Σελίδα τίτλου

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

ΤΜΗΜΑΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΣΧΕΔΙΑΣΗ ΤΗΣ ΑΡΙΘΜΗΤΙΚΗΣ ΚΑΙ ΛΟΓΙΚΗΣ ΜΟΝΑΔΑΣ (ALU)

Ευάγγελος Κοσκίνας

Επιβλέπων: Φώτιος Βαρτζιώτης

Επίκουρος Καθηγητής

Ιωάννινα, Ιούλιος, 2022

DESIGN OF THE ARITHMETIC AND LOGIC UNIT (ALU)

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, 6 Ιουλίου 2022

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής

   Φώτιος Βαρτζιώτης,

2. Μέλος επιτροπής

   Ελευθέριος Στεργίου,

3. Μέλος επιτροπής

   Σπυριδούλα Μαργαρίτη,

Δήλωσημηλογοκλοπής

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

Κοσκίνας Ευάγγελος

Υπογραφή

ΕΥΧΑΡΙΣΤΙΕΣ

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω, τον επιβλέποντα καθηγητή μου κ. Φώτιο Βαρτζιώτη για την άρτια καθοδήγηση, την υποστήριξη και τις συμβουλές που μου προσέφερε καθ' όλη τη διάρκεια της Πτυχιακής μου Εργασίας.

Ευχαριστώ επίσης, την οικογένεια μου που στάθηκε δίπλα μου, που με στήριξε με υπομονή καθ' όλο το διάστημα της φοίτησής μου, καθώς και κατά τη διάρκεια της συγγραφής της Πτυχιακής μου.

ΠΕΡΙΛΗΨΗ

Σκοπός της εν λόγω πτυχιακής εργασίας είναι η σχεδίαση και η κατασκευή της αριθμητικής και λογικής μονάδας 32 μπιτ. Ο σχεδιασμός της μονάδας υλοποιήθηκε στο πρόγραμμα Intel Quartus Prime Lite Edition. Η εν λόγω μονάδα που σχεδιάστηκε πραγματοποιεί αριθμητικές πράξεις, όπως πρόσθεση, αφαίρεση, πολλαπλασιασμό, διαίρεση, καθώς και λογικές πράξεις ANDκαι OR. Αρκετά από τα κυκλώματα της μονάδας δημιουργήθηκαν σχηματικά, ενώ κάποια άλλα λόγω πολυπλοκότητας υλοποιήθηκαν με την γλώσσα περιγραφής υλικού VHDL.

Στα κεφάλαια της πτυχιακής εργασίας που θα ακολουθήσουν θα αναλύσουμε όλα τα κυκλώματα που υλοποιήσαμε καθώς και τις σχετικές εξομοιώσεις προκειμένου να εξετάσουμε αν τα κυκλώματα μας λειτουργούν σωστά. Με το πέρας της υλοποίησης της εργασίας κατανοήσαμε καλύτερα την σχεδίαση των ψηφιακών κυκλωμάτων καθώς εξοικειωθήκαμε με το πρόγραμμα Intel Quartus και με την γλώσσα VHDL. Τέλος, θα αναφερθούμε στις δυσκολίες που αντιμετωπίσαμε καθώς και σε μελλοντικές βελτιώσεις αυτής.

Λέξεις-κλειδιά: Αριθμητική και λογική μονάδα, σχεδίαση, εξομοίωση, ψηφιακά κυκλώματα

ABSTRACT

The purpose of this thesis is to design and construct a 32-bit arithmetic and logic unit. The unit was designed in the Intel Quartus Prime Lite Edition program. The designed unit performs arithmetic operations such as addition, subtraction, multiplication, division as well as logical operations ANDand OR. Several of the unit's circuits were created schematically, while others due to their complexity were implemented with the VHDL hardware description language.

In the chapters of the thesis that will follow we will analyze all the circuits we have implemented as well as, the relevant simulations in order to examine if our circuits work properly. By completing theproject we were able to understandbetter the design of digital circuits as we became familiar with the Intel Quartus program and the VHDL language. Finally, we will make reference to difficulties faced as well as future improvements of the implementation.

Keywords: Arithmetic logic unit, simulation, VHDL, Quartus, digital circuits

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΑΤΑΛΟΓΟΣ ΔΙΑΓΡΑΜΜΑΤΩΝ/ΕΙΚΟΝΩΝ

ΠΙΝΑΚΑΣ ΣΥΝΤΟΜΟΓΡΑΦΙΩΝ

ALU..……ArithmeticLogicUnit

HDL..……..Hardware Description Language

VHSIC..……..Very High Speed Integrated Circuit Program

VHDL..……..Very High Speed Integrated Circuit Hardware Description Language

IEEE..……..Institute of Electrical and Electronics Engineers

CAD..……Computer Aided Design

FPGA..……..Field Programmable Gate Arrays

CPLD..……..Complex Programmable Logic Devices

ASIC..……..Application Specific Integrated Circuit

ΕΙΣΑΓΩΓΗ

Στην καρδιά ενός υπολογιστή, ήτοι στον επεξεργαστή του, υπάρχει μια μονάδα που εκτελεί τις αριθμητικές και λογικές πράξεις του συστήματος. Αυτή η μονάδα λέγεται ALU δηλαδή Arithmetic Logic Unit, όπου στα ελληνικά σημαίνει αριθμητική και λογική μονάδα. Υπάρχουνκαι επεξεργαστές που διαθέτουν παραπάνω από μια μονάδα ALU, όπως εκείνοι που διαθέτουν μια μονάδα που ελέγχει τις λειτουργίες του σταθερού σημείου και μια άλλη που ελέγχει τις λειτουργίες της κινητής υποδιαστολής. Η μονάδα ALU για να αναγνωρίσει ποια εντολή θα πρέπει να εκτελέσει έχει μια είσοδο που αποτελείται από μια λέξη εντολής ή αλλιώς έναν κωδικό λειτουργίας και με βάση αυτή εκτελεί την ανάλογη λειτουργία. Επίσης, διαθέτει εισόδους για τους δυαδικούς αριθμούς και έξοδο για το αποτέλεσμα. Η μονάδα ALU εκτελεί αριθμητικές πράξεις όπως πρόσθεση, αφαίρεση, πολλαπλασιασμό, διαίρεση ή λογικές όπως AND, OR, XOR κ.α. [1].[2]

Στην παρούσα πτυχιακή εργασία θα υλοποιήσουμε την αριθμητική και λογική μονάδα 32 μπιτ (bit) η οποία θα εκτελεί αριθμητικές πράξεις πρόσθεσης, αφαίρεσης, πολλαπλασιασμού, διαίρεσης και λογικές πράξεις ANDκαι OR. Η υλοποίηση της μονάδας έγινε στο πρόγραμμα Intel Quartus Prime Lite Edition. Τα περισσότερα στοιχεία της μονάδας έγιναν σχηματικά ενώ κάποια άλλα λόγω πολυπλοκότητας έγιναν με την χρήση της γλώσσας περιγραφής υλικού VHDL.

Η εργασία απαρτίζεται από έξι κεφάλαια. Στο πρώτο κεφάλαιο θα αναφερθούμε γενικά στο πρόγραμμα Intel Quartus Prime, θα μιλήσουμε για τις γλώσσες περιγραφής υλικού και για τις διατάξεις FPGA. Στο δεύτερο κεφάλαιο θα αναφερθούμεστη δημιουργία της μονάδας ALU, καθώς και των στοιχείων από τα οποία αποτελείται. Στο τρίτο κεφάλαιο θα αναλύσουμε την τροποποίηση της μονάδας ALU προσθέτοντας σε αυτήν την γεννήτρια πρόβλεψης κρατουμένου με σκοπό να εκτελέσουμε τις πράξεις της πρόσθεσης και της αφαίρεσης πιο γρήγορα. Ακολούθως, στο τέταρτο και πέμπτο κεφάλαιο θα αναλύσουμε την υλοποίηση του πολλαπλασιασμού και της διαίρεσης. Στο έκτο κεφάλαιο θα ενώσουμε όλες τις μονάδες σε μια. Τέλος, στα συμπεράσματα θα αναφερθούμε στις δυσκολίες που κληθήκαμε να αντιμετωπίσουμε καθώς και σε μελλοντικές βελτιώσεις της.

1.Κεφάλαιο 1

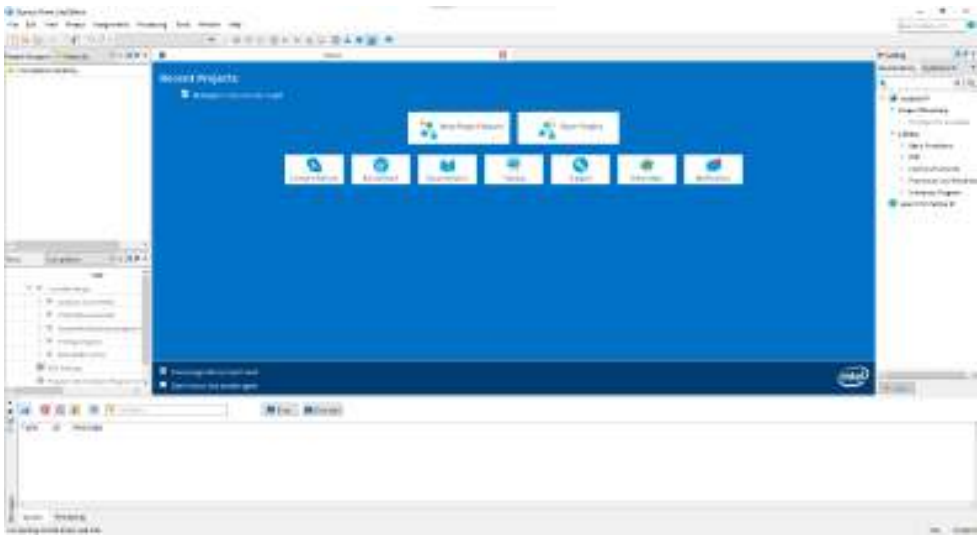
1.1 Πρόγραμμα Intel Quartus Prime

Το Intel Quartus Prime είναι ένα πρόγραμμα σχεδίασης, προγραμματισμού και προσομοίωσης κυκλωμάτων. Έχει αναπτυχθεί από την Intel και είναι διαθέσιμο για λογισμικά Linux και Windows. Ανάλογα με τις απαιτήσεις των σχεδιαστών το πρόγραμμα διατίθεται σε τρεις εκδόσεις, την Intel Quartus Prime Pro Edition, την Intel Quartus Prime Standard Edition και την Intel Quartus Prime Lite Edition. Η έκδοση η οποία χρησιμοποιήσαμε στην παρούσα πτυχιακή εργασία είναι η Intel Quartus Prime Lite Edition καθώς δεν απαιτείται πληρωμή για άδεια χρήσης και χρησιμοποιείται για εκπαιδευτικούς σκοπούς. Πιο συγκεκριμένα ο αριθμός της έκδοσης που χρησιμοποιήθηκε είναι ο Quartus Prime Version 20.1.0 Build 711 06/05/2020 SJ Lite Edition. Επιπρόσθετα, για τον σχεδιασμό η οικογένεια συσκευών που επιλέχθηκε είναι η Cyclone IV E με συσκευή την EP4CE6F17. [3]

Εικόνα 1

Το πρόγραμμα IntelQuartusPrimeLite χρησιμοποιείται για τον σχεδιασμό ψηφιακών κυκλωμάτων για τις διατάξεις FPGA. Ο σχεδιασμόςτους μπορεί να γίνει με δύο τρόπους.Ο πρώτοςείναι σχηματικά, ο οποίος είναι και ο πιο εύκολος, ωστόσο δεν είναι κατάλληλος για περίπλοκα κυκλώματα και ο δεύτερος είναι με την χρήση της γλώσσας περιγραφής υλικού, δηλαδή με την ανάπτυξη πηγαίου κώδικα. Για την υλοποίηση της αριθμητικής και λογικής μονάδας (ALU) έγινε η χρήση και των δύο τρόπων. Αφού έγινε η σύνθεση των κυκλωμάτων με τους παραπάνω τρόπους κατά περίπτωση (άλλοτε με χρήση σχηματικού κώδικά και άλλοτε με χρήση γλώσσας περιγραφής υλικού) ακολούθως εκτελέσαμε την εντολή αποσφαλμάτωσης (Start Compilation) προκειμένω να ελέγξουμε αν λειτουργούν σωστά ή αν θέλουν κάποια διόρθωση. Η διόρθωσηγίνεταισχετικά εύκολα χάρει στον μεταγλωττιστή (compiler) που παρέχει το πρόγραμμα καθώς έχει αυτόματο εντοπισμό λαθών. Στη συνέχεια δημιουργήσαμε αρχεία κυματομορφών της χρονικής και λογικής εξομοίωσης με τις οποίες επιβεβαιώσαμε για ακόμη μια ακόμη φορά την σωστή

λειτουργία τους. Στα επόμενα κεφάλαια που ακολουθούν αναλύουμε με περισσότερη λεπτομέρεια την δημιουργία των προαναφερόμενων αρχείων. [3],[16]



Εικόνα 2

1.2Γλώσσα περιγραφής υλικού VHDL

Όπως προαναφέρθηκε παραπάνω για την υλοποίηση μερικών κυκλωμάτων έγινε η χρήση της γλώσσας περιγραφής υλικού (hardware description language, HDL). Οι HDL έχουν αρκετές ομοιότητες με τις γλώσσες προγραμματισμού διαφέρουν ωστόσο στο ότι είναι προσαρμοσμένες στην περιγραφή των δομών υλικού και των λογικών κυκλωμάτων. Οι δύο πιο δημοφιλέστερες γλώσσες HDL είναι η VHDL και η Verilog. Το κείμενο των προγραμμάτων των γλωσσών αυτών αναπαριστάτε με τέτοιο τρόπο ώστε να είναι αντιληπτό τόσο στους ανθρώπους που το γράφουν όσο και στους υπολογιστές που το εκτελούν. Η χρήση των γλωσσών περιγραφής υλικού αντί των σχηματικών συμβάλει στην πιο εύκολη δημιουργία μεγάλων και πολύπλοκων κυκλωμάτων. Πλέον η υλοποίηση ψηφιακών κυκλωμάτων γίνεται αποκλειστικά με τις γλώσσες περιγραφής υλικού. [6],[7]

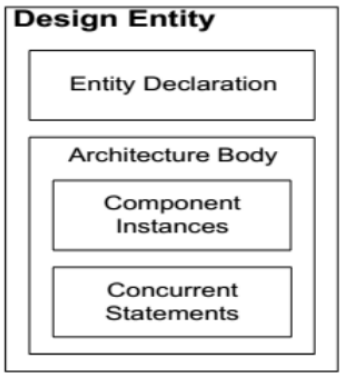
Η γλώσσα περιγραφής υλικού που χρησιμοποιήθηκε στην παρούσα πτυχιακή εργασία είναι η VHDL. Τα αρχικά της VHDL προέρχονται από το VHSIC Hardware Description Language, το VHSIC ή αλλιώς Very High Speed Integrated Circuit Program (δηλαδή Ολοκληρωμένα Κυκλώματα Πολύ Υψηλής Ταχύτητας). Πρόκειταιγια ένα ερευνητικό έργο του Υπουργείου Άμυνας των Ηνωμένων Πολιτειών το οποίο ξεκίνησε το 1980 και είχε διάρκεια μέχρι το 1990. Η εν λόγω έρευνα έγινε με σκοπό την ανάπτυξη των ολοκληρωμένων κυκλωμάτων υψηλής ταχύτητας για τις ένοπλες δυνάμεις των Ηνωμένων Πολιτείων. Η γλώσσα αυτή έγινε πρότυπη από το ινστιτούτο IEEE το 1980 ονόματι ως IEEE 1076.Με το πέρασμα των χρόνων έγιναν αναβαθμίσεις του προτύπου όπως η έκδοση

IEEE 1076 το 2002 και η IEEE 1076 το 2008. Η γλώσσα VHDL αποτελεί βασικό τύπο αρχείου στα λογισμικά σχεδίασης ψηφιακών κυκλωμάτων CAD (Computer Aided Design). Χρησιμοποιείται δε, ευρύτατα σε προγραμματιζόμενες λογικές διατάξεις τύπου FPGAs (Field Programmable Gate Arrays) και CPLDs (Complex Programmable Logic Devices), καθώς και στο σχεδιασμό των κυκλωμάτων ASIC. [4],[9]

Εικόνα 3

Μερικά από τα χαρακτηριστικά της γλώσσας VHDL είναι ότι υποστηρίζει ιεραρχίες (δηλαδή block diagram), διαθέτει διαχείριση λαθών και επιβεβαίωση σωστής λειτουργίας (verification), ενώ τα components που δημιουργούνται μπορούν να αποθηκευτούν ακολούθως στην βιβλιοθήκη (library) για την επαναχρησιμοποίηση τους. Επίσης, η VHDL δεν είναι case sensitive γλώσσα, δηλαδή τα κεφαλαία γράμμα δεν επηρεάζουν το κώδικα. Υπάρχουν τρείς τύποι περιγραφής των κυκλωμάτων στο σώμα της αρχιτεκτονικής ως προς τη δομή τους (structural), ως προς την ροή των δεδομένων (dataflow), ή ως προς την συμπεριφορά (behavioral). Μια οντότητα βέβαια μπορεί να χρησιμοποιήσει παραπάνω από μια αρχιτεκτονικές. [8],[6],[5],[6]

Όσον αφορά την δομή του κώδικα της VHDL αυτή αποτελείται από οντότητες (entities) και αρχιτεκτονικές (architectures). Στην οντότητα βρίσκεται το όνομα της (module), επιπλέον μέσα της δηλώνουμε τα ports, τα οποία μπορεί να λειτουργούν σαν είσοδοι, έξοδοι δηλαδή in, out, inout, buffer. Αναφορικά μετο κομμάτι της αρχιτεκτονικής, αυτό περιγράφει με εντολές και δηλώσεις τις λειτουργίες του κυκλώματος της οντότητας (entity). [5],[6]



Εικόνα 4

1.3 Διατάξεις FPGA

Οι διατάξεις προγραμματιζόμενων πυλών πεδίου ή αλλιώς FPGA, (Field Programmable Gate Array) είναι ένα ηλεκτρονικό στοιχείο το οποίο χρησιμοποιείται από την βιομηχανία των ηλεκτρικών για την κατασκευή πολύπλοκων ψηφιακών κυκλωμάτων. Το FPGA εφευρέθηκε από τον Ross Freeman. Για να προγραμματιστεί χρησιμοποιήθηκαν στατικές μνήμες SRAM. Δυο εταιρίες οι οποίες παράγουν FPGA με την χρήση των μνημών SRAM είναι η Xilinx και η Altera. Άλλη μια αρχιτεκτονική που αναπτύχθηκε από την Actel είναι η χρήση αντιασφαλειών. [10]

Εικόνα 5

Μια FPGA για να λειτουργήσει πρέπει να προγραμματιστεί.Αυτό πραγματοποιείται με την χρήση των γλωσσών HDL αφού πρώτα γίνει η εξομοίωση του προγράμματος και λειτουργεί σωστά. Στη συνέχεια γίνεται η μετάφραση του κώδικα στα λογικά στοιχεία σύμφωνα με την επιλεγμένη FPGA και μετά η φόρτωση τους σε αυτή. Οι κατασκευαστές των FPGA προσφέρουν εργαλεία λογισμικού για την σχεδίαση κυκλωμάτων και τον προγραμματισμό τους, ένα από αυτά είναι και το Quartus που χρησιμοποιήσαμε. Λόγω ότι τα FPGA είναι ολοκληρωμένα κυκλώματα ο σχεδιαστής τους μπορεί να ενσωματώσει και άλλα στοιχεία ολοκληρωμένου κυκλώματος ώστε να επεκτείνει τις δυνατότητες της. Τα στοιχεία αυτά ονομάζονται αλλιώς και πυρήνες. Οι πυρήνες χρησιμοποιούνται για το σχεδιασμό διάφορων κυκλωμάτων σε διάφορους τομείς όπως στην ιατρική, στην άμυνα, στην ρομποτική, στην επικοινωνία και άλλους. [10]

2. Κεφάλαιο 2

2.1 Αριθμητική και λογική μονάδα 32 μπιτ (ArithmeticLogicUnit 32 bit)

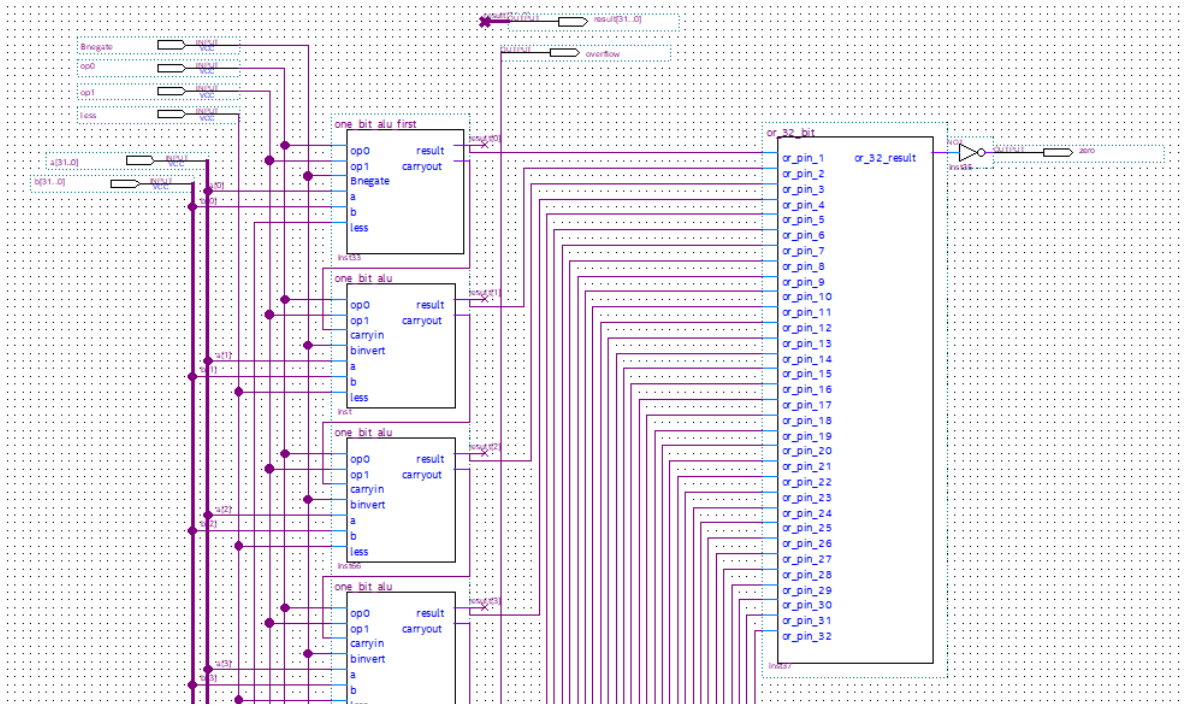
2.1.1 Υλοποίηση

Η αριθμητική και λογική μονάδα όπως προαναφέραμε εκτελεί αριθμητικές και λογικές πράξεις. Η αριθμητική και λογική μονάδα των 32 bit που υλοποιήσαμε αποτελείται από 32 αριθμητικές και λογικές μονάδες του 1 bit, με την πρώτη και την τελευταία μονάδα να έχει κάποια μικρή σχηματικά αλλαγή, την οποία και θα αναλύσουμε στη συνέχεια. Το σχηματικό που υλοποιήσαμε για την αριθμητική και λογική μονάδα πέρα από τις 32 μονάδες 1 bit διαθέτει ακόμη έξι (6) εισόδους, (3) εξόδους και μια λογική πύλη OR των 32 bit. [11]

Αναφορικά με τις εισόδους υπάρχουν μιαείσοδος ονόματι Bnegate, μέσω της οποίας επιλέγουμε την πρόσθεση ή την αφαίρεση (0 bit είναι η πρόσθεση, 1 bit είναι η αφαίρεση), δυοείσοδοι op0, op1 με τις οποίες επιλέγουμε ποια αριθμητική ή λογική πράξη θέλουμε να εκτελέσουμε, μιαείσοδο ονόματι less, η οποία είναι πάντα μηδέν και τέλος άλλες δυοείσοδοι a,b 32 bit, στις οποίες σε κάθε μια από αυτές εισάγονται τα δεδομένα προς επεξεργασία. Οι έξοδοι, οι οποίες προκύπτουν είναι αυτές που μας δίνουν το τελικό αποτέλεσμα μετά την επεξεργασία(result), η οποία είναι 32 bit, η overflow η οποία μας δείχνει αν έχουμε υπερχείλιση, δηλαδή αν το τελικό αποτέλεσμα είναι μεγαλύτερο από το μέγεθος των 32 bit και η έξοδος zero η οποία γίνεται 1 μόνο όταν τα 32 bit του αποτελέσματος είναι μηδέν (0). Για την κατασκευή του συμβόλου OR των 32 bit θα μιλήσουμε παρακάτω. [11]

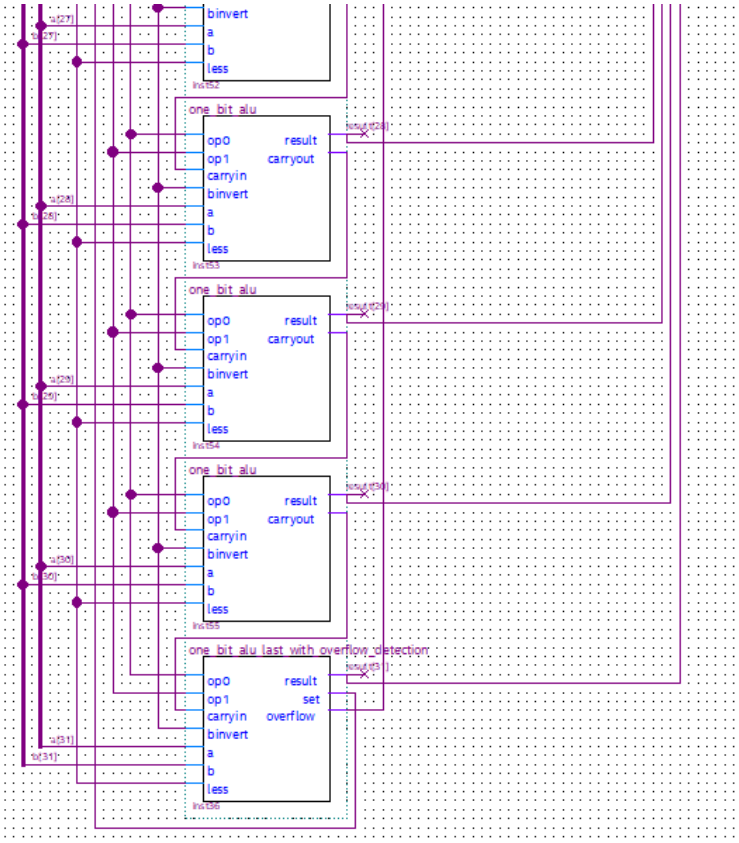
Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε alu_32_bit.

Σχηματικό Αριθμητικής και λογικής μονάδας 32 μπιτ (alu_32_bit), εικόνα των πρώτων μονάδων



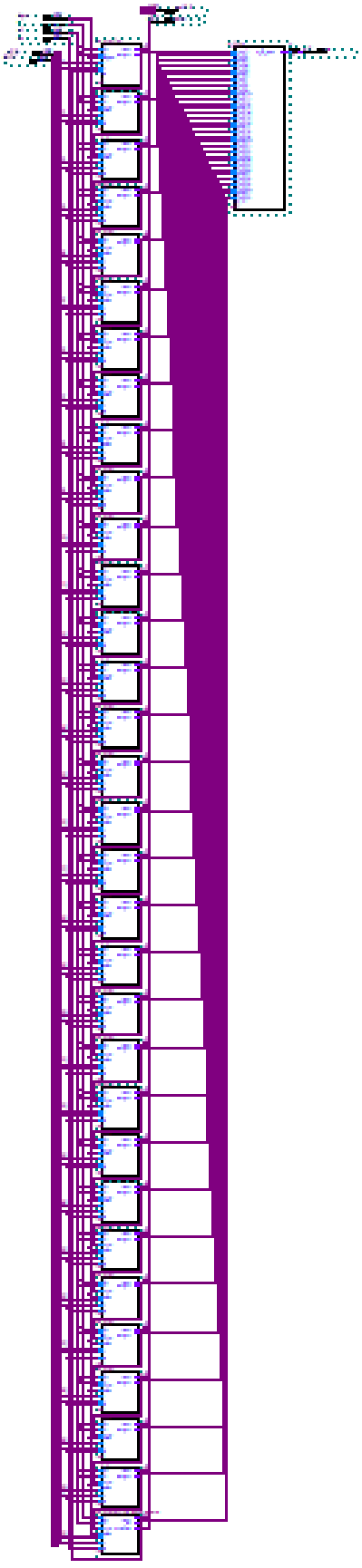
Εικόνα 6

Σχηματικό Αριθμητικής και λογικής μονάδας 32 μπιτ (alu_32_bit), εικόνα των τελευταίων μονάδων



Εικόνα 7

Σχηματικό Αριθμητικής και λογικής μονάδας 32 μπιτ (alu_32_bit), εικόνα όλων των μονάδων



Εικόνα 8

2.1.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά η αριθμητική και λογική μονάδα εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το πίνακα επιλογών πράξεων είδαμε ότι λειτουργεί σωστά. Παρακάτω θα δείξουμε με εικόνες όλες τις λειτουργείες πράξεις της ALU.

Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε alu_32_bit_Waveform.

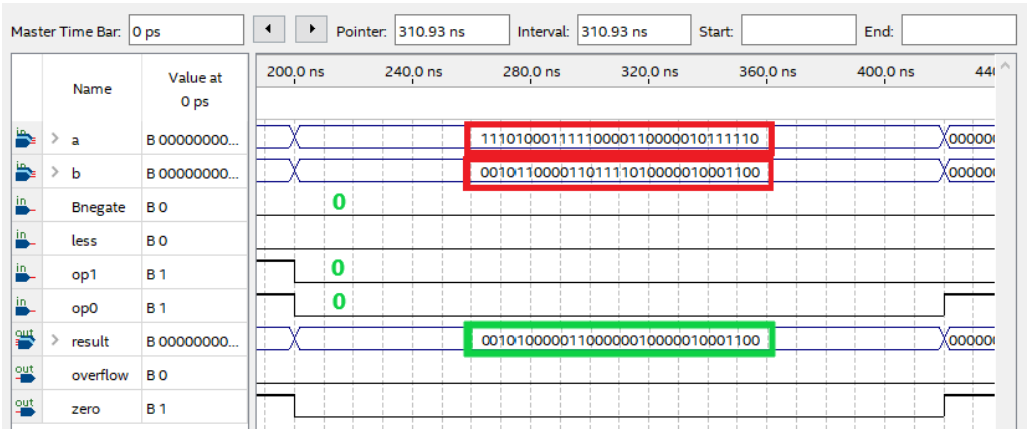
Πίνακας επιλογής πράξεων αριθμητικής και λογικής μονάδας

alu

op1	op0	Bnegate	
0	0	0	and
0	1	0	or
1	0	0	add
1	0	1	subtract
1	1	0	less

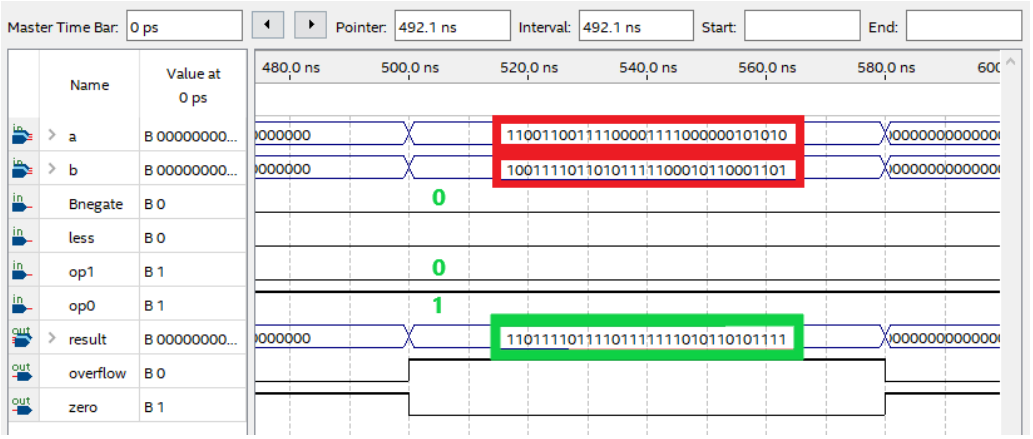
Πίνακας 1

Λογική εξομοίωση Αριθμητικής και λογικής μονάδας 32 bit(alu_32_bit_Waveform), πράξη and



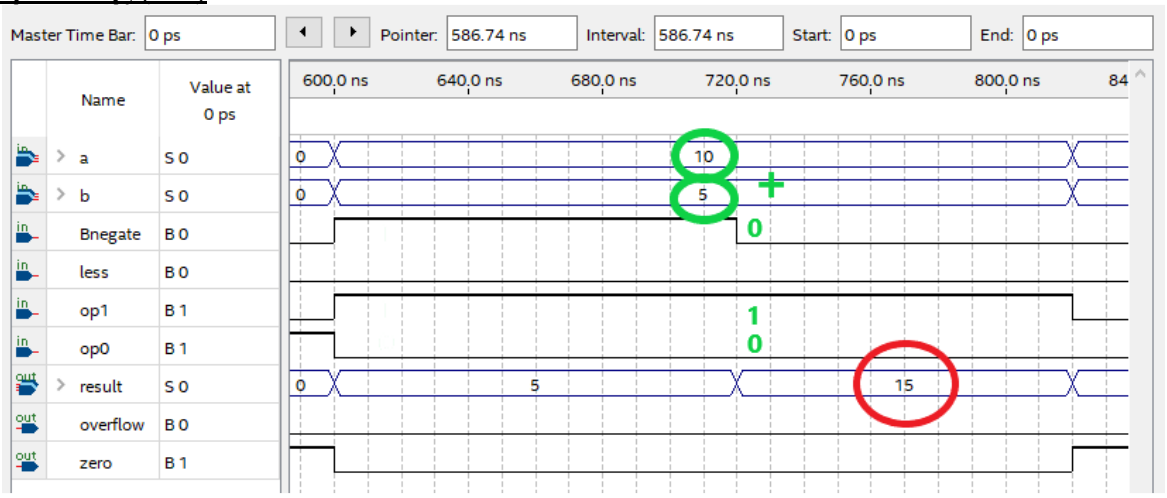
Εικόνα 9

Λογική εξομοίωση Αριθμητικής και λογικής μονάδας 32 bit(alu_32_bit_Waveform), πράξη or



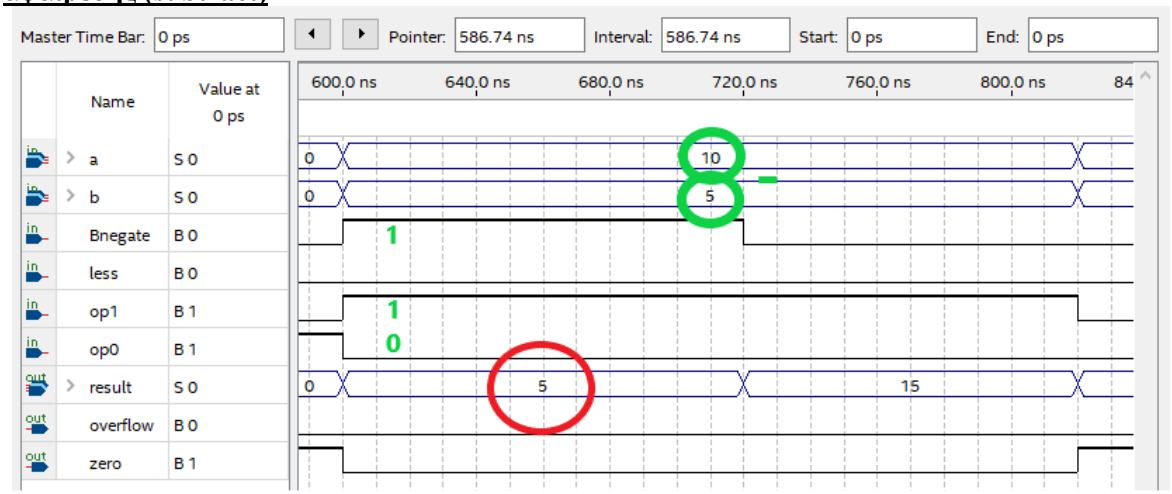
Εικόνα 10

Λογική εξομοίωση Αριθμητικής και λογικής μονάδας 32 bit(alu_32_bit_Waveform), πράξη πρόσθεσης (add)



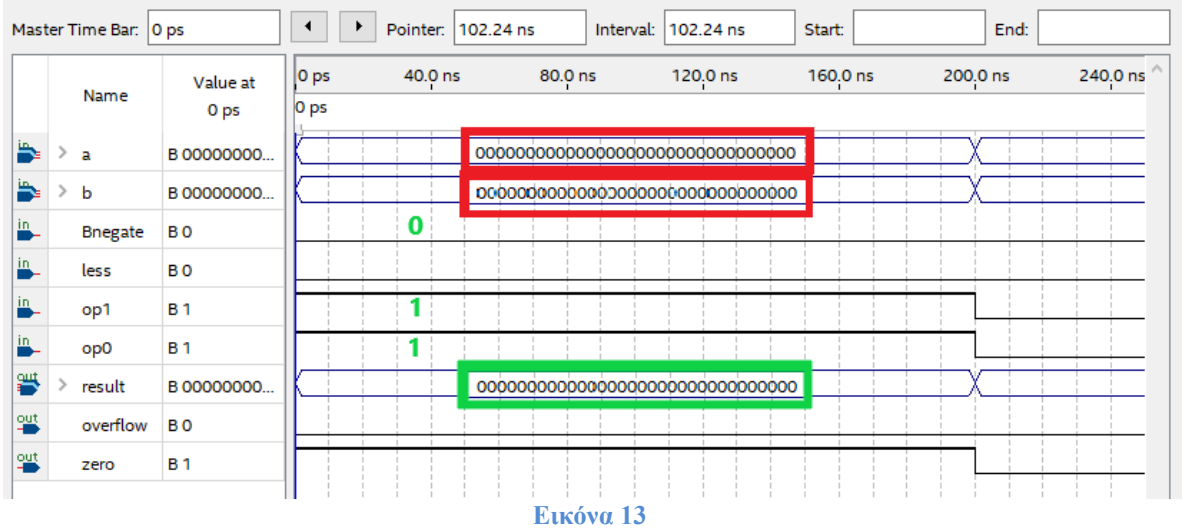
Εικόνα 11

Λογική εξομοίωση Αριθμητικής και λογικής μονάδας 32 bit(alu_32_bit_Waveform), πράξη αφαίρεσης (subtract)



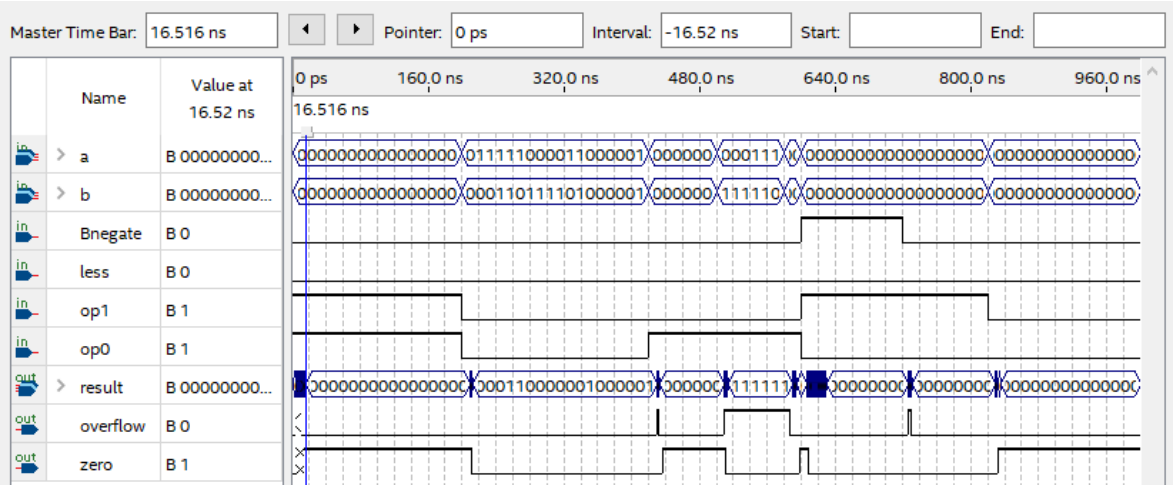
Εικόνα 12

Λογική εξομοίωση Αριθμητικής και λογικής μονάδας 32 bit(alu_32_bit_Waveform), πράξη less



Εικόνα 13

<u>Χρονική εξομοίωση Αριθμητικής και λογικής μονάδας 32 bit</u>(alu_32_bit_Waveform)



Εικόνα 14

2.2 Αριθμητική και λογική μονάδα 1 μπιτ (ArithmeticLogicUnit 1 bit)

2.2.1 Υλοποιήσεις

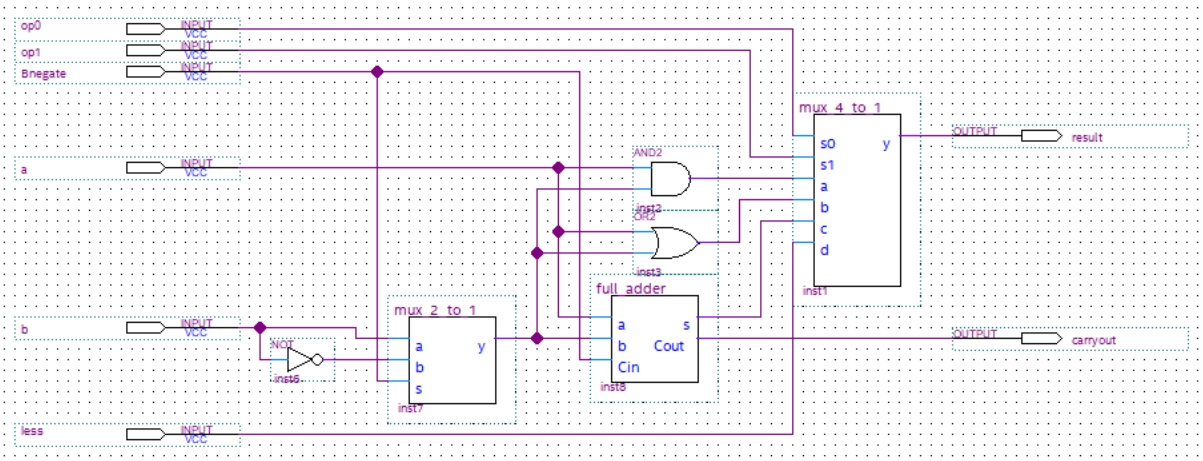
Όπως προαναφέρθηκε παραπάνω υπάρχουν τριών ειδών αριθμητικές και λογικές μονάδες του ενός (1) bit, οι οποίες διαφέρουν ελάχιστα μεταξύ τους. Αρχικά θα αναλύσουμε την πρώτη ALU που χρησιμοποιήθηκε στο σχηματικό της 32 bit ALU και μετά με βάση αυτή θα συγκρίνουμε τις διαφορές της με τις άλλες μονάδες. Όπως βλέπουμε στην παρακάτω εικόνα το σχηματικό της πρώτης αριθμητικής και λογικής μονάδας αποτελείται από έξι (6) εισόδους, δυο(2) εξόδους, μια λογική πύλη and, μια or, μια not (αντιστροφέας), ένα πλήρη αθροιστή, δυο πολυπλέκτες, έναν 2 σε 1 και έναν 4 σε 1. Όσον αφορά τις εισόδους, όπως αναφέραμε και παραπάνω οι είσοδοι op0, op1 είναι για την επιλογή των πράξεων, η Bnegate είναι για την επιλογή της πρόσθεσης ή της αφαίρεσης, οι a,b είσοδοι είναι για τους τελεστέους (αριθμούς που θα εισαχθούν προς επεξεργασία)και η είσοδος less που εισάγεται πάντα λογικό μηδέν (0). Σχετικά με τους πολυπλέκτες, ο πρώτος χρησιμοποιήθηκε για την επιλογή της πρόσθεσης ή της αφαίρεσης από τον οποίο το αποτέλεσμα που προκύπτει πηγαίνει στον πλήρη αθροιστή και ο δεύτερος πολυπλέκτης που χρησιμοποιήθηκε είναι για την επιλογή μεταξύ των λογικών και αριθμητικών πράξεων. Από εξόδους στην ένα (1) bit ALU έχουμε το result που είναι το αποτέλεσμα και το carryin το οποίο συνδέεται με την επόμενη ενός bit μονάδα. Για τις υλοποιήσεις των πολυπλεκτών και του πλήρη αθροιστή θα αναφερθούμε παρακάτω. [11]

Σε αυτή την παράγραφο θα αναλύσουμε τις διαφορές μεταξύ των τριών (3) αριθμητικών και λογικών μονάδων. Όπως βλέπουμε στις παρακάτω εικόνες οι είσοδοι των άλλον δυο μονάδων είναι επτά (7) ενώ της πρώτης είναι έξι (6), η διαφορά τους είναι ότι

αντί για μια είσοδο Bnegate όπως στην πρώτη μονάδα υπάρχουν δυο ακόμη, δηλαδή η binvert που είναι για την επιλογή της πρόσθεσης ή της αφαίρεσης και την carryin που είναι για το κρατούμενο. Στην πρώτη μονάδα αυτές οι δυο είσοδοι γίνονται μια καθώς δεν μπορούν να έχουν διαφορετικό λογικό bit παρά μόνο το ίδιο, δηλαδή δεν γίνεται η binvert να έχει λογικό 0 και η carryin να έχει λογικό 1. Αυτό συμβαίνει γιατί δεν υπάρχει προηγούμενο κρατούμενο για να συνδεθεί με την πρώτη μονάδα. Στις δυο επόμενες μονάδες αυτό δεν δημιουργεί πρόβλημα καθώς η είσοδος binvert ενώνεται με την Bnegate και παίρνει το ίδιο λογικό bit ενώ η carryin αλλάζει συνέχεια αφού είναι συνδεδεμένη με το κρατούμενο carryout της προηγούμενης μονάδας. Άλλη μια διαφορά που εντοπίζουμε στα τρία (3) σχηματικά των αριθμητικών και λογικών μονάδων είναι στην τελευταία η οποία έχει επιπλέον άλλο ένα σύμβολο ονόματι overflow και αντί για μια έξοδο carryout όπως οι άλλες μονάδες έχει δυο την set και την overflow. Στην τελευταία μονάδα ALU δεν υπάρχει έξοδος carryout καθώς δεν υπάρχει επόμενη μονάδα που να χρειάζεται το κρατούμενο. Το σύμβολο overflow που έχει είναι για τον εντοπισμό της υπερχείλισης και από αυτό προκύπτουν οι έξοδοι overflow που μας δείχνει αν έχει γίνει υπερχείλιση και η set, η οποία ενώνεται με την είσοδο less της πρώτης μονάδας ALU. Την κατασκευή του συμβόλου overflow θα την αναλύσουμε εκτενέστερα στη συνέχεια. [11]

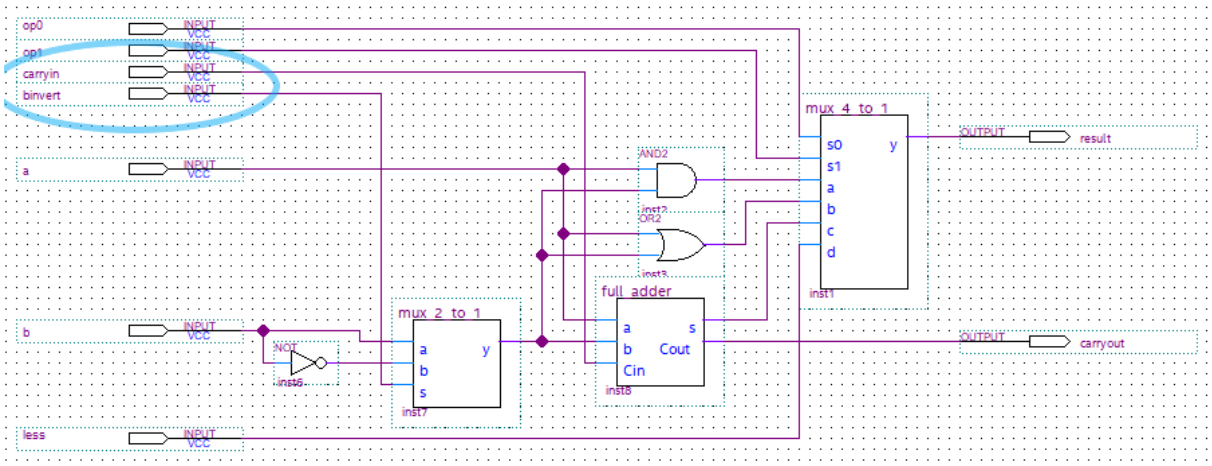
Τα αρχεία των σχηματικών στο πρόγραμμα Quartus Elite Prime για την πρώτη μονάδα το ονομάσαμε one_bit_alu_first,για τις ενδιάμεσες one_bit_alu και για τη τελευταία one_bit_alu_last_with_overflow_detection.

<u>Σχηματικό αριθμητικής και λογικής μονάδας ενός Bit, πρώτο Bit ALU της αριθμητικής και λογικής μονάδας 32 Bit</u> (one_bit_alu_first)



Εικόνα 15

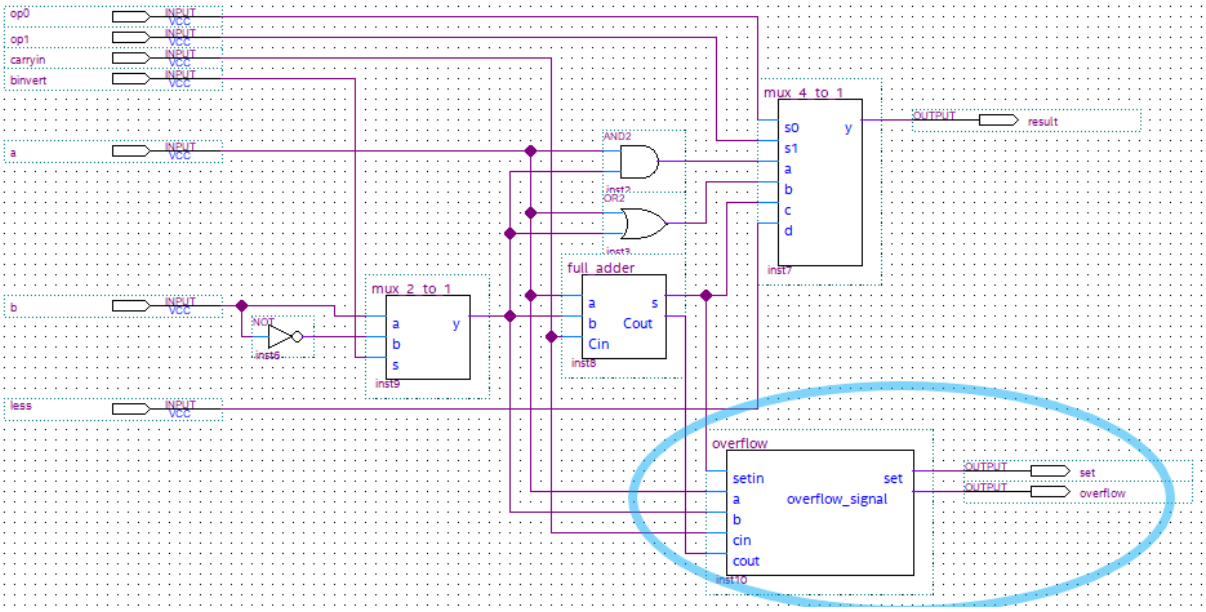
Σχηματικό αριθμητικής και λογικής μονάδας ενός Bit, τα ενδιάμεσα 30 Bit (one_bit_alu)



Εικόνα 16

Σχηματικό αριθμητικής και λογικής μονάδας ενός Bit, τελευταίο Bit ALU της αριθμητικής και λογικής μονάδας 32 Bit με ελεγκτή υπερχείλισης

(one_bit_alu_last_with_overflow_detection)



Εικόνα 17

2.2.2 Εξομοιώσεις

Για να δούμε αν λειτουργούν σωστά οι αριθμητικές και λογικές μονάδες του 1 bit εκτελέσαμε χρονική, λογική εξομοίωση για την κάθε μια ξεχωριστά και σύμφωνα με το πίνακα επιλογών πράξεων είδαμε ότι λειτουργεί σωστά.

Τα αρχεία των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime τα ονομάσαμεως εξής: για την πρώτη μονάδα ονομάζεται one_bit_alu_first_Waveform,για τις ενδιάμεσες

one_bit_alu_Waveform και για τη τελευταία one_bit_alu_last_with_overflow_detection_Waveform.

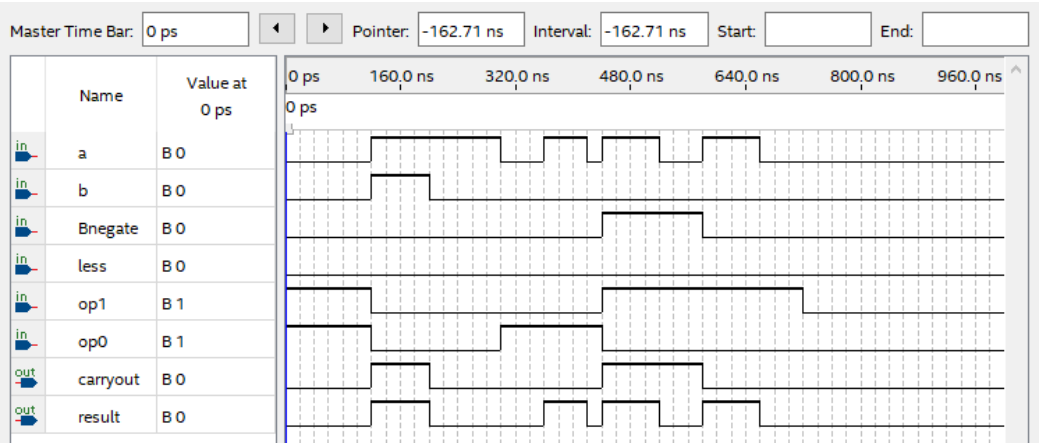
Πίνακας επιλογής πράξεων αριθμητικής και λογικής μονάδας

alu

op1	op0	Bnegate	
0	0	0	and
0	1	0	or
1	0	0	add
1	0	1	subtract
1	1	0	less

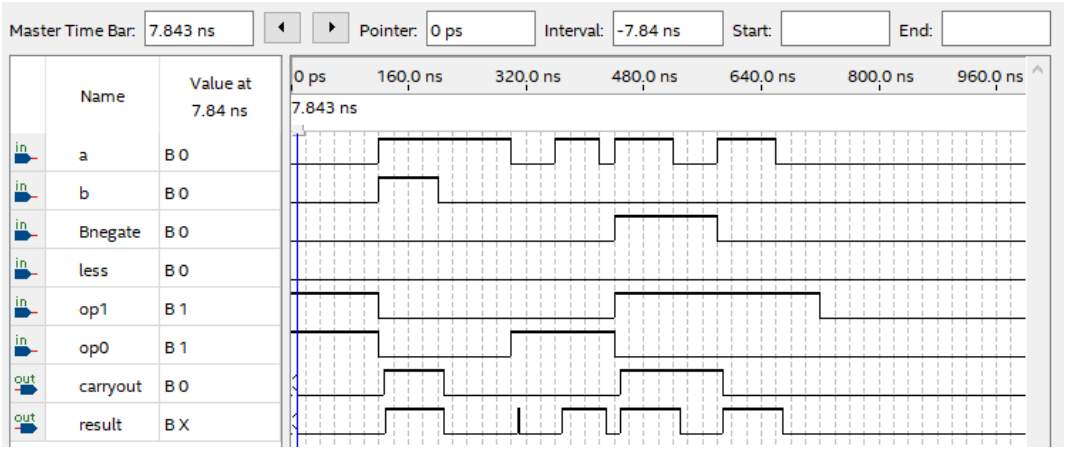
Πίνακας 2

Λογική εξομοίωση της πρώτης Αριθμητικής και λογικής μονάδας1 bit(one_bit_alu_first_Waveform)



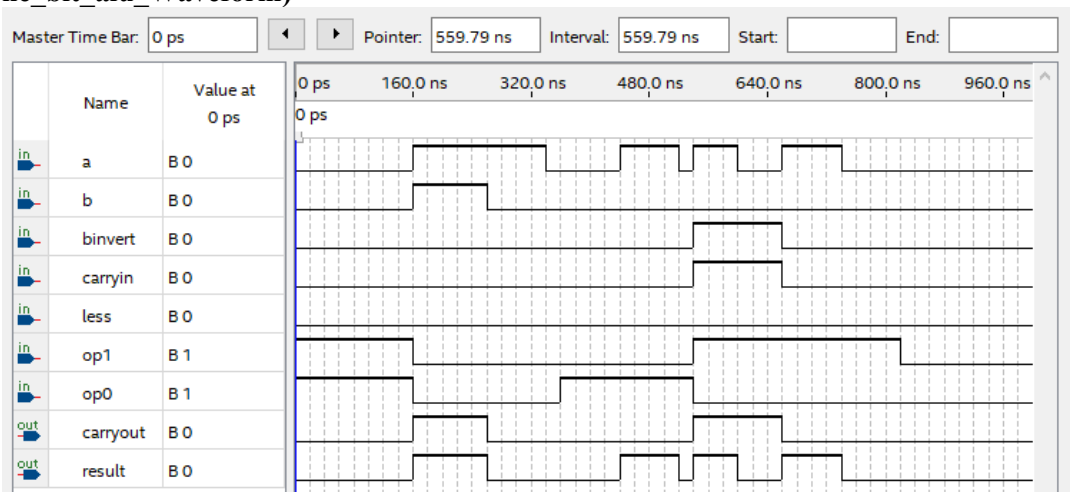
Εικόνα 18

Χρονική εξομοίωση της πρώτης Αριθμητικής και λογικής μονάδας 1 bit(one_bit_alu_first_Waveform)



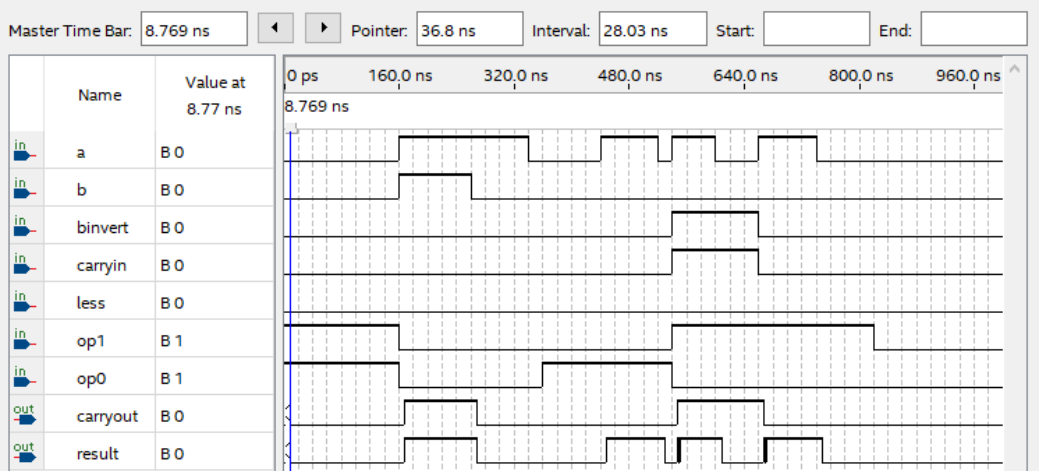
Εικόνα 19

Λογική εξομοίωση των ενδιάμεσων Αριθμητικών και λογικών μονάδων 1 bit(one_bit_alu_Waveform)



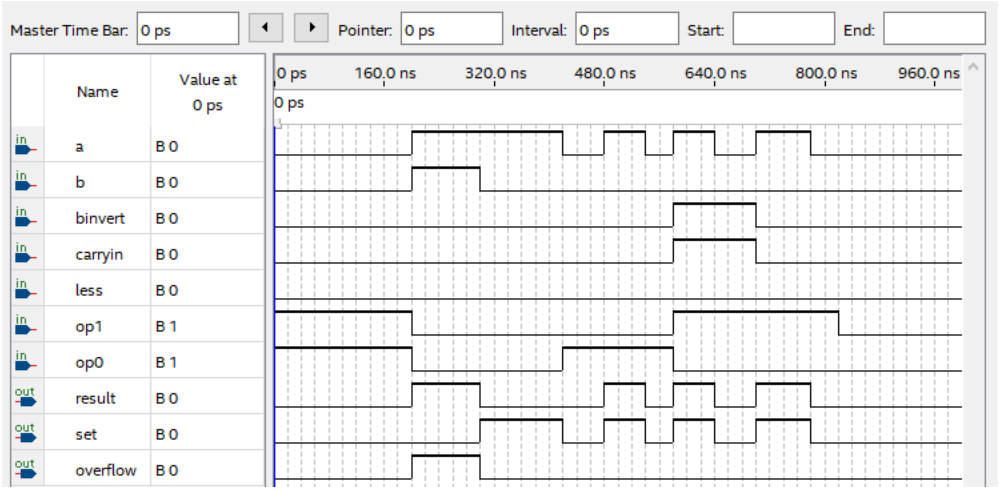
Εικόνα 20

Χρονική εξομοίωση των ενδιάμεσων Αριθμητικών και λογικών μονάδων 1 bit(one_bit_alu_Waveform)



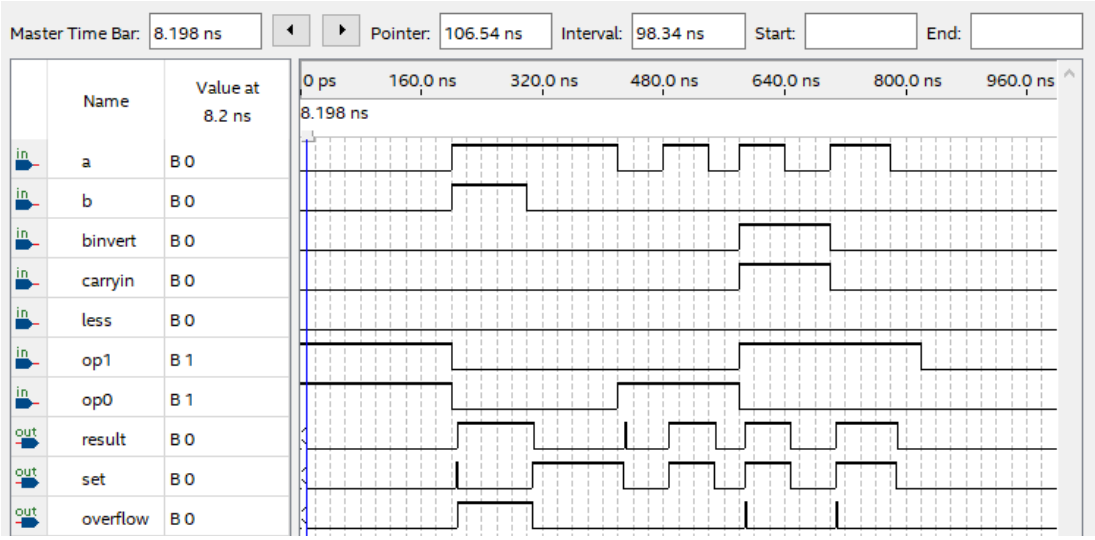
Εικόνα 21

Λογική εξομοίωση της τελευταίας Αριθμητικής και λογικής μονάδας 1 bit(one_bit_alu_last_with_overflow_detection_Waveform)



Εικόνα 22

<u>Χρονική εξομοίωση της τελευταίας Αριθμητικής και λογικής μονάδας 1 bit</u>(one_bit_alu_last_with_overflow_detection_Waveform)



Εικόνα 23

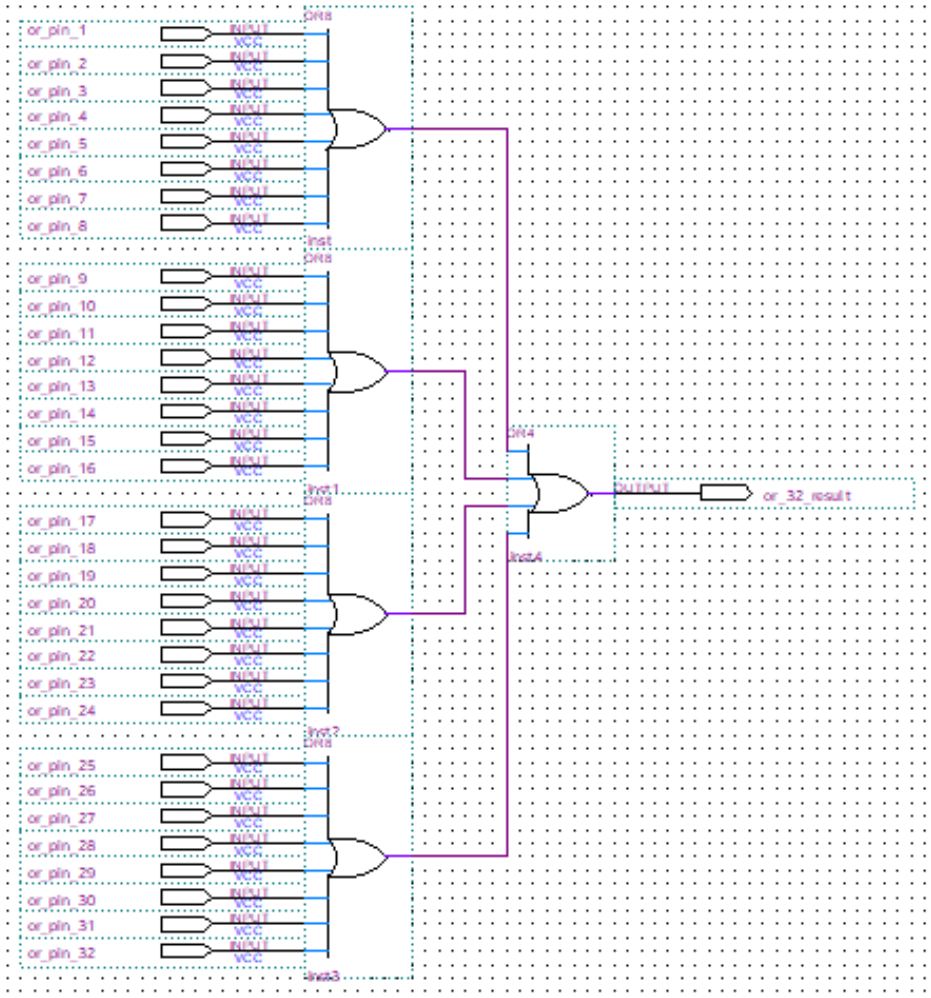
2.3Πύλη OR 32 μπιτ (OR gate 32 bit)

2.3.1 Υλοποίηση

Η πύλη OR 32 bit όπως είδαμε και παραπάνω μας δείχνει αν το αποτέλεσμα της μονάδας ALU είναι μηδέν. Προβήκαμε στην κατασκευή της (της πύλης) καθώς δεν υπήρχε έτοιμο στοιχείο από το πρόγραμμα Quartus Elite Prime. Για την δημιουργία χρησιμοποιήθηκαν τέσσερεις (4) πύλες OR οχτώ (8) εισόδων οι οποίες ενώθηκαν με μια (1) πύλη OR τεσσάρων (4) εισόδων από την οποία η έξοδος της μας δίνει το τελικό αποτέλεσμα. Οπότε το σχηματικό έχει στο σύνολο 32 εισόδους και μια έξοδο.

Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε or_32_bit.

Σχηματικόπύλης OR 32 Bit (or_32_bit)



Εικόνα 24

2.3.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά η πύλη OR 32 bit εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το πίνακα αλήθειας είδαμε ότι λειτουργεί σωστά.

Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε or_32_bit_Waveform.

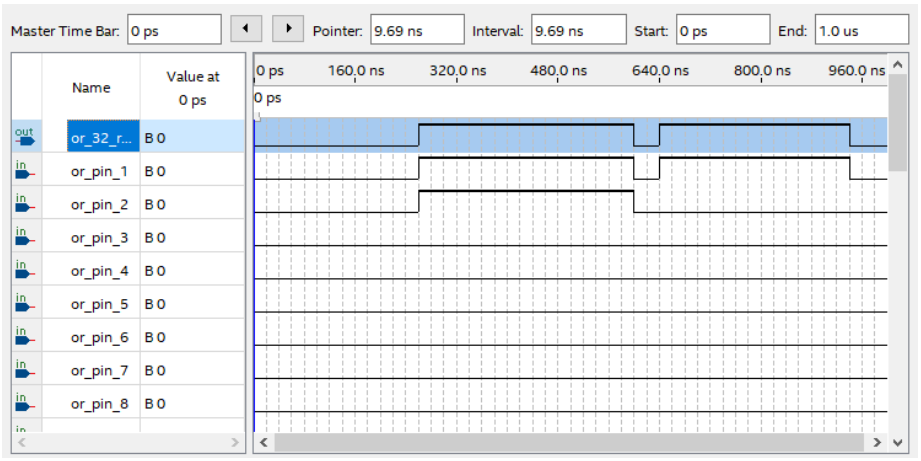
Πίνακας αληθείας OR 32 Bit

or_32_bit

input1	...	input32	y
0	0	0	0
0	1 or 0	1	1
1	1 or 0	0	1
1	1	1	1

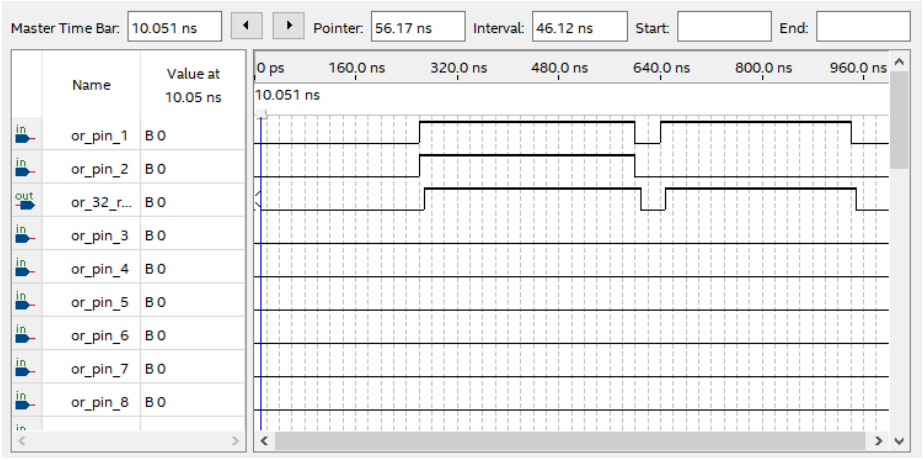
Πίνακας3

Λογικήεξομοίωσηπύλης OR 32 bit (or_32_bit _Waveform)



Εικόνα 25

Χρονικήεξομοίωσηπύλης OR 32 bit (or_32_bit _Waveform)



Εικόνα 26

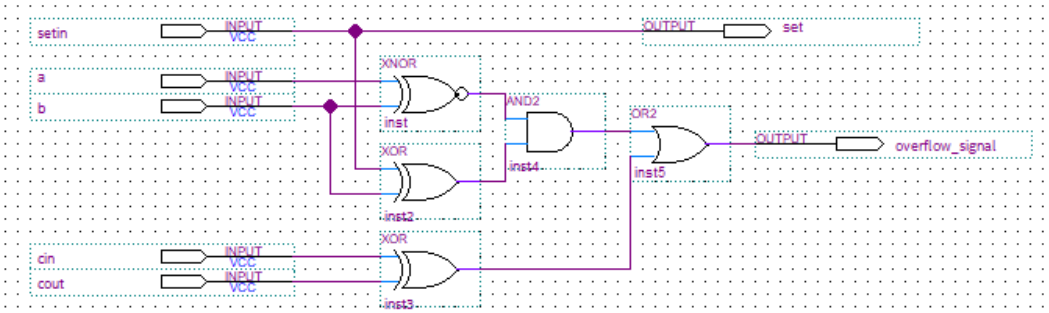
2.4Ελεγκτής υπερχείλισης (OverflowDetector)

2.4.1 Υλοποίηση

Ο ελεγκτής υπερχείλισης (overflow) χρησιμοποιείται για να μας δείξει αν το τελικό αποτέλεσμα είναι μεγαλύτερο από το μέγεθος των 32 bit. Υπάρχουν δυο τρόποι για τον εντοπισμό υπερχείλισης, ο πρώτος είναι συγκρίνοντας τις εισόδους a,b με την έξοδο του πλήρη αθροιστή (full_adder) setin ή συγκρίνοντας το κρατούμενο πού έρχεται cin με αυτό που παράγεται cout. Στο σχηματικό μας θα χρησιμοποιήσουμε και τους δυο τρόπους. Για να δημιουργηθείτο σχηματικό της υπερχείλισης χρειάστηκανπέντε (5)είσοδοι, δυο(2)έξοδοι , δυο (2) πύλες XOR 2 εισόδων, μια (1) πύλη XNOR 2 εισόδων, μια (1) πύλη AND 2 εισόδων και μια (1) πύλη OR 2 εισόδων. [11],[12]

Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε overflow.

<u>Σχηματικό ελεγκτή υπερχείλισης</u> (overflow)



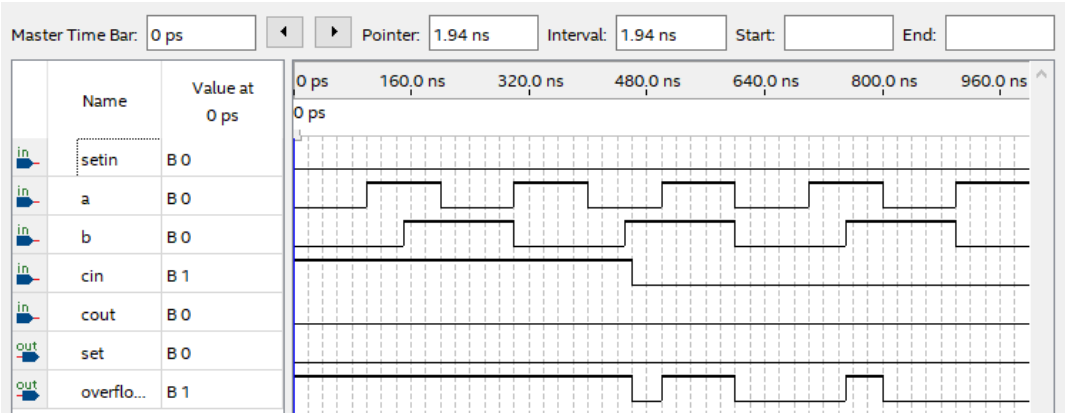
Εικόνα 27

2.4.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά ο ελεγκτής υπερχείλισης (overflow) εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με την λογική του σχηματικού είδαμε ότι λειτουργεί σωστά.

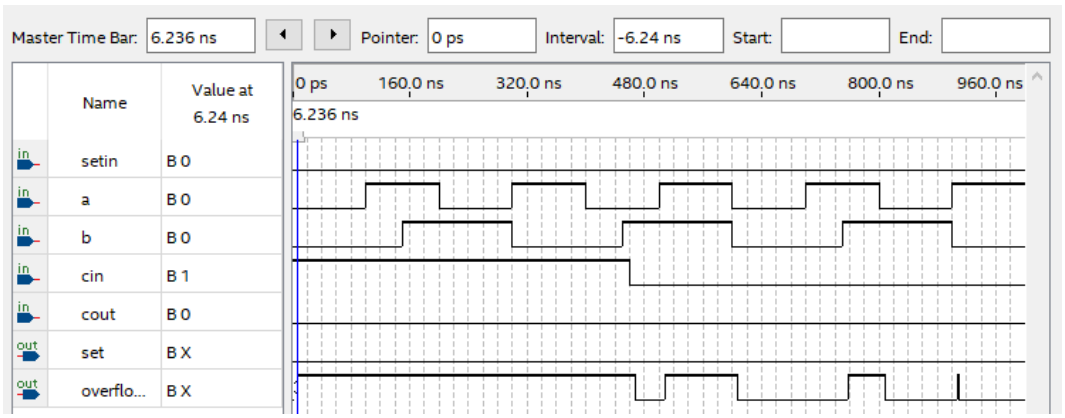
Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε overflow_Waveform.

<u>Λογική εξομοίωση ελεγκτή υπερχείλισης</u> (overflow_Waveform)



Εικόνα 28

<u>Χρονική εξομοίωση ελεγκτή υπερχείλισης</u> (overflow_Waveform)



Εικόνα 29

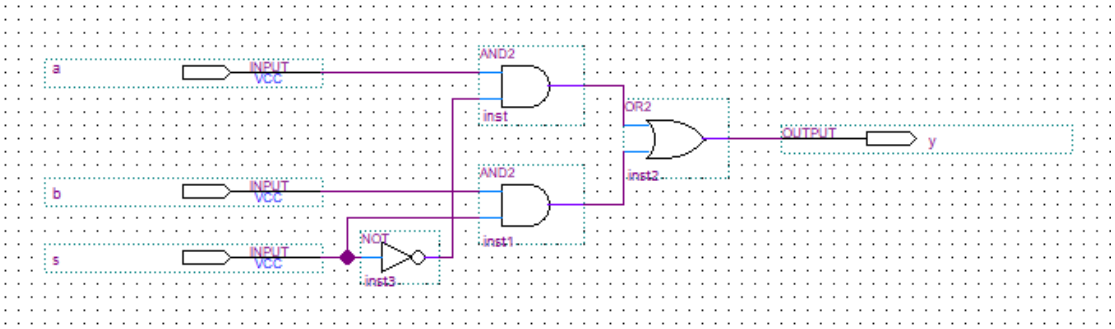
2.5Πολυπλέκτης 2 σε 1 (Multiplexer 2 to 1)

2.5.1 Υλοποίηση

Ο πολυπλέκτης 2 σε 1 λειτουργεί σαν επιλογέας μεταξύ των δυο εισόδων (a,b), επίσης διαθέτει μια έξοδο (y) που δίνει το αποτέλεσμα και τον διακόπτη (s) που χρησιμοποιείται για την επιλογή. Ο πολυπλέκτης όπως απεικονίζεται στην παρακάτω εικόνα αποτελείται από δυο λογικές πύλες AND δυο εισόδων, μια OR δυο εισόδων και μια NOT(αντιστροφέας). [7]

Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε mux_2_to_1.

Σχηματικό Πολυπλέκτη 2 σε 1 (mux_2_to_1)



Εικόνα 30

2.5.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά ο πολυπλέκτης 2 σε 1 εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το πίνακα αλήθειας είδαμε ότι λειτουργεί σωστά.

Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε mux_2_to_1_Waveform.

Πίνακας αληθείας Πολυπλέκτη 2 σε 1

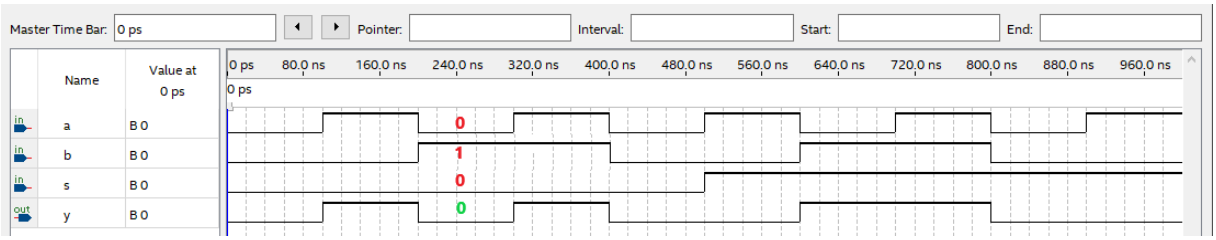
mux_2_to_1 (Πολυπλέκτης 2 σε 1)

s	a	b	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

s	y
0	a
1	b

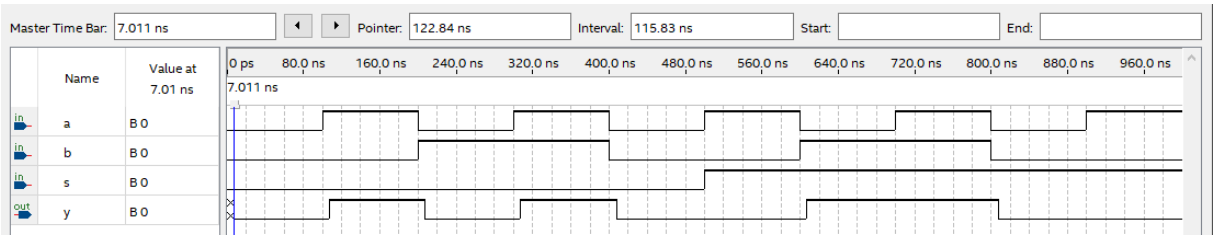
Πίνακας 4

Λογική εξομοίωση Πολυπλέκτη 2 σε 1 (mux_2_to_1_Waveform)



Εικόνα 31

Χρονική εξομοίωση Πολυπλέκτη 2 σε 1 (mux_2_to_1_Waveform)



Εικόνα 32

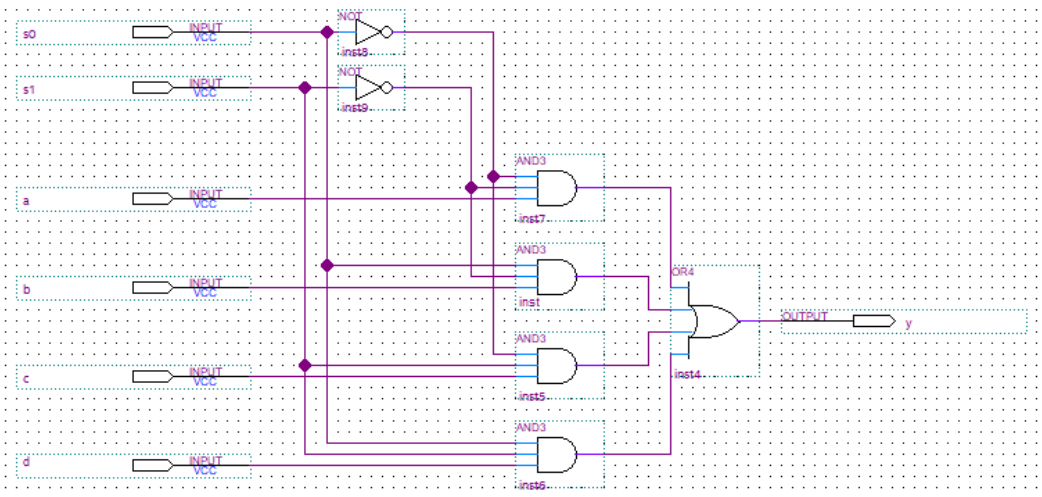
2.6 Πολυπλέκτης 4 σε 1 (Multiplexer 4 to 1)

2.6.1 Υλοποίηση

Ο πολυπλέκτης 4 σε 1 λειτουργεί σαν επιλογέας μεταξύ των τεσσάρων εισόδων (a,b,c,d), επίσης διαθέτει μια έξοδο (y) που δίνει το αποτέλεσμα και έχει δυο διακόπτες (s0, s1) που χρησιμοποιούνται για την επιλογή. Ο πολυπλέκτης όπως βλέπουμε στην εικόναπου ακολουθεί αποτελείται από τέσσερις λογικές πύλες AND τριών εισόδων, μια OR τεσσάρων εισόδων και δυο NOT(αντιστροφείς). [7]

Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε mux_4_to_1.

Σχηματικό Πολυπλέκτη 4 σε 1 (mux_4_to_1)



Εικόνα 33

2.6.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά ο πολυπλέκτης 4 σε 1 εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το πίνακα αλήθειας είδαμε ότι λειτουργεί σωστά.

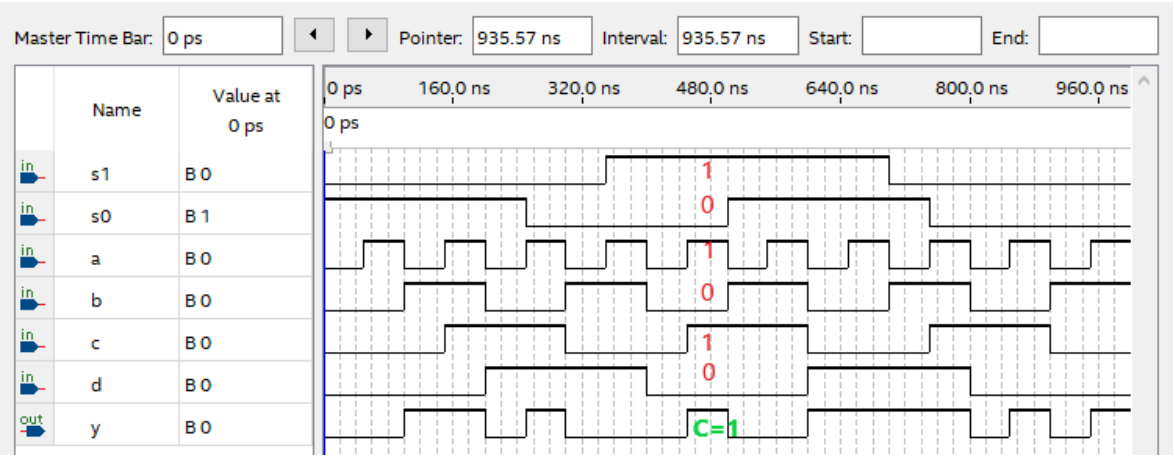
Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε mux_4_to_1_Waveform.

Πίνακας αληθείας Πολυπλέκτη 4 σε 1

mux_4_to_1(Πολυπλέκτης 4 σε 1)		
s1	s0	y
0	0	a
0	1	b
1	0	c
1	1	d

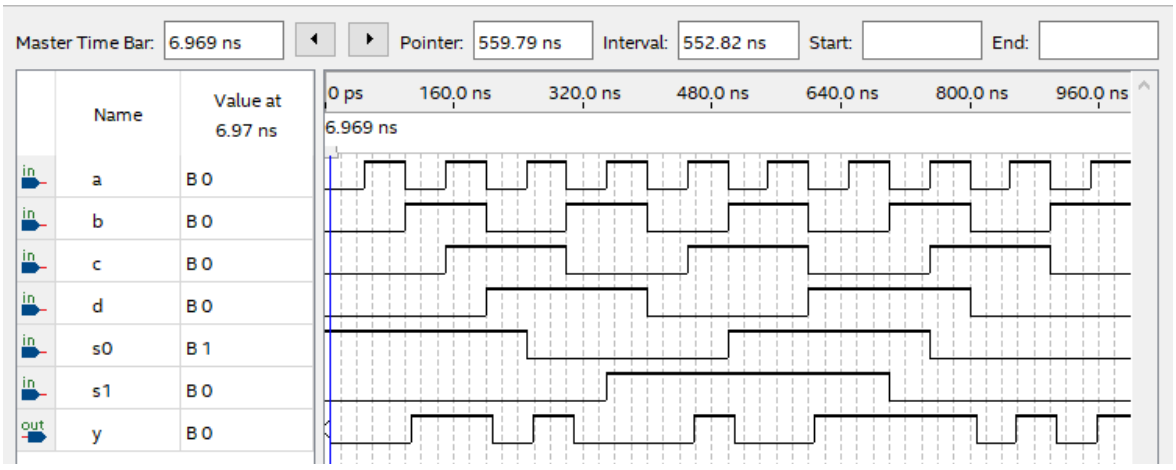
Πίνακας 5

Λογική εξομοίωση Πολυπλέκτη 4 σε 1 (mux_4_to_1_Waveform)



Εικόνα 34

Χρονική εξομοίωση Πολυπλέκτη 4 σε 1 (mux_4_to_1_Waveform)



Εικόνα 35

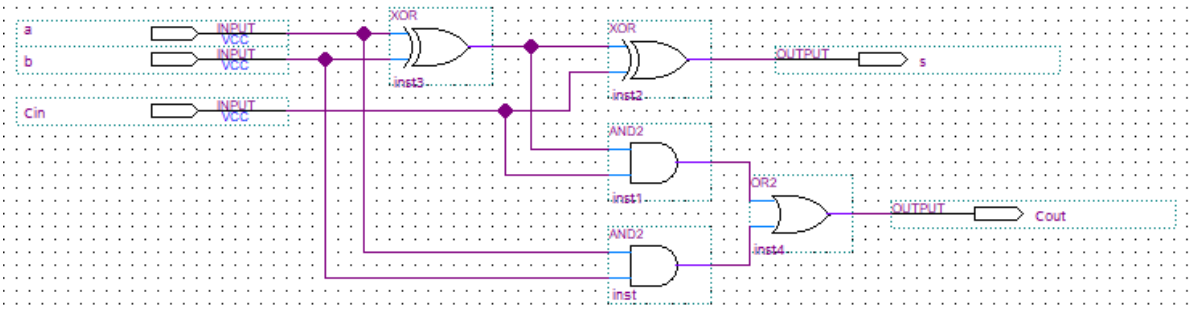
2.7 Πλήρης αθροιστής (FullAdder)

2.7.1 Υλοποίηση

Ο πλήρης αθροιστής χρησιμοποιείται για την πρόσθεση δύο δυαδικών αριθμών. Έχει δυο εισόδους για τους δυαδικούς αριθμούς (a,b), μια είσοδο για το προηγούμενο κρατούμενο(Cin), μια έξοδο για το νέο κρατούμενο (Cout) και άλλη μια έξοδο που δίνει το τελικό αποτέλεσμα (y). Ο πλήρης αθροιστής όπως βλέπουμε στην παρακάτω εικόνα αποτελείται από δυο λογικές πύλες AND δυο εισόδων, μια OR δυο εισόδων και δυο XOR δυο εισόδων. [11]

Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε full_adder.

Σχηματικό πλήρη αθροιστή (full_adder)



Εικόνα 36

2.7.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά ο πλήρης αθροιστής εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το πίνακα αλήθειας είδαμε ότι λειτουργεί σωστά.

Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε full_adder_Waveform.

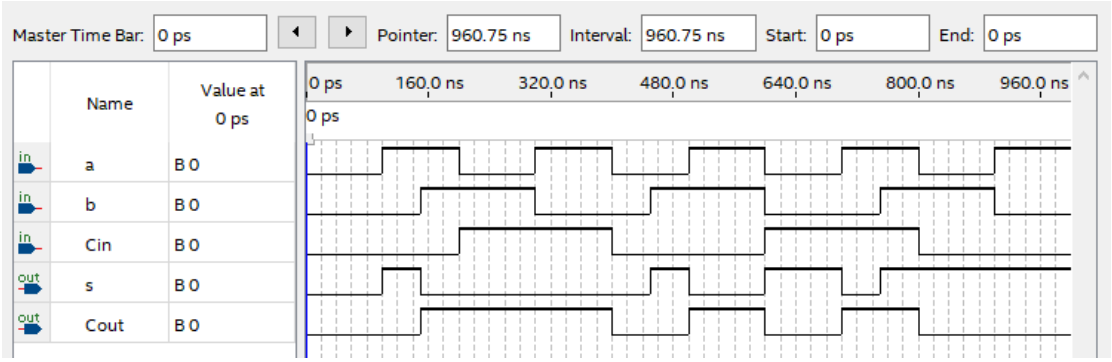
Πίνακας αληθείας Πλήρη αθροιστή

full_adder(Πλήρης αθροιστής)

a	b	Cin	s	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

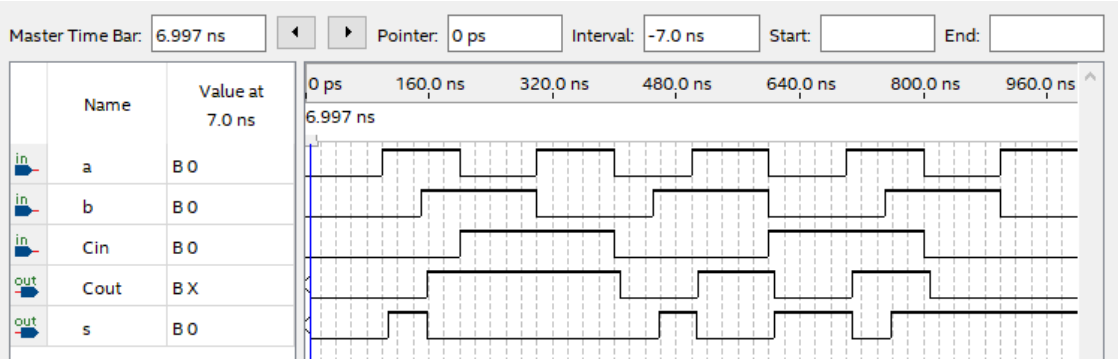
Πίνακας 6

Λογική εξομοίωση Πλήρη αθροιστή (full_adder_Waveform)



Εικόνα 37

Χρονική εξομοίωση Πλήρη αθροιστή (full_adder_Waveform)



Εικόνα 38

3.Κεφάλαιο 3

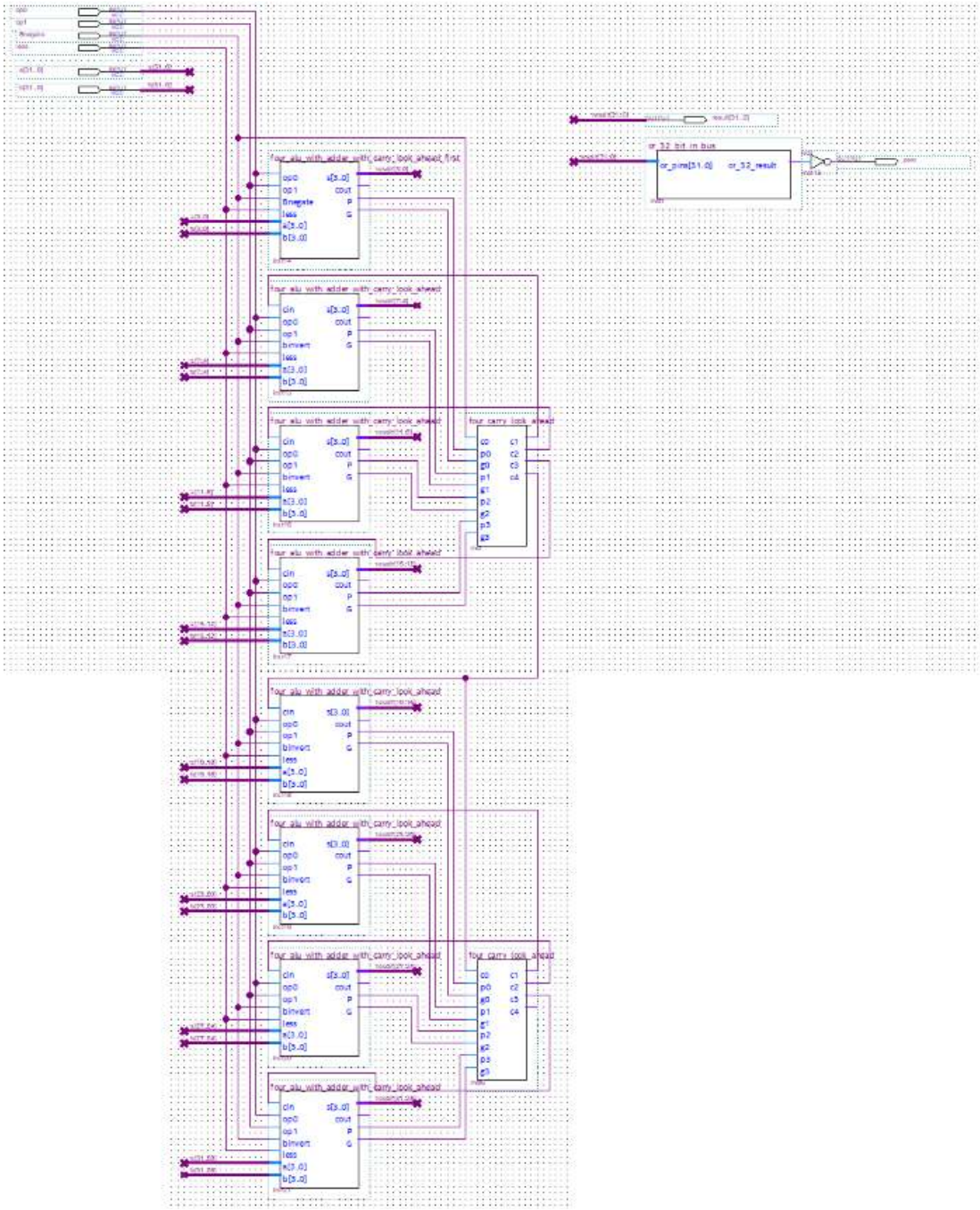
3.1 Αριθμητική και λογική μονάδα 32 μπιτ με γεννήτρια πρόβλεψης κρατουμένου (ALU 32 bitwithcarrylookahead)

3.1.1 Υλοποίηση

Η αριθμητική και λογική μονάδα με την γεννήτρια πρόβλεψης κρατουμένου εκτελεί τις ίδιες αριθμητικές και λογικές πράξεις με την απλή ALU μονάδα.Ωστόσο η ALUμε την γεννήτρια πρόβλεψης διαφέρει στο ότι κάνει τουςυπολογισμούς της πρόσθεσης και της αφαίρεσης σε πιο σύντομους χρόνους, όπωσκαι το κρατούμενο, το οποίο υπολογίζεται επίσης πολύ πιο γρήγορα.Αυτό συμβαίνει καθώς το κρατούμενο δεν υπολογίζεται σε σειρά από 32 μονάδες ALU 1 bit δηλαδή 32 φορές αλλά προβλέπεται μια φορά για τέσσερεις ALU του 1 bit. Το σχηματικό της αριθμητικής και λογικής μονάδας 32 bit με την γεννήτρια πρόβλεψης κρατουμένου αποτελείται από έξι (6) εισόδους, δυο(2) εξόδους, μια πύλη OR 32 bit, μια πύλη NOT (αντιστροφέας), δυο γεννήτριες πρόβλεψης τεσσάρων κρατουμένων και 8 αριθμητικές και λογικές μονάδες 4 bit με γεννήτρια πρόβλεψης κρατουμένου. Όπως και στην απλή μονάδα ALU έτσι και σε αυτή εδώ οι λειτουργίες των εισόδων και των εξόδων είναι ακριβώς οι ίδιες. Όσον αφορά τις 8 μονάδες ALU 4bitμε τη γεννήτρια πρόβλεψης κρατουμένου υπάρχει μια διαφορά μεταξύ της πρώτης και των υπολοίπων που θα την αναλύσουμε αργότερα καθώς θα δούμε και τον τρόπο κατασκευής της γεννήτριας πρόβλεψης τεσσάρων κρατουμένων, της πύλης OR 32 bit με την χρήση διαύλου και θα διαπιστώσουμε αν η ALU Carry Look Ahead μονάδα εκτελεί τις πράξεις πιο γρήγορα από την απλή. [11],[7]

Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε carry_look_ahead_adder_32_bit.

<u>Σχηματικό αριθμητικής και λογικής μονάδας 32 μπιτ με γεννήτρια πρόβλεψης κρατουμένου</u>
(carry_look_ahead_adder_32_bit)



Εικόνα 39

3.1.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά η αριθμητική και λογική μονάδα 32 bit με την γεννήτρια πρόβλεψης κρατουμένου εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το πίνακα επιλογών πράξεων είδαμε ότι λειτουργεί σωστά. Παρακάτω θα παραθέσουμε εικόνες μέσω των οποίων θα παρουσιάσουμε τον υπολογισμό της πρόσθεσης και της αφαίρεσης καθώς οι άλλες λειτουργίες παραμένουν ίδιες χρονικά και σχηματικά.

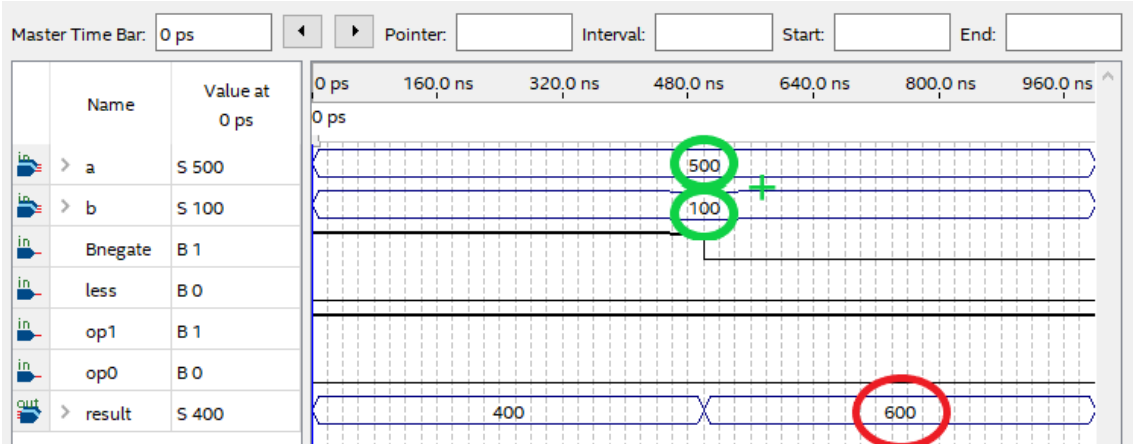
Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε carry_look_ahead_adder_32_bit_Waveform.

Πίνακας επιλογής πράξεων αριθμητικής και λογικής μονάδας

alu			
op1	op0	Bnegate	
0	0	0	and
0	1	0	or
1	0	0	add
1	0	1	subtract
1	1	0	less

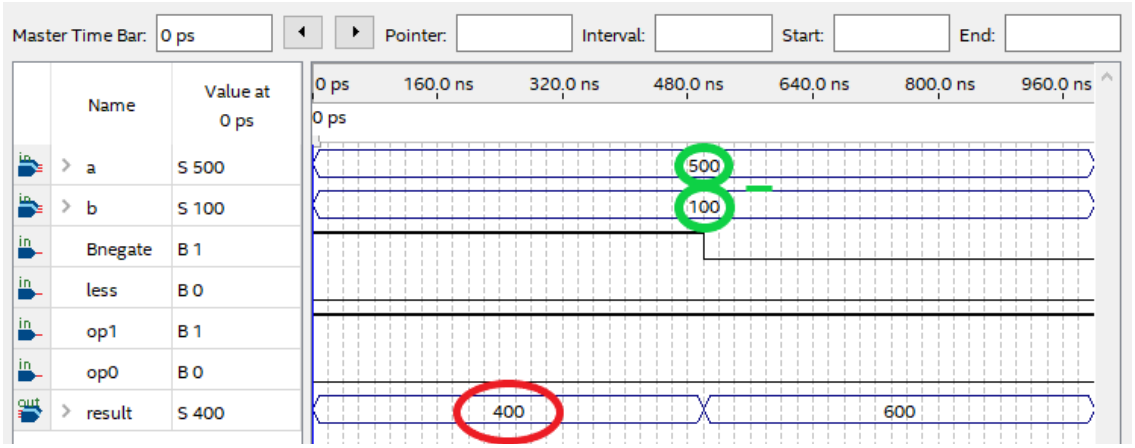
Πίνακας 7

Λογική εξομοίωση Αριθμητικής και λογικής μονάδας με γεννήτρια πρόβλεψης κρατουμένου (carry_look_ahead_adder_32_bit_Waveform), πράξη πρόσθεσης (add)



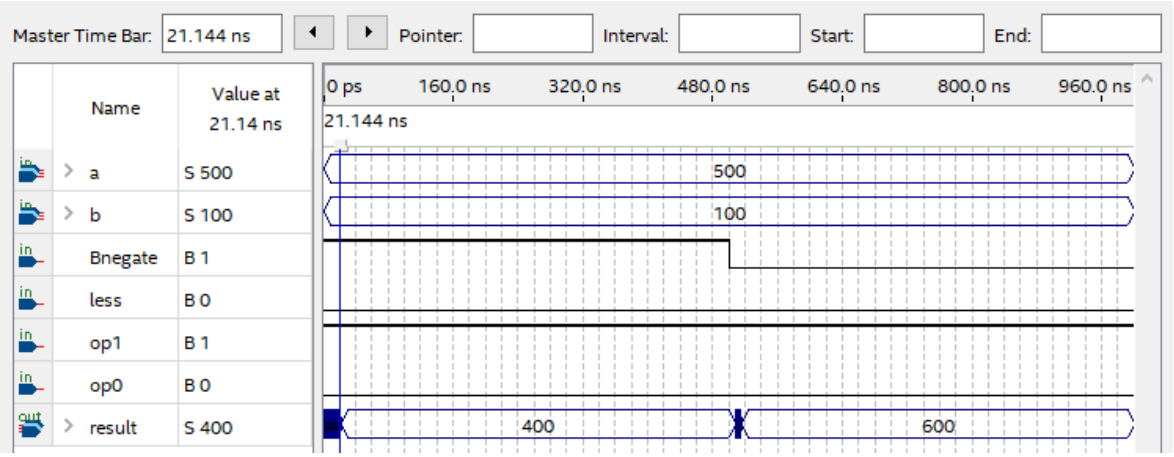
Εικόνα 40

Λογική εξομοίωση Αριθμητικής και λογικής μονάδας με γεννήτρια πρόβλεψης κρατουμένου (carry_look_ahead_adder_32_bit_Waveform), πράξη αφαίρεσης (subtract)



Εικόνα 41

Χρονική εξομοίωση Αριθμητικής και λογικής μονάδας με γεννήτρια πρόβλεψης κρατουμένου (carry_look_ahead_adder_32_bit_Waveform)



Εικόνα 42

3.2 Αριθμητική και λογική μονάδα 4 μπιτ με γεννήτρια πρόβλεψης κρατουμένου (ALU4bitwithcarrylookahead)

3.2.1 Υλοποίηση

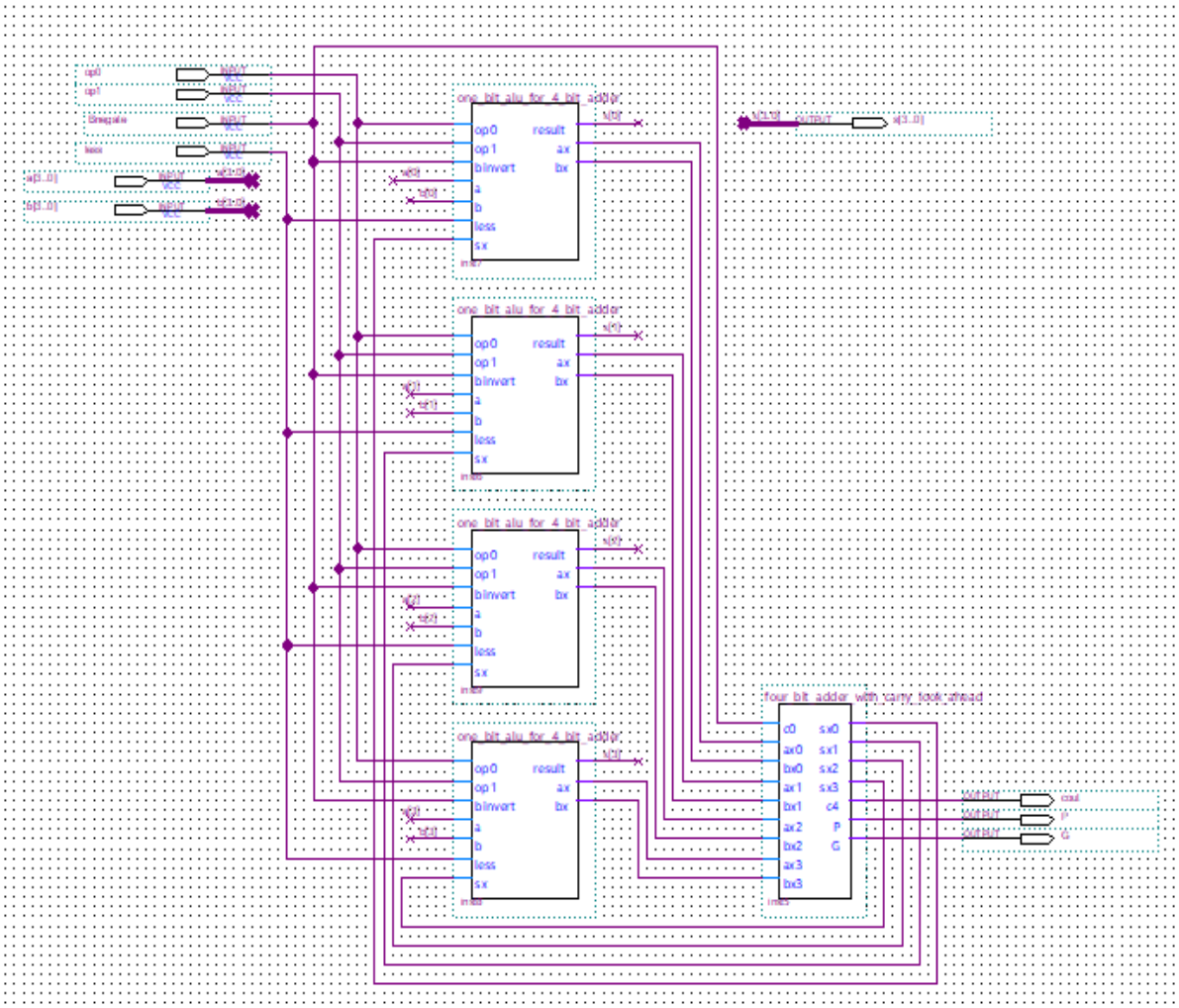
Όπως αναφέραμε υπάρχουν δυο ειδών αριθμητικές και λογικές μονάδες τεσσάρων bit με γεννήτρια πρόβλεψης κρατουμένου οι οποίες διαφέρουν ελάχιστα μεταξύ τους. Αρχικά θα αναλύσουμε την πρώτη 4 bit ALU Carry Look Ahead που χρησιμοποιήθηκε στο σχηματικό της 32 bit ALU Carry Look Ahead και μετά με βάση αυτή θα συγκρίνουμε τις διαφορές που έχει με τις άλλες μονάδες. [7],[11]

Στην παρακάτω εικόνα παρουσιάζουμε το σχηματικό της πρώτης μονάδας το οποίο αποτελείται από έξι (6) εισόδους, τέσσερεις(4) εξόδους, τέσσερεις ενός bit μονάδες ALU χωρίς τον πλήρη αθροιστή και μια γεννήτρια πρόβλεψης κρατουμένου 4 bit με πλήρη αθροιστή. Όσον αφορά τις εισόδους, όπως αναφέραμε και σε προηγούμενο κεφάλαιο οι είσοδοι op0, op1 είναι για την επιλογή των πράξεων, η Bnegate είναι για την επιλογή της πρόσθεσης ή της αφαίρεσης, οι a,b είσοδοι είναι για τους αριθμούς που θα εισαχθούν προς υπολογισμό και η είσοδος less που μπαίνει πάντα λογικό μηδέν (0). Όσον αφορά τις εξόδους διαθέτει την έξοδο x τεσσάρων bit που είναι για το αποτέλεσμα, την cout που είναι για το κρατούμενο, την P που είναι για την διάδοση των σημάτων και την G που είναι για την δημιουργία των σημάτων για κάθε ένα από τα τέσσερα μπλοκ πλήρη αθροιστή τεσσάρων bit. Για τις υλοποιήσεις της μονάδας ALU χωρίς αθροιστή και της γεννήτριας πρόβλεψης κρατουμένου τεσσάρων bit με πλήρη αθροιστή θα αναφερθούμε παρακάτω. [7],[11]

Σχετικά με τις διαφορές των δυο μονάδων 4 bit ALU with Carry Look Ahead όπως βλέπουμε και στις παρακάτω εικόνες η μόνη διαφορά τους είναι ότι αντί για μια είσοδο Bnegate όπως στην πρώτη μονάδα υπάρχουν δυο ακόμη είσοδοι, ήτοι η binvert και η carryin . Η binvert αναφέραμε είναι για την επιλογή της πρόσθεσης ή της αφαίρεσης, ενώ η carryin σε αυτό το σχηματικό αντί να υπάρχει στις μονάδες 1 bit ALU, πηγαίνει στην γεννήτρια πρόβλεψης κρατουμένου, καθώς εκεί βρίσκεται ο πλήρης αθροιστής και όχι στις ALU. [7],[11]

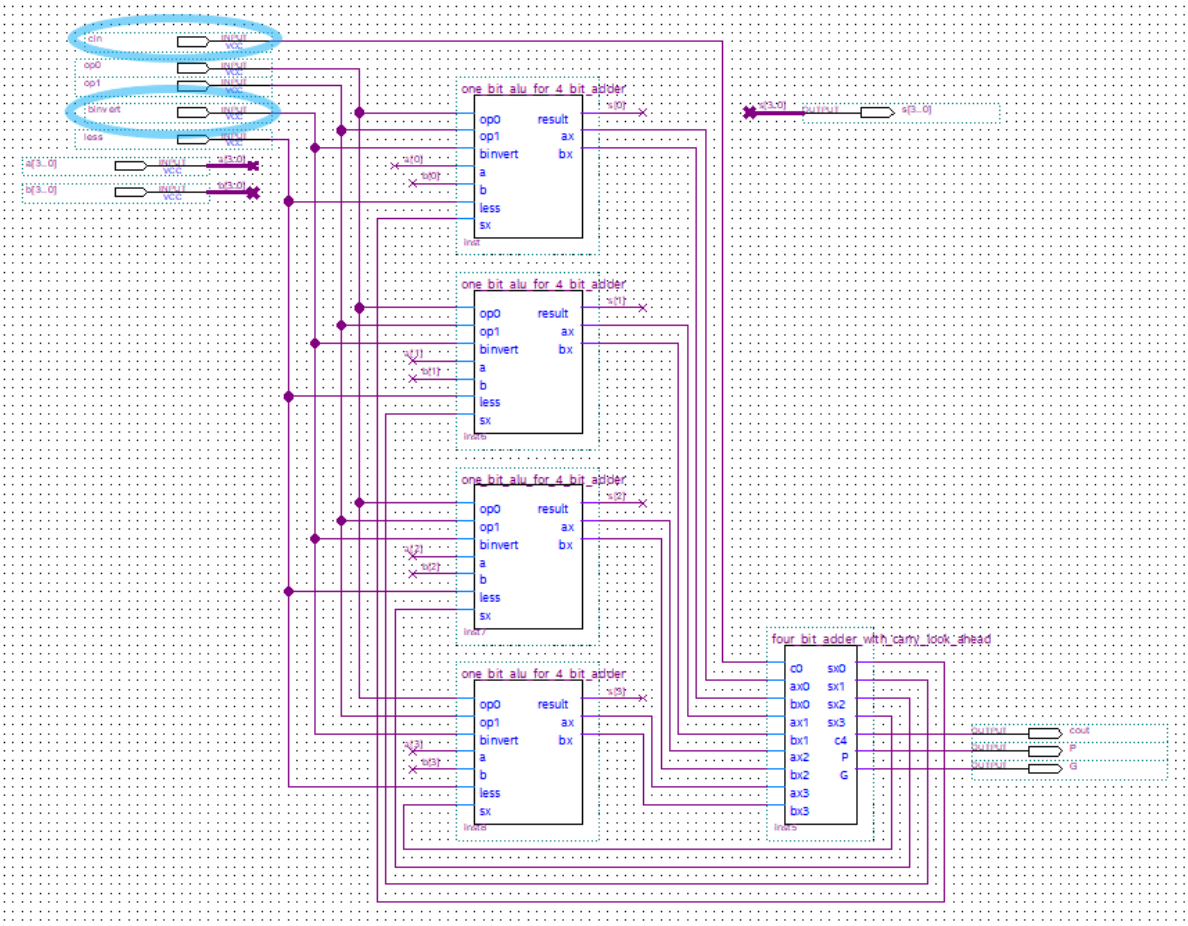
Τα αρχεία των σχηματικών στο πρόγραμμα Quartus Elite Prime τα ονομάσαμεως εξής: για την πρώτη μονάδα four_alu_with_adder_with_carry_look_ahead_first και για τις υπόλοιπες four_alu_with_adder_with_carry_look_ahead.

<u>Σχηματικό της πρώτης αριθμητικής και λογικής μονάδας 4bit με γεννήτρια πρόβλεψης κρατουμένου</u> (four_alu_with_adder_with_carry_look_ahead_first)



Εικόνα 43

Σχηματικό των υπολοίπων αριθμητικών και λογικών μονάδων 4 bit με γεννήτρια πρόβλεψης κρατουμένου (four_alu_with_adder_with_carry_look_ahead)



Εικόνα 44

3.2.2 Εξομοιώσεις

Για να δούμε αν λειτουργούν σωστά οι αριθμητικές και λογικές μονάδες των 4 bit εκτελέσαμε χρονική, λογική εξομοίωση για την κάθε μια ξεχωριστά όπου σύμφωνα με το πίνακα επιλογών πράξεων είδαμε ότι λειτουργεί σωστά. Όπως αναφέραμε και παραπάνω θα δούμε την πράξη της πρόσθεσης και της αφαίρεσης καθώς οι υπόλοιπες λειτουργίες των μονάδων δεν επηρεάστηκαν.

Τα αρχεία των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime για την πρώτη μονάδα τα ονομάσαμε four_alu_with_adder_with_carry_look_ahead_first και για τις υπόλοιπες four_alu_with_adder_with_carry_look_ahead.

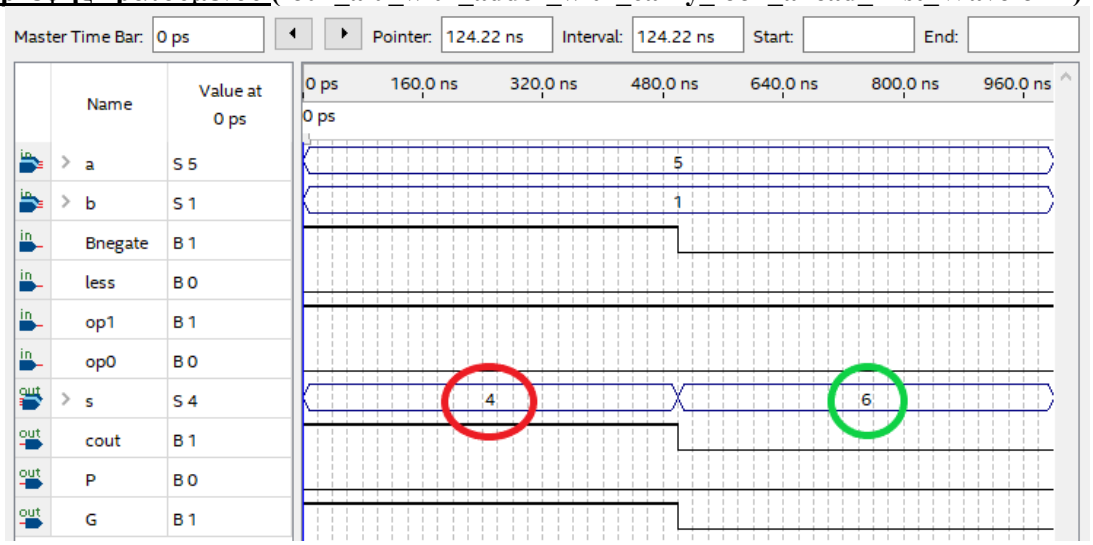
Πίνακας επιλογής πράξεων αριθμητικής και λογικής μονάδας

alu

op1	op0	Bnegate	
0	0	0	and
0	1	0	or
1	0	0	add
1	0	1	subtract
1	1	0	less

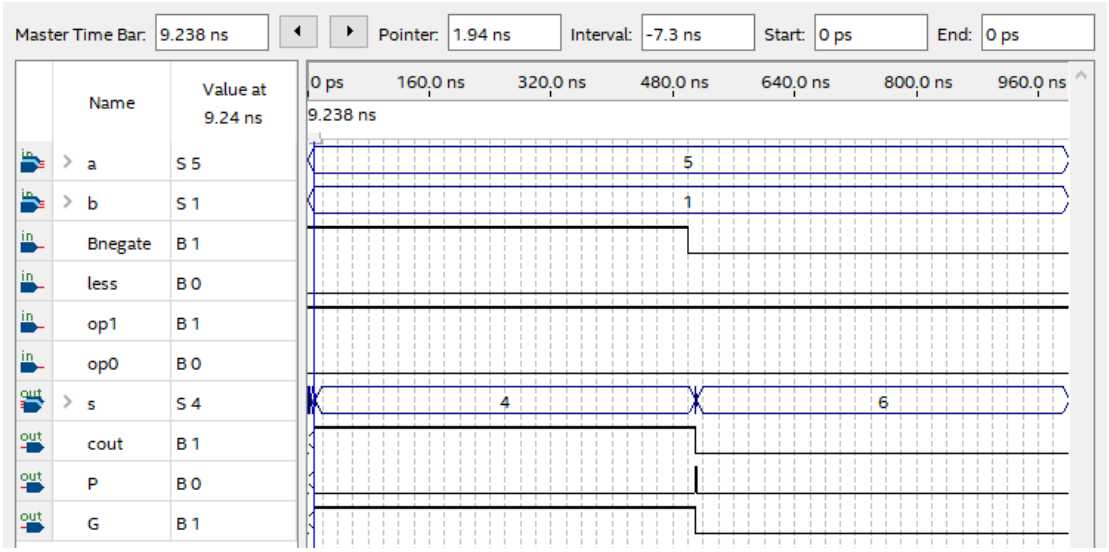
Πίνακας 8

Λογική εξομοίωση της πρώτης αριθμητικής και λογικής μονάδας 4bit με γεννήτρια πρόβλεψης κρατουμένου (four_alu_with_adder_with_carry_look_ahead_first_Waveform)



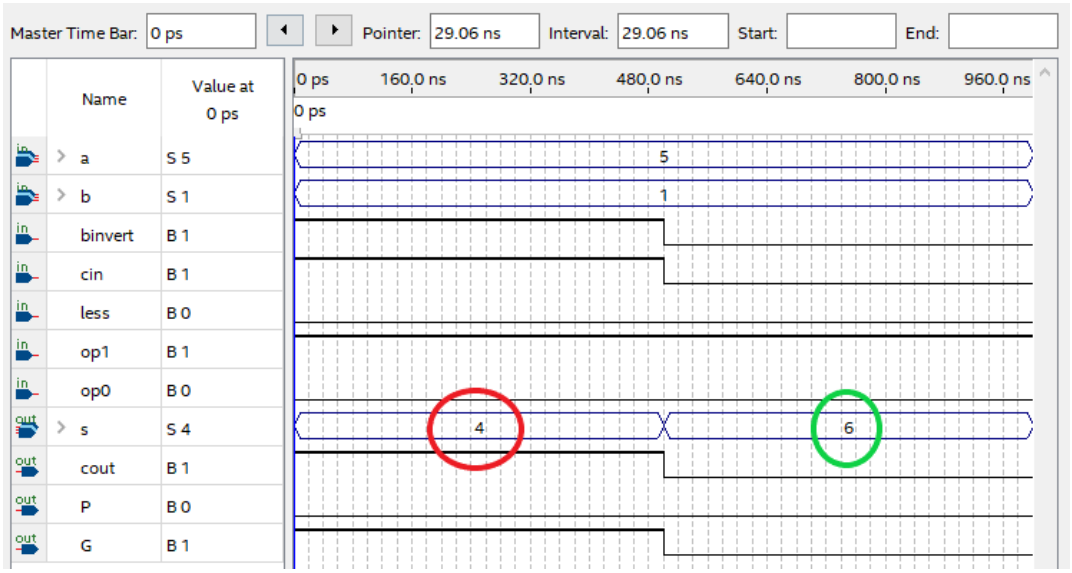
Εικόνα 45

Χρονική εξομοίωση της πρώτης αριθμητικής και λογικής μονάδας 4 bit με γεννήτρια πρόβλεψης κρατουμένου (four_alu_with_adder_with_carry_look_ahead_first_Waveform)



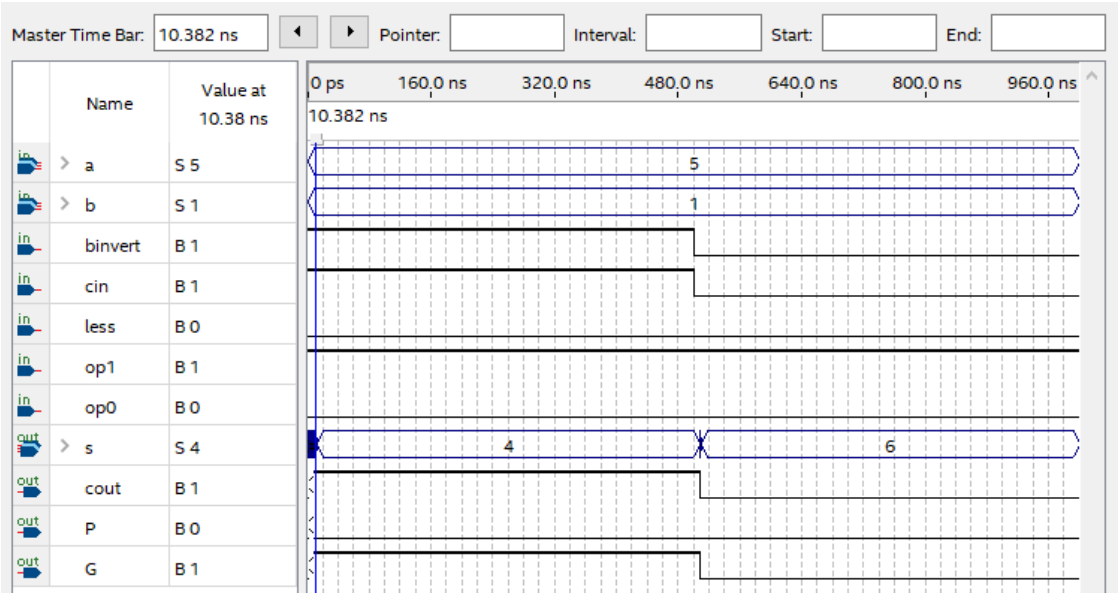
Εικόνα 46

Λογική εξομοίωση των υπολοίπων αριθμητικών και λογικών μονάδων 4 bit με γεννήτρια πρόβλεψης κρατουμένου(four_alu_with_adder_with_carry_look_ahead_first_Waveform)



Εικόνα 47

Χρονική εξομοίωση των υπολοίπων αριθμητικών και λογικών μονάδων 4 bit με γεννήτρια πρόβλεψης κρατουμένου (four_alu_with_adder_with_carry_look_ahead_first_Waveform)



Εικόνα 48

3.3Πύλη OR 32 μπιτμέσωδιαύλου (32 bit gate in bus)

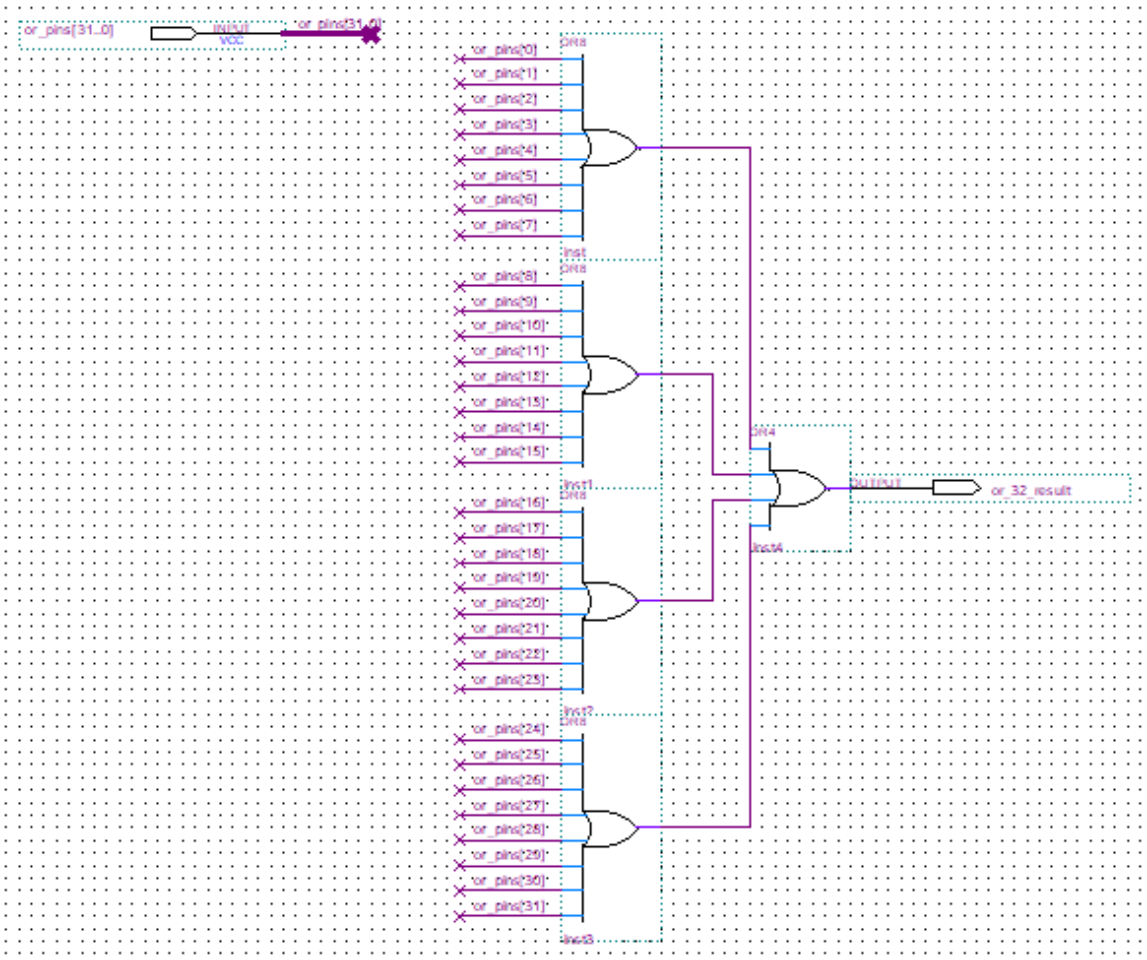
3.3.1 Υλοποίηση

Η πύλη OR 32 bit που χρησιμοποιήθηκε στο σχηματικό της αριθμητικής και λογικής με την γεννήτρια πρόβλεψης κρατουμένου έχει ακριβώς την ίδια χρήση με την αριθμητική και λογική μονάδα του προηγούμενου κεφαλαίου με την διαφορά ότι η εισαγωγή των δεδομένων γίνεται μέσω ενός διαύλου. Όπως για την άλλη μονάδα έτσι και για αυτή

προβήκαμε στην κατασκευή της πύλης καθώς δεν υπήρχε έτοιμο στοιχείο/μπλοκ από το πρόγραμμα Quartus Elite Prime. Για την δημιουργία της πύλης χρησιμοποιήσαμε την πύλη της άλλης μονάδας με την μόνη αλλαγή ότι αντί για 32 εισόδους βάλαμε μια των 32 bit.

Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε or_32_bit_in_bus.

Σχηματικόπύλης OR 32 Bit μέσωδιαύλου(or_32_bit_in_bus)



Εικόνα 49

3.3.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά η πύλη OR 32 bit εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το πίνακα αλήθειας είδαμε ότι λειτουργεί σωστά.

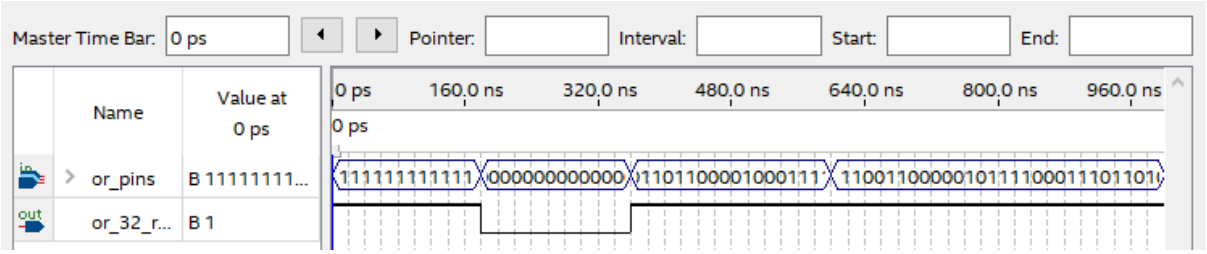
Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε or_32_bit_with_bus_Waveform.

Πίνακας αληθείας OR 32 Bit

or_32_bit			
input1	...	input32	y
0	0	0	0
0	1 or 0	1	1
1	1 or 0	0	1
1	1	1	1

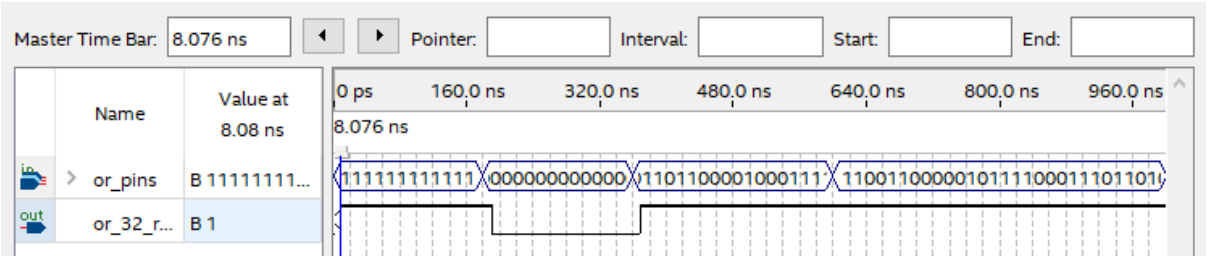
Πίνακας9

Λογικήεξομοίωσηπύλης OR 32 bit μέσωδιαύλου(or_32_bit_with_bus_Waveform)



Εικόνα 50

Χρονική εξομοίωση πύλης OR 32 bit μέσω διαύλου(or_32_bit_with_bus_Waveform)



Εικόνα 51

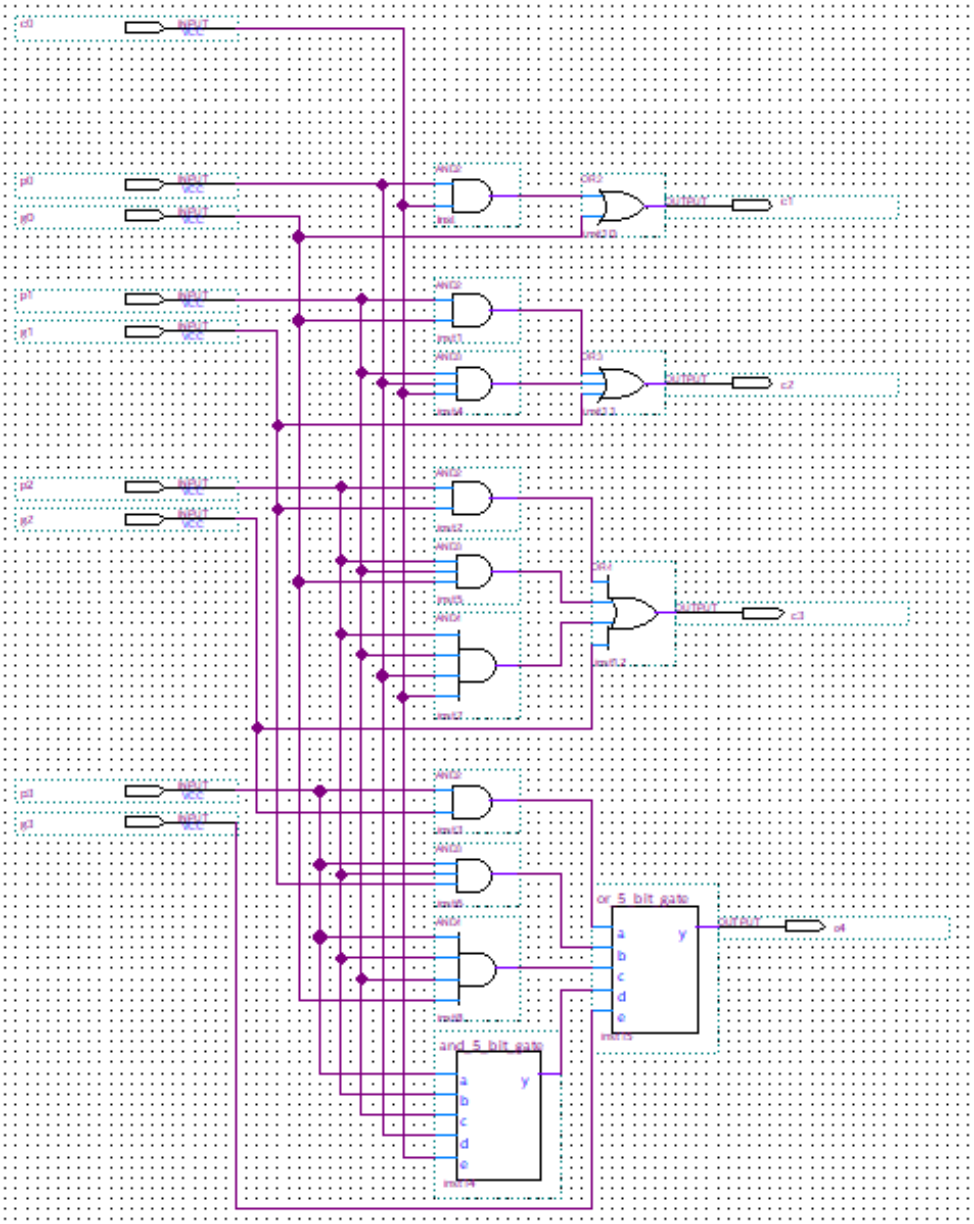
3.4Γεννήτρια πρόβλεψης κρατουμένων 4 μπιτ (4 bitcarrylookcarry)

3.4.1 Υλοποίηση

Η γεννήτρια πρόβλεψης κρατουμένων 4 bit χρησιμοποιείται για αυτό που λέει και το όνομα της δηλαδή να προβλέπει τα κρατούμενα ώστε να μην υπάρχει καθυστέρηση. Για την κατασκευή του σχηματικού της γεννήτριας χρειάστηκαν μια (1) είσοδος c0 για το εισερχόμενο κρατούμενο, τέσσερις (4) είσοδοι p0,p1,p2,p3 για την διάδοση των σημάτων, τέσσερις (4) είσοδοι g0,g1,g2,g3 για την δημιουργία σημάτων και τέσσερις (4) έξοδοι c0,c1,c2,c3 που είναι τα προβλεπόμενα κρατούμενα. Όσον αφορά τις λογικές πύλες διαθέτει τέσσερις (4) AND δυο (2) εισόδων, τρείς (3) AND τριών (3) εισόδων, δυο (2) AND τεσσάρων (4) εισόδων, μια (1) AND πέντε (5) εισόδων και τέσσερεις (4) OR, η πρώτη δυο εισόδων, η δεύτερη τριών, η τρίτη τεσσάρων και η τέταρτη 5 εισόδων. Οι πύλες OR και AND πέντε εισόδων δεν υπήρχαν στο πρόγραμμα Quartus Elite Prime όποτε και έγινε η κατασκευή τους, την οποία θα αναλύσουμε παρακάτω. [7],[11],[13]

Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε four_carry_look_ahead.

Γεννήτρια πρόβλεψης κρατουμένων 4 bit (four_carry_look_ahead)



Εικόνα 52

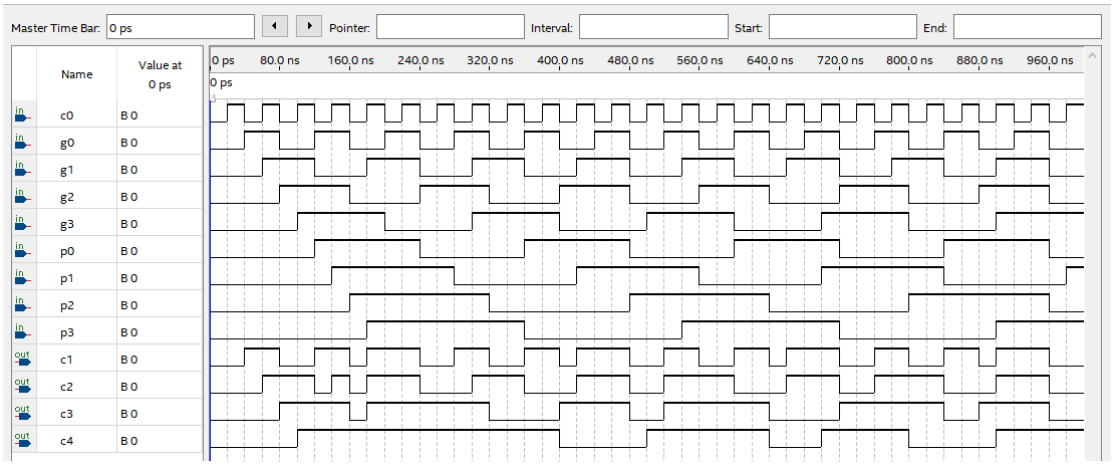
3.4.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά ο ελεγκτής υπερχείλισης (overflow) εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με την λογική του σχηματικού είδαμε ότι λειτουργεί σωστά.

Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε four_carry_look_ahead_Waveform.

Λογική εξομοίωση της γεννήτριας πρόβλεψης κρατουμένου

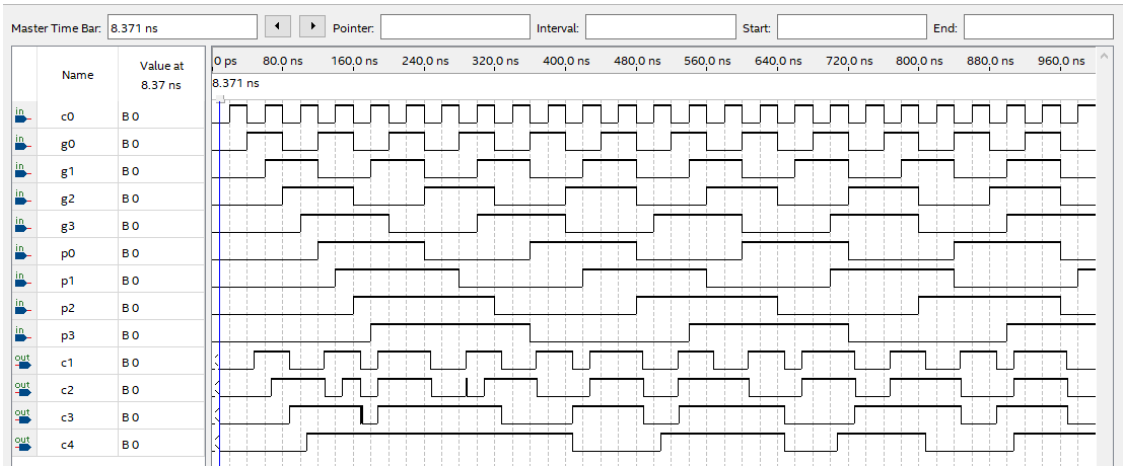
(four_carry_look_ahead_Waveform)



Εικόνα 53

Χρονική εξομοίωση της γεννήτριας πρόβλεψης κρατουμένου

(four_carry_look_ahead_Waveform)



Εικόνα 54

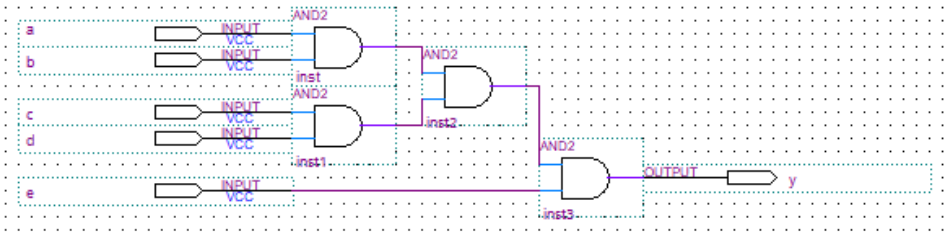
3.5Πύλη AND 5 μπιτ (5 bit AND gate)

3.5.1 Υλοποίηση

Όσον αφορά την πύλη AND 5 bit προβήκαμε στον σχεδιασμό της καθώς δεν υπήρχε έτοιμο στοιχείο/μπλοκ στο πρόγραμμα Quartus Elite Prime για να χρησιμοποιήσουμε. Για την δημιουργία της πύλης αυτής χρειαστήκαμε τέσσερεις πύλες AND δυο εισόδων, 5 εισόδους και μια έξοδο.

Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε and_5_bit_gate.

Σχηματικόπύλης AND 5 bit (and_5_bit_gate)



Εικόνα 55

3.5.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά η πύλη AND 5 bit εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το πίνακα αλήθειας της απλής πύλης AND είδαμε ότι λειτουργεί σωστά, δηλαδή το y είναι 1 μόνο όταν και οι 5 είσοδοι είναι λογικό 1.

Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε and_5_bit_Waveform.

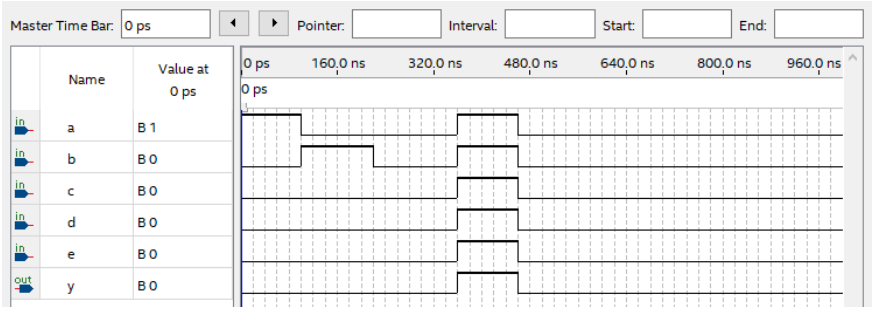
Πίνακας αληθείας της πύλης AND

AND

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

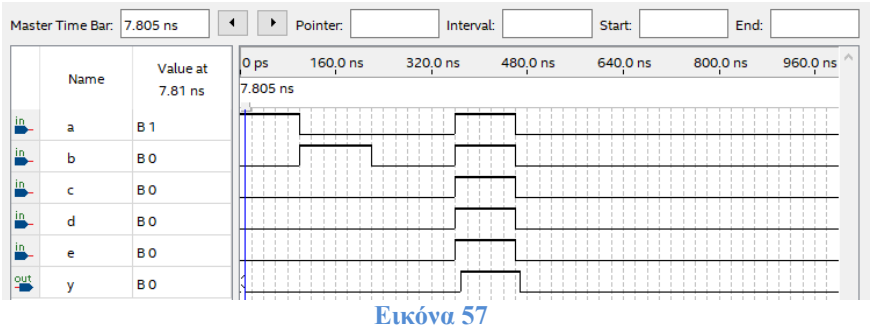
Πίνακας10

Λογικήεξομοίωσηπύλης AND 5 bit (and_5_bit_Waveform)



Εικόνα 56

Χρονικήεξομοίωσηπύλης AND 5 bit (and_5_bit_Waveform)



Εικόνα 57

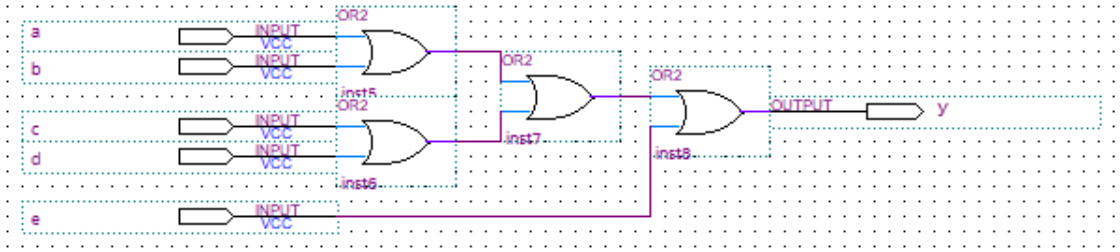
3.6Πύλη OR 5μπιτ (5 bit OR gate)

3.6.1 Υλοποίηση

Όσον αφορά την πύλη OR 5 bit όπως και για την παραπάνω, προβήκαμε στον σχεδιασμό της καθώς δεν υπήρχε έτοιμο στοιχείο/μπλοκ στο πρόγραμμα Quartus Elite Prime για να χρησιμοποιήσουμε. Για την δημιουργία της πύλης αυτής χρειαστήκαμε τέσσερεις πύλες OR δυο εισόδων, 5 εισόδους και μια έξοδο.

Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε or_5_bit_gate.

Σχηματικόπύλης OR 5 bit (or_5_bit_gate)



Εικόνα 58

3.6.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά η πύλη OR 5 bit εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το πίνακα αλήθειας της απλής πύλης OR είδαμε ότι λειτουργεί σωστά, δηλαδή το y είναι 0 μόνο όταν και οι 5 είσοδοι είναι λογικό 0.

Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε or_5_bit_Waveform.

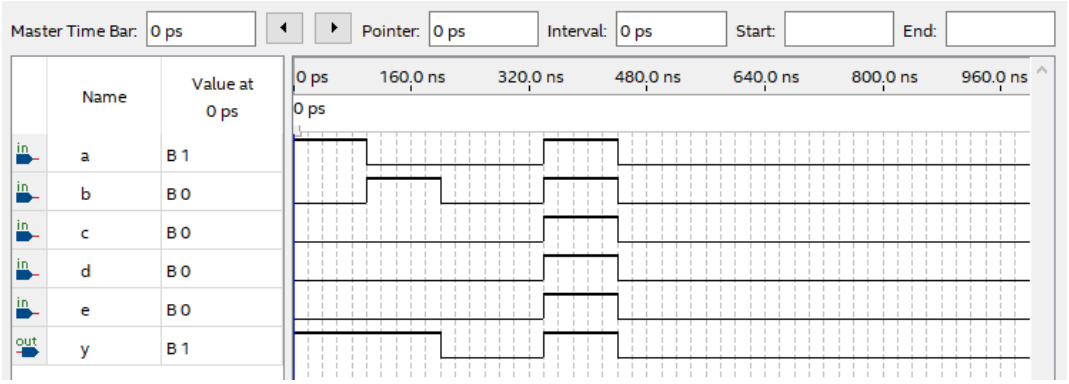
Πίνακας αληθείας της πύλης OR

OR

a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

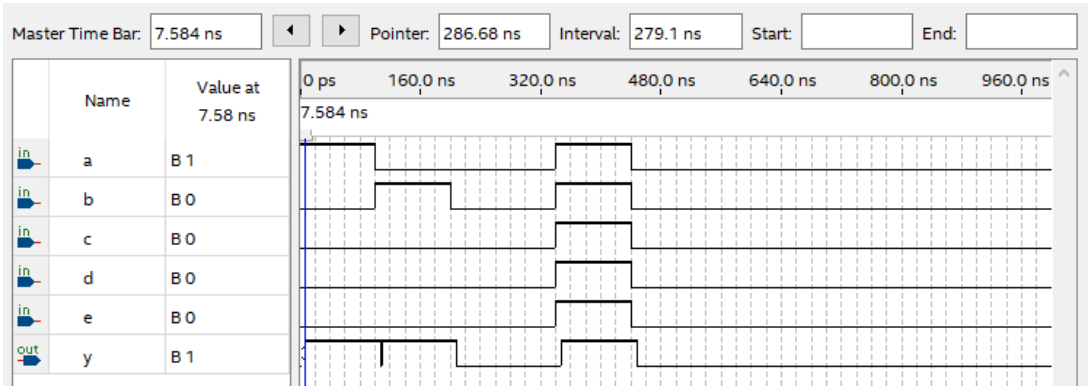
Πίνακας 11

Λογικήεξομοίωσηπύλης OR 5 bit (or_5_bit_Waveform)



Εικόνα 59

Χρονικήεξομοίωσηπύλης OR 5 bit (or_5_bit_Waveform)



Εικόνα 60

3.7 Αθροιστής 4 μπιτ με την γεννήτρια πρόβλεψης κρατουμένου (4 bitadderwithcarrylookcarry)

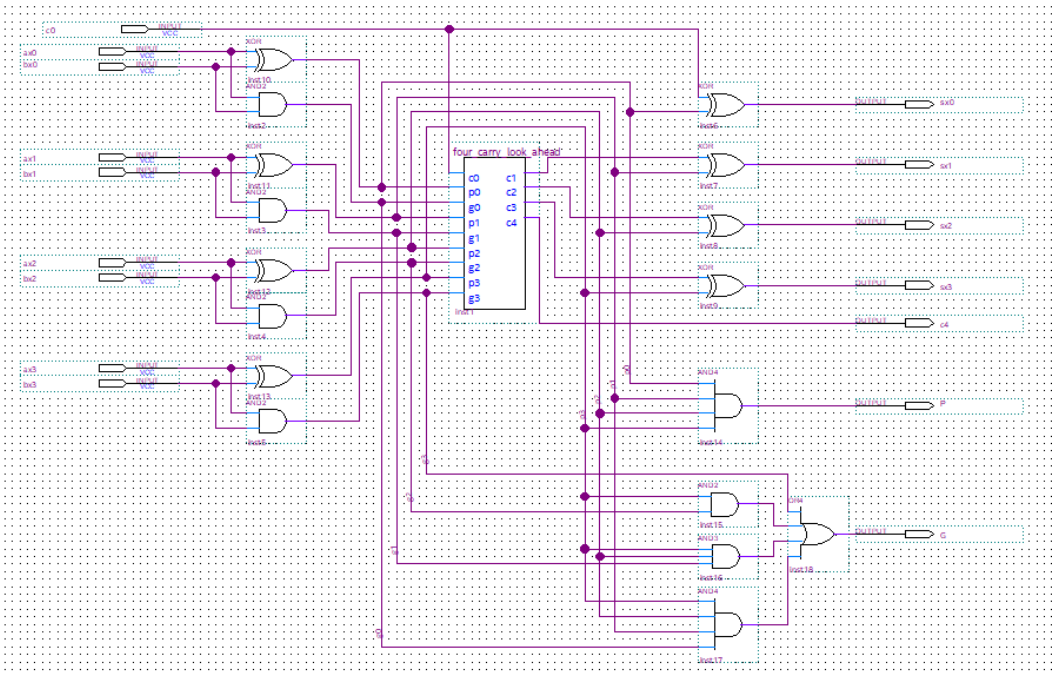
3.7.1 Υλοποίηση

Ο αθροιστής 4 bit με την γεννήτρια πρόβλεψης κρατουμένων χρησιμοποιείται με σκοπό να προβλέπει πιο γρήγορα τα κρατούμενα και μαζί με τους αθροιστές που διαθέτει να υπολογίζει ταχύτερα το αποτέλεσμα. Το σχηματικό του αθροιστή περιέχει την γεννήτρια πρόβλεψης κρατουμένων που αναλύσαμε παραπάνω, 4 αθροιστές οι οποίο αποτελούνται από δυο XOR δυο (2) εισόδων και μια AND δυο (2) εισόδων. Επιπλέον, έχει μια είσοδο c0 για το κρατούμενο, 4 εισόδους ax0,ax1,ax2,ax3 για εισαγωγή του πρώτου αριθμού, 4 εισόδους bx0, bx1,bx2, bx3 για τον δεύτερο και 4 εξόδους sx0,sx1,sx2,sx3 για το αποτέλεσμα. Επιπροσθέτως, διαθέτει μια πύλη AND τεσσάρων είσοδών με έξοδο P που είναι για την διάδοση των σημάτων, τρείς πύλες AND με 2, 3, 4 εισόδους η κάθε μια ξεχωριστά, μια OR τεσσάρων (4) εισόδων, οι οποίες συνδέονται όπως φαίνεται στην

παρακάτω εικόνα δίνοντας μας την έξοδο G,η οποία είναι για την δημιουργία των σημάτων. [7],[11]

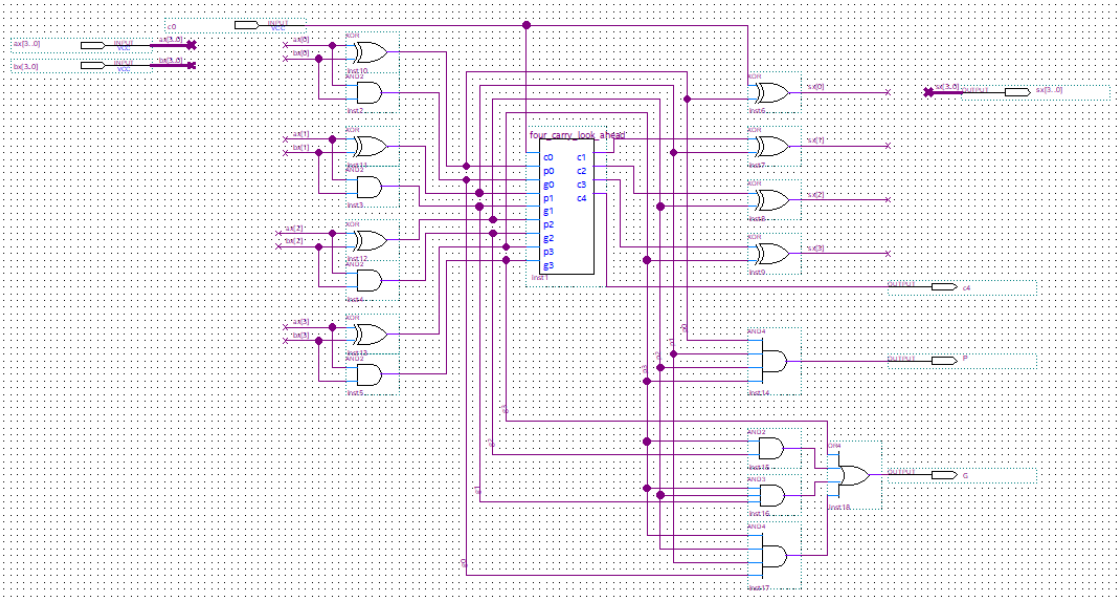
Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε four_bit_adder_with_carry_look_ahead.

<u>Σχηματικό αθροιστή 4 bit με την γεννήτρια πρόβλεψης κρατουμένων</u>
(four_bit_adder_with_carry_look_ahead)



Εικόνα 61

<u>Σχηματικό αθροιστή 4 bit με την γεννήτρια πρόβλεψης κρατουμένων με την χρήση διαύλων για δοκιμή</u>(four_bit_adder_with_carry_look_ahead_for_test_with_buses)



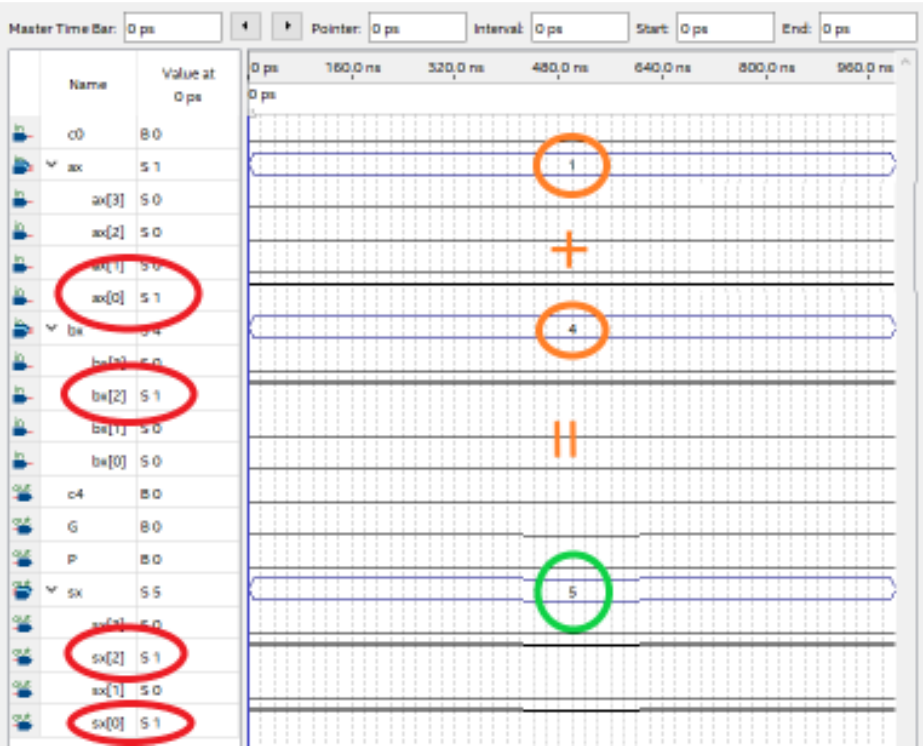
Εικόνα 62

3.7.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά ο αθροιστής 4 bit με την γεννήτρια πρόβλεψης κρατουμένων δημιουργήσαμε άλλο ένα σχηματικό το οποίο φαίνεται στην παραπάνω εικόνα με την μόνη διαφορά ότι παίρνει την είσοδο των bit μέσω διαύλων, το ίδιο κάναμε και με την έξοδο. Αυτή η δημιουργία του σχηματικού, μέσω των δίαυλων, έγινε με σκοπό στην εξομοίωση να δηλώσουμε τις εισόδους και την έξοδο σαν προσημασμένο αριθμό και να δούμε αν εκτελεί σωστά την πράξη. Έτσι με αυτό τον τρόπο θα συγκρίνουμε τα bits και των δυο λογικών προσομοιώσεων για να δούμε αν το σχηματικό που θέλουμε να χρησιμοποιήσουμε λειτουργεί σωστά. Επιπρόσθετα, αφού είδαμε ότι λειτουργεί σωστά η λογική εξομοίωση εκτελέσαμε και μια χρονική για να δούμε την καθυστέρηση.

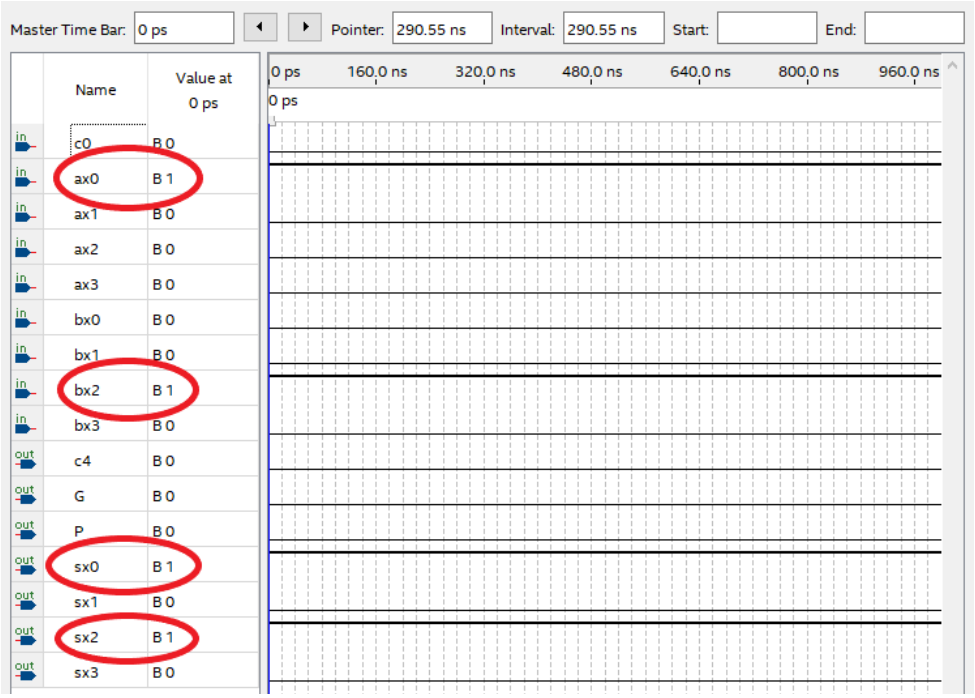
Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime για το σχηματικό χωρίς την χρήση διαύλων το ονομάσαμε four_bit_adder_with_carry_look_ahead_Waveform ενώ, αυτό που χρησιμοποιήσαμε για δοκιμή το ονομάσαμε four_bit_adder_with_carry_look_ahead_for_test_with_buses.

Λογική εξομοίωση αθροιστή 4 bit με την γεννήτρια πρόβλεψης κρατουμένων με την χρήση διαύλων για δοκιμή(four_bit_adder_with_carry_look_ahead_for_test_with_buses_Waveform)



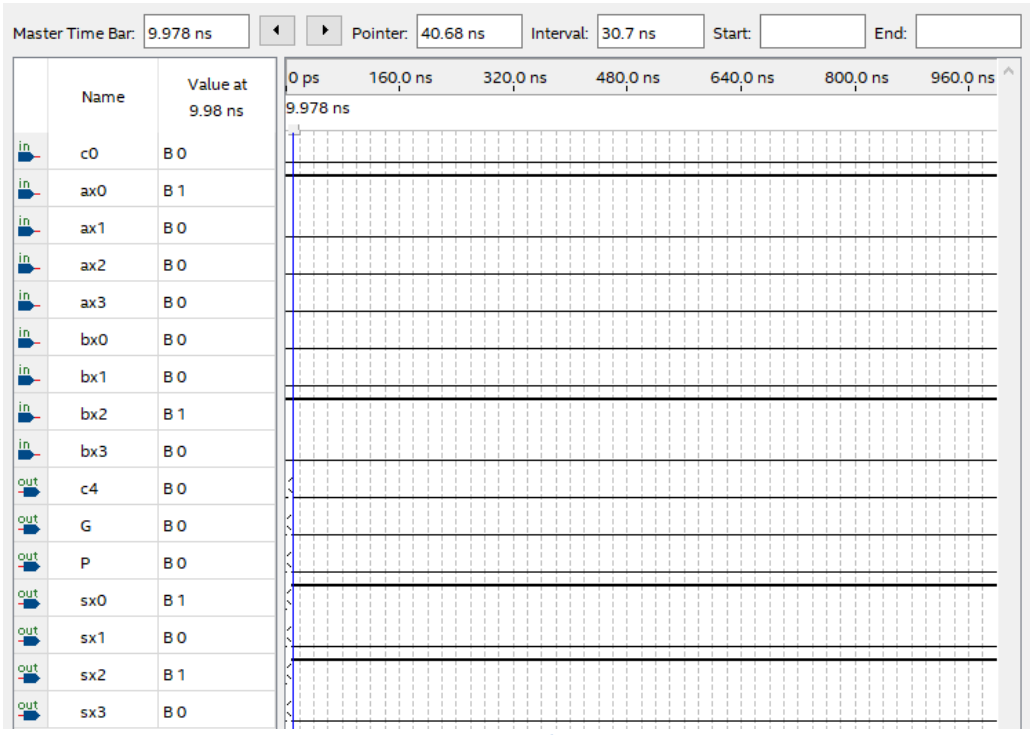
Εικόνα 63

Λογική εξομοίωση αθροιστή 4 bit με την γεννήτρια πρόβλεψης κρατουμένου (four_bit_adder_with_carry_look_ahead_Waveform)



Εικόνα 64

Χρονικήεξομοίωσηαθροιστή 4 bit μετηνγεννήτριαπρόβλεψηςκρατουμένου(four_bit_adder_with_carry_look_ahead_Waveform)



Εικόνα 65

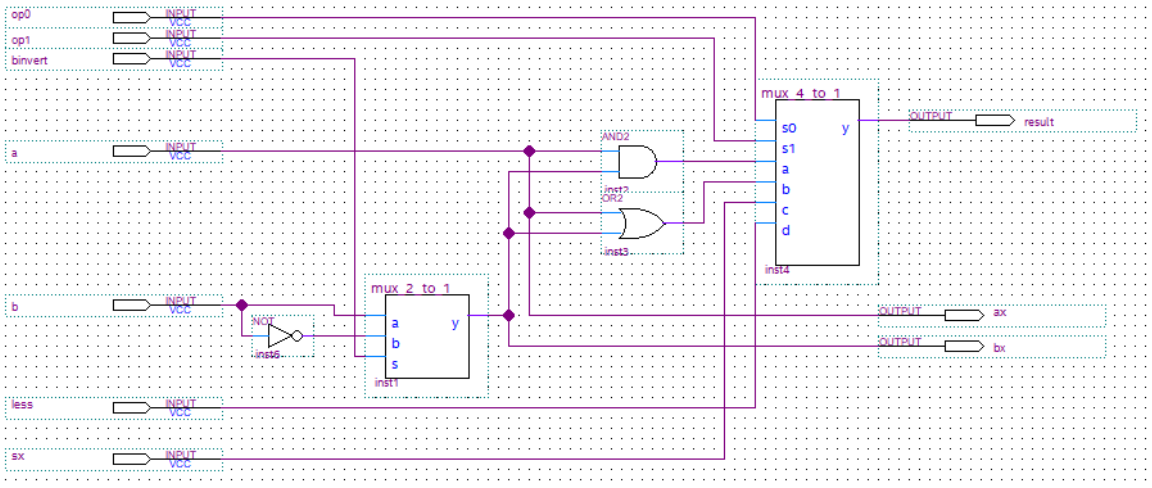
3.8Αριθμητική και λογική μονάδα 1 μπιτ για τον αθροιστή 4 μπιτ με την γεννήτρια πρόβλεψης κρατουμένου (1 bitALUfor 4 bitadderwithcarrylookahead)

3.8.1 Υλοποίηση

Η αριθμητική και λογική μονάδα 1 bit που δημιουργήσαμε εκτελεί τις ίδιες πράξεις με την μονάδα του πρώτου κεφαλαίου με την μόνη διαφορά ότι δεν διαθέτει τον πλήρη αθροιστή καθώς αυτός βρίσκεται στο μπλοκ του αθροιστή 4 bit με την γεννήτρια πρόβλεψης. Άλλη μια διαφορά εντοπίστηκε στις εισόδους και τις εξόδους. Στις εισόδους δεν υπάρχει carryin καθώς το κρατούμενο πηγαίνει στον αθροιστή 4 bit, ενώ παράλληλα προστέθηκε η είσοδος sx στην οποία έρχεται το αποτέλεσμα του αθροιστή των 4 bit. Όσον αφορά τις εξόδους δεν υπάρχει η carryout επειδή το κρατούμενο παράγεται από τον αθροιστή 4 bit και προστέθηκαν δυο ακόμη έξοδοι οι ax,bx, οι οποίοι είναι οι αντίστοιχοι που στην μονάδα ALU του προηγούμενου κεφαλαίου πηγαίνουν στον πλήρη αθροιστή έτσι και σε αυτή την περίπτωση ενώνονται με τον αθροιστή των 4 bit. [11]

Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε one_bit_alu_for_4_bit_adder.

Σχηματικό αριθμητικής και λογικής μονάδας 1 bit για τον αθροιστή 4 bit με την γεννήτρια πρόβλεψης κρατουμένου (one_bit_alu_for_4_bit_adder)



Εικόνα 66

3.8.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά η αριθμητική και λογική μονάδα 1 bit για τον αθροιστή 4 bit με την γεννήτρια πρόβλεψης κρατουμένου εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το πίνακα επιλογών πράξεων είδαμε ότι λειτουργεί σωστά.

Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε one_bit_alu_for_4_bit_adder_Waveform.

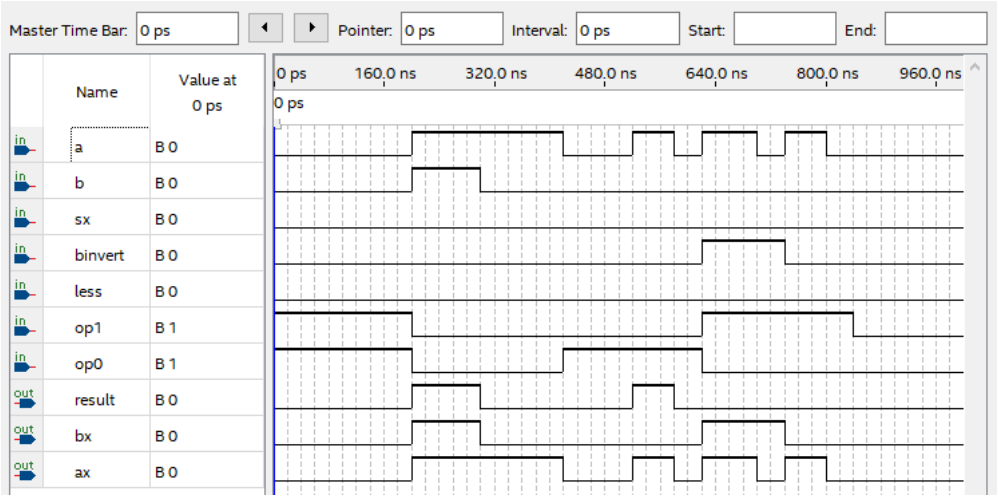
Πίνακας επιλογής πράξεων αριθμητικής και λογικής μονάδας

alu

op1	op0	Bnegate	
0	0	0	and
0	1	0	or
1	0	0	add
1	0	1	subtract
1	1	0	less

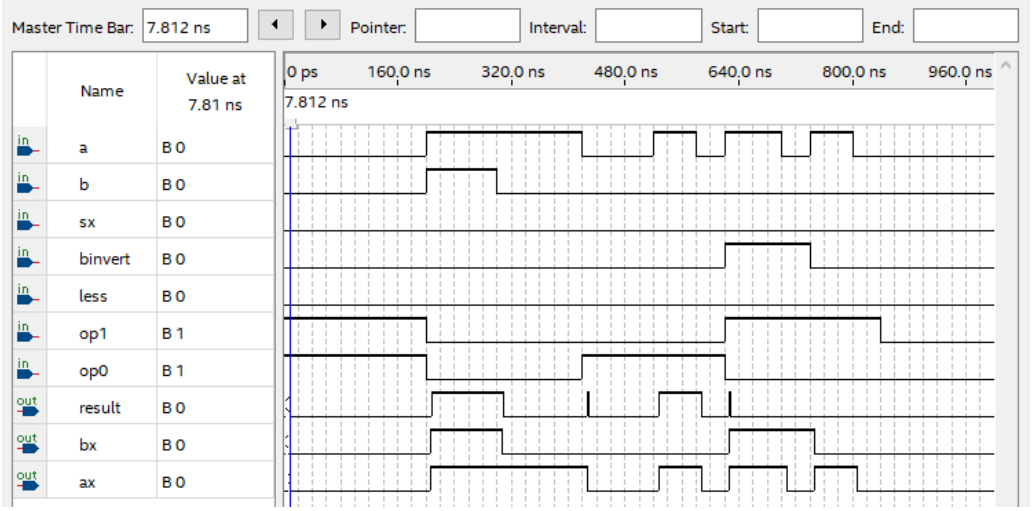
Πίνακας 12

Λογική εξομοίωση της αριθμητικής και λογικής μονάδας 1 bit για τον αθροιστή 4 bit με την γεννήτρια πρόβλεψης κρατουμένου (one_bit_alu_for_4_bit_adder)



Εικόνα 67

Χρονική εξομοίωση της αριθμητικής και λογικής μονάδας 1 bit για τον αθροιστή 4 bit με την γεννήτρια πρόβλεψης κρατουμένου (one_bit_alu_for_4_bit_adder)



Εικόνα 68

3.9 Σύγκριση καθυστέρησης αριθμητικής και λογικής μονάδας με και χωρίς την χρήση της γεννήτριας πρόβλεψης

Όπως είπαμε και στην αρχή του κεφαλαίου δημιουργήσαμε την αριθμητική και λογική μονάδα 32 bit με την χρήση της γεννήτριας πρόβλεψης κρατουμένου με σκοπό να γίνεται η πράξη της πρόσθεσης πιο γρήγορα σε σύγκριση με την απλή αριθμητική και λογική μονάδα. Για να τεκμηριώσουμε ότι όντως λειτουργεί πιο γρήγορά τρέξαμε την εντολή Timing Analyzer στο πρόγραμμα Electric και από την καρτέλα Propagation Delay όπου και διαπιστώθηκε ότι ο χρόνος είναι πολύ λιγότερος σε σχέση με την απλή και αριθμητική μονάδα. Επιπλέον, εκτελέσαμε και χρονικές εξομοιώσεις στην πράξη της πρόσθεση και διαπιστώσαμε επίσης ότι εκτελεί την πράξη πιο γρήγορα. Στις εικόνες που ακολουθούν φαίνεται καθαρά η χρονική τους διαφορά. [11]

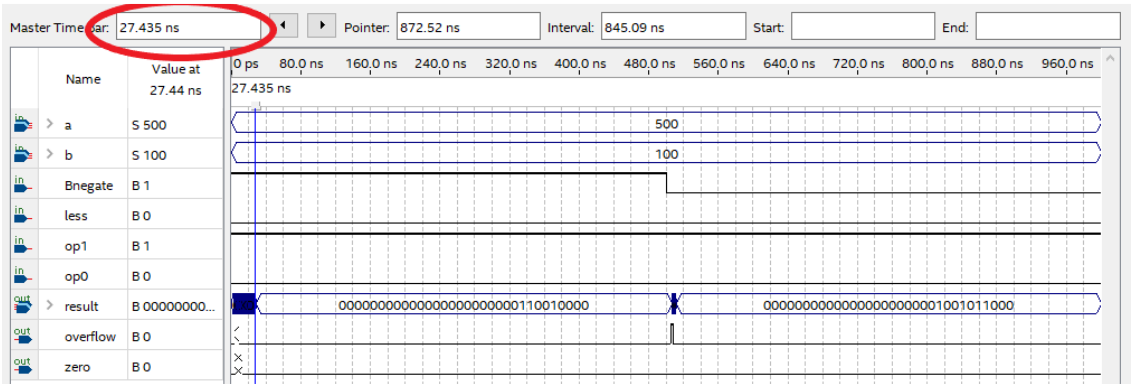
Propagation Delay της απλής αριθμητικής και λογικής μονάδας

Propagation Delay

	Input Port	Output Port	RR	RF	FR	FF
1	a[0]	result[0]	37.170	36.961	37.581	37.372
2	b[0]	result[0]	36.427	36.218	36.815	36.606
3	b[1]	result[0]	36.361	36.152	36.716	36.507
4	a[1]	result[0]	35.245	35.098	36.393	36.184
5	b[2]	result[0]	35.364	35.155	35.739	35.530
6	a[2]	result[0]	34.402	34.255	35.560	35.351
7	a[0]	zero	34.607	34.726	35.018	35.137
8	Bnegate	result[0]	34.846	34.637	34.971	34.762
9	a[3]	result[0]	33.640	33.493	34.685	34.476
10	b[0]	zero	33.864	33.983	34.252	34.371
11	b[4]	result[0]	34.104	33.895	34.543	34.334
12	b[1]	zero	33.798	33.917	34.153	34.272
13	a[1]	zero	32.744	32.801	33.830	33.949
14	b[3]	result[0]	33.686	33.477	34.040	33.831
15	a[4]	result[0]	32.834	32.687	33.984	33.775
16	a[5]	result[0]	32.436	32.289	33.529	33.320
17	b[2]	zero	32.801	32.920	33.176	33.295
18	a[0]	result[31]	32.877	32.782	33.288	33.193
19	a[2]	zero	31.901	31.958	32.997	33.116
20	a[6]	result[0]	32.241	32.094	33.312	33.103

Εικόνα 69

Χρονική εξομοίωση της απλής αριθμητικής και λογικής μονάδας



Εικόνα 70

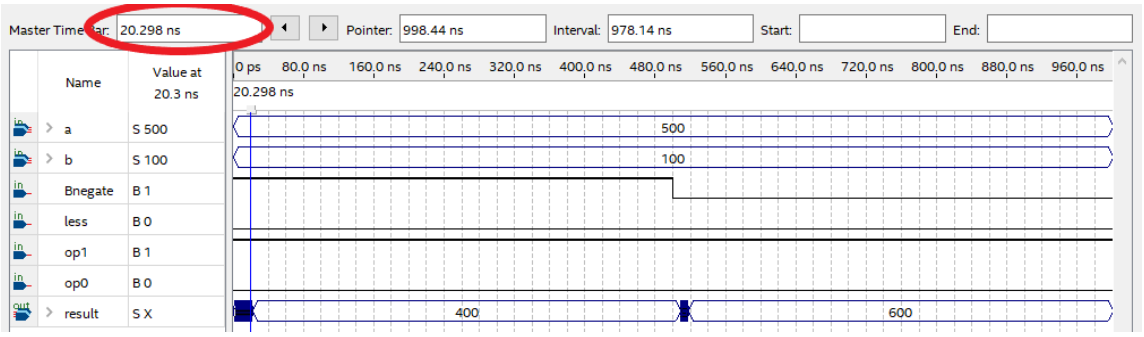
Propagation Delay της αριθμητικής και λογικής μονάδας με την χρήση της γεννήτριας πρόβλεψης κρατουμένου

Propagation Delay

	Input Port	Output Port	RR	RF	FR	FF
1	Bnegate	zero	23.958	24.004	24.306	24.355
2	b[9]	zero	23.669	23.709	24.061	24.116
3	b[11]	zero	23.197	23.237	23.519	23.559
4	b[2]	zero	23.025	23.063	23.443	23.483
5	a[10]	zero	22.959	22.999	23.384	23.424
6	b[5]	zero	22.940	22.980	23.319	23.359
7	b[4]	zero	22.835	22.875	23.206	23.246
8	b[3]	zero	22.764	22.804	23.158	23.198
9	b[10]	zero	22.713	22.753	23.115	23.155
10	b[0]	zero	22.557	22.597	22.940	22.985
11	a[8]	zero	22.494	22.534	22.899	22.958
12	b[1]	zero	22.505	22.545	22.821	22.867
13	b[6]	zero	22.287	22.327	22.722	22.762
14	a[6]	zero	22.286	22.326	22.656	22.715
15	b[8]	zero	22.238	22.286	22.601	22.641
16	a[9]	zero	22.221	22.261	22.575	22.634
17	a[5]	zero	22.050	22.090	22.402	22.461
18	a[1]	zero	22.087	22.127	22.404	22.444
19	a[4]	zero	21.957	21.997	22.301	22.360
20	b[7]	zero	21.846	21.886	22.249	22.289

Εικόνα 71

Χρονική εξομοίωση της αριθμητικής και λογικής μονάδας με την χρήση της γεννήτριας πρόβλεψης κρατουμένου



Εικόνα 72

4.Κεφάλαιο 4

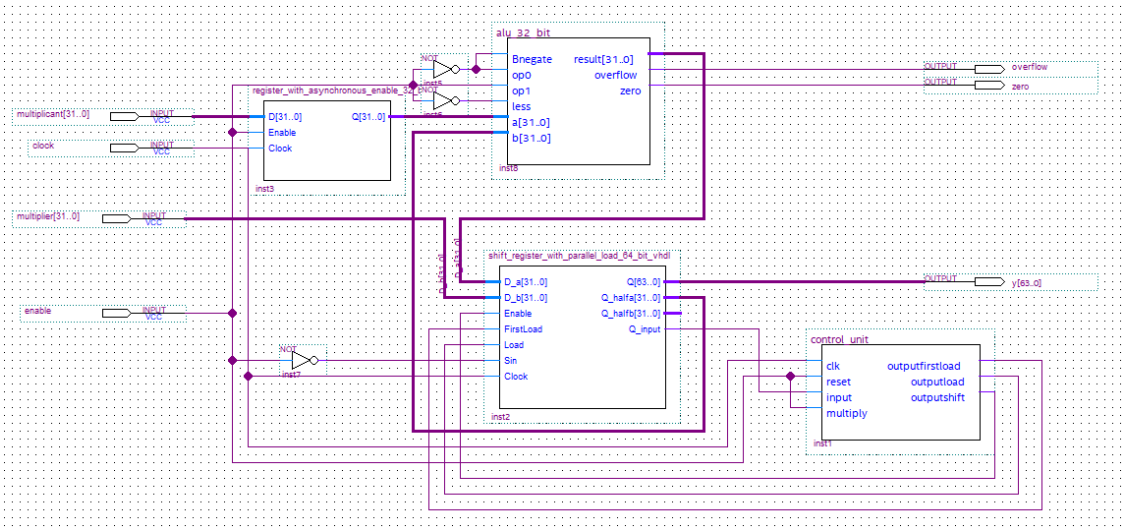
4.1 Πολλαπλασιασμός (Multiply)

4.1.1 Υλοποίηση

Στο παρών κεφάλαιο θα αναλύσουμε τον μηχανισμό του πολλαπλασιασμού, ο οποίος γίνεται με την χρήση της αριθμητικής και λογικής μονάδας καθώς και με βοήθεια κάποιων άλλων στοιχείων που θα αναφέρουμε. Ο πολλαπλασιασμός επιτυγχάνεται με μια σειρά πράξεων πρόσθεσης και δεξιάς μετατόπισης.Πιο συγκεκριμένα ο μηχανισμός πραγματοποιήθηκε με βάση το διάγραμμα ροής που απεικονίζεται παρακάτω.Το σχηματικό του πολλαπλασιασμού που υλοποιήσαμε αποτελείται από τρείς πύλες NOT, μια 32 bit αριθμητική και λογική μονάδα, έναν καταχωρητή 32 bit με ασύγχρονη ενεργοποίηση, έναν καταχωρητή ολίσθησης 64 bit, μια μονάδα ελέγχου, 4 εισόδους και 3 εξόδους. Όσον αφορά τις εισόδους διαθέτει μια είσοδο ονόματι clock που είναι για τον χρόνο, μια είσοδο ονόματι enable που είναι για την ενεργοποίηση, μια multiplicand 32 bit που είναι για τον πολλαπλασιαστέο και μια multiplier 32 bit που είναι ο πολλαπλασιαστής. Από εξόδους διαθέτει την y 64 bit που είναι για το αποτέλεσμα, την overflow για την υπερχείλιση και την zero η οποία γίνεται 1 όταν το τελικό αποτέλεσμα είναι μηδενικό. Για την κατασκευή των στοιχείων του καταχωρητή 32 bit με ασύγχρονη ενεργοποίηση του καταχωρητή 64 bit και της μονάδας ελέγχου θα μιλήσουμε παρακάτω. [11],[14],[15]

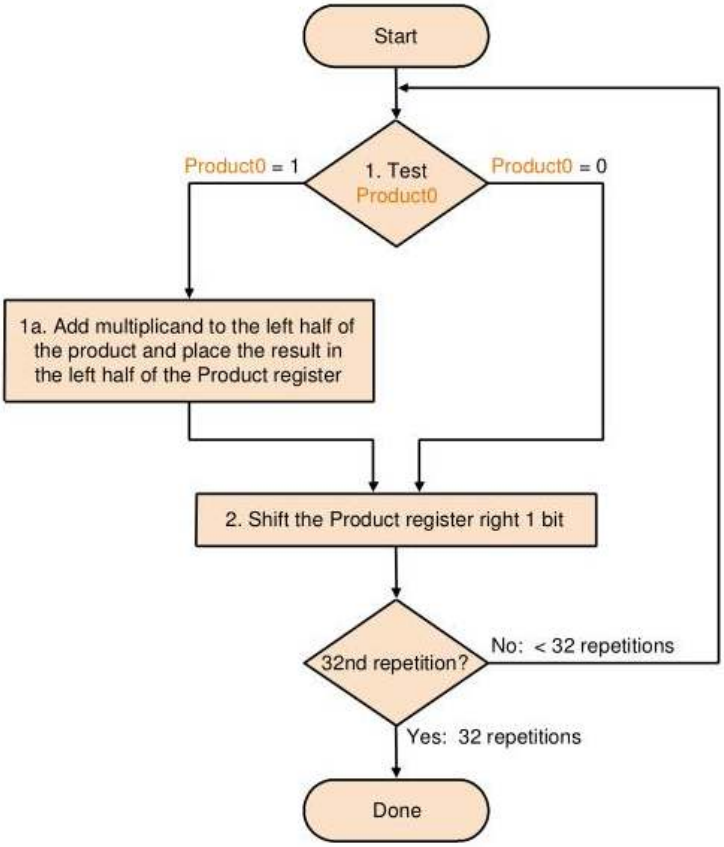
Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε multiply.

<u>Σχηματικό πολλαπλασιασμού</u> (multiply)



Εικόνα 73

<u>Διάγραμμα ροής πολλαπλασιασμού</u> (Flowchart)



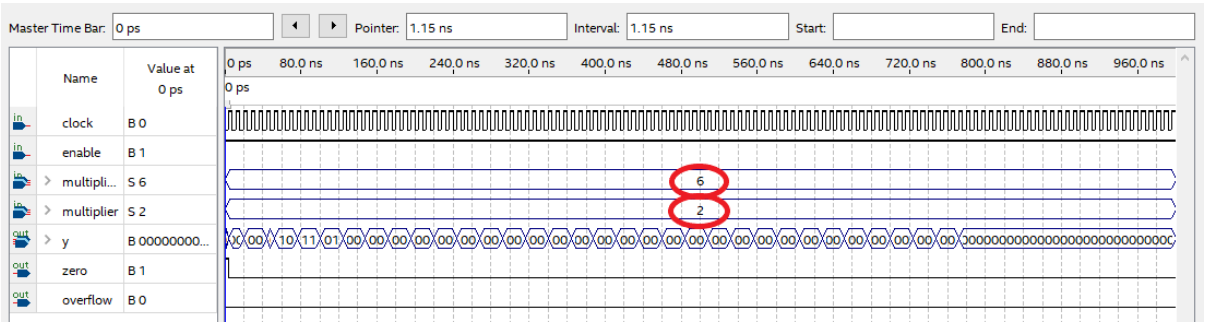
Εικόνα 74

4.1.2 Εξομοίωση

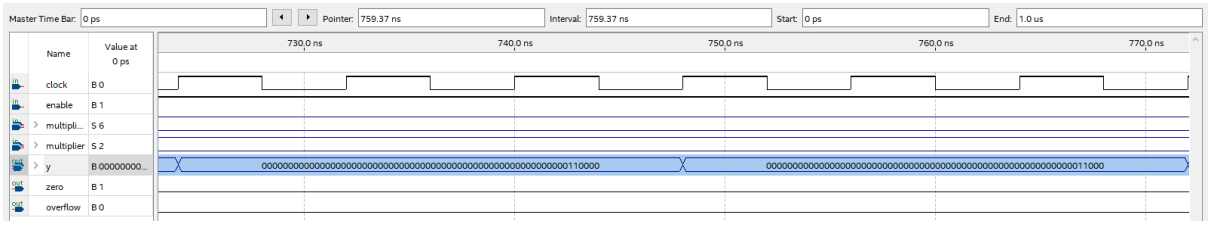
Για να δούμε αν λειτουργεί σωστά ο μηχανισμός του πολλαπλασιασμού εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το αποτέλεσμα είδαμε ότι λειτουργεί σωστά.

Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε multiply_Waveform.

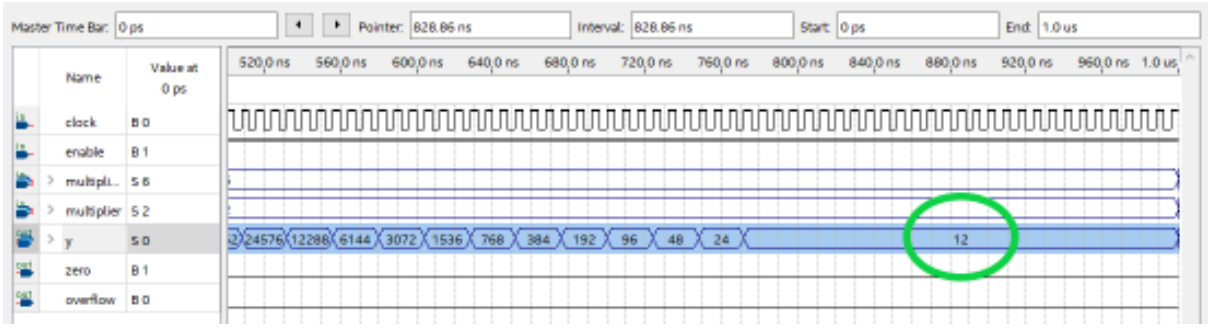
<u>Λογικές εξομοιώσεις του πολλαπλασιασμού</u> (multiply_Waveform)



Εικόνα 75

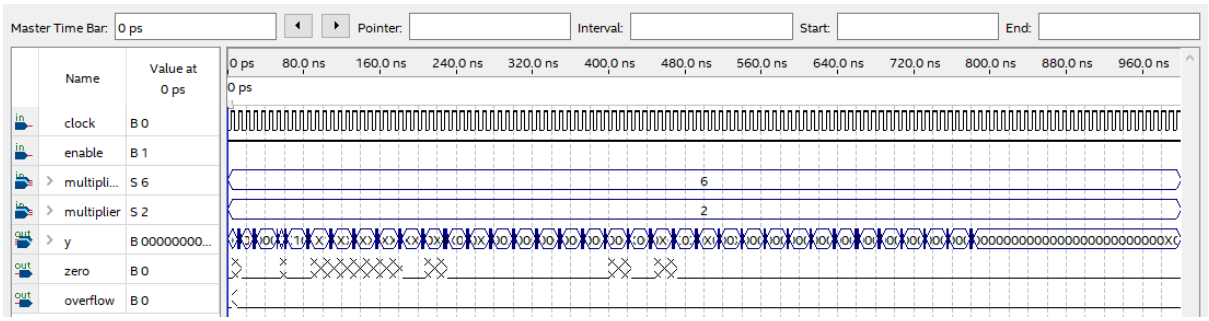


Εικόνα 76



Εικόνα 77

<u>Χρονική εξομοίωση του πολλαπλασιασμού</u> (multiply_Waveform)



Εικόνα 78

4.2Καταχωρητής 32 μπιτ με ασύγχρονη ενεργοποίηση (32 bitregisterwithasynchronousenable)

4.2.1 Υλοποίηση

Η δημιουργία του καταχωρητή 32 bit με την ασύγχρονη ενεργοποίηση έγινε με την χρήση της γλώσσας VHDL. Ο καταχωρητής αποτελείται από μια είσοδο D 32 bit, μια είσοδο clock που είναι για τον χρόνο, μια είσοδο enable που είναι για την ενεργοποίηση και μια έξοδο Q 32 bit. Η λειτουργία του καταχωρητή είναι να κρατάει αποθηκευμένο τον αριθμό της εισόδου με σκοπό όταν ενεργοποιηθεί η είσοδος enable να το εμφανίσει στην έξοδο Q 32 bit. [11],[14]

Το αρχείο του προγράμματος VHDL στο πρόγραμμα Quartus Elite Prime το ονομάσαμε register_with_asynchronous_enable_32_bit_vhdl.

Κώδικας VHDL του καταχωρητή 32 bit με ασύγχρονη ενεργοποίηση

(register_with_asynchronous_enable_32_bit_vhdl)

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

ENTITY register_with_asynchronous_enable_32_bit_vhdl IS

    PORT ( D: IN STD_LOGIC_VECTOR(31 DOWNTO 0);
           Enable, Clock: IN STD_LOGIC;
           Q: OUT STD_LOGIC_VECTOR(31 DOWNTO 0));

END register_with_asynchronous_enable_32_bit_vhdl ;

ARCHITECTURE behavioral OF register_with_asynchronous_enable_32_bit_vhdl IS
BEGIN
    PROCESS ( Clock )
    BEGIN

        IF rising_edge(Clock) THEN
          IF Enable = '1' THEN
            Q <= D;
          END IF;
        END IF;
    END PROCESS;
END behavioral;
```

Εικόνα 79

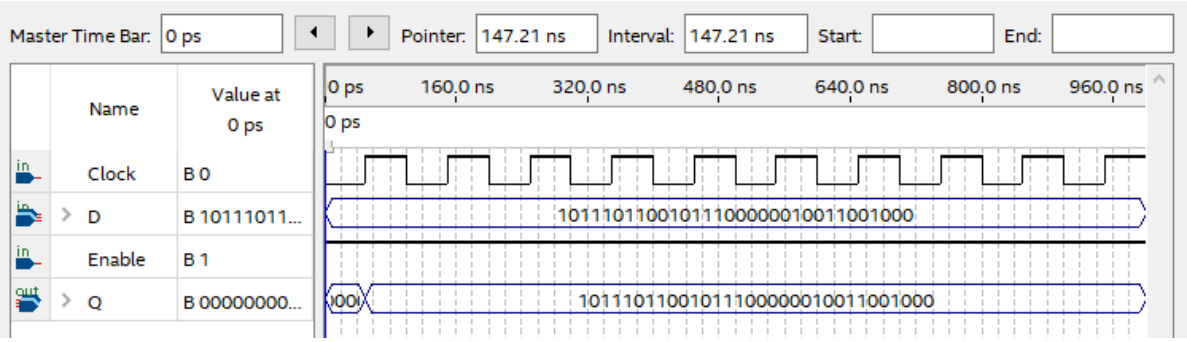
4.2.2 Εξομοίωση

Για να δούμε αν λειτουργεί σωστά ο καταχωρητής 32 bit με ασύγχρονη ενεργοποίηση εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το αποτέλεσμα διαπιστώσαμε ότι λειτουργεί σωστά.

Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε register_with_asynchronous_enable_32_bit_vhdl_Waveform.

Λογική εξομοίωση του καταχωρητή 32 bit με ασύγχρονη ενεργοποίηση

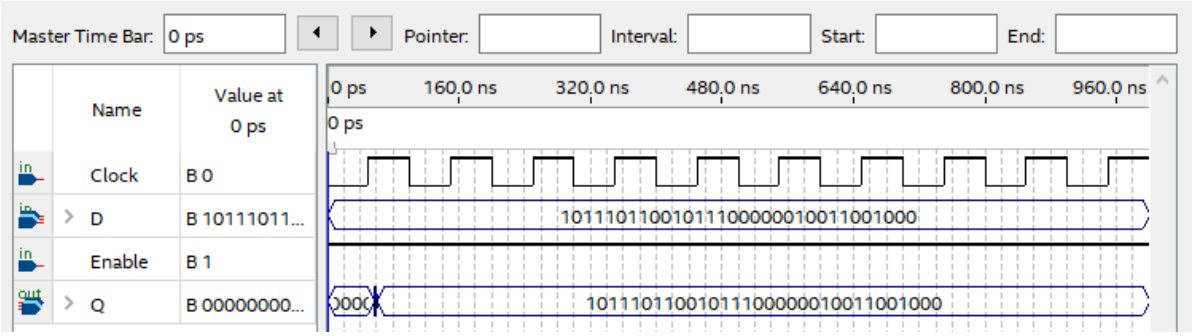
(register_with_asynchronous_enable_32_bit_vhdl_Waveform)



Εικόνα 80

Χρονική εξομοίωση του καταχωρητή 32 bit με ασύγχρονη ενεργοποίηση

(register_with_asynchronous_enable_32_bit_vhdl_Waveform)



Εικόνα 81

4.3Καταχωρητής ολίσθησης 64 μπιτ με παράλληλη φόρτιση (64 bitshiftregisterwithparallelload)

4.3.1 Υλοποίηση

Η δημιουργία του καταχωρητή ολίσθησης 64 bit με παράλληλη φόρτιση έγινε με την χρήση της γλώσσας VHDL καθώς το κύκλωμα σχηματικά θα ήταν πολύ δύσκολο να το υλοποιήσουμε. Ο καταχωρητής αποτελείται από δυο εισόδους την D_a, την D_b που είναι για την εισαγωγή του 64 bit αριθμού, όπου η πρώτη είσοδος έρχεται από την μονάδα ALU και η άλλη από την είσοδο multiplier των 32 bit. Επιπλέον, διαθέτειτην είσοδο enable που είναι για την ενεργοποίηση της ολίσθησης, την είσοδο FirstLoad που είναι για την φόρτωση όλου του αριθμού, την είσοδο Load που είναι για την φόρτωση του μισού αριθμού, την είσοδο Sin που είναι ο αριθμός του τελευταίου ψηφίου που θα ολισθηθεί και η είσοδος Clock που είναι για το ρολόι. Από εξόδους έχουμε την Q που είναι για το αποτέλεσμα, την Q_halfa, Q_halfb που είναι η Q 64 Bit χωρισμένη σε δυο 32 bit αριθμούς και την έξοδος Q_input που είναι ο πρώτος αριθμός bit του 64 bit αριθμού. Η λειτουργία του καταχωρητή είναι να κρατάει αποθηκευμένο τον αριθμό της εισόδου με σκοπό να τον ολισθήσει ή να του φορτώσει κάποιο καινούργιο αποτέλεσμα από την ALU. [11],[14]

Το αρχείο του προγράμματος VHDL στο πρόγραμμα Quartus Elite Prime το ονομάσαμε shift_register_with_parallel_load_64_bit_vhdl.

Κώδικας VHDL του καταχωρητή ολίσθησης 64 bit με παράλληλη φόρτιση (shift_register_with_parallel_load_64_bit_vhdl)

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

ENTITY shift_register_with_parallel_load_64_bit_vhdl IS
        PORT
        (

                D_a         : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
                D_b         : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
                Enable      : IN STD_LOGIC;
                FirstLoad   : IN STD_LOGIC;
                Load        : IN STD_LOGIC;
                Sin         : IN STD_LOGIC;
                Clock       : IN STD_LOGIC;
                Q           : OUT STD_LOGIC_VECTOR(63 DOWNTO 0);
                Q_halfa     : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
                Q_halfb     : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
                Q_input     : OUT STD_LOGIC

         );
END shift_register_with_parallel_load_64_bit_vhdl;

ARCHITECTURE behavioral OF shift_register_with_parallel_load_64_bit_vhdl IS
        SIGNAL Qt: STD_LOGIC_VECTOR(63 DOWNTO 0);
BEGIN


PROCESS (Clock)

 variable D : STD_LOGIC_VECTOR(63 DOWNTO 0);

 BEGIN

 D := D_a & D_b;

  IF rising_edge(Clock) THEN
   IF FirstLoad = '1' THEN
    Qt <= D;
   ELSIF Load = '1' THEN
     Qt(63 downto 32) <= D_a;
   ELSIF Enable = '1' THEN
    Genbits: FOR i IN 0 TO 62 LOOP
      Qt(i) <= Qt(i+1);
     END LOOP;
    Qt(63) <= Sin;
  END IF;
 END IF;
END PROCESS;
 Q <= Qt;

 Q_halfa <= Qt(63 downto 32);
 Q_halfb <= Qt(31 downto 0);
 Q_input <= Qt(0);


END behavioral;
```

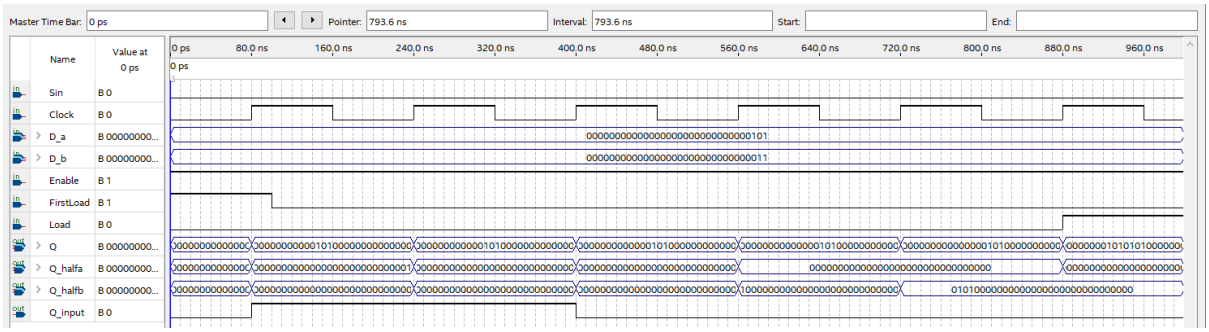
Εικόνα 82

4.3.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά ο καταχωρητής ολίσθησης 64 bit με παράλληλη φόρτιση εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το αποτέλεσμα διαπιστώσαμε ότι λειτουργεί σωστά.

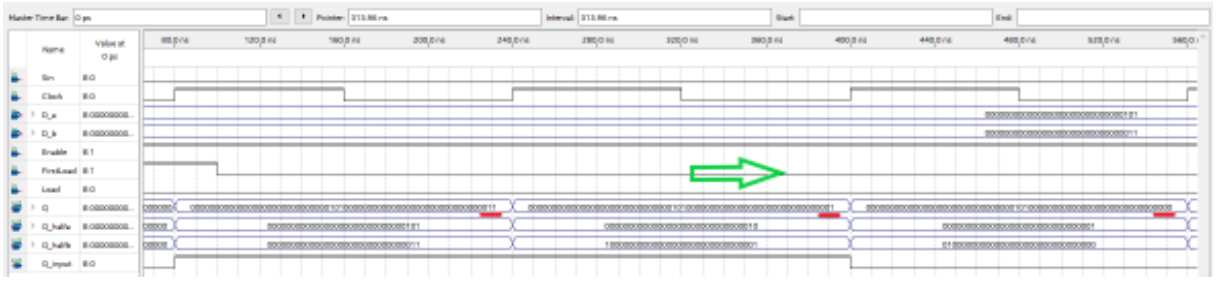
Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε shift_register_with_parallel_load_64_bit_vhdl _Waveform.

Λογικές εξομοιώσεις του καταχωρητή ολίσθηση 64 bit με παράλληλη φόρτιση(shift_register_with_parallel_load_64_bit_vhdl_Waveform)



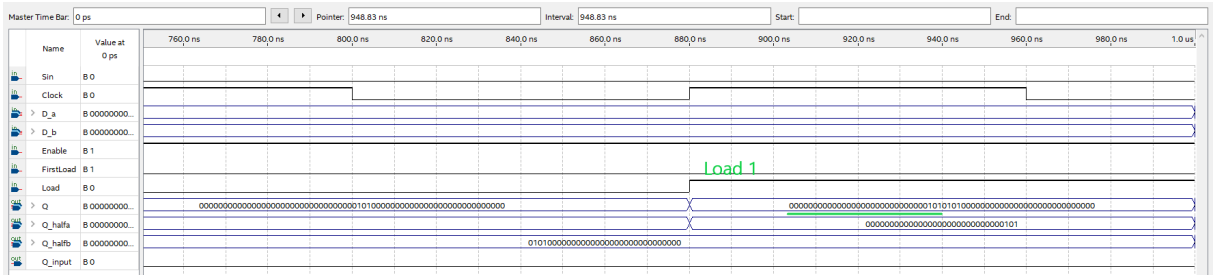
Εικόνα 83

Ολίσθηση



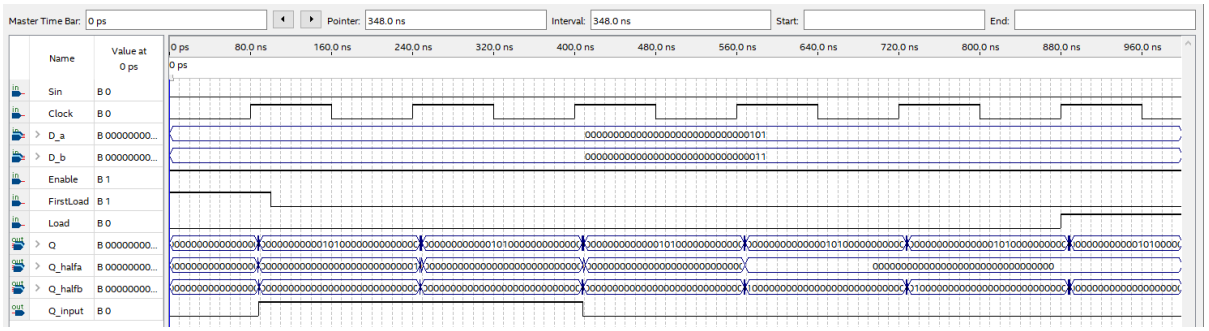
Εικόνα 84

Φόρτωση load



Εικόνα 85

Χρονική εξομοίωση του καταχωρητή ολίσθηση 64 bit με παράλληλη φόρτιση(shift_register_with_parallel_load_64_bit_vhdl_Waveform)



Εικόνα 86

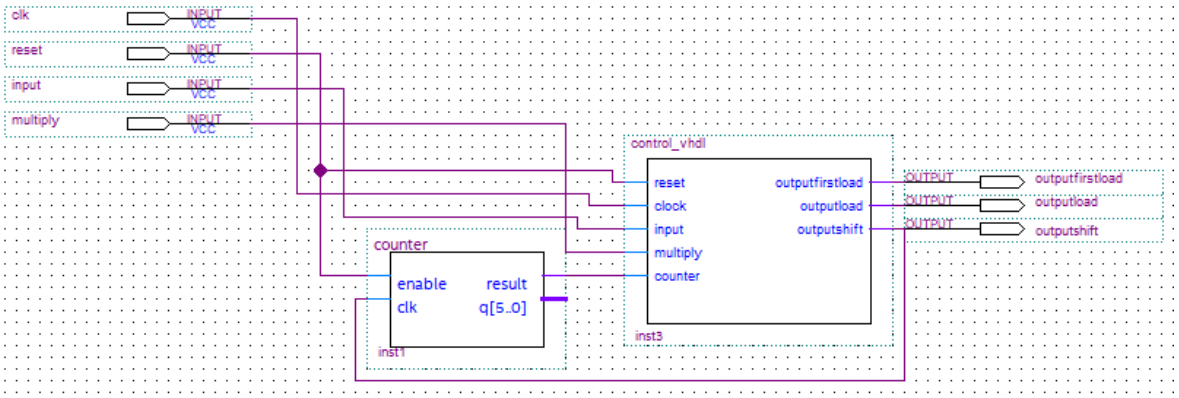
4.4Μονάδα ελέγχου (Controlunit)

4.4.1 Υλοποίηση

Η μονάδα ελέγχου χρησιμοποιείται για να καθοδηγεί σωστά τις λειτουργίες των καταχωρητών και της μονάδας ALU. Η μονάδα ελέγχου στο σχηματικό της αποτελείται από ένα μετρητή counter που είναι υπεύθυνος για να μετράει 32 κύκλους μηχανής και μια ακόμα υποδεέστερη μονάδα ελέγχου. Επιπλέον, το σχηματικό της διαθέτει τέσσερεις (4) εισόδους,μια clk για το κύκλους μηχανής, μια reset που όταν γίνεται λογικό 1 ενεργοποιούνται τα συστήματα, μια input που είναι για το πρώτο bit του 64 bit καταχωρητή και η είσοδος multiply που είναι για την ενεργοποίηση των καταστάσεων στην υποδεέστερη μονάδα ελέγχου. Οι έξοδοι που διαθέτει είναι η outputfirstload, η outputload και η outshift οι οποίες προορίζονται για τον καταχωρητή ολίσθησης και οι λειτουργίες τους είναι για αυτά που αναφέραμε πάνω. Για τις υλοποιήσεις του μετρητή counter και της υποδεέστερης μονάδας ελέγχου θα αναφερθούμε στη συνέχεια. [11]

Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε control_unit.

Σχηματικό μονάδας ελέγχου (control_unit)



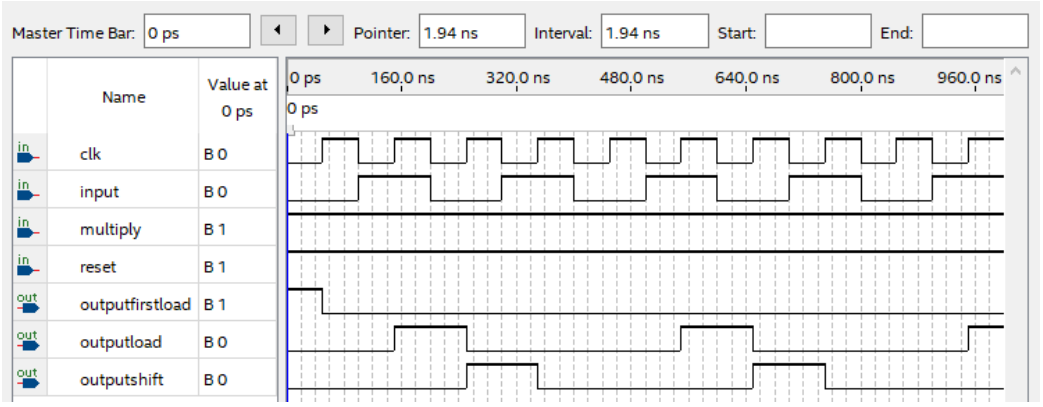
Εικόνα 87

4.4.2 Εξομοίωση

Για να δούμε αν λειτουργεί σωστά η μονάδα ελέγχου εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το αποτέλεσμα διαπιστώσαμε ότι λειτουργεί σωστά.

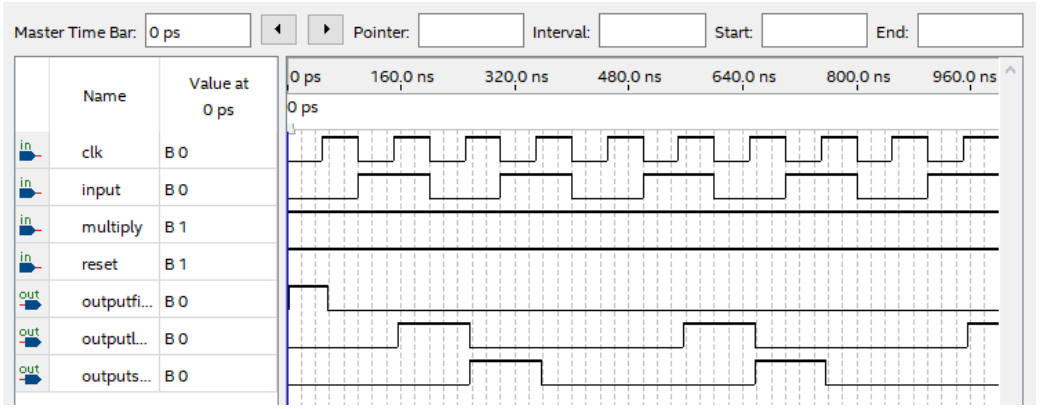
Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε control_unit_Waveform.

Λογική εξομοίωση μονάδας ελέγχου (control_unit _Waveform)



Εικόνα 88

Χρονική εξομοίωση μονάδας ελέγχου (control_unit_Waveform)



Εικόνα 89

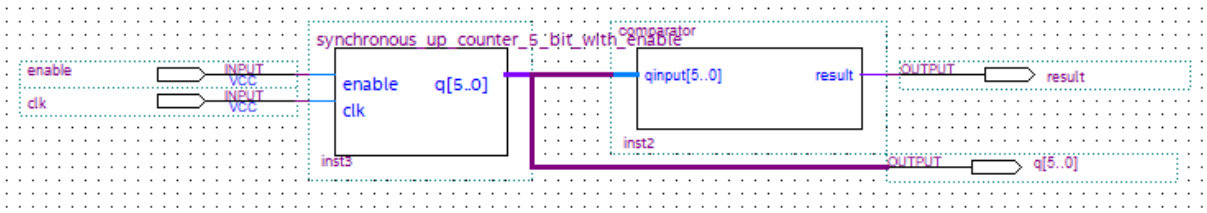
4.5 Μετρητής (Counter)

4.5.1 Υλοποίηση

Ο μετρητής όπως προαναφέρθηκε χρησιμοποιείται για να μετράει 32 κύκλους μηχανής ώστε όταν τους φτάσει να δώσει την εντολή στην υποδεέστερή μονάδα ελέγχου. Το σχηματικό αποτελείται από δύο στοιχεία τον σύγχρονο μετρητή 5 bit (synchronous up counter) και τον συγκριτή comparator. Επίσης έχει δυο εισόδους ήτοι την enable που είναι για την ενεργοποίηση και την clk που είναι για τους κύκλους μηχανής και 2 εξόδους ήτοι την result που είναι το αποτέλεσμα και πάει στην υποδεέστερη μονάδα ελέγχου και την q 5 bit που είναι απλά για τον έλεγχο του μετρητή. Για τις υλοποιήσεις του ασύγχρονου μετρητή και συγκριτή θα αναφερθούμε παρακάτω. [11]

Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime ονομάζεται counter.

Σχηματικό μετρητή (counter)



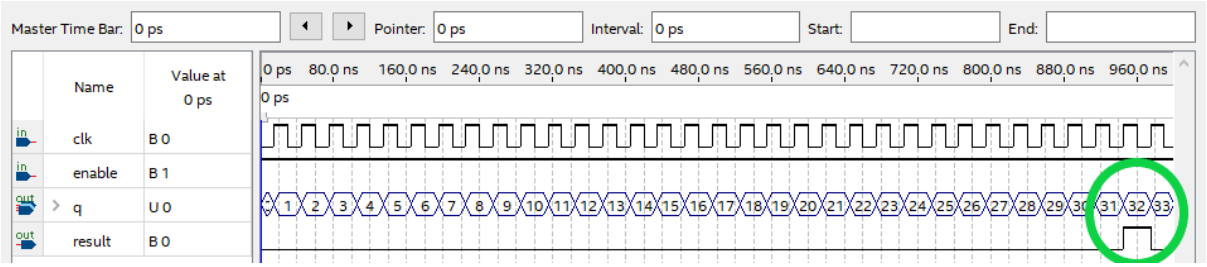
Εικόνα 90

4.5.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά η μονάδα ελέγχου εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το αποτέλεσμα διαπιστώσαμε ότι λειτουργεί σωστά.

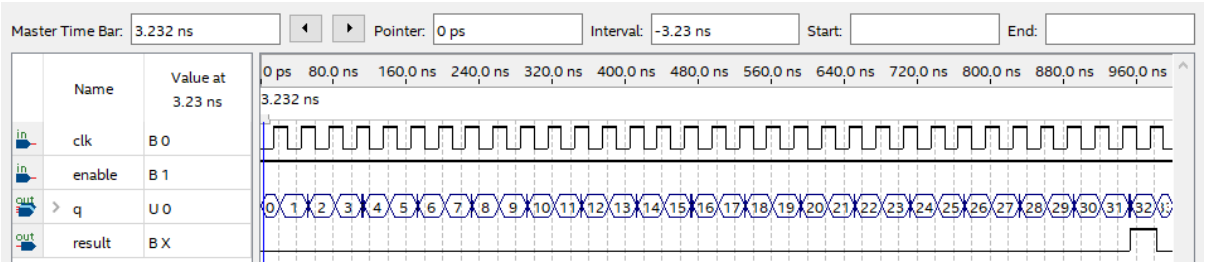
Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε counter_Waveform.

Λογική εξομοίωση του μετρητή (counter_Waveform)



Εικόνα 91

Χρονική εξομοίωση του μετρητή (counter_Waveform)



Εικόνα 92

4.6 Σύγχρονος μετρητής 5 μπιτ με ενεργοποίηση (Synchronousupcounter 5 bitwithenable)

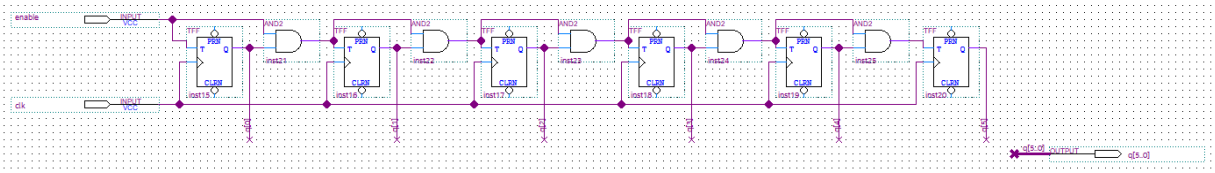
4.6.1 Υλοποίηση

Ο σύγχρονος μετρητής 5 bit όπως προαναφέρθηκε χρησιμοποιείται για να μετράει τους κύκλους μηχανής. Το σχηματικό του αποτελείται από 5 πύλες AND δυο εισόδων, 5 T flip-flops τα οποία τα πήραμε έτοιμα από την βιβλιοθήκη στο Quartus, μια είσοδο enable για

την ενεργοποίηση, μια clk για τους κύκλους ρολογιού και μια έξοδο την q 5 bit που μας εμφανίζει τους κύκλους που πέρασαν. [11],[14]

Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε synchronous_up_counter_5_bit_with_enable.

Σχηματικό σύγχρονου μετρητή 5 bit με ενεργοποίηση
(synchronous_up_counter_5_bit_with_enable)



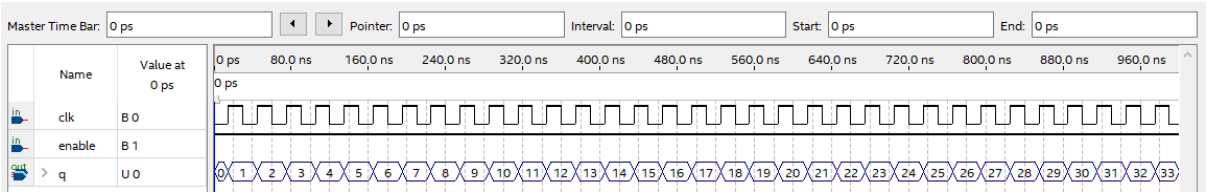
Εικόνα 93

4.6.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά ο σύγχρονος μετρητής 5 bit εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το αποτέλεσμα διαπιστώσαμε ότι λειτουργεί σωστά.

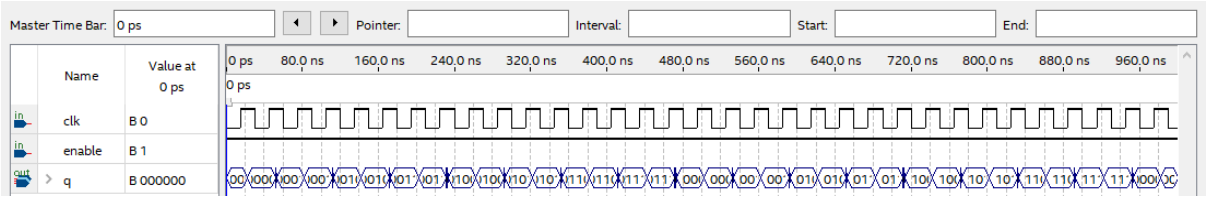
Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε synchronous_up_counter_5_bit_with_enable_Waveform.

Λογική εξομοίωση σύγχρονου μετρητή 5 bit με ενεργοποίηση
(synchronous_up_counter_5_bit_with_enable_Waveform)



Εικόνα 94

Χρονική εξομοίωση σύγχρονου μετρητή 5 bit με ενεργοποίηση
(synchronous_up_counter_5_bit_with_enable_Waveform)



Εικόνα 95

4.7 Συγκριτής (Comparator)

4.7.1 Υλοποίηση

Ο συγκριτής χρησιμοποιήθηκε με σκοπό την σύγκριση των τιμών που περνάνε από αυτόν και τον εντοπισμό της τιμής με αριθμό 32, όπου 32 είναι οι κύκλοι μηχανής που ψάχνουμε. Η υλοποίηση του συγκριτή έγινε με την χρήση της γλώσσας VHDL. Ο συγκριτής διαθέτει μια είσοδο qinput 5 bit πού είναι για τις εισερχόμενες τιμές που έρχονται από τον σύγχρονο μετρητή 5 bit και μια έξοδο result που ενημερώνει αν εμφανίστηκε η τιμή 32. [11]

Το αρχείο του προγράμματος VHDL στο πρόγραμμα Quartus Elite Prime το ονομάσαμε comparator.

Κώδικας VHDL του συγκριτή (comparator)

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY comparator IS
5         PORT
6         (
7
8                qinput    : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
9                result    : OUT STD_LOGIC
10
11          );
12
13  END comparator;
14
15  ARCHITECTURE behavioral OF comparator IS
16  BEGIN
17
18  process (qinput)
19
20
21   BEGIN
22
23
24
25    IF qinput = "100000" THEN
26
27      result <= '1';
28
29     ELSE
30
31      result <= '0';
32
33
34
35    END IF;
36
37  END process;
38
39
40
41  END behavioral;
```

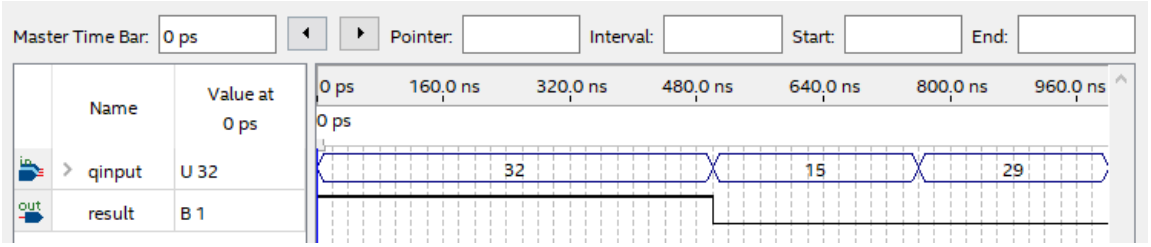
Εικόνα 96

4.7.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά ο συγκριτής εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το αποτέλεσμα διαπιστώσαμε ότι λειτουργεί σωστά.

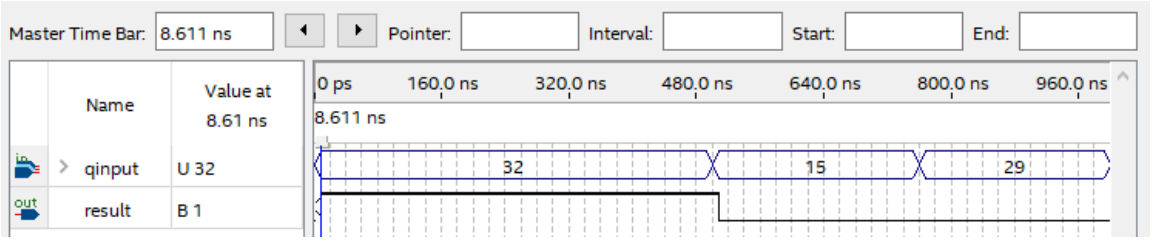
Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε comparator_Waveform.

<u>Λογική εξομοίωση του συγκριτή</u> (comparator_Waveform)



Εικόνα 97

<u>Χρονική εξομοίωση του συγκριτή</u> (comparator_Waveform)



Εικόνα 98

4.8 Υποδεέστερη μονάδα ελέγχου (Control)

4.8.1 Υλοποίηση

Η υποδεέστερη μονάδα ελέγχου λειτουργεί σαν συλλέκτης πληροφοριών και με βάση αυτών δίνει τις κατάλληλες εντολές στα άλλα στοιχεία. Η υλοποίηση της μονάδας ελέγχου έγινε με την χρήση της γλώσσας VHDL καθώς θα ήταν πολύ δύσκολο να την υλοποιήσουμε σχηματικά. Στην συγκεκριμένη έγινε η χρήση της μηχανής πεπερασμένων καταστάσεων (Finite State Machines) καθώς με αυτό τον τρόπο η πληροφορία που θα έρχεται θα οδηγείται στην σωστή κατάσταση μέχρι να βγει το τελικό αποτέλεσμα. Η μονάδα διαθέτει 5 εισόδους, την reset η οποία ενεργοποιεί την μονάδα όταν είναι λογικό 1, την clock που είναι για τους κύκλους του ρολογιού, την input πού είναι το πρώτο bit του 64 bit αριθμού που αναφέραμε και παραπάνω, την multiply είναι λογικό 1 όταν εκτελούμε τον πολλαπλασιασμό καθώς και την είσοδο counter η οποία γίνεται 1 όταν ολοκληρωθούν 32 κύκλοι. Από εξόδους έχουμε την outputfirstload , την outputload και την outputshift οι οποίες δρομολογούνται στο καταχωρητή ολίσθησης 64 bit. Η outputfirst είναι για να φορτώσει όλο τον 64 bit αριθμό του καταχωρητή , η outputload είναι για να φορτώσει τον καινούργιο αριθμό από την μονάδα ALU και η outshift που είναι για να ολισθήσει τον 64 bit αριθμό 1 bit δεξιά. [11],[15]

Το αρχείο του προγράμματος VHDL στο πρόγραμμα Quartus Elite Prime το ονομάσαμε control_vhdl.

Κώδικας VHDL της υποδεέστερης μονάδας ελέγχου (control_vhdl)

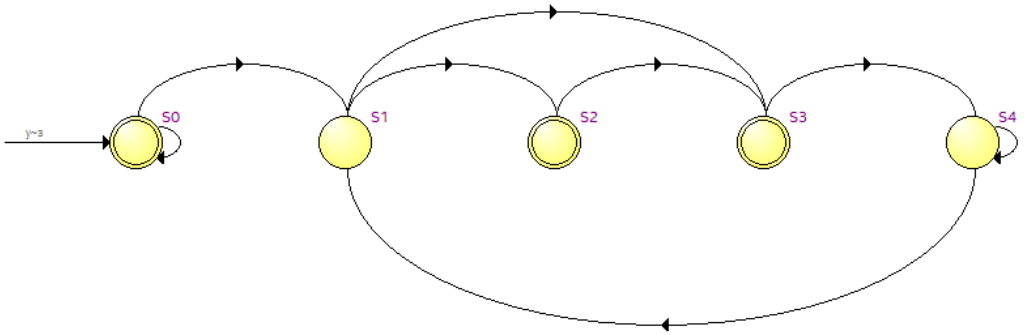
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity control_vhdl is
        PORT
        (
                reset           : IN STD_LOGIC;
                clock           : IN STD_LOGIC;
                input           : IN STD_LOGIC;
                multiply        : IN STD_LOGIC;
                counter         : IN STD_LOGIC;
                outputfirstload: OUT STD_LOGIC;
                outputload      : OUT STD_LOGIC;
                outputshift     : OUT STD_LOGIC
         );
end control_vhdl;

architecture FSM of control_vhdl is

TYPE state IS (S0, S1, S2, S3, S4);
SIGNAL y: state;

begin
 PROCESS (clock, reset)

BEGIN
   IF reset = '0' THEN
    y <= S0;
   ELSIF (clock = '1' AND clock' event) THEN


    CASE y IS

      WHEN S0 =>
        IF multiply = '1' THEN
                    y <= S1;
                ELSE
                    y <= S0;
                END IF;

      WHEN S1 =>
        IF input = '1' THEN
                    y <= S2;
                ELSE
                    y <= S3;
                 END IF;

      WHEN S2 =>
        IF multiply = '1' THEN
                    y <= S3;
                 ELSE
                    y <= S3;
                 END IF;

      WHEN S3 =>
        IF multiply = '1' THEN
                    y <= S4;
                ELSE
                    y <= S4;
                 END IF;

      WHEN S4 =>
        IF counter = '1' THEN
                    y <= S4;
                ELSE
                    y <= S1;
                END IF;


END CASE;
end if;
END PROCESS;


outputfirstload <= '1' WHEN y = S0 ELSE '0';
outputload <= '1' WHEN y = S2 ELSE '0';
outputshift <= '1' WHEN y = S3 ELSE '0';

END FSM;
```

Εικόνα 99

4.8.2 Έλεγχος λειτουργικότητας

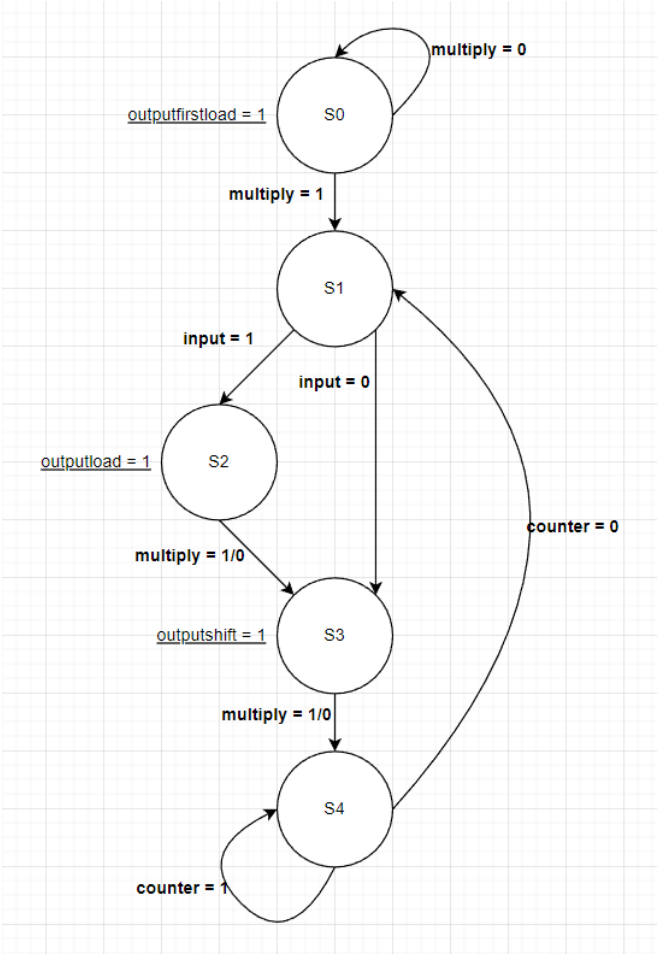
Για να δούμε αν λειτουργεί σωστά η υποδεέστερη μονάδα ελέγχου εκτελέσαμε την εντολή State Machine Viewer που βρίσκεται στα Tools, Netlist Viewers και αφού την συγκρίναμε με το σχηματικό που δημιουργήσαμε διαπιστώσαμε ότι λειτουργεί σωστά.

State Machine Viewer της υποδεέστερης μονάδας ελέγχου



Εικόνα 100

Σχηματικό της υποδεέστερης μονάδας ελέγχου



Εικόνα 101

5.Κεφάλαιο 5

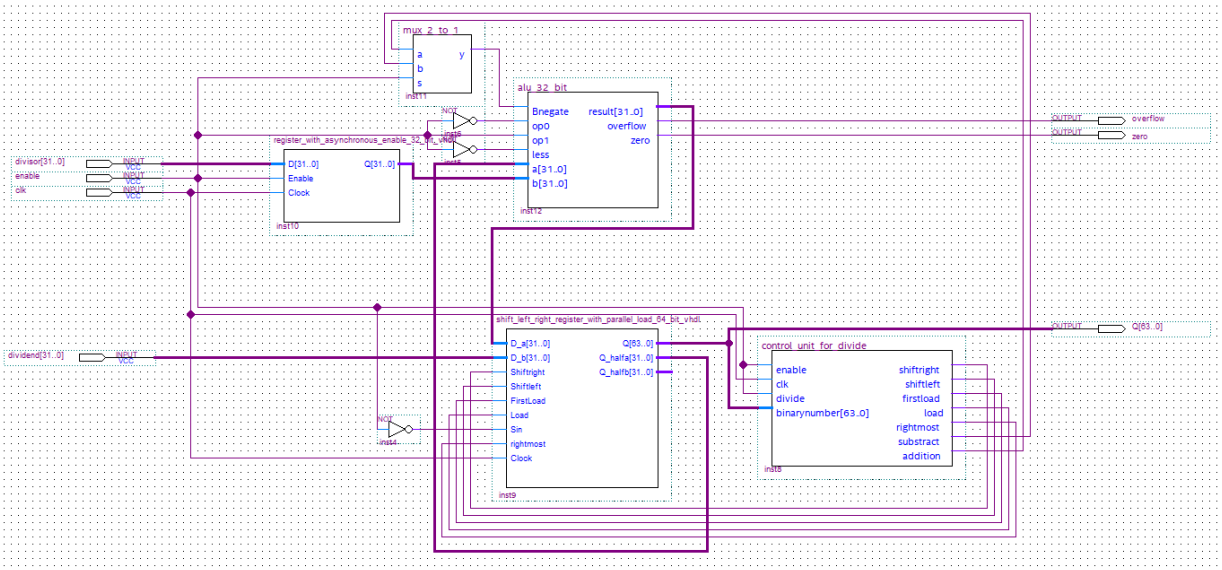
5.1 Διαίρεση (Divide)

5.1.1 Υλοποίηση

Ο μηχανισμός της διαίρεσης γίνεται με την βοήθεια της αριθμητικής και λογικής μονάδας 32 bit καθώς και με την χρήση κάποιον άλλων στοιχείων. Η διαίρεση γίνεται με μια σειρά πράξεων πρόσθεσης, αφαίρεσης και αριστερής ή δεξιάς μετατόπισης .Συγκεκριμένα ο μηχανισμός πραγματοποιήθηκε με βάση το διάγραμμα ροής που απεικονίζεται παρακάτω.Το σχηματικό που υλοποιήσαμε αποτελείται από τρείς πύλες NOT, μια 32 bit αριθμητική και λογική μονάδα, έναν καταχωρητή 32 bit με ασύγχρονη ενεργοποίηση, έναν καταχωρητή ολίσθησης 64 bit, μια μονάδα ελέγχου, 4 εισόδους και 3 εξόδους. Από εισόδους έχουμε την clock που είναι για τον χρόνο, την enable που είναι την ενεργοποίηση, την dividend 32 bit που είναι ο διαιρετέος και την divisor 32 bit που είναι ο διαιρέτης. Από εξόδους έχουμε την Q 64 bit που είναι για το αποτέλεσμα, την overflow και την zero που έχουν την ίδια χρήση με αυτή του πολλαπλασιασμού. Για την κατασκευή των στοιχείων, του καταχωρητή 64 bit και της μονάδας ελέγχου θα αναφερθούμε στη συνέχεια καθώς δεν είναι ίδια με αυτά τουπολλαπλασιασμού. [11],[14],[15]

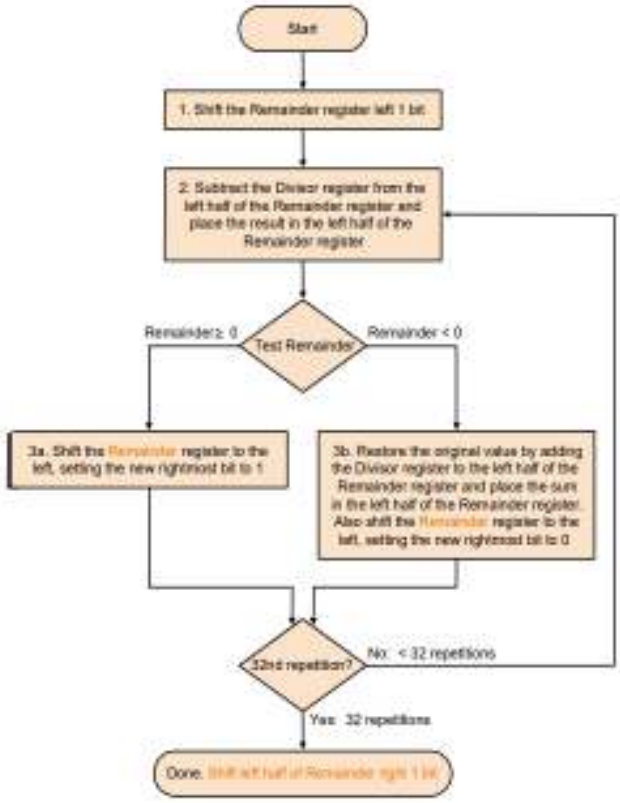
Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε divide.

<u>Σχηματικό διαίρεσης</u> (divide)



Εικόνα 102

<u>Διάγραμμα ροής</u>(Flowchart)



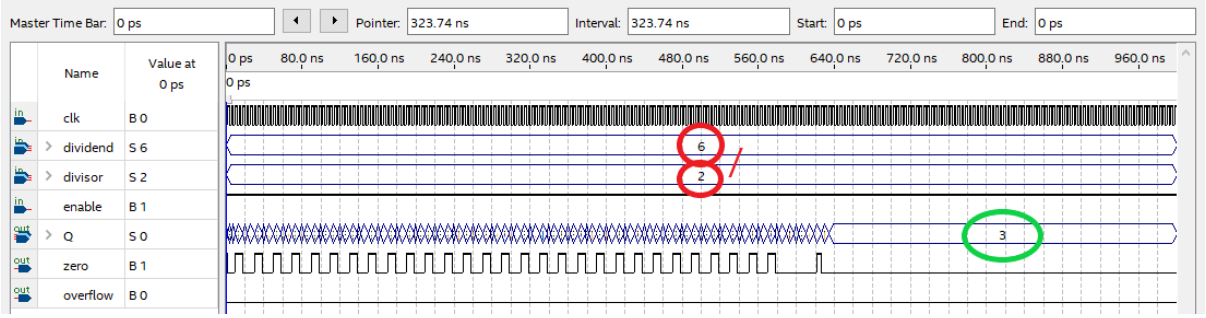
Εικόνα 103

5.1.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά ο μηχανισμός της διαίρεσης εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το αποτέλεσμα είδαμε ότι λειτουργεί σωστά.

Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε divide_Waveform.

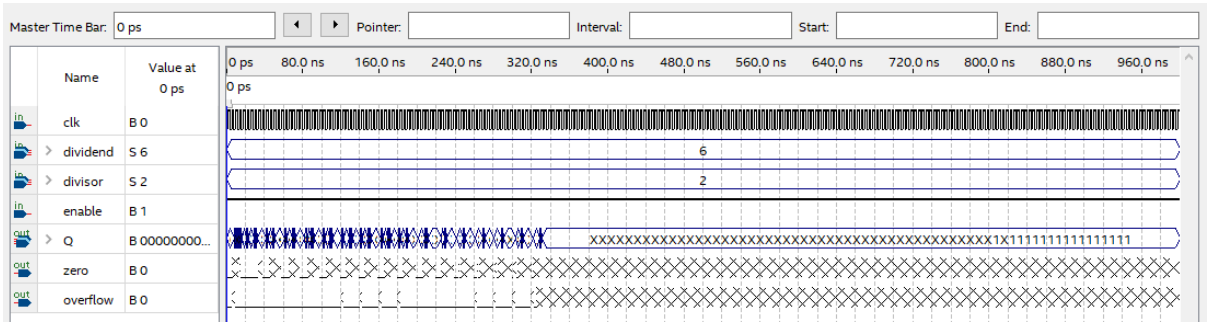
<u>Λογικές εξομοιώσεις της διαίρεσης</u> (divide_Waveform)



Εικόνα 104

Εικόνα 105

<u>Χρονική εξομοίωση της διαίρεσης</u> (divide_Waveform)



Εικόνα 106

5.2 Καταχωρητής ολίσθησης 64 μπιτ με παράλληλη φόρτιση για την διαίρεση (64 bitshiftregisterwithparallelloadfordivide)

5.2.1 Υλοποίηση

Η δημιουργία του καταχωρητή ολίσθησης 64 bit με παράλληλη φόρτιση έγινε με την χρήση της γλώσσας VHDL καθώς το κύκλωμα σχηματικά θα ήταν πολύ δύσκολο έως αδύνατο να υλοποιηθεί. Ο καταχωρητής αποτελείται από δυο εισόδους την D_a που είναι ο διαιρέτης όπου έρχεται από την μονάδα ALU και μπαίνει στα αριστερά 32 bit του 64 bit αριθμού και την D_b που έρχεται από τον διαιρετέο και μπαίνει στα δεξιά 32 bit του αριθμού. Επιπλέον, διαθέτει την είσοδο enable που είναι για την ενεργοποίηση της ολίσθησης, τη FirstLoad που είναι για την φόρτωση όλου του αριθμού, την είσοδο Load που είναι για την φόρτωση του μισού αριστερού αριθμού, την είσοδο Sin που είναι ο αριθμός του τελευταίου ψηφίου που θα ολισθηθεί δεξιά, την rightmost του τελευταίου ψηφίου που θα ολισθηθεί αριστερά, την είσοδο Clock που είναι για το ρολόι και τις εισόδους Shiftright, Shiftleft που είναι για την ενεργοποίηση της δεξιάς ολίσθησης του μισού αριστερού αριθμού ή της αριστερής ολίσθησης αντίστοιχα. Από εξόδους έχουμε την Q 64 bit που είναι για το αποτέλεσμα , την Q_halfa 32 bit η οποία επιστρέφει στην ALU και την Q_halfb 32 bit οι οποίες αν ενωθούν μας κάνουν την Q 64 bit. Η λειτουργία του καταχωρητή είναι να κρατάει αποθηκευμένο τον αριθμό της εισόδου,με σκοπό να τον

ολισθήσει δεξιά ή αριστερά ή να του φορτώσει κάποιο καινούργιο αποτέλεσμα από την ALU. [11],[14]

Το αρχείο του προγράμματος VHDL στο πρόγραμμα Quartus Elite Prime το ονομάσαμε shift_left_right_register_with_parallel_load_64_bit_vhdl.

<u>Κώδικας VHDL του καταχωρητή ολίσθησης 64 bit με παράλληλη φόρτιση για την διαίρεση</u> (shift_left_right_register_with_parallel_load_64_bit_vhdl)

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

ENTITY shift_left_right_register_with_parallel_load_64_bit_vhdl IS
        PORT
        (

                D_a         : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
                D_b         : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
                Shiftright  : IN STD_LOGIC;
                Shiftleft   : IN STD_LOGIC;
                FirstLoad   : IN STD_LOGIC;
                Load        : IN STD_LOGIC;
                Sin         : IN STD_LOGIC;
                rightmost   : IN STD_LOGIC;
                Clock       : IN STD_LOGIC;
                Q           : OUT STD_LOGIC_VECTOR(63 DOWNTO 0);
                Q_halfa     : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
                Q_halfb     : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)

         );
END shift_left_right_register_with_parallel_load_64_bit_vhdl;

ARCHITECTURE behavioral OF shift_left_right_register_with_parallel_load_64_bit_vhdl IS
        SIGNAL Qt: STD_LOGIC_VECTOR(63 DOWNTO 0);
BEGIN


PROCESS (Clock)

 variable D : STD_LOGIC_VECTOR(63 DOWNTO 0);

 BEGIN

 D := D_a & D_b;

  IF rising_edge(Clock) THEN
   IF FirstLoad = '1' THEN
    Qt <= D;

   ELSIF Load = '1' THEN
     Qt(63 downto 32) <= D_a;

   ELSIF Shiftright = '1' THEN
     FOR i IN 32 TO 62 LOOP
      Qt(i) <= Qt(i+1);
     END LOOP;
    Qt(63) <= Sin;

   ELSIF Shiftleft = '1' THEN
     FOR j IN 1 TO 63 LOOP
      Qt(j) <= Qt(j-1);
     END LOOP;
    Qt(0) <= rightmost;

  END IF;
 END IF;
END PROCESS;
 Q <= Qt;

 Q_halfa <= Qt(63 downto 32);
 Q_halfb <= Qt(31 downto 0);


END behavioral;
```

Εικόνα 107

5.2.2 Υλοποίηση

Για να δούμε αν λειτουργεί σωστά ο καταχωρητής ολίσθησης 64 bit με παράλληλη φόρτιση εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το αποτέλεσμα διαπιστώσαμε ότι λειτουργεί σωστά.

Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε shift_left_right_register_with_parallel_load_64_bit_vhdl _Waveform.

Λογικές εξομοιώσεις του καταχωρητή ολίσθηση 64 bit με παράλληλη φόρτιση για την διαίρεση(shift_left_right_register_with_parallel_load_64_bit_vhdl_Waveform)

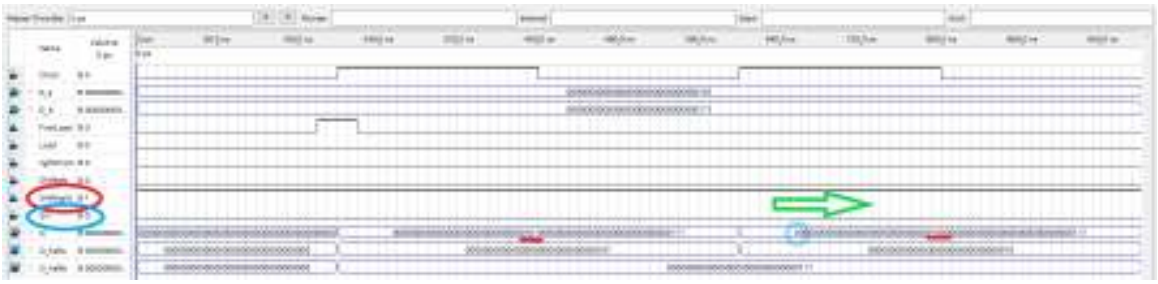
Ολίσθηση αριστερά με rightmost = 0

Εικόνα 108

Ολίσθηση αριστερά με rightmost = 1

Εικόνα 109

Ολίσθηση δεξιά (του μισού αριθμού 64 bit) με Sin = 0

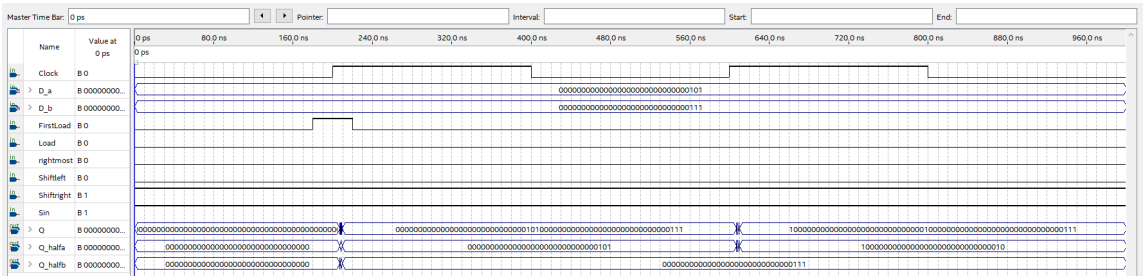


Εικόνα 110

Ολίσθηση δεξιά (του μισού αριθμού 64 bit) με Sin = 1

Εικόνα 111

Χρονική εξομοίωση του καταχωρητή ολίσθηση 64 bit με παράλληλη φόρτιση για την διαίρεση(shift_left_right_register_with_parallel_load_64_bit_vhdl_Waveform)



Εικόνα 112

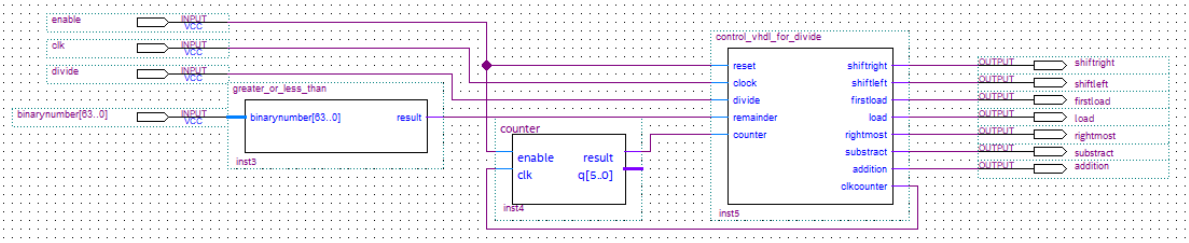
5.3Μονάδα ελέγχου για την διαίρεση (Controlunitfordivide)

5.3.1 Υλοποίηση

Η μονάδα ελέγχου όπως και στο πολλαπλασιασμό έτσι και στην διαίρεση χρησιμοποιείται για να καθοδηγεί σωστά τις λειτουργίες των στοιχείων του κυκλώματος. Η μονάδα ελέγχου στο σχηματικό της αποτελείται από ένα μετρητή counter που είναι υπεύθυνος για να μετράει 32 κύκλους μηχανής, από ένα συγκριτή για να εντοπίζει αν η τιμή του 64 bit αριθμού είναι μεγαλύτερη μικρότερη ή ίση με το 0 και μια ακόμα υποδεέστερη μονάδα ελέγχου. Επιπρόσθετα, το σχηματικό της διαθέτει 4 εισόδους, μια clk για το κύκλους μηχανής, μια enable για την ενεργοποίηση των συστημάτων, μια binarynumber 64 bit που είναι για τον 64 bit αριθμό και την είσοδο divide που είναι για την ενεργοποίηση των καταστάσεων στην υποδεέστερη μονάδα ελέγχου. Οι έξοδοι που διαθέτει είναι η shiftright, η shiftleft, η firstload, η load, η rightmost, η subtract, η addition, οι οποίες προορίζονται για τον καταχωρητή ολίσθησης και οι λειτουργίες τους είναι για αυτές που αναφέραμε παραπάνω. Για τις υλοποιήσεις του συγκριτή και της υποδεέστερης μονάδας ελέγχου θα μιλήσουμε παρακάτω. [11]

Το αρχείο του σχηματικού στο πρόγραμμα Quartus Elite Prime το ονομάσαμε control_unit_for_divide.

Σχηματικό μονάδας ελέγχου για την διαίρεση (control_unit_for_divide)



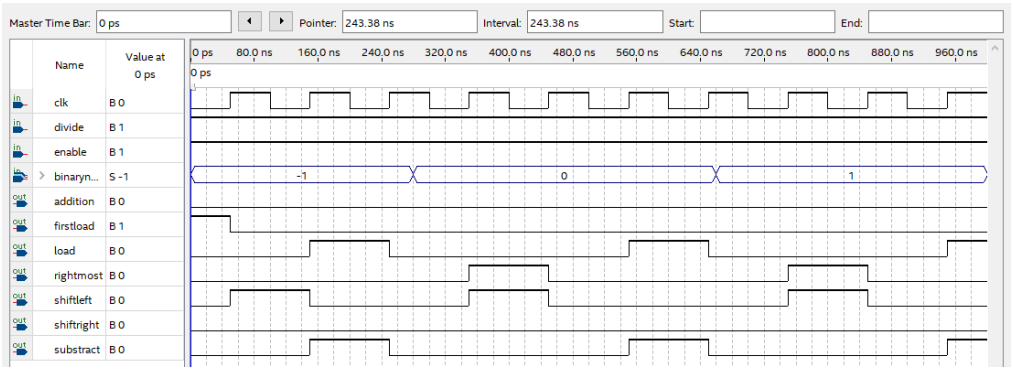
Εικόνα 113

5.3.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά η μονάδα ελέγχου εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το αποτέλεσμα διαπιστώσαμε ότι λειτουργεί σωστά.

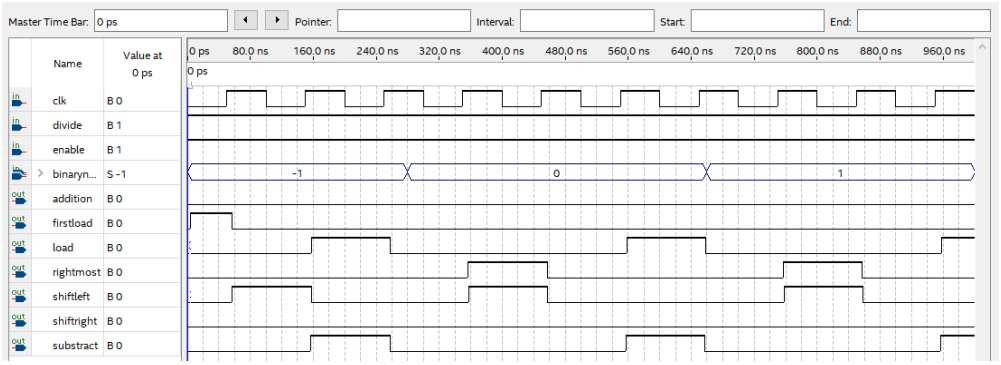
Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε control_unit_for_divide_Waveform.

Λογική εξομοίωση μονάδας ελέγχου για την διαίρεση (control_unit_for_divide_Waveform)



Εικόνα 114

Χρονική εξομοίωση μονάδας ελέγχου για την διαίρεση (control_unit_for_divide_Waveform)



Εικόνα 115

5.4 Συγκριτής αριθμού μεγαλύτερου, ίσου ή μικρότερου του 0 (Greater, sameorlessthanzerounit)

5.4.1 Υλοποίηση

Ο συγκριτής χρησιμοποιείται ώστε να εντοπίζει αν ο 64 bit αριθμός που εισέρχεται στην είσοδο binarynumber είναι μεγαλύτερος ή ίσος ή μικρότερος του μηδενός. Όταν ο αριθμός είναι μεγαλύτερος ή ίσος η έξοδος result που βγάζει γίνεται 1 ενώ αν είναι μικρότερος βγάζει 0. Η υλοποίηση του στοιχείου έγινε με την γλώσσα περιγραφής υλικού VHDL και το αρχείο του στο πρόγραμμα Quartus Elite Prime το ονομάσαμε greater_or_less_than.[11]

Κώδικας προγράμματος συγκριτή αριθμού μεγαλύτερου, ίσου η μικρότερου του μηδέν (greater_or_less_than)

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

ENTITY greater_or_less_than IS
        PORT
        (

                binarynumber    : IN STD_LOGIC_VECTOR(63 DOWNTO 0);
                result          : OUT STD_LOGIC

         );
END greater_or_less_than;
ARCHITECTURE behavioral OF greater_or_less_than IS
BEGIN
process (binarynumber)


  BEGIN


  IF binarynumber = "0000000000000000000000000000000000000000000000000000000000000000" THEN
    result <= '1';
  ELSIF binarynumber(63) = '0' THEN
    result <= '1';
  ELSE
    result <= '0';
  END IF;
 END process;

END behavioral;
```

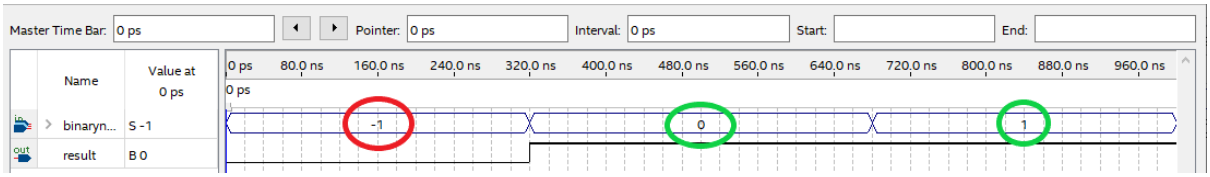
Εικόνα 116

5.4.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά η μονάδα ελέγχου εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με το αποτέλεσμα διαπιστώσαμε ότι λειτουργεί σωστά.

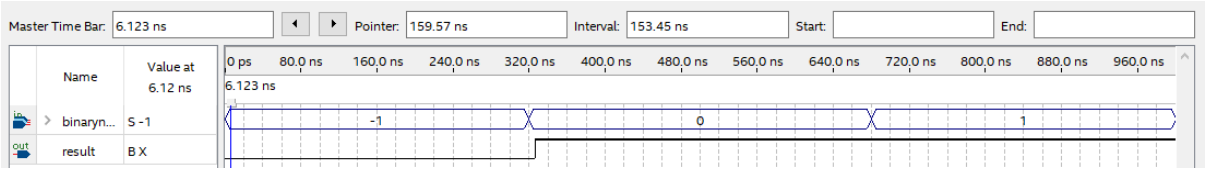
Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε greater_or_less_than_Waveform.

Λογική εξομοίωση του συγκριτή αριθμού μεγαλύτερου, ίσου ή μικρότερου του μηδέν (greater_or_less_than _Waveform)



Εικόνα 117

<u>Χρονική εξομοίωση του συγκριτή αριθμού μεγαλύτερου, ίσου ή μικρότερου του μηδέν</u>
(greater_or_less_than _Waveform)



Εικόνα 118

5.5 Υποδεέστερη μονάδα ελέγχου για την διαίρεση (Controlunitfordivide)

5.5.1 Υλοποίηση

Η υποδεέστερη μονάδα ελέγχου λειτουργεί σαν συλλέκτης πληροφοριών και με βάση αυτών δίνει τις κατάλληλες εντολές στα άλλα στοιχεία. Η υλοποίηση της μονάδας ελέγχου έγινε με την χρήση της γλώσσας VHDL καθώς θα ήταν πολύ δύσκολο να αναπαρασταθεί σχηματικά. Στην συγκεκριμένη περίπτωση έγινε η χρήση της μηχανής πεπερασμένων καταστάσεων (Finite State Machines) καθώς με αυτό τον τρόπο η πληροφορία που θα έρχεται θα οδηγείται στην σωστή κατάσταση μέχρι να βγει το τελικό αποτέλεσμα. Η μονάδα διαθέτει 5 εισόδους, την reset η οποία ενεργοποιεί την μονάδα όταν είναι λογικό 1, την clock που είναι για τους κύκλους του ρολογιού, την remainder η οποία συνδέεται με τον συγκριτή και γίνεται λογικό 1 όταν ο αριθμός είναι μεγαλύτερος ή ίσος του μηδενός και το αντίθετο, την divide που όταν είναι λογικό 1εκτελεί την διαίρεση και η είσοδος counter η οποία γίνεται 1 όταν ολοκληρωθούν 32 κύκλοι. Από εξόδους διαθέτει την shiftright, την shiftleft, την firstload, την load, την rightmost, την subtract, την addition, για την χρησιμότητα των οποίων αναφερθήκαμε παραπάνω και τέλος έχουμε την έξοδο clkcounter η οποία συνδέεται με τον μετρητή ώστε να μετρήσει τους 32 κύκλους και να σταματήσει τη λειτουργία της διαίρεσης. [11],[15]

Το αρχείο του προγράμματος VHDL στο πρόγραμμα Quartus Elite Prime το ονομάσαμε control_vhdl_for_divide.

Κώδικας VHDL της υποδεέστερης μονάδας ελέγχου για την διαίρεση(control_vhdl_for_divide)

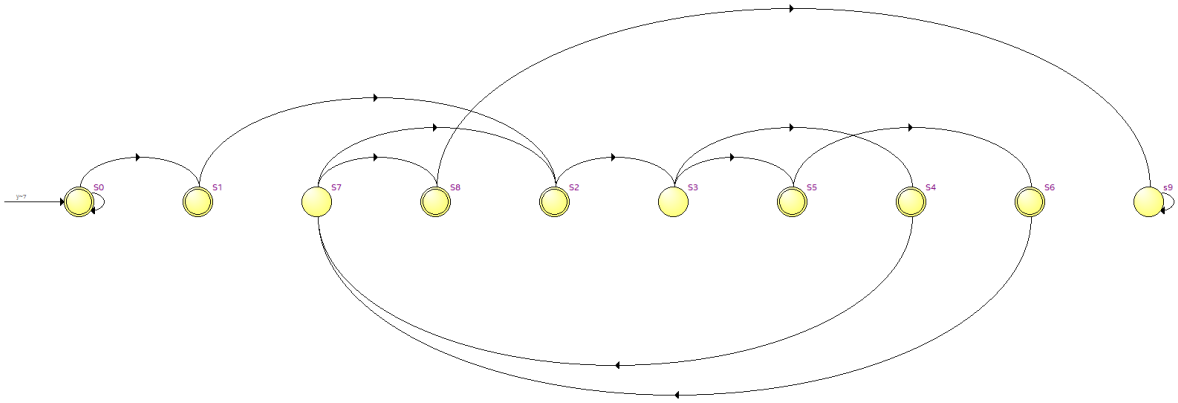
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity control_vhdl_for_divide is
        PORT
        (
                reset           : IN STD_LOGIC;
                clock           : IN STD_LOGIC;
                divide          : IN STD_LOGIC;
                remainder       : IN STD_LOGIC;
                counter         : IN STD_LOGIC;
                shiftright      : OUT STD_LOGIC;
                shiftleft       : OUT STD_LOGIC;
                firstload       : OUT STD_LOGIC;
                load            : OUT STD_LOGIC;
                rightmost       : OUT STD_LOGIC;
                substract       : OUT STD_LOGIC;
                addition        : OUT STD_LOGIC;
                clkcounter      : OUT STD_LOGIC
         );
end control_vhdl_for_divide;

architecture FSM of control_vhdl_for_divide is

TYPE state IS (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9);
SIGNAL y: state;

begin
 PROCESS (clock, reset)

BEGIN
   IF reset = '0' THEN
    y <= S0;
   ELSIF (clock = '1' AND clock' event) THEN


    CASE y IS

      WHEN S0 =>
        IF divide = '1' THEN
                y <= S1;
              ELSE
                y <= S0;
              END IF;

      WHEN S1 =>
        IF divide = '1' THEN
                y <= S2;
              ELSE
                y <= S2;
               END IF;

      WHEN S2 =>
        IF divide = '1' THEN
                y <= S3;
               ELSE
                y <= S3;
               END IF;

      WHEN S3 =>
        IF remainder = '1' THEN
                y <= S4;
              ELSE
                y <= S5;
               END IF;

      WHEN S4 =>
        IF divide = '1' THEN
                y <= S7;
              ELSE
                y <= S7;
              END IF;

      WHEN S5 =>
        IF divide = '1' THEN
                y <= S6;
              ELSE
                y <= S6;
               END IF;

      WHEN S6 =>
        IF divide = '1' THEN
                y <= S7;
               ELSE
                y <= S7;
               END IF;

      WHEN S7 =>
        IF counter = '1' THEN
                y <= S8;
              ELSE
                y <= S2;
               END IF;

      WHEN S8 =>
        IF divide = '1' THEN
                y <= S9;
              ELSE
                y <= S9;
              END IF;

      WHEN S9 =>
        IF divide = '1' THEN
                y <= S9;
              ELSE
                y <= S9;
              END IF;


END CASE;
end if;
 END PROCESS;


firstload  <= '1' WHEN y = S0 ELSE '0';
load       <= '1' WHEN y = S2 or y = S5 ELSE '0';

shiftleft  <= '1' WHEN y = S1 or y = S4 or y = S6 ELSE '0';
shiftright <= '1' WHEN y = S8 ELSE '0';

substract  <= '1' WHEN y = S2 ELSE '0';
addition   <= '1' WHEN y = S5 ELSE '0';

rightmost  <= '1' WHEN y = S4 ELSE '0';

clkcounter <= '1' WHEN y = S4 or y = S6 ELSE '0';

 END FSM;
```

Εικόνα 119

5.5.2 Έλεγχος λειτουργικότητας

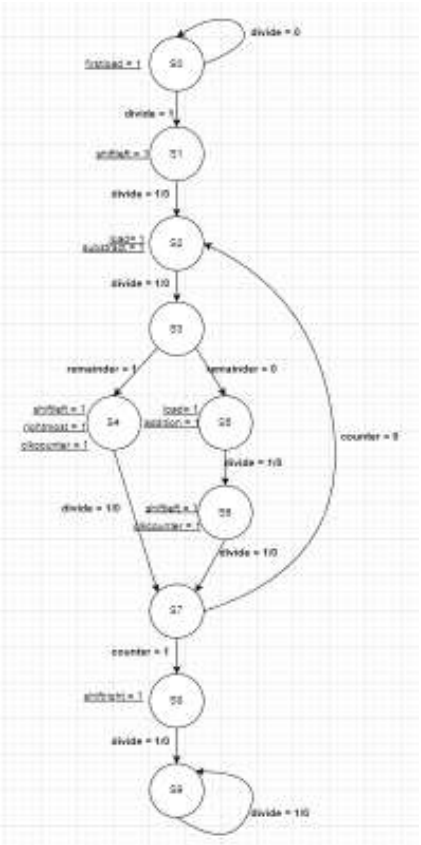
Για να δούμε αν λειτουργεί σωστά η υποδεέστερη μονάδα ελέγχου για την διαίρεση εκτελέσαμε την εντολή State Machine Viewer που βρίσκεται στα Tools, Netlist Viewers και αφού την συγκρίναμε με το σχηματικό που δημιουργήσαμε διαπιστώσαμε ότι λειτουργεί σωστά.

State Machine Viewer της υποδεέστερης μονάδας ελέγχου



Εικόνα 120

Σχηματικό της υποδεέστερης μονάδας ελέγχου για την διαίρεση



Εικόνα 121

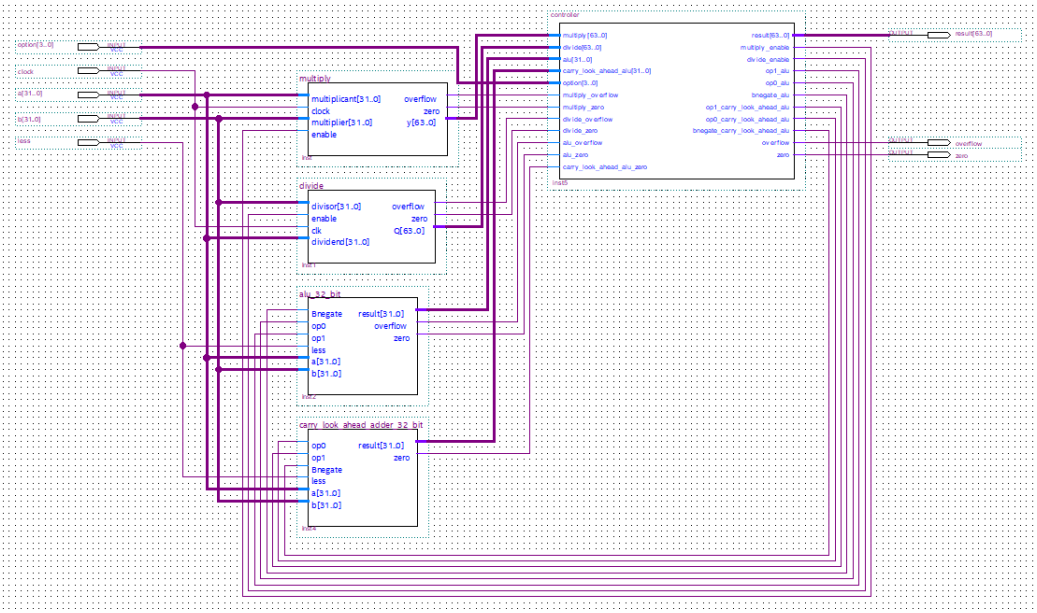
6.Κεφάλαιο 6

6.1 Συνδυασμός όλων των μονάδων σε μια (Combineallunitsinone)

6.1.1 Υλοποίηση

Σε αυτό το κεφάλαιο θα αναλύσουμε το σχηματικό του συνδυασμού όλων των μονάδων σε μια. Το σχηματικό αποτελείται από όλες τις μονάδες που δημιουργήσαμε παραπάνω δηλαδή από τον πολλαπλασιασμό, την διαίρεση, την απλή αριθμητική και λογική μονάδα και την μονάδα με την γεννήτρια πρόβλεψης κρατουμένου. Για να μπορέσουμε να συνδυάσουμε όλες τις μονάδες μεταξύ τους χρειάστηκε ένα ακόμη στοιχείο ονόματι controller το οποίο θα ελέγχει τις λειτουργίες των μονάδων. Το σχηματικό περιλαμβάνει επίσης 5 εισόδους, την option 4 bit που καθορίζει ποια πράξη θέλουμε να εκτελέσουμε, την clock που είναι για τους κύκλους μηχανής, την less που είναι πάντα λογικό 1 και τις εισόδους a,b 32 bit που είναι για τους αριθμούς που θέλουμε να επεξεργαστούμε. Από εξόδους όπως όλες οι μονάδες έχει την result που είναι το αποτέλεσμα , την overflow που είναι για την υπερχείλιση και την zero που μας δείχνει αν το αποτέλεσμα είναι μηδενικό. Η υλοποίηση του στοιχείου έγινε με την γλώσσα περιγραφής υλικού VHDL και το αρχείο του στο πρόγραμμα Quartus Elite Prime το ονομάσαμε compined.

Σχηματικό του συνδυασμού όλων των μονάδων σε μια (compined)



Εικόνα 122

6.1.2 Εξομοιώσεις

Για να δούμε αν λειτουργεί σωστά η συνδυαστική μονάδα εκτελέσαμε χρονική, λογική εξομοίωση και σύμφωνα με τον πίνακα λειτουργειών που έχουμε παρακάτω διαπιστώσαμε ότι λειτουργεί σωστά. Παρακάτω θα αναδείξουμε μέσω εικόνων όλες τις λειτουργίες της συνδυαστικής μονάδας.

Το αρχείο των εξομοιώσεων στο πρόγραμμα Quartus Elite Prime το ονομάσαμε compined_Waveform.

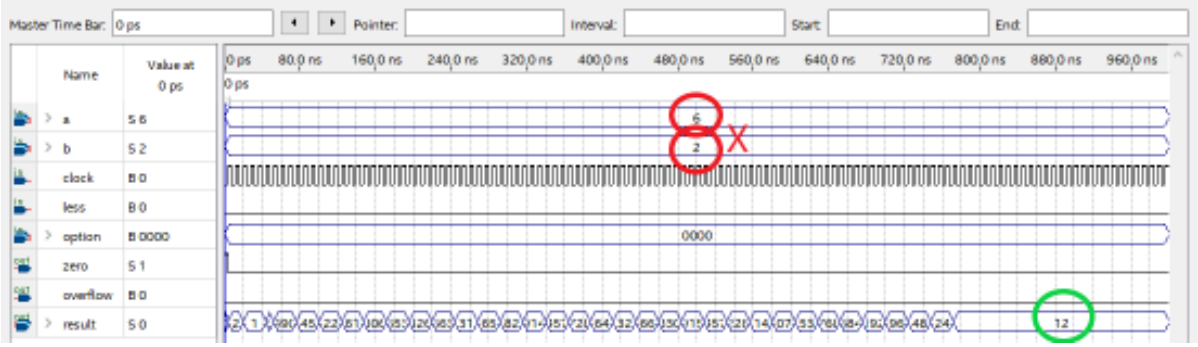
Πίνακας λειτουργιών της συνδυαστικής μονάδας

compined				
multiply	0	0	0	0
divide	0	0	0	1
addition (alu)	0	0	1	0
subtract (alu)	0	0	1	1
and (alu)	0	1	0	0
or (alu)	0	1	0	1
subtract (look_ahead)	0	1	1	0
addition (look_ahead)	0	1	1	1
less (alu)	1	0	0	0

Πίνακας 13

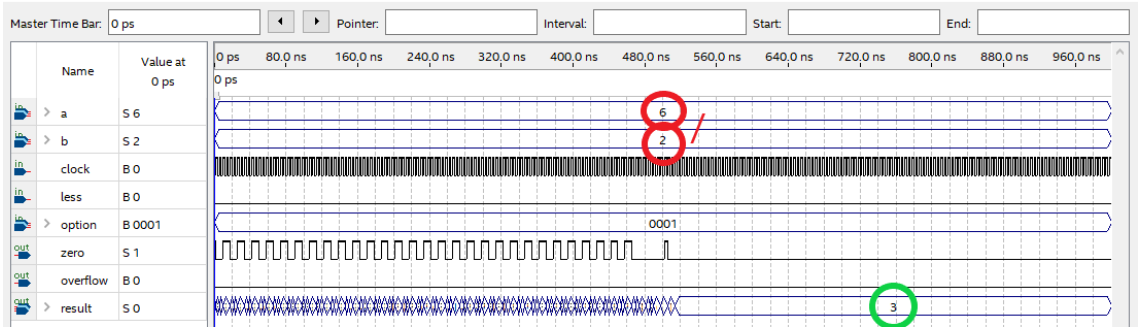
Λογικές εξομοιώσεις της συνδυαστικής μονάδας (compined_Waveform)

Πολλαπλασιασμός (0000)



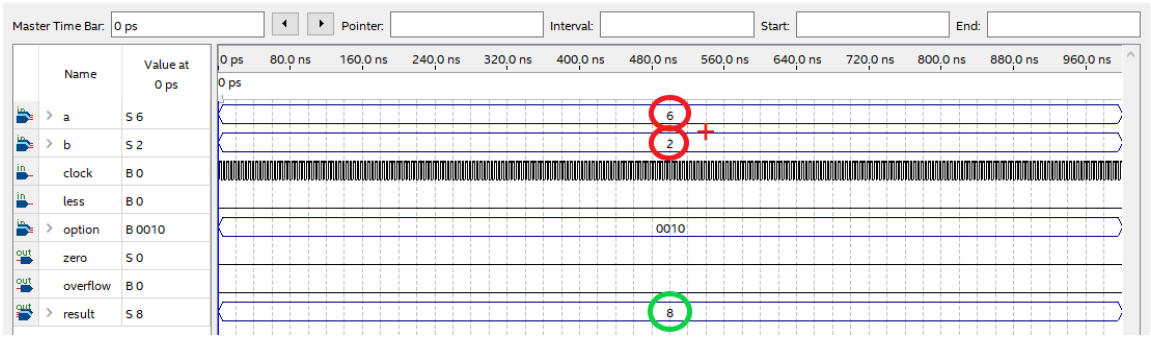
Εικόνα 123

Διαίρεση (0001)



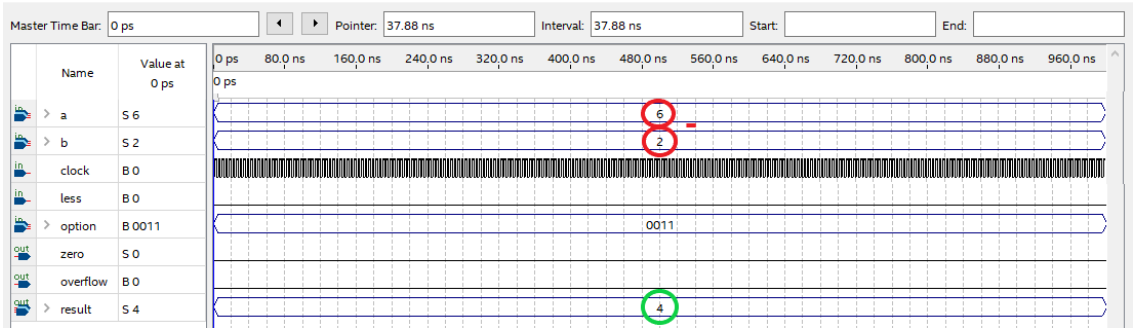
Εικόνα 124

Πρόσθεση με την απλή μονάδα ALU (0010)



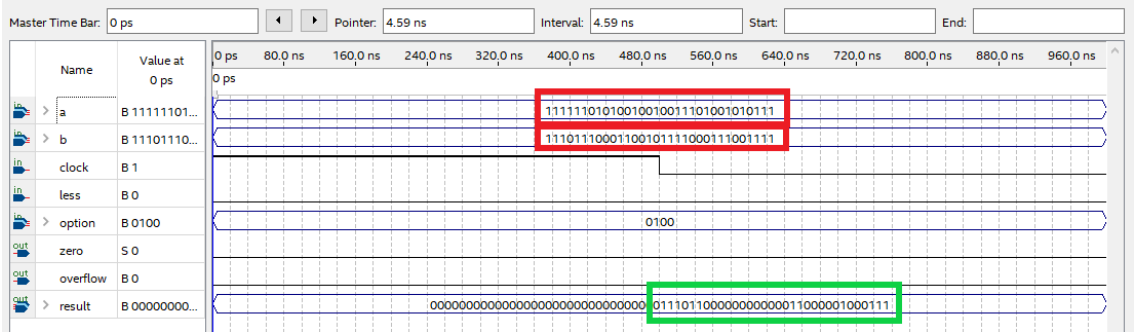
Εικόνα 125

Αφαίρεση με την απλή μονάδα ALU (0011)



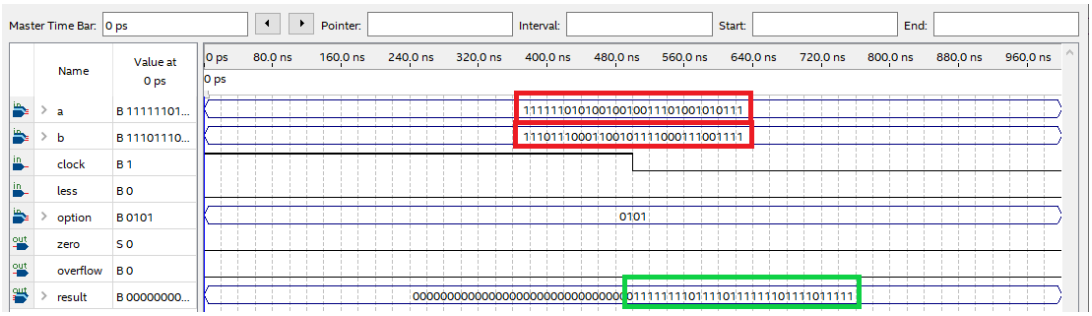
Εικόνα 126

Λογική πράξη AND με την απλή μονάδα ALU (0100)



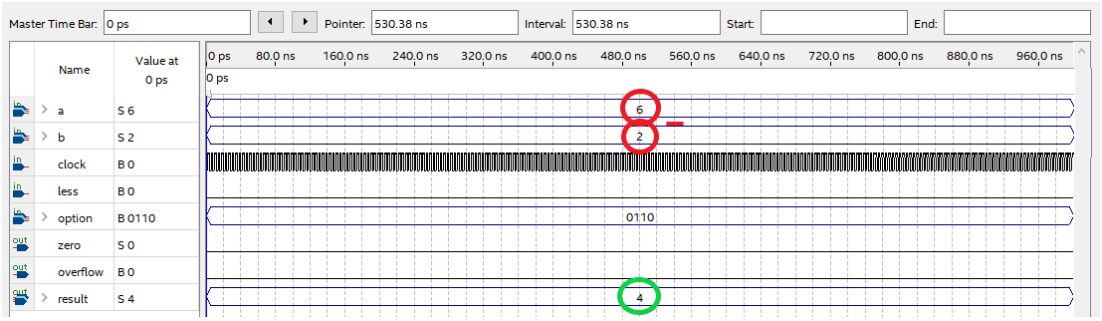
Εικόνα 127

Λογική πράξη OR με την απλή μονάδα ALU (0101)



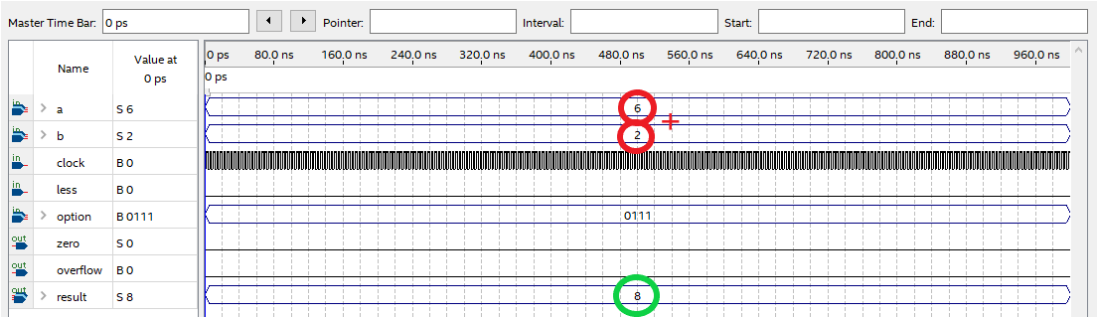
Εικόνα 128

Αφαίρεση με την μονάδα ALU με την γεννήτρια πρόβλεψης κρατουμένου(0110)



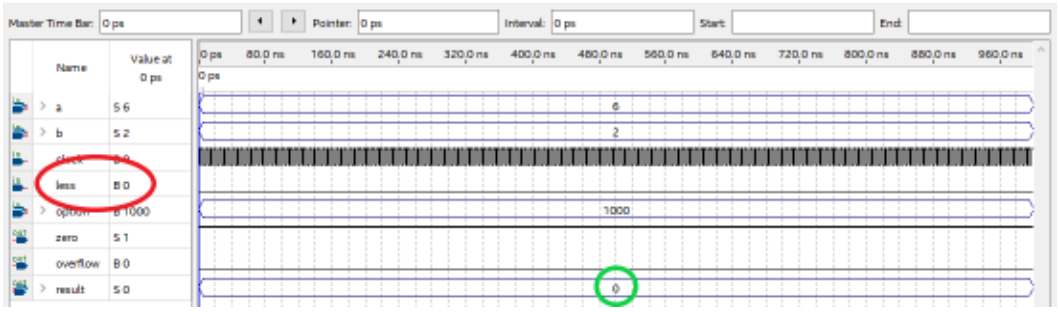
Εικόνα 129

Πρόσθεση με την μονάδα ALU με την γεννήτρια πρόβλεψης κρατουμένου(0111)



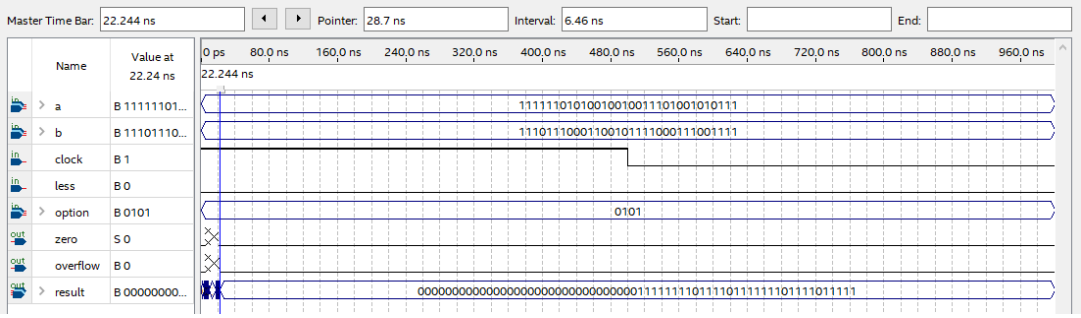
Εικόνα 130

Λογική πράξη less με την απλή μονάδα ALU (1000)



Εικόνα 131

<u>Χρονική εξομοίωση της συνδυαστικής μονάδας</u> (compined _Waveform)



Εικόνα 132

6.2 Ελεγκτής (Controller)

6.2.1 Υλοποίηση/Ελεγχος λειτουργικότητας

Ο ελεγκτής όπως προαναφέραμε,ελέγχει τις λειτουργίες των μονάδων ανάλογα με την εντολή που θα του δώσουμε. Η υλοποίηση του ελεγκτή έγινε με την χρήση της γλώσσας περιγραφής υλικού VHDL. Προτού γράψουμε το πρόγραμμα,υλοποιήσαμε τον πίνακα λειτουργειών, για τον οποίο κάναμε αναφορά παραπάνω και μετά προβήκαμε στην συγγραφή του. Το αρχείο του προγράμματος VHDL στο πρόγραμμα Quartus Elite Prime το ονομάσαμεcontrol_unit_for_divide. Για το σχηματικό αυτό δεν εκτελέσαμε εξομοιώσεις καθώς ήταν πιο εύκολο να το δοκιμάσουμε σε συνδυασμό με τα υπόλοιπα στοιχεία. [11]

<u>ΚώδικαςVHDLτουελεγκτή</u>(control_unit_for_divide)

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

ENTITY controller IS

    PORT ( multiply               : IN STD_LOGIC_VECTOR(63 DOWNTO 0);
           divide                 : IN STD_LOGIC_VECTOR(63 DOWNTO 0);
           alu                    : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
           carry_look_ahead_alu    : IN STD_LOGIC_VECTOR(31 DOWNTO 0);

           option                    : IN STD_LOGIC_VECTOR(3 DOWNTO 0);

           multiply_overflow         : IN STD_LOGIC;
           multiply_zero             : IN STD_LOGIC;
           divide_overflow           : IN STD_LOGIC;
           divide_zero               : IN STD_LOGIC;
           alu_overflow              : IN STD_LOGIC;
           alu_zero                  : IN STD_LOGIC;
           carry_look_ahead_alu_zero     : IN STD_LOGIC;


           result   : OUT STD_LOGIC_VECTOR(63 DOWNTO 0);

           multiply_enable : OUT STD_LOGIC;
           divide_enable : OUT STD_LOGIC;
           op1_alu : OUT STD_LOGIC;
           op0_alu : OUT STD_LOGIC;
           bnegate_alu : OUT STD_LOGIC;
           op1_carry_look_ahead_alu : OUT STD_LOGIC;
           op0_carry_look_ahead_alu : OUT STD_LOGIC;
           bnegate_carry_look_ahead_alu : OUT STD_LOGIC;

           overflow : OUT STD_LOGIC;
           zero     : OUT STD_LOGIC

           );

END controller ;

ARCHITECTURE behavioral OF controller IS
BEGIN
    PROCESS ( option )

    variable first_bits : STD_LOGIC_VECTOR(31 DOWNTO 0);

    BEGIN

       first_bits := "00000000000000000000000000000000";

       IF option="0000" THEN
```

```
51          .
52          multiply_enable <= '1';
53
54          result <= multiply;
55
56          overflow <= multiply_overflow;
57          zero <= multiply_zero;
58
59      ELSIF option="0001" THEN
60
61          divide_enable <= '1';
62
63          result <= divide;
64
65          overflow <= divide_overflow;
66          zero <= divide_zero;
67
68      ELSIF option="0010" THEN
69
70          op1_alu <= '1';
71          op0_alu <= '0';
72          bnegate_alu <= '0';
73
74          result <= first_bits & alu;
75
76          overflow <= alu_overflow;
77          zero <= alu_zero;
78
79      ELSIF option="0011" THEN
80
81          op1_alu <= '1';
82          op0_alu <= '0';
83          bnegate_alu <= '1';
84
85          result <= first_bits & alu;
86
87          overflow <= alu_overflow;
88          zero <= alu_zero;
89
90      ELSIF option="0100" THEN
91
92          op1_alu <= '0';
93          op0_alu <= '0';
94          bnegate_alu <= '0';
95
96          result <= first_bits & alu;
97
98          overflow <= alu_overflow;
99          zero <= alu_zero;
100
101     ELSIF option="0101" THEN
102
103         op1_alu <= '0';
104         op0_alu <= '1';
105         bnegate_alu <= '0';
106
107         result <= first_bits & alu;
108
109         overflow <= alu_overflow;
110         zero <= alu_zero;
111
112     ELSIF option="0110" THEN
113
114         op1_carry_look_ahead_alu <= '1';
115         op0_carry_look_ahead_alu <= '0';
116         bnegate_carry_look_ahead_alu <= '1';
117
118         result <= first_bits & carry_look_ahead_alu;
119
120         overflow <= '0';
121         zero <= carry_look_ahead_alu_zero;
122
123     ELSIF option="0111" THEN
124
125         op1_carry_look_ahead_alu <= '1';
126         op0_carry_look_ahead_alu <= '0';
127         bnegate_carry_look_ahead_alu <= '0';
128
129         result <= first_bits & carry_look_ahead_alu;
130
131         overflow <= '0';
132         zero <= carry_look_ahead_alu_zero;
133
134     ELSE
135
136         op1_alu <= '1';
137         op0_alu <= '1';
138         bnegate_alu <= '0';
139
140         result <= first_bits & alu;
141
142         overflow <= alu_overflow;
143         zero <= alu_zero;
144
145     END IF;
146   END PROCESS;
147 END behavioral;
148
```

Εικόνα 133

ΣΥΜΠΕΡΑΣΜΑΤΑ

Σε αυτό το τελευταίο μέρος της πτυχιακής εργασίας θα παραθέσουμε τα τελικά μας συμπεράσματα, τις δυσκολίες που αντιμετωπίσαμε κατά την σχεδίαση της αριθμητικής και λογικής μονάδας και θα αναφέρουμε ορισμένες μελλοντικές βελτιώσεις της.

Η παρούσα πτυχιακή εργασία έγινε με σκοπό την σχεδίαση των ψηφιακών κυκλωμάτων μέσω της υλοποίησης της αριθμητικής και λογικής μονάδας ALU. Για τα σχηματικά και τα προγράμματα που υλοποιήσαμε χρησιμοποιήσαμε το πρόγραμμα Intel Quartus καθώς και την γλώσσα περιγραφής υλικού VHDL.Με την υλοποίηση των κυκλωμάτων ταυτόχρονα προβήκαμε στην εξομοίωση αυτών προκειμένω να επιβεβαιώσουμε ότι λειτουργούν ορθά, χρησιμοποιώντας τις λογικές και χρονικές εξομοιώσεις του προγράμματος.

Για την υλοποίηση της αριθμητικής και λογικής μονάδας ALU 32 bitχρησιμοποιήσαμε 32 φορές την αριθμητική και λογική μονάδα του 1 bit. Όσον αφορά τους μηχανισμούς του πολλαπλασιασμού παρατηρήσαμε ότι γίνεται μια σειρά πράξεων πρόσθεσης και δεξιάς μετατόπισης. Στην δε διαίρεση παρατηρήσαμε ότι γίνεται μια σειρά πράξεων αφαίρεσης, πρόσθεσης και αριστερής, δεξιάς μετατόπισης. Σχετικά με την γεννήτρια πρόβλεψης κρατουμένου από τα αποτελέσματα που καταγράψαμε συγκρίνοντας τους χρόνους καθυστέρησης της αριθμητικής και λογικής μονάδας με και χωρίς την χρήση της γεννήτριας πρόβλεψης παρατηρήσαμε ότι με την τελευταία χρειαστήκαμε λιγότερο χρόνο για να πάρουμε το τελικό αποτέλεσμα.

Για την υλοποίηση των κυκλωμάτων δεν αντιμετώπισα ιδιαίτερες δυσκολίες καθώς με το πρόγραμμα Intel Quartus και με την γλώσσα περιγραφής υλικού υπήρχε μια πρότερη εξοικείωση. Ωστόσο οι δυσκολίες που κλήθηκα να αντιμετωπίσω επιλύθηκαν ακολουθώντας τις συμβουλές και τις οδηγίες του καθηγητή μου. Αυτές αφορούσαν την υλοποίηση της μονάδας ελέγχου του πολλαπλασιασμού καθώς δεν γνώριζα με πιο τρόπο θα ήταν πιο σωστό να την υλοποιήσω (σχηματικά ή με συγγραφή προγράμματος). Οι λοιπές δυσκολίες αφορούσαν την συγγραφή του κώδικα ορισμένων στοιχείων που καταχωρούνταν από απροσεξία δική μου λανθασμένα με αποτέλεσμα να βγάζουν σφάλμα γεγονός που με έφερνε πίσω καθώς απαιτούνταν χρόνος να τα εντοπίσω και να τα διορθώσω. Από αυτό οδηγήθηκα στο συμπέρασμα ότι για την συγγραφή των μηχανών πεπερασμένων καταστάσεων που είχαμε στην υποδεέστερη μονάδα ελέγχου του πολλαπλασιασμού και της διαίρεσης καλό θα ήταν πρώτα να σχεδιαστούν στο χαρτί οι καταστάσεις να δοθούν ονομασίες σε αυτές και ακολούθως να περαστούν στον κώδικα.

Επιπλέον, στην ονομασία των αρχείων δώσαμε ίδια ονόματα στην υλοποίηση του αρχείου καθώς και του αρχείου της εξομοίωσης του με την μόνη διαφορά ότι στο τέλος της ονομασίας του αρχείου της εξομοίωσης πρόσθεσα τον όρο _Waveform ώστε να γίνεται ευκολά ο εντοπισμός του (π.χ. το σχηματικό του μετρητή ονομάζεται counter και το αρχείο της εξομοίωσής του counter_Waveform).

Στις μελλοντικές βελτιώσεις θα μπορούσαμε να αναφέρουμε την υλοποίηση κυκλώματος που να εκτελεί πράξεις στα 64 bit, πράξειςκινητής υποδιαστολής, αλλά και περαιτέρω βελτίωση της συνδυαστικής μονάδας του έκτου κεφαλαίου.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- [1]https://www.csd.uoc.gr/~hy120/10f/lab11_dpath.html
- [2]https://www.techtarget.com/whatis/definition/arithmetic-logic-unit-ALU
- [3]https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html
- [4]https://en.wikipedia.org/wiki/Very_High_Speed_Integrated_Circuit_Program
- [5]https://www.ceid.upatras.gr/webpages/faculty/alexiou/vlsi2/other/VHDL_4Presentation
- [6]http://archive.eclass.uth.gr/eclass/modules/document/file.php/DIB260/2.pdf
- [7]M. MorrisMano, Michael D. Ciletti, *Ψηφιακή Σχεδίαση, Έκτη Έκδοση*. Εκδόσεις Παπασωτηρίου, 2018
- [8]N. Weste& D. Harris, *Σχεδίαση Ολοκληρωμένων Συστημάτων CMOS VLSI* Εκδόσεις: Παπασωτηρίου, 2011.
- [9]http://teachers.cm.ihu.gr/kalomiros/Proigmena_Psifiaka_Theoria/didaktiko_yliko/Intro_VHDL_Kalomiros_teuxos2.pdf
- [10]https://newtech-pub.com/wp-content/uploads/2014/03/FPGAKefalaio1.pdf
- [11]ALU (σημειώσεις ΚαθηγητήEclass)
- [12]https://www.youtube.com/watch?v=p4yVpZGZ9tA
- [13]https://en.wikipedia.org/wiki/Carry-lookahead_adder
- [14]Ακολουθιακά μπλοκς (σημειώσεις ΚαθηγητήEclass:https://www.dit.uoi.gr/e-class/modules/document/file.php/151/%CE%94%CE%B9%CE%AC%CE%BB%CE%B5%CE%BE%CE%B7/3_%CE%91%CE%BA%CE%BF%CE%BB%CE%BF%CF%85%CE%B8%CE%B9%CE%B1%CE%BA%CE%B1_%CE%BC%CF%80%CE%BB%CE%BF%CE%BA%CF%82_v1.pdf)
- [15]Μηχανή πεπερασμένων καταστάσεων (σημειώσεις ΚαθηγητήEclass:https://www.dit.uoi.gr/e-class/modules/document/file.php/151/%CE%94%CE%B9%CE%AC%CE%BB%CE%B5%CE%BE%CE%B7/4_Finite_state_machinesv1.pdf)
- [16]Πρόγραμμα QuartusPLE (σημειώσεις ΚαθηγητήEclass:https://www.dit.uoi.gr/e-class/modules/document/file.php/151/%CE%95%CE%A1%CE%93%CE%91%CE%A3%CE%A4%CE%97%CE%A1%CE%99%CE%91%CE%9A%CE%95%CE%

A3%20%CE%91%CE%A3%CE%9A%CE%97%CE%A3%CE%95%CE%99%CE%A3/%CE%95%CE%B9%CF%83%CE%B1%CE%B3%CF%89%CE%B3%CE%AE%20%CF%83%CF%84%CE%BF%20%CE%B5%CF%81%CE%B3%CE%B1%CE%BB%CE%B5%CE%AF%CE%BF%20Quartus%20PLE.pdf)