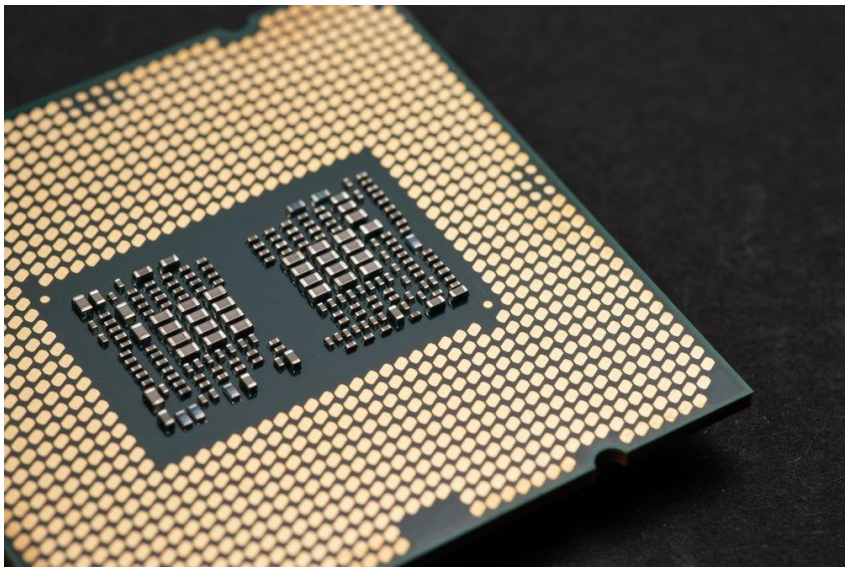
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΔΗΜΙΟΥΡΓΙΑ – ΥΛΟΠΟΙΗΣΗ ΕΠΕΞΕΡΓΑΣΤΗ



Φουρτζής Αναστάσιος

Επιβλέπων: Φώτιος Βαρτζιώτης

Άρτα , Φεβρουάριος, 2025

DESIGN AND IMPLEMENTATION OF A PROCESSOR

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, Ημερομηνία

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής

   Βαρτζιώτης Φώτιος,

2. Μέλος επιτροπής

   Δουμένης Γρηγόριος,

3. Μέλος επιτροπής

   Ελευθέριος Στεργίου.

©Φουρτζής, Αναστάσιος, 2025.
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα πτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Φουρτζής, Αναστάσιος

Υπογραφή

ΕΥΧΑΡΙΣΤΙΕΣ

Πρώτα απ' όλους, θα ήθελα να ευχαριστήσω τον επιβλέποντα Καθηγητή μου, κύριο Φώτιο Βαρτζιώτη, για την έμπνευση, τη συνεργασία και τη σωστή καθοδήγηση που μου προσέφερε. Θα ήθελα επίσης να ευχαριστήσω την οικογένειά μου, που πάντα ήταν δίπλα μου, προσφέροντάς μου αγάπη, ενίσχυση, σταθερότητα και παντός είδους βοήθεια στηρίζοντας τις επιλογές μου, τους φίλους και συμφοιτητές μου για τις όμορφες και δύσκολες στιγμές που ζήσαμε, για τις εμπειρίες που μοιραστήκαμε όλα αυτά τα χρόνια, για την φιλία, τη συναδελφικότητα και γενικά για την βοήθεια και την στήριξή τους.

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία ενημερώνει σύντομα τον αναγνώστη για την ιστορία των επεξεργαστών και πραγματεύεται τη **σχεδίαση και υλοποίηση ενός επεξεργαστή** με χρήση της **V.H.D.L.** και του **Quartus Prime**. Ο επεξεργαστής, εκτελεί τέσσερις βασικές λειτουργίες: mv (move), mvt (move top), add (addition) και sub (subtract). Η αρχιτεκτονική του στηρίζεται στη **μηχανή πεπερασμένων καταστάσεων (F.S.M.)**, η οποία διαχειρίζεται τα χρονικά βήματα εκτέλεσης των εντολών μέσω καταστάσεων T0 έως T3. Στο πρώτο στάδιο, ο επεξεργαστής σχεδιάστηκε με καταχωρητές, πολυπλέκτες και μια A.L.U. για την εκτέλεση αριθμητικών πράξεων. Η ορθή λειτουργία ελέγχθηκε μέσω testbench στο περιβάλλον **ModelSim**, επιβεβαιώνοντας τη σωστή ροή δεδομένων και ελέγχου, με σήματα όπως τα Clock, Resetn, Run και Done. Στο δεύτερο στάδιο, προστέθηκε μία μνήμη ROM 32x16 bit με program counter (PC) για ακολουθιακή ανάγνωση εντολών με όνομα inst_mem.mif. Η επικοινωνία της μνήμης με τον επεξεργαστή εξασφάλισε τη συνεχή ροή εκτέλεσης των εντολών με συγχρονισμό ρολογιού. Η εργασία, αποδεικνύει τη σημασία των **F.S.M.,** ALU, πολυπλεκτών και της μνήμης στη λειτουργία ενός απλού επεξεργαστή καθώς και έχει στόχο την κατανόηση τέτοιων θεμάτων και από τον κοινό αναγνώστη. Η ορθή λειτουργία επιβεβαιώθηκε με επιτυχημένα αποτελέσματα προσομοίωσης στο **ModelSim**, ανταποκρινόμενη πλήρως στις απαιτήσεις της σχεδίασης και των εντολών όσον αφορά το κομμάτι της υλοποίησης.

Λέξεις-κλειδιά: **Σχεδίαση και υλοποίηση ενός επεξεργαστή, V.H.D.L., μηχανή πεπερασμένων καταστάσεων (F.S.M.), ModelSim, Quartus Prime** .

ABSTRACT

The present thesis informs the reader briefly about the story of Processors and deals with the **design and implementation of a processor** using **V.H.D.L.** and **Quartus Prime**. The processor, in this instance performs four basic operations: mv (move), mvt (move top), add (addition), and sub (subtract). Its architecture is based on a **finite state machine (F.S.M.)**, which manages the execution timing of instructions through states T0 to T3. In the first stage, the processor was designed with registers, multiplexers, and an A.L.U. for performing arithmetic operations. The correct operation was verified using a testbench in the **ModelSim** environment, confirming the proper flow of data and control signals such as Clock, Resetn, Run, and Done. In the second stage, a 32x16-bit ROM memory was added, along with a program counter (PC), for sequential reading of instructions from an inst_mem.mif file. The communication between the memory and the processor ensured the continuous flow of instruction execution synchronized with the clock. The thesis demonstrates the importance of an **F.S.M.**, ALU, multiplexers and memory in the operation of a simple processor, while also aiming to provide an understanding of such topics to the average reader. The correct functionality was confirmed with successful simulation results in **ModelSim**, fully meeting the design and instruction requirements.

Keywords: **design and implementation of a processor, V.H.D.L., Quartus Prime, Finite State Machine (F.S.M.), ModelSim**.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΔΙΑΓΡΑΜΜΑΤΩΝ/ΕΙΚΟΝΩΝ

ΠΙΝΑΚΑΣ ΣΥΝΤΟΜΟΓΡΑΦΙΩΝ

VHDL…[VHSIC (Very High Speed Integrated Circuit) Hardware Description Language].

F.S.M…[Finite State Machine].

E.N.I.A.C…[Electronic Numerical Integrator And Computer].

C.P.U…[Central Processing Unit].

G.P.U…[Graphics Processing Unit].

A.L.U…[Arithmetic Logical Unit].

P.C…[Program Counter].

I.R…[Instruction Register].

C.MOS…[Complementary Metal-Oxide-Semiconductor].

R.I.S.C…[Reduced Instruction Set Computer].

C.I.S.C…[Complex Instruction Set Computer].

A.R.M…[Advanced R.I.S.C. Machine].

F.P.G.A…[Field Programable Gate Arrays].

C.P.L.D…[Complex Programmable Logic Devices].

Op. Code…[Operations Code].

Mv…[Move].

Mvt…[Move Top].

Add…[Addition].

Sub…[Subtraction].

R.T.L sim…[Register Transfer Level Simulation].

M.I.F…[Memory Initialization File].

B.U.S…[Binary Unit System].

R.O.M…[Read Only Memory].

ΕΙΣΑΓΩΓΗ

Η δημιουργία και υλοποίηση επεξεργαστών αποτελεί ένα συναρπαστικό πεδίο ερευνητικής εξέλιξης στην τεχνολογία των υπολογιστών. Η διαδικασία από την αρχιτεκτονική των επεξεργαστών μέχρι την σχεδίαση και την υλοποίησή τους, σε επίπεδο “hardware”, απαιτεί εμβάθυνση σε πολλούς τομείς της ηλεκτρονικής, της μηχανικής και της πληροφορικής. Η πτυχιακή εργασία που ακολουθεί, θα καλύπτει θέματα όπως η εξέλιξη των αρχιτεκτονικών επεξεργαστών και θα εστιάσει σε συγκεκριμένες τεχνικές σχεδίασης. Είναι ένας τομέας που συνδυάζει θεωρία και πρακτική, προσφέροντας πολλές δυνατότητες για εξερεύνηση και καινοτομία. Τέλος, θα αναφερθούν οι τεχνικές που χρησιμοποιήθηκαν, για την υλοποίηση της εργασίας και θα ληφθούν στοιχεία παρουσιάζοντας τις κυματομορφές μέσω του προγράμματος που χρησιμοποιήθηκε.

Η κεντρική μονάδα επεξεργασίας, ή επεξεργαστής, "C.P.U."("Central Processing Unit"), ακριβέστερα μικροεπεξεργαστής αλλά για λόγους συντομίας το «μικρο» παραλείπεται, είναι το «μυαλό» ενός υπολογιστικού συστήματος, δηλαδή το βασικότερο μέρος για τις λειτουργίες του. Η λειτουργία του επεξεργαστή πραγματοποιεί την εκτέλεση των προγραμμάτων που βρίσκονται αποθηκευμένα στην κύρια μνήμη. Τα μέρη, εν συντομία, από τα οποία αποτελείται ένας επεξεργαστής είναι τα εξής:

1. Η Μονάδα Ελέγχου ("Control Unit"), η οποία είναι υπεύθυνη για την ανάκτηση των εντολών από την κύρια μνήμη και για την αναγνώριση του τύπου τους.

2. Η Αριθμητική και Λογική Μονάδα ("Arithmetic Logic Unit"), η οποία εκτελεί αποκλειστικά πράξεις, που χρειάζονται για την εκτέλεση των εντολών.

3. Οι καταχωρητές ("Registers"), οι οποίοι είναι μια μικρή μνήμη υψηλής ταχύτητας, που χρησιμοποιείται για την αποθήκευση προσωρινών αποτελεσμάτων και ορισμένων δεδομένων ελέγχου. Ο καθένας από αυτούς, εκτελεί μια συγκεκριμένη λειτουργία, όπως είναι ο μετρητής προγράμματος ("program counter – P.C."), ο οποίος δείχνει την επόμενη προς εκτέλεση εντολή και ο καταχωρητής εντολών ("instruction register – I.R."), που περιέχει την τρέχουσα εντολή της στιγμής.

Ο επεξεργαστής έχει σημαντική θέση όσον αφορά τα διάφορα θέματα του υπολογιστικού συστήματος, όπως είναι:
Απόδοση: Ο επεξεργαστής είναι ο καθοριστικός παράγοντας της απόδοσης του

συστήματος. Παρ’ όλο που και αρκετοί άλλοι παράγοντες έχουν σημαντική θέση στην απόδοσή, οι ικανότητες του επεξεργαστή καθορίζουν τη μέγιστη απόδοση του συστήματος. Τα άλλα μέρη του υπολογιστή επιτρέπουν στον επεξεργαστή να φτάσει στο μέγιστο των δυνατοτήτων του, σε συγχρονισμό με αυτόν. Υποστήριξη λογισμικού: Νεότεροι επεξεργαστές καθιστούν ικανή τη χρήση πιο σύγχρονου λογισμικού. Επιπρόσθετα, επιτρέπουν τη χρήση ειδικού λογισμικού, το οποίο δεν ήταν σε θέση να υποστηρίξουν επεξεργαστές προηγούμενων γενιών. Αξιοπιστία και Σταθερότητα: Η ποιότητα του επεξεργαστή είναι ένας καθοριστικός παράγοντας αξιοπιστίας του συστήματος.

Κατανάλωση ενέργειας και ψύξη: Οι πρώτοι επεξεργαστές κατανάλωναν μικρή ποσότητα ενέργειας, ενώ οι σύγχρονοι μεγαλύτερη. Η κατανάλωση ενέργειας έχει καθοριστική σημασία για την επιλογή της μεθόδου ψύξης του επεξεργαστή.

Υποστήριξη μητρικής πλακέτας: Ο επεξεργαστής που θα επιλεχθεί για το κάθε σύστημά είναι ένας βασικός και καθοριστικός παράγοντας για το είδος της μητρικής πλακέτας που θα χρησιμοποιηθεί. Η μητρική πλακέτα καθορίζει πολλές διαστάσεις των δυνατοτήτων και της απόδοσης του συστήματος. Επί παραδείγματι, δεν είναι δυνατόν να χρησιμοποιηθεί ένα chipset τύπου "A.M.4" σε μία μητρική που υποστηρίζει chipset τύπου 1700 (το ένα είναι "A.M.D.", το άλλο είναι "Intel"). Επίσης, εάν η αρχιτεκτονική είναι παρόμοια, μπορούν να γίνουν αναβαθμίσεις.

Όπως κάθε μέρος ενός υπολογιστικού συστήματος, έτσι και ο επεξεργαστής βρίσκεται συνεχώς σε εξέλιξη. Οι επεξεργαστές, εδώ και πολλά χρόνια, διπλασιάζουν την απόδοση τους σχεδόν κάθε ενάμιση χρόνο και γίνονται γρηγορότεροι και μικρότεροι. Απ’ ότι δείχνουν τα πράγματα, αυτός ο ρυθμός θα συνεχίσει και θα μειώνεται ο απαιτούμενος χρόνος δημιουργίας νέων επεξεργαστών.

Ο στόχος της παρούσας πτυχιακής εργασίας είναι η εμβάθυνση στις αρχιτεκτονικές επεξεργαστών και η υλοποίηση ενός μοντέλου επεξεργαστή με τη χρήση σύγχρονων εργαλείων σχεδίασης, όπως το "Quartus" και η γλώσσα περιγραφής υλικού "V.H.D.L.". Παράλληλα, επιδιώκεται να παρουσιαστεί το θέμα με τρόπο που να είναι κατανοητός και προσιτός ακόμη και σε άτομα που δεν διαθέτουν προηγούμενη εξοικείωση με το

αντικείμενο, εξηγώντας βασικές έννοιες και τεχνικές ώστε να καλυφθούν οι ανάγκες ενός ευρύτερου κοινού.

Στα επόμενα κεφάλαια αναλύονται διαδοχικά τα θέματα που πλαισιώνουν την εργασία. Στο πρώτο κεφάλαιο γίνεται μια ιστορική αναδρομή στους επεξεργαστές, περιγράφοντας τη διαρκή εξέλιξη και τη σημαντικότητά τους στην τεχνολογία των υπολογιστών. Στο δεύτερο κεφάλαιο παρουσιάζεται η διαδικασία προγραμματισμού ψηφιακών κυκλωμάτων με τη χρήση του λογισμικού "Quartus" και της γλώσσας "V.H.D.L.", εστιάζοντας σε τεχνικές και εργαλεία που χρησιμοποιούνται για τη σχεδίαση και την προσομοίωση. Τέλος, στο τρίτο κεφάλαιο περιγράφεται η υλοποίηση της πτυχιακής εργασίας, αναλυτικά, βήμα-βήμα από αρχή του στησίματος, μέχρι την ανάλυση των αποτελεσμάτων, αποδεικνύοντας τη λειτουργικότητα και την ακρίβεια του σχεδιασμένου συστήματος.

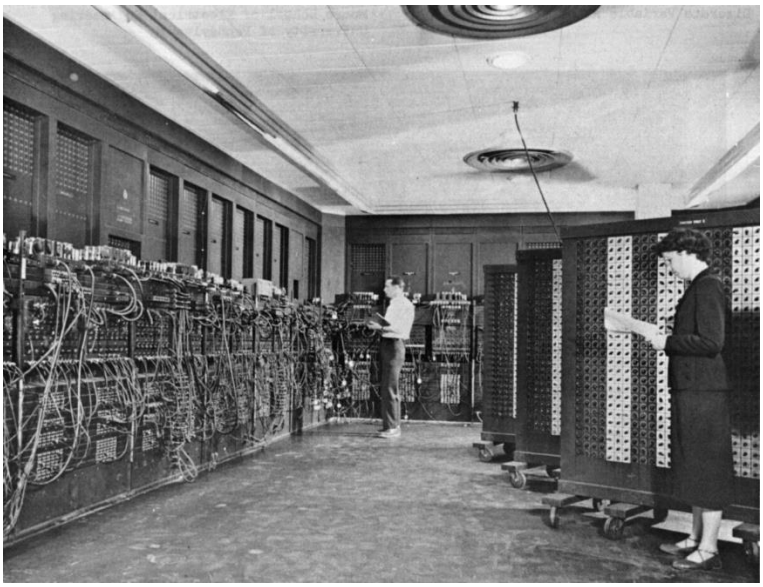
ΚΕΦΑΛΑΙΟ 1 : ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ ΓΙΑ ΤΗΝ ΔΗΜΙΟΥΡΓΙΑ ΤΩΝ ΕΠΕΞΕΡΓΑΣΤΩΝ

ΕΙΣΑΓΩΓΗ 1ου ΚΕΦΑΛΑΙΟΥ:

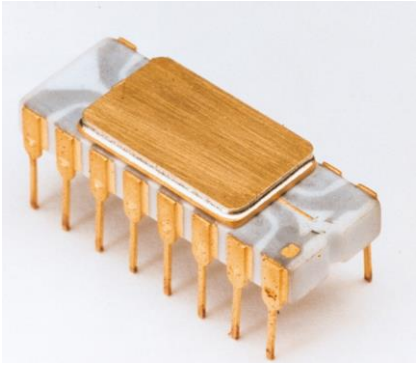
Οι επεξεργαστές αποτελούν την καρδιά της σύγχρονης τεχνολογίας, με την ιστορία τους να ξεκινά από τις πρώτες μηχανικές υπολογιστικές συσκευές και να φτάνει έως τους σημερινούς προηγμένους μικροεπεξεργαστές. Η αρχή έγινε με τα πρώτα συστήματα βασισμένα σε λυχνίες κενού, που σταδιακά αντικαταστάθηκαν από "τρανζίστορ" και ολοκληρωμένα κυκλώματα, οδηγώντας σε επαναστατική αύξηση της υπολογιστικής ισχύος. Η εμφάνιση του πρώτου μικροεπεξεργαστή, του "Intel 4004" το 1971, σηματοδότησε μια νέα εποχή στην τεχνολογία, όπου οι επεξεργαστές έγιναν πιο συμπαγείς, ισχυροί και οικονομικά προσβάσιμοι, ανοίγοντας τον δρόμο για την ευρεία χρήση τους σε υπολογιστές, φορητές συσκευές και βιομηχανικά συστήματα.

1.1 Η ΕΞΕΛΙΞΗ ΤΗΣ ΙΣΤΟΡΙΑΣ ΑΠΟ ΤΗΝ ΔΕΚΑΕΤΙΑ ΤΟΥ 1960 ΕΩΣ ΤΟ 2000

Η ιστορία των επεξεργαστών είναι μια συναρπαστική αναδρομή στην εξέλιξη της τεχνολογίας. Ξεκινάει από τις αρχές του 20ού αιώνα, όταν η έννοια του επεξεργαστή άρχισε να παίρνει σάρκα και οστά. Τα πρώτα βήματα των επεξεργαστών ήταν οι μηχανικοί υπολογιστές, όπως ο “E.N.I.A.C.” (Electronic Numerical Integrator And Computer), που κατασκευάστηκε το 1945 από τους John Mauchly και J. Presper Ecket και χρησιμοποιούσε ηλεκτρονικούς σωλήνες για την εκτέλεση υπολογισμών. Αυτές οι συσκευές ήταν τεράστιες, κατανάλωναν μεγάλη ποσότητα ενέργειας και φυσικά είχαν περιορισμένη απόδοση. Ωστόσο, η έννοια του μικροεπεξεργαστή άρχισε να παίρνει μορφή στα τέλη της δεκαετίας του '60 με την εμφάνιση των πρώτων μικροεπεξεργαστών. Ο “Intel 4004”, που κυκλοφόρησε το 1971, ήταν το πρώτο ενσωματωμένο κύκλωμα μικροεπεξεργαστή και σηματοδότησε την έναρξη της εποχής των προσωπικών υπολογιστών. [Wikipedia, 2024]



Εικόνα 1: ENIAC 1947 [Glenn A. Beck, Betty Snyder, 1947–1955 https://en.wikipedia.org/wiki/ENIAC#/media/File:Glen_Beck_and_Betty_Snyder_program_the_ENIAC_in_building_328_at_the_Ballistic_Research_Laboratory.jpg]



Εικόνα 2: Ο "Πρώτος" επεξεργαστής, Intel 4004 [https://timeline.intel.com/1971/the-first-programmable-microprocessor:-the-4004]

Η συνεχής ανάπτυξη της τεχνολογίας οδήγησε στη βελτίωση της απόδοσης, στη μείωση του μεγέθους και στην αύξηση της αποτελεσματικότητας των επεξεργαστών. Ο νόμος του "Moore" προέβλεπε ότι ο αριθμός των διαστάσεων ενός ενσωματωμένου κυκλώματος θα διπλασιαζόταν περίπου κάθε δύο χρόνια, κάτι που συνέβαινε για πολλές δεκαετίες και συνέβαλε στη σημαντική εξέλιξη των επεξεργαστών.[Hennessy, J. L., & Patterson, 2017]

Μετά την εμφάνιση του "Intel 4004" το 1971, οι εταιρείες όπως η "Intel", η "AMD", η "Motorola" και άλλες, συνέχισαν την καινοτομία στον τομέα των επεξεργαστών. Η εισαγωγή της τεχνολογίας "C.MOS" (Complementary Metal-Oxide-Semiconductor) στη δεκαετία του 80, σηματοδότησε μια σημαντική αλλαγή, καθώς αυτή η τεχνολογία επέτρεψε τη μείωση της κατανάλωσης ενέργειας και την αύξηση της ταχύτητας των επεξεργαστών. [Intel]

Η δεκαετία του 90 είχε σημαντικές καινοτομίες όπως, την αύξηση των μεγεθών των "cache" μνημών για βελτιωμένη απόδοση, καθώς και την ανάπτυξη πολυπύρηνων επεξεργαστών που επέτρεπαν την εκτέλεση περισσότερων εργασιών ταυτόχρονα. Από το 2000 και μετά, η αύξηση της ταχύτητας των επεξεργαστών και η απαίτησή τους για περισσότερη ενέργεια, ήταν η αιτία υπερθέρμανσής τους, λόγω των περιορισμών στην ψύξη. Για το λόγο αυτό, οι εταιρείες άρχισαν να εστιάζουν στην αύξηση της απόδοσης μέσω της βελτίωσης της αρχιτεκτονικής των επεξεργαστών, της προσθήκης περισσότερων πυρήνων και της βελτίωσης των δυνατοτήτων παράλληλης επεξεργασίας. Στην παρακάτω εικόνα απεικονίζεται ένας διπύρηνος επεξεργαστής της Intel. [Hennessy, J. L., & Patterson, 2017]

Εικόνα 3: Intel Core i3 2013 (2 cores 4 threads)[https://www.allhdd.com/product/intel-sr1np-processor/]

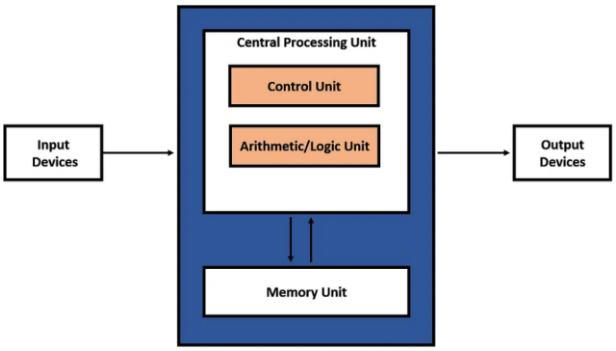
Επίσης, η μετάβαση στην τεχνολογία των 7nm, 5nm και ακόμα και πιο λεπτών κατασκευαστικών αρχιτεκτονικών, έπαιξε σημαντικό ρόλο όχι μόνο στην αύξηση της απόδοσης, αλλά και τη μείωση της κατανάλωσης ενέργειας των επεξεργαστών. Επιπλέον, η ανάπτυξη της τεχνητής νοημοσύνης και η ανάγκη για υπολογιστική ισχύ σε εφαρμογές όπως η μηχανική μάθηση και η ανάλυση μεγάλων όγκων δεδομένων έχουν οδηγήσει στη δημιουργία εξειδικευμένων επεξεργαστών για τις προαναφερθείσες ανάγκες, όπως οι “G.P.U.” (Graphics Processing Units) που χρησιμοποιούνται για παράλληλη επεξεργασία και έχουν σημαντική συμβολή σε αυτούς τους τομείς. [Paul A. Freiberger, Michael R. Swaine, 2023], [Jason Auerbach, Vineeta Agarwala, Kimber Lockhart, 2005].

1.2 ΨΗΦΙΑΚΗ ΕΞΕΛΙΞΗ ΤΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

Τα ψηφιακά συστήματα έχουν διαμορφώσει τη σύγχρονη κοινωνία με την τεχνολογική τους εξέλιξη και εφαρμογή σε διάφορους τομείς. Κατά την πάροδο των χρόνων, αυτά τα συστήματα έχουν υποστεί εξελίξεις, που τα καθιστούν απολύτως απαραίτητα για την καθημερινότητάς μας. Τη δεκαετία του '70, η ανάπτυξη των πρώτων ψηφιακών συστημάτων, όπως οι υπολογιστές, άνοιξε τον δρόμο για την ψηφιοποίηση των δεδομένων και των εργασιών. Η εξέλιξη της τεχνολογίας δημιούργησε μια συνεχή έρευνα και ανάπτυξη για νέες μεθόδους επεξεργασίας και αποθήκευσης δεδομένων.[Hennessy, J. L., & Patterson, 2017]

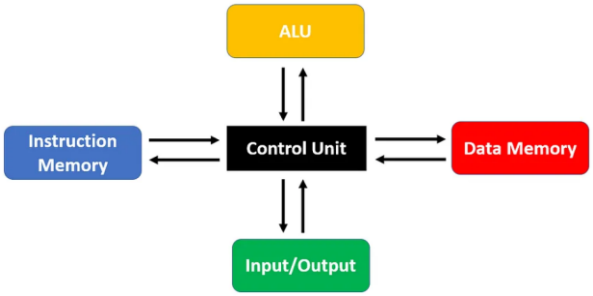
Τα πρώτα ψηφιακά συστήματα βασίζονταν σε απλές λογικές (standard logics), ενώ με την πάροδο του χρόνου, οι απαιτήσεις της κοινωνίας και η εξέλιξη της τεχνολογίας οδήγησε στη δημιουργία πιο πολύπλοκων και ισχυρών ψηφιακών συστημάτων. Η ανάγκη για τη μείωση του μεγέθους και την αύξηση της απόδοσής τους, έπαιξε καθοριστικό ρόλο στην εξέλιξή τους. Επιπροσθέτως, η ψηφιακή τεχνολογία έχει ενσωματωθεί σε πολλούς τομείς, όπως στους υπολογιστές, στα κινητά τηλέφωνα, καθώς επίσης και στα συστήματα ελέγχου στις βιομηχανίες. Η δυνατότητα διαχείρισης μεγάλου όγκου δεδομένων και η ταχύτητα επεξεργασίας των δεδομένων αυτών έχουν επαναπροσδιορίσει τον τρόπο με τον οποίο λειτουργούν οι οργανισμοί και οι σημερινές κοινωνίες. Με την πρόοδο στην τεχνητή νοημοσύνη και τη μηχανική μάθηση, τα ψηφιακά συστήματα επιτυγχάνουν αυτοματοποιημένες λειτουργίες και προσφέρουν προηγμένες λύσεις σε προβλήματα που παλαιότερα απαιτούσαν ανθρώπινη παρέμβαση, η οποία φυσικά απαιτούσε πολύ χρόνο και μεγάλο αριθμό ανθρώπινου δυναμικού. Όμως, παρά την ανάπτυξη, γενικά, τα ψηφιακά συστήματα αντιμετωπίζουν προκλήσεις ασφάλειας δεδομένων. [*Tanenbaum A. S., & Austin T. 2012*]

Οι τέσσερις κύριες αρχιτεκτονικές υπολογιστών, “R.I.S.C.” (Reduced Instruction Set Computer) “C.I.S.C.” (Complex Instruction Set Computer), “John Von Neuman” και “Harvard” αναδεικνύουν την εξέλιξη της τεχνολογίας. Κάθε μία από αυτές τις αρχιτεκτονικές έχει διαφορετικά χαρακτηριστικά, τα οποία επηρεάζουν τον τρόπο λειτουργίας των ψηφιακών συστημάτων. Η αρχιτεκτονική “Von Neumann”, ήταν μια από τις πρώτες και η πιο επιρρεπής στην εξέλιξη. Η βασική της δομή περιλάμβανε μια κεντρική μονάδα επεξεργασίας, μνήμη, σύστημα εισόδου/εξόδου και ελεγκτική μονάδα, επιτρέποντας την εκτέλεση εντολών με συγκεκριμένη σειρά, όπως φαίνεται στην παρακάτω εικόνα. (Βλ. Εικόνα 4) [Hennessy, J. L., & Patterson, 2017]



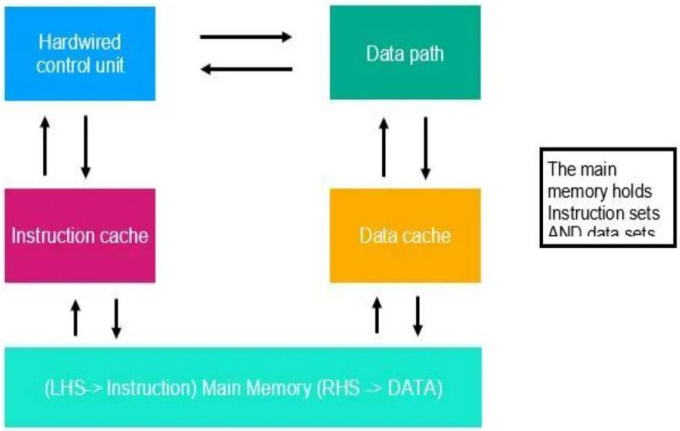
Εικόνα 4: Αρχιτεκτονική John Von Neuman[Oshadee Gangangana, https://oshadeegangangana.medium.com/von-neumann-architecture-vs-harvard-architecture-fdb20d6f2a70]

Η αρχιτεκτονική Harvard διαφοροποιείται από την Von Neumann μέσω της χρήσης διαφορετικών μονάδων μνήμης για δεδομένα και εντολές. Στην εικόνα που ακολουθεί φαίνεται η αρχιτεκτονική Harvard.



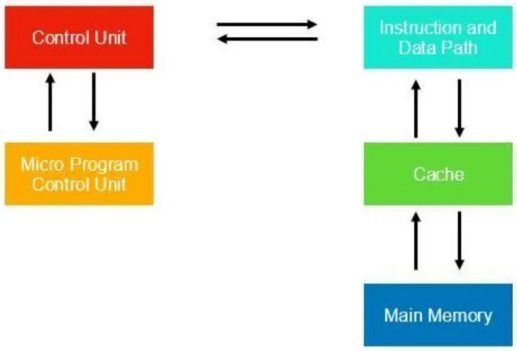
Εικόνα 5: Αρχιτεκτονική Harvard[Oshadee Gangangana, https://oshadeegangangana.medium.com/von-neumann-architecture-vs-harvard-architecture-fdb20d6f2a70]

Η “R.I.S.C.” (Reduced Instruction Set Computer) αρχιτεκτονική επικεντρώνεται σε μια απλή και συγκεκριμένη συλλογή εντολών επεξεργασίας. Η απλότητά της φαίνεται στην εικόνα που ακολουθεί. [Teach Computer Science]



Εικόνα 6: Αρχιτεκτονική RISC (Reduced Instruction Set Computer)[Teach Computer Science, https://teachcomputerscience.com/risc-and-cisc-processors/]

Η “C.I.S.C.” (Complex Instruction Set Computer) περιλαμβάνει εκτενή και πιο πολύπλοκη συλλογή εντολών, όπως προφανώς δείχνει η παρακάτω εικόνα. [Teach Computer Science]



Εικόνα 7: Αρχιτεκτονική CISC (Complex Instruction Set Computer) [Teach Computer Science, https://teachcomputerscience.com/risc-and-cisc-processors/]

Η αρχιτεκτονική “A.R.M.” (“Advanced R.I.S.C. Machine”) έχει ξεχωρίσει για τη χαμηλή κατανάλωση ενέργειας, ενώ παράλληλα έχει και την υψηλή απόδοση, χρησιμοποιώντας απλές και αποδοτικές εντολές RISC σε φορητές συσκευές.[Furber S. , 2000]

Κάθε μία από τις παραπάνω αρχιτεκτονικές έχει τα πλεονεκτήματά της, η “Von Neuman”, ευελιξία, η “Harvard”, βελτίωση της απόδοσης, η “R.I.S.C.”, εστίαση στην απλότητα, η “C.I.S.C.”, εξασφάλιση πληθώρας εντολών και τέλος, η “A.R.M.”, προσφορά καινοτόμων χαρακτηριστικών για φορητές συσκευές.

Η αρχιτεκτονική “A.R.M.”, αποτελεί μια από τις πιο σημαντικές εξελίξεις στον κόσμο των ψηφιακών συστημάτων, ιδίως στον τομέα των κινητών συσκευών και των φορητών υπολογιστών. Η αρχιτεκτονική αυτή βασίζεται σε έναν εξαιρετικά απλό και αποδοτικό σύνολο εντολών (“R.I.S.C.”), το οποίο επιτρέπει την υψηλή απόδοση εξασφαλίζοντας ταυτόχρονα χαμηλή κατανάλωση ενέργειας. [Wikipedia, 2022]

Η ευελιξία στον σχεδιασμό, η αποδοτικότητα και η εξοικονόμηση ενέργειας, καθιστά τους επεξεργαστές “A.R.M.” ιδανικούς για μικρού μεγέθους συσκευές, με μεγάλης διάρκειας μπαταρίες και φυσικά για υψηλή απόδοση, όπως είναι τα κινητά τηλέφωνα, τα τάμπλετ, οι έξυπνες τηλεοράσεις, όλες οι έξυπνες οικιακές συσκευές, τα ενσωματωμένα συστήματα στα τελευταίας τεχνολογίας αυτοκίνητα και σε άλλα μέσα μεταφορών, στην υγειονομική περίθαλψη καθώς επίσης και σε πάρα πολλές άλλες φορητές συσκευές που είναι απαραίτητες, εξυπηρετούν και διευκολύνουν την ζωή κάθε ανθρώπου στη σημερινή εποχή. [Oshadee Gangangana, 2021],[Tech Computer Science],[Furber S. , 2000].

1.3 ΕΠΕΞΕΡΓΑΣΤΗΣ ΤΟΤΕ ΕΝΑΝΤΙ ΣΗΜΕΡΑ

Γενικά, οι επεξεργαστές, από την εμφάνισή τους μέχρι και σήμερα, έχουν εξελιχθεί σημαντικά. Αρχικά, οι πρώτοι επεξεργαστές σε σύγκριση με τους σημερινούς, ήταν πολύ απλοί και λειτουργούσαν σε χαμηλές ταχύτητες. Επίσης, οι παλαιότεροι επεξεργαστές εργάζονταν με μικρότερη ακρίβεια και δεν είχαν τη δυνατότητα να εκτελούν τόσες πολλές εντολές ταυτόχρονα όσες μπορούν να εκτελούν οι σύγχρονοί επεξεργαστές. Όμως, με τα άλματα της τεχνολογίας έχει επιτραπεί η μείωση του μεγέθους των επεξεργαστών και η αύξηση της απόδοσής τους με την χρήση μικρότερων τρανζίστορ. Οι σύγχρονοι επεξεργαστές έχουν πολυπλοκότητα και επίπεδα ολοκλήρωσης που πριν από δεκαετίες θεωρούνταν αδιανόητα. Οι πολυπύρηνοι επεξεργαστές (“Ryzen 7, Intel i7”) είναι συνηθισμένοι πλέον, επιτρέποντας την εκτέλεση πολλαπλών διεργασιών ταυτόχρονα με αποτελεσματικότητα. Τα ρολόγια λειτουργίας των σύγχρονων επεξεργαστών έχουν αυξηθεί σημαντικά, επιτρέποντας ταχύτητες υψηλών συχνοτήτων που δίνουν τη δυνατότητα για εκτέλεση πολύπλοκων υπολογισμών μέσα σε πολύ σύντομο χρονικό διάστημα. Η αύξηση αυτή, προφανώς επηρεάζει και το θέμα τις ενέργειας, δηλαδή έχει αυξηθεί δραματικά η κατανάλωση ενέργειας, πράγμα το οποίο είναι λογικό. Ωστόσο, η αρχιτεκτονική των επεξεργαστών έχει βελτιωθεί σημαντικά με τη χρήση προηγμένων τεχνικών όπως, η προσθήκη προσωρινής μνήμης (“cache”) για την επιτάχυνση της πρόσβασης στα δεδομένα, η βελτίωση των προγραμματιζόμενων εντολών και η βελτιστοποίηση των προγραμμάτων εκτέλεσης. Επίσης, οι σύγχρονοι επεξεργαστές έχουν σχεδιαστεί με έμφαση στην ενεργειακή απόδοση. Όσο περισσότερη κατανάλωση ρεύματος, τόσο περισσότερη απόδοση.

Συνοψίζοντας, αν συγκρίνει κανείς τους σύγχρονους επεξεργαστές με τους αρχικούς, είναι τεχνολογικά προηγμένοι και έχουν καλύτερη απόδοση, αντικατοπτρίζοντας τη συνεχή πρόοδο και την καινοτομία στον τομέα της τεχνολογίας. Ο πρώτος επεξεργαστής που δημιουργήθηκε, για παράδειγμα, ο “Intel core 4004” το 1971 δεν μπορεί να συγκριθεί με έναν σύγχρονο επεξεργαστή της εποχής, όπως είναι ο “Ryzen 9 7950X3D” της “AMD”. Πρόκειται, φυσικά για έναν επεξεργαστή ο οποίος μπορούσε να φτάσει τα 740 kHz με 2300 “transistors” που λειτουργούσαν σε τάση 12 V καθώς και είχε μόνο έναν πυρήνα χωρίς δυνατότητες πολυνηματικής επεξεργασίας. Όπως και είναι

φυσικό, δεν μπορούν να συγκριθούν οι τιμές αυτού του επεξεργαστή με τις τιμές ενός σύγχρονου, ο οποίος έχει δεκαέξι πυρήνες, τριάντα-δύο νήματα “Threads”, βάσιμο ρολόι στα 4,2 GHz. [AMD, 2023],[Jason Auerbach, Vineeta Agarwala, Kimber Lockhart, 2005].

ΚΕΦΑΛΑΙΟ 2: ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΨΗΦΙΑΚΩΝ ΚΥΚΛΩΜΑΤΩΝ ΜΕ QUARTUS ΚΑΙ VHDL.

ΕΙΣΑΓΩΓΗ 2ου ΚΕΦΑΛΑΙΟΥ:

Ο προγραμματισμός ψηφιακών κυκλωμάτων με τη χρήση του "Quartus" και της γλώσσας "VHDL" αποτελεί ένα από τα πιο ισχυρά εργαλεία στον σχεδιασμό σύγχρονων ηλεκτρονικών συστημάτων. Το "Quartus" παρέχει ένα ολοκληρωμένο περιβάλλον ανάπτυξης για την προσομοίωση, σύνθεση και υλοποίηση ψηφιακών κυκλωμάτων, ενώ η "VHDL", ως γλώσσα περιγραφής υλικού, επιτρέπει τον ακριβή σχεδιασμό και την προσαρμογή λειτουργιών σε επίπεδο υλικού. Αυτή η συνδυαστική προσέγγιση προσφέρει τη δυνατότητα δημιουργίας αποδοτικών και ευέλικτων κυκλωμάτων, καλύπτοντας ευρεία γκάμα εφαρμογών, από απλούς ελεγκτές έως πολύπλοκα ενσωματωμένα συστήματα.

2.1 ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΨΗΦΙΑΚΩΝ ΚΥΚΛΩΜΑΤΩΝ

Ο προγραμματισμός ψηφιακών κυκλωμάτων, αποτελεί βασικό κομμάτι στο χώρο της ηλεκτρονικής και της τεχνολογίας. Για τον σχεδιασμό και την περιγραφή των λειτουργιών του, οι μηχανικοί χρησιμοποιούν γλώσσες προγραμματισμού όπως είναι η "Verilog" και η "VHDL". Μια σημαντική πτυχή του είναι η συμβατότητα με ενσωματωμένα συστήματα και "F.P.G.A.". Τα "F.P.G.A." ("Field Programmable Gate Arrays") είναι επαναπρογραμματιζόμενα λογικά κυκλώματα που μπορούν να προσαρμοστούν για να εκτελέσουν διάφορες λειτουργίες. Ο προγραμματισμός τους μέσω γλωσσών όπως το "Verilog" είναι ουσιώδης για τη δημιουργία προσαρμοσμένων κυκλωμάτων που ανταποκρίνονται σε συγκεκριμένες απαιτήσεις. Οι εφαρμογές του επεκτείνονται σε πολλούς τομείς, όπως για παράδειγμα, στην τηλεπικοινωνία, σε κινητές συσκευές, σε δίκτυα επικοινωνιών και αλλού. Επίσης, στον τομέα της υπολογιστικής τεχνολογίας, χρησιμοποιείται για τον σχεδιασμό επεξεργαστών και ενσωματωμένων συστημάτων που απαιτούνται σε υπολογιστές και συστήματα αυτοματισμών.

Τα ψηφιακά κυκλώματα προγραμματίζονται για τα συστήματα ελέγχου, τα συστήματα ασφαλείας και τις ενσωματωμένες τεχνολογίες που βρίσκονται σε σύγχρονα οχήματα. Η διαδικασία του προγραμματισμού ψηφιακών κυκλωμάτων απαιτεί συνήθως την κατανόηση των βασικών αρχών των ψηφιακών κυκλωμάτων και της λογικής σχεδίασης. Επιπλέον, η εξοικείωση με τα εργαλεία σχεδίασης και τις γλώσσες προγραμματισμού είναι ουσιώδης για την επιτυχή ανάπτυξη ψηφιακών συστημάτων. Σε πολλές περιπτώσεις, οι μηχανικοί χρησιμοποιούν εξομοιώσεις και προσομοιώσεις για να ελέγξουν την λειτουργία των ψηφιακών κυκλωμάτων πριν από την κατασκευή τους, εξοικονομώντας έτσι χρόνο και πόρους.

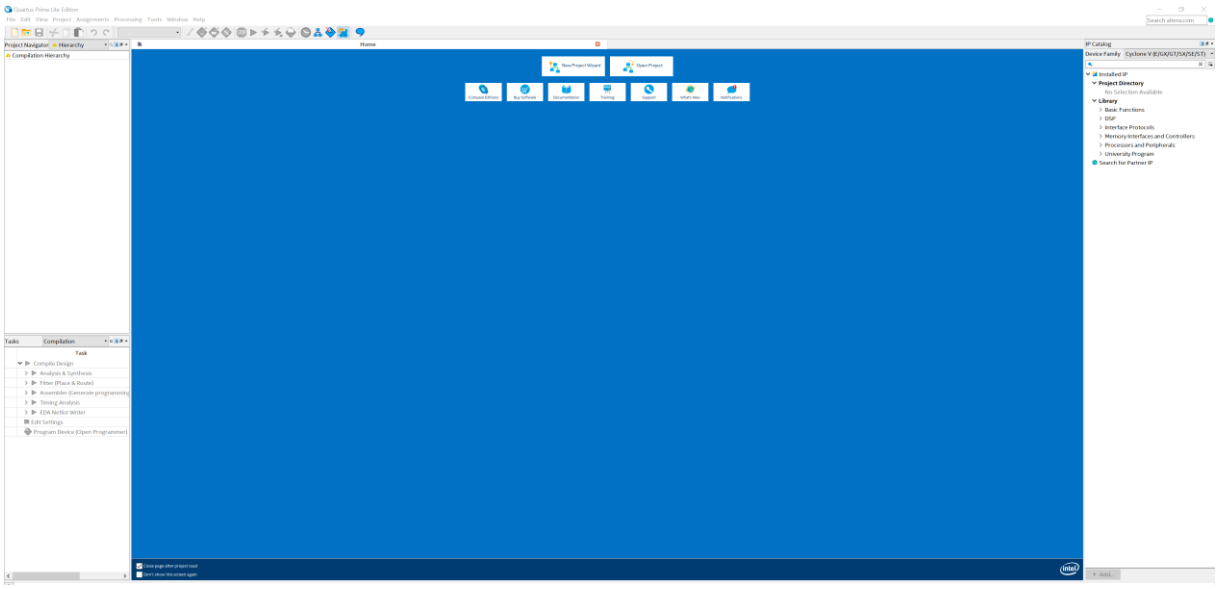
Η γνώση των γλωσσών όπως η "Verilog" και η "VHDL" που θα αναλυθεί στο επόμενο κεφάλαιο, είναι θεμελιώδης. Ο προγραμματισμός ψηφιακών κυκλωμάτων διευκολύνει την ανάπτυξη προηγμένων συστημάτων ελέγχου και αναγνώρισης σε αυτοκίνητα, ενώ στην

ιατρική βοηθά στη δημιουργία ιατρικών εργαλείων υψηλής ακρίβειας. Οι ενδιαφερόμενοι αναζητούν διαρκώς νέους τρόπους βελτίωσης της απόδοσης και της αξιοπιστίας των ψηφιακών συστημάτων μέσω του προγραμματισμού. Έτσι, είναι ένας σημαντικός κλάδος που απαιτεί εξειδικευμένες γνώσεις και εμπειρία για την ανάπτυξη προηγμένων ψηφιακών συστημάτων που χρησιμοποιούνται σε πολλούς τομείς της τεχνολογίας και της βιομηχανίας.

Εν κατακλείδι, η συνεχής εξέλιξη των τεχνολογιών και η ανάγκη για ψηφιακά συστήματα υψηλής απόδοσης και αξιοπιστίας κάνει τον προγραμματισμό αυτών των κυκλωμάτων απαραίτητο και ουσιώδη στον τομέα της τεχνολογίας. ["Intel Corporation", 2023]

2.2 QUARTUS

Το "Quartus" αποτελεί ένα εξαιρετικά ισχυρό περιβάλλον ανάπτυξης λογισμικού που είναι κρίσιμο για τον ψηφιακό σχεδιασμό κυκλωμάτων σε "F.P.G.A." ("Field Programmable Gate Arrays") και "C.P.L.D." ("Complex Programmable Logic Devices"). Ανήκει στα πιο δημοφιλή εργαλεία της κατηγορίας του, αναπτύσσοντας κυκλώματα με υψηλή απόδοση και ευελιξία, κάνοντάς το κατάλληλο για να φέρει εις πέρας αυτήν την εργασία. Το "Quartus" προσφέρει ένα εύρος εργαλείων για τον ψηφιακό σχεδιασμό και την ανάπτυξη ενσωματωμένων συστημάτων. Έχει δυνατότητες σχεδίασης σε επίπεδο "Register-Transfer Level" ("RTL") μέσω γλωσσών όπως η "VHDL" και η "Verilog", επιτρέποντας στους μηχανικούς να προγραμματίσουν και να περιγράψουν την λειτουργικότητα των κυκλωμάτων. Ένα από τα κύρια χαρακτηριστικά του "Quartus" είναι η δυνατότητα σχεδίασης με βάση τα μπλοκ. Στην παρακάτω εικόνα φαίνεται το περιβάλλον ανάπτυξης "Quartus".



Εικόνα 8: Quartus Prime Lite 20.1 Home screen

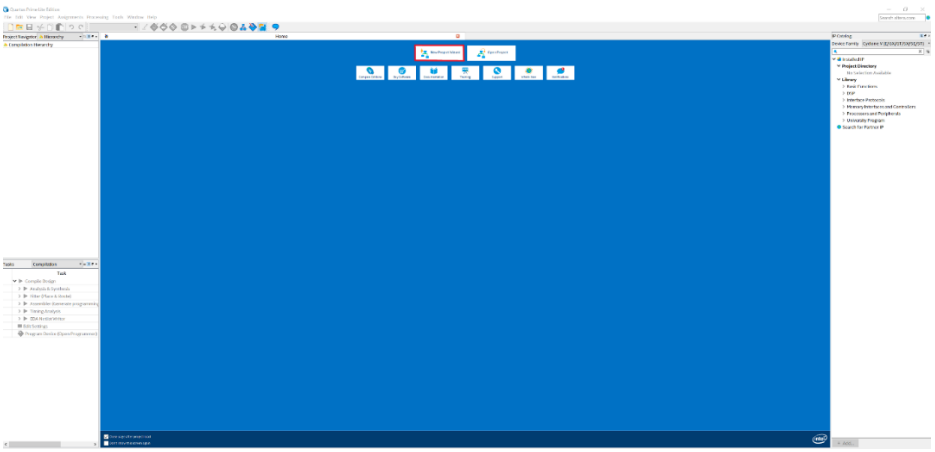
Οι μηχανικοί μπορούν να χρησιμοποιήσουν προκατασκευασμένα μπλοκ υλικού (“IP blocks”) ή να δημιουργήσουν δικά τους, συνδυάζοντάς τα, με στόχο τη δημιουργία πολύπλοκων κυκλωμάτων με μεγαλύτερη ευκολία και ταχύτητα. Το λογισμικό παρέχει επίσης εργαλεία προσομοίωσης που επιτρέπουν τον έλεγχο της σωστής λειτουργίας των κυκλωμάτων πριν από την υλοποίησή τους. Η συνθετική φάση μετατρέπει τον κώδικα σε επίπεδο λογικής που μπορεί να εφαρμοστεί στο “F.P.G.A.”. Το “Quartus” διαθέτει επίσης εργαλεία για την ανάλυση της απόδοσης και την αποσφαλμάτωση του κυκλώματος, επιτρέποντας στους χρήστες να βελτιστοποιήσουν τον σχεδιασμό τους. Επιπλέον, παρέχει εργαλεία διαχείρισης έργου για την οργάνωση και την αποθήκευση του κώδικα και των αρχείων σχεδίασης. Εκτός από την υλοποίηση κυκλωμάτων σε “F.P.G.A.” και τη διαχείριση της σχεδίασης, παρέχει επίσης εξαιρετική υποστήριξη και ενσωμάτωση με άλλα εργαλεία, δηλαδή, συνδέεται με εργαλεία προσομοίωσης όπως το “ModelSim”, επιτρέποντας να εκτελούνται πιο λεπτομερείς προσομοιώσεις και να επαληθεύεται η λογική των κυκλωμάτων πιο αποτελεσματικά πριν από την υλοποίηση των εργασιών αυτών.

Το “Quartus”, εκτός από την υλοποίηση κυκλωμάτων σε “F.P.G.A.” και τη διαχείριση της σχεδίασης, παρέχει επίσης εξαιρετική υποστήριξη και ενσωμάτωση με άλλα εργαλεία, όπως το “MATLAB/Simulink” το οποίο διευκολύνει την ολοκλήρωση διαδικασιών σχεδίασης, επιτρέποντας σε οποιονδήποτε να επωφεληθεί από την ισχυρή ανάλυση του “MATLAB” και την ευελιξία του “Simulink” για πιο περίπλοκα σχέδια. Στην παρούσα εργασία θα χρησιμοποιηθεί το εργαλείο “ModelSim”.

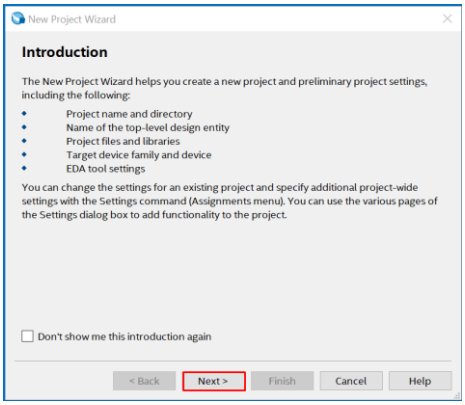
Με τη δυνατότητα αυτής της ολοκλήρωσης μεταξύ διαφόρων εργαλείων, οι μηχανικοί έχουν τη δυνατότητα να δημιουργήσουν ολοκληρωμένες διαδικασίες σχεδίασης, βελτιώνοντας την αποτελεσματικότητα και την ποιότητα των κυκλωμάτων που αναπτύσσουν. Με βάση αυτές τις συνεργασίες και την ευελιξία που προσφέρει, το “Quartus” καθίσταται βασικό εργαλείο στον τομέα του ψηφιακού σχεδιασμού κυκλωμάτων, προσφέροντας μια πλήρη και ολοκληρωμένη πλατφόρμα για τους χρήστες να δημιουργήσουν, να προσομοιώσουν και να υλοποιήσουν τις ιδέες τους με αξιοπιστία. [Intel Corporation, 2023]

Στη συγκεκριμένη εργασία, το “Quartus” χρησιμοποιήθηκε για την απλότητα και την ευελιξία που προσφέρει σε τέτοιου είδους περιπτώσεις, οι οποίες αφορούν τον σχεδιασμό

λογικών ψηφιακών κυκλωμάτων. Πιο συγκεκριμένα, στις εικόνες που ακολουθούν, αναπαρίσταται η διαδικασία για την δημιουργία ενός καινούριου project στο “Quartus” (Εικόνες 9 - 15). Όπως φαίνεται στην εικόνα εννέα, πατώντας την επιλογή “New Project Wizard” ξεκινάει η διαδικασία δημιουργίας ενός καινούριου “Project”.

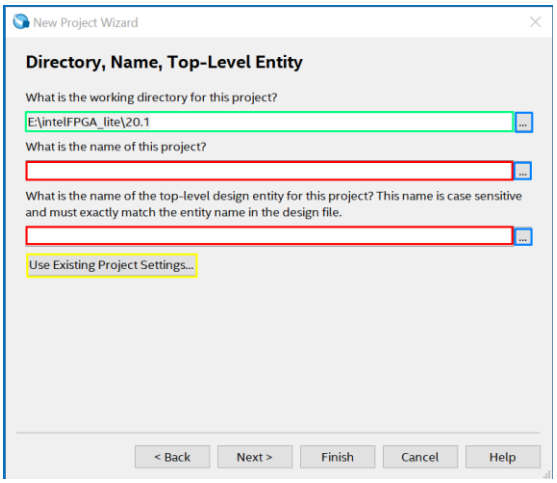


Εικόνα 9: New Project Wizard



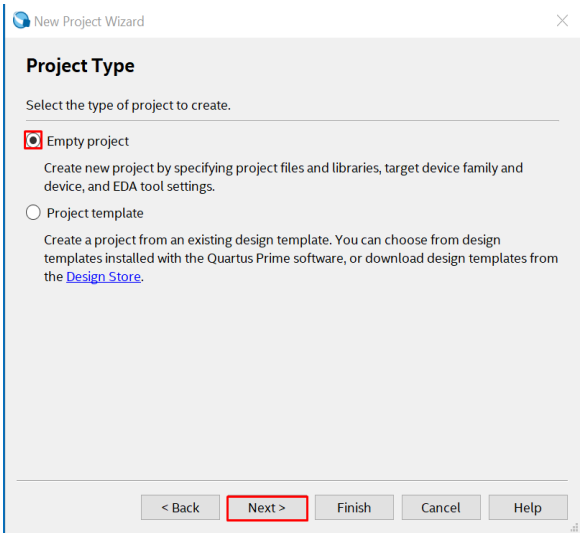
Εικόνα 10: Introduction

Το παραπάνω παράθυρο (εικόνα 10) ανοίγει αφού πατηθεί το “New Project Wizard”. Η επόμενή επιλογή είναι “Next >”.

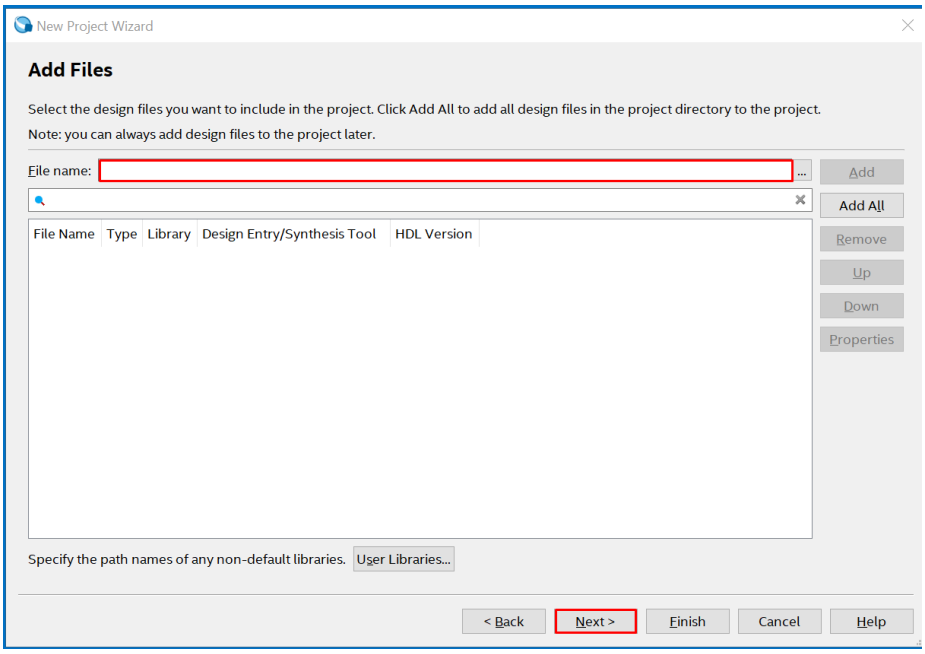


Εικόνα 11:Μονοπάτι, όνομα και Μεγίστου επιπέδου οντότητα

Στην εικόνα 11, το πράσινο κουτί δείχνει το μονοπάτι στο οποίο θα δημιουργηθεί το Project. Τα κόκκινα κουτάκια είναι το όνομα του νέου "Project", οι τελίτσες με την μπλε σήμανση είναι για να εκχωρηθεί κάποιο ήδη υπάρχον μονοπάτι και το κουτάκι με την κίτρινη σήμανση επιτρέπει να χρησιμοποιηθούν οι ρυθμίσεις ενός ήδη υπάρχοντος project. Αφού δοθεί κάποιο όνομα, στη συνέχεια επιλέγεται το "Next>". (Το "Quartus" θα ενημερώσει το χρήστη, εάν υπάρχουν άλλα project στο ίδιο "Directory" αλλά δεν έχει σημασία, αρκεί τα αρχεία να μην μοιράζονται ίδια ονόματα.)

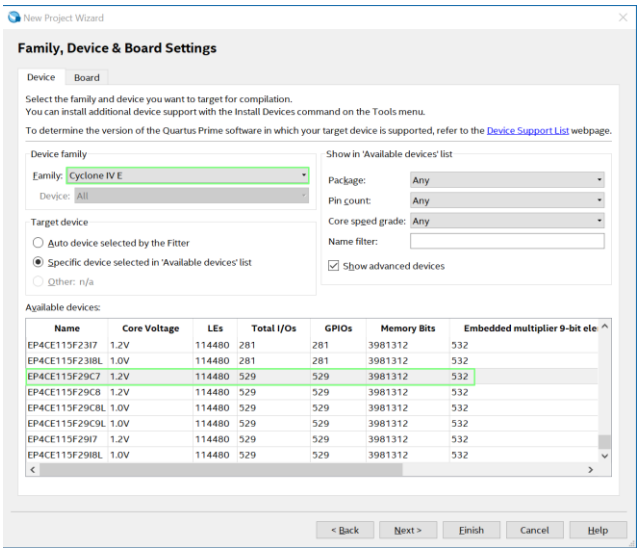


Εικόνα 12: Τύπος Project Επιλογή "Empty" project και "Next >"



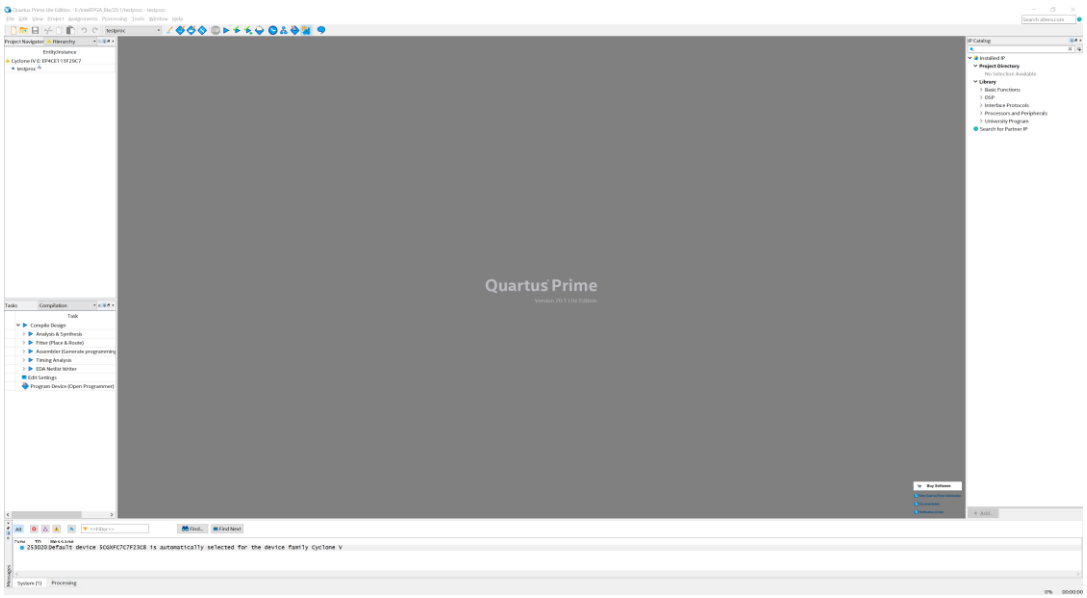
Εικόνα 13: Πρόσθεση Φακέλων

Στην παραπάνω εικόνα (Εικόνα 13) φαίνεται ότι μπορεί να επιλεχθεί κάποιο ήδη υπάρχον script που έχει προηγουμένως ετοιμαστεί, ή απλά να παραλειφθεί αυτό το βήμα και να συνεχιστεί η διαδικασία πατώντας απ' ευθείας "Next >".



Εικόνα 14: Οικογένεια, Συσκευή και ρυθμίσεις Πίνακα

Παραπάνω, (Εικόνα 14) απεικονίζεται η επιλογή οικογένειας και διαθέσιμων συσκευών. (Μπορεί και να επιλεχθεί "Cyclone V/E/GX/GT/SX/SE/ST" που συμπεριλαμβάνει και άλλες οικογένειες αλλά οι συντεινόμενες ρυθμίσεις είναι οι προαναφερθείσες).



Εικόνα 15: Επιτυχής δημιουργία κενού Project

Στην εικόνα 15, φαίνεται η οθόνη που δείχνει το “Quartus” αφού ολοκληρωθεί η δημιουργία ενός κενού “Project” επιτυχώς.

Συνολικά, το “Quartus” αποτελεί ένα απαραίτητο εργαλείο για μηχανικούς ή και καθημερινούς χρήστες που ασχολούνται με τον ψηφιακό σχεδιασμό κυκλωμάτων, παρέχοντας την υποδομή και τα εργαλεία που απαιτούνται για την ανάπτυξη πολύπλοκων ψηφιακών συστημάτων.

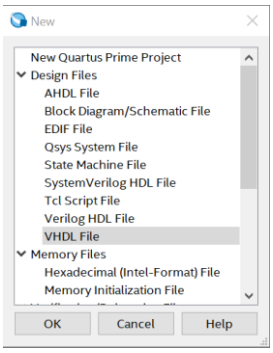
Στο κεφάλαιο που θα ακολουθήσει θα ασχοληθούμε με την γλώσσα προγραμματισμού “V.H.D.L.”.

2.3 V.H.D.L.

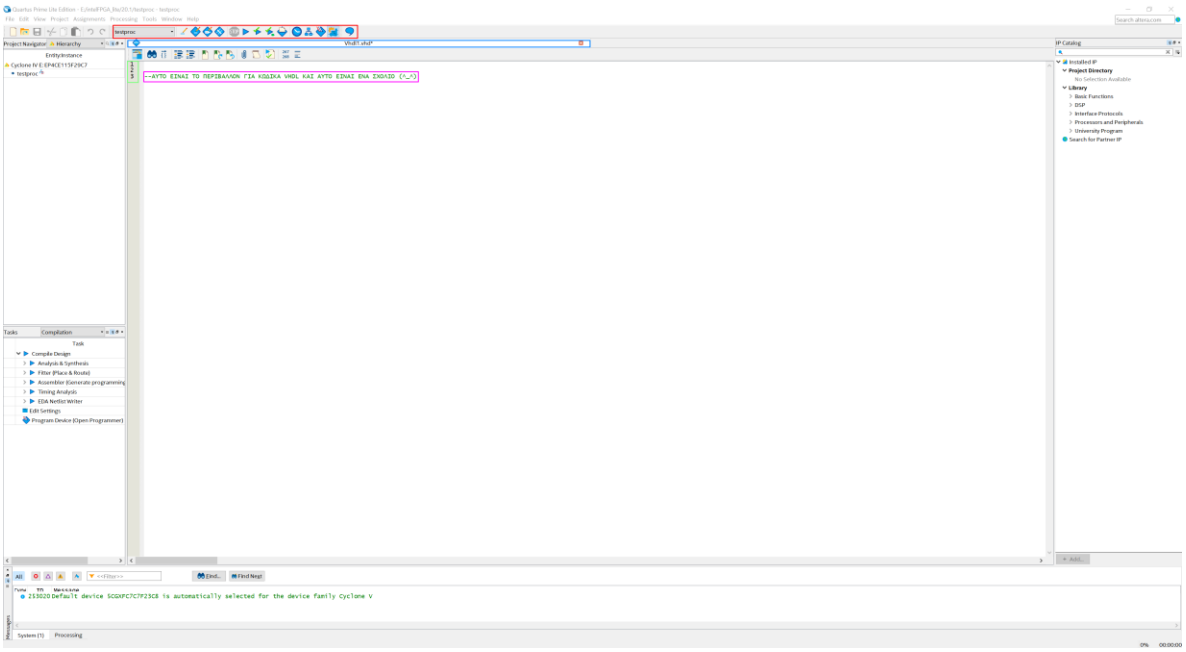
Η "V.H.D.L." ("VHSIC Hardware Description Language") είναι μία γλώσσα περιγραφής υλικού ("hardware") ψηφιακών συστημάτων, η οποία χρησιμοποιείται για τον προγραμματισμό και τον σχεδιασμό ψηφιακών κυκλωμάτων. Αποτελεί έναν τρόπο, για να περιγράψει κανείς τη λειτουργία και τη δομή ενός ψηφιακού κυκλώματος, χρησιμοποιώντας κωδικοποιημένα συντακτικά στοιχεία. Αναπτύχθηκε από την "IEEE" και έγινε πρότυπο το 1987 ("IEEE" 1076). Έκτοτε, έχει εξελιχθεί και έχει γίνει ένα από τα κυρίαρχα εργαλεία στον τομέα του ψηφιακού σχεδιασμού. Η γλώσσα αυτή επιτρέπει την περιγραφή των λογικών συναρτήσεων, των κυκλωμάτων και των συστημάτων σε υψηλό επίπεδο περιγραφής ("H.D.L."), επιτρέποντας τη σχεδίαση, τον έλεγχο και την επαλήθευση ψηφιακών κυκλωμάτων πριν την υλοποίησή τους σε φυσικό επίπεδο.

Η γλώσσα παρέχει ένα σύνολο στοιχείων και δομών που μπορούν να περιγράψουν τη λογική και τη συμπεριφορά ενός κυκλώματος. Ένας από τους βασικούς τύπους δεδομένων που χρησιμοποιεί είναι τα bits και τα αντίστοιχα "vectors", που αναπαριστούν σήματα ψηφιακών καταστάσεων. Η περιγραφή ενός κυκλώματος σε "V.H.D.L." γίνεται μέσω ονομάτων οντοτήτων ("entities") και διαδικασιών ("processes"). Οι οντότητες περιγράφουν τα επιμέρους κομμάτια ενός συστήματος, ενώ οι διαδικασίες περιέχουν τη λογική και τις λειτουργίες που εκτελούνται από το σύστημα.

Στη συνέχεια των εικόνων (Εικόνες 16-21) περιγράφεται το πώς ορίζεται ένα αρχείο σε γλώσσα "V.H.D.L."ως "Top Level Entity". Στη παρακάτω περίπτωση υπάρχουν οι προεπιλεγμένες ρυθμίσεις για αυτό. Εν συνεχεία επιλέγεται πάνω αριστερά στο "File"> "New"> "V.H.D.L. File" όπως φαίνεται στην εικόνα 16 και στην ακόλουθη, εικόνα 17, φαίνεται το περιβάλλον ανάπτυξης κώδικα με τη χρήση της "V.H.D.L.".

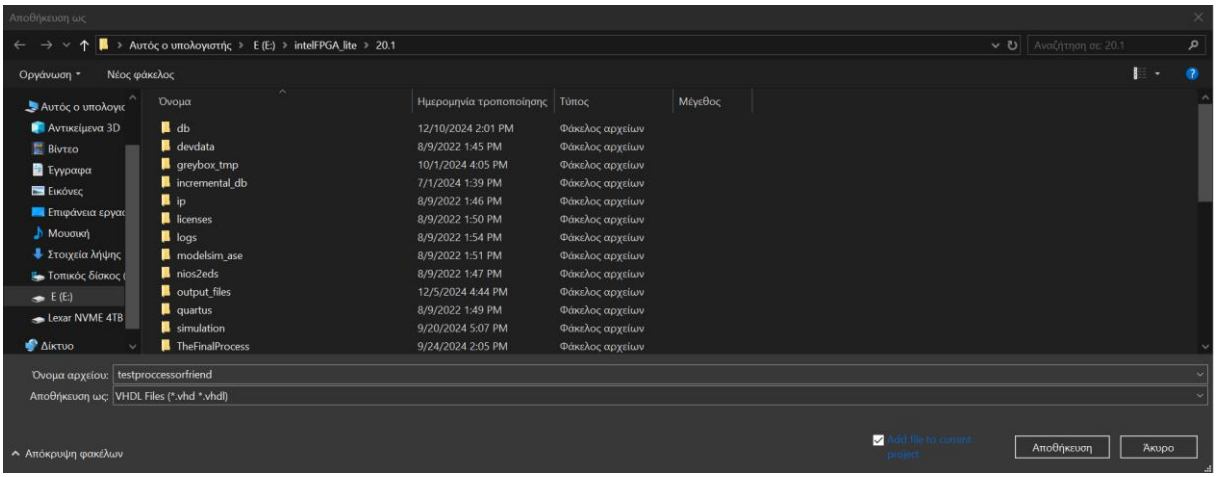


Εικόνα 16: Νέο VHDL αρχείο προς εγγραφή κώδικα



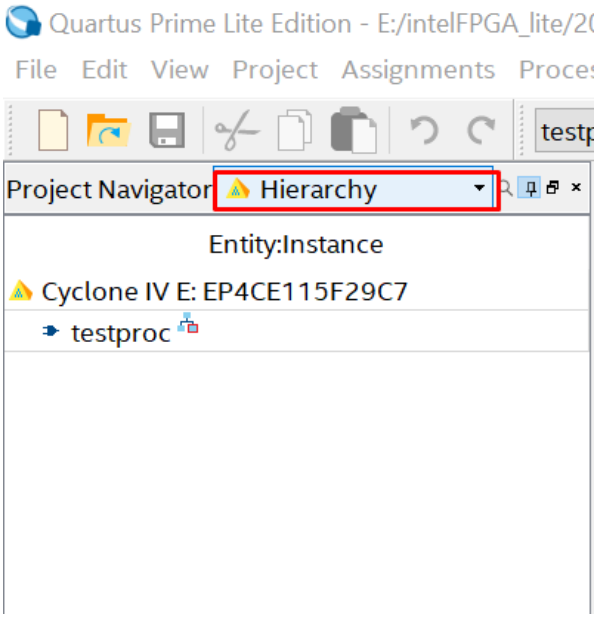
Εικόνα 17: Περιβάλλον ανάπτυξης κώδικα μετά την δημιουργία νέου αρχείου

(Κόκκινο πλαίσιο: Εργαλειοσειρά, Μπλε πλαίσιο : Παράθυρο αρχείου, Πράσινο πλαίσιο : Αρίθμηση Γραμμών κώδικα, Ροζ πλαίσιο: Σχόλιο με δύο παύλες “--”)

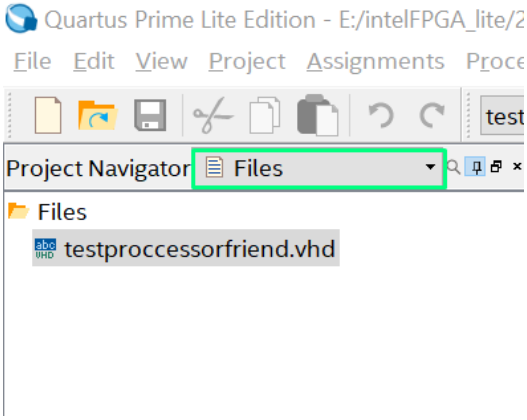


Εικόνα 18: Αποθήκευση νέου αρχείου

Παραπάνω, δίνεται όνομα στο αρχείο που έχει δημιουργηθεί και αποθηκεύεται στο αυτόματο μονοπάτι, πατώντας «Αποθήκευση».

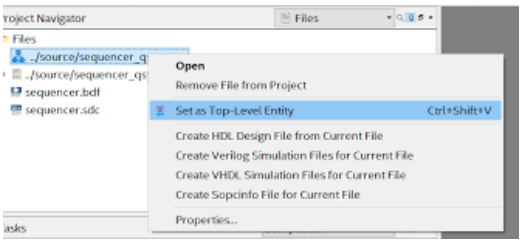


Εικόνα 19: Επιλογή Hierarchy > Files



Εικόνα 20: Επιλογή "set as top level entity"

Οι παραπάνω εικόνες (Βλ. εικόνες 19, 20), περιγράφουν τα βήματα που πρέπει να ακολουθήσει ο χρήστης για τον ορισμό του "Top Level Entity". Η παρακάτω εικόνα (Βλ. εικόνα 21), δείχνει ένα παράδειγμα ενός αρχείου να τίθεται ως "top level entity". Αυτό σημαίνει ότι είναι το κύριο μπλοκ κώδικα από το οποίο θα τρέξουν και θα κληθούν και τα υπόλοιπα "components".



Εικόνα 21: Ορισμός ως Top Level Entity

Σημείωση: Το κομμάτι κώδικα που ορίζεται ως "top level entity", θα πρέπει να έχει κάποιον τρόπο επικοινωνίας με όλες τις μορφές κώδικα στο πρόγραμμα.

Ένα από τα θεμελιώδη στοιχεία της "V.H.D.L." είναι οι διαδικασίες διασχίσεως ("simulations"), οι οποίες επιτρέπουν τον έλεγχο και την επαλήθευση της λειτουργικότητας ενός κυκλώματος πριν από την υλοποίησή του. Η "V.H.D.L." χρησιμοποιείται ευρέως στην αεροναυπηγική, την αυτοκινητοβιομηχανία, την ηλεκτρονική, τις τηλεπικοινωνίες και πολλούς άλλους τομείς, όπου απαιτείται ο σχεδιασμός και η υλοποίηση ψηφιακών κυκλωμάτων για τη δημιουργία προηγμένων και πολύπλοκων συστημάτων.[Intel, 2023]

Η γλώσσα, υποστηρίζει την ανάπτυξη διαφόρων τύπων κυκλωμάτων, συμπεριλαμβανομένων των συνδυαστικών και σειριακών, αλλά και αυτών των κυκλωμάτων με διαύλους επικοινωνίας (“bus-based circuits”). Επιτρέπει, επίσης, τη χρήση ετερογενών τύπων δεδομένων όπως “integers”, “reals”, “booleans” και άλλα, καθώς και τη δημιουργία δομημένων τύπων δεδομένων, όπως “records” και “arrays”. Μέσω της “V.H.D.L.”, οι σχεδιαστές μπορούν να περιγράψουν τη συμπεριφορά των κυκλωμάτων με μεγάλη ακρίβεια, προσφέροντας τη δυνατότητα σχεδιασμού, επαλήθευσης και εκτέλεσης μικροεπεξεργαστών, ενσωματωμένων συστημάτων, αισθητήρων, και άλλων ψηφιακών εφαρμογών. Σε γενικές γραμμές υπάρχουν τέσσερεις κύριες κατηγορίες σχεδιαστικών μονάδων στην “V.H.D.L.”, οι οποίες είναι οι ακόλουθες:

- Οντότητα (“Entity”) : Μία οντότητα (“Entity”) στη “VHDL” ορίζει τη διεπαφή (“interface”) ενός σχεδιαστικού μπλοκ. Με απλά λόγια, η οντότητα είναι σαν το "κουτί" του κυκλώματος που καθορίζει πώς επικοινωνεί με τον έξω κόσμο, χωρίς να περιλαμβάνει την εσωτερική λειτουργία.
- Αρχιτεκτονική (“Architecture”): Η αρχιτεκτονική, είναι η σχεδιαστική μονάδα που περιλαμβάνει την εσωτερική λειτουργία που τρέχει στο παρασκήνιο χωρίς να φαίνεται.
- Διαμόρφωση (“Configuration”): Η διαμόρφωση (“Configuration”) στη “VHDL” χρησιμοποιείται για να συνδέσει μία συγκεκριμένη αρχιτεκτονική (“Architecture”) μιας οντότητας (“Entity”) με τον τρόπο που αυτή χρησιμοποιείται σε ένα σχέδιο.
- Πακέτο (“Package”): Το πακέτο (“package”) στη “VHDL” είναι ένας τρόπος οργάνωσης και επαναχρησιμοποίησης κώδικα. Ένα παράδειγμα που χρησιμοποιείται πολύ είναι οι βιβλιοθήκες.[Intel, 2013]

Στην ακόλουθη εικόνα (Βλ. Εικόνα 22), αναπαρίσταται ο κώδικας για τη λογική πράξη “AND” δύο εισόδων “IN1” και “IN2”, με τη χρήση των τριών από τεσσάρων παραπάνω σχεδιαστικών μονάδων. Η χρήση βιβλιοθήκης (“library IEEE”), είναι ένα πακέτο. Η οντότητα “name_of_entity” είναι μία οντότητα στην οποία παρατηρείται και η δομή της, καθώς η αρχιτεκτονική “name_of_architecture” της οντότητας “name_of_entity” φαίνεται να εκτελεί την λογική πράξη “AND”. Παρατηρείται ότι δεν γίνεται χρήση κάποιας διαμόρφωσης. Εφόσον οι οντότητες μπορούν να έχουν περισσότερες από μία

αρχιτεκτονικές συσχετισμένες με αυτές, η διαμόρφωση εκχωρεί ένα μοναδικό όνομα σε ένα ζεύγος αρχιτεκτονικής οντοτήτων. Στη συνέχεια, αντί να χρειάζεται να επιλεχθεί ένα ζεύγος, απλώς επιλέγεται το όνομα διαμόρφωσης και το εργαλείο κατανοεί σε ποιο ζεύγος αρχιτεκτονικής οντοτήτων αναφέρεται ο χρήστης. Αυτή η διαδικασία χρησιμοποιείται περισσότερο σε πιο πολύπλοκες περιπτώσεις σχεδίασης. [Intel, 2013]

```
-- (this is a VHDL comment)

-- import std_logic from the IEEE library
library IEEE;
use IEEE.std_logic_1164.all;

-- this is the entity
entity name_of_entity is
   port (
         IN1 : in std_logic;
         IN2 : in std_logic;
         OUT1: out std_logic);
end entity name_of_entity;

-- here comes the architecture
architecture name_of_architecture of name_of_entity is

-- Internal signals and components would be defined here

begin

  OUT1 <= IN1 and IN2;

end architecture name_of_architecture;
```

Εικόνα 22: Απλό μπλοκ κώδικα σε VHDL
[https://upload.wikimedia.org/wikipedia/en/8/86/VHDL_SAMPLE_01_converted_by_Neonil.png]

Η γλώσσα διαθέτει επίσης σύνταξη για την αναδρομική περιγραφή και την παραλληλία ενεργειών, επιτρέποντας την αποτελεσματική χρήση πολυπύρηνων συστημάτων και συναρτήσεων.

Η “V.H.D.L.”, ξεχωρίζει από άλλες γλώσσες περιγραφής υλικού και προγραμματισμού ψηφιακών κυκλωμάτων, λόγω διαφόρων χαρακτηριστικών που την καθιστούν ένα ισχυρό εργαλείο στον τομέα του ψηφιακού σχεδιασμού. Πιο συγκεκριμένα διαθέτει:

i) Επίπεδο Αφαίρεσης: Η "V.H.D.L." παρέχει υψηλό επίπεδο αφαίρεσης, επιτρέποντας στους σχεδιαστές να περιγράψουν τα κυκλώματα χωρίς να ανησυχούν για τις λεπτομέρειες της φυσικής υλοποίησης.

ii) Προσομοίωση: Η "V.H.D.L." παρέχει ισχυρά εργαλεία προσομοίωσης που επιτρέπουν στους σχεδιαστές να ελέγχουν τη συμπεριφορά των κυκλωμάτων πριν από την πραγματική υλοποίησή τους.

iii) Δυνατότητα Συμπεριγραφής: Μπορεί να περιγράψει διάφορα επίπεδα υλικού, από τα πιο αφηρημένα μέχρι τα φυσικά επίπεδα.

iv) Συναρτησιακή και Χρονική Προσδιοριστικότητα: Η "V.H.D.L." επιτρέπει την περιγραφή της λογικής συνάρτησης καθώς και τη χρονική συμπεριφορά ενός συστήματος.

v) Ευελιξία: Μπορεί να χρησιμοποιηθεί για το σχεδιασμό και την περιγραφή πολλών ειδών κυκλωμάτων, από απλά συνδυαστικά μέχρι πολύπλοκα ενσωματωμένα συστήματα.

vi) Προσαρμογή στις Επιπλέον Ανάγκες: Επιτρέπει τη δημιουργία ετερογενών τύπων δεδομένων και την εκφραστικότητα μέσω της ανάπτυξης δομημένων τύπων δεδομένων.

vii) Επαλήθευση: Η δομή της "V.H.D.L." διευκολύνει την επαλήθευση και τον έλεγχο του κυκλώματος πριν από την υλοποίησή του.

Συνολικά, αποτελεί ένα ισχυρό εργαλείο για τους σχεδιαστές ψηφιακών κυκλωμάτων, προσφέροντας ένα ολοκληρωμένο περιβάλλον για τον σχεδιασμό, τον έλεγχο και την υλοποίηση πολύπλοκων συστημάτων με αξιοπιστία και ευελιξία στον ψηφιακό σχεδιασμό.

ΚΕΦΑΛΑΙΟ 3 : ΔΗΜΙΟΥΡΓΙΑ-ΥΛΟΠΟΙΗΣΗ ΕΠΕΞΕΡΓΑΣΤΗ

ΕΙΣΑΓΩΓΗ 3ου ΚΕΦΑΛΑΙΟΥ:

Η δημιουργία ενός απλού επεξεργαστή σε πρώτη φάση θα δέχεται μία δεκαέξι bit είσοδο και θα αναγνωρίζει την λειτουργία που πρέπει να πραγματοποιηθεί. Οι λειτουργίες που πρέπει να εξεταστούν είναι τέσσερεις: “mv” (move), “mvt” (move top), “add” (addition), “sub” (subtract). Η εντολή έχει έναν συγκεκριμένο τρόπο εκχώρησης δηλαδή, τα bits δεκαπέντε έως δεκατρία, είναι το Op Code και λένε τί διεργασία θα πρέπει να εκτελεστεί. Το δωδέκατο bit, αν είναι ένα, τότε αυτή η διεργασία θα χρησιμοποιήσει την άμεση τιμή από την δεκαέξι bit εντολή, δηλαδή τα bit από οκτώ έως μηδέν και θα αποθηκεύσει αυτή την τιμή στην διεύθυνση ενός καταχωρητή. Η διεύθυνση του καταχωρητή καθορίζεται, από τα bit ένδεκα έως εννέα. Αν το δωδέκατο bit είναι μηδέν, τότε θα χρησιμοποιηθεί και δεύτερος καταχωρητής. Για παράδειγμα, αν η εντολή μας είναι η: "0001000000011100" τότε δεκαπέντε έως δεκατρία = “000” (είναι η εντολή mv) δωδέκατο = “1” (θα πάρουμε την immediate τιμή δηλαδή τα bits 8 - 0), ενδέκατο έως ένατο = “000” (και το αποτέλεσμα των 9 αυτών bit θα φορτωθεί στον register 0. Δηλαδή η τιμή, “value” = όγδοο έως μηδενικό bit = “000011100”= 28) (decimal σε δεκαδικό), άρα αυτή η εντολή σημαίνει «φόρτωσε τον καταχωρητή r0 με την τιμή 28».

Οι ίδιοι κανόνες ισχύουν και σε περίπτωση πρόσθεσης, αφαίρεσης ή της εντολής mvt. Αν υπάρξει Add/Sub τότε θα χρειαστεί περισσότερος χρόνος για τις εντολές αυτές, οπότε θα χρειαστούν περισσότερα βήματα προς την εκτέλεσή τους. Αν για παράδειγμα πρέπει να εκτελεστεί η εντολή "0011001011111111" στον καταχωρητή r1, εκχωρείται η τιμή FF00 (Hexadecimal) “1111111100000000” (binary) στα bit δεκαπέντε έως οκτώ, λόγω της move top. Αυτό που κάνει η move top (mvt Op code: 001) είναι να πάρει είτε την τιμή του δεύτερου καταχωρητή και να την αντικαταστήσει στα δεκαπέντε έως οκτώ bit ή να πάρει την άμεση τιμή όπως συμβαίνει στην περίπτωσή μας και να την αντικαταστήσει σε αυτά τα bit. Η επόμενη εντολή "0101001011111111", που είναι η add, προσθέτει στο r1 την τιμή 00FF ή “0000000011111111”. Έπρεπε πρώτα το r1 να πάρει κάποια τιμή, να γίνει η πράξη και να αποθηκευτεί στο r1. Με άλλα λόγια δεν θα έφτανε ένας απλός κύκλος ρολογιού, οπότε αυτό που συμβαίνει είναι ότι χρειάζεται και άλλες καταστάσεις να διευκρινιστούν, πράγμα το οποίο είναι σχετικά απλό στην VHDL. Η αφαίρεση έχει το ίδιο σκεπτικό.

Στην δεύτερη φάση της εργασίας σε αυτές τις λειτουργίες, απλά πρέπει να προστεθεί μία μνήμη από την οποία θα παίρνονται δεκαέξι bit εντολές. Η μνήμη αυτή πρέπει να έχει τριάντα-δύο λέξεις των δεκαέξι bit και να είναι σε μορφή .mif (memory initialization file).

Για να είναι βέβαιο ότι όλα λειτουργούν σωστά, χρειάζονται μόνο οι εντολές που δίνονται από την εργασία, δηλαδή οι “mv”(move), “mvt”(move top), “add”(addition) και “sub”(subtraction), όπως και προαναφέρεται στο πρώτο μέρος της. Εάν στην προσομοίωση που θα ακολουθήσει τα αποτελέσματα είναι ίδια με αυτά που δίνονται από την εκφώνηση της εργασίας, αυτός είναι και ο τρόπος επαλήθευσης, της ορθής λειτουργίας του επεξεργαστή, όπως και φαίνεται στο κεφάλαιο αυτό.

3.1 ΔΗΜΙΟΥΡΓΙΑ ΑΠΛΟΥ ΕΠΕΞΕΡΓΑΣΤΗ ΜΕ ΤΗ ΧΡΗΣΗ TEST BENCH ΣΤΟ MODEL SIM.

Η εργασία του “Laboratory Exercise 9” παρουσιάζει ένα ψηφιακό σύστημα που περιλαμβάνει έναν απλό επεξεργαστή όπως μπορούμε να δούμε και στην παρακάτω εικόνα (Βλ. Εικόνα 23). Το σύστημα αυτό αποτελείται από μια ομάδα 16-bit καταχωρητών, έναν πολυπλέκτη, μια μονάδα πρόσθεσης/αφαίρεσης, καθώς και μια μονάδα ελέγχου που υλοποιείται ως πεπερασμένη μηχανή καταστάσεων (FSM). Οι πληροφορίες εισάγονται στο σύστημα μέσω ενός 16-bit σήματος εισόδου (DIN), το οποίο αποθηκεύεται στον καταχωρητή “IR” (Instruction Register). Μέσω ενός πολυπλέκτη 16-bit, τα δεδομένα μπορούν να μεταφερθούν από ένα σημείο του συστήματος σε άλλο, όπως από τον καταχωρητή “IR” σε έναν από τους καταχωρητές γενικής χρήσης (r0 έως r7).

Η έξοδος του πολυπλέκτη ονομάζεται Buswires και χρησιμοποιείται για τη σύνδεση διαφόρων μερών του συστήματος, επιτρέποντας τη μεταφορά δεδομένων. Η μονάδα ελέγχου “FSM” ελέγχει τις γραμμές “Select” του πολυπλέκτη, καθορίζοντας ποια δεδομένα θα τοποθετηθούν στον πολυπλέκτη και ποιος καταχωρητής θα λάβει τα δεδομένα μέσω του Buswires. Για παράδειγμα, αν η “FSM” επιλέξει τον “r0” ως την έξοδο του πολυπλέκτη και ενεργοποιήσει το “Ain”, τότε τα περιεχόμενα του “r0” θα φορτωθούν στον καταχωρητή “A” στην επόμενη ανερχόμενη ακμή του ρολογιού.

Το σύστημα μπορεί να εκτελεί διάφορες λειτουργίες σε κάθε κύκλο ρολογιού, ανάλογα με τη “FSM”. Αυτές περιλαμβάνουν την αποθήκευση δεδομένων στον “A” ή σε άλλον καταχωρητή, καθώς και την εκτέλεση αριθμητικών πράξεων μέσω της μονάδας πρόσθεσης/αφαίρεσης. Για την εκτέλεση πρόσθεσης ή αφαίρεσης, ο πολυπλέκτης αρχικά τοποθετεί έναν αριθμό 16-bit στο “Buswires”, ο οποίος αποθηκεύεται στον καταχωρητή “A”. Στη συνέχεια, ένας δεύτερος αριθμός τοποθετείται στο “Buswires” και μεταφέρεται στη μονάδα πρόσθεσης/αφαίρεσης μαζί με το περιεχόμενο του “A”. Το αποτέλεσμα αποθηκεύεται στον καταχωρητή “G” και μπορεί να μεταφερθεί μέσω του πολυπλέκτη σε άλλον καταχωρητή, ανάλογα με τις απαιτήσεις της εφαρμογής.

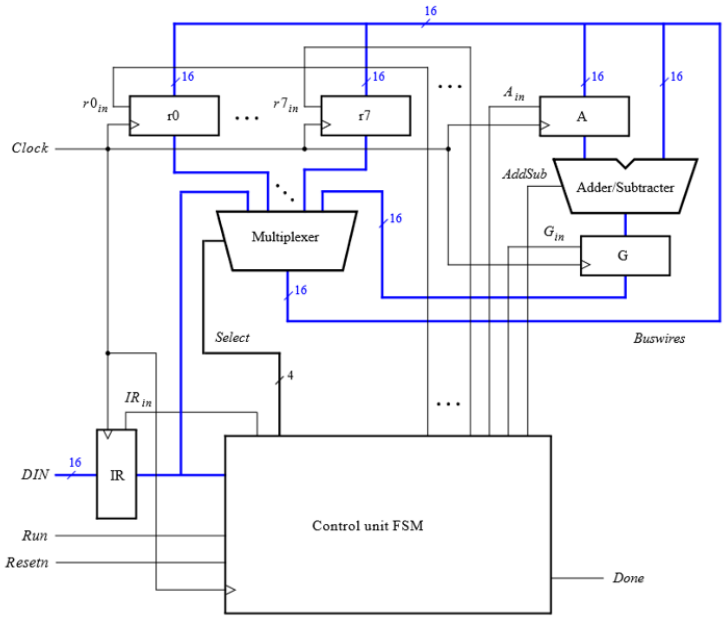


Figure 1: A digital system.

1

Εικόνα 23: Γενική σχεδίαση επεξεργαστή με σήμανση στη λειτουργία πολυπλεκτών του

Η εικόνα 24 παρουσιάζει την περιγραφή και τη λειτουργία ενός "processor", όπως αυτός που φαίνεται στην προηγούμενη εικόνα. Ο "processor" εκτελεί λειτουργίες που περιγράφονται με τη μορφή εντολών, οι οποίες παρατίθενται σε έναν πίνακα. Κάθε εντολή περιλαμβάνει το όνομά της και τη λειτουργία που εκτελεί. Ο γενικός τύπος εντολής είναι της μορφής "rX ← Op2", όπου το "rX" είναι ο καταχωρητής στον οποίο αποθηκεύεται το αποτέλεσμα, ενώ το "Op2" μπορεί να είναι είτε ένας άλλος καταχωρητής είτε μια άμεση τιμή ("#D").

Ο πίνακας παρουσιάζει τέσσερις εντολές: η εντολή "mv" αντιγράφει την τιμή ενός καταχωρητή ή μιας άμεσης τιμής σε έναν άλλο καταχωρητή. Η εντολή "mvt" χρησιμοποιείται για την αντιγραφή δεδομένων στον καταχωρητή, όπου τα πιο σημαντικά 8 "bit" ορίζονται από την άμεση τιμή. Οι εντολές "add" και "sub" εκτελούν πρόσθεση και αφαίρεση αντίστοιχα, είτε μεταξύ καταχωρητών είτε μεταξύ ενός καταχωρητή και μιας άμεσης τιμής.

Οι εντολές φορτώνονται από την είσοδο "DIN" και αποθηκεύονται στον καταχωρητή εντολών "IR". Κάθε εντολή κωδικοποιείται σε μορφή "16-bit". Αν η εντολή χρησιμοποιεί άμεση τιμή, τότε τα "bit" κωδικοποιούν τη λειτουργία, τον καταχωρητή προορισμού και την άμεση τιμή. Αντίστοιχα, αν χρησιμοποιεί καταχωρητή ως πηγή, τα "bit" κωδικοποιούν τη λειτουργία, τον καταχωρητή προορισμού και τον καταχωρητή πηγής. Το σύστημα χρησιμοποιεί επέκταση πρόσημου για την επεξεργασία άμεσων τιμών.

Η λειτουργία των αριθμητικών εντολών, όπως η πρόσθεση ή η αφαίρεση, απαιτεί περισσότερους κύκλους ρολογιού, καθώς τα δεδομένα μεταφέρονται μέσα στο σύστημα μέσω του "multiplexer" και αποθηκεύονται στους κατάλληλους καταχωρητές. Η πεπερασμένη μηχανή καταστάσεων ("FSM") ελέγχει τις απαραίτητες μεταβάσεις και καθορίζει τα σήματα ελέγχου για κάθε κύκλο. Ο πίνακας που παρατίθεται στην εικόνα 24, δείχνει τα σήματα ελέγχου που απαιτούνται για την εκτέλεση κάθε εντολής. Σημαντικά σήματα, όπως το "Select" ή το "R", καθορίζουν αν η είσοδος του "multiplexer" είναι ένας καταχωρητής ή μια άμεση τιμή.

Το κείμενο περιγράφει επίσης ένα παράδειγμα ακολουθίας εντολών που αρχικά φορτώνει την τιμή 28 στον καταχωρητή "r0", και στη συνέχεια υπολογίζει και αποθηκεύει στον καταχωρητή "r1" την τιμή -28, χρησιμοποιώντας τη μέθοδο του συμπληρώματος δύο. Με αυτόν τον τρόπο, ο "processor" μπορεί να εκτελεί διάφορες λειτουργίες που βασίζονται σε εντολές της μορφής αυτής.

Instruction		Function performed
mv	*rX, Op2*	$rX \leftarrow Op2$
mvt	*rX, #D*	$rX_{15-8} \leftarrow D_{7-0}$
add	*rX, Op2*	$rX \leftarrow rX + Op2$
sub	*rX, Op2*	$rX \leftarrow rX - Op2$

Εικόνα 24: Οδηγίες λειτουργιών εντολών εισόδου και ο πίνακας Table 1

Οι εικόνες 25 έως 28, περιγράφουν τη διαδικασία υλοποίησης του επεξεργαστή, η οποία χωρίζεται σε τρία μέρη. Αρχικά, παρέχεται ένας πίνακας που δείχνει τα σήματα ελέγχου που ενεργοποιούνται σε κάθε χρονικό βήμα ("T0" έως "T3") για τις εντολές "mv", "mvt", "add" και "sub". Αυτός ο πίνακας καθορίζει τη λογική λειτουργία του επεξεργαστή και το πώς μεταβαίνει από βήμα σε βήμα κατά την εκτέλεση κάθε εντολής (Βλ. εικόνα 25).

Στη συνέχεια, παρουσιάζεται ο σκελετός του κώδικα "VHDL" για την υλοποίηση του επεξεργαστή. Ο κώδικας περιλαμβάνει τη δήλωση της οντότητας "proc", όπου ορίζονται οι είσοδοι και οι έξοδοι, όπως το σήμα εισόδου "DIN" και το σήμα εξόδου "Done". Η αρχιτεκτονική, περιγράφει τα συστατικά μέρη του επεξεργαστή, όπως οι καταχωρητές και ο πολυπλέκτης, καθώς και τη λογική της μηχανής πεπερασμένων καταστάσεων ("FSM") για τη μετάβαση μεταξύ χρονικών βημάτων ("T0", "T1" κ.λπ.). Οι διαδικασίες και οι δηλώσεις μέσα στην αρχιτεκτονική είναι υπεύθυνες για τον έλεγχο των σημάτων ελέγχου ("control signals") και καθορίζουν πώς εκτελούνται οι εντολές. Δηλαδή, η αρχιτεκτονική περιλαμβάνει τη λογική που ενεργοποιεί τα σήματα ελέγχου, όπως τον επιλογέα ("selector") που καθορίζει την είσοδο ή την έξοδο στον "multiplexer", και κατευθύνει τη ροή των δεδομένων (Βλ. εικόνες 26, 27).

Στην εικόνα 28, στο πρώτο μέρος της εργασίας ζητείται να υλοποιηθεί και να προσομοιωθεί ο επεξεργαστής με τη χρήση της "VHDL", για να επαληθευτεί η σωστή λειτουργία του. Η εικόνα δείχνει τα αναμενόμενα αποτελέσματα της προσομοίωσης για έναν σωστά σχεδιασμένο επεξεργαστή. Συγκεκριμένα, το διάγραμμα των κυματομορφών παρουσιάζει τις τιμές των σημάτων καθώς εκτελούνται οι εντολές:

- mv r0, #28: εκτελείται στα 50 ns.
- mvt r1, #0xFF: εκτελείται στα 70 ns.
- add r0, #0xFF: ξεκινά στα 110 ns.
- sub r1, r0: ξεκινά στα 190 ns.

Τα αποτελέσματα της προσομοίωσης πρέπει να επαληθεύουν τη σωστή εκτέλεση των παραπάνω εντολών, όπως φαίνεται στις κυματομορφές. Επιπλέον, απαιτείται η ρύθμιση του προγράμματος προσομοίωσης ("ModelSim" ή "Questa Simulator") με τα κατάλληλα αρχεία, όπως το "testbench.vhd", το σενάριο εκτέλεσης "testbench.tcl" και το

αρχείο κυματομορφών "wave.do". Τέλος, η εμφάνιση των κυματομορφών όπως φαίνονται στην εικόνα αποτελεί απαραίτητη προϋπόθεση για να θεωρηθεί το πρώτο μέρος της εργασίας επιτυχημένο. Ακολουθούν οι προαναφερθείσες εξηγημένες εικόνες (Βλ. Εικόνες 25-28). [intel, exercise vhdl_lab9]

This sequence of instructions produces the same result as the single instruction `mv r1,#-28`.

	T_0	T_1	T_2	T_3
mv	IR_{in}	*Select rY* or *IR*, rX_{in}, *Done*		
mvt	IR_{in}	*Select IR*, rX_{in}, *Done*		
add	IR_{in}	*Select rX*, A_{in}	*Select rY* or *IR*, G_{in}	*Select G*, rX_{in}, *Done*
sub	IR_{in}	*Select rX*, A_{in}	*Select rY* or *IR*, *AddSub*, G_{in}	*Select G*, rX_{in}, *Done*

Table 2: Control signals asserted in each instruction/time step.

Part I

Implement the processor shown in Figure 1 using VHDL code, as follows:

1. Make a new folder for this part of the exercise. Part of the VHDL code for the processor is shown in parts (*a*) to (*c*) of Figure 2, and a more complete version of the code is provided with this exercise, in a file named *proc.vhd*. You can modify this code to suit your own coding style if desired—the provided code is just a suggested solution. Fill in the missing parts of the VHDL code to complete the design of the processor.

```
library ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY proc IS
    PORT ( DIN                 : IN  STD_LOGIC_VECTOR(15 DOWNTO 0);
           Resetn, Clock, Run  : IN  STD_LOGIC;
           Done                : BUFFER  STD_LOGIC);
END proc;

ARCHITECTURE Behavior OF proc IS
    ... declare components ...
    TYPE State_type IS (T0, T1, T2, T3);
    SIGNAL Tstep_Q, Tstep_D: State_type;
    ... declare other signals ...
    CONSTANT mv : STD_LOGIC_VECTOR(2 DOWNTO 0) := "000";
    ... encodings for each instruction ...
    CONSTANT SEL_R0 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
    ... selectors for the bus multiplexer
    CONSTANT SEL_G : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1000";
    CONSTANT SEL_IR8_IR8_0 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1001";
    CONSTANT SEL_IR7_0_0 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1010" ;
BEGIN
```

Figure 2: Skeleton VHDL code for the processor. (Part *a*)

Εικόνα 25: Αρχή σκελετού κώδικα και πίνακας Table 2

ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

```
III <= IR(15 DOWNTO 13);
Imm <= IR(12);
rX <= IR(11 DOWNTO 9);
rY <= IR(2 DOWNTO 0);
decX: dec3to8 PORT MAP (rXin, rX, Rin); -- r0 - r7 register enables

statetable: PROCESS(Tstep_Q, Run, Done)
BEGIN
    CASE Tstep_Q IS
        WHEN T0 =>     -- data is loaded into IR in this time step
            IF Run = '0' THEN Tstep_D <= T0;
            ELSE Tstep_D <= T1;
            END IF;
        WHEN T1 =>   ...
                ...
    END CASE;
END PROCESS;

controlsignals: PROCESS (Tstep_Q, III, Imm, rX, rY)
BEGIN
    rXin <= '0'; Done <= '0'; ... -- default assignments
    CASE Tstep_Q IS
        WHEN T0 => -- store DIN into IR
            IRin <= '1';
        WHEN T1 => -- define signals in time step T1
            CASE III IS
                WHEN mv =>
                    IF Imm = '0' THEN Sel <= '0' & rY;   -- mv rX, rY
                    ELSE Sel <= SEL_IR8_IR8_0;           -- mv rX, #D
                    END IF;
                    rXin <= '1';                         -- enable rX
                    Done <= '1';
                WHEN mvt =>                              -- mvt Rx, #D
                    ...
                ...
            END CASE;
        WHEN T2 => -- define signals in time step T2
            CASE III IS
                ...
            END CASE;
        WHEN T3 => -- define signals in time step T3
            CASE III IS
                ...
            END CASE;
    END CASE;
END PROCESS;

fsmflipflops: PROCESS (Clock, Resetn, Tstep_D)
BEGIN
    IF (Resetn = '0') THEN
        ...
```

Figure 2: Skeleton VHDL code for the processor. (Part *b*)

4

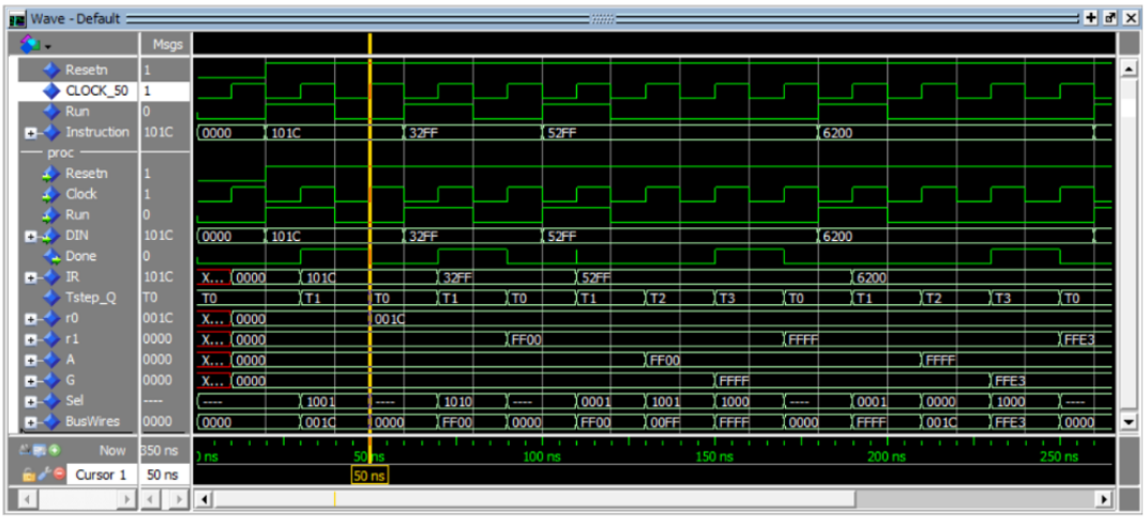
Εικόνα 26: Κώδικας διάσπασης εντολής, σήματα ελέγχου και μηχανή πεπερασμένων καταστάσεων

```
    reg_0:  regn PORT MAP (BusWires, Resetn, Rin(0), Clock, R0);
    ...
    reg_A:  regn PORT MAP (BusWires, Resetn, Ain, Clock, A);
    reg_IR: regn PORT MAP (DIN, Resetn, IRin, Clock, IR);

    alu: PROCESS (AddSub, A, BusWires)
    BEGIN
        IF AddSub = '0' THEN
            ...
    END PROCESS;
    reg_G: regn PORT MAP (Sum, Resetn, Gin, Clock, G);

    busmux: PROCESS (Sel, R0, R1, R2, R3, R4, R5, R6, R7, G, IR)
    BEGIN
        CASE Sel IS
            WHEN SEL_R0 => BusWires <= R0;
            ...
            WHEN SEL_G => BusWires <= G;
            WHEN SEL_IR8_IR8_0 => BusWires <= (15 DOWNTO 9 => IR(8)) &
                IR(8 DOWNTO 0);
            WHEN SEL_IR7_0_0 => BusWires <= IR(7 DOWNTO 0) &
                "00000000";
            WHEN OTHERS => BusWires <= (OTHERS => '0');
        END CASE;
    END PROCESS;
END Behavior;

...

ENTITY dec3to8 IS
    PORT ( E   : IN   STD_LOGIC;
           W   : IN   STD_LOGIC_VECTOR(2 DOWNTO 0);
           Y   : OUT  STD_LOGIC_VECTOR(0 TO 7));
END dec3to8;

ARCHITECTURE Behavior OF dec3to8 IS
BEGIN
    PROCESS (E, W)
    BEGIN
        IF E = '0'THEN
            Y <= "00000000";
        ELSE
            CASE W IS
                WHEN "000" => Y <= "10000000";
                ...
                WHEN "111" => Y <= "00000001";
                WHEN OTHERS => Y <= "00000000";
            END CASE;
        END IF;
    END PROCESS;
END Behavior;
```

Figure 2: Skeleton VHDL code for the processor. (Part *c*)

5

Εικόνα 27: Ενεργοποίηση καταχωρητών μέσω επιλογέα (Selector), κώδικας αποκωδικοποιητή



Εικόνα 28: Αποτελέσματα για επαλήθευση

Με βάση τον κώδικα σκελετό που δόθηκε και χρησιμοποιήθηκε, δημιουργήθηκαν οι ακόλουθοι κώδικες σαν απάντηση για το πρώτο μέρος της εργασίας.

```vhdl
-- Βιβλιοθήκες που απαιτούνται για τη χρήση τυποποιημένων λογικών και αριθμητικών λειτουργιών
library ieee; -- Εισαγωγή της βιβλιοθήκης IEEE
USE ieee.std_logic_1164.all; -- Χρήση τυποποιημένων τύπων λογικής std_logic
USE ieee.std_logic_unsigned.all; -- Χρήση αριθμητικών λειτουργιών unsigned

-- Οντότητα του επεξεργαστή (processor)
ENTITY proc IS
    PORT ( DIN                : IN  STD_LOGIC_VECTOR(15 DOWNTO 0);  -- Είσοδος δεδομένων 16-bit
           Resetn, Clock, Run  : IN  STD_LOGIC;                      -- Σήματα reset, ρολόι και εκκίνησης
           Done               : BUFFER  STD_LOGIC);                  -- Σήμα ολοκλήρωσης εντολής
END proc; -- Κλείσιμο οντότητας του επεξεργαστή

-- Αρχιτεκτονική συμπεριφοράς του επεξεργαστή
ARCHITECTURE Behavior OF proc IS
    -- Δημιουργία του αποκωδικοποιητή 3 προς 8
    COMPONENT dec3to8
        PORT ( E   : IN   STD_LOGIC;                          -- Σήμα ενεργοποίησης αποκωδικοποιητή
               W   : IN   STD_LOGIC_VECTOR(2 DOWNTO 0);       -- Είσοδος 3-bit για επιλογή
               Y   : OUT  STD_LOGIC_VECTOR(0 TO 7));          -- Έξοδος 8-bit αποκωδικοποίησης
    END COMPONENT;

    -- Δημιουργία του καταχωρητή (γενικής χρήσης)
    COMPONENT regn
        GENERIC ( n : INTEGER := 16); -- Καθορισμός πλάτους καταχωρητή
        PORT ( R                 : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);  -- Δεδομένα εισόδου
               Resetn, Rin, Clock  : IN  STD_LOGIC;                    -- Σήματα reset, εγγραφής και ρολογιού
               Q                 : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0));-- Έξοδος καταχωρητή
    END COMPONENT;

    -- Ορισμός τύπου καταστάσεων για το κύκλωμα FSM
    TYPE State_type IS (T0, T1, T2, T3); -- Καθορισμός των καταστάσεων T0 έως T3
    SIGNAL Tstep_Q, Tstep_D: State_type;  -- Σήματα για τρέχουσα (Q) και επόμενη (D) κατάσταση

    -- Σήματα για πολυπλέκτες, εγγραφές και αποτελέσματα
    SIGNAL Sel : STD_LOGIC_VECTOR(3 DOWNTO 0); -- Επιλογή για πολυπλέκτη
    SIGNAL Rin : STD_LOGIC_VECTOR(0 TO 7);     -- Ενεργοποίηση καταχωρητών R0 έως R7
    SIGNAL Sum : STD_LOGIC_VECTOR(15 DOWNTO 0);-- Αποτέλεσμα από ALU
    SIGNAL rXin, IRin, Imm, Ain, Gin, AddSub : STD_LOGIC; -- Σήματα ελέγχου για διάφορες λειτουργίες
    SIGNAL III, rX, rY : STD_LOGIC_VECTOR(2 DOWNTO 0);    -- Κωδικοί για εντολές και καταχωρητές
    SIGNAL r0, r1, r2, r3, r4, r5, r6, r7, A : STD_LOGIC_VECTOR(15 DOWNTO 0); -- Καταχωρητές
    SIGNAL G : STD_LOGIC_VECTOR(15 DOWNTO 0);             -- Καταχωρητής αποτελέσματος ALU
    SIGNAL IR, BusWires : STD_LOGIC_VECTOR(15 DOWNTO 0);  -- Εντολές (IR) και data bus (BusWires)

    -- Καθορισμός εντολών
    CONSTANT mv : STD_LOGIC_VECTOR(2 DOWNTO 0) := "000"; -- Εντολή μεταφοράς (move)
    CONSTANT mvt : STD_LOGIC_VECTOR(2 DOWNTO 0) := "001"; -- Εντολή μεταφοράς με μετατόπιση
    CONSTANT add : STD_LOGIC_VECTOR(2 DOWNTO 0) := "010"; -- Εντολή πρόσθεσης
    CONSTANT sub : STD_LOGIC_VECTOR(2 DOWNTO 0) := "011"; -- Εντολή αφαίρεσης

    -- Επιλογές πολυπλέκτη για BusWires
    CONSTANT SEL_R0 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000"; -- Επιλογή R0
    CONSTANT SEL_R1 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0001"; -- Επιλογή R1
    CONSTANT SEL_R2 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0010"; -- Επιλογή R2
    CONSTANT SEL_R3 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0011"; -- Επιλογή R3
    CONSTANT SEL_R4 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0100"; -- Επιλογή R4
    CONSTANT SEL_R5 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0101"; -- Επιλογή R5
    CONSTANT SEL_R6 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0110"; -- Επιλογή R6
    CONSTANT SEL_R7 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0111"; -- Επιλογή R7
    CONSTANT SEL_G : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1000"; -- Επιλογή καταχωρητή G
    CONSTANT SEL_IR8_IR8_0 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1001"; -- IR[8] επεκτεταμένο
    CONSTANT SEL_IR7_0_0 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1010";  -- IR[7:0] μετατοπισμένο

```

Εικόνα 29: Αρχή κώδικα του proc.vhd (Γραμμές 1-62)

Παραπάνω φαίνεται η αρχή του πηγαίου κώδικα της εργασίας. Όπως προαναφέρθηκε, ένα αρχείο σε κώδικα "vhdl" πρέπει να έχει βιβλιοθήκες, οντότητες, αντικείμενα, αρχιτεκτονική και διεργασίες. Πιο συγκεκριμένα ο παραπάνω κώδικας περιλαμβάνει την δημιουργία της οντότητας “proc” και την αρχιτεκτονική της που αποτελείται από "components", όπως είναι τα "dec3to8" και το "regn". Έπειτα, δηλώνονται μεταβλητά και σταθερά εσωτερικά σήματα, τα οποία χρησιμοποιούνται αργότερα για τις πράξεις που πρέπει να γίνουν από τον επεξεργαστή. Ας σημειωθεί επίσης ότι και πολλά από αυτά τα σήματα είναι σήματα ελέγχου. Στις δηλώσεις αυτές περιλαμβάνονται εσωτερικά σήματα για τους καταχωρητές, για την "A.L.U." και την δημιουργία καταστάσεων στις γραμμές 31 και 32 του κώδικα, οι οποίες θα χρειαστούν σε μεταγενέστερο κομμάτι για την μηχανή πεπερασμένων καταστάσεων ("Finite State Machine - FSM").

```
63  BEGIN
64      -- Απόσπασμα κωδικού εντολών από την IR
65      III <= IR(15 DOWNTO 13); -- Εξαγωγή opcode από τα bits 15-13 της IR
66      Imm <= IR(12); -- Εξαγωγή του immediate flag από το bit 12
67      rX <= IR(11 DOWNTO 9); -- Επιλογή του καταχωρητή πηγής (rX)
68      rY <= IR(2 DOWNTO 0); -- Επιλογή του καταχωρητή προορισμού (rY)
69
70      -- Αποκωδικοποιητής για ενεργοποίηση καταχωρητών
71      decX: dec3to8 PORT MAP (rXin, rX, Rin); -- Ενεργοποίηση κατάλληλου καταχωρητή μέσω αποκωδικοποιητή
72
73      -- Μηχανή Καταστάσεων (FSM) για T0-T3
74      statetable: PROCESS(Tstep_Q, Run, Done) -- FSM που διαχειρίζεται την αλλαγή καταστάσεων
75      BEGIN
76          CASE Tstep_Q IS
77              WHEN T0 =>  -- Κατάσταση T0: Φόρτωση εντολής στην IR
78                  IF Run = '0' THEN -- Αν δεν υπάρχει ενεργό Run
79                      Tstep_D <= T0; -- Παραμονή στην κατάσταση T0
80                  ELSE
81                      Tstep_D <= T1; -- Μετάβαση στην κατάσταση T1
82                  END IF;
83              WHEN T1 =>  -- Κατάσταση T1: Εκτέλεση εντολών
84                  IF Done = '1' THEN -- Αν η εντολή έχει ολοκληρωθεί
85                      Tstep_D <= T0; -- Επιστροφή στην T0 για την επόμενη εντολή
86                  ELSE
87                      Tstep_D <= T2; -- Μετάβαση στην T2 για περισσότερα βήματα
88                  END IF;
89              WHEN T2 =>  -- Κατάσταση T2: Προετοιμασία για την ολοκλήρωση
90                  Tstep_D <= T3; -- Μετάβαση στην T3
91              WHEN T3 =>  -- Κατάσταση T3: Ολοκλήρωση εντολής και επιστροφή στην T0
92                  Tstep_D <= T0; -- Επιστροφή στην αρχική κατάσταση
93          END CASE;
94      END PROCESS;
```

Εικόνα 30: Δημιουργία χρονικών βημάτων μηχανής πεπερασμένων καταστάσεων (Γραμμές 63-94 proc.vhd)

Στο παραπάνω κομμάτι κώδικα φαίνονται οι γραμμές 63-94. Σε αυτό το κομμάτι συμβαίνει η διάσπαση της κάθε εντολής, η ενεργοποίηση κατάλληλου καταχωρητή μέσω του αποκωδικοποιητή και η ενέργεια προσπέλασης καταστάσεων "T0"-"T3". Ας αναφερθεί ότι σε προηγούμενη εικόνα και συγκεκριμένα στην εικόνα 25, φαίνεται το πώς θέλουμε να γίνεται προσπέλαση των εντολών καθώς και πότε έχουμε ολοκλήρωση μιας εντολής ή μετάβαση σε επόμενη κατάσταση με την χρήση του σήματος ελέγχου "Done". Ο πίνακας της εικόνας, θα δείχνει ότι οι εντολές "mv" και "mvt" χρειάζονται μόνο μια κατάσταση για να ολοκληρωθούν, ενώ, όπως προαναφέρθηκε και στην εισαγωγή του κεφαλαίου τρία, η "add" και η "sub" χρειάζονται τρεις καταστάσεις ("T1", "T2" και "T3")

ώστε το "Done" να γίνει ένα, πράγμα το οποίο ακριβώς καθορίζει το πότε ολοκληρώνεται μια εντολή στον επεξεργαστή μας. Αυτό ακριβώς απεικονίζεται και στην εικόνα 30 με τη χρήση δύο εσωτερικών εναλλασσόμενων σημάτων "Tstep_Q" και "Tstep_D" (Τρέχουσα κατάσταση = "Q", επόμενη = "D").

```
96      -- Διαδικασία ελέγχου σημάτων (control signals) ανάλογα με την κατάσταση
97      controlsignals: PROCESS (Tstep_Q, III, Imm, rX, rY)
98      BEGIN
99          -- Αρχικοποίηση σημάτων ελέγχου
100         rXin <= '0'; Ain <= '0'; Gin <= '0'; AddSub <= '0'; IRin <= '0'; Sel <= "----";
101         Done <= '0';
102
103         CASE Tstep_Q IS
104             WHEN T0 => -- Κατάσταση T0: Φόρτωση εντολής από το DIN στην IR
105                 IRin <= '1'; -- Ενεργοποίηση εγγραφής στην IR
106
107             WHEN T1 => -- Κατάσταση T1: Καθορισμός σημάτων ανάλογα με την εντολή
108                 CASE III IS
109                     WHEN "000" =>  -- Εντολή mv
110                         IF Imm = '0' THEN -- Αν δεν είναι άμεση τιμή
111                             Sel <= "0" & rX; -- Επιλογή του καταχωρητή rX
112                         ELSE -- Αν είναι άμεση τιμή
113                             Sel <= SEL_IR8_IR8_0; -- Επιλογή άμεσης τιμής
114                         END IF;
115                         rXin <= '1'; -- Ενεργοποίηση εγγραφής στον καταχωρητή
116                         Done <= '1'; -- Ολοκλήρωση εντολής
117
118                     WHEN "001" =>  -- Εντολή mvt
119                         Sel <= SEL_IR7_0_0; -- Επιλογή IR με μετατόπιση
120                         rXin <= '1'; -- Ενεργοποίηση εγγραφής
121                         Done <= '1'; -- Ολοκλήρωση εντολής
122
123                     WHEN "010" =>  -- Εντολή add
124                         Ain <= '1'; -- Ενεργοποίηση εγγραφής στον A
125                         Sel <= "0" & rX; -- Επιλογή καταχωρητή rX
126
127                     WHEN "011" =>  -- Εντολή sub
128                         Ain <= '1'; -- Ενεργοποίηση εγγραφής στον A
129                         AddSub <= '1'; -- Σήμα για αφαίρεση
130                         Sel <= "0" & rX; -- Επιλογή καταχωρητή rX
131
132                     WHEN OTHERS =>
133                         null; -- Καμία ενέργεια για άλλες εντολές
134                 END CASE;
135
136             WHEN T2 => -- Κατάσταση T2: Προετοιμασία για πρόσθεση/αφαίρεση
137                 CASE III IS
138                     WHEN "010" =>  -- Εντολή add
139                         IF Imm = '0' THEN -- Αν δεν είναι άμεση τιμή
140                             Sel <= "0" & rY; -- Επιλογή καταχωρητή rY
141                         ELSE -- Αν είναι άμεση τιμή
142                             Sel <= SEL_IR8_IR8_0; -- Επιλογή άμεσης τιμής
143                         END IF;
144                         Gin <= '1'; -- Ενεργοποίηση εγγραφής στον G
145
146                     WHEN "011" =>  -- Εντολή sub
147                         AddSub <= '1'; -- Σήμα για αφαίρεση
148                         IF Imm = '0' THEN -- Αν δεν είναι άμεση τιμή
149                             Sel <= "0" & rY; -- Επιλογή καταχωρητή rY
150                         ELSE -- Αν είναι άμεση τιμή
151                             Sel <= SEL_IR8_IR8_0; -- Επιλογή άμεσης τιμής
152                         END IF;
153                         Gin <= '1'; -- Ενεργοποίηση εγγραφής στον G
154
155                     WHEN OTHERS =>
156                         null; -- Καμία ενέργεια για άλλες εντολές
157                 END CASE;
158
159             WHEN T3 => -- Κατάσταση T3: Ολοκλήρωση εντολών add/sub
160                 CASE III IS
161                     WHEN "010" =>  -- Εντολή add
162                         Sel <= SEL_G; -- Επιλογή του καταχωρητή G
163                         rXin <= '1'; -- Ενεργοποίηση εγγραφής
164                         Done <= '1'; -- Ολοκλήρωση εντολής
165
166                     WHEN "011" =>  -- Εντολή sub
167                         Sel <= SEL_G; -- Επιλογή του καταχωρητή G
168                         rXin <= '1'; -- Ενεργοποίηση εγγραφής
169                         Done <= '1'; -- Ολοκλήρωση εντολής
170
171                     WHEN OTHERS =>
172                         null; -- Καμία ενέργεια για άλλες εντολές
173                 END CASE;
174         END CASE;
175     END PROCESS;
176
```

Εικόνα 31: Καθορισμός ενεργειών διαδικασίας "Controlsignals" (Γραμμές 95-176 proc.vhd)

Στην εικόνα 31 φαίνονται οι ενέργειες που θα πρέπει να εκτελέσει ο επεξεργαστής κάνοντας συγκεκριμένες εσωτερικές τιμές “1”, δηλαδή "enable". Τα εσωτερικά σήματα "Rxin", "Ain", "Gin", "AddSub", "IRin", "Sel" και "Done" παίρνουν τις τιμές μηδέν ή ένα. Όπως αναφέρθηκε και στην εισαγωγή του κεφαλαίου και πιο συγκεκριμένα στο πώς χωρίζονται οι δεκαέξι "bit" εντολές, ο κώδικας περιγράφει μια μηχανή πεπερασμένων καταστάσεων ("F.S.M.") σε γλώσσα "V.H.D.L." που διαχειρίζεται σήματα ελέγχου ανάλογα με την τρέχουσα κατάσταση και την εντολή που εκτελείται. Στην αρχή της διαδικασίας αρχικοποιούνται όλα τα σήματα ελέγχου στις κατάλληλες τιμές τους. Στη συνέχεια, ανάλογα με την κατάσταση “Tstep_Q”, ορίζονται τα σήματα που απαιτούνται για την υλοποίηση των εντολών. Στην κατάσταση "T0" η τιμή από το σήμα “DIN” φορτώνεται στον καταχωρητή εντολών "IR" ενεργοποιώντας το σήμα "IRin". Στην κατάσταση “T1” η "F.S.M." καθορίζει ποια εντολή θα εκτελεστεί βάσει του "opcode" ("III"). Για την εντολή "mv", αν το “Imm” είναι μηδέν, επιλέγεται ο καταχωρητής “rX”, ενώ αν είναι ένα, επιλέγεται μια άμεση τιμή. Στη συνέχεια, ενεργοποιείται η εγγραφή στον καταχωρητή ("rXin") και η εντολή ολοκληρώνεται ("Done"). Για την εντολή “mvt”, επιλέγεται το “IR”, με κατάλληλη μετατόπιση, ενεργοποιείται η εγγραφή στον καταχωρητή ("rXin") και η εντολή ολοκληρώνεται. Για την εντολή "add", το αποτέλεσμα της οποίας απαιτεί δύο καταχωρητές ή έναν καταχωρητή και μια άμεση τιμή, ενεργοποιείται η εγγραφή στον καταχωρητή "A" ("Ain") και επιλέγεται ο καταχωρητής “rX”. Για την εντολή “sub”, ενεργοποιείται το σήμα “AddSub”, ώστε να πραγματοποιηθεί αφαίρεση. Στη συνέχεια, ενεργοποιείται η εγγραφή στον καταχωρητή “A” ("Ain") και επιλέγεται ο καταχωρητής “rX”. Στην κατάσταση "T2", η "F.S.M." προετοιμάζει την πρόσθεση ή αφαίρεση φορτώνοντας το δεύτερο τελεστέο. Για την εντολή “add”, αν το “Imm” είναι μηδέν, επιλέγεται ο καταχωρητής “rY”, ενώ αν είναι ένα, επιλέγεται μια άμεση τιμή και ενεργοποιείται η εγγραφή στον καταχωρητή “G” ("Gin"). Το ίδιο συμβαίνει και για την εντολή "sub", με τη διαφορά ότι το σήμα "AddSub" παραμένει ενεργό ώστε να δηλώνει αφαίρεση. Στην κατάσταση "T3" ολοκληρώνεται η εντολή “add” ή “sub”. Το αποτέλεσμα επιλέγεται από τον καταχωρητή “G” και γράφεται στον καταχωρητή που καθορίζεται από το "opcode", ενώ η εκτέλεση της εντολής σηματοδοτείται ως ολοκληρωμένη ("Done = 1"). Ο σχεδιασμός της "F.S.M." εξασφαλίζει τη σωστή αλληλουχία ενεργοποίησης σημάτων για την εκτέλεση εντολών και είναι επεκτάσιμος για την υποστήριξη πρόσθετων εντολών.

```
    fsmflipflops: PROCESS (Clock, Resetn, Tstep_D)
    BEGIN
        IF (Resetn = '0') THEN -- Αν το Reset είναι ενεργό, επαναφορά στην T0
  Tstep_Q <= T0; -- Ορισμός κατάστασης στην αρχική (T0)
        ELSIF (rising_edge(Clock)) THEN -- Στο ανερχόμενο μέτωπο του ρολογιού
            Tstep_Q <= Tstep_D; -- Μεταφορά της επόμενης κατάστασης στην τρέχουσα
        END IF;
    END PROCESS;

    -- Καταχωρητές r0 - r7, A, IR, και G
    reg_0:  regn PORT MAP (BusWires, Resetn, Rin(0), Clock, r0); -- Καταχωρητής r0
    reg_1:  regn PORT MAP (BusWires, Resetn, Rin(1), Clock, r1); -- Καταχωρητής r1
    reg_2:  regn PORT MAP (BusWires, Resetn, Rin(2), Clock, r2); -- Καταχωρητής r2
    reg_3:  regn PORT MAP (BusWires, Resetn, Rin(3), Clock, r3); -- Καταχωρητής r3
    reg_4:  regn PORT MAP (BusWires, Resetn, Rin(4), Clock, r4); -- Καταχωρητής r4
    reg_5:  regn PORT MAP (BusWires, Resetn, Rin(5), Clock, r5); -- Καταχωρητής r5
    reg_6:  regn PORT MAP (BusWires, Resetn, Rin(6), Clock, r6); -- Καταχωρητής r6
    reg_7:  regn PORT MAP (BusWires, Resetn, Rin(7), Clock, r7); -- Καταχωρητής r7
    reg_A:  regn PORT MAP (BusWires, Resetn, Ain, Clock, A); -- Καταχωρητής A
    reg_IR: regn PORT MAP (DIN, Resetn, IRin, Clock, IR); -- Καταχωρητής εντολών IR
    reg_G:  regn PORT MAP (Sum, Resetn, Gin, Clock, G); -- Καταχωρητής αποτελέσματος G

    -- ALU που εκτελεί πρόσθεση ή αφαίρεση
    alu: PROCESS (AddSub, A, BusWires)
    BEGIN
        IF AddSub = '0' THEN -- Αν η πράξη είναι πρόσθεση
            Sum <= A + BusWires; -- Υπολογισμός πρόσθεσης
        ELSE -- Διαφορετικά, εκτέλεση αφαίρεσης
            Sum <= A - BusWires; -- Υπολογισμός αφαίρεσης
        END IF;
    END PROCESS;

    -- Πολυπλέκτης για το BusWires που επιλέγει καταχωρητές ή άμεσες τιμές
    busmux: PROCESS (Sel, r0, r1, r2, r3, r4, r5, r6, r7, G, IR)
    BEGIN
        CASE Sel IS
            WHEN SEL_R0 => BusWires <= r0; -- Επιλογή καταχωρητή r0
            WHEN SEL_R1 => BusWires <= r1; -- Επιλογή καταχωρητή r1
            WHEN SEL_R2 => BusWires <= r2; -- Επιλογή καταχωρητή r2
            WHEN SEL_R3 => BusWires <= r3; -- Επιλογή καταχωρητή r3
            WHEN SEL_R4 => BusWires <= r4; -- Επιλογή καταχωρητή r4
            WHEN SEL_R5 => BusWires <= r5; -- Επιλογή καταχωρητή r5
            WHEN SEL_R6 => BusWires <= r6; -- Επιλογή καταχωρητή r6
            WHEN SEL_R7 => BusWires <= r7; -- Επιλογή καταχωρητή r7
            WHEN SEL_G => BusWires <= G; -- Επιλογή αποτελέσματος ALU από G
            WHEN SEL_IR8_IR8_0 => BusWires <= (15 DOWNTO 9 => IR(8)) & IR(8 DOWNTO 0); -- IR[8] επεκτεταμένο
            WHEN SEL_IR7_0_0 => BusWires <= IR(7 DOWNTO 0) & "00000000"; -- IR[7:0] μετατοπισμένο
            WHEN OTHERS => BusWires <= BusWires; -- Καμία αλλαγή στο bus
        END CASE;
    END PROCESS;
END Behavior;

-- Οντότητα και Αρχιτεκτονική για τον αποκωδικοποιητή 3 προς 8
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dec3to8 IS
    PORT ( E   : IN   STD_LOGIC;                -- Σήμα ενεργοποίησης
           W   : IN   STD_LOGIC_VECTOR(2 DOWNTO 0); -- 3-bit επιλογή
           Y   : OUT  STD_LOGIC_VECTOR(0 TO 7)); -- Έξοδος αποκωδικοποίησης
END dec3to8;

ARCHITECTURE Behavior OF dec3to8 IS
BEGIN
    PROCESS (E, W)
    BEGIN
        IF E = '0' THEN -- Αν το E είναι 0, απενεργοποίηση εξόδου
            Y <= "00000000";
        ELSE -- Διαφορετικά, αποκωδικοποίηση
            CASE W IS
                WHEN "000" => Y <= "10000000"; -- Ενεργοποίηση εξόδου 0
                WHEN "001" => Y <= "01000000"; -- Ενεργοποίηση εξόδου 1
                WHEN "010" => Y <= "00100000"; -- Ενεργοποίηση εξόδου 2
                WHEN "011" => Y <= "00010000"; -- Ενεργοποίηση εξόδου 3
                WHEN "100" => Y <= "00001000"; -- Ενεργοποίηση εξόδου 4
                WHEN "101" => Y <= "00000100"; -- Ενεργοποίηση εξόδου 5
                WHEN "110" => Y <= "00000010"; -- Ενεργοποίηση εξόδου 6
                WHEN "111" => Y <= "00000001"; -- Ενεργοποίηση εξόδου 7
                WHEN OTHERS => Y <= "00000000"; -- Default: Απενεργοποίηση εξόδου
            END CASE;
        END IF;
    END PROCESS;
END Behavior;
```

Εικόνα 32: Σχεδίαση Ψηφιακού Συστήματος με FSM, Καταχωρητές, ALU, Πολυπλέκτη και Αποκωδικοποιητή 3 προς 8. (Γραμμές 178-260 proc.vhd)

Ο παραπάνω κώδικας περιγράφει μια αρχιτεκτονική και διαδικασία σχεδίασης ψηφιακού κυκλώματος που περιλαμβάνει μια μηχανή πεπερασμένων καταστάσεων ("F.S.M."), καταχωρητές, μια αριθμητική λογική μονάδα ("A.L.U."), πολυπλέκτη ("BusMux") και αποκωδικοποιητή ("dec3to8"). Στο πρώτο τμήμα, η "FSM" ελέγχει την αλλαγή καταστάσεων, χρησιμοποιώντας ένα "flip-flop" με βάση το ρολόι ("Clock") και το

σήμα επαναφοράς ("Resetn"). Όταν το “Resetn” είναι ενεργό, η "F.S.M." επιστρέφει στην αρχική της κατάσταση ("T0"). Στο ανερχόμενο μέτωπο του ρολογιού, η επόμενη κατάσταση ("Tstep_D") μετατρέπεται ως τρέχουσα ("Tstep_Q"). Στη συνέχεια, υλοποιούνται οι καταχωρητές ("r0" έως "r7", "A", "IR" και "G") μέσω του "component" "regn". Ο κάθε καταχωρητής λαμβάνει, δεδομένα από την κοινή "bus" ("BusWires"), ένα σήμα επαναφοράς ("Resetn"), ένα σήμα εγγραφής ("Rin" ή άλλα αντίστοιχα σήματα) και το ρολόι ("Clock"). Ο καταχωρητής “IR”, χρησιμοποιείται για την αποθήκευση εντολών, ενώ ο καταχωρητής “G” αποθηκεύει αποτελέσματα αριθμητικών πράξεων.

Η “A.L.U.” πραγματοποιεί βασικές αριθμητικές πράξεις, δηλαδή πρόσθεση και αφαίρεση, ανάλογα με το σήμα ελέγχου “AddSub”. Αν το σήμα ελέγχου “AddSub” είναι μηδέν, πραγματοποιείται πρόσθεση των δεδομένων από τον καταχωρητή "A" και το "BusWires". Αν το σήμα ελέγχου “AddSub” είναι ένα, πραγματοποιείται αφαίρεση των ίδιων σημάτων. Ο πολυπλέκτης, “BusMux” καθορίζει ποιά δεδομένα θα μεταφερθούν στην κοινή "b.u.s." ("BusWires"). Η επιλογή γίνεται με βάση το σήμα ελέγχου “Sel”, το οποίο μπορεί να επιλέξει τους καταχωρητές “r0” έως “r7”, το αποτέλεσμα της "A.L.U.", “G”, ή συγκεκριμένα πεδία του “IR”. Για παράδειγμα, το “SEL_IR8_IR8_0” επεκτείνει το “IR[8]” στα ανώτερα “bits”, ενώ το “SEL_IR7_0_0” μετατοπίζει το “IR[7:0]” στα κατώτερα "bits". Ο αποκωδικοποιητής τρία προς οκτώ “dec3to8” λαμβάνει ως είσοδο το σήμα ενεργοποίησης “E” και ένα τριών-"bit" σήμα επιλογής “W”. Με αυτήν τη διαδικασία, ενεργοποιεί μία από τις οκτώ εξόδους “Y” βάσει του “W”, εφ’ όσον, φυσικά το “E” είναι ενεργό. Αν το “E” είναι μηδέν, όλες οι έξοδοι τίθενται σε μηδενική κατάσταση. Η αποκωδικοποίηση υλοποιείται μέσω μιας δομής “CASE”, η οποία αντιστοιχίζει κάθε τιμή του “W” σε μία έξοδο του “Y”. Αυτό το σχέδιο είναι βασικό για τον έλεγχο ροής δεδομένων σε αρχιτεκτονικές μικροεπεξεργαστών και μπορεί να επεκταθεί για να υποστηρίξει περισσότερες λειτουργίες και εντολές.

Ενδιαφέρουσα επισήμανση σχετικά με τη λειτουργία μηχανών πεπερασμένων καταστάσεων σε συνδυασμό με άλλα ψηφιακά υποσυστήματα, προέρχεται από μία θεμελιώδη βιβλιογραφία στον τομέα των ψηφιακών συστημάτων: "Η χρήση μηχανών πεπερασμένων καταστάσεων ("F.S.M."), ως κεντρικός ελεγκτής επιτρέπει την απλοποίηση του ελέγχου της ροής δεδομένων και τη συστηματική αλληλουχία λειτουργιών σε

πολύπλοκα κυκλώματα και ιδιαίτερα σε σχεδιάσεις μικροεπεξεργαστών." Αυτή η δήλωση υπογραμμίζει τη σημασία του συνδυασμού "F.S.M." με υποσυστήματα, όπως καταχωρητές, αποκωδικοποιητές και αριθμητική λογική μονάδα, για τη δημιουργία ευέλικτων και λειτουργικών ψηφιακών αρχιτεκτονικών. [Morris Mano & Michael Ciletti 6η Έκδοση, 2017].

```
-- Οντότητα και Αρχιτεκτονική για τον καταχωρητή
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY regn IS
    GENERIC ( n : INTEGER := 16); -- Παράμετρος για το πλάτος του καταχωρητή
    PORT ( R                  : IN  STD_LOGIC_VECTOR(n-1 DOWNTO 0); -- Είσοδος δεδομένων
           Resetn, Rin, Clock : IN  STD_LOGIC;                      -- Σήματα reset, εγγραφής και ρολογιού
           Q                  : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0));-- Έξοδος καταχωρητή
END regn;

ARCHITECTURE Behavior OF regn IS
BEGIN
    PROCESS (Clock)
    BEGIN
        IF rising_edge(Clock) THEN -- Στο ανερχόμενο μέτωπο του ρολογιού
            IF Resetn = '0' THEN -- Αν το Resetn είναι 0, επαναφορά
                Q <= (OTHERS => '0'); -- Όλα τα bits γίνονται 0
            ELSIF Rin = '1' THEN -- Αν το Rin είναι 1, εγγραφή δεδομένων
                Q <= R; -- Ενημέρωση εξόδου με την τιμή εισόδου
            END IF;
        END IF;
    END PROCESS;
END Behavior;
```

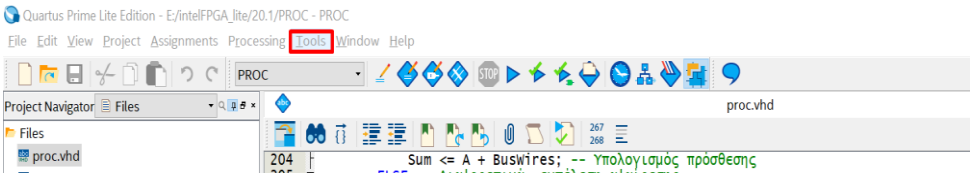
Εικόνα 33: Καταχωρητής N-bit με λειτουργίες επαναφοράς και εγγραφής δεδομένων

Ο παραπάνω κώδικας περιγράφει την οντότητα και την αρχιτεκτονική ενός γενικευμένου καταχωρητή “regn” σε γλώσσα “VHDL”. Ο καταχωρητής έχει παραμετρικό πλάτος (n), που ορίζεται μέσω του τμήματος “GENERIC” επιτρέποντας τη χρήση του με διαφορετικά μήκη δεδομένων. Η οντότητα «regn» περιλαμβάνει μια είσοδο δεδομένων “R” τύπου “STD_LOGIC_VECTOR”, ένα σήμα επαναφοράς “Resetn”, ένα σήμα εγγραφής “Rin”, ένα ρολόι “Clock” και μια έξοδο “Q”, επίσης τύπου “STD_LOGIC_VECTOR”, που αντιπροσωπεύει την τρέχουσα τιμή του καταχωρητή. Η αρχιτεκτονική “Behavior” υλοποιεί τη λειτουργία του καταχωρητή, μέσω μιας διαδικασίας, η οποία ενεργοποιείται σε κάθε ανερχόμενο μέτωπο του σήματος ρολογιού (“Clock”). Στη διαδικασία αυτή, ελέγχεται πρώτα το σήμα επαναφοράς (“Resetn”). Αν το “Resetn” είναι μηδέν, ο καταχωρητής επανέρχεται σε μηδενική κατάσταση, θέτοντας όλα τα bits της εξόδου “Q” σε μηδέν. Διαφορετικά, εάν το σήμα εγγραφής “Rin” είναι ένα, η τιμή της εισόδου “R” αποθηκεύεται στον καταχωρητή και εμφανίζεται στην έξοδο “Q”. Αν καμία από αυτές τις συνθήκες δεν ισχύει, η τιμή του καταχωρητή παραμένει αμετάβλητη. Σε ψηφιακά συστήματα, αυτός ο καταχωρητής είναι βασικό δομικό στοιχείο και χρησιμοποιείται για την αποθήκευση δεδομένων σε διάφορες χρονικές στιγμές, ανάλογα με το σήμα ρολογιού και τις συνθήκες ελέγχου. Η παραμετρική του φύση

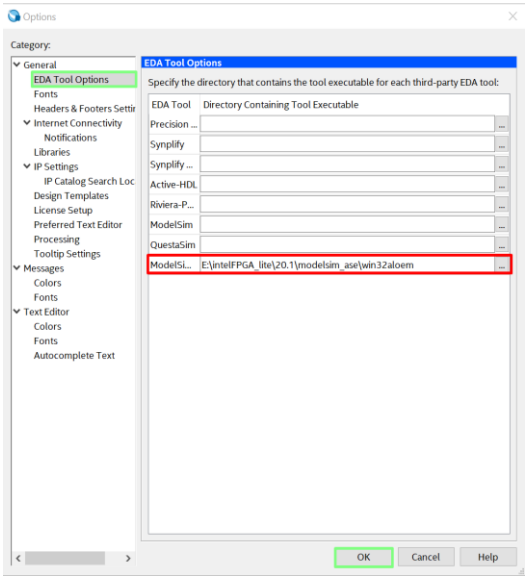
επιτρέπει την επαναχρησιμοποίησή του σε ευρύτερες σχεδιάσεις με διαφορετικές απαιτήσεις.

Στα ψηφιακά συστήματα, οι καταχωρητές είναι θεμελιώδη ακολουθιακά κυκλώματα, σχεδιασμένα για την αποθήκευση και μεταφορά δεδομένων. Αποτελούνται από μια σειρά “flip-flop”, όπου κάθε “flip-flop” αποθηκεύει ένα bit πληροφορίας. Οι καταχωρητές διακρίνονται σε διάφορους τύπους, όπως σειριακοί και παράλληλοι, ανάλογα με τον τρόπο εισαγωγής και εξαγωγής των δεδομένων. Η κατανόηση της λειτουργίας τους είναι κρίσιμη για τον σχεδιασμό και την ανάλυση σύνθετων ψηφιακών κυκλωμάτων. [Γιάννης Αμπατζόγλου 2018]

Στη συνέχεια, για να επιβεβαιωθεί ότι το pathing του “Model Sim” είναι σωστό, χρειάζεται να επιλεχθεί το “Tools” (Βλ. εικόνα 34) και πάνω-πάνω το “Options”. Μόλις ανοίξει ένα παράθυρο με διάφορες επιλογές (Βλ. εικόνα 35) επιλέγεται το “EDA Tool Options” και θα πρέπει να οριστεί το Model-sim στο μονοπάτι που έχει εγκατασταθεί το Quartus. Στην συγκεκριμένη περίπτωση, έχει εγκατασταθεί στο Δίσκο “E:” και το μονοπάτι θέσης του είναι «E:\intelFPGA_lite\20.1\modelsim_ase\win32aloem». Ο φάκελος πρέπει να βρίσκεται στο “win32aloem”.

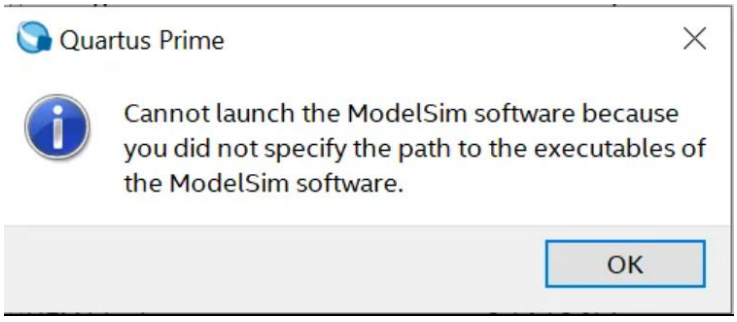


Εικόνα 34: Επιλογή tools



Εικόνα 35: Το σωστό path για το Model Sim

Προσοχή!! Εάν δεν οριστεί το μονοπάτι σωστά, θα υπάρξει το ακόλουθο Error (εικόνα 36)



Εικόνα 36: Σφάλμα, ανικανότητας εκτέλεσης Model-sim

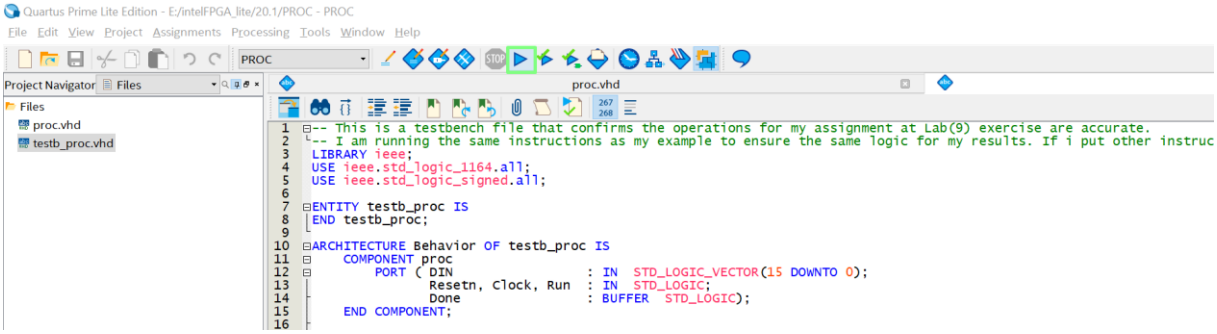
Από αυτό το σημείο και μετά, πρέπει να δημιουργηθεί ο κώδικας του φακέλου «test bench», ώστε να πάρουν τιμές οι είσοδοι και να επιστρέψουν τιμές οι έξοδοι για τις κυματομορφές του «Model Sim». Το αρχείο που δημιουργήθηκε, το "testb_proc.vhd" είναι το αρχείο του test bench, το οποίο είναι ένα μικρό κομμάτι κώδικα για την επαλήθευση της λειτουργίας του επεξεργαστή. Αυτό καλύπτει τις 4 εντολές που πρέπει να λειτουργούν, ώστε το πρόγραμμά να είναι σωστό. Πιο συγκεκριμένα η εικόνα 37 δείχνει τον κώδικα του test bench.

```vhdl
-- This is a testbench file that confirms the operations for my assignment at Lab(9) exercise are accurate.
-- I am running the same instructions as my example to ensure the same logic for my results. If i put other instructions this indicates they run perfectly.
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY testb_proc IS
END testb_proc;

ARCHITECTURE Behavior OF testb_proc IS
    COMPONENT proc
        PORT ( DIN                  : IN  STD_LOGIC_VECTOR(15 DOWNTO 0);
               Resetn, Clock, Run   : IN  STD_LOGIC;
               Done                 : BUFFER  STD_LOGIC);
    END COMPONENT;

    SIGNAL CLOCK_50 : STD_LOGIC := '0';
    SIGNAL Instruction : STD_LOGIC_VECTOR(15 DOWNTO 0);
    SIGNAL Resetn : STD_LOGIC := '0';
    SIGNAL Run, Done : STD_LOGIC;

    CONSTANT clock_period : TIME := 20 ns;
BEGIN
    U1: proc PORT MAP (DIN => Instruction, Resetn => Resetn, Clock => CLOCK_50,
                       Run => Run, Done => Done);

    clock_process: PROCESS
    BEGIN
        CLOCK_50 <= '0';
        WAIT FOR clock_period / 2;
        CLOCK_50 <= '1';
        WAIT FOR clock_period / 2;
    END PROCESS;

    vectors: PROCESS
    BEGIN
        Resetn <= '0'; Run <= '0'; Instruction <= "0000000000000000";
        WAIT FOR 20 ns;
        Resetn <= '1'; Run <= '1'; Instruction <= "0001000000011100";  -- mv  r0, #28
        WAIT FOR 20 ns;
        Run <= '0';
        WAIT FOR 20 ns;
        Run <= '1'; Instruction <= "0011001011111111";  -- mvt r1, #0xFF00
        WAIT FOR 20 ns;
        Run <= '0';
        WAIT FOR 20 ns;
        Run <= '1'; Instruction <= "0101001011111111";   -- add r1, #0xFF
        WAIT FOR 20 ns;
        Run <= '0';
        WAIT FOR 60 ns;
        Run <= '1'; Instruction <= "0110001000000000";  -- sub r1, r0
        WAIT FOR 20 ns;
        Run <= '0';
        WAIT FOR 60 ns;
        Run <= '1'; Instruction <= "0101001000000001";  -- add r1, #1
        WAIT FOR 20 ns;
        Run <= '0';
        WAIT;
    END PROCESS;
END;
```

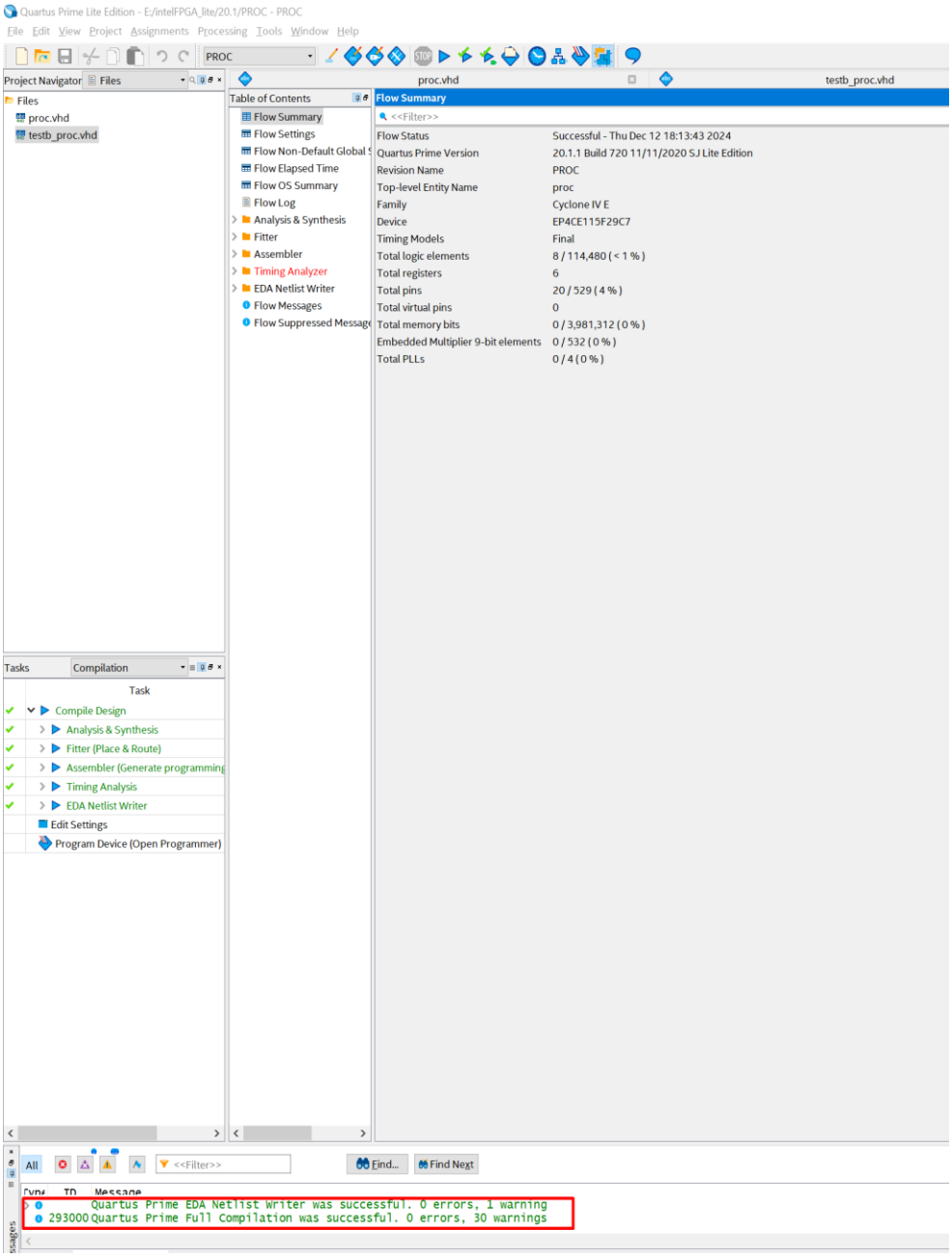
Εικόνα 37: Κώδικας testbench για τις εντολές mv, mvt, add και sub

Ο παραπάνω κώδικας αποτελεί ένα αρχείο testbench για την επαλήθευση των λειτουργιών μιας μονάδας επεξεργαστή (“proc”) που χρησιμοποιείται σε μια εργασία εργαστηρίου. Το “testbench” αρχίζει με την εισαγωγή απαραίτητων βιβλιοθηκών της “VHDL” για την υποστήριξη λογικών πράξεων και προτύπων σήματος. Στη συνέχεια, δηλώνεται η οντότητα “testb_proc” η οποία δεν έχει θύρες, καθώς λειτουργεί αποκλειστικά για την προσομοίωση της συμπεριφοράς του επεξεργαστή. Μέσα στην αρχιτεκτονική “Behavior”, γίνεται η δήλωση της μονάδας υπό έλεγχο, της μονάδας «proc», με τις αντίστοιχες θύρες εισόδου και εξόδου όπως δεδομένα εισόδου “DIN”, σήμα επαναφοράς “Resetn”, ρολόι “Clock”, σήμα εκτέλεσης “Run” και σήμα ολοκλήρωσης “Done”. Δηλώνονται τα σήματα και μια σταθερά για την περίοδο του ρολογιού που είναι “20ns”. Στη συνέχεια γίνεται αντιστοίχιση των θυρών της μονάδας proc με τα τοπικά σήματα μέσω του “component instantiation”. Έπειτα, ορίζεται μια διαδικασία “clock_process” που δημιουργεί το σήμα του ρολογιού “CLOCK_50” αλλάζοντας την τιμή του κάθε μισή περίοδο (10 ns), ούτως ώστε να προσομοιώνεται η λειτουργία ενός ρολογιού.

Επίσης, υπάρχει μια άλλη διαδικασία “vectors” όπου καθορίζονται οι συνθήκες εισόδου για τη δοκιμή. Αρχικά, για την αρχικοποίηση και επαναφορά του επεξαργαστή, τα σήματα “Resetn” και “Run” είναι μηδέν, με αποτέλεσμα η πρώτη εντολή να είναι μηδενική. Μετά από “20 ns”, τα “Resetn” και “Run” τίθενται σε 1 και δίνεται η πρώτη εντολή “mv”, για να φορτωθεί η τιμή 28 στον καταχωρητή “r0”. Έπειτα, το “Run” γίνεται μηδενικό και επαναφέρεται για να εκτελεστεί η επόμενη εντολή mvt που φορτώνει την τιμή “0xFF00” στον “r1”. Ακολουθούν εντολές add για να προστεθεί η τιμή “0xFF” στον “r1” και “sub” για να αφαιρεθεί η τιμή του “r0” από τον “r1”. Στη συνέχεια, δίνεται ακόμα μια εντολή add για να προστεθεί η τιμή ένα στον “r1”. Μετά από κάθε εντολή, η διαδικασία “vectors” συνεχίζει να θέτει το “Run” σε μηδέν, για να εξασφαλίζεται η σωστή εκτέλεση και ολοκλήρωση κάθε λειτουργίας. Αυτό το “testbench”, προσομοιώνει τη λειτουργία του επεξεργαστή με συγκεκριμένες εντολές, ώστε να επαληθευτεί η σωστή εκτέλεσή τους και η αλληλουχία λειτουργίας.



Εικόνα 38: Compile ελέγχου

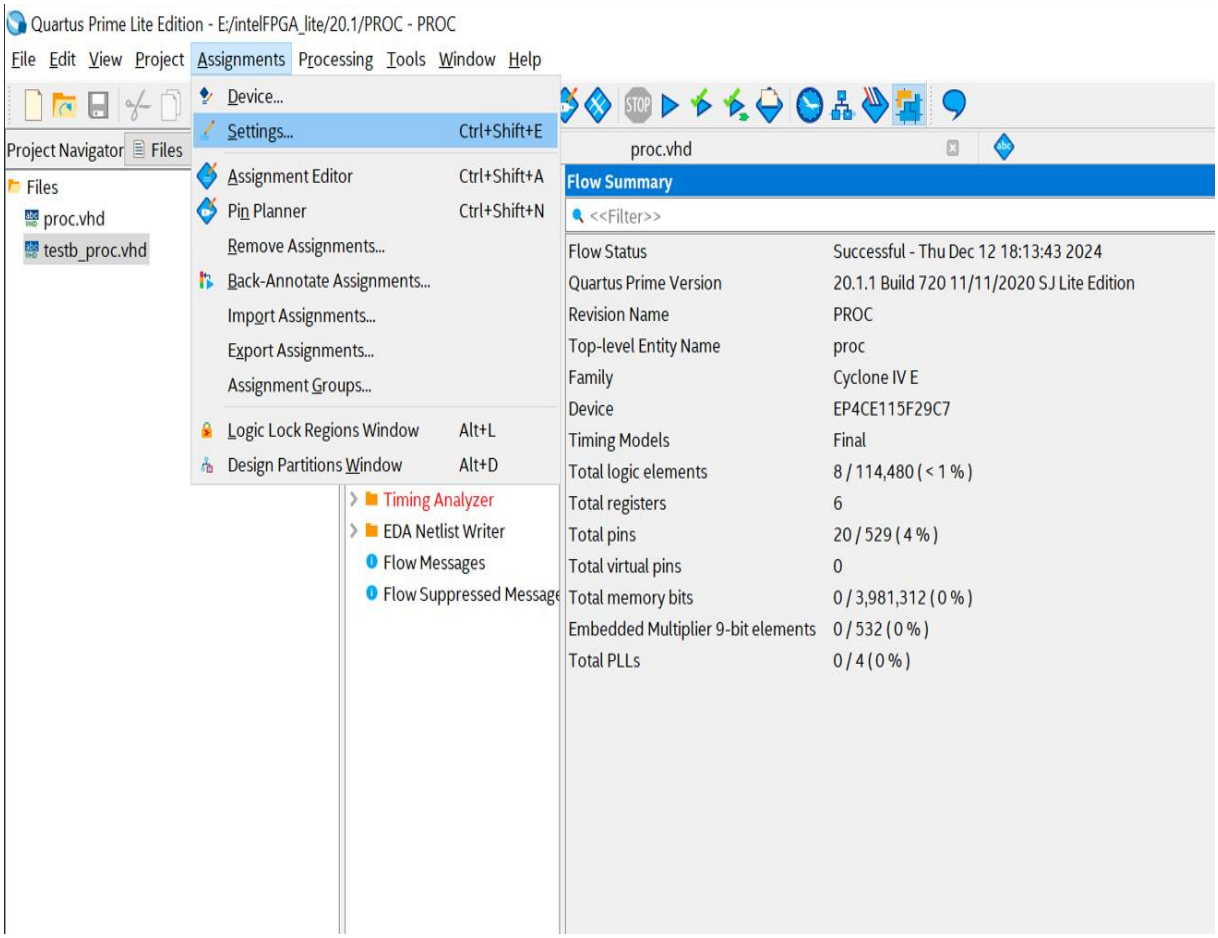


Εικόνα 39: Compilation Report πράσινο

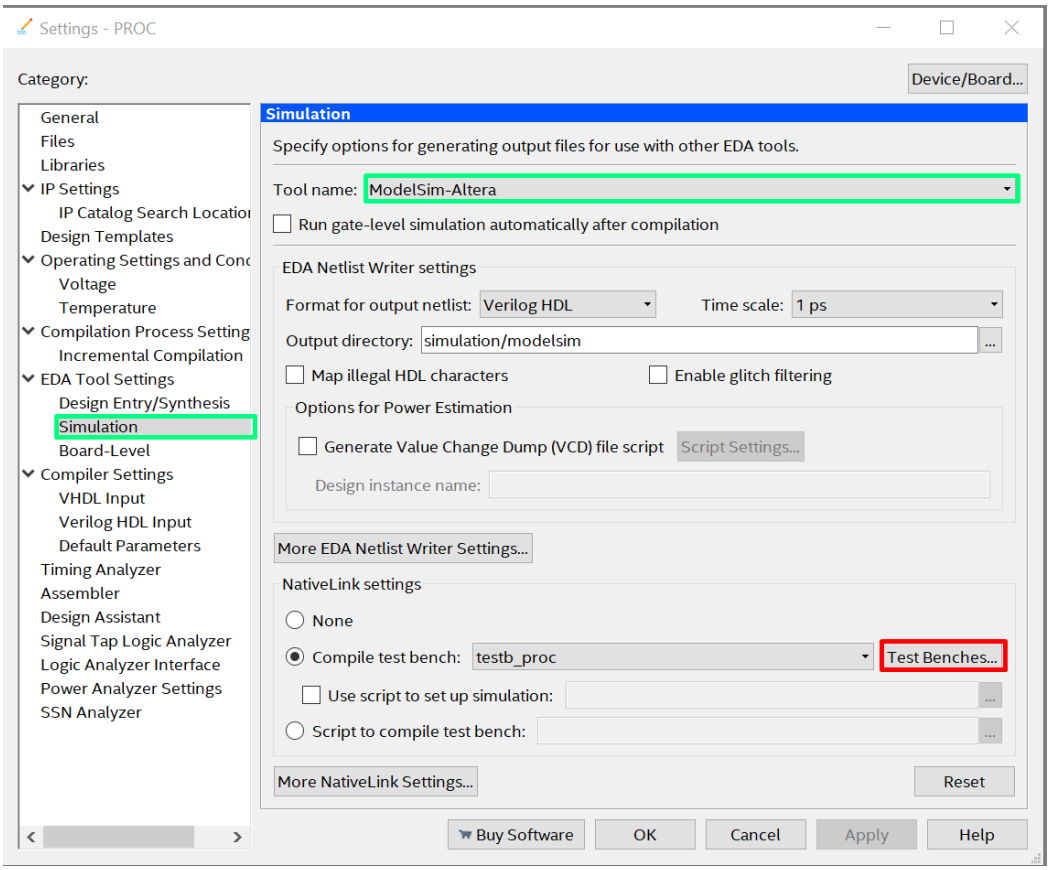
Η εικόνα 39 δείχνει πότε δεν υπάρχει κάποιο λογικό λάθος στον κώδικά και όλα είναι σωστά από την άποψη συντακτικού.

Σε αυτό το σημείο, για να γίνει έλεγχος των αποτελεσμάτων στο “model-sim”, πρέπει να οριστεί ο κώδικας του “testb_proc.vhd” ως αρχείο “test bench”. Για να γίνει αυτό, θα πρέπει στο “Assignments” > “Settings”, όπως φαίνεται παρακάτω (Βλ. εικόνα 40) να επιλεχθεί στο “EDA Tool Settings” > “Simulation” (Βλ. εικόνα 41)> “Compile test bench” >”Test Benches”>”New” (Βλ. εικόνα 42) > “Test Bench name”> το μονοπάτι του

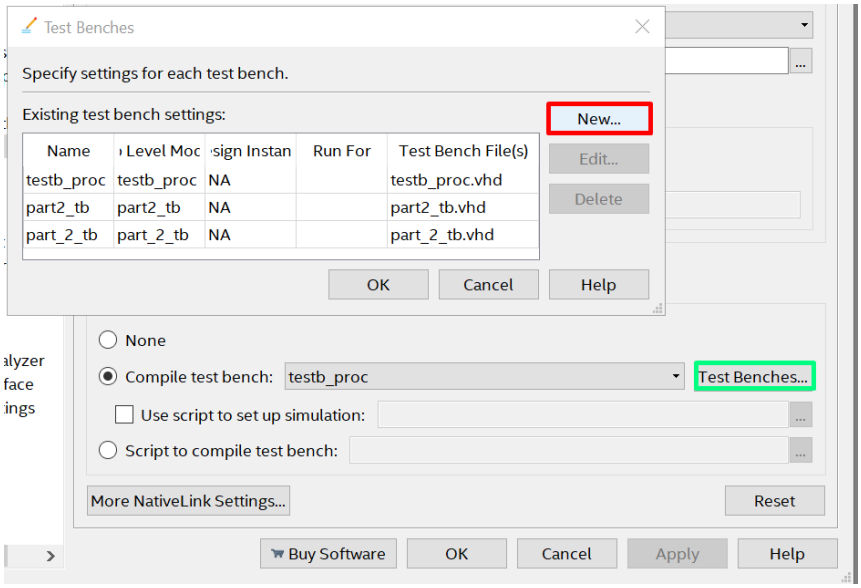
ονόματος του φακέλου ".vhd" που είναι το "test bench" (Βλ. εικόνα 43)> "Ok" > "Ok" > "Apply"> "Ok".



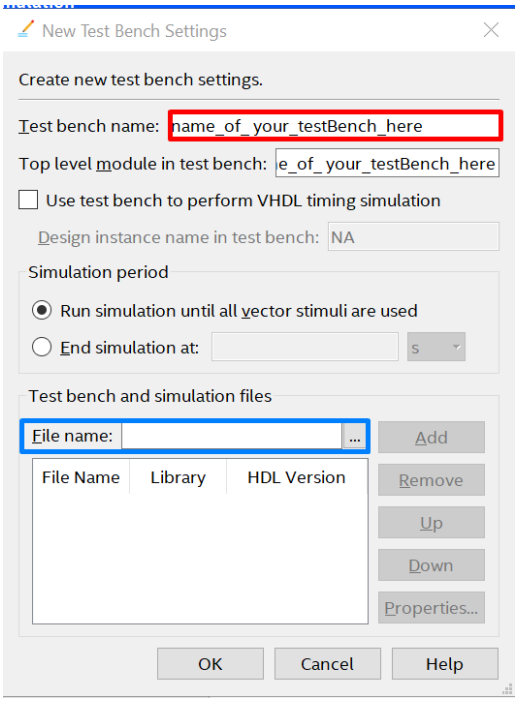
Εικόνα 40: Επιλογή ρυθμίσεων Assignments



Εικόνα 41: Επιλογή Simulation , ModelSim-Altera

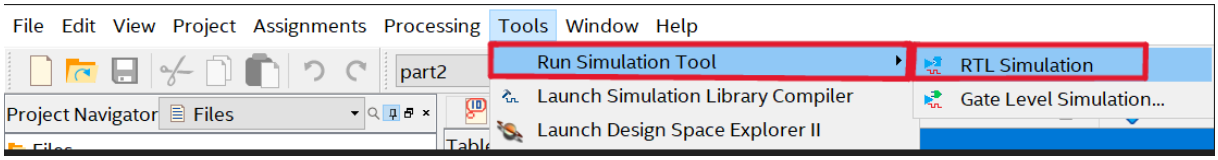


Εικόνα 42: Επιλογή "New" στο "Test Benches"



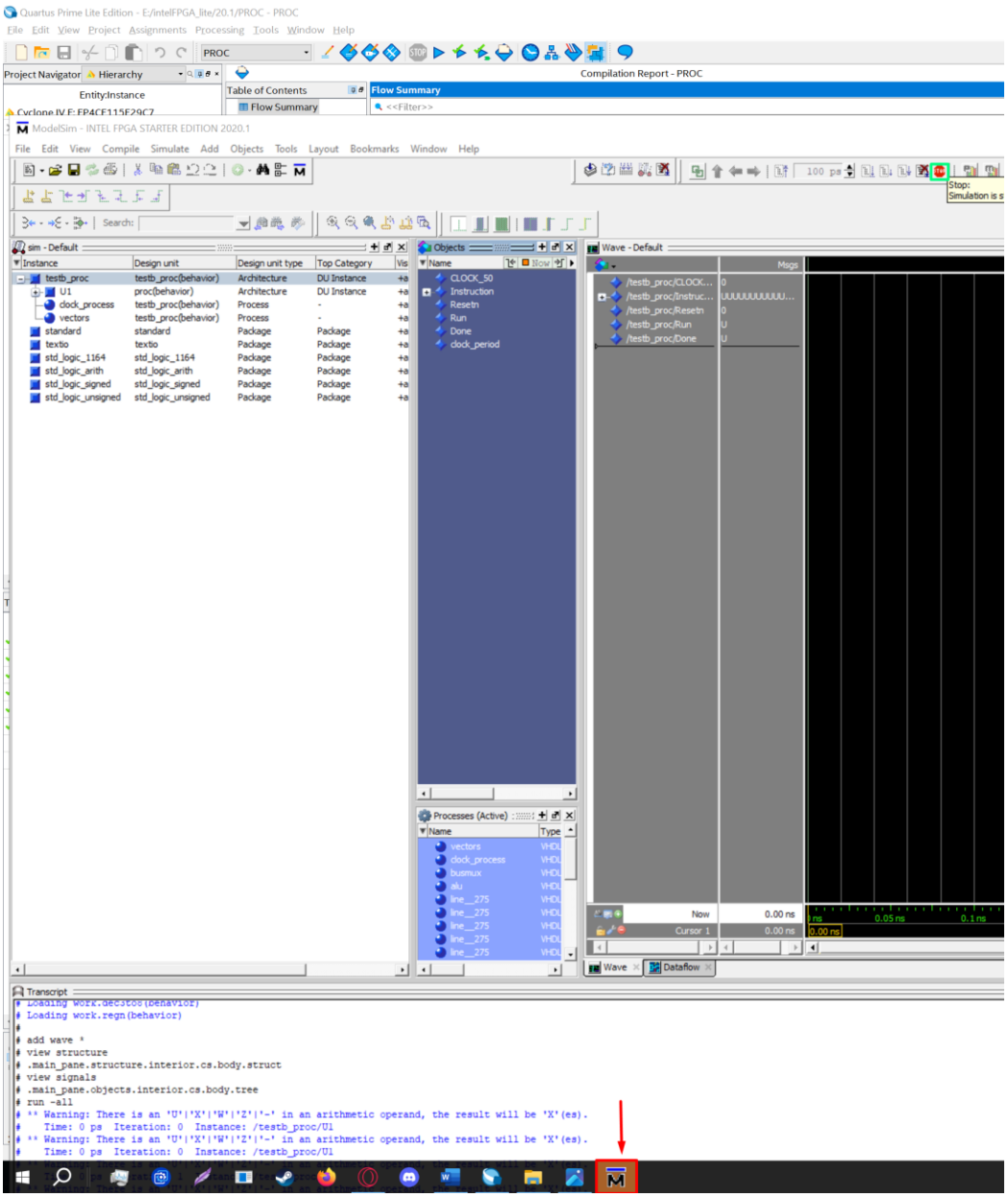
Εικόνα 43: Όνομα του test bench

Αφού ολοκληρωθεί η διαδικασία αυτή, μπορεί να γίνει ένας έλεγχος των αποτελεσμάτων του κώδικα, εάν αυτά συμπίπτουν με της εικόνας 28, πατώντας στο "Tools" > "Run Simulation Tool"> "RTL Simulation" όπως φαίνεται και στην παρακάτω εικόνα. (Βλ. Εικόνα 44).



Εικόνα 44: Διαδικασία χρήσης του Model-Sim

Πατώντας "RTL Simulation", στην μπάρα εργαλείων του Quartus, ανοίγει το "Model sim" και πρέπει να επιλεχθεί το "Stop", όπως σημειώνεται στην παρακάτω εικόνα.(Βλ. εικόνα 45)

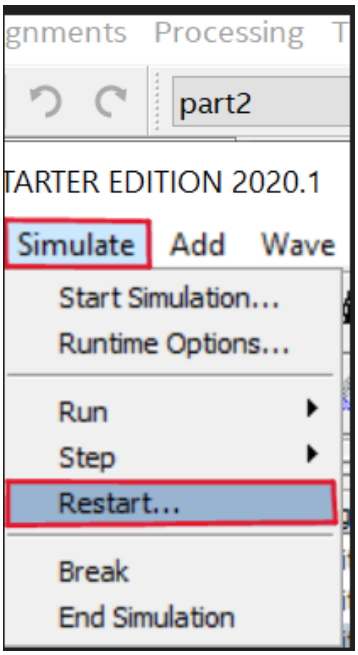


Εικόνα 45: Άνοιγμα του Model-Sim και τερματισμός της προσομοίωσης

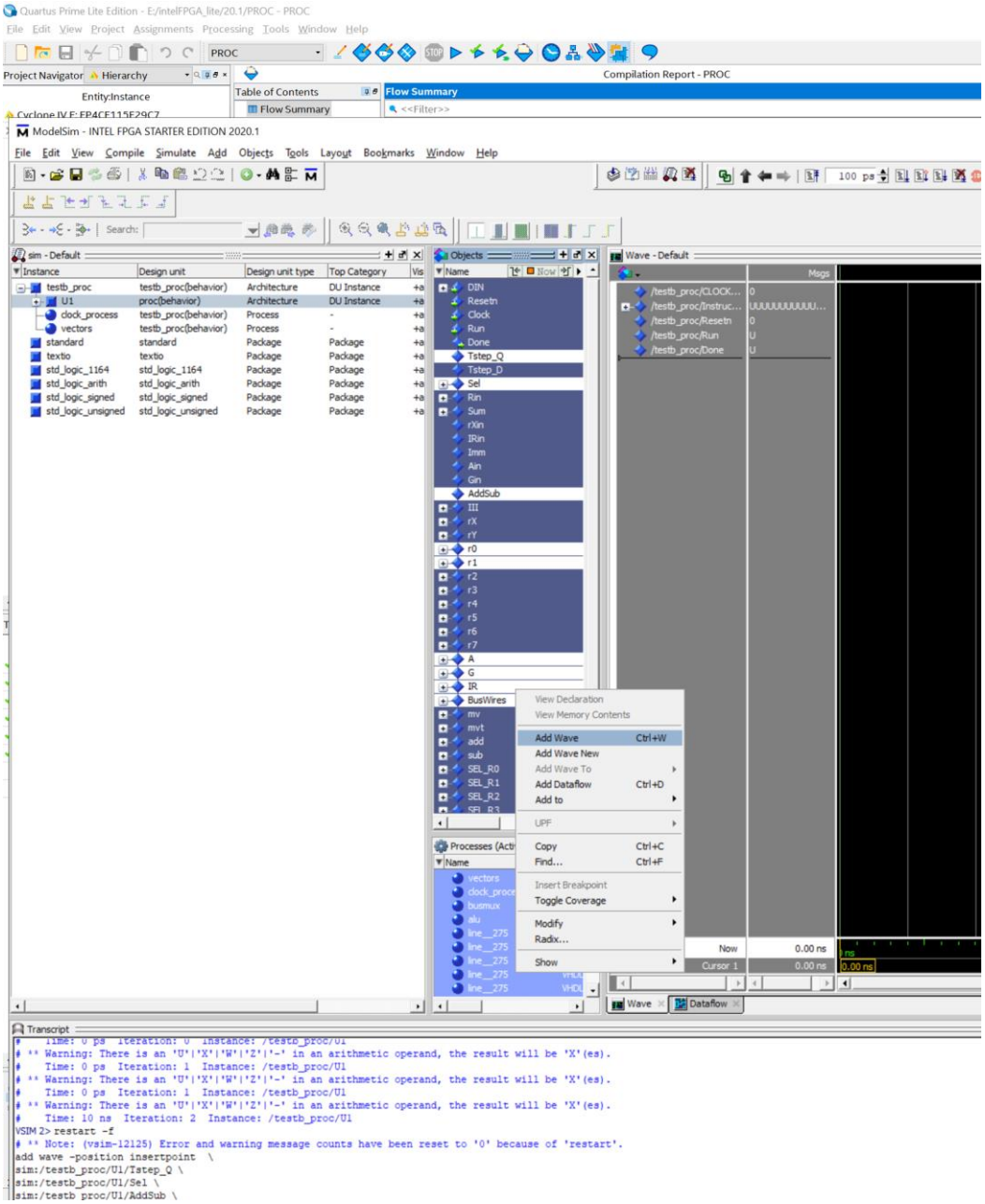
Σημείωση: Πρέπει να επιλεχθεί το “stop”, διότι στο αρχείο “testb_proc.vhd” δεν έχει οριστεί χρονικό όριο τερματισμού. Το χρονικό αυτό όριο, ορίζεται στο δεύτερο μέρος της εργασίας.

Στην συνέχεια ακολουθούνται τα εξής βήματα, όπως ακριβώς φαίνονται στις εικόνες 46 έως 49. Αφού έχει επιλεχθεί το “Stop” στη μπάρα εργαλείων, επιλέγοντας το Simulate και έπειτα το “Restart…”, για να χρησιμοποιηθούν τα σήματα του “U1” που χρειάζονται, ο χρήστης, τα επιλέγει ένα-ένα με “control” και “left click”. Ακολούθως, κάνοντας “right click”, χρειάζεται να επιλεχθεί το “Add wave”, όπως φαίνεται στην εικόνα

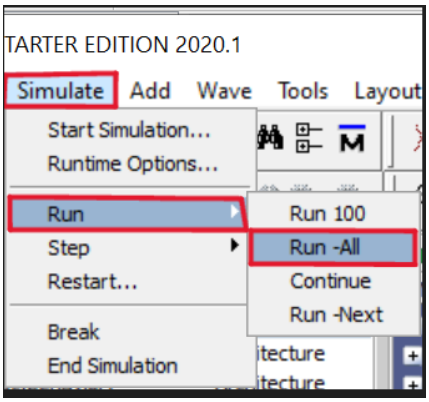
47. Η διαδικασία αυτή, εισχωρεί τις επιλογές του χρήστη στο παράθυρο των σημάτων προς εκτέλεση. Πατώντας πάλι “simulate”, αλλά, αυτή τη φορά επιλέγοντας απλά με το ποντίκι πάνω από την επιλογή “-Run”, ο χρήστης καλείται να κάνει κλικ στο “Run all” και πατώντας την τερματική επιλογή “stop” καταφέρνει την επίδειξη αποτελεσμάτων όπως αναφέρεται και στις εικόνες 48 και 49. Παρατηρείται ότι οι τιμές των αποτελεσμάτων της εικόνας 49, είναι ακριβώς ίδιες με τις τιμές της εικόνας 28, συμπεραίνοντας εξ’ αυτού ότι όλα λειτουργούν σωστά.



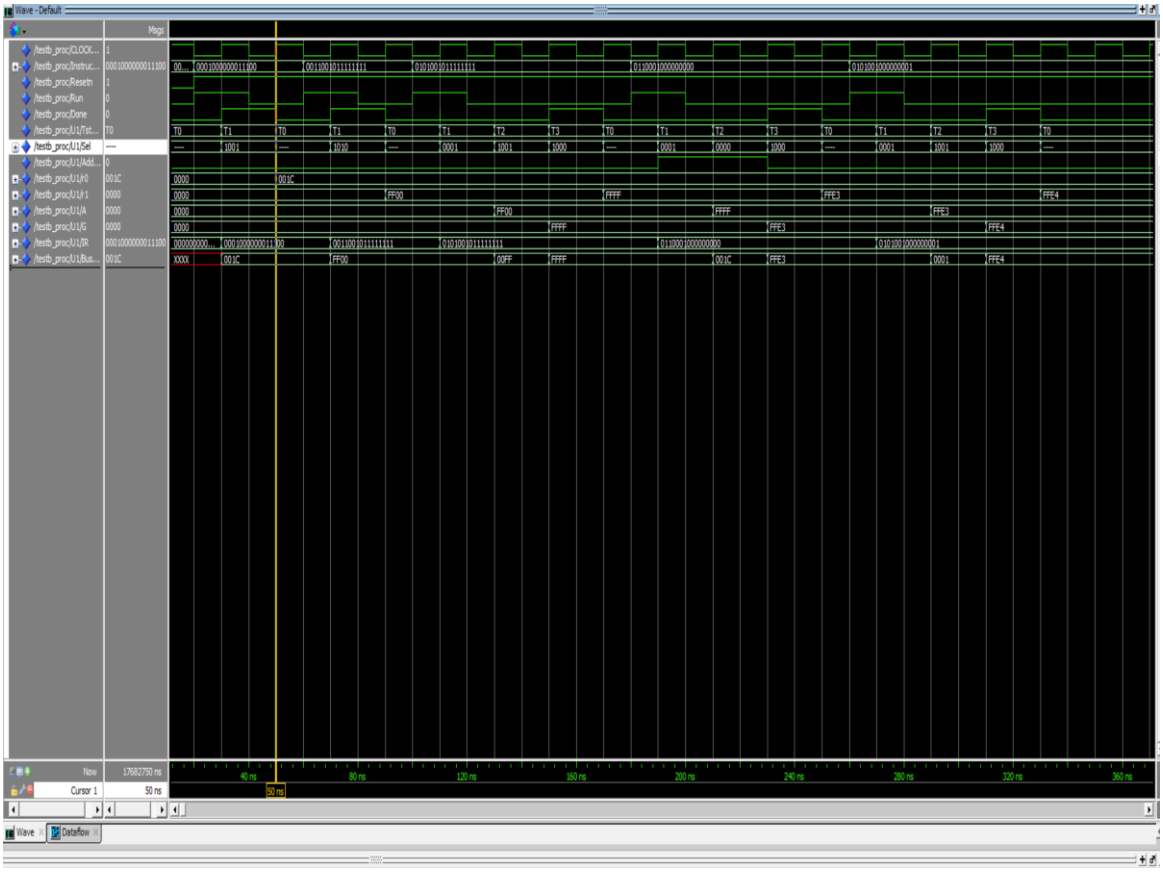
Εικόνα 46: Επιλογή Simulate > Restart



Εικόνα 47: Επιλογή σημάτων και εκτέλεση του προγράμματος



Εικόνα 48: Εκτέλεση προγράμματος με νέα σήματα



Εικόνα 49: Αποτελέσματα μέρους 1ου

3.1.1 ΣΧΟΛΙΑΣΜΟΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ 1ου ΜΕΡΟΥΣ

Στην εικόνα 49, φαίνεται η εξομοίωση του επεξεργαστή μας proc. Εδώ, παρατηρείται ένα **κύμα χρονικής προσομοίωσης,** το οποίο απεικονίζει τη συμπεριφορά των σημάτων με την πάροδο του χρόνου. Ας εξεταστεί αναλυτικά το περιεχόμενο. Αριστερά, εμφανίζεται η λίστα των **σημάτων** που συμμετέχουν στην προσομοίωση. Τα πιο σημαντικά από αυτά περιλαμβάνουν το **CLOCK**, το **Resetn**, το **Run**, το **Done** και διάφορα άλλα σήματα που σχετίζονται με καταχωρητές, επιλογές και λειτουργίες.

Το **CLOCK** είναι το βασικό σήμα ρολογιού του κυκλώματος. Η ταλάντωσή του είναι καθοριστική για τη ροή της λειτουργίας της μηχανής πεπερασμένων καταστάσεων και των επιμέρους μονάδων. Στο διάγραμμα φαίνεται η περιοδική εναλλαγή του μεταξύ υψηλού και χαμηλού επιπέδου.

Το **Resetn** είναι το σήμα επαναφοράς, το οποίο βρίσκεται αρχικά σε **χαμηλή κατάσταση** (active low) για την εκκίνηση του κυκλώματος. Μόλις το Resetn γίνει **υψηλό**, το κύκλωμα αρχίζει να λειτουργεί κανονικά. Κατά τη διάρκεια της επαναφοράς στην αρχική κατάσταση “reset”, οι τιμές άλλων σημάτων είναι είτε μηδενικές είτε ακαθόριστες ("XXXX").

Το σήμα “**Run”** παραμένει υψηλό, υποδεικνύοντας ότι η λειτουργία του κυκλώματος βρίσκεται σε **εκτέλεση**. Το “**Done”** είναι το σήμα που δηλώνει την ολοκλήρωση μιας διαδικασίας και φαίνεται να αλλάζει κατάσταση κατά τη διάρκεια των χρονικών στιγμών.

Στο κυματομορφικό διάγραμμα παρατηρείται η εξέλιξη των καταστάσεων της **F.S.M. (**Τα σήματα **T0**, **T1**, **T2**, **T3** αντιπροσωπεύουν χρονικά βήματα εκτέλεσης). Στο **T1** γίνεται αποκωδικοποίηση της εντολής, ενώ στα **T2** και **T3** εκτελούνται πράξεις, όπως η φόρτωση καταχωρητών ή υπολογισμοί μέσω της αριθμητικής λογικής μονάδας, “A.L.U”.

Τα σήματα, όπως “**A”**, “**B”**, και “**BusWires”** σχετίζονται με την κίνηση των δεδομένων. Οι τιμές των σημάτων αυτών αλλάζουν στις διαφορετικές χρονικές στιγμές, αντανακλώντας τις λειτουργίες που εκτελούνται. Για παράδειγμα:

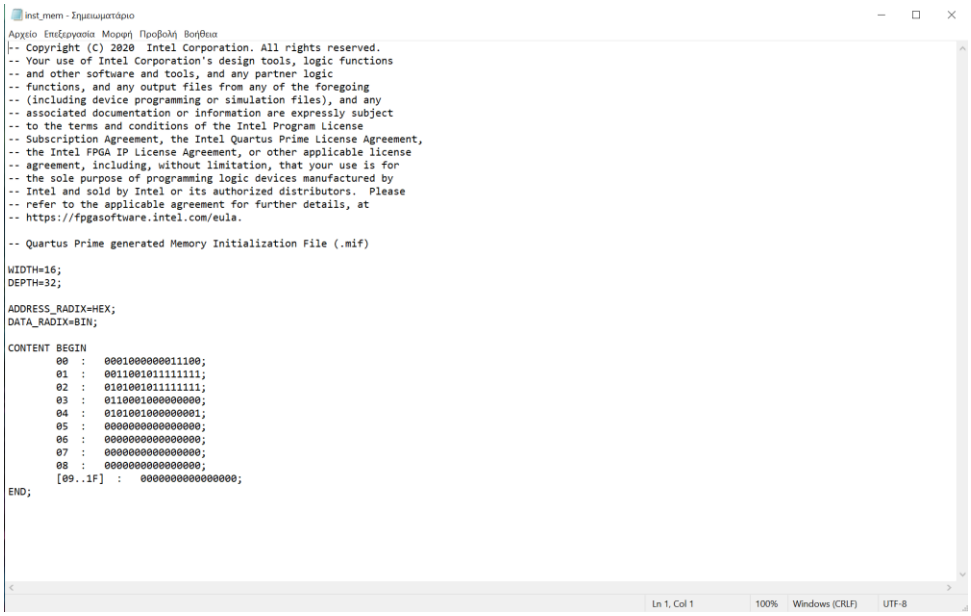
Το σήμα “**Sel”** (Selector - επιλογέας) φαίνεται να ελέγχει την επιλογή συγκεκριμένων καταχωρητών ή γραμμών στο “**Bus”**. Η τιμή του αλλάζει για να καθοδηγήσει τα δεδομένα που θα εμφανιστούν στο “**BusWires”**. Επιπλέον, οι **καταχωρητές** όπως “**R0”**, “**R1”** κ.λπ., φαίνονται να ενημερώνονται σε διαφορετικά χρονικά διαστήματα. Οι αλλαγές αυτές συνδέονται άμεσα με τη λειτουργία της F.S.M. και της A.L.U.

Συνολικά, η εικόνα 49, παρουσιάζει την εκτέλεση διαφόρων εντολών σε ένα σύστημα που χρησιμοποιεί “F.S.M.” για τον έλεγχο της ροής, με δεδομένα να διακινούνται μέσω του “**Bus”** και να επεξεργάζονται μέσω μονάδων όπως η “**A.L.U.”** και οι καταχωρητές. Το χρονικό βήμα και η εναλλαγή σημάτων αντικατοπτρίζουν τη σωστή λειτουργία του κυκλώματος, με κάθε εντολή να ολοκληρώνεται στα βήματα **T1 και T3.** Όσον αφορά τις εντολές “**mv” (move, op code 000) και “mvt” (move top, op code 001),** δεν χρειάζονται παραπάνω χρονικές καταστάσεις εφ’ όσον πληρούν απλές εκχωρήσεις σε καταχωρητές, οπότε τερματίζουν στην T1. Αυτό σημαίνει ότι το εσωτερικό σήμα “Done” γίνεται ένα. Αντιθέτως, η **“add” (add, op code 010)** ή η **“sub” (subtract, op code 011)** χρειάζονται όλες τις καταστάσεις, δηλαδή τις T0, T1, **T2 και T3,** ούτως ώστε να εκπληρωθούν και να γίνει το σήμα ελέγχου “Done” ίσο με ένα.

Σημείωση: Τα αποτελέσματα που προκύπτουν είναι κανονικά σε δυαδική, μορφή αλλά τα μετατρέπουμε σε δεκαεξαδική, όχι μόνο επειδή είναι πιο ευανάγνωστα, αλλά και επειδή έτσι παρουσιάζονται στο παράδειγμά μας. (Βλ. Εικόνα 28).

3.2 ΠΡΟΣΘΗΚΗ ΜΝΗΜΗΣ ΚΑΙ ΜΕΤΡΗΤΗ ΠΡΟΓΡΑΜΜΑΤΟΣ ΓΙΑ ΑΚΟΛΟΥΘΙΑΚΗ ΑΝΑΓΝΩΣΗ ΕΝΤΟΛΩΝ ΑΠΟ ΤΗ ΜΝΗΜΗ.

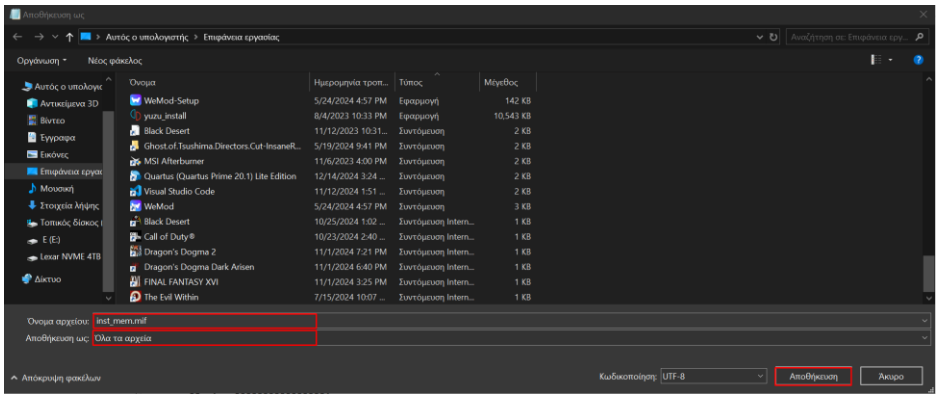
Το δεύτερο μέρος της εργασίας ξεκινά με την δημιουργία ενός "memory initialization file". Για τη σύνθεση του αρχείου αυτού, θα χρειαστεί να γραφθεί ο κώδικάς του σε ένα αρχείο "notepad" όπως φαίνεται στην παρακάτω εικόνα. (Βλ. εικόνα 50):



Εικόνα 50: Σύνθεση memory initialization file

Αφού τελειώσει η σύνθεση του αρχείου, χρειάζεται ο χρήστης να επιλέξει πάνω αριστερά "Αρχείο" και έπειτα "Αποθήκευση Ως". Προσοχή! Στην ονομασία του φακέλου πρέπει να υπάρχει η αναγνώριση ".mif" με την επιλογή "Όλα τα αρχεία" όπως φαίνεται στην παρακάτω εικόνα (Βλ. εικόνα 51).

Σημείωση: Δεν χρειάζεται να γραφθούν τα κομμάτια με τις "- -".



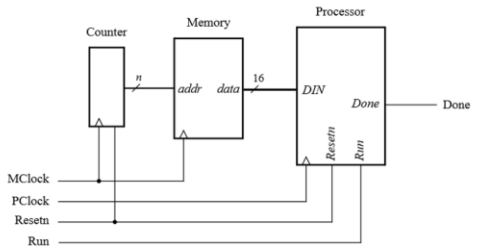
Εικόνα 51 : Τρόπος αποθήκευσης inst_mem.mif

Στη συνέχεια της εργασίας θα ξεκινήσει η υλοποίηση για το 2^{ου} μέρους. Ουσιαστικά σε αυτό το μέρος θα δημιουργηθεί ένα “memory initialization file” (.mif), που θα περιέχει τριάντα-δύο θέσεις μνήμης των δεκαέξι bit η κάθε μία. Δεν χρειάζεται να γεμίσουν και οι τριάντα-δύο θέσεις, αφού οι λειτουργίες που χρειάζεται να εκτελούνται είναι μόλις 4, όπως ακριβώς στο προηγούμενο κεφάλαιο. Το μόνο που αλλάζει είναι το top level entity, το οποίο και δίνεται ως “part2.vhd”. Το “part2.vhd” είναι ένας τρόπος για να απλοποιηθεί ο επεξεργαστής. Ακολουθούν οι εικόνες που περιγράφουν το 2^{ο} μέρος της εργασίας.(Βλ. Εικόνες 52-54).

Συγκεκριμένα, η εικόνα 52 παρουσιάζει τη σύνδεση ενός επεξεργαστή με μία μονάδα μνήμης και έναν μετρητή. Στο σύστημα αυτό, ο μετρητής χρησιμοποιείται για την ανάγνωση των περιεχομένων διαδοχικών θέσεων μνήμης, παρέχοντας τα δεδομένα στον επεξεργαστή ως μία ροή εντολών. Για την απλοποίηση της σχεδίασης και της δοκιμής του κυκλώματος, χρησιμοποιούνται ξεχωριστά σήματα ρολογιού: το "PClock" για τον επεξεργαστή και το "MClock" για τη μνήμη.

Ο μετρητής παράγει τις διευθύνσεις που χρησιμοποιούνται για την ανάγνωση των δεδομένων από τη μνήμη. Η μονάδα μνήμης δέχεται τη διεύθυνση μέσω του σήματος "addr" και επιστρέφει τα δεδομένα μέσω του σήματος "data", τα οποία είναι 16-bit. Ο επεξεργαστής λαμβάνει τα δεδομένα μέσω του σήματος "DIN" και εκτελεί τις αντίστοιχες εντολές, ενώ παράγει το σήμα "Done" για να υποδείξει την ολοκλήρωση οποιασδήποτε διαδικασίας εκτελείται.

Η σχεδίαση προορίζεται να υλοποιηθεί σε μία πλακέτα "FPGA", όπως η "DE1-SoC", "DE10-Standard" ή "DE10-Lite", και περιλαμβάνει όλες τις απαραίτητες λειτουργίες για τη σωστή επικοινωνία μεταξύ της μνήμης, του επεξεργαστή και του μετρητή.



Εικόνα 52: Περιγραφή του 2ου μέρους

Η παρακάτω εικόνα (Βλ. εικόνα 53) παρουσιάζει τον κώδικα "VHDL" για την υλοποίηση του κορυφαίου επιπέδου ("top-level module") της σχεδίασης του επεξεργαστή. Ο κώδικας ορίζει τη δομή του συστήματος, ενσωματώνοντας τα διάφορα υποσυστήματα, όπως ο επεξεργαστής ("proc"), η μνήμη εντολών ("inst_mem") και ο μετρητής ("count5").

Αρχικά, γίνεται η δήλωση της βιβλιοθήκης "IEEE" και η χρήση του πακέτου "std_logic_1164.all", το οποίο παρέχει τους βασικούς τύπους δεδομένων για σήματα. Στη συνέχεια, ορίζεται η οντότητα ("entity") "part2", που περιλαμβάνει τις εισόδους και τις εξόδους του συστήματος. Οι είσοδοι περιλαμβάνουν τα "KEY" (σήματα ελέγχου) και "SW" (διακόπτες), ενώ οι έξοδοι περιλαμβάνουν τα "LEDR" (φωτεινές ενδείξεις).

Η αρχιτεκτονική ("architecture") "Behavior" της οντότητας "part2" ορίζει τα "components" που χρησιμοποιούνται στο σύστημα. Αυτά περιλαμβάνουν :

- Τον επεξεργαστή ("proc"), ο οποίος λαμβάνει ως είσοδο δεδομένα ("DIN") και σήματα ελέγχου, ενώ παράγει το σήμα "Done" για την ολοκλήρωση μιας εντολής.
- Τη μνήμη εντολών ("inst_mem"), η οποία χρησιμοποιεί το σήμα "address" για να παρέχει δεδομένα εντολών.

- Τον μετρητή ("count5"), ο οποίος παράγει διευθύνσεις μνήμης μέσω του σήματος "Q".

Ο κώδικας περιλαμβάνει τη δήλωση εσωτερικών σημάτων, όπως τα "Resetn", "PClock", "MClock", "Run", "Done", "DIN" και "pc". Αυτά τα σήματα συνδέονται με τις αντίστοιχες εισόδους και εξόδους των υποσυστημάτων. Για παράδειγμα, το σήμα "Resetn" συνδέεται με τον πρώτο διακόπτη του "SW", ενώ το "PClock" συνδέεται με το πρώτο πλήκτρο του "KEY".

Η ενότητα "begin" της αρχιτεκτονικής περιλαμβάνει τη σύνδεση των υποσυστημάτων μέσω των "PORT MAP" δηλώσεων. Ο επεξεργαστής ("U1") λαμβάνει δεδομένα και σήματα ελέγχου από τα αντίστοιχα εσωτερικά σήματα. Η μνήμη εντολών ("U2") και ο μετρητής ("U3") συνδέονται με τα σήματα ρολογιού ("MClock") και τη διεύθυνση ("pc"), ώστε να συνεργάζονται για την παροχή εντολών στον επεξεργαστή.

Ο κώδικας συνολικά παρουσιάζει τη σύνθεση του συστήματος σε επίπεδο κορυφής, εξασφαλίζοντας τη σωστή αλληλεπίδραση μεταξύ των υποσυστημάτων. Αυτό επιτρέπει τη ροή δεδομένων και τη λειτουργία των εντολών που θα υλοποιηθούν από τον επεξεργαστή.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY part2 IS
    PORT ( KEY  : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
           SW   : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
           LEDR : OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
END part2;

ARCHITECTURE Behavior OF part2 IS
   COMPONENT proc
      PORT ( DIN                : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
             Resetn, Clock, Run : IN STD_LOGIC;
             Done               : BUFFER STD_LOGIC);
   END COMPONENT;
   COMPONENT inst_mem
      PORT ( address : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
             clock   : IN STD_LOGIC ;
             q       : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
   END COMPONENT;
   COMPONENT count5
      PORT ( Resetn, Clock : IN STD_LOGIC;
             Q             : OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
   END COMPONENT;

   SIGNAL Resetn, PClock, MClock, Run, Done : STD_LOGIC;
   SIGNAL DIN : STD_LOGIC_VECTOR(15 DOWNTO 0);
   SIGNAL pc : STD_LOGIC_VECTOR(4 DOWNTO 0);
BEGIN
   Resetn <= SW(0);
   MClock <= KEY(0);
   PClock <= KEY(1);
   Run <= SW(9);
   U1: proc PORT MAP (DIN, Resetn, PClock, Run, Done);
   LEDR(0) <= Done;
   LEDR(9) <= Run;

   U2: inst_mem PORT MAP (pc, MClock, DIN);
   U3: count5 PORT MAP (Resetn, MClock, pc);
END Behavior;
```

Figure 5: VHDL code for the top-level module.

Εικόνα 53: Κορμός κώδικα για το part2.vhd

Στη συνέχεια θα δημιουργηθεί βήμα-βήμα το memory initialization file. Η παρακάτω εικόνα δείχνει την σύνθεση της “Read Only Memory” (Βλ. εικόνα 54).

Η εικόνα παρουσιάζει την υλοποίηση μιας μνήμης "32 x 16 ROM" με τη χρήση ενός καταχωρητή διευθύνσεων ("address register"). Η μνήμη αυτή διαθέτει 32 θέσεις αποθήκευσης, καθεμία από τις οποίες περιέχει 16-bit δεδομένα. Ο καταχωρητής διευθύνσεων λαμβάνει το σήμα διεύθυνσης ("Address") και το σήμα ρολογιού ("Clock") και παράγει τις κατάλληλες διευθύνσεις για την ανάγνωση δεδομένων από τη μνήμη. Τα δεδομένα που διαβάζονται εξέρχονται μέσω της εξόδου "DataOut".

Η μνήμη είναι σχεδιασμένη να χρησιμοποιηθεί ως μέρος του συστήματος, όπως περιγράφεται στο υπόλοιπο της εργασίας. Το σήμα "Address" είναι 5-bit, επιτρέποντας την πρόσβαση σε 32 διακριτές θέσεις μνήμης, ενώ η έξοδος "DataOut" είναι 16-bit, αντιπροσωπεύοντας τα δεδομένα που αποθηκεύονται σε κάθε θέση.

Στο πλαίσιο της υλοποίησης, παρέχεται ένα αρχείο έργου "Quartus Prime", που ονομάζεται "part2.qpf". Ο χρήστης καλείται να χρησιμοποιήσει το εργαλείο "Quartus IP Catalog" για τη δημιουργία του υποσυστήματος μνήμης. Στο "IP Catalog" επιλέγεται το "ROM: 1-PORT" μέσα από την κατηγορία "Basic Functions > On Chip Memory". Η μνήμη καθορίζεται με όνομα αρχείου εξόδου "inst_mem.vhd" σε μορφή "VHDL".

Για την αρχικοποίηση της μνήμης με εντολές επεξεργαστή, απαιτείται ένα αρχείο αρχικοποίησης μνήμης ("memory initialization file - MIF"). Στο παράδειγμα, καθορίζεται ένα αρχείο με το όνομα "inst_mem.mif". Το αρχείο αυτό πρέπει να δημιουργηθεί στον φάκελο του έργου "Quartus". Η διαδικασία περιλαμβάνει τη ρύθμιση της εξόδου "q" ώστε να μην είναι εγγεγραμμένη (μη επιλεγμένο). Τέλος, όταν η διαδικασία ρυθμίσεων ολοκληρωθεί, μόνο το αρχείο "inst_mem.vhd" θα πρέπει να επιλεγεί στη σύνοψη και να ολοκληρωθεί η διαδικασία πατώντας "Finish".

Αυτό το σχέδιο αποτελεί τμήμα της διαδικασίας σχεδιασμού της μνήμης, επιτρέποντας την αποθήκευση και ανάγνωση εντολών επεξεργαστή μέσω της ROM.

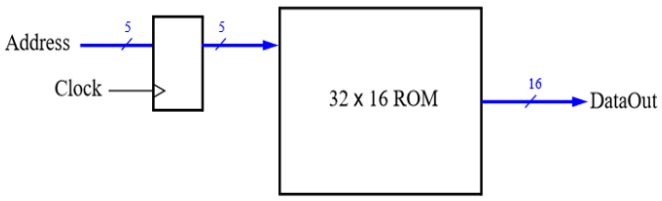
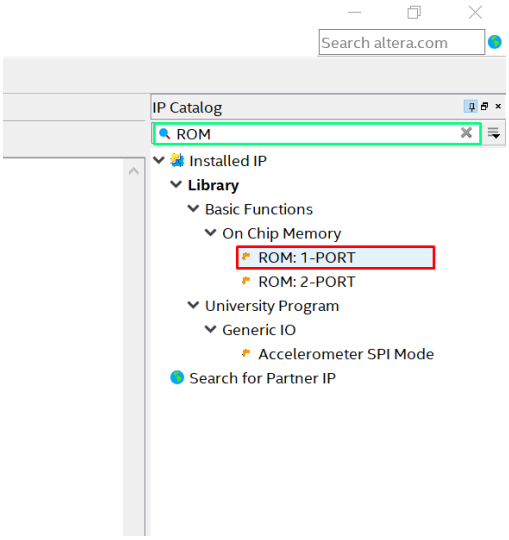


Figure 6: The 32 x 16 ROM with address register.

Εικόνα 54: Σχηματικό Read Only Memory 32x16 και περιγραφή δεύτερου βήματος

Η παρακάτω εικόνα (Βλ. εικόνα 55), δείχνει την επιλογή “ROM” στο “IP catalog” και οι επόμενες εικόνες (Βλ. εικόνες 56 – 58), παρουσιάζουν τις ρυθμίσεις που πρέπει να χρησιμοποιηθούν για την εργασία.



Εικόνα 55: “ROM” στο “IP Catalog" και επιλογή του “ROM:1-PORT”

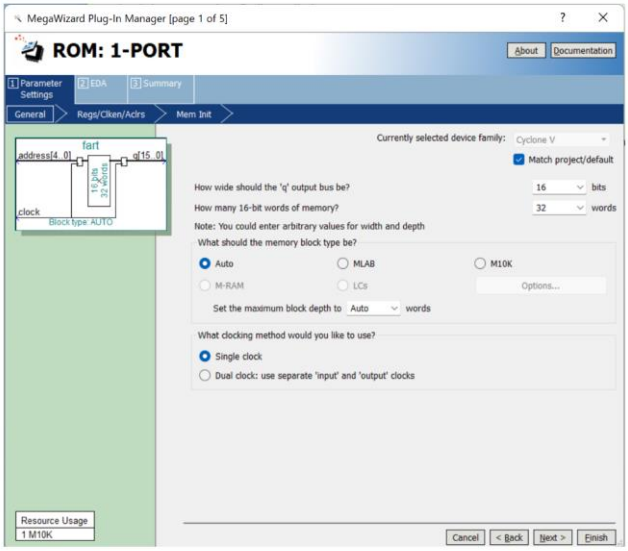


Figure 7: Specifying memory size.

3. As described above, you need to provide a memory initialization file (MIF) called *inst_mem.mif* to specify the contents of the ROM. An example of a memory initialization file is given in Figure 10. Note that comments (% . . . %) are included in this file as a way of documenting the meaning of the provided instructions. Set the contents of your `MIF` file such that it provides enough processor instructions to test your circuit.

4. The VHDL code in Figure 5 includes the appropriate port names for implementation of the design on an FPGA board, like the DE1-SoC. The switch SW_9 drives the processor's *Run* input, SW_0 is connected to *Resetn*, KEY_0 to *MClock*, and KEY_1 to *PClock*. The Run signal is displayed on $LEDR_9$ and *Done* is connected to $LEDR_0$.

Εικόνα 56: Βήμα 1ο για τη δημιουργία του "inst_mem.vhd"

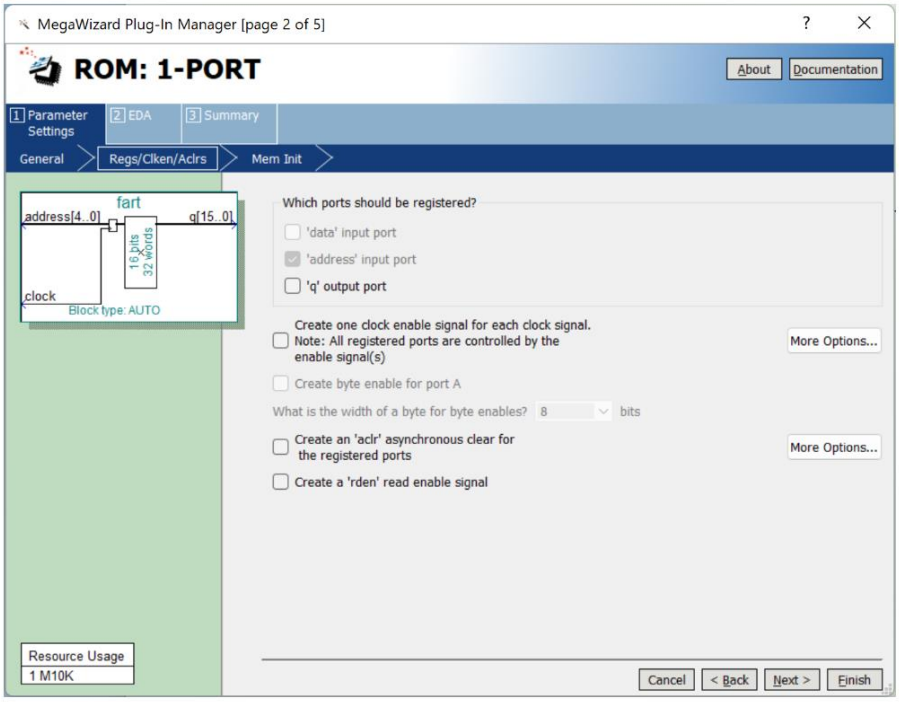
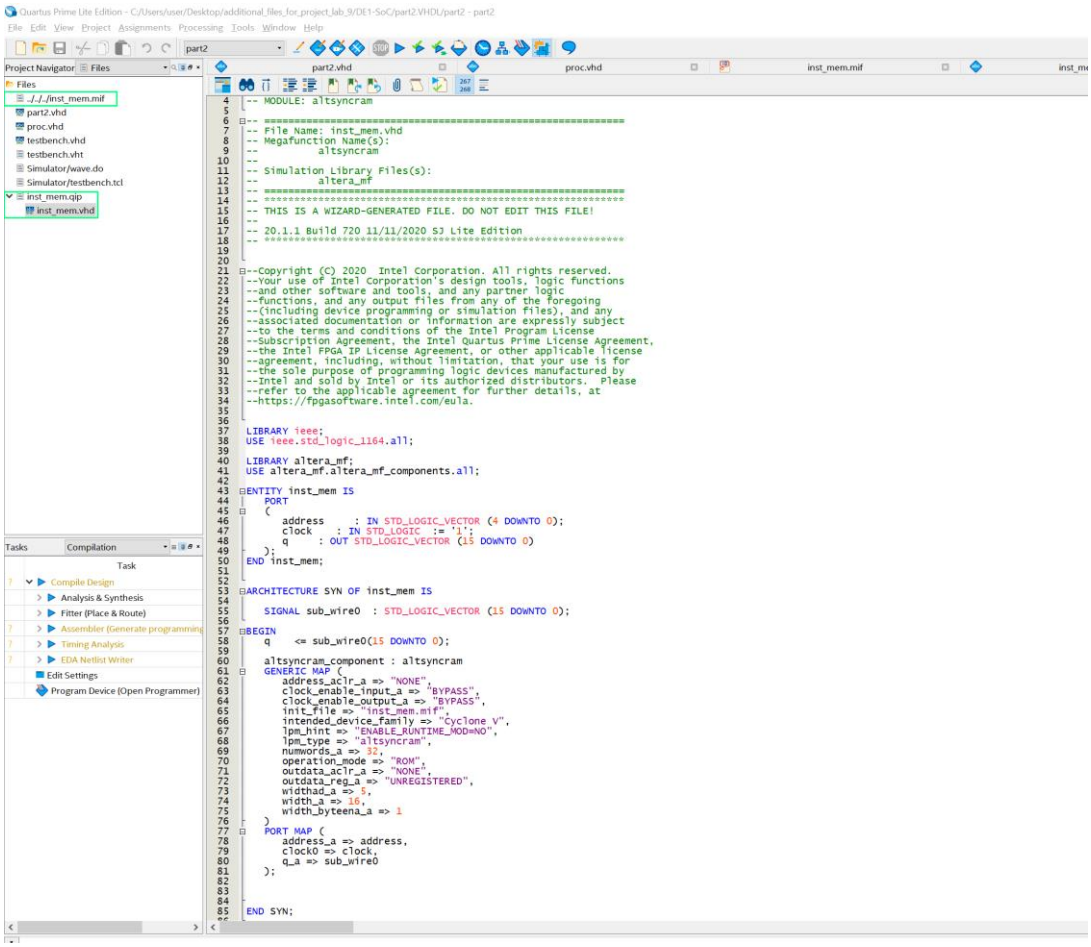


Figure 8: Specifying which memory ports are registered.

Εικόνα 57: Βήμα 2ο για τη δημιουργία του "inst_mem.vhd"

ΠΡΟΣΟΧΗ: Η εικόνα που ακολουθεί, βασίζεται στη δημιουργία του "memory initialization file". Αυτό το βήμα πρέπει να έχει γίνει εκ των προτέρων, ώστε η παραπάνω διαδικασία να δημιουργήσει αυτόματα το "inst_mem.vhd", το οποίο ουσιαστικά λαμβάνει την εντολή από την μνήμη, ανατρέχοντάς την με την χρήση του "PC" (program counter). Στη συνέχεια, πατώντας "next" και τέλος, "finish", δημιουργείται το αρχείο που προαναφέρθηκε. (Βλ. εικόνα 58)



Εικόνα 58: Memory Initialization File και αρχείο inst_mem.vhd

Ο παραπάνω κώδικας περιγράφει τη σχεδίαση μιας μονάδας μνήμης εντολών (“inst_mem”), η οποία είναι μια μνήμη μόνο ανάγνωσης (Read Only Memory), σχεδιασμένη να αποθηκεύει τριάντα-δύο λέξεις των δεκαέξι bit. Για τη λειτουργία της, η μνήμη χρησιμοποιεί, από τη βιβλιοθήκη της “Altera”, ένα έτοιμο υποσύστημα μνήμης (“altsyncram”). Η οντότητα “inst_mem” περιλαμβάνει μία είσοδο 5-bit για τη διεύθυνση (“address”), ένα σήμα ρολογιού (“clock”), το οποίο συγχρονίζει τη μνήμη και μία έξοδο δεκαέξι-bit (q) η οποία επιστρέφει, από την επιλεγμένη διεύθυνση, τη λέξη δεδομένων.

Στην συγχρονισμένη αρχιτεκτονική, για τη μεταφορά της εξόδου της υπομονάδας “altsyncram” προς το σήμα εξόδου “q”, η μνήμη χρησιμοποιεί το εσωτερικό σήμα “sub_wire0”. Η υπομονάδα “altsyncram” ρυθμίζεται με συγκεκριμένες παραμέτρους, όπως το μέγεθος της μνήμης (32 λέξεις των 16 bit), το πλάτος της διεύθυνσης (5 bit) και το όνομα του αρχείου αρχικοποίησης (inst_mem.mif). Η μνήμη λειτουργεί σαν “Read Only Memory” (operation_mode => "ROM") και οι έξοδοί της δεν είναι εγγεγραμμένες

(outdata_reg_a => "UNREGISTERED"). Ο συγχρονισμός της γίνεται μέσω του σήματος ρολογιού (“clock”), ενώ η διεύθυνση που εισάγεται καθορίζει ποια λέξη δεδομένων θα διαβαστεί από τη μνήμη.

Η παρακάτω εικόνα δείχνει το πώς συντάσσεται ο κώδικας του memory initialization file, καθώς επίσης και τα αποτελέσματα που πρέπει να εμφανιστούν στο model sim. [intel, exercise vhdl_lab9]

```
DEPTH = 32;
WIDTH = 16;
ADDRESS_RADIX = HEX;
DATA_RADIX = BIN;
CONTENT
BEGIN
00 : 0001000000011100;    % mv  r0, #28      %
01 : 0011001011111111;    % mvt r1, #0xFF    %
02 : 0101001011111111;    % add r1, #0xFF    %
03 : 0110001000000000;    % sub r1, r0       %
04 : 0101001000000001;    % add r1, #1       %
05 : 0000000000000000;
. . . (some lines not shown)
1F : 0000000000000000;
END;
```

Figure 10: An example memory initialization file (MIF).

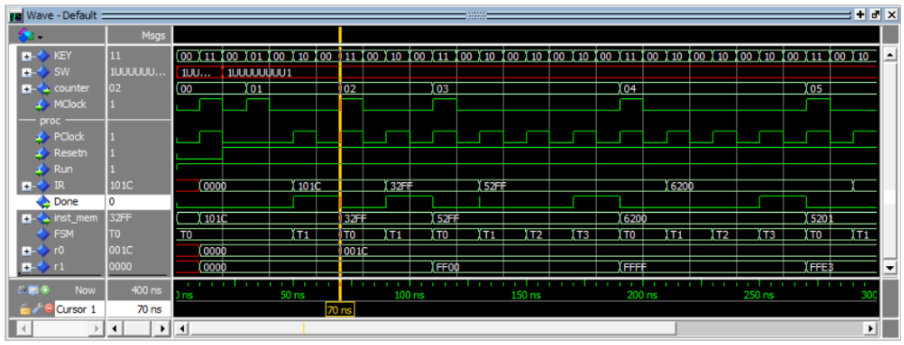


Figure 11: An example simulation output using the MIF in Figure 10.

Εικόνα 59: Κώδικας memory initialization file και αποτελέσματα model-sim

```vhdl
-- Reset με το SW(0). Ο μετρητής και η μνήμη συγχρονίζονται με το KEY(0).
-- Κάθε εντολή εκτελείται από τον επεξεργαστή με το KEY(1).
-- Το SW(9) είναι η είσοδος Run.
-- Χρησιμοποιήστε το KEY(0) για προώθηση της μνήμης πριν από κάθε κύκλο ρολογιού του επεξεργαστή.

LIBRARY ieee; -- Εισαγωγή της βιβλιοθήκης IEEE
USE ieee.std_logic_1164.all; -- Χρήση των τυποποιημένων λογικών τύπων της βιβλιοθήκης IEEE
ENTITY part2 IS -- Ορισμός της οντότητας part2
    PORT (
        KEY   : IN   STD_LOGIC_VECTOR(1 DOWNTO 0); -- Δύο πλήκτρα ως είσοδοι
        SW    : IN   STD_LOGIC_VECTOR(9 DOWNTO 0); -- 10 διακόπτες ως είσοδοι
        LEDR  : OUT  STD_LOGIC_VECTOR(9 DOWNTO 0) -- 10 LEDs ως έξοδοι
    );
END part2;
ARCHITECTURE Behavior OF part2 IS -- Αρχιτεκτονική του part2
   COMPONENT proc -- Επεξεργαστής
      PORT (
         DIN                : IN      STD_LOGIC_VECTOR(15 DOWNTO 0); -- Είσοδος δεδομένων
         Resetn, Clock, Run  : IN      STD_LOGIC; -- Σήματα επαναφοράς, ρολογιού και εκτέλεσης
         Done               : BUFFER  STD_LOGIC -- Σήμα ολοκλήρωσης
      );
   END COMPONENT;
   COMPONENT inst_mem -- Μνήμη εντολών
      PORT (
         address   : IN STD_LOGIC_VECTOR (4 DOWNTO 0); -- Διεύθυνση εισόδου (5-bit)
         clock     : IN STD_LOGIC ; -- Σήμα ρολογιού
         q         : OUT STD_LOGIC_VECTOR (15 DOWNTO 0) -- Δεδομένα εξόδου (16-bit)
      );
   END COMPONENT;
   COMPONENT count5 -- Μετρητής 5-bit
      PORT (
         Resetn, Clock   : IN   STD_LOGIC; -- Σήματα επαναφοράς και ρολογιού
         Q               : OUT  STD_LOGIC_VECTOR(4 DOWNTO 0) -- Έξοδος τιμής μετρητή (5-bit)
      );
   END COMPONENT;
   -- Εσωτερικά σήματα
   SIGNAL Resetn, PClock, MClock, Run, Done : STD_LOGIC; -- Σήματα ελέγχου και κατάστασης
   SIGNAL DIN : STD_LOGIC_VECTOR(15 DOWNTO 0); -- Είσοδος δεδομένων από τη μνήμη
   SIGNAL pc : STD_LOGIC_VECTOR(4 DOWNTO 0); -- Διεύθυνση που παράγεται από τον μετρητή
BEGIN
   -- Αντιστοίχιση των εισόδων
   Resetn <= SW(0); -- Το SW(0) ελέγχει την επαναφορά
   MClock <= KEY(0); -- Το KEY(0) είναι το ρολόι για τη μνήμη και τον μετρητή
   PClock <= KEY(1); -- Το KEY(1) είναι το ρολόι για τον επεξεργαστή
   Run <= SW(9); -- Το SW(9) ενεργοποιεί τη λειτουργία Run
   -- Αντιστοίχιση του επεξεργαστή
   U1: proc PORT MAP (DIN, Resetn, PClock, Run, Done); -- Σύνδεση εισόδων και εξόδων του επεξεργαστή
   LEDR(0) <= Done; -- Το πρώτο LED δείχνει την ολοκλήρωση της εκτέλεσης
   LEDR(9) <= Run; -- Το τελευταίο LED δείχνει την ενεργοποίηση της λειτουργίας Run
   LEDR(8 DOWNTO 1) <= "00000000"; -- Τα υπόλοιπα LEDs είναι απενεργοποιημένα
   -- Αντιστοίχιση της μνήμης
   U2: inst_mem PORT MAP (pc, MClock, DIN); -- Σύνδεση εισόδων και εξόδων της μνήμης
   -- Αντιστοίχιση του μετρητή
   U3: count5 PORT MAP (Resetn, MClock, pc); -- Σύνδεση εισόδων και εξόδων του μετρητή
END Behavior;
-- Μετρητής 5-bit
LIBRARY ieee; -- Εισαγωγή της βιβλιοθήκης IEEE
USE ieee.std_logic_1164.all; -- Χρήση των τυποποιημένων λογικών τύπων της βιβλιοθήκης IEEE
USE ieee.std_logic_unsigned.all; -- Χρήση της αριθμητικής υποστήριξης για STD_LOGIC_VECTOR
ENTITY count5 IS -- Οντότητα count5
    PORT (
        Resetn, Clock   : IN   STD_LOGIC; -- Σήματα επαναφοράς και ρολογιού
        Q               : OUT  STD_LOGIC_VECTOR(4 DOWNTO 0) -- Έξοδος τιμής μετρητή
    );
END count5;
ARCHITECTURE Behavior OF count5 IS -- Αρχιτεκτονική του μετρητή
   SIGNAL Count : STD_LOGIC_VECTOR(4 DOWNTO 0); -- Εσωτερικό σήμα για την τιμή του μετρητή
BEGIN
   -- Διαδικασία λειτουργίας του μετρητή
   PROCESS (Clock, Resetn)
   BEGIN
         IF (Resetn = '0') THEN -- Αν το Resetn είναι 0
            Count <= "00000"; -- Μηδενισμός του μετρητή
         ELSIF (rising_edge(Clock)) THEN -- Στο ανερχόμενο μέτωπο του ρολογιού
            Count <= Count + '1'; -- Αύξηση του μετρητή κατά 1
         END IF;
   END PROCESS;
   Q <= Count; -- Αντιστοίχιση της εξόδου Q στην τιμή του μετρητή
END Behavior;

```

Εικόνα 60: Σχεδίαση κυκλώματος για ακολουθιακή ανάγνωση εντολών

Ο παραπάνω κώδικας περιγράφει τη σχεδίαση ενός κυκλώματος που συνδυάζει έναν επεξεργαστή, μία μνήμη εντολών και έναν μετρητή 5-bit. Στόχος του κυκλώματος είναι η διαδοχική ανάγνωση εντολών από τη μνήμη μέσω του μετρητή και η εκτέλεσή τους από τον επεξεργαστή. Η κύρια οντότητα, που ονομάζεται “part2”, περιλαμβάνει ως εισόδους τα πλήκτρα “KEY” και τους διακόπτες “SW”, ενώ εξάγει σήματα κατάστασης μέσω των “LEDs” και “LEDR”. Το σήμα “Resetn”, που συνδέεται με τον διακόπτη “SW(0)”, χρησιμοποιείται για την επαναφορά όλων των υποσυστημάτων του κυκλώματος. Το σήμα “Mclock”, που προκύπτει από το πλήκτρο “KEY(0)”, ελέγχει το ρολόι της μνήμης, ενώ το σήμα “Pclock”, που προκύπτει από το πλήκτρο “KEY(1)”, συγχρονίζει τον επεξεργαστή. Ο διακόπτης “SW(9)”, μέσω του σήματος “Run”, ενεργοποιεί τη λειτουργία εκτέλεσης εντολών.

Το κύκλωμα αποτελείται από τρία υποσυστήματα: τον επεξεργαστή (proc), τη μνήμη εντολών (inst_mem) και τον μετρητή πέντε-bit (count5). Ο επεξεργαστής λαμβάνει δεδομένα από τη μνήμη μέσω του σήματος “DIN”, ενώ ελέγχεται από τα σήματα “Resetn”, “Pclock” και “Run”. Ολοκληρώνοντας την εκτέλεση μιας εντολής, ενεργοποιεί το σήμα “Done”, το οποίο εμφανίζεται στο πρώτο “LED” της εξόδου “LEDR”. Η μνήμη εντολών δέχεται ως είσοδο τη διεύθυνση που παρέχεται από τον μετρητή και επιστρέφει την αντίστοιχη εντολή στο σήμα “DIN”. Ο μετρητής, που λειτουργεί σε συγχρονισμό με το ρολόι “Mclock”, παράγει διαδοχικές διευθύνσεις πέντε-bit για τη μνήμη. Σε κάθε παλμό του ρολογιού, ο μετρητής αυξάνει τη διεύθυνση κατά ένα, επιτρέποντας τη διαδοχική ανάγνωση εντολών.

Το κύκλωμα λειτουργεί ως εξής: κατά την επαναφορά μέσω του διακόπτη SW(0), ο μετρητής μηδενίζεται και τα υποσυστήματα επανέρχονται στην αρχική τους κατάσταση. Με τη χρήση του πλήκτρου KEY(0), ο μετρητής και η μνήμη συγχρονίζονται, ενώ το πλήκτρο KEY(1) συγχρονίζει την εκτέλεση εντολών από τον επεξεργαστή. Ο μετρητής παράγει διαδοχικές διευθύνσεις, η μνήμη επιστρέφει την αντίστοιχη εντολή και ο επεξεργαστής την εκτελεί με την ανύψωση του σήματος “Done” να υποδεικνύει την ολοκλήρωση κάθε εντολής. Έτσι, το κύκλωμα παρέχει έναν απλό τρόπο ανάγνωσης και εκτέλεσης εντολών από τη μνήμη, με δυνατότητα ελέγχου μέσω των διακοπτών και πλήκτρων.

Για παρατηρηθούν τα αποτελέσματα στο model sim πρέπει να δημιουργηθεί ένας φάκελος “test bench”, το οποίο θα περιέχει όλα τα σήματα των components όπως προαναφέρθηκε και στο κεφάλαιο 3.1. Στην ακόλουθη εικόνα φαίνεται το αρχείο “testbench.vhd”.

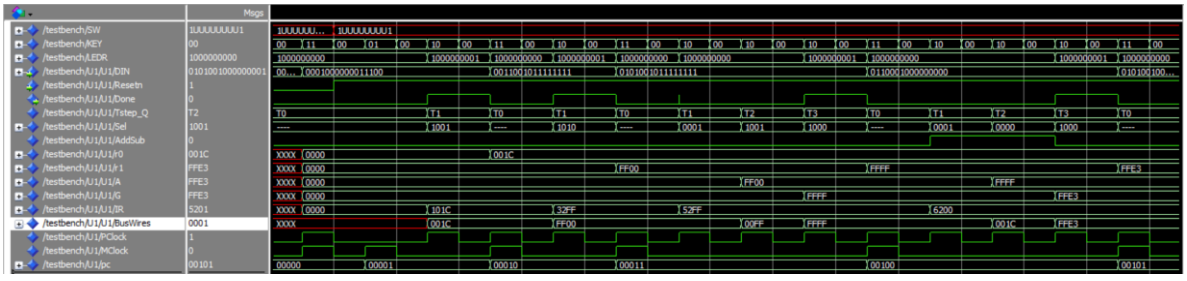
```
1  LIBRARY ieee; -- Εισαγωγή της βιβλιοθήκης IEEE
2  USE ieee.std_logic_1164.all; -- Χρήση των τυποποιημένων λογικών τύπων της βιβλιοθήκης IEEE
3  USE ieee.std_logic_signed.all; -- Χρήση υποστήριξης αριθμητικών πράξεων για signed λογικούς διανύσματα
4
5  ENTITY testbench IS -- Οντότητα του testbench
6  END testbench;
7
8  ARCHITECTURE Behavior OF testbench IS -- Αρχιτεκτονική του testbench
9      COMPONENT part2 -- Δήλωση του component "part2", που είναι το κύκλωμα προς δοκιμή
10         PORT (  KEY   : IN   STD_LOGIC_VECTOR(1 DOWNTO 0); -- Είσοδος πλήκτρων
11                 SW    : IN   STD_LOGIC_VECTOR(9 DOWNTO 0); -- Είσοδος διακοπτών
12                 LEDR  : OUT  STD_LOGIC_VECTOR(9 DOWNTO 0)); -- Έξοδος LEDs
13     END COMPONENT;
14
15     SIGNAL SW : STD_LOGIC_VECTOR(9 DOWNTO 0); -- Σήμα για τη σύνδεση των διακοπτών
16     SIGNAL KEY : STD_LOGIC_VECTOR(1 DOWNTO 0); -- Σήμα για τη σύνδεση των πλήκτρων
17     SIGNAL LEDR : STD_LOGIC_VECTOR(9 DOWNTO 0); -- Σήμα για τη σύνδεση των LEDs
18 BEGIN
19     U1: part2 PORT MAP (KEY, SW, LEDR); -- Σύνδεση των σημάτων στο component part2
20
21     SW_process: PROCESS -- Διαδικασία για την προσομοίωση των διακοπτών
22     BEGIN
23         SW(9) <= '1'; SW(0) <= '0'; -- Ενεργοποίηση του σήματος Run (SW(9)) και Resetn σε μηδενική κατάσταση (SW(0))
24         WAIT FOR 20 ns;             -- Αναμονή 20 ns για σταθεροποίηση
25         SW(0) <= '1'; -- Επαναφορά του Resetn σε ενεργή κατάσταση
26         WAIT; -- Αναμονή επ' αόριστον (τέλος προσομοίωσης για αυτή τη διαδικασία)
27     END PROCESS;
28
29     vectors: PROCESS -- Διαδικασία για την προσομοίωση του ρολογιού (KEY)
30     BEGIN
31         KEY(0) <= '0'; KEY(1) <= '0';   -- Αρχικοποίηση των ρολογιών MClock (KEY(0)) και PClock (KEY(1)) σε μηδενική κατάσταση
32         WAIT FOR 10 ns; KEY(0) <= '1'; KEY(1) <= '1'; -- Ενεργοποίηση των ρολογιών MClock και PClock
33         WAIT FOR 10 ns; KEY(0) <= '0'; KEY(1) <= '0'; -- Απενεργοποίηση των MClock και PClock
34         WAIT FOR 10 ns; KEY(0) <= '1'; -- Ενεργοποίηση μόνο του MClock
35         WAIT FOR 10 ns; KEY(0) <= '0'; -- Απενεργοποίηση του MClock
36         WAIT FOR 10 ns; KEY(1) <= '1'; -- Ενεργοποίηση μόνο του PClock
37         WAIT FOR 10 ns; KEY(1) <= '0'; -- Απενεργοποίηση του PClock
38         WAIT FOR 10 ns; KEY(0) <= '1'; KEY(1) <= '1'; -- Ενεργοποίηση και των δύο ρολογιών
39         WAIT FOR 10 ns; KEY(0) <= '0'; KEY(1) <= '0'; -- Απενεργοποίηση και των δύο ρολογιών
40         WAIT FOR 10 ns; KEY(1) <= '1'; -- Ενεργοποίηση μόνο του PClock
41         WAIT FOR 10 ns; KEY(1) <= '0'; -- Απενεργοποίηση του PClock
42         WAIT FOR 10 ns; KEY(0) <= '1'; KEY(1) <= '1'; -- Ενεργοποίηση και των δύο ρολογιών
43         WAIT FOR 10 ns; KEY(0) <= '0'; KEY(1) <= '0'; -- Απενεργοποίηση και των δύο ρολογιών
44         WAIT FOR 10 ns; KEY(1) <= '1'; -- Ενεργοποίηση μόνο του PClock
45         WAIT FOR 10 ns; KEY(1) <= '0'; -- Απενεργοποίηση του PClock
46         WAIT FOR 10 ns; KEY(1) <= '1'; -- Ενεργοποίηση μόνο του PClock
47         WAIT FOR 10 ns; KEY(1) <= '0'; -- Απενεργοποίηση του PClock
48         WAIT FOR 10 ns; KEY(1) <= '1'; -- Ενεργοποίηση μόνο του PClock
49         WAIT FOR 10 ns; KEY(1) <= '0'; -- Απενεργοποίηση του PClock
50         WAIT FOR 10 ns; KEY(0) <= '1'; KEY(1) <= '1'; -- Ενεργοποίηση και των δύο ρολογιών
51         WAIT FOR 10 ns; KEY(0) <= '0'; KEY(1) <= '0'; -- Απενεργοποίηση και των δύο ρολογιών
52         WAIT FOR 10 ns; KEY(1) <= '1'; -- Ενεργοποίηση μόνο του PClock
53         WAIT FOR 10 ns; KEY(1) <= '0'; -- Απενεργοποίηση του PClock
54         WAIT FOR 10 ns; KEY(1) <= '1'; -- Ενεργοποίηση μόνο του PClock
55         WAIT FOR 10 ns; KEY(1) <= '0'; -- Απενεργοποίηση του PClock
56         WAIT FOR 10 ns; KEY(1) <= '1'; -- Ενεργοποίηση μόνο του PClock
57         WAIT FOR 10 ns; KEY(1) <= '0'; -- Απενεργοποίηση του PClock
58         WAIT FOR 10 ns; KEY(0) <= '1'; KEY(1) <= '1'; -- Ενεργοποίηση και των δύο ρολογιών
59         WAIT FOR 10 ns; KEY(0) <= '0'; KEY(1) <= '0'; -- Απενεργοποίηση και των δύο ρολογιών
60         WAIT FOR 10 ns; KEY(1) <= '1'; -- Ενεργοποίηση μόνο του PClock
61         WAIT FOR 10 ns; KEY(1) <= '0'; -- Απενεργοποίηση του PClock
62         WAIT FOR 10 ns; KEY(1) <= '1'; -- Ενεργοποίηση μόνο του PClock
63         WAIT FOR 10 ns; -- Αναμονή 10 ns
64         KEY(1) <= '0'; -- Απενεργοποίηση του PClock
65         WAIT; -- Αναμονή επ' αόριστον
66     END PROCESS;
67 END;
```

Εικόνα 61: Αρχείο testbench.vhd

Ο παραπάνω κώδικας υλοποιεί ένα “testbench” για την προσομοίωση της λειτουργίας του κυκλώματος “part2”. Η οντότητα του “testbench” δεν έχει θύρες, καθώς χρησιμοποιείται μόνο για προσομοίωση. Η αρχιτεκτονική του περιλαμβάνει τη δήλωση του “component part2”, που συνδέεται με τα κατάλληλα σήματα εισόδου και εξόδου. Τα σήματα “SW” και “KEY” χρησιμοποιούνται ως είσοδοι για τους διακόπτες και τα πλήκτρα

αντίστοιχα, ενώ τα “LEDR” λειτουργούν ως έξοδοι που αντιστοιχούν στα “LEDs” του κυκλώματος. Η διαδικασία “SW_process” χρησιμοποιείται για την προσομοίωση των εισόδων διακοπτών. Αρχικά, το σήμα “SW(9)”, που αντιπροσωπεύει τη λειτουργία “Run”, τίθεται σε ενεργή κατάσταση ενώ το “SW(0)”, που αντιστοιχεί στο “Resetn” μηδενίζεται για να προσομοιωθεί η κατάσταση επαναφοράς. Μετά από “20 ns”, το “SW(0)” ενεργοποιείται για να αποκατασταθεί η κανονική λειτουργία του συστήματος. Η διαδικασία “vectors” διαχειρίζεται τη λειτουργία των σημάτων ρολογιού “KEY(0)” και “KEY(1)”, που αντιστοιχούν στο “MClock” και “PClock”. Τα σήματα αυτά ενεργοποιούνται και απενεργοποιούνται διαδοχικά σε χρονικά διαστήματα των “10 ns” για να προσομοιωθούν οι κύκλοι ρολογιού, που απαιτούνται για τη λειτουργία του κυκλώματος. Η διαδικασία αυτή περιλαμβάνει πολλαπλούς παλμούς ρολογιού για να καλύψει διαφορετικά σενάρια εκτέλεσης και αλληλουχίας εντολών στο κύκλωμα. Τέλος, το “WAIT”, στο τέλος κάθε διαδικασίας, επιτρέπει την αναμονή για την ολοκλήρωση της προσομοίωσης.

Προφανώς, και γι’ αυτό το “project”, πρέπει να τεθεί το “testbench” με την διαδικασία που αναφέρεται στο προηγούμενο κεφάλαιο (Βλ. εικόνες 40-43). Αφού τεθεί το “testbench.vhd” ως “test bench” και επιβεβαιωθεί ότι το model sim είναι σε σωστό «μονοπάτι» (Βλ. εικόνες 34, 35), τότε, κάνοντας “compile” και εάν το “compile” είναι επιτυχές, ακολουθεί η διαδικασία εκτέλεσης του προγράμματος από το “Model-Sim” όπως στο προηγούμενο κεφάλαιο (Βλ. εικόνες 44-48), αφού έχουν πρώτα επιλεχθεί τα σήματα προς επαλήθευση. Στην ακόλουθη εικόνα φαίνονται τα αποτελέσματα του δεύτερου μέρους (Βλ. εικόνα 62)



Εικόνα 62: Αποτελέσματα 2ου μέρους εργασίας

3.2.1 ΣΧΟΛΙΑΣΜΟΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ 2ου ΜΕΡΟΥΣ

Η εικόνα παρουσιάζει ένα στιγμιότυπο από εξομοίωση στο περιβάλλον του "Model Sim", το οποίο χρησιμοποιείται για την προσομοίωση ψηφιακών κυκλωμάτων με βάση την "VHDL". Στο αριστερό μέρος της εικόνας εμφανίζονται οι διάφορες καταχωρημένες τιμές των σημάτων, οι οποίες προσομοιώνονται, ενώ δεξιά εμφανίζεται το αντίστοιχο κυματομορφικό διάγραμμα, το οποίο δείχνει την εξέλιξη των σημάτων στο χρόνο.

Ξεκινώντας από τα αριστερά, φαίνεται μια λίστα σημάτων. Στο **δοκιμαστικό περιβάλλον** "testbench" περιλαμβάνονται σήματα που συνδέονται με την εξομοίωση. Κάποια από τα σημαντικά σήματα που παρατηρούνται, είναι το "**Resetn"**, το "**Clock"** και το "**Done"**, τα οποία χρησιμοποιούνται για τον έλεγχο και τη ροή του σχεδίου, όπως ήδη έχει προαναφερθεί και στο 3.1.1. Στην αρχή, για ένα μικρό χρονικό διάστημα, το σήμα "**Resetn"** φαίνεται να είναι χαμηλό. Αυτό μας δηλώνει την επαναφορά του συστήματος (reset). Όταν αυτό το σήμα είναι χαμηλό, οι αρχικές συνθήκες επαναφέρονται. Μετά την αποδέσμευση του reset (όταν το **Resetn** γίνεται υψηλό), το κύκλωμα αρχίζει να λειτουργεί κανονικά. Το σήμα "**Clock"** αποτελεί το ρολόι του κυκλώματος, το οποίο ταλαντώνεται. Κάθε παλμός του ρολογιού χρησιμοποιείται για τη μετάβαση της κατάστασης του κυκλώματος. Παρατηρείται ότι υπάρχουν συγκεκριμένα γεγονότα και αλλαγές στα σήματα κατά τη διάρκεια των **θετικών** και **αρνητικών ακμών** του ρολογιού. Το σήμα "**Done"** χρησιμοποιείται για να δείξει πότε ολοκληρώθηκε μια συγκεκριμένη εντολή στο κύκλωμα και αυτό φαίνεται στην προσομοίωση.

Τα σήματα "**A"** και "**B"** αντιστοιχούν σε εισόδους που δίνονται στην "**A.L.U."**. Οι τιμές τους μεταβάλλονται σε διαφορετικά χρονικά σημεία, ενώ η λειτουργία της "**A.L.U."** εξαρτάται και από το σήμα "**AddSub"**, το οποίο καθορίζει αν θα γίνει πρόσθεση ή αφαίρεση. Παρατηρείται ότι οι είσοδοι, σε σχέση με τη ροή του ρολογιού, αλλάζουν και ανάλογα με τη ρύθμιση του "**AddSub"** η έξοδος "**G"** προσαρμόζεται.

Το σήμα “**BusWires”** είναι σημαντικό, καθώς αντιπροσωπεύει τα δεδομένα που βρίσκονται στο “**Bus”** του κυκλώματος. Η τιμή του “**BusWires”** ενημερώνεται διαδοχικά, ανάλογα με την κατάσταση των υπολοίπων σημάτων και της διαδικασίας που εκτελείται. Σε ορισμένα χρονικά σημεία, οι τιμές φαίνεται να είναι "XXXX" και αυτό συνήθως δηλώνει ακαθόριστες καταστάσεις ή μεταβατικά στάδια κατά την προσομοίωση. Τα σήματα, **T0**, **T1**, **T2 και T3,** φαίνεται να αντιστοιχούν σε συγκεκριμένα **χρονικά βήματα** (states) της μηχανής πεπερασμένων καταστάσεων (Finite State Machine). Αυτά τα σήματα καθορίζουν στο κύκλωμα την ακολουθία εκτέλεσης των εντολών. Για παράδειγμα, στο **T1** συμβαίνει η αποκωδικοποίηση μιας εντολής, ενώ στο **T3** φαίνεται να εκτελείται κάποια μαθηματική πράξη ή άλλη λειτουργία. Τα σήματα, όπως “**Out”**, “**Q”** και “**Key”** χρησιμοποιούνται για την αποθήκευση και την έξοδο δεδομένων. Οι τιμές αυτών των σημάτων, φαίνεται πως, σε διάφορα χρονικά βήματα, αλλάζουν, ανάλογα με τις λειτουργίες οι οποίες έχουν οριστεί στη σχεδίαση.

Συνολικά, η εικόνα δείχνει τη λειτουργία ενός συστήματος, το οποίο χρησιμοποιεί μια μηχανή πεπερασμένων καταστάσεων, η οποία καλείται να εκτελέσει συγκεκριμένες λειτουργίες, με την “**A.L.U.”,** να παίζει σημαντικό ρόλο στις αριθμητικές πράξεις και το “**Bus”** να διακινεί τα δεδομένα μεταξύ των μονάδων. Η ακολουθία των χρονικών στιγμών και η αντιστοίχιση των σημάτων σε κάθε μία απ’ αυτές, αποδεικνύουν ότι το σύστημα και λειτουργεί με σωστή ροή και εκτελεί διαφορετικές εντολές σε συγκεκριμένα χρονικά βήματα.

ΕΠΙΛΟΓΟΣ

Συνοψίζοντας, ο επεξεργαστής, από τη στιγμή της δημιουργίας του, είναι ένα σημαντικό εργαλείο για κάθε υπολογιστικό σύστημα, ο οποίος πληροί όλες τις λογικές και αριθμητικές διεργασίες αυτού. Διαχρονικά η τεχνολογία εξελίχθηκε και έτσι βελτιώθηκε η κεντρική μονάδα επεξεργασίας έχοντας μεγάλη επιτυχία για την ανθρωπότητα. Έκτοτε οι μοντέρνοι επεξεργαστές σε σύγκριση με τους πρώτους διακρίνονται για την απόδοση και την σύνθετη αρχιτεκτονική τους.

Τα ψηφιακά κυκλώματα είναι μια καινοτομία συστημάτων που έχουν εφαρμογή στον κλάδο της βιομηχανίας για να ικανοποιηθούν οι απαιτήσεις της σύγχρονης κοινωνίας. Τα κυκλώματα αυτά σχεδιάζονται μέσα από πολλά προγράμματα ανάπτυξης λογισμικού όπως το “Quartus”, το οποίο διευκολύνει στη δημιουργία σύνθετων αρχιτεκτονικών. Επιπρόσθετα, ένα από τα κύρια εργαλεία του “Quartus” είναι η γλώσσα περιγραφής υλικού “V.H.D.L.”, η οποία χρησιμοποιείται για να υλοποιήσει και να ελέγξει περίπλοκα συστήματα.

Ένα επιπλέον εργαλείο του παραπάνω προγράμματος είναι το Model-Sim, το οποίο επιτρέπει στον χρήστη να προσομοιώσει τον επεξεργαστή ώστε να επαληθευτεί η ορθότητα των αποτελεσμάτων του. Οι εντολές προς εκτέλεση φορτώνονται μέσω ενός αρχείου “test bench” για να διαπιστώσει ο σχεδιαστής ότι λειτουργούν με τον σωστό τρόπο. Στη συνέχεια, με τη χρήση μιας μνήμης “R.O.M.” και ενός νέου “test bench”, οι εντολές διαβάζονται μέσα από αυτή με αποτέλεσμα να δημιουργηθεί ένα αυτόνομο σύστημα επεξεργαστή.

Τέλος, η εξέλιξη των επεξεργαστών και των ψηφιακών κυκλωμάτων έχει οδηγήσει στη δημιουργία αποδοτικών και σύνθετων υπολογιστικών συστημάτων. Με εργαλεία όπως το “Quartus”, το “Model – Sim” και τη γλώσσα “V.H.D.L.”, οι σχεδιαστές μπορούν να υλοποιούν, να προσομοιώνουν και να επαληθεύουν πολύπλοκες αρχιτεκτονικές, συμβάλλοντας στην ανάπτυξη αυτόνομων και αξιόπιστων συστημάτων για τη βιομηχανία και την κοινωνία.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Jason Auerbach, Vineeta Agarwala, Kimber Lockhart, 2005, 64bit Processors διαθέσιμο στον ιστότοπο:
https://cs.stanford.edu/people/eroberts/courses/soco/projects/2005-06/64-bit-processors/history1.html

Hennessy, J. L., & Patterson, D. A. (2017). Computer Architecture: A Quantitative Approach. Morgan Kaufmann, διαθέσιμο στον Ιστότοπο:

https://acs.pub.ro/~cpop/SMPA/Computer%20Architecture%20A%20Quantitative%20Approach%20(5th%20edition).pdf

Tanenbaum, A. S., & Austin, T. (2012). Structured Computer Organization. Pearson διαθέσιμο στον ιστότοπο:

https://csc-knu.github.io/sys-prog/books/Andrew%20S.%20Tanenbaum%20-%20Structured%20Computer%20Organization.pdf

Intel, assignment vhdl_lab9:

file:///C:/Users/user/Desktop/vhdl_lab9%20(1).pdf

Intel Corporation, 2013,Altera Online Presentation VHDL Basics, διαθέσιμο στον ιστότοπο:

https://www.intel.com/content/www/us/en/programmable/customertraining/webex/VHDL/presentation_html5.html

Intel Corporation, 2023, Features of Quartus Prime, Διαθέσιμο στον ιστότοπο:
https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime/article.html

Intel Corporation, Explore Intel's History Διαθέσιμο στον ιστότοπο:
https://timeline.intel.com/1971/the-first-programmable-microprocessor:-the-4004

AMD, 2023, Ryzen 9 7950X3D specs, Διαθέσιμο στον ιστότοπο:

https://www.amd.com/en/products/apu/amd-ryzen-9-7950x3d

Paul A. Freiberger, Michael R. Swaine, 2023, Διαθέσιμο στον ιστότοπο:

https://www.britannica.com/technology/ENIAC

Wikipedia ARM (αρχιτεκτονική), 2022, Διαθέσιμο στον ιστότοπο:

https://el.wikipedia.org/wiki/ARM_(αρχιτεκτονική)

Furber, S. 2000, ARM System-on-Chip Architecture, διαθέσιμο στον ιστότοπο:

https://www.ele.uva.es/~jesman/BigSeti/ftp/Microcontroladores/ARM/Arm%20System-On-Chip%20Architecture.pdf

Morris Mano & Michael Ciletti, Digital Design with an Introduction to the Verilog HDL, VHDL, and SystemVerilog, Pearson, 6η Έκδοση, 2017, Διαθέσιμο στον ιστότοπο :

https://api.pageplace.de/preview/DT0400.9781292231181_A37760511/preview-9781292231181_A37760511.pdf

Γιάννης Αμπατζόγλου, 2018, Διαθέσιμο στον ιστότοπο :

https://users.sch.gr/jabatzo/files/yliko/live%20ebooks/psifiaka_G_2018_final/____2.html

Γιάννης Αμπατζόγλου «Ψηφιακά Συστήματα» 2018, διαθέσιμο στον ιστότοπο :

https://users.sch.gr/jabatzo/files/yliko/live%20ebooks/psifiaka_G_2018_final/__3.html

Oshadee Gangangana «Von-Neumann architecture vs Harvard architecture» 2021, διαθέσιμο στον ιστότοπο:

https://oshadeegangangana.medium.com/von-neumann-architecture-vs-harvard-architecture-fdb20d6f2a70

Teach Computer Science «RISC and CISC Processors», διαθέσιμο στον ιστότοπο:

https://teachcomputerscience.com/risc-and-cisc-processors/

Wikipedia «ENIAC», διαθέσιμο στον ιστότοπο:

https://el.wikipedia.org/wiki/ENIAC