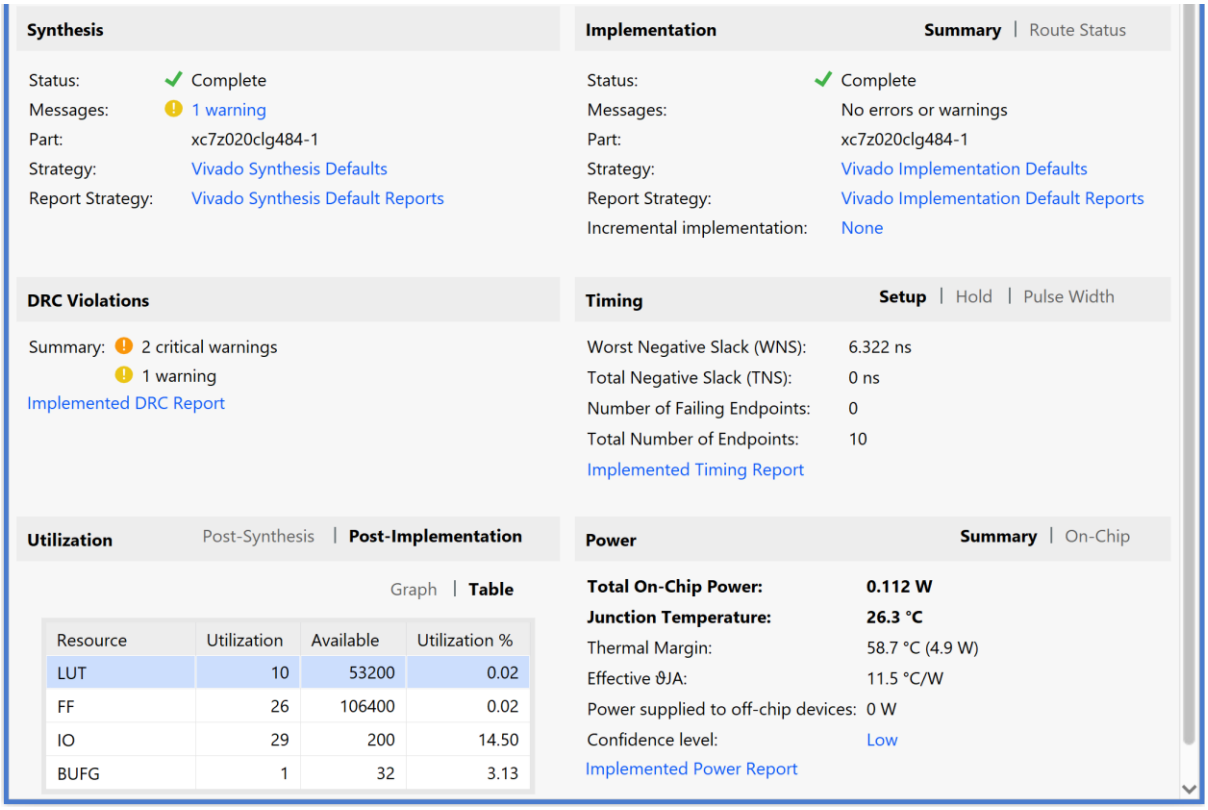
- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Run Implementation**. Πατήστε **OK** στα παράθυρα προειδοποίησης που εμφανίζονται.
- Στο παράθυρο *Implementation Completed* υπάρχουν τρεις επιλογές. Επιλέξτε το **Open Implemented Design** και πατήστε **OK** για να μελετήσετε το αποτέλεσμα της υλοποίησης (implementation).
- Αρχικά, επιλέξτε το παράθυρο *Project Summary* και μελετήστε τα διάφορα υπο–παράθυρα.



Εικόνα 133 - Επιλογές μετά την υλοποίηση

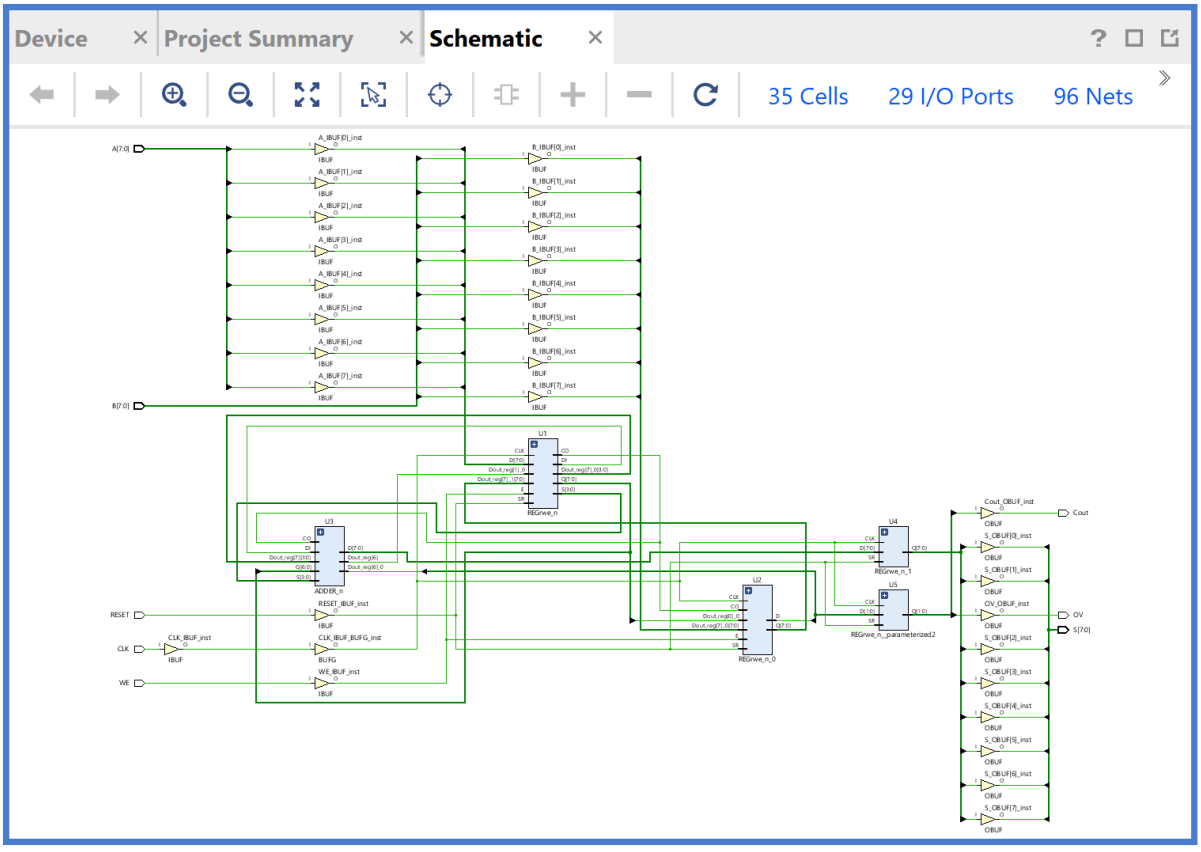


(η εικόνα συνεχίζεται στην επόμενη σελίδα)



Εικόνα 134 – Παράθυρο Project Summary του implemented design model

- Στη συνέχεια, στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, μέσα στο *Open Implemented Design* επιλέξετε το **Schematic** για να δείτε το σχηματικό διάγραμμα του **implemented design model**.



Εικόνα 135 – Σχηματικό διάγραμμα του implemented design model

Παρατηρείστε ότι έχουν αυτόματα προστεθεί τα απαραίτητα IBUFs, OBUFs και BUFG primitives στο σχηματικό διάγραμμα, καθώς και ότι οι είσοδοι και οι έξοδοι από το FPGA είναι buffered. Επίσης, παρατηρείστε ότι έχει διατηρηθεί εν μέρει η ιεραρχία που έχει ορισθεί στην οντότητα **ADDER_REG_8**. Πατήστε το (+) σε όλες τις υπομονάδες U1–U5 και μελετήστε τις μία προς μία. Είναι όμοιες με τις υπομονάδες που παράχθηκαν μετά τη σύνθεση.

Υπομονάδα U1 (REGrwe_n): Συμπεριλαμβάνει τον καταχωρητή εισόδου A των 8 bit, 8 πύλες XOR (LUT2), έναν αντιστροφέα (LUT1) και μία μονάδα CARRY4.

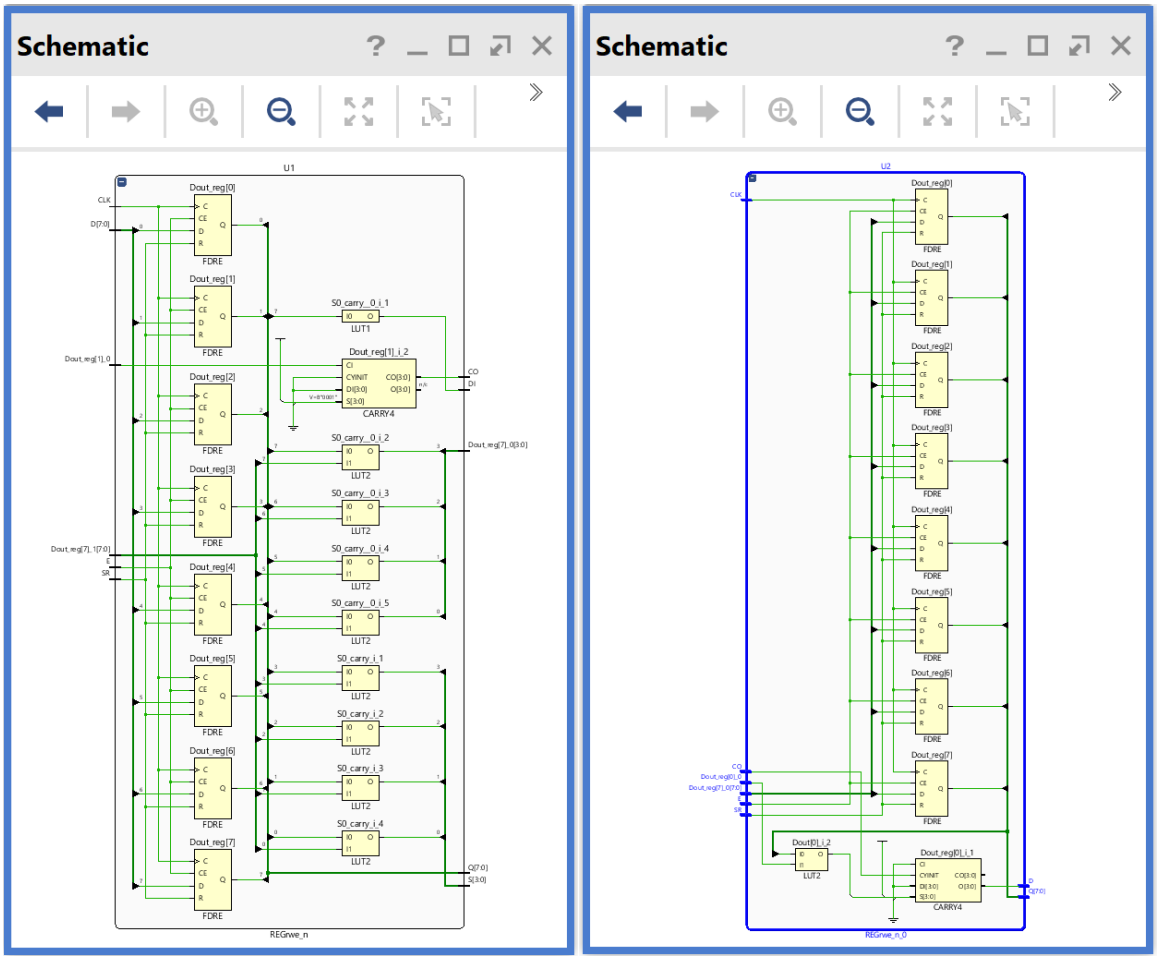
Υπομονάδα U2 (REGrwe_n_0): Συμπεριλαμβάνει τον καταχωρητή εισόδου B των 8 bit, 1 πύλη XOR (LUT2) και μία μονάδα CARRY4.

Υπομονάδα U3 (ADDER_n): Συμπεριλαμβάνει 1 πύλη XOR (LUT2) και 2 μονάδες CARRY4.

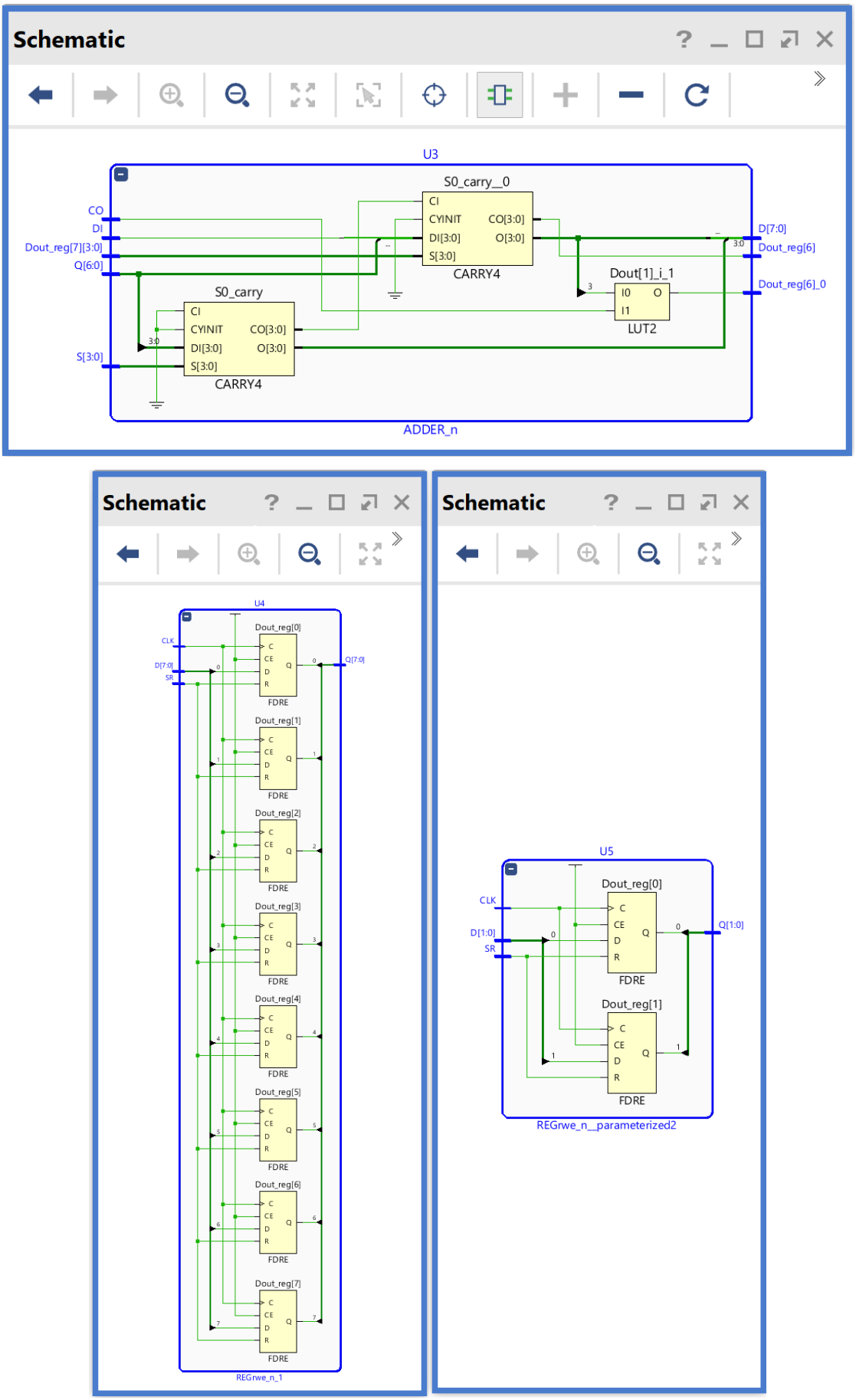
Υπομονάδα U4 (REGrwe_n_1): Συμπεριλαμβάνει τον καταχωρητή εξόδου S των 8 bit.

Υπομονάδα U5 (REGrwe_n_parameterized2): Συμπεριλαμβάνει τον καταχωρητή εξόδου των 2 bit για τις σημαίες Cout και OV.

Προσοχή! Οι υπομονάδες **U1**, **U2** και **U3** είναι διαφορετικές από πλευράς λογικής στο **synthesized (implemented) design model**, που προκύπτει μετά τη σύνθεση (υλοποίηση), σε σχέση με το **elaborated design model**. Αντίθετα, οι υπομονάδες **U4** και **U5** παραμένουν ίδιες από πλευράς λογικής.



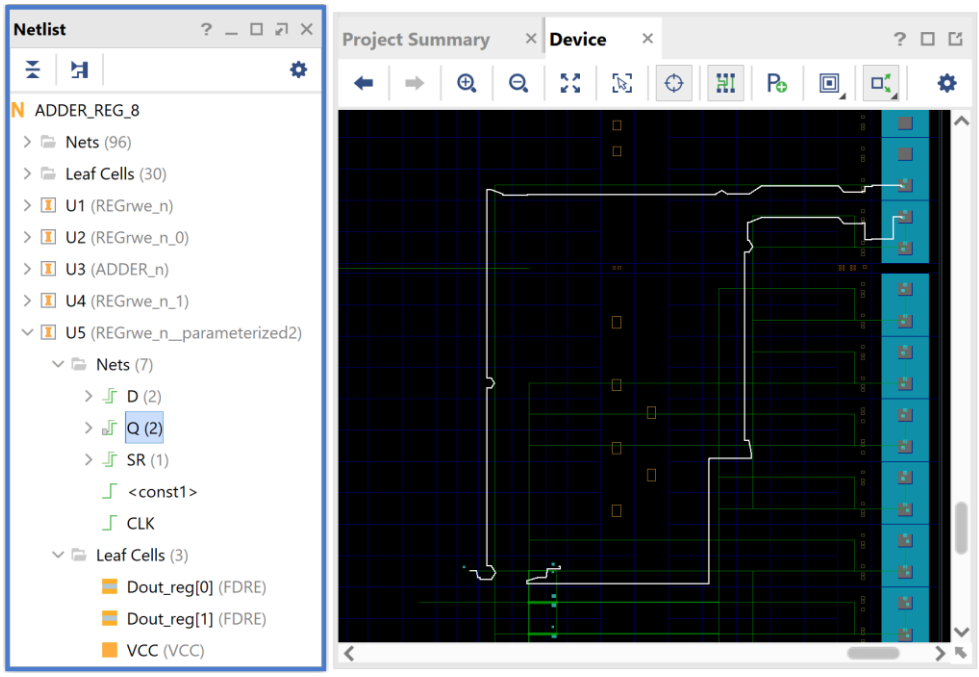
Εικόνα 136 – Σχηματικά διαγράμματα επί μέρους μονάδων του implemented design model (#1)



Εικόνα 137 – Σχηματικά διαγράμματα επί μέρους μονάδων του implemented design model (#2)

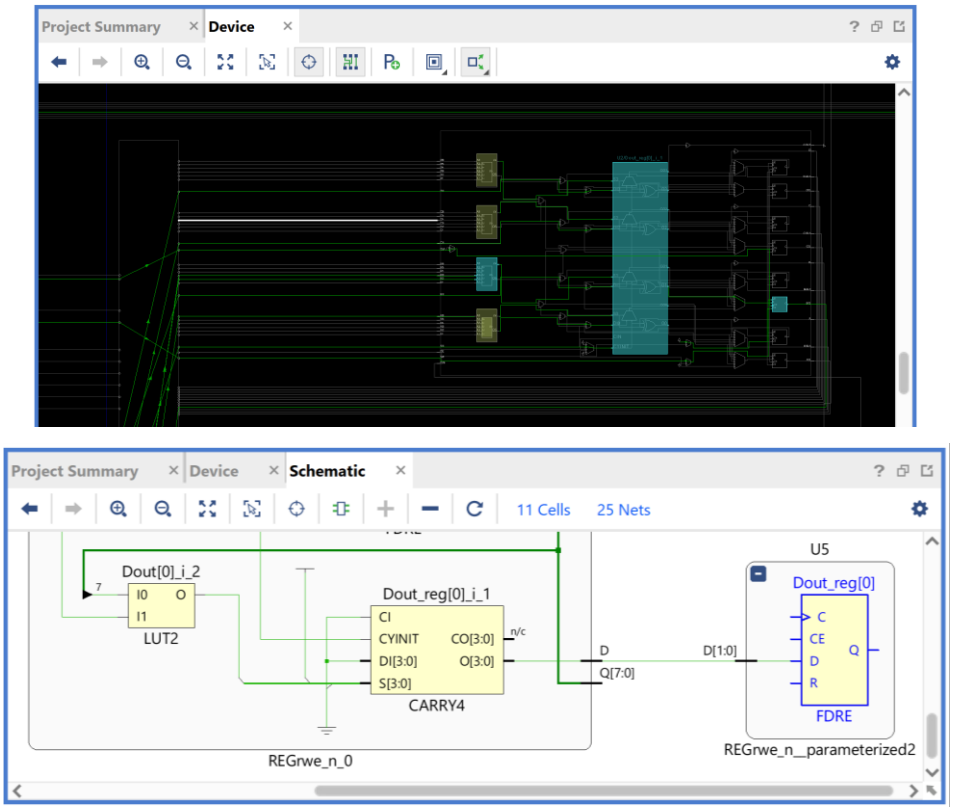
- Στη συνέχεια μελετήστε το παράθυρο *Device* έχοντας πατήσει τα κουμπιά *auto–fit selection*, *routing resources* και *show cell connections*.

  Στη μελέτη σας επιλέξτε κάποιο *net*, όπως για παράδειγμα το **Q[1:0]** (OV, Cout) της υπομονάδας U5 με κατάλληλη επιλογή στο παράθυρο *Netlist*.



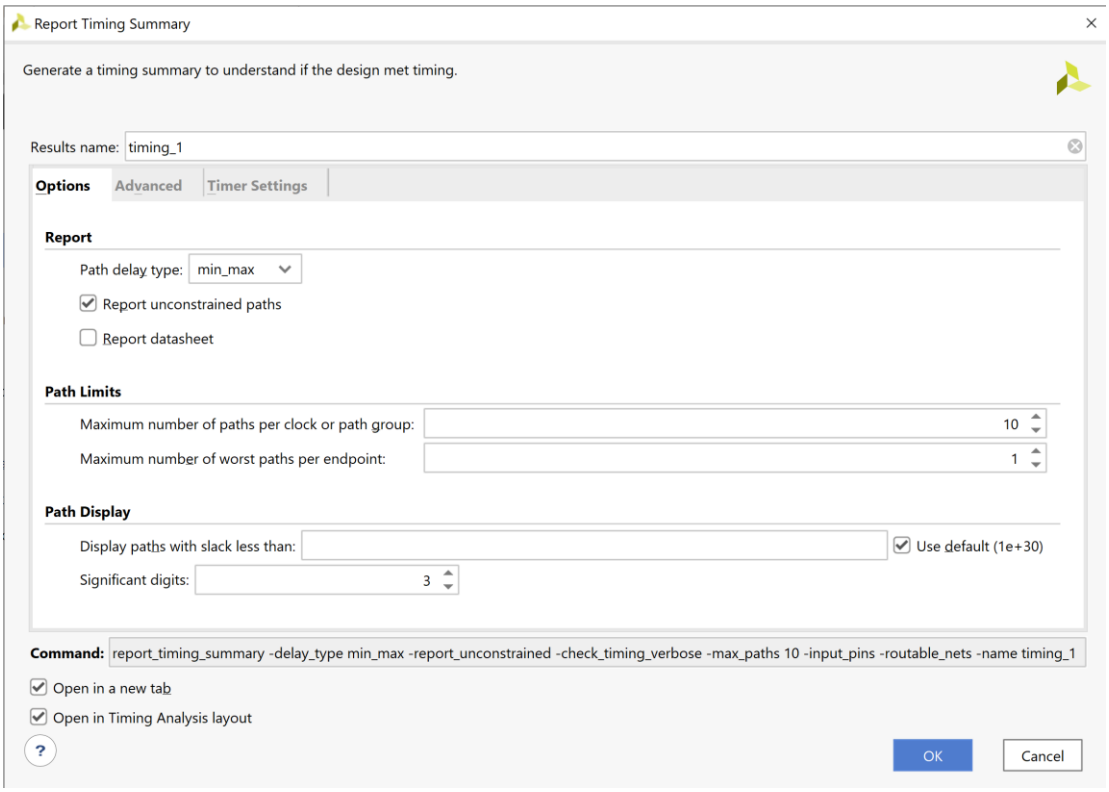
Εικόνα 138 – Παράδειγμα μελέτης net από το παράθυρο Netlist

- Επικεντρώστε στη δημιουργία του Q[0] (Cout) με παράλληλη μελέτη των παραθύρων *Device* και *Schematic*, εναλλάξ. Απαιτείται εντοπισμένη μεγέθυνση. Φαίνονται: το LUT2 (U2), το CARRY4 (U2) και το FDRE (U5) στο SLICE_X113Y15!



Εικόνα 139 – Μελέτη δημιουργίας σήματος Q[0] (Cout)

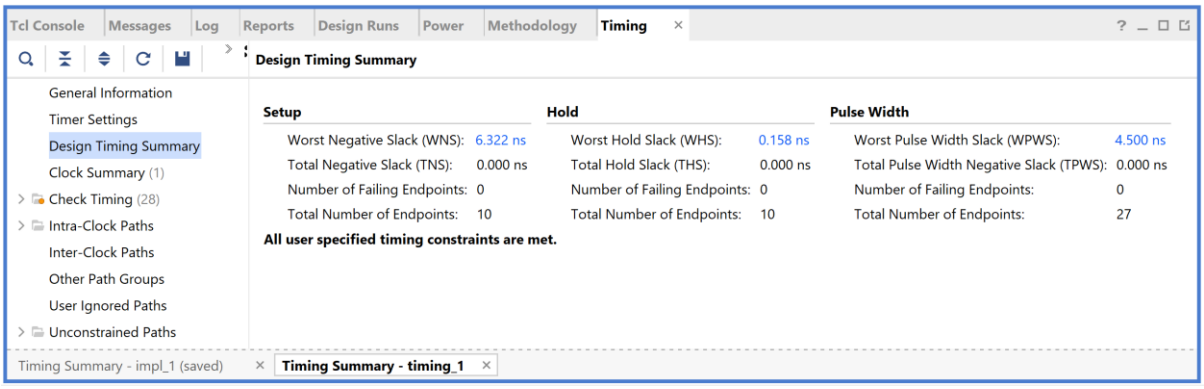
- Στη συνέχεια, στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, μέσα στο *Open Implemented Design* επιλέξετε το **Report Timing Summary** για να δείτε την ανάλυση χρονισμού που κάνει το εργαλείο Vivado IDE στο **implemented design model**. Αυτή είναι η πιο ακριβής ανάλυση χρονισμού που έχουμε διαθέσιμη.



Εικόνα 140 – Επιλογές για την παραγωγή του timing summary για το implemented design model

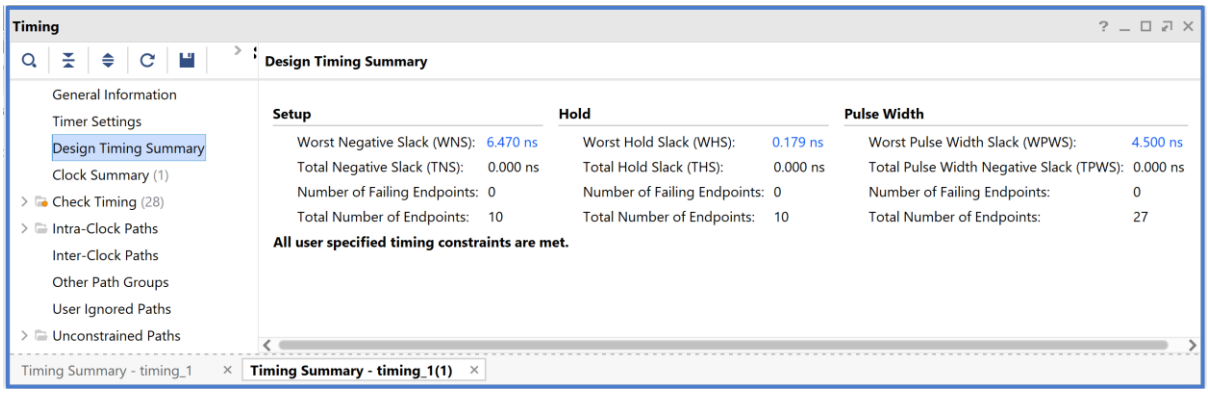
Στην επιλογή *Path delay type* ορίζεται ο τύπος της ανάλυσης που θα εκτελεσθεί. Το *max delay analysis* αφορά στην εύρεση της κρίσιμης διαδρομής (με τη μεγαλύτερη καθυστέρηση διάδοσης) και συνεπώς στην εύρεση της μέγιστης συχνότητας λειτουργίας χωρίς την παραβίαση του χρόνου σταθεροποίησης (setup time). Το *min delay analysis* αφορά στην εύρεση της σύντομης διαδρομής (με τη μικρότερη καθυστέρηση διάδοσης) και συνεπώς στην πιθανή παραβίαση του χρόνου διατήρησης (hold time).

- Πατήστε **OK** για να παραχθεί το Timing_1 report.



Εικόνα 141 – Timing report για το implemented design model

Συγκρίνετε με το Timing_1 report που παράχθηκε μετά τη σύνθεση.



Εικόνα 142 – Timing report για το synthesized design model

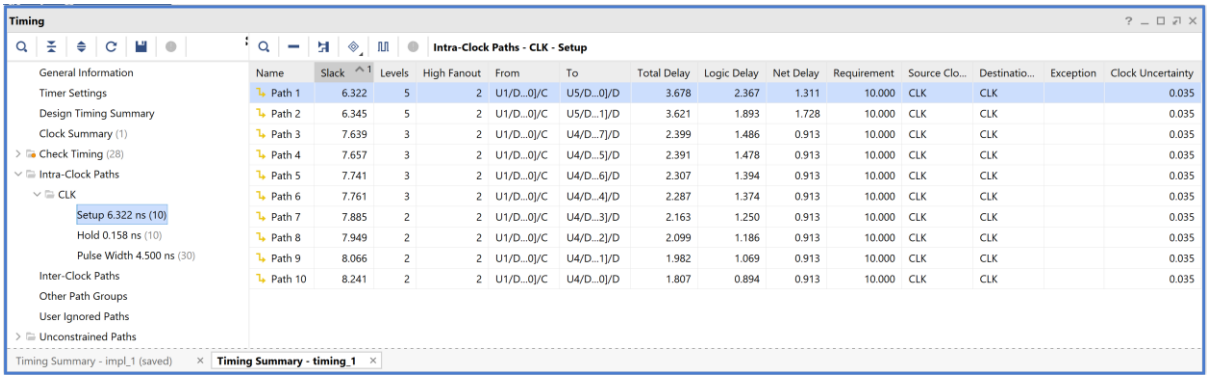
Στη στήλη **Setup** παρουσιάζονται τα αποτελέσματα του *max delay analysis*.

- Το *Worst Negative Slack* (**WNS**) είναι μια τιμή (με link) που αντιστοιχεί στο μικρότερο διαθέσιμο περιθώριο (slack) που προκύπτει από την ανάλυση όλων των κρίσιμων διαδρομών του max delay analysis. Μπορεί να είναι θετικό η αρνητικό. Ένα θετικό slack (6.322 ns) δηλώνει ότι η κρίσιμη διαδρομή ικανοποιεί την περίοδο του CLK. Ένα αρνητικό slack δηλώνει ότι παραβιάζεται ο χρόνος σταθεροποίησης (setup time).
- Το *Total Negative Slack* (**TNS**) είναι το άθροισμα όλων των αρνητικών WNS για κάθε timing path endpoint. Η τιμή είναι 0.000 ns όταν δεν υπάρχουν παραβιάσεις του χρόνου σταθεροποίησης. Το ψηφιακό κύκλωμα λειτουργεί κανονικά στην επιλεγμένη συχνότητα λειτουργίας.

Στη στήλη **Hold** παρουσιάζονται τα αποτελέσματα του min delay analysis.

- Το *Worst Hold Slack* (**WHS**) είναι μια τιμή (με link) που αντιστοιχεί στο μικρότερο διαθέσιμο περιθώριο (slack) που προκύπτει από την ανάλυση όλων των σύντομων διαδρομών του min delay analysis. Μπορεί να είναι θετικό η αρνητικό. Ένα αρνητικό slack δηλώνει πόσο παραβιάζεται ο χρόνος διατήρησης (hold time).
- Το *Total Hold Slack* (**THS**) είναι το άθροισμα όλων των αρνητικών WHS για κάθε timing path endpoint. Η τιμή είναι 0.000 ns όταν δεν υπάρχουν παραβιάσεις του χρόνου διατήρησης. Το ψηφιακό κύκλωμα λειτουργεί κανονικά.
- Επιλέξτε το **WNS link** και δείτε τις 10 χειρότερες κρίσιμες διαδρομές (δηλαδή με το μικρότερο θετικό slack) που δεν παραβιάζουν το χρόνο σταθεροποίησης. Η κρίσιμη διαδρομή έχει καθυστέρηση διάδοσης 3.678 ns, εκ των οποίων τα 2.367 ns

αφορούν στη λογική (logic), ενώ τα 1.311 ns αφορούν στη δικτύωση (net). Η αβεβαιότητα του CLK εκτιμάται στα 0.035 ns. Τα επίπεδα λογικής (logic level) είναι 5.

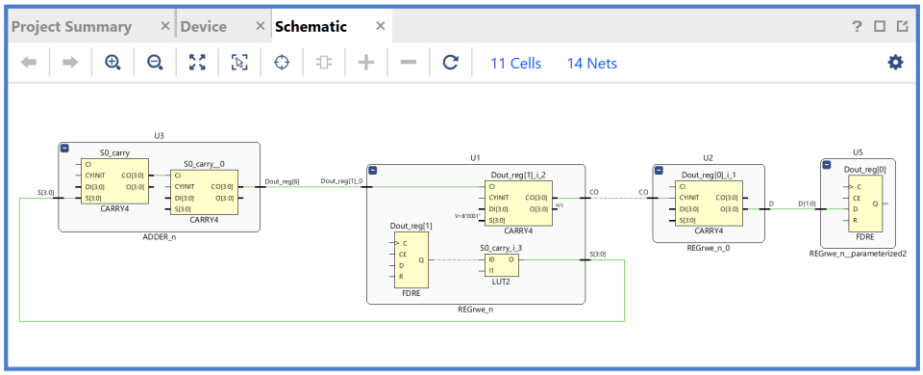


Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clo...	Destinatio...	Exception	Clock Uncertainty
Path 1	6.322	5	2	U1/D...0]/C	U5/D...0]/D	3.678	2.367	1.311	10.000	CLK	CLK		0.035
Path 2	6.345	5	2	U1/D...0]/C	U5/D...1]/D	3.621	1.893	1.728	10.000	CLK	CLK		0.035
Path 3	7.639	3	2	U1/D...0]/C	U4/D...7]/D	2.399	1.486	0.913	10.000	CLK	CLK		0.035
Path 4	7.657	3	2	U1/D...0]/C	U4/D...5]/D	2.391	1.478	0.913	10.000	CLK	CLK		0.035
Path 5	7.741	3	2	U1/D...0]/C	U4/D...6]/D	2.307	1.394	0.913	10.000	CLK	CLK		0.035
Path 6	7.761	3	2	U1/D...0]/C	U4/D...4]/D	2.287	1.374	0.913	10.000	CLK	CLK		0.035
Path 7	7.885	2	2	U1/D...0]/C	U4/D...3]/D	2.163	1.250	0.913	10.000	CLK	CLK		0.035
Path 8	7.949	2	2	U1/D...0]/C	U4/D...2]/D	2.099	1.186	0.913	10.000	CLK	CLK		0.035
Path 9	8.066	2	2	U1/D...0]/C	U4/D...1]/D	1.982	1.069	0.913	10.000	CLK	CLK		0.035
Path 10	8.241	2	2	U1/D...0]/C	U4/D...0]/D	1.807	0.894	0.913	10.000	CLK	CLK		0.035

Εικόνα 143 – Μελέτη κρίσιμου μονοπατιού του implemented design model

- Επιλέξτε με διπλό κλικ το **Path 1**, ώστε να εμφανιστεί το παράθυρο *Path 1 – timing_1*. Η κρίσιμη διαδρομή περνάει από 4 μονάδες CARRY4 και 1 LUT2 (πύλη XOR). Υπολογίστε το WNS slack (σε παρένθεση τα αποτελέσματα της σύνθεσης):
- η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Source Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK του καταχωρητή της υπομονάδας U1) είναι **5.637** (2.975) ns,
- η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Data Path* (από την είσοδο CLK μέχρι την έξοδο Q του καταχωρητή της υπομονάδας U1 και μέσω του LUT2 (U1) και των 4 μονάδων CARRY4 (U3–U1–U2) μέχρι την είσοδο D του καταχωρητή της υπομονάδας U5) είναι **3.678** (3.426) ns,
- το **Arrival Time**, ως άθροισμα των ανωτέρων χρόνων, είναι **9.315** (6.401) ns,
- το **Required Time**, ως η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Destination Clock Path* (από την πηγή του CLK τη χρονική στιγμή **10.000** ns μέχρι την είσοδο CLK του καταχωρητή της υπομονάδας U5) συν τον χρόνο σταθεροποίησης είναι **15.637** (12.871) ns.

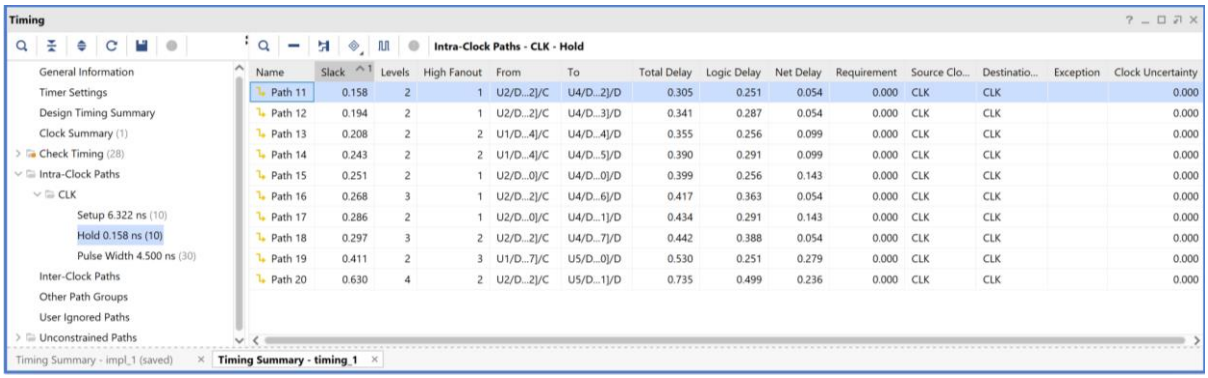
  WNS slack = Required Time – Arrival Time = 15.637 – 9.315 = 6.322 ns
- Επιλέξτε με δεξί κλικ στο **Path 1**, το **Schematic**. Μελετήστε το σχηματικό διάγραμμα της κρίσιμης διαδρομής της οντότητας **ADDER_REG_8**. (Δεν έχει αλλάξει η διαδρομή).



Εικόνα 144 – Σχηματικό διάγραμμα κρίσιμου μονοπατιού του implemented design model

- Επιλέξτε το **WHS link** και δείτε τις 10 χειρότερες σύντομες διαδρομές (δηλαδή με το μικρότερο θετικό slack) που δεν παραβιάζουν το χρόνο διατήρησης. Η σύντομη

διαδρομή έχει καθυστέρηση μόλυνσης 0.305 ns, εκ των οποίων τα 0.251 ns αφορούν στη λογική (logic), ενώ τα 0.054 ns αφορούν στη δικτύωση (net). Η αβεβαιότητα του CLK εκτιμάται στα 0.000 ns. Τα επίπεδα λογικής (logic level) είναι 2 (ήταν 1 μετά τη σύνθεση).



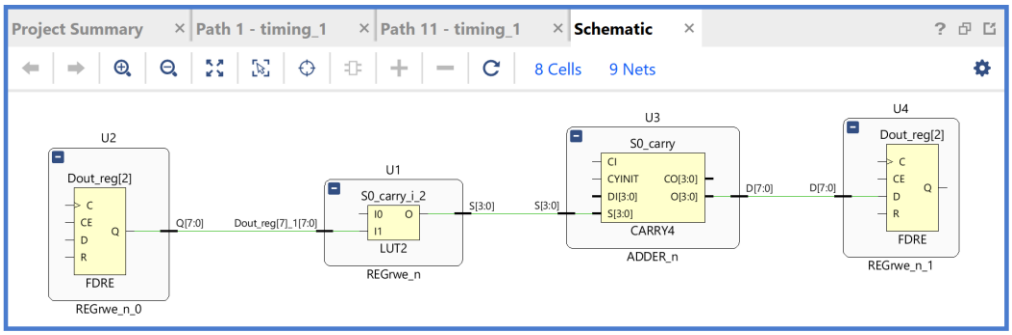
Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clo...	Destinatio...	Exception	Clock Uncertainty
Path 11	0.158	2	1	U2/D...2]/C	U4/D...2]/D	0.305	0.251	0.054	0.000	CLK	CLK		0.000
Path 12	0.194	2	1	U2/D...2]/C	U4/D...3]/D	0.341	0.287	0.054	0.000	CLK	CLK		0.000
Path 13	0.208	2	2	U1/D...4]/C	U4/D...4]/D	0.355	0.256	0.099	0.000	CLK	CLK		0.000
Path 14	0.243	2	2	U1/D...4]/C	U4/D...5]/D	0.390	0.291	0.099	0.000	CLK	CLK		0.000
Path 15	0.251	2	1	U2/D...0]/C	U4/D...0]/D	0.399	0.256	0.143	0.000	CLK	CLK		0.000
Path 16	0.268	3	1	U2/D...2]/C	U4/D...6]/D	0.417	0.363	0.054	0.000	CLK	CLK		0.000
Path 17	0.286	2	1	U2/D...0]/C	U4/D...1]/D	0.434	0.291	0.143	0.000	CLK	CLK		0.000
Path 18	0.297	3	2	U2/D...2]/C	U4/D...7]/D	0.442	0.388	0.054	0.000	CLK	CLK		0.000
Path 19	0.411	2	3	U1/D...7]/C	U5/D...0]/D	0.530	0.251	0.279	0.000	CLK	CLK		0.000
Path 20	0.630	4	2	U2/D...2]/C	U5/D...1]/D	0.735	0.499	0.236	0.000	CLK	CLK		0.000

Εικόνα 145 – Μελέτη χειρότερης σύντομης διαδρομής του implemented design model

- Επιλέξτε με διπλό κλικ το **Path 11**, ώστε να εμφανιστεί το παράθυρο *Path 11 – timing_1*. Η σύντομη διαδρομή περνάει από 1 μονάδα CARRY4 και 1 LUT2. Υπολογίστε το WHS slack (σε παρένθεση τα αποτελέσματα της σύνθεσης):
- η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Source Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK του καταχωρητή της υπομονάδας U2) είναι **1.583** (0.735) ns,
- η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Data Path* (από την είσοδο CLK μέχρι την έξοδο Q του καταχωρητή U2 και μέσω του LUT2 (U1) και 1 μονάδας CARRY4 (U3) μέχρι την είσοδο D του καταχωρητή U4) είναι **0.305** (0.438) ns,
- το **Arrival Time**, ως άθροισμα των ανωτέρων χρόνων, είναι **1.888** (1.173) ns,
- το **Required Time**, ως η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Destination Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK του καταχωρητή της υπομονάδας U4) συν τον χρόνο διατήρησης, είναι **1.730** (0.993) ns.

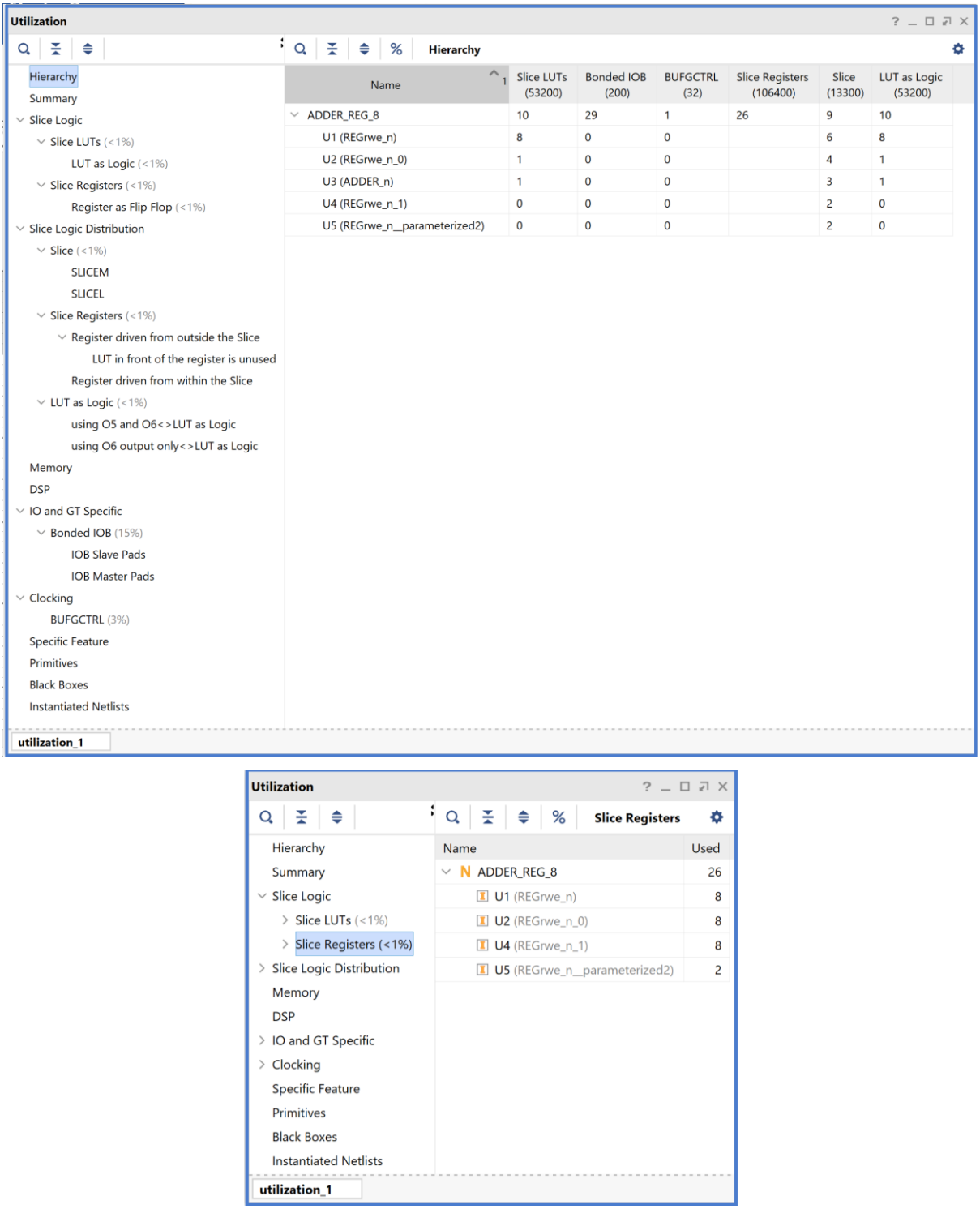
  WHS slack = Arrival Time – Required Time = 1.888 – 1.730 = 0.158 ns

- Επιλέξτε με δεξί κλικ στο **Path 11**, το **Schematic**. Μελετήστε το σχηματικό διάγραμμα της σύντομης διαδρομής της οντότητας **ADDER_REG_8**. (Έχει αλλάξει η διαδρομή).



Εικόνα 146 – Σχηματικό διάγραμμα χειρότερης σύντομης διαδρομής του impl. design model

- Τέλος, στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, μέσα στο *Open Implemented Design* επιλέξετε το **Report Utilization** και μελετήστε τους πόρους που χρησιμοποιεί η οντότητα **ADDER_REG_8**. Πατήστε **OK**. (Μπορείτε να μελετήσετε και άλλα ενδιαφέροντα reports, εάν επιλέξετε το παράθυρο *Reports*).



Εικόνα 147 – Μελέτη των πόρων της οντότητας ADDER_REG_8

1.10.2 Εύρεση συχνότητας λειτουργίας και ρύθμιση του σήματος CLK στο VIVADO IDE για συνδυαστική λογική ανάμεσα σε καταχωρητές

Όταν η διαδικασία bottom–up σχεδίασης και υλοποίησης ενός ψηφιακού συστήματος ολοκληρωθεί (η σχεδίαση έχει πλέον ωριμάσει στο επίπεδο της αποσφαλμάτωσης και της βελτιστοποίησης) και έχουμε φθάσει στη σχεδίαση και υλοποίηση της κορυφαίας οντότητας της ιεραρχίας του ψηφιακού συστήματός μας, προχωράμε στη διαδικασία της εύρεσης της μέγιστης συχνότητας λειτουργίας και της ρύθμισης του σήματος **CLK**. Αυτή η διαδικασία εφαρμόζεται στο **implemented design model** της οντότητας που έχει ορισθεί ως η κορυφαία οντότητα της ιεραρχίας (**top**) των design resources.

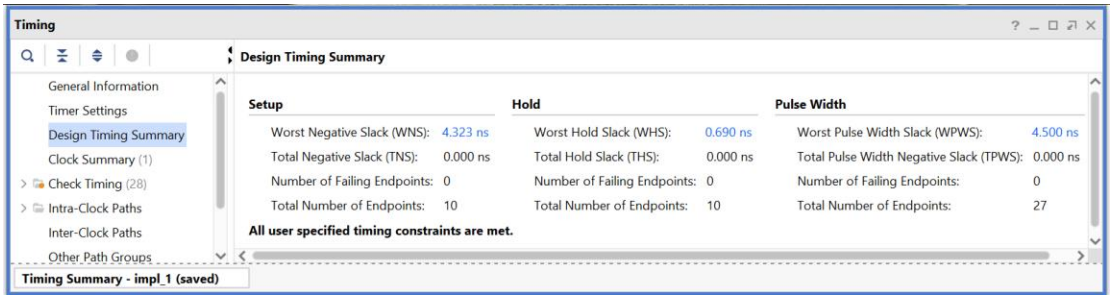
Η συχνότητα λειτουργίας εξαρτάται από την τεχνολογία FPGA (π.χ. οικογένεια, speed grade), το περιβάλλον υλοποίησης (π.χ. θερμοκρασία, κατανάλωση ισχύος, ροή αέρα) και τους περιορισμούς που τίθενται σε επίπεδο board για τη διατήρηση του σήματος.

Για την ορθή εμφάνιση των εξόδων στο χρονικό διάγραμμα της χρονικής προσομοίωσης του **implemented design model** επιλέγουμε να τοποθετήσουμε τους **καταχωρητές εξόδου στα IOB** του FPGA.

- Αρχικά, επιλέξτε να τοποθετήσετε τους καταχωρητές εξόδου στα **IOB** του FPGA. Στο παράθυρο *Sources*, επιλέξτε το constraint αρχείο **zedboard.xdc** και ανοίξτε το. Στο τέλος του αρχείου προσθέστε τον περιορισμό που υποχρεώνει το εργαλείο Vivado IDE να θέσει τους καταχωρητές εξόδου στα **IOB** του FPGA και πατήστε **Save**.

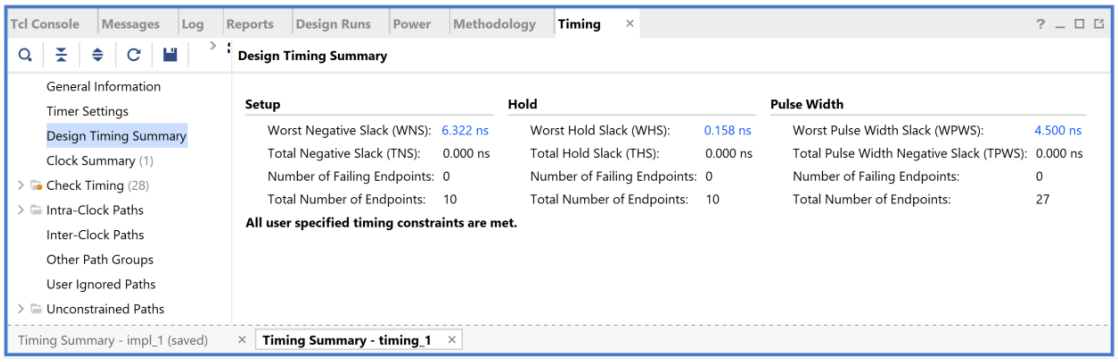
```
44 # Set output registers to IOB
45 set_property IOB TRUE [all_outputs]
```

- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Open Implemented Design** και στη συνέχεια επιλέξτε το **Run Implementation**. Επαναλάβετε τη διαδικασία της σύνθεσης και της υλοποίησης πατώντας **Yes** και **OK**.
- Στο παράθυρο *Implementation Completed* κρατήστε την επιλογή **Open Implemented Design** και πατήστε **OK**. Στο παράθυρο *Timing* βλέπετε την ανάλυση χρονισμού που κάνει το εργαλείο Vivado IDE στο **implemented design model** με τη χρήση των **IOB** του FPGA για περίοδο **CLK** στα **10.000** ns.



Εικόνα 148 – Ανάλυση χρονισμού του implemented design model με χρήση IOB και περίοδο 10 ns

Συγκρίνετε με το timing_1 report μετά την υλοποίηση, χωρίς τη χρήση των **IOB**, με περίοδο **CLK** στα **10.000** ns.



Εικόνα 149 – Ανάλυση χρονισμού μετά την υλοποίηση χωρίς τη χρήση IOB με περίοδο 10 ns

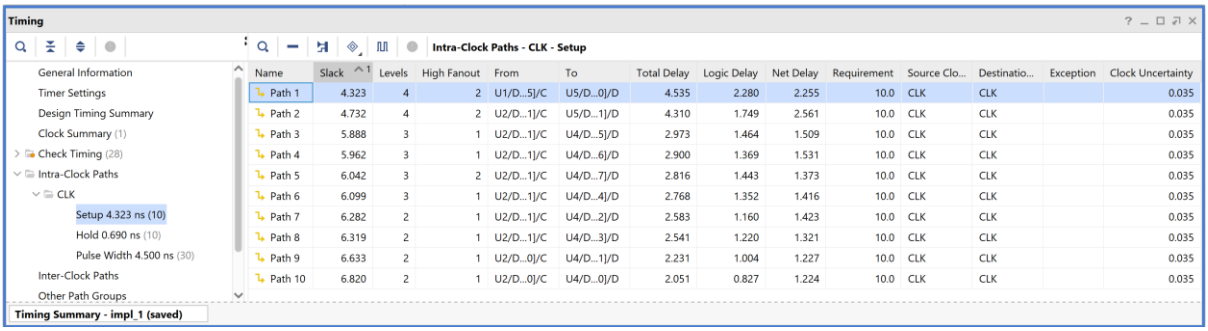
Στη στήλη **Setup** παρουσιάζονται τα αποτελέσματα του *max delay analysis*.

- Το *Worst Negative Slack* (**WNS**) είναι μια τιμή (με link) που αντιστοιχεί στο μικρότερο διαθέσιμο περιθώριο (slack) που προκύπτει από την ανάλυση όλων των κρίσιμων διαδρομών του max delay analysis. Μπορεί να είναι θετικό η αρνητικό. Ένα θετικό slack (4.323 ns) δηλώνει ότι η κρίσιμη διαδρομή ικανοποιεί την περίοδο του CLK. Η τοποθέτηση των καταχωρητών εξόδου στα **IOB** του FPGA **μειώνουν** το **WNS** από **6.322** σε **4.323**!
- Το *Total Negative Slack* (**TNS**) είναι το άθροισμα όλων των αρνητικών WNS για κάθε timing path endpoint. Η τιμή είναι 0.000 ns όταν δεν υπάρχουν παραβιάσεις του χρόνου σταθεροποίησης. Το ψηφιακό κύκλωμα λειτουργεί κανονικά στην επιλεγμένη συχνότητα λειτουργίας.

Στη στήλη **Hold** παρουσιάζονται τα αποτελέσματα του min delay analysis.

- Το *Worst Hold Slack* (**WHS**) είναι μια τιμή (με link) που αντιστοιχεί στο μικρότερο διαθέσιμο περιθώριο (slack) που προκύπτει από την ανάλυση όλων των σύντομων διαδρομών του min delay analysis. Μπορεί να είναι θετικό η αρνητικό. Ένα θετικό slack δηλώνει ότι δεν παραβιάζεται ο χρόνος διατήρησης (hold time). Η τοποθέτηση των καταχωρητών εξόδου στα **IOB** του FPGA **αυξάνουν** το **WHS** από **0.158** σε **0.690**!
- Το *Total Hold Slack* (**THS**) είναι το άθροισμα όλων των αρνητικών WHS για κάθε timing path endpoint. Η τιμή είναι 0.000 ns όταν δεν υπάρχουν παραβιάσεις του χρόνου διατήρησης. Το ψηφιακό κύκλωμα λειτουργεί κανονικά.

- Επιλέξτε το **WNS link** και δείτε τις 10 χειρότερες κρίσιμες διαδρομές (δηλαδή με το μικρότερο θετικό slack) που δεν παραβιάζουν το χρόνο σταθεροποίησης. Η κρίσιμη διαδρομή έχει καθυστέρηση διάδοσης 4.535 ns, εκ των οποίων τα 2.280 ns αφορούν στη λογική (logic), ενώ τα 2.255 ns αφορούν στη δικτύωση (net). Η αβεβαιότητα του CLK εκτιμάται στα 0.035 ns. Τα επίπεδα λογικής (logic level) είναι πλέον 4 (δεν συμπεριλαμβάνεται μία μονάδα CARRY4).

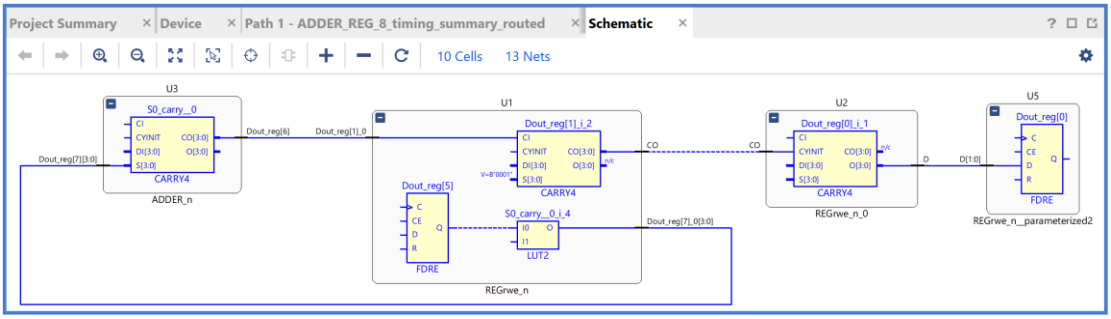


Εικόνα 150 – Μελέτη κρίσιμης διαδρομής με χρήση IOB

- Επιλέξτε με διπλό κλικ το **Path 1**, ώστε να εμφανιστεί το παράθυρο *Path 1 – ADDER_REG_8_timing_summary_routed*. Η κρίσιμη διαδρομή περνάει από 3 μονάδες CARRY4 και 1 LUT2. Υπολογίστε το WNS slack (σε παρένθεση τα αποτελέσματα της υλοποίησης, χωρίς τη χρήση IOB):
- η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Source Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK του καταχωρητή της υπομονάδας U1) είναι **5.632** (5.637) ns,
- η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Data Path* (από την είσοδο CLK μέχρι την έξοδο Q του καταχωρητή της υπομονάδας U1, του LUT2 (U1) και των 3 μονάδων CARRY4 (U3–U1–U2) μέχρι την είσοδο D του καταχωρητή της υπομονάδας U5) είναι **4.535** (3.678) ns,
- το **Arrival Time**, ως άθροισμα των ανωτέρων χρόνων, είναι **10.167** (9.315) ns,
- το **Required Time**, ως η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Destination Clock Path* (από την πηγή του CLK τη χρονική στιγμή **10.000** ns μέχρι την είσοδο CLK του καταχωρητή της υπομονάδας U5) συν τον χρόνο σταθεροποίησης (–1.016) είναι **14.490** (15.637) ns.

  WNS slack = Required Time – Arrival Time = 14.490 – 10.167 = 4.323 ns

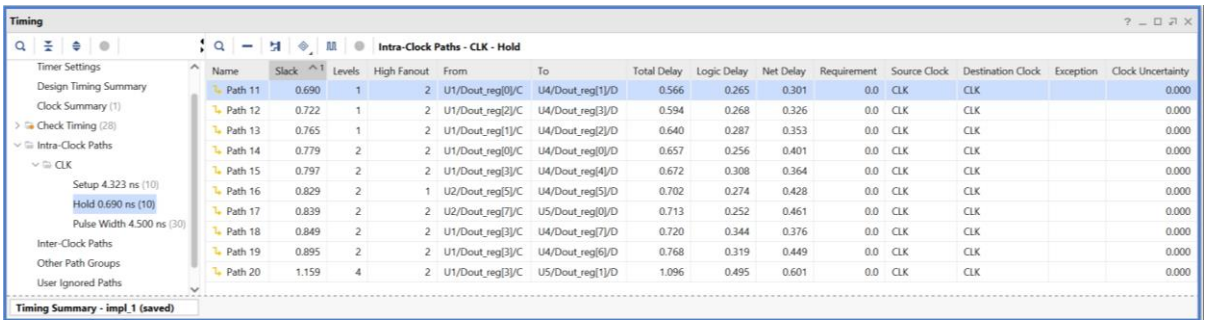
- Επιλέξτε με δεξί κλικ στο **Path 1**, το **Schematic**. Μελετήστε το σχηματικό διάγραμμα της κρίσιμης διαδρομής της οντότητας **ADDER_REG_8**. (Έχει αλλάξει η διαδρομή).



Εικόνα 151 – Σχηματικό διάγραμμα κρίσιμης διαδρομής με χρήση IOB

- Επιλέξτε το **WHS link** και δείτε τις 10 χειρότερες σύντομες διαδρομές (δηλαδή με το μικρότερο θετικό slack) που δεν παραβιάζουν το χρόνο διατήρησης. Η σύντομη

διαδρομή έχει καθυστέρηση μόλυνσης 0.566 ns, εκ των οποίων τα 0.265 ns αφορούν στη λογική (logic), ενώ τα 0.301 ns αφορούν στη δικτύωση (net). Η αβεβαιότητα του CLK εκτιμάται στα 0.000 ns. Τα επίπεδα λογικής (logic level) είναι 1 (ήταν 2 μετά την υλοποίηση, χωρίς τη χρήση IOB).

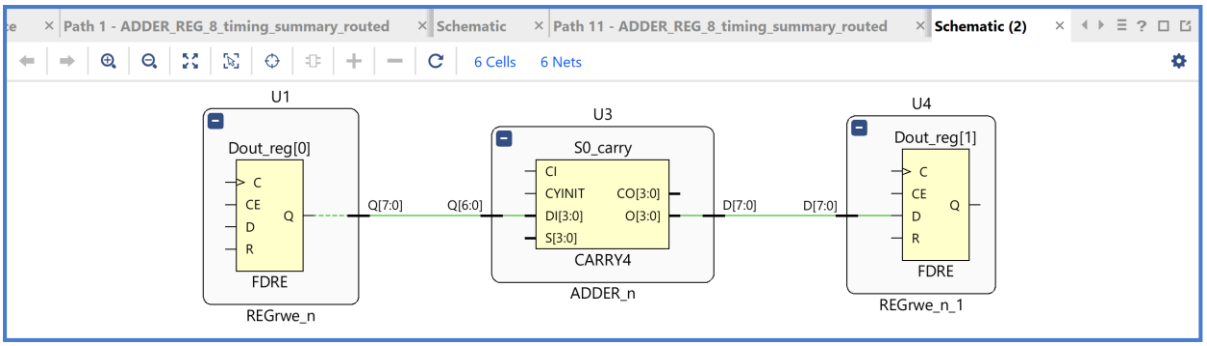


Εικόνα 152 – Μελέτη χειρότερης σύντομης διαδρομής με χρήση IOB

- Επιλέξτε με διπλό κλικ το **Path 11**, ώστε να εμφανιστεί το παράθυρο *Path 11 – ADDER_REG_8_timing_summary_routed*. Η σύντομη διαδρομή περνάει από 1 μονάδα CARRY4 της υπομονάδας U3. Υπολογίστε το WHS slack (σε παρένθεση τα αποτελέσματα της υλοποίησης, χωρίς τη χρήση IOB):
- η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Source Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK του καταχωρητή της υπομονάδας U1) είναι **1.580** (1.583) ns,
- η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Data Path* (από την είσοδο CLK μέχρι την έξοδο Q του καταχωρητή της υπομονάδας U2 και μέσω του LUT2 (U1) και της 1 μονάδας CARRY4 (U3) μέχρι την είσοδο D του καταχωρητή της υπομονάδας U4) είναι **0.566** (0.305) ns,
- το **Arrival Time**, ως άθροισμα των ανωτέρων χρόνων, είναι **2.146** (1.888) ns,
- το **Required Time**, ως η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Destination Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK του καταχωρητή της υπομονάδας U4) συν τον χρόνο διατήρησης (–0.155), είναι **1.456** (1.730) ns.

  WHS slack = Arrival Time – Required Time = 2.146 – 1.456 = 0.690 ns

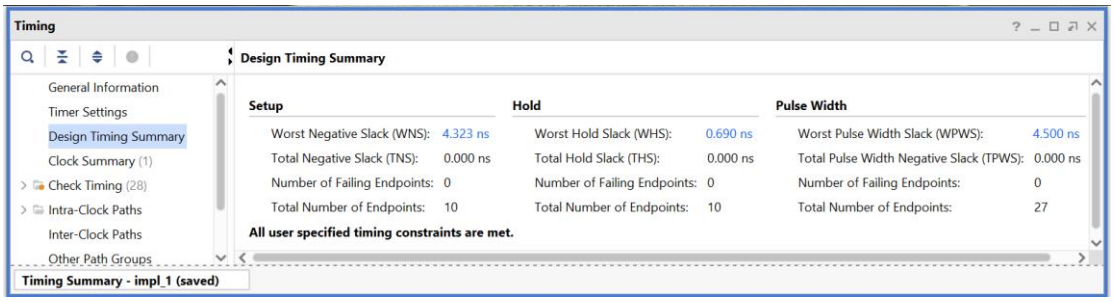
- Επιλέξτε με δεξί κλικ στο **Path 11**, το **Schematic**. Μελετήστε το σχηματικό διάγραμμα της σύντομης διαδρομής της οντότητας **ADDER_REG_8**. (Έχει αλλάξει διαδρομή).



Εικόνα 153 – Σχηματικό διάγραμμα χειρότερης σύντομης διαδρομής με χρήση IOB

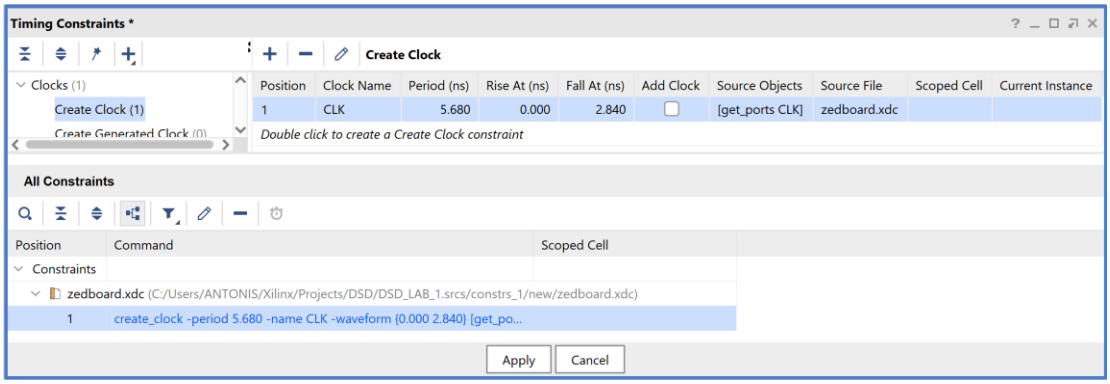
- Στη συνέχεια, με βάση το **Design Timing Summary** του παράθυρου *timing* ορίστε τη νέα περίοδο του σήματος **CLK**:

New_CLK_period = CLK_period – WNS = 10.000 – 4.320 = 5.680 ns



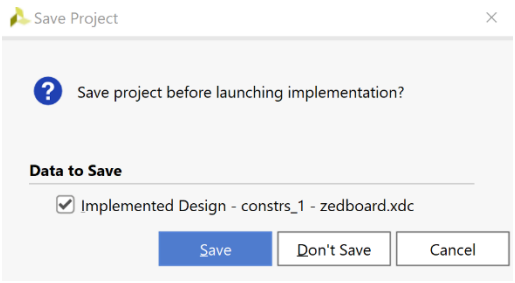
Εικόνα 154 – Design timing summary με χρήση IOB

- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Open Implemented Design** και στη συνέχεια επιλέξτε το **Edit Timing Constraints**, ώστε να δημιουργήσετε ένα νέο clock constraint. Στο παράθυρο *Timing Constraints* αλλάξτε με διπλό κλικ το **Period** από 10.000 ns σε **5.680 ns** (10.000 – 4.320 ns) και το **Fall At** από 5.000 ns σε **2.840 ns**. Πατήστε **Apply**.



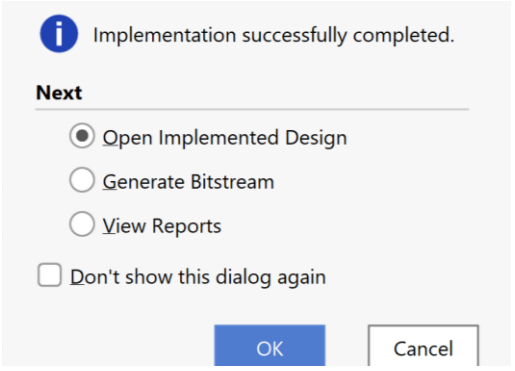
Εικόνα 155 – Δημιουργία νέου clock constraint

- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Run Implementation**. Στο παράθυρο *Save Project* επιλέξτε **Save**, για να αποθηκευτεί η νέα περίοδος του σήματος **CLK**. Στη συνέχεια πατήστε **OK** και **Update**. Επαναλάβετε τη διαδικασία της σύνθεσης και της υλοποίησης πατώντας **Yes** και **OK**.

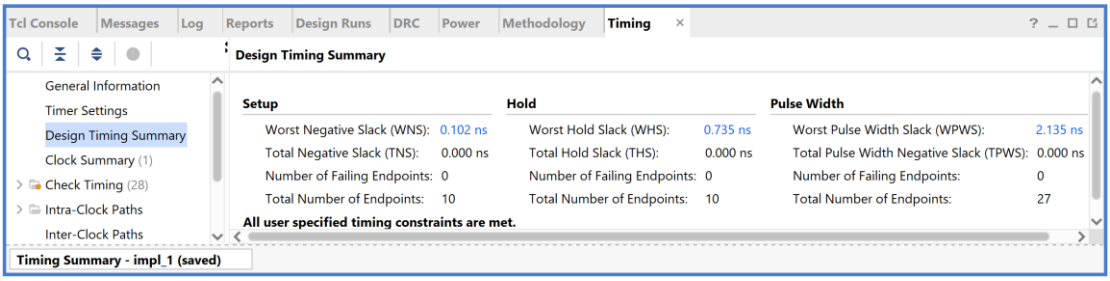


Εικόνα 156 - Αποθήκευση αλλαγών στο project

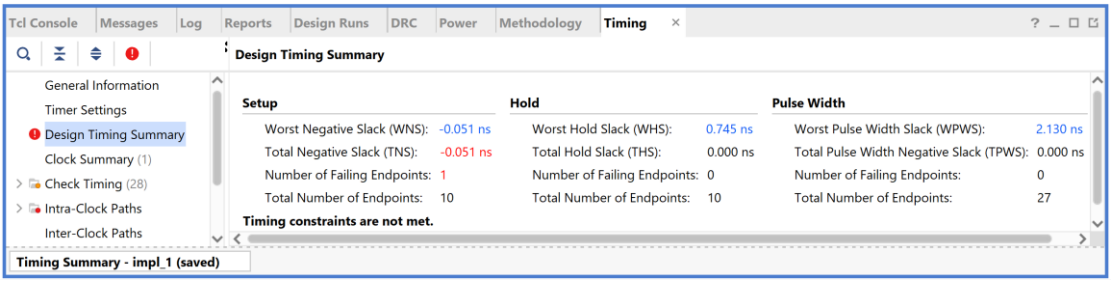
- Στο παράθυρο *Implementation Completed*, διατηρήστε την επιλογή **Open Implemented Design** και πατήστε **OK**.
- Επαναλάβετε τα βήματα **5–2.11–14** μέχρι να προκύψει ένα αρνητικό WNS. Επιλέξτε, τη μικρότερη περίοδο με θετικό WNS. Δοκιμάστε τις περιόδους: **5.680 ns** (WNS = 0.287), **5.400 ns** (WNS = 0.135), **5.270 ns** (WNS = 0.102), **5.260 ns** (WNS = –0.051).



Εικόνα 157 - Επιλογές μετά την υλοποίηση



Εικόνα 158 – Μικρότερη περίοδος με θετικό WNS (5.270 ns)



Εικόνα 159 – Μεγαλύτερη περίοδος με αρνητικό WNS (5.260 ns)

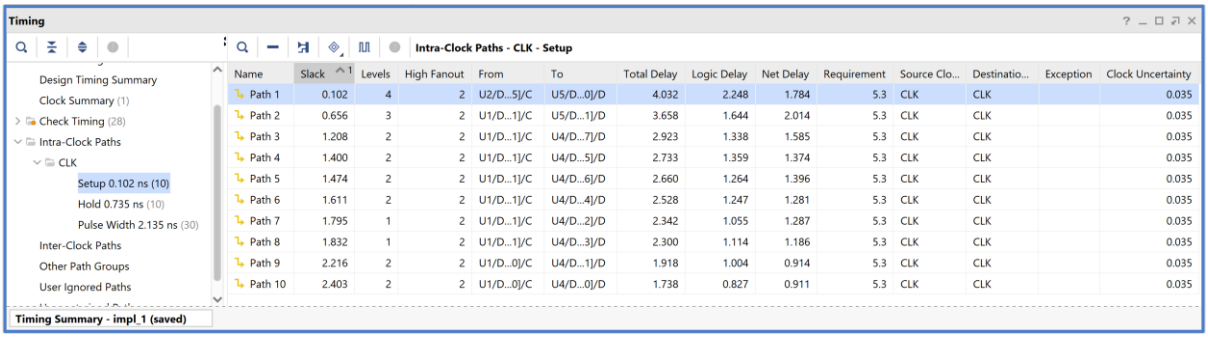
Με βάση τα impl_1 report επιλέγουμε την περίοδο των **5.270 ns** ως μία ικανοποιητική περίοδο για το σήμα **CLK**. Μέγιστη συχνότητα λειτουργίας: **189.8 MHz**.

Στη στήλη **Setup** παρουσιάζονται τα αποτελέσματα του *max delay analysis*.

- Το *Worst Negative Slack* (**WNS**) είναι μια τιμή (με link) που αντιστοιχεί στο μικρότερο διαθέσιμο περιθώριο (slack) που προκύπτει από την ανάλυση όλων των κρίσιμων διαδρομών του max delay analysis. Ένα θετικό slack (0.102 ns) δηλώνει ότι η κρίσιμη διαδρομή ικανοποιεί την περίοδο του CLK.
- Το *Total Negative Slack* (**TNS**) είναι το άθροισμα όλων των αρνητικών WNS για κάθε timing path endpoint. Η τιμή είναι 0.000 ns όταν δεν υπάρχουν παραβιάσεις του χρόνου σταθεροποίησης. Το ψηφιακό κύκλωμα λειτουργεί κανονικά στην επιλεγμένη συχνότητα λειτουργίας.

Στη στήλη **Hold** παρουσιάζονται τα αποτελέσματα του min delay analysis.

- Το *Worst Hold Slack* (**WHS**) είναι μια τιμή (με link) που αντιστοιχεί στο μικρότερο διαθέσιμο περιθώριο (slack) που προκύπτει από την ανάλυση όλων των σύντομων διαδρομών του min delay analysis. Ένα θετικό slack (0.735) δηλώνει ότι δεν παραβιάζεται ο χρόνος διατήρησης (hold time).
- Το *Total Hold Slack* (**THS**) είναι το άθροισμα όλων των αρνητικών WHS για κάθε timing path endpoint. Η τιμή είναι 0.000 ns όταν δεν υπάρχουν παραβιάσεις του χρόνου διατήρησης. Το ψηφιακό κύκλωμα λειτουργεί κανονικά.
- Επιλέξτε το **WNS link** και δείτε τις 10 χειρότερες κρίσιμες διαδρομές (δηλαδή με το μικρότερο θετικό slack) που δεν παραβιάζουν το χρόνο σταθεροποίησης. Η κρίσιμη διαδρομή έχει καθυστέρηση διάδοσης 4.032 ns, εκ των οποίων τα 2.248 ns αφορούν στη λογική (logic), ενώ τα 1.784 ns αφορούν στη δικτύωση (net). Η αβεβαιότητα του CLK εκτιμάται στα 0.035 ns. Τα επίπεδα λογικής (logic level) είναι 4.

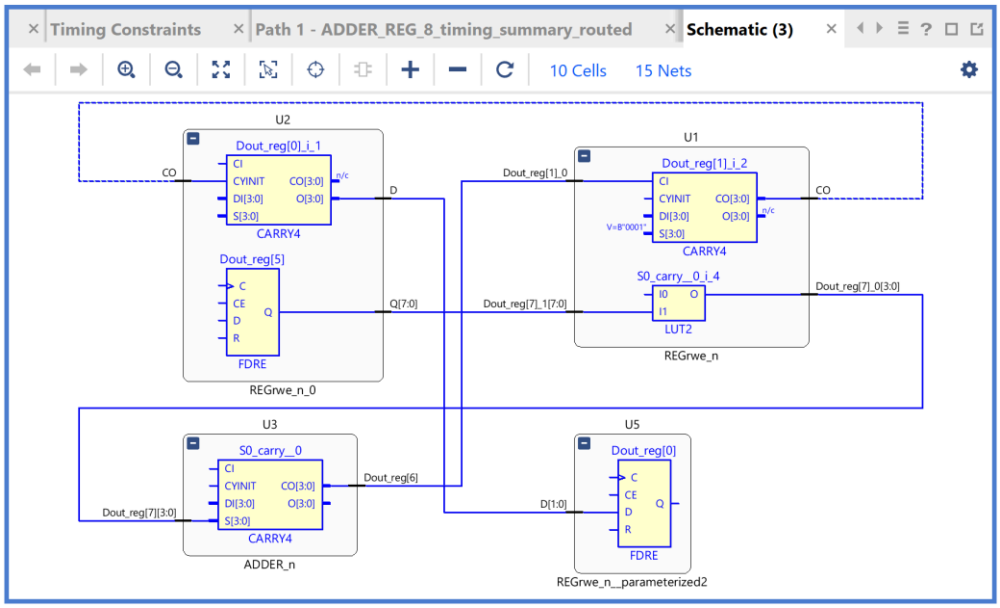


Εικόνα 160 – Μελέτη κρίσιμης διαδρομής με το νέο clock constraint

- Επιλέξτε με διπλό κλικ το **Path 1**, ώστε να εμφανιστεί το παράθυρο *Path 1 – ADDER_REG_8_timing_summary_routed*. Η κρίσιμη διαδρομή περνάει από 3 μονάδες CARRY4 και 1 LUT2. Υπολογίστε το WNS slack (σε παρένθεση τα αποτελέσματα της υλοποίησης με σήμα **CLK** στα 10.000 ns):
- η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Source Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK του καταχωρητή της υπομονάδας U2) είναι **5.629** (5.632) ns,
- η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Data Path* (από την είσοδο CLK μέχρι την έξοδο Q του καταχωρητή της υπομονάδας U2, το LUT2 (U1) και των 3 μονάδων CARRY4 (U3–U1–U2) μέχρι την είσοδο D του καταχωρητή της υπομονάδας U5) είναι **4.032** (4.535) ns,
- το **Arrival Time**, ως άθροισμα των ανωτέρων χρόνων, είναι **9.661** (10.167) ns,
- το **Required Time**, ως η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Destination Clock Path* (από την πηγή του CLK τη χρονική στιγμή **5.270** ns μέχρι την είσοδο CLK του καταχωρητή της υπομονάδας U5) συν τον χρόνο σταθεροποίησης (–1.013) είναι **9.763** (14.490) ns.

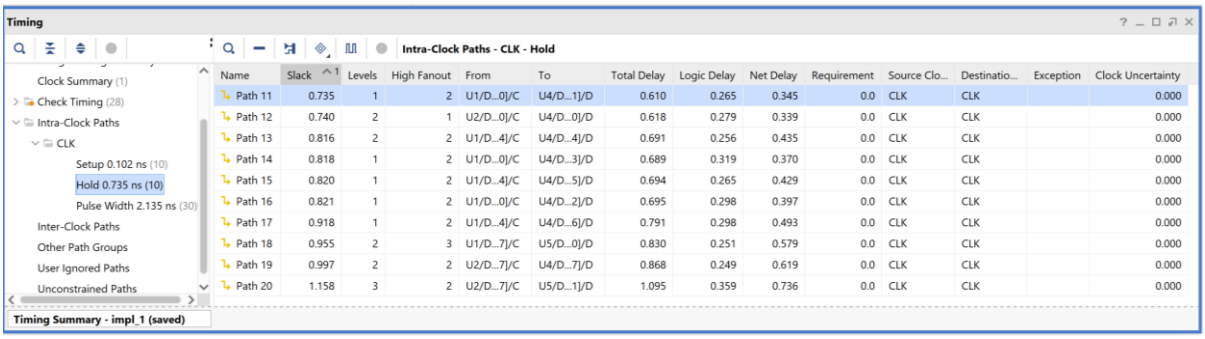
  WNS slack = Required Time – Arrival Time = 9.763 – 9.661 = 0.102 ns

- Επιλέξτε με δεξί κλικ στο **Path 1**, το **Schematic**. Μελετήστε το σχηματικό διάγραμμα της κρίσιμης διαδρομής της οντότητας **ADDER_REG_8**. (Έχει αλλάξει η διαδρομή).



Εικόνα 161 – Σχηματικό διάγραμμα κρίσιμης διαδρομής με το νέο clock constraint

- Επιλέξτε το **WHS link** και δείτε τις 10 χειρότερες σύντομες διαδρομές (δηλαδή με το μικρότερο θετικό slack) που δεν παραβιάζουν το χρόνο διατήρησης). Η σύντομη διαδρομή έχει καθυστέρηση μόλυνσης 0.610 ns, εκ των οποίων τα 0.265 ns αφορούν στη λογική (logic), ενώ τα 0.345 ns αφορούν στη δικτύωση (net). Η αβεβαιότητα του CLK εκτιμάται στα 0.000 ns. Τα επίπεδα λογικής (logic level) είναι 1.

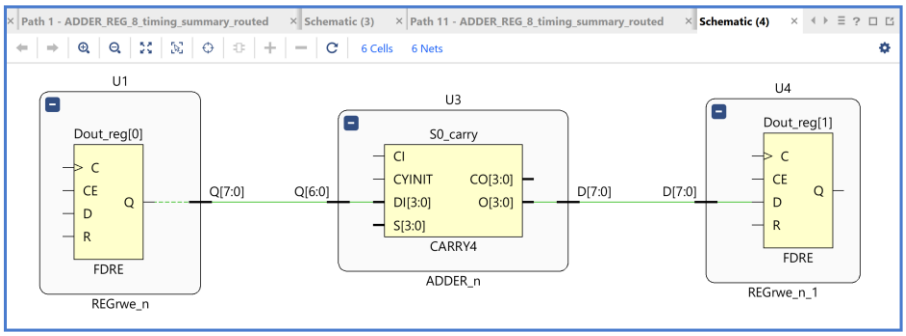


Εικόνα 162 – Μελέτη χειρότερης σύντομης διαδρομής με το νέο clock constraint

- Επιλέξτε με διπλό κλικ το **Path 11**, ώστε να εμφανιστεί το παράθυρο *Path 11 – ADDER_REG_8_timing_summary_routed*. Η σύντομη διαδρομή περνάει από 1 μονάδα CARRY4. Υπολογίστε το WHS slack (σε παρένθεση τα αποτελέσματα της υλοποίησης με σήμα **CLK** με περίοδο των 10.000 ns):
- η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Source Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK του καταχωρητή της υπομονάδας U1) είναι **1.581** (1.580) ns,
- η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Data Path* (από την είσοδο CLK μέχρι την έξοδο Q του καταχωρητή της υπομονάδας U1 και μέσω της 1 μονάδας CARRY4 (U3) μέχρι την είσοδο D του καταχωρητή της υπομονάδας U4) είναι **0.610** (0.566) ns,
- το **Arrival Time**, ως άθροισμα των ανωτέρων χρόνων, είναι **2.191** (2.146) ns,
- το **Required Time**, ως η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Destination Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK του καταχωρητή της υπομονάδας U4) συν τον χρόνο διατήρησης (–0.155), είναι **1.456** (1.456) ns.

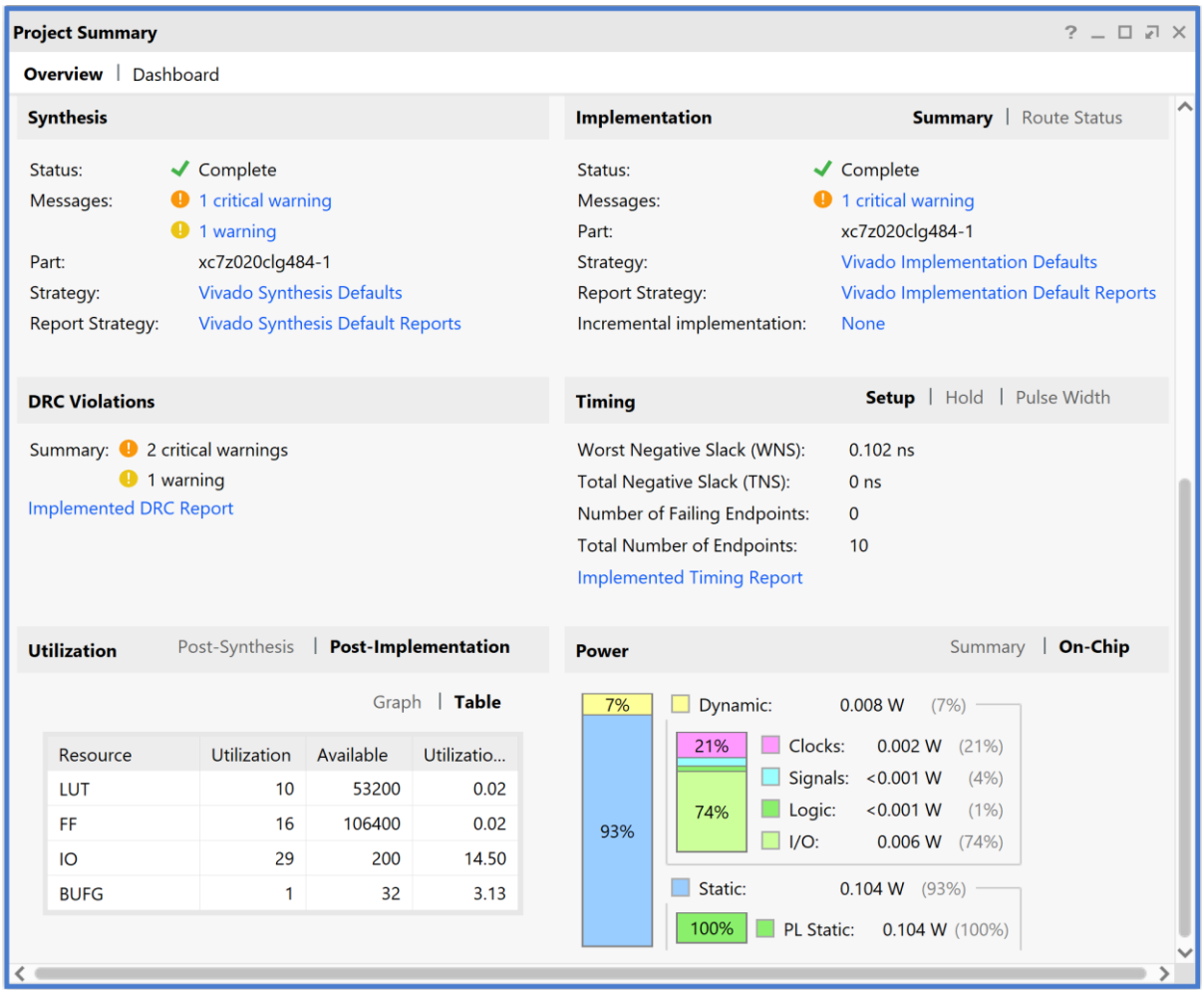
  WHS slack = Arrival Time – Required Time = 2.191 – 1.456 = 0.735 ns

- Επιλέξτε με δεξί κλικ στο **Path 11**, το **Schematic**. Μελετήστε το σχηματικό διάγραμμα της σύντομης διαδρομής της οντότητας **ADDER_REG_8**. (Έχει αλλάξει η διαδρομή).



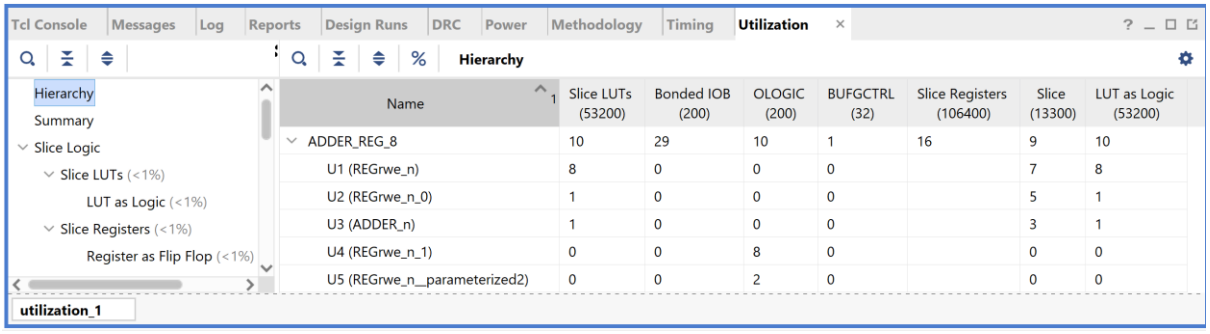
Εικόνα 163 – Σχηματικό διάγραμμα χειρότερης σύντομης διαδρομής με το νέο clock constraint

- Στη συνέχεια, επιλέξτε το παράθυρο *Project Summary* και μελετήστε τα διάφορα υπο–παράθυρα.



Εικόνα 164 – Project summary με το νέο clock constraint

- Στη συνέχεια, στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, μέσα στο *Open Implemented Design* επιλέξετε το **Schematic** για να δείτε το σχηματικό διάγραμμα του **implemented design model**. Δεν έχει αλλάξει με την αλλαγή της περιόδου του σήματος **CLK**.
- Τέλος, στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, μέσα στο *Open Implemented Design* επιλέξετε το **Report Utilization** και μελετήστε τους πόρους που χρησιμοποιεί η οντότητα **ADDER_REG_8**. Πατήστε **OK**. (Μπορείτε να μελετήσετε και άλλα ενδιαφέροντα reports, εάν επιλέξετε το παράθυρο *Reports*).



Εικόνα 165 – Report utilization για την οντότητα ADDER_REG_8

1.10.3 Εκτέλεση προσομοίωσης μετά την υλοποίηση (λογική και χρονική) στο VIVADO IDE για συνδυαστική λογική ανάμεσα σε καταχωρητές

Η προσομοίωση μετά την υλοποίηση (λογική και χρονική) εκτελείται στην οντότητα **ADDER_REG_8_TB** του προγράμματος δοκιμών (testbench) που έχει ορισθεί ως η κορυφαία οντότητα της ιεραρχίας (**top**) των simulation resources. Κατά την προσομοίωση, η οντότητα **ADDER_REG_8_TB** καλεί το *UUT* της (που είναι το **implemented design model** της οντότητας **ADDER_REG_8**).

- Στο αρχείο **ADDER_REG_8_TB.vhd** αλλάξτε την περίοδο του σήματος **CLK** στη βέλτιστη περίοδο που βρήκατε στο Βήμα 5–2. Πατήστε **Save**.

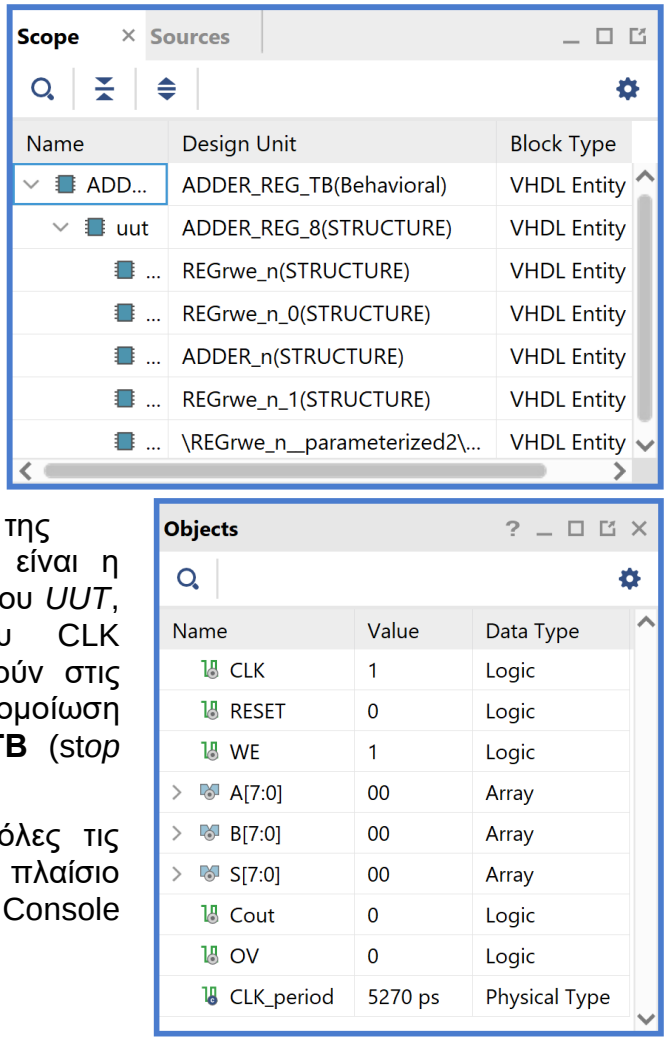
```
67  -- Clock period definitions
68  constant CLK_period : time := 5.270 ns;
```

- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Run Simulation**, ώστε να εμφανιστούν όλες οι πιθανές προσομοιώσεις που υποστηρίζει το Vivado IDE. Επιλέξτε **Run Post–Implementation Functional Simulation**.
- Το πρόγραμμα δοκιμής (testbench) και όλες οι οντότητες του *UUT* θα γίνουν compiled και θα τρέξει το Vivado simulator (εφόσον βέβαια δεν υπάρχουν σφάλματα). Θα εμφανιστεί το παράθυρο *SIMULATION* που απαρτίζεται από 4 παράθυρα:

Το παράθυρο *Scope*, όπου παρουσιάζεται η οντότητα **ADDER_REG_8_TB**, το **implemented design model** της οντότητας **ADDER_REG_8** (*UUT*) καθώς και οι υπόλοιπες νέες οντότητες του *UUT* που προκύπτουν μετά την υλοποίηση. Όλες οι οντότητες που απαρτίζουν το *UUT* είναι *structural*.

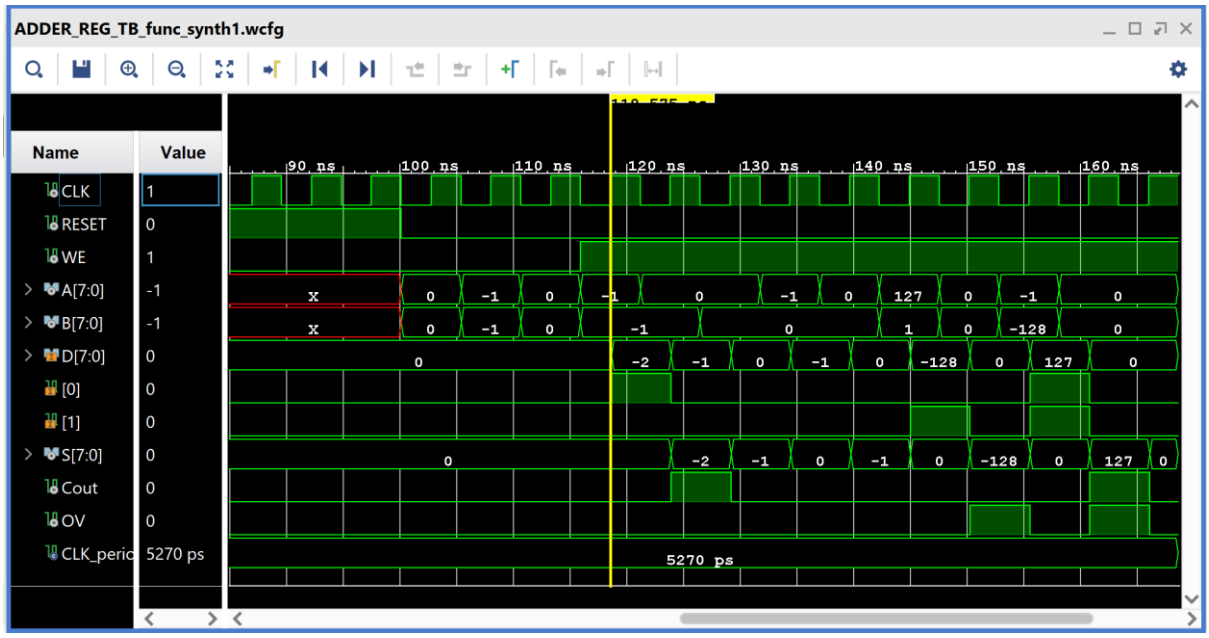
Το παράθυρο *Objects*, όπου εμφανίζονται τα σήματα top–level, δηλαδή οι είσοδοι και οι έξοδοι της οντότητας **ADDER_REG_8**, που είναι η κορυφαία οντότητα της ιεραρχίας του *UUT*, καθώς και η περίοδος του CLK (*CLK_period*). Οι τιμές αντιστοιχούν στις τιμές που έχει σταματήσει η προσομοίωση της οντότητας **ADDER_REG_8_TB** (*stop (2)*).

Το παράθυρο *Tcl Console* με όλες τις διαδικασίες που εκτελούνται στο πλαίσιο της προσομοίωσης. Το Tcl Console καθαρίζει με την επιλογή *Clear*.



Εικόνα 166 - Παράθυρα Scope και Objects

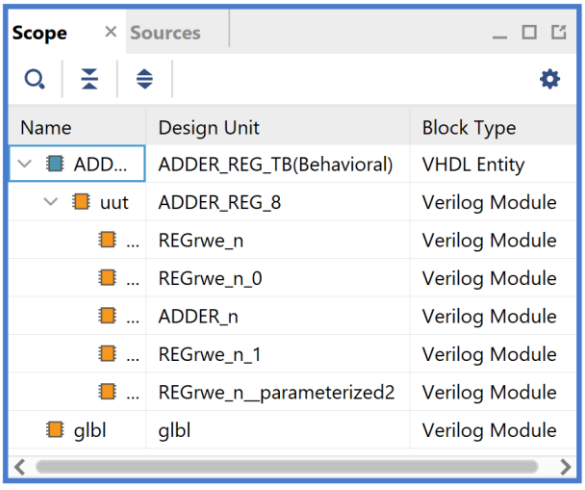
Επιλέξτε το παράθυρο *ADDER_REG_TB_func_synth1.wcfg*, που δημιουργήσατε για την προσομοίωση του **synthesized design model**, όπου συμπεριλαμβάνονται πέραν των top–level εισόδων/εξόδων και τα απαραίτητα **εσωτερικά σήματα** της εξόδου του αθροιστή (**D[7:0]** που αντιστοιχεί στην έξοδο **S[7:0]**, **[0]** που αντιστοιχεί στο **Cout** και **[1]** που αντιστοιχεί στο **OV**. Βλέπετε ολόκληρο το διάγραμμα χρονισμού της λογικής προσομοίωσης μετά τη σύνθεση με κατάλληλο **zoom out** και επιλέγοντας το **zoom fit**.



Εικόνα 167 – Διάγραμμα χρονισμού λογικής προσομοίωσης μετά την υλοποίηση

- Κλείστε τον simulator επιλέγοντας το κουμπί × πάνω δεξιά στο παράθυρο του *SIMULATION*. Στο παράθυρο *Confirm Close* που εμφανίζεται πατήστε **OK**.
- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Run Simulation**, ώστε να εμφανιστούν όλες οι πιθανές προσομοιώσεις που υποστηρίζει το Vivado IDE. Επιλέξτε **Run Post–Implementation Timing Simulation**.
- Το πρόγραμμα δοκιμής (testbench) και όλες οι οντότητες του *UUT* θα γίνουν compiled και θα τρέξει το Vivado simulator (εφόσον βέβαια δεν υπάρχουν σφάλματα). Θα εμφανιστεί το παράθυρο *SIMULATION* που απαρτίζεται από 4 παράθυρα:

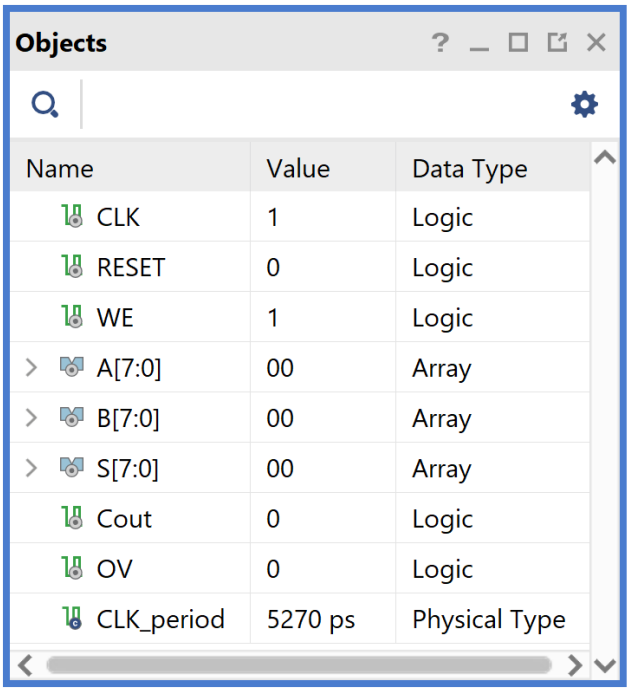
  Το παράθυρο *Scope*, όπου παρουσιάζεται η οντότητα **ADDER_REG_8_TB**, το **implemented design model** της οντότητας **ADDER_REG_8** (*UUT*) καθώς και οι υπόλοιπες νέες οντότητες του *UUT* που προκύπτουν μετά τη σύνθεση για χρονική προσομοίωση. Όλες οι οντότητες που απαρτίζουν πλέον το *UUT* είναι *Verilog Module!*



Εικόνα 168 - Παράθυρο Scope

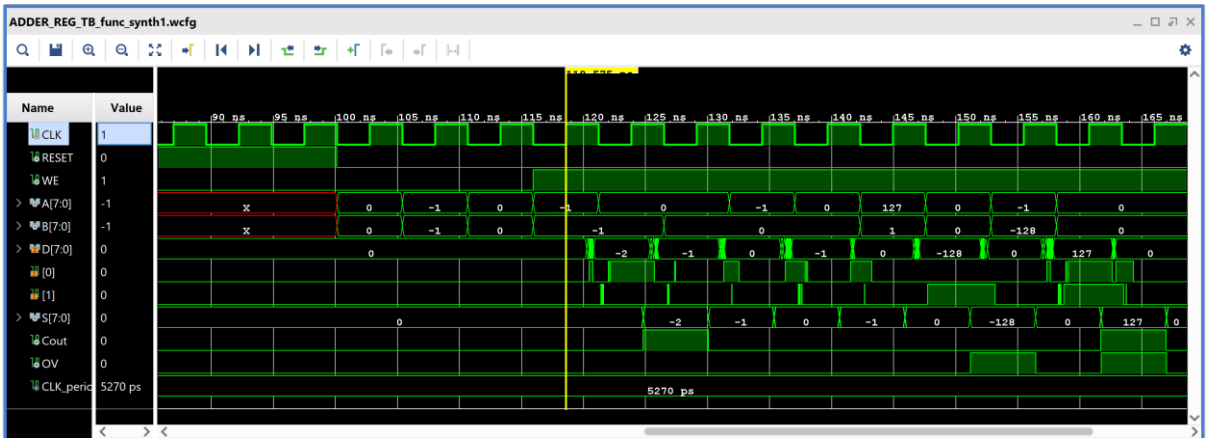
Το παράθυρο *Objects*, όπου εμφανίζονται τα σήματα top–level, δηλαδή οι είσοδοι και οι έξοδοι της οντότητας **ADDER_REG_8**, που είναι η κορυφαία οντότητα της ιεραρχίας του *UUT*, καθώς και η περίοδος του CLK (*CLK_period*). Οι αρτηρίες (array) αναλύονται στα σήματα που τις απαρτίζουν. Οι τιμές αντιστοιχούν στις τιμές που έχει σταματήσει η προσομοίωση της οντότητας **ADDER_REG_8_TB** (st*op (2)*).

Το παράθυρο *Tcl Console* με όλες τις διαδικασίες που εκτελούνται στο πλαίσιο της προσομοίωσης. Το Tcl Console καθαρίζει με την επιλογή *Clear*.



Εικόνα 169 - Παράθυρο Objects

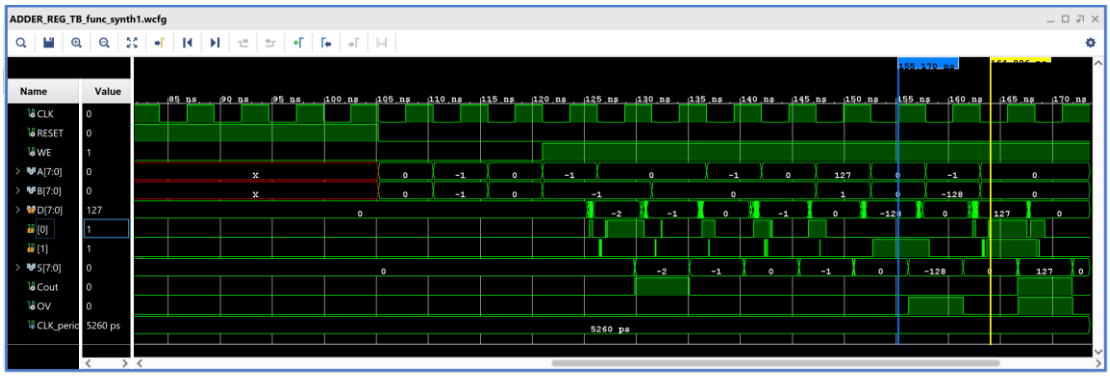
Επιλέξτε το παράθυρο *ADDER_REG_TB_func_synth1.wcfg*. Βλέπετε ολόκληρο το διάγραμμα χρονισμού της χρονικής προσομοίωσης μετά την υλοποίηση με κατάλληλο **zoom out** ή επιλέγοντας το **zoom fit** .



Εικόνα 170 – Διάγραμμα χρονισμού χρονικής προσομοίωσης μετά την υλοποίηση

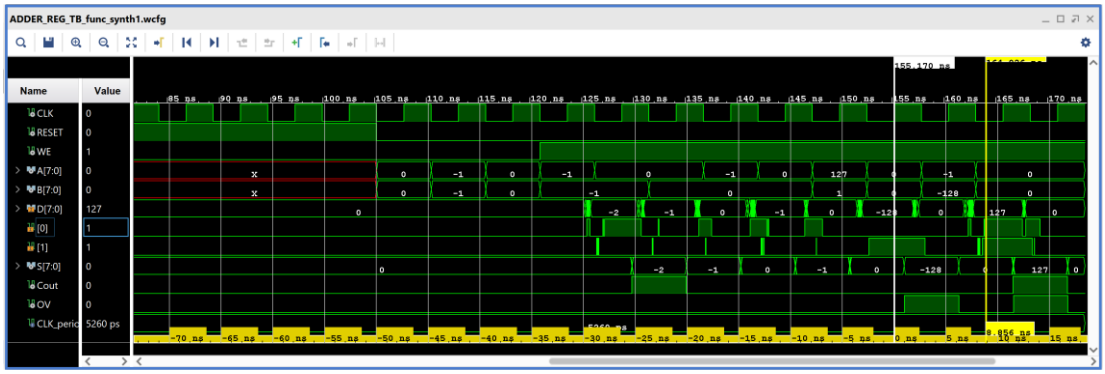
- Συγκρίνετε τα διαγράμματα χρονισμού της χρονικής και λογικής προσομοίωσης μετά την υλοποίηση. Είναι εμφανείς οι καθυστερήσεις διάδοσης στο πρώτο διάγραμμα χρονισμού.
- Υπολογίστε το **Arrival Time** ενός σήματος στο διάγραμμα χρονισμού της χρονικής προσομοίωσης μετά την υλοποίηση με τη χρήση των marker. Για παράδειγμα, βάλτε τον μπλε marker στην ανερχόμενη ακμή του **CLK** στα 155.170 ns και τον κίτρινο marker στην ανερχόμενη ακμή του εσωτερικού σήματος **[0]** (που αντιστοιχεί στο Cout) στα 164.026 ns. Απέχουν 8.856 ns. Συγκρίνετε με το **Arrival Time** που είχατε βρει κατά τη χρονική ανάλυση για την κρίσιμη διαδρομή, που ήταν 9.661 ns.

  Προσοχή! Πρέπει να βάλουμε τον μπλε marker **έναν κύκλο πριν**, επειδή η περίοδος του **CLK** είναι **μικρότερη** του **Arrival Time**.



Εικόνα 171 – Επαλήθευση του arrival time (βήμα #1)

Με κλικ πάνω στον μπλε marker (γίνεται λευκός), ορίζουμε τη θέση του στα 0.000 ns και είναι πλέον εμφανές ότι ο κίτρινος marker απέχει 8.856 ns.



Εικόνα 172 – Επαλήθευση του arrival time (βήμα #2)

- Κλείστε τον simulator επιλέγοντας το κουμπί × πάνω δεξιά στο παράθυρο του *SIMULATION*. Στο παράθυρο *Confirm Close* που εμφανίζεται πατήστε **OK**. Εάν επιθυμείτε να μη σώσετε ένα επιπλέον waveform configuration, στο παράθυρο *Save Waveform Configuration* επιλέξτε **Discard**.

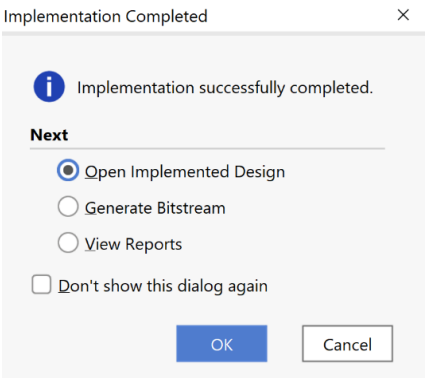
Η διαδικασία που ήδη περιγράψαμε στα Βήματα 5–1, 5–2 και 5–3 της «**Υλοποίησης στη τεχνολογία FPGA και προσομοίωση (λογική, χρονική)**» εφαρμόζεται παρομοίως σε κάθε πιθανή υπομονάδα συνδυαστικής λογικής της διαδρομής δεδομένων του επεξεργαστή.

1.10.4 Εκτέλεση της διαδικασίας της υλοποίησης και ανάλυση των αποτελεσμάτων μετά την υλοποίηση στο VIVADO IDE για μηχανή πεπερασμένων καταστάσεων (FSM)

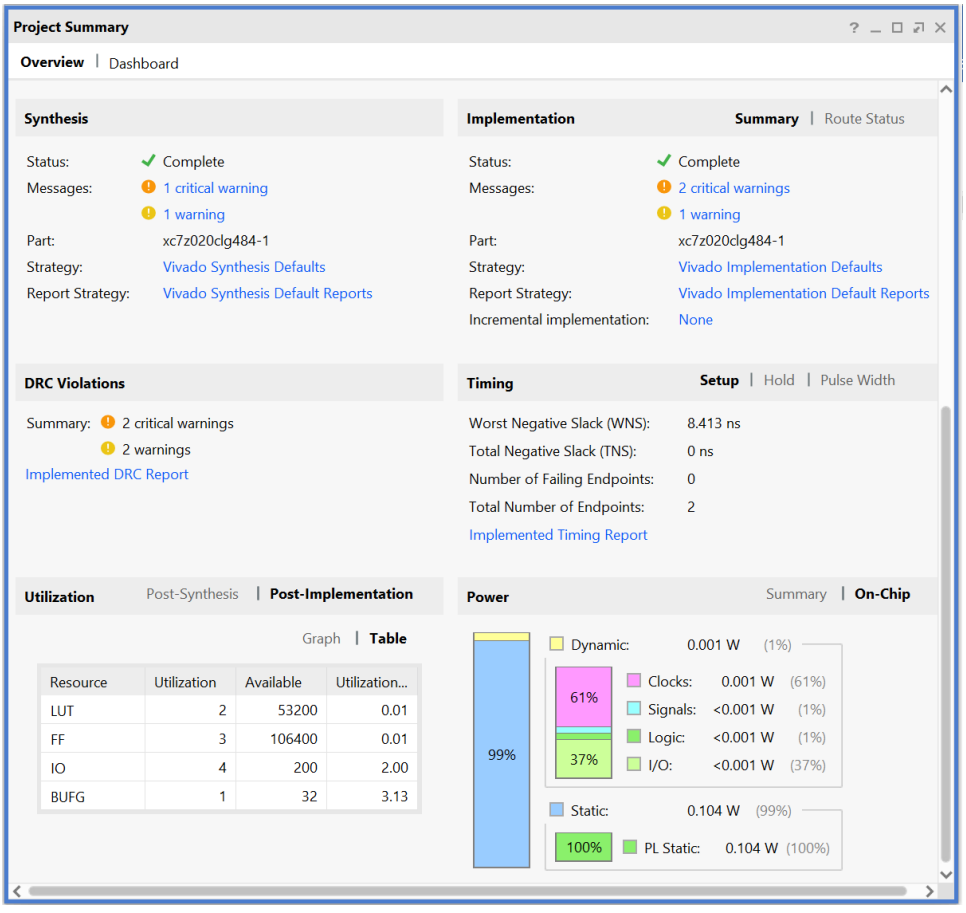
Η διαδικασία της υλοποίησης εφαρμόζεται στο **synthesized design model** της οντότητας που έχει ορισθεί ως η κορυφαία οντότητα της ιεραρχίας (**top**) των design resources και μετά από την ολοκλήρωση της υλοποίησης παράγεται το αντίστοιχο **implemented design model**.

Η υλοποίηση στην τεχνολογία **Zynq–7000 (device xc7z020clg484–1)** εκτελείται στο **synthesized design model** της οντότητας **PATTERN_FSM** που έχει ορισθεί ως η κορυφαία οντότητα της ιεραρχίας (**top**) των design resources. Με τη υλοποίηση το εργαλείο Vivado IDE παράγει το **implemented design model** της οντότητας **PATTERN_FSM**.

- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Run Implementation**. Πατήστε **OK** στα παράθυρα προειδοποίησης που εμφανίζονται.
- Στο παράθυρο *Implementation Completed* υπάρχουν τρεις επιλογές. Επιλέξτε το **Open Implemented Design** και πατήστε **OK** για να μελετήσετε το αποτέλεσμα της υλοποίησης (implementation).
- Αρχικά, επιλέξτε το παράθυρο *Project Summary* και μελετήστε τα διάφορα υπο–παράθυρα.

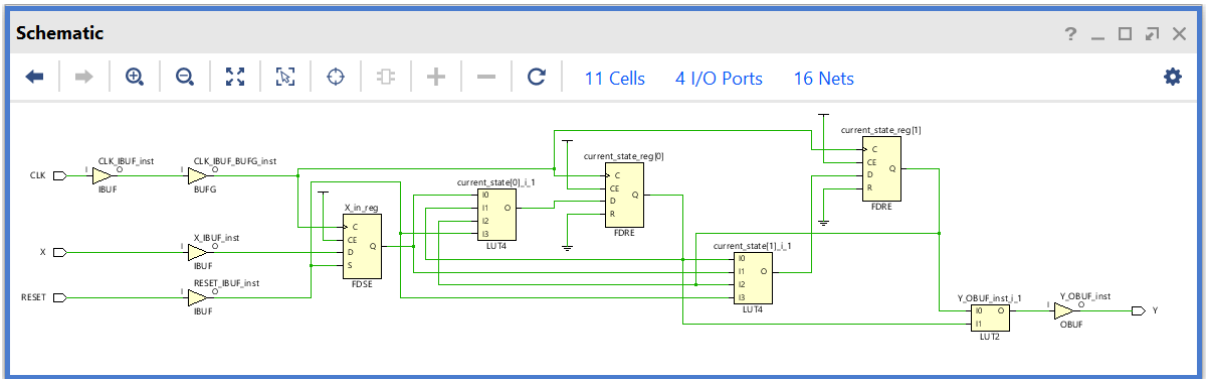


Εικόνα 173 - Επιλογές μετά την υλοποίηση



Εικόνα 174 – Project Summary για το PATTERN_FSM implemented design model

- Στη συνέχεια, στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, μέσα στο *Open Implemented Design* επιλέξετε το **Schematic** για να δείτε το σχηματικό διάγραμμα του **implemented design model**.

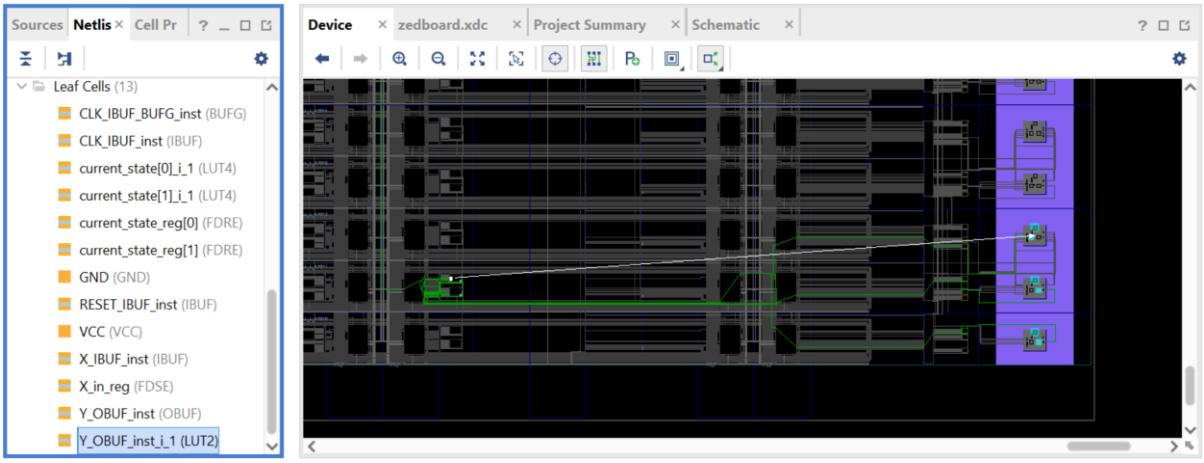


Εικόνα 175 – Σχηματικό διάγραμμα του PATTERN_FSM implemented design model

Παρατηρείστε ότι έχουν αυτόματα προστεθεί τα απαραίτητα IBUFs, OBUFs και BUFG primitives στο σχηματικό διάγραμμα, καθώς και ότι οι είσοδοι και οι έξοδοι από το FPGA είναι buffered. Παραμένει ίδιο με το σχηματικό διάγραμμα μετά τη σύνθεση.

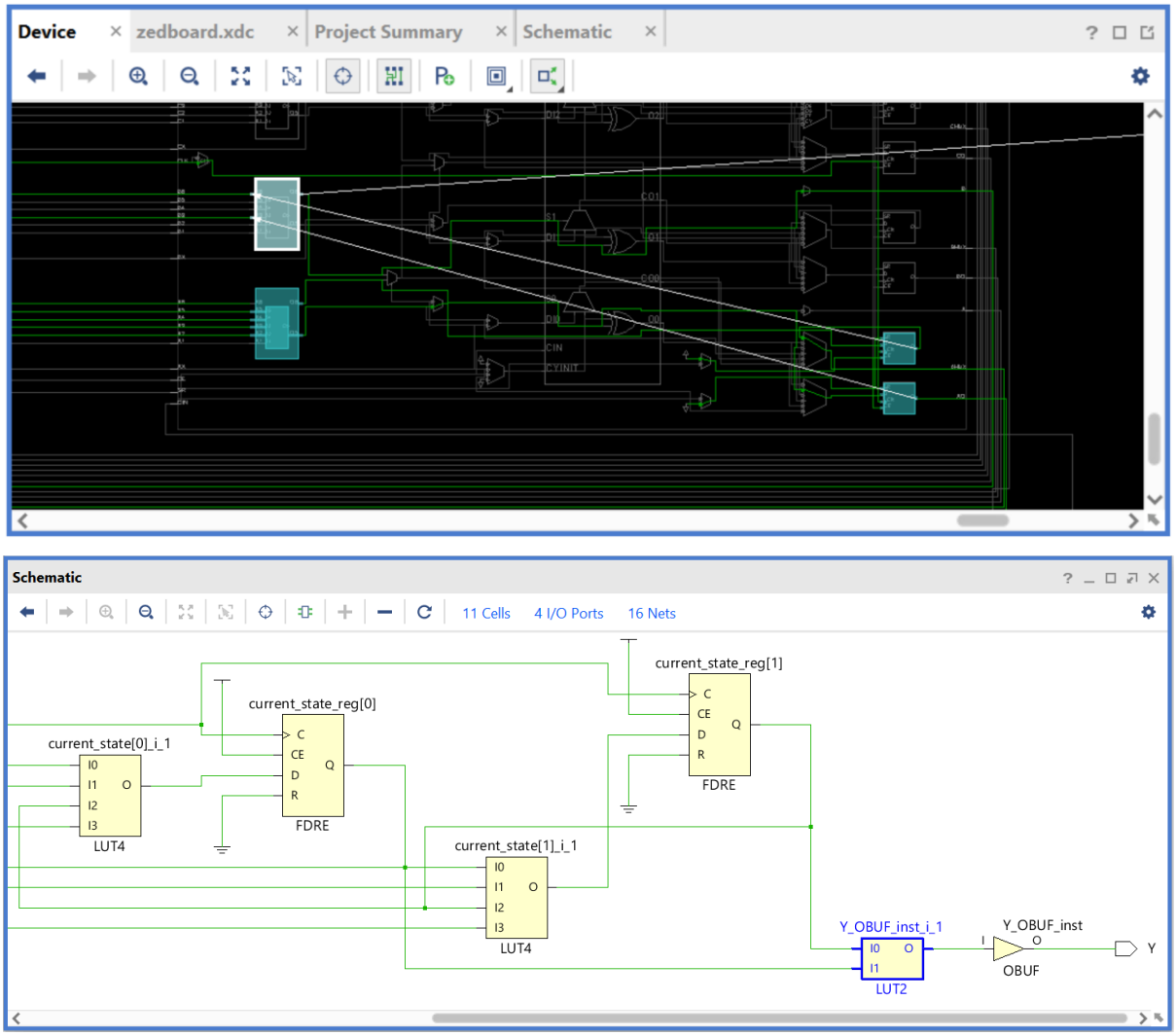
- Στη συνέχεια μελετήστε το παράθυρο *Device* έχοντας πατήσει τα κουμπιά *auto–fit selection*, *routing resources* και *show cell connections*.

Στη μελέτη σας επιλέξτε κάποιο *leaf cell*, όπως για παράδειγμα το LUT2 που παράγει το Y με κατάλληλη επιλογή στο παράθυρο *Netlist*.



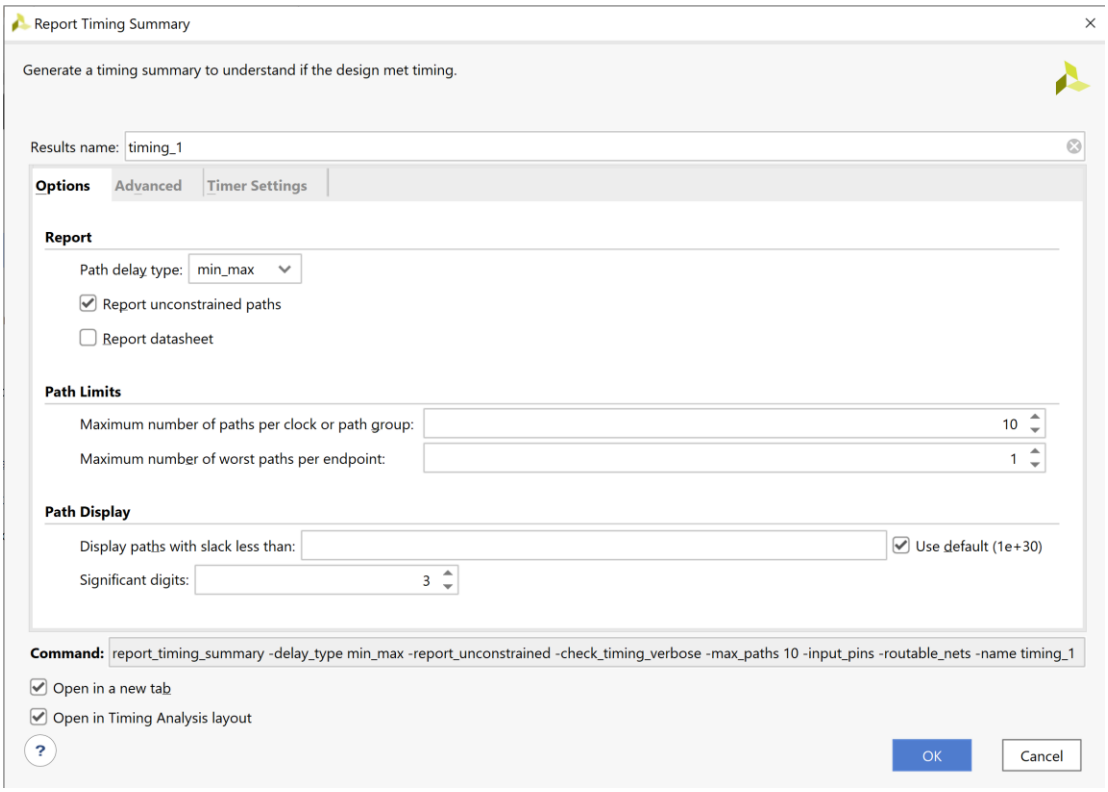
Εικόνα 176 – Μελέτη leaf cell από το παράθυρο Netlist

- Επικεντρώστε στη δημιουργία του Y με παράλληλη μελέτη των παραθύρων *Device* και *Schematic*, εναλλάξ. Απαιτείται εντοπισμένη μεγέθυνση. Στο SLICE_X113Y1 φαίνονται: το **LUT2** (Y_OBUF_inst_i_1), το **LUT4** (current_state_[0]_i_1), το **FDRE** (current_state_reg[1]) και το **FDRE** (current_state_reg[0]).



Εικόνα 177 – Παράλληλη μελέτη δημιουργίας του σήματος Y στα παράθυρα Device και Schematic

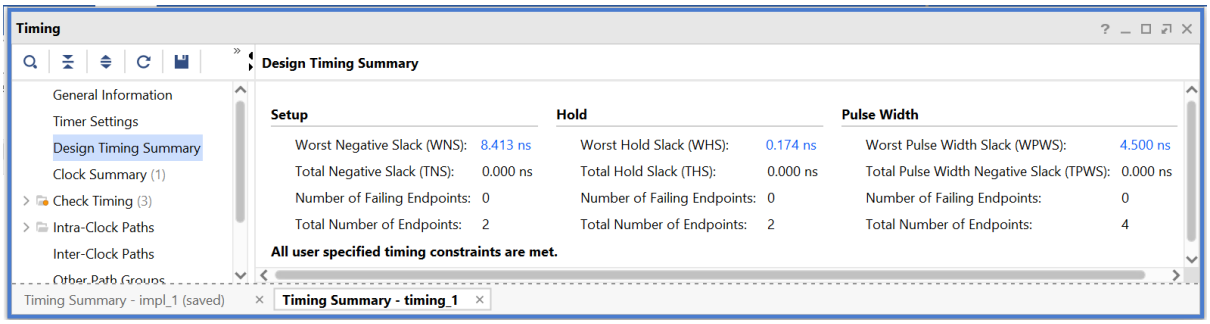
- Στη συνέχεια, στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, μέσα στο *Open Implemented Design* επιλέξετε το **Report Timing Summary** για να δείτε την ανάλυση χρονισμού που κάνει το εργαλείο Vivado IDE στο **implemented design model**. Αυτή είναι η πιο ακριβής ανάλυση χρονισμού που έχουμε διαθέσιμη.



Εικόνα 178 – Επιλογές για την παραγωγή του timing summary για το implemented design model

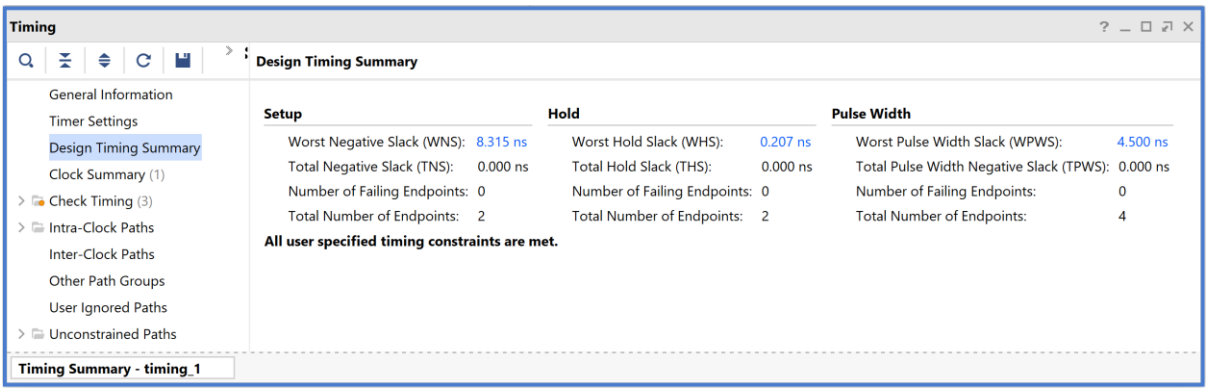
Στην επιλογή *Path delay type* ορίζεται ο τύπος της ανάλυσης που θα εκτελεσθεί. Το *max delay analysis* αφορά στην εύρεση της κρίσιμης διαδρομής (με τη μεγαλύτερη καθυστέρηση διάδοσης) και συνεπώς στην εύρεση της μέγιστης συχνότητας λειτουργίας χωρίς την παραβίαση του **χρόνου σταθεροποίησης** (setup time). Το *min delay analysis* αφορά στην εύρεση της σύντομης διαδρομής (με τη μικρότερη καθυστέρηση διάδοσης) χωρίς την παραβίαση του **χρόνου διατήρησης** (hold time).

- Πατήστε **OK** για να παραχθεί το Timing_1 report.



Εικόνα 179 – Timing report για το implemented design model

Συγκρίνετε με το Timing_1 report που είχε παραχθεί μετά τη σύνθεση.



Εικόνα 180 – Timing report για το synthesized design model

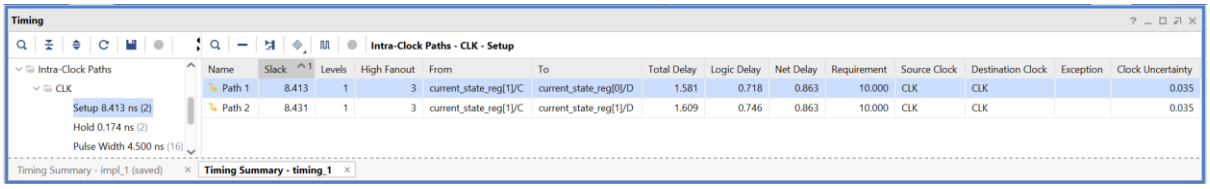
Στη στήλη **Setup** παρουσιάζονται τα αποτελέσματα του *max delay analysis*.

- Το *Worst Negative Slack* (**WNS**) είναι μια τιμή (με link) που αντιστοιχεί στο μικρότερο διαθέσιμο περιθώριο (slack) που προκύπτει από την ανάλυση όλων των κρίσιμων διαδρομών του max delay analysis. Μπορεί να είναι θετικό η αρνητικό. Ένα θετικό slack (8.413 ns) δηλώνει ότι η κρίσιμη διαδρομή ικανοποιεί την περίοδο του CLK. Ένα αρνητικό slack δηλώνει ότι παραβιάζεται ο χρόνος σταθεροποίησης (setup time).
- Το *Total Negative Slack* (**TNS**) είναι το άθροισμα όλων των αρνητικών WNS για κάθε timing path endpoint. Η τιμή είναι 0.000 ns όταν δεν υπάρχουν παραβιάσεις του χρόνου σταθεροποίησης. Το ψηφιακό κύκλωμα λειτουργεί κανονικά στην επιλεγμένη συχνότητα λειτουργίας.

Στη στήλη **Hold** παρουσιάζονται τα αποτελέσματα του min delay analysis.

- Το *Worst Hold Slack* (**WHS**) είναι μια τιμή (με link) που αντιστοιχεί στο μικρότερο διαθέσιμο περιθώριο (slack) που προκύπτει από την ανάλυση όλων των σύντομων διαδρομών του min delay analysis. Μπορεί να είναι θετικό η αρνητικό. Ένα αρνητικό slack δηλώνει ότι παραβιάζεται ο χρόνος διατήρησης (hold time).
- Το *Total Hold Slack* (**THS**) είναι το άθροισμα όλων των αρνητικών WHS για κάθε timing path endpoint. Η τιμή είναι 0.000 ns όταν δεν υπάρχουν παραβιάσεις του χρόνου διατήρησης. Το ψηφιακό κύκλωμα λειτουργεί κανονικά.

- Επιλέξτε το **WNS link** και δείτε τις 2 διαθέσιμες κρίσιμες διαδρομές (δηλαδή με το μικρότερο θετικό slack) που δεν παραβιάζουν το χρόνο σταθεροποίησης. Η κρίσιμη διαδρομή έχει καθυστέρηση διάδοσης 1.581 ns, εκ των οποίων τα 0.718 ns αφορούν στη λογική (logic), ενώ τα 0.863 ns αφορούν στη δικτύωση (net). Η αβεβαιότητα του CLK εκτιμάται στα 0.035 ns. Τα επίπεδα λογικής (logic level) είναι 1.

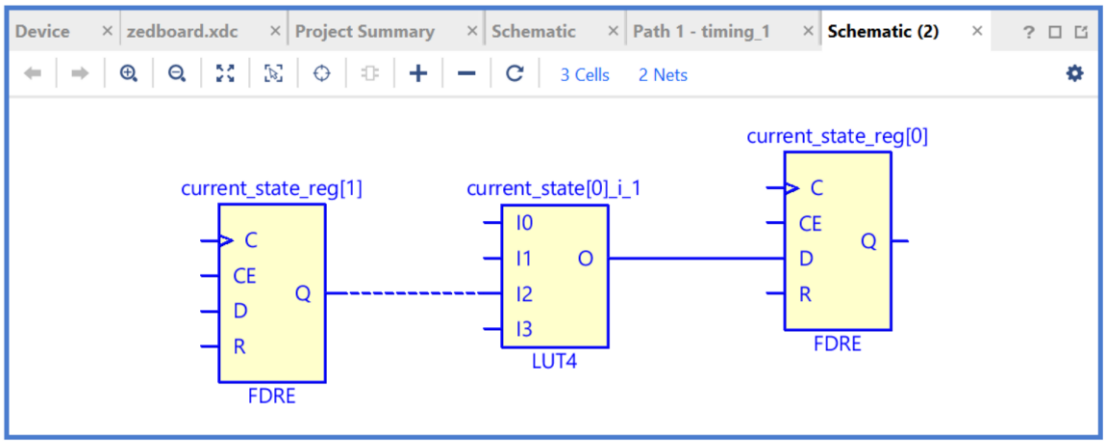


Εικόνα 181 – Μελέτη κρίσιμης διαδρομής στο implemented design model

- Επιλέξτε με διπλό κλικ το **Path 1**, ώστε να εμφανιστεί το παράθυρο *Path 1 – timing_1*. Η κρίσιμη διαδρομή περνάει από 1 LUT4. Υπολογίστε το WNS slack (σε παρένθεση το αποτέλεσμα της σύνθεσης):
- η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Source Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK του καταχωρητή κατάστασης 1) είναι **5.642** (2.975) ns,
- η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Data Path* (από την είσοδο CLK μέχρι την έξοδο Q του καταχωρητή κατάστασης 1 και μέσω του LUT4 μέχρι την είσοδο D του καταχωρητή κατάστασης 0) είναι **1.581** (1.549) ns,
- το **Arrival Time**, ως άθροισμα των ανωτέρων χρόνων, είναι **7.223** (4.524) ns,
- το **Required Time**, ως η καθυστέρηση διάδοσης της κρίσιμης διαδρομής του *Destination Clock Path* (από την πηγή του CLK τη χρονική στιγμή 10.000 ns μέχρι την είσοδο CLK του καταχωρητή κατάστασης 0) συν τον χρόνο σταθεροποίησης είναι **15.636** (12.839) ns.

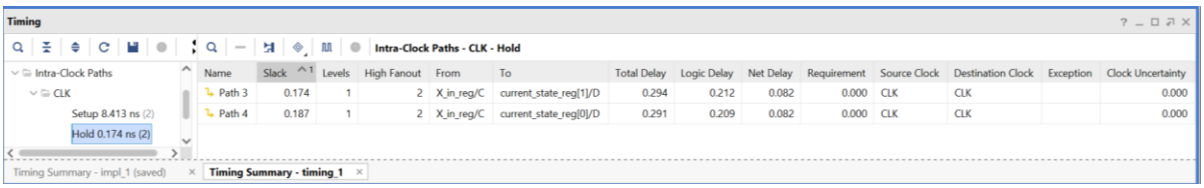
  WNS slack = Required Time – Arrival Time = 15.636 – 7.223 = 8.413 ns

- Επιλέξτε με δεξί κλικ στο **Path 1**, το **Schematic**. Μελετήστε το σχηματικό διάγραμμα της κρίσιμης διαδρομής της οντότητας **PATTERN_FSM**. (Δεν έχει αλλάξει η διαδρομή).



Εικόνα 182 – Σχηματικό διάγραμμα κρίσιμης διαδρομής στο implemented design model

- Επιλέξτε το **WHS link** και δείτε τις 2 σύντομες διαδρομές (δηλαδή με το μικρότερο θετικό slack) που δεν παραβιάζουν το χρόνο διατήρησης. Η σύντομη διαδρομή έχει καθυστέρηση μόλυνσης 0.294 ns, εκ των οποίων τα 0.212 ns αφορούν στη λογική (logic), ενώ τα 0.082 ns αφορούν στη δικτύωση (net). Η αβεβαιότητα του CLK εκτιμάται στα 0.000 ns. Τα επίπεδα λογικής (logic level) είναι 1.

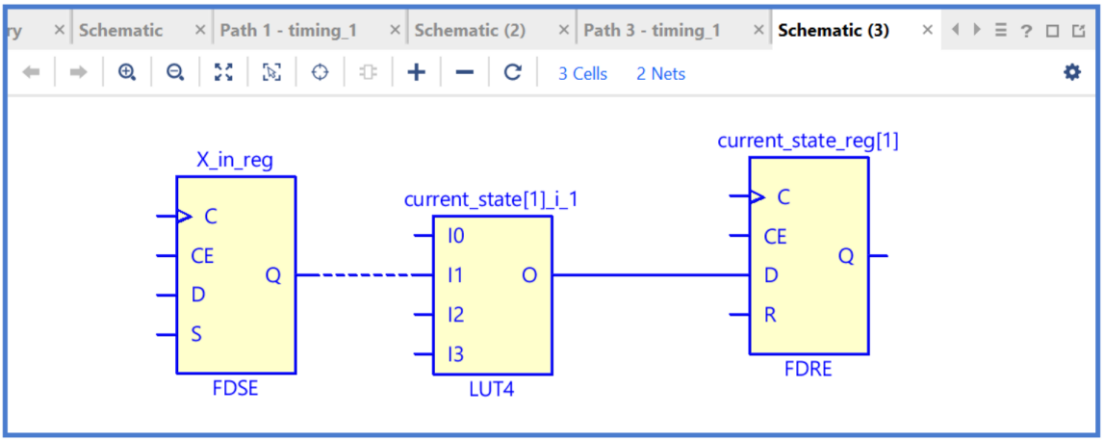


Εικόνα 183 – Μελέτη χειρότερης σύντομης διαδρομής στο implemented design model

- Επιλέξτε με διπλό κλικ το **Path 3**, ώστε να εμφανιστεί το παράθυρο *Path 3 – timing_1*. Η σύντομη διαδρομή περνάει από 1 μονάδα LUT4. Υπολογίστε το WHS slack (σε παρένθεση το αποτέλεσμα της σύνθεσης):
- η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Source Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK του καταχωρητή εισόδου) είναι **1.588** (0.735) ns,
- η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Data Path* (από την είσοδο CLK μέχρι την έξοδο Q του καταχωρητή εισόδου και μέσω του LUT4 μέχρι την είσοδο D του καταχωρητή κατάστασης 1) είναι **0.294** (0.451) ns,
- το **Arrival Time**, ως άθροισμα των ανωτέρων χρόνων, είναι **1.882** (1.186) ns,
- το **Required Time**, ως η καθυστέρηση μόλυνσης της σύντομης διαδρομής του *Destination Clock Path* (από την πηγή του CLK τη χρονική στιγμή 0.000 ns μέχρι την είσοδο CLK στον καταχωρητή κατάστασης 1) συν τον χρόνο διατήρησης, είναι **1.708** (0.979) ns.

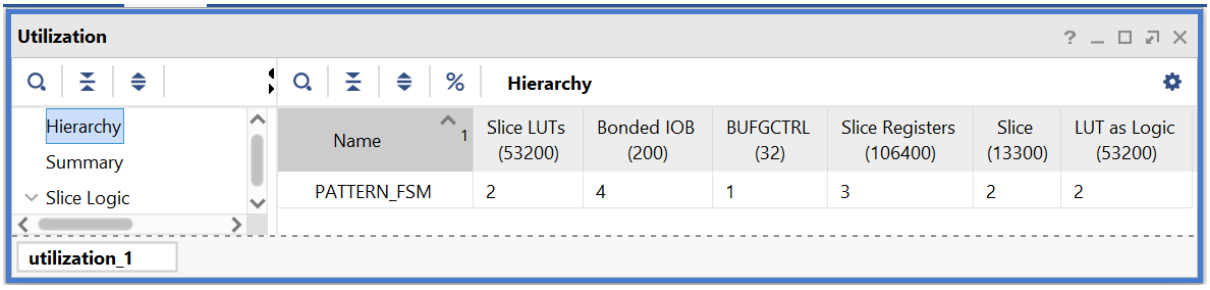
  WHS slack = Arrival Time – Required Time = 1.882 – 1.708 = 0.174 ns

- Επιλέξτε με δεξί κλικ στο **Path 3**, το **Schematic**. Μελετήστε το σχηματικό διάγραμμα της σύντομης διαδρομής της οντότητας **PATTERN_FSM**. (Έχει αλλάξει η διαδρομή).



Εικόνα 184 – Σχηματικό διάγραμμα χειρότερης σύντομης διαδρομής στο impl. design model

- Τέλος, στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, μέσα στο *Open Implemented Design* επιλέξετε το **Report Utilization** και μελετήστε τους πόρους που χρησιμοποιεί η οντότητα **PATTERN_FSM**. Πατήστε **OK**. (Μπορείτε να μελετήσετε και άλλα ενδιαφέροντα reports, εάν επιλέξετε το παράθυρο *Reports*).



Εικόνα 185 – Report utilization στο implemented design model

1.10.5 Εκτέλεση προσομοίωσης μετά την υλοποίηση (λογική και χρονική) στο VIVADO IDE για μηχανή πεπερασμένων καταστάσεων (FSM)

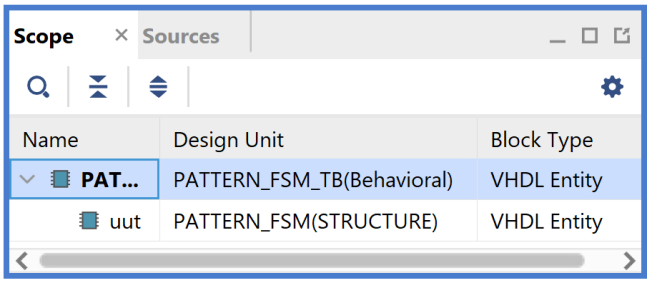
Η προσομοίωση μετά την υλοποίηση (λογική και χρονική) εκτελείται στην οντότητα **PATTERN_FSM_TB** του προγράμματος δοκιμών (testbench) που έχει ορισθεί ως η κορυφαία οντότητα της ιεραρχίας (**top**) των simulation resources. Κατά την προσομοίωση, η οντότητα **PATTERN_FSM_TB** καλεί το *UUT* της (που είναι το **implemented design model** της οντότητας **PATTERN_FSM**).

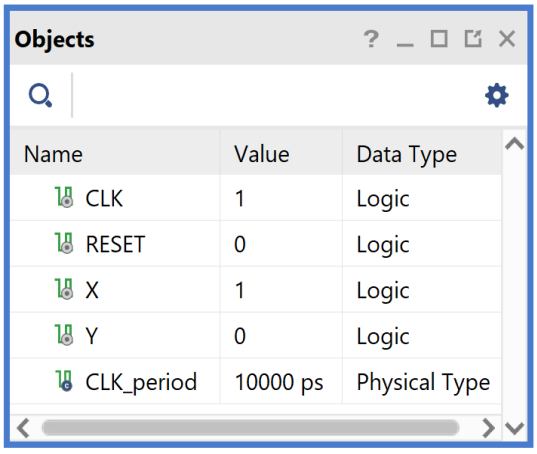
- Ενεργοποιήστε το waveform configuration file *PATTERN_FSM_TB_func_synth.wcfg* στην περίπτωση που είναι απενεργοποιημένο. Στο παράθυρο *Sources* κάντε δεξί κλικ και επιλέξτε **Enable File**. Απενεργοποιήστε όλα τα υπόλοιπα waveform configuration file. Στο παράθυρο *Sources* κάντε δεξί κλικ και επιλέξτε **Disable File**.
- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Run Simulation**, ώστε να εμφανιστούν όλες οι πιθανές προσομοιώσεις που υποστηρίζει το Vivado IDE. Επιλέξτε **Run Post–Implementation Functional Simulation**.
- Το πρόγραμμα δοκιμής (testbench) και όλες οι οντότητες του *UUT* θα γίνουν compiled και θα τρέξει το Vivado simulator (εφόσον βέβαια δεν υπάρχουν σφάλματα). Θα εμφανιστεί το παράθυρο *SIMULATION* που απαρτίζεται από 4 παράθυρα:

  Το παράθυρο *Scope*, όπου παρουσιάζεται η οντότητα **PATTERN_FSM_TB** και το **implemented design model** της οντότητας **PATTERN_FSM** (*UUT*) μετά την υλοποίηση, η οποία είναι *structural*.

  Το παράθυρο *Objects*, όπου εμφανίζονται τα σήματα top–level, δηλαδή οι είσοδοι και οι έξοδοι της οντότητας **PATTERN_FSM**, που είναι η κορυφαία οντότητα της ιεραρχίας του *UUT*, καθώς και η περίοδος του CLK (*CLK_period*). Οι τιμές αντιστοιχούν στις τιμές που έχει σταματήσει η προσομοίωση της οντότητας **PATTERN_FSM_TB** (*stop (2)*).

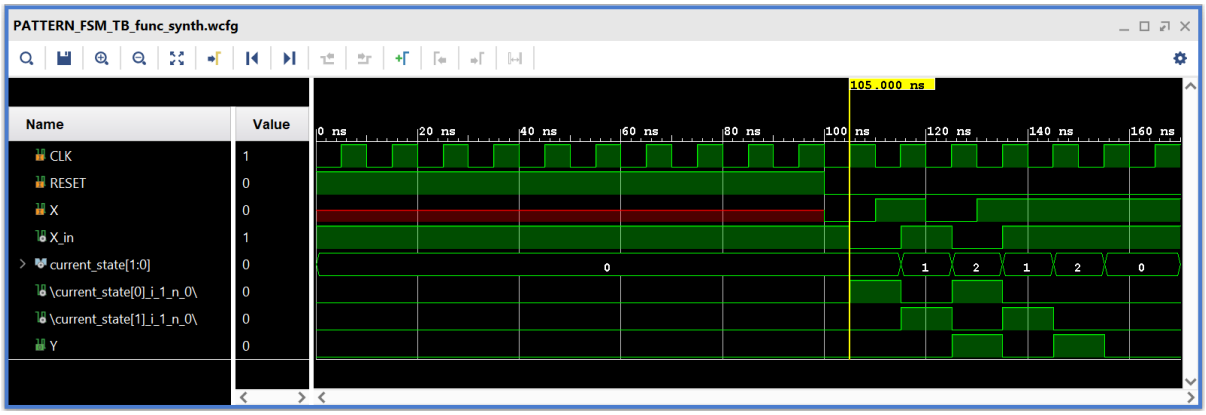
  Το παράθυρο *Tcl Console* με όλες τις διαδικασίες που εκτελούνται στο πλαίσιο της προσομοίωσης. Το Tcl Console καθαρίζει με την επιλογή *Clear*.





Εικόνα 186 - Παράθυρα Scope και Objects

Επιλέξτε το παράθυρο *PATTERN_FSM_TB_func_synth.wcfg*, που δημιουργήσατε για την προσομοίωση του **synthesized design model**, όπου συμπεριλαμβάνονται πέραν των top–level εισόδων/εξόδων και τα απαραίτητα **εσωτερικά σήματα current_state[0]_i_1_n_0** (αντιστοιχεί στο next_state[0]) και **current_ state[1]_i_1_n_0** (αντιστοιχεί στο next_state[1]). Βλέπετε ολόκληρο το διάγραμμα χρονισμού της λογικής προσομοίωσης μετά τη σύνθεση με κατάλληλο **zoom out** και επιλέγοντας το **zoom fit**. Τα διαγράμματα χρονισμού της λογικής προσομοίωσης μετά τη σύνθεση και μετά την υλοποίηση είναι ίδια.



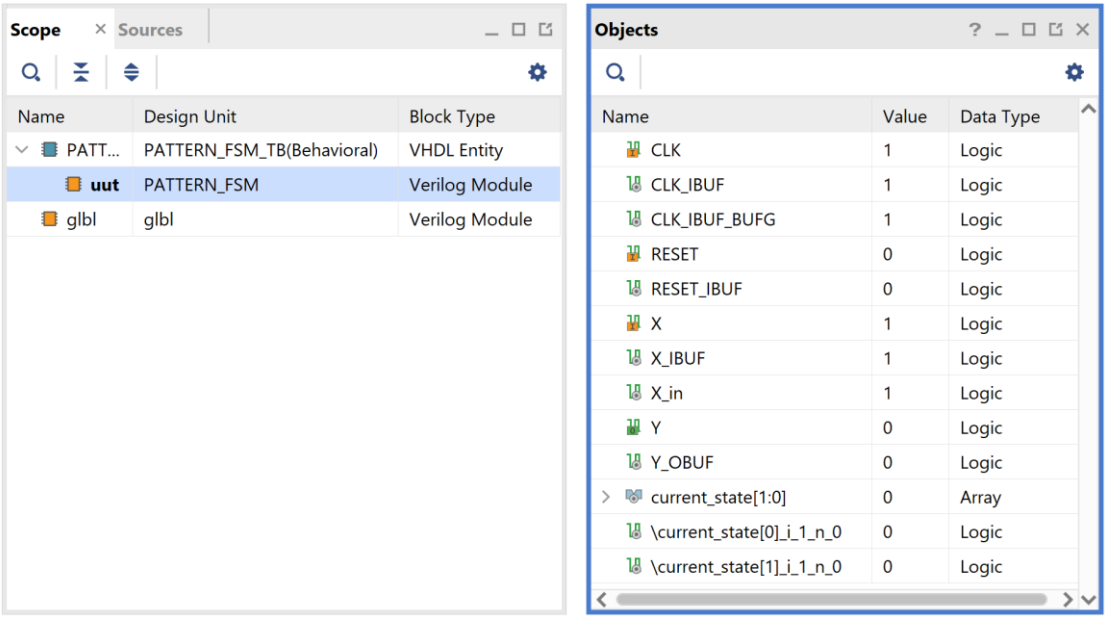
Εικόνα 187 – Διάγραμμα χρονισμού λογικής προσομοίωσης μετά την υλοποίηση

- Κλείστε τον simulator επιλέγοντας το κουμπί × πάνω δεξιά στο παράθυρο του *SIMULATION*. Στο παράθυρο *Confirm Close* που εμφανίζεται πατήστε **OK**.
- Στο κατακόρυφο παράθυρο αριστερά του *Flow Navigator*, επιλέξτε το **Run Simulation**, ώστε να εμφανιστούν όλες οι πιθανές προσομοιώσεις που υποστηρίζει το Vivado IDE. Επιλέξτε **Run Post–Implementation Timing Simulation**.
- Το πρόγραμμα δοκιμής (testbench) και όλες οι οντότητες του *UUT* θα γίνουν compiled και θα τρέξει το Vivado simulator (εφόσον βέβαια δεν υπάρχουν σφάλματα). Θα εμφανιστεί το παράθυρο *SIMULATION* που απαρτίζεται από 4 παράθυρα:

  Το παράθυρο Scope, όπου παρουσιάζεται η οντότητα **PATTERN_ FSM_TB** και το **implemented design model** της οντότητας **PATTERN_ FSM** (*UUT*) που προκύπτει μετά την υλοποίηση για χρονική προσομοίωση. Όλες οι οντότητες που απαρτίζουν πλέον το *UUT* είναι *Verilog Module!*

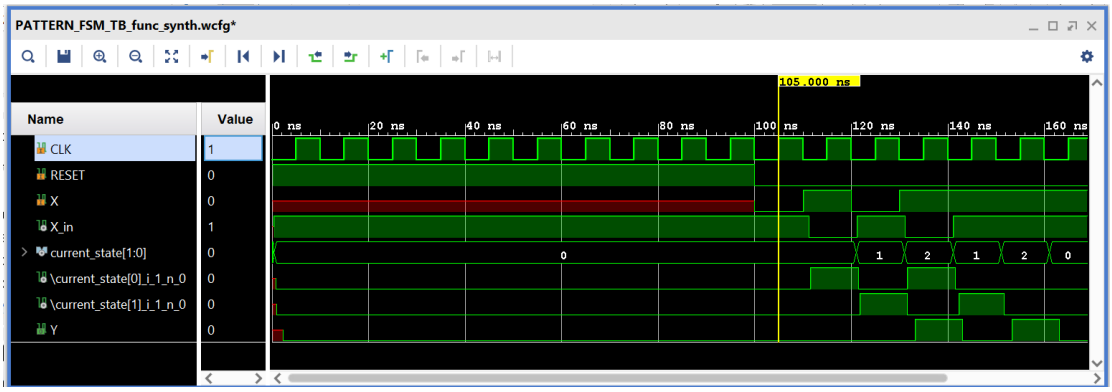
  Το παράθυρο *Objects*, όπου εμφανίζονται τα σήματα της οντότητας **PATTERN_ FSM** (*UUT*). Οι αρτηρίες (array) αναλύονται στα σήματα που τις απαρτίζουν. Οι τιμές αντιστοιχούν στις τιμές που έχει σταματήσει η προσομοίωση της οντότητας **PATTERN_ FSM_TB** (stop (2)).

  Το παράθυρο *Tcl Console* με όλες τις διαδικασίες που εκτελούνται στο πλαίσιο της προσομοίωσης. Το Tcl Console καθαρίζει με την επιλογή *Clear*.



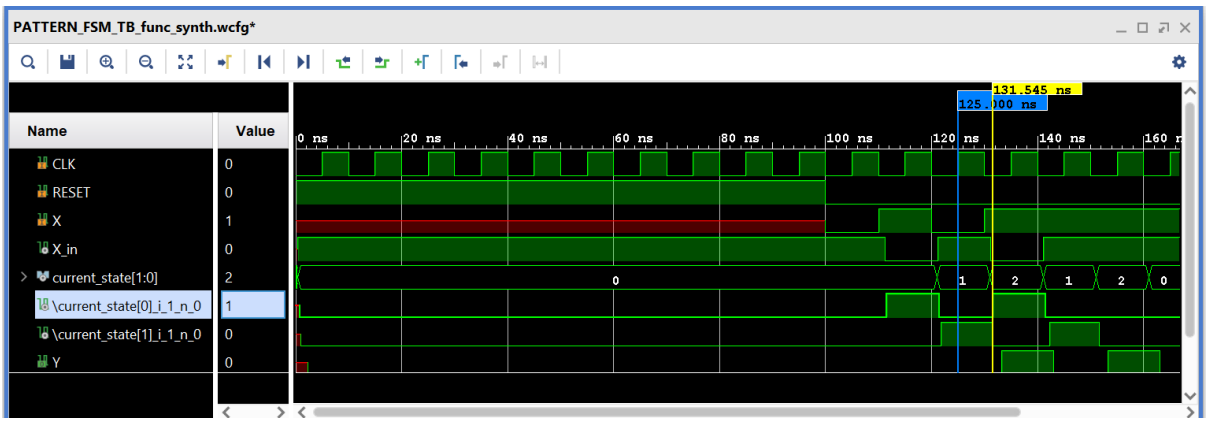
Εικόνα 188 - Παράθυρα Scope & Objects

Επιλέξτε το παράθυρο *PATTERN_ FSM_TB_func_synth.wcfg*. Βλέπετε ολόκληρο το διάγραμμα χρονισμού της χρονικής προσομοίωσης μετά τη σύνθεση με κατάλληλο **zoom out** ή επιλέγοντας το **zoom fit**. (Εάν λείπουν τα εσωτερικά σήματα, τα μεταφέρετε από το παράθυρο *Objects* και επαναλάβετε τη διαδικασία της προσομοίωσης από την αρχή με επιλογή του κουμπιού **Restart** και στη συνέχεια του κουμπιού **Run All**.)



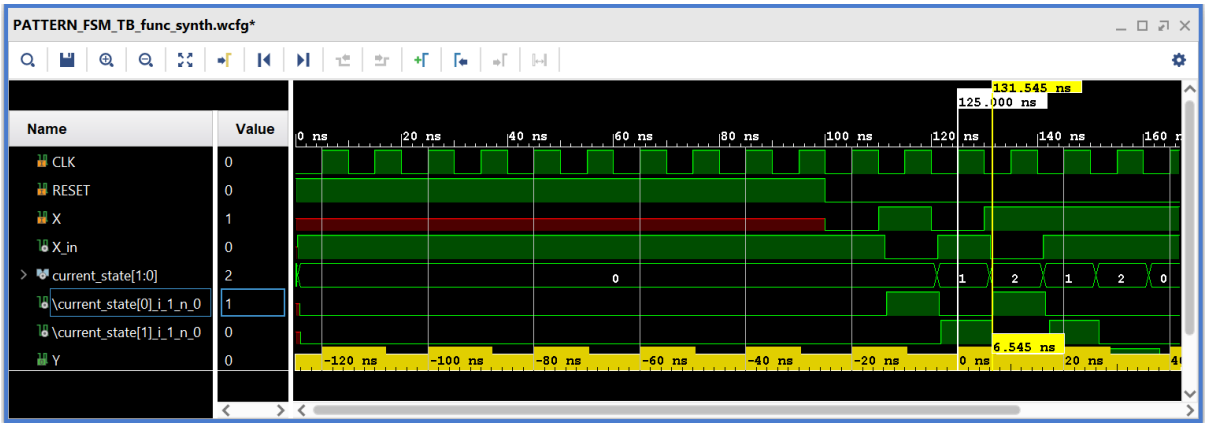
Εικόνα 189 – Διάγραμμα χρονισμού χρονικής προσομοίωσης μετά την υλοποίηση

- Συγκρίνετε τα διαγράμματα χρονισμού της χρονικής και της λογικής προσομοίωσης μετά την υλοποίηση. Είναι εμφανείς οι καθυστερήσεις διάδοσης στο πρώτο διάγραμμα χρονισμού.
- Υπολογίστε το **Arrival Time** ενός σήματος στο διάγραμμα χρονισμού της χρονικής προσομοίωσης μετά την υλοποίηση με τη χρήση των marker. Για παράδειγμα, βάλτε τον μπλε marker στην ανερχόμενη ακμή του **CLK** στα 125.000 ns και τον κίτρινο marker στην ανερχόμενη ακμή του εσωτερικού σήματος **current_state[0]_i_1_n_0** (που αντιστοιχεί στο next_state[0]) στα 131.545 ns. Απέχουν 6.545 ns. Συγκρίνετε με το **Arrival Time** που είχατε βρει κατά τη χρονική ανάλυση για την κρίσιμη διαδρομή, που ήταν 7.223 ns.



Εικόνα 190 – Επαλήθευση του arrival time (βήμα #1)

Με κλικ πάνω στον μπλε marker (γίνεται λευκός), ορίζουμε τη θέση του στα 0.000 ns και είναι πλέον εμφανές ότι ο κίτρινος marker απέχει 6.545 ns.



Εικόνα 191 – Επαλήθευση του arrival time (βήμα #2)

- Κλείστε τον simulator επιλέγοντας το κουμπί × πάνω δεξιά στο παράθυρο του *SIMULATION*. Στο παράθυρο *Confirm Close* που εμφανίζεται πατήστε **OK**. Εάν επιθυμείτε να μη σώσετε ένα επιπλέον waveform configuration, στο παράθυρο *Save Waveform Configuration* επιλέξτε **Discard**.

Η διαδικασία που ήδη περιγράψαμε στα Βήματα 5–4 και 5–5 της «**Υλοποίησης στη τεχνολογία FPGA και προσομοίωση (λογική, χρονική)**» εφαρμόζεται παρομοίως σε κάθε πιθανή μηχανή πεπερασμένων καταστάσεων (FSM).

2. Μέρος Β΄: Μεθοδολογία σχεδίασης επεξεργαστή ενός κύκλου

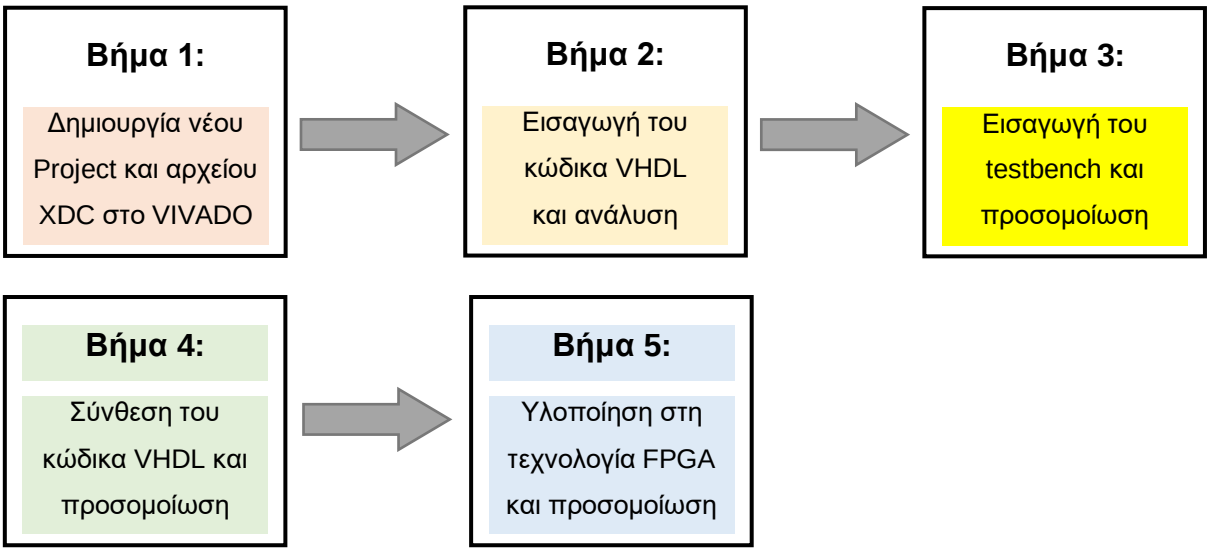
2.1 Εισαγωγικά στοιχεία

Στον παρόντα εργαστηριακό οδηγό θα περιγράψουμε αναλυτικά τη μεθοδολογία σχεδίασης της μικροαρχιτεκτονικής ενός απλοποιημένου **επεξεργαστή ενός κύκλου** αρχιτεκτονικής **ARM** σε τεχνολογία **FPGA** με τη χρήση του εργαλείου **Vivado IDE** (Integrated Development Environment) της Xilinx.

Αρχικά, αναλύουμε τις απαιτήσεις του επεξεργαστή που απορρέουν από την εφαρμογή της μικροαρχιτεκτονικής ARM, από τα 5 βήματα εκτέλεσης της εντολής στη μικροαρχιτεκτονική ARM (με τις αντίστοιχες λειτουργίες που εκτελούνται ανά βήμα) και από το σύνολο εντολών που πρόκειται να υλοποιηθεί. Στη συνέχεια, προσδιορίζουμε τα ψηφιακά δομικά στοιχεία της διαδρομής δεδομένων που απαιτούνται σε κάθε βήμα εκτέλεσης της εντολής και προχωράμε στη σχεδίαση της διαδρομής δεδομένων του επεξεργαστή ενός κύκλου ακολουθώντας την ιεραρχική προσέγγιση bottom–up. Μετά, προσδιορίζουμε τα ψηφιακά δομικά στοιχεία της μονάδας ελέγχου και προχωράμε στη σχεδίαση της. Τέλος, στο ανώτερο επίπεδο (top–level) του επεξεργαστή, τοποθετούμε μαζί τη διαδρομή δεδομένων και τη μονάδα ελέγχου, επαληθεύουμε την ορθή σχεδίαση με κατάλληλο πρόγραμμα σε συμβολική γλώσσα ARM, βρίσκουμε τη μέγιστη συχνότητα λειτουργίας και αναλύουμε τις επιδόσεις του επεξεργαστή.

2.2 Γενική ροή σχεδίασης του επεξεργαστή

Ακολουθούμε τα Βήματα 1–5 σε όλα τα ιεραρχικά επίπεδα της σχεδίασης του επεξεργαστή: τα δομικά στοιχεία της διαδρομής δεδομένων, τη μονάδα ελέγχου, το ανώτερο επίπεδο του επεξεργαστή.



2.3 Βήμα 1: Ανάλυση των απαιτήσεων του επεξεργαστή

2.3.1 Απαιτήσεις που απορρέουν από την εφαρμογή της μικροαρχιτεκτονικής ARM

- Κάθε μικροαρχιτεκτονική ARM πρέπει να υλοποιεί όλους τους **αρχιτεκτονικούς καταχωρητές**, που είναι οι εξής:
- **αρχείο καταχωρητών** (register file, RF) με 16 καταχωρητές **R0–R15** των 32 bit,
- **μετρητής προγράμματος** (program counter, PC) που θεωρείται ως ο καταχωρητής **R15** του αρχείου καταχωρητών,
- **καταχωρητής κατάστασης** (status register, SR) των 4 bit για την αποθήκευση των σημαιών **N**, **Z**, **C**, **V**.
- Στη μικροαρχιτεκτονική ARM η μνήμη είναι προσπελάσιμη **ανά byte** με λέξεις εντολών/δεδομένων των 32 bit (4 byte). Η διεύθυνση των 32 bit της λέξης ταυτίζεται με τη διεύθυνση του **λιγότερου σημαντικού byte** (least significant byte, LSB) της λέξης και είναι **ευθυγραμμισμένη** (σε πολλαπλάσια του 4). Το περισσότερο σημαντικό byte (most significant byte, **MSB**) της λέξης βρίσκεται στα **αριστερά**, ενώ το λιγότερο σημαντικό byte (least significant byte, **LSB**) στα **δεξιά**.

  Για τον προσδιορισμό της διεύθυνσης της λέξης δεδομένων στη μνήμη χρησιμοποιείται ο τρόπος διευθυνσιοδότησης **μνήμης με σχετική απόσταση** (offset addressing) που χρησιμοποιεί έναν καταχωρητή βάσης (base register) που περιέχει τη διεύθυνση βάσης, και μια σχετική απόσταση (offset) ως **μη προσημασμένος ακέραιος των 12 bit** που είναι άμεσα αποθηκευμένος στην ίδια την εντολή. Για να σχηματιστεί η διεύθυνση μνήμης, προσθέτουμε/ αφαιρούμε τη σχετική απόσταση (που μπορεί να είναι 0) στο/από το περιεχόμενο του καταχωρητή βάσης.
- Η μικροαρχιτεκτονική ARM υποστηρίζει εντολές που **ενεργοποιούν σημαίες συνθήκης** (*condition flags*) ανάλογα με το αποτέλεσμα μίας πράξης στη μονάδα ALU. Οι εντολές, που έπονται της εντολής που ενεργοποιεί σημαίες συνθήκης, εκτελούνται **υπό συνθήκη**, ανάλογα με τις τιμές των σημαιών.
- Η αρχιτεκτονική ARM υποστηρίζει **εντολές υπό συνθήκη**. Το μνημονικό της εντολής ακολουθείται από ένα μνημονικό συνθήκης, το οποίο υποδεικνύει τη συνθήκη (CondEx) που πρέπει να ικανοποιείται για να εκτελεσθεί η συγκεκριμένη εντολή. Το μνημονικό συνθήκης αποθηκεύεται στο πεδίο συνθήκης (cond) της εντολής μεγέθους 4 bit. Στον επόμενο πίνακα φαίνονται τα **μνημονικά συνθήκης** με τις **εξισώσεις Boole** των σημαιών που τις ικανοποιούν.

Πίνακας 1 – Μνημονικά συνθήκης με εξισώσεις Boole

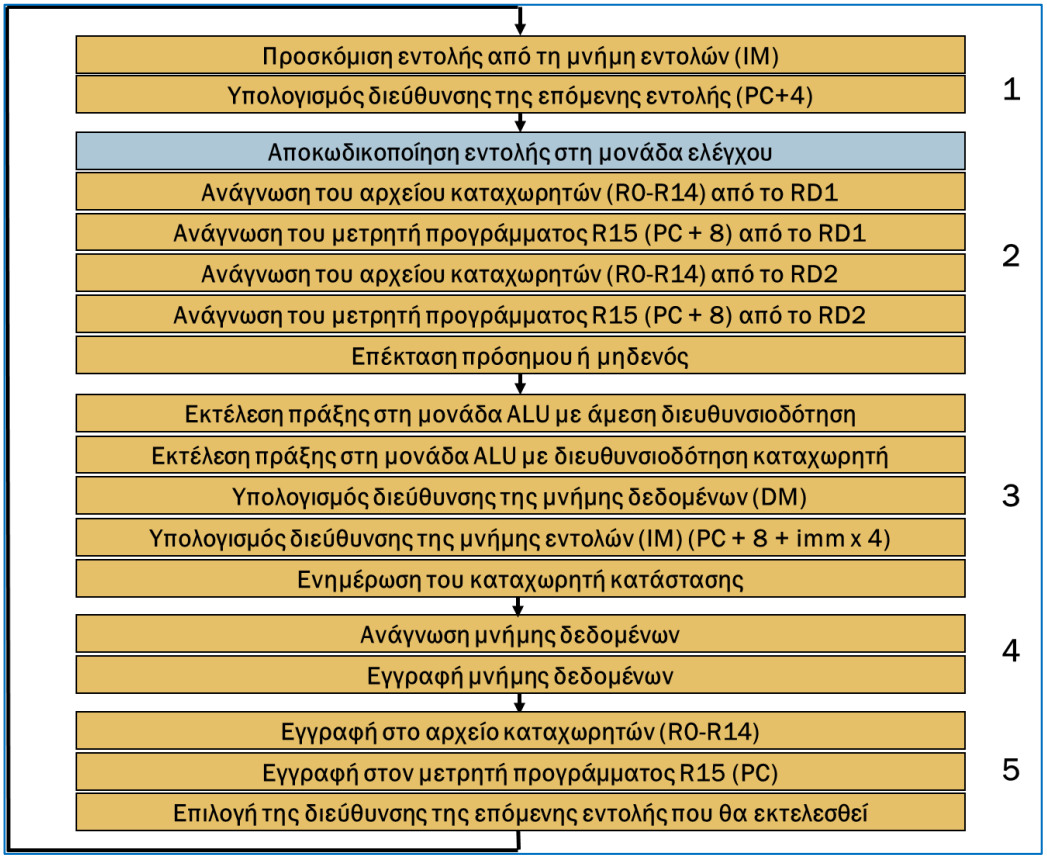
$cond_{3:0}$	Μνημονικό	Όνομα	CondEx
0000	EQ	Equal	Z
0001	NE	Not equal	$\bar{Z}$
0010	CS/HS	Carry set / unsigned higher or same	C
0011	CC/LO	Carry clear / unsigned lower	$\bar{C}$
0100	MI	Minus / negative	N
0101	PL	Plus / positive or zero	$\bar{N}$
0110	VS	Overflow / overflow set	V
0111	VC	No overflow / overflow clear	$\bar{V}$
1000	HI	Unsigned higher	$\bar{Z}C$
1001	LS	Unsigned lower or same	$Z+\bar{C}$
1010	GE	Signed greater or equal	$\overline{N \oplus V}$
1011	LT	Signed less	$N \oplus V$
1100	GT	Signed greater	$\bar{Z}\,\overline{N \oplus V}$
1101	LE	Signed less or equal	$Z+(N \oplus V)$
1110	AL (ή none)	Always / unconditional	1
1111	none	For unconditional instructions	1

- Η μικροαρχιτεκτονική ARM του επεξεργαστή ενός κύκλου απαρτίζεται από δύο διακριτές μονάδες που αλληλοεπιδρούν μεταξύ τους:
- τη **διαδρομή δεδομένων** (datapath) των 32 bit, που εκτελεί λειτουργίες με λέξεις δεδομένων και περιέχει υπομονάδες όπως: διακριτές μνήμες εντολών (ROM 64 x 32) και δεδομένων (RAM 32 x 32) μεγέθους 32 bit (αρχιτεκτονική Harvard), αρχιτεκτονικούς καταχωρητές, μονάδα ALU για την εκτέλεση των πράξεων που απορρέουν από το σύνολο εντολών που πρόκειται να υλοποιηθεί, πολυπλέκτες για τον έλεγχο της ροής δεδομένων, επιπλέον συνδυαστική λογική (2 αθροιστές αύξησης κατά 4, μονάδα επέκτασης προσήμου/μηδενός),
- τη **μονάδα ελέγχου** (control unit), ως συνδυαστική λογική, που αποκωδικοποιεί την εντολή και για κάθε εντολή υπό συνθήκη παράγει τα κατάλληλα σήματα ελέγχου των υπομονάδων της διαδρομής δεδομένων, λαμβάνοντας υπόψη τις τιμές των σημαιών (N, Z, C, V).
- Στη μικροαρχιτεκτονική ARM του επεξεργαστή ενός κύκλου ολόκληρη η εντολή εκτελείται σε **έναν κύκλο ρολογιού** (CLK), όπου η περίοδος του CLK προσδιορίζεται από την εντολή **LDR** που έχει τη μεγαλύτερη καθυστέρηση διάδοσης από όλες τις εντολές του επεξεργαστή.

2.3.2 Τα 5 βήματα εκτέλεσης της εντολής στη μικροαρχιτεκτονική ARM

1. Προσκόμιση εντολής και υπολογισμός της επόμενης διεύθυνσης (PC+4)
2. Αποκωδικοποίηση εντολής, ανάγνωση 2 καταχωρητών από το αρχείο καταχωρητών (συμπεριλαμβάνεται και ο PC) και επέκταση προσήμου/μηδενός.
3. Εκτέλεση πράξεων στη μονάδα ALU
4. Ανάγνωση ή εγγραφή στη μνήμη δεδομένων
5. Ετεροχρονισμένη εγγραφή ενός καταχωρητή στο αρχείο καταχωρητών (συμπεριλαμβάνεται και ο μετρητής προγράμματος) και επιλογή της διεύθυνσης της επόμενης εντολής που πρόκειται να εκτελεσθεί.

Στο επόμενο σχήμα φαίνονται οι 18 λειτουργίες που εκτελούνται ανά βήμα.



Εικόνα 192 – Βήματα και λειτουργίες κατά την εκτέλεση μιας εντολής ARM

2.3.3 Το σύνολο εντολών που πρόκειται να υλοποιηθεί

- Η εντολή **LDR**

  Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

  - LDR Rd, [Rn, #imm12]; Rd = DM[Rn + #imm12] (U = 1)
  - LDR Rd, [Rn, #–imm12]; Rd = DM[Rn – #imm12] (U = 0)

  Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

  - ο τελεστέος στον καταχωρητή προέλευσης Rn,
  - ο τελεστέος στον καταχωρητή προορισμού Rd,
  - ο μη προσημασμένος άμεσος τελεστέος των 12 bit με επέκταση μηδενός στα 32 bit.

  Πράξεις που εκτελούνται στη μονάδα ALU:

  - πρόσθεση και αφαίρεση.

  Μορφή εντολής ($\bar{I}$ = 0, P = 1, W = 0, B = 0, L = 1):

- Η εντολή **STR**

  Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

  - STR Rd, [Rn, #imm12]; DM[Rn + #imm12] = Rd (U = 1)
  - STR Rd, [Rn, #–imm12]; DM[Rn – #imm12] = Rd (U = 0)

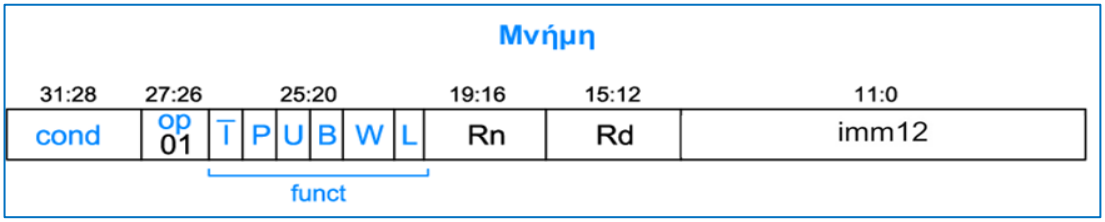
  Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

  - ο τελεστέος στον καταχωρητή προέλευσης Rn,
  - ο τελεστέος στον καταχωρητή προέλευσης Rd,
  - ο μη προσημασμένος άμεσος τελεστέος των 12 bit με επέκταση μηδενός στα 32 bit.

  Πράξεις που εκτελούνται στη μονάδα ALU:

  - πρόσθεση και αφαίρεση.

  Μορφή εντολής ($\bar{I}$ = 0, P = 1, W = 0, B = 0, L = 0):



- Οι εντολές επεξεργασίας δεδομένων με άμεση διευθυνσιοδότηση **ALU(S)–I**

  Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

  - ALU(S) Rd, Rn, #imm8; Rd = Rn +/–/and/or/xor imm8
  - ενημέρωση σημαιών SR = (N, Z, C, V) (S = 1)

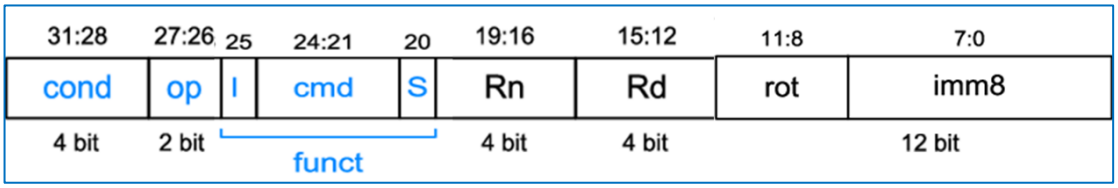
  Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

  - ο τελεστέος στον καταχωρητή προέλευσης Rn,
  - ο τελεστέος στον καταχωρητή προορισμού Rd,
  - ο μη προσημασμένος άμεσος τελεστέος των 12 bit (rot = 0) με επέκταση μηδενός στα 32 bit.

  Πράξεις που εκτελούνται στη μονάδα ALU:

  - αριθμητικές πράξεις (πρόσθεση, αφαίρεση),
  - λογικές πράξεις (and, or, xor)

  Μορφή εντολής (I = 1, cmd = 0100 (+), 0010 (–), 0000 (and), 1100 (or), 0001 (xor)):



- Οι εντολές επεξεργασίας δεδομένων με διευθυνσιοδότηση καταχωρητή **ALU(S)–R**

  Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

  - ALU(S) Rd, Rn, Rm; Rd = Rn +/–/and/or/xor Rm
  - ενημέρωση σημαιών SR = (N, Z, C, V) (S = 1)

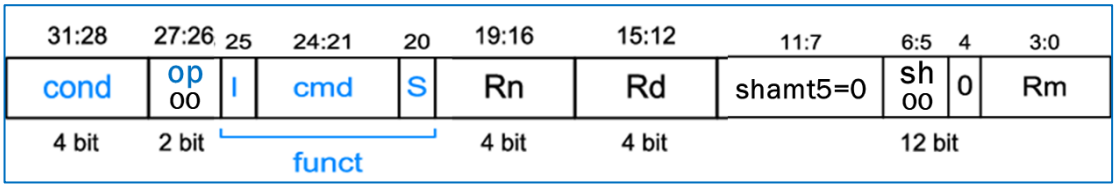
  Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

  - ο τελεστέος στον καταχωρητή προέλευσης Rn,
  - ο τελεστέος στον καταχωρητή προέλευσης Rm,
  - ο τελεστέος στον καταχωρητή προορισμού Rd.

  Πράξεις που εκτελούνται στη μονάδα ALU:

  - αριθμητικές πράξεις (πρόσθεση, αφαίρεση),
  - λογικές πράξεις (and, or, xor)

  Μορφή εντολής (I = 0, cmd = 0100 (+), 0010 (–), 0000 (and), 1100 (or), 0001 (xor)):



- Η εντολή **CMP**

  Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

  - CMP Rn, #imm8; ενημέρωση σημαιών για Rn – imm8 (I = 1)
  - CMP Rn, Rm; ενημέρωση σημαιών για Rn – Rm (I = 0)

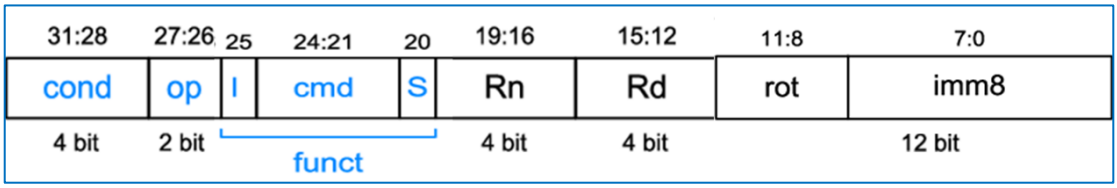
  Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

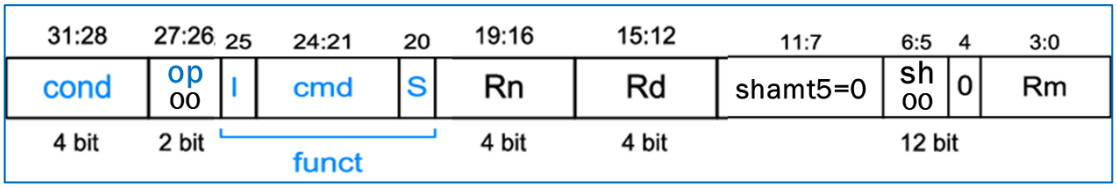
  - ο τελεστέος στον καταχωρητή προέλευσης Rn,
  - ο τελεστέος στον καταχωρητή προέλευσης Rm (I = 0),
  - ο μη προσημασμένος άμεσος τελεστέος των 12 bit (rot = 0) με επέκταση μηδενός στα 32 bit (I = 1).

  Πράξεις που εκτελούνται στη μονάδα ALU:

  - αφαίρεση.

  Μορφές εντολής (S = 1, cmd = 1010):





- Οι εντολές μεταφοράς δεδομένων **MOV**, **NOP**, **MVN** (μόνο για S = 0)

  Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

  - MOV Rd, #imm8; Rd = imm8 (I = 1)
  - MOV Rd, Rm; Rd = Rm (I = 0)
  - NOP = MOV R0, R0; R0 = R0 (I = 0)
  - MVN Rd, #imm8; Rd = not imm8 (I = 1)
  - MVN Rd, Rm; Rd = not Rm (I = 0)

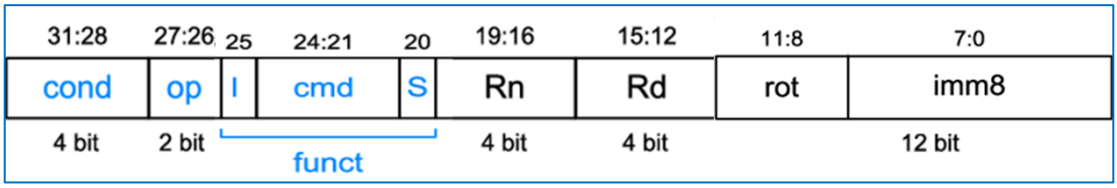
  Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

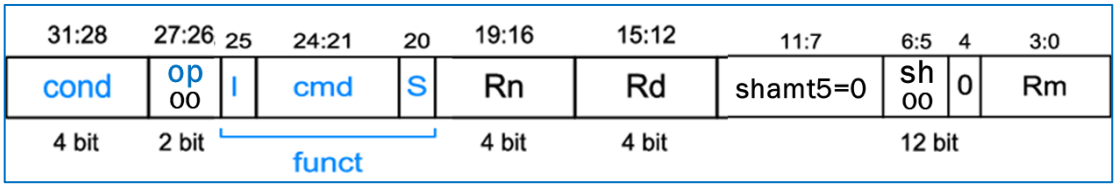
  - ο τελεστέος στον καταχωρητή προέλευσης Rm (I = 0),
  - ο τελεστέος στον καταχωρητή προορισμού Rd,
  - ο μη προσημασμένος άμεσος τελεστέος των 12 bit (rot = 0) με επέκταση μηδενός στα 32 bit (I = 1).

  Πράξεις που εκτελούνται στη μονάδα ALU:

  - μεταφορά (MOV, NOP),
  - αντιστροφή και μεταφορά (MVN).

  Μορφές εντολής (S = 0, cmd = 1101 (MOV, NOP), 1111 (MVN), Rn = 0):





- Οι εντολές ολίσθησης **LSL**, **LSR**, **ASR**, **ROR** (μόνο για S = 0)

  Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

  - LSL Rd, Rm, #shamt5; Rd = Rm LSL by #shamt5
  - LSR Rd, Rm, #shamt5; Rd = Rm LSR by #shamt5
  - ASR Rd, Rm, #shamt5; Rd = Rm ASR by #shamt5
  - ROR Rd, Rm, #shamt5; Rd = Rm ROR by #shamt5

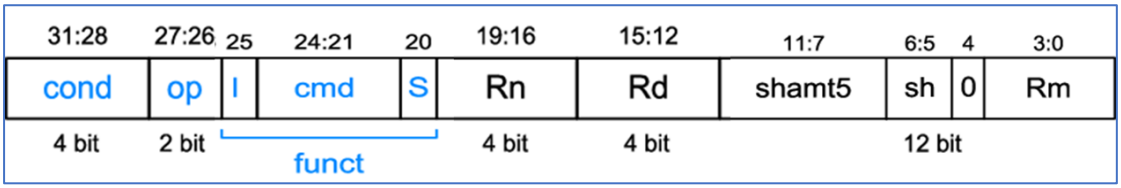
  Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

  - ο τελεστέος στον καταχωρητή προέλευσης Rm (I = 0),
  - ο τελεστέος στον καταχωρητή προορισμού Rd,
  - η ποσότητα ολίσθησης shamt5 των 5 bits.

  Πράξεις που εκτελούνται στη μονάδα ALU:

  - ολίσθησης (LSL, LSR, ASR),
  - περιστροφής (ROR).

Μορφή εντ. (S = 0/1, cmd = 1101, sh = 00 (LSL), 01 (LSR), 10 (ASR), 11 (ROR), Rn = 0):



- Οι εντολές διακλάδωσης **B, BL**

  Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

  - B label ; PC = BTA = PC + 8 + imm24 x 4 (L = 0)
  - BL label ; PC = BTA = PC + 8 + imm24 x 4; R14 = LR = PC + 4 (L = 1)

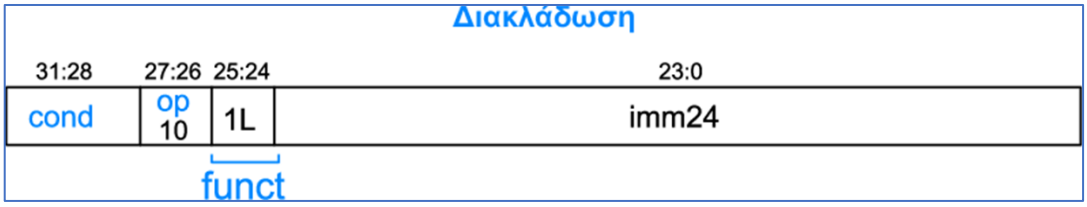
  Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

  - ο τελεστέος στον καταχωρητή προορισμού R14 (link register, LR),
  - ο προσημασμένος άμεσος τελεστέος των 24 bit, πολλαπλασιασμένος επί 4 και με επέκταση προσήμου στα 32 bit.

  Πράξεις που εκτελούνται στη μονάδα ALU:

  - πρόσθεση.

Μορφή εντολής (L = 0 (B), 1 (BL)):



2.4 Βήμα 2: Σχεδίαση των ψηφιακών δομικών στοιχείων της διαδρομής δεδομένων

Στο κατώτερο ιεραρχικά επίπεδο της σχεδίασης περιγράφεται στη γλώσσα VHDL η συμπεριφορά των ψηφιακών δομικών στοιχείων που απαιτούνται σε κάθε βήμα εκτέλεσης της εντολής. Το μέγεθος των αρτηριών των λειτουργικών μονάδων και των καταχωρητών, καθώς και το μέγεθος των μνημών παραμετροποιείται με τη δήλωση της εντολής generic.

2.4.1 Προσκόμιση εντολής και υπολογισμός της επόμενης διεύθυνσης (PC+4)

- Παραμετροποιημένος καταχωρητής των N bit (με αρχική τιμή N = 32) με RESET και WE (σταθερά στο 1 στην περίπτωση του επεξεργαστή ενός κύκλου) για να χρησιμοποιηθεί ως μετρητής προγράμματος (**program counter, PC**).
- Παραμετροποιημένη διάταξη μνήμης ROM με 2^N λέξεις μεγέθους M bit (με αρχικές τιμές N = 6 και M = 32) για να χρησιμοποιηθεί ως μνήμη εντολών (**instruction memory, IM**). Προσοχή! A[N–1:0] = PC[N+1:2].
- Παραμετροποιημένος αθροιστής κατά 4 με έξοδο PCPlus4 για τον υπολογισμό της διεύθυνσης της αμέσως επόμενης εντολής PC + 4 (**incrementer by 4, INC4**).

2.4.2 Ανάγνωση 2 καταχωρητών από το αρχείο καταχωρητών (συμπεριλαμβάνεται και ο μετρητής προγράμματος) και επέκταση προσήμου/μηδενός

- Παραμετροποιημένο αρχείο καταχωρητών των 2^N καταχωρητών μεγέθους M bit (με αρχικές τιμές N = 4 και M = 32) με δυνατότητα ασύγχρονου διαβάσματος δύο καταχωρητών και σύγχρονης εγγραφής ενός καταχωρητή, όταν RegWrite = 1 (**register file, RF**). Προσοχή! Ο καταχωρητής R15 είναι ο μετρητής προγράμματος PC και δεν υλοποιείται εντός του αρχείου καταχωρητών μαζί με τους υπόλοιπους καταχωρητές R0 έως R14. Όταν διαβάζεται ο R15 επιστρέφει την τιμή PC + 8. Δεν ορίζεται εγγραφή του R15 που να αφορά στο αρχείο καταχωρητών, αφού υλοποιείται ξεχωριστά.
- Παραμετροποιημένος πολυπλέκτης 2 σε 1 των N bit (με αρχική τιμή N = 32) για να χρησιμοποιηθεί στον προσδιορισμό της διεύθυνσης του πρώτου καταχωρητή προέλευσης του αρχείου καταχωρητών (**mux2to1**). Επιλέγει μεταξύ του πεδίου Rn της εντολής (RegSrc[0] = 0) και της σταθερής τιμής 15 (RegSrc[0] = 1). Η έξοδος συνδέεται στην είσοδο A1 του αρχείου καταχωρητών.
- Παραμετροποιημένος πολυπλέκτης 2 σε 1 των N bit (με αρχική τιμή N = 32) για να χρησιμοποιηθεί στον προσδιορισμό της διεύθυνσης του δεύτερου καταχωρητή προέλευσης του αρχείου καταχωρητών (**mux2to1**). Επιλέγει μεταξύ του πεδίου Rm της εντολής (RegSrc[1] = 0) και του πεδίου Rd της εντολής (RegSrc[1] = 1). Η έξοδος συνδέεται στην είσοδο A2 του αρχείου καταχωρητών.
- Παραμετροποιημένος πολυπλέκτης 2 σε 1 των N bit (με αρχική τιμή N = 32) για να χρησιμοποιηθεί στον προσδιορισμό της διεύθυνσης του καταχωρητή προορισμού του αρχείου καταχωρητών (**mux2to1**). Επιλέγει μεταξύ του πεδίου Rd της εντολής (RegSrc[2] = 0) και της σταθερής τιμής 14 (RegSrc[2] = 1). Η έξοδος συνδέεται στην είσοδο A3 του αρχείου καταχωρητών.

- Παραμετροποιημένος αθροιστής κατά 4 με είσοδο PCPlus4 και έξοδο PCPlus8 για τον υπολογισμό της διεύθυνσης PC + 8 (**incrementer by 4, INC4**). Η έξοδος συνδέεται στην θύρα R15 του αρχείου καταχωρητών.
- Μονάδα επέκτασης μηδενός από τα 12 bit στα 32 Bit (για ImmSrc = 0) ή προσήμου από τα 26 bit (άμεσος τελεστέος των 24 bit πολλαπλασιασμένος επί 4) στα 32 bit (για ImmSrc = 1) (**Extend**).

2.4.3 Εκτέλεση πράξεων στη μονάδα ALU

- Παραμετροποιημένος πολυπλέκτης 2 σε 1 των N bit (με αρχική τιμή N = 32) για να χρησιμοποιηθεί στον προσδιορισμό των δεδομένων της εισόδου SrcB της μονάδας ALU (**mux2to1**). Επιλέγει μεταξύ της θύρας ανάγνωσης A2/RD2 του αρχείου καταχωρητών (ALUSrc = 0) και της εξόδου ExtImm της μονάδας επέκτασης προσήμου–μηδενός (ALUSrc = 1). Η έξοδος συνδέεται στην είσοδο SrcB της μονάδας ALU.
- Παραμετροποιημένη μονάδα ALU μεγέθους N bit (με αρχική τιμή N = 32) (**ALU**). Η μονάδα ALU έχει δύο εισόδους SrcA και SrcB, μία έξοδο ALUResult και συμπε–ριλαμβάνει αθροιστή/αφαιρέτη των N bit, μονάδα λογικών πράξεων των N bit, μονάδα μεταφοράς δεδομένων των N bit, μονάδα ολίσθησης των N bit, πύλη NOR των N bit για τη σημαία Z και τους απαραίτητους πολυπλέκτες 2 σε 1 των N bit. Η μονάδα ALU παράγει τις τιμές των σημαιών N, Z, C, V. Οι σημαίες C και V ενεργοποιούνται (παίρνουν τιμή 1) μόνο όταν εκτελούνται αριθμητικές πράξεις. Το μέγεθος του σήματος ελέγχου ALUControl εξαρτάται από το πλήθος των πράξεων που εκτελούνται στη μονάδα ALU.
- Παραμετροποιημένος καταχωρητής των N bit (με αρχική τιμή N = 4) με RESET και WE (FlagsWrite) για να χρησιμοποιηθεί ως καταχωρητής καταστάσεων (**status register, SR**). Χρησιμοποιείται για την αποθήκευση των τιμών σημαιών N, Z, C, V, μόνο όταν το πεδίο S των εντολών επεξεργασίας δεδομένων έχει την τιμή 1.

2.4.4 Ανάγνωση ή εγγραφή στη μνήμη δεδομένων

- Παραμετροποιημένη διάταξη μνήμης RAM με 2^N λέξεις μεγέθους M bit (με αρχικές τιμές N = 5 και M = 32) για να χρησιμοποιηθεί ως μνήμη δεδομένων (**data memory, DM**). Προσοχή! A[N–1:0] = ALUResult[N+1:2]. Λόγω του μικρού μεγέθους υλοποιείται ως distributed RAM. Το διάβασμα γίνεται ασύγχρονα, ενώ η εγγραφή γίνεται σύγχρονα, όταν MemWrite = 1. Δεν πρέπει να έχει καταχωρητή εξόδου.

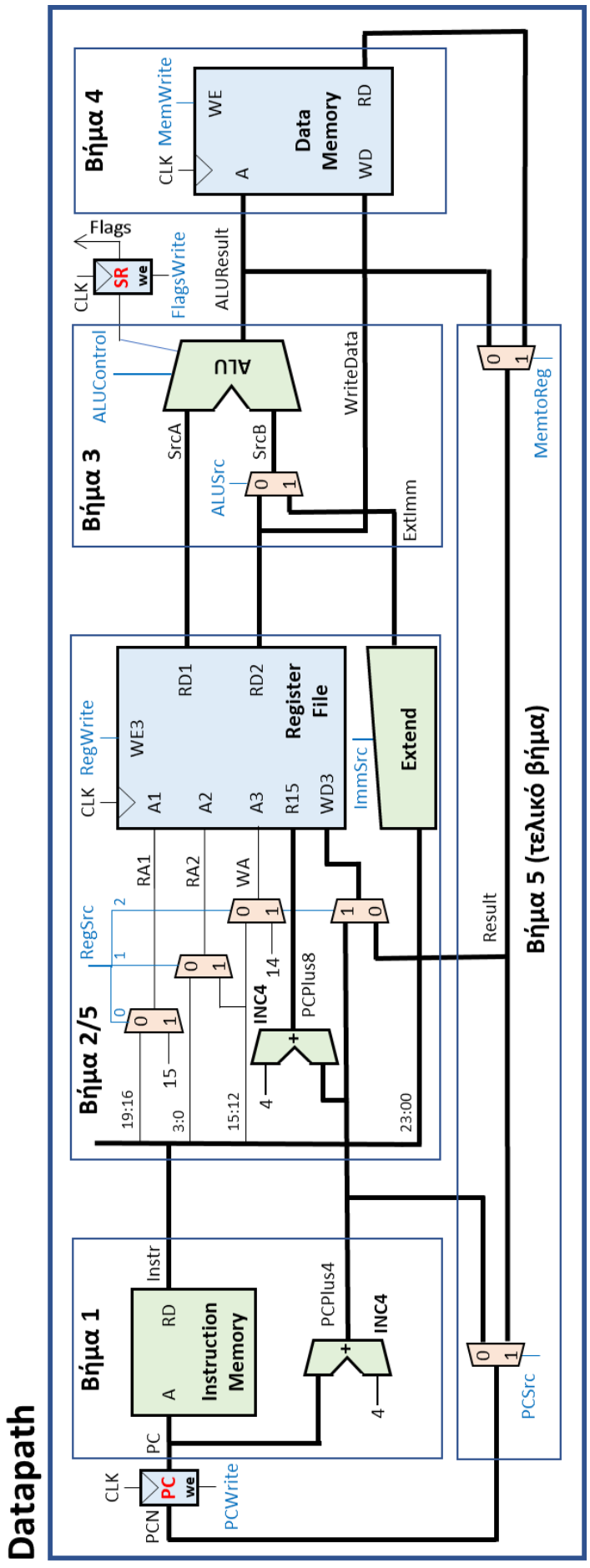
2.4.5 Ετεροχρονισμένη εγγραφή ενός καταχωρητή στο αρχείο καταχωρητών (συμπεριλαμβάνεται και ο μετρητής προγράμματος) και επιλογή της διεύθυνσης της επόμενης εντολής που πρόκειται να εκτελεσθεί.

- Παραμετροποιημένος πολυπλέκτης 2 σε 1 των N bit (με αρχική τιμή N = 32) για να χρησιμοποιηθεί στην επιλογή των δεδομένων που εγγράφονται ετεροχρονισμένα στον καταχωρητή προορισμού Rd (R0–R15) του αρχείου καταχωρητών (**mux2to1**). Επιλέγει μεταξύ του αποτελέσματος ALUResult της πράξης που εκτελείται στη μονάδα ALU (MemtoReg = 0) και της θύρας ανάγνωσης RD της μνήμης δεδομένων (MemtoReg = 1). Η έξοδος Result χρησιμοποιείται ως είσοδος δεδομένων άλλων πολυπλεκτών 2 σε 1.

- Παραμετροποιημένος πολυπλέκτης 2 σε 1 των N bit (με αρχική τιμή N = 32) για να χρησιμοποιηθεί στην επιλογή των δεδομένων που εγγράφονται ετεροχρονισμένα στον καταχωρητή προορισμού (R0–R14) του αρχείου καταχωρητών (**mux2to1**). Επιλέγει μεταξύ του αποτελέσματος Result (RegSrc[2] = 0) και της τιμής PC + 4 (RegSrc[2] = 1). Η έξοδος συνδέεται στην είσοδο WD3 του αρχείου καταχωρητών. Η εγγραφή γίνεται στους καταχωρητές R0–R14, όταν RegWrite = 1.
- Παραμετροποιημένος πολυπλέκτης 2 σε 1 των N bit (με αρχική τιμή N = 32) για να χρησιμοποιηθεί στην επιλογή της διεύθυνσης της επόμενης εντολής που πρόκειται να εκτελεσθεί (**mux2to1**). Επιλέγει μεταξύ της τιμής PC + 4 (PCSrc = 0) και του αποτελέσματος Result (PCSrc = 1). Η έξοδος συνδέεται στην είσοδο δεδομένων του μετρητή προγράμματος PC. Η τιμή PC + 4 επιλέγεται ως διεύθυνση της επόμενης εντολής που πρόκειται να εκτελεσθεί, όταν εκτελούνται εντολές που δεν αλλάζουν τη ροή του προγράμματος και όταν δεν ικανοποιείται η συνθήκη κατά την εκτέλεση εντολών υπό συνθήκη. Το αποτέλεσμα Result επιλέγεται ως διεύθυνση της επόμενης εντολής που πρόκειται να εκτελεσθεί, είτε όταν ο καταχωρητής προορισμού Rd στις εντολές επεξεργασίας δεδομένων και στην εντολή LDR είναι ο R15, είτε όταν εκτελείται εντολή διακλάδωσης (B, BL).

2.5 Βήμα 3: Σχεδίαση της διαδρομής δεδομένων (datapath)

Η σχεδίαση της διαδρομής δεδομένων (**datapath**) του επεξεργαστή ενός κύκλου ολοκληρώνεται με τη χρήση περιγραφής δομής στη γλώσσα VHDL ακολουθώντας την ιεραρχική προσέγγιση bottom–up, όπως αυτή αποτυπώνεται στο ακόλουθο σχηματικό διάγραμμα.



Εικόνα 193 – Διαδρομή δεδομένων επεξεργαστή ARM ενός κύκλου

2.6 Βήμα 4: Σχεδίαση της μονάδας ελέγχου

Αρχικά, στο κατώτερο ιεραρχικά επίπεδο της σχεδίασης περιγράφεται στη γλώσσα VHDL η συμπεριφορά των υπομονάδων που απαρτίζουν τη μονάδα ελέγχου. Στη συνέχεια, η σχεδίαση της μονάδας ελέγχου (**control**) του επεξεργαστή ενός κύκλου ολοκληρώνεται με τη χρήση περιγραφής δομής στη γλώσσα VHDL ακολουθώντας την ιεραρχική προσέγγιση bottom–up.

2.6.1 Σχεδίαση του αποκωδικοποιητή εντολής (InstrDec).

- Συμπλήρωση του πίνακα αλήθειας του αποκωδικοποιητή εντολής (**InstrDec**).

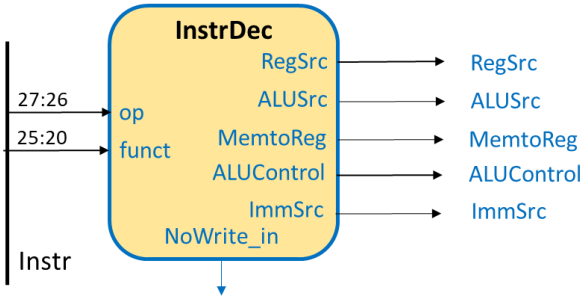
  Είσοδοι, τα πεδία op και funct της εντολής. Έξοδοι τα σήματα ελέγχου RegSrc[1:0], ALUSrc, ImmSrc, ALUControl[1:0] και MemtoReg, καθώς και το εσωτερικό σήμα NoWrite_in. Απαιτείται τροποποίηση, ώστε να καλύπτονται οι επιπλέον εντολές επεξεργασίας δεδομένων που υλοποιούνται, καθώς και η εντολή BL.

  Δίδονται ως παράδειγμα οι πίνακες αλήθειας αφενός για τις εντολές ADD, SUB, CMP, AND και ORR και αφετέρου για τις εντολές LDR, STR και B.

Πίνακας 2 – Παράδειγμα πίνακα αλήθειας αποκωδικοποιητή εντολής (InstrDec)

Εντολή	$Instr_{27:26}$ op	$Instr_{25:20}$ funct	Τύπος	RegSrc	ALUSrc	Imm Src	ALU Control	Memto Reg	NoWrite_in
ADD	00	1 0100 X	DP Imm	X0	1	0	00	0	0
ADD	00	0 0100 X	DP Reg	00	0	X	00	0	0
SUB	00	1 0010 X	DP Imm	X0	1	0	01	0	0
SUB	00	0 0010 X	DP Reg	00	0	X	01	0	0
CMP	00	1 1010 1	DP Imm	X0	1	0	01	X	1
CMP	00	0 1010 1	DP Reg	00	0	X	01	X	1
AND	00	1 0000 X	DP Imm	X0	1	0	10	0	0
AND	00	0 0000 X	DP Reg	00	0	X	10	0	0
ORR	00	1 1100 X	DP Imm	X0	1	0	11	0	0
ORR	00	0 1100 X	DP Reg	00	0	X	11	0	0
LDR	01	0 1100 1	M Imm +	X0	1	0	00	1	0
LDR	01	0 1000 1	M Imm –	X0	1	0	01	1	0
STR	01	0 1100 0	M Imm +	10	1	0	00	X	0
STR	01	0 1000 0	M Imm –	10	1	0	01	X	0
B	10	1 0XXX X	B Imm +	X1	1	1	00	0	0

- Με βάση τον πίνακα αλήθειας γίνεται η σχεδίαση του αποκωδικοποιητή εντολής (**InstrDec**) σε γλώσσα VHDL (περιγραφή συμπεριφοράς με τη χρήση της εντολής case). Προσοχή! Η αξιοποίηση των αδιάφορων τιμών X στην έξοδο οδηγούν σε απλοποίηση του συνδυαστικού κυκλώματος.



Εικόνα 194 – Σχηματικό διάγραμμα του αποκωδικοποιητή εντολής

2.6.2 Σχεδίαση του αποκωδικοποιητή των σημάτων έγκρισης εγγραφής (WELogic).

- Συμπλήρωση του πίνακα αλήθειας του αποκωδικοποιητή των σημάτων έγκρισης εγγραφής (**WELogic**).

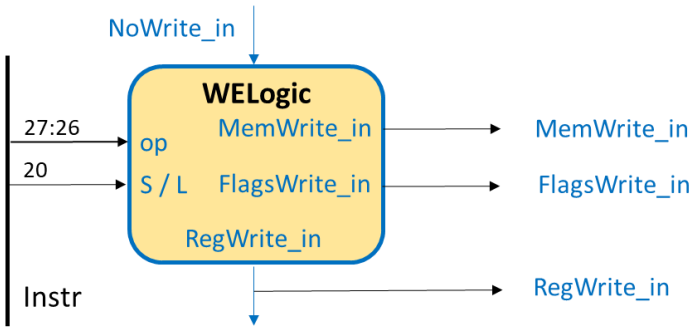
  Είσοδοι, τα πεδία op και S/L της εντολής, καθώς και το εσωτερικό σήμα NoWrite_in.

  Έξοδοι, τα εσωτερικά σήματα RegWrite_in, FlagsWrite_in και MemWrite_in, που ελέγχονται από το εσωτερικό σήμα CondEx_in, το οποίο εμποδίζει την εγγραφή στο αρχείο καταχωρητών (RegWrite = 0), στον καταχωρητή κατάστασης (FlagsWrite = 0) και στη μνήμη δεδομένων (MemWrite = 0), όταν CondEx_in = 0.

Πίνακας 4 – Παράδειγμα πίνακα αλήθειας αποκωδικοποιητή σημάτων έγκρισης εγγραφής

Τύπος	$Instr_{27:26}$ op	$Instr_{20}$ S / L	No Write_in	Reg Write_in	Mem Write_in	Flags Write_in
DP	00	0	0	1	0	0
DP	00	1	0	1	0	1
CMP	00	1	1	0	0	1
LDR	01	1	0	1	0	0
STR	01	0	0	0	1	0
B	10	X	0	0	0	0

- Με βάση τον πίνακα αλήθειας γίνεται η σχεδίαση του αποκωδικοποιητή των σημάτων έγκρισης εγγραφής (**WELogic**) σε γλώσσα VHDL (περιγραφή συμπεριφοράς με τη χρήση της εντολής case).



Εικόνα 195 – Σχηματικό διάγραμμα αποκωδικοποιητή σημάτων έγκρισης εγγραφής

2.6.3 Σχεδίαση της λογικής επιλογής διεύθυνσης επόμενης εντολής (PCLogic).

- Συμπλήρωση του πίνακα αλήθειας της λογικής επιλογής διεύθυνσης επόμενης εντολής (**PCLogic**) που ενεργοποιεί το εσωτερικό σήμα PCSrc_in, είτε όταν απαιτείται ετεροχρονισμένη εγγραφή στον καταχωρητή R15 (PC) (δηλαδή όταν Rd = 15 και RegWrite_in = 1), είτε όταν εκτελείται εντολή διακλάδωσης (op[1] = 1).

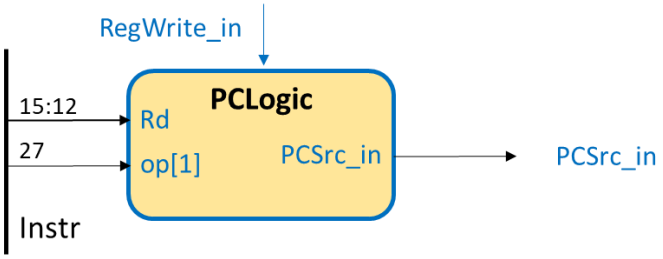
  Είσοδοι, τα πεδία Rd και op[1] της εντολής καθώς και το εσωτερικό σήμα RegWrite_in. Έξοδος, το εσωτερικό σήμα PCSrc_in που ελέγχεται από το εσωτερικό σήμα CondEx_in, το οποίο παίρνει την τιμή 0, όταν CondEx_in = 0.

  Δίδεται ως παράδειγμα ο πίνακας αλήθειας για εντολές επεξεργασίας δεδομένων, μνήμης και διακλάδωσης.

Πίνακας 5 – Πίνακας αλήθειας εντολών επεξεργασίας δεδομένων, μνήμης και διακλάδωσης

Τύπος	$Instr_{27:26}$ op	$Instr_{15:12}$ Rd	Reg Write_in	PCSrc_in
DP	00	0000 1110	1	0
DP	00	1111	1	1
CMP	00	XXXX	0	0
LDR	01	0000 1110	1	0
LDR	01	1111	1	1
STR	01	XXXX	0	0
B	10	XXXX	0	1

- Με βάση τον πίνακα αλήθειας γίνεται η σχεδίαση της λογικής επιλογής διεύθυνσης επόμενης εντολής (**PCLogic**) σε γλώσσα VHDL (περιγραφή συμπεριφοράς με τη χρήση της εντολής case).



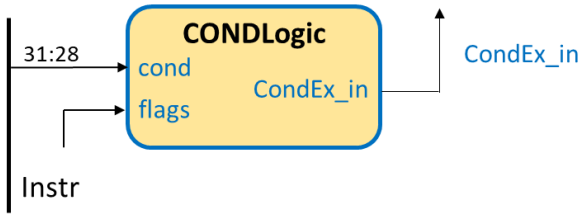
Εικόνα 196 – Σχηματικό διάγραμμα της λογικής επιλογής διεύθυνσης επόμενης εντολής

2.6.4 Σχεδίαση της λογικής ελέγχου συνθήκης (CONDLogic).

- Συμπλήρωση του πίνακα αλήθειας της λογικής ελέγχου συνθήκης (**CONDLogic**), που ελέγχει εάν ικανοποιείται η συνθήκη που ορίζεται στο πεδίο cond της εντολής με βάση τις τρέχουσες τιμές των σημαιών N, Z, C, V (flags).

  Είσοδοι, το πεδίο cond της εντολής και η έξοδος flags του καταχωρητή καταστάσεων. Έξοδος: το σήμα CondEx_in που εγκρίνει την εκτέλεση της εντολής, όταν παίρνει την τιμή 1.

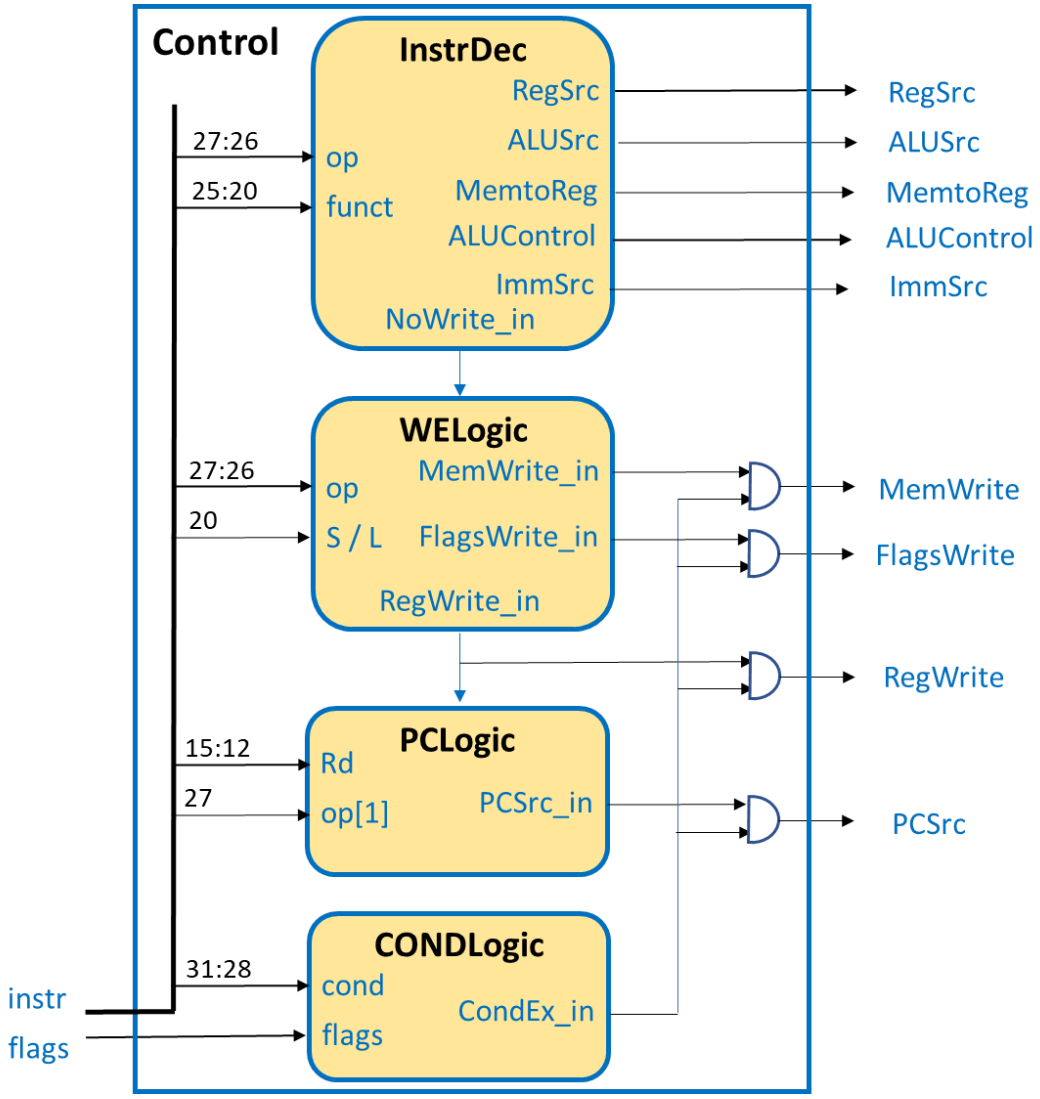
- Με βάση τον πίνακα της παραγράφου 1–1.4, όπου φαίνονται τα **μνημονικά συνθήκης** με τις **εξισώσεις Boole** των σημαιών που τις ικανοποιούν, γίνεται η σχεδίαση της λογικής ελέγχου συνθήκης (**CONDLogic**) σε γλώσσα VHDL (περιγραφή συμπεριφοράς με τη χρήση της εντολής case).



Εικόνα 197 – Σχηματικό διάγραμμα λογικής ελέγχου συνθήκης

2.6.5 Σχεδίαση της μονάδας ελέγχου (control)

Η σχεδίαση της μονάδας ελέγχου (**control**) του επεξεργαστή ενός κύκλου ολοκληρώνεται με τη χρήση περιγραφής δομής στη γλώσσα VHDL ακολουθώντας την ιεραρχική προσέγγιση bottom–up, όπως αυτή αποτυπώνεται στο ακόλουθο σχηματικό διάγραμμα (σε αυτό το επίπεδο εισάγονται και οι απαραίτητες πύλες AND).



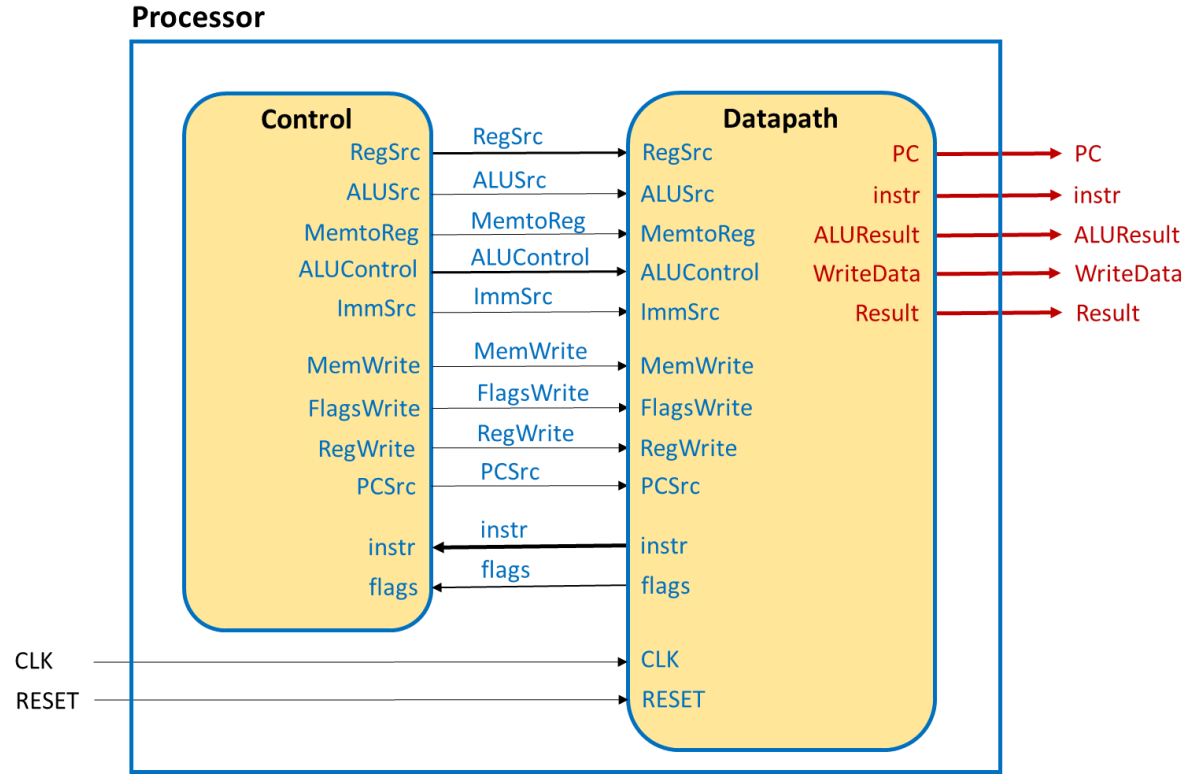
Εικόνα 198 – Σχηματικό διάγραμμα μονάδας ελέγχου (control) επεξεργαστή ενός κύκλου

2.7 Βήμα 5: Σχεδίαση του επεξεργαστή (processor)

Στο υψηλότερο ιεραρχικά επίπεδο σχεδιάζεται ο επεξεργαστής (**processor**) με τη χρήση περιγραφής δομής στη γλώσσα VHDL ακολουθώντας την ιεραρχική προσέγγιση bottom–up, όπως αυτή αποτυπώνεται στο ακόλουθο σχηματικό διάγραμμα.

Στο πλαίσιο του Project 1, επιλέγεται ο επεξεργαστής να έχει ως εξόδους τις αρτηρίες **PC** και **Instr**, που σχετίζονται με τη μνήμη εντολών, και τις αρτηρίες **ALUResult**, **WriteData** και **Result**, που σχετίζονται με τη μνήμη δεδομένων και τη μονάδα ALU. Οι είσοδοι του επεξεργαστή είναι τα σήματα CLK και RESET. Πριν την υλοποίηση θα πρέπει να φορτωθεί το απαραίτητο πρόγραμμα στη μνήμη ROM που να συμπεριλαμβάνει όλες τις εντολές που υλοποιούνται. Προσοχή! Θα πρέπει να μελετηθούν προσεκτικά οποιεσδήποτε ανεπιθύμητες απλοποιήσεις που ενδεχομένως να γίνουν στο στάδιο της υλοποίησης.

Η διασύνδεση της διαδρομής δεδομένων (**datapath**) και της μονάδας ελέγχου (**control**) φαίνεται στο επόμενο σχηματικό διάγραμμα στο επίπεδο του επεξεργαστή (**processor**).



Εικόνα 199 – Σχηματικό διάγραμμα επιπέδου επεξεργαστή

2.8 Βήμα 6: Επαλήθευση της ορθής σχεδίασης του επεξεργαστή (processor)

Η επαλήθευση της ορθής σχεδίασης του επεξεργαστή (**processor**) που βασίζεται στην προσομοίωση (simulation based verification) εφαρμόζεται σε επιλεγμένες υπομονάδες της διαδρομής δεδομένων, όπως η μονάδα ALU (**ALU**) και το αρχείο καταχωρητών (**RF**), στη μονάδα ελέγχου (**control**) και στον επεξεργαστή (**processor**) σαν ολότητα.

2.8.1 Επαλήθευση της ορθής σχεδίασης της μονάδας ALU (ALU).

- Ακολουθήστε όλα τα Βήματα 1–5 της ιεραρχικής σχεδίασης της οντότητας της μονάδας ALU (**ALU**), όπως αυτά περιγράφονται στον οδηγό χρήσης των βασικών λειτουργειών του **Vivado IDE** δημιουργώντας και το απαραίτητο πρόγραμμα δοκιμής (**ALU_TB**), που εκτελεί όλες τις πράξεις. Χρησιμοποιείστε ως βάση τον αθροιστή με καταχωρητές εισόδου και εξόδου.

2.8.2 Επαλήθευση της ορθής σχεδίασης του αρχείου καταχωρητών (RF).

- Ακολουθήστε όλα τα Βήματα 1–5 της ιεραρχικής σχεδίασης της οντότητας του αρχείου καταχωρητών (**RF**), όπως αυτά περιγράφονται στον οδηγό χρήσης των βασικών λειτουργειών του **Vivado IDE** δημιουργώντας και το απαραίτητο πρόγραμμα δοκιμής (**RF_TB**), που γράφει στους καταχωρητές R0–R14 (δεν ορίζεται εγγραφή στον καταχωρητή R15) και διαβάζει όλους τους καταχωρητές R0–R15 (και τον καταχωρητή R15 – απαιτείται κατάλληλη τιμή στην είσοδο R15). Χρησιμοποιείστε ως βάση τον αθροιστή με καταχωρητές εισόδου και εξόδου.

2.8.3 Επαλήθευση της ορθής σχεδίασης της μονάδας ελέγχου (control).

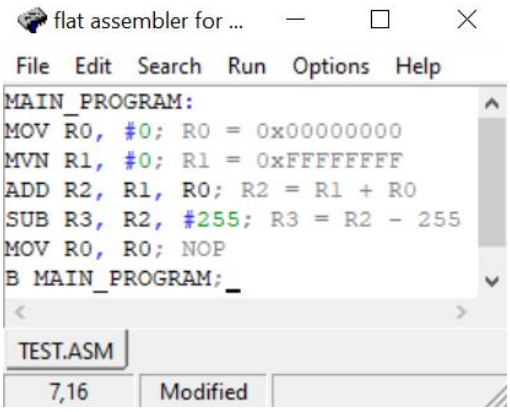
- Ακολουθήστε όλα τα Βήματα 1–5 της ιεραρχικής σχεδίασης της οντότητας της μονάδας ελέγχου (**control**), όπως αυτά περιγράφονται στον οδηγό χρήσης των βασικών λειτουργειών του **Vivado IDE** δημιουργώντας και το απαραίτητο πρόγραμμα δοκιμής (**control_TB**), που επιβεβαιώνει την ορθή λειτουργία όλων των εντολών και όλων των διακλαδώσεων των εντολών case και if του κώδικα στη γλώσσα VHDL. Χρησιμοποιείστε ως βάση τον αθροιστή, αλλά χωρίς καταχωρητές εισόδου και εξόδου.

2.8.4 Επαλήθευση της ορθής σχεδίασης του επεξεργαστή (processor).

- Ακολουθήστε όλα τα Βήματα 1–5 της ιεραρχικής σχεδίασης της οντότητας του επεξεργαστή (**processor**), όπως αυτά περιγράφονται στον οδηγό χρήσης των βασικών λειτουργειών του Vivado IDE δημιουργώντας και το απαραίτητο πρόγραμμα δοκιμής (**processor_TB**). Προσοχή! Το πλήθος των κύκλων ρολογιού που θα τρέξει ο επεξεργαστής πρέπει να αντιστοιχεί στο πλήθος των εντολών που είναι αποθηκευμένες στη μνήμη ROM. Σε κάθε περίπτωση, η τελευταία εντολή του προγράμματός σας επαναφέρει τη συνέχεια της εκτέλεσης του προγράμματος στην πρώτη εντολή του. Να γίνει χρήση και της ψευδοεντολής NOP.
- Δημιουργήστε ένα πρόγραμμα σε συμβολική γλώσσα της αρχιτεκτονικής ARM, που να συμπεριλαμβάνει όλες τις εντολές που υλοποιείτε και ενεργοποιεί όλες τις πιθανές ροές δεδομένων και λειτουργίες στη διαδρομή δεδομένων.

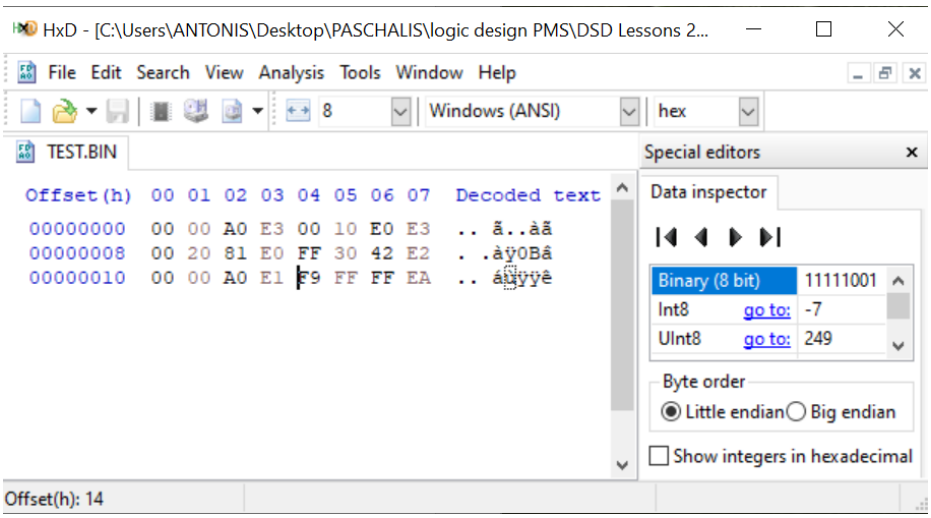
Για τη δημιουργία του προγράμματος μπορείτε να χρησιμοποιήσετε έναν ελεύθερο συμβολομεταφραστή (assembler) της αρχιτεκτονικής ARM, όπως είναι ο **FASMARM** (https://arm.flatassembler.net). Ο FASMARM στην πραγματικότητα είναι cross assembler υπό την έννοια ότι παράγει μεν κώδικα σε γλώσσα μηχανής ARM, ο ίδιος όμως δεν τρέχει σε επεξεργαστή αρχιτεκτονικής ARM αλλά αρχιτεκτονικής X86/X64.

Εκτελέστε την εφαρμογή **FASMWARM.EXE**. Εμφανίζεται το παράθυρο του editor στο οποίο γράφετε τον κώδικα ARM, όπως στο παράδειγμα. Αναφορικά με τα labels στον κώδικα, ο FASMARM έχει μία ιδιαιτερότητα: μετά από κάθε label στην αρχή μίας γραμμής (όχι αν το label αποτελεί μέρος της εντολής, όπως η B) πρέπει να βάζετε το σύμβολο «:». Ο assembler θεωρεί συντακτικό λάθος την απουσία του. Αποθηκεύστε και ονομάστε το πηγαίο αρχείο στη συμβολική γλώσσα της αρχιτεκτονικής ARM επιλέγοντας **file–>save as** και δηλώνοντας το όνομα (**TEST.ASM**).



Εικόνα 200 – Παράδειγμα παραθύρου Flat Assembler

- Μετατρέψτε το πρόγραμμά σας σε γλώσσα μηχανής, επιλέγοντας **Run–>Compile**. Εάν ο assembler εντοπίσει συντακτικά λάθη, η συμβολομετάφραση αποτυγχάνει και εμφανίζεται ένα αναδυόμενο παράθυρο που επισημαίνει το πρώτο συντακτικό σφάλμα που συναντήθηκε. Μετά την αποσφαλμάτωση του πηγαίου αρχείου .ASM παράγεται το αντίστοιχο αρχείο σε γλώσσα μηχανής που είναι binary (**TEST.BIN**). Φαίνεται στο ίδιο directory με το πηγαίο αρχείο TEST.ASM.
- Ανοίξτε το binary αρχείο με έναν ελεύθερο hex editor, όπως είναι ο **HxD**, αφού πρώτα τον εγκαταστήσετε στον υπολογιστή σας.



Εικόνα 201 – Παράδειγμα παραθύρου HxD

Κάθε 4 συνεχόμενα byte (1 byte = 2 δεκαεξαδικά ψηφία) αποτελούν μία εντολή, ξεκινώντας από την πρώτη κατά σειρά έως και την τελευταία, όπως αυτές εμφανίζονται στο πηγαίο αρχείο **TEST.ASM**.

- Αντιγράψτε τις εντολές σε γλώσσα μηχανής στο αρχείο **ROM_array.vhd** που περιγράφει τη μνήμη εντολών, ως μνήμη ROM. Αντιγράψτε μία–μία τις τετράδες των byte στις κατάλληλες θέσεις του ROM_array ξεκινώντας από την πρώτη θέση. Θα χρειαστεί να αφαιρέσετε με το χέρι τα ενδιάμεσα κενά μεταξύ των διαδοχικών byte και να **αντιστρέψετε** τη σειρά τους, αφού το περισσότερο σημαντικό byte (most significant byte, **MSB**) της λέξης εντολών βρίσκεται στα **αριστερά**, ενώ το λιγότερο σημαντικό byte (least significant byte, **LSB**) της λέξης εντολών βρίσκεται στα **δεξιά**. Προσοχή! Στην περίπτωση που οι εντολές είναι λιγότερες από τη συνολική χωρητικότητα της μνήμης ROM θα πρέπει να γεμίσετε τις υπόλοιπες θέσεις με X”00000000”. Για παράδειγμα, για μία μνήμη ROM χωρητικότητας 16 εντολών των 32 bit (N = 4, M = 32) ο κώδικας σε γλώσσα VHDL διαμορφώνεται ως εξής:

ROM_array.vhd

C:/Users/ANTONIS/Xilinx/Projects/DSD/Examples/Examples.srcs/sources_1/new/ROM_array.vhd

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  -- Uncomment the following library declaration if using
26  -- arithmetic functions with Signed or Unsigned values
27  use IEEE.NUMERIC_STD.ALL;
28
29  -- Uncomment the following library declaration if instantiating
30  -- any Xilinx leaf cells in this code.
31  --library UNISIM;
32  --use UNISIM.VComponents.all;
33
34  entity ROM_array is
35    generic (
36      N : positive := 4;    -- address length
37      M : positive := 32);   -- data word length
38    port (
39      ADDR:      in STD_LOGIC_VECTOR (N-1 downto 0);
40      DATA_OUT: out STD_LOGIC_VECTOR (M-1 downto 0));
41  end ROM_array;
42
43  architecture Behavioral of ROM_array is
44    type ROM_array is array (2**N-1 downto 0)
45      of STD_LOGIC_VECTOR (M-1 downto 0);
46    constant ROM : ROM_array := (
47      X"E3A00000", X"E3E01000", X"E0812000", X"E24230FF",
48      X"E1A00000", X"EAFFFFF9", X"00000000", X"00000000",
49      X"00000000", X"00000000", X"00000000", X"00000000",
50      X"00000000", X"00000000", X"00000000", X"00000000");
51  begin
52    DATA_OUT <= ROM(to_integer(unsigned(ADDR)));
53  end Behavioral;
```

Εικόνα 202 – Παράδειγμα αρχείου μνήμης ROM με φορτωμένο πρόγραμμα

- Ολοκληρώστε την επαλήθευση της ορθής σχεδίασης του επεξεργαστή (**processor**) συγκρίνοντας για κάθε εντολή, που εκτελείται σε έναν κύκλο, τις τιμές των αρτηριών **PC** και **Instr**, που σχετίζονται με τη μνήμη εντολών, και των αρτηριών **ALUResult**, **WriteData** και **Result**, που σχετίζονται με τη μνήμη δεδομένων και τη μονάδα ALU που προκύπτουν μετά την προσομοίωση με τις αντίστοιχες αναμενόμενες τιμές που φαίνονται στον ακόλουθο πίνακα επαλήθευσης ορθής λειτουργίας του επεξεργαστή και προκύπτουν μετά από μελέτη των εντολών που έχουν αποθηκευτεί στη μνήμη εντολών και έχουν υλοποιηθεί στον επεξεργαστή. Επιπλέον μελετήστε και τιμές των σημαιών N, Z, C, V, ως εσωτερικά σήματα.

Πίνακας 6 – Τιμές αρτηριών για κάθε εντολή του προγράμματος

α.α.	Εντολή	PC	Instr	ALUResult	WriteData	Result	Σχόλια
1	MOV R0, #0	0x00	E3A00000	00000000		00000000	
2	MVN R1, #0	0x04	E3E01000	FFFFFFFF		FFFFFFFF	
3	ADD R2, R1, R0	0x08	E0812000	FFFFFFFF	00000000	FFFFFFFF	
4	SUB R3, R2, #255	0x0C	E24230FF	FFFFFF00		FFFFFF00	
5	MOV R0, R0	0x10	E1A00000	00000000	00000000	00000000	
6	B MAIN_PROGRAM	0x14	EAFFFFF9	00000000		00000000	PC' = 0x00

2.9 Βήμα 7: Παράδοση τεχνικής αναφοράς της σχεδίασης του επεξεργαστή

Μετά την ολοκλήρωση της σχεδίασης του επεξεργαστή (**processor**) παραδίδετε το project συνοδευόμενο από μία τεχνική αναφορά η οποία συμπεριλαμβάνει τα ακόλουθα:

2.9.1 Περιγραφή των στοιχείων και της δομής του επεξεργαστή.

- Περιγράψτε το σύνολο των εντολών που έχετε υλοποιήσει.
- Περιγράψτε τα κύρια ψηφιακά δομικά στοιχεία (components) της διαδρομής δεδομένων, όπως είναι το αρχείο καταχωρητών, η μονάδα ALU, η μνήμη εντολών (ως μνήμη ROM) και η μνήμη δεδομένων (ως μνήμη distributed RAM). Παραθέστε τον κώδικα σε γλώσσα VHDL (περιγραφή συμπεριφοράς) και τα προκύπτοντα σχηματικά διαγράμματα στο επίπεδο RTL (elaborated design).
- Επιπλέον για το αρχείο καταχωρητών και τη μονάδα ALU, παραθέστε τα απαραίτητα προγράμματα δοκιμής και τα διαγράμματα χρονισμού των προσομοιώσεων του behavioral model, του post–synthesis model (μόνο λογική προσομοίωση που πρέπει να ταυτίζεται με εκείνη του behavioral model) και του post–implementation model (μόνο χρονική προσομοίωση) και τεκμηριώστε την ορθή τους σχεδίαση και λειτουργία.
- Περιγράψτε τη δομή της διαδρομής δεδομένων (**datapath**) του επεξεργαστή στα ανώτερα ιεραρχικά επίπεδα. Παραθέστε τους κώδικες σε γλώσσα VHDL (περιγραφή δομής) και σχολιάστε τα προκύπτοντα σχηματικά διαγράμματα στο επίπεδο RTL (elaborated design).
- Περιγράψτε όλες τις υπομονάδες της μονάδας ελέγχου συμπληρώνοντας τους αντίστοιχους πίνακες αλήθειας και παραθέστε τους προκύπτοντες κώδικες σε γλώσσα VHDL (περιγραφή συμπεριφοράς).
- Περιγράψτε τη δομή της μονάδας ελέγχου (**control**) του επεξεργαστή στο ανώτερο ιεραρχικό επίπεδο. Παραθέστε τον κώδικα σε γλώσσα VHDL (κυρίως περιγραφή δομής) και σχολιάστε τα προκύπτοντα σχηματικά διαγράμματα στο επίπεδο RTL (elaborated design). Παραθέστε το απαραίτητο πρόγραμμα δοκιμής και τα διαγράμματα χρονισμού των προσομοιώσεων του behavioral model, του post–synthesis model (μόνο λογική προσομοίωση που πρέπει να ταυτίζεται με εκείνη του behavioral model) και του post–implementation model (μόνο χρονική προσομοίωση) και τεκμηριώστε την ορθή του σχεδίαση και λειτουργία.

- Περιγράψτε τη δομή του επεξεργαστή (**processor**) στο ανώτερο ιεραρχικά επίπεδο με στοιχεία τη διαδρομή δεδομένων και τη μονάδα ελέγχου. Παραθέστε τον κώδικα σε γλώσσα VHDL (περιγραφή δομής) και το προκύπτον σχηματικό διάγραμμα στο επίπεδο RTL (elaborated design).
- Βρείτε τη μέγιστη συχνότητα λειτουργίας στο επίπεδο του επεξεργαστή (**processor**), προσδιορίζοντας τη χειρότερη κρίσιμη διαδρομή και τη χειρότερη σύντομη διαδρομή.

2.10 Επαλήθευση της ορθής σχεδίασης και λειτουργίας του επεξεργαστή (processor).

- Δημιουργείστε κατάλληλο πρόγραμμα σε συμβολική γλώσσα αρχιτεκτονικής ARM, που συμπεριλαμβάνει όλες τις υλοποιημένες εντολές και ενεργοποιεί όλες τις πιθανές ροές δεδομένων και λειτουργίες στη διαδρομή δεδομένων. Τεκμηριώστε γιατί το πρόγραμμα σας σε συμβολική γλώσσα επαληθεύει την ορθή σχεδίαση και λειτουργία του επεξεργαστή (**processor**) (δηλαδή, ποιο τμήμα του προγράμματος επαληθεύει ποιες εντολές και με ποιον τρόπο).
- Παραθέστε τον κώδικα σε γλώσσα VHDL που περιγράφει τη συμπεριφορά της μνήμης εντολών και το απαραίτητο πρόγραμμα δοκιμής στη μέγιστη συχνότητα λειτουργίας.
- Για αντιπροσωπευτικές εντολές και λειτουργίες, παραθέστε τα διαγράμματα χρονισμού των προσομοιώσεων του behavioral model, του post–synthesis model (μόνο λογική προσομοίωση που πρέπει να ταυτίζεται με εκείνη του behavioral model) και του post–implementation model (μόνο χρονική προσομοίωση) και τεκμηριώστε την ορθή του σχεδίαση και λειτουργία.

2.11 Ανάλυση των αποτελεσμάτων της σύνθεσης και της υλοποίησης του επεξεργαστή (processor).

- Αναλύστε τα αποτελέσματα της σύνθεσης και της υλοποίησης του επεξεργαστή (**processor**) μελετώντας το project summary και το report utilization. Είναι οι χρησιμοποιούμενοι πόροι αυτοί που περιμένατε;

ΣΥΜΠΕΡΑΣΜΑΤΑ

Η εκπόνηση της παρούσας πτυχιακής εργασίας αποτέλεσε μία πολύ ευχάριστη εμπειρία για τον συγγραφέα. Αφενός, έδωσε αφορμή για μία εις βάθος μελέτη του σχεδιαστικού εργαλείου Xilinx Vivado IDE και αφετέρου οδήγησε στην δημιουργία ενός αναλυτικού εργαστηριακού οδηγού για τη σχεδίαση και υλοποίηση ενός επεξεργαστή της οικογένειας ARM. Η εκπαιδευτική διαδικασία έλκει ιδιαίτερα το ενδιαφέρον του συγγραφέα και η πτυχιακή εργασία του έδωσε μία εξαιρετική αφορμή για ενασχόληση με αυτό το αντικείμενο. Το τελικό αποτέλεσμα ικανοποιεί πλήρως τον συγγραφέα.

Όπως γίνεται φανερό από την ίδια την δομή του κειμένου, έχει καταβληθεί ιδιαίτερη προσπάθεια στη λεπτομέρεια σε κάθε βήμα της διαδικασίας σχεδίασης. Ο φοιτητής έχει στη διάθεσή του έναν πλήρη οδηγό σχεδίασης με συγκεντρωμένες όλες τις απαραίτητες πληροφορίες τόσο για το σχεδιαστικό εργαλείο όσο και για τον επεξεργαστή που καλείται να υλοποιήσει. Έτσι, δεν είναι απαραίτητο να χρησιμοποιεί συνεχώς δύο ή περισσότερες πηγές υλικού για να προχωρήσει την εργασία του.

Τα τρία μέρη του εργαστηριακού οδηγού χρησιμοποιήθηκαν επιτυχώς για πρώτη φορά στο εαρινό εξάμηνο 2020 στα πλαίσια του μαθήματος του τμήματος Πληροφορικής & Τηλεπικοινωνιών «Space Data Systems». Το ίδιο υλικό αναμένεται να χρησιμοποιηθεί και στα επόμενα ακαδημαϊκά εξάμηνα, φυσικά με όποιες επιπλέον τροποποιήσεις προκύψουν, μιας και ο μεγαλύτερος δάσκαλος είναι η πράξη.

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	**Ελληνικός Όρος**
Advanced RISC Machine	Προηγμένη Μηχανή RISC
Arithmetic & Logic Unit	Αριθμητική και Λογική Μονάδα
Condition Flag	Σημαία Συνθήκης
Constraint	Περιορισμός
Control unit	Μονάδα ελέγχου
Data Memory	Μνήμη Δεδομένων
Datapath	Διαδρομή δεδομένων
Entity	Οντότητα
Field Programmable Gate Array	Επιτόπια Συστοιχία Προγραμματιζόμενων Πυλών
Finite State Machine	Μηχανή Πεπερασμένων Καταστάσεων
Implementation	Υλοποίηση
Instruction Memory	Μνήμη Εντολών
Integrated Development Environment	Ολοκληρωμένο Περιβάλλον Ανάπτυξης
Multiplexer	Πολυπλέκτης
Reduced Instruction Set Computer	Υπολογιστής Περιορισμένου Συνόλου Εντολών
Register	Καταχωρητής
Register File	Αρχείο Καταχωρητών
Schematic Diagram	Σχηματικό Διάγραμμα
Simulation	Προσομοίωση
Simulation Waveform	Κυματομορφή Προσομοίωσης
Single–Cycle Processor	Επεξεργαστής ενός κύκλου
Slack	Χρονικό περιθώριο
Synthesis	Σύνθεση
System on a Chip	Σύστημα σε chip
Testbench	Πρόγραμμα Δοκιμών
Timing Report	Αναφορά Χρονισμού
Top Design Source/Entity	Κορυφαία οντότητα της ιεραρχίας
Verification	Επαλήθευση Ορθής Σχεδίασης

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ

ARM	Advanced RISC Machine
CPU	Central Processing Unit
FPGA	Fiend Programmable Gate Array
FSM	Finite State Machine
IDE	Ολοκληρωμένο Περιβάλλον Ανάπτυξης
RISC	Reduced Instruction Set Computer
SoC	System on a Chip
ΕΚΠΑ	Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

ΑΝΑΦΟΡΕΣ

[1] Sarah L. Harris & David Money Harris, «Ψηφιακή σχεδίαση και αρχιτεκτονική υπολογιστών – Έκδοση ARM», εκδόσεις Κλειδάριθμος
[2] Vivado Design Suite User Guide Getting Started UG910 (v2018.2) June 6, 2018
[3] Vivado Design Suite User Guide Synthesis UG901 (v2019.2) January 27, 2020
[4] Vivado Design Suite User Guide Implementation UG904 (v2019.2) December 18, 2019
[5] Vivado Design Suite User Guide Logic Simulation UG900 (v2018.3) December 14, 2018
[6] Synthesis and Simulation Design Guide UG626 (v 11.4) December 2, 2009