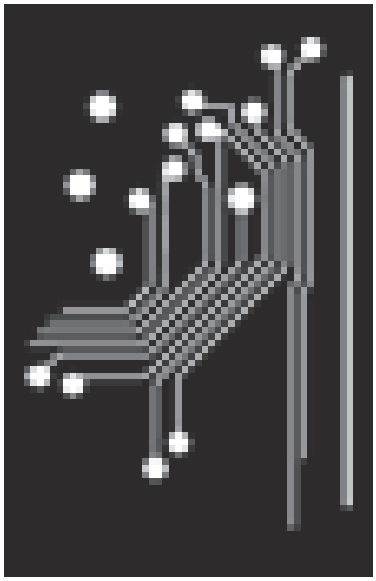
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Τ.Ε.Ι ΔΥΤΙΚΗΣ ΕΛΛΑΔΟΣ



ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΑΝΑΠΤΥΞΗ IP REPUTATION ΣΥΣΤΗΜΑΤΟΣ ΣΕ ΠΡΑΓΜΑΤΙΚΟ ΧΡΟΝΟ

ΜΠΟΥΝΤΑΣ ΠΑΝΑΓΙΩΤΗΣ

ΑΜ:0789

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ : ΝΙΚΟΛΑΟΣ ΒΩΡΟΣ

Πολλά ευχαριστώ στον **κ. Νικόλαο Βώρο** για την ευκαιρία που μου έδωσε να εκπονήσω την συγκεκριμένη εργασία καθώς και την αμέριστη συμπαράσταση και βοήθεια του, τόσο κατά την εκπόνηση της παρούσας πτυχιακής όσο και σε όλα τα χρόνια των σπουδών μου . Πολλά ευχαριστώ και στον **κ. Αποστόλη Φούρναρη** που ήταν δίπλα μου σε όλα τα στάδια αλλά και στα προβλήματα που προέκυψαν κατά την υλοποίηση του συστήματος.

Περίληψη

Στην παρούσα πτυχιακή εργασία σχεδιάστηκε και υλοποιήθηκε ένα σύστημα IP Reputation σε Hardware. Με το σύστημα αυτό, προσπαθήσαμε να λύσουμε το πρόβλημα της ασφάλειας στο διαδίκτυο των αντικειμένων. Βασική ιδέα του συστήματος είναι να αποτρέψουμε μια κακόφημη ip να εισέλθει στο δίκτυό μας.

Στο πρώτο κεφάλαιο αυτής της πτυχιακής εισάγουμε την έννοια της ασφάλειας, ορίζουμε τι είναι ασφάλεια και εξετάζουμε την ασφάλεια στο διαδίκτυο των αντικειμένων.

Στο επόμενο κεφάλαιο αναλύουμε το προτεινόμενο σύστημα εισάγοντας την έννοια του IP Reputation συστήματος και την έννοια του IP Reputation αρχείου. Τέλος βλέπουμε τα εργαλεία που είναι διαθέσιμα για την συλλογή και ανάλυση πληροφοριών δικτύου.

Στο τρίτο κεφάλαιο κάνουμε την ανάλυση και τον σχεδιασμό του συστήματός μας. Σε αυτό το κεφάλαιο θα δούμε τα διαγράμματα περιπτώσεων χρήσης αλλά και τα block διαγράμματα του συστήματός μας.

Στο τέταρτο κεφάλαιο θα δούμε και θα αναλύσουμε τα εργαλεία σχεδίασης που χρειαστήκαμε για την υλοποίηση του συστήματός μας. Θα δούμε την πλακέτα που χρησιμοποιήσαμε, την γλώσσα περιγραφής υλικού VHDL και τέλος το εργαλείο που χρησιμοποιήσαμε για να γράψουμε και να προσομοιώσουμε τον κώδικα.

Το πέμπτο κεφάλαιο περιλαμβάνει την υλοποίηση του συστήματός μας. Σε αυτό το κεφάλαιο θα δούμε βήμα-βήμα τον τρόπο που υλοποιήθηκε το σύστημά μας. Θα αναλύσουμε τον κώδικα που γράφτηκε και τέλος θα τον εξομοιώσουμε για την επαλήθευση της ορθής λειτουργίας του συστήματος.

Τέλος θα δούμε τα προβλήματα που δημιουργήθηκαν κατά την ανάπτυξη του συστήματος και πως τα επιλύσαμε.

Περιεχόμενα

1. ΕΙΣΑΓΩΓΗ

1.1 Το Πρόβλημα Της Ασφάλειας

Το **Διαδίκτυο** είναι παγκόσμιο σύστημα διασυνδεδεμένων δικτύων υπολογιστών, οι οποίοι χρησιμοποιούν καθιερωμένη ομάδα πρωτοκόλλων, η οποία συχνά αποκαλείται "TCP/IP" (αν και αυτή δεν χρησιμοποιείται από όλες τις υπηρεσίες του Διαδικτύου) για να εξυπηρετεί εκατομμύρια χρηστών καθημερινά σε ολόκληρο τον κόσμο. Οι διασυνδεδεμένοι ηλεκτρονικοί υπολογιστές ανά τον κόσμο, οι οποίοι βρίσκονται σε ένα κοινό δίκτυο επικοινωνίας, ανταλλάσσουν μηνύματα (πακέτα) με τη χρήση διαφόρων πρωτοκόλλων (τυποποιημένοι κανόνες επικοινωνίας), τα οποία υλοποιούνται σε επίπεδο υλικού και λογισμικού. Το κοινό αυτό δίκτυο καλείται Διαδίκτυο.

Το Διαδίκτυο αποτελεί παράδειγμα συστήματος τύπου open system interconnection. Καλείται ανοιχτό σύστημα (open system), γιατί σε αντίθεση με προηγούμενα επικοινωνιακά συστήματα ανεπτυγμένα από ιδιωτικές εταιρίες, η περιγραφή του είναι δημόσια διαθέσιμη. Έτσι, οποιοσδήποτε μπορεί να γράψει λογισμικό που να συμβαδίζει με τις προδιαγραφές του συστήματος. Σαν τέτοιο σύστημα, το Διαδίκτυο μπορεί να συγκριθεί με το μοντέλο OSI (Open System Interconnection).Η αρχιτεκτονική του Διαδικτύου έχει λιγότερα επίπεδα από αυτή του OSI και τα δεδομένα από το επίπεδο εφαρμογής (application) ως το επίπεδο φυσικής πρόσβασης (network access). Στο επίπεδο της φυσικής πρόσβασης (network access) ανήκουν τα πρωτόκολλα LAN όπως Ethernet, Token Ring, FDDI και πρωτόκολλα WAN όπως X.25, Frame Relay, SLIP, PPP που επιτρέπουν την φυσική διασύνδεση, την πρόσβαση στο μέσο και τον έλεγχο της ζεύξης. Στο επίπεδο δικτύου (network) χρησιμοποιείται το πρωτόκολλο IP, του οποίου τα πακέτα δρομολογούνται με ειδικές συσκευές, τους δρομολογητές (routers).Στο επίπεδο μεταφορά (transport) χρησιμοποιείται το πρωτόκολλο TCP και δευτερευόντως το UDP. Η πρόσβαση στο Διαδίκτυο σήμερα δεν είναι ακίνδυνη, ανεξάρτητα από τον τρόπο χρήσης των υπηρεσιών του. Υπάρχουν κακόβουλοι χρήστες και αρκετές δυνατότητες πρόκλησης ζημιών, τόσο στο επίπεδο του χρησιμοποιούμενου λογισμικού και υλικού, όσο και σε προσωπικό επίπεδο.

Ο κύριος κίνδυνος πρόκλησης ζημιών στο υπολογιστικό σύστημα ενός ανυποψίαστου χρήστη είναι η μόλυνση του συστήματος με κάποιον ιό. Η μόλυνση γίνεται όταν ο χρήστης καλείται να λάβει κάποιο -φαινομενικά αθώο- αρχείο όπως ένα κείμενο ή μια φωτογραφία και όταν δοκιμάσει να το χρησιμοποιήσει, ο ιός αναλαμβάνει δράση επιμολύνοντας το σύστημα. Μπορεί να καταστρέψει αρχεία ή και ολόκληρο το σκληρό δίσκο του συστήματος. Άλλες φορές είναι δυνατή η αποστολή ιού απευθείας από τον ιστότοπο που επισκέπτεται ο χρήστης, χωρίς να εμφανισθεί κάποια ένδειξη λήψης αρχείου. Η περίπτωση αυτή εκμεταλλεύεται κενά ασφαλείας στο λογισμικό του χρήστη (φυλομμετριτή ή Λειτουργικό σύστημα).

Παρόμοιας δράσης είναι και ένα πρόγραμμα που αποκαλείται worm (=σκουλήκι). Είναι παρόμοιο σε αποτέλεσμα με τον ιό, αλλά, αντίθετα από αυτόν, δεν απαιτεί την "προσκόλλησή" του σε ένα αρχείο, έχοντας έτσι περισσότερη αυτονομία. Η βλάβη που προκαλεί το worm δεν είναι τόσο ευρεία στο σύστημα, όσο στο δίκτυο σύνδεσης, επειδή καταναλώνει σημαντικό εύρος ζώνης (bandwidth).

Άλλος κίνδυνος είναι ο Δούρειος Ίππος (Trojan horse), ένα πρόγραμμα που ξεγελά το χρήστη του, ο οποίος χρησιμοποιώντας το νομίζει ότι εκτελεί κάποια εργασία, ενώ στην πραγματικότητα εκτελεί κάποια άλλη, συνήθως εγκατάσταση άλλων κακόβουλων προγραμμάτων. Αντίθετα από τους ιούς, οι δούρειοι ίπποι δεν επιμολύνουν αρχεία.

1.2 Τι Είναι Ασφάλεια

Τι είναι ένα ασφαλές δίκτυο; Μπορεί ένα διαδίκτυο να γίνει ασφαλές; Αν και η έννοια του ασφαλούς δικτύου (secure network) γοητεύει τους περισσότερους χρήστες, τα δίκτυα δεν μπορούν να ταξινομηθούν απλώς σε ασφαλή και μη ασφαλή, επειδή ο όρος αυτός δεν είναι απόλυτος –κάθε οργανισμός ορίζει ένα επίπεδο πρόσβασης που επιτρέπει ή απαγορεύει.

Για παράδειγμα μερικοί οργανισμοί αποθηκεύουν δεδομένα τα οποία είναι πολύτιμα. Οι οργανισμοί αυτοί ορίζουν ως ασφαλές δίκτυο ένα σύστημα πουεμποδίζει τους ξένους να αποκτούν πρόσβαση στους υπολογιστές του οργανισμού. Άλλοι οργανισμοί χρειάζεται να κάνουν τις πληροφορίες γνωστές στους ξένους, αλλά να μην τους επιτρέπουν να κάνουν αλλαγές στα δεδομένα. Οι οργανισμοί αυτοί μπορεί να ορίσουν ως ασφαλές ένα δίκτυο το οποίο επιτρέπει απεριόριστη πρόσβαση στα δεδομένα, αλλά περιλαμβάνει μηχανισμούς που εμποδίζουν τις αλλαγές από αναρμόδιους. Άλλες ομάδες πάλι, εστιάζουν την προσοχή τους στο να διατηρούν τις επικοινωνίες εμπιστευτικές. Αυτές ορίζουν ως ασφαλές ένα δίκτυο στο οποίο κανένας άλλος εκτός από τον τελικό αποδέκτη δεν μπορεί να υποκλέψει και να διαβάσει ένα μήνυμα.

Τέλος, πολλοί μεγάλοι οργανισμοί χρειάζονται έναν σύνθετο ορισμό της ασφάλειας, που να επιτρέπει την πρόσβαση σε κάποια επιλεγμένα δεδομένα

ή υπηρεσίες που έχει επιλέξει ο οργανισμός να κάνει δημόσια, ενώ εμποδίζει την πρόσβαση ή την τροποποίηση των ευαίσθητων δεδομένων και υπηρεσιών τα οποία διατηρούνται εμπιστευτικά.

Επειδή δεν υπάρχει απόλυτος ορισμός του ασφαλούς δικτύου, το πρώτο βήμα που πρέπει να κάνει ένας οργανισμός για να πετύχει ένα ασφαλές σύστημα είναι να ορίσει την πολιτική ασφαλείας (security policy) του. Η πολιτική αυτή δεν καθορίζει πως θα επιτευχθεί η προστασία. Καθορίζει όμως ρητά και με σαφήνεια τα στοιχεία που πρέπει να προστατεύονται. Ο ορισμός μιας πολιτικής ασφάλειας δικτύου είναι σύνθετη δουλειά.

Ειδικότερα, ο ορισμός μιας πολιτικής για τα δεδομένα που περνούν από ένα δίκτυο δεν εγγυάται ότι τα δεδομένα θα είναι ασφαλή. Για παράδειγμα, ας υποθέσουμε ότι κάποια αρχεία είναι αποθηκευμένα σε ένα αρχείο που επιτρέπεται η ανάγνωσή του. Η ασφάλεια του δικτύου δεν μπορεί να εμποδίσει τους αναρμόδιους χρήστες που έχουν λογαριασμό στον

υπολογιστή να πάρουν ένα αντίγραφο των δεδομένων. Επομένως για να είναι αποτελεσματική μια πολιτική ασφαλείας, πρέπει να ισχύει πάντοτε.

Η πολιτική πρέπει να ισχύει για τα δεδομένα που είναι αποθηκευμένα σε δίσκο, για τα δεδομένα που μεταφέρονται μέσω τηλεφωνικής γραμμής με ένα μόντεμ, για τις πληροφορίες που τυπώνονται σε χαρτί, για τα δεδομένα που μεταφέρονται σε φορητά μέσα, όπως μια δισκέτα, και για τα δεδομένα που μεταφέρονται μέσω ενός δικτύου υπολογιστών.

Η εκτίμηση του κόστους και της ωφέλειας των διαφόρων πολιτικών ασφαλείας κάνει επίσης το ζήτημα πιο σύνθετο. Συγκεκριμένα, μια πολιτική ασφαλείας δεν μπορεί να οριστεί παρά μόνο αν ο οργανισμός αντιληφθεί την αξία των πληροφοριών του. Σε πολλές περιπτώσεις η αξία των πληροφοριών είναι δύσκολο να εκτιμηθεί. Φανταστείτε λοιπόν, για παράδειγμα, μια απλή βάση δεδομένων μισθοδοσίας, η οποία περιέχει μια εγγραφή για κάθε υπάλληλο, τις ώρες που εργάστηκε ο υπάλληλος, και την αμοιβή του. Η ευκολότερη άποψη της αξιολόγησης είναι να εκτιμηθεί το κόστος αντικατάστασης. Δηλαδή μπορεί να υπολογιστούν οι ώρες εργασίας που χρειάζονται για να ξαναδημιουργηθεί ή να επαληθευθεί το περιεχόμενο της βάσης δεδομένων (π.χ. με την αποκατάσταση των δεδομένων από ένα εφεδρικό αντίγραφο ή με την πραγματοποίηση της εργασίας που χρειάζεται για να συλλεχθούν οι πληροφορίες). Μια δεύτερη άποψη της αξιολόγησης είναι να εκτιμηθεί το παθητικό που μπορεί να προκληθεί στον οργανισμό αν οι πληροφορίες είναι λανθασμένες. Για παράδειγμα αν ένα αναρμόδιο άτομο αυξήσει τις αμοιβές σε μια βάση δεδομένων μισθοδοσίας, η εταιρία θα μπορούσε να επιβαρυνθεί με οποιοδήποτε κόστος αν οι υπάλληλοι θα πληρώνονταν περισσότερο. Μια τρίτη άποψη της αξιολόγησης είναι το έμμεσο κόστος που θα μπορούσε να προκληθεί από παραβιάσεις της ασφάλειας. Για παράδειγμα, αν οι πληροφορίες μισθοδοσίας κοινοποιηθούν, μπορεί οι ανταγωνιστές να επιλέξουν να προσλάβουν κάποιους εργαζόμενους, με αποτέλεσμα ένα κόστος πρόσληψης και εκπαίδευσης αντικαταστατών, καθώς και αυξήσεις αμοιβών για να κρατήσει η εταιρία κάποιους υπαλλήλους. Μιλώντας γενικά, πρέπει να τονίσουμε ότι η επινόηση μιας πολιτικής ασφάλειας δικτύου μπορεί να είναι σύνθετη δουλειά, επειδή μια λογική πολιτική απαιτεί να εκτιμήσει ο οργανισμός την αξία των πληροφοριών. Η πολιτική πρέπει να εφαρμόζεται και για τις πληροφορίες που είναι αποθηκευμένες στους υπολογιστές και για τις πληροφορίες που περνούν από ένα δίκτυο.

1.3 Ασφάλεια Στο Διαδίκτυο

Όπως τονίστηκε η περιήγηση στο διαδίκτυο για τον απλό χρήστη, ενέχει πολλούς κινδύνους. Υπάρχουν πολλά ζητήματα που μπορεί να επηρεάσουν τόσο την ασφαλή περιήγηση του χρήστη στο διαδίκτυο, την ιδιωτικότητα του όσο και την επιτυχημένη και χωρίς απρόοπτα διενέργεια συναλλαγών.

Για την αύξηση της ευχρηστίας και της φιλικότητας, πολλές ιστοσελίδες στο διαδίκτυο χρησιμοποιούν scripts που εκτελούν προγράμματα μέσα στον περιηγητή δικτύου. Αυτό το ενεργό περιεχόμενο χρησιμοποιείται για να δημιουργήσει επιλογές όπως κυλιόμενα μενού. Δυστυχώς, όμως, αυτά τα script αποτελούν συχνά και ένα μέσο για τον επιτιθέμενο να ανεβάσει η να εκτελέσει επιβλαβές κώδικα στον υπολογιστή του χρήστη.

Μεγάλος κίνδυνος επίσης για τον χρήστη είναι η απώλεια της ιδιωτικότητας του. Οι ιστοσελίδες μπορούν να αποκτήσουν πληροφορίες για την ταυτότητα του χρήστη με μια ποικιλία τρόπων. Ο πιο βασικός είναι το αρχείο καταγραφών του εξυπηρετητή (server log), που καταγράφει την ώρα και ημερομηνία της σύνδεσης, την ταυτότητα του εγγράφου που ο χρήστης αιτήθηκε καθώς και το URL του προηγούμενου εγγράφου που ο χρήστης διάβαζε. Περισσότερη πληροφορία για τις κινήσεις του χρήστη υπάρχει στον παροχέα ίντερνετ του χρήστη όπου οι εξυπηρετητές proxy καταγράφουν κάθε website που ο χρήστης επισκέφθηκε.

Ένας ακόμα τρόπος συλλογής πληροφοριών για τους χρήστες από τις ιστοσελίδες είναι μέσω τον “cookies”, μικρών ταυτοτήτων που οι ιστοσελίδες χρησιμοποιούν για να σημαδεύουν περιηγητές δικτύου, δηλαδή να θυμούνται έναν συγκεκριμένο περιηγητή όταν επιστρέφει στην ιστοσελίδα μετά από καιρό. Τα “cookies” σχεδιάστηκαν για να εμπλουτίσουν την περιήγηση στο διαδίκτυο, επιτρέποντας στους χρήστες να μεταβάλουν τις ιστοσελίδες σύμφωνα με τις δικές τους ανάγκες, να προσπελαύνουν βάσεις δεδομένων και να εκτελούν και άλλες εργασίες που απαιτούν συνέχεια ανάμεσα στις περιόδους περιήγησης.

Η μεγαλύτερη απειλή όμως για την ιδιωτικοτητα του χρήστη δεν είναι οι πληροφορίες για αυτόν που εξάγονται με έμμεσες μεθόδους αλλά πληροφορίες που οι χρήστες δίνουν οικειοθελώς -σε έρευνες χρηστών, e-mail μηνύματα και on-line παραγγελίες. Μόλις αυτές οι πληροφορίες δοθούν από τον χρήστη δεν υπάρχει κάποια εγγύηση για το πως χρησιμοποιούνται και από ποιους.

Άμεσος κίνδυνος για την ασφάλεια των προσωπικών δεδομένων του χρήστη είναι και οι επιθέσεις phising. Το phising είναι μια τεχνική για την απόκτηση ευαίσθητων πληροφοριών όπως ονομάτων χρήστη, κωδικών πρόσβασης, και λεπτομερειών πιστωτικών καρτών, στην οποία ο επιτιθέμενος παριστάνει μια εμπιστεύσιμη οντότητα, όπως μια τράπεζα, έναν κοινωφελή οργανισμό, διαχειριστές δικτύου ή ακόμα και ιστοσελίδες κοινωνικής δικτύωσης. Το phising διεξάγεται κυρίως μέσω e-mail spoofing ή instant messaging, και συχνά απαιτεί από τους χρήστες να δώσουν πληροφορίες σε μια ιστοσελίδα που είναι όμοια με την αυθεντική ιστοσελίδα της οντότητας που ο επιτιθέμενος προσποιείται ότι είναι.

Το phising είναι μόνο μια από τις τεχνικές social engineering που χρησιμοποιούνται για να χειραγωγήσουν ανθρώπους στο να διαπράξουν ενέργειες ή να δώσουν προσωπικά τους δεδομένα και εκμεταλλεύονται την ελλιπή χρήση των τωρινών τεχνολογιών ασφαλείας δικτύου. Τρόποι αντιμετώπισης του αυξανόμενου αριθμού phising επιθέσεων είναι η εκπαίδευση του χρήστη, η ενημέρωση του κοινού και τεχνικά μέτρα ασφαλείας.

Για να αυξήσουν την ευχρηστία και την φιλικότητα της ιστοσελίδας οι περιηγητές δικτύου έχουν την ικανότητα να κατεβάσουν και να εκτελέσουν κώδικα αυτόματα, χωρίς προειδοποίηση. Ο κώδικας αυτός είναι κοινώς γνωστός ως “ενεργό περιεχόμενο”. Δυστυχώς όμως το ενεργό περιεχόμενο, που είτε κατασκευάστηκε για αυτό τον σκοπό είτε περιέχει σφάλματα που μπορούν να εκμεταλλευτούν από άλλους για επικίνδυνους σκοπούς, δίνει συχνά την δυνατότητα στον επιτιθέμενο να εκτελέσει επιβλαβές κώδικα στον

υπολογιστή του χρήστη. Παρακάτω παρουσιάζονται κάποιες κατηγορίες ενεργού περιεχομένου.

Βοηθητικές εφαρμογές και πρόσθετα Οι βοηθητικές εφαρμογές (helper applications ή και external viewers), είναι κανονικές εφαρμογές που μπορούν να τρέξουν και ανεξάρτητα του περιηγητή δικτύου. Δημιουργούν τα δικά τους παράθυρα, επεξεργάζονται τα δικά τους μενού εντολών και αλληλεπιδρούν απευθείας με εξωτερικές συσκευές.

Τα πρόσθετα, αντίθετα, είναι μικρές βιβλιοθήκες κώδικα που μπορούν να εκτελεστούν μόνο στο περιθώριο μιας συνεδρίας του περιηγητή δικτύου. Ο τελευταίος μάλιστα εκτελεί τον κύριο όγκο της επεξεργασίας που απαιτείται από το πρόσθετο για να δουλέψει. Τα πρόσθετα είναι κατά κύριο λόγο γραμμένα από τρίτους κατασκευαστές για να καταστήσουν ικανό τον περιηγητή να μεταφράσει ή να αναπαράγει συγκεκριμένους τύπους δεδομένων ή ακόμα και για να επεκτείνουν τις δυνατότητες του.

Η εγκατάστασή των προσθέτων γίνεται από τον χρήστη, κατεβάζοντας τον κώδικα τους και εκτελώντας το πρόγραμμα εγκατάστασης. Κατά τη διάρκεια της, η βιβλιοθήκη του προσθέτου αντιγράφεται σε ένα καθορισμένο κατάλογο αρχείων (directory) και ο περιηγητής ειδοποιείται για την ύπαρξη του.

Παρά τις διαφορές τους, και οι βοηθητικές εφαρμογές και τα πρόσθετα είναι όμοια από την σκοπιά της ασφάλειας. Είναι κομμάτια κώδικα που έχουν πλήρη πρόσβαση στον υπολογιστή του χρήστη.

Ακόμα και αν είναι κατασκευασμένα με τις καλύτερες των προθέσεων δεν υπάρχει κάποια εγγύηση ότι είναι ασφαλή. Τα περισσότερα από τα προβλήματα που παρουσιάζουν είναι μικρές ενοχλήσεις αλλά άλλα μπορεί να είναι πιο απειλητικά. Αμφότερα μπορούν να είναι μολυσμένα με ιούς, όπως και οποιοδήποτε άλλο πρόγραμμα.

Ακόμα κάτι που οι χρήστες πρέπει να προσέχουν είναι βοηθητικές εφαρμογές που περιέχουν διερμηνευτές εντολών. Ακόμα και αν η εφαρμογή είναι ελεύθερη από σφάλματα υπάρχει η πιθανότητα ότι κάποιος θα δημιουργήσει ένα σύνολο εντολών, το οποίο όταν διερμηνευτεί από το πρόγραμμα θα το ωθήσει σε λάθος συμπεριφορά. Ως ένας γενικός κανόνας, οτιδήποτε περιέχει επεξεργαστή μακροεντολών είναι πιθανά επικίνδυνο. Οποιοδήποτε πρόγραμμα μπορεί να εκκινήσει εξωτερικές εφαρμογές, ενέχει επίσης κινδύνους. Ακόμα όμως και μια απλή εφαρμογή μπορεί να μην είναι εντελώς ασφαλής.

Ευτυχώς και οι βοηθητικές εφαρμογές και τα πρόσθετα απαιτούν ενέργειες από τον χρήστη για να εγκατασταθούν. Αυτή η διαδικασία καθιστά σαφές στον χρήστη ότι τα προγράμματα αυτά είναι κατασκευασμένα από τρίτους. Οι χρήστες πρέπει να σκεφτούν καλά πρώτου εγκαταστήσουν ένα τέτοιο πρόγραμμα. Η καλύτερη συμβουλή που μπορεί να δοθεί είναι οι χρήστες να κρατούν τον αριθμό των εγκαταστημένων βοηθητικών εφαρμογών και προσθέτων στο ελάχιστο.

1.4 Διαδίκτυο Των Πραγμάτων (IoT)

Το Internet Of Things (IoT) ή «Διαδίκτυο των Πραγμάτων» είναι η επερχόμενη εξέλιξη του Διαδικτύου των υπηρεσιών, που υπάρχει σήμερα. Πρόκειται για

ένα δίκτυο όχι μόνο υπολογιστών αλλά και διασυνδεδεμένων αντικειμένων. Τα αντικείμενα αυτά θα περιέχουν ενσωματωμένα ηλεκτρονικά συστήματα και μπορούν να είναι διάφορες οικιακές συσκευές, μέσα μεταφοράς,μέσα τηλεπικοινωνίας, βιβλία, αυτοκίνητα, ακόμα και τρόφιμα. Πέρα από την εξασφάλιση της καλής λειτουργίας των διασυνδεδεμένων αυτών αντικειμένων, θα γίνει προσπάθεια να επιτευχθεί και συνεργασία μεταξύ των συστημάτων αυτών. Κάθε αντικείμενο θα χρησιμοποιεί συστήματα αναγνώρισης ραδιοσυχνοτήτων (τα γνωστά ως RFID), ένα είδος αισθητήρων, δηλαδή. Απαραίτητη προϋπόθεση για την επιτυχία του καινούριο αυτού Διαδικτύου είναι να καταστεί το σημερινό Διαδίκτυο πιο ασφαλές.

Το Διαδίκτυο των Πραγμάτων θα είναι η κορύφωση της προσπάθειας για την ολοκλήρωση και αυτοματοποίηση των υπηρεσιών που παρέχουν τα ενσωματωμένα συστήματα παντός είδους. Το διαδίκτυο θα γίνει διαδραστικό, ένα τεράστιο ιεραρχικά οργανωμένο «νευρικό σύστημα» που θα απολήγει σε συσκευές με αισθητήρες και ενεργοποιητές (actuators) που θα συνεργάζονται για έξυπνες υπηρεσίες για την υγεία, τις μεταφορές, τη διανομή και κατανάλωση ενέργειας κλπ. Στις μεταφορές σύντομα θα έχουμε συστήματα αυτόματης οδήγησης και οργάνωσης των μεταφορικών μέσων για περισσότερη ασφάλεια και οικονομία. Στον τομέα της υγείας προβλέπονται μία σειρά από καινοτομίες, από τη διαδραστική τηλεπαρακολούθηση των ασθενών, μέχρι την τηλεχειρουργική και τα έξυπνα φάρμακα. Η μεγάλη πρόκληση είναι η αυτοματοποίηση της διαχείρισης πόρων όπως για παράδειγμα σε αυτό που ονομάζεται smart grids, συνδυασμένη και αποτελεσματική χρήση εναλλακτικών μορφών ενέργειας.

Όλες αυτές θα είναι μερικές εφαρμογές που θα αλλάξουν ριζικά το σημερινό τρόπο ζωής τις ερχόμενες δεκαετίες. Ωστόσο ακόμα η κατάσταση του διαδικτύου παραμένει ιδιαίτερα επισφαλής και χρειάζεται μεγάλη δράση και κινητοποίηση για να μπορέσουν όλα αυτά να λειτουργήσουν με ασφάλεια και αποτελεσματικότητα.

Ο συνδυασμός του internet, των αντικειμένων και κινητών υπηρεσιών, ανοίγει το δρόμο σε αυτό που λέμε «διάχυτη νοημοσύνη». Πανταχού παρούσα και απρόσκοπτη πρόσβαση σε παντοειδείς υπηρεσίες, αποτελεσματικός έλεγχος πόρων, διαδραστικότητα και συνέργεια για την επίτευξη ολοκληρωμένων στόχων. Μεγάλες εταιρίες δεν αποσκοπούν πλέον στη μεμονωμένη πώληση λογισμικού ή υπολογιστών αλλά μελετούν ολοκληρωμένες λύσεις όπου τα πληροφορικά συστήματα χρησιμοποιούνται για τη βέλτιστη διαχείριση φυσικών πόρων και την ανάπτυξη έξυπνων υπηρεσιών για τη δημιουργία ενός «έξυπνου πλανήτη».

1.4.1 Ασφάλεια Σε IoT

Το διαδίκτυο των αντικειμένων είναι ένα πολύ τρωτό σύστημα σε επιθέσεις για αρκετούς λόγους. Πρώτον τα αντικείμενα των χρηστών πολλές φορές αφήνονται απαρατήρητα, πράγμα που διευκολύνει την φυσική επίθεση εναντίον τους. Δεύτερον επειδή όλες σχεδόν οι συνδέσεις και η επικοινωνία των αντικειμένων γίνονται με ασύρματο τρόπο, γίνεται εύκολη η παρακολούθηση των δεδομένων που αποστέλλονται. Τρίτον, εξαιτίας των

χαμηλών δυνατοτήτων των αντικειμένων, σε ισχύ αλλά και σε υπολογιστικούς πόρους, το οποίο συμβαίνει ιδιαιτέρως στις παθητικές ετικέτες RFID, δεν γίνεται δυνατή η υλοποίηση πολύπλοκων αλγορίθμων ασφαλείας. Η ασφάλεια μπορεί να χωριστεί σε δύο κατηγορίες, αυτής που αφορά στην ιδιωτικότητα (privacy) του χρήστη, δηλαδή στην ακεραιότητα των δεδομένων που αποστέλλονται ή λαμβάνονται και αυτή που αφορά στη ταυτοποίηση (identification) για τη ανάγνωση των πληροφοριών του χρήστη.

1.4.1.1 Ιδιωτικότητα

Η ιδιωτικότητα είναι μία από τα βασικά και νομικά θεμελιωμένα δικαιώματα του κάθε ανθρώπου. Εξαιτίας αυτού του δικαιώματος οποιαδήποτε μη ασφαλής παροχή υπηρεσιών μπορεί να οδηγήσει σε εμπόδια στην ανάπτυξη του διαδικτύου των αντικειμένων. Ένας τρόπος παραβίασης της ιδιωτικότητας είναι η δυνατότητα της ετικέτας να διαβαστεί από απόσταση χωρίς την γνώση του χρήστη, με άμεση συνέπεια τη παρακολούθηση του χρήστη από απόσταση. Επιπλέον είναι εφικτή η καταγραφή του χρήστη αφού μπορούν τα αντικείμενα που αγοράζει ο χρήστης να καταγράφονται σε βάσεις δεδομένων κάτω από το όνομά του. Ένα άλλο πρόβλημα είναι η παραμονή των ετικετών και των πληροφοριών που διαθέτουν στις συσκευές ανάγνωσης και μετά το πέρας κάποιας διαδικασίας, φαινόμενο το οποίο μπορεί να οδηγήσει σε διαρροή των προσωπικών πληροφοριών του χρήστη.

Για να επιλυθεί το πρόβλημα της συλλογής δεδομένων χρειάζεται να βρεθούν λύσεις στα επιμέρους υποσυστήματα που απαρτίζουν το διαδίκτυο των αντικειμένων. Θα μπορούσε να ακολουθηθεί το μοντέλο που παραδοσιακά χρησιμοποιείται στο διαδίκτυο όπου παρέχεται στο χρήστη η επιλογή των προσωπικών του στοιχείων που χρησιμοποιηθούν σα παράμετροι για την χρήση μιας εφαρμογής. Επιπλέον για την διασφάλιση των δεδομένων που αποστέλλονται ή λαμβάνονται θα μπορούσαν να χρησιμοποιηθούν αλγόριθμοι ταυτοποίησης όπως περιγράφονται παρακάτω. Βέβαια όλες αυτές οι λύσεις προϋποθέτουν την χρήση των πόρων των αντικειμένων που τυχαίνει να είναι περιορισμένοι.

Τέλος για να μην αφήνεται έρμαιο στα χέρια των συσκευών ανάγνωσης και σε αυτούς που τις διαχειρίζονται, ο χρήστης θα πρέπει να μπορεί να προστατεύει τα προσωπικά του δεδομένα, εξασφαλίζοντας ποια δεδομένα του θα ανταλλάσσονται σε περίπτωση ερώτησης από κάποια συσκευής ανάγνωσης, ποιος θα συλλέγει τα δεδομένα αυτά, όπως επίσης και το χρόνο της συλλογής τους. Επιπλέον θα πρέπει να γνωρίζει το χρόνο παραμονής των δεδομένων του στις συσκευές ανάγνωσης, ο οποίος δε πρέπει να υπερβαίνει τον απαιτούμενο.

1.4.1.2 Κρυπτογραφία

Γενικά όλες οι λύσεις που έχουν προταθεί μέχρι τώρα για την στήριξη της ασφάλειας χρησιμοποιούν κάποια μεθοδολογία κρυπτογραφίας. Τυπικά οι αλγόριθμοι κρυπτογράφησης χρειάζονται ένα μεγάλο μέρος από τους πόρους του συστήματος, δηλαδή σε ισχύ και εύρος ζώνης, και στη πηγή αλλά και στο προορισμό. Τέτοιου είδους λύσεις όμως δεν μπορούν να εφαρμοστούν στο

διαδίκτυο των αντικειμένων με δεδομένο τις περιορισμένες δυνατότητες σε ισχύ και υπολογιστική ικανότητα των ετικετών ή των αισθητήρων. Συνεπώς πρέπει να βρεθούν λύσεις ασφαλείας που να παρέχουν την επιθυμητή ποιότητα ασφαλείας με την ελάχιστη χρήση πόρων. Ένας τρόπος κρυπτογραφίας είναι η χρήση ελαφριού τύπου συμμετρικών κλειδιών. Όμως η διαχείριση των κλειδιών αυτών είναι σε πρώιμο στάδιο και υπό έρευνα.

Μία έξυπνη λύση που έχει προταθεί είναι η επανετικετοποίηση (Relabeling). Η ιδέα αυτή προτείνει την εξάλειψη των μοναδικών αναγνωριστικών και την αντικατάστασή τους με ένα μηχανισμό δημιουργίας καινούριων ετικετών, αλλά και την αποθήκευση των ήδη χρησιμοποιούμενων ώστε να είναι δυνατή η μελλοντική τους χρήση. Μπορεί επιπλέον να είναι δυνατή και η επιβεβαίωση της αλλαγής των ετικετών από το χρήστη για να ελαχιστοποιηθεί η εκπομπή δεδομένων. Όπως είναι φυσικό η εξάλειψη των μοναδικών ετικετών δεν επιλύει το πρόβλημα της μη εξουσιοδοτημένης ανάγνωσης του αντικειμένου, ούτε και την παρακολούθηση του, αφού ακόμα και εάν εκπέμπουν οι ετικέτες μόνο επιλεγμένες πληροφορίες σχετικά με το αντικείμενο, είναι και πάλι μοναδικά ανιχνεύσιμες.

Ενώ όμως οι συσκευές με δυνατότητες υψηλής ισχύς όπως οι συσκευές ανάγνωσης μπορούν να επανετικετοποιούν, οι ετικέτες συνήθως παθητικές ετικέτες RFID δε μπορούν. Για αυτό το λόγο προτείνεται η χρήση ενός μινιμαλιστικού τρόπου κρυπτογράφησης, όπου κάθε ετικέτα μπορεί να περιέχει μία συλλογή από ψευδώνυμα. Με περιστροφή των ψευδωνύμων δίνεται σε κάθε αναζήτηση από τη συσκευή ανάγνωσης και διαφορετικό. Με τη χρήση του συγκεκριμένου τρόπου κρυπτογράφησης μία εξουσιοδοτημένη συσκευή ανάγνωσης μπορεί να αποθηκεύσει τη λίστα με τα ψευδώνυμα του αντικειμένου, σε αντίθεση με μία μη εξουσιοδοτημένη, η οποία δεν είναι σε θέση να συνεργαστεί με το αντικείμενο με τις εμφανίσεις διαφορετικών ψευδωνύμων. Επιπλέον στη περίπτωση ταχέων ερωτήσεων από τη συσκευή ανάγνωσης ώστε να υποκλέψει τη λίστα με τα ψευδώνυμα, το αντικείμενο μπορεί να επιβραδύνει το χρόνο απάντησης στη συσκευή ανάγνωσης.

1.4.1.3 Πλεονεκτήματα Του IoT

Το πλεονέκτημα χρήσης του Διαδικτύου των Αντικειμένων (IOT) είναι ότι προσφέρει δυνατότητες οι οποίες κάνουν εφικτή την ανάπτυξη ενός μεγάλου πλήθους εφαρμογών που προς το παρόν μόνο ένα μικρό κομμάτι από αυτές προσφέρονται στη κοινωνία. Οι τομείς που μπορεί να υλοποιηθεί η καινούρια αυτή τεχνολογία βελτιώνοντας με αυτό τον τρόπο τη ποιότητα ζωής είναι το οικιακό περιβάλλον, το εργασιακό περιβάλλον, τα ταξίδια, η γυμναστική και πολλοί άλλοι.

Τα περιβάλλοντα αυτά είναι εξοπλισμένα με αντικείμενα τα οποία έχουν πρωτόγονη νοημοσύνη και χωρίς καμία ικανότητα επικοινωνίας. Με τη παροχή της δυνατότητας επικοινωνίας των αντικειμένων μεταξύ τους και τη δυνατότητα επεξεργασίας των λαμβανομένων, από το περιβάλλον πληροφοριών, γίνεται εφικτή η κάθε φορά από το αντικείμενο αντίληψη του περιβάλλοντος στο οποίο βρίσκεται και ποιες από τις εφαρμογές μπορούν να υλοποιηθούν στο περιβάλλον αυτό. Ο διαχωρισμός των εφαρμογών μπορεί να χωριστεί ανάλογα με το τομέα που αναφέρονται ως εξής:

- Τομέας Μεταφορών και Χειρισμού Υλικών (logistics)
- Τομέας Υγείας.
- Τομέας Έξυπνου Περιβάλλοντος (σπίτι, γραφείο, κτηριακές εγκαταστάσεις)
- Τομέας Προσωπικός και Κοινωνικός

1.5 Στόχος της Πτυχιακής

Βλέποντας τα προβλήματα ασφάλειας που προκύπτουν στο διαδίκτυο των πραγμάτων, στα πλαίσια αυτής της πτυχιακής, προτείνουμε τον σχεδιασμό ενός συστήματος στο οποίο μπορεί να πραγματοποιηθεί ανάλυση δεδομένων σε πραγματικό χρόνο.

2.ΠΡΟΤΕΙΝΟΜΕΝΟ ΣΥΣΤΗΜΑ

2.1 Τι είναι το IP Reputation

IP Reputation είναι μια σύνοψη της συμπεριφοράς που είχε μια IP στο παρελθόν. Η γνώση της 'φήμης' μιας IP επιτρέπει στους οργανισμούς να αναγνωρίζουν και να μπλοκάρουν γνωστές «κακόθυμες IP» στην περίμετρο του δικτύου τους. Αναλύοντας τα δεδομένα από παγκόσμιες βάσεις δεδομένων «IP Reputation », οι οργανισμοί μπορούν ενεργά να εκτελούν και να διαχειρίζονται τις πολιτικές ασφάλειάς τους.

Χρησιμοποιώντας μια συσκευή που βασίζεται στο δίκτυο που ελέγχει την κυκλοφορία σε πραγματικό χρόνο και επιβάλλει συγκεκριμένες πολιτικές ασφαλείας, οι οργανισμοί θα είναι σε θέση να :

- Μπλοκάρουν downloads Trojans,και την εξάπλοσή τους στο δίκτυό τους
- Μπλοκάρουν Malwares, Spywares και worms
- Μπλοκάρουν Spam από γνωστά spamming sites
- Μπλοκάρουν phishing emails από γνωστά phishing sites
- Μπλοκάρουν την πρόσβαση σε γνωστά botnet και Control sites
- Μπλοκάρουν την πρόσβαση σε γνωστά phishing sites
- Μπλοκάρουν DDos και Web application επιθέσεις από γνωστά botnet hosts.
- Μπορούν να δημιουργούν πολιτικές ασφάλειας βασισμένες στο IP Reputation
- Έχουν περιορισμούς ή ειδοποιήσεις για εισερχόμενες συνδέσεις δικτύου με βάση την χώρα καταγωγής
- Έχουν περιορισμούς ή ειδοποιήσεις για εξερχόμενες συνδέσεις δικτύου με βάση την χώρα καταγωγής

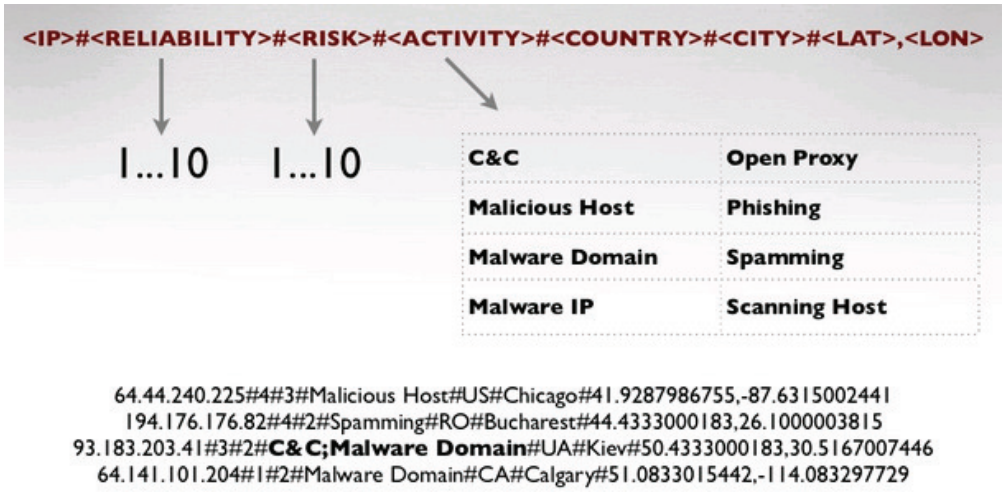
2.2 IP Reputation Engine

Το IP Reputation engine είναι ένα σύστημα για να ταξινομεί μεγάλα σύνολα από IPs, σε υψηλή ή χαμηλή «φήμη». Το IP Reputation engine συλλέγει πληροφορίες από διάφορες πηγές όπως

- Malware trackers
- Malicious Hosts λίστες
- Open proxy λίστες
- Scanning hosts λίστες
- Spam tracks και πολλά άλλα

2.3 IP Reputation Data File

Ένα αρχείο δεδομένων που περιλαμβάνει IPs με «φήμη» παρέχετε για τις ανάγκες της ανάπτυξης ενός IP Reputation engine. Το αρχείο δεδομένων έχει την δομή της Εικόνας 2.1. Το Reliability αναφέρεται για να προσδιορίσει την αξιοπιστία που έχει μία ip, το risk δείχνει το ρίσκο που έχει μια ip να είναι μολυσμένη, και το activity προσδιορίζει την δραστηριότητα που είχε αυτή η ip, στο παρελθόν.



Εικόνα 2.1 Δομή IP Reputation Data File

Το **Reliability** αναφέρεται για να προσδιορίσει την αξιοπιστία που έχει μία ip, το **risk** δείχνει το ρίσκο που έχει μια ip να είναι μολυσμένη, και το **activity** προσδιορίζει την δραστηριότητα που είχε αυτή η ip, στο παρελθόν.

2.4 Συλλογή Πληροφοριών Δικτύου

Η συλλογή πληροφοριών δικτύου μπορεί να γίνει μέσω των αναλυτών πακέτων. Ένας αναλυτής πακέτων είναι ένα πρόγραμμα ή υλικό υπολογιστή, το οποίο μπορεί να “υποκλέπτει” και να καταγράφει διερχόμενα από ένα δίκτυο-ή κομμάτι του- δεδομένα.

Η ανάλυση των κυκλοφοριακών ροών μέσω του IP Reputation Engine μπορεί να βασίζεται σε διάφορα εργαλεία ανοικτού κώδικα διαθέσιμα συμπεριλαμβανομένων:

- **Whireshark :** Το Wireshark αποτελεί ένα από τα διασημότερα προγράμματα παγκοσμίως για την ανάλυση των δικτύων. Αυτό το πολύ δυνατό εργαλείο παρέχει πληροφορίες για το δίκτυο σας και

των πρωτοκόλλων ανώτερου επιπέδου σχετικά με τα δεδομένα που διακινούνται σ' αυτό. Όπως πολλά άλλα δικτυακά προγράμματα, το Wireshark χρησιμοποιεί τη δικτυακή βιβλιοθήκη pcap για την σύλληψη (ανάλυση) των πακέτων. Η δύναμη του Wireshark πηγάζει από: - την ευκολία εγκατάστασής του. - την απλότητα της χρήσης του μέσω της γραφικής διεπαφής του (GUI). - το μεγάλο αριθμό της λειτουργικότητας του. Το Wireshark ονομαζόταν Ethereal μέχρι το 2006, όταν ο επικεφαλής προγραμματιστής, αποφάσισε την αλλαγή του ονόματός του λόγω δικαιωμάτων χρήσης που προϋπήρχαν για το όνομα Ethereal, το οποίο ήταν κατοχυρωμένο από την εταιρεία από την οποία αποφάσισε να αποχωρήσει το 2006.

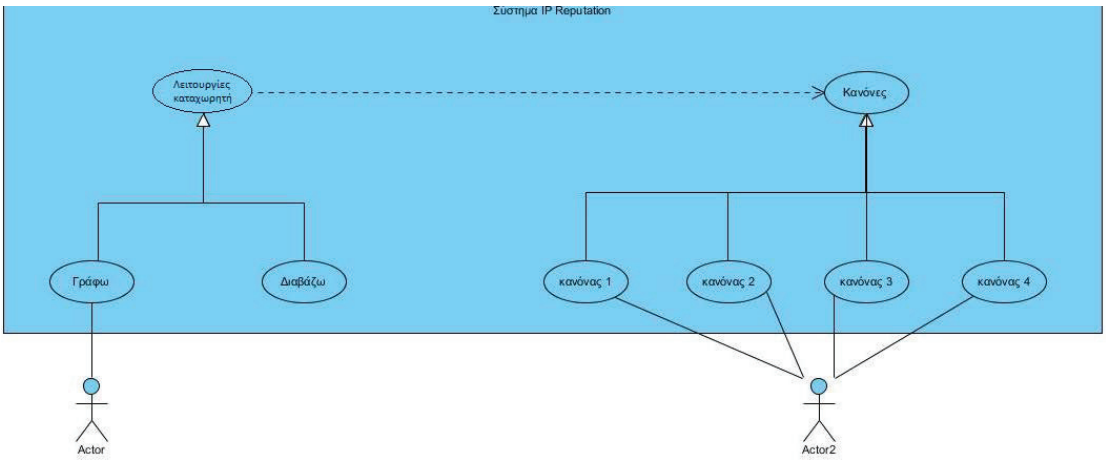
- **TcpDump :** Αποτελεί ένα εργαλείο ανάλυσης πακέτων δικτύου που εκτελείται στη γραμμή εντολών. Επιτρέπει στο χρήστη να παρακολουθήσει τα πακέτα που μεταδίδονται ή λαμβάνονται μέσω ενός δικτύου στο οποίο είναι συνδεδεμένος ο υπολογιστής. Η χρήση του προσφέρει σε κάποιον τη δυνατότητα να κατανοήσει το πρωτόκολλο TCP/IP.
- **Snort :** Το Snort είναι ένα ΄lightweight΄ Network Intrusion Detection System (NIDS), καθώς και ένα εργαλείο ανάλυσης πακέτων το οποίο στηρίζεται στην βιβλιοθήκη libpcap. Ο δημιουργός του Snort είναι ο Martin Roesch ο οποίος ξεκίνησε την ανάπτυξη του κώδικά σε γλώσσα προγραμματισμού C, ενώ σήμερα ένας μεγάλος αριθμός ατόμων έχει εμπλακεί σε αυτήν την διαδικασία, με στόχο την προσθήκη νέων λειτουργιών στο Snort και την βελτίωση των δυνατοτήτων του. Σημαντικό ρόλο για το γεγονός αυτό παίζει ότι το Snort είναι ένα Open Source λογισμικό, το οποίο διατίθεται κάτω από την GNU General Public License (GPL) που καθιστά ελεύθερη την χρήση και ανάπτυξη του κώδικά του από τον καθένα. Ο όρος ΄lightweight΄ έχει δύο έννοιες. Η μία έχει να κάνει με την εφαρμογή του σε σχετικά μικρά δίκτυα, ενώ η δεύτερη έχει να κάνει με το μικρό μέγεθος του και την ευκολία εφαρμογής και χρήσης του.

3. Ανάλυση και Σχεδιασμός Συστήματος

Στο κεφάλαιο αυτό για την καλύτερη κατανόηση του συστήματός μας θα δούμε το διάγραμμα Περιπτώσεων Χρήσης και δραστηριοτήτων.

3.1 Διάγραμμα Περιπτώσεων Χρήσης

Αναλύοντας το διάγραμμα της Εικόνας 3.1 βλέπουμε πως το σύστημα IP Reputation αποτελείται από τον **καταχωρητή** τους **κανόνες** και **δυο χρήστες**. Ο χρήστης **Actor** διαχειρίζεται το καταχωρητή και μπορεί να γράφει σε αυτόν. Ο **Actor2** διαχειρίζεται τους κανόνες και μπορεί να επιλέξει και να χρησιμοποιήσει έναν από τους τέσσερις κανόνες. Οι **κανόνες** και ο **καταχωρητής** αλληλοεπιδρούν μεταξύ τους, με τρόπο που θα δούμε παρακάτω στο διάγραμμα δραστηριοτήτων.



Εικόνα 3.1 Διάγραμμα Περίπτωσης Χρήσης

3.2 Διάγραμμα Δραστηριοτήτων

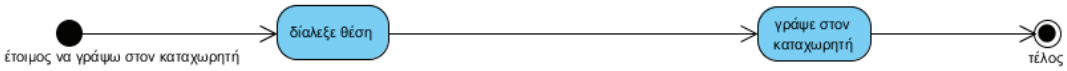
Για να κατανοήσουμε καλύτερα το διάγραμμα περιπτώσεων χρήσης θα δούμε και το διάγραμμα δραστηριότητας της κάθε περίπτωσης χρήσης ξεχωριστά.

3.2.1 Διάγραμμα Δραστηριοτήτων Καταχωρητή

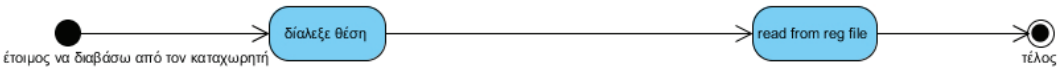
Στο διάγραμμα του καταχωρητή ο διαχειριστής του καταχωρητή μπορεί να κάνει ή εγγραφή ή να διαβάσει. Στο διάγραμμα της Εικόνας 3.2 βλέπουμε το διάγραμμα δραστηριότητας της περίπτωσης χρήσης **Γράφω.**

Στην δραστηριότητα **διάλεξε θέση** ο διαχειριστής επιλέγει σε ποια θέση του καταχωρητή θέλει να κάνει την εγγραφή, και αμέσως μετά κάνει την εγγραφή στον καταχωρητή.

Στο διάγραμμα της Εικόνας 3.3 βλέπουμε το διάγραμμα δραστηριότητας της περίπτωσης χρήσης **Διαβάζω**. Αντίστοιχα και σε αυτό το διάγραμμα ο διαχειριστής επιλέγει από ποια θέση του καταχωρητή θέλει να διαβάσει και έπειτα διαβάζει.



Εικόνα 3.2 Διάγραμμα Δραστηριοτήτων Καταχωρητή Γράψε



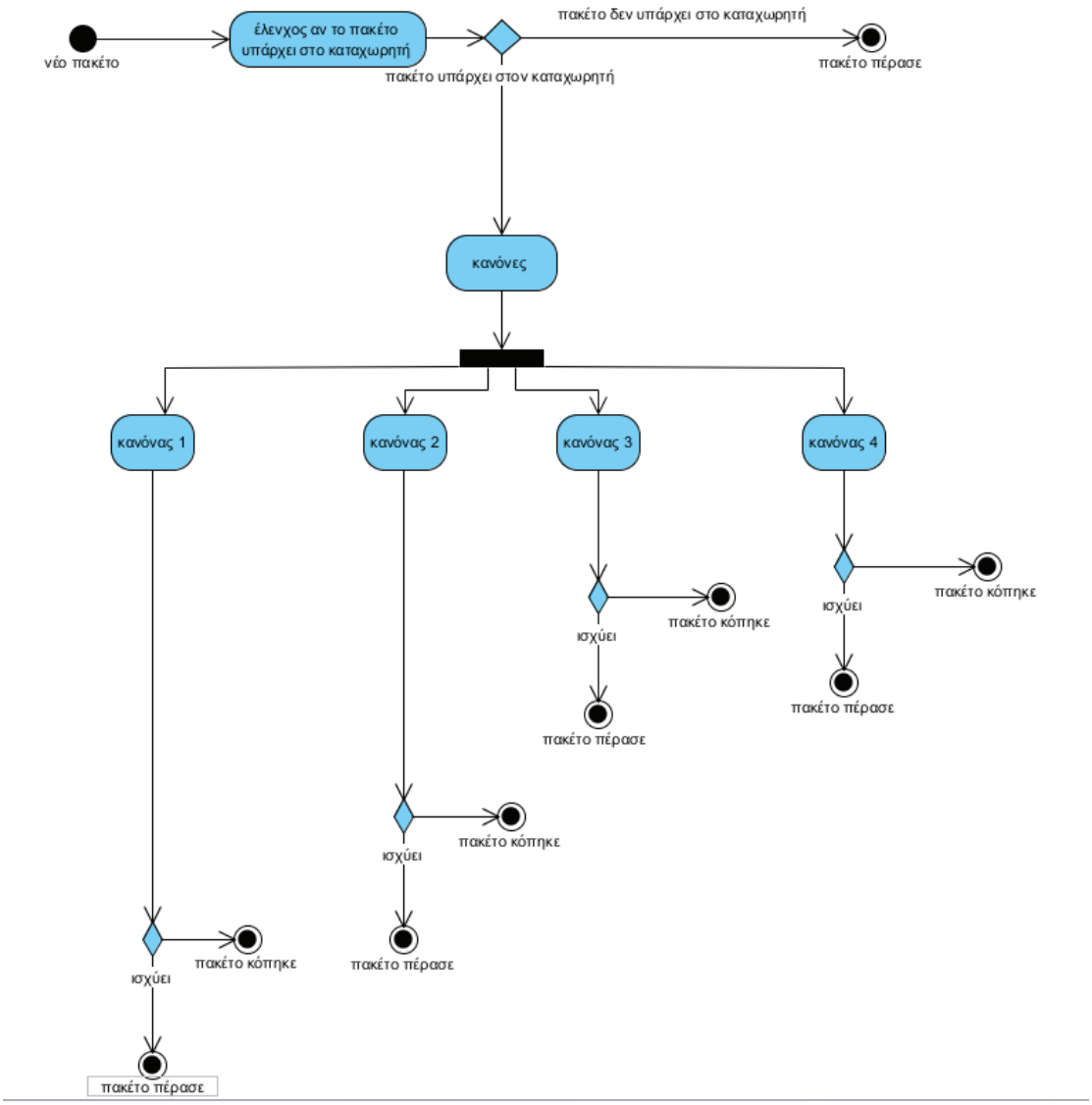
Εικόνα 3.3 Διάγραμμα Δραστηριοτήτων Καταχωρητή Διάβασε

Αντίστοιχα και σε αυτό το διάγραμμα ο διαχειριστής επιλέγει από ποια θέση του καταχωρητή θέλει να διαβάσει και έπειτα διαβάζει.

3.2.2 Διάγραμμα Δραστηριοτήτων Κανόνων

Στο διάγραμμα δραστηριοτήτων κανόνων (Εικόνα 3.4), η αρχή γίνεται όταν έχουμε ένα νέο πακέτο στο σύστημά μας. Το νέο πακέτο αυτό θα συγκριθεί με τα πακέτα που υπάρχουν στον καταχωρητή. Αν το πακέτο υπάρχει στο καταχωρητή, τότε το νέο πακέτο θα «περάσει» από τους **κανόνες**. Ο διαχειριστής κανόνων θα επιλέξει από ποιο από τους τέσσερις κανόνες θέλει να περάσει το νέο του πακέτο, και αν το πακέτο «ακούει» στους κανόνες τότε το πακέτο θα προωθηθεί στον **τελικό χρήστη.**

Αν το νέο πακέτο δεν υπάρχει στον καταχωρητή, τότε το πακέτο δεν θα έχει κακόφημη φήμη, οπότε θα περάσει στον **τελικό χρήστη.**



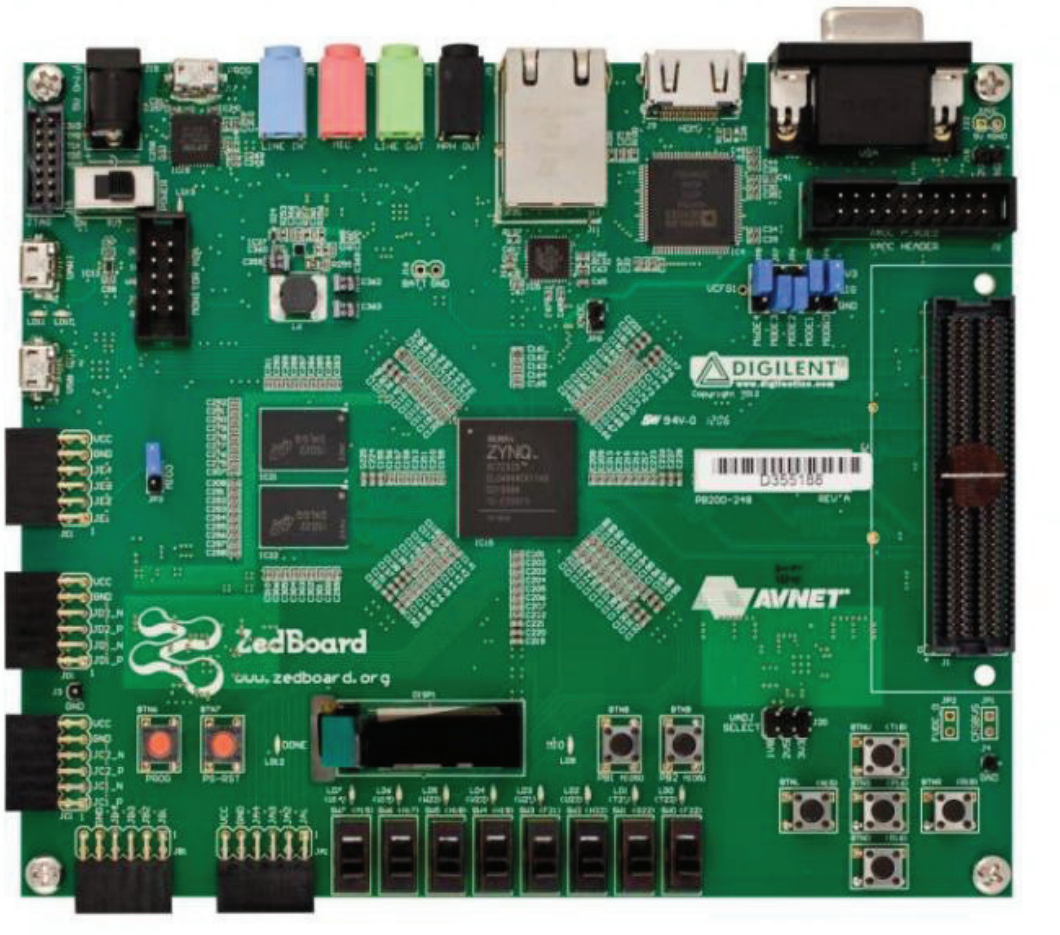
Εικόνα 3.4 Διάγραμμα Δραστηριοτήτων Κανόνων

4. Εργαλεία Σχεδίασης

Για την υλοποίηση του IP Reputation συστήματος χρησιμοποιήσαμε την πλατφόρμα ZedBoard (Zynq Evaluation and Development),και για τον προγραμματισμό της χρησιμοποιήθηκε το Vivado Design Suit.Παρακάτω θα δούμε το καθένα από αυτά ξεχωριστά.

4.1 Το ZedBoard

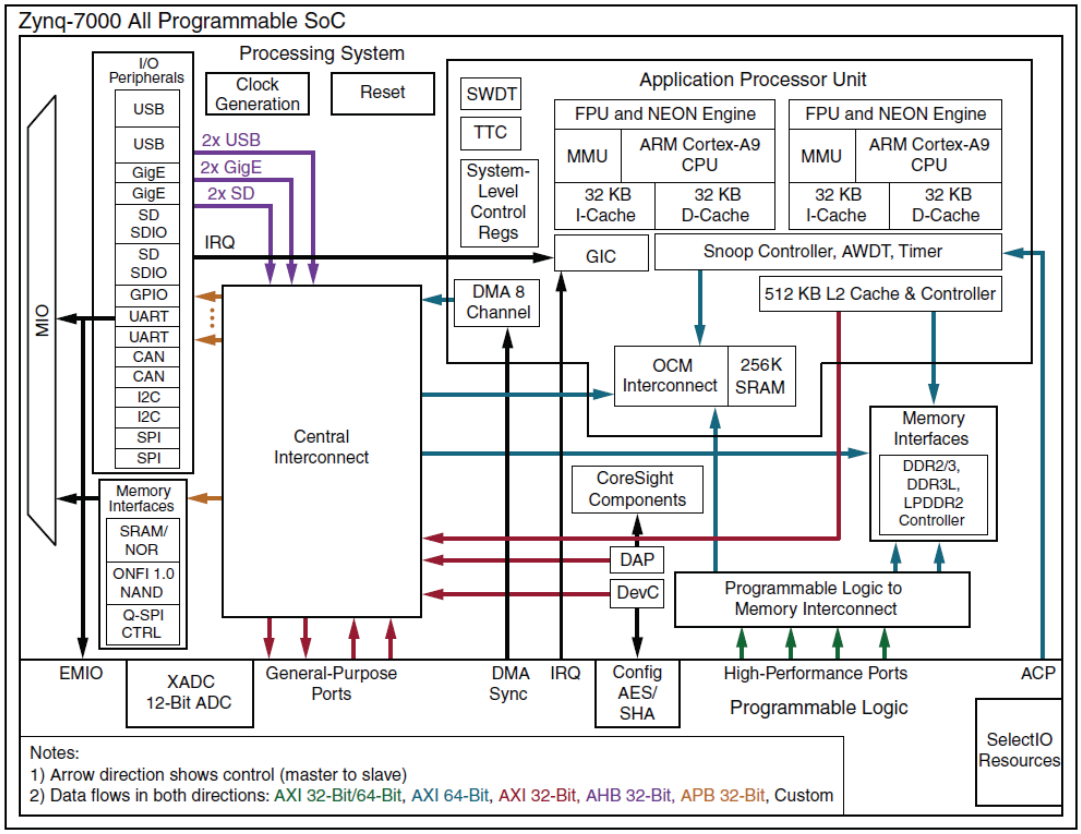
Το ZedBoard (Εικόνα 4.1) είναι ένα ολοκληρωμένο σύστημα βασισμένο στο Xilinx XC7Z020, ένα από τα μικρότερα της οικογένειας των Zynq-7000. Αποτελείται από τα δύο τμήματα του Zynq, το υποσύστημα επεξεργαστή (PS) και το υποσύστημα προγραμματιζόμενης λογικής (PL) που μπορούν λειτουργήσουν αυτόνομα ή σε συνεργασία δημιουργώντας έτσι ένα ευέλικτο σύστημα υψηλής απόδοσης, και από περιφερειακές μονάδες, ενσωματωμένες στο σύστημα. Αυτές συνδέονται με τα παραπάνω τμήματα με φυσικές ή προγραμματιζόμενες διασυνδέσεις υψηλής ταχύτητας, δίνοντας στο σύστημα πολλαπλές δυνατότητες επικοινωνίας με άλλα συστήματα. Έτσι το ZedBoard και το XC7Z020 αποτελούν ιδανικές επιλογές για ανάπτυξη πολυδιάστατων συστημάτων, λόγω της δυνατότητας επικοινωνίας με ανομοιόμορφων προδιαγραφών συστήματα, αλλά ταυτόχρονα υψηλής απόδοσης, λόγω των πολλαπλών δυνατοτήτων για επεξεργασία των δεδομένων που διαχειρίζεται.



Εικόνα 4.1 Το ZedBoard (πηγή http://zedboard.org/product/zedboard)

Το καθοριστικό χαρακτηριστικό του είναι ότι συνδυάζει ένα διπύρηνο επεξεργαστή μαζί με το κλασσικό FPGA και το board ικανό να τρέχει ολόκληρα λειτουργικά συστήματα, όπως Linux, ενώ τα υπόλοιπα προγραμματιζόμενα λογικά τμήματα βασίζονται στην Xilinx 7-series αρχιτεκτονική. Αυτή βασίζεται στις AXI interfaces, οι οποίες παρέχουν μεγάλο εύρος ζώνης και συνδέσεις με μικρή καθυστέρηση. Τα πλεονεκτήματα αυτά βασίζονται στο ότι το Zynq είναι ένα ολοκληρωμένο System on Chip (SOC) και είναι απλούστερο εισάγοντας έτσι μειώσεις στο συνολικό κόστος και το μέγεθος.

Το επεξεργαστικό υποσύστημα αποτελείται από τον διπύρηνο επεξεργαστή ARM CortexA9, ρυθμιζόμενης συχνότητας μέχρι τα 667 MHz, 256 MB ενσωματωμένης μνήμης καθώς και προγραμματιζόμενες διασυνδέσεις για μια σειρά από επίσης προγραμματιζόμενες περιφερειακές μονάδες. Κάθε πυρήνας διαθέτει μια προσωπική κρυφή μνήμη για δεδομένα και μια για εντολές, σε πρώτο επίπεδο, χωρητικότητας 32 KB, ενώ οι δύο πυρήνες διαμοιράζονται με κοινή κρυφή μνήμη σε δεύτερο επίπεδο μεγέθους 512 KB. Με τον τρόπο αυτό η διεπαφή με τη μνήμη φαίνεται ταχύτερη από την πλευρά του επεξεργαστή ανεβάζοντας και την ταχύτητα του συστήματος.



Εικόνα4.2 Το επεξεργαστικό υποσύστημα του Zynq-7000 (πηγή : https://embeddedcentric.com/zynq-training-course/)

Η επικοινωνία του επεξεργαστικού υποσυστήματος με τις περιφερειακές μονάδες επικοινωνίας γίνεται μέσω των πολυπλεγμένων μονάδων εισόδου/εξόδου (Multiplexed Input / Output) ή μέσω πυρήνων, ανεπτυγμένων στο υποσύστημα προγραμματιζόμενης λογικής, που υλοποιούν την ίδια λειτουργία και ονομάζονται πολυπλεγμένες μονάδες επέκτασης εισόδου/εξόδου (Extended Multiplexed Input/Output). Οι μονάδες αυτές, συνδέονται κατάλληλα στον κεντρικό πυρήνα διασύνδεσης κι από εκεί απεικονίζονται στη μνήμη (την ενσωματωμένη στο υπολογιστικό σύστημα ή την επιπλέον DDR3 μνήμη που προσφέρει το ZedBoard,) ώστε τα δεδομένα τους να γίνουν προσπελάσιμα από το επεξεργαστικό υποσύστημα.

Το υποσύστημα προγραμματιζόμενης λογικής παρέχει τη δυνατότητα ανάπτυξης περιφερειακών (soft - peripherals), η οποία επιτρέπει τη δημιουργία εξειδικευμένων συνεπεξεργαστών δεδομένων, που μπορούν να επεξεργαστούν τα δεδομένα αποδοτικότερα από το γενικού σκοπού προγραμματιστικό υποσύστημα. Με αυτόν τον τρόπο επιτυγχάνεται περαιτέρω παραλληλία στο σύστημα, η οποία οδηγεί σε αύξηση της απόδοσης. Ακόμα η δυνατότητα αυτή, είναι σημαντική και για την ανάπτυξη προγραμματιζόμενων διασυνδέσεων. Τέτοιες μονάδες διασυνδέσεων είναι ήδη υλοποιημένες από τη Xilinx με σκοπό την διασύνδεση του υπολογιστικού υποσυστήματος με περιφερειακές μονάδες που δεν συνδέονται με φυσικές συνδέσεις με αυτό.

4.1.1 Προγραμματισμός ZedBoard

Το ZedBoard μπορεί να προγραμματιστεί με 4 διαφορετικούς τρόπους:

USB-JTAG: Αυτή είναι η προεπιλεγμένη και πιο απλή μέθοδο προγραμματισμού του ZedBoard, δεδομένου ότι αυτό μπορεί να γίνει απευθείας πάνω από το καλώδιο USB-micro-USB που παρέχεται με το κιτ ZedBoard.

Traditional JTAG: Μια υποδοχή Xilinx JTAG είναι διαθέσιμη στο Board και μπορεί να χρησιμοποιηθεί στη θέση της σύνδεσης USB-JTAG, αν αυτό είναι επιθυμητό.

Quad-SPI flash: Η μνήμη flash είναι non-volatile και ως εκ τούτου μπορεί να χρησιμοποιηθεί για τη μόνιμη αποθήκευση δεδομένων καταργώντας την απαίτηση για ενσύρματη σύνδεση για τον προγραμματισμό του ZedBoard.

SD κάρτα: Υπάρχει μια υποδοχή SD στην κάτω πλευρά του ZedBoard. Με αυτόν τον τρόπο είναι δυνατή η αποθήκευση των προγραμμάτων στην κάρτα SD προσφέροντας, ουσιαστικά την ίδια δυνατότητα με την Quad-SPI flash.

4.2 Η VHDL

Η VHDL είναι μία γλώσσα περιγραφής υλικού, που χρησιμοποιείται για την ανάπτυξη ολοκληρωμένων ψηφιακών κυκλωμάτων και συστημάτων. Αρχικά η δημιουργία της αποσκοπούσε στη μοντελοποίηση και στην προσομοίωση κυκλωμάτων, γι' αυτό πολλά χαρακτηριστικά της γλώσσας έχουν σκοπό την προσομοίωση των λειτουργιών. Αργότερα, η γλώσσα χρησιμοποιήθηκε και ως εργαλείο σύνθεσης. Κατά τη σύνθεση, ο μεταγλωττιστής συνθέτει ένα πραγματικό κύκλωμα που ανταποκρίνεται με ακρίβεια στη λογική και χρονική περιγραφή, την οποία μοντελοποιεί ο κώδικας. Ο όρος VHDL είναι συντόμευση των λέξεων VHSIC Hardware Description Language, όπου VHSIC σημαίνει Very High Speed Integrated Circuit. Η γλώσσα αυτή αναπτύχθηκε στις αρχές της δεκαετίας του 1980 με χρηματοδότηση από το υπουργείο άμυνας των ΗΠΑ και έγινε πρότυπη το 1987 από το ινστιτούτο IEEE ως IEEE 1076. Αργότερα, δημιουργήθηκε μια βελτιωμένη έκδοσή της, η IEEE 1164 (1993). Ακολούθησαν και νεότερες αναβαθμίσεις του προτύπου, όπως η IEEE 1076-2002 και IEEE 1076-2008.

4.2.1 Χαρακτηριστικά της γλώσσας VHDL

Η VHDL υπακούει στις αρχές των παράλληλων γλωσσών και δεν είναι τυχαίο ότι έχει τις ρίζες της στην Ada, που χρησιμοποιείται για να προγραμματιστεί παράλληλες διεργασίες. Ταυτόχρονα, υπακούει στις αρχές του δομημένου προγραμματισμού, που δίνει τη δυνατότητα της σχεδίασης ιεραρχικών κυκλωμάτων. Τέλος, περιέχει τις αρχές του συγχρονισμού και του χρονισμού, που είναι εγγενείς στα κυκλώματα. Ένα από τα χαρακτηριστικά της VHDL

είναι ότι μοντελοποιεί με ακρίβεια τόσο τις λειτουργίες του κυκλώματος, όσο και τους χρόνους κατά τους οποίους οι λειτουργίες παράγουν αποτελέσματα.

Η VHDL διαφέρει από τις συμβατικές γλώσσες κατά το ότι δεν προορίζεται να περιγράψει λειτουργίες που εκτελούνται σειριακά, η μία μετά την άλλη. Κάθε πρόταση ή τμήμα κώδικα περιγράφει λειτουργίες, οι οποίες παράγουν αποτελέσματα σε συγχρονισμό με άλλες λειτουργίες,. Τα αποτελέσματα της προσομοίωσης παράγονται με βάση αυστηρές χρονικές προδιαγραφές σε διάφορα σημεία του κυκλώματος και εν τέλει στις εξόδους. Με την έννοια αυτή, είναι δυνατό ένα τμήμα κώδικα να μπορεί να γραφεί σε οποιοδήποτε σημείο του προγράμματος, αφού παράγει αποτελέσματα ανεξαρτήτως της σειράς του. Στο τέλος, ο compiler θα συνθέσει κυκλώματα που θα υλοποιούν στην πράξη τις σύγχρονες λειτουργίες που περιγράφει ο κώδικας.

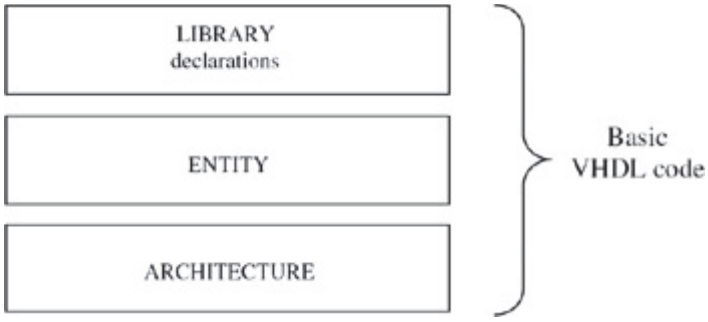
Η VHDL μπορεί να περιγράψει τόσο σύγχρονες όσο και ακουλουθιακές λογικές λειτουργίες. Οι διεργασίες που περιγράφονται στη VHDL παράγουν αποτέλεσμα είτε ταυτόχρονα με την εφαρμογή των εισόδων, όπως συμβαίνει στα συνδυαστικά κυκλώματα, είτε σε συγχρονισμό με συμβάντα (π.χ. παλμούς ρολογιού), όπως συμβαίνει στα ακολουθιακά κυκλώματα. Για το λόγο αυτό η σύνταξη του κώδικα μπορεί να γίνει με σύγχρονες δομές (concurrent code) ή και με ακουλουθιακές δομές (sequential code).

Ο ιεραρχικός τρόπος σχεδίασης γίνεται δυνατός στη VHDL εξαιτίας της δομής του κώδικα, που διακρίνει ανάμεσα στην περιγραφή του κυκλώματος ως βαθμίδα (block) και στη λειτουργική του περιγραφή. Τα δύο αυτά μέρη του κώδικα αναφέρονται ως «οντότητα» και «αρχιτεκτονική» (βλέπε επόμενη παράγραφο). Έχοντας σχεδιάσει ένα κύκλωμα σε VHDL, είναι δυνατό να το ενσωματώνουμε σε πιο περίπλοκα κυκλώματα, κάνοντας απλώς αναφορά στην «οντότητά» του, χωρίς να χρειάζεται να επαναλαμβάνουμε την περιγραφή της αρχιτεκτονικής.

Μπορούμε να παράγουμε «στιγμιότυπα» (Instances) ενός κυκλώματος, όσες φορές χρειάζεται, κάνοντας αναφορά σ' αυτό μέσα στην αρχιτεκτονική μιας ανώτερης ιεραρχικής βαθμίδας. Η δυνατότητα της επανάληψης των στιγμιοτύπων συνιστά ένα χαρακτηριστικό της γλώσσας VHDL που καθιστά την περιγραφή του υλικού (hardware) επαναχρησιμοποιήσιμη (reusable). Κάθε κύκλωμα που περιγράφεται μία φορά μπορεί να χρησιμοποιηθεί ξανά σαν βαθμίδα υποκυκλώματος, σε οποιοδήποτε άλλο ψηφιακό σύστημα. Με τον τρόπο αυτό, το hardware στην κωδική του περιγραφή, καθίσταται ευέλικτο όπως και το Software, αφού πλέον μπορεί να διαμοιραστεί στους χρήστες σαν λογισμικό ή να χρησιμοποιηθεί για την ανάπτυξη μμεγάλων συστημάτων, μέσω βιβλιοθηκών που περιλαμβάνουν βαθμίδες υποκυκλωμάτων. Τέτοιες βιβλιοθήκες υπάρχουν έτοιμες στα μεγάλα εργαλεία ψηφιακής σχεδίασης, όπως το Quartus II, αλλά μπορούν να δημιουργηθούν και από τον κάθε σχεδιαστή, ανάλογα με τις δικές του ανάγκες σχεδίασης.

4.2.2 Τα βασικά μέρη του κώδικα

Τα βασικά μέρη ενός κώδικα που περιγράφει κύκλωμα σε VHDL είναι η «οντότητα» (entity) και η «αρχιτεκτονική» (architecture). Στις περισσότερες περιπτώσεις, πριν το τμήμα της οντότητας θα πρέπει να δηλωθούν και κάποια πακέτα βιβλιοθηκών, που περιγράφουν τους τύπους δεδομένων ή υποκυκλώματα που χρησιμοποιεί ο σχεδιαστής. Το σχήμα παρακάτω δείχνει την βασική δομή ενός αρχείου VHDL. Παρακάτω περιγράφεται με συντομία αυτή η δομή.



Εικόνα 4.3 Δομή της VHDL

Οι δηλώσεις των βιβλιοθηκών (libraries) και των πακέτων (packages) επιτρέπουν τη χρήση συγκεκριμένων τμημάτων κώδικα, που έχουν ήδη δημιουργηθεί στο παρελθόν και χρησιμοποιούνται συχνά. Μια βιβλιοθήκη, λοιπόν, είναι συλλογή χρήσιμων τμημάτων κώδικα. Τέτοια τμήματα κώδικα είναι δηλώσεις τύπων δεδομένων, υποκυκλώματα, συναρτήσεις (functions), διαδικασίες (procedures). Όταν τοποθετούνται σε μια βιβλιοθήκη, οι κώδικες αυτοί μπορούν να χρησιμοποιηθούν ξανά, σε άλλα σχέδια, κάνοντας απλά μια δήλωση της βιβλιοθήκης στην αρχή του κώδικα. Ένα έτοιμο, τυποποιημένο πακέτο που χρησιμοποιείται πολύ συχνά είναι το std_logic_1164 της βιβλιοθήκης ieee, το οποίο περιγράφει τον τύπο δεδομένων std_logic. Οι βιβλιοθήκες δηλώνονται με τη λέξη-κλειδί LIBRARY, ενώ τα πακέτα εισάγονται με τη λέξη- κλειδί USE. Εκτός από τις τυποποιημένες βιβλιοθήκες, ο κάθε χρήστης μπορεί να δημιουργεί τις δικές του.

Η οντότητα (entity) περιγράφει το κύκλωμα ως βαθμίδα, με εισόδους και εξόδους. Περιλαμβάνει μόνο τις διασυνδέσεις που έχει το κύκλωμα με άλλες βαθμίδες αλλά αποκρύπτει τη λειτουργία του κυκλώματος. Το αναγνωριστικό όνομα που δίνει ο σχεδιαστής στην οντότητα καθώς και τα σήματα εισόδου και εξόδου είναι καθοριστικά για κάθε υλοποίηση του κυκλώματος αυτού.

Η αρχιτεκτονική (architecture) περιλαμβάνει όλες τις λεπτομέρειες της λειτουργίας ενός κυκλώματος. Οι εντολές και οι δηλώσεις που περιλαμβάνονται στο σώμα της αρχιτεκτονικής περιγράφουν με ακρίβεια ποιες λογικές συναρτήσεις θα υλοποιεί το κύκλωμα και τι τιμές θα λαμβάνουν τα σήματα του κυκλώματος σε κάθε χρονική στιγμή.

4.2.3 Πλεονεκτήματα της VHDL

Το βασικό πλεονέκτημα της VHDL, όταν αυτή χρησιμοποιείται για σχεδίαση συστημάτων, είναι ότι επιτρέπει την περιγραφή (μοντελοποίηση) και την επαλήθευση (προσομοίωση) του επιθυμητού συστήματος, πριν τα εργαλεία σύνθεσης μεταφράσουν τη σχεδίαση σε πραγματικό υλικό (πύλες και γραμμές).

Ένα άλλο όφελος της VHDL είναι ότι επιτρέπει τον ορισμό ταυτόχρονων συστημάτων (concurrent systems). Η VHDL είναι γλώσσα ροής δεδομένων, σε αντίθεση με τις διαδικαστικές γλώσσες προγραμματισμού όπως η BASIC, η C και η συμβολική γλώσσα, οι οποίες εκτελούνται ακολουθιακά, με κάθε εντολή να ακολουθεί την προηγούμενη.

Ένα έργο σε VHDL έχει πολλές εφαρμογές. Ένα μπλοκ υπολογισμού (calculation block) δημιουργείται μια φορά αλλά μπορεί να χρησιμοποιηθεί σε άλλα έργα. Μπορούν επίσης να ρυθμιστούν διάφορες παράμετροι διαμόρφωσης και λειτουργίας του μπλοκ (παράμετροι χωρητικότητας, το μέγεθος της μνήμης, η βάση των στοιχείων (element base), η σύνθεση μπλοκ και η δομή διασύνδεσης).

Ένα έργο σε VHDL είναι επίσης μεταφέρσιμο. Αν έχει δημιουργηθεί για μια βάση στοιχείων, μπορεί να μεταφερθεί σε μια άλλη βάση, για παράδειγμα σε ένα VLSI με διάφορες τεχνολογίες.

4.3 Vivado Design Suit

Για να ξεκινήσει κανείς να σχεδιάζει για την πλατφόρμα της Zynq θα πρέπει να έχει τα κατάλληλα εργαλεία. Υπάρχουν πολλά, αλλά αυτό που πραγματικά χρειαστήκαμε είναι το **Vivado Design Suit**.Υπάρχουν τρείς διαθέσιμες εκδόσεις για τη λειτουργία του. Αυτές είναι WebPACK, Design edition και System Edition.

Για την υλοποίηση του συστήματός μας χρησιμοποιήσαμε το System Edition.Θα δούμε πως το πακέτο Vivado IDE και SDK μας παρέχουν τα απαραίτητα εργαλεία σχεδίασης για το hardware και το software αντίστοιχα.

- **Vivado IDE**: είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης δημιουργίας hardware συστημάτων, τα οποία είναι τμήμα κάποιου Soc. Όπως για παράδειγμα μνήμες, επεξεργαστές, εξωτερικές διασυνδέσεις και κανάλια διαύλων. Το πακέτο Vivado IDE αλληλοεπιδρά με τα άλλα σχεδιαστικά εργαλεία και μπορεί να διευκολύνει την ολοκλήρωση με το υπόλοιπο σύστημα και έτσι να βελτιώσει την πιθανότητα επαναχρησιμοποιήσεις της ίδιας τοπολογίας hardware.
- **SDK** : είναι ένα δημοφιλές εργαλείο ανάπτυξης κώδικα βασισμένο στην πλατφόρμα Eclipse, η οποία περιλαμβάνει drivers για όλες Xilinx IPs, GCC βιβλιοθήκες για ARM και για NEON επεκτάσεις που χρησιμοποιούν C και C++ γλώσσες προγραμματισμού.

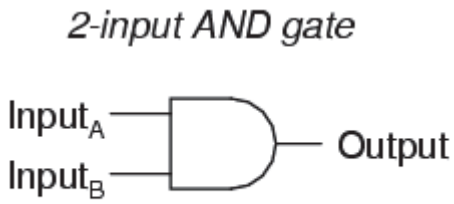
- **Vivado Simulator:** είναι ένα περιβάλλον για testing, των hardware τμημάτων του συστήματος.
- **Vivado Logic Analyzer**: είναι ένα εργαλείο το οποίο παρέχει διευκολύνσεις και επιτρέπει την ανάλυση εσωτερικών σημάτων πάνω σε οποιαδήποτε τοπολογία έχουμε δημιουργήσει και τρέχει πάνω στο PL.
- **Vivado Serial I/O Analyzer**: όπως υπονοεί και το όνομά του είναι ένα εργαλείο το οποίο σχετίζεται με τις διεπαφές επικοινωνίας ανάμεσα στα περιφερειακά του συστήματός μας.
- **Vivado High Level Synthesis (HLS) and system generator for DSP:** χρησιμοποιούνται για δημιουργία, testing και διαχείριση των IP για να συμπεριληφθούν τελικά μέσα στο hardware σύστημα.

4.3.1 Υλοποίηση στο Vivado

Έχοντας δει τα βασικά στοιχεία του Vivado αλλά και τα βασικά στοιχεία της γλώσσας περιγραφής υλικού VHDL, θα δείξουμε ένα παράδειγμα για το πώς υλοποιούνται όλα αυτά στο Vivado.

Στο παράδειγμά μας θα χρησιμοποιήσουμε την λογική πράξη **AND** μεταξύ δύο αριθμών. Θα δούμε λοιπόν πως ξεκινάμε ένα καινούργιο project στο Vivado, πως προσθέτουμε την δική μας λογική, αλλά και πως κάνουμε εξομοίωση.

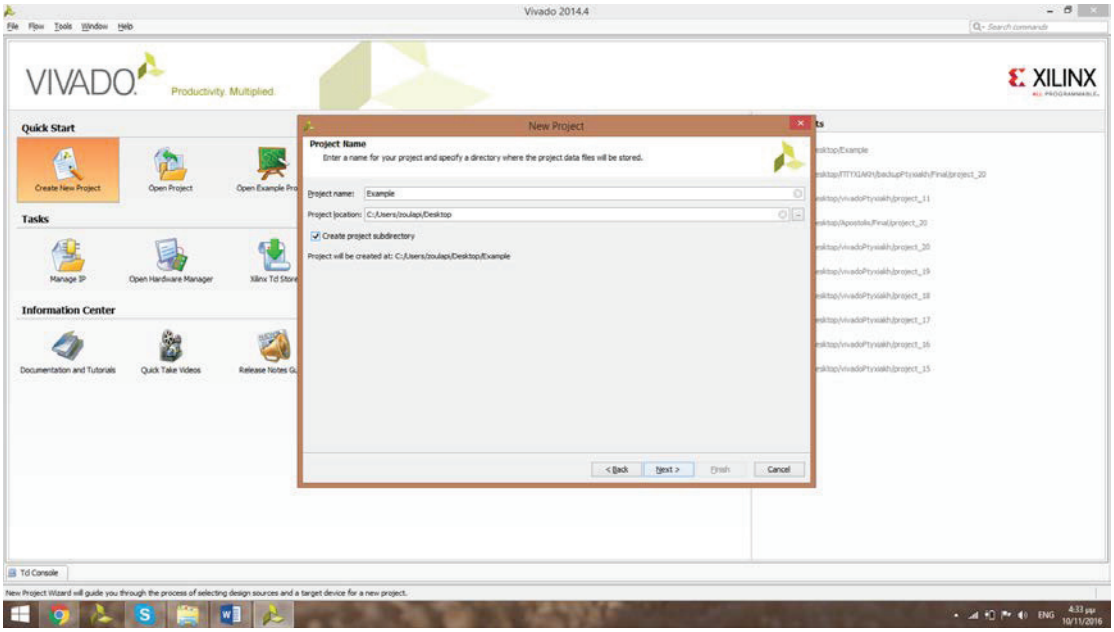
Δίνεται παρακάτω στην Εικόνα 4.4 ο πίνακας αληθείας της λογικής πύλης **AND** που θα χρειαστούμε για το παράδειγμά μας.



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

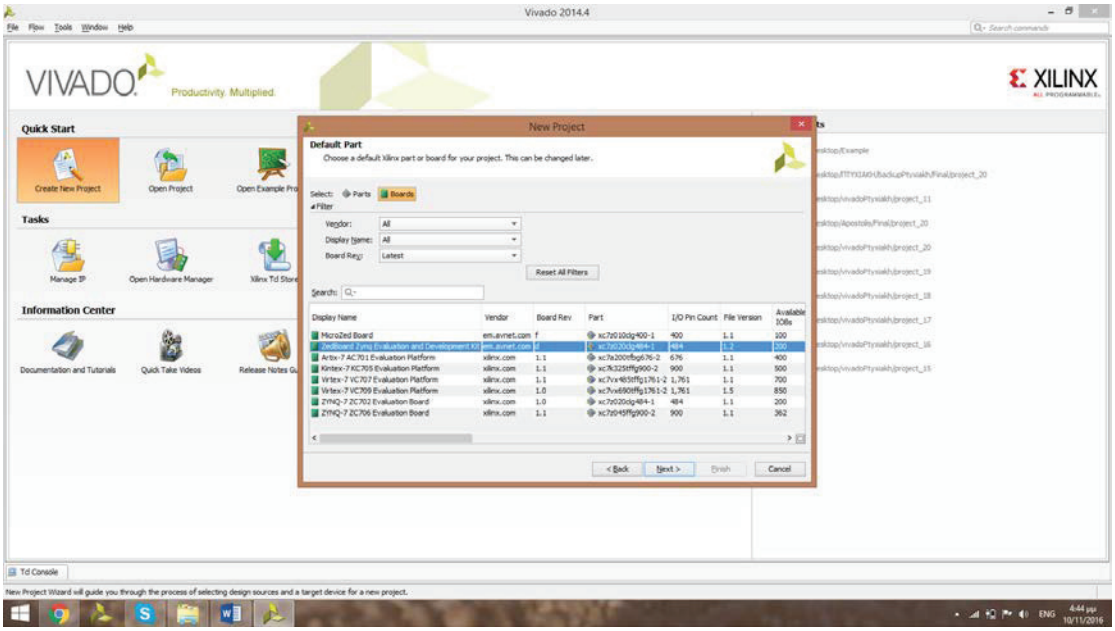
Εικόνα 4.4 Πίνακας αληθείας πύλης AND (πηγή: http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/and.html)

Ανοίγοντας το Vivado επιλέγουμε την επιλογή **Create New Project** και έπειτα **Next.** Στο σημείο αυτό επιλέγουμε το όνομα του project μας. Εγώ για το παράδειγμά μας έβαλα **Example.**



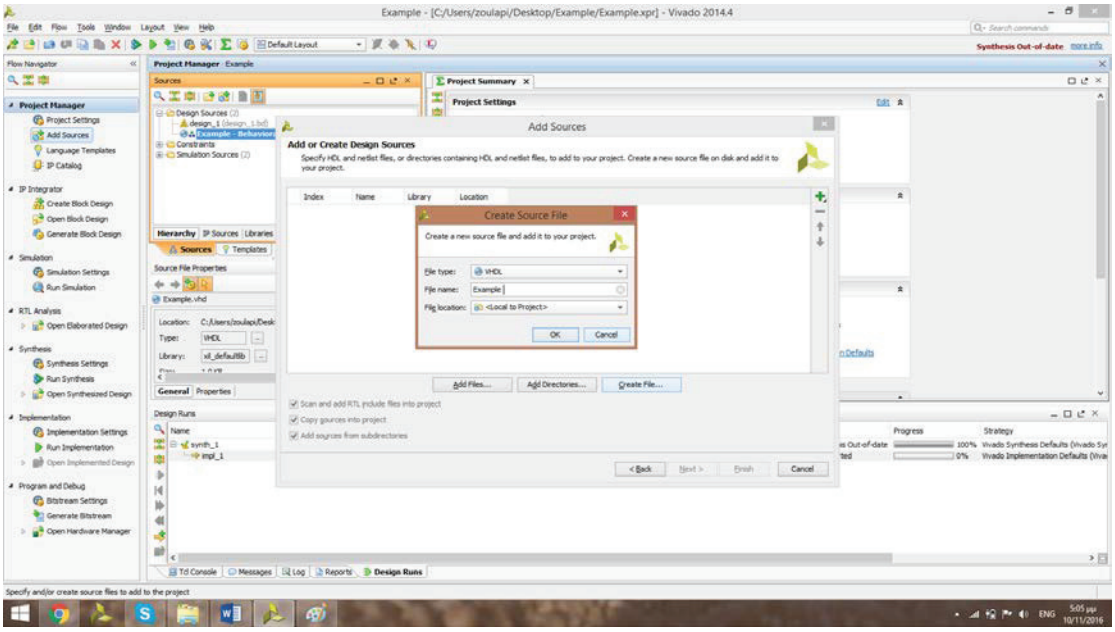
Εικόνα 4.5 Δημιουργία νέου project

Έπειτα το Vivado μας δίνει την επιλογή να επιλέξουμε ποια πλακέτα της Xilinx θα χρησιμοποιήσουμε. Για το παράδειγμά μας δεν θα χρησιμοποιήσουμε κάποια πλακέτα, αλλά για το Σύστημα IP Reputation βάλαμε την επιλογή **ZedBoard Zynq Evaluation and Development Kit**



Εικόνα 4.6 Επιλογή πλακέτας

Πατώντας **Next** είμαστε έτοιμοι με την δημιουργία του νέου project και το Vivado έχει ανοίξει το Wizard. Επιλέγουμε πάνω αριστερά **Add Sources** για να προσθέσουμε την δική μας λογική στο project και επιλέγουμε **Create File**, έπειτα επιλέγουμε όνομα για την δική μας αρχιτεκτονική.



Εικόνα 4.7 Δημιουργία source File

Ο wizard του Vivado δημιούργησε και ενσωμάτωσε την αρχιτεκτονική μας στο project και είμαστε πλέον έτοιμοι να προσθέσουμε την δική μας λογική η οποία φαίνεται στην Εικόνα 4.8.

```
entity Example is
 Port (NUM1 : in  STD_LOGIC;
           NUM2 : in  STD_LOGIC;
           SUM : out  STD_LOGIC );
end Example;

architecture Behavioral of Example is

begin
SUM <= NUM1 and NUM2;

end Behavioral;
```

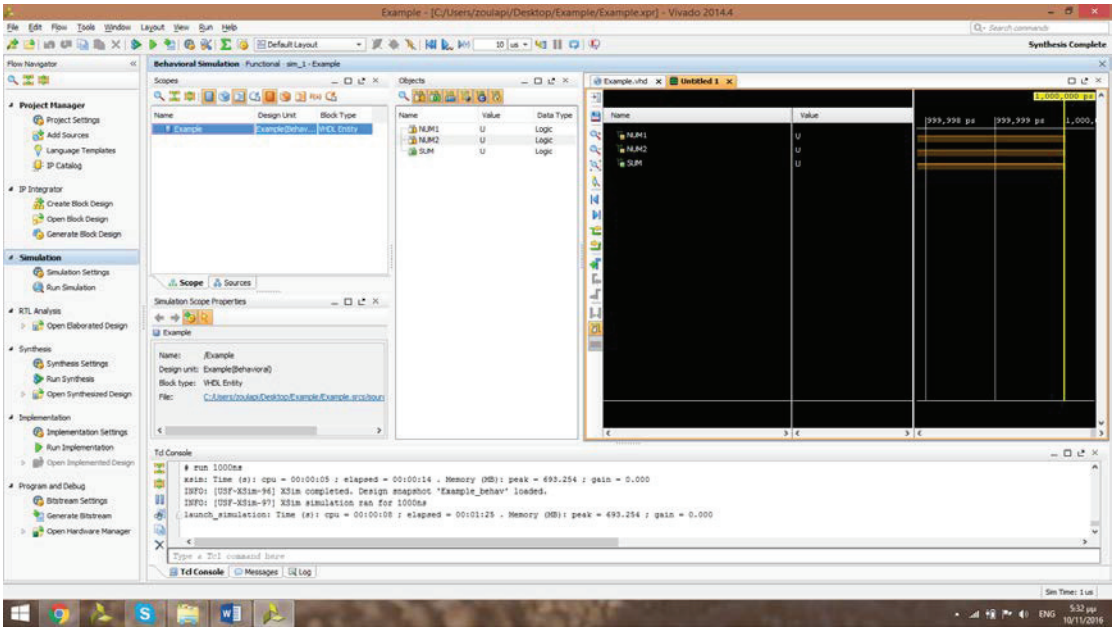
Εικόνα 4.8 Κώδικας της AND

Αφού δημιουργήσαμε και την δική μας λογική που θέλουμε να υλοποιήσουμε επιλέγουμε στο αριστερό μενού την επιλογή **Run Synthesis**. Εφόσον όλα πάνε καλά το Vivado θα μας βγάλει μήνυμα **Synthesis Successfully Complete.**

Σε αυτό το στάδιο έχουμε δημιουργήσει το project μας και έχουμε συμπεριλάβει την δική μας λογική. Πως γνωρίζουμε όμως ότι η λογική μας

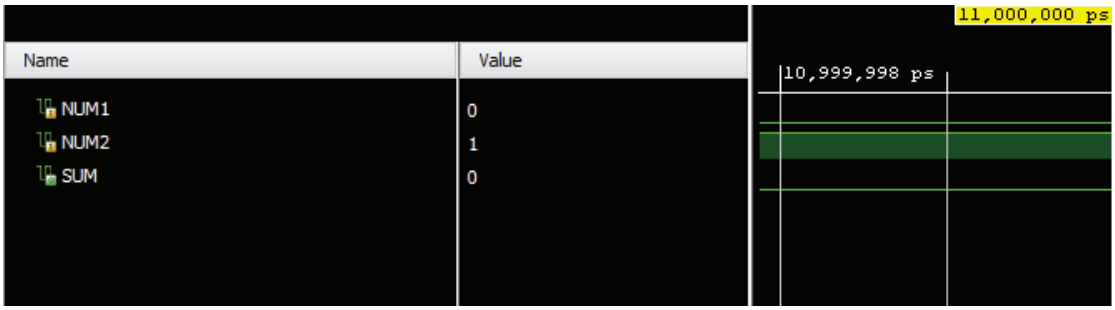
δουλεύει και κάνει αυτό που εμείς θέλουμε. Το Vivado παρέχει εξομοιωτή που θα χρησιμοποιήσουμε για επαλήθευση.

Πάλι στο menu αριστερά, επιλέγουμε την επιλογή **Run Simulation**



Εικόνα 4.8 Εξομοιωτής

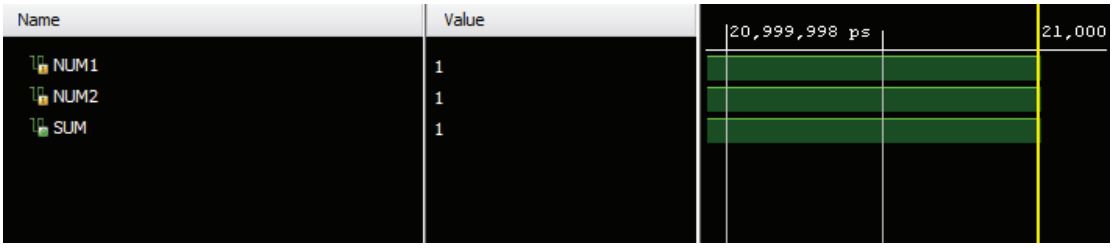
Αφού ανοίξει ο εξομοιωτής του Vivado, έχουμε την δυνατότητα να δώσουμε τιμές για τις εισόδους μας **NUM1** και **NUM2**. Στα πλαίσια του παραδείγματος θα δώσω την πρώτη φορά για το NUM1 την τιμή 0 και για το NUM2 την τιμή 1 και θα τρέξω την εξομοίωση.



Εικόνα 4.9 Αποτέλεσμα εξομοίωσης 1

Στο αριστερό μέρος της Εικόνας 4.9 βλέπουμε τα ονόματα τον μεταβλητών μας και στο δεξί μέρος βλέπουμε τις τιμές που παίρνουν από την στιγμή που θα ενεργοποιηθεί το σύστημα. Το Sum παρατηρούμε πώς πήρε την τιμή 0.Ο πίνακας αληθείας της Εικόνας 4.4 μας επιβεβαιώνει το αποτέλεσμα.

Θα αλλάξουμε τώρα τις τιμές στις εισόδους μας και θα βάλουμε τόσο στο NUM1 όσο και στο NUM2 την τιμή 1.

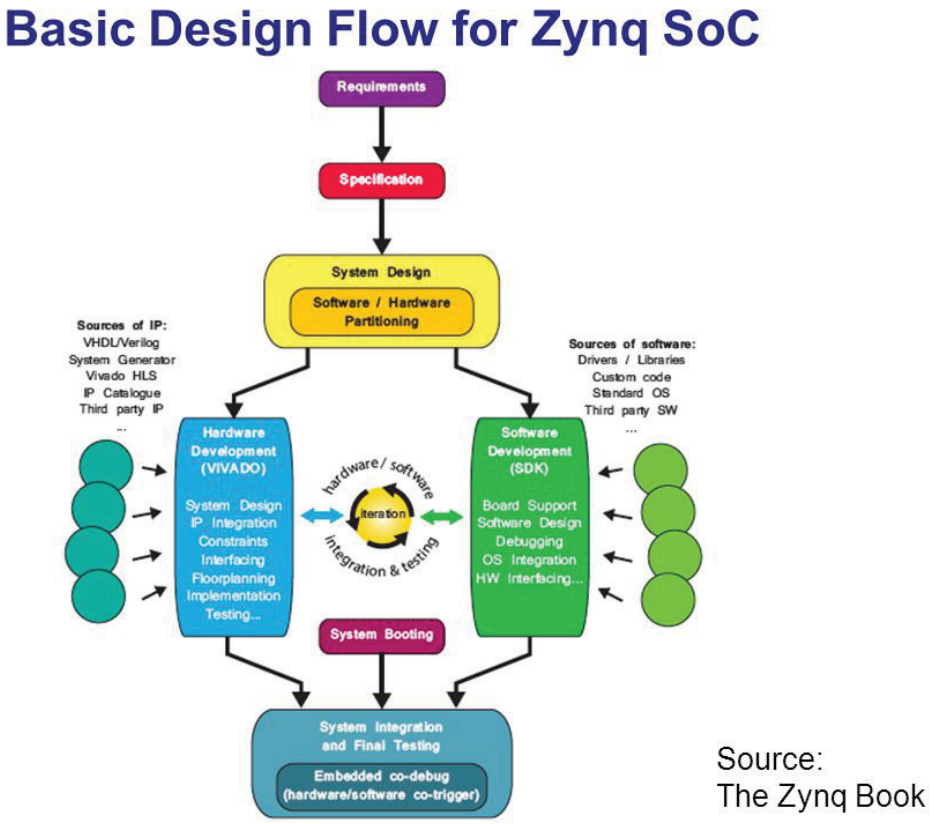


Εικόνα 4.10 Αποτέλεσμα εξομοίωσης 2

Παρατηρούμε πως το Sum πήρε την τιμή 1 αυτή την φορά. Ο πίνακας αληθείας της Εικόνας 4.4 μας επιβεβαιώνει και πάλι το αποτέλεσμα.

4.4 Ροή Σχεδίασης σε Zynq

Έχοντας θέσει τις απαιτήσεις του συστήματος όσον αφορά το software και το hardware οδηγείτε κανείς στη σχεδίαση ενός διαγράμματος ροής για την ανάπτυξη εφαρμογών πάνω σε Zynq.



Εικόνα 4.11 Ροή σχεδίασης

Στην κορυφή τους διαγράμματος της Εικόνας 4.11 έχουμε τις απαιτήσεις και τους προσδιορισμούς του συστήματος. Ακολουθεί ο σχεδιασμός του

συστήματος, που χωρίζεται σε δυο μέρη. Το ένα του κομμάτι είναι ο σχεδιασμός του hardware και το άλλο του κομμάτι είναι η ανάπτυξη software κώδικα για τον προγραμματισμό του Arm. Τέλος για την ολοκλήρωση μένει το τελικό testing και εξομοιώσεις που θα μας βοηθήσουν στην αποσφαλμάτωση και στην επαλήθευση της ορθής λειτουργίας του συστήματος.

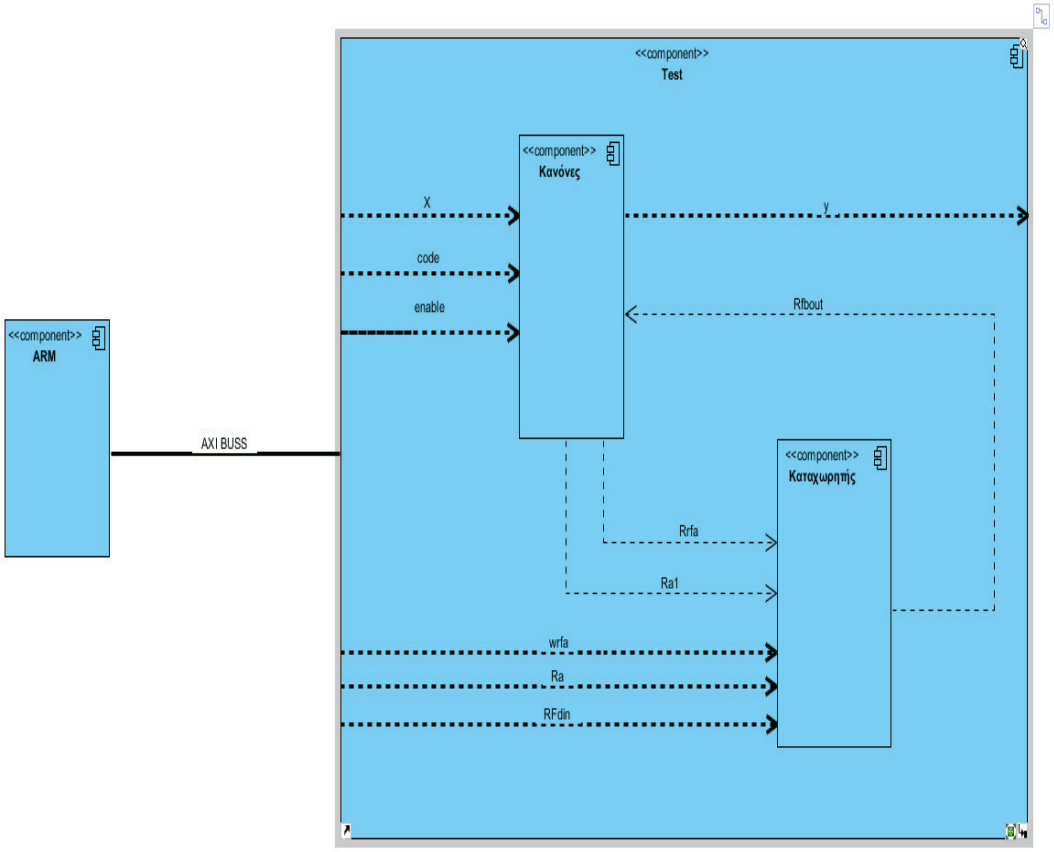
Βάση αυτής της ροής σχεδίασης προχωρήσαμε στην υλοποίηση του συστήματός μας που θα δούμε στο επόμενο κεφάλαιο.

5. Υλοποίηση

Έχοντας θέσει τα προβλήματα ασφάλειας που προκύπτουν στο διαδίκτυο των αντικειμένων, έχοντας πει πλέον το προτεινόμενο σύστημα και έχοντας αναλύσει και σχεδιάσει τα διαγράμματα περιπτώσεων χρήσης και δραστηριοτήτων, είναι η ώρα να παρουσιάσουμε την υλοποίηση του συστήματός μας. Στο κεφάλαιο αυτό, πού είναι και η καρδία αυτής της πτυχιακής, θα παρουσιαστεί το Block διάγραμμα, ο VHDL κώδικας με πλήρη επεξήγηση, θα εξομοιωθούν τέσσερα διαφορετικά σενάρια και τέλος θα αναφέρουμε σχεδιαστικές ικανότητες και προβλήματα που προέκυψαν στην υλοποίηση του συστήματος.

5.1 Block Διάγραμμα

Στην Εικόνα 5.1 παρουσιάζετε το block διάγραμμα του συστήματος που αναπτύχθηκε. Το διάγραμμα αποτελείται από τον **καταχωρητή** και τους **κανόνες,** από το **AXI BUS,**τον **ARM** και τέλος από τα **εσωτερικά σήματα** που δείχνουν πως αλληλοεπιδρούν μεταξύ τους τα blocks.



Εικόνα 5.1 Block διάγραμμα

Η λογική του συστήματος είναι πως κάθε φορά που θα έχουμε μια καινούρια εισερχόμενη ip, αυτή θα πηγαίνει στο Block **Κανόνες**,θα συγκρίνεται με το IP reputation που θα βρίσκετε στον **καταχωρητή** και αν υπάρχει, τότε η ip θα περνάει από ένα σύνολο κανόνων. Αν η ip ικανοποιεί τους κανόνες τότε θα προωθείτε αλλιώς θα κόβετε. Να σημειωθεί σε αυτό το σημείο πως το IP αποτελείται από 44 bits, από τα οποία τα 32 πρώτα bits είναι η ip, τα επόμενα 4 είναι το Reliability, τα επόμενα 4 είναι το Risk και τα τελευταία 4 είναι το Activity. Το Reliability, Risk, και το activity παίρνουν τις τιμές από 0 έως 15 δηλαδή από 0000 έως 1111.

Τα components κανόνες και καταχωρητής είναι το δικό μας περιφερειακό. Για να τροφοδοτήσουμε το σύστημά μας με καινούργιες ip,για να γεμίσουμε τον καταχωρητή αλλά και για να κάνουμε επιλογή κανόνα που θα χρησιμοποιηθεί χρησιμοποιήσαμε τον επεξεργαστή ARM που βρίσκεται πάνω στο ZedBoard.Η επικοινωνία του Arm με το δικό μας περιφερειακό γίνεται μέσω του Axi bus.

5.1.1 Ο καταχωρητής

Ξεκινώντας με τον καταχωρητή αναφέρεται πως στη μηχανική υπολογιστών, ο καταχωρητής είναι τύπος μικρής αλλά πολύ γρήγορης μνήμης που βρίσκεται μέσα στο τσιπ του επεξεργαστή. Η μνήμη αυτή χρησιμοποιείται για την βελτίωση της ταχύτητας εκτέλεσης των διαφόρων προγραμμάτων, αφού σε αυτήν συνήθως αποθηκεύονται δεδομένα που χρησιμοποιούνται συνέχεια από τα προγράμματα. Στην περίπτωση αυτή ο καταχωρητής παρέχει πολύ γρήγορη πρόσβαση σε αυτά τα δεδομένα και έτσι το πρόγραμμα εκτελείται πιο γρήγορα.

Για τις ανάγκες του συστήματός μας, υλοποιήθηκε καταχωρητής δεκαέξι θέσεων, όπου θα αποθηκεύονται τα κακόφημα IP.Οι βασικές λειτουργίες του καταχωρητή είναι το γράψιμο και το διάβασμα. Ο καταχωρητής επιπλέον θα μπορεί να ανανεώνετε από τον διαχειριστή του συστήματος και μάλιστα θα μπορεί να αλλάξει μια η περισσότερες τιμές του, ανάλογα με την επιθυμία του. Αυτό επιτυγχάνετε με την βοήθεια ενός σήματος όπου θα ανατρέχει σε οποιαδήποτε θέση εμείς του ορίσουμε. Τα σήματα θα τα δούμε αναλυτικότερα παρακάτω.

5.1.2 Οι κανόνες

Οι κανόνες είναι το κύριο Block του συστήματος. Οι κανόνες είναι το Block που θα δέχεται την καινούρια ip και με μια σειρά από ελέγχους θα αποφασίζεται αν αυτή η ip θα επιτραπεί να περάσει η όχι. Οι κανόνες με τον καταχωρητή όπως είναι φυσικό είναι σε άμεση επικοινωνία μεταξύ τους. Η επικοινωνία αυτή επιτυγχάνεται μέσω σημάτων που θα δούμε παρακάτω. Όταν αναφέρεται πως μια ip ικανοποιεί τους κανόνες, εννοούμε πως το Reliability, Risk και activity βρίσκετε μέσα στις επιτρεπόμενες τιμές. Για τις ανάγκες του δικού μας συστήματος υλοποιήθηκαν 4 κανόνες.

Κανόνας 1 : Ο πρώτος κανόνας που υλοποιήσαμε εξετάζει το ρίσκο που έχει μια ip να είναι μολυσμένη. Ορίσαμε επιτρεπόμενες τιμές για αυτό τον κανόνα το **Risk να είναι μικρότερο του 9**. Οποιαδήποτε εισερχόμενη ip έχει Risk μικρότερο του 9 θα επιτρέπεται να περάσει, σε διαφορετική περίπτωση η ip θα κρίνεται επικίνδυνη και θα κόβεται.

Κανόνας 2 : Ο δεύτερος κανόνας που υλοποιήθηκε εξετάζει μια ip από την πλευρά της αξιοπιστίας της. Σε αυτή την περίπτωση μια εισερχόμενη ip θα επιτραπεί να περάσει μόνο αν έχει **Reliability μικρότερο του 8**.

Κανόνας 3 : Ο τρίτος κανόνας εξετάζει την δραστηριότητα μιας ip. Σε αυτόν τον κανόνα θα επιτραπεί σε μια ip να περάσει μόνο εάν το **activity είναι μικρότερο του 9.**

Κανόνας 4 : Ο τέταρτος κανόνας εξετάζει και συνδυάζει τους προηγούμενες κανόνες μαζί. Κρίθηκε απαραίτητο η ύπαρξη ενός συνδυαστικού κανόνα για να αποφύγουμε την υπερβολική απόρριψη των εισερχομένων ip. Σε αυτόν τον

κανόνα επιτρέπουμε σε μια ip να περάσει αν ακολουθεί τα παρακάτω κριτήρια:

Risk μεγαλύτερο του 12

Activity μικρότερο του 6

Reliability μικρότερο του 3

Στην περίπτωση που δεν ικανοποιούνται όλα τα κριτήρια, η ip θα κρίνεται ακατάλληλη και θα κόβεται.

5.1.3 Τα σήματα

Τα σήματα που χρησιμοποίησα είναι είτε τύπου std_Logic είτε std_Logic_vector. Τα std_Logic είναι σήματα που αποτελούνται από 2 bits (0 ή 1) ενώ τα std_Logic_vector είναι σήματα που αποτελούνται με πάνω από 2 bits. Τα σήματα που χρησιμοποιήθηκαν και φαίνονται και στην Εικόνα 5.1 είναι τα εξής :

X : είναι ένα σήμα τύπου std_Logic_vector που αποτελείται από 32 bits και απεικονίζει την καινούρια εισερχόμενη ip.

Y : είναι ένα σήμα τύπου std_Logic_vector που αποτελείται από 32 bits και απεικονίζει την έξοδο του συστήματος (στην περίπτωση που ισχύουν οι κανόνες το Y παίρνει την τιμή του X)

Rfdin : είναι ένα σήμα τύπου std_Logic_vector που αποτελείται από 44 bits και είναι το IP reputation. Το Rfdin είναι οι τιμές που θα γεμίσουν τον καταχωρητή.

Wrfa : είναι ένα σήμα τύπου std_Logic. Όταν το wrfa γίνεται 1 τότε μπορούμε να γράψουμε στον καταχωρητή.

Code : είναι ένα σήμα τύπου std_Logic_vector που αποτελείται από 4 bits. Το σήμα αυτό είναι υπεύθυνο για την επιλογή κανόνα που θα χρησιμοποιηθεί.

Enable : είναι ένα σήμα τύπου std_Logic που χρησιμοποιείτε από τον πολυπλέκτη και τους κανόνες.

Ra : είναι ένα σήμα τύπου std_Logic_vector που αποτελείται από 4 bits, χρησιμοποιείτε από τον καταχωρητή για να δείξει σε ποια θέση γράφει ή διαβάζει ο καταχωρητής.

Ra1 : είναι ένα σήμα τύπου std_Logic_vector που αποτελείται από 4 bits, χρησιμοποιείτε από τους κανόνες για να ανατρέξουν τον καταχωρητή.

Clk : είναι ένα σήμα τύπου std_Logic. Είναι χαρακτηριστικό σήμα καθώς όταν το clk γίνεται 1 τότε ξεκινάει μια διεργασία.

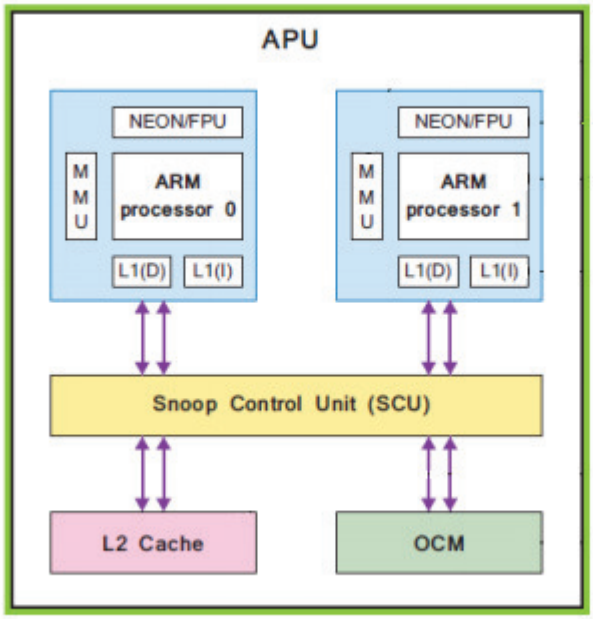
Rrfa : είναι ένα σήμα τύπου std_Logic. Όταν γίνεται 1 τότε το block κανόνες μπορούν να διαβάσουν από τον καταχωρητή.

Rfbout : είναι ένα σήμα τύπου std_Logic_vector που αποτελείται από 44 bits. Είναι έξοδος του καταχωρητή και είσοδος των κανόνων. Το rfbout και το rrfa είναι τα 2 σήματα που είναι υπεύθυνα για την επικοινωνία του Block κανόνες με το block καταχωρητής.

5.1.4 Ο Arm και το AXI Bus

Όπως φαίνεται στην Εικόνα 5.2 κάθε επεξεργαστής διαθέτει ένα ξεχωριστό 1ο επίπεδο Cache (data & instructions). Αυτό επιτρέπει την τοπική αποθήκευση των συχνά χρησιμοποιούμενων δεδομένων και εντολών έτσι ώστε να επιτυγχάνονται γρήγοροι χρόνοι πρόσβασης και βέλτιστη απόδοση του επεξεργαστή. Οι δύο πυρήνες μοιράζονται επιπλέον μία μεγαλύτερη 2ουεπιπέδου Cache των 512KB για εντολές και δεδομένα, και υπάρχει μία ακόμα on-chip μνήμη 256KB εντός της APU (Application Processing Unit). Ο πρωταρχικός ρόλος του MMU είναι να μεταφράζει τις φυσικές σε εικονικές διευθύνσεις μνήμης και το αντίστροφο. Η μηχανή NEON παρέχει εντολές τύπου Single Instruction Multiple Data (SIMD), επιταχύνοντας την προσπέλαση των περιφερειακών και των αλγορίθμων τύπου DSP.

Η επικοινωνία μεταξύ του ARM και του δικού μας περιφερειακού γίνεται μέσω του Axi bus.



Εικόνα 5.2 Επεξεργαστής Arm

5.2 Σχεδιασμός

Έχοντας δει και αναλύσει πλέον κάθε block του συστήματός μας είναι ώρα να δούμε τον κώδικα σε VHDL που αναπτύχθηκε για κάθε ένα από τα Block ξεχωριστά. Επίσης θα δούμε και τον κώδικα που αναπτύχθηκε σε γλώσσα προγραμματισμού C για τον προγραμματισμό του ARM.

5.2.1 Ο Κώδικας του Καταχωρητή

Ο καταχωρητής, όπως έχουμε αναφέρει και νωρίτερα, έχει δύο λειτουργίες, αυτή του γράψε και αυτή του διάβασε.

Στην Εικόνα 5.3 βλέπουμε τον κώδικα σε VHDL για την διεργασία του καταχωρητή γράψε. Τα σήματα reg0 έως reg15 υποδηλώνουν την μνήμη, τον χώρο δηλαδή που αποθηκεύονται τα IP Reputation. Το σήμα Ra είναι δείκτης για να ανατρέχουμε σε οποιαδήποτε θέση θέλουμε. Με την χρήση της εντολής Case, ανάλογα με την τιμή του Ra γράφουμε στον καταχωρητή.

Αντίστοιχα στην Εικόνα 5.4 βλέπουμε τον κώδικα για την διεργασία διάβασε. Με την ίδια λογική, χρησιμοποιώντας πάλι το σήμα Ra, ανατρέχουμε σε οποία θέση θέλουμε και διαβάζουμε από τον καταχωρητή.

```
38 SIGNAL reg0,reg1, reg2, reg3,reg4, reg5, reg6, reg7,reg8,reg9,reg10,reg11,reg12,reg13,reg14,reg15:
39 std_logic_vector(43 DOWNTO 0);
40
41 BEGIN
42
43
44 WRITE: PROCESS (clk, wRFA,reg0,reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8,reg9,reg10,reg11,reg12,reg13,reg14,reg15)
45 BEGIN
46
47 IF clk 'EVENT AND clk= '1' THEN
48 IF wRFA = '1' THEN
49 CASE Ra IS
50 WHEN "0000" => reg0 <= RFdin;
51 WHEN "0001" => reg1 <= RFdin;
52 WHEN "0010" => reg2 <= RFdin;
53 WHEN "0011" => reg3 <= RFdin;
54 WHEN "0100" => reg4 <= RFdin;
55 WHEN "0101" => reg5 <= RFdin;
56 WHEN "0110" => reg6 <= RFdin;
57 WHEN "0111" => reg7 <= RFdin;
58 WHEN "1000" => reg8 <= RFdin;
59 WHEN "1001" => reg9 <= RFdin;
60 WHEN "1010" => reg10 <= RFdin;
61 WHEN "1011" => reg11 <= RFdin;
62 WHEN "1100" => reg12 <= RFdin;
63 WHEN "1101" => reg13 <= RFdin;
64 WHEN "1110" => reg14 <= RFdin;
65 WHEN "1111" => reg15 <= RFdin;
66 WHEN OTHERS => NULL;
67 END CASE;
68 END IF;
69 END IF;
```

Εικόνα 5.3 Διεργασία Γράψε

```
75
76 READ: PROCESS (rRFA,Ra,reg0,reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8,reg9,reg10,reg11,reg12,reg13,reg14,reg15)
77 BEGIN
78 IF rRFA = '1' THEN
79 CASE Ra IS
80 WHEN "0000" => Rfbout <= reg0;
81 WHEN "0001" => Rfbout <= reg1;
82 WHEN "0010" => Rfbout <= reg2;
83 WHEN "0011" => Rfbout <= reg3;
84 WHEN "0100" => Rfbout <= reg4;
85 WHEN "0101" => Rfbout <= reg5;
86 WHEN "0110" => Rfbout <= reg6;
87 WHEN "0111" => Rfbout <= reg7;
88 WHEN "1000" => Rfbout <= reg8;
89 WHEN "1001" => Rfbout <= reg9;
90 WHEN "1010" => Rfbout <= reg10;
91 WHEN "1011" => Rfbout <= reg11;
92 WHEN "1100" => Rfbout <= reg12;
93 WHEN "1101" => Rfbout <= reg13;
94 WHEN "1110" => Rfbout <= reg14;
95 WHEN "1111" => Rfbout <= reg15;
96 WHEN OTHERS => Rfbout <= "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
97 END CASE;
98 END IF;
```

Εικόνα 5.4 Διεργασία Διάβασε

5.2.2 Ο Κώδικας των Κανόνων

Οι κανόνες θέτονται σε λειτουργία όταν μία καινούργια εισερχόμενη ip έρχεται στο σύστημά μας. Στην γραμμή 75 της Εικόνας 5.5 βλέπουμε ακριβώς αυτό.

Από την γραμμή 77 ως την γραμμή 84 είναι ο πρώτος κανόνας, και επιτρέπει να περάσει η ip μόνο αν το risk είναι μικρότερο του 9,σε διαφορετική περίπτωση στην έξοδο **y** θα βλέπουμε την τιμή 000..1111.

Ο δεύτερος κανόνας ξεκινάει στην γραμμή 87 και τελειώνει στην 94. Σε αυτόν τον κανόνα η ip επιτρέπεται να περάσει όταν το Reliability είναι μικρότερο του 8, σε διαφορετική περίπτωση στην έξοδο **y** θα βλέπουμε την τιμή 000..1001.

Ο τρίτος κανόνας ξεκινάει στην γραμμή 96 και τελειώνει στην 103 και επιτρέπει στην ip να περάσει μόνο αν το Activity είναι μικρότερο του 8, σε διαφορετική περίπτωση στην έξοδο **y** θα βλέπουμε την τιμή 000..1010.

Τέλος έχουμε έναν επιπλέον συνδυαστικό κανόνα που επιτρέπει στην ip να περάσει μόνο αν το Risk είναι μεγαλύτερο του 12, το activity μικρότερο του 6 και το Reliability μικρότερο του 3, σε διαφορετική περίπτωση στην έξοδο **y** θα βλέπουμε την τιμή 000..1011. Η διαφοροποίηση στην τιμή εξόδου, όταν ο κανόνας δεν ισχύει έγινε για δική μας ευκολία, ώστε να μπορούμε να καταλαβαίνουμε από ποιον κανόνα δεν πέρασε η ip.

```
75 if x=ip then
76
77 if code="00" then
78     if risk<"1001" then -- kanones
79         temp_y<=x;
80    elsif risk>"1001" then
81        temp_y<="00000000000000000000000000001111";
82     else
83     temp_y<="00000000000000000000000000001111";
84     end if;
85
86 --when"01"=>
87 elsif code="01" then
88    if reliability< "1000" then
89    temp_y<=x;
90    elsif reliability>"1000" then
91    temp_y<="00000000000000000000000000001001";
92     else
93     temp_y<="00000000000000000000000000001001";
94     end if;
95 --when "10"=>
96 elsif code="10" then
97    if activity<"1000" then
98    temp_y<=x;
99    elsif activity>"1000" then
100    temp_y<="00000000000000000000000000001010";
101    else
102    temp_y<="00000000000000000000000000001010";
103   end if;
104
105 elsif code="11" then
106   if risk>"1100" and activity<"0110" and reliability<"0011" then
107   temp_y<=x;
108    else
109    temp_y<="00000000000000000000000000001011";
110   end if;
111
```

Εικόνα 5.5 Κανόνες

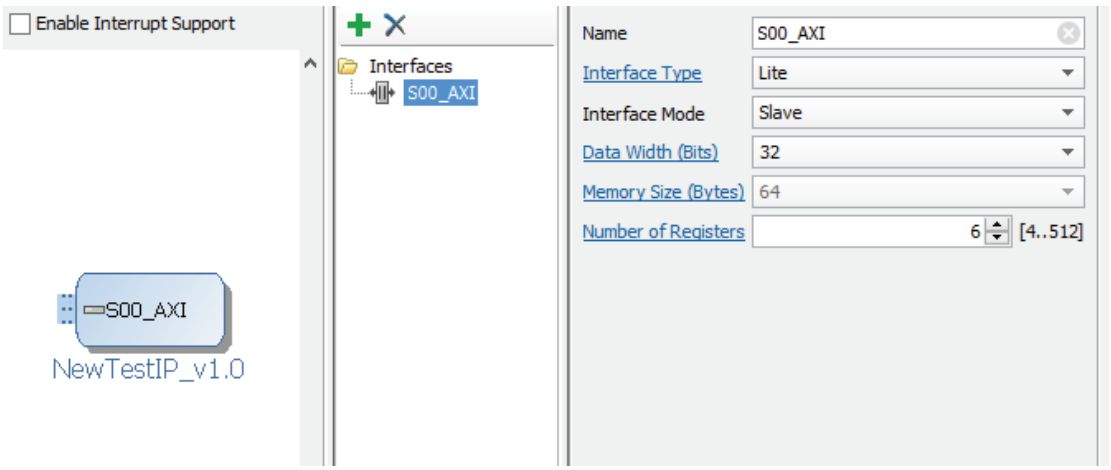
5.2.3 Δημιουργία Axi Bus και ενσωμάτωση του περιφερειακού

Έχοντας τελειώσει και με τους κανόνες, πλέον μένει να συνδέσουμε το σύστημά μας στον επεξεργαστή Arm. Για να μπορέσουμε να το κάνουμε αυτό θα χρειαστούμε έναν τρόπο διασύνδεσης. Το Vivado μας παρέχει την δυνατότητα να δημιουργήσουμε το δικό μας Axi IP block.

5.2.3.1 Δημιουργία Axi Bus Περιφερειακού

Στο βασικό μενού του Vivado (Εικόνα 4.7) στο πάνω μέρος, στην επιλογή **tools,** υπάρχει η επιλογή **create and package ip...** Επιλέγοντας αυτήν την επιλογή, μια νέα σελίδα θα ανοίξει, και επιλέγουμε την επιλογή **Create a new axi4 peripheral.** Στην νέα σελίδα που θα εμφανιστεί βάζουμε το όνομα που επιθυμούμε. Στην νέα σελίδα που θα εμφανιστεί, πατώντας επόμενο, μπορούμε να τροποποιήσουμε τη διεπαφή του Axi bus. Οι αρχικές ρυθμίσεις εδώ μας έδιναν 4 καταχωρητές, για τις δικές μας ανάγκες το αλλάξαμε αυτό

και τους κάναμε 6. Η Διεπαφή μας λοιπόν αποτελείται από 6 καταχωρητές με 32 bit πλάτος ο κάθε ένας (Εικόνα 5.6). Αναφέρουμε πως οι καταχωρητές αυτοί θα χρησιμοποιηθούν για να τροφοδοτήσουμε το σύστημά μας με τα απαραίτητα σήματα. Ενδεικτικά, ένα από αυτά θα είναι η καινούργια εισερχόμενη ip, δηλαδή το **X**. Πατώντας επόμενο επιλέγουμε **Edit IP,** σε αυτό το σημείο το axi bus έχει δημιουργηθεί.



Εικόνα 5.6 Διεπαφή Axi bus

5.2.3.2 Ενσωμάτωση του περιφερειακού

. Τελειώνοντας με την δημιουργία του axi bus, επόμενο βήμα είναι να προσθέσουμε τον δικό μας κώδικα στο περιφερειακό που μόλις δημιουργήσαμε. Ο wizard του Vivado μας βοηθάει σε αυτό. Στο μενού αριστερά (Εικόνα 4.7) στο **Project Manager** επιλέγουμε την επιλογή **Add sources** και προσθέτουμε την δική μας αρχιτεκτονική στο axi bus που δημιουργήσαμε. Παρόλο που προσθέσαμε την δική μας αρχιτεκτονική, ακόμα δεν την έχουμε συνδέσει με τους καταχωρητές του axi bus.

5.2.4 Σύνδεση του περιφερειακού με το Axi bus

Με την δημιουργία του axi bus το Vivado δημιούργησε και τον κώδικά του σε VHDL. Σε αυτόν τον κώδικα πρέπει να προσθέσουμε και το δικό μας περιφερειακό και να το συνδέσουμε στους καταχωρητές του axi bus. Η σύνδεση επιτυγχάνεται με την δήλωση του δικού μας component μέσα στην αρχιτεκτονική του axi bus (Εικόνα 5.7) και με την τεχνική που χρησιμοποιεί η VHDL σε αυτές τις περιπτώσεις που ονομάζεται **PORT MAP** (Εικόνα 5.8). Από την γραμμή 444 της Εικόνας 5.8 και μετά φαίνονται οι αντιστοιχίες που έχουν γίνει μεταξύ του δικού μας περιφερειακού με τους καταχωρητές του axi bus.Τα σήματα **slv_reg** αναφέρονται στους καταχωρητές του Axi bus, τα οποία τα συνδέομαι με τα δικά μας σήματα. Να θυμίσουμε σε αυτό το σημείο ότι το Rfdin είναι 44 bits ενώ οι καταχωρητές του axi bus έχουν πλάτος 32

bits. Αυτό μας δημιούργησε πρόβλημα και την αντιμετώπισή του θα τι δούμε σε παρακάτω κεφάλαιο.

```
125
126
127     component test is
128     PORT(
129     clk:in std_logic;
130     x:in std_logic_vector(31 downto 0);
131     y:out std_logic_vector(31 downto 0);
132     rfdin:in std_logic_vector(43 downto 0);
133     wrfa:in std_logic;
134     code:in std_logic_vector(1 downto 0);
135     enable:in std_logic;
136     ra2:in std_logic_vector(3 downto 0));
137     end component;
```

Εικόνα 5.7 Δήλωση του component

```
438 temp_rfdin(43 downto 12)<=slv_reg2;
439 temp_rfdin(11 downto 0)<=slv_reg3(31 downto 20);
440
441     -- Add user logic here
442 test0:test
443                 port map(
444                 clk=>S_AXI_ACLK,
445                 x=>slv_reg0,
446                 y=>y_out,-------------slv_reg1
447                 rfdin=>temp_rfdin,--concatanate
448                 wrfa=>slv_reg4(0),
449                 code=>slv_reg5(31 downto 30),
450                 enable=>slv_reg4(1),
451                 ra2=>slv_reg6(3 downto 0));
452     -- User logic ends
```

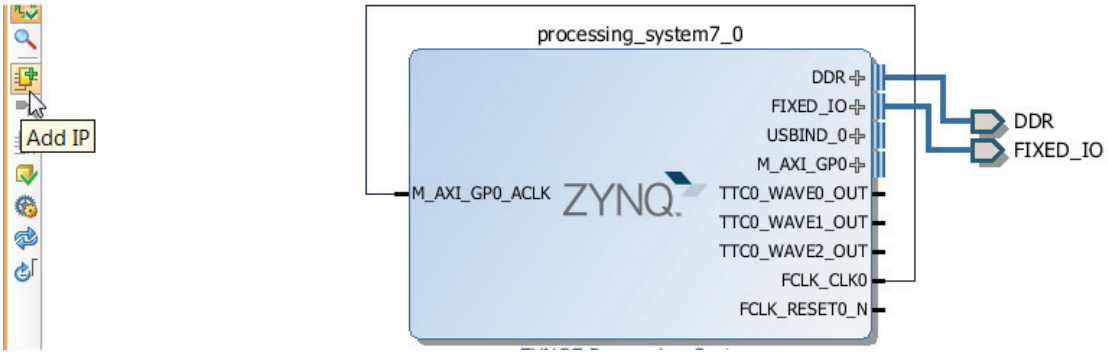
Εικόνα 5.8 Σύνδεση με τους καταχωρητές του axi bus

Αναλύοντας την Εικόνα 5.8 βλέπουμε πλέον ότι το περιφερειακό μας είναι συνδεμένο με το axi bus. Το x θα παίρνει τιμές από τον slv_reg0, η έξοδος y είναι συνδεμένη με τον slv_reg1, το rfdin είναι συνδεμένο με τους slv_reg2 και slv_reg3, το wrfa είναι συνδεμένο με τον slv_reg4 και μάλιστα με το πρώτο του bit, το code είναι συνδεμένο με τον slv_reg5, το enable με τον slv_reg4 αυτή την φορά με το δεύτερό του bit, και τέλος το Ra2 είναι συνδεμένο με τον slv_reg6.

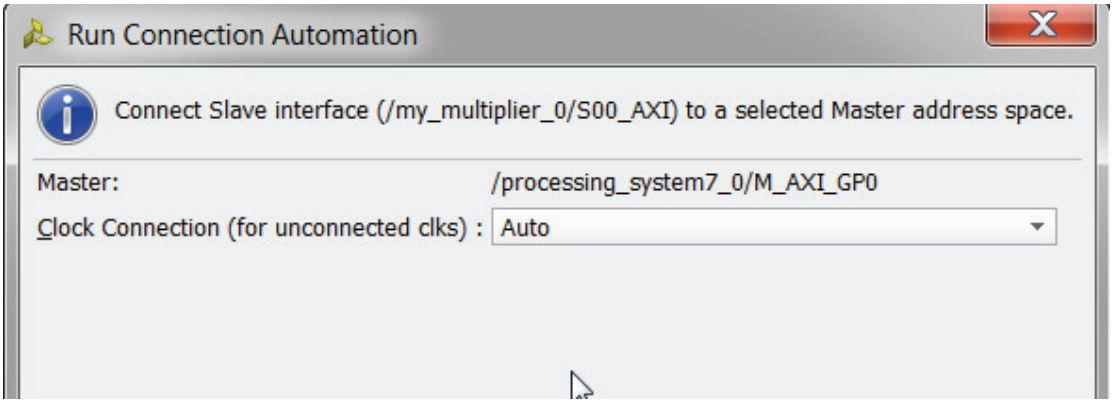
5.2.5 Χρονισμός ρολογιών Συστήματος Και block διάγραμμα

Ανακεφαλαιώνοντας, αναφέρεται πώς σε αυτό το σημείο έχουμε δημιουργήσει το περιφερειακό μας (τους κανόνες δηλαδή και το καταχωρητή) και το axi bus (που από την μία μεριά συνδέεται στον Arm και στην άλλη στο περιφερειακό

μας). Σε αυτή τη φάση της σχεδίασης θα συγχρονίσουμε όλα τα ρολόγια του συστήματός μας. Ο wizard του Vivado μας διευκολύνει και πάλι, με μία σειρά από αυτοματοποιημένες ρυθμίσεις. Στο μενού αριστερά (Εικόνα 5.9) επιλέγουμε add IP και προσθέτουμε το δικό μας περιφερειακό στο διάγραμμα. Μόλις το προσθέσουμε στο διάγραμμα ένα καινούργιο παράθυρο ανοίγει που μας επιτρέπει να συγχρονίσουμε τα ρολόγια αυτόματα (Εικόνα 5.10).

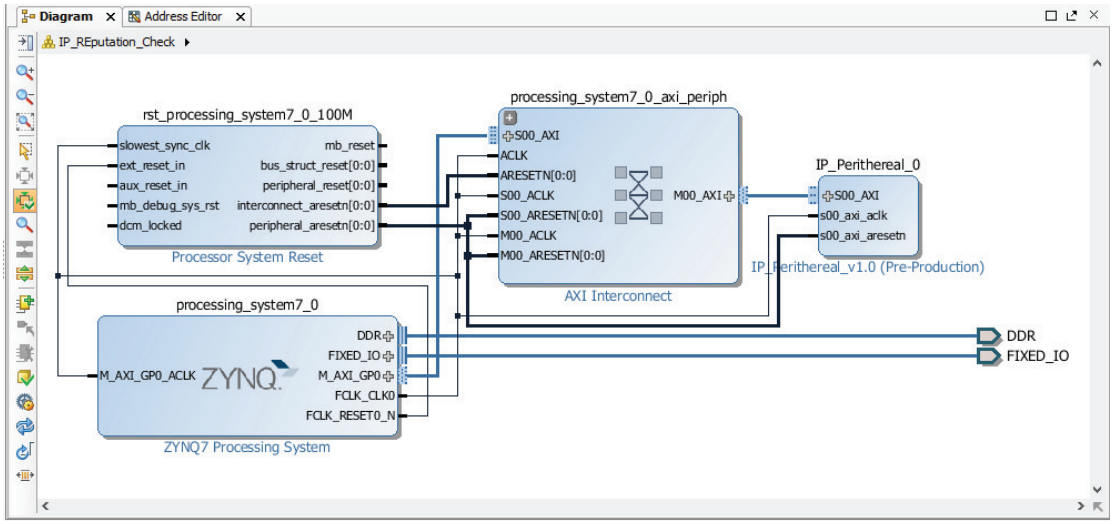


Εικόνα 5.9 Προσθήκη του περιφερειακού στο διάγραμμα



Εικόνα 5.10 Συγχρονισμός ρολογιών

Μόλις τελειώσει και ο χρονισμός των ρολογιών το Vivado δημιουργεί ένα νέο block διάγραμμα, που περιλαμβάνει πλέον όλα τα κομμάτια του συστήματος. Το διάγραμμα που φαίνεται στην Εικόνα 5.11 αποτελείται από τον επεξεργαστή ARM (**ZYNQ 7 processing System**), από τον **Processor System Reset** που μηδενίζει και αρχικοποιεί τις τιμές του Arm, από το **Axi Interconnect** που είναι το axi bus και τέλος από το IP_Peripheral που είναι το δικό μας περιφερειακό. Είναι σαφές πλέον πως το σύστημά μας τροφοδοτείτε από τον Arm μέσω του Axi bus.



Εικόνα 5.11 Block διάγραμμα

Το σύστημά μας είναι πλέον συνδεδεμένο και χρονισμένο με τον επεξεργαστή Arm. Επόμενο βήμα είναι εξάγουμε τον σχεδιασμό hardware στο SDK και τέλος να το κατεβάσουμε στο ZedBoard.

5.3 Εξαγωγή του Hardware στο SDK

Για να εξάγουμε το hardware στο Sdk απαραίτητη προϋπόθεση είναι παράγουμε το Bitstream. Στο μενού αριστερά στο Vivado επιλέγουμε την επιλογή Generate Bitstream. Μόλις το Bitstream δημιουργηθεί, από το File μενού επιλέγουμε Export hardware και launch Sdk. Το sdk ανοίγει και πλέον είμαστε έτοιμοι να γράψουμε τον κώδικα σε γλώσσα C για τον επεξεργαστή ARM.

5.4 Προγραμματισμός του ARM σε C

Σε αυτό το στάδιο είμαστε έτοιμοι να προγραμματίσουμε τον ARM για να τροφοδοτήσει το σύστημά μας με τα σήματα που θέλουμε. Υπενθυμίζουμε πως τα σήματα που θα τροφοδοτούνται από τον Arm είναι το X, το Rfdin το enable, το wrfa,το code και το Ra. Αναπτύχθηκε κώδικας ώστε να γεμίζουμε τον καταχωρητή με τιμές, έπειτα να εισάγουμε μια ip και να επιλέγουμε με ποιόν κανόνα θέλουμε να ελέγχουμε την φήμη αυτής της ip.

```
void clearReg(void)
{
    Xuint32 *baseaddr_p = (Xuint32 *)XPAR_IP_PERITHEREAL_0_S00_AXI_BASEADDR  ;
    int i;
    for (i==0;i<16;i++){
        *(baseaddr_p+2) =0;// τιμη για το rfdin τα πρωτα 32 bits
        *(baseaddr_p+3)=0;           //στο low part δωσε 12 νουμερα για το rfdin
        *(baseaddr_p+6)=i;         //Ra
        *(baseaddr_p+4)=1;//wrfa,enable
    }


    *(baseaddr_p+4)=0; // στο τελος της αναζητησης μηδενιζει το enable
}
```

Εικόνα 5.12 Γέμισμα του καταχωρητή και τιμές για το Ra,Wrfa,enable

Στο τμήμα κώδικα που φαίνεται στην Εικόνα 5.12 βλέπουμε πώς γεμίζουμε τον καταχωρητή μας με την βοήθεια της for. Σε κάθε κύκλο το i δηλαδή το Ra αυξάνετε κατά ένα, και σε κάθε θέση που πηγαίνει δίνουμε τιμές για το Rfdin.Το Rfdin παίρνει τιμές από το τμήμα του κώδικα που βλέπουμε στην Εικόνα 5.13.

```
void ReputationInput(Xuint32 ra,Xuint32 IP_Addr,Xuint32 Reputation)
{
    Xuint32 *baseaddr_p = (Xuint32 *)XPAR_IP_PERITHEREAL_0_S00_AXI_BASEADDR  ;
    *(baseaddr_p+2) =IP_Addr;// τιμη για το rfdin τα πρωτα 32 bits
    *(baseaddr_p+3)=Reputation;           //στο low part δωσε 12 νουμερα για το rfdin
    *(baseaddr_p+6)=ra;         //Ra
    *(baseaddr_p+4)=1;//wrfa,enable

    //*(baseaddr_p+4)=0; // στο τελος της αναζητησης μηδενιζει το enable
}
```

Εικόνα 5.13 Τιμές για το rfdin

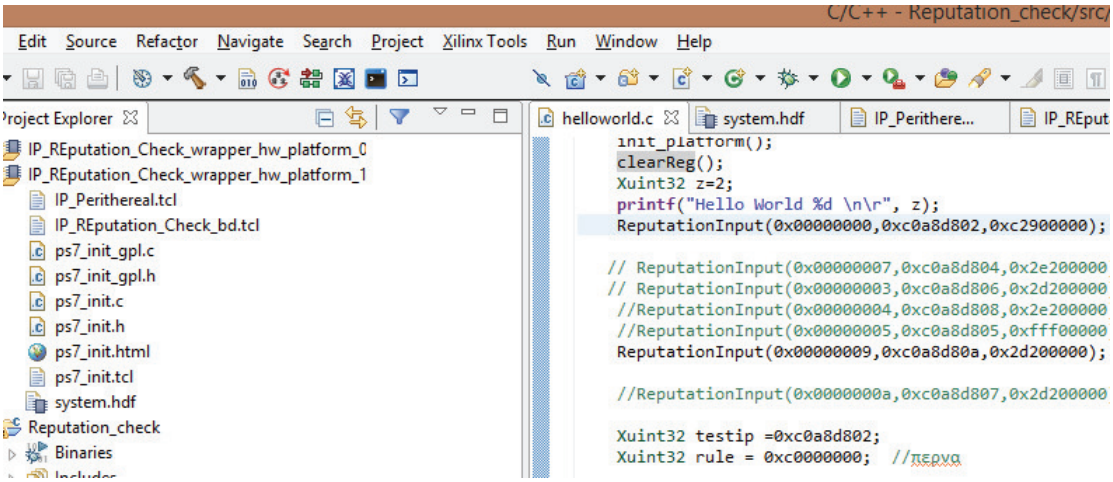
```
Xuint32 testip =0xc0a8d802;
Xuint32 rule = 0xc0000000;  //περνα

z=IPinput(testip,rule);
if (z==0x00000000){
     printf("package with ip : 0x%08x not passed (malicious) \n\r", testip);
}
else if (z==testip) {
    printf("package with ip : 0x%08x passed successfully (non malicious) \n\r", testip);
}
else
{printf("problem \n");}
```

Εικόνα 5.14 Έλεγχος Ip

Στο τμήμα κώδικα της Εικόνας 5.14 η μεταβλητή **testip** απεικονίζει την εισερχόμενη ip και η μεταβλητή **rule** απεικονίζει ποιο κανόνα θέλουμε να χρησιμοποιήσουμε για να εξετάσουμε την φήμη αυτής της ip. Ακολουθούν 2 **printf** που θα μας εμφανίζουν το ανάλογο μήνυμα ανάλογα με το αν η ip πέρασε ή κόπηκε.

5.5 Κατέβασμα στο ZedBoard

Μετά το τέλος του προγραμματισμού του Arm, είμαστε έτοιμοι να τα κατεβάσουμε όλα αυτά στο ZedBoard. Αρχικά συνδέουμε το ZedBoard στον υπολογιστή μας, έπειτα στο παράθυρο του sdk (Εικόνα 5.14) στο μενού επιλέγουμε Xilinx tools και program FPGA. Το Bitstream που είχαμε δημιουργήσει πριν θα φορτωθεί στον ARM και στο μενού του sdk πατάμε Run. Αυτό είναι και το τελευταίο βήμα της δημιουργίας του συστήματός μας. Μένει τώρα να κάνουμε εξομοίωσες για να σιγουρευτούμε για την ορθή λειτουργία του. `



Εικόνα 5.14 SDK

5.6 Εξομοιώσεις Συστήματος

Σε αυτή την παράγραφο θα κάνουμε 4 διαφορετικές εξομοιώσεις για να επαληθεύσουμε την ορθή λειτουργία του συστήματός μας. Η εξομοίωση που θα πραγματοποιήσουμε περιλαμβάνει το γέμισμα του καταχωρητή, την είσοδο μίας καινούργιας ip και τέλος να δούμε αν αυτή η ip θα επιτραπεί να περάσει η όχι. Θα γίνει μία εξομοίωση για κάθε κανόνα που έχουμε ορίσει. Ο τρόπος που εξομοιώνουμε στο Vivado φαίνεται στο Κεφάλαιο 4 παράγραφος 3.1.

5.6.1 Εξομοίωση Κανόνα 1

Η πρώτη μας εξομοίωση θα είναι για τον πρώτο κανόνα. Θυμίζουμε ότι,ο πρώτος κανόνας επιτρέπει σε μια ip να περάσει μόνο αν το risk αυτής της ip είναι μικρότερο του 9. Παρακάτω βλέπουμε τις τιμές που δώσαμε σαν είσοδο.

X : 11111111111111111111111111111111 (32 bits)

Rfdin : 11111111111111111111111111111111000010001111 (44 Bits)

Ra: 0000

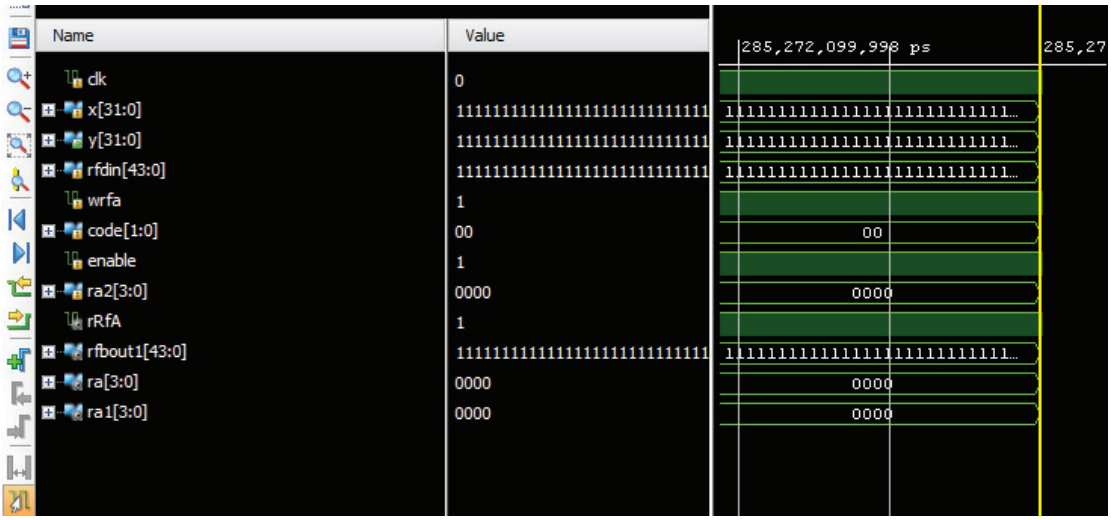
Enable: 1

Wrfa: 1

Clk: 1

Code: 00 (στο 00 αντιστοιχεί ο πρώτος κανόνας)

Παρατηρούμε πως στο Risk δώσαμε τιμή μικρότερη του 9 (1 0 0 0=8). Άρα με βάση τον πρώτο κανόνα στην έξοδο **y** πρέπει να δούμε το **x**.



Εικόνα 5.15 Εξομοίωση κανόνα 1

Στην εξομοίωση της Εικόνας 5.15, στο αριστερό μέρος βλέπουμε την ονομασία των σημάτων εισόδου και εξόδου, στο μέσο της Εικόνας βλέπουμε τις τιμές που παίρνουν τα σήματα από την στιγμή που το σύστημά μας ενεργοποιήθηκε και δεξιά βλέπουμε τις τιμές που έχουν τα σήματά μας ακριβώς την χρονική στιγμή που αναγράφεται στο πάνω μέρος. Παρατηρούμε πως στην έξοδο **y** η τιμή είναι ίδια με αυτή της εισόδου **x**, πράγμα που μας επιβεβαιώνει τις προβλέψεις μας.

5.6.2 Εξομοίωση Κανόνα 2

Συνεχίζουμε με τον κανόνα 2. Υπενθυμίζουμε πως ο κανόνας 2 επιτρέπει σε μια ip να περάσει μόνο αν αυτή η ip έχει Reliability μικρότερο του 8. Σε αντίθετη περίπτωση στην έξοδο **y** θα πάρουμε την τιμή 00000000000000000000000000001001.Παρακάτω βλέπουμε τις τιμές που βάλαμε για είσοδο του συστήματος.

X : 11111111111111110000000000000000 (32 bits)

Rfdin : 11111111111111110000000000000000000001111100000000 (44 Bits)

Ra: 0001

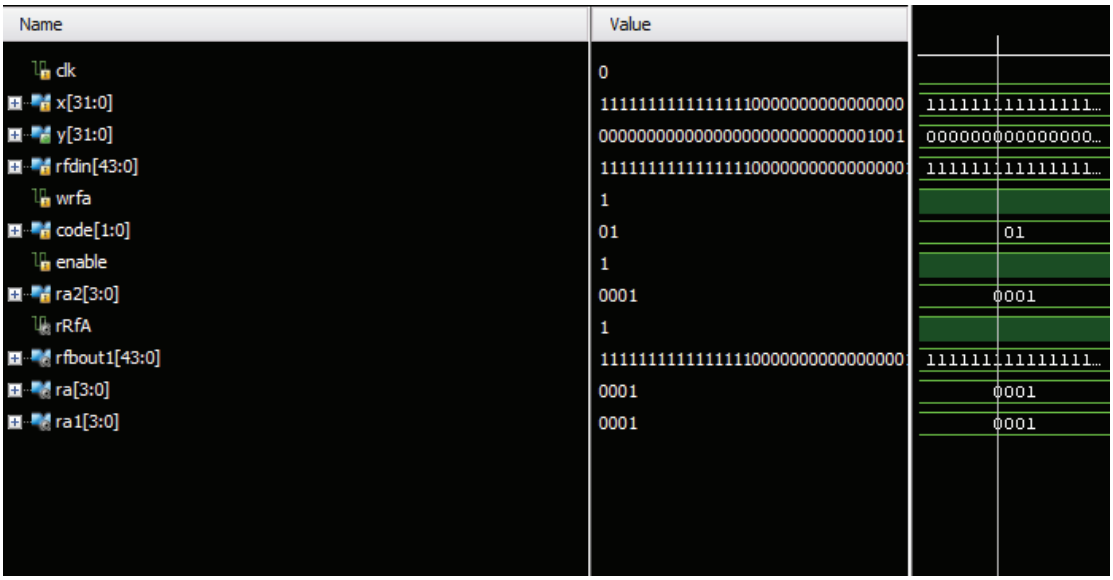
Enable: 1

Wrfa: 1

Clk: 1

Code: 01 (στο 01 αντιστοιχεί ο δεύτερος κανόνας)

Βάση αυτών τον τιμών η ip δεν πρέπει να περάσει.



Εικόνα 5.16 Εξομοίωση κανόνα 2

Στην εξομοίωση της Εικόνας 5.15, παρατηρούμε πως η έξοδος **y** έχει πάρει διαφορετική τιμή από αυτή της εισόδου **x**. Μάλιστα παρατηρούμε πως το **y** έχει πάρει την τιμή 00000000000000000000000000**1001**. Όπως είπαμε και στην παράγραφο 5.2.2 αυτό υποδηλώνει πως η ip δεν πέρασε και ότι κόπηκε από τον κανόνα 2.

5.6.3 Εξομοίωση Κανόνα 3

Για τον κανόνα 3 το σύστημα επιτρέπει σε μια ip να περάσει μόνο εάν αυτή έχει activity μικρότερο του 8. Παρακάτω βλέπουμε τις τιμές που βάλαμε σε αυτή την περίπτωση για είσοδο του συστήματος.

X : 11111111111111111111111111111111 (32 bits)

Rfdin : 11111111111111111111111111111111111111110011 (44 Bits)

Ra: 0010

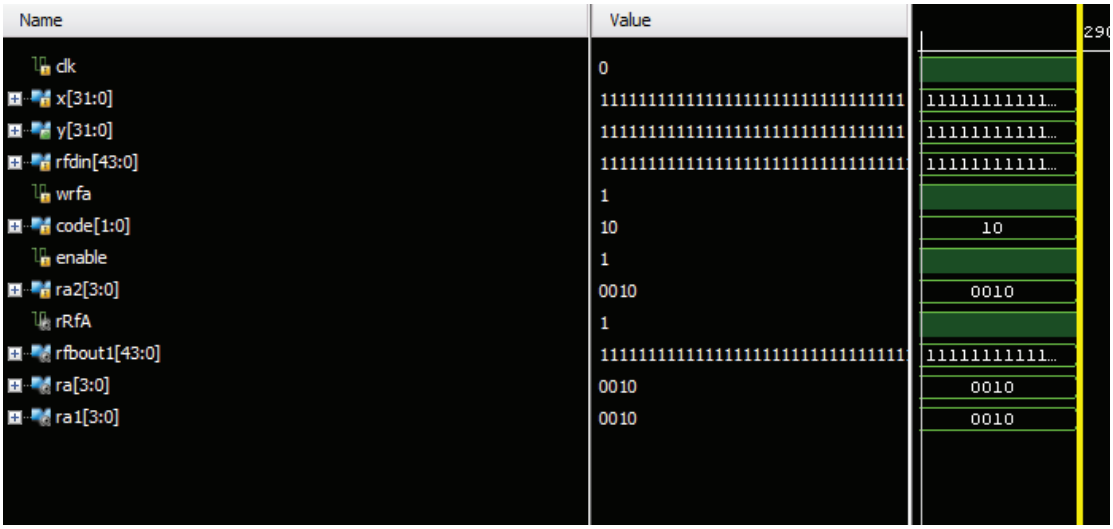
Enable: 1

Wrfa: 1

Clk: 1

Code: 10 (στο 10 αντιστοιχεί ο τρίτος κανόνας)

Για Activity βάλαμε 3 (0011), που είναι στις επιτρεπόμενες τιμές για τον κανόνα 3,άρα στην έξοδο **y** θα πρέπει να δούμε το **x**.



Εικόνα 5.17 Εξομοίωση κανόνα 3

Στην εξομοίωση της Εικόνας 5.17 παρατηρούμε πως στην έξοδο **y** έχουμε την τιμή της εισόδου **x**. Επιβεβαιώνετε δηλαδή η πρόβλεψή μας ότι η ip 11111111111111111111111111111111 με τον κανόνα 3 θα επιτραπεί να περάσει.

5.6.4 Εξομοίωση Κανόνα 4

Ο τελευταίος κανόνας είναι συνδυαστικός. Συνδυάζει τους προηγούμενους 3 κανόνες σε έναν. Επιτρέπει σε μια ip να περάσει μόνο αν το risk αυτής της ip είναι μεγαλύτερο του 12, το activity μικρότερο του 6 και το Reliability μικρότερο του 3. Παρακάτω βλέπουμε τις τιμές που βάλαμε για είσοδο του συστήματος.

X : 10101010101010101010101010101010 (32 bits)

Rfdin : 10101010101010101010101010101010000011110010 (44 Bits)

Ra: 1111

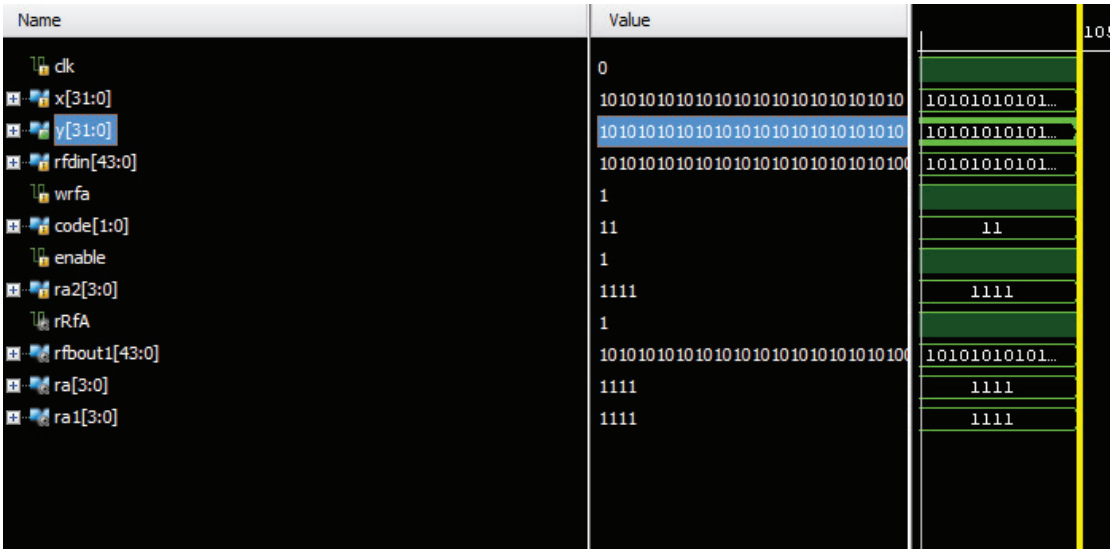
Enable: 1

Wrfa: 1

Clk: 1

Code: 11 (στο 11 αντιστοιχεί ο τέταρτος κανόνας)

Στην εξομοίωση αυτή βάλαμε για reliability 0, για risk 16 και για Activity 3. Βάση αυτόν των τιμών η ip πρέπει να περάσει.



Εικόνα 5.18 Εξομοίωση κανόνα 4

Στην Εικόνα 5.18 παρατηρούμε πως στην έξοδο **y** έχουμε την τιμή της εισόδου **x**, που σημαίνει πως η ip 10101010101010101010101010101010 με τον κανόνα 3 πέρασε.

6. Προβλήματα

Κατά την ανάπτυξη του συστήματός μας ήρθαμε αντιμέτωποι με αρκετά προβλήματα. Για την αντιμετώπιση αυτών τον προβλημάτων χρειάστηκε να εμβαθύνουμε τις γνώσεις μας στην VHDL αλλά και σκεφτούμε τρόπους διαφορετικής προσέγγισης του προβλήματος.

6.1 Πρόβλημα Πλάτους Καταχωρητών

Το βασικότερο πρόβλημα αντιμετωπίστηκε όταν χρειάστηκε να ενώσουμε το δικό μας περιφερειακό με το Axi bus. Θυμίζουμε ότι ο Arm περνάει τις τιμές στους καταχωρητές του axi bus και το axi bus τροφοδοτεί το σύστημά μας. Ο καταχωρητής του συστήματός μας έχει πλάτος 44 bits. Οι καταχωρητές του axi bus έχουν 32 bits πλάτος. Αυτό αναπόφευκτα δημιουργεί πρόβλημα. Για την αντιμετώπιση του προβλήματος χρειάστηκε να χρησιμοποιήσουμε δύο από τους καταχωρητές του axi bus και να τους ενώσουμε.

```
431             -- Read address mux
432               axi_rdata <= reg_data_out;     -- register read data
433           end if;
434         end if;
435       end if;
436     end process;
437
438 temp_rfdin(43 downto 12)<=slv_reg2;
439 temp_rfdin(11 downto 0)<=slv_reg3(31 downto 20);
440
441     -- Add user logic here
442 test0:test
443                 port map(
444                 clk=>S_AXI_ACLK,
445                 x=>slv_reg0,
446                 y=>y_out,-------------slv_reg1
447                 rfdin=>temp_rfdin,--concatanate
448                 wrfa=>slv_reg4(0),
449                 code=>slv_reg5(31 downto 30),
450                 enable=>slv_reg4(1),
451                 ra2=>slv_reg6(3 downto 0));
452     -- User logic ends
```

Εικόνα 6.1 Ένωση δύο καταχωρητών σε ένα σήμα

Στην Εικόνα 6.1 γραμμή 438 χρησιμοποιούμε το σήμα **temp_rfdin**. Το σήμα αυτό είναι ένα τοπικό σήμα που αποτελείται από 44 bits και χρησιμοποιείτε από τον axi bus. Στην δήλωση της γραμμής 438 το σήμα temp_rfdin τροφοδοτείτε τα πρώτα του 32 bits από τον καταχωρητή του axi bus **slv_reg2**, και τα υπόλοιπα 12 bits τα τροφοδοτείτε από τον καταχωρητή του axi bus **slv_reg3** (γραμμή 439). Έπειτα στην γραμμή 447 αναθέτουμε την τιμή του temp_rfdin στο rfdin που είναι και είσοδος για τον καταχωρητή μας.

Βιβλιογραφία

1. Xillinx.com, Zynq-700 All Programmable SoC Software DevelopersGuide, October 2012

2. Zedboard.com, Getting Started Guide

3. Xillybus.com, Getting started with Xillybus on a Windows host, Ver- sion 2.0

4. Xillybus.com, Xillybus host application programming guide for Win- ndows

5. Xillinx.com, Zedboard: Zynq-7000 AP SoC Concepts, Tools and Tech- niques, A hand -On Guide to E ective Embedded System Design, Ver- sion 14.3

6. Zedboard.com, Zynq Evaluation and Development Hardware User'sGuide,January 2013,Version 1.9

Ηλεκτρονικές Πηγές

1. http://en.wikipedia.org/wiki/VHDL
2. http://www.vhdl-online.de/
3. http://www.fpga4fun.com/
4. http://www.fpgajournal.com/
5. http://www.fpga-guide.com/startpage_frame.html
6. http://en.wikipedia.org/wiki/Field-programmable_gate_array
7. http://www.ashenden.com.au/designers-guide/VHDL-quick-start/ppframe.htm
8. http://www2.uic.edu/~wahmad1/quartus_ii_tutorial.pdf
9. http://esd.cs.ucr.edu/labs/tutorial/
10. http://www.d.umn.edu/~gshute/spimsal/talref.html
11. http://coe.uncc.edu/~amukherj/INTRO2VHDL/alu.eg1.pdf
12. http://users.eecs.northwestern.edu/~debjit/files/361_Project.pdf
13. http://www.fpgadeveloper.com/2014/08/creating-a-custom-ip-block-in-vivado.html

ΠΑΡΑΡΤΗΜΑ Α

Hardware Κώδικας

Component Test

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer: Panagiotis Bountas
--
-- Create Date: 19.10.2015 21:27:50
-- Design Name:
-- Module Name: test - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;
--use ieee.numeric_std.all;
```

```
entity test is
PORT(
clk:in std_logic;
x:in std_logic_vector(31 downto 0);
y:out std_logic_vector(31 downto 0);
rfdin:in std_logic_vector(43 downto 0);
wrfa:in std_logic;
code:in std_logic_vector(1 downto 0);
enable:in std_logic;
ra2:in std_logic_vector(3 downto 0));
--sel:in std_logic);
end test;

architecture Behavioral of test is
component plzzz is   ----------------------------component plzzz
 Port (
Ra1:out std_logic_vector(3 downto 0);
rRfA:out std_logic;
Rfbout:in std_logic_vector(43 downto 0);
x:in std_logic_vector(31 downto 0);
y:out std_logic_vector(31 downto 0);
clk:in std_logic;
enable:in std_logic;
code:in std_logic_vector(1 downto 0 ));

end component;


component Registerfile is -------------------Component RegisterFile
PORT(
clk:in std_logic;
Ra:in std_logic_vector(3 DOWNTO 0);
rRFA,wRFA:in std_logic;
RFdin:in std_logic_vector(43 DOWNTO 0);
```

```
rfbout:out std_logic_vector(43 DOWNTO 0));
end component;
--signal Ra:std_logic_vector(3 downto 0);
signal rRfA:std_logic;
signal rfbout1:std_logic_vector(43 downto 0);
signal ra:std_logic_vector(3 downto 0);
signal ra1:std_logic_vector(3 downto 0);
begin

multiplexer: process(clk,enable,ra2,ra1)
begin
if  enable= '1' then -------sto plzzzzzz
ra<=ra1;
elsif enable='0' then
    ra<=ra2;
 end if;
 end process;



u1:plzzz        port map(ra1,rRfa,rfbout1,x,y,clk,enable,code);
u2:RegisterFile port map (clk,Ra,rRFA,wRFA,rfdin,rfbout1);

end Behavioral;
```

Component Rules-

```
-- Company:
-- Engineer:
--
-- Create Date: 14.10.2015 21:20:33
-- Design Name:
-- Module Name: plzzz - Behavioral
-- Project Name:
-- Target Devices: ZedBoard
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;
use ieee.numeric_std.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
```

```
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity plzzz is
 Port (

Ra1:out std_logic_vector(3 downto 0);
rRfA:out std_logic;
Rfbout:in std_logic_vector(43 downto 0);
x:in std_logic_vector(31 downto 0);
y:out std_logic_vector(31 downto 0);
clk:in std_logic;
enable:in std_logic;
code:in std_logic_vector(1 downto 0 ));
end plzzz;

architecture Behavioral of plzzz is
signal reliability,risk,activity :std_logic_vector(3 downto 0);
signal ip, temp_y:std_logic_vector(31 downto 0);
signal rfbout1:std_logic_vector(43 downto 0);
--signal clk_enable:std_logic;
signal counter: integer;
signal temp_ra1:std_logic_vector(3 downto 0);
begin
--clk_enable<=clk and enable;
rfbout1<=Rfbout ;
test:process(clk,x,Rfbout1,enable,temp_y,code,reliability,risk,activity,counter,rRfa)
--Variable t : Integer range 0 to 15 := 0;
begin
if enable='1' then
if clk 'EVENT AND clk= '1' THEN

rRfa<='1';
```

```
if counter<16 then


ip <= rfbout1(43 downto 12 );
reliability <= rfbout1(11 downto 8);
risk<=rfbout1(7 downto 4);
activity<=rfbout1(3 downto 0);

if x=ip then

if code="00" then
    if risk<"1001" then -- kanones
        temp_y<=x;
   elsif risk>"1001" then
       temp_y<="00000000000000000000000000001111";
                                    else
                                    temp_y<="00000000000000000000000000001111";
    end if;

--when"01"=>
elsif code="01" then
   if reliability< "1000" then
   temp_y<=x;
   elsif reliability>"1000" then
   temp_y<="00000000000000000000000000001001";
                                    else
                                    temp_y<="00000000000000000000000000001001";
    end if;
--when "10"=>
elsif code="10" then
   if activity<"1000" then
   temp_y<=x;
   elsif activity>"1000" then
    temp_y<="00000000000000000000000000001010";
```

```
                                    else
                                    temp_y<="00000000000000000000000000001010";
  end if;

elsif code="11" then
  if risk>"1100" and activity<"0110" and reliability<"0011" then
  temp_y<=x;
                                    else
                                    temp_y<="00000000000000000000000000001011";
  end if;

else
   temp_y<="11111111111111111111111111111111";
end if;



end if;

counter<=counter+1;
  temp_ra1<=temp_ra1+1;
end if;
end if;
else
rRfa<='0';
counter<=0;
temp_ra1<="0000";
end if;
end process test;

y<=temp_y;
ra1<=temp_ra1;

End Behavioral ;
```

Component Register File

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:
-- Design Name:
-- Module Name: RegisterFile - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;


entity RegisterFile is
PORT(
clk:in std_logic;
Ra:in std_logic_vector(3 DOWNTO 0);
rRFA,wRFA: IN std_logic;
RFdin: IN std_logic_vector(43 DOWNTO 0);
```

```
Rfbout:OUT std_logic_vector(43 DOWNTO 0));
end RegisterFile;

architecture Behavioral of RegisterFile is
SIGNAL reg0,reg1, reg2, reg3,reg4, reg5, reg6, reg7,reg8,reg9,reg10,reg11,reg12,reg13,reg14,reg15:
std_logic_vector(43 DOWNTO 0);

BEGIN

WRITE: PROCESS (clk, wRFA,reg0,reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8,reg9,reg10,reg11,reg12,reg13,reg14,reg15)
BEGIN

IF clk 'EVENT AND clk= '1' THEN
IF wRFA = '1' THEN
CASE Ra IS
WHEN "0000" => reg0 <= RFdin;
WHEN "0001" => reg1 <= RFdin;
WHEN "0010" => reg2 <= RFdin;
WHEN "0011" => reg3 <= RFdin;
WHEN "0100" => reg4 <= RFdin;
WHEN "0101" => reg5 <= RFdin;
WHEN "0110" => reg6 <= RFdin;
WHEN "0111" => reg7 <= RFdin;
WHEN "1000" => reg8 <= RFdin;
WHEN "1001" => reg9 <= RFdin;
WHEN "1010" => reg10 <= RFdin;
WHEN "1011" => reg11 <= RFdin;
WHEN "1100" => reg12 <= RFdin;
WHEN "1101" => reg13 <= RFdin;
WHEN "1110" => reg14 <= RFdin;
WHEN "1111" => reg15 <= RFdin;
```

```
WHEN OTHERS => NULL;

END CASE;

END IF;

END IF;

END PROCESS WRITE;

READ:PROCESS
(rRFA,Ra,reg0,reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8,reg9,reg10,reg11,reg12,reg13,reg14,
reg15)

BEGIN

IF rRFA = '1' THEN

CASE Ra IS

WHEN "0000" => Rfbout <= reg0;

WHEN "0001" => Rfbout <= reg1;

WHEN "0010" => Rfbout <= reg2;

WHEN "0011" => Rfbout <= reg3;

WHEN "0100" => Rfbout <= reg4;

WHEN "0101" => Rfbout <= reg5;

WHEN "0110" => Rfbout <= reg6;

WHEN "0111" => Rfbout <= reg7;

WHEN "1000" => Rfbout <= reg8;

WHEN "1001" => Rfbout <= reg9;

WHEN "1010" => Rfbout <= reg10;

WHEN "1011" => Rfbout <= reg11;

WHEN "1100" => Rfbout <= reg12;

WHEN "1101" => Rfbout <= reg13;

WHEN "1110" => Rfbout <= reg14;

WHEN "1111" => Rfbout <= reg15;

WHEN OTHERS => Rfbout <=
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";

END CASE;

END IF;


END PROCESS READ;


end Behavioral;
```

AXI BUS

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity IP_PerithereaI_v1_0_S00_AXI is
                                    generic (
                                       -- Users to add parameters here

                                       -- User parameters ends
                                       -- Do not modify the parameters beyond this line

                                       -- Width of S_AXI data bus
                                       C_S_AXI_DATA_WIDTH          : integer           := 32;
                                       -- Width of S_AXI address bus
                                       C_S_AXI_ADDR_WIDTH          : integer           := 5
                                    );
                                    port (
                                       -- Users to add ports here

                                       -- User ports ends
                                       -- Do not modify the ports beyond this line

                                       -- Global Clock Signal
                                       S_AXI_ACLK    : in std_logic;
                                       -- Global Reset Signal. This Signal is Active LOW
                                       S_AXI_ARESETN         : in std_logic;
                                       -- Write address (issued by master, acceped by Slave)
                                       S_AXI_AWADDR          :                                   in
std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
                                       -- Write channel Protection type. This signal indicates the
                                       -- privilege and security level of the transaction, and whether
                                       -- the transaction is a data access or an instruction access.
```

```
                S_AXI_AWPROT    : in std_logic_vector(2 downto 0);
                -- Write address valid. This signal indicates that the master signaling
                -- valid write address and control information.
                S_AXI_AWVALID   : in std_logic;
                -- Write address ready. This signal indicates that the slave is ready
                -- to accept an address and associated control signals.
                S_AXI_AWREADY   : out std_logic;
                -- Write data (issued by master, acceped by Slave)
                S_AXI_WDATA     : in std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
                -- Write strobes. This signal indicates which byte lanes hold
                -- valid data. There is one write strobe bit for each eight
                -- bits of the write data bus.
                S_AXI_WSTRB     : in std_logic_vector((C_S_AXI_DATA_WIDTH/8)-1 downto 0);
                -- Write valid. This signal indicates that valid write
                -- data and strobes are available.
                S_AXI_WVALID    : in std_logic;
                -- Write ready. This signal indicates that the slave
                -- can accept the write data.
                S_AXI_WREADY    : out std_logic;
                -- Write response. This signal indicates the status
                -- of the write transaction.
                S_AXI_BRESP     : out std_logic_vector(1 downto 0);
                -- Write response valid. This signal indicates that the channel
                -- is signaling a valid write response.
                S_AXI_BVALID    : out std_logic;
                -- Response ready. This signal indicates that the master
                -- can accept a write response.
                S_AXI_BREADY    : in std_logic;
                -- Read address (issued by master, acceped by Slave)
                S_AXI_ARADDR    : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
                -- Protection type. This signal indicates the privilege
```

```
                -- and security level of the transaction, and whether the
                -- transaction is a data access or an instruction access.
                S_AXI_ARPROT    : in std_logic_vector(2 downto 0);
                -- Read address valid. This signal indicates that the channel
                -- is signaling valid read address and control information.
                S_AXI_ARVALID   : in std_logic;
                -- Read address ready. This signal indicates that the slave is
                -- ready to accept an address and associated control signals.
                S_AXI_ARREADY   : out std_logic;
                -- Read data (issued by slave)
                S_AXI_RDATA : out
std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
                -- Read response. This signal indicates the status of the
                -- read transfer.
                S_AXI_RRESP : out std_logic_vector(1 downto 0);
                -- Read valid. This signal indicates that the channel is
                -- signaling the required read data.
                S_AXI_RVALID : out std_logic;
                -- Read ready. This signal indicates that the master can
                -- accept the read data and response information.
                S_AXI_RREADY    : in std_logic
            );
end IP_Perithereal_v1_0_S00_AXI;

architecture arch_imp of IP_Perithereal_v1_0_S00_AXI is

            -- AXI4LITE signals
            signal axi_awaddr : std_logic_vector(C_S_AXI_ADDR_WIDTH-
1 downto 0);
            signal axi_awready      : std_logic;
            signal axi_wready : std_logic;
            signal axi_bresp    : std_logic_vector(1 downto 0);
            signal axi_bvalid   : std_logic;
            signal axi_araddr   : std_logic_vector(C_S_AXI_ADDR_WIDTH-
1 downto 0);
```

```
                signal axi_arready : std_logic;
                signal axi_rdata    :  std_logic_vector(C_S_AXI_DATA_WIDTH-
1 downto 0);
                signal axi_rresp    : std_logic_vector(1 downto 0);
                signal axi_rvalid   : std_logic;

                -- Example-specific design signals
                -- local parameter for addressing 32 bit / 64 bit
C_S_AXI_DATA_WIDTH
                -- ADDR_LSB is used for addressing 32/64 bit
registers/memories
                -- ADDR_LSB = 2 for 32 bits (n downto 2)
                -- ADDR_LSB = 3 for 64 bits (n downto 3)
                constant ADDR_LSB : integer :=
(C_S_AXI_DATA_WIDTH/32)+ 1;
                constant OPT_MEM_ADDR_BITS : integer := 2;
                ------------------------------------------------
                ---- Signals for user logic register space example
                ------------------------------------------------
                ---- Number of Slave Registers 7
                signal slv_reg0     :std_logic_vector(C_S_AXI_DATA_WIDTH-1
downto 0);
                signal slv_reg1     :std_logic_vector(C_S_AXI_DATA_WIDTH-1
downto 0);
                signal slv_reg2     :std_logic_vector(C_S_AXI_DATA_WIDTH-1
downto 0);
                signal slv_reg3     :std_logic_vector(C_S_AXI_DATA_WIDTH-1
downto 0);
                signal slv_reg4     :std_logic_vector(C_S_AXI_DATA_WIDTH-1
downto 0);
                signal slv_reg5     :std_logic_vector(C_S_AXI_DATA_WIDTH-1
downto 0);
                signal slv_reg6     :std_logic_vector(C_S_AXI_DATA_WIDTH-1
downto 0);
                signal slv_reg_rden         : std_logic;
                signal slv_reg_wren         : std_logic;
                signal                                  reg_data_out
:std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
                signal byte_index  : integer;
```

```
                    signal y_out        :std_logic_vector(31 downto 0);
                    signal temp_rfdin:std_logic_vector(43 downto 0);



                    component test is
  PORT(
  clk:in std_logic;
  x:in std_logic_vector(31 downto 0);
  y:out std_logic_vector(31 downto 0);
  rfdin:in std_logic_vector(43 downto 0);
  wrfa:in std_logic;
  code:in std_logic_vector(1 downto 0);
  enable:in std_logic;
  ra2:in std_logic_vector(3 downto 0));
  end component;




begin
                    -- I/O Connections assignments

                    S_AXI_AWREADY           <= axi_awready;
                    S_AXI_WREADY  <= axi_wready;
                    S_AXI_BRESP     <= axi_bresp;
                    S_AXI_BVALID    <= axi_bvalid;
                    S_AXI_ARREADY <= axi_arready;
                    S_AXI_RDATA     <= axi_rdata;
                    S_AXI_RRESP     <= axi_rresp;
                    S_AXI_RVALID    <= axi_rvalid;
                    -- Implement axi_awready generation
                    -- axi_awready is asserted for one S_AXI_ACLK clock cycle
when both
```

```
                    -- S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_awready is
                    -- de-asserted when reset is low.

                    process (S_AXI_ACLK)
                    begin
                      if rising_edge(S_AXI_ACLK) then
                        if S_AXI_ARESETN = '0' then
                          axi_awready <= '0';
                        else
                          if (axi_awready = '0' and S_AXI_AWVALID = '1' and S_AXI_WVALID = '1') then
                            -- slave is ready to accept write address when
                            -- there is a valid write address and write data
                            -- on the write address and data bus. This design
                            -- expects no outstanding transactions.
                            axi_awready <= '1';
                          else
                            axi_awready <= '0';
                          end if;
                        end if;
                      end if;
                    end process;

                    -- Implement axi_awaddr latching
                    -- This process is used to latch the address when both
                    -- S_AXI_AWVALID and S_AXI_WVALID are valid.

                    process (S_AXI_ACLK)
                    begin
                      if rising_edge(S_AXI_ACLK) then
                        if S_AXI_ARESETN = '0' then
                          axi_awaddr <= (others => '0');
                        else
```

```
        if (axi_awready = '0' and S_AXI_AWVALID = '1' and
S_AXI_WVALID = '1') then
          -- Write Address latching
          axi_awaddr <= S_AXI_AWADDR;
        end if;
      end if;
    end if;
  end process;

  -- Implement axi_wready generation
  -- axi_wready is asserted for one S_AXI_ACLK clock cycle
when both
  -- S_AXI_AWVALID and S_AXI_WVALID are asserted.
axi_wready is
  -- de-asserted when reset is low.

  process (S_AXI_ACLK)
  begin
    if rising_edge(S_AXI_ACLK) then
      if S_AXI_ARESETN = '0' then
        axi_wready <= '0';
      else
        if (axi_wready = '0' and S_AXI_WVALID = '1' and
S_AXI_AWVALID = '1') then
          -- slave is ready to accept write data when
          -- there is a valid write address and write data
          -- on the write address and data bus. This design
          -- expects no outstanding transactions.
          axi_wready <= '1';
        else
          axi_wready <= '0';
        end if;
      end if;
    end if;
  end process;
```

```
                                -- Implement memory mapped register select and write logic
generation

                                -- The write data is accepted and written to memory mapped
registers when

                                -- axi_awready, S_AXI_WVALID, axi_wready and
S_AXI_WVALID are asserted. Write strobes are used to

                                -- select byte enables of slave registers while writing.

                                -- These registers are cleared when reset (active low) is
applied.

                                -- Slave register write enable is asserted when valid address
and data are available

                                -- and the slave is ready to accept the write address and write
data.

                                slv_reg_wren <= axi_wready and S_AXI_WVALID and
axi_awready and S_AXI_AWVALID ;


                                process (S_AXI_ACLK)

                                variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS
downto 0);

                                begin

                                  if rising_edge(S_AXI_ACLK) then

                                    if S_AXI_ARESETN = '0' then

                                      slv_reg0 <= (others => '0');

                                      slv_reg1 <= (others => '0');

                                      slv_reg2 <= (others => '0');

                                      slv_reg3 <= (others => '0');

                                      slv_reg4 <= (others => '0');

                                      slv_reg5 <= (others => '0');

                                      slv_reg6 <= (others => '0');

                                    else

                                      loc_addr := axi_awaddr(ADDR_LSB +
OPT_MEM_ADDR_BITS downto ADDR_LSB);

                                      if (slv_reg_wren = '1') then

                                        case loc_addr is

                                          when b"000" =>

                                            for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1)
loop

                                              if ( S_AXI_WSTRB(byte_index) = '1' ) then
```

```
                                        -- Respective byte enables are asserted as per write
strobes
                                        -- slave registor 0
                                        slv_reg0(byte_index*8+7 downto byte_index*8) <=
S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
                                      end if;
                                    end loop;
                                  when b"001" =>
                                    for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1)
loop
                                      if ( S_AXI_WSTRB(byte_index) = '1' ) then
                                        -- Respective byte enables are asserted as per write
strobes
                                        -- slave registor 1
                                        slv_reg1(byte_index*8+7 downto byte_index*8) <=
S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
                                      end if;
                                    end loop;
                                  when b"010" =>
                                    for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1)
loop
                                      if ( S_AXI_WSTRB(byte_index) = '1' ) then
                                        -- Respective byte enables are asserted as per write
strobes
                                        -- slave registor 2
                                        slv_reg2(byte_index*8+7 downto byte_index*8) <=
S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
                                      end if;
                                    end loop;
                                  when b"011" =>
                                    for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1)
loop
                                      if ( S_AXI_WSTRB(byte_index) = '1' ) then
                                        -- Respective byte enables are asserted as per write
strobes
                                        -- slave registor 3
                                        slv_reg3(byte_index*8+7 downto byte_index*8) <=
S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
```

```
          end if;
        end loop;
      when b"100" =>
        for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1)
loop
          if ( S_AXI_WSTRB(byte_index) = '1' ) then
            -- Respective byte enables are asserted as per write
strobes
            -- slave registor 4
            slv_reg4(byte_index*8+7 downto byte_index*8) <=
S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
          end if;
        end loop;
      when b"101" =>
        for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1)
loop
          if ( S_AXI_WSTRB(byte_index) = '1' ) then
            -- Respective byte enables are asserted as per write
strobes
            -- slave registor 5
            slv_reg5(byte_index*8+7 downto byte_index*8) <=
S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
          end if;
        end loop;
      when b"110" =>
        for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1)
loop
          if ( S_AXI_WSTRB(byte_index) = '1' ) then
            -- Respective byte enables are asserted as per write
strobes
            -- slave registor 6
            slv_reg6(byte_index*8+7 downto byte_index*8) <=
S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
          end if;
        end loop;
      when others =>
        slv_reg0 <= slv_reg0;
        slv_reg1 <= slv_reg1;
```

```
                    slv_reg2 <= slv_reg2;
                    slv_reg3 <= slv_reg3;
                    slv_reg4 <= slv_reg4;
                    slv_reg5 <= slv_reg5;
                    slv_reg6 <= slv_reg6;
                  end case;
                end if;
              end if;
            end if;
          end process;


          -- Implement write response logic generation
          -- The write response and response valid signals are asserted by the slave
          -- when axi_wready, S_AXI_WVALID, axi_wready and S_AXI_WVALID are asserted.
          -- This marks the acceptance of address and indicates the status of
          -- write transaction.


          process (S_AXI_ACLK)
          begin
            if rising_edge(S_AXI_ACLK) then
              if S_AXI_ARESETN = '0' then
                axi_bvalid  <= '0';
                axi_bresp   <= "00"; --need to work more on the responses
              else
                if (axi_awready = '1' and S_AXI_AWVALID = '1' and axi_wready = '1' and S_AXI_WVALID = '1' and axi_bvalid = '0'  ) then
                  axi_bvalid <= '1';
                  axi_bresp  <= "00";
                elsif (S_AXI_BREADY = '1' and axi_bvalid = '1') then    -- check if bready is asserted while bvalid is high)
                  axi_bvalid <= '0';                                    -- (there is a possibility that bready is always asserted high)
                end if;
```

```
    end if;
  end if;
end process;

-- Implement axi_arready generation
-- axi_arready is asserted for one S_AXI_ACLK clock cycle when
-- S_AXI_ARVALID is asserted. axi_awready is
-- de-asserted when reset (active low) is asserted.
-- The read address is also latched when S_AXI_ARVALID is
-- asserted. axi_araddr is reset to zero on reset assertion.

process (S_AXI_ACLK)
begin
  if rising_edge(S_AXI_ACLK) then
    if S_AXI_ARESETN = '0' then
      axi_arready <= '0';
      axi_araddr  <= (others => '1');
    else
      if (axi_arready = '0' and S_AXI_ARVALID = '1') then
        -- indicates that the slave has acceped the valid read address
        axi_arready <= '1';
        -- Read Address latching
        axi_araddr  <= S_AXI_ARADDR;
      else
        axi_arready <= '0';
      end if;
    end if;
  end if;
end process;

-- Implement axi_arvalid generation
-- axi_rvalid is asserted for one S_AXI_ACLK clock cycle when both
```

```
                    -- S_AXI_ARVALID and axi_arready are asserted. The slave
registers
                    -- data are available on the axi_rdata bus at this instance. The
                    -- assertion of axi_rvalid marks the validity of read data on the
                    -- bus and axi_rresp indicates the status of read
transaction.axi_rvalid
                    -- is deasserted on reset (active low). axi_rresp and axi_rdata
are
                    -- cleared to zero on reset (active low).
                    process (S_AXI_ACLK)
                    begin
                      if rising_edge(S_AXI_ACLK) then
                        if S_AXI_ARESETN = '0' then
                          axi_rvalid <= '0';
                          axi_rresp  <= "00";
                        else
                          if (axi_arready = '1' and S_AXI_ARVALID = '1' and
axi_rvalid = '0') then
                            -- Valid read data is available at the read data bus
                            axi_rvalid <= '1';
                            axi_rresp  <= "00"; -- 'OKAY' response
                          elsif (axi_rvalid = '1' and S_AXI_RREADY = '1') then
                            -- Read data is accepted by the master
                            axi_rvalid <= '0';
                          end if;
                        end if;
                      end if;
                    end process;


                    -- Implement memory mapped register select and read logic
generation
                    -- Slave register read enable is asserted when valid address is
available
                    -- and the slave is ready to accept the read address.
                    slv_reg_rden <= axi_arready and S_AXI_ARVALID and (not
axi_rvalid) ;
```

```
                process (slv_reg0,y_out, slv_reg1, slv_reg2, slv_reg3, slv_reg4,
slv_reg5, slv_reg6, axi_araddr, S_AXI_ARESETN, slv_reg_rden)
                variable    loc_addr    :std_logic_vector(OPT_MEM_ADDR_BITS
downto 0);
                begin
                    -- Address decoding for reading registers
                    loc_addr          :=          axi_araddr(ADDR_LSB          +
OPT_MEM_ADDR_BITS downto ADDR_LSB);
                    case loc_addr is
                      when b"000" =>
                        reg_data_out <= slv_reg0;
                      when b"001" =>
                     reg_data_out<= y_out;
                      when b"010" =>
                        reg_data_out <= slv_reg2;
                      when b"011" =>
                        reg_data_out <= slv_reg3;
                      when b"100" =>
                        reg_data_out <= slv_reg4;
                      when b"101" =>
                        reg_data_out <= slv_reg5;
                      when b"110" =>
                        reg_data_out <= slv_reg6;
                      when others =>
                        reg_data_out  <= (others => '0');
                    end case;
                end process;

                -- Output register or memory read data
                process( S_AXI_ACLK ) is
                begin
                  if (rising_edge (S_AXI_ACLK)) then
                    if ( S_AXI_ARESETN = '0' ) then
                      axi_rdata  <= (others => '0');
                    else
```

```
                                    if (slv_reg_rden = '1') then
                                      -- When there is a valid read address (S_AXI_ARVALID)
with
                                      -- acceptance of read address by the slave (axi_aready),
                                      -- output the read dada
                                      -- Read address mux
                                        axi_rdata <= reg_data_out;     -- register read data
                                    end if;
                                  end if;
                                end if;
                              end process;

temp_rfdin(43 downto 12)<=slv_reg2;
temp_rfdin(11 downto 0)<=slv_reg3(31 downto 20);

                              -- Add user logic here
test0:test
        port map(
        clk=>S_AXI_ACLK,
        x=>slv_reg0,
        y=>y_out,----------slv_reg1
        rfdin=>temp_rfdin,--concatanate
        wrfa=>slv_reg4(0),
        code=>slv_reg5(31 downto 30),
        enable=>slv_reg4(1),
        ra2=>slv_reg6(3 downto 0));
                              -- User logic ends

end arch_imp;
```

ΠΑΡΑΡΤΗΜΑ Β
SOFTWARE Κώδικας

```
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "xbasic_types.h"
void ReputationInput(Xuint32 ra,Xuint32 IP_Addr,Xuint32 Reputation);
Xuint32 IPinput(Xuint32 x,Xuint32 code);
void clearReg(void);
void print(char *str);

int main()
```

```
{
                                    //Xuint32 *baseaddr_p = (Xuint32
*)XPAR_IPFORME_0_S00_AXI_BASEADDR ;
    init_platform();
    clearReg();
    Xuint32 z=2;
    printf("Hello World %d \n\r", z);
   // ReputationInput(0x00000000,0xc0a8d802,0xc2900000);

   // ReputationInput(0x00000007,0xc0a8d804,0x2e200000);
   // ReputationInput(0x00000003,0xc0a8d806,0x2d200000);
    //ReputationInput(0x00000004,0xc0a8d808,0x2e200000);
    //ReputationInput(0x00000005,0xc0a8d805,0xfff00000);
   // ReputationInput(0x00000009,0xc0a8d80a,0x2d200000);

    ReputationInput(0x00000000,0xc0a8d80a,0x2d200000);  //mono ayto diabazei

    Xuint32 testip =0xc0a8d80a;
    Xuint32 rule = 0xc1000000;  //περνα

    z=IPinput(testip,rule);
    if (z==0x00000000){
                                    printf(" 0x%08x\n",z);
                                     printf("package with ip : 0x%08x not passed (malicious) \n\r",
testip);
    }
    else if (z==testip) {
                                    printf(" 0x%08x\n",z);
                                    printf("package with ip : 0x%08x passed successfully (non
malicious) \n\r", testip);
    }
    else

    {
                                    printf(" 0x%08x\n",z);
```

```
                                printf("problem \n");}

   //ReputationInput(0x00000004,0xc0a8d803,0xc2900000);
  // printf("Test My_Design : 0x%08x\n\r", IPinput(0xc0a8d802,0xc0000000)); //περνα
 //  ReputationInput(0x00000002,0xc0a8d801,0xc2900000);
     // printf("Test My_Design : 0x%08x\n\r", IPinput(0xc0a8d80a,0xc0000000)); //οχι
     // ReputationInput(0x00000001,0xc0a8d800,0xc2900000);
     // printf("Test My_Design : 0x%08x\n\r", IPinput(0xc0a8d806,0xc0000000)); //οχι
    //  ReputationInput(0x00000009,0xc0a8d80b,0xc2900000);
     // printf("Test My_Design : 0x%08x\n\r", IPinput(0xc0a8d804,0xc0000000));


      // testip =0xc0a8d805;
      // rule = 0xc0000000;  //perna
      // z=IPinput(testip,rule);
      // if (z==0x00000000){
       //                  printf("package with ip : 0x%08x not passed (malicious) 0x%08x\n\r", testip,z);
      // }
      // else if (z==testip) {
       //                 printf("package with ip : 0x%08x passed successfully (non malicious) \n\r", testip);
      // }
      // else
      // {printf("problem \n");}
      // printf("Test My_Design : 0x%08x\n\r", IPinput(0xc0a8d806,0xc0000000)); //οχι
   cleanup_platform();
   return 0;
}
void ReputationInput(Xuint32 ra,Xuint32 IP_Addr,Xuint32 Reputation)
{
                                Xuint32 *baseaddr_p = (Xuint32 *)XPAR_IP_PERITHEREAL_0_S00_AXI_BASEADDR ;
                                *(baseaddr_p+2) =IP_Addr;// τιμη για το rfdin ta prwta 32 bits
                                *(baseaddr_p+3)=Reputation;  //sto low part dwse 12 noumera gia to rfdin
                                *(baseaddr_p+6)=ra;  //Ra
```

```
                        *(baseaddr_p+4)=1;//wrfa,enable


                        //*(baseaddr_p+4)=0; // sto telos ths anazhthshs mhdenizei to
enable
}
Xuint32 IPinput(Xuint32 x, Xuint32 code ){
                        Xuint32 *baseaddr_p = (Xuint32
*)XPAR_IP_PERITHEREAL_0_S00_AXI_BASEADDR ;
                        *(baseaddr_p+0)=x;
                        *(baseaddr_p+5)=code;
                        //*(baseaddr_p+4)=1;
                        //*(baseaddr_p+4)=0; // sto telos ths anazhthshs mhdenizei to
enable
                        *(baseaddr_p+4)=2;//wrfa,enable
                        //*(baseaddr_p+4)=0; // sto telos ths anazhthshs mhdenizei to
enable
                        //*(baseaddr_p+4)=3;
                        *(baseaddr_p+6)=0;
                        *(baseaddr_p+4)=0; // sto telos ths anazhthshs mhdenizei to
enable
                        //*(baseaddr_p+5)=0;
                        return *(baseaddr_p+1);


}


void clearReg(void)
{
                        Xuint32 *baseaddr_p = (Xuint32
*)XPAR_IP_PERITHEREAL_0_S00_AXI_BASEADDR ;
                        int i;
                        for (i==0;i<16;i++){
                            *(baseaddr_p+2) =0;// τιμη για το rfdin ta prwta 32 bits
                            *(baseaddr_p+3)=0;          //sto low part dwse 12 noumera
gia to rfdin
                            *(baseaddr_p+6)=i;      //Ra
                            *(baseaddr_p+4)=1;//wrfa,enable
                        }
```

```
                    *(baseaddr_p+4)=0; // sto telos ths anazhthshs mhdenizei to
enable
}
```