



ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

*Διατμηματικό ΠΜΣ των τμημάτων Φυσικής και Πληροφορικής και
Τηλεπικοινωνιών με ειδίκευση στον Ηλεκτρονικό Αυτοματισμό*

ΣΧΕΔΙΑΣΗ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Project 2: Σχεδίαση Επεξεργαστή Πολλών Κύκλων
Αρχιτεκτονικής ARM

ΛΟΥΚΑΣ ΔΡΟΣΟΣ

Επιβλέπων Καθηγητές:

Κρανίτης Νεκτάριος, Βασιλόπουλος Διονύσης

Αθήνα 2023 – 2024

Περιεχόμενα

Εισαγωγή.....	1
1. Περιγραφή των στοιχείων και της δομής του επεξεργαστή	
1.1. Σύνολο των εντολών που υλοποιήθηκαν.....	2
1.2. Σχηματικό διάγραμμα στο επίπεδο RTL του Datapath του επεξεργαστή.....	4
1.3. Υπομονάδες του Control Unit.....	14
1.4. Σχηματικό διάγραμμα στο επίπεδο RTL του Control Unit του επεξεργαστή.....	20
1.5. Μηχανή πεπερασμένων καταστάσεων (FSM) του Control Unit.....	24
1.6. Σχηματικό διάγραμμα στο επίπεδο RTL του επεξεργαστή.....	29
1.7. Μέγιστη συχνότητα λειτουργίας και κρίσιμες διαδρομές.....	30
2. Επαλήθευση της ορθής σχεδίασης και λειτουργίας του επεξεργαστή	
2.1. Πρόγραμμα ARM σε γλώσσα Assembly για επαλήθευση της ορθής σχεδίασης και λειτουργίας του επεξεργαστή.....	31
2.2. Προγράμματα δοκιμής και τα διαγράμματα χρονισμού για Register File και ALU.....	37
2.3. Διαγράμματα χρονισμού του επεξεργαστή.....	52
3. Ανάλυση των αποτελεσμάτων της σύνθεσης και της υλοποίησης του επεξεργαστή.....	57
4. Περιγραφή της λειτουργίας της εντολής ROR.....	61

Εισαγωγή

Η παρούσα βιβλιογραφική εργασία έχει ως κύριο στόχο την υλοποίηση ενός απλοποιημένου επεξεργαστή πολλών κύκλων αρχιτεκτονικής ARM σε τεχνολογία FPGA. Ο επεξεργαστής αυτός σχεδιάστηκε και υλοποιήθηκε με σκοπό την κατανόηση και την επίδειξη των βασικών αρχών λειτουργίας της αρχιτεκτονικής ARM σε περιβάλλον πολλών κύκλων. Η εργασία πραγματοποιήθηκε στο εργαλείο σύνθεσης VIVADO IDE (Integrated Development Environment) της Xilinx, χρησιμοποιώντας ως target language την γλώσσα VHDL και ως simulator language την επιλογή Mixed (VHDL και Verilog). Τέλος, η υλοποίηση του επεξεργαστή γίνεται πάνω σε μια αναπτυξιακή κάρτα της Xilinx με κωδικό xc7z020clg484-1 (FPGA Zynq 7000 ZC702).

Η εργασία οργανώνεται σε τέσσερα κύρια κεφάλαια. Στο 1^ο κεφάλαιο, παρουσιάζεται αναλυτικά η δομή του επεξεργαστή, συμπεριλαμβανομένων των βασικών συστατικών του, όπως η Διαδρομή Δεδομένων (Datapath) και η Μονάδα Ελέγχου (Control Unit), μαζί με τα διάφορα στοιχεία τους. Στο 2^ο κεφάλαιο ελέγχεται η σωστή λειτουργία του επεξεργαστή και η απόδοσή του επεξεργαστή μέσα από προσομοιώσεις και δοκιμές, ενώ παρουσιάζονται και τα αποτελέσματα των δοκιμών και τα testbenches που χρησιμοποιήθηκαν για την επαλήθευση. Το κεφάλαιο 3 αναλύει τα αποτελέσματα της σύνθεσης και της υλοποίησης του επεξεργαστή. Παρουσιάζονται τα αποτελέσματα της σύνθεσης, όπως η χρήση των πόρων (resource utilization), η χρονική απόδοση (timing performance) και η κατανάλωση ενέργειας (power consumption). Το κεφάλαιο 4 είναι το τελευταίο κεφάλαιο της εργασίας και εξηγεί θεωρητικά μόνο την εντολή ROR (Rotate Right) στην αρχιτεκτονική ARM στον επεξεργαστή, χωρίς όμως η εντολή αυτή να υλοποιείται στην εργασία.

1. Περιγραφή των στοιχείων και της δομής του επεξεργαστή

1.1. Σύνολο των εντολών που υλοποιήθηκαν

Οι εντολές που εκτελεί ο επεξεργαστής χωρίζονται σε 3 κατηγορίες, εντολές επεξεργασίας δεδομένων, εντολές μνήμης και εντολές διακλάδωσης. Όλες οι παρακάτω εντολές εκτελούνται με βάση κάποιες συγκεκριμένες συνθήκες που καθορίζονται από τις σημαίες N, Z, C και V ανάλογα την εντολή.

1.1.1. Εντολές Επεξεργασίας Δεδομένων

Αυτές οι εντολές περιλαμβάνουν αριθμητικές και λογικές πράξεις που εκτελούνται από την ALU (Αριθμητική Λογική Μονάδα). Η λειτουργία της ALU καθορίζεται από ένα σήμα ελέγχου (ALUControl) που έχει 3 bit.

1. ADD(S), SUB(S), CMP – επηρεάζουν τις σημαίες N, Z, C, V

- **ADD(S):** Αριθμητική εντολή η οποία προσθέτει δύο αριθμούς. Συγκεκριμένα προσθέτει το περιεχόμενο είτε δύο καταχωρητών προέλευσης είτε ενός καταχωρητή και μιας ακέραιας σταθεράς και το αποθηκεύει σε έναν καταχωρητή προορισμού. Οι σημαίες N (Negative), Z (Zero), C (Carry), V (Overflow) ενημερώνονται ανάλογα με το αποτέλεσμα της πράξης.
- **SUB(S):** Αριθμητική εντολή η οποία αφαιρεί δύο αριθμούς. Συγκεκριμένα αφαιρεί το περιεχόμενο είτε δύο καταχωρητών προέλευσης είτε ενός καταχωρητή και μιας ακέραιας σταθεράς και το αποθηκεύει σε έναν καταχωρητή προορισμού. Οι σημαίες N, Z, C, V ενημερώνονται ανάλογα με το αποτέλεσμα της πράξης.
- **CMP:** Εντολή Σύγκρισης η οποία συγκρίνει το περιεχόμενο δύο καταχωρητών προέλευσης με αφαίρεση, χωρίς να αποθηκεύει το αποτέλεσμα. Οι σημαίες N, Z, C, V ενημερώνονται ανάλογα με το αποτέλεσμα της πράξης.

2. AND(S), EOR(S) – επηρεάζουν τις σημαίες N, Z

- **AND(S):** Λογική εντολή η οποία εκτελεί λογική AND μεταξύ του περιεχομένου δύο καταχωρητών προέλευσης είτε ενός καταχωρητή και μιας ακέραιας σταθεράς και το αποθηκεύει σε έναν καταχωρητή προορισμού, ενώ ταυτόχρονα ενημερώνει τις σημαίες N και Z.
- **EOR(S):** Λογική εντολή η οποία εκτελεί λογική XOR μεταξύ του περιεχομένου δύο καταχωρητών προέλευσης είτε ενός καταχωρητή και μιας ακέραιας σταθεράς και το αποθηκεύει σε έναν καταχωρητή προορισμού, ενώ ταυτόχρονα ενημερώνει τις σημαίες N και Z.

3. MOV, MVN – S = 0

- **MOV**: Μεταφέρει είτε το περιεχόμενο ενός καταχωρητή προέλευσης είτε μια ακέραια σταθερά σε έναν καταχωρητή προορισμού. Δεν επηρεάζει καμία σημαία ($S = 0$).
- **MVN**: Μεταφέρει το αντίστροφο είτε του περιεχομένου ενός καταχωρητή προέλευσης είτε μιας ακέραιας σταθεράς σε έναν καταχωρητή προορισμού. Δεν επηρεάζει καμία σημαία ($S = 0$).

4. LSL, ASR – S = 0

- **LSL**: Εντολή ολίσθησης, πραγματοποιεί λογική ολίσθηση προς τα αριστερά των bits του περιεχομένου ενός καταχωρητή προέλευσης κατά μια σταθερά και το αποθηκεύει σε έναν καταχωρητή προορισμού. Δεν επηρεάζει καμία σημαία ($S = 0$).
- **ASR**: Εντολή ολίσθησης, πραγματοποιεί αριθμητική ολίσθηση προς τα δεξιά των bits του περιεχομένου ενός καταχωρητή προέλευσης κατά μια σταθερά και το αποθηκεύει σε έναν καταχωρητή προορισμού. Δεν επηρεάζει καμία σημαία ($S = 0$).

1.1.2. Εντολές Μνήμης

Αυτές οι εντολές χειρίζονται την ανάγνωση και εγγραφή δεδομένων από/προς τη Μνήμη Δεδομένων (RAM).

- **LDR** : Φορτώνει δεδομένα από τη RAM σε έναν καταχωρητή προορισμού.
- **STR** : Αποθηκεύει δεδομένα από έναν καταχωρητή προέλευσης στην RAM.

1.1.3. Εντολές Διακλάδωσης

Αυτές οι εντολές χειρίζονται τις διακλαδώσεις στο πρόγραμμα, δηλαδή την αλλαγή της ροής εκτέλεσης του προγράμματος και εκτέλεσης εντολών σε άλλα σημεία του κώδικα.

- **B** : Πηγαίνει σε μια νέα διεύθυνση προγράμματος, όπου εκεί πλέον εκτελούνται εντολές.
- **BL** : Πηγαίνει σε μια νέα διεύθυνση προγράμματος, όπου εκεί πλέον εκτελούνται εντολές, και την αποθηκεύει για να μπορεί να επιστρέψει σε αυτήν αργότερα.

1.2. Σχηματικό διάγραμμα στο επίπεδο RTL του Datapath του επεξεργαστή

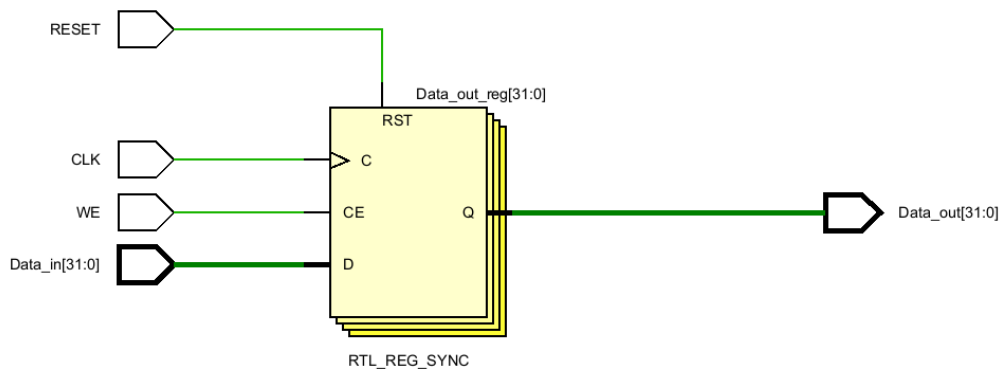
Γενικά, τα σήματα εξόδου του Control Unit αποτελούν εισόδους στο Datapath. Τα σήματα εισόδου που δέχεται το Datapath από το Control Unit είναι:

- **CLK** (1 bit): Το σήμα ρολογιού του γενικού κυκλώματος.
- **RESET** (1 bit): Το σήμα reset του γενικού κυκλώματος.
- **RegSrc** (3 bits): Ελέγχει πολυπλέκτες για να αποφασίσει μεταξύ πολλών εισόδων για τους καταχωρητές.
- **ALUSrc** (1 bit): Επιλέγει την πηγή της δεύτερης εισόδου της ALU μεταξύ ενός καταχωρητή και μιας αμέσως τιμής.
- **MemtoReg** (1 bit): Επιλέγει την πηγή των δεδομένων που θα γραφτούν σε καταχωρητή (PC ή αρχείο καταχωρητών).
- **ALUControl** (3 bits): Επιλέγει την κατάλληλη πράξη που θα εκτελέσει η ALU.
- **ImmSrc** (1 bit): Επιλέγει επέκταση πρόσημου ή επέκταση μηδενός.
- **IRWrite** (1 bit): Επιτρέπει την εγγραφή στον Instruction Register.
- **RegWrite** (1 bit): Επιτρέπει την εγγραφή στον Register File.
- **MAWrite** (1 bit): Επιτρέπει την εγγραφή στον Memory Address Register.
- **MemWrite** (1 bit): Επιτρέπει την εγγραφή στην RAM.
- **FlagsWrite** (1 bit): Επιτρέπει την ανανέωση των σημαιών συνθήκης στον status register με βάση το αποτέλεσμα της ALU.
- **PCSrc** (2 bits): Καθορίζει την προέλευση της επόμενης τιμής του PC.
- **PCWrite** (1 bit): Επιτρέπει την ανανέωση του PC.

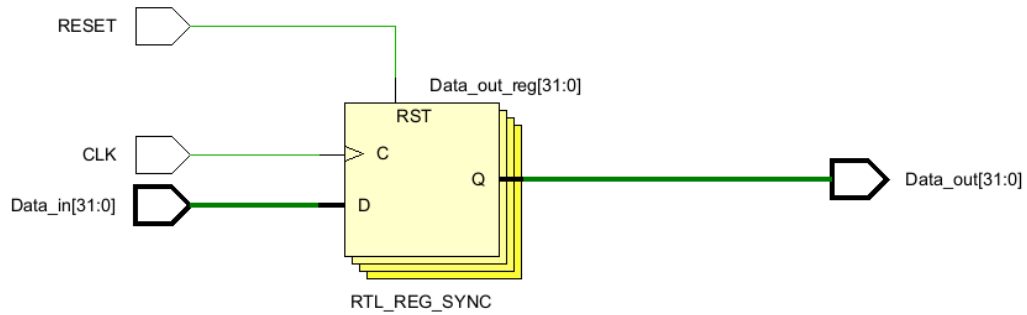
Τα σήματα εξόδου του datapath είναι:

- **flags** (4 bits): Σημαίες συνθήκης που δείχνουν το αποτέλεσμα της τελευταίας πράξης που εκτέλεσε η ALU. Το bit 0 είναι η σημαία V, το bit 1 είναι η σημαία C, το bit 2 είναι η σημαία Z και το bit 3 είναι η σημαία N.
- **PC** (32 bits): Η έξοδος του PC Register.
- **instr** (32 bits): Η έξοδος της μνήμης εντολών ROM που δείχνει την τωρινή εντολή.
- **ALUResult** (32 bits): Αποτέλεσμα της πράξης της ALU.
- **WriteData** (32 bits): Δεδομένα που θα γραφτούν στην RAM.
- **Result** (32 bits): Δεδομένα που θα γραφτούν στο αρχείο καταχωρητών.

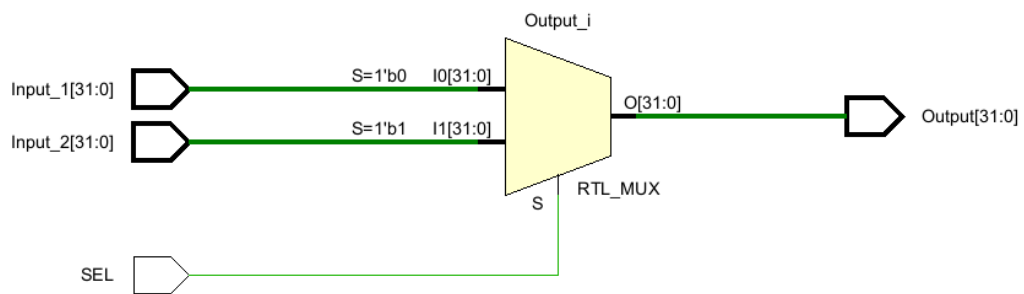
Η υλοποίηση του datapath έγινε με την χρήση 25 components, της μνήμης ROM, της μνήμης RAM, της αριθμητικής και λογικής μονάδας ALU, του αρχείου καταχωρητών, της μονάδας επέκτασης μηδενός ή προσήμου, 6 πολυπλεκτών 2 σε 1, ενός πολυπλέκτη 3 σε 1, 2 αθροιστών κατά 4, και συνολικά 11 καταχωρητών, 4 με σήματα Reset και Write Enable και 7 με μόνο σήμα Reset. Τα σχηματικά διαγράμματα στο επίπεδο RTL των component αυτών δίνονται παρακάτω:



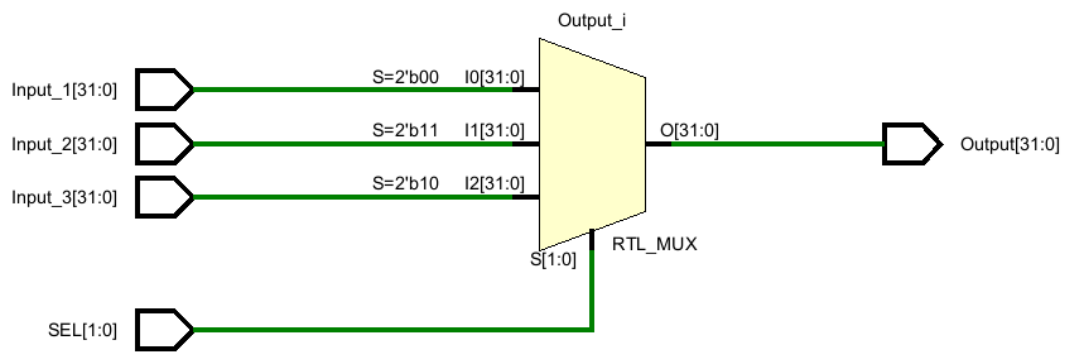
Εικόνα 1.1 RTL Schematic καταχωρητή με σήματα RESET και Write Enable (WE).



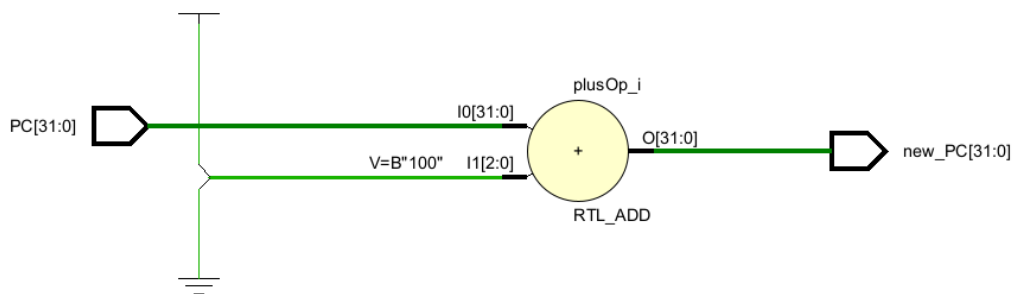
Εικόνα 1.2 RTL Schematic καταχωρητή με σήμα RESET.



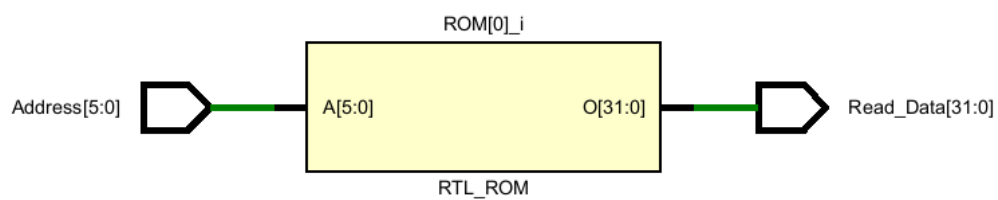
Εικόνα 1.3 RTL Schematic πολυπλέκτη 2 σε 1.



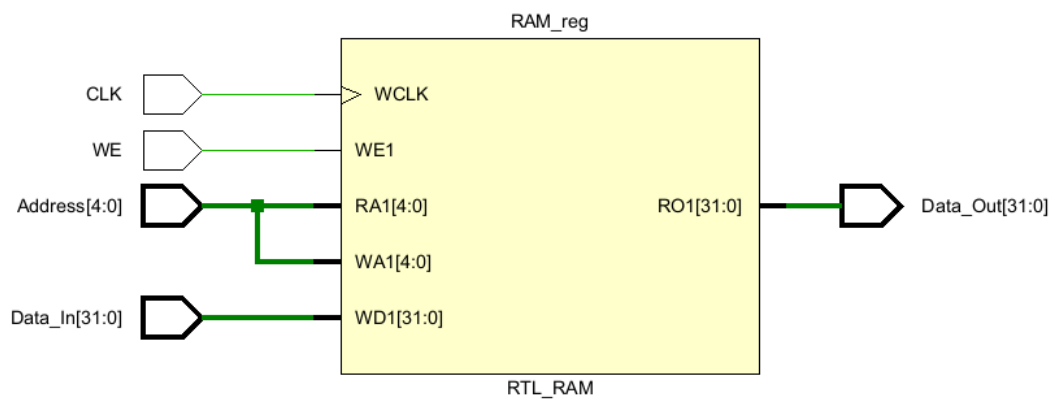
Εικόνα 1.4 *RTL Schematic πολυπλέκτη 3 σε 1.*



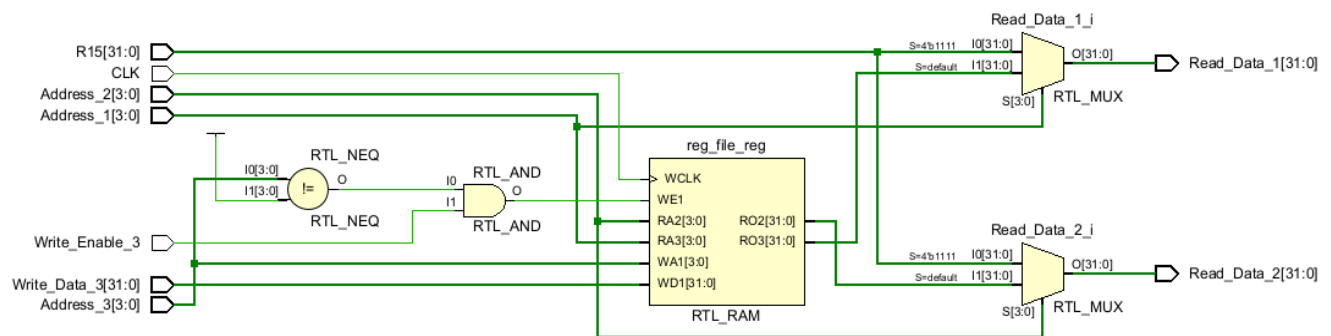
Εικόνα 1.5 *RTL Schematic αθροιστή κατά 4.*



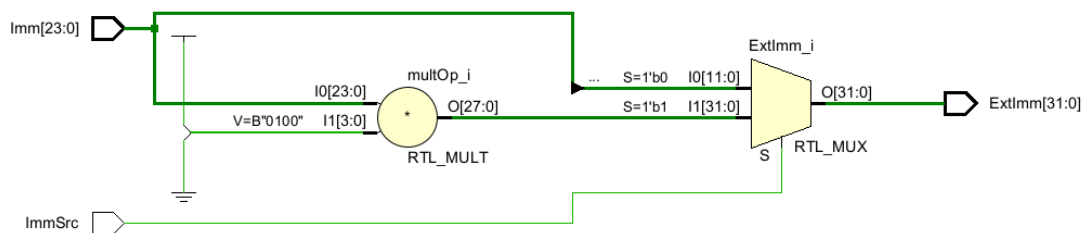
Εικόνα 1.6 *RTL Schematic της Μνήμης Εντολών ROM.*



Εικόνα 1.7 RTL Schematic της Μνήμης Δεδομένων RAM.



Εικόνα 1.8 RTL Schematic του αρχείου καταχωρητών (Register File).



Εικόνα 1.9 RTL Schematic της μονάδας επέκτασης (Extend Unit).

Στην εικόνα 1.11 παρουσιάζεται το σχηματικό διάγραμμα στο επίπεδο RTL του Datapath του επεξεργαστή. Στην συνέχεια θα εξηγήσουμε τα στοιχεία που αποτελούν το datapath.

Ο Μετρητής Προγράμματος PC_Register (Program Counter) είναι ένας καταχωρητής 32 bit με σήματα reset και write enable, ο οποίος κρατά τη διεύθυνση της επόμενης εντολής. Οι είσοδοι CLK και RESET συνδέονται με τα σήματα CLK και RESET του γενικού κυκλώματος αντίστοιχα. Το σήμα WE είναι το σήμα PCWrite που παράγεται από το control unit και είναι αυτό που επιτρέπει την εγγραφή ή όχι στον καταχωρητή. Αν η είσοδος WE έχει τιμή '1' τότε η τιμή στην είσοδο Data_in του καταχωρητή εγγράφεται σε αυτόν και διατηρείται και στην έξοδό του Data_out, μέχρι την επόμενη εγγραφή. Η έξοδος του PC είναι επίσης και το περιεχόμενο του σήματος εξόδου PC 32 bit του datapath.

Η Μνήμη Εντολών ROM (Instruction_Memory) περιέχει 64 καταχωρήσεις λέξεων των 32 bit και διαβάζει ασύγχρονα την διεύθυνση που περιέχεται στο PC και η έξοδος της ROM είναι το περιεχόμενο της διεύθυνσης αυτής, δηλαδή η επόμενη εντολή που θα εκτελεστεί. Συγκεκριμένα, δέχεται ως είσοδο τα bits 7 έως 2 της εξόδου του PC, καθώς η είσοδος της ROM πρέπει να είναι μεγέθους 6 bit (καθώς $2^6 = 64$, αναπαριστούνται όλες οι εντολές της μνήμης δεδομένων RAM με τόσα bits).

Η έξοδος της ROM είναι είσοδος στον μη αρχιτεκτονικό καταχωρητή Instruction_Register. Όμοια με τον PC_Register, είναι ένας καταχωρητής 32 bit με σήματα reset και write enable, ο οποίος διαβάζει την μνήμη του προγράμματος με τη διεύθυνση που είναι αποθηκευμένη στον PC. Οι είσοδοι CLK και RESET συνδέονται με τα σήματα CLK και RESET του γενικού κυκλώματος αντίστοιχα. Το σήμα WE είναι το σήμα IRWrite που παράγεται από το control unit και είναι αυτό που επιτρέπει την εγγραφή ή όχι στον καταχωρητή. Αν η είσοδος WE έχει τιμή '1' τότε η τιμή στην είσοδο Data_in του καταχωρητή εγγράφεται σε αυτόν και διατηρείται και στην έξοδό του Data_out, μέχρι την επόμενη εγγραφή. Η έξοδος του Instruction_Register είναι επίσης και το περιεχόμενο του σήματος εξόδου instr 32 bit του datapath, ενώ ταυτόχρονα είναι και σήμα εισόδου στο control unit.

Ο αθροιστής κατά 4 PCPlus4 αυξάνει την διεύθυνση του PC κατά 4, βρίσκοντας έτσι την διεύθυνση της επόμενης σειριακής εντολής που θα εκτελεστεί. Η μία είσοδος του αθροιστή είναι η τωρινή διεύθυνση του PC, ενώ η άλλη είναι η σταθερή τιμή 4 σε μορφή 32 bit (00000000000000000000000000000100). Το αποτέλεσμα της πρόσθεσης είναι και η έξοδος του αθροιστή.

Η έξοδος του PCPlus4 είναι είσοδος στον μη αρχιτεκτονικό καταχωρητή PCPlus4_Register. Όμοια με τον Instruction_Register, είναι ένας καταχωρητής 32 bit με σήματα reset και write enable, ο οποίος υπολογίζει την διεύθυνση της επόμενης κατά σειρά εντολής από τον PC. Σε περίπτωση, όμως, που έχουμε εντολές διακλάδωσης και γενικά δεν έχουμε σειριακή εκτέλεση του προγράμματος, η έξοδος του PCPlus4_Register εισάγεται στον πολυπλέκτη 3 σε 1 PCSource. Οι είσοδοι CLK

και RESET συνδέονται με τα σήματα CLK και RESET του γενικού κυκλώματος αντίστοιχα. Το σήμα WE είναι το σήμα IRWrite που παράγεται από το control unit και είναι αυτό που επιτρέπει την εγγραφή ή όχι στον καταχωρητή. Αν η είσοδος WE έχει τιμή '1' τότε η τιμή στην είσοδο Data_in του καταχωρητή εγγράφεται σε αυτόν και διατηρείται και στην έξοδό του Data_out, μέχρι την επόμενη εγγραφή.

Ο αθροιστής κατά 4 PCPlus8 αυξάνει την διεύθυνση του PCPlus4 κατά 4. Αυτό χρειάζεται καθώς στην αρχιτεκτονική ARM, όταν διαβάζεται ο καταχωρητής R15 τότε επιστρέφεται η τιμή PC+8 και όταν γράφεται ο καταχωρητής R15 ανανεώνεται το PC. Η μία είσοδος του αθροιστή είναι η τωρινή διεύθυνση του PCPlus4, ενώ η άλλη είναι η σταθερή τιμή 4 σε μορφή 32 bit (00000000000000000000000000000100). Το αποτέλεσμα της πρόσθεσης είναι και η έξοδος του αθροιστή.

Το Αρχείο Καταχωρητών (Register_File) κρατά και διαβάζει ή γράφει σε καταχωρητές. Αποτελείται συνολικά από 15 αρχιτεκτονικούς καταχωρητές, ενώ οι καταχωρητές R15 (δηλαδή PC) και Status_Register είναι ξεχωριστές οντότητες. Συγκεκριμένα, το Register_File χρησιμοποιεί τις εισόδους ανάγνωσης Address_1 (4 bits) και Address_2 (4 bits) για ασύγχρονη ανάγνωση και την είσοδο Address_3 (4 bits) για σύγχρονη εγγραφή στον Register_File. Οι εισοδοί αυτοί είναι οι έξοδοι των πολυπλεκτών 2 σε 1 RA1, RA2 και WA αντίστοιχα. Τα δεδομένα εγγραφής περιλαμβάνονται στην είσοδο Write_Data_3 (32 bits), η οποία περιλαμβάνει το αποτέλεσμα του πολυπλέκτη 2 σε 1 WD3. Η εγγραφή είναι σύγχρονη, κατά την ακμή του ρολογιού και αν το σήμα RegWrite του control unit έχει την τιμή 1. Η είσοδος CLK συνδέεται με το σήμα CLK του γενικού κυκλώματος, ενώ η είσοδος R15 περιέχει το αποτέλεσμα του αθροιστή PCPlus8. Τα δεδομένα των Address_1 και Address_2 διαβάζονται από τις αντίστοιχες θύρες εξόδου Read_Data_1 (32 bits) και Read_Data_2 (32 bits). Αν μια θύρα ανάγνωσης (Address_1 και Address_2) έχει την τιμή 15, τότε η αντίστοιχη θύρα εξόδου (Read_Data_1 και Read_Data_2) θα είναι η είσοδος R15.

Ο πολυπλέκτης 2 σε 1 RA1 4 bit επιλέγει την είσοδο Address_1 του Register_File με βάση το 1^ο από τα 3 bits του σήματος RegSrc του control unit (RegSrc(0)). Η πρώτη είσοδος του πολυπλέκτη είναι τα bits 19 με 16 της εξόδου του Instruction_Register, τα οποία για τις εντολές επεξεργασίας δεδομένων ADD, SUB, AND, EOR και CMP αντιστοιχούν στην διεύθυνση του πρώτου καταχωρητή προέλευσης Rn, ενώ για τις εντολές μνήμης είναι η διεύθυνση του καταχωρητή βάσης. Η άλλη είσοδος είναι η σταθερή τιμή 1111 (=15), που χρησιμοποιείται στις εντολές διακλάδωσης για την ανάγνωση της τιμής PC+8. Όταν RegSrc(0) = 0 επιλέγεται η πρώτη είσοδος, ενώ όταν είναι 1 επιλέγεται η δεύτερη είσοδος.

Ο πολυπλέκτης 2 σε 1 RA2 4 bit επιλέγει την είσοδο Address_2 του Register_File με βάση το 2^ο από τα 3 bits του σήματος RegSrc του control unit (RegSrc(1)). Η πρώτη είσοδος του πολυπλέκτη είναι τα bits 3 με 0 της εξόδου του Instruction_Register, τα οποία για τις εντολές επεξεργασίας δεδομένων ADD, SUB, AND, EOR και CMP αντιστοιχούν στην διεύθυνση του δεύτερου καταχωρητή προέλευσης Rm, ενώ για τις εντολές επεξεργασίας δεδομένων MVN, MOV, LSL, ASR είναι ο μοναδικός

καταχωρητής προέλευσης. Η δεύτερη είσοδος του πολυπλέκτη είναι τα bits 15 με 12 της εξόδου του Instruction_Register, τα οποία αντιστοιχούν στην διεύθυνση του καταχωρητή προορισμού Rd. Όταν RegSrc(1) = 0 επιλέγεται η πρώτη είσοδος, ενώ όταν είναι 1 επιλέγεται η δεύτερη είσοδος.

Ο πολυπλέκτης 2 σε 1 WA 4 bit επιλέγει την είσοδο Address_3 του Register_File με βάση το 3^ο από τα 3 bits του σήματος RegSrc του control unit (RegSrc(2)). Η πρώτη είσοδος του πολυπλέκτη είναι τα bits 15 με 12 της εξόδου του Instruction_Register, τα οποία αντιστοιχούν στην διεύθυνση του καταχωρητή προορισμού Rd. Η δεύτερη είσοδος του πολυπλέκτη είναι η σταθερή τιμή 1110 (=14), που χρησιμοποιείται στην εντολή διακλάδωσης BL για την εγγραφή της τιμής PC+4 στην διεύθυνση R14. Όταν RegSrc(2) = 0 επιλέγεται η πρώτη είσοδος, ενώ όταν είναι 1 επιλέγεται η δεύτερη είσοδος.

Ο πολυπλέκτης 2 σε 1 WD3 32 bit επιλέγει την είσοδο Write_Data_3 του Register_File με βάση το 3^ο από τα 3 bits του σήματος RegSrc του control unit (RegSrc(2)). Η πρώτη είσοδος του πολυπλέκτη είναι η έξοδος του πολυπλέκτη 2 σε 1 Rd_Source. Η δεύτερη είσοδος του πολυπλέκτη είναι η έξοδος του PCPlus4_Register. Όταν RegSrc(2) = 0 επιλέγεται η πρώτη είσοδος, ενώ όταν είναι 1 επιλέγεται η δεύτερη είσοδος.

Η έξοδος Read_Data_1 του Register_File είναι είσοδος στον μη αρχιτεκτονικό καταχωρητή A_Register. Είναι ένας καταχωρητής 32 bit με σήμα reset ο οποίος κρατά την τιμή RD1. Οι είσοδοι CLK και RESET συνδέονται με τα σήματα CLK και RESET του γενικού κυκλώματος αντίστοιχα. Η εγγραφή στον A_Register γίνεται κατά την ακμή του σήματος του clock.

Η έξοδος Read_Data_2 του Register_File είναι είσοδος στον μη αρχιτεκτονικό καταχωρητή B_Register. Είναι ένας καταχωρητής 32 bit με σήμα reset ο οποίος κρατά την τιμή RD2. Οι είσοδοι CLK και RESET συνδέονται με τα σήματα CLK και RESET του γενικού κυκλώματος αντίστοιχα. Η εγγραφή στον B_Register γίνεται κατά την ακμή του σήματος του clock. Η έξοδος του B_Register είναι επίσης και το περιεχόμενο του σήματος εξόδου WriteData 32 bit του datapath.

Η μονάδα επέκτασης μηδενός ή προσήμου Extend_Unit έχει 2 εισόδους, τα bits 23 με 0 της εξόδου instr του datapath, δηλαδή της εξόδου του Instruction_Register, και το σήμα ImmSrc που παράγεται από το control unit. Το πρώτο σήμα εισόδου αντιστοιχεί στο πεδίο προσδιορισμού διεύθυνσης στόχου της διακλάδωσης imm24 των εντολών διακλάδωσης. Αν έχουμε ImmSrc = 0, τότε στα bit 11 με 0 του instr, τα οποία αντιστοιχούν στο τμήμα imm12 για τις εντολές μνήμης αλλά και για τις εντολές επεξεργασίας δεδομένων που έχουν άμεσο τελεστή προέλευσης, γίνεται επέκταση μηδενός από τα 12 στα 32 bit. Αν έχουμε ImmSrc = 1, τότε στα bit 23 με 0 του instr πραγματοποιείται αρχικά επέκταση προσήμου από 24 σε 26 bit, πολλαπλασιάζοντας το instr επί 4, και στην συνέχεια γίνεται επέκταση στα 32 bit.

Η έξοδος ExtImm του Extend_Unit είναι είσοδος στον μη αρχιτεκτονικό καταχωρητή I_Register. Είναι ένας καταχωρητής 32 bit με σήμα reset ο οποίος κρατά την τιμή

Extend_Output. Οι είσοδοι CLK και RESET συνδέονται με τα σήματα CLK και RESET του γενικού κυκλώματος αντίστοιχα. Η εγγραφή στον I_Register γίνεται κατά την ακμή του σήματος του clock.

Ο πολυπλέκτης 2 σε 1 ALUSrcB 32 bit επιλέγει την είσοδο SrcB του ALU_Unit με βάση το σήμα ALUSrc του control unit. Η πρώτη είσοδος του πολυπλέκτη είναι η έξοδος του B_Register. Η δεύτερη είσοδος του πολυπλέκτη είναι η έξοδος του I_Register. Όταν ALUSrc = 0 επιλέγεται η πρώτη είσοδος, ενώ όταν είναι 1 επιλέγεται η δεύτερη είσοδος.

Η Αριθμητική και Λογική Μονάδα ALU_Unit (Arithmetic Logic Unit) εκτελεί αριθμητικές και λογικές πράξεις στα δεδομένα εισόδου της. Συγκεκριμένα, εκτελεί πρόσθεση, αφαίρεση, λογικό “and”, λογικό “xor”, αντιγραφή, αντιστροφή, λογική ολίσθηση αριστερά και αριθμητική ολίσθηση δεξιά. Οι πράξεις αυτές εκτελούνται με βάση της τιμής του σήματος ALUControl (3 bit) του control unit. Συγκεκριμένα, για ALUControl ίσο με 000 εκτελείται πρόσθεση, για 001 αφαίρεση, για 010 λογικό “and”, για 011 λογικό “xor”, για 100 είτε λογική ολίσθηση αριστερά, είτε αριθμητική ολίσθηση δεξιά, είτε αντιγραφή με τελεστή πηγής έναν ακέραιο αριθμό, για 101 αντιγραφή με τελεστή πηγής τα δεδομένα ενός καταχωρητή και για 110 αναστροφή. Η πρώτη είσοδος SrcA (32 bit) δέχεται την έξοδο του A_Register ενώ η δεύτερη είσοδος SrcB (32 bit) δέχεται την έξοδο του ALUSrcB, και οι είσοδοι αυτές θα χρησιμοποιηθούν για τις πράξεις. Η είσοδος shamt5 (5 bit) είναι τα bit 11 με 7 της εξόδου του I_Register και είναι η ποσότητα ολίσθησης στις πράξεις λογικής ολίσθησης αριστερά και αριθμητικής ολίσθησης δεξιά. Η είσοδος sh (2 bit) είναι τα bit 6 με 5 της εξόδου του I_Register και δείχνει την κατεύθυνση της ολίσθησης. Αν sh = 10 τότε πραγματοποιείται αριθμητική ολίσθηση δεξιά, ενώ αν sh = 00 τότε για shamt διαφορετικό του μηδενός εκτελείται λογική ολίσθηση αριστερά, ενώ για shamt ίσο με το μηδέν έχουμε την ειδική περίπτωση που εκτελείται αντιγραφή (εντολή MOV). Η έξοδος ALUResult (32 bit) περιέχει το αποτέλεσμα της πράξης της ALU. Οι έξοδοι N, Z, C και V (1 bit η καθεμία) είναι οι σημαίες κατάστασης του προγράμματος. Οι εντολές ADD(S), SUB(S), CMP επηρεάζουν τις σημαίες N, Z, C, V, ενώ οι εντολές AND(S), EOR(S) επηρεάζουν τις σημαίες N, Z. Η σημαία N ενημερώνεται όταν το αποτέλεσμα είναι αρνητικό και η σημαία Z ενημερώνεται όταν το αποτέλεσμα είναι μηδέν, δηλαδή όλα τα bit του είναι 0, ενώ οι σημαίες C και V ενημερώνονται όταν το αποτέλεσμα πρόσθεσης ή αφαίρεσης παράγει κρατούμενο C, για μη προσημασμένους, και overflow V, για προσημασμένους αριθμούς, αντίστοιχα. Η έξοδος της ALU_Unit είναι επίσης και το περιεχόμενο του σήματος εξόδου ALUResult 32 bit του datapath.

Οι έξοδοι N, Z, C και V της ALU_Unit είναι είσοδος στον καταχωρητή κατάστασης Status_Register. Είναι ένας καταχωρητής 4 bit με σήματα reset και write enable, ο οποίος κρατά τα flags της ALU. Οι είσοδοι CLK και RESET συνδέονται με τα σήματα CLK και RESET του γενικού κυκλώματος αντίστοιχα. Το σήμα WE είναι το σήμα FlagsWrite που παράγεται από το control unit και είναι αυτό που επιτρέπει την εγγραφή ή όχι στον καταχωρητή. Αν η είσοδος WE έχει τιμή ‘1’ τότε η τιμή στην

είσοδο Data_in του καταχωρητή εγγράφεται σε αυτόν και διατηρείται και στην έξοδό του Data_out, μέχρι την επόμενη εγγραφή. Η έξοδος του Status_Register είναι επίσης και το περιεχόμενο του σήματος εξόδου flags 4 bit του datapath, ενώ ταυτόχρονα είναι και σήμα εισόδου στο control unit.

Τα bit 6 με 2 της εξόδου της ALU_Unit είναι είσοδος στον μη αρχιτεκτονικό καταχωρητή Memory_Address_Register. Είναι ένας καταχωρητής 5 bit με σήματα reset και write enable, ο οποίος αποθηκεύει το αποτέλεσμα της ALU. Οι είσοδοι CLK και RESET συνδέονται με τα σήματα CLK και RESET του γενικού κυκλώματος αντίστοιχα. Το σήμα WE είναι το σήμα MAWrite που παράγεται από το control unit και είναι αυτό που επιτρέπει την εγγραφή ή όχι στον καταχωρητή. Αν η είσοδος WE έχει τιμή '1' τότε η τιμή στην είσοδο Data_in του καταχωρητή εγγράφεται σε αυτόν και διατηρείται και στην έξοδό του Data_out, μέχρι την επόμενη εγγραφή.

Η έξοδος του B_Register είναι είσοδος στον μη αρχιτεκτονικό καταχωρητή Write_Data_Register. Είναι ένας καταχωρητής 32 bit με σήμα reset ο οποίος κρατά τα δεδομένα που θα γραφτούν στη μνήμη. Οι είσοδοι CLK και RESET συνδέονται με τα σήματα CLK και RESET του γενικού κυκλώματος αντίστοιχα. Η εγγραφή στον Write_Data_Register γίνεται κατά την ακμή του σήματος του clock.

Η έξοδος της ALU_Unit είναι είσοδος στον μη αρχιτεκτονικό καταχωρητή S_Register. Είναι ένας καταχωρητής 32 bit με σήμα reset. Οι είσοδοι CLK και RESET συνδέονται με τα σήματα CLK και RESET του γενικού κυκλώματος αντίστοιχα. Η εγγραφή στον S_Register γίνεται κατά την ακμή του σήματος του clock.

Η μνήμη δεδομένων Data_Memory 32 bit του επεξεργαστή έχει ως είσοδο Address την έξοδο του Memory_Address_Register και ως είσοδο Data_In την έξοδο του Write_Data_Register. Η είσοδος CLK συνδέεται με το σήμα CLK του γενικού κυκλώματος, ενώ η είσοδος WE είναι το σήμα MemWrite που παράγεται από το control unit και είναι αυτό που επιτρέπει ή όχι την εγγραφή. Η έξοδος της μνήμης Data_Out περιέχει τα αποθηκευμένα δεδομένα της διεύθυνσης που ορίζει η είσοδος Address.

Η έξοδος της RAM είναι είσοδος στον μη αρχιτεκτονικό καταχωρητή Read_Data_Register. Είναι ένας καταχωρητής 32 bit με σήμα reset. Οι είσοδοι CLK και RESET συνδέονται με τα σήματα CLK και RESET του γενικού κυκλώματος αντίστοιχα. Η εγγραφή στον Read_Data_Register γίνεται κατά την ακμή του σήματος του clock.

Ο πολυπλέκτης 2 σε 1 Rd_Source 32 bit δέχεται ως πρώτη είσοδο την έξοδο του του S_Register και ως δεύτερη είσοδο την έξοδο του Read_Data_Register με βάση το σήμα MemtoReg του control unit. Όταν MemtoReg = 0 επιλέγεται η πρώτη είσοδος, ενώ όταν είναι 1 επιλέγεται η δεύτερη είσοδος. Η έξοδος του Rd_Source είναι επίσης και το περιεχόμενο του σήματος εξόδου Result 32 bit του datapath.

Ο πολυπλέκτης 3 σε 1 PCSource 32 bit δέχεται ως πρώτη είσοδο την έξοδο του του PCPlus4_Register, ως δεύτερη είσοδο την έξοδο της ALU_Unit και ως τρίτη είσοδο

την έξοδο του πολυπλέκτη `Rd_Source`, με βάση το σήμα `PCSrc` του control unit. Όταν `PCSrc = 00` επιλέγεται η πρώτη είσοδος, όταν είναι 11 επιλέγεται η δεύτερη είσοδος και όταν είναι 10 επιλέγεται η τρίτη είσοδος. Η έξοδος του `PCSource` ενημερώνει τον `PC_Register` με τη διεύθυνση της επομένης προς εκτέλεση εντολής.

1.3. Υπομονάδες του Control Unit

Η μονάδα ελέγχου του επεξεργαστή αποτελείται από τρία βασικά components, τον αποκωδικοποιητή εντολής (Instruction Decoder), την μηχανή πεπερασμένων καταστάσεων (Finite State Machine) και την λογικής ελέγχου συνθήκης (Conditional Logic).

1.3.1. Instruction Decoder

Ο Instruction Decoder δέχεται μια εντολή προς εκτέλεση, την αποκωδικοποιεί και την εισάγει στο datapath ως σήματα σταθερής τιμής κατά την εκτέλεση της. Παρακάτω σχεδιάστηκε ο πίνακας αληθείας για τις εντολές ADD, SUB, CMP, AND, EOR, MVN, MOV, LDR, STR, B και BL. Για τις εντολές αυτές είσοδοι είναι τα πεδία `op` και `funct`, καθώς και τα πεδία `shamt5` και `sh` της εντολής προς εκτέλεση `instr`. Συγκεκριμένα, το πεδίο `op` (bit 27 και 26) καθορίζει το είδος της εντολής (επεξεργασίας δεδομένων, μνήμης ή διακλάδωσης), το πεδίο `funct` (25 έως 20) είναι το πεδίο κωδικοποίησης της εντολής και καθορίζει τα χαρακτηριστικά της, το πεδίο `shamt5` (bit 11 έως 7) δηλώνει την ποσότητα ολίσθησης για τις εντολές MOV, LSL, ASR, ενώ το πεδίο `sh` (bit 6 έως 5) δηλώνει τον τύπο της ολίσθησης για αυτές τις εντολές. Έξοδοι είναι τα σήματα ελέγχου `RegSrc[2:0]`, `ALUSrc`, `ImmSrc`, `ALUControl[2:0]` και `MemtoReg`, καθώς και το εσωτερικό σήμα `NoWrite_in`.

Όπως αναφέρθηκε, το πεδίο `op` (bit 27 και 26) καθορίζει το είδος της εντολής ("00" για επεξεργασίας δεδομένων, "01" για μνήμης και "10" για διακλάδωσης). Το πεδίο `funct` καθορίζει ποια εντολή επιλέχτηκε. Για τις εντολές επεξεργασίας δεδομένων, αν το τελευταίο bit του `funct` (`funct(5)`) είναι 0, τότε έχουμε τελεστή πηγής τα δεδομένα ενός καταχωρητή, ενώ αν είναι 1 τότε έχουμε τελεστή πηγής έναν ακέραιο αριθμό. Οι εντολές LSL, ASR και MOV με καταχωρητή έχουν το ίδιο ακριβώς πεδίο `funct` και η διαφοροποίηση τους γίνεται από τα πεδία `shamt5` και `sh`. Το πεδίο `sh` της εντολής ASR είναι '10', ενώ το πεδίο `shamt5` μπορεί να έχει οποιαδήποτε τιμή. Οι εντολές LSL και MOV με καταχωρητή έχουν πεδίο `sh` ίσο με '01', οπότε η διαφοροποίηση τους γίνεται από το πεδίο `shamt5`. Η εντολή MOV με καταχωρητή έχει πεδίο `shamt5` αναγκαστικά ίσο με '00000', ενώ το πεδίο `shamt5` της εντολής LSL έχει οποιαδήποτε άλλη τιμή. Στον πίνακα, το πεδίο `shamt5` της εντολής LSL παρουσιάζεται με την τιμή 'XXXXX' ως οποιαδήποτε τιμή διάφορη του μηδενός, καθώς ο Instruction Decoder έχει ειδική συνθήκη για την περίπτωση όπου το `shamt5` είναι μηδενικό.

Το σήμα `RegSrc` χρησιμοποιείται για τον έλεγχο των πολυπλεκτών των θυρών εισόδου του Register File. Ο πολυπλέκτης της RA1 της εισόδου `Address_1` ελέγχεται από το bit 0 του σήματος `RegSrc` (`RegSrc(0)`), Ο πολυπλέκτης της RA2 της εισόδου `Address_2` ελέγχεται από το bit 1 του σήματος `RegSrc` (`RegSrc(1)`) και οι πολυπλέκτες

WA και WD3 των εισόδων Address_3 και Write_Data_3 ελέγχονται από το bit 2 του σήματος RegSrc (RegSrc(2)). Για τις εντολές επεξεργασίας δεδομένων ADD, SUB, AND και EOR ισχύει ότι RegSrc = '000' όταν έχουμε δύο τελεστές προέλευσης ως δεδομένα καταχωρητών. Στην περίπτωση αυτή στην είσοδο Address_1 εισάγεται η διεύθυνση του πρώτου τελεστή προέλευσης Rn, στην είσοδο Address_2 εισάγεται η διεύθυνση του δεύτερου τελεστή προέλευσης Rm, στην είσοδο Address_3 εισάγεται η διεύθυνση προορισμού Rd και στην είσοδο Write_Data_3 να εισάγεται το αποτέλεσμα της ALU. Για τις εντολές επεξεργασίας δεδομένων ADD, SUB, AND και EOR ισχύει ότι RegSrc = '0X0' όταν έχουμε έναν τελεστή προέλευσης ως δεδομένα καταχωρητή και έναν τελεστή προέλευσης άμεσο ακέραιο. Στην περίπτωση αυτή στην είσοδο Address_1 εισάγεται η διεύθυνση του πρώτου τελεστή προέλευσης Rn, η είσοδος Address_2 δεν χρησιμοποιείται (RegSrc(1) = 'X'), στην είσοδο Address_3 εισάγεται η διεύθυνση προορισμού Rd και στην είσοδο Write_Data_3 να εισάγεται το αποτέλεσμα της ALU. Για τις εντολές MOV και MVN με τελεστή προορισμού δεδομένα καταχωρητή ισχύει RegSrc = '00X', καθώς διαβάζεται ο δεύτερος τελεστής προέλευσης Rm και το αποτέλεσμα της ALU εγγράφεται στον καταχωρητή προορισμού Rd. Για τις εντολές MOV και MVN με τελεστή προορισμού άμεσο ακέραιο ισχύει RegSrc = '0XX', καθώς χρησιμοποιούνται μόνο οι θύρες Address_3 και Write_Data_3 στις οποίες εγγράφονται η διεύθυνση τελεστή προορισμού Rd και το αποτέλεσμα της ALU, αντίστοιχα. Για τις εντολές LSL και ASR με τελεστή προορισμού άμεσο ακέραιο ισχύει RegSrc = '00X', καθώς διαβάζεται ο δεύτερος τελεστής προέλευσης Rm και το αποτέλεσμα της ALU εγγράφεται στον καταχωρητή προορισμού Rd. Για την εντολή CMP με τελεστή προέλευσης άμεσο ακέραιο ισχύει RegSrc = 'XX0' καθώς διαβάζεται μόνο ο τελεστής προέλευσης Rn στην θύρα Address_1. Για την εντολή CMP με τελεστή προέλευσης δεδομένα καταχωρητή ισχύει RegSrc = 'X00' καθώς διαβάζεται ο τελεστής προέλευσης Rn στην θύρα Address_1, αλλά και καταχωρητή προέλευσης Rm στην Address_2. Για την εντολή STR ισχύει RegSrc = 'X10' καθώς διαβάζεται ως offset στη μνήμη δεδομένων ο τελεστής προέλευσης Rn της θύρας Address_1, και διαβάζονται επίσης τα δεδομένα της μνήμης δεδομένων που περιέχονται στον τελεστή προορισμού Rd της θύρας Address_2. Για την εντολή LDR ισχύει RegSrc = '0X0' καθώς διαβάζεται ως offset στη μνήμη δεδομένων ο τελεστής προέλευσης Rn της θύρας Address_1, και εγγράφονται στην μνήμη δεδομένων τα δεδομένα του τελεστή προορισμού Rd της θύρας Address_3. Για την εντολή B ισχύει RegSrc = 'XX1', καθώς απαιτείται μόνο ανάγνωση της τιμής PC+8 παίρνοντας την διεύθυνση της θύρας Address_1 ως 15. Για την εντολή BL ισχύει RegSrc = '1X1', καθώς απαιτείται ανάγνωση της τιμής PC+8 παίρνοντας την διεύθυνση της θύρας Address_1 ως 15 και ταυτόχρονα εγγράφεται η διεύθυνση PC+4 στον link register R14.

Το σήμα ALUSrc ελέγχει τον πολυπλέκτη που επιλέγει το σήμα του δεύτερου τελεστή προέλευσης. Ισχύει ALUSrc = '0' για τις εντολές επεξεργασίας δεδομένων ADD, SUB, AND, EOR, CMP, MVN και MOV, όταν έχουμε τελεστή προέλευσης την έξοδο RD2 του Register File. Ισχύει ALUSrc = '0' και για τις εντολές LSL και ASR. Ισχύει ALUSrc = '1' για τις εντολές επεξεργασίας δεδομένων ADD, SUB, AND, EOR, CMP, MVN και MOV όταν έχουμε τελεστή προέλευσης άμεσο ακέραιο. Στην περίπτωση αυτή το περιεχόμενο του δεύτερου τελεστή προέλευσης είναι η έξοδος

της μονάδας επέκτασης μηδενός ή προσήμου. Ισχύει $ALUSrc = '1'$ και για όλες τις εντολές μνήμης και όλες τις εντολές διακλάδωσης.

Το σήμα $ImmSrc$ προσδιορίζει τη λειτουργία της μονάδας επέκτασης μηδενός ή προσήμου. Για όλες τις εντολές επεξεργασίας δεδομένων που δεν έχουν τελεστή προορισμού άμεσο ακέραιο ισούται με την αδιάφορη τιμή X , δηλαδή δεν χρησιμοποιείται η μονάδα επέκτασης μηδενός ή προσήμου. Για όλες τις εντολές επεξεργασίας δεδομένων με άμεσο ακέραιο τελεστή προέλευσης και για τις εντολές επεξεργασίας μνήμης ισχύει $ImmSrc = '0'$ καθώς πραγματοποιείται επέκταση μηδενός. Για όλες τις εντολές διακλάδωσης ισχύει $ImmSrc = '1'$ καθώς πραγματοποιείται επέκταση προσήμου και πολλαπλασιασμό επί 4.

Το σήμα $ALUControl$ χρησιμοποιείται για να επιλέξει την πράξη που θα πραγματοποιήσει η ALU. Συγκεκριμένα, για $ALUControl$ ίσο με 000 εκτελείται πρόσθεση, για 001 αφαίρεση, για 010 λογικό “and”, για 011 λογικό “xor”, για 100 είτε λογική ολίσθηση αριστερά, είτε αριθμητική ολίσθηση δεξιά, είτε αντιγραφή με τελεστή πηγής έναν ακέραιο αριθμό, για 101 αντιγραφή με τελεστή πηγής τα δεδομένα ενός καταχωρητή και για 110 αναστροφή. Η εντολή CMP χρησιμοποιεί την τιμή 001 που αντιστοιχεί σε αυτήν της αφαίρεσης. Για εντολές μνήμης με πρόσθεση του offset έχει την τιμή 000, ενώ για αφαίρεση του offset έχει την τιμή 001. Για εντολές διακλάδωσης έχει πάντα την τιμή 000, που αντιστοιχεί σε αυτήν της πρόσθεσης.

Το σήμα $MemtoReg$ ελέγχει την είσοδο του $Register_File$ και του PC. Για την εντολή LDR ισχύει $MemtoReg = '1'$ όπου το σήμα εισόδου θα είναι η έξοδος ανάγνωσης της Μνήμης Δεδομένων. Για όλες τις άλλες εντολές εκτός των εντολών CMP και STR έχει την τιμή '0', καθώς επιλεγεί την έξοδο της ALU για εγγραφή στον $Register_File$. Για τις εντολές CMP και STR δεν πραγματοποιείται εγγραφή στον $Register_File$, άρα έχει την αδιάφορη τιμή X .

Το σήμα $NoWrite_in$ είναι ένα εσωτερικό σήμα το οποίο χρησιμοποιείται για την διαφοροποίηση της εντολής CMP από τις άλλες εντολές επεξεργασίας. Για το σήμα αυτό ισχύει $NoWrite_in = '1'$ μόνο για την εντολή CMP.

Εντολή	Instr27:26 op	Instr25:20 funct	Instr11:7 shamt5	Instr6:5 sh	Τύπος	RegSrc	ALUSrc	ImmSrc	ALUControl	MemtoReg	NoWrite_in
ADD	00	1 0100 X	XXXXX	XX	DP Imm	0X0	1	0	000	0	0
ADD	00	0 0100 X	XXXXX	XX	DP Reg	000	0	X	000	0	0
SUB	00	1 0010 X	XXXXX	XX	DP Imm	0X0	1	0	001	0	0
SUB	00	0 0010 X	XXXXX	XX	DP Reg	000	0	X	001	0	0
CMP	00	1 1010 1	XXXXX	XX	DP Imm	XX0	1	0	001	X	1
CMP	00	0 1010 1	XXXXX	XX	DP Reg	X00	0	X	001	X	1
AND	00	1 0000 X	XXXXX	XX	DP Imm	0X0	1	0	010	0	0
AND	00	0 0000 X	XXXXX	XX	DP Reg	000	0	X	010	0	0
EOR	00	1 0001 X	XXXXX	XX	DP Imm	0X0	1	0	011	0	0
EOR	00	0 0001 X	XXXXX	XX	DP Reg	000	0	X	011	0	0
MVN	00	1 1111 X	XXXXX	XX	DP Imm	0XX	1	0	110	0	0
MVN	00	0 1111 X	XXXXX	XX	DP Reg	00X	0	X	110	0	0
MOV	00	1 1101 X	XXXXX	XX	DP Imm	0XX	1	0	101	0	0
MOV	00	0 1101 X	00000	00	DP Reg	00X	0	X	100	0	0
LSL	00	0 1101 X	XXXXX	00	DP Imm	00X	0	X	100	0	0
ASR	00	0 1101 X	XXXXX	10	DP Imm	00X	0	X	100	0	0
LDR	01	X X1XX 1	XXXXX	XX	M Imm +	0X0	1	0	000	1	0
LDR	01	X X0XX 1	XXXXX	XX	M Imm -	0X0	1	0	001	1	0
STR	01	X X1XX 0	XXXXX	XX	M Imm +	X10	1	0	000	X	0
STR	01	X X0XX 0	XXXXX	XX	M Imm -	X10	1	0	001	X	0
B	10	X 0XXX X	XXXXX	XX	B Imm +	XX1	1	1	000	0	0
BL	10	X 1XXX X	XXXXX	XX	B Imm +	1X1	1	1	000	0	0

1.3.2. Conditional Logic

Η αρχιτεκτονική ARM υποστηρίζει εντολές υπό συνθήκη. Το μνημονικό της εντολής ακολουθείται από ένα μνημονικό συνθήκης, το οποίο υποδεικνύει τη συνθήκη (CondEx) που πρέπει να ικανοποιείται για να εκτελεσθεί η συγκεκριμένη εντολή. Το μνημονικό συνθήκης αποθηκεύεται στο πεδίο συνθήκης (cond) της εντολής μεγέθους 4 bit (cond = instr_{31:28}), το οποίο δέχεται ως είσοδο το Conditional Logic. Η λογική ελέγχου συνθήκης (Conditional Logic) χρησιμοποιείται για την παραγωγή των σημάτων ελέγχου που παραμένουν σταθερά κατά την εκτέλεση της εντολής, επομένως δέχεται ως είσοδο και τα flags N, Z, C, V, δηλαδή την έξοδο του Status Register (Status_Register_Output). Το σήμα εξόδου του Conditional Logic είναι το σήμα CondEx_in, το οποίο αποτελεί και σήμα εισόδου στην Μηχανή Καταστάσεων FSM του control unit. Στον επόμενο πίνακα φαίνονται τα μνημονικά συνθήκης με τις εξισώσεις Boole των σημαιών που τις ικανοποιούν. Αν οι σημαίες του σήματος εισόδου flags έχουν τιμή που αντιστοιχεί στα μνημονικά βάσει του πίνακα τότε το σήμα εξόδου CondEx_in έχει τιμή '1' και θα εκτελεστεί η εντολή. Από την άλλη, αν δεν υπάρχει αντιστοίχιση βάσει του πίνακα τότε ισχύει CondEx_in = '0' και θα υπολογιστεί η επόμενη εντολή προς εκτέλεση.

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

1.3.3. Finite State Machine (FSM)

Στη μικροαρχιτεκτονική ARM του επεξεργαστή πολλών κύκλων ολόκληρη η εντολή εκτελείται σε 3 έως 5 κύκλους ρολογιού, δηλαδή εκτελούνται σε πολλά βήματα τα οποία καθορίζονται από τη Μηχανή Πεπερασμένων Καταστάσεων FSM (Finite State Machine) του control unit. Τα βήματα αυτά θα εξεταστούν στην ενότητα 1.5 “Μηχανή πεπερασμένων καταστάσεων (FSM) του Control Unit” παρακάτω.

Τα σήματα εισόδου του FSM είναι τα σήματα CLK και RESET, τα σήματα op, NoWrite_in και CondEx_in, αλλά και τα σήματα S, L και Rd. Τα σήματα CLK και RESET συνδέονται με τα σήματα CLK και RESET του γενικού κυκλώματος αντίστοιχα. Τα δύο εσωτερικά σήματα NoWrite_in, το οποίο διαφοροποιεί την εντολή CMP από τις υπόλοιπες data-processing εντολές, και CondEx_in, το οποίο προσδιορίζει αν θα εκτελεστεί η εντολή ή όχι με βάση το μνημονικό της αλλά και το περιεχόμενου του Status Register, και είναι τα σήματα που παράγονται από τον Instruction Decoder και το Conditional Logic αντίστοιχα. Ένα άλλο σήμα εισόδου είναι το πεδίο op (bit 27 έως 26 της εντολής instr), το οποίο προσδιορίζει τον τύπο εντολής. Το σήμα εισόδου S (bit 20 της εντολής instr) έχει διαφορετικές ιδιότητες ανάλογα το είδος της εντολής. Για εντολές επεξεργασίας δεδομένων είναι το πεδίο S, δηλαδή το πεδίο ενημέρωσης σημαίων. Για εντολές μνήμης είναι το πεδίο L το οποίο διαφοροποιεί την εντολή LDR από την εντολή STR. Το σήμα εισόδου L (bit 24 της εντολής instr) χρησιμοποιείται στις εντολές διακλάδωσης για να διαφοροποιήσει την εντολή B από την εντολή BL. Τέλος είναι το σήμα εισόδου Rd (bit 15 έως 12 της εντολής instr) το οποίο διαθέτει τον τελεστή προορισμού της εντολής.

Τα σήματα εξόδου του FSM είναι τα είναι σήματα εισόδου στο Datapath και συγκεκριμένα είναι το σήμα IRWrite για τον Instruction Register, το σήμα RegWrite για τον Register File, το σήμα MAWrite για τον Memory Address Register, το σήμα MemWrite για την μνήμη δεδομένων RAM, το σήμα FlagsWrite για τον Status Register, το σήμα PCWrite για τον Μετρητή Προγράμματος PC και το σήμα PCSrc για τον πολυπλέκτη επιλογής σήματος εισόδου του PC.

1.4. Σχηματικό διάγραμμα στο επίπεδο RTL του Control Unit του επεξεργαστή

Γενικά, οι εισόδους του Control Unit είναι οι έξοδοι του Datapath και είναι τμήματα του σήματος instr της εντολής προς εκτέλεση και το σήμα flags, δηλαδή οι σημαίες κατάστασης του Status Register. Τα σήματα εισόδου που δέχεται το Control Unit από το Datapath είναι:

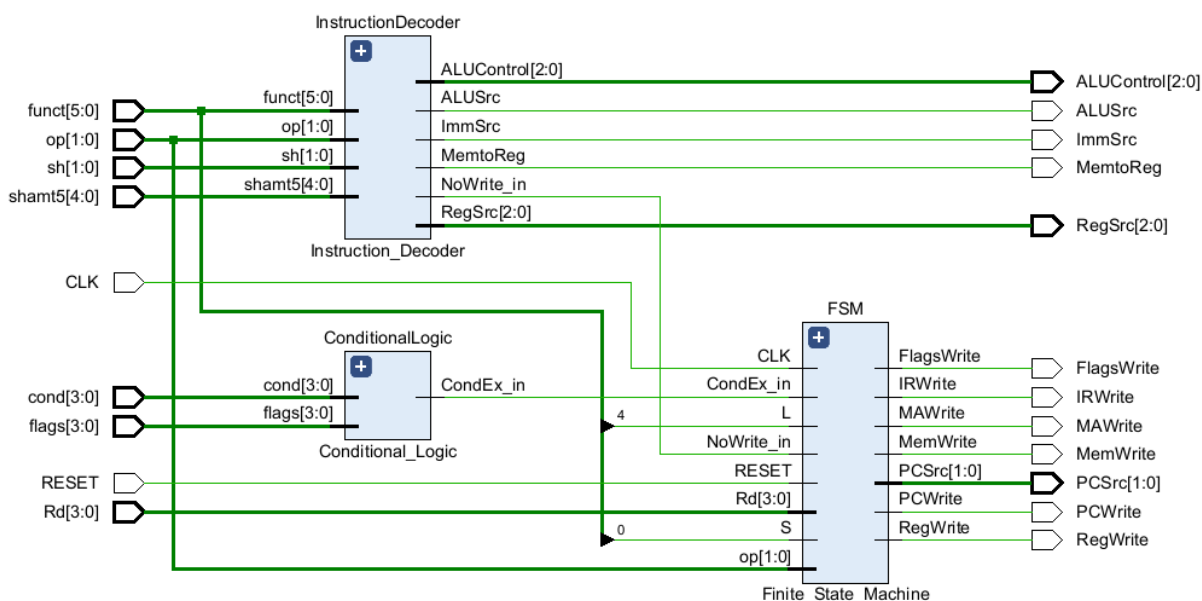
- **CLK** (1 bit): Το σήμα ρολογιού του γενικού κυκλώματος.
- **RESET** (1 bit): Το σήμα reset του γενικού κυκλώματος, επαναφέρει το FSM στην αρχική του κατάσταση S0.
- **cond** (4 bits): Τα bit 31 με 28, του σήματος instr της εντολής.
- **op** (2 bits): Τα bit 27 με 26 του σήματος instr της εντολής.
- **funct** (6 bits): Τα bit 25 με 20 του σήματος instr της εντολής.
- **shamt5** (5 bits): Τα bit 11 με 7 του σήματος instr της εντολής.
- **sh** (2 bits): Τα bit 6 με 5 του σήματος instr της εντολής.
- **Rd** (4 bits): Καταχωρητής εξόδου της εντολής.
- **flags** (4 bits): Σημαίες συνθήκης που δείχνουν το αποτέλεσμα της τελευταίας πράξης που εκτέλεσε η ALU. Το bit 0 είναι η σημαία V, το bit 1 είναι η σημαία C, το bit 2 είναι η σημαία Z και το bit 3 είναι η σημαία N.

Τα σήματα εξόδου του Control Unit είναι:

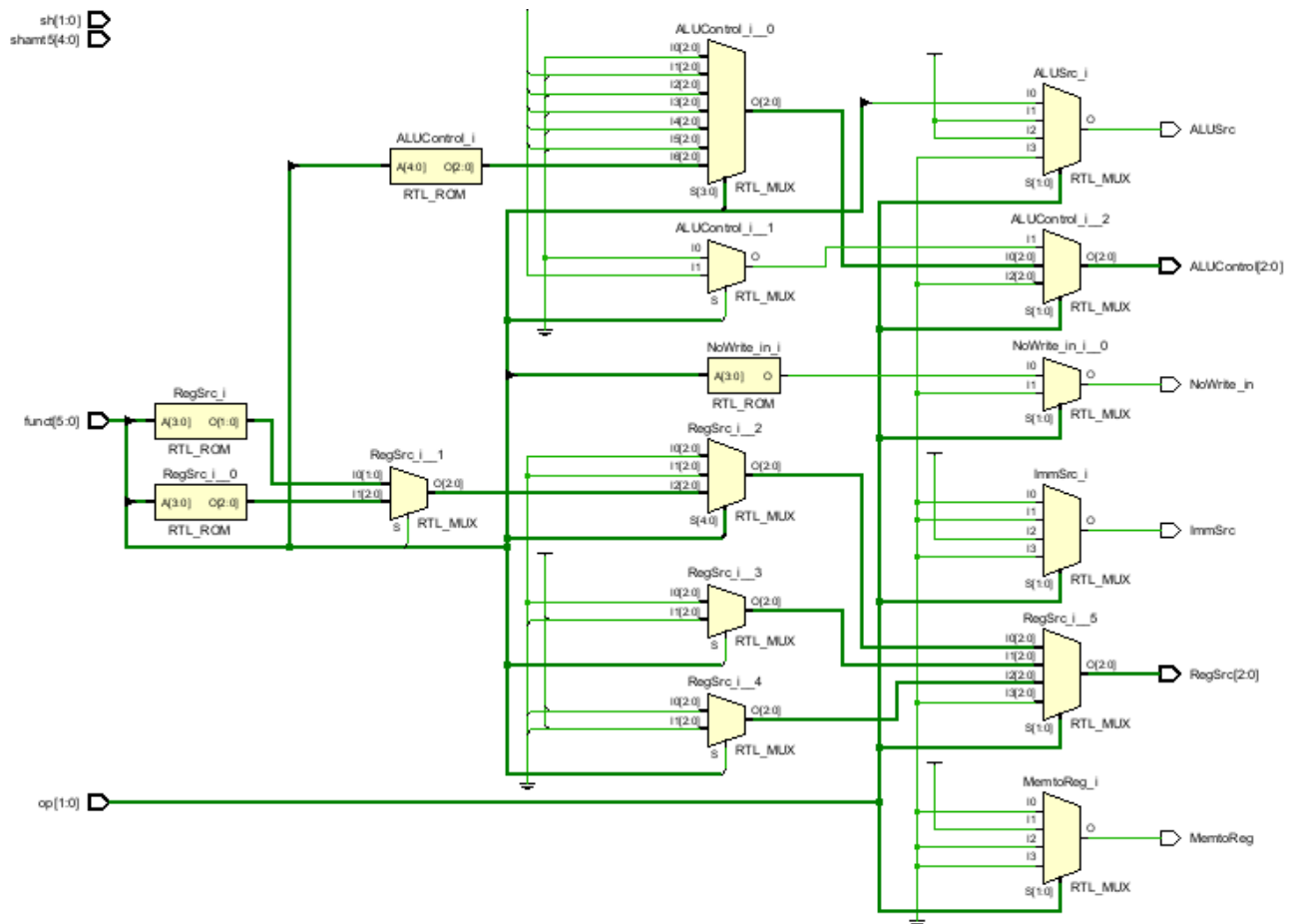
- **RegSrc** (3 bits): Ελέγχει πολυπλέκτες για να αποφασίσει μεταξύ πολλών εισόδων για τους καταχωρητές.
- **ALUSrc** (1 bit): Επιλέγει την πηγή της δεύτερης εισόδου της ALU μεταξύ ενός καταχωρητή και μιας ακέραιας τιμής.
- **MemtoReg** (1 bit): Επιλέγει την πηγή των δεδομένων που θα γραφτούν σε καταχωρητή (PC ή αρχείο καταχωρητών).
- **ALUControl** (3 bits): Επιλέγει την κατάλληλη πράξη που θα εκτελέσει η ALU.
- **ImmSrc** (1 bit): Επιλέγει επέκταση πρόσημου ή επέκταση μηδενός.
- **IRWrite** (1 bit): Επιτρέπει την εγγραφή στον Instruction Register.
- **RegWrite** (1 bit): Επιτρέπει την εγγραφή στον Register File.
- **MAWrite** (1 bit): Επιτρέπει την εγγραφή στον Memory Address Register.
- **MemWrite** (1 bit): Επιτρέπει την εγγραφή στην RAM.
- **FlagsWrite** (1 bit): Επιτρέπει την ανανέωση των σημαιών συνθήκης στον status register με βάση το αποτέλεσμα της ALU.
- **PCSrc** (2 bits): Καθορίζει την προέλευση της επόμενης τιμής του PC.
- **PCWrite** (1 bit): Επιτρέπει την ανανέωση του PC.

Το Control Unit (Μονάδα Ελέγχου) του επεξεργαστή παρουσιάζεται στο σχηματικό διάγραμμα παρακάτω και αποτελείται από από τρία βασικά στοιχεία, τον Instruction Decoder (Αποκωδικοποιητή Εντολών), το Conditional Logic (Λογική Συνθηκών) και το Finite State Machine (Μηχανή Πεπερασμένων Καταστάσεων). Το τμήμα του Instruction Decoder παράγει τα σήματα ελέγχου του Datapath, τα οποία

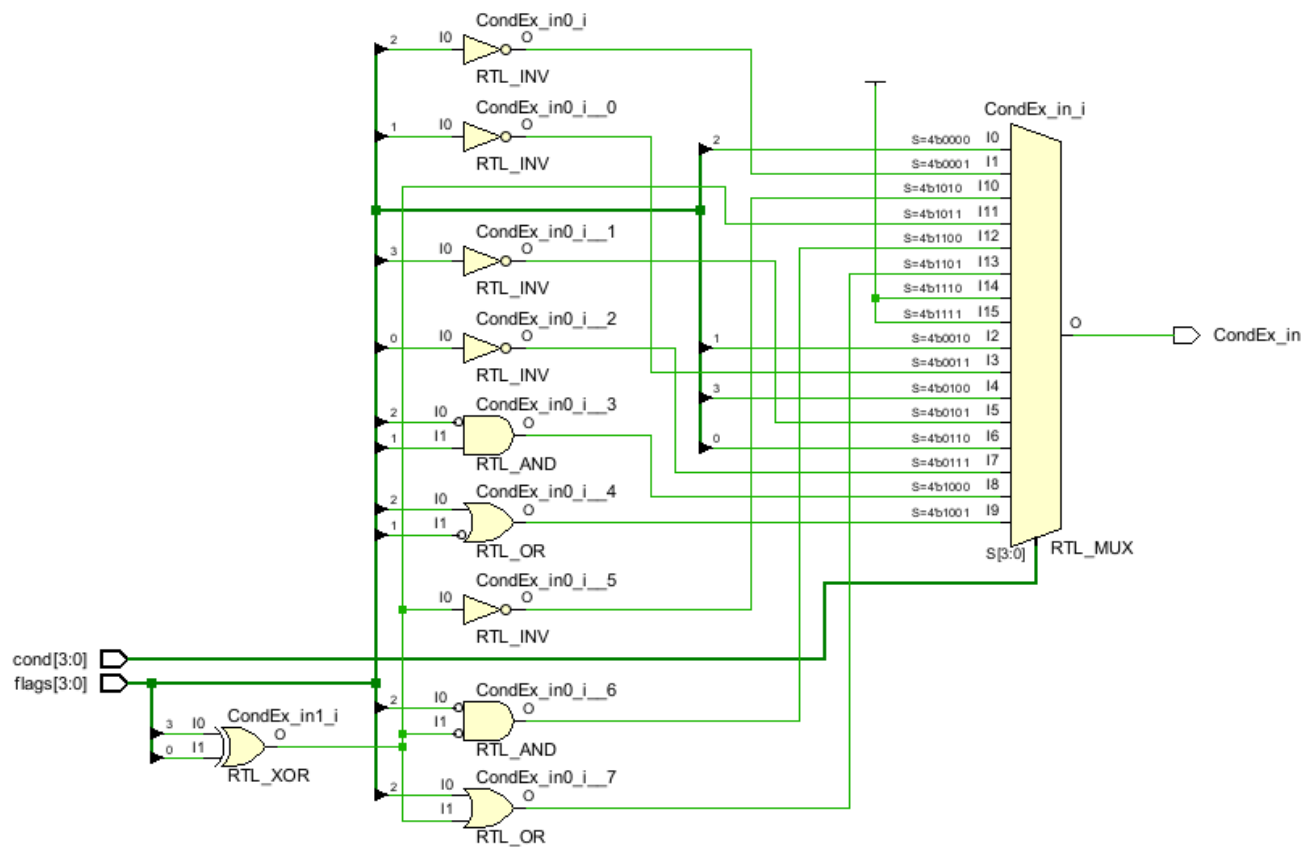
παραμένουν σταθερά καθ' όλη τη διάρκεια εκτέλεσης μιας εντολής και κυρίως αφορούν σήματα επιλογής πολυπλεκτών. Το FSM δημιουργεί σήματα που μεταβάλλονται σε κάθε κύκλο και συνήθως αφορούν τη δυνατότητα εγγραφής σε καταχωρητές και μνήμες. Το τμήμα Conditional Logic παράγει μόνο ένα εσωτερικό σήμα, το οποίο εισέρχεται ως είσοδος στο FSM. Τα στοιχεία Instruction Decoder και Conditional Logic είναι συνδυαστικά κυκλώματα, καθώς αποτελούνται μόνο από πολυπλέκτες μαζί με μνήμες ROM και πολυπλέκτες μαζί με λογικές πύλες αντίστοιχα, και παράγουν εξόδους που εξαρτώνται από τις τρέχουσες εισόδους τους. Το Finite State Machine, από την άλλη, είναι ένα ακολουθιακό σύγχρονο κύκλωμα το οποίο ρυθμίζει τον ρυθμό εκτέλεσης των εντολών. Συγκεκριμένα, αποτελείται από ένα συνδυαστικό κύκλωμα οι έξοδοι του οποίου εισέρχονται σε έναν πολυπλέκτη και στην συνέχεια φτάνει εισέρχεται το σήμα στον Status Register. Η έξοδος του Status Register συνδέεται σε μνήμες ROM μνήμες οι οποίες προσδιορίζουν τις τιμές των εξόδων του Finite State Machine. Ο Instruction Decoder και το Conditional Logic συνδέονται μέσω των εσωτερικών σημάτων NoWrite_in και CondEx_in με το Finite State Machine.



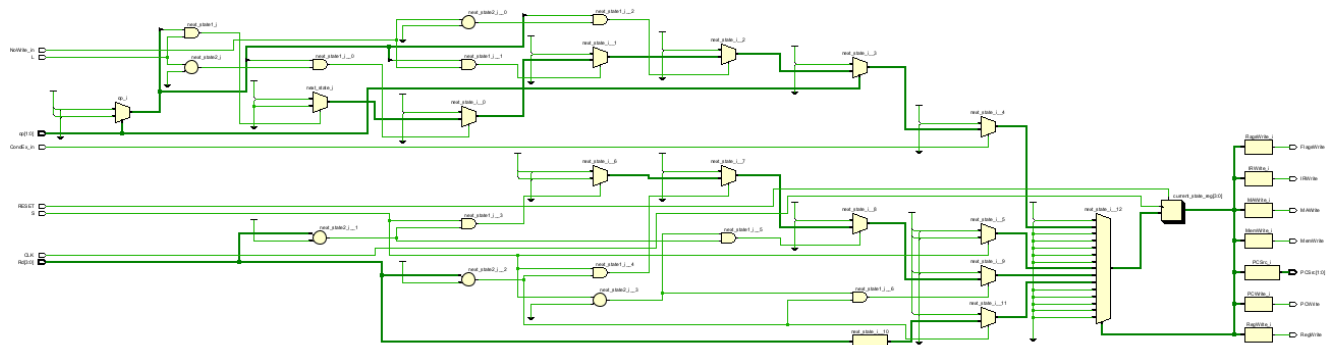
Εικόνα 1.12 RTL Schematic της μονάδας ελέγχου (Control Unit).



Εικόνα 1.13 RTL Schematic του Instruction Decoder.



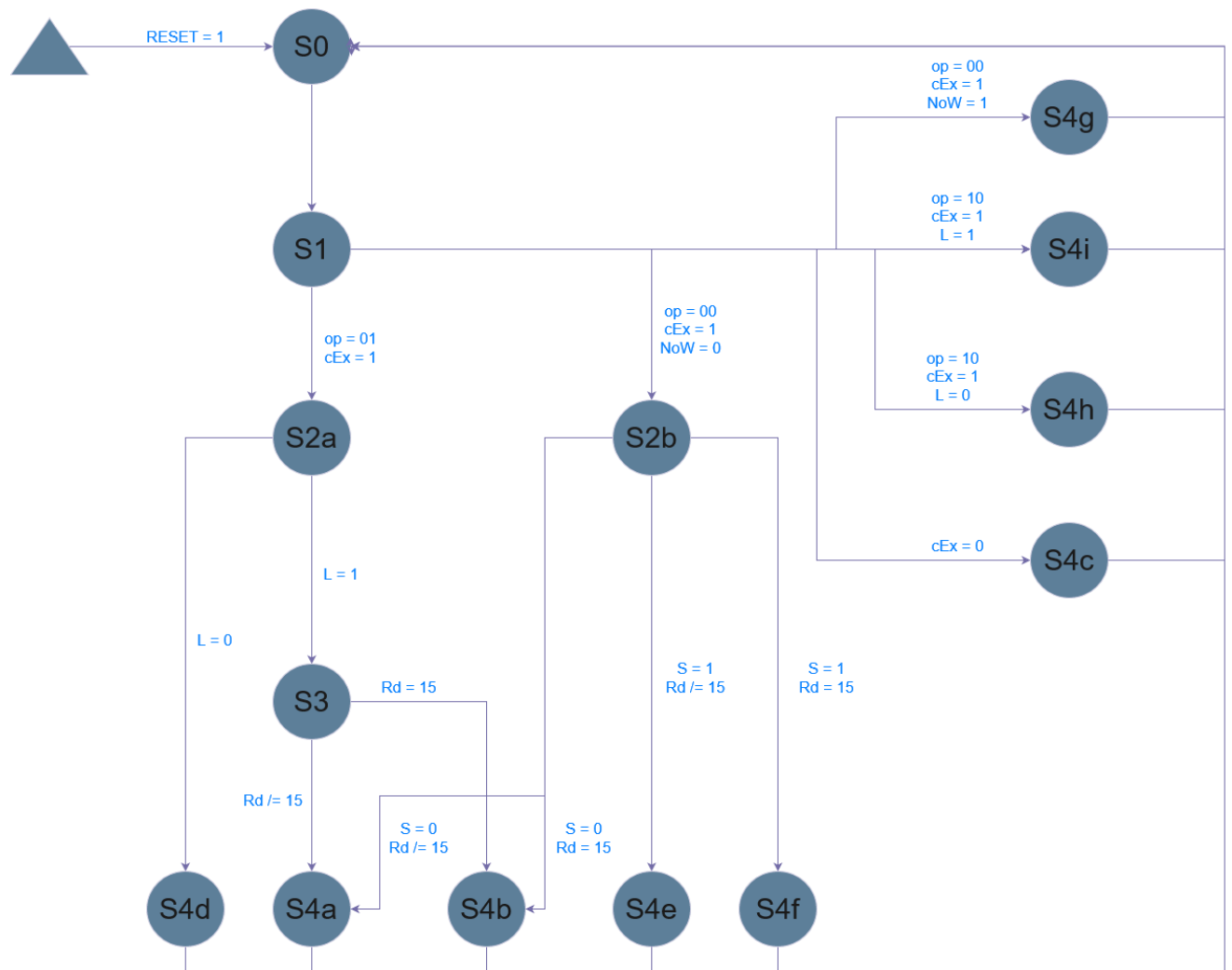
Εικόνα 1.14 RTL Schematic του Conditional Logic.



Εικόνα 1.15 RTL Schematic του Finite State Machine.

1.5. Μηχανή πεπερασμένων καταστάσεων (FSM) του Control Unit

Η Μηχανή Πεπερασμένων Καταστάσεων που υλοποιήθηκε για τον επεξεργαστή πολλών κύκλων αρχιτεκτονικής ARM είναι τύπου Moore. Στις μηχανές αυτού του είδους τα σήματα εξόδου εξαρτώνται μόνο από την κατάσταση που βρίσκεται το σύστημα κάθε στιγμή και όχι τις εισόδους του συστήματος εκείνη τη στιγμή. Κάθε τύπος εντολής απαιτεί έναν διαφορετικό αριθμό εκτέλεσης βημάτων. Τα βήματα αντιστοιχίζονται σε καταστάσεις και κατά την ανοδική ακμή του σήματος του ρολογιού η επόμενη κατάσταση προσδιορίζεται βάσει της προηγούμενης και των σημάτων εισόδου της Μονάδας Ελέγχου. Παρακάτω παρουσιάζεται το διάγραμμα μεταβολής κατάστασης και προσδιορίζονται οι συνθήκες εισόδου που καθορίζουν τη μετάβαση από τη μία κατάσταση στην άλλη.



Εικόνα 1.16 Διάγραμμα μεταβολής κατάστασης.

Ας εξετάσουμε τώρα τι κάνει η κάθε κατάσταση.

- **S0** -> Προσκομίζεται η εντολή από την Μνήμη Εντολών και αποθηκεύεται στον Instruction Register.
- **S1** -> Αποκωδικοποιείται η εντολή από την Μονάδα Ελέγχου και παράγονται τα σήματα ανάγνωσης του Register File.
- **S2a** -> Υπολογίζεται η διεύθυνση ανάγνωσης της Μνήμης Δεδομένων στην ALU και εγγράφεται στον Memory Address Register.
- **S2b** -> Εκτελείται η πράξη στην ALU.
- **S3** -> Διαβάζονται από την Μνήμη τα δεδομένα προς εγγραφή στον Register File.
- **S4a** -> Εγγράφονται στον Register File τα ανακτημένα δεδομένα από τη μνήμη, ενώ ταυτόχρονα εγγράφεται η διεύθυνση της επόμενης εντολής στον PC.
- **S4b** -> Εγγράφεται διεύθυνσης επόμενης εντολής στον PC.
- **S4c** -> Τερματίζεται πρόωρα η εκτέλεση της εντολής.
- **S4d** -> Εγγράφονται τα δεδομένα στην Μνήμη Δεδομένων και η διεύθυνση επόμενης εντολής στον PC.
- **S4e** -> Εγγράφονται στον Register File τα ανακτημένα δεδομένα από τη μνήμη και ενημερώνεται ο Status Register, ενώ ταυτόχρονα εγγράφεται η διεύθυνση της επόμενης εντολής στον PC.
- **S4f** -> Ενημερώνεται ο Status Register, ενώ ταυτόχρονα εγγράφεται η διεύθυνση της επόμενης εντολής στον PC.
- **S4g** -> Ενημερώνεται ο Status Register και επιλέγεται η διεύθυνση της επόμενης εντολής.
- **S4h** -> Υπολογίζεται η Branch Target Address στην ALU και εγγράφεται στον PC.
- **S4i** -> Υπολογίζεται η Branch Target Address στην ALU και εγγράφεται στον PC, ενώ ταυτόχρονα εγγράφεται η διεύθυνση επιστροφής στον R14.

Τώρα θα εξετάσουμε τα βήματα (κύκλους) εκτέλεσης μέσα από τους πίνακες κατάστασης κάθε εντολής που υλοποιήθηκε στη μικροαρχιτεκτονική ARM στον επεξεργαστή.

Not executed															
Βήμα	Current State	Next State	Instr				NoWrite_in	Condex_in	IRWrite	RegWrite	MAWrite	MemWrite	FlagsWrite	PCSrc	PCWrite
			op27:26	S/L20	L24	Rd15:12									
1	S0	S1	XX	X	X	XXXX	X	X	1	0	0	0	0	00	0
2	S1	S4c	XX	X	X	XXXX	X	0	0	0	0	0	0	00	0
3	S4c	S0	XX	X	X	XXXX	X	X	0	0	0	0	0	00	1

Ο πίνακας καταστάσεων αυτός αφορά την περίπτωση που το Conditional Logic ελέγχει το περιεχόμενο του Status Register και το μνημονικό της εντολής και καταλήγει ότι η εντολή δεν πρέπει να εκτελεστεί. Η προσπέραση της εντολής γίνεται σε τρία βήματα. Στο πρώτο βήμα είμαστε στην κατάσταση S0 όπου εγγράφεται η εντολή στον Instruction Register, για αυτό και IRWrite = '1'. Στο δεύτερο βήμα πηγαίνουμε στην κατάσταση S1 όπου αποκωδικοποιείται η εντολή από την Μονάδα Ελέγχου και παράγονται τα σήματα ανάγνωσης του Register File. Όπως θα παρατηρήσουμε και στην συνέχεια η εκκίνηση από την κατάσταση S0 και η

μετάβαση στην κατάσταση S1 γίνεται σε όλες τις εντολές που υλοποιούνται. Στην κατάσταση S1, υπολογίζεται η επόμενη κατάσταση από τα σήματα εισόδου. Στην περίπτωση του παραπάνω πίνακα στην κατάσταση S1 το σήμα CondEx_in έχει τιμή '0', οπότε η επόμενη κατάσταση είναι η S4c, στην οποία τερματίζεται πρόωρα η εκτέλεση της εντολής. Στην κατάσταση S4c το σήμα PCWrite παίρνει τιμή '1', ενώ για το σήμα PCSrc ισχύει PCSrc = '00'. Η επόμενη κατάσταση θα είναι η αρχική κατάσταση S0. Η S0 πάντα θα ακολουθεί την τελευταία κατάσταση κάποιας εντολής. Για να μην διέλθει αμέσως το σύστημα από τις καταστάσεις του παραπάνω πίνακα πρέπει να ισχύει CondEx_in = '1'.

LDR															
Βήμα	Current State	Next State	Instr				NoWrite_in	Condex_in	IRWrite	RegWrite	MAWrite	MemWrite	FlagsWrite	PCSrc	PCWrite
			op27:26	S/L20	L24	Rd15:12									
1	S0	S1	XX	X	X	XXXX	X	X	1	0	0	0	0	00	0
2	S1	S2a	01	X	X	XXXX	X	1	0	0	0	0	0	00	0
2	S1	S4c	XX	X	X	XXXX	X	0	0	0	0	0	0	00	0
3	S2a	S3	XX	1	X	XXXX	X	X	0	0	1	0	0	00	0
4	S3	S4a	XX	X	X	not 1111	X	X	0	0	0	0	0	00	0
4	S3	S4b	XX	X	X	1111	X	X	0	0	0	0	0	00	0
5	S4a	S0	XX	X	X	XXXX	X	X	0	1	0	0	0	00	1
5	S4b	S0	XX	X	X	XXXX	X	X	0	0	0	0	0	10	1
5	S4c	S0	XX	X	X	XXXX	X	X	0	0	0	0	0	00	1

Η εντολή LDR εκτελείται σε πέντε βήματα με βάση τον πίνακα καταστάσεων της που παρουσιάζεται από πάνω. Τα δύο πρώτα βήματα είναι οι κοινές καταστάσεις S0 και S1 που περιεγράφηκαν προηγουμένως. Η τρίτη κατάσταση είναι η S2a, η οποία είναι κοινή για όλες τις εντολές επεξεργασίας μνήμης. Για να πάει το σύστημα στην κατάσταση S2a πρέπει να ισχύει op = '01' και CondEx_in = '1' στην κατάσταση S1. Εκτός από την S2a, το σύστημα μετά την S1 μπορεί να πάει στην κατάσταση S4c αν το σήμα CondEx_in = '0' και να τερματιστεί πρόωρα η εκτέλεση της εντολής, όπως αναφέρθηκε παραπάνω. Γενικά από δω και κάτω στην εξήγηση των πινάκων καταστάσεων θα θεωρούμε ότι ισχύει CondEx_in = '1' και δεν θα τερματίζεται πρόωρα η εντολή. Στην κατάσταση S2a ενεργοποιείται το σήμα MAWrite και παίρνει τιμή '1', ενώ αν στη συνέχεια για το bit 20 του σήματος της εντολής προς εκτέλεση, δηλαδή πεδίο L, ισχύει L = '1', τότε πρόκειται για την εντολή LDR και πηγαίνουμε στην κατάσταση S3, στην οποία διαβάζονται από την μνήμη τα δεδομένα προς εγγραφή στον Register File. Η επόμενη κατάσταση εξαρτάται από το αν η εγγραφή πρέπει να γίνει στον PC ή σε κάποιον άλλο καταχωρητή του Register File. Ο προορισμός της εγγραφής εξαρτάται από το πεδίο Rd. Αν το πεδίο Rd είχε οποιαδήποτε άλλη τιμή εκτός της '1111' τότε η τελική κατάσταση είναι η S4a κατά την οποία ενεργοποιείται το σήμα PCWrite = '1', ενώ η τιμή του PCSrc είναι PCSrc = '00', δηλαδή η πηγή του PC θα είναι η διεύθυνση επόμενης εντολής PC+4. Ταυτόχρονα, ενεργοποιείται το σήμα RegWrite = '1' για εγγραφή στον Register File των δεδομένων που ανακτήθηκαν από τη Μνήμη Δεδομένων. Αντίθετα, αν το πεδίο αυτό έχει την τιμή Rd = '1111' τότε ο καταχωρητής προορισμού είναι ο R15, οπότε η τελική κατάσταση είναι η S4b. Στην κατάσταση αυτή γίνεται επίσης εγγραφή στον PC (PCWrite = '1') και η πηγή του PC θα είναι η έξοδος της Μνήμης Δεδομένων και το σήμα PCSrc θα έχει τιμή PCSrc = '10'.

STR															
Βήμα	Current State	Next State	Instr				NoWrite_in	Condex_in	IRWrite	RegWrite	MAWrite	MemWrite	FlagsWrite	PCSrc	PCWrite
			op27:26	S/L20	L24	Rd15:12									
1	S0	S1	XX	X	X	XXXX	X	X	1	0	0	0	0	00	0
2	S1	S2a	01	X	X	XXXX	X	1	0	0	0	0	0	00	0
2	S1	S4c	XX	X	X	XXXX	X	0	0	0	0	0	0	00	0
3	S2a	S4d	XX	0	X	XXXX	X	X	0	0	1	0	0	00	0
5	S4c	S0	XX	X	X	XXXX	X	X	0	0	0	0	0	00	1
4,5	S4d	S0	XX	X	X	XXXX	X	X	0	0	0	1	0	00	1

Ο πίνακας κατάστασης της εντολής STR παρουσιάζεται από πάνω. Τα τρία πρώτα βήματα είναι οι κοινές καταστάσεις S0, S1 και S2a που περιεγράφηκαν προηγουμένως για την εντολή LDR, καθώς για τις εντολές μνήμης ισχύει ότι op = '01'. Αντίθετα με την LDR, όμως για την εντολή STR ισχύει L = '0' και η τέταρτη κατάσταση είναι η S4d. Σε αυτή την κατάσταση ενεργοποιείται το σήμα MemWrite = '1' και το σήμα PCWrite = '1', ενώ ισχύει PCSrc = '00', άρα η πηγή του PC είναι η επόμενη κατά σειρά εντολή.

Data Processing (except CMP)															
Βήμα	Current State	Next State	Instr				NoWrite_in	Condex_in	IRWrite	RegWrite	MAWrite	MemWrite	FlagsWrite	PCSrc	PCWrite
			op27:26	S/L20	L24	Rd15:12									
1	S0	S1	XX	X	X	XXXX	X	X	1	0	0	0	0	00	0
2	S1	S2b	00	X	X	XXXX	0	1	0	0	0	0	0	00	0
2	S1	S4c	XX	X	X	XXXX	X	0	0	0	0	0	0	00	0
3	S2b	S4a	XX	0	X	not 1111	X	X	0	0	0	0	0	00	0
3	S2b	S4b	XX	0	X	1111	X	X	0	0	0	0	0	00	0
3	S2b	S4e	XX	1	X	not 1111	X	X	0	0	0	0	0	00	0
3	S2b	S4f	XX	1	X	1111	X	X	0	0	0	0	0	00	0
5	S4a	S0	XX	X	X	XXXX	X	X	0	1	0	0	0	00	1
5	S4b	S0	XX	X	X	XXXX	X	X	0	0	0	0	0	10	1
5	S4c	S0	XX	X	X	XXXX	X	X	0	0	0	0	0	00	1
5	S4e	S0	XX	X	X	XXXX	X	X	0	1	0	0	1	00	1
5	S4f	S0	XX	X	X	XXXX	X	X	0	0	0	0	1	10	1

Ο παραπάνω πίνακας καταστάσεων αναφέρεται στις εντολές επεξεργασίας δεδομένων, εκτός από την εντολή CMP. Τα δύο πρώτα βήματα είναι οι κοινές καταστάσεις S0 και S1 που περιεγράφηκαν προηγουμένως. Μετά την S1, αν ισχύει ότι op = '00', NoWrite_in = '0' και CondEx_in = '1', τότε ακολουθεί η κατάσταση S2b. Η κατάσταση αυτή αντιστοιχεί στην εκτέλεση πράξης στην ALU για όλες τις εντολές επεξεργασίας δεδομένων. Το σύστημα μετά την κατάσταση S2b μπορεί να προχωρήσει σε 5 διαφορετικές καταστάσεις ανάλογα με τις τιμές των πεδίων S και Rd. Αν S = '0' και Rd διάφορο της τιμής '1111' τότε το σύστημα θα μεταβεί στην τελική κατάσταση S4a, στην οποία εγγράφονται στον Register File τα ανακτημένα δεδομένα από τη μνήμη, ενώ ενεργοποιείται το σήμα RegWrite = '1' και το σήμα PCWrite = '1', ενώ PCSrc = '00' οπότε η διεύθυνση της επόμενης κατά σειρά εντολής που θα εγγραφεί στον PC είναι η PC+4. Αν S = '0' και Rd = '1111' τότε το σύστημα θα μεταβεί στην τελική κατάσταση S4b. Στην κατάσταση αυτή ο καταχωρητής προορισμού είναι ο R15, ενώ ισχύει PCWrite = '1', ενώ PCSrc = '10' οπότε η διεύθυνση της επόμενης κατά σειρά εντολής που θα εγγραφεί στον PC είναι το αποτέλεσμα της ALU. Αν S = '1' και Rd διάφορο της τιμής '1111' τότε το σύστημα θα μεταβεί στην τελική κατάσταση S4e, στην οποία εγγράφονται στον Register File τα ανακτημένα δεδομένα από τη μνήμη και ενημερώνεται ο Status Register, ενώ

ενεργοποιείται το σήμα RegWrite = '1', το σήμα FlagsWrite = '1', και το σήμα PCWrite = '1', ενώ PCSrc = '00' οπότε η διεύθυνση της επόμενης κατά σειρά εντολής που θα εγγραφεί στον PC είναι η PC+4. Αν S = '1' και Rd = '1111' τότε το σύστημα θα μεταβεί στην τελική κατάσταση S4f. Στην κατάσταση αυτή ενημερώνεται ο Status Register και ο καταχωρητής προορισμού είναι ο R15, ενώ ισχύει PCWrite = '1' και FlagsWrite = '1', ενώ PCSrc = '10' οπότε η διεύθυνση της επόμενης κατά σειρά εντολής που θα εγγραφεί στον PC είναι το αποτέλεσμα της ALU.

CMP															
Βήμα	Current State	Next State	Instr				NoWrite_in	Condex_in	IRWrite	RegWrite	MAWrite	MemWrite	FlagsWrite	PCSrc	PCWrite
			op27:26	S/L20	L24	Rd15:12									
1	S0	S1	XX	X	X	XXXX	X	X	1	0	0	0	0	00	0
2	S1	S4g	00	X	X	XXXX	1	1	0	0	0	0	0	00	0
2	S1	S4c	XX	X	X	XXXX	X	0	0	0	0	0	0	00	0
5	S4c	S0	XX	X	X	XXXX	X	X	0	0	0	0	0	00	1
3,5	S4g	S0	XX	X	X	XXXX	X	X	0	0	0	0	1	00	1

Ο παραπάνω πίνακας αναφέρεται στην εντολή επεξεργασίας δεδομένων CMP. Τα δύο πρώτα βήματα είναι οι κοινές καταστάσεις S0 και S1 που περιεγράφηκαν προηγουμένως. Στην κατάσταση S1 ενεργοποιούνται τα σήματα NoWrite_in = '1' και CondEx_in = '1', ενώ το πεδίο op έχει τιμή op = '00'. Η επόμενη κατάσταση είναι η S4g, στην οποία ενημερώνεται ο Status Register με τις σημαίες που παράγονται στην ALU αφού ενεργοποιείται το σήμα FlagsWrite = '1'. Επίσης το σήμα PCWrite = '1', ενώ PCSrc = '00' οπότε η διεύθυνση της επόμενης κατά σειρά εντολής που θα εγγραφεί στον PC είναι η PC+4.

B															
Βήμα	Current State	Next State	Instr				NoWrite_in	Condex_in	IRWrite	RegWrite	MAWrite	MemWrite	FlagsWrite	PCSrc	PCWrite
			op27:26	S/L20	L24	Rd15:12									
1	S0	S1	XX	X	X	XXXX	X	X	1	0	0	0	0	00	0
2	S1	S4h	10	X	0	XXXX	X	1	0	0	0	0	0	00	0
2	S1	S4c	XX	X	X	XXXX	X	0	0	0	0	0	0	00	0
5	S4c	S0	XX	X	X	XXXX	X	X	0	0	0	0	0	00	1
3,5	S4h	S0	XX	X	X	XXXX	X	X	0	0	0	0	0	11	1

Από τον πίνακα κατάστασης της εντολής διακλάδωσης B βλέπουμε ότι τα δύο πρώτα βήματα είναι οι κοινές καταστάσεις S0 και S1 που περιεγράφηκαν προηγουμένως. Οι εντολές διακλάδωσης προσδιορίζονται από το πεδίο op = '10', ενώ η εντολή B προσδιορίζεται από το 24ο bit του σήματος εντολής Instr, δηλαδή το πεδίο L, το οποίο έχει τιμή '0'. Η επόμενη κατάσταση είναι η S4h στην οποία υπολογίζεται η Branch Target Address στην ALU και εγγράφεται στον PC, αφού ισχύει PCWrite = '1', ενώ η πηγή του PC είναι το αποτέλεσμα της ALU αφού PCSrc = '11'.

BL															
Βήμα	Current State	Next State	Instr				NoWrite_in	Condex_in	IRWrite	RegWrite	MAWrite	MemWrite	FlagsWrite	PCSrc	PCWrite
			op27:26	S/L20	L24	Rd15:12									
1	S0	S1	XX	X	X	XXXX	X	X	1	0	0	0	0	00	0
2	S1	S4i	10	X	1	XXXX	X	1	0	0	0	0	0	00	0
2	S1	S4c	XX	X	X	XXXX	X	0	0	0	0	0	0	00	0
5	S4c	S0	XX	X	X	XXXX	X	X	0	0	0	0	0	00	1
3,5	S4i	S0	XX	X	X	XXXX	X	X	0	1	0	0	0	11	1

Από τον πίνακα κατάστασης της εντολής διακλάδωσης BL βλέπουμε ότι τα δύο πρώτα βήματα είναι οι κοινές καταστάσεις S0 και S1 που περιεγράφηκαν προηγουμένως. Οι εντολές διακλάδωσης προσδιορίζονται από το πεδίο op = '10', ενώ η εντολή BL προσδιορίζεται από το 24ο bit του σήματος εντολής Instr, δηλαδή το πεδίο L, το οποίο έχει τιμή '1', αντί για '0' που ήταν για την εντολή B. Η επόμενη κατάσταση είναι η S4i στην οποία υπολογίζεται η Branch Target Address στην ALU και εγγράφεται στον PC, αφού ισχύει PCWrite = '1', ενώ η πηγή του PC είναι το αποτέλεσμα της ALU αφού PCSrc = '11', ενώ επιπλέον ισχύει RegWrite = '1' και γίνεται εγγραφή στο Register File στον Link Register R14 η διεύθυνση επιστροφής,

1.6. Σχηματικό διάγραμμα στο επίπεδο RTL του επεξεργαστή

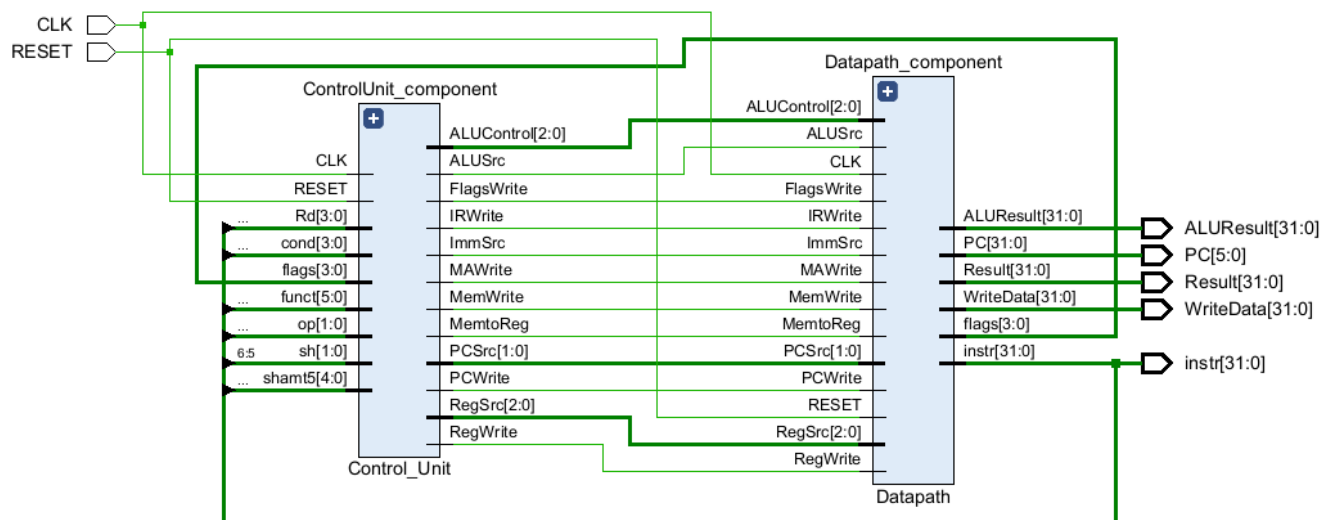
Τα σήματα εισόδου που δέχεται ο processor είναι:

- **CLK** (1 bit): Το σήμα ρολογιού του γενικού κυκλώματος, καθορίζει τη συχνότητα λειτουργίας του επεξεργαστή.
- **RESET** (1 bit): Το σήμα reset του γενικού κυκλώματος, επαναφέρει τον επεξεργαστή στην αρχική του κατάσταση.

Τα σήματα εξόδου του processor είναι:

- **PC** (6 bits): Η έξοδος του PC Register.
- **instr** (32 bits): Η έξοδος της μνήμης εντολών ROM που δείχνει την τωρινή εντολή.
- **ALUResult** (32 bits): Αποτέλεσμα της πράξης της ALU.
- **WriteData** (32 bits): Δεδομένα που θα γραφτούν στην RAM.
- **Result** (32 bits): Δεδομένα που θα γραφτούν στο αρχείο καταχωρητών.

Ο επεξεργαστής πολλών κύκλων αρχιτεκτονικής ARM αποτελείται από τη Διαδρομή Δεδομένων (Datapath) και τη Μονάδα Ελέγχου (Control Unit). Οι εντολές αποθηκεύονται στη Μνήμη Εντολών ROM, ενώ τα components που αποτελούν το Datapath εξασφαλίζουν τη βηματική εκτέλεση των εντολών που αναλύθηκαν παραπάνω. Το Control Unit παίρνει ως σήμα εισόδου την εντολή που πρόκειται να εκτελεστεί και το σήμα flags του Status Register και παράγει σήματα ελέγχου που καθορίζουν τι θα εγγραφεί στον Register File και τι πράξη θα εκτελέσει η ALU.



Εικόνα 1.17 RTL Schematic του Processor πολλών κύκλων.

1.7. Μέγιστη συχνότητα λειτουργίας και κρίσιμες διαδρομές

Το implementation του VIVADO πραγματοποιεί την αναλυτικότερη και πιο ρεαλιστική χρονική ανάλυση του κυκλώματος του επεξεργαστή πολλών κύκλων αρχιτεκτονικής ARM. Τα timing reports που προκύπτουν μετά το implementation χρησιμοποιούνται για να καθορίσουμε τους χρονικούς περιορισμούς της περιόδου του ρολογιού που επιθυμούμε για την υλοποίηση του επεξεργαστή, οι οποίοι υπάρχουν και μέσα στο αρχείο των constraints, δηλαδή το αρχείο zedboard.xdc. Σκοπός είναι να βρούμε την μέγιστη συχνότητα λειτουργίας του κυκλώματος, δηλαδή την ελάχιστη περίοδο ρολογιού που μπορούμε να ορίσουμε για τον επεξεργαστή για την οποία να μην παρουσιάζονται χρονικές παραβιάσεις, αλλά και την χειρότερη κρίσιμη διαδρομή και την χειρότερη σύντομη διαδρομή για αυτήν την συχνότητα λειτουργίας.

Η μέγιστη συχνότητα λειτουργίας που βρέθηκε είναι τα 135,465 MHz, η οποία αντιστοιχεί σε περίοδο ρολογιού 7,382 ns.

Name	Waveform	Period (ns)	Frequency (MHz)
CLK	{0.000 3.691}	7.382	135.465

Εικόνα 1.18 Μέγιστη συχνότητα λειτουργίας.

Για εύρεση της κρίσιμης διαδρομής, δηλαδή της διαδρομής με την μεγαλύτερη καθυστέρηση διάδοσης, και συνεπώς στην εύρεση της μέγιστης συχνότητας λειτουργίας πρέπει να μην παραβιάζεται ο χρόνος σταθεροποίησης (setup time). Ο χρόνος σταθεροποίησης είναι το χρονικό διάστημα που το σήμα εισόδου δεδομένων πρέπει να παραμείνει σταθερό και να μην αλλάξει πριν από την ακμή του ρολογιού. Εξασφαλίζει ότι το σήμα εισόδου δεδομένων έχει επαρκή χρόνο να σταθεροποιηθεί και να γίνει σωστά αντιληπτό από το κύκλωμα πριν από την αποθήκευσή του.

Για εύρεση της σύντομης διαδρομής πρέπει να μην παραβιάζεται ο χρόνος διατήρησης (hold time). Ο χρόνος διατήρησης είναι το ελάχιστο χρονικό διάστημα που το σήμα εισόδου δεδομένων πρέπει να παραμείνει σταθερό και να μην αλλάξει μετά από την ακμή του ρολογιού που ενεργοποιεί την αποθήκευση αυτών των δεδομένων. Εξασφαλίζει ότι το σήμα εισόδου δεδομένων έχει επαρκή χρόνο να αποθηκευτεί σωστά στο κύκλωμα μετά την ακμή του ρολογιού.

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 0,033 ns	Worst Hold Slack (WHS): 0,060 ns	Worst Pulse Width Slack (WPWS): 2,441 ns
Total Negative Slack (TNS): 0,000 ns	Total Hold Slack (THS): 0,000 ns	Total Pulse Width Negative Slack (TPWS): 0,000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 1007	Total Number of Endpoints: 1007	Total Number of Endpoints: 433

All user specified timing constraints are met.

Εικόνα 1.19 Ανάλυση χρονισμού στο *implemented design model* για περίοδο ρολογιού 7,382 ns.

Για την στήλη Setup το Worst Negative Slack (WNS) είναι μια τιμή (με link) που αντιστοιχεί στο μικρότερο διαθέσιμο περιθώριο (slack) που προκύπτει από την ανάλυση όλων των κρίσιμων διαδρομών. Μπορεί να είναι θετικό ή αρνητικό. Ένα θετικό slack δηλώνει ότι η κρίσιμη διαδρομή ικανοποιεί την περίοδο του ρολογιού, ενώ ένα αρνητικό slack δηλώνει ότι παραβιάζεται ο χρόνος σταθεροποίησης (setup time). Το Total Negative Slack (TNS) είναι το άθροισμα όλων των αρνητικών WNS για κάθε timing path endpoint. Η τιμή είναι 0.000 ns όταν δεν υπάρχουν παραβιάσεις του χρόνου σταθεροποίησης κατά την λειτουργία του κυκλώματος στην επιλεγμένη συχνότητα λειτουργίας. Το Number of Failing Endpoints αφορά στον συνολικό αριθμό των endpoints που έχουν παραβιάσει το χρόνο σταθεροποίησης (αρνητικό WNS). Το Total Number of Endpoints αφορά στον συνολικό αριθμό των endpoints που έχουν αναλυθεί.

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

Για την εύρεση της μέγιστης συχνότητας λειτουργίας τέθηκε αρχικά μία περίοδος ρολογιού ίση με 10ns, ενώ χρησιμοποιήθηκαν και τα IOB του FPGA. Καθώς τα WNS και WHS είχαν αρκετά μεγάλο χρονικό περιθώριο, μειώθηκε η τιμή της περιόδου ρολογιού κατά WNS. Η διαδικασία αυτή επαναλήφθηκε αρκετές φορές μέχρι να παραβιαστεί είτε το Setup ή το Hold time. Μετά από πολλές δοκιμές βρέθηκε η

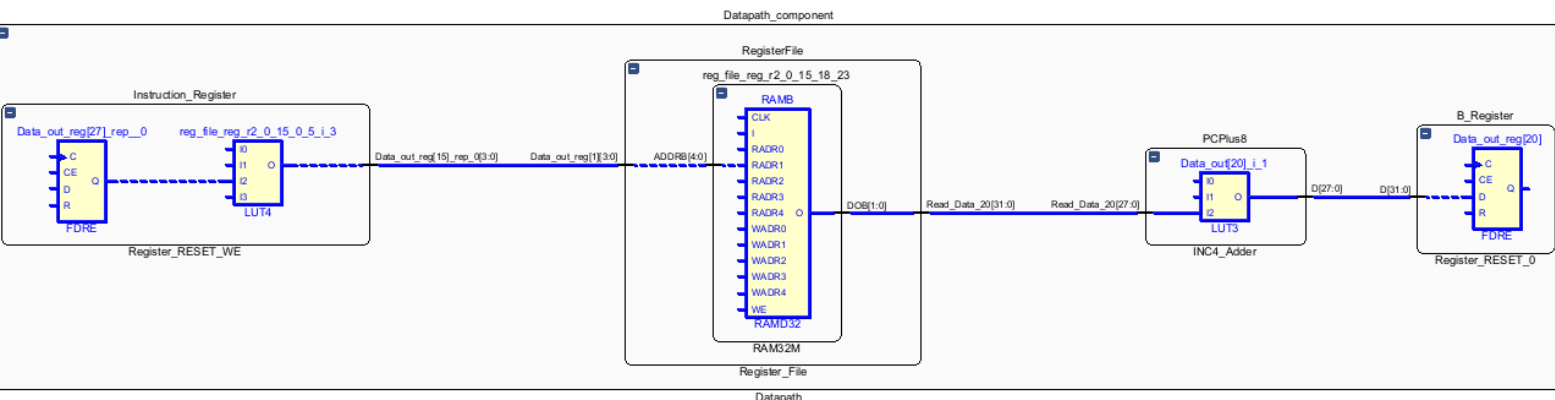
μέγιστη συχνότητα λειτουργίας στα 135,465 MHz, η οποία αντιστοιχεί σε περίοδο ρολογιού 7,382 ns. Μείωση της περιόδου του ρολογιού ακόμα και κατά 0,001 ns έχει ως αποτέλεσμα αρνητικό WNS.

Στην εικόνα 1.20 παρουσιάζεται η χειρότερη κρίσιμη διαδρομή που δεν παραβιάζει τον χρόνο σταθεροποίησης. Η κρίσιμη διαδρομή έχει καθυστέρηση διάδοσης 6,473 ns, εκ των οποίων τα 1,332 ns αφορούν στη λογική (logic), ενώ τα 5,141 ns αφορούν στη δικτύωση (net).

Name	Slack ^{^1}	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	0.033	3	123	Datapath_compo...[27]_rep__0/C	Datapath_comp...out_reg[20]/D	6.473	1.332	5.141

Εικόνα 1.20 Χειρότερη κρίσιμη διαδρομή.

Στην εικόνα 1.21 εμφανίζεται το Path 1 του Timing Report, δηλαδή η χειρότερη κρίσιμη διαδρομή. Η κρίσιμη διαδρομή αποτελείται από 3 λογικά επίπεδα και περνάει από 1 block RAMD32, 1 LUT4 και 1 LUT3. Το μονοπάτι αυτό είναι μια σύνδεση του Instruction Register σε ένα port του Register File, από εκεί σε ένα port του PCPlus8 και τέλος σε ένα port του B Register.



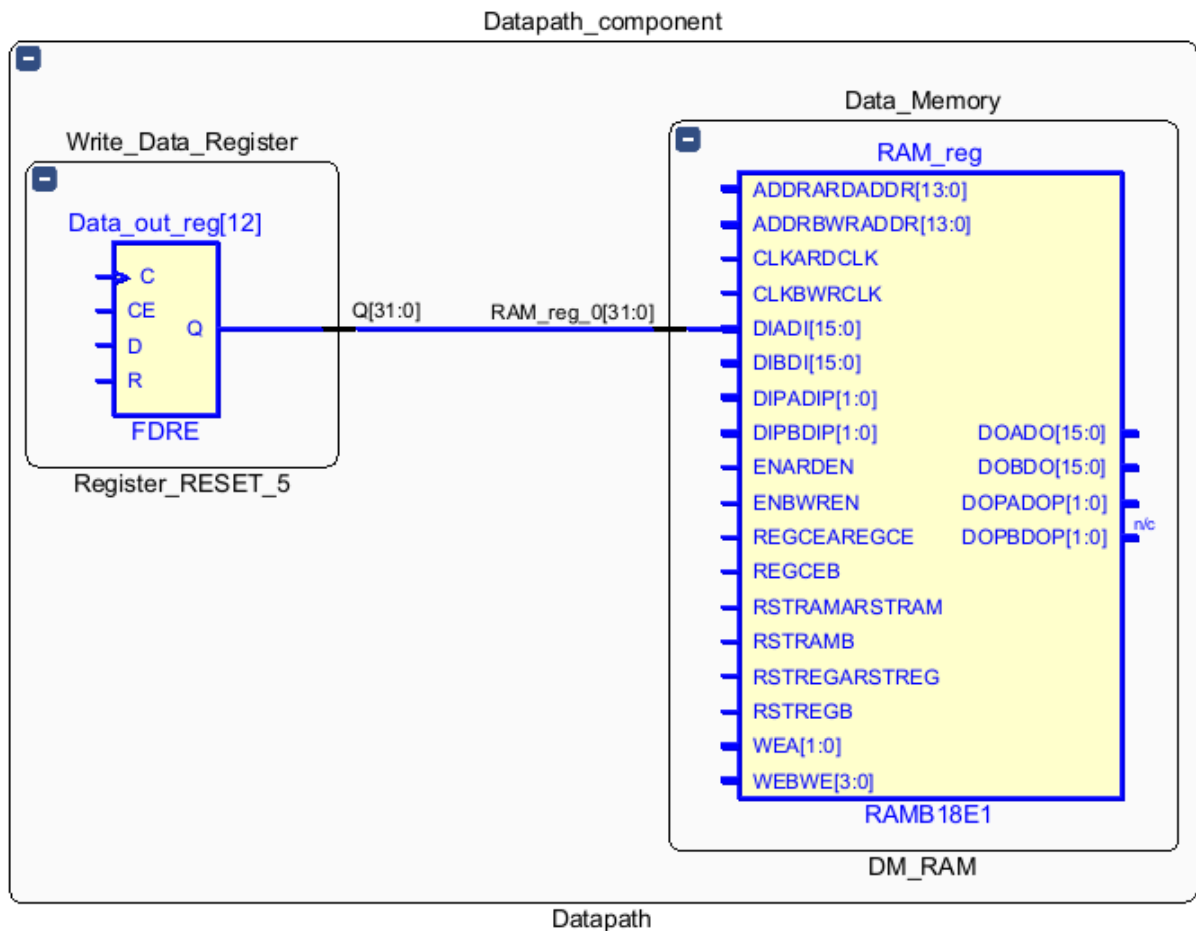
Εικόνα 1.21 Χειρότερη κρίσιμη διαδρομή.

Στην εικόνα 1.22 παρουσιάζεται η χειρότερη σύντομη διαδρομή που δεν παραβιάζει τον χρόνο διατήρησης. Η σύντομη διαδρομή έχει καθυστέρηση μόλυνσης 0,273 ns, εκ των οποίων τα 0,164 ns αφορούν στη λογική (logic), ενώ τα 0,109 ns αφορούν στη δικτύωση (net).

Name	Slack ^{^1}	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 11	0.060	0	1	Datapath_compo...out_reg[12]/C	Datapath_compo..._reg/DIADI[12]	0.273	0.164	0.109

Εικόνα 1.22 Χειρότερη σύντομη διαδρομή.

Στην εικόνα 1.23 εμφανίζεται το Path 11 του Timing Report, δηλαδή η χειρότερη σύντομη διαδρομή. Η σύντομη διαδρομή αποτελείται από 0 λογικά επίπεδα και περνάει από ένα block RAMB18E1. Το μονοπάτι αυτό είναι μια σύνδεση του Write Data Register σε ένα port της Μνήμης Δεδομένων RAM.



Εικόνα 1.23 Χειρότερη σύντομη διαδρομή.

2. Επαλήθευση της ορθής σχεδίασης και λειτουργίας του επεξεργαστή

2.1. Πρόγραμμα ARM σε γλώσσα Assembly για επαλήθευση της ορθής σχεδίασης και λειτουργίας του επεξεργαστή

Το πρόγραμμα της παρακάτω εικόνας σε ARM Assembly εκτελεί όλες τις εντολές του επεξεργαστή πολλών κύκλων εκτελώντας υπολογισμούς οι οποίοι πρέπει να παράγουν το σωστό αποτέλεσμα μόνο όταν όλες οι εντολές εκτελεστούν σωστά. Συγκεκριμένα, το πρόγραμμα θα κάνει εγγραφή της τιμής 7 στην θέση μνήμης 100 αν τρέξει σωστά. Επιπλέον, ενημερώνουμε τις σημαίες του Status Register, αν η εντολή έχει το πρόθεμα S και δοκιμάζουμε τα μνημονικά κατά την εκτέλεση ορισμένων εντολών.

	PROGRAM	COMMENTS	HEX CODE
1.	main:		
2.	MOV R0, #0	; R0 = 0	E3A00000
3.	MOV R1, #5	; R1 = 5	E3A01005
4.	MOV R2, #12	; R2 = 12	E3A0200C
5.	MOV R3, #15	; R3 = 15	E3A0300F
6.	MOV R4, #15	; R4 = 15	E3A0400F
7.	MOV R5, #1	; R5 = 1	E3A05001
8.	ADD R6, R0, R1	; R6 = R0 + R1 = 0 + 5 = 5	E0806001
9.	SUB R7, R2, #9	; R7 = R2 - 9 = 12 - 9 = 3	E2427009
10.	AND R8, R6, R7	; R8 = R6 AND R7 = 5 AND 3 = 1	E0068007
11.	ADD R8, R8, R7	; R8 = R8 + R7 = 1 + 3 = 4	E0888007
12.	CMP R8, #4	; Compare R8 with 4, set Flags	E3580004
13.	EOR R9, R7, R8	; R9 = R7 XOR R8 = 3 XOR 4 = 7	E0279008
14.	MOV R10, #15	; R10 = 15	E3A0A00F
15.	MVN R11, R10	; R11 = NOT R10 = 0xFFFFFFF0 = 4294967280	E1E0B00A
16.	LSL R12, R11, #1	; R12 = R11 << 1 = 0xFFFFFFE0 = 4294967264	E1A0C08B
17.	ASR R13, R12, #4	; R13 = R12 >> 4 = 0xFFFFF000 = 4294967296	E1A0D24C
18.	SUBS R14, R8, R6	; R14 = R8 - R6 = 4 - 5 = -1, set Flags	E058E006
19.	BEQ end	; should not be taken (Z flag is 0)	0A00000A
20.	SUBS R14, R2, R8	; R14 = R2 - R8 = 12 - 4 = 8, set flags	E052E008
21.	BGE end	; should be taken (N flag is 0)	AA000000
22.	ADD R8, R0, #0	; should be skipped	E2808000
23.			
24.	end:		
25.	SUBS R8, R7, R1	; R8 = R7 - R1 = 3 - 5 = -2, set Flags	E0578001
26.	ADDT R7, R8, #1	; R7 = R8 + 1 = -2 + 1 = -1, should be taken (N flag is 1)	B2887001
27.	SUB R7, R7, R1	; R7 = R7 - R1 = -1 - 5 = -6	E0477001
28.	STR R7, [R2, #84]	; mem[R2+84] = mem[12+84] = mem[96] = R7 = -6	E5827054
29.	LDR R1, [R0, #96]	; R1 = mem[96+R0] = mem[96] = -6	E5901060
30.	ADD R15, R15, R0	; PC = PC + 8 (skip next instruction)	E08FF000
31.	ADD R2, R0, #14	; should be skipped	E280200E
32.	ADDS R1, R1, #13	; R1 = R1 + 13 = -6 + 13 = 7, set Flags	E291100D
33.	STRVC R1, [R0, #100]	; mem[R0+100] = mem[100] = R1 = 7, should be taken (V flag is 0)	75801064
34.	BL main	; always taken	EBFFFFE0

Εικόνα 2.1 Πρόγραμμα επαλήθευσης του επεξεργαστή.

Δίπλα από κάθε εντολή παρατίθεται σε σχόλιο η ανάλυση της εντολής αλλά και η αντίστοιχη εντολή σε δεκαεξαδική βάση σε γλώσσα μηχανής, με την οποία βάση εισάχθηκε στην Μνήμη Εντολών ROM. Το πρόγραμμα ξεκινάει στον κλάδο main, όπου οι πρώτες 6 εντολές αποθηκεύουν στους καταχωρητές R0, R1, R2, R3, R4, R5

τις τιμές 0, 5, 12, 15, 15 και 1 αντίστοιχα. Η 7^η εντολή προσθέτει τις τιμές των καταχωρητών R0 και R1 και εισάγει το αποτέλεσμα 5 της πρόσθεσης στον καταχωρητή R6. Η 8^η εντολή αφαιρεί από την τιμή του καταχωρητή R2 την ακέραια τιμή 9 και εισάγει το αποτέλεσμα 3 της αφαίρεσης στον καταχωρητή R7. Η 9^η εντολή εκτελεί την λογική πράξη AND για τις τιμές των καταχωρητών R6 και R7 και εισάγει το αποτέλεσμα 1 της πράξης στον καταχωρητή R8. Η 10^η εντολή, δηλαδή η εντολή στην γραμμή 11, προσθέτει την τιμή του καταχωρητή R7 στην ήδη υπάρχουσα τιμή του καταχωρητή R8 και εισάγει το αποτέλεσμα 4 της πρόσθεσης στον καταχωρητή R8. Η 11^η εντολή είναι εντολή σύγκρισης η οποία συγκρίνει το περιεχόμενο του καταχωρητή R8 με την ακέραια τιμή 4 θα παράξει μηδενική διαφορά γιατί έχουν ίδια τιμή, ενώ ταυτόχρονα θα ενημερώσει και την σημαία Z. Η 12^η εντολή εκτελεί την λογική πράξη EOR (XOR) για τις τιμές των καταχωρητών R7 και R8 και εισάγει το αποτέλεσμα 7 της πράξης στον καταχωρητή R9. Η 13^η εντολή εισάγει την τιμή 15 στον καταχωρητή R10. Η εντολή της γραμμής 15, δηλαδή η 14^η εντολή, αποθηκεύει στον καταχωρητή R11 τον αριθμό 0xFFFFF0 (δεκαεξαδική μορφή), δηλαδή τον αριθμό 4294967280 (δεκαδική μορφή), ο οποίος είναι ο αριθμός που προκύπτει αν αντιστρέψουμε όλα τα bit της διαδικής μορφής του αριθμού 15, δηλαδή της τιμής του καταχωρητή R10 (111111111111111111111111111111110000 = NOT 000000000000000000000000000000001111). Η εντολή της γραμμής 16, δηλαδή η 15^η εντολή, αποθηκεύει στον καταχωρητή R12 τον αριθμό 0xFFFFFE0 (δεκαεξαδική μορφή), δηλαδή τον αριθμό 4294967264 (δεκαδική μορφή), ο οποίος είναι ο αριθμός που προκύπτει αν πραγματοποιηθεί λογική ολίσθηση προς τα αριστερά της τιμής του καταχωρητή R11 κατά μία θέση (1111111111111111111111111111111100000 = 111111111111111111111111111111110000 << 1). Η εντολή της γραμμής 17, δηλαδή η 16^η εντολή, αποθηκεύει στον καταχωρητή R13 τον αριθμό 0xFFFFFFE (δεκαεξαδική μορφή), δηλαδή τον αριθμό 4294967294 (δεκαδική μορφή), ο οποίος είναι ο αριθμός που προκύπτει αν πραγματοποιηθεί αριθμητική ολίσθηση προς τα δεξιά της τιμής του καταχωρητή R12 κατά 4 θέσεις (1111111111111111111111111111111110= 1111111111111111111111111111111100000 >> 4). Η 17^η εντολή αφαιρεί από την τιμή του καταχωρητή R8 την τιμή του καταχωρητή R6 και εισάγει το αποτέλεσμα -1 της αφαίρεσης στον καταχωρητή R14, ενώ λόγω του προθέματος S της εντολής ενημερώνονται οι σημαίες και ενεργοποιείται η σημαία N. Η εντολή διακλάδωσης B στην γραμμή 19 η οποία θα πήγαινε το πρόγραμμα στον κλάδο end αλλά δεν θα εκτελεστεί καθώς έχει το μνημονικό EQ, δηλαδή πρέπει η εντολή που ενημέρωσε τελευταία τις σημαίες να είχε μηδενικό αποτέλεσμα, άρα να είναι ενεργοποιημένη η σημαία Z, η οποία σε αυτό το σημείο του προγράμματος δεν είναι ενεργοποιημένη. Η 19^η εντολή αφαιρεί από την τιμή του καταχωρητή R2 την τιμή του καταχωρητή R8 και εισάγει το αποτέλεσμα 8 της αφαίρεσης στον καταχωρητή R14, ενώ λόγω του προθέματος S της εντολής ενημερώνονται οι σημαίες και ενεργοποιείται η σημαία C. Η εντολή διακλάδωσης B στην γραμμή 21 η οποία πηγαίνει το πρόγραμμα στον κλάδο end θα εκτελεστεί καθώς έχει το μνημονικό GE, δηλαδή πρέπει η εντολή που ενημέρωσε τελευταία τις σημαίες να είχε αποτέλεσμα μη μηδενικό, άρα να μην είναι ενεργοποιημένη η σημαία N, η οποία σε αυτό το σημείο του προγράμματος δεν

είναι ενεργοποιημένη. Η εντολή πρόσθεσης στην γραμμή 22 δεν θα εκτελεστεί καθώς το πρόγραμμα την προσπερνάει και πηγαίνει σε άλλον κλάδο. Το πρόγραμμα τώρα πηγαίνει στον κλάδο end και εκτελεί την εντολή της γραμμής 25, η οποία αφαιρεί από την τιμή του καταχωρητή R7 την τιμή του καταχωρητή R1 και εισάγει το αποτέλεσμα -2 της αφαίρεσης στον καταχωρητή R8, ενώ λόγω του προθέματος S της εντολής ενημερώνονται οι σημαίες και ενεργοποιείται η σημαία N. Η εντολή της γραμμής 26, η οποία προσθέτει την τιμή του καταχωρητή R8 και την ακέραια τιμή 1 και εισάγει το αποτέλεσμα -1 της πρόσθεσης στον καταχωρητή R7, θα εκτελεστεί καθώς έχει το μνημονικό LT, δηλαδή πρέπει η εντολή που ενημέρωσε τελευταία τις σημαίες να είχε αρνητικό αποτέλεσμα χωρίς υπερχείληση, άρα να είναι ενεργοποιημένη η σημαία N και να μην είναι ενεργοποιημένη η σημαία Z. Η σημαία N είναι ενεργοποιημένη σε αυτό το σημείο του προγράμματος, ενώ η σημαία Z δεν είναι. Η εντολή στην γραμμή 27 αφαιρεί από την τιμή του καταχωρητή R7 την τιμή του καταχωρητή R1. Η εντολή στην γραμμή 28 αποθηκεύει στην διεύθυνση 96 της μνήμης την τιμή -6 του καταχωρητή R7. Η εντολή στην γραμμή 29 αποθηκεύει στον καταχωρητή R1 την τιμή της διεύθυνσης 96 της μνήμης. Η εντολή της γραμμής 30 προσθέτει την τιμή του καταχωρητή R0 (που είναι 0) στην τιμή του καταχωρητή R15, ο οποίος είναι ο PC. Το αποτέλεσμα είναι ότι ο PC αυξάνεται κατά 8, δηλαδή η επόμενη εντολή θα εκτελεστεί μετά από δύο θέσεις μνήμης. Αυτό σημαίνει ότι η εντολή πρόσθεσης στην γραμμή 31 θα παραληφθεί και το πρόγραμμα θα συνεχιστεί από τη γραμμή 32. Στην γραμμή 32 προστίθεται στον καταχωρητή R1 η ακέραια τιμή 13, και η τιμή του R1 τώρα είναι 7, ενώ ενημερώνονται και οι σημαίες. Η εντολή στην γραμμή 33 αποθηκεύει στην διεύθυνση 100 της μνήμης την τιμή 7 του καταχωρητή R1, αν ικανοποιείται η συνθήκη της μη ύπαρξης υπερχείλησης, δηλαδή η σημαία V να μην είναι ενεργοποιημένη, όπου εδώ ισχύει πως δεν είναι. Επομένως η σωστή λειτουργία του επεξεργαστή επιβεβαιώνεται από αυτό το πρόγραμμα. Στην γραμμή 34 εκτελείται η εντολή διακλάδωσης BL η οποία επαναφέρει την συνέχεια της εκτέλεσης του προγράμματος στον κλάδο main, δηλαδή στην αρχή του προγράμματος. Αν το πρόγραμμα δεν επανέλθει στην πρώτη του εντολή σημαίνει ότι δεν λειτούργησε σωστά ο επεξεργαστής.

2.2. Προγράμματα δοκιμής και τα διαγράμματα χρονισμού για Register File και ALU

2.2.1. Πρόγραμμα δοκιμής και διάγραμμα χρονισμού του Register File

Το testbench παρακάτω γράφτηκε για τον Register File για να επιβεβαιωθεί η σωστή λειτουργία του. Η προσομοίωση αυτή λειτουργεί με σταθερή περίοδο ρολογιού 10 ns.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity Register_File_tb is
6  end entity Register_File_tb;
7
8  architecture Behavioral of Register_File_tb is
9
10     constant N : integer := 4; -- Address Size (bits), 2^N registers
11     constant M : integer := 32; -- Word Size (bits)
12
13     -- Unit Under Test (UUT) Component
14     component Register_File
15     port (
16         CLK          : in std_logic;           -- Clock Signal
17         Write_Enable_3 : in std_logic;         -- Signal that enables writting to register
18         Address_1     : in std_logic_vector(N-1 downto 0); -- Read address 1
19         Address_2     : in std_logic_vector(N-1 downto 0); -- Read address 2
20         Address_3     : in std_logic_vector(N-1 downto 0); -- Write address 3
21         Write_Data_3   : in std_logic_vector(M-1 downto 0); -- Write data to address 3
22         R15           : in std_logic_vector(M-1 downto 0); -- PC + 8 address
23         Read_Data_1    : out std_logic_vector(M-1 downto 0); -- Read data from address 1
24         Read_Data_2    : out std_logic_vector(M-1 downto 0); -- Read data from address 2
25     );
26 end component Register_File;
27
28 -- UUT Signals
29 signal CLK_tb          : std_logic;
30 signal Write_Enable_3_tb : std_logic;
31 signal Address_1_tb    : std_logic_vector(N-1 downto 0);
32 signal Address_2_tb    : std_logic_vector(N-1 downto 0);
33 signal Address_3_tb    : std_logic_vector(N-1 downto 0);
34 signal Write_Data_3_tb : std_logic_vector(M-1 downto 0);
35 signal R15_tb          : std_logic_vector(M-1 downto 0);
36 signal Read_Data_1_tb  : std_logic_vector(M-1 downto 0);
37 signal Read_Data_2_tb  : std_logic_vector(M-1 downto 0);
38
39 constant clk_period : time := 10 ns;
40
41 begin
42
43     -- Instantiate the UUT
44     utt : Register_File
45     port map(
46         Address_1  => Address_1_tb,
47         Address_2  => Address_2_tb,
48         Address_3  => Address_3_tb,
49         Write_Data_3 => Write_Data_3_tb,
50         R15        => R15_tb,
51         CLK        => CLK_tb,
52         Write_Enable_3 => Write_Enable_3_tb,
53         Read_Data_1 => Read_Data_1_tb,
54         Read_Data_2 => Read_Data_2_tb
55     );
```

```

56 |
57 | -- Clock process
58 | clock: process is
59 |     begin
60 |         CLK_tb <= '0';
61 |         wait for clk_period/2;
62 |         CLK_tb <= '1';
63 |         wait for clk_period/2;
64 |     end process clock;
65 |
66 | -- Stimulus process
67 | stimulus : process is
68 |     begin
69 |         wait for 100 ns;
70 |
71 |         -- Initialize signals
72 |         Write_Enable_3_tb <= '0';
73 |         Address_1_tb <= (others => '0');
74 |         Address_2_tb <= (others => '0');
75 |         Address_3_tb <= (others => '0');
76 |         Write_Data_3_tb <= (others => '0');
77 |         R15_tb <= (others => '0');
78 |         wait for 100 ns;
79 |
80 |         -- Write to register 0
81 |         Write_Enable_3_tb <= '1';
82 |         Address_3_tb <= "0000"; -- Register 0
83 |         Write_Data_3_tb <= "00000000000000000000000000000001";
84 |         wait for clk_period;
85 |
86 |         -- Write to register 1
87 |         Address_3_tb <= "0001"; -- Register 1
88 |         Write_Data_3_tb <= "11110000000000000000000000000000";
89 |         wait for clk_period;
90 |
91 |         -- Write to register 15 (R15 should not change)
92 |         Address_3_tb <= "1111"; -- Register 15
93 |         Write_Data_3_tb <= "00000000000000000000000000000000";
94 |         wait for clk_period;
95 |
96 |         -- Disable write enable
97 |         Write_Enable_3_tb <= '0';
98 |         wait for clk_period;
99 |
100 |         -- Read from register 0
101 |         Address_1_tb <= "0000";
102 |         wait for clk_period;
103 |         assert Read_Data_1_tb = "00000000000000000000000000000001"
104 |             report "Test failed for register 0" severity error;
105 |

```



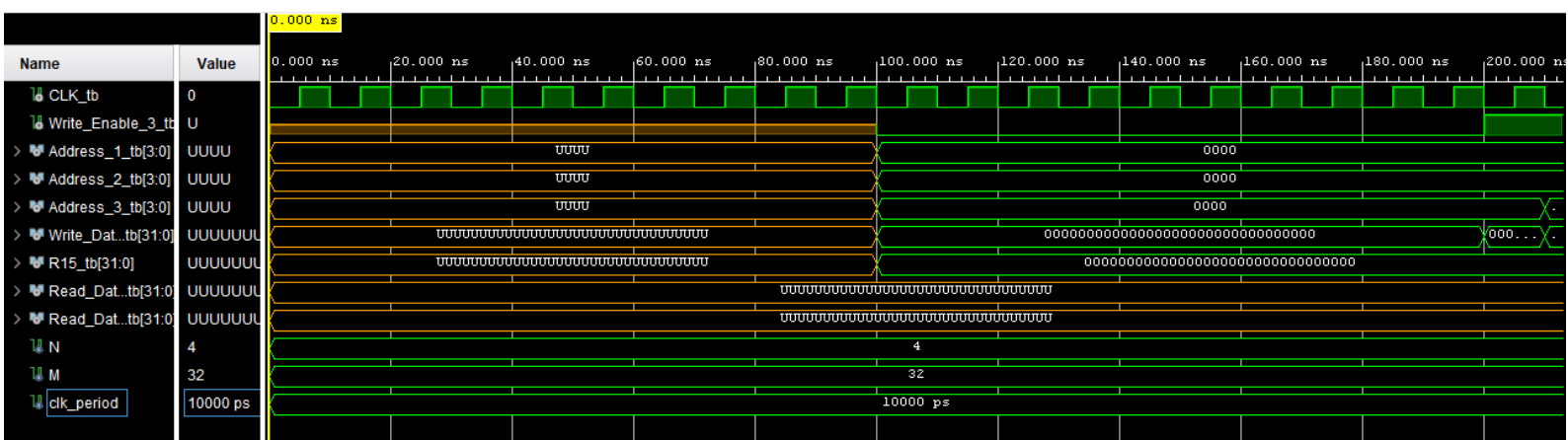
```

105
106 -- Read from register 15 for address 1
107 -- PC = 4
108 R15_tb <= std_logic_vector(unsigned(R15_tb) + 4);
109 Address_1_tb <= "1111";
110 wait for clk_period;
111 assert Read_Data_1_tb = "0000000000000000000000000000100"
112     report "Test failed for register 15" severity error;
113
114 -- Read from register 1
115 Address_2_tb <= "0001";
116 wait for clk_period;
117 assert Read_Data_2_tb = "11110000000000000000000000000000"
118     report "Test failed for register 1" severity error;
119
120 -- Read from register 15 for address 2
121 -- PC = 8
122 R15_tb <= std_logic_vector(unsigned(R15_tb) + 4);
123 Address_2_tb <= "1111";
124 wait for clk_period;
125 assert Read_Data_1_tb = "00000000000000000000000000001000"
126     report "Test failed for register 15" severity error;
127
128 -- End of the testbench
129 report "Testbench completed successfully";
130 wait;
131 end process stimulus;
132 end architecture Behavioral;

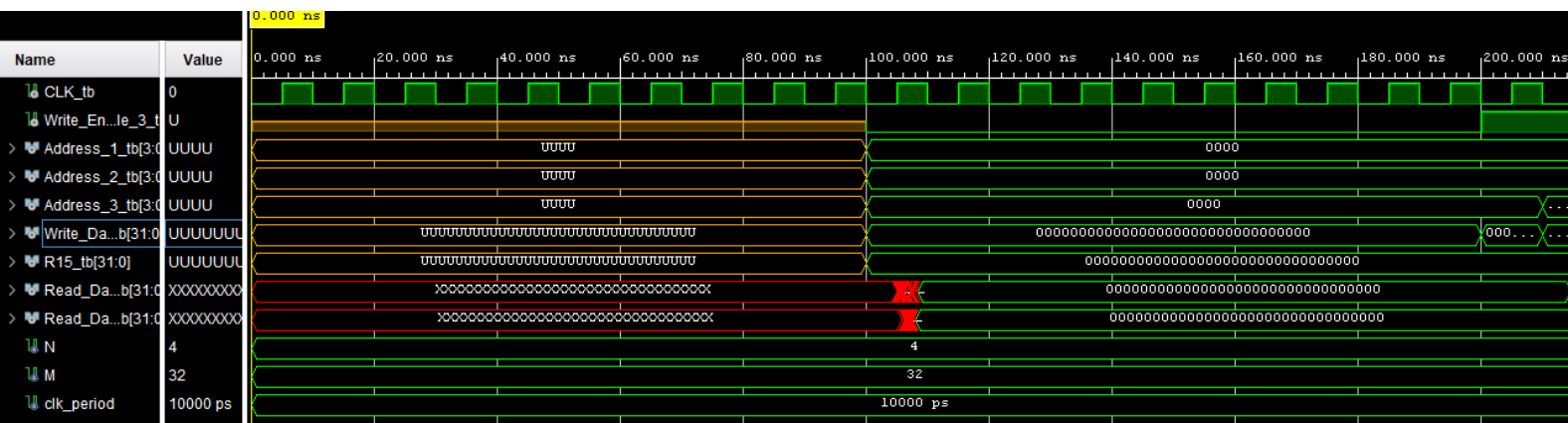
```

Εικόνα 2.2 Πρόγραμμα προσομοίωσης του Register File.

Αρχικά, το testbench αναμένει 100 ns πριν την έναρξη των διεργασιών. Στην συνέχεια αρχικοποιούνται όλα τα σήματα ελέγχου και δεδομένων σε μηδενική τιμή και αναμένουν άλλα 100 ns.

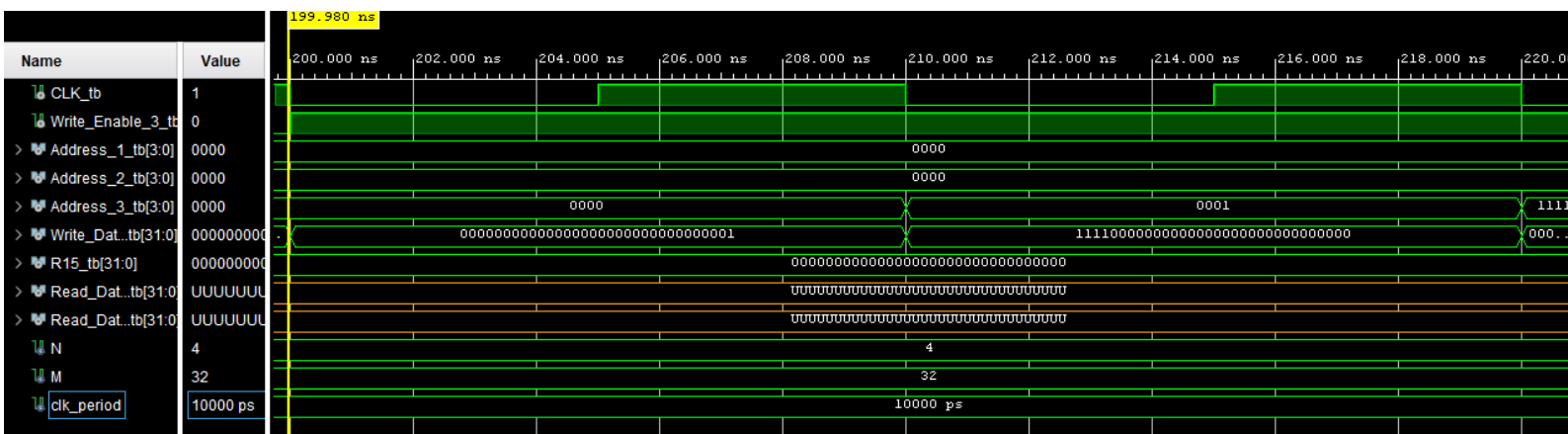


Εικόνα 2.3 Behavioral simulation για την αρχικοποίηση σημάτων.

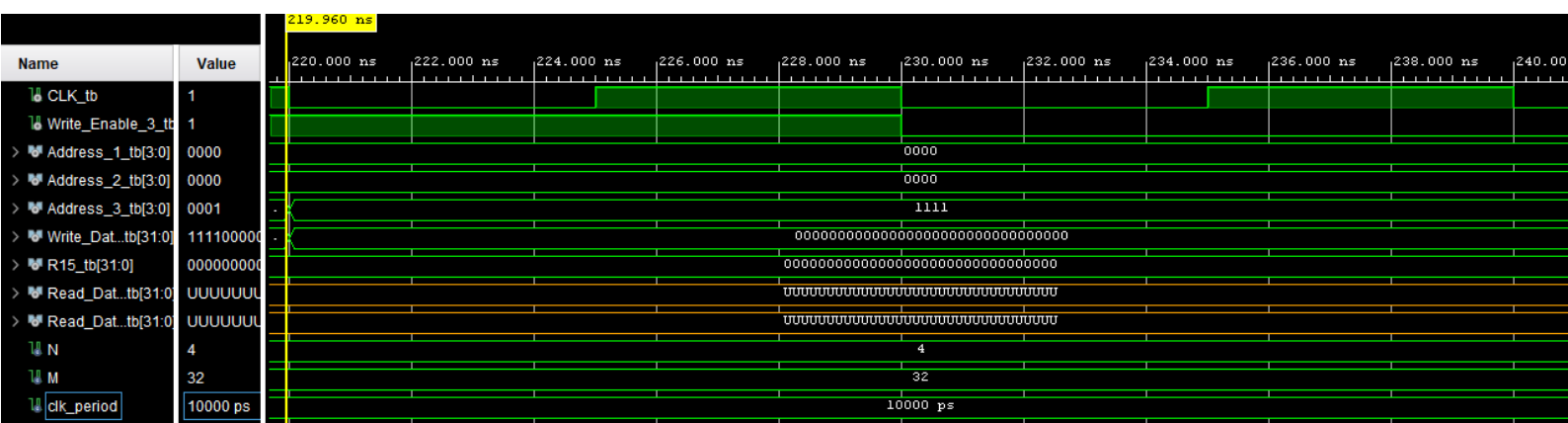


Εικόνα 2.4 *Post-Implementation Timing simulation για την αρχικοποίηση σημάτων.*

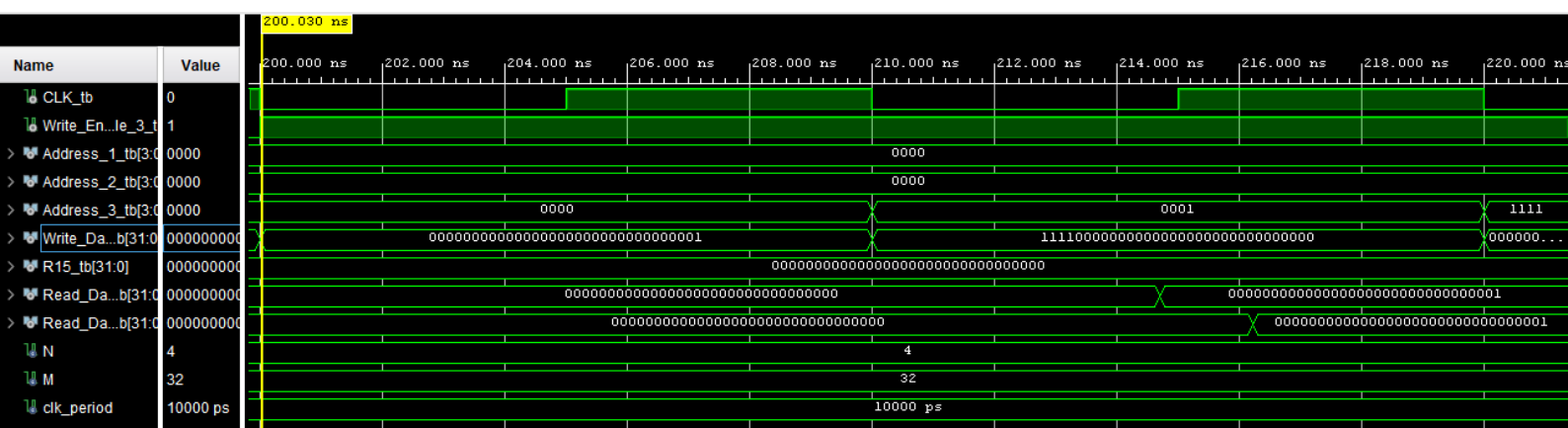
Στην συνέχεια του προγράμματος εγγράφονται τα δεδομένα 00000000000000000000000000000001 στον καταχωρητή με διεύθυνση 0000 (Register 0), Τα δεδομένα 11110000000000000000000000000000 στον καταχωρητή με διεύθυνση 0001 (Register 1), ενώ γράφονται τα δεδομένα 00000000000000000000000000000000 στον καταχωρητή 15 (R15), ο οποίος όμως δεν αλλάζει. Τέλος, Η εγγραφή απενεργοποιείται. Στα παρακάτω διαγράμματα και του behavioral simulation και του post-implementation timing simulation για την χρονική περίοδο 200 ns έως 240 ns παρατηρείται σωστά η αλλαγή των σημάτων Write_Enable_3_tb, Address_3_tb και Write_Data_3_tb.



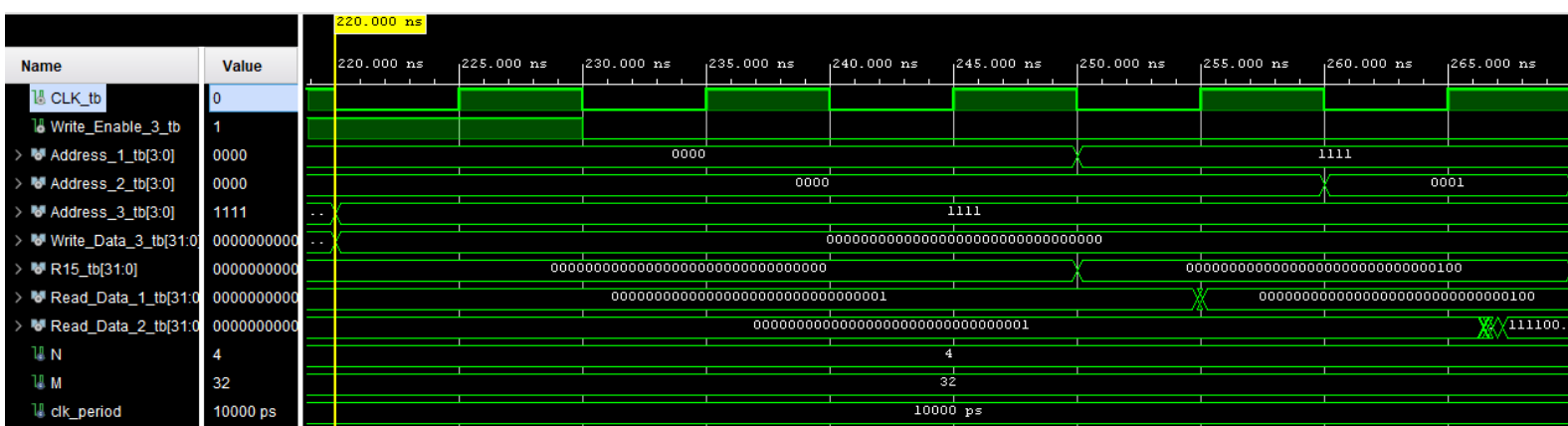
Εικόνα 2.5 *Behavioral simulation για εγγραφή σε καταχωρητές.*



Εικόνα 2.6 Behavioral simulation για εγγραφή σε καταχωρητές.

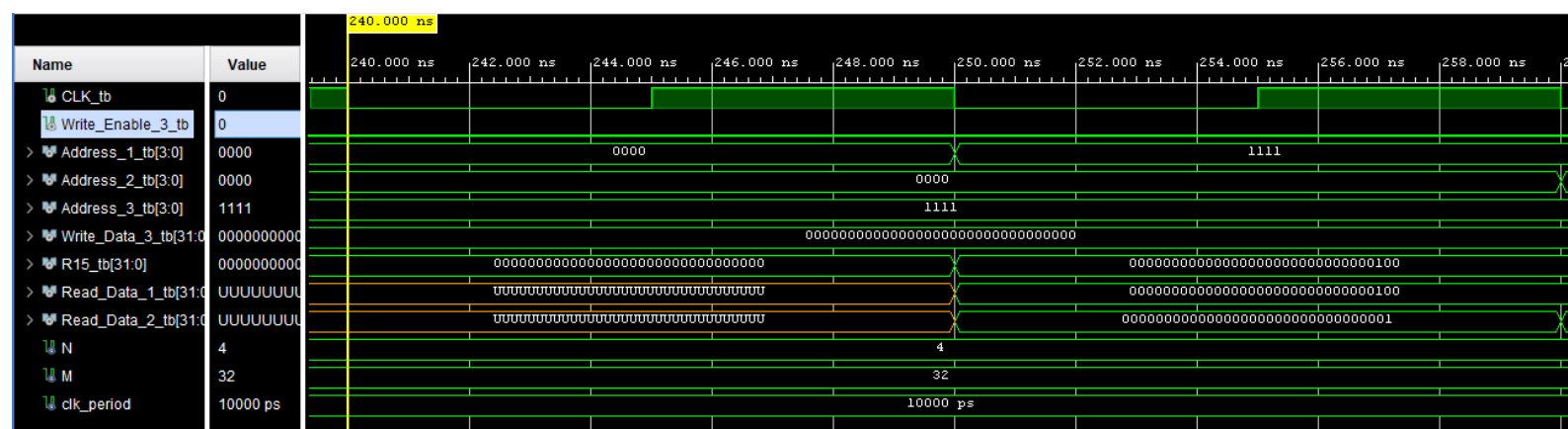


Εικόνα 2.7 Post-implementation Timing simulation για εγγραφή σε καταχωρητές.

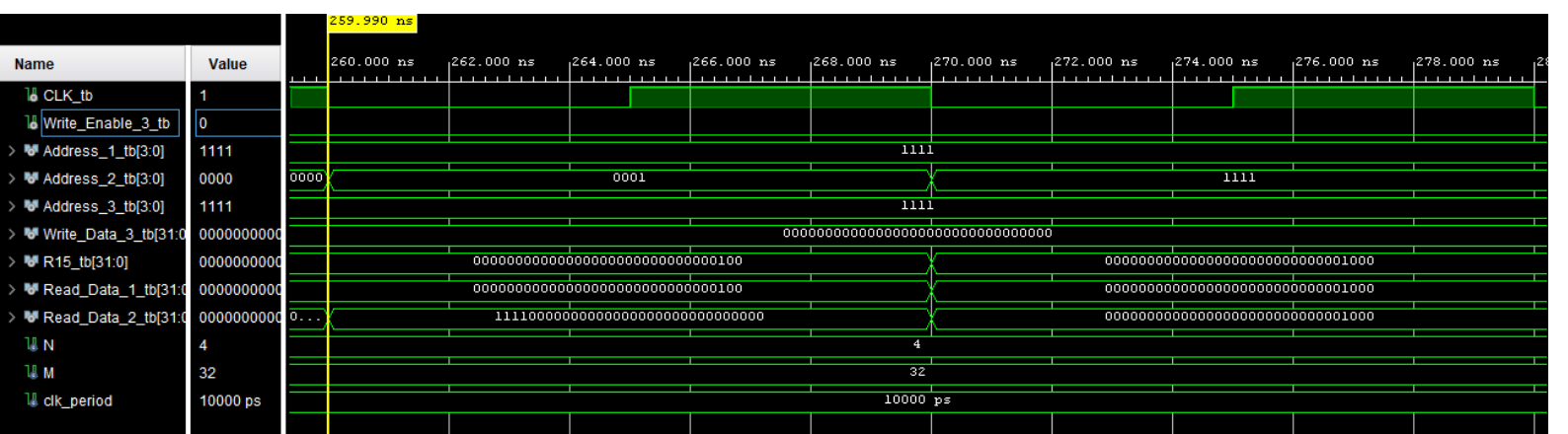


Εικόνα 2.8 Post-implementation Timing simulation για εγγραφή σε καταχωρητές.

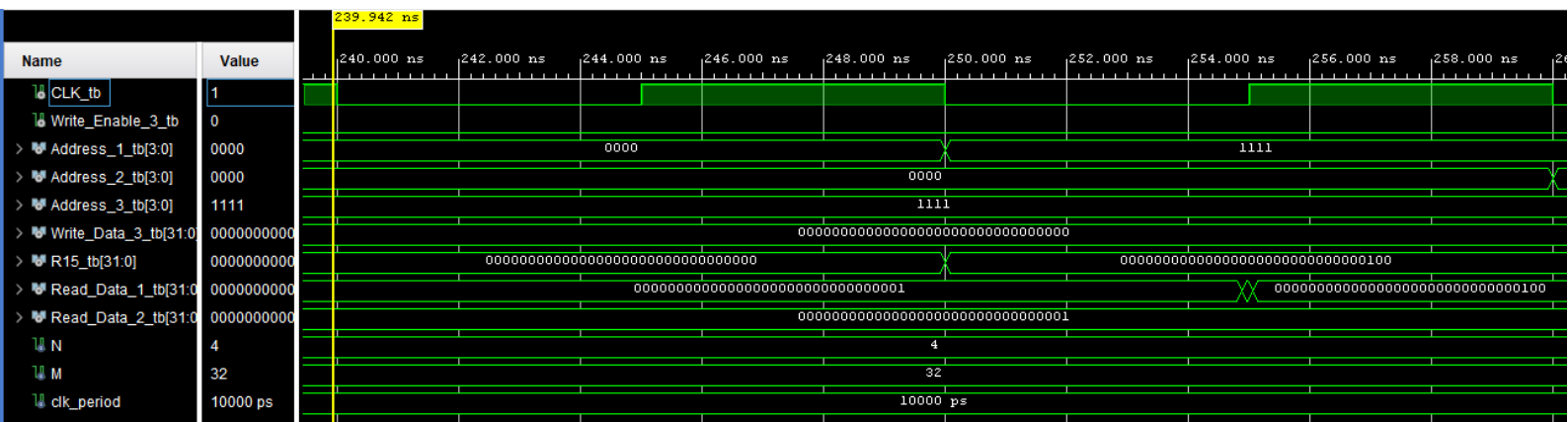
Στην συνέχεια του προγράμματος διαβάζονται τα δεδομένα από τον καταχωρητή 0 και ελέγχεται αν είναι ίσα με 00000000000000000000000000000001. Αν όχι, αναφέρει σφάλμα. Έπειτα, το testbench αυξάνει την τιμή του R15 κατά 4 και διαβάζει από τον καταχωρητή 15 για τη διεύθυνση 1, ελέγχοντας αν η τιμή είναι 0000000000000000000000000000000100. Μετά διαβάζει τα δεδομένα από τον καταχωρητή 1 και ελέγχει αν είναι ίσα με 11110000000000000000000000000000. Αν όχι, αναφέρει σφάλμα. Τέλος, αυξάνει την τιμή του R15 κατά 4 και διαβάζει από τον καταχωρητή 15 για τη διεύθυνση 2, ελέγχοντας αν η τιμή είναι 00000000000000000000000000000001000. Στα παρακάτω διαγράμματα και του behavioral simulation και του post-implementation timing simulation για την χρονική περίοδο 240 ns έως το τέλος της προσομοίωσης παρατηρείται σωστά η αλλαγή των σημάτων R15_tb, Address_1_tb, Address_2_tb, Read_Data_1_tb και Read_Data_2_tb. Στις εικόνες 2.11 και 2.12 παρατηρείται και η καθυστέρηση διάδοσης κατά την ασύγχρονη ανάγνωση κατά το post-implementation timing simulation.



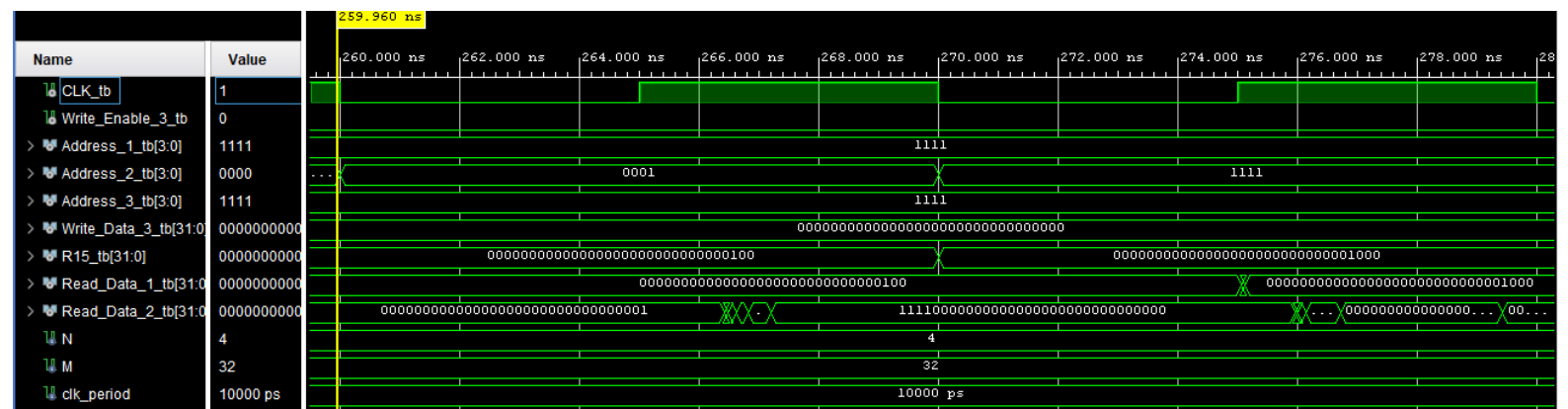
Εικόνα 2.9 Behavioral simulation για ανάγνωση από καταχωρητές.



Εικόνα 2.10 Behavioral simulation για ανάγνωση από καταχωρητές.



Εικόνα 2.11 *Post-implementation simulation* για ανάγνωση από καταχωρητές.



Εικόνα 2.12 *Post-implementation simulation* για ανάγνωση από καταχωρητές.

2.2.2. Πρόγραμμα δοκιμής και διάγραμμα χρονισμού της μονάδας ALU

Το testbench παρακάτω γράφτηκε για την μονάδα ALU για να επιβεβαιωθεί η σωστή λειτουργία της. Η προσομοίωση αυτή λειτουργεί με σταθερή περίοδο ρολογιού 10 ns.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity ALU_tb is
6  end entity ALU_tb;
7
8  architecture Behavioral of ALU_tb is
9
10     constant WIDTH : integer := 32;
11
12     -- Unit Under Test (UUT) Component
13     component ALU
14     port (
15         SrcA      : in  std_logic_vector (WIDTH - 1 downto 0); -- 1st Input of ALU
16         SrcB      : in  std_logic_vector (WIDTH - 1 downto 0); -- 2nd Input of ALU
17         shamt5     : in  std_logic_vector (4 downto 0);          -- shift amount
18         sh         : in  std_logic_vector (1 downto 0);          -- shift type
19         ALUControl : in  std_logic_vector (2 downto 0);          -- 3-bit control signal to select the operation
20         ALUResult  : out std_logic_vector (WIDTH - 1 downto 0); -- ALU Output
21         N          : out std_logic;                               -- Negative flag
22         Z          : out std_logic;                               -- Zero flag
23         C          : out std_logic;                               -- Carry flag
24         V          : out std_logic;                               -- Overflow flag
25     );
26 end component;
27
28 -- UUT Signals
29 signal SrcA_tb      : std_logic_vector(WIDTH - 1 downto 0);
30 signal SrcB_tb      : std_logic_vector(WIDTH - 1 downto 0);
31 signal shamt5_tb    : std_logic_vector(4 downto 0);
32 signal sh_tb        : std_logic_vector(1 downto 0);
33 signal ALUControl_tb : std_logic_vector(2 downto 0);
34 signal ALUResult_tb : std_logic_vector(WIDTH - 1 downto 0);
35 signal N_tb         : std_logic;
36 signal Z_tb         : std_logic;
37 signal C_tb         : std_logic;
38 signal V_tb         : std_logic;
39
```

```

40 : begin
41 :
42 :     -- Instantiate the UUT
43 : uut: ALU
44 :     port map (
45 :         SrcA      => SrcA_tb,
46 :         SrcB      => SrcB_tb,
47 :         shamt5     => shamt5_tb,
48 :         sh         => sh_tb,
49 :         ALUControl => ALUControl_tb,
50 :         ALUResult  => ALUResult_tb,
51 :         N          => N_tb,
52 :         Z          => Z_tb,
53 :         C          => C_tb,
54 :         V          => V_tb
55 :     );
56 :
57 :     -- Stimulus process
58 : stimulus: process
59 : begin
60 :     wait for 100 ns;
61 :
62 :     -- Initialize inputs
63 :     SrcA_tb <= (others => '0');
64 :     SrcB_tb <= (others => '0');
65 :     shamt5_tb <= (others => '0');
66 :     sh_tb <= (others => '0');
67 :     ALUControl_tb <= (others => '0');
68 :     wait for 100 ns;
69 :
70 :     -- Test case 1: ADD positive result
71 :     SrcA_tb <= "00000000000000000000000000000001"; -- 1
72 :     SrcB_tb <= "00000000000000000000000000000001"; -- 1
73 :     ALUControl_tb <= "000"; -- ADD
74 :     wait for 10 ns;
75 :     assert (ALUResult_tb = "00000000000000000000000000000010" and N_tb = '0' and Z_tb = '0' and C_tb = '0' and V_tb = '0')
76 :         report "Test case 1 failed"
77 :         severity error;
78 :
79 :     -- Test case 2: ADD overflow
80 :     SrcA_tb <= "01111111111111111111111111111111"; -- 2147483647 (max positive 32-bit int)
81 :     SrcB_tb <= "00000000000000000000000000000001"; -- 1
82 :     ALUControl_tb <= "000"; -- ADD
83 :     wait for 10 ns;
84 :     assert (ALUResult_tb = "10000000000000000000000000000000" and N_tb = '1' and Z_tb = '0' and C_tb = '0' and V_tb = '1')
85 :         report "Test case 2 failed"
86 :         severity error;

```

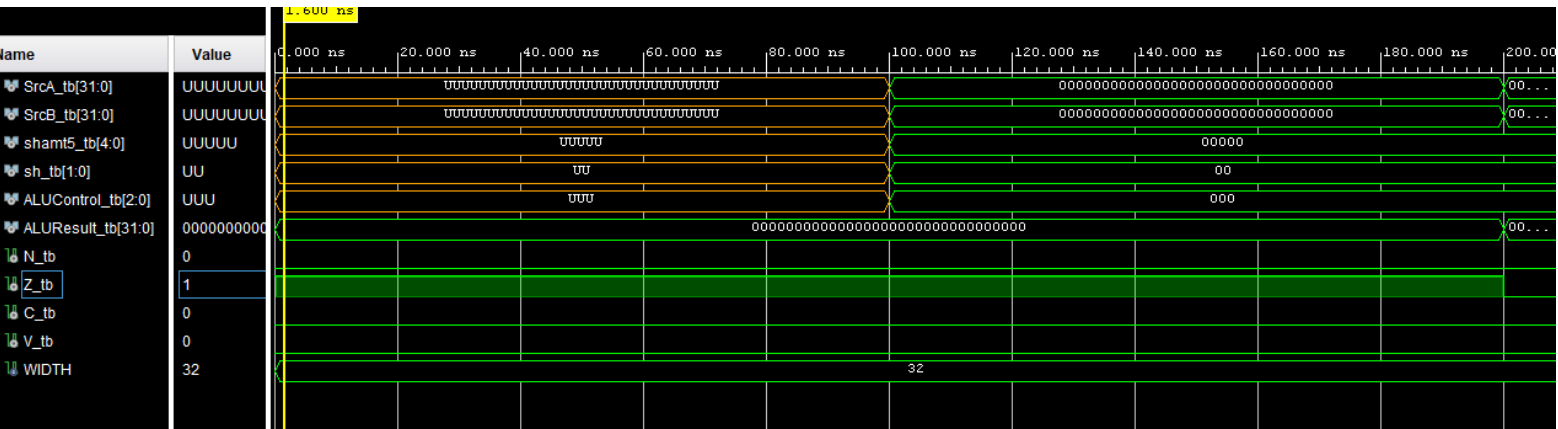
```

88      -- Test case 3: SUB negative result
89      SrcA_tb <= "00000000000000000000000000000001"; -- 1
90      SrcB_tb <= "00000000000000000000000000000010"; -- 2
91      ALUControl_tb <= "001"; -- SUB
92      wait for 10 ns;
93      assert (ALUResult_tb = "11111111111111111111111111111111" and N_tb = '1' and Z_tb = '0' and C_tb = '0' and V_tb = '0')
94          report "Test case 3 failed"
95          severity error;
96
97      -- Test case 4: SUB underflow
98      SrcA_tb <= "10000000000000000000000000000000"; -- -2147483648 (min negative 32-bit int)
99      SrcB_tb <= "00000000000000000000000000000001"; -- 1
100     ALUControl_tb <= "001"; -- SUB
101     wait for 10 ns;
102     assert (ALUResult_tb = "01111111111111111111111111111111" and N_tb = '0' and Z_tb = '0' and C_tb = '1' and V_tb = '1')
103         report "Test case 4 failed"
104         severity error;
105
106     -- Test case 5: AND
107     SrcA_tb <= "00000000000000000000000000000001"; -- 1
108     SrcB_tb <= "00000000000000000000000000000011"; -- 3
109     ALUControl_tb <= "010"; -- AND
110     wait for 10 ns;
111     assert (ALUResult_tb = "00000000000000000000000000000001" and N_tb = '0' and Z_tb = '0' and C_tb = '0' and V_tb = '0')
112         report "Test case 5 failed"
113         severity error;
114
115     -- Test case 6: XOR
116     SrcA_tb <= "00000000000000000000000000000001"; -- 1
117     SrcB_tb <= "00000000000000000000000000000011"; -- 3
118     ALUControl_tb <= "011"; -- EOR
119     wait for 10 ns;
120     assert (ALUResult_tb = "00000000000000000000000000000010" and N_tb = '0' and Z_tb = '0' and C_tb = '0' and V_tb = '0')
121         report "Test case 6 failed"
122         severity error;
123
124     -- Test case 7: LSL
125     SrcB_tb <= "00000000000000000000000000000001"; -- 1
126     shamt5_tb <= "00001"; -- Shift by 1
127     sh_tb <= "00"; -- LSL
128     ALUControl_tb <= "100"; -- Shift operation
129     wait for 10 ns;
130     assert (ALUResult_tb = "00000000000000000000000000000010" and N_tb = '0' and Z_tb = '0' and C_tb = '0' and V_tb = '0')
131         report "Test case 7 failed"
132         severity error;
133
134     -- Test case 8: ASR
135     SrcB_tb <= "00000000000000000000000000000001"; -- Most significant bit is 1
136     shamt5_tb <= "00001"; -- Shift by 1
137     sh_tb <= "10"; -- ASR
138     ALUControl_tb <= "100"; -- Shift operation
139     wait for 10 ns;
140     assert (ALUResult_tb = "00000000000000000000000000000001" and N_tb = '0' and Z_tb = '0' and C_tb = '0' and V_tb = '0')
141         report "Test case 8 failed"
142         severity error;
143
144     -- Test case 9: MOV
145     SrcB_tb <= "00000000000000000000000000000001"; -- 1
146     ALUControl_tb <= "101"; -- MOV
147     wait for 10 ns;
148     assert (ALUResult_tb = "00000000000000000000000000000001" and N_tb = '0' and Z_tb = '0' and C_tb = '0' and V_tb = '0')
149         report "Test case 9 failed"
150         severity error;
151
152     -- Test case 10: MVN
153     SrcB_tb <= "11111111111111111111111111111110"; -- 1
154     ALUControl_tb <= "110"; -- MVN
155     wait for 10 ns;
156     assert (ALUResult_tb = "00000000000000000000000000000001" and N_tb = '0' and Z_tb = '0' and C_tb = '0' and V_tb = '0')
157         report "Test case 10 failed"
158         severity error;
159
160     report "Testbench completed";

```

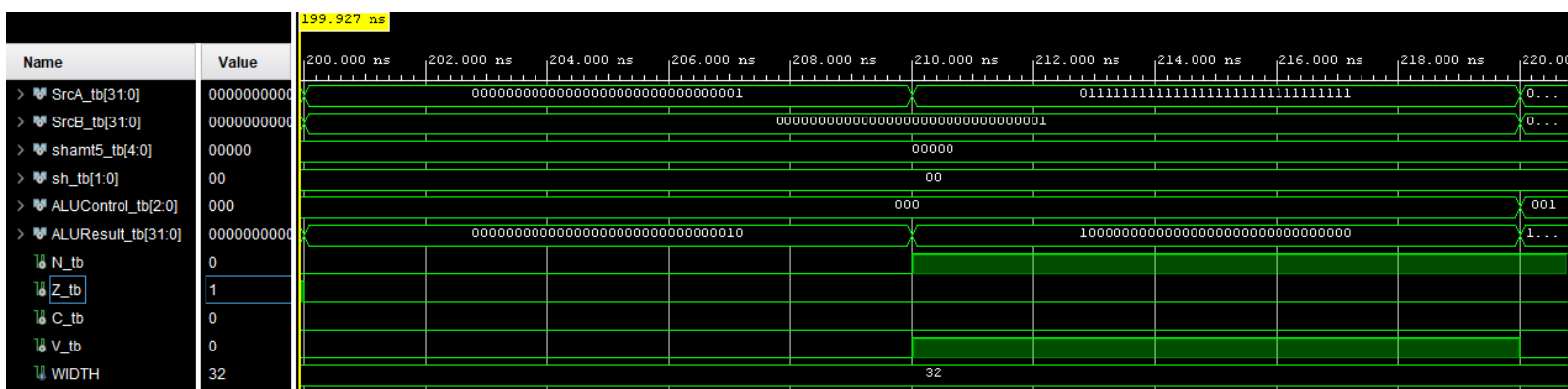
Εικόνα 2.13 Πρόγραμμα προσομοίωσης της μονάδας ALU.

Αρχικά το testbench ξεκινά με μια αναμονή 100 ns πριν την αρχικοποίηση των σημάτων εισόδου. Έπειτα τα σήματα εισόδου αρχικοποιούνται σε μηδενικές τιμές και ακολουθεί αναμονή 100 ns για σταθεροποίηση των σημάτων. Παρατηρούμε ότι το σήμα Z είναι ενεργό καθώς το ALU_Result_tb είναι μηδενικό στην αρχή.



Εικόνα 2.14 Behavioral simulation για αρχικοποίηση σημάτων.

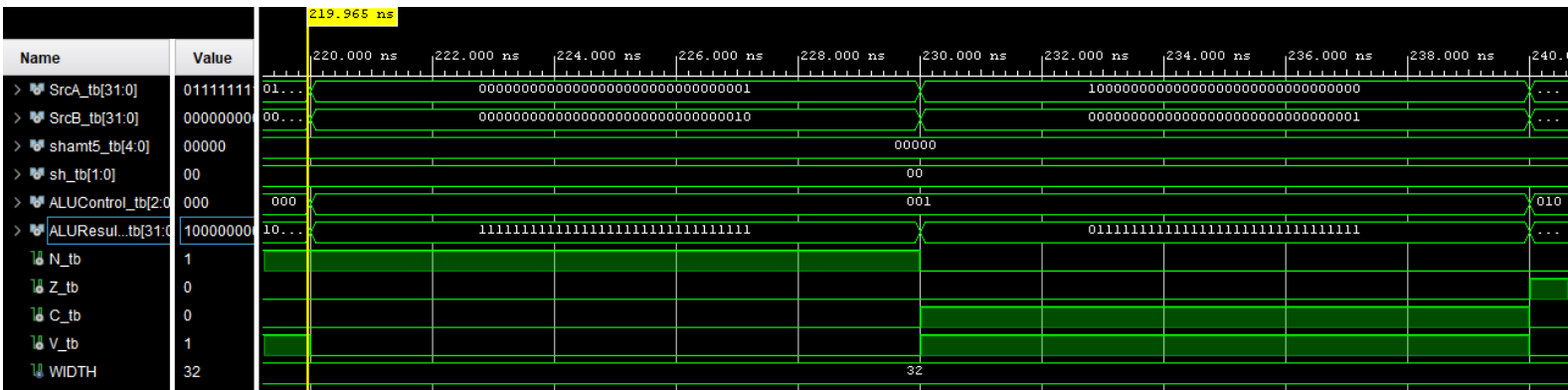
Στην συνέχεια στο 1^ο test case στους χρόνους 200 ns με 210 ns δοκιμάζεται η προσθήκη δύο θετικών αριθμών ($1 + 1$) με το σήμα ελέγχου ALUControl_tb να ορίζεται σε 000 για την λειτουργία της προσθήκης. Αναμένεται το αποτέλεσμα να είναι 2 και τα flags N, Z, C, V να είναι όλα 0. στο 2^ο test case στους χρόνους 210 ns με 220 ns δοκιμάζεται η προσθήκη του μέγιστου θετικού 32-bit ακέραιου με 1, προκαλώντας υπερχείλιση. Αναμένεται το αποτέλεσμα να είναι -2147483648, το N flag να είναι 1, το V flag να είναι 1, και τα Z και C flags να είναι 0. Στην εικόνα 2.16 του behavioral simulation για την πράξη ADD της ALU παρατηρείται σωστά η αλλαγή των σημάτων SrcA_tb, SrcB_tb, ALUControl_tb, ALUResult_tb, N_tb, Z_tb και V_tb.



Εικόνα 2.15 Behavioral simulation για την πράξη ADD.

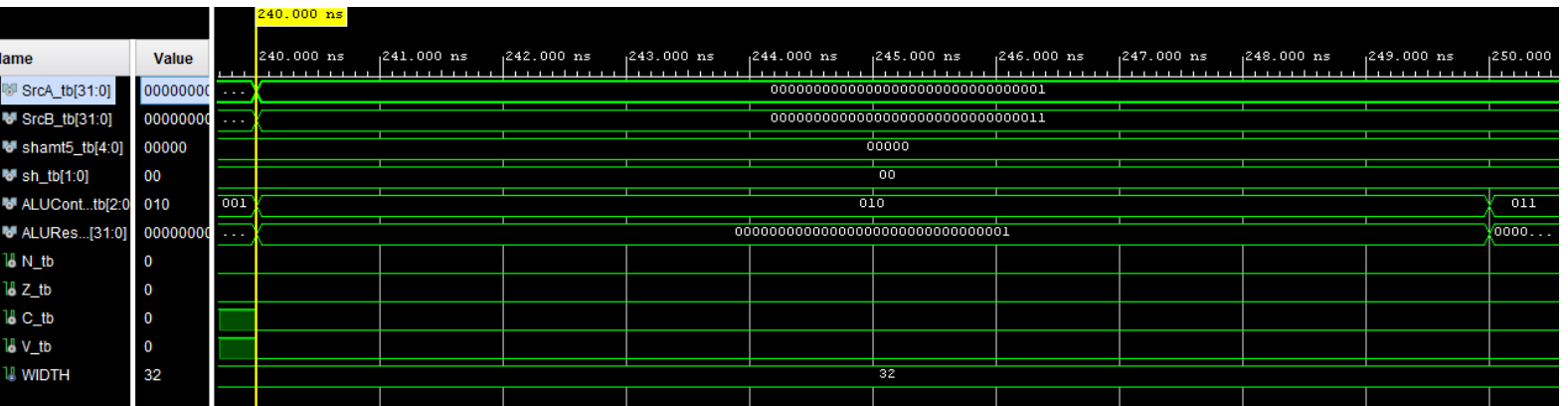
Στο 3^ο test case στους χρόνους 220 ns με 240 ns γίνεται αφαίρεση $1 - 2$, που δίνει αποτέλεσμα -1. Αναμένεται το αποτέλεσμα να είναι -1, το N flag να είναι 1, και τα Z, C, V flags να είναι 0. Στο 4^ο test case στους χρόνους 230 ns με 240 ns δοκιμάζεται η αφαίρεση του ελάχιστου αρνητικού 32-bit ακέραιου με 1. Αναμένεται το αποτέλεσμα να είναι 2147483647, το C flag να είναι 1, το V flag να είναι 1, και τα N και Z flags να είναι 0. Στην εικόνα 2.16 του behavioral simulation για την πράξη SUB

της ALU παρατηρείται σωστά η αλλαγή των σημάτων SrcA_tb, SrcB_tb, ALUControl_tb, ALUResult_tb, N_tb, Z_tb, C_tb και V_tb.



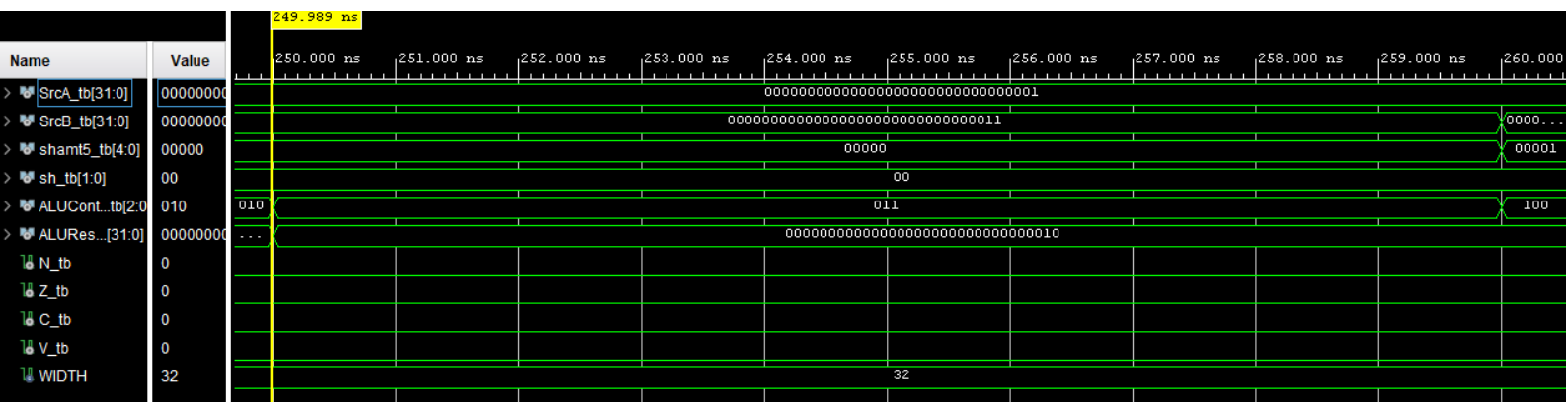
Εικόνα 2.16 Behavioral simulation για την πράξη SUB.

Στο 5^ο test case στους χρόνους 240 ns με 250 ns γίνεται η λογική πράξη AND μεταξύ 1 και 3, αναμένεται αποτέλεσμα 1 και όλα τα flags να είναι 0. Στην εικόνα 2.17 του behavioral simulation για την λογική πράξη AND της ALU παρατηρείται σωστά η αλλαγή των σημάτων SrcA_tb, SrcB_tb, ALUControl_tb, ALUResult_tb, N_tb, Z_tb, C_tb και V_tb.



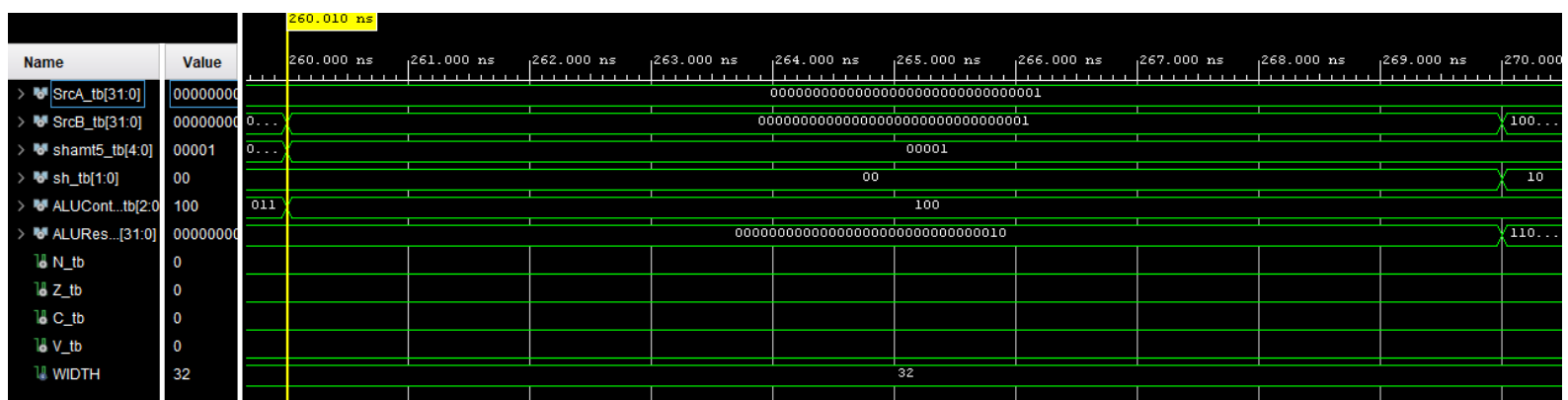
Εικόνα 2.17 Behavioral simulation για την εντολή AND.

Στο 6^ο test case στους χρόνους 250 ns με 260 ns γίνεται η εντολή EOR, δηλαδή η λογική πράξη XOR, μεταξύ 1 και 3, αναμένεται αποτέλεσμα 2 και όλα τα flags να είναι 0. Στην εικόνα 2.18 του behavioral simulation για την λογική πράξη XOR της ALU παρατηρείται σωστά η αλλαγή των σημάτων SrcA_tb, SrcB_tb, ALUControl_tb, ALUResult_tb, N_tb, Z_tb, C_tb και V_tb.



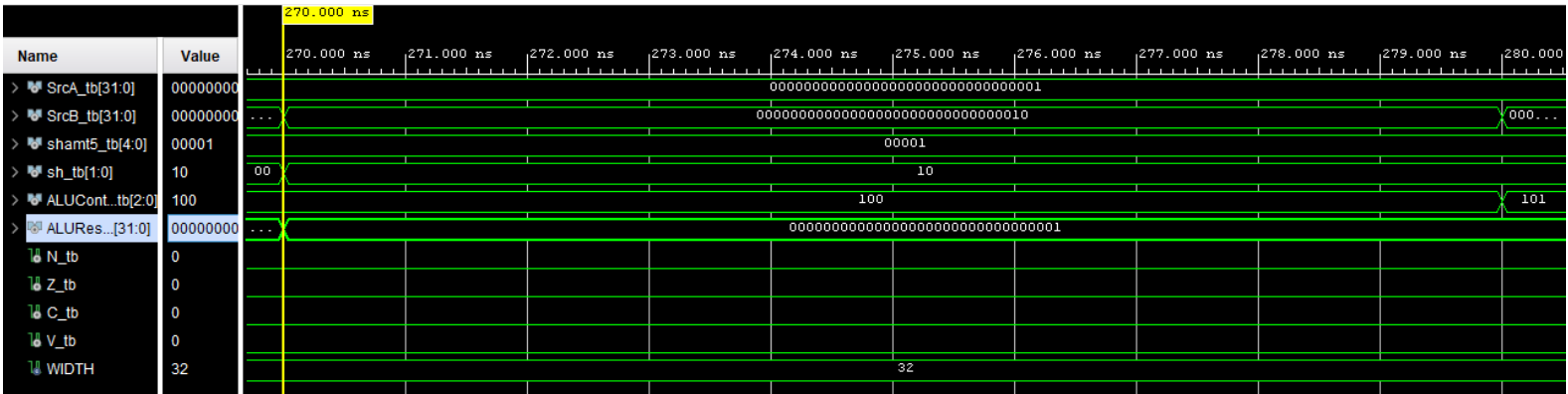
Εικόνα 2.18 Behavioral simulation για την εντολή EOR.

Στο 7^ο test case στους χρόνους 260 ns με 270 ns γίνεται λογική αριστερή μετατόπιση του 1 κατά 1 θέση, αναμένεται αποτέλεσμα 2 και όλα τα flags να είναι 0. Στην εικόνα 2.19 του behavioral simulation για την εντολή LSL της ALU αριστερή μετατόπιση παρατηρείται σωστά η αλλαγή των σημάτων SrcA_tb, SrcB_tb, ALUControl_tb, ALUResult_tb, shamt5_tb, sh_tb, N_tb, Z_tb, C_tb και V_tb.



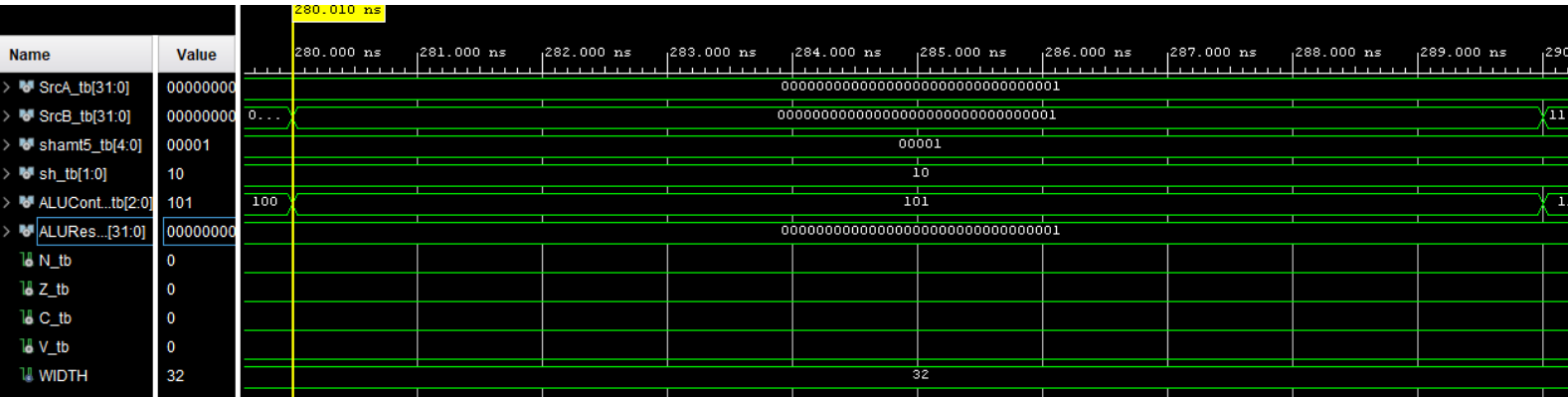
Εικόνα 2.19 Behavioral simulation για την εντολή LSL.

Στο 8^ο test case στους χρόνους 270 ns με 280 ns γίνεται αριθμητική δεξιάς μετατόπιση με το πιο σημαντικό bit να είναι 0, αναμένεται αποτέλεσμα 00000000000000000000000000000001. Στην εικόνα 2.20 του behavioral simulation για την εντολή ASR της ALU αριστερή μετατόπιση παρατηρείται σωστά η αλλαγή των σημάτων SrcA_tb, SrcB_tb, ALUControl_tb, ALUResult_tb, shamt5_tb, N_tb, Z_tb, C_tb και V_tb.



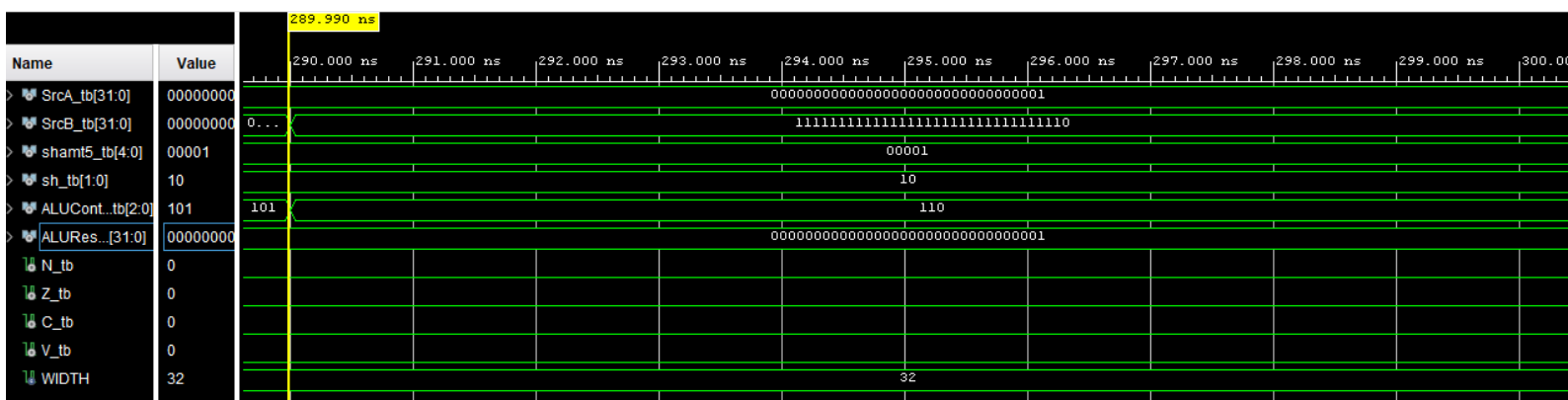
Εικόνα 2.20 Behavioral simulation για την εντολή ASR.

Στο 9^ο test case στους χρόνους 280 ns με 290 ns γίνεται μεταφορά του 1, αναμένεται αποτέλεσμα 1 και όλα τα flags να είναι 0. Στην εικόνα 2.21 του behavioral simulation για την εντολή MOV της ALU παρατηρείται σωστά η αλλαγή των σημάτων SrcA_tb, SrcB_tb, ALUControl_tb, ALUResult_tb, shamt5_tb, sh_tb, N_tb, Z_tb, C_tb και V_tb.



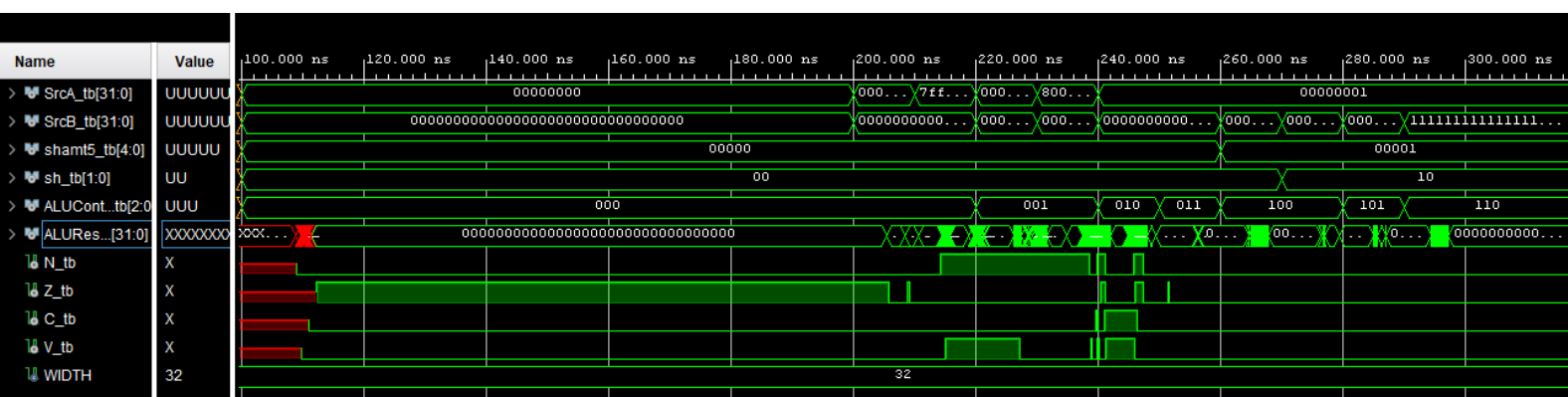
Εικόνα 2.21 Behavioral simulation για την εντολή MOV.

Στο 10^ο test case από το 290 ns μέχρι το τέλος της προσομοίωσης γίνεται μεταφορά του -2, αναμένεται αποτέλεσμα 1 και όλα τα flags να είναι 0. Στην εικόνα 2.22 του behavioral simulation για την εντολή MVN της ALU παρατηρείται σωστά η αλλαγή των σημάτων SrcA_tb, SrcB_tb, ALUControl_tb, ALUResult_tb, shamt5_tb, sh_tb, N_tb, Z_tb, C_tb και V_tb.



Εικόνα 2.22 Behavioral simulation για την εντολή MVN.

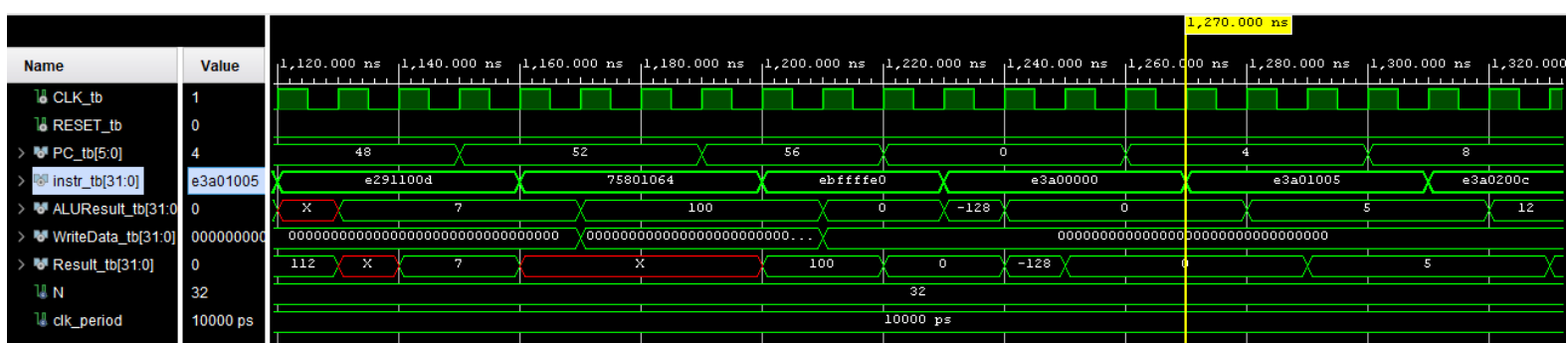
Από το Post-implementation timing simulation παρατηρείται μεγάλη καθυστέρηση διάδοσης, η οποία φαίνεται και στην εικόνα 2.23.



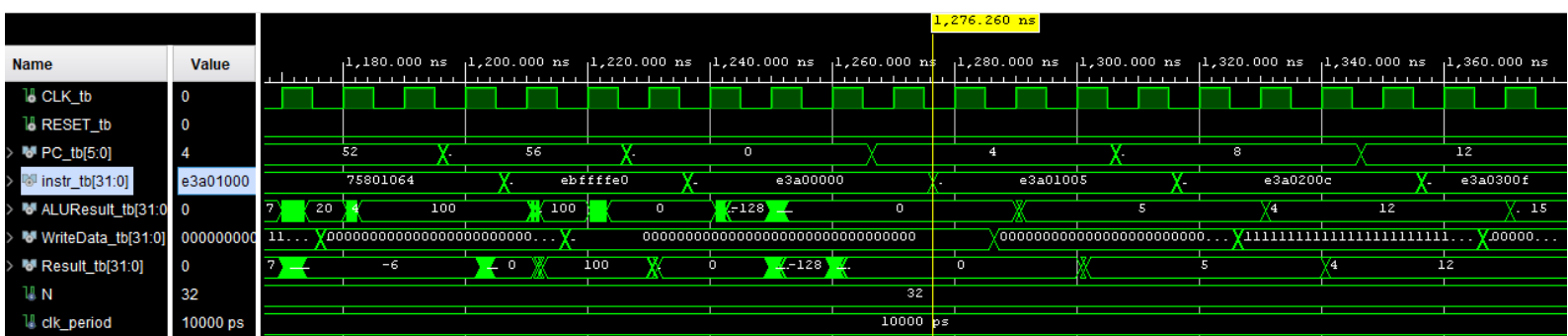
Εικόνα 2.23 Post-implementation timing simulation για την ALU.

2.3. Διαγράμματα χρονισμού του επεξεργαστή

Για να επαληθευθεί η ορθή σχεδίαση του επεξεργαστή, πρέπει πρώτα να επαληθευθεί η σωστή λειτουργία όλων των στοιχείων που τον αποτελούν. Για κάθε στοιχείο του Datapath (Register WE, Register Reset WE, Inc4 Adder, ROM, Register File, Mux 2 to 1, Mux 3 to 1, Extend, Alu, RAM) έχει φτιαχτεί πρόγραμμα δοκιμής το οποίο επιβεβαιώνει την σωστή λειτουργία του. Συγκεκριμένα για η σωστή λειτουργία του Register File και της ALU εξετάστηκε αναλυτικά προηγουμένως. Για το Datapath δεν έχει φτιαχτεί ολοκληρωτικό testbench, αφού λειτουργούν testbench όλων των components του Datapath και εξετάστηκε μέσω του επεξεργαστή ότι λειτουργεί και αυτό. Επιβεβαιώθηκε επίσης η σωστή λειτουργία των components Instruction Decoder και Conditional Logic του Control Unit μέσα από τα δικά τους προγράμματα δοκιμής, ενώ το Finite State Machine δεν έχει δικό του πρόγραμμα δοκιμής, αλλά η σωστή λειτουργία του εξετάστηκε και επιβεβαιώθηκε κατά την εξέταση και επιβεβαίωση της σωστής λειτουργίας ολόκληρου του Control Unit με το πρόγραμμα δοκιμής του. Ο έλεγχος της ορθής λειτουργίας του επεξεργαστή δεν καθορίζεται από το testbench του, καθώς αυτό καθορίζει μόνο το σήμα RESET και το σήμα της χρονικής περιόδου του ρολογιού, αλλά από το πρόγραμμα δοκιμής Assembly που εξετάστηκε προηγουμένως και είναι αποθηκευμένο στην Μνήμη Εντολών ROM. Το πρόγραμμα αυτό, όπως αναφέρθηκε και στην ανάλυση του, έχει σχεδιαστεί ώστε να τρέξει ολόκληρο και όταν τελειώσει να επιστρέψει στην αρχή του, δηλαδή στην πρώτη εντολή. Ο συνολικός χρόνος εκτέλεσης του προγράμματος, δηλαδή μέχρι να επιστρέψει στην αρχή του και να εκτελεστεί για δεύτερη φορά η πρώτη εντολή του, είναι 1.270.000 ns, όπως φαίνεται από το Behavioral Simulation στην εικόνα 2.24. Στην εικόνα 2.25 φαίνεται ο καθυστέρηση 6,260ns της εκτέλεσης του προγράμματος από το Post-implementation timing simulation.

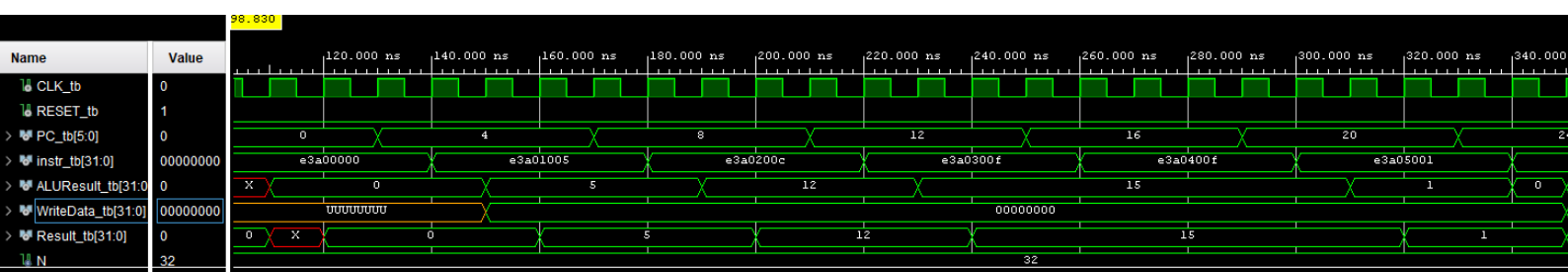


Εικόνα 2.24 Χρόνος εκτέλεσης του προγράμματος δοκιμής του επεξεργαστή από Behavioral simulation.

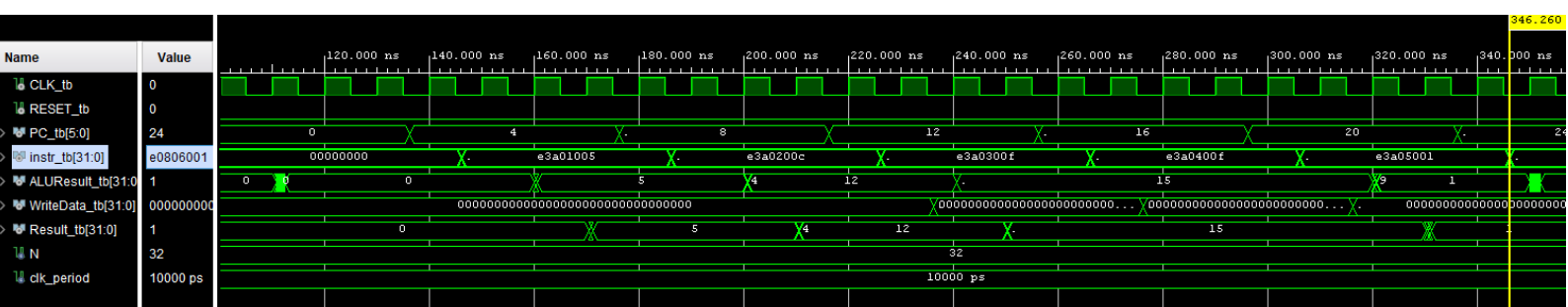


Εικόνα 2.25 Καθυστέρηση εκτέλεσης του προγράμματος δοκιμής του επεξεργαστή από το Post-implementation timing simulation.

Στην εικόνα 2.26 βλέπουμε από το Behavioral Simulation την σωστή απόδοση της τιμής '0' στον R0 για την εντολή E3A00000, την σωστή απόδοση της τιμής '5' στον R1 για την εντολή E3A01005, την σωστή απόδοση της τιμής '12' στον R2 για την εντολή E3A0200C, την σωστή απόδοση της τιμής '15' στον R3 για την εντολή E3A0300F, την σωστή απόδοση της τιμής '15' στον R4 για την εντολή E3A0400F και την σωστή απόδοση της τιμής '1' στον R5 για την εντολή E3A05001. Στην εικόνα 2.27 βλέπουμε από το Post-implementation timing simulation την καθυστέρηση διάδοσης για τις εντολές αυτές.



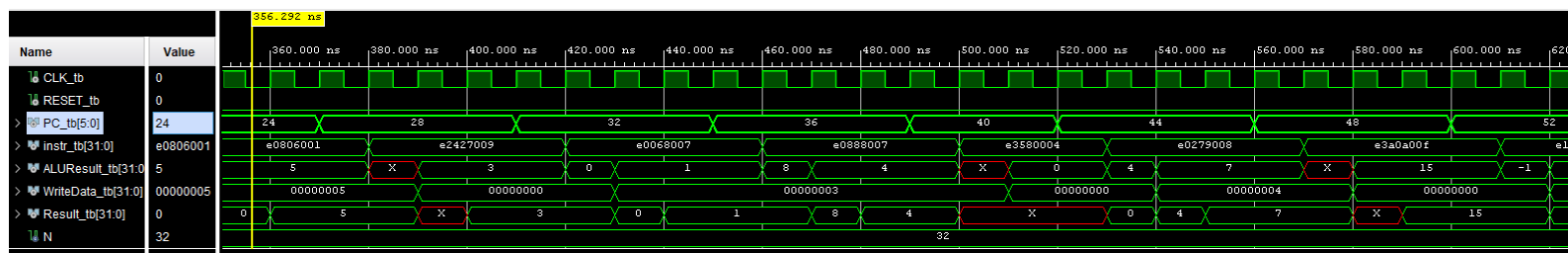
Εικόνα 2.26 Behavioral simulation του επεξεργαστή.



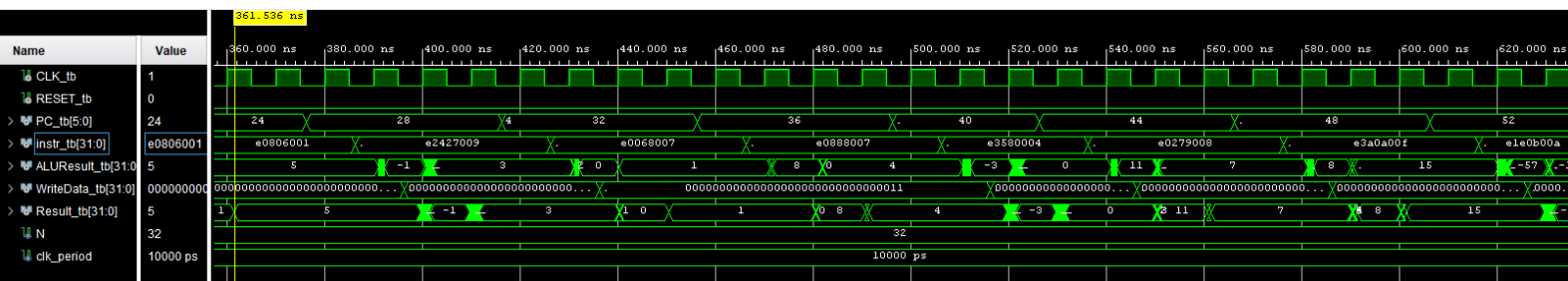
Εικόνα 2.27 Post-implementation timing simulation του επεξεργαστή.

Στην εικόνα 2.28 βλέπουμε από το Behavioral Simulation την σωστή απόδοση της τιμής '5' στον R6 για την εντολή E0806001, την σωστή απόδοση της τιμής '3' στον R7 για την εντολή E2427009, την σωστή απόδοση της τιμής '12' στον R2 για την εντολή E3A0200C, την σωστή απόδοση της τιμής '1' στον R8 για την εντολή E0068007, την σωστή απόδοση της τιμής '4' στον R8 για την εντολή E0888007, την σωστή εντολή σύγκρισης για την εντολή E3580004, την σωστή απόδοση της τιμής '7' στον R9 για

την εντολή E0279008 και την σωστή απόδοση της τιμής '15' στον R10 για την εντολή E3A0A00F. Στην εικόνα 2.29 βλέπουμε από το Post-implementation timing simulation την καθυστέρηση διάδοσης για τις εντολές αυτές.

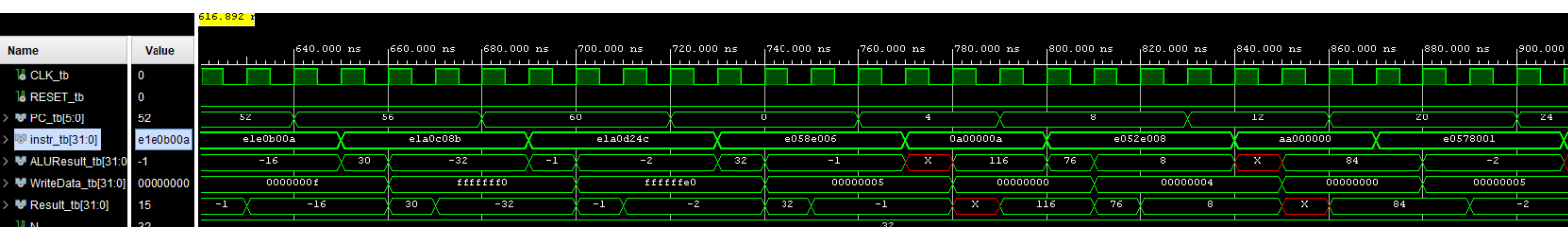


Εικόνα 2.28 Behavioral simulation του επεξεργαστή.

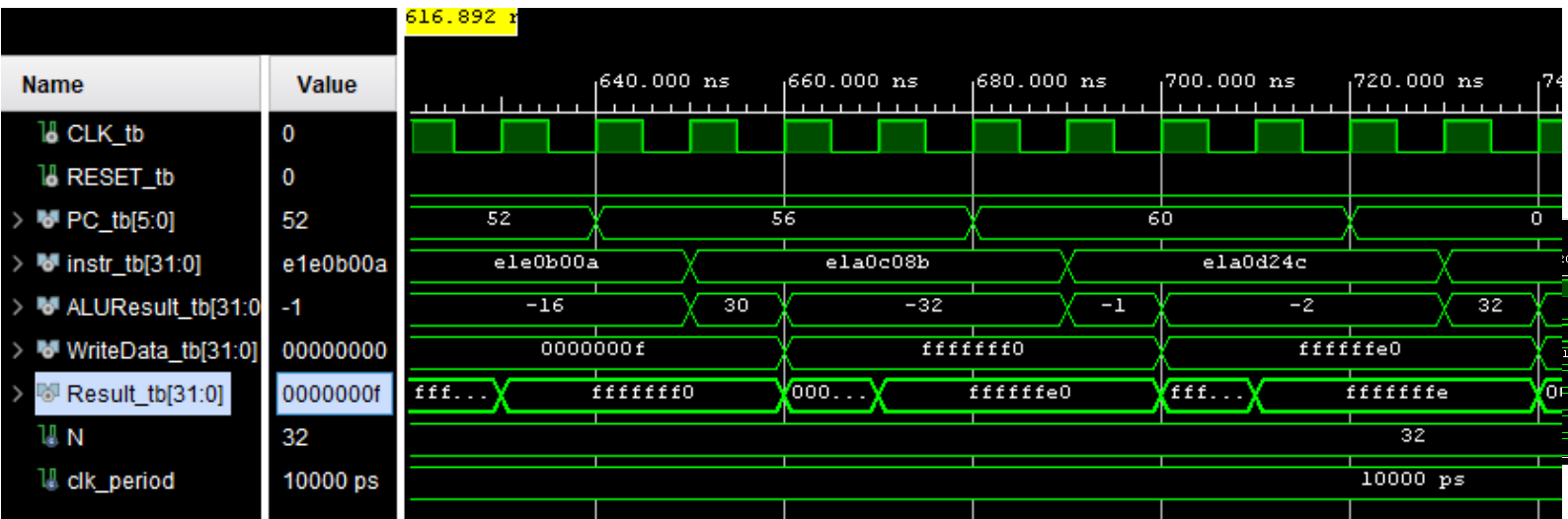


Εικόνα 2.29 Post-implementation timing simulation του επεξεργαστή.

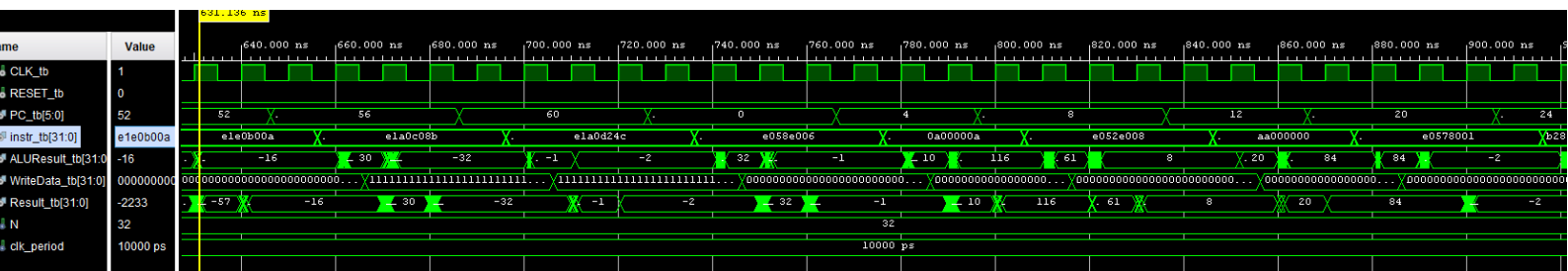
Στην εικόνα 2.30 και 2.31 βλέπουμε από το Behavioral Simulation την σωστή απόδοση της τιμής '4294967280' στον R11 για την εντολή E1E0B00A, την σωστή απόδοση της τιμής '4294967264' στον R12 για την εντολή E1A0C08B, την σωστή απόδοση της τιμής '4294967294' στον R13 για την εντολή E1A0D24C, την σωστή απόδοση της τιμής '-1' στον R14 για την εντολή E058E006, την εντολή διακλάδωσης 0A00000A η οποία σωστά δεν θα εκτελεστεί, την σωστή απόδοση της τιμής '8' στον R14 για την εντολή E052E008, την σωστή εκτέλεση της εντολής διακλάδωσης AA000000 και την και την σωστή απόδοση της τιμής '-2' στον R8 για την εντολή E0578001, ενώ προσπερνάται σωστά η εκτέλεση της εντολής E2808000 μετά την εντολή AA000000. Στην εικόνα 2.32 βλέπουμε από το Post-implementation timing simulation την καθυστέρηση διάδοσης για τις εντολές αυτές.



Εικόνα 2.30 Behavioral simulation του επεξεργαστή.

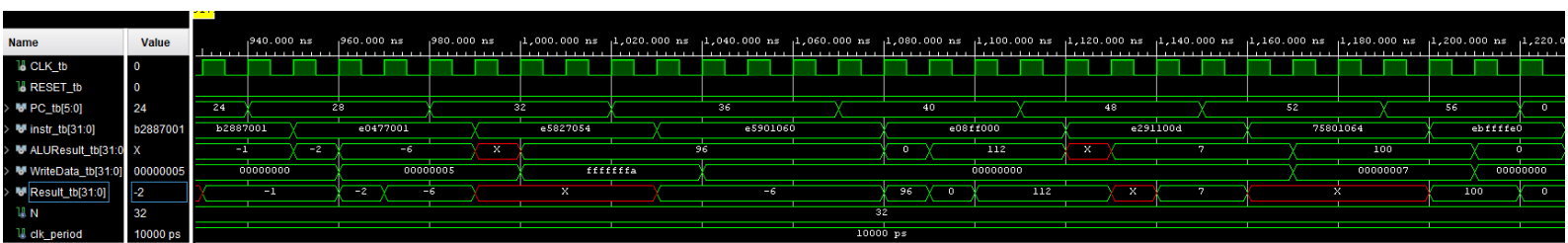


Εικόνα 2.31 Behavioral simulation του επεξεργαστή.

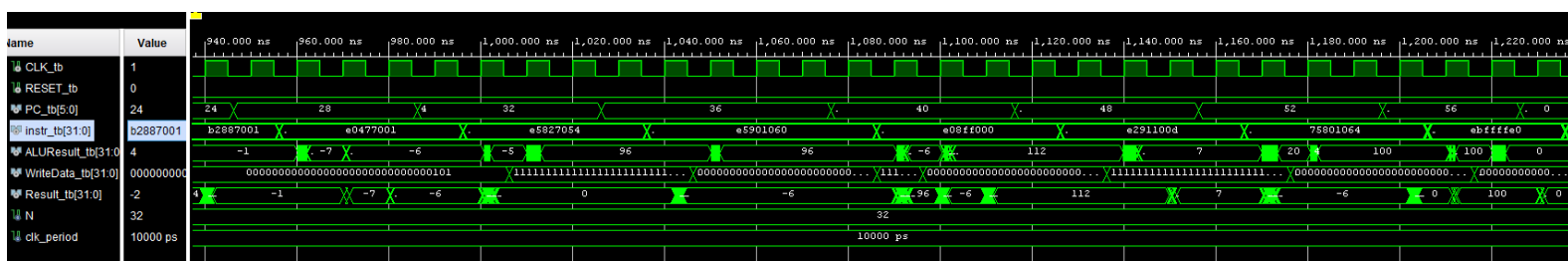


Εικόνα 2.32 Post-implementation timing simulation του επεξεργαστή.

Στην εικόνα 2.33 βλέπουμε από το Behavioral Simulation την σωστή απόδοση της τιμής '-1' στον R7 για την εντολή B2887001, την σωστή απόδοση της τιμής '-6' στον R7 για την εντολή E0477001, την σωστή απόδοση της τιμής '-6' στην θέση μνήμης 96 για την εντολή E5827054, την σωστή απόδοση της τιμής '-6' στον R1 για την εντολή E5901060, την σωστή αύξηση του PC Register για την εντολή E08FF000 και το σωστό προσπέρασμα της επόμενης εντολής E280200E, την σωστή απόδοση της τιμής '7' στον R1 για την εντολή E291100D, την σωστή ανάθεση της τιμής 7 στην θέση μνήμης 100 για την εντολή 75801064 και την σωστή εκτέλεση της εντολής διακλάδωσης EBFFFFE0. Στην εικόνα 2.34 βλέπουμε από το Post-implementation timing simulation την καθυστέρηση διάδοσης για τις εντολές αυτές.

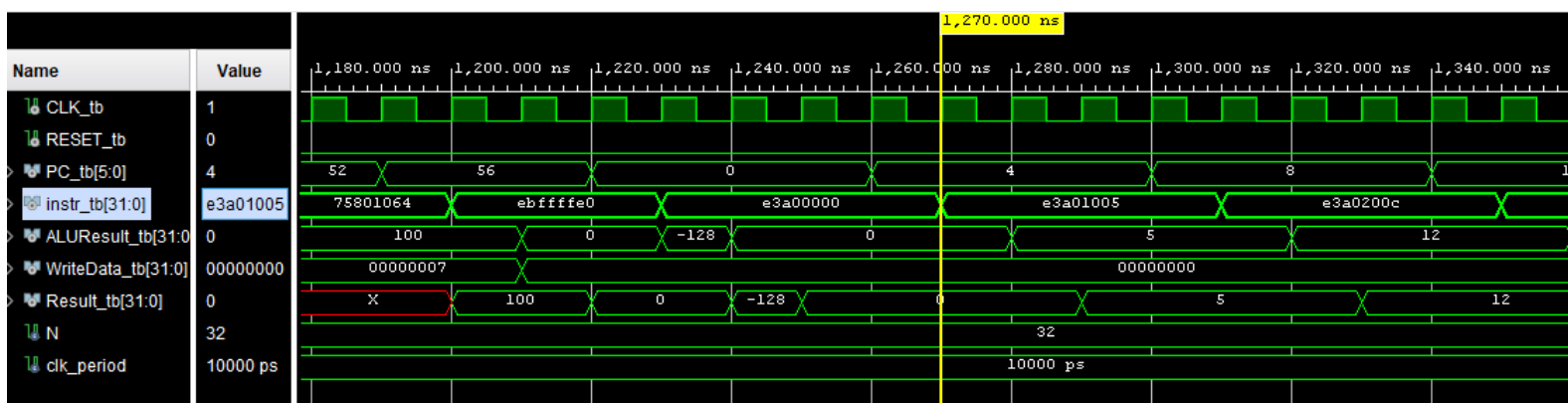


Εικόνα 2.33 Behavioral simulation του επεξεργαστή.



Εικόνα 2.34 Post-implementation timing simulation του επεξεργαστή.

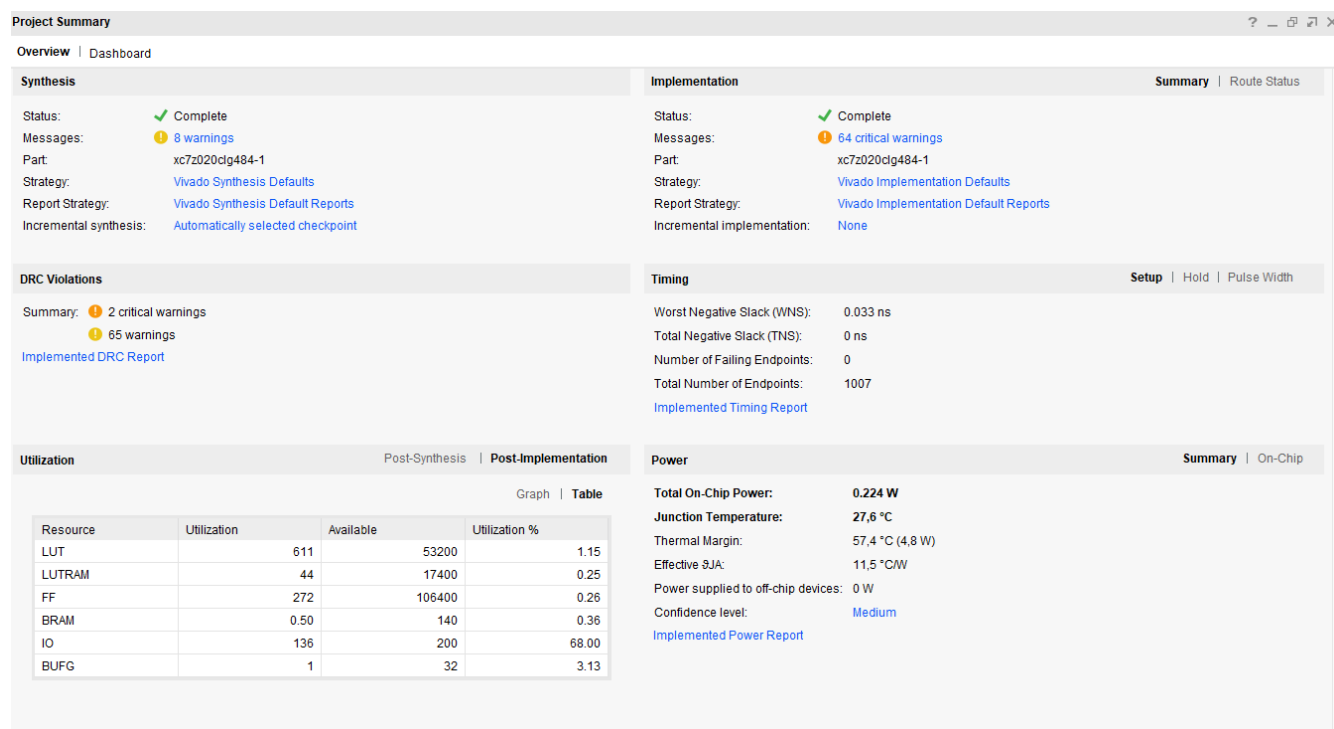
Τέλος, στην εικόνα 2.35 βλέπουμε από το Behavioral Simulation το πρόγραμμα να ξαναπηγαίνει στην πρώτη του εντολή. Επομένως επιβεβαιώνεται η σωστή λειτουργία του επεξεργαστή.



Εικόνα 2.35 Behavioral simulation του επεξεργαστή.

3. Ανάλυση των αποτελεσμάτων της σύνθεσης και της υλοποίησης του επεξεργαστή

Το project summary της εικόνας 3.1 που δημιουργήθηκε μετά το implementation παρουσιάζει περιληπτικά τα βασικά χαρακτηριστικά της σχεδίασης του επεξεργαστή πολλών κύκλων. Τα warnings που φαίνονται στην εικόνα αφορούν το αρχείο zedboard με τα constraints, το οποίο χρησιμοποιούμε μόνο για την περίοδο του σήματος του ρολογιού.



Εικόνα 3.1 Project summary του επεξεργαστή.

Στο παράθυρο Utilization της εικόνας 3.1 παρουσιάζεται ο αριθμός και ο τύπος των πόρων που χρησιμοποιήθηκαν για την υλοποίηση του επεξεργαστή, ενώ στην εικόνα 3.2 παρουσιάζεται αναλυτικά η χρήση των πόρων ανά component. Ο επεξεργαστής χρησιμοποιεί ένα BUFG, δηλαδή ένα buffer που σχετίζεται με την περίοδο του σήματος του ρολογιού. Επιπλέον, χρησιμοποιεί 136 IO, δηλαδή 136 συνολικά bit για τα σήματα εισόδου και εξόδου του.






Name	Slice LUTs (53200)	Slice Registers (106400)	F7 Muxes (26600)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)	Block RAM Tile (140)	Bonded IOB (200)	OLOGIC (200)	BUFGCTRL (32)
Processor	611	272	1	223	567	44	0.5	136	70	1
ControlUnit_component (Control_Unit)	36	14	0	27	36	0	0	0	0	0
FSM (Finite_State_Machine)	36	14	0	27	36	0	0	0	0	0
Datapath_component (Datapath)	575	258	1	222	531	44	0.5	0	70	0
A_Register (Register_RESET)	2	32	0	29	2	0	0	0	0	0
ALU_Unit (ALU)	16	0	0	27	16	0	0	0	0	0
B_Register (Register_RESET_0)	0	32	0	26	0	0	0	0	32	0
Data_Memory (DM_RAM)	0	0	0	0	0	0	0.5	0	0	0
I_Register (Register_RESET_1)	4	25	0	22	4	0	0	0	0	0
Instruction_Register (Register_RESE	440	32	1	154	440	0	0	0	32	0
Memory_Address_Register (Register	0	5	0	2	0	0	0	0	0	0
PC_Register (Register_RESET_WE_	33	32	0	50	33	0	0	0	6	0
PCPlus4_Register (Register_RESET_	5	32	0	13	5	0	0	0	0	0
PCPlus8 (INC4_Adder)	28	0	0	20	28	0	0	0	0	0
RegisterFile (Register_File)	44	0	0	11	0	44	0	0	0	0
S_Register (Register_RESET_4)	0	32	0	31	0	0	0	0	0	0
Status_Register (Register_RESET_W	3	4	0	4	3	0	0	0	0	0
Write_Data_Register (Register_RESE	0	32	0	10	0	0	0	0	0	0






Εικόνα 3.2 Πλήρης Utilization του επεξεργαστή.

Η υλοποίηση του επεξεργαστή πολλών κύκλων απαιτεί την χρήση συνολικά 272 Flip Flops, τα οποία χρησιμοποιούν συγκεκριμένα οι καταχωρήτες του επεξεργαστή στην παρουσιάζεται στην εικόνα 3.3. Στην εικόνα 3.4 φαίνεται ότι ο επεξεργαστής, και συγκεκριμένα η μνήμη δεδομένων RAM, χρησιμοποιεί μισού στοιχείου BRAM. Στην εικόνα 3.5 φαίνεται ότι ο επεξεργαστής, και συγκεκριμένα ο Register File, χρησιμοποιεί 44 στοιχεία LUTRAM. Τέλος, στην εικόνα 3.6 παρουσιάζεται η συνολική χρήση στοιχείων από τον επεξεργαστή. Ο επεξεργαστής χρησιμοποιεί συνολικά 652 LUTs, τα οποία είναι της μορφής LUT6, LUT5, LUT4, LUT3, LUT2 και LUT1. Χρησιμοποιεί επίσης ένα στοιχείο μνήμης RAMB18E1, καθώς και κατανεμημένες μνήμες RAMS32 και RAMD32. Επιπλέον, χρησιμοποιεί μονάδες CARRY4 λόγω της ύπαρξης κρατούμενων, ενώ χρησιμοποιεί και έναν πολυπλέκτη MUXF7.

Name	Used
Processor	272
ControlUnit_component (Control_Unit)	14
FSM (Finite_State_Machine)	14
Datapath_component (Datapath)	258
A_Register (Register_RESET)	32
B_Register (Register_RESET_0)	32
I_Register (Register_RESET_1)	25
Instruction_Register (Register_RESE	32
Memory_Address_Register (Register	5
PC_Register (Register_RESET_WE_	32
PCPlus4_Register (Register_RESET_	32
S_Register (Register_RESET_4)	32
Status_Register (Register_RESET_W	4
Write_Data_Register (Register_RESE	32

Εικόνα 3.3 Flip-Flop Utilization του επεξεργαστή.






Block RAM Tile

Name	Used
<div>   Processor </div>	0.500
<div> <div>   Datapath_component (Datapath) </div> <div>  Data_Memory (DM_RAM) </div> </div>	0.500
	0.500

Εικόνα 3.4 *BRAM Utilization του επεξεργαστή.*

LUT as Memory

Name	Used
<div> Processor </div>	44
<div> <div> Datapath_component (Datapath) </div> <div> RegisterFile (Register_File) </div> </div>	44
	44

Εικόνα 3.5 *LUTRAM Utilization του επεξεργαστή.*

Primitives

Ref Name	Used	Functional Category
LUT6	366	LUT
FDRE	341	Flop & Latch
OBUF	134	IO
LUT5	116	LUT
LUT3	93	LUT
RAMD32	68	Distributed Memory
LUT4	67	LUT
CARRY4	36	CarryLogic
RAMS32	20	Distributed Memory
LUT2	7	LUT
LUT1	3	LUT
IBUF	2	IO
RAMB18E1	1	Block Memory
MUXF7	1	MuxFx
FDSE	1	Flop & Latch
BUFG	1	Clock

Εικόνα 3.6 *Utilization του επεξεργαστή.*

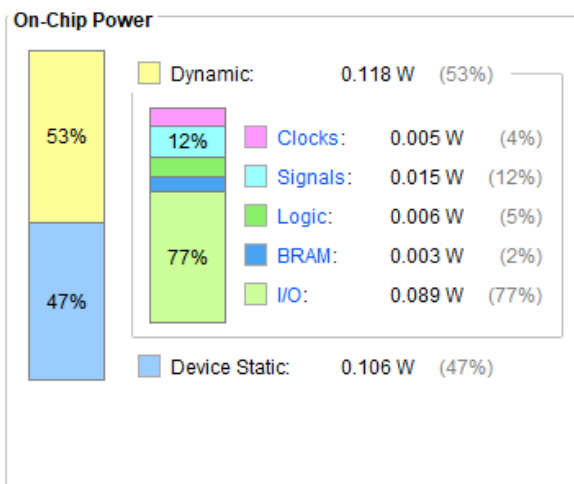
Στην εικόνα 3.7 υπολογίζεται η κατανάλωση του επεξεργαστή πολλών κύκλων, η οποία ισούται με 0,224 W. Το 47% αυτής της κατανάλωσης είναι στατική και καταναλώνεται από τον επεξεργαστή ακόμα και αν βρίσκεται σε idle κατάσταση, ενώ το 53% αυτής της κατανάλωσης είναι δυναμική και εξαρτάται από την λειτουργία του κυκλώματος και το πόσο παραπάνω ισχύ χρειάζεται ο επεξεργαστής. Το junction temperature είναι η υψηλότερη θερμοκρασία που μπορεί να έχουν οι ημιαγωγοί στον επεξεργαστή και ισούται με 27,6°C, ενώ ο επεξεργαστής παρατηρείται ότι έχει ένα εύρος θερμοκρασίας 57,4°C, όπου εύρος θερμοκρασίας είναι η διαφορά της μέγιστης θερμοκρασίας που αντέχει ο επεξεργαστής από την μέση θερμοκρασία του.

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	0.224 W
Design Power Budget:	Not Specified
Power Budget Margin:	N/A
Junction Temperature:	27,6°C
Thermal Margin:	57,4°C (4,8 W)
Effective θ_{JA} :	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity



Εικόνα 3.7 *Power Report του επεξεργαστή.*

4. Περιγραφή της λειτουργίας της εντολής ROR

Η εντολή ROR (Rotate Right) στην αρχιτεκτονική ARM είναι μια εντολή επεξεργασίας δεδομένων που εκτελεί περιστροφή των bits μιας λέξης προς τα δεξιά κατά έναν ακέραιο αριθμό. Η ROR παίρνει την τιμή από έναν καταχωρητή, την περιστρέφει προς τα δεξιά για έναν συγκεκριμένο αριθμό bits και αποθηκεύει το αποτέλεσμα σε έναν καταχωρητή προορισμού. Κατά την περιστροφή, τα bits βγαίνουν από το λιγότερο σημαντικό bit και επιστρέφουν στο πιο σημαντικό bit.

Στον παρακάτω πίνακα παρουσιάζεται η μορφή της εντολής ROR. Το πεδίο cond είναι το μνημονικό συνθήκης και μπορεί να έχει τον κωδικό οποιουδήποτε μνημονικού που αναφέρθηκε προηγουμένως στην αναφορά. Το πεδίο op έχει την τιμή '00', καθώς η ROR είναι εντολή επεξεργασίας δεδομένων. Το πεδίο l είναι 0, όμοια με τις εντολές LSL και ASR. Το πεδίο cmd έχει την τιμή '1101', όπως και στις εντολές LSL και ASR, και καθορίζει τον τύπο εντολής. Το πεδίο S είναι αυτό που επιτρέπει ή όχι την ενημέρωση των σημαιών από την εντολή, ανάλογα με την τιμή του πεδίου αυτού '0' ή '1' αντίστοιχα. Το πεδίο Rn είναι ένας καταχωρητής προέλευσης, ο οποίος όμως έχει την τιμή '0000' γιατί δεν χρησιμοποιείται. Το πεδίο Rd είναι ο τελεστής προορισμού. Το πεδίο shamt5 καθορίζει τον ποσό της περιστροφής, ομοίως με τις εντολές LSL και ASR. Οι εντολές LSL, ASR και ROR διαφοροποιούνται μεταξύ τους με το πεδίο sh, που έχει την τιμή '11' μόνο για την εντολή ROR. Τέλος, το πεδίο Rm είναι ο τελεστής προέλευσης.

Εντολή	Τύπος	Instr31:28 cond	Instr27:26 op	Instr25 l	Instr24:21 cmd	Instr20 S	Instr19:16 Rn	Instr15:12 Rd	Instr11:7 shamt5	Instr6:5 sh	Instr4 0	Instr3:0 Rm
ROR	DP Imm	XXXX	00	0	1101	X	0000	XXXX	XXXXX	11	-	XXXX

Όμοια με τις υπόλοιπες εντολές, το Control Unit θα παράξει τα κατάλληλα σήματα για την εκτέλεση της εντολής ROR. Η εντολή αυτή είναι παρόμοια με τις εντολές LSL και ASR, και έτσι θα έχουν και παρόμοιες υλοποιήσεις. Η εντολή ROR διαφέρει από τις άλλες εντολές από τα πεδία op = '00', l = '0', cmd = '1101' και sh = '11'. Το Conditional Logic θα παραμείνει ίδιο, καθώς η λειτουργία του είναι ίδια για όλες τις εντολές, δέχεται μνημονικά συνθήκης και παράγει τα κατάλληλα flags. Το Finite State Machine παραμένει επίσης ίδιο, καθώς η εντολή ROR εκτελείται σε ίδιο αριθμό βημάτων με τις εντολές LSL και ASR, οπότε δεν χρειάζεται να προστεθούν νέες καταστάσεις στο Finite State Machine. Ο Instruction Decoder, από την άλλη, θα πρέπει να αλλάξει και να προστεθούν σε αυτόν κατάλληλες συνθήκες και να επιλεγούν για την εντολή ROR κατάλληλες τιμές για τα σήματα RegSrc, ALUSrc, ImmSrc, ALUControl, MemtoReg και NoWrite_in. Το σήμα RegSrc θα παίρνει τιμή '00X', όπως παίρνουν και οι εντολές LSL και ASR. Καθώς το πεδίο l είναι 0 για την εντολή ROR, το σήμα ALUSrc θα έχει τιμή 0, όπως και στις εντολές LSL και ASR. Λόγω

του πεδίου I και ότι η εντολή ROR δεν χρησιμοποιεί το Extend Unit, δεν μας ενδιαφέρει η τιμή του σήματος ImmSrc, όπως και στις εντολές LSL και ASR. Το σήμα ALUControl μπορεί είτε να πάρει την τιμή '111' που δεν χρησιμοποιείται από κάποια εντολή, είτε να πάρει την τιμή '100' όπως οι εντολές LSL και ASR και να ξεχωρίζεται από το πεδίο sh με τιμή '11' από τις εντολές LSL και ASR για την τιμή αυτή του ALUControl. Όπως και για τις άλλες εντολές επεξεργασίας δεδομένων εκτός από την CMP, το σήμα MemtoReg για την εντολή ROR θα έχει τιμή '0'. Τέλος, το σήμα NoWrite_in για την εντολή ROR θα έχει τιμή '0', όπως και για τις άλλες εντολές επεξεργασίας δεδομένων εκτός από την CMP.