Τ.Ε.Ι. ΚΡΗΤΗΣ ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΠΟΛΥΜΕΣΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη Συστήματος Αποδοτικής Κρυπτογράφησης για Ενσωματωμένα Συστήματα

ΑΥΓΕΝΑΚΗΣ ΙΩΑΝΝΗΣ Α.Μ. 623

Επιβλέπων Καθηγητής:

Κορνάρος Γεώργιος

22/2/2013

Ευχαριστίες

Θέλω να εκφράσω τις ευχαριστίες μου στον επιβλέποντα καθηγητή κύριο Κορνάρο Γεώργιο για την πολύτιμη στήριξή του καθ' όλη τη διάρκεια εκπόνησης της πτυχιακής , όπως επίσης και στον Χριστοφοράκη Γιάννη που χωρίς τη βοήθεια του θα ήταν το έργο μου πολύ δυσκολότερο.

Abstract

Purpose of this thesis is the creation of a model of safe communication between two microcontrollers to be used in embedded systems. Specific encryption algorithm will be evaluated by the timing of the procedures of encryption , transmission and decryption. For that reason it was developed a protocol that gives the opportunity of choosing different levels of safety and speed of communication.

The measurements will help us to make important conclusions about the performance of the algorithm used in this model in order to be able to be compared with times measured in similar work in other models using different encryption algorithms or different microcontrollers.

Σύνοψη

Αντικείμενο αυτής της πτυχιακής είναι η δοκιμή συγκεκριμένου αλγορίθμου κρυπτογράφησης σε μοντέλο επικοινωνίας ενσωματωμένων συστημάτων, με σκοπό την χρονομέτρηση των διαδικασιών κρυπτογράφησης, αποστολής και αποκρυπτογράφησης.

Ένα μέρος της εργασίας είναι η ανάπτυξη του τρόπου επικοινωνίας των μικροελεγκτών. Δηλαδή η δημιουργία ενός πρωτοκόλλου το οποίο να επιτρέπει την επικοινωνία με κρυπτογραφημένα αλλά και μη κρυπτογραφημένα μηνύματα.

Με τους χρόνους που μας δίνουν οι μετρήσεις μπορούμε να βγάλουμε σπουδαία συμπεράσματα για την απόδοση του αλγορίθμου που χρησιμοποιήθηκε στο συγκεκριμένο μοντέλο αλλά και δοθεί η δυνατότητα σύγκρισης με χρόνους που έχουν μετρηθεί σε αντίστοιχες εργασίες σε άλλα μοντέλα με χρήση διαφορετικών αλγορίθμων κρυπτογράφησης ή διαφορετικών μικροελεγκτών.

Πίνακας Περιεχομένων

Κατάλογος εικόνων

Κατάλογος πινάκων

Κατάλογος γραφημάτων

1. ΕΙΣΑΓΩΓΗ

Οι απαιτήσεις που υπάρχουν για ασφαλή και ταχεία μεταφορά πληροφορίας αυξάνονται ολοένα και περισσότερο. Καθότι νέες τεχνολογίες εφευρίσκονται όπως είναι τα radio-frequency identification (r.f.i.d) και οι έξυπνες κάρτες αλλά και η ανάπτυξη νέων ενσωματωμένων συστημάτων που εξυπηρετούν σκοπούς στους οποίους χρειάζεται ασφαλής επικοινωνία, είναι φυσικό να γίνεται παράλληλη έρευνα νέων πιο αποδοτικών αλγορίθμων κρυπτογράφησης που να λειτουργούν ιδανικά για κάθε μια από τις νέες αυτές εφαρμογές.

Κάθε είδος εφαρμογής στην οποία είναι απαραίτητη η κρυπτογράφηση, έχει διαφορετικές δυνατότητες. Ο λόγος είναι οι διαθέσιμοι πόροι του συστήματος που χρησιμοποιείται. Σε κάποια συστήματα κυρίως λόγο της φορητότητας τους ,έχουν περιορισμένο μέγεθος, γεγονός που κοστίζει στο σύστημα σε υπολογιστική ισχύ, μνήμη και ενέργεια. Όμως δεν έχουν όλες οι εφαρμογές τις ίδιες απαιτήσεις σε επίπεδο ασφάλειας. Για παράδειγμα στο σύστημα ηλεκτρονικών συναλλαγών στις τράπεζες ,πρέπει το επίπεδο ασφάλειας να είναι υψηλό αλλά υπάρχουν αρκετοί διαθέσιμοι πόροι για να χρησιμοποιηθεί ένας ‘βαρύς’ αλγόριθμος κρυπτογράφησης. Από την άλλη στα r.f.i.d οι διαθέσιμοι πόροι είναι ελάχιστοι . Οπότε πρέπει να χρησιμοποιηθεί ένας αλγόριθμος ελαφριάς κρυπτογράφησης που να παρέχει την καλύτερη δυνατή ασφάλεια.

Ένας από τους αλγόριθμους που δημοσιεύτηκαν τα τελευταία χρόνια είναι ο PRESENT. Ο αλγόριθμος αυτός είναι ένας αλγόριθμος ελαφριάς κρυπτογράφησης και χρησιμοποιείται σε αυτήν την εργασία για να χρονομετρηθεί η απόδοσή του σε περίπτωση χρήσης του σε ενσωματωμένα συστήματα.

Για να επιτευχθούν οι μετρήσεις αυτές προσπαθήσαμε να δημιουργήσουμε ένα μοντέλο σύνδεσης δύο μικροελεγκτών .Οι μικροελεγκτές αυτοί θα μπορούν να επικοινωνούν μεταξύ τους μέσω ενός πρωτοκόλλου επικοινωνίας. Το πρωτόκολλο αυτό θα επιτρέπει την επιλογή μεταξύ διαφορετικών επιπέδων ταχύτητας και ασφάλειας, στην ανταλλαγή δεδομένων μεταξύ των μικροελεγκτών.

Οι μικροελεγκτές που χρησιμοποιήσαμε είναι οι at91r40008 της εταιρίας Atmel. Η γλώσσα προγραμματισμού που επιλέξαμε για την υλοποίηση είναι η c.

1.1 Σκοπός και Στόχοι Εργασίας

Η εργασία αυτή έγινε με σκοπό την δοκιμή του συνδυασμού λειτουργίας του μικροελεγκτή at91r40008 της Atmel με τον αλγόριθμο ελαφριάς κρυπτογράφησης present , έτσι ώστε να αξιολογηθεί η περίπτωση χρήσης τους σε ενσωματωμένα συστήματα. Υπάρχουν πολλές εφαρμογές στις οποίες θα μπορούσε να χρησιμοποιηθεί το συγκεκριμένο μοντέλο όπως σε ιατρικές εφαρμογές. Ένα παράδειγμα χρήσης είναι σε medical sensor networks, όπου η ασύρματη μεταφορά των μετρήσεων των βιο-

αισθητήρων του ασθενή , που είναι προσωπικά δεδομένα, μεταφέρονται σε πραγματικό χρόνο στο pda του γιατρού .

Οι στόχοι της εργασίας είναι δύο:
- Ο πρώτος στόχος είναι η δημιουργία πρωτοκόλλου με τη βοήθεια του οποίου θα είναι δυνατή η επικοινωνία των δύο μικροελεγκτών είτε μέσω κρυπτογραφημένων μηνυμάτων είτε μέσω απλών μηνυμάτων.
- Ο δεύτερος στόχος είναι η καταγραφή των μετρήσεων των χρόνων που χρειάζονται για την αποστολή των μηνυμάτων καθώς και για την κρυπτογράφηση και την αποκρυπτογράφηση στους συγκεκριμένους μικροελεγκτές.

1.2 Δομή Εργασίας

- Στο κεφάλαιο 2 γίνεται μία εισαγωγή στους μικροεπεξεργαστές. Από τι αποτελούνται , ποια είναι τα βασικά χαρακτηριστικά τους και τι είναι οι μικροελεγκτές.

- Στο κεφάλαιο 3 εξηγείται τι είναι η κρυπτογραφία και η κρυπτανάλυση, αναφέρονται οι κατηγορίες των κρυπτοσυστημάτων , αλλά και κάποιοι γνωστοί αλγόριθμοι και γίνεται περιγραφή της έννοιας των υβριδικών συστημάτων . Ακόμη περιγράφεται η έννοια των αλγορίθμων ελαφριάς κρυπτογράφησης και περιγράφονται μερικές εφαρμογές που τους χρησιμοποιούν.

- Στο κεφάλαιο 4 γίνεται αναφορά σε εργασίες που έχουν γίνει σχετικά με το θέμα που εξετάζουμε.

- Στο κεφάλαιο 5 γίνεται η περιγραφή της υλοποίησης σε ότι αφορά το hardware και το software. Γίνεται περιγραφή της συνδεσμολογίας αλλά και του αλγορίθμου κρυπτογράφησης present.

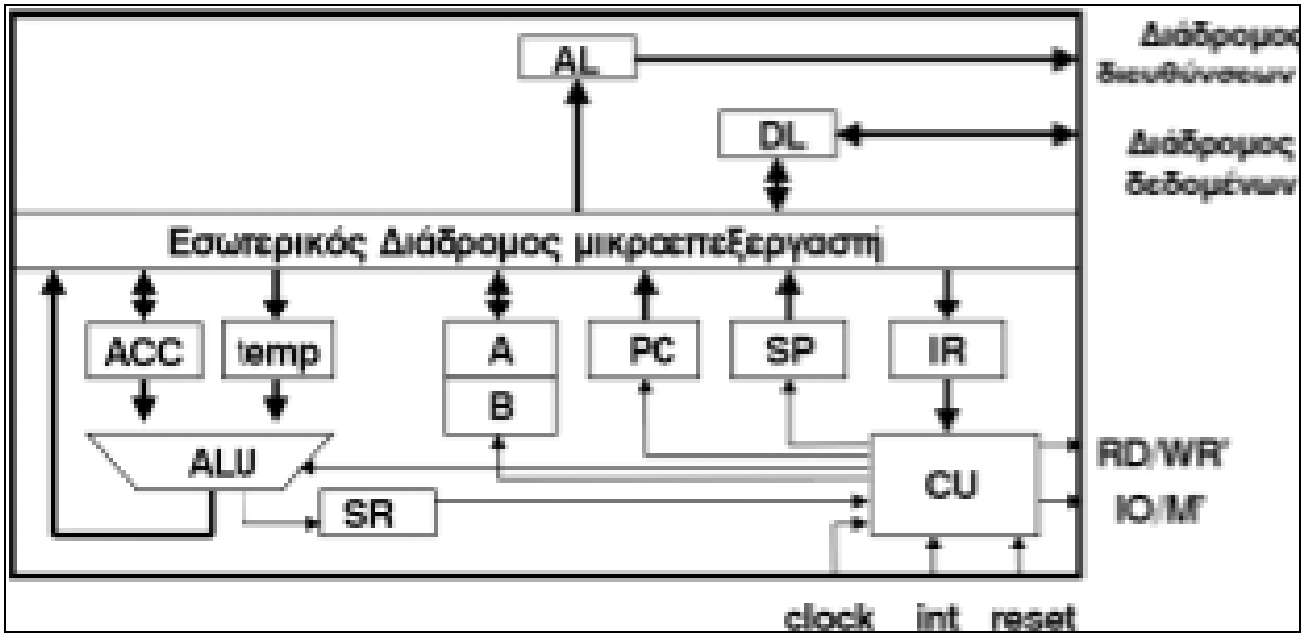
- Στο κεφάλαιο 6 δίνονται οι μετρήσεις.

- Στο κεφάλαιο 7 δίνονται τα συμπεράσματα της εργασίας.

2. Εισαγωγή στους μικροεπεξεργαστές

Μικρουπολογιστικό σύστημα ονομάζεται αυτό που χρησιμοποιεί για κεντρική μονάδα επεξεργασίας ένα μικροεπεξεργαστή. Μικροεπεξεργαστής είναι ένα ολοκληρωμένο κύκλωμα γενικού σκοπού το οποίο μπορεί να προγραμματιστεί.
Ένας Μικροεπεξεργαστής αποτελείται από τα εξής τμήματα:

1. Την αριθμητική και λογική μονάδα.
2. Τη μονάδα ελέγχου.
3. Τους καταχωρητές.



Εικόνα 2.1 Σχέδιο μικροεπεξεργαστή

- ALU: αριθμητική και λογική μονάδα.
- CU: μονάδα ελέγχου.
- ACC: συσσωρευτής
- temp: προσωρινός καταχωρητής
- A,B: καταχωρητές γενικού σκοπού
- IR: καταχωρητής εντολών

- SR: καταχωρητής κατάστασης
- PC: μετρητής προγράμματος
- SP: δείκτης στοίβας
- AL: απομονωτής διευθύνσεων
- DL: απομονωτής δεδομένων

2.1 χαρακτηριστικά μικροεπεξεργαστών

Τα πιο σημαντικά χαρακτηριστικά των μικροεπεξεργαστών είναι τρία.

1. Η συχνότητα λειτουργίας
2. Το μήκος λέξης.
3. Το ρεπερτόριο εντολών

2.1.1 Συχνότητα λειτουργίας του μικροεπεξεργαστή.

Για τον συγχρονισμό των στοιχειωδών λειτουργιών ενός μικροεπεξεργαστή είναι απαραίτητη η παρουσία κάποιου ρολογιού. Σε κάθε χτύπο του ρολογιού εκτελείται μια στοιχειώδης λειτουργία. Το ρολόι δημιουργεί έναν τετραγωνικό παλμό που εναλλάσσεται μεταξύ της στάθμης των 0 και των 5 volt .Επειδή ο κάθε μικροεπεξεργαστής είναι σχεδιασμένος να λειτουργεί μέχρι κάποια μέγιστη συχνότητα ,το ρολόι ρυθμίζεται έτσι ώστε να λειτουργεί κοντά σε αυτή τη συχνότητα ,που ονομάζεται συχνότητα λειτουργίας του μικροεπεξεργαστή. Αν για παράδειγμα η συχνότητα αυτή είναι 100 MHZ σημαίνει ότι εκτελούνται 100.000.000 κύκλοι μηχανής το δευτερόλεπτο.

2.1.2 Μήκος λέξης του μικροεπεξεργαστή.

Ο αριθμός των bits που χωράει ο κάθε καταχωρητής που είναι συνήθως ίδιος με αυτόν των υπόλοιπων εσωτερικών μονάδων του μικροεπεξεργαστή όπως η ALU(αριθμητική και λογική μονάδα) ονομάζεται μήκος λέξης. Είναι σημαντικό χαρακτηριστικό γιατί όσο μεγαλύτερο είναι το μήκος λέξης του μικροεπεξεργαστή τόσο περισσότερα δεδομένα θα μπορεί να επεξεργαστεί σε κάθε κύκλο μηχανής. Υπάρχουν λοιπόν διαφορετικοί τύποι μικροεπεξεργαστών ανάλογα με το μήκος λέξης. 8 bit (οχτάμπιτοι) ,16 bit (δεκαεξάμπιτοι), 32 bit (τριανταδυάμπιτοι).

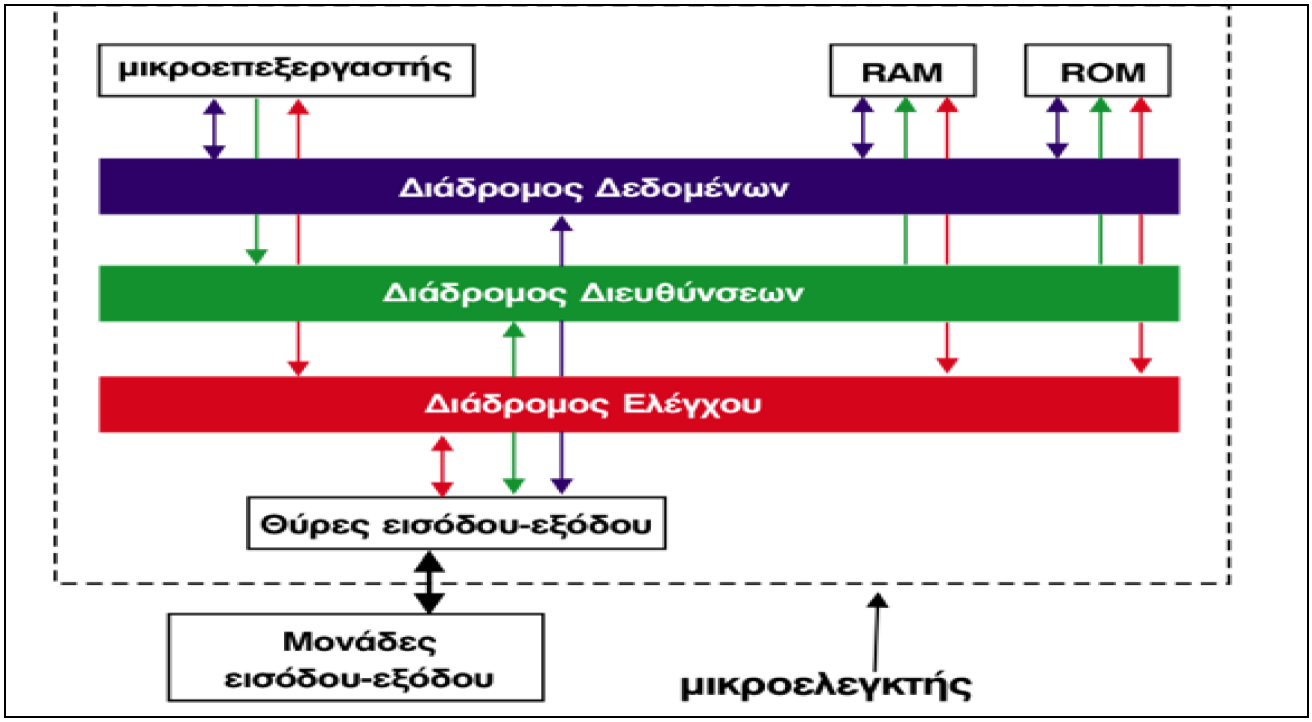
2.1.3 Ρεπερτόριο εντολών του μικροεπεξεργαστή

Με τον όρο ρεπερτόριο εντολών ενός μικροεπεξεργαστή αναφερόμαστε στις εντολές που μπορεί να εκτελέσει. Με βάση το μέγεθος του ρεπερτορίου εντολών οι μικροεπεξεργαστές χωρίζονται σε δύο κατηγορίες: τους μικροεπεξεργαστές διευρυμένου ρεπερτορίου εντολών (CISC) και τους

μικροεπεξεργαστές μειωμένου ρεπερτορίου εντολών (RISC). Στην πρώτη κατηγορία ο μικροεπεξεργαστής μπορεί να εκτελέσει πολύπλοκες διαδικασίες με λιγότερες εντολές. Όμως όσο μεγαλύτερο ρεπερτόριο εντολών έχει ένας μικροεπεξεργαστής τόσο πιο πολύπλοκη είναι η σχεδίασή του με αποτέλεσμα να υπάρχει μεγαλύτερη καθυστέρηση στην εκτέλεση κάθε εντολής. Κι ακόμη ενώ κάποιες από τις εντολές εκτελούνται συχνά, οι υπόλοιπες εκτελούνται πολύ σπάνια. Για τους λόγους αυτούς δημιουργήθηκε η ανάγκη σχεδίασης μικροεπεξεργαστών με όσο το δυνατόν πιο μειωμένο ρεπερτόριο εντολών που όμως η κάθε μία εντολή εκτελείται όσο το δυνατόν πιο γρήγορα.(RISC)

2.2 Μικροελεγκτές.

Μικροελεγκτής ονομάζεται ένα ολοκληρωμένο κύκλωμα το οποίο περιλαμβάνει μικροεπεξεργαστή, μνήμη και περιφερειακές μονάδες εισόδου –εξόδου. Είναι ένα πλήρες υπολογιστικό σύστημα βελτιστοποιημένο για τον έλεγχο hardware. Οι μικροελεγκτές είναι κατάλληλοι για εφαρμογές με αυξημένη ανάγκη για χρήση περιφερειακών συσκευών. Ένα κριτήριο επιλογής μικροελεγκτή είναι τα περιφερειακά που διαθέτει καθώς και οι δυνατότητές τούς.



Εικόνα 2.2 Σχέση μικροεπεξεργαστή και μικροελεγκτή

2.2.1 Σύγκριση μικροεπεξεργαστή – μικροελεγκτή**.**

Στους μικροεπεξεργαστές δίνεται έμφαση στην υπολογιστική ισχύ. Η λειτουργικότητα του τελικού συστήματος δεν είναι εξειδικευμένη ,αλλά έχει την ευελιξία να αναπτύξει πολλές διαφορετικές εφαρμογές. Αντίθετα οι μικροελεγκτές έχουν χαμηλότερη υπολογιστική ισχύ και μικρότερη ευελιξία. Έχουν όμως χαμηλό κόστος και εξειδίκευση. Μερικά από τα πλεονεκτήματα των μικροελεγκτών είναι τα εξής:

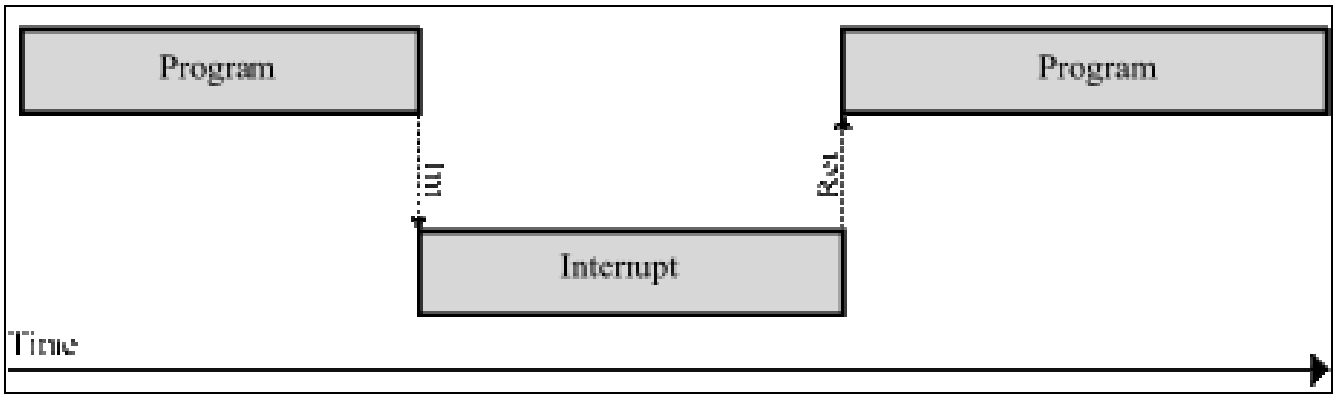
- λόγο της ενσωμάτωσης σύνθετων περιφερειακών υποσυστημάτων όπως μνήμες και θύρες επικοινωνίας έχει αυτονομία.
- Η ενσωμάτωση των περιφερειακών βοηθά στην υλοποίηση των εφαρμογών λόγο των απλούστερων διασυνδέσεων.
- Έχει χαμηλό κόστος.
- Έχει μεγαλύτερη αξιοπιστία.
- Έχει μικρό μέγεθος το συνολικό υπολογιστικό σύστημα.

Οι πιο γνωστές περιφερειακές συσκευές που χρειάζονται σε ένα μικροελεγκτή είναι:

- Μνήμη(RAM,ROM)
- Θύρες εισόδου – εξόδου, σειριακής και παράλληλης επικοινωνίας
- Μονάδα αναγνώρισης διακοπών
- Μετατροπείς αναλογικού σήματος σε ψηφιακό και το αντίστροφο
- Timers

2.2.2 interrupts

Τα interrupts είναι σήματα στον επεξεργαστή που προκαλούνται είτε από το hardware είτε από το software. Όταν φτάνει ένα interrupt στον επεξεργαστή αυτός καταλαβαίνει ότι συμβαίνει κάτι σημαντικό που χρειάζεται να το χειριστεί άμεσα. Αυτό έχει σαν αποτέλεσμα ο επεξεργαστής να αποθηκεύσει την κατάστασή του και να μεταβεί σε άλλο μέρος του κώδικα το οποίο ονομάζεται interrupt handler. Μόλις τελειώσει ,επανακτά την κατάστασή του πριν τη διακοπή και συνεχίζει κανονικά από ‘ κει που είχε σταματήσει.



Εικόνα 2.3 Σχέδιο ροής προγράμματος με διακοπή

Τα interrupts που προκαλούνται από το hardware είναι προειδοποιητικά σήματα που στέλνονται στον επεξεργαστή από εξωτερικές συσκευές ή από περιφερειακά. Για παράδειγμα όταν πατάμε ένα κουμπί στο πληκτρολόγιο στέλνεται ένα interrupt στον επεξεργαστή για να διαβάσει το πάτημα του πλήκτρου.

Τα interrupts που προκαλούνται από το software προκαλούνται είτε από εξαίρεση μέσα στον ίδιο τον επεξεργαστή, είτε από κάποια εντολή μέσα από το σετ εντολών ,που αν εκτελεστεί προκαλεί interrupt. Για παράδειγμα όταν δοθεί στην αριθμητική και λογική μονάδα εντολή να διαιρέσει κάποιον αριθμό με το 0, θα προκληθεί interrupt που θα κάνει τον υπολογιστή να εγκαταλείψει την πράξη ή να βγάλει μήνυμα λάθους.

3. Εισαγωγή στην κρυπτογραφία

Κρυπτογραφία είναι ένας κλάδος της επιστήμης της κρυπτολογίας που ασχολείται με τη μελέτη της ασφαλούς επικοινωνίας. Στόχος της είναι να παρέχει μηχανισμούς οι οποίοι θα κάνουν εφικτή την επικοινωνία ανάμεσα σε δύο ή περισσότερα άτομα ,ενώ κανείς άλλος δεν θα μπορεί να διαβάσει την πληροφορία.

Κρυπτογράφηση ονομάζεται η διαδικασία μετασχηματισμού ενός μηνύματος σε μια ακατανόητη μορφή με τη χρήση κάποιου κρυπτογραφικού αλγορίθμου ούτως ώστε να μην μπορεί να διαβαστεί από κανέναν εκτός του νόμιμου παραλήπτη. Αποκρυπτογράφηση είναι η αντίστροφη διαδικασία όπου από το κρυπτογραφημένο κείμενο παράγεται το αρχικό μήνυμα. Η μέθοδος με την οποία μετασχηματίζονται τα δεδομένα για να γίνει η κρυπτογράφηση ονομάζεται κρυπτογραφικός αλγόριθμος και είναι μια πολύπλοκη μαθηματική συνάρτηση.

Ένα από τα πιο γνωστά συστήματα κρυπτογράφησης στην ιστορία είναι αυτό που χρησιμοποιούσε ο Ιούλιος καίσαρας , ο οποίος αντικαθιστούσε το κάθε γράμμα του κειμένου που ήθελε να κρυπτογραφήσει με το γράμμα που βρισκόταν 3 θέσεις μετά στο λατινικό αλφάβητο. Γενικά σε όλες τις παλαιότερες μορφές κρυπτογράφησης η επεξεργασία γινόταν πάνω στη γλωσσική δομή ,ενώ στις νεότερες μορφές γίνεται σε αριθμούς. Γι' αυτό σημαντικό ρόλο έχουν τα μαθηματικά.

3.1 κατηγορίες κρυπτοσυστημάτων

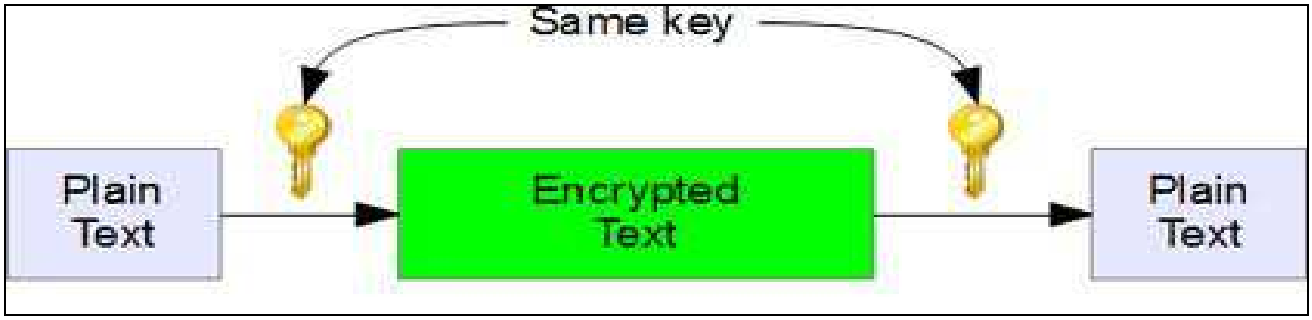
Τα κρυπτοσυστήματα χωρίζονται σε δυο κατηγορίες, τα συμμετρικά και τα ασύμμετρα.

Στα συμμετρικά κρυπτοσυστήματα για την κρυπτογράφηση και την αποκρυπτογράφηση χρησιμοποιείται ένα κοινό κλειδί. Η ασφάλεια των αλγορίθμων αυτών βασίζεται στην μυστικότητα του κλειδιού .Για το λόγο αυτό η ανταλλαγή του κλειδιού πρέπει να γίνει είτε μέσα από ένα ασφαλές κανάλι επικοινωνίας είτε μέσα από τη φυσική παρουσία των προσώπων.

Τα συμμετρικά κρυπτοσυστήματα χωρίζονται σε δύο κατηγορίες.

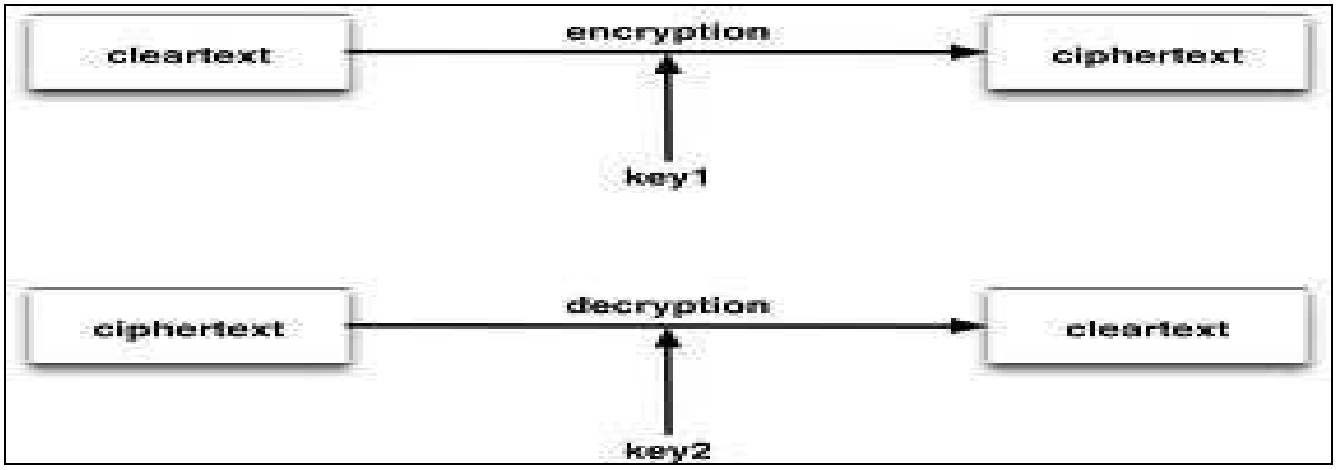
- Δέσμης(block ciphers). Χωρίζουν το μήνυμα σε κομμάτια και κρυπτογραφούν χωριστά κάθε κομμάτι.

- Ροής(stream ciphers).Κρυπτογραφούν μια ροή μηνύματος χωρίς να τη χωρίζουν σε κομμάτια.



Εικόνα 3.1 Συμμετρική κρυπτογράφηση

Στα ασύμμετρα κρυπτοσυστήματα υπάρχουν δύο είδη κλειδιών. Ένα ιδιωτικό και ένα δημόσιο. Το δημόσιο είναι διαθέσιμο σε όλους ενώ το ιδιωτικό είναι μυστικό. Ότι κρυπτογραφεί το δημόσιο κλειδί μπορεί να το αποκρυπτογραφήσει μόνο το ιδιωτικό. Το πλεονέκτημα αυτού του τρόπου έναντι του συμμετρικού είναι ότι δε χρειάζεται να αποσταλεί κανένα κλειδί με ασφάλεια.



Εικόνα 3.2 Ασύμμετρη κρυπτογράφηση

Μερικές εφαρμογές που είναι απαραίτητη η κρυπτογράφηση είναι:

1. ασφάλεια συναλλαγών σε τράπεζες(ATM)
2. σταθερή και κινητή τηλεφωνία
3. στρατιωτικά δίκτυα
4. ηλεκτρονικές οικονομικές συναλλαγές
5. έξυπνες κάρτες
6. συστήματα συναγερμών
7. ηλεκτρονικό γραμματοκιβώτιο
8. R.F.I.D.

3.1.2 Κρυπτανάλυση

Κρυπτανάλυση ονομάζεται η επιστήμη της ανάκτησης του αρχικού μηνύματος όταν είναι γνωστό το κρυπτογραφημένο μήνυμα , χωρίς πρόσβαση στο κλειδί. Μια επιτυχείς κρυπτανάλυση θεωρείται όταν μπορέσει να ανακτηθεί το αρχικό μήνυμα ή το κλειδί. Μία προσπάθεια κρυπτανάλυσης ονομάζεται επίθεση. Αυτός που κάνει την επίθεση μπορεί να γνωρίζει κάποια μηνύματα κρυπτογραφημένα ή και το αρχικό μήνυμα πριν κρυπτογραφηθεί. Επίθεση ωμής βίας ονομάζεται η επίθεση κατά την οποία όλα τα πιθανά κλειδιά δοκιμάζονται ένα ένα μέχρι να βρεθεί το σωστό.

3.1.3 Γνωστοί αλγόριθμοι

- AES (advanced encryption standard) είναι ο πιο γνωστός αλγόριθμος κρυπτογρφάφησης. Ο AES είναι ένας συμμετρικός αλγόριθμος που χρησιμοποιείται σαν διεθνές πρότυπο .Το ίδιο κλειδί χρησιμοποιείται για την κρυπτογράφηση αλλά και για την αποκρυπτογράφηση των μηνυμάτων.
- RSA (από τα αρχικά των ονομάτων των δημιουργών του- Rivest, Shamir, Adleman.)Είναι ο πιο γνωστός αλγόριθμος δημοσίου κλειδιού. Μπορεί να χρησιμοποιηθεί και για κρυπτογράφηση αλλά και για ψηφιακή υπογραφή.
- DSA (digital signature algorithm) Ο DSA είναι αλγόριθμος δημοσίου κλειδιού όμως δεν χρησιμοποιείται για κρυπτογράφηση, αλλά μόνο για ψηφιακή υπογραφή.

3.1.4 Υβριδικά συστήματα

Επειδή οι αλγόριθμοι δημοσίου κλειδιού είναι αργοί δεν χρησιμοποιούνται συνήθως για να κρυπτογραφούν μηνύματα, αλλά για να κρυπτογραφούν κλειδιά. Έτσι αν συνδυάσουμε έναν αλγόριθμο δημοσίου κλειδιού και έναν συμμετρικό αλγόριθμο έχουμε ένα υβριδικό κρυπτοσύστημα. Με τον αλγόριθμο δημοσίου κλειδιού κρυπτογραφούμε το κλειδί για να μας παρέχει ασφαλή μεταφορά του κλειδιού του συμμετρικού αλγορίθμου, που θα χρησιμοποιηθεί για την κρυπτογράφηση των μηνυμάτων.

3.2 Ελαφριά κρυπτογράφηση

Ένας αλγόριθμος κρυπτογράφησης όπως ο AES είναι ιδανικός σχεδόν για όλες τις εφαρμογές αφού καταφέρνει να παρέχει μεγάλη προστασία στα προς αποστολή δεδομένα. Για κάποιες εφαρμογές όμως παρέχει μεγαλύτερη προστασία απ' ότι χρειάζεται με μεγάλο κόστος σε τιμή αλλά και σε αυτονομία λόγο του ότι χρειάζεται περισσότερη ενέργεια για να λειτουργήσει . Γι' αυτές τις εφαρμογές υπάρχουν οι αλγόριθμοι ελαφριάς κρυπτογράφησης (lightweight encryption algorithms).

3.2.1 Αλγόριθμοι ελαφριάς κρυπτογράφησης

Γενικός σκοπός στη σχεδίαση των αλγορίθμων αυτών είναι να πετύχουν μία καλή αναλογία τριών παραμέτρων , την ασφάλεια ,το κόστος και την απόδοση. Στις εφαρμογές αυτές που είναι απαραίτητη η χρήση αλγορίθμων ελαφριάς κρυπτογράφησης, μικρότερη προστασία είναι επαρκείς σε σχέση με αυτήν του AES. Η προστασία αυτή μπορεί να επιτευχθεί με κλειδιά μικρότερα από 128 bit γεγονός που κάνει το κόστος σημαντικά μικρότερο.

Μια στρατηγική σχεδιασμού ευρείας διαδρομής αποτελείται από τρία στρώματα.

- Μη γραμμικό στρώμα για σύγχυση
- Στρώμα ανάμειξης για διάχυση
- Στρώμα μεταφοράς για διασπορά

Κάποιοι αλγόριθμοι αποφεύγουν να χρησιμοποιήσουν το στρώμα ανάμειξης για να περιορίσουν το κόστος , αλλά και για να περιορίσουν τους γύρους κρυπτογράφησης .Ένα παράδειγμα τέτοιου αλγόριθμου είναι ο Present που είναι ένας αλγόριθμος αντικατάστασης – μετάθεσης (substitution-permutation).

Οι αλγόριθμοι αυτοί δεν είναι ιδανικοί για να χρησιμοποιηθούν για όλες τις εφαρμογές , ούτε σχεδιάστηκαν για να αντικαταστήσουν τους συμβατικούς αλγορίθμους καθότι περιορίζονται από λειτουργικούς περιορισμούς. Αυτό δε σημαίνει ότι η κρυπτογράφηση που παρέχουν είναι αδύναμη, απλά δεν προορίζεται για εφαρμογές στις οποίες αυτοί που θα προσπαθήσουν να τους σπάσουν είναι υπερβολικά δυνατοί.

3.2.2 Εφαρμογές με ελαφριά κρυπτογράφηση

Υπάρχουν πολλές εφαρμογές στις οποίες είναι απαραίτητη η χρήση αλγορίθμων ελαφριάς κρυπτογράφησης σε διαφορετικά πεδία όπως είναι η ιατρική περίθαλψη , τα στρατιωτικά συστήματα και η αυτοκινητοβιομηχανία .

Μία εφαρμογή είναι τα medical sensor network. Με τη χρήση τους υπάρχει η δυνατότητα άμεσης καταγραφής της κατάστασης της υγείας του ασθενή μέσω των body sensor nodes και ενημέρωσης των γιατρών σε πραγματικό χρόνο. Αυτό μπορεί να γίνει όταν ο ασθενής είναι στο νοσοκομείο μέσω ασύρματου δικτύου ,αλλά μπορεί να γίνει και όταν είναι μακριά απ' αυτό μέσω της σύνδεσης του κινητού τηλεφώνου του. Από την άλλη ο γιατρός μπορεί να ενημερωθεί την ίδια στιγμή μέσω του

δικού του κινητού τηλεφώνου ή μέσω φορητού pda. Όλα αυτά τα ιατρικά δεδομένα της υγείας του ασθενή είναι απόρρητα προσωπικά δεδομένα που πρέπει κάπως να διαφυλαχθούν από τρίτους. Οι φορητές συσκευές όμως που χρησιμοποιούνται και λόγο του μικρού μεγέθους τους , έχουν περιορισμένους πόρους όσο αφορά μνήμη και ενέργεια , γεγονός που καθιστά ιδανική τη χρήση ελαφριάς κρυπτογράφησης.

Σε πολλές στρατιωτικές εφαρμογές θα μπορούσε να χρησιμοποιηθεί ένας αλγόριθμος ελαφριάς κρυπτογράφησης. Για παράδειγμα στην πρώτη γραμμή της μάχης που χρησιμοποιούνται συσκευές που στέλνουν δεδομένα σχετικά με τη ακριβής θέση των εχθρών. Οι πληροφορίες αυτές είναι ευαίσθητες για περιορισμένο χρονικό διάστημα , όσο διαρκεί η συγκεκριμένη μάχη. Η ελαφριά κρυπτογράφηση στην συγκεκριμένη εφαρμογή είναι σημαντική γιατί παρέχει μεγαλύτερη αυτονομία στη συσκευή μέσω της εξοικονόμησης σε ενέργεια, ενώ η προστασία που παρέχει στα δεδομένα είναι αρκετή για το μικρό χρονικό διάστημα που είναι ευαίσθητα.

Στην αυτοκινητιστική βιομηχανία επίσης υπάρχουν πολλές εφαρμογές με χρήση ελαφριάς κρυπτογράφησης. Δυο από αυτές είναι το σύστημα αυτόματου κλειδώματος – ξεκλειδώματος χωρίς κλειδί και το σύστημα immobilizer. Και στις δύο αυτές εφαρμογές η συσκευή που χρησιμοποιείται αντί του παραδοσιακού κλειδιού για να στείλει καταρχήν το σήμα για να ξεκλειδώσει το αυτοκίνητο και κατόπιν το σήμα στο σύστημα του immobilizer για να μπορέσει να ξεκινήσει , πρέπει να είναι αρκετά μικρή , ώστε να είναι λειτουργική για τον χρήστη της. Αυτό έχει σαν αποτέλεσμα και οι πόροι της συσκευής να είναι περιορισμένοι , κάτι που κάνει απαραίτητη τη χρήση αλγορίθμου ελαφριάς κρυπτογράφησης.

4. Συναφείς εργασίες

Καθότι η προστασία της πληροφορίας αλλά και ο χρόνος και η ενέργεια που χρειάζεται για να επιτευχθεί είναι σημαντικοί τομείς προς εξερεύνηση , πολλές είναι οι σχετικές εργασίες που έχουν πραγματοποιηθεί . Παρακάτω παρατίθενται μερικές από αυτές.

4.1 microcontroller application in cryptography techniques

Η δημοσίευση με τίτλο “microcontroller application in cryptography techniques” **18** ασχολείται με νέες τεχνικές κρυπτογράφησης και αποκρυπτογράφησης. Η γλώσσα που χρησιμοποιείται για να εκτελεστούν αλλά και να αξιολογηθούν οι αλγόριθμοι είναι η C . Απ ‘αυτούς τους αλγορίθμους επιλέγεται ο αλγόριθμος Blowfish ως ο πιο εξελιγμένος. Η υλοποίηση γίνεται με τη χρήση ενός μικροελεγκτή avr της Atmel. Από τα αποτελέσματα της υλοποίησης βρίσκεται ότι η διαδικασία επέκτασης και μεταλλαγής χρειάζονται τον περισσότερο χρόνο. Γι’ αυτό το λόγο το λογισμικό είναι σχεδιασμένο να εκτελεί αυτές τις δύο διαδικασίες . Η συγκεκριμένη δημοσίευση σχεδίασε τον αρχικό αλγόριθμο κρυπτογράφησης BLOWFISH σε hardware εφαρμογή, και έναν προτεινόμενο Blowfish block cipher και για hardware και για software εφαρμογή. Και οι δύο είναι συμμετρικοί αλγόριθμοι δέσμης (block) με μέγεθος block 64 bit,ενώ το μέγεθος κλειδιού είναι μέχρι 448 bits για τον πρώτο και 128 bits για τον προτεινόμενο.

4.2 Optimizing AES Implementation for High-Speed Embedded Application

Στην πτυχιακή με τίτλο “ Optimizing AES Implementation for High-Speed Embedded Application ” **19** ο αλγόριθμος κρυπτογράφησης AES βελτιστοποιείται για ταχεία εκτέλεση σε μια ενσωματωμένη πλατφόρμα με επεξεργαστή ARM και συγκεκριμένα στο AT43DK380 Development Kit . Αρχικά συγκρίνει διαθέσιμες μεθόδους για να γίνει υλοποίηση για καλύτερη απόδοση ταχύτητας. Στη συνέχεια μελετά τα βασικά χαρακτηριστικά του επεξεργαστή ARM και τα αξιοποιεί στην εφαρμογή για την ενίσχυση της ταχύτητας εκτέλεσης του αλγορίθμου. Τέλος δίνονται μετρήσεις χρόνου κρυπτογράφησης και αποκρυπτογράφησης.

4.3 AES Implementation and Performance Evaluation on 8-bit Microcontrollers

Σε αυτή τη δημοσίευση **20** προτάθηκε μία λύση για αξιόπιστο δίκτυο αισθητήρων για να αναλύσει την αποτελεσματικότητα της επικοινωνίας μέσω της μέτρησης της απόδοσης του αλγορίθμου κρυπτογράφησης AES ανάλογα με το μέγεθος του plaintext και το κόστος λειτουργίας σύμφωνα με την

κλίμακα του δικτύου. Περιγράφεται ο αλγόριθμος κρυπτογράφησης AES στη συμμετρική κρυπτογράφηση και μετράται η απόδοση κρυπτογράφησης και αποκρυπτογράφησης σε έναν 8 μπιτο μικροελεγκτή. Έπειτα αναλύεται η αποτελεσματικότητα της επικοινωνίας μέσα από τη συνολική καθυστέρηση ανά hop στο δίκτυο αισθητήρων . Ο μικροελεγκτής που χρησιμοποιείται για την ανάλυση απόδοσης του αλγορίθμου κρυπτογράφησης AES είναι ο ATmega644p.

4.4 Investigating and Analyzing the Light-weight ciphers for Wireless Sensor Networks

Στη Δημοσίευση αυτή **21** γίνονται προσομοιώσεις πειράματα και αναλύσεις που δείχνουν οτι η βελτιστοποιημένη εφαρμογή τους ,του διορθωμένου αλγορίθμου tiny encryption (xxtea) με κλειδί μεγέθους 128 bit, μπορεί να είναι μία καλή εναλλακτική λύση για skipjack, για περιβάλλοντα με πριορισμένους πόρους όπως είναι αυτά των Wireless Sensor Networks (WSNs).

4.5 Encryption overhead in embedded systems and sensor network nodes :modeling and analysis.

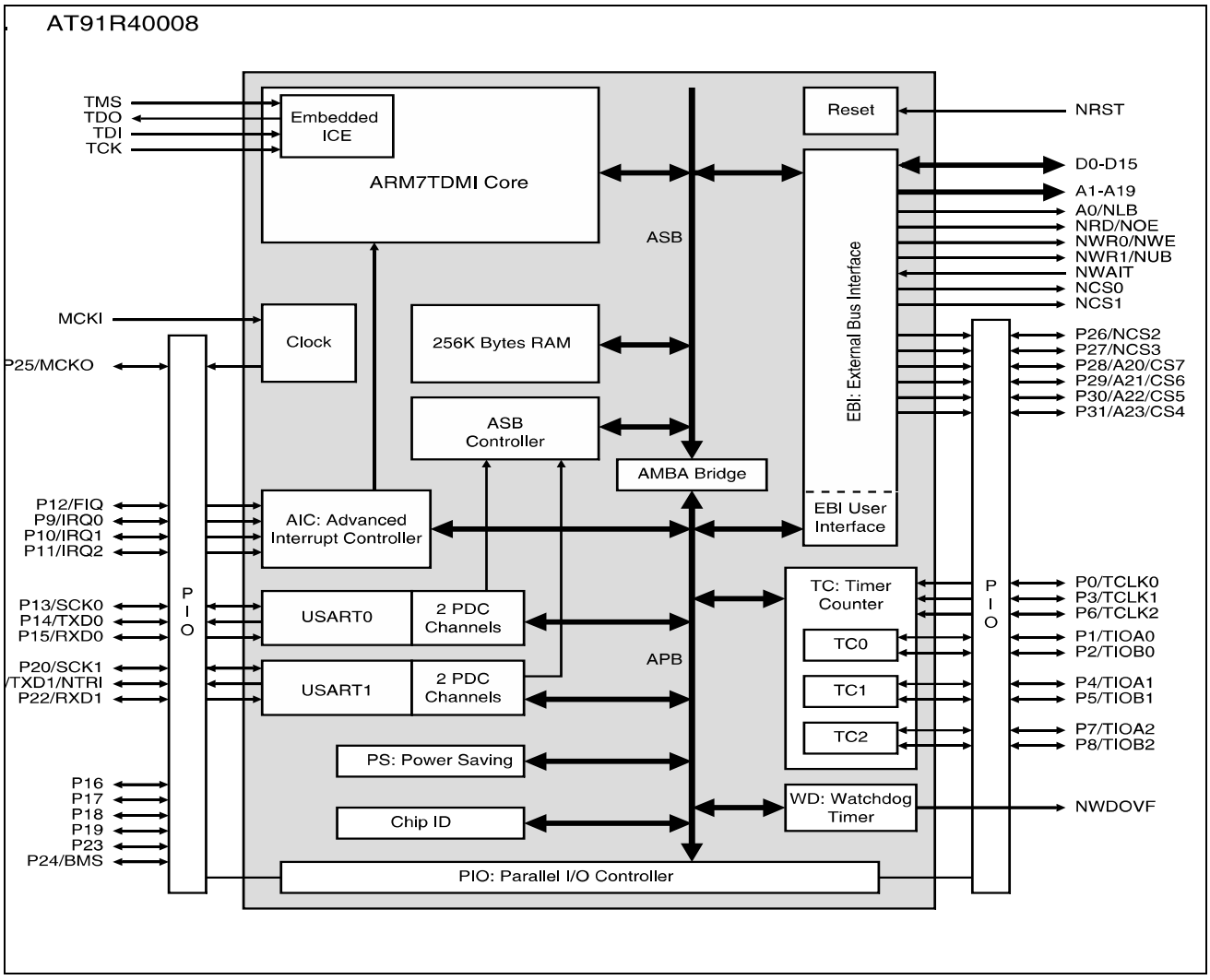
Σε αυτή τη δημοσίευση **22** δοκιμάζονται πολλοί διαφορετικοί αλγόριθμοι κρυπτογράφησης για χρήση σε ενσωματωμένα συστήματα με σκοπό να ανακαλύψει τις επιπτώσεις που έχει η αρχιτεκτονική των ενσωματωμένων συστημάτων στην απόδοση των αλγορίθμων, για να βοηθήσει τους σχεδιαστές να προβλέπουν την απόδοση του συστήματος όταν χρησιμοποιείται για κρυπτογράφηση.

5. Υλοποίηση

Στο κεφάλαιο αυτό αναφέρεται και περιγράφεται το υλικό (hardware) και το λογισμικό (software) που χρησιμοποιήθηκαν για την υλοποίηση της εργασίας αυτής.

5.1 Hardware

Στη συγκεκριμένη εργασία χρησιμοποιήθηκαν δύο μικροελεγκτές AT91R40008 της εταιρίας Atmel.



Εικόνα 5.1 Σχέδιο μικροελεγκτή at91r40008

Ο επεξεργαστής είναι τεχνολογίας RISC υψηλής απόδοσης με μήκος λέξης των 16 bit , ενώ έχει πολύ χαμηλή κατανάλωση ενέργειας. Ακόμη έχει πάνω στο τσιπ SRAM με μέγεθος 256 KB όπως και μεγάλο αριθμό εσωτερικά κατατεθειμένων καταχωρητών. Αυτό έχει σαν αποτέλεσμα ο μικροελεγκτής να έχει πολύ γρήγορη απόκριση στη διαχείριση των exception που το κάνει ιδανικό για εφαρμογές ελέγχου πραγματικού χρόνου.

Ο μικροελεγκτής AT91R4008 έχει χαρακτηριστικό την απευθείας σύνδεση με τη μνήμη που βρίσκεται εκτός τσίπ συμπεριλαμβανομένης της μνήμης flash μέσω ενός πλήρως προγραμματιζόμενου εξωτερικού διαύλου .Ένα χειριστή διακοπών 8 επιπέδων σε συνδυασμό με τον ελεγκτή περιφερειακών δεδομένων βελτιώνει τις επιδόσεις πραγματικού χρόνου.

5.1.1 Αρχιτεκτονική μικροελεγκτή

Η αρχιτεκτονική του μικροελεγκτή αποτελείται απο δύο κύριους διαδρόμους. Τον advanced system Bus (ASB) και τον advanced peripheral Bus(APB). Ο ASB Ελέγχεται από τον ελεγκτή μνήμης και διασυνδέει τον επεξεργαστή με τις on-chip 32 μπιτες μνήμες, τον EBI(external bus interface)και την γέφυρα AMBA. Ο APB έχει σχεδιαστεί για να έχει πρόσβαση στα on-chip περιφερειακά και είναι σχεδιασμένο για χαμηλή κατανάλωση ενέργειας.

5.1.2 Μνήμες.

Ο μικροελεγκτής ενσωματώνει μια εσωτερική SRAM με χωρητικότητα 256 KB. Η εσωτερική μνήμη είναι απευθείας συνδεδεμένη στον 32 μπιτο data bus. Ο AT91R40008 διαθέτει έναν εξωτερικό διάδρομο(EBI) ο οποίος επιτρέπει τη σύνδεση των εξωτερικών μνημών και την εφαρμογή ειδικών περιφερειακών. Ο EBI υποστηρίζει συσκευές των 8 ή 16 bit και μπορεί να χρησιμοποιήσει δυο 8 μπιτα συστήματα για να εξομοιώσει μία 16 μπιτη συσκευή.

5.1.3 Περιφερειακά

Ο μικροελεγκτής ενσωματώνει ακόμη αρκετά περιφερειακά που ταξινομούνται ως περιφερειακά συστήματος ή περιφερειακά χρήστη. Όλα τα on-chip περιφερειακά είναι 32 bit και είναι προσβάσιμα από τη γέφυρα AMBA. Το περιφερειακό σετ καταχωρητών αποτελείται από τους καταχωρητές control, mode, data, status, enable, disable. Ένας on-chip ελεγκτής περιφερειακών δεδομένων μεταφέρει δεδομένα μεταξύ on-chip usarts και on-off-chip χώρο διευθύνσεων μνημών χωρίς την παρέμβαση του επεξεργαστή.

5.1.3.1 Περιφερειακά συστήματος.

Ο δίαυλος εξωτερικής διασύνδεσης (EBI) ελέγχει την εξωτερική μνήμη ή τις περιφερειακές συσκευές μέσω ενός 8 μπιτου ή 16 μπιτου διαδρόμου δεδομένων και προγραμματίζεται μέσω του APB.

Η μονάδα εξοικονόμησης ενέργειας (PS) εφαρμόζει την κατάσταση αναμονής και δίνει τη δυνατότητα στο χρήστη να προσαρμόσει την κατανάλωση ενέργειας του μικροελεγκτή σύμφωνα με τις απαιτήσεις της εφαρμογής.

Ο advanced interrupt controller (AIC) ελέγχει τις εσωτερικές πηγές διακοπής από τα εσωτερικά περιφερειακά και τις τέσσερις γραμμές εξωτερικής διακοπής ,συμπεριλαμβανομένου του fiq, για να δώσει μια διακοπή irq ή fiq. Ο έλεγχος προτεραιότητας είναι 8 επιπέδων και έτσι μειώνεται ο χρόνος καθυστέρησης του interrupt. Ο παράλληλος ελεγκτής εισόδου εξόδου ελέγχει εώς32 γραμμές εισόδου εξόδου.

5.1.3.2 Περιφερειακά χρήστη.

Δυο ανεξάρτητης διαμόρφωσης USARTs επιτρέπουν την επικοινωνία σε υψηλό ρυθμό με σύγχρονο ή ασύγχρονο τρόπο. Κάθε USART περιλαμβάνει ένα time out register και ένα time guard register , που διευκολύνουν τη χρήση των δύο ειδικών καναλιών των ελεγκτών περιφερειακών δεδομένων. Ακόμη διαθέτει ένα timer/counter block που περιλαμβάνει τρία πανομοιότυπα timer/counter κανάλια των 16 bit. Το κάθε κανάλι μπορεί να προγραμματιστεί ανεξάρτητα και να εκτελέσει ένα ευρύ φάσμα λειτουργιών όπως μέτρηση συχνότητας , καταμέτρηση γεγονότων , παραγωγή παλμού, μέτρηση χρόνου καθυστέρησης και διαμόρφωση παλμού πλάτους .

5.1.4 Χαρακτηριστικά μικροελεγκτή

• Incorporates the ARM7TDMI® ARM® Thumb® Processor Core

- – High-performance 32-bit RISC Architecture – High-density 16-bit Instruction Set□– Leader in MIPS/Watt□– Little-endian
- – EmbeddedICETM (In-circuit Emulation) • 8-, 16- and 32-bit Read and Write Support • 256K Bytes of On-chip SRAM
- – 32-bit Data Bus
- – Single-clock Cycle Access□• Fully Programmable External Bus Interface (EBI)
- – Maximum External Address Space of 64M Bytes□– Up to Eight Chip Selects□– Software Programmable 8/16-bit External Data Bus

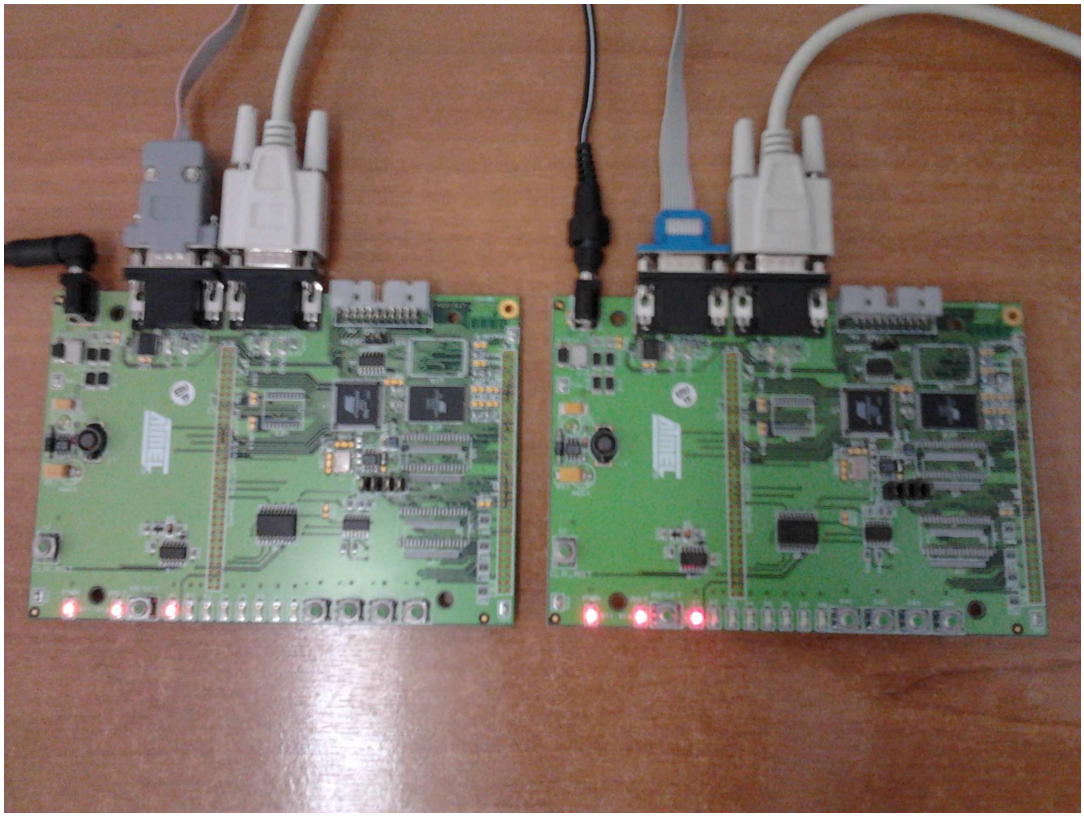
• Eight-level Priority, Individually Maskable, Vectored Interrupt Controller□– Four External Interrupts, including a High-priority, Low-latency Interrupt Request

• 32 Programmable I/O Lines□• Three-channel 16-bit Timer/Counter

- – Three External Clock Inputs
- – Two Multi-purpose I/O Pins per Channel • Two USARTs
- – Two Dedicated Peripheral Data Controller (PDC) Channels per USART • Programmable Watchdog Timer□• Advanced Power-saving Features
- – CPU and Peripheral Can be Deactivated Individually • Fully Static Operation:
- – 0 Hz to 75 MHz Internal Frequency Range at VDDCORE = 1.8V, 85° C • 2.7V to 3.6V I/O Operating Range□• 1.65V to 1.95V Core Operating Range□• -40°Cto+85°CTemperatureRange□• Available in 100-lead LQFP Package (Green)

5.1.5 Συνδεσμολογία

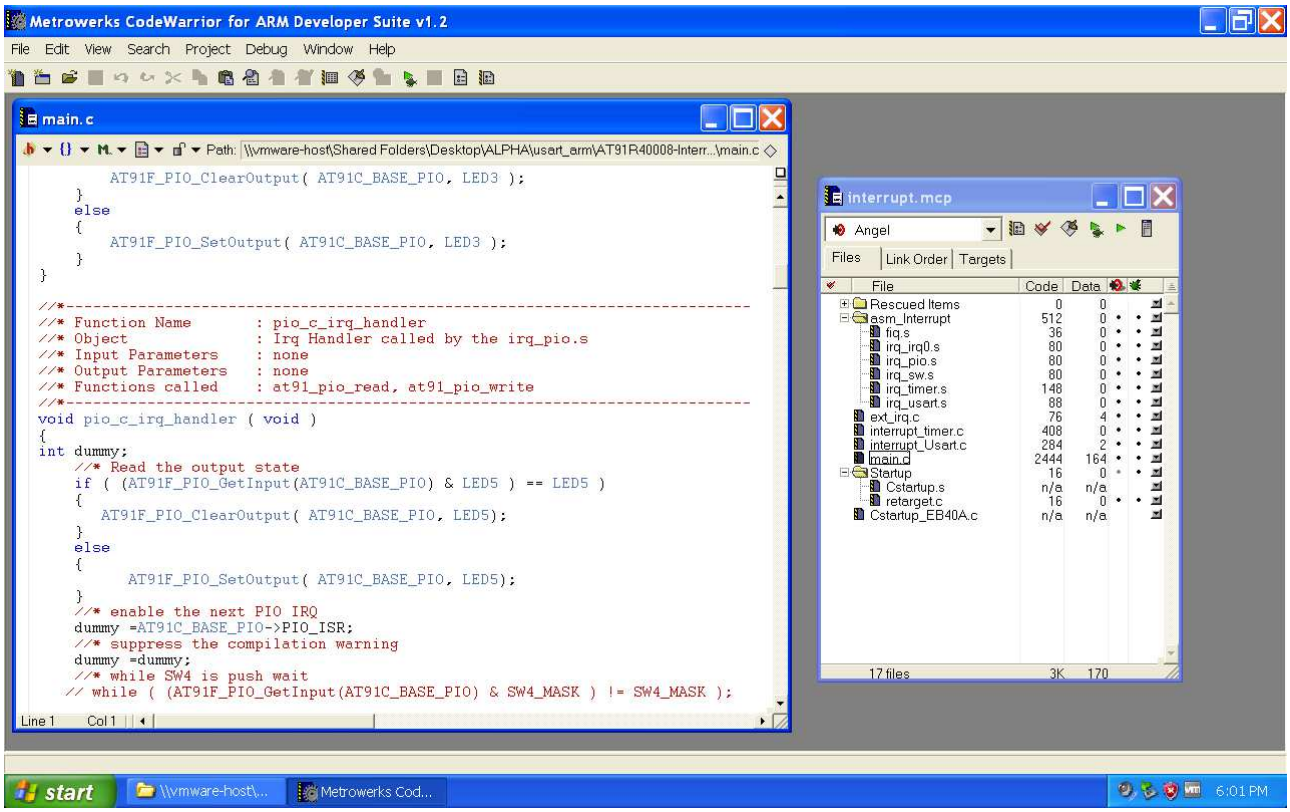
Οι δύο μικροελεγκτές είναι συνδεδεμένοι μεταξύ τους μέσω της μίας σειριακής θύρας επικοινωνίας usart η οποία έχει ρυθμιστεί να λειτουργεί με ρυθμό μετάδοσης 115 200 bps. Η άλλη σειριακή θύρα επικοινωνίας usart χρησιμοποιείται για να συνδεθεί ο κάθε μικροελεγκτής με έναν υπολογιστή μέσου του οποίου φορτώνεται το πρόγραμμα στους μικροελεγκτές. Τέλος ο κάθε μικροελεγκτής τροφοδοτείται με ρεύμα από ένα μετασχηματιστή με την απαιτούμενη τάση των 9 v.



Εικόνα 5.2 Φωτογραφία των μικροελεγκτών της εργασίας

5.2 Software

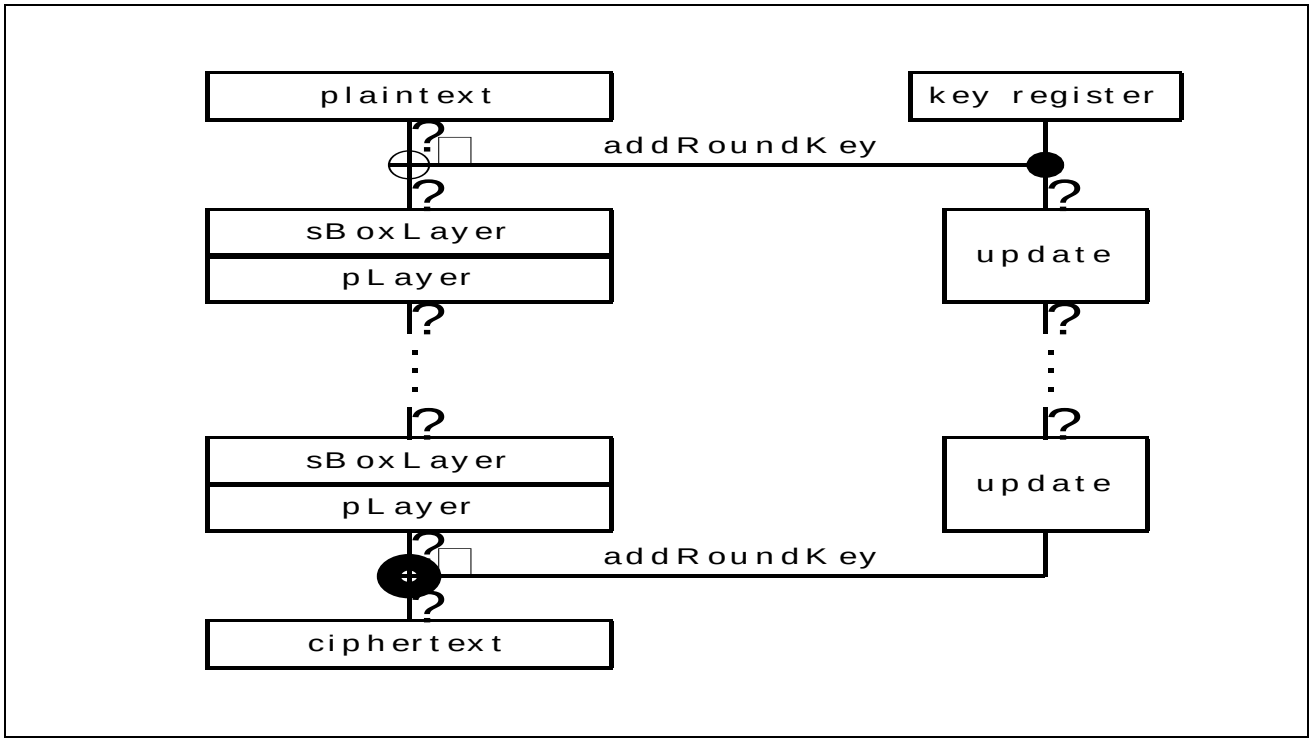
Το πρόγραμμα που χρησιμοποιήθηκε για την υλοποίηση της εργασίας είναι το Metrowerks codewarrior for arm developer suit v1.2. Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε είναι η c γιατί είναι μία πολύ καλή γλώσσα για εφαρμογές ενσωματωμένων συστημάτων. Κάποια από τα πλεονεκτήματα που μπορούμε να πούμε οτι έχει σε σχέση με τις άλλες γλώσσες προγραμματισμού είναι οτι είναι απλή αλλά και μικρή και έχει έναν συμπαγή και γρήγορο κώδικα.



Εικόνα 5.3 Λογισμικό Metrowerks

5.2.1 Αλγόριθμος κρυπτογράφησης

Ο αλγόριθμος κρυπτογράφησης που χρησιμοποιήθηκε σ' αυτήν την πτυχιακή είναι ο Present. Είναι ένας συμμετρικός αλγόριθμος δέσμης ελαφριάς κρυπτογράφησης . Το μέγεθος της δέσμης του είναι 64 bits και έχει τη δυνατότητα να χρησιμοποιήσει δύο μεγέθη κλειδιών των 80 ή 128 bits. Εδώ χρησιμοποιείται το πρώτο κλειδί των 80 bits.



Εικόνα 5.4 Σχηματική περιγραφή αλγορίθμου present

5.2.1.1 Περιγραφή Κρυπτογράφησης present

Η κρυπτογράφηση γίνεται σε 31 γύρους. Σε κάθε γύρο γίνονται τα παρακάτω: Στην αρχή γίνεται η πράξη xor ανάμεσα στη δέσμη και στα 64 από τα 80 bits του κλειδιού.

Πράξη xor ανάμεσα σε δέσμη και κλειδί

```
round=0;
        do
        {
//      ****************** addRoundkey ******************
                i=0;
                do
                {
                        state[i] = state[i] ^ key[i+2];
                        i++;
                }
                while(i<=7);
```

1

Αμέσως μετά έχουμε το sBoxlayer . Το sBox που χρησιμοποιείται έχει 4 bit είσοδο και 4 bit έξοδο και είναι το εξής.

Πίνακας 5.1 sLAYER

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Στο επίπεδο αυτό τα 64 bits της δέσμης αλλάζουν ανά τετράδες σύμφωνα με τον παραπάνω πίνακα. Για παράδειγμα το byte 11110011 ή F3 στο δεκαεξαδικό θα γίνει 00101011 ή 2B στο δεκαεξαδικό.

Κώδικας sLayer

```
****************** sBox ********************************
   do
   {
            i--;
            state[i] = sBox4[state[i]>>4]<<4 | sBox4[state[i] & 0xF];
   }
   while(i>0);
```

8

Έπειτα ακολουθεί το Player(permutation layer).Εδώ κάθε bit της δέσμης μετατοπίζεται .Η μετατόπιση των bit δίνεται από τον παρακάτω πίνακα.

Πίνακας 5.2 pLAYER

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P(i)	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
I	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P(i)	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
I	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
P(i)	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
I	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
P(i)	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

Για παράδειγμα το bit που βρίσκεται στη θέση 12 θα μεταφερθεί στη θέση 3.

Κώδικας pLayer

```
****************** pLayer ******************************

                for(i=0;i<8;i++)//clearing of the temporary array temp_pLayer
                {
                        temp_pLayer[i] = 0;
                }
                for(i=0;i<64;i++)
                {
                        position = (16*i) % 63;   //arithmetic calculation of the pLayer
                        if(i == 63)
                    //exception for bit 63
                                position = 63;
                        element_source          = i / 8;
                        bit_source                      = i % 8;
                        element_destination     = position / 8;
                        bit_destination  = position % 8;
                        temp_pLayer[element_destination] |= ((state[element_source]>>bit_source) & 0x1)
<<bit_destination;
                }
                for(i=0;i<=7;i++)                                                                       {
                        state[i] = temp_pLayer[i];
                }
//          ****************** End pLayer ****************************
```

Τέλος μετατοπίζονται τα bit του κλειδιού και μετά εφαρμόζεται στο κλειδί το sBoxlayer όπως εφαρμόστηκε παραπάνω στη δέσμη.

Κώδικας key scheduling

```
********** Key Scheduling **********

//              on-the-fly key generation

                save1  = key[0];
                save2  = key[1];
                i = 0;
                do
                {
                        key[i] = key[i+2];
                        i++;
                }
                while(i<8);
                key[8] = save1;
        //61-bit left shift
                key[9] = save2;
                i = 0;
                save1 = key[0] & 7;
                do
                {
                        key[i] = key[i] >> 3 | key[i+1] << 5;
                        i++;
                }
                while(i<9);
                key[9] = key[9] >> 3 | save1 << 5;
                key[9] = sBox4[key[9]>>4]<<4 | (key[9] & 0xF);
        //S-Box application
                if((round+1) % 2 == 1)
        //round counter addition
                        key[1] ^= 128;
                key[2] = ((((round+1)>>1) ^ (key[2] & 15)) | (key[2] & 240));

//      ********** End Key Scheduling **********
                round++;
        }
        while(round<31);
```

12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

Μετά το τέλος των 31 γύρων γίνεται η πράξη xor ανάμεσα στη δέσμη και το κλειδί.

Πράξη xor ανάμεσα σε κλειδί και δέσμη

```
****************** addRoundkey ***************************

        i = 0;
        do
       //final key XOR
        {
                state[i] = state[i] ^ key[i+2];
                i++;
        }
        while(i<=7);
```

1
2
3
4
5
6
7
8
9
10

5.2.1.2 Περιγραφή αποκρυπτογράφησης present.

Στο πρώτο μέρος της αποκρυπτογράφησης βρίσκεται η τιμή που έχει το κλειδί σε κάθε ένα από τους 31 γύρους και αποθηκεύεται στον πίνακα subkey.
Αμέσως μετά ξεκινούν οι 31 γύροι κατα τους οποίους εφαρμόζεται η πράξη xor μεταξύ της δέσμης και του κλειδιού του κάθε γύρου, αλλά και οι ανεστραμμένοι πίνακες των PLAYER και SLAYER.

Πίνακας 5.3 pLAYER αποκρυπτογράφησης

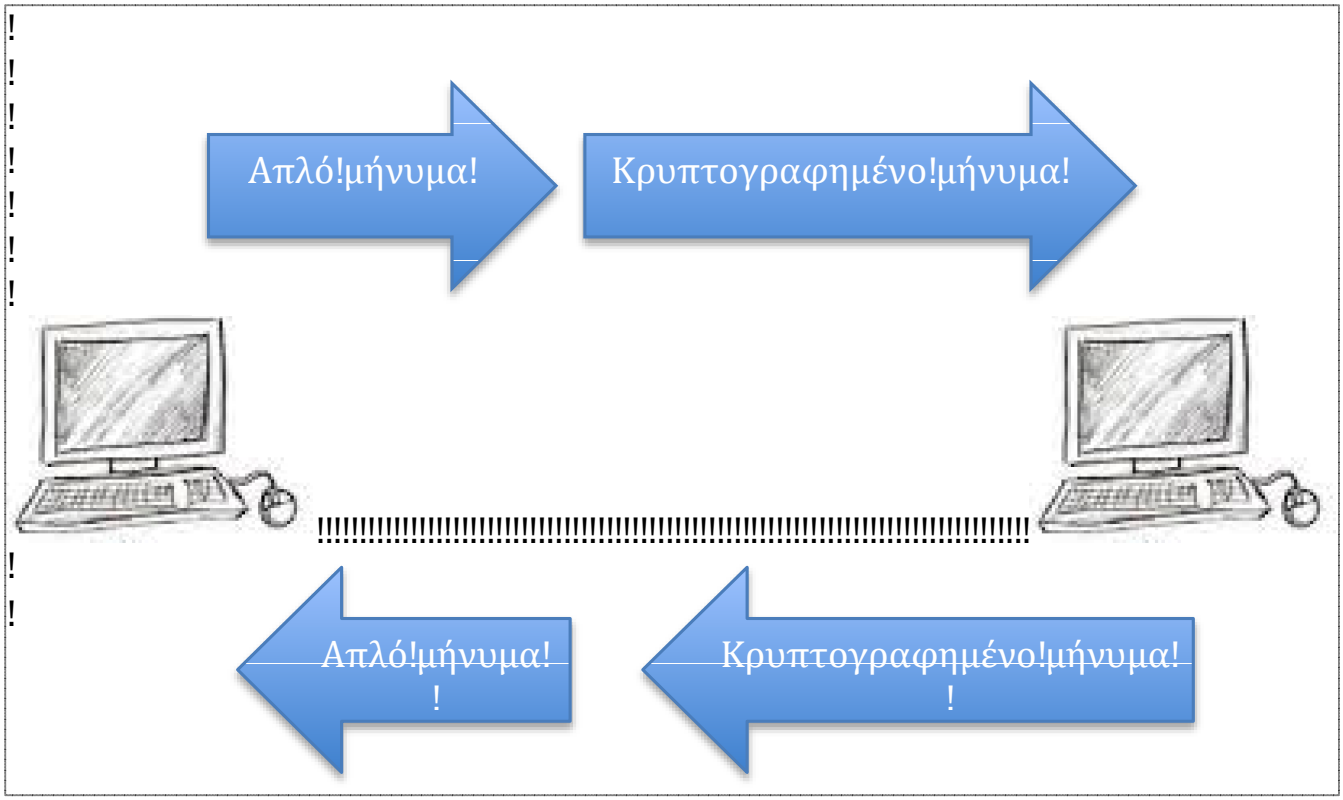
I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P(i)	0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
I	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P(i)	1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
I	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
P(i)	2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
I	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
P(i)	3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63

Πίνακας 5.4 sLAYER αποκρυπτογράφησης

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	5	E	F	8	C	1	2	D	B	4	6	3	0	7	9	A

5.2.2 Πρωτόκολλο επικοινωνίας

Η πληροφορία που αλλάζουν μεταξύ τους οι μικροελεγκτές ποικίλει. Μπορεί να περιέχει πολύ σημαντικά δεδομένα που είναι απαραίτητο να κρυπτογραφηθούν. Μπορεί όμως μεγάλο μέρος της πληροφορίας να μη χρειάζεται να κρυπτογραφηθεί. Επειδή η διαδικασία της κρυπτογράφησης και αποκρυπτογράφησης χρειάζεται χρόνο και ενέργεια για να εκτελεστεί, όταν γίνεται χωρίς να είναι απαραίτητο υπάρχει σπατάλη. Για να αποφευχθεί κάτι τέτοιο θα πρέπει να είναι εφικτή η επικοινωνία των μικροελεγκτών μέσω κρυπτογραφημένων, αλλά και μη κρυπτογραφημένων μηνυμάτων.



Εικόνα 5.5 Επικοινωνία μοντέλου

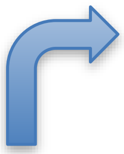
Για να γίνει αυτό θα πρέπει

- 1. ο μικροελεγκτής που στέλνει το μήνυμα να έχει δύο επιλογές. Η μία είναι να κρυπτογραφεί το μήνυμα που θέλει να στείλει ,και μετά να το στέλνει. Η άλλη επιλογή είναι να στέλνει το μήνυμα όπως είναι χωρίς να το κρυπτογραφεί.

- 2. Ο μικροελεγκτής που λαμβάνει το μήνυμα να καταλαβαίνει αν αυτό είναι κρυπτογραφημένο ή όχι.

Για να επιτευχθεί το πρώτο χρησιμοποιήθηκαν δύο κουμπιά που αναλογούν σε αυτές τις δύο διαφορετικές λειτουργίες. Όταν πατηθεί το κουμπί 3 τότε καλείται η συνάρτηση simplesend. Η συνάρτηση αυτή στέλνει μία δέσμη δεδομένων (ένα block), χωρίς πρώτα να το κρυπτογραφήσει. Όταν πατηθεί το κουμπί 4 καλείται η συνάρτηση crypto, Η συνάρτηση αυτή κρυπτογραφεί τη δέσμη δεδομένων και μετά την αποστέλλει.

Για να καταλαβαίνει ο δέκτης του μηνύματος αν αυτό είναι κρυπτογραφημένο ή όχι , ο αποστολέας τον ενημερώνει στέλνοντας πριν τη δέσμη ένα απλό flag. Όταν θέλει να στείλει κρυπτογραφημένο μήνυμα στέλνει πρώτα τον αριθμό 00000010(δυαδικό, ή 2 στο δεκαδικό).Όταν θέλει να στείλει ένα απλό μήνυμα στέλνει πριν απ' αυτό ,τον αριθμό 00000011 (δυαδικό, ή 3 στο δεκαδικό). Έτσι αν ο δέκτης λάβει τον αριθμό 2 καταλαβαίνει ότι το μήνυμα που ακολουθεί θα πρέπει να το αποκρυπτογραφήσει, ενώ αν λάβει τον αριθμό 3 καταλαβαίνει ότι το μήνυμα που ακολουθεί δεν χρειάζεται αποκρυπτογράφηση.



1!!BYTE!πρωτοκόλλου!

!!!8!BYTES!!πληροφορίας!

02!	FF!	38!	A7!	42!	B3!	52!	2C!	18!

Εικόνα 5.6 Πακέτο μεταφοράς

5.2.3 Συγχρονισμός μετάδοσης

Για να επιτευχθεί η σωστή μετάδοση των δεδομένων από τον ένα μικροελεγκτή στον άλλο χρησιμοποιήθηκε ένας interrupt timer1 για τον κάθε μικροελεγκτή. Το ρολόι των μικροελεγκτών λειτουργεί με ρυθμό 75 MHZ .Οι interrupt timers έχουν συγχρονιστεί να λειτουργούν σε ρυθμό mck 8. Δηλαδή F = 75 MHZ / 8 = 9,375MHZ T=1/9,375 MHZ T= 106 ns. Ο μικροελεγκτής που κάνει την αποστολή του μηνύματος αμέσως μετά που στέλνει το flag ,για να δείξει αν είναι κρυπτογραφημένο ή όχι, ενεργοποιεί τον timer. Όποτε ο counter του timer αλλάζει ,στέλνει από ένα byte της δέσμης που θέλει να στείλει, μέχρι να σταλεί ολόκληρη η δέσμη. Αμέσως μετά απενεργοποιείται ο timer. Για να εξασφαλίσουμε οτι θα γίνει μόνο μία αποστολή κάθε φορά που θα αλλάζει τιμή ο counter του timer χρησιμοποιούμε ένα flag το 'timingflag'.

Υπόδειγμα κώδικα αποστολής block.

```
forSend[1] = 2;
AT91F_US_SendFrame(COM0,(char *)forSend,sizeof(forSend));
timer_init();
q=0;
timingflag=0;
while(count_timer1_interrupt<9)
   {
     if((count_timer1_interrupt % 1)==0&& count_timer1_interrupt  !=timingflag)
        {
         forSend[1] = state[q];
         AT91F_US_SendFrame(COM0,(char *)forSend,sizeof(forSend));
         q++;
         timingflag= count_timer1_interrupt;
        }
   }
AT91C_BASE_TC1->TC_CCR = AT91C_TC_CLKDIS ;
AT91C_BASE_TC0->TC_CCR = AT91C_TC_CLKDIS ;
AT91F_PIO_SetOutput( AT91C_BASE_PIO, LED7 );
AT91F_PIO_SetOutput( AT91C_BASE_PIO, LED8 );
```

Ο μικροελεγκτής που λαμβάνει το μήνυμα, μόλις λάβει το flag ,καλεί τη συνάρτηση λήψης. Μέσα στην συνάρτηση αυτή ενεργοποιείται ο αντίστοιχος timer. Κάθε φορά που αλλάζει ο counter του timer κάνει λήψη ,και γεμίζει τον 64 μπιτο πίνακα state.Χρησιμοποιούμε κι εδώ το flag “timingflag”για τον ίδιο λόγο που το χρησιμοποιήσαμε πριν. Μετά το τέλος της λήψης ολόκληρης της δέσμης απενεργοποιούμε τον timer.

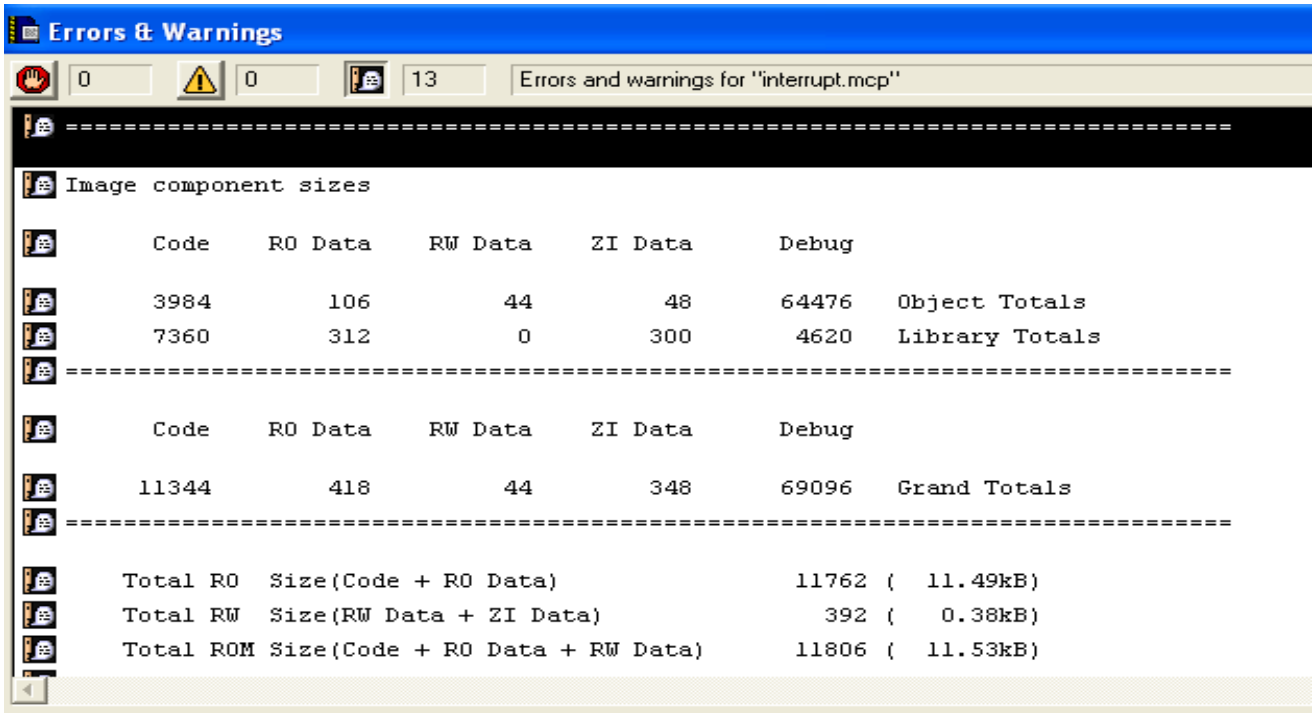
Συνάρτηση λήψης block.

```
Int recieveblok(void)
{
 AT91PS_USART COM0;

 COM0= AT91C_BASE_US1;

timer_init();
for(;;)
    {
 if((count_timer1_interrupt % 1)==0&&  count_timer1_interrupt!=timingflag)
         {
          delay3();

          AT91F_US_ReceiveFrame(COM0,(char *)my_eol,sizeof(my_eol));
          j=(int)my_eol[1];
          fillstate2();
          timingflag=count_timer1_interrupt;
          }

  if(i==8)
     {
      if(decflag==0)
        {
         for(q=0;q<i;q++)
           {
             printf(" %x",state2[q]);
           }
         printf("\n") ;
         i=0;
        }
      AT91C_BASE_TC1->TC_CCR = AT91C_TC_CLKDIS ;
      AT91C_BASE_TC0->TC_CCR = AT91C_TC_CLKDIS ;
      AT91F_PIO_SetOutput( AT91C_BASE_PIO, LED7 );
      AT91F_PIO_SetOutput( AT91C_BASE_PIO, LED8 );
     return0;
     }
   }
}
```

10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39

6. Μετρήσεις

Το μέγεθος του προγράμματος είναι 11.53KB. Αποτελείται από τα 11,344 KB του κώδικα ,τα 418 B των read only δεδομένων , τα 44 B των read write δεδομένων και τα 348 των Zero Initialized δεδομένων.



Εικόνα 6.1 Πίνακας debug

Για να μετρήσουμε τους χρόνους που χρειάζονται οι μικροελεγκτές για να ολοκληρώσουν τις διαδικασίες κρυπτογράφησης και αποκρυπτογράφησης χρησιμοποιήσαμε έναν αντίστοιχο timer με αυτόν που χρησιμοποιήσαμε για τον συγχρονισμό αποστολής και λήψης των δεδομένων. Ο timer αυτός είναι ο timer0 και είναι συγχρονισμένος σε ταχύτητα mck2. Δηλαδή αφού η συχνότητα F του ρολογιού των μκροελεγκτών είναι 75 MHZ , η συχνότητα F του timer0 θα είναι : F=75/2 F= 37,5 MHZ ενώ η περίοδος T θα είναι : T= 1/37,5 MHZ T = 26 ns. Αυτό σημαίνει ότι κάθε 26 ns ,ο counter του timer0 αυξάνεται κατά 1.

-Από την αρχή μέχρι το τέλος της διαδικασίας της κρυπτογράφησης ο counter έφτασε μέχρι την τιμή 62.

Αυτό σημαίνει ότι στον μικροελεγκτηAT91R40008 της ATMEL, η διαδικασία της κρυπτογράφησης με τον αλγόριθμο ελαφριάς κρυπτογράφησης present διαρκεί 62 * 26= 1,612 μs.

- Κατά την χρονομέτρηση της διαδικασίας της αποκρυπτογράφησης η τιμή που πήρε ο counter είναι 210. Δηλαδή διαρκεί :210 * 26 =5,46 μs.

-Για την αποστολή χρησιμοποιούμε τον timer1 με mck8 .Κάθε φορά που αλλάζει ο counter, στέλνει από ένα byte. Άρα για να στείλει ολόκληρη τη δέσμη(block) συν το byte πρωτοκόλλου χρειάζεται: 8 * 13 ns *9=0,94 μs

Παρακάτω βλέπουμε σε πίνακες τα μεγέθη και τον χρόνο που χρειάζονται μία φωτογραφία, ένα αρχείο ήχου και ένα βίντεο για να κρυπτογραφηθούν, να σταλούν και να αποκρυπτογραφηθούν το καθένα απ ‘ αυτά.

Πίνακας 6.1 Μεγέθη πληροφορίας

Είδος πληροφορίας	Καθαρό μέγεθος αρχείου	μέγεθος πληροφορίας προς μετάδοση
Φωτογραφία jpeg 2048x1536 pixels	1,4 MB	1,575 MB
Ήχος mp3 192 Kbps Διάρκεια 3.31 min	5.1 MB	5,7375 MB
Βίντεο mpeg 4, 640x480 pixels , Διάρκεια 1.51 min	22,8 MB	24,85 MB

Το κάθε αρχείο για να αποσταλεί χρειάζεται να τεμαχιστεί σε πακέτα των 8 byte το καθ' ένα. Σε κάθε πακέτο προστίθεται ένα ακόμη byte σαν επικεφαλίδα , για το πρωτόκολλο επικοινωνίας .

Πίνακας 6.2 Χρόνοι επιμέρους διαδικασιών

είδος πληροφορίας	Διάρκεια κρυπτογράφησης	Διάρκεια αποστολής	Διάρκεια αποκρυπτογράφησης
Jpeg	282,1 ms	164,5 ms	955,5 ms
Mp3	1,027 sec	599 ms	3,48 sec
Mpeg 4	4,594 sec	2,679 sec	15,561 sec

Πίνακας 6.3 Συνολικοί χρόνοι

είδος πληροφορίας	Συνολικός χρόνος
Jpeg	1,402 sec
Mp3	5,106 sec
Mpeg 4	22,834 sec

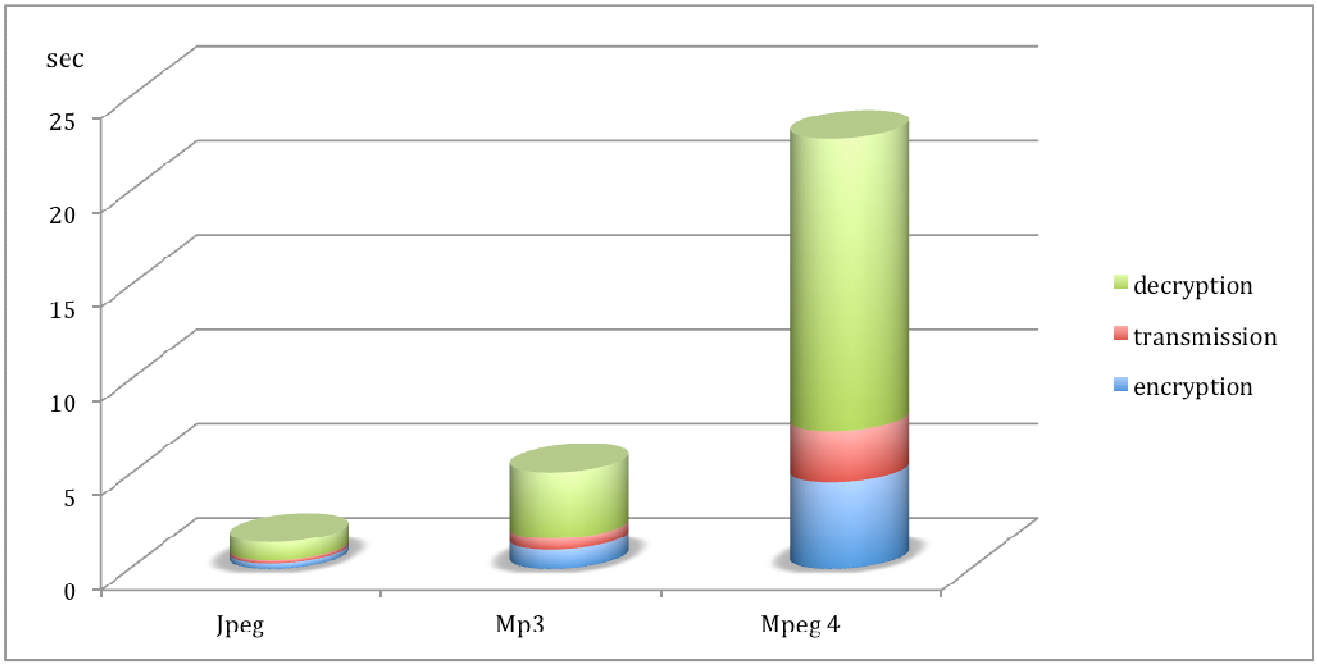


Figure 6.1 Διάρκεια κρυπτογράφησης, αποστολής , αποκρυπτογράφησης σε sec για κάθε αρχείο.

Παρακάτω θα δούμε την απόδοση του συστήματος σε περίπτωση ασφαλούς επικοινωνίας , που περιλαμβάνει την κρυπτογράφηση ,αποστολή και αποκρυπτογράφηση τριών φωτογραφιών Jpeg ,τεσσάρων αρχείων ήχου Mp3 και δύο βίντεο Mpeg 4 .Τα προς αποστολή αρχεία είναι τα ίδια που χρησιμοποιήθηκαν στο προηγούμενο παράδειγμα.

Πίνακας 6.4 Μεγέθη πληροφορίας

Είδος πληροφορίας	Όγκος πληροφορίας	Πληροφορία προς αποστολή
3 Jpeg 2048x1536 pixels	4,2 MB	4,725 MB
4 mp3 192 Kbps Διάρκεια 14,06 min	20,4 MB	22,95 MB
2 mpeg 4, 640x480 pixels , Διάρκεια 13,2 min	45,6 MB	49,7 MB
Σύνολο :	70,2 MB	77,375 MB

Πίνακας 6.5 Χρόνοι επιμέρους διαδικασιών

Όγκος αρχείων	Όγκος προς αποστολή δεδομένων	Διάρκεια κρυπτογράφησης	Διάρκεια αποστολής	Διάρκεια αποκρυπτογράφησης
70,2 MB	77,375 MB	14,145 sec	8,248 sec	47,911 sec

Ο συνολικός χρόνος που χρειάζεται για κρυπτογράφηση, αποστολή και αποκρυπτογράφηση για όλα αυτά τα αρχεία μαζί είναι 70,304 sec.

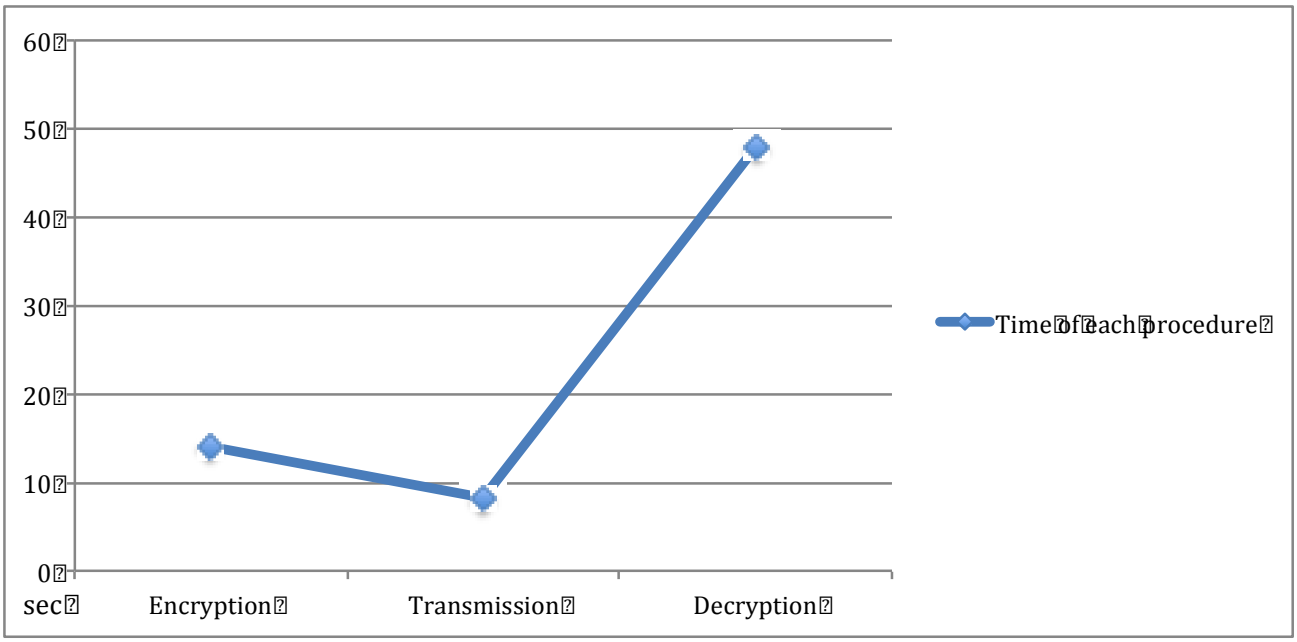


Figure 2 Χρόνοι επιμέρους διαδικασιών για όλο τον όγκο των δεδομένων.

7. Συμπεράσματα

Στην εργασία αυτή αφού φτιάξαμε το μοντέλο σύνδεσης δύο μικροελεγκτών at91r40008 μέσω ενός πρωτοκόλλου επικοινωνίας ,καταγράψαμε την απόδοση του αλγορίθμου ελαφριάς κρυπτογράφησης present Ο αλγόριθμος present είναι πολύ γρήγορος σε σχέση με άλλους αλγορίθμους κρυπτογράφησης , από μετρήσεις που έχουν γίνει σε σχετικές εργασίες.

Ακόμη είδαμε ότι ο χρόνος αποκρυπτογράφησης είναι μεγαλύτερος από το χρόνο κρυπτογράφησης. Τέλος είδαμε ότι ο συνδυασμός του αλγορίθμου κρυπτογράφησης present με τον συγκεκριμένο μικροελεγκτή, θα μπορούσε να δώσει ένα πολύ αποδοτικό ενσωματωμένο σύστημα, με δυνατότητα επιλογής διαβάθμισης του επιπέδου των παραμέτρων ασφάλειας και ταχύτητας .

Βιβλιογραφία

(1). APPLIED CRYPTOGRAPHY by Bruce Schneier

(2). http://www.cse.iitk.ac.in/users/anuag/crypto.pdf

(3).AT91 ARM Thumb Microcontroller, Atmel Corporation, 2002.

(4).http://www.atmel.com/Images/1732s.pdf

(5).http://imperia.rz.rub.de:9085/imperia/md/content/texte/theses/thesis_poschmann.pdf

(6).http://mathsci.ucd.ie/~gmg/ECC2007Talks/poschmann_LWC.pdf

(7).http://el.wikipedia.org/wiki/Μικροελεγκτής

(8).http://kremlinencrypt.com/algorithms.htm#Rijndael

(9).http://www.garykessler.net/library/crypto.html

(10).http://www.cl.cam.ac.uk/~rja14/Papers/SE-05.pdf

(11).http://en.wikibooks.org/wiki/Cryptography

(12).http://www.uwspace.uwaterloo.ca/bitstream/10012/5064/1/Xinxin-ethesis.pdf

(13).http://www.scis.ulster.ac.uk/~kevin/IJISP-12.pdf

(14).http://www.uclouvain.be/crypto/ecrypt_lc11/static/pre_proceedings_2.pdf

(15).http://www.fileheap.com/dbquery/1/encryption+algorithms+c

(16).http://www.lightweightcrypto.org/implementations.php

(17).http://en.wikipedia.org/wiki/Sensor_node

(18).http://www.ampublisher.com/June%202010/EEE-1005-011-Microcontroller-Application-in-Cryptography-Techniques.pdf

(19).http://etd.aau.edu.et/dspace/bitstream/123456789/1510/1/Microsoft%20Word%20-%20final.pdf

(20).http://arxiv.org/pdf/0911.0482.pdf

(21) http://www.dcc.ufla.br/infocomp/artigos/v8.2/art05.pdf

(22) http://moss.csc.ncsu.edu/~mueller/ftp/pub/mueller/papers/cases03.pdf

Παραρτημα

Παρουσίαση

Στόχοι εργασίας

Στόχοι της εργασίας είναι 2:

* Δημιουργία πρωτοκόλλου για επικοινωνία με κρυπτογραφημένα αλλά και μη κρυπτογραφημένα μηνύματα
* Χρονομέτρηση απόδοσης αλγορίθμου

Ανάπτυξη συστήματος αποδοτικής κρυπτογράφησης για ενσωματωμένα συστήματα

Αυγενάκης Ιωάννης ΑΜ 623
Επιβλέπων καθηγητής :Κορνάρος Γεώργιος

Τμήμα εφαρμοσμένης πληροφορικής και πολυμέσων

Σύγκριση μικροεπεξεργαστών –μικροελεγκτών

Μικροεπεξεργαστής.

* έμφαση σε υπολογιστική ισχύ
* μη εξειδικευμένη λειτουργικότητα
* ευελιξία σε ποικιλία εφαρμογών

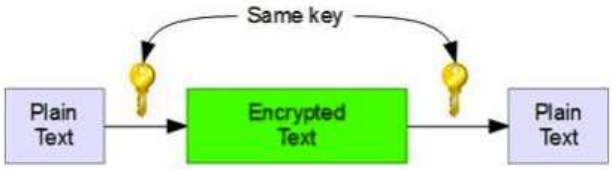
Μικροελεγκτής

* Μικρότερη ποικιλία εφαρμογών
* Μεγαλύτερη εξειδίκευση
* Αυτονομία
* Χαμηλό κόστος

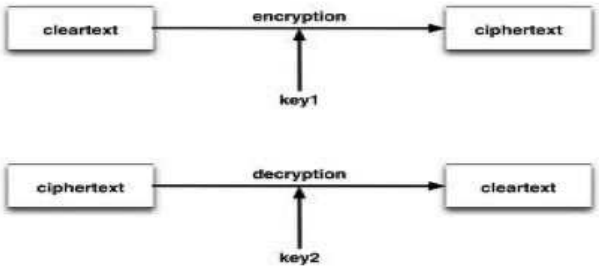
Κρυπτογραφία

Κατηγορίες κρυπτοσυστημάτων

- Συμμετρικά
  - stream cipher
  - block cipher



❖ Ασύμμετρα



* Κρυπτανάλυση

* Γνωστοί αλγόριθμοι
  - DES (data encryption standart)
  - RSA(Rivest, Shamir, Adleman.)
  - DSA(digital signature algorithm)

* Υβριδικά συστήματα

Ελαφριά κρυπτογράφηση

Υλοποίηση

* Hardware

μικροελεγκτής at91r40008 της atmel

τεχνολογίας RISC
μήκος λέξης 16 bit
SRAM 256 KB
χειριστής διακοπών 8 επιπέδων

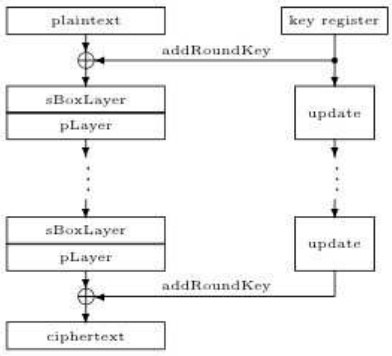
* Συνδεσμολογία

Υλοποίηση

* Software

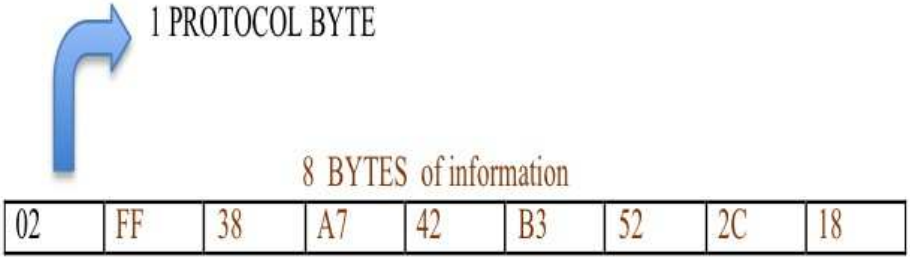
Αλγόριθμος present
lightweight block cipher
μήκος block 64 bit
2 επιλογές για μήκος κλειδιού 80 bit ή 128 bit
στην εργασία χρησιμοποιείται ο πρώτος των 80 bit

present



Πρωτόκολλο επικοινωνίας

πακέτο μεταφοράς



Συγχρονισμός μετάδοσης

χρήση των timers σε αποστολέα και δέκτη παράλληλη ενεργοποίηση

* Αποστολή ενός byte κάθε φορά που αυξάνεται ο count timer interrupt κατα 1 στον αποστολέα

* Λήψη ενός byte κάθε φορά που αυξάνεται ο count timer interrupt κατα 1 στον δέκτη

Μετρήσεις

* Η συχνότητα F του ρολογιού των μκροελεγκτών είναι 75 MHZ
* Ο timero είναι ρυθμισμένος σε συχνότητα mck2 άρα: η συχνότητα F του timero θα είναι : F=75/2 F= 37,5 MHZ
* η περίοδος T θα είναι : T= 1/37,5 MHZ T = 26 ns.

* Διάρκεια κρυπτογράφησης = 62 count timer interrupt. Αρα διαρκεί 62*26 ns =1,612 μs
* Διάρκεια κρυπτογράφησης = 210 count timer interrupt. Αρα διαρκεί 210 * 26 ns =5,46 μs
* Διάρκεια αποστολής 9 count timer interrupt του timer 1 με ρύθμιση mck8 .Αρα διαρκεί 8 * 13 ns *9 = 0,94 μs

Συμπεράσματα

* Καταγραφή απόδοσης αλγορίθμου present στον μικροελεγκτή at91r40008.
* Χρόνος κρυπτογράφησης < χρόνος αποκρυπτογράφησης.
* Δημιουργία αποδοτικού μοντέλου ενσωματωμένου συστήματος με δυνατότητα διαβάθμισης παραμέτρων ασφάλειας και ταχύτητας.

Εισαγωγή στους μικροεπεξεργαστές

* Χαρακτηριστικά μικροεπεξεργαστών
  - Συχνότητα λειτουργίας
  - Μήκος λέξης
  - Ρεπερτόριο εντολών
* Μικροελεγκτές

Κώδικας

```
//* File Name           :main.c


// Include Standard LIB  files
#include"eb40A.h"
#include"stdio.h"
#include"stdint.h"
#include"stdlib.h"

//*   Waiting time between LED1 and LED2
#define     WAIT_TIME       MCK

#define PIO_INTERRUPT_LEVEL             6
#define IRQ0_INTERRUPT_LEVEL    2
#define SOFT_INTERRUPT_LEVEL    5
#define FIQ_INTERRUPT_LEVEL             0



//* Global variable
int count_timer0_interrupt;
int count_timer1_interrupt;
volatile uint8_t state[8] = {0x80,0x90,0x70,0xff,0x05,0x06,0x07,0x08};
volatile uint8_t state2[8] = {0x56,0x40,0xfd,0x62,0x17,0xf3,0xef,0x2a};
volatile uint8_t state3[8] = {0x56,0x40,0xfd,0x62,0x17,0xf3,0xef,0x2a};
int i=0,j=0,timingflag=0,eingabe=5,q=0,decflag;
char forSend[2];
staticconstcharmy_eol[]= {" "} ;


// Use the Library Handler defined in file periph/pio/pio_irq/irq_pio.s

externvoid pio_asm_irq_handler(void);
externvoid irq0_asm_irq_handler(void);
externvoid sw_asm_irq_handler(void);
externvoid fiq_asm_handler(void);

// External Function Prototype
externint timer_init (void );
externvoid Usart_init (void);

//internal functions
void decipher(void);
```

```
void fillstate2(void);
int simplesend(void);
int recieveblok(void);
int crypto(void);
int value ;
//*-----------------------------------------------------------------------------
//* Function Name       :aic_software_interrupt
//* Object              : Software interrupt function
//* Input Parameters    : none
//* Output Parameters   : none
//* Functions called    : at91_pio_write
//*-----------------------------------------------------------------------------
void aic_software_interrupt(void)
{

//  readval( &value );
//  if (value==10) {
//      AT91F_PIO_ClearOutput( AT91C_BASE_PIO, LED8 );
//  }

//* Read the output state
if ( (AT91F_PIO_GetInput(AT91C_BASE_PIO) & LED3 ) == LED3 )
   {
      AT91F_PIO_ClearOutput( AT91C_BASE_PIO, LED3 );
   }
else
   {
      AT91F_PIO_SetOutput( AT91C_BASE_PIO, LED3 );
   }
}

//*-----------------------------------------------------------------------------
//* Function Name       :pio_c_irq_handler
//* Object              :Irq Handler called by the irq_pio.s
//* Input Parameters    : none
//* Output Parameters   : none
//* Functions called    : at91_pio_read, at91_pio_write
//*-----------------------------------------------------------------------------
void pio_c_irq_handler ( void )
{
int dummy;
//* Read the output state
if ( (AT91F_PIO_GetInput(AT91C_BASE_PIO) & LED5 ) == LED5 )
   {
     AT91F_PIO_ClearOutput( AT91C_BASE_PIO, LED5);
   }
else
   {
       AT91F_PIO_SetOutput( AT91C_BASE_PIO, LED5);
```

```
    }
//* enable the next PIO IRQ
dummy =AT91C_BASE_PIO->PIO_ISR;
//* suppress the compilation warning
dummy =dummy;
//* while SW4 is push wait
// while ( (AT91F_PIO_GetInput(AT91C_BASE_PIO) & SW4_MASK ) != SW4_MASK );
}

//*-----------------------------------------------------------------------
//* Function Name       : delay
//* Object              : Wait
//* Input Parameters    : none
//* Output Parameters   : none
//* Functions called    : none
//*-----------------------------------------------------------------------


void delay ( void )
{
  unsigned int i ;

   //* loop delay
   for ( i = 0 ;(i<4000000 );i++ ) ;

}

void seconddelay ( void );
void seconddelay ( void )
{
   unsigned int i ;

   //* loop delay
   for ( i = 0 ;(i<1000000 );i++ ) ;

}


void decipher(void)
{
 for(q=0;q<i;q++)
   {
    printf(" %x",state2[q]);
    }
 printf("\n") ;
 eingabe=1;
 crypto();
 eingabe=7;
 q=0;
```

```
  i=0;
}



void fillstate2(void)
{
  state2[i]=j;
  i++;

}




int crypto(void)

{
        const uint8_t sBox4[] =    {

        0xc,0x5,0x6,0xb,0x9,0x0,0xa,0xd,0x3,0xe,0xf,0x8,0x4,0x7,0x1,0x2
                                                        };
        const uint8_t invsBox4[] = {

        0x5,0xe,0xf,0x8,0xc,0x1,0x2,0xd,0xb,0x4,0x6,0x3,0x0,0x7,0x9,0xa
                                                          };
//      Input values
        uint8_t key[] =     {0x15,0x3f,0x04,0x70,0x40,0xa4,0xd5,0x04,0x70,0x20};
        volatile uint8_t state[8];




//      Counter
        uint8_t i = 0;
//      pLayer variables
        uint8_t position = 0;
        uint8_t element_source = 0;
        uint8_t bit_source = 0;
        uint8_t element_destination         = 0;
        uint8_t bit_destination     = 0;
        uint8_t temp_pLayer[8];
//      Key scheduling variables
        uint8_t round;
        uint8_t save1;
        uint8_t save2;
        uint8_t subkey[32][8];
        //volatile inteingabe=0;
//      ******************** Encryption ***************************
```

```
        AT91PS_USART COM0;
        COM0= AT91C_BASE_US1;


      if (eingabe == 0)
      {
      for(i=0;i<8;i++)
      {
      state[i]=state3[i];

      }

      round=0;
      do
      {
//      ******************* addRoundkey *************************
        i=0;
        do
        {
                state[i] = state[i] ^ key[i+2];
                i++;
        }
        while(i<=7);
//      ******************* sBox ********************************
        do
        {
                i--;
                state[i] = sBox4[state[i]>>4]<<4 | sBox4[state[i] &0xF];
        }
        while(i>0);
//      ******************* pLayer ******************************
        for(i=0;i<8;i++)                                        //clearing of the temporary array
temp_pLayer
        {
                temp_pLayer[i] = 0;
        }
        for(i=0;i<64;i++)
        {
                position = (16*i) % 63;                         //arithmetic calculation of the pLayer
                if(i == 63)                                                     //exception for bit 63
                        position = 63;
                element_source          = i / 8;
                bit_source                      = i % 8;
                element_destination     = position / 8;
                bit_destination  = position % 8;
                temp_pLayer[element_destination] |= ((state[element_source]>>bit_source) &0x1)
<<bit_destination;
```

```
        }
        for(i=0;i<=7;i++)
        {
                state[i] = temp_pLayer[i];
        }
//      ******************** End pLayer ***************************
//      ******************** Key Scheduling ***********************
//        on-the-fly key generation
        save1  = key[0];
        save2  = key[1];
        i = 0;
        do
        {
                key[i] = key[i+2];
                i++;
        }
        while(i<8);
        key[8] = save1;                                         //61-bit left shift
        key[9] = save2;
        i = 0;
        save1 = key[0] &7;
        do
        {
                key[i] = key[i] >>3 | key[i+1] <<5;
                i++;
        }
        while(i<9);
        key[9] = key[9] >>3 | save1 <<5;

        key[9] = sBox4[key[9]>>4]<<4 | (key[9] &0xF);//S-Box application

        if((round+1) % 2 == 1)                                  //round counter addition
                key[1] ^= 128;
        key[2] = ((((round+1)>>1) ^ (key[2] &15)) | (key[2] &240));
//      ******************** End Key Scheduling ******************
        round++;

      }
      while(round<31);
//    ******************** addRoundkey ***************************
      i = 0;
      do                                                        //final key XOR
      {
        state[i] = state[i] ^ key[i+2];
        i++;
      }
      while(i<=7);
```

```
        for(q=0;q<=7;q++)
        {
        printf("%x",state[q]);
        }
        printf("\n");
forSend[1] = 2;
      AT91F_US_SendFrame(COM0,(char *)forSend,sizeof(forSend));
timer_init();
q=0;
timingflag=0;
while(count_timer1_interrupt<9)

      {
if((count_timer1_interrupt % 1)==0&& count_timer1_interrupt !=timingflag)
         {
forSend[1] = state[q];
            AT91F_US_SendFrame(COM0,(char *)forSend,sizeof(forSend));
q++;
timingflag= count_timer1_interrupt;

         }

      }
      AT91C_BASE_TC1->TC_CCR = AT91C_TC_CLKDIS ;
      AT91C_BASE_TC0->TC_CCR = AT91C_TC_CLKDIS ;
      AT91F_PIO_SetOutput( AT91C_BASE_PIO, LED7 );
      AT91F_PIO_SetOutput( AT91C_BASE_PIO, LED8 );

        eingabe = 7;
        return 0;
        }
//      ****************** End addRoundkey **********************
//      ****************** End Encryption  ***********************




//      ****************** Decryption ****************************
        if(eingabe == 1)
        {


        for(i=0;i<8;i++)
        {
        state[i]=state2[i];
        }
```

```
//      ****************** Key Scheduling **********************
//      keyprecomputation
        for(i=2;i<=9;i++)
        {
          subkey[0][i-2] = key[i];
        }
        do
        {
          i=0;
          save1  = key[0];
          save2  = key[1];
          i = 0;
          do
          {
                    key[i] = key[i+2];
                    i++;
          }
          while(i<8);
          key[8] = save1;
          key[9] = save2;
          i = 0;
          save1 = key[0] &7;                                        //61-bit left shift
          do
          {
                    key[i] = key[i] >>3 | key[i+1] <<5;
                    i++;
          }
          while(i<9);
          key[9] = key[9] >>3 | save1 <<5;

          key[9] = sBox4[key[9]>>4]<<4 | (key[9] &0xF);//S-Box application

          if((round+1) % 2 == 1)                                    //round counter addition
                    key[1] ^= 128;
          key[2] = ((((round+1)>>1) ^ (key[2] &15)) | (key[2] &240));

          for(i=2;i<=9;i++)                                         //Subkey storage
          {
                    subkey[round+1][i-2] = key[i];
          }

          round++;
        }
        while(round<31);

//      ****************** End Key Scheduling ******************
        do
        {
```

```
//      ****************** addRoundkey ************************
        i=0;
        do
        {
                state[i] = state[i] ^ subkey[round][i];
                temp_pLayer[i] = 0;
                i++;
        }
        while(i<=7);
//      ****************** End addRoundkey ********************
//      ****************** pLayer *****************************
        for(i=0;i<64;i++)
        {
                position = (4*i) % 63;                          //arithmentic calculation of the pLayer
                if(i == 63)                                                     //exception for bit 63
                        position = 63;
                element_source = i / 8;
                bit_source = i % 8;
                element_destination = position / 8;
                bit_destination = position % 8;
                temp_pLayer[element_destination] |= ((state[element_source]>>bit_source) &0x1)
<<bit_destination;
        }
        for(i=0;i<=7;i++)
        {
                state[i] = temp_pLayer[i];
        }
//      ****************** End pLayer *************************
//      ****************** sBox *******************************
        i=0;
        do
        {
                state[i] = invsBox4[state[i]>>4]<<4 | invsBox4[state[i] &0xF];
                i++;
        }
        while(i<=7);



//      ****************** End sBox ***************************
        round--;


      }
      while(round>0);
//      ****************** addRoundkey ************************
      i = 0;
      do                                                                        //final key XOR
      {
```

```
            state[i] = state[i] ^ subkey[0][i];
            i++;
          }
          while(i<=7);

          }
//        ****************** End addRoundkey **********************
//        ****************** End Decryption ***********************



for(q=0;q<8;q++)
          {
          printf(" %x",state[q]);
          }
          printf("\n");

return0;


}


int simplesend(void)
{
   AT91PS_USART COM0;
   COM0= AT91C_BASE_US1;
delay();
forSend[1] = 3;
   AT91F_US_SendFrame(COM0,(char *)forSend,sizeof(forSend));
timer_init();
q=0;
timingflag=0;
while(count_timer1_interrupt<9)

   {


if((count_timer1_interrupt % 1)==0&& count_timer1_interrupt !=timingflag)
       {
forSend[1] = state3[q];
          AT91F_US_SendFrame(COM0,(char *)forSend,sizeof(forSend));
q++;
timingflag= count_timer1_interrupt;

       }

   }
   AT91C_BASE_TC1->TC_CCR = AT91C_TC_CLKDIS ;
```

```
    AT91C_BASE_TC0->TC_CCR = AT91C_TC_CLKDIS ;
    AT91F_PIO_SetOutput( AT91C_BASE_PIO, LED7 );
    AT91F_PIO_SetOutput( AT91C_BASE_PIO, LED8 );


return 0;

}




int recieveblok(void)
{
 AT91PS_USART COM0;

 COM0= AT91C_BASE_US1;

timer_init();
for(;;)
 {

   if((count_timer1_interrupt % 1)==0&& count_timer1_interrupt!=timingflag)
    {
      seconddelay();
      AT91F_US_ReceiveFrame(COM0,(char *)my_eol,sizeof(my_eol));
       j=(int)my_eol[1];
       fillstate2();
       timingflag=count_timer1_interrupt;



     }

if(i==8)
      {
       if(decflag==0)
          {
            for(q=0;q<i;q++)
               {
               printf(" %x",state2[q]);
               }
         printf("\n") ;
         i=0;
          }
       AT91C_BASE_TC1->TC_CCR = AT91C_TC_CLKDIS ;
       AT91C_BASE_TC0->TC_CCR = AT91C_TC_CLKDIS ;
```

```
        AT91F_PIO_SetOutput( AT91C_BASE_PIO, LED7 );
        AT91F_PIO_SetOutput( AT91C_BASE_PIO, LED8 );

      return 0;

       }
    }
}



//*-----------------------------------------------------------------------
//* Function Name       : main
//* Object              : Main interrupt function
//*        level timer 0 => 1
//*      SW2       level Irq0    => 2
//*        level timer 1 => 4
//*      SW4       level PIOA    => 6
//*        level USART   => 7
//*        LEVEL FIQ     => MAX
//* Input Parameters    : none
//* Output Parameters   : TRUE
//*-----------------------------------------------------------------------
int main( void )
//* Begin
{
    int loop_count ;
   AT91PS_USART COM0;
   AT91PS_AIC     pAic;
   // Load System pAic Base address
   pAic = AT91C_BASE_AIC;
   COM0= AT91C_BASE_US1;
   //* Init
    loop_count = 0 ;
    // First, enable the clock of the PIOB
    AT91F_PS_EnablePeriphClock ( AT91C_BASE_PS, 1<<AT91C_ID_PIO ) ;
    // then, we configure the PIO Lines corresponding to LED1 to LED8
    // to be outputs. No need to set these pins to be driven by the PIO because it is GPIO pins  only.
     AT91F_PIO_CfgOutput( AT91C_BASE_PIO, LED_MASK ) ;
    // Clear the LED's. On the EB55 we must apply a "1" to turn off LEDs
    AT91F_PIO_SetOutput( AT91C_BASE_PIO, LED_MASK ) ;
     //* open  FIQ interrupt
     AT91F_AIC_ConfigureIt ( pAic, AT91C_ID_FIQ,
FIQ_INTERRUPT_LEVEL,AT91C_AIC_SRCTYPE_INT_EDGE_TRIGGERED, fiq_asm_handler);
      AT91F_AIC_EnableIt (pAic, AT91C_ID_FIQ);

    //* InitUsart
    Usart_init();
```

```
        //* generate software interrupt
        AT91F_AIC_Trig (pAic,AT91C_ID_SYS) ;

       for (;;)
          {
            AT91F_PIO_ClearOutput( AT91C_BASE_PIO, LED1 );
            delay () ;
            AT91F_PIO_SetOutput( AT91C_BASE_PIO, LED1 );
            delay () ;
            if((AT91F_PIO_GetInput(AT91C_BASE_PIO)&SW3_MASK)==0)
               {
                simplesend();
               }
           if((AT91F_PIO_GetInput(AT91C_BASE_PIO)&SW4_MASK)==0)
               {
               eingabe=0;
               crypto();
               eingabe=7;
               }

           AT91F_US_ReceiveFrame(COM0,(char *)my_eol,sizeof(my_eol));
            j=(int)my_eol[1];
            if(j==2)
               {
                decflag=1;
                recieveblok();
                }
            if(j==3)
                {
                decflag=0;
                recieveblok();
                }
            if(i==8)
                {
               decipher();
                }
       }
//* End
}
```

```
//* File Name           :interrupt_timer.c


// Include Standard LIB  files
#include"eb40A.h"
#include"stdio.h"

externvoid timer0_asm_irq_handler(void);
externvoid timer1_asm_irq_handler(void);

//* Global variable
externint count_timer0_interrupt;
externint count_timer1_interrupt;

#define TIMER0_INTERRUPT_LEVEL                  1
#define TIMER1_INTERRUPT_LEVEL                  4

/*-----------------*/
/* Clock Selection */
/*-----------------*/
#define TC_CLKS                 0x7
#define TC_CLKS_MCK2            0x0
#define TC_CLKS_MCK8            0x1
#define TC_CLKS_MCK32           0x2
#define TC_CLKS_MCK128          0x3
#define TC_CLKS_MCK1024         0x4

//*------------------------- Internal Function ------------------------------
//*--------------------------------------------------------------------------
//* Function Name       : AT91F_TC_Open
//* Object              : Initialize Timer Counter Channel and enable is clock
//* Input Parameters    :<tc_pt> = TC Channel Descriptor Pointer
//*                      <mode> = Timer Counter Mode
//*                     :<TimerId> = Timer peripheral ID definitions
//* Output Parameters   : None
//*--------------------------------------------------------------------------
void AT91F_TC_Open ( AT91PS_TC TC_pt, unsigned int Mode, unsigned int TimerId)
//* Begin
{
    unsigned int dummy;

     //* First, enable the clock of the TIMER
     AT91F_PS_EnablePeriphClock ( AT91C_BASE_PS, 1<<TimerId ) ;

     //* Disable the clock and the interrupts
      TC_pt->TC_CCR = AT91C_TC_CLKDIS ;
      TC_pt->TC_IDR = 0xFFFFFFFF ;
```

```
    //* Clear status bit
    dummy = TC_pt->TC_SR;
    //* Suppress warning variable "dummy" was set but never used
    dummy = dummy;
    //* Set the Mode of the Timer Counter
    TC_pt->TC_CMR = Mode ;

    //* Enable the clock
     C_pt->TC_CCR = AT91C_TC_CLKEN ;
    //* End
}

//*------------------------- Interrupt Function -----------------------------

//*----------------------------------------------------------------------------
//* Function Name       : timer0_c_irq_handler
//* Object              : C handler interrupt function called by the interrupts
//*                     assembling routine
//* Input Parameters    :<RTC_pt> time rtc descriptor
//* Output Parameters   : increment count_timer0_interrupt
//*----------------------------------------------------------------------------
void timer0_c_irq_handler(AT91PS_TC TC_pt)
{
  unsigned int dummy;
  //* Acknowledge interrupt status
  dummy = TC_pt->TC_SR;
  //* Suppress warning variable "dummy" was set but never used
  dummy = dummy;
  count_timer0_interrupt++;
/   /* Read the output state
  If ( (AT91F_PIO_GetInput(AT91C_BASE_PIO) & LED8 ) == LED8 )
     {
      AT91F_PIO_ClearOutput( AT91C_BASE_PIO, LED8 );
     }
 else
     {
      AT91F_PIO_SetOutput( AT91C_BASE_PIO, LED8 );
     }
}
//*----------------------------------------------------------------------------
//* Function Name       : timer1_c_irq_handler
//* Object              : C handler interrupt function called by the interrupts
//*                     assembling routine
//* Input Parameters    :<RTC_pt> time rtc descriptor
//* Output Parameters   : increment count_timer1_interrupt
//*----------------------------------------------------------------------------
void timer1_c_irq_handler(AT91PS_TC TC_pt)
{
```

```
  unsigned int dummy;
  //* Acknowledge interrupt status
  dummy = TC_pt->TC_SR;
  //* Suppress warning variable "dummy" was set but never used
  dummy = dummy;
  count_timer1_interrupt++;

  //* Read the output state
  if ( (AT91F_PIO_GetInput(AT91C_BASE_PIO) & LED7 ) == LED7 )
      {
       AT91F_PIO_ClearOutput( AT91C_BASE_PIO, LED7 );
      }
  else
      {
       AT91F_PIO_SetOutput( AT91C_BASE_PIO, LED7 );
      }
}
//*--------------------------- External Function ------------------------------

//*-----------------------------------------------------------------------------
//* Function Name       :timer_init
//* Object              :Init timer counter
//* Input Parameters    : none
//* Output Parameters   : TRUE
//*-----------------------------------------------------------------------------
void timer_init ( void )
//* Begin
{
  //init the timer interrupt counter
  count_timer0_interrupt=0;
  count_timer1_interrupt=0;

  //* Open timer0
  AT91F_TC_Open(AT91C_BASE_TC0,TC_CLKS_MCK2,AT91C_ID_TC0);

  //* Open Timer 0 interrupt
  AT91F_AIC_ConfigureIt ( AT91C_BASE_AIC, AT91C_ID_TC0,
TIMER0_INTERRUPT_LEVEL,AT91C_AIC_SRCTYPE_INT_LEVEL_SENSITIVE,
timer0_asm_irq_handler);
  AT91C_BASE_TC0->TC_IER = AT91C_TC_CPCS;  //  IRQ enable CPC
  AT91F_AIC_EnableIt (AT91C_BASE_AIC, AT91C_ID_TC0);

  //* Open timer1
   AT91F_TC_Open(AT91C_BASE_TC1,TC_CLKS_MCK128,AT91C_ID_TC1);

   //* Open Timer 1 interrupt
   AT91F_AIC_ConfigureIt ( AT91C_BASE_AIC, AT91C_ID_TC1,
TIMER1_INTERRUPT_LEVEL,AT91C_AIC_SRCTYPE_INT_LEVEL_SENSITIVE,
timer1_asm_irq_handler);
```

```
  AT91C_BASE_TC1->TC_IER  = AT91C_TC_CPCS;  //  IRQ enable CPC
  AT91F_AIC_EnableIt (AT91C_BASE_AIC, AT91C_ID_TC1);


  //* Generate interrupt by software
    AT91F_AIC_Trig (AT91C_BASE_AIC,AT91C_ID_TC0) ;
    AT91F_AIC_Trig (AT91C_BASE_AIC,AT91C_ID_TC1) ;
 //* Start timer0
     AT91C_BASE_TC0->TC_CCR = AT91C_TC_SWTRG ;

   //* Start timer1
     AT91C_BASE_TC1->TC_CCR = AT91C_TC_SWTRG ;

//* End
}
```

```
//* File Name           :interrupt_Usart.c




// Include Standard LIB  files
#include"eb40A.h"

extern void usart_asm_irq_handler(void);
#define USART_INTERRUPT_LEVEL           3

static const char atmel_header[]=
{
  " "

};


#define AT91C_US_USMODE_NORMAL       AT91C_US_CHMODE_NORMAL
//*------------------------ Internal Function ------------------------------
//*------------------------ Interrupt Function ------------------------------

//*-----------------------------------------------------------------------------
//* Function Name       :Usart_c_irq_handler
//* Object              : C handler interrupt function called by the interrupts
//*                     assembling routine
//* Input Parameters    :<RTC_pt> time rtc descriptor
//* Output Parameters   : increment count_timer0_interrupt
//*-----------------------------------------------------------------------------
void Usart_c_irq_handler(AT91PS_USART USART_pt)
{
        unsigned int status;
        //* getUsart status register
        status = USART_pt->US_CSR;
        if ( status & AT91C_US_RXRDY)
         {
           //* Get byte and send
          AT91F_US_PutChar (USART_pt, AT91F_US_GetChar(USART_pt));
         }

        if ( status & AT91C_US_OVRE)
         {
          //* clear US_RXRDY
           AT91F_US_GetChar(USART_pt);
           AT91F_US_PutChar (USART_pt, 'O');
         }
```

```
        //* Check error
        if ( status & AT91C_US_PARE)
         {
           AT91F_US_PutChar (USART_pt, 'P');
         }

        if ( status & AT91C_US_FRAME)
         {
           AT91F_US_PutChar (USART_pt, 'F');
         }

        if ( status & AT91C_US_TIMEOUT)
         {
          USART_pt->US_CR = AT91C_US_STTTO;
           AT91F_US_PutChar (USART_pt, 'T');
         }

        //* Reset the satus bit
        USART_pt->US_CR = AT91C_US_RSTSTA;
}
//*--------------------------- External Function ------------------------------

//*-----------------------------------------------------------------------------
//* Function Name       :Usart_init
//* Object              : USART initialization
//* Input Parameters    : none
//* Output Parameters   : TRUE
//*-----------------------------------------------------------------------------
void Usart_init ( void )
//* Begin
{
        AT91PS_USART COM0;

        COM0= AT91C_BASE_US1;

//* Define RXD and TXD as peripheral
      AT91F_PIO_CfgPeriph(AT91C_BASE_PIO,AT91C_P22_RXD1 | AT91C_P21_TXD1,0);

        // First, enable the clock of the PIOB
        AT91F_PS_EnablePeriphClock ( AT91C_BASE_PS, 1<<AT91C_ID_US1 ) ;

        // Usart Configure
        AT91F_US_Configure (COM0, MCK,AT91C_US_ASYNC_MODE, 115200, 0);

        // Enable usart
        COM0->US_CR = AT91C_US_RXEN | AT91C_US_TXEN;

        //* Enable USART IT error and RXRDY
        AT91F_US_EnableIt(COM0,AT91C_US_TIMEOUT | AT91C_US_FRAME | AT91C_US_OVRE
```

```
|AT91C_US_RXRDY);

        //* openUsart 1 interrupt
        AT91F_AIC_ConfigureIt ( AT91C_BASE_AIC, AT91C_ID_US1,
USART_INTERRUPT_LEVEL,AT91C_AIC_SRCTYPE_INT_LEVEL_SENSITIVE, usart_asm_irq_handler);
        AT91F_AIC_EnableIt (AT91C_BASE_AIC, AT91C_ID_US1);

// Wait for USART CLOCK ready
        while ( !( COM0->US_CSR & AT91C_US_TXRDY ));
        AT91F_US_PutChar (COM0,'.');
        AT91F_US_SendFrame(COM0,(char *)atmel_header,sizeof(atmel_header));
//* End
}
```