Διερεύνηση των παραμέτρων , συνθεσης και υλοποιησης , του λογισμικού XILINX ISE , προκειμένου να επιτευχθεί η βέλτιστη κατανάλωση ισχύος , των FPGA κυκλωμάτων , Virtex II Pro , κατά την χρήση των ευρέως διαδεδομένων αλγορίθμων ασφαλείας.

Εκπόνηση

Ντόρτου Ειρήνη

Εξεταστική επιτροπή

Παπαευσταθίου Ιωάννης (επιβλέπων)

Δόλλας Απόστολος

Πνευματικάτος Διονύσης

ΧΑΝΙΑ 2009

Ευχαριστίες

Οφείλω να ευχαριστήσω τον κύριο Μεϊντάνη Δημητρη για τις συμβουλές του , αλλά και για την βοήθεια , που μου προσέφερε κατά τη διάρκεια της εκπόνησής της τής εργασίας αυτής.

Αφιερώνω αυτήν την εργασία στην οικογένειά μου.

Και στο laptop μου.

Merci

ΠΡΟΛΟΓΟΣ

Στο πλαίσιο αυτής της διπλωματικής εργασίας , διερευνήθηκαν οι συνδυασμοί, διάφορων παραμέτρων , σύνθεσης και υλοποίησης , της Xilinx , με σκοπό την μέτρηση και βελτιστοποίηση της κατανάλωσης ισχύος των FPGA κυκλωμάτων , Virtex-II PRO , κατά τη διάρκεια της εφαρμογής ευρέως διαδεδομένων αλγορίθμων ασφαλείας.

Γενικά , η κατανάλωση ισχύος των επαναπρογραμματίσιμων συστημάτων (FPGAs) αποτελεί ένα από τα βασικά και ίσως το σημαντικότερο πρόβλημά τους , έναντι των μη επαναπρογραμματίσιμων συστημάτων , για το σχεδιασμό εφαρμογών.

Η κατανάλωση ισχύος απασχολεί , ειδικά , τις εφαρμογές ασφαλείας (security applications) , όπου η ανάλυση της ισχύος αποτελεί ένα σημαντικό και αποτελεσματικό μέσο για επιθέσεις. Μέσω της παρακολούθησης της κατανάλωσης ισχύος των συσκευών , κατά την διάρκεια διάφορων εφαρμογών ασφαλείας , είναι δυνατόν κάποιος να συγκεντρώσει διάφορες κρίσιμες πληροφορίες. Όσο μεγαλύτεση είναι η κατανάλωση ισχύος , τόσο πιο εύτρωτες είναι οι εφαρμογές ασφαλείας σε power analysis επιθέσεις.

Εμείς , συγκεντρώνουμε οχτώ διαφορετικούς αλγόριθμους ασφαλείας και επιχειρούμε να πετύχουμε , όσο το δυνατόν καλύτερα , μείωση της κατανάλωσης ισχύος του καθενός , με σκοπό αυτές οι επιπτώσεις της διαρροής ισχύος να ελλατωθούν ή και να εξαληφθούν τελείως.

Ο προσανατολισμός της εργασίας μας είναι να βρεθούν οι καλύτερες δυνατές παράμετροι , σύνθεσης και υλοποίησης , κατά τις οποίες θα έχουμε τη μικρότερη κατανάλωση ισχύος , των αλγορίθμων που χρησιμοποιούμε.

ΠΕΡΙΛΗΨΗ

Στη διπλωματική αυτή , παρουσιάζονται και αναλύονται οι μετρήσεις κατανάλωσης ισχύος , γνωστών αλγορίθμων ασφαλείας , όταν αυτοί εφαρμόζονται σε Virtex-II PRO FPGA , καθώς έχουν προκύψει από τους συνδυασμούς των παραμέτρων σύνθεσης και υλοποίησης της Xilinx , τους οποίους εξετάζουμε , με στόχο τη βελτιστοποίηση της κατανάλωσης ισχύος των αλγορίθμων , που χρησιμοποιούμε.

Αρχικά , δημιουργούμε μια top level FSM , για να ελέγχουμε τα σήματα κάθε εφαρμογής. Η FSM μας εκτελεί έναν ατέρμωνα βρόγχο , ο οποίος μας παρέχει με μια σταθερή κατανάλωση ρεύματος, καθ'όλη την διάρκεια του πειράματος.

Οι αλγόριθμοι , που εξετάζουμε είναι οι εξής: BasicRSA , BasicDES , Camellia_1 , Camellia_2 , TripleDES , AES , DES και MD5.

Κάθε μέτρηση που παίρνουμε , αντιπροσωπεύει μία σειρά παραμέτρων synthesis και implementation της Xilinx. Αυτές είναι οι παράμετροι , που θα διερευνήσουμε , για να βρούμε την βέλτιστη κατανάλωση ισχύος.

Οι παράμετροι , με τις οποίες εργαστήκαμε , είναι οι ακόλουθοι: Optimization goal , Optimization effort , Map effort level , Place & route mode και Power reduction.

Για την υλοποίηση της εργασίας μας , χρησιμοποιήσαμε: οχτώ αλγόριθμους ασφαλείας , το αναπτυξιακό σύστημα της FPGA , Virtex II pro XUP , ένα κύκλωμα αντίστασης , γνωστή στα 0.1 Ohm , ένα τροφοδοτικό και έναν παλμογράφο.

Τα αποτελέσματά μας κυμαινονται από πτώση της κατανάλωσης ισχύος , κατά 306mW , ως αυξηση της κατανάλωσης ισχύος κατά 33mW. Επίσης , η μέση ποσοστιαία βελτίωση της κατανάλωσης ισχύος , κυμαίνεται από 7,62% ως - 3,55%.

ΠΕΡΙΕΧΟΜΕΝΑ

1. **Εισαγωγή** …….....5
   - ***FPGA*** …….....5
   - ***Πρόβλημα Κατανάλωσης Ισχύος*** …….....7
2. **Related Work** …….....8
3. **Αλγόριθμοι Ασφαλείας και Σύστημα Virtex-II PRO** 10
   - ***Αλγόριθμοι ασφαλείας*** 10
     - ***AES*** 11
     - ***DES*** 12
     - ***3DES*** 13
     - ***BASIC DES*** 13
     - ***RSA*** 14
     - ***CAMELLIA*** 15
     - ***MD5*** 16
   - **Virtex-II PRO**.....17
4. **Μέτρηση Κατανάλωσης Ισχύος** 21
   - ***Τύποι Εφαρμογών*** 21
   - ***Μέθοδοι Μέτρησης*** 22
   - ***Τεχνικές Μέτρησης***.....24
5. **Μεθοδολογία** 32
   - ***Αρχική Διαδικασία*** 32
   - ***Μέτρηση Κατανάλωσης Ισχύος*** 34
   - ***Βελτιστοποίηση Κατανάλωσης Ισχύος*** 35
6. **Συμπεράσματα** 40
7. **Βιβλιογραφία** 63
8. **Appendix** 64

1. Εισαγωγή

FPGA

Όταν μιλάμε για Ολοκληρωμένο Κύκλωμα Προγραμματιζόμενης Λογικής (Field Programmable Gate Array—FPGA) , κάνουμε λόγο για μία κατηγορία ολοκληρωμένων κυκλωμάτων , που αρχικά δημιουργήθηκε από την Xilinx,INC , στην οποία η λογική λειτουγρία του κυκλώματος καθορίζεται από τον χρήστη , χρησιμοποιώντας εργαλεία ανάπτυξης λογισμικού της Xilinx , αφού το ολοκληρωμένο κύκλωμα έχει κατασκευαστεί και έχει παραδοθεί στον τελικό προορισμό του. Τα Gate arrays είναι ένας άλλος τύπος ολοκληρωμένων κυκλωμάτων (Integrated Circuits—IC) , η λογική των οποίων καθορίζεται κατά τη διαδικασία κατασκευής και δεν μπορεί να επαναπρογραμματιστεί.

Τα FPGAs μπορούν να επαναπρογραμματιστούν πολλές φορές , για διαφορετικές εφαρμογές , την κάθε φορά. Οι εφαρμογές με FPGAs περιλαμβάνουν γρήγορους μετρητές (fast counters) , γρήγορα σχέδια διοχέτευσης (pipelined designs) , σχέδια ενταντικών καταχωρητών (register intensive designs) , και πολυεπίπεδη λογική τροφοδοτουμενη από μπαταρία (battery powered multi-level logic).

Η διαμόρφωση των FPGAs προσδιορίζεται γενικά με βάση μια γλώσσα περιγραφής υλικού (hardware description language—HDL) , παρόμοια με αυτή που χρησιμοποιείται για τα ολοκληρωμένα κυκλωμάτα συγκεκριμένης εφαρμογής (application-specific integrated circuit—ASIC). Στο παρελθόν , χρησιμοποιήθηκαν τα διαγράμματα κυκλωμάτων , για να καθορίσουν τις ρυθμίσεις , όπως ήταν για τα ASICs , αλλά αυτό τώρα είναι πολύ σπάνιο. Τα FPGAs μπορούν να χρησιμοποιηθούν , για την υλοποίηση οποιασδήποτε λογικής εφαρμογής , που μπορεί να εκτελέσει ένα ASIC.

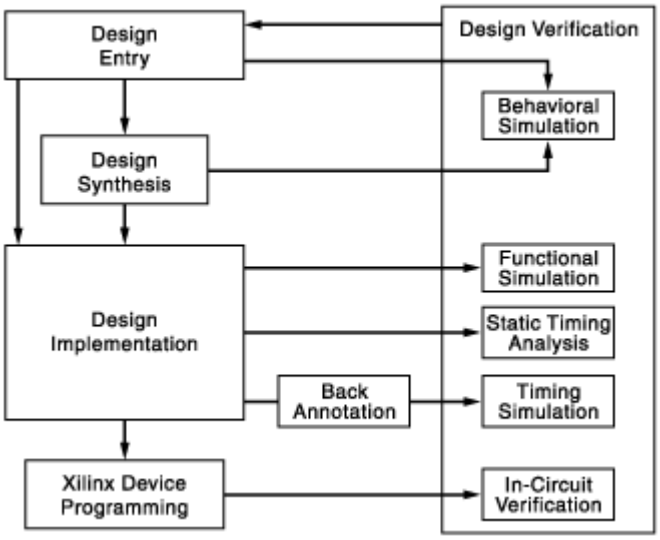
Η δυνατότητα , που μας παρέχουν τα FPGAs , να ενημερώνουμε τη λειτουργικότητα τού ολοκληρωμένου μας , μετά την κατασκευή , είναι μεγάλο πλεονέκτημα. Αυτό , σε συνδυασμό με το χαμηλό και μη επαναλαμβανόμενο κόστος μηχανικού , σε σχέση με ένα σχεδιασμό ASIC , καθιστά τα FPGAs ως την πρώτη επιλογή για πολλές εφαρμογές.

Τα FPGAs περιέχουν συστατικά (components) προγραμματιζόμενης λογικής , που ονομάζονται «μπλοκ λογικής» (logic blocks) , καθώς και μία ιεράρχηση των αναδιαρθρώσιμων και επαναπρογραμματιζομένων διασυνδέσεων , που επιτρέπουν στα μπλοκ να προγραμματιστούν «μαζί». Κάτι σαν μία breadboard (πλακέτα κυκλωμάτων) , που προγραμματίζεται από ένα μόνο τσιπ.Τα «λογικά μπλοκ» μπορεί να ρυθμιστούν έτσι , ώστε να εκτελούν περίποκες και πολυσυνδυαστικές λειτουργίες , ή απλά και μόνο βασικές λογικές πύλες , όπως είναι η AND και η XOR. Στις περισσότερες FPGAs τα «λογικά μπλοκ» περιλαμβάνουν επίσης και στοιχεία μνήμης , τα οποία μπορεί να είναι απλοί μανδαλωτές (flip-flops) ή και πιο ολοκληρωμένα μπλοκ μνήμης.

Η βασική δομή της ροής σχεδιασμού (design flow) , για ένα FPGA , φαίνεται και στο σχήμα 1 , έχει ως εξής :

1. Δημιουργούμε ένα ISE project.
2. Τρέχουμε behavioral προσομοίωση (behavioral simulation).
3. Επαναλαμβάνουμε τα βήματα 1 και 2 μέχρι το design μας να τρέχει σωστά.
4. Συνθέτουμε το design μας (synthesize design).
5. Υλοποιούμε το design μας (implement design).
6. Εξετάζουμε τις αναφορές , που δημιουργούνται από τη διαδικασία implement και αν είναι απαραίτητο επαναλαμβάνουμε απο το βήμα 2 , εώς ότου να καταλήξουμε στο αποτέλεσμα που επιθυμούμε.
7. Εκτέλεση χρονικής προσομοίωση , για την επαλήθευση της λειτουργικότητας και του χρονοδιαγράμματος του design.
8. Προγραμματίζουμε την Xilinx® συσκευή μας (εμείς χρησιμοποιούμε την Virtex-II PRO) με τον ακόλουθο τρόπο :

   - Παράγουμε ένα αρχείο προγραμματισμού .bit , με το οποίο θα προγραμματίσουμε το FPGA μας
   - Παράγουμε ένα PROM , ACE ή JTAG αρχείο , για εκσφαλμάτωση (debugging) ή για να το μεταφέρουμε (download) στην συσκευή
   - Προγραμματίσουμε την συσκευή μας με ένα καλώδιο προγραμματισμού(programming cable).



Σχήμα 1.

Όπως προαναφέραμε τα πλεονεκτήματα ενός FPGA , έναντι ενός ASIC , περιλαμβάνουν τη δυνατότητα επαναπρογραμματισμού του , για την διόρθωση και αποφυγή σφαλμάτων , χαμηλότερο και μη επαναλαμβανόμενο κόστος μηχανικού , αλλά και μικρότερο χρονικό διάστημα από την παραγωγή του μέχρι την εμφάνισή του στην αγορά. Οι πωλητές μπορούν , επίσης , να λάβουν μια μέση οδό , με την ανάπτυξη του υλικού τους από συνήθεις FPGAs , αλλά κατασκευάζοντας την τελική μορφή , έτσι ώστε να μην μπορεί πλέον να τροποποιηθεί , απαξ και το design έχει δεσμευτεί.

Από την άλλη πλευρά , όμως εχουμε και τα μειονεκτήματα. Τα FPGAs ειναι πιο αργά. Είναι λιγότερο αποδοτικά , ενεργειακά , δηλαδή καταναλώνουν περισσότερη ενέργεια , και γενικώς επιτυγχάνουν χαμηλότερη λειτουργικότητα από τα αντίστοιχά τους ASICs.

Πρόβλημα Κατανάλωσης Ισχύος

Όπως αναφέραμε και παραπάνω , η κατανάλωση ισχύος των επαναπρογραμματίσιμων συστημάτων (FPGAs) αποτελεί ένα από τα βασικά και ίσως το σημαντικότερο μειονέκτημά τους , έναντι των μη επαναπρογραμματίσιμων συστημάτων , για το σχεδιασμό εφαρμογών. Αυτό συμβαίνει γιατί , γενικώς , τα FPGAs περιέχουν μεγαλύτερο αριθμό τρανζίστορ , σε σχέση τα ASICs , τα οποία βρίσκονται στον αντίποδα. Ο υψηλός αριθμός τρανζίστορ στο ίδιο τσιπ , έχει καταστήσει την κατανάλωση ισχύος το θεμελιώδες πρόβλημά τους.

Η ευελιξία , μετά κατασκευής , που παρέχουν τα FPGAs υλοποιείται χρησιμοποιώντας ένα μεγάλο αριθμό από προκατασκευασμένα μονοπάτια δρομολόγησης (routing tracks) και προγραμματιζόμενων διακοπτών. Αυτά τα μονοπάτια , μπορεί να είναι αρκετά μεγάλα και να καταναλώνουν σημαντική ποσότητα ενέργειας , κάθε φορά που αλλάζουν. Επιπλέον , οι προγραμματιζόμενοι διακόπτες , προσθέτουν χωρητικότητα σε κάθε track , γεγονός που αυξάνει περαιτέρω την κατανάλωση ισχύος των FPGAs.

Από την άλλη πλευρά , συσκευές χαμηλής ισχύος (low power devices) , χρειάζονται έναν γρήγορο , αλλά και ισχυρό , μηχανισμό κρυπτογράφησης / αποκρυπτογράφησης . Αλλά αυτοί οι αλγόριθμοι , κρυπτογράφησης ή ελέγχου ταυτότητας (encryption/authentication) , χρειάζονται υψηλή ισχύ επεξεργασίας (high processing power) και / ή έχουν μεγάλο χρόνο εκτέλεσης , με αποτέλεσμα να καταναλώνουν σημαντικές ποσότητες ενέργειας. Για τα ενσωματωμένα συστήματα , που λαμβάνουν ενέργεια από μπαταρία (battery powered) , ίσως μία από τις σημαντικότερες προκλήσεις τους , είναι η αναντιστοιχία μεταξύ της ενέργειας και της απόδοσης , που απαιτείται στις εφαρμογές ασφαλείας , και των δυνατοτήτων της μπαταρίας και του επεξεργαστή.

Η κατανάλωση ισχύος απασχολεί , ειδικά , τις εφαρμογές ασφαλείας (security applications) , όπου η ανάλυση της ισχύος αποτελεί ένα σημαντικό και αποτελεσματικό μέσο για επιθέσεις. Μέσω

της παρακολούθησης της κατανάλωσης ισχύος των συσκευών , κατά την διάρκεια διάφορων εφαρμογών , είναι δυνατόν κάποιος να συγκεντρώσει διάφορες πληροφορίες ασφαλείας , όπως π.χ. το κλειδί κρυπτογράφησης (encryption key). Όσο μεγαλύτεση είναι η κατανάλωση ισχύος , τόσο πιο εύτρωτες είναι οι εφαρμογές ασφαλείας σε power analysis επιθέσεις.

Αυτές οι επιπτώσεις της διαρροής ισχύος μπορεί να ελλατωθούν ή και να εξαληφθούν τελείως , εφόσον καταφέρουμε να πετύχουμε , όσο το δυνατόν καλύτερα , μείωση της κατανάλωσης ισχύος του design μας. Είναι , λοιπόν , απαραίτητο για τους σχεδιαστές να μπορεί να προσδιοριστεί και να βελτιστοποιηθεί η κατανάλωση ισχύος , κατά την διαδικασία αναπτυξης , ειδικά για τους ευρέως διαδεδομένους , power consuming , αλγορίθμους ασφαλείας.

Για τον λόγο αυτό , η διπλωματική αυτή ασχολείται με τον προσδιορισμό και τη βελτιστοποίηση της κατανάλωσης ισχύος , οχτώ ευρέως διαδεδομένων αλγορίθμων ασφαλείας , που εφαρμόζονται σε Virtex-II Pro FPGA , της Xilinx.

2. Related Work

Αυτή η διπλωματική είναι η πρώτη που ασχολείται με τη δερεύνηση των παραμέτρων , συνθεσης και υλοποιησης , του λογισμικού Xilinx ISE , για τη μέτρηση και βελτιστοποίηση της κατανάλωσης ισχύος , για αλγόριθμους ασφαλείας , του πηρύνα του FPGA μας , απευθείας από το hardware.

Σχετική δουλειά βρήκαμε πάνω στο θέμα μέτρησης κατανάλωσης ισχύος έναντι των θεωρητικών υπολογισμών κατανάλωσης ισχύος , σε αλγορίθμους ασφαλείας [6] , όπου οι Μεϊντάνης και Παπαευσταθίου , παρουσιάζουν και συγκρίνουν τη θεωρητική και πειραματική τιμή κατανάλωσης ισχύος , για αλγορίθμους ασφαλείας , όταν αυτοί εφαρμόζονται σε Xilinx και Altera FPGAs. Οι διαφορές , μεταξύ θεωρητικών και πειραματικών τιμών κατανάλωσης ισχύος , σύμφωνα με τα αποτελέσματά τους , διακυμαινονται από 15.52% ως 208% για την Xilinx και 5,64% ως 32.15% για την Altera. Ο εξοπλισμός , που χρησιμοποίησαν για τα πειράματά τους , όσο αναφορά στην Xilinx , είναι ο ίδιος που χρησιμοποιήσαμε και εμείς (XUP Virtex-II PRO Development System) , για τα δικά μας.

Επίσης , σχετική εργασία , έχει γίνει και για την κατανάλωση ισχύος , αλγορίθμων ασφαλείας , σε ασύρματα δίκτυα [7] , όπου οι Μεϊντάνης και Παπαευσταθίου , παρουσιάζουν και συγκρίνουν την τιμή κατανάλωσης ισχύος για hardware υλοποίηση με την τιμή κατανάλωσης ισχύος για software υλοποίηση , για αλγορίθμους ασφαλείας , όταν αυτοί εφαρμόζονται σε ασύρματα δίκτυα. Στην εργασία τους , αυτή , μετράται η συνολική κατανάλωση ενέργειας για κάθε αλγόριθμο ασφαλείας , όταν εφαρμόζεται σε αναδιαρθρώσιμο υλικό (FPGAs) και συγκρίνεται με τη συνολική κατανάλωση

ενέργειας της ισοδύναμης εφαρμογής λογισμικού. Αποδεικνύουν , ότι οι εφαρμογές του υλικού για ένα state-of-the-art FPGA διεκπεραιώνονται με πολύ ταχύτερους ρυθμούς , ενώ καταναλώνουν λιγότερη ενέργεια σε σύγκριση με τις εφαρμογές λογισμικού , που εκτελούνται σε μία state-of-the-art hard-core CPU , η οποία είναι ενσωματωμένη στην ίδια συσκευή FPGA.

Δουλειά πάνω στη δυναμική κατανάλωση ενέργειας των FPGAs συσκευών , που ανήκουν στην οικογένεια Virtex-II [4] , έχει γίνει από τους Li Shang , Alireza S Kaviani και Kusuma Bathala , οι οποίοι αναλύουν τη δυναμική της κατανάλωσης ενέργειας , από τις προγραμματιζόμενες διατάξεις θυρών πεδίου (FPGA) , χρησιμοποιώντας και προσομοιώσεις και μετρήσεις. Εντοπίζουν σημαντικούς πόρους στην αρχιτεκτονική των FPGAs συσκευών και υπολογίζουν πόση ισχύς καταναλώνει ο κάθε ένας χωριστά. Σύμφωνα με τα αποτελέσματά τους , η ισχύς , που καταναλώνεται για τις διαδικασίες της δρομολόγησης , της λογικής και του χρονισμού , είναι 60% , 16% και 14% , αντίστοιχα.

Παραπλήσια εργασία έχει γίνει για τα trade-offs κατανάλωσης ισχύος. Οι Becker , Huebner και Ullmann [3] εξετάζουν ποιά ακριβώς είναι τα trade-offs , οι συμβιβασμοί , κατανάλωσης ισχύος , ανάμεσα στη μέτρηση κατανάλωσης , κατά τον χρόνο εκτέλεσης , συγκεκριμένης εφαρμογής και στη μέτρηση κατανάλωσης του χρόνου αναδιαμόρφωσής της , για διαφορετικές δυναμικά αναδιαρθρώμενες εφαρμογές. Επιπλέον, κάνουν λόγο για τις δυνατότητες και τους περιορισμούς , των σημερινά διαθέσιμων εργαλείων εκτίμησης κατανάλωσης ισχύος και σε σύγκριση με τις ακριβείς μετρήσεις.

Βλέπουμε , λοιπόν , ότι η κατανάλωση ισχύος των επαναπρογραμματίσιμων συσκευών (FPGAs) αποτελεί ένα βασικό θέμα , με το οποίο έχουν ασχοληθεί και άλλοι μηχανικοί. Πιο ειδικά , μας απασχολεί η κατανάλωση ισχύος για τις εφαρμογές ασφαλείας (security applications) , όπου η ανάλυση της ισχύος αποτελεί ένα σημαντικό και αποτελεσματικό μέσο για επιθέσεις. Σκοπός μας , εδώ , είναι να πειραματιστούμε , προκειμένου να επιτευχθεί η βέλτιστη κατανάλωση ισχύος , του εσωτερικού πηρύνα των FPGA κυκλωμάτων , Virtex II Pro , κατά την χρήση ευρέως διαδεδομένων αλγορίθμων ασφαλείας , απευθείας από το hardware υλικό. Για να το πετύχουμε , αυτό , πειραματιζόμαστε με τις παραμέτρους συνθεσης και υλοποιησης , του λογισμικού Xilinx.

Κάθε μέτρηση που παίρνουμε , αντιπροσωπεύει μία σειρά παραμέτρων synthesis και implementation της Xilinx. Οι παράμετροι , που διερευνούμε , για να βρούμε την βέλτιστη κατανάλωση ισχύος , είναι Optimization goal , Optimization effort , Map effort level , Place & route mode και Power reduction. Πιο συγκεκριμένα, μετράμε και συγκρίνουμε , τις τιμές κατανάλωσης ισχύος , των οχτώ ευρέως διαδεδομένων αλγορίθμων ασφαλείας , που έχουμε επιλέξει , για κάθε μία μέτρηση , απευθείας από το hardware υλικό.

Από όσο γνωρίζουμε , αυτή η διπλωματική είναι η πρώτη που ασχολείται με τη δερεύνηση αυτών των παραμέτρων , συνθεσης και υλοποιησης , του λογισμικού Xilinx ISE , και παίρνει τις αντίστοιχες μετρήσεις απευθείας από το hardware.

3. Αλγόριθμοι Ασφαλείας και Σύστημα Virtex-II PRO

Αλγόριθμοι ασφαλείας

Αλγόριθμοι ασφαλείας (security algorithms) είναι οι αλγόριθμοι , που χρησιμοποιούνται για την ασφαλή επικοινωνία και μετάδοση πληροφορίας , μεταξύ δύο ή και περισσότερων τερματικών σταθμών. Χρησιμοποιούν πολύπλοκους μαθηματικούς υποογισμούς , με τους οποίους δεν θα ασχοληθούμε στην συγκεκριμένη περίπτωση. Αντ'αυτού παρουσιάζουμε μία πιο γενική εικόνα , όσο αναφορά στο πώς λειτουργούν.

Στην ουσία , η επίτευξη ασφαλούς επικοινωνίας αποτελείται από δύο μέρη : την κρυπτογράφηση (encryption) και την αυθεντικοποίηση (authentication) .Στην κρυπτογράφηση , οι πιο διαδεδομένοι αλγόριθμοι , που χρησιμοποιούνται είναι οι RSA , DES (Data Encryption Standard) και ο AES (Advanced Encryption Standard). Στην αυθεντικοποίηση , χρησιμοποιείται κατά κύριο λόγο ο MD5 (Message Digest) αλγόριθμος.

Εμείς χρησιμοποιούμε όλους τους παραπάνω αλγόριθμους , καθώς και άλλους , εξίσου διαδεδομένους ,όπως Camellia , 3DES , Basic DES και Basic RSA.

Κρυπτογράφηση

Η κρυπτογράφηση μυστικού κλειδιού (secret key cryptography) βασίζεται στην χρήση μιας μυστικής πληροφορίας , που ονομάζεται κλειδί , για την κρυπτογράφηση και αποκρυπτογράφηση δεδομένων. Αν κάποιος θέλει να αποστείλει σε έναν τρίτο ένα κωδικοποιημένο μήνυμα , τότε χρησιμοποιεί το μυστικό κλειδί για να κρυπτογραφήσει την πληροφορία (στην ορολογία της κρυπτογραφίας η μη κρυπτογραφημένη πληροφορία αναφέρεται ως plaintext). Το αποτέλεσμα της κρυπτογράφησης είναι να προκύψει μια κωδικοποιημένη πληροφορία (που ονομάζεται ciphertext) και η οποία αποστέλεται. Για να μπορέσει κάποιος να ανακτήσει την αρχική πληροφορία, θα πρέπει να γνωρίζει το ίδιο μυστικό κλειδί που χρησιμοποιήθηκε για την κωδικοποίηση της.

Σε γενικές γραμμές υπάρχουν δύο τεχνικές κρυπτογράφησης , που χρησιμοποιούνται για την ασφαλή επικοινωνία: η συμμετρική και η ασύμμετρη.

Στη συμμετρική κρυπτογράφηση, το ίδιο κλειδί χρησιμοποιείται , τόσο για την κρυπτογράφηση , όσο και για την αποκρυπτογράφηση. Αυτό είναι γνωστό ως το ιδιωτικό κλειδί. Σκεφτείτε δύο μέρη , Α και Β , που επιθυμούν να συμμετάσχουν σε μια κρυπτογραφημένη επικοινωνία. Το μέρος Α δημιουργεί ένα ιδιωτικό κλειδί και στέλνει αντίγραφο του στο μέρος Β. Εξ' ου και τα δύο μέρη χρησιμοποιούν αυτό το κλειδί για την κρυπτογράφηση, καθώς και την αποκρυπτογράφηση των μηνυμάτων.

Στην ασύμμετρη κρυπτογράφηση , το μέρος Α παράγει ένα δημόσιο-ιδιωτικό ζεύγος κλειδιού , και στέλνει μόνο το δημόσιο κλειδί στο μέρος Β. Όταν ο Β θέλει να στείλει ένα μυστικό μήνυμα στον Α , κρυπτογραφεί το μήνυμα χρησιμοποιώντας το δημόσιο κλειδί του Α. Όταν ο Α λάβει αυτό το κρυπτογραφημένο μήνυμα , μπορεί να το αποκρυπτογραφήσει , μόνο με το αντίστοιχο ιδιωτικό κλειδί του. Ομοίως, μπορεί να συμβεί και το αντίθετο. Η διαδικασία αυτή είναι επίσης γνωστή και ως ΥΔΚ ή Υποδομή Δημόσιου Κλειδιού.

Αυθεντικοποίηση

Στην αυθεντικοποίηση , τα δύο μέρη Α και Β , προσπαθούν να αναγνωρίσουν το ένα το άλλο , με την χρήση ψηφιακών πιστοποιητικών. Το πιστοποιητικό , αυτό , αποτελείται από μία ψηφιακή υπογραφή , η οποία παράγεται χρησιμοποιώντας , κατα κύριο λόγο DSA και MD5 αλγόριθμους. Εμείς ασχολούμαστε με τον hashing αλγόριθμο MD5.

Πάμε να δούμε , λίγο καλύτερα , τους αλγορίθμους ασφαλείας , που χρησιμοποιήσαμε.

AES

Το πρότυπο κρυπτογράφησης AES (Advanced Encryption Standard) έχει προτυποποιηθεί από το NIST (National Institute of Technology) τον Νοέμβριο του 2001, αντικαθιστώντας το πρότυπο DES (Data Encryption Standard) και πλέον αποτελεί τον προτεινόμενο αλγόριθμο για εφαρμογές κρυπτογράφησης.

Το πρότυπο AES περιγράφει μια συμμετρική διαδικασία κρυπτογράφησης. Το πρότυπο υποστηρίζει την χρήση κλειδιών μήκους 128 , 192 και 256 bits. Ανάλογα με το ποιο μήκος κλειδιού χρησιμοποιείται , συνήθως χρησιμοποιείται η συντόμευση AES-128 , AES-192 και AES-256 αντίστοιχα. Εμείς εργαζόμαστε με το πρότυπο AES-128. Ανεξάρτητα από το μήκος κλειδιού , βεβαια , ο αλγόριθμος επενεργεί πάνω σε μπλοκ δεδομένων μήκους 128 bits. Η διαδικασία κρυπτογράφησης είναι επαναληπτική. Αυτό σημαίνει ότι σε κάθε μπλοκ δεδομένων γίνεται μια επεξεργασία η οποία επαναλαμβάνεται τόσες φορές , ανάλογα με το μήκος κλειδιού. Κάθε επανάληψη ονομάζεται γύρος (round). Στον πρώτο γύρο επεξεργασίας ως είσοδος είναι ένα plaintext μπλοκ και το αρχικό κλειδί, ενώ στους γύρους που ακολουθούν ως είσοδος είναι το μπλοκ που έχει προκύψει από τον προηγούμενο γύρο καθώς και ένα κλειδί που έχει παραχθεί από το αρχικό , με βάση κάποια διαδικασία που ορίζει ο αλγόριθμος. Το τελικό προϊόν της επεξεργασίας είναι το κρυπτογραφημένο μπλοκ (ciphertext). Το μπλοκ αυτό πρέπει να σημειωθεί ότι έχει ακριβώς το ίδιο μέγεθος (128 bits) με το plaintext μπλοκ.

Είπαμε ότι ο AES τροφοδοτείται με ακολουθίες από bits των 128 bits (μπλοκ) καθώς και από κλειδιά, που μπορεί να έχουν μέγεθος 128, 192 ή 256 bits. Τα κλειδιά αυτά ονομάζονται κλειδιά κρυπτογράφησης (cipher keys) για να διαχωριστούν από τα κλειδιά που παράγονται κατά την λειτουργία του αλγορίθμου.

Τόσο κατά την διάρκεια της διαδικασίας κρυπτογράφησης όσο και της αποκρυπτογράφησης , κάθε γύρος επεξεργασίας αποτελείται από μια σειρά μετασχηματισμών σε επίπεδο byte. Όλες οι λειτουργίες που επιτελεί ο αλγόριθμος γίνονται πάνω σε ένα διδιάστατο πίνακα που αποκαλείται Κατάσταση (State). Για την ακρίβεια, χρησιμοποιούνται 4 τύποι μετασχηματισμών :

1. *Ο Μετασχηματισμός SubBytes :* μετασχηματισμός αντικατάστασης bytes χρησιμοποιώντας κάποιον σχετικό πίνακα αντικατάστασης
2. *Ο Μετασχηματισμός ShiftRows :* μηχανισμός ολίσθησης των bytes της State κατά διαφορετικά offsets
3. *Ο Μετασχηματισμός MixColumns :* διαδικασία ανάμειξης των bytes της State
4. *Ο Μετασχηματισμός AddRoundKey :* πρόσθεση ενός κλειδιού στην State

Οι μετασχηματισμοί της διαδικασίας κρυπτογράφησης μπορούν να αντιστραφούν και να τοποθετηθούν σε αντίστροφη σειρά , ώστε να παραχθεί μια διαδικασία που θα αποκρυπτογραφεί ένα ciphertext του AES. Έτσι , όπως και κατά την κρυπτογράφηση, υπάρχουν τέσσερις διακριτοί μετασχηματισμοί που επενεργούν πάνω στην State κατά την αποκρυπτογράφηση.Πρόκειται για τους InvShiftRows , InvSubBytes , InvMixColumns και AddRoundKey.

DES

Ο αλγόριθμος DES (Data Encryption Standard) αναπτύχθηκε από την IBM. Οι αλγόριθμοι της σειράς DES είναι οι πλέον χρησιμοποιούμενοι μέχρι σήμερα αλγόριθμοι, αν και πια αντικαθιστούνται από τον AES. Είναι μια συμμετρική τεχνική κρυπτογράφησης , δηλαδή με ένα κλειδί , που κρυπτογραφεί μηνύματα σε κομμάτια των 64-bit. Αν και το πραγματικό μέγεθος του κλειδιού είναι 64-bits , χρησιμοποιεί μόνο 56 bit για κρυπτογράφηση / αποκρυπτογράφηση. Τα υπόλοιπα 8 bit χρησιμοποιούνται για να ελεγχθεί εάν το κλειδί έχει αλλάξει κατά τη διάρκεια της μεταφοράς του , είτε τυχαία είτε εσκεμμένα.

Τόσο το 56-bit κλειδί , όσο και τα 64-bit δεδομένα περνούν από μια διαδικασία μεταλλαγής και μετασχηματισμού. Στόχος είναι η δημιουργία δεκαέξι υποκλειδιών , από 48-bits το καθένα , χρησιμοποιώντας το 56-bit κλειδί και τα 64-bit δεδομένα σε 16 βρόγχους. Η διαδικασία είναι επαναληπτική. Κάθε επανάληψη ονομάζεται γύρος (round) ή βρόγχος (loop). Ακολουθεί η εξήγηση του ενός βρόχου.

Το 56-bit κλειδί αλλάζει αρχικά, σύμφωνα με έναν πίνακα μεταλλαγής κλειδιών. Οι πίνακες μεταλλαγής αλλάζουν τις θέσεις των bits. Τότε το αλλαγμένο κλειδί χωρίζεται σε δύο μισά των 28-bit. Έπειτα τα bits σε κάθε ήμισυ μετατοπίζονται κατά δύο θέσεις. Η μετατόπιση (shifting) γίνεται σε όλους τους βρόγχους , εκτός από τους πρώτο , δεύτερο , ένατο και δέκατο έκτο βρόγχο. Στη συνέχεια, από αυτά τα δύο μισά , ένα 48-bit κλειδί επιλέγεται σύμφωνα με ένα πίνακα συμπίεσης μεταλλαγής.

Τα δεδομένα των 64-bits χωρίζονται σε δύο 32-bit μισά , που ονομάζονται L , το αριστερά μισό και R , το δεξιά μισό. Τώρα το R υποβάλλεται σε έναν άλλο πίνακα μεταλλαγής , που καλείται πίνακας επέκτασης μεταλλαγής , όπου κάθε 32-bit μπλοκ επεκτείνεται σε 48 bit (με γέμισμα και επανάληψη κάποιων bits).

Μετά από αυτό , το R περνά από πύλη XOR (προφέρεται Αποκλειστικό Ή , η οποία είναι μια ψηφιακή λειτουργική πύλη), με το 48-bit υποκλειδί , που προέρχεται από το 56-bit κλειδί. Το αποτέλεσμα αυτής της προσαγωγής δίνεται σε 8 πίνακες μεταλλαγής γνωστοί ως S-boxes. Κάθε S-box, δέχεται 6 bits (8x6 = 48) και δημιουργεί μία έξοδο των 4-bit.Η συνολική παραγωγή από τα οκτώ S-boxes συνδυάζεται στη συνέχεια, με αποτέλεσμα ένα κομμάτι των 32-bit. Αυτό το κομμάτι των 32-bit τροφοδοτείται στη συνέχεια σε έναν άλλο πίνακα μεταλλαγής , που ονομάζεται P-box. Το P-box παράγει επίσης ένα κομμάτι των 32-bit , το οποίο στη συνέχεια περνάει από XOR με τον L. Τέλος, εάν αυτός δεν είναι ο δέκατος έκτος γύρος , το L γίνεται R και το αντίστροφο. Αυτή η εναλλαγή ονομάζεται μετασχηματισμός.

Τα δεδομένα των 64-bit , λοιπόν , υπόκεινται σε ακόμα 15 τέτοιους γύρους , για την κρυπτογράφηση.Κατά τη διάρκεια της αποκρυπτογράφησης επαναλαμβάνεται η αντίθετη διαδικασία. Δεδομένου ότι ο αλγόριθμος περιλαμβάνει μόνο XORing και αλλαγές στις θέσεις των bits , ο DES είναι σχετικά πιο γρήγορος .

3DES

Ο αλγόριθμος Triple DES ή 3DES επιτυγχάνει μεγαλύτερη δύναμη , με την κρυπτογράφηση των δεδομένων ή/και του μηνύματος *τρεις* φορές , χρησιμοποιώντας μεν τον ίδιο αλγόριθμο DES , αλλά κάθε φορά διαφορετικό κλειδί.

BASIC DES

Ο αλγόριθμος κρυπτογράφησης Basic DES είναι στην ουσία μία μικρή και γρήγορη υλοποίηση του γνωστού και προαναφερθέντος αλγόριθμου DES. Εκτελεί συναρτήσεις κρυπτογράφησης και αποκρυπτογράφησης του απλού DES με μέθοδο ECB (*Electronic Codebook)* , και έχει την δυνατότητα να δέχεται νέο κλειδί για κάθε διεργασία χωρίς κόστος στην απόδοση.

Ο τρόπος λειτουργίας ECB είναι ο απλούστερος και ο πλέον προφανής. Το κλειδί χρησιμοποιείται για την κρυπτογράφηση κάθε μπλοκ δεδομένων του plaintext. Κατά συνέπεια με την χρήση του ίδιου κλειδιού, το ίδιο plaintext μπλοκ θα μετατρέπεται πάντα στο ίδιο ciphertext μπλοκ. Είναι ο πλέον κοινός τρόπος λειτουργίας των κρυπτογράφων μπλοκ γιατί είναι ο απλούστερος και άρα ο πιο εύκολα υλοποιήσιμος και συνάμα ο πιο γρήγορος καθώς δεν χρησιμοποιείται κάποιου είδους

ανατροφοδότηση. Μειονέκτημα του είναι ότι είναι ο πιο ευάλωτος τρόπος κρυπτογράφησης σε επιθέσεις τύπου brute-force (ως επίθεση brute-force θεωρείται η προσπάθεια εύρεσης του μυστικού κλειδιού με την εξαντλητική δοκιμή πιθανών κλειδιών).

Αναφέραμε πριν , ότι στον αλγόριθμο DES, αν και το πραγματικό μέγεθος του κλειδιού είναι 64-bits , χρησιμοποιεί μόνο 56 bit για κρυπτογράφηση / αποκρυπτογράφηση. Τα υπόλοιπα 8 bit χρησιμοποιούνται για να ελεγχθεί εάν το κλειδί έχει αλλάξει κατά τη διάρκεια της μεταφοράς του. Με την μέθοδο ECB κάθε μπλοκ επεξεργάζεται χωριστά από τα υπόλοιπα , χωρίς δηλαδή να σχετίζεται με το προηγούμενο ή το ακόλουθο μπλοκ.Η διαδικασία αυτή έχει επιγραμματικά ώς εξής:

- Αρχικά , τα bits του μπλοκ που περιέχει το μήνυμα ανταλλάσσονται με βάση ένα συγκεκριμένο πρότυπο. Εκτελούνται 16 ίδιοι γύροι για την διαδικασία της κρυπτογράφησης . Για κάθε γύρο:
- Το 64- bits μπλοκ εισόδου χωρίζεται σε δύο 32- bits μπλοκ , που ονομάζονται αριστερό (left) και δεξί (right).
- Το δεξί μπλοκ πάει στην έξοδο κατευθείαν , χωρίς περετέρω επεξεργασία ως το αριστερό μπλοκ του επόμενου γύρου.
- Το δεξί μπλοκ επεκτείνεται από 32 bits σε 56 bits και περνάει από xor μαζί με το ισχύον υποκλειδί (round key).
- Η τελική αλληλουχία των 56 bits αντικαθίσταται σε γκρουπ των 6 bits με S-boxes των 4 bits , δίνοντας ένα τελικό 32- bits αποτέλεσμα.
- Τα bits αυτής της αλληλουχίας ανταλλάσσονται σύμφωνα με συγκεκριμένο πρότυπο ανταλλαγής και το αποτέλεσμα περνά από xor με το αριστερό μπλοκ.
- Το τελικό 32-bits μπλοκ είναι η έξοδος καθώς και το δεξί μπλοκ εισόδου , για τον επόμενο γυρο.

Αφού ολοκληρωθούν και οι 16 κύκλοι έχουμε μία τελική μεταλλαγή των bits του μπλοκ της εξόδου , με βάση πάντα συγκεκριμένο πίνακα , όπου και παράγεται το τελικό κρυπτογραφημένο κείμενο (cyphertext).

Η αποκρυπτογράφηση είναι στην ουσία ή ίδια διαδικασία με την κρυπτογράφηση , μόνο που τα υποκλειδία εφαρμόζονται αντίστροφα.

RSA

Ο RSA αλγόριθμος , που ονομάστηκε έτσι από τους δημιουργούς του (Rivest, Shamir, Adleman), είναι ένας ασύμμετρος αλγόριθμος ή αλγόριθμος δημοσίου κλειδιού.

Στο πλαίσιο αυτό, το δημόσιο-ιδιωτικό ζεύγος κλειδιού έχει ένα σταθερό μήκος σε bits, το οποίο μπορεί να αποφασιστεί κατά τη στιγμή της δημιουργίας του , όπως π.χ. 512 , 768 , 1.024 , 2.096 , όπου τα υψηλότερα νούμερα , αντιστοιχούν και σε ισχυρότερη κρυπτογράφηση. Όταν το δημόσιο κλειδί παράγεται , αποτελείται από το μέγεθος του κλειδιού και ένα θετικό ακέραιο που ονομάζεται δημόσιος εκθέτης , ο οποίος έχει κάποιες τυπικές κατ 'αποκοπή τιμές. 'Οταν το ιδιωτικό κλειδί παράγεται περιλαμβάνει αυτά τα δύο , καθώς και έναν ιδιωτικό εκθέτη και δύο πρώτους αριθμούς. Οι δύο πρώτοι αριθμοί προέρχονται κατά τέτοιο τρόπο , ώστε το προϊόν τους να είναι ίσο με το μέγεθος του κλειδιού. Στον RSA, το μέγεθος του κλειδιού είναι το ίδιο και για τα δύο κλειδιά--δημόσιο και ιδιωτικό.Ο ιδιωτικός εκθέτης στο ιδιωτικό κλειδί υπολογίζεται από το δημόσιο εκθέτη και τους δύο πρώτους αριθμούς.

Άπαξ και τα κλειδιά έχουν δημιουργηθεί , είναι έτοιμα για την κρυπτογράφηση ή/και αποκρυπτογράφηση δεδομένων ή/και μηνυμάτων. Ο αριθμός των bits στο μήνυμα που υφίσταται κρυπτογράφηση , πρέπει να είναι μικρότερος ή ίσος με το μέγεθος του κλειδιού. Αν δεν είναι, τότε το μήνυμα διασπάται σε ξεχωριστά , μικρότερα , τμήματα , τα οποία στη συνέχεια κρυπτογραφούνται. Εάν το μέγεθος του μηνύματος είναι μικρότερο από το μέγεθος του κλειδιού ,τότε με κάποια επιπλέον bits , «παραγεμίζουμε» το μήνυμα.

Το κωδικοποιημένο μήνυμα δημιουργείται χρησιμοποιώντας το ίδιο το αρχικό μήνυμα , τον δημόσιο εκθέτη , και τις πληροφορίες για το μέγεθος του κλειδιού , ληφθείσες από το δημόσιο κλειδί. Όταν το κωδικοποιημένο μήνυμα ληφθεί στο άλλο άκρο , ο ιδιωτικός εκθέτης και το μέγεθος του κλειδιού χρησιμοποιούνται για την αποκρυπτογράφηση. Δεδομένου ότι ο ιδιωτικός εκθέτης υπολογίζεται χρησιμοποιώντας το δημόσιο εκθέτη , μόνο το σωστό ιδιωτικό κλειδί μπορεί να αποκρυπτογραφήσει το μήνυμα. Η κρυπτογράφηση και η αποκρυπτογράφηση του μηνύματος απαιτεί πολλούς εκθετικούς υπολογισμούς. Άρα , ο RSA ή αλγόριθμοι σαν αυτόν , με δημόσιο κλειδί κρυπτογράφησης είναι αργοί.

Εμείς χρησιμοποιούμε μία απλή εκδοχή του RSA, η οποία ενώ μπορεί να δεχτεί εως 1024 bits σε μέγεθος κλειδιού και μηνύματος , είναι διαμορφωμένο να λειτουργεί στα 32 bits. Πρόκειται για τον αλγόριθμο Basic RSA.

CAMELLIA

Στην κρυπτογραφία , ο αλγόριθμος Camellia είναι μία συμμετρική διαδικασία κρυπτογράφησης , που έχει αξιολογηθεί θετικά από πολλές οργανώσεις , συμπεριλαμβανομένων και του πρότζεκτ NESSIE της Ευρωπαικής Ένωσης και του ιαπωνικού πρόζεκτ CRYPTREC. Αναπτύχθηκε από κοινού από την Mitsubishi και την NTT το 2000 και έχει παρόμοια στοιχεία με το σχεδιασμό προγενέστερων αλγορίθμων κρυπτογράφησης , από αυτές τις εταιρείες.

Ο Camellia έχει ένα μέγεθος μπλοκ των 128 bits και μπορεί να χρησιμοποιήσει 128-bit , 192-bit ή 256-bit κλειδιά --το ίδιο περιβάλλον εργασίας όπως ο AES. Είναι ένας κρυπτογράφος δομής Feistel με είτε 18 γύρους (εφόσον το κλειδί είναι 128 bits) ή 24 γύρους (εφόσον το κλειδί είναι 192 ή 256 bits). Κάθε έξι γύρους , εφαρμόζεται ένα στρώμα λογικού μετασχηματισμού: η λεγόμενη "FL-function" ή η αντίστροφή της . Ο Camellia χρησιμοποιεί τέσσερα 8 x 8-bit S-boxes με είσοδο και έξοδο γραμμικούς μετασχηματισμούς και λογικές πράξεις. Επίσης χρησιμοποιείται key whitening – τεχνική που αποσκοπεί στην αύξηση της ασφάλειας μίας συμμετρικής διαδικασίας κρυπτογράφησης .

Επιπλεον , έχουμε διάχυση , ιδιότητα που σημαίνει ότι τα bits εξόδου πρέπει να εξαρτώνται από τα bits εισόδου με έναν πολύ περίπλοκο τρόπο. Σε μία καλή κρυπτογράφηση με τη διάδοση , αν ένα κομμάτι του plaintext έχει αλλάξει , τότε η ciphertext πρέπει να αλλάξει τελείως. Το στρώμα διάχυσης , εδώ χρησιμοποιεί ένα γραμμικό μετασχηματισμό που βασίζεται σε έναν πίνακα MDS με αριθμό υποκαταστήματος 5.

Ο Camellia είναι ένας από τους αλγόριθμους κρυπτογράφησης που μπορούν να οριστούν πλήρως από ελάχιστα συστήματα με πολυμεταβλητά πουώνυμα. Τα S-boxes του Camellia (καθώς και του AES) μπορούν να περιγραφούν από ένα σύστημα 23 δευτεροβάθμιων εξισώσεων σε 80 όρους. Το βασικό σχήμα μπορεί να περιγραφεί από 1120 εξισώσεις σε 768 μεταβλητές χρησιμοποιώντας 3328 γραμμικούς και τετραγωνικούς όρους. Το σύνολο του μπλοκ κρυπτογράφησης μπορεί να περιγραφεί από 5104 εξισώσεις σε 2816 μεταβλητές χρησιμοποιώντας 14592 γραμμικούς και τετραγωνικούς όρους. Συνολικά, απαιτούνται 6224 εξισώσεις σε 3584 μεταβλητές που χρησιμοποιούν 17920 γραμμικούς και τετραγωνικούς όρους.

Θεωρητικά , τέτοιες ιδιότητες θα μπορούσαν να καταστήσουν δυνατό το «σπάσιμο» του Camellia (και του AES) , χρησιμοποιώντας μια αλγεβρική επίθεση, όπως η Extended Sparse Linearisation (με την προϋπόθεση ότι η επίθεση γίνεται εφικτή). Κάτι τέτοιο , όμως δεν πρόκειται να μας απασχοήσει εδώ.

Εμείς χρησιμοποιούμε δύο διαφορετικές εκδόσεις του Camellia. Η πρώτη δέχεται κλειδί μεγέθους μόνο 128- bits , ενώ η δεύτερη δέχεται όλα τα μεγέθη. Στην ουσία εργαζόμαστε με δύο αλγόριθμους-παρακλάδια , που ονομάσαμε Camellia_1 και Camellia_2 , αντίστοιχα.

MD5

Ο MD5 (Message Digest) είναι ένας αλγόριθμος , που χρησιμοποιείται για τη δημιουργία ψηφιακών υπογραφών. Η έξοδος του MD5 είναι ένα «μήνυμα χώνεψης» (Message Digest) , το οποίο μπορεί να χρησιμοποιηθεί για τον έλεγχο ταυτότητας του ιδιοκτήτη ενός ιδιωτικού κλειδιού.

Ο αλγόριθμος MD5 λαμβάνει ένα μήνυμα και καλείται να ελέγξει , αν το μέγεθος του είναι 448- bits. Εάν δεν είναι, τότε το συμπληρώνει με επιπλέον bits. Στη συνέχεια , παίρνει και πάλι το

αρχικό μήνυμα και το μετατρέπει σε 64 bits. Αυτά , προστίθενται στη συνέχεια με τα 448 bits , για να δώσει ένα μπλοκ των 512 bits. Αυτό το μπλοκ στη συνέχεια το σπάει σε 32 μπλοκ μηνυμάτων ,όπου καθένα από αυτά αποτελείται από 16-bit. Ένας βρόχος ξεκινάει μέσα στον οποίο , κάθε ένα από αυτά τα 32 μπλοκ , τίθεται σε επεξεργασία. Έξω από το βρόχο , λαμβάνονται τέσσερις διαφορετικές μεταβλητές των 32-bit -Α, Β, Γ και Δ-με συγκεκριμένες τιμές .

Στη συνέχεια, οι τιμές των τεσσάρων αυτών μεταβλητών-Α, Β, Γ, Δ-αντιγράφονται σε άλλες τέσσερις διαφορετικές μεταβλητές π.χ. α, β, γ, και δ. 'Επειτα , εντός του βρόχου υπολογίζονται νέες τιμές για τα α, β, γ, δ , χρησιμοποιώντας και τα 32 μπλοκ των 16-bit , αλλά και τις τιμές των αρχικών α, β, γ, δ . Μια διαφορετική εξίσωση χρησιμοποιείται , για κάθε μία από αυτές τις τέσσερις μεταβλητές. Τώρα Α, Β, Γ και Δ προσαυξάνονται με τις νέες τιμές των α, β, γ, και δ.

Τέλος Α, Β, Γ και Δ , που συνολικά είναι 128 bit (32x4) , είναι ο υπολογισμός του hash , ο οποίος καλείται επίσης και μήνυμα χώνεψης (message digest).

Ανακεφαλαιώνοντας , οι αλγόριθμοι που χρησιμοποιούμε στην διπλωματική , αυτή , είναι οι ακόλούθοι : Basic RSA , Basic DES , Camellia_1 , Camellia_2 , Triple DES , AES , DES και MD5.

Για κάθε έναν από τους παραπάνω αλγόριθμους ακολουθείται η διαδικασία design flow , για ένα FPGA , που περιγράφεται παραπάνω και φαίνεται στο σχήμα 1. Μετά το «πάντα» απαραίτητο debugging και αφού έχουμε σιγουρευτεί απολύτως , ότι όλα λειτουργούν σωστά , δημιουργούμε ένα top level source με εισόδους clock και reset και εξόδους 4 leds. Το αρχείο , αυτό μας χρησιμεύει στον έλεγχο της ορθής λειτουργίας του κάθε αλγόριθμου. Έπειτα , κάνουμε synthesize και implement , με διαφορετικές κάθε φορά παραμέτρους , τις οποιες θα αναλύσουμε αργότερα. Τέλος , παράγουμε το αρχείο . bit , με το οποίο θα προγραμματίσουμε το FPGA μας και το φορτώνουμε στην Virtex-II Pro συσκευή μας.

Virtex-II Pro

Τα τελευταία χρόνια έχει επικρατήσει η συνδυαστική αρχιτεκτονική προσέγγιση στα FPGAs. Με αυτό εννοούμε το συνδυασμό των «λογικών μπλοκ» και των παραδοσιακών ενδοσυνδυασμών των FPGAs , με ενσωματωμένους μικροεπεξεργαστές και σχετικά περιφερειακά συστήματα , ώστε να δημιουργηθεί ένα πλήρες «σύστημα σε ένα προγραμματιζόμενο τσιπ». Το έργο αυτό αντικατοπτρίζει την αρχιτεκτονική από τους Ron Perlof και Hana Potash της Burroughs Advanced Systems Group , οι οποίοι κατάφεραν να συνδυάσουν μία αναδιαρθρώσιμη αρχιτεκτονική Κεντρικής Μονάδας Επεξεργασίας (reconfigurable CPU architecture) , σε ένα μόνο chip που ονομάζεται SB24. Αυτό το έργο έγινε το 1982. Παραδείγματα τέτοιων υβριδικων τεχνολογιών μπορούν να βρεθούν στις συσκευές

Virtex-II PRO (είναι η συσκευή που χρησιμοποιούμε και εμείς) και Virtex-4 της Xilinx. Πρόκειται για συσκευές, οι οποίες περιλαμβάνουν έναν ή και περισσότερους επεξεργαστές PowerPC (Power Performance Computing—PPC) , ενσωματωμένους στο ύφασμα λογικής του FPGA.

Η Xilinx,inc έχει δύο κύριες FPGA σειρές : την Virtex και την Spartan. Η Virtex είναι σειρά υψηλής απόδοση , ενώ η Spartan είναι σειρά υψηλού όγκου.

Η σειρά Spartan στοχεύει σε εφαρμογές με χαμηλό αποτύπωμα ενέργειας , ακραία ευαισθησία στο κόστος και με υψηλό όγκο , όπως π.χ. παρουσιάσεις , αποκωδικοποιητές , ασύρματους routers και άλλες εφαρμογές.

Η σειρά Virtex των FPGAs έχει ενσωματωμένα χαρακτηριστικά , όπως είναι ο ενσύρματος και ασύρματος εξοπλισμός υποδομής , προηγμένος ιατρικός εξοπλισμός , χαρακτηριστικά δοκιμών και μετρήσεων , καθώς και χαρακτηριστικά αμυντικών συστημάτων. Εκτός από την λογική FPGA , η σειρά Virtex περιλαμβάνει ενσωματωμένο εξοπλισμό σταθερής λειτουργίας (fixed function hardware) , για ευρέως διαδεδομένες και χρησιμοποιούμενες λειτουργίες , όπως πολλαπλασιαστές , μνήμες , serial πομποδέκτες και πυρήνες μικροεπεξεργαστών.

Βασικά , ένα Virtex FPGA αποτελείται από δύο στρώματα (layers) και επιπλέον λογική αναδιαμόρφωσης και ελέγχου , που χειρίζεται τη φόρτωση του διαμορφωμένου bitstream (loading configuration bitstream) και τη διανομή των διαμορφωμένων δεδομένων (bitstream data) σε συγκεκριμένες θέσεις του δεύτερου στρώματος. Το πρώτο στρώμα περιέχει το αναδιαρθρώσιμο υλικό hardware , όπως είναι τα λογικά μπλοκ (CLBs), RAM –μπλοκ , I / O-μπλοκ και τους επαναρυθμίσιμους πόρους καλωδίωσης . Όλα μπλοκ του ίδιου τύπου (I / O-μπλοκ εξαιρουμένων) έχουν εναρμονιστεί σε στήλες. Το δεύτερο στρώμα , που ταιριάζει με το πρώτο , είναι επίσης οργανωμένο σε στήλες. Κάθε στήλη , το πλάτος των οποίων εξαρτάται από τις καλυμένες block-στήλες , από το πρώτο στρώμα , αποτελείται από ένα σύνολο υποστηλών (sub-columns) του ενος bit , που ονομάζονται πλαίσια (frames). Ένα πλαίσιο αποτελεί το μικρότερο μέγεθος αναδιαρθρώσιμης πληροφορίας , που μπορεί να γραφτεί σε ένα FPGA και κάθε πλαίσιο περιέχει κομμάτια (fractions) της αναδιαρθρώσιμης πληροφορίας που χρειάζεται , για τον προγραμματισμό των μπλοκ , που έχουν ανατεθεί στις στήλες. Έτσι, αν ένα μπλοκ από μια στήλη πρέπει να αναδιαρθρωθεί , τότε πρέπει να ξαναγραφτούν και όλα τα υπόλοιπα μπλοκ της ίδιας στήλης. Αυτός ο περιορισμός οδηγεί στο γεγονός , ότι μία μερική αναδιαμόρφωση των εφαρμογών μπορεί να γίνει μόνο σε γκρουπ από διαδοχικές στήλες.

Οι οικογένειες Virtex-II Pro και Virtex-II Pro X περιέχουν πλατφόρμες FPGAs , για designs , τα οποία βασίζονται σε IP πυρήνες (Intellectual property core-- IP cores) και προσαρμοσμένες ενότητες. Η οικογένεια ενσωματώνει multi-gigabit πομποδέκτες και μπλοκ της CPU PowerPC στην FPGA σειρά αρχιτεκτονικής Virtex-II Pro. Παρέχει ολοκληρωμένες λύσεις για τηλεπικοινωνίες , ασύρματες εφαρμογές , δικτύωση , βίντεο και εφαρμογές DSP (Digital signal processor).

Εμείς , πιο συγκεκριμένα , χρησιμοποιούμε συσκευή Virtex-II Pro , γιατί είναι από τις λίγες FPGA , όπου μπορούμε να απομονώσουμε τον εσωτερικό πηρύνα της και να μετρήσουμε την κατανάλωση ισχύος του.

Συνδυάζοντας ένα ευρύ φάσμα ευέλικτων χαρακτηριστικών και IP πυρήνων , η οικογένεια Virtex-II Pro ενισχύει τις δυνατότητες σχεδιασμού προγραμματιζόμενης λογικής και αποτελεί μία σεβαστή εναλλακτική λύση των mask-programmed gate arrays.

Στις οικογένειες Virtex-II Pro και Virtex-II Pro X ανήκουν οι παρακάτω συσκευές , με τα χαρακτηριστηκά τους , όπως ακριβώς φαίνεται στον πίνακα A:

Table **A** *:* **Virtex-II Pro / Virtex-II Pro X FPGA Family Members**

Device[1]	RocketIO Transceiver Blocks	PowerPC Processor Blocks	Logic Cells[2]	CLB (1 = 4 slices = max 128 bits)		18 X 18 Bit Multiplier Blocks	Block SelectRAM+		DCMs	Maximum User I/O Pads
				Slices	Max Distr RAM (Kb)		18 Kb Blocks	Max Block RAM (Kb)		
XC2VP2	4	0	3,168	1,408	44	12	12	216	4	204
XC2VP4	4	1	6,768	3,008	94	28	28	504	4	348
XC2VP7	8	1	11,088	4,928	154	44	44	792	4	396
XC2VP20	8	2	20,880	9,280	290	88	88	1,584	8	564
XC2VPX20	8[4]	1	22,032	9,792	306	88	88	1,584	8	552
XC2VP30	8	2	30,816	13,696	428	136	136	2,448	8	644
XC2VP40	0[3], 8, or 12	2	43,632	19,392	606	192	192	3,456	8	804
XC2VP50	0[3] or 16	2	53,136	23,616	738	232	232	4,176	8	852
XC2VP70	16 or 20	2	74,448	33,088	1,034	328	328	5,904	8	996
XC2VPX70	20[4]	2	74,448	33,088	1,034	308	308	5,544	8	992
XC2VP100	0[3] or 20	2	99,216	44,096	1,378	444	444	7,992	12	1,164

Notes:
1. -7 speed grade devices are not available in Industrial grade.
2. Logic Cell ≈ (1) 4-input LUT + (1)FF + Carry Logic
3. These devices can be ordered in a configuration without RocketIO transceivers.
4. Virtex-II Pro X devices equipped with RocketIO X transceiver cores.

Κάθε μία συσκευή μπορεί να συνδυαστεί με κάποια συγκεκριμένα πακέτα (packages) , όπως φαίνεται και στον πίνακα B. Πακέτα στους ηλεκτρονικούς υπολογιστές λέμε τον τύπο file format , όπου προγράμματα λογισμικού «πακετάρονται» μαζί με υλικό εγκατάστασης. Έχουμε πακέτα τύπου ball grid array (BGA) [1] , τα γνωστά πακέτα αλληλοσύνδεσης wire-bond (FG/FGG packages) [1] , και επιπλέον πακέτα αλληλοσύνδεσης τύπου flip-chip (FF packages) [1]. Η κατασκευή με διασύνδεση Flip-chip υποστηρίζει περισσότερες I/O από ότι είναι δυνατόν στις wire-bond εκδόσεις παρόμοιων πακέτων , παρέχοντας έτσι έναν υψηλό αριθμό PIN και καλύτερη κατανάωση ισχύος.

Table B : **Virtex-II Pro Device/Package Combinations and Maximum Number of Available I/Os**

Package(1)	FG256/ FGG256	FG456/ FGG456	FG676	FF672	FF896	FF1152	FF1148	FF1517	FF1704	FF1696
Pitch (mm)	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Size (mm)	**17 x 17**	**23 x 23**	**26 x 26**	**27 x 27**	**31 x 31**	**35 x 35**	**35 x 35**	**40 x 40**	**42.5 x 42.5**	**42.5 x 42.5**
XC2VP2	140/4	156/4		204/4						
XC2VP4	140/4	248/4		348/4						
XC2VP7		248/8		396/8	396/8					
XC2VP20			404/8		556/8	564/8				
XC2VPX20					552/8(2)					
XC2VP30			416/8		556/8	644/8				
XC2VP40			416/8			692/12	804/0(3)			
XC2VP50						692/16	812/0(3)	852/16		
XC2VP70								964/16	996/20	
XC2VPX70									992/20(2)	
XC2VP100									1,040/20	1,164/0(3)

Notes:
1. Wirebond packages FG256, FG456, and FG676 are also available in Pb-free versions FGG256, FGG456, and FGG676.
2. Virtex-II Pro X device is equipped with RocketIO X transceiver cores.
3. The RocketIO transceivers in devices in the FF1148 and FF1696 packages are not bonded out to the package pins.

Εμείς χρησιμοποιούμε την συσκευή XC2VP30 , της οικογένειας Virtex-II Pro, σε συνδυασμό με το πακέτο FF896 , όπως απεικονίζεται από κάτω , στην Figure 1.

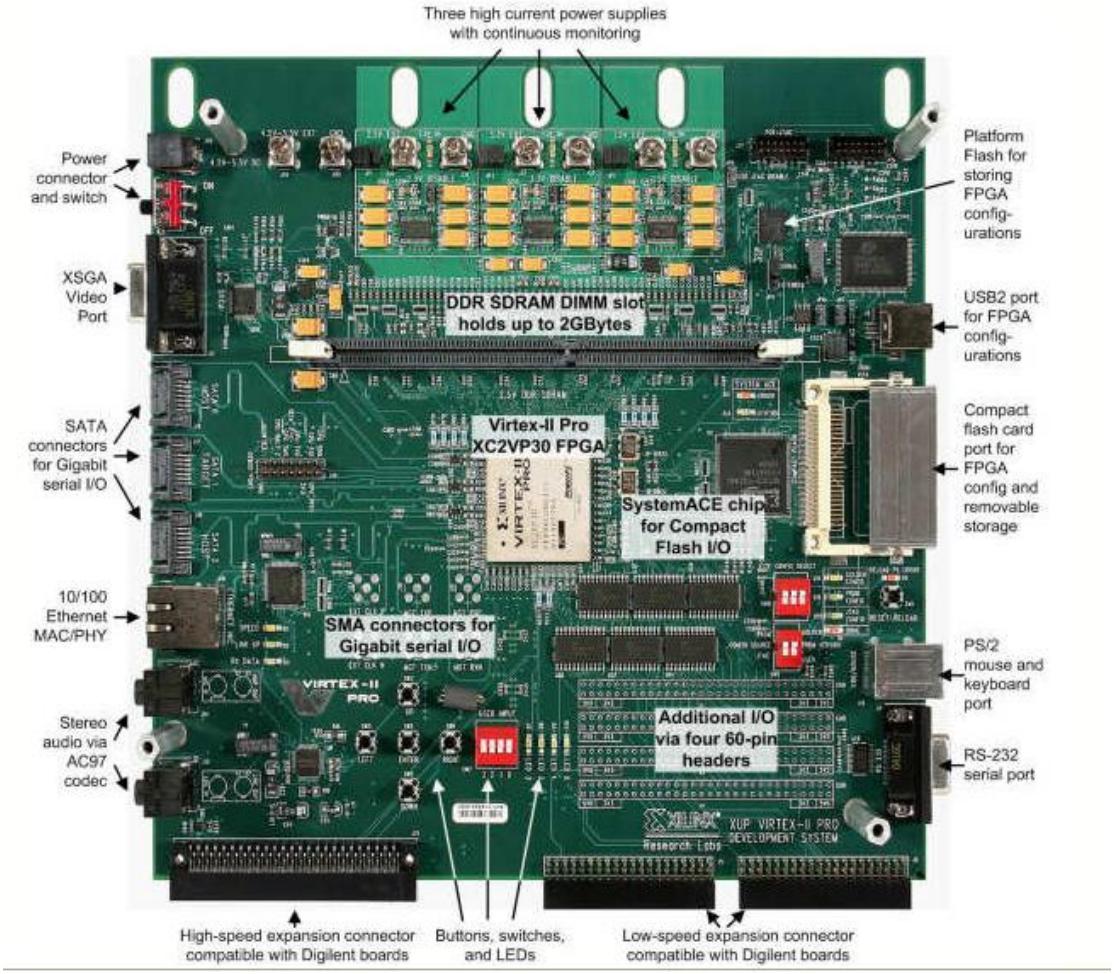


Figure 1.XUP Virtex-II PRO Development System

4. Μέτρηση Κατανάλωσης Ισχύος

Η μέτρηση κατανάλωσης ισχύος , που μας απασχολεί στην συγκεκριμένη διπλωματική , είναι άρρηκτα συνδεδεμένη με την μέτρηση καταναλωσης ρεύματος , την οποία θα αναλύσουμε σε αυτό το κεφάλαιο. Αργότερα , δείχνουμε τη διαδικασία , με την οποία μετράμε την κατανάλωση ισχύος , μέσω της κατανάλωσης ρεύματος , για τις διάφορες εφαρμογές ασφαλείας , που μας ενδιαφέρουν.

Η μέτρηση κατανάλωσης ρεύματος , εντός ηλεκτρονικών κυκλωμάτων , είναι μία κοινή απαίτηση πόλλων διαφορετικών τύπων εφαρμογών. Τέτοιες , μπορεί να περιλαμβάνουν οτιδήποτε , από φορητό εξοπλισμό χειρός ως και εφαρμογές στην αυτοκινητοβιομηχανία. Παρακάτω , θα διερευνήσουμε τους παράγοντες , που έχουν σημασία για μετρήσεις ρεύματος AC και DC , καθώς και τις επιπτώσεις σε κόστος και απόδοση , των διαφορετικών προσεγγίσεων , αλλά και πώς μπορούν να χρησιμοποιηθούν οι Zetex συσκευές για καλύτερη παρακολούθηση ρεύματος , στις διάφορες εφαρμογές.

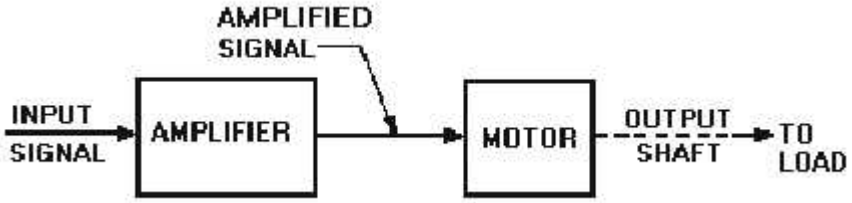
Τύποι Εφαρμογών

Υπάρχουν δύο είδη εφαρμογών : κλειστού βρόγχου (closed loop) και ανοιχτού βρόγχου (open loop).

Στα συστήματα κλεισού βρόχου, το ρεύμα μετράται σε σχέση με μια τιμή αναφοράς και στη συνέχεια να τροποποιείται κατάλληλα , από κάποιο στοιχείο ελέγχου. Ο χρόνος απόκρισης (response time) είναι κρίσιμος εδώ , ιδίως , όταν πρέπει να ληφθούν άμεσες ενέργειες , με βάση την στιγμιαία αξία του ρεύματος. Αυτό είναι παρόμοιο , με το σύστημα ελέγχου της θέρμανσης σε πολλά σπίτια. Ο θερμοστάτης (input) «καλεί» για θερμότητα. Η κάμινος (output) παράγει και διανέμει θερμότητα. Μέρος αυτής της θερμότητας επιστρέφει στον θερμοστάτη (feedback). Όταν το feedback αυξηθεί στην θερμοκρασία , που έχει ο θερμοστάτης για σημείο αναφοράς , τότε ο θερμοστάτης απαντά με το κλείσιμο του συστήματος , εώς ότου να ξαναχρειαστεί θερμότητα. Σε ένα τέτοιο σύστημα , το μονοπάτι του feedback , input>output>input , αποτελεί αυτό που ονομάζουμε κλειστό βρόγχο.

Αντιθέτως , τα συστήματα μέτρησης ρεύματος ανοιχτού βρόγχου , χαρακτηρίζονται από το γεγονός ότι η τιμή που μετράμε , δεν χρησιμοποιείται άμεσα. Μπορεί , για παράδειγμα , να διατεθεί σε ένα άλλο σύστημα , που συνήθως είναι λιγότερο εξερτώμενο από τον χρόνο. Γενικώς , ένα σύστημα ελέγχου ανοιχτού βρόγχου , ελέγχεται άμεσα , και μόνο , από ένα σήμα input , χωρίς το πλεονέκτημα του feedback. Οι βασικές μονάδες αυτού του συστήματος , είναι ένας ενισχυτής και ένα μοτέρ. Ο ενισχυτής λαμβάνει ένα χαμηλό σήμα input και το ενισχύει όσο είναι απαραίτητο για το μοτέρ , να εκπερατώσει την επιθυμητή λειτουργία. Το βασικό σχεδιάγραμμα ενός συστήματος ελέγχου ανοιχτού βρόγχου , φαίνεται στο σχήμα 2. Σε αυτό το σύστημα , το input είναι ένα σήμα που εισάγεται στον

ενισχυτή. Η έξοδος του ενισχυτή είναι ένα σήμα ανάλογο τού πλάτους τού σήματος εισόδου. Η φάση (ac system) και η πολικότητα (dc system) , του σήματος εισόδου , καθορίζουν την κατεύθυνση προς την οποία θα κινηθεί το μοτέρ.Μετά την ενίσχυσή του , το σήμα εισόδου τροφοδοτεί το μοτέρ , το οποίο με τη σειρά του κινεί το output shaft (load) , προς την κατεύθυνση που δέχεται από το σήμα εισόδου. Το μοτέρ δεν θα σταματήσει να κινεί το output shaft , μέχρι το σήμα εισόδου να αφαιρεθεί ή να μειωθεί στο μηδέν. Τέτοια συστήματα , όπως είναι και το δικό μας , απαιτούν και έναν χειριστή , ο οποίος ελέγχει ταχύτητα και κατεύθυνση της εξόδου , με το να αλλάζει την είσοδο.



Σχήμα 2. βασικό σχεδιάγραμμα ενός open loop control system

Μέθοδοι Μέτρησης

Στην ουσία , υπάρχουν τρεις μέθοδοι μέτρησης ρεύματος , οι οποίοι ονομαστικά είναι : Resistive , Optical and Magnetic . Στον ακόλουθο πίνακα 1 , βλέπουμε μία σύντομη επισκόπηση , των θετικών και αρνητικών τους στοιχείων.

Table 1 Common current measuring methods

Type	Current range	Isolated	Accuracy	AC response	Non-Intrusive	Cost
Resistive	Very Low/ High	No	High	Medium/ High	No	Low
Optically isolated resistive	Medium/ High	Yes	Low/ Medium	Low/ Medium	No	Low/ Medium
Magnetic	Medium/ Very High	Yes	Medium	Medium/ High	Yes	High

Η απόφαση , για το ποιά μέθοδος είναι η καταλληλότερη , για την κάθε εφαρμογή , πρέπει να ληφθεί , αφού πρώτα «ζυγιστούν» τα υπέρ και τα κατά της καθεμίας , σε σχέση με τους ακόλουθους παράγοντες:

- Μέγεθος ρεύματος
- Γαλβανική απομόνωση
- Ακρίβεια
- Χρόνος απόκρισης
- Κόστος

Ανεξάρτητα από την επιλογή που γίνεται , όλες οι μέθοδοι , απαιτούν ορισμένη επεξεργασία σήματος πριν ή μετά.

Εμείς επιλέξαμε την Resistive μέθοδο , η οποία έχει την υψηλότερη ακρίβεια και δέχεται πολυ χαληλό ρεύμα.

Resistive Μέθοδος

Πρόκειται για την απλούστερη , φθηνότερη και η πιο βασική μέθοδο μέτρησης ρεύματος. Επίσης , είναι εξ ορισμού η πιο ακριβής και γραμμική μέθοδος όλων. Η τοποθέτηση αντίστασης στην διαδρομή , που ακολουθεί το ρεύμα , έχει το πλεονέκτημα του να μετατρέψει αυτό το ρεύμα σε τάση σε με γραμμικό τρόπο που ακολουθεί εγγενώς , τον νόμο του Ohm : V = I x R.

Έχει , φυσικά , και αυτή τα ελλατώματά της , αν και αυτά μπορούν να ελαχιστοποιηθούν για πολλές εφαρμογές. Το πρώτο και πιο προφανές , από αυτά τα μειονεκτήματα , είναι ότι εισάγει επιπλέον αντίσταση σε ένα ηλεκτρικό κύκλωμα. Αυτό μπορεί να οδηγήσει σε απαράδεκτα υψηλή απώλεια ηλεκτρικής ενέργειας , που εκδηλώνεται ως θερμότητα και ως μειωμένη απόδοση.

Η κατανάλωση ισχύος ισούται με το ρεύμα στο τετράγωνο , επί την αντίσταση , δηλαδή $P = I^2R$ (Joule's law). Το γεγονός αυτό , αυξάνει την απώλεια ισχύος , ως μια εκθετική συνάρτηση του ρεύματος ,και αποτελεί το λόγο , γιατί η Resistive μέθοδος χρησιμοποιείται σχεδόν αποκλειστικά για εφαρμογές με χαμηλή / μέση ένταση ρεύματος. Στην Figure 2 , από κάτω , φαίνεται πόσο πολύ γρήγορα αυξάνεται η κατανάλωση ισχύος , σε ένα κύκλωμα , όπου χρησιμοποιείται η Resistive μέθοδο ,για την παρακολούθηση ρεύματος.

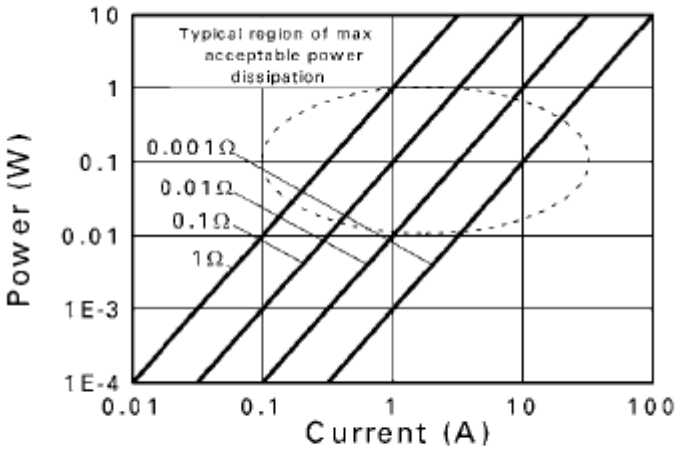


Figure 2 Power dissipated vs current measured

Ένα άλλο μειονέκτημα είναι ότι η μέθοδος , αυτή , αυξάνει και την αντίσταση εξόδου. Το γεγονός , αυτό , έχει διάφορες επιπτώσεις , οι οποίες μπορεί να κυμαίνονται από ελαφρώς ενοχλητικές (π.χ. ελαφρά μειωμένη τάση τερματικού) , ως και καταστροφικές (design με υπερβολικά υψηλό θόρυβο), ειδικά στις περιπτώσεις , όπου η αντίσταση παρεμβάλεται μεταξύ του κυκλώματος και της γείωσής του.

Καθένα από αυτά τα προβλήματα θα μπορούσε να αμβλυνθεί , με τη χρήση όσο το δυνατόν μικρότερης αντίστασης. Δηλαδή , όσο πιο κοντά στο μηδέν , τόσο το καλύτερο. Επιπλέον , η χαμηλότερη τιμή της αντίστασης θα παράγει και την χαμηλότερη κατανάλωση ισχύος. Ωστόσο , μία εξαιρετικά χαμηλή αντίσταση , θα αναπτύξει και χαμηλή διαφορά δυναμικού , στα άκρα της , και έτσι γίνεται συγκρίσιμη με κύκλωμα offset τάσης (V_{os}) , τα οποία έχουν αναξιόπιστη ακρίβεια. Κατά συνέπεια , πρέπει να βρεθεί μια ισορροπία μεταξύ της απαιτούμενης ακρίβειας και της κατανάλωσης ισχύος , για μία συγκεκριμένη ένταση ρεύματος.

Τεχνικές Μέτρησης

Έχουμε δύο τεχνικές μέτρησης ρεύματος : την High-side resistive μέτρηση και την Low-side resistive μέτρηση.

Η υψηλή πλευρά (High-side) αναφέρεται στη διαδρομή εφοδιασμού *προς* το φορτίο , ενώ η χαμηλή πλευρά (Low-side) αναφέρεται στη διαδρομή επιστροφής *από* το φορτίο. Η απόφαση να τοποθετήθεί ένα shunt ρεύματος (Shunt είναι μια συσκευή που επιτρέπει στο ηλεκτρικό ρεύμα να περάσει γύρω σε ένα άλλο σημείο του κυκλώματος) σε οποιαδήποτε θέση , είτε υψηλά είτε χαμηλά , έχει πλεονεκτήματα και μειονεκτήματα , που πρέπει να λαμβάνονται υπόψη και να αξιολογούνται με

βάση τη συγκεκριμένη ,κάθε φορά εφαρμογή. Οι δύο αυτές περιπτώσεις φαίνονται στις Figures 3 και 4 , αντίστοιχα.

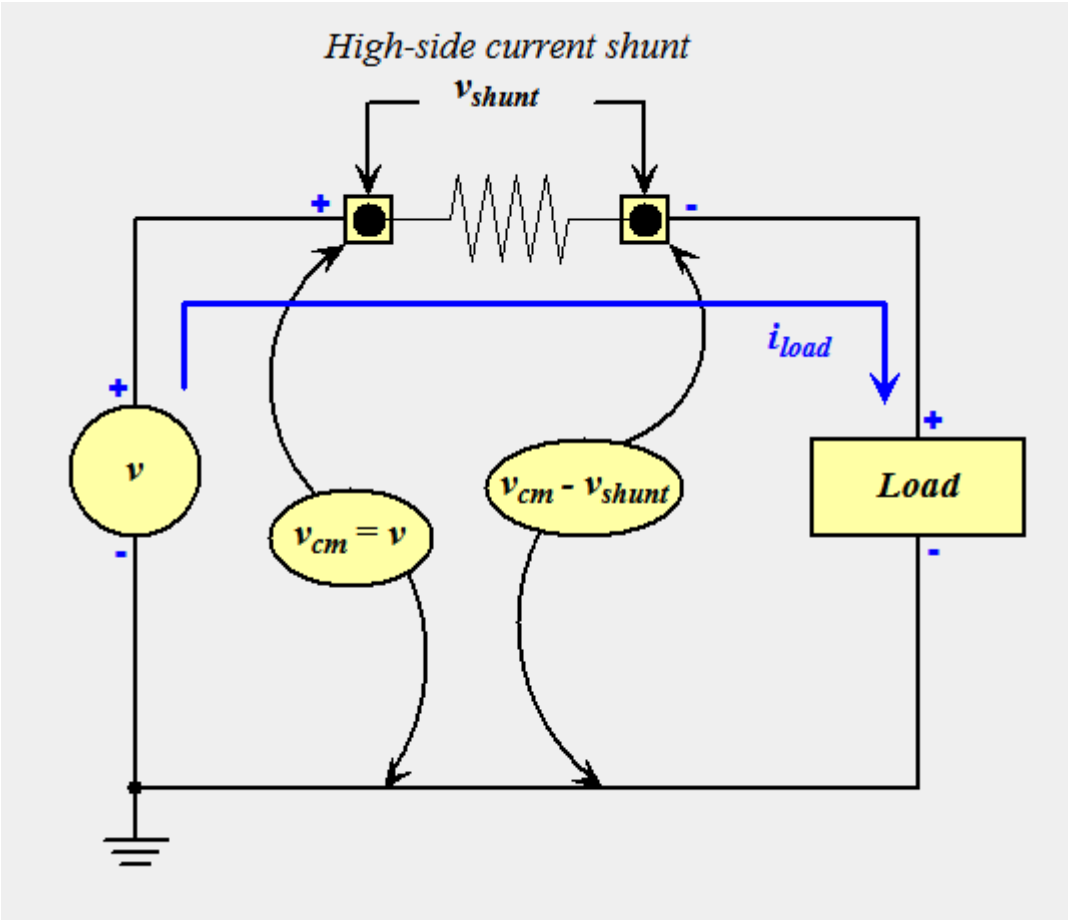


Figure 3. High-side current shunt

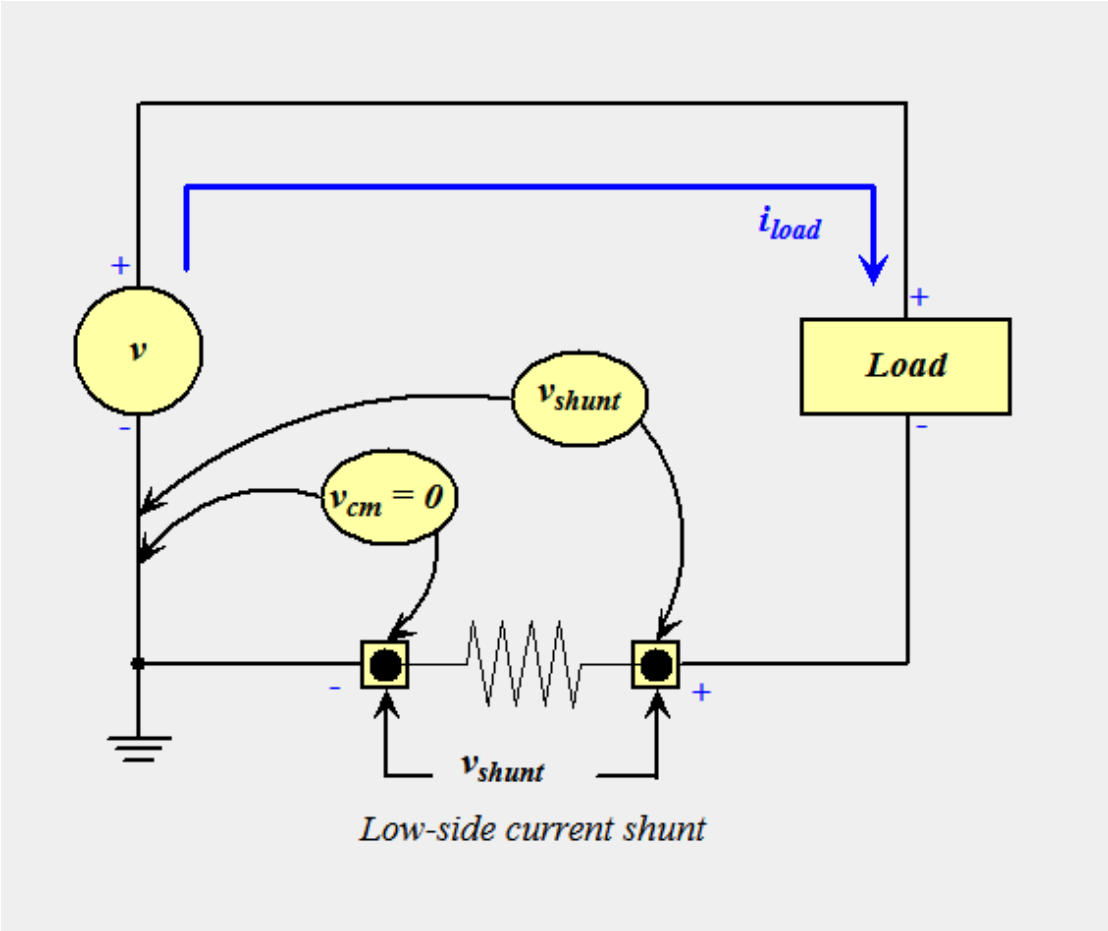


Figure 4. Low -side current shunt

Η κύρια διαφορά , μεταξύ χαμηλής και υψηλής πλευράς , τοποθέτησης , της shunt ρεύματος , είναι ότι η πρώτη μπορεί να εξαλείψει την στιγμιαία διαφορά δυναμικού (common mode voltage) , η οποία εμφανίζεται ταυτόχρονα και σε φάση και στις δύο πλευρές της shunt ρεύματος. Ωστόσο, η προσέγγιση χαμηλής πλευράς , δεν είναι χωρίς μειονεκτήματα , τα οποία περιλαμβάνουν τα ακόλουθα:

- Το φορτίο έχει αφαιρεθεί από την άμεση επαφή του με τη γείωση , η οποία μπορεί να δημιουργήσει προβλήματα για κυκλώματα ελέγχου, να οδηγήσει σε ανεπιθύμητες εκπομπές, ή και τα δύο.
- Μόνο το ρεύμα , που τρέχει στο μονοπάτι επιστροφής *από* το φορτίο μετριέται. Διαρροές ρεύματος στο έδαφος , μέσω τού πλαισίου τού φορτίου του , του κυκλώματος ελέγχου , των καλωδιώσεων , κλπ. δεν μετρώνται , γεγονός που μπορεί να οδηγήσει σε λανθασμένα διαγνωστικά αποτελέσματα.
- Ο μόνος θετικός λόγος , για να χρησιμοποιηθεί ένα low-side shunt , η υπόθεση ότι οι στιγμιαίες διαφορές δυναμικού θα είναι μηδενικές , δεν επαρκεί για εφαρμογές , όπου υψηλά ρεύματα αιχμής δημιουργούν bounce εδάφους. Αυτό έχει ως αποτέλεσμα , ουσιαστικά , μια στιγμιαία διαφορά δυναμικού (common mode

voltage) μεταξύ της γειωμένης πλευράς της shunt ρεύματος και της γείωσης του οργάνου μέτρησης. Αυτό το γεγονός μπορεί να διαταράξει τη μέτρηση και να προκαλέσει πιθανές ζημίες στο όργανο μέτρησης , εφόσον προκληθεί επαρκής διαφορά δυναμικού.

Η high-side τοποθέτηση της shunt , που είναι και η τοποθέτηση που χρησιμοποιήσαμε και εμείς , επιλύει όλα σχεδόν τα παραπάνω προβλήματα , αλλά ουσιαστικά εγκυάται την παρουσία διαφοράς δυναμικού , με αποτέλεσμα , να περιπλέξει το μέσο που χρησιμοποιείται για να κάνει τη μέτρηση. Αν δεν ληφθεί αυτό υπ'όψην και αν δεν γίνουν οι κατάλληλες προσαρμογές των οργάνων , ιδιαίτερα σε εφαρμογές υψηλής τάσης , μπορεί να έχουμε τρομερές συνέπειες , που περιλαμβάνουν εκρηκτική καταστροφή του μέσου , και πιθανό τραυματισμό του γύρω προσωπικού.

Πολλές εφαρμογές , κυρίως εφαρμογές με υψηλό θόρυβο , για λόγους ηλεκτρομαγνητικής συμβατότητας (EMC) , πρέπει να συνδέονται σταθερά με την γείωση , με όσο χαμηλότερη αντίσταση είναι δυνατόν. Η πρόκληση , τώρα , είναι να μεταφραστεί αυτό το μικρό σήμα , με υψηλή διαφορά τάσης στην γραμμή τροφοδοσίας (high common mode element on the supply rail) , σε ένα γειωμένο σήμα (ground referenced signal) , όπως απαιτείται στις περισσότερες περιπτώσεις κυκλωμάτων.

Για την απαίτηση αυτή , η Zetex έχει δημιουργήσει ένα ευρύ φάσμα συμπαγών ολοκληρωμένων κυκωμάτων παρακολούθησης ρεύματος , τη σειρά ZXCT. Η εφαρμογή της στο κύκλωμα είναι όπως δείχνει η figure 5.

Αυτό σημαίνει ότι το σύνθετο έργο της επεξεργασίας των μικρών σημάτων τάσης , που σχετίζονται με το ρεύμα (current-related) , έχει αντιμετωπιστεί και ο σχεδιαστής μπορεί να επικεντρωθεί αντ'αυτού στην χρήση του επεξεργασμένου σήματος.

Με σειρά Zetex η παρακολούθηση ρεύματος , high-side ή low-side , αποτελεί πλέον ένα πολύ πιο απλό έργο. Οι συσκευές είναι πάρα πολύ μικρές και έρχονται είτε σε SOT23 ή SOT23-5 package , με πολύ λίγες εξαιρέσεις σε μεγαλύτερες συσκευασίες.

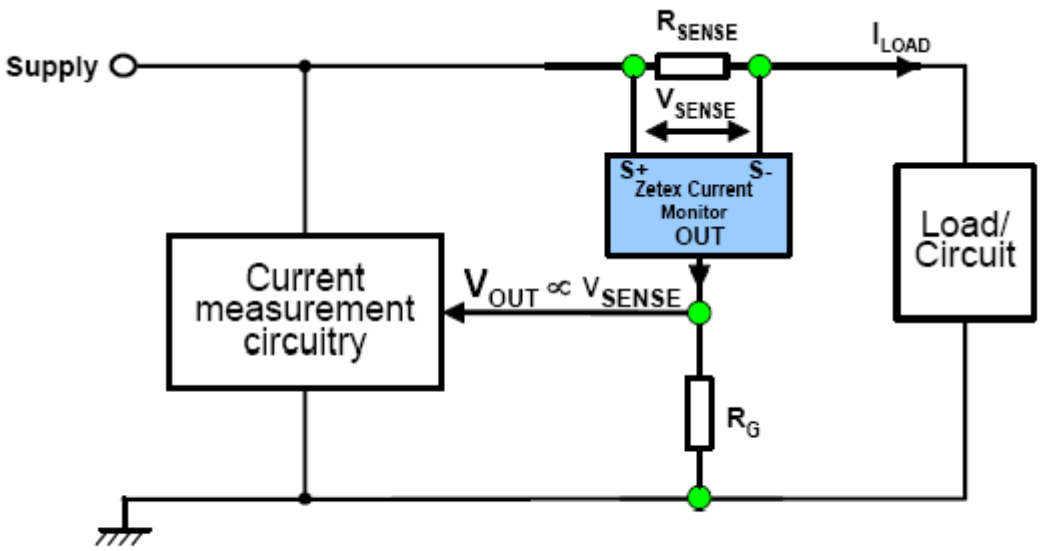


Figure 5 High side current measurement

Ιδιότητες Κυκλωμάτων Μετάφρασης

Ακρίβεια

Η ακρίβεια εξαρτάται από το μέγεθος της sense τάσης (Vsense) για ένα συγκεκριμένο σχέδιο κυκλώματος. Αυτό που θα πρέπει να ληφθεί υπ'όψην , εδώ , είναι η περιοχή εύρους του δυναμικού. Αν , για παράδειγμα , ένα όργανο παρακολούθησης ρεύματος πρέπει να παρακολουθεί ρεύμα , οι τιμές του οποίου διακυμαινονται από , ας πούμε , 100mA ως 1A , έχει ένα δυναμικό εύρος 10:1. Αυτό σημαίνει ότι το offset σφάλμα , κατά τη μέτρηση στα 100mA , θα είναι τουλάχιστον δέκα φορές χειρότερο από ό, τι κατά τη μέτρηση στο 1A. Η σχέση αυτή γίνεται προφανώς χειρότερη , όσο αυξάνεται το εύρος της περιοχής δυναμικού.

Απόκριση Συχνότητας

Αυτό για κάποια κυκλώματα δεν είναι καθόλου σημαντικό. Για παράδειγμα, η αυτόματη παύση της λειτουργίας ενός μοτέρ DC , πιθανότατα δεν θα απαιτεί χρόνο απόκρισης μικρότερο από μερικά χιλιοστά του δευτερολέπτου. Αντίθετα , ρεύμα κύκλου προς κύκλο εναλλασσόμενης τάσης (cycle by cycle current limiting in a switching power supply) , μπορεί να χρειάζεται αποκρίσηση της τάξης του μικροδευτερολέπτου ή μέχρι και τα δέκατα του νανοδευτερολέπτου.

Κατανάλωση Ισχύος

Το ρεύμα , κυρίως για τις συσκευές που λειτουργούν με μπαταρία , πρέπει να μεταφέρει φορτίο χωρητικότητας , τέτοιο ώστε η ταχύτητα του ρεύματος , με την οποία μεταφέρεται , να έρχεται σε ισορροπία με το ρεύμα τροφοδοσίας. Για παράδειγμα , για να μεταφέρει σήμα του 1 volt σε φορτίο χωρητικότητας 5pF ,στα 10kHz , χρειάζονται μόνο εκατοντάδες νανοαμπερ (nanoamps) ρεύματος , με

αποτέλεσμα να κυριαρχήσει η διπολικότητα του ρεύματος. Αν θέλουμε να μεταφέρουμε ένα σήμα με άκρες των10ns σε ένα βολτ , στο ίδιο φορτίο , θα χρειαστούμε εκατοντάδες microamps , αλλάζοντας την διπολικότητα του κυκλώματος , με τρόπο που το καθιστά λιγότερο κατάλληλο για εφαρμογές με μπαταρία. Ένα κύκλωμα δεν θα προσφέρει και τον καλύτερο δυνατό τρόπο λειτουργίας ρευμάτων με microamp και εξαιρετική απόδοση AC , άρα μία επιλογή πρέπει να γίνει.

Μέγιστη Επιτρεπτή Sense Τάση

Ακόμη και αν το μέγεθος της sense τάσης (Vsense) δεν είναι κρίσιμο , από πλευράς κατανάλωσης ισχύος (λόγω των μικρών ρευμάτων προς μέτρηση) , ενδέχεται να υπάρξει μέγιστη πτώση τάσης στο υπό μέτρηση κύκλωμα. Για παράδειγμα, σε ένα κύκλωμα 3.3V, δεν θα ήταν επιθυμητό η μέτρηση ρεύματος να πέσει κατά 500mV , γεγονός που θα αποτελούσε απώλεια της τάξης του 15% και στην τάση και στην ισχύ.

Ιδιότητες Sense Αντίστασης

Η κατανάλωση ισχύος στις αντιστάσεις προκαλεί θερμότητα. Το ερώτημα είναι , πόση θερμότητα μπορεί να ανεχτεί , και ποιές άλλες επιπτώσεις παρουσιάζονται. Για ρεύματα υπό του 1A , σε γενικές γραμμές , είναι απίθανο να δημιουργηθεί κάποιο πρόβλημα , καθώς μια τυπική sense τάση θα απέδιδε απώλειες ισχύος μικρότερες των 100mW , κάτι το οποίο χειρίζεται εύκολα ,από τις περισσότερες mount και through-hole αντιστάσεις. Όπως δείχνει το figure 2 , η κατανάλωση ισχύος είναι άμεσα συνδεδεμένη με την απαιτούμενη ακρίβεια. Θα ήταν πιο κρίσιμο για ένα μεγάλο ρεύμα να έχει χαμηλή sense τάση , με σκοπό την ελαχιστοποίηση της κατανάλωσης ισχύος και έτσι , για μια δεδομένη ακρίβεια , οι αντιστάσεις του κυκλώματος μετάφρασης θα πρέπει να είναι μικρότερες.

Στις περισσότερες περιπτώσεις προτιμώνται mount αντιστάσεις και , για τα μεγαλύτερα ρεύματα και τα σχετικά επίπεδα ισχύος τους , μικρές παρατάξεις αντιστάσεων , σειράς ή παράλληλες , μπορούν να χρησιμοποιηθούν για να μοιραστούν τις απώλειες. Οι αντιστάσεις wire-wound παρέχουν υψηλότερες θερμοκρασίες λειτουργίας , αλλά συνήθως έχουν υψηλότερους συντελεστές θερμοκρασίας και υψηλότερη αυτεπαγωγή , που θα μπορούσε να προκαλέσει προβλήματα για τα σήματα υψηλής συχνότητας.

Γενικώς , υπάχουν δύο τύποι συσκευών μέτρησης ρεύματος : οι Voltage Output Current Monitors και οι Current Output Current Monitors. Οι Current Output συσκευές , μετατρέπουν την sense voltage σε output ρεύμα , ενώ οι Voltage Output συσκευές σε ενισχυμένη τάση. Στον ακόλουθο πίνακα 2 παραθέτονται όλες οι Zetex συσκευές , μέτρησης ρεύματος , με τις ιδιότητες και τα packages , που τους αντιστοιχούν.

Table 2 Current monitor summary/selection chart

Monitor	Output type			Sensing rail[1]		External Resistors	Special feature	Target application	Package
	Current	Voltage gain[2]							
		x10	x100	LS	HS				
ZXCT1008	•				•	1	Simplest COCM[3] + Extended temperature range	General	SOT23
ZXCT1009	•				•	1	Simplest COCM	General	SOT23 or SM8
ZXCT1010	•				•	1	Ground pin[4]	Battery discharge/ General	SOT23-5
ZXCT1011	•				•	2	Ground pin[4] + External transconductance resistor + Extended temperature range	General	SOT23-5
ZXCT1012	•				•	1	Ground pin[4]	Battery discharge/ General	TSOT23-5 TDFN3-5
ZXCT1020	•				•	2	Ground pin[4] + External transconductance resistor + Extended temperature range	General	SOT23-5
ZXCT1021		•			•	0	Simplest VOCM[5]	General	SOT23-5
ZXCT1022			•		•	0	Simplest VOCM	General	SOT23-5
ZXCT1030		•			•	0	High speed + Bandgap reference + OC Comparator	General	SO8
ZXCT1032					•	2	Series transistor drive + Programmable threshold + Output flag	In-rush or over-current protection	SO8
ZXCT1041		•			•	0	Bidirectional + Output flag	General	SOT23-5
ZXCT1050	•			•	•	2	Separate V_{CC} pin + current gain (x20) + Short circuit operation	General	SOT23-5
ZXCT1051		•		•	•	0	Separate V_{CC} pin + Extended temperature + Short circuit operation	General	SOT23-5
ZXCT1080		•				0	Separate V_{CC} pin + 60V CM + Extended temperature	Automotive	SOT23-5
ZXCT1081		•			•	0	Separate V_{CC} pin + 40V CM + Extended temperature		SOT23-5

NOTES;

[1] LS = Low Side Sensing, HS = High Side Sensing.

[2] Output/V_{SENSE}.

[3] Current Output Current Monitor.

[4] Eliminates quiescent current error.

[5] Voltage Output Current Monitor

Σε μία συσκευή VOCM (figure 6) , η αντίσταση ενίσχυσης Rs , έχει ενσωματωθεί. Ωστόσο, εκτός από την RS αντίσταση , η VOCM δεν απαιτεί κανένα άλλο εξωτερικό στοιχείο (component) , σε αντίθεση με μία συσκευή COCM , που πάντα θα απαιτεί , τουλάχιστον , μία. Ως εκ τούτου, όπου ο χώρος και αριθμός στοιχείων είναι κρίσιμα , η VOCM θα έχει ένα σαφές πλεονέκτημα.

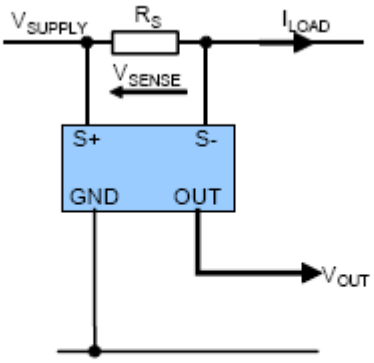


Figure 6 Simplest voltage-output current monitor (VOCM)

Εμείς χρησιμοποιούμε την συσκευή ZXCT1021 με το package SOT23-5 , που της αντιστοιχεί. Πρόκειται για την απλούστερη συσκευή τύπου VOCM (Voltage Output Current Monitor) , η οποία μετατρέπει τη μικρή sense voltage σε ένα ενισχυμένο γειωμένο σήμα , με σταθερή ενίσχυση αύξησης 10. Έχουμε , δηλαδή ,

$$V_{out} = 10 * V_{sense}$$

Από την οποία V*out* , υπολογίζουμε και την μέτρηση κατανάλωσης ισχύος , που περιγράφεται αναλυτικά σε επόμενο κεφάλαιο.

5. Μεθοδολογία

Πάμε να δούμε πιο αναλυτικά τα βήματα , που ακολουθήσαμε και τον τρόπο με τον οποίο εργαστήκαμε , για να μετρήσουμε την κατανάλωση ισχύος , των οχτώ διαφορετικών εφαρμογών ασφαλείας.

Οι οχτώ αλγόριθμοι ασφαλείας , που επιλέξαμε , είναι ονομαστικά οι ακόλουθοι : Basic RSA , Basic DES , Camellia_1 , Camellia_2 , 3DES , AES , DES και MD5. Είναι σύνθετοι , ισχυροί και ευρέως διαδεδομένοι στην σημερινή αγορά. Οι αλγόριθμοι Basic RSA , Basic DES , Camellia_1 , Camellia_2 και AES είναι γραμμένοι σε γλώσσα VHDL. Οι υπόλοιποι αλγόριθμοι 3DES , DES και MD5 είναι γραμμένοι σε γλώσσα VERILOG. Δουλεύουμε στο περιβάλλον της Xilinx ISE 9.1 .

Η διαδικασία , που περιγράφεται παρακάτω για τον ένα αλγόριθμο , εφαρμόστηκε ακριβώς η ίδια και για όλους τους υπόλοιπους. Ας δούμε , λοιπόν , πως εργαστήκαμε με ένα τον αλγόριθμο , π.χ. τον Basic DES.

Αρχική Διαδικασία

Κάθε αλγόριθμος είχε διαφορετική αρχιτεκτονική διαύλου (bus) και διαφορετική λογική εισόδου και εξόδου. Έτσι , ξεκινάμε τροποποιώντας την λογική τους , ενσωματώνοντας μια top level FSM , για να ελέγχουμε τα σήματα κάθε εφαρμογής. Η οντότητα του top level μας , έχει δύο σήματα εισόδου και τέσσερα σήματα εξόδου. Ως εισόδους δέχεται τα σήματα CLOCK και RESET , ενώ οι έξοδοι είναι τέσσερα σήματα ελέγχου , που συνδέονται με τέσσερις λυχνίες (LEDS).

Η FSM μας εκτελεί έναν ατέρμωνα βρόγχο , γράφοντας και διαβάζοντας δεδομένα , από και προς την εσωτερικκή λογική του πηρύνα. Αυτός ο ατέρμων βρόγχος , μας παρέχει με μια σταθερή καταναλωση ρεύματος , καθ’όλη την διάρκεια του πειράματος. Επιπλέον , το γεγονός ότι ο πυρήνας καταναλώνει την ίδια ποσότητα ισχύος , καθ’όλη την διάρκεια του πειράματος , καθιστά τις μετρήσεις που πήραμε πιο αξιόπιστες και ιδιαίτερα ακριβείς. Αλλά , ας μην προτρέχουμε. Έχουμε δημιουργήσει την top level FSM μας , για τον vhdl αλγόριθμο Basic DES , τον οποίο και μετονομάζουμε σε E_basicDES. Παρόμοιες top level FSM , με αυτήν του E_basicDES , κάνουμε και για τους υπόλοιπους αλγόριθμους.(Η κάθε top level FSM , που δημιουργήσαμε , για τον κάθε αλγόριθμο , ξεχωριστά , παρουσιάζεται στο appendix).

Όπως , προαναφέραμε έχουμε ως εισόδους τα σήματα clock και reset , ενώ ως εξόδους 4 leds. Ενώνουμε τα clock και reset με τα προϋπάρχοντα , αντίστοιχα , σήματα του αλγορίθμου , ενώ στα leds αναθέτουμε τα σήματα των προϋπαρχθέντων εξόδων ή ενα bit τους , αν είναι μεγαλύτερου μεγέθους. Οι έξοδοι μας , στην ουσία , είναι σήματα ελέγχου , ότι λειτουργεί σωστά ο αλγόριθμος. Μετά το –

πάντα- απαραίτητο debugging και την υλοποίηση του αντίστοιχου testbench , έχουμε το σωστό simulation , που δείχνει ότι το πρόγραμμά μας λειτουργεί σωστά.

Τώρα , που ο αλγόριθμος τρέχει σωστά , πρέπει να παράγουμε αρχείο προγραμματισμού .bit , που αργότερα θα προγραμματίσει το FPGA μας. Για να το κάνουμε , αυτό , πρέπει πρώτα να δημιουργήσουμε ένα αρχείο .ucf. Το User Constraints File (UCF) είναι ένα αρχείο ASCII , που προσδιορίζει τους περιορισμούς σχετικά με τη λογική τού σχεδιασμού. Οι εν λόγω περιορισμοί επηρεάζουν τον τρόπο που , η λογική του σχεδιασμού , εφαρμόζεται στην συσκευή προορισμού. Χρειάζεται , δηλαδή , για τη μετάβαση του αλγορίθμου E_basicDES (software) στην πλακέτα XUP Virtex-II PRO (hardware) , που χρησιμοιούμε.

Το αρχείο XUP.ucf είναι ίδιο για όλους τους αλγόριθμους.

Προσθέτουμε το XUP.ucf , μέσα στο design μας και επιβεβαιώνουμε , για άλλη μια φορά , ότι όλα λειτουργούν ομαλά , τρέχοντας ένα behavioural simulation (RLT simulation).

Έπειτα , συνθέτουμε (synthesis) και υλοποιούμε (implement design) το design μας. Η διαδικασία της υλοποίησης , τρέχει αυτόματα τις επιμέρους διαδικασίες:

- translate , η οποία συγχωνεύει τις εισερχόμενες netlists και τους περιορισμούς
- map , η οποία εντάσσει στο σχέδιο στους διαθέσιμους πόρους της συσκευής προορισμού
- place & route , η οποία θέτει και δρομολογεί το σχέδιο , στους χρονικούς περιορισμούς.

Μόνο , εφόσον , όλες οι παραπάνω διαδικασίες εκτελεστούν σωστά , μπορούμε να προχωρήσουμε στην διαδικασία παραγωγής αρχείου προγραμματισμού (generate programming file). Όταν και αυτή τρέξει σωστά , δημιουργεί ένα bitsteam αρχείο , το οποίο μπορεί να φορτωθεί στην συσκευή μας. Ονομάζουμε το αρχείο αυτό, π.χ. , 1.bit .

Τώρα μέσω του εργαλείου της Xilinx Impact πάμε να διαμορφώσουμε το FPGA μας. Υπάρχουν διάφορες τεχνικές διαμόρφωσης και προγραμματισμού. Αυτή , που χρησιμοποιούμε εμείς , η Boundary-Scan (γνωστή και ως JTAG) , είναι η πιο διαδεδομένη. Μέσω αυτής φορτώνουμε το αρχείο προγραμματισμού μας , 1.bit , στην πλακετα XUP Virtex-II PRO , που φαίνεται στην παραπάνω εικόνα 1. Στο κέντρο της πλακέτας διακρίνεται η Virtex-II PRO συσκευή μας XC2VP30. Το download του αρχείου μας , από τον υπολογιστή στο FPGA , γίνεται με καλώδιο usb για FPGA.

Μέτρηση Κατανάλωσης Ισχύος

Η μέτρηση κατανάλωσης ισχύος τού πηρύνα τού FPGA μας , είναι μια δύσκολη διαδικασία , ιδίως αν η πλακέτα , στην οποία εφαρμόζονται οι μετρήσεις , δεν είναι ειδικά σχεδιασμένη για τέτοιου είδους πειράματα. Για να μετρηθεί με ακρίβεια η κατανάλωση ισχύος , έπρεπε να απομονώσουμε πρώτα την τάση της εσωτερικής λογικής της πλακέτας (στα 1,5 volt) , και να την οδηγήσουμε από εξωτερικο ρεύμα DC (Displacement Current) ρυθμιζόμενης παροχής.

Για να μπορέσουμε να μετρήσουμε την ισχύ που καταναλώνεται στο εσωτερικό πηρύνα τού FPGA μας , παρεμβάλουμε μία αντίσταση. Στα άκρα της αντίστασης αυτής , δημιουργείται μία διαφορά δυναμικού , και γι'αυτό την επιλέγουμε όσο μικρότερη γίνεται , στα 0,1 Ohm , ώστε η διαφορά τάσης να είναι αμηλητέα. Για να μπορούμε να δούμε καθαρά τις διαφορές στην τάση , όταν τρέχει ο αλγόριθμος , παρεμβάλουμε και έναν ενισχυτή Zetex ZXCT1021 , ο οποίος ενισχύει την τάση x10 , ώστε να διακρίνονται ξεκάθαρα οι διαφορές τάσης του σήματος στον παλμογράφο.

Πριν μπορέσουμε να πάρουμε οποιαδήποτε μέτρηση , πρέπει να συνδέσουμε σωστά το κύκλωμά μας. Τα όργανα , που χρησιμοποιούμε , πέραν την πλακέτας , είναι μία αντίσταση R γνωστή στα 0,1 Ohm (με συσκευή ενισχυσης ZXCT1021), ένα τροφοδοτικό και ένας παλμογράφος.

Οι συνδέσεις έχουν ως εξής :

- Το συν της τροφοδοσίας , σταθερά ρυθμισμένη στα 1,5V , συνδέεται με το in της αντίστασης.
- Το out της αντίστασης συνδέεται με το συν της πλακέτας , που δέχεται 1,5V.
- Το ground της πλακέτας συνδέεται και με το ground της αντίστασης και με το ground του τροφοδοτικού.
- Τέλος , ο παλμογράφος συνδέεται με το in και το ground της αντίστασης.

Μόνο όταν βεβαιωθούμε ότι όλες οι παραπάνω συνδέσεις έχουν γίνει σωστά , μπορούμε να προχωρήσουμε στην μέτρηση της κατανάλωσης ισχύος , του FPGA μας , επαναπρογραμματισμένο τώρα , με το αρχείο 1.bit .

Το σήμα , που φαίνεται στον παλμογράφο , μετράμε ότι παρουσιάζει διαφορα δυναμικού π.χ. 62mV, ΔV=62mV. Η μέτρηση , αυτή , με απλούς μαθηματικούς υπολογισμούς , μας δίνει την ισχύ , που της αντιστοιχεί:

$\Delta V = I * R$ *Νόμος του Ohm*

$\Delta V/10 = I * 0,1$ *Διαιρώ δια δέκα την ΔV για να πάρω την πραγματική μέτρηση , και όχι την ενισχυμένη. Επίσης έχω γνωστή αντίσταση των 0,1 Ohm*

$\Delta V = I$ *Σχεση α*

P = V * I *Νόμος του Joule*

P = 1,5 * I *Σταθερή τάση τροφοδοσίας στα 1,5V*

P = 1,5 * ΔV *Σχέση β (από παραπάνω σχέση α)*

Στο παράδειγμά μας , δηλαδή , από τις σχέσεις α και β , θα έχουμε:

ΔV = I * R

62 = I * R

62/10 = I * 0,1

I = 62 mA

P = V * I

P = 1,5 * I

P = 1,5 * 62

P = 93 mW

Η ισχύς που καταναλώνεται , λοιπόν , είναι 93 mW . Η μέτρηση , αυτή τώρα , *μέτρηση α* ή αλλιώς *default* , αντιπροσωπεύει μία σειρά παραμέτρων synthesis και implementation της Xilinx. Αυτές είναι οι παράμετροι , που θα διερευνήσουμε , για να βρούμε την βέλτιστη κατανάλωση ισχύος.

Βελτιστοποίηση Κατανάλωσης Ισχύος

Στην διαδικασία σύνθεσης (synthesize) υπάρχουν διάφορες ιδιότητες , από τις οποίες επιλέγουμε τις δύο βασικότερες : optimization goal και optimization effort.

Optimization , ονομάζεται η διαδικασία βελτιστοποίησης , που είτε μειώνει την έκταση (area) ή αυξάνει την ταχύτητα (speed) ενός σχεδίου. Optimization goal , λοιπόν , είναι τι θέτουμε , εμείς , ως στόχο βελτιστοποίησης speed ή area και optimization effort , η προσπάθεια που καταβάλεται γι'αυτήν την βελτιστοποίηση , με τιμές normal ή high. Για τον σκοπό αυτής της διπλωματικής και μετά από κάποια αρχικά πειράματα , επιλέγουμε το optimization effort να είναι πάντα high. Προκύπτουν , επομένως , οι δύο ακόλουθοι συνδυασμοί.

optimization goal(speed/area)	optimization effort(normal/high)
speed	high
area	high

Στην διαδικασία υλοποίησης (implement design) υπάρχουν , επίσης , διάφορες ιδιότητες , από τις οποίες επιλέγουμε τις τρεις σημαντικότερες: map effort level , place & route mode και power reduction.

Mapping λέγεται η διαδικασία ανάθεσης των στοιχείων λογικής σε συγκεκριμένα φυσικά στοιχεία , τα οποία , όντως , εφαρμόζουν λογικές λειτουργίες σε μία συσκευή. Η προσπάθεια για αυτήν τη διαδικασία , δηλαδή το map effort level , μπορεί να είναι medium , standar ή high. Εμείς κρατάμε τα δύο υψηλότερα επίπεδα , το standar και το high.

Η διαδικασία Place & Route, παίρνει το αρχείο που έχει παραχθεί , από την άνω διαδικασία mapping , τοποθετεί και δρομολογεί το design , και παράγει ένα νέο αρχείο , που θα χρησιμοποιηθεί ως είσοδος από την programming file generating process , για την παραγωγή του bitstream αρχείου. Ο τύπος place & route , δηλαδή το place & route mode , διευκρινίζει τον συγκεκριμένο τύπο τοποθέτησης και δρομολόγησης , με τον οποίο θέλουμε να υλοποιήσουμε το design μας. Από τις πέντε πιθανές επιλογές Normal Place and Route , Place Only , Route Only , Reentrant Route και Multi Pass Place and Route , διαλέγουμε την πρώτη , που αποτελεί και την default επιλογή της Xilinx και την τελευταία , που τρέχει PAR (Place and Route) μέσω διαφορετικών προσεγγίσεων , για να μας παρέχει με την καλύτερη δυνατή τοποθέτηση.

Τέλος , η διαδικασία Power Reduction , που καθορίζει αν θα γίνει βελτιστοποίηση της δρομολόγησης , με σκοπό την ελάττωση της κατανάλωσης ισχύος. Οι δυνατές επιλογές , εδώ , είναι on ή off.

Όλοι οι δυνατοί συνδυασμοί , των παραπάνω παραμέτρων , δεκαέξι στο σύνολο , δίνονται στον ακόλουθο πίνακα 3.

Τύπος Μέτρησης	optimization goal (speed/area)	optimization effort (normal/high)	map effort level (medium/standar/high)	place & route mode (normal/multi)	power reduction (on/off)
Μέτρηση **a**	speed	high	standar	normal	off
Μέτρηση **b**	area	high	standar	normal	off
Μέτρηση **C**	speed	high	high	normal	off
Μέτρηση **d**	area	high	high	normal	off
Μέτρηση **e**	speed	high	standar	multi	off
Μέτρηση **f**	area	high	standar	multi	off
Μέτρηση **g**	speed	high	high	multi	off
Μέτρηση **h**	area	high	high	multi	off
Μέτρηση **i**	speed	high	standar	normal	on
Μέτρηση **j**	area	high	standar	normal	on
Μέτρηση **k**	speed	high	high	normal	on
Μέτρηση **l**	area	high	high	normal	on
Μέτρηση **m**	speed	high	standar	multi	on
Μέτρηση **n**	area	high	standar	multi	on
Μέτρηση **o**	speed	high	high	multi	on
Μέτρηση **p**	area	high	high	multi	on

Πίνακας 3. Δυνατοί συνδυασμοί των επιλεγμένων παραμέτρων συνθεσης και υλοποίησης

Κάθε σειρά του παραπάνω πίνακα , αντιστοιχεί και σε έναν τύπο μέτρησης. Όταν αναφερόμαστε , λοιπόν , σε μία μέτρηση a, b, c , κ.λ.π. , εννοούμε και το συνδυασμό των παραπάνω παραμέτρων , που αντιπροσωπεύει. Στη μέτρηση g , για παράδειγμα , αντιστοιχούν οι ακόλουθες παράμετροι:

	optimization goal (speed/area)	**optimization effort (normal/high)**	**map effort level (medium/standar/high)**	**place & route mode (normal/multi)**	**power reduction (on/off)**
Μέτρηση **g**	speed	high	high	multi	off

Η μέτρηση α είναι η μέτρηση , που έχουμε θέσει ως default. Πάμε να υπολογίσουμε την μέτρηση b. Επιλέγουμε τις παραμέτους που της αντιστοιχούν και τρέχουμε , εκ νέου , τις διαδικασίες synthesize , implement design και generate programming file. Το αρχείο προγραμματισμού , που προκυπτει , έστω ότι το ονομάζουμε 2.bit. Τώρα , επαναλαμβάνουμε τη διαδικασία που περιγράφεται στην παραπάνω υποενότητα *μέτρηση κατανάλωσης ισχύος* , για να πάρουμε την μέτρηση P = 99mW , η οποία αντιστοιχεί στην μέτρηση b , για τον αλγόριθμο E_basicDES.

Αυτή η διαδικασία επαναλαμβάνεται και για τις 16 μετρήσεις , που αναγράφονται στον παραπάνω πίνακα 3 , δίνοντας μας δεκαέξι διαφορετικές τιμές κατανάλωσης ισχύος , για τον αλγόριθμο E_basicDES.

Οι δεκαέξι τιμές κατανάλωσης ισχύος που αντιστοιχούν στον αλγόριθμο E_basicDES δίνονται ακολούθως στον πίνακα 4:

Τύπος μέτρησης	Power (mW)
Μέτρηση **a**	93
Μέτρηση **b**	99
Μέτρηση **c**	102
Μέτρηση **d**	93
Μέτρηση **e**	102
Μέτρηση **f**	102
Μέτρηση **g**	102
Μέτρηση **h**	99
Μέτρηση **i**	99
Μέτρηση **j**	93
Μέτρηση **k**	99
Μέτρηση **l**	93
Μέτρηση **m**	99
Μέτρηση **n**	99
Μέτρηση **o**	99
Μέτρηση **p**	93

Πίνακας 4. Τιμές κατανάλωσης ισχύος του αλγορίθμου E_basicDES

Όλες οι μετρήσεις επαναλήφθηκαν τουλάχιστον τρεις φορές , για έλεγχο και ελαχιστοποίηση λαθών , με σκοπό να πετύχουμε τη μέγιστη δυνατή ακρίβεια.

Διακρίνουμε , λοιπόν , ότι η ελάχιστη , και άρα βέλτιστη , κατανάλωση ισχύος , του αλγορίθμου E_basicDES , είναι στα 93mW , που αντιστοιχούν στις μετρήσεις a , d , j , l και p. Επιπλέον ανάλυση των αποτελεσμάτων δίνεται στην ενότητα ΣΥΜΠΕΡΑΣΜΑΤΑ.

Έπειτα , επαναλαμβάνουμε όλη τη διαδικασία , από την αρχή της ενότητας ΜΕΘΟΔΟΛΟΓΙΑ , και για τους υπόλοιπους εφτά αλγόριθμους. Στο τέλος , καταλήγουμε με 125 ξεχωριστές τιμές κατανάλωσης ισχύος , συνολικά , τις οποίες χρησιμοποιούμε και αναλύουμε , αργότερα. (κανονικά έπρεπε να έχουμε 128 τιμές , αλλά παρουσιάστηκε πρόβλημα σε τρεις περιπτώσεις και δεν καταφέραμε να παράγουμε τρία αρχεία .bit).

6. Συμπεράσματα

Ξεκινάμε να αναλύσουμε τα αποτελέσματα , των μετρήσεών μας , με τη βοήθεια διαγραμμάτων , για τον κάθε ένα αλγόριθμο ξεχωριστά.

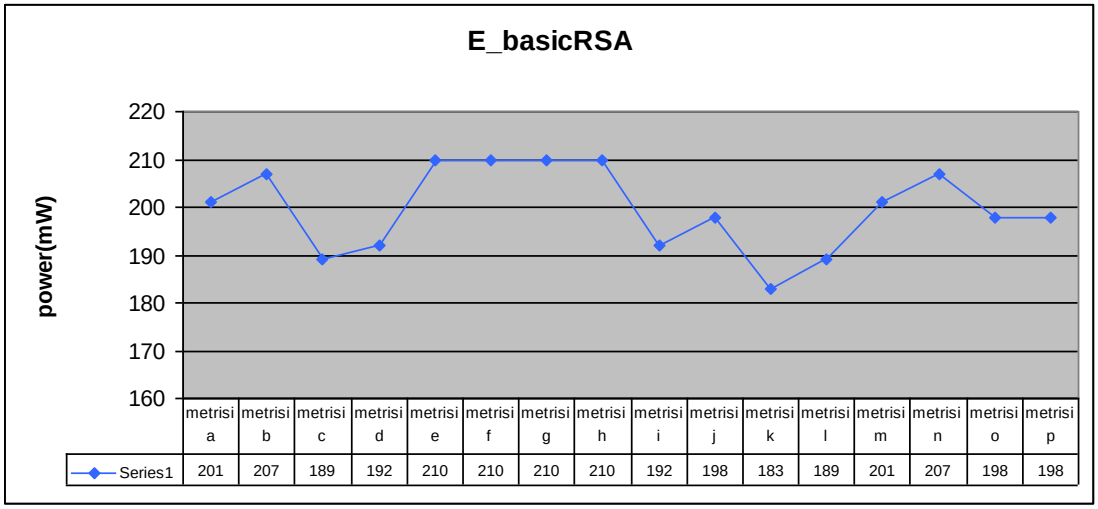
E_basicrsa

Για τον αλγόριθμο E_basicrsa έχουμε μετρήσει τις τιμές κατανάλωσης ισχύος , που φαίνονται στον πίνακα 5.

Τύπος μέτρησης	**design name**	**optimization goal (speed/ area)**	**optimization effort (normal/ high)**	**map effort level (standar/ high)**	**place & route mode (normal/ multi)**	**power reduction (on/off)**	**Power (mW)**
A	E_basicrsa	speed	high	standar	normal	off	201
B	E_basicrsa	area	high	standar	normal	off	207
C	E_basicrsa	speed	high	high	normal	off	189
D	E_basicrsa	area	high	high	normal	off	192
E	E_basicrsa	speed	high	standar	multi	off	210
F	E_basicrsa	area	high	standar	multi	off	210
G	E_basicrsa	speed	high	high	multi	off	210
H	E_basicrsa	area	high	high	multi	off	210
I	E_basicrsa	speed	high	standar	normal	on	192
J	E_basicrsa	area	high	standar	normal	on	198
K	E_basicrsa	speed	high	high	normal	on	183
L	E_basicrsa	area	high	high	normal	on	189
M	E_basicrsa	speed	high	standar	multi	on	201
N	E_basicrsa	area	high	standar	multi	on	207
O	E_basicrsa	speed	high	high	multi	on	198
P	E_basicrsa	area	high	high	multi	on	198

Πίνακας 5. Τιμές κατανάλωσης ισχύος για τον αλγόριθμο E_basicrsa

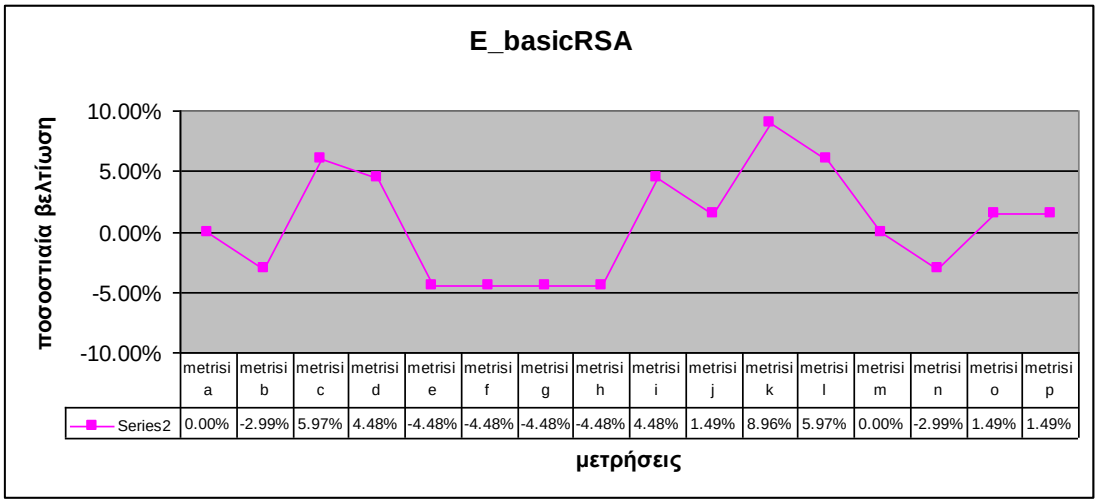
Το διάγραμμα 1 , που ακολουθεί , δείχνει τις τιμές κατανάλωσης ισχύος , για κάθε μέτρηση , δηλαδή για κάθε συνδυασμό παραμέτρων.



Διάγραμμα 1. Κατανάλωση ισχύος σε σχέση με το είδος μέτρησης

Φαίνεται , λοιπόν , καθαρά , ότι μικρότερη κατανάλωση ισχύος έχουμε για την μέτρηση k , με τις μετρήσεις l και c , να ακολουθούν αμέσως μετά. Αυτό σημαίνει πως , κάποιος που θέλει να χρησιμοποιήσει την εφαρμογή BasicRSA και ενδιαφέρεται για την ελάχιστη κατανάλωση ισχύος της , θέτει απλά τις παραμέτρους που αντιστοιχούν στη μέτρηση k , πριν τρέξει τον αλγόριθμο.

Το διάγραμμα 2 , που ακολουθεί , δείχνει την ποσοστιαία βελτίωση της κατανάλωσης ισχύος , του αλγόριθμου , σε σχέση με τον τύπο μέτρησης.



Διάγραμμα 2. Ποσοστιαία βελτίωση

Βλέπουμε , καθαρά λοιπόν , ότι η βέλτιστη μέτρηση k έχει την μεγαλύτερη ποσοστιαία βελτίωση , όπως είναι αναμενόμενο , έναντι των άλλων μετρήσεων , με ποσοστό 8,89%.

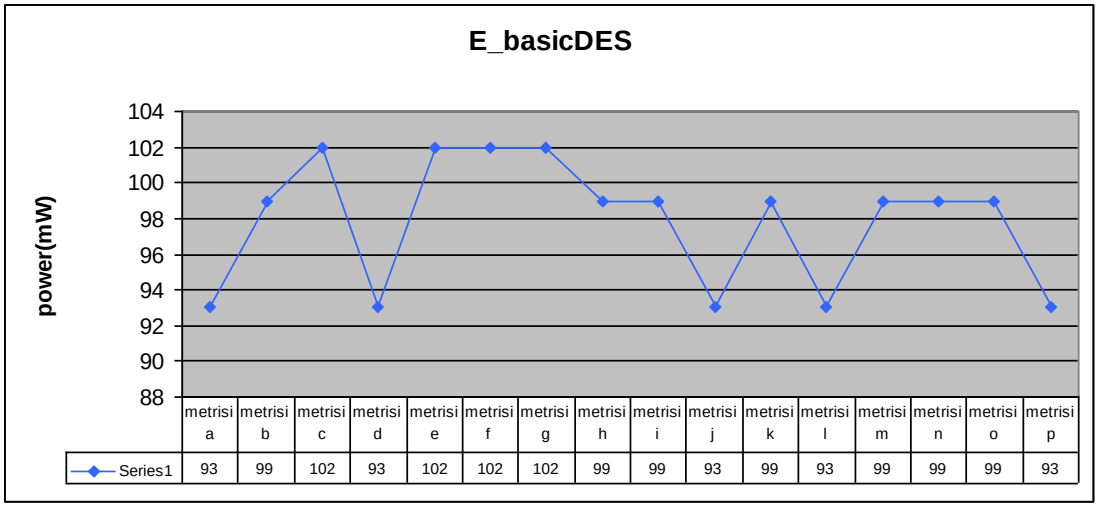
E_basicdes

Για τον αλγόριθμο E_basicdes έχουμε μετρήσει τις τιμές κατανάλωσης ισχύος , που φαίνονται στον πίνακα 6.

Τύπος μέτρησης	**design name**	**optimization goal (speed/ area)**	**optimization effort (normal/ high)**	**map effort level (standar/ high)**	**place & route mode (normal/ multi)**	**power reduction (on/off)**	**Power (mW)**
A	E_basicDES	speed	high	standar	normal	off	93
B	E_basicDES	area	high	standar	normal	off	99
C	E_basicDES	speed	high	high	normal	off	102
D	E_basicDES	area	high	high	normal	off	93
E	E_basicDES	speed	high	standar	multi	off	102
F	E_basicDES	area	high	standar	multi	off	102
G	E_basicDES	speed	high	high	multi	off	102
H	E_basicDES	area	high	high	multi	off	99
I	E_basicDES	speed	high	standar	normal	on	99
J	E_basicDES	area	high	standar	normal	on	93
K	E_basicDES	speed	high	high	normal	on	99
L	E_basicDES	area	high	high	normal	on	93
M	E_basicDES	speed	high	standar	multi	on	99
N	E_basicDES	area	high	standar	multi	on	99
O	E_basicDES	speed	high	high	multi	on	99
P	E_basicDES	area	high	high	multi	on	93

Πίνακας 6. Τιμές κατανάλωσης ισχύος για τον αλγόριθμο E_basicdes

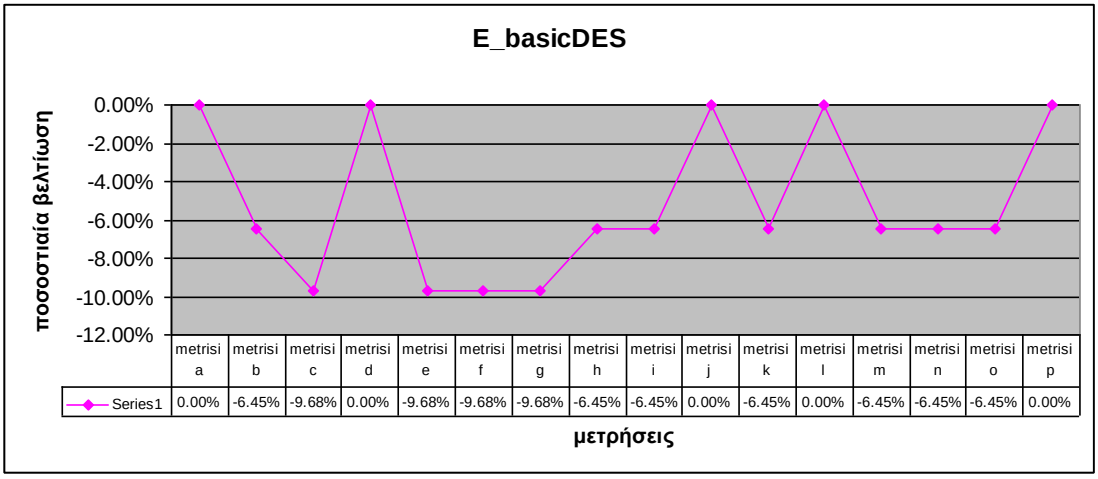
Το διάγραμμα 3 , που ακολουθεί , δείχνει τις τιμές κατανάλωσης ισχύος , για κάθε μέτρηση , δηλαδή για κάθε συνδυασμό παραμέτρων.



Διάγραμμα 3. Κατανάλωση ισχύος σε σχέση με το είδος μέτρησης

Εδώ , παρατηρούμε , ότι μικρότερη κατανάλωση ισχύος έχουμε για πέντε διαφορετικές μετρήσεις. Πρόκειται για την default μέτρηση a και τις μετρήσεις d , j , l και p. Το γεγονός αυτό , δεν είναι κακό , αφού δίνει τη δυνατοτητα σε κάποιον που θέλει να χρησιμοποιήσει την εφαρμογή BasicDES και ενδιαφέρεται για την ελάχιστη κατανάλωση ισχύος της , να διαλέξει ποιές από τις παραμέτρους των παραπάνω μετρήσεων , τον βολεύουν καλύτερα.

Το διάγραμμα 4 , που ακολουθεί , δείχνει την ποσοστιαία βελτίωση της κατανάλωσης ισχύος , του αλγόριθμου , σε σχέση με τον τύπο μέτρησης.



Διάγραμμα 4. Ποσοστιαία βελτίωση

Όπως , παρατηρούμε εδώ , στην ουσία , δεν έχουμε βελτίωση της κατανάλωσης ισχύος. Οι καλύτερες μετρήσεις είναι αυτές που διατηρούν την ποσοστιαία διαφορά τους , στο 0%.

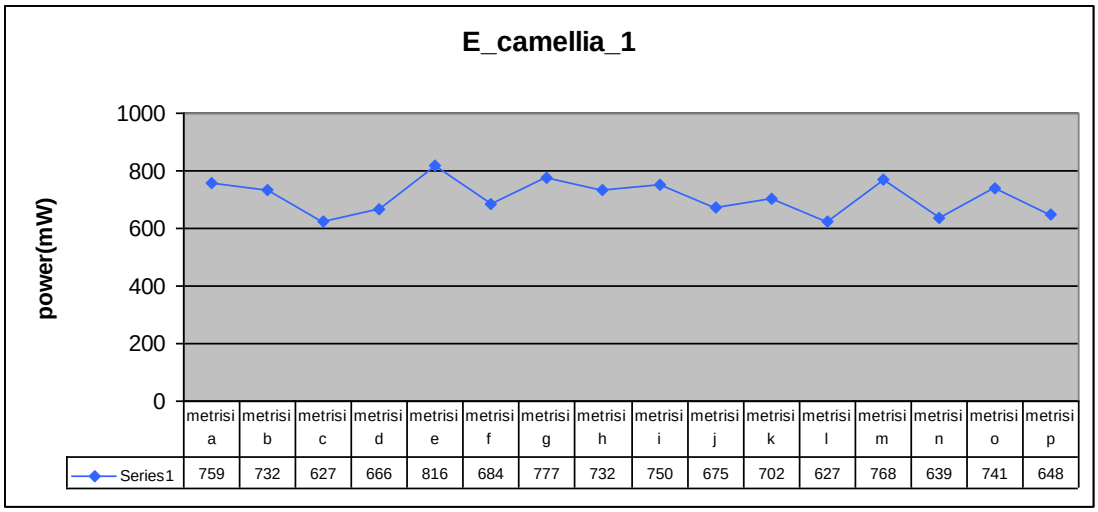
E_camellia_1

Για τον αλγόριθμο E_camellia_1 έχουμε μετρήσει τις τιμές κατανάλωσης ισχύος , που φαίνονται στον πίνακα 7.

Τύπος μέτρησης	**design name**	**optimization goal (speed/ area)**	**optimization effort (normal/ high)**	**map effort level (standar/ high)**	**place & route mode (normal/ multi)**	**power reduction (on/off)**	**Power (mW)**
A	E_camelia_1	speed	high	standar	normal	off	759
B	E_camelia_1	area	high	standar	normal	off	732
C	E_camelia_1	speed	high	high	normal	off	627
D	E_camelia_1	area	high	high	normal	off	666
E	E_camelia_1	speed	high	standar	multi	off	816
F	E_camelia_1	area	high	standar	multi	off	684
G	E_camelia_1	speed	high	high	multi	off	777
H	E_camelia_1	area	high	high	multi	off	732
I	E_camelia_1	speed	high	standar	normal	on	750
J	E_camelia_1	area	high	standar	normal	on	675
K	E_camelia_1	speed	high	high	normal	on	702
L	E_camelia_1	area	high	high	normal	on	627
M	E_camelia_1	speed	high	standar	multi	on	768
N	E_camelia_1	area	high	standar	multi	on	639
O	E_camelia_1	speed	high	high	multi	on	741
P	E_camelia_1	area	high	high	multi	on	648

Πίνακας 7. Τιμές κατανάλωσης ισχύος για τον αλγόριθμο E_camellia_1.

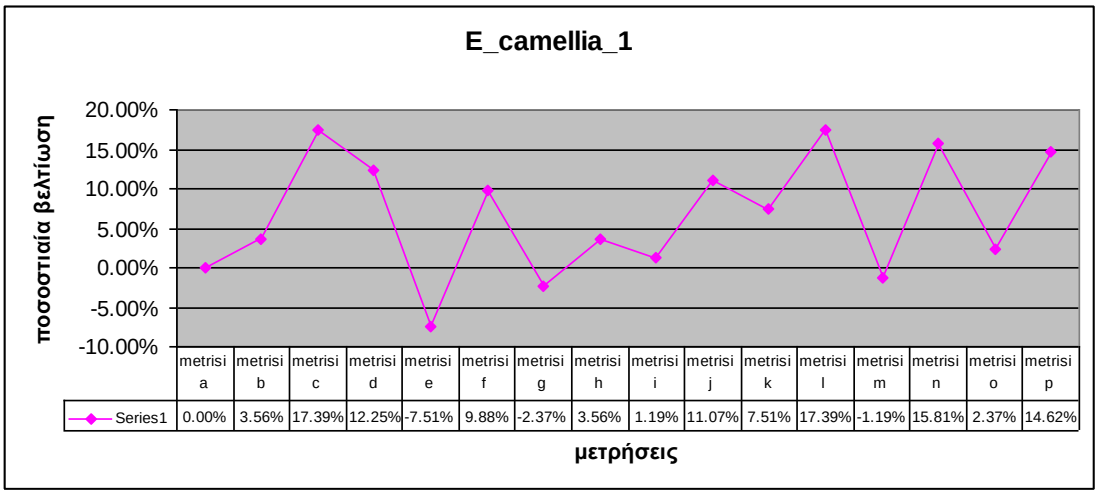
Το διάγραμμα 5 , που ακολουθεί , δείχνει τις τιμές κατανάλωσης ισχύος , για κάθε μέτρηση , δηλαδή για κάθε συνδυασμό παραμέτρων.



Διάγραμμα 5. Κατανάλωση ισχύος σε σχέση με το είδος μέτρησης

Εδώ , παρατηρούμε , ότι μικρότερη κατανάλωση ισχύος έχουμε για δύο διαφορετικές μετρήσεις , τις c και l , με την μέτρηση n , βρίσκεται αρκετά κοντά τους. Το γεγονός αυτό , δεν είναι κακό , αφού δίνει τη δυνατοτητα σε κάποιον που θέλει να χρησιμοποιήσει την εφαρμογή Camellia_1 και ενδιαφέρεται για την ελάχιστη κατανάλωση ισχύος της , να διαλέξει ποιές από τις παραμέτρους των παραπάνω μετρήσεων , τον βολεύουν καλύτερα.

Το διάγραμμα 6 , που ακολουθεί , δείχνει την ποσοστιαία βελτίωση της κατανάλωσης ισχύος , του αλγόριθμου , σε σχέση με τον τύπο μέτρησης.



Διάγραμμα 6. Ποσοστιαία βελτίωση

Φαίνεται , λοιπόν , ότι οι βέλτιστες μετρήσεις c και l έχουν την μεγαλύτερη ποσοστιαία βελτίωση , έναντι των άλλων μετρήσεων , με ποσοστό 17,39%.

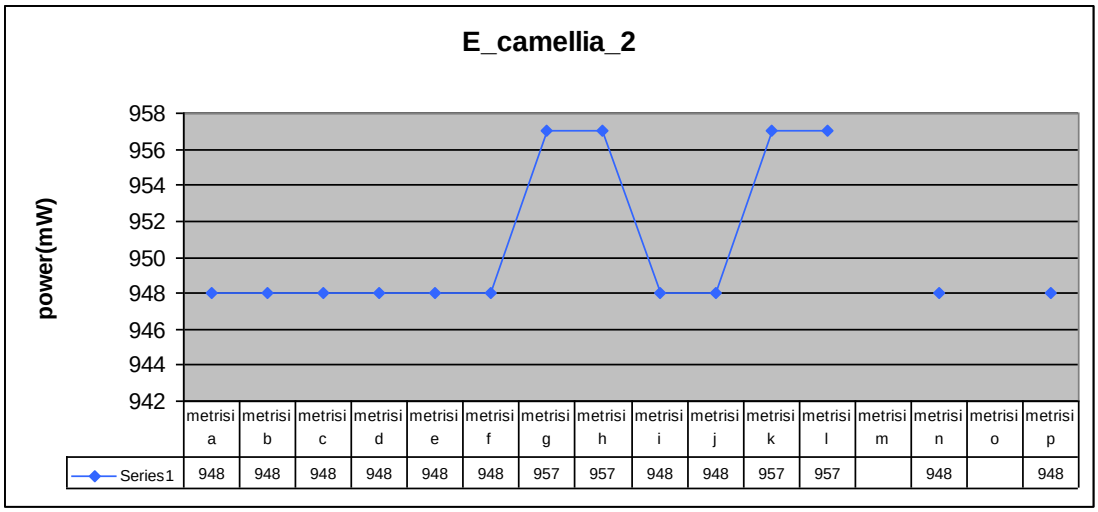
E_camellia_2

Για τον αλγόριθμο E_camellia_2 έχουμε μετρήσει τις τιμές κατανάλωσης ισχύος , που φαίνονται στον πίνακα 8.

Τύπος μέτρησης	**design name**	**optimization goal (speed/ area)**	**optimization effort (normal/ high)**	**map effort level (standar/ high)**	**place & route mode (normal/ multi)**	**power reduction (on/off)**	**Power (mW)**
A	E_camelia_2	speed	high	standar	normal	off	948
B	E_camelia_2	area	high	standar	normal	off	948
C	E_camelia_2	speed	high	high	normal	off	948
D	E_camelia_2	area	high	high	normal	off	948
E	E_camelia_2	speed	high	standar	multi	off	948
F	E_camelia_2	area	high	standar	multi	off	948
G	E_camelia_2	speed	high	high	multi	off	957
H	E_camelia_2	area	high	high	multi	off	957
I	E_camelia_2	speed	high	standar	normal	on	948
J	E_camelia_2	area	high	standar	normal	on	948
K	E_camelia_2	speed	high	high	normal	on	957
L	E_camelia_2	area	high	high	normal	on	957
M	E_camelia_2	speed	high	standar	multi	on	
N	E_camelia_2	area	high	standar	multi	on	948
O	E_camelia_2	speed	high	high	multi	on	
P	E_camelia_2	area	high	high	multi	on	948

Πίνακας 8. Τιμές κατανάλωσης ισχύος για τον αλγόριθμο E_camellia_2.

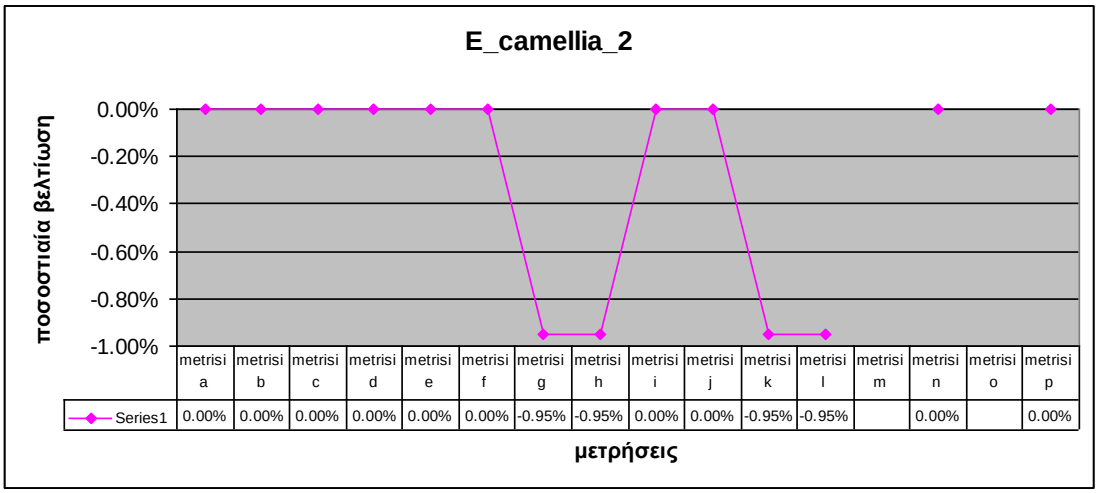
Το διάγραμμα 7 , που ακολουθεί , δείχνει τις τιμές κατανάλωσης ισχύος , για κάθε μέτρηση , δηλαδή για κάθε συνδυασμό παραμέτρων.



Διάγραμμα 7. Κατανάλωση ισχύος σε σχέση με το είδος μέτρησης

Εδώ , το πρώτο που παρατηρούμε , είναι ότι λείπουν δύο τιμές. Είναι τα δύο από τα τρία bit αρχεία , που δεν καταφέραμε να παράγουμε , και άρα να πάρουμε μέτρηση. Το δεύτερο που παρατηρούμε είναι πως η κατανάλωση ισχύος , είναι ίση *και* ελάχιστη για τις περισσότερες μετρήσεις. Το γεγονός αυτό , δίνει τη δυνατοτητα σε κάποιον που θέλει να χρησιμοποιήσει την εφαρμογή Camellia_2 και ενδιαφέρεται για την ελάχιστη κατανάλωση ισχύος της , να διαλέξει ποιές από τις παραμέτρους των παραπάνω μετρήσεων , τον βολεύουν καλύτερα.

Το διάγραμμα 8 , που ακολουθεί , δείχνει την ποσοστιαία βελτίωση της κατανάλωσης ισχύος , του αλγόριθμου , σε σχέση με τον τύπο μέτρησης.



Διάγραμμα 8. Ποσοστιαία βελτίωση

Όπως και στο E_basicDES , δεν έχουμε βελτίωση της κατανάλωσης ισχύος. Οι καλύτερες μετρήσεις είναι αυτές που διατηρούν την ποσοστιαία διαφορά τους , στο 0%.

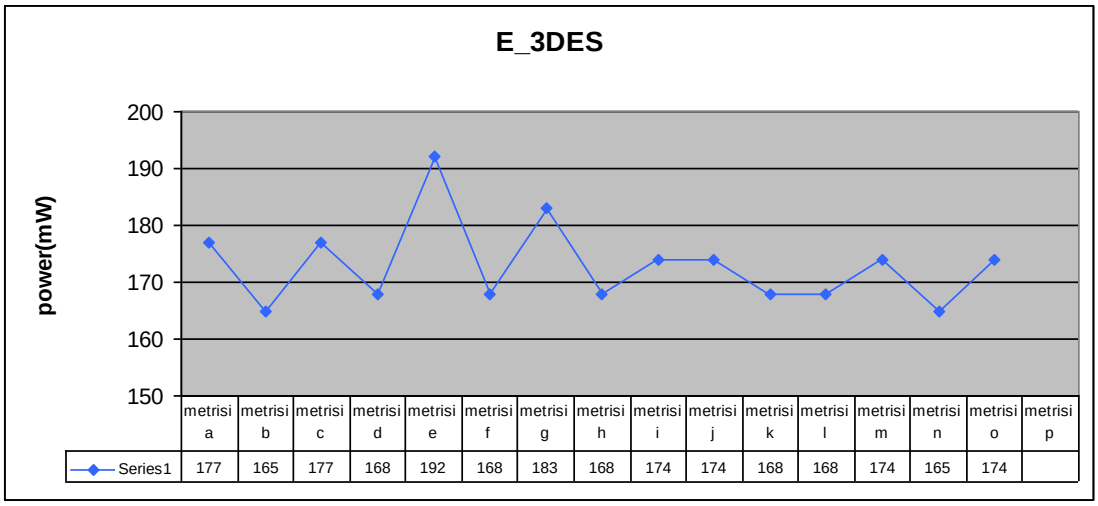
E_3des

Για τον αλγόριθμο E_3des έχουμε μετρήσει τις τιμές κατανάλωσης ισχύος , που φαίνονται στον πίνακα 9.

Τύπος μέτρησης	**design name**	**optimization goal (speed/ area)**	**optimization effort (normal/ high)**	**map effort level (standar/ high)**	**place & route mode (normal/ multi)**	**power reduction (on/off)**	**Power (mW)**
A	E_3des	speed	high	standar	normal	off	177
B	E_3des	area	high	standar	normal	off	165
C	E_3des	speed	high	high	normal	off	177
D	E_3des	area	high	high	normal	off	168
E	E_3des	speed	high	standar	multi	off	192
F	E_3des	area	high	standar	multi	off	168
G	E_3des	speed	high	high	multi	off	183
H	E_3des	area	high	high	multi	off	168
I	E_3des	speed	high	standar	normal	on	174
J	E_3des	area	high	standar	normal	on	174
K	E_3des	speed	high	high	normal	on	168
L	E_3des	area	high	high	normal	on	168
M	E_3des	speed	high	standar	multi	on	174
N	E_3des	area	high	standar	multi	on	165
O	E_3des	speed	high	high	multi	on	174
P	E_3des	area	high	high	multi	on	

Πίνακας 9. Τιμές κατανάλωσης ισχύος για τον αλγόριθμο E_3des.

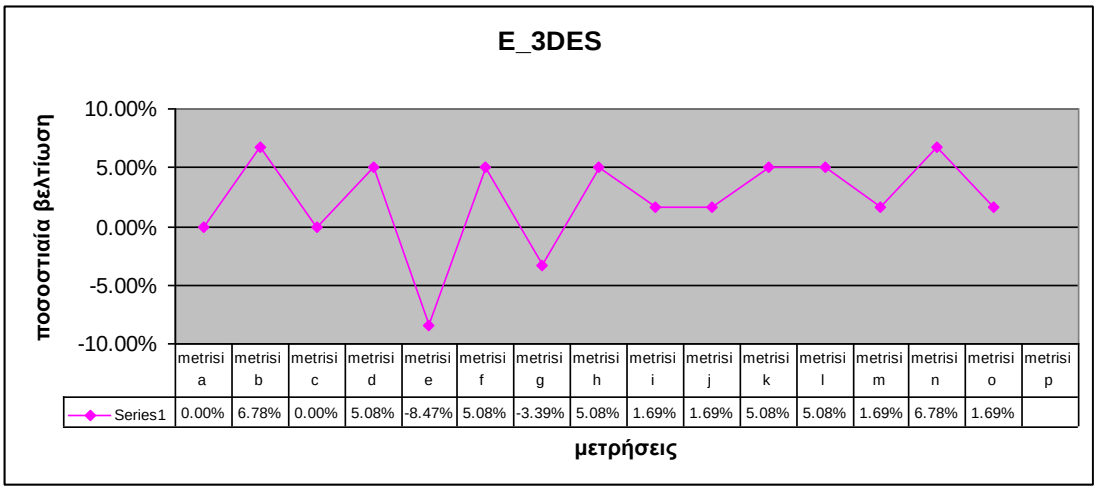
Το διάγραμμα 9 , που ακολουθεί , δείχνει τις τιμές κατανάλωσης ισχύος , για κάθε μέτρηση , δηλαδή για κάθε συνδυασμό παραμέτρων.



Διάγραμμα 9. Κατανάλωση ισχύος σε σχέση με το είδος μέτρησης

Εδώ , το πρώτο που παρατηρούμε , για τελευταία φορά , είναι ότι λείπει η τιμή της τελευταίς μέτρησης ρ(αντιστοιχεί στο τρίτο bit αρχείο , που δεν καταφέραμε να παράγουμε). Το δεύτερο που παρατηρούμε είναι πως η ελάχιστη κατανάλωση ισχύος , αντιστοιχεί στις μετρήσεις b και n. Το γεγονός αυτό , δίνει τη δυνατοτητα σε κάποιον που θέλει να χρησιμοποιήσει την εφαρμογή 3DES και ενδιαφέρεται για την ελάχιστη κατανάλωση ισχύος της , να διαλέξει ποιές από τις παραμέτρους των παραπάνω μετρήσεων , τον βολεύουν καλύτερα.

Το διάγραμμα 10 , που ακολουθεί , δείχνει την ποσοστιαία βελτίωση της κατανάλωσης ισχύος , του αλγόριθμου , σε σχέση με τον τύπο μέτρησης.



Διάγραμμα 10. Ποσοστιαία βελτίωση

Φαίνεται , λοιπόν , ότι οι βέλτιστες μετρήσεις b και n έχουν την μεγαλύτερη ποσοστιαία βελτίωση , έναντι των άλλων μετρήσεων , με ποσοστό 6,78%.

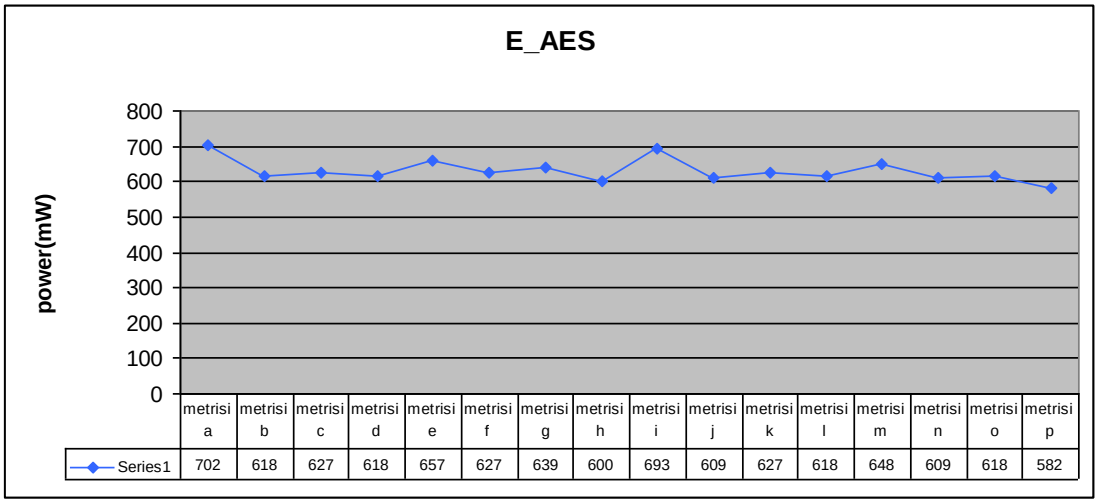
E_aes

Για τον αλγόριθμο E_aes έχουμε μετρήσει τις τιμές κατανάλωσης ισχύος , που φαίνονται στον πίνακα 10.

Τύπος μέτρησης	**design name**	**optimization goal (speed/ area)**	**optimization effort (normal/ high)**	**map effort level (standar/ high)**	**place & route mode (normal/ multi)**	**power reduction (on/off)**	**Power (mW)**
A	E_aes	speed	high	standar	normal	off	702
B	E_aes	area	high	standar	normal	off	618
C	E_aes	speed	high	high	normal	off	627
D	E_aes	area	high	high	normal	off	618
E	E_aes	speed	high	standar	multi	off	657
F	E_aes	area	high	standar	multi	off	627
G	E_aes	speed	high	high	multi	off	639
H	E_aes	area	high	high	multi	off	600
I	E_aes	speed	high	standar	normal	on	693
J	E_aes	area	high	standar	normal	on	609
K	E_aes	speed	high	high	normal	on	627
L	E_aes	area	high	high	normal	on	618
M	E_aes	speed	high	standar	multi	on	648
N	E_aes	area	high	standar	multi	on	609
O	E_aes	speed	high	high	multi	on	618
P	E_aes	area	high	high	multi	on	582

Πίνακας 10. Τιμές κατανάλωσης ισχύος για τον αλγόριθμο E_aes.

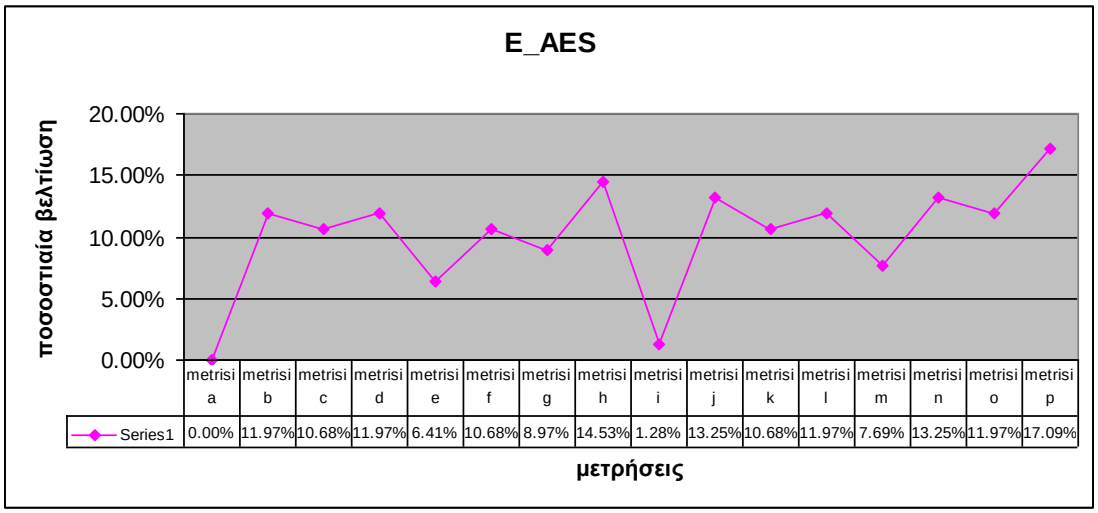
Το διάγραμμα 11 , που ακολουθεί , δείχνει τις τιμές κατανάλωσης ισχύος , για κάθε μέτρηση , δηλαδή για κάθε συνδυασμό παραμέτρων.



Διάγραμμα 11. Κατανάλωση ισχύος σε σχέση με το είδος μέτρησης

Εδώ , παρατηρούμε , πως η ελάχιστη κατανάλωση ισχύος , αντιστοιχεί στην μετρήση ρ. Αυτό σημαίνει πως , κάποιος που θέλει να χρησιμοποιήσει την εφαρμογή AES και ενδιαφέρεται για την ελάχιστη κατανάλωση ισχύος της , θέτει απλά τις παραμέτρους που αντιστοιχούν στη μέτρηση ρ , πριν τρέξει τον αλγόριθμο.

Το διάγραμμα 12 , που ακολουθεί , δείχνει την ποσοστιαία βελτίωση της κατανάλωσης ισχύος , του αλγόριθμου , σε σχέση με τον τύπο μέτρησης.



Διάγραμμα 12. Ποσοστιαία βελτίωση

Βλέπουμε , καθαρά λοιπόν , ότι η βέλτιστη μέτρηση p έχει την μεγαλύτερη ποσοστιαία βελτίωση , όπως είναι αναμενόμενο , έναντι των άλλων μετρήσεων , με ποσοστό 17,09%.

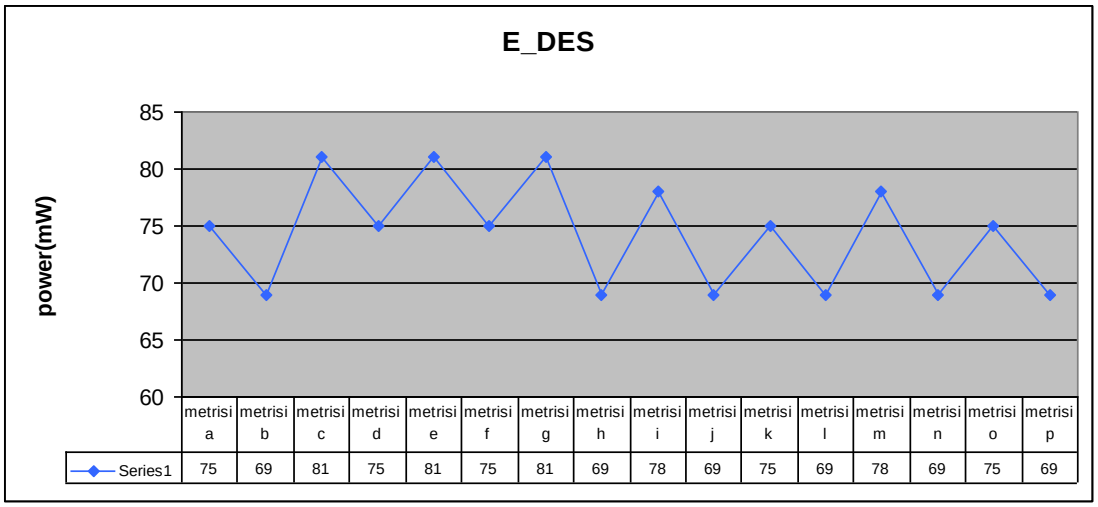
E_des

Για τον αλγόριθμο E_des έχουμε μετρήσει τις τιμές κατανάλωσης ισχύος , που φαίνονται στον πίνακα 11.

Τύπος μέτρησης	**design name**	**optimization goal (speed/ area)**	**optimization effort (normal/ high)**	**map effort level (standar/ high)**	**place & route mode (normal/ multi)**	**power reduction (on/off)**	**Power (mW)**
A	E_des	speed	high	standar	normal	off	75
B	E_des	area	high	standar	normal	off	69
C	E_des	speed	high	high	normal	off	81
D	E_des	area	high	high	normal	off	75
E	E_des	speed	high	standar	multi	off	81
F	E_des	area	high	standar	multi	off	75
G	E_des	speed	high	high	multi	off	81
H	E_des	area	high	high	multi	off	69
I	E_des	speed	high	standar	normal	on	78
J	E_des	area	high	standar	normal	on	69
K	E_des	speed	high	high	normal	on	75
L	E_des	area	high	high	normal	on	69
M	E_des	speed	high	standar	multi	on	78
N	E_des	area	high	standar	multi	on	69
O	E_des	speed	high	high	multi	on	75
P	E_des	area	high	high	multi	on	69

Πίνακας 11. Τιμές κατανάλωσης ισχύος για τον αλγόριθμο E_des.

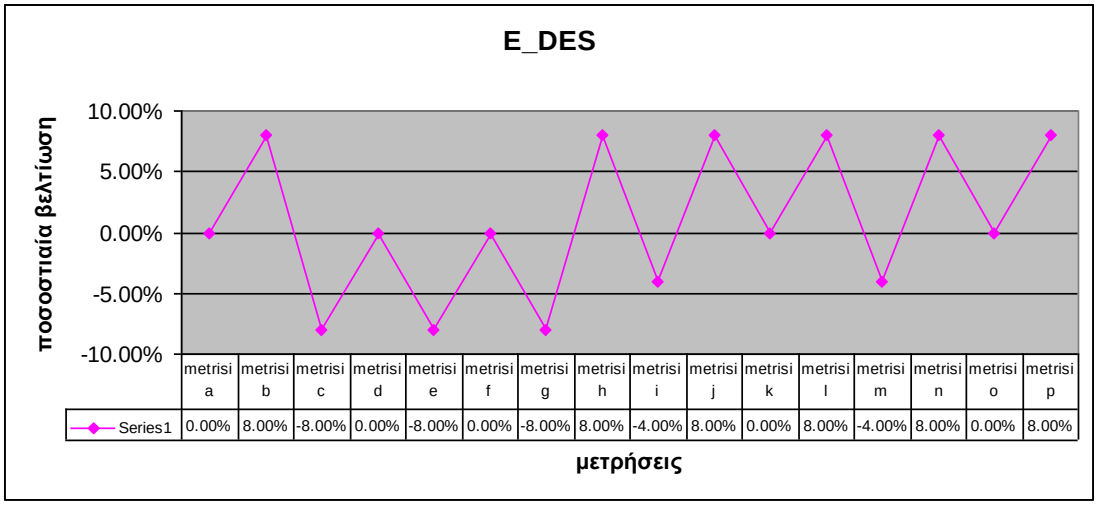
Το διάγραμμα 13 , που ακολουθεί , δείχνει τις τιμές κατανάλωσης ισχύος , για κάθε μέτρηση , δηλαδή για κάθε συνδυασμό παραμέτρων.



Διάγραμμα 13. Κατανάλωση ισχύος σε σχέση με το είδος μέτρησης

Εδώ , παρατηρούμε , ότι μικρότερη κατανάλωση ισχύος έχουμε για έξι διαφορετικές μετρήσεις. Πρόκειται για τις μετρήσεις b , h , j , l , n και p. Το γεγονός αυτό , δεν είναι κακό , αφού δίνει τη δυνατοτητα σε κάποιον που θέλει να χρησιμοποιήσει την εφαρμογή DES και ενδιαφέρεται για την ελάχιστη κατανάλωση ισχύος της , να διαλέξει ποιές από τις παραμέτρους των παραπάνω μετρήσεων , τον βολεύουν καλύτερα.

Το διάγραμμα 14 , που ακολουθεί , δείχνει την ποσοστιαία βελτίωση της κατανάλωσης ισχύος , του αλγόριθμου , σε σχέση με τον τύπο μέτρησης.



Διάγραμμα 14. Ποσοστιαία βελτίωση

Φαίνεται , λοιπόν , ότι οι βέλτιστες μετρήσεις b , h , j , l , n και p έχουν την μεγαλύτερη ποσοστιαία βελτίωση , έναντι των άλλων μετρήσεων , με ποσοστό 8,00%.

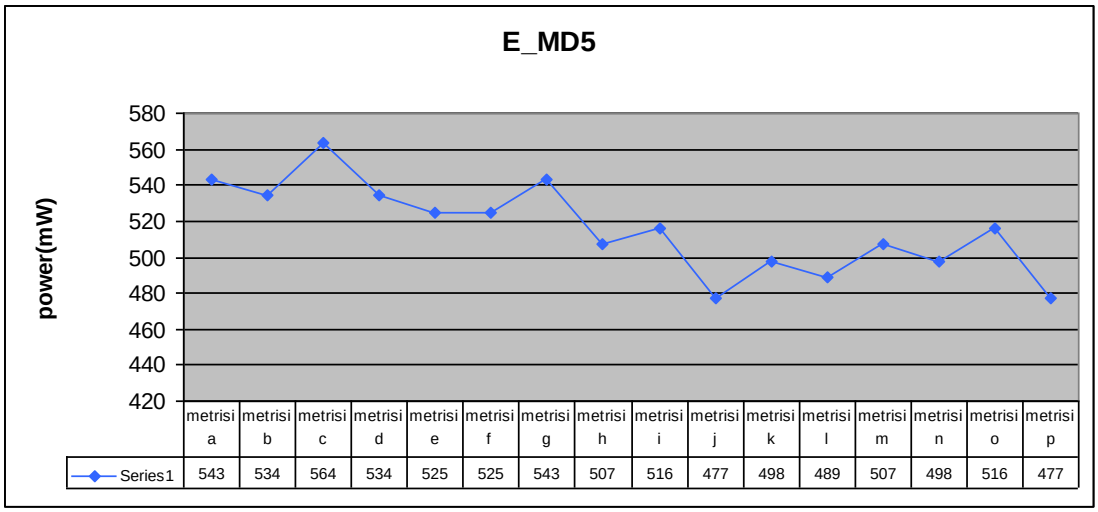
E_md5

Για τον αλγόριθμο E_md5 έχουμε μετρήσει τις τιμές κατανάλωσης ισχύος , που φαίνονται στον πίνακα 12.

Τύπος μέτρησης	**design name**	**optimization goal (speed/ area)**	**optimization effort (normal/ high)**	**map effort level (standar/ high)**	**place & route mode (normal/ multi)**	**power reduction (on/off)**	**Power (mW)**
A	E_md5	speed	high	standar	normal	off	543
B	E_md5	area	high	standar	normal	off	534
C	E_md5	speed	high	high	normal	off	564
D	E_md5	area	high	high	normal	off	534
E	E_md5	speed	high	standar	multi	off	525
F	E_md5	area	high	standar	multi	off	525
G	E_md5	speed	high	high	multi	off	543
H	E_md5	area	high	high	multi	off	507
I	E_md5	speed	high	standar	normal	on	516
J	E_md5	area	high	standar	normal	on	477
K	E_md5	speed	high	high	normal	on	498
L	E_md5	area	high	high	normal	on	489
M	E_md5	speed	high	standar	multi	on	507
N	E_md5	area	high	standar	multi	on	498
O	E_md5	speed	high	high	multi	on	516
P	E_md5	area	high	high	multi	on	477

Πίνακας 12. Τιμές κατανάλωσης ισχύος για τον αλγόριθμο E_md5.

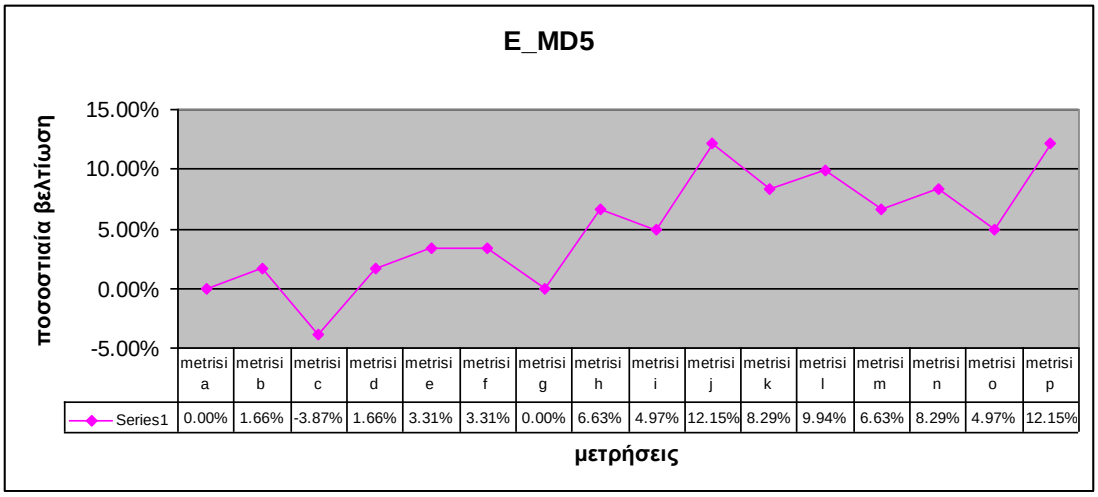
Το διάγραμμα 15 , που ακολουθεί , δείχνει τις τιμές κατανάλωσης ισχύος , για κάθε μέτρηση , δηλαδή για κάθε συνδυασμό παραμέτρων.



Διάγραμμα 15. Κατανάλωση ισχύος σε σχέση με το είδος μέτρησης

Εδώ , παρατηρούμε , ότι μικρότερη κατανάλωση ισχύος έχουμε για δύο διαφορετικές μετρήσεις. Πρόκειται για τις μετρήσεις j και p. Το γεγονός αυτό , δεν είναι κακό , αφού δίνει τη δυνατοτητα σε κάποιον που θέλει να χρησιμοποιήσει την εφαρμογή MD5 και ενδιαφέρεται για την ελάχιστη κατανάλωση ισχύος της , να διαλέξει ποιές από τις παραμέτρους των παραπάνω μετρήσεων , τον βολεύουν καλύτερα.

Το διάγραμμα 16 , που ακολουθεί , δείχνει την ποσοστιαία βελτίωση της κατανάλωσης ισχύος , του αλγόριθμου , σε σχέση με τον τύπο μέτρησης.



Διάγραμμα 16. Ποσοστιαία βελτίωση

Βλέπουμε , καθαρά λοιπόν , ότι οι βέλτιστες μετρήσεις j και p έχουν την μεγαλύτερη ποσοστιαία βελτίωση , όπως είναι αναμενόμενο , έναντι των άλλων μετρήσεων , με ποσοστό 12,15%.

Τώρα , που αναλύσαμε τις μετρήσεις κατανάλωσης ισχύος και βρήκαμε τις βέλτιστες περιπτώσεις , για τον κάθε έναν αλγόριθμο χωριστά , πάμε να τις υπολογίσουμε και για όλους τους αλγόριθμους , συνολικά. Σκοπός μας , δηλαδή , είναι να βρούμε ποιά από τις μετρήσεις a , b , c , d , e , f , g , h , i , j , k , l , m , n , o και p , παρουσιάζει την μεγαλύτερη βελτιστοποίηση κατανάλωσης ισχύος , και για τους οχτώ αλγόριθμους μαζί.

Για να το πετύχουμε αυτό , εργαζόμαστε ως εξής :

Όπως αναφέραμε και προηγουμένως έχουμε θέσει την μέτρηση α , κάθε αλγορίθμου , ως default. Ξεκινώντας , βρίσκουμε τη διαφορά που έχουν οι υπόλοιπες μετρήσεις b , c , d κ.λ.π. από την παραπάνω μέτρηση default , που αντιστοιχεί στον κάθε φορά αλγόριθμο.(Η διαφορά της μέτρησης α από την μέτρηση α , προφανώς είναι μηδέν).

Για παράδειγμα , για τον αλγόριθμο E_basicrsa , έχουμε τιμή default ίση με 201. Αφαιρούμε τη τιμή της μέτρησης b από τη default τιμή , της μέτρησης c από τη default τιμή , της μέτρησης d από τη default τιμή κ.ό.κ. , εώς ότου έχουμε δεκαέξι νέες διαφορετικές τιμές. Τώτα , για τον αλγόριθμο E_basicdes , ακολουθούμε την ίδια διαδικασία με τη διαφορά , ότι τώρα έχουμε έχουμε τιμή default ίση με 93.

Επαναλαμβάνουμε αυτή τη διαδικασία και για τους υπόλοιπους αλγόριθμους και συγκεντρώνουμε 125 καινούργιες τιμές , που δείχνουν τη διαφορά κατανάλωσης ισχύος της κάθε μέτρησης , από την αντιστοιχη default τιμή τού κάθε φορά αλγορίθμου.

Συνεχίζουμε , προσθέτοντας τις μετρήσεις α , όλων των αλγορίθμων, μαζί , όλες τις μετρήσεις b μαζί , όλες τις μετρήσεις c μαζί , όλες τις μετρήσεις d μαζί κ.λ.π. και πέρνουμε τις δεκαέξι παρακάτω τελικές τιμές.

Τύπος μέτρησης	Αθροιστική διαφορά από default
A	0
B	126
C	183
D	204
E	-33
F	159
G	6
H	156
I	48
J	255
K	189
L	288
M	75
N	264
O	129
P	306

Πίνακας 13. Αθροιστική διαφορά ,ανά μέτρηση

Οι τιμές του πίνακα 13 , μας δείχνουν την συνολική διαφορά , για όλα τα designs , της τιμής κατανάλωσης ισχύος , της κάθε μέτρησης , από την default τιμη κατανάλωσης ισχύος , που τους αντιστοιχεί. Όσο μεγαλύτερη είναι η συνολική τιμή διαφοράς , τόσο μεγαλύτερη βελτίωση είχαμε. Στην ουσία , ο παραπάνω πίνακας , μας δείχνει τη βελτίωση που είχαμε στην κατανάλωση ισχύος , για το σύνολο των αλγορίθμων μας , ανά τύπο μέτρησης.

Με βάση τον παραπάνω πίνακα , προκύπτει η ακόλουθη γραφική παράσταση (διάγραμμα 17).

Διάγραμμα 17. Βελτίωση κατανάλωσης ισχύος , για όλους τους αλγόριθμους , ανά τύπο μέτρησης

Όπως , είναι φανερό , λοιπόν , από το διάγραμμα 17 , μεγαλύτερη βελτίωση της κατανάλωσης ισχύος πετύχαμε με την μέτρηση ρ. Αυτό σημαίνει ότι με τις παραμέτρους , που αντιπροσωπεύει η μέτρηση ρ , είχαμε ελάττωση της κατανάλωσης ισχύος , κατά 306mW. Αντιθέτως , χειρότερη είναι η μέτρηση e , με την οποία είχαμε αυξηση της κατανάλωσης ισχύος κατά 33mW.

Με βάση το διάγραμματα 17 και τον πίνακα 13 , οδήγουμαστε στο συμπέρασμα , πως καλύτερη επιλογή για την βελτιστοποίηση της κατανάλωσης ισχύος , είναι η μέτρηση ρ και όλες οι παράμετροι σύνθεσης και υλοποίησης , που αυτή αντιπροσωπεύει.

Βέλτιστος τύπος μέτρησης :

Τύπος μέτρησης	**optimization goal (speed/ area)**	**optimization effort (normal/ high)**	**map effort level (standar/ high)**	**place & route mode (normal/ multi)**	**power reduction (on/off)**
P	area	high	high	multi	on

Χείριστος τύπος μέτρησης :

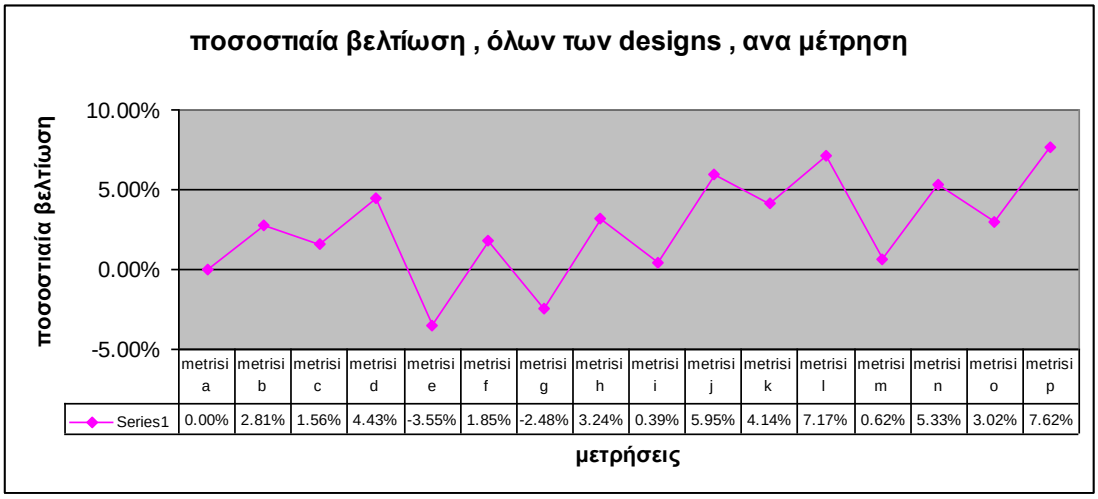
Τύπος μέτρησης	**optimization goal (speed/ area)**	**optimization effort (normal/ high)**	**map effort level (standar/ high)**	**place & route mode (normal/ multi)**	**power reduction (on/off)**
E	speed	high	standar	multi	off

Ομοίως , εργαζόμαστε και για την εύρεση της ποσοστιαίας βελτίωσης της κατανάλωσης ισχύος , για όλους τους αλγόριθμους συνολικά. Τα αποτελέσματά μας , δίνονται στον ακόλουθο πίνακα 14.

Τύπος μέτρησης	**Ποσοστιαία διαφορά από default**
A	0.00%
B	2.81%
C	1.56%
D	4.43%
E	-3.55%
F	1.85%
G	-2.48%
H	3.24%
I	0.39%
J	5.95%
K	4.14%
L	7.17%
M	0.62%
N	5.33%
O	3.02%
P	7.62%

Πίνακας 14. Ποσοστιαία βελτίωση , ανά μέτρηση

Από τον προηγούμενο πίνακα 14 , προκύπτει και η ακόλουθη γραφική παράσταση (διάγραμμα 18).



Διάγραμμα 18. Ποσοστιαία βελτίωση κατανάλωσης ισχύος , για όλους τους αλγόριθμους , ανά τύπο μέτρησης

Όπως , φαίνεται και από το παραπάνω διάγραμμα 18 , την μεγαλύτερη ποσοστιαία βελτίωση , παρουσιάζει ο τύπος μέτρησης p με 7,62% και αμέσως μετά ο l με 7,17%. Αυτό σημαίνει , ότι κατά μέσο όρο για όλους τους αλγόριθμους , το μεγαλύτερο ποσοστό βελτίωσης είχαμε για τη μέτρηση p.

Τύπος μέτρησης	**optimization goal (speed/ area)**	**optimization effort (normal/ high)**	**map effort level (standar/ high)**	**place & route mode (normal/ multi)**	**power reduction (on/off)**
P	area	high	high	multi	on

Αντίθετα , μέση ποσοστιαία αύξηση της κατανάλωσης ισχύος , με ποσοστό 3,55% , είχαμε για τη μέτρηση E.

Τύπος μέτρησης	**optimization goal (speed/ area)**	**optimization effort (normal/ high)**	**map effort level (standar/ high)**	**place & route mode (normal/ multi)**	**power reduction (on/off)**
E	speed	high	standar	multi	off

Οι παραπάνω μετρήσεις , πιο αναλυτικά , δείχνουν ότι μεγαλύτερη ποσοστιαία βελτίωση της κατανάλωσης ισχύος , είχαμε όταν θέσαμε ως optimization goal την area , το mapping effort στο high , το place & route mode στο high και το power reduction στο on. Δηλαδή , ως στόχο βελτιστοποίησης , πρέπει να θέσουμε να μειωθεί η έκταση , η διαδικασία ανάθεσης των στοιχείων λογικής σε

συγκεκριμένα φυσικά στοιχεία (mapping) , πρέπει να γίνει στο υψηλότερο επίπεδο high , η τοποθετηση και δρομολόγηση του design στους χρονικούς περιορισμούς (place & route mode) , γίνεται multi pass place and route και τέλος , η διαδικασία power reduction, που καθορίζει αν θα γίνει βελτιστοποίηση της δρομολόγησης , με σκοπό την ελάττωση της κατανάλωσης ισχύος , πρέπει να είναι προφανώς on.

Αντίθετα , για τη μεγαλύτερη ποσοστιαία αύξηση της κατανάλωσης ισχύος , το μόνο που δεν αλλάζει στις παραμέτρους μας , είναι το place & route mode , που γίνεται με multi pass place and route. Ως στόχο βελτιστοποίησης , πρέπει να θέσουμε να αυξηθεί η ταχύτητα , το map effort level είναι standard και το power reduction γίνεται off.

7. Βιβλιογραφία

[1] “Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet” , Xilinx Inc, 05/11/07 , pages : 1 - 430

[2] “AN39 Current measurement applications handbook” , Zetex semiconductors , Peter Abiodun Bode, 05/01/08 , pages : 1- 42

[3] “Power Estimation and Power Measurement of Xilinx Virtex FPGAs: Trade-offs and Limitations” , Juergen Becker, Michael Huebner, Michael Ullmann , 2003 , pages 1 - 6

[4] “Dynamic Power Consumption in Virtex™-II FPGA Family” , Li Shang, Alireza S Kaviani, Kusuma Bathala , pages : 1 - 8

[5] “The 40% Performance Advantage of Virtex-II Pro FPGAs over Competitive PLDs” , Xilinx Inc, Hitesh Patel , 01/03/04 , pages : 1 - 7

[6] “Power consumption estimations vs measurements for FPGA-based security cores” , Dimitrios Meintanis , Ioannis Papaefstathiou , pages : 1 - 6

[7] “On the Power Consumption of security algorithms employed in wireless networks” , Dimitrios Meintanis , Ioannis Papaefstathiou , pages : 1 – 5

[8] “DES/TripleDes IP Cores :: Overview” , Rudolf Usselmann , 22/05/07 , page 1 , *www.opencores.org*

[9] “Σχεδιασμός και ανάπτυξη συστήματος μέτρησης παραμέτρων ηλεκτρικής ισχύος με χρήση FPGA” , Ioannis Kantaris , 2009 , pages : 39 – 54

[10] “AES128 :: Overview” , Hemanth , 28/12/04 , page 1 , *www.opencores.org*

[11] “System C / Verilog MD5 :: Overview” , Javier Castillo Villar , 15/09/05 , page 1 , *www.opencores.org*

[12] “Virtex-II Pro™ FPGAs The Highest System Performance The Lowest System Cost” , Xilinx Inc, 2004 , pages : 1 – 2

[13] “Basic RSA Encryption Engine :: Overview” , Srmcqueen , 15/10/05 , page 1 , *www.opencores.org*

[14] “Camellia Cores :: Overview” , Paolo Fulgoni , 23/06/08 , page 1 , *www.opencores.org*

[15] “The techniques involved in authentication and conducting secure e-transactionsq” , Shekhar Govindarejan , 01/02/01 , pages :1 – 2

[16] “Το πρότυπο κρυπτογράφησης AES” , Andreas Mprokalakis , 20/01/06 , pages : 1 – 7 , *www.techarticles.gr*

[17] “Basic DES Crypto Core :: Overview” , Srmcqueen , 15/10/05 , page 1 , *www.opencores.org*

[18] Xup development board , Digilent Inc. , *www.digilentinc.com*

8. Appendix

Παραθέτονται οι top level FSM , που δημιουργήσαμε , για τον κάθε ένα αλγόριθμο ξεχωριστά.

E_basicRSA

```
-----------------------------------------------------------------------
-------------
-- Company:
-- Engineer:
--
-- Create Date:    18:58:33 03/16/2009
-- Design Name:
-- Module Name:    E_basicRSA_top - 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;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity E_basicRSA_top is
port (
clock : in std_logic ; ------to clock kai to reset
reset : in std_logic ;  -----einai oi monoi eisodoi ths fsm mas
led1  : out std_logic ; ------4 shmata elegxoy
led2  : out std_logic ; ------antistoixoun se 4 leds
led3  : out std_logic ; --me ta opoia elegxoume an o
                                --algoriumos trexei
led4  : out std_logic    ------swsta
      );
end E_basicRSA_top;

architecture Behavioral of E_basicRSA_top is

component RSACypher is
Generic (KEYSIZE: integer := 32);
    Port (indata: in std_logic_vector(KEYSIZE-1 downto 0);
                  inExp: in std_logic_vector(KEYSIZE-1 downto 0);
                  inMod: in std_logic_vector(KEYSIZE-1 downto 0);
```

```
                    cypher: out std_logic_vector(KEYSIZE-1 downto 0);
                    clk: in std_logic;
                    ds: in std_logic;
                    reset: in std_logic;
                    ready: out std_logic
                    );
end component;


----signals
signal indata1 : std_logic_vector (31 downto 0);
signal inExp1 : std_logic_vector(31 downto 0);
signal inMod1 : std_logic_vector(31 downto 0);
signal cypher1 : std_logic_vector(31 downto 0);
signal ds1 : std_logic;
signal ready1 : std_logic;
signal stage : std_logic_vector(1 downto 0);
signal next_stage : std_logic_vector(1 downto 0);


begin


---port map

R1 : RSACypher port map (  indata1 ,
                                                        inExp1 ,
                                                        inMod1 ,
                                                        cypher1 ,
                                                        clock ,
                                                        ds1 ,
                                                        reset ,
                                                        ready1
                                                  );



process(clock,reset)
      begin
            if (reset ='1') then           --process gia to
            stage <= "00";                  ---reset mas
            ds1 <= '1';                     ---phgaine sto stage 0
            elsif (rising_edge (clock)) then  -----'h gia high reset
            stage <= next_stage;               ----sto epomeno stage
            end if;
end process;

process (stage , next_stage)
begin
case stage is
when "00" =>                      --------original stage 0
ds1 <= '1';                         --------arxikopoihseis
inExp1 <= x"00903ad9";          ----the exponent to be used on the
                             -- inExp bus,dinetai h timh x"00903ad9"
inMod1 <= x"03b2c159";           -----the modulus on the inMod
                                 --bus,dinetai h timh x"03b2c159"
indata1 <= x"00724183";        ---It expects to receive the data
                                  --block,dinetai h timh x"00724183"
                             --- to be encrypted or decrypted on the
                             -- indata bus,
```

```
next_stage <= "01";
when "01" =>
ds1 <= '1';                           ------stage 1
inExp1 <= x"12345678";             ----the exponent to be used on the
                             -- inExp bus,dinetai h timh x"12345678"
inMod1 <= x"04d5ac81";             -----the modulus on the inMod
                                 --- bus,dinetai h timh x"04d5ac81"
indata1 <= x"05050505";            ---It expects to receive the data
                                 ---block,dinetai h timh x"05050505"
next_stage <= "10";
when "10" =>
ds1 <='1';                         ------stage 2
inExp1 <= x"00205ad1";             ----the exponent to be used on the
                            --- inExp bus,dinetai h timh x"00205ad1"
inMod1 <= x"03bc26ad";             -----the modulus on the inMod
                                   -- bus,dinetai h timh x"03bc26ad"
indata1 <= x"7200ad09";            ---It expects to receive the data
                                 --block,dinetai h timh x"7200ad09"
next_stage <= "11";
when "11" =>
ds1 <='1';                           ------stage 3
inExp1 <= x"32ffef00";                ----the exponent to be used on
                           -- the inExp bus,dinetai h timh x"32ffef00"
inMod1 <= x"0a2d5c0b";                -----the modulus on the inMod
                                    -- bus,dinetai h timh x"0a2d5c0b"
indata1 <= x"10abcd01";              ---It expects to receive the
                            -- data block,dinetai h timh x"10abcd01"
next_stage <= "00";                 ------kleinoyme ton kyklo ths fsm
                                     --- mas , ksanagurnaei sthn arxh
when others =>
ds1 <='0';
inExp1 <= x"00000000";             -----gia otidhpote allo exoyme 0 ,
                                 ---- gia opoio la8os emfanizetai 0
inMod1 <= x"00000000";
indata1 <= x"00000000";
next_stage <= "00";
end case;

end process;

                                 -----ana8eseis sta leds
led1 <= cypher1(10);             --------ta shmata elegxou mas
led2 <= cypher1(20);             -------bits ths ejodou cypher kai to
led3 <= cypher1(15);             ---ready1 stelnontai sta 4 leds mas
led4 <= ready1;




end Behavioral;
```

E_basicDES

```
----------------------------------------------------------------------
-------------
```

```
-- Company:
-- Engineer:
--
-- Create Date:    17:41:43 03/16/2009
-- Design Name:
-- Module Name:    E_basicDES_top - 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;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity E_basicDES_top is
      port(
      clock : in std_logic;       ------to clock kai to reset
      reset : in std_logic;     ---einai oi monoi eisodoi ths fsm mas
      led1  : out std_logic;        ----4 shmata elegxoy
      led2  : out std_logic;      ------antistoixoun se 4 leds
      led3  : out std_logic;   --me ta opoia elegxoume an o
algoriumos trexei
      led4  : out std_logic        ------swsta

      );
end E_basicDES_top;

architecture Behavioral of E_basicDES_top is

component des56 is
PORT(
      indata         : IN     std_logic_vector (0 TO 63);
      inkey          : IN     std_logic_vector (0 TO 63);
      outdata        : OUT    std_logic_vector (0 TO 63);          --
----After 16 rounds, the resulting message block
-- is presented on the OUTDATA bus and the RDY signal is set high.
decipher       : IN     std_logic;               -----If the DECIPHER
                      --signal is low when the DS signal is raised,
 ---the operation will be encryption. If the DECIPHER signal
--- is high when the DS signal goes high, the operation will be
-----decryption
ds             : IN     std_logic;   ----- When the DS signal is
                                --high, encryption/decryption begins
clk            : IN     std_logic;   ------With each clock cycle, one
                                  --round of encryption is performed
rst            : IN     std_logic;
```

```
rdy_next_next_cycle : OUT    std_logic;    -- output will be ready in
                                  -- two clock cycles - optional signal
rdy_next_cycle : OUT    std_logic;      -- output will be ready in
                                -- one clock cycle - optional signal
rdy            : OUT    std_logic         -- output is ready NOW
   );
end component;

                                                    ----signals
signal indata1 :std_logic_vector (0 TO 63);
signal inkey1  :      std_logic_vector (0 TO 63);
signal outdata1 :     std_logic_vector (0 TO 63);
signal decipher1 :      std_logic;
signal ds1 :      std_logic;
signal rdy_next_next_cycle1 :     std_logic;
signal rdy_next_cycle1 :     std_logic;
signal rdy1 :     std_logic;
signal state : std_logic_vector (0 to 2);
signal nextstate : std_logic_vector (0 to 2);


begin
                                                      ------port maps

B1 : des56 port map (
                                          indata=>indata1,
                                          inkey=>inkey1,
                                          outdata=>outdata1,
                                          decipher=>decipher1,
                                          ds=>ds1,
                                          clk=>clock,
                                          rst=>reset,

      rdy_next_next_cycle=>rdy_next_next_cycle1,

      rdy_next_cycle=>rdy_next_cycle1,
                                          rdy=>rdy1

                                          );

process(clock,reset)
begin                                 ----- 8etoume to reset
if (reset='1') then
state<= "000";
elsif (rising_edge(clock)) then
state<= nextstate;
end if;
end process;

process(state,nextstate)
begin
case state is
when "000"=>     ---------original stage 0
decipher1 <= '0';   --------arxikopoihseis
ds1 <= '0';
nextstate <="001";    ------mas stelnei kateu8eian sto state 1
when "001"=>
inkey1 <=x"133457799BBCDFF1";             ------the 64-bit key on the
                    -- inKey bus pairnei thn timh x"133457799BBCDFF1"
indata1 <=x"0123456789ABCDEF";             -----It expects to receive
                                                        -- the 64-bit
```

```
-- data block to be encrypted or decrypted on the indata bus

-----edw sto indata bus dinetai h timh x"0123456789ABCDEF"
ds1 <= '0';                                 ----- When the DS signal
                        -- is high, encryption/decryption begins
nextstate <="010";

when "010"=>
inkey1 <=x"133457799BBCDFF1";               ------the inKey bus
                        --pairnei thn timh x"133457799BBCDFF1"
indata1 <=x"0123456789ABCDEF";              -------sto indata bus
                        --dinetai h timh x"0123456789ABCDEF"
ds1 <= '1';                                -------DS signal is high
nextstate <="011";
when "011"=>
ds1 <='0';                                  -------DS signal is low
nkey1 <= x"0E329232EA6D0D73";              ------the inKey bus pairnei
                                     -- thn timh x"0E329232EA6D0D73"
indata1 <= x"8787878787878787";             -------sto indata bus
                                 -- dinetai h timh x"8787878787878787"
if (rdy1 = '1') then                 -------teleiwsan oi 16 loops??
                                     -- an nai to rdy 8a einai 1
ds1 <='1';                             ------to ds 8a ginei 1
else                                  -----an oxi
ds1 <='0';                           ------to ds 8a ginei 0
end if;
nextstate <="100";
when "100"=>
indata1 <= x"596F7572206C6970";         -------sto indata bus dinetai
                                     -- h timh x"596F7572206C6970"
ds1 <='0';                              ------to ds na ginei 0
if (rdy1 = '1') then
ds1 <='1';
else
ds1 <='0';
end if;
nextstate <="101";
when "101"=>
indata1 <= x"732061726520736D";         -------sto indata bus dinetai
                                     --h timh x"732061726520736D"
                                                 --ds1 <='0';
if (rdy1 = '1') then
ds1 <='1';
else
ds1 <='0';
end if;
nextstate <="110";
when "110"=>
indata1 <= x"6F6F746865722074";          -------sto indata bus
                                 -- dinetai h timh x"6F6F746865722074"
                                              --ds1 <='0';
if (rdy1 = '1') then
ds1 <='1';
else
ds1 <='0';
end if;
nextstate <="111";
when "111"=>
if (rdy1 = '1') then                          -------an rdy einai 1
indata1 <= x"68616E2076617365";               -------to indata bus  na
```

```
                                          --ginei x"68616E2076617365"
ds1 <= '1';                                  ------to ds na ginei 1
else                                        ------alliws
ds1 <='0';                                ------to ds na parameinei 0
indata1 <= x"6C696E650D0A0000";           -------to indata bus  na
ginei x"6C696E650D0A0000"
end if;
nextstate <="000";
                  when others =>
------when others
inkey1 <=x"133457799BBCDFF1";             -----the inKey bus pairnei
                                     -- thn timh x"133457799BBCDFF1"
indata1 <=x"0123456789ABCDEF";        -------to indata bus  na ginei
                                                --x"0123456789ABCDEF"
ds1 <= '0';                                     -----to ds na einai0
nextstate <="000";                        ----epistrofh sthn arxh


            end case;


end process;

led1 <= outdata1(27);                -------ta 4 leds mas elegxoun
led2 <= rdy_next_next_cycle1;        -------tis ejodous tou basicDES
led3 <= rdy_next_cycle1;
led4 <= rdy1;


end Behavioral;
```

E_camellia_1

```
----------------------------------------------------------------------
-------------
-- Company:
-- Engineer:
--
-- Create Date:    12:56:51 03/09/2009
-- Design Name:
-- Module Name:    E_camelia_1_top - 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;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity E_camelia_1_top is
      port(
clk      : in STD_LOGIC;     --oi duo eisodoi ths fsm mas
reset_n  : in STD_LOGIC;     ---clock kai reset
led1     : out STD_LOGIC;    ----oi tesseris ejodoi ths fsm mas
led2     : out STD_LOGIC;     ----tessera shmata elegxou
led3     : out STD_LOGIC;       -----tou enos bit
led4     : out STD_LOGIC         ----led1 , led2 , led3 , led4

      );
end E_camelia_1_top;

architecture Behavioral of E_camelia_1_top is

component CAMELLIA_IF is
       port    (
clk       : in  STD_LOGIC;
reset     : in  STD_LOGIC;
data_in   : in  STD_LOGIC_VECTOR (0 to 15);   ----ta data pou vazoume
enc_dec   : in  STD_LOGIC;           ----encryption=0 'h decryption=1
en_data   : in  STD_LOGIC;         ---enable ta data na diavastoun
next_data : out STD_LOGIC;
key_in    : in  STD_LOGIC_VECTOR (0 to 15);    ----eisodos kleidiou
k_len     : in  STD_LOGIC_VECTOR (0 to 1);        ---mhkos kleidiou
en_key    : in  STD_LOGIC;           ---enable to key na diavastei
next_key  : out STD_LOGIC;
data_out  : out STD_LOGIC_VECTOR (0 to 15);  ----h ejodos data out
out_rdy   : out STD_LOGIC          -----h ejodos einai etoimh
            );
end component;

                                                                  ---
signals
signal data_in_1   : STD_LOGIC_VECTOR (0 to 15);
signal data16      : STD_LOGIC_VECTOR (0 to 15);
signal enc_dec_1   : STD_LOGIC;
signal en_data_1   : STD_LOGIC;
signal next_data_1 : STD_LOGIC;
signal key_in_1    : STD_LOGIC_VECTOR (0 to 15);
signal k_len_1     : STD_LOGIC_VECTOR (0 to 1);
signal en_key_1    : STD_LOGIC;
signal next_key_1  : STD_LOGIC;
signal data_out_1  : STD_LOGIC_VECTOR (0 to 15);
signal out_rdy_1   : STD_LOGIC;
signal reset       : STD_LOGIC;
signal state       : STD_LOGIC_VECTOR(3 downto 0);
signal nextstate   : STD_LOGIC_VECTOR(3 downto 0);


                              -----prohgoumena shmata
                              ---pou den xreiazontai
--signal  dout_count    : STD_LOGIC_VECTOR (2 downto 0);
--signal  int_out_rdy   : STD_LOGIC;
```

```
--signal  reg_dout      : STD_LOGIC_VECTOR (0 to 127);
--signal  s_data_out   : STD_LOGIC_VECTOR (0 to 127);
--signal  s_output_rdy : STD_LOGIC;
                                                  ------8umomaste tis
                                                  -- antistoixies
                                                  ----constant KLEN_128
                              -- STD_LOGIC_VECTOR (0 to 1) := "00";
                                                  ----constant KLEN_192
                              --: STD_LOGIC_VECTOR (0 to 1) := "01";
                                                  -----constant
                 --KLEN_256 : STD_LOGIC_VECTOR (0 to 1) := "10";


begin


reset <= not reset_n;


                                  ---port maps

E1 : CAMELLIA_IF port map(
                                    clk=>clk ,
                                    reset=>reset_n ,
                                    data_in=>data_in_1,
                                    enc_dec=>enc_dec_1,
                                    en_data=>en_data_1,
                                    next_data=>next_data_1,
                                    key_in=>key_in_1,
                                    k_len=>k_len_1,
                                    en_key=>en_key_1,
                                    next_key=>next_key_1,
                                    data_out=>data_out_1,
                                    out_rdy=>out_rdy_1
                                    );


---
process(clk,reset)
begin                                                  ----process gia to
reset mas , low reset
if(reset = '0') then                                 ----an to reset
                                                         -- einai 0
data16 <= "0000000000000000";                      ----kane to data16 0
elsif (rising_edge(clk)) then                        ----alliws gia
                                                         -- reset 1
data16 <= data16 + '1';                            ----aujhsh data16 kata
                                                         -- ena
end if;                                            -----data16 = data16 + 1
end process;
--
--
process(clk,reset)
begin                                              -----process gia ta state
if(reset = '0') then                             -----gia reset 0
state <= "0000";                               ----phgaine sto arxiko state 0
elsif (rising_edge(clk)) then                    ----alliws
state <= nextstate;                              ----phgaine sto epomeno state
end if;
end process;
--
```

```
--
process(state, nextstate)
begin
case state is
when "0000" => nextstate <= "0001";   -----8etoume poio einai to
                                          --epomeno state
data_in_1   <= data16;                -----ta data mas pairnoun thn
                                          -- ekastote timh tou data16
enc_dec_1   <= '0';                   -----dialegoume encryption
k_len_1     <= "00";                 -----mhkos key 128 bits
key_in_1   <= "0100001101110010";     -----eisodos timhs kleidiou
"0100001101110010"
en_data_1 <= '1';        ----diavase data
en_key_1  <= '1';        ----diavase key


                 --
when "0001" => nextstate <= "0010";         -----8etoume poio einai
                                                -- to epomeno state
key_in_1   <= "1111100001011111";         -----eisodos timhs kleidiou
                                          -- "1111100001011111"
data_in_1 <= data16;                      -----ta data mas pairnoun
                                     -- thn ekastote timh tou data16
en_data_1 <= '1';                          ----diavase data
enc_dec_1  <= '0';                        -----dialegoume
encryption
en_key_1  <= '1';           ----diavase key
k_len_1 <= "00";            -----mhkos key 128 bits

                 --
when "0010" => nextstate <= "0011";                   -----8etoume
                                     -- poio einai to epomeno state
key_in_1   <= "1010101010101010";       -----eisodos timhs kleidiou
                                          -- "1010101010101010"
data_in_1 <= data16;                   -----ta data mas pairnoun thn
                                          -- ekastote timh tou data16
en_data_1 <= '1';                      ----diavase data
enc_dec_1  <= '0';                     -----dialegoume encryption
en_key_1  <= '1';        ----diavase key
k_len_1 <= "00";         -----mhkos key 128 bits

                 --
when "0011" => nextstate <= "0100";                   -----8etoume
                                     -- poio einai to epomeno state
key_in_1   <= "1111011010001000";      -----eisodos timhs kleidiou
                                          -- "1111011010001000"
data_in_1 <= data16;                   -----ta data mas pairnoun thn
                                          -- ekastote timh tou data16
en_data_1 <= '1';                      ----diavase data
enc_dec_1  <= '0';                     -----dialegoume encryption
en_key_1  <= '0';       ----mhn diavaseis key
k_len_1 <= "00";        -----mhkos key 128 bits

--                 --
when "0100" => nextstate <= "0101";                     -----8etoume
                                     -- poio einai to epomeno state
key_in_1   <= "0100001101100010";         -----eisodos timhs kleidiou
                                                -- "0100001101100010"
data_in_1 <= data16;                      -----ta data mas pairnoun
                                     -- thn ekastote timh tou data16
en_data_1 <= '1';                         ----diavase data
```

```
enc_dec_1  <= '0';                        -----dialegoume encryption
en_key_1  <= '0';          ----mhn diavaseis key
k_len_1 <= "01";          -----mhkos key 192 bits

                 --
when "0101" => nextstate <= "0110";                     -----8etoume
                                     -- poio einai to epomeno state
key_in_1   <= "0100000011010010";           -----eisodos timhs
                                     --kleidiou "0100000011010010"
data_in_1 <= data16;                       -----ta data mas pairnoun
                                     -- thn ekastote timh tou data16
en_data_1 <= '1';                          ----diavase data
enc_dec_1  <= '1';                        -----dialegoume decryption
en_key_1  <= '0';             ----mhn diavaseis key
k_len_1 <= "01";            -----mhkos key 192 bits

                 --
when "0110" => nextstate <= "0000";          ----8etoume poio einai
                                                -- to epomeno state
key_in_1   <= "0111110101110010";            -----eisodos timhs
                                     -- kleidiou "0111110101110010"
data_in_1 <= data16;                         -----ta data mas
                       ---pairnoun thn ekastote timh tou data16
en_data_1 <= '1';                            ----diavase data
enc_dec_1  <= '1';                            -----dialegoume
                                                -- decryption
en_key_1  <= '1';               ----diavase key
k_len_1 <= "01";                -----mhkos key 192 bits
                --

when others => nextstate <= "0000";       ----gyrna sto state 0 gia
                                          -- when others
key_in_1   <= "0000000000000000";           -----timh kleidiou
                                          -- "0000000000000000"
data_in_1 <= data16;                          -----ta data mas
                       -- pairnoun thn ekastote timh tou data16
en_data_1 <= '0';                            ----mhn diavaseis data
enc_dec_1  <= '0';                          -----dialegoume
                                          -- encryption
en_key_1  <= '0';               ----mhn diavaseis key
k_len_1 <= "00";               -----mhkos key 128 bits


--
end case;


end process;


led1  <= next_data_1;                       -----ana8esh shmatwn sta
                                                -- leds mas
led2  <= next_key_1;                      ----treis ejodoi tou 1 bit
led3  <= data_out_1(7);                   -----kai o data out tvn 16
                                          -- bits
led4  <= out_rdy_1;                       ----dialegoume tuxaia to
                                          -- bit 6

end Behavioral;
```

E_camellia_2

```
-----------------------------------------------------------------------
-------------
-- Company:
-- Engineer:
--
-- Create Date:    12:33:51 03/12/2009
-- Design Name:
-- Module Name:    E_camelia_2_256 - 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;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity E_camelia_2_256 is
port(
clk      : in STD_LOGIC;     --oi duo eisodoi ths fsm mas
reset    : in STD_LOGIC;     ---clock kai reset
led1     : out STD_LOGIC;    ----oi tesseris ejodoi ths fsm mas
led2     : out STD_LOGIC;     ----tessera shmata elegxou
led3     : out STD_LOGIC;       -----tou enos bit
led4     : out STD_LOGIC         ----led1 , led2 , led3 , led4

      );
end E_camelia_2_256;

architecture Behavioral of E_camelia_2_256 is

component CAMELLIA256 is
port(
reset      : in STD_LOGIC;
clk        : in STD_LOGIC;
input      : in STD_LOGIC_VECTOR (0 to 127);         -- input data
input_en   : in  STD_LOGIC;                           -- input enable
key        : in STD_LOGIC_VECTOR (0 to 255);          -- key
key_len    : in STD_LOGIC_VECTOR (0 to 1);            -- key lenght
enc_dec    : in STD_LOGIC;                            -- gia dec=0
                                --- exoume enc, gia dec=1 exoume dec
output     : out STD_LOGIC_VECTOR (0 to 127);    -- en/decrypted data
```

```
output_rdy : out STD_LOGIC                      -- output ready
            );



end component;
                                                ---signals
signal input1 :STD_LOGIC_VECTOR (0 to 127);
signal input_en1 :STD_LOGIC;
signal key1 :STD_LOGIC_VECTOR (0 to 255);
signal key_len1 :STD_LOGIC_VECTOR (0 to 1);
signal enc_dec1 : STD_LOGIC;
signal output1 :STD_LOGIC_VECTOR (0 to 127);
signal output_rdy1 :STD_LOGIC;
signal stage :STD_LOGIC_VECTOR (0 to 1);
signal next_stage :STD_LOGIC_VECTOR (0 to 1);


                  ------8umomaste tis antistoixies
           ----constant KLEN_128 : STD_LOGIC_VECTOR (0 to 1) := "00";
           ----constant KLEN_192 : STD_LOGIC_VECTOR (0 to 1) := "01";
           ----constant KLEN_256 : STD_LOGIC_VECTOR (0 to 1) := "10";

begin

                        ----port maps
E2_256 : CAMELLIA256 port map (
                                            reset=>reset,
                                            clk=>clk,
                                            input=>input1,
                                            input_en=>input_en1,
                                            key=>key1,
                                            key_len=>key_len1,
                                            enc_dec=>enc_dec1,
                                            output=>output1,
                                            output_rdy=>output_rdy1

                                                              );



process (clk,reset)
begin                                          ----process gia to reset mas
if (reset='1') then                     ----an reset einai 1
stage <= "00";                          ----phgaine sto arxiko stage 0
elsif (rising_edge(clk)) then            ----alliws
stage <= next_stage;                      ----phgaine sto epomeno stage
end if;
end process;

process(stage,next_stage)
begin
case stage is                                ----stage 0 ,
                                            --- arxikopoihseis
when "00" =>
input1     <= X"0123456789ABCDEFFEDCBA9876543210";      ---dinoume
                                                     --- timh sto input
key1       <= X"0123456789ABCDEFFEDCBA9876543210" &
X"00112233445566778899AABBCCDDEEFF";     ---dinoume timh sto key
key_len1   <= "00";                       ----mhkos kleidiou 128 bits
```

```
enc_dec1   <= '0';                        ----dialegoume encryption
input_en1  <= '1';                        ----diavase to input
next_stage <="01";                       ----8etoume to epomeno
                                         --stage

when "01" =>
input1     <= X"676731385496697308570656 48EABE43";    ---dinoume timh
                                                       -- sto input
key1       <= X"0123456789ABCDEFFEDCBA9876543210" &
X"00112233445566778899AABBCCDDEEFF";         ----dinoume timh sto key

key_len1   <= "00";                       ----mhkos kleidiou 128
                                         -- bits
enc_dec1   <= '1';                       ---dialegoume decryption
next_stage <="10";                        ----8etoume to epomeno
                                         -- stage

when "10" =>
input1     <= X"0123456789ABCDEFFEDCBA9876543210";    ---dinoume timh
                                                       --- sto input
key1       <= X"0123456789ABCDEFFEDCBA9876543210" &
X"00112233445566778899AABBCCDDEEFF";              ----dinoume timh
                                                       -- sto key
key_len1   <= "10";                      ----mhkos kleidiou 256 bits
enc_dec1   <= '0';                       ----dialegoume encryption
next_stage <="11";                       ----8etoume to epomeno
                                         -- stage

when "11" =>
input1     <= X"B4993401B3E996F84EE5CEE7D79B09B9";     ---dinoume
                                                  -- timh sto input
key1       <= X"0123456789ABCDEFFEDCBA9876543210" &
X"00112233445566778899AABBCCDDEEFF";     ----dinoume timh sto key
key_len1   <= "10";                   ----mhkos kleidiou 256 bits
enc_dec1   <= '1';                      ----dialegoume decryption
next_stage <="00";                 ----gurname sthn arxh ths fsm mas

when others =>                                    ----when others
input1     <= X"00000000000000000000000000000000";      ----8etoume
                                                       --- ta panta 0
key1       <= X"00000000000000000000000000000000" &
                     X"00000000000000000000000000000000";
key_len1   <= "00";
enc_dec1   <= '0';
next_stage <="00";                            --gurname sthn arxh ths
                                                  --fsm mas



            end case;



end process;

                                            ----ana8etoume times sta
                                             ---ledakia
led1<= output1(0);                         ----eixame duo output , opote
led2<= output1(57);                       ----leds 1, 2 kai 3 pairnoun
                                             ---times apo to output1
led3<= output1(99);                       -----apo to output1
```

```
led4<= output_rdy1;                        ----led4 diavazei thn timh
                                           ---tou output_rdy!


end Behavioral;
```

E_3des

```
module des3_top (clk , reset , led0 , led1 , led2 , led3 );

input clk;                                 //duo shmata eisodou
input reset;                               // clock kai reset
output led0;                               // 4 shmata ejodou
output led1;                               //led0 ,led1 , led2 , led3
output led2;
output led3;

wire [63:0] data_o;                       //ta data out se wire
reg [63:0] data_i;                         //ta data out se kataxwrhth

wire led0 = data_o[3];                     //ana8esh twn bits
wire led1 = data_o[2];                     //3, 2, 1 kai 0
wire led2 = data_o[1];                      //tou shmatos data_o
wire led3 = data_o[0];                    //sta led0 ,led1 , led2 kai
led3 antistoixa

wire decrypt = 0;
reg [5:0] roundSel;
                                                        //wire [55:0] key
= data_i[55:0] &&  64'h6666666666666666;
wire [55:0] key0 = 64'h4444444444444444;                //ana8esh timwn
sta 3 kleidia tou 3des
wire [55:0] key1 = 64'h1111111111111111;                //xrhsimopoioyme
56 apo ta 64 bits
wire [55:0] key2 = 64'h2222222222222222;
                                                                 //
                                                                //
//
always @(negedge clk or negedge reset)
begin
   if(!reset)                                             //gia reset 0
   begin
      data_i = 64'h0;                                     //arxikopoihsh
tou data in sto 0
   end
   else                                                   //allws
   begin
     data_i = data_i + 1;                                //aujhsh data in
kata ena
   end
end
//
                                                       //
                                                       //
always @(negedge clk or negedge reset)
begin
```

```
   if(!reset)                                      //gia reset 0
   begin
      roundSel = 6'b0;                            //arxikopoihsh tou
roundSel sto 0
   end
   else                                            //allws
   begin
     roundSel = roundSel + 1;                     //aujhsh roundSel
kata ena
   end
end

des3 des(data_o, data_i, key0, key1, key2, decrypt, roundSel, clk);


endmodule
```

E_aes

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.aes_package.all;

entity aes128_top is
port(
clock : in std_logic;                 ------to clock kai to reset
reset : in std_logic;                -----einai oi monoi eisodoi ths
                                           -- fsm mas
led1  : out std_logic;                 ------4 shmata elegxoy
led2  : out std_logic;                    ------antistoixoun se 4 leds
led3  : out std_logic;                     ----me ta opoia elegxoume
                                          --an o algoriumos trexei
led4  : out std_logic                          ------swsta

     );

end aes128_top;

architecture mapping of aes128_top is
--
--
component aes128_fast
port(
clk       : in std_logic;
reset     : in std_logic;
start     : in std_logic;                        -- to initiate the
                                          --encryption/decryption
                                          -- process after loading
mode      : in std_logic;                        -- to select
                                          -- encryption or decryption
load      : in std_logic;                          -- to load the
                                               -- input and keys
key       : in std_logic_vector(63 downto 0);
data_in   : in std_logic_vector(63 downto 0);
data_out  : out std_logic_vector(127 downto 0);
done      : out std_logic
```

```
      );
end component;
--
signal mode1 : std_logic;
signal start1: std_logic;
signal done1 : std_logic;
signal load1 : std_logic;
signal reset : std_logic;
signal key1   : std_logic_vector(63 downto 0);
signal datain :std_logic_vector(63 downto 0);
signal data64 :std_logic_vector(63 downto 0);
signal dataout :std_logic_vector(127 downto 0);
signal state: std_logic_vector(3 downto 0);
signal nextstate: std_logic_vector(3 downto 0);
--
BEGIN
--
reset <= not reset_n;
mode1 <= '1';                                 -------mode iso me ena
                                           -- gia na kanei encryption
led0  <= done1;                               ---------ana8esh twm
                                           --shmatwn output toy aes
led1  <= dataout(120);                        ---------sta 4 leds mas
led2  <= dataout(80);                         --------gia ton elegxo
                                           -- ths swsths leitourgias
led3  <= dataout(0);                          --------tou algori8mou
--
aes_core: aes128_fast
  port map(
      clk      => clk,
      reset    => reset,
      start    => start1,
      mode     => mode1,
      load     => load1,
      key      => key1,
      data_in  => datain,
      data_out => dataout,
      done     => done1
  );
--
--
process(clk,reset)
begin
if(reset = '1') then
data64 <= X"0000000000000000";        ----arxikopoioume ta data mas
                                           --sto 0
elsif rising_edge(clk) then              -----se ka8e kyklo tou clock
data64 <= data64 + '1';               ------kanoume aujhsh kata ena
                                           -- data = data +1
end if;
end process;
--
--
process(clk,reset)
begin
if(reset = '1') then
state <= "0000";                     ----stelnoume sto arxiko state 0
elsif falling_edge(clk) then
state <= nextstate;
end if;
end process;
```

```
--
--
process(state, done1) begin
--
case state is
when "0000" => nextstate <= "0001";          -----arxikopoihseis sto
                                                -- state 0
key1   <= X"0123456789ABCDEF";                   -----to key pairnei
                                  --thn timh X"0123456789ABCDEF"
datain <= data64;                                ------sto data in
                       --ana8etoyme thn ekastote timh tou data 64
start1 <= '0';                                    ------na arxisei to
                                  -- encryption/decryption?? oxi
load1  <= '0';                                   -----na fortwsei key
                                        -- kai dara?? oxi akoma
                 --
when "0001" => nextstate <= "0010";
key1   <= X"0123456789ABCDEF";              -----to key krataei thn
                                  --- timh X"0123456789ABCDEF"
datain <= data64;                          ------data = data +1
start1 <= '0';                              ------na arxisei to
                                  -- encryption/decryption?? oxi
load1  <= '1';                               -----na fortwsei key kai
                                        -- dara?? nai
                 --
when "0010" => nextstate <= "0011";
key1   <= X"0123456789ABCDEF";              -----to key krataei thn
-                                 -- timh X"0123456789ABCDEF"
datain <= data64;                         ------data = data +1
start1 <= '0';                              ------na arxisei to
                                  ---encryption/decryption?? oxi
load1  <= '1';                               -----na fortwsei key kai
                                        ---dara?? nai
                 --
when "0011" => nextstate <= "0100";
key1   <= X"0123456789ABCDEF";              -----to key krataei thn
                                  ---timh X"0123456789ABCDEF"
datain <= data64;                          ------data = data +1
start1 <= '0';                              ------na arxisei to
                                  ---encryption/decryption?? oxi
load1  <= '0';                              -----na fortwsei key kai
                                        ---dara?? oxi
                 --
when "0100" => nextstate <= "0101";
key1   <= X"0123456789ABCDEF";              -----to key krataei thn
                                  ---timh X"0123456789ABCDEF"
datain <= data64;                            ------data = data +1
start1 <= '1';                               ------na arxisei to
                                  ---- encryption/decryption?? nai
load1  <= '0';                               -----na fortwsei key kai
                                        ---- dara?? oxi
                 --
when "0101" => nextstate <= "0110";
key1   <= X"0123456789ABCDEF";              -----to key krataei thn
                                  ----timh X"0123456789ABCDEF"
datain <= data64;                          ------data = data +1
start1 <= '0';                              ------na arxisei to
                                  --- encryption/decryption?? oxi
load1  <= '0';                              -----na fortwsei key kai
                                        ----dara?? oxi
                 --
```

```
when "0110" => if(done1 = '1') then           -----exoume done 1??an
                                                -- nai
nextstate <= "0000";                       -----phgaine sto state 0
else                                        -----alliws
nextstate <= "0110";                     -------kane pali auto to state
end if;
               --
key1   <= X"0123456789ABCDEF";              -----to key krataei thn
                                     --timh X"0123456789ABCDEF"
datain <= data64;                           ------data = data +1
start1 <= '0';                              ------na arxisei to
                                     -- encryption/decryption?? oxi
load1  <= '0';                              -----na fortwsei key kai
                                           ---dara?? oxi


                 --
--  when "1010" => nextstate <= "0000";
--  when "1011" => nextstate <= "0000";
--  when "1100" => nextstate <= "0000";
--  when "1101" => nextstate <= "0000";
--  when "1110" => nextstate <= "0000";
--  when "1111" => nextstate <= "0000";
                 --
when others => nextstate <= "0000";              -------epistrofh sto
                                                   ---state 0
key1   <= X"0123456789ABCDEF";              -----to key krataei thn
                                     ---- timh X"0123456789ABCDEF"
datain <= data64;                           ------data = data +1
start1 <= '0';                              ------na arxisei to
                                     ---encryption/decryption?? oxi
load1  <= '0';                              -----na fortwsei key kai
                                                --- dara?? oxi


end case;
--
end process;



end mapping;
```

E_des

```
module des_top(clk, reset,led0,led1,led2,led3);

input clk;                                //duo shmata eisodou
input reset;                              // clock kai reset
output led0;                              // 4 shmata ejodou
output led1;                              //led0 ,led1 , led2 , led3
output led2;
output led3;

wire [63:0] data_o;                      //ta data out se wire
reg [63:0] data_i;                        //ta data out se kataxwrhth

wire led0 = data_o[3];                    //ana8esh twn bits
```

```
wire led1 = data_o[2];                    //3, 2, 1 kai 0
wire led2 = data_o[1];                     //tou shmatos data_o
wire led3 = data_o[0];                   //sta led0 ,led1 , led2 kai
led3 antistoixa

wire decrypt = 0;
reg [3:0] roundSel;
wire [55:0] key = data_i[55:0] &&  64'h6666666666666666; //ana8esh
timhs sto kleidi
                                                            //sta 56
                                          // prwta bits tou data in
                                                            //kollame
                                  // kai 8 bits 64'h6666666666666666
//
always @(negedge clk or negedge reset)
begin
   if(!reset)                            //gia reset 0
   begin
      data_i = 64'h0;                    //arxikopoihsh tou data in
                                           // sto 0
   end
   else                                 //allws
   begin
     data_i = data_i + 1;               //aujhsh data in kata ena
   end
end
//
                                          //
                                           //
always @(negedge clk or negedge reset)
begin
   if(!reset)                         //gia reset 0
   begin
      roundSel = 4'b0;                //arxikopoihsh tou roundSel sto
                                     // 0
   end
   else                               //allws
   begin
     roundSel = roundSel + 1;           //aujhsh roundSel kata ena
   end
end

des des1(data_o, data_i, key, decrypt, roundSel, clk );

endmodule
```

E_md5

```
module top1_md5(clk, reset,data_o,led0,led1,led2,led3);

input clk;                  //duo eisodous clock
input reset;                //kai reset
output data_o;              // pente shmata ejodou
output led0;                 //to data out
output led1;                //kai tessera leds
output led2;               //pou xrhsimopoioyme ws
output led3;              //shmata elegxou
```

```
reg [3:0] state;            //kataxwrhsh se 4bit reg
reg [3:0] next_state;      //ta state kai next_state
wire [127:0] data_o;       //kataxwrhsh 128 bit data out se wire
reg [127:0] data_i;         //kataxwrhsh 128 bit data in se register
wire newtext_i;
reg load_data;            //kataxwrhths fortwshs data
reg data_r;               //kataxwrhths data ready
wire ready_data;

wire led0 = data_o[3];       //ana8esh bits tou shmatos data out
wire led1 = data_o[2];        //sta 4 shmata elegxou mas
wire led2 = data_o[1];        //led0 , led1 , led2 kai led3
wire led3 = data_o[0];

always @(negedge clk or negedge reset)
begin

   if(!reset)                     //gia reset 0
   begin
      state = 4'b1111;            //pame sto arxiko state
      data_r = 0;                 //ta data den einai ready , 8etoume
                                    // data_r 0
   end
   else                           //alliws gia reset 1
   begin
     state = next_state;           //proxwra sto epomeno state
     if(ready_data) data_r = 1;    //an ta data einai ready , 8etoume
                                    //data_r 1
     else data_r = 0;              //alliws to afhnoume 0
   end
end


always @(state or data_r)
begin
     case(state)
        4'b1111:begin                   //arxiko state ,
arxikopoihseis
                  load_data = 0;         //na fortw8oun data??oxi
                  data_i = 128'h0;        //data in einai 0
                  next_state = 4'b1110;   //phgaine sto epomeno state
                end
        4'b1110:begin
                  load_data = 0;         //na fortw8oun data??oxi
                  data_i = 128'h0;         //data in einai 0
                  next_state = 4'b0000;    //phgaine sto epomeno
                                          // state
                end
        4'b0000:begin
                  load_data = 1;          //na fortw8oun data??nai
data_i = 128'h00008061000000000000000000000000;  //kane data in
                       // 128'h00008061000000000000000000000000
                  next_state = 4'b0001;    //phgaine sto epomeno
                                          // state
                end
        4'b0001:begin
                  load_data = 1;            //na fortw8oun data??nai
data_i = 128'h00008061000000000000000000000000;    //krata data in
                             //128'h00008061000000000000000000000000
                  next_state = 4'b0010;         //phgaine sto epomeno
```

```
                                                  // state
                end
        4'b0010:begin
                  load_data = 0;              //na fortw8oun
                                                  // data??oxi
data_i = 128'h00008061000000000000000000000000;    //krata data in ws
                                                  // exoun
                  next_state = 4'b0011;      //phgaine sto epomeno
                                                  // state
                end
        4'b0011:begin
                  load_data = 0;              //na fortw8oun
                                                  // data??oxi
                  data_i= 128'h800000000;       //data in na einai
                                                  // 128'h800000000
                  next_state = 4'b0100;        //phgaine sto epomeno
                                                  // state
                end
        4'b0100:begin
                  load_data = 1;                 //na fortw8oun
                                                 // data??nai
                  data_i= 128'h800000000;        //data in na einai
                                                 // 128'h800000000
                  next_state = 4'b0101;         //phgaine sto epomeno
                                                 // state
                end
        4'b0101:begin
                  load_data = 1;                 //na fortw8oun
                                                 // data??nai
                  data_i= 128'h800000000;         //data in einai
                                                 // 128'h800000000
                  next_state = 4'b0110;           //phgaine sto
                                                 //epomeno state
                end
        4'b0110:begin
                  load_data = 0;                 //na fortw8oun
                                                 //data??oxi
                  data_i= 128'h800000000;       //data in einai
                                                 //128'h800000000
                  next_state = 4'b0111;       //phgaine sto epomeno
                                                  // state
                end
        4'b0111:begin
                  load_data = 0;             //na fortw8oun data??oxi
data_i=128'h61616161616161616161616161616160;    //kane to data in na
                  // einai 128'h61616161616161616161616161616160
                  next_state = 4'b1000;        //phgaine sto epomeno
                                                  //state
                end
        4'b1000:begin
                  load_data = 1;                 //na fortw8oun
                                                 //data??nai
data_i=128'h61616161616161616161616161616160;    //kane to data in na
                        //einai 128'h61616161616161616161616161616160
                  next_state = 4'b1001;        //phgaine sto epomeno
                                                  //state
                end
        4'b1001:begin
                  load_data = 1;               //na fortw8oun
                                                //     data??nai
data_i=128'h61616161616161616161616161616160;    //kane to data in na
```

```
                        //einai 128'h61616161616161616161616161616160
                  next_state = 4'b1010;      //phgaine sto epomeno
                                           // state
                end
        4'b1010:begin
                  load_data = 0;                   //na fortw8oun
                                                 //data??oxi
                  data_i = 128'hF;                 //kane data in iso
                                                 // me 128'hF
                  if(data_r) next_state = 4'b1111; //an to data_r
                              // einai 1 phgaine sto arxiko state
                  else next_state = 4'b1010;      //alliws kane pali
                                                 // auto to state
                end
        default:begin                             //when others
                  load_data = 0;                 //mh fortwneis data
                  next_state = 4'b1111;           //phgaine sto arxiko
                                                 //state
                  data_i = 128'h0;                //arxikopoihsh data
                                                 // in sto 0
                end
    endcase
end

md5 m1(clk,reset,load_data,ready_data,newtext_i,data_i,data_o);

endmodule
```