ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

Τμήμα Πληροφορικής και Τηλεπικοινωνιών

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΑΞΙΟΛΟΓΗΣΗ ΜΗΧΑΝΙΣΜΟΥ ΜΟΝΑΔΙΚΗΣ ΑΝΑΓΝΩΡΙΣΗΣ ΣΕ FPGA

Μπασούνας Διονύσιος

Α.Μ. 1401

Επιβλέπων: Βαρτζιώτης Φώτιος

ΔΕΠ ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ

Άρτα, Φεβρουάριος 2022

IMPLEMENTATION AND EVALUATION OF A UNIQUE BOARD IDENTIFICATION MECHANISM IN FPGA

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, 25/2/2022

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής

   Βαρτζιώτης Φώτιος,

2. Μέλος επιτροπής

   Γιαννακέας Νικόλαος,

3. Μέλος επιτροπής

   Δουμένης Γρηγόριος,

Δήλωση μη λογοκλοπής

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

Μπασούνας, Διονύσης

Υπογραφή

Περίληψη

Η παρούσα πτυχιακή εργασία στοχεύει στη σχεδίαση και υλοποίηση σε υλικό (hardware) ενός μηχανισμού μοναδικής αναγνώρισης τσιπ FPGA. Ο παραπάνω μηχανισμός πιστοποιεί ένα τσιπ FPGA και το καθιστά αξιόπιστο για χρήση σε ιδιωτικές αλλά και ευαίσθητες εφαρμογές. Η λειτουργία του μηχανισμού βασίζεται στη θεωρία των φυσικών μη κλωνοποιήσιμων συναρτήσεων (ΦΜΚΣ – Physical Unclonable Functions (PUF)). Οι ΦΜΚΣ επιτρέπουν τη δημιουργία ενός μοναδικού ψηφιακού αποτυπώματος για κάθε συσκευή, μέσω της εργαλειοποίησης των μοναδικών φυσικών χαρακτηριστικών της, που οφείλονται στις ατέλειες της κατασκευαστικής διαδικασίας.

Στο πλαίσιο της πτυχιακής μελετήθηκαν ΦΜΚΣ που ανήκουν σε δυο κατηγορίες, τις βασισμένες στην καθυστέρηση (Delay-based PUFs) και τις βασισμένες στη μνήμη (Memory-based PUF). Για τη σχεδίαση και υλοποίηση του μηχανισμού μοναδικής αναγνώρισης επιλέχθηκε μια ΦΜΚΣ βασισμένη στην καθυστέρηση. Ως κύριο συστατικό μέρος του μηχανισμού επιλέχθηκαν οι ταλαντωτές δακτυλίου (Ring Oscillators). Με την βοήθεια του εργαλείου σχεδίασης Quartus II της εταιρείας INTEL και του εργαλείου προσομοίωσης ModelSim της εταιρείας Siemens σχεδιάσαμε, συνθέσαμε, προσομοιώσαμε και πρωτοτυποποιήσαμε τον μηχανισμό ενδιαφέροντος σε τσιπ FPGA Cyclone IV μιας πλατφόρμας δοκιμής DE2. Το τελικό αποτέλεσμα είναι η παραγωγή μιας ψηφιακής υπογραφής της συσκευής, βασισμένη σε διάνυσμα εισόδου της επιλογής μας.

Abstract

The present thesis aims at the design and implementation in hardware of a unique FPGA chip recognition mechanism. The above mechanism certifies an FPGA chip and makes it reliable for use in both private and sensitive applications. The operation of the mechanism is based on the theory of Physical Unclonable Functions (PUF). FMCs allow the creation of a unique digital footprint for each device, through the instrumentation of its unique physical characteristics, due to the imperfections of the manufacturing process.

In the context of the dissertation, FMSs belonging to two categories were studied, the Delay-based PUFs and the Memory-based PUFs. A delay-based FMS was selected for the design and implementation of the unique recognition mechanism. Ring Oscillators were chosen as the main component of the mechanism. With the help of INTEL Quartus II design tool and Siemens ModelSim simulation tool we designed, synthesized, simulated and prototyped the FPGA Cyclone IV chip of interest of a DE2 test platform. The end result is the production of a digital signature of the device, based on an input vector of our choice.

Περιεχόμενα

1. Τι θέλουμε να κάνουμε στην εργασία

Ο σκοπός ενός μηχανισμού μοναδικής αναγνώρισης τσιπ, ο οποίος βασίζεται σε μια φυσική μη κλωνοποιήσιμη συνάρτηση (ΦΜΚΣ), είναι να παρέχει μια μοναδική ψηφιακή υπογραφή (με τη μορφή δυαδικής ακολουθίας) σε κάθε συσκευή, η οποία δημιουργείται από τα μοναδικά φυσικά χαρακτηριστικά της.

Αυτή η υπογραφή μπορεί να χρησιμοποιηθεί για να ξεχωρίσει τη συσκευή από άλλες συσκευές ή ακόμη και να είναι μέρος ενός κρυπτογραφικού πρωτοκόλλου για να επιτρέψει μόνο στον κάτοχο της συσκευής να έχει πρόσβαση σε ορισμένους πόρους. Με δεδομένο ότι ένας εισβολέας μπορεί να έχει φυσική πρόσβαση σε μια μνήμη αποθήκευσης μιας ψηφιακής υπογραφής με συγκριτικά μικρή προσπάθεια, το πλεονέκτημα ενός μηχανισμού βασισμένου σε ΦΜΚΣ είναι ότι η υπογραφή δημιουργείται μόνο όταν χρειάζεται και αποθηκεύεται προσωρινά, καθιστώντας μια επίθεση πολύ πιο δύσκολη. Επιπλέον, καθώς μια υπογραφή βασισμένη σε ΦΜΚΣ δημιουργείται από φυσικά χαρακτηριστικά, μια επεμβατική επίθεση στη συσκευή είναι πιθανό να διαταράξει αυτά τα χαρακτηριστικά, έτσι ώστε να αλλάξει η υπογραφή.

Το αν μια υπογραφή βασισμένη σε ΦΜΚΣ μπορεί να διαβαστεί εύκολα από έναν εισβολέα ή αν παραμένει κρυφή εξαρτάται από την υλοποίηση. Στην πρώτη περίπτωση, ένας εισβολέας μπορεί να διαβάσει την υπογραφή και ενδεχομένως να δημιουργήσει μια συσκευή με την ίδια υπογραφή. Στην δεύτερη περίπτωση εάν η υπογραφή χρησιμοποιείται μόνο στο εσωτερικό της συσκευής ένας επιτιθέμενος πρέπει να κάνει πολύ περισσότερες προσπάθειες για να αποκτήσει τη μυστική υπογραφή. Ωστόσο, η χρήση της εξόδου ΦΜΚΣ ως cryptographic seed απαιτεί τέλεια αξιοπιστία, διότι ένα μόνο bit-flip στο seed οδηγεί σε ένα εντελώς διαφορετικό κλειδί. Καθώς η τέλεια αξιοπιστία σχεδόν δεν επιτυγχάνεται σε οποιοδήποτε ΦΜΚΣ, πρέπει να χρησιμοποιούνται σχήματα διόρθωσης σφαλμάτων. Εάν η υπογραφή διαβάζεται απευθείας, η μη τέλεια αξιοπιστία μπορεί να αντισταθμιστεί θεωρώντας μια υπογραφή ως "σωστή" όταν αρκετά bits υπογραφής έχουν τις αναμενόμενες τιμές τους.

Στα επόμενα κεφάλαια περιγράφεται λεπτομερώς ο τρόπος χρήσης του λογισμικού INTEL Quartus II για το σχεδιασμό μιας ΦΜΚΣ με καθυστέρηση. Ως κύριο συστατικό μέρος του μηχανισμού επιλέχθηκαν οι ταλαντωτές δακτυλίου (Ring Oscillators). Με το Quartus II και το Modelsim σχεδιάσαμε, συνθέσαμε, προσομοιώσαμε και πρωτοτυποποιήσαμε τον μηχανισμό ενδιαφέροντος σε τσιπ FPGA Cyclone IV μιας πλατφόρμας δοκιμής DE2.

2. Hardware security

Η ασφάλεια υλικού (hardware security) αφορά στην προστασία και πιστοποίηση του υλικού ψηφιακών συστημάτων που αποτελούν τη βάση για την εκτέλεση ενός αριθμού εφαρμογών όπως την επαλήθευση και την εξασφάλιση ψηφιακών ταυτοτήτων, συστήματα τραπεζών, μέσα κοινωνικής δικτύωσης και τα κρυπτονομίσματα,. Κακόβουλες επιθέσεις στο υλικό μπορεί να συμβούν κατά τη διάρκεια της κατασκευής του ή / και κατά τη διάρκεια κανονικής λειτουργίας του. Για την αντιμετώπιση τους στο στάδιο της κατασκευής μπορεί να χρησιμοποιούνται ειδικά πρωτόκολλα στη διαδικασία σχεδίασης ή / και η προσθήκη εξειδικευμένων φυσικών διατάξεων που έχουν ως στόχο την αναγνώριση αλλά και την «απόκρουση» των επιθέσεων. Στη διάρκεια της κανονικής λειτουργίας, φυσικές διατάξεις αναλαμβάνουν να προστατεύσουν το υλικό αλλά και να ενισχύσουν την κρυπτογραφική «ισχύ» ενός λογισμικού ασφαλείας σε μια κρίσιμη εφαρμογή. Συνήθη παραδείγματα ασφάλειας υλικού αποτελούν τα τοίχοι προστασίας υλικού (hardware firewalls) και οι μονάδες ασφαλείας υλικού (Hardware Security Modules - HSM), οι οποίες παρέχουν κρυπτογραφικά κλειδιά για κρίσιμες λειτουργίες όπως κρυπτογράφηση, αποκρυπτογράφηση και έλεγχο ταυτότητας για διάφορα συστήματα.

Η εξέλιξη της έρευνας όσον αφορά στην ασφάλεια υλικού μπορεί να περιλαμβάνει μεθοδολογίες για την ανίχνευση κακόβουλου υλικού (Hardware Trojans), μεθοδολογίες προστασίας από επιθέσεις σε παράπλευρα κανάλια του υλικού (side-channel attacks) ή και μεθοδολογίες για την ανάπτυξη αξιόπιστου υλικού από την κατασκευή (root-of-trust). Στην τελευταία περίπτωση, εγγενείς ιδιότητες των συσκευών που μπορεί να έχουν αρνητικό αντίκτυπο στην απόδοση τους, χρησιμοποιούνται για την πιστοποίηση τους. Ένα κορυφαίο παράδειγμα είναι η χρήση φυσικών μη κλωνοποιήσιμων συναρτήσεων, που αξιοποιούν τη διαφορετικότητα των συσκευών λόγω ατελειών στη διαδικασία κατασκευής, για τη δημιουργία «δακτυλικών» αποτυπωμάτων συγκεκριμένων τσιπ που αναγνωρίζονται μέσω ζευγών πρόκλησης-απόκρισης.

Η ασφάλεια υλικού παίζει σημαντικό ρόλο στη διασφάλιση της εμπιστοσύνης, της ακεραιότητας και της γνησιότητας των ολοκληρωμένων κυκλωμάτων (IC) και των ηλεκτρονικών συστημάτων. Ειδικότερα στην περίπτωση κρίσιμων εφαρμογών, η ενίσχυση της αξιοπιστίας και της προστασίας από κακόβουλες επιθέσεις μπορεί να δικαιολογήσει ακόμη και σημαντική αύξηση του κόστους στη σχεδίαση και υλοποίηση ενός υλικού. Κρίσιμα συστήματα και υποδομές είναι για παράδειγμα δίκτυα και πόροι των οποίων η συνεχής λειτουργία κρίνεται απαραίτητη για τη διασφάλιση της δημόσιας ασφάλειας, της οικονομίας και της υγείας. Η ασφάλεια των υποδομών κρίσιμης σημασίας αποτελεί μια αυξανόμενη περιοχή ανησυχίας σε όλο τον κόσμο.

3. Φυσικά μη κλωνοποιήσιμες συναρτήσεις (ΦΜΚΣ)

Μια ΦΜΚΣ αντιπροσωπεύει ένα μηχανισμό που εκμεταλλεύεται την εγγενή τυχαιότητα που εισήχθη κατά τη διάρκεια της κατασκευής, για να δώσει σε μια φυσική οντότητα ένα μοναδικό «δακτυλικό αποτύπωμα». Αυτοί οι μηχανισμοί μπορούν να χρησιμοποιηθούν σε μια ποικιλία εφαρμογών, για την προστασία από παραχάραξη και ταυτοποίηση. Στην παρούσα πτυχιακή εξετάζουμε την ανάπτυξη ΦΜΚΣ που εκμεταλλεύονται τις χρονικές καθυστερήσεις ενός κυκλώματος και τα πτητικά / μη-πτητικά (volatile/non-volatile) χαρακτηριστικά των μνημών. Εξετάζουμε επίσης τις διάφορες εφαρμογές των ΦΜΚΣ και τα ζητήματα ασφαλείας που πρέπει να αντιμετωπίσουν, επισημαίνοντας τις γνωστές επιθέσεις μέχρι σήμερα και πιθανά αντίμετρα. Επιπρόσθετα, εξετάζουμε τις βασικές κατευθύνσεις για μελλοντική ανάπτυξη, όπως η αξιοπιστία των bit, η επαναδιαμόρφωση και η υποδομή δημόσιου κλειδιού.

Έστω ότι η ΦΜΚΣ είναι μια μαθηματική έκφραση θ που δέχεται έναν αριθμό εισόδων και παράγει αντίστοιχα έναν αριθμό εξόδων, δηλαδή

$$\theta \in \Theta : C \rightarrow R | \theta\,(c) = r,\ c \in C,\ r \in R.$$

Το πεδίο ορισμού της συνάρτησης θ, δηλαδή το σύνολο C των επιτρεπόμενων εισόδων, θα αναφέρεται από εδώ και στο εξής ως σύνολο προκλήσεων, ενώ το σύνολο R της παραπάνω έκφρασης, θα αναφέρεται ως σύνολο αποκρίσεων της θ. Αντίστοιχα το ζεύγος (c, θ(c)) = (c, r) ορίζεται ως ζεύγος πρόκλησης / απόκρισης (ΖεΠΑ). Τότε, η ΦΜΚΣ μπορεί να θεωρηθεί ως ένας κυκλωματικός μηχανισμός ικανός να παράγει ένα μοναδικό ΖεΠΑ.

Λόγω της φύσης των κυκλωμάτων, μπορεί εύκολα να θεωρηθεί ότι η συνάρτηση θ δεν είναι αναγκαίο να αποτελεί μια συνάρτηση με την αυστηρή μαθηματική έννοια. Μπορεί να αντιπροσωπεύει ένα σύνολο χρονικά μεταβαλλόμενων φυσικών παραμέτρων του κυκλώματος λόγω για παράδειγμα της μεταβολής του θερμικού θορύβου, της τάσης και της θερμοκρασίας. Αυτή η διακύμανση των φυσικών παραμέτρων του κυκλώματος μπορεί να συνυπολογιστεί αν αντικαταστήσουμε την απόκριση r του κυκλώματος σε μια είσοδο, με μια τυχαία μεταβλητή Xr της οποίας η κατανομή εξαρτάται από την τεχνολογία κατασκευής και υλοποίησης, την αρχιτεκτονική της ΦΜΚΣ και τις θεωρούμενες περιβαλλοντικές συνθήκες. Στην περίπτωση αυτή, η απόκριση r θεωρείται ως η πιθανότερη απόκριση της ΦΜΚΣ όταν εφαρμόζουμε μια πρόκληση c, δηλαδή θ(c)~= r. Όσο μεγαλύτερη είναι η πιθανότητα να έχουμε απόκριση r από την εφαρμογή του c, τόσο πιο σταθερή θεωρείται η ΦΜΚΣ.

3.1. Προϋποθέσεις για μια αξιοποιήσιμη ΦΜΚΣ

Για να έχουμε μια αξιοποιήσιμη ΦΜΚΣ, δηλαδή κάτι μοναδικό και δύσκολο να δεχθεί επίθεση, θα πρέπει να ισχύουν ένα σύνολο από προϋποθέσεις.

3.1.1 Αδυναμία Κλωνοποίησης - Unclonability

Η κύρια ιδιότητα που πρέπει να έχει μια ΦΜΚΣ είναι φυσικά, από το όνομα της, η μη δυνατότητα κλωνοποίησης της. Αυτό σημαίνει ότι είναι αδύνατον να δημιουργηθεί μια συνάρτηση θ', ισοδύναμη της θ, που μπορεί να την αντικαταστήσει είτε με μαθηματικό είτε με φυσικό τρόπο. Συγκεκριμένα, η από μαθηματικής άποψης αδυναμία κλωνοποίησης σημαίνει ότι είναι δύσκολο να βρεθεί μια μαθηματική διαδικασία f που να είναι σε θέση να παρέχει το ίδιο ζεύγος πρόκλησης / απόκρισης, ενώ η από φυσικής άποψης αδυναμία κλωνοποίησης δείχνει ότι είναι δύσκολο να αναπαραχθεί μια συσκευή με ΦΜΚΣ θ΄ που να μπορεί να αναγνωριστεί ως θ.

$$\nexists f : \theta(c) = f(c) = r, \ \forall c \in C.$$

$$\nexists \theta' : \theta(c) = \theta'(c) = r, \ \forall c \in C.$$

Εξίσωση 1 Unclonability

Η ισχύς της παραπάνω ιδιότητας μπορεί να επιτευχθεί από τη διαδικασία κατασκευής και τον εσωτερικό σχεδιασμό της ΦΜΚΣ. Είναι αδύνατον σήμερα να μοντελοποιήσουμε τη διαδικασίας κατασκευής μια συσκευής και των επιπτώσεών της σε κάθε φυσική παράμετρο που καθορίζει τη συμπεριφορά τηςΦΦ ΦΜΚΣ.

3.1.2 Μοναδικότητα - Uniqueness

Η μοναδικότητα μοιάζει με την ιδιότητα της αδυναμίας κλωνοποίησης. Ωστόσο υπάρχουν διαφορές. Για να είναι μοναδική η κάθε ΦΜΚΣ θ θα πρέπει να μην υπάρχει μια άλλη θ΄ που να ανήκει συγκεκριμένα στο Θ, και να είναι ισοδύναμη της θ. Αντίστοιχα, η αδυναμία κλωνοποίησης απαιτεί να μην υπάρχει μια οποιαδήποτε γενικά συνάρτηση που είναι σε θέση να υποκαταστήσει τη ΦΜΚΣ θ. Ακόμη, η μοναδικότητα απαιτεί να ισχύει μόνο για ένα υποσύνολο του πεδίου ορισμού.

$$\nexists \widetilde{\theta} \in \Theta : \widetilde{\theta}(c) = \theta(c) = r, \ \forall c \in \widehat{C} \subseteq C.$$

Εξίσωση 2 Uniqueness

3.1.3 Αδυναμία πρόβλεψης - Unpredictability

Η εν λόγω ιδιότητα μπορεί να προκύψει άμεσα από τον ορισμό της αδυναμίας κλωνοποίησης από μαθηματικής άποψης.

$$\Psi = \{(c, \theta(c))\}, \ \nexists \Phi : \Phi(\Psi, c_p) = \theta(c_p) = r_p, \ \forall c_p \in C.$$

Εξίσωση 3 Unpredictability

Δηλώνει την αδυναμία δημιουργίας μιας διαδικασίας Φ, που μέσα από ένα συγκεκριμένο αριθμό ζευγών πρόκλησης / απόκρισης μιας ΦΜΚΣ, να είναι σε θέση να προβλέψει το αποτέλεσμα της θ για μια γενική πρόκληση .

3.1.4 Ιδιότητα One-Way

Η εν λόγω ιδιότητα δηλώνει ότι με δεδομένη την έξοδο μιας ΦΜΚΣ, είναι σχεδόν αδύνατο ή εξαιρετικά απαιτητικό, από την άποψη των πόρων υπολογισμού και του χρόνου, να βρεθεί μια συνάρτηση λ:λ(r)=c' που να υπολογίζει μια είσοδο c' της ΦΜΚΣ, τέτοια ώστε θ(c') = r.

3.1.5 Υλοποιησιμότητα

Μια ΦΜΚΣ επιβαρύνει ένα κύκλωμα όσον αφορά στην επιφάνεια του αλλά πιθανώς και από την πλευρά της απόδοσης. Όσον αφορά στην επιφάνεια, απαιτεί φυσικά την υλοποίηση ενός κυκλωματικού μηχανισμού πρόκλησης / απόκρισης. Όσον αφορά την απόδοση, ο υπολογισμός της απόκρισης θα μπορούσε υπό συνθήκες να απαιτήσει σημαντικό χρόνο. Σε κάθε περίπτωση μια ΦΜΚΣ δεν πρέπει να απαιτεί σημαντικούς πόρους και χρόνο για την εκτέλεσή της.

3.1.6 Απόδειξη επιθέσεων (Tamper-Evident)

Μια ΦΜΚΣ πρέπει να είναι ευαίσθητη σε οποιαδήποτε προσπάθεια αλλοίωσης του κυκλώματος και να αλλάζει οριστικά τα ζεύγη πρόκλησης / απόκρισης, ώστε να παρέχει απόδειξη επίθεσης.

3.2 Μέτρηση ποιότητας σε ΦΜΚΣ πυριτίου

Στα επόμενα κεφάλαια θα δούμε πολλές αρχιτεκτονικές ΦΜΚΣ και κάθε μία από αυτές προσπαθεί να βελτιώσει μία από τις ιδιότητες που απεικονίζονται στην προηγούμενη ενότητα. Αυτό έχει δημιουργήσει μια ανάγκη για δίκαιες μετρήσεις για την σύγκριση της ποιότητας τον προτεινόμενων ΦΜΚΣ, αυτές οι μετρήσεις έχουν οδηγήσει σε ορισμένες παραμέτρους ποιότητας που πρέπει να τηρούνται από της ΦΜΚΣ.

3.2.1 Μοναδικότητα - Uniqueness

Μπορούμε να εκτιμήσουμε τη μοναδικότητα με τον τύπο

$fHD(r_i,r_j) = \frac{1}{N} \sum_{N=0}^{N-1}(ri, n \oplus rj, n)$

fDH= fractional Hamming Distance $r_{i,j}$=bit responses

Εάν οι ΦΜΚΣ παρέχουν ομοιόμορφα κατανεμημένα και ανεξάρτητα bits απόκρισης, τότε, η μοναδικότητα αποδεικνύεται ότι είναι κοντά στο 50 % κατά μέσο όρο. Τιμές υψηλότερες ή χαμηλότερες από 50 % είναι συμπτώματα χαμηλότερης διακριτότητας τσιπ.

3.2.2 Αξιοπιστία - Reliability

Στην ιδανική περίπτωση, μια ΦΜΚΣ θα πρέπει πάντα να μπορεί να αναπαράγει ακριβώς την ίδια απάντηση όταν εφαρμόζεται η ίδια πρόκληση. Ωστόσο, δεδομένου ότι οι ΦΜΚΣ βασίζονται σε αλλαγές των ηλεκτρικών χαρακτηριστικών, ένας αριθμός δυαδικών ψηφίων απόκρισης μπορεί να αλλάξει υπό σταθερές ή μεταβλητές περιβαλλοντικές συνθήκες, όπως η θερμοκρασία και η παροχή ρεύματος. Για το σκοπό αυτό, η μέτρηση σταθερότητας μπορεί να χρησιμοποιηθεί για την εκτίμηση του ποσοστού των bit σε μια απόκριση που αλλάζει την τιμή μεταξύ των απαντήσεων που λαμβάνονται από μια επανειλημμένα εφαρμοζόμενη πρόκληση.

Reliability (d) = 100 − Stability (d) = $(1-\frac{1}{M}\sum_{J=1}^{M} fDH(r_d, r'_{d,j}))*100\%$
d=device M=number of measurements $r_{d,j}$= bit responses

Ένα ΦΜΚΣ με σταθερές αποκρίσεις επιτυγχάνει υψηλή τιμή αξιοπιστίας, επομένως η τιμή του πρέπει να είναι όσο το δυνατόν πιο κοντά στο 100%.

3.2.3 Ομοιομορφία - Uniformity

Ένα ΦΜΚΣ αναμένεται να παράγει αποκρίσεις που περιέχουν ιδανικά τον ίδιο αριθμό από λογικά - 0 και λογικά - 1. Επομένως, η μέτρηση ομοιομορφίας μπορεί να αξιοποιηθεί για να εκτιμηθεί η κατανομή τον λογικά - 0 και τον λογικά - 1 στις αποκρίσεις ΦΜΚΣ. Έστω N ο αριθμός των δυαδικών ψηφίων απόκρισης, το ποσοστό μέτρησης για την ομοιομορφία της απόκρισης μπορεί να οριστεί ως :

Uniformity = $\frac{1}{N}\sum_{n=0}^{N-1} r_{I,n}$ *100 %.
r_j= bit responses

Μια τιμή 100 % σημαίνει ότι όλα τα δυαδικά ψηφία απόκρισης είναι λογικά - 1. Για αληθινά τυχαία bits, η ομοιομορφία πρέπει να είναι όσο το δυνατόν πιο κοντά στην ιδανική τιμή του 50 %. Έστω R ο αριθμός των αποκρίσεων, που προκύπτουν από το προϊόν μεταξύ της ποσότητας διαφορετικών περιπτώσεων ΦΜΚΣ και των προκλήσεων εισόδου (εάν υπάρχουν). Η μέση ομοιομορφία για R συσκευές μπορούν να υπολογιστούν ως :

Uniformity = $\frac{1}{R}\sum_{i-1}^{E}$ Uniformity

3.2.4 Αλλοίωση Bit - Bit Aliasing

Η μέτρηση ομοιομορφίας δεν είναι αρκετή για να χαρακτηρίσει την τυχαιότητα των αποκρίσεων ΦΜΚΣ. Πράγματι, ακόμη και με μια καλύτερη τιμή ομοιομορφίας bit, ορισμένα bit θα μπορούσε να αποδειχθεί ότι έχουν προκαθορισμένη τιμή όσων αφορά στης απάντησης των ΦΜΚΣ. Αυτό θα μπορούσε να συμβεί κάθε φορά που η διαδικασία κατασκευής εισάγει στατικές αλλαγές που θέτουν σε κίνδυνο όλα τα bits στις αποκρίσεις, προκαλώντας μια σταθερή προτιμώμενη τιμή. Για το σκοπό αυτό, μπορούμε να υπολογίσουμε το bit aliasing.

Bit-Aliasing (n) = $\frac{1}{N}\sum_{i=0}^{R-1} r_{i,n}$*100%,$\forall n$

r_j =bit responses

Εάν ορισμένα bit είναι προκαθορισμένα, το bit-aliasing έχει ως αποτέλεσμα μια τιμή που απέχει πολύ από 50 %.

3.3 ΦΜΚΣ καθυστέρησης - Delay-based PUF

Οι ΦΜΚΣ που εκμεταλλεύονται τη μέτρηση καθυστέρησης για την εξαγωγή απαντήσεων κατηγοριοποιούνται ως βασισμένα σε καθυστέρηση. Μπορούμε να ορίσουμε ένα ΦΜΚΣ καθυστέρησης ως έναν ψηφιακό διαγωνισμό μεταξύ δύο διαδρομών που πρέπει να επιλυθούν από ένα κύκλωμα, το λεγόμενο Arbiter (διαιτητής), το οποίο αποφάσισε ποια διαδρομή κέρδισε τον διαγωνισμό. Τα μονοπάτια πρέπει να σχεδιάζονται συμμετρικά έτσι ώστε να χαρακτηρίζονται από την ίδια καθυστέρηση. Με αυτόν τον τρόπο, ο νικητής δεν μπορεί να καθοριστεί από νωρίς, αλλά μόνο όταν κατασκευαστεί το τσιπ. Πράγματι, οι παραλλαγές που εισάγονται κατά τη διαδικασία παραγωγής τροποποιούν τις φυσικές παραμέτρους του τσιπ με τυχαίο τρόπο και αποφασίζουν την ακριβή καθυστέρηση που χαρακτηρίζει κάθε διαδρομή και, ως εκ τούτου, το αποτέλεσμα κάθε κυκλώματος ΦΜΚΣ.

3.3.1 Διαιτητής ΦΜΚΣ - Arbiter PUF

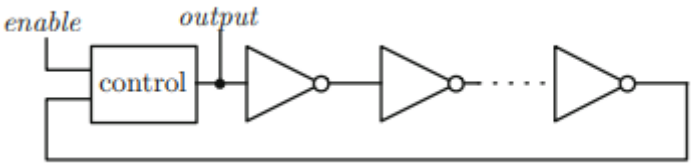
Το Arbiter PUF εισήχθη από τον Lim et al και ο μηχανισμός του βασίζεται στον προηγουμένως καθιερωμένο διαγωνισμό καθυστέρησης. Για να έχουν περισσότερες διαθέσιμες διαδρομές για σύγκριση αποτελείται από μια αλληλουχία μπλοκ διακοπτών. Κάθε μπλοκ διακόπτης μπορεί να διαδώσει ένα σήμα εισόδου μέσω δύο διαμορφώσιμων διαδρομών, δηλαδή συνδεδεμένων σε ευθεία ή με μεταγωγή. Ένα bit διαμόρφωσης καθορίζει σε ποια διαμόρφωση πρέπει να λειτουργήσει το μπλοκ. Ένας Διαιτητής, συγκεκριμένα ένα flip-flop, παράγει μια έξοδο που εξαρτάται από τη διαφορά μεταξύ των δύο καθυστερήσεων διάδοσης, η έξοδος θα είναι λογικό - 1 εάν το σήμα που κινεί την είσοδο δεδομένων (D) είναι γρηγορότερο από το σήμα ρολογιού, διαφορετικά θα είναι λογικό - 0. Οι προκλήσεις που θέτουν την καρδινότητα αυξάνονται εκθετικά με τον αριθμό των μπλοκ διακοπτών, με μπλοκ διακόπτη N, το κύκλωμα είναι σε θέση να συγκρίνει 2N διαφορετικά ζεύγη διαδρομών.

Το κύκλωμα μπορεί να υποφέρει από κατάσταση μετασταθερότητας εάν η αντιστάθμιση μεταξύ των καθυστερήσεων είναι κοντά στο 0: μια τέτοια παραβίαση χρόνου θα προκαλέσει τυχαία συμπεριφορά επειδή η έξοδος δεν είναι ντετερμινιστική.

3.3.2 Ταλαντωτής δακτυλίου ΦΜΚΣ - Ring Oscillator PUF

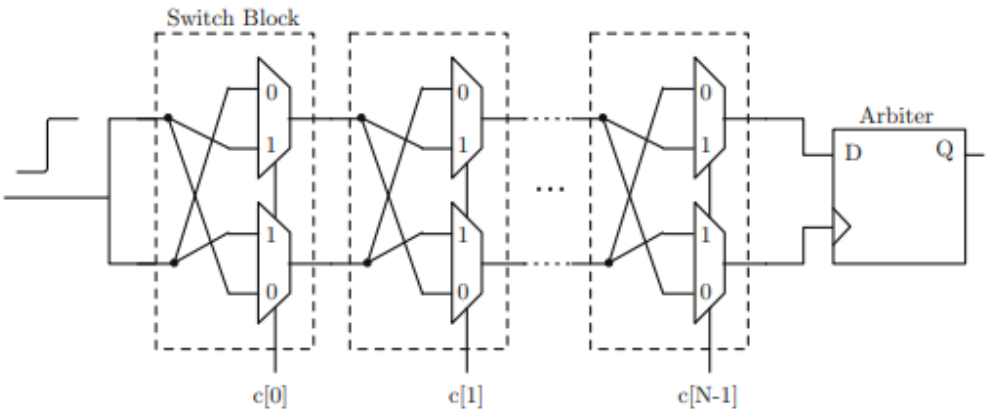
Διαφορετικά από το Διαιτητή ΦΜΚΣ, ο Ταλαντωτής Δακτυλίου ΦΜΚΣ (RO PUF) αξιολογεί τις διαφορές καθυστέρησης μέσω μετρήσεων συχνότητας αξιοποιώντας

ασύγχρονους βρόχους ταλάντωσης. Κάθε βρόχος περιέχει έναν περιττό αριθμό αναστροφικών σταδίων, έτσι ώστε, μόλις ενεργοποιηθεί το κύκλωμα, να αρχίσει να ταλαντεύεται απεριόριστα.



Εικόνα 1 Ταλαντωτής δακτυλίου

Στην εικόνα 1 αναφέρεται ένας γενικός ελεγχόμενος ταλαντωτής δακτυλίου, η έξοδος του τελευταίου σταδίου του δακτυλίου επαναφέρεται στην πύλη ελέγχου, μαζί με το σήμα ενεργοποίησης. Εάν ο αριθμός των σταδίων αναστροφής στον βρόχο είναι ζυγός, η πύλη ελέγχου είναι and και, διαφορετικά είναι ένα nand. Ο βρόχος μπορεί επίσης να αποτελείται από μόνο ένα στάδιο αναστροφής και άλλα στοιχεία καθυστέρησης, όπως buffer. Σε αντίθεση με το Διαιτητή ΦΜΚΣ, το οποίο απαιτεί τέλεια συμμετρία δύο γραμμών καθυστέρησης, το RO PUF απαιτεί κυκλώματα ταλαντωτών δακτυλίων εξίσου διαμορφωμένα. Για το λόγο αυτό, το RO PUF μπορεί εύκολα να πραγματοποιηθεί σε κάθε τεχνολογία πυριτίου, συμπεριλαμβανομένου του FPGA. Το RO PUF αποτελείται από N πανομοιότυπους ταλαντωτές δακτυλίων, δύο πολυπλέκτες που επιλέγουν ένα ζεύγος μεταξύ τους και δύο μετρητές που μετρούν τις συχνότητες του ζεύγους. Λόγω των ατελειών κατασκευής, η συχνότητα ταλαντώσεων κάθε βρόχου είναι τυχαία και εξαρτάται από τη συσκευή, επομένως διαφέρει ελαφρώς από την ιδανική τιμή συχνότητας. Η μέτρηση συχνότητας περιλαμβάνει επίσης έναν άλλο μετρητή, οδηγούμενο από το ρολόι του συστήματος, ο οποίος καθορίζει το χρονικό παράθυρο στο οποίο πρέπει να μετρηθούν οι άκρες ταλαντώσεων.

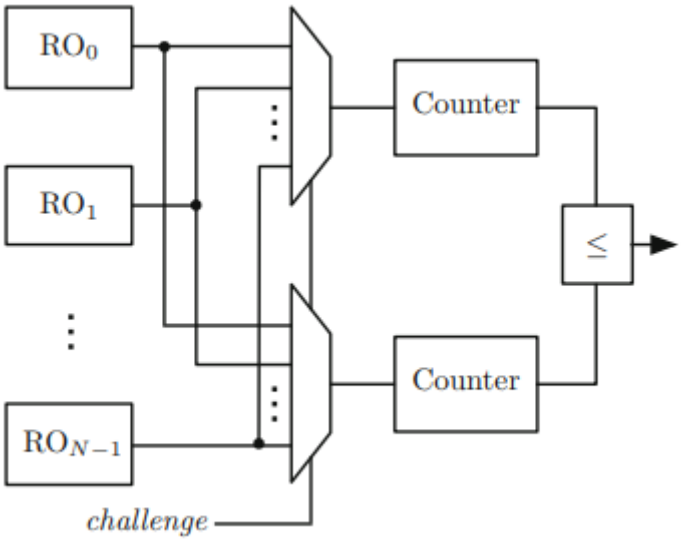


Εικόνα 2 RO

Μια πρόκληση αποφασίζει ποιο ζεύγος ταλαντωτή δακτυλίου πρέπει να μετρηθεί. Μόλις εξαχθούν οι συχνότητες, ένα κύκλωμα τις συγκρίνει δίνοντας λογικά - 1 ή λογικά - 0. Η στρατηγική πλήρους αντιστοίχησης επιτρέπει τη σύγκριση N (N − 1) /2 διαφορετικών ζευγών ταλαντωτών δακτυλίων, ωστόσο η εντροπία που είναι σε θέση να παράγει το κύκλωμα είναι μικρότερη από N (N − 1) /2, δεδομένου ότι ορισμένα bit που λαμβάνονται

από συγκρίσεις συχνοτήτων συσχετίζονται. Για να μεγιστοποιηθεί η εντροπία, οι στρατηγικές σύζευξης πρέπει να επιλέξουν μόνο.

Μόλις μετρηθούν οι καθυστερήσεις, αθροίζονται και τα δυαδικά ψηφία πρόκλησης καθορίζουν το πρόσημο κάθε καθυστέρησης.



Εικόνα 3 RO

Μια πιο σύνθετη στρατηγική σύζευξης έχει προταθεί από το Yin and Qu, η οποία εκμεταλλεύεται έναν αλγόριθμο διαδοχικής σύζευξης που δημιουργεί αξιόπιστα bits. Συγκεκριμένα, η πρόταση δίνει N/2 απαντήσεις bits από N ταλαντωτές δακτυλίου. Μια άλλη στρατηγική ζεύξης, η κωδικοποίηση γείτονα σαν αλυσίδα, έχει προταθεί. Αποτελείται από δύο αρχές σχεδιασμού:

1. Οι Ταλαντωτές Δακτύλου πρέπει να τοποθετούνται όσο το δυνατόν πιο κοντά μεταξύ τους για να ελαχιστοποιηθούν οι επιδράσεις συστηματικών διαφοροποιήσεων.

2. Η σύζευξη πρέπει να λαμβάνει υπόψη μόνο τον παρακείμενο ταλαντωτή δακτυλίου, επομένως παράγει μόνο N - 1 δυαδικά ψηφία.

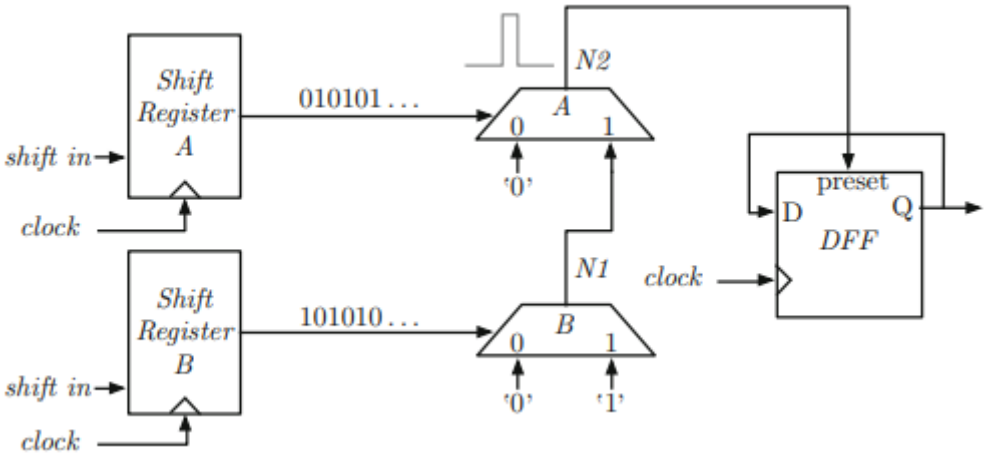
Η δοκιμή αυτής της στρατηγικής στα FPGA οδηγεί σε βελτίωση της μοναδικότητας.

Εκτός από τη στρατηγική σύζευξης και το μέγεθος της πρόκλησης, η συχνότητα ταλαντωτή δακτυλίου χαρακτηρίζεται από υψηλή ευαισθησία σε περιβαλλοντικές αλλαγές, όπως η θερμοκρασία του πύρινα και η τάση εργασίας, που θα μπορούσαν να προκαλέσουν αστάθεια στις αποκρίσεις, ειδικά για ζεύγη των οποίων οι συχνότητες είναι πραγματικά παρόμοιες. Μέχρι στιγμής αυτό το πρόβλημα έχει προκύψει και έχει αντιμετωπιστεί με ένα σχέδιο κάλυψης 1-out-of-k, το οποίο έχει ως στόχο να επιλέξει ζεύγη των οποίων η απόσταση είναι η μέγιστη μεταξύ των k ζευγών. Η γενική επιβάρυνση που εισάγεται μπορεί να εκτιμηθεί ως 2 k · 100 %, αφού η αρχιτεκτονική απαιτεί μόνο 2 ζεύγη k για την εξαγωγή ενός bit απόκρισης. Η διαμορφώσιμη δομή του βρόχου περιέχει έξι μετατροπείς, επιλεγμένους από τους πολυπλέκτες, οι οποίοι είναι σε θέση να ορίσουν οκτώ διαφορετικούς ταλαντωτές δακτυλίων. Η περιοχή πάνω από

αυτήν τη διαμόρφωση είναι η ίδια ενός κανονικού ταλαντωτή δακτυλίου, αφού η διαμορφώσιμη αρχιτεκτονική καταλαμβάνει ολόκληρο το CLB.

3.3.3 Anderson ΦΜΚΣ - Anderson PUF

Το Anderson PUF είναι μια αρχιτεκτονική ΦΜΚΣ που σχεδιάστηκε για να συνδεθεί σε συσκευές Xilinx Virtex-5. Το Anderson PUF είναι μια σύνθεση βασικών στοιχείων, που ορίζονται ως κύτταρα Anderson, και το καθένα εξάγει μόνο 1-bit απόκριση ΦΜΚΣ. Το κελί περιέχει δύο καταχωρητές μετατόπισης, δύο πολυπλέκτες 2 προς 1 και ένα D flip-flop. Η είσοδος δεδομένων 0 των πολυπλεκτών είναι κολλημένη στο λογικό - 0, η επιλεγμένη είσοδος οδηγείται από τις εξόδους των καταχωρητών μετατόπισης, οι οποίες είναι σύγχρονες με το ίδιο σήμα ρολογιού και να δημιουργήσει μια αντιφάση από λογικά-1 και λογικά-0 ακολουθία. Ο πολυπλέκτης B έχει την είσοδο δεδομένων 1 συνδεδεμένη με τα λογικά-1 και η έξοδός του N1 συνδέεται με την είσοδο δεδομένων 1 του πολυπλέκτη A.



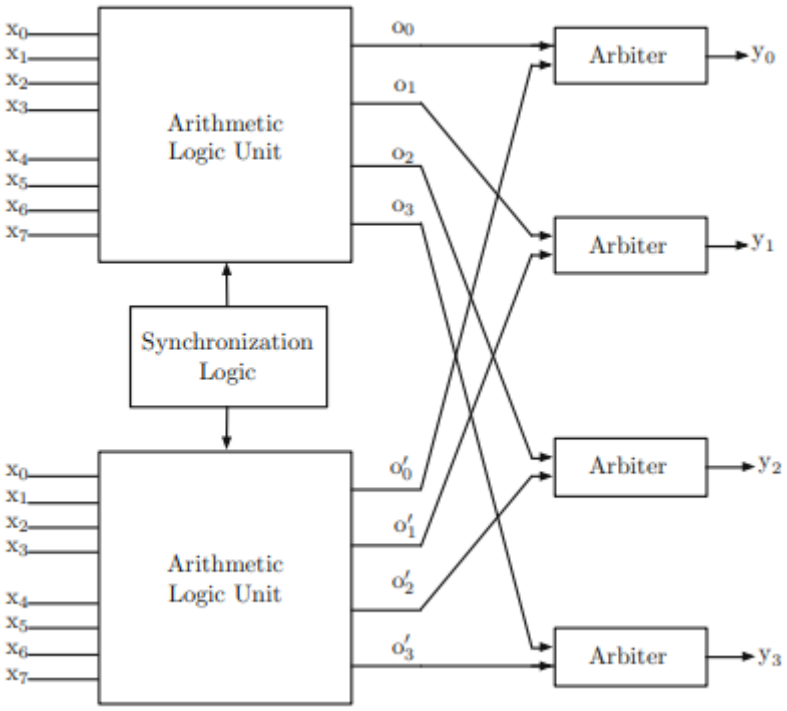
Εικόνα 4 ADERSON PUF

Η εικόνα 4 Περιγράφει όλες τις συνδέσεις εντός του κελιού Anderson PUF. Λόγω της παράλληλης κατασκευής, οι έξοδοι των καταχωρητών μετατόπισης δεν είναι τέλειοι στην αντίστροφη τους, αλλά υπάρχουν κάποιες επικαλύψεις κατά τη διάρκεια της μεταγωγής που δημιουργούν θετικές δυσλειτουργίες. Αλλαγή σε λογικό - 1 από λογικό - 0 από ένα θετικό σφάλμα που εμφανίζεται στην προκαθορισμένη θύρα. Επιπλέον, το flip-flop έχει διαμορφωθεί ως one-catcher. Οι λειτουργίες μετατόπισης του Anderson PUF υλοποιούνται χρησιμοποιώντας δύο SLICEM LUT, διαμορφωμένες ως καταχωρητές αλλαγής 16 bit. Οι δύο πολυπλέκτες διασυνδέονται χρησιμοποιώντας την αλυσίδα μεταφοράς και το flip-flop μπορεί να βρίσκεται στο ίδιο κομμάτι των άλλων στοιχείων. Επομένως, το κελί Anderson PUF θα μπορούσε θεωρητικά να εφαρμοστεί χρησιμοποιώντας μόνο ένα κομμάτι, αλλά πρέπει να ληφθεί υπόψη η διαμόρφωση του σφάλματος. Για το σκοπό αυτό, ο Anderson αύξησε το πλάτος του παλμού με δυσλειτουργία μεταβάλλοντας την απόσταση μεταξύ των δύο πολυπλέκτων. Η καλύτερη διαμόρφωση όσον αφορά την ποιότητα των αποκρίσεων αποτελείται από πέντε ενδιάμεσους πολυπλέκτες αλυσίδας μεταφοράς.

Τα πειραματικά αποτελέσματα έδειξαν ότι οι αποκρίσεις ΦΜΚΣ έχουν καλή συμπεριφορά όταν αλλάζει η θερμοκρασία εργασίας, δηλαδή η σταθερότητα ΦΜΚΣ έχει καλή αξία για να χρησιμοποιηθεί με ασφαλές στο FPGA.

3.3.4 ALU PUF

Το ALU PUF, που εισήχθη από τον Kong et al, είναι ένα ΦΜΚΣ με καθυστέρηση που εκμεταλλεύεται δύο συμμετρικές αριθμητικές λογικές μονάδες (ALU). Λόγω καθυστερημένων αναντιστοιχιών που εισάγονται από τη μεταβλητότητα της κατασκευής, οι ALU εκτελούν λειτουργίες με διαφορετικό χρόνο. Αυτό ισχύει επίσης για την αλυσίδα μεταφοράς των αθροιστών μεταφοράς κυματισμού, ένα θεμελιώδες κύκλωμα για τις ALU. Στην εικόνα 5 εικονίζεται ένα αρχιτεκτονικό σχήμα υψηλού επιπέδου ενός 4-bit ALU PUF. Η αναζήτηση σε ALU με τους ίδιους τελεστές (πρόκληση x0, . . . , x7) και η αξιολόγηση των διαφορών στην καθυστέρηση διάδοσης, καθιστούν δυνατή τη δημιουργία bits απόκρισης (y0, ... , y3) με ελάχιστη επιβάρυνση υλικού, δηλαδή λογική συγχρονισμού, για να εξασφαλιστεί που εισάγει



Εικόνα 5 ALU

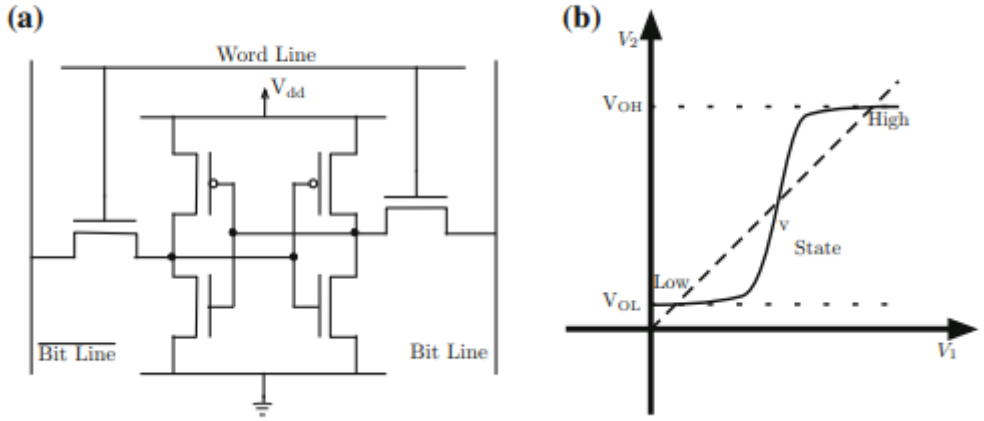
3.4 ΦΜΚΣ μνήμης - Memory-Based PUF

Ένα ΦΜΚΣ που βασίζεται στη μνήμη χρησιμοποιεί την τυχαία αρχική κατάσταση ενός κελιού μνήμης κατά την εκκίνηση μιας συσκευής για να εξαγάγει μια υπογραφή που εξαρτάται από τη συσκευή. Σε αντίθεση με τα προηγούμενα ΦΜΚΣ καθυστέρησης, οι

ΦΜΚΣ μνήμης δεν απαιτούν συγκεκριμένο σχεδιασμό για τη δημιουργία υπογραφής, αλλά λειτουργούν με τυπικά κελιά μνήμης.

3.4.1 ΣΜΤΠ ΦΜΚΣ -SRAM PUF

Η στατική μνήμη τυχαίας πρόσβασης (SRAM) ΦΜΚΣ, εκμεταλλεύεται την τιμή εκκίνησης των κελιών SRAM, τα οποία είναι μπλοκ μνήμης που πραγματοποιούνται με δύο μετατροπείς διασταυρούμενης ζεύξης, για να δημιουργήσει ένα μοναδικό και μη κλωνοποιήσιμο δακτυλικό αποτύπωμα. Στην τεχνολογία CMOS, όπως φαίνεται στην εικόνα 6, κάθε κύτταρο απαιτεί έξι τρανζίστορ και η δομή πραγματοποιείται συμμετρικά με δύο μισά. Κατά την εκκίνηση, σε κάθε κύτταρο φτάνει και διατηρεί την αρχική κατάσταση ακόμη και αν τα τρανζίστορ δεν οδηγούνται απευθείας από εξωτερικά σήματα. Μόλις τροφοδοτηθούν, τα δύο μισά χαρακτηρίζονται από ένα ασταθές σημείο τάσης και, λόγω των μικρών παραλλαγών που εισάγονται από τη διαδικασία κατασκευής, αναγκάζουν το ένα το άλλο να φτάσει σε ένα από τα δύο πιθανά σταθερά σημεία, χαμηλής ή υψηλής κατάστασης, ενισχύοντας τις διαφορές τάσεων, όπως απεικονίζεται στην εικόνα 6. Υπάρχουν τρεις τύποι κυττάρων SRAM:



Εικόνα 6 SPAM

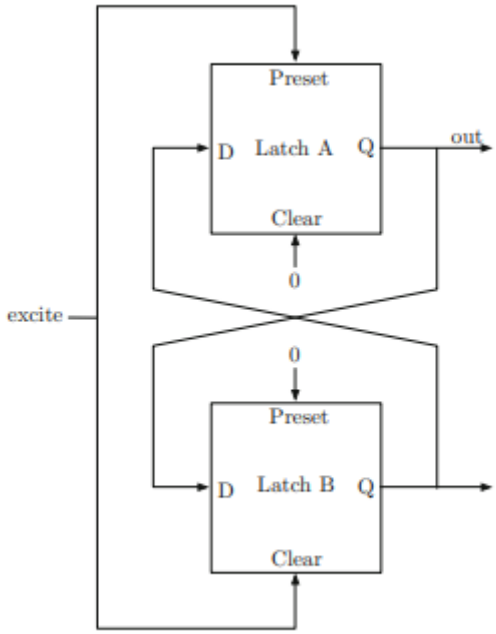
a) μη κυρτά κύτταρα, που χαρακτηρίζονται από τυχαία συμπεριφορά, αφού η επίδραση των παραλλαγών παραγωγής στα δύο μισά αποδεικνύεται αμοιβαία εξουδετερωμένη

b) μερικώς κεκλιμένα κύτταρα, που έχουν ασυμφωνία μεταξύ των δύο μετατροπέων, έτσι ώστε να έχουν μια προτιμώμενη κατάσταση, αλλά οι εξωτερικές συνθήκες μπορεί να προκαλέσουν μια ανατροπή της κατάστασης.

c) τα πλήρως λοξά κύτταρα, όπως και τα μερικώς στραμμένα, χαρακτηρίζονται από μια προτιμώμενη κατάσταση η οποία δεν αλλάζει, ακόμη και υπό διαφορετικές εξωτερικές συνθήκες. Οι Maes et al απεικόνισαν μια προσέγγιση για την εφαρμογή ενός αλγορίθμου χαμηλής επιβάρυνσης για την επίτευξη υψηλής σταθερότητας απόκρισης SRAM PUF.

3.4.2 ΦΜΚΣ Πεταλούδας και Flip-Flop - Butterfly and Flip-Flop PUF

Προκειμένου να υπάρχει διαθέσιμο ΦΜΚΣ μνήμης στο FPGA και να αντιμετωπιστεί τα μειονέκτημα του SRAM ΦΜΚΣ, ο Kumar et al εισήγαγε την ΦΜΚΣ Πεταλούδας, το οποίο είναι ένα ΦΜΚΣ που βασίζεται σε στοιχεία μνήμης σταυρωτά συνδεδεμένα. Συγκεκριμένα, υιοθέτησε δύο διαφανείς μανδαλωτες διασταυρούμενων συζεύξεων που έχουν διαμορφωθεί όπως φαίνεται στην εικόνα 7. Κάθε μανταλωτής έχει μια προεπιλογή και μια σαφή είσοδο, που λειτουργούν ασύγχρονα. Η προεπιλογή του ενός μανδαλωτή και η απόσταση του άλλου οδηγούνται από το ίδιο σήμα, που ονομάζεται διέγερση, ενώ η είσοδος D οδηγείται από το Q του άλλου μανδαλωτή. Όταν επιβεβαιώνεται το σήμα διέγερσης, το κύκλωμα βρίσκεται σε ασταθή κατάσταση λόγω των μανδαλωτών που έχουν αντίθετες καταστάσεις. Στην ιδανική περίπτωση, το κύκλωμα θα πρέπει να διατηρεί επ' αόριστον αυτήν την κατάσταση, αλλά μόλις η διέγερση ρυθμιστεί στο χαμηλό, το κύκλωμα τείνει σε μια από τις δύο σταθερές καταστάσεις λόγω φυσικών αναντιστοιχιών. Οι καταχωρημένες καταστάσεις μπορούν να χρησιμοποιηθούν ως κομμάτι απόκρισης ΦΜΚΣ.



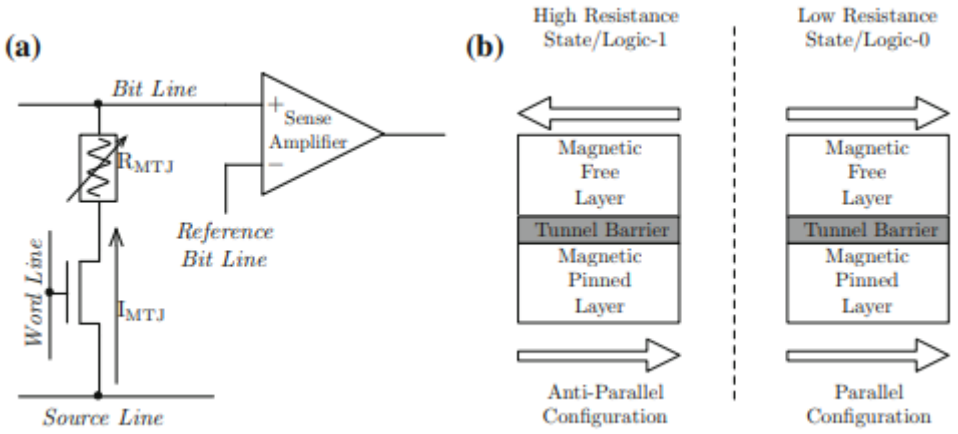
Εικόνα 7 butterfly

Ισοδύναμα με τα SRAM ΦΜΚΣ, ο Maes et al πρότεινε την ανάγνωση των τιμών εκκίνησης των flip-flop σε FPGA. Η τιμή αρχικοποίησης flip-flop καθορίζεται από αυτή η τιμή δεν φορτώνεται αμέσως στο flip-flop, αλλά μόνο μετά την πλήρη φόρτωση της νέας διαμόρφωσης στο FPGA. Πράγματι, μόλις ολοκληρωθεί η διαμόρφωση, επιβεβαιώνεται η γενική γραμμή επαναφοράς, προκαλώντας την επαναφορά των flip-flop στην τιμή που καθορίζεται στο bitstream. Βασικά, η ιδέα του Flip-Flop ΦΜΚΣ βασίζεται στην απενεργοποίηση της λογικής επαναφοράς των κανονικών flip-flop όταν προγραμματίζεται η συσκευή.

3.4.3 STT-MRAM PUF

Τα SRAM PUF είναι μία από τις πιο διερευνημένες λύσεις, καθώς, όπως αναμενόταν προηγουμένως, δεν απαιτούν πρόσθετη επιβάρυνση υλικού για την εξαγωγή μοναδικών και μη κλωνοποιήσιμων δακτυλικών αποτυπωμάτων από ολοκληρωμένα κυκλώματα. Επιπλέον, η ποιότητα και η σταθερότητα των αποκρίσεων τις καθιστά πραγματικά ελκυστικές, σε σύγκριση με άλλες αρχιτεκτονικές ΦΜΚΣ. Πρόσφατα, η εστίαση κινείται προς αναδυόμενες τεχνολογίες μνήμης, όπως οι μαγνητικές αντιστασιακές μνήμες. Ο Vatajelu et al πρότεινε μια πρωτοποριακή αρχιτεκτονική ΦΜΚΣ, η οποία εκμεταλλεύεται τη μεταβλητότητα κατασκευής της Spin-Transfer Torque Magnetic RAM. Πράγματι, η ηλεκτρική αντίσταση της διασταύρωσης μαγνητικού τούνελ στην αντιπαράλληλη μαγνήτιση υποφέρει πολύ από την ατέλεια που προκαλείται από τη διαδικασία κατασκευής.

Στην εικόνα 8 περιγράφονται λεπτομερώς ένα κύκλωμα κυττάρων STT-MRAM και η διαμόρφωση στην οποία αποθηκεύει την λογική τιμή - 0 και λογική τιμή - 1. Για την εξαγωγή της αποθηκεμένης τιμής, επομένως για την αξιολόγηση της αντίστασης MTJ, ένα ρεύμα αναφοράς εμπλέκεται στη διαδικασία και δημιουργείται από κελιά αναφοράς. Τα ενεργά και τα κύτταρα αναφοράς κατασκευάζονται ταυτόσημα.



Εικόνα 8 STT-MRAM

Η ΦΜΚΣ εκμεταλλεύεται τη μεταβλητότητα των αντιστάσεων MTJ στην αντιπαράλληλη διαμόρφωση. Για την εξαγωγή ενός μοναδικού δακτυλικού αποτυπώματος, όλα τα κελιά διαμορφώνονται σε αντιπαράλληλη κατάσταση (γράφοντας λογικές τιμές -1) και τα ενεργά διαβάζονται μέσω του ενισχυτή αίσθησης, συγκρίνοντας το ρεύμα τους με το ρεύμα των κελιών αναφοράς. Η κατάσταση των ενεργών κελιών θα ερμηνευτεί ως λογικο-0 εάν IMTJ> IRef, διαφορετικά ως λογικο-1 εάν IMTJ <IRef. Ακόμα κι αν όλα τα κελιά έχουν ρυθμιστεί σε κατάσταση λογικής τιμής-1, καθώς οι αντιστάσεις MTJ κανονικά κατανέμονται, στατιστικά τα μισά από αυτά θα ερμηνευτούν ως λογικό-0.

4 ModelSim

Το ModelSim συνδυάζει υψηλή απόδοση όσον αφορά στην επεξεργασία των προσομοιώσεων και δυνατότητες εντοπισμού σφαλμάτων, χαρακτηριστικά που απαιτούνται για την προσομοίωση μεγαλύτερων συστημάτων. Η υποστήριξη Verilog και VHDL δίνει την δυνατότητα στο ModeSim για επαλήθευση σχεδιασμού ενός Project που χρησιμοποιεί τις δυο γλώσσες, χωρίς να χρειάζονται άλλα προγράμματα επαλήθευσης.

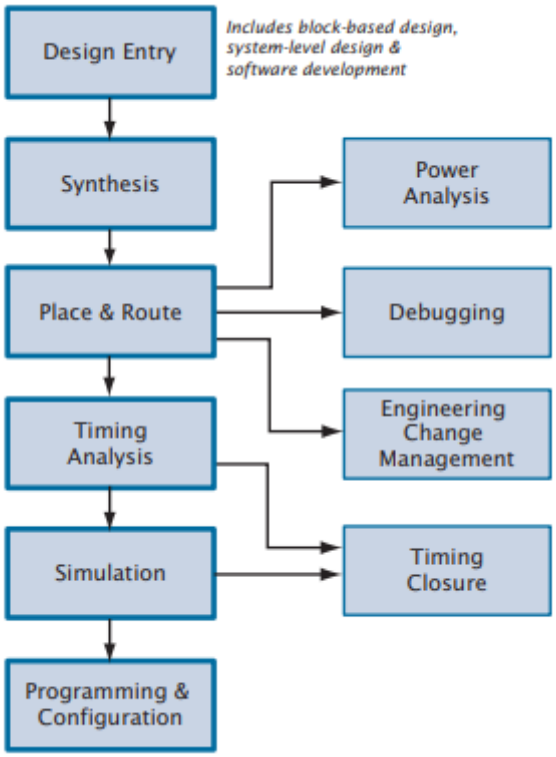
Μέσα από το ModelSim θα μπορέσουμε να δούμε την λειτουργεία του κάθε κομματιού του project από την μεταγλώττιση του ποιο μικρού κομματιού μέχρι και την μεταγλώττιση ολοκλήρου του project, είτε μέσα από το Quartus, είτε από το πρόγραμμα του ModelSim που μας παρέχει περισσότερες δυνατότητες για μεγάλα και πολύπλοκα προγράμματα.

Το ModelSim μας επιτρέπει να εντοπίζουμε γρήγορα προβλήματα, με τη βοήθεια δυναμικά ενημερωμένων παραθύρων. Για παράδειγμα, η επιλογή ενός design στο structure ενημερώνει αυτόματα το source, signals, process και variables. Αυτά δημιουργούν ένα εύχρηστο περιβάλλον εντοπισμού σφαλμάτων. Μόλις εντοπιστεί πρόβλημα, μπορούμε να επεξεργαστούμε, να μεταγλωττίσουμε και να προσομοιώσουμε ξανά χωρίς να φύγουμε από τον προσομοιωτή. Το ModelSim υποστηρίζει πλήρως τα τρέχοντα πρότυπα γλώσσας VHDL και Verilog.

Η διάταξη των παραθύρων, οι γραμμές εργαλείων και το μενού καθιστά εύκολη την προβολή και την πρόσβαση στις πολλές δυνατότητες του ModelSim. Το αποτέλεσμα είναι ένα πρόγραμμα πλούσιο σε χαρακτηριστικά που είναι εύκολο στη χρήση.

5 Quartus

Το Intel Quartus II περιλαμβάνει όλα όσα χρειαζόμαστε για να σχεδιάσουμε ένα κύκλωμα σε ένα FPGA, από την εισαγωγή και τη σύνθεση του σχεδιασμού έως τη βελτιστοποίηση, την επαλήθευση και την προσομοίωση. Το Intel Quartus II παρέχει ένα πλήρες περιβάλλον σχεδιασμού πολλαπλών πλατφορμών που προσαρμόζεται εύκολα στις συγκεκριμένες σχεδιαστικές μας ανάγκες. Είναι ένα ολοκληρωμένο περιβάλλον για το σχεδιασμό συστήματος σε προγραμματιζόμενο τσιπ. Το λογισμικό Quartus II περιλαμβάνει λύσεις για όλες τις φάσεις του σχεδιασμού FPGA.



Εικόνα 9 quartus design flow

Επιπλέον, το λογισμικό Quartus II μάς επιτρέπει να χρησιμοποιούμε το γραφικό περιβάλλον εργασίας χρήστη του Quartus II και την γραμμή εντολών για κάθε φάση της ροής σχεδιασμού. Μπορούμε να χρησιμοποιήσουμε μία από αυτές για ολόκληρη τη ροή ή μπορούμε να χρησιμοποιήσουμε διαφορετικές επιλογές σε διαφορετικές φάσεις.

Για να χρησιμοποιήσουμε σωστά το Quartus II θα πρέπει να έχουμε κατεβάσει την κατάλληλη «υποστήριξη» για την συσκευή που θέλουμε να προσομοιάσουμε, επίσης πρέπει να εγκαταστήσουμε και το ModelSim-Altera Edition για να τρέξουμε την προσομοίωση και να μπορέσουμε να δούμε τα αποτελέσματα.

5.1 Chip planner

Το Chip Planner παρέχει μια οπτική απεικόνιση των τσιπ. Μπορεί να εμφανίσει λογική τοποθέτηση, περιοχές LogicLock, σχετική χρήση πόρων, λεπτομερείς πληροφορίες δρομολόγησης, διαδρομές μεταξύ καταχωρητών και κανάλια πομποδεκτών υψηλής ταχύτητας. Μπορούμε να δούμε εκτιμήσεις φυσικού χρονισμού, συμφόρηση δρομολόγησης και περιοχές ρολογιού.

Το Chip Planner μπορεί να εκτελέσει αλλαγές εκχώρησης, όπως δημιουργία και διαγραφή αναθέσεων πόρων, καθώς και αλλαγές μετά τη μεταγλώττιση, όπως δημιουργία, μετακίνηση και διαγραφή λογικών κελιών. Μπορούμε να χρησιμοποιήσουμε το Chip Planner σε συνδυασμό με το Resource Property Editor, για να αλλάξουμε τις συνδέσεις μεταξύ πόρων και να κάνουμε αλλαγές μετά τη μεταγλώττιση στις ιδιότητες των λογικών κελιών, των στοιχείων εισόδου/εξόδου και των PLL.

Με το Chip Planner, μπορούμε να προσαρμόσουμε τις υπάρχουσες αντιστοιχίσεις σε πόρους συσκευής, όπως pins και λογικά κελιά. Μπορούμε επίσης να προβάλουμε εξισώσεις και πληροφορίες δρομολόγησης και να υποβιβάσουμε τις εργασίες σύροντας και αποθέτοντας στις περιοχές Logic Lock (Standard) στο παράθυρο Regions Lock (Standard).

Το Chip Planner εμφανίζει γραφικά τη λογική τοποθέτηση, τις περιοχές Logic Lock (Standard), τη σχετική χρήση πόρων, λεπτομερείς πληροφορίες δρομολόγησης, fan-in και fan-out, διαδρομές καταχώρισης και κανάλια πομποδεκτών υψηλής ταχύτητας. Μπορούμε να δούμε εκτιμήσεις φυσικού χρονισμού, συμφόρηση δρομολόγησης και περιοχές ρολογιού.

5.2 Γλώσσα προγραμματισμού TCL

Χρησιμοποιείται συνήθως ενσωματωμένο σε εφαρμογές C, για γρήγορη δημιουργία πρωτοτύπων, σεναριακές εφαρμογές, GUI και δοκιμές. Το Tcl είναι διαθέσιμο για πολλά λειτουργικά συστήματα, επιτρέποντας στον κώδικα Tcl να τρέχει σε μεγάλη ποικιλία συστημάτων. Επειδή το Tcl είναι μια πολύ συμπαγής γλώσσα, χρησιμοποιείται σε πλατφόρμες ενσωματωμένων συστημάτων, τόσο σε πλήρη μορφή όσο και σε αρκετές άλλες εκδόσεις μικρού αποτυπώματος.

Οι εντολές Quartus II Tcl μειώνουν τις απαιτήσεις μνήμης και μας επιτρέπουν να βελτιστοποιήσουμε την απόδοση αυτοματοποιώντας κοινές εργασίες. Αυτές οι εντολές ομαδοποιούνται σε πακέτα Tcl. Για παράδειγμα, μπορούμε να χρησιμοποιήσουμε το πακέτο του project για την πραγματοποίηση και την επεξεργασία όλων των αναθέσεων σε project level και το πακέτο συσκευής για τον καθορισμό όλων των πληροφοριών συσκευής.

Το Quartus II παρέχει την κονσόλα Tcl, η οποία μας επιτρέπει να ελέγχουμε το λογισμικό Quartus II με εντολές και σενάρια Tcl. Το Tcl μπορεί να χρησιμοποιηθεί χωρίς την χρηστή του quartus αν έχουμε φτιάξει τα αρχεία που θέλουμε να επεξεργαστούμε πρώτα στο quartus, μέσα από το command prompt.

6 FPGA

6.1 Τι είναι το FPGA

Το FPGA σημαίνει Field Programmable Gate Array ή συστοιχία επιτόπια προγραμματιζόμενων πυλών. Είναι ένα ολοκληρωμένο κύκλωμα που μπορεί να προγραμματιστεί από έναν χρήστη για συγκεκριμένη χρήση μετά την κατασκευή του. Τα σύγχρονα FPGA περιέχουν προσαρμοστικές λογικές ενότητες (ALM) και λογικά στοιχεία (LE) που συνδέονται μέσω προγραμματιζόμενων διασυνδέσεων. Αυτά τα μπλοκ δημιουργούν μια φυσική σειρά λογικών πυλών που μπορούν να προσαρμοστούν για την εκτέλεση συγκεκριμένων υπολογιστικών εργασιών. Αυτό τους κάνει πολύ διαφορετικούς από άλλους τύπους μικροελεγκτών ή κεντρικών μονάδων επεξεργασίας (CPU), των οποίων η διαμόρφωση ορίζεται και σφραγίζεται από έναν κατασκευαστή και δεν μπορεί να τροποποιηθεί.

Τα πρώτα προγραμματιζόμενα κυκλώματα ήταν πολύ απλά και περιείχαν μόνο λογικές πύλες. Αυτό ήταν αρκετό για να εκτελέσει πολλές λογικές συναρτήσεις όπου μηδενικά και ένα ήταν οι είσοδοι και οι έξοδοι. Με τον καιρό, τα προγραμματιζόμενα κυκλώματα γίνονταν όλο και πιο ισχυρά. Σε προγραμματιζόμενα κυκλώματα, προγραμματίζουμε λογικά κελιά που μπορούν να λειτουργήσουν ως καταχωρητές, αθροιστές, πολυπλέκτες ή πίνακες αναζήτησης. Ο τρόπος λειτουργίας των κυψελών και η δομή του μπορούν να αλλάξουν και τα δύο ενώ το κύκλωμα λειτουργεί. Ένα κύκλωμα μπορεί να επαναπρογραμματιστεί για να εκτελεί διαφορετικές λειτουργίες, για παράδειγμα, αυτόν ενός επεξεργαστή στην αρχιτεκτονική ARM, μιας κάρτας διασύνδεσης δικτύου ή ενός κωδικοποιητή βίντεο, για να ονομάσει τρεις.

6.2 Πως δουλεύει ένα FPGA

Τα FPGA αποτελούνται από λογικές ενότητες που συνδέονται με κανάλια δρομολόγησης. Κάθε ενότητα αποτελείται από έναν προγραμματιζόμενο πίνακα αναζήτησης που χρησιμοποιείται για τον έλεγχο των στοιχείων από τα οποία αποτελείται κάθε κελί και για την εκτέλεση λογικών συναρτήσεων των στοιχείων που αποτελούν το κελί. Εκτός από τον πίνακα αναζήτησης, κάθε κελί περιέχει κατακερματισμένους αθροιστές που επιτρέπουν την προσθήκη. Η αφαίρεση μπορεί επίσης να γίνει αλλάζοντας τις λογικές καταστάσεις της εισόδου. Πέρα από αυτά, υπάρχουν επίσης καταχωρητές (λογικά στοιχεία που χρησιμοποιούνται για την εκτέλεση των απλούστερων λειτουργιών μνήμης) και πολυπλέκτες (στοιχεία εναλλαγής).

Τα FPGA μπορούν επίσης να περιλαμβάνουν στατικές και δυναμικές μνήμες στα τσιπ, ανάλογα με το συγκεκριμένο μοντέλο του κατασκευαστή. Επιπλέον, στα FPGAs μπορούμε να βρούμε έτοιμα εξαρτήματα, όπως πυρήνες CPU, ελεγκτές μνήμης, ελεγκτές USB ή κάρτες δικτύου. Αυτά τα στοιχεία είναι τόσο δημοφιλή που δεν χρειάζεται να τα εφαρμόσουμε στη δομή FPGA. Αντ' αυτού, μπορούμε να χρησιμοποιήσουμε ένα ήδη κατασκευασμένο εξάρτημα.

6.3 Που χρησιμοποιείτε ένα FPGA

Τα FPGA χρησιμοποιούνται κυρίως για το σχεδιασμό ολοκληρωμένων κυκλωμάτων ειδικά για εφαρμογές (ASIC). Πρώτον, σχεδιάζουμε την αρχιτεκτονική ενός τέτοιου κυκλώματος. Στη συνέχεια, χρησιμοποιούμε ένα FPGA για να δημιουργήσουμε και να ελέγξουμε το πρωτότυπο. Τα λάθη μπορούν να διορθωθούν. Μόλις το πρωτότυπο λειτουργήσει όπως αναμενόταν, δημιουργούμε και κατασκευάζουμε ένα έργο ASIC βάσει του σχεδιασμού FPGA. Αυτό μας επιτρέπει να εξοικονομήσουμε χρόνο, καθώς η κατασκευή ενός ολοκληρωμένου κυκλώματος μπορεί να είναι μια πολύ περίπλοκη και χρονοβόρα διαδικασία. Εξοικονομεί επίσης χρήματα, καθώς ένα FPGA μπορεί να χρησιμοποιηθεί για την προετοιμασία πολλών επαναλήψεων του ίδιου έργου. Σε αυτό το πλαίσιο αξίζει να αναφερθεί ότι οι Σύγχρονες Μονάδες Επεξεργασίας Tensor (TPU) ή η εξόρυξη κρυπτονομισμάτων σχεδιάστηκαν αρχικά ως FPGA και κατασκευάστηκαν μόνο μετά τον έλεγχο τους.

Τα FPGA χρησιμοποιούνται επίσης σε συστήματα πραγματικού χρόνου όπου ο χρόνος απόκρισης παίζει καθοριστικό ρόλο. Στους τυπικούς επεξεργαστές, ο χρόνος απόκρισης δεν έχει οριστεί και δεν γνωρίζουμε ακριβώς πότε θα λάβουμε μια απάντηση αφού εμφανιστεί το αρχικό σήμα. Για να ελαχιστοποιηθεί ή να διατηρηθεί εντός συγκεκριμένου εύρους, χρησιμοποιούνται λειτουργικά συστήματα σε πραγματικό χρόνο. Ωστόσο, στα σενάρια όπου απαιτείται γρήγορος χρόνος απόκρισης (κάτω από χιλιοστά του δευτερολέπτου), αυτό υπολείπεται. Για την επίλυση αυτού του προβλήματος, ο αιτούμενος αλγόριθμος πρέπει να εφαρμοστεί στο FPGA χρησιμοποιώντας συνδυαστική ή διαδοχική λογική για να εξασφαλιστεί ο χρόνος απόκρισης που είναι πάντα ο ίδιος και κάτω από χιλιοστά του δευτερολέπτου. Ένα τέτοιο σύστημα πραγματικού χρόνου που εφαρμόζεται στο FPGA μπορεί να τροποποιηθεί και να μεταφερθεί στην κατασκευή μόλις είναι έτοιμο. Ένα ολοκληρωμένο κύκλωμα που δημιουργείται με αυτόν τον τρόπο θα είναι πολύ πιο γρήγορο και πιο ενεργειακά αποδοτικό.

6.4 Πως προγραμματίζουμε ένα FPGA

Η έννοια του "προγραμματισμού FPGA" μπορεί να είναι λίγο παραπλανητική. Σε τελική ανάλυση, δεν υπάρχει πραγματικό πρόγραμμα για διαδοχική εκτέλεση, όπως CPU ή GPU. Ο προγραμματισμός FPGA συνίσταται στη δημιουργία αρχιτεκτονικής υλικού που θα εκτελέσει έναν απαιτούμενο αλγόριθμο και θα τον περιγράψει σε γλώσσα περιγραφής υλικού (HDL). Κατά συνέπεια, τα δομικά στοιχεία αυτού του αλγορίθμου δεν θα είναι ο καταχωρητής μνήμης και ένα σύνολο λειτουργιών που πρέπει να εκτελεστούν, όπως με τα τυπικά προγράμματα που εκτελούνται από CPU ή GPU. Ένα «πρόγραμμα FPGA» θα αποτελείται από στοιχεία χαμηλού επιπέδου που περιλαμβάνουν λογικές πύλες, αθροιστές, καταχωρητές και πολυπλέκτες.

Αυτό μας προσφέρει μεγάλη ευελιξία. Για παράδειγμα, εάν έχουμε έναν τύπο δεδομένων 20-bit, μπορουμε να χρησιμοποιήσουμε οδηγίες ακριβώς 20-bit για να εκτελέσουμε της λειτουργίες. Στης CPU, έχουμε μόνο μητρώα και οδηγίες που έχουν οριστεί από τον κατασκευαστή, τα οποία δεν μπορούν να αλλάξουν. Στα FPGAs, από την άλλη πλευρά, μπορούμε να προσαρμοστούμε στον τύπο δεδομένων επειδή σχεδιάζουμε μόνοι μας την αρχιτεκτονική υλικού.

Μπορούμε επίσης να εφαρμόσουμε λειτουργίες που είναι πολύ πολύπλοκες ή χρονοβόρες για CPU γενικής χρήσης. Οι μπλοκ κρυπτογράφηση και οι κρυπτογραφικές λειτουργίες, για παράδειγμα, εκτελούνται από CPU σε πολλούς κύκλους, απαιτώντας πολύ περισσότερο χρόνο από ένα FPGA.

6.5 Ποιες γλώσσες μπορούμε να χρησιμοποιήσουμε σε ένα FPGA

Όπως αναφέρθηκε παραπάνω, με τα FPGA, η αρχιτεκτονική υλικού έχει σχεδιαστεί για να εκτελεί συγκεκριμένες εργασίες. Σε CPU γενικής χρήσης, η αρχιτεκτονική, η μνήμη και οι οδηγίες καθορίζονται και σφραγίζονται από τον κατασκευαστή.

Για τον προγραμματισμό FPGA, χρησιμοποιούμε συγκεκριμένες γλώσσες όπως VHDL ή Verilog. Η σύνταξη του VHDL μοιάζει περισσότερο με το Pascal παρά με το C, κάνοντας τον προγραμματισμό διαφορετικό από ό, τι με τις τυπικές γλώσσες υψηλού επιπέδου. Το Verilog, ωστόσο, είναι παρόμοιο με το C, το οποίο θα το κάνει πιο εύκολο στη χρήση για άτομα που δεν έχουν προηγούμενη εμπειρία με προγραμματισμό χαμηλού επιπέδου.

Το VHDL είναι μια κάπως αρχαϊκή γλώσσα με ορισμένες παγίδες, μία από τις οποίες είναι ότι ο έλεγχος αν η αρχιτεκτονική λειτουργεί όπως αναμένεται είναι πολύ δύσκολος. Σε ορισμένα έργα, για να διευκολύνουμε τη ζωή μας, η Python χρησιμοποιείται για τη δημιουργία τμημάτων του κώδικα. Φυσικά, όλα θα μπορούσαν να γραφτούν σε VHDL, αλλά είναι πιο εύκολο να το κάνουμε σε Python.

Ο προσομοιωτής HDL είναι το βασικό εργαλείο για τη δημιουργία αρχιτεκτονικής υλικού. Μας επιτρέπει να εκτελέσουμε την προσομοίωση του τρόπου λειτουργίας της αρχιτεκτονικής όταν δίνονται δείγματα δεδομένων εισόδου. Αυτό με τη σειρά του μας δίνει τη δυνατότητα να δούμε πώς ρέουν τα δεδομένα. Ο προσομοιωτής HDL είναι επίσης κρίσιμος επειδή η διαδικασία σύνταξης μιας δεδομένης περιγραφής υλικού σε έναν πίνακα FPGA και προγραμματισμού του ίδιου του πίνακα, μπορεί να είναι πολύ χρονοβόρος, ακόμη και για ένα απλό πρόγραμμα. Ο προσομοιωτής μάς επιτρέπει να επαληθεύσουμε διεξοδικά τον αλγόριθμο που θέλουμε να εφαρμόσουμε σε ένα FPGA.

Εξετάζουμε εάν μια αρχιτεκτονική υλικού σε διαμόρφωση που είναι έτοιμη για εφαρμογή και εκκίνηση σε έναν πίνακα FPGA μπορεί να περάσει από έλεγχο εντοπισμού σφαλμάτων, γιατί αλλιώς θα έχει χειρότερα χαρακτηριστικά χρόνου. Επομένως, πριν εφαρμόσουμε οποιονδήποτε αλγόριθμο σε έναν πίνακα FPGA, εκτελούμε προσομοιώσεις για να ελέγξουμε αν λειτουργεί όπως αναμενόταν ή όχι.

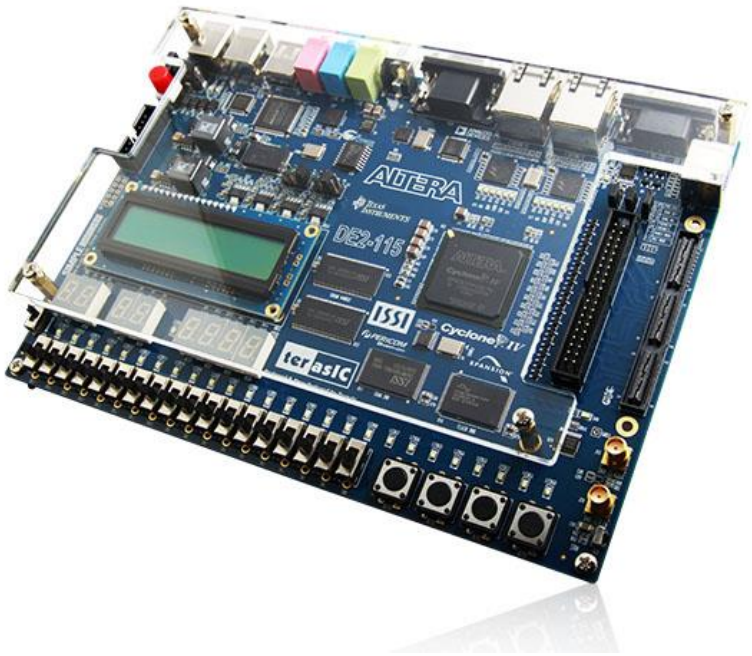
6.6 DE2-115

Η συσκευή Cyclone EP4CE115 εξοπλισμένη με το DE2-115 διαθέτει 114.480 λογικά στοιχεία (LE), τα μεγαλύτερα που προσφέρονται στη σειρά Cyclone IV E, έως 3,9 Mbits μνήμης RAM και 266 πολλαπλασιαστές. Επιπλέον, προσφέρει έναν άνευ προηγουμένου συνδυασμό χαμηλού κόστους και λειτουργικότητας και χαμηλότερης ισχύος σε σύγκριση με τις συσκευές Cyclone προηγούμενης γενιάς.

Το DE2-115 υιοθετεί παρόμοια χαρακτηριστικά από την προηγούμενη σειρά DE2 κυρίως το DE2-70, καθώς και πρόσθετες δυνατότητες για την υποστήριξη γενικών πρωτοκόλλων, συμπεριλαμβανομένου του Gigabit Ethernet (GbE). Παρέχεται μια

υποδοχή υψηλής ταχύτητας (HSMC) για υποστήριξη πρόσθετης λειτουργικότητας και συνδεσιμότητας μέσω καρτών και καλωδίων HSMC. Για την ανάπτυξη πρωτοτύπου ASIC μεγάλης κλίμακας, μπορεί να γίνει σύνδεση με δύο ή περισσότερους πίνακες που βασίζονται σε FPGA μέσω καλωδίου HSMC μέσω του συνδέσμου HSMC.

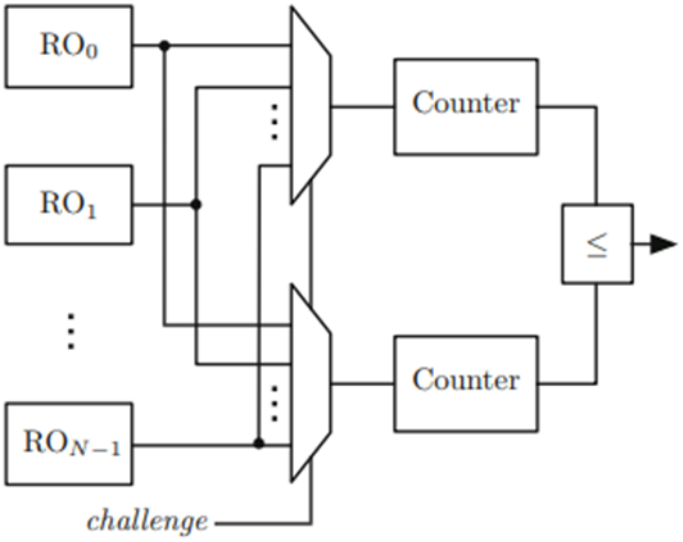
Στην περίπτωση μας δεν θα χρησιμοποιήσουμε όλες της δυνατότητες του FPGA αλλά θα βασιστούμε μόνο στο USB – blaster για την σύνδεση με τον υπολογιστή και τα αποτελέσματα μας θα τα πάρουμε από τα LED που είναι πάνω στο FPGA



Εικόνα 10 FPGA DE2 - 115

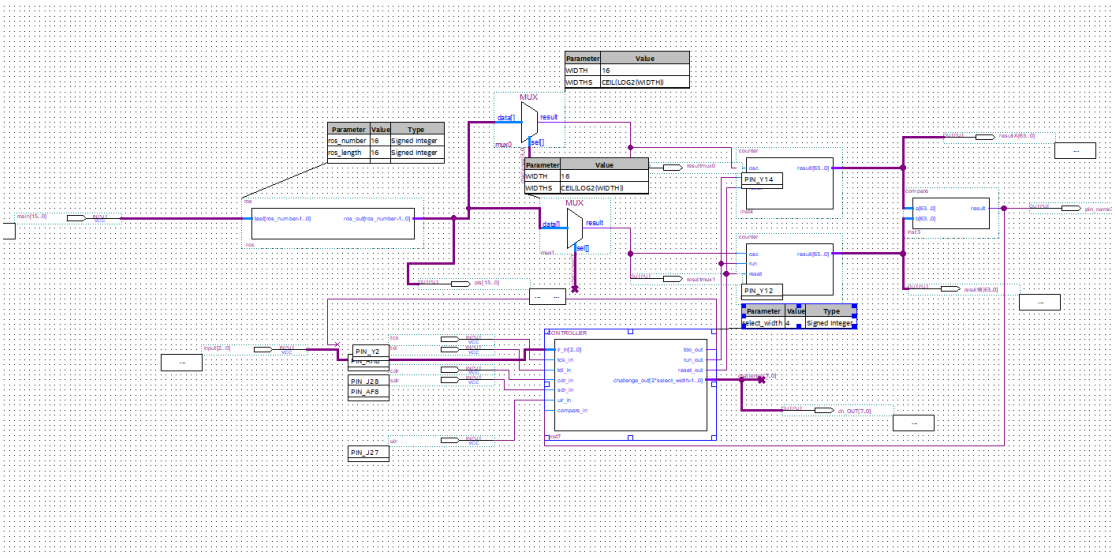
7 Υλοποίηση

Σε αυτό το κεφάλαιο θα δείξουμε πως μπορούμε να φτιάξουμε μια ΦΜΚΣ βασισμένη στην καθυστέρηση και ποιο συγκεκριμένα μία «*Ring Oscillator ΦΜΚΣ*» στο Quartus.



Εικόνα 11 Ring Oscillator

Το project μας θα αποτελείτε από 6 βασικά στοιχεία, την είσοδο, τους πολυπλέκτες, τους counters, το compare, το JTAG και το controller. Μερικά από αυτά τα στοιχεία είναι διαθέσιμα μέσα από το Quartus, τα υπόλοιπα πρέπει να τα προγραμματίσουμε εμείς σε VHDL γλώσσα. Ο ΦΜΚΣ μηχανισμός στο τέλος πρέπει να μοιάζει με την εικόνα 10.

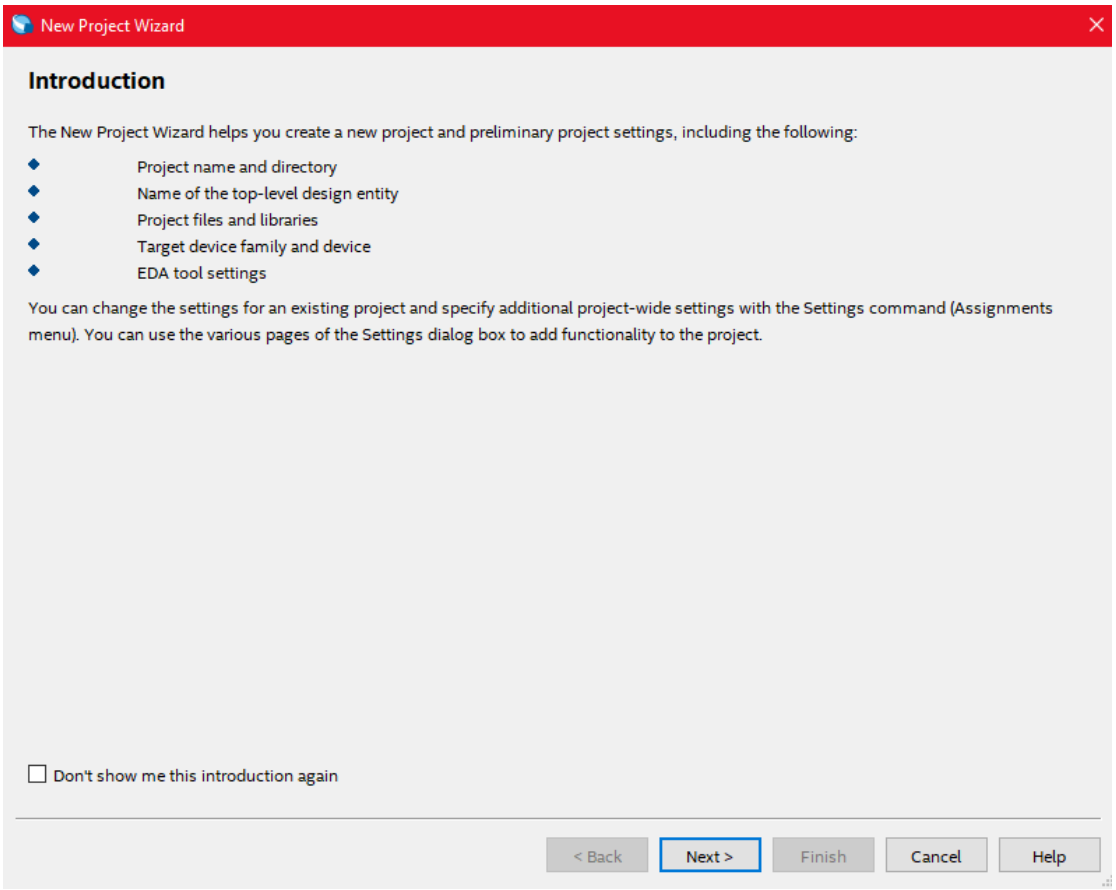


Εικόνα 12 Ring Oscillator ΦΜΚΣ

7.1 Δημιουργία project

Για να ξεκινήσουμε να εργαζόμαστε στην κατασκευή μιας ΦΜΚΣ το πρώτο πράγμα που πρέπει να κάνουμε είναι τα δημιουργήσουμε ένα project, στην δημιουργία του project πρέπει να είμαστε πολύ προσεκτικοί γιατί οποιοδήποτε λάθος δεν θα μπορέσουμε να το εντοπίσουμε με ευκολία. Για να δημιουργήσουμε ένα project επιλέγουμε πάνω αριστερά file και μετά new project wizard.

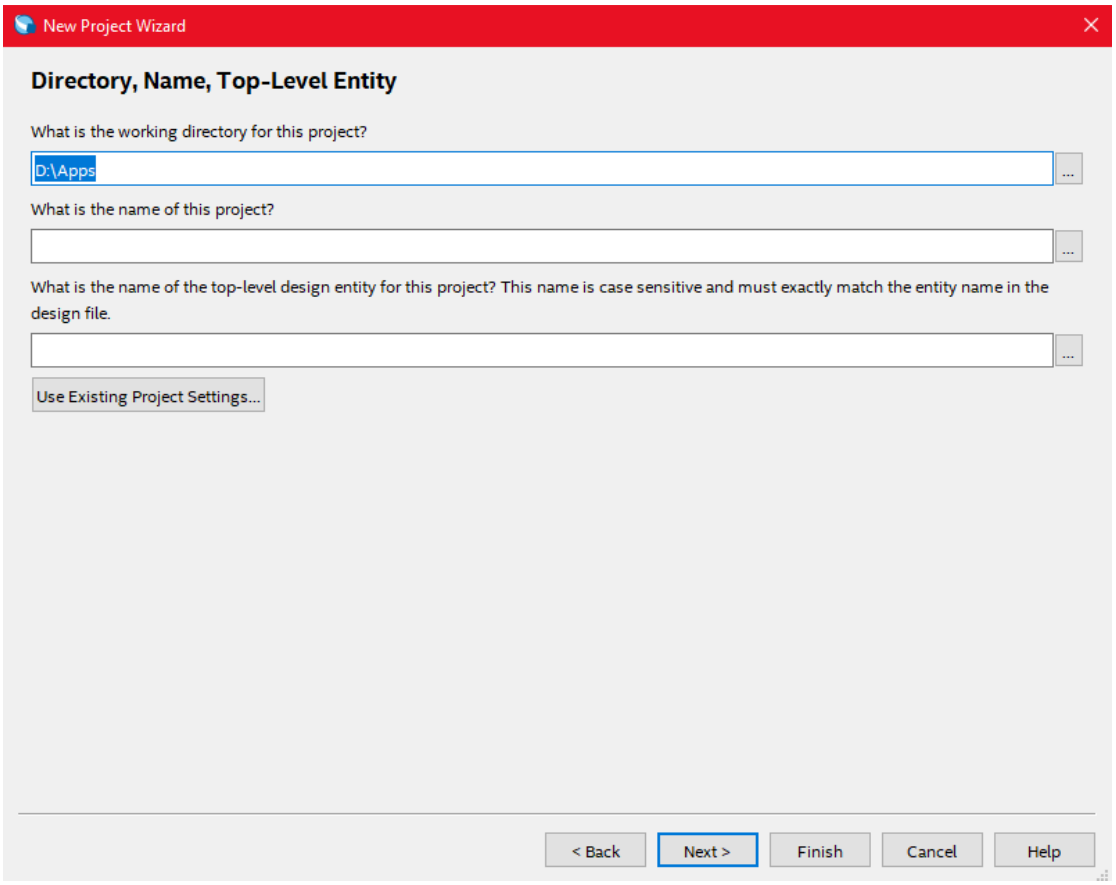
Αφού επιλέξουμε το New Project Wizard θα ανοίξει ένα καινούριο παράθυρο, σε αυτό το παράθυρο θα επιλέξουμε της ρυθμίσεις που θέλουμε για το project μας.



Εικόνα 13 introduction

Στην αρχή μας εμφανίζει το Introduction, στο οποίο μας δείχνει ποιες είναι οι δυνατότητες που προσφέρονται από το Quartus στην δημιουργία ενός καινούριου project, όπως είναι το όνομα του project και η θέση που θα αποθηκευτεί, το όνομα του top-level entity, τα αρχεία και οι βιβλιοθήκες που θέλουμε να προσθέσουμε στο project καθώς και η προσομοίωση του device family και η συσκευή που χρησιμοποιεί το FPGA, τέλος μα δίνει την δυνατότητα να ρυθμίσουμε το Quartus έτσι ώστε να μπορούμε να

χρησιμοποιούμε λειτουργίες του ModelSim μέσα στο Quartus. Για να προχωρήσουμε στο επόμενο βήμα πατάμε next, και θα μας εμφανιστεί η σελίδα επιλογής ονόματος.



Εικόνα 14 name

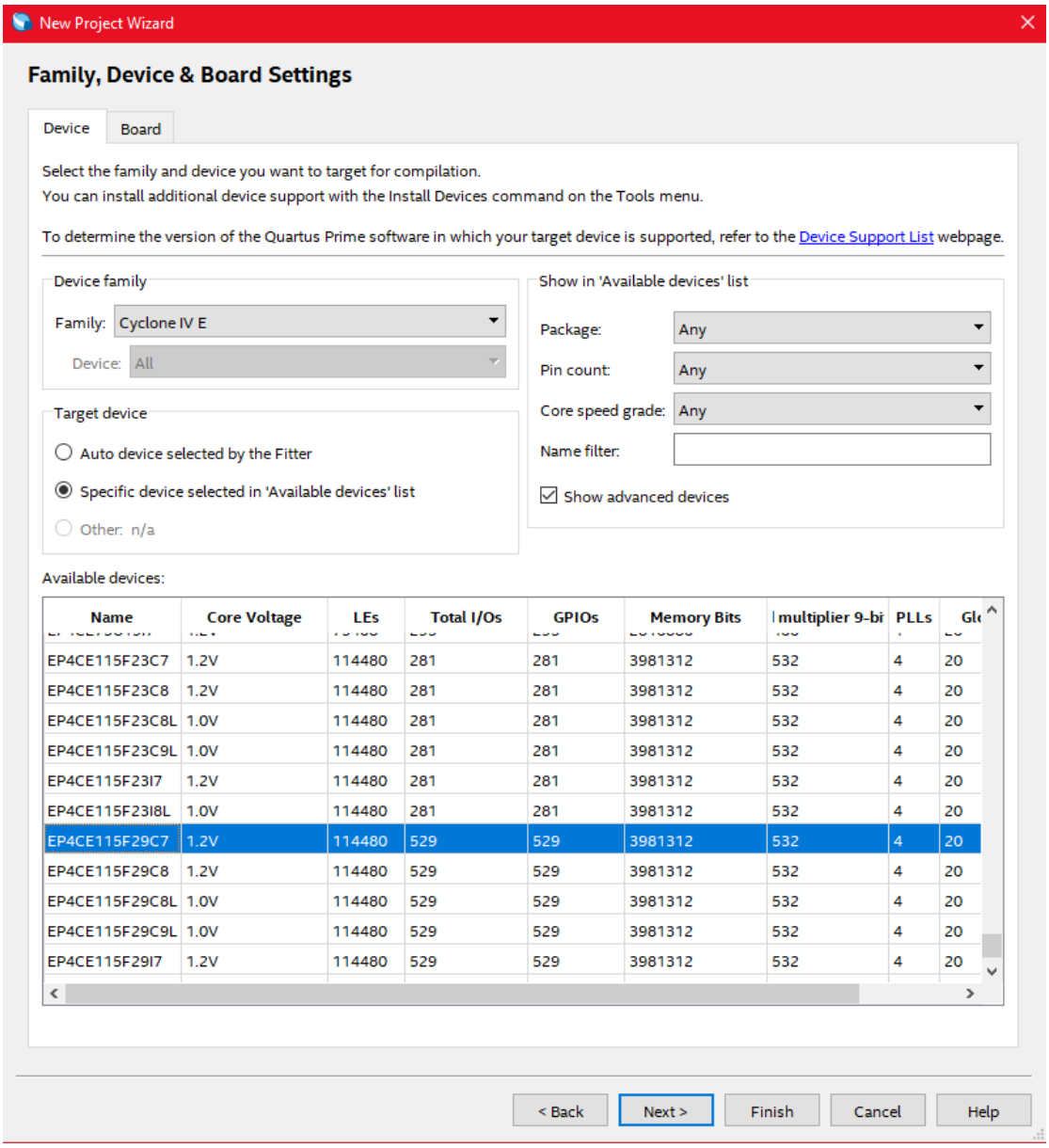
Σε αυτήν την σελίδα επιλέγουμε το σημείο που θέλουμε να αποθηκευτή το project καθώς και το όνομα του project, επίσης καθορίζουμε και το top-level entity το οποίο θα χρησιμοποιήσουμε σε όλη την διάρκεια του project. Στο top-level entity πρέπει να είμαστε προσεκτικοί καθότι πρέπει να είναι ίδιο με το όνομα του entity στον φάκελο.

Αφού ονομάσουμε το project μας θα μας ζητηθεί να επιλέξουμε αναμεσά στην δημιουργία ενός καινούριου project ή την χρήση ρυθμίσεων από κάποιο άλλο project που έχουμε στον υπολογιστή μας ή κατεβάζοντας ένα από το Design Store. Στην παρούσα φάση θα επιλέξουμε την δημιουργία ενός καινούριου project και πατάμε next.

Αφού επιλέξουμε να φτιάξουμε καινούριο project στο επόμενο βήμα θα έχουμε την δυνατότητα να προσθέσουμε κάποια αρχεία και βιβλιοθήκες αν το επιθυμούμε, πατάμε next καθώς δεν χριζόμαστε κάποιον αρχείο ή βιβλιοθήκη .

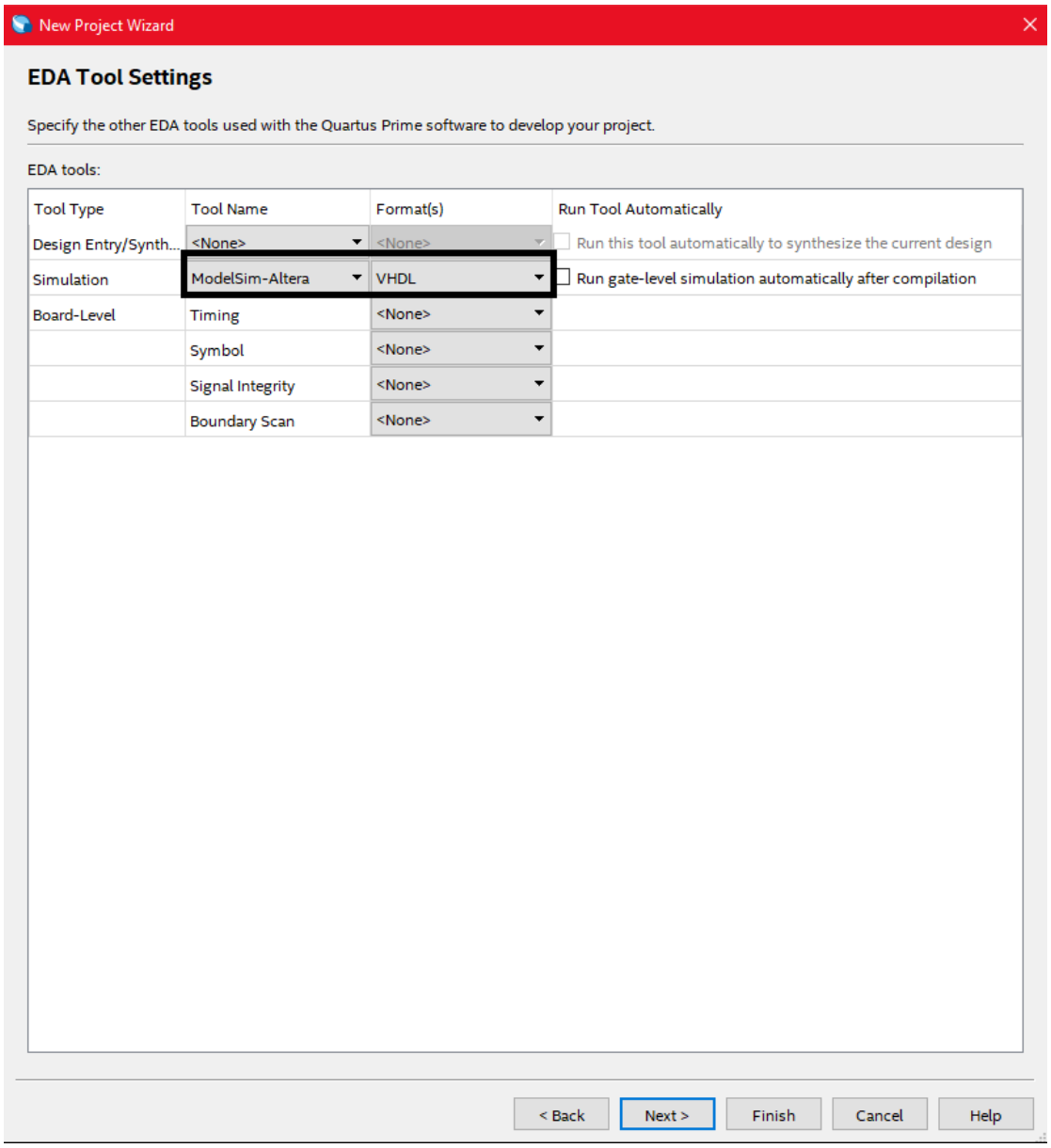
Αφού δεν χρειαστήκαμε κάποιο αρχείο ή βιβλιοθήκη ήρθε η ώρα να καθορίσουμε της ρυθμίσεις που θα έχει το project μας. Για να καθορίσουμε της ρυθμίσεις θα πρέπει να γνωρίζουμε τα χαρακτηριστικά του FPGA μας. Εμείς θα χρησιμοποιήσουμε το FPGA της ALTERA DE@-115 που χρησιμοποιεί την οικογένεια συσκευών cyclone IV E, στο

target device δεν κάνουμε κάποια αλλαγή, ούτε στο show available device list, στο available device θα επιλέξουμε το EP4CE115F29C7 και πατάμε next.



Εικόνα 15 Family, Device @ Board Settings

Αφού καθορίσαμε της ρύθμισης του project σύμφωνα με προδιάγραφες του FPGA σειρά έχει να ρυθμίσουμε και το ModelSim. Στο project μας για μπορέσουμε να χρησιμοποιήσουμε την δυνατότητα που μας δίνει το Quartus για να ελέγξουμε την σωστή λειτουργία του project. Για να το κάνουμε αυτό θα επιλέξουμε στην σειρά του Simulation στο Tool Name Modelsim-Altera και στο format(s) την προγραμματιστική γλωσσά VHDL όπως φαίνεται και στην εικόνα 15.



Εικόνα 16 Εργαλεία Quartus

Αφού επιλέξουμε τα καταλληλά εργαλεία για να μπορέσει το Quartus να χρησιμοποιήσει το ModelSim και να μπορούμε να ελέγχουμε το project μας θα πατήσουμε next. Το επόμενο βήμα μας δείχνει όλες της ρυθμίσεις που έχουμε επιλέξει, αν οι ρυθμίσεις αυτές είναι σωστές τότε πατάμε finish και το project μας θα έχει δημιουργηθεί επιτυχώς. Αν όμως έχουμε κάποιο λάθος μπορούμε να πάμε πίσω πατώντας το back, αυτό θα μας πάει στα προηγούμενα βήματα που μας δίνει την δυνατότητα να αλλάξουμε οποία ρύθμιση επιθυμούμε ανάλογα με το βήμα στο οποίο είμαστε.

7.2 Κατασκευή των κομματιών (vhdl)

Αφού κατασκευάσαμε το project πρέπει να κατασκευάσουμε όλα τα κομμάτια της εργασίας που θα χρειαστούμε. Τα hardware κομμάτια του project θα τα κατασκευάσουμε κυρίως με κώδικα VHDL. Για την επικοινωνία μεταξύ του FPGA και ενός υπολογιστή θα κατασκευάσουμε ένα JTAG παρόμοιο με το προκαθορισμένο στοιχείο virtual JTAG που παρέχεται από το Quartus. Για τους πολυπλέκτες n-bit θα χρησιμοποιήσουμε τις Megafuction του Quartus. Όλα αυτά τα κομμάτια στη συνέχεια συνδυάζονται σε ένα block diagram/schematic file. Ξεκινώντας όμως από κάτω προς τα πάνω, αρχικά ορίζουμε τους ίδιους τους Ring Oscillator.

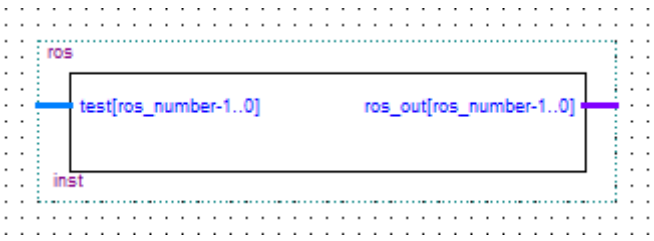
7.2.1 Ring Oscillator

Η γραμμή 5 ορίζει το ποσό των στοιχείων καθυστέρησης (συμπεριλαμβανομένης της πύλης nand της RO) ως γενική μεταβλητή. Η γραμμή 6 ορίζει την είσοδο και την έξοδο όπως οι γραμμές 10 και 11 καθορίζουν τα εσωτερικά σήματα. Στις γραμμές 12 και 13, το στοιχείο LCELL αναφέρεται ότι χρησιμοποιείται στις γραμμές 17 και 19, από το πρώτο στοιχείο καθυστέρησης.

```
1  LIBRARY IEEE ;
2  LIBRARY altera_mf;
3  USE IEEE . STD_LOGIC_1164 . ALL;
4  USE altera_mf.altera_mf_components.all;
5
6  ENTITY ro IS
7   GENERIC ( length : INTEGER := 16);
8  PORT ( enable,test: IN STD_LOGIC ; output : OUT STD_LOGIC );
9  END ro;
10
11 ARCHITECTURE arch OF ro IS
12  SIGNAL path : STD_LOGIC_VECTOR ( length DOWNTO 1);
13  SIGNAL nand_out : STD_LOGIC ;
14 COMPONENT LCELL PORT ( a_in : IN STD_LOGIC ; a_out : OUT STD_LOGIC );
15  END COMPONENT ;
16  BEGIN
17  --nand_out <= enable nand path ( length );
18  nand_out <= enable nand  test;
19  output <= path ( length );
20  lc_0 : LCELL PORT MAP ( a_in => nand_out , a_out => path (1));
21 lc_gen : FOR i IN 1 TO ( length -1) GENERATE
22  lc_i : LCELL PORT MAP ( a_in => path (i), a_out => path (i +1));
23  END GENERATE ;
24  END arch ;
```

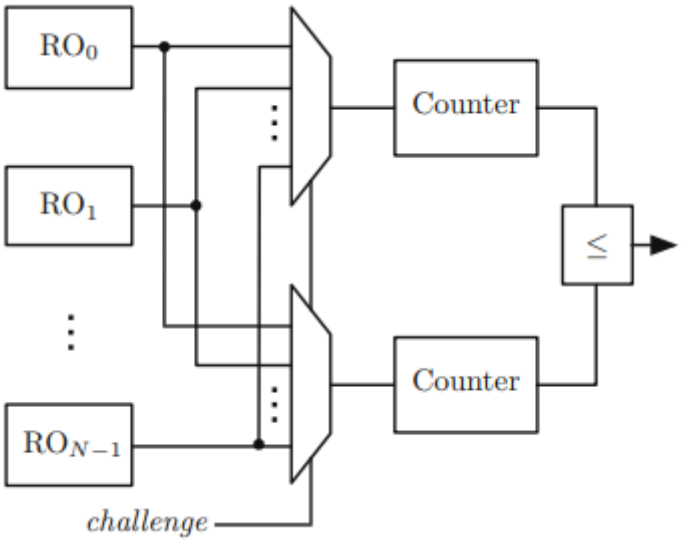
Εικόνα 17 RO

7.2.2 Ring Oscillators



Εικόνα 18 Ros

Η εικόνα 18 δείχνει πώς ορίζεται ένας αυθαίρετος αριθμός Ring Oscillators (που αναφέρεται ως γενική μεταβλητή στη γραμμή 5) επαναχρησιμοποιεί τον ορισμό του Ring Oscillator (Γραμμές 10-13). Το στοιχείο ros έχει μία έξοδο για κάθε RO του (Γραμμές 6 και 18). Σε αυτό το παράδειγμα, ο αριθμός των RO είναι 17. Για λόγους απλότητας, όλα τα ROr είναι μόνιμα ενεργοποιημένα (Γραμμή 17). Σε μια πιο περίτεχνη εφαρμογή, θα μπορούσαν να υπάρχουν μεμονωμένες θύρες εισόδου για να ενεργοποιήσουμε μόνο τα επιθυμητά RO. Η ενεργοποίηση όλων των RO, παρόλο που λαμβάνουν μόνο δύο δείγματα κάθε φορά, είναι σπατάλη ενέργειας, αλλά μπορεί επίσης να θεωρηθεί αντίμετρο έναντι επιθέσεων στα πλευρικά κανάλια που εξάγουν συχνότητες RO μέσω ηλεκτρομαγνητικών εκπομπών. καθώς όταν όλα τα RO ταλαντεύονται, είναι πιο δύσκολο να ξεχωρίσουμε μεμονωμένες συχνότητες RO

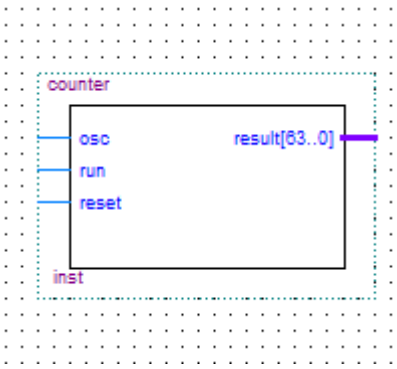


Εικόνα 19 ros

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164 . ALL;

ENTITY ros IS
GENERIC ( ros_number : INTEGER := 16; ros_length : INTEGER := 16);
PORT (test : IN STD_LOGIC_VECTOR ( ros_number-1 downto 0);
 ros_out : OUT STD_LOGIC_VECTOR ( ros_number-1 downto 0));
END ros;

ARCHITECTURE arch OF ros IS
COMPONENT ro
GENERIC ( length : INTEGER );
PORT ( enable,test
: IN STD_LOGIC ; output : OUT STD_LOGIC );
END component ;
BEGIN
ro_gen : FOR i IN 0 TO ros_number -1 GENERATE
 ro_i : ro GENERIC MAP ( length => ros_length ) PORT MAP (
  enable => '1',
  test => test(i),
  output => ros_out (i)
 );
 END GENERATE ;
 END arch ;
```

Εικόνα 20 Ros vhdl

7.2.3 Counter



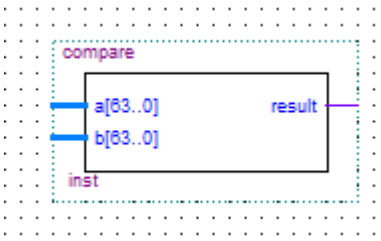
Εικόνα 21 counter

Για να μετρήσουμε τις ταλαντώσεις των Ring Oscillator, μπορούν να χρησιμοποιηθούν απλοί διαδοχικοί μετρητές όπως ορίζονται στην εικόνα 22. Η τιμή του μετρητή αποθηκεύεται σε έναν εσωτερικό καταχωρητή 64-bit (Γραμμή 15), που συνδέεται με το αποτέλεσμα εξόδου του μετρητή (Γραμμή 10 και 17). Το σήμα εισόδου osc συνδέεται με την έξοδο ενός RO που επιλέγεται από ένα MUX. Σε κάθε ανερχόμενη άκρη του osc, η τιμή του μετρητή αυξάνεται κατά ένα (Γραμμή 23), δεδομένου ότι το σήμα εισόδου λειτουργίας έχει οριστεί σε 1 (Γραμμή 22). Ρυθμίζοντας το σήμα εισόδου επαναφοράς στο 1, η τιμή του μετρητή μηδενίζεται (Γραμμή 15).

```
LIBRARY IEEE ;
 USE IEEE . STD_LOGIC_1164 . ALL;
 USE IEEE . STD_LOGIC_UNSIGNED .ALL;

ENTITY counter IS
PORT (
 osc : IN STD_LOGIC ;
run : IN STD_LOGIC ;
 reset : IN STD_LOGIC ;
 result : OUT STD_LOGIC_VECTOR (63 DOWNTO 0)
 );
 END ENTITY counter ;

ARCHITECTURE arch OF counter IS
SIGNAL count : STD_LOGIC_VECTOR (63 DOWNTO 0);
BEGIN
 result <= count ;

PROCESS (osc , run , reset )
 BEGIN
IF reset = '1' THEN count <= ( OTHERS => '0');
ELSIF run = '0' THEN
IF RISING_EDGE (osc) THEN count <= count + 1;
 END IF;
 END IF;
 END PROCESS ;
 END ARCHITECTURE arch ;
```

Εικόνα 22 counter vhdl

7.2.4 Compare



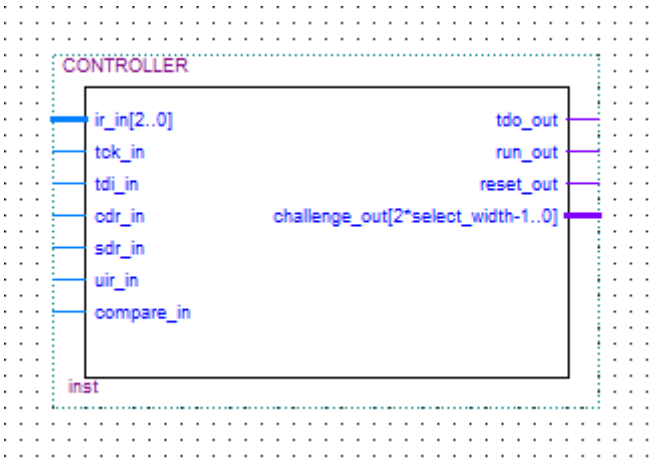
Εικόνα 23 Compare

Για να αποφανθούμε ποιο από τα δυο έκανε της περισσότερες ταλαντώσεις θα χρειαστούμε μια μέθοδο για να συγκρίνουμε της εισόδους. Εφόσον μας έρθει η έξοδος από τους μετρητές αρκεί να συγκρίνουμε ποια έξοδος είναι μεγαλύτερη, αν η μεταβλητή a είναι μεγαλύτερη η έξοδος του compare είναι 1, ενώ αν η μεταβλητή b είναι μεγαλύτερη η έξοδος είναι 0 (γραμμή 14)

```
LIBRARY IEEE ;
USE IEEE . STD_LOGIC_1164 . ALL;

ENTITY compare IS
PORT (
a : IN STD_LOGIC_VECTOR (63 DOWNTO 0);
b : IN STD_LOGIC_VECTOR (63 DOWNTO 0);
result : OUT STD_LOGIC
);
END ENTITY ;

ARCHITECTURE arch OF compare IS
BEGIN
result <= '1' WHEN a > b ELSE '0';
END arch ;
```

Εικόνα 24 Compare vhdl

7.2.5 controller



Εικόνα 25 controller

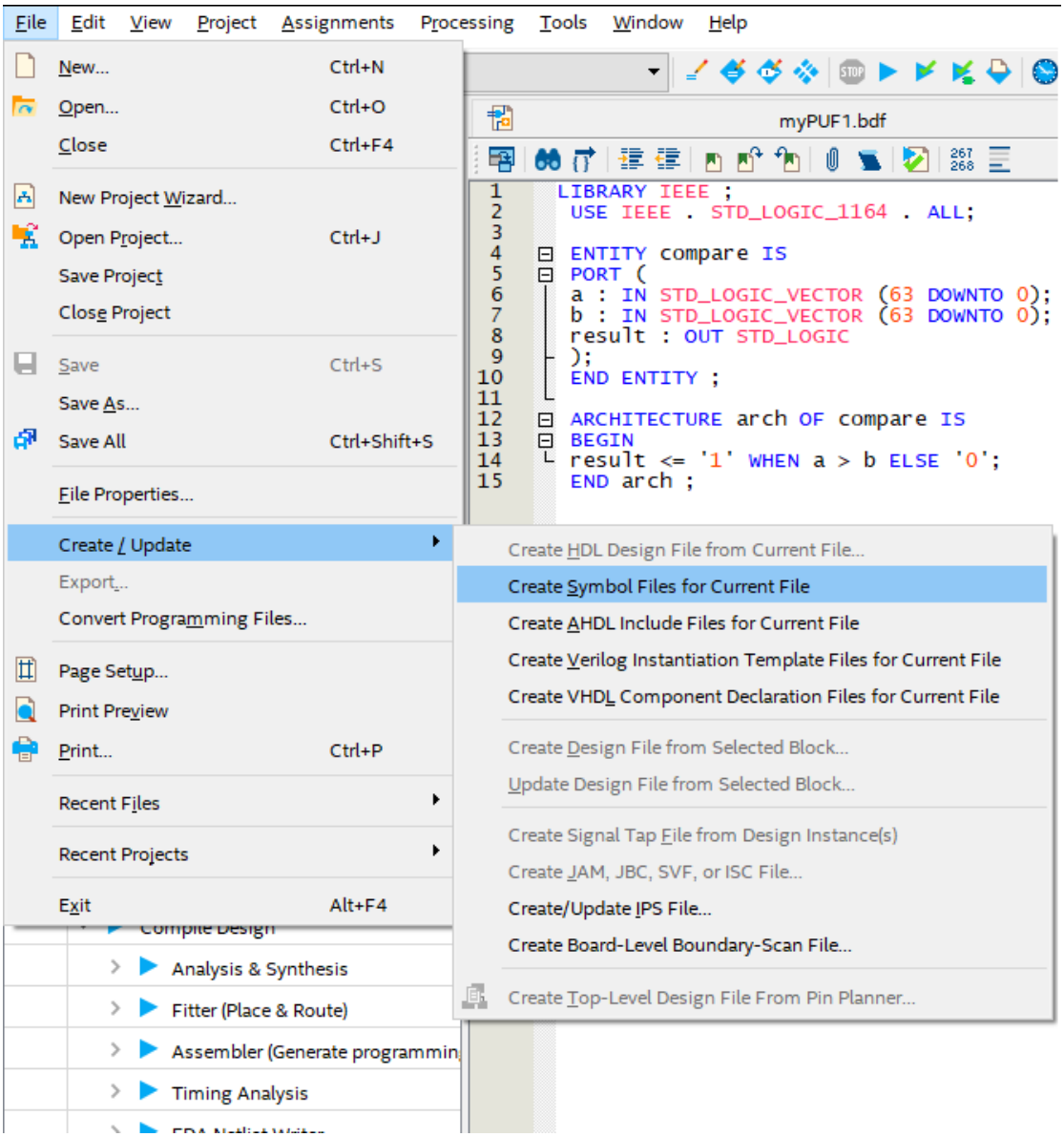
Από το controller μπορούμε να ελέγξουμε την λειτουργία του προγράμματος. Αρχικά στο controller έρχονται όλες οι εντολές ir_in, tck_in, tdi_in, cdr_in, sdr_in, uir_in και compare_in από την γραμμή 7 έως την γραμμή 13, από την οποίες θα καθορίσουν οι λειτουργείες που θα κάνει ο controller, από την γραμμή 23 μέχρι την γραμμή 27 μπορούμε να δούμε της επιλογές που μας δίνει ο controller και της δυνατότητες που έχει. Οι επιλογές αυτές είναι το push challenge και το pop response τα οποία έχουν να κάνουν με την έξοδο που θα μας δώσει ο controller και τα start counter, stop counter και reset counter με τα οποία ρυθμίζουμε τους μετρητές για να τους ξεκινήσουμε, να τους σταματήσουμε και να τους κάνουμε reset αντίστοιχα, συμφώνα με τους μετρητές θα καθοριστούν και οι έξοδοι. Από την γραμμή 39 μέχρι την γραμμή 44 μπορούνε να δούμε ποια θα είναι ακριβώς η λειτουργεία που θα κάνει ο controller δηλαδή, ποιες συνθήκες πρέπει να επικρατούν για να πάρουμε challenge και ποιο θα είναι αυτό και συμφώνα με ποιες συνθήκες τα counter μπορούν να ξεκινήσουν και να σταματήσουν αντίστοιχα. Επίσης στην γραμμή 33 γίνετε αναφορά για την επαναφορά του μετρητή στο 0.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY CONTROLLER IS
   GENERIC (select_width : INTEGER :=4);
   PORT (
      ir_in          :IN STD_LOGIC_VECTOR ( 2 DOWNTO 0);
      tck_in         :IN STD_LOGIC;
      tdi_in         :IN STD_LOGIC;
      cdr_in         :IN STD_LOGIC;
      sdr_in         :IN STD_LOGIC;
      uir_in         :IN STD_LOGIC;
      compare_in     :IN STD_LOGIC;

      tdo_out        :OUT STD_LOGIC;
      run_out        :OUT STD_LOGIC;
      reset_out      :OUT STD_LOGIC;
      challenge_out  :OUT STD_LOGIC_VECTOR(2*select_width-1 DOWNTO 0)
      );

   END controller;
   ARCHITECTURE arch OF controller IS
      CONSTANT PUSH_CHALLANGE: STD_LOGIC_VECTOR(2 DOWNTO 0):="000";
      CONSTANT POP_RESPONSE  : STD_LOGIC_VECTOR(2 DOWNTO 0):="001";
      CONSTANT START_COUNTERS: STD_LOGIC_VECTOR(2 DOWNTO 0):="010";
      CONSTANT STOP_COUNTERS : STD_LOGIC_VECTOR(2 DOWNTO 0):="011";
      CONSTANT RESET_COUNTERS: STD_LOGIC_VECTOR(2 DOWNTO 0):="100";
      SIGNAL challenge : STD_LOGIC_VECTOR(2*select_width-1 DOWNTO 0);
      SIGNAL run       : STD_LOGIC;
   BEGIN
      challenge_out <= challenge;
      run_out       <= run;
      reset_out     <='1' WHEN ir_in = RESET_COUNTERS and uir_in = '1'
         ELSE '0';

      PROCESS(tck_in)
      BEGIN
         IF RISING_EDGE(tck_in) THEN
            IF ir_in = PUSH_CHALLANGE and sdr_in = '1' THEN
               challenge <=tdi_in & challenge(2*select_width-1 DOWNTO 1);
            ELSIF ir_in = POP_RESPONSE and cdr_in = '1' THEN
               tdo_out <= compare_in;
            ELSIF ir_in = START_COUNTERS and uir_in = '1' THEN run <='0';
            ELSIF ir_in = STOP_COUNTERS and uir_in = '1' THEN run <='1';
            END IF;
         END IF;
      END PROCESS;
   END arch;
```

Εικόνα 26 Controller vhdl

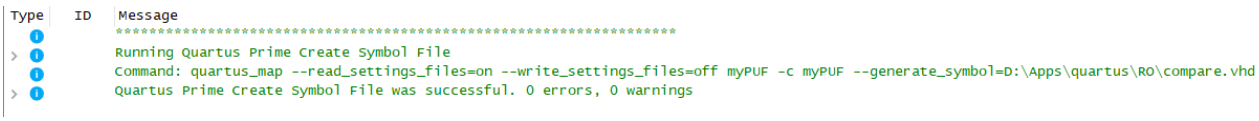
7.3 Τοποθέτηση τον κομματιών στον block diagram / schematic file

Αφού έχουμε κατασκευάσει όλα τα κομμάτια του project μας είμαστε έτοιμοι να τα τοποθετήσουμε στο block diagram / schematic file. Για να το κάνουμε αυτό, θα χρειαστεί να έχουμε φτιάξει τα αρχεία σε κατάλληλη μορφή ώστε να μπορούν τα τοποθετηθούν πάνω στο block diagram / schematic file, δίνουμε την μορφή που θέλουμε στα αρχεία πηγαίνοντας σε κάθε αρχείο που θέλουμε να το τοποθετήσουμε στο block diagram και να επιλέξουμε πάνω αριστερά file > Create / update > Create Symbol Files for current File.



Εικόνα 27 δημιουργία συμβόλου

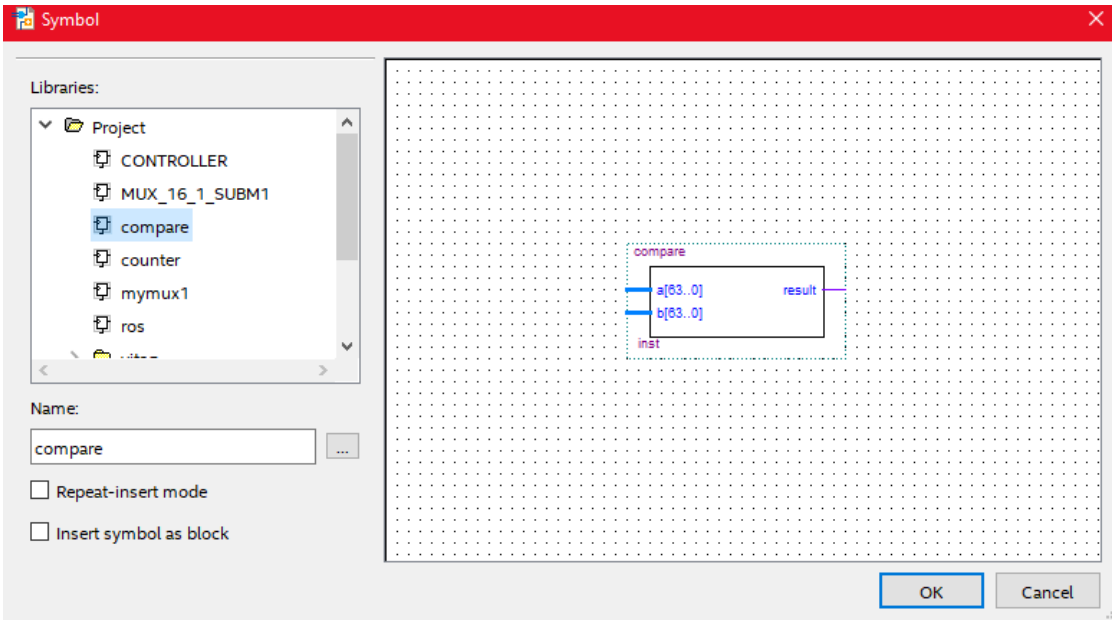
Μετά από αυτό το Quartus θα τρέξει την εντολή Create Symbol file και όταν τελειώσει με το compile,το αρχείο θα είναι έτοιμο για χρήση.



Εικόνα 28 έλεγχος

τοποθετούμε το αρχείο που μόλις φτιάξαμε κάνοντας δεξί κλικ ή διπλό αριστερό κλικ πάνω στο block diagram και θα ανοίξουμε τον φάκελο Project, μέσα στον φάκελο project θα βρούμε όλα τα αρχεία που έχουμε διαμόρφωση, ούτος ώστε να μπορούμε να χρησιμοποιήσουμε στο block diagram. Αφού επιλέξουμε το αρχείο που θέλουμε θα μας δείξει το Quartus σε μια μικρογραφία πως θα είναι το αρχείο μας όταν το τοποθετήσουμε στο block diagram. Οι γραμμές που φαίνονται στα αριστερά και τα δεξιά του σύμβολου καθορίζουν της εισόδους και της εξόδους που έχει το σύμβολο. Για να τοποθετήσουμε

το σύμβολο στο block diagram αρκεί να πατήσουμε το ok και να επιλέξουμε που θέλουμε να τοποθετηθεί πάνω στο block diagram με ένα αριστερό κλικ.

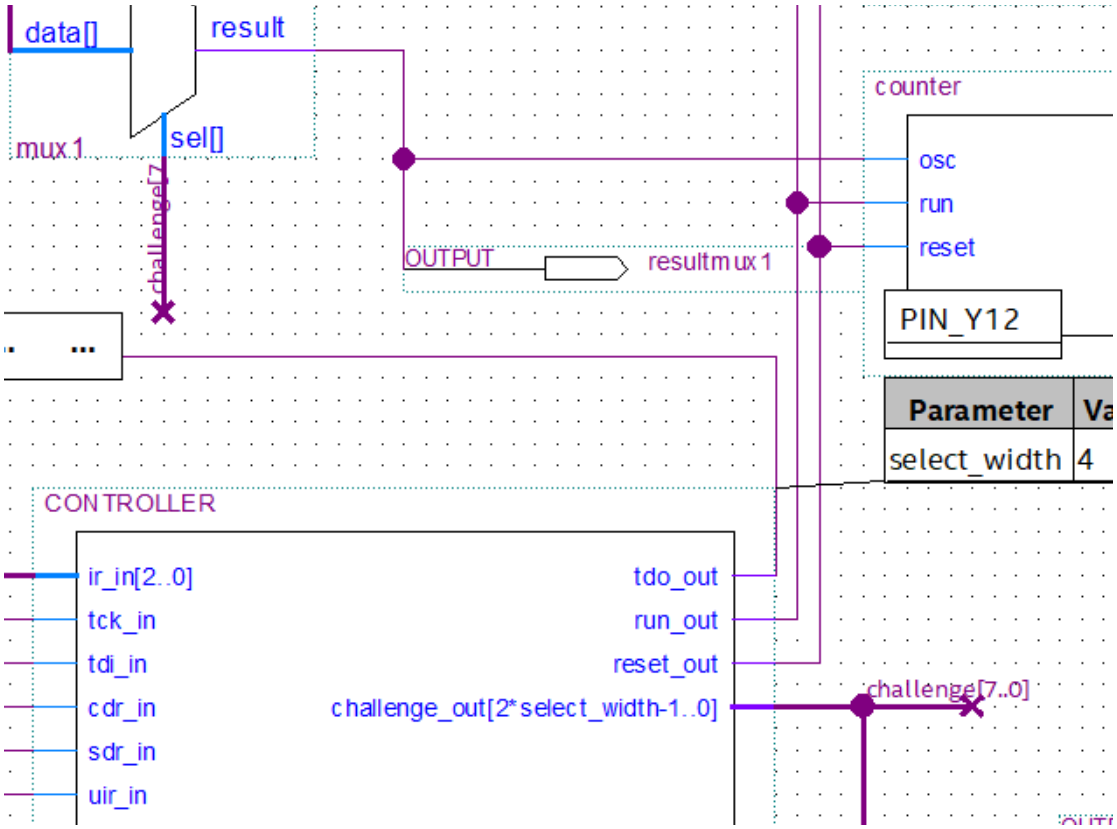


Εικόνα 29 τοποθέτηση συμβόλου

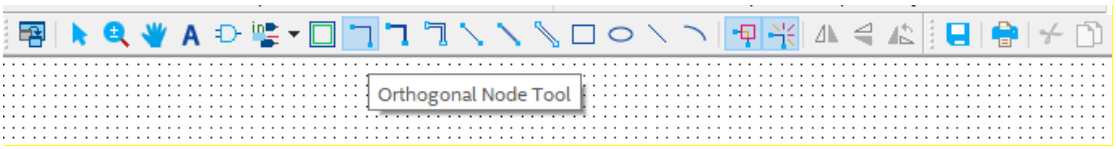
Αφού είδαμε πως μπορούμε να τοποθετήσουμε τα σύμβολα που κατασκευάσαμε στο block diagram, τώρα πρέπει να δούμε και πως θα τα συνδέσουμε όλα αυτά μετάξι τους. Για να συνδέσουμε τα σύμβολα μεταξύ τους θα χρειαστεί να δούμε πως μπορούμε να δημιουργήσουμε κόμβους οι οποίοι θα μεταφέρουν την πληροφορία από το έναν σύμβολο στο άλλο.

Για να χαράξουμε ένα κόμβο από το ένα σύμβολο αρκεί να επιλέξουμε το Orthogonal Node Tool ή το Orthogonal Bus Tool από το Tool bar και να χαράξουμε την διαδρομή που θέλουμε. Πρέπει να προσέξουμε ποιο από τα δυο θα χρησιμοποιήσουμε γιατί έχουν διαφορετική χρήση. Το πρώτο (εικόνα 31) χρησιμοποιείται για να μεταφερθεί μόνο μια πληροφορία ενώ το δεύτερο (εικόνα 32) μπορούμε να μεταφέρουμε πολλές πληροφορείς από την ίδια γραμμή. Πολλές φορές για να μην φορτώνουμε το block diagram χρησιμοποιούμε την δυνατότητα που πας δίνει το Quartus να συνδέσουμε δυο η περιστέρα σημεία του project χωρίς να τα συνδέει μια συνεχομένη γραμμή. Μπορούμε απλά να δώσουμε το ίδιο όνομα σε δυο διαφορετικές γραμμές, μια γραμμή στην έξοδο

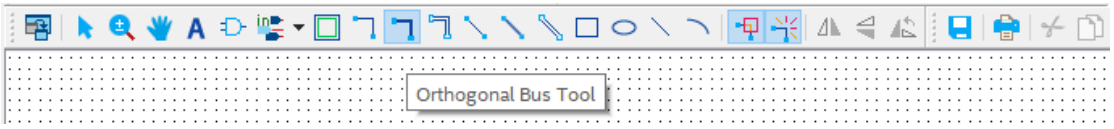
και μια στην είσοδο, στο project μπορούμε να το δούμε αυτό στην έξοδο του controller και στην είσοδο του mux εικόνα 30.



Εικόνα 30 challenge

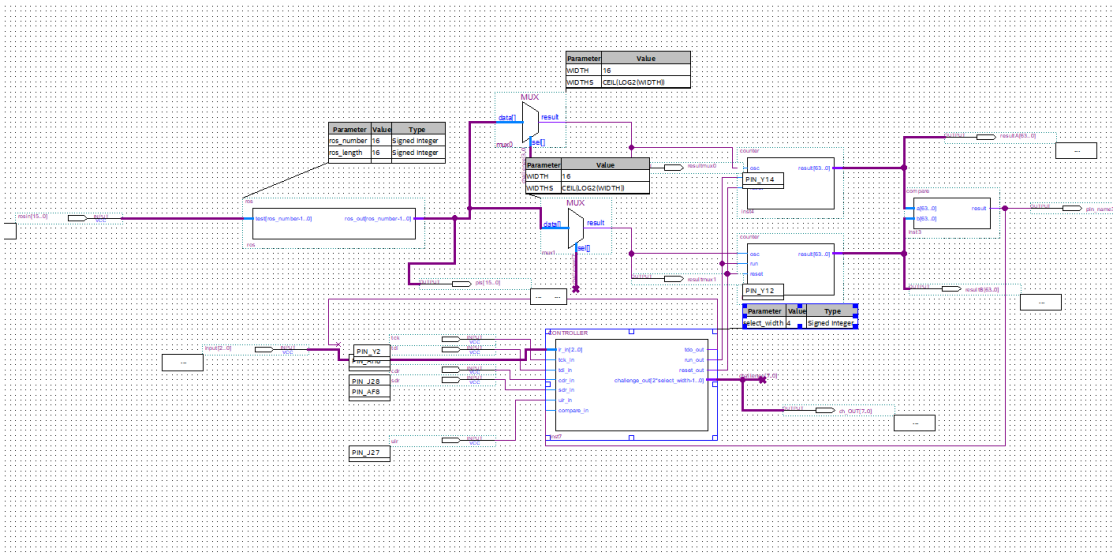


Εικόνα 31 Orthogonal Node Tool



Εικόνα 32 Orthogonal Bus Toll

Αρχικά παίρνουμε το σήμα από το ros και το περνάμε στους πολυπλέκτες, οι πολύπλέκτες μαζί με το σήμα από το ros δέχονται τιμή και από το controller και περνούν τα ανάλογο αποτέλεσμα στους counters, οι οποίοι μετράνε της τιμές του ros που περνάνε από το mux εφόσον το run είναι 1. Στην συνέχεια το compare συγκρίνει της τιμές από τους counters και τους περνάει στο controller, στο οποίο μπορούμε να ξεκινήσουμε τους counter, να τους σταματήσουμε, να κάνομε reset, να πάρουμε απάντηση και να δώσουμε, σε συνδυασμό με το vjtag.



Εικόνα 33 PUF

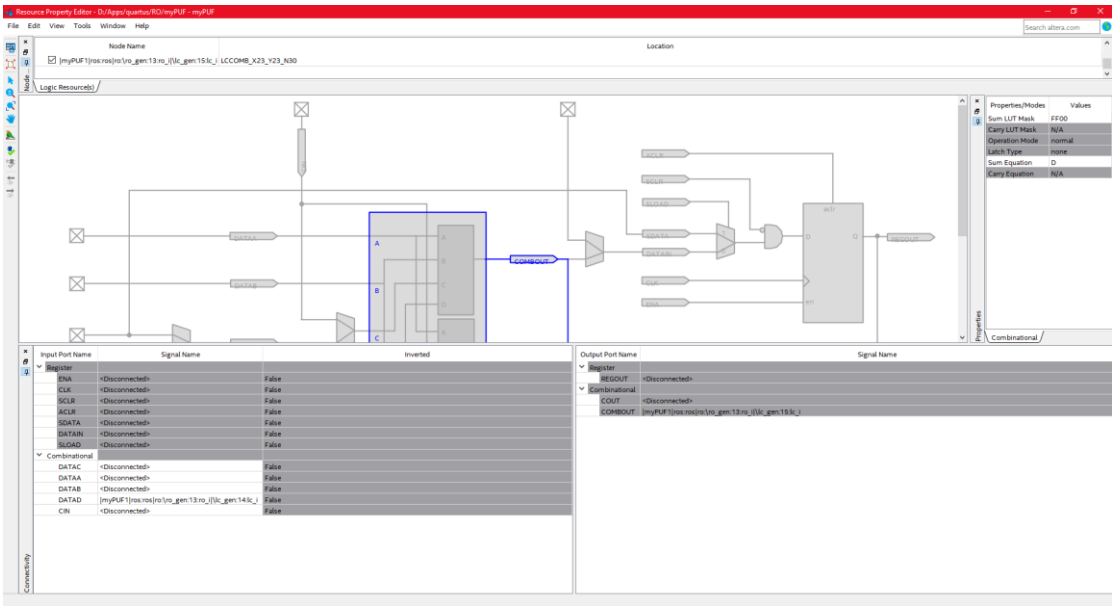
Τα output που βλέπουμε στην παραπάνω εικόνα είναι αυτά που μας βοηθάνε στο να γνωρίζουμε σε κάθε σημείο ποια είναι η τιμή στους κόμβους, και αν είναι συμβατά με τα αναμενόμενα αποτελέσματα. Είναι ενός τρόπος για να ελέγχουμε την ορθότητα του project σε όλα τα σημεία του και όχι μόνο με την τελική τιμή του.

7.4 LUT placement

Αφού ολοκληρώσουμε την τοποθέτηση τον συμβολών στο block diagram για να μπορέσουμε να πάρουμε τα αναμενόμενα αποτελέσματα από την ΦΜΚΣ θα δώσουμε συγκεκριμένες τοποθεσίες στα LUT στο chip planner. Αυτό μπορούμε να το κάνουμε αλλάζοντας απευθείας το Quartus settings file. Το Quartus setings file με την επέκταση .qsf μπορούμε να το βρούμε στο root directory του project. Στο αρχείο .qsf μπορούμε να καθορίσουμε την τοποθεσία από όλα τα LCELL, για να το καταφέρουμε αυτό χρειάζεται να γνωρίζουμε όλο το όνομα του «κόμβου» με το οποίο προσδιορίζεται το αντίστοιχο LCELL στο Quartus. Το όνομα του «κόμβου» μπορούμε να το βρούμε στο Node Propeties (εικόνα 35) στο chip planner όταν κλικάρουμε στο LE του LCELL.

```
set_location_assignment LCCOMB_X23_Y10_N0 -to "ros:ros|ro:\\ro_gen:0:ro_i|lc_0"
set_location_assignment LCCOMB_X23_Y10_N2 -to "ros:ros|ro:\\ro_gen:0:ro_i|\\lc_gen:1:lc_i"
set_location_assignment LCCOMB_X23_Y10_N4 -to "ros:ros|ro:\\ro_gen:0:ro_i|\\lc_gen:2:lc_i"
set_location_assignment LCCOMB_X23_Y10_N6 -to "ros:ros|ro:\\ro_gen:0:ro_i|\\lc_gen:3:lc_i"
set_location_assignment LCCOMB_X23_Y10_N8 -to "ros:ros|ro:\\ro_gen:0:ro_i|\\lc_gen:4:lc_i"
set_location_assignment LCCOMB_X23_Y10_N10 -to "ros:ros|ro:\\ro_gen:0:ro_i|\\lc_gen:5:lc_i"
set_location_assignment LCCOMB_X23_Y10_N12 -to "ros:ros|ro:\\ro_gen:0:ro_i|\\lc_gen:6:lc_i"
set_location_assignment LCCOMB_X23_Y10_N14 -to "ros:ros|ro:\\ro_gen:0:ro_i|\\lc_gen:7:lc_i"
set_location_assignment LCCOMB_X23_Y10_N16 -to "ros:ros|ro:\\ro_gen:0:ro_i|\\lc_gen:8:lc_i"
set_location_assignment LCCOMB_X23_Y10_N18 -to "ros:ros|ro:\\ro_gen:0:ro_i|\\lc_gen:9:lc_i"
set_location_assignment LCCOMB_X23_Y10_N20 -to "ros:ros|ro:\\ro_gen:0:ro_i|\\lc_gen:10:lc_i"
set_location_assignment LCCOMB_X23_Y10_N22 -to "ros:ros|ro:\\ro_gen:0:ro_i|\\lc_gen:11:lc_i"
set_location_assignment LCCOMB_X23_Y10_N24 -to "ros:ros|ro:\\ro_gen:0:ro_i|\\lc_gen:12:lc_i"
set_location_assignment LCCOMB_X23_Y10_N26 -to "ros:ros|ro:\\ro_gen:0:ro_i|\\lc_gen:13:lc_i"
set_location_assignment LCCOMB_X23_Y10_N28 -to "ros:ros|ro:\\ro_gen:0:ro_i|\\lc_gen:14:lc_i"
set_location_assignment LCCOMB_X23_Y10_N30 -to "ros:ros|ro:\\ro_gen:0:ro_i|\\lc_gen:15:lc_i"
```

Εικόνα 34 lut placement

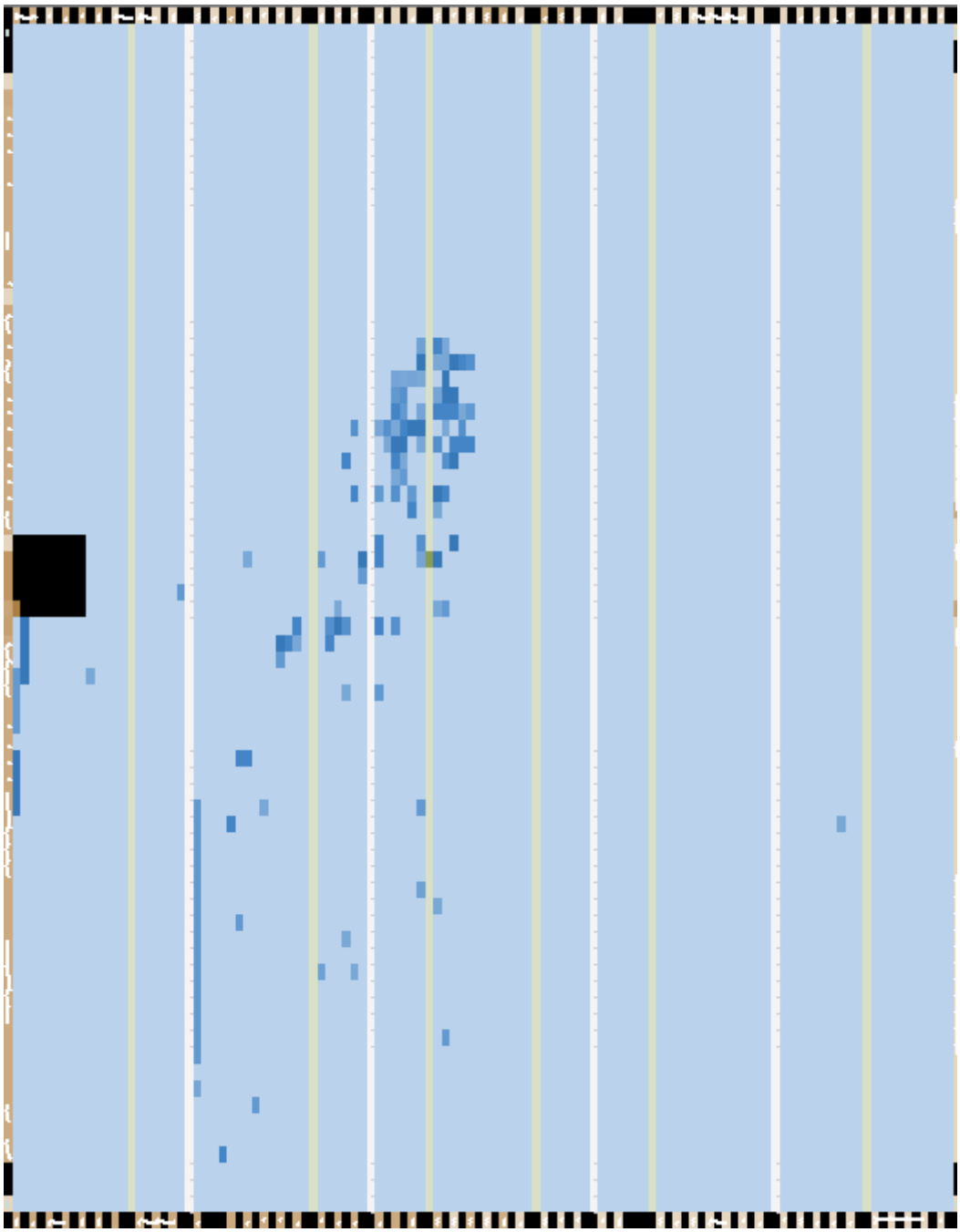


Εικόνα 35 node properties

Εδώ παρατηρούμε ότι το mypuf αντιστοιχεί στο όνομα του project καθώς τα ros και ro αντιστοιχούν στις οντότητες VHDL. Τα αναγνωριστικά ro_gen και ro_i είναι αυτά που χρησιμοποιούνται στην εντολή GENERATE του ros, καθώς τα lc_gen και lc_i είναι αυτά από το ro. Στο myPUF.qsf πρέπει να φτιάξουμε 16 RO από 0 μέχρι 15 και 16 LC πάλι από 0 μέχρι το 15.

Αφού τοποθετήσουμε τα LCELL το chip planner θα πρέπει να φαίνονται όπως και στην εικόνα 36 με όλα τα LCELL που φτιάξαμε στην σειριά. Αυτό που μένει να κάνουμε

τώρα είναι να ορίσουμε ποιο από τα τέσσερα LUT input του LCELL θέλουμε να χρησιμοποιήσουμε.



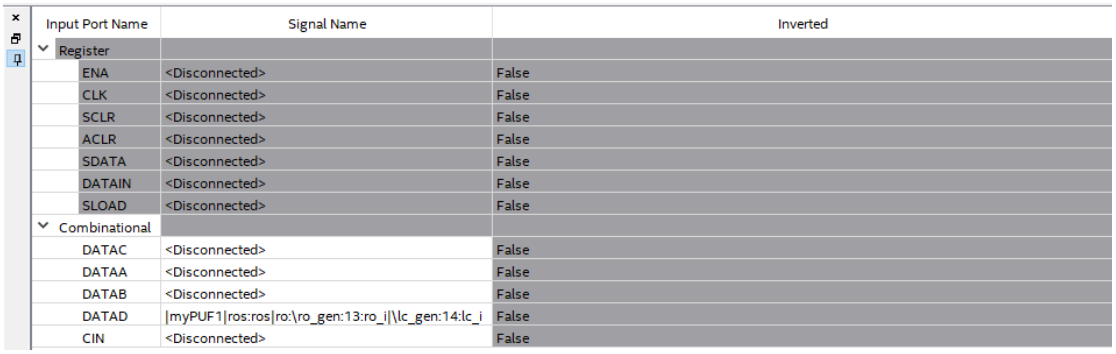
Εικόνα 36 chip planner

Εξετάζουμε το απλό παράδειγμα ενός LUT που μεταδίδει μόνο ένα σήμα εισόδου (input) στην έξοδο (output) του. Δηλαδή μια τυπική εφαρμογή σε delay based ΦΜΚΣ σε FPGA. Για να χρησιμοποιήσουμε το LUTinputA, η LUTmask έχει οριστεί σε 1010101010101010 = AAAA για είσοδο LUTB σε 1100110011001100 = CCCC. Για είσοδο LUTCto 1111000011110000 = F0F0 και για είσοδο LUT D σε 1111111100000000 = FF00. Όλες αυτές οι διαμορφώσεις εκτελούν λογικά την ίδια εργασία. Ωστόσο, η διαδρομή από την είσοδο A στην έξοδο είναι η μεγαλύτερη από όλες τις εισόδους, ενώ η διαδρομή από την είσοδο D στην έξοδο είναι η μικρότερη. Στο

πραγματικό FPGA μπορεί να μην είναι απαραίτητο ότι η καθυστέρηση της εισόδου D στην έξοδο είναι πράγματι η ταχύτερη. Αυτό εξαρτάται από το πώς είναι πραγματικά δομημένο το υλικό. Αλλά υπάρχουν σίγουρα σημαντικές διαφορές καθυστέρησης μεταξύ των διαφορετικών εισόδων LUT.

Για την εφαρμογή μιας ΦΜΚΣ με καθυστέρηση, τέτοιες διαφορές μπορεί να έχουν σοβαρό αντίκτυπο στην ποιότητα της ΦΜΚΣ. Εάν, για παράδειγμα, τα LCELLs ενός RO x χρησιμοποιούν κυρίως την είσοδό τους LUT A και τα LCELL ενός άλλου RO y χρησιμοποιούν κυρίως την είσοδό τους D, το RO x πιθανότατα θα είναι πιο αργό από το RO y σε όλα τα FPGA. οδηγώντας σε κακή μοναδικότητα. Τέτοια σενάρια συμβαίνουν στην πραγματικότητα, εάν η δρομολόγηση LUT αφεθεί στον compiler. Αυτή η ενότητα παρουσιάζει επομένως τον τρόπο επιβολής της routing στα LUT.

Ο εύκολος τρόπος για να επιβάλουμε routing στα LUT είναι κατασκευάζοντας ένα script (changes.tcl) το οποίο όταν το καλέσουμε τε την εντολή quartus_cdb –t changes.tcl θα μας αλλάζει όλα τα sum Equation και τα Lut mask στης κατάλληλες θέσεις, αν αυτό δεν μπορέσουμε να το κάνουμε, υπάρχει και η επιλογή να το κάνουμε όλο χειροκίνητα. Για να αλλάξουμε χειροκίνητα τις χρησιμοποιούμενες εισόδους LUT, πρέπει να ανοίξουμε το Quartus Resource Property Editor κάνοντας διπλό κλικ σε ένα LE στο Chip Planner. Εκεί, το "Signal Name" (εικόνα 37)κάθε εισόδου LUT μπορεί να επεξεργαστεί ή να αφαιρεθεί. Για αλλαγή, π.χ. το LUT χρησιμοποιώντας την είσοδο C, για τη χρήση της εισόδου D, αντιγράφουμε το όνομα του σήματος για την είσοδο C και το επικολλούμε στην την είσοδο D. Στη συνέχεια αφαιρούμε τη σύνδεση για την είσοδο C. Τέλος, αλλάζουμε τη μάσκα LUT από F0F0 σε FF00. Στη συνέχεια, οι αντίστοιχες αλλαγές παρατίθενται στο "Change Manager" του Chip Planner και μπορούν να εφαρμοστούν κάνοντας κλικ στο κουμπί "Check and Save All NetlistChanges".



Input Port Name	Signal Name	Inverted
Register		
ENA	<Disconnected>	False
CLK	<Disconnected>	False
SCLR	<Disconnected>	False
ACLR	<Disconnected>	False
SDATA	<Disconnected>	False
DATAIN	<Disconnected>	False
SLOAD	<Disconnected>	False
Combinational		
DATAC	<Disconnected>	False
DATAA	<Disconnected>	False
DATAB	<Disconnected>	False
DATAD	|myPUF1|ros:ros|ro:\ro_gen:13:ro_i|\lc_gen:14:lc_i	False
CIN	<Disconnected>	False

Εικόνα 37 signal name

Properties/Modes	Values
Sum LUT Mask	FF00
Carry LUT Mask	N/A
Operation Mode	normal
Latch Type	none
Sum Equation	D
Carry Equation	N/A

Εικόνα 38 values

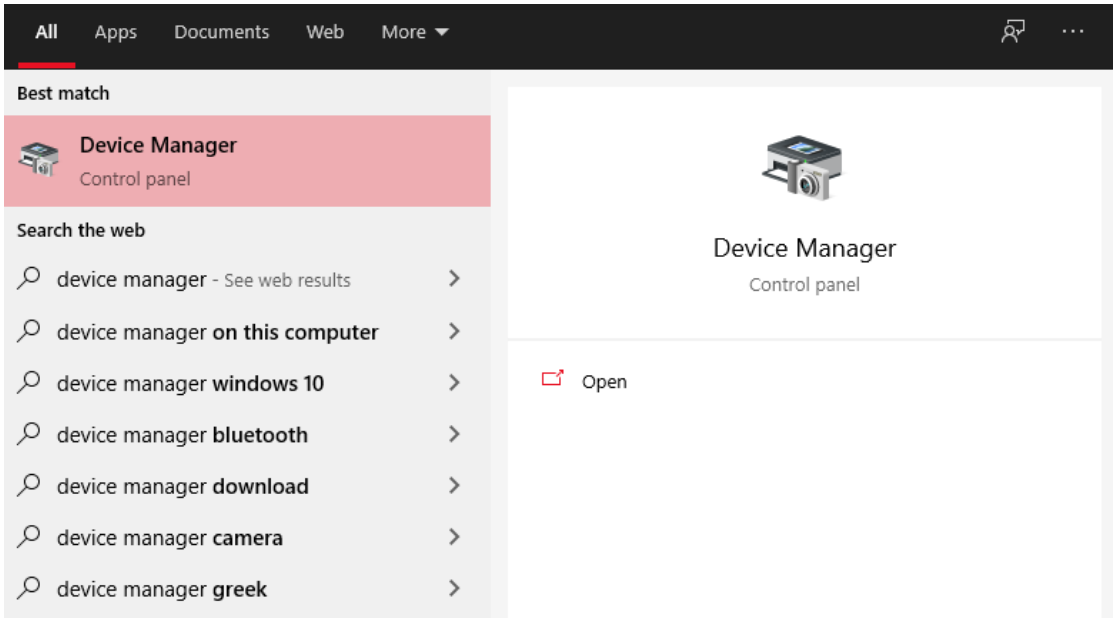
7.5 Πως συνδέουμε το FPGA

Για να συνδέσουμε ένα FPGA πρώτα πρέπει να κατεβάσουμε και να διαβάσουμε το starter guide ή το user guide για το FPGA μας για να είμαστε σίγουροι ότι έχουμε κατεβασιές τα απαραίτητα αρχεία και να είμαστε σίγουρη ότι το FPGA λειτουργεί σωστά .

Επόμενο βήμα είναι η εγκατάσταση του προγράμματος οδήγησης USB-Blaster. Συνδέουμε το adapter 12 volt για να τροφοδοτήσουμε την πλακέτα. Χρησιμοποιούμε το καλώδιο USB για να συνδέσουμε τον πιο αριστερό σύνδεσμο USB (αυτόν που είναι πιο κοντά στον διακόπτη τροφοδοσίας) στην πλακέτα DE2-115 σε μια θύρα USB σε έναν υπολογιστή που εκτελεί το λογισμικό Quartus II. Αφού ολοκληρώσουμε αυτά τα βήματα μπορούμε να ενεργοποιήσουμε τον διακόπτη τροφοδοσίας στην πλακέτα DE2-115.

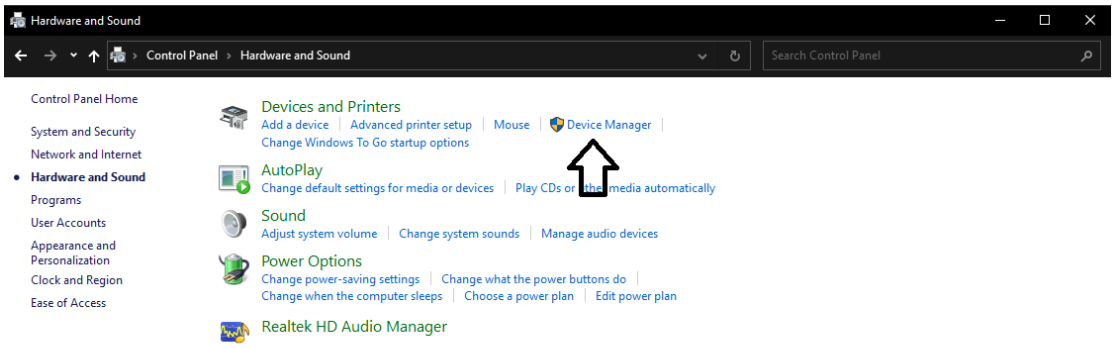
Ο υπολογιστής θα αναγνωρίσει το νέο υλικό που είναι συνδεδεμένο στη θύρα USB του, αλλά δεν θα μπορεί να προχωρήσει εάν δεν έχει ήδη εγκαταστήσει το απαιτούμενο πρόγραμμα οδήγησης. Το DE2-115 είναι προγραμματισμένο χρησιμοποιώντας τον μηχανισμό Altera USB-Blaster. Εάν το πρόγραμμα οδήγησης USB-Blaster δεν είναι ήδη εγκατεστημένο, θα εμφανιστεί ο οδηγός νέου υλικού, ο οποίος δεν μπορεί να μας βοηθήσει γιατί τον driver δεν μπορούμε να το βρούμε στο διαδίκτυο. Όμως ο driver είναι διαθέσιμος μέσα στον φάκελο του Quartus II, οπότε για να πούμε στον υπολογιστή από πού να πάρει τον driver θα πάμε στο device manager και να επιλέξουμε το κατάλληλο αρχείο.

Για να ανοίξουμε το device manager κλικάρουμε κάτω αριστερά στο εικονίδιο τον windows. Μόλις ανοίξει μπορούμε να πληκτρολογήσουμε την λέξη device manager και να κλικάρουμε στην πρώτη επιλογή.



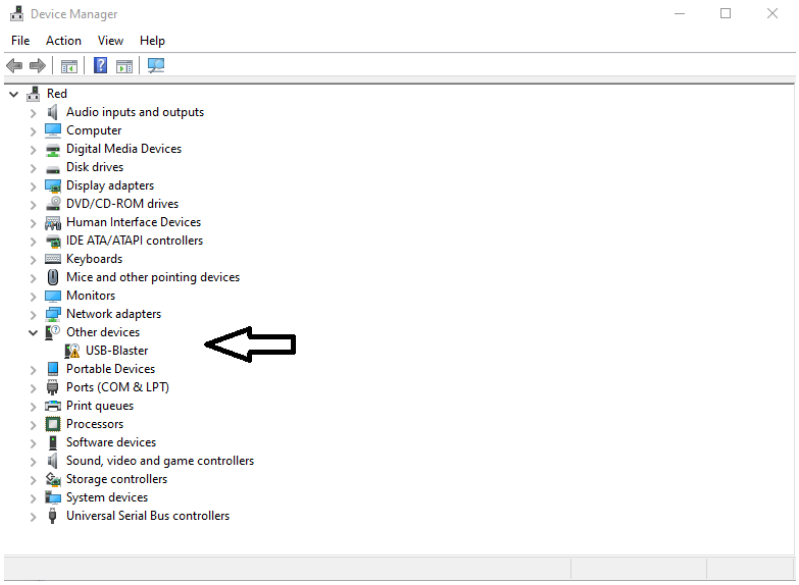
Εικόνα 39 device manager search

Επίσης μπορούμε να βρούμε το Device manager από το control panel επιλέγοντας Devices and printer και μετρά Device manager.



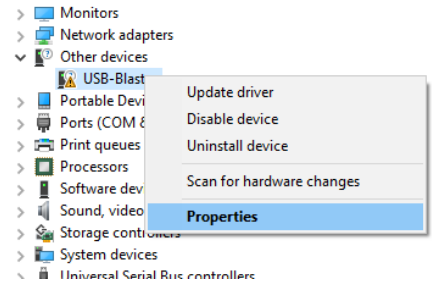
Εικόνα 40 Device manager control panel

Αφού ανοίξουμε το device manager αρκεί να βρούμε το πεδίο στο οποίο θα έχει ενταχθεί το USB – Blaster, συνηθής το USB – Blaster πριν γίνει η σωστή εγκατάσταση του θα είναι στην κατηγορία Other Devices.



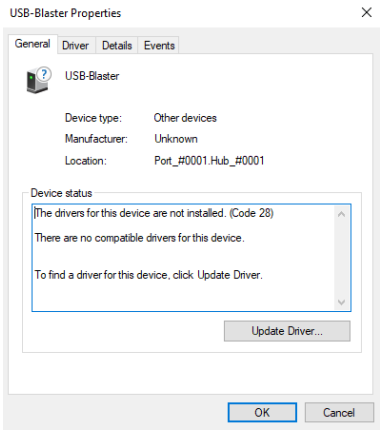
Εικόνα 41 USB – Blaster

Αφού εντοπίσουμε την τοποθεσία του πατάμε δεξί κλικ και properties πάνω στο USB – Blaster.



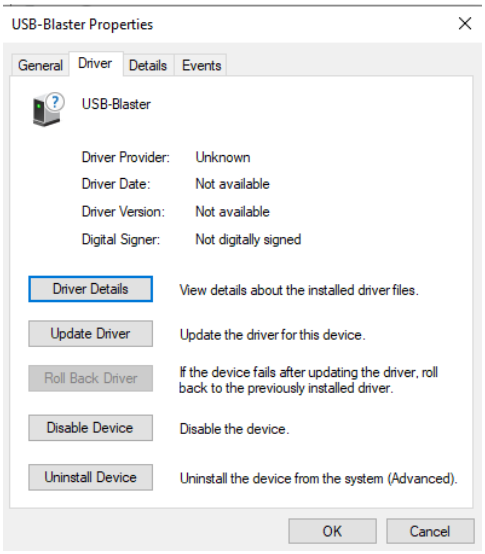
Εικόνα 42 USB - Blaster Properties

Θα μας ανοίξει ένα καινούριο παράθυρο, μέσα σε αυτό το παράθυρο μπορούμε να δύομε όλες της ιδιότητες του USB – Blaster.



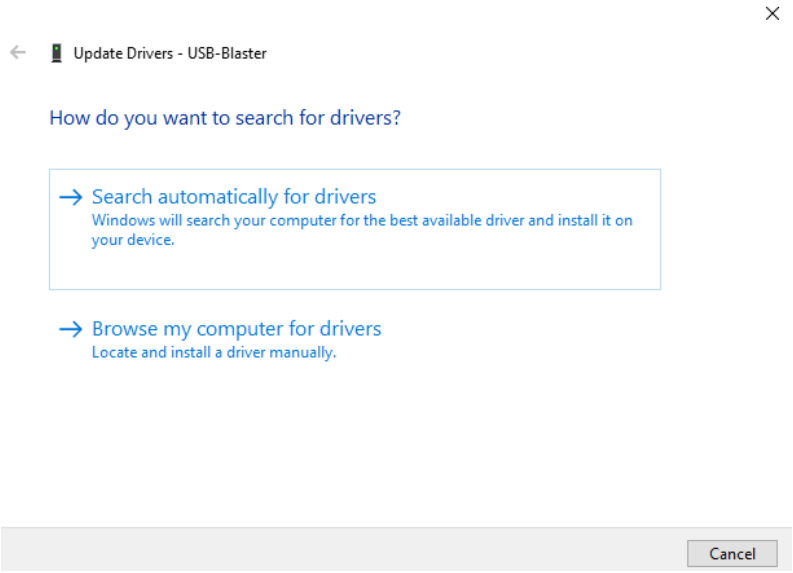
Εικόνα 43 General Properties

Εμάς αυτήν την στιγμή μας ενδιαφέρουν οι drivers της συσκευής για αυτό θα πάμε στην δεύτερη καρτέλα με το όνομα Driver, εδώ μπορούμε να δούμε αν έχει κάποιον Driver η συσκευή και ποια Version έχει, επίσης μπορούμε να δούμε την χρονολογία εγκατάστασης.



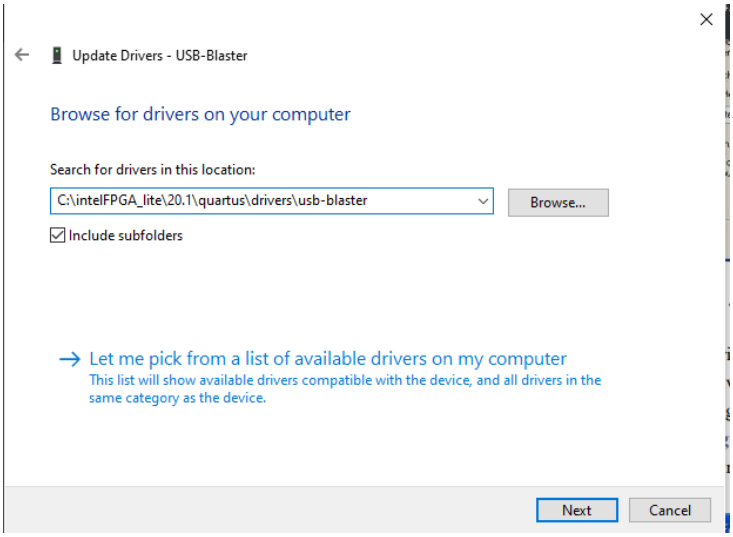
Εικόνα 44 Driver Properties

Αφού διαπιστώσουμε ότι δεν υπάρχει κάποιος driver εγκατεστημένος κλικάρουμε στο Update Driver, Αυτό θα μας ανοίξει ένα καινούριο παράθυρο, σε αυτό το παράθυρο μας δίνονται δυο επιλογές, η αυτόματη αναζήτηση driver από τον υπολογιστή, και η αναζήτηση driver στα αρχεία του υπολογιστή, όπως προαναφέραμε θα ψάξουμε στα αρχεία του υπολογιστή για να βρούμε τους drivers που είναι αποθηκεμένοι στα αρχεία του Quartus II.



Εικόνα 45 αναζήτηση driver

Όταν επιλέξουμε να αναζητήσουμε driver στα αρχεία του υπολογιστή θα μας πάει σε ένα καινούριο παράθυρο, σε αυτό το παράθυρο έχουμε την δυνατότητα πατώντας το Browse να αναζητήσουμε τους drivers στα αρχεία του υπολογιστή, και ποιο συγκεκριμένα στα αρχεία του Quartus, στην θέση C:\intelFPGA_lite\20.1\quartus\drivers\usb-blaster.



Εικόνα 46 θέση αρχείου USB – Blaster

Για να βεβαιωθούμε ότι έχουμε κατεβάσει σωστά τους drivers μπορούμε να κάνουμε μερικά ευκολά και γρηγορά τεστ, μέσα από εφαρμογές που θα βρούμε στο φάκελο του FPGA μας, το DE2-115 συνοδεύεται έναν Πίνακα Ελέγχου που επιτρέπει στους χρήστες να έχουν πρόσβαση σε διάφορα στοιχεία στο FPGA από έναν κεντρικό υπολογιστή. Η εφαρμογή μπορεί να χρησιμοποιηθεί για να επαληθεύσει τη λειτουργικότητα των εξαρτημάτων στον πίνακα ή να χρησιμοποιηθεί ως εργαλείο εντοπισμού σφαλμάτων κατά την ανάπτυξη κώδικα RTL.

Ο πίνακας ελέγχου βρίσκετε μέσα στον φάκελο DE2-155\DE2_115_tools\DE2_115_control_panel για να τον χρησιμοποιήσουμε αρκεί να τρέξουμε το πρόγραμμα DE2_115_ControlPanel.exe

Πρέπει να γίνει λήψη συγκεκριμένου κυκλώματος ελέγχου στην πλακέτα FPGA πριν ο πίνακας ελέγχου μπορεί να του ζητήσει να εκτελέσει τις απαιτούμενες εργασίες. Το πρόγραμμα θα καλέσει εργαλεία Quartus II για λήψη του κυκλώματος ελέγχου στην πλακέτα FPGA μέσω σύνδεσης USB-Blaster [USB-0].

Για να ενεργοποιήσουμε τον Πίνακα Ελέγχου, εκτελούμε τα ακόλουθα βήματα:

1. Πρέπει να βεβαιωθούμε ότι το Quartus II 10.0 ή νεότερη έκδοση είναι εγκατεστημένη με επιτυχία στον υπολογιστή μας.

2. Ρυθμίζουμε τον διακόπτη RUN/PROG στη θέση RUN.

3. Συνδέουμε το παρεχόμενο καλώδιο USB στη θύρα USB Blaster, συνδέουμε το τροφοδοτικό 12V και ενεργοποιούμε τον διακόπτη τροφοδοσίας.

4. Ξεκινάμε το εκτελέσιμο DE2_115_ControlPanel.exe στον κεντρικό υπολογιστή. Θα εμφανιστεί ο Πίνακας Ελέγχου που φαίνεται στην εικόνα 47.

Εικόνα 47 DE2-115 Control Panel

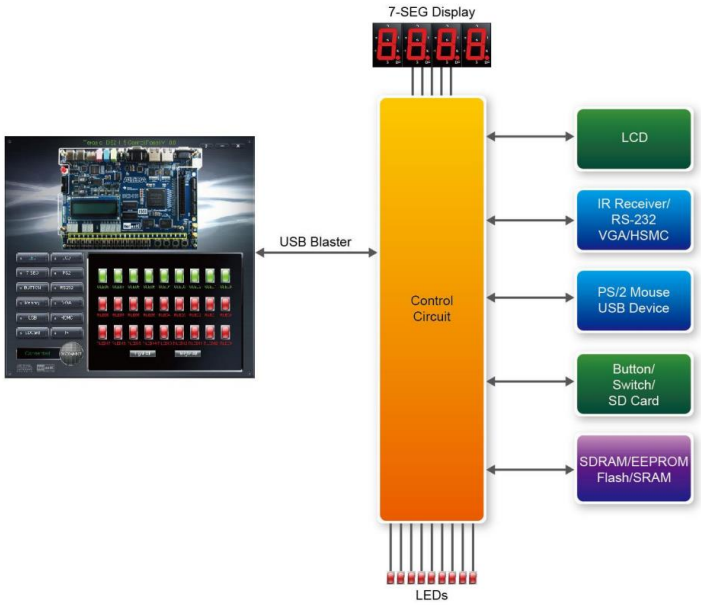
5. Η ροή δυαδικών ψηφίων DE2_115_ControlPanel.sof φορτώνεται αυτόματα μόλις ξεκινήσει το DE2_115_control_panel.exe.

6. Σε περίπτωση που η σύνδεση αποσυνδεθεί, κάντε κλικ στο CONNECT όπου το .sof θα φορτωθεί ξανά στο FPGA.

7. Σημείωση, ο Πίνακας Ελέγχου θα καταλαμβάνει τη θύρα USB μέχρι να κλείσουμε αυτήν τη θύρα. δεν μπορούμε να χρησιμοποιήσουμε το Quartus II για λήψη ενός αρχείου διαμόρφωσης στο FPGA μέχρι να κλείσει η θύρα USB.

8. Ο Πίνακας Ελέγχου είναι τώρα έτοιμος για χρήση. Το ελέγχουμε ρυθμίζοντας την κατάσταση ON/OFF για ορισμένα LED και παρατηρώντας το αποτέλεσμα στον πίνακα DE2-115.

Η ιδέα του πίνακα ελέγχου DE2-115 απεικονίζεται στην εικόνα 48. Το "κύκλωμα ελέγχου" που εκτελεί τις λειτουργίες ελέγχου υλοποιείται στο FPGA. Επικοινωνεί με το παράθυρο Control Panel, το οποίο είναι ενεργό στον κεντρικό υπολογιστή, μέσω του συνδέσμου USB Blaster.

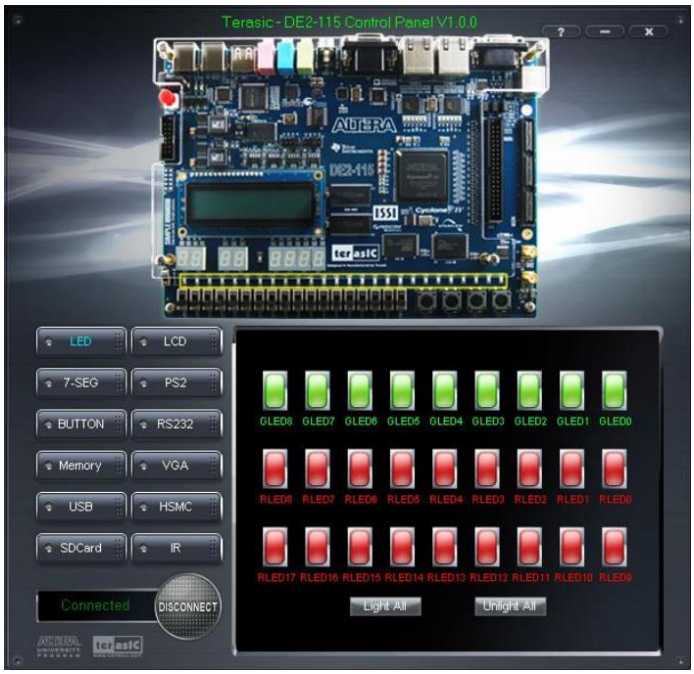


Εικόνα 48 DE2-115 Control Panel concept

Ο πίνακας ελέγχου DE2-115 μπορεί να χρησιμοποιηθεί για να ανάψει τα LED, να αλλάξει τις τιμές που εμφανίζονται σε οθόνες 7 - segment και LCD, να παρακολουθεί την κατάσταση των κουμπιών/διακοπτών, να διαβάζει/γράφει τα SDRAM, SRAM, EEPROM και Flash Memory, να παρακολουθεί την κατάσταση συσκευής USB, επικοινωνία με το ποντίκι PS/2, εξαγωγή μοτίβου χρώματος VGA στην οθόνη VGA, επαληθεύση τη λειτουργικότητα των εισόδων/εξόδων σύνδεσης HSMC, επικοινωνία με τον υπολογιστή μέσω προτύπου RS-232 και διάβασμα τις πληροφορίες προδιαγραφών της κάρτας SD. Η δυνατότητα ανάγνωσης/εγγραφής μιας λέξης ή ενός ολόκληρου αρχείου από/προς τη μνήμη Flash επιτρέπει στο χρήστη να αναπτύξει εφαρμογές πολυμέσων (Flash Audio Player, Flash Picture Viewer) χωρίς να ανησυχεί για το πώς θα δημιουργήσει έναν προγραμματιστή μνήμης.

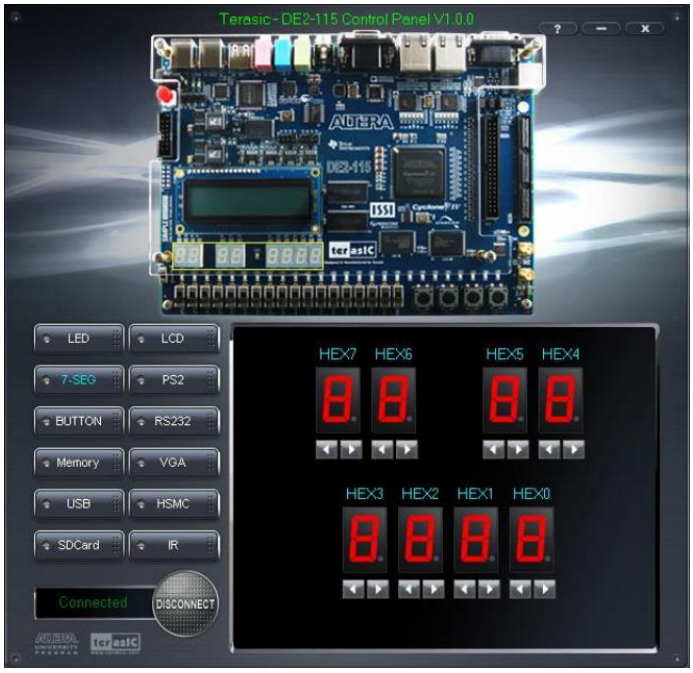
Μια απλή λειτουργία του Πίνακα Ελέγχου είναι να επιτρέψει τη ρύθμιση; των τιμών που εμφανίζονται σε LED, 7 - segment displays και οθόνη LCD χαρακτήρων.

Η επιλογή της καρτέλας LED οδηγεί στο παράθυρο της εικόνας 49. Εδώ, μπορούμε να ενεργοποιήσουμε ή να απενεργοποιήσουμε απευθείας τις λυχνίες LED μεμονωμένα ή ομαδικά, κάνοντας κλικ στο "Light All" ή "Unlight All".



Εικόνα 49 Controlling LEDs

Η επιλογή της καρτέλας 7-SEG οδηγεί στο παράθυρο που εμφανίζεται στην εικόνα 50. Από το παράθυρο, χρησιμοποιούμε τα βέλη αριστερά-δεξιά για να ελέγξουμε τα μοτίβα 7-SEG στον πίνακα DE2-115 που ενημερώνονται αμέσως.



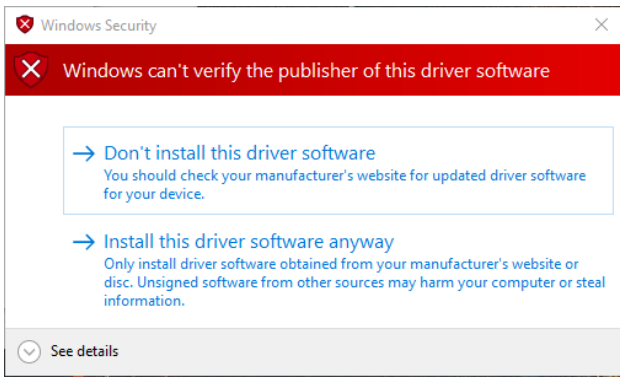
Εικόνα 50 Controlling 7-SEG display

7.6 ΤΑ ΠΡΟΒΛΗΜΑΤΑ ΣΤΗΝ ΣΥΝΔΕΣΗ (DRIVERS)

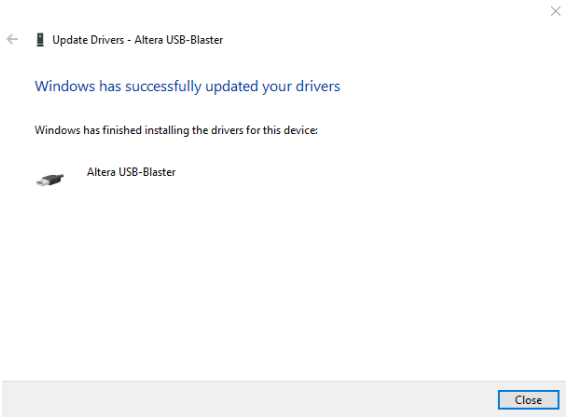
Εάν αντιμετωπίσουμε προβλήματα κατά την διαρκείας της εγκατάστασης του USB – driver υπάρχουν κάποια βήματα που μπορούμε να ακολουθήσουμε για να αντιμετωπίσουμε αυτό το πρόβλημα. Ο σκοπός αυτόν τον βημάτων είναι η απενεργοποιήσει της επιβολής του driver signature.

Τα βήματα που πρέπει να ακολουθήσουμε είναι τα εξής:

1. Πρώτα θα πάμε στης ρύθμισης για τα windows 10 και θα αναζητήσουμε advanced startup options
2. Κάτω από το Advanced startup θα επιλέξουμε restart now
3. Μετά από λίγο θα εμφανιστή η οθόνη για να επιλέξουμε ποιο βήμα θέλουμε να πάρουμε, εμείς θα επιλέξουμε troubleshoot, θα μετάβουμε σε μια άλλη οθόνη με επιλογές, εκεί θα επιλέξουμε Advanced options, που θα μας μεταφέρει σε μια ακόμη οθόνη επιλογής, θα επιλέξουμε Startup Settings
4. Θα εμφανιστεί μια οθόνη που θα μας λέει τι θα μπορούμε να αλλάξουμε και ένα κουμπί "Restart" – το πατάμε
5. Εάν ο boot drive έχει BitLocker κρυπτογράφηση θα χρειαστεί να εισάγουμε το κλειδί ανάκτησης
6. Θα μας δοθεί ένα μενού επιλογών, ο αριθμός 7 είναι αυτός που ψάχνουμε γιατί απενεργοποιεί την επιβολής του driver signature
7. Όταν ολοκληρωθεί η επανεκκίνηση του υπολογιστή θα επαναλάβουμε την διαδικασία ενημέρωσης του USB – Blaster από το Device manager, με τον ίδιο τρόπο που προσπαθήσαμε και πριν, θα επιλέξουμε τον φάκελο με τους drivers στην θέση C:\intelFPGA_lite\20.1\quartus\drivers\usb-blaster, θα μας εμφανιστεί το μήνημα της εικόνας 51. Παρόλα αυτά η εγκατάσταση του driver έχει γίνει κανονικά οπότε θα επιλέξουμε την δεύτερη επιλογή “Install this driver software anyway”.

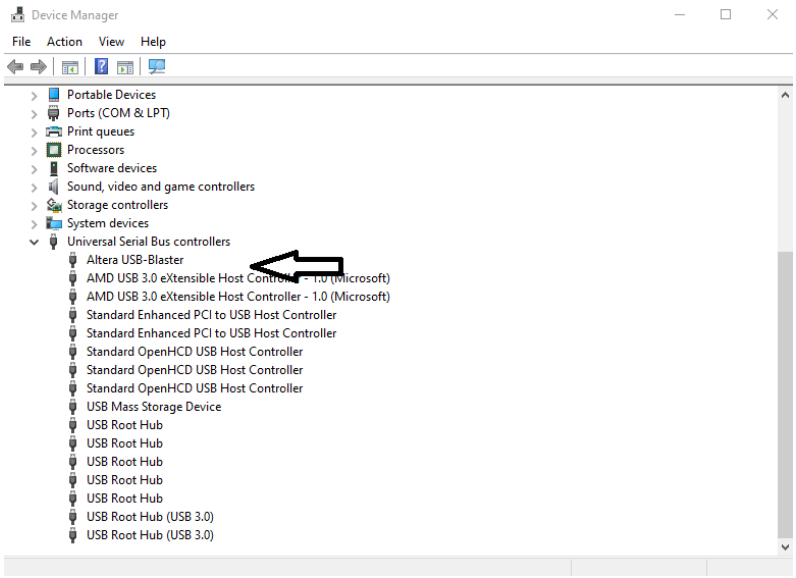


Εικόνα 51 windows can't verify the publisher of this driver



Εικόνα 52 ολοκλήρωση ενημέρωσης Altera USB - Blaster

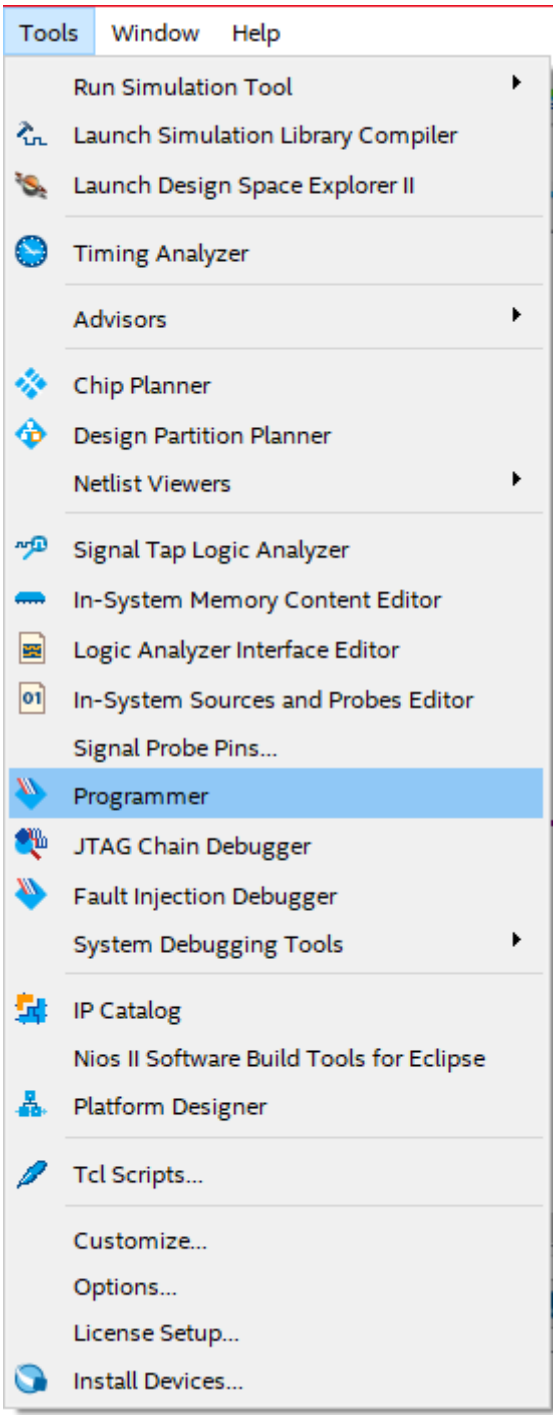
Μετά από αυτήν την διαδικασία ο USB – Blaster θα πρέπει να έχει μεταφερθεί στην θέση Universal Serial Bus controllers



Εικόνα 53 Universal Serial Bus controllers

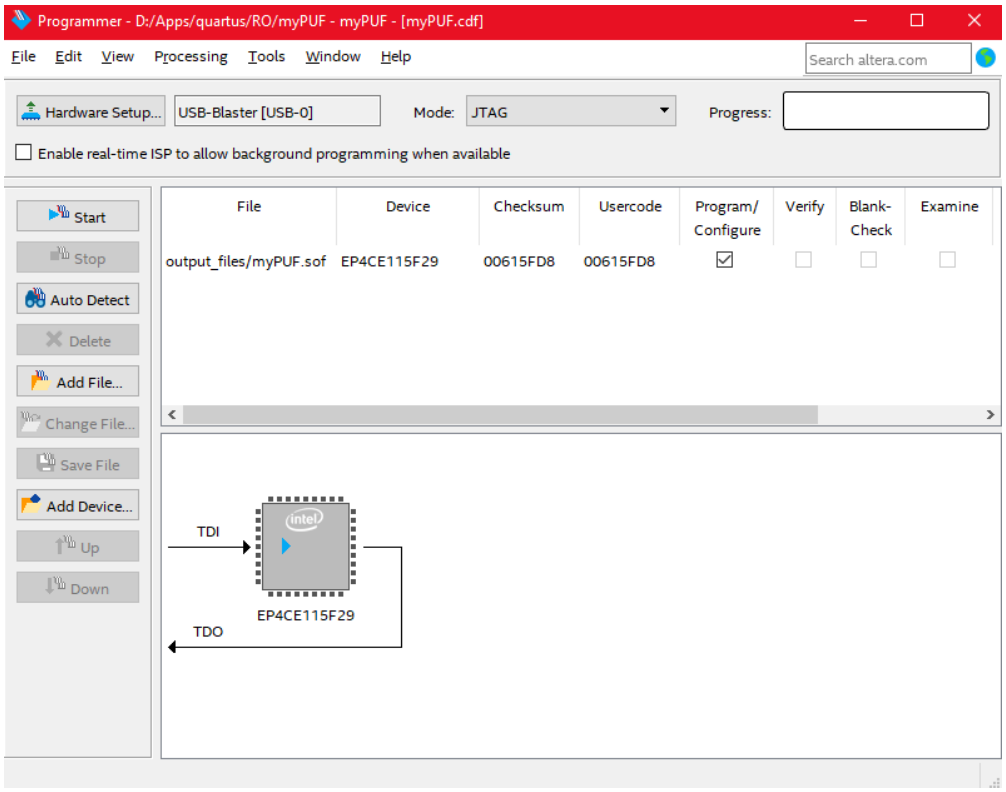
7.7 Πως περνάμε το πρόγραμμα στο FPGA

Για να περάσουμε το πρόγραμμα στο FPGA θα χρειαστούμε την εφαρμογή Programmer του Quartus, αυτήν την εφαρμογή την βρίσκουμε στα tools -> Programmer.



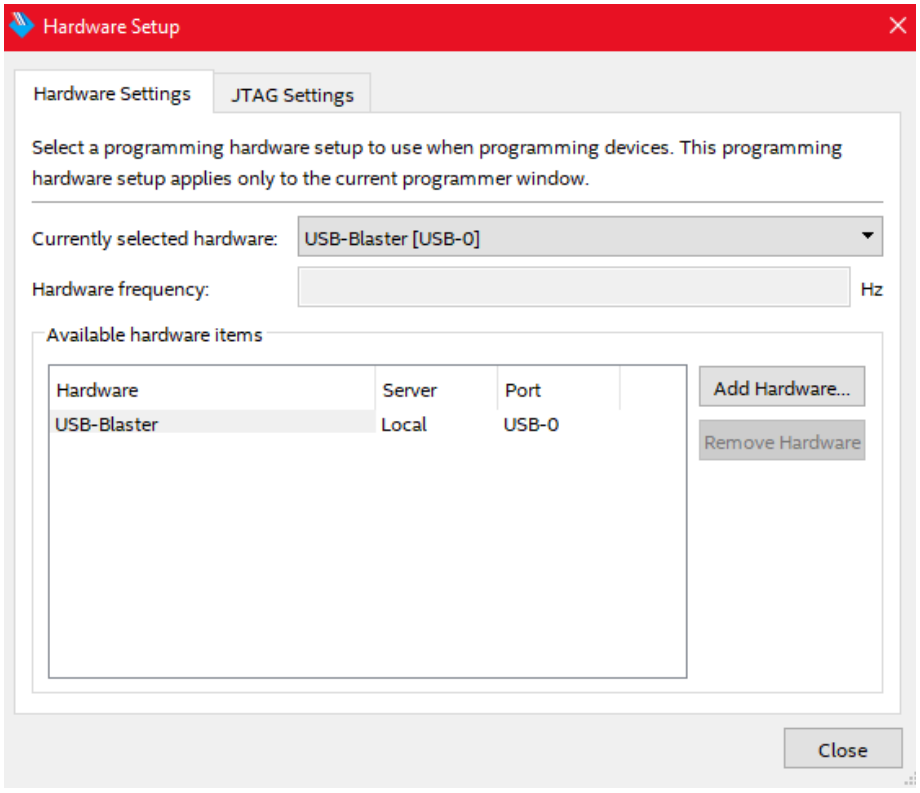
Εικόνα 54 θέση Programmer

Όταν επιλέξουμε τον Programmer θα ανοίξει ένα καινούριο παράθυρο, μέσα από αυτό μπορούμε να επιλέξουμε ποιο αρχείο θέλουμε να τρέξει η εφαρμογή και σε ποια συσκευή.



Εικόνα 55 Programmer

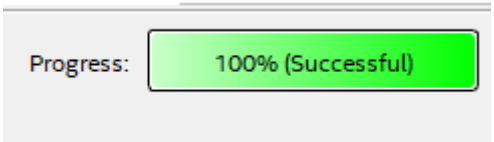
Από το κουμπί Hardware Setup μπορούμε να επιλέξουμε της ρυθμίσεις για την σύνδεση του Hardware μας, στην περίπτωση μας θα επιλέξουμε USB – Blaster [USB-0] και για Mode JTAG.



Εικόνα 56 Currently selected hardware

Αφού επιλέξουμε το hardware και το mode σειριά έχει η επιλογή του αρχείου που θέλουμε να περάσουμε στο FPGA αυτό μπορούμε να το κάνουμε με το κουμπί Auto Detect, αν το Auto Detect δεν μας φέρει τον επιθυμητό φάκελο μέσα από το Add File μπορούμε να βρούμε το κατάλληλο αρχείο, το αρχείο που ψάχνουμε βρίσκετε στο φάκελο output_files και έχει κατάληξη .sof.

Πλέον είμαστε έτοιμοι να τρέξουμε το πρόγραμμα πατώντας το start, αν όλα έχουν πάει καλά θα πρέπει στο Progress να γραφεί 100% (Successful) όπως φαίνεται στην εικόνα 57. Αν δεν φαίνεται αυτό πρέπει να ξανακοιτάξουμε κάποιο από τα προηγούμενα βήματα για να αντιμετωπίσουμε το πρόβλημα.



Εικόνα 57 Progress

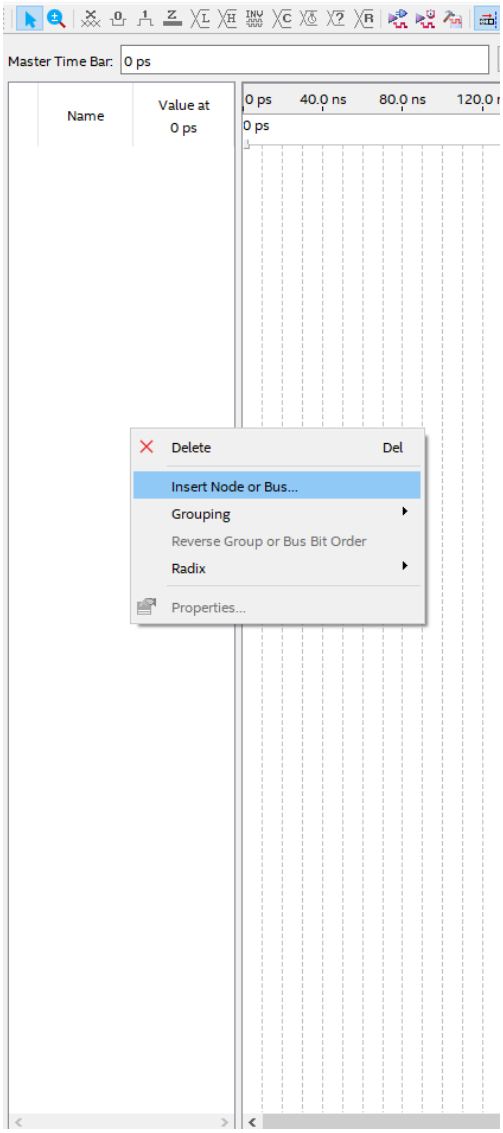
8 Αποτελέσματα προσομοιώσεων

Αφού έχουμε κατασκευάσει το project και έχουμε βεβαιωθεί ότι όλα τα κομμάτια του έχουν τα αποτελέσματα που περιμέναμε, και αφού εγκαταστήσαμε το FPGA στο υπολογιστή μας και έχουμε βεβαιωθεί ότι υπάρχει η σωστή ανταπόκριση μέσα από της εντολές που του δώσαμε από την εφαρμογή control panel είμαστε σε θέση να δοκιμάσουμε να τρέξουμε όλο το project πρώτα στο Quartus II και μετά να δοκιμάσουμε να τρέξει και στο FPGA.

8.1 Quartus VWF

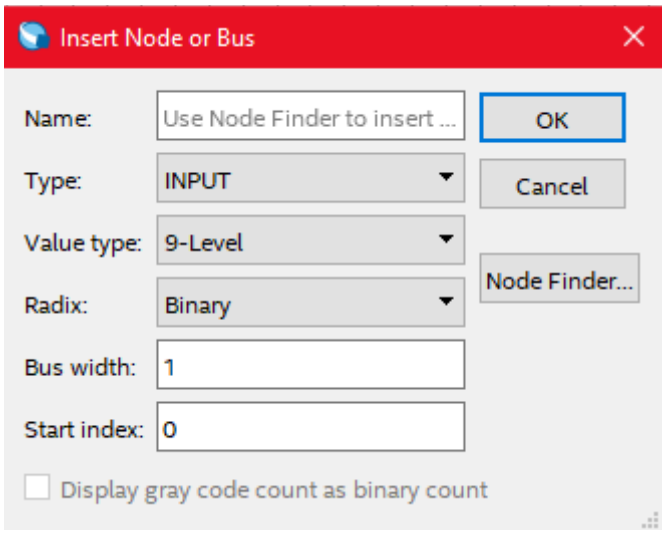
Για να φτιάξουμε ένα αρχείο vwf θα πάμε στο file -> new -> University program VWF όταν μας ανοίξει το καινούριο παράθυρο έχουμε την δυνατότητα να προσθέσουμε όλα τα inputs και όλα τα outputs που έχουμε στο project μας.

Για να προσθέσουμε τα inputs και outputs θα πάμε στο αριστερό μέρος της οθόνης, από εκεί θα πατήσουμε δεξί κλικ και επιλέξουμε το insert Node or BUS.



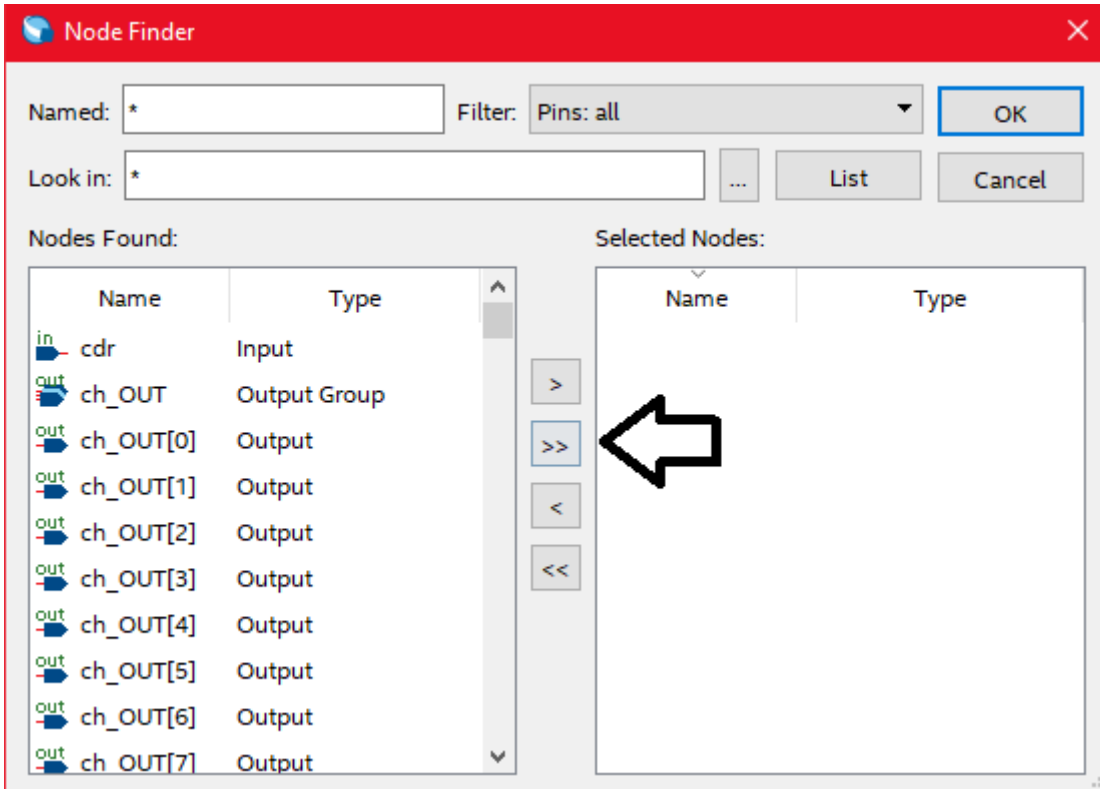
Εικόνα 58 inser Node or Bus

Θα μας ανοίξει ένα καινούριο παράθυρο, από αυτό θα επιλέξουμε το Node Finder οπού θα μας δώσει την δυνατότητα να προσθέσουμε όλα τα inputs και τα outputs.



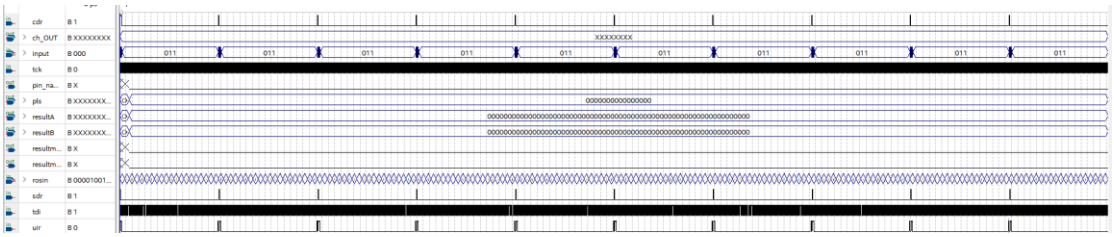
Εικόνα 59 Node Finder

Στο επόμενο παράθυρο όταν επιλέξουμε το list θα ανοίξει μια λίστα με όλα τα inputs και τα outputs που έχουμε στο block diagram/schematic file, θα επιλέξουμε το " >> "για να περάσουν όλα στο VWF. Αφού πατήσουμε OK και στα δυο παράθυρα θα περάσουν όλα τα inputs και τα outputs στο VWF.



Εικόνα 60 Node Finder List

Θα μας εμφανιστεί κάτι παρόμοιο με την εικόνα 61, δηλαδή όλα τα inputs και τα outputs μαζί στο ίδιο παράθυρο. Τα inputs είναι αυτά που έχουν λογική τιμή 0 ή τιμή 0, ενώ τα output είναι όλα αυτά που μπορούμε να δούμε " XXXXXX ".



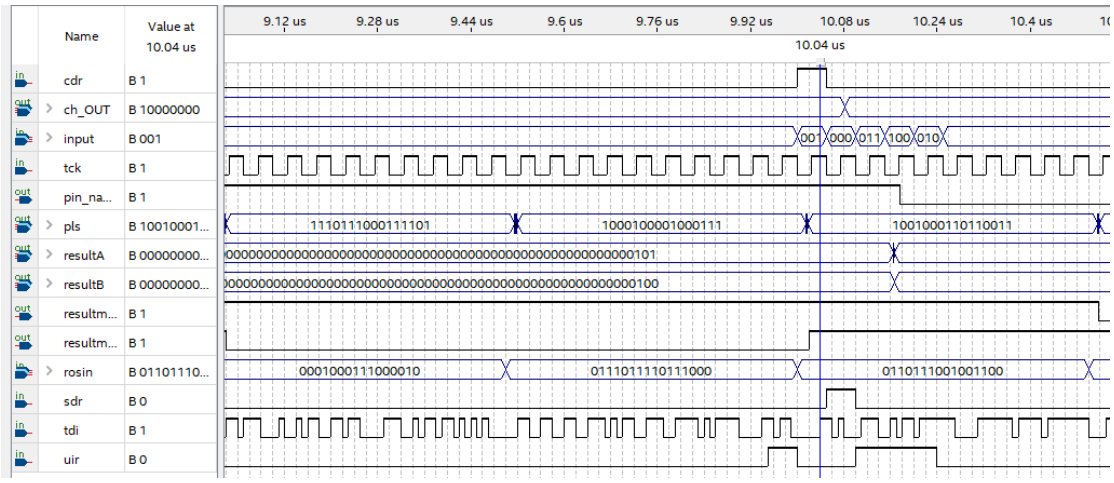
Εικόνα 61 αρχικό VWF

Αφού σιγουρευτούμε ότι έχουν περάσει όλα τα inputs και outputs που θέλουμε είμαστε έτοιμοι να καταχωρήσουμε της κατάλληλες τιμές στα inputs για να τρέξουμε το πρόγραμμα.

Αρχικά στο input για 50ns θα αρχίσουμε με την τιμή “000 ” για να κάνουμε push challenge και τα ετοιμαστούμε για να πάρουμε την επόμενη τιμή, άμεσος επόμενη τιμή θα είναι το “011 “ για να σταματήσουμε τους μετρητές, ακολουθεί το “100” που θα κάνει reset τον μετρητή και το “010” και θα ξεκινήσει πάλι τον μετρητή, στην συνέχεια θα ξανά δώσουμε την εντολή “011” για να σταματήσουν οι μετρητές μέχρι τα 10us. Θα ακολουθήσει η εντολή “001” η οποία θα μας δώσει την απάντηση.

Στην συνέχεια στο cbr θα βάλουμε την τιμή 1 σε όλα τα σημεία που στο input δίνουμε την τιμή “001”. Στο rosin θα βάλουμε τυχαίες τιμές που θα αλλάζουν κάθε 500 ns. το sdr θα έχει την τιμή 1 σε όλα τα σημεία που στο input έχουμε δώσει την τιμή “000”. Ακολουθεί το tck, στο οποίο θα βάλουμε ένα ρολόι που θα αλλάζει κάθε 50ns ξεκινώντας με τιμή “0”. Στο uir θα βάλουμε τιμή 1 οπού στο input έχουμε δώσει της τιμές “011” ”100” ”010”. Κλείνοντας στο tdi θα του βάλουμε από την αρχή μέχρι το τέλος random τιμές.

Το VWF αρχείο πρέπει να είναι κάπως έτσι.

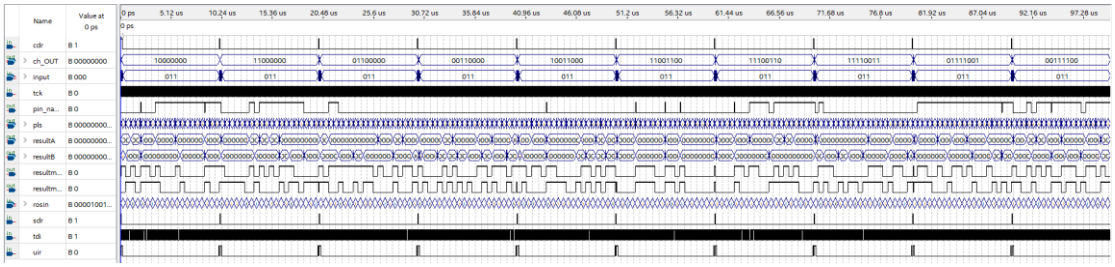


Εικόνα 62 πρώτο bit της υπογραφής

Αφού ολοκληρώσαμε την τοποθέτηση των inputs και τους αναθέσουμε και τιμές ήρθε η ώρα να δούμε τι αποτελέσματα θα μας δώσουν τα output,. Για να πάρουμε αποτελέσματα θα επιλέξουμε από την μπάρα το run timing simulation.

Στα αποτελέσματα της προσομοίωσης το μοναδικό “αποτύπωμα” της συσκευής το παίρνουμε βλέποντας το pin_name3 κάθε φορά που στο input δίνουμε την εντολή “001”.

Αυτά θα είναι και τα bit της υπογραφής, δηλαδή, θα πάρουμε 9 τιμές από το pin_name3 που θα διαμορφώσουν την τελική υπογραφή. Στο παράδειγμα μας η υπογραφή θα είναι “100000101”.

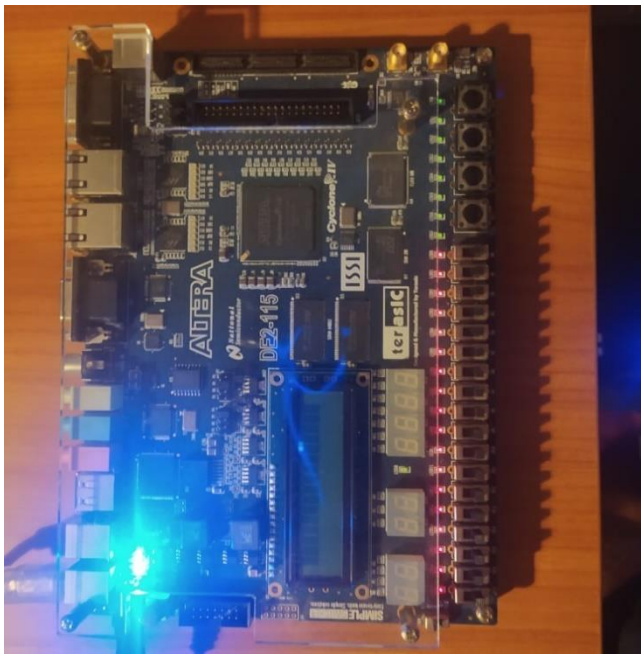


Εικόνα αποτελέσματα προσομοίωσης

8.2 FPGA

Αφού είδαμε τα αποτελέσματα που παίρνουμε μέσα από το Quartus II ήρθε η ώρα να τρέξουμε το project και σε ένα πραγματικό FPGA, για να τρέξουμε το project σε ένα FPGA θα ακολουθήσουμε τα βήματα που είδαμε σε προηγούμενο κεφάλαιο.

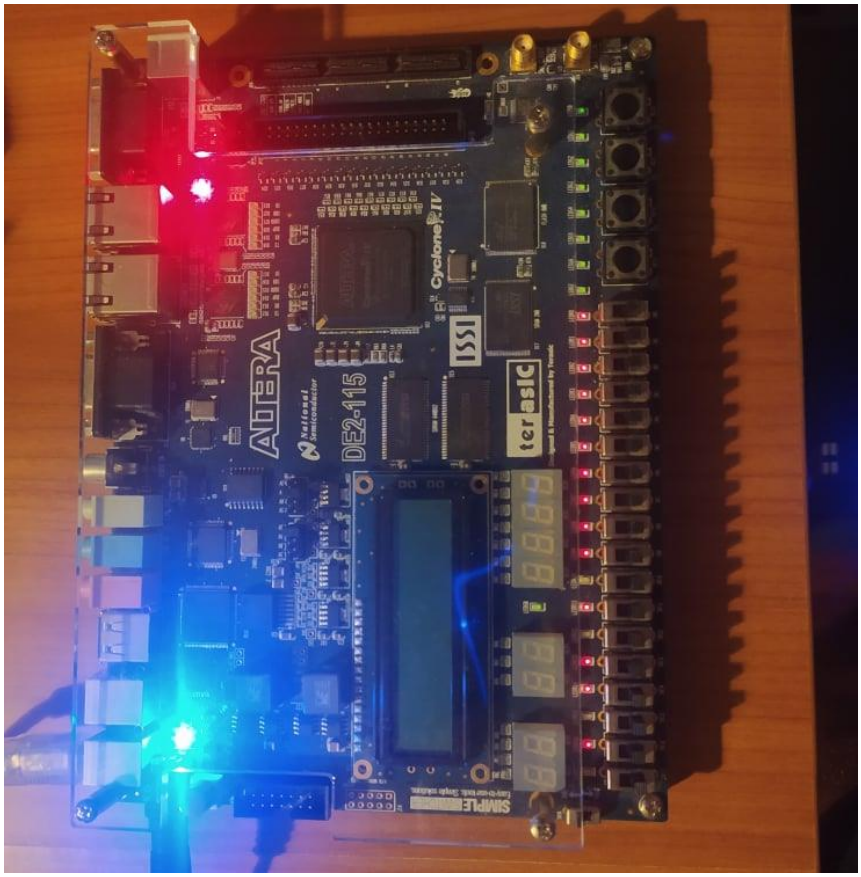
Είχαμε δει ότι για να περάσουμε το project σε ένα FPGA πρέπει να χρησιμοποιήσουμε την εφαρμογή Programmer που την βρίσκουμε στα tools του Quartus II, μέσα από την εφαρμογή αφού επιλέξουμε το κατάλληλο αρχείο και τα καταλληλά setting για την επικοινωνία του FPGA μας είμαστε έτοιμη για τα αποτελέσματα μας, πατώντας το start ξεκινάει η διαδικασία του Programmer για να περάσουν οι τιμές στο FPGA, κατά την διάρκεια της διαδικασία στο FPGA θα ανοίξουν όλα τα LED όπως φαίνεται στην εικόνα 64.



Εικόνα 63 αρχηγό FPGA

Όταν η διαδικασία του Programmer τελειώσει, δηλαδή όταν το progress του programmer γράψει 100% (Successful) και οι τιμές περάσουν στο FPGA πρέπει να σβήσουν κάποια

από τα LED που είναι ανοιχτά. Το γεγονός ότι ο programmer έκανε επιτυχώς το πέρασμα τον τιμών και έγραψε 100% (Successful) δεν σημαίνει ότι το project είναι σωστό, σημαίνει απλά ότι δεν υπάρχει κάποιο λογικό λάθος. Κανονικά από το project πρέπει να πάρουμε αποτέλεσμα παρόμοιο με την εικόνα 65.



Εικόνα 64 Τελικό FPGA

9 Συμπεράσματα

Στην παρούσα εργασία κατασκευάσαμε μια Φυσικά μη κλωνοποιήσιμη συνάρτηση και ποιο συγκεκριμένα ένα ΦΜΚΣ καθυστέρησης με σκοπό την Υλοποίηση και αξιολόγηση μηχανισμού μοναδικής αναγνώρισης σε FPGA. Η υλοποίηση και η αξιολόγηση έγινε στα προγράμματα Quartus II και το Modelsim της Intel, και για FPGA χρησιμοποιήσαμε το Altera DE2-115. Το Quartus και το Modelsim μας έδωσε την δυνατότητα να προσομοιώσουμε έναν Ταλαντωτή δακτυλίου μέσα από της λειτουργείς που προφέρουν για δημιουργία ενός project με της ρύθμισης που θέλουμε εμείς και να επιλέξουμε και ποια οικογένεια cyclone επιθυμούμε ανάλογα με ποιο FPGA θα χρησιμοποιήσουμε. Επίσης μέσα από το Quartus έχουμε την δυνατότητα να ελέγξουμε και να αξιολογήσουμε όλα τα κομμάτια του project ξεχωριστά για να βεβαιωθούμε ότι δουλεύουν σωστά πριν τρέξουμε όλο το project μαζί. Όλες αυτές της δυνατότητες που μας δίνει το Quartus της θέλουμε γιατί πρέπει να είμαστε σίγουροι ότι όλες οι ιδιότητες μίας Φυσικά μη κλωνοποιήσιμης συνάρτησης τηρούνται στο project μας. Οι ιδιότητες μίας Φυσικά μη κλωνοποιήσιμης συνάρτησης είναι οι εξής: α) Unclonability ή αλλιώς η αδύναμά να αποκτηθεί το "αποτύπωμα" της συσκευής μας από μια άλλη β) Uniqueness δηλαδή η μοναδικότητα μιας ΦΜΚΣ, η επόμενη ιδιότητα είναι γ) Unpredictability, από της ποιο βασικές ιδιότητες μιας ΦΜΚΣ, η δυνατότητα να είναι απρόβλεπτο και να μην ακολουθεί ένα συγκεκριμένο μοτίβο το οποίο είναι εύκολο κάποιος που δεν θέλουμε να αποκτήσει πρόσβαση σε αυτό δ) One-Way Property που αντιπροσωπεύει την ευκολία για υπολογισμούς με κάθε είσοδο και ταυτόχρονα την δυσκολία αντιγραφής από μια τυχαία είσοδο. Τέλος οι ΦΜΚΣ πρέπει να έχουν ε) Feasibility δηλαδή αυτό που θα σχεδιάσουμε να μπορεί να περαστεί σε ένα ενσωματωμένο κύκλωμα. Ένα ΦΜΚΣ που θα σχεδιάσουμε πρέπει πάντα με τα ιδιά inputs να μας δίνει τα ιδιά outputs, όμως αυτό μπορεί να επηρεαστεί από εξωτερικούς παράγοντες όπως η θερμοκρασία ή η τροφοδοσία.

Όπως είδαμε και παραπάνω υπάρχουν πολύ μέθοδοι για να κατασκευάσουμε ένα ΦΜΚΣ που χωρίζονται σε δυο κατηγορείς οι ΦΜΚΣ καθυστέρησης και οι ΦΜΚΣ μνήμης, εμείς ασχοληθήκαμε με της ΦΜΚΣ καθυστέρησης και ποιο συγκεκριμένα με τον ταλαντωτή δακτυλίου που αξιόλογη της διαφορές καθυστέρησης μέσω μετρήσεων συχνότητας που εκμεταλλεύονται της ταλάντωσης στους βρόχους. Αυτός είναι ένας σχετικά εύκολος και γρήγορος τρόπος για την δημιουργία μιας ΦΜΚΣ που μπορεί να χρησιμοποιηθεί σε ένα FPGA.

Με την βοήθεια του Quartus II και του Modelsim είδαμε από πρώτο χέρι πως δουλεύει ο ταλαντωτή δακτυλίου αφού κατά την διάρκεια του project μπορούσαμε να βλέπουμε της τιμές από της εξόδους, και ποιες είναι οι μεταβολές τους όταν δίνουμε μια διαφορετική εντολή και αν αυτές οι μεταβολές συμβαδίζουν με τις ιδιότητες που είδαμε παραπάνω.

10 Βιβλιογραφία

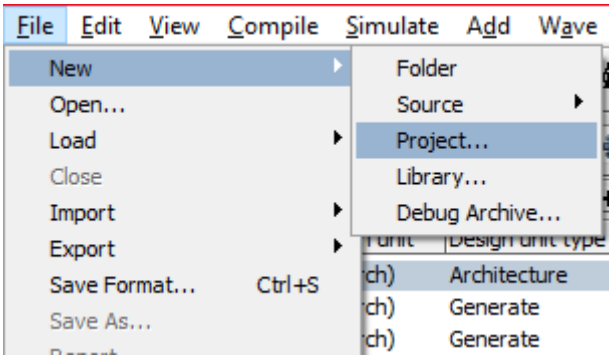
1. https://www.nature.com/articles/s41928-020-0372-5?proof=t
2. https://searchitoperations.techtarget.com/definition/hardware-security
3. https://www.sciencedirect.com/science/article/pii/B9780128124772000174
4. https://www.researchgate.net/publication/283776026_Introduction_to_Hardware_Security
5. https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html
6. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/archives/intro_to_quartus2.pdf
7. https://information-technology.web.cern.ch/services/software/modelsim
8. https://www.intel.com/content/www/us/en/programmable/quartushelp/13.0/mergedProjects/optimize/ace/acv_view_acv_overview.htm
9. https://www.intel.com/content/www/us/en/programmable/quartushelp/13.0/mergedProjects/reference/scripting/tcl_pro_command.htm
10. https://community.intel.com/t5/Programmable-Devices/Windows-10-driver-support-for-USB-Blaster/td-p/55323?fbclid=IwAR2mZy4nTWHHwVd5G7vkFfdZR-HXgxPKZlpg4j6tt1rILSzSAGedVF0UjPw
11. https://codilime.com/blog/FPGA-programming-how-it-works-and-where-it-can-be-used/
12. https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=502

11 Παράρτημα

11.1 Modelsim

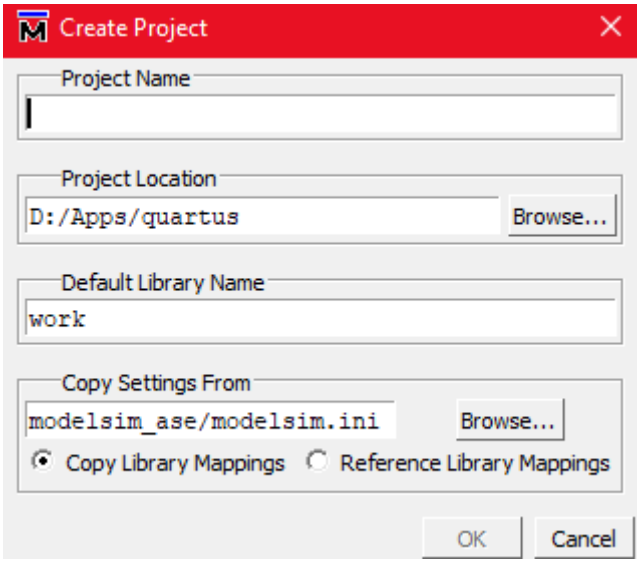
Όταν τελειώσουμε με την ρύθμιση του chip planner θα ελέγξουμε το project πριν προσπαθήσουμε να το τρέξουμε σε ένα πραγματικό FPGA. Για να ελέγξουμε όλο το project είναι καλυτέρα αν πάμε στην εφαρμογή του Modelsim, αντί της δυνατότητας που μας δίνει το Quartus για τον έλεγχο του input και του output. Στην εφαρμογή του Modelsim έχουμε περισσότερες δυνατότητες για ρύθμισης και μπορούμε να πάρουμε ποιο αναλυτικά αποτελέσματα, επίσης είναι ποιο εύκολο να προγραμματίσουμε την λειτουργία ενός μεγάλου project με πολλά και διαφορετικά inputs και outputs.

Για να φορτώσουμε το project στο Modelsim πατάμε πάνω αριστερά στο file -> new -> project και επιλέγουμε μέσα από τον φάκελο quartus τα αρχεία που θέλουμε να ανοίξουμε.



Εικόνα 65 new project Modelsim

μόλις το κάνουμε αυτό θα πρέπει να μας εμφανίζονται ένα καινούριο παράθυρο, μέσα από αυτό θα μας ζητηθεί να επιλέξουμε το όνομα του καινούριου project.

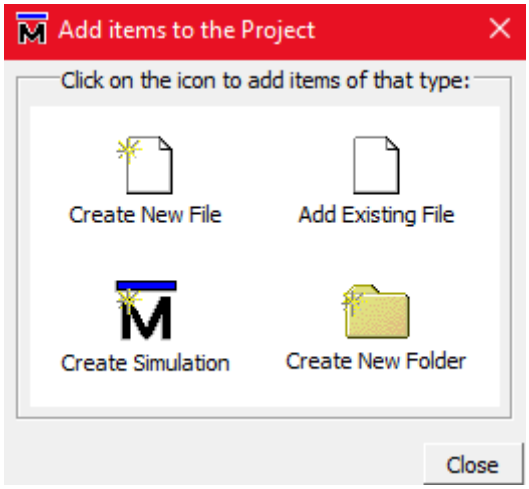


Εικόνα 66 όνομα project modalism

Στο επόμενο βήμα έχουμε την δυνατότητα να δημιουργήσουμε έναν καινούριο αρχείο με την επιλογή Create new File ή να προσθέσουμε ένα ήδη υπάρχον αρχείο με την

επιλογή Add Existing File, την επιλογή δημιουργίας προσομοίωσης με την επιλογή Create Simulation και την επιλογή δημιουργίας καινούριου φακέλου.

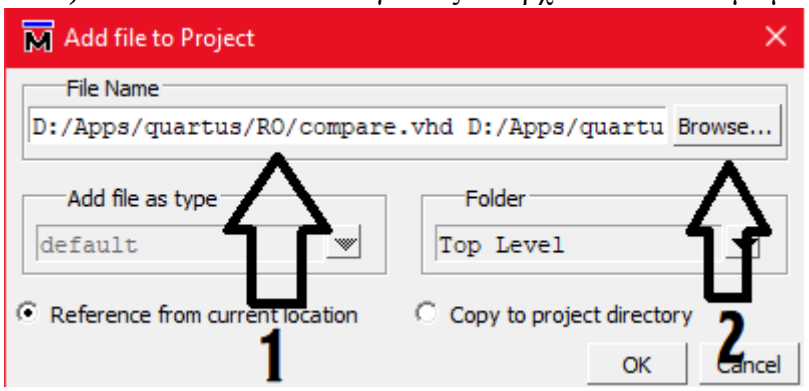
Εμείς αφού έχουμε δημιουργήσει τα αρχεία που θέλουμε στο Quartus έχουμε ήδη έτοιμα τα αρχεία για το Modelsim. Οπότε θα επιλέξουμε το Add Existing file για την επιλογή τον αρχείων μέσα από τον φάκελο του project που χέουμε φταίξει στο Quartus.



Εικόνα 67 επιλογή πρόσθεσης αρχείων στο Modelsim

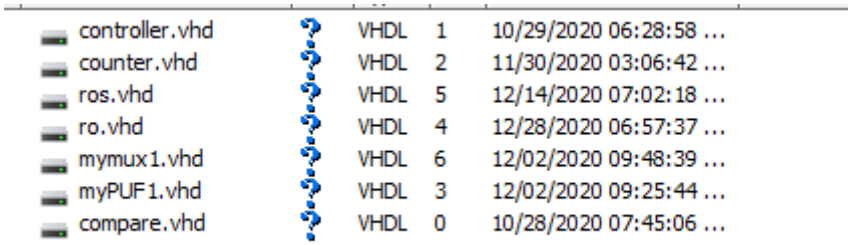
Αφού επιλέξουμε την επιλογή για προθήκη είδη υπάρχων αρχείων έχουμε την δυνατότητα να προσθέσουμε τα αρχεία με δυο μέσα :

a) Γράφοντας το όνομα του αρχείου μαζί με το file directory του
b) Από το Browse επιλέγοντας τα αρχεία που θέλουμε μέσα από το project



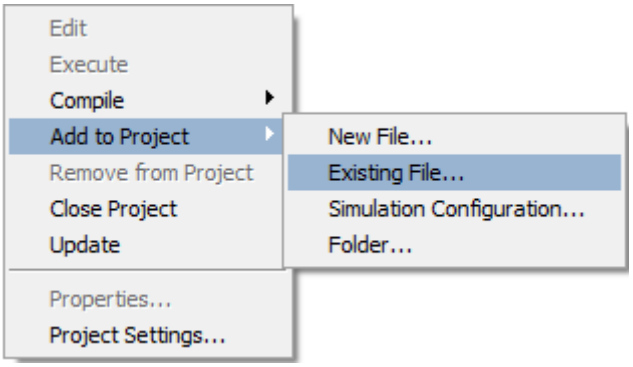
Εικόνα 68 File Directory / Browse for files

Τα αρχεία πρέπει να εμφανίζονται κάπως έτσι.



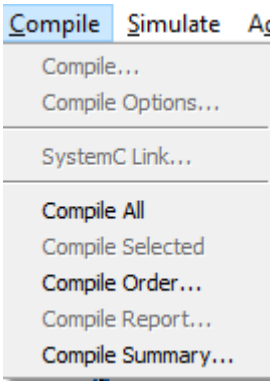
Εικόνα 69 Modelsim

Αν διαπιστώσουμε ότι μας λυπεί κάποιο αρχείο από το καινούριο μας project μπορούμε πολύ ευκολά να το προσθέσουμε πατώντας δεξί κλικ και επιλέγοντας Add to Project -> Existing File, με την ιδιά διαδικασία που ακολουθήσαμε για την αρχική προσθήκη αρχείων στο project.



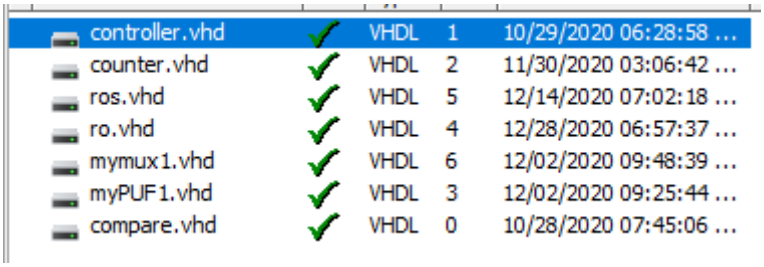
Εικόνα 70 Προσθήκη Αρχείου

Αφού φορτώσουμε το project και βεβαιωθούμε ότι έχουν περάσει όλα τα αρχεία που έχουμε κατασκευάσει, πρέπει να τα κάνουμε compile, για να το κάνουμε αυτό επιλέγουμε πάνω από την μπάρα το Compile -> Compile All.



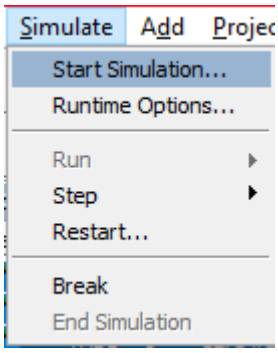
Εικόνα 71 Compile

Όταν τελειώσει το Compile, αν όλα έχουν γίνει σωστά θα μας εμφανιστεί ένα πράσινο «τικ» όπως φαίνεται στην εικόνα 45. Αν δεν μας εμφανιστεί το «τικ» σημάνει ότι υπάρχει κάποιο λάθος στον κώδικα VHDL και χρειάζεται διόρθωση.



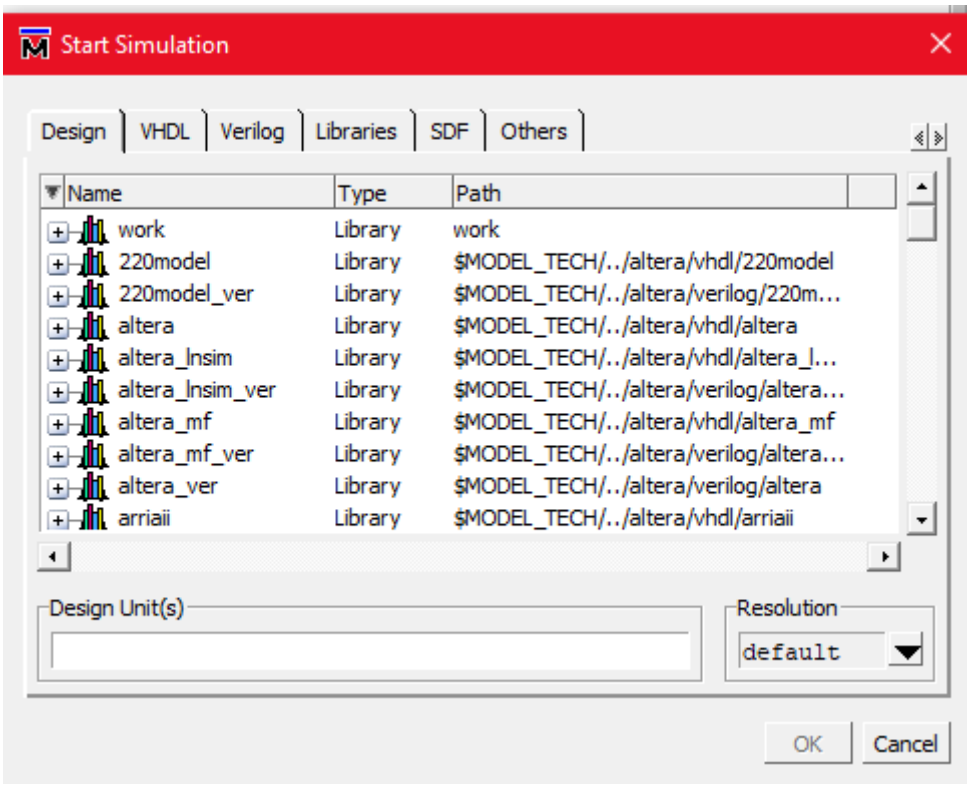
Εικόνα 72 Compile complete

Το επόμενο βήμα είναι να ξεκινήσουμε την προσομοίωση, για να κάνουμε προσομοίωση επιλέγουμε από την μπάρα το Simulate -> Start Simulation.



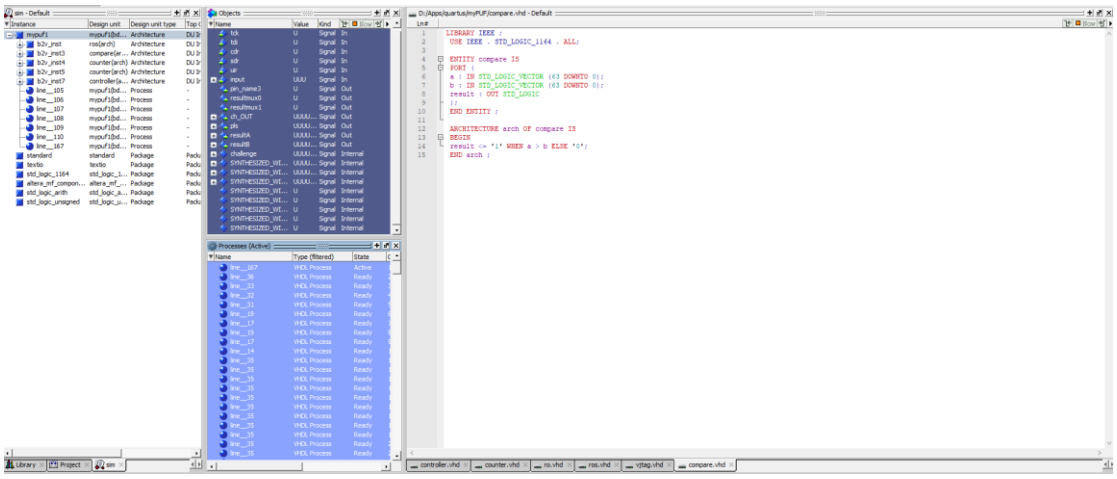
Εικόνα 73 Simulation

Θα μας ανοίξει αυτό το παράθυρο, στο design units θα βάλουμε το όνομα του project που φτιάξαμε «myPUF1» στην δικιά μας περίπτωση.

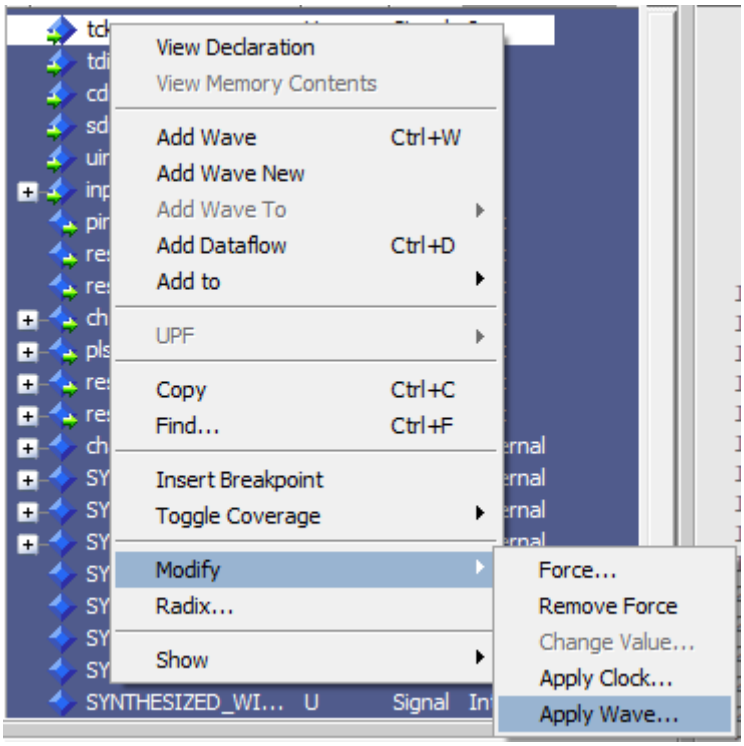


Εικόνα 74 Start Simulation

Μόλις πατήσουμε «οκ» μας ανοίξει ένα καινούργιο παράθυρο,



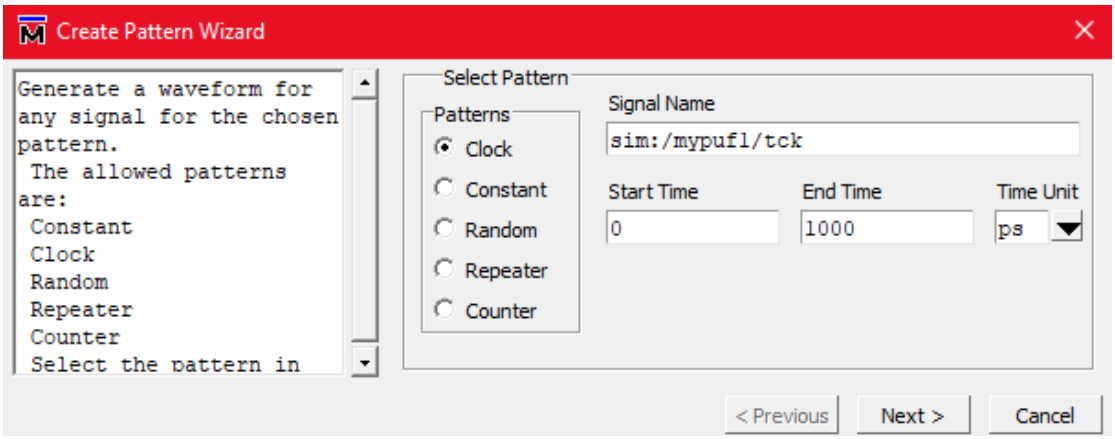
Σε αυτό το παράθυρο έχουμε την δυνατότητα τα τρέξουμε όλα τα κομμάτια του project. Για να βάλουμε τα κομμάτια που θέλουμε τρέξουμε, όταν επιλέξουμε το κομμάτι θα επιλέξουμε να του βάλουμε ένα σήμα ή ένα «κύμα» για να μπορέσουμε να πάρουμε αποτελέσματα. Η επιλογή μας δίνετε αν πατήσουμε δεξιά κλικ Modify -> Apply Clock / Apply Wave ανάλογα με ποιο από τα δυο θέλουμε.



Εικόνα 75 Apply Wave

Στο παράθυρο που θα ανοίξει μπορούμε να επιλέξουμε σε πόσα nanosecond θέλουμε να ξεκινήσει το σήμα και για ποσά χρονικό διάστημα. Επίσης μπορούμε να επιλέξουμε και

ποια μορφή θα έχει το σήμα, αν δηλαδή είναι συνεχόμενο, αν ακολουθεί κάποιο συγκείμενο μοτίβο, αν είναι τυχαίο ή επαναλαμβανόμενο.



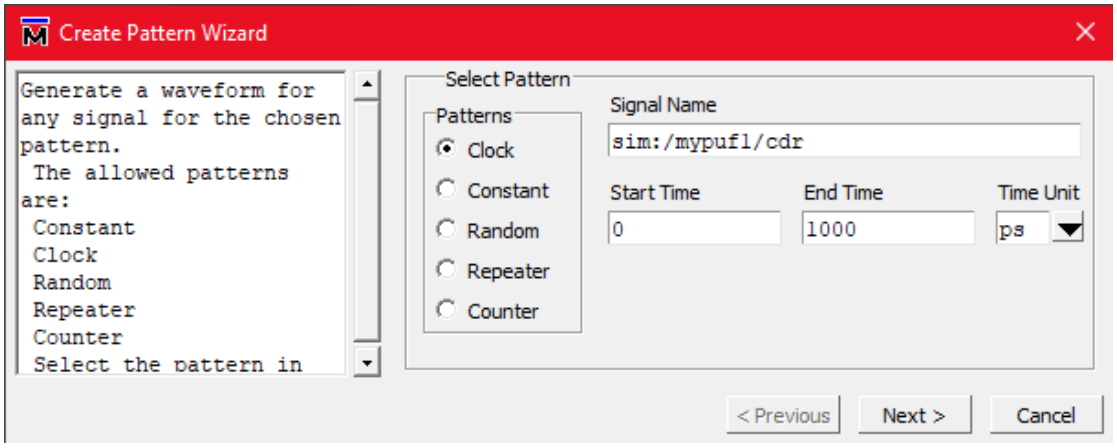
Εικόνα 76 παράθυρο επιλογής σήματος

Στην περίπτωση μας θα χρησιμοποιήσουμε την επιλογή Apply Wave.

Θα χρειαστεί να βάλουμε συνολικά εφτά inputs, οι ρυθμίσεις για τα input είναι οι εξής:

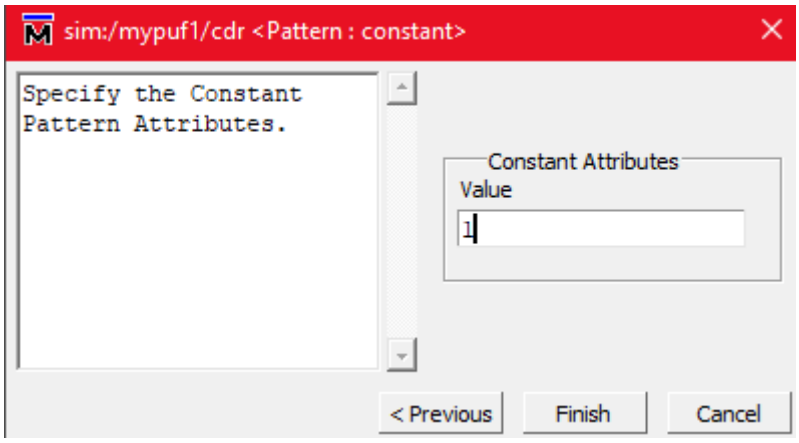
11.1.1 Cbr

Το cbr θα το έχουμε συνέχεια στην τιμή 1 για να μπορούμε να κάνουμε push challenge οπότε μας δίνετε η δυνατότητα. θα βάλουμε το Pattern στο Constant για να κρατάει συνεχιόμενη την τιμή από τα 0 ns μέχρι το End Time στα 1000 ns.



Εικόνα 77 cbr Pattern

Για να έχουμε τιμή 1 καθ' όλη την διαρκείας θα χρειαστεί στο Value του Constant Attributes να βάλουμε 1.



Εικόνα 78 cbr Constant Attributes

Το σήμα που θα πάρουμε θα πρέπει να είναι μια ευθεία γραμμή

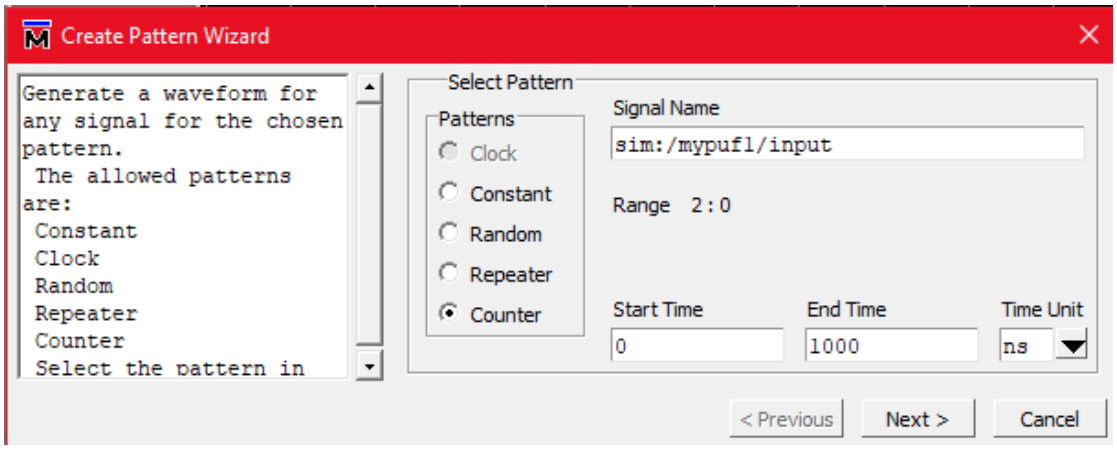
Εικόνα 79 cbr wave

11.1.2 Input

Με το Input επιλέγουμε ποια πράξει θέλουμε να κάνει ο controller οπότε είναι πολύ σημαντικό να ξέρουμε ποιες είναι η εντολές που πρέπει να δώσουμε και πως θα της περάσουμε στο modelsim.

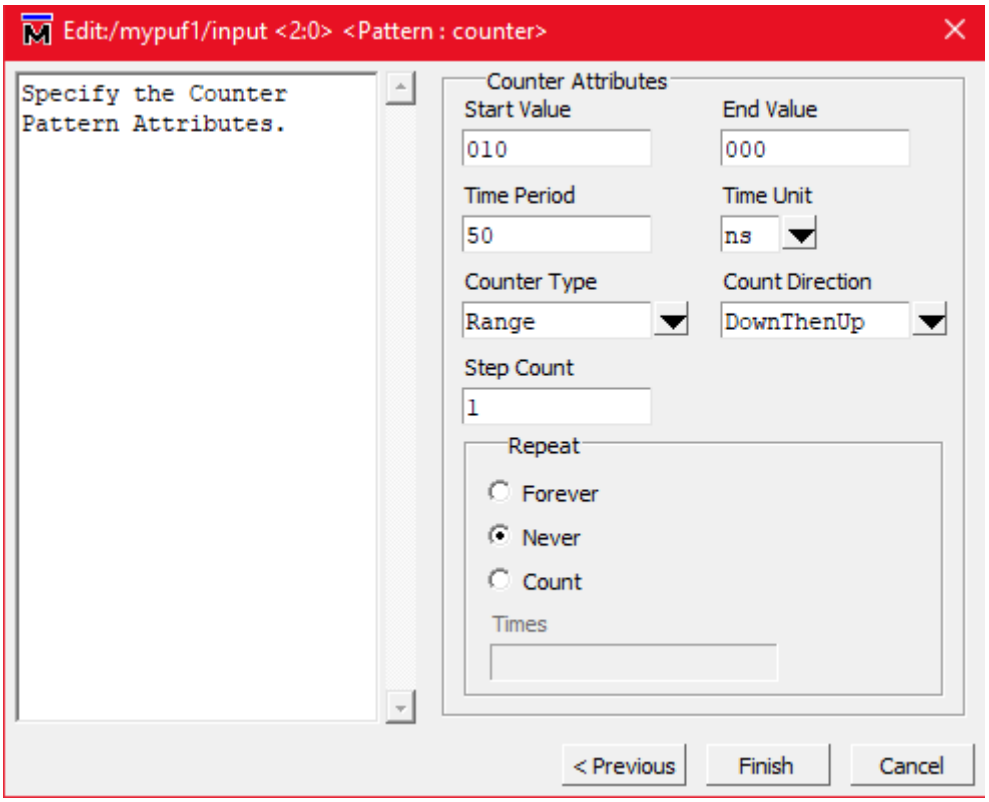
Το input δέχεται τρία δεκαδικά ψηφιά, με αυτά τα ψηφιά έχει την δυνατότητα να κάνει πέντε ενέργειες, push challenge, pop response, start counters, stop counters, και reset counter που αντιστοιχούν στο “000”, “001”, “010”, “011”,”100” οπότε πρέπει να βρούμε τρόπο να περάσουμε αυτές της τιμές στο modelsim.

Για αυτό στο Pattern θα επιλέξουμε Counter με Start Time 0 ns και End Time 1000ns.



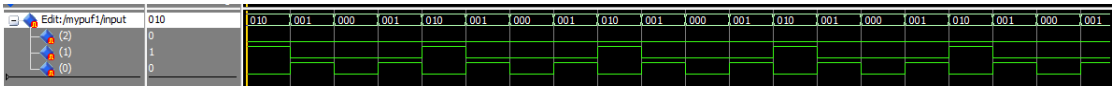
Εικόνα 80 input Pattern

Για να επιλέξουμε της κατάλληλες ενεργείς πρέπει να ξεκινήσουμε με “010” για να ξεκινήσει ο counter και μετά να πάει στης τιμές “000” και “001” για να κάνει push challenge και pop response.



Εικόνα 81 input counter

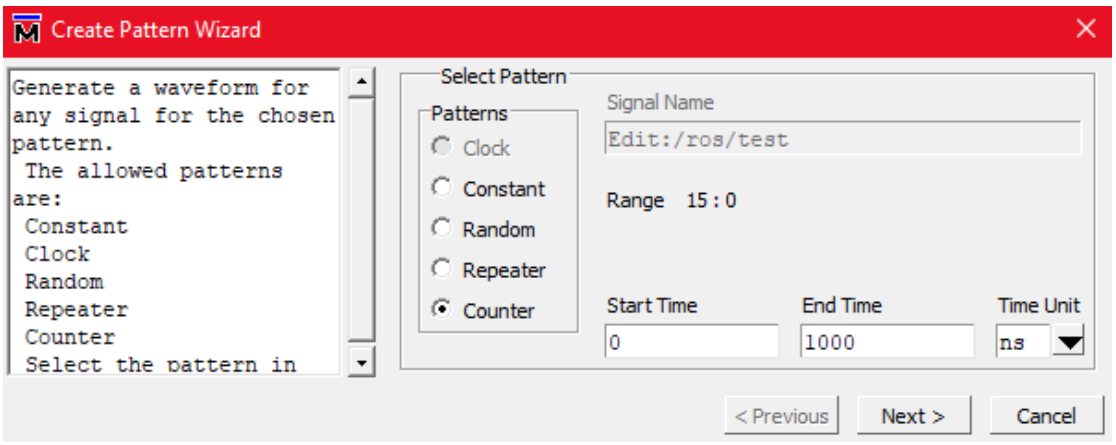
Το αποτέλεσμα πρέπει να είναι κάπως έτσι.



Εικόνα 82 input wave

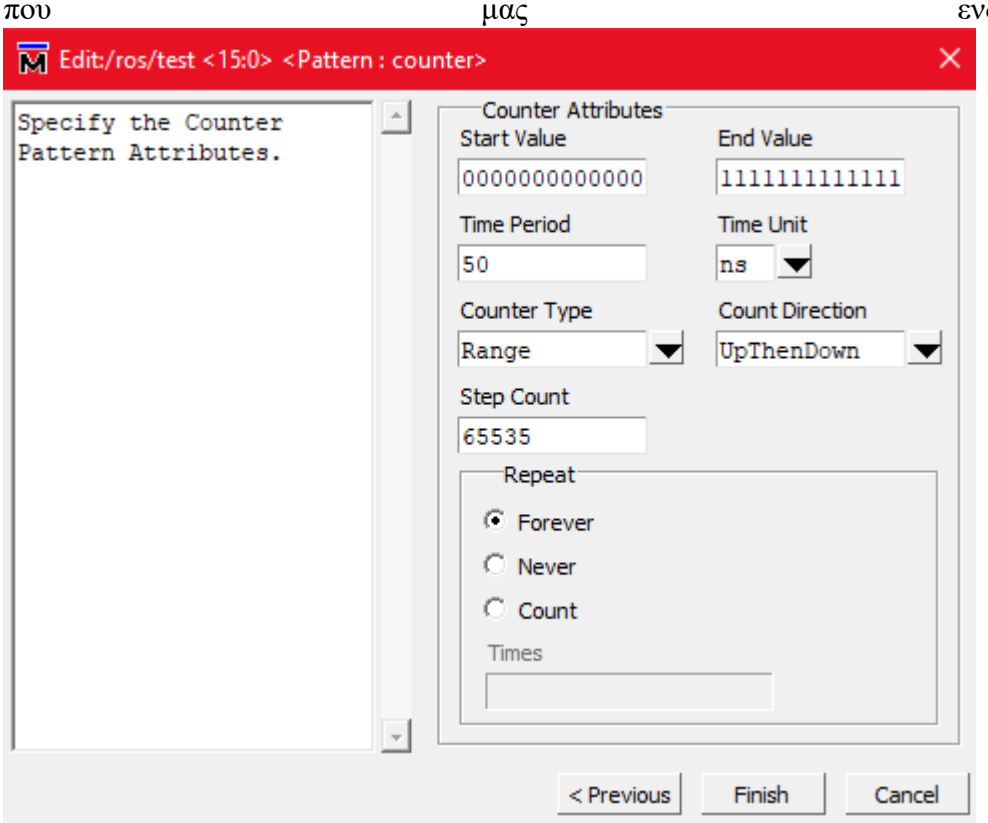
11.1.3 Rosin

Το rosin το χρησιμοποιούμε για να δώσουμε της τιμές στο ros για να εκκίνηση η παραγωγή τιμών από το κάθε RO στο ros, στο παράδειγμα μας χρησιμοποιούμε 16 ros οπότε θα χρειαστούμε 16 ψηφιά για το rosin. Για να δημιουργήσουμε αυτά τα ψηφιά στο modelsim αρκεί να βάλουμε για pattern, counter με Start Time 0 ns και End Time τα 1000 ns



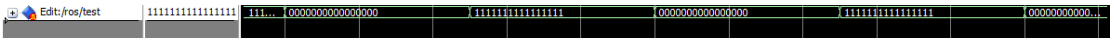
Εικόνα 83 rosin Pattern

Από το Pattern εμείς θέλουμε να περνούμε μόνο δυο τιμές, οι τιμές αυτές είναι “0000000000000000” δηλαδή όλα τα ro μας είναι κλειστά και ”1111111111111111” δηλαδή όλα τα ro είναι ανοιχτά. Αυτό στο Modelsim μπορούμε να το καταφέρουμε βάζοντας Start Value 0000000000000000 και End Value 1111111111111111, με Time Period 50 ns Counter Type Range και Count Direction UpThenDown για να μπορεί να πάει από την μηδενική λογική τιμή σε όλα τα ro στην λογική τιμή 1 και τον αντίθετο. Όμως εμείς θέλουμε να χρησιμοποιούνε μόνο αυτές οι δυο λογικές τιμές για να το πετύχουμε αυτό στο Modelsim θα χρειαστεί να βάλουμε στο Step Count 65535 δηλαδή όσο κάνει το 2^{16} έτσι θα αγνοούνται όλες οι υπέλειπες τιμές και θα περνούμε μόνο αυτές που μας ενδιαφέρουν.



Εικόνα 84 rosin Counter Attributes

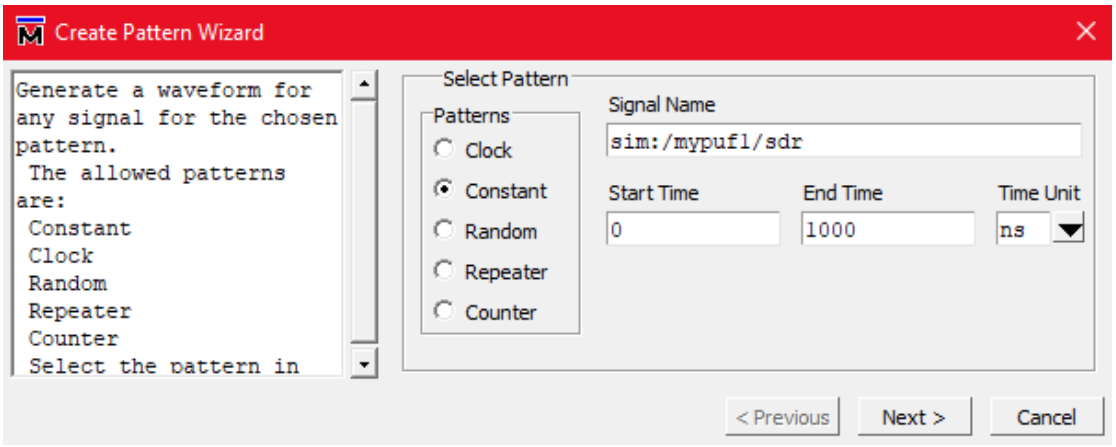
Μετά από αυτήν την διαδικασία το wave του σήματος πρέπει να είναι κάπως έτσι, αποτελούμενο μόνο από “0000000000000000” και “1111111111111111”



Εικόνα 85 rosin wave

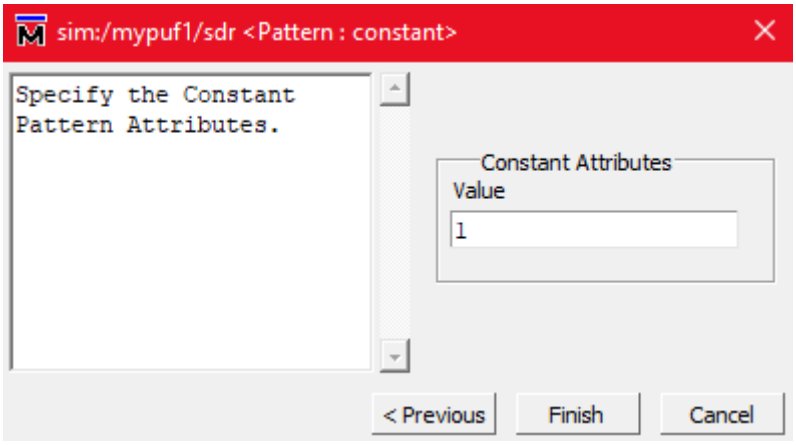
11.1.4 Sdr

Το sdr θα το έχουμε συνέχεια στην τιμή 1 για να μπορούμε να κάνουμε push challenge οπότε μας δίνετε η δυνατότητα. Θα βάλουμε το Pattern στο Constant για να κρατάει συνεχιόμενη την τιμή από τα 0 ns μέχρι το End Time στα 1000 ns.



Εικόνα 86 sdr Pattern

Για να έχουμε τιμή 1 καθ' όλη την διάρκεια θα χρειαστεί στο Value του Constant Attributes να βάλουμε 1.



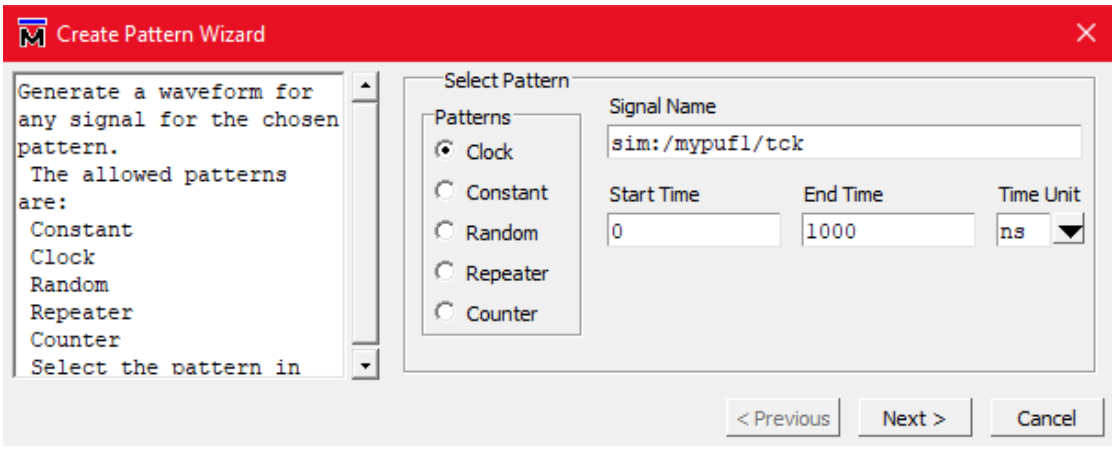
Εικόνα 87 sdr Constant Attributes

Το σήμα που θα πάρουμε θα πρέπει να είναι μια ευθεία γραμμή.

Εικόνα 88 sdr wave

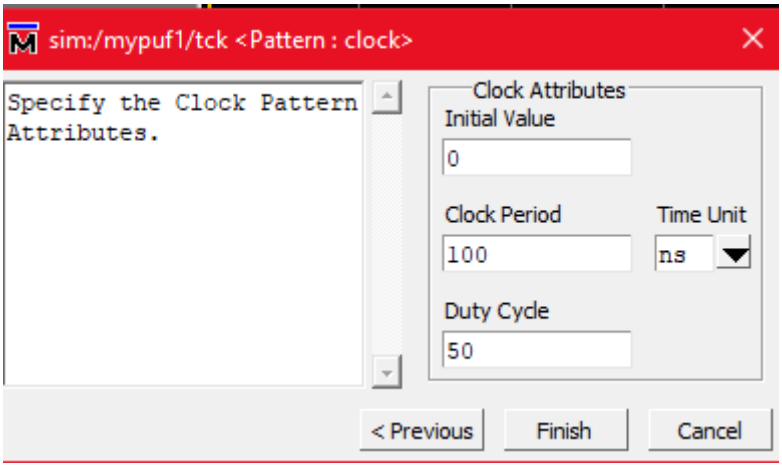
11.1.5 Tck

Στο tck που χρησιμοποιούμε για να ελέγξουμε της τιμές όλον το υπολιπόν inputs θα του δώσουμε Pattern clock με Start Time 0 ns και End Time 1000 ns.



Εικόνα 89 tck Pattern

Και θα επιλέξουμε για Attributes Initial Value 0 και Clock Period 100 ns με Duty Cyde 50 ns.



Εικόνα 90 tck clock Attributes

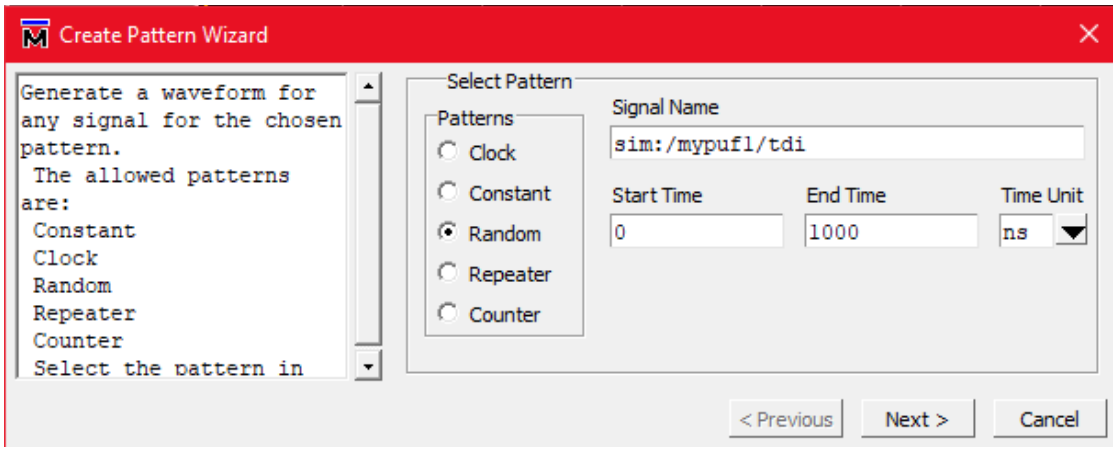
Από αυτήν την διαδικασία πρέπει να μας εμφανιστεί αυτό το Wave. 

Εικόνα 91 tck Wave

11.1.6 Tdi

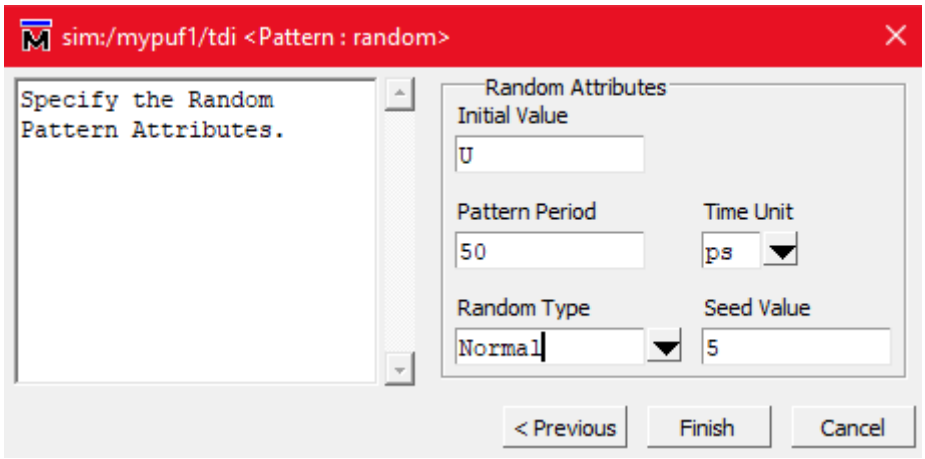
Το Tdi είναι από τα ποιο σημαντικά inputs στο project και αυτό που καθορίζει την τυχαιότητα της ΦΜΚΣ. Δηλαδή αυτό που δίνει την διαφορετικότητα αναμεσά στην συσκευές. Για αυτό όταν προσθέσουμε το wave του θα πρέπει να είμαστε προσεκτικοί και να επιλέξουμε της σωστές ρυθμίσεις. Οπότε όταν βάλουμε δημιουργήσουμε το Pattern για το tdi θα επιλέξουμε για Pattern Random, με αυτήν την επιλογή έχουμε την

δυνατότητα κάθε φορά να περνούμε διαφορετικές τιμες το οποίο το κάνει και μοναδικό. Για Start Time θα βάλουμε 0 ns και για End Time 100 ns.



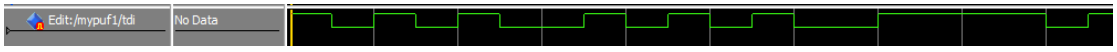
Εικόνα 92 tdi Pattern

Στο επόμενο παράθυρο θα επιλέξουμε την μορφή που θα έχει το κύμα μας, για Initial Value θα βάλουμε U, γιατί δεν θέλουμε να ξεκινήσουμε με κάποια συγκεκριμένη τιμή γιατί θέλουμε να είναι συνέχεια τυχαία τιμή, για Pattern Period βάζουμε 50 ps, Random Type θα επιλέξουμε Normal με Seed Value 5.



Εικόνα 93 tdi Random Attributes

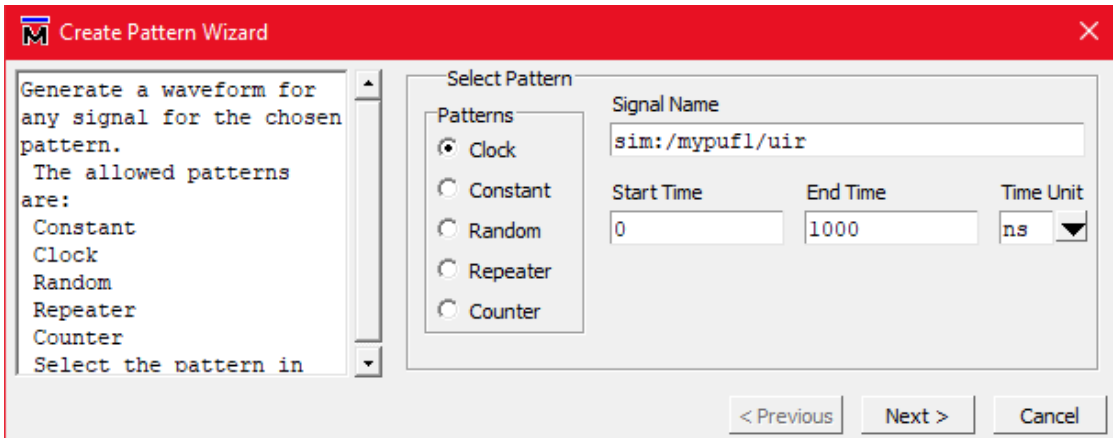
Το wave κάθε φορά θα είναι διαφορετικό οπότε δεν είναι αναγκαίο να έχει κάποια ομοιότητα το σήμα παρακάτω



Εικόνα 94 tdi wave

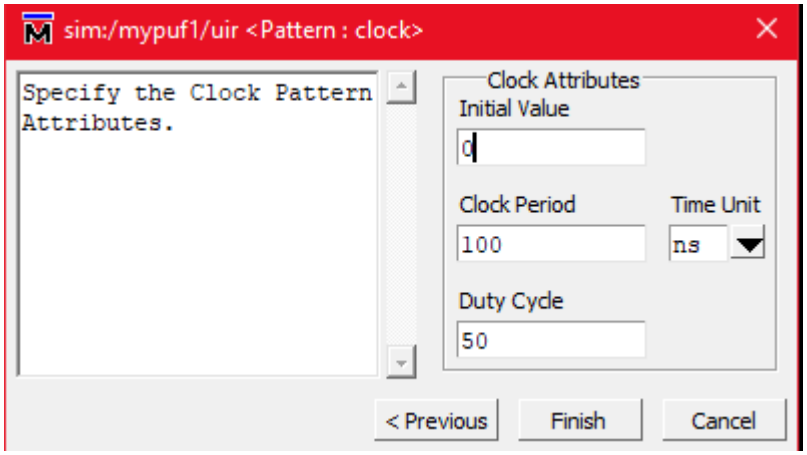
11.1.7 uir

Όπως και στο tck στο uir θα επιλέξουμε το pattern clock με start time στα 0 ns και το end time στα 1000 ns πατάμε next



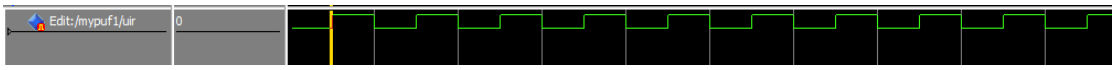
Εικόνα 95 uir pattern

Στο επόμενο παράθυρο θα βάλουμε Initial Value 0, το Clock Period στα 100 ns και το Duty Cyde στα 50 ns.



Εικόνα 96 uir Clock Attributes

Το σήμα μας τώρα πρέπει να έχει αυτήν την μορφή.



Εικόνα 97 uir wave