ΠΑΝΕΠΙΣΤΉΜΙΟ ΙΩΑΝΝΊΝΩΝ

ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδίαση και Υλοποίηση ενός Βελτιωμένου Επεξεργαστή

Γεώργιος-Παύλος Χριστοφόρου

Α.Μ. : 1783

Επιβλέπων: Φώτιος Βαρτζιώτης

Επίκουρος καθηγητής

Άρτα , Ιούνιος 2026

An Enhanced Processor

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα,18 Ιουνίου 2026

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής
   Φώτιος Βαρτζιώτης

2. Μέλος επιτροπής
   Γρηγόριος Δουμένης

3. Μέλος επιτροπής
   Σπυριδούλα Μαργαρίτη

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα μεταπτυχιακή 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.

Χριστοφόρου,Γεώργιος-Παύλος

Υπογραφή

ΕΥΧΑΡΙΣΤΙΕΣ

Με την ολοκλήρωση της πτυχιακής μου εργασίας, θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου, κ. Φώτιο Βαρτζιώτη, για την καθοδήγηση, τον χρόνο και την υποστήριξη που μου προσέφερε σε όλα τα στάδια της εκπόνησής της.

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία πραγματεύεται τη σχεδίαση και υλοποίηση ενός βελτιωμένου επεξεργαστή (Enhanced Processor) 16-bit με τη χρήση της γλώσσας περιγραφής υλικού VHDL. Ο επεξεργαστής βασίζεται σε μια αρχιτεκτονική που ενσωματώνει τον Program Counter (PC) στο register file, επιτρέποντας την αυτόνομη ανάκτηση και εκτέλεση εντολών από τη μνήμη. Στο πλαίσιο της εργασίας, υλοποιήθηκε η διασύνδεση του επεξεργαστή με εξωτερική μνήμη RAM και περιφερειακές μονάδες εισόδου/εξόδου (διακόπτες και LED) μέσω ενός συστήματος αποκωδικοποίησης διευθύνσεων. Η λειτουργία του συστήματος επαληθεύτηκε μέσω εκτενών προσομοιώσεων σε περιβάλλον ModelSim, όπου εξετάστηκε η ορθή εκτέλεση εντολών μεταφοράς δεδομένων, αριθμητικών πράξεων και αλμάτων υπό συνθήκη. Τα αποτελέσματα δείχνουν ότι ο επεξεργαστής είναι σε θέση να εκτελεί σύνθετα προγράμματα Assembly, αποτελώντας μια στέρεη βάση για την κατανόηση των σύγχρονων υπολογιστικών συστημάτων.

Λέξεις-κλειδιά: VHDL, Επεξεργαστής, FPGA, Program Counter, Μηχανή Καταστάσεων (FSM).

ABSTRACT

This thesis presents the design and implementation of an enhanced 16-bit processor using the VHDL hardware description language. The processor is based on an architecture that integrates the Program Counter (PC) into the register file, enabling autonomous instruction fetch and execution from memory. As part of this work, the processor was interfaced with external RAM and input/output peripherals (switches and LEDs) through an address decoding system. The system's functionality was verified through extensive simulations in the ModelSim environment, validating the correct execution of data transfer, arithmetic, and conditional branch instructions. The results demonstrate that the processor is capable of executing complex Assembly programs, providing a solid foundation for understanding modern computing systems.

Keywords: VHDL, Processor, FPGA, Program Counter, Finite State Machine (FSM).

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΑΤΑΛΟΓΟΣ ΔΙΑΓΡΑΜΜΑΤΩΝ/ΕΙΚΟΝΩΝ

ΠΙΝΑΚΑΣ ΣΥΝΤΟΜΟΓΡΑΦΙΩΝ

VHDL: VHSIC Hardware Description Language (Γλώσσα Περιγραφής Υλικού Ολοκληρωμένων Κυκλωμάτων Πολύ Υψηλής Ταχύτητας)

FPGA: Field-Programmable Gate Array (Πεδίο Προγραμματιζόμενων Πυλών Διάταξης)

RTL: Register-Transfer Level (Επίπεδο Μεταφοράς Καταχωρητών)

ALU: Arithmetic Logic Unit (Αριθμητική και Λογική Μονάδα)

PC: Program Counter (Μετρητής Προγράμματος)

RAM: Random Access Memory (Μνήμη Τυχαίας Προσπέλασης)

ROM: Read-Only Memory (Μνήμη Ανάγνωσης Μόνο)

MIF: Memory Initialization File (Αρχείο Αρχικοποίησης Μνήμης)

HEX: Hexadecimal File (Δεκαεξαδικό Αρχείο)

LED: Light Emitting Diode (Δίοδος Εκπομπής Φωτός)

CPU: Central Processing Unit (Κεντρική Μονάδα Επεξεργασίας)

DIN: Data Input (Δίαυλος Δεδομένων Εισόδου)

DOUT: Data Output (Δίαυλος Δεδομένων Εξόδου)

ΑΠΟΔΟΣΗ ΟΡΩΝ / ΓΛΩΣΣΑΡΙΟ

Simulation: Προσομοίωση

Testbench: Κώδικας Επαλήθευσης

Hardware Description Language: Γλώσσα Περιγραφής Υλικού

Waveforms: Κυματομορφές (χρονισμού)

Compiler / Synthesis: Μεταcompiler / Σύνθεση (κυκλώματος)

Signal: Σήμα

Clock: Ρολόι (Σήμα Συγχρονισμού)

Reset: Σήμα Αρχικοποίησης

Data Bus: Δίαυλος Δεδομένων

ΕΙΣΑΓΩΓΗ

Ιστορική Αναδρομή και η Εξέλιξη των Επεξεργαστών

Η σύγχρονη ψηφιακή τεχνολογία βασίζεται αναμφισβήτητα στην αλματώδη εξέλιξη των συστημάτων υπολογιστών, η οποία ξεκίνησε δυναμικά στις αρχές της δεκαετίας του 1970 με την εμφάνιση των πρώτων μικροεπεξεργαστών περιορισμένης αρχιτεκτονικής 4-bit και 8-bit. Έκτοτε, η σχεδίαση των επεξεργαστών εξελίσσεται διαρκώς με στόχο τη μεγιστοποίηση της υπολογιστικής ισχύος, τη βελτιστοποίηση της διαχείρισης πόρων και τη μείωση της ενεργειακής κατανάλωσης. Στο σύγχρονο βιομηχανικό και ερευνητικό περιβάλλον, η ανάγκη για ευέλικτες, γρήγορες και χαμηλού κόστους ψηφιακές λύσεις οδήγησε στην κυριαρχία των αναπρογραμματιζόμενων διατάξεων FPGA (Field-Programmable Gate Arrays) και των Γλωσσών Περιγραφής Υλικού (HDLs). Οι τεχνολογίες αυτές έφεραν επανάσταση, καθώς επιτρέπουν την πλήρη ανάπτυξη και δοκιμή custom αρχιτεκτονικών σε επίπεδο υλικού (hardware), παρακάμπτοντας το εξαιρετικά υψηλό κόστος και τον χρόνο που απαιτεί η κατασκευή ενός παραδοσιακού ολοκληρωμένου κυκλώματος ειδικού σκοπού (ASIC).

Αντικείμενο και Σκοπός της Πτυχιακής Εργασίας

Στο πλαίσιο αυτό, αντικείμενο της παρούσας πτυχιακής εργασίας αποτελεί η σχεδίαση, η ανάλυση και η υλοποίηση ενός βελτιωμένου ψηφιακού επεξεργαστή (Enhanced Processor) αρχιτεκτονικής 16-bit, με τη χρήση της γλώσσας VHDL (VHSIC Hardware Description Language). Η εργασία επικεντρώνεται στη δημιουργία ενός αυτόνομου υπολογιστικού συστήματος, ικανού να εκτελεί αυτόματα εντολές απευθείας από τη μνήμη, να διαχειρίζεται περιφερειακές συσκευές εισόδου/εξόδου και να συντονίζει τις λειτουργίες του μέσω μιας κεντρικής Μηχανής Πεπερασμένων Καταστάσεων (Finite State Machine - FSM).

Ο θεμελιώδης στόχος της παρούσας μελέτης είναι η αναβάθμιση από το απλό μοντέλο ενός επεξεργαστή που ελέγχεται χειροκίνητα από έναν εξωτερικό μετρητή, σε ένα πλήρως αυτοματοποιημένο και κυρίαρχο σύστημα. Η καινοτομία της προτεινόμενης αρχιτεκτονικής έγκειται στην ενσωμάτωση του Μετρητή Προγράμματος (Program Counter - PC) απευθείας μέσα στο αρχείο καταχωρητών (Register File) ως ο καταχωρητής $r7$. Η δομική αυτή αλλαγή αυτοματοποιεί πλήρως τη διαδικασία

ανάκτησης εντολών (Instruction Fetch) και επιτρέπει την υλοποίηση εντολών διακλάδωσης (branching) και αλμάτων υπό συνθήκη. Η δυνατότητα αυτή είναι καθοριστική, καθώς επιτρέπει στον επεξεργαστή να εκτελεί σύνθετα προγράμματα με βρόχους επανάληψης (loops) και υπορουτίνες, καθιστώντας τον μια ολοκληρωμένη και λειτουργική υπολογιστική μονάδα γενικής χρήσης.

Δομή της Εργασίας και Επεξήγηση Κεφαλαίων

Για την ομαλή και συστηματική παρουσίαση του θέματος, η εργασία διαρθρώνεται στα ακόλουθα πέντε κεφάλαια:

- **Κεφάλαιο 1 - Θεωρητικό Υπόβαθρο:** Παρουσιάζονται οι βασικές τεχνολογικές έννοιες που πλαισιώνουν την εργασία. Αναλύονται τα χαρακτηριστικά των FPGAs, οι Γλώσσες Περιγραφής Υλικού (VHDL και Verilog), καθώς και τα εργαλεία λογισμικού Quartus Prime και ModelSim που χρησιμοποιήθηκαν για τη σύνθεση και την προσομοίωση.

- **Κεφάλαιο 2 - Αρχιτεκτονική του Επεξεργαστή:** Περιγράφεται αναλυτικά η δομή του 16-bit συστήματος, το σύνολο εντολών (Instruction Set), οι καταχωρητές, η Μονάδα Ελέγχου και ο τρόπος με τον οποίο ο Program Counter ενσωματώνεται ως καταχωρητής $r7$.

- **Κεφάλαιο 3 - Υλοποίηση σε VHDL:** Παρουσιάζεται η συγγραφή του κώδικα και η δομική σχεδίαση των επιμέρους μονάδων (επεξεργαστής, μνήμη, οντότητα κορυφής), αναλύοντας πώς μεταφράζεται η θεωρητική αρχιτεκτονική σε πραγματικό ψηφιακό κύκλωμα.

- **Κεφάλαιο 4 - Έλεγχος και Αποτελέσματα Προσομοίωσης:** Αναλύεται η συμπεριφορά του επεξεργαστή κατά την εκτέλεση των εντολών. Παρουσιάζονται οι χρονισμοί και οι κυματομορφές (waveforms) στο ModelSim, ενώ γίνεται ειδική αναφορά στη χρήση του διαδραστικού εξομοιωτή DeSim για την οπτική αναπαράσταση της λειτουργίας της πλακέτας.

- **Κεφάλαιο 5 - Συμπεράσματα και Μελλοντικές Επεκτάσεις:** Συνοψίζονται τα τελικά συμπεράσματα που προέκυψαν από την υλοποίηση και προτείνονται πιθανές αναβαθμίσεις για τη βελτίωση του επεξεργαστή στο μέλλον.

1. ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ ΚΑΙ ΤΕΧΝΟΛΟΓΙΕΣ

1.1 Ψηφιακά Συστήματα και Τεχνολογία FPGA

Τα σύγχρονα ψηφιακά συστήματα βασίζονται στην ικανότητα επεξεργασίας πληροφοριών σε δυαδική μορφή. Για την υλοποίηση τέτοιων συστημάτων, πέρα από τους κλασικούς μικροελεγκτές, χρησιμοποιούνται ευρέως τα FPGA (Field Programmable Gate Arrays). Τα FPGA είναι ολοκληρωμένα κυκλώματα τα οποία αποτελούνται από έναν πίνακα προγραμματιζόμενων λογικών στοιχείων (Logic Blocks) και διασυνδέσεων, τα οποία μπορούν να διαμορφωθούν από τον τελικό χρήστη/σχεδιαστή μετά την κατασκευή τους.

Η σημασία των FPGA στην αρχιτεκτονική υπολογιστών είναι θεμελιώδης, καθώς επιτρέπουν στον σχεδιαστή να δημιουργήσει προσαρμοσμένο υλικό (hardware) που εκτελεί συγκεκριμένες λειτουργίες με εξαιρετικά μεγάλη ταχύτητα μέσω φυσικής παραλληλίας. Στην παρούσα εργασία, η χρήση της τεχνολογίας FPGA επιτρέπει την υλοποίηση και τον έλεγχο του σχεδιασμένου 16-bit επεξεργαστή σε πραγματικές συνθήκες λειτουργίας.

1.2 Η Γλώσσα Περιγραφής Υλικού VHDL

Η VHDL (VHSIC Hardware Description Language) αποτελεί μια από τις πιο διαδεδομένες και ισχυρές γλώσσες περιγραφής υλικού παγκοσμίως, η οποία χρησιμοποιείται για τη σχεδίαση, την προσομοίωση και την υλοποίηση ψηφιακών συστημάτων μεγάλης κλίμακας. Αναπτύχθηκε αρχικά τη δεκαετία του 1980 υπό την αιγίδα του Υπουργείου Άμυνας των ΗΠΑ (DoD) με σκοπό την τυποποίηση και την τεκμηρίωση της συμπεριφοράς των ψηφιακών κυκλωμάτων. Το 1987, η γλώσσα τυποποιήθηκε επίσημα από τον οργανισμό IEEE (πρότυπο IEEE 1076), καθιερώνοντας μια κοινή πλατφόρμα επικοινωνίας για τους σχεδιαστές ψηφιακών συστημάτων και τα εργαλεία αυτοματοποιημένης σχεδίασης (EDA tools).

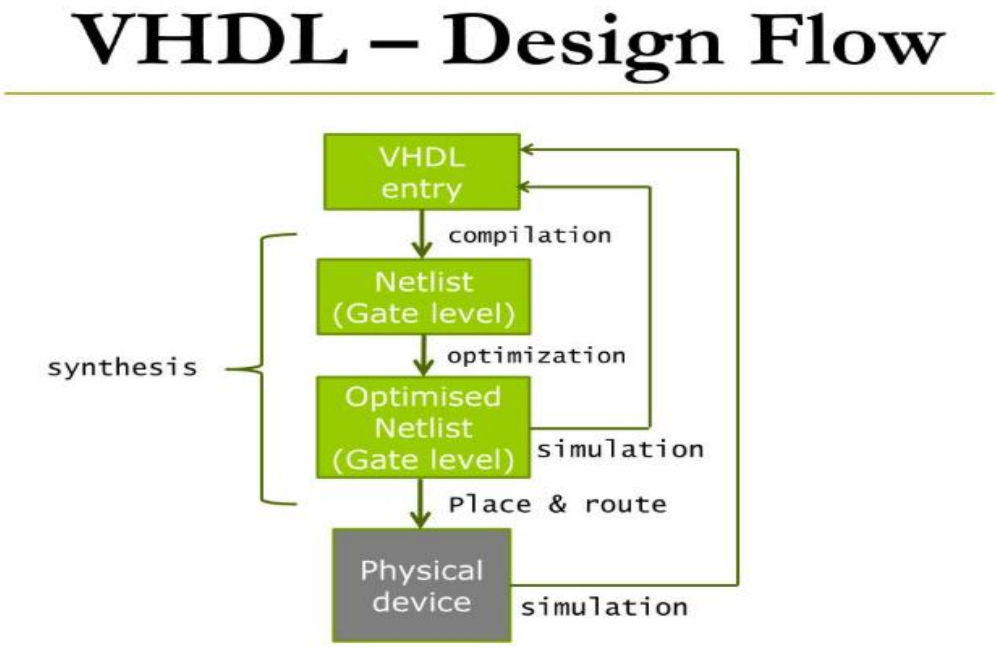
Η αρχιτεκτονική δομή ενός ψηφιακού συστήματος σε VHDL βασίζεται σε δύο κύρια συστατικά: την **Οντότητα (Entity)** και την **Αρχιτεκτονική (Architecture)**. Η οντότητα καθορίζει τη διεπαφή (interface) του κυκλώματος με τον εξωτερικό κόσμο, ορίζοντας τις θύρες εισόδου και εξόδου (ports) καθώς και τους τύπους των δεδομένων

τους. Αντίστοιχα, η αρχιτεκτονική περιγράφει την εσωτερική λειτουργία ή τη δομική διασύνδεση του κυκλώματος, καθορίζοντας τον τρόπο με τον οποίο οι είσοδοι μετασχηματίζονται σε εξόδους.

Η σχεδιαστική ροή (Design Flow) σε περιβάλλον VHDL ξεκινά από την κωδικοποίηση της συμπεριφοράς ή της δομής του συστήματος και χωρίζεται σε δύο θεμελιώδη στάδια: την **Προσομοίωση (Simulation)** και τη **Λογική Σύνθεση (Logic Synthesis)**.

Κατά την προσομοίωση, ο κώδικας ελέγχεται λειτουργικά σε ένα εικονικό περιβάλλον (όπως το ModelSim) χρησιμοποιώντας ένα κατάλληλο ψηφιακό ερέθισμα (Testbench). Σκοπός αυτού του σταδίου είναι να επιβεβαιωθεί ότι οι έξοδοι ανταποκρίνονται σωστά στα ερεθίσματα των εισόδων σε σχέση με τον χρόνο, χωρίς να λαμβάνονται ακόμη υπόψη οι φυσικοί περιορισμοί και οι καθυστερήσεις του πραγματικού υλικού.

Αντίθετα, η Λογική Σύνθεση είναι η διαδικασία κατά την οποία το εργαλείο ανάπτυξης (όπως το Quartus) αναλύει τον αφαιρετικό κώδικα VHDL και τον μετατρέπει σε ένα δίκτυο πυλών (Netlist) και δομικών στοιχείων. Αυτά τα στοιχεία μπορούν στη συνέχεια να αποτυπωθούν απευθείας στα Logic Elements (LEs) και στα εσωτερικά Flip-Flops ενός πραγματικού ολοκληρωμένου κυκλώματος FPGA (Field Programmable Gate Array).



Εικόνα 1.1: Τυπική ροή σχεδίασης ψηφιακών συστημάτων με χρήση γλωσσών περιγραφής υλικού (HDL). (*Πηγή: slideserve.com)*

Η βασικότερη διαφορά της VHDL από τις κλασικές γλώσσες προγραμματισμού λογισμικού (όπως η C, η C++ ή η Python) εντοπίζεται στην έννοια της **παραλληλίας (concurrency)**. Ενώ στις γλώσσες λογισμικού οι εντολές εκτελούνται αυστηρά σειριακά, η μία μετά την άλλη από την CPU, στη VHDL οι διεργασίες (processes) και οι αναθέσεις σημάτων περιγράφουν φυσικό υλικό (hardware) το οποίο λειτουργεί ταυτόχρονα και παράλληλα στον χρόνο.

Αυτή η εγγενής παραλληλία επιτρέπει τη μοντελοποίηση σύνθετων ψηφιακών συστημάτων με υψηλή ταχύτητα εκτέλεσης, απαιτεί όμως από τον σχεδιαστή μια διαφορετική προσέγγιση (hardware-oriented thinking), καθώς κάθε γραμμή κώδικα μεταφράζεται σε φυσικά καλώδια, πύλες και καταχωρητές που λειτουργούν συγχρονισμένα υπό την καθοδήγηση ενός κοινού σήματος ρολογιού (clock).

1.3 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΕΠΕΞΕΡΓΑΣΤΩΝ RISC

Η αρχιτεκτονική RISC (Reduced Instruction Set Computer - Υπολογιστής Μειωμένου Συνόλου Εντολών) αποτελεί τη βάση σχεδίασης των περισσότερων σύγχρονων και αποδοτικών επεξεργαστών. Εμφανίστηκε στις αρχές της δεκαετίας του 1980 ως απάντηση στην αρχιτεκτονική CISC (Complex Instruction Set Computer), με σκοπό τη ριζική απλοποίηση του υλικού (hardware) και τη δραστική αύξηση της ταχύτητας εκτέλεσης των εντολών. Η βασική φιλοσοφία των RISC συστημάτων στηρίζεται στη διαπίστωση ότι η συντριπτική πλειονότητα των προγραμμάτων χρησιμοποιεί κυρίως ένα μικρό και απλό υποσύνολο εντολών, το οποίο μπορεί να βελτιστοποιηθεί ώστε να εκτελείται εξαιρετικά γρήγορα.

Σε αντίθεση με τους επεξεργαστές CISC, οι οποίοι διαθέτουν εκατοντάδες εντολές μεταβλητού μήκους που απαιτούν πολλούς κύκλους ρολογιού για να ολοκληρωθούν, οι επεξεργαστές RISC χαρακτηρίζονται από τις εξής θεμελιώδεις σχεδιαστικές αρχές:

1. **Σταθερό Μήκος Εντολών:** Όλες οι εντολές έχουν το ίδιο ακριβώς μέγεθος (στην παρούσα πτυχιακή εργασία επιλέχθηκε το μέγεθος των 16-bit). Αυτό απλοποιεί δραστικά τη φάση της ανάκτησης (Fetch) και της αποκωδικοποίησης (Decode) από τη Μονάδα Ελέγχου, καθώς ο επεξεργαστής γνωρίζει εκ των προτέρων τα ακριβή όρια κάθε εντολής στη μνήμη.

2. **Εκτέλεση σε Έναν Κύκλο Ρολογιού:** Οι εντολές είναι τόσο απλές που η εκτέλεσή τους (Execute) ολοκληρώνεται ιδανικά σε έναν μόνο κύκλο ρολογιού, επιτρέποντας υψηλό ρυθμό επεξεργασίας δεδομένων.

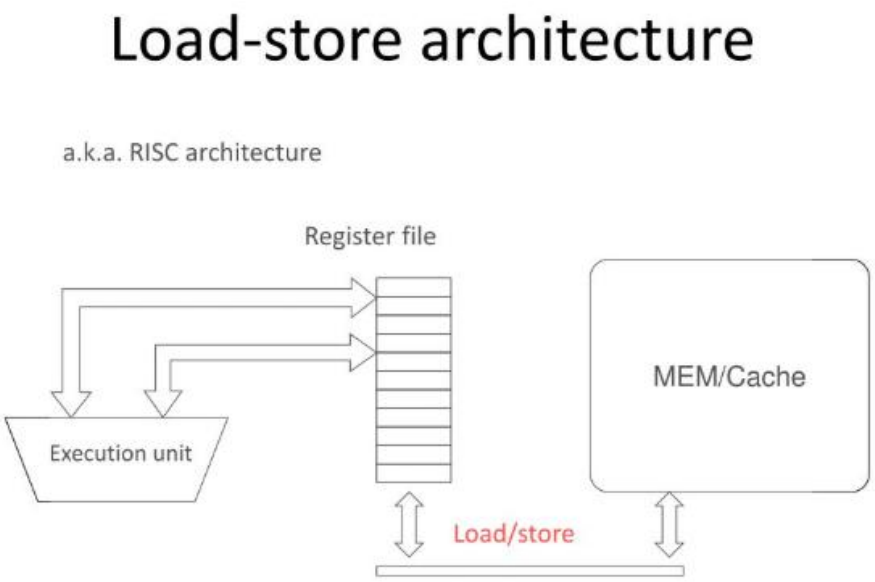
3. **Μεγάλο Αρχείο Καταχωρητών (Register File):** Οι RISC επεξεργαστές βασίζονται σε έναν σημαντικό αριθμό εσωτερικών καταχωρητών γενικής χρήσης για την προσωρινή αποθήκευση των δεδομένων, μειώνοντας έτσι τις αργές προσπελάσεις στην εξωτερική μνήμη.

Ένα από τα πιο κρίσιμα και αναγνωρίσιμα χαρακτηριστικά των αρχιτεκτονικών RISC είναι η δομή **Load/Store**. Αυτό σημαίνει ότι οι αριθμητικές και λογικές πράξεις που πραγματοποιούνται από την Αριθμητική και Λογική Μονάδα (ALU) εκτελούνται αποκλειστικά και μόνο μεταξύ των εσωτερικών καταχωρητών του επεξεργαστή.

Η επικοινωνία με την κύρια μνήμη (RAM) διαχωρίζεται πλήρως από τις πράξεις και περιορίζεται αυστηρά σε δύο βασικές εντολές:

- Την εντολή **ld (Load)**, η οποία ανακτά ένα δεδομένο από μια διεύθυνση μνήμης και το αποθηκεύει σε έναν εσωτερικό καταχωρητή.

- Την εντολή **st (Store)**, η οποία παίρνει το περιεχόμενο ενός εσωτερικού καταχωρητή και το γράφει πίσω σε μια θέση μνήμης.



Εικόνα 1.2: Γενικό δομικό διάγραμμα αρχιτεκτονικής ενός επεξεργαστή RISC Load/Store. (*Πηγή: slideserve.com)*

Αυτή η προσέγγιση Load/Store επιτρέπει την πλήρη αποσύνδεση των χρονικά απαιτητικών προσπελάσεων της μνήμης από τις γρήγορες, εσωτερικές λειτουργίες της ALU. Ως αποτέλεσμα, η Μονάδα Ελέγχου του επεξεργαστή γίνεται πολύ πιο απλή, καταναλώνει λιγότερους πόρους στο FPGA και μπορεί να χρονιστεί σε σημαντικά υψηλότερες συχνότητες λειτουργίας, προσφέροντας βέλτιστη απόδοση κατά την εκτέλεση του ψηφιακού κώδικα.

1.4 Οργάνωση Μνήμης και Διαύλων (Buses)

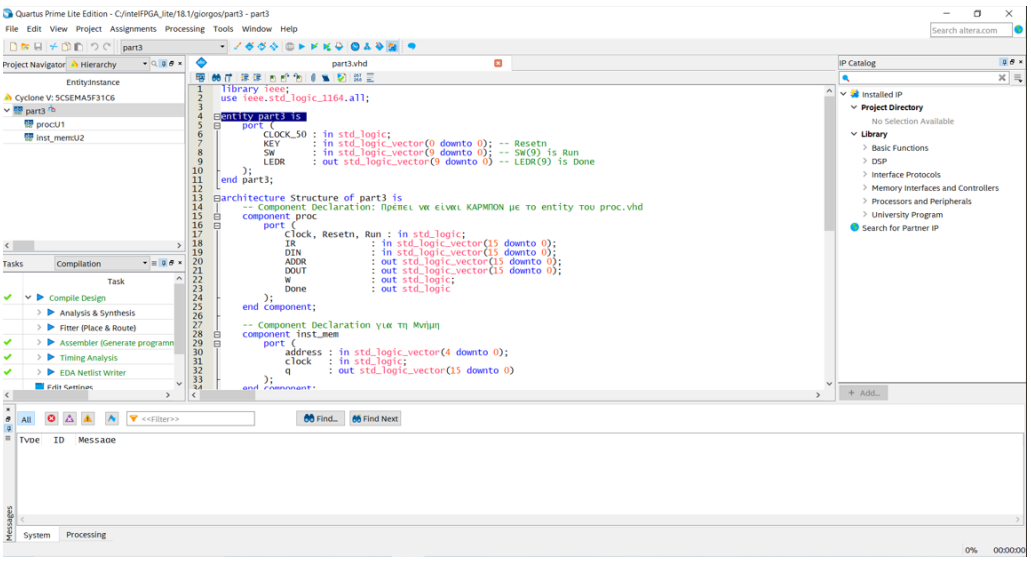
Για τη λειτουργία του επεξεργαστή, η επικοινωνία μεταξύ των εσωτερικών καταχωρητών, της Αριθμητικής και Λογικής Μονάδας (ALU) και της εξωτερικής μνήμης γίνεται μέσω διαύλων. Ο δίαυλος (Bus) είναι μια ομάδα κοινών γραμμών μεταφοράς πληροφορίας. Στην εργασία μας χρησιμοποιείται ένας κεντρικός δίαυλος (Buswires) εύρους 16-bit, στον οποίο συνδέονται όλες οι επιμέρους μονάδες. Η αποφυγή διενέξεων στον δίαυλο επιτυγχάνεται μέσω πολυπλεκτών (multiplexers) και τρίκαταστατων απομονωτών (tri-state buffers). Η επιλογή του ποιο δεδομένο θα "οδηγήσει" τον δίαυλο και ποιο θα αναγνώσει από αυτόν καθορίζεται δυναμικά από τη Μονάδα Ελέγχου σε κάθε χρονικό βήμα.

1.5 Εργαλεία Σχεδίασης και Προσομοίωσης: Quartus και ModelSim

Για τη σχεδίαση, τη δοκιμή και την ολοκληρωμένη λειτουργική επαλήθευση του προτεινόμενου επεξεργαστή RISC, χρησιμοποιήθηκε το σύγχρονο βιομηχανικό περιβάλλον ανάπτυξης **Intel Quartus Prime** σε άμεση συνδυαστική λειτουργία με το εξειδικευμένο λογισμικό προσομοίωσης **ModelSim - Intel FPGA Edition**. Τα εργαλεία αυτά αποτελούν μια ολοκληρωμένη πλατφόρμα αυτοματοποιημένης σχεδίασης (EDA tools), η οποία επιτρέπει τη μετάβαση από τον αφαιρετικό κώδικα περιγραφής υλικού στην τελική φυσική αποτύπωση πάνω σε chip σιλικόνης.

Το λογισμικό **Intel Quartus Prime** ανέλαβε την κεντρική διαχείριση του project και τη διαδικασία της **Λογικής Σύνθεσης (Logic Synthesis)**. Κατά το στάδιο αυτό, ο κώδικας VHDL αναλύεται συντακτικά και μετατρέπεται σε ένα δίκτυο διασυνδεδεμένων πυλών (Netlist), το οποίο στη συνέχεια αντιστοιχίζεται (mapping) πάνω στους πραγματικούς πόρους του FPGA (Logic Elements, ενσωματωμένα blocks μνήμης και Flip-Flops).

Ένα από τα πιο χρήσιμα εργαλεία του Quartus που χρησιμοποιήθηκαν κατά τη σχεδίαση είναι ο **RTL Viewer**. Ο RTL Viewer επιτρέπει στον σχεδιαστή να επιθεωρήσει οπτικά το παραγόμενο hardware σε μορφή σχηματικού διαγράμματος πριν από την τελική υλοποίηση. Με τον τρόπο αυτό, επιβεβαιώνεται ότι οι εντολές και οι διεργασίες της VHDL έχουν μεταφραστεί με ακρίβεια στους επιθυμητούς καταχωρητές, πολυπλέκτες και αριθμητικές μονάδες.



Εικόνα 1.3: Στιγμιότυπο από το περιβάλλον σχεδίασης και σύνθεσης Intel Quartus Prime.

Παράλληλα, το λογισμικό ModelSim αποτέλεσε το βασικό εργαλείο για τη λειτουργική επαλήθευση (Functional Verification) του επεξεργαστή. Επειδή η σχεδίαση ψηφιακών κυκλωμάτων απαιτεί απόλυτη ακρίβεια, είναι αδύνατο να μεταφερθεί ένας κώδικας στο hardware χωρίς προηγουμένως να ελεγχθεί η συμπεριφορά του στον χρόνο. Μέσω του ModelSim, εκτελείται το Testbench (ένα ειδικό αρχείο VHDL που δημιουργεί τα σήματα του ρολογιού και της επανεκκίνησης) και αποτυπώνονται οι εσωτερικές λειτουργίες του επεξεργαστή σε μορφή χρονικών κυματομορφών (Waveforms). Ο σχεδιαστής μπορεί να παρακολουθήσει την τιμή του Program Counter (PC), τις μεταβολές στον κεντρικό δίαυλο (BusWire), τα περιεχόμενα των καταχωρητών του Register File, καθώς και την τρέχουσα κατάσταση της FSM σε κάθε παλμό ρολογιού. Η λεπτομερής αυτή παρατήρηση καθιστά εφικτό τον εντοπισμό και τη διόρθωση λογικών σφαλμάτων (debugging), διασφαλίζοντας ότι ο επεξεργαστής εκτελεί

τις εντολές assembly με απόλυτη ορθότητα. Τα αναλυτικά στιγμιότυπα και οι χρονικές κυματομορφές (Waveforms) της προσομοίωσης που παρήχθησαν από το εργαλείο για το πρόγραμμα ελέγχου, παρουσιάζονται εκτενώς στο Κεφάλαιο 4 κατά τη διαδικασία της λειτουργικής επαλήθευσης του συστήματος.

2. ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΗ

2.1 Γενική Περιγραφή

Ο επεξεργαστής που υλοποιήθηκε είναι ένα ψηφιακό σύστημα 16-bit, σχεδιασμένο με γνώμονα την εκπαιδευτική σαφήνεια και την αποδοτικότητα σε περιβάλλον FPGA. Η αρχιτεκτονική του βασίζεται σε έναν κεντρικό δίαυλο δεδομένων (Bus), μέσω του οποίου επικοινωνούν οι καταχωρητές, η αριθμητική και λογική μονάδα (ALU) και η εξωτερική μνήμη.

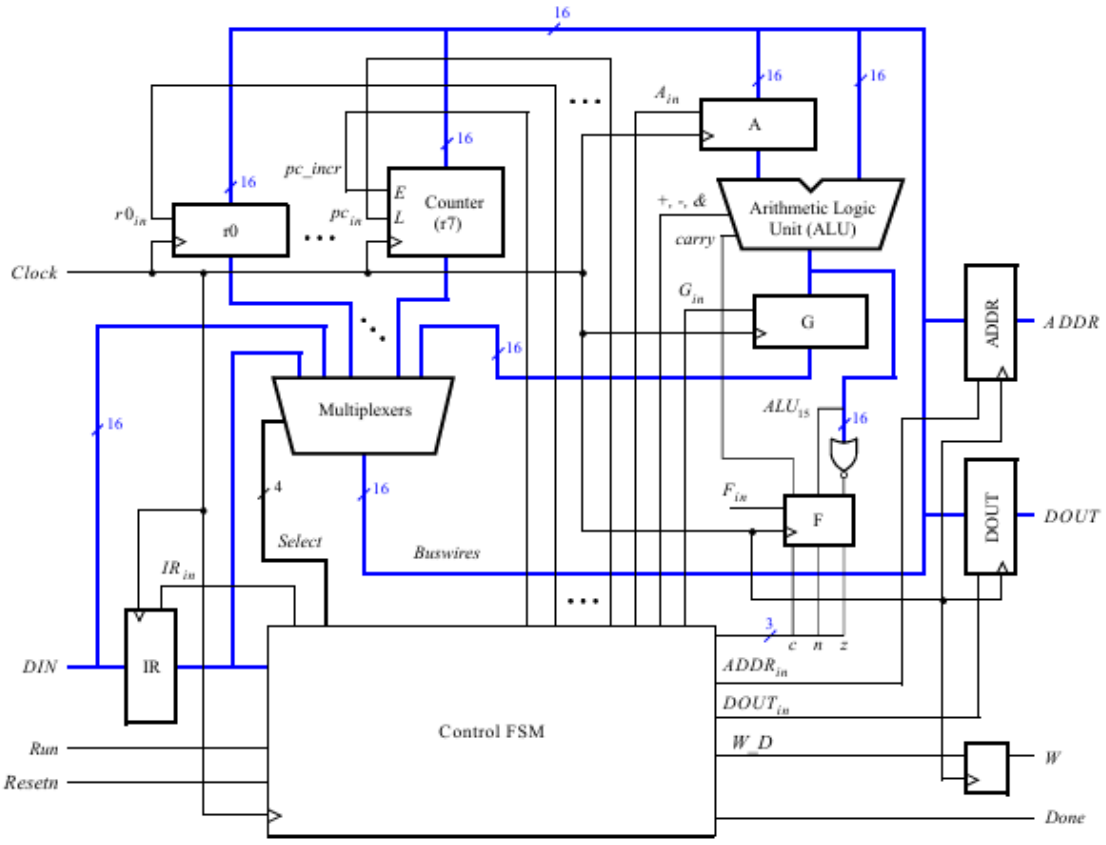
Operation	Function performed
ld rX, [rY]	$rX \leftarrow [rY]$
st rX, [rY]	$[rY] \leftarrow rX$
and rX, Op2	$rX \leftarrow rX \,\&\, Op2$
b{cond} Label	if (*cond*), $pc \leftarrow Label$

Πίνακας 2.1: Το διευρυμένο σύνολο εντολών (Instruction Set) του επεξεργαστή.

2.2 Το Αρχείο Καταχωρητών (Register File)

Το σύστημα διαθέτει οκτώ γενικούς καταχωρητές (r0 έως r7), ο καθένας μεγέθους 16-bit.

- Καταχωρητές r0-r6: Χρησιμοποιούνται για την προσωρινή αποθήκευση δεδομένων κατά την εκτέλεση των υπολογισμών.
- Καταχωρητής r7 (Program Counter): Διαδραματίζει τον ρόλο του μετρητή προγράμματος. Περιέχει τη διεύθυνση της επόμενης εντολής προς εκτέλεση και αυξάνεται αυτόματα (auto-increment) κατά τη φάση της ανάκτησης (Instruction Fetch).



Εικόνα 2.2: Αναλυτικό μπλοκ διάγραμμα του διαύλου δεδομένων (Datapath) με ενσωμάτωση του r7 ως Program Counter.

2.3 Κεντρικός Δίαυλος (Bus) και Πολυπλέκτης

Η μεταφορά δεδομένων στο εσωτερικό του επεξεργαστή γίνεται μέσω ενός πολυπλέκτη (Multiplexer), ο οποίος ελέγχεται από το σήμα Sel. Ο πολυπλέκτης επιτρέπει την επιλογή μίας πηγής δεδομένων κάθε φορά για την οδήγηση του διαύλου BusWires. Οι διαθέσιμες πηγές περιλαμβάνουν:

1. Τους καταχωρητές r0-r7.
2. Την είσοδο δεδομένων από την εξωτερική μνήμη (DIN).
3. Το αποτέλεσμα της ALU που είναι αποθηκευμένο στον καταχωρητή G.

	T_0	T_1	T_2	T_3	T_4	T_5
mv	*Select pc*, $ADDR_{in}$, *pc_incr*		IR_{in}	*Select rY* or *IR*, rX_{in}, *Done*		
mvt	*Select pc*, $ADDR_{in}$, *pc_incr*		IR_{in}	*Select IR*, rX_{in}, *Done*		
add	*Select pc*, $ADDR_{in}$, *pc_incr*		IR_{in}	*Select rX*, A_{in}	*Select rY* or *IR*, G_{in}	*Select G*, rX_{in}, *Done*
sub	*Select pc*, $ADDR_{in}$, *pc_incr*		IR_{in}	*Select rX*, A_{in}	*Select rY* or *IR*, *AddSub*, G_{in}	*Select G*, rX_{in}, *Done*
and	*Select pc*, $ADDR_{in}$, *pc_incr*		IR_{in}	*Select rX*, A_{in}	*Select rY* or *IR*, *ALU_and*, G_{in}	*Select G*, rX_{in}, *Done*
ld	*Select pc*, $ADDR_{in}$, *pc_incr*		IR_{in}	*Select rY*, $ADDR_{in}$		*Select DIN*, rX_{in}, *Done*
st	*Select pc*, $ADDR_{in}$, *pc_incr*		IR_{in}	*Select rY*, $ADDR_{in}$	*Select rX*, $DOUT_{in}$, W_D, *Done*	

Πίνακας 2.3: Χαρτογράφηση των σημάτων ελέγχου της FSM ανά χρονικό βήμα (T-step) για κάθε εντολή.

2.4 Αριθμητική και Λογική Μονάδα (ALU)

Η ALU είναι υπεύθυνη για την εκτέλεση των πράξεων. Στη συγκεκριμένη υλοποίηση, η ALU υποστηρίζει δύο βασικές λειτουργίες:

- Πρόσθεση (ADD): Το περιεχόμενο του καταχωρητή A προστίθεται με την τιμή που βρίσκεται στον δίαυλο.
- Λογικό ΚΑΙ (AND): Πραγματοποιείται λογική πράξη AND μεταξύ του καταχωρητή A και του διαύλου. Το αποτέλεσμα αποθηκεύεται προσωρινά στον καταχωρητή G πριν μεταφερθεί στον τελικό καταχωρητή προορισμού.

2.5 Μονάδα Ελέγχου και Μηχανή Καταστάσεων (FSM)

Η Μονάδα Ελέγχου (Control Unit) αποτελεί τον «εγκέφαλο» του επεξεργαστή, καθώς είναι υπεύθυνη για τον συγχρονισμό όλων των υπομονάδων (Register File, ALU, Μνήμη) και την καθοδήγηση της ροής των δεδομένων μέσω του κεντρικού διαύλου (BusWire). Η λειτουργία της βασίζεται σε μια Μηχανή Πεπερασμένων Καταστάσεων (Finite State Machine - FSM), η οποία αλλάζει κατάσταση σε κάθε θετική ακμή του ρολογιού (clock), παράγοντας τα κατάλληλα σήματα ελέγχου ανάλογα με την εντολή που εκτελείται.

Η εκτέλεση κάθε εντολής χωρίζεται σε διακριτά χρονικά βήματα (Time-steps), τα οποία ονομάζονται καταστάσεις T_0, T_1, T_2, T_3. Ακολουθεί η αναλυτική λειτουργία των μηχανισμών της FSM ανά χρονικό βήμα:

• Κατάσταση T_0 (Instruction Fetch - Ανάκτηση Εντολής): Η κατάσταση αυτή είναι κοινή για όλες τις εντολές του επεξεργαστή. Σε αυτό το βήμα, η Μονάδα Ελέγχου ενεργοποιεί το σήμα Sel_PC (ή Sel_R7 αν ο Program Counter είναι ο καταχωρητής R7), ώστε η διεύθυνση της επόμενης εντολής να διοχετευθεί στον δίαυλο BusWire. Ταυτόχρονα, ενεργοποιείται το σήμα ADDRin της μνήμης, ώστε η διεύθυνση αυτή να μανδαλωθεί στον καταχωρητή διευθύνσεων μνήμης.

• Κατάσταση T_1 (Instruction Decode - Αποκωδικοποίηση & Ανανέωση PC): Η μνήμη επιστρέφει τα δεδομένα της εντολής στον δίαυλο. Η FSM ενεργοποιεί το σήμα IRin (Instruction Register In), ώστε η εντολή 16-bit να αποθηκευτεί στον Καταχωρητή Εντολής για να αποκωδικοποιηθεί. Παράλληλα, ο Program Counter (PC) αυξάνεται κατά ένα (PC = PC + 1) ώστε να δείχνει στην επόμενη εντολή, και η νέα τιμή αποθηκεύεται πίσω στον PC.

• Κατάσταση T_2 (Execution - Φάση Εκτέλεσης 1): Σε αυτό το βήμα, η FSM διαφοροποιείται ανάλογα με το opcode της εντολής:

○ Στις εντολές add / sub, το περιεχόμενο του πρώτου καταχωρητή (Rx) διοχετεύεται στο BusWire και μεταφέρεται στον εσωτερικό καταχωρητή A της ALU (Ain = 1).

○ Στην εντολή mvi (Move Immediate), η FSM διαβάζει την άμεση τιμή (immediate value) από την εντολή, τη διοχετεύει στο BusWire και ενεργοποιεί το σήμα εισόδου του καταχωρητή προορισμού (Rx_in = 1), ολοκληρώνοντας την εντολή.

○ Στις εντολές ld (Load) / st (Store), η FSM υπολογίζει τη διεύθυνση μνήμης και την προωθεί στον καταχωρητή διευθύνσεων.

• Κατάσταση T_3 (Execution - Φάση Εκτέλεσης 2):

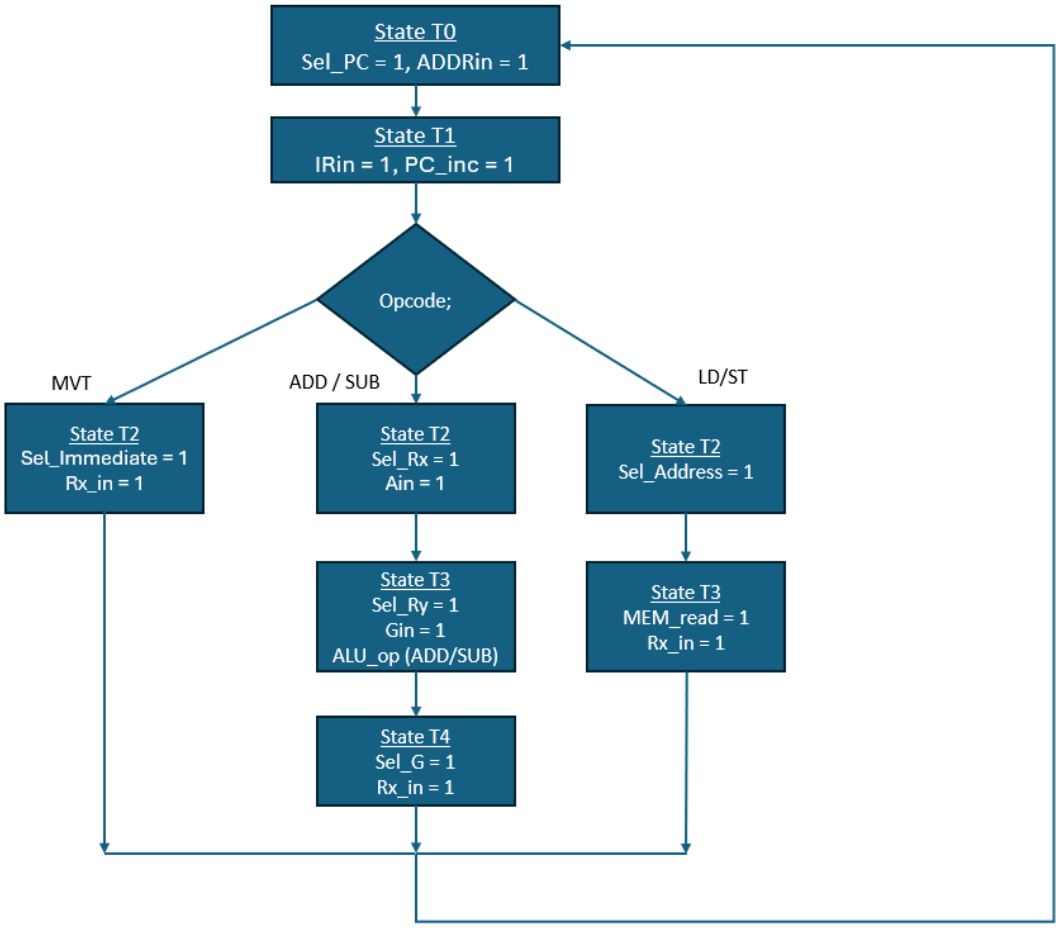
○ Στις εντολές add / sub, το περιεχόμενο του δεύτερου καταχωρητή (Ry) βγαίνει στο BusWire. Η ALU εκτελεί την πράξη με το περιεχόμενο του καταχωρητή A και το αποτέλεσμα περνάει στον καταχωρητή G (Gin = 1). Στον επόμενο παλμό, η τιμή του G μεταφέρεται στον τελικό καταχωρητή Rx.

○ Στην εντολή ld, ενεργοποιείται το σήμα ανάγνωσης της μνήμης (MEM_read) και τα δεδομένα από τη RAM γράφονται στον καταχωρητή Rx.

○ Στην εντολή st, το περιεχόμενο του Rx βγαίνει στο BusWire και ενεργοποιείται το σήμα εγγραφής της μνήμης (MEM_write), αποθηκεύοντας την τιμή στη RAM.

Αλγοριθμικό Διάγραμμα Καταστάσεων (ASM Chart)

Για την πληρέστερη κατανόηση και τον ακριβή σχεδιασμό της Μονάδας Ελέγχου, η FSM αποτυπώνεται μέσω ενός Αλγοριθμικού Διαγράμματος Καταστάσεων (Algorithmic State Machine - ASM). Το διάγραμμα ASM προσφέρει μια αυστηρή ψηφιακή αναπαράσταση που συνδυάζει τη ροή των καταστάσεων με τις συνθήκες των εισόδων (αποκωδικοποίηση εντολών) και τις παραγόμενες εξόδους (σήματα ελέγχου), διευκολύνοντας τη μεταφορά του κυκλώματος σε κώδικα VHDL.



Εικόνα 2.3: Αλγοριθμικό διάγραμμα καταστάσεων (ASM Chart) της Μονάδας Ελέγχου του επεξεργαστή.

Όπως φαίνεται στο παραπάνω διάγραμμα ASM (Εικόνα 2.3), η FSM ξεκινά πάντα από τις κοινές καταστάσεις T_0 και T_1. Στην κατάσταση T_2, η FSM διακλαδώνεται (μέσω των decision boxes) ανάλογα με το Instruction Register (IR). Κάθε κλάδος ενεργοποιεί μόνο τα απαραίτητα σήματα ελέγχου (State Outputs και Conditional Outputs) που απαιτούνται για τη συγκεκριμένη εντολή, διασφαλίζοντας ότι δεν θα υπάρξει διένεξη δεδομένων (bus contention) στον κεντρικό δίαυλο.

ΚΕΦΑΛΑΙΟ 3: ΣΧΕΔΙΑΣΗ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΒΕΛΤΙΩΜΕΝΟΥ ΕΠΕΞΕΡΓΑΣΤΗ (PART III) [7]

3.1 Ενσωμάτωση Μνήμης και Περιφερειακών

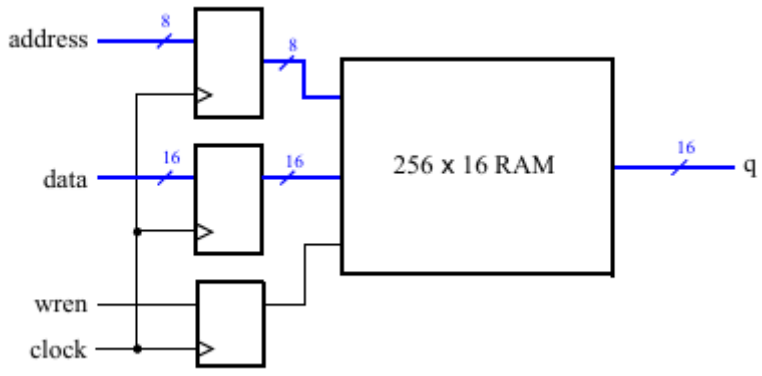
Στο στάδιο αυτό, ο επεξεργαστής παύει να λειτουργεί ως μεμονωμένη μονάδα και εντάσσεται σε ένα ολοκληρωμένο υπολογιστικό σύστημα. Η κύρια πρόκληση στην υλοποίηση του Μέρους III (Part III) είναι η διασύνδεση του επεξεργαστή με μια σύγχρονη μνήμη RAM και η δυνατότητα επικοινωνίας με θύρες εισόδου/εξόδου (I/O).

3.2 Σχεδίαση της Μονάδας Μνήμης (inst_mem)

Η μνήμη που υλοποιήθηκε (αρχείο inst_mem.vhd) είναι μια σύγχρονη μνήμη (Synchronous RAM) χωρητικότητας 32 λέξεων των 16-bit.

- Λειτουργία Ανάγνωσης/Εγγραφής: Η μνήμη χρησιμοποιεί το σήμα wren (Write Enable) για να καθορίσει αν θα εκτελέσει εγγραφή των δεδομένων data στη διεύθυνση address. Η έξοδος q επιστρέφει το περιεχόμενο της διεύθυνσης στην επόμενη ακμή του ρολογιού.

- Αρχικοποίηση: Για την εκτέλεση προγραμμάτων, η μνήμη αρχικοποιείται μέσω ενός αρχείου .mif (Memory Initialization File). Η οδηγία ram_init_file στον κώδικα διασφαλίζει ότι το Machine Code του προγράμματος φορτώνεται στη μνήμη κατά το compile στο Quartus.

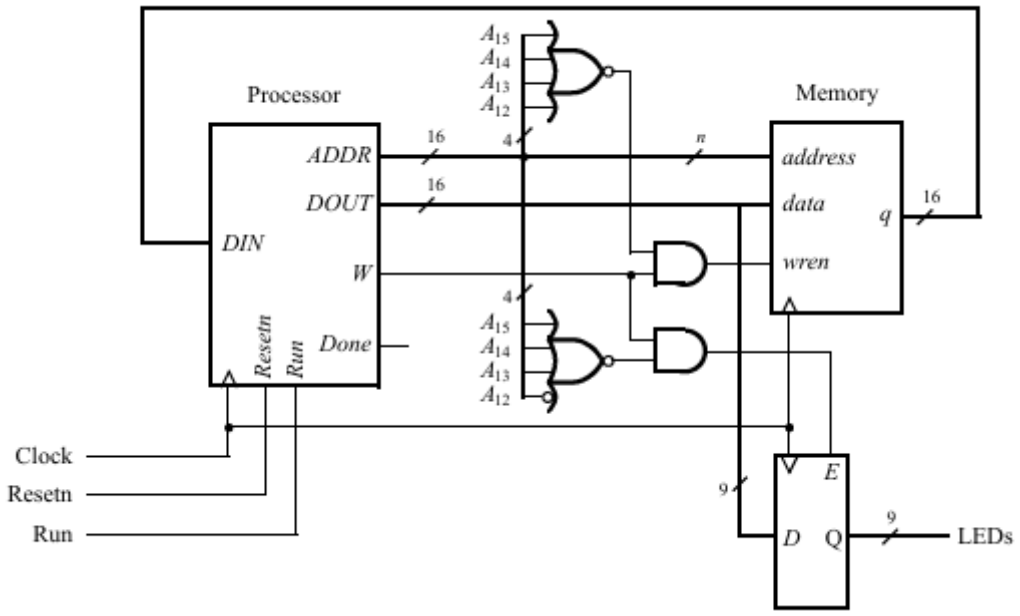


Εικόνα 3.1: Μπλοκ διάγραμμα της σύγχρονης μνήμης SRAM (256 x 16).

3.3 Διασύνδεση (Entity Mapping)

Η οντότητα κορυφής part3.vhd αναλαμβάνει τη σύνδεση των δύο υπομονάδων. Η διασύνδεση βασίζεται στα εξής σήματα:

- ADDR (4 downto 0): Ο επεξεργαστής παράγει μια διεύθυνση 16-bit, αλλά για τις ανάγκες της συγκεκριμένης μνήμης χρησιμοποιούνται μόνο τα 5 λιγότερο σημαντικά ψηφία (LSBs) για τη διευθυνσιοδότηση των 32 θέσεων.
- DIN / DOUT: Ο δίαυλος DIN του επεξεργαστή συνδέεται με την έξοδο q της μνήμης, ενώ ο δίαυλος DOUT συνδέεται με την είσοδο δεδομένων της μνήμης.
- W (Write): Το σήμα ελέγχου εγγραφής του επεξεργαστή συνδέεται απευθείας στο wren της μνήμης.



Εικόνα 3.2: Συνολικό διάγραμμα διασύνδεσης του επεξεργαστή με τη μνήμη και τα περιφερειακά (Top-Level Circuit).

3.4 Αποκωδικοποίηση Διευθύνσεων και Έξοδος LED

Στον κώδικα της οντότητας part3, υλοποιήθηκε μια απλή μορφή απεικόνισης δεδομένων. Τα 10 λιγότερο σημαντικά bits του διαύλου δεδομένων εξόδου (DOUT) οδηγούνται απευθείας στα LED της πλακέτας FPGA (LEDR(9 downto 0)). Αυτό επιτρέπει στον χρήστη να παρακολουθεί σε πραγματικό χρόνο τα αποτελέσματα των εντολών st (Store) που εκτελεί ο επεξεργαστής.

3.5 Συγχρονισμός και Χρονισμός

Λόγω της σύγχρονης φύσης της μνήμης, απαιτείται προσεκτικός χρονισμός. Όπως φαίνεται στη Μηχανή Καταστάσεων του επεξεργαστή, εισάγεται ένας κύκλος αναμονής (Wait State - T1/T4) ώστε να διασφαλιστεί ότι τα δεδομένα από τη μνήμη έχουν σταθεροποιηθεί στον δίαυλο DIN πριν αυτά αναγνωστούν από τον επεξεργαστή.

4. ΠΡΟΣΟΜΟΙΩΣΗ ΚΑΙ ΕΠΑΛΗΘΕΥΣΗ

4.1 Περιβάλλον και Μεθοδολογία Προσομοίωσης

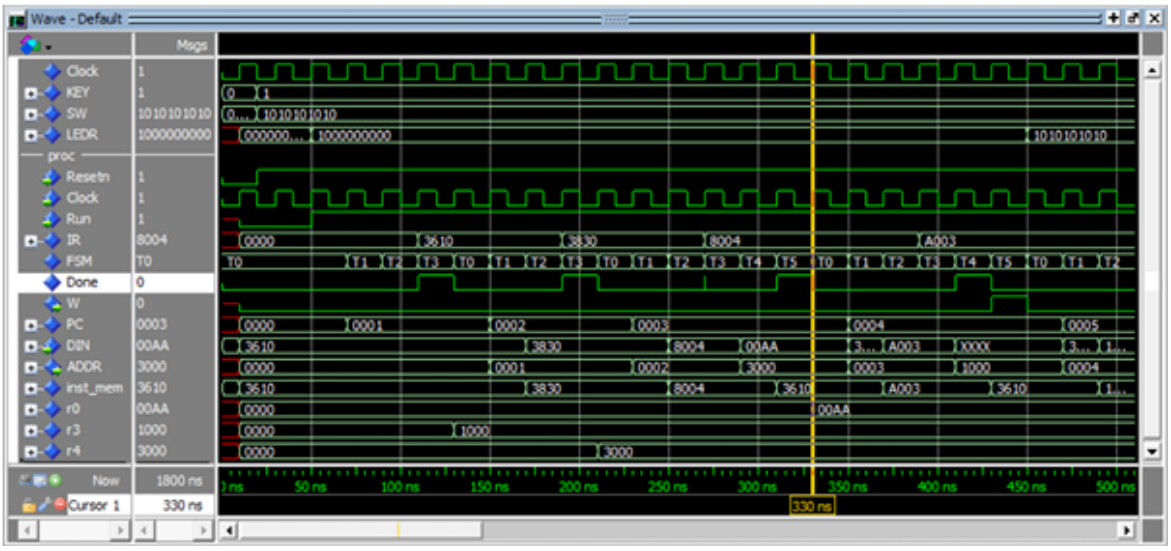
Η επαλήθευση της ορθής λειτουργίας του επεξεργαστή πραγματοποιήθηκε στο περιβάλλον ModelSim. Για τον έλεγχο του σχεδιασμού, ακολουθήθηκε η μέθοδος της διαδραστικής προσομοίωσης (interactive simulation). Αντί για τη συγγραφή ενός στατικού αρχείου testbench σε VHDL, χρησιμοποιήθηκαν σήματα διέγερσης (stimuli) απευθείας μέσω του κελύφους εντολών του ModelSim.

Συγκεκριμένα, εφαρμόστηκαν οι εξής ενέργειες:

- Αρχικοποίηση: Χρησιμοποιήθηκαν εντολές force για την απόδοση αρχικών τιμών στα σήματα εισόδου. Το σήμα Clock ρυθμίστηκε να παράγει παλμοσειρά συγκεκριμένης περιόδου, ενώ το σήμα Resetn ενεργοποιήθηκε αρχικά για την επαναφορά του συστήματος στην κατάσταση ηρεμίας.

- Φόρτωση Προγράμματος: Η μνήμη inst_mem φορτώθηκε αυτόματα με το περιεχόμενο του αρχείου inst_mem.mif, το οποίο περιλαμβάνει τις εντολές προς εκτέλεση.

- Εκτέλεση: Με την εντολή run, ο επεξεργαστής ξεκίνησε την εκτέλεση του προγράμματος, επιτρέποντας την παρακολούθηση της εξέλιξης των σημάτων σε πραγματικό χρόνο.



Εικόνα 4.1: Κυματομορφές λειτουργικής προσομοίωσης του επεξεργαστή στο ModelSim κατά την εκτέλεση προγράμματος.

4.2 ΠΡΟΓΡΑΜΜΑ ΕΠΑΛΗΘΕΥΣΗΣ ΚΑΙ ΔΙΑΣΥΝΔΕΣΗ ΕΙΣΟΔΟΥ/ΕΞΟΔΟΥ (I/O)

Για την ολοκληρωμένη αξιολόγηση των δυνατοτήτων του επεξεργαστή, κρίθηκε απαραίτητο να ελεγχθεί όχι μόνο η εσωτερική ροή δεδομένων, αλλά και η ικανότητά του να επικοινωνεί με περιφερειακές συσκευές εισόδου και εξόδου (I/O). Για τον σκοπό αυτό, χρησιμοποιήθηκε το επίσημο πρόγραμμα ελέγχου από το εργαστήριο (Laboratory Exercise 10, Figure 16), το οποίο στηρίζεται στις εντολές προσπελάσης μνήμης.

1. Δημιουργία και Σκοπός του Προγράμματος (Τι κάνει)

Το πρόγραμμα **δημιουργήθηκε** με στόχο τη συνεχή ανάγνωση της κατάστασης εξωτερικών διακοπτών (Slide Switches) και την άμεση προβολή της τιμής τους στα LED της πλακέτας. Ο κώδικας σε γλώσσα Assembly περιλαμβάνει τις εξής εντολές:

```
.define LED_ADDRESS 0x10
.define SW_ADDRESS 0x30

// Read SW switches and display on LEDs
        mvt    r3, #LED_ADDRESS   // point to LED port
        mvt    r4, #SW_ADDRESS    // point to SW port
MAIN:   ld     r0, [r4]           // read SW values
        st     r0, [r3]           // light up LEDs
        mv     pc, #MAIN
```

Εικόνα 4.2 Αντιστοίχιση εντολών Assembly (I/O Program) και κώδικα μηχανής 16-bit για την επαλήθευση του ενισχυμένου επεξεργαστή.

Τι κάνει το πρόγραμμα: Υλοποιεί έναν ατέρμονα βρόχο (infinite loop). Σε κάθε επανάληψη, απομονώνει μέσω της εντολής ld τα δεδομένα εισόδου από τη διεύθυνση των διακοπτών (0x30) και τα αποθηκεύει προσωρινά στον καταχωρητή $r0$. Αμέσως μετά, μέσω της εντολής st, στέλνει τα δεδομένα αυτά στη διεύθυνση εξόδου των LED (0x10). Το αποτέλεσμα είναι ότι οποιοσδήποτε διακόπτης ενεργοποιείται από τον χρήστη, ανάβει ακαριαία το αντίστοιχο LED.

2. Διαδικασία Φόρτωσης στη Μνήμη (Πώς φορτώνεται)

Επειδή η αρχιτεκτονική αναγνωρίζει μόνο ψηφιακές λέξεις, οι εντολές Assembly μεταφράζονται σε 16-bit κώδικα μηχανής (Machine Code). Το πρόγραμμα **φορτώνεται** στον επεξεργαστή μέσω του αρχείου αρχικοποίησης μνήμης .mif (inst_mem.mif) στο περιβάλλον του Intel Quartus Prime.

Κατά τη διαδικασία της Λογικής Σύνθεσης (Synthesis), το Quartus ενσωματώνει αυτό το αρχείο στα εσωτερικά blocks μνήμης (Embedded Memory Blocks) του FPGA που έχουν δεσμευτεί για τη ROM. Έτσι, με την εκκίνηση του συστήματος, το πρόγραμμα βρίσκεται ήδη προφορτωμένο στη μνήμη εντολών.

3. Τρόπος Εκτέλεσης και Επαλήθευσης (Πώς εκτελείται & Πώς επαληθεύεται)

- **Πώς εκτελείται:** Η **εκτέλεση** καθοδηγείται από τη Μονάδα Ελέγχου (FSM). Μετά το Reset, ο Program Counter ($pc / r7$) δείχνει στη διεύθυνση 00. Σε κάθε βήμα T_0 και T_1 γίνεται η ανάκτηση και αποκωδικοποίηση της εντολής στον καταχωρητή IR. Στα βήματα T_2-T_3 των εντολών ld και st, η FSM ενεργοποιεί τα σήματα MEM_read και MEM_write αντίστοιχα, ανοίγοντας τις πύλες του κεντρικού διαύλου (BusWires) για τη μεταφορά των δεδομένων.

- **Πώς επαληθεύεται:** Η εγκυρότητα του προγράμματος **επαληθεύεται** πλήρως στα επόμενα υποκεφάλαια. Στο υποκεφάλαιο 4.4 (Κυματομορφές ModelSim), αποδεικνύεται ψηφιακά ότι οι καταχωρητές $r3, r4, r0$ δέχονται τις σωστές τιμές στους προβλεπόμενους κύκλους ρολογιού. Αντίστοιχα, στο υποκεφάλαιο 4.5 (Εξομοιωτής DeSim), η ορθότητα επιβεβαιώνεται οπτικά στο hardware, καθώς η αλλαγή των διακοπτών μεταβάλλει σε πραγματικό χρόνο την κατάσταση των LED στην οθόνη.

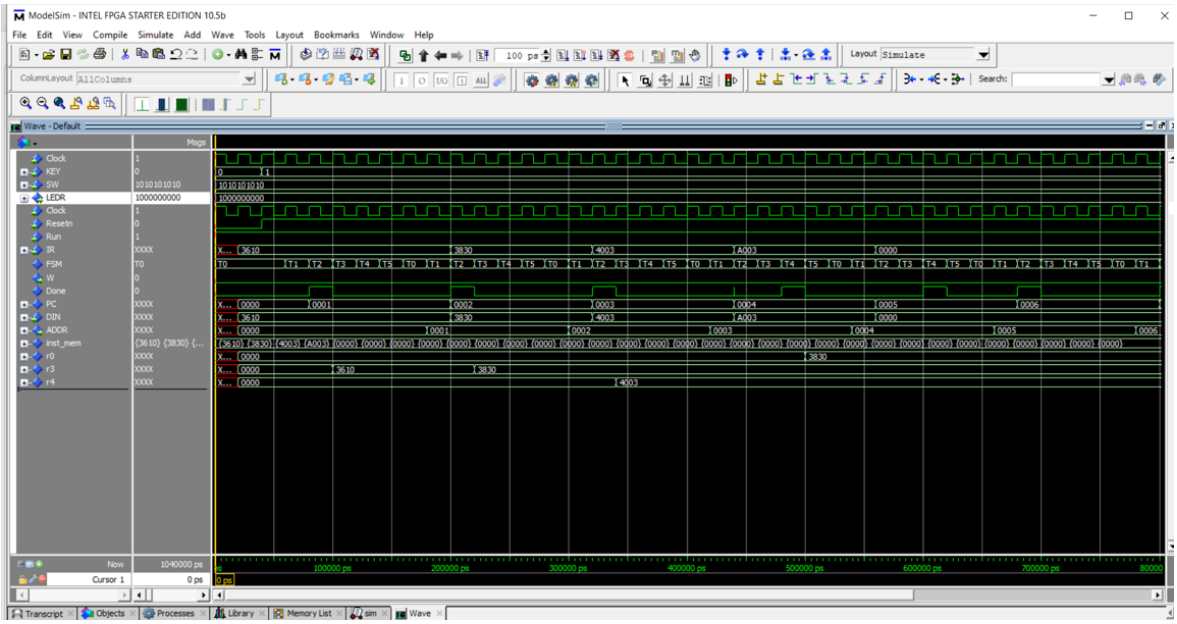
4.3 Ανάλυση της Διαδικασίας Ανάκτησης (Instruction Fetch)

Μέσω των κυματομορφών, επιβεβαιώθηκε ότι η διαδικασία ανάκτησης εντολής ακολουθεί πιστά τα χρονικά βήματα (T-steps) που ορίστηκαν στη Μηχανή Καταστάσεων:

- Βήμα T0: Το σήμα Sel παίρνει την τιμή "1111", οδηγώντας την τιμή του r7 (Program Counter) στον εσωτερικό δίαυλο, ενώ το σήμα ADDRin ενεργοποιείται για να οριστεί η διεύθυνση στη μνήμη.

- Βήμα T1: Το σύστημα εισέρχεται σε κατάσταση αναμονής (Wait state). Αυτό είναι απαραίτητο καθώς η μνήμη RAM είναι σύγχρονη και απαιτεί έναν κύκλο ρολογιού για να παρουσιάσει τα δεδομένα στην έξοδό της.

- Βήμα T2: Το σήμα IRin ενεργοποιείται, "κλειδώνοντας" την εντολή από τη μνήμη στον καταχωρητή εντολών (IR), ενώ ταυτόχρονα το σήμα pc_inc αυξάνει την τιμή του r7.



Εικόνα 4.3 Πραγματικές κυματομορφές λειτουργικής προσομοίωσης στο ModelSim από την υλοποίηση του κώδικα VHDL..

4.4 Επαλήθευση Λειτουργίας Εντολών

Κατά την προσομοίωση εξετάστηκαν αντιπροσωπευτικές εντολές για την επιβεβαίωση της αρχιτεκτονικής:

1. Εντολές Μεταφοράς (mov, ld): Επιβεβαιώθηκε ότι τα δεδομένα μεταφέρονται σωστά μεταξύ των καταχωρητών και από τη μνήμη προς τον επεξεργαστή, με τη σωστή ενεργοποίηση των σημάτων Rin και Rout.

2. Αριθμητικές Πράξεις: Η ALU εκτελεί τις πράξεις (ADD, AND) στο βήμα T4 και αποθηκεύει το αποτέλεσμα στον καταχωρητή G, το οποίο στη συνέχεια μεταφέρεται στον τελικό προορισμό στο βήμα T5.

3. Έλεγχος Περιφερειακών: Επαληθεύτηκε ότι οι εντολές αποθήκευσης (st) οδηγούν τα δεδομένα στον δίαυλο DOUT, ο οποίος με τη σειρά του ενεργοποιεί τα LED της πλακέτας FPGA μέσω του address decoder.

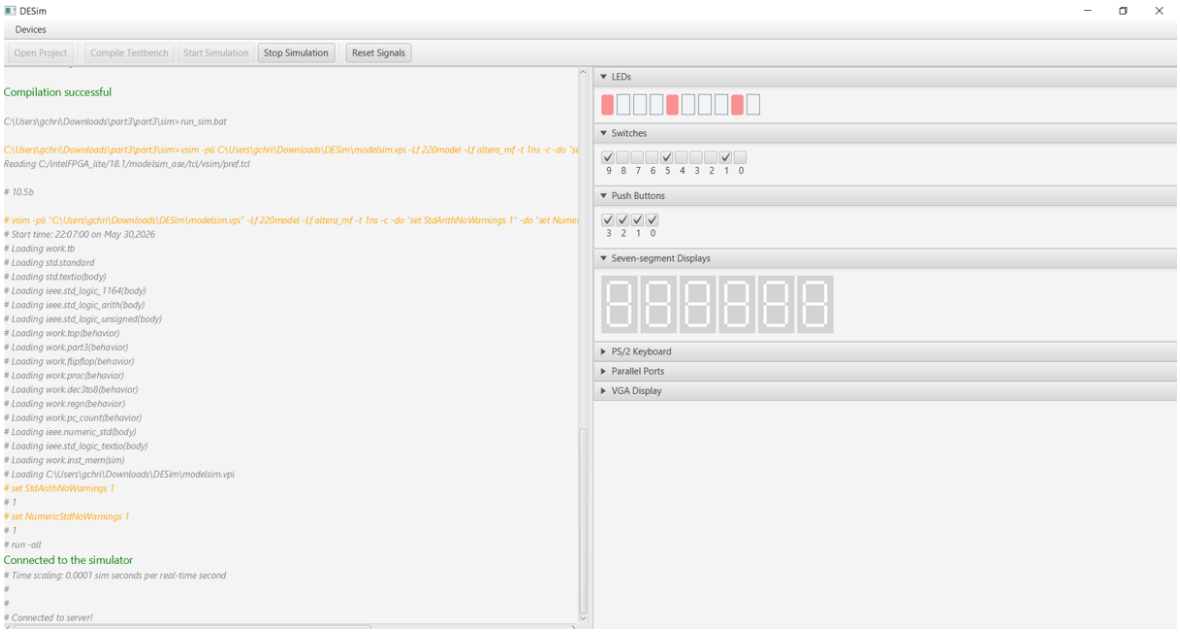
4.5 Εξομοίωση Υλικού σε Περιβάλλον DeSim

Για την ολοκληρωμένη επαλήθευση του αναπτυχθέντος επεξεργαστή σε συνθήκες πραγματικού υλικού, πέρα από τις κυματομορφές σημάτων, χρησιμοποιήθηκε ο εξομοιωτής DeSim. Το συγκεκριμένο εργαλείο επιτρέπει τη δυναμική οπτικοποίηση και τον έλεγχο της λειτουργίας του κυκλώματος VHDL, προσομοιώνοντας με ακρίβεια τα περιφερειακά στοιχεία και τις αποκρίσεις μιας αναπτυξιακής πλακέτας FPGA (Altera/Intel DE-series).

Κατά τη διαδικασία αυτή, ο κώδικας VHDL συνδέθηκε εικονικά με τους διακόπτες εισόδου (switches) και τις ενδεικτικές λυχνίες εξόδου (LEDs) της πλακέτας. Ιδιαίτερη έμφαση δόθηκε στην επιβεβαίωση των εντολών μεταφοράς δεδομένων στη μνήμη. Όπως αναλύεται στη δομή του συστήματος, τα 10 λιγότερο σημαντικά bits (LSBs) του διαύλου δεδομένων εξόδου DOUT έχουν συνδεθεί απευθείας με τις λυχνίες LEDR(9 downto 0).

Κατά την εκτέλεση της εντολής εγγραφής στη μνήμη (st - Store), όταν το σήμα ελέγχου εγγραφής wren (ή W) ενεργοποιείται, τα δεδομένα που προορίζονται για αποθήκευση

εμφανίζονται ταυτόχρονα και στον δίαυλο εξόδου. Η ορθή λειτουργία του επεξεργαστή επιβεβαιώθηκε οπτικά στο περιβάλλον του DeSim, καθώς οι λυχνίες LED άναψαν σε δυαδικό συνδυασμό που αντιστοιχούσε επακριβώς στην τιμή των δεδομένων που εγγράφηκαν. Η επιτυχής αυτή δοκιμή αποδεικνύει τον σωστό χρονισμό και τη σταθερότητα του επεξεργαστή κατά την αλληλεπίδρασή του με το hardware κομμάτι του συστήματος.



Εικόνα 4.4 Οπτική επιβεβαίωση της λειτουργίας του επεξεργαστή και απεικόνιση των δεδομένων εξόδου στα LEDs της πλακέτας στο περιβάλλον DeSim.

4.6 Συμπεράσματα Προσομοίωσης

Η interactive μέθοδος επέτρεψε την άμεση επέμβαση στα σήματα και την ταχεία ανίχνευση τυχόν σφαλμάτων στη λογική της FSM. Οι κυματομορφές που παρήχθησαν αποδεικνύουν ότι ο επεξεργαστής λειτουργεί με απόλυτο συγχρονισμό, χωρίς συγκρούσεις στον δίαυλο δεδομένων, και είναι έτοιμος για την τελική σύνθεση και μεταφόρτωση στο υλικό.

5. ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

5.1 Ανασκόπηση Εργασίας

Στην παρούσα πτυχιακή εργασία ολοκληρώθηκε με επιτυχία ο σχεδιασμός και η υλοποίηση ενός βελτιωμένου επεξεργαστή 16-bit σε περιβάλλον VHDL. Ξεκινώντας από τις βασικές αρχές ενός απλού επεξεργαστή, προχωρήσαμε στην ενσωμάτωση κρίσιμων λειτουργιών όπως ο Program Counter (PC) και η διασύνδεση με εξωτερική μνήμη RAM. Η υλοποίηση του Μέρους III (Part III) επέτρεψε στο σύστημα να λειτουργεί αυτόνομα, εκτελώντας προγράμματα από τη μνήμη και αλληλεπιδρώντας με το χρήστη μέσω διακοπτών και LED.

5.2 Συμπεράσματα

Από τη διαδικασία σχεδίασης και τις δοκιμές που πραγματοποιήθηκαν, προκύπτουν τα εξής συμπεράσματα:

1. **Αποτελεσματικότητα VHDL:** Η χρήση της VHDL αποδείχθηκε εξαιρετικά ισχυρή για την περιγραφή σύνθετων ψηφιακών συστημάτων, επιτρέποντας τον έλεγχο κάθε σήματος σε επίπεδο κύκλου ρολογιού.

2. **Σημασία της FSM:** Η σωστή οργάνωση της Μηχανής Πεπερασμένων Καταστάσεων είναι το "κλειδί" για την αποφυγή συγκρούσεων στο δίαυλο (bus contention) και την ορθή εκτέλεση των εντολών.

3. **Ευελιξία FPGA:** Η υλοποίηση σε FPGA προσφέρει τη δυνατότητα άμεσης επαλήθευσης του σχεδιασμού σε πραγματικό χρόνο, γεφυρώνοντας το χάσμα μεταξύ θεωρίας και πράξης.

5.3 Δυσκολίες και Προκλήσεις

Κατά τη διάρκεια της υλοποίησης, οι κυριότερες προκλήσεις αφορούσαν:

- Τον ακριβή συγχρονισμό της ανάγνωσης από τη μνήμη SSRAM, η οποία απαιτεί τα δεδομένα να είναι έτοιμα πριν από την ακμή του ρολογιού.
- Την ορθή υλοποίηση του address decoder ώστε να μην υπάρχουν επικαλύψεις στις διευθύνσεις της μνήμης και των περιφερειακών.

5.4 Προτάσεις για Μελλοντικές Επεκτάσεις

Παρά την πλήρη λειτουργικότητα του επεξεργαστή, υπάρχουν περιθώρια για περαιτέρω βελτιώσεις, όπως:

- **Pipelining:** Η εισαγωγή τεχνικής διοχέτευσης για την παράλληλη εκτέλεση διαφορετικών σταδίων εντολών, αυξάνοντας την ταχύτητα του επεξεργαστή.
- **Διακοπές (Interrupts):** Η προσθήκη συστήματος διακοπών για την άμεση ανταπόκριση του επεξεργαστή σε εξωτερικά γεγονότα.
- **Επέκταση Instruction Set:** Η προσθήκη εντολών πολλαπλασιασμού ή υποστήριξης κινητής υποδιαστολής (floating point).

ΠΑΡΑΡΤΗΜΑ

Κώδικας VHDL του Συστήματος

Α.1 ΜΟΝΑΔΑ ΕΠΕΞΕΡΓΑΣΤΗ (proc.vhd)

```
-- ΑΡΧΕΙΟ: proc.vhd

-- ΠΕΡΙΓΡΑΦΗ: Η κεντρική μονάδα επεξεργασίας (CPU / Επεξεργαστής).

--            Περιλαμβάνει τη Μονάδα Ελέγχου (FSM 6 καταστάσεων T0-
T5),

--            την Αριθμητική και Λογική Μονάδα (ALU), το σετ
καταχωρητών

--            και τον εσωτερικό δίαυλο δεδομένων (Bus Wires).


library ieee;

USE ieee.std_logic_1164.all;

USE ieee.std_logic_unsigned.all;


-- Ορισμός της οντότητας του επεξεργαστή

ENTITY proc IS

    PORT (  DIN                 : IN  STD_LOGIC_VECTOR(15 DOWNTO 0); -
- Είσοδος δεδομένων από τη μνήμη

            Resetn, Clock, Run  : IN  STD_LOGIC;                     -
- Σήματα ελέγχου (Reset, Ρολόι, Εκκίνηση)

            DOUT                : OUT STD_LOGIC_VECTOR(15 DOWNTO 0); -
- Έξοδος δεδομένων προς τη μνήμη

            ADDR                : OUT STD_LOGIC_VECTOR(15 DOWNTO 0); -
- Διεύθυνση μνήμης
```

```
            W                    : OUT STD_LOGIC);                    -
- Σήμα ελέγχου εγγραφής στη μνήμη (Write Enable)

END proc;


ARCHITECTURE Behavior OF proc IS

    -- Δήλωση Component: Μετρητής Προγράμματος (Program Counter)

    COMPONENT pc_count

        PORT (  R    : IN   STD_LOGIC_VECTOR(15 DOWNTO 0);

                Resetn, Clock, E, L   : IN   STD_LOGIC;

                Q    : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));

    END COMPONENT;


    -- Δήλωση Component: Αποκωδικοποιητής 3 σε 8 (για την ενεργοποίηση
των καταχωρητών)

    COMPONENT dec3to8

        PORT ( E   : IN   STD_LOGIC;

               W   : IN   STD_LOGIC_VECTOR(2 DOWNTO 0);

               Y   : OUT  STD_LOGIC_VECTOR(0 TO 7));

    END COMPONENT;


    -- Δήλωση Component: Παραμετρικός Καταχωρητής n-bit

    COMPONENT regn

        GENERIC ( n : INTEGER := 16);

        PORT ( R                   : IN STD_LOGIC_VECTOR(n-1 DOWNTO
0);
```

```
            Resetn, E, Clock    : IN STD_LOGIC;
            Q                   : OUT STD_LOGIC_VECTOR(n-1 DOWNTO
0));
    END COMPONENT;

    -- Δήλωση Component: D-FlipFlop (για το σήμα Write)
    COMPONENT flipflop
        PORT (  D, Resetn, Clock  : IN  STD_LOGIC;
                Q                 : OUT STD_LOGIC);
    END COMPONENT;

    -- Ορισμός των καταστάσεων της FSM (Μονάδας Ελέγχου)
    TYPE State_type IS (T0, T1, T2, T3, T4, T5);
    SIGNAL Tstep_Q, Tstep_D: State_type; -- Τρέχουσα και επόμενη
κατάσταση

    -- Εσωτερικά σήματα ελέγχου και δεδομένων
    SIGNAL Sel : STD_LOGIC_VECTOR(3 DOWNTO 0);      -- Επιλογέας του
πολυπλέκτη του Bus
    SIGNAL Rin : STD_LOGIC_VECTOR(0 TO 7);          -- Σήματα
ενεργοποίησης καταχωρητών r0 έως r7
    SIGNAL Sum : STD_LOGIC_VECTOR(15 DOWNTO 0);     -- Έξοδος της ALU
    SIGNAL rXin, IRin, Done, ADDRin, DOUTin, Imm, Ain, Gin, AddSub,
ALUand : STD_LOGIC;
    SIGNAL III, rX, rY : STD_LOGIC_VECTOR(2 DOWNTO 0); -- Πεδία της
εντολής (Opcode, RegX, RegY)
    SIGNAL Xreg : STD_LOGIC_VECTOR(0 TO 7);
```

```
    SIGNAL r0, r1, r2, r3, r4, r5, r6, PC, A : STD_LOGIC_VECTOR(15
DOWNTO 0); -- Γενικοί καταχωρητές και PC

    SIGNAL G : STD_LOGIC_VECTOR(15 DOWNTO 0);       -- Καταχωρητής
εξόδου της ALU

    SIGNAL IR, BusWires : STD_LOGIC_VECTOR(15 DOWNTO 0); --
Καταχωρητής Εντολής και Εσωτερικό Bus

    SIGNAL pc_incr, W_D : STD_LOGIC;


    -- Σταθερές για τους Κωδικούς Λειτουργίας (Opcodes)

    CONSTANT mv : STD_LOGIC_VECTOR(2 DOWNTO 0) := "000";

    CONSTANT mvt : STD_LOGIC_VECTOR(2 DOWNTO 0) := "001";

    CONSTANT add : STD_LOGIC_VECTOR(2 DOWNTO 0) := "010";

    CONSTANT sub : STD_LOGIC_VECTOR(2 DOWNTO 0) := "011";

    CONSTANT ld : STD_LOGIC_VECTOR(2 DOWNTO 0) := "100";

    CONSTANT st : STD_LOGIC_VECTOR(2 DOWNTO 0) := "101";

    CONSTANT and_it : STD_LOGIC_VECTOR(2 DOWNTO 0) := "110";


    -- Σταθερές επιλογής (Selectors) για την οδήγηση δεδομένων στο
εσωτερικό Bus

    CONSTANT SEL_R0 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";

    CONSTANT SEL_R1 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0001";

    CONSTANT SEL_R2 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0010";

    CONSTANT SEL_R3 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0011";

    CONSTANT SEL_R4 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0100";

    CONSTANT SEL_R5 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0101";

    CONSTANT SEL_R6 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0110";
```

```
    CONSTANT SEL_PC : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0111";

    CONSTANT SEL_G : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1000";

    CONSTANT SEL_IR8_IR8_0 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1001"; -
- Άμεση τιμή με επέκταση προσήμου

    CONSTANT SEL_IR7_0_0 : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1010" ;
-- Άμεση τιμή ολισθημένη (Shift Left 8)

    CONSTANT SEL_DIN : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1011" ;   --
Δεδομένα εισόδου από μνήμη



BEGIN

    -- Αποκωδικοποίηση των πεδίων του Καταχωρητή Εντολής (IR)

    III <= IR(15 DOWNTO 13); -- Τα 3 MSB ορίζουν την εντολή (Opcode)

    Imm <= IR(12);           -- Το bit 12 δείχνει αν υπάρχει άμεσο
operand (Immediate)

    rX <= IR(11 DOWNTO 9);   -- Καταχωρητής προορισμού/πρώτου
τελεστέου

    rY <= IR(2 DOWNTO 0);    -- Καταχωρητής πηγής/δεύτερου τελεστέου



    -- Παραγωγή των σημάτων Rin (r0-r7) βάσει του κωδικού rX

    decX: dec3to8 PORT MAP (rXin, rX, Rin);



    -- Διεργασία Μετάβασης Καταστάσεων της FSM (State Table)

    statetable: PROCESS(Tstep_Q, Run, Done)

    BEGIN

        CASE Tstep_Q IS

            WHEN T0 =>    -- Βήμα 0: Ανάκτηση εντολής (Fetch)
```

```
                IF Run = '0' THEN Tstep_D <= T0;
                ELSE Tstep_D <= T1;
                END IF;
            WHEN T1 =>    -- Βήμα 1: Κύκλος αναμονής σύγχρονης μνήμης
                Tstep_D <= T2;
            WHEN T2 =>    -- Βήμα 2: Αποθήκευση της εντολής στον IR
                Tstep_D <= T3;
            WHEN T3 =>    -- Βήμα 3: Εκτέλεση 1ου σταδίου εντολής
                IF Done = '1' THEN Tstep_D <= T0;
                ELSE Tstep_D <= T4;
                END IF;
            WHEN T4 =>    -- Βήμα 4: Εκτέλεση 2ου σταδίου εντολής
                Tstep_D <= T5;
            WHEN T5 =>    -- Βήμα 5: Εκτέλεση 3ου σταδίου εντολής
(Τέλος)
                Tstep_D <= T0;
        END CASE;
    END PROCESS;

    -- Διεργασία Παραγωγής Σημάτων Ελέγχου (Control Signals) ανά
Κατάσταση και Εντολή
    controlsignals: PROCESS (Tstep_Q, III, Imm, rX, rY, Run)
    BEGIN
        -- Αρχικές/Προεπιλεγμένες τιμές ασφαλείας για όλα τα σήματα
ελέγχου
```

```
        rXin <= '0'; Ain <= '0'; Gin <= '0'; AddSub <= '0'; IRin <=
'0'; Sel <= "----";

        ADDRin <= '0'; DOUTin <= '0'; W_D <= '0'; ALUand <= '0'; Done
<= '0';

        pc_incr <= '0';


        CASE Tstep_Q IS

            WHEN T0 => -- Κατάσταση T0: Ανάκτηση Διεύθυνσης Εντολής

                Sel <= SEL_PC;  -- Προώθηση του PC στο εσωτερικό Bus

                ADDRin <= '1';   -- Ενεργοποίηση καταχωρητή διεύθυνσης
μνήμης

                pc_incr <= Run; -- Αυξάνει ο PC αν ο επεξεργαστής
τρέχει

            WHEN T1 =>

                null;           -- Αναμονή για τη σύγχρονη μνήμη

            WHEN T2 =>

                IRin <= '1';    -- Αποθήκευση των δεδομένων DIN στον
Καταχωρητή Εντολής (IR)


            WHEN T3 => -- Κατάσταση T3: Αρχικό Στάδιο Εκτέλεσης
Εντολών

                CASE III IS

                    WHEN mv =>  -- Εντολή Move

                        IF Imm = '0' THEN Sel <= '0' & rY;  -- mv rX,
rY (μεταφορά από καταχωρητή)

                        ELSE Sel <= SEL_IR8_IR8_0;          -- mv rX,
#D (μεταφορά άμεσης τιμής)

                        END IF;
```

```
                        rXin <= '1';                        -- Εγγραφή
στον καταχωρητή rX

                        Done <= '1';                        -- Η
εντολή ολοκληρώθηκε

                    WHEN mvt => -- Εντολή Move Top (Μετατόπιση 8-bit)

                        Sel <= SEL_IR7_0_0;

                        rXin <= '1';

                        Done <= '1';

                    WHEN add | sub | and_it => -- Αριθμητικές/Λογικές
εντολές

                        Sel <= '0' & rX;                    --
Προώθηση του rX στο Bus

                        Ain <= '1';                         -- Φόρτωση
του rX στον βοηθητικό καταχωρητή A της ALU

                    WHEN ld | st => -- Εντολές Load και Store

                        Sel <= '0' & rY;                    --
Προώθηση της διεύθυνσης από τον rY στο Bus

                        ADDRin <= '1';                      --
Ενημέρωση του καταχωρητή διευθύνσεων μνήμης

                    WHEN OTHERS =>

                        null;

                END CASE;


            WHEN T4 => -- Κατάσταση T4: Δεύτερο Στάδιο Εκτέλεσης

                CASE III IS

                    WHEN add => -- Πρόσθεση

                        IF Imm = '0' THEN Sel <= '0' & rY;
```

```
                        ELSE Sel <= SEL_IR8_IR8_0;

                        END IF;

                        Gin <= '1';                         --
Ενεργοποίηση εγγραφής του αποτελέσματος στον G

                    WHEN sub => -- Αφαίρεση

                        IF Imm = '0' THEN Sel <= '0' & rY;

                        ELSE Sel <= SEL_IR8_IR8_0;

                        END IF;

                        AddSub <= '1';                      --
Ενεργοποίηση λειτουργίας αφαίρεσης στην ALU

                        Gin <= '1';

                    WHEN and_it => -- Λογικό AND

                        IF Imm = '0' THEN Sel <= '0' & rY;

                        ELSE Sel <= SEL_IR8_IR8_0;

                        END IF;

                        ALUand <= '1';                      --
Ενεργοποίηση λειτουργίας AND στην ALU

                        Gin <= '1';

                    WHEN ld =>

                        null;                               -- Κύκλος
αναμονής για ανάγνωση από τη μνήμη

                    WHEN st => -- Αποθήκευση στη μνήμη (Store)

                        Sel <= '0' & rX;                    --
Προώθηση των δεδομένων του rX στο Bus

                        DOUTin <= '1';                      -- Φόρτωση
στον καταχωρητή εξόδου DOUT
```

```
                        W_D <= '1';                         --
Ενεργοποίηση σήματος εγγραφής μνήμης

                        Done <= '1';                        --
Ολοκλήρωση εντολής

                    WHEN OTHERS =>

                        null;

                END CASE;


            WHEN T5 => -- Κατάσταση T5: Τελικό Στάδιο (Write-back)

                CASE III IS

                    WHEN add | sub | and_it =>

                        Sel <= SEL_G;                       --
Προώθηση του αποτελέσματος από τον G στο Bus

                        rXin <= '1';                        --
Αποθήκευση του αποτελέσματος στον καταχωρητή rX

                        Done <= '1';                        --
Ολοκλήρωση

                    WHEN ld => -- Ολοκλήρωση Load

                        Sel <= SEL_DIN;                     --
Προώθηση των δεδομένων από τη μνήμη στο Bus

                        rXin <= '1';                        --
Αποθήκευση στον καταχωρητή rX

                        Done <= '1';                        --
Ολοκλήρωση

                    WHEN OTHERS =>

                        null;

                END CASE;

        END CASE;
```

```
    END PROCESS;


    -- Σύγχρονη ενημέρωση της κατάστασης της FSM (Καταχωρητής
Κατάστασης)
    fsmflipflops: PROCESS (Clock, Resetn, Tstep_D)
    BEGIN
        IF (Resetn = '0') THEN
            Tstep_Q <= T0;
        ELSIF (rising_edge(Clock)) THEN
            Tstep_Q <= Tstep_D;
        END IF;
    END PROCESS;


    -- Διασύνδεση των Γενικών Καταχωρητών r0 έως r6 (Registers 16-bit)
    reg_0:  regn PORT MAP (BusWires, Resetn, Rin(0), Clock, r0);
    reg_1:  regn PORT MAP (BusWires, Resetn, Rin(1), Clock, r1);
    reg_2:  regn PORT MAP (BusWires, Resetn, Rin(2), Clock, r2);
    reg_3:  regn PORT MAP (BusWires, Resetn, Rin(3), Clock, r3);
    reg_4:  regn PORT MAP (BusWires, Resetn, Rin(4), Clock, r4);
    reg_5:  regn PORT MAP (BusWires, Resetn, Rin(5), Clock, r5);
    reg_6:  regn PORT MAP (BusWires, Resetn, Rin(6), Clock, r6);


    -- Διασύνδεση του Μετρητή Προγράμματος (PC ως r7)
    Upc: pc_count PORT MAP (BusWires, Resetn, Clock, pc_incr, Rin(7),
PC);
```

```
    -- Διασύνδεση Εσωτερικών Καταχωρητών του Datapath

    reg_A:    regn PORT MAP (BusWires, Resetn, Ain, Clock, A);       -
- Καταχωρητής ALU εισόδου A

    reg_ADDR: regn PORT MAP (BusWires, Resetn, ADDRin, Clock, ADDR); -
- Καταχωρητής Διευθύνσεων

    reg_DOUT: regn PORT MAP (BusWires, Resetn, DOUTin, Clock, DOUT); -
- Καταχωρητής Δεδομένων Εξόδου

    reg_IR:   regn PORT MAP (DIN, Resetn, IRin, Clock, IR);          -
- Καταχωρητής Εντολών (IR)


    -- Flip-Flop για τον συγχρονισμό του σήματος εγγραφής W (Write
Enable)

    reg_W: flipflop PORT MAP (W_D, Resetn, Clock, W);


    -- Αριθμητική και Λογική Μονάδα (ALU)

    -- Εκτελεί Πρόσθεση, Αφαίρεση ή Λογικό AND μεταξύ του καταχωρητή A
και του Bus

    alu: PROCESS (AddSub, A, BusWires, ALUand)

    BEGIN

        IF ALUand = '0' THEN

            IF AddSub = '0' THEN

                Sum <= A + BusWires;  -- Λειτουργία Πρόσθεσης

            ELSE

                Sum <= A - BusWires;  -- Λειτουργία Αφαίρεσης

            END IF;

        ELSE
```

```
            Sum <= A AND BusWires;     -- Λειτουργία Λογικού AND
        END IF;
    END PROCESS;

    -- Καταχωρητής G: Αποθηκεύει προσωρινά την έξοδο της ALU
    reg_G: regn PORT MAP (Sum, Resetn, Gin, Clock, G);

    -- Πολυπλέκτης Διαύλου (Bus Multiplexer)
    -- Επιλέγει ποια πηγή δεδομένων θα οδηγήσει το κοινό εσωτερικό Bus (BusWires)
    busmux: PROCESS (Sel, r0, r1, r2, r3, r4, r5, r6, PC, G, IR, DIN)
    BEGIN
        CASE Sel IS
            WHEN SEL_R0 => BusWires <= r0;
            WHEN SEL_R1 => BusWires <= r1;
            WHEN SEL_R2 => BusWires <= r2;
            WHEN SEL_R3 => BusWires <= r3;
            WHEN SEL_R4 => BusWires <= r4;
            WHEN SEL_R5 => BusWires <= r5;
            WHEN SEL_R6 => BusWires <= r6;
            WHEN SEL_PC => BusWires <= PC;
            WHEN SEL_G  => BusWires <= G;
            -- Επέκταση προσήμου του άμεσου τελεστέου 9-bit (Sign Extension σε 16-bit)
```

```
            WHEN SEL_IR8_IR8_0 => BusWires <= (15 DOWNTO 9 => IR(8)) &
IR(8 DOWNTO 0);

            -- Ολίσθηση αριστερά κατά 8 θέσεις (χρήσιμο για την εντολή
mvt)

            WHEN SEL_IR7_0_0   => BusWires <= IR(7 DOWNTO 0) &
"00000000";

            WHEN SEL_DIN       => BusWires <= DIN;

            WHEN OTHERS        => BusWires <= (OTHERS => '0');

        END CASE;

    END PROCESS;

END Behavior;


-- ΥΠΟ-ΟΝΤΟΤΗΤΑ: pc_count (Μετρητής Προγράμματος)

LIBRARY ieee;

USE ieee.std_logic_1164.all;

USE ieee.std_logic_unsigned.all;


ENTITY pc_count IS

   PORT ( R   : IN  STD_LOGIC_VECTOR(15 DOWNTO 0);

          Resetn, Clock, E, L  : IN  STD_LOGIC;

          Q   : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));

END pc_count;


ARCHITECTURE Behavior OF pc_count IS

   SIGNAL Count : STD_LOGIC_VECTOR(15 DOWNTO 0);
```

```
BEGIN

   PROCESS (Clock)

   BEGIN

      IF (Clock'EVENT AND Clock = '1') THEN

          IF (Resetn = '0') THEN

            Count <= (OTHERS => '0');  -- Μηδενισμός με
ασύγχρονο/σύγχρονο Reset

         ELSIF (L = '1') THEN

            Count <= R;                -- Παράλληλη φόρτωση νέας
διεύθυνσης (π.χ. διακλάδωση)

         ELSIF (E = '1') THEN

            Count <= Count + 1;        -- Αυτόματη αύξηση για την
επόμενη εντολή

         END IF;

      END IF;

   END PROCESS;

   Q <= Count;

END Behavior;


-- ΥΠΟ-ΟΝΤΟΤΗΤΑ: dec3to8 (Αποκωδικοποιητής 3 σε 8)


LIBRARY ieee;

USE ieee.std_logic_1164.all;


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"; -- Αν δεν είναι ενεργοποιημένος
(Enable=0), όλες οι έξοδοι είναι 0
        ELSE
            CASE W IS        -- Ενεργοποίηση της αντίστοιχης γραμμής
εξόδου
                WHEN "000" => Y <= "10000000";
                WHEN "001" => Y <= "01000000";
                WHEN "010" => Y <= "00100000";
                WHEN "011" => Y <= "00010000";
                WHEN "100" => Y <= "00001000";
                WHEN "101" => Y <= "00000100";
                WHEN "110" => Y <= "00000010";
                WHEN "111" => Y <= "00000001";
                WHEN OTHERS => Y <= "00000000";
            END CASE;
```

```
        END IF;
    END PROCESS;
END Behavior;


-- ΥΠΟ-ΟΝΤΟΤΗΤΑ: regn (Παραμετρικός Καταχωρητής)


LIBRARY ieee;
USE ieee.std_logic_1164.all;


ENTITY regn IS
    GENERIC ( n : INTEGER := 16); -- Προεπιλεγμένο μέγεθος 16-bit
    PORT ( R                  : IN  STD_LOGIC_VECTOR(n-1 DOWNTO 0);
           Resetn, E, Clock   : IN  STD_LOGIC;
           Q                  : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0));
END regn;


ARCHITECTURE Behavior OF regn IS
BEGIN
    PROCESS (Clock)
    BEGIN
        IF Clock'EVENT AND Clock = '1' THEN -- Στο θετικό μέτωπο του
ρολογιού
            IF Resetn = '0' THEN
                Q <= (OTHERS => '0');       -- Καθαρισμός καταχωρητή
            ELSIF E = '1' THEN
```

```
                Q <= R;                         -- Αποθήκευση δεδομένων
εισόδου αν το Enable=1

            END IF;

        END IF;

    END PROCESS;

END Behavior;
```

Α.2 ΜΟΝΑΔΑ ΜΝΗΜΗΣ (inst_mem.vhd)

```
-- ΑΡΧΕΙΟ: inst_mem.vhd

-- ΠΕΡΙΓΡΑΦΗ: Η Μνήμη Εντολών και Δεδομένων (Instruction / Data RAM).

--            Υλοποιεί έναν πίνακα 256 θέσεων x 16-bit και περιέχει

--            προ-φορτωμένο τον κώδικα μηχανής (Machine Code) του
προγράμματος

--            που θα εκτελέσει ο επεξεργαστής κατά την εκκίνηση.

library ieee;

use ieee.std_logic_1164.all;

use ieee.numeric_std.all;

use ieee.std_logic_textio.all;

use std.textio.all;



-- Ορισμός οντότητας της μνήμης

entity inst_mem is

    port (

        address : in  std_logic_vector(7 downto 0);  -- Διεύθυνση
μνήμης 8-bit (καλύπτει 256 θέσεις)

        clock   : in  std_logic;                     -- Σήμα ρολογιού
για σύγχρονη ανάγνωση/εγγραφή

        data    : in  std_logic_vector(15 downto 0); -- Δεδομένα προς
εγγραφή (από τον επεξεργαστή via DOUT)

        wren    : in  std_logic;                     -- Σήμα ελέγχου
εγγραφής (Write Enable, '1' = Εγγραφή)

        q       : out std_logic_vector(15 downto 0)  -- Δεδομένα
εξόδου (προς τον επεξεργαστή via DIN)
```

```
    );
end inst_mem;

-- Αρχιτεκτονική Προσομοίωσης / Συμπεριφοράς της μνήμης
architecture sim of inst_mem is

    -- Ορισμός του τύπου δεδομένων RAM: ένας πίνακας 256 γραμμών, όπου κάθε γραμμή έχει μέγεθος 16-bit
    type ram_type is array (0 to 255)
        of std_logic_vector(15 downto 0);

    -- Δήλωση και αρχικοποίηση του πίνακα RAM με το πρόγραμμα σε κώδικα μηχανής (Hex μορφή)
    signal ram : ram_type := (
        0 => x"3610",  -- Θέση 0: 1η Εντολή προγράμματος (π.χ. ανάγνωση switches ή αρχικοποίηση)
        1 => x"3830",  -- Θέση 1: 2η Εντολή προγράμματος
        2 => x"8004",  -- Θέση 2: 3η Εντολή προγράμματος
        3 => x"a003",  -- Θέση 3: 4η Εντολή προγράμματος
        4 => x"1e02",  -- Θέση 4: 5η Εντολή προγράμματος
        others => x"0000" -- Όλες οι υπόλοιπες 251 θέσεις της μνήμης γεμίζουν με μηδενικά
    );

begin
```

```
    -- Σύγχρονη διεργασία (Process) για τη λειτουργία της μνήμης

    process(clock)

    BEGIN

        if rising_edge(clock) then -- Λειτουργία στο θετικό μέτωπο του
ρολογιού



            -- Λειτουργία Εγγραφής: Αν το wren είναι '1', αποθήκευσε
τα δεδομένα 'data' στη σωστή διεύθυνση

            -- Γίνεται μετατροπή του std_logic_vector σε unsigned και
μετά σε integer για να βρεθεί ο δείκτης του πίνακα

            if wren = '1' then

                ram(to_integer(unsigned(address))) <= data;

            end if;



            -- Λειτουργία Ανάγνωσης (Σύγχρονη): Τα δεδομένα της
επιλεγμένης διεύθυνσης οδηγούνται στην έξοδο Q

            q <= ram(to_integer(unsigned(address)));



        end if;

    end process;


end sim;
```

Α.3 ΟΝΤΟΤΗΤΑ ΚΟΡΥΦΗΣ(part3.vhd)

```
-- ΑΡΧΕΙΟ: part3.vhd

-- ΠΕΡΙΓΡΑΦΗ: Το ενδιάμεσο επίπεδο σχεδίασης (Sub-system).

--            Συνδέει τον επεξεργαστή (proc) με τη μνήμη εντολών
(inst_mem)

--            και τα περιφερειακά στοιχεία (Switches, LEDs) μέσω της

--            τεχνικής χαρτογράφησης μνήμης (Memory-Mapped I/O).


LIBRARY ieee;

USE ieee.std_logic_1164.all;


-- Ορισμός οντότητας συστήματος part3

ENTITY part3 IS

    PORT ( KEY      : IN  STD_LOGIC_VECTOR(0 DOWNTO 0); -- Πλήκτρο
KEY(0) για τη λειτουργία Reset (Active-Low)

           SW       : IN  STD_LOGIC_VECTOR(9 DOWNTO 0); -- Οι 10
διακόπτες της πλακέτας (SW9=Run, SW8-0=Δεδομένα)

           CLOCK_50 : IN  STD_LOGIC;                    -- Κύριο ρολόι
συστήματος 50 MHz

           LEDR     : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)  -- Τα 10 LED
(LEDR9=Κατάσταση Run, LEDR8-0=Δεδομένα)

    );

END part3;


ARCHITECTURE Behavior OF part3 IS

    -- Δήλωση Component: Ο επεξεργαστής μας
```

```
    COMPONENT proc
        PORT ( DIN                  : IN  STD_LOGIC_VECTOR(15 DOWNTO
0);
               Resetn, Clock, Run   : IN  STD_LOGIC;
               DOUT                 : OUT STD_LOGIC_VECTOR(15 DOWNTO
0);
               ADDR                 : OUT STD_LOGIC_VECTOR(15 DOWNTO
0);
               W                    : OUT STD_LOGIC);
    END COMPONENT;

    -- Δήλωση Component: Η μνήμη ROM/RAM (Instruction & Data Memory)
    COMPONENT inst_mem
        PORT ( address  : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
               clock    : IN  STD_LOGIC ;
               data     : IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
               wren     : IN  STD_LOGIC  := '1';
               q        : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
    END COMPONENT;

    -- Δήλωση Component: Παραμετρικός Καταχωρητής για τα περιφερειακά
    COMPONENT regn
        GENERIC ( n : INTEGER := 16);
        PORT ( R                    : IN STD_LOGIC_VECTOR(n-1 DOWNTO
0);
               Resetn, E, Clock     : IN STD_LOGIC;
```

```
            Q                   : OUT STD_LOGIC_VECTOR(n-1 DOWNTO
0));
    END COMPONENT;

    -- Δήλωση Component: Flip-Flop για τον συγχρονισμό σημάτων
    COMPONENT flipflop
        PORT ( D, Resetn, Clock  : IN   STD_LOGIC;
               Q                 : OUT  STD_LOGIC);
    END COMPONENT;

    -- Εσωτερικά σήματα διασύνδεσης
    SIGNAL DOUT, ADDR : STD_LOGIC_VECTOR(15 DOWNTO 0);
    SIGNAL Sync, Run, High, inst_mem_cs, SW_cs, LED_reg_cs :
STD_LOGIC;
    SIGNAL W, W_mem, W_LED : STD_LOGIC;
    SIGNAL DIN, inst_mem_q : STD_LOGIC_VECTOR(15 DOWNTO 0);
    SIGNAL LED_reg, SW_reg : STD_LOGIC_VECTOR(8 DOWNTO 0);
BEGIN
    High <= '1'; -- Μόνιμη λογική κατάσταση '1' (Always Enabled)

    -- Διπλό Flip-Flop (U1 και U2) για τον συγχρονισμό του διακόπτη
SW(9) [Run]
    -- Χρησιμοποιείται για την αποφυγή προβλημάτων μεταστατικότητας
(metastability) με το ρολόι των 50MHz
    U1: flipflop PORT MAP (SW(9), KEY(0), CLOCK_50, Sync);
    U2: flipflop PORT MAP (Sync, KEY(0), CLOCK_50, Run);
```

```
    -- Διασύνδεση του Επεξεργαστή (U3)

    U3: proc PORT MAP (DIN, KEY(0), CLOCK_50, Run, DOUT, ADDR, W);


    -- Αποκωδικοποίηση Διευθύνσεων (Address Decoding) βάσει των 4 MSB
του διαύλου ADDR(15 DOWNTO 12):

    -- Επιλογή Μνήμης (Chip Select): Αν ADDR = 0x0... (0000)

    inst_mem_cs <= '1' WHEN (ADDR(15 DOWNTO 12) = "0000") ELSE '0';

    -- Επιλογή Καταχωρητή LED: Αν ADDR = 0x1... (0001)

    LED_reg_cs <= '1' WHEN (ADDR(15 DOWNTO 12) = "0001") ELSE '0';

    -- Επιλογή Καταχωρητή Διακοπτών: Αν ADDR = 0x3... (0011)

    SW_cs <= '1' WHEN (ADDR(15 DOWNTO 12) = "0011") ELSE '0';


    -- Παραγωγή σήματος εγγραφής στη μνήμη (Write Enable)

    W_mem <= inst_mem_cs AND W;


    -- Διασύνδεση της Μνήμης Εντολών (U4) - Προσβαση στα πρώτα 256
bytes (ADDR 7-0)

    U4: inst_mem PORT MAP (ADDR(7 DOWNTO 0), CLOCK_50, DOUT, W_mem,
inst_mem_q);


    -- Πολυπλέκτης Ανάγνωσης Δεδομένων (Multiplexer) προς τον
Επεξεργαστή:

    -- Επιλέγει ποια δεδομένα θα οδηγηθούν στην είσοδο DIN του
επεξεργαστή ανάλογα με τη διεύθυνση

    multiplexer: PROCESS (inst_mem_cs, SW_cs, inst_mem_q, SW_reg)
```

```
    BEGIN

        IF inst_mem_cs = '1' THEN

            DIN <= inst_mem_q;          -- Ανάγνωση από τη μνήμη
εντολών

        ELSIF SW_cs = '1' THEN

            DIN <= "0000000" & SW_reg;  -- Ανάγνωση των τιμών των
διακοπτών (με επέκταση μηδενικών)

        ELSE

            DIN <= (OTHERS => '-');     -- Κατάσταση Don't Care αν η
διεύθυνση δεν αντιστοιχεί κάπου

        END IF;

    END PROCESS;



    -- Παραγωγή σήματος εγγραφής για τα LED

    W_LED <= LED_reg_cs AND W;



    -- Καταχωρητής U5 (regn, 9-bit): Αποθηκεύει τα δεδομένα εξόδου για
τα LED

    -- Ενεργοποιείται μόνο όταν ο επεξεργαστής εκτελεί εντολή Store
(W=1) στη διεύθυνση των LED

    U5: regn GENERIC MAP (n => 9)

              PORT MAP (DOUT(8 DOWNTO 0), KEY(0), W_LED, CLOCK_50,
LED_reg);



    -- Μόνιμη οδήγηση των εξόδων της πλακέτας

    LEDR(8 DOWNTO 0) <= LED_reg; -- Τα LEDR8-0 δείχνουν τα περιεχόμενα
του καταχωρητή LED
```

```
    LEDR(9) <= Run;                -- Το LEDR9 δείχνει αν ο επεξεργαστής
εκτελεί πρόγραμμα (Run)



    -- Καταχωρητής U6 (regn, 9-bit): Δειγματοληπτεί συνεχώς (High) τις
τιμές των φυσικών διακοπτών SW8-0

    -- ώστε να είναι διαθέσιμες για ανάγνωση από τον επεξεργαστή

    U6: regn GENERIC MAP (n => 9)

              PORT MAP (SW(8 DOWNTO 0), KEY(0), High, CLOCK_50,
SW_reg);

END Behavior;
```

Α.4 ΚΩΔΙΚΑΣ VHDL ΑΝΩΤΑΤΟΥ ΕΠΙΠΕΔΟΥ (top3.vhd)

```
-- ΑΡΧΕΙΟ: top.vhd

-- ΠΕΡΙΓΡΑΦΗ: Το ανώτατο επίπεδο (Top-Level Entity) του συστήματος.

--            Συνδέει τις φυσικές εισόδους/εξόδους της πλακέτας Altera
DE

--            με το υποσύστημα "part3" και απενεργοποιεί τις οθόνες
HEX.



LIBRARY ieee;

USE ieee.std_logic_1164.all;

USE ieee.std_logic_unsigned.all;



-- Ορισμός της οντότητας (Entity) με τις φυσικές θύρες της πλακέτας

ENTITY top IS

    PORT (

        CLOCK_50  : IN    STD_LOGIC;                      -- Κύριο
σήμα ρολογιού συχνότητας 50 MHz

        KEY       : IN    STD_LOGIC_VECTOR( 3 DOWNTO 0);  -- Τα 4
παλμικά πλήκτρα (Pushbuttons) της πλακέτας

        SW        : IN    STD_LOGIC_VECTOR( 9 DOWNTO 0);  -- Οι 10
toggles διακόπτες (Switches) εισόδου

        HEX0      : OUT   STD_LOGIC_VECTOR( 6 DOWNTO 0);  -- Έξοδος
για την 7-segment οθόνη HEX0

        HEX1      : OUT   STD_LOGIC_VECTOR( 6 DOWNTO 0);  -- Έξοδος
για την 7-segment οθόνη HEX1

        HEX2      : OUT   STD_LOGIC_VECTOR( 6 DOWNTO 0);  -- Έξοδος
για την 7-segment οθόνη HEX2
```

```
        HEX3      : OUT   STD_LOGIC_VECTOR( 6 DOWNTO 0);  -- Έξοδος
για την 7-segment οθόνη HEX3

        HEX4      : OUT   STD_LOGIC_VECTOR( 6 DOWNTO 0);  -- Έξοδος
για την 7-segment οθόνη HEX4

        HEX5      : OUT   STD_LOGIC_VECTOR( 6 DOWNTO 0);  -- Έξοδος
για την 7-segment οθόνη HEX5

        LEDR      : OUT   STD_LOGIC_VECTOR( 9 DOWNTO 0)   -- Τα 10
κόκκινα LED εξόδου της πλακέτας

    );

END top;


-- Περιγραφή της συμπεριφοράς και της αρχιτεκτονικής του Top-Level

ARCHITECTURE Behavior OF top IS


    -- Δήλωση του εσωτερικού component "part3" που περιέχει τον
επεξεργαστή

    COMPONENT part3

        PORT (

            KEY      : IN  STD_LOGIC_VECTOR( 0 DOWNTO 0); -- Πλήκτρο
ελέγχου (Reset ή χειροκίνητο Clock)

            SW       : IN  STD_LOGIC_VECTOR( 9 DOWNTO 0); -- Διακόπτες
εισόδου δεδομένων

            CLOCK_50 : IN  STD_LOGIC;                     -- Σήμα
ρολογιού συστήματος

            LEDR     : OUT STD_LOGIC_VECTOR( 9 DOWNTO 0)  -- Έξοδος
στα LED

        );

    END COMPONENT;
```

```
BEGIN


    -- Διασύνδεση (Instantiating) του component part3 ως U1 με τις θύρες της πλακέτας

    -- Περνάει μόνο το πρώτο πλήκτρο KEY(0), τους διακόπτες SW, το ρολόι 50MHz και τα LEDR

    U1: part3 PORT MAP (KEY(0 DOWNTO 0), SW, CLOCK_50, LEDR);



    -- Απενεργοποίηση των οθονών 7-Segment (HEX0 έως HEX5)

    -- Στις συγκεκριμένες πλακέτες, η λογική είναι Active-Low.

    -- Η τιμή "1111111" (όλα άσσοι) σβήνει όλα τα τμήματα της κάθε οθόνης.

    HEX0 <= "1111111";

    HEX1 <= "1111111";

    HEX2 <= "1111111";

    HEX3 <= "1111111";

    HEX4 <= "1111111";

    HEX5 <= "1111111";


END Behavior;
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ελληνική Βιβλιογραφία

[1] Ασθενίδης, Γ., & Παπαδόπουλος, Κ. (2015). *Ψηφιακά Συστήματα και Σχεδίαση με VHDL*. Εκδόσεις Τζιόλα.

[2] Καλαϊτζάκης, Κ., & Κουτρούλης, Ε. (2012). *Ψηφιακά Συστήματα*. Εκδόσεις Κλειδάριθμος.

Ξενόγλωσση Βιβλιογραφία

[3] Brown, S., & Vranesic, Z. (2014). *Fundamentals of Digital Logic with VHDL Design* (3rd ed.). McGraw-Hill Education.

[4] Chu, P. P. (2008). *FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version*. Wiley-Interscience.

[5] Intel Corporation. (2020). *Quartus Prime Infrastructure Essentials*. Intel FPGA Training.

[6] Intel FPGA University Program. (2018). *Laboratory Exercise 9: A Simple Processor*. Intel Corporation.

[7] Intel FPGA University Program. (2018). *Laboratory Exercise 10: A Better Processor*. Intel Corporation.

[8] Roth, C. H., & John, L. K. (2016). *Digital Systems Design Using VHDL*. Cengage Learning.

[9] SlideServe Platform. (2026). *Educational Diagrams for VHDL Design Flow and Load-Store RISC Architecture*.

[Οπισθόφυλλο. Κενή σελίδα]