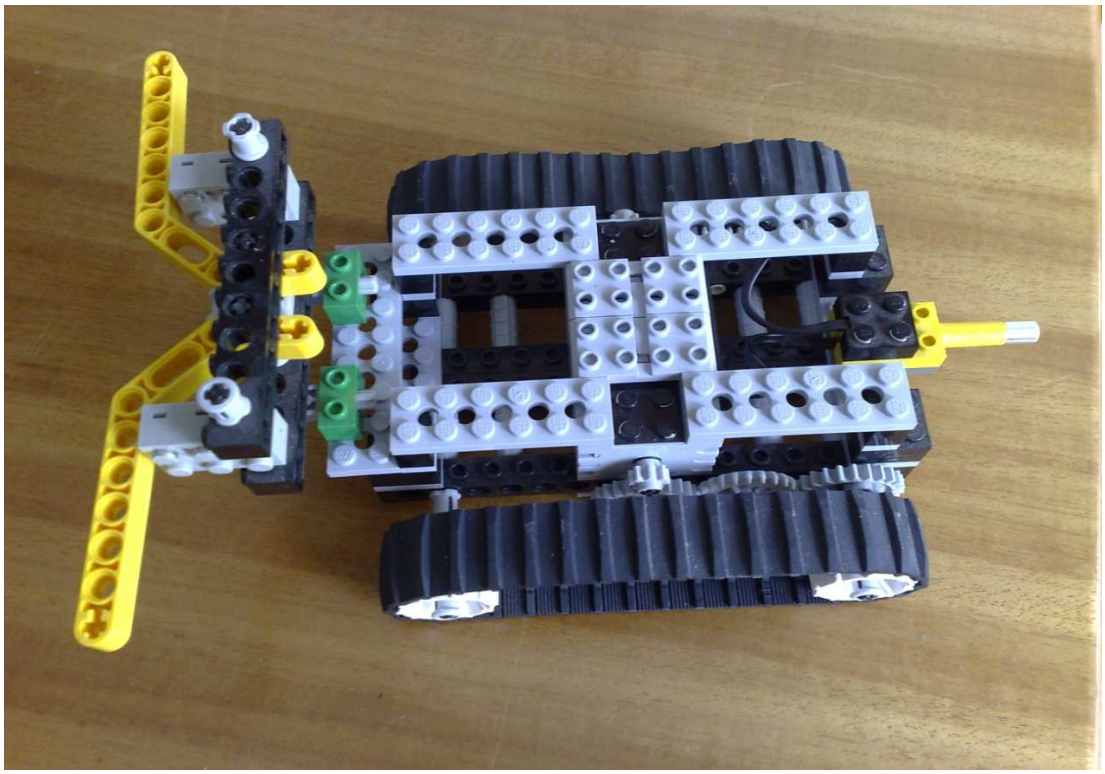
ΑΛΕΞΑΝΔΡΕΙΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
ΘΕΜΑ:

ΑΣΥΡΜΑΤΟΣ ΕΛΕΓΧΟΣ ΑΥΤΟΚΙΝΗΤΟΥ



ΦΟΙΤΗΤΕΣ ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΡΑΜΕΤΣΙΟΣ ΝΙΚΟΛΑΟΣ
ΛΑΛΟΣ ΒΑΣΙΛΕΙΟΣ

ΕΠΟΠΤΗΣ ΚΑΘΗΓΗΤΗΣ
ΧΡΗΣΤΟΣ Β. ΤΖΙΚΑΣ
ΚΑΘΗΓΗΤΗΣ ΕΦΑΡΜΟΦΩΝ

ΘΕΣΣΑΛΟΝΙΚΗ
ΣΕΠΤΕΜΒΡΙΟΣ 2007

ΑΣΥΡΜΑΤΟΣ ΕΛΕΓΧΟΣ ΑΥΤΟΚΙΝΗΤΟΥ

ΦΟΙΤΗΤΕΣ ΗΛΕΚΤΡΟΝΙΚΗΣ

ΚΑΡΑΜΕΤΣΙΟΣ ΝΙΚΟΛΑΟΣ
ΛΑΛΟΣ ΒΑΣΙΛΕΙΟΣ

ΕΠΟΠΤΗΣ ΚΑΘΗΓΗΤΗΣ

ΧΡΗΣΤΟΣ Β. ΤΖΙΚΑΣ
ΚΑΘΗΓΗΤΗΣ ΕΦΑΡΜΟΦΩΝ

ΘΕΣΣΑΛΟΝΙΚΗ
ΣΕΠΤΕΜΒΡΙΟΣ 2007

Ευχαριστούμε,

‘’ για την βοίθεια που μας προσέφεραν,
κ. Τζίκα Χρίστο, καθ. εφαρμογών
κ. Αρζουμανίδη Ευσέβιο, εργ.συνεργάτης
κ. Βάσσιο Βασίλειο, εργ.συνεργάτης
κ. Γαβριηλίδη Δημήτριο, φοιτητής’’

ΚΑΡΑΜΕΤΣΙΟΣ ΝΙΚΟΛΑΟΣ
ΛΑΛΟΣ ΒΑΣΙΛΕΙΟΣ

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ…….

ΜΕΡΟΣ Α’

CPLDs κια FPGAs.....
1.1 ΤΥΠΙΚΑ ΛΟΓΙΚΑ ΟΛΟΚΛΗΡΩΜΕΝΑ ΚΥΚΛΩΜΑΤΑ…….....
1.2 ΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΕΣ ΛΟΓΙΚΕΣ ΣΕΙΡΕΣ (PLA ή PLD)…….
1.3 ΠΟΛΥΠΛΟΚΕΣ ΔΙΑΤΑΞΕΙΣ ΠΡΟΓΡΑΜΑΤΙΖΟΜΕΝΗΣ ΛΟΓΙΚΗΣ (CPLDs)……
1.4 Η ΟΙΚΟΓΕΝΕΙΑ XC9500 CPLD ΤΗΣ XILINX…….
1.5 ΔΙΑΤΑΞΕΙΣ ΠΥΛΩΝ ΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΟΥ ΠΕΔΙΟΥ (FPGAs)…….

ΜΕΡΟΣ Β’

ΔΟΜΗ ΑΣΥΡΜΑΤΟΥ ΕΛΕΓΧΟΥ…….
2.1 ΚΥΚΛΩΜΑ ΔΕΔΟΜΕΝΩΝ ΕΙΣΟΔΟΥ…….
2.2 ΚΥΚΛΩΜΑ ΠΟΜΠΟΥ……
2.3 ΠΟΜΠΟΣ……
2.4 ΔΕΚΤΗΣ……
2.5 ΚΥΚΛΩΜΑ ΔΕΚΤΗ……
2.6 ΚΙΝΗΤΙΡΕΣ……
2.7 ΧΡΟΝΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ……
2.8 ΕΠΑΝΑΤΟΠΟΘΕΤΗΣΗ (RESET)……

ΜΕΡΟΣ Γ’

ΑΝΑΠΤΥΞΗ ΑΣΥΡΜΑΤΟΥ ΕΛΕΓΧΟΥ…….
3.1 ΑΝΑΠΤΥΞΗ ΚΥΚΛΩΜΑΤΟΣ ΠΟΜΠΟΥ ΣΤΟ CPLD XC9572 ΤΗΣ XILINX……
3.2 ΠΡΟΣΟΜΟΙΩΣΗ ΚΥΚΛΩΜΑΤΟΣ ΠΟΜΠΟΥ ΜΕ ΤΟ MODELSIM…….
3.3 ΑΝΑΠΤΥΞΗ ΚΥΚΛΩΜΑΤΟΣ ΔΕΚΤΗ ΣΤΟ CPLD XC9572 ΤΗΣ XILINX…….
3.4 ΠΡΟΣΟΜΟΙΩΣΗ ΚΥΚΛΩΜΑΤΟΣ ΔΕΚΤΗ ΜΕ ΤΟ MODELSIM……

ΜΕΡΟΣ Δ’

ΣΤΟΙΧΕΙΑ ΑΣΥΡΜΑΤΟΥ ΕΛΕΓΧΟΥ……..
4.1 ΤΟ ΣΥΜΒΟΛΟ AND2B1……
4.2 ΤΟ ΣΥΜΒΟΛΟ baud_cnt…….
4.3 ΣΥΜΒΟΛΟ kodikopiitis……
4.4 ΣΥΜΒΟΛΟ LD4……..
4.5 ΣΥΜΒΟΛΟ OR4……
4.6 ΤΟ ΣΥΜΒΟΛΟ rx……
4.7 ΤΟ ΣΥΜΒΟΛΟ tx……..

ΜΕΡΟΣ Ε’

ΤΟ ΠΡΟΓΡΑΜΜΑ XILINX ISE 7.1i……..
2.1 ΕΙΣΑΓΩΓΗ ΣΤΟ ΠΡΟΓΡΑΜΜΑ Xilinx 7.1……
2.2 ΣΧΕΔΙΑΣΜΟΣ ΜΕ ΧΡΗΣΗ ΣΧΗΜΑΤΙΚΟΥ……
2.3 ΤΟΠΟΘΕΤΗΣΗ ΥΛΙΚΩΝ……
2.4 Τοποθέτηση Συνδέσεων ……
2.5 Ορισμός Ακίδων ……
2.6 Εξομοίωση της λειτουργίας ……
2.7 Προγραμματισμός του FPGA ……
2.8 Δημιουργία αρχείων βιβλιοθήκης (CORE) ……
2.9 Δημιουργία Μηχανής καταστάσεων ……
2.10 Δημιουργία αρχείου VHDL ……..

ΜΕΡΟΣ ΣΤ’

ΑΝΑΠΤΥΞΗ ΠΛΑΚΕΤΩΝ……
6.1 Η ΑΝΑΠΤΥΞΗ ΠΛΑΞΕΤΩΝ ΠΟΜΠΟΥ ΚΑΙ ΔΕΚΤΗ……
6.2 ΜΟΝΑΔΑ ΠΡΟΓΡΑΜΑΤΙΣΤΗ……
6.3 ΜΟΝΑΔΑ ΣΤΑΘΕΡΟΠΟΙΗΤΗ ΤΑΣΗΣ……

ΠΑΡΑΡΤΗΜΑ Α.

ΣΧΟΛΙΑ ΚΑΙ ΠΑΡΑΤΗΡΗΣΕΙΣ……

A BRIEFING IN ENGLISH……

ΒΙΒΛΙΟΓΡΑΦΙΑ……

ΕΙΣΑΓΩΓΗ

Το αντικείμενο αυτής της μελέτης διαπραγματεέεται την ανάπτυξη ενώς ασύρματου ελέγχου αυτοκινήτου μέσα σε ολοκληρωμένο κύκλωμα προγραμματοζώμενης λογικής, CPLD (Complex Programmable Logic Device).

Ένα CPLD αποτελείται από πολλές βαθμίδες κυκλωμάτων που βρίσκονται μέσα σε ένα ολοκληρωμένο κύκλωμα, IC (Integrated Circuit) και συνδέονται μεταξύ τους με εσωτερικές καλωδιώσεις. Μπορούν να υλοποιήσουν λογικά κυκλώματα που περιλαμβάνουν περισσότερα από 100,000 ισοδύναμες πύλες.

Τα CPLDς χρησιμοποιούνται ευρέως στην ανάπτυξη ψηφιακών συστημάτων θψηλής τεχνολογίας. Λόγω της μεγάλης ταχύτητας λειτουργίας, αλλά και της ευελιξίας του, είναι ιδανικά για πληθώρα εφαρμογών, πιο συγκεκριμένα σε τομείς τηλεπικοινωνιακών και υπολογιστικών συστημάτων.

Στο πρώτο μέρος του βιβλίου αυτού γίνεται μια ιστορική αναδρομή της τεχνολογίας των ολοκληρωμένων κυκλωμάτων προγραμματιζόμενης λογικης, καθώς και στα βασικά χαρακτηριστικά των *CPLDs* (*Complex Programmable Logic Devices*) και των *FPGAs* (*Field Programmable Gate Arrays*)

Στο δεύτερο μέρος περιγράφουμε την δομή του ασύρματου ελέγχου. Έχουμε μια λεπτομερή αναφορά στα μέρη που αποτελούν την μονάδα αυτή, στα επιμέρους κυκλώματα της μονάδας ,καθώς και στον τρόπο με τον οποίο γίνεται η μεταφορά των δεδομένων. Τέλος περιγράφεται το κύκλωμα χρονισμού του *CPLD/FPGA* καθώς και των άλλων εξαρτημάτων.

Στο τρίτο μερος έχουμε την ανάπτυξη του ασύρματου ελέγχου στο CPLD XC 9572, το γραφικό συμβολισμό του, αναφορά στα κύρια στοιχεία που υλοποιούν το κύκλωμα και το σχηματικό διάγραμμά του. Τέλος έχουμε μια αναφορά στους πόρους που καταλαμβάνει στο CPLD

καθώς και μια προσομοίωση του κυκλώματος στο πρόγραμμα Modelsim.

Στο τέταρτο μέρος του βιβλίου έχουμε μια λεπτομερή αναφορα στα επιμέρους κυκλώματα του δέκτη και του πομπού, τον κώδικα υλοποίησής τους καθώς και τα σχηματικά διαγράμματά τους. Τέλος γινετε αναφορά στο κύκλωμα ασύρματης μετάδοσης, στο κύκλωμα χρονισμού και στον τρόπο σύνδεσης με τα άλλα εξαρτήματα.

Στο πέμτο μέρος του βιβλίου γίνεται μια αναφορά στο λογισμικό που θα χρησιμοποιηθεί, ώστε να υλοποιηθεί η εφαρμογή, είτε σε γλώσσα V*HDL*, είτε σε σχηματική μορφή (schematic). Το λογισμικό αυτό διανέμεται από την εταιρία Xilinx μέσω του διαδικτύου, όπου βρίσκουμετην απλή έκδοση (web pack) δωρεάν, ή διάφορες άλλες επαγγελματικές εκδόσεις με πληρωμή.

Στο έκτο και τελευταίο μέρος έχουμε την ανάπτυξη των πλακετών του πομπού και δέκτη, τον τροπο σύνδεσης του programmer, καθώς και την μέθοδο υλοποίησης του οχήματος.

ΚΕΦΑΛΑΙΟ 1

CPLDs ΚΑΙ FPGAs

Η αυξανόμενη χρήση των ψηφιακών κυκλωμάτων οφείλεται εν μέρει στην ύπαρξη φθηνών ολοκληρωμένων κυκλωμάτων. Οι κατασκευαστές έχουν αναπτύξη πολλλες οικογένεις ψηφιακών ολοκληρωμένων κυκλωμάτων, δηλαδή ομάδες που μπορουν να συνδεθούν μεταξύ τος και να συγκροτήσουν ένα ψηδικό σύστημα. Τα ολοκλυρωμένα μίας οικογένειας λέγεται ότι είναι συμβατα μεταξύ του (compatible) και μπορούν να διασυνδεθούν εύκολα.

Τα ψηφιακά ολοκληρωμένα κυκλώματα διακρίνονται σε δύο κατηγορίες, τα διπολικά (bipolar) και τα μονοπολικά (unipolar). Τα διπολικά ψηφιακά ολοκληρωμένα κυκλώματα δομούνται από διακριτά τμήματα που είναι ανάλογα των διπολικών τρανζίστορ, διόδων και αντιστάσεων. Η οικογένεια TTL (transistor-transistor-logic) είναι η πιο δημοφιλής οικογένεια ολοκλυρωμένων κυκλωμάτων διπολικής τεχνολογίας. Τα μονοπολικα ψηδιακά ολοκληρωμένα κυκλώματα κατασκευάζονται από τμήμα που είναι ανάλογα των τρανζιστορ επίδρασης πεφίου μονωμένης πύλης (insulated-gate field-effect transistor, IGFETs). Η οικογένεια CMOS (complementary metal-oxide-semiconductor FET) έχει σήμερα ευρεία χρήση και στηρίζεται στην τεχνολογία MOS (metal-oxide-semiconductor).

1.1 ΤΥΠΙΚΑ ΛΟΓΙΚΑ ΟΛΟΚΛΗΡΩΜΕΝΑ ΚΥΚΛΩΜΑΤΑ

Τα ολοκληρωμένα κυκλώματα κατατάσσονται συχνά από τους κατασκευαστές, σύμφωνα με την πολυπλοκότητα του κυκλώματός τους. Η πολυπλοκότητα των ολοκληρωμένων κυκλωμάτων διακρίνεται στις ακόλουθες κατηγορίες:

- **Ολοκλήρωση μικρής κλίμκας, SSI (Small Scale Integration).** Με την μέθοδο αυτή κατασκευάζονται κατ' αρχάς απλά κυκλώματα μικρού αριθμού στοιχειωδών δομικών στιχελιων της τάξεως των 20, όπως οι φιάφορες πύλες. Αριθμός τρανζιστορ 1-10.
- **Ολοκήρωση μέσης κλίμακος, MSI(Medium Scale Integration).**
  Κατασκεθάζονται συνθετότερα κυκλώματα, αριθμού δομικών στοιχείων, π.χ. πυλών, της τάξεως των 100, όπως φιάφοροι πολυδονητές, μετρητές, αποκωδικοποιητές κ.τ.λ.. αριθμός τρανζίστοε 100-500.
- **Ολοκλήρωση μεγάλης κλίμακας, LSI (Large Scale Integration).**
  Κατασκευή σύνθετων κυκλωμάτων τα οποία απαιτουν για την κατασκευή τους π.χ. πύλες της τάξεως των 1000. Στην κατηγορία αυτή ανήκουν όλα τα κυκλώματα C-MOS και η μικρής και μέσης χωρητικότητας στερεάς κατάστασης. Αριθμός τρανζιστορ 10,000-20,000
- **Ολοκήρωση πολύ μεγάλης κλίμακας, VLSI (Very Large Scale Integration).**
  Κατασκευή εξαιρετικά σύν8ετων κυκλωμάτων, όπου υπάρχουν περισσότερα μεγάλα σύνθετα κυκλώματα, που παλαιότερα κατασκευάζονταν σαν αυτοτελή κυκλώματα με τις προηγούμενες μεθόδους. Χρήση ομοειδών στοιχειωδών κυκλωμάτων της τάξης άνω των 1000. Στην κατηγορία αυτή ανήκουν όλα τα κυκλώματα μικροεπεξεργαστών και η μεγάλης χωρητικότητας μνήμες στερεας καταστασης. Αριθμός τρανζίστορ 50,000-100,000.

- **Ολοκλήρωση πάρα πολύ μεγάλης κλίμακας, SLSI (Super Large Scale Integration).**
  Στην κατηγορία αυτή ανήκουν όλα τα σύγχρονα ολοκλυρωμένα κυκλώματα. Αριθμός τρανζίστορ πάνω από 100,000.

Σήμερα ευρίσκονται στη διάθεση του μελετητή πολλές οικογένεις ψηφιακών κυκλωμάτων και μερικές από αυτές είναι οι παρακάτω:

- **RTL (Resistor Transistor Logic), λογική αντίστασης τρανζίστορ.**
  Σειρά 900
- **DTL (Diode Transistor Logic), λογική διόδου τρανζίστορ.**
  Σειρά 930
- **HTL (High Threshold Logic), λογική υψηλού κατωφλίου.**
  Σειρά MC660
- **TTL (Transistor Transistor Logic), λογική τρανζίστορ τρανζίστορ**
  Σειρά 54/74
- **L TTL (Low Power TTL), TTL μικρής ισχύος.**
  Σειρά 54L/74L
- **LS TTL (Low Power Schottky TTL), Schottky TTL μικρής ισχύος.**
  Σειρά 54LS/74LS
- **H TTL (High Speed TTL), TTL μεγάλης ταχύτητας.**
  Σειρά 54H/74H
- **S TTL (Schottky TTL), Schottky TTL.**
  Σειρά 54S/74S
- **AS TTL (Advanced Schottky TTL), εξελιγμένη Schottky TTL.**
  Σειρά 54ALS/74ALS
- **ALS TTL (Advanced Low Power Schottky TTL), εξελιγμένη Schottky TTL μικρής ισχύος.**
  Σειρά 54ALS/74ALS
- **IIL (Integrated Injection Logic), λογική εγχύσεως ρεύματος.**
- **ECL (Emitter Coupled Logic), λογική συζεύξεως εκπομπού.**
  Σειρα MC 1000/1200/1600
  Σειρα MC 2500
  Σειρα MC 3000

- **C-MOS (Complementary Symmetry MOSFET),κυκλώματα συμπληρωματικής συμετρίας MOSFET**
  Σειρά 400
  Σειρα 74C
  Σειρά 74HC
- **P-MOS (P MOSFET), κυκλώματα με MOSFET**
- **N-MOS (N MOSFET), κυκλώματα με N MOSFET**
- **CCD (Charge Coupled Devices),κυκλώματα σύζευξης φορτίου**

Οι τεχνολογίες TTL και CMOS χρησιμοποιούνται για την κατασκευή ολοκληρωμένων κυκλωμάτων SSI και MSI. Τα κυκλώματα αυτά περιλαμβάνουν συσκευές όπως λογικές πίλες, flip-flop, κωδικοποιητές και αποκωδικοποιητές, πολυπλέκτες, μανδαλοτες και καταχωρητές. Οι συσκευές MOS (PMOS, NMOS και CMOS) κυριαρχούν στην κατασκευή συσκευών LSI και VLSI. Η τεχνολογία NMOS είναι ιδιαίτερα δημοφιλής για την κατασκευή μικροεπεξεργαστών και μνημών. Η τεχνολογία CMOS χρησιμοποιείται σε εφαρμογές πολύ χαμηλής ισχύος, όπως οι μυπολογιστές τσέπης, τα ρολόγια χειρός και οι φορητοί υπολογιστές.

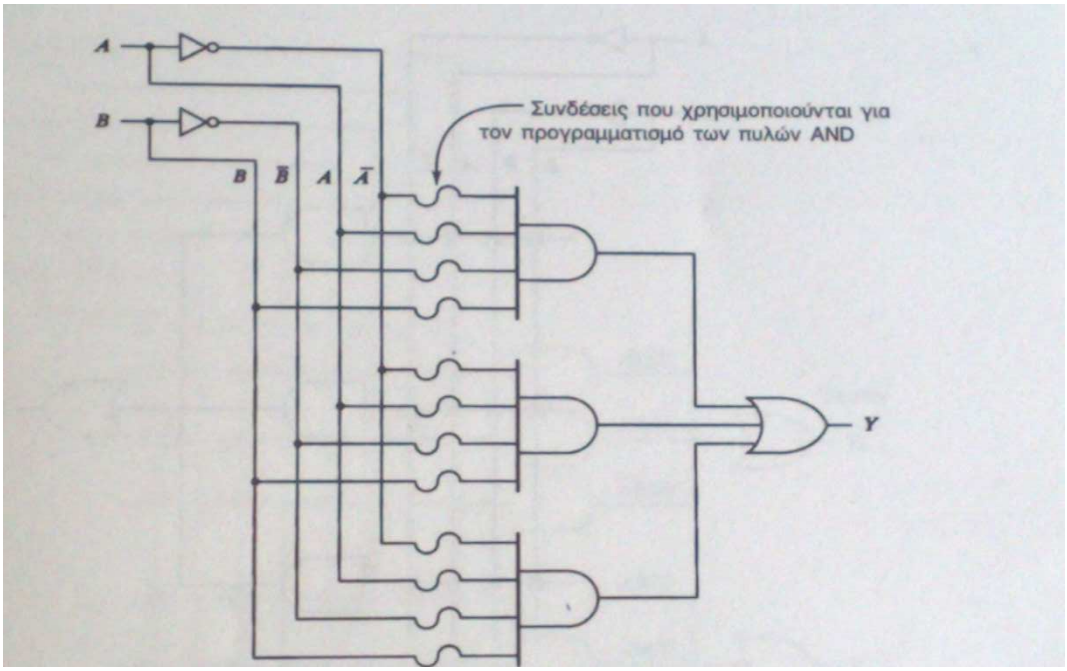
1.2 ΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΕΣ ΛΟΓΙΚΕΣ ΣΕΙΡΕΣ (PLA ή PLD)

Προγραμματιζόμενη λογική διάτjη (programmable logical array, PLA) ονομάζεται ένα ολοκληρωμένο κύκλωμα που μπορεί να προγραμματίσει, ώστε να εκτελεί μια περίπλοκη λογική λειτουργία. Συνήθως χρησιμοποιούνται για την υλοποίηση προβλημάτων συνδυαστικής λογικής αλλά μερικές προγραμματιζόμενες λογικές διατάξεις μπορούν και να χρησιμοποιηθούν και σε ακολουθιακά κυκλώματα. Οι διατάξεος αυτές, που στην συνέχεια θα τις αναφέρουμε ως PLA, αποτελούν μια συμπαγή λύση σε πολλά λογικά προβλήματα που έχουν πολλές εισόδους και εξόδους. Οι PLA σχετίζονται με τις μνήμες PROM και μάλιστα προγραμματίζονται περίπου όπως αυτές. Οι PLA μπορού και να ονομαστούν προγραμματιζόμενες λογικες μονάδες (programmable logic device, PLD). Οι όροι PLD και PLA χρησιμοποιούνταο αδιάκριτα, για να δηλώσουν αυτές τις προγραμματιζόμενες λογικές μονάδες. Μια δημοφιλής

προγραμματιζόμενη μονάδα ονομάζεται PAL (programmable array logic) και αποτελεί σήμα κατατεθέν της εταιρίας Advanced Micro Devices Inc

Η χρήση των PLD επιτρέπει την μείωση του κόστους, επειδή χρησιμοποιούνται λιγότερα ολοκληρωμένα κυκλώματα για να υλοποιήσουν ένα λογικό κύκλωμα. Τα PLD είναι ταχύτερα από τα ολοκληυρωμ΄ρνα κυκλώματα πυλών, με κλίμακα ολοκλήρωσης SSI, τα οποία μπορούν να προσαρμοστούν σε ένα τυπωμένο κύκλωμα. Υπάεχουν σήμερα διάφορα προγράμματα λογισμικού,τα οποία επιτρέπουν τον προγραμματοσμό των PLD και καθιστούν εύκολη υπόθεση την πραγματοποίησηαλλαγών στο,αρχικό σχέδιο.άλλα πλεονεκτήματα των PLD είναι ότι έχουν μειωμένο κόστος καινοτομίας,επειδή αποτελούν μονάδες βάσης, και ότι έχουν χαμιλό κόστος αωαβαθμίσεων και τροποποιήσεων. Τα PLD αποτελούν εξαιρετικά αξιόπιστες μονάδες και τα πνευματικά δικαιώματα τον λογικών σχεδίων που υλοποιούνται από τα PLD μπορούν να αποκρυφθούν από ανταγωνιστές εάν χρησιμοποιηθεί η ασφάλεια προστασίας που έχει τοποθετήσει για το σκοπό αυτό ο κατασκευαστής.

Το λογικό διάγραμμα μιας απλής μονάδας PLA εικονίζεται στο Σχ.1.1. Παρατηρούμε ότι αυτή η μονάδα διαθέτει μόνο δύο εισόδους και μια έξοδο. Μια τυπική έκδοση του εμπορίου μπορεί να έχει έως 12 εισώδους και 10 εξόδους. Στο Σχ.1.1 παρατηρούμε την οργάνωση AND-OR των λογικών πυλών, οι οποίες μπορούν να θλοποιήσουν οποιαδήποτε λογική έκφραση γραμμένη σε μορφή minterm. Η απλοποιημένη μονάδα PLA του σχήματος διαθέτει συνδέσεις που μπορούν να καταστραφούν, ώστε να προγραμματιστουν οι πύλες AND. Η πύλη OR σε αυτή την μονάδα δεν προγραμματίζεται. Η μονάδα PLA του Σχ.1.1 εικονίζεται όπως παρέχεται από τον κατασκευαστή, δηλαδή με όλες τις συνδέσεις στη θέση τους και μπορεί να προγραμματιστεί καταστρέφοντας επιλεγμένες συνδέσεις.



Σχ1.1 Λογικό διάγραμμα μιας μονάδας PLA

1.3 ΠΟΛΥΠΛΟΚΕΣ ΔΙΑΤΑΞΕΙΣ ΠΡΟΓΡΑΜΑΤΙΖΟΜΕΝΗΣ ΛΟΓΙΚΗΣ (CPLDs)

Τα PLAs, PAL, GAL και MAPL είναι χρήσιμα για την υλοποίηση μια ευρείας ποικιλίας μικρών ψηφιακών κυκλωμάτων. Κάθε ένα από αυτά μπορεί να χρησιμοποιηθεί για την υλοποίηση κυκλωμάτων που δεν απαιτουν περισσότερες εισόδους, εξόδους και όρους γινιμένου, από αυτά που παρέχει το εκάστοτε ολοκληρωμένο κύκλωμα. Τα ολοκλυρωμένα αυτά κυκλώματα περιορίζονται σε εν γένει μικρά μεγέθη και συνήθως υποστηρίζουν ένα συνολικό αριθμό εισόδων και εξόδων όχι μεγαλύτερο από 32. Για την υλοποίηση κυκλωμάτων που απαιτουν περισσότερες εισόδους και εξόδουε ενδείκνυνται πιο εξειδικευμένα ολοκληρβμένα κυκλώμαζτα, όπως είναι οι πολύπλοκες διατάξεις προγραμματιζόμενης λογικής, CPLDς (Complex PLDs). Ένα CPLD αποτελείτε από πολλές βαθμίδες κυκλωμάτων (μακροκυψέλες) που βρίσκονται μέσα στο ολοκληρωμένο κύκλωμα και συνδέονται μεταξύ τους με εσωτερικές καλωδιώσεις. Κάθε βαθμίδα κυκλώματος ομοιάζει με ένα PLA ή ένα PAL. Τα CPLDs του εμπορίου έχουν μεγέθη που κυμαίνονται μεταξύ δύο βαθμίδων PAL και περισσοτέρων από 100 βαθμίδων PAL. Μπορούν να υλοποιήσουν κυκλώματα που περιλαμβάνουν έως και 100.000 ισοδύναμες πύλες. Κατασκευάζονται από διάφορες εταιρείες, όπως η Xilinx, η Atmel και η Altera και κυκλοφορούν σε διάφορες συσκευασίες όπως PLCC και QFP.

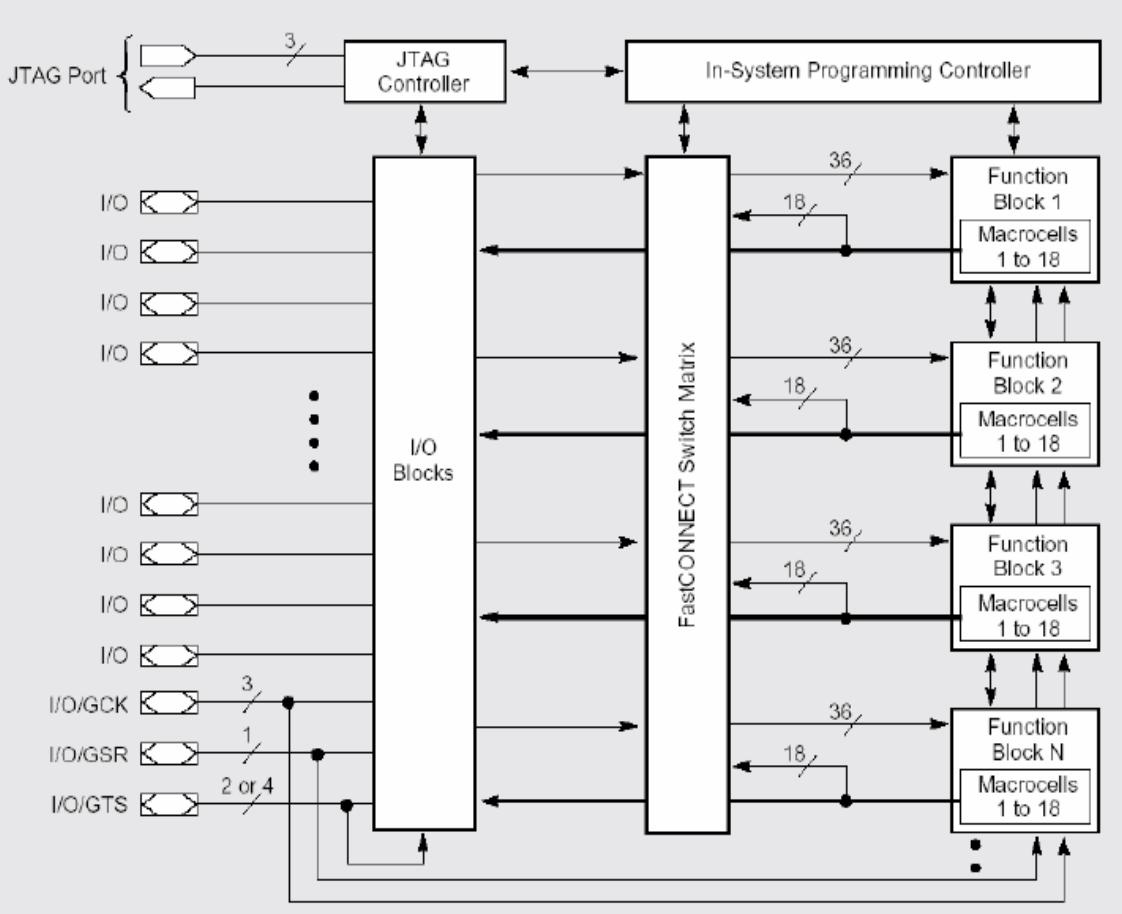
Η Αμερικανικής προελεύσεως εταιρεία, Xilinx, προσφέρι μια πολύ μεγάλη ποικιλία από CPLD's που καλύπτουν τις ανάγκες κάθε σχεδιαστή. Αυτή την στιγμή η εν λόγο εταιρεία διαθέτει πέντε οικογένειες CPLDs. Κάθε οικογένεια αποτελείται από μια σειρά ολοκληρωμένων κυκλωμάτων με κοινή αρχιτεκτονική και χαρακτηριστικά, η διαφορά τους βρισκεται στο μέγεθος του λογικού κυκλώματος που μπορουν να υλοποιήσουν. Σε κάθε ολοκληρωμένο κύκλωμα η χαρακτηριστική του κωδική ονομασία δηλώνει σε ποια οικογένεια ανήκει και πόσες μακροκυψέλες περιέχει. Πιο κάτω παρουσιάζοντια οι οικογένειες αυτές και τα ολοκληρωμένα κυκλώματά τους.

- **XC9500 ή XC9500XL ή XC9500XV**
  - XC9536\XL\XV
  - XC9572\XL\XV
  - XC95108
  - XC95144\XL\XV
  - XC95216
  - XC95218\XL\XV
- **Cool Runner XPLA3**
  - XCR3032XL
  - XCR3064XL
  - XCR3128XL
  - XCR3256XL
  - XCR3384XL
  - XCR3512XL
- **Cool Runner 2**
  - XC2C32
  - XC2C64
  - XC2C128
  - XC2C256
  - XC2C384
  - XC2C512

1.4 Η ΟΙΚΟΓΕΝΕΙΑ XC9500 CPLD ΤΗΣ XILINX

Κάθε συσκευή XC9500 αποτελήται από πολλαπλές βαθμίδες λειτουργιών, FBs (Function Blocks) και πολλαπλές βαθμίδες εισόδων/εξόδων, IOBs (I\O Blocks) που συνδέονται μεταξύ τους με πλέγμα διακοπτών γρήγορης σύνδεσης (Fast CONNECT switch matrix).

Οι βαθμίδες εισόδων\εξόδων απομονώνουν τα εσωτερικά κυκλώματα του CPLD από τους ακροδέκτες εισόδου και εξόδου. Κάθε βαθμίδα λειτουργιών παρουσιάζει λογική και μπορεί και μπορεί να προγραμματιστεί και προσφέρει 36 εισόδους και 18 εξόδους. Τα πλέγματα διακοπτων γρήγορης σύνδεσης συνδέουν τις εξόδους των βαθμίδων λειτουργιών αλλά και τα εισερχόμενα στο CPLD σήματα, στις εισόδους των βαθμίδων λειτουργιών. Για κάθε βαθμίδα λειτουργιών, 12 έως 18 έξοδοι (ανάλογα από το CPLD) οδηγούνται στις βα8μίδες εισόδων\εξόδων. Στο Σχ1.2 παρουσιάζεται η αρχιτεκτονική της οικογένειας XC9500.



Σχ1.2 Αρχιτεκτονική XC9500

1.5 ΔΙΑΤΑΞΕΙΣ ΠΥΛΩΝ ΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΟΥ ΠΕΔΙΟΥ (FPGAs)

Ακόμη και στην περίπτωση των CPLDs, μόνο μετρίως μεγάλα λογικά κυκλώματα(100.000 πύλες) μπορούν να χωρέσουν σε ένα ολοκληρωμένο κύκλωμα. Με βάση τα σύγχρονα πρότυπα, ένα λογικό κύκλωμα με 100.000 πύλες δεν είναι μεγάλο. Για υλοποίηση μεγαλύτερων κυκλωμάτωνείναι βολικό να χρησιμοποιείται ένα διαφορετικό είδος ολοκληρωμένων κυκλωμάτων που έχει μεγαλύτερη λογική χωρητικότητα. Οι λογικές διατάξεις πυλών προγραμματιζόμενου πεδίου, FPGAs (Field-Programmable Gate Arrays), είναι διατάξεις προγραμματιζόμενης λογικής που υποστηρίζουν την υλοποίηση μεγάλων κυκλωμάτων, μεγαλύτερων από 2.000.000.000 πύλες. Οι διατάξεις FPGA διαφέρουν σημαντικά από τα CPLDs επειδή δεν περιέχουν πύλες AND και OR. Αντίθετα τα FPGAs περιέχουν λογικές βαθμίδες για την υλοποίηση των ζητούμενων συναρτήσεων. Στα FPGAs ο προγραμματισμός είναι πτητικός, το κύκλωμα που έχει σχεδιάσει χάνεται με τη διακοπή της τροφοδοσία του ολοκληρωμένου, κάτι το οποίο δεν συμβαίνει στα CPLDs. Κατασκευάζονται από διάφορες εταιρείες, όπως η Xilinx, η Atmel και η Altera και κυκλοφορούν σε διάφορες συσκευασίες όπως QFP,PGA και BGA, οιο ακροδέκτες του φτάνουν μέχρι και τους 500.

Η εταιρεία Xilinx προσφέρι μια πολύ μεγάλη ποικιλία από FPGAs που καλύπτουν τις ανάγκες κάθε σχεδιαστή. Αυτή την στιγμή η εν λόγο εταιρεία διαθέτει έξι οικογένειες FPGAs. Κάθε οικογένεια αποτελείται από μια σειρά ολοκληρωμένων κυκλωμάτων με κοινή αρχιτεκτονική και χαρακτηριστικά, η διαφορά τους βρισκεται στο μέγεθος του λογικού κυκλώματος που μπορουν να υλοποιήσουν. Σε κάθε ολοκληρωμένο κύκλωμα η χαρακτηριστική του κωδική ονομασία δηλώνει σε ποια οικογένεια ανήκει και πόσες βαθμίδες περιέχει. Πιο κάτω παρουσιάζοντια οι οικογένειες αυτές και τα ολοκληρωμένα κυκλώματά τους.

- SPARTAN / XL
- SPARTAN – II
- SPARTAN – IIE
- SPARTAN – 3
- SPARTA – 3E
- SPARTAN – 3A
- SPARTAN – 3AN
- SPARTAN – 3A DSP
- VIRTEX
- VIRTEX E

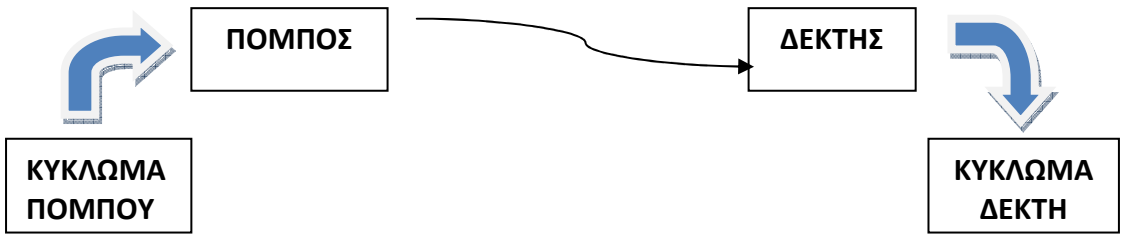
- VIRTEX EM
- VIRTEX – II Pro Platform
- VIRTEX – 4 Multi Platform
- VIRTEX – 5 Multi Platform

ΚΕΦΑΛΑΙΟ 2

ΔΟΜΗ ΑΣΥΡΜΑΤΟΥ ΕΛΕΓΧΟΥ

Ο ασύρματος έλεγχος είναι μια διαδικασία κατά την οποία γίνεται απομακρισμένος έλεγχος μιας σισκευής, στην περίπτοσή μας ένα αυτοκίνητο. Ο έλεγχος αυτός αποτελήται από δύο βασικά κυκλώματα, την επεξεργασεία και την αποστολή και λίψη των δεδομένων. Δηλαδή έχουμε μια μονάδα ασύγχρονου δέκτη-πομπού γενικής χρήσης (universal asynchronous receiver-transmitter, UART). Η μονάδα αυτή εκτελεί τις μετατροπές παράλληλου σε σειριακό και σειριακό σε παράλλλο που απαιτουν ο πομπός και δέκτης αντίστοιχα. Ένα τυπικό κύκλωμα UART είναι το ολοκληρωμένο κύκλωμα AY-5-1013 της εταιρίας General Instrument.

Το αυτοκίνητό μας θα αποτελήται από δύο κινητίρες. Η αριστερή και δεξιά κίνηση του αυτοκινήτου θα χίνεται με την αντίστροφη κίνηση τωμ δύο κινητίρων. Ο κάθε κινητίρας θα πρέπει να εκτελεί τρεις καταστάσεις, δεξιόστροφη κίνηση, αριστερόστροφη κίνηση και να παραμένει ακίνητος. Για τον έλεγχο λοιπόν του κάθε κινητίρα απαιτούνται 2 – bits, συνολικά 4 – bits για τον έλεγχο του αυτοκινήτου. Θα πρέπει λοιπόν να υλοποιήσουμε ένα 4 – bit UART για να μπορούμε να ελέγχουμε τις κινήσεις του αυτοκινήτου. Στο Σχ.2.1 φαίνεται το βασικό μοντέλο ενός ασύρματου ελέγχου αυτοκινήτου και αμέσως πιο κάτω περιγράφονται οι μονάδες από τις οποίες αποτελήται.



Στο Σχ.2.1 Βασικό μοντέλο ενός ασύρματου ελέγχου αυτοκινήτου

- **DATA.**
  Αποτελήται από το κύκλωμα που θα στέλνει τα δεδομένα για επεξεργασία και αποστολή.
- **ΚΥΚΛΩΜΑ ΠΟΜΠΟΥ.**
  Είναι το κύκλωμα στο οποίο θα γίνεται η μετατροπή των δεδομένων από παράλληλο σε σειριακό και στην συνέχεια θα στέλνονται στον πομπό.
- **ΠΟΜΠΟΣ.**
  Ο πομπός θα παίρνει τα δεδομένα που έχουν μετατραπεί σε σειριακά, θα τα διαμορφώνει κατάλληλα και θα τα εκπέμπει.
- **ΔΕΚΤΗΣ.**
  Ο δέκτης παίρνει το διαμορφωμένο σήμα, το αποδιαμορφώνει και το στέλνει έτοιμο προς επεξεργασία.
- **ΚΥΚΛΩΜΑ ΔΕΚΤΗ.**
  Στο κύκλωμα αυτό γίνεται η μετατροπή των δεδομένων από σειριακά σε παράλληλα, επεξεργάζονται και είναι ετοιμα να σταλθουν στους κινητήρες.
- **ΚΙΝΗΤΙΡΕΣ.**
  Εδώ λαμβάνονται τα σήματα και έπειτα μετά από κατάλληλη ενίσχυση και απομόνωση οδηγούνται στους κινητήρες για την κίνηση του οχήματος.

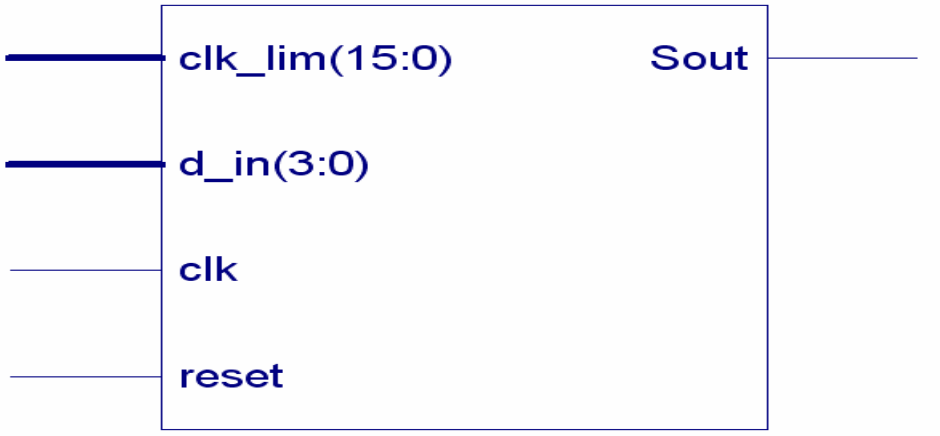
2.1 ΚΥΚΛΩΜΑ ΔΕΔΟΜΕΝΩΝ ΕΙΣΟΔΟΥ

Είναι ένα μικρό κύκλωμα το οποίο βρίσκεται πάνω στο τηλεχειριστήριο. Αποτελήται από τέσσερα κουμπιά (push button) τα

οποία είναι σε τέτεια διάτακση που αντιπροσοπεύουν μια από τις τέσσερις κινήσεις. Με το πάτημα κέθε κουμπιού στέλνονται τα κατάλληλα δεδομένα για την κίνηση του οχήματος και για την ενεργοποίηση το κύκλωμα επεξεργασείας. Αν πατηθεί πάνω από ένα κουμπί, δήλαδή το όχημα να κάνει ταυτόχρονα τουλάχιστον δύο κινήσεις, το όχημα θα παραμήνει ακίνητο μέσω προγράμματος το οποίο βρίσκετε στο κύκλωμα του δέκτη.

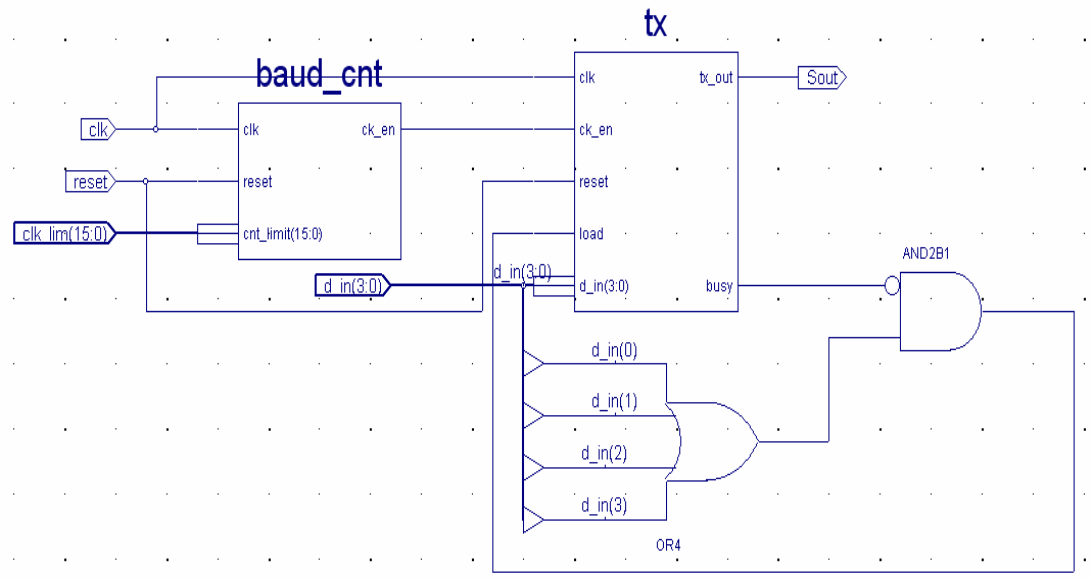
2.2 ΚΥΚΛΩΜΑ ΠΟΜΠΟΥ

Όπως έχουμε αναφερθεί και προηγουμένος στο κύκλωμα αυτό θα γίνεται η επεξεργασεία των πληροφοριών και η αποστολή τους προς τον πομπό. Όπως φαίνεται και στο Σχ.2.2 αποτεληται από τέσσερις εισόδους και μία έξοδο.



Σχ2.2 ΜΠΛΟΚ ΔΙΑΓΡΑΜΜΑ ΚΥΚΛΩΜΑΤΟΣ ΠΟΜΠΟΥ

Η εξοδό του, Sout, είναι σειριακή η οποία θα συνδαίεται στον πομπό για αποστολή των πληροφοριών. Το reset χρισημοποιήται για να γίνη η εκκίνηση από κάποια προκαθορισμένη αρχική κατάσταση. Στην είσοδο clk διοχετεύσαμε ένα τετραγωνικό σήμα στην αντίστοιχη είσοδο του CPLD. Η d_in(3:0) είναι μία 4 – bit είσοδο η οποία αντιστηχεί στις πληροφορείες εισόδου για την κίνηση του οχήματος. Τέλος η είσοδος clk_lim(15:0) αντιπροσοπεύει τον αρθμό με τον οποίο θα γίνει η διαίρεση με την αρχική συχνότητα για να πετύχουμε το σωστό data rate του πομπού και δέκτη. Στο Σχ.2.3 φαίνοντια τα επιμέρους κυκλώματα του δέκτη.



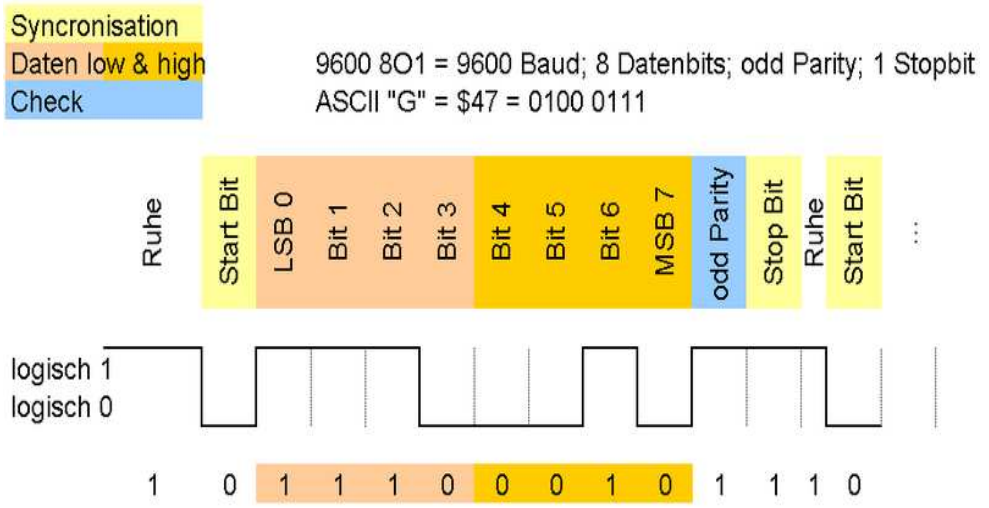
Σχ.2.3 ΕΠΙΜΕΡΟΥΣ ΚΥΚΛΩΜΑΤΑ ΤΟΥ ΠΟΜΠΟΥ

Τα κυκλώματα από τα οποία αποτελήται ο δέκτης είναι:

- **Μετρητης δημιουργείας clk_en.**
  Ο μετριτής αυτός αποτελήται από μια είσοδο clock που είναι το global clock (25MHz), από μία είσοδο reset και από έναν δεκαεξάμπιτο αριθμό εισόδου. Ο μετριτής κάνει την διαίρεση του clock με τον δεκαεξάμπιτο αριθμό (clk_lim(15:0)) και κάθε φορά που τελειώνει η διαίρεση βγάζει στην έξοδο clk_en έναν παλμό. Ο παλμός αυτός θα αποτελέσει το clock για το data rate του πομπού και δέκτη. Το data rate του πομπού είναι 4kHz και του δέκτη είναι 2kHz. Άρα κοινό data rate έχουμε τα 2kHz. Για να μπορέσουμε να βρούμε τον δεκαεξάμπιτο αριθμό θα πρέπει να διαιρέσουμε το global clock με το data rate (25MHz÷2kHz). Ο αριθμός που προκίπτει είναι $(30D4)_{16}$ ή $(0011000011010100)_2$.

- **Μονάδα επεκσεργασείας δεδομένων, tx.**
  Η μονάδα αυτή αποτελήται από μια τετράμπιτη είσοδο δεδομένων (δεδομένα για την κίνηση του οχήματος), από την είσοδο load στην οποία διλώνουμε πότε να φορτώσει τα

δεδομένα και από το clk_en που θα αποτελέσει το clock για το data rate του πομπού και δέκτη. Τέλος σαν είσοδο έχει το reset και το global clock. Το κύκλωμα αποτελήται και από δύο εξόδους, την busy και την tx_out. Η busy νας ενημερώνει για το αν το κύκλωμα βρίσκεται σε λιτουργεία ή όχι και αν μπορούμε να φορτώσουμε τις επόμενες πληροφορίες. Τέλος στην tx_out έχουμε την έξοδο της πληροφορίας πλέον σειριακά και όχι παράλληλα. Στην πληροφορία αυτή έχει προστεσθεί στην αρχή και στο τέλος ένα start_bit και ένα stop_bit. Αυτό έγινε γιατί ο έλεγχος του ασύρματου οχήματος είναι ασίχρονος, θα έπρεπε με κάπιο τρόπο να αντιληφθεί ο δέκτης το πότε και πού σταματάει η πληροφορεία. Κάτι αντοίστηχο όπως συμβαίνει και στο προτόκολο RS232. Ένα τέτοιο παράδιγμα φαίνετε στην παρακάτω εικώνα.

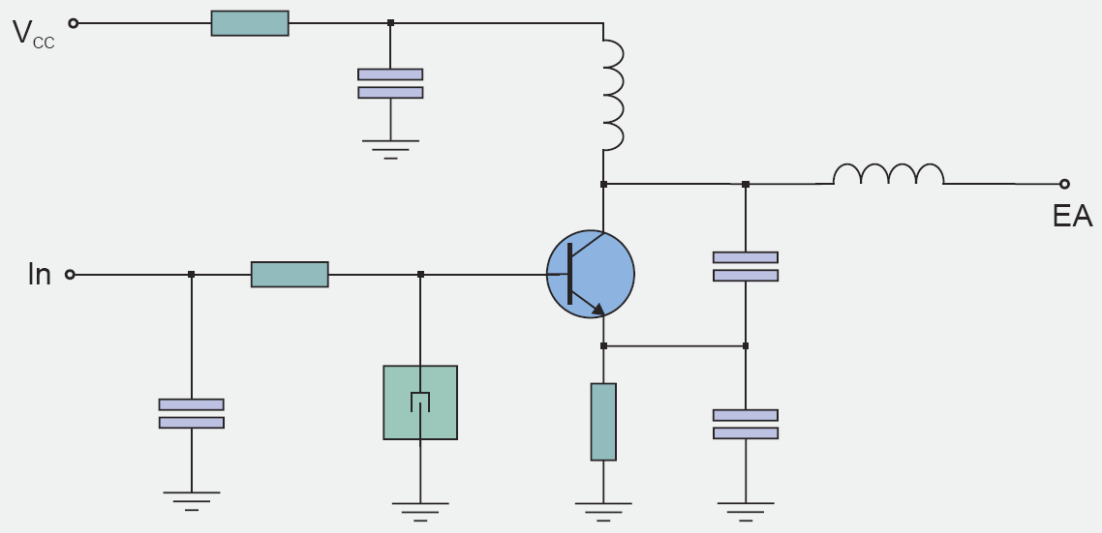


Σχ.2.4 ΑΣΥΓΧΡΟΝΗ ΕΚΠΟΜΠΗ ΔΕΔΟΜΕΝΩΝ

Τέλος όπως φαίνεται και στο Σχ.2.3 έχουμε χρισήμοποιήσει μία πύλη OR τεσσάρων εισόδων και μια πύλη AND δύο εισόδων και μιας αναστροφής. Στις εισόδους της OR έχουμε συνδέσει τις πληροφορείες που στέλνουμε από τα push button και την έξοδό της στην είσοδο της πύλης OR. Την αναστρέφον είσοδο της πύλης AND την συνδέσαμε στην έξοδο busy του tx και την έξοδό της στην είσοδο load. Με αυτόν τον τρόπο πετίχαμε την μη λειτουργεία του tx όταν δεν πατιέται κανένα πλήκτρο, καθώς επίσης και το φόρτομα των πληροφοριών μόνο όταν το tx έχει ολοκληρώσει την προηγούμενη διεργασεία.

2.3 ΠΟΜΠΟΣ

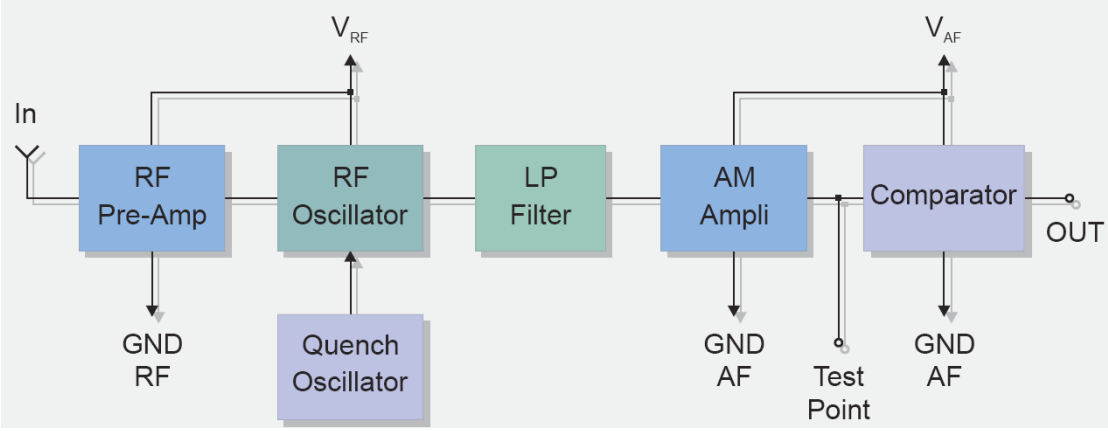
Στον ασύρματο έλεγχο αυτοκινήτου χρησιμοποιήσαμε για πομπό τον RT4 – 433.92. Είναι ένα ιβιδρικό κύκλωμα που το κύκλωμά του φαίνεται στο Σχ.2.5. Αποτελήται από τέσσερα ποδαράκια, έχει μία σειριακή είσοδο όπου λαμβάνει τις πληροφορίες, τις διαμορφώνει και τις βγάζει από μια έξοδο η οποία είναι έτοιμη να οδηγηθεί σε μια καιρέα και να εκπεμπσει τις πληροφορείες. Ο πομπός δουλεύει σε συχνότιτα 433.92MHz, τροφοδοτήται με τάση 2 – 14VDC και έχει μέγιστο data rate 4kHz.



Σχ.2.5 ΣΧΗΜΑΤΙΚΟ ΚΥΚΛΩΜΑ ΠΟΜΠΟΥ

2.4 ΔΕΚΤΗΣ

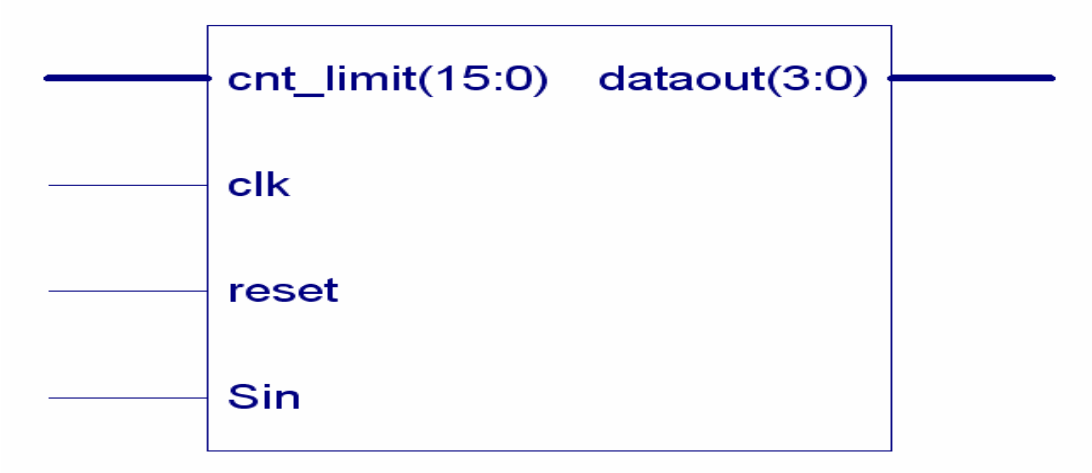
Όπως και στον πομπό έτσι και στον δέκτη χρισιμοποιήσαμε ένα ιβιδρικό κύκλωμα με όνομα RR3 – 433.92. Λαμβάνει το διαμορφομένο σήμα από μια καιρέα, το αποδιαμορφώνει και στην συνέχεια το βγάζει από μια σειριακή έξοδο έτοιμο για επεξεργασεία. Δουλεύει σε συχνότιτα 200 - 450MHz, τροφοδοτήται με τάση 4,5 – 5,5VDC, έχει μέγιστο data rate 2kHz και RF ευαισθησεία -100 - -105dBm. Στο παρακάτω σχήμα φαίνεται το μπλοκ διάγραμμά του.



Σχ.2.6 ΜΠΛΟΚ ΔΙΑΓΡΑΜΜΑ ΔΕΚΤΗ

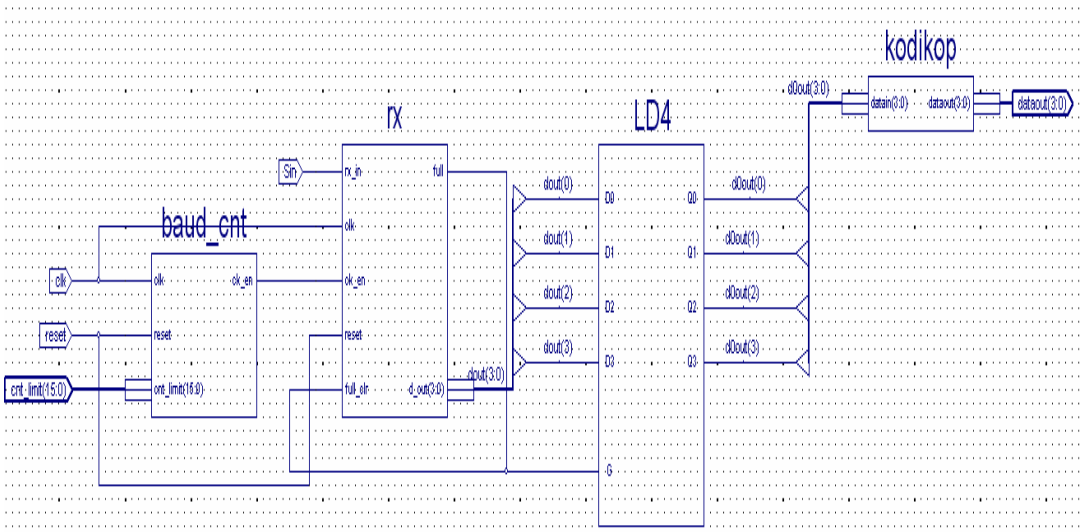
2.5 ΚΥΚΛΩΜΑ ΔΕΚΤΗ

Στο κύκλωμα αυτό θα γίνεται η λήψη και η επεξεργασεία των πληροφοριών καθώς και η αποστολή τους προς το κύκλωμα των κινητίρων. Όπως φαίνεται και στο Σχ.2.7 αποτεληται από τέσσερις εισόδους και μία έξοδο.



Σχ2.7 ΜΠΛΟΚ ΔΙΑΓΡΑΜΜΑ ΚΥΚΛΩΜΑΤΟΣ ΔΕΚΤΗ

Η εξοδό του, dataout(3:0), είναι παράλληλη και μετά από κατάλληλη ενύσχηση και απομόνωση από το υπόλιπο κύκλωμα, θα οδηγηθεί στους κινητίρες. Το reset χρισημοποιήται για να γίνη η εκκίνηση από κάποια προκαθορισμένη αρχική κατάσταση. Στην είσοδο clk διοχετεύσαμε ένα τετραγωνικό σήμα στην αντίστοιχη είσοδο του CPLD. Η Sin είναι μία σειριακή είσοδος που λαμβάνει τις πληροφορείες αποδιαμορφομένες και έτοιμες πρως επεξεργασεία από τον δέκτη. Τέλος η είσοδος cnt_limit(15:0) αντιπροσοπεύει τον αρθμό με τον οποίο θα γίνει η διαίρεση με την αρχική συχνότητα για να πετύχουμε το σωστό data rate του πομπού και δέκτη. Στο Σχ.2.8 φαίνοντια τα επιμέρους κυκλώματα του δέκτη.



Σχ.2.8 ΕΠΙΜΕΡΟΥΣ ΚΥΚΛΩΜΑΤΑ ΤΟΥ ΔΕΚΤΗ

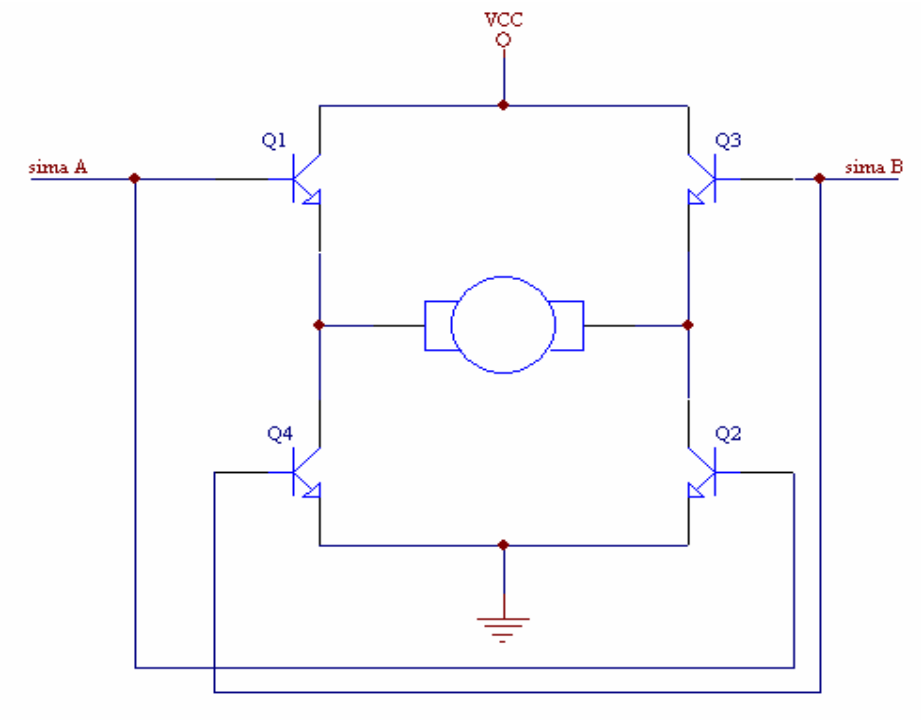
Τα κυκλώματα από τα οποία αποτελήται ο πομπός είναι:

- **Μετρητης δημιουργείας clk_en.**
  Ο μετριτής αυτός αποτελήται από μια είσοδο clock που είναι το global clock (25MHz), από μία είσοδο reset και από έναν δεκαεξάμπιτο αριθμό εισόδου (cnt_limit(15:0)). Ο μετριτής κάνει την διαίρεση του clock με τον δεκαεξάμπιτο αριθμό και κάθε φορά που τελειώνει η διαίρεση βγάζει στην έξοδο clk_en έναν παλμό. Ο παλμός αυτός θα αποτελέσει το clock για το data rate του πομπού και δέκτη. Ο δεκαεξάμπιτος αριθμός που προκίπτει όπως υπολογίσαμε και προηγουμένος είναι ο $(30D4)_{16}$ ή $(0011000011010100)_2$.
- **Μονάδα επεκσεργασείας δεδομένων, rx.**
  Η μονάδα αυτή αποτελήται από μια σειριακή είσοδο δεδομένων (δεδομένα μετά την λήψη κα αποδιαμόρφωση από τον δέκτη), από την είσοδο full_clr στην οποία δηλώνουμε κάθε πότε θέλουμε να ξεκιμάει η επεξεργασελεία του επόμενου πακέτου πληροφοριών και από το clk_en που θα αποτελέσει το clock για το data rate του πομπού και δέκτη. Τέλος σαν είσοδο έχει το reset και το global clock. Το κύκλωμα αποτελήται και από δύο εξόδους, την full και την dout(3:0). Η full μας ενημερώνει για το πότε το κύκλωμα βρίσκεται σε λειτουργεία και το ποτε έχει τελειόσει την επεκσεργασεία των δεδομένω. Τέλος στην rx_out έχουμε την έξοδο της πληροφορίας πλέον παράλληλα και όχι σειριακά. Στην πληροφορία αυτή έχει αφαιρεθεί από την αρχή και το τέλος το start_bit και το stop_bit.

- **Μανδαλωτής, (latch) LD4.**
  Ο μανδαλωτής είναι ένα κύκλωμα, το οποίο κρατάει τις τιμές στις εξόδους του αμετάβατες ότι και αν συμβεί στις εισόδους του, αν δεν έχει ενεργοποιηθεί το gate enable (G). Όταν ενεργοποιηθεί το gate enable τότε οι τιμές τις εισόδου μεταφέρονται στην έξοδο. Όπως φαίνεται και στο Σχ. 2.8 έχουμε συνδέσει την έξοδο full του rx στην είσοδο full_clr και στην είσοδο G του μανδαλωτή. Επίσης συνδέσαμε την έξοδο d_out(3:0) του rx στην είσοδο του μανδαλωτή. Με αυτό τον τρόπο πετύχαμε τη μεταφορά των πληροφοριών στην έξοδο του μανδαλωτή κάθε φορά που τελειώνει ο rx την επεξεργασία των πληροφοριών. Επίσης προστατεύουμε τις πληροφορίες στην έξοδο του μανδαλώτη από ότι παρεμβολές και να γίνουν στην εισοδό του μανδαλωτή, αφού είναι γνοστό ότι λειτουργεί και ως στοιχείο μνήμης.
- **Κωδικοποιητής.**
  Οι πληροφορίες που βρίσκονται στην είσοδο του κωδικοποιητή είναι αυτές που στάλθικαν από το κύκλωμα του πομπού έπειτα από κατάληλη επεξεργασία. Με λίγα λόγια είναι οι πληροφορίες που στάλθικαν από τα push buttons του τηλεχειρηστηρίου. Σε αυτό το κύκλωμα κωδικοποιούνται τα δεδομένα με τέτοιο τρόπο ώστε η έξοδος του κωδικοποιητή να κινήσειι, μετά από κατάλληλη ενισχισή και απομώνοση, τους κινητίρες του οχήματος προς την σωστή κατεύθινση. Δηλαδή η κίνηση του κινητίρα να αντιστιχεί στην σωστή ένδιξη κίνησης του αντίστοιχου πλήκτρου.
  Τέλος το κύκλωμα αυτό ελέγχει αν έχει πατηθεί περισσότερα από ένα πλήκτα. Αν συμβεί αυτό το όχημα θα παραμίνει ακίνητο.

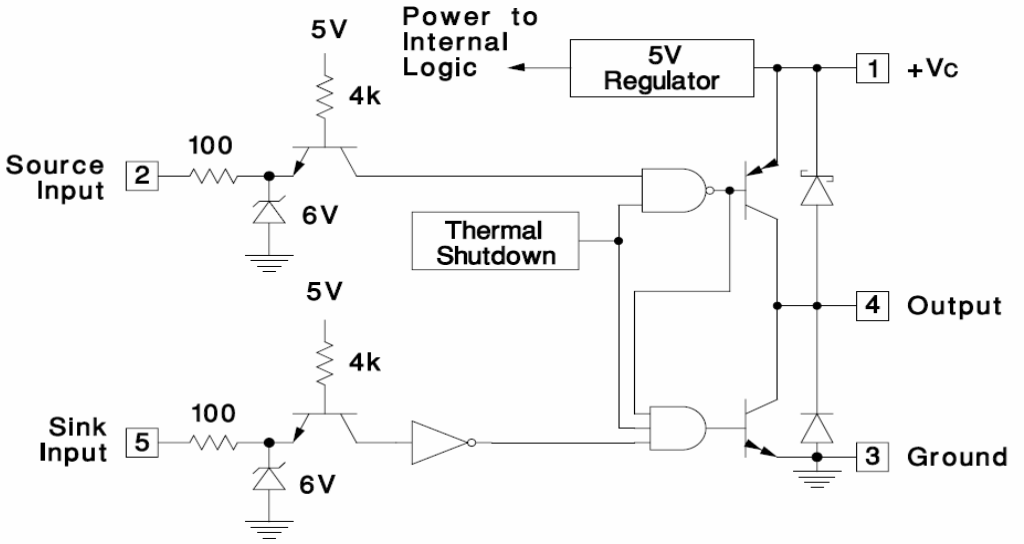
2.6 Κινητίρες.

Στο κύκλωμα αυτό έχουμε απομονώσει και ενισχήση τα σήματα που θα οδηγήσουν τους κινητίρες. Όπως έχουμε πει, για κάθε κινητήρα θα χριαστούμε δύο σήματα για τον έλεγχο της κίνισής τους. Αυτό πετιχένεται με την χρίση μιας γέφυρας «Η» για κάθε κινητίρα. Για να το πετίχουμε αυτό χρισιμοποιήσαμε δύο Half – Bridfe Bipolar Switch. Ένα μπλοκ διάγραμμα για τον κάθε κινητίρα φάινεται παρακάτω.



Σχ.2.9 ΜΛΟΚ ΔΙΑΓΡΑΜΜΑ ΓΕΦΥΡΑΣ «Η»

Έτσι ανάλλογα με τις πληροφορίες που θα έχουμε στα σήματα Α και Β, οκινητίρας θα περιστρέφετε προς την αντίστιχη κατεύθηνση. Ο Half – Bridfe Bipolar Switch που χρισιμοποιήσαμε ινε ο UC2950 της Texas Instruments. Είναι συμβατό με TTL και CMOS τεχνολογεία, έχουν ρευμα εισόδου πηγής 4^{A}, τάση τροφοδοσίας μέχρι 35V, υψιλό ρεύμα εξόδου, θερμική προστασία και δουλεύουν σε συχνότιτα μέχρι και 3000kHz. Στο παρακάτω σχήμα φαίνεται το σχηματικό του.



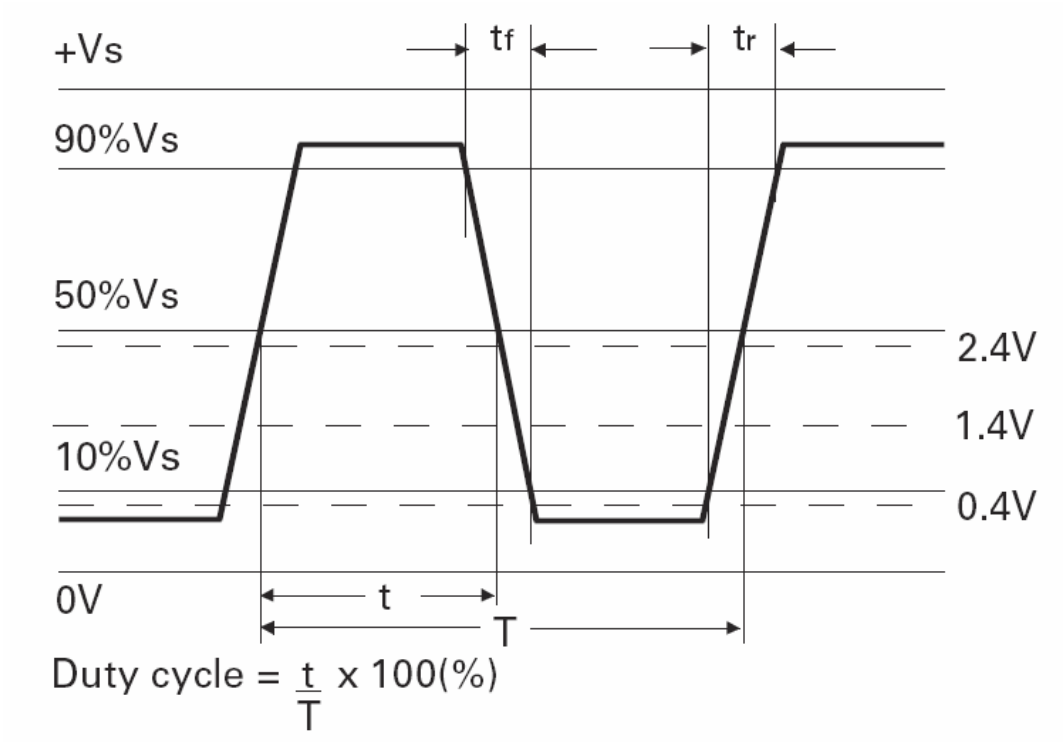
Σχ.2.10 ΣΧΗΜΑΤΙΚΟ ΤΟΥ UC2950

Οι κινητήρες που θα χρισιμοποιήσουμε είναι της Lego Technic Motors και φαίνονται στο Σχ,2.11. Οι κινητίρες αυτοί τροφοδοτούνται με τάση 9VDC, μέγιστο ρεύμα που τραβάνε είναι 250mA, μέγιστη ταχύτητα χωρίς φορτίο είναι 350rpm και μέγιστη ταχύτητα υπό φυσιολογικό φορτίο είναι 200rpm.

Σχ.2.11 ΚΙΝΗΤΙΡΑΣ LEGO

2.7 ΧΡΟΝΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

Όλες οι εντολές εκτελούνται σε ένα κύκλο εργασίας. Ένας κύκλος εργασίς αντιστοιχεί σε ένα κύκλο του ρολογιού του συστήματος. Ο χρόνος διάρκειας ενός κύκλου εργασείας εξαρτάτε δηλαδή από την συχνότιτα του τετραγωνικού σήματος που συνδέεται στην είσοδο CLK. Το τετραγωνικό αυτό σήμα διοχετεύεται από έναν κρύσταλο στην αντίστοιχη είσοδο του CPLD. Φυσικά υπάρχει ένα ανώτερο όριο στην συχωότητα του σήματος αυτού το οποίο καθορίζεται από το CPLD για κατασκευαστικούς λόγους, για παράδειγμα στην οικογένεια XC9500 της Xilinx η συχνότυτα αυτή φτάνει μέχρι τα 125MHz. Ο κρύσταλος που χρισιμοποιήσαμε είναι ο iqxo – 350 της MicroTechnology, δουλεύει στην συχνότιτα των 25MHz και ο τετραγωνικός παλμός εξόδου του φαίνετε στο παρακάτο σχήμα. Ο iqxo – 350 είναι συμβατός με HCMOS/TTL/LSTTL τεχνολογελιες. Έχει τάση τροφοδοσείας 5V και τραβάει ρεύμα 40mA.



Σχ.2.12 ΤΕΤΡΑΓΩΝΙΚΟΣ ΠΑΛΜΟΣ ΤΟΥ IQXO - 350

2.8 ΕΠΑΝΑΤΟΠΟΘΕΤΗΣΗ (RESET)

Το σύστημα επανατοποθέτησης εξασφαλίζει το γεγονός ότι κατά την εφαρμογή της τροφοδοσίας οι μετρητές, οι μανδαλωτές και οι καταχωριτές , βρίσκονται σε κατάλληλες αρχικές τιμές. Η επανατοποθέτηση επιτυχάνεται επίσης από εξωτερικό ακροδέκτη του κυκλώματος από τον χρίστη όποτε έιναι αυτό επιθυμητό.

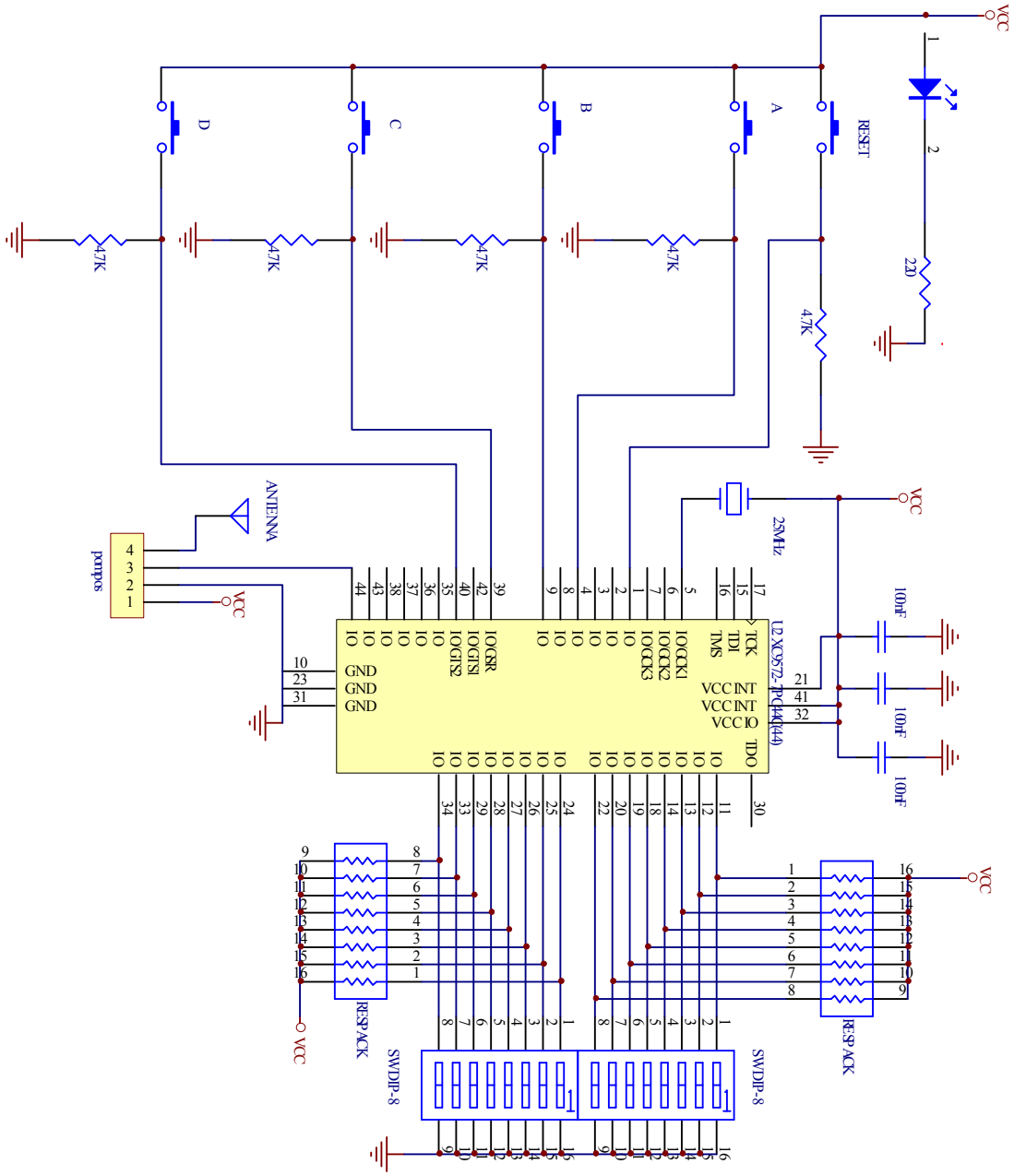
ΚΕΦΑΛΑΙΟ 3

ΑΝΑΠΤΥΞΗ ΑΣΥΡΜΑΤΟΥ ΕΛΕΓΧΟΥ

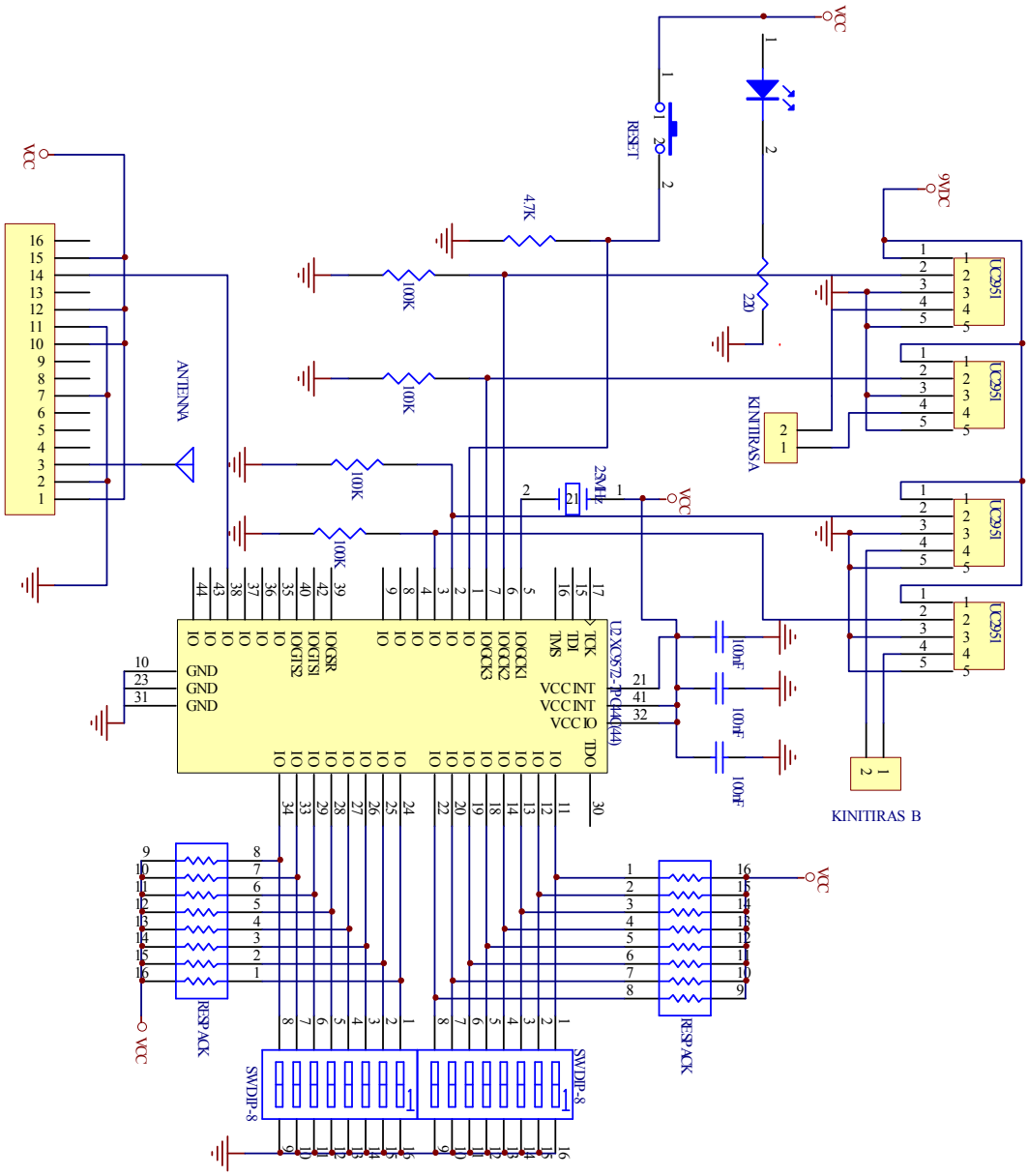
Ο ασύρματος έλεγχος του αυτοκινήτου αποτελήται όπψς έχουμε αναφερθεί και προηγουμέωος σε δύο μεγάλλα και βασικά κυκλώματα. Στο κύκλωμα του πομπύ και στο κύκλωμα του δέκτη. Και τα δύο κυκλώματα είναι τετράμπιτα, όσα δηλαδή και τα κουμπιά επιλογής κατεύθηνσης (οι τέσσερεις αντίστοιχες κινήσεις του οχήματος. Οι μονάδες από τις οποίες αποτελούνται και τα δύο κυκλώματα είναι:

- Μετρητης δημιουργείας clk_en
- Μονάδα επεκσεργασείας δεδομένων, tx
- Μονάδα επεκσεργασείας δεδομένων, rx
- Μανδαλωτής, (latch) LD4
- Κωδικοποιητής

Όλες οι παραπάνω μονάδες υλοποιούνται με την βοήθεια δύο CPLDs, ένα για κέθε κύκλωμα αφού τα κυκλώματ6α είναι ανεξάρτητα μεταξύ τους, των XC9572 της Xilinx. Τα σχηματικά διαγράμματα του πομπού και του δέκτη δίνονται στα σχήματα 3.1 και 3.2 αντίστοιχα.



Σχ.3.1 ΣΧΗΜΑΤΙΚΟ ΔΙΑΓΡΑΜΜΑ ΠΟΜΠΟΥ



Σχ.3.2 ΣΧΗΜΑΤΙΚΟ ΔΙΑΓΡΑΜΜΑ ΔΕΚΤΗ

3.1 ΑΝΑΠΤΥΞΗ ΚΥΚΛΩΜΑΤΟΣ ΠΟΜΠΟΥ ΣΤΟ CPLD XC9572 ΤΗΣ XILINX

Το XC9572 της Xilinx ανήκει στην οικογένεια XC9500 της εν λόγο εταιρείας. Διαθέτει 72 ακροδέκτες εισόδων\εξόδων (I\O Pins), 72 μακροκυψέλες και 1.600 χρησιμοποιήσιμες πύλες, αυτό σημαίνει ότι μπορεί να υλοποιήσει ένα λογικό κύκλωμα ισοδύναμο των 1.600 πυλών. Μπορεί να επαναπρογραματιστεί 10.000 φορές, κρατάει τα δεδομένατου για 20 χρόνια και προσφέρεται στις συσκευασίες PLCC, PQFP και PQFP. Η σχεδίαση του κυκλώματος πομπού και ο προγραμματισμός του ολοκληρωμένου αυτού κυκλώματος γίνεται με το πρόγραμμα ISE 7.1i της Xilinx.

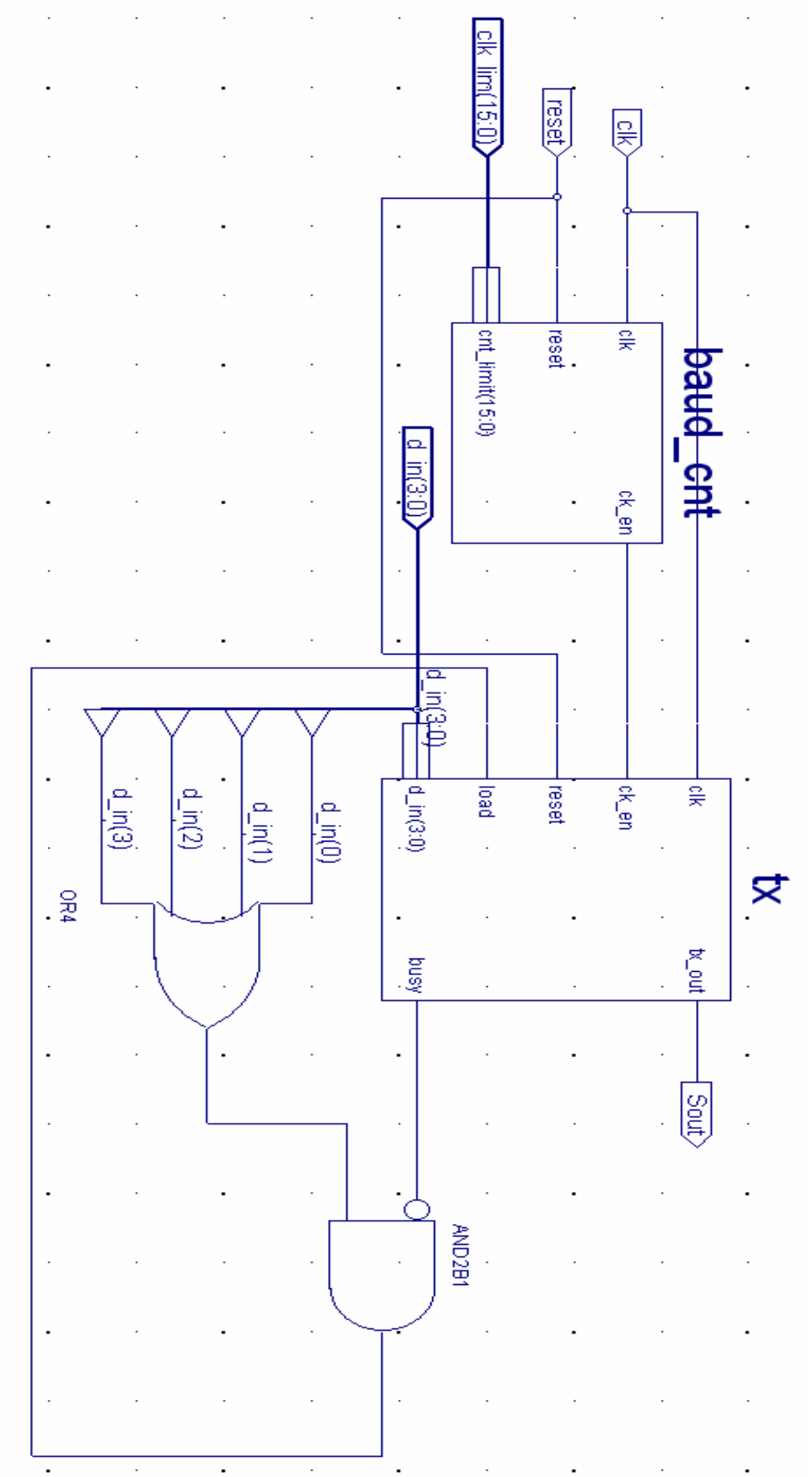
Ανάλογα από τη συσκευασία που επιλέγεται για την ανάπτυξη του κυλώματος πομπού, αλλάζει η θέση όλων των ακροδεκτών, εισόδου/εξόδου (I\O), προγραμματισμού (TDI, TDO, TCK, TMS) και τροφοδοσίας (VCCIO, VCCINT, GND). Για αυτό το λόγω, ανάλογα με τη συσκευασία του XC9572,αλλάζουν και οι ακροδέκτες του στο σχηματικό διάγραμμα του κυκλώματος πομπού. Οι ακροδέκτες εισόδων/εξόδων του ολοκληρωμένου κυκλώματος που χρησιμοποιούνται για το κύκλωμα του πομπού επιλέγονται από το εργαλείο ChipViewer του προγράμματος ISE 5. 1i.

Το κύκλωμα του πομπού συμπτύχθηκε στο σύμβολο pompos, σχήμα 3.3. Μπορεί να χρησιμοποιηθεί σαν το τελικό κύκλωμα που θα προγραμματιστεί σε ένα CPLD ή να χρησιμοπιηθεί για την σύνθεση άλλων κυκλωμάτων. Η σύνδεση των στοιχείων του κυκλώματος του πομπού (top level module) έγινε με χρήση σχηματικού (κυκλωματική σύνδεση) και όχι με τη γλώσσα VHDL. Τα στοιχεία που συνθέτουν το κύκλωμα πομπου υλοποιούνται είτε μα χρήση της γλώσσας προγραμματισμού VHDL είτε με τη κυκλωματική σύνδεση άλλων πιο απλών στοιχείων που και αυτά με τη σειρά τους υλοποιούνται από άλλα στοιχεία.

Σχ3.3 ΜΠΛΟΚ ΔΙΑΓΡΑΜΜΑ ΚΥΚΛΩΜΑΤΟΣ ΠΟΜΠΟΥ

Η εξοδό του, Sout, είναι σειριακή η οποία θα συνδαίεται στον πομπό για αποστολή των πληροφοριών. Το reset χρισημοποιήται για να γίνη η εκκίνηση από κάποια προκαθορισμένη αρχική κατάσταση. Στην είσοδο clk διοχετεύσαμε ένα τετραγωνικό σήμα στην αντίστοιχη είσοδο του CPLD. Η d_in(3:0) είναι μία 4 – bit είσοδο η οποία αντιστηχεί στις πληροφορείες εισόδου για την κίνηση του οχήματος. Τέλος η είσοδος clk_lim(15:0) αντιπροσοπεύει τον αρθμό με τον οποίο θα γίνει η διαίρεση με την αρχική συχνότητα για να πετύχουμε το σωστό data rate του πομπού και δέκτη. Στο Σχ.3.4 φαίνεται το κύκλωμα που υπάρχει στο εσωτερικώ του συμβόλου αυτύ, δήλαδή το σχηματικό διάγραμμα του πομπού. Αμέσως πιο κάτω παρουσιάζονται τα κύρια στοιχεία που υλοποιού το κύκλωμα του πομπού.

- Μετρητης δημιουργείας clk_en.
- Μονάδα επεκσεργασείας δεδομένων, tx.



Σχ.3.4 ΕΠΙΜΕΡΟΥΣ ΚΥΚΛΩΜΑΤΑ ΤΟΥ ΠΟΜΠΟΥ

Final Results

RTL Top Level Output File Name : pompos.ngr
Top Level Output File Name : pompos
Output Format : NGC
Optimization Goal : Speed
Keep Hierarchy : YES
Target Technology : xc9500
Macro Preserve : YES
XOR Preserve : YES
wysiwyg : NO

Design Statistics

 # IOs : 23

Macro Statistics :

 # Registers : 67
 # 1-bit register : 67
 # Xors : 21
 # 1-bit xor2 : 21

Cell Usage :

 # BELS : 293
 # AND2 : 96
 # AND3 : 5
 # AND8 : 2
 # INV : 134
 # OR2 : 35
 # OR3 : 1
 # OR4 : 1
 # XOR2 : 19
 # FlipFlops/Latches : 31
 # FD : 1
 # FDC : 23
 # FDP : 7
 # IO Buffers : 23
 # IBUF : 22
 # OBUF : 1
 # Others : 1
 # AND2B1 : 1

Σχ.3.5 ΑΝΑΦΟΡΑ ΣΥΝΘΕΣΗΣ ΤΟΥ ΚΥΚΛΩΜΑΤΟΣ ΠΟΜΠΟΥ

RESOURCES SUMMARY

Macrocells Used	Pterms Used	Registers Used	Pins Used	Function Block Inputs Used
31/72 (44%)	97/360 (27%)	31/72 (44%)	23/34 (68%)	68/144 (48%)

PIN RESOURCES

Signal Type	Required	Mapped	Pin Type	Used	Total
Input	20	20	I/O	21	28
Output	1	1	GCK/IO	1	3
Bidirectional	0	0	GTS/IO	0	2
GCK	1	1	GSR/IO	1	1
GTS	0	0			
GSR	1	1			

GLOBAL RESOURCES

Signal mapped onto global clock net (GCK1)	clk
Signal mapped onto global output enable net (GSR)	reset

POWER DATA

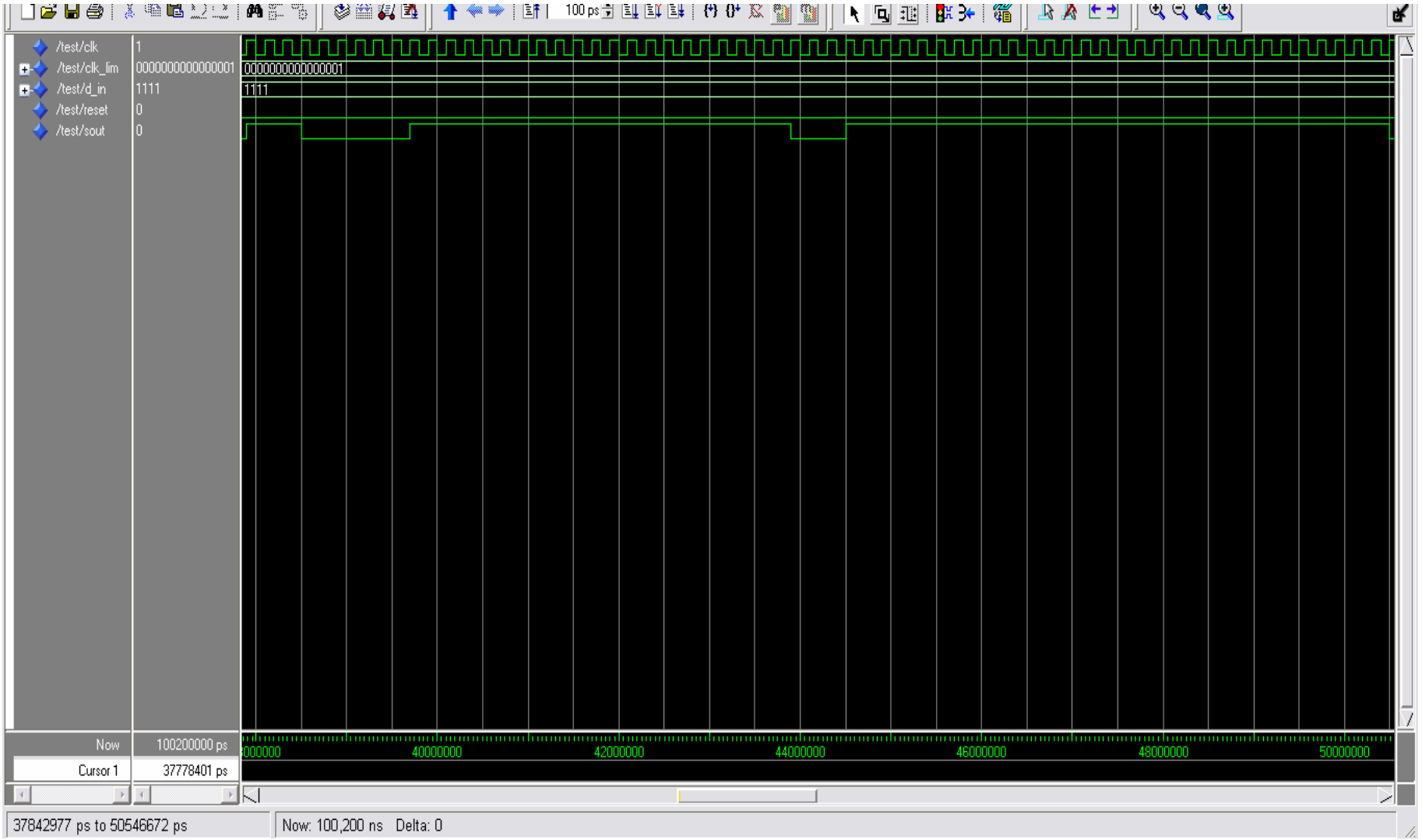
Macrocells in high performance mode (MCHP)	31
Macrocells in low power mode (MCLP)	0
Total macrocells used (MC)	31

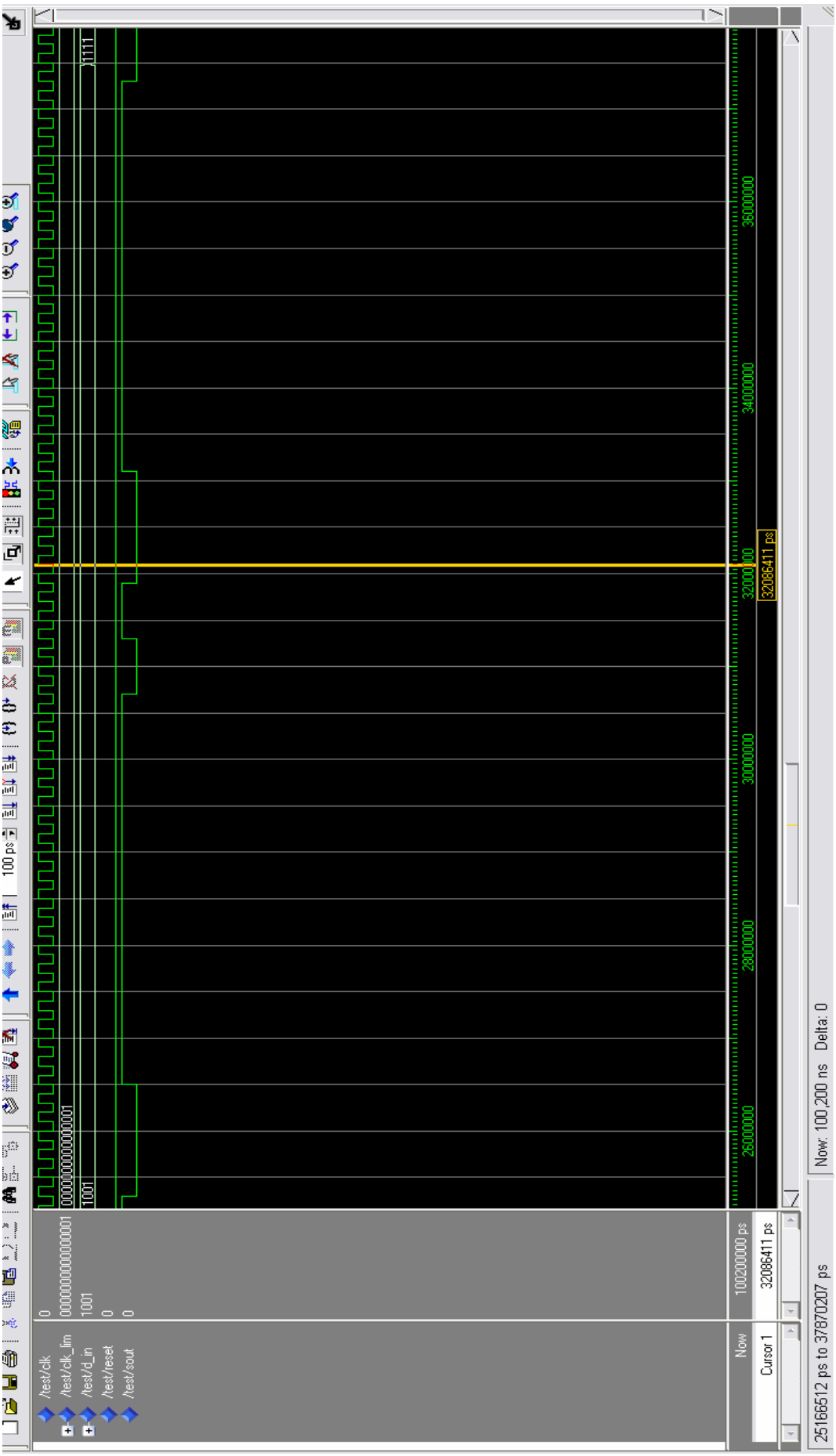
Σχ.3.6 ΑΝΑΦΙΡΑ ΠΟΡΩΝ ΤΟΥXC9572 ΠΟΥ ΚΑΤΑΛΑΜΒΑΝΕΙ ΤΟ ΚΥΚΛΩΜΑ ΠΟΜΠΥ

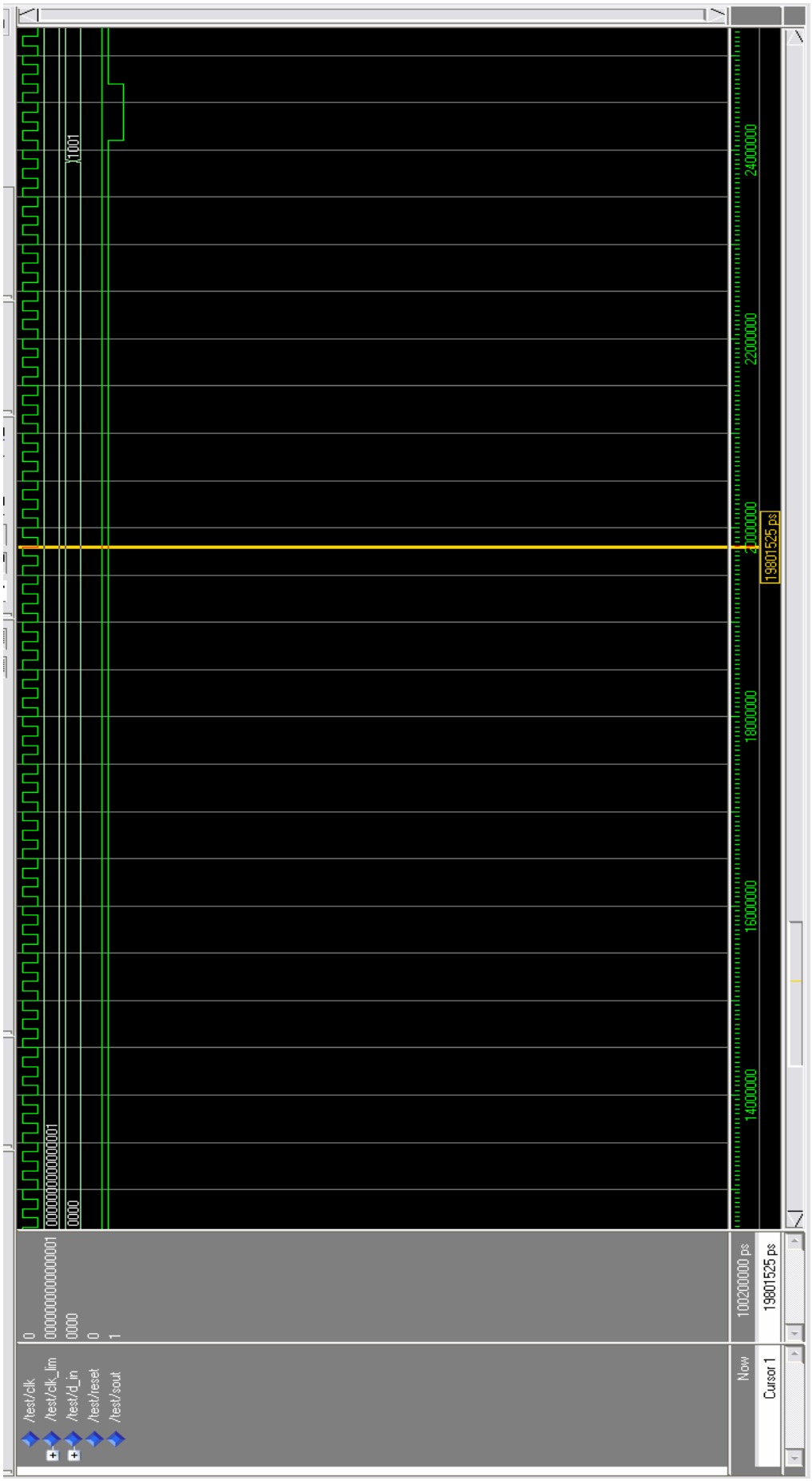
Στο σχήμα 3.5 παραιτίθεται η αναφορά της σύνθεσης του κυκλώματος του πομπού (synthesis report). Η αναφορά αυτή δίχνει την αντιστοιχία του κυκλώματος σε λογικές πύλες. Στο σχήμα 3.6 παρουσιάζεται η αναφορά των πόρων του XC9572 που καταλαμβάνει ο πομπός, δηλαδή ο αριθμός των μακροκυψέλων (macrocells), των ακροδεκτών (I\O Pins), των βαθμίδων λειτουργιών (Function Blocks) κ.τ.λ.. Οι δύο αναφορές αντλήθικαν από το κύκλωμα του πομπού που υλοποιήθικε αποκλειστικά με τη σχηματική μέθοδο.

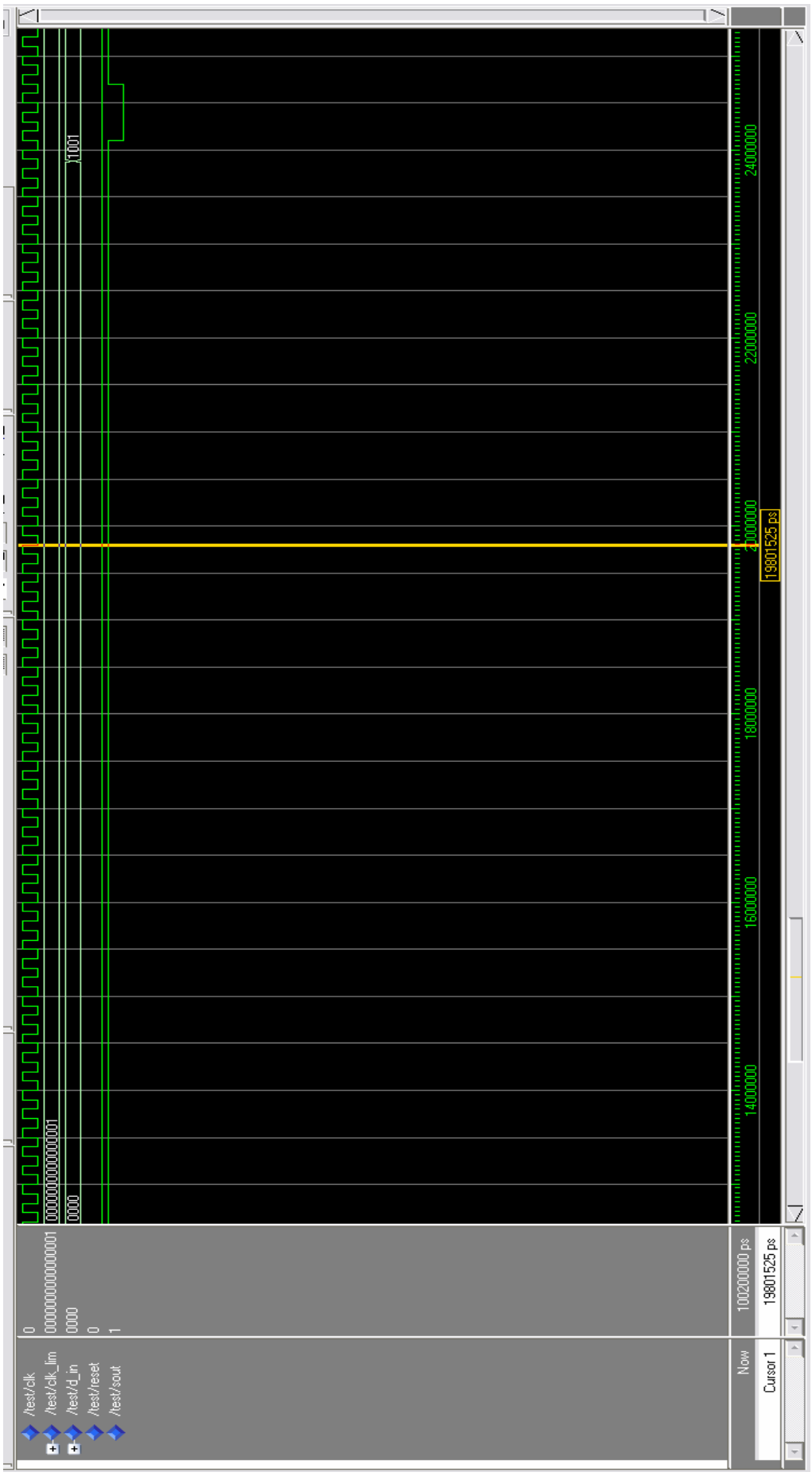
3.2 ΠΡΟΣΟΜΟΙΩΣΗ ΚΥΚΛΩΜΑΤΟΣ ΠΟΜΠΟΥ ΜΕ ΤΟ MODELSIM

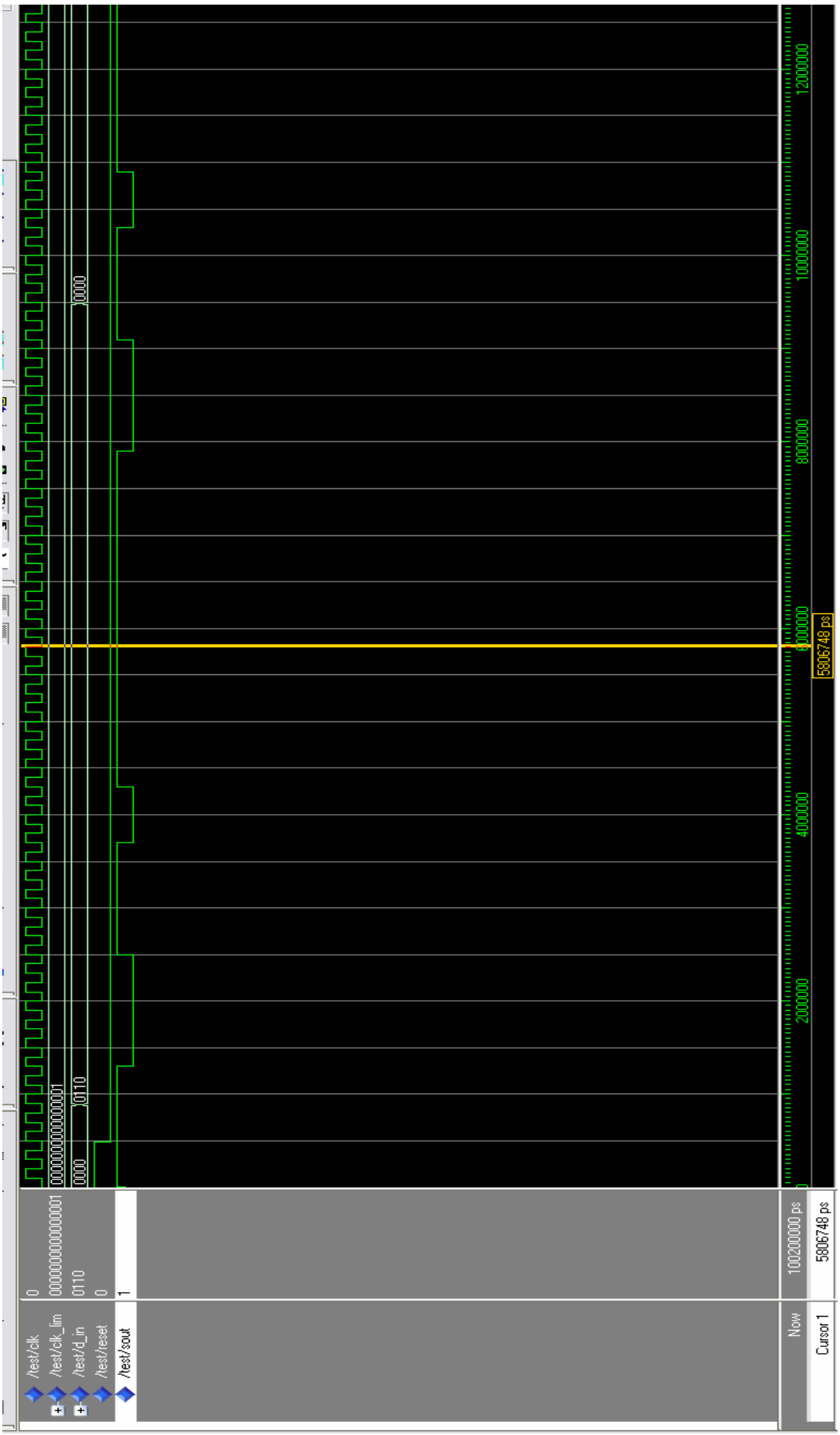
Αφού ολοκληρωθούν όλες οι διαδικασίες που απαιτούνται για την σύνθεση (synthesis) και την εφαρμογή (fit) του πομπού μπορεί να δημιουργηθεί ένα αρχείο test bench waveform το οποίο χρησιμοποιείται από το πρόγραμμα Modelsim για προσομοίωση. Στο αρχείο αυτό εμφανίζονται τα σήματα εισόδου και εξόδου του πομπού στα οποία δίνονται οι επιθυμητές τιμές που θα χρησιμοποιηθούν στην προσομοίωση. Πιο συγκεκριμένα στην είσοδο clk δίνεται το ρολόι του συστήματος και στο reset ο παλμός επανατοποθέτισης οποιαδήποτε στιγμή κρίνεται απαραίτητο. Στην είσοδο clk_lim(15:0) δίνουμε τον αρθμό με τον οποίο θα γίνει η διαίρεση με την αρχική συχνότητα για να πετύχουμε το σωστό data rate του πομπού και δέκτη. Στην είσοδο d_in(3:0) δίνουμε τις τιμές των δεδομένων που θα παίρνει το κλυκλωμα του πομπού από το τηλεχιριστήρειο. Από την έξοδο Sout έχουμε την έξοδο της πληροφορίας πλέον σειριακά και όχι παράλληλα. Στην πληροφορία αυτή έχει προστεσθεί στην αρχή και στο τέλος ένα start_bit και ένα stop_bit. Για την προσομοίωση του πομπού συντάχθηκε ένα αντιπροσωπευτικό πρόγραμμα σε ένα αρχείο test bench waveform και εκτελέστηκε προσομοίωση Post-Fit VHDL model. Η προσομοίωση αυτή δείχνει σε πραγματικούς χρόνους την ορθή λειτουργία του πομπού κατά την εκτέλεση του συγκεκριμένου προγράμματος. Τα αποτελέσματα της προσομοίωσης φαίνονται στο σχήμα 3.7.


100 ps
/test/clk
/test/clk_lim
/test/d_in
/test/reset
/test/sout
1
0000000000000001
1111
0
0
0000000000000001
1111
Now
100200000 ps
Cursor 1
37778401 ps
000000
40000000
42000000
44000000
46000000
48000000
50000000
37842977 ps to 50546672 ps
Now: 100,200 ns Delta: 0


/test/clk
/test/clk_lim
/test/d_in
/test/reset
/test/sout
0
00000000000001
1001
0
0
1111
Now
1002000000 ps
Cursor 1
32086411 ps
26000000
28000000
30000000
32000000
34000000
36000000
32086411 ps
Now: 100,200 ns Delta: 0
25166512 ps to 37870207 ps

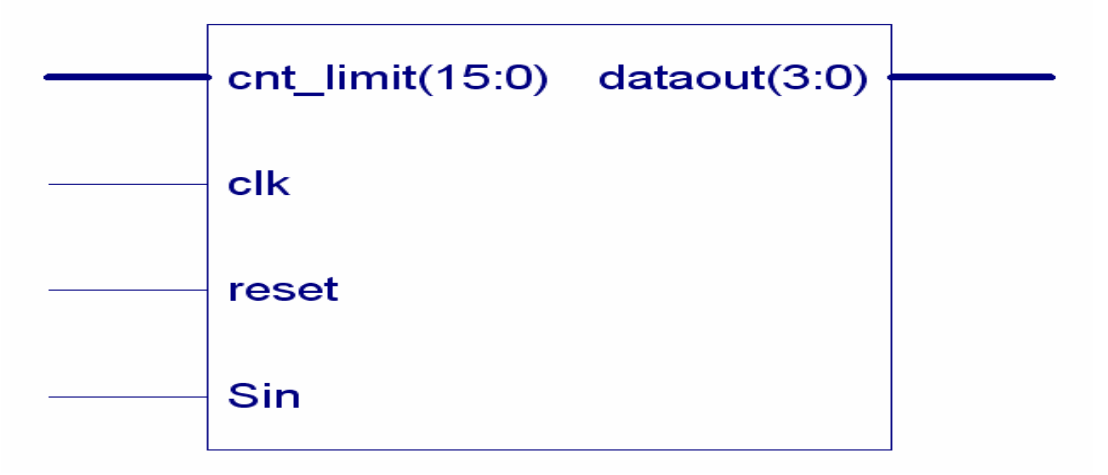

/test/clk
/test/clk_lim
/test/d_in
/test/reset
/test/sout
0
0000000000000001
0000
0
1
0000000000000001
0000
1001
Now
100200000 ps
Cursor 1
198015525 ps
198015525 ps
14000000
16000000
18000000
20000000
22000000
24000000


/test/clk
/test/clk_lim
/test/d_in
/test/reset
/test/sout
0
00000000000000001
0000
0
1
Now
1002000000 ps
Cursor 1
19801525 ps
14000000
16000000
18000000
20000000
22000000
24000000
19801525 ps
1001
0000


/test/clk
/test/clk_lim
/test/d_in
/test/reset
/test/sout
0
0000000000000001
0110
0
1
0000
0000000000000001
0110
Now
1002000000 ps
Cursor 1
5806748 ps
2000000
4000000
6000000
8000000
10000000
12000000
5806748 ps

3.3 ΑΝΑΠΤΥΞΗ ΚΥΚΛΩΜΑΤΟΣ ΔΕΚΤΗ ΣΤΟ CPLD XC9572 ΤΗΣ XILINX

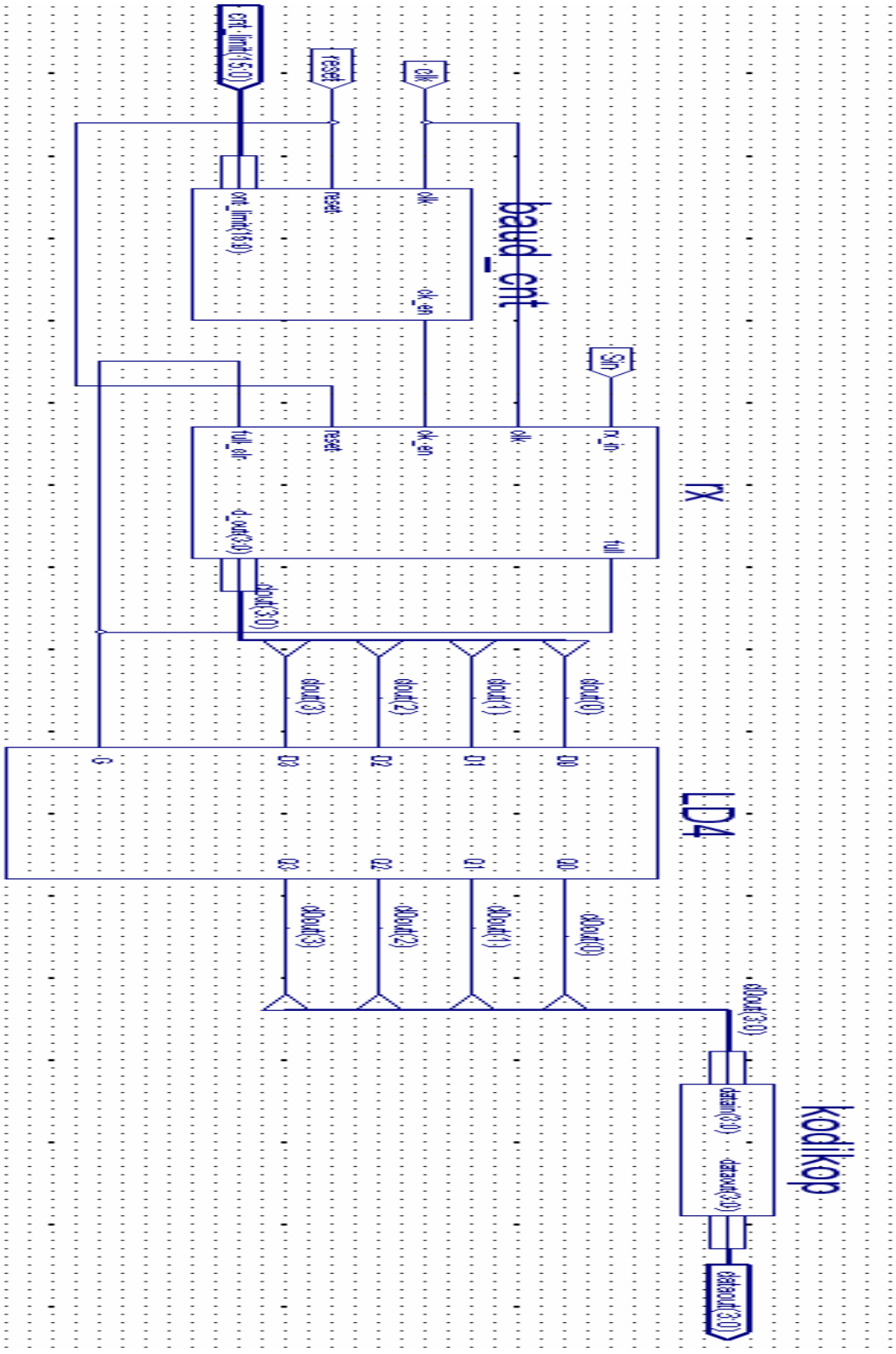
Το κύκλωμα του δέκτη συμπτύχθηκε στο σύμβολο dektis, σχήμα 3.8. Μπορεί να χρησιμοποιηθεί σαν το τελικό κύκλωμα που θα προγραμματιστεί σε ένα CPLD ή να χρησιμοπιηθεί για την σύνθεση άλλων κυκλωμάτων. Η σύνδεση των στοιχείων του κυκλώματος του δέκτη (top level module) έγινε με χρήση σχηματικού (κυκλωματική σύνδεση) και όχι με τη γλώσσα VHDL. Τα στοιχεία που συνθέτουν το κύκλωμα πομπου υλοποιούνται είτε μα χρήση της γλώσσας προγραμματισμού VHDL είτε με τη κυκλωματική σύνδεση άλλων πιο απλών στοιχείων που και αυτά με τη σειρά τους υλοποιούνται από άλλα στοιχεία.



Σχ3.8 ΜΠΛΟΚ ΔΙΑΓΡΑΜΜΑ ΚΥΚΛΩΜΑΤΟΣ ΔΕΚΤΗ

Η εξοδό του, dataout(3:0), είναι παράλληλη και μετά από κατάλληλη ενύσχηση και απομόνωση από το υπόλιπο κύκλωμα, θα οδηγηθεί στους κινητίρες. Το reset χρισημοποιήται για να γίνη η εκκίνηση από κάποια προκαθορισμένη αρχική κατάσταση. Στην είσοδο clk διοχετεύσαμε ένα τετραγωνικό σήμα στην αντίστοιχη είσοδο του CPLD. Η Sin είναι μία σειριακή είσοδος που λαμβάνει τις πληροφορείες αποδιαμορφομένες και έτοιμες πρως επεξεργασεία από τον δέκτη. Τέλος η είσοδος cnt_limit(15:0) αντιπροσοπεύει τον αρθμό με τον οποίο θα γίνει η διαίρεση με την αρχική συχνότητα για να πετύχουμε το σωστό data rate του πομπού και δέκτη. Στο Σχ.3.9 φαίνοντια τα επιμέρους κυκλώματα του δέκτη. Αμέσως πιο κάτω παρουσιάζονται τα κύρια στοιχεία που υλοποιού το κύκλωμα του δέκτη.

- Μετρητης δημιουργείας clk_en.
- Μονάδα επεκσεργασείας δεδομένων, rx.
- Μανδαλωτής, (latch) LD4.
- Κωδικοποιητής.



Σχ.3.9 ΕΠΙΜΕΡΟΥΣ ΚΥΚΛΩΜΑΤΑ ΤΟΥ ΔΕΚΤΗ

Final Results
RTL Top Level Output File Name : dektis.ngr
Top Level Output File Name : dektis
Output Format : NGC
Optimization Goal : Speed
Keep Hierarchy : YES
Target Technology : xc9500
Macro Preserve : YES
XOR Preserve : YES
wysiwyg : NO

Design Statistics
 # IOs : 23

Macro Statistics :
 # Registers : 35
 # 1-bit register : 35
 # Xors : 15
 # 1-bit xor2 : 15

Cell Usage :
 # BELS : 295
 # AND2 : 92
 # AND3 : 1
 # AND4 : 4
 # AND8 : 2
 # INV : 146
 # OR2 : 33
 # OR3 : 1
 # OR4 : 1
 # XOR2 : 15
 # FlipFlops/Latches : 35
 # FDC : 29
 # FDP : 2
 # LD : 4
 # IO Buffers : 23
 # IBUF : 19
 # OBUF : 4

Σχ.3.10 ΑΝΑΦΟΡΑ ΣΥΝΘΕΣΗΣ ΤΟΥ ΚΥΚΛΩΜΑΤΟΣ ΔΕΚΤΗ

RESOURCES SUMMARY

Macrocells Used	*Pterms Used*	*Registers Used*	*Pins Used*	*Function Block Inputs Used*
39/72 (55%)	*142/360 (40%)*	*35/72 (49%)*	*23/34 (68%)*	*79/144 (55%)*

PIN RESOURCES

Signal Type	*Required*	*Mapped*	*Pin Type*	*Used*	*Total*
Input	*18*	*18*	*I/O*	*20*	*28*
Output	*4*	*4*	*GCK/IO*	*2*	*3*
Bidirectional	*0*	*0*	*GTS/IO*	*0*	*2*
GCK	*1*	*1*	*GSR/IO*	*1*	*1*
GTS	*0*	*0*			
GSR	*0*	*0*			

GLOBAL RESOURCES

Signal mapped onto global clock net (GCK3)	*clk*

POWER DATA

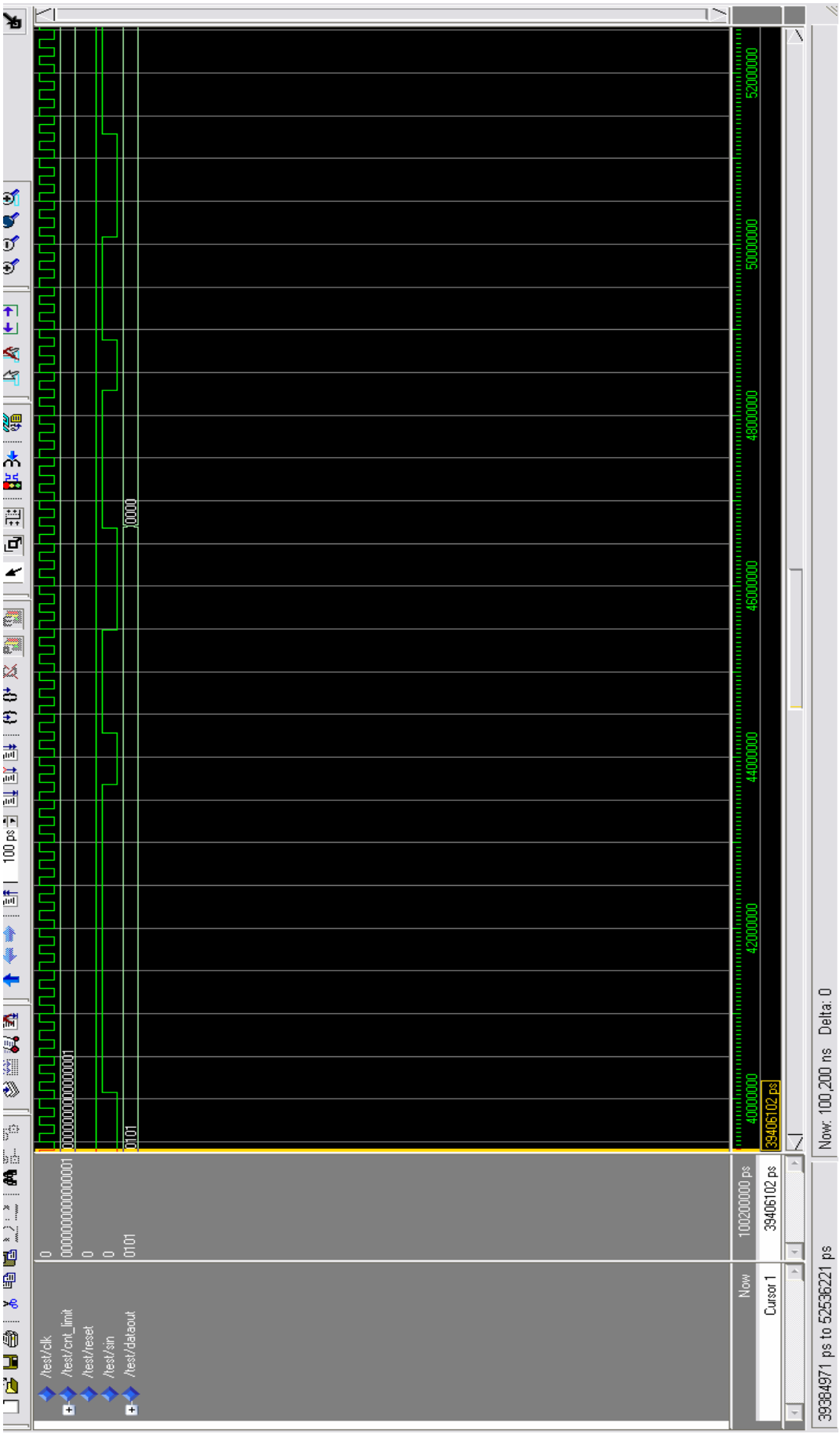
Macrocells in high performance mode (MCHP)	*39*
Macrocells in low power mode (MCLP)	*0*
Total macrocells used (MC)	*39*

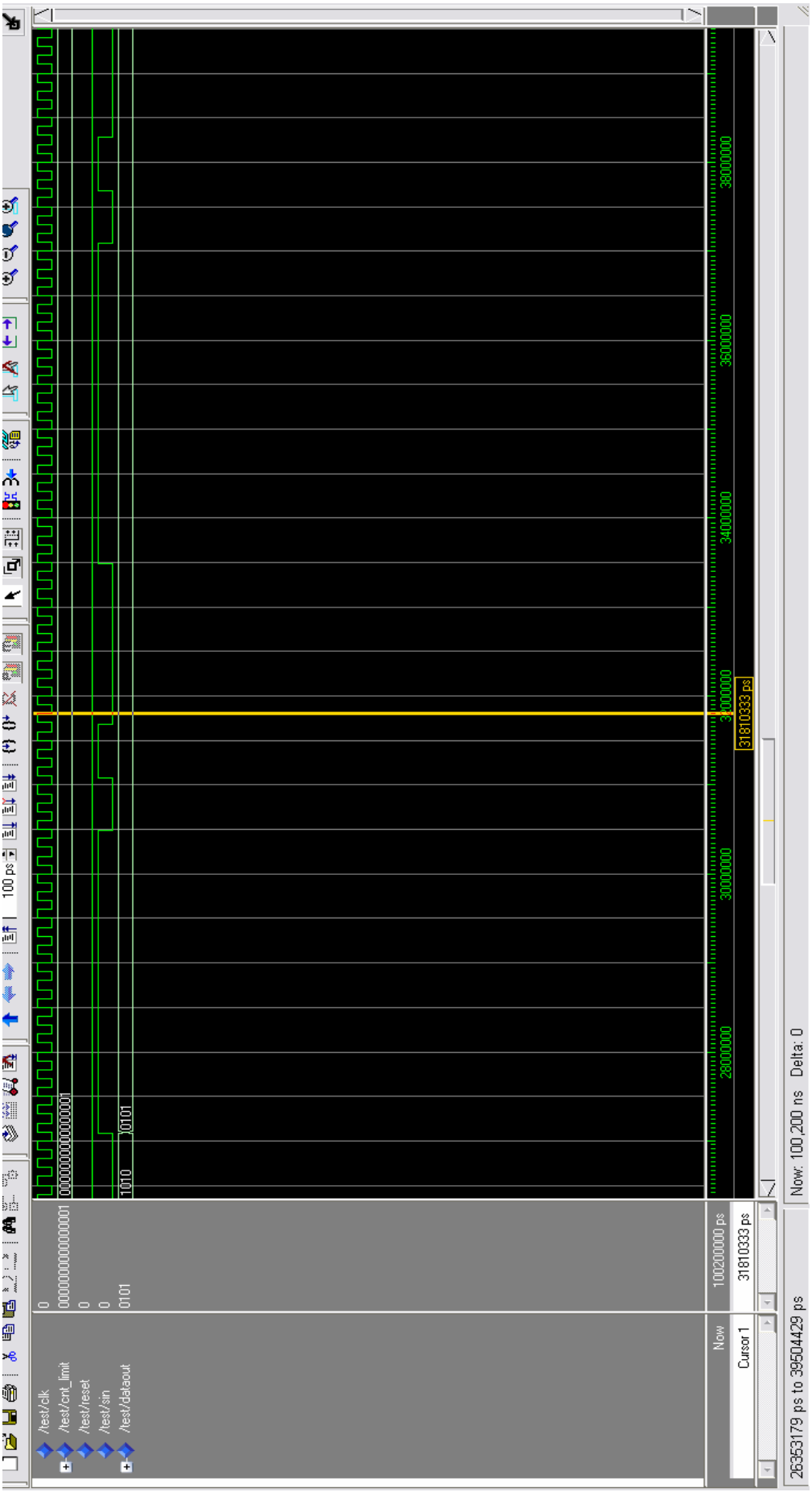
Σχ.3.11 ΑΝΑΦΙΡΑ ΠΟΡΩΝ ΤΟΥXC9572 ΠΟΥ ΚΑΤΑΛΑΜΒΑΝΕΙ ΤΟ ΚΥΚΛΩΜΑ ΠΟΜΠΥ

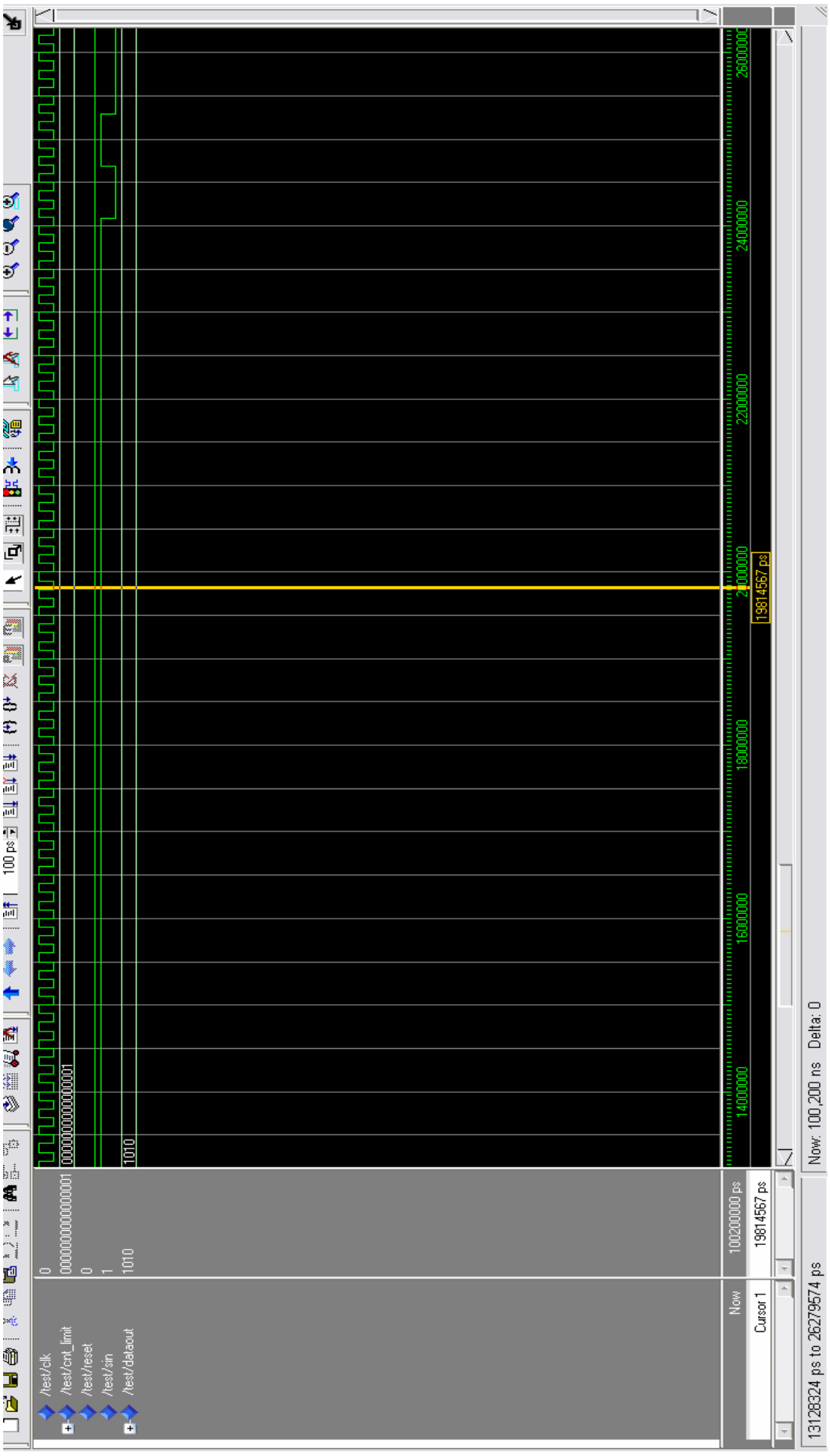
Στο σχήμα 3.10 παραιτίθεται η αναφορά της σύνθεσης του κυκλώματος του δέκτη (synthesis report). Η αναφορά αυτή δίχνει την αντιστοιχία του κυκλώματος σε λογικές πύλες. Στο σχήμα 3.11 παρουσιάζεται η αναφορά των πόρων του XC9572 που καταλαμβάνει ο δέκτης, δηλαδή ο αριθμός των μακροκυψέλων (macrocells), των ακροδεκτών (I\O Pins), των βαθμίδων λειτουργιών (Function Blocks) κ.τ.λ.. Οι δύο αναφορές αντλήθικαν από το κύκλωμα του δέκτη που υλοποιήθικε αποκλειστικά με τη σχηματική μέθοδο.

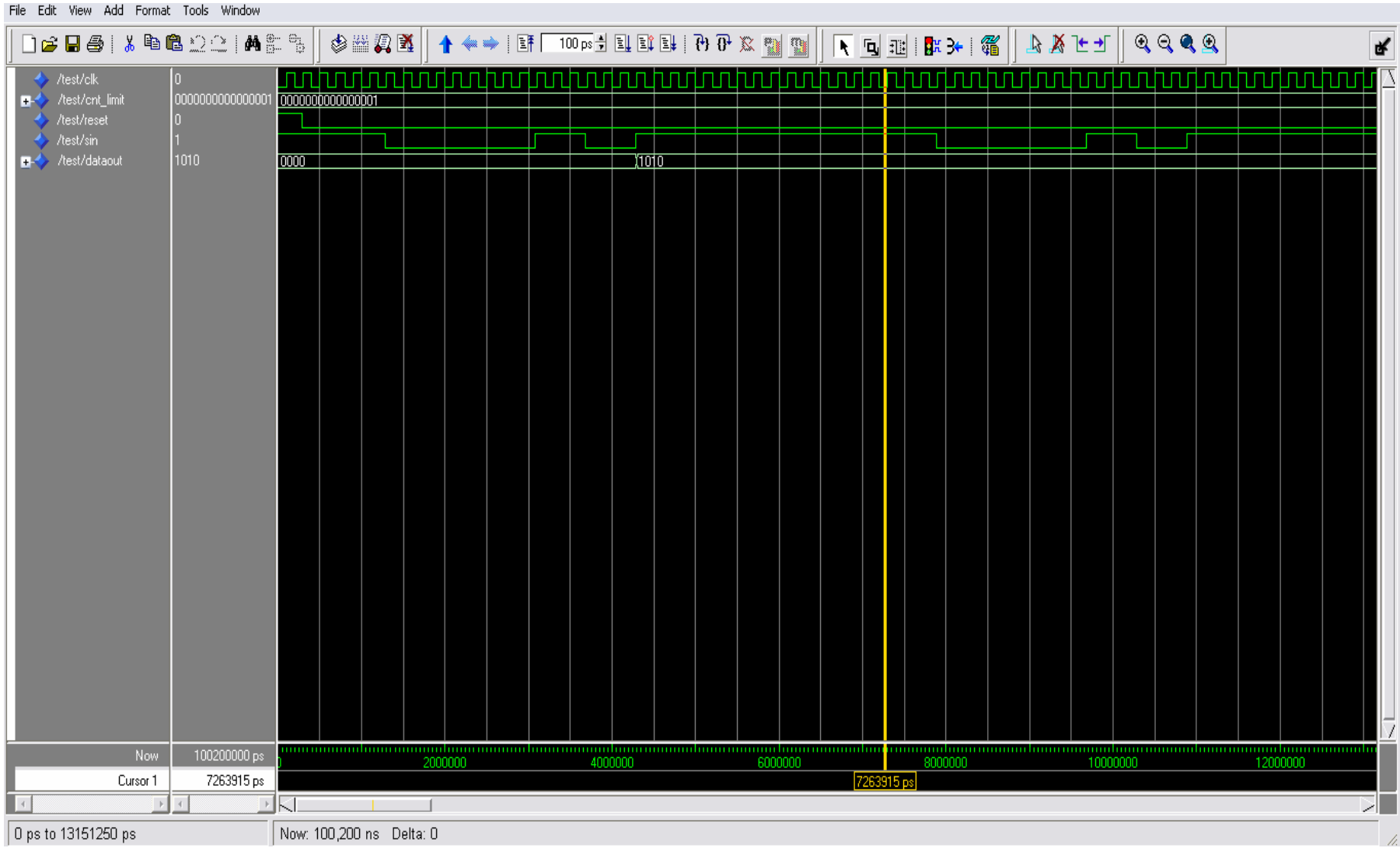
3.4 ΠΡΟΣΟΜΟΙΩΣΗ ΚΥΚΛΩΜΑΤΟΣ ΔΕΚΤΗ ΜΕ ΤΟ MODELSIM

Αφού ολοκληρωθούν όλες οι διαδικασίες που απαιτούνται για την σύνθεση (synthesis) και την εφαρμογή (fit) του δέκτη μπορεί να δημιουργηθεί ένα αρχείο test bench waveform το οποίο χρησιμοποιείται από το πρόγραμμα Modelsim για προσομοίωση. Στο αρχείο αυτό εμφανίζονται τα σήματα εισόδου και εξόδου του δέκτη στα οποία δίνονται οι επιθυμητές τιμές που θα χρησιμοποιηθούν στην προσομοίωση. Πιο συγκεκριμένα στην είσοδο clk δίνεται το ρολόι του συστήματος και στο reset ο παλμός επανατοποθέτισης οποιαδήποτε στιγμή κρίνεται απαραίτητο. Στην είσοδο clk_lim(15:0) δίνουμε τον αρθμό με τον οποίο θα γίνει η διαίρεση με την αρχική συχνότητα για να πετύχουμε το σωστό data rate του πομπού και δέκτη. Στην είσοδο Sin δίνουμε τις τιμές των δεδομένων που θα παίρνει το κύκλωμα του δέκτη μετά από την λύψη και αποδιαμόρφβση που θα κάνει το ιβριδικό κύκλωμα του δέκτη. Από την έξοδο dataout(3:0) έχουμε την έξοδο της πληροφορίας πλέον παράλληλα και όχι σειριακά. Στην πληροφορία αυτή έχει αφαιρεθεί το start_bit και το stop_bit. Για την προσομοίωση του δέκτη συντάχθηκε ένα αντιπροσοωπευτικό πρόγραμμα σε ένα αρχείο test bench waveform και εκτελέστηκε προσομοίωση Post-Fit VHDL model. Η προσομοίωση αυτή δείχνει σε πραγματικούς χρόνους την ορθή λειτουργία του δέκτη κατά την εκτέλεση του συγκεκριμένου προγράμματος. Τα αποτελέσματα της προσομοίωσης φαίνονται στο σχήμα 3.12.


/test/clk
/test/cnt_limit
/test/reset
/test/sin
/test/dataout
0
00000000000000001
0
0
0101
Now
100200000 ps
39406102 ps
Cursor 1
39384971 ps to 52536221 ps
Now: 100,200 ns Delta: 0
40000000
42000000
44000000
45000000
48000000
50000000
52000000
39406102 ps
0000
0101


/test/clk
/test/cnt_limit
/test/reset
/test/sin
/test/dataout
0
00000000000001
0
0
0101
1010
0101
Now
100200000 ps
Cursor 1
31810333 ps
26000000
30000000
31000000
31810333 ps
34000000
36000000
38000000
26353179 ps to 39504429 ps
Now: 100,200 ns
Delta: 0


/test/clk
/test/cnt_limit
/test/reset
/test/sin
/test/dataout
0
00000000000001
0
1
1010
Now
100200000 ps
Cursor 1
19814567 ps
14000000
16000000
18000000
20000000
22000000
24000000
26000000
19814567 ps
Now: 100,200 ns Delta: 0
13128324 ps to 26279574 ps


File Edit View Add Format Tools Window
100 ps
/test/clk
/test/cnt_limit
/test/reset
/test/sin
/test/dataout
0
0000000000000001
0
1
1010
0000
1010
Now
100200000 ps
Cursor 1
7263915 ps
2000000
4000000
6000000
8000000
10000000
12000000
7263915 ps
0 ps to 13151250 ps
Now: 100,200 ns Delta: 0

ΚΕΦΑΛΑΙΟ 4

ΣΤΟΙΧΕΙΑ ΑΣΥΡΜΑΤΟΥ ΕΛΕΓΧΟΥ

Το πρόγραμμα της Xilinx, ISE 7.1i προσφέρει τρεις βασικούς τρόπους σχεδίασης, τη σχεδίαση με σχηματικό, με γλώσσα προγραμματισμού (VHDL, ABEL) και με διάγραμμα καταστάσεων. Για κάθε τρόπο σχεδίασης υπάρχουν τα κατάλληλα εργαλεία που βοηθούν στην σύνθεση ενός κυκλώματος.

Στην σχεδίαση με σχηματικό το πρόγραμμα διαθέτει μια μεγάλη ποικιλία έτοιμων στοιχείων σύνθεσης κυκλωμάτων, στο σχηματικό ονομάζονται σύμβολα. Στα σύμβολα αυτά περιλαμβάνονται όλα τα βασικά (primitives) στοιχεία σύνθεσης ενός λογικού κυκλώματος όπως είναι οι πύλες, οι απομονωτές και οι πολυδονητές αλλά και πιο σύνθετα στοιχεία όπως οι μετρήτες, οι μνήμες και οι αριθμητικές μονάδες. Τα βασικά σύμβολα υλοποιούνται με χρήση της γλώσσας VHDL, ενώ τα πιο σύνθετα υλοποιούνται είτε με τη χρήση της γλωσσας VHDL, είτε με κυκλωματική σύνδεση βασικών στοιχείων. Στη σχεδίαση με τη γλώσσα VHDL, το πρόγραμμα προσφέρει πρότυπους κώδικες οι οποίοι αποτελούν και αυτοί στοιχεία ενός κυκλώματος.

Ο χρήστης μπορεί να χρησιμοποιήσει τα έτοιμα σύμβολα και τους πρότυπους κώδικες για να συνθέσει τα δικά του κυκλώματα. Η σύνθεση των στοιχείων αυτών μπορεί να γίνει με κυκλωματική σύνδεση ή με χρήση γλώσσας προγραμματισμού. Κάθε κύκλωμα που υλοποιήται, με οποιοδήποτε από τους τρόπους που προσφέρι το πρόγραμμα, μπορεί να αποτελέσει μια ξεχωριστή μονάδα, ένα σύμβολο το οποίο μπορεί να χρησιμοποιηθεί για την υλοποίηση άλλων κυκλωμάτων. Στο κεφάλαιο αυτό περιγράφονται, κατά αλφαβητική σειρά, όλα τα στοιχεία που

συνθέτουν τον ασύρματο έλεγχο αυτοκινήτου και στον πίνακα του σχήματος 4.1 παρουσιάζονται εν συντομία.

ΣΥΜΒΟΛΟ	ΛΕΙΤΟΥΡΓΙΑ
AND2B1	Πύλη And δύο εισόδων με μία ανεστραμμένη είσοδο
boud_cnt	Μετρητης δημιουργείας clk_en
kodikopiitis	Κύκλωμα μετατροπης σημάτων εισόδου σε σήματο κίνησης των κινητίρων
LD4	Μανδαλωτής, (latch)
OR4	Πύλη OR τεσσάρων εισόδων
rx	Μονάδα επεκσεργασείας δεδομένων κυκλώματος δέκτη
tx	Μονάδα επεκσεργασείας δεδομένων κυκλώματος πομπού

Σχ.4.1 ΣΤΟΙΧΕΙΑ ΣΥΝΘΕΣΗΣ ΑΣΥΡΜΑΤΟΥ ΕΛΕΓΧΟΥ

4.1 ΤΟ ΣΥΜΒΟΛΟ AND2B1

Το σύμβολο AND2B1 είναι μία πύλη AND δύο εισόδων με μία αναστροφή. Η πύλη AND2B1 εμφανίζει λογικό ΄1΄στην έξοδο μόνο όταν η μία είσοδο τοποθετηθεί σε λογικό ΄1΄ και η ανεστραμένη είσοδο τοποθετηθεί σε λογικό ΄0΄. Ο πίνακας αλήθειας της πύλης AND2B1 φαίνεται στο σχήμα 4.2.

ΕΙΣΟΔΟΙ		**ΕΞΟΔΟΙ**
A	**B**	F
0	0	0
0	1	0
1	0	1
1	1	0

ΣΧ.4.2 ΠΙΝΑΚΑΣ ΑΛΗΘΕΙΣ AND2B1

Τα σύμβολα αυτά όπως και όλες οι πύλες είναι στοιχειώδη (primitive) σύμβολα και υλοποιούνται με χρήση της γλώσσα VHDL. Στο σχήμα 4.3 παρατήθεται ο κώδικας VHDL της AND2B1.

```
process (A,B)
begin
F<= A and (not B);
end process;
```

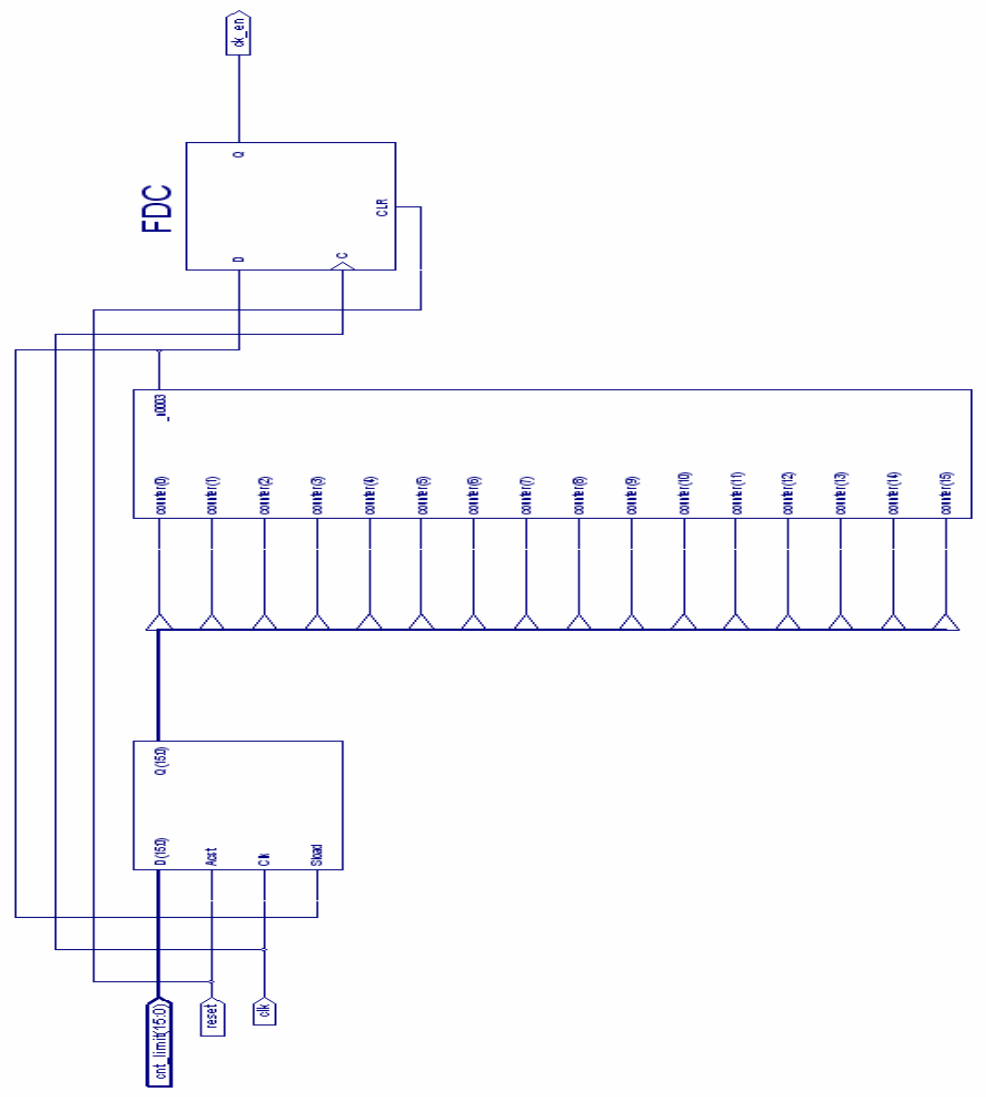
Σχ.4.3 ΚΩΔΙΚΑΣ AND2B1 ΣΕ ΓΛΩΣΣΑ VHDL

4.2 ΤΟ ΣΥΜΒΟΛΟ baud_cnt

Το σύμβολο αυτό δεν είναι στην ουσία τιποτα άλλο από έναν μετρητή που κάνει μια γυγκεκριμένη δουλεί. Ο μετριτής αυτός αποτελήται από μια είσοδο clock που είναι το global clock (25MHz), από μία είσοδο reset, από έναν δεκαεξάμπιτο αριθμό εισόδου και μια έξοδο την clk_en. Ο μετριτής κάνει την διαίρεση του clock με τον δεκαεξάμπιτο αριθμό (clk_lim(15:0)) και κάθε φορά που τελειώνει η διαίρεση βγάζει στην έξοδο clk_en έναν παλμό. Ο παλμός αυτός θα αποτελέσει το clock για το data rate του πομπού και δέκτη. Το data rate του πομπού είναι 4kHz και του δέκτη είναι 2kHz. Άρα κοινό data rate

έχουμε τα 2kHz. Για να μπορέσουμε να βρούμε τον δεκαεξάμπιτο αριθμό θα πρέπει να διαιρέσουμε το global clock με το data rate (25MHz÷2kHz). Ο αριθμός που προκίπτει είναι $(30D4)_{16}$ ή $(0011000011010100)_2$.

Το σχηματικό διάγραμμα του baunt_cnt φαίνεται στο σχήμα 4.4. Αποτελήται από τρία διαφορετικά κυκλώματα. Το πρώτο κύκλωμα αποτελήται από καταχωρητές στους οποίους αποθικεύ εται ο δεκαεξάμπιτος αριθμός. Το δεύτερο και μεγαλύτερο μπλοκ κύκλωμα, αποτελήται από μια πύλη AND δεκαέξι εισόδων. Επιδή μια τέτοι πύλη είναι αδήνατον να υπάρξη, η υλοποίησή της γίνεται με δύο πύλες AND οκτώ εισόδων και μιας πύλης AND δύο εισόδων. Τέλος το τρίτο κύκλωμα έναι ένα D – Flip Flop το οποίο βγάζει την έξοδο clk_en όταν είναι αυτό απαραίτιτο.



Σχ.4.4 ΣΧΗΜΑΤΙΚΟ ΔΙΑΓΡΑΜΜΑ ΤΟΥ baud cnt

Στο σχήμα 4.5 παρατίθεται ο κώδηκας του baunt_cnt σε γλώσσα VHDL. Η λειτουργία του κυκλωματος βασίζεται στην εντολή IF για τον έλεγχο των σημάτων reset και clk. Αρχικά ορίζεται μια μεταβλητή counter που χρησιμεύει σαν ενδιάμεσο στάδιο για την επεξεργασία των σημάτων εισόδου.

Αν το reset=’1’ τότε η έξοδος του κυκλώματος είναι σε λογική κατάσταση ‘0’ και ο counter επανέρχεται στην αρχική του τιμή. Αν έχουμε όμως μεταβολή στο σήμα clk στο θετικό μέτοπο, τότε αν ο counter έχει την αρχική του τιμή το κύκλωμα να βγάλει στην έξοδο έναν παλμό και να αποθηκεύσει στον counter το σήμα εισόδου του cnt_limit. Σε κάθε άλλη περίπτοση η έξοδος του κυκλώματος να είναι ‘0’ και να γίνει μείωση κατά ένα στον counter. Με αυτό τον τρόπο πετιχαίνουμε την διαίρεση του CLOCK μειώνοντας κατά ένα τον counter σε κάθε παλμό.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity baud_cnt is
   Port (
        clk       : in std_logic;
        reset     : in std_logic;
        cnt_limit  : in std_logic_vector(15 downto 0);
        ck_en     : out std_logic
   );
end baud_cnt;

architecture RTL of baud_cnt is

   signal counter :  std_logic_vector(15 downto 0);

begin

   BAUD_CNT: process(clk, reset)
   begin
      if (reset ='1') then
         counter <= "0000000000000001";
```

```
        ck_en <= '0';
      elsif(clk'event and clk = '1') then
        if (counter = "0000000000000001") then
          ck_en <= '1';
          counter <= cnt_limit;
        else
          ck_en <= '0';
          counter <= counter - 1;
        end if;
      end if;
   end process;
end RTL;
```

Σχ. 4.5 Ο ΚΩΔΗΚΑΣ baunt_cnt ΣΕ ΓΛΩΣΣΑ VHDL

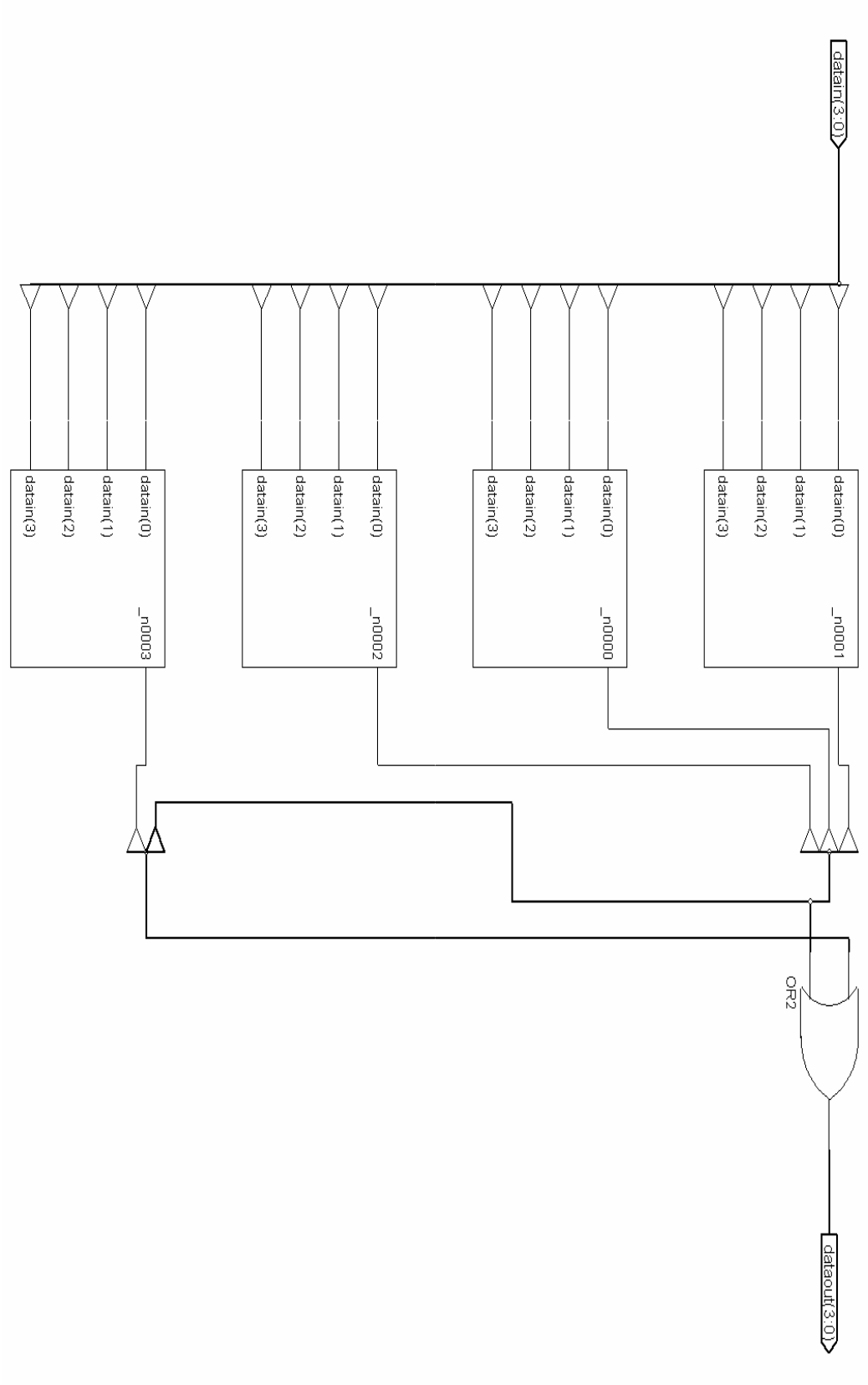
4.3 ΣΥΜΒΟΛΟ kodikopiitis

Οι πληροφορίες που βρίσκονται στην είσοδο του κωδικοποιητή είναι αυτές που στάλθικαν από το κύκλωμα του πομπού έπειτα από κατάληλη επεξεργασία. Με λίγα λόγια είναι οι πληροφορίες που στάλθικαν από τα push buttons του τηλεχειρηστηρίου. Σε αυτό το κύκλωμα κωδικοποιούνται τα δεδομένα με τέτοιο τρόπο ώστε η έξοδος του κωδικοποιητή να κινήσειι, μετά από κατάλληλη ενισχισή και απομώνοση, τους κινητίρες του οχήματος προς την σωστή κατεύθινση. Δηλαδή η κίνηση του κινητίρα να αντιστιχεί στην σωστή ένδιξη κίνησης του αντίστοιχου πλήκτρου. Τέλος το κύκλωμα αυτό ελέγχει αν έχει πατηθεί περισσότερα από ένα πλήκτα. Αν συμβεί αυτό το όχημα θα παραμίνει ακίνητο. Έχουμε ορίσει ότι για την δεξιά κίνηση του κινητίρα τα σήματά του θα πρεί να είναι οι λογικές καταστάσεις ‘01’, για την αριστερή κίνηση ‘10’ και για να παραμένι ακήνιτο ‘00’. Έστω με Α ορίζουμε το πλίκτρο που θα κινήσει το όχημα μπροστά, με Β το πλίκτρο που θα στρίπσει το όχημα δεκσιά, με C το πλίκτρο για την πίσω κίνηση, D για το πλίκρο για την αριστερή κίνιση και με K1 τον δεξί κινιτήρα και K2 τον αριστερό κινητίρα, τότε ο πίνακας αλήθειας του κυκλώματος φαίνεται στο σχήμα 4.6.

ΕΙΣΟΔΟΙ				**ΕΞΟΔΟΙ**	
D	C	B	A	K2	K1
0	0	0	0	00	00
0	0	0	1	01	01
0	0	1	0	10	01
0	0	1	1	00	00
0	1	0	0	10	10
0	1	0	1	00	00
0	1	1	0	00	00
0	1	1	1	00	00
1	0	0	0	01	10
1	0	0	1	00	00
1	0	1	0	00	00
1	0	1	1	00	00
1	1	0	0	00	00
1	1	0	1	00	00
1	1	1	0	00	00
1	1	1	1	00	00

ΣΧ.4.6 ΠΙΝΑΚΑΣ ΑΛΗΘΕΙΣ kodikopiiti

Το μπλοκ διάγραμα του kodikopiiti φαίνεται στο παρακάτω σχήμα.



Ο κώδικας του kodikopiiti, σχίμα 4.8, κάνει χρήση της εντολής CASE, για έλεγχο υων καταστάσεων εισόδου. Ελέγχονται τέσσερεις από τους δεκαέξι πιθανούς συνδιασμούς της εισόδου, αφού υπάρχουν μόνο τέσσερις επιλογές κίνησης του οχήματος. Αρχικά ορίζεται το σήμα που θα ελέγχεται είναι το data, με την εντολή ‘’CASE DATA IS’’. Ακολούθως ελέγχονται οι τέσσερις πιθανοί συνδιασμοί του data με την εντολή WHEN. Κάθε φορά που ικανοποιήται μια συνθίκη, εκτελείται η ανάθεση της αντίστοιχης εισόδου στην έξοδο. Σε περίπτωση που το σήμα data πάρει τιμές που δεν ικανοποιούν καμιά συνθήκη, αναθέτη στην έξοδο την κατάσταση ‘’0000’’.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity kodikop is
   Port ( datain : in std_logic_vector(3 downto 0);
        dataout : out std_logic_vector(3 downto 0));
end kodikop;

architecture Behavioral of kodikop is
begin
        process (datain)
                variable data:std_logic_vector(3 downto 0);
                        begin
                        data:=datain;
                        case data is
                                when "0001"         =>
                                        dataout<="0101";
                                when "0010"         =>
                                        dataout<="1001";
                                when "0100"         =>
                                        dataout<="1010";
                                when "1000"         =>
                                        dataout<="0110";
                                when others =>
                                        dataout<="0000";
                        end case;
        end process;
```

```
end Behavioral;
```

Σχ.4.8 Ο ΚΩΔΗΚΑΣ ΤΟΥ kodikopiiti ΣΕ ΓΛΩΣΣΑ VHDL

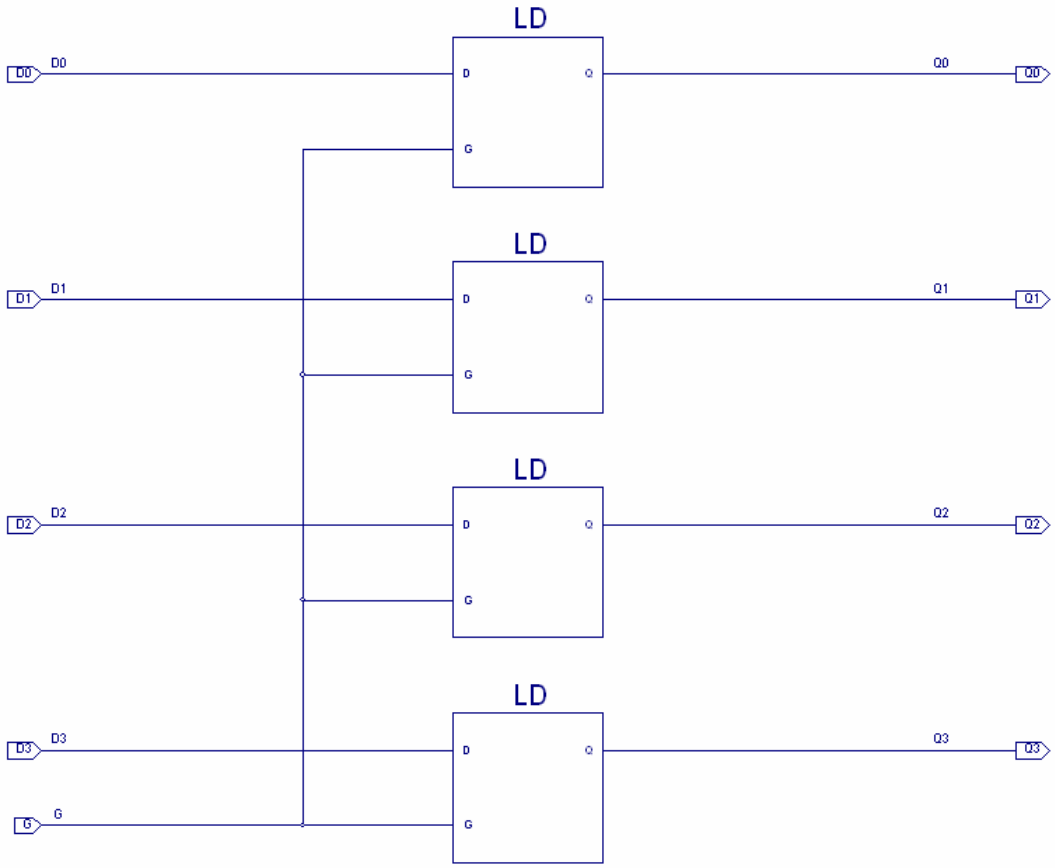
4.4 ΣΥΜΒΟΛΟ LD4

Ο μανδαλωτής είναι ένα κύκλωμα, το οποίο κρατάει τις τιμές στις εξόδους του αμετάβατες ότι και αν συμβεί στις εισόδους του, αν δεν έχει ενεργοποιηθεί το gate enable (G). Όταν ενεργοποιηθεί το gate enable τότε οι τιμές τις εισόδου μεταφέρονται στην έξοδο. Όπως φαίνεται και στο Σχ. 2.8 έχουμε συνδέσει την έξοδο full του rx στην είσοδο full_clr και στην είσοδο G του μανδαλωτή. Επίσης συνδέσαμε την έξοδο d_out(3:0) του rx στην είσοδο του μανδαλωτή. Με αυτό τον τρόπο πετύχαμε τη μεταφορά των πληροφοριών στην έξοδο του μανδαλωτή κάθε φορά που τελειώνει ο rx την επεξεργασία των πληροφοριών. Επίσης προστατεύουμε τις πληροφορίες στην έξοδο του μανδαλώτη από ότι παρεμβολές και να γίνουν στην εισοδό του μανδαλωτή, αφού είναι γνοστό ότι λειτουργεί και ως στοιχείο μνήμης. Ο πίνακας αλήθεις του LD4 φαίνεται στο σχήμα 4.9.

Inputs		Outputs
G	D	Q
1	0	0
1	1	1
0	X	No Chg
↓	D	D

Σχ.4.9 ΠΙΝΑΚΑΣ ΑΛΗΘΕΙΑΣ ΤΟΥ LD4

Το σχηματικό διάγραμμα του LD4 φαίνεται στο σχήμα 4.10 και αποτελήται από τέσσερα D – flop flops.



Σχ.4.10 ΣΧΗΜΑΤΙΚΟ ΔΙΑΓΡΑΜΜΑ ΤΟΥ LD4

Το πρόγραμμα του σχήματος 4.11 ορίζει μια οντότητα που ονομάζεταο latch, η οποία έχει εισόδους D και CLK και έξοδο Q. Η διαδικασία χρησιμοποιεί μια εντολή if – then – else για να ορίσει την τιμή τα εξόδου Q. Έτσι, όταν είναι Clk=1, η έξοδος Q λαμβάνει την τιμή D. Όταν όμως το CLK δεν είναι ίσο με ένα, το πρόγραμμα δεν καθορίζει ποια τιμή πρέπει να έχει η έξοδος Q. Επομένως η έξοδος Q θα διατηρήσει τηντρέχουσα τιμή της και έτσι το πρόγραμμα περιγράφει το χρονιζόμενο μανδαλωτή τύπου D. Ο κατάλογος ευαισθησίας της διαδικασίας περιλαμβάνει και τις δύο εισόδους Clk και D επειδή τα ασήματα αυτά μπορούν να προκαλέσουν αλλαγή στην τιμή της εξόδου Q.

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

ENTITY latch IS
        PORT (D, Clk  : IN STD_LOGIC;
                Q       : OUT STD_LOGIC);
END latch

ARCHITECTURE Behavio OF latch IS
BEGIN
    PROCESS (D,Clk)
     BEGIN
          IF Clk='1' THEN
                Q<=D;
           END IF;
      END PROCESS;
END Behavior;
```

Σχ.4.11 Ο ΚΩΔΗΚΑΣ ΤΟΥ LD4 ΣΕ ΓΛΩΣΣΑ VHDL

4.5 ΣΥΜΒΟΛΟ OR4

Το σύμβολο OR4 είναι μια πύλη OR τεσσάρων εισόδων. Η πύλη OR εμφανίζει λογικί ΄΄1΄΄ στην έξοδό τους όταν έστω και μία από τις εισόδους της είναι τοποθετημένη σε λογικό ΄΄1΄΄. Ο πίνακας αληθειας της πύλης OR4 φαίνεται στο σχήμα 4.12.

ΕΙΣΟΔΟΙ				ΕΞΟΔΟΣ
A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1

0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Σχ.4.12 ΠΙΝΑΚΣ ΑΛΗΘΕΙΑΣ ΤΗΣ ΠΥΛΗΣ OR4

Τα σύμβολα αυτά όπως και όλες οι πύλες είναι στοιχειώδη (primitive) σύμβολα και υλοποιούνται με χρήση της γλώσσα VHDL. Στο σχήμα 4.13 παρατήθεται ο κώδικας VHDL της OR4.

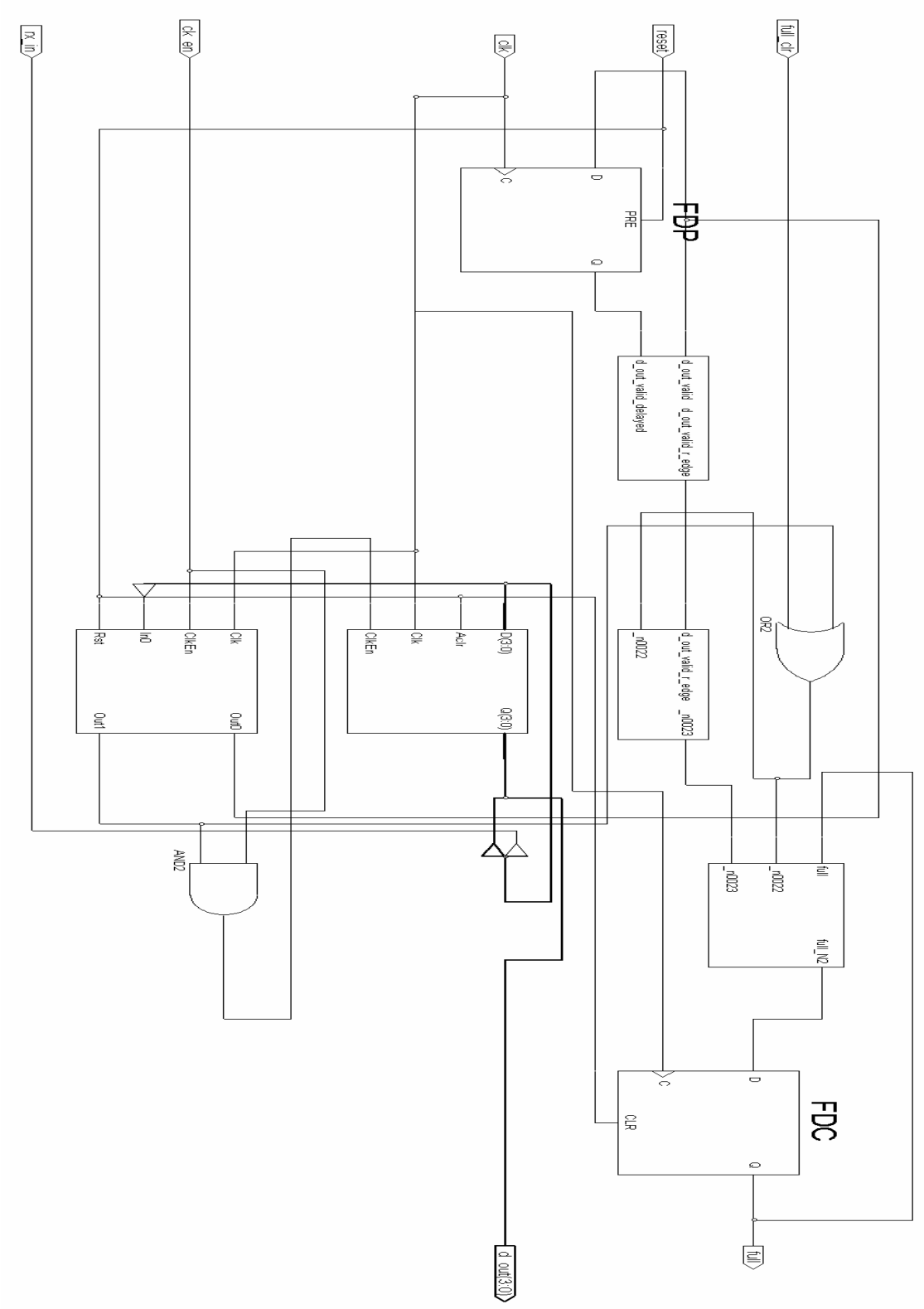
```
Architecture Behavioral of or4 is
Begin
Process (A,B,C,D)
Begin
Q<=A or B or C or D;
End process;
End Behavioral;
```

Σχ.4.13 Ο ΚΩΔΗΚΑΣ ΤΗΣ OR4 ΣΕ ΓΛΩΣΣΑ VHDL

4.6 ΤΟ ΣΥΜΒΟΛΟ rx

Η μονάδα αυτή αποτελήται από μια σειριακή είσοδο δεδομένων (δεδομένα μετά την λήψη κα αποδιαμόρφωση από τον δέκτη), από την είσοδο full_clr στην οποία δηλώνουμε κάθε πότε θέλουμε να ξεκιμάει η επεξεργασελεία του επόμενου πακέτου πληροφοριών και από το clk_en που θα αποτελέσει το clock για το data rate του πομπού και δέκτη. Τέλος σαν είσοδο έχει το reset και το global clock. Το κύκλωμα αποτελήται και από δύο εξόδους, την full και την dout(3:0). Η full μας ενημερώνει για το πότε το κύκλωμα βρίσκεται σε λειτουργεία και το ποτε έχει τελειόσει την επεκσεργασεία των δεδομένω. Τέλος στην

rx_out έχουμε την έξοδο της πληροφορίας πλέον παράλληλα και όχι σειριακά. Στην πληροφορία αυτή έχει αφαιρεθεί από την αρχή και το τέλος το start_bit και το stop_bit. Το σχηματκό διάγραμμα του rx φαίνεται στο παακάτω σχήμα.



Σχ.4.14 ΣΧΗΜΑΤΙΚΟ ΔΙΑΓΡΑΜΜΑ ΤΟΥ rx

Στο σχήμα 4.15 φαίνεται ο κώδικας του rx σε γλωσσα VHDL. Ο κώδικας του rx αποτελήται από πέντε πολύ βασικά υποπρογράματα τα οποία εκτελούνται ταυτόχρονα. Τα προγράμματα αυτά είναι:

- **Rising edge detection**
  Στη οποία γίνεται η ανίχνευση των bit.
- **full flag handling**
  Στο κομμάτι αυτό του προγράμματος χηριζόμασται την σημαία full η οποία μας ιδοποιεί για το αν έχει τελειώσει η επεξεργασεία των δεδομένων (με την έξοδο full) και με το πότε θα ξεκινίση τον νέο κύκλο εκτέλεσης των νέων δεδομένωνω (με την είσοδο full_clr)
- **Serial to parallel shift register**
  Στο κομάτι αυτό έχουμε την μετατροπή της πληροφορίας από σειριακή σε παράλληλη χωρίς την εμφάνιση του start_bit και stop_bit.
- **Synchronous part of the receive state machine**
  Εδώ έχουμε τον συγχρονισμό του διαγράμματος των καταστάσεων
- **Receive state machine**
  Και τέλος το τελευταίο κομάτι στο οποίο περιγράφεται το κύκλομα του δέκτη με πρόγραμμα γραμμέσο σε διάγραμμα καταστάσεων.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity RX is
   Port (  rx_in   : in std_logic;                -- Serial data in
           clk     : in std_logic;                -- Main clock. Rising edge
           ck_en   : in std_logic;   -- Clock enable. Must be 3x faster than baud
           reset   : in std_logic;                  -- Main Reset
           d_out   : out std_logic_vector(3 downto 0);     -- Received byte
           full    : out std_logic;       -- Flag indicating that a byte was received
           full_clr: in  std_logic        -- Clears the full flag
   );
```

```
end RX;

architecture RTL of RX is

   type STATE_TYPE is (IDLE,
                START_START,   START_END,
                B0_START,      B0_SAMPLE,     B0_END,
                B1_START,      B1_SAMPLE,     B1_END,
                B2_START,      B2_SAMPLE,     B2_END,
                B3_START,      B3_SAMPLE,     B3_END,
                STOP
                );
 signal CS, NS: STATE_TYPE;                    -- current andnext state

signal rx_shift_reg : std_logic_vector(3 downto 0); -- data reg.RX is
shifted here
signal rx_shift_en        : std_logic;            -- enables the rx shift
 signal d_out_valid        : std_logic;           -- active in stop and idle mode
   signal d_out_valid_delayed  : std_logic;
   signal d_out_valid_r_edge   : std_logic;    -- indicates rising edge

begin


   -- Rising edge detection

   D_OUT_VALID_REDG: process (clk, reset)
   begin
      if (reset ='1') then
         d_out_valid_delayed <= '1';
      elsif (clk'event and clk = '1') then
         d_out_valid_delayed <= d_out_valid;
      end if;
   end process;

   d_out_valid_r_edge <= d_out_valid and (not d_out_valid_delayed);


   -- full flag handling
```

```
FULL_FLAG : process (clk, reset)
begin
   if (reset ='1') then
      full <= '0';
   elsif (clk'event and clk = '1') then
      if( full_clr = '1' or rx_shift_en= '1' ) then
         full <= '0';
      elsif (d_out_valid_r_edge = '1') then
         full <= '1';
      end if;
   end if;
end process;


-- Serial to parallel shift register

RX_SHIFT: process (clk, reset)
begin
   if (reset ='1') then
      rx_shift_reg <= (OTHERS => '0');
   elsif (clk'event and clk = '1') then
      if(rx_shift_en='1' and ck_en='1') THEN
         rx_shift_reg <= rx_in & rx_shift_reg(3 downto 1);
      end if;
   end if;
end process;
d_out <= rx_shift_reg;


-- Synchronous part of the receive state machine

SYNC_PROC: process (clk, reset)
begin
   if (reset ='1') then
      CS <= IDLE;
   elsif (clk'event and clk = '1') then
      if(ck_en ='1') THEN
         CS <= NS;
      end if;
   end if;
```

```
end process;


-- Receive state machine

COMB_PROC: process (CS, rx_in)
begin
  case CS is
      when IDLE =>

         rx_shift_en <= '0';
         d_out_valid <= '1';

         if ( rx_in = '0' ) then
            NS <= START_START;
         else
            NS <= IDLE;
         end if;
      -------------------------
      when START_START =>
         rx_shift_en <= '0';
         d_out_valid <= '1';
         NS <= START_END;
      when START_END =>
         d_out_valid <= '0';
         rx_shift_en <= '0';
         NS <= B0_START;
      -------------------------
      when B0_START =>
         rx_shift_en <= '0';
         d_out_valid <= '0';
         NS <= B0_SAMPLE;
      when B0_SAMPLE =>
         rx_shift_en <= '1';
         d_out_valid <= '0';
         NS <= B0_END;
      when B0_END =>
         rx_shift_en <= '0';
         d_out_valid <= '0';
         NS <= B1_START;
```

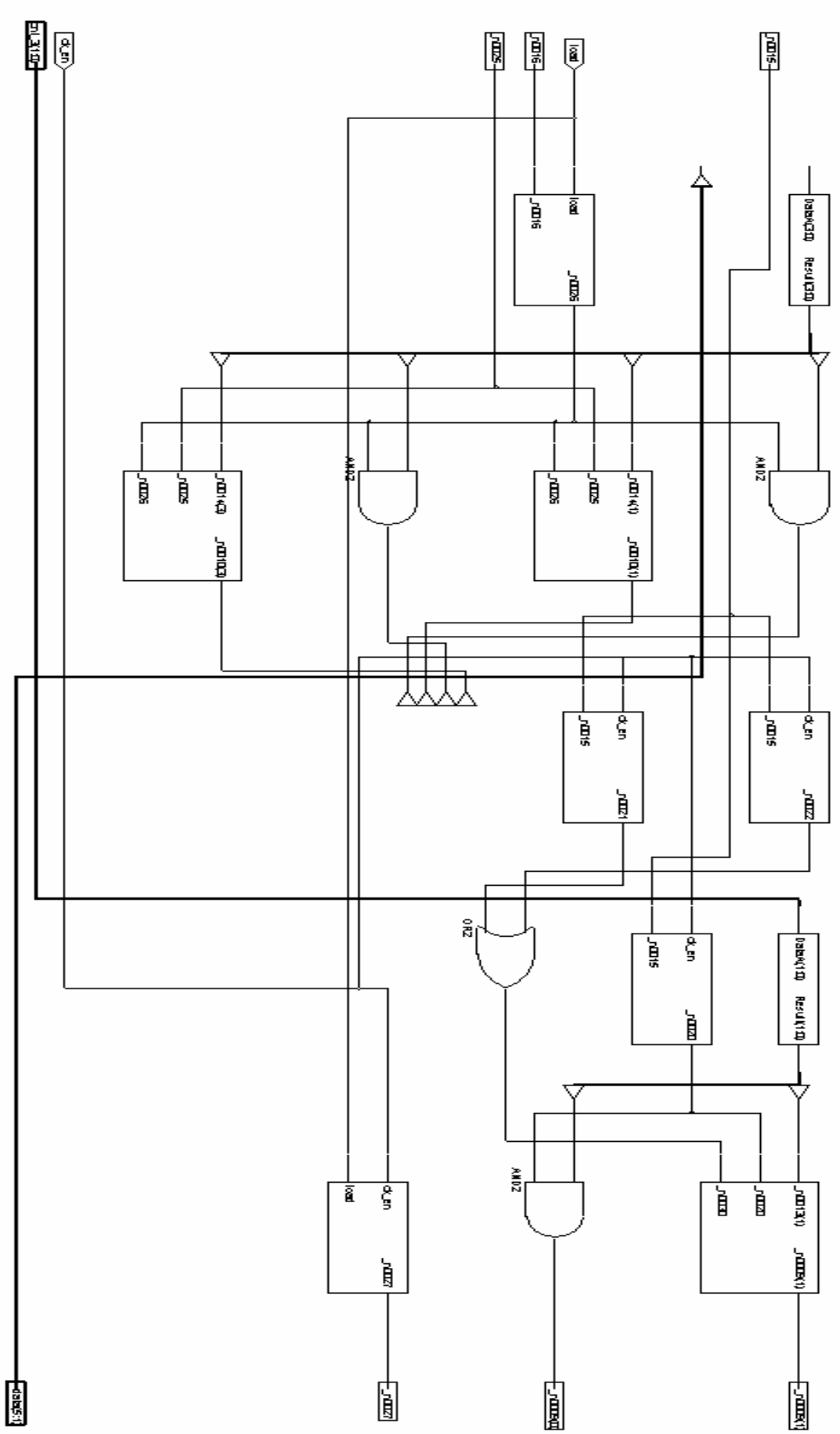
```
-------------------------
when B1_START =>
   rx_shift_en <= '0';
   d_out_valid <= '0';
   NS <= B1_SAMPLE;
when B1_SAMPLE =>
   rx_shift_en <= '1';
   d_out_valid <= '0';
   NS <= B1_END;
when B1_END =>
   rx_shift_en <= '0';
   d_out_valid <= '0';
   NS <= B2_START;
-------------------------
when B2_START =>
   rx_shift_en <= '0';
   d_out_valid <= '0';
   NS <= B2_SAMPLE;
when B2_SAMPLE =>
   rx_shift_en <= '1';
   d_out_valid <= '0';
   NS <= B2_END;
when B2_END =>
   rx_shift_en <= '0';
   d_out_valid <= '0';
   NS <= B3_START;
-------------------------
when B3_START =>
   rx_shift_en <= '0';
   d_out_valid <= '0';
   NS <= B3_SAMPLE;
when B3_SAMPLE =>
   rx_shift_en <= '1';
   d_out_valid <= '0';
   NS <= B3_END;
when B3_END =>
   rx_shift_en <= '0';
   d_out_valid <= '0';
   NS <= STOP;
when STOP =>
```

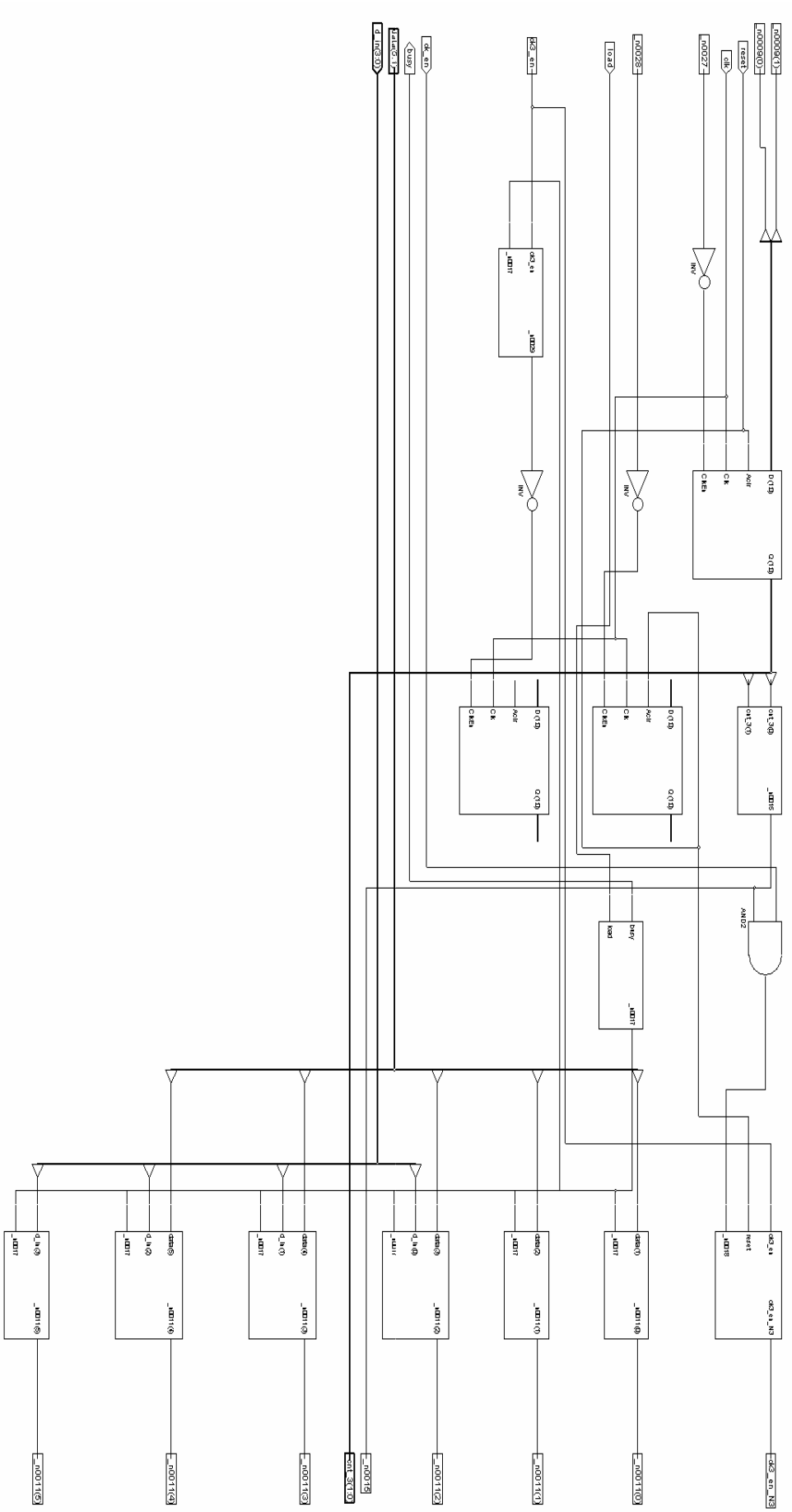
```
            rx_shift_en <= '0';
            d_out_valid <= '1';
            NS  <= IDLE;
        end case;
    end process;

end RTL;
```

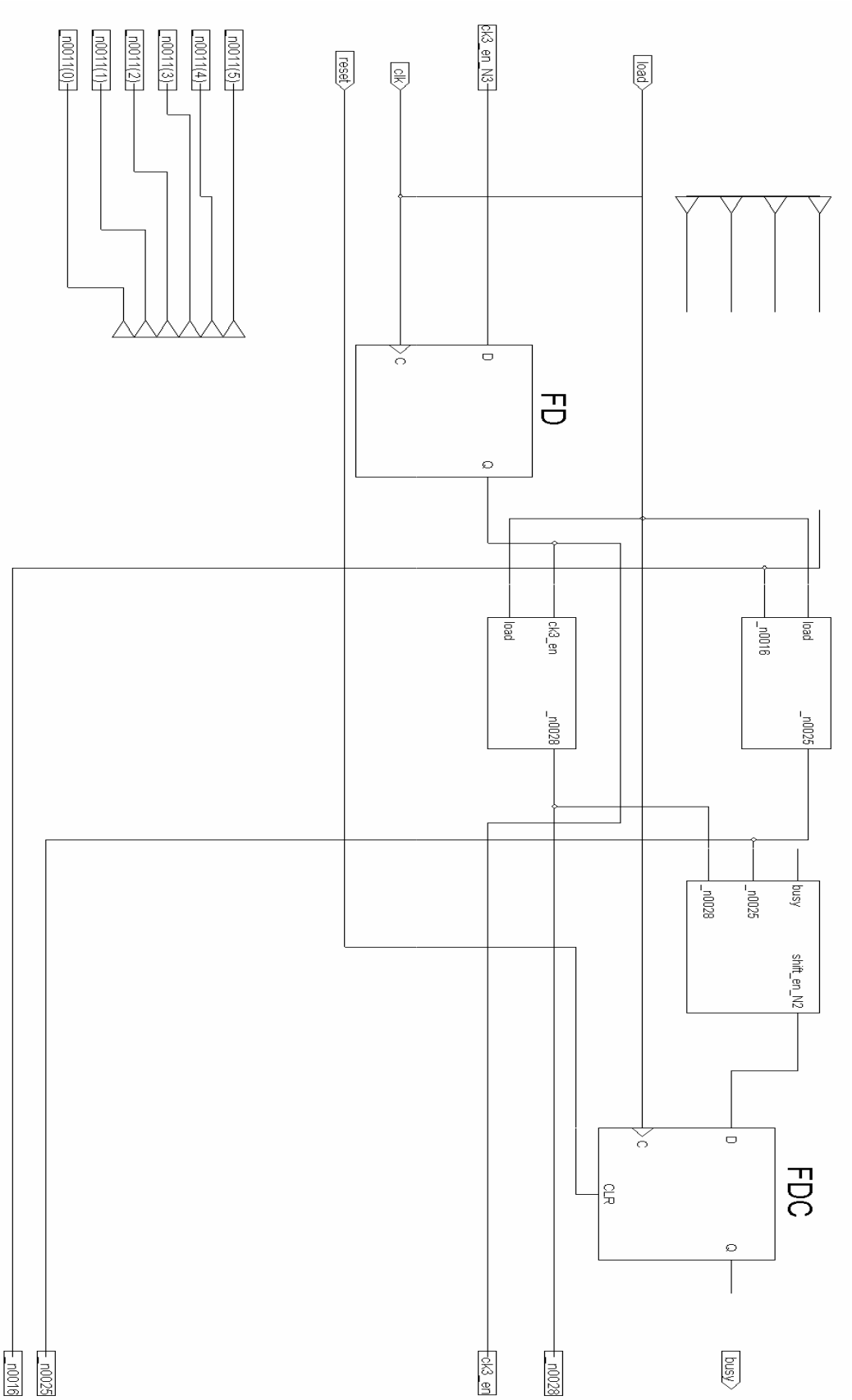
Σχ.4.15 Ο ΚΩΔΗΚΑΣ ΤΟΥ rx ΣΕ ΓΛΩΣΣΑ VHDL

4.7 ΣΥΜΒΟΛΟ tx

Η μονάδα αυτή αποτελήται από μια τετράμπιτη είσοδο δεδομένων (δεδομένα για την κίνηση του οχήματος), από την είσοδο load στην οποία διλώνουμε πότε να φορτώσει τα δεδομένα και από το clk_en που θα αποτελέσει το clock για το data rate του πομπού και δέκτη. Τέλος σαν είσοδο έχει το reset και το global clock. Το κύκλωμα αποτελήται και από δύο εξόδους, την busy και την tx_out. Η busy νας ενημερώνει για το αν το κύκλωμα βρίσκεται σε λιτουργεία ή όχι και αν μπορούμε να φορτώσουμε τις επόμενες πληροφορίες. Τέλος στην tx_out έχουμε την έξοδο της πληροφορίας πλέον σειριακά και όχι παράλληλα. Στην πληροφορία αυτή έχει προστεσθεί στην αρχή και στο τέλος ένα start_bit και ένα stop_bit. Αυτό έγινε γιατί ο έλεγχος του ασύρματου οχήματος είναι ασίχρονος, θα έπρεπε με κάπιο τρόπο να αντιληφθεί ο δέκτης το πότε και πού σταματάει η πληροφορεία. Κάτι αντοίστηχο όπως συμβαίνει και στο προτόκολο RS232. Το σχηματκό διάγραμμα του rx φαίνεται στο παακάτω σχήμα







Σχ.4.16 ΣΧΗΜΑΤΙΚΟ ΔΙΑΓΡΑΜΜΑ ΤΟΥ tx

Στο σχήμα 4.17 φαίνεται ο κώδικας του tx σε γλωσσα VHDL. Ο κώδικας του tx αποτελήται από τρεις πολύ βασικά υποπρογράματα τα οποία εκτελούνται ταυτόχρονα. Τα προγράμματα αυτά είναι:

- **Counter for dividing by 3 the ck_en. Generates ck3_en**
  έχουμε έναν μετριτή ο οποίος κάνει την διαίρεση του clk_en με το τρία.
- **Shift register**
  Στο κομμάτι αυτό του προγράμματος έχουμε την μετατροπή του παράλληλου σήματος σε σειριακό καθώς επίσης και την προσθίκη του start_bit και του stop_bit
- **Timer. Counts the number of bytes**
  Και τέλος το τελευταίο κομάτι στο οποίο περιγράφεται το κύκλωμα με το οποίο μετρολυνται τα δεδομένα που στάλθικάν με σκοπό την ενεργοποίηση της σημαίας Busy. Η σημαια αυτή μας ενημερώνει για το πότε έχει ολοκληρωθεί η αποστολή ενώς πακέτου πληροφοριών.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;



entity tx is
   Port (
        clk     : in std_logic;            -- main clock
        ck_en    : in std_logic;          -- clk enable. 3x faster than baud
        reset    : in std_logic;           -- main reset
        tx_out   : out std_logic;        -- TX data
         d_in     : in std_logic_vector(3 downto 0);  -- byte tobe  transmited
        load     : in std_logic;           -- load signal for d_in
        busy     : out std_logic          -- '1' during transmission
   );
end tx;
```

```
architecture RTL of tx is

signal data: std_logic_vector(5 downto 0); -- includes stop and start bits

signal cnt_3 : std_logic_vector(1 downto 0); -- cnt divides by 3 the ck_en
freq
signal ck3_en: std_logic;             -- clk enable 3x slower than ck_en
signal shift_en : std_logic;           -- enables the shifting
signal byte_timer   : std_logic_vector(3 downto 0); -- counts the
transmited bits

begin

   -- Counter for dividing by 3 the ck_en. Generates ck3_en

   BIT_CNT_PROC: process(clk, reset)
   begin
      if (reset ='1') then
         cnt_3 <= "00";
      elsif(clk'event and clk = '1') then
         if(load = '1') then
               cnt_3 <= "10";
         end if;
         if(ck_en ='1') then
            if (cnt_3 = "10") then
               ck3_en <= '1';
               cnt_3 <= "00";
            else
               ck3_en <= '0';
               cnt_3 <= cnt_3 + 1;
            end if;
         else
            ck3_en <= '0';
         end if;
      end if;
   end process;



   --  Shift register
```

```
  SHIFT_DATA: process (clk, reset, data)
  begin
    if (reset ='1') then
      data <= (OTHERS => '1');
    elsif clk'event and clk='1' then
      if (load='1' and shift_en = '0') then
       data <=  d_in & "01" ;          --"01" - idle tx folowed by start
      elsif(ck3_en = '1') then
        data <= '1' & data(5 downto 1); -- '1' at end for stop bit and idle
tx
      end if;
    end if;
    tx_out <= data(0);
  end process;

  --  Timer. Counts the number of bytes transmited
  BYTE_TIME: process (clk, reset)
  begin
    if (reset ='1') then
      byte_timer <= (OTHERS => '0');
      shift_en <= '0';
    elsif clk'event and clk='1' then
      if (load='1') then
        byte_timer <=  (OTHERS => '0');
        shift_en <= '1';
      elsif( ck3_en = '1') then
        if (byte_timer = "1010") then
          byte_timer <= "1010";
          shift_en <= '0';
        else
          byte_timer <= byte_timer + 1;
          shift_en <= '1';
        end if;
      end if;
    end if;
  end process;
  busy <= shift_en;
end RTL;
```

ΚΕΦΑΛΑΙΟ 5

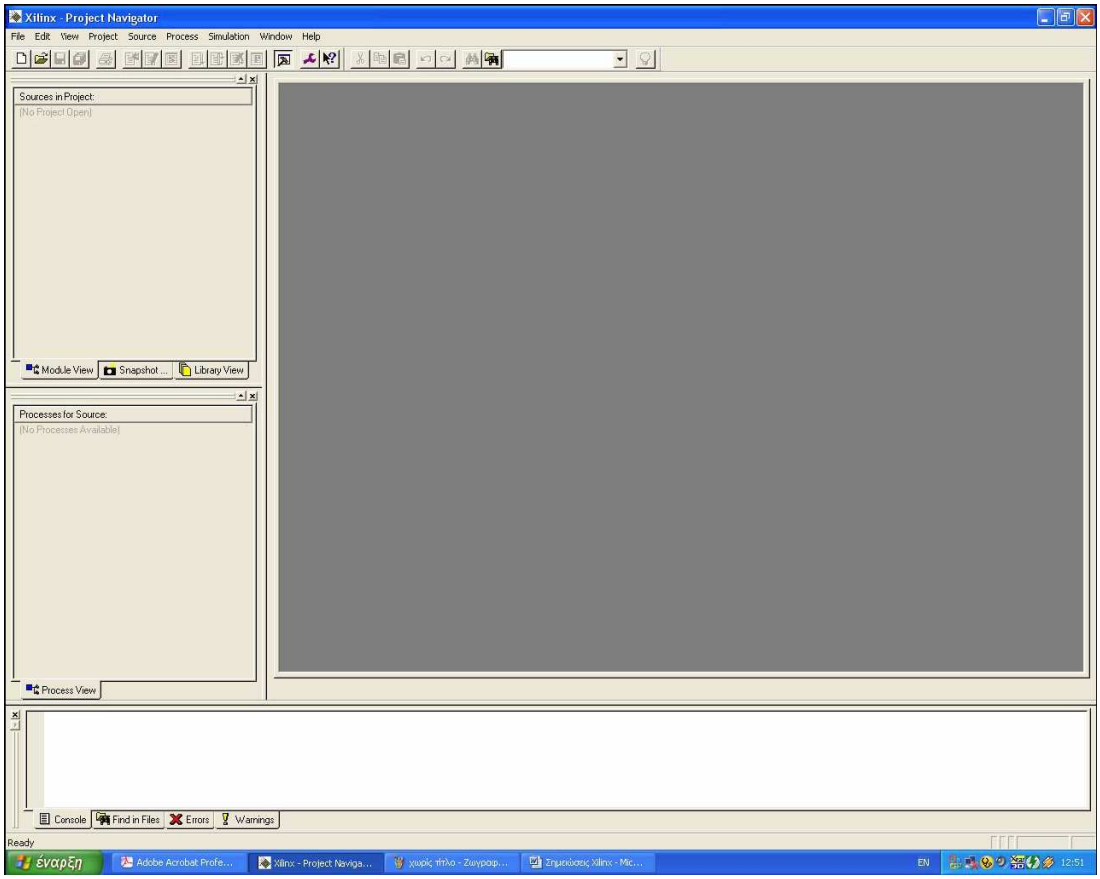
ΤΟ ΠΡΟΓΡΑΜΜΑ XILINX ISE 7.1i

5.1 ΕΙΣΑΓΩΓΗ ΣΤΟ ΠΡΟΓΡΑΜΜΑ Xilinx 7.1

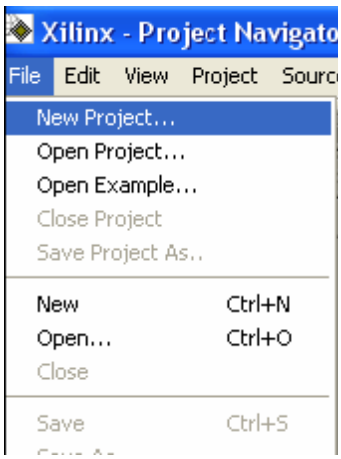
Το πρόγραμμα αυτό μας δίνει τη δυνατότητα να εργαστούμε σε μία μελέτη χρησιμοποιώντας αρκετές διαφορετικές προσεγγίσεις. Σχεδίαση μέσω σχηματικού, διαγράμματος ροής και γλώσσα VHDL. Παρακάτω θα επιχειρηθεί να δοθούν αρκετές πληροφορίες έτσι ώστε να μπορεί να χρησιμοποιηθούν και οι τρεις μεθόδους. Εδώ θα πρέπει να τονιστεί ότι ‘’ το καλύτερο πρόγραμμα είναι αυτό που ξέρω να δουλεύω’’ αυτό σημαίνει ότι σε αυτές τις σημειώσεις θα δοθούν οι βασικές αρχές και κατευθύνσεις και από εκεί και μετά όσο περισσότερο ο αναγνώστης δουλεύει με το πρόγραμμα αυτό τόσο περισσότερα θα μπορεί να κάνει.

5.2 ΣΧΕΔΙΑΣΜΟΣ ΜΕ ΧΡΗΣΗ ΣΧΗΜΑΤΙΚΟΥ

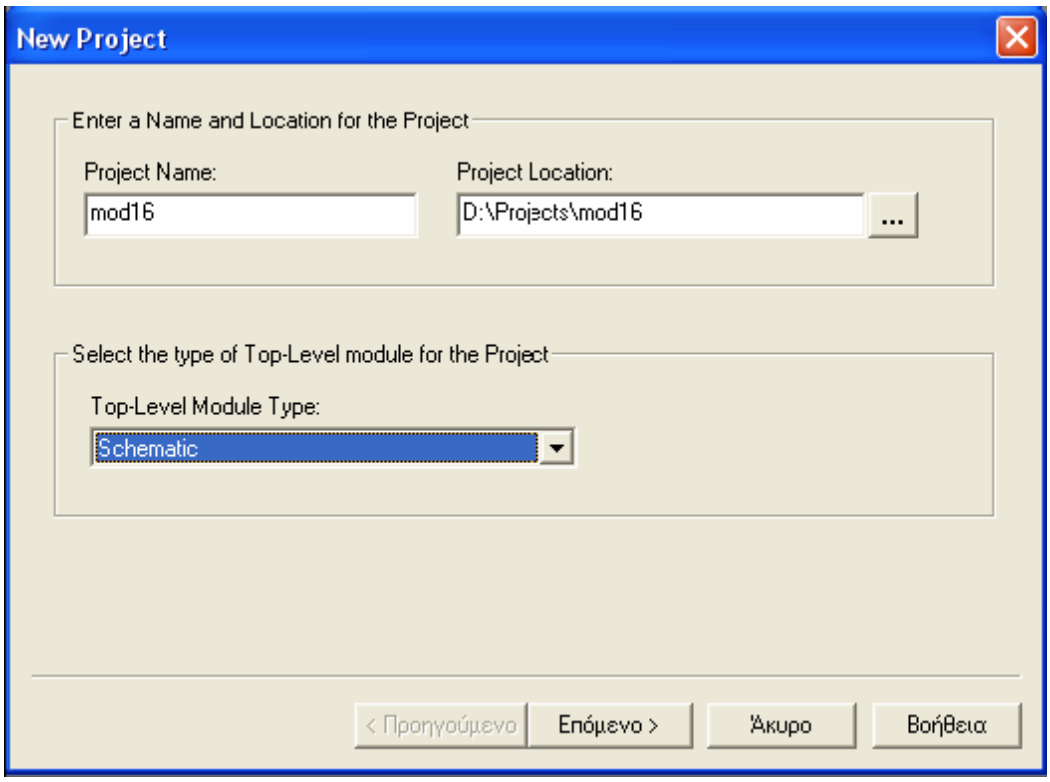
Το πρόγραμμα το οποίο θα χρησιμοποιήσουμε μας δίνει τη δυνατότητα να έχουμε όλα τα απαραίτητα αρχεία που θα χρησιμοποιήσουμε σε ένα Project. Εμείς ,για το συγκεκριμένο παράδειγμα θα εργαστούμε σε έναν ασύγχρονο μετρητή MOD 14. Ξεκινώντας ανοίγουμε το πρόγραμμα Xilinx ISE 7.1 και μας εμφανίζεται τοπαρακάτω περιβάλλον εργασίας



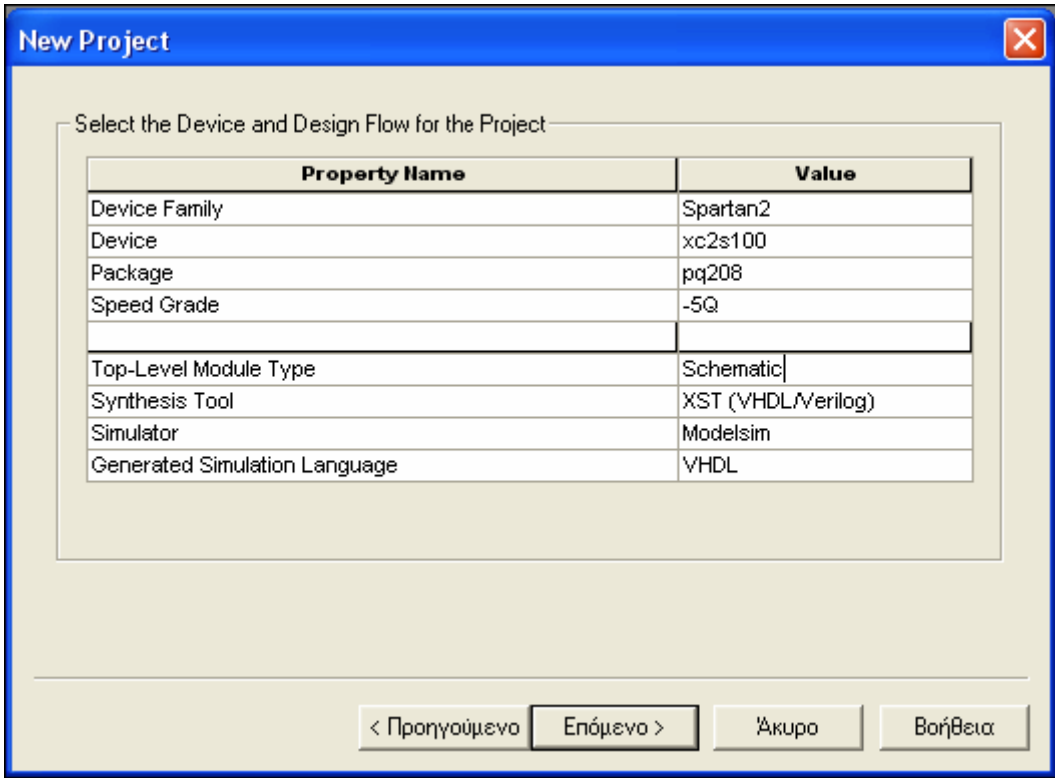
Για να ξεκινήσουμε το Project το πρόγραμμα μας δίνει και ένα Project Wizard μέσωτου οποίου θα ορίσουμε τις διάφορες απαραίτητες παραμέτρους για τη μελέτη.



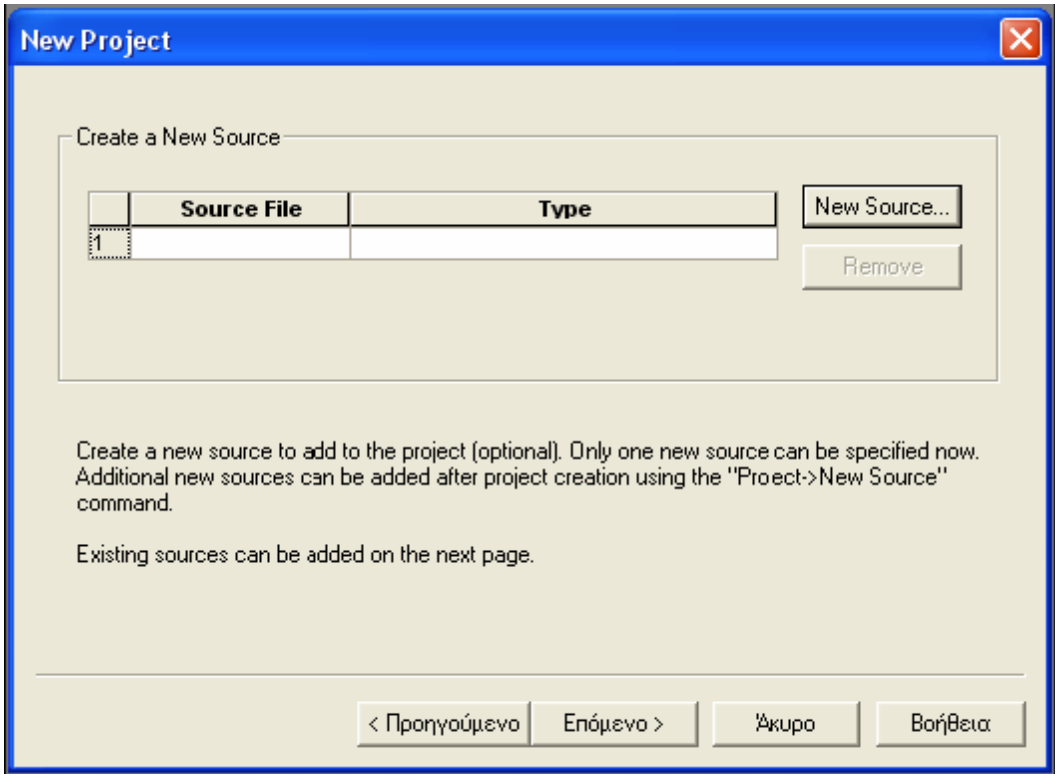
Εμφανίζεται το παρακάτω παράθυρο, όπου δίνουμε το όνομα του Project, τη θέση του μέσα στα αρχεία και το είδος της προσέγγισης που θα χρησιμοποιήσουμε.



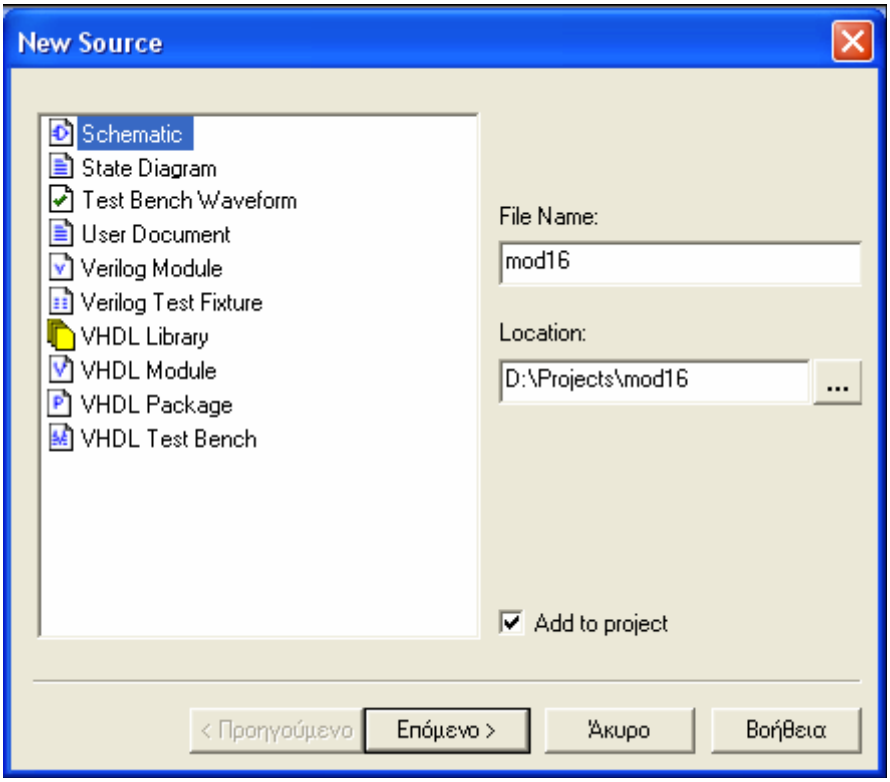
Κλικάροντας στο κουμπί ‘’Επόμενο’’ μας εμφανίζεται το παράθυρο στο οποίο ορίζουμε τα στοιχεία το FPGA ή CPLD που θα χρησιμοποιήσουμε. Στη συγκεκριμένη περίπτωση όμως επειδή θα δουλέψουμε με το αναπτυξιακό Spartan II FPGA ορίζουμε τα χαρακτηριστικά για το συγκεκριμένο FPGA.



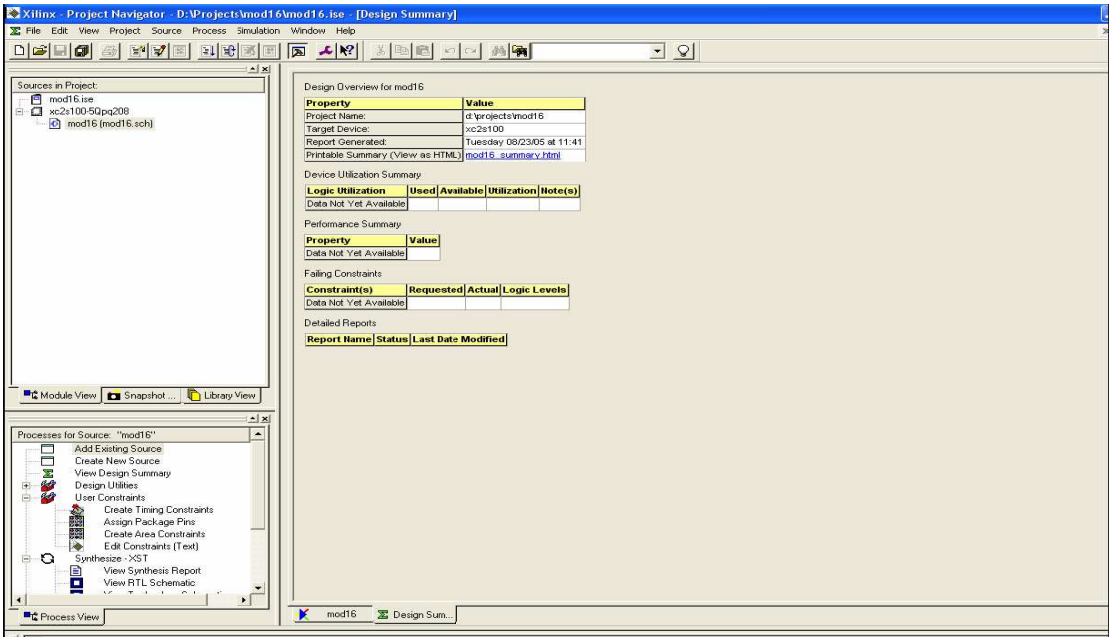
Συνεχίζουμε κλικάροντας στο ‘’Επόμενο’’ και μας ανοίγει το παράθυρο επιλογής τουείδους του αρχείου που θα χρησιμοποιήσουμε.

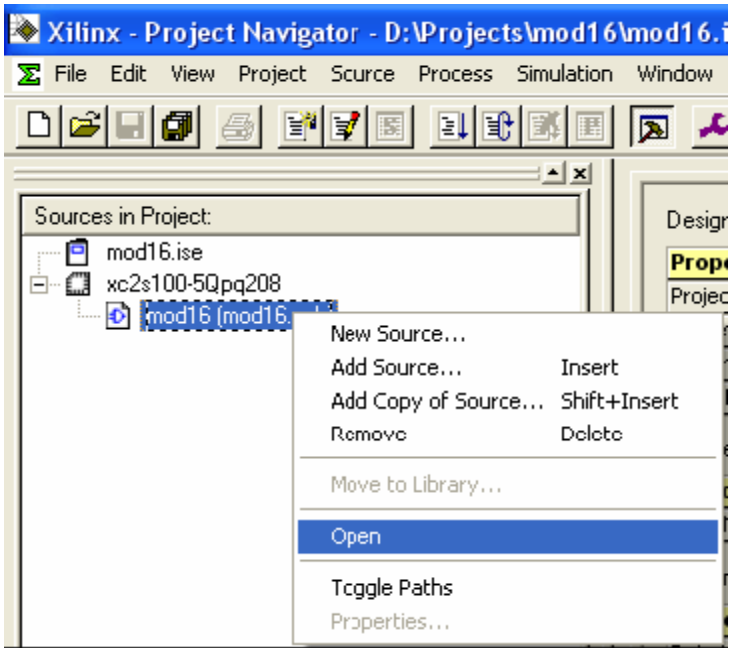


Επιλέγοντας το ‘’ New source’’ μας ανοίγει το παράθυρο ορισμού του αρχείου που θα εργαστούμε. Όπου ορίζουμε το όνομα ‘’MOD16” και τον τρόπο εργασίας ‘’Schematic.



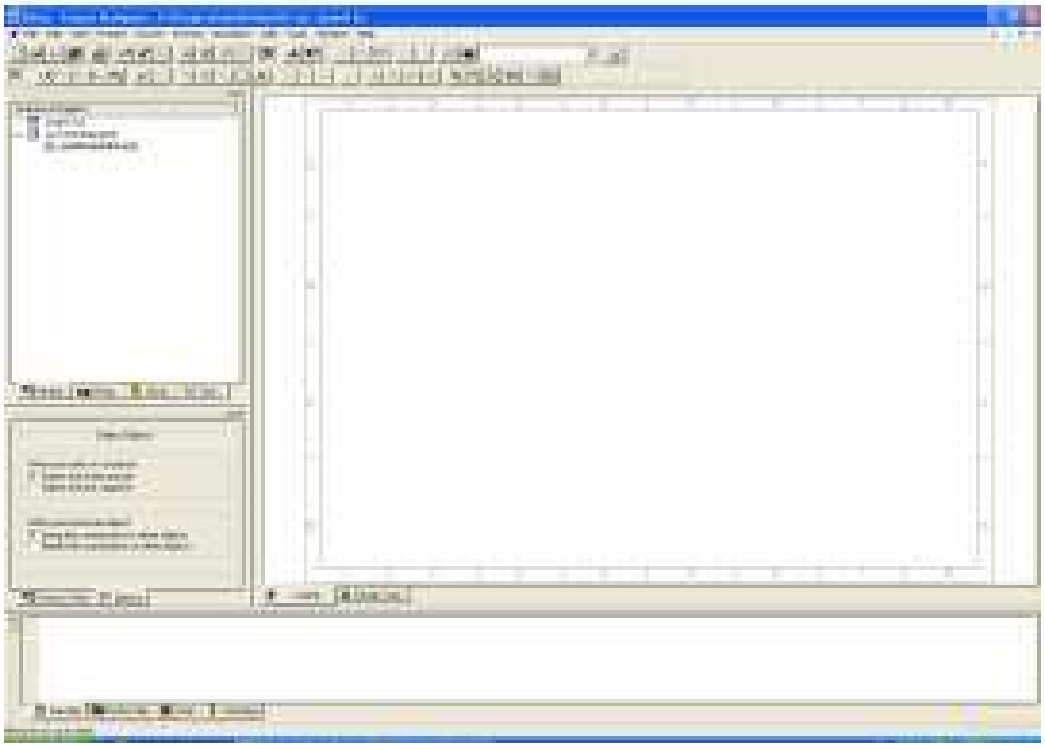
Προχωρούμε στο ‘’Επόμενο’’ και τελικά βλέπουμε στη επιφάνεια εργασίας του προγράμματος μία συνοπτική περιγραφή των στοιχείων που έχουμε εισάγει μέχρι στιγμής.



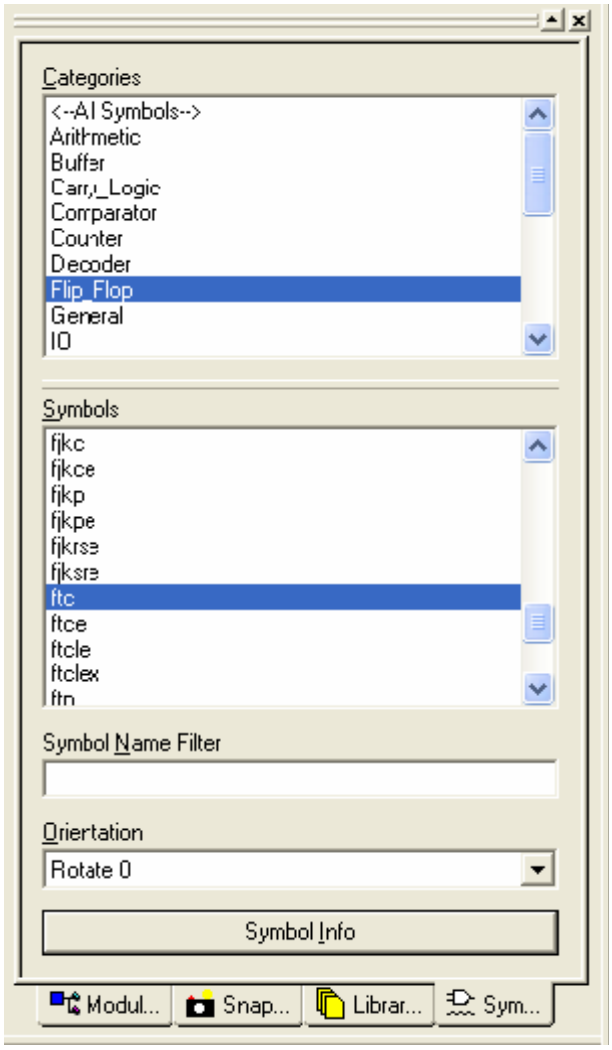


Επιλέγοντας το σχηματικό αρχείο, με δεξί κλικ, ανοίγει το σχηματικό περιβάλλον εργασίας, πάνω στο οποίο θα σχεδιάσουμε το κύκλωμα που θέλουμε.

5.3 ΤΟΠΟΘΕΤΗΣΗ ΥΛΙΚΩΝ



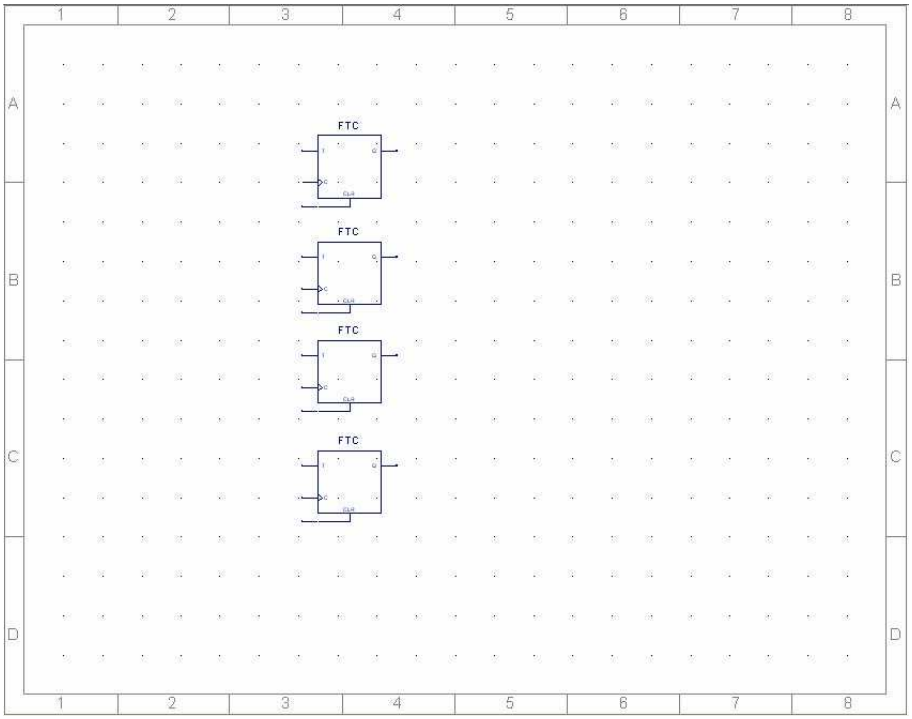
Κλικάρουμε στην επιλογή  και εμφανίζεται η λίστα με τα υλικά τα οποία έχουμεστη διάθεση μας για να σχεδιάσουμε το κύκλωμα.



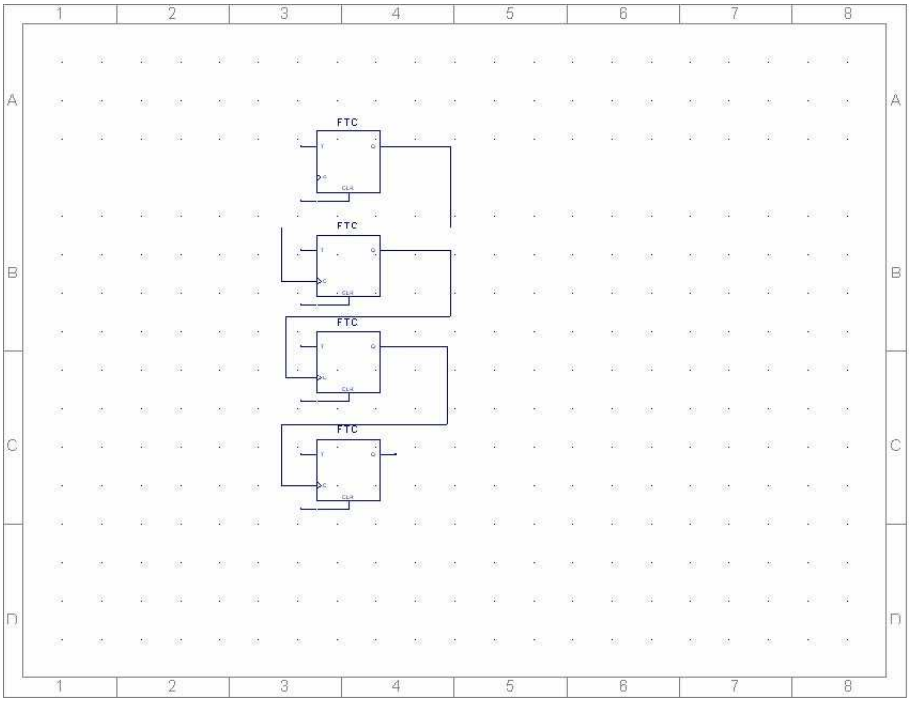
Επειδή τα υλικά είναι πάρα πολλά μας δίνεται η δυνατότητα να δούμε τη λειτουργία κάθε υλικού αν πατήσουμε το κουμπί ‘’Symbol Info’’.Τότε μας ανοίγει ένα αρχείο το οποίο περιγράφει τη λειτουργία του συγκεκριμένου υλικού που επιλέξαμε. Για τη δική μας περίπτωση το επιλεγμένο υλικό είναι το ’ftc’’. Ανοίγει ένα αρχείο pdf το οποίο περιέχεται μέσα στο ήδη εγκατεστημένο πρόγραμμα.

5.4 Τοποθέτηση Συνδέσεων

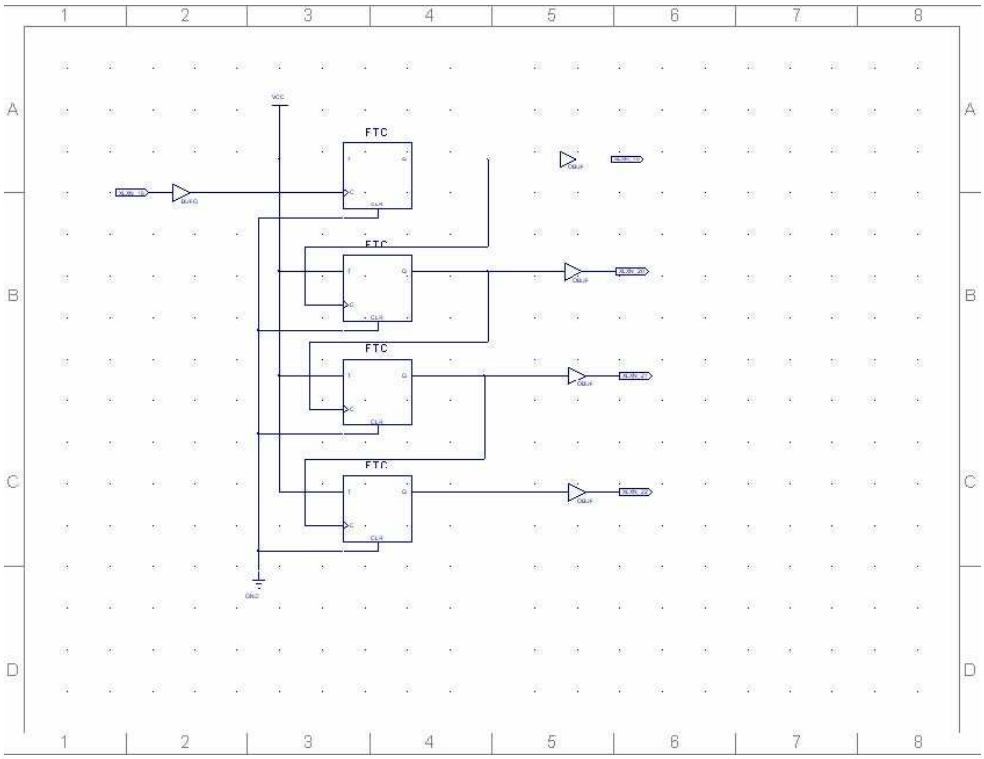
Αφού τοποθετήσουμε τα υλικά πάνω στην επιφάνεια εργασίας θα πρέπει να ορίσουμε και τις συνδέσεις μεταξύ των υλικών .



Κάνοντας κλίκ στο  μας δίνεται η δυνατότητα να δημιουργήσουμε συνδέσεις μεταξύ των υλικών.

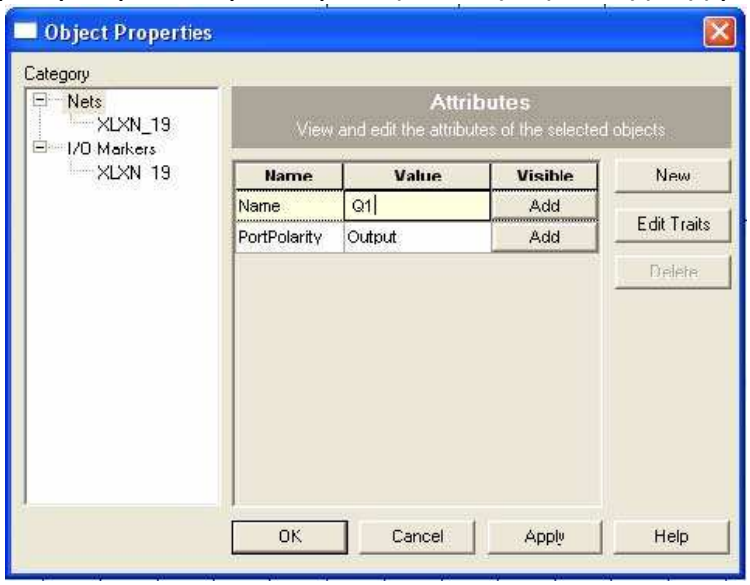


Το τελικό κύκλωμα θα πρέπει να είναι:

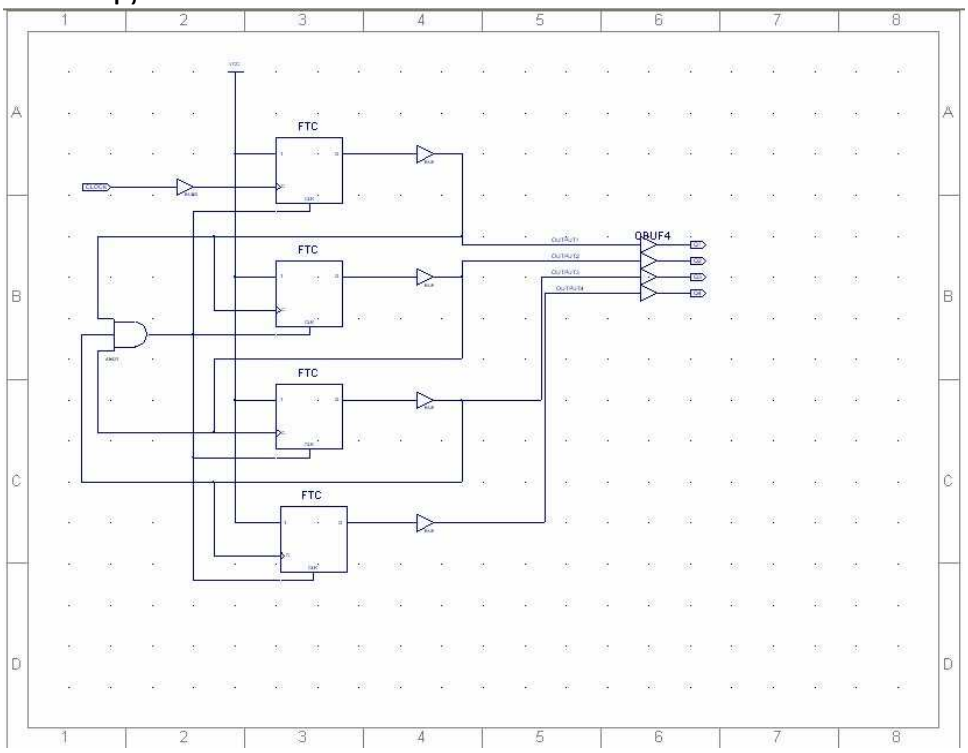


Πριν προχωρήσουμε παρακάτω θα πρέπει να τονίσουμε ότι θα πρέπει επίσης να ορίσουμε και τις μονάδες Buffers αλλά και τις μονάδες εισόδου/εξόδου του συστήματος που σχεδιάσαμε. Επίσης είναι αρκετά χρήσιμο και βολικό, όπως θα δούμε στη συνέχεια , να δώσουμε ονομασίες στις διάφορες συνδέσεις , προκειμένου να μη

μπερδευτούμε. Επίσης θα πρέπει να τονιστεί εδώ ότι η μονάδα ‘’bufg’’ εξυπηρετεί μόνο το παλμό χρονισμού. Για τις μονάδες εισόδου/εξόδου χρησιμοποιούμε το κουμπί  το οποίο μας δίνει τη δυνατότητα να τοποθετήσουμε μία μονάδα I/O στο σύστημα . Κάνοντας κλικ πάνω στο I/O marker μπορούμε να ορίσουμε τις ιδιότητες αυτής της μονάδας.



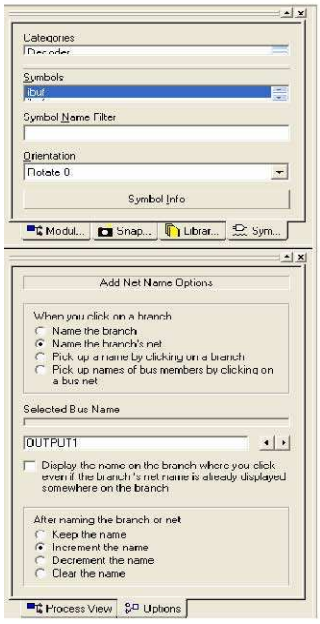
Πληκτρολογώντας το επιθυμητό όνομα στο αντίστοιχο πεδίο και πατώντας ‘’Apply’’ και ‘’OK’’ ορίζουμε τα ονόματα τα οποία θέλουμε. Να υπενθυμίσουμε εδώ ότι είναι φρόνιμο να δίνουμε ονόματα τέτοια ώστε να έχουμε μία σύντομη περιγραφή της λειτουργίας τις μονάδας ή της σύνδεσης.

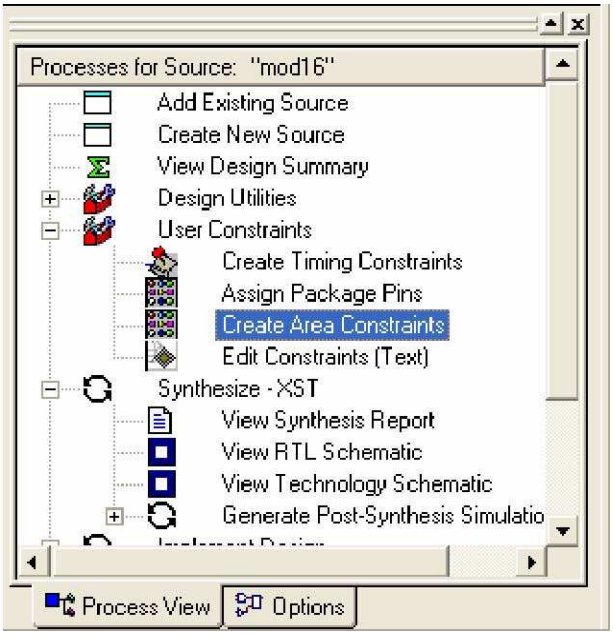


Το ίδιο επαναλαμβάνουμε και στις συνδέσεις τις οποίες έχουμε δημιουργήσει. Πατώντας το κουμπί  μπορούμε να ορίσουμε το όνομα μιας σύνδεσης. Στο πεδίο του ονόματος δίνουμε το όνομα που επιθυμούμε και στη συνέχεια κλικάρουμε πάνω στο όνομα της σύνδεσης που θέλουμε. Το κύκλωμα θα έχει πάρει αυτή τη μορφή.

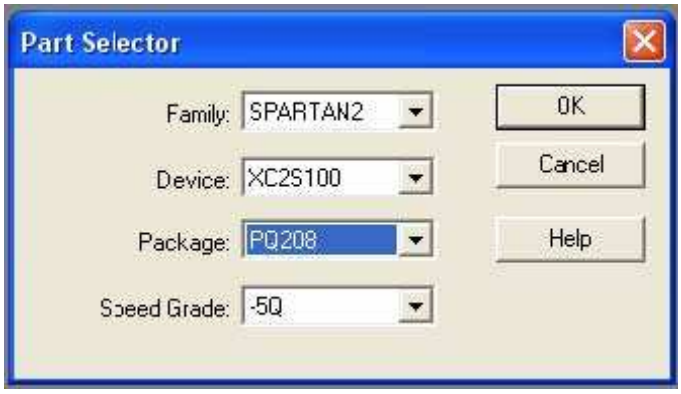
5.5 Ορισμός Ακίδων

Τελειώνοντας την επεξεργασία του σχηματικού θα πρέπει να ορίσουμε και τις ακίδες Εισόδου/Εξόδου με βάση το αναπτυξιακό το οποίο έχουμε στη διάθεση μας. Για αυτό το σκοπό το πρόγραμμα μας παρέχει και κάποιες άλλες λειτουργίες.

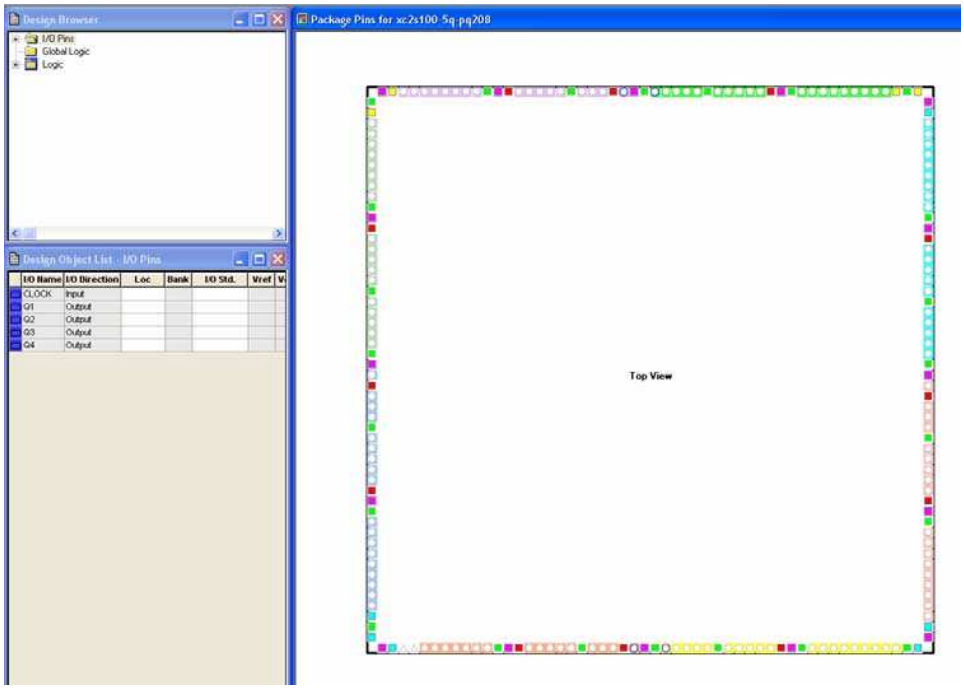




Πηγαίνουμε στην επιλογή ‘’Create Area Constrains’’ όπου θα ορίσουμε τις ακίδες εισόδου και εξόδου στο FPGA . Κλικάροντας μας ανοίγει ένα καινούργιο περιβάλλον εργασίας. Στο πρώτο παράθυρο που μας εμφανίζει ορίζουμε πάλι τα στοιχεία το FPGA που θα χρησιμοποιήσουμε.



Κατόπιν μας εμφανίζει το FPGA από την άποψη των ακίδων.

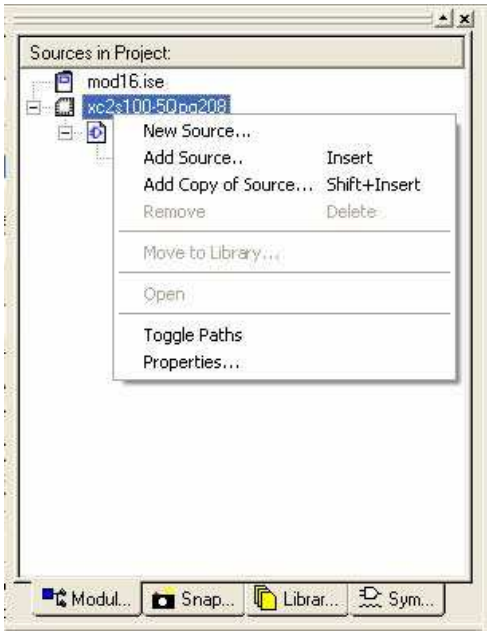


Ο ορισμός των ακίδων γίνεται στο πεδίο ‘’Design Object Line”, στο πεδίο Loc δηλώνουμε τις ακίδες που επιθυμούμε να συνδέσουμε το κύκλωμα μας δίνοντας το με τη μορφή ‘’Pxxx’’. Όπου xxx ο αριθμός της ακίδας που θέλουμε , Εδώ θα πρέπει να λάβουμε υπόψη ότι δε μπορούμε να χρησιμοποιήσουμε όποια ακίδα θέλουμε, και ότι περιοριζόμαστε αφενός από το ίδιο το FPGA (ακίδες ειδικής χρήσης, προγραμματισμός Ο.Κ κλπ), αφετέρου δε από την αναπτυξιακή πλακέτα.

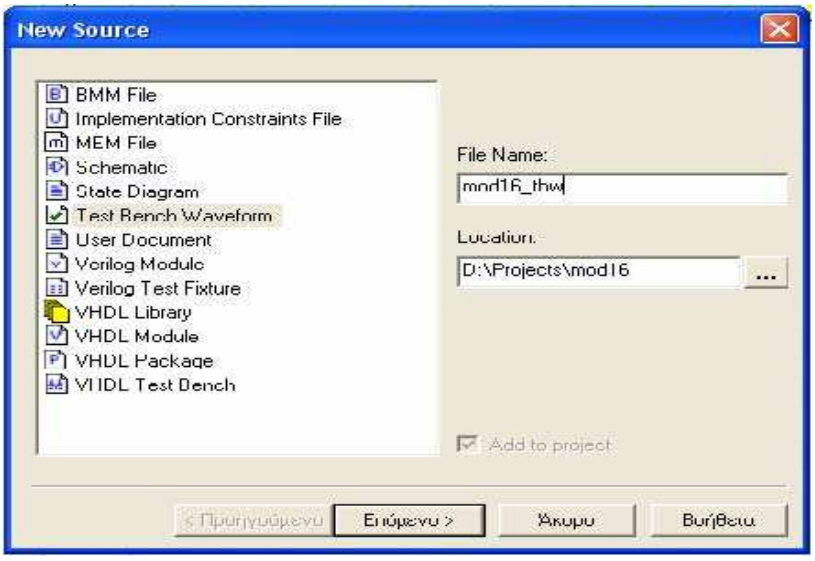
5.6 Εξομοίωση της λειτουργίας

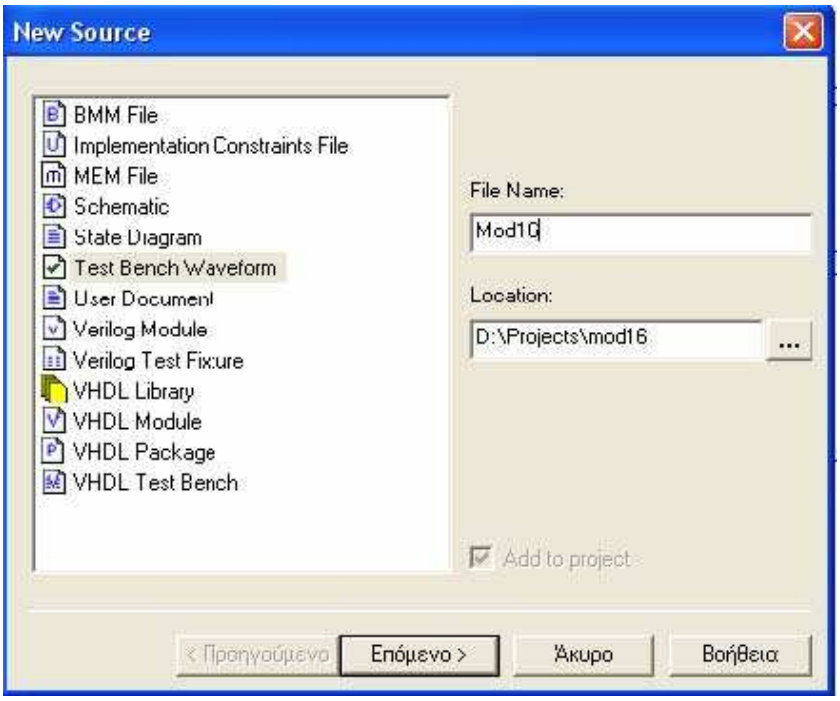
Τελειώνοντας με τους ορισμούς των ακίδων εισόδου/εξόδου θα πρέπει να ελέγξουμε ότι το κύκλωμα συμπεριφέρεται ακριβώς έτσι όπως επιθυμούμε. Για το σκοπό αυτό υπάρχει εγκατεστημένο και το πρόγραμμα εξομοίωσης Modelsim με το οποίο θα κάνουμε και τον τελικό έλεγχο του κυκλώματος μας.

Για το σκοπό αυτό θα πρέπει να δημιουργήσουμε ένα ειδικό αρχείο το οποίο ονομάζεται Test bench file όπου θα περιέχονται όλα τα δεδομένα τα οποία χρειάζονται για την εξομοίωση. Κάνουμε δεξί κλικ στην περιοχή “Sources in Project” και επιλέγουμε “New source”.

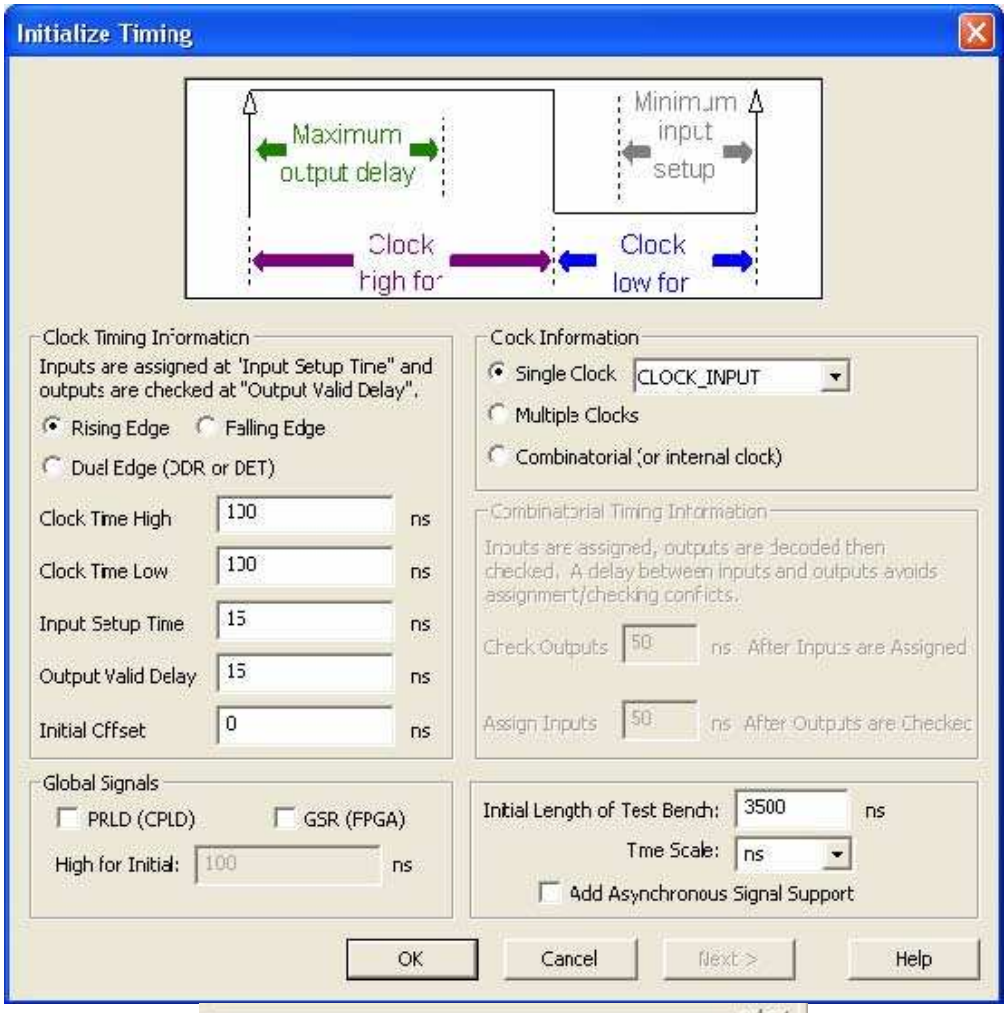


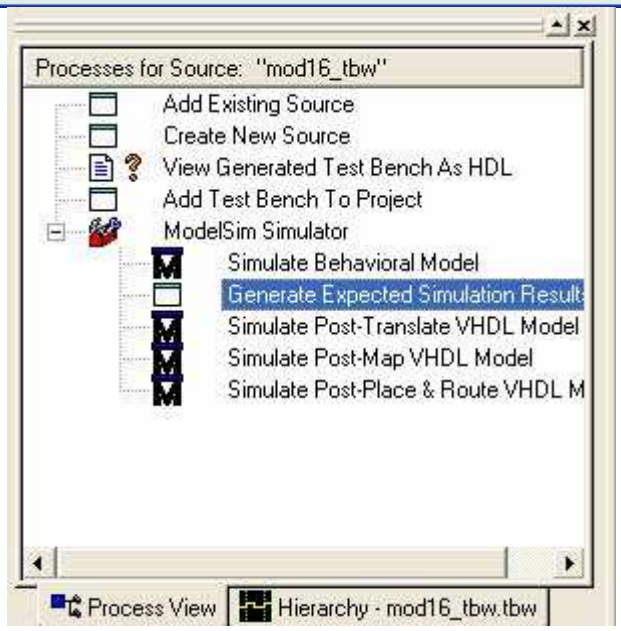
Ανοίγει ένα καινούργιο παράθυρο στο οποίο θα δηλωθεί ότι θέλουμε να δημιουργήσουμε ένα αρχείο εξομοίωσης. Σε αυτό το παράθυρο δηλώνουμε ότι θέλουμε να δημιουργήσουμε ένα “Test Bench Waveform” αρχείο και το όνομα του αρχείου. Φρόνιμο είναι να δώσουμε το ίδιο όνομα αλλά να προσθέσουμε και _tbw το οποίο θα μας υποδηλώνει ότι πρόκειται για αρχείο εξομοίωσης. Στη συνέχεια πατάμε το κουμπί ‘’Επόμενο’’ και μας εμφανίζει δύο ακόμα παράθυρα τα οποία είναι κάποια συγκεντρωτικά στοιχεία για τις ρυθμίσεις που έχουμε κάνει.





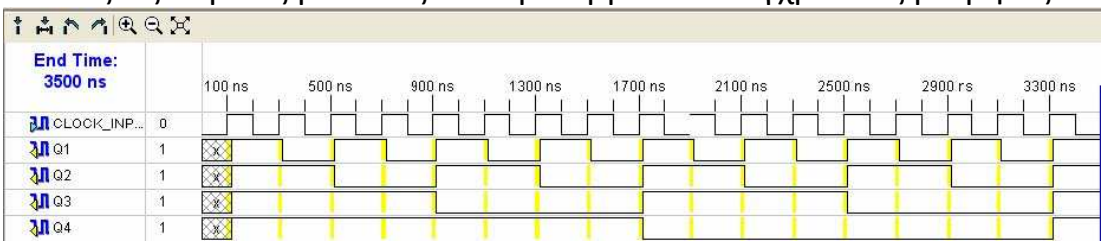
Πατώντας το κουμπί ‘’Επόμενο’’ κάθε φορά που μας το ζητάει θα μας ανοίξει τελικά το παράθυρο χρονισμού που θα πρέπει να δώσει τις αρχικές τιμές στους παλμούς χρονισμού που θέλουμε να χρησιμοποιήσουμε Σε αυτό το παράθυρο ορίζουμε του διάφορους τύπους χρονισμού που μας δίνεται η δυνατότητα να χρησιμοποιήσουμε, στο πεδίο “Clock Information”. Εδώ μπορούμε να επιλέξουμε αν θέλουμε μονό ή πολλαπλούς παλμούς χρονισμού, ή ακόμα και καθαρά συνδυαστική επιλογή προσομοίωσης. Όπως επίσης και το μέτωπο σκανδαλισμού (αρνητικό, θετικό ή ακόμα και διπλού μετώπου). Επίσης μπορούμε να ορίσουμε τη χρονική διάρκεια των σταθμών του παλμού χρονισμού, όπως επίσης και το συνολικό χρόνο εξομοίωσης. Στη συνέχεια μπορούμε να έχουμε ένα “Preview” των εξόδων πηγαίνοντας στην επιλογή, όπου το πρόγραμμα θα κάνει μία πρώτη ανάλυση των σημάτων και θα μας δώσει τις εξόδους.



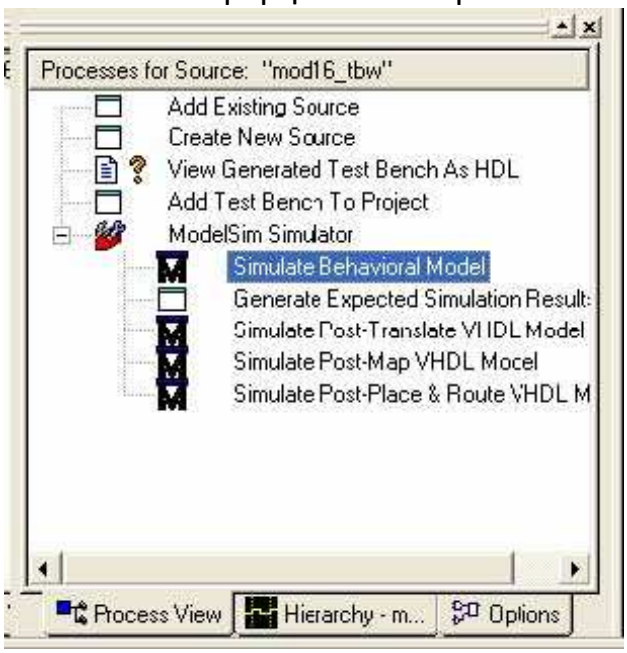


Εδώ θα πρέπει να τονιστεί ιδιαίτερα το γεγονός ότι ενώ ο μετρητής είναι σχεδιασμένος σαν αύξων τα αποτελέσματα των εξόδων δείχνουν ότι συμπεριφέρεται σαν φθίνων. Αυτό συμβαίνει λόγω της μεγάλης

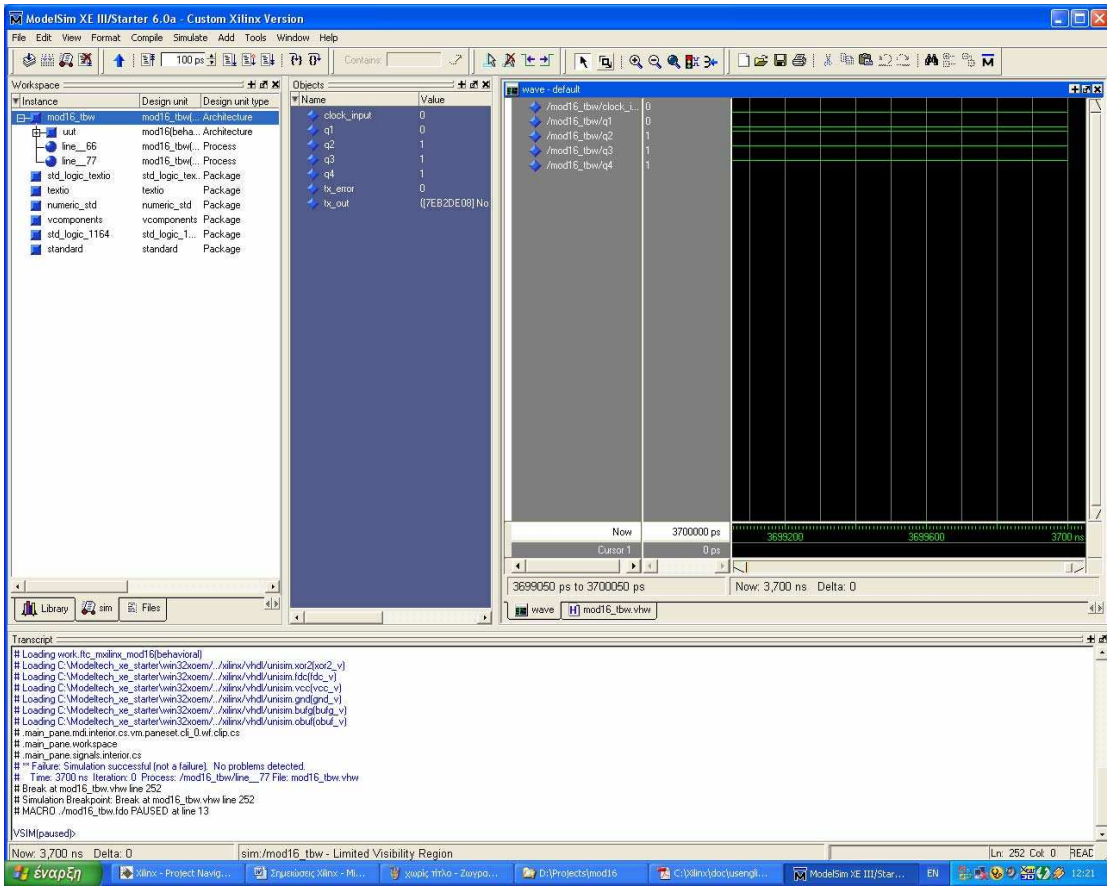
ταχύτητας αντίδρασης που έχουν τα υλικά των FPGA με αποτέλεσμα να χρονίζονται όλα ταυτόχρονα και για το λόγο αυτό και η ίδια η εταιρεία σε όλες τις δομικές μονάδες δε περιλαμβάνει ασύγχρονους μετρητές.



Για να μπορέσουμε να κάνουμε όμως πλήρη και εμπεριστατωμένη εξομοίωση θα πρέπει να ανατρέξουμε στο πρόγραμμα “Modelsim” το οποίο θα μας δώσει πιο λεπτομερή αποτελέσματα .

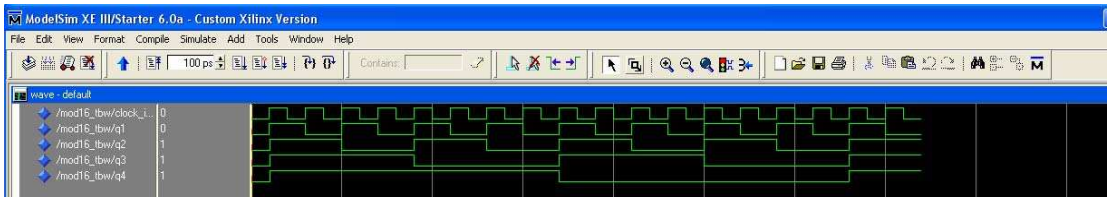


Για να γίνει αυτό επιλέγουμε το “Simulate Behavioral Model” από το ίδιο παράθυρο επιλογών και μας ανοίγει ένα καινούργιο περιβάλλον εργασίας.



Από όλα αυτά τα παράθυρα μας ενδιαφέρει το “wave” το οποίο απεικονίζει όλα τα σήματα τα οποία μας ενδιαφέρουν , και όχι μόνο τις εισόδους και εξόδους που δημιουργεί το προηγούμενο πρόγραμμα εξομοίωσης που αναφέραμε.

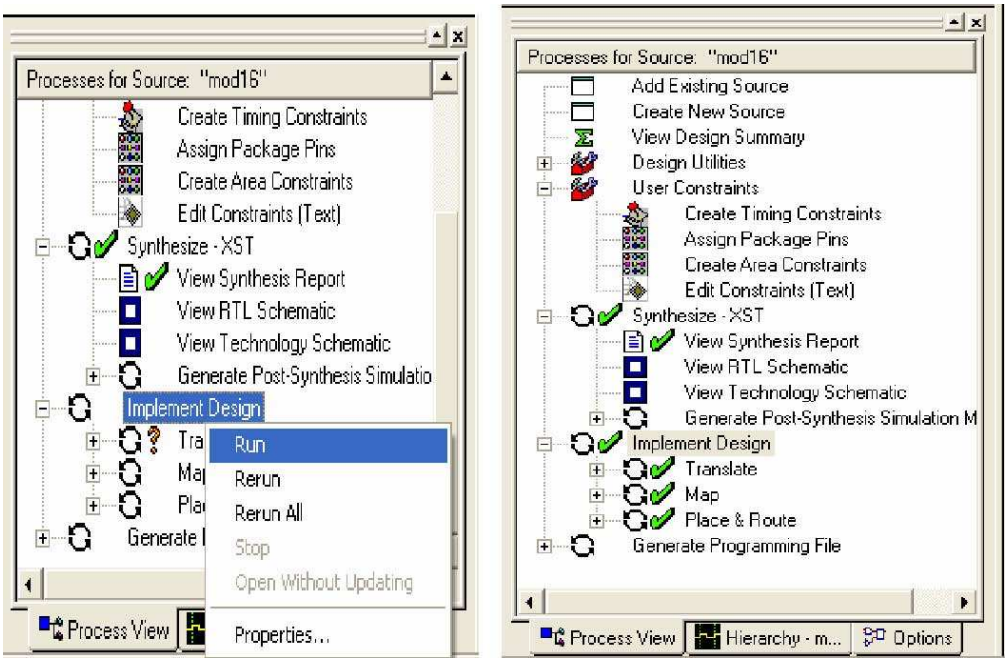
Πατώντας το κουμπί  κάνουμε “Zoom In” στο κυματοσυρμό ώσπου να δούμε ολόκληρο το κυματοσυρμό.



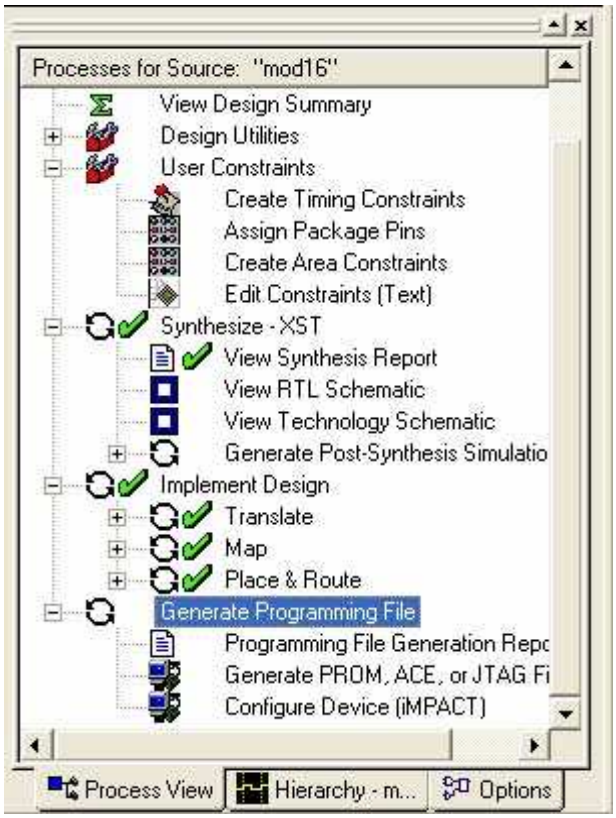
Αυτό που διαπιστώνουμε είναι ότι ο μετρητής όντως συμπεριφέρεται σαν φθίνων μετρητής, για του λόγους που αναλύσαμε παραπάνω, αλλά από άποψη χρονισμού συμπεριφέρεται σωστά.

5.7 Προγραμματισμός του FPGA

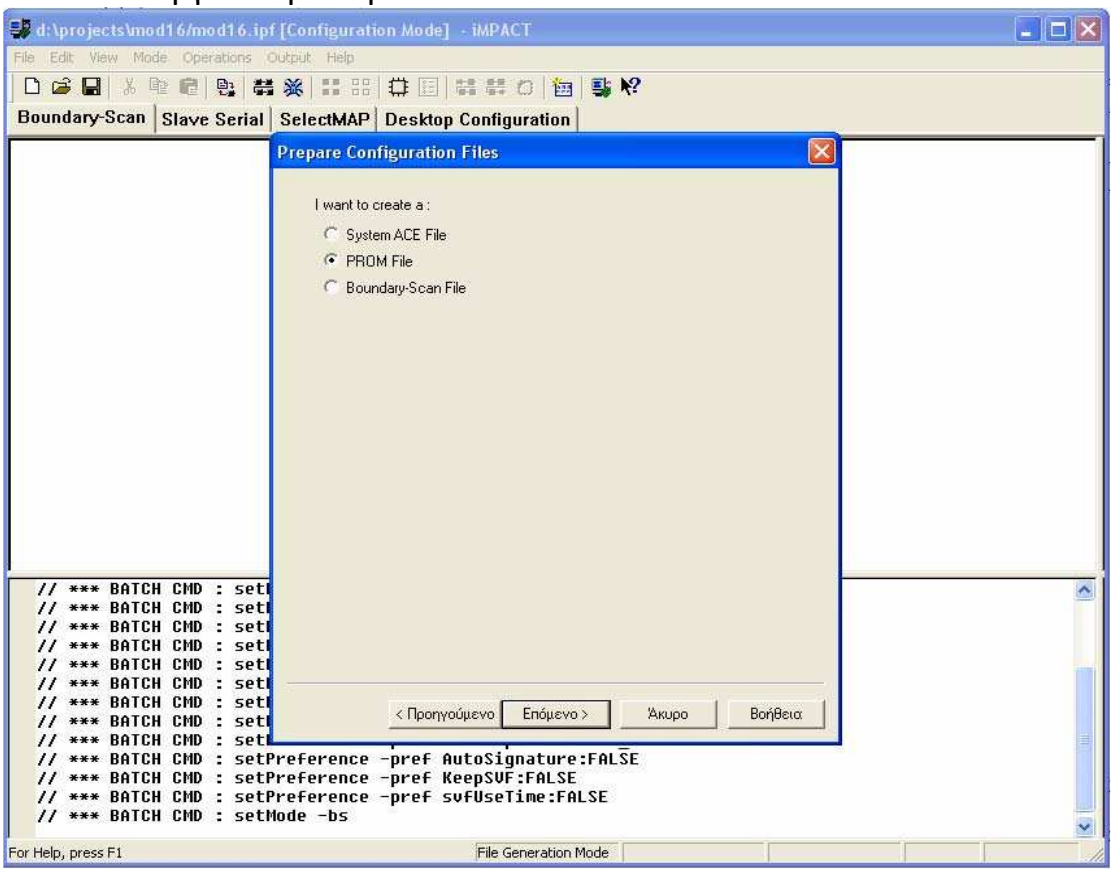
Ο προγραμματισμός του FPGA θα γίνει μέσω της αναπτυξιακής πλακέτας και του συνδετήρα JTAG ο οποίος θα κάνει και το κύριο προγραμματισμό. Το Xilinx 7.1 όμως μας δίνει και τη δυνατότητα εξαγωγής του σχηματικού μας κυκλώματος σε Jedec αρχείο το οποίο και θα αποτελεί το κύριο δυαδικό κώδικα προγραμματισμού του FPGA. Για τη διαδικασία αυτή θα πρέπει να αναφερθεί ότι υπάρχει η δυνατότητα εξαγωγής κώδικα για απευθείας προγραμματισμό του FPGA ,ή μέσω σειριακής μνήμης. Για τη δημιουργία το δυαδικού κώδικα προγραμματισμού επιλέγουμε το Η διαδικασία θα έχει ολοκληρωθεί σωστά όταν όλα τα ερωτηματικά γίνουν πράσινα

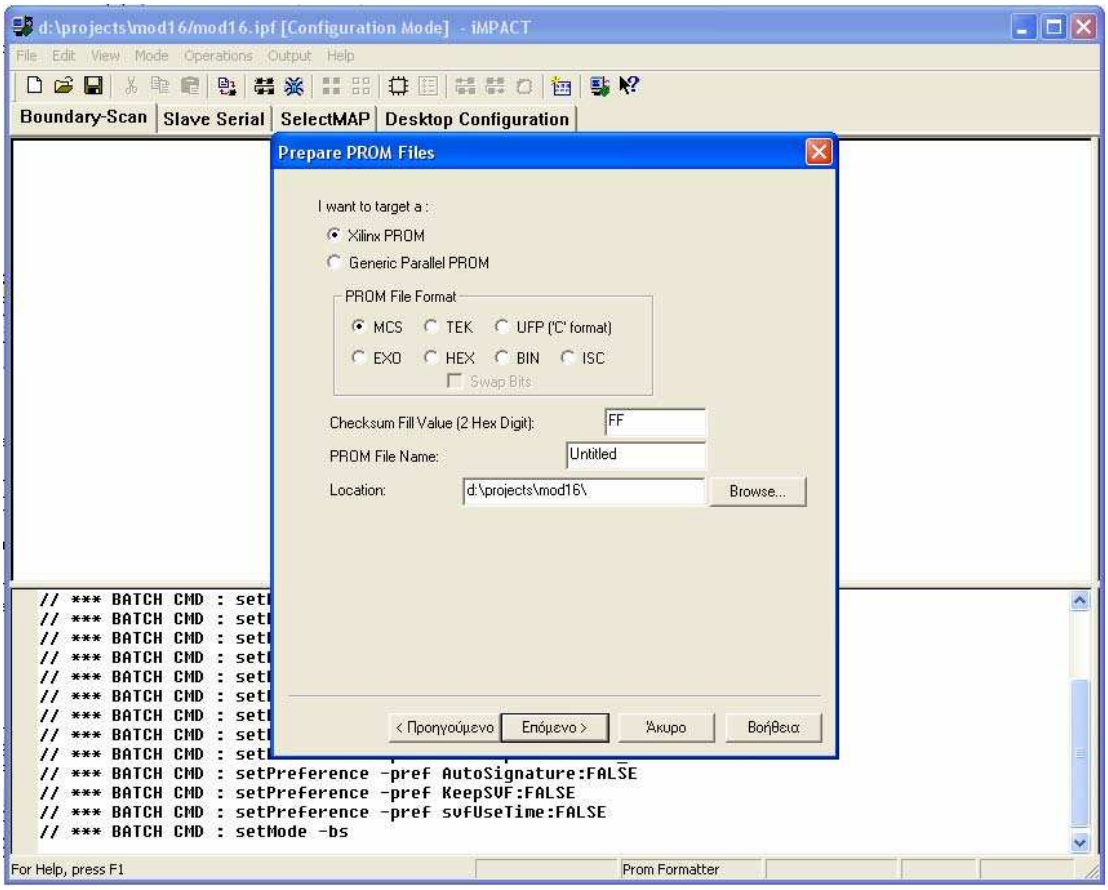


Με την ολοκλήρωση και της διαδικασίας μετάφρασης μπορούμε να προχωρήσουμε στη δημιουργία του αρχείου προγραμματισμού επιλέγοντας το “Generate Programming File” το οποίο και δημιουργεί το αρχείο.

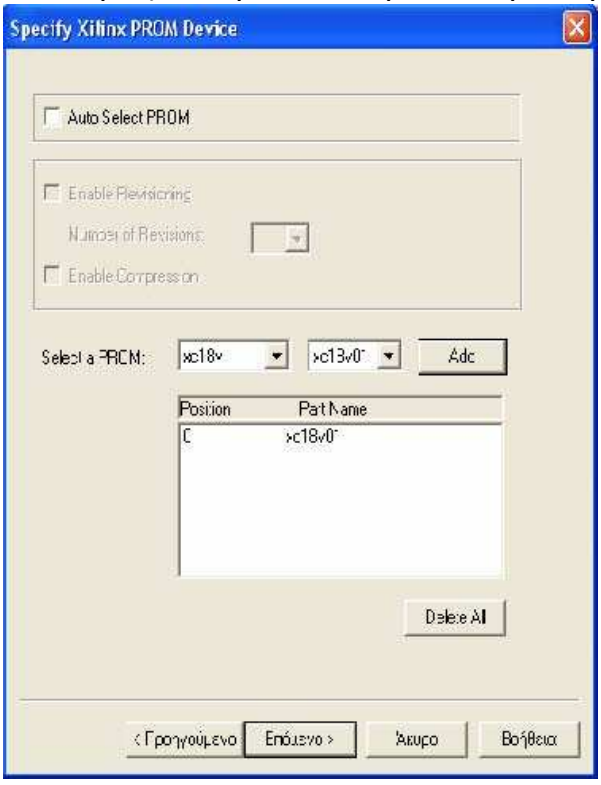


Για το "κατέβασμα" (downloading) του προγράμματος στο FPGA επιλέγουμε "Generate PROM,ACE,or JTAG File", όπου και μας ανοίγει ένα καινούργιο παράθυρο.

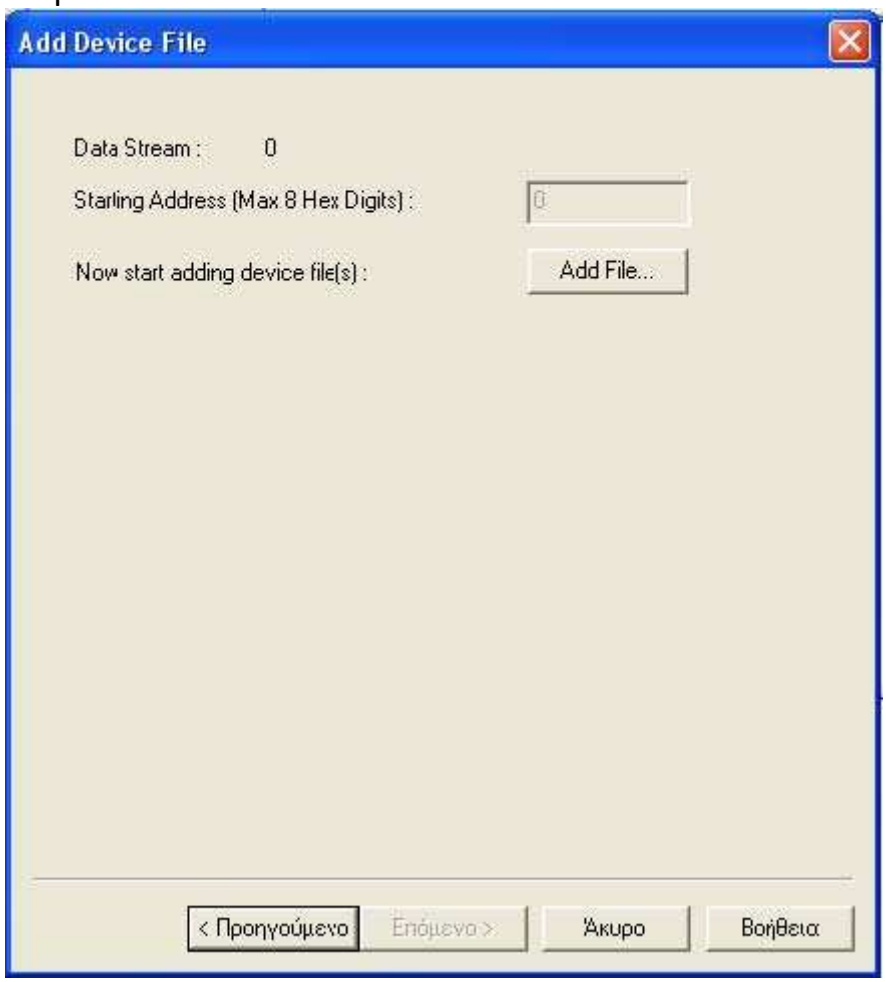




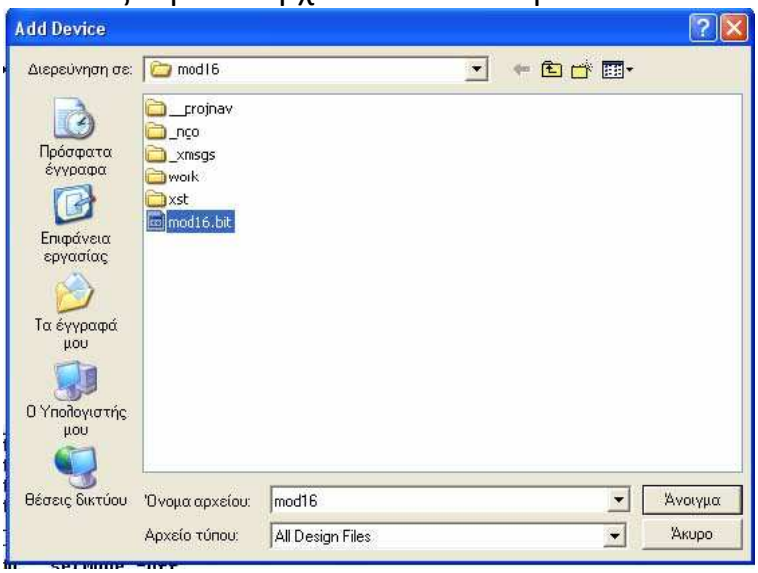
το επόμενο παράθυρο όπου επιλέγουμε το Format του αρχείου που θέλουμε να φορτώσουμε. Εμείς επιλέγουμε το MCS και πατάμε στο κουμπί ''Επόμενο'' και μας ανοίγει το επόμενο παράθυρο.



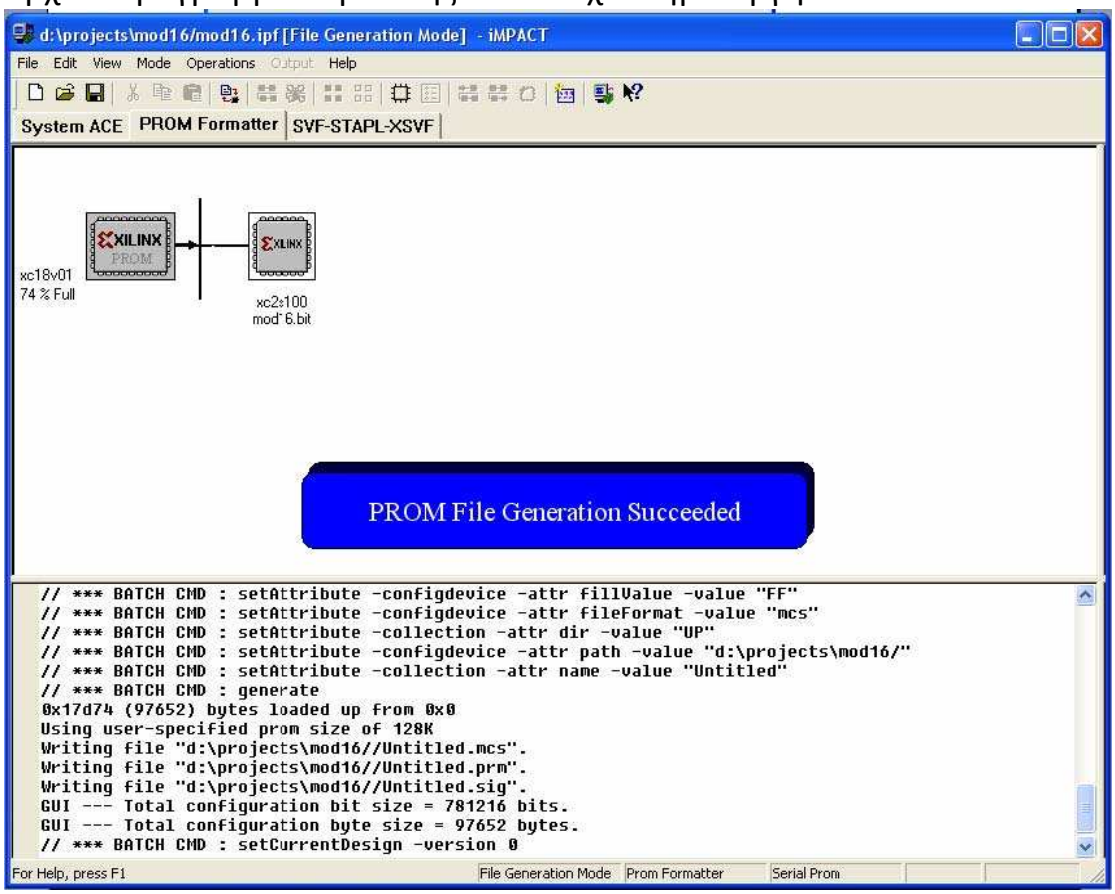
Εδώ επιλέγουμε τη μνήμη που υπάρχει πάνω στην αναπτυξιακή πλακέτα και η οποία είναι η Xilinx XC18v01, πατάμε μετά στο κουμπί “Add” και ‘’Επόμενο’’. Στο τέλος μας ανοίγει ένα παράθυρο εξερεύνησης από όπου θα επιλέξουμε το αρχείο που θέλουμε να φορτώσουμε .



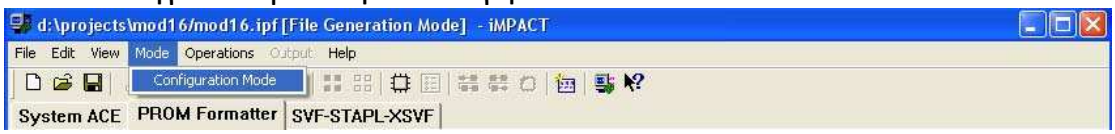
Πατώντας στο κουμπί “Add File” μας ανοίγει το παρακάτω παράθυρο από όπου θα επιλέξουμε το αρχείο που θέλουμε.



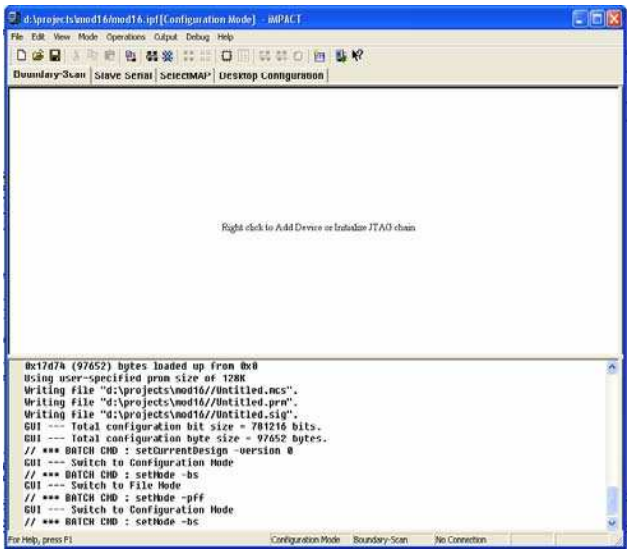
Τελικά θα πρέπει να πάρουμε αυτό το μήνυμα το οποίο μας λέει ότι το αρχείο προγραμματισμού της PROM έχει δημιουργηθεί.



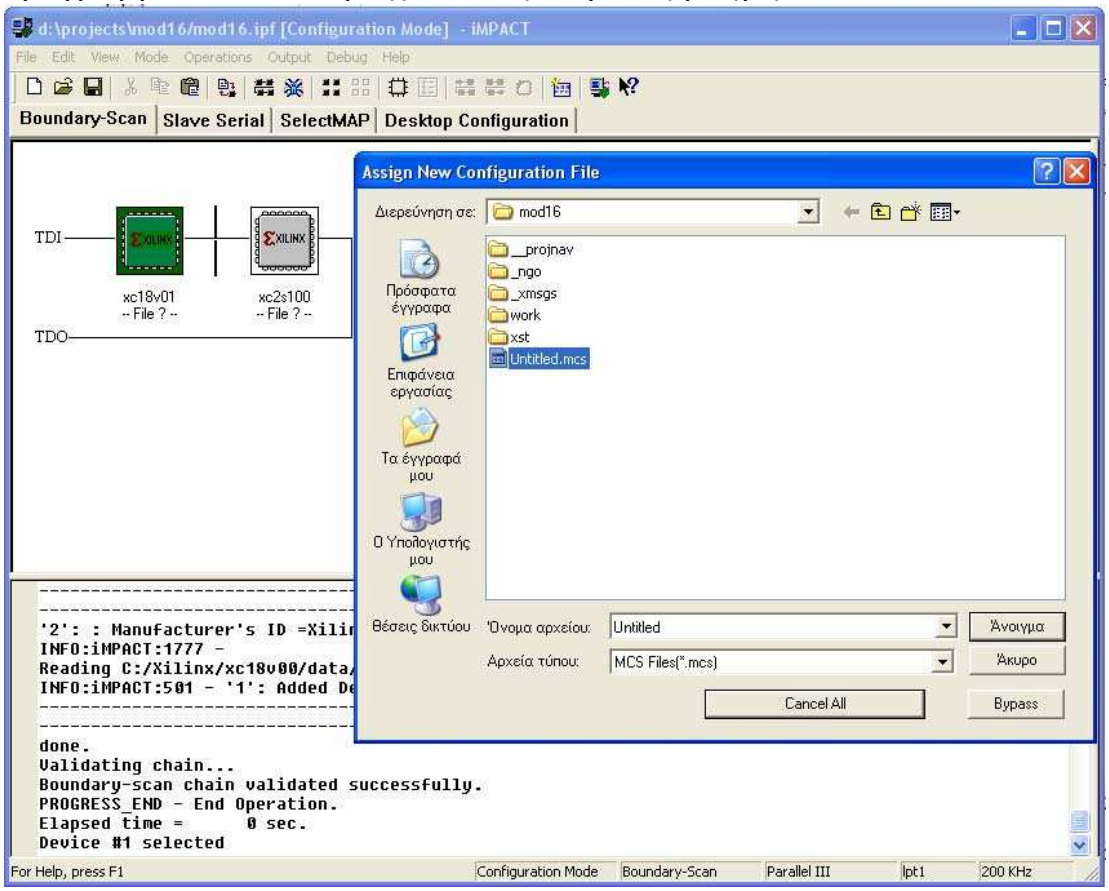
Έπειτα πηγαίνουμε στην επιλογή



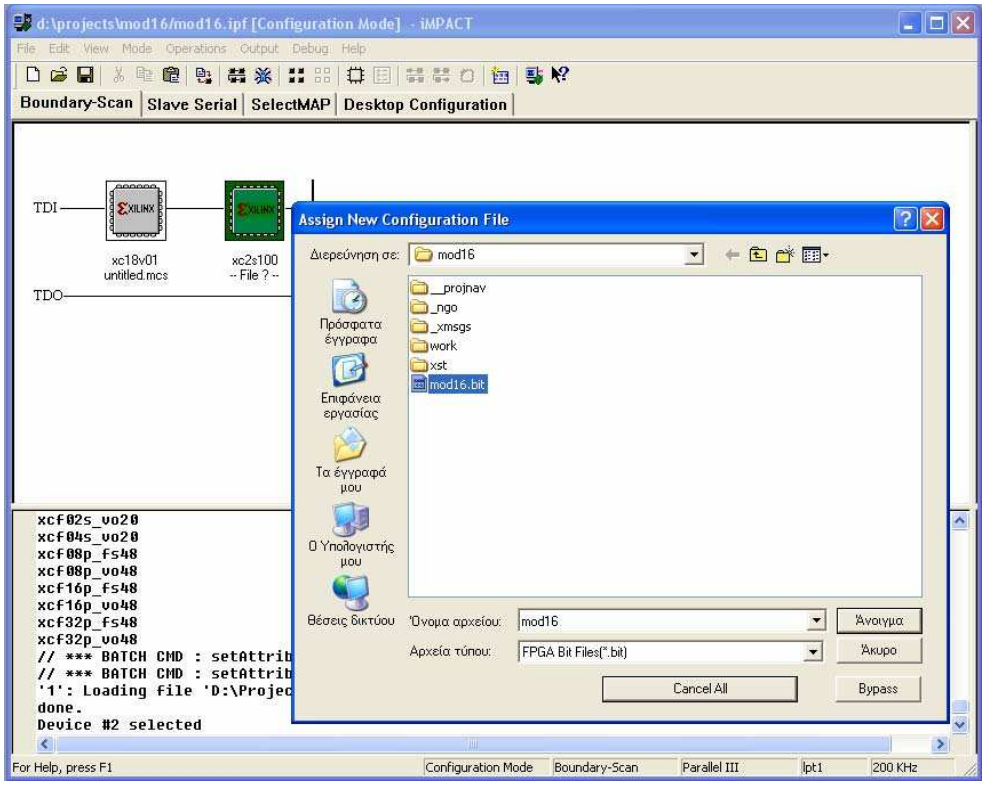
για να ενεργοποιήσουμε την αλληλουχία προγραμματισμού και μας εμφανίζει το παρακάτω παράθυρο.



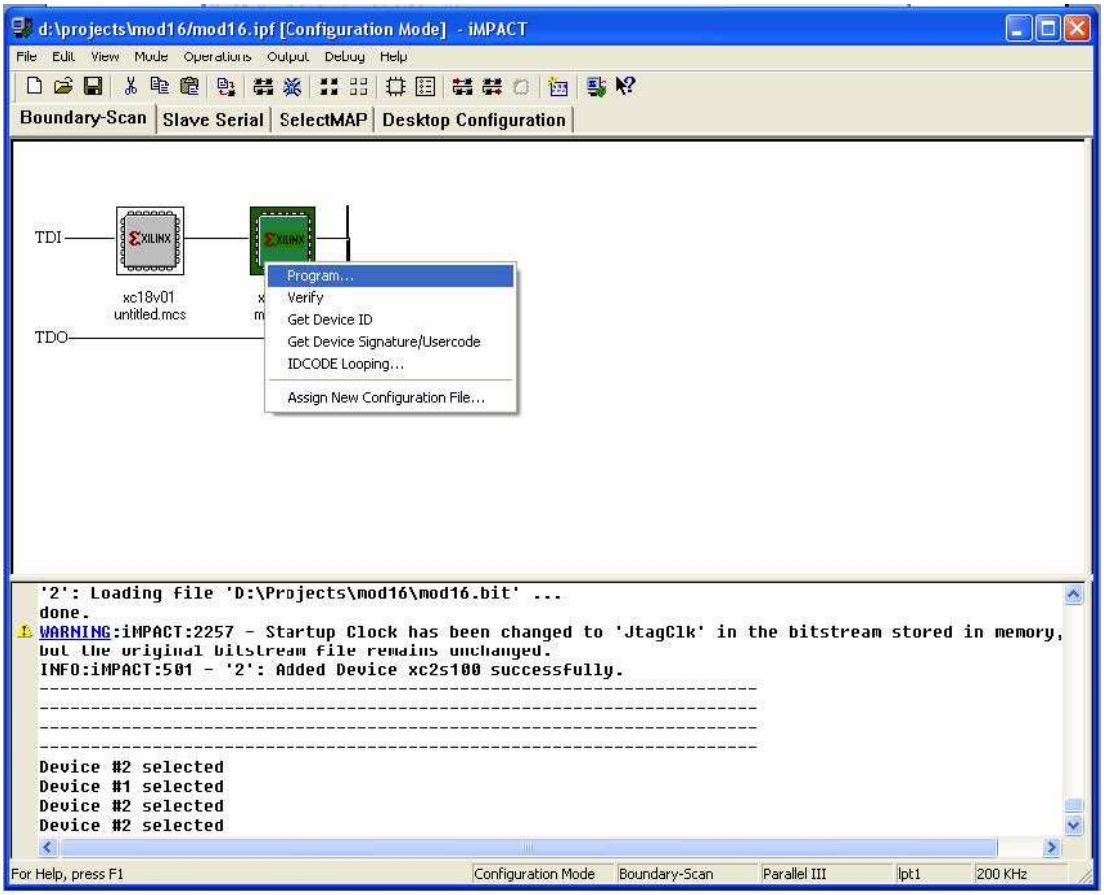
Κάνοντας δεξί κλικ επιλέγουμε την εντολή “Initialize Chain” η οποία θα ενεργοποιήσει την αλληλουχία προγραμματισμού, πρώτα για το πρόγραμμα που θα περιέχεται στη σειριακή μνήμη.

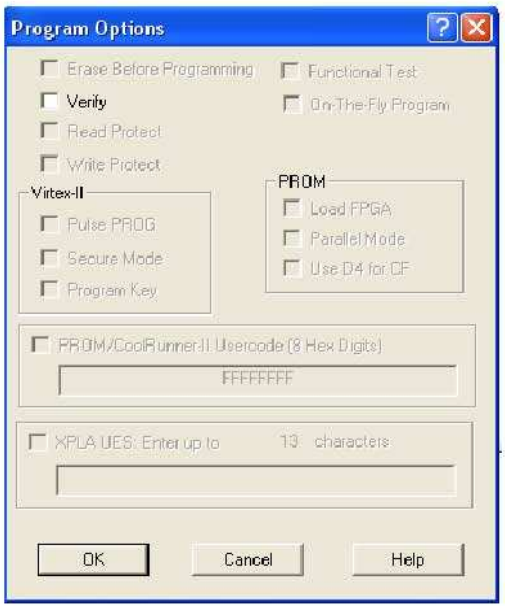


Και έπειτα για το πρόγραμμα που θα περιέχεται στο FPGA.

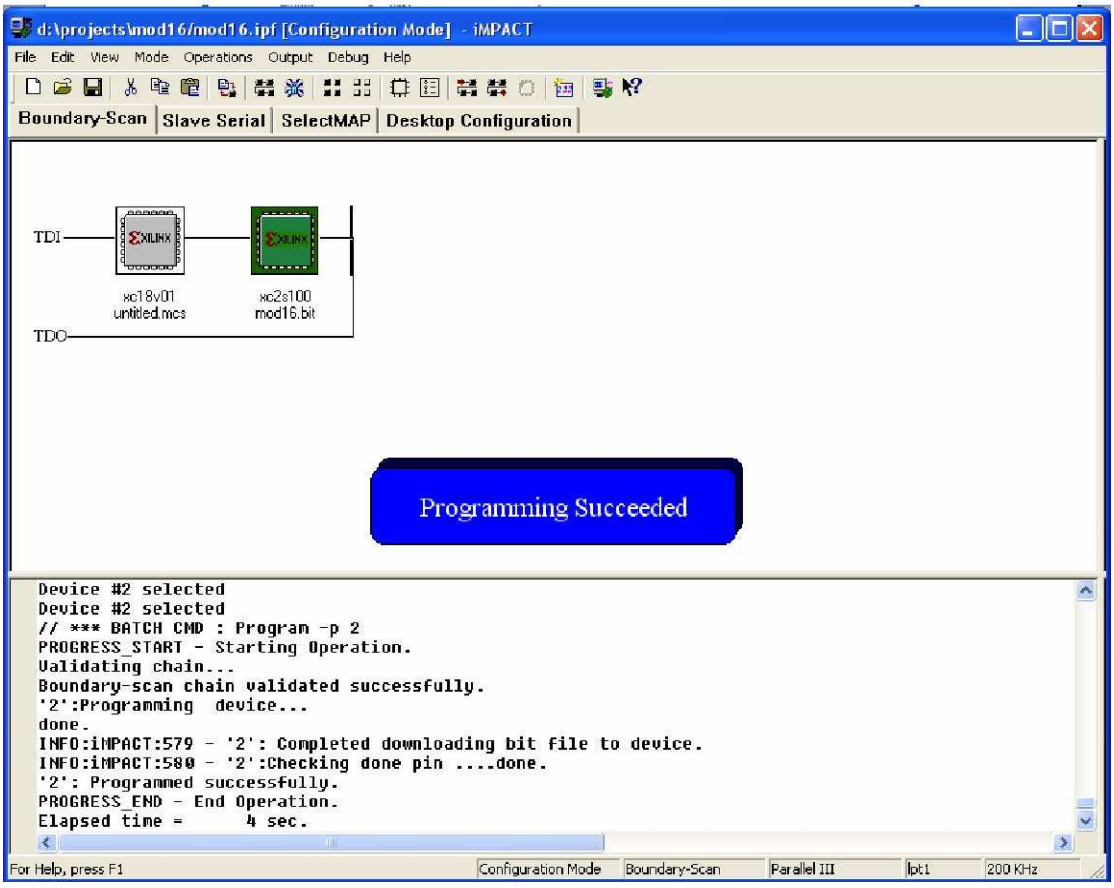


Τώρα μπορούμε να προγραμματίσουμε τη συσκευή που θέλουμε FPGA, ή μνήμη κάνοντας δεξί κλικ πάνω από το αντίστοιχο υλικό.





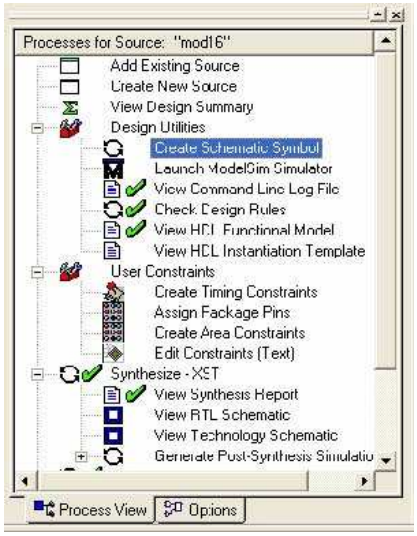
Και πατώντας “OK” αρχίζει η διαδικασία προγραμματισμού η οποία θα έχει αυτό το μήνυμα όταν θα είναι επιτυχημένη



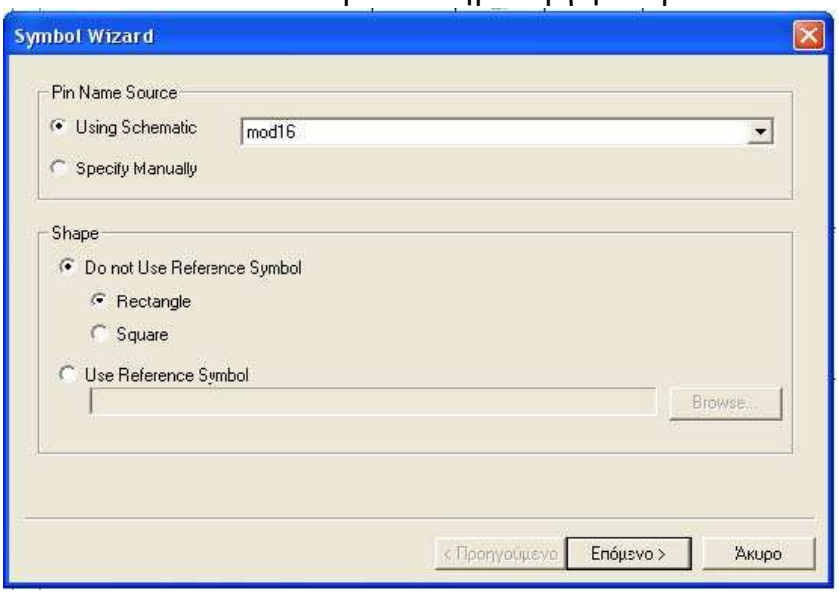
Εδώ θα πρέπει να τονιστεί ότι το πρόγραμμα έχει περάσει απευθείας στο FPGA και σε οποιαδήποτε περίπτωση πτώσης τάσης το πρόγραμμα αυτό θα έχει χαθεί. Για αυτό το λόγο υπάρχει και η σειριακή μνήμη. Σε περίπτωση που θέλουμε να φορτώσουμε το πρόγραμμα που περιείχε πιο πριν ή θα πρέπει να επαναλάβουμε τη διαδικασία προγραμματισμού, ή θα πρέπει να το γράψουμε και στη σειριακή μνήμη. Πατώντας το κουμπί “Program” το οποίο βρίσκεται πάνω στην αναπτυξιακή πλακέτα, φορτώνουμε το πρόγραμμα στο FPGA.

5.8 Δημιουργία αρχείων βιβλιοθήκης (CORE)

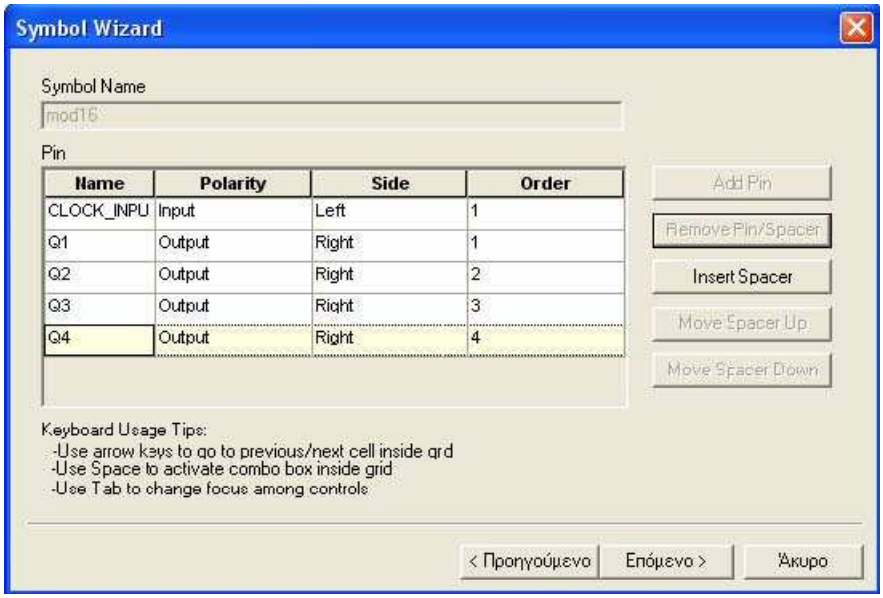
Η δημιουργία αρχείων βιβλιοθήκης είναι πάρα πολύ χρήσιμη ειδικά όταν σχεδιάζουμε μεγάλα και πολύπλοκα κυκλώματα τα οποία περιέχουνε βαθμίδες που χρησιμοποιούνται συχνά. Με τον τρόπο αυτό δημιουργούμε μία φορά το κύκλωμα και το χρησιμοποιούμε όταν και όποτε το χρειαζόμαστε. Για να μπορέσουμε να υλοποιήσουμε το αρχείο βιβλιοθήκης πρέπει πρώτα να ενεργοποιήσουμε τη διαδικασία από την επιλογή.



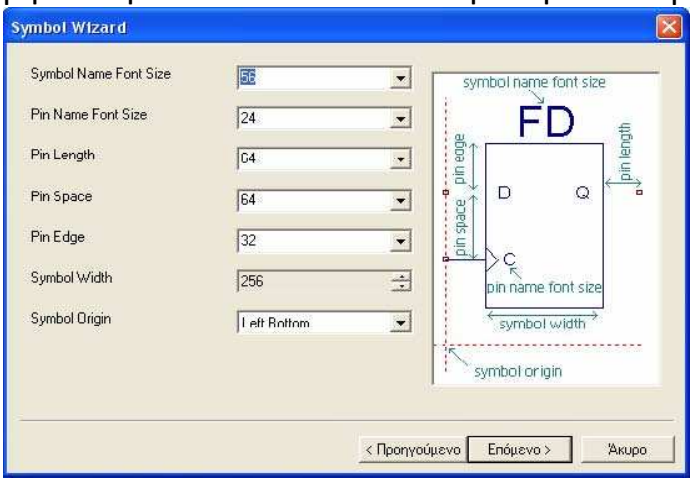
Τώρα πηγαίνοντας στο μενού εντολών και στην επιλογή **Tools-⇗Symbol Wizzard** μας ανοίγει ένα νέο παράθυρο στο οποίο θα δηλώσουμε τα στοιχεία του υλικου που θέλουμε να δημιουργήσουμε



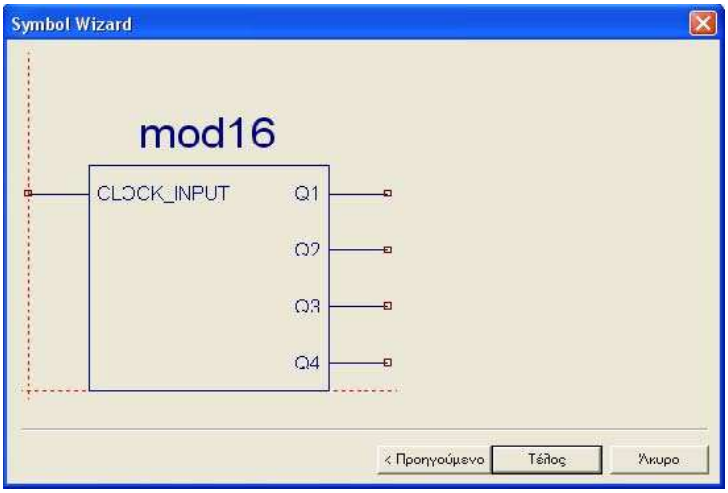
Πατώντας στο επόμενο μας εμφανίζει το παράθυρο εισόδων και εξόδων του συστήματος που θα δημιουργήσουμε.



Στη συνέχεια χρειάζεται να δηλώσουμε κάποια στοιχεία τα οποία αφορούν τη εμφάνιση του υλικού αυτού στη επιφάνεια εργασίας μας.



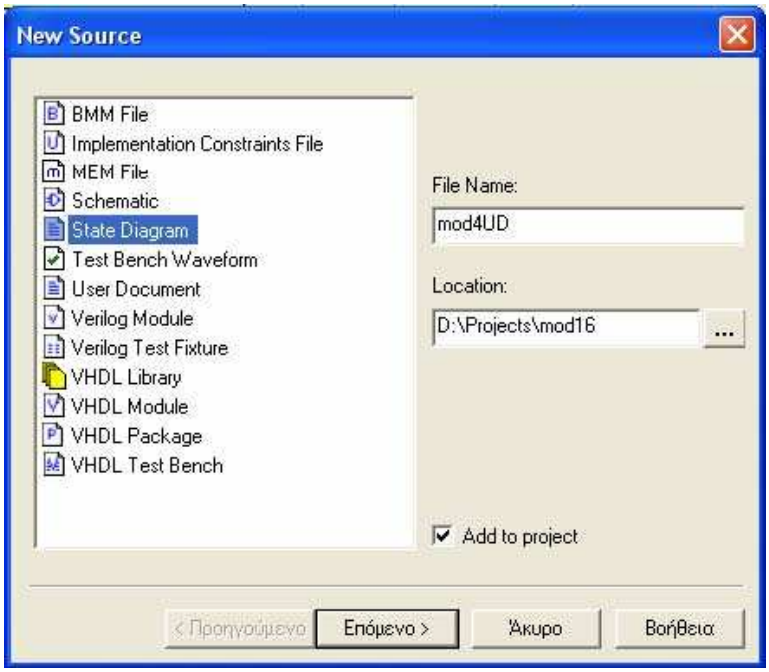
Και τελικά η δημιουργία του νέου μας υλικού.



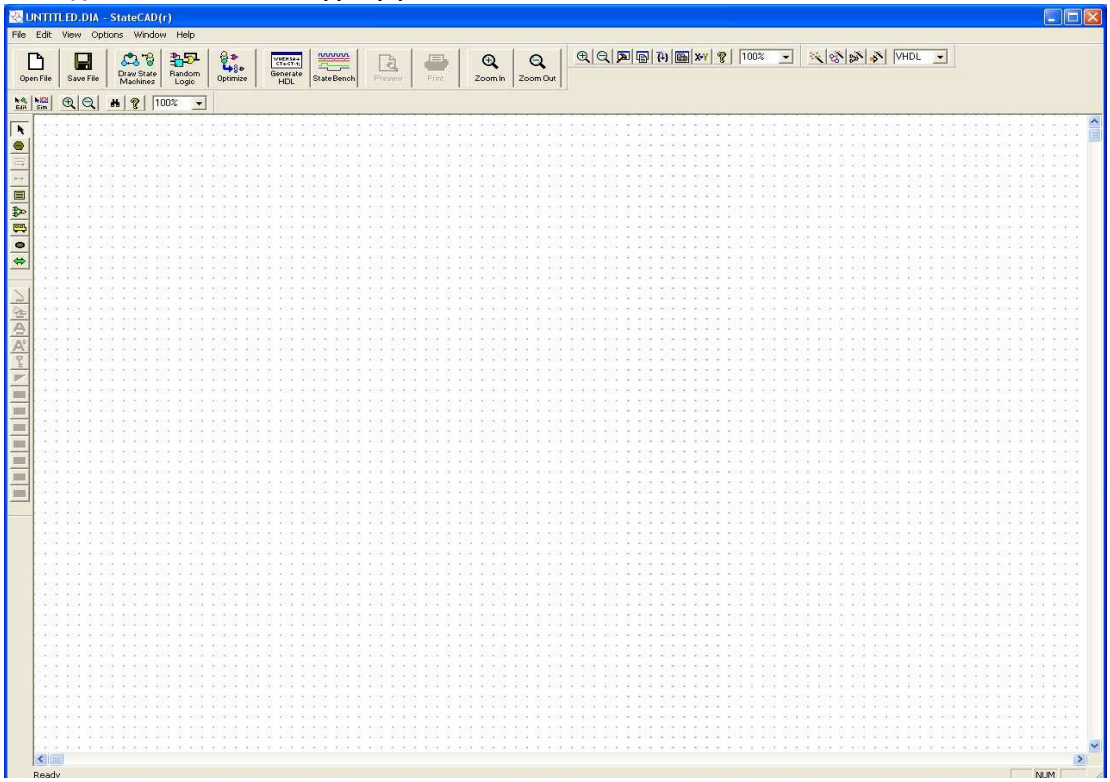
Τώρα το νέο μας υλικό έχει προστεθεί στα διαθέσιμα υλικά στη βιβλιοθήκη του σχηματικού και μπορούμε να το χρησιμοποιήσουμε όσες φορές θέλουμε χωρίς να χρειάζεται να το ξανασχεδιάσουμε.

5.9 Δημιουργία Μηχανής καταστάσεων

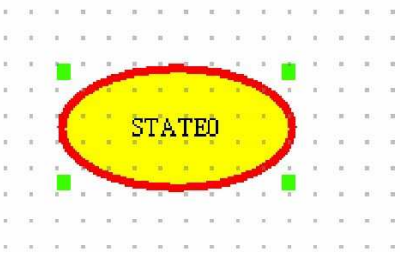
Το πακέτο Xilinx Webpack περιέχει επίσης και το πρόγραμμα StateCAD με το οποίο μπορούμε να σχεδιάζουμε μηχανές πεπερασμένων καταστάσεων, οι οποίες μπορούν να περιέχουν, καταστάσεις, εισόδους/εξόδους αλλά και συνθήκες μετάβασης. Το πρόγραμμα εξάγει το διάγραμμα σε γλώσσα VHDL και μπορούμε να το είσάγουμε σε μακροεντολή στο σχεδιαστικό μας περιβάλλον. Για να ξεκινήσουμε εκτελούμε τις εντολές **PROJECT→New Source** και μας ανοίγει το παράθυρο επιλογής του αρχείου που θέλουμε να δημιουργήσουμε .



Θα δημιουργήσουμε έναν αύξων/φθίνων μετρητή Mod4. Δηλώνουμε το όνομα και το είδος του αρχείου που θέλουμε να δημιουργήσουμε και πατάμε το κουμπί ‘’Επόμενο’’. Επειδή υπάρχει κάποιο πρόβλημα με τη έκδοση 8.3 του DOS θα εμφανίσει ένα μήνυμα λάθους το οποίο θα αναφέρει αυτό το πρόβλημα, και ότι θα πρέπει να ακολουθηθεί η διαδικασία **File Save As** για να δημιουργηθεί το αρχείο. Μας εμφανίζεται τώρα ένα καινούργιο περιβάλλον εργασίας πάνω στο οποί θα σχεδιαστεί το διάγραμμα καταστάσεων.

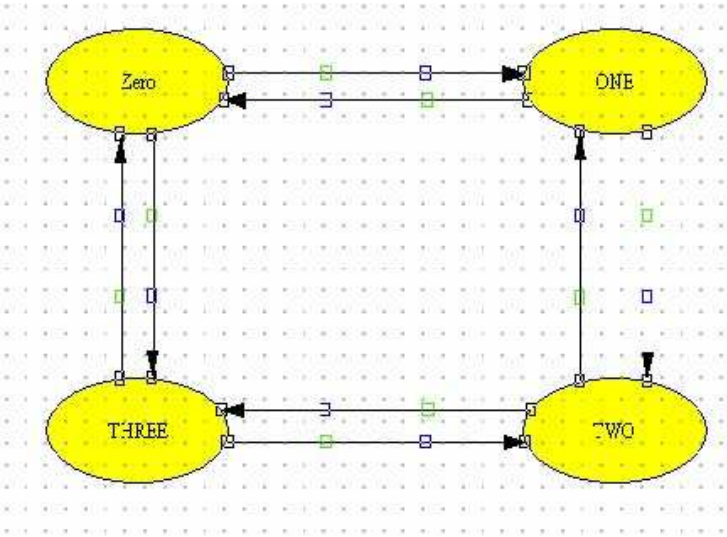


Για να τοποθετήσουμε τις καταστάσεις στην επιφάνεια εργασίας πατάμε το κουμπί και τοποθετούμε τη κατάσταση στην επιφάνεια εργασίας. Στην κατάσταση αυτή θα δοθεί αυτόματα το όνομα STATE0, το οποίο όμως μπορούμε να ονομάσουμε όπως θέλουμε κάνοντας διπλό κλικ πάνω στην εικόνα

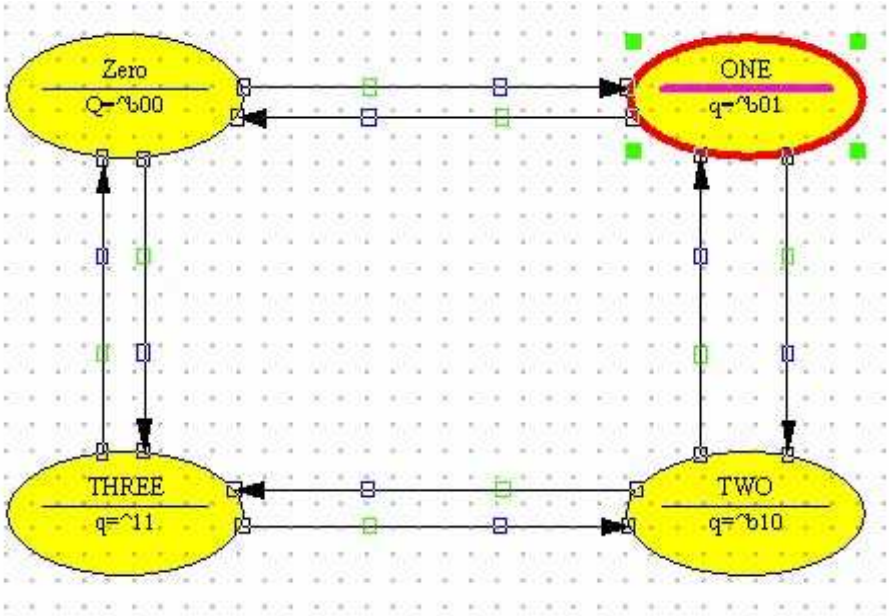


Για τη δή
κατάσταση
και ο
κατάσταση.
καταστάσει

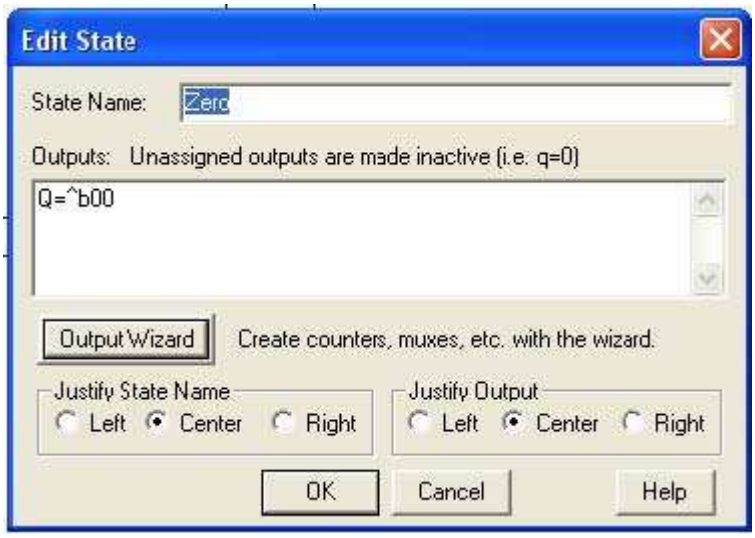
Για τη δήλωση της μετάβασης από τη μία κατάσταση στην άλλη χρησιμοποιούμε το κουμπί και ορίζουμε την αρχική και τη τελική Επειδή έχουμε έναν μετρητή 4ων καταστάσεων το τελικό διάγραμμα είναι ως εξής:



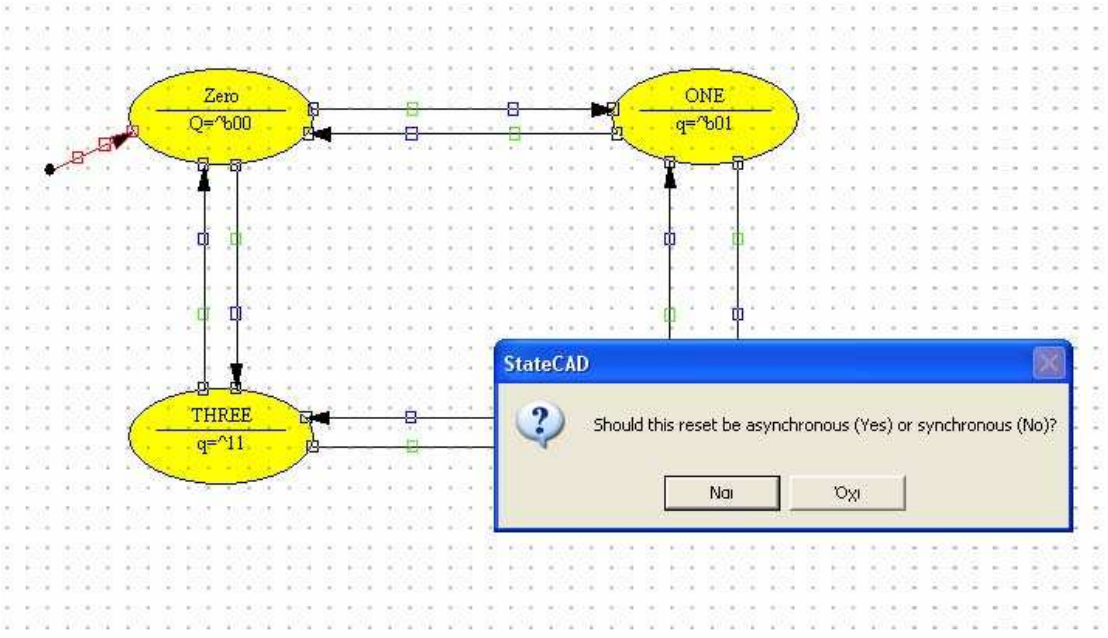
Για τον ορισμό των επιθυμητών εξόδων σε κάθε κατάσταση κάνω διπλό κλικ πάνω στην κατάσταση και μας ανοίγει ένα καινούργιο παράθυρο όπου ορίζουμε τις εξόδους. Τελικά το διάγραμμα θα έχει πάρει τη μορφή.



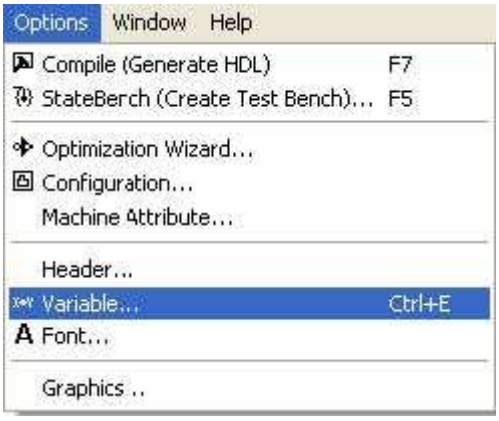
Τώρα θα πρέπει να ορίσουμε τις συνθήκες μετάβασης μεταξύ των καταστάσεων. Κάνοντας διπλό κλικ στις μεταβάσεις ανοίγει το παράθυρο ορισμού παραμέτρων από όπου επιλέγουμε το “Output Wizard”. Σε αυτό το παράθυρο δίνονται αρκετές επιλογές για τη συμπεριφορά των στοιχείων του διαγράμματος. Στο πεδίο αυτό μπορούμε να ορίσουμε τις εξόδους μας με αρκετούς τρόπους, ο πιο απλός όμως είναι η απευθείας ανάθεση τιμών στις εξόδους.



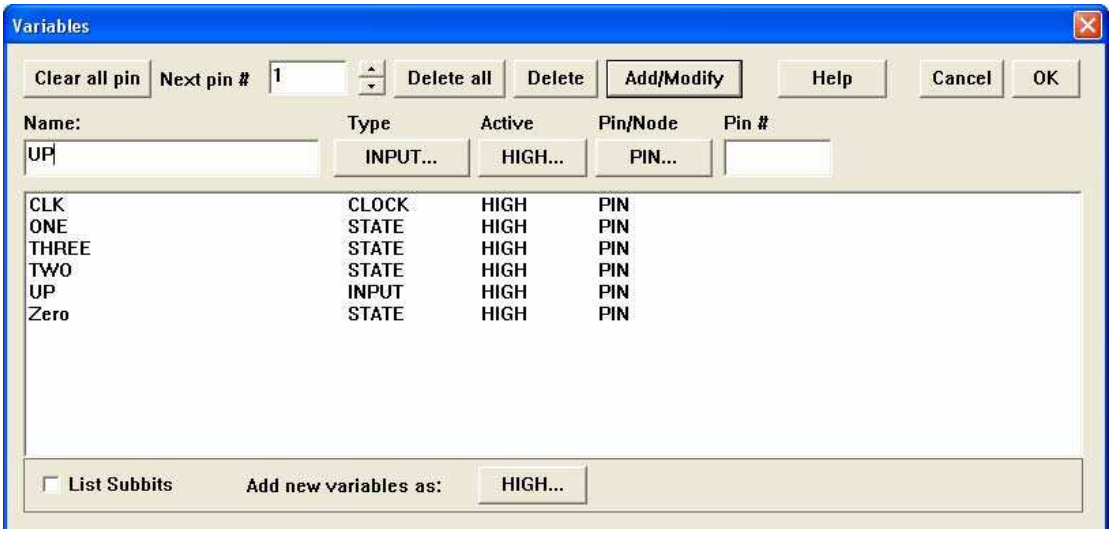
Και τελικά το διάγραμμα θα έχει αυτή τη μορφή. Μπορούμε επίσης να χρησιμοποιήσουμε και μία εντολή Reset για την επανεκκίνηση
του συστήματος. Πατώντας το κουμπί με τον ίδιο τρόπο που τοποθετήσαμε τις μεταβάσεις ορίζουμε ποια θα είναι η κατάσταση Reset και αν θα είναι σύγχρονη η ασύγχρονη.



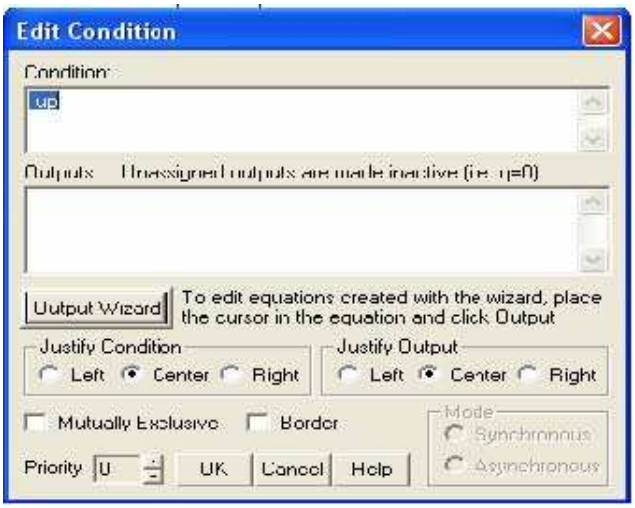
Τώρα θα πρέπει να γίνει η εισαγωγή της μεταβλητής η οποία θα είναι υπεύθυνη για την αύξουσα η φθίνουσα μέτρηση.



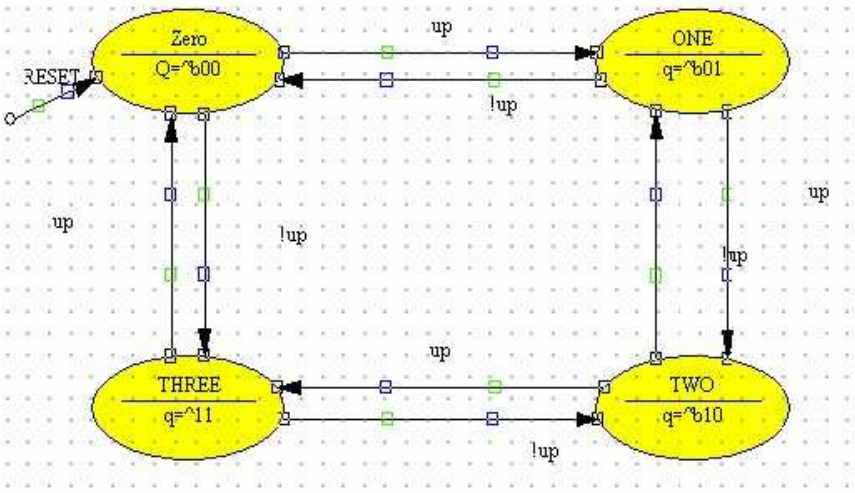
Πηγαίνοντας στην επιλογή “Options⇨ Variable” μπορούμε να ορίσουμε μια σειρά από παραμέτρους και μεταβλητές για να μπορέσουμε να μελετήσουμε ακόμα καλύτερα το διάγραμμα ροής. Εδώ θα πρέπει να τονιστεί ότι ακόμα δεν έχουμε ορίσει τον τρόπο με τον οποίο θα γίνεται η μετάβαση μεταξύ των καταστάσεων. Αυτό θα γίνει μετά τον ορισμό των μεταβλητών. Με αυτή την επιλογή μας ανοίγει ένα καινούργιο παράθυρο. Στο παράθυρο αυτό ορίζουμε τις μεταβλητές εισόδου και εξόδου καθώς και τη συμπεριφορά των ακίδων (είσοδοι/έξοδοι). Γράφουμε το όνομα της μεταβλητής πχ. UP στο αντίστοιχο πεδίο και πατάμε το πλήκτρο Add/Modify, και μετά το πλήκτρο OK. Τώρα το σύστημα είναι σε θέση να επεξεργαστεί και τη μεταβλητή UP η οποία θα ορίζει το είδος της μέτρησης, αύξουσας ή φθίνουσας.



Κάνοντας κλίκ πάνω στα βέλη των μεταβάσεων μας ανοίγει το παράθυρο ορισμού των μεταβολών το οποίο είναι ίδιο με το παράθυρο ορισμού των καταστάσεων. Μόνο που εδώ εμείς θα ορίσουμε μόνο τη συνθήκη μεταβολής της κατάστασης στο πεδίο “Condition”.Έτσι όπως το έχουμε ορίσει αυτό σημαίνει ότι θα γίνεται αύξουσα μέτρηση όταν η μεταβλητή UP θα έχει τιμή ‘’1’’, ενώ στην αντίθετη περίπτωση η μέτρηση θα είναι φθίνουσα, οπότε στη συνθήκη θα πρέπει να γράψουμε !UP όπου το ‘’!’’ δηλώνει την αντιστροφή.

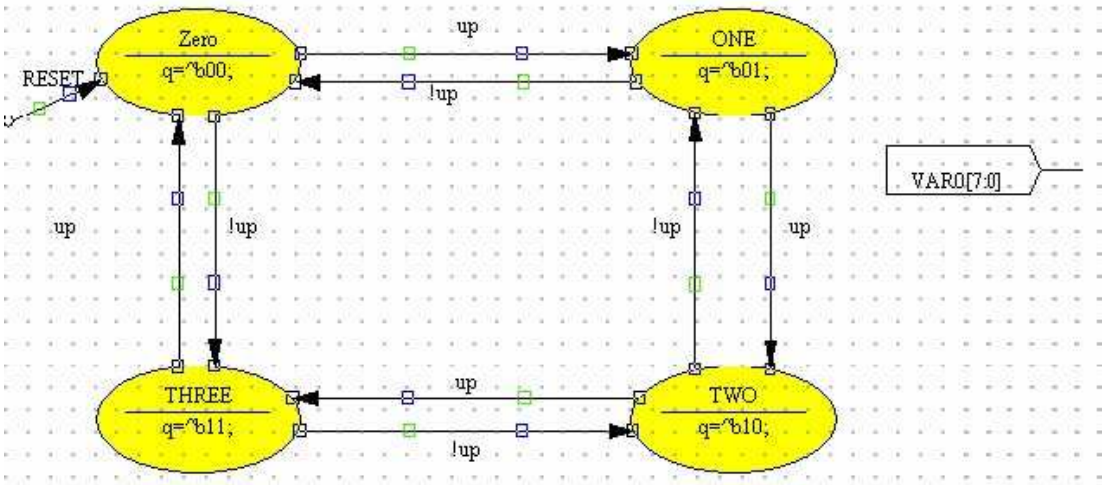


Αφού δηλώσουμε όλες τις συνθήκες μετάβασης το διάγραμμα θα πρέπει να έχει ως εξής:

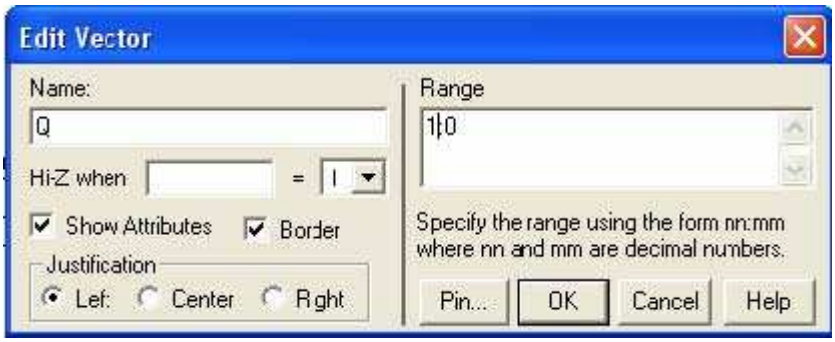


Επίσης θα πρέπει να ορίσουμε και το ότι οι έξοδοι αποτελούν ένα διάνυσμα με συγκεκριμένες ιδιότητες, Στη προκειμένη περίπτωση το Q θα πρέπει να είναι ένα διάνυσμα το οποίο έχει βάθος 2 bit και ότι είναι η έξοδος του συστήματος.

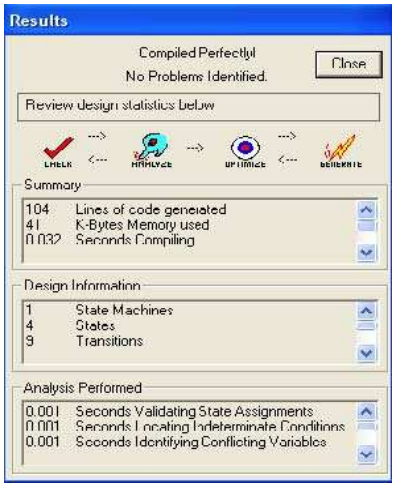
Για να γίνει αυτό κάνουμε κλικ στο πλήκτρο  “Add Vector” και το τοποθετούμε στη επιφάνεια εργασίας.



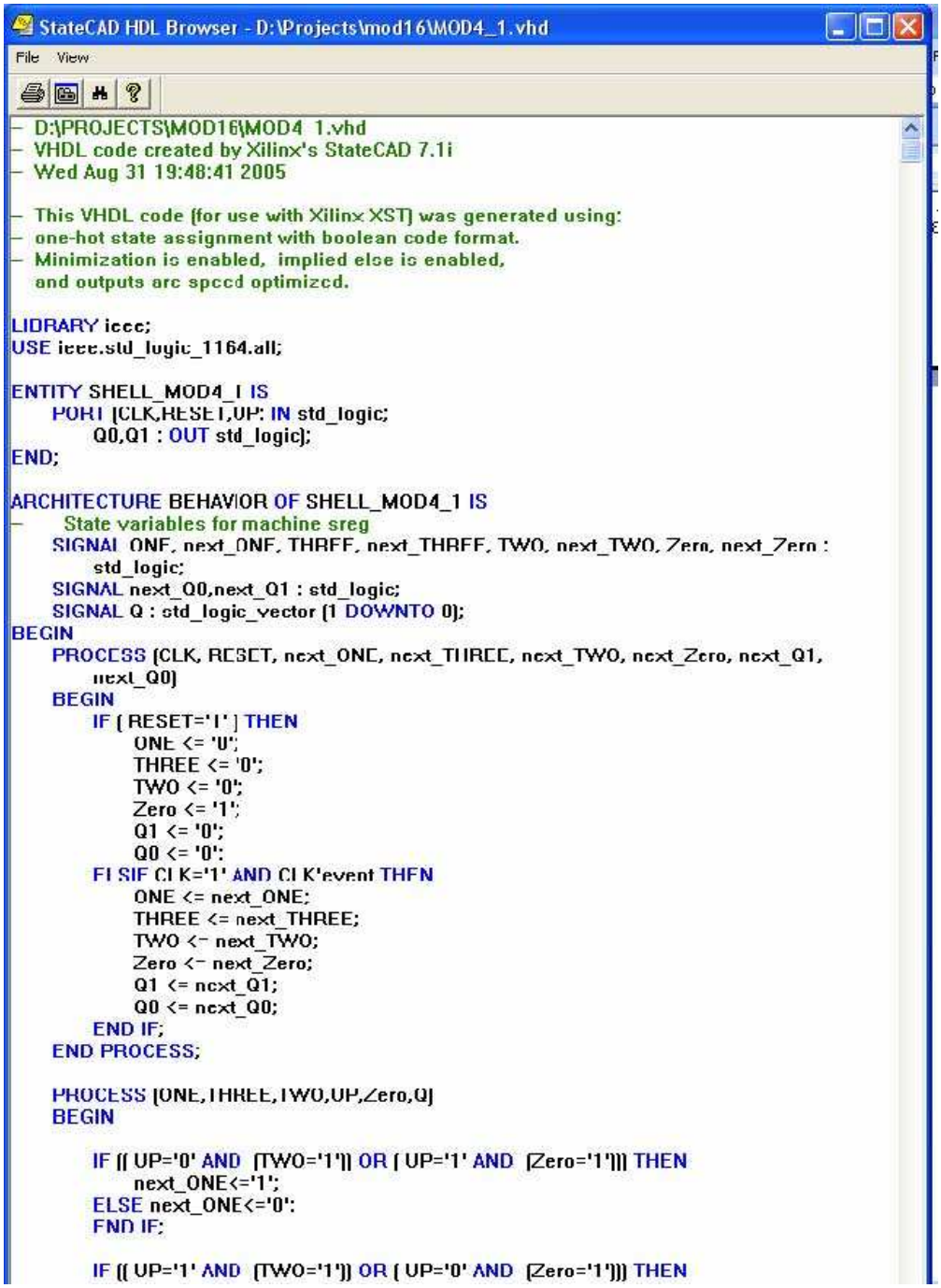
Κάνοντας διπλό κλικ πάνω στο διάνυσμα ανοίγει το παράθυρο ιδιοτήτων του διανύσματος. Στο πεδίο “Name” ορίζουμε το όνομα του διανύσματος και στο πεδίο “Range” το εύρος του . Στη προκειμένη περίπτωση το διάνυσμα αποτελείται από δυο bits.Και πατώντας “OK” καταχωρούμε το διάνυσμα στην εργασία μας



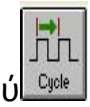
Τώρα θα πρέπει να μεταφράσουμε το διάγραμμα σε γλώσσα VHDL η οποία θα είναι και υπεύθυνη για το εξομοίωση αλλά και για την εξαγωγή του κώδικα προγραμματισμού στη συνέχεια. Με την εντολη "Options⇨ Compile"" γίνεται η μετάφραση, και αν όλα είναι σωστά θα



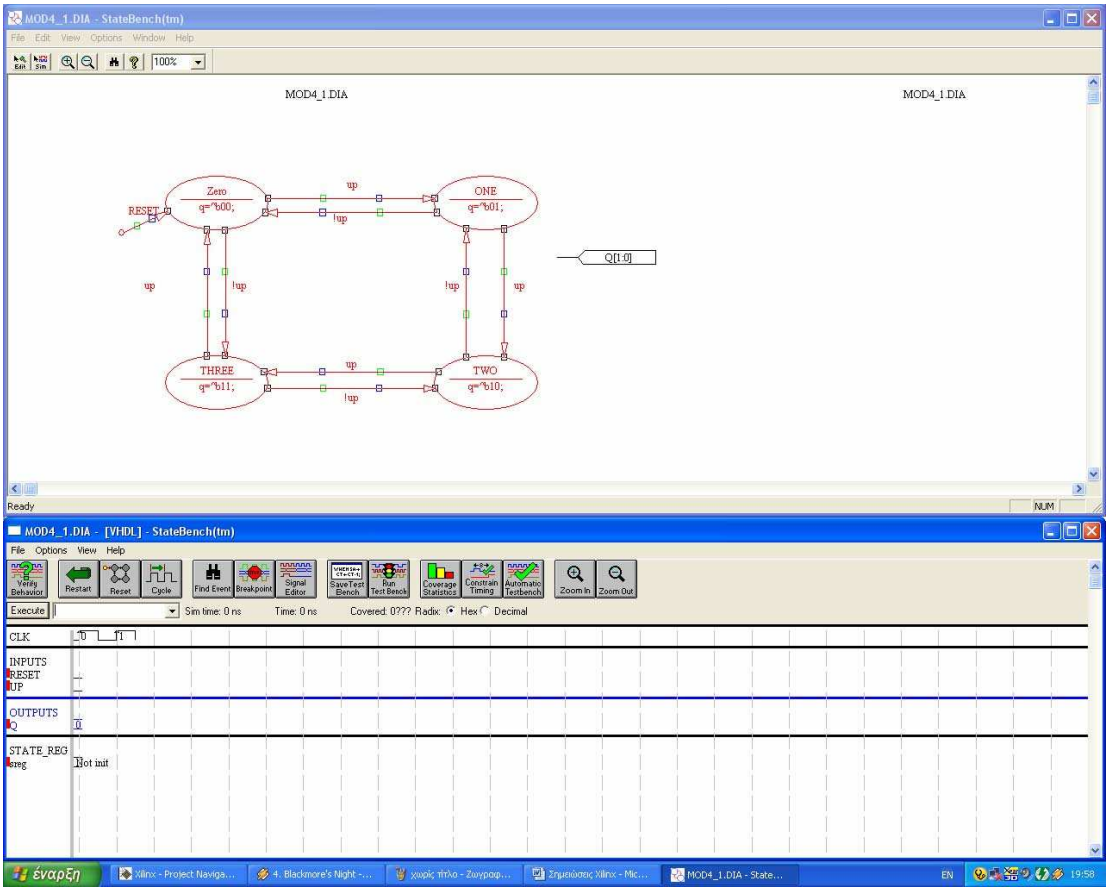
πρέπει να πάρουμε το παραπάνω μήνυμα. Πατώντας το πλήκτρο OK θα μας εμφανίσει και τον αντίστοιχο κώδικα VHDL του διαγράμματος μας.

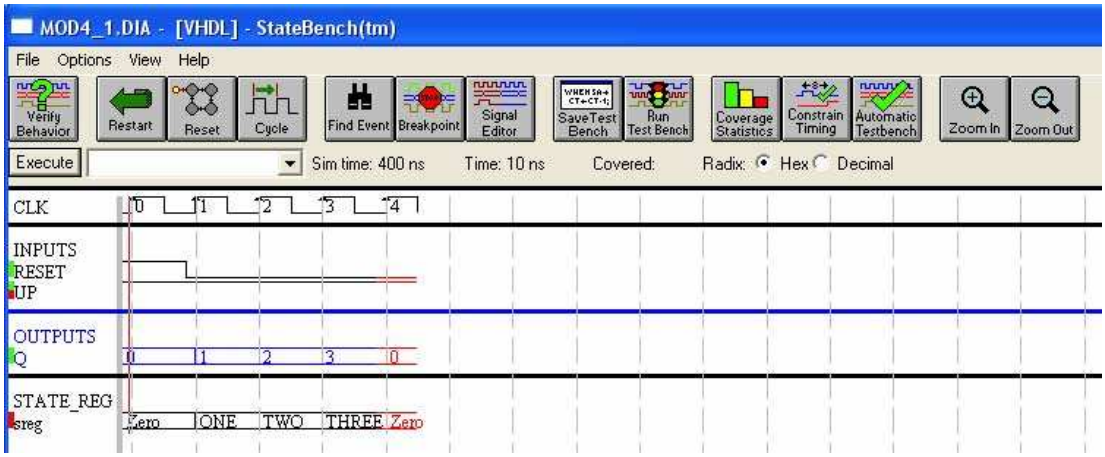


Στη συνέχεια θα πρέπει να βεβαιωθούμε ότι τι διάγραμμα ανταποκρίνεται στις απαιτήσεις μας. Αυτό θα το δούμε μέσω του εξομοιωτή, εδώ θα πρέπει να τονίσουμε ότι ο συγκεκριμένος εξομοιωτής είναι ανεξάρτητος του Modelsim. Για να τρέξουμε τον εξομοιωτή πατάμε το πλήκτρο , το οποίο μας ανοίγει το πρόγραμμα εξομοίωσης

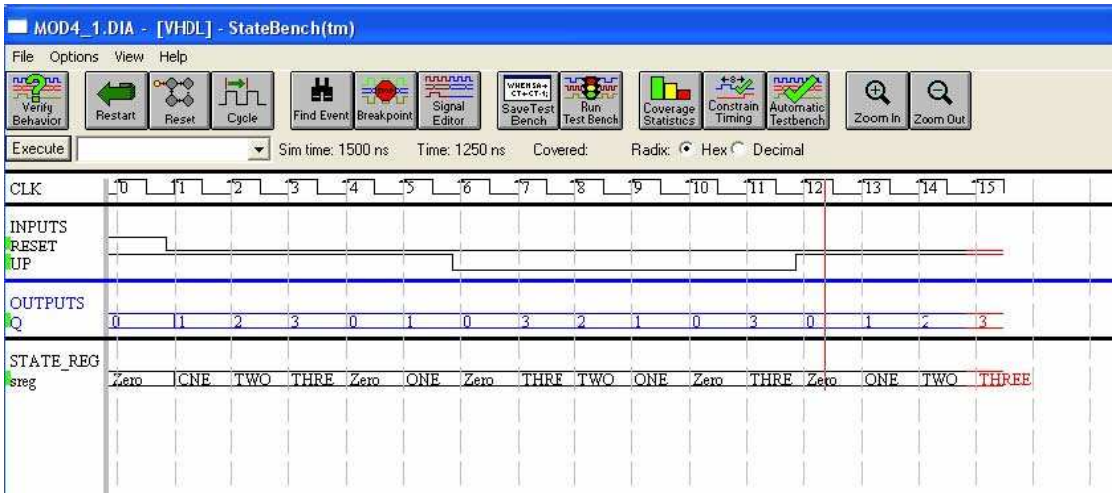
Εδώ τώρα μπορούμε να δούμε τη ροή των δεδομένων βήμα προς βήμα. Πατώνταστο πλήκτρο μπορούμε να δούμε τη συμπεριφορά του διαγράμματος σε κάθε παλμό χρονισμού 

. Όπως επίσης και σε ποια κατάσταση βρισκόμαστε στο διάγραμμα Με το πλήκτρο Reset εφαρμόζουμε παλμό Reset σε όλο το σύστημα ενώ με το πλήκτρο Restart γίνεται επανεκκίνηση σε όλη τη διαδικασία.





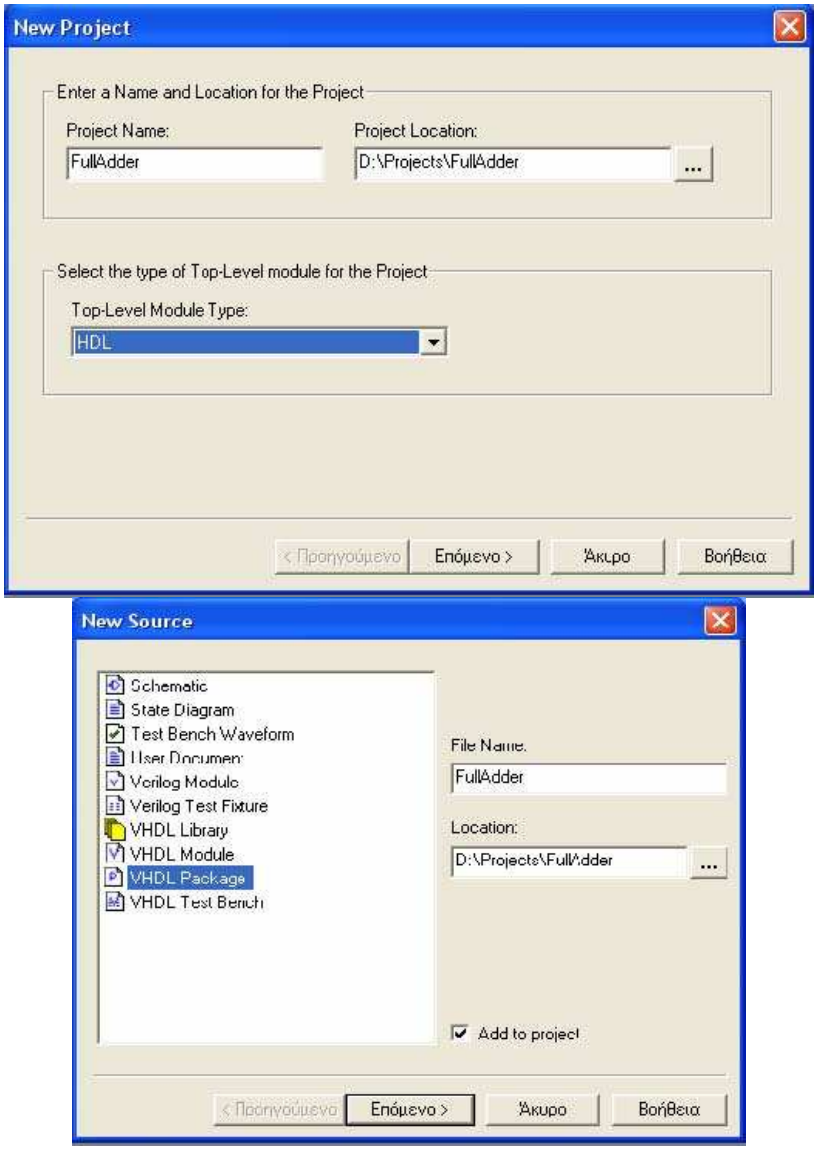
Κάνοντας διπλό κλικ πάνω στο σήμα UP μπορούμε να αλλάξουμε τη λογική του κατάσταση και να εξετάσουμε τη συμπεριφορά του διαγράμματος με αυτή τη μεταβλητή.



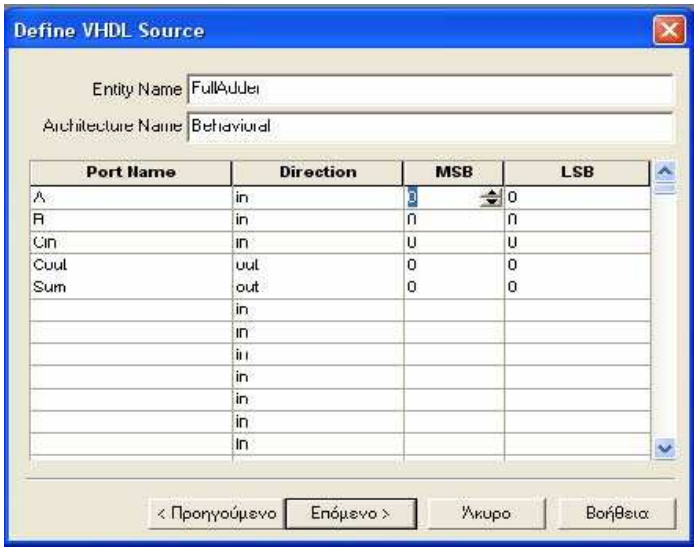
Από τις κυματομορφές των εξόδων συμπεραίνουμε ότι το διάγραμμα λειτουργεί έτσι ακριβώς όπως το θέλουμε.

5.10 Δημιουργία αρχείου VHDL

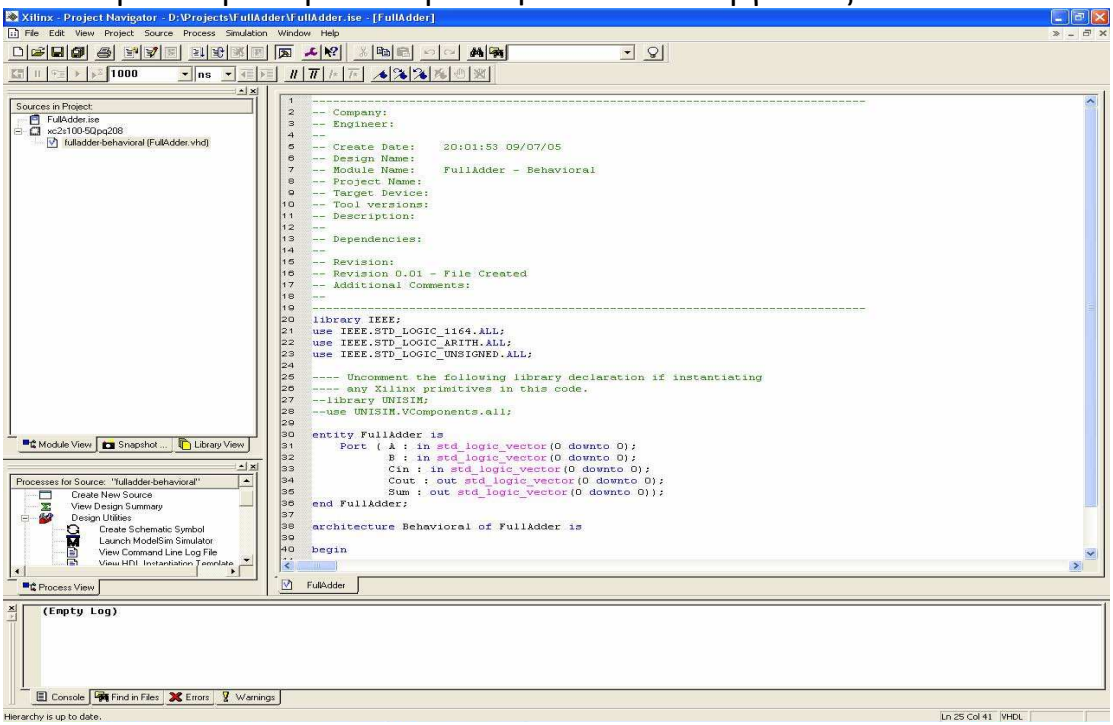
Η δημιουργία αρχείου που θα περιέχει μόνο κώδικα σε γλώσσα VHDL δε διαφέρει με σχεδόν σε τίποτα από τη δημιουργία άλλου τύπου αρχείων στο παρόν πρόγραμμα, αλλά υπάρχουν αρκετοί διαφορετικοί τρόποι προσέγγισης και λύσης κάποιου συγκεκριμένου προβλήματος. Αυτό έχει να κάνει περισσότερο με τις διαθέσιμες εντολές που έχει η γλώσσα. Αυτό όμως θα αναπτυχθεί περισσότερο παρακάτω. Δημιουργούμε πάλι ένα Project μέσα στο όποιο θα περιέχονται όλα τα απαραίτητα αρχεία, μόνο που τώρα θα πρέπει να δηλωθεί ότι η μέθοδος ανάπτυξης θα είναι η γλώσσα VHDL.



Στη συνέχεια θα πρέπει να δηλώσουμε τα στοιχεία εισόδου και εξόδου τα οποία θα χρησιμοποιήσουμε, μια διαδικασία η οποία μπορεί να παραληφθεί τελείως σε αυτό το στάδιο με την προϋπόθεση όμως ότι τα στοιχεία αυτά θα δηλωθούν μέσα στον ίδιο τον κώδικα. Αν δηλωθούν τώρα το πρόγραμμα θα προσθέσει αυτόματα τον κώδικα περιγραφής των στοιχείων εισόδου/εξόδου.



Το στοιχείο “Port Name” ορίζει το όνομα της εισόδου/εξόδου, στο πεδίο “Direction” ορίζεται αν το προηγούμενο στοιχείο θα συμπεριφέρεται σαν είσοδος ή σαν έξοδος. Τα πεδία “MSB” και “LSB” δηλώνονται το μεγαλύτερο και το μικρότερο bit δηλαδή το εύρος των ψηφίων που έχει το στοιχείο εισόδου/εξόδου . Στη προκείμενη περίπτωση επειδή οι είσοδοι και έξοδοι είναι μονοψήφιοι το MSB και το LSB έχουν την ίδια τιμή. Σε περίπτωση που θα θέλαμε διψήφιους αριθμούς τότε το MSB θα είχε την τιμή “1” ενώ το LSB θα είχε την τιμή “0”, δηλαδή η είσοδος A π.χ. θα περιείχε τα ψηφία A1 και A0 .Θα εμφανιστεί το συγκεκριμένο περιβάλλον εργασίας . Τα στοιχεία που περιέχονται στο αρχείο θα αναλυθούν παρακάτω. Για την ώρα θα επικεντρωθούμε περισσότερο στη διαδικασία εργασίας

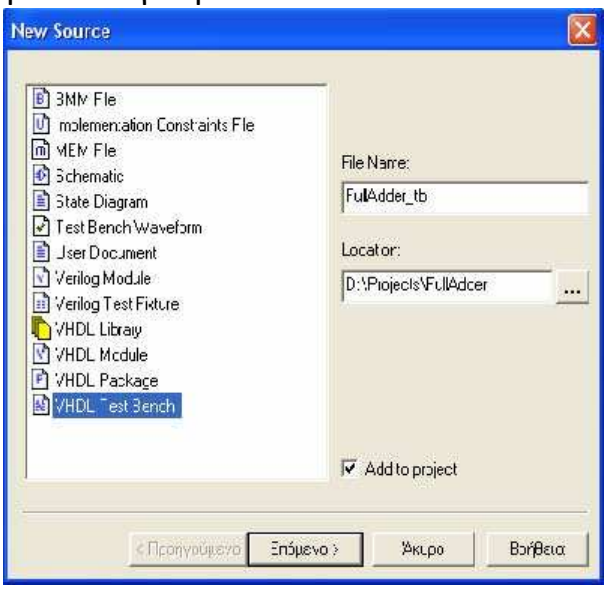


Β AΒ (Οι εξισώσεις που έχουμε είναι $Sum = A \oplus \oplus C$ και $Cout = \cdot + \oplus AB$)

Αυτές τις εξισώσεις θα πρέπει να τις δηλώσουμε και μέσα στο πρόγραμμα.

```
30  entity FullAdder is
31      Port ( A : in std_logic_vector(0 downto 0) ;
32             B : in std_logic_vector(0 downto 0) ;
33             Cin : in std_logic_vector(0 downto 0) ;
34             Cout : out std_logic_vector(0 downto 0) ;
35             Sum : out std_logic_vector(0 downto 0));
36  end FullAdder;
37
38  architecture Behavioral of FullAdder is
39
40  begin
41        Sum <= (A XOR B) XOR Cin  ;
42        Cout <= (A AND B) OR ( NOT (A XOR B))  ;
43
44  end Behavioral;
45
```

Τώρα θα πρέπει να βεβαιωθούμε ότι αυτά που γράψαμε συμπεριφέροντε όπως θέλουμε, για το λόγο αυτό θα πρέπει πάλι να περάσουμε, το κώδικα αυτή τη φορά, από τον εξομοιωτή. Η διαδικασία είναι παραπλήσια με τη διαδικασία που παρουσιάστηκε παραπάνω, μόνο που αυτή τη φορά θα πρέπει να δημιουργήσουμε ένα αρχείο σε κώδικα VHDL το οποίο θα παρέχει τις αναγκαίες διεγέρσεις στο σύστημα. Δημιουργούμε με τον ίδιο τρόπο ένα αρχείο “Test Bench” αλλά αυτή τη φορά επιλέγουμε το “VHDL Test Bench” .



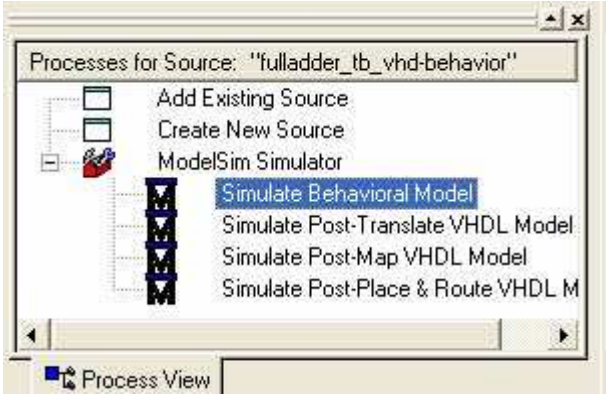
Η συγγραφή του κώδικα σε αυτού του είδους τα αρχεία είναι παρόμοια με τα κανονικά αρχεία που γράφουμε σε VHDL μόνο που στα κανονικά δίνουμε περισσότερο έμφαση στον καθορισμό τις λειτουργίας των συστημάτων και υποσυστημάτων ενώ σε ένα " Test Bench" αρχείο δίνουμε περισσότερο έμφαση στους ορισμούς λειτουργίας των εισόδων και εξόδων των συστημάτων. Στη προκειμένη περίπτωση πρέπει να δηλώσουμε το πώς θέλουμε να συμπεριφέροντε ι είσοδοι, για να μπορέσουμε να μελετήσουμε τη συμπεριφορά των εξόδων , ενώ στο βασικό μας αρχείο VHDL μας ενδιαφέρει περισσότερο να ορίσουμε τη λειτουργία των εξόδων συναρτήσει των εισόδων.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;


ENTITY FullAdder_tb_vhd IS
END FullAdder_tb_vhd;


ARCHITECTURE behavior OF FullAdder_tb_vhd IS


        -- Component Declaration for the Unit Under Test (UUT)
         COMPONENT fulladder
         PORT(

                Cin : IN std_logic;
        a : IN std_logic;
        b : IN std_logic; Sum : OUT std_logic; Cout : OUT std_logic   );

         END COMPONENT;

        --Inputs
        SIGNAL Cin : std_logic := '0';
        SIGNAL a : std_logic := '0';
        SIGNAL b : std_logic := '0';


         --Outputs
        SIGNAL Sum :  std_logic;
        SIGNAL Cout :  std_logic;


BEGIN

            -- Instantiate the Unit Under Test (UUT) uut:
            fulladder PORT MAP(   Cin => Cin,
                  a => a,
```

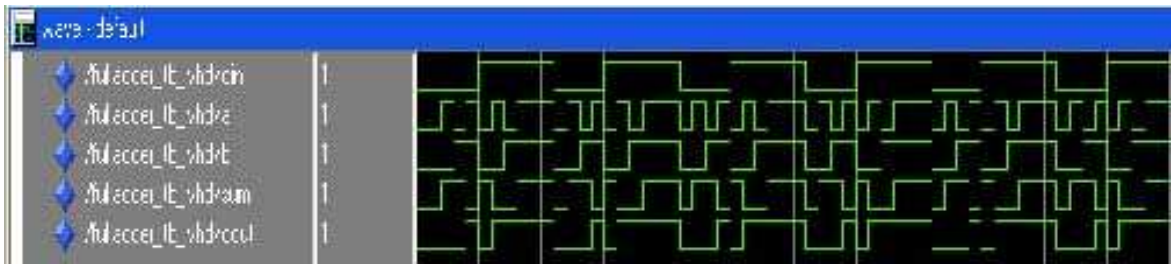
```
            b => b,
            Sum => Sum,
            Cout => Cout
    );
    tb : PROCESS
     BEGIN
            -- Wait 100 ns for global reset to finish
wait for 100 ns; -- stage 0      '000'
            A <= '0' ;
            b <= '0' ;
            cin  <= '0' ;
            wait for 100 ns; -- stage 1      '001'
            A <= '1' ;
            b <= '0' ;
            cin  <= '0' ;
            wait  for 100 ns; -- stage 2      '010'
            A <= '0' ;
            b <= '1' ;
            cin  <= '0' ;
            wait  for 100 ns; -- stage 3      '011'
            A <= '1' ;
            b <= '1' ;
            cin  <= '0' ;

            wait  for 100 ns; -- stage 4      '100'
            A <= '0' ;
            b <= '0' ;
            cin  <= '1' ;
            wait for 100 ns; -- stage 5      '101'
            A <= '1' ;
            b <= '0' ;
            cin  <= '1' ;
            wait  for 100 ns; -- stage 6      '110'
            A <= '0' ;
            b <= '1' ;
            cin  <= '1' ;
            wait for  100 ns ; -- stage 7      '111'
            A <= '1' ;
            b <= '1' ;
            cin  <= '1' ;
            wait  for 100 ns;
            wait  for 100 ns;
            --wait ; --will wait forever
     END PROCESS; END;
```

Αφού γίνει ο ορισμός της λειτουργίας των εισόδων είμαστε σε θέση να κάνουμε εξομοίωση του συστήματος. Εδώ θα πρέπει να τονίσουμε ότι θα πρέπει να δώσουμε ιδιαίτερη προσοχή και επιμέλεια στο “Test Bench” αρχείο γιατί αυτό είναι και κατ’ουσίαν υπεύθυνο για τη εξαγωγή των συμπερασμάτων αναφορικά με τη σωστή λειτουργία του

συστήματος. Δηλαδή υπάρχει ο κίνδυνος ο σχεδιασμός του συστήματος να είναι σωστός, αλλά επειδή το " Test Bench " αρχείο είναι προχειρογραμμένο να βγάζουμε λανθασμένα συμπεράσματα.
Το επόμενο βήμα είναι να τρέξουμε το πρόγραμμα εξομοίωσης το οποίο θα μας δώσει και τις τιμές των εξόδων συναρτήσει των εισόδων.



Στο επόμενο βήμα ανοίγει το πρόγραμμα "Modelsim" με όλα τα απαραίτητα εργαλεία για τη σωστή εξομοίωση. Και τα αποτελέσματα της διεργασίας αυτής είναι ορατά στο "wave" αρχείο το οποίο μας ανοίγει .



Με μια λίγο προσεκτική μελέτη καταλήγουμε στο συμπέρασμα ότι το κύκλωμα το οποίο περιγράψαμε σε γλώσσα VHDL συμπεριφέρεται ακριβώς όπως θα έπρεπε. Τονίζουμε ότι περισσότερες διευκρινήσεις σχετικά μα τον κώδικα περιγραφής του παραπάνω συστήματος θα δοθούν αναλυτικά σε παρακάτω άσκηση .

ΚΕΦΑΛΑΙΟ 6

ΑΝΑΠΤΥΞΗ ΠΛΑΚΕΤΩΝ

Ο προγραμματισμός των CPLD's και FPGA's της Xilinx γίνεται με την βοήθεια του προγράμματος της Xilinx και ονομάζεται Xilinx ISE 7.1i. Το προγραμμα αυτό, χρησιμοποιήται αποκλειστικά για το προγραμματισμό των ολοκληρωμένων κυκλωμάτων της Xilinx. Παρουσιάζει μια πληθώρα εργαλείων και λειτουργιών που οδηγούν το χρήστη βήμα-βήμα από την σχεδίαση του κυκλώματος στο προγραμματισμό του PLD. Ένα από τα πολλά χρήσιμα εργαλεία του είναο το iMPACT. Το iMPACT επιτρέπει στον χρήστη να κάνει κάπειες ρυθμίσεις και κάποιες επιλογές που έχουν να κάνουν με τον προγραμματισμό του PLD. Παράγει το αρχείο πτογραμματισμού JTAG. Ο προγραμματισμός επιτθγχάνεται μέσω παράλληλης θύρας ή της θύρας USB, του Ηλεκτρονικού Υπολογιστή, στην οποία συνδέεται η συσκευή προγραματισμού.

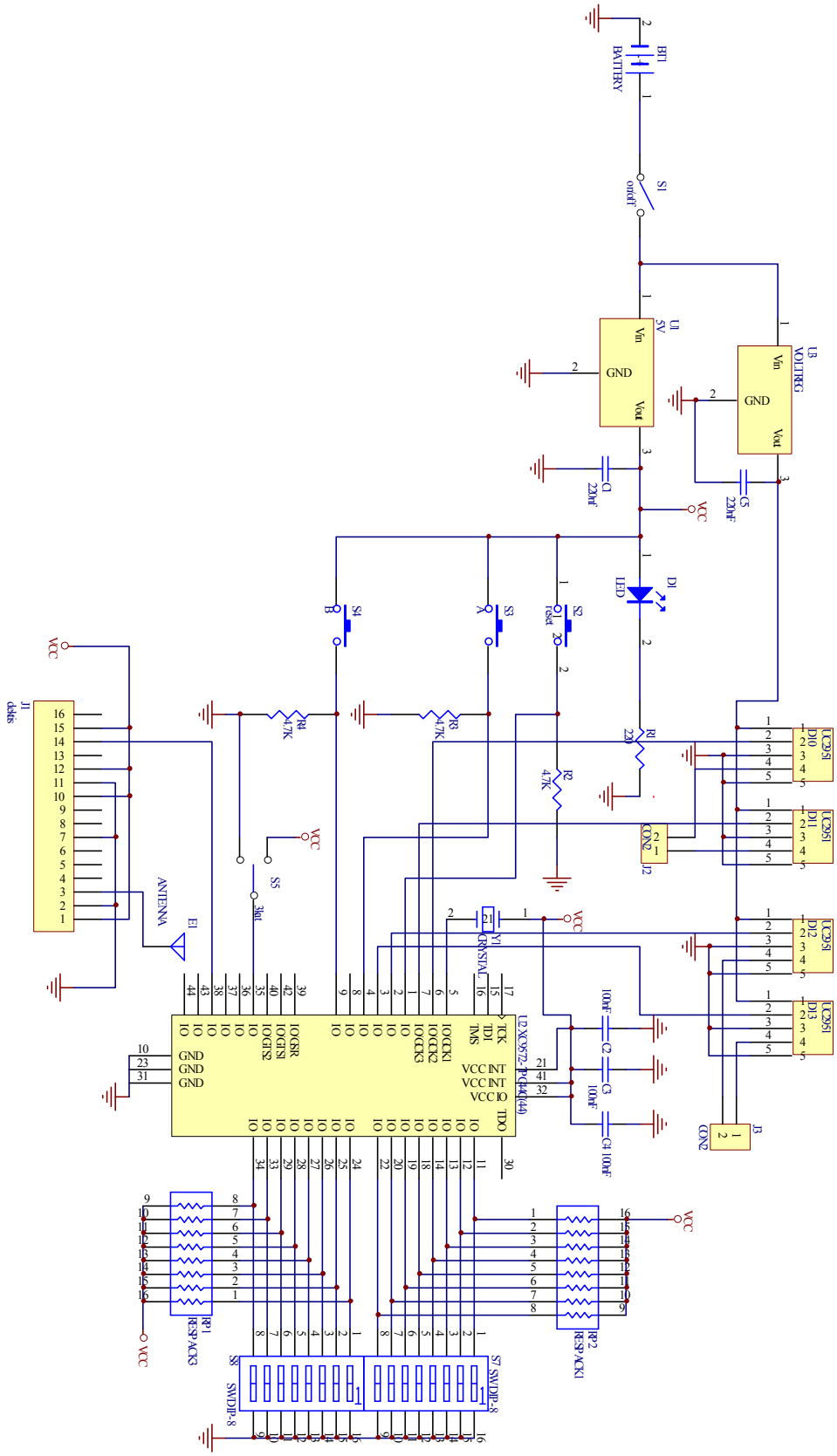
Η εταιρεία Xilinx προσφέρει μερικές τέτοιες συσκευές προγραμματισμού. Οι προγραμματιστές της Xilinx όπως είναι το MultiLINX Cable, το Parallel CableIV, το Parallel Cable III και το MultiPRO, απευθύνονται αποκλειστικά στα εξαρτήματα της ίδιας της εταιρείας. Φυσικά κυκλοφορούν και προγραμματιστές από άλλες εταιρείες που μπορούν να συνεργαστού απόλυτα με το ISE 5.1i της Xilinx. Κάποιες εταιρείες, που συνεργάζονται με την Xilinx, κατασκευάζουν διάφορες αναπτυξιακές πλακέτες. Στις πλακέτες αυτές υπάρχουν διάφορα υλικά

βοηθητικά υλικά όπως LCD οθόνες , θύρες επικοινωνίας με Η/Υ κ.τ.λ., ανάλογα με την χρήση που προορίζονται.

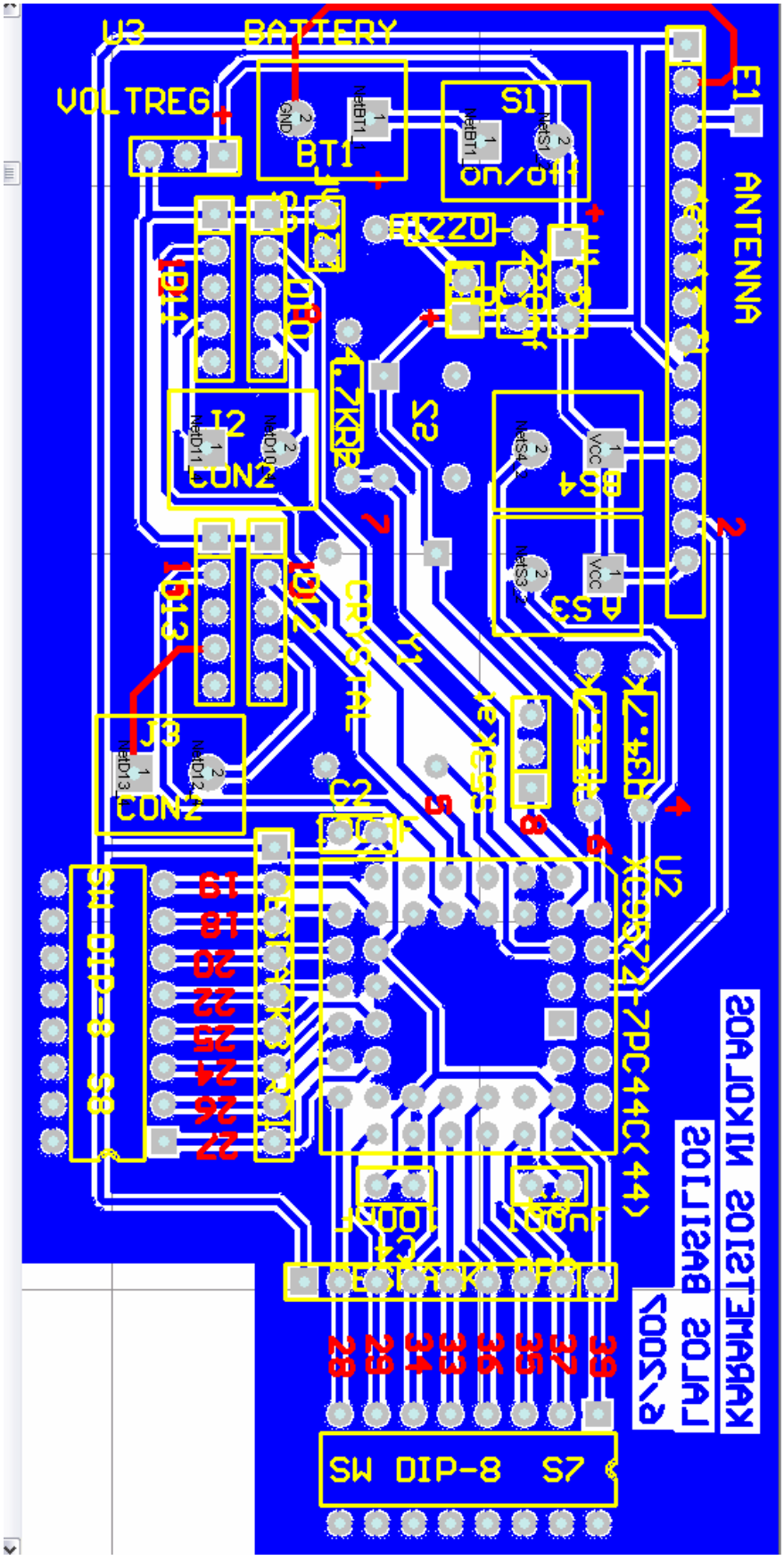
6.1 Η ΑΝΑΠΤΥΞΗ ΠΛΑΞΕΤΩΝ ΠΟΜΠΟΥ ΚΑΙ ΔΕΚΤΗ

Με βάση τα κυκλώματα του πομπού και του δέκτη που αναφερθίκαμε σε προηγούμενο καφάλαιο, σχεδιάστηκαν και κατασκευάστηκαν οι δύο αναπτυξιακές πλακέτες με το ολοκληρωμένο κύκλωμα της Xilinx XC9572-PC44. Οι πλακέτες αυτές μπορούν να συνδεθούν στον Ηλεκτρονικό Υπολογιστή μέσω της παράλληλης θύρας και να προγραμματίσουν το XC9572 που βρίσκεται τοποθετημένο σ' αυτές, με την βοήθεια του προγράμματος Xilinx ISE 7.1i και του εργαλείου iMPACT.

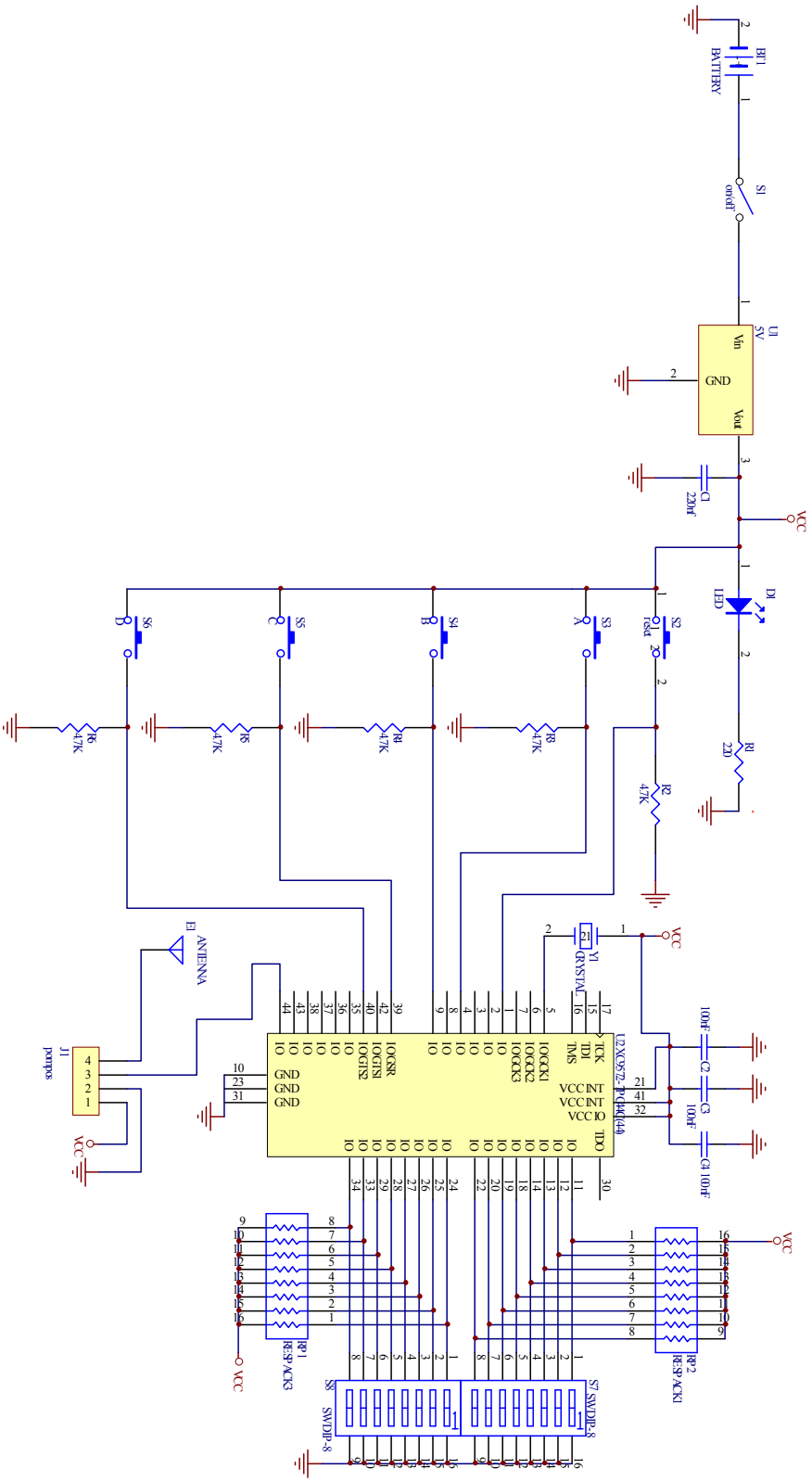
Το XC9572 είναι ένα CPLD της οικογένειας XC9500, περιέχει 72 μακροκυψέλες και 1600 πύλες. Κυκλοφορεί στις συσκευασίες PLCC44, PLCC84, TQFP100 και PQFP100. Για τις αναπτυξιακές πλακέτες επιλέχθηκε η συσκευασία PLCC44 γιατί η συσκευασία αυτή, σε αντίθεση με τις άλλες, χρησιμοποιεί μια ειδική υποδοχή (PLCC84 socket), η οποία ευκολύνη την τοποθέτηση του ολοκληρωμένου στην πλακέτα καθώς και την απομάκρυνση του από αυτή. Ο σχεδιασμός των αναπτυξιακών πλακετών έγινε με τη πρόγραμμα Protel 99 Se της Altium. Το σχηματικό διάγραμμα και το σχέδιο με το οποίο κατασκευάζεται το τυπωμένο κύκλωμα (PCB) του δέκτη δίνεται στο σχήμα 6.1 και 6.2 αντιστοιχα. Ενώ το σχηματικό διάγραμμα και το σχέδιο με το οποίο κατασκευάζεται το τυπωμένο κύκλωμα (PCB) του πομπού δίνεται στο σχήμα 6.3 και 6.4 αντίστοιχα.



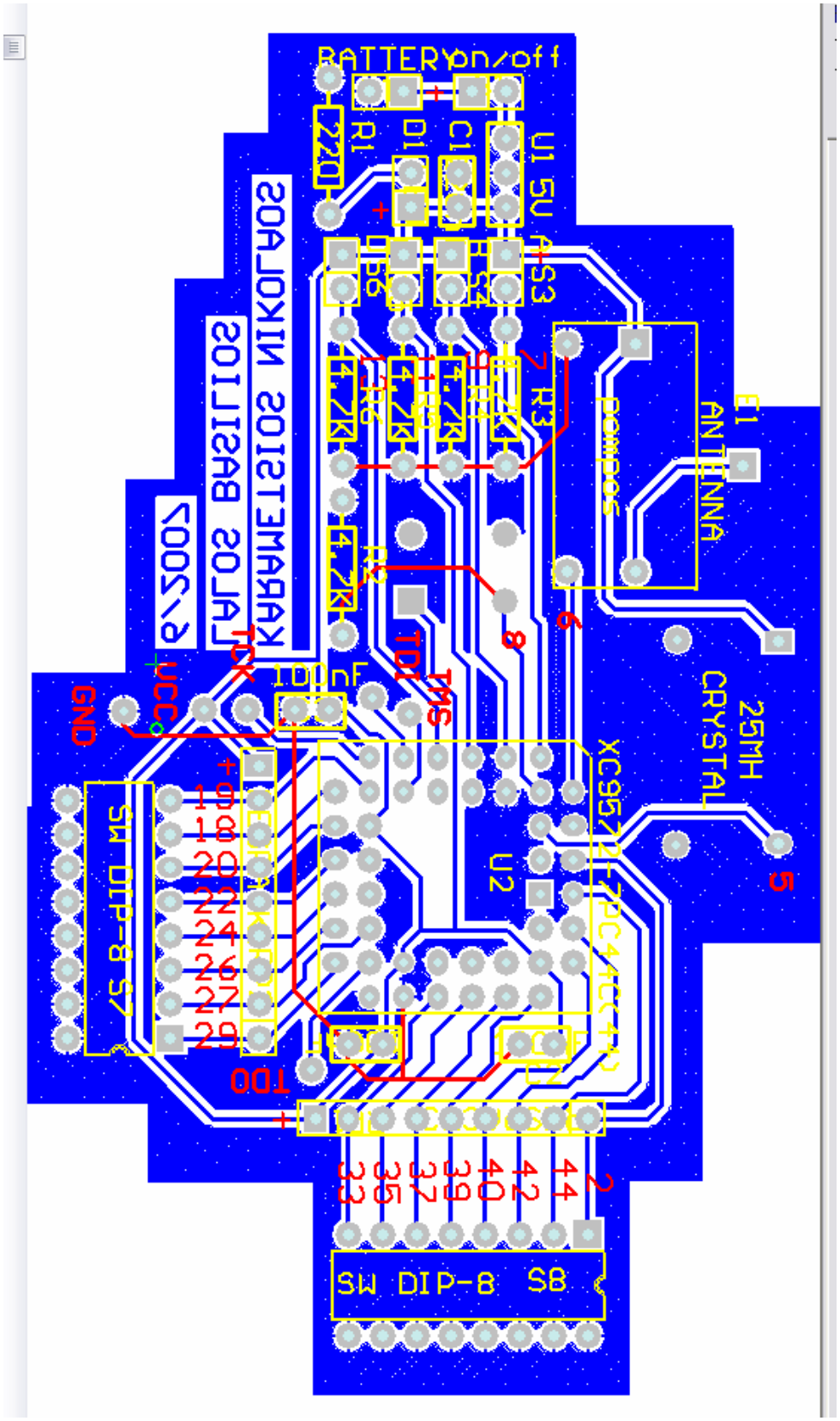
Σχ.6.1 ΣΧΗΜΑΤΙΚΟ ΔΙΑΓΡΑΜΜΑ ΑΝΑΠΤΥΞΙΑΚΗΣ ΠΛΑΚΕΤΑΣ ΔΕΚΤΗ



Σχ.6.2 ΤΥΠΩΜΕΝΟ ΚΥΚΛΩΜΑ ΑΝΑΠΤΥΞΙΑΚΗΣ ΠΛΑΚΕΤΑΣ ΔΕΚΤΗ



Σχ.6.3 ΣΧΗΜΑΤΙΚΟ ΔΙΑΓΡΑΜΜΑ ΑΝΑΠΤΥΞΙΑΚΗΣ ΠΛΑΚΕΤΑΣ ΠΟΜΠΥ



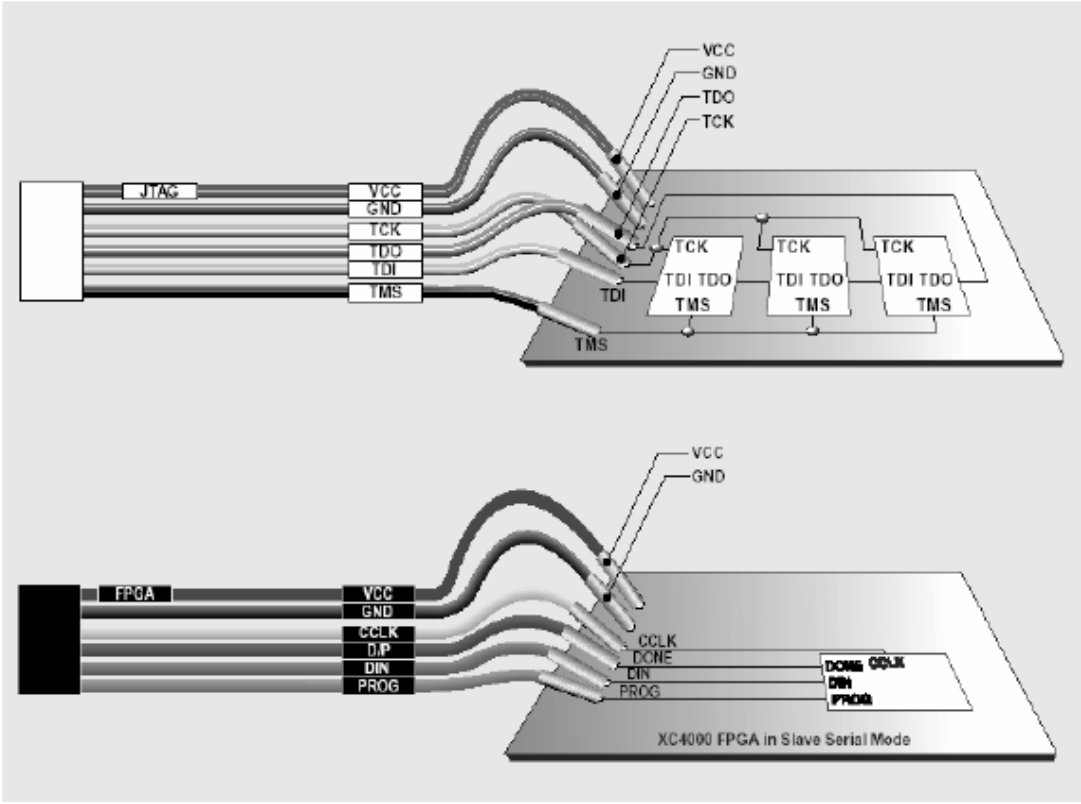
Σχ.6.4 ΤΥΠΩΜΕΝΟ ΚΥΚΛΩΜΑ ΑΝΑΠΤΥΞΙΑΚΗΣ ΠΛΑΚΕΤΑΣ ΠΟΜΠΟΥ

Οι αναπτυξιακές πλακέτες αποτελούνται από τρεις βασικές μονάδες, την μονάδα σταθεροποίησης της τάσης λειτουργίας, την μονάδα προγραμματιστή και την μονάδα εφαρμογών.

6.2 ΜΟΝΑΔΑ ΠΡΟΓΡΑΜΑΤΙΣΤΗ

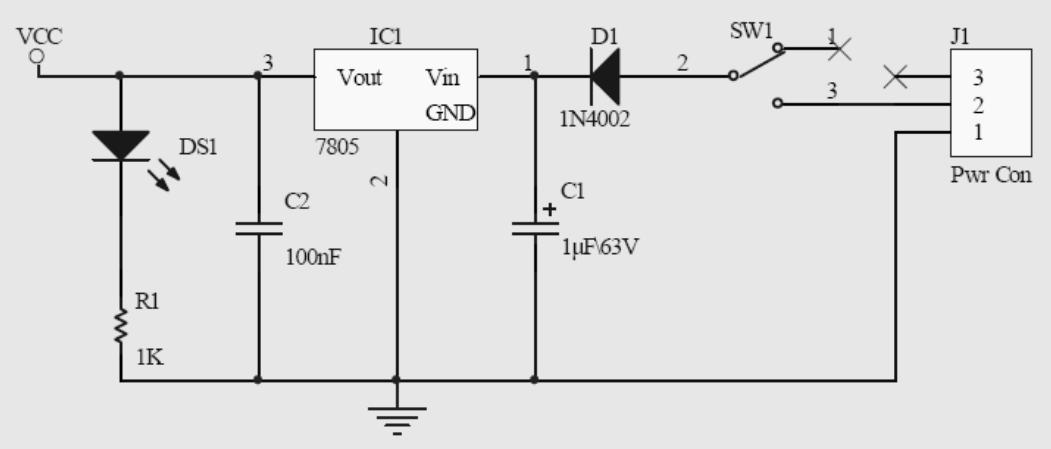
Ο προγραμματιστής, όπως έχει αναφερθεί, βασίζεται στο κύκλωμα του προγραμματιστή Paralle Cable IV της Xilinx, που δίνει η ίδια εταιρεία.

Σε τέσσερα διαφορετικά pin καταλήγουν οι τέσσερις γραμμές προγραμματισμολυ TCK\CCLK, TDI\DIN και TMS\PROG, αλλά και οι δύο ακροδέκτες τροφοδισίας του ολοκλυρωμένου κυκλώματος, Vcc και GND. Μ' αυτούς τους ακροδέκτες η αναπτυξιακή πλακέτα μπορεί να προγραμμαρίσει οποιοδίποτε CPLD και FPGA της Xilinx συνδεθεί στους ακροδέκτες αυτούς. Ο τρόπος που συνδέονται οι ακροδέκτες αυτοί με τα CPLD's και τα FPGA's για τον προγραμματισμό τους, δαίνεται στο σχήμα 6.5.



6.3 ΜΟΝΑΔΑ ΣΤΑΘΕΡΟΠΟΙΗΤΗ ΤΑΣΗΣ

Ο σταθεροποιητής τάσης εξασφαλίζει τη σταθερότητα της τάσης λειτουργείας και προγραμματισμού, VCC, στα 5V DC. Η τάση αυτή αποτελεί την τάση λειτουργίας της πλακέτας αλλά και του ολοκληρωμένου κυκλώματος. Ο σταθεροποιητής αυτός υλοποιήται με το ολοκληρωμένο κύκλωμα LM7805 (3-terminal positive regulator). Η τάση της πλακέτας εξαρτάται από την είσοδο του LM7805 και κυμαίνεται μεταξύ 8 και 20V.



ΠΑΡΑΡΤΗΜΑ Α

ΣΧΟΛΙΑ ΚΑΙ ΠΑΡΑΤΗΡΗΣΕΙΣ

Παρατηρούμε ότι η χρήση των CPLDS και FPGAs αποτελεί την ιδανική λύση για ανάπτυξη μεγάλων και πολύπλοκων λογικών κυκλωμάτων. Ο μικρός όγκος αυτών των ολοκληρωμένων κυκλωμάτων, σε σχέση με το ψηφιακό κύκλωμα που εσωκλείουν, αλλά και η ευκολία της ολοκλήρωσης μιας εφαρμογής, λύνουν πολλά από τα προβλήματα της κλασσικής ανάπτυξης ψηφιακών συστημάτων.

Το βασικό πρόβλημα που αντιμετωπίσαμε ήταν το υψηλό κόστος των αναπτυξιακών εργαλείων που απαιτούνται για σωστό και πλήρη προγραμματισμό αυτών των ολοκληρωμένων. Η τεχνολογία αυτή απευθύνεται κυρίως σε επαγγελματίες που διαθέτουν μεγάλα καφάλαια. Η εταιρεία Xilinx προσφέρει το λογισμικό ISE 7.1i, το οποίο είναι απαραίτητο για την εκπλήρωση οποιασδήποτε εφαρμογής σε ένα από τα ολοκληρωμένα της, αντί υψηλού αντίτιμου. Τον ίδιο κανόνα ακολουθούν όλες οι εταιρίες που ασχολούνται με αυτόν το τομέα.

Αντιμετωπίσαμε δυσκολία με την προσομοίωση. Στ πακέτο προγράμματος δεν συμπεριλαμβάνεται κάποιο προγράμμα προσομοίωσης. Η εταιρία παραπέμπει στην εταιρία Model Technology με την οποία συνεργάζεται. Η εταιρία παρέχει δωρεάν μια υποτυπώδη χρήση του προϊόντος (ModelSim XE), με αρκετους περιορισμους, ενώ η πλήρης χρήση των δυνατοτήτων του απαιτεί καταβολή συμαντικού χρηματικού ποσου.

Μια προσωπική παρατήρηση των περιορισμένων δυνατοτήτων του προϊόντος παρουσιάστηκε κατά την προσομοίωση VHDL κώδκα. Κάποιες εντολές δεν τις δεχόταν, αναγκάζοντας μας να χρησιμοποιήσουμε μεγαλύτερους κώδικές.

Η γλώσσα VHDL μας φάνηκε αρκετά εύκολη για την κατασκευή των επιμέρους στοιχείων, αλλά σε μερικές περιπτώσεις η χρήση σχηματικού ήταν αναγκάια. Η VHDL πολλές φορές μπορεί να αντικαταστήσει μεγάλα κυκλώματα με λίγες εντολές ενώ κάποιες αλλές, λόγω της πολυπλοκότητας της σύνταξης, κάναμε χρήση του σχηματικού. Αυτό πολύ πιθανώς να οφειλέται στην απειρία μας. Για παράδειγμα το σχηματικό ανωτάτου συνδέθηκε αποκλειστικά μα σχηματικη μέθοδο.

Υλοποιήσαμε όλες τις επιμέρους βαθμίδες τόσο με σχηματικό όσο και με κώδικα VHDL. Κατά την εφαρμογή των δύο στο ολοκληρωμένο παρατηρήσαμε ότι απαιτούν περίπου τον ίδιο αριθμό μακροκυψέλων (macrocells). Αυτό είναι λογικό διότι και τα δύο, τελικά μεταφράζονται σε κώδικα.

A BRIEFING IN ENGLISH

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. ΔΡΑΚΑΚΗ ΜΑΡΙΑ, ΔΙΔΑΚΤΙΚΕΣ ΣΗΜΕΙΩΣΕΙΣ ΜΙΚΡΟΗΛΕΚΤΡΟΝΙΚΗ-ΩΛΣΙ, ΘΕΣΣΑΛΟΝΙΚΗ 2003

2. ΑΝΑΣΤ. Π. ΧΟΝΤΟΛΙΔΗΣ, ΣΗΜΕΙΩΣΕΙΣ ΕΡΓΑΣΤΗΡΙΑΚΩΝ ΑΣΚΗΣΕΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΙΣΧΥΟΣ, ΣΕΠΤΕΜΒΡΙΟΣ 2003

3. ΙΟΡΔΑΝΗΣ Ν. ΚΙΑΣΚΕΡΙΔΗΣ, ΗΛΕΚΤΡΟΝΙΚΑ ΙΣΧΥΟΣ ΘΕΣΣΑΛΟΝΙΚΗ 2004

4. ΠΑΠΑΒΡΑΜΙΔΟΥ ΠΑΝΑΓΙΩΤΑ, ΒΑΣΣΙΟΣ ΒΑΣΙΛΕΙΟΣ, ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΕ ΓΛΩΣΣΑ ΩΗΔΛ

5. ΧΡΗΣΤΟΣ Β. ΤΖΙΚΑΣ, ΕΡΓΑΣΤΗΡΙΑΚΕΣ ΑΣΚΗΣΕΙΣ ΨΗΦΙΑΚΩΝ ΚΥΚΛΩΜΑΤΩΝ ΙΙ, ΘΕΣΣΑΛΟΝΙΚΗ 2000

6. ΧΡΗΣΤΟΣ Β. ΤΖΙΚΑΣ, ΕΡΓΑΣΤΗΡΙΑΚΕΣ ΑΣΚΗΣΕΙΣ ΨΗΦΙΑΚΩΝ ΚΥΚΛΩΜΑΤΩΝ Ι, ΘΕΣΣΑΛΟΝΙΚΗ 2000

7. STEPHEN BROWN, ZVONKO VRANESIC, ΣΧΕΔΙΑΣΗ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΜΕ ΤΗ ΓΛΩΣΣΑ VHDL, ΘΕΣΣΑΛΟΝΙΚΗ 2001

8. ROGER L. TOKHEIM, ΕΙΣΑΓΩΓΗ ΣΤΑ ΨΗΦΙΑΚΑ ΗΛΕΚΤΡΟΝΙΚΑ, ΘΕΣΣΑΛΟΝΙΚΗ 2000

9. SYNOPSYS FPGA COMPILER II/FPGA EXPRESS VHDL REFERENCE MANUAL, ΜΑΙΟΣ 1999

10. Χ. ΚΑΨΑΛΗΣ, Π. ΚΩΤΤΗΣ, ΚΕΡΑΙΕΣ ΑΣΥΡΜΑΤΕΣ ΖΕΥΞΗΣ ΘΕΣΣΑΛΟΝΙΚΗ 2005

11. FRENZEL, ΗΛΕΚΤΡΟΝΙΚΕΣ ΕΠΙΚΟΙΝΩΝΙΕΣ, ΘΕΣΣΑΛΟΝΙΚΗ 1999