Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Δοκιμή ολοκληρωμένων κυκλωμάτων με χρήση του Inovys Personal Ocelot και του προτύπου IEEE Std. 1450 STIL (Standard Test Interface Language)** **Testing integrated circuits on Inovys Personal Ocelot with the use of IEEE Std. 1450 STIL (Standard Test Interface Language)**
Ονοματεπώνυμο Φοιτητή	**Ευστράτιος Γκλιάτης**
Πατρώνυμο	**Παναγιώτης**
Αριθμός Μητρώου	**ΜΠΣΠ/ 11005**
Επιβλέπων	**Ψαράκης Μιχαήλ, Επίκουρος Καθηγητής**

Ημερομηνία Παράδοσης **Ιανουάριος 2015**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Ψαράκης Μιχαήλ, Επίκουρος Καθηγητής	Κοτζανικολάου Παναγιώτης, Λέκτορας	Πατσάκης Κωνσταντίνος, Λέκτορας

Περιεχόμενα

Περίληψη

Σκοπός της παρούσης εργασίας είναι η μελέτη του προτύπου IEEE Std. 1450 STIL, των δυνατοτήτων που παρέχει ένας ελεγκτής ολοκληρωμένων κυκλωμάτων (hardware tester), όπως ο INOVYS PERSONAL OCELOT, και η δημιουργία ενός οδηγού χρήσης για τον παραπάνω ελεγκτή. Ο ελεγκτής Personal Ocelot μας δίνει τη δυνατότητα να επαληθεύσουμε την ορθή λειτουργία ενός νέου ολοκληρωμένου κυκλώματος προτού αυτό περάσει στο στάδιο της παραγωγής. Ο έλεγχος της ορθής λειτουργίας του ολοκληρωμένου κυκλώματος με την χρήση του ελεγκτή περιλαμβάνει: (α) την περιγραφή της αναμενόμενης συμπεριφοράς του ολοκληρωμένου κυκλώματος υπό δοκιμή, (β) την περιγραφή των δοκιμών (ή αλλιώς διανυσμάτων δοκιμής, test vectors), (γ) την σύνδεση του κυκλώματος στον ελεγκτή και (δ) την εκτέλεση των δοκιμών και συλλογή των αποτελεσμάτων. Η περιγραφή των αναμενόμενων εξόδων του κυκλώματος και των διανυσμάτων δοκιμής γίνεται με τη χρήση του προτύπου IEEE Std. 1450 STIL (standard test interface language). Το πρότυπο STIL, το οποίο προφέρεται ηθελημένα λάθος ως «στάιλ» από την ομάδα εργασίας ανάπτυξης του, είναι ένα πρότυπο το οποίο δημιούργησαν οι μεγαλύτερες εταιρίες στη βιομηχανία των κατασκευαστών ATE (automatic test equipment) και εργαλείων σχεδίασης (EDA tools, electronic design automation) για ευκολότερη και ταχύτερη εκτέλεση της διαδικασίας παραγωγής τους. Στα πλαίσια της παρούσας εργασίας μελετήθηκαν σε βάθος οι λειτουργικές δυνατότητες του ελεγκτή INOVYS PERSONAL OCELOT που διαθέτει το εργαστήριο Ενσωματωμένων Υπολογιστικών Συστημάτων του Πανεπιστημίου Πειραιά και του προτύπου STIL, και εκτελέστηκαν οι διαδικασίες ελέγχου ορθής λειτουργίας δύο (2) κυκλωμάτων μικρής κλίμακας: ενός αθροιστή και ενός πολλαπλασιαστή των 3-bit. Για την εκτέλεση των πειραματικών δοκιμών έγινε χρήση της πλακέτας FPGA Xilinx Spartan 3E στην οποία και υλοποιήθηκαν κυκλώματα ώστε να εκτελεστούν δοκιμές πραγματικών δεδομένων. Για την σύνδεση του ελεγκτή με τα ολοκληρωμένα κυκλώματα υπό δοκιμή, έγινε σχεδίαση και χρήση της πλακέτας Inovys Ocelot IO Board, η οποία και παρέχει σειρά ακροδεκτών (pin header) για τις εισόδους και εξόδους του ελεγκτή.

Abstract

The purpose of this paper is to study IEEE Std. 1450 STIL, explore the capabilities of a hardware tester such as INOVYS PERSONAL OCELOT, and create an operation guide for the tester. Personal Ocelot tester gives us the ability to verify the correct operation of a new hardware integrated circuit design before it enters the production stage. The complete verification includes: (a) the description of the expected behavior of the device under test (DUT), (b) the description of the tests to be executed (test vectors), (c) the physical connection of the integrated circuit to the tester and (d) the execution of the tests and the collection of the results. To describe the expected output of the integrated circuit and the input test vectors the user takes advantage of IEEE Std. 1450 STIL (standard test interface language, consciously mispronounced “style” by the working group). STIL is a standard that was created by the main ATE vendors (automatic test equipment) and EDA tool providers (electronic device automation) to simplify their production procedure. The scope of this paper was the in depth study of the operational capabilities of the hardware tester INOVYS PERSONAL OCELOT, provided by the Embedded Computing Systems Laboratory of the University of Piraeus, and the IEEE Std. 1450 STIL. The test of two (2) small scale integrated circuits was executed, an adder and a multiplier of 3-bit. For the implementation of the two circuits, the use of FPGA Xilinx Spartan 3E board was needed to perform actual test. To connect the hardware tester with the implemented circuits under test there was a PCB (printed circuit board) designed and used, that provided a pin header for the Input and Output pins of the hardware tester.

1.Εισαγωγή

Κατά τη διαδικασία παραγωγής ενός ολοκληρωμένου κυκλώματος ο εκάστοτε κατασκευαστής διαδέχεται δεδομένα στάδια από τη σύλληψη της ιδέας μέχρι την ολοκληρωτική απελευθέρωση του σχεδίου στην παραγωγική διαδικασία και την προώθηση του στην αγορά. Αρχικά καθορίζονται οι απαιτήσεις και οι επιθυμητές λειτουργίες του προϊόντος και στη συνέχεια ακολουθεί η συλλογή των επιμέρους υλικών που θα χρησιμοποιηθούν για την υλοποίηση ενός πρωτοτύπου. Έως την οριστική κατάληξη στο τελικό σχέδιο υπάρχει μια δύσκολη διαδρομή που εμπεριέχει εκτενείς δοκιμές για την αποκάλυψη των αδυναμιών και ελλείψεων του σχεδίου. Στο παρελθόν η πολυπλοκότητα των ολοκληρωμένων κυκλωμάτων παρέμενε σε επίπεδα ικανά να ελεγχθούν από εξειδικευμένους επιστήμονες επιφορτισμένους με το ρόλο αυτό. Εδώ και κάποια χρόνια το επίπεδο πολυπλοκότητας έχει γιγαντωθεί καθιστώντας πολλές φορές ανεπαρκή τον έλεγχο βασισμένο αποκλειστικά σε ανθρώπινο παράγοντα. Η κρισιμότητα και χρησιμότητα του ελέγχου και της επαλήθευσης ενός προς ανάπτυξη προϊόντος γίνονται εύκολα αντιληπτά και με γνώμονα τα λόγια του Έντσγκερ Γουάιμπε Ντάικστρα, Ολλανδού επιστήμονος της πληροφορικής, ο οποίος και ανέφερε χαρακτηριστικά

“The computing scientist’s main challenge is not to get confused by the complexities of his own making. Edsger W. Dijkstra

Η βασική πρόκληση του επιστήμονα της πληροφορικής είναι να μην του προκαλέσει σύγχυση η πολυπλοκότητα του δημιουργήματος του” (ελεύθερη μετάφραση)

Κάνοντας χρήση των λεγόμενων του μπορούμε να περιγράψουμε πρακτικά το σημείο στο οποίο κατέστει πλέον αναγκαία η χρήση αυτόματων εργαλείων (tester) στη διαδικασία ελέγχου και επαλήθευσης. Αρχικά και όσο τα προς υλοποίηση κυκλώματα παρέμεναν σε σχετικά απλό βαθμό απαιτήσεων οι εταιρίες κατασκεύαζαν δείγματα με breadboard και τρανζίστορ, chip και λογικές πύλες. Όσο όμως οι απαιτήσεις για πολυπλοκότητα και ταχύτητα λειτουργίας αυξάνονταν οι λύσεις αυτές δεν επαρκούσαν. Στη συνέχεια οδηγηθήκαμε στη χρήση FPGA προτύπων για δοκιμές και επαλήθευση των σχεδίων καθώς οι δυνατότητες που παρέχουν είναι αρκετά αυξημένες. Κατά τη διάρκεια του σταδίου δοκιμής και επαλήθευσης οι μηχανικοί των τμημάτων έρευνας και ανάπτυξης ισορροπούν μεταξύ δύο αντίθετων δυνάμεων. Την ανάγκη για έγκαιρη και όσο το δυνατόν ταχύτερη απελευθέρωση του σχεδίου στην αγορά και την ανάγκη για πλήρη κάλυψη των πιθανών λαθών του σχεδίου και της κατασκευής. Καθώς ο χρόνος μέχρι την απελευθέρωση ενός προϊόντος στην αγορά μπορεί να κρίνει την επιτυχή ή όχι κυκλοφορία του (time to market), η εκπλήρωση και των δύο στόχων παραμένει αντικρουόμενο γεγονός.

Η διαδικασία ελέγχου και επαλήθευσης ενός προϊόντος αποτελεί βασικό στάδιο για κάθε κατασκευαστή καθώς χωρίς αυτά τα στάδια το προϊόν μετά την απελευθέρωση του εμφανίζει προβλήματα και λάθη τα οποία ο κατασκευαστής δεν μπορεί να προβλέψει κατά το σχεδιασμό. Η γνωστή σε όλους μας στο χώρο της πληροφορικής διαδικασία αποσφαλμάτωσης (debugging) στην παραγωγή λογισμικού αποτελεί άλλωστε παράδειγμα σταδίων ελέγχου και επαλήθευσης προϊόντος, και οσοδήποτε μικρός η μεγάλος σε έκταση και αν είναι ο κώδικάς που παράγεται εύκολα ο αναγνώστης αντιλαμβάνεται την κρισιμότητα του σταδίου αποσφαλμάτωσης.

Ο έλεγχος και η επαλήθευση παρουσιάζουν ιδιαίτερες δυσκολίες ασχέτως προϊόντος καθώς εμπεριέχει εμπόδια άγνωστα την περίοδο του σχεδιασμού. Ένα βασικό θέμα είναι το γεγονός ότι πρέπει να δοκιμαστεί όσο το δυνατόν στο έπακρο η ορθή λειτουργία κάθε πτυχής ενός προϊόντος κάτι που μπορεί να μην είναι επακριβώς καθορισμένο ακόμα. Ίσως το βασικότερο εμπόδιο στην διαδικασία αυτή παραμένει η λειτουργία του προϊόντος, στην παρούσα φάση ενός κυκλώματος, σε πραγματικές συνθήκες και σε διάρκεια χρήσης. Τα εργαλεία αυτομάτου ελέγχου μας δίνουν τη δυνατότητα προσομοίωσης πραγματικών συνθηκών και εφαρμογών μεγάλης διάρκειας. Έτσι ο κατασκευαστής μπορεί να εκτελέσει δοκιμές στα όρια λειτουργίας του κυκλώματος (stress test) αποκτώντας το δυνατόν ολοκληρωμένη εικόνα για τη συνολική συμπεριφορά του προς υλοποίηση κυκλώματος.

Οι ελεγκτές ολοκληρωμένων κυκλωμάτων είναι το κλειδί στην επίλυση του θέματος αυτού. Οι ελεγκτές είναι συσκευές οι οποίες συνδέονται με τα προς ανάπτυξη ολοκληρωμένα κυκλώματα. Στη συνέχεια κάνοντας χρήση των ακροδεκτών εισόδου – εξόδου και των δυνατοτήτων που έχουν, προσομοιώνουν το περιβάλλον λειτουργίας ενός ολοκληρωμένου κυκλώματος και παρέχουν μαζικά πραγματικά δεδομένα εισόδου ελέγχοντας παράλληλα τα δεδομένα εξόδου του κυκλώματος. Πρακτικά στον ελεγκτή περιγράφεται η αναμενόμενη λειτουργία ενός κυκλώματος και καθορίζονται τα διανύσματα δοκιμής (test vectors) με τα οποία θα τροφοδοτηθεί το κύκλωμα από τον ελεγκτή. Στη συνέχεια ο ελεγκτής επαληθεύει την ορθή λειτουργία του κυκλώματος με βάση την συμπεριφορά που του έχει καθοριστεί από το χρήστη σε σχέση με τη συμπεριφορά που το κύκλωμα παρουσιάζει. Η δυνατότητα μαζικής, εκτενούς και ταχύτατης δοκιμής των ολοκληρωμένων κυκλωμάτων από τους ελεγκτές αποτελεί και το βασικό τους προσόν. Η εναλλακτική περίπτωση κατά την οποία ο κατασκευαστής θα δημιουργούσε κατάλληλες συνθήκες δοκιμής με υπολογιστή ή άλλο εργαλείο θα ήταν τρομερά χρονοβόρα και στην πλειοψηφία των περιπτώσεων δεν θα μπορούσε να πλησιάσει, πόσο μάλλον να καλύψει, τις πραγματικές συνθήκες χρήσης και λειτουργίας ενός ολοκληρωμένου κυκλώματος.

Στο πνεύμα επίλυσης του προβλήματος αυτού οι βιομηχανίες κατασκευής ολοκληρωμένων κυκλωμάτων αναζητούσαν τρόπους βελτίωσης του ποιοτικού ελέγχου και επιτάχυνσης της διαδικασίας επαλήθευσης (testing & verification). Παρόλα αυτά το γεγονός ότι κάθε εταιρία είχε δικά της εργαλεία χωρίς να υπάρχει ένα δεδομένο πρότυπο λειτουργίας πρόσθετε επιπλέον καθυστέρηση στη διαδικασία αυτή. Ειδικά σε περιπτώσεις όπου η ανάπτυξη περιλάμβανε συνεργασία μεταξύ εταιριών, όπως στην περίπτωση του επεξεργαστή PowerPC που αποτέλεσε το έναυσμα για τη δημιουργία του προτύπου, η καθυστέρηση γινόταν απαγορευτική για την επιτυχή εκπόνηση της διαδικασίας παραγωγής του. Έτσι στην προσπάθεια εύρεσης μιας μεθόδου που να συνδέει τα εργαλεία CAD των σχεδιαστών εργαλείων αυτοματισμού ηλεκτρονικών σχεδίων με τους προμηθευτές εξοπλισμού αυτόματου ελέγχου ξεκίνησε η προσπάθεια δημιουργίας του προτύπου IEEE Std. 1450 STIL. (Standard Test Interface Language).

Σκοπός της παρούσης είναι η ανάλυση της δομής, σύνταξης και χρήσης του προτύπου IEEE Std. 1450 STIL σε πραγματικές συνθήκες. Το εργαστήριο Ενσωματωμένων Υπολογιστικών Συστημάτων στάθηκε αρωγός στην προσπάθεια αυτή παρέχοντας τον ελεγκτή ολοκληρωμένων κυκλωμάτων που έχει στη διάθεση του, τον INOVYS PERSONAL OCELOT ο οποίος τυγχάνει να είναι από τους πιο ευρέως διαδεδομένους ελεγκτές στη βιομηχανία παραγωγής ολοκληρωμένων κυκλωμάτων. Το περιβάλλον χρήσης και λειτουργίας του ελεγκτή (Stylus) υποστηρίζει το πρότυπο IEEE Std. 1450 STIL. Για το σκοπό αυτό έγινε χρήση πραγματικών κυκλωμάτων μικρής κλίμακας υλοποιημένων σε πλακέτα FPGA XILINX Spartan 3E. Έτσι μας δόθηκε η ευκαιρία εμβάθυνσης στα εργαλεία του ελεγκτή σε υλοποιημένα ολοκληρωμένα κυκλώματα με την εφαρμογή πραγματικών δεδομένων. Κατά την διάρκεια μελέτης της παρούσης ο αναγνώστης έρχεται σε επαφή με τα διάφορα εργαλεία που χρησιμοποιήθηκαν και τα οποία επιγραμματικά αναφέρονται παρακάτω με τη σειρά χρήσης και παρουσίασης τους.

Το πρώτο κεφάλαιο της εργασίας αποτελεί κεφάλαιο εξοικείωσης του αναγνώστη με το πρότυπο IEEE Std. 1450 STIL. Γίνεται μια ιστορική αναδρομή από την πρώτη σύλληψη της ιδέας μέχρι την τελική αποδοχή του προτύπου από την ομάδα εργασίας του IEEE. Όπως γίνεται εύκολα κατανοητό το να φτάσουμε από μια ιδέα σε ένα πρότυπο είναι μια διαδικασία χρονοβόρα με διαρκείς εναλλαγές περιεχομένου και προσέγγισης του επιθυμητού αποτελέσματος από μέρους της ομάδας εργασίας. Στη συνέχεια παρουσιάζεται η δομή του αρχείου δοκιμής (test), ο τρόπος σύνταξης και τα απαραίτητα κομμάτια ενός ολοκληρωμένου test, γραμμένου με βάση το πρότυπο STIL. Στο τέλος παρατίθεται ένα αρχείο STIL αυτούσιο σαν παράδειγμα για να έχει ο αναγνώστης μια πιο ολοκληρωμένη εικόνα.

Το επόμενο κεφάλαιο αφιερώνεται στον ελεγκτή ολοκληρωμένων κυκλωμάτων INOVYS PERSONAL OCELOT καθώς η ενεργοποίηση και χρήση του ελεγκτή σε πραγματικές συνθήκες αποτελεί θεμελιώδες στόχο της παρούσης. Στο κομμάτι αυτό θα γίνει μια ιστορική αναδρομή αναφορικά με τη συσκευή, τα στοιχεία της, την προέλευση και την εξέλιξή της. Στη συνέχεια παρουσιάζεται το λογισμικό χρήσης του ελεγκτή (Verigy Stylus 3.1.6). Γίνεται εκτενής αναφορά στα επιμέρους μενού του λογισμικού και τις βασικότερες επιλογές σε κάθε περίπτωση.

Παρουσιάζεται η δημιουργία και εκτέλεση ενός απλού προγράμματος με χρήση τόσο μιας από τις δύο δοκιμαστικές πλακέτες που μας παρέχονται με το εργαλείο (ODD / EVEN PERSONAL OCELOT LB (load board)) όσο και με χρήση εικονικής σύνδεσης σε υλικό. Το εργαλείο εικονικής σύνδεσης παρέχεται για εκπαιδευτικούς σκοπούς καθώς δεν είναι πάντα διαθέσιμο στο χρήστη πραγματικό υλικό για δοκιμαστική εκτέλεση και πειραματισμό στον ελεγκτή. Περισσότερες λεπτομέρειες και ουσιαστική επεξήγηση των παραδειγμάτων ακολουθεί.

Για την χρήση και λειτουργία σε πραγματικές συνθήκες του ελεγκτή ολοκληρωμένων κυκλωμάτων Personal Ocelot θα υλοποιηθούν δύο βασικά κυκλώματα, ένας αθροιστής 3bit και ένας πολλαπλασιαστής 3bit. Εδώ θα παρουσιαστεί εκ νέου από την αρχή ο τρόπος δημιουργίας ενός test αρχείου δοκιμών για το εκάστοτε κύκλωμα, η φυσική σύνδεση του κυκλώματος στον ελεγκτή, ο προγραμματισμός του και φυσικά η εκτέλεση των δοκιμών. Κατά τη διάρκεια της εκτέλεσης των δοκιμών θα δούμε και τον τρόπο αποσφαλμάτωσης του εκάστοτε test και τον τρόπο πειραματισμού και αντοχής των χρονικών ορίων κάθε κυκλώματος. Για την υλοποίηση των κυκλωμάτων αυτών γίνεται χρήση του αναπτυξιακού εργαλείου FPGA Xilinx Spartan 3E με χρήση γλώσσας περιγραφής υλικού VHDL.

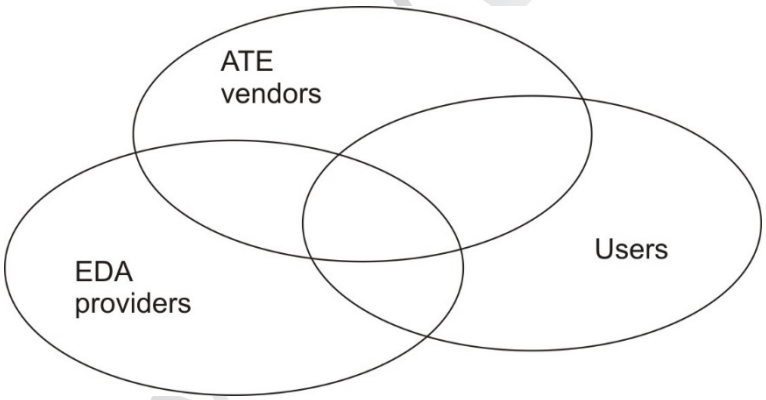
Για την καλύτερη κατανόηση θα ακολουθήσει παρουσίαση της υλοποίησης των δύο κυκλωμάτων στο εργαλείο Xilinx Spartan 3E. Ο τρόπος παρουσίασης θα είναι επικεντρωμένος στην υλοποίηση των επιθυμητών κυκλωμάτων καθώς η εκμάθηση και αναλυτική παρουσίαση του περιβάλλοντος Xilinx ISE Design Suite είναι εκτός σκοπού της παρούσης. Τα σημαντικά σημεία του περιβάλλοντος Xilinx ISE Design Suite για τον τύπο εφαρμογών που μας αφορά και μας εξυπηρετεί θα τονιστούν επαρκώς.

Τέλος παρατίθενται τα αρχεία που αφορούν την πλακέτα Ocelot IO Load board που σχεδιάστηκε για την ευκολότερη σύνδεσή ολοκληρωμένων κυκλωμάτων με τον ελεγκτή ολοκληρωμένων κυκλωμάτων Personal Ocelot. Η πλακέτα σχεδιάστηκε στο περιβάλλον σχεδίασης τυπωμένων κυκλωμάτων Easy-Pc και έχει τη λειτουργία αντάπτορα σύνδεσης εξωτερικού κυκλώματος με τον ελεγκτή ολοκληρωμένων κυκλωμάτων Personal Ocelot καθώς παρέχει τους ακροδέκτες εισόδου / εξόδου (IO pins) του ελεγκτή σε pin header με απόσταση μεταξύ των ακροδεκτών (pitch) 2.54mm και σταθερό πάχος ακροδεκτών για απρόσκοπτη και ασφαλή μελλοντική χρήση χωρίς κινδύνου φθοράς της πλακέτας IOPWIREBDSC που μας παρέχεται με τον ελεγκτή.

2.Πρότυπο IEEE Std. 1450 STIL (Standard Test Interface Language)

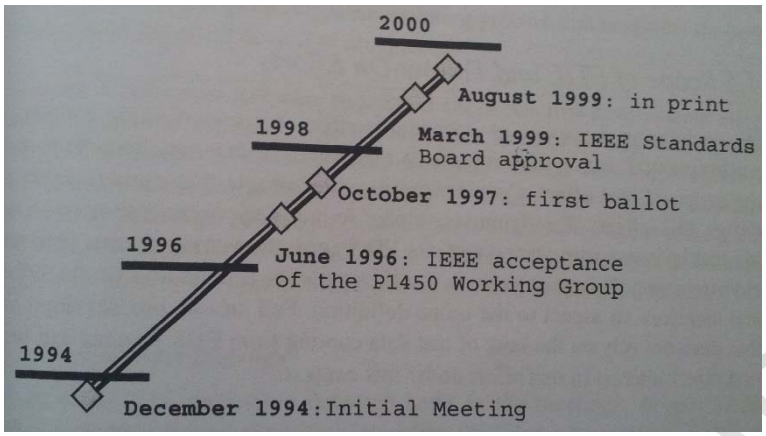
2.1 ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ

Το πρότυπο IEEE Std. 1450 - 1999 (Standard Test Interface Language - STIL) είναι μια προσπάθεια που ξεκίνησε σε ένα συνέδριο της Motorola στο Οστιν του Τέξας. Το συνέδριο, διοργανωμένο από το τμήμα της Motorola για τον επεξεργαστή PowerPC είχε σκοπό την συγκέντρωση των διαφορετικών εμπλεκομένων για τη δημιουργία ενός προτύπου που θα έλυνε ένα σοβαρό πρόβλημα της εποχής. Τόσο η Motorola όσο και η IBM κατά την ανάπτυξη του PowerPc έφτασαν σε ένα δεδομένο σημείο όπου χρειαζόταν η δημιουργία ενός προτύπου για τη διαδικασία ελέγχου και δοκιμών του υλικού ώστε να είναι δυνατή η μεταφορά πληροφορίας / φορητότητα των δεδομένων ελέγχου μεταξύ των δύο τμημάτων των εταιριών. Η κάθε εταιρία ακολουθώντας το δικό της πρότυπο για να χρησιμοποιήσει δεδομένα ελέγχου έχανε χρόνο για τη μετατροπή τους στη δικιά της μορφή. Έτσι μια ομάδα από προμηθευτές εξοπλισμού αυτόματου ελέγχου, σχεδιαστών εργαλείων αυτοματισμού ηλεκτρονικών σχεδίων και χρηστών των παραπάνω εργαλείων ξεκίνησε μια προσπάθεια σε αυτή την κατεύθυνση.



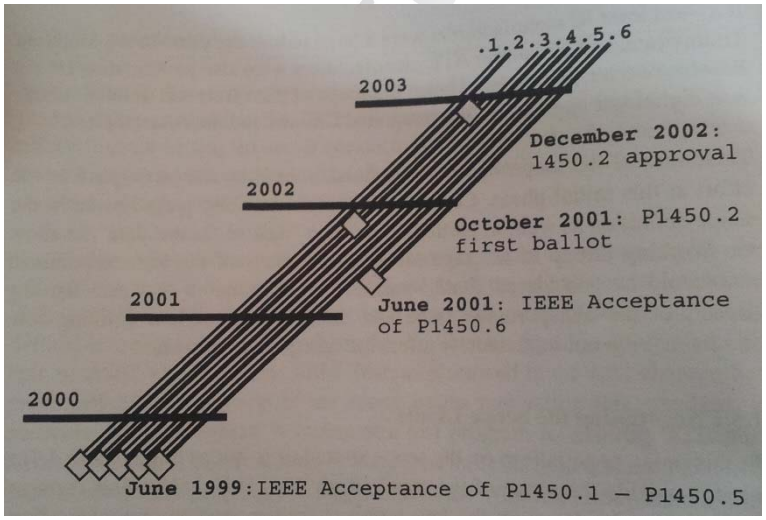
Εικόνα 1. Ιδρυτική τριανδρία του προτύπου STIL

Για να φτάσουμε στη δημοσίευση του αποτελέσματος πέρασαν 5 χρόνια καθώς ο οργανισμός IEEE ενέκρινε το πρότυπο το Μάρτιο του 1999 όπως φαίνεται και στο ακόλουθο σχήμα



Εικόνα 2. Χρονοδιάγραμμα προόδου του προτύπου STIL

Σημαντικό θέμα το οποίο και αντιμετώπισαν οι εμπλεκόμενες μεριές γρήγορα ήταν ο καθορισμός των ορίων της προσπάθειας τυποποίησης. Τί θα περιλάμβανε το πρότυπο; Μία συνήθης παγίδα είναι η γενίκευση της προσπάθειας τυποποίησης ενός θέματος σε τέτοιο βαθμό που γίνεται δυσκίνητο και χαώδες. Ως αποτέλεσμα προκαλείται μεγάλη καθυστέρηση και πολλές φορές και διακοπή των προσπαθειών από μη αποδοχή . Επιπλέον όταν μια ομάδα ανθρώπων γενικεύει πολύ το στόχο της και προσπαθεί να συμπεριλάβει πολλές πλευρές ενός θέματος έγκειται ο κίνδυνος η επικοινωνία και η συνεννόηση να δυσκολέψουν αρκετά. Αρχικός στόχος ήταν η αντιμετώπιση του προβλήματος του όγκου δεδομένων αναφορικά με τα patterns και το χρονισμό των δοκιμών. Ακολουθώντας την κοινή τακτική παρόμοιων προσπαθειών με αρχικό πλάνο αυτό το πρότυπο εμπλουτίστηκε με επιπλέον θεματικές ενότητες οι οποίες και ολοκληρώνονταν σταδιακά. Έτσι καταλήξαμε στην κατάσταση που περιγράφεται σχηματικά παρακάτω τόσο χρονικά όσο και θεματικά.



Εικόνα 3. Χρονοδιάγραμμα προόδου επεκτάσεων του προτύπου STIL

Έτσι αναφορικά με το θεματικό διαχωρισμό έχουμε την αρχική έκδοση που αφορούσε τη διεπαφή επικοινωνίας μεταξύ των εργαλείων παραγωγής test δοκιμών και του εξοπλισμού

εκτέλεσης δοκιμών. Εδώ καθορίστηκαν τα δεδομένα των διανυσμάτων ελέγχου, τα patterns, η διαμόρφωση, τα στοιχεία χρονισμού και οι βασικές αρχές του προτύπου.(IEEE Std. 1450.0). Η αρχική προσθήκη περιελάβανε την σχεδιαστική κατεύθυνση (IEEE Std. 1450.1-2005). εδώ καθορίστηκε η δομή του προτύπου στη περιγραφή των σχεδιαστικών εργαλείων, η χρήση της STIL γενικότερα στη φάση του σχεδιασμού μιας συσκευής - σχεδίου - μοντέλου. Στην επόμενη προσθήκη του προτύπου (IEEE Std. 1450.2 - 2002) θεματικός σκοπός ήταν τα επίπεδα ισχύος (DC LEVELS). Εδώ η ομάδα ανάπτυξης ασχολήθηκε με την περιγραφή της χρήσης της STIL για τον καθορισμό των επιπέδων ισχύος τροφοδοσίας καθώς και τις υπόλοιπες χρήσης ενέργειας τόσο αναφορικά με τον τρόπο δήλωσης όσο και με τον καθορισμό των σχετικών ορίων. Η επόμενη προσθήκη του προτύπου (IEEE Std. 1450.3-2007) αφορούσε τις επεκτάσεις της STIL για τις προδιαγραφές δοκιμών στα μηχανήματα ελέγχου. Μετά την κάλυψη αυτού του κομματιού έγινε ξεκάθαρο με ποιο τρόπο καθορίζει ο σχεδιαστής περιορισμούς στις κατασκευές με τη STIL σχετικά με δεδομένα περιβάλλοντα ελέγχου και δοκιμών. Με την επόμενη προσθήκη (IEEE Std. 1450.4) καλύφθηκε η ροή των δοκιμών. Εδώ καθορίστηκε ο τρόπος οργάνωσης , αλληλουχίας δοκιμών, επεξεργασίας των πιθανών σφαλμάτων και γενικότερα ο έλεγχος των δοκιμών προς εφαρμογή. Με τα υπόλοιπα κομμάτια (IEEE Std. 1450.5 - 1450.7) καθορίστηκαν επεκτάσεις για μεθόδους ελέγχου ημιαγωγών, καθώς και μεθόδους συγκεκριμένων ελέγχων. Ένα επιπλέον κομμάτι αφορά στη εκ νέου χρήση δοκιμών και πληροφορίας από ήδη καθορισμένα λογικά μπλοκ (πυρήνες). Επίσης καλύφθηκε η χρήση αναλογικών και μεικτών σημάτων επικοινωνίας. Η πιο πρόσφατη εξέλιξη του προτύπου αφορά αυτή ακριβώς την κατεύθυνση, τα αναλογικά και μικτά σήματα. (STIL AMS : Analog & mixed signal). Απώτερος στόχος είναι η χρήση αναλογικών σημάτων για διέγερση συμβάντων και μετρήσεις καθώς και συγχρονισμός με ψηφιακά patterns ελέγχων.

IEEE ident.	Title/Purpose
P1450.1	*Extensions to STIL for Semiconductor Design Environments.* Supports EDA-specific application of STIL data, test development and test refinement processes. Defines constructs (notably Variables) applied in other subproject efforts as well, and as such is an important extension for additional STIL work.
1450.2-2002	*Extensions to STIL for DC Level Specification.* Identifies voltage, current, and other parametric relationships to apply or measure during test operation.
P1450.3	*Extensions to STIL for Tester Target Specification.* Provides mechanisms to identify limitations/constraints on STIL constructs as applied to specific test environments. Also used as a metric for an existing STIL test, to identify how much or how many elements are present.
P1450.4	*Extensions to STIL for Test Flow Specification.* Addresses test sequencing, organization, failure processing, and overall test control.
P1450.5	*Extensions to STIL for Semiconductor Test Method Specification.* An adjunct to the flow definition, provides definition for the constructs necessary to perform specific test operations.
P1450.6	*Extensions to STIL for Core Test Language (CTL) Support.* Defines constructs and test organizations to facilitate the reuse of test information generated for previously defined logic blocks ("cores").

Εικόνα 4. Περιεχόμενα επεκτάσεων προτύπου STIL

Το πρότυπο IEEE Std. 1450 - 1999 (Standard Test Interface Language - STIL) αποτελεί μια διαρκώς εξελισσόμενη προσπάθεια. Για τη δημιουργία και την εξέλιξή του ασχολήθηκαν συμμετέχοντες από διάφορους κλάδους. Παραδείγματα εταιριών που ασχολούνται με εργαλεία αυτομάτου ελέγχου και εταιρίες σχεδιασμού κυκλωμάτων καθώς και εταιρίες αυτοματισμών και ηλεκτρονικών που συμμετείχαν σαν χρήστες λόγω άμεσου ενδιαφέροντος αναγράφονται στους ακόλουθους πίνακες.

Κατασκευαστής	**Εργαλεία**	**Περιγραφή**
Advantest	T3324	VLSI Tester
Advantest	T3326	Memory Tester
Advantest	T5335P	Memory Tester
Advantest	T5335	Memory Tester
Advantest	T3347	Memory Tester
Advantest	T5365P	Memory Tester
Advantest	T5371	Memory Tester
Advantest	T5381	Memory Tester
Advantest	T5382A	Memory Tester
Advantest	T5381H	Memory Tester
Advantest	T6683	SOC Test System
Agilent / HP	4062UX	Parametric Test System
Agilent	4071A	Parametric Test System
Agilent / HP	82000	VLSI tester
Agilent / HP	83000	VLSI tester
Agilent / HP	84000	RFIC Tester
Agilent / HP	94000	Mixed Signal Test System for Bluetooth applications
Agilent / HP	93000	SOC Test System
Credence	ASL1000	Mixed Signal Tester
Credence	Duo	Mixed Signal Tester
Credence	Duo XP	Mixed Signal Tester
Credence	Duo SX	Mixed Signal Tester
Credence	Octet 200	Mixed Signal Tester
Credence	Quartet	Mixed Signal Tester
Credence	STS 8256	Mixed Signal Tester
Credence	Vista Vision	Mixed Signal Tester
EDAptive Computing		
Finley Design		Tester Loadboards
IMS	XL, XL2, ATS, FT & MX, Blazer	
LTX	Fusion MX	Mixed Signal Tester
LTX	Fusion CX	Mixed Signal Tester
MCT	2000, 2020, 3100	
Mentor		
Mosaid	MS4155	Memory Tester
Mosaid	MS4205EX	Bench Tester
Mosaid	MS3480	Memory Tester
Mosaid	MS3495	Memory Tester
Nextest	Maverick PT, VT	
Roos Instruments		
Sentry	S10	10 MHz Digital Tester

Sentry	S20	20 MHz Digital Tester
Sentry	Sentinel	Digital Tester
Sentry	ITS9000KX	VLSI Logic Test System
Sentry	S1650	Digital IC Test System
Schlumberger	IDS 3000, 5000, 10000, ITS9000, FX, SX, EX, EXA, IX, AX, ZX	
Shibakosu	WL-93A	
Synopsis		
Teradyne	A567	Mixed Signal Tester
Teradyne	A575	Mixed Signal Tester
Teradyne	A585	Mixed Signal Tester
Teradyne	J937	Memory Test System
Teradyne	J971	Logic Test System
Teradyne	J993	Memory Test System
Teradyne	J995	Memory Test System
Teradyne	J997	Memory Test System
TI	Impact, V Series	
Versatest	VT2104	Memory Tester
Xincom	5582	

Πίνακας 1. Κατασκευαστές Ελεγκτών Υλικού, εργαλείων αυτόματου ελέγχου

Κατασκευαστής
3M/Textool
AMP
AQL
Aries
ASE
Azimuth
Enplas
Exatron
Gold Technologies (Gtec)
HP Test
Johnstech
Loranger
Nepethe
Oztek/K&S
Plastronics
Robinson Nugent
RS Tech
Tecknit

Weel/CTI
Yamaichi
VN-TEK

Πίνακας 2. Κατασκευαστές ολοκληρωμένων κυκλωμάτων

Κατασκευαστής	**Εργαλεία**
Aetrium	
Aseco	S130, S170, S450
Daymarc	1156/1157, 757, Z50
Delta Flex	
Exatron	
Fico	
Kuwano	
MCT	3600, 3608, 3616, 4600, 4610, 5100
Multitest	
Rasco	
Seiko Epson	
Symtek	300, 360, 7936, 429, KP, KL
Synax	
Tesam,Tesec	
Trigon	

Πίνακας 3. Κατασκευαστές μηχανών τοποθέτηση και συναρμολόγησης

Εταιρία
Freescale
Infineon
Intel
Panasonic
Philips
ST Micro
TI
Toshiba
GE Consumer & Industrial
LOAD Technology
SeginSemi
Syswave

Πίνακας 4. Χρήστες εργαλείων λειτουργίας με βάση το πρότυπο STIL

Με το πέρασμα του χρόνου παρατηρείτε μια τάση όλο και περισσότερης αφομοίωσης και χρήσης του προτύπου IEEE Std. 1450. Έτσι εμφανίζονται προσπάθειες εμπλουτισμού και βελτίωσης της υπάρχουσας έκδοσης του προτύπου. Έτσι οι τελευταίες ανανεώσεις του προτύπου χρονικά έγιναν μέσα στο 2014 , τον Απρίλιο συγκεκριμένα. Με πρωτοβουλία του οργανισμού SEMI ο οποίος και δέχεται εδώ και 40 χρόνια σαν μέλη εταιρίες κατασκευαστές μίκρο- και νάνο-ηλεκτρονικών εφαρμογών, και με την ευκαιρία της έκθεσης SEMICON στην Ιαπωνία το 2013 έγιναν συναντήσεις και ανοιχτές συζητήσεις για την περαιτέρω εξέλιξη της STIL καθώς με την ολοκλήρωση και τελειοποίηση του IEEE Std. 1450.4 σχετικά με τη ροή των ελέγχων , την επεξεργασία των σφαλμάτων και γενικότερα των ελέγχων των δοκιμών και τις επεκτάσεις που ετοιμάζονται σε αυτήν την κατεύθυνση το πρότυπο STIL θα αποτελέσει ένα ακόμα πιο ισχυρό εργαλείο για τους εμπλεκόμενους οργανισμούς, φορείς , εταιρίες και απλούς χρήστες.

2.2 ΒΑΣΙΚΑ ΣΤΟΙΧΕΙΑ ΠΡΟΤΥΠΟΥ

2.2.1 Γενική περιγραφή στοιχείων

Η κεντρική ιδέα του προτύπου είναι η περιγραφή του τρόπου δοκιμών ολοκληρωμένων κυκλωμάτων σε μορφή υλικού. Βασικό στοιχείο στον τρόπο προσέγγισης αποτελεί όμως η εκάστοτε εφαρμογή καθώς με άλλο γνώμονα προχωρά ο χρήστης σε περίπτωση μικροεπεξεργαστή / μικροελεγκτή και με άλλο στην περίπτωση ολοκληρωμένου κυκλώματος εξειδικευμένης εφαρμογής (Application Specific Integrated Circuit, ASIC). Στις περιπτώσεις μικροεπεξεργαστών / μικροελεγκτών ο όγκος παραγωγής είναι μεγάλος και η ακρίβεια και η ταχύτητα στη διαδικασία ελέγχου πολύ κρίσιμη. Δεν αποτελεί βασική ανάγκη η φορητότητα ενώ τα patterns που θα εφαρμοστούν είναι πολύ στοχευμένα. Επιπλέον η παραμικρή μείωση στο χρόνο ελέγχου μπορεί να αποβεί ιδιαίτερα προσοδοφόρα στο σύνολό της. Στον αντίποδα στην περίπτωση των ASIC ολοκληρωμένων κυκλωμάτων ο όγκος παραγωγής είναι μικρότερος αλλά και το εύρος αναγκαίου ελέγχου πολύ πιο καθορισμένο και περιορισμένο. Έχει όμως μεγαλύτερη κρισιμότητα η φορητότητα του τρόπου ελέγχου σε διαφορετικούς τύπους ελεγκτών.

Το πρότυπο μπορεί να εκτελεί συγκεκριμένου τύπου ελέγχους στα ολοκληρωμένα κυκλώματα. Οι βασικοί τύποι εξ αυτών είναι οι παρακάτω.

- Έλεγχος σύνδεσης μεταξύ του υπό δοκιμή ολοκληρωμένου κυκλώματος (DUT Device under Test) και του ελεγκτή, και των βασικών λειτουργιών των ακροδεκτών
- Λειτουργικός έλεγχος του υπό δοκιμή ολοκληρωμένου κυκλώματος (DUT) στη βασική λειτουργία του
- Δομικός έλεγχος διασυνδέσεων και ορθής λειτουργίας στο χαμηλότερο επίπεδο υλικού
- Έλεγχος χρονικής συμπεριφοράς
- Μέτρηση ρεύματος σε κατάσταση ηρεμίας, ανίχνευση ελαττωμάτων που προκαλούν διαρροές
- Επικύρωση προδιαγραφών εισόδου / εξόδου
- Δοκιμή σε τιμές κατωφλίου με βάση τις προδιαγραφές λειτουργίας

Πρακτικά με τη χρήση του προτύπου περιγράφουμε τη λειτουργία του υπό δοκιμή ολοκληρωμένου κυκλώματος (DUT) στον ελεγκτή τόσο χρονικά όσο και από πλευράς τάσεων λειτουργίας, ακροδεκτών εισόδου και ακροδεκτών εξόδου. Αρχικά περιγράφουμε πίνακες με κυματομορφές (Waveform Tables) οι οποίοι και εκφράζουν τις εκάστοτε εισόδους του ολοκληρωμένου κυκλώματος υπό δοκιμή (DUT). Στην ουσία καθορίσουμε τα σήματα (signals) που δέχεται το DUT υπό μορφή διανυσμάτων και τη χρονική συμπεριφορά τους. Στο κομμάτι του χρονισμού και των προδιαγραφών (Timing Section, Specs Section) γίνεται ακριβής περιγραφή της χρονικής συμπεριφοράς και των τάσεων εισόδου , εξόδου καθώς και των λοιπών προδιαγραφών κατά περίπτωση. Στο τμήμα των pattern ορίζουμε τόσο την είσοδο όσο και την

έξοδο των ακροδεκτών. Τα patterns φορτώνονται στον ελεγκτή και εκτελούν τους επιθυμητούς ελέγχους στον εκάστοτε ελεγκτή ολοκληρωμένων κυκλωμάτων. Τα pattern που εκτελούνται σε κάθε ολοκληρωμένο έλεγχο σε ένα ελεγκτή ολοκληρωμένων κυκλωμάτων που λειτουργεί με βάση το πρότυπο IEEE Std. 1450 STIL, μας οδηγούν κα στο αποτέλεσμα είτε της πετυχημένης είτε της αποτυχημένης προσπάθειας ελέγχου.

Στο Παράρτημα 1 γίνεται παρουσίαση της Backus-Naur μορφής του προτύπου όπου και ο αναγνώστης έχει την ευκαιρία να μελετήσει τη βασικής σύνταξης των στοιχείων ενός αρχείου προτύπου IEEE Std. 1450 STIL. Η Backus-Naur μορφή του προτύπου δεν αποτελεί ολοκληρωμένη περιγραφή του και δεν αντικαθιστά σε καμία περίπτωση το ακριβές κείμενο του αλλά χρησιμοποιείται σαν οδηγός επεξήγησης.

Ένα από τα βασικά στοιχεία του προτύπου είναι η δυνατότητα σπονδυλωτής δημιουργίας patterns. Καθώς τα ολοκληρωμένα κυκλώματα παρουσιάζουν επικαλύψεις λειτουργιών και δομικών μονάδων, παρέχεται η δυνατότητα ύπαρξης patterns ελέγχου συγκεκριμένων δομών που επανεμφανίζονται σε παραπάνω του ενός ολοκληρωμένα κυκλώματα. Έτσι μπορούμε patterns που έχουμε χρησιμοποιήσει στο παρελθόν να τα ξαναχρησιμοποιούμε σε νέα ολοκληρωμένα κυκλώματα με ελάχιστη η καθόλου τροποποίηση. Επιπλέον με τη χρήση της σύνταξης του προτύπου ο χρήστης μπορεί να δημιουργήσει επεκτάσεις που να σχετίζονται με συγκεκριμένο υλικό ελεγκτή ολοκληρωμένων κυκλωμάτων. Με αυτό τον τρόπο κάθε χρήστης ασχέτως ποιου εργαλείου χρησιμοποιεί μπορεί να παραμετροποιήσει κατάλληλα το αρχείο προτύπου ώστε να συμβαδίζει με το ακριβές περιβάλλον και εργαλείο χρήσης.

Οι λεπτομέρειες και η συνολική δομή θα γίνουν περισσότερο κατανοητά στα ακόλουθα κεφάλαια όπου και θα παρουσιαστεί το περιβάλλον του Inovys Personal Ocelot καθώς και η ανάπτυξη παραδειγμάτων σε αυτό μα χρήση του προτύπου IEEE Std. 1450 STIL.

2.2.2 Παράδειγμα αρχείου STIL

Ακολουθεί παράθεση αρχείου STIL που αφορά στη δοκιμή του ολοκληρωμένου κυκλώματος 74LS245 της Motorola (Octal bus transceiver).

```
STIL 0.0;
Signals {
    DIR In;
    OE_ In;
    A0 InOut; A1 InOut; A2 InOut; A3 InOut;
    A4 InOut; A5 InOut; A6 InOut; A7 InOut;
    B0 InOut; B1 InOut; B2 InOut; B3 InOut;
    B4 InOut; B5 InOut; B6 InOut; B7 InOut;
}
SignalGroups {
    ABUS 'A7 + A6 + A5 + A4 + A3 + A2 + A1 + A0';
    BBUS 'B7 + B6 + B5 + B4 + B3 + B2 + B1 + B0';
    BUSES 'ABUS + BBUS';
    ALL 'DIR + OE_ + BUSES';
}
SignalGroups more {
    ABUS_I 'ABUS' { Base Hex 01; }
    BBUS_I 'BBUS' { Base Hex 01; }
}
Spec tmode_spec {
```

```
    Category tmode {
          tplh { Typ ’8.00ns’; Max ’12.00ns’; }
          tphl { Typ ’8.00ns’; Max ’12.00ns’; }
          tpzl { Typ ’27.00ns’; Max ’40.00ns’; }
          tpzh { Typ ’25.00ns’; Max ’40.00ns’; }
          tplz { Typ ’15.00ns’; Max ’25.00ns’; }
          tphz { Typ ’15.00ns’; Max ’25.00ns’; }
          strobe_width ’20ns’;
    }
}
Selector tmode_typ {
    tplh Typ;
    tphl Typ;
    tpzl Typ;
    tpzh Typ;
    tplz Typ;
    tphz Typ;
}
Timing to_specs {
    WaveformTable pulsed_oe {
          Period ’500ns’;
          Waveforms {
          DIR{ 01{ ’0ns’ D/U; }}
          OE_{ 01{ ’0ns’ U; OE_MARK: ’200ns’ D/U;
                OE_CLOSE: ’OE_MARK+100ns’ U; }}
          BUSES{ 01{ ’10ns’ D/U; }
                L { ’0ns’ Z;’0ns’ X; ’OE_MARK+tpzl’ l;
                      ’@+strobe_width’ X;}
                H { ’0ns’ Z;’0ns’ X; ’OE_MARK+tpzh’ h;
                      ’@+strobe_width’ X;}
                D { ’0ns’ Z;’0ns’ X; ’OE_CLOSE+tplz’ t;
                      ’@+strobe_width’ X;}
                U { ’0ns’ Z;’0ns’ X; ’OE_CLOSE+tphz’ t;
                      ’@+strobe_width’ X;}
                X { ’0ns’ Z;’0ns’ X; }}
    } // end Waveforms
} // end WaveformTable pulsed_oe
WaveformTable const_oe {
    Period ’500ns’;
    Waveforms {
    DIR{ 01{ ’0ns’ D/U; }}
    OE_{ 01{ ’0ns’ D; ’480ns’ U;}}
    BUSES{ 01{ IN_MARK: ’50ns’ D/U; }
                L { ’0ns’ Z;’0ns’ X; ’IN_MARK+tphl’ l;
```

```
                    '@+strobe_width' X;}
                H { '0ns' Z;'0ns' X; 'IN_MARK+tplh' h;
                    '@+strobe_width' X;}
                X { '0ns' Z;'0ns' X; }}
          } // end Waveforms
    } // end WaveformTable const_oe
} // end Timing to_specs
PatternBurst spec_check_burst {
    spec_check;
} //end PatternBurst spec_check_burst
PatternExec {
    SignalGroups more;
    Timing to_specs;
    Selector tmode_typ;
    Category tmode;
    spec_check_burst;
} //end PatternExec
Pattern spec_check {
    W pulsed_oe;
    // the first vector must specify states on all signals.
V { ALL=00DDDDDDDDXXXXXXXX; }
// first set of tests check delays from OE_ signal
// check BBUS tpzl spec
V { ABUS_I=00; BBUS=LLLLLLLL; }
//check BBUS tpzh spec
V { ABUS_I=FF; BBUS=HHHHHHHH; }
// check BBUS tplz spec
V { ABUS_I=00; BBUS=DDDDDDDD; }
// check BBUS tphz spec
V { ABUS_I=FF; BBUS=UUUUUUUU; }
V { DIR=1; ABUS=XXXXXXXX;
BBUS=DDDDDDDD; }
// check ABUS tpzl spec
V { BBUS_I=00; ABUS=LLLLLLLL; }
// check ABUS tpzh spec
V { BBUS_I=FF; ABUS=HHHHHHHH; }
// check ABUS tplz spec
V { BBUS_I=00; ABUS=DDDDDDDD; }
// check ABUS tphz spec
V { BBUS_I=FF; ABUS=UUUUUUUU; }
W const_oe;
// second set of tests check data propagation delays
// check ABUS tphl spec
V { BBUS_I=00; ABUS=LLLLLLLL; }
```

```
  // check ABUS tplh spec
  V { BBUS_I=FF; ABUS=HHHHHHHH; }
  V { DIR=0; BBUS=XXXXXXXX;
  ABUS=DDDDDDDD; }
  V { ABUS_I=00; BBUS=LLLLLLLL; }
  // check BBUS tplh spec
  V { ABUS_I=FF; BBUS=HHHHHHHH; }
  // check BBUS tphl spec
  V { ABUS_I=00; BBUS=LLLLLLLL; }
} //end Pattern spec_check
```

2.2.3 Επεξήγηση βασικών μπλοκ κώδικα αρχείου STIL

Signals {}

Στον κώδικα αρχείου STIL που περικλείεται στο μπλοκ Signals {} περιγράφουμε τα υπάρχοντα σήματα καθώς και το εάν είναι σήματα εισόδου, εξόδου η και εισόδου και εξόδου ανάλογα την δεδομένη χρονική λειτουργία. Επιπλέον υπάρχει και το ψεύδο-σήμα το οποίο και συνδέεται με τη διαδικασία ελέγχου χωρίς απαραίτητα να είναι κομμάτι του υλικού και το σήμα τροφοδοσίας. Στην πράξη κάθε σήμα αντιστοιχίζεται με έναν ακροδέκτη (pin) του υπό δοκιμή ολοκληρωμένου κυκλώματος. Ο διαχωρισμός γίνεται με τις λέξεις κλειδιά

In

Out

InOut

Pseudo

Supply

Ακολουθεί ο πλήρης τρόπος δήλωσης των ορισμάτων του μπλοκ.

```
Signals {
      signal_name ;
            -or-
      signal_name {
        Termination     (TerminateHigh,     TerminateLow,TerminateOff,
TerminateUnknown) ;
        DefaultSate (U, D, Z) ;
        Base (Hex, Dec) wfclist     ;
        Alignment (MSB, LSB)  ;
        ScanIn optional_integer_value     ; - or –
        ScanOut optional_integer_value    ;
        DataBitCount integer  ;
}}
```

SignalGroups {}

Στον κώδικα αρχείου STIL που περικλείεται στο μπλοκ SignalGroups {} ομαδοποιούμε τα σήματα σε γκρουπ σημάτων τα οποία και θα χρησιμοποιήσουμε αργότερα στα pattern για να καθορίσουμε τη συμπεριφορά των σημάτων.
Παράδειγμα δήλωσης SignalGroup και αντιστοίχησης τιμών σε αυτά :

```
group = 'A + B + C'     ;
V { group = 101; }
```

Όπου και το σήμα A παίρνει την τιμή 1, το σήμα B την τιμή 0 και το σήμα C την τιμή 1.

Ακολουθεί ο πλήρης τρόπος δήλωσης των ορισμάτων του μπλοκ.

```
SignalGroups optional_signalgroups_block_name {
      group_name = ' signal_reference_expression'     ;
      -  or –
      group_name = 'signal_reference_expression' {
            signal_and_group_attribute_statements     ;
      }  }
```

Spec {}
Στον κώδικα αρχείου STIL που περικλείεται στο μπλοκ Spec {} καθορίσουμε κατά βάση τις προδιαγραφές αναφορικά με το χρονική συμπεριφορά και τις τάσεις λειτουργίας. Εδώ μπορούμε να δώσουμε σε κάθε προδιαγραφή που γράφουμε τυπική, ελάχιστη και μέγιστη τιμή. Ο καθορισμός των προδιαγραφών όπως θα δούμε και αργότερα στο περιβάλλον Stylus του ελεγκτή Inovys Personal Ocelot μπορεί να γίνει με διάφορους τρόπου είτε απόλυτα αναγράφοντας τιμή είτε ονομάζοντας μια προδιαγραφή και χρησιμοποιώντας την επιθυμητή ονομασία της σε κάθε Pattern.

Type	Keyword	Behavior
Input Voltage	**VIH**	Input high voltage
Input Voltage	**VIL**	Input low voltage
Input Voltage	**VICM**	Common-mode differential voltage
Input Voltage	**VID**	Differential input voltage
Input Voltage	**VIHD**	Differential input high voltage
Input Voltage	**VILD**	Differential input low voltage
Output Voltage	**VOH**	Output high voltage measured
Output Voltage	**VOL**	Output low voltage measured
Output Voltage	**VOCM**	Common-mode differential voltage
Output Voltage	**VOD**	Differential output voltage
Output Voltage	**VOHD**	Differential output high voltage
Output Voltage	**VOLD**	Differential output low voltage
Slew Rates	**VIHSlew**	Input voltage high slew rate

Slew Rates	**VILSlew**	Input voltage low slew rate
Output Currents	**IOH**	Output high load current
Output Currents	**IOL**	Output low load current
Other Statements	**LoadRef**	Load threshold or commutating voltage
Other Statements	**ClampHi, ClampLo**	Voltages high and low clamps
Other Statements	**RestistiveTermination**	Terminating resistance
Other Statements	**TermVRef**	Terminating voltage
Other Statements	**VForce, IForce**	Forcing voltage, current
Other Statements	**IClamp, VClamp**	Maximum (clamp) current, voltage

Πίνακας 5. Δηλώσεις επιπέδων τάσης με βάση το πρότυπο IEEE Std. 1450 STIL

Type	**Keyword**	**Behavior**
Override statements	**ForceHi, ForceLo**	Overrides all pattern data on the Signals; holds Signals at a high or low logic level
Override statements	**InitHi, InitLo**	Holds Signals at high or low logic level before the Pattern Exec starts

Πίνακας 6. Δηλώσεις επιπέδων τάσης παράκαμψης με βάση το πρότυπο IEEE Std. 1450 STIL

Timing {}

Στον κώδικα αρχείου STIL που περικλείεται στο μπλοκ Timing {} καθορίζεται και η βασική συμπεριφορά σε σχέση με το χρόνο του εκάστοτε ακροδέκτη. Εδώ συνδυάζεται ο ακροδέκτης με τη χρονική συμπεριφορά, την τάση λειτουργίας και την τρέχουσα κατάσταση του.

Εδώ ορίζεται η περίοδος λειτουργίας του ολοκληρωμένου κυκλώματος. Έτσι έχουμε την εντολή Period ‘500ns’ όπου και ορίζεται περίοδος στα 500 nanoseconds.

Εδώ καθορίζονται και οι πίνακες κυματομορφών (WaveformTables). Ομαδοποιούμαι συμπεριφορές σημάτων, τις ονομάζουμε και χρησιμοποιούμαι τα ονόματα αυτά για να περιγράψουμε μετά στα patterns την επιθυμητή ή αναμενόμενη συμπεριφορά για τη διαδικασία ελέγχου του ολοκληρωμένου κυκλώματος. Έτσι εδώ βλέπουμε τη δημιουργία 2 Waveform tables με ονόματα pulsed_oe, και const_oe με τις παρακάτω εντολές

WaveformTable pulsed_oe {}

WaveformTable const_oe {}

Κάθε Waveform Table μπορεί να έχει τη δική του περίοδο και διαφορετική συμπεριφορά στους ακροδέκτες του, όπως καθορίζεται μέσα στο μπλοκ κώδικα. Οι ακροδέκτες των ολοκληρωμένων με βάση το πρότυπο STIL μπορούν να βρίσκονται σε συγκεκριμένου τύπου καταστάσεις.

Type	**State**	**Behavior**
Drive	**D** or **ForceDown**	Assert a “low” state on the Signal
Drive	**U** or **ForceUp**	Assert a “high” state on the Signal
Drive	**Z** or **ForceOff**	Remove or disconnect a drive state
Drive	**P** or **ForcePrior**	Reapply the last D or U asserted
Drive	**N** or **ForceUnknown**	Provide a drive state, but unknown value

Compare	**L** or **CompareLow**	Detect a “low” state on the Signal
Compare	**H** or **CompareHigh**	Detect a “high” state on the Signal
Compare	**X**, **x** or **CompareUnknown**	Remove or discontinue a detect state
Compare	**T** or **CompareOff**	Detect a mid-band state on signal
Compare	**v** or **CompareValid**	Detect either a “high” or “low” state
Compare	**l** or **CompareLowWindow**	Detect a “low” region on the signal
Compare	**h** or **CompareHighWindow**	Detect a “high” region on the signal
Compare	**t** or **CompareOffWindow**	Detect a mid-band region
Compare	**v** or **CompareValidWindow**	Detect either a “high” or “low” region on the signal
Expect	**R** or **ExpectLow**	Expect a “low” state
Expect	**G** or **ExpectHigh**	Expect a “high” state
Expect	**Q** or **ExpectOff**	Expect a “mid-band” state
Expect	**M** or **Marker**	Provide a timing reference point
Indeterminate	**A** or **LogicLow**	Unknown direction but “low” logic state
Indeterminate	**B** or **LogicHigh**	Unknown direction “high” logic state
Indeterminate	**F** or **LogicZ**	Unknown direction “mid-band” logic state
Indeterminate	**?** or **Unknown**	Nothing is known about the state

Πίνακας 7. Καταστάσεις σημάτων με βάση το πρότυπο IEEE Std. 1450 STIL

Στα χρονικά σημεία τύπου Compare ο εκάστοτε ελεγκτής εκτελεί μέτρηση του δεδομένου σήματος για την ακριβή τιμή του. Πριν επιλέξουμε σημείο μέτρησης λαμβάνουμε υπ’ όψη τυχόν χρονοκαθυστέρηση ώστε το σήμα να προλάβει να πάρει την τιμή της κατάστασης που θέλουμε να μετρήσουμε. Σκοπός ύπαρξης των σημάτων τύπου Expect είναι η απεικόνιση της πραγματικής χρονικής συμπεριφοράς. Στη διαδικασία ελέγχου μπορεί και να μην λαμβάνεται υπ’ όψη καθόλου.

State	**Level**
D or **ForceDown**	**VIL**
U or **ForceUp**	**VIH**
L or **CompareLow**	**VOL**
l or **CompareLowWindow**	**VOL**
H or **CompareHigh**	**VOH**
h or **CompareHighWindow**	**VOH**

Πίνακας 8. Συσχετισμός κατάστασης σήματος και επιπέδου τάσης με βάση το πρότυπο IEEE Std. 1450 STIL

Ακολουθεί ο πλήρης τρόπος δήλωσης των ορισμάτων του μπλοκ DCLevels. Ένα μπλοκ κώδικα που καθορίζει συγκεκριμένα επίπεδα τάσης και μπορεί να χρησιμοποιείται σε συνδυασμό με το μπλοκ specs.

```
DCLevels optional_dclevels_block_name {
      InheritDCLevels dclevels_block_name ;
      SignaGroups signalgroup_block_name  ;
      Signal_expr {
        Dclevels_statements   ;
      }
}
```

PatternBurst {}

PaternExec {}

Pattern {}

Στον κώδικα αρχείου STIL που περικλείεται στα 3 παραπάνω μπλοκ καθορίζουμε τα επιθυμητά patterns.

Αρχικά στο μπλοκ Pattern {} καθορίζουμε σε ποιο πίνακα κυματομορφών (Waveform table) αναφαίρετε το συγκεκριμένο pattern και τι τιμές παίρνουν τα σήματα του. Στο ανωτέρω παράδειγμά έχουμε το pattern με ονομασία spec_check, που κάνει χρήση του Waveform Table pulsed_oe και εμφανίζει στα σήματα του SignalGoup ALL αρχικά τις παρακάτω τιμές:

```
V { ALL=00DDDDDDDDXXXXXXXX; }
```

Όπου και αν ανατρέξουμε στις δηλώσεις Signals και SignalGroups βλέπουμε ότι το SignalGroup ALL περιλαμβάνει τα σήματα DIR, OE_ και το SignalGroup BUSES που με τη σειρά του περιλαμβάνει τα SignalGroups ABUS και BBUS που με τη σειρά τους περιλαμβάνουν τα σήματα A7, A6, A5, A4, A3, A2, A1, A0, B7, B6, B5, B4, B3, B2, B1, B0,

Στη συνέχεια ακολουθούν τα υπόλοιπα διανύσματα (Vectors) του pattern.

Κάθε αρχείο STIL μπορεί να έχει παραπάνω του ενός patterns. Επιπλέον όπως βλέπουμε και στο ανωτέρω παράδειγμα σε ένα Pattern μπορεί να καθορίζεται η συμπεριφορά για παραπάνω του ενός Waveform Tables.

Στον κώδικα αρχείου STIL που περικλείεται στο μπλοκ PatternBurst {} αναγράφονται τα προς εκτέλεση pattern στο συνολικό αρχείο. Όπως παρατηρεί ο αναγνώστης το μπλοκ του PatternBurst ονοματίζεται καθώς δεν είναι απαραίτητα μοναδικό. Στην ουσία ομαδοποιεί τα υπάρχοντα Patterns που πρόκειται να εκτελεστούν μαζί σε κάποιο σημείο του test. Έτσι μπορούμε να έχουμε συνδυασμούς pattern και PatternBurst μπλοκ κώδικα.

Στον κώδικα αρχείου STIL που περικλείεται στο μπλοκ PatternExec {} τέλος συνδυάζουμε τα Patterns, τα PatternBurst με τις υπόλοιπες αναγκαίες πληροφορίες για εκτέλεση των ελέγχων όπως χρονική συμπεριφορά και τάσεις σημάτων και λειτουργίας.

Ακολουθεί ο πλήρης τρόπος δήλωσης των ορισμάτων των μπλοκ Pattern, PatternBurst και PatternExec.

```
Pattern pat_name {
      W | WaveformTable wft_name    ;
      V | Vector {  vector_statements }
      C | Condition {  vector_statements }
      Call | proc_name {  vector_statements }
      Macro macro_name {  vector_statements }
      Loop integer_count {  vector_statements }
      MatchLoop integer_count {  vector_statements }
      Goto label  ;
```

```
      BreakPoint {  vector_statements }
      IDDQTestPoint     ;
      Stop  ;
      ScanChain chain_name    ;
}



PatternBurst pat_burst_name {
      (Environment/Context statements)
      PatList     {
        Pattern_or_burst_name  {
          (Environment/Context statements)
            }
      }
}

PatternExec optional_patternexec_name {
      Catergory category_name  ;
      Selector selector_name  ;
      Timing timing_name      ;
      PatternBurst patternburst_name  ;
}
```

Το αρχείο STIL που παρουσιάστηκε αποτελεί μια πρώτη γνωριμία με τη σύνταξη και τη δομή για το χρήστη. Με τη μελέτη του ανωτέρω παραδείγματος ο αναγνώστης αρχίζει να αντιλαμβάνεται τα δομικά στοιχεία του προτύπου και τη χρησιμότητά τους. Για την καλύτερη κατανόηση της σύνταξης ενός αρχείου βασισμένου στο πρότυπο ο αναγνώστης μπορεί να μελετήσει το Παράρτημα 1 όπου και γίνεται παρουσίαση του προτύπου σε Backus-Naur μορφή όπως έχει προαναφερθεί. Τέλος στα επόμενα κεφάλαια όπου και θα γίνει παρουσίαση πλήρους και πραγματικού ελέγχου ολοκληρωμένου κυκλώματος, η εικόνα σχεδιασμού, προγραμματισμού και εφαρμογής του προτύπου σε πραγματική χρήση ολοκληρώνεται με λεπτομέρεια.

3. Inovys Personal Ocelot tester

3.1 ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ ΚΑΙ ΠΕΡΙΓΡΑΦΗ ΥΛΙΚΟΥ ΤΟΥ ΕΛΕΓΚΤΗ

Το εργαλείο που θα χρησιμοποιήσουμε σαν ελεγκτή ολοκληρωμένων κυκλωμάτων είναι το μοντέλο PERSONAL OCELOT. Αποτελεί δημιούργημα της εταιρίας INOVYS. Η Inovys Corporation δημιουργήθηκε το 1999 με βασικό σκοπό τη δημιουργία και παροχή εργαλείων ελέγχου ολοκληρωμένων κυκλωμάτων με χρήση τεχνολογιών DFT (Design For Test). Τα προϊόντα της αφορούσαν κυκλώματα συστημάτων ήχου, κινητών τηλεφώνων, βίντεο και λοιπών ηλεκτρονικών εξαρτημάτων. Προήλθε από ανώτερο προσωπικό γνωστών εταιριών του χώρου όπως Credence systems, LTX και Motorola. Το 2002 ανακοινώθηκε από την INOVYS η κυκλοφορία του PERSONAL OCELOT, ενός νέου συστήματος DFT που θα παρείχε δυνατότητες χρήσης τόσο σε συσκευές χαμηλού κόστους για απλές εφαρμογές όσο και σε μικροεπεξεργαστές και σε SoCs (System On Chip). Θεωρείτο εργαλείο χαμηλού κόστους συγκριτικά με τον ανταγωνισμό και με τη χρήση νέων μεθόδων (δυναμικοί πίνακες δεδομένων) μπορούσε να αξιοποιήσει patterns ιδιαίτερης λεπτομέρειας. Το πρόγραμμα λειτουργίας του (STYLUS) έκανε χρήση του προτύπου STIL IEEE 1450 και παρείχε επιπλέον και δυνατότητα σύνδεσης πληροφορίας εσφαλμένων δοκιμών με το υλικό που ήταν συνδεδεμένο στον ελεγκτή. Κυκλοφόρησε με διαφορετικές δυνατότητες ανάλογα με τα εργαλεία που υποστήριζε και τις δυνατότητες συνδεσιμότητας (data pins). Το 2007 η INOVYS πέρασε στα χέρια της VERIGY, εταιρίας αρχικά δημιουργημένης από τη HEWLETT PACKARD και εισηγμένης στο χρηματιστήριο. Για τη τελική της μετάλλαξη και ενώ έφτασε κοντά στο να αγοραστεί από την LTX-CREDENCE τελικά το 2011 απορροφήθηκε από την Ιαπωνική ADVANTEST.

Εικόνα 5. Ελεγκτής Inovys Personal Ocelot

Τη θέση του Personal Ocelot έχουν πάρει νεώτερα μοντέλα πλέον καθώς έχει διακοπεί η κυκλοφορία του, παρόλα αυτά η ADVANTEST συνεχίζει να το υποστηρίζει. Οι δυνατότητες του PERSONAL OCELOT TESTER που έχουμε στην κατοχή μας αναφέρονται εν τάχει παρακάτω.

- 64 Data Pins
- 32M Data Pattern Memory
- 16M Data Capture Memory
- 1 DC Parametric Measurement Unit
- 1 Device Power Supply Expansion
- 1 HPCC (High Performance Clock Channel)

Στο σύνολο του ο ελεγκτής μπορεί να έχει από 64 ως 256 ακροδέκτες γενικού σκοπού. Έχει μπλοκ 64 ακροδεκτών που ονομάζονται TRG (Tester Resource Group) τα οποία ενεργοποιούνται αναλόγως. Η μνήμη του ελεγκτή είναι τεχνολογίας PC133 και χρησιμοποιείται για patterns, instructions και data capture. Η αρχική ρύθμιση είναι χωρητικότητας 256M με δυνατότητα αναβάθμισης σε 512M για τη μνήμη pattern και instruction. Κατά τη διάρκεια εκτέλεσης των ελέγχων η μνήμη του συστήματος αλλάζει δυναμικά και διαμοιράζεται ανάλογα με τα μεγέθη των vectors που πρέπει να εφαρμοστούν, αν είναι δηλαδή σειριακά η μεγάλου μεγέθους (serial or broad-size vectors). Ο παρακάτω πίνακας μας δίνει τη χωρητικότητα σε σειριακά διανύσματα ανάλογα με τους ακροδέκτες που χρησιμοποιούνται για την ανάγνωση των διανυσμάτων σε κάθε TRG. (Mv = 1.048.576 vectors)

	256MB DIMM's	**512MB DIMM's**
All pins (parallel vectors)	16Mv	32Mv
33-64 scan pins in any TRG	16Mv	32Mv
17-32 scan pins in any TRG	32Mv	64Mv
9-16 scan pins in any TRG	64Mv	128Mv
5-8 scan pins in any TRG	128Mv	256Mv
3-4 scan pins in any TRG	256Mv	512Mv
2 scan pins in any TRG	512Mv	1.024Mv
1 scan pin in any TRG	1.024Mv	2.048Mv

Πίνακας 9. Χωρητικότητα μνήμης Pattern

Data Capture μνήμη υπάρχει ανεξάρτητη σε κάθε TRG. Έχει μήκος 255 words και Option μπορεί να αυξηθεί προσθέτοντας έξτρα DIMMs σε κάθε TRG. Κατά την εκτέλεση των patterns αυξάνεται ένας 32bit counter σε κάθε κύκλο ρολογιού. Εδώ ανάλογα με την επιλογή μπορεί να γίνεται καταγραφή είτε σε κάθε αποτυχία ενός ακροδέκτη κρατώντας και τον κύκλο στον οποίον συμβαίνει η αποτυχία (default συμπεριφορά) είτε να γίνεται καταγραφή των ακροδεκτών σε συγκεκριμένα διανύσματα.

Σχετικά με τα χρονικά δεδομένα του ελεγκτή αυτά καταγράφονται στον παρακάτω πίνακα

Data Period Range (parallel and serial)	20ns – 635ns
Clock Period range (Clock Format)	10ns – 635ns
Data/Clock Period Resolution	5.0ns
Max Data Pin TG Programmable Range	Lesser of 4 cycles or 615ns
Data Pin TG Resolution	625ps

Data Pin Edge Placement Accuracy	± 1.8ns
Minimum Pulse Width	5ns

Πίνακας 10. Χρονικές προδιαγραφές tester

Σχετικά με τα δεδομένα τάσης οδήγησης και σύγκρισης του ελεγκτή παρατίθεται ο παρακάτω πίνακας.

	Minimum	**Maximum**	**Resolution**	**Accuracy**
Drive high level	650mV	3.5V	1.3mV	± 20mV
Drive low level	0V	N/A	N/A	N/A
Comparator Threshold	200mV	2.0V	1.3mV	± 50mV

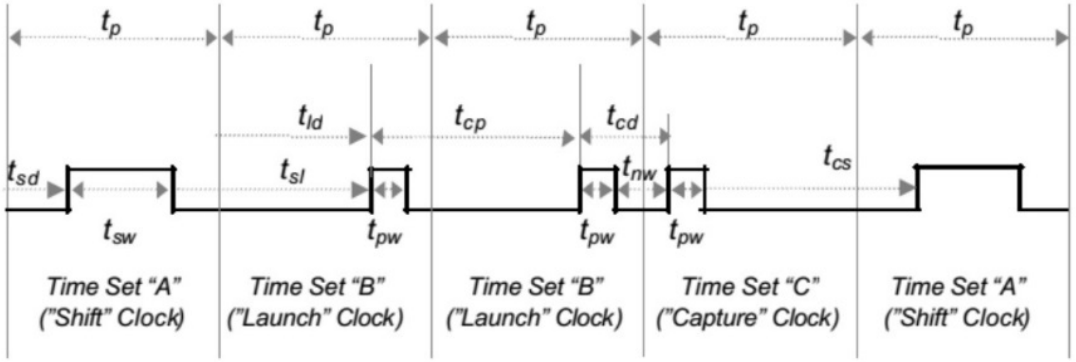
Πίνακας 11. Προδιαγραφές τάσης tester

Οι προδιαγραφές της PMU (DC Parametric measurement unit) του ελεγκτή παρουσιάζονται παρακάτω.

Voltage Force Range	-2.0V to +5.0V
Voltage Force Resolution	0.65mV
Voltage Force Accuracy	±6.0mV
Voltage Measure Range	-2.0V to +5.0V
Voltage Measure Resolution	125mV
Voltage Measure Accuracy	±8.0mV

Πίνακας 12. Προδιαγραφές PMU tester

Ο ελεγκτής παρέχει και 1 κανάλι ρολογιού υψηλής απόδοσης. Δύο σήματα ρολογιού πολυπλέκονται και παρέχουν σήμα σε 2 ακροδέκτες που μπορούν να χρησιμοποιηθούν είτε σαν ακροδέκτες δεδομένων είτε σαν ακροδέκτες ρολογιού. Υπάρχει ένα ζεύγος τέτοιων σε κάθε TRG. (Ακροδέκτες 0/1, 64/65, 128/129, 192/193). Στον ένα ακροδέκτη το σήμα λειτουργεί σαν frequency synthesizer δημιουργώντας ένα ελεύθερο σήμα ρολογιού που δεν είναι συναφές ούτε με τη συχνότητα ούτε με τη φάση των patterns που εκτελούνται. Το σήμα αυτό ενεργοποιείται από ένα pattern. Στον δεύτερο ακροδέκτη οδηγούμε σήμα από τρεις πιθανές προελεύσεις. Ένα ρολόι ολίσθησης (shift clock generator) που παρέχει ρολόι χαμηλής ανάλυσης (625ps), ένα ρολόι εκτέλεσης (launch clock) και ένα ρολόι σύλληψης (capture clock) τα οποία και παρέχουν σήμα ρολογιού υψηλής ανάλυσης (15ps) για ακρίβεια σε εφαρμογές ελέγχου AC.



Εικόνα 6. Παράμετροι HPCC

	Min	**Max**	**Resolution**
Vector Period (Tp)	20ns	635ns	5ns
Shift clock delay (Tsd)	0ns	635ns	625ps
Shift clock width (Tsw)	3.7ns	635ns	625ps
Shift trail to launch/capture clock time (Tsl)	3.0ns	n/a	15ps
Launch/Capture clock delay (Tld)	0ns	635ns	15ps
Launch to capture clock delay (Tcd)	2.5ns	635ns	15ps
Launch/Capture clock width (Tpw)	1.25ns	635ns	15ps
Launch trail to capture clock delay (Tnw)	1.25ns	635ns	15ps
Launch to launch clock delay (Tcp)	20ns	n/a	5ns
Capture trail to shift clock delay (Tcs)	3.0ns	n/a	n/s
Launch to capture accuracy (Tcd)		±350ps	

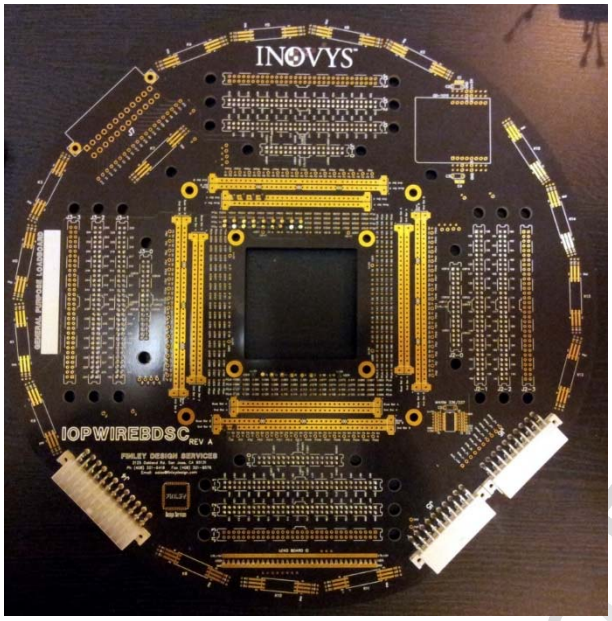
Πίνακας 13. Χρονικές προδιαγραφές HPCC

	Specifications
Minimum drive low level (Vil)	0V
Maximum drive low level (Vil)	3.0V
Minimum drive high level (Vih)	0.25V
Maximum drive high level (Vih)	3.6V
Maximum drive voltage swing (Vih – Vil)	3.6V
Minimum drive voltage swing (Vih – Vil)	0.25V
Driver level resolution	1.2mV
Driver static level accuracy	±25mV

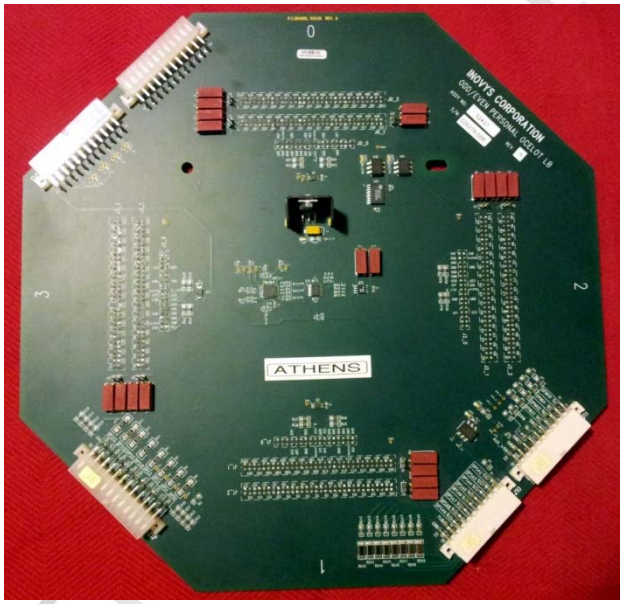
Πίνακας 14. Προδιαγραφές οδήγησης HPCC

Ο ελεγκτής συνδέεται μέσω δύο USB καλωδίων (1.1 και 2.0) με τον υπολογιστή στον οποίο είναι εγκατεστημένο το Stylus. Στο πίσω μέρος του υπάρχει η σύνδεση με την τροφοδοσία (220V) και 2 επιπλέον βύσματα σύνδεσης. Οι διαστάσεις του είναι 58cm X 50 cm X 18cm και

ζυγίζει 18,2 Kg. Για τη σύνδεση οποιουδήποτε υλικού στον ελεγκτή μας παρέχονται κενές πλακέτες (Load Board, LB) οι οποίες και κατασκευάζονται κατά περίπτωση με βάση το DUT (device under test). Στη διάθεσή μας βρίσκεται μια κενή πλακέτα και μία της οποίας οι ακροδέκτες είναι συνδεδεμένοι με δεδομένο τρόπο για πειραματισμό (odd / even Load Board).



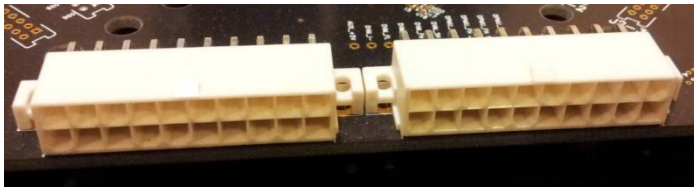
Εικόνα 7. IOPWIREBDSC Loadboard



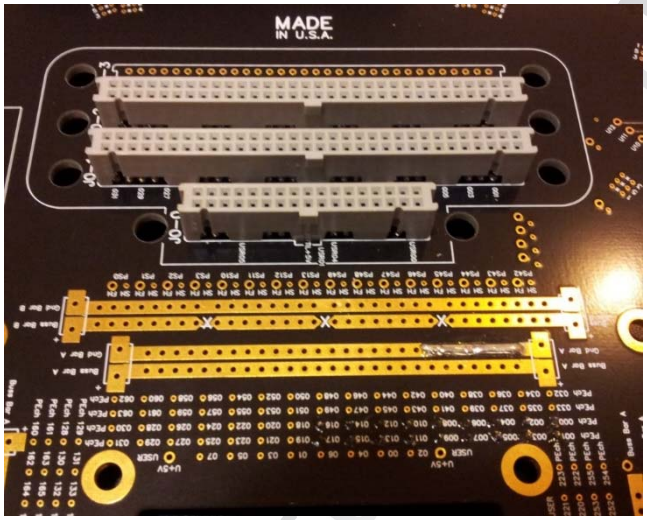
Εικόνα 8. ODD/EVEN Loadboard

Και στις δύο πλακέτες υπάρχουν οι ακροδέκτες σύνδεσης καλωδίων (J4, J5, J6) για τα καλώδια του ελεγκτή. Επιπλέον η βασική σύνδεση γίνεται με τους ακροδέκτες (pin header) που βρίσκονται στη βάση του ελεγκτή και στην κάτω επιφάνεια της εκάστοτε πλακέτας (Εικόνα 5). Οι ακροδέκτες καλωδίων και οι ακροδέκτες βάσης απεικονίζονται στις παρακάτω εικόνες. Είναι όμοιοι και στις δύο πλακέτες.

Εικόνα 9. J4 Loadboard Connector

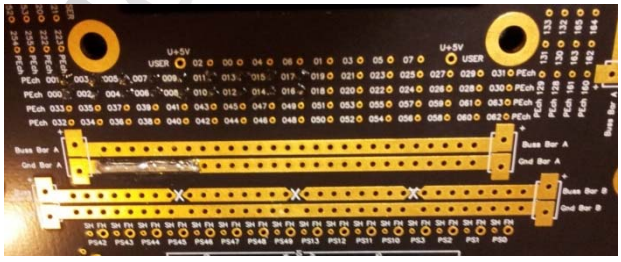


Εικόνα 10. J5, J6 Loadboard Connectors



Εικόνα 11. Ακροδέκτες (Pin Header) βάσης πλακέτας LB (Loadboard)

Στην ODD / EVEN LB υπάρχει φυσική σύνδεση μεταξύ των περιττών και άρτιων ακροδεκτών του ελεγκτή για την εκτέλεση των ανάλογων παραδειγμάτων που παρέχονται με την εγκατάσταση του προγράμματος Stylus. Επιπλέον υπάρχουν ολοκληρωμένα κυκλώματα στην πλακέτα (ULN2803 / SOIC18 και LT1963 / TO-220) για εφαρμογές παραδειγμάτων. Η πλακέτα IOWIREBDSC της Finley Design, είναι κενή. Παρέχει Through Hole VIAs στα οποία τερματίζει η σύνδεση με τους ακροδέκτες του ελεγκτή.

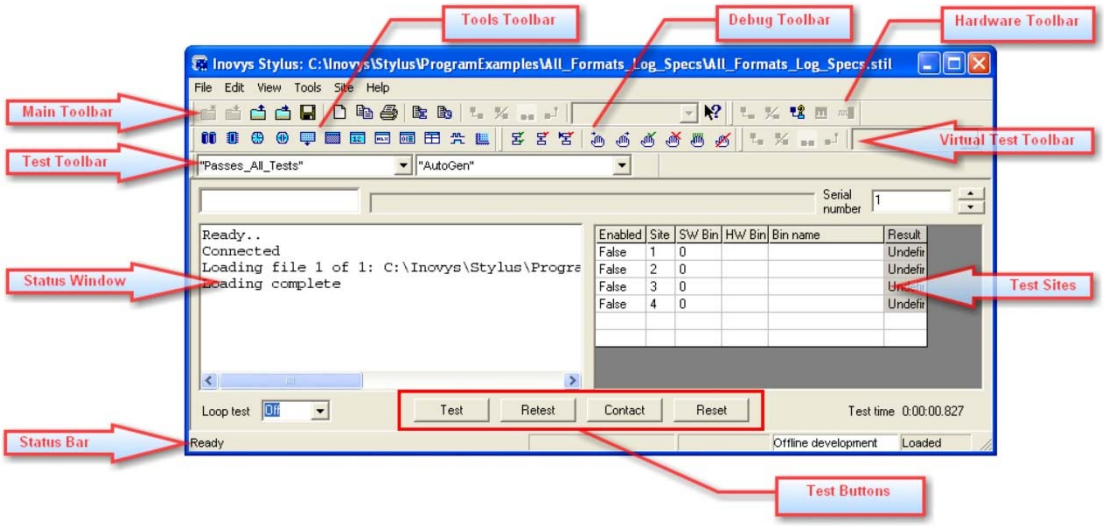


Εικόνα 12. IOWIREBDSC through hole VIAs

Το γεγονός αυτό μας οδήγησε στο σχεδιασμό και τη δημιουργία της πλακέτας Inovys Ocelot IO Board, η οποία και παρέχει σειρά ακροδεκτών (pin header) για τις εισόδους και εξόδους του ελεγκτή. Οι ακροδέκτες που παρέχει στο χρήστη είναι οι συνήθεις ακροδέκτες με απόσταση μεταξύ τους (pitch) 2.54mm για την ευκολότερη και απρόσκοπτη φυσική σύνδεση των κυκλωμάτων υπό δοκιμή στον ελεγκτή. Κατά τη διαδικασία παραγωγής σε μια εταιρία δημιουργούνται συγκεκριμένες πλακέτες σύνδεσης (Loadboards) για την εκάστοτε δοκιμή ολοκληρωμένου κυκλώματος ανάλογα τη συσκευασία (package) του προς παραγωγή κυκλώματος. Η πλακέτα IOWIREBDSC έχει σχεδιασμένα πάνω της και σχέδια ολοκληρωμένων κυκλωμάτων (footprints). Στην περίπτωση μας όπου και τα κυκλώματα υλοποιούνται στην πλακέτα Xilinx Spartan 3E δεν μας ενδιαφέρει κάτι τέτοιο επομένως κάνουμε χρήση των ακροδεκτών που παρέχονται από την σχεδιασμένη πλακέτα πλακέτας Inovys Ocelot IO Board.

Επιπλέον το περιβάλλον του ελεγκτή παρέχει εικονική σύνδεση με πλακέτες (virtual server για επιλογή board) για εκπαιδευτικούς και πειραματικούς λόγους καθώς η ύπαρξη πλακέτας ανά περίπτωση είναι δύσκολη και όχι πάντα εφικτή. Για να ξεπεραστεί αυτό το εμπόδιο έγινε κατασκευή πλακέτας εισόδων εξόδων με την οποία ο χρήστης μπορεί να συνδέσει το επιθυμητό προς έλεγχο υλικό με τα IO Pins του ελεγκτή. Επιπλέον παρέχονται και ακροδέκτες γείωσης στη συγκεκριμένη πλακέτα. Η πλακέτα παρουσιάζεται στη συνέχεια του παρόντος.

3.2 ΠΕΡΙΒΑΛΛΟΝ STYLUS



Εικόνα 13. Περιβάλλον Stylus

Στην εικόνα 6 παρουσιάζεται το περιβάλλον Stylus με το οποίο και χειρίζεται ο χρήστης τον ελεγκτή ολοκληρωμένων κυκλωμάτων Inovys Personal Ocelot. Στο συγκεκριμένο σημείο κρίνεται σκόπιμη μια βηματική παρουσίαση των βασικών μενού που θα χρειαστεί ο χρήστης για να ολοκληρώσει ένα νέο project. Προτείνεται φυσικά μια πιο ενδελεχής μελέτη του προγράμματος για καλύτερη κατανόηση των λειτουργιών και των δυνατοτήτων του.

Αρχικά ξεκινάμε το πρόγραμμα Stylus επιλέγοντας από το συνδεδεμένο με τον ελεγκτή υπολογιστή

Start > Programs > Verigy > Stylus 3.1.6

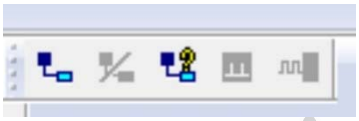
Με το ξεκίνημα του προγράμματος τα περισσότερα μενού είναι απενεργοποιημένα (γκρι). Πρώτο μέλημα είναι η σύνδεση του προγράμματος με τον ελεγκτή. Υπάρχουν τέσσερεις πιθανές καταστάσεις.

- Not connected
- Offline
- Hardware Connection
- Virtual Test

Στην πρώτη περίπτωση το Stylus δεν είναι συνδεδεμένο. Στη δεύτερη το Stylus είναι ενεργό και έτοιμο για σύνδεση. Χρησιμοποιείται για ανάπτυξη προγραμμάτων (προεργασία). Στην τρίτη κατάσταση το Stylus έχει συνδεθεί με τον ελεγκτή Inovys Personal Ocelot είτε τοπικά με φυσική σύνδεση είτε απομακρυσμένα μέσω δικτύου. Τέλος παρέχεται εικονικός προσομοιωτής του ελεγκτή σαν εργαλείο εκπαίδευσης και ανάπτυξης καθώς μπορεί να μην είναι πάντα εφικτός ο πειραματισμός με τον ελεγκτή. Εδώ ο χρήστης επιλέγει τη σύνδεση πατώντας το παρακάτω εικονίδιο

Εικόνα 14. Μενού σύνδεσης Stylus – Connect Menu

Με τη εκτέλεση της λειτουργίας το πρόγραμμα αντιδρά ενεργοποιώντας τα μενού σύνδεσης με τον ελεγκτή ή τον εικονικό προσομοιωτή του ελεγκτή.



Εικόνα 15. Μενού σύνδεσης με τον ελεγκτή Stylus – Connect to hardware Menu

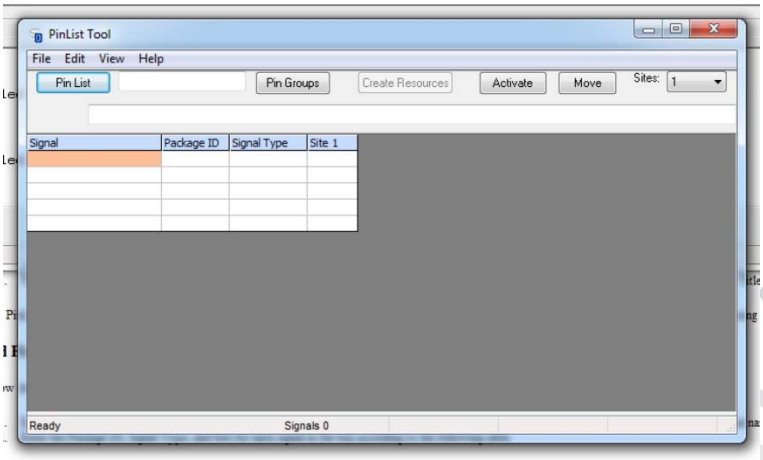
Εικόνα 16. Μενού σύνδεσης με τον εικονικό προσομοιωτή Stylus – Virtual test connect Menu

Για την ανάπτυξη ενός νέου προγράμματος δεν χρειάζεται σύνδεση σε πρώτη φάση ούτε με τον ελεγκτή ούτε με τον εικονικό προσομοιωτή. Έτσι για την δημιουργία μιας νέας εφαρμογής επιλέγουμε

File > New

Τα υπόλοιπα εργαλεία είναι πια ενεργοποιημένα και μπορούμε να προχωρήσουμε στην υλοποίηση της νέας εφαρμογής. Στο παρόν αναφέρονται τα εργαλεία ξεκινώντας από το βασικό μενού. Μετά την εξοικείωση με το περιβάλλον ο χρήστης για ευκολία μπορεί να κάνει χρήση και των συντομεύσεων στη γραμμή εργαλείων “Tools Toolbar”. Αρχικά επιλέγουμε το εργαλείο “Pin List” όπου και δημιουργούμε τα αναγκαία σήματα με βάση το πρότυπο STIL (Signals) συνδέοντας τα με τους υπάρχοντες ακροδέκτες του προς ελέγχου ολοκληρωμένου κυκλώματος.

Tools > Pin List



Εικόνα 17. Pin List Tool Stylus

Πατώντας στο κουμπί “Pin List” ενεργοποιείται η επιλογή “Object Selection” όπου και δημιουργούμε και ονοματίζουμε μια λίστα σημάτων. Το αναδυόμενο μενού “Object Selection”θα το συναντήσουμε σε αρκετά μενού του Stylus. Έπειτα στη στήλη “Signal” δίνουμε όνομα ακροδέκτη, στη στήλη “Package ID” γράφουμε το ανάλογο ακροδέκτη πάνω στο ολοκληρωμένο κύκλωμα, στη στήλη “Signal Type” επιλέγουμε από την αναδυόμενη λίστα τον τύπο του σήματος και στη στήλη “Site 1” επιλέγουμε τον ακροδέκτη του ελεγκτή στον οποίο συνδέεται το εκάστοτε σήμα. Το περιβάλλον Stylus μας δίνει τη δυνατότητα να εκτελούμε ελέγχους με την ίδια εφαρμογή σε παραπάνω του ενός ολοκληρωμένα κυκλώματα ίδιου τύπου (Multisite Testing). Σε τέτοια περίπτωση επιλέγουμε στο μενού “Pin List” από το drop down menu “Sites” τον αριθμό των συνδεδεμένων ολοκληρωμένων κυκλωμάτων και αντίστοιχα συμπληρώνουμε και τις επιπλέον στήλες τις απαραίτητες πληροφορίες για κάθε σήμα.

Στη συνέχεια επιλέγοντας το κουμπί “Pin Groups” ομαδοποιούμε τα σήματα με βάση το πρότυπο STIL δημιουργώντας ομάδες σημάτων (Signal Groups). Τα περισσότερα εργαλεία του περιβάλλοντος Stylus εμφανίζονται σε “Spreadsheet Mode”. Υπάρχει πάντα και η επιλογή “ASCII Mode” οπότε και αντιμετωπίζεις τα μενού σαν κείμενο γράφοντας τις εκάστοτε πληροφορίες με βάση τη σύνταξη του προτύπου STIL. Σε κάθε αλλαγή χρειάζεται ο χρήστης να επιλέγει το κουμπί “Parse” για να προωθούνται οι αλλαγές από το κείμενο.

Κάνοντας μια διακοπή προτείνεται η εφαρμογή να σώζεται κατά διαστήματα έτσι σε αυτό το σημείο επιλέγουμε

File > Save

Και παράλληλα ονοματίζουμε το αρχείο της εφαρμογής μας. Για να ξεκινήσουμε νέα εφαρμογή για τον ελεγκτή επιλέγουμε

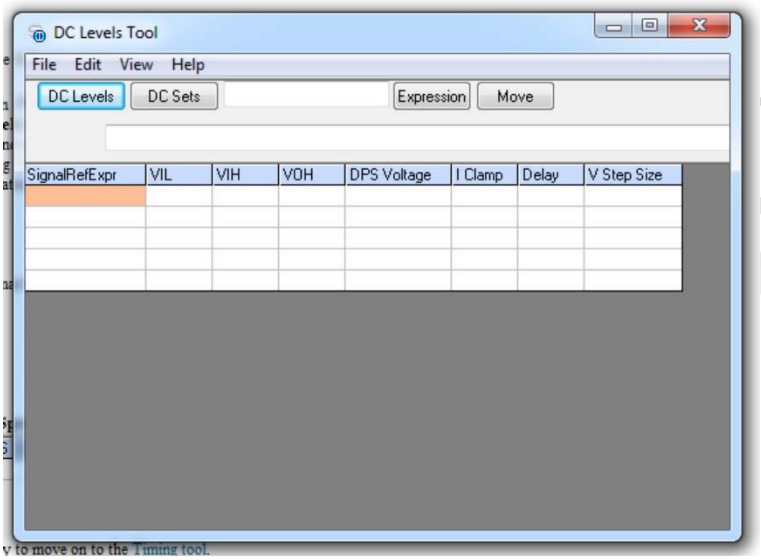
File > Unload

Όπως και για να φορτώσουμε μια υπάρχουσα εφαρμογή για επεξεργασία επιλέγουμε όπως γίνεται εύκολα αντιληπτό

File > Load

Επόμενο βήμα είναι η επιλογή του εργαλείου για τον καθορισμό του επιπέδου τάσεων στο κύκλωμα μας. Έτσι επιλέγουμε το εργαλείο “DC Levels”.

Tools > DC Levels

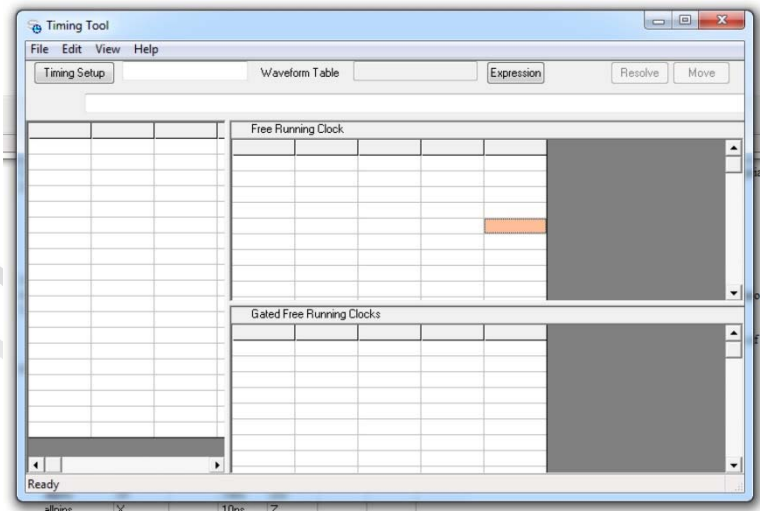


Εικόνα 18. DC Levels Tool Stylus

Πατώντας του κουμπί “DC Levels” ενεργοποιείται η επιλογή “Object Selection” όπου και καθορίζουμε τις προδιαγραφές επιπέδου τάσεων στα σήματα της εφαρμογής. Με δεξί κλικ στη στήλη “SignalRefExpr” επιλέγουμε από τα δημιουργημένα σήματα ή τις ομάδες σημάτων (Signals / Signal Groups) και έπειτα ορίσουμε τα επίπεδα τάσης που αντιστοιχούν σε κάθε ένα από αυτά. Βασικές ρυθμίσεις εδώ είναι οι “VIH” (Voltage Input High) και “VOH“ (Voltage Output High) συμπληρώνονται με βάση τα όρια που έχουν αναφερθεί στην προηγούμενη ενότητα.

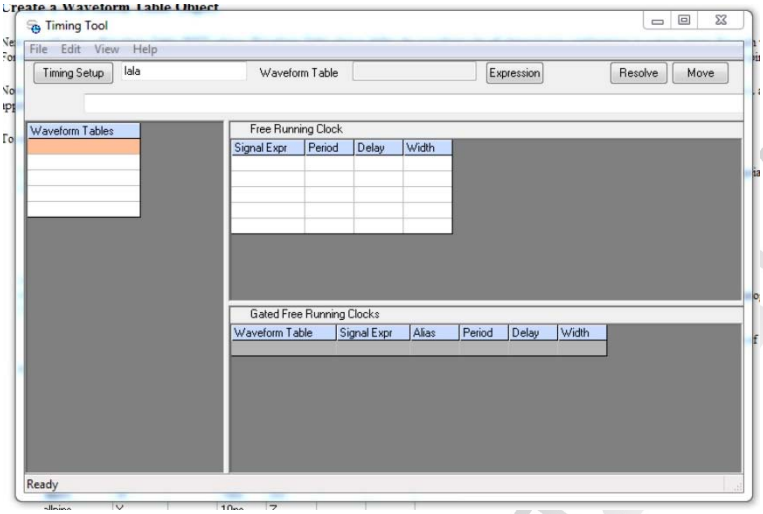
Επόμενο βήμα είναι η επιλογή του εργαλείου για τον καθορισμό του χρονισμού της εφαρμογής ελέγχου στο κύκλωμα μας. Έτσι επιλέγουμε το εργαλείο “Timing”

Tools > Timing



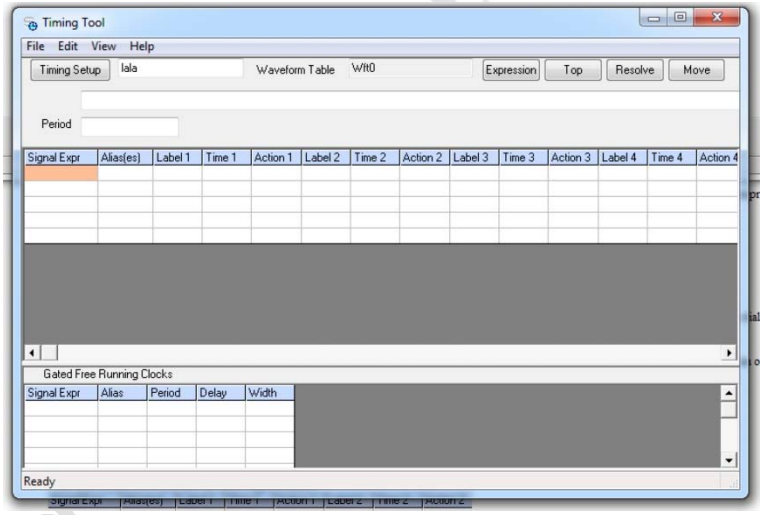
Εικόνα 19. Timing Tool 1 Stylus

Πατώντας το κουμπί “Timing Setup” ενεργοποιείται η επιλογή “Object Selection” όπου και ονοματίζουμε τη ρύθμιση των χρονικών δεδομένων της εφαρμογής. Η εμφάνιση του εργαλείου τροποποιείται στην παρακάτω μορφή



Εικόνα 20. Timing Tool 2 Stylus

Επιλέγοντας το πρώτο κελί στη στήλη “Waveform Tables” δημιουργούμε με βάση το πρότυπο STIL τους πίνακες κυματομορφών. Γράφουμε το όνομα του πίνακα και κάνουμε διπλό κλικ πάνω του ώστε το εργαλείου να φτάσει στην τελική του μορφή ως παρακάτω



Εικόνα 21. Timing Tool 3 Stylus

Εδώ αφού καθορίσουμε την περίοδο των σημάτων συμπληρώνοντας το ανάλογο πεδίο, στη συνέχεια επιλέγουμε στη στήλη “Signal Expr” το εκάστοτε σήμα ή ομάδα σημάτων. Στη στήλη “Alias(es)” δίνουμε όνομα στην προς περιγραφή συμπεριφορά ενός σήματος. Με χρήση αυτού του ονόματος αργότερα στο εργαλείο “Pattern” θα καθορίσουμε την επιθυμητή συμπεριφορά κάθε σήματος με βάση ότι έχουμε συμπληρώσει εδώ. Για κάθε σήμα στη συνέχεια συμπληρώνουμε τις στήλες “Time #” και “Action #” όπου και καθορίσουμε το χρόνο (στήλη “Time”) όταν και θα γίνει η εκάστοτε ενέργεια (στήλη “Action”). Η στήλη “Action” συμπληρώνεται από τις πιθανές τιμές καταστάσεων σημάτων του προτύπου STIL όπως αναφέρονται στον πίνακα 3.

Στη συνέχεια ο χρήστης δημιουργεί τα προς εκτέλεση Patterns και ρυθμίζει τις υπόλοιπες λεπτομέρειες για την εκτέλεσή τους. Το κομμάτι της δημιουργίας του Pattern μπορεί να γίνει με δύο τρόπους. Ο πρώτος είναι το εργαλείο “Pattern”.

Tools > Pattern

Καθώς το αρχείο Pattern (*.spat) είναι αρχείο κειμένου στην ουσία μπορεί να δημιουργηθεί και με έναν οποιοδήποτε κειμενογράφο (text editor) και να αποθηκευτεί με την κατάληξη “.spat”. Στο παρόν αναφέρουμε παράδειγμα δημιουργίας Pattern αρχείου με έναν κειμενογράφο. Το κείμενο του Pattern έχει την παρακάτω μορφή οπότε ανοίγουμε ένα text editor και κάνουμε επικόλληση του κειμένου ως έχει

```
      Pattern Pattern1 {
W Wft1;
V { allpins = 00000; }
V { allpins = 11111; }
V { allpins = 00000; }
V { allpins = 0L000; }
V { allpins = 1H111; }
V { allpins = 0L000; }
V { allpins = 1H111; }
V { allpins = 0L000; }
V { allpins = 1H111; }
V { allpins = 0L000; }
V { allpins = 1H111; }
V { allpins = 0L000; }
V { allpins = 1H111; }
V { allpins = 00000; }
V { allpins = 00000; }
}
```

Αποθηκεύουμε το αρχείο με το επιθυμητό όνομα για παράδειγμα “Pattern1.spat”.

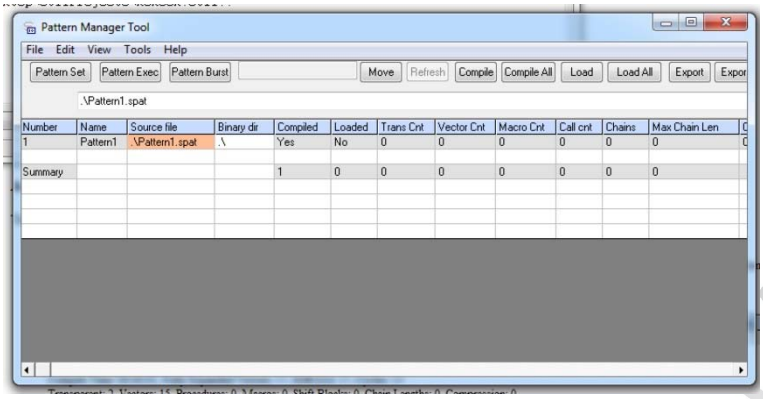
Αρχικά το αρχείο ξεκινά με τη λέξη Pattern ακολουθούμενη από το όνομα του συγκεκριμένου pattern. Στη συνέχεια ακολουθεί η λέξη κλειδί “W” που υποδηλώνει τον πίνακα κυματομορφών στον οποίο αναφέρεται το Pattern και το όνομα του πίνακα κυματομορφών (Waveform Table). Στη συνέχεια κάθε γραμμή αποτελεί και ένα διάνυσμα (Vector) το οποίο εκτελείται και καθορίζει τις τιμές των σημάτων (Signals). Έτσι η γραμμή

```
V { allpins = 1H111; }
```

Αναφέρεται στο διάνυσμα “V” που περιλαμβάνει το γκρουπ σημάτων (Signal Group) “allpins” πέντε σήματα (5) ως έχει οριστεί με τη χρήση του εργαλείου “Pin List” νωρίτερα. Οι τιμές που παίρνουν τα πέντε αυτά σήματα είναι {1, H, 1, 1, 1}. Κάθε διάνυσμα εκτελείται με χρονική διαφορά ίση με την περίοδο ως αυτή έχει οριστεί με το εργαλείο “Timing” νωρίτερα.

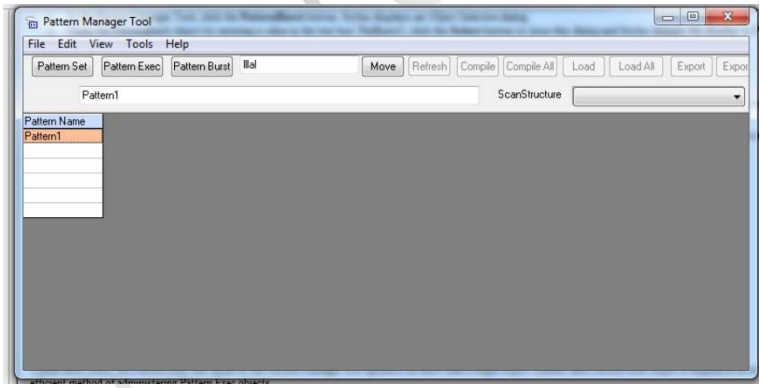
Για να εκμεταλλευτούμε το δημιουργημένο Pattern κάνουμε χρήση του εργαλείου “Pattern Manager”

Tools > Pattern Manager



Εικόνα 22. Pattern Manager Tool (Pattern Set) Stylus

Εδώ κάνοντας διπλό κλικ στη στήλη “Source file” επιλέγουμε από το αναδυόμενο παράθυρο το spat αρχείο (Pattern1.spat). Μπορούμε να επιλέξουμε και παραπάνω του ενός spat αρχεία για εκτέλεση. Τέλος επιλέγουμε το κουμπί “Compile All” για να μεταγλωττιστούν τα δημιουργημένα Pattern αρχεία. Σε περίπτωση που στο βασικό παράθυρο “Status Window” του Stylus εμφανιστεί μήνυμα λάθους προχωράμε στις απαραίτητες διορθώσεις. Κατά το άνοιγμα του εργαλείου “Pattern Manager” αυτό βρίσκεται σε κατάσταση “Pattern Set” ως αναγράφεται σε ένα από τα βασικά κουμπιά του. Εδώ καταγράφουμε και μεταγλωττίζουμε τα patterns που θα χρησιμοποιηθούν. Στη συνέχεια πατώντας το κουμπί “Pattern Burst” ενεργοποιείται η επιλογή “Object Selection” όπου δίνουμε ένα όνομα και καθορίζουμε ποια από τα test vectors που έχουμε δημιουργήσει θα εφαρμοστούν στην εκτέλεση των patterns. Σε κάθε “Pattern Burst” μπορούν τα patterns να επαναλαμβάνονται δημιουργώντας υπο-ρουτίνες. Μετά το πάτημα του κουμπιού “Pattern Burst” κάνουμε δεξί κλικ στο πρώτο κελί της λίστας “Pattern Name” και επιλέγουμε από το αναδυόμενο παράθυρο το επιθυμητό από τα Patterns που έχουμε δημιουργήσει στην αρχική οθόνη (Pattern Set) του εργαλείου “Pattern Manager”.



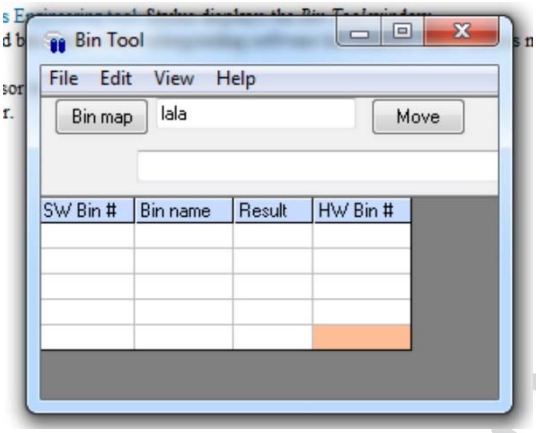
Εικόνα 23. Pattern Manager Tool (Pattern Burst) Stylus

Αφού ολοκληρώσουμε και τις εγγραφές του “Pattern Burst” προχωράμε στην επιλογή του κουμπιού “Pattern Exec”. Εδώ συμπληρώνουμε τις απαιτούμενες πληροφορίες για την ορθή εκτέλεση κάθε pattern. Το κομμάτι “Pattern Exec” του εργαλείου “Pattern Manager” μπορεί να επεξεργάζεται και να καθορίζει παραπάνω του ενός αντικείμενα. Έτσι εδώ κάθε γραμμή – εγγραφή απεικονίζει ένα αντικείμενο τύπου “Pattern Exec”. Στο κελί της λίστας “Exec” ονοματίζουμε την κάθε εγγραφή. Στη συνέχεια με δεξί κλικ στη στήλη “Burst” επιλέγουμε από το αναδυόμενο παράθυρο το αντικείμενο τύπου “Burst” που δημιουργήσαμε νωρίτερα και αναφέρεται στη συγκεκριμένη εγγραφή. Ομοίως με δεξί κλικ στη στήλη “Timing” επιλέγουμε από το αναδυόμενο παράθυρο το αντικείμενο τύπου “Timing” που δημιουργήσαμε νωρίτερα με το εργαλείο “Timing”. Τέλος με δεξί κλικ στη στήλη “DCLevels” επιλέγουμε το αντικείμενο που έχουμε

δημιουργήσει νωρίτερα με το εργαλείο “DC Levels” και καθορίζει τις τάσεις των σημάτων. Στην ουσία πλέον έχουμε καθορίσει πλήρως τα προς εκτέλεση patterns και τα στοιχεία εκτέλεσης τους.

Επόμενο βήμα είναι η χρήση του εργαλείου “Bin” με το οποίο καθορίζουμε την καταγραφή αποτελεσμάτων είτε θετικών (Pass) είτε αρνητικών (Fail).

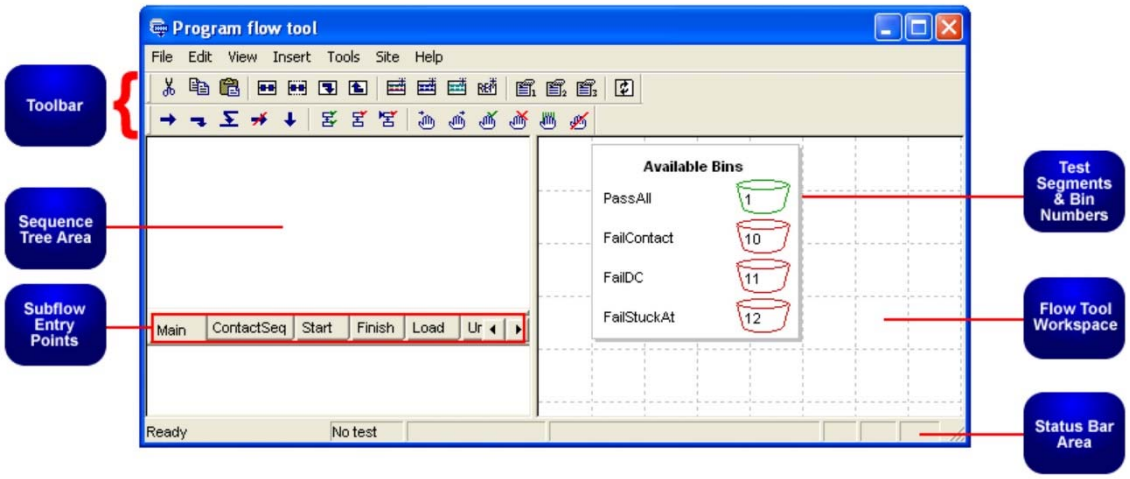
Tools > Bin



Εικόνα 24. Bin Tool Stylus

Το εργαλείο “Bin” χρησιμοποιείται σε συνδυασμό με το εργαλείο “Program Flow” όπως θα δούμε παρακάτω. Το εργαλείο “Program Flow” καθορίζει τη σειρά εκτέλεσης των αντικειμένων και του τρόπου συνολικής λειτουργίας του ελέγχου καθώς παρέχει και επιπλέον πληροφορία πέραν των προς εκτέλεση patterns.

Tools > Program Flow



Εικόνα 25. Program Flow Tool Stylus

Όπως βλέπουμε στην εικόνα του εργαλείου υπάρχουν υπο-σελίδες στην πορεία εκτέλεσης ενός ελέγχου με τον ελεγκτή Inovys Personal Ocelot. Οι τρεις βασικές που θα μας απασχολήσουν είναι οι { Start – Main – Finish }.

Ξεκινώντας από την καρτέλα “Start” κάνουμε δεξί κλικ στην επιφάνεια “Flow Tool Workspace” και επιλέγουμε “New Test Segment”. Ένα μπλοκ εμφανίζεται με διακεκομμένες

γραμμές περιθωρίου και κάνοντας κλικ στο χώρο το τοποθετούμε οριστικά. Με διπλό κλικ πάνω στο μπλοκ ενεργοποιείται το εργαλείο “Segment”. Σκοπός του μπλοκ αυτού στην καρτέλα “Start” είναι η σύνδεση των ηλεκτρικών ρελέ (Pin Electronic Relays) στην αρχή του ελέγχου. Έτσι στο εργαλείο “Segment” πατώντας το κουμπί “Params” ενεργοποιείται η επιλογή “Object Selection” όπου και ονοματίζουμε το συγκεκριμένο μπλοκ για παράδειγμα “ConnectDUT”. Τώρα με δεξί κλικ στη στήλη “Method” επιλέγουμε μια από τις υπάρχουσες μεθόδους που υποστηρίζει ο ελεγκτής. Στην συγκεκριμένη φάση επιλέγουμε τη μέθοδο “Connect”. Επιλέγοντας στις υπόλοιπες στήλες το επιθυμητό γκρουπ σημάτων “SignalRefExpr” με δεξί κλικ, τα “PE” από τη στήλη “Resources” ομοίως με δεξί κλικ, την ενέργεια “Apply” στη στήλη “Action” και το χρόνο που θα γίνει αυτό στη στήλη “Delay” που θα είναι τώρα (0s). Ομοίως στην καρτέλα “Finish”, ακολουθούμε την ίδια διαδικασία επιλέγοντας όμως “New Unconditional Segment”, κάνουμε διπλό κλικ στον μπλοκ και πατώντας το κουμπί “Params” ενεργοποιείται η επιλογή “Object Selection” όπου και ονοματίζουμε το συγκεκριμένο μπλοκ για παράδειγμα “DisconnectDUT”. Επιπλέον επιλέγουμε πάλι τη μέθοδο “Connect” αλλά κάνουμε στη στήλη “Resources” την επιλογή “All” και στη στήλη “Action” την επιλογή “Remove” αποσυνδέοντας έτσι όλα τα συνδεδεμένα τμήματα του tester.

Το βασικό κομμάτι του ελέγχου παραμένει στην καρτέλα “Main”. Εκεί πατώντας το κουμπί “Params” ενεργοποιείται η επιλογή “Object Selection” όπου και ονοματίζουμε το συγκεκριμένο μπλοκ για παράδειγμα “TestDUT”. Εδώ στη στήλη “Method” επιλέγουμε τη μέθοδο “Pattern” καθώς εδώ θα φορτώσουμε το Pattern που δημιουργήσαμε προς εκτέλεση. Στη συνέχεια στις στήλες που εμφανίζονται μετά την επιλογή της μεθόδου “Pattern” επιλέγουμε με δεξί κλικ τις απαιτούμενες και αναμενόμενες ρυθμίσεις για το κάθε pattern με βάση όσα έχουμε καθορίσει νωρίτερα με το εργαλείο “Pattern Manager”. Επιλέγουμε δηλαδή τα ανάλογα “SignalRefExpr”, “PatternExec” , “PatternBurst”, Timing”, “DCLevels”. Επιλέγοντας τέλος με δεξί κλικ και “add action” ή με διπλό κλικ πάνω στο κόκκινο και πράσινο βέλος των μπλοκ, καθορίζουμε με ποιο “Bin” συνδέεται το αποτέλεσμα εκτέλεσης κάθε μπλοκ εφόσον αυτή είναι επιτυχής (Pass) η αποτυχημένη (Fail).

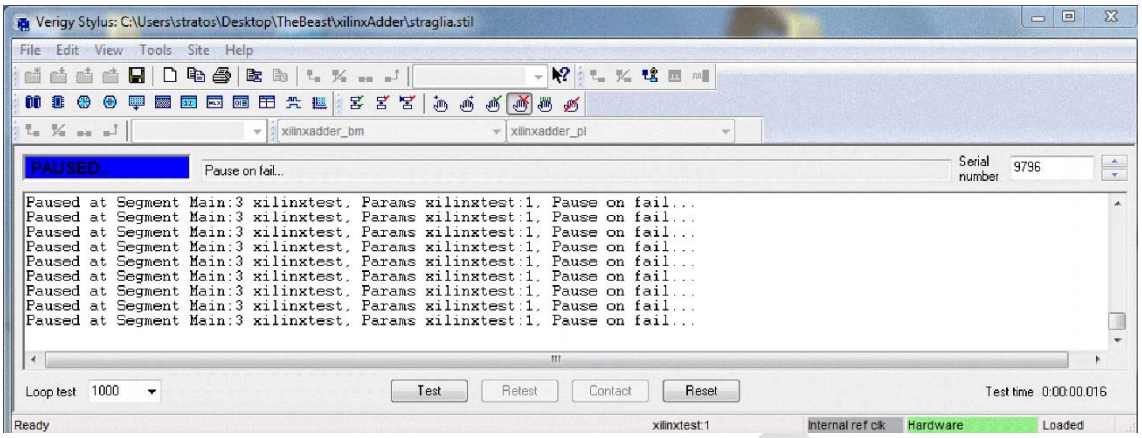
Μετά και από την ολοκλήρωση των παραπάνω βημάτων και αφού σώσουμε το ολοκληρωμένο ποια αρχείο ελέγχου μπορούμε πατώντας το κουμπί “Test” στο περιβάλλον Stylus και να εκτελέσουμε το δημιουργημένο αρχείο ελέγχου παρατηρώντας τα αποτελέσματά του. Εάν κατευθυνθούμε στο φάκελο που έχουμε σώσει το δημιουργημένο αρχείο και το ανοίξουμε με ένα κειμενογράφο μπορούμε να αναγνώσουμε τις οντότητες που έχουμε δημιουργήσει με τη χρήση των εργαλείων του περιβάλλοντος Stylus καθώς είναι σε πλήρως αναγνώσιμη μορφή.

Το περιβάλλον Stylus μας δίνει τη δυνατότητα αποσφαλμάτωσης των προγραμμάτων ελέγχου που εκτελούμε. Αυτό γίνεται με συγκεκριμένη προεπιλογή σε ανάλογο μενού με το οποίο μπορούμε να παύσουμε την εκτέλεση ενός προγράμματος ελέγχου σε δεδομένες στιγμές όπως

- Στην αρχή εκτέλεσης κάθε δομικού στοιχείου (segment) του εργαλείου “Program Flow Tool”
- Στην τέλος εκτέλεσης κάθε δομικού στοιχείου (segment) του εργαλείου “Program Flow Tool”
- Σε κάθε έλεγχο
- Σε κάθε αποτυχημένο έλεγχο (Fail)
- Σε κάθε ενέργεια

Εικόνα 26. Pause execution Menu Stylus

Έτσι σε περίπτωση σφάλματος μπορούμε να ελέγξουμε την κατάσταση κάθε σήματος και να εντοπίσουμε το σημείο του σφάλματος. Σε περίπτωση επιλογής παύσης της εκτέλεσης σε αποτυχημένο έλεγχο (Pause on Fail) το περιβάλλον Stylus μας οδηγεί στην παρακάτω κατάσταση



Εικόνα 27. Pause execution Menu – Result Stylus

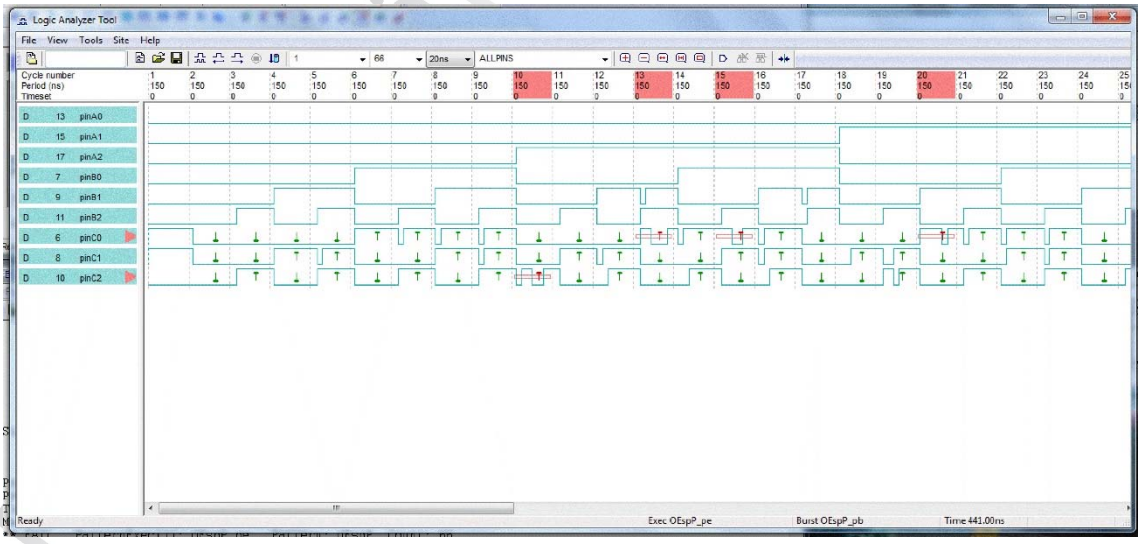
Εδώ ο χρήστης καλεί το εργαλείο “Logic Analyzer” του ελεγκτή

Tools > Logic Analyzer

Με την επιλογή και εκκίνηση του εργαλείου επιλέγουμε

File > Execute Waveform Capture

Έτσι το εργαλείο αναλύει τις κυματομορφές των σημάτων και μας απεικονίζει τις πραγματικές τιμές και τις αναμενόμενες τιμές χρονικά. Παράλληλα εμφανίζει κατάλληλη σήμανση σε σωστές και λανθασμένες τιμές όπως φαίνεται παρακάτω

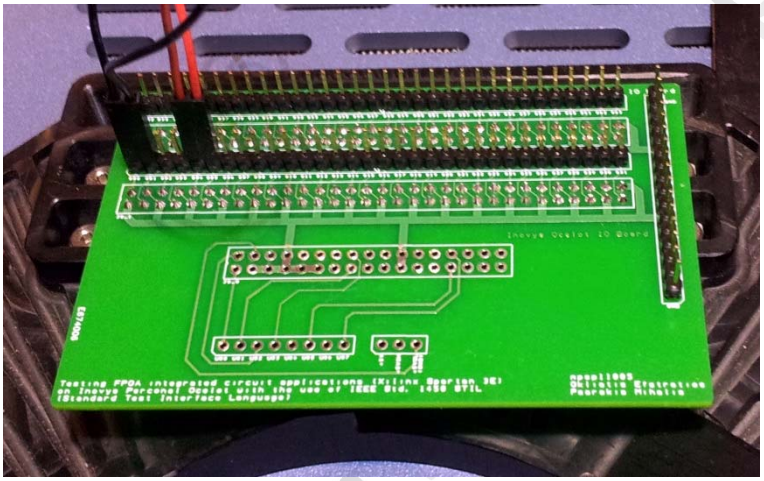


Εικόνα 28. Logic Analyzer Tool Stylus

Για τον τερματισμό της εφαρμογής επιλέγουμε “Unload application”, “Disconnect from hardware” και τέλος “Disconnect”. Μετά από αυτό κλείνουμε το περιβάλλον Stylus και αποσυνδέουμε από την τροφοδοσία τον ελεγκτή Inovys Personal Ocelot”.

3.3 ΠΑΡΑΔΕΙΓΜΑ ΠΛΗΡΟΥΣ ΑΡΧΕΙΟΥ ΕΛΕΓΧΟΥ

Στο ακόλουθο παράδειγμα δημιουργούμε ένα αρχείο ελέγχου στο οποίο δοκιμάζουμε την επιτυχή εκτέλεση ενός «ολοκληρωμένου κυκλώματος» στο οποίο δύο ακροδέκτες λειτουργούν σαν έξοδοι και δύο ακροδέκτες λειτουργούν σαν είσοδοι. Οι ακροδέκτες είναι γεφυρωμένοι σε ζευγάρια έτσι η κάθε έξοδος γεφυρώνεται με την είσοδο. Πρακτικά προσομοιώνουμε ένα από τα παραδείγματα χρήσης της μια από τις παρεχόμενες πλακέτες με τον ελεγκτή Inovys Personal Ocelot, την ODD/EVEN PERSONAL OCELOT LB (Load board). Η διαφορά έγγυται στο γεγονός ότι χάριν απλότητας υλοποιούμε το πρόγραμμα ελέγχου με 2 μόνο ζεύγη ακροδεκτών καθώς σκοπός του παραδείγματος είναι η πρώτη γνωριμία με ένα κατανοητό πρόγραμμα ελέγχου βασισμένο στο πρότυπο IEEE Std. 1450 STIL υλοποιημένο στον ελεγκτή Inovys Personal Ocelot. Επιπλέον όπως θα δούμε γίνεται χρήση του τυπωμένου κυκλώματος (PCB) που κατασκευάστηκε για τις ανάγκες του παρόντος οδηγού χρήσης, INOVYS OCELOT IO Board. Το σχέδιο της πλακέτας INOVYS OCELOT IO Board καθώς και επιπλέον λεπτομέρειες αναφέρονται στο Παράρτημα 5 της παρούσης.



Εικόνα 29. Ocelot IO Board

3.3.1 Αρχείο STIL

```
STIL   1.00;

Signals {
    pin000 InOut;
    pin001 InOut;
    pin002 InOut;
    pin003 InOut;
}
PinList OEspPL {
    Sites 1;
    pin000 (    0);
    pin001 (    1);
    pin002 (    4);
    pin003 (    5);
```

```
}
SignalGroups {
    ALLPINS = 'pin000+pin001+pin002+pin003';
}
Timing OEspTM {
    WaveformTable Wft0 {
          Period '100ns';
          Waveforms {
                ALLPINS { 01 {  '0s' D/U; }}
                ALLPINS { LH {  '0s' Z;  '50ns' L/H; }}
                ALLPINS { X {  '0s' Z; }}
          }
    }
}
DCLevels OEspDCL {
    ALLPINS{
    VIH 2.5V;
    VOH 1.0V;
    }
}
PatternRef OEspP {
    SourceFile ".\OEspP.spat";
    BinaryDir ".\";
    Timing OEspTM;
}
PatternBurst OEspPB {
    PatList {
          OEspP;
    }
}
PatternExec OEspPE {
    Timing OEspTM;
    PatternBurst OEspPB;
    DCLevels OEspDCL;
}
Bin "OEspPass"{
    ProgramResult = "Pass";
    BinNumber =    1;
}
Bin "OEspFail"{
    ProgramResult = "Fail";
    BinNumber =    2;
}
TestParams ConnDut{
```

```
      Method Connect {"SignalRefExpr" 'ALLPINS';"Resource" PE;"Action"
Apply;"Delay" '0s';}
   }
   TestParams DiscDut{
      Method Connect {"SignalRefExpr" 'ALLPINS';"Resource" ALL;"Action"
Remove;"Delay" '0s';}
   }
   TestParams OEspTest{
      Method    Pattern    {"SignalRefExpr"    'ALLPINS';"PatternExec"
OEspPE;"BurstMode" true;}
   }
   ProgSeq "Main"{
      Segment 3{
            Type TEST;
            TestParams OEspTest;
            Start true;
            Title "OEspTest";
            Position 120,200;
            Action  { 'Result==Pass'; Bin=1; }
            Action  { 'Result==Fail'; Bin=2; }
      }
   }
   ProgSeq "ContactSeq"{
   }
   ProgSeq "Start"{
      Segment 1{
            Type UNCONDITIONAL;
            TestParams ConnDut;
            Start true;
            Title "ConnDut";
            Position 120,200;
      }
   }
   ProgSeq "Finish"{
      Segment 2{
            Type UNCONDITIONAL;
            TestParams DiscDut;
            Start true;
            Title "DiscDut";
            Position 120,200;
      }
   }
   ProgSeq "Load"{
   }
   ProgSeq "Unload"{
```

```
  }
  ProgSeq "API"{
  }
  ProgramObjects {
      ActivePinList OEspPL;
}
```

3.3.2 Αρχείο SPAT

```
  Pattern OEspP {
  W Wft0;
  V{ALLPINS=XXXX;}
  V{ALLPINS=XXXX;}
  V{ALLPINS=0L0L;}
  V{ALLPINS=1H1H;}
  V{ALLPINS=0L0L;}
  V{ALLPINS=1H1H;}
  V{ALLPINS=0L0L;}
  V{ALLPINS=1H1H;}
  V{ALLPINS=0L0L;}
  V{ALLPINS=1H1H;}
  V{ALLPINS=0L0L;}
  V{ALLPINS=1H1H;}
  V{ALLPINS=0L0L;}
  V{ALLPINS=1H1H;}
  V{ALLPINS=0L0L;}
  V{ALLPINS=1H1H;}
  V{ALLPINS=XXXX;}
  V{ALLPINS=XXXX;}
  V{ALLPINS=L0L0;}
  V{ALLPINS=H1H1;}
  V{ALLPINS=L0L0;}
  V{ALLPINS=H1H1;}
  V{ALLPINS=L0L0;}
  V{ALLPINS=H1H1;}
  V{ALLPINS=L0L0;}
  V{ALLPINS=H1H1;}
  V{ALLPINS=L0L0;}
  V{ALLPINS=H1H1;}
  V{ALLPINS=L0L0;}
  V{ALLPINS=H1H1;}
  V{ALLPINS=L0L0;}
  V{ALLPINS=H1H1;}
```

```
   V{ALLPINS=L0L0;}
   V{ALLPINS=H1H1;}
   V{ALLPINS=XXXX;}
   V{ALLPINS=XXXX;}
}
```

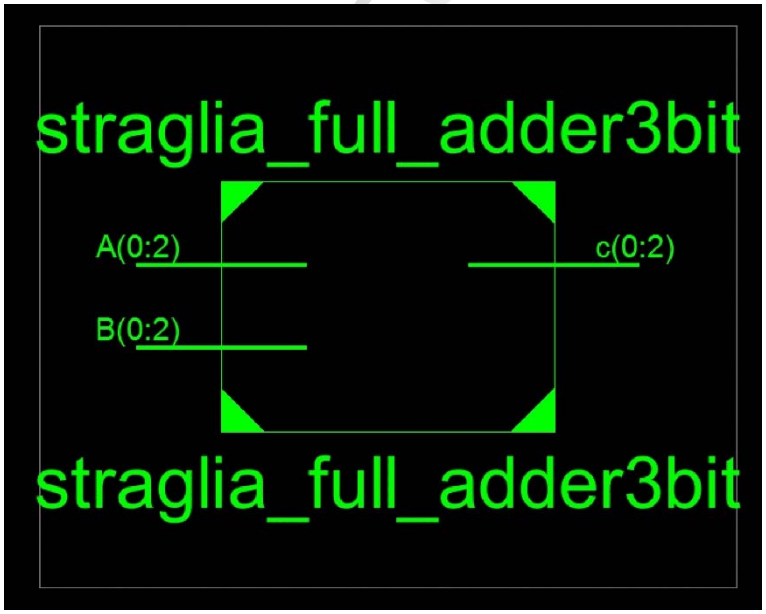
4. Εφαρμογές ελέγχου ολοκληρωμένων κυκλωμάτων υλοποιημένων με χρήση γλώσσας περιγραφής VHDL με το XILINX ISE σε πλακέτα Spartan 3E, σε περιβάλλον Stylus με χρήση του προτύπου IEEE Std. 1450 STIL και του εργαλείου Inovys Personal Ocelot

Στο παρόν κεφάλαιο παρουσιάζεται η εκτέλεση του ελέγχου 2 κυκλωμάτων υλοποιημένων με χρήση γλώσσας περιγραφής VHDL με το XILINX ISE σε πλακέτα Spartan 3E σε περιβάλλον Stylus με χρήση του προτύπου IEEE Std. 1450 STIL και του εργαλείου Inovys Personal Ocelot. Αρχικά υλοποιούμε ένα κύκλωμα ενός αθροιστή τριών bit (3-bit adder). Έπειτα συνδέουμε το κύκλωμα στον ελεγκτή και με χρήση ενός pattern δημιουργημένου με το εργαλείο προσομοίωσης του κυκλώματος, εκτελούμε τους απαραίτητους ελέγχους ορθής λειτουργίας του υπό δοκιμή κυκλώματος. Στη συνέχεια προχωράμε στην ίδια διαδικασία αλλάζοντας αυτή τη φορά το κύκλωμα σε ένα πολλαπλασιαστή τριών bit (3-bit multiplier). Οι διαφορές και οι λεπτομέρειες της κάθε περίπτωσης αναφέρονται παρακάτω.

4.1 ΚΥΚΛΩΜΑ ΑΘΡΟΙΣΤΗ 3-BIT

4.1.1 Δημιουργία αρχείου ελέγχου

Για την εκτέλεση του ελέγχου ενός κυκλώματος πλήρους αθροιστή τριών bit (3 bit adder) αρχικά υλοποιούμε το κύκλωμα. Αυτό γίνεται με χρήση του Xilinx ISE (Project Navigator) με αποτέλεσμα την παρακάτω σχηματική απεικόνισή του. Η χρήση του Xilinx ISE και η υλοποίηση του κυκλώματος παρουσιάζεται σε επόμενο κεφάλαιο. Στο παρόν κεφάλαιο επικεντρώνουμε την προσοχή στο περιβάλλον Stylus και τις απαραίτητες ενέργειες σε αυτό, για τη δημιουργία και εκτέλεση του ελέγχου. Αρχικά γίνεται εκκίνηση του περιβάλλοντος Stylus. Εν συνεχεία συνδεόμαστε στον ελεγκτή με χρήση του μενού σύνδεσης και προχωράμε σε σύνδεση και με το υλικό. Εφόσον η διαδικασία ολοκληρωθεί απρόσκοπτα ξεκινά η δημιουργία του αρχείου ελέγχου για το συγκεκριμένο κύκλωμα.



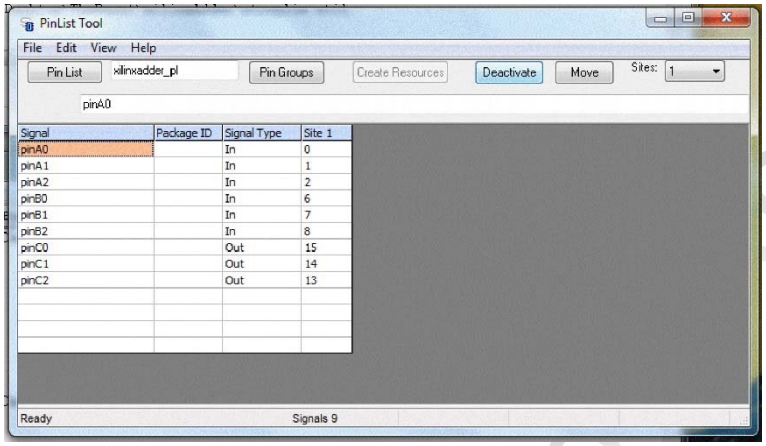
Εικόνα 30. 3 Bit adder schematic

Όπως έχει αναφερθεί νωρίτερα δημιουργούμε ένα νέο αρχείο

File > New

Με χρήση του εργαλείου “Pin List” δημιουργούμε τα απαραίτητα σήματα εισόδου και εξόδου που αντιστοιχούν σε κάθε bit τελεστέων και αποτελέσματος. Έτσι έχουμε

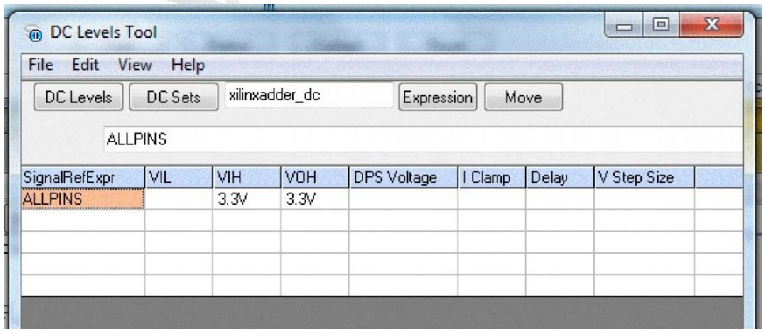
Tools > Pin List



Εικόνα 31. 3 Bit adder Pin List Tool

Εδώ δημιουργούμε ένα αντικείμενο τύπου “Pin List” με το οποίο και ονομάζουμε τα σήματα των δύο τελεστέων pinA0, pinA1, pinA2, pinB0, pinB1, pinB2 και τα δηλώνουμε σαν σήματα εισόδου (In). Ο καθορισμός εισόδου η εξόδου αναφέρεται στο κύκλωμα όχι στον ελεγκτή. Στη στήλη “Package ID” θα μπορούσαμε να βάλουμε το όνομα του ακροδέκτη του ολοκληρωμένου μας. Τέλος στη στήλη “Site 1” δηλώνουμε σε ποιο ακροδέκτη από τα 64 διαθέσιμα του ελεγκτή συνδέεται το κάθε σήμα. Εν συνεχεία δηλώνουμε τα σήματα του αποτελέσματος pinC0, pinC1, pinC2 ως σήματα εξόδου (Out) ακολουθώντας την ίδια διαδικασία. Επιπλέον δηλώνουμε και γκρουπ σημάτων για την ομαδοποίηση και περιγραφή της συμπεριφοράς τους αργότερα στο pattern ελέγχου. Στο παρόν παράδειγμα δημιουργούμε μόνο ένα “Signal Group” το οποίο ονομάζεται “ALLPINS” και περιλαμβάνει όλα τα υπάρχοντα σήματα. Εδώ είναι ένα καλό σημείο να σώσουμε και το αρχείο μας δίνοντας του και το κατάλληλο, με βάση το κύκλωμα προς έλεγχο, όνομα. Στη συνέχει επιλέγουμε το εργαλείο “DC Levels”

Tools > DC Levels

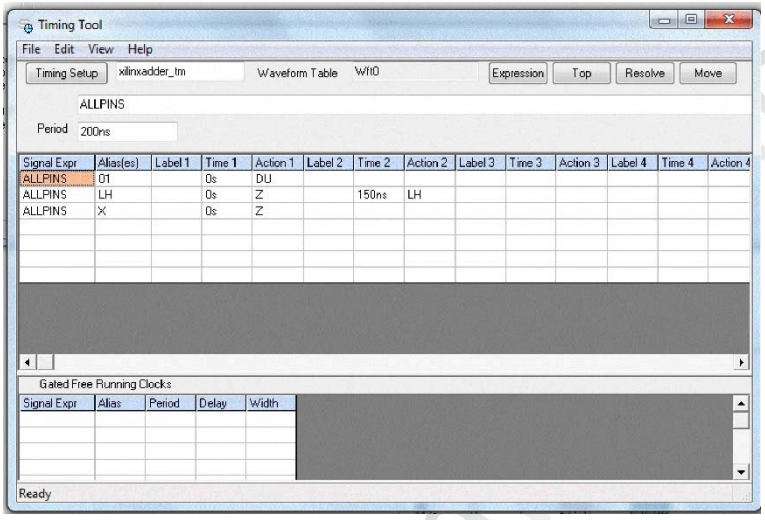


Εικόνα 32. 3 Bit adder DC Levels Tool

Εδώ με τη δημιουργία ενός αντικειμένου τύπου “Timing” καθορίζουμε την τάση που θα εφαρμόζεται στους ακροδέκτες - σήματα του ολοκληρωμένου. Προσοχή χρειάζεται καθώς κάθε περίπτωση λάθους μπορεί να αποβεί μοιραία για το ολοκληρωμένο. (αναφέρουμε το ολοκληρωμένο καθώς ο ελεγκτής έχει κατά βάση μεγαλύτερη δυνατότητα οδήγησης τροφοδοσίας και αντοχής σε τάσεις). Εδώ η επιλογή γίνεται πάντα με βάση τα τεχνικά χαρακτηριστικά του ολοκληρωμένου κυκλώματος. Στο παρόν επιλέγουμε τάση 3.3V τόσο για σήματα εισόδου όσο και για σήματα εξόδου καθώς αυτή είναι η τάση που εφαρμόζεται στους ακροδέκτες της πλακέτας Xilinx Spartan 3E, στην οποία και έχουμε υλοποιήσει το κύκλωμά μας.

Επόμενο βήμα είναι η χρήση του εργαλείου “Timing” με το οποίο και καθορίζουμε τη συμπεριφορά των σημάτων σε σχέση με το χρόνο. Έτσι έχουμε

Tools > Timing



Εικόνα 33. 3 Bit adder Timing Tool

Μία βασική πληροφορία που δηλώνεται εδώ είναι η περίοδος του κυκλώματος μας. Αφορά τη χρονική διαφορά μεταξύ της εφαρμογής των διανυσμάτων (Vectors) στα pattern ελέγχου που θα εκτελεστούν αργότερα. Επιπλέον όπως γίνεται κατανοητό από την παραπάνω εικόνα δηλώνουμε το ψευδώνυμο κάθε σήματος (Alias) και την ενέργεια (Action #) που θα εκτελεστεί σε κάθε χρονική στιγμή (Time #). Στο συγκεκριμένο παράδειγμα με βάση τον Πίνακα 3 της παρούσης και τις δηλώσεις σημάτων που έγιναν με το εργαλείο “Pin List” έχουμε τα παρακάτω.

- Για τα σήματα του Signal group “ALLPINS” όταν στο pattern εμφανίζεται το Alias “0” το σήμα εμφανίζει συμπεριφορά οδήγησης χαμηλής κατάστασης [Action 1 = D] (Drive state , assert Low Value) μετά από 0ns [Time 1 = 0ns]
- Για τα σήματα του Signal group “ALLPINS” όταν στο pattern εμφανίζεται το Alias “1” το σήμα εμφανίζει συμπεριφορά οδήγησης υψηλής κατάστασης [Action 1 = U] (Drive state , assert High Value) μετά από 0ns [Time 1 = 0ns]
- Για τα σήματα του Signal group “ALLPINS” όταν στο pattern εμφανίζεται το Alias “L” το σήμα αποσυνδέεται [Action 1 = Z] (Drive State, remove or disconnect) μετά από 0ns [Time 1 = 0ns] και εμφανίζει συμπεριφορά σύγκρισης χαμηλής κατάστασης [Action 2 = L] (Compare state , detect Low Value) μετά από 150ns [Time 2 = 150ns]
- Για τα σήματα του Signal group “ALLPINS” όταν στο pattern εμφανίζεται το Alias “H” το σήμα αποσυνδέεται [Action 1 = Z] (Drive State, remove or disconnect) μετά από 0ns [Time 1 = 0ns] και εμφανίζει συμπεριφορά σύγκρισης υψηλής

κατάστασης [Action 2 = H] (Compare state , detect High Value) μετά από 150ns [Time 2 = 150ns]

- Για τα σήματα του Signal group “ALLPINS” όταν στο pattern εμφανίζεται το Alias “X” το σήμα αποσυνδέεται [Action 1 = Z] (Drive State, remove or disconnect) μετά από 0ns [Time 1 = 0ns]

Με αυτό τον τρόπο καθορίζουμε τη συμπεριφορά κάθε σήματος για την εκτέλεση ενός pattern δημιουργώντας τα απαραίτητα σήματα (Signals) και τις απαραίτητες ομάδες σημάτων (Signal Groups).

Το επόμενο εργαλείο σε χρήση είναι το “Pattern” με το οποίο και δημιουργούμε pattern για την εκτέλεση των ελέγχων.

Tools > Pattern

Όπως έχει όμως αναφερθεί και νωρίτερα τα αρχεία των pattern είναι πλήρως αναγνώσιμα από τον άνθρωπο σαν κείμενα. Έτσι μπορούμε να τα δημιουργήσουμε με χρήση ενός απλού κειμενογράφου. Προχωρώντας ένα βήμα παραπέρα εδώ κάνουμε χρήση της αυτόματης δημιουργίας του αρχείου pattern μέσω του εργαλείου προσομοίωσης του κυκλώματος κατά τη δημιουργία του με το περιβάλλον Xilinx ISE Project Navigator. Στο αρχείο “testbench” που δημιουργούμε κατά την ανάπτυξη ενός κυκλώματος με χρήση VHDL στο περιβάλλον Xilinx ISE προσθέτουμε τις αναγκαίες εντολές έτσι ώστε δοκιμάζοντας κάθε πιθανό συνδυασμό εισόδου στο κύκλωμα να καταγράφουμε τόσο τις τιμές εισόδου όσο και τις τιμές εξόδου. Αυτές συγκεντρώνονται σε ένα έγγραφο κειμένου (text) το οποίο και αποθηκεύουμε αλλάζοντας την κατάληξη του μόνο από “ .txt ” σε “ .spat ”. Ο τρόπος παρουσιάζεται με λεπτομέρεια στο επόμενο κεφάλαιο όπου και περιγράφεται ο τρόπος ανάπτυξης του κυκλώματος στο περιβάλλον Xilinx ISE. Με αυτόν τον τρόπο μπορούμε να δημιουργήσουμε το αρχείο pattern αυτόματα με την ελάχιστη δυνατή ανάγκη επεξεργασίας.

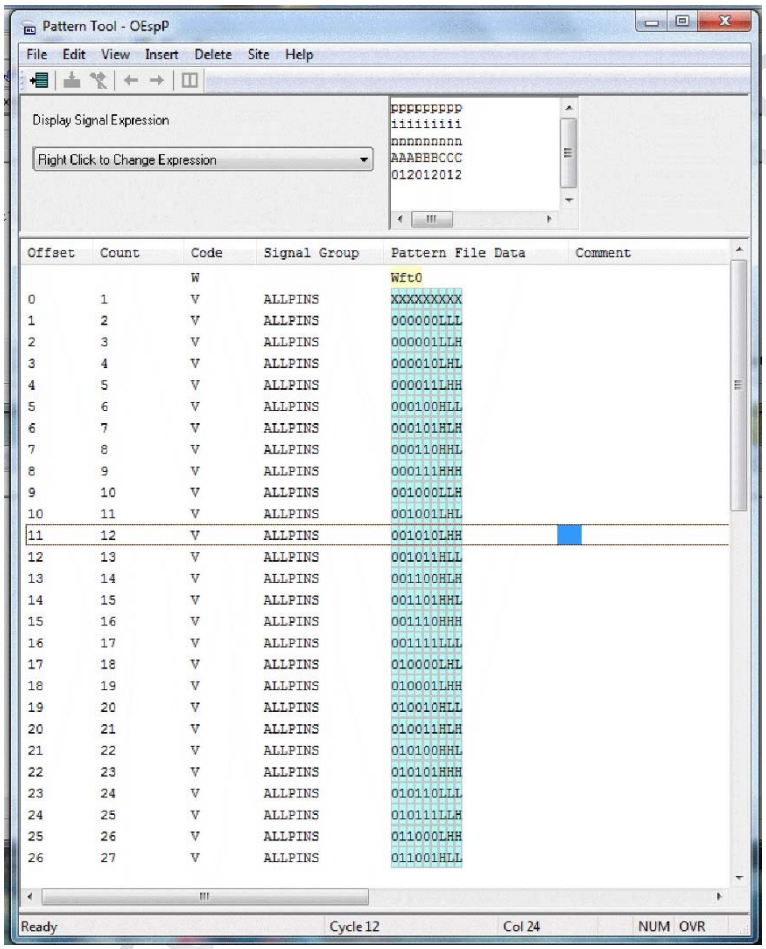
Έτσι το αρχείο pattern μετά τη δημιουργία του από το εργαλείο προσομοίωσης του κυκλώματος και μετά την τυχόν αναγκαία επεξεργασία έχει την παρακάτω μορφή

```
Pattern OEspP{
W Wft0;
V{ALLPINS=XXXXXXXXX;}
V{ALLPINS=000000LLL;}
V{ALLPINS=000001LLH;}
V{ALLPINS=000010LHL;}
V{ALLPINS=000011LHH;}
V{ALLPINS=000100HLL;}
V{ALLPINS=000101HLH;}
V{ALLPINS=000110HHL;}
V{ALLPINS=000111HHH;}
V{ALLPINS=001000LLH;}
V{ALLPINS=001001LHL;}
V{ALLPINS=001010LHH;}
V{ALLPINS=001011HLL;}
V{ALLPINS=001100HLH;}
V{ALLPINS=001101HHL;}
V{ALLPINS=001110HHH;}
V{ALLPINS=001111LLL;}
```

```
V{ALLPINS=010000LHL;}
V{ALLPINS=010001LHH;}
V{ALLPINS=010010HLL;}
V{ALLPINS=010011HLH;}
V{ALLPINS=010100HHL;}
V{ALLPINS=010101HHH;}
V{ALLPINS=010110LLL;}
V{ALLPINS=010111LLH;}
V{ALLPINS=011000LHH;}
V{ALLPINS=011001HLL;}
V{ALLPINS=011010HLH;}
V{ALLPINS=011011HHL;}
V{ALLPINS=011100HHH;}
V{ALLPINS=011101LLL;}
V{ALLPINS=011110LLH;}
V{ALLPINS=011111LHL;}
V{ALLPINS=100000HLL;}
V{ALLPINS=100001HLH;}
V{ALLPINS=100010HHL;}
V{ALLPINS=100011HHH;}
V{ALLPINS=100100LLL;}
V{ALLPINS=100101LLH;}
V{ALLPINS=100110LHL;}
V{ALLPINS=100111LHH;}
V{ALLPINS=101000HLH;}
V{ALLPINS=101001HHL;}
V{ALLPINS=101010HHH;}
V{ALLPINS=101011LLL;}
V{ALLPINS=101100LLH;}
V{ALLPINS=101101LHL;}
V{ALLPINS=101110LHH;}
V{ALLPINS=101111HLL;}
V{ALLPINS=110000HHL;}
V{ALLPINS=110001HHH;}
V{ALLPINS=110010LLL;}
V{ALLPINS=110011LLH;}
V{ALLPINS=110100LHL;}
V{ALLPINS=110101LHH;}
V{ALLPINS=110110HLL;}
V{ALLPINS=110111HLH;}
V{ALLPINS=111000HHH;}
V{ALLPINS=111001LLL;}
V{ALLPINS=111010LLH;}
V{ALLPINS=111011LHL;}
```

```
V{ALLPINS=111100LHH;}
V{ALLPINS=111101HLL;}
V{ALLPINS=111110HLH;}
V{ALLPINS=111111HHL;}
V{ALLPINS=XXXXXXXXX;}
}
```

Όπως βλέπουμε κάθε γραμμή του αρχείου pattern αντιστοιχεί σε ένα διάνυσμα (Vector) σημάτων το οποίο αναφέρεται στην ομάδα σημάτων (Signal group) “ALLPINS”, και καθορίζει τα 9 σήματα που έχουμε ορίσει, ένα για κάθε bit. Κάθε Vector εκτελείται σε χρόνο ίσο με την περίοδο “Period” που ισούται στο παρόν με 200ns.

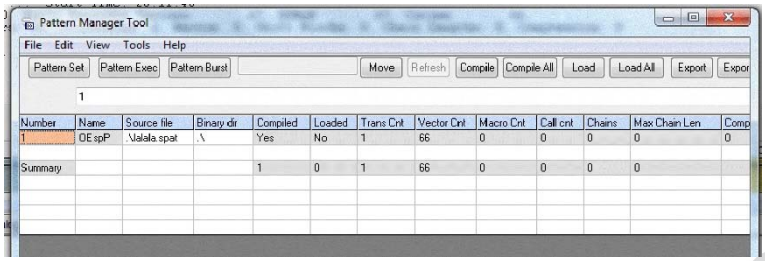


Εικόνα 34. 3 Bit adder Pattern Tool

Επόμενο εργαλείο που καλείται είναι το “Pattern Manager” με το οποίο επιλέγουμε τα προς χρήση patterns, δημιουργούμε αντικείμενα τύπου “Pattern Burst” και “Pattern Exec” όπου και συνδυάζουμε τις πληροφορίες τάσεων και χρονισμού με τα patterns και τα σήματα (Signals, Signal groups).

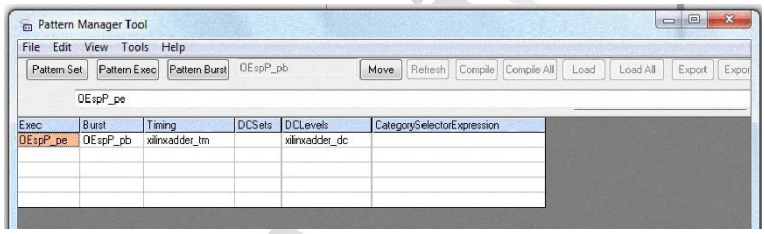
Tolls > Pattern Manager

Σε πρώτη φάση επιλέγουμε τα pattern και τα μεταγλωττίζουμε (compile). Εν συνεχεία δημιουργούμε ένα αντικείμενο τύπου “Pattern Burst” όπου και καθορίζουμε το όνομα του pattern ξανά και το επεξεργαζόμαστε σε περίπτωση που είναι αναγκαίο. Τέλος δημιουργώντας ένα αντικείμενο τύπου “Pattern Exec” συνδυάζουμε τις απαραίτητες πληροφορίες τροφοδοσίας και χρονισμού, βάση των οποίων θα εκτελεστεί το κάθε pattern.



Εικόνα 35. 3 Bit adder Pattern Set Tool

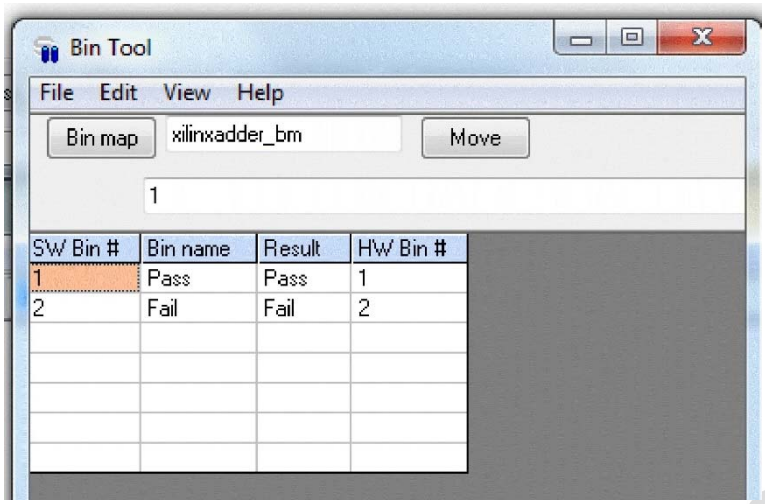
Εικόνα 36. 3 Bit adder Pattern Burst Tool



Εικόνα 37. 3 Bit adder Pattern Exec Tool

Τελειώνοντας με τα εργαλεία διαχείρισης των αρχείων pattern δημιουργούμε ένα αντικείμενο τύπου “Bin map” για την καταγραφή δεδομένων επιτυχίας ή αποτυχίας του εκάστοτε τμήματος (Segment) ελέγχου όπως αυτά αναφέρονται παρακάτω.

Tools > Bin



Εικόνα 38. 3 Bit adder Bin Tool

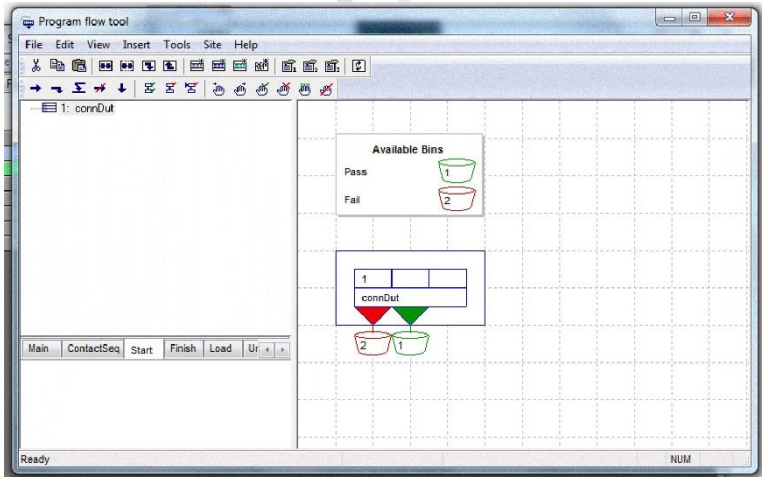
Τελευταίο βήμα πριν την εκτέλεση ενός πλήρους ελέγχου ενός ολοκληρωμένου κυκλώματος είναι η δημιουργία της ροής του προγράμματος ελέγχου. Αυτό γίνεται με τη χρήση του εργαλείου “Program Flow”

Tool > Program Flow

Όπως έχει αναφερθεί νωρίτερα το εργαλείο “Program Flow” με χρήση καρτελών χωρίζει την εκτέλεση του ελέγχου σε στάδιο. Τα βασικά στάδια τα οποία και κάνουμε χρήση εμείς στην παρούσα εργασία είναι, με σειρά εκτέλεσης, τα παρακάτω

Start – Main – Finish

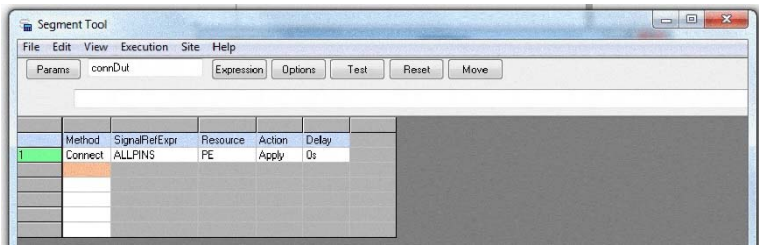
Αρχίζοντας με την καρτέλα “Start”, δημιουργούμε ένα δομικό στοιχείο ελέγχου (Test Segment).



Εικόνα 39. 3 Bit adder Program Flow Tool (Start)

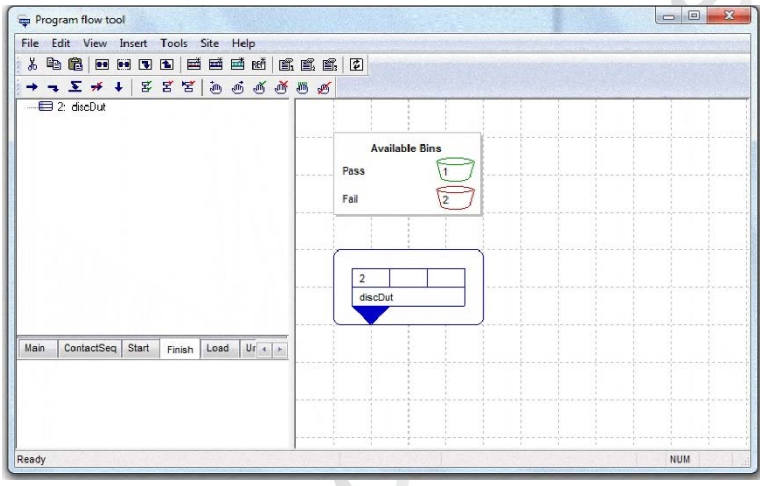
Επεξεργαζόμενοι τις ιδιότητες του δημιουργούμε ένα αντικείμενο που το ονομάζουμε “ConnDut” στο οποίο και επιλέγουμε την επιθυμητή μέθοδο εκτέλεσης. Στην παρούσα καρτέλα

αυτή είναι η “Connect” καθώς εδώ θα ελέγξουμε τη σύνδεση των ηλεκτρικών ρελέ (Pin Electronic Relays) στην αρχή του ελέγχου. Η μέθοδος θα εφαρμοστεί στα σήματα του Signal Group “ALLPINS”, στους πόρους του συστήματος “PE” (Pin Electronic), με επιλεγμένη ενέργεια “Apply” τη χρονική στιγμή 0ns.



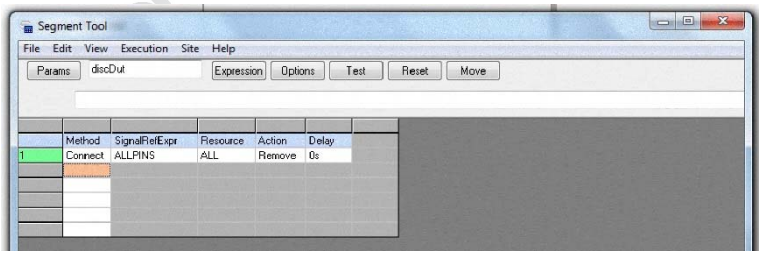
Εικόνα 40. 3 Bit adder “ConnDut” Segment

Στη συνέχει περνώντας στην καρτέλα “Finish” δημιουργούμε ένα δομικό στοιχείο άνευ όρων “Unconditional Segment”



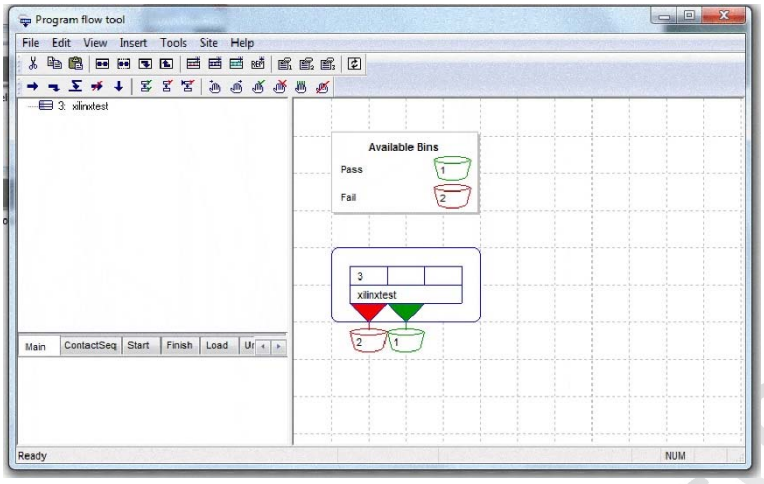
Εικόνα 41. 3 Bit adder Program Flow Tool (Finish)

Επεξεργαζόμενοι τις ιδιότητες του δημιουργούμε ένα αντικείμενο που το ονομάζουμε “DiscDut” στο οποίο και επιλέγουμε την επιθυμητή μέθοδο εκτέλεσης. Στην παρούσα καρτέλα αυτή είναι η “Connect” καθώς εδώ θα τερματίσουμε τη σύνδεση όλων των πόρων στο τέλος του ελέγχου. Η μέθοδος θα εφαρμοστεί στα σήματα του Signal Group “ALLPINS”, στους πόρους του συστήματος “ALL” , με επιλεγμένη ενέργεια “Remove” τη χρονική στιγμή 0ns.



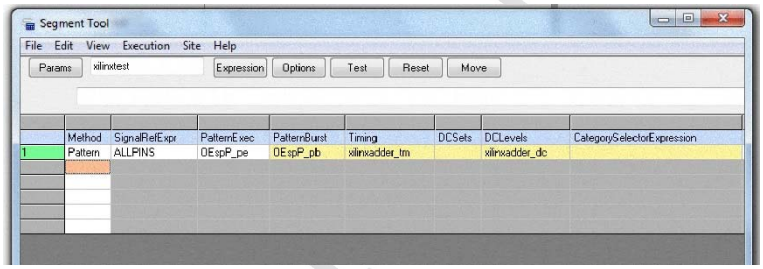
Εικόνα 42. 3 Bit adder “DiscDut” Segment

Η βασικότερη καρτέλα παραμένει η καρτέλα “Main”. Εδώ επιλέγονται και τα προς εκτέλεση pattern. Αρχικά δημιουργούμε ένα δομικό στοιχείο ελέγχου (Test Segment).



Εικόνα 43. 3 Bit adder Program Flow Tool (Main)

Στη συνέχεια επιλέγουμε την επιθυμητή μέθοδο εκτέλεσης η οποία στην παρούσα φάση είναι η μέθοδος “Pattern”. Συμπληρώνοντας τις υπόλοιπες πληροφορίες επιλέγουμε τη μέθοδο που θα εφαρμοστεί στα σήματα του Signal Group “ALLPINS”, με χρήση των “Pattern Exec”, “Pattern burst” αντικειμένων που ορίσαμε νωρίτερα, με στοιχεία χρονισμού από το αντικείμενο “Timing” και τάσεις ορισμένες από το αντικείμενο “DC Levels” που και αυτά καθορίσαμε στη διάρκεια ανάπτυξης της εφαρμογής ελέγχου για τον αθροιστή (3 bit Adder).

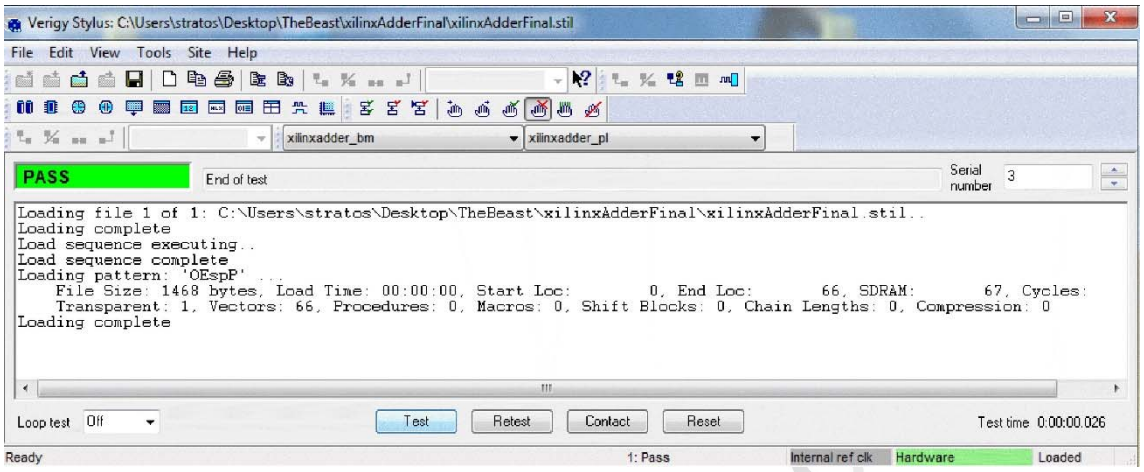


Εικόνα 44. 3 Bit adder “xilinxaddertest” Segment

Μετά το πέρας χρήσης και αυτού του εργαλείου ο χρήστης είναι πλέον έτοιμος να περάσει σε διαδικασία εκτέλεσης του επιθυμητού ελέγχου.

4.1.2 Εκτέλεση αρχείου ελέγχου

Για την εκτέλεση του επιθυμητού ελέγχου αρκεί η ενεργοποίηση της επιλογής “Test” από το περιβάλλον Stylus. Ο ελεγκτής εκτελεί τα επιλεγμένα pattern από την εφαρμογή που έχουμε φορτωμένη και ανάλογα το αποτέλεσμα του μας δίνει αποτέλεσμα “PASS” η “FAIL”. Σε περίπτωση που όλα εκτελεστούν καλώς η εικόνα που παρουσιάζεται είναι η παρακάτω



Εικόνα 45. 3 Bit adder successful test execution

Στο σημείο αυτό το ολοκληρωμένο κύκλωμα μας θεωρείται επιτυχώς ελεγμένο λειτουργικά. Ένα βασικό στοιχείο με το οποίο μπορεί να πειραματιστεί ο χρήστης είναι η χρονική συμπεριφορά του ολοκληρωμένου κυκλώματος. Στο εργαλείο “Timing” ο χρήστης έχει δηλώσει την περίοδο εφαρμογής των διανυσμάτων (Vector Period) και τις χρονικές στιγμές μετά την εφαρμογή κάθε vector στις οποίες το κύκλωμα εκτελεί εις δεδομένες ενέργειες του (Time #, Action #). Μέγιστη περίοδο για κάθε pattern που μπορεί να ορίσει ο χρήστης είναι τα 635ns. Σε περίπτωση που ο χρήστης δηλώσει περίοδο μεγαλύτερη από τα 635ns στο εργαλείο “Timing” το περιβάλλον Stylus ενημερώνει το χρήστη με μήνυμα λάθους στο Status Window. Για τον καλύτερο χρονικό έλεγχο και τον έλεγχο των χρονικών κατωφλιών του ολοκληρωμένου κυκλώματος ο χρήστης μπορεί να εκτελέσει το εργαλείο λογικής ανάλυσης (Logic Analyzer Tool). Στην ουσία γίνεται αποσφαλμάτωση του αρχείου ελέγχου. Εδώ ο χρήστης μπορεί να ξεκινήσει ορίζοντας Period = 630ns και σταδιακά όσο συναντά επιτυχείς εκτελέσεις του ελέγχου να μειώνει την περίοδο και να αλλάζει και τις υπόλοιπες χρονικές επιλογές του αρχείου ελέγχου. Στο συγκεκριμένο παράδειγμα κατεβήκαμε ως το χρόνο περιόδου 200ns και το χρόνο ελέγχου του ορθού αποτελέσματος στους ακροδέκτες εξόδου pinC0, pinC1, pinC2 ίσο με 150ns. Σημαντικό εργαλείο σε αυτή την κατεύθυνση αποτελεί και η δυνατότητα εκτέλεσης διαδοχικών επαναλήψεων (LoopTest) του αρχείου ελέγχου επιλέγοντας την ανάλογη τιμή από το περιβάλλον Stylus όπως φαίνεται και στο κάτω αριστερά μέρος του περιβάλλοντος (εικόνα 39).

Loop Test Selection

- OFF (χωρίς επανάληψη)
- 5
- 10
- 15
- 20
- 25
- 50
- 100
- 500
- 1000
- Infinite (έως ότου διακοπεί από το χρήστη)

Κατά τη διάρκεια διαδοχικών επαναλήψεων κρίνεται σκόπιμη η ενεργοποίηση της επιλογής “Pause on Fail” από το μενού παύσης εκτέλεσης για την άμεση αναγνώριση των σφαλμάτων. Από τις διαδοχικές εκτελέσεις αρχείων ελέγχου και τον πειραματισμό με τα στοιχεία χρονισμού παρατηρήθηκαν τα παρακάτω αξιοσημείωτα συμπεράσματα

Όσο το εκάστοτε κύκλωμα πλησιάζει στην οριακή συμπεριφορά του η εκτέλεση ενός μόνο ελέγχου η μικρού αριθμού επαναλήψεων δεν μας αποδίδει σε πλήρως αξιόπιστο βαθμό την αλήθεια σε περίπτωση επιτυχών ελέγχων. Κρίνεται σκόπιμη η εκτενής επανάληψη ελέγχων με μεγάλο βαθμό επαναλήψεων καθώς παρατηρήθηκαν απρόβλεπτες αποτυχίες μετά από σημαντικό βαθμό επιτυχών εκτελέσεων μοναδικών δοκιμών ή δοκιμών με Loop test ρυθμισμένο μεταξύ των επιλογών 5 – 100.

Ένα επιπλέον κρίσιμο στοιχείο που παρατηρήθηκε είναι η απόκλιση στην χρονική συμπεριφορά των κυκλωμάτων. Το εργαλείο ανάπτυξης Xilinx ISE Project Navigator παρέχει στοιχεία χρονικής ανοχής των υλοποιημένων κυκλωμάτων. Με χρήση του εργαλείου “Timing Analyzer” μας παρουσιάζεται η ελάχιστη χρονική καθυστέρηση μετάδοσης των σημάτων μεταξύ των ακροδεκτών εντός του FPGA ολοκληρωμένου στην πλακέτα Xilinx Spartan 3E. Στην εφαρμογή ελέγχου όμως η χρονική ανοχή στην περίοδο εφαρμογής των διανυσμάτων (vectors) κατά την εκτέλεση των patterns παρουσιάζεται σημαντικά μεγαλύτερη. Κατά τη διάρκεια εκτέλεσης δοκιμών προς εύρεση πληροφορίας σχετικά με αυτό το θέμα καταλήξαμε στη χρήση μέσων σύνδεσης (καλωδίων), μεταξύ του ελεγκτή Personal Ocelot και της πλακέτας που φέρει το ολοκληρωμένο κύκλωμα Xilinx Spartan 3E,διαφορετικού μήκους. Έτσι παρατηρήθηκε ότι η αύξηση στο μήκος των καλωδίων σύνδεσης επενεργεί αρνητικά στην ελάχιστη δυνατή συχνότητα αναγκαία για επιτυχείς εκτελέσεις ελέγχου. Όσο μεγαλώνει το μήκος των καλωδίων τόσο μεγαλώνει και η ελάχιστη απαιτούμενη συχνότητα για την ορθή και επιτυχή εκτέλεση πολλαπλών επαναλήψεων ελέγχου του εκάστοτε ολοκληρωμένου κυκλώματος. Η ποσοτικοποίηση και ο υπολογισμός της σχέσης μεταξύ του μήκους των καλωδίων και της καθυστέρησης (overhead) δεν μπορεί να υπολογιστεί τουλάχιστον στην παρούσα προσπάθεια δημιουργίας ενός οδηγού χρήσης του ελεγκτή Inovys Personal Ocelot με χρήση του προτύπου IEEE Std. 1450 STIL (Standard Test Interface Language).

4.1.3 Παράθεση ολοκληρωμένου αρχείου ελέγχου (xilinxAdderFinal.stil)

```
STIL   1.00;

Signals {
    pinA0 In;
    pinA1 In;
    pinA2 In;
    pinB0 In;
    pinB1 In;
    pinB2 In;
    pinC0 Out;
    pinC1 Out;
    pinC2 Out;
}
PinList xilinxadder_pl {
    Sites 1;
    pinA0 (    0);
    pinA1 (    1);
    pinA2 (    2);
    pinB0 (    6);
    pinB1 (    7);
    pinB2 (    8);
    pinC0 (   15);
```

```
      pinC1 (   14);
      pinC2 (   13);
  }
  SignalGroups {
      ALLPINS                                                                    =
'pinA0+pinA1+pinA2+pinB0+pinB1+pinB2+pinC0+pinC1+pinC2';
  }
  Timing xilinxadder_tm {
      WaveformTable Wft0 {
            Period '200ns';
            Waveforms {
                  ALLPINS { 01 {  '0ns' D/U; }}
                  ALLPINS { LH {  '0ns' Z;  '150ns' L/H; }}
                  ALLPINS { X {  '0ns' Z; }}
            }
      }
  }
  DCLevels xilinxadder_dc {
      ALLPINS{
      VIH 3.3V;
      VOH 3.3V;
      }
  }
  PatternRef OEspP {
      SourceFile ".\lalala.spat";
      BinaryDir ".\";
      Timing xilinxadder_tm;
  }
  PatternBurst xilinxadder_pb {
      PatList {
            OEspP;
      }
  }
  PatternExec OEspP_pe {
      Timing xilinxadder_tm;
      PatternBurst xilinxadder_pb;
      DCLevels xilinxadder_dc;
  }
  Bin "Pass"{
      ProgramResult = "Pass";
      BinNumber =    1;
  }
  Bin "Fail"{
      ProgramResult = "Fail";
      BinNumber =    2;
```

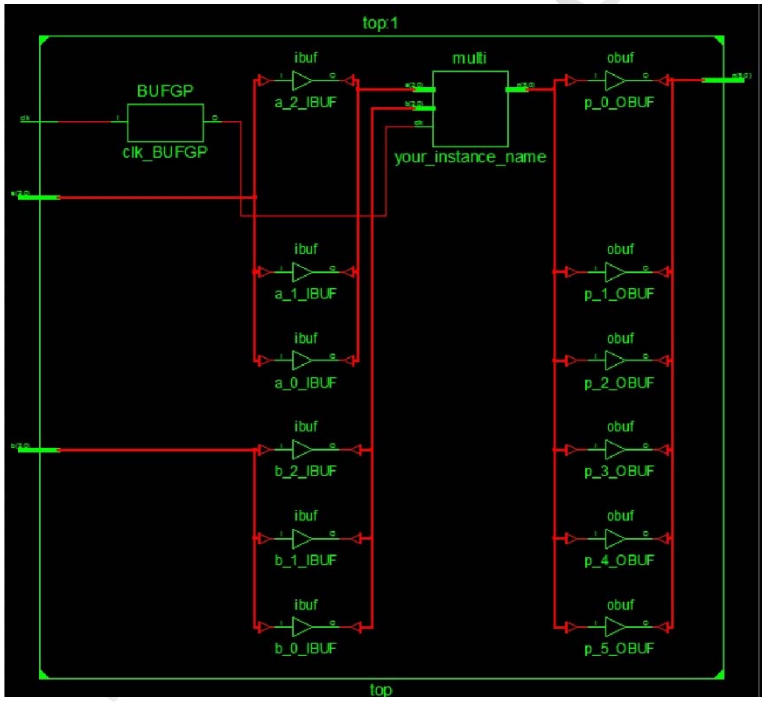
```
  }
  BinMap xilinxadder_bm{
      "Pass" ->    1;
      "Fail" ->    2;
  }
  TestParams ConnDut{
      Method Connect {"SignalRefExpr" 'ALLPINS';"Resource" PE;"Action"
Apply;"Delay" '0s';}
  }
  TestParams DisccDut{
      Method Connect {"SignalRefExpr" 'ALLPINS';"Resource" ALL;"Action"
Remove;"Delay" '0s';}
  }
  TestParams xilinxaddertest{
      Method    Pattern    {"SignalRefExpr"    'ALLPINS';"PatternExec"
OEspP_pe;"BurstMode" true;}
  }
  ProgSeq "Main"{
      Segment 3{
            Type TEST;
            TestParams xilinxaddertest;
            Start true;
            Title "xilinxaddertest";
            Position 120,200;
            Action  { 'Result==Pass'; Bin=1; }
            Action  { 'Result==Fail'; Bin=2; }
      }
  }
  ProgSeq "ContactSeq"{
  }
  ProgSeq "Start"{
      Segment 1{
            Type TEST;
            TestParams ConnDut;
            Start true;
            Title "ConnDut";
            Position 120,200;
      }
  }
  ProgSeq "Finish"{
      Segment 2{
            Type UNCONDITIONAL;
            TestParams DisccDut;
            Start true;
            Title "DisccDut";
```

```
            Position 120,200;
        }
    }
    ProgSeq "Load"{
    }
    ProgSeq "Unload"{
    }
    ProgSeq "API"{
    }
    ProgramObjects {
        ActiveBinMap xilinxadder_bm;
        ActivePinList xilinxadder_pl;
}
```

4.2 ΚΥΚΛΩΜΑ 3-BIT MULTIPLIER

4.2.1 Δημιουργία αρχείου ελέγχου

Για την εκτέλεση του ελέγχου ενός κυκλώματος πολλαπλασιαστή τριών bit (3 bit parallel multiplier) αρχικά υλοποιούμε το κύκλωμα. Αυτό γίνεται με χρήση του Xilinx ISE (Project Navigator) με αποτέλεσμα την παρακάτω σχηματική απεικόνισή του.



Εικόνα 46. Multiplier Schematic block contents

Η δημιουργία του παράλληλου πολλαπλασιαστή (multiplier) γίνεται με χρήση του εργαλείου “Core Generator” του Xilinx ISE Project Navigator και παρουσιάζεται σε επόμενο κεφάλαιο. Η βασική διαφορά με το ολοκληρωμένο κύκλωμα του 3-bit Adder έγκειται στο ότι το παρόν κύκλωμα έχει και ακροδέκτη λειτουργίας ρολογιού, το οποίο και θα τροφοδοτείται επίσης

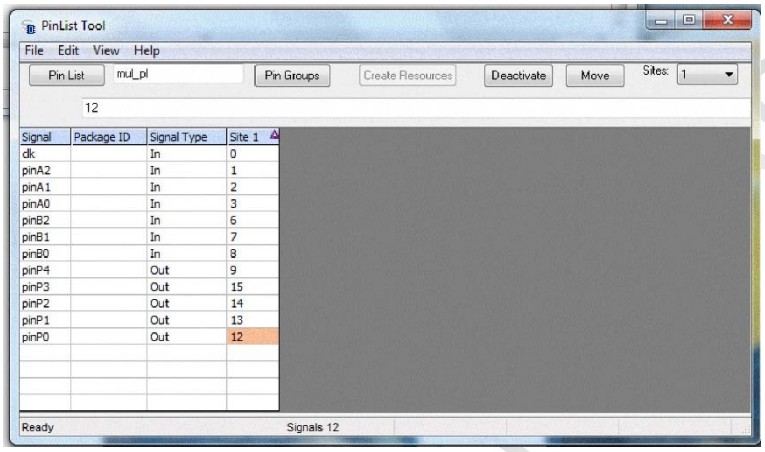
εξωτερικά από τον ελεγκτή κατά τη διάρκεια εκτέλεσης του pattern ελέγχου. Επίσης λόγω ανεπαρκούς αριθμού ακροδεκτών στην πλακέτα Xilinx Spartan 3E αγνοείται το ένα bit του αποτελέσματος της πράξης του πολλαπλασιασμού.

Όπως έχει αναφερθεί νωρίτερα δημιουργούμε ένα νέο αρχείο

File > New

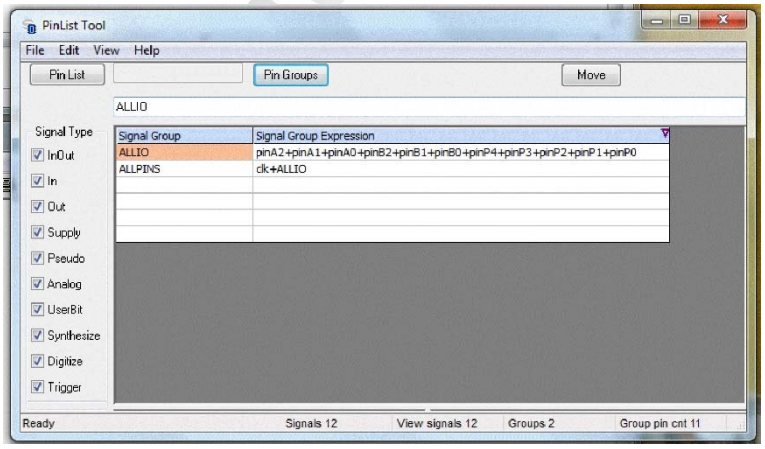
Με χρήση του εργαλείου “Pin List” δημιουργούμε τα απαραίτητα σήματα εισόδου και εξόδου που αντιστοιχούν σε κάθε bit τελεστέων και αποτελέσματος στο αντικείμενο “mul_pl”. Έτσι έχουμε

Tools > Pin List



Εικόνα 47. Multiplier Pin List Tool (Pin List)

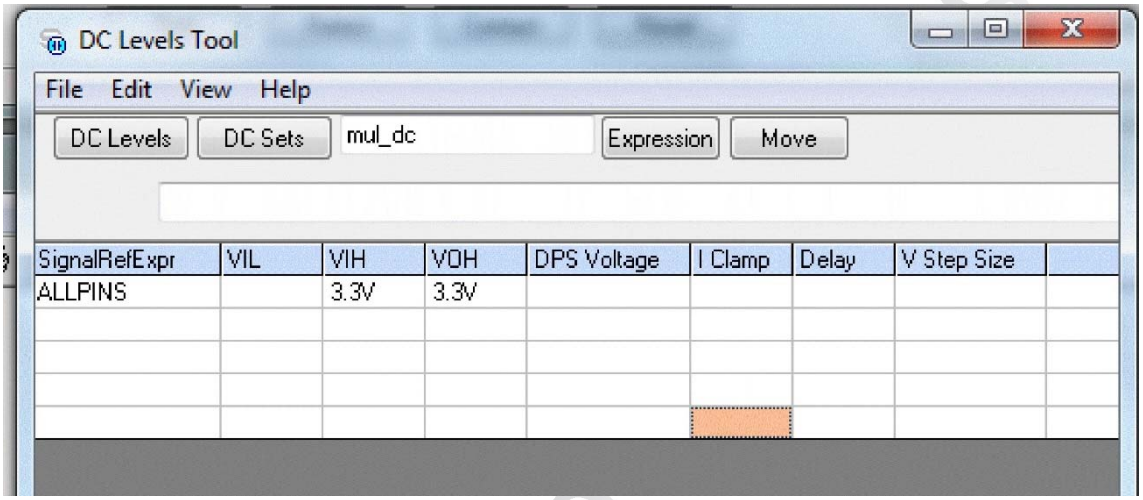
Ο χρήστης εδώ δημιουργεί ένα αντικείμενο τύπου “Pin List” και καθορίζει τα σήματα clk, pinA2, pinA1, pinA0, pinB2, pinB1, pinB0 τα οποία και είναι σήματα εισόδου. Επιπλέον καθορίζονται τα σήματα pinP4, pinP3, pinP2, pinP1, pinP0 τα οποία και είναι σήματα εξόδου και μας δίνουν το αποτέλεσμα της πράξης του πολλαπλασιασμού. Όπως φαίνεται αγνοείται το σήμα που θα αντιστοιχούσε στο pinP5 καθώς δεν επαρκούν οι ακροδέκτες εισόδου / εξόδου της πλακέτας στην οποία υλοποιούνται τα κυκλώματά μας (Xilinx Spartan 3E). Στη συνέχεια ομαδοποιούμε τα σήματα δημιουργώντας τα απαιτούμενα “Signal Groups”



Εικόνα 48. Multiplier Pin List Tool (PinGroups)

Στη συνέχεια σώνουμε το αρχείο της εφαρμογής ελέγχου και προχωράμε επιλέγοντας το εργαλείο “DC Levels”. Δημιουργούμε ένα αντικείμενο τύπου “DC Levels” το “mul_dc” και καθορίζουμε τις τάσεις που εφαρμόζονται στους ακροδέκτες εισόδου και εξόδου (VIH, VOH) στα 3.3V καθώς αυτή η τάση εφαρμόζεται στους ακροδέκτες της πλακέτας Xilinx Spartan 3E με βάση τις τεχνικές προδιαγραφές της.

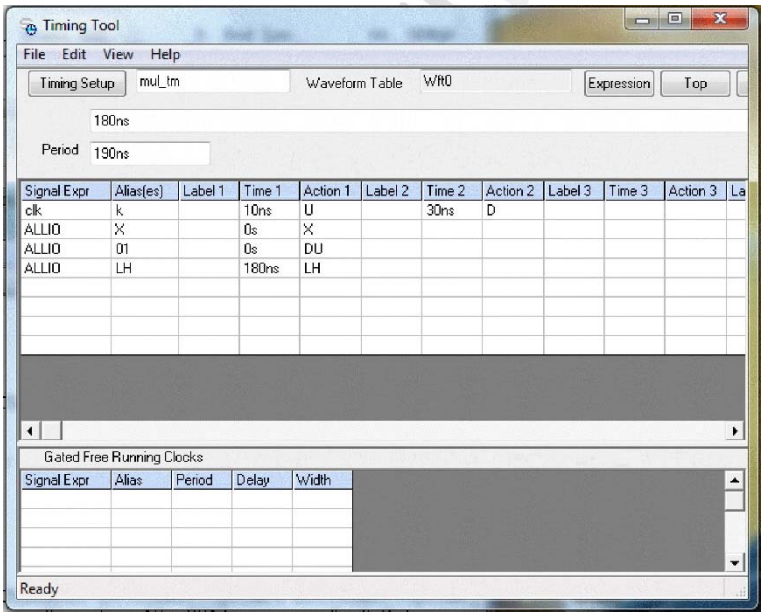
Tools > DC Levels



Εικόνα 49. Multiplier DC Levels Tool

Επόμενο εργαλείο σε χρήση είναι το “Timing Tool”

Tools > Timing



Εικόνα 50. Multiplier Timing Tool

Δημιουργώντας το αντικείμενο “mul_tm” τύπου “Timing”, καθορίζουμε τη συμπεριφορά των σημάτων που δημιουργήθηκαν πριν. Στο συγκεκριμένο παράδειγμα με βάση τον Πίνακα 3 της παρούσης και τις δηλώσεις σημάτων που έγιναν με το εργαλείο “Pin List” έχουμε τα παρακάτω.

- Για το σήμα clk, όταν το pattern εμφανίζει το Alias “k” το σήμα εμφανίζει συμπεριφορά οδήγησης υψηλής κατάστασης [Action 1 = U] (Drive state , assert High Value) μετά από 10ns [Time 1 = 10ns] και συμπεριφορά οδήγησης χαμηλής κατάστασης [Action 2 = D] (Drive state , assert Low Value) μετά από 30ns [Time 2 = 30ns]. Με αυτό τον τρόπο ενεργοποιείται το κύκλωμα του πολλαπλασιαστή που με βάση τα τεχνικά του χαρακτηριστικά είναι τύπου “rising edge clock input”
- Για τα σήματα του Signal group “ALLIO” όταν στο pattern εμφανίζεται το Alias “X” το σήμα αποσυνδέεται [Action 1 = Z] (Drive State, remove or disconnect) μετά από 0ns [Time 1 = 0ns]
- Για τα σήματα του Signal group “ ALLIO ” όταν στο pattern εμφανίζεται το Alias “0” το σήμα εμφανίζει συμπεριφορά οδήγησης χαμηλής κατάστασης [Action 1 = D] (Drive state , assert Low Value) μετά από 0ns [Time 1 = 0ns]
- Για τα σήματα του Signal group “ALLIO” όταν στο pattern εμφανίζεται το Alias “1” το σήμα εμφανίζει συμπεριφορά οδήγησης υψηλής κατάστασης [Action 1 = U] (Drive state , assert High Value) μετά από 0ns [Time 1 = 0ns]
- Για τα σήματα του Signal group “ ALLIO ” όταν στο pattern εμφανίζεται το Alias “L” το σήμα εμφανίζει συμπεριφορά σύγκρισης χαμηλής κατάστασης [Action 1 = L] (Compare state , detect Low Value) μετά από 180ns [Time 1 = 180ns]
- Για τα σήματα του Signal group “ALLIO” όταν στο pattern εμφανίζεται το Alias “H” το σήμα εμφανίζει συμπεριφορά σύγκρισης υψηλής κατάστασης [Action 1 = H] (Compare state , detect High Value) μετά από 180ns [Time 1 = 180ns]

Το επόμενο εργαλείο σε χρήση είναι το “Pattern” με το οποίο και δημιουργούμε pattern για την εκτέλεση των ελέγχων.

Tools > Pattern

Όπως έχει αναφερθεί νωρίτερα αντί να κάνουμε χρήση του εργαλείου “Pattern” του περιβάλλοντος Stylus επιλέγουμε την αυτόματη παραγωγή ενός αρχείου κειμένου (text) από το εργαλείο προσομοίωσης του ολοκληρωμένου κυκλώματος υπό δοκιμή, μέσω του περιβάλλοντος Xilinx ISE Project Navigator και τον προσομοιωτή ISim ή τον Modelsim κατά τη διάρκεια εκτέλεσης του testbench αρχείου . Επεξεργαζόμαστε μετά κατάλληλα το αρχείο κειμένου και το αποθηκεύουμε στην κατάλληλη μορφή μετατρέποντάς το από “ .txt ” σε “.spat”. Ο τρόπος παρουσιάζεται με λεπτομέρεια στο επόμενο κεφάλαιο όπου και περιγράφεται ο τρόπος ανάπτυξης του κυκλώματος στο περιβάλλον Xilinx ISE. Με αυτόν τον τρόπο μπορούμε να δημιουργήσουμε το αρχείο pattern αυτόματα με την ελάχιστη δυνατή ανάγκη επεξεργασίας.

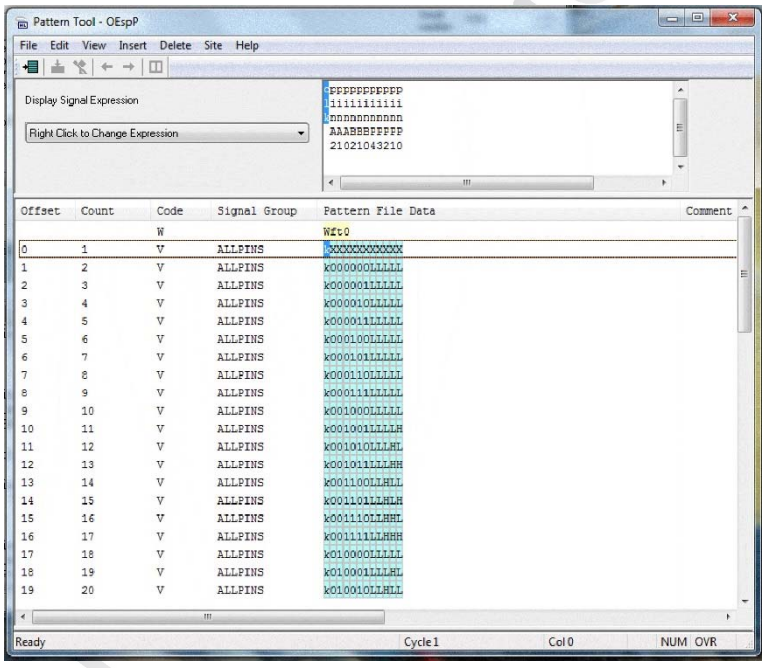
Έτσι το αρχείο pattern μετά τη δημιουργία του από το εργαλείο προσομοίωσης του κυκλώματος και μετά την τυχόν αναγκαία επεξεργασία έχει την παρακάτω μορφή

```
Pattern OEspP{
W Wft0;
V{ALLPINS=kXXXXXXXXXXX;}
V{ALLPINS=k000000LLLLL;}
V{ALLPINS=k000001LLLLL;}
V{ALLPINS=k000010LLLLL;}
V{ALLPINS=k000011LLLLL;}
```

```
V{ALLPINS=k000100LLLLL;}
V{ALLPINS=k000101LLLLL;}
V{ALLPINS=k000110LLLLL;}
V{ALLPINS=k000111LLLLL;}
V{ALLPINS=k001000LLLLL;}
V{ALLPINS=k001001LLLLH;}
V{ALLPINS=k001010LLLHL;}
V{ALLPINS=k001011LLLHH;}
V{ALLPINS=k001100LLHLL;}
V{ALLPINS=k001101LLHLH;}
V{ALLPINS=k001110LLHHL;}
V{ALLPINS=k001111LLHHH;}
V{ALLPINS=k010000LLLLL;}
V{ALLPINS=k010001LLLHL;}
V{ALLPINS=k010010LLHLL;}
V{ALLPINS=k010011LLHHL;}
V{ALLPINS=k010100LHLLL;}
V{ALLPINS=k010101LHLHL;}
V{ALLPINS=k010110LHHLL;}
V{ALLPINS=k010111LHHHL;}
V{ALLPINS=k011000LLLLL;}
V{ALLPINS=k011001LLLHH;}
V{ALLPINS=k011010LLHHL;}
V{ALLPINS=k011011LHLLH;}
V{ALLPINS=k011100LHHLL;}
V{ALLPINS=k011101LHHHH;}
V{ALLPINS=k011110HLLHL;}
V{ALLPINS=k011111HLHLH;}
V{ALLPINS=k100000LLLLL;}
V{ALLPINS=k100001LLHLL;}
V{ALLPINS=k100010LHLLL;}
V{ALLPINS=k100011LHHLL;}
V{ALLPINS=k100100HLLLL;}
V{ALLPINS=k100101HLHLL;}
V{ALLPINS=k100110HHLLL;}
V{ALLPINS=k100111HHHLL;}
V{ALLPINS=k101000LLLLL;}
V{ALLPINS=k101001LLHLH;}
V{ALLPINS=k101010LHLHL;}
V{ALLPINS=k101011LHHHH;}
V{ALLPINS=k101100HLHLL;}
V{ALLPINS=k101101HHLLH;}
V{ALLPINS=k101110HHHHL;}
V{ALLPINS=k101111LLLHH;}
```

```
V{ALLPINS=k110000LLLLL;}
V{ALLPINS=k110001LLHHL;}
V{ALLPINS=k110010LHHLL;}
V{ALLPINS=k110011HLLHL;}
V{ALLPINS=k110100HHLLL;}
V{ALLPINS=k110101HHHHL;}
V{ALLPINS=k110110LLHLL;}
V{ALLPINS=k110111LHLHL;}
V{ALLPINS=k111000LLLLL;}
V{ALLPINS=k111001LLHHH;}
V{ALLPINS=k111010LHHHL;}
V{ALLPINS=k111011HLHLH;}
V{ALLPINS=k111100HHHLL;}
V{ALLPINS=k111101LLLHH;}
V{ALLPINS=k111110LHLHL;}
V{ALLPINS=k111111HLLLH;}
V{ALLPINS=kXXXXXXXXXXX;}
}
```

Ανοίγοντας το αρχείο pattern με χρήση του εργαλείου “Pattern” έχουμε την παρακάτω εικόνα

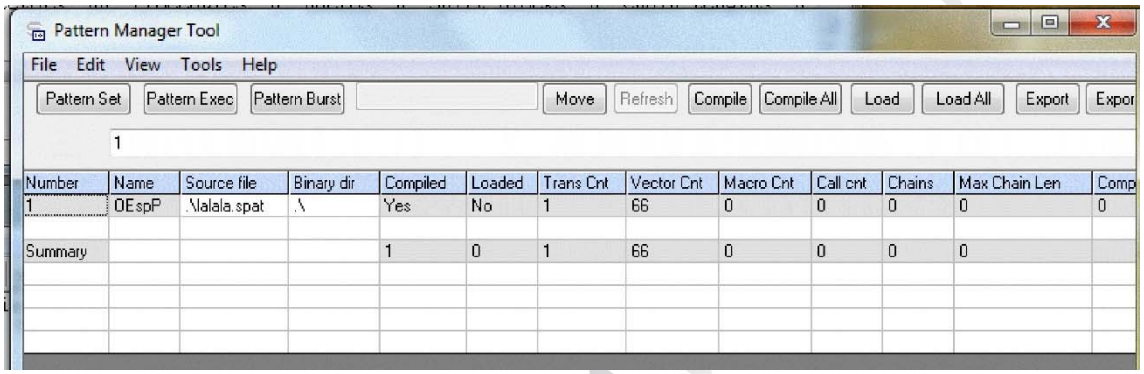


Εικόνα 51. Multiplier Pattern Tool

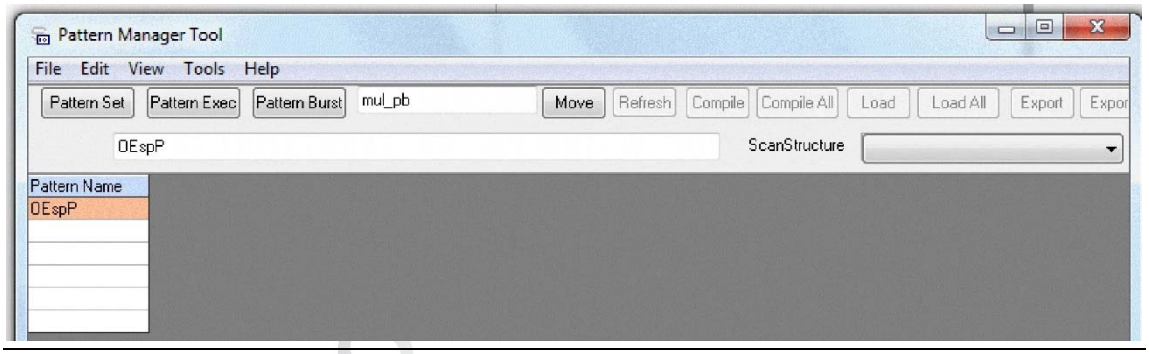
Επόμενο εργαλείο που καλείται είναι το “Pattern Manager” με το οποίο επιλέγουμε τα προς χρήση patterns, δημιουργούμε αντικείμενα τύπου “Pattern Burst” και “Pattern Exec” όπου και συνδυάζουμε τις πληροφορίες τάσεων και χρονισμού με τα patterns και τα σήματα (Signals, Signal groups).

Tolls > Pattern Manager

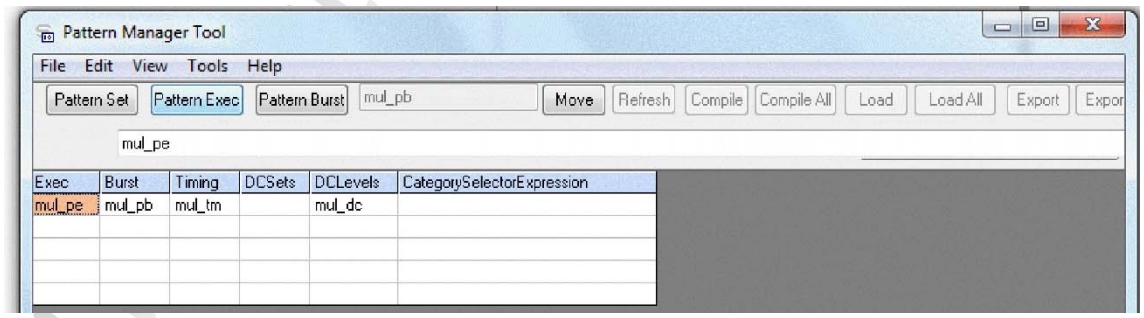
Σε πρώτη φάση επιλέγουμε τα pattern και τα μεταγλωττίζουμε (compile). Εν συνεχεία δημιουργούμε ένα αντικείμενο τύπου “Pattern Burst” όπου και καθορίζουμε το όνομα του pattern ξανά και το επεξεργαζόμαστε σε περίπτωση που είναι αναγκαίο. Τέλος δημιουργώντας ένα αντικείμενο τύπου “Pattern Exec” συνδυάζουμε τις απαραίτητες πληροφορίες τροφοδοσίας και χρονισμού, βάση των οποίων θα εκτελεστεί το κάθε pattern.



Εικόνα 52. Multiplier Pattern Set Tool



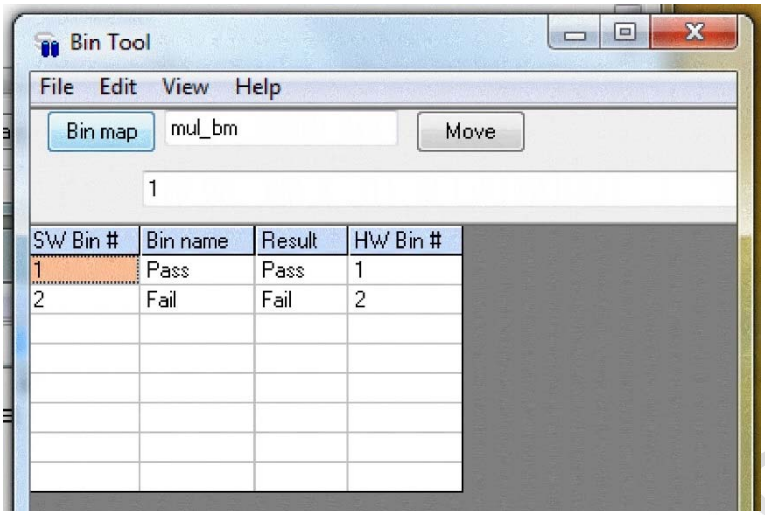
Εικόνα 53. Multiplier Pattern Burst Tool



Εικόνα 54. Multiplier Pattern Exec Tool

Τελειώνοντας με τα εργαλεία διαχείρισης των αρχείων pattern δημιουργούμε το αντικείμενο τύπου “Bin map”, “mul_bm” για την καταγραφή δεδομένων επιτυχίας ή αποτυχίας του εκάστοτε τμήματος (Segment) ελέγχου όπως αυτά αναφέρονται παρακάτω.

Tools > Bin



Εικόνα 55. Multiplier Bin Tool

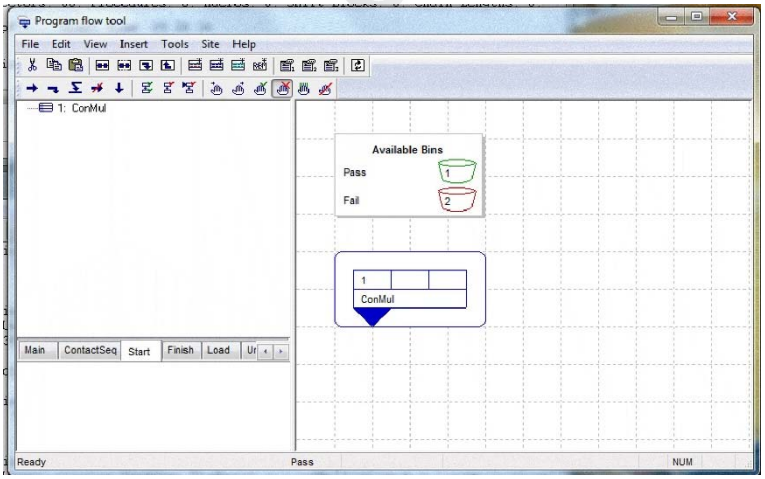
Τελευταίο βήμα πριν τον εκτέλεση ενός πλήρους ελέγχου ενός ολοκληρωμένου κυκλώματος είναι η δημιουργία της ροής του προγράμματος ελέγχου. Αυτό γίνεται με τη χρήση του εργαλείου “Program Flow”

Tool > Program Flow

Θυμίζουμε τα βασικά στάδια τα οποία και κάνουμε χρήση εμείς στην παρούσα εργασία είναι, με σειρά εκτέλεσης, τα παρακάτω

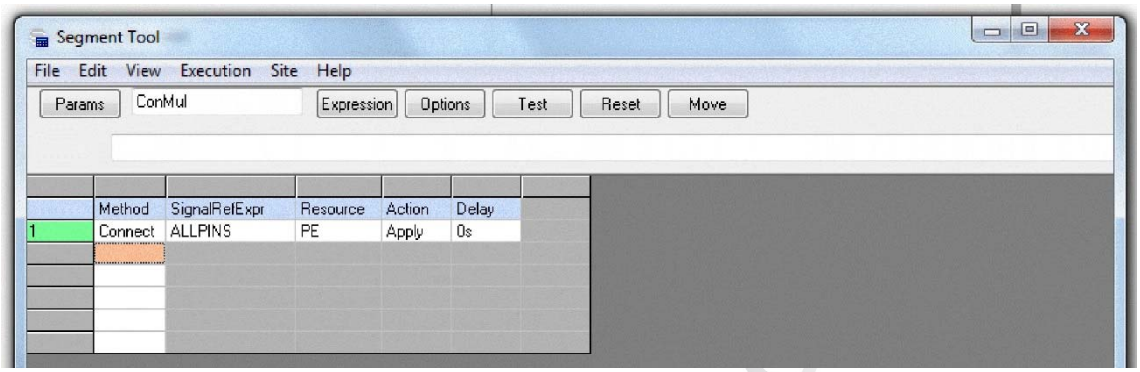
Start – Main – Finish

Αρχίζοντας με την καρτέλα “Start”, δημιουργούμε ένα δομικό στοιχείο ελέγχου (Test Segment).



Εικόνα 56. Multiplier Program Flow Tool (Start)

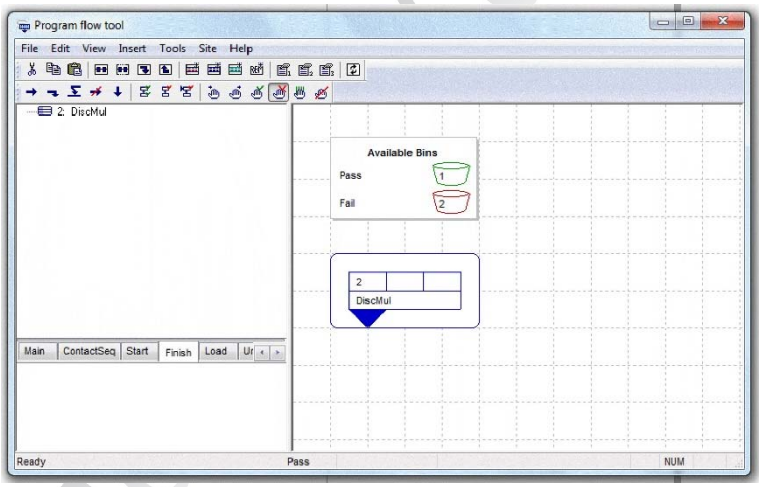
Εδώ σε αντίθεση με το προηγούμενο παράδειγμα του αθροιστή στην καρτέλα “Start” του εργαλείου “Program Flow” δημιουργούμε ένα άνευ όρων δομικό στοιχείο (Unconditional Segment). Λειτουργικά στην παρούσα καρτέλα έχει το ίδιο αποτέλεσμα καθώς εκτελεί την ίδια μέθοδο. Έτσι έχουμε



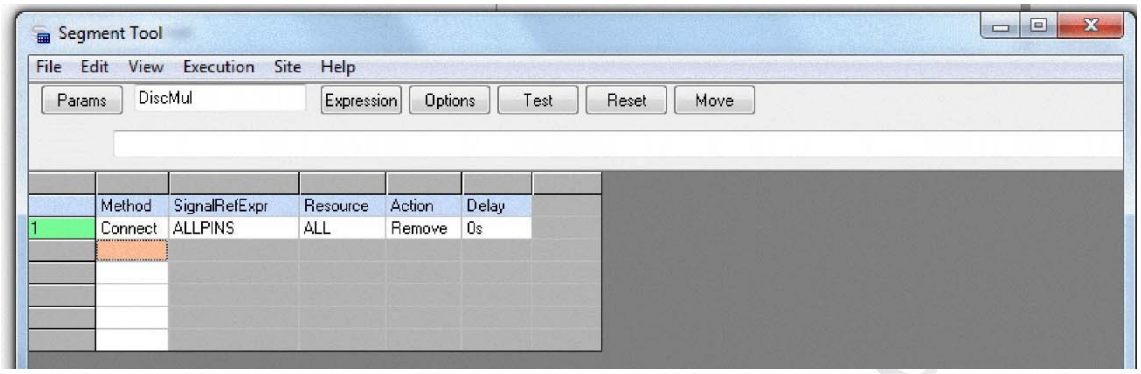
Εικόνα 57. Multiplier “ConMul” Segment

Επεξεργαζόμενοι τις ιδιότητες του άνευ όρων δομικού στοιχείου, δημιουργούμε ένα αντικείμενο που το ονομάζουμε “ConnMul” στο οποίο και επιλέγουμε την επιθυμητή μέθοδο εκτέλεσης. Στην παρούσα καρτέλα αυτή είναι η “Connect” καθώς εδώ θα ελέγξουμε τη σύνδεση των ηλεκτρικών ρελέ (Pin Electronic Relays) στην αρχή του ελέγχου. Η μέθοδος θα εφαρμοστεί στα σήματα του Signal Group “ALLPINS”, στους πόρους του συστήματος “PE” (Pin Electronic), με επιλεγμένη ενέργεια “Apply” τη χρονική στιγμή 0ns.

Στη συνέχεια επεξεργαζόμαστε την καρτέλα “Finish” όπου και δημιουργούμε ένα ακόμα άνευ όρων στοιχείο δομικού ελέγχου με το οποίο και διακόπτουμε τη σύνδεση όλων των πόρων ως εξής



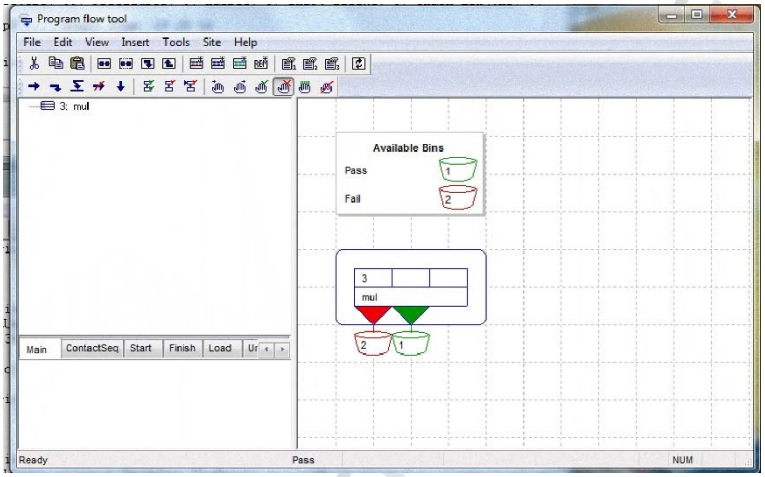
Εικόνα 58. Multiplier Program Flow Tool (Finish)



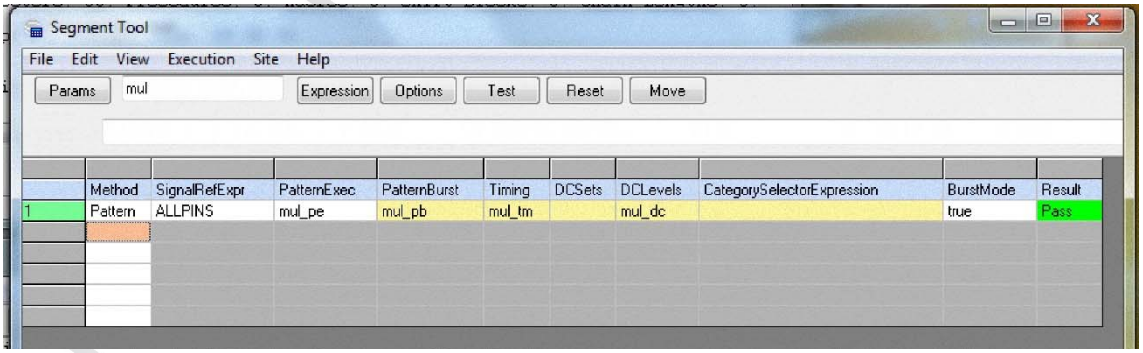
Εικόνα 59. Multiplier "ConMul" Segment

Κάνουμε χρήση της μεθόδου "Connect" με την οποία 0s μετά την εκτέλεση της διακόπτει (Action = Remove) κάθε σύνδεση με όλους τους πόρους (Resources = ALL).

Στη βασική καρτέλα του εργαλείου "Program Flow" εμφανίζεται το δομικό στοιχείο ελέγχου "mul".



Εικόνα 60. Multiplier Program Flow Tool (Main)



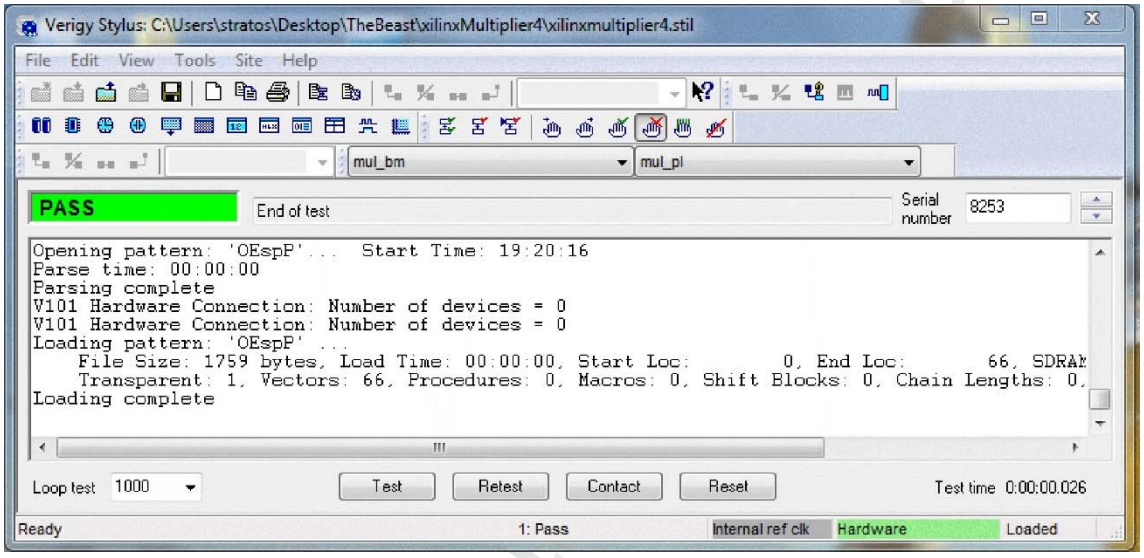
Εικόνα 61. Multiplier "mul" Segment

Εδώ όπως φαίνεται στις παραπάνω εικόνες επιλέγουμε τη μέθοδο "Pattern" μέσω της οποίας θα καλέσουμε το pattern που έχει δημιουργηθεί νωρίτερα. Εφαρμόζεται στα σήματα που περιλαμβάνονται στην ομάδα σημάτων "ALLPINS" που περιέχει το σήμα clk, και την ομάδα σημάτων "ALLIO", συνδυάζοντας πληροφορίες από τα αντικείμενα τύπου "Pattern Exec", "Pattern Burst", "DC Levels" και "Timing" που έχουν δημιουργηθεί νωρίτερα στην εφαρμογή μας. Έτσι

ενώνοντας τα κομμάτια της εφαρμογής ελέγχου είμαστε πλέον έτοιμη να περάσουμε στην εκτέλεση του ελέγχου του ολοκληρωμένου κυκλώματος του πολλαπλασιαστή.

4.2.2 Εκτέλεση αρχείου ελέγχου

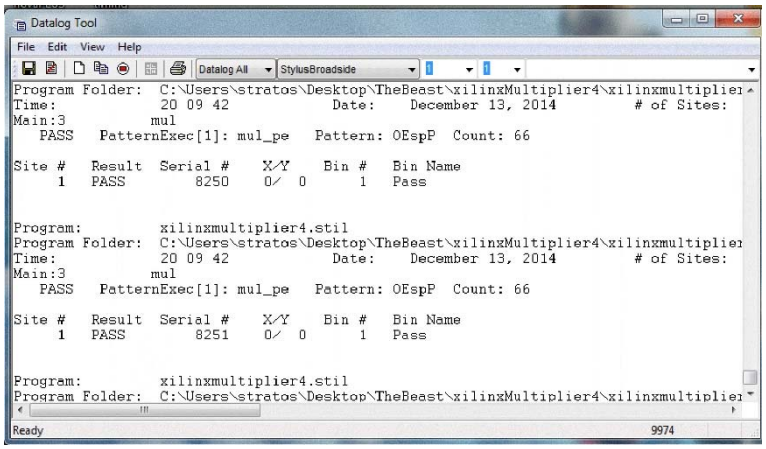
Ακολουθώντας την ίδια διαδικασία με το προηγούμενο παράδειγμα, μετά και την ολοκλήρωση της παραμετροποίησης της εφαρμογής, μπορούμε πλέον να εκτελέσουμε το πρόγραμμα ελέγχου. Επιλέγοντας την εκτέλεση είτε σε μοναδικές επαναλήψεις είτε ενεργοποιώντας την επιλογή “Loop Test” .



Εικόνα 62. Multiplier successful test execution

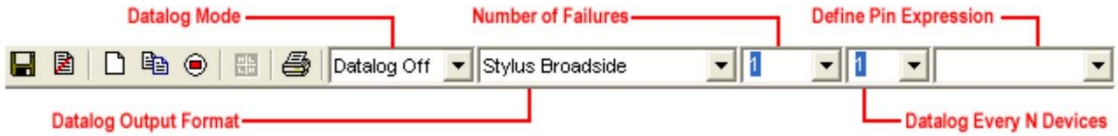
Στην παραπάνω εικόνα βλέπουμε το επιτυχημένο αποτέλεσμα εκτέλεσης 1000 διαδοχικών επαναλήψεων της εφαρμογής ελέγχου, του ολοκληρωμένου κυκλώματος πολλαπλασιαστή στην πλακέτα Xilinx Spartan 3E, στον ελεγκτή ολοκληρωμένων κυκλωμάτων Inovys Personal Ocelot με χρήση του προτύπου IEEE Std. 1450 STIL (Standard Test Interface Language).

Επιπλέον πληροφορίες για τις εκτελέσεις ελέγχων, όπως και ειδοποιήσεων και σφαλμάτων μας δίνει και το εργαλείο “Datalog”. Αν και έχει παραληφθεί η αναφορά του καθώς ενεργοποιείται αυτόματα με την εκκίνηση του περιβάλλοντος Stylus μας δίνει πληροφορίες κάθε είδους σε περίπτωση που το ενεργοποιήσουμε. Όπως βλέπουμε στην παρακάτω εικόνα εκτελέστηκε το πρόγραμμα “xilinxmultiplier4.stil” το οποίο βρίσκεται στο φάκελο “C:\Users\stratos\Desktop\TheBeast\xilinxMultipier4” ώρα και ημέρα “20 09 42, December 13.2014” σε ένα site, με αποτέλεσμα “PASS”. Ήταν ο αριθμός εκτέλεσης 8251 και το αποτέλεσμα οδηγήθηκε στον Bin με όνομα “Pass” και αριθμό 1.



Εικόνα 63. Multiplier Datalog Tool

Από το μενού του εργαλείου “Datalog” μπορούμε να επιλέξουμε τον τρόπο λειτουργίας, τον τρόπο εμφάνισης, τον αριθμό των σφαλμάτων που θα εμφανίζονται και τη συχνότητα με την οποία θα εμφανίζονται σφάλματα. Αυτές οι επιλογές ρυθμίζονται από τα ανάλογα μενού της μπάρας εργασίας του εργαλείου όπως φαίνεται παρακάτω.



Εικόνα 64. Datalog Tool Menu bar

Οι επιλογές κάθε μενού είναι οι εξής

Datalog Mode

- Datalog Off : απενεργοποιημένο
- Datalog All : καταγραφή όλων των αποτελεσμάτων
- Fails Only : καταγραφή αποτυχημένων αποτελεσμάτων μόνο

Datalog Output Format (βασικότερες επιλογές)

- Stylus Broadside : παρουσίαση όπως στα περισσότερα εργαλεία εξοπλισμού αυτομάτου ελέγχου (ATE) όπου αναγράφεται το όνομα του ακροδέκτη και εάν το αποτέλεσμα του είναι σωστό σημειώνεται με “.” ενώ εάν είναι λάθος το αποτέλεσμα σημειώνεται με “*”.
- Stylus Bradside2Pass : όταν ανιχνεύεται λάθος ο έλεγχος εκτελείται ξανά και καταγράφονται περισσότερες λεπτομέρειες.
- FastScan : τρόπος εμφάνισης πληροφοριών της εταιρίας Mentor Graphics
- TetraMAX : τρόπος εμφάνισης πληροφοριών της εταιρίας Synopsis

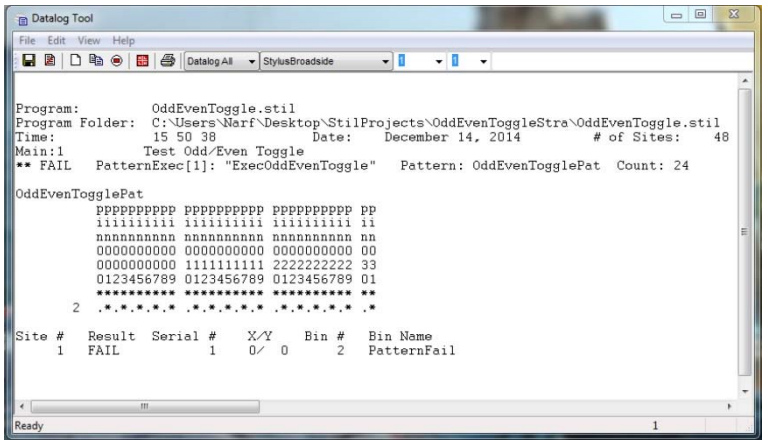
Number of Failures

- Ο χρήστης επιλέγει είτε από το drop down menu είτε δηλώνοντας αριθμό τις αποτυχίες που θα εμφανιστούν στην οθόνη. Μετά το πέρας του αριθμού δεν εμφανίζονται άλλες

Datalog Every N Devices

- Ο χρήστης επιλέγει είτε από το drop down menu είτε δηλώνοντας αριθμό μετά από ποια εκτέλεση θα καταγράφονται αποτυχίες

Define Pin Expression : Δεν εμφανίζεται στην έκδοση Stylus που έχουμε.



Εικόνα 65. Datalog Tool Result

Στην παραπάνω εικόνα βλέπουμε τον τρόπο εμφάνισης του αποτελέσματος μιας δοκιμής με σφάλματα. Αναγράφεται η λίστα με τους ακροδέκτες και στη συνέχεια με τη χρήση των συμβόλων "." και "*" δηλώνεται εάν το αποτέλεσμα κάθε ακροδέκτη είναι σωστό η λανθασμένο.

4.2.3 Παράθεση ολοκληρωμένου αρχείου ελέγχου (xilinxmultiplier.stil)

```
STIL   1.00;

  Signals {
      clk In;
      pinA2 In;
      pinA1 In;
      pinA0 In;
      pinB2 In;
      pinB1 In;
      pinB0 In;
      pinP4 Out;
      pinP3 Out;
      pinP2 Out;
      pinP1 Out;
      pinP0 Out;
  }
  PinList mul_pl {
      Sites 1;
      clk (    0);
      pinA2 (    1);
      pinA1 (    2);
      pinA0 (    3);
      pinB2 (    6);
      pinB1 (    7);
      pinB0 (    8);
      pinP4 (    9);
```

```
      pinP3 (   15);
      pinP2 (   14);
      pinP1 (   13);
      pinP0 (   12);
  }
  SignalGroups {
      ALLIO                                                                    =
'pinA2+pinA1+pinA0+pinB2+pinB1+pinB0+pinP4+pinP3+pinP2+pinP1+pinP0';
      ALLPINS = 'clk+ALLIO';
  }
  Timing mul_tm {
      WaveformTable Wft0 {
            Period '190ns';
            Waveforms {
                  clk { k {  '10ns' U;  '30ns' D; }}
                  ALLIO { X {  '0s' X; }}
                  ALLIO { 01 {  '0s' D/U; }}
                  ALLIO { LH {  '180ns' L/H; }}
            }
      }
  }
  DCLevels mul_dc {
      ALLPINS{
      VIH 3.3V;
      VOH 3.3V;
      }
  }
  PatternRef OEspP {
      SourceFile ".\lalala.spat";
      BinaryDir ".\";
      Timing mul_tm;
  }
  PatternBurst mul_pb {
      PatList {
            OEspP;
      }
  }
  PatternExec mul_pe {
      Timing mul_tm;
      PatternBurst mul_pb;
      DCLevels mul_dc;
  }
  Bin "Pass"{
      ProgramResult = "Pass";
      BinNumber =    1;
```

```
  }
  Bin "Fail"{
      ProgramResult = "Fail";
      BinNumber =    2;
  }
  BinMap mul_bm{
      "Pass" ->    1;
      "Fail" ->    2;
  }
  TestParams ConMul{
      Method Connect {"SignalRefExpr" 'ALLPINS';"Resource" PE;"Action"
Apply;"Delay" '0s';}
  }
  TestParams DiscMul{
      Method Connect {"SignalRefExpr" 'ALLPINS';"Resource" ALL;"Action"
Remove;"Delay" '0s';}
  }
  TestParams mul{
      Method    Pattern    {"SignalRefExpr"    'ALLPINS';"PatternExec"
mul_pe;"BurstMode" true;}
  }
  ProgSeq "Main"{
      Segment 3{
            Type TEST;
            TestParams mul;
            Start true;
            Title "mul";
            Position 120,200;
            Action  { 'Result==Pass'; Bin=1; }
            Action  { 'Result==Fail'; Bin=2; }
      }
  }
  ProgSeq "ContactSeq"{
  }
  ProgSeq "Start"{
      Segment 1{
            Type UNCONDITIONAL;
            TestParams ConMul;
            Start true;
            Title "ConMul";
            Position 120,200;
      }
  }
  ProgSeq "Finish"{
      Segment 2{
```

```
            Type UNCONDITIONAL;
            TestParams DiscMul;
            Start true;
            Title "DiscMul";
            Position 120,200;
        }
    }
    ProgSeq "Load"{
    }
    ProgSeq "Unload"{
    }
    ProgSeq "API"{
    }
    ProgramObjects {
        ActiveBinMap mul_bm;
        ActivePinList mul_pl;
}
```

5. Υλοποίηση ολοκληρωμένων κυκλωμάτων σε περιβάλλον Xilinx ISE Project Navigator στην πλακέτα Xilinx Spartan 3E

Στο παρόν κεφάλαιο γίνεται μια περιγραφική παρουσίαση της υλοποίησης των ολοκληρωμένων κυκλωμάτων που χρησιμοποιήθηκαν για την δημιουργία του οδηγού χρήσης του ελεγκτή Inovys Personal Ocelot και του προτύπου IEEE Std. 1450 STIL (Standard Test Interface Language). Η αναλυτική παρουσίαση του περιβάλλοντος Xilinx ISE Project Navigator δεν εμπίπτει στο θέμα της παρούσης. Για την καλύτερη κατανόηση του περιβάλλοντος προτείνεται η ανάγνωση του οδηγού «Σύντομος Οδηγός Εκμάθησης του Λογισμικού Xilinx ISE» του τμήματος Πληροφορικής του Πανεπιστημίου Πειραιά.

5.1 ΥΛΟΠΟΙΗΣΗ ΚΥΚΛΩΜΑΤΟΣ ΑΘΡΟΙΣΤΗ 3-BIT

5.1.1 Δημιουργία Xilinx project

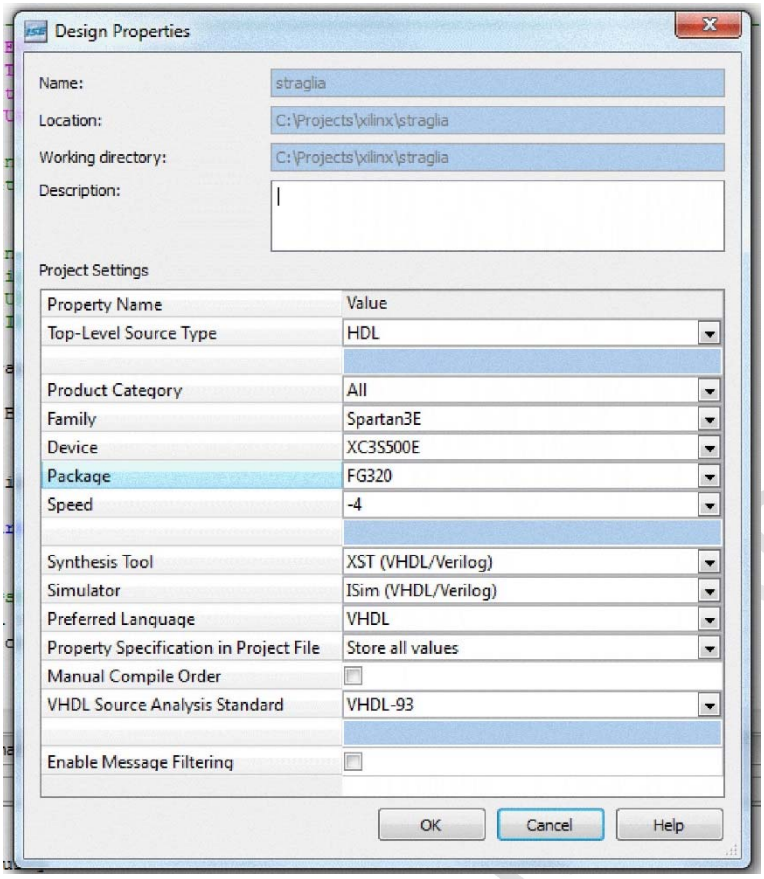
Ξεκινώντας το περιβάλλον Xilinx ISE δημιουργούμε ένα νέο project επιλέγοντας τις κατάλληλες ρυθμίσεις για την πλακέτα Xilinx Spartan 3E. Έτσι έχουμε

Start > Programs > Xilinx ISE > Project Navigator

Με την εκτέλεση του Xilinx ISE επιλέγουμε

File > New Project

Μετά την απόδοση κατάλληλου ονόματος στη νέα εφαρμογή μας επιλέγουμε στοιχεία πλακέτας και προσομοιωτή



Εικόνα 66. Xilinx ISE New project Design Properties

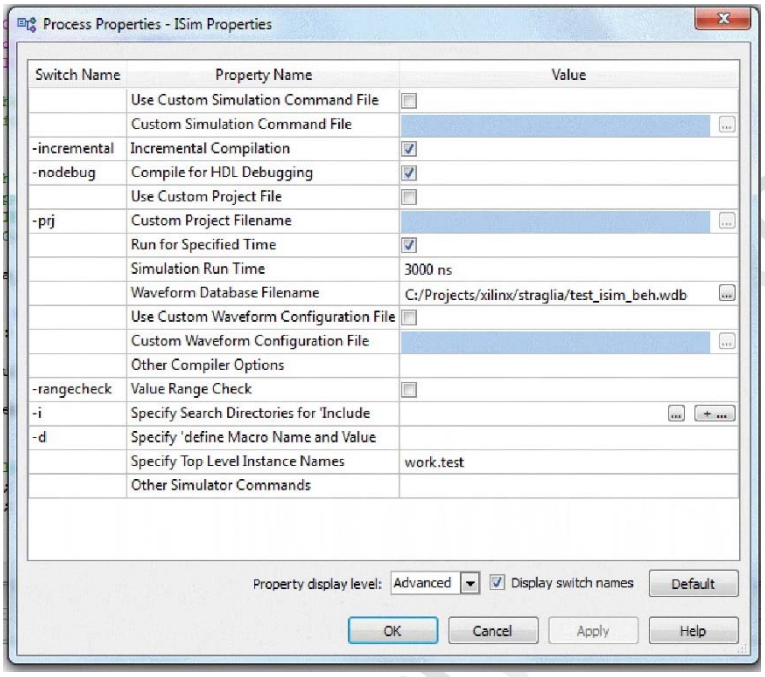
Οι επιλογές μπορούν να αλλάξουν αργότερα. Στα παραδείγματα που έχουμε υλοποιήσει κάναμε χρήση του προσομοιωτή ISim αντί για τον Modelsim. Μετά και την ολοκλήρωση των βασικών επιλογών της νέας εφαρμογής προσθέτουμε ένα αρχείο πηγαίου κώδικα. Τα κυκλώματα που δημιουργούμε στο περιβάλλον Xilinx ISE Project Navigator μπορούν να υλοποιηθούν τόσο με χρήση λογικών πυλών σε σχηματικό αρχείο όσο και με χρήση γλώσσας περιγραφής υλικού (VHDL κώδικα). Εδώ γίνεται χρήση της VHDL με την οποία και δημιουργούμε το αρχείο πηγαίου κώδικα “straglia_adder3bit.vhd”. Σε αυτό δημιουργούμε 3 λογικά διανύσματα των 3 bit, 2 εισόδου για τους τελεστέους και ένα εξόδου για το αποτέλεσμα.

```
entity straglia_adder3bit is
    port (
        A,B :  in std_logic_vector(0 to 2);
        c     : out std_logic_vector(0 to 2)
    );
end straglia_adder3bit;
```

Επιπλέον περιγράφουμε τη συμπεριφορά του κυκλώματός μας η οποία μας δίνει και το αποτέλεσμα.

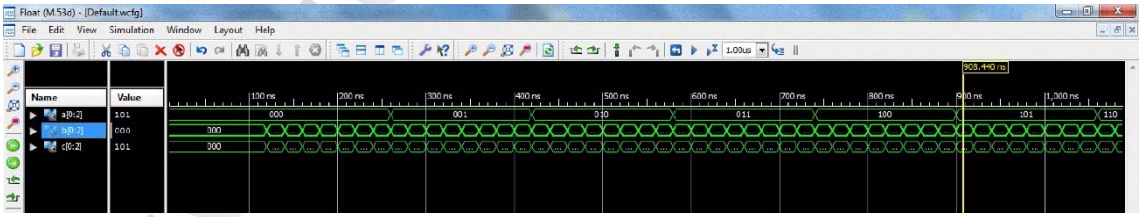
```
begin
        c <= A + B;
end Behavioral;
```

Για την λειτουργική προσομοίωση του κυκλώματος προσθέτουμε ένα ακόμα αρχείο πηγαίου κώδικα τύπου “VHDL Test Bench”. Με αυτό τον τρόπο και επιλέγοντας τον κατάλληλο προσομοιωτή από τις επιλογές του περιβάλλοντος Xilinx ISE επαληθεύουμε τη λογική του κυκλώματος σε πρώιμο στάδιο ώστε να αποφύγουμε λάθη σχεδιαστικά που θα προκύψουν αργότερα.



Εικόνα 67. Xilinx ISE Project Process Properties

Έτσι δημιουργούμε το αρχείο “test.vhd” το οποίο και εκτελείται μέσω του ISim όπως έχουμε επιλέξει νωρίτερα.



Εικόνα 68. ISim Simulator 3-bit Adder Execution

Στο εργαλείο ISim ελέγχουμε τη λογική συμπεριφορά του κυκλώματος καθώς μπορούμε σε κάθε χρονική στιγμή να δούμε τις τιμές των διανυσμάτων a, b που είναι η τελεστέοι και του διανύσματος c που είναι το αποτέλεσμα.

Το βασικό προτέρημα χρήσης του προσομοιωτή και του αρχείου “test.vhd” είναι το γεγονός ότι με κατάλληλες εντολές, καταγράφουμε των διανυσμάτων με τέτοιο τρόπο ώστε δημιουργούμε ένα αρχείο κειμένο (text) το οποίο με ελάχιστη επεξεργασία χρησιμοποιείται ως αρχείο pattern στο περιβάλλον Stylus του ελεγκτή Inovys Personal Ocelot για τον έλεγχο με αυτόν του υλοποιημένου ολοκληρωμένου κυκλώματος.

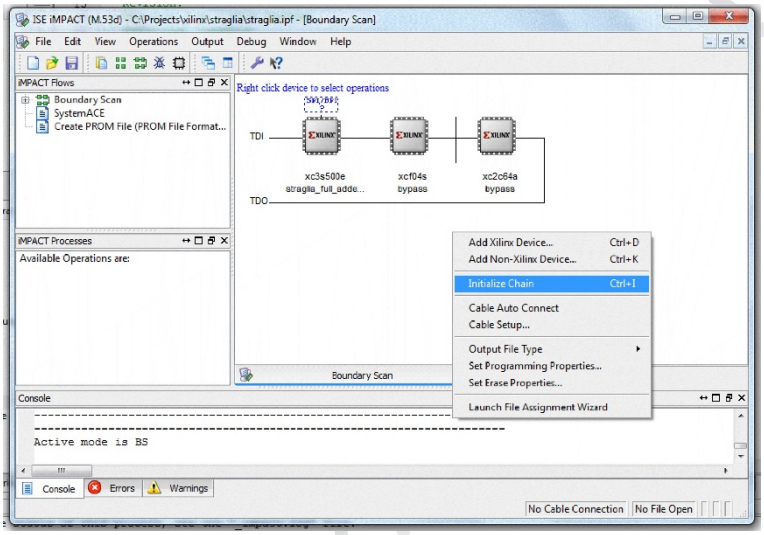
```
begin
        file_open(outfile, "lalala.txt",WRITE_MODE);
        write(outline, string'("Pattern OEspP{"));
        writeline(outfile, outline);
        write(outline, string'("W Wft0;"));
        writeline(outfile, outline);
      wait for 100 ns;

      for I in 0 to 7 loop
          for J in 0 to 7 loop
            A <= std_logic_vector(to_unsigned(I,A'length));
            B <= std_logic_vector(to_unsigned(J,A'length));
            wait for 20 ns;
            write(outline, string'("V{ALLPINS="));
            write(outline, A);
            write(outline, B);
            if (c(0) = '0') then
                write(outline, string'("L"));
            else
                write(outline, string'("H"));
            end if;
            if (c(1) = '0') then
                write(outline, string'("L"));
            else
                write(outline, string'("H"));
            end if;
            if (c(2) = '0') then
                write(outline, string'("L"));
            else
                write(outline, string'("H"));
            end if;
            write(outline, string'(";}"));
            writeline(outfile, outline);
          end loop;
      end loop;

      write(outline, string'("}"));
      writeline(outfile, outline);
      file_close (outfile);
```

Σε περίπτωση που το δημιουργημένο αρχείο δεν περιλαμβάνει όλους τους πιθανούς συνδυασμούς όπως του έχουμε ορίσει αυξάνουμε το χρόνο προσομοίωσης στο μενού της εικόνα 60.

Επόμενο βήμα είναι η εκτέλεση της επιλογής “Synthesize-XST” όπου γίνεται και η σύνθεση του κυκλώματος. Κάθε βήμα στο περιβάλλον Xilinx ISE θα πρέπει να ολοκληρώνεται με

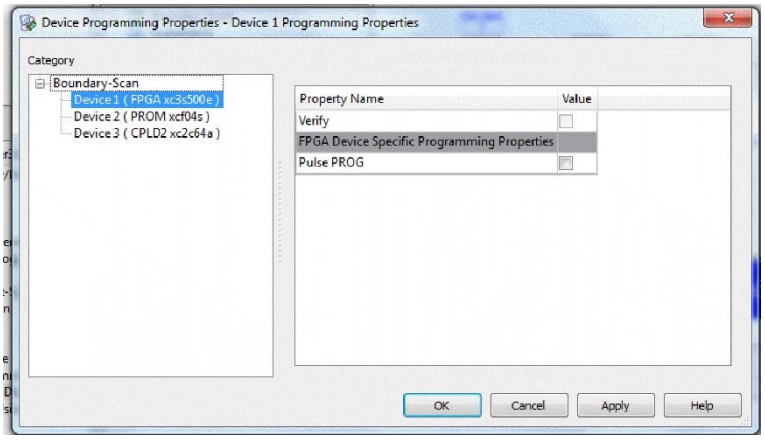
την ανάλογη σήμανση επιτυχίας στο τέλος και χωρίς την εμφάνιση “Errors” στο ανάλογο παράθυρο του Xilin ISE. Επόμενο βήμα είναι η υλοποίηση της σχεδίασης του κυκλώματος. Στο σημείο αυτό προσθέτουμε ακόμα ένα αρχείο πηγαίου κώδικα τύπου “Implementation Contstraints File” (*.UCF). Σε αυτό αναγράφεται η απεικόνιση κάθε θύρας υψηλότερου επιπέδου σχεδίασης στους φυσικούς ακροδέκτες του FPGA στο οποίο υλοποιείται το κύκλωμα μας. Εδώ είτε επεξεργαζόμαστε το αρχείο σαν αρχείο κειμένου είτε κάνουμε χρήση του εργαλείου “Plan Ahead” επιλέγοντας “I/O Planning – Post Synthesis”. Έπειτα εκτελούμε την επιλογή “Implement Design” η οποία και πρέπει να ολοκληρωθεί δίχως σφάλματα.

Μετά και την επιτυχή ολοκλήρωση των παραπάνω επιλέγουμε “Generate Programming File” με το οποίο και δημιουργούμε το αρχείο προγραμματισμού του FPGA ολοκληρωμένου. Πλέον είμαστε έτοιμοι για τον προγραμματισμό της πλακέτας Xilinx Spartan 3E. Αρχικά συνδέουμε την πλακέτα με την τροφοδοσία και το USB καλώδιο στο PC στο οποίο έχουμε ετοιμάσει την εφαρμογή.Για το κατέβασμα του αρχείου στο FPGA ολοκληρωμένο επιλέγουμε “Configure Target Device” το οποίο και φορτώνει το πρόγραμμα ISE iMPACT. Εδώ με δεξί κλικ επιλέγουμε “Initialize Chain”.



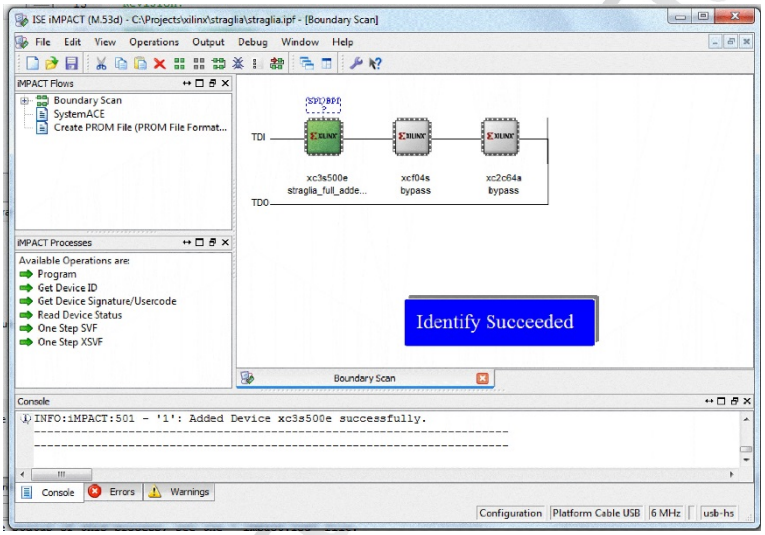
Εικόνα 69. ISE iMPACT Initialize Chain

Στο αναδυόμενο παράθυρο επιλέγουμε το “.bit” αρχείο που έχει δημιουργηθεί στο φάκελο της εφαρμογής μας κάνωντας bypass τις επιλογές 2 που ακολουθούν. Τέλος φτάνωντας στο παρκάτω μενού επιλέγουμε “OK”



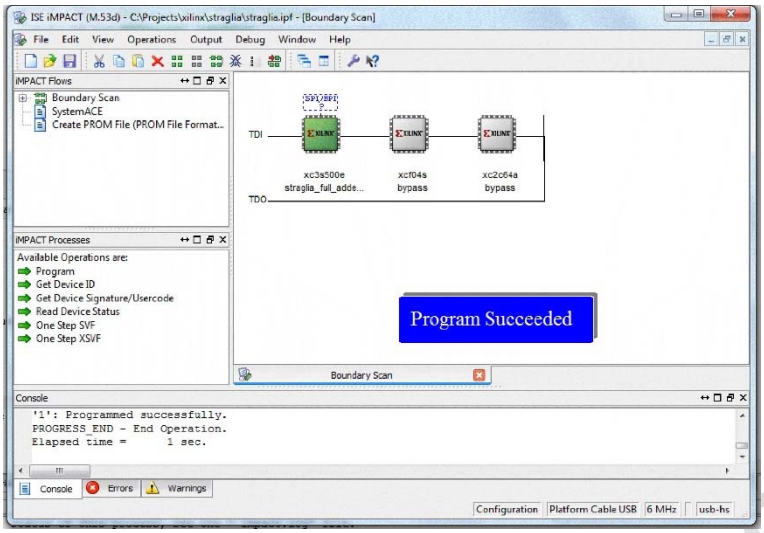
Εικόνα 70. ISE iMPACT OK

Το περιβάλλον μας ενημερώνει ότι η αναγνώριση του ολοκληρωμένου ήταν επιτυχής



Εικόνα 71. ISE iMPACT Identification Successful

Για την ολοκλήρωση της ενέργειας πατάμε δεξί κλικ και “Program” και θα πρέπει να καταλήγουμε στην παρακάτω εικόνα που σημαίνει πως η διαδικασία ήταν επιτυχής



Εικόνα 72. ISE iMPACT Programming Successful

Αξίζει να σημειωθεί πως το περιβάλλον Xilinx ISE Project Navigator θα πρέπει να παραμένει ανοιχτό κατά τη διάρκεια δημιουργίας του προγράμματος ελέγχου για να παραμένει το κύκλωμα ενεργό στη πλακέτα Xilinx Spartan 3E. Εναλλακτικά υπάρχει δυνατότητα προγραμματισμού τη μνήμης της πλακέτας Xilinx Spartan 3E ώστε να λειτουργεί με βάση το προγραμματισμένο κύκλωμα με την παροχή τροφοδοσίας στην πλακέτα.

5.1.2 Παράθεση αρχείων VHDL κυκλώματος αθροιστή 3-Bit

Straglia_adder3bit.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.NUMERIC_STD.ALL;

entity straglia_adder3bit is
    port (
        A,B :  in std_logic_vector(0 to 2);
        c   : out std_logic_vector(0 to 2)
    );
end straglia_adder3bit;

architecture Behavioral of straglia_adder3bit is

begin
    --c <=std_logic_vector(unsigned(A) + unsigned(B));
     c <= A + B;
end Behavioral;
```

Straglia.ucf

```
# PlanAhead Generated physical constraints

NET "A[0]" LOC = B4;
NET "A[1]" LOC = A4;
NET "A[2]" LOC = D5;
NET "B[0]" LOC = A6;
NET "B[1]" LOC = B6;
NET "B[2]" LOC = E7;
NET "c[0]" LOC = D7;
NET "c[1]" LOC = C7;
NET "c[2]" LOC = F8;
```

Test.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_textio.ALL;
USE std.textio.all;


ENTITY test IS
END test;

ARCHITECTURE behavior OF test IS
    COMPONENT straglia_adder3bit
    PORT(
         A : IN  std_logic_vector(0 to 2);
         B : IN  std_logic_vector(0 to 2);
         c : OUT  std_logic_vector(0 to 2)
        );
    END COMPONENT;


   --Inputs
   signal A : std_logic_vector(0 to 2) := (others => '0');
   signal B : std_logic_vector(0 to 2) := (others => '0');

    --Outputs
   signal c : std_logic_vector(0 to 2);
   -- No clocks detected in port list. Replace <clock> below with
   -- appropriate port name
         file outfile : TEXT;
```

```
BEGIN
   uut: straglia_adder3bit PORT MAP (
          A => A,
          B => B,
          c => c
        );
   stim_proc: process
        variable outline : LINE;
   begin
        file_open(outfile, "lalala.txt",WRITE_MODE);
        write(outline, string'("Pattern OEspP{"));
        writeline(outfile, outline);
        write(outline, string'("W Wft0;"));
        writeline(outfile, outline);
      wait for 100 ns;

      for I in 0 to 7 loop
          for J in 0 to 7 loop
            A <= std_logic_vector(to_unsigned(I,A'length));
            B <= std_logic_vector(to_unsigned(J,A'length));
            wait for 20 ns;
            write(outline, string'("V{ALLPINS="));
            write(outline, A);
            write(outline, B);
            if (c(0) = '0') then
                write(outline, string'("L"));
            else
                write(outline, string'("H"));
            end if;
            if (c(1) = '0') then
                write(outline, string'("L"));
            else
                write(outline, string'("H"));
            end if;
            if (c(2) = '0') then
                write(outline, string'("L"));
            else
                write(outline, string'("H"));
            end if;
            write(outline, string'(";}"));
            writeline(outfile, outline);
          end loop;
      end loop;
```

```
        write(outline, string'("}"));
        writeline(outfile, outline);
        file_close (outfile);
        wait;
    end process;

END;
```

5.2 ΥΛΟΠΟΙΗΣΗ ΚΥΚΛΩΜΑΤΟΣ MULTIPLIER

5.2.1 Δημιουργία Xilinx project

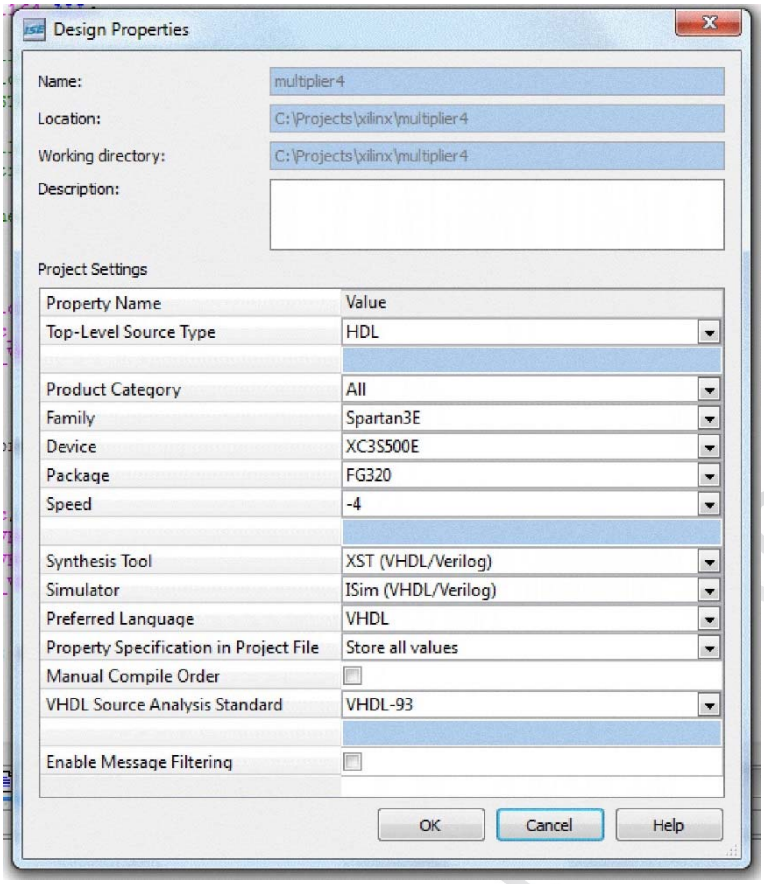
Ακολουθώντας την ίδια διαδικασία δημιουργίας νέας εφαρμογής, ξεκινάμε το περιβάλλον Xilinx ISE και δημιουργούμε ένα νέο project επιλέγοντας τις κατάλληλες ρυθμίσεις για την πλακέτα Xilinx Spartan 3E. Έτσι έχουμε

Start > Programs > Xilinx ISE > Project Navigator

Με την εκτέλεση του Xilinx ISE επιλέγουμε

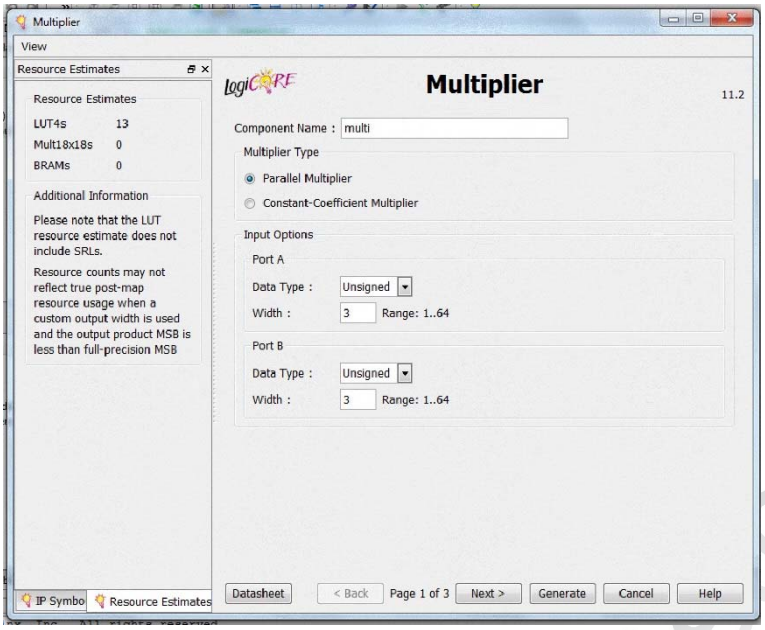
File > New Project

Μετά την απόδοση κατάλληλου ονόματος στη νέα εφαρμογή μας επιλέγουμε στοιχεία πλακέτας και προσομοιωτή



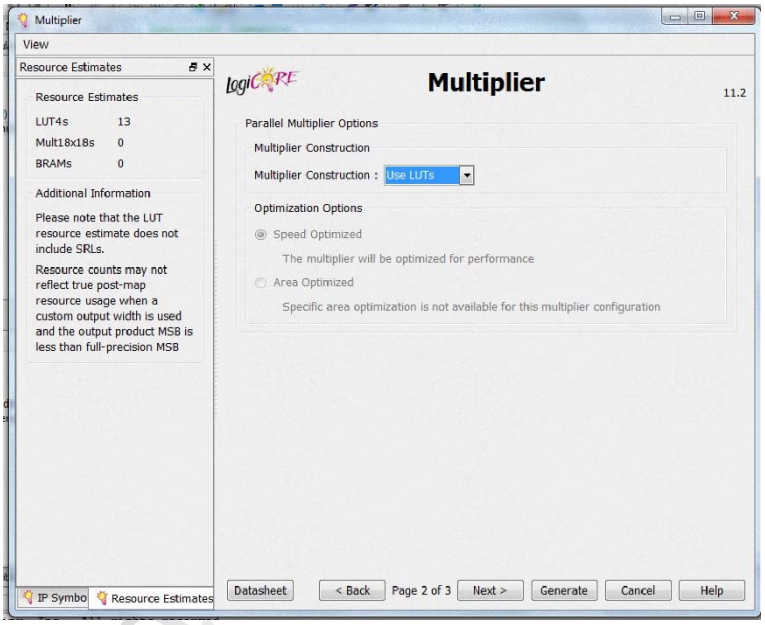
Εικόνα 73. Xilinx ISE New project Design Properties

Για την υλοποίηση του κυκλώματος του πολλαπλασιαστή γίνεται χρήση του εργαλείου “Core Generator” του περιβάλλοντος Xilinx ISE Project Navigator. Έτσι καλούμε στην εφαρμογή έναν από τους έτοιμους πυρήνες (core) του εργαλείου ρυθμίζοντας τις απαραίτητες επιλογές ανάλογα με την εφαρμογή μας. Αρχικά ονοματίζουμε τον πολλαπλασιαστή και επιλέγουμε τον τύπο του. Στην παρούσα εφαρμογή καθώς θέλουμε να περιλαμβάνει δύο τελεστέους επιλέγουμε παράλληλο πολλαπλασιαστή (Parallel) επιλέγοντας ταυτόχρονα και το πλάτος του κάθε τελεστέου και το εάν θα είναι προσημασμένος ή όχι (για την εφαρμογή και λόγω του περιορισμού των ακροδεκτών εισόδου – εξόδου επιλέγουμε 3-bit unsigned input options).



Εικόνα 74. Xilinx ISE Core Generator Tool, Multiplier Creation 1/3

Στην επόμενη καρτέλα του εργαλείου “Core Generator” επιλέγουμε τη χρήση “look up tables” (LUTs) αντί για dedicated multipliers.



Εικόνα 75. Xilinx ISE Core Generator Tool, Multiplier Creation 2/3

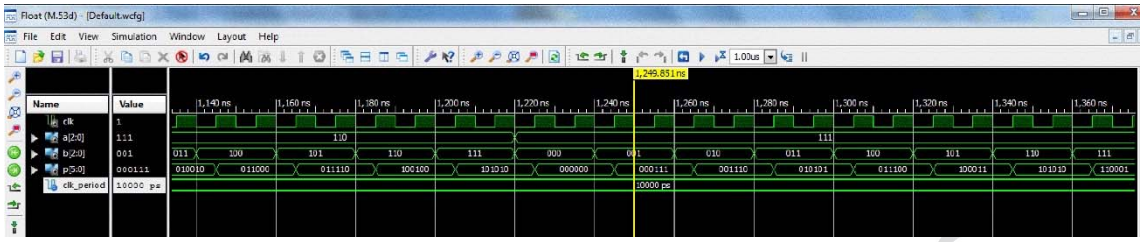
Στην τελευταία καρτέλα του εργαλείου “Core Generator” επιλέγουμε το πλάτος του αποτελέσματος, τη χρήση σταδίων “pipeline” καθώς και εάν επιθυμούμε τη χρήση των ακροδεκτών “Clock Enable” και “Synchronous Clear”. Λόγω των προαναφερθέντων περιορισμών επιλέγουμε να μην χρησιμοποιήσουμε τα δύο επιπλέον σήματα. Επίσης αν για λόγους ολοκληρωμένης δημιουργίας του πολλαπλασιαστή δεν επηρεάζουμε το πλάτος του αποτελέσματος της πράξης παρόλο που θα αγνοήσουμε το ένα bit Στην εφαρμογή μας.

Εικόνα 76. Xilinx ISE Core Generator Tool, Multiplier Creation 3/3

Επιλέγοντας “Generate” το εργαλείο ολοκληρώνεται προσθέτοντας ένα “Multiplier Core” στην εφαρμογή μας. Επόμενο βήμα είναι η προσθήκη του απαραίτητου κώδικα για την αρχικοποίηση του πυρήνα του πολλαπλασιαστή. Έτσι έχουμε

```
architecture Behavioral of top is
component multi
      port (
      clk: IN std_logic;
      a: IN std_logic_VECTOR(2 downto 0);
      b: IN std_logic_VECTOR(2 downto 0);
      p: OUT std_logic_VECTOR(5 downto 0));
end component;
begin
your_instance_name : multi
            port map (
                  clk => clk,
                  a => a,
                  b => b,
                  p => p);

end Behavioral;
```

Για τη λειτουργική προσομοίωση του κυκλώματος μας προσθέτουμε ένα αρχείο πηγαίου κώδικα τύπου “Test Bench”. Κατά την εκτέλεση της προσομοίωσης του με χρήση του προσομοιωτή “ISim” παίρνουμε ως αποτέλεσμα την παρακάτω απεικόνιση των σημάτων του πολλαπλασιαστή προς υλοποίηση.



Εικόνα 77. ISim Simulator Multiplier Execution

Παρακάτω παρατίθεται το ολοκληρωμένο αρχείο πηγαίου κώδικα “test.vhd” για την εφαρμογή του πολλαπλασιαστή. Εδώ ο αναγνώστης μπορεί να παρατηρήσει και τις απαραίτητες εντολές για τη δημιουργία του αρχείου κειμένου που θα αποτελέσει τη βάση δημιουργίας του αρχείου “.spat” (pattern) για την εφαρμογή του ελεγκτή.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_textio.ALL;
USE std.textio.all;

ENTITY test IS
END test;

ARCHITECTURE behavior OF test IS
     COMPONENT top
    PORT(
         clk : IN  std_logic;
         a : IN  std_logic_vector(2 downto 0);
         b : IN  std_logic_vector(2 downto 0);
         p : OUT  std_logic_vector(5 downto 0)
        );
    END COMPONENT;

   signal clk : std_logic := '0';
   signal a : std_logic_vector(2 downto 0) := (others => '0');
   signal b : std_logic_vector(2 downto 0) := (others => '0');
   signal p : std_logic_vector(5 downto 0);
   constant clk_period : time := 10 ns;
         file outfile : TEXT;
BEGIN

   uut: top PORT MAP (
         clk => clk,
         a => a,
         b => b,
         p => p
```

```
     );
clk_process :process
begin
         clk <= '0';
         wait for clk_period/2;
         clk <= '1';
         wait for clk_period/2;
end process;

stim_proc: process
           variable outline : LINE;
begin
     file_open(outfile, "lalala.txt",WRITE_MODE);
     write(outline, string'("Pattern OEspP{"));
     writeline(outfile, outline);
     write(outline, string'("W Wft0;"));
     writeline(outfile, outline);
   wait for 100 ns;

   for I in 0 to 7 loop
       for J in 0 to 7 loop
         A <= std_logic_vector(to_unsigned(I,A'length));
         B <= std_logic_vector(to_unsigned(J,A'length));
         wait for 10 ns;
         write(outline, string'("V{ALLPINS=k"));
         write(outline, A);
         write(outline, B);
                    wait for 10 ns;
         if (p(4) = '0') then
             write(outline, string'("L"));
         else
             write(outline, string'("H"));
         end if;
         if (p(3) = '0') then
             write(outline, string'("L"));
         else
             write(outline, string'("H"));
         end if;
         if (p(2) = '0') then
             write(outline, string'("L"));
         else
             write(outline, string'("H"));
         end if;
         if (p(1) = '0') then
```

```
                write(outline, string'("L"));
            else
                write(outline, string'("H"));
            end if;
            if (p(0) = '0') then
                write(outline, string'("L"));
            else
                write(outline, string'("H"));
            end if;
            write(outline, string'(";}"));
            writeline(outfile, outline);
          end loop;
      end loop;

      write(outline, string'("}"));
      writeline(outfile, outline);
      file_close (outfile);

      wait;
   end process;

END;
```

Επόμενο βήμα είναι η εκτέλεση της επιλογής “Synthesize-XST” όπου γίνεται και η σύνθεση του κυκλώματος. Κάθε βήμα στο περιβάλλον Xilinx ISE θα πρέπει να ολοκληρώνεται με την ανάλογη σήμανση επιτυχίας στο τέλος και χωρίς την εμφάνιση “Errors” στο ανάλογο παράθυρο του Xilin ISE. Επόμενο βήμα είναι η υλοποίηση της σχεδίασης του κυκλώματος. Στο σημείο αυτό προσθέτουμε ακόμα ένα αρχείο πηγαίου κώδικα τύπου “Implementation Contstraints File” (*.UCF). Σε αυτό αναγράφεται η απεικόνιση κάθε θύρας υψηλότερου επιπέδου σχεδίασης στους φυσικούς ακροδέκτες του FPGA στο οποίο υλοποιείται το κύκλωμα μας. Εδώ είτε επεξεργαζόμαστε το αρχείο σαν αρχείο κειμένου είτε κάνουμε χρήση του εργαλείου “Plan Ahead” επιλέγοντας “I/O Planning – Post Synthesis”. Στην εφαρμογή του πολλαπλασιαστή σε σχέση με αυτή του αθροιστή υπάρχει μια βασική διαφορά. Το ολοκληρωμένο κύκλωμα χρειάζεται σήμα ρολογιού το οποίο και δίνεται εξωτερικά από τον ελεγκτή Inovys Personal Ocelot. Για την ορθή χρήση αυτού του σήματος ο χρήστης οφείλει να προσθέσει μια εντολή στο πηγαίο αρχείο “UCF” ειδάλλως το περιβάλλον Xilinx ISE παράγει μήνυμα σφάλματος. Έτσι έχουμε την προσθήκη της εντολής

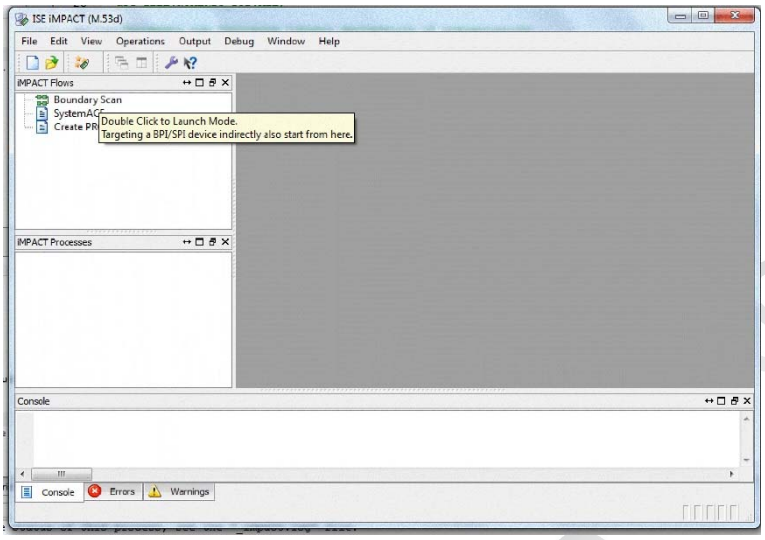
```
NET "clk" CLOCK_DEDICATED_ROUTE = FALSE;
```

Επεξεργαζόμενοι το αρχείο σαν αρχείο κειμένου.

Έπειτα εκτελούμε την επιλογή “Implement Design” η οποία και πρέπει να ολοκληρωθεί δίχως σφάλματα.

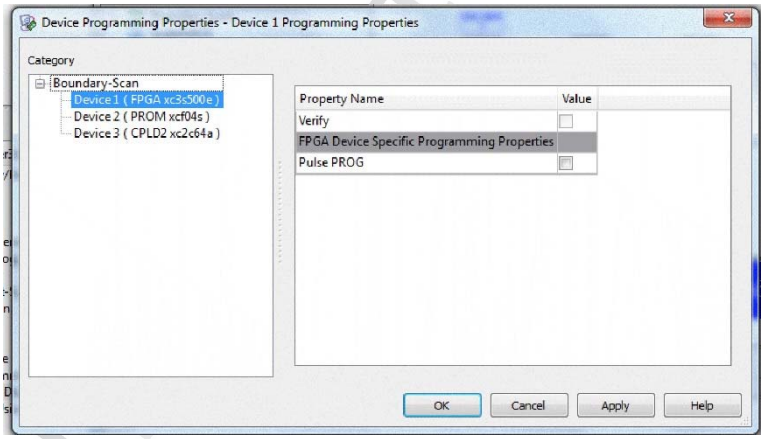
Μετά και την επιτυχή ολοκλήρωση των παραπάνω επιλέγουμε “Generate Programming File” με το οποίο και δημιουργούμε το αρχείο προγραμματισμού του FPGA ολοκληρωμένου. Πλέον είμαστε έτοιμοι για τον προγραμματισμό της πλακέτας Xilinx Spartan 3E. Αρχικά

συνδέουμε την πλακέτα με την τροφοδοσία και το USB καλώδιο στο PC στο οποίο έχουμε ετοιμάσει την εφαρμογή.Για το κατέβασμα του αρχείου στο FPGA ολοκληρωμένο επιλέγουμε “Configure Target Device” το οποίο και φορτώνει το πρόγραμμα ISE iMPACT. Εδώ αρχικά κάνουμε διπλό κλικ στην επιλογή “Boundary Scan” και έπειτα με δεξί κλικ επιλέγουμε “Initialize Chain”.



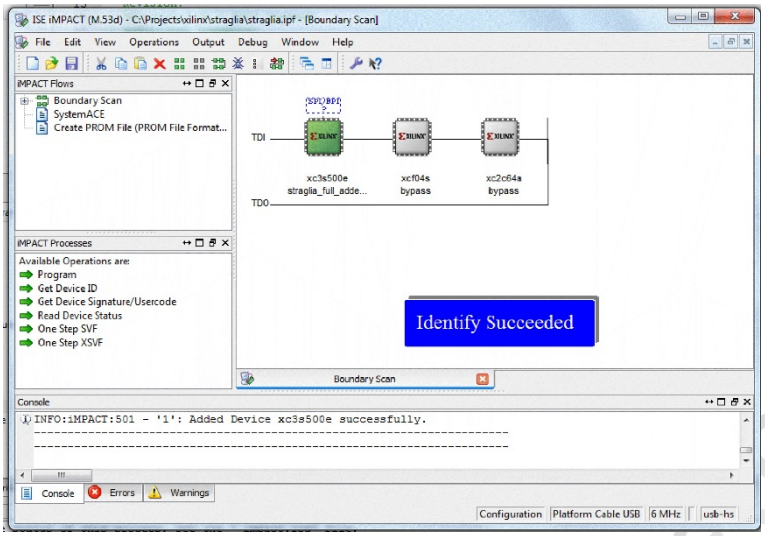
Εικόνα 78. ISE iMPACT Initialize Chain

Στο αναδυόμενο παράθυρο επιλέγουμε το “.bit” αρχείο που έχει δημιουργηθεί στο φάκελο της εφαρμογής μας κάνωντας bypass τις επιλογές 2 που ακολουθούν. Τέλος φτάνωντας στο παρκάτω μενού επιλέγουμε “OK”



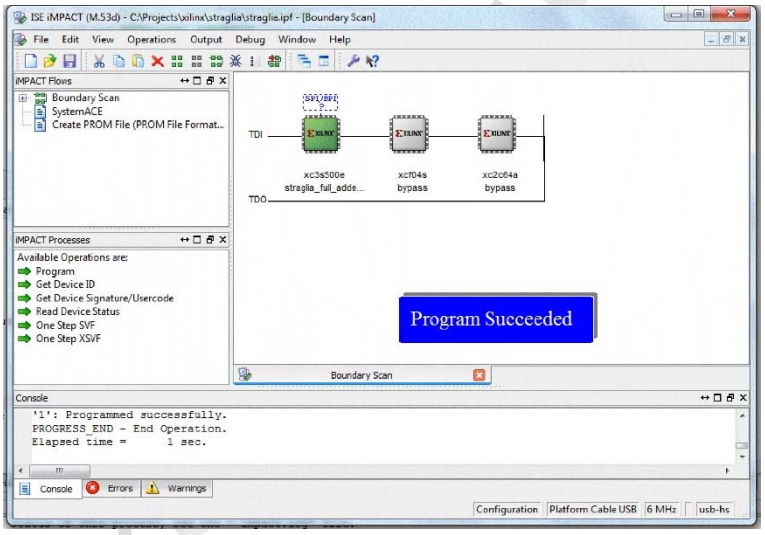
Εικόνα 79. ISE iMPACT OK

Το περιβάλλον μας ενημερώνει ότι η αναγνώριση του ολοκληρωμένου ήταν επιτυχής



Εικόνα 80. ISE iMPACT Identification Successful

Για την ολοκλήρωση της ενέργειας πατάμε δεξί κλικ και “Program” και θα πρέπει να καταλήγουμε στην παρακάτω εικόνα που σημαίνει πως η διαδικασία ήταν επιτυχής



Εικόνα 81. ISE iMPACT Programming Successful

Αξίζει να σημειωθεί πως το περιβάλλον Xilinx ISE Project Navigator θα πρέπει να παραμένει ανοιχτό κατά τη διάρκεια δημιουργίας του προγράμματος ελέγχου για να παραμένει το κύκλωμα ενεργό στη πλακέτα Xilinx Spartan 3E. Εναλλακτικά υπάρχει δυνατότητα προγραμματισμού τη μνήμης της πλακέτας Xilinx Spartan 3E ώστε να λειτουργεί με βάση το προγραμματισμένο κύκλωμα με την παροχή τροφοδοσίας στην πλακέτα.

5.2.2 Παράθεση αρχείων VHDL κυκλώματος Multiplier

Top.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity top is
port (
      clk : in std_logic;
      a,b: IN std_logic_VECTOR(2 downto 0);
      p: OUT std_logic_VECTOR(5 downto 0));

end top;

architecture Behavioral of top is
component multi
      port (
      clk: IN std_logic;
      a: IN std_logic_VECTOR(2 downto 0);
      b: IN std_logic_VECTOR(2 downto 0);
      p: OUT std_logic_VECTOR(5 downto 0));
end component;
begin
your_instance_name : multi
            port map (
                  clk => clk,
                  a => a,
                  b => b,
                  p => p);

end Behavioral;
```

koko.ucf

```
# PlanAhead Generated physical constraints
NET "clk" CLOCK_DEDICATED_ROUTE = FALSE;
NET "a[2]" LOC = A4;
NET "a[1]" LOC = D5;
NET "a[0]" LOC = C5;
NET "b[2]" LOC = A6;
NET "b[1]" LOC = B6;
NET "b[0]" LOC = E7;
NET "clk" LOC = B4;
NET "p[5]" LOC = F12;
NET "p[4]" LOC = F7;
NET "p[3]" LOC = D7;
NET "p[2]" LOC = C7;
```

```
NET "p[1]" LOC = F8;
NET "p[0]" LOC = E8;
```

Test.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_textio.ALL;
USE std.textio.all;

ENTITY test IS
END test;
ARCHITECTURE behavior OF test IS
    COMPONENT top
    PORT(
         clk : IN  std_logic;
         a : IN  std_logic_vector(2 downto 0);
         b : IN  std_logic_vector(2 downto 0);
         p : OUT  std_logic_vector(5 downto 0)
        );
    END COMPONENT;
   signal clk : std_logic := '0';
   signal a : std_logic_vector(2 downto 0) := (others => '0');
   signal b : std_logic_vector(2 downto 0) := (others => '0');
   signal p : std_logic_vector(5 downto 0);
   constant clk_period : time := 10 ns;
          file outfile : TEXT;
BEGIN
   uut: top PORT MAP (
          clk => clk,
          a => a,
          b => b,
          p => p
        );
   clk_process :process
   begin
            clk <= '0';
            wait for clk_period/2;
            clk <= '1';
            wait for clk_period/2;
   end process;
   stim_proc: process
              variable outline : LINE;
   begin
```

```
  file_open(outfile, "lalala.txt",WRITE_MODE);
  write(outline, string'("Pattern OEspP{"));
  writeline(outfile, outline);
  write(outline, string'("W Wft0;"));
  writeline(outfile, outline);
-- hold reset state for 100 ns.
wait for 100 ns;
for I in 0 to 7 loop
    for J in 0 to 7 loop
      A <= std_logic_vector(to_unsigned(I,A'length));
      B <= std_logic_vector(to_unsigned(J,A'length));
      wait for 10 ns;
      write(outline, string'("V{ALLPINS=k"));
      write(outline, A);
      write(outline, B);
                 wait for 10 ns;
      if (p(4) = '0') then
          write(outline, string'("L"));
      else
          write(outline, string'("H"));
      end if;
      if (p(3) = '0') then
          write(outline, string'("L"));
      else
          write(outline, string'("H"));
      end if;
      if (p(2) = '0') then
          write(outline, string'("L"));
      else
          write(outline, string'("H"));
      end if;
      if (p(1) = '0') then
          write(outline, string'("L"));
      else
          write(outline, string'("H"));
      end if;
      if (p(0) = '0') then
          write(outline, string'("L"));
      else
          write(outline, string'("H"));
      end if;
      write(outline, string'(";}"));
      writeline(outfile, outline);
    end loop;
```

```
            end loop;

            write(outline, string'("}"));
            writeline(outfile, outline);
            file_close (outfile);
            wait;
         end process;

      END;
```

6. Επίλογος

Το πρότυπο IEEE Std. 1450 STIL (Standard Test Interface Language) προήλθε μετά από συνεργατική προσπάθεια μεταξύ των εταιριών IBM και Motorola κατά την ανάπτυξη του επεξεργαστή Power PC. Οι δύο εταιρίες κατάφεραν να ξεκινήσουν μια προσπάθεια δημιουργίας η οποία αρκετά χρόνια μετά απέφερε το πρότυπο αυτό το οποίο και διευκολύνει τις προσπάθειες των εταιριών κατασκευής εργαλείων αυτόματου ελέγχου, τους χρήστες τέτοιων εργαλείων, των εταιριών ανάπτυξης ολοκληρωμένων κυκλωμάτων καθώς και των υπολοίπων εμπλεκομένων σε διεργασίες σχετικές με τα παραπάνω. Η παρούσα εργασία είχε σκοπό την εξοικείωση του αναγνώστη με το πρότυπο, τη σύνταξη, τη δομή και τη λειτουργία του. Φυσικά δεν μπορούν αν καλυφθούν όλες οι παράμετροι ενός προτύπου με ένα οδηγό εκμάθησης και περαιτέρω εμβάθυνση επαφίεται στον κάθε πιθανό χρήστη του προτύπου ατομικά. Στη συνέχεια παρουσιάστηκε το εργαλείο που μας παρέχεται από το Πανεπιστήμιο Πειραιά , Inovys Personal Ocelot, τις δυνατότητες του και το περιβάλλον λειτουργίας και χρήσης του Stylus. Ο αναγνώστης κάνει μια πρώτη γνωριμία με τον εν λόγω ελεγκτή ολοκληρωμένων κυκλωμάτων βλέποντας τις δυνατότητες του και τα βασικά μενού του περιβάλλοντος χρήσης του. Μετά από ένα σύντομο οδηγό δημιουργίας μιας απλής εφαρμογής ελέγχου οδηγηθήκαμε στην παρουσίαση πραγματικού ελέγχου ολοκληρωμένων κυκλωμάτων. Αυτά υλοποιήθηκαν με σε περιβάλλον Xilinx ISE με χρήση γλώσσας περιγραφής υλικού (VHDL) σε FPGA πλακέτα Xilinx Spartan 3E. Η ταχύτητα και αμεσότητα ενός hardware ολοκληρωμένου κυκλώματος σε FPGA θεωρήθηκε ότι κοντινότερο μπορούσε να υλοποιηθεί σε πραγματική εφαρμογή για την πληρέστερη και αναλυτικότερη παρουσίαση του εργαλείου Stylus. Απώτερος σκοπός άλλωστε της παρούσης ήταν η ενεργοποίηση του ελεγκτή και η χρήση του προτύπου IEEE Std. 1450 STIL.

Στα υλοποιημένα κυκλώματα υπάρχει αρκετός χώρος για περαιτέρω πειραματισμό και εμβάθυνση τόσο από πλευράς πολυπλοκότητας όσο και από πλευράς παραγόμενου αποτελέσματος πληροφορίας και γνώσης. Τόσο η λεπτομερής ανάλυση των χρονικών στοιχείων των κυκλωμάτων όσο και περαιτέρω ανάλυση των χρονικών ορίων είναι κάτι που μπορεί να αποτελέσει από μόνο του θέμα μελλοντικής ανάλυσης.

Για την διευκόλυνση της χρήσης του ελεγκτή και της σύνδεσης ολοκληρωμένων κυκλωμάτων σε αυτόν δημιουργήθηκε και ένα τυπωμένο κύκλωμα (Inovys Ocelot IO Board) με βάση τις τεχνικές προδιαγραφές του ελεγκτή και των υπαρχόντων πλακετών φόρτωσης τυπωμένων κυκλωμάτων (Load Board). Η δυνατότητα δημιουργίας πιο περίπλοκών πλακετών υποστήριξης έγγυται στις δυνατότητες των μελλοντικών χρηστών.

Με την παρούσα προσπάθεια ανοίγει ο δρόμος χρήσης του ελεγκτή Inovys Personal Ocelot. Έγινε γνωριμία με τις βασικές δυνατότητες ελέγχου μα εύκολα γίνεται κατανοητό πως ο ελεγκτής μπορεί να παρέχει στο χρήστη ακόμα περισσότερες δυνατότητες ελέγχου και πειραματισμού με το συνδυασμό χρήσης ενός δυνατού εργαλείου και ενός προτύπου δημιουργημένου για την μεγιστοποίηση του οφέλους χρήσης του ελεγκτή αυτού.

Παράρτημα 1

IEEE Std. 1450 STIL (Standard Test Interface Language) Backus-Naur Form

Introductory STIL BNF

Document Version 1.1, December 4, 1998. Revisions will change this version.

The following BNF representation of STIL is available here to provide an introductionto STIL. It

is not a complete representation, but rather intended as an overview to the primary structural

elements of the language. In the interests of simplification, this BNF does not represent ordering

requirements, or identify multiplicity issues on statements. Please be aware when referencing this

BNF that it is also incomplete with respect to the detailed semantics of the language. For a complete representation please review the P1450 document.

The meta-symbols of BNF are:

::= meaning "is defined as"

| meaning "or"

optional items are enclosed in meta symbols "[" and "]"

terminals are distinguished by using bold face type

terminals of only one character are surrounded by quotes (") to distinguish them from

meta-symbols

This BNF is a representation of the allowed syntax of the language. It will be revised as different

ways to "look at" this information are considered. For a complete definition of STIL please refer

to the P1450 document.

Please remember this BNF is considered an incomplete representation of the language.

1.0 STIL Organization

```
stil_session ::= stil [header] session
session ::= block
| session block
block ::=  user_keywords
| user_functions
| signals
| signal_groups
| pattern_exec
| pattern_burst
| timing
| spec
| selector
| scan_structs
```

```
  | pattern
  | procedures
  | macro_defs
  | include | annotation | udb | (null)
  2.0 STIL Statement
  stil ::= STILstil_version_number ";"
  stil_version_number ::= integer "." integer
  3.0 Header Block
  header ::= Header"{" [header_list] "}"
  header_list ::= header_item
  | header_list header_item
  header_item ::= Titlestring ";"
  | Date date_string ";"
  | Sourcestring ";"
  | History"{" [ history_list ]"}"
  | include | annotation | udb | (null)
  date_string ::= """weekday month day_of_month time year """
  weekday ::= Mon | Tue | Wed | Thu | Fri | Sat | Sun
  month ::= Jan | Feb | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct |
Nov | Dec
  day_of_month ::= digit digit
  time ::= hour ":" minute ":" second
  hour ::= digit digit
  minute ::= digit digit
  second ::= digit digit
  year ::= digit digit digit digit
  history_list ::= annotation
  | history_list annotation
  4.0 Include Statement
  include ::= Includefile_name [IfNeedblocktype ] ";"
  stil [header]
  (Note: The STIL and optional header statements are the first
statements in an included file. All
  subsequent statements are in the context of the "Include" statement
in the including file)
  blocktype ::= Include
  | Header
  | UserKeywords
  | UserFunctions
  | Signals
  | SignalGroups
  | PattenExec
  | PatternBurst
  | Timing
  | Spec
```

```
   | Selector
   | ScanStructures
   | Pattern
   | Procedures
   | MacroDefs
   | Ann
   file_name ::= identifier
   5.0 UserKeywords Statement
   user_keywords ::= UserKeywordsuser_defined_keywords ";"
   user_defined_keywords ::= identifier
   | user_defined_keywords identifier
   udb ::=  identifier "{"udb_text "}" (Note: allowed identifiers must
first be declared in
   | identifier udb_2_text ";" the user_keywords stmt)
   udb_text ::=any sequence of characters, with the restriction that any
'{' be matched with '}'
   udb_2_text ::=any sequence of charactersexcept '{' and '}' and ';'
   6.0 UserFunctions Statement
   user_functions ::= UserFunctionsuser_defined_function ";"
   user_defined_function ::= identifier
   | user_defined_function identifier
   7.0 Ann Statement
   annotation ::= Ann"{*" ann_text "*}"
   ann_text ::=any sequence of characters except '*}'
   8.0 Signals Block
   signals ::= Signals"{" [ signals_list ] "}"
   signals_list ::= signals_item
   | signals_list signals_item
   signals_item ::= signal_name_array_opt signal_type ";"
   | signal_name_array_opt signal_type "{" [ sig_statements ] "}"
   | include | annotation | udb | (null)
   signal_name_array_opt ::= signal_name
   | identifier "[" integer ".." integer "]"
   signal_name ::= identifier
   | identifier "[" integer "]"
   signal_type ::=In
   | Out
   | InOut
   | Supply
   | Pseudo
   sig_statements ::=sig_statement
   | sig_statements sig_statement
   sig_statement := terminations
   | default_state_stmt
   | ScanIn[ integer ] ";"
```

```
  | ScanOut[ integer ] ";"
  | Basebase_type ";"
  | Alignmentorient_type ";"
  | DataBitCountinteger ";"
  | include | annotation | udb | (null)
  terminations ::= Terminationtermination_state ";"
  termination_state ::= TerminateHigh
  | TerminateLow
  | TerminateOff
  | TerminateUnknown
  default_state_stmt ::= DefaultStatedefault_state ";"
  default_state ::= "U" | ForceUp
  | "D" | ForceDown
  | "Z" | ForceOff
  base_type ::= Hexwfcs
  | Decwfcs
  orient_type ::= LSB
  | MSB
  9.0 SignalGroups Block
  signal_groups ::=SignalGroups [domain_name] "{" [groups_list ] "}"
  domain_name ::= identifier
  groups_list ::= groups_item
  | groups_list groups_item
  groups_item ::= group_name "=" sigref_expr ";"
  | group_name "=" sigref_expr "{" [ sig_statements ] "}"
  | include | annotation | udb | (null)
  group_name ::= identifier
  sigref_expr ::= signal_or_group_name
  | "'" grp_name_exp_list "'"
  grp_name_exp_list ::= signal_or_group_name
  | "(" grp_name_exp_list ")"
  | grp_name_exp_list plus_or_minus grp_name_exp_list
  signal_or_group_name ::= signal_name_array_opt(Note signal_name and
group_name may be
  | group_name identifiers; they are both here to indicate either ref.)
  plus_or_minus ::= "+" | "-"
  10.0 PatternExec Block
  pattern_exec ::= PatternExec[pat_exec_name] "{" [ pat_exec_list_items
] "}"
  pat_exec_name ::= identifier
  pat_exec_list_items ::= pat_exec_item
  | pat_exec_list_items pat_exec_item
  pat_exec_item ::= Timingtiming_name ";"
  | PatternBurstpat_burst_name ";"
  | Categorycategory_name ";"
```

```
  | Selectorselector_name ";"
  | include | annotation | udb | (null)
  category_name ::= identifier
  selector_name ::= identifier
  timing_name ::= identifier
  pat_burst_name ::= identifier
  11.0 Pattern Burst Block
  pattern_burst ::= PatternBurstpat_burst_name "{" [ pat_burst_stmnts ]
"}"
  pat_burst_name ::= identifier
  pat_burst_stmnts ::= pat_burst_stmnt
  | pat_burst_stmnts pat_burst_stmnt
  pat_burst_stmnt ::= SignalGroupsgroups_domain ";"
  | MacroDefsscan_macros_domain ";"
  | Proceduresprocedures_domain ";"
  | ScanStructuresscan_name ";"
  | Startpat_label ";"
  | Stoppat_label ";"
  | Termination"{" [ termination_statements ] "}"
  | PatList"{" pat_list_items "}"
  | include | annotation | udb | (null)
  pat_list_items ::= pat_list_item
  | pat_list_items pat_list_item
  pat_list_item ::= pat_name ";"
  | pat_name "{" [ pat_list_stmts ] "}"
  | pat_burst_name ";"
  | pat_burst_name "{" [ pat_list_stmts ] "}"
  pat_name ::=identifier
  pat_burst_name ::= identifier
  pat_list_stmts ::= pat_list_stmt
  | pat_list_stmts pat_list_stmt
  pat_list_stmt ::= SignalGroupsgroups_domain ";"
  | MacroDefsscan_macros_domain ";"
  | ScanStructuresscan_name ";"
  | Startpat_label ";"
  | Stoppat_label ";"
  | Proceduresprocedures_domain ";"
  | Termination"{" [ termination_statements ] "}"
  | include | annotation | udb | (null)
  groups_domain ::= identifier
  scan_macros_domain ::= identifier
  procedures_domain ::= identifier
  pat_label ::= identifier
  termination_statements ::= termination_statement
  | termination_statements termination_statement
```

```
  termination_statement ::= sigref_expr termination_state ";"
  12.0 Timing Block and WaveformTable Block
  timing ::= Timing[timing_label] "{" [ timing_list ] "}"
  timing_list ::= timing_item
  | timing_list timing_item
  timing_item ::= WaveformTablewft "{" wft_list "}"
  | SignalGroupsdomain_name ";"
  | include | annotation | udb | (null)
  wft ::= identifier
  timing_label ::=identifier
  cell ::= identifier
  wft_list ::= wft_item
  | wft_list wft_item
  wft_item ::= Period"'" time_expr "'" ";"
  | Waveforms"{" waveforms_list "}"
  | InheritWaveformTable[timing_label "."]wft ";"
  | SubWaveforms"{" subwaveforms_list "}"
  | include | annotation | udb | (null)
  waveforms_list ::= waveforms_item
  | waveforms_list waveforms_item
  waveforms_item ::= sigref_expr [label] "{" waveform_items "}"
  waveform_items ::= waveform_item
  | waveform_items waveform_item
  waveform_item ::= InheritWaveform[[timing_label "."]wft "."]cell ";"
  | wfc "{" wfc_def_list "}"
  | wfcs "{" wfcs_def_list "}"
  | include | annotation | udb | (null)
  subwaveforms_list ::= subwaveforms_item
  | subwaveforms_list subwaveforms_item
  subwaveforms_item ::= swf_label ":" Duration"'" time_expr "'" "{"
sub_def_list "}"
  | include | annotation | udb | (null)
  swf_label ::=identifier
  wfc_def_list ::= wfc_definition
  | wfc_def_list wfc_definition
  wfcs_def_list ::= wfcs_definition
  | wfcs_def_list wfcs_definition
  sub_def_list ::= sub_definition
  | sub_def_list sub_definition
  wfc_definition ::= [label ":"] "'" time_expr "'" event ";"
  | [label ":"] "'" time_expr "'" ";"
  | [label ":"] event ";"
  | [label ":"] ["'" time_expr "'"] [\rinteger] swf_label ";"
  | [label ":"] ["'" time_expr "'"] [\rinteger] swf_label "[" integer
"]" ";"
```

```
   | [label ":"] ["’" time_expr "’"] [\rinteger] swf_label "[" #"]" ";"
   | InheritWaveform[[[timing_label "."]wft "."]cell "."]wfc ";"
   | include | annotation | udb | (null)
  wfcs_definition ::= [label ":"] "’" time_expr "’" events ";"
   | [label ":"] "’" time_expr "’" events "[" integer "]" ";"
   | [label ":"] "’" time_expr "’" ";"
   | [label ":"] events ";"
   | [label ":"] events "[" integer "]" ";"
   | [label ":"] ["’" time_expr "’"] [\rinteger] swf_label ";"
   | [label ":"] ["’" time_expr "’"] [\rinteger] swf_label "[" integer
"]" ";"
   | [label ":"] ["’" time_expr "’"] [\rinteger] swf_label "[" #"]" ";"
   | InheritWaveform[[[timing_label "."]wft "."]cell "."]wfc ";"
   | include | annotation | udb | (null)
  sub_definition ::= "’" time_expr "’" events ";"
   | "’" time_expr "’" events "[" integer "]" ";"
   | "’" time_expr "’" ";"
   | events ";"
   | events "[" integer "]" ";"
   | label ":"
   | include | annotation | udb | (null)
  wfc ::=  letter
   | digit
   | "#"
   | "%"
  wfcs ::= wfc
   | wfcs wfc
  time_expr ::= time_expr "+" time_expr
   | time_expr "-" time_expr
   | time_expr "*" time_expr
   | time_expr "/" time_expr
   | "-" time_expr
   | "@" time_expr
   | function "(" [ function_args ] ")"
   | time_expr "==" time_expr
   | time_expr "<="time_expr
   | time_expr ">="time_expr
   | time_expr "<" time_expr
   | time_expr ">" time_expr
   | time_expr "!="time_expr
   | time_expr "?" time_expr ":" time_expr
   | "(" time_expr ")"
   | decimal
   | decimal engineering_units
   | ref_varname
```

```
  engineering_units ::= [engineering_prefix ] engineering_unit
  engineering_prefix ::= "E" | "P" | "T" | "G" | "M" | "k" | "m" | "u"
| "n" | "p" | "f" | "a"
  engineering_unit ::= "A" | Cel | "F" | "H" | Hz | "m" | Ohm | "s" |
"W" | "V"
  ref_varname ::=identifier
  events ::=  event
  | events "/" event
  event ::= "D" | ForceDown
  | "U" | ForceUp
  | "Z" | ForceOff
  | "P" | ForcePrior
  | "L" | CompareLow
  | "H" | CompareHigh
  | "x" | "X" | CompareUnknown
  | "T" | CompareOff
  | "V" | CompareValid
  | "l" | CompareLowWindow
  | "h" | CompareHighWindow
  | "t" | CompareOffWindow
  | "v" | CompareValidWindow
  | "N" | ForceUnknown
  | "A" | LogicLow
  | "B" | LogicHigh
  | "F" | LogicZ
  | "?" | Unknown
  | "G" | ExpectHigh
  | "R" | ExpectLow
  | "Q" | ExpectOff
  | "M" | Marker
  function ::= min
  | max
  |  identifier   (note:  allowed  identifiers  are  declared  in
user_functions stmt)
  function_args ::= time_expr
  | function_args "," time_expr
  13.0 Spec and Selector Block
  spec ::=  Spec[spec_name] "{" [ spec_list ] "}"
  spec_name ::= identifier
  spec_list ::= spec_item
  | spec_list spec_item
  spec_item ::= Categorycat_name "{" [ var_spec_info ] "}"
  | Variablevar_name "{" [ cat_spec_info ] "}"
  | include | annotation | udb | (null)
  cat_name ::= identifier
```

```
var_name ::= identifier
var_spec_info ::= var_spec_info_item
| var_spec_info var_spec_info_item
cat_spec_info ::= cat_spec_info_item
| cat_spec_info cat_spec_info_item
var_spec_info_item ::= var_name "=" "’" time_expr "’" ";"
| var_name "{" [Min"’" time_expr "’" ";"] [Typ"’" time_expr "’" ";"]
[Max"’" time_expr "’" ";"] "}"
| include | annotation | udb | (null)
cat_spec_info_item ::= cat_name "’" time_expr "’" ";"
| cat_name "{" [Min"’" time_expr "’" ";"] [Typ"’" time_expr "’" ";"]
[Max"’" time_expr "’" ";"] "}"
| include | annotation | udb | (null)
selector ::= Selectorselector_name "{" [ selector_list ] "}"
selector_name ::= identifier
selector_item ::= var_name selector_type ";"
selector_list ::=selector_item
| selector_list selector_item
selector_type ::= Min| Typ| Max| Meas
14.0 ScanStructures Block
scan_structs::= ScanStructuresscan_name "{" [ scanchains ] "}"
scanchains ::= scanchain
| scanchains scanchain
scanchain ::= ScanChainchainname "{" [ scan_struct_list ] "}"
| include | annotation | udb | (null)
chainname ::= identifier
scan_struct_list ::= scan_struct_item
| scan_struct_list scan_struct_item
scan_struct_item ::= ScanLengthinteger ";"
| ScanOutLengthinteger ";"
| ScanCellscellname_list ";"
| ScanInsignal_name ";"
| ScanOutsignal_name ";"
| ScanMasterClocksignal_name ";"
| ScanSlaveClocksignal_name ";"
| ScanInversionbit ";"
| include | annotation | udb | (null)
cellname_list ::= cellname | cellname_list cellname
cellname ::= identifier
| "!" identifier
bit ::=  "0" | "1"
15.0 Pattern Block
pattern_set ::= Patternpattern_name "{" [ pattern_statements ] "}"
pattern_name ::= identifier
```

```
  pattern_statements ::= pattern_stmt
  | pattern_statements pattern_stmt
  pattern_stmt ::= label pat_stmt
  | pat_stmt
  pat_stmt ::= waveform_table_stmt wft ";"
  | Loopinteger "{" [ pattern_statements ] "}"
  |              MatchLoopinteger              "{"pattern_statements
BreakPoint"{"pattern_statements"}" "}"
  |             MatchLoop              Infinite"{"pattern_statements
BreakPoint"{"pattern_statements"}""}"
  | vector_stmt
  | condition_stmt
  | Callprocedure_name ";"
  | Callprocedure_name "{" vec_data "}"
  | Macromacro_name ";"
  | Macromacro_name "{" vec_data "}"
  | GoTo pat_label ";"
  | Stop";"
  | ScanChain chain_name ";"
  | BreakPoint";"
  | BreakPoint"{" pattern_statements "}"
  |IddqTestPoint";"
  | TimeUnit"'" time_def "'" ";"
  | include | annotation | udb | (null)
  waveform_table_stmt ::= "W" |WaveformTable
  label ::= identifier ":"
  non_cyclized_data ::= "@" time_value event_pair ";"
  | "@" time_value "{" [ event_pair_list ] "}"
  event_pair ::= sigref_expr "=" event
  | include | annotation | udb | (null)
  event_pair_list ::= event_pair
  | event_pair_list ";" event_pair
  vector_stmt ::= "V" "{" vec_data "}"
  | Vect o r"{" vec_data "}"
  condition_stmt ::= "C" "{" vec_data "}"
  | Condition"{" vec_data "}"
  time_value ::= integer
  time_def ::= decimal [engineering_units]
  vec_data ::= vec_data_block
  | vec_data vec_data_block
  vec_data_block ::= sigref_expr "=" vec_data_string ";"
  | sigref_expr "{" vec_data_strings "}"
  | non_cyclized_data
  | include | annotation | udb | (null)
  vec_data_strings ::= vec_data_string ";"
```

```
  | vec_data_strings vec_data_string ";"
  | include | annotation | udb | (null)
  vec_data_string ::=wfc_data_string (Note: string type is runtime
  | hex_data_string dependent based on the
  | dec_data_string sig_refs Base definition)
  wfc_mode ::= "\w " wfc_data_string
  hex_mode ::= "\h " hex_data_string
  | "\h"wfcs hex_data_string
  dec_mode ::= "\d " dec_data_string
  | "\d"wfcs dec_data_string
  wfc_data_string ::= wfc_data_string wfc_data
  | wfc_data
  wfc_data ::= wfcs
  | "\r"integer wfcs
  | hex_mode
  | dec_mode
  hex_data_string ::= hex_data_string hex_data
  | hex_data
  hex_data ::= hexchars
  | "\r"integer hexchars
  | wfc_mode
  | dec_mode
  dec_data_string ::= dec_data_string dec_data
  | dec_data
  dec_data ::= integer
  | "\r"integer integer
  | wfc_mode
  | hex_mode
  16.0 Procedures Block
  procedures    ::=    Procedures[procedure_domain_name]    "{"    [
procedure_definitions ] "}"
  procedure_domain_name ::= identifier
  procedure_definitions ::= procedure
  | procedure_definitions procedure
  procedure_name ::= identifier
  procedure ::= procedure_name "{" [ procedure_statements ] "}"
  | include | annotation | udb | (null)
  procedure_statements ::= procedure_or_macro_item
  | procedure_statements procedure_or_macro_item
  procedure_or_macro_item ::= Shift"{" pattern_statements "}"
  | pat_stmt
  17.0 Macrodefs Block
  macrodefs ::= MacroDefs[ macro_domain_name ] "{" [ macro_definitions
] "}"
  macro_domain_name ::= identifier
```

```
  macro_definitions ::= macro
  | macro_definitions macro
  macro ::= macro_name "{" [ macro_statements ] "}"
  | include | annotation | udb | (null)
  macro_name ::= identifier
  macro_statements ::= procedure_or_macro_item
  | macro_statements procedure_or_macro_item
  18.0 Other Miscellaneous Statements
  identifier ::=identifier_segment
  | identifier "." identifier_segment
  identifier_segment ::= simple_identifier (Note the maximum length of
an identifier segment | escaped_identifier  is 1024 characters)
  simple_identifier ::= letter_or_underline simple_characters
  simple_characters ::= simple_characters simple_character
  | (null)
  letter_or_underline ::= letter
  | underline
  simple_character ::= letter | digit | underline
  letter ::=  upper_case_letter | lower_case_letter
  upper_case_letter ::= "A"| "B" | ... | "Z"
  lower_case_letter ::= "a"| "b" | ... | "z"
  underline ::="_"
  escaped_identifier ::="" escaped_characters ""
  escaped_characters ::= escaped_characters escaped_character
  | escaped_character
  escaped_character ::= simple_character
  | special_character
  | whitespace_character
  special_character ::= !@#$%^&*()-+=|`~{[}]:;',<.>/?\
  whitespace_character ::= " " | "\t" | "\n"
  string ::=  escaped_identifier
  hexdigit ::= digit | "a" | "A" | "b" | "B" | "c" | "C" | "d" | "D" |
"e" | "E" | "f" | "F"
  digit ::=  "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
  hexdigits ::= hexdigit | hexdigits hexdigit
  integer ::= digit | integer digit
  signed_integer ::= integer | "-" integer
  decimal ::= signed_integer
  | signed_integer "." integer
  | signed_integer "e" signed_integer
| signed_integer "." integer "e" signed_integer
```

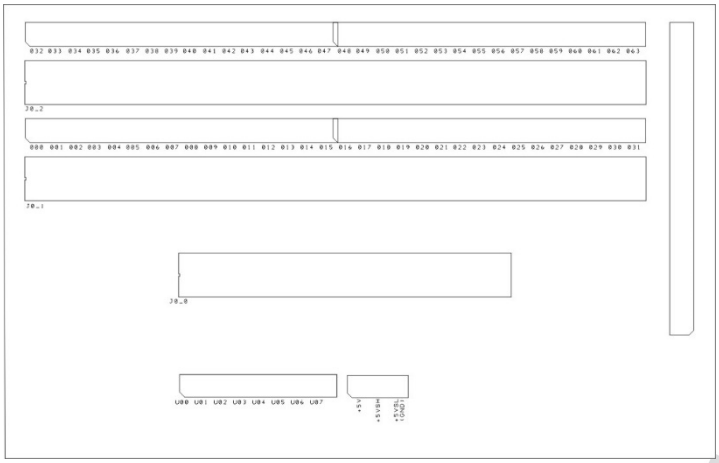
Παράρτημα 2

Λίστα Πινάκων

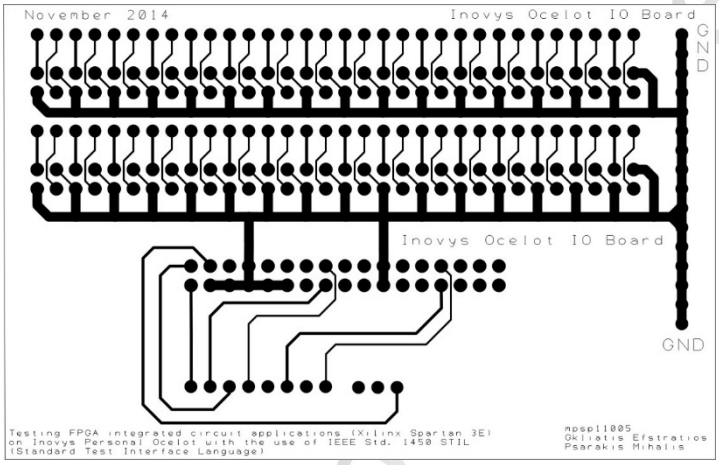
Λίστα Εικόνων

Παράρτημα 3

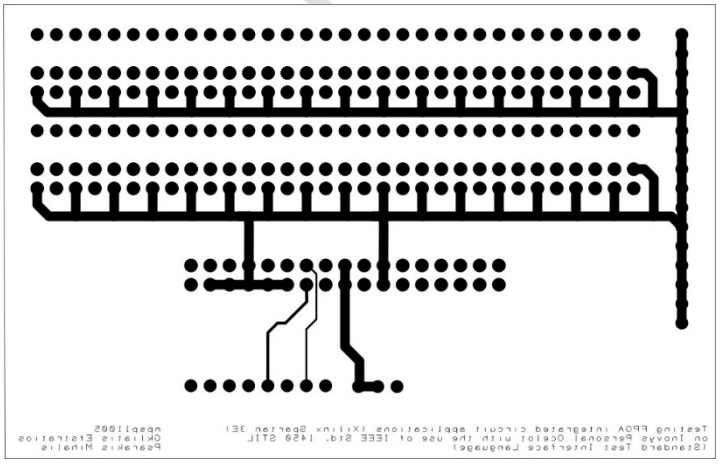
Inovys Ocelot I/O Board Layers



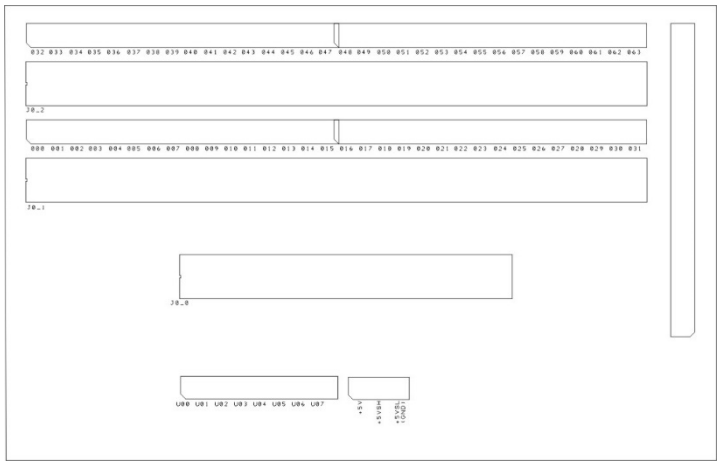
Εικόνα 82. Inovys Ocelot IO Board – Top Silk



Εικόνα 83. Inovys Ocelot IO Board – Top Copper



Εικόνα 84. Inovys Ocelot IO Board – Bottom Silk



Εικόνα 85. Inovys Ocelot IO Board – Drill Ident

Inovys Personal Ocelot Load Board Pin Out

DOCUMENT, INOVYS PERSONAL OCELOT LOAD BOARD PINOUTS

01685 Rev F 9-March-2006 Page 1 of 2

Table 1 below shows the pin connections for the 34 and 64 pin TRG connectors JX-0 thru JX-3, where X is the TRG number.
Table 2 shows the connections for the two 20 pin (J5 & J6) and one 24 pin (J4) Mini-Fit-Jr Molex connectors.

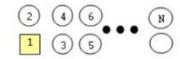
Table 1: The TRG Connectors

There are 3 connectors per TRG, one is a 34 pin style and two are a 64 pin style.
The 34 pin connectors provide User Bit controls, and has +5V Utility to power optional support circuitry on the load board.
The 64 pin connectors provide the high speed Pin Channels for interfacing with the Device Under Test (DUT).
The 34 pin connector (J1-0) on TRG1 also contains the signal LBP for detecting the presence of a load board.
The +5V Utility supply can provide up to 3Amps of current. This is enough for most applications, such as relays and LEDs.

Notes for the TRG connectors:

1 The LBP* pin on J1-0[26] should be grounded on your loadboard to enable the DPS and UTILITY supplies.
2 If optional load board circuitry is needed, connect all 4 UTIL+5V and all 4 U+5VSH signals from the TRGs to your Utility +5V supply plane on your loadboard. All 4 U_+5VSL signals should be connected to GND.
3 All traces for Pin Channels should be 75 ohm. This makes the load board thicker. You may want to use Top,Plane,Signal,Signal,Plane,Signal,Signal,Plane,Bottom style of layer stackup to achieve enough routing layers.
4 Load boards can not be greater than 0.1 inches thick, or standard connectors pins will be hard to solder in place. If using Inovys PN 02239 (64 pin) and 02240 (34 pin) connectors, load boards can be up to 0.185 inches thick.
5 Connect all of the pins labeled GND to your load board's Ground Plane.
6 Leave the undefined pins on the 34 pin connectors unconnected on your load board.

All JX-0 thru JX-3 connectors use this pin numbering scheme



N is the last pin of the connector.
This is looking from the top side of load board.
But note, these connectors are mounted on the bottom of the load board.

Table 1

	TRG0				TRG1				TRG2				TRG3			
PIN	J0-0	J0-1	J0-2	J0-3	J1-0	J1-1	J1-2	J1-3	J2-0	J2-1	J2-2	J2-3	J3-0	J3-1	J3-2	J3-3
1	USER00	GND	GND	Future Expansion (Not USed)	USER08	GND	GND	Future Expansion (Not USed)	USER16	GND	GND	Future Expansion (Not USed)	USER24	GND	GND	Future Expansion (Not USed)
2	USER02	P000	P032		USER10	P064	P096		USER18	P128	P160		USER26	P192	P224	
3	GND	P001	P033		GND	P065	P097		GND	P129	P161		GND	P193	P225	
4	U_+5VSL	GND	GND		U_+5VSL	GND	GND		U_+5VSL	GND	GND		U_+5VSL	GND	GND	
5	GND	GND	GND		GND	GND	GND		GND	GND	GND		GND	GND	GND	
6		P002	P034			P066	P098			P130	P162			P194	P226	
7	GND	P003	P035		GND	P067	P099		GND	P131	P163		GND	P195	P227	
8	GND	GND	GND		GND	GND	GND		GND	GND	GND		GND	GND	GND	
9	GND	GND	GND		GND	GND	GND		GND	GND	GND		GND	GND	GND	
10		P004	P036			P068	P100			P132	P164			P196	P228	
11	GND	P005	P037		GND	P069	P101		GND	P133	P165		GND	P197	P229	
12		GND	GND			GND	GND			GND	GND			GND	GND	
13	USER04	GND	GND		USER12	GND	GND		USER20	GND	GND		USER28	GND	GND	
14	USER06	P006	P038		USER14	P070	P102		USER22	P134	P166		USER30	P198	P230	
15	USER01	P007	P039		USER09	P071	P103		USER17	P135	P167		USER25	P199	P231	
16	USER03	GND	GND		USER11	GND	GND		USER19	GND	GND		USER27	GND	GND	
17	UTIL+5V	GND	GND		UTIL+5V	GND	GND		UTIL+5V	GND	GND		UTIL+5V	GND	GND	
18	U+5VSH	P008	P040		U+5VSH	P072	P104		U+5VSH	P136	P168		U+5VSH	P200	P232	
19		P009	P041			P073	P105			P137	P169			P201	P233	
20		GND	GND			GND	GND			GND	GND			GND	GND	
21	GND	GND	GND		GND	GND	GND		GND	GND	GND		GND	GND	GND	
22	GND	P010	P042		GND	P074	P106		GND	P138	P170		GND	P202	P234	
23		P011	P043			P075	P107			P139	P171			P203	P235	
24		GND	GND			GND	GND			GND	GND			GND	GND	
25		GND	GND			GND	GND			GND	GND			GND	GND	
26		P012	P044		LBP*	P076	P108			P140	P172			P204	P236	
27	USER05	P013	P045		USER13	P077	P109		USER21	P141	P173		USER29	P205	P237	
28	USER07	GND	GND		USER15	GND	GND		USER23	GND	GND		USER31	GND	GND	
29		GND	GND			GND	GND			GND	GND			GND	GND	
30		P014	P046			P078	P110			P142	P174			P206	P238	
31		P015	P047			P079	P111			P143	P175			P207	P239	
32		GND	GND			GND	GND			GND	GND			GND	GND	
33		GND	GND			GND	GND			GND	GND			GND	GND	
34		P016	P048			P080	P112			P144	P176			P208	P240	
35		P017	P049			P081	P113			P145	P177			P209	P241	
36		GND	GND			GND	GND			GND	GND			GND	GND	
37		GND	GND			GND	GND			GND	GND			GND	GND	
38		P018	P050			P082	P114			P146	P178			P210	P242	
39		P019	P051			P083	P115			P147	P179			P211	P243	
40		GND	GND			GND	GND			GND	GND			GND	GND	
41		GND	GND			GND	GND			GND	GND			GND	GND	
42		P020	P052			P084	P116			P148	P180			P212	P244	
43		P021	P053			P085	P117			P149	P181			P213	P245	
44		GND	GND			GND	GND			GND	GND			GND	GND	
45		GND	GND			GND	GND			GND	GND			GND	GND	
46		P022	P054			P086	P118			P150	P182			P214	P246	
47		P023	P055			P087	P119			P151	P183			P215	P247	
48		GND	GND			GND	GND			GND	GND			GND	GND	
49		GND	GND			GND	GND			GND	GND			GND	GND	
50		P024	P056			P088	P120			P152	P184			P216	P248	
51		P025	P057			P089	P121			P153	P185			P217	P249	
52		GND	GND			GND	GND			GND	GND			GND	GND	
53		GND	GND			GND	GND			GND	GND			GND	GND	
54		P026	P058			P090	P122			P154	P186			P218	P250	
55		P027	P059			P091	P123			P155	P187			P219	P251	
56		GND	GND			GND	GND			GND	GND			GND	GND	
57		GND	GND			GND	GND			GND	GND			GND	GND	
58		P028	P060			P092	P124			P156	P188			P220	P252	
59		P029	P061			P093	P125			P157	P189			P221	P253	
60		GND	GND			GND	GND			GND	GND			GND	GND	
61		GND	GND			GND	GND			GND	GND			GND	GND	
62		P030	P062			P094	P126			P158	P190			P222	P254	
63		P031	P063			P095	P127			P159	P191			P223	P255	
64		GND	GND			GND	GND			GND	GND			GND	GND	

Εικόνα 86. Inovys Ocelot Pin Out 1/2

DOCUMENT, INOVYS PERSONAL OCELOT LOAD BOARD PINOUTS

01685 Rev F 9-March-2006 Page 2 of 2

Table 2: The Molex Connectors

The 3 Molex connectors provide Device Power Supplies (DPS), Direct PMU (DPMU) connections and auxiliary functions. There are 4 standard DPS and 12 optional DPS at these connectors.

Notes for the Molex connectors:

1 Connect the DGS (Device Ground Sense) pin to your load board's Ground Plane.
2 The first 4 DPS supplies (0-3) are supplied by the 4 standard DPSs built into the system PowerBoard. They do not support Current Measurent.
3 The 12 optional DPS supplies (10-13 and 42-49) do support Current Measurent.
4 DPS 0-3 and 10-13 are 2.5Amp supplies. DPS 42-49 are 200mAmp supplies.
5 Each 2.5Amp DPS has its own Force High (FH), Sense High (SH) and Force Low (FL) pin. Each 200mAmp DPS has its own Force High (FH) and Sense High (SH) pin, but they share a common Force Low (FLA, FLB).
6 Connect the Force High to the Sense High for any supply used.
7 Connect the Force Low to the GND plane for any supply used.
8 DPS_FLA, DPS_FLB: Connect them to GND plane if you plan to use any of DPS42 - DPS49.
9 Connect all of the pins labeled GND to your load board's Ground Plane.
10 DPMU_FH0..DPMU_FH3 and DPMU_SH0..DPMU_SH3: Are the force high and sense high of each respective Direct PMU. Connect the Force High to the Sense High for any Direct PMU used.
11 DVM_FH, DVM_SH, DVM_FL, DVM_SL, DVM_I+ : These pins are for Inovys internal use only, so leave them unconnected.
12 AUX_+5V and AUX_+5V_SH : These pins are for Inovys internal use only, so leave them unconnected.

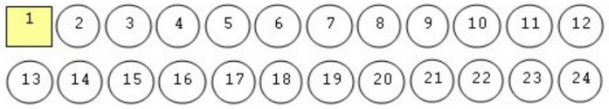
Table 2

Molex J4 (24 Pin)

1	DPS_FH0	DPS_SH0	13	PWRBRD 2.5A
2	DPS_FL0	DPS_FL1	14	
3	DPS_FH1	DPS_SH1	15	
4	DPS_FH2	DPS_SH2	16	
5	DPS_FL2	DPS_FL3	17	
6	DPS_FH3	DPS_SH3	18	
7	DPS_FH10	DPS_SH10	19	PPRG 2.5A
8	DPS_FL10	DPS_FL11	20	
9	DPS_FH11	DPS_SH11	21	
10	DPS_FH12	DPS_SH12	22	
11	DPS_FL12	DPS_FL13	23	
12	DPS_FH13	DPS_SH13	24	

These Molex connectors have the pin numbers molded into the plastic.

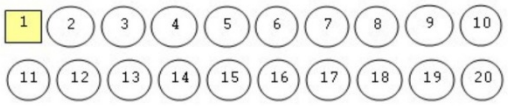
Looking from top view of the mounting holes for J4.
24 pin right-angle through-hole PCB mount. Mounted on top side of load board.



Molex J5 (20 Pin)

1	DPS_FH42	DPS_SH42	11	PPRG 200mA/16V
2	DPS_FH43	DPS_SH43	12	
3	DPS_FH44	DPS_SH44	13	
4	DPS_FH45	DPS_SH45	14	
5	DPS_FH46	DPS_SH46	15	
6	DPS_FH47	DPS_SH47	16	
7	DPS_FH48	DPS_SH48	17	
8	DPS_FH49	DPS_SH49	18	
9	DPS_FLA	DPS_FLB	19	
10	DGS	GND	20	

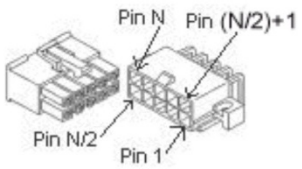
Looking from top view of the mounting holes for J5 & J6.
20 pin right-angle through-hole PCB mount. Mounted on top side of load board.



Molex J6 (20 Pin)

1	DPMU_FH0	DPMU_SH0	11
2	DPMU_FH1	DPMU_SH1	12
3	DPMU_FH2	DPMU_SH2	13
4	DPMU_FH3	DPMU_SH3	14
5	DVM_FH	DVM_SH	15
6	DVM_FL	DVM_SL	16
7		DVM_I+	17
8	AUX_+5V_SH	GND	18
9	GND	GND	19
10	GND	AUX_+5V	20

This is a picture to help clarify the mating of the Molex connectors.



On load board:
24 pin Molex PN 39-29-1248, Inovys PN 01450
20 pin Molex PN 39-29-1207, Inovys PN 00068

N is the highest pin on the connector

Εικόνα 87. Inovys Ocelot Pin Out 2/2

Βιβλιογραφία

[1] Elements of STIL, Principles and Applications of IEEE Std. 1450, 2003
[2] Introductory STIL BNF, 1998
[3] IEEE STIL (P1450) Standardization Status, 2005
[4] Standard Test Interface Language (STIL) A New Language for Patterns and Waveforms, 1996
[5] Test Techniques for System-on-a-Chip, 2005
[6] Inovys Stylus Tools User Guide, 2002
[7] Inovys User / Maintenance Manual 7.9
[8] Test System Specifications
[9] The Role of Verification and Validation in System Development Life Cycle, 2012