ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΣΕΡΡΩΝ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μελέτη των επεξεργαστών 32 bit της εταιρίας Microchip Technology και ανάπτυξη σειράς υποδειγματικών εφαρμογών

ΤΑΣΙΟΥ ΕΥΘΥΜΙΑ (1563)

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: Δρ.ΚΑΛΟΜΟΙΡΟΣ ΙΩΑΝΝΗΣ ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ

Σέρρες 2012

ΤΙΤΛΟΣ

Μελέτη των επεξεργαστών 32 bit της εταιρίας Microchip Technology και ανάπτυξη σειράς υποδειγματικών εφαρμογών

ΠΕΡΙΛΗΨΗ

Γίνεται μελέτη της αρχιτεκτονικής των 32-bit μικροεπεξεργαστών της εταιρίας Microchip-Technology και συγκεκριμένα του PIC32MX795F512L. Αρχικά, γίνεται αναφορά στην αρχιτεκτονική του επεξεργαστή, η οποία υπακούει στα πρότυπα της αρχιτεκτονικής Von-Newmann και MIPS των επεξεργαστών RISC. Περιγράφεται η μνήμη του επεξεργαστή και γίνεται αναφορά στη μνήμη Cache. Επίσης, γίνεται αναφορά στην διαχείριση περιφερειακών συσκευών όπως διαχείριση I/O (PORTS), διαχείριση χρονισμού (timers), διαχείριση σημάτων διακοπής. Επίσης, η μελέτη αναφέρεται και στην διαχείριση επικοινωνίας και συγκεκριμένα στα κανάλια επικοινωνίας που υποστηρίζονται από τον PIC32 (USB, ETHERNET, σύγχρονη και ασύγχρονη επικοινωνία),
Δημιουργείται μία σειρά εφαρμογών που μελετά την δυνατότητα για ψηφιακή είσοδο/έξοδο, χρονισμό, διακοπές, USB επικοινωνία και σειριακή επικοινωνία (σύγχρονη και ασύγχρονη).
Χρησιμοποιείται ο C Compiler του περιβάλλοντος MPLAB καθώς και οι κατάλληλες βιβλιοθήκες εφαρμογών.

ΠΡΟΛΟΓΟΣ

Η πτυχιακή αυτή εργασία έχει ως θέμα τη μελέτη των επεξεργαστών 32bit της εταιρίας Microchip Technology και την ανάπτυξη μιας σειράς εφαρμογών. Σκοπός της συγκεκριμένης εργασίας είναι η εγκατάσταση των εργαλείων λογισμικού, η μελέτη και της βασικής αρχιτεκτονικής της οικογένειας των μικροεπεξεργαστών, καθώς και ορισμένες πρακτικές εφαρμογές.

Στο πρώτο μέρος αναπτύσσεται το θεωρητικό κομμάτι του συστήματος. Η αρχιτεκτονική που χρησιμοποιεί ο επεξεργαστής PIC32MX795F512L βασίζεται στην αρχιτεκτονική VON-Newmann. Ο πυρήνας MIPS βρίσκεται στην καρδιά του PIC32. Επιπλέον, το σύστημα χρησιμοποιεί την μνήμη Cache για την βελτίωση της ταχύτητάς του. Η μνήμη του μικροελεγκτή είναι μία ολοκληρωμένη συσκευή που προσφέρει διαφορετικά χαρακτηριστικά σε σχέση με τους 8-bit και 16-bit μικροελεγκτές. Τέλος, για το χρονισμό των περιφερειακών συσκευών χρησιμοποιείται το peripheral bus clock.

Στη συνέχεια, η διαχείριση της εισόδου και της εξόδου γίνεται μέσω των θυρών και συγκεκριμένα της PORTD. Οι χρονιστές timer1 και timer2 μπορούν να χρησιμοποιηθούν για τον έλεγχο χρονικά κρίσιμων διεργασιών. Τέλος σημαντικό ρόλο παίζουν τα σήματα διακοπών, που αποτελούν τον πιο αποδοτικό τρόπο χρήσης της Κεντρικής Μονάδας Επεξεργασίας (CPU). Η αρχιτεκτονική του PIC32 περιέχει ένα πλούσιο σύστημα διακοπών που μπορεί να διαχειριστεί μέχρι 64 διακοπές.

Σχετικά με την διαχείριση της επικοινωνίας τα κανάλια που υποστηρίζονται είναι η σύγχρονη και η ασύγχρονη σειριακή επικοινωνία, η USB και ο δίαυλος ETHERNET.

Στο δεύτερο μέρος αναπτύσσεται μια σειρά εφαρμογών. Αρχικά, γίνεται περιγραφή των αναπτυξιακών κυκλωμάτων που χρησιμοποιούνται, τα οποία είναι το Ethernet Starter Kit και ο Explorer16. Για τον προγραμματισμό του επεξεργαστή πάνω στο κύκλωμα (in-circuit) χρησιμοποιούμε το σύστημα PICK it 3.

Επίσης, παρουσιάζεται κώδικας για τον προγραμματισμό των χρονιστών και του συστήματος ελέγχου των διακοπών. Ο κώδικας παρουσιάζεται με τις απαραίτητες επεξηγήσεις.

Στη συνέχεια υλοποιούνται εφαρμογές που παρουσιάζουν τον προγραμματισμό της UART για ασύγχρονη σειριακή επικοινωνία, καθώς και κώδικας για τη βασική λειτουργία της σύγχρονης σειριακής επικοινωνίας με βάση τον δίαυλο SPI.

Τέλος, στο τελευταίο κεφάλαιο γίνεται η παράθεση των συμπερασμάτων.

Η εργασία πραγματοποιήθηκε στον τομέα Αρχιτεκτονικής Υπολογιστών και Βιομηχανικών Εφαρμογών του Τμήματος Πληροφορικής και Επικοινωνιών του ΤΕΙ Σερρών.

ΠΕΡΙΕΧΟΜΕΝΑ

ΜΕΡΟΣ Α

ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ , ΠΙΝΑΚΩΝ ΚΑΙ ΕΙΚΟΝΩΝ

ΜΕΡΟΣ Α

ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

ΚΕΦΑΛΑΙΟ 1^{ο}

ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΗ PIC32

1.1 Τι είναι ένας μικροελεγκτής

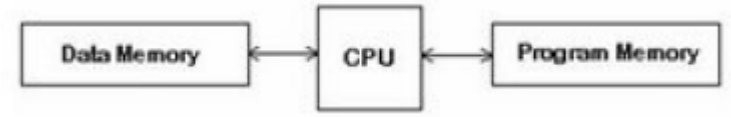
Ένας μικροελεγκτής είναι ένα μικρό υπολογιστικό κύκλωμα, σχεδιασμένο σ' ένα μόνο ολοκληρωμένο κύκλωμα υψηλής κλίμακας ολοκλήρωσης. Όπως κάθε υπολογιστικό κύκλωμα περιέχει και αυτό κεντρική μονάδα επεξεργασίας, έναν αριθμό καταχωρητών, κυκλώματα μνήμης και κυκλώματα ελέγχου περιφερειακών συσκευών. Ακόμη περιέχει μέσα στο μοναδικό ολοκληρωμένο κύκλωμα έναν αριθμό από καταχωρητές ειδικού σκοπού(συσσωρευτή, καραχωρητή κατάστασης, μετρητή προγράμματος, καταχωρητή εντολών, καταχωρητή δείκτη, εσωτερικούς χρονιστές-απαριθμητές, αριθμητική και λογική μονάδα (ALU), μονάδα αποκωδικοποίησης εντολών. Αποτελείται από κάποια βασικά στοιχεία όπως την μνήμη προγράμματος (ROM ή EPROM) και την μνήμη καταχωρητών / μεταβλητών (RAM).Μπορούμε να διακρίνουμε τα κυκλώματα χρονισμού και ελέγχου και τέλος τα βασικά μέρη ενός μικροελεγκτή που είναι οι παράλληλες θύρες και τα άλλα περιφερειακά κυκλώματα (UART, A/D μετατροπείς).

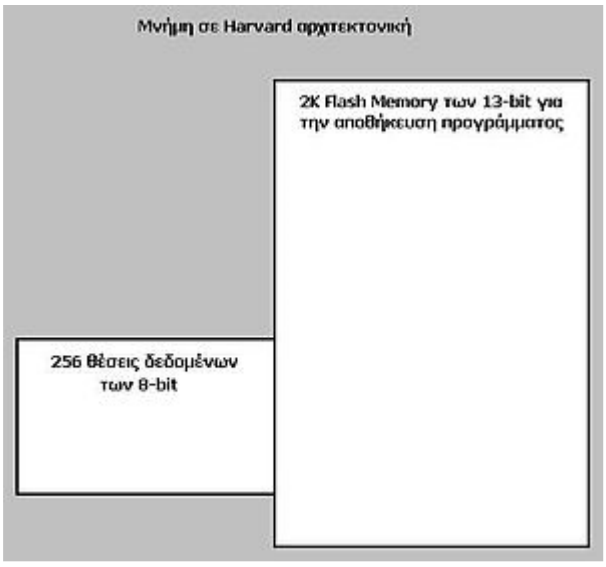
Οι θύρες I/O ενός μικροελεγκτής μπορεί να δέχονται σήματα εισόδου με την μορφή λογικών ψηφιακών καταστάσεων, χαρακτήρες ή bytes δεδομένων με την τεχνική της ασύγχρονης η σύγχρονης σειριακής επικοινωνίας, σήματα διακοπών , ή σε ορισμένες περιπτώσεις και αναλογικά σήματα, τα οποία στην συνέχεια μετατρέπονται σε ψηφιακά. Επίσης μπορεί να αποστέλλει σήματα σε άλλες συσκευές μέσα από θύρες εξόδου, να οδηγεί ηλεκτρονόμους, διόδους LED και άλλα κατάλληλα κυκλώματα, που συνήθως περιλαμβάνονται σε κάθε μορφής αυτοματισμό.

Τέλος, ένας μικροελεγκτής εκτελεί μια συγκεκριμένη λογική ακολουθία εντολών, οι οποίες έχουν καταχωρηθεί στην προγραμματιζόμενη μόνιμη μνήμη του. Κάθε φορά που θα επανεκκινείται ο μικροελεγκτής, θα εκτελεί τη ίδια λογική. Θα ανακαλεί τα δεδομένα, θα τα επεξεργάζεται και με βάση τα αποτελέσματα της επεξεργασίας θα ελέγχει το περιβάλλον του. Πρόκειται για ένα σύστημα ειδικού σκοπού αφιερωμένο στον έλεγχο και την εξυπηρέτηση ενός συγκεκριμένου αυτοματισμού. Παρακάτω αναφέρονται οι βασικές αρχιτεκτονικές που συναντάμε σε μικροεπεξεργαστές και σε μικροελεγκτές.

1.2 Αρχιτεκτονική Harvard-Von Newmann

Η αρχιτεκτονική HARVARD είναι μια αρχιτεκτονική πολύ δημοφιλής στους μικροελεγκτές. Ο χώρος διευθύνσεων του προγράμματος είναι ανεξάρτητος από εκείνον των δεδομένων. Το μήκος των λέξεων στο χώρο του προγράμματος μπορεί να είναι διαφορετικό από το μήκος των λέξεων στο χώρο των δεδομένων. Οι εντολές μπορούν να κινούνται την ίδια χρονική στιγμή στους διαδρόμους, ενώ μπορεί να εκτελείται μία εντολή ενώ προσκομίζεται η επόμενη. Οι περισσότεροι μικροελεγκτές χρησιμοποιούν αυτή την αρχιτεκτονική, αλλά ο PIC32 χρησιμοποιεί την αρχιτεκτονική VON-Newman.

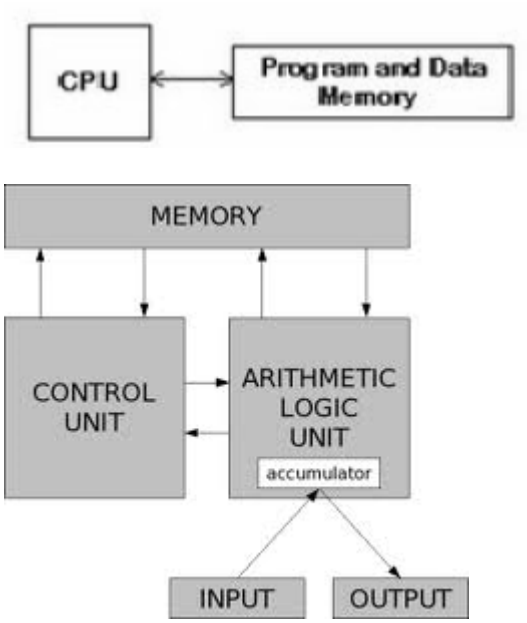




Σχήμα 1.1 Αρχιτεκτονική HARVARD

Αρχιτεκτονική VON NEWMANN

Σε αντίθεση, η μνήμη στην αρχιτεκτονική Von-Newmann μπορεί να χρησιμοποιηθεί τόσο για εντολές προγράμματος όσο και για δεδομένα σε μία εφαρμογή. Τα περισσότερα σύγχρονα υπολογιστικά συστήματα χρησιμοποιούν την αρχιτεκτονική Von-Newmann. Αν και οι περισσότεροι μικροελεγκτές χρησιμοποιούν την αρχιτεκτονική Harvard, ο PIC32 είναι ο πρώτος μικροελεγκτής της οικογένειας PIC που χρησιμοποιεί βασικά το μοντέλο Von-Neumann. Δηλαδή, διαθέτει ένα διάδρομο 32-bits τόσο για τα δεδομένα όσο και για τις εντολές του προγράμματος.



Σχήμα 1.2 Αρχιτεκτονική VON-Newmann

1.3 Αρχιτεκτονική MIPS

Ο μικροελεγκτής PIC32MX795F512L χρησιμοποιεί την αρχιτεκτονική MIPS. Ο πυρήνας MIPS βρίσκεται στην καρδιά του μικροελεγκτή. Παρακάτω αναφέρεται η λειτουργία της αρχιτεκτονικής MIPS, κάποιοι όροι που χρησιμοποιούνται και τα χαρακτηριστικά της.

Τον όρο ομοχειρία θα τον χρησιμοποιήσουμε παρακάτω για αυτό θα δούμε τι ακριβώς σημαίνει. Είναι μία τεχνική υλοποίησης στην οποία η εκτέλεση εντολών επικαλύπτεται. Ακόμη εκμεταλλεύεται τον παραλληλισμό που υπάρχει μεταξύ των διαφόρων ενεργειών που χρειάζεται να γίνουν για να εκτελεστεί μία εντολή. Σήμερα αυτή η τεχνική υλοποίησης χρησιμοποιείται για να φτιάξουμε γρήγορους επεξεργαστές.

Η αρχιτεκτονική MIPS είναι μία απλή αρχιτεκτονική φόρτωσης και αποθήκευσης των 64-bit. Βασίζεται στην χρήση καταχωρητών γενικού σκοπού, υποστηρίζει μεθόδους άμεσης και έμμεσης δευθυνσιοδότησης μέσω καταχωρητών και τέλος, υποστηρίζει τα μεγέθη και τους τύπους δεδομένων ακεραίων των 8-bit,16-bit, 32-bit και 64-bit.

Επιπλέον, υποστηρίζει απλές εντολές, όπως φόρτωσης, αποθήκευσης, πρόσθεσης, αφαίρεσης, μεταφοράς μεταξύ καταχωρητών και καταχωρητών ολίσθησης. Καθώς και τις εντολές σύγκρισης, διακλάδωσης, άλματος, κλήσης και επανόδου. Τέλος υπάρχουν τουλάχιστον 16 καταχωρητές γενικού σκοπού που εξασφαλίζουν όλες τις μεθόδους διευθυνσιοδότησης και εφαρμόζονται σε όλες τις εντολές μεταφοράς δεδομένων. Αυτό έχει ως στόχο την ελαχιστοποίηση του συνόλου εντολών. Επίσης, χρησιμοποιούνται ξεχωριστοί καταχωρητές κινητής υποδιαστολής για την αύξηση του συνολικού αριθμού των καταχωρητών χωρίς να δημιουργούνται προβλήματα στην μορφή των εντολών ή στην ταχύτητα των καταχωρητών γενικού σκοπού.

Η αρχιτεκτονική MIPS στους σύγχρονους υπολογιστές δίνει περισσότερο έμφαση σε ένα απλό σύνολο εντολών φόρτωσης και αποθήκευσης, στο σχεδιασμό για την αποτελεσματική ομοχειρία με σταθερή κωδικοποίηση του συνόλου των εντολών καθώς και την αποτελεσματική μετάφραση του μεταγλωττιστή.

Η αρχιτεκτονική MIPS έχει 32 καταχωρητές γενικού σκοπού General Purpose Registers (GPR) των 64-bit. Ακόμη υπάρχει ένα σύνολο 32 καταχωρητών κινητής υποδιαστολής Floating Point Registers (FPR), οι οποίοι μπορούν να αποθηκεύουν 32 τιμές απλής ακρίβειας (32 bit) ή 32 τιμές διπλής ακρίβειας (64 bit). Έχουμε πράξεις κινητής υποδιαστολής τόσο για την απλή όσο και για την διπλή ακρίβεια. Η αρχιτεκτονική MIPS περιλαμβάνει επίσης εντολές που ενεργούν με δύο τελεστές απλής ακρίβειας που βρίσκονται στον ίδιο καταχωρητή κινητής υποδιαστολής των 64 bit. Η τιμή του R0 είναι πάντα 0.

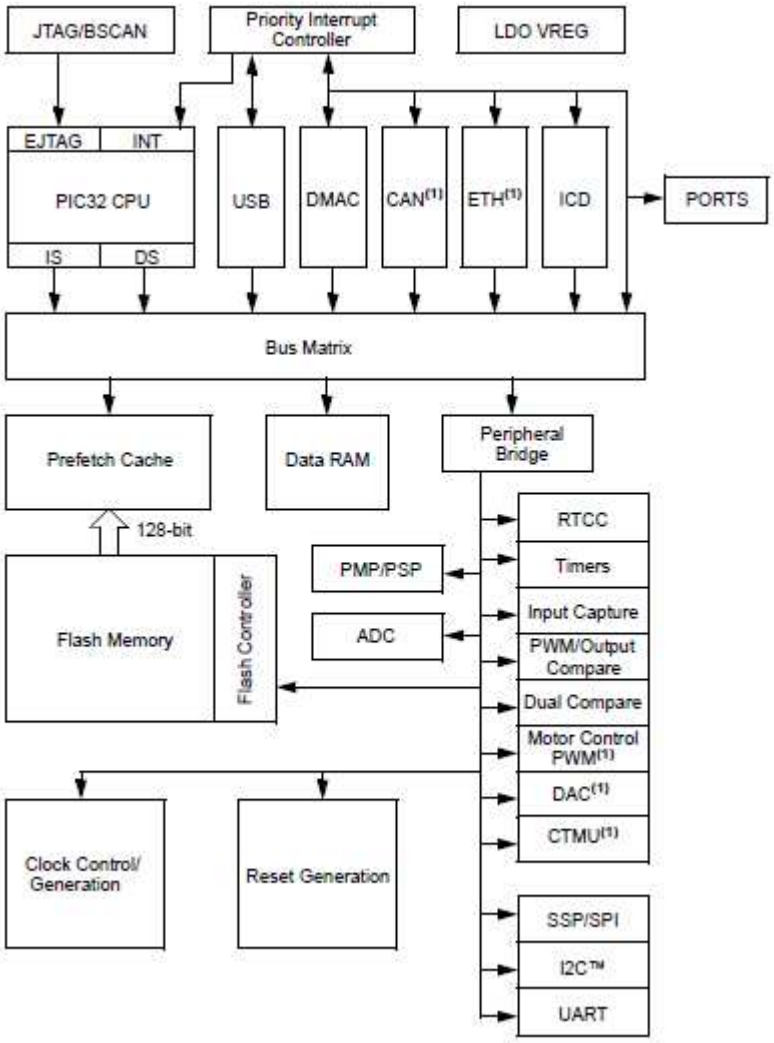
1.4 Αρχιτεκτονικές RISC

Ο μικροελεγκτής PIC32MX795F512L χρησιμοποιεί το μοντέλο αρχιτεκτονικής Reduced instruction set computer (RISC). Τα υπολογιστικά συστήματα που χρησιμοποιούν την αρχιτεκτονική RISC είναι ταχύτατα και μία εντολή υψηλού επιπέδου μπορεί να πραγματοποιηθεί με περισσότερες εντολές της γλώσσας μηχανής. Επιπλέον, κάποια από τα πλεονεκτήματα που έχει η RISC είναι η αφθονία καταχωρητών γενικής χρήσης, οι εντολές οι οποίες προσπελαύνουν την μνήμη είναι περιορισμένες, όλες οι εντολές έχουν το ίδιο μήκος και την ίδια μορφή έτσι οι εντολές κωδικοποιούνται ευκολότερα. Τέλος ο ρυθμός που υποβάλλονται οι εντολές μπορεί να μεγαλώνει καθώς εκτελούνται οι εντολές. Οι εντολές που χρησιμοποιούνται πιο συχνά εκτελούνται πιο γρήγορα, ενώ οι εντολές που δεν χρησιμοποιούνται τόσο συχνά εκτελούνται πιο αργά.

Τέλος, αξίζει να αναφερθούν κάποια από τα προβλήματα που υπάρχουν στην αρχιτεκτονική RISC. Δεν υπάρχει καθορισμένος τρόπος για την δοκιμή προγραμμάτων και η απόδοση του συστήματος ποικίλλει ανάλογα με το πρόγραμμα. Ένα ακόμη πρόβλημα είναι ότι διακρίνονται δύσκολα οι επιδράσεις που έχουν τα μηχανήματα σε σχέση με τις επιδράσεις που οφείλονται στον τρόπο που γράφουμε στον compiler.

1.5 Κεντρική μονάδα επεξεργασίας (CPU) του PIC32

Η οικογένεια PIC32 των συσκευών είναι σύνθετα συστήματα on-a-chip που περιέχουν πολλά χαρακτηριστικά. Σε όλους τους επεξεργαστές της οικογένειας PIC32 η CPU είναι υψηλής απόδοσης RISC, η οποία μπορεί να προγραμματιστεί σε 32-bit και 16-bit λειτουργίες. Οι συσκευές PIC32 περιέχουν υψηλής απόδοσης Interrupt controller. Ακόμη περιέχουν USB controller καθώς και DMA controller (Direct Memory Access). Τέλος, περιέχουν on-chip τις μνήμες RAM και EPROM που κρατούν τα δεδομένα και τα προγράμματα.

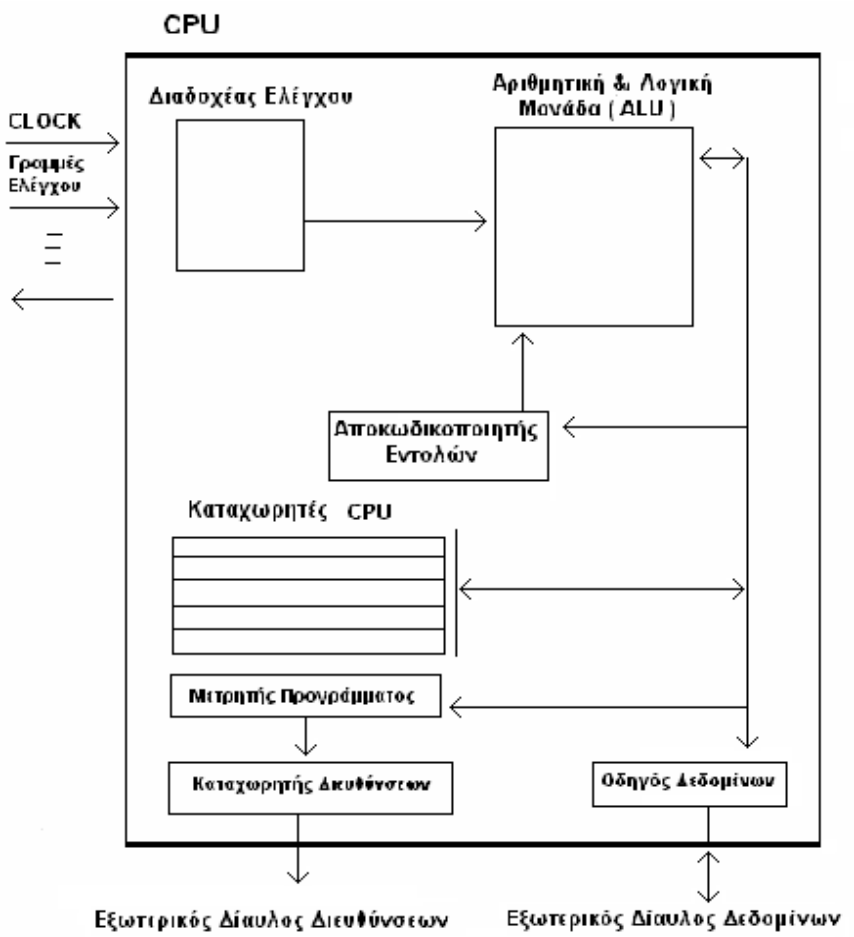


Σχήμα 1.3 Διάγραμμα της CPU

Υπάρχουν δύο εσωτερικά busses στη συσκευή του PIC32 για σύνδεση με όλα τα περιφερειακά. Το κύριο περιφερειακό bus συνδέει τις περισσότερες από τις περιφερειακές μονάδες στο matrix bus μέσω μιας περιφερειακής γέφυρας. Ανάλογα με την παραλλαγή της συσκευής, υπάρχει επίσης μία υψηλής ταχύτητας περιφερειακή γέφυρα που συνδέει τη διακοπή, το DMA, το Controller Area Network (CAN) και τους Ethernet controllers, με τα in-circuit κυκλώματα εντοπισμού σφαλμάτων και τα USB περιφερειακά. Η CPU εκτελεί εργασίες υπό τον έλεγχό του προγράμματος.

Η CPU του PIC32 βασίζεται σε μια αρχιτεκτονική load/store και εκτελεί περισσότερες λειτουργίες σε ένα σύνολο εσωτερικών καταχωρητών. Ειδικές εντολές load και store χρησιμοποιούνται για τη μεταφορά δεδομένων ανάμεσα σ΄αυτούς τους εσωτερικούς καταχωρητές και στον έξω κόσμο.

Τέλος μπορούμε να δούμε ένα μπλοκ διάγραμμα της CPU



Σχήμα 1.4 Block διάγραμμα της CPU

Επιπλέον, υπάρχουν δύο ξεχωριστά busses στον PIC32. Το ένα είναι το Bus matrix και το άλλο το Peripheral Bus. Επιπλέον, υπάρχουν και άλλα bus master στον PIC32 όπως είναι

- DMA controller
- In-Circuit-Debugger Unit
- USB controller
- CAN controller
- Ethernet controller

1.6 Η Μνήμη του Συστήματος

1.6.1 Memory

Η μνήμη είναι μια ολοκληρωμένη συσκευή του μικροελεγκτή. Η αυξημένη ευρωστία της και η ευκολία της καθώς και το σύνολο των περιφερειακών της είναι προσανατολισμένο προς το χρήστη. Το μέγεθος της

διαθέσιμης μνήμης (flash & RAM) τις περισσότερες φορές φαίνεται να καθορίζει το κόστος και τη διαθεσιμότητα του προϊόντος. Ο πυρήνας PIC32 προσφέρει μερικά χαρακτηριστικά που είναι διαφορετικά από την αρχιτεκτονική του 8-bit & 16-bit. Τα χαρακτηριστικά αυτά έχουν την δυνατότητα να αντιστοιχιστούν τους χώρους μνήμης στη μνήμη cache και την διαμοίραση της μνήμης bus με το μηχανισμό άμεσης πρόσβασης (DMA).

1.6.1.2 Ο PIC32MX Bus

Ο PIC32 ακολουθεί την αρχιτεκτονική Von-Neumann. Η μεγάλη διαφορά από την αρχιτεκτονική Harvard είναι ότι οι δύο τελείως διαφορετικοί διάδρομοι για τα δεδομένα και τις εντολές δεν είναι πλέον διαθέσιμοι. Ένα ενιαίο μεγάλο (32-bit) bus παρέχει πρόσβαση στην μνήμη προγράμματος (flash) και στα data από την μνήμη Ram. Η αρχιτεκτονική Von Neumann είναι πιο οικονομική και την ίδια στιγμή παρέχει ένα απλούστερο μοντέλο Έτσι, για πρώτη φορά ένας PIC εκτελεί κώδικα από την μνήμη RAM.

Λαμβάνοντας υπόψη την ίδια συχνότητα ρολογιού 20MHZ ο PIC32 μπορεί να εκτελέσει μέχρι και 4 φορές περισσότερες εντολές ανά δευτερόλεπτο σε σχέση με τις οικογένειες PIC16 και PIC18. Επίσης, σημαίνει ότι μπορεί να εκτελέσει 2 φορές τον αριθμό των εντολών ανά δευτερόλεπτο του PIC24. Έτσι έχουμε μια πραγματική αύξηση της υπολογιστικής ισχύος που παρέχει ο PIC 32.

1.6.1.3 Ο χάρτης της μνήμης του PIC32

Ο πυρήνας MIPS είναι στην καρδιά του PIC32 και έχει μια σειρά από χαρακτηριστικά που έχουν σχεδιαστεί για να επιτρέπουν τον διαχωρισμό του χώρου της μνήμης σε χώρο κατάλληλο για εφαρμογές χρήστη (user address space) και σε χώρο που προορίζεται για τις εφαρμογές λειτουργικού συστήματος (kernel address space). Έτσι, ο χάρτης των φυσικών διευθύνσεων (physical addresses) χωρίζεται σε δύο μέρη, στις διευθύνσεις πάνω από τη από τη διεύθυνση 0xBD000000 (kernel space) και στις διευθύνσεις κάτω από αυτήν (user space).

Μια μονάδα διαχείρισης του χάρτη της μνήμης (Memory Management Unit-MMU) αναλαμβάνει να μεταφέρει τις φυσικές διευθύνσεις στις λεγόμενες εικονικές διευθύνσεις (virtual addresses). Έτσι στους επεξεργαστές MIPS είναι σύνηθες να εμφανίζεται αυτός ο διαχωρισμός. Με βάση τον χάρτη της εικονικής μνήμης, οι δύο διαχωρισμένες περιοχές της μνήμης, για τις εφαρμογές και το λειτουργικό σύστημα έχουν ως εξής:

Το κάτω μέρος του χάρτη εικονικών διευθύνσεων 32 bit, από 0x00000000 μέχρι 0x80000000 προορίζεται για τις εφαρμογές χρήστη, ενώ από κει και πάνω για τις εφαρμογές λειτουργικού συστήματος.

Ο χάρτης της φυσικής και εικονικής μνήμης του PIC32 φαίνεται στο σχήμα.

	Memory Type	Virtual Addresses		Physical Addresses		Size in Bytes
		Begin Address	End Address	Begin Address	End Address	Calculation
Kernal Address Space	Boot Flash	0xBFC00000	0xBFC02FFF	0x1FC00000	0x1FC02FFF	12 KB
	Program Flash[1]	0xBD000000	0xBD000000 + BMXPUPBA − 1	0x1D000000	0x1D000000 + BMXPUPBA − 1	BMXPUPBA
	Program Flash[2]	0x9D000000	0x9D000000 + BMXPUPBA − 1	0x1D000000	0x1D000000 + BMXPUPBA − 1	BMXPUPBA
	RAM (Data)	0x80000000	0x80000000 + BMXDKPBA − 1	0x00000000	BMXDKPBA − 1	BMXDKPBA
	RAM (Prog)	0x80000000 + BMXDKPBA	0x80000000 + BMXDUDBA − 1	BMXDKPBA	BMXDUDBA − 1	BMXDUDBA − BMXDKPBA
	Peripheral	0xBF800000	0xBF8FFFFF	0x1F800000	0x1F8FFFFF	1 MB
User Address Space	Program Flash	0x7D000000 + BMXPUPBA	0x7D000000 + PFM Size − 1	0xBD000000 + BMXPUBPA	0xBD000000 + PFM Size − 1	PFM Size − BMXPUBPA
	RAM (Data)	0x7F000000 + BMXDUDBA	0x7F000000 + BMXDUPBA − 1	0xBF000000 + BMXDUDBA	0xBF000000 + BMXDUPBA − 1	BMXDUPBA − BMXDUDBA
	RAM (Prog)	0x7F000000 + BMXDUPBA	0x7F000000 + RAM Size[3] − 1	0xBF000000 + BMXDUPBA	0xBF000000 + RAM Size[3] − 1	DRM Size − BMXDUPBA

Πίνακας 1.1: Ο χάρτης της φυσικής και εικονικής μνήμης

Ο διαχωρισμός της μνήμης, όπως περιγράφηκε παραπάνω έχει σκοπό την ακεραιότητα των εφαρμογών και την προστασία των δεδομένων μας. Οι σχεδιαστές του PIC32 ήθελαν να βεβαιωθούν ότι θα είναι σε θέση να επιβάλλουν ειδικούς κανόνες για την προστασία και τα σφάλματα που απομονώνουν τις περιοχές μνήμης. Για παράδειγμα όταν εκτελείται ένα λειτουργικό σύστημα operator system (OS), θα πρέπει να εμποδίζεται ο κώδικας της εφαρμογής να αγγίξει τα δεδομένα της RAM, σε περιοχές που αποτελούν μέρος του λειτουργικού συστήματος. Με άλλα λόγια ο κώδικας του χρήστη πρέπει να μην επιτρέπει την πρόσβαση στα δεδομένα του πυρήνα.

Επειδή, όμως, σε πολλές περιπτώσεις, όπως συμβαίνει στις δικές μας εφαρμογές, δεν χρησιμοποιούμε λειτουργικό σύστημα (RTOS), οι παραπάνω διαχωρισμοί δεν ισχύουν, ο χρήσιμος χώρος της μνήμης είναι το kernel space, καθώς οι εφαρμογές μας, απουσία λειτουργικού συστήματος εκτελούνται σε kernel mode.

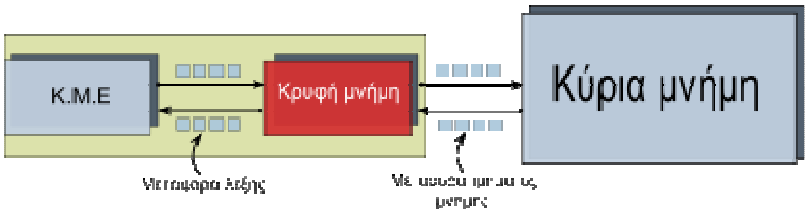
1.6.2 Η Μνήμη Cache

Κρυφή Μνήμη

Η μνήμη Cache του μικροελεγκτή PIC32MX795F512L είναι στα 256 byte. Οι μικροεπεξεργαστές είναι γενικά γρηγορότεροι από τις μνήμες με τις οποίες συνεργάζονται. Όταν η CPU διαβάζει ή γράφει στην μνήμη περιμένει αρκετούς κύκλους μηχανής μέχρι να ανταποκριθεί η μνήμη. Η διαφορά στην ταχύτητα που έχουν μεταξύ τους η CPU και η RAM μειώνει σημαντικά την συνολική απόδοση του συστήματος. Η τεχνική που χρησιμοποιούμε για την γεφύρωση του χάσματος ονομάζεται ΚΡΥΦΗ ΜΝΗΜΗ (CACHE MEMORY) .

Η CACHE MEMORY αποτελεί το πρώτο επίπεδο μνήμης σ' ένα υπολογιστικό σύστημα. Ο ρόλος της κρυφής μνήμης είναι να βελτιώνει την ταχύτητα ενός υπολογιστικού συστήματος. Ένα από τα πιο γνωστά

παραδείγματα κρυφής μνήμης είναι η κρυφή μνήμη του επεξεργαστή, η οποία μεσολαβεί μεταξύ της κεντρικής μνήμης RAM και της ΚΜΕ.



Σχήμα 1.5 Βασική οργάνωση ΚΜΕ-Κρυφής μνήμης-Κύριας Μνήμης

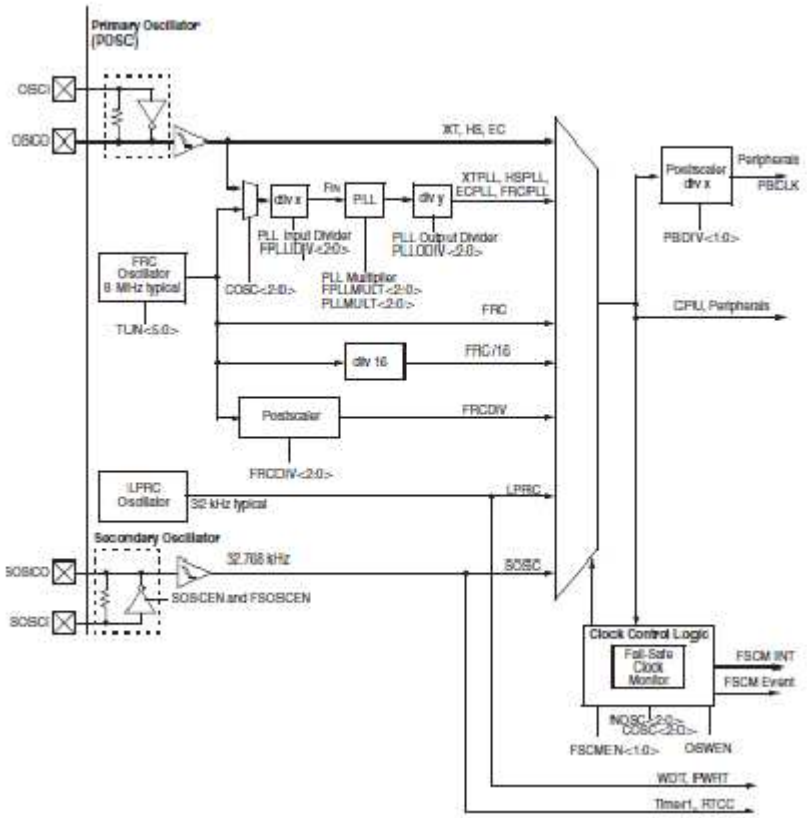
Άρα και σύμφωνα με τα παραπάνω η μνήμη CACHE τοποθετείται ανάμεσα στην CPU και στη μνήμη RAM ως ενδιάμεσο buffer επικοινωνίας και εκεί αποθηκεύονται τα δεδομένα ή οι εντολές που χρησιμοποιούνται πιο συχνά. Όλα τα συστήματα της μνήμης CACHE βασίζονται στην αρχή της τοπικότητας. Η αρχή της τοπικότητας έχει δύο είδη την χρονική τοπικότητα που μας λέει ότι την λέξη αυτή θα την ξαναχρησιμοποιήσουμε κάποια στιγμή, έτσι είναι χρήσιμο να την βάλουμε στην κρυφή μνήμη όπου μπορεί να είναι γρήγορα προσπελάσιμη και την χωρική τοπικότητα που αναφέρεται στα δεδομένα ή τις εντολές που βρίσκονται σε παραπλήσιες διευθύνσεις με αυτά που χρησιμοποιούνται τη δεδομένη στιγμή.

1.7 Χρονισμός

Σημαντικό ρόλο στο σύστημα του μικροελεγκτή εκτός από τις μνήμες που αναφέρθηκαν παραπάνω παίζει και το ρολόι συστήματος. Παρακάτω στο διάγραμμα φαίνονται οι ταλαντωτές και τα ρολόγια που χρησιμοποιούνται.

Στην αριστερή πλευρά του διαγράμματος υπάρχουν πέντε ταλαντωτές ή αλλιώς πηγές ρολογιού. Δύο από αυτές χρησιμοποιούν εσωτερικούς ταλαντωτές και τρεις από αυτούς χρησιμοποιούν εξωτερικούς κρυστάλλους. Αναλυτικότερα έχουμε :

- Εσωτερικός ταλαντωτής (FRC) έχει σχεδιαστεί για μεγάλη ταχύτητα και χαμηλή κατανάλωση ισχύος. Δεν χρειάζεται εξωτερικά εξαρτήματα και παρέχει ακρίβεια των 8MHz (+-2%) μετά τις ρυθμίσεις.
- Εσωτερικός ταλαντωτής χαμηλής συχνότητας και χαμηλής κατανάλωσης low power oscillator (LPRC). Έχει σχεδιαστεί για χαμηλή ταχύτητα και χαμηλή κατανάλωση ενέργειας. Δεν απαιτεί εξωτερικά εξαρτήματα και παρέχει ένα βασικό ρολόι 32 kHz .



Σχήμα 1.6 PIC32MX clock block diagram

- Εξωτερικός πρωτοβάθμιος ταλαντωτής primary oscillator (POSC): έχει σχεδιαστεί για μεγάλη ταχύτητα και ακριβείς λειτουργίες. Μπορούν να συνδεθούν κρύσταλλοι έως και 20MHz (με τα OSCI, OSCO pins) ενώ είναι διαθέσιμες δύο ρυθμίσεις οι οποίες είναι: XT για τυπικούς κρυστάλλους κάτω από 10MHz και HS για κρυστάλλους με ίδια συχνότητα ίση ή μεγαλύτερη από 10MHz.
- Εξωτερικός ταλαντωτής χαμηλής συχνότητας και χαμηλής κατανάλωσης (αλλιώς και ως δευτερεύον ταλαντωτής SOSC). Αυτός έχει σχεδιαστεί για χαμηλή ταχύτητα και χαμηλή κατανάλωση σε συνεργασία με έναν εξωτερικό κρύσταλλο των 32,768Hz. Μπορεί να χρησιμοποιηθεί ως κύριος ταλαντωτής για ολόκληρο το chip ή απλά ως πηγή για το χρονιστή Timer1 και τις ενότητες RTCC. Η υψηλή ακρίβεια που έχει τον καθιστά ως το ιδανικό ρολόι πηγής για εφαρμογές που απαιτούν ακριβή υπολογισμό του χρόνου.
- Εξωτερική πηγή ρολογιού (EC) η οποία επιτρέπει σ' ένα εξωτερικό κύκλωμα την πλήρη αντικατάσταση ταλαντωτή και παρέχει στον μικροελεγκτή την είσοδο ενός τετραγωνικού κύματος για οποιαδήποτε τετραγωνική συχνότητα.

Οι πέντε παραπάνω πηγές προσφέρουν τις βασικές επιλογές για την δημιουργία ενός σήματος εισόδου του ρολογιού επιθυμητής συχνότητας, την κατανάλωση ενέργειας και την ακρίβεια. Όλα αυτά γίνονται με την δεξιά πλευρά του διαγράμματος. Το ρολόι που παράγεται από την κάθε πηγή

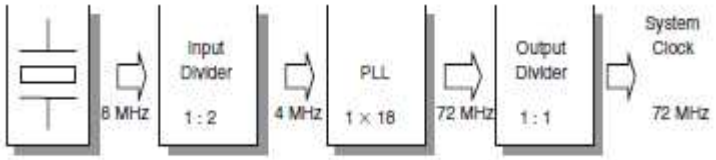
μπορεί να πολλαπλασιαστή ή να διαιρεθεί για να προσφέρει την επιλογή των συγκεκριμένων συχνοτήτων.

1.7.1 Οι ρυθμίσεις συχνότητας του κεντρικού ρολογιού.

Οι εφαρμογές οι οποίες θα αναπτυχθούν θα χρειαστούν ένα ρολόι υψηλής ακριβείας. Ο primary Oscillator Clock Chain είναι το πιο κοινό ρολόι που μπορεί να βοηθήσει. Ακόμη ένας κρύσταλλος 8MHz είναι συνδεδεμένος στα pins OSCI & OSCO. Στην ίδια συχνότητα ρυθμίζουμε την λειτουργία του κύριου ταλαντωτή σε κατάσταση XT. Ανάλογα με την εφαρμογή έχουμε να αντιμετωπίσουμε δύο δυνατότητες, δηλαδή μπορούμε να χρησιμοποιήσουμε το σήμα εισόδου στα 8MHz ή να τροφοδοτήσουμε το κύκλωμα σ' ένα πολλαπλασιαστή PLL. Χρησιμοποιείται συνήθως η δεύτερη επιλογή.

Τα PLLs είναι λίγο πολύπλοκα κυκλώματα αλλά η πολυπλοκότητά τους κρύβεται μέσα στον PIC32 PLL με την προϋπόθεση ότι τηρούνται κάποιοι κανόνες. Πρώτον, πρέπει να οδηγούν μία συγκεκριμένη συχνότητα εισόδου στα 4MHz και δεύτερον πρέπει να επιτρέψουν την σταθεροποίηση ή το κλείδωμα πριν την εκτέλεση του κώδικα και τον συγχρονισμό με αυτά. Επίσης, ένας απλός μηχανισμός ελέγχου παρέχεται (μέσω του καταχωρητή OSCCON), για την επιλογή του συντελεστή της συχνότητας πολλαπλασιασμού (PLLMULT) και τον έλεγχο της κύριας ασφάλειας (SLOCK).

Έτσι, με την χρήση του Explorer16 και του PIC32 Starter Kit χρησιμοποιείται ο πρώτος κανόνας και μειώνεται η συχνότητα από 8MHz στα 4MHz. Παρατηρώντας το διάγραμμα παρακάτω φαίνεται η διαίρεση εισόδου για να αρχίζει να εκτελείται η πρώτη μείωση της συχνότητας.



Σχήμα 1.7 The primary Oscillator Clock Chain

Ο συντελεστής πολλαπλασιασμού των PLL μπορεί να επιλέξει ανάμεσα σε μία σειρά από τιμές που κυμαίνονται από 15X έως 24X και ελέγχεται από τα bits PLLMULT. Η μέγιστη συχνότητα λειτουργίας του PIC32 είναι έως 75MHz, επιλέγοντας τον συντελεστή 18X, θα δώσει 72 MHz και είναι η πλησιέστερη και περισσότερο συμβατή με τις προδιαγραφές της συσκευής. Ο διαιρέτης εξόδου παρέχει μία τελευταία ευκαιρία για την διαχείριση της συχνότητας ρολογιού. Όταν θα χρειαστεί η μέγιστη απόδοση, θα αφήσουμε τον διαιρέτη εξόδου να οριστεί στο 1:1. Εάν η εφαρμογή μας απαιτεί αυτό, θα μπορούμε να μειώσουμε την κατανάλωση ενέργειας από την διαίρεση της συχνότητας εξόδου σε όλη την διαδρομή μέχρι 1:256 ή περίπου στα 280kHz. Κάτω από αυτή την συχνότητα θα μας εξυπηρετούσε ο δευτερεύων ταλαντωτής SOSC, η λειτουργία του οποίου είναι μεταξύ 32kHz και 100kHz και ο εσωτερικός ταλαντωτής χαμηλής ισχύος LPRC που λειτουργεί στα

32kHz. Η τυπική κατανάλωση ρεύματος του PIC32, όταν είναι σε λειτουργία off το LPRC είναι στα 200μA.

1.8 Peripheral Bus Clock

Ένας άλλος τρόπος για να βελτιώσουμε την απόδοση και την κατανάλωση ενέργειας σε μία εφαρμογή είναι όταν ο PIC32 τροφοδοτεί ένα ξεχωριστό σήμα ρολογιού σε όλα τα περιφερειακά. Αυτό επιτυγχάνεται με την αποστολή του σήματος ρολογιού μέσω ενός κύκλου διαίρεσης, που παράγεται από το σήμα ρολογιού PB(Peripheral Bus**)**. Επιπλέον, πολύ συχνά για την ταχύτητα ενός υψηλού επεξεργαστή απαιτείται ένας μεγάλος prescaler μπροστά από έναν χρονιστή για να αποκτήσει το κατάλληλο συγχρονισμό ή απαιτείται ένας μεγάλος διαιρέτης baud rate για την σειριακή θύρα. Άρα λόγω του peripheral bus divider το μερίδιο της ενέργειας που καταναλώνεται από το Peripheral Bus μπορεί να μειωθεί, ενώ ο επεξεργαστής είναι ελεύθερος να λειτουργεί στην μέγιστη ταχύτητα. Αυτή η λειτουργία ελέγχεται από τα PBDIV bits που βρέθηκαν για μία ακόμη φορά μέσα στον καταχωρητή OSCCON. Η τιμή που χρησιμοποιούμε και θα χρησιμοποιείται και στο μέλλον για το Peripheral Bus είναι τα 36MHz που αντιστοιχούν σε 1:2 μεταξύ του ρολογιού του συστήματος και του PB.

1.9 Παράδειγμα με τα Configurations Bits (in code)

Η ικανότητα που έχει το ρολόι να ελέγχει τον χρόνο εκτέλεσης, είναι ένα μεγάλο εργαλείο για τη διαχείριση της ενέργειας. Υπάρχει μία ομάδα bits που είναι γνωστά ως configurations bits τα οποία αποθηκεύονται στην μνήμη flash του PIC32. Αυτά περιέχουν την αρχική διαμόρφωση της συσκευής. Ο ταλαντωτής χρησιμοποιεί μερικά από αυτά για να ρυθμίσει στην αρχή τον καταχωρητή OSCCON. Αυτές οι ρυθμίσεις μπορούν να γίνουν χρησιμοποιώντας των MPLAB από το μενού configure| configuration bits.

Οι επιλογές που χρησιμοποιούνται για την διαμόρφωση του ταλαντωτή είναι :

1. Χρησιμοποιείται το πρωταρχικό κύκλωμα του ταλαντωτή με PLL .
2. Επιλέγεται η λειτουργία XT για τον πρωταρχικό ταλαντωτή.
3. Ρυθμίζεται ο PLL διαιρέτης εισόδου σε 1:2 (για την παραγωγή εισόδου 4MHz).
4. Ρυθμίζεται ο PLL πολλαπλασιαστής σε 18X.
5. Ρυθμίζεται ο PLL διαιρέτης εξόδου σε 1:1 (για την παραγωγή ενός σήματος εξόδου ρολογιού στα 72MHz).
6. Ρυθμίζεται το peripheral clock σε 1:2 (για την παραγωγή εξόδου PB ρολογιού στα 36MHz).
7. Ενεργοποιείται η έξοδος του ρολογιού. Αυτό μπορεί να είναι απενεργοποιημένο όταν χρησιμοποιεί οποιοδήποτε εσωτερικό ταλαντωτή για την απόκτηση ελέγχου των I/O pins.
8. Απενεργοποιείται ο δευτερεύον ταλαντωτής (θα μπορεί να ενεργοποιηθεί αργότερα κατά τον χρόνο εκτέλεσης).
9. Απενεργοποιείται η εσωτερική /εξωτερική μετάβαση ταλαντωτή (Θα χρησιμοποιηθεί μόνο ο εξωτερικός κρύσταλλος).

10. Μοιράζεται ο DBG2 και PGM2 εάν χρησιμοποιεί το ICD/ICSP interface.
11. Αφήνει το flash boot να τροποποιηθεί (Bootloader διαγραφή προστασίας).
12. Απενεργοποίηση κωδικού προστασίας (τουλάχιστον κατά την διάρκεια της ανάπτυξης).
13. Απενεργοποιείται ο Watchdog timer.
14. Απενεργοποιείται τις αλλαγές ρολογιού και του FailSafe Clock Monitor.

Μετά από αυτές τις ρυθμίσεις τα configuration bits αποθηκεύονται στο φάκελο .mcw και θα προγραμματιστούν μέσα στην συσκευή τα configuration bits.

Ακόμη να αναφέρουμε ότι η αξία του PLL ως εσωτερικός διαιρέτης αντιπροσωπεύετε μόνο με την επιλογή των configurations bits και αυτό δεν μπορεί να τροποποιηθεί μέσω του καταχωρητή OSCCON. Η αξία του εξωτερικού κρυστάλλου δεν μπορεί να αλλάξει (εκτός αν το μέρος είναι unsoldered από το PCB και μία νέα συχνότητα μπαίνει στην θέση της), δεν υπάρχει καμία πιθανότητα για να τροποποιηθεί ο εσωτερικός διαιρέτης κατά τον χρόνο εκτέλεσης. Αν η τιμή που καθορίζεται από τα configurations bits ήταν εσφαλμένη ο πολλαπλασιαστής PLL δεν θα λειτουργεί κανονικά και ο PIC32 δεν θα μπορούσε να εκτελέσει οποιοδήποτε κώδικα.

Το MPLAB C32 compiler προσφέρει έναν πρόσθετο μηχανισμό για την αντιστοίχηση τιμών με τα configurations bits των συσκευών. Χρησιμοποιείτε το #pragma config. Επίσης, ο αριθμός των configurations bits μπορεί να αλλάξει από συσκευή σε συσκευή και το MPLAB προσφέρει μία λίστα με τις διαθέσιμες επιλογές για κάθε συσκευή του PIC32 .

Παρακάτω είναι ένας πίνακας με το PLL εξόδου.

FPLLODIV = DIV_1	Divide by 1
FPLLODIV = DIV_2	Divide by 2
FPLLODIV = DIV_4	Divide by 4
FPLLODIV = DIV_8	Divide by 8
FPLLODIV = DIV_16	Divide by 16
FPLLODIV = DIV_32	Divide by 32
FPLLODIV = DIV_64	Divide by 64
FPLLODIV = DIV_256	Divide by 256

Πίνακας 1.2 PLL εξόδου

Πολλά configurations bits μπορούν να ρυθμιστούν μέσα από την δήλωση #pragma config. Ένα παράδειγμα είναι παρακάτω και αναφέρεται στην ρύθμιση του ταλαντωτή .

#pragma config POSCMOD=XT, FNOSC=PRIPLL

#pragma config FPLLDIV=DIV_2, FPLLMUL=MUL_18, FPLLODIV=DIV_1

Η διαμόρφωση μπορεί να ολοκληρωθεί ορίζοντας στο #pragma το peripheral bus clock divider, απενεργοποιούμε το watchdog και την προστασία του κώδικα και ενεργοποιούμε τον προγραμματισμό της μνήμης εκκίνησης και έχουμε

#pragma config FPBDIV=DIV_2, FWDTEN=OFF, CP=OFF, BWP=OFF

Ένα ακόμη ολοκληρωμένο παράδειγμα με τα configurations bits είναι το παρακάτω

```
// Configuration Bits

#pragma config FNOSC    = PRIPLL        // Oscillator Selection
#pragma config FPLLIDIV = DIV_2          // PLL Input Divider (PIC32 Starter Kit: use divide by 2 only)
#pragma config FPLLMUL  = MUL_20        // PLL Multiplier
#pragma config FPLLODIV = DIV_1         // PLL Output Divider
#pragma config FPBDIV   = DIV_1         // Peripheral Clock divisor
#pragma config FWDTEN   = OFF           // Watchdog Timer
#pragma config WDTPS    = PS1           // Watchdog Timer Postscale
#pragma config FCKSM    = CSDCMD        // Clock Switching & Fail Safe Clock Monitor
#pragma config OSCIOFNC = OFF           // CLKO Enable
#pragma config POSCMOD  = XT            // Primary Oscillator
#pragma config IESO     = OFF           // Internal/External Switch-over
#pragma config FSOSCEN  = OFF           // Secondary Oscillator Enable
#pragma config CP       = OFF           // Code Protect
#pragma config BWP      = OFF           // Boot Flash Write Protect
#pragma config PWP      = OFF           // Program Flash Write Protect
#pragma config ICESEL   = ICS_PGx2      // ICE/ICD Comm Channel Select
#pragma config DEBUG    = OFF           // Debugger Disabled for Starter Kit
```

ΚΕΦΑΛΑΙΟ 2ο

ΔΙΑΧΕΙΡΙΣΗ ΠΕΡΙΦΕΡΕΙΑΚΩΝ ΣΥΣΚΕΥΩΝ

2.1 Διαχείριση I/O (PORTS)

Η διαχείριση εισόδου/εξόδου είναι πολύ σημαντική για το σύστημα του μικροελεγκτή. Χρησιμεύει για την ανάγνωση και την εγγραφή δεδομένων καθώς και για επικοινωνία με άλλες συσκευές. Η επικοινωνία του PIC γίνεται μέσω των θυρών με τα pins. Οι καταχωρητές θυρών που έχει ένας PIC32 είναι οι PORTA, PORTB, PORTC, PORTD, PORTE. Στον συγκεκριμένο μικροελεγκτή PIC32MX795F512L χρησιμοποιούμε την PORTD η οποία αποτελείται από 13 bits. Με τον καταχωρητή TRIS ελέγχεται αν μία θύρα θα λειτουργήσει ως είσοδος (με λογικό 1) ή έξοδος (με λογικό 0) στον μικροελεγκτή. Το κάθε pin μιας θύρας μπορεί να οριστεί ανεξάρτητα από τους άλλους. Έτσι, έχουμε μία θύρα που τα pin της έχουν οριστεί ως είσοδος και κάποια άλλα ως έξοδοι.

Οι ψηφιακές είσοδοι ενεργοποιούν κάποια διακοπή, ανάλογα και την λογική κατάσταση που βρίσκονται οι ακροδέκτες που αντιστοιχούν σε αυτές. Τέλος, τα δεδομένα που στέλνονται στην έξοδο μιας θύρας παραμένουν σε αυτήν μέχρι να αλλάξει.

2.2 Διαχείριση Χρονισμού

Οι χρονιστές είναι κυκλώματα που υπάρχουν στο εσωτερικό του μικροελεγκτή και χρησιμοποιούνται για τον συγχρονισμό μιας εφαρμογής. Κάθε χρονιστής διαθέτει έναν καταχωρητή που αυξάνεται βάση μιας πηγής χρονισμού. Όταν συμβαίνει υπερχείλιση ο καταχωρητής αυτός ενεργοποιεί μια διακοπή. Οι οικογένεια του PIC32MX έχει δύο διαφορετικούς τύπους χρονομέτρων. Οι timers είναι χρήσιμοι για την παραγωγή χρόνου με βάση τις περιοδικές εκδηλώσεις που διακόπτουν τις εφαρμογές του λογισμικού ή τη λειτουργία του συστήματος πραγματικού χρόνου. Επίσης, περιλαμβάνουν την μέτρηση των εξωτερικών παλμών ή την ακριβή μέτρηση χρόνου από εξωτερικά γεγονότα, χρησιμοποιώντας το χαρακτηριστικό της θύρας του timer. Με ορισμένες εξαιρέσεις, όλοι οι timers έχουν τα ίδια λειτουργικά κυκλώματα.

Επιπλέον, κάποια ακόμη χαρακτηριστικά σχετικά με τον timer1 είναι ότι έχει μία εσωτερική πηγή χρονισμού του κύκλου εντολής (λειτουργία χρονιστή) ή έναν εξωτερικό παλμό (λειτουργία απαριθμητή). Στην περίπτωση του εξωτερικού παλμού ο παλμός μπορεί να συγχρονιστεί με την φάση της εσωτερικής πηγής του κύκλου εντολής ή να λειτουργήσει ανεξάρτητα. Επίσης, ο Timer1 διαθέτει ξεχωριστό κύκλωμα χρονισμού από αυτό του κεντρικού συστήματος και έτσι επιτρέπει την υλοποίηση ενός ξεχωριστού ταλαντωτή χρονισμού. Ο Timer1 διαθέτει και αυτός διαιρέτη συχνότητας, διαθέτει τον καταχωρητή TMR1(που αποτελείται από δύο καταχωρητές των 8-bit, τον TMR1H και TMR1L). Στον καταχωρητή αυτό μπορούμε να γράψουμε

οποιαδήποτε τιμή ανάμεσα στο 0 και στο 65535. Σε κάθε παλμό της πηγής (όταν δεν έχει συνδεθεί ο διαιρέτης συχνότητας) η τιμή του καταχωρητή TMR1 αυξάνεται κατά 1. Όταν συμβεί υπερχείλιση ενεργοποιείται η σημαία TMR1IF. Ο TMR1 συνεχίζει να αυξάνεται. Αν είναι ενεργοποιημένες οι σημαίες GIE, PEIE και TMR1IE όταν ενεργοποιηθεί η σημαία TMR1IF παράγεται ένα σήμα διακοπής. Επιπλέον, ο Timer1 έχει την δυνατότητα να διακόψει προσωρινά την απαρίθμηση επειδή ο καταχωρητής TMR1 δεν μπορεί να προσπελαστεί απευθείας αλλά μόνο μέσω των καταχωρητών TMR1H και TMR1L η δυνατότητα αυτή μπορεί να αποτρέψει λάθη απαρίθμησης. Όταν απαιτείται μεγάλη ακρίβεια στην χρονική καθυστέρηση μπορούμε να προσθέσουμε στην τιμή του TMR1 μια σταθερά που περιέχει την χρονική διάρκεια κατά την οποία ο Timer1 ήταν σταματημένος. Ο τύπος για τον υπολογισμό της καθυστέρησης του Timer1 σε δευτερόλεπτα είναι:

Τύπος 2.1: Delay=((65536-InitTMR1)*Prescaler)/Frequency

Για τον υπολογισμό της απαιτούμενης αρχικής τιμής του TMR1 για μία δεδομένη χρονική καθυστέρηση χρησιμοποιούμε την παρακάτω εξίσωση:

Τύπος 2.2: InitTMR1=65536-(Delay* Frequency)/ Prescaler

Στην εσωτερική πηγή χρονισμού του κύκλου εντολής, η συχνότητα της πηγής είναι η συχνότητα του ταλαντωτή (Fosc)/4. Ο προγραμματισμός του TIMER1 γίνεται αλλάζοντας την τιμή του καταχωρητή T1CON.

2.2.1 Timer2

Ο χρονισμός του Timer2 γίνεται από μία εσωτερική πηγή χρονισμού του κύκλου εντολής. Δεν λειτουργεί σαν απαριθμητής όπως ο Timer1. Επιπλέον, διαθέτει έναν διαιρέτη συχνότητας και έναν καταχωρητή μεγέθους 8-bit, τον TMR2. Στον καταχωρητή αυτό μπορούμε να γράψουμε οποιαδήποτε τιμή ανάμεσα στο 0 και το 255. Ακόμη διαθέτει έναν καταχωρητή περιόδου τον PR2. Όταν δεν έχει συνδεθεί ο διαιρέτης συχνότητας σε κάθε παλμό της πηγής η τιμή του TMR2 αυξάνεται κατά ένα μέχρι η τιμή του να γίνει ίση με την PR2. Στον επόμενο παλμό της πηγής ο TMR2 μηδενίζεται και ενεργοποιείται η σημαία TMR2IF. Ο TMR2 συνεχίζει να αυξάνεται. Αν είναι ενεργοποιημένες οι σημαίες GIE, PEIE, TMR2IE και όταν ενεργοποιηθεί και η σημαία TMR2IF, παράγεται ένα σήμα διακοπής. Με τον επιπλέον καταχωρητή περιόδου PR2 που διαθέτει ο Timer2 αποφεύγεται η αρχικοποίηση του καταχωρητή TMR2 κάθε φορά που μηδενίζεται, όταν η χρονική διάρκεια της καθυστέρησης παραμένει σταθερή.

Επιπλέον, ο Timer2 διαθέτει και ένα δεύτερο διαιρέτη συχνότητας (postscaler) ο οποίος βρίσκεται μετά τον καταχωρητή TMR2, ώστε οι διακοπές να συμβούν μετά από ένα συγκεκριμένο αριθμό επαναλήψεων. Ο TMR2 έχει την δυνατότητα να διακόψει προσωρινά την απαρίθμηση.

Η εξίσωση που χρησιμοποιούμε για να υπολογίσουμε τη καθυστέρηση ενεργοποίησης της σημαίας TMR2IF του Timer2 σε δευτερόλεπτα είναι:

Τύπος 2.3: Delay=((1+PR2-InitTMR2)*Prescaler * Postscaler)/Frequency

Και η εξίσωση που χρησιμοποιούμε για να υπολογίσουμε την απαιτούμενη αρχική τιμή του PR2 για μία δεδομένη χρονική στιγμή είναι:

Τύπος 2.4: PR2=(InitTMR2-1)+((Delay*Frequency)/(Prescaler * Postscaler))

Ο προγραμματισμός του Timer2 γίνεται αλλάζοντας τιμή στον καταχωρητή T2CON. Ο καταχωρητής αυτός έχει μέγεθος 8 bit και είναι αναγνώσιμος και εγγράψιμος. Τα 7 λιγότερα σημαντικά bits του καταχωρητή T2CON αποτελούν τα ψηφία ελέγχου του Timer2.

2.3 Διαχείριση Σημάτων Διακοπής

Τα σήματα διακοπών αποτελούν γενικά τον πιο αποδοτικό τρόπο χρήσης την Κεντρικής Μονάδας Επεξεργασίας (CPU). Η αρχιτεκτονική του PIC32 περιέχει ένα πλούσιο σύστημα διακοπών που μπορεί να διαχειριστεί μέχρι 64 διακοπές. Μία διακοπή μπορεί να έχει ένα συγκεκριμένο κομμάτι κώδικα που ονομάζεται Interrupt Service Routine ή Interrupt Handler. Μέχρι να δεχτεί κάποια διακοπή, η CPU μπορεί να εκτελεί τη βασική αλληλουχία εντολών του κυρίως προγράμματος, για παράδειγμα να παίρνει μετρήσεις κάποιου φυσικού μεγέθους. Όταν δεχτεί ένα σήμα διακοπής, η CPU σταματά την εκτέλεση του προγράμματος, αποθηκεύει τη διεύθυνση της επόμενης εντολής στον λεγόμενο καταχωρητή στοίβας, πηγαίνει στην εντολή του κυρίως προγράμματος και συνεχίζει την εκτέλεση από εκεί. Οι διακοπές μπορεί να είναι τελείως ασύγχρονες με την εκτέλεση του προγράμματος. Η γρήγορη απόκριση του συστήματος σε μία διακοπή είναι απαραίτητη ώστε να επιτραπεί η έγκαιρη ανταπόκριση σε περίπτωση ενεργοποίησης αλλά και σε γρήγορη αλλαγή στην ανερχόμενη εκτέλεση του προγράμματος. Στόχος είναι να μειωθούν οι αργοπορημένες διακοπές, που είναι ο χρόνος ανάμεσα στο αρχικό γεγονός και στην εκτέλεση από την πρώτη εντολή του Interrupt Service Routine ISR. Ο πυρήνας MIPS που είναι μέσα στον PIC32, περιέχει όλα τα interrupts. Ο μηχανισμός MIPS προβλέπει μια ενιαία ρουτίνα εξυπηρέτησης διακοπών η οποία βοηθάει σε όλες τις πιθανές διακοπές που μπορεί να συμβούν. Μετά την διακοπή συμβαίνει το περιεχόμενο του ειδικού καταχωρητή να δίνει στην ρουτίνα εξυπηρέτησης όλες τις απαραίτητες πληροφορίες για την ανάγνωση του γεγονότος και μετά να λαμβάνει απάντηση. Για να συνεχιστεί η εκτέλεση του προγράμματος από εκεί που έμεινε πριν την διακοπή, η ρουτίνα είναι σε θέση να αποθηκεύσει το περιβάλλον επεξεργασίας και είναι σε θέση για την αποκαταστήσει του προγράμματος όπως ήταν πριν την διακοπή.

Κάθε ένας από τους PIC32 έχει διαφορετικό συνδυασμό των εσωτερικών ή των εξωτερικών πηγών διακοπών. Πολλές από αυτές με την σειρά τους μπορούν να προκαλέσουν πολλές διαφορετικές διακοπές

Από την σχεδίαση έχουμε πάνω από 96 συνολικά ανεξάρτητα γεγονότα που θα μπορούσαν να διαχειρίζονται οι PIC32 για το Interrupt Control module. Πολλά flags και πρόσθετοι μηχανισμοί ελέγχου βοηθούν τον προγραμματισμό.

2.3.1 Interrupts Priorities

Τα Interrupt Control bits χρησιμοποιούνται στις ειδικές συναρτήσεις καταχωρητών.

- Interrupt Enable bit (το συγκεκριμένο bit αντιπροσωπεύεται με την κατάληξη –IE στην διακοπή).
    1. Όταν καθοριστεί από την ειδική trigger είναι εμπόδιο από τις γενικές διακοπές.
    2. Όταν έχει οριστεί επιτρέπει την διακοπή να υποβληθεί σε επεξεργασία.

- Interrupt Flag (το συγκεκριμένο bit αντιπροσωπεύεται με την κατάληξη–IF στην διακοπή). Ένα μόνο bit δεδομένων βρίσκει κάθε φορά η συγκεκριμένη trigger και ενεργοποιείται, ανεξάρτητα από την κατάσταση του επιτρεπόμενου bit. Επίσης, πρέπει να καθοριστεί από τον χρήστη και πρέπει να καθοριστεί πριν από την έξοδο της ISR.
- Group Priority Level (το συγκεκριμένο bit αντιπροσωπεύεται με την κατάληξη –IP στην διακοπή). Οι διακοπές μπορεί να έχουν ως 7 επίπεδα προτεραιότητες (ipl1-ipl7). Εφόσον δύο γεγονότα που συμβαίνουν την ίδια στιγμή διακόπτονται αυτό που θα εξυπηρετηθεί πρώτο είναι αυτό που εμφανίζει την υψηλότερη δραστηριότητα. Σε οποιαδήποτε δεδομένο σημείο ο PIC32 έχει την δυνατότητα να εκτελείται και να διατηρεί τον πυρήνα MIPS τον Status Register. Οι διακοπές που γίνονται σε ένα χαμηλό επίπεδο προτεραιότητας από την τρέχουσα τιμή θα αγνοηθούν.
- Subpriority Level. Δύο ή περισσότερα bits που διατίθενται για τον καθορισμό τεσσάρων επιπέδων προτεραιότητας. Αν έχω 2 γεγονότα του ίδιου επιπέδου προτεραιότητας που συμβαίνουν ταυτόχρονα, επιλέγω πρώτο αυτό με το υψηλότερο Subpriority. Όμως, μόλις μια διακοπή δεδομένης προτεραιότητας έχει επιλεγεί, οποιαδήποτε παρακάτω διακοπή του ίδιου επιπέδου θα αγνοηθεί μέχρι να καθαριστεί η τρέχουσα διακοπή.

Οι τρείς macros συναρτήσεις που χρησιμοποιούμε είναι :

- INTEnableSystemSingleVectoredInt(): Είναι μία συνάρτηση που ακολουθεί ακριβώς το αποτέλεσμα τοποθέτησης αρχικών τιμών των διακοπών ρύθμισης της μονάδος, ενδυναμώνει την βασική διαχείριση των διακοπών του PIC32. Το ασυνήθιστο μεγάλο όνομα μας βοηθάει να αποφύγουμε το βάρος και φτιάχνουμε εύκολα κώδικα και με ασφάλεια.
- mXXSetIntPriority(x): Είναι ένα σύμβολο για μία αντικατάσταση μεγάλης λίστας όμοια με τον συναρτήσεων macro. Αυτό αναθέτει να δώσει προτεραιότητα στο επίπεδο (από 0 μέχρι 7) και να διαλέξει την αρχή της διακοπής. Υπάρχει μεγάλη ευκαιρία για την ποσότητα παραγωγής για τον σωστό καταχωρητή IPxx που διαλέγουμε το σωστό IPbits στην αρχή διακοπών.

- mXXClearIntFlag(): Είναι μία macro συνάρτηση που μπορεί να αντιπροσωπεύσει ολόκληρη την κλάση των macros συναρτήσεων το οποίο επιτρέπει εμάς να καθαρίσουμε την σημαία διακοπών.

2.3.2 Διαχείριση Διακοπών (Multivectored Interrupt Management)

Ο βασικός μηχανισμός του PIC32 της υπηρεσίας διακοπών δεν διαφέρει από την αρχιτεκτονική του 8-bit PIC. Όλες οι πηγές διακοπών διοχετεύονται από μία ή περισσότερες ρουτίνες εξυπηρέτησης διακοπών. Η ρύθμιση αυτή είναι απλή, αλλά, λαμβάνουμε υπόψη και τη ταχύτητα του PIC32 (και την ικανότητα να εκτελεί μια εντολή ανά κύκλο ρολογιού).

Επίσης, παρέχει μια μικρή δυνατή επιβάρυνση και δίνει απάντηση σε υψηλής προτεραιότητας διακοπές, ο PIC32 προσφέρει έναν εναλλακτικό μηχανισμό που χρησιμοποιεί διακοπές vectored και σύνολα multiple register . Η οικογένεια PIC32MX προσφέρει 64-vector table και 2 πλήρεις σειρές από 32 working registers. Ο μέγιστος ρυθμός φορέων είναι 64 που συσχετίζονται με τον πυρήνα MIPS.

2.3.3 The Secondary Oscillator

Υπάρχει και άλλο χαρακτηριστικό του PIC32 ο χρονιστής TIMER1 που μπορεί να χρησιμοποιηθεί για να αποκτήσει ένα ρολόι πραγματικού χρόνου (real time clock). Στην πραγματικότητα υπάρχει ένας ταλαντωτής χαμηλής συχνότητας (γνωστός και ως secondary oscillator) που μπορεί να χρησιμοποιηθεί για να τροφοδοτήσει το χρονιστή TIMER1 αντί της υψηλής συχνότητας του ρολογιού. Δεδομένου ότι είναι σχεδιασμένο για χαμηλή συχνότητα λειτουργίας (συνήθως χρησιμοποιείται σε συνδυασμό με έναν κρύσταλλο 32.768 Hz) απαιτείται πολύ λίγη ενέργεια για να λειτουργήσει αν και αυτό είναι ανεξάρτητο από την κύρια λειτουργία του ρολογιού, μπορεί να διατηρηθεί σε λειτουργία όταν το κύριο ρολόι είναι απενεργοποιημένο και ο επεξεργαστής εισέρχεται σε μία από τις πολύ πιθανές χαμηλής ισχύος εφαρμογές. Στην πραγματικότητα ο Secondary Oscillator είναι ένα ουσιαστικό μέρος για πολλές από τις εφαρμογές της χαμηλής ισχύος. Σε ορισμένες περιπτώσεις χρησιμοποιείται για να αντικαταστήσει το κύριο ρολόι, ενώ σε άλλες περιπτώσεις παραμένει ενεργός για να τροφοδοτεί τον TIMER1 ή να επιλέξει την ομάδα των περιφερειακών.

2.3.4 The Real Time Clock Calendar(RTCC)

Η οικογένεια του PIC32 έχει τον πλήρη real time clock calendar(RTCC) είναι μονάδες που κατασκευάστηκαν και είναι έτοιμες για χρήση. Δεν εργάζονται μόνο με την χαμηλή ισχύ του secondary oscillator αλλά έχουν μία λειτουργία που μπορούν να παράγουν διακοπές. Όταν η μονάδα έχει προετοιμαστεί είναι δυνατό να ενεργοποιηθεί η μονάδα RTCC και περιμένουμε να παραχθεί μία διακοπή.

Για να προετοιμαστεί η RTCC εφαρμογή θέλει πολλές τροποποιήσεις, θέλει τα δεδομένα να προσπελαστούν με την σωστή σειρά. Το πρότυπο του

PIC32 συμπεριλαμβανομένου και το plib.h θα αποκτήσει πρόσβαση σε ένα σύνολο λειτουργίας που κάνει την διαδικασία αρκετά απλή.

Αλλά αν το τμήμα του κώδικα που προσπαθούμε να προστατέψουμε από τις διακοπές θα μπορούσε να χρησιμοποιηθεί κατά περιόδους και όταν δεν ξέρουμε αν τα Interrupts είναι ενεργοποιημένα/απενεργοποιημένα, μπορούμε να χρησιμοποιήσουμε και να ζητήσουμε μία από τις δύο συναρτήσεις που είναι :

- INTDisableInterrupts(): αυτή χρησιμοποιείται όχι μόνο για την διακοπεί ενεργοποίησης αλλά επιστρέφει μία τιμή που αντιστοιχεί στην αρχική κατάσταση διακοπής.
- Όταν τελείωσουμε χρησιμοποιούμε το INTRestoreInterrupts(status) . Για την αποκατάσταση της αρχικής κατάστασης του συστήματος.

Επιπλέον, σύμφωνα με τον PIC32 για την ενεργοποίηση του secondary oscillator χαμηλής ισχύος, πρέπει να οριστεί το SOSEN bit και ο καταχωρητής OSCOON. Παρατηρούμε ότι ο OSCOON έχει σημασία για την MCU και επηρεάζει την επιλογή του κύριου ενεργού ταλαντωτή και την ταχύτητά του, αλλά προστατεύεται από ένα μηχανισμό κλειδώματος. Ως μέτρο ασφάλειας, μπορεί και πρέπει να εκτελέσει την ειδική ακολουθία ξεκλειδώματος ή η εντολή μας θα αγνοηθεί.

Ο PIC32 στην περίπτωση αυτή μας σώζει από μια σειρά χρήσιμων λειτουργιών που διαμορφώνουν την ενότητα του ταλαντωτή και ακολουθούν όλα τα απαραίτητα κλειδώματα, οι ακολουθίες ξεκλειδώματος περιλαμβάνουν:

- mOSCEnableSOSC(), μας επιτρέπει να ενεργοποιήσουμε η να απεργοποιήσουμε το (mOSCDisableSOSC()) του secondary oscillator (SOSC()) κατά τον χρόνο εκτέλεσης.
- OSSConfig(), μπορεί να αλλάξει δυναμικά την επιθυμητή πηγή του ρολογιού(κατά την διάρκεια εκτέλεσης του προγράμματος), του πολλαπλασιαστής PLL και του PLL postscaler ή/και ο διαιρετής FRC.
- mOSCSetPBDIV(), μας επιτρέπει να αλλάξουμε το περιφερειακό διαχωριστικό ρολόι BUS δυναμικά. Χρησιμοποιούμε την λειτουργία αυτή με μεγάλη προσοχή, διότι θα επηρεάσει ταυτόχρονα την λειτουργία όλων των περιφερειακών μας.

Δύο επιπλέον λειτουργίες που φροντίζουν την επαναρύθμιση των παραμέτρων του PIC32MX για idle και sleep τρόπο λειτουργίας είναι :

- mPowerSaveSleep(), οι στάσεις των δύο από το ρολόι του συστήματος και το bus περιφερειακό ρολόι του PIC32 όπου η συσκευή μπαίνει σε μία κατάσταση κατανάλωσης. Η τυχόν επαναφορά και η ασύγχρονη περιφερειακή εκδήλωση θα ξυπνήσει την συσκευή ακόμη και αν η αντίστοιχη διακοπή δεν είναι ενεργοποιημένη.
- mPowerSaveIdle() σταματάει το ρολόι του συστήματος αλλά αφήνει το τρέξιμο του περιφερειακού ρολογιού. Κάθε ενεργή περιφερειακή πηγή διακοπής μπορεί να ξυπνήσει τη συσκευή. Για παράδειγμα κάποιες είναι η UART, SPI, οι χρονοδιακόπτες, οι είσοδοι Capture, οι έξοδοι Compare και πολλά άλλα περιφερειακά.

ΚΕΦΑΛΑΙΟ 3ο

ΔΙΑΧΕΙΡΙΣΗ ΕΠΙΚΟΙΝΩΝΙΑΣ

3.1 Σειριακή Επικοινωνία

Η σειριακή επικοινωνία είναι μία μέθοδος μετάδοσης δεδομένων ανάμεσα σε μία περιφερειακή συσκευή και έναν υπολογιστή. Ακόμη μεταδίδει διαδοχικά δεδομένα ένα bit κάθε φορά από μία ενιαία γραμμή επικοινωνίας κάθε φορά σ' ένα δέκτη. Η σειριακή επικοινωνία είναι το πιο δημοφιλές πρωτόκολλο επικοινωνίας που χρησιμοποιείται από πολλές συσκευές και όργανα. Η σειριακή επικοινωνία είναι δημοφιλής επειδή οι περισσότεροι υπολογιστές έχουν μία ή περισσότερες σειριακές θύρες και έτσι δεν χρειάζεται κανένα επιπλέον υλικό, εκτός από ένα καλώδιο για τη σύνδεση της περιφερειακής συσκευής με τον υπολογιστή ή των δύο υπολογιστών μαζί. Συγκεκριμένα θα δούμε όλες τις διαθέσιμες περιφερειακές συσκευές επικοινωνίας της οικογένειας του PIC32MX. Ειδικότερα θα εξερευνήσουμε την ασύγχρονη σειριακή επικοινωνία (UART) και την σύγχρονη σειριακή επικοινωνία (SPI).

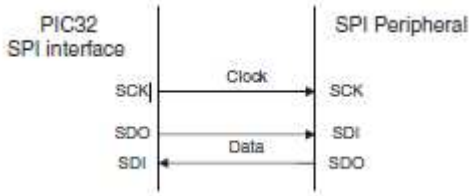
Η οικογένεια του PIC32MX προσφέρει 7 περιφερειακά επικοινωνίας που έχουν σχεδιαστεί για να βοηθούν όλες τις εφαρμογές ελέγχου. Μέχρι 6 από αυτά είναι συσκευές σειριακής επικοινωνίας. Μεταδίδουν και λαμβάνουν κάθε φορά ένα κομμάτι των πληροφοριών. Αυτά είναι :

- 2X Universal Asynchronous Receiver and Transmitters UARTS
- 2X SPI σύγχρονη σειριακή επικοινωνία
- 2X I^2C σύγχρονη σειριακή επικοινωνία

Η κύρια διαφορά ανάμεσα στη σύγχρονη και την ασύγχρονη σειριακή επικοινωνία είναι ο τρόπος που περνούν οι χρονικές πληροφορίες από τον πομπό στο δέκτη. Η περιφερειακή επικοινωνία χρειάζεται μια φυσική γραμμή να είναι αφιερωμένη στο σήμα ρολογιού που παρέχεται μεταξύ των 2 σύγχρονων συσκευών. Η συσκευή που προέρχεται από το σήμα ρολογιού ονομάζεται master και η συσκευή που συγχρονίζει ονομάζεται slave.

3.1.1 Σύγχρονη Σειριακή Επικοινωνία

Το SPI interface χωρίζει την μία γραμμή των δεδομένων σε δύο. Ένα για την είσοδο (SDI) και ένα για την έξοδο (SDO). Επίσης, χρειάζεται ένα ακόμη καλώδιο για το clock. Έτσι, τα δεδομένα μεταφέρονται ταυτόχρονα προς τις δύο κατευθύνσεις.



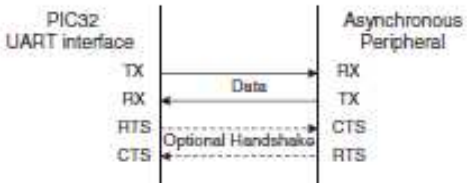
Σχήμα 3.1 Διάγραμμα SPI

Το SPI interface απαιτεί μια πρόσθετη φυσική γραμμή. Ο slave επιλέγει να συνδεθεί με κάθε συσκευή. Αυτό σημαίνει κατά τη χρήση ενός bus SPI, καθώς αυξάνεται ο αριθμός των συνδεδεμένων συσκευών, ο αριθμός των I/O pins που απαιτούνται μεγαλώνει αναλογικά. Το κύριο πλεονέκτημα του SPI interface είναι η απλότητα και η ταχύτητα του, που είναι μεγαλύτερη από την ταχύτητα του I^2C.

Επίσης, το SPI interface είναι το πιο απλό όλων των interface. Ακόμη τα bits είναι μετατοπισμένα στο πιο σημαντικό bits (MSB), από την γραμμή SDI τα δεδομένα μετατοπίζονται στην γραμμή SDO σε συγχρονισμό με το ρολόι στο συγκεκριμένο pin στο SCK. Το μέγεθος της μετατόπισης του καταχωρητή μπορεί να κυμαίνεται από 8, 16 ή 32 bit. Εάν η συσκευή έχει ρυθμιστεί ως master bus το ρολόι παράγεται εσωτερικά, προέρχεται από το περιφερειακό ρολόι (Fpb) για το baud rate generator και η έξοδος βγαίνει στο pins SCK. Διαφορετικά η συσκευή είναι ένας slave bus. Ακόμη το λιγότερο σημαντικό bit περιέχει όλα τα απαραίτητα κομμάτια διαμόρφωσης. Όλο αυτό κάνει τον καταχωρητή ελέγχου συμβατό με τις προηγούμενες γενιές των μικροελεγκτών 16-bit PIC.

Για να αποδείξουμε όλη αυτή την λειτουργικότητα του SPI χρησιμοποιούμε τον explorer 16 κατά τον οποίο ο PIC32 είναι συνδεδεμένος με την 25LC256 EEPROM . Αυτή είναι μία μικρή συσκευή που περιέχει 256Kbits ή 32Kbits.

Στο ασύγχρονο σειριακό Interface δεν υπάρχει γραμμή ρολογιού, υπάρχουν δύο γραμμές –TX και η –RX που χρησιμοποιούνται αντίστοιχα για είσοδο και έξοδο. Επίσης, οι δύο αυτές γραμμές μπορεί να χρησιμοποιηθούν και για την χειραψία υλικού. Ο συγχρονισμός μεταξύ πομπού και δέκτη προέρχεται αν εξάγουμε τον χρόνο των πληροφοριών από το ρεύμα που βρίσκονται τα ίδια τα δεδομένα. Η έναρξη και η διακοπή των bits προστίθενται στα δεδομένα και η μορφοποίηση (με σταθερό ρυθμό baud) πρέπει να ρυθμιστεί για την αξιόπιστη μεταφορά των δεδομένων. Το σχήμα παρακάτω δείχνει το block διάγραμμα του σύγχρονου σειριακού interface .



Σχήμα 3.2 Block διάγραμμα του ασύγχρονου σειριακού interface

3.1.2 Ασύγχρονη Επικοινωνία

Η ασύγχρονη επικοινωνία προκύπτει με την αφαίρεση της γραμμής ρολογιού από την σειριακή διεπαφή μεταξύ δύο συσκευών, αυτό που θα αποκτήσουμε είναι μια ασύγχρονη σειριακή διεπαφή επικοινωνίας. Υπάρχουν πολλά πρωτόκολλα που θα κάνουν πιο αποτελεσματική την ασύγχρονη επικοινωνία. Μερικά από αυτά είναι το UART1 και UART2 και εφαρμόζονται στο RS232 Interface .

Η διεπαφή UART είναι η παλαιότερη διεπαφή που χρησιμοποιείται στον ενσωματωμένο έλεγχο. Επιπλέον, οι 4 βασικές κατηγορίες που χρησιμοποιούνται στον ασύγχρονη σειριακή εφαρμογή ακολουθούν τα εξής βήματα :

1. RS232 ακολουθεί την σύνδεση από σημείο σε σημείο (point to point). Είναι μια σειριακή θύρα που χρησιμοποιείται για modems καθώς και για προσωπικούς υπολογιστές που χρησιμοποιούν πομποδέκτες από +12V/-12V.

2. RS485 (EIA-485) είναι σειριακή σύνδεση πολλαπλών σημείων. Χρησιμοποιείται σε βιομηχανικές εφαρμογές, χρησιμοποιεί μία 9-bit λέξη και είναι ειδική για ημιαφίδρομους πομποδέκτες.

3.LIN bus είναι χαμηλού κόστους, σχεδιασμένο με χαμηλής τάσης bus για μη κρίσιμες εφαρμογές. Χρειάζεται μία UART για αυτόματη ανίχνευση baud rate.

4. Υπέρυθρη ασύρματη επικοινωνία. Απαιτείται 32-40KHz διαμόρφωση σήματος και οπτικούς πομποδέκτες.

Για την βασική λειτουργία της UART θα χρησιμοποιηθεί ο Explorer16. Αυτός μπορεί να είναι συνδεδεμένος σε οποιαδήποτε σειριακή θύρα του PC και μαζί με το πρόγραμμα Windows Hyper Terminal ανταλλάσσονται δεδομένα.

Αρχικά ορίζονται οι παράμετροι μετάδοσης. Οι επιλογές Περιλαμβάνουν

- Baud rate
- Αριθμό των bits δεδομένων
- Bits ισοτιμίας αν υπάρχουν
- Αριθμό των bit διακοπής
- Πρωτόκολλο χειραψίας

Για τον Explorer16 οι ρυθμίσεις είναι 115200,8,N,1,CTS/RTS –αυτό είναι

- 115,200 Baud
- 8 bits δεδομένων
- Χωρίς ισοτιμία
- 1 stop bit
- Χειραψία υλικού χρησιμοποιώντας τις γραμμές CTS και το RTS

Αρχικά στη ασύγχρονη σειριακή επικοινωνία ορίζονται κάποια πράγματα για διευκόλυνση της χειραψίας

```
#include <p32xxxx.h>

#define CTS _RF12 //clear to send , input

#define RTS _RF13 //request to send , output

#define TRTS TRISFbits.TRISF13 //tris control for RTS pin
```

Η χειραψία του υλικού είναι ιδιαίτερα σημαντική στην επικοινωνία με το Windows hyper terminal. Θα χρησιμοποιηθεί ένα I/O pins ως είσοδος (RF12 για την εξερεύνηση των 16 baud), για την αίσθηση ότι το τερματικό είναι έτοιμο να δεχθεί ένα νέο χαρακτήρα (clear to send) και ένα I/O pins ως ισχύς εξόδου (RF13 για την εξερεύνηση των 16 board), για να πληροφορεί το τερματικό όταν η εφαρμογή μας είναι έτοιμη να λάβει ένα χαρακτήρα (αίτημα για αποστολή).

Για να ρυθμιστεί η ταχύτητα του Baud Rate, χρησιμοποιείται το baud rate generator (U2BREG) που είναι ένας 16-bit μετρητής που χρησιμοποιεί το περιφερειακό ρολόι bus. Η κανονική κατάσταση λειτουργίας (BREGH=0) λειτουργεί από τη μία 1:16 ως διαχωριστικό σε σχέση με μια υψηλή ταχύτητα λειτουργίας (BRECH=1) όπου το ρολόι δραστηριοποιείται από τον διαιρέτη 1:14.

Baud Rate=FpB/4*(U X BRG + 1)

U X BRG= (FpB/4*Baud Rate)-1

Μπορούμε επίσης να ρυθμίσουμε το BRate

```
#define BRATE 77  // 115,200 Bd (BRECH=1)
```

Επιπλέον, υπάρχουν δύο ακόμη μεταβλητές που βοηθούν στην αρχικοποίηση και στον καθορισμό τιμών της UART2 στον control registers είναι ο U2MODE και η U2STA.

Η τιμή αρχικοποίησης του U2MODE περιλαμβάνεται στο BRECH=1, ο αριθμός των σταματημένων bits καθώς και οι ρυθμίσεις του bit ισοτιμίας είναι

```
#define U_ENABLE 0x8008 //enable, BRECH=1, 1 stop, no parity
```

Με την αρχικοποίηση του U2STA θα ενεργοποιηθεί η λήψη και η μετάδοση των δεδομένων.

```
#define U_RX_TX 0x1400 //Enable Receive and Transimt module
```

Για την αρχικοποίηση του UART module και τον υπολογισμό του baud rate χρησιμοποιείται η συνάρτηση void initU2(void). Στο πρακτικό μέρος θα γίνει αναλυτική περιγραφή της συνάρτησης.

3.1.2.1 Αποστολή και Λήψη Δεδομένων

Η αποστολή χαρακτήρων στην σειριακή θύρα είναι με τρία βήματα:

1. Βεβαιωνόμαστε ότι το τερματικό (το PC που τρέχει στον Windows Hyper Terminal) είναι έτοιμο να ελέγχει την γραμμή clear to send (CTS). Το CTS είναι ενεργά χαμηλό σήμα και περιμένουμε για να γίνει υψηλό.

2. Βεβαιωνόμαστε ότι το UART δεν είναι απασχολημένο για την αποστολή ορισμένων προηγούμενων δεδομένων. Ο PIC32 του UART έχει 4 επίπεδα βάθους buffer FIFO, οπότε χρειάζεται να περιμένουμε μέχρι να απελευθερωθεί το ανώτερο επίπεδο. Αυτό γίνεται ελέγχοντας την μετάδοση buffer της πλήρης σημαίας UTxBF να είναι καθαρό.

3. Τέλος μεταφέρεται ο νέος χαρακτήρας μετάδοσης της UART στο buffer FIFO

Ο κώδικας της αποστολής δεδομένων υλοποιείται με συγκεκριμένα βήματα που θα αναφερθούν στο πρακτικό μέρος. Η συνάρτηση που χρησιμοποιείται είναι η int putU2(int c).

3.1.2.2 Για να λάβω δεδομένα-χαρακτήρα

Για να λάβουμε ένα δεδομένο χαρακτήρα από την σειριακή θύρα ακολουθούμε μία παρόμοια λειτουργία:

1. Ειδοποιούμε το τερματικό ότι είμαστε έτοιμοι να λάβουμε ένα χαρακτήρα από την διεκδίκηση των σημάτων RTS(active low).

2. Περιμένουμε υπομονετικά να φτάσει στο buffer λήψης ο έλεγχος της σημαίας URxDA στο εσωτερικό του καθεστώτος UART2 του καταχωρητή STATUS U2STA .

3. Φέρνει τον χαρακτήρα από την λήψη buffer (FIFO).

Όλα τα παραπάνω βήματα είναι οργανωμένα σε μία λειτουργία και συγκεκριμένα στην συνάρτηση char getU2(void) όπου γίνεται αναφορά στο μέρος Β και αναλύεται η λειτουργία τους.

3.2 Πλεονεκτήματα και Μειονεκτήματα της Ασύγχρονης και Σύγχρονης επικοινωνίας

Το πλεονέκτημα της σύγχρονης επικοινωνίας στην μεταφορά δεδομένων είναι η χαμηλή επιβάρυνση και η μεγάλη απόδοση σε σχέση με την ασύγχρονη επικοινωνία. Όσον αφορά τα μειονεκτήματα είναι λίγο πιο πολύπλοκη και το Hardware είναι πιο ακριβό.

Σχετικά με την ασύγχρονη επικοινωνία κάποια από τα πλεονεκτήματα είναι ότι είναι απλή και δεν απαιτεί πολύ συγχρονισμό και στις δύο πλευρές της επικοινωνίας, ακόμη ο χρόνος δεν παίζει τόσο σημαντικό ρόλο όπως στην σύγχρονη επικοινωνία και το hardware είναι φθηνότερο. Τέλος, ένα ακόμη πλεονέκτημα είναι ότι το Set-up είναι πολύ γρήγορο, αυτό είναι κατάλληλο για την δημιουργία μηνυμάτων σε μη τακτά διαστήματα όπως για παράδειγμα την εισαγωγή δεδομένων από το πληκτρολόγιο. Σχετικά με το μειονέκτημα της ασύγχρονης επικοινωνίας είναι η μεγάλη επιβάρυνση, όπου ένα υψηλό ποσοστό των bits που μεταδίδονται είναι μοναδικά για τον έλεγχο και δεν φέρουν χρήσιμες πληροφορίες.

Συμπερασματικά, μπορούμε να αναφέρουμε ότι παρά την μείωση της δημοτικότητας, την ανώτερη απόδοση, τα χαρακτηριστικά του USB interface, η ασύγχρονη σειριακή διεπαφή αντιστέκεται έντονα στον κόσμο της ενσωματωμένης εφαρμογή, λόγω της μεγάλης απλότητας του και του χαμηλού κόστους της εφαρμογής.

ΜΕΡΟΣ Β

ΕΦΑΡΜΟΓΕΣ ΜΕ ΤΟΝ PIC32

ΚΕΦΑΛΑΙΟ 4ο

ΕΦΑΡΜΟΓΕΣ ΔΙΑΧΕΙΡΙΣΗΣ Ι/Ο

4.1 Αναπτυξιακό Κύκλωμα (Starter Kit, Explorer16,PicKit 3)

4.1.1 Ο PIC32

Ο μικροελεγκτής που χρησιμοποιούμε για τις παρακάτω εφαρμογές είναι ο PIC32MX795F512L είναι 32-bit. Πιο συγκεκριμένα παρακάτω είναι το αναπτυξιακό και θα το περιγράψουμε αναλυτικά.

Εικόνα 4.1 Το αναπτυξιακό

1. Ο μικροελεγκτής που έχω PIC32MX795F512L των 32-bit

2. Το USB του μικροελεγκτή PIC32MX440F512H για εξωτερική εκσφαλμάτωση.

3. Το LED του PIC32 είναι πράσινο και δείχνει ότι ο μικροελεγκτής είναι σε λειτουργία.

4. Ο εξωτερικός Κρύσταλλος για ακρίβεια χρονισμού του μικροελεγκτή (8 MHz).

5. Το USB σύνδεσης για επικοινωνία και εξωτερική εκσφαλμάτωση.

6. Το συγκεκριμένο LED είναι πορτοκαλί και έχουμε εκσφαλμάτωση .

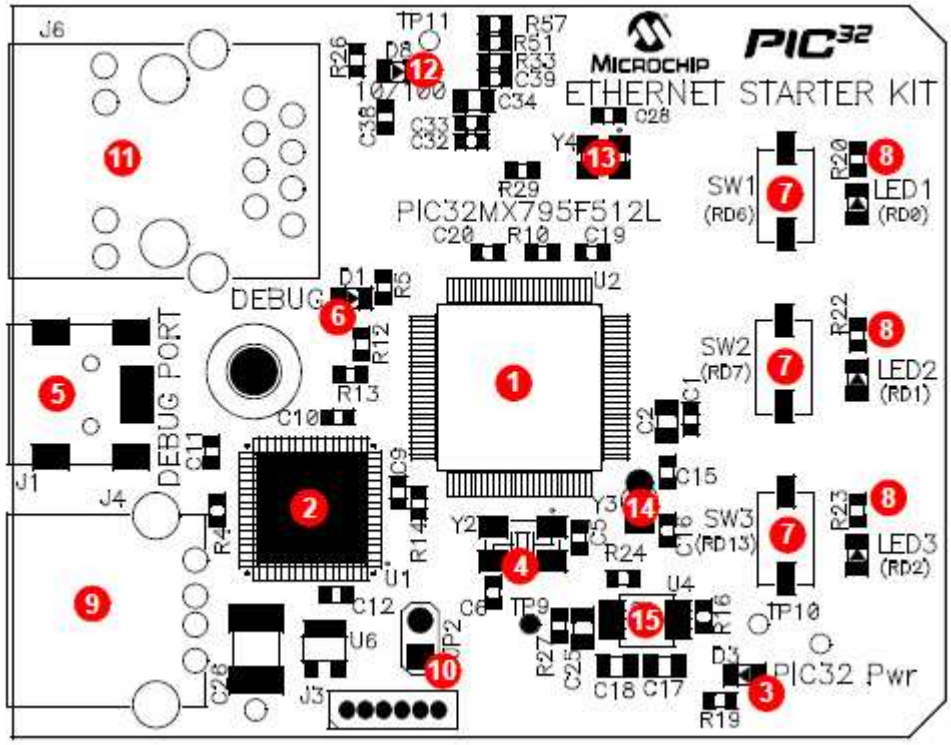
7. Τα τρία switches που χρησιμοποιούνται για είσοδο.

8. Τα τρία LEDs που οδηγούν στην είσοδο.

9. Η θύρα USB για συνδέσεις με host υπολογιστή αλλά και με άλλες συσκευές.

10. Ένα jumper για συνδέσεις με host υπολογιστή.

11. Θύρα για το Ethernet RJ-45 (καλώδιο).



Εικόνα 4.2 Η μπροστά όψη του PIC32 64

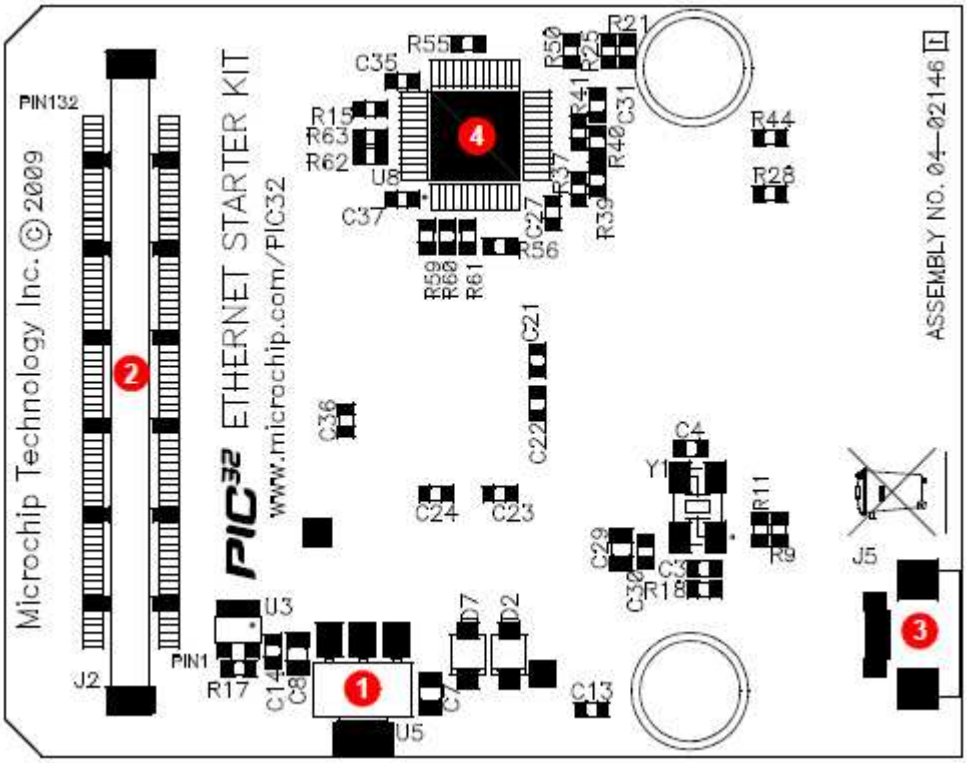
12. Το LED που είναι ενδεικτικό της ταχύτητας 10/100 του δικτύου Ethernet .

13. Ο ταλαντωτής Ethernet PHY είναι στα 50 MHz.

14. Ο ταλαντωτής είναι στα 32 KHz.

15.Το USB host και το on the go(OTG) χρησιμοποιούνται για εφαρμογές που υποστηρίζουν συνδέσεις USB του μικροελεγκτή PIC32.

Από την πίσω μεριά του αναπτυξιακού έχω τα εξής



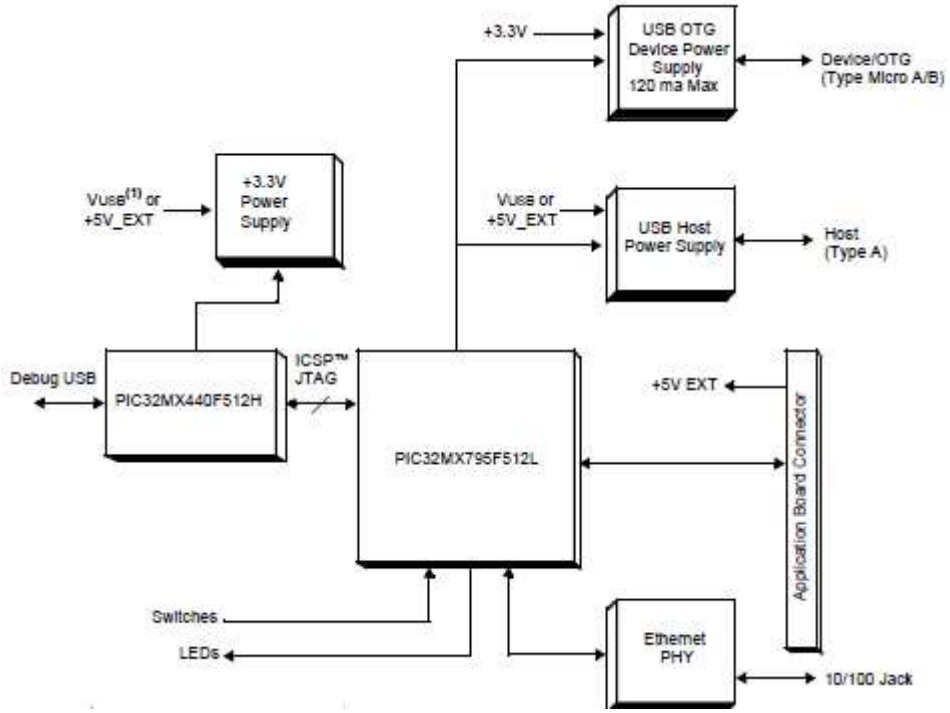
Εικόνα 4.3 Η πίσω όψη του PIC32

1. Το Starter kit ρυθμίζεται στα 3,3V για να υποστηρίξει εφαρμογές μέσω του USB ή κάποια εξωτερική επέκταση.

2. Σύνδεση για ποικιλία εξωτερικής επέκτασης.

3. Θύρα USB Micro-AB για OTG(on the go).

4. Εξωτερικές PHY Ethernet.



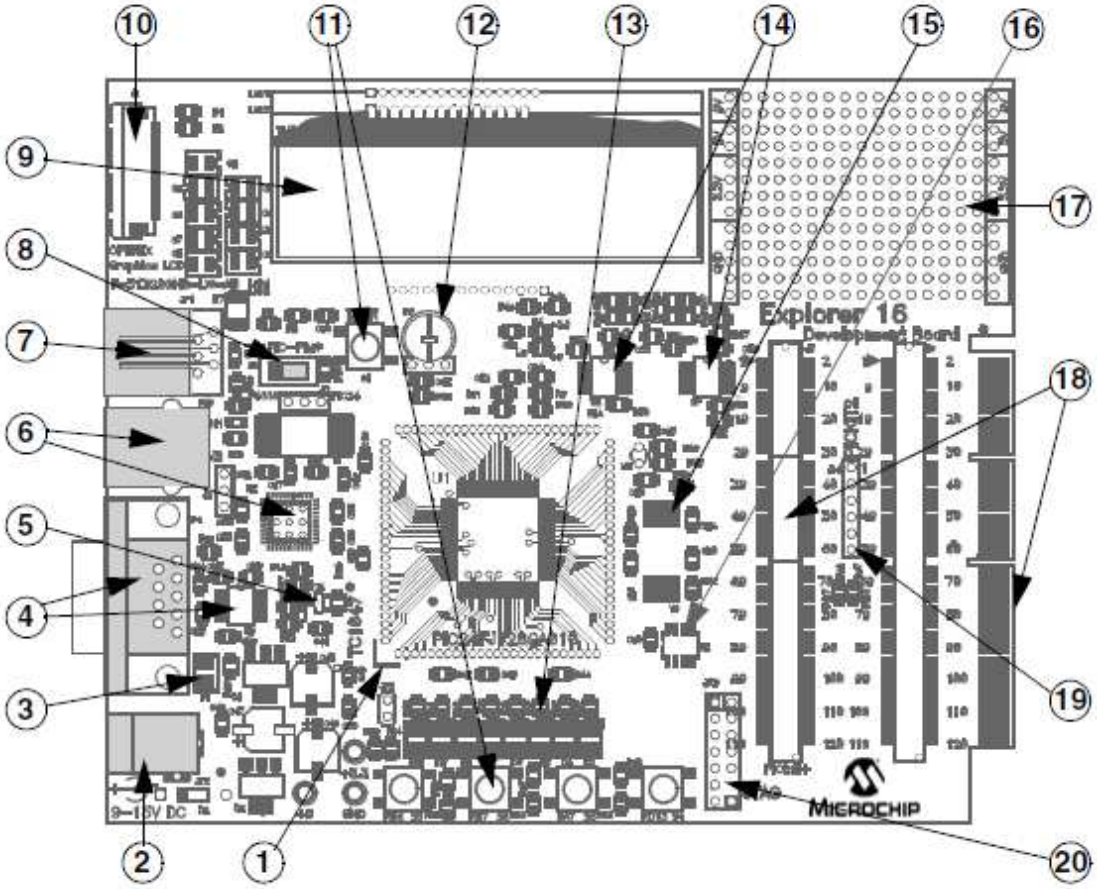
Σχήμα 4.1 Block διάγραμμα του PIC32 Ethernet Starter Kit

Επιπλέον, ο PIC32 Ethernet Starter Kit έχει σχεδιαστεί με 120 ακροδέκτες, ο επεξεργαστής του είναι ο PIC32MX795F512L. Ακόμη υπάρχουν δύο τρόποι υποστήριξης ο PIC32 Ethernet Starter Kit έχει το USB που συνδέεται με το usb debug connector j1 καθώς και μία εξωτερική εφαρμογή με ρύθμιση DC η οποία υποστηρίζεται στα 5V η οποία συνδέεται με το j2 στην άκρη. Το LED είναι πράσινο και αυτό δείχνει ότι όλα λειτουργούν σωστά. Επιπλέον, υπάρχουν τρεις τρόποι για να συνδέσουμε το PIC32 οι οποίοι είναι με host mode, με device mode και OTG(on the go) mode. Κάθε ένα από τα τρία switches συνδέεται με 1 LED αντίστοιχα. Το SW1 συνδέεται με τον ακροδέκτη RD6 που είναι το LED1, το SW2 συνδέεται με τον ακροδέκτη RD7 που είναι το LED2, το SW3 συνδέεται με τον ακροδέκτη RD13 που είναι το LED3. Ακόμη στα LEDs το RD0 μέσω του RD2 είναι συνδεδεμένα με την portd του επεξεργαστή. Οι ακροδέκτες της portd είναι σε high όταν ανάβουν τα LEDs. Ακόμη στην εγκατάσταση του μικροελεγκτή έχει ένα κύκλωμα ταλαντωτή συνδεδεμένο με αυτό. Ο κύριος ταλαντωτής χρησιμοποιεί 8MHz κρύσταλλο και λειτουργεί ως πρωταρχικός ελεγκτής ταλαντωτής. Χρησιμοποιεί εξωτερικό κρύσταλλο που χρειάζεται για τις εφαρμογές USB. Οι προδιαγραφές USB θέλουν συχνότητα +/- 0,25% για την πλήρη ταχύτητα. Οι εφαρμογές Non-USB μπορούν να χρησιμοποιήσουν εσωτερικό ταλαντωτή. Παρόλα αυτά το Starter Kit έχει διάταξη για ταλαντωτή στα 32KHz. Τέλος ο PIC32 Ethernet Starter Kit χρησιμοποιεί το RJ-45 καλώδιο δικτύου.

4.1.2 Explorer16

Ο Explorer16 είναι μία συσκευή χαμηλού κόστους και χρησιμοποιείται αποτελεσματικά σε συσκευές 32-bit στον PIC32MX. Κάποια από τα χαρακτηριστικά που έχει η συσκευή είναι :

1. 100-pin PIM τα οποία είναι συμβατά με όλες τις εκδόσεις των συσκευών της Microchip.
2. Άμεση ισχύ στα 9VDC η οποία προσφέρει από +3,3 V και +5 V σε ολόκληρη την συσκευή.
3. Δείκτης για τα LEDs ότι είναι σε λειτουργία.
4. RS-232 σειριακή θύρα και συναφή hardware.
5. On-board αναλογικός θερμικός αισθητήρας.
6. Σύνδεση με το USB για επικοινωνία, προγραμματισμός και εκσφαλμάτωση της συσκευής.
7. Βασικό 6-wire In-Circuit Debugger (ICD) με υποδοχή για σύνδεση με ένα MPLAB ICD 2 προγραμματιστή/debugger.
8. Επιλογή του hardware PIM.
9. 2-line για 16 χαρακτήρες στην οθόνη LCD.
10. Τροφοδότηση του PCB για να προστεθούν στην οθόνη γραφικών LCD .
11. Πατάμε τον διακόπτη για reset της συσκευής και καθορίζεται η είσοδος από τον χρήστη.
12. Το Ποντεσιόμετρο για αναλογική είσοδο.
13. 8 ενδεικτικές λυχνίες LEDs.
14. 74HCT4053 πολυπλέκτες για την επιλογή διαμόρφωσης crossover για τις γραμμές της σειριακής επικοινωνίας.
15. Σειριακή EEPROM
16. Ανεξάρτητος κρύσταλλος για την ακρίβεια χρονισμού του μικροελεγκτή (8 MHz) και λειτουργία RTCC στα (32.768 kHz).
17. Πρωτότυπη περιοχή για την ανάπτυξη προσαρμοσμένων εφαρμογών.
18. Υποδοχή και σύνδεση στην άκρη για την συμβατότητα της κάρτας PICtail Plus.
19. Έξι-pin interface για τον Προγραμματιστής PICKIT 3.
20. JTAG υποδοχή για προαιρετική λειτουργία σάρωσης ορίων.



Εικόνα 4.4 Ο Explorer16 69

Εικόνα 4.5 Ο Explorer16

4.1.3 Η Μνήμη EEPROM 25LC256

Η μνήμη είναι προσβάσιμη μέσω του σειριακού interface SPI και συμβατή με το σειριακό bus. Τα σήματα που απαιτούνται για το bus είναι το ρολόι εισόδου (SCK), τα δεδομένα εισόδου serial input (SI) και τα δεδομένα εξόδου serial output (SO). Η πρόσβαση στη συσκευή ελέγχου γίνεται μέσω του Chip Select(CS). Η επικοινωνία με την συσκευή μπορεί να σταματήσει μέσω του hold pin(HOLD). Ενώ η συσκευή είναι σε παύση η μετάβαση στη είσοδο πρέπει να αγνοηθεί με εξαίρεση το Chip Select(CS), επιτρέποντας στο host την υπηρεσία υψηλότερης προτεραιότητας διακοπών.

Η 25XX256 είναι 32.768 byte-σειριακής EEPROM. Σχεδιάστηκε για να επικοινωνεί απευθείας με το SPI και είναι πολύ δημοφιλής σήμερα στην οικογένεια των μικροελεγκτών. Μπορεί επίσης να διασυνδεθεί με μικροελεγκτές που δεν έχουν μια ενσωματωμένη θύρα SPI χρησιμοποιώντας διακριτές I/O γραμμές αρκεί να προγραμματιστεί σωστά το firmware για να ταιριάζει με το πρωτόκολλο SPI.

Η 25XX256 περιέχει ένα 8-bit καταχωρητή διδασκαλίας. Η συσκευή είναι προσβάσιμη μέσω του ακροδέκτη SI, με τα δεδομένα να χρονίζονται στην σχετική αυξανόμενη ακμή του SCK. Ο ακροδέκτης CS πρέπει να είναι low και το host πρέπει να είναι high για το σύνολο της λειτουργία. Όλες οι οδηγίες, οι διευθύνσεις και τα δεδομένα μεταφέρονται πρώτα από το MSB στο LSB.

4.1.4 PICK it 3

1. Lanyard Connection
2. USB port Connection
3. Pin 1 Marker
4. Programming Connector
5. Status LEDS
6. Push Button

Εικόνα 4.6 PICK it 3

Ο pick it 3 programmer/debugger χρησιμοποιείται για τον εντοπισμό σφαλμάτων. Επιπλέον, είναι ένα απλό κύκλωμα, χαμηλού κόστους και ελέγχεται από ένα υπολογιστή που τρέχει MPLAB IDE. Ακόμη χρησιμοποιείται για το υλικό και την ανάπτυξη λογισμικού των μικροελεγκτών της Microchip Pic και σε dsPIC Digital. Ακόμη σε Ελεγκτές Σήματος (DSCs) που βασίζονται σε In-Circuit Serial Programming(ICSP). Εκτός από τις λειτουργίες εντοπισμού σφαλμάτων χρησιμοποιείται και ως προγραμματιστής ανάπτυξης και όχι ως προγραμματιστής παραγωγής.

4.2 Εφαρμογή Διαχείρισης Θυρών

INPUT & OUTPUT ΘΥΡΩΝ

ΚΩΔΙΚΑΣ 1: Leds και Switches-Ανάβει το Led με το πάτημα του Switch

Το παρακάτω πρόγραμμα υλοποιεί είσοδο και την έξοδο των δεδομένων, με τη βοήθεια της θύρας PORTD. Η θύρα PORTD που χρησιμοποιεί ο μικροελεγκτής PIC32 αποτελείται από 13 bits. Τα τρία LEDS που έχει το Starter Kit, LED1, LED2 και LED3 αντιστοιχούν στα bit0, bit1 και bit2 και τα τρία switches SW1, SW2 και SW3 αντιστοιχούν στα bit6, bit7 και bit13 της θύρας PORTD. Ο κώδικας που γράψαμε λαμβάνει είσοδο από τα bits της θύρας PORTD που αντιστοιχούν στους διακόπτες (SW1, SW2, SW3), οπότε αναβοσβήνει το LED1 αν είναι πατημένο το SW1 (δηλαδή το bit6 είναι σε λογικό 0). Αντίστοιχα, αναβοσβήνουν τα LED2 ή LED3 αν είναι πατημένα τα SW2 και SW3 (δηλαδή αν τα bit7 ή bit13 είναι μηδέν). Στον κώδικά μας οι τιμές της εισόδου της PORTD που αντιστοιχούν στις παραπάνω καταστάσεις είναι 0x2080, 0x2040 και 0x00C0 αντίστοιχα. Προκειμένου να αναβοσβήνουν τα leds με ρυθμό που να είναι ορατός στο μάτι, χρησιμοποιούμε μια ρουτίνα καθυστέρησης, την delayMS(). Το όρισμα της συνάρτησης είναι η καθυστέρηση σε millisecond.

```
#include <plib.h>

// Configuration Bits

#pragma config FNOSC    = PRIPLL        // Oscillator Selection

#pragma config FPLLIDIV = DIV_2          // PLL Input Divider (PIC32 Starter
Kit: use divide by 2 only)

#pragma config FPLLMUL  = MUL_20        // PLL Multiplier

#pragma config FPLLODIV = DIV_1         // PLL Output Divider

#pragma config FPBDIV   = DIV_1         // Peripheral Clock divisor

#pragma config FWDTEN   = OFF           // Watchdog Timer

#pragma config WDTPS    = PS1           // Watchdog Timer Postscale
```

```
#pragma config FCKSM     = CSDCMD          // Clock Switching & Fail Safe
Clock Monitor

#pragma config OSCIOFNC = OFF           // CLKO Enable

#pragma config POSCMOD  = XT            // Primary Oscillator

#pragma config IESO     = OFF           // Internal/External Switch-over

#pragma config FSOSCEN  = OFF           // Secondary Oscillator Enable

#pragma config CP       = OFF           // Code Protect

#pragma config BWP      = OFF           // Boot Flash Write Protect

#pragma config PWP      = OFF           // Program Flash Write Protect

#pragma config ICESEL   = ICS_PGx2      // ICE/ICD Comm Channel Select

#pragma config DEBUG    = OFF           // Debugger Disabled for Starter Kit

// application defines

#define SYS_FREQ          (80000000)

// prototype

void DelayMs(unsigned int);

int main(void)

        {

                SYSTEMConfig    (SYS_FREQ,    SYS_CFG_WAIT_STATES    |
        SYS_CFG_PCACHE);

                PORTD=0x00;

                TRISD=0x20C0;

                 // endless loop

                while(1)

                 {

                   if(PORTD==0x2080)

                         {
```

```
                DelayMs(100);
                PORTD=0x01;     // Set LED0 ON
                DelayMs(100);
                PORTD=0x00;
            }               // Set LED0 OFF
        if(PORTD==0x2040)
            {
                DelayMs(100);
                PORTD=0x02;     // Set LED1 ON
                DelayMs(100);
                PORTD=0x00;
            }                       // Set LED1 OFF
        if(PORTD==0x00C0)
            {
                DelayMs(100);
                PORTD=0x04;     // Set LED2 ON
                DelayMs(100);
                PORTD=0x00;     // Set LED2 OFF
            }
        }
        return 0;
        }
void DelayMs(unsigned int msec)
    {
        unsigned int tWait, tStart;
```

```
        tWait=(SYS_FREQ/2000)*msec;

        tStart=ReadCoreTimer();

         while((ReadCoreTimer()-tStart)<tWait);          // wait for the
time to pass

    }
```

ΚΩΔΙΚΑΣ 2: Εναλλακτική χρήση των leds και των switches.

Το παρακάτω πρόγραμμα είναι μία εναλλακτική χρήση των LEDS και των Switches.Ο τρόπος λειτουργίας του προγράμματος είναι παρόμοιος με το παραπάνω πρόγραμμα. Η θύρα PORTD που χρησιμοποιεί ο μικροελεγκτής PIC32 αποτελείται από 13 bits. Τα τρία LEDS που έχει το Starter Kit, LED1, LED2 και LED3 αντιστοιχούν στα bit0, bit1 και bit2 και τα τρία switches SW1, SW2 και SW3 αντιστοιχούν στα bit6, bit7 και bit13 της θύρας PORTD. Ο κώδικας που γράψαμε λαμβάνει είσοδο από τα bits της θύρας PORTD που αντιστοιχούν στους διακόπτες (SW1, SW2, SW3), οπότε αναβοσβήνει το LED1 αν είναι πατημένο τώρα το SW2 (δηλαδή το bit7 είναι σε λογικό 0). Αντίστοιχα, αναβοσβήνουν τα LED2 ή LED3 αν είναι πατημένα τα SW1 και SW3 (δηλαδή αν τα bit6 ή bit13 είναι μηδέν). Στον κώδικά μας οι τιμές της εισόδου της PORTD που αντιστοιχούν στις παραπάνω καταστάσεις είναι 0x2080, 0x2040 και 0x00C0 αντίστοιχα. Επίσης, χρησιμοποιείται η μια ρουτίνα καθυστέρησης, η delayMS() προκειμένου να αναβοσβήνουν τα leds με ρυθμό που να είναι ορατός στο μάτι. Το όρισμα της συνάρτησης είναι η καθυστέρηση σε millisecond.

```
#include <plib.h>

// Configuration Bits

#pragma config FNOSC    = PRIPLL        // Oscillator Selection

#pragma config FPLLIDIV = DIV_2          // PLL Input Divider (PIC32 Starter
Kit: use divide by 2 only)

#pragma config FPLLMUL  = MUL_20        // PLL Multiplier

#pragma config FPLLODIV = DIV_1         // PLL Output Divider

#pragma config FPBDIV   = DIV_1         // Peripheral Clock divisor

#pragma config FWDTEN   = OFF           // Watchdog Timer

#pragma config WDTPS    = PS1           // Watchdog Timer Postscale

#pragma config FCKSM     = CSDCMD          // Clock Switching & Fail Safe
Clock Monitor
```

```
#pragma config OSCIOFNC = OFF          // CLKO Enable
#pragma config POSCMOD  = XT           // Primary Oscillator
#pragma config IESO     = OFF          // Internal/External Switch-over
#pragma config FSOSCEN  = OFF          // Secondary Oscillator Enable
#pragma config CP       = OFF          // Code Protect
#pragma config BWP      = OFF          // Boot Flash Write Protect
#pragma config PWP      = OFF          // Program Flash Write Protect
#pragma config ICESEL   = ICS_PGx2     // ICE/ICD Comm Channel Select
#pragma config DEBUG    = OFF          // Debugger Disabled for Starter Kit
// application defines
#define SYS_FREQ          (80000000)
// prototype
void DelayMs(unsigned int);
int main(void)
{
        SYSTEMConfig(SYS_FREQ,          SYS_CFG_WAIT_STATES          |
SYS_CFG_PCACHE);
        PORTD=0x00;
        TRISD=0x20C0;
         // endless loop
        while(1)
         {
           if(PORTD==0x2080)
        {
                 DelayMs(100);
                PORTD=0x02;     // Set LED2 ON
```

```
                DelayMs(100);
                PORTD=0x00;     // Set LED2 OFF
        }
        if(PORTD==0x2040)
        {
                 DelayMs(100);
                PORTD=0x01;   // Set LED1 ON
                DelayMs(100);
                PORTD=0x00;        // Set LED1 OFF
        }
        if(PORTD==0x00C0)
        {
                DelayMs(100);
                PORTD=0x04;     // Set LED3 ON
                DelayMs(100);
                PORTD=0x00;     // Set LED3 OFF
        }
  }
   return 0;
}
void DelayMs(unsigned int msec)
{
        unsigned int tWait, tStart;
         tWait=(SYS_FREQ/2000)*msec;
        tStart=ReadCoreTimer();
```

```
        while((ReadCoreTimer()-tStart)<tWait);   // wait for the time to pass

}
```

ΚΩΔΙΚΑΣ 3: Τα Leds αναβοσβήνουν μόνα τους

Στο παρακάτω κομμάτι κώδικα τα leds αναβοσβήνουν μόνα τους χωρίς να πατήσουμε τα switches. Μέσα στον βρόγχο της while ορίζουμε μία καθυστέρηση DelayMs(100). Έτσι, βλέπουμε πόσο αργά ή πόσο γρήγορα αναβοσβήνουν τα LEDS ανάλογα με την καθυστέρηση που έχουμε ορίσει. Αν η καθυστέρηση που έχουμε ορίσει είναι μεγάλη για παράδειγμα 100 ή 200 αναβοσβήνουν πιο αργά, αν η καθυστέρηση είναι μικρή για παράδειγμα 20 αναβοσβήνουν πιο γρήγορα. Ανάβει πρώτα το LED1 παραμένει αναμένω και μετά σβήνει. Στη συνέχεια ανάβει το LED2 παραμένει αναμένω και μετά σβήνει και τέλος το LED3 ακολουθεί τη ίδια διαδικασία η οποία επαναλαμβάνεται από την αρχή. Τέλος, ορίζεται η ρουτίνα καθυστέρησης για να γίνεται ορατή όλη αυτή η εναλλαγή των LEDS.

```
#include <plib.h>

// Configuration Bits

#pragma config FNOSC    = PRIPLL        // Oscillator Selection

#pragma config FPLLIDIV = DIV_2          // PLL Input Divider (PIC32 Starter
Kit: use divide by 2 only)

#pragma config FPLLMUL = MUL_20        // PLL Multiplier

#pragma config FPLLODIV = DIV_1         // PLL Output Divider

#pragma config FPBDIV = DIV_1         // Peripheral Clock divisor

#pragma config FWDTEN = OFF           // Watchdog Timer

#pragma config WDTPS    = PS1           // Watchdog Timer Postscale

#pragma config FCKSM     = CSDCMD          // Clock Switching & Fail Safe
Clock Monitor

#pragma config OSCIOFNC = OFF           // CLKO Enable

#pragma config POSCMOD  = XT            // Primary Oscillator

#pragma config IESO     = OFF           // Internal/External Switch-over

#pragma config FSOSCEN  = OFF           // Secondary Oscillator Enable

#pragma config CP       = OFF           // Code Protect
```

```
#pragma config BWP      = OFF          // Boot Flash Write Protect

#pragma config PWP      = OFF          // Program Flash Write Protect

#pragma config ICESEL   = ICS_PGx2     // ICE/ICD Comm Channel Select

#pragma config DEBUG    = OFF          // Debugger Disabled for Starter Kit


// application defines

#define SYS_FREQ          (80000000)

// prototype

void DelayMs(unsigned int);

int main(void)

{

SYSTEMConfig(SYS_FREQ,              SYS_CFG_WAIT_STATES              |
SYS_CFG_PCACHE);

        PORTD=0x00;

        TRISD=0x20C0;

        // endless loop

        while(1)

        {

                DelayMs(100);

                PORTD=0x01;

                DelayMs(100);

                PORTD=0x00;


                DelayMs(100);

                PORTD=0x02;

                DelayMs(100);
```

```
                PORTD=0x00;


                DelayMs(100);
                PORTD=0x04;
                DelayMs(100);
                PORTD=0x00;


    }
  return 0;
}
void DelayMs(unsigned int msec)
{
        unsigned int tWait, tStart;
        tWait=(SYS_FREQ/2000)*msec;
        tStart=ReadCoreTimer();
        while((ReadCoreTimer()-tStart)<tWait);   // wait for the time to pass
}
```

ΚΕΦΑΛΑΙΟ 5ο

ΧΡΟΝΙΣΜΟΣ ΚΑΙ ΣΗΜΑΤΑ ΔΙΑΚΟΠΗΣ

5.1 Περιγραφή Κώδικα με Χρονιστές

Παραδείγματα για τον TIMER1

Παράδειγμα 1:

Χρονική καθυστέρηση με χρήση του timer1 και τα LED1 και LED2 αναβοσβήνουν.

Το παρακάτω πρόγραμμα υλοποιεί μία καθυστέρηση που βασίζεται στον timer1. Ο Timer1 αυξάνει το περιεχόμενο του καταχωρητή TMR1 κατά 1, με βάση την περίοδο του Peripheral bus clock, καθώς και με βάση την τιμή του prescaler. Συγκεκριμένα παρακάτω αναβοσβήνει το led1 και το led2, σβήνει το πρώτο και ανάβει το δεύτερο και αυτό επαναλαμβάνεται. Περιμένει στη while όσο η τιμή του timer1 είναι μικρότερη από την καθυστέρηση που έχουμε, όταν τελειώνει ανάβει το led2 και περιμένει για 0,5s όσο ο TMR1 είναι μικρότερος από την καθυστέρηση. Όταν τελειώνει ξεκινάει από την αρχή.

```
#include <plib.h>
//External Crystal Frequency=8MHz
//Configuration Bits are set in code:
#pragma config POSCMOD=XT //Primary Oscillator Mode=Crystal Oscillator
(XT)
#pragma config FNOSC=PRIPLL //Select Primary Oscillator with PLL Circuit
#pragma config FPLLIDIV=DIV_2 //PLL Input Divider 1:2
#pragma config FPLLMUL=MUL_20 //PLL Multiplier x20
#pragma config FPLLODIV=DIV_1 //System PLL output clock divider 1:1
        System Clock=80MHz
#pragma config FPBDIV=DIV_4 //Periferal Clock divisor 1:4. Periferal Bus
Clock=20MHz
#pragma config FWDTEN=OFF //Watchdog Timer disable
#pragma config CP=OFF //Code Protection OFF
#pragma config BWP=OFF //Boot Flash Write Protect is OFF

#define DELAY 39062 //The relation is:
Real_delay=Period_of_the_Peripheral_bus_Clock*Prescaler value
(256)*DELAY
// We want Real_delay=500ms

main()
{
//Initialize Registers
```

```
//configure all PORTD pins
TRISD=0;     //Configure all RD pins as output
//Output zeroes on PORTD
PORTD=0x0000;//Turn Leds OFF
T1CON=0x8030; //We set TON=1, TCS=0, TCKPS=11 (Prescaler=1:256),
TGATE=0, TSYNC=0, SIDL=0
PR1=0xFFFF; // We set the Period Register to the maximum value

//Main Loop

 while(1) //loop endlessly
 {
        PORTD=0x01;                          //Turn LED1 ON, LED2 OFF
        TMR1=0;
        while(TMR1<DELAY)
        {}                                   //wait 0,5 s
        PORTD=0x02;                          //Turn LED2 ON, LED1 OFF
        TMR1=0;
        while (TMR1<DELAY)
        {}                                   //wait 0,5 s

 }
}
```

Παράδειγμα 2:

Χρονική καθυστέρηση με χρήση του timer1 και μηδενισμός της σημαίας διακοπής.

Το παρακάτω πρόγραμμα υλοποιεί μία καθυστέρηση που βασίζεται στον timer1. Ο TMR1 έχει αρχική τιμή 26476. Ο Timer1 αυξάνει το περιεχόμενο του καταχωρητή TMR1 κατά 1, με βάση την περίοδο του Peripheral bus clock, καθώς και με βάση την τιμή του prescaler. Συγκεκριμένα μπαίνει στην loop ανάβει το LED1 και μηδενίζεται η σημαία διακοπής. Μπαίνει στην while όσο η σημαία είναι 0 και περιμένει για 0,5s. Βγαίνει σβήνει το LED1 και ανάβει το LED2 και μετά μηδενίζεται πάλι η σημαία. Μπαίνει στην δεύτερη while περιμένει για 0,5s και σβήνει το LED2. Όλη η διαδικασία επαναλαμβάνεται από την αρχή.

```
#include <p32xxxx.h>
//External Crystal Frequency=8MHz
//Configuration Bits are set in code:
#pragma config POSCMOD=XT //Primary Oscillator Mode=Crystal Oscillator
(XT)
#pragma config FNOSC=PRIPLL //Select Primary Oscillator with PLL Circuit
#pragma config FPLLIDIV=DIV_2 //PLL Input Divider 1:2
#pragma config FPLLMUL=MUL_20 //PLL Multiplier x20
#pragma config FPLLODIV=DIV_1 //System PLL output clock divider 1:1
        System Clock=80MHz
```

```
#pragma config FPBDIV=DIV_4 //Periferal Clock divisor 1:4. Periferal Bus Clock=20MHz
#pragma config FWDTEN=OFF //Watchdog Timer disable
#pragma config CP=OFF //Code Protection OFF
#pragma config BWP=OFF //Boot Flash Write Protect is OFF

//The time delay is: Real_delay=Period_of_the_Peripheral_bus_Clock*Prescaler value (256)*(65536-TMR1)
// We want Real_delay=500ms. Hence, TMR1=26473

#define T1IF IFS0bits.T1IF  //We define a shortcut (T1IF) for Timer1 Interrupt Flag
main()
{

TRISD=0;     //Configure all PORTD pins as output
//Output zeroes on PORTD
PORTD=0x0000;//Turn Leds OFF
T1CON=0x8030; //We set TON=1, TCS=0, TCKPS=11 (Prescaler=1:256), TGATE=0, TSYNC=0, SIDL=0
PR1=0xFFFF; // We set the Period Register to the maximum value

//Main Loop

 while(1) //loop endlessly
 {
        PORTD=0x01;
        T1IF=0;                         //Turn LED1 ON, LED2 OFF
        TMR1=26473;
        while(T1IF==0)
        {}                      //wait 0,5 s
        PORTD=0x02;                     //Turn LED2 ON, LED1 OFF
        T1IF=0;
        TMR1=26473;
        while (T1IF==0)
        {}                      //wait 0,5 s


        }
}
```

5.2 Περιγραφή Κώδικα με Interrupts

Interrupt 1

Η λειτουργία του παρακάτω προγράμματος βασίζεται σε μία διακοπή που συμβαίνει στον ακροδέκτη RD0 που επάνω στον PIC32 δεν μπορούμε να την διακρίνουμε γιατί το Starter Kit δεν προσφέρει το INT0 push button. Στο πρόγραμμα ορίζεται η υπορουτίνα καθυστέρησης interruptHandler(void) η οποία έχει έναν μετρητή count που αυξάνεται κατά 1, αναστρέφει τα αποτελέσματα της PORTD και μηδενίζει την σημαία διακοπής. Παρακάτω στην main ορίζουμε την είσοδο και την έξοδο και παρατηρούμε ότι ανάβει το LED3. Το INTCONbits.INT0EP=1 ορίζει τον παλμό του INT0. Το priority level ορίζεται από την mINT0SetIntPriority(1); η διακοπή αρχίζει στο 1. Ορίζουμε την συνάρτηση INTEnableSystemSingleVectoredInt(); που μας διευκολύνει στην διαχείριση των διακοπών και τέλος αρχικοποιούμε το mINT0IntEnable(1); το οποίο αντιπροσωπεύει τα bits των διακοπών. Τέλος, η while(1) είναι ατέρμον βρόγχος και περιμένει για την διακοπή .

```
#include <plib.h>

int count=0;

#pragma interrupt InterruptHandler ipl1 vector 0 //INT0 External Interrupt,
Priority level 1.

void InterruptHandler(void) //ISR

{

        count++;        //Increase count at each interrupt

        PORTD=~PORTD; // PORTD toggles at each interrupt

        mINT0ClearIntFlag();

}

main()

{

        TRISD=0x01;          //RD0/INT0 is INPUT

        PORTD=4; //RD2 ON

        //INTCONbits.INT0EP=1; Define pulse edge at INT0

        mINT0SetIntPriority(1);
```

```
        INTEnableSystemSingleVectoredInt();

        mINT0IntEnable(1);


        while(1)        //Endless loop. It waits for the interrupt here.

        {}

}
```

Interrupt 2

Το παρακάτω παράδειγμα εκμεταλλεύεται τον timer2 και παράγει διακοπή με την υπερχείλιση του timer. Ο Timer2 χρησιμοποιεί τον καταχωρητή TMR2 τον οποίο και μηδενίζει. Επιπλέον, χρησιμοποιεί τον καταχωρητή περιόδου PR2 τον λεγόμενο postscaler ο οποίος παίρνει την μέγιστη περίοδο. Η τιμή του prescaler είναι στα 256. Μέσα στη συνάρτηση έχουμε έναν μετρητή count που αυξάνεται κατά 1, στην PORTD εμφανίζονται τα αποτελέσματα του count που έχει αυξηθεί και τέλος έχουμε και την συνάρτηση mT2ClearIntFlag(); για να καθαρίζει η σημαία. Στην main() ορίζω έξοδο την TRISD=0 και PORTD=0. Μηδενίζοντας τον TMR2, μπαίνει στην loop.

```
#include <p32xxxx.h>

#include <plib.h> //peripheral library

int count=0;

#pragma interrupt InterruptHandler ipl1 vector 0 //we declare an interrupt handler function with priority level 1 and at the default interrupt vector (0)

void InterruptHandler(void)  //Interrupt handler routine returns no value and receives no parameters

{

                count++;

                PORTD=count; //Increase port count at each interrupt

                mT2ClearIntFlag(); //use C32 compiler macro to clear int flag

}

main()
```

```
{
        TRISD=0;
        PORTD=0;
        /*Timer initiallization*/
        PR2=0xFFFF; //set period register at max value
        T2CON=0x8070; //Timer2 Prescaler 1:256
        /*Interrupt settings*/
        mT2SetIntPriority(1); //(T2IP=1)
        INTEnableSystemSingleVectoredInt();    //Enable    single    vectored
Interrupts
        mT2IntEnable(1); //Enable T2 interrupts with priority level 1 (T2IE=1)
        TMR2=0;
        while(1)
        {}
}
```

ΚΕΦΑΛΑΙΟ 6ο

ΣΕΙΡΙΑΚΗ ΕΠΙΚΟΙΝΩΝΙΑ

6.1 Ασύγχρονη Σειριακή Επικοινωνία

Παρακάτω περιγράφεται αναλυτικά ο κώδικας που χρησιμοποιείται για την εφαρμογή της ασύγχρονης επικοινωνίας. Αρχικά αναφέρονται οι λειτουργίες των συναρτήσεων. Παρακάτω φαίνεται η σύνθεση του κώδικα καθώς και το πρόγραμμα που χρησιμοποιείται για την αποστολή και λήψη δεδομένων. Όλη αυτή η διαδικασία γίνεται μέσω του Hyper Terminal.

Ο κώδικας παρακάτω χρησιμοποιείται ώστε να αρχικοποιηθεί το UART module, υπολογίζεται ο ρυθμός της γεννήτριας το baud rate και χρησιμοποιούνται τα I/O pins για την χειραψία. Ο κώδικας παρακάτω είναι της χειραψίας.

```
 void initU2(void)

{

        U2BRG=BRATE; //initialize the baud rate generator

        U2MODE=U_ENABLE;//initialize the UART module

        U2STA=U_RX_TX; //enable the trasmitter

        TRTS=0; //make RTS an output pin

        RTS=1; //set RTS default status (not ready)

}
```

Παραπάνω στην χειραψία αρχικοποιούμε τον ρυθμό γεννήτριας, αρχικοπούμε την UART module και έτσι ενεργοποιείται ο πομπός. Ορίζεται το pin εξόδου στην έξοδο TRTS=0 και προεπιλέγεται η κατάσταση RTS=1.

Ο κώδικας της αποστολής και λήψης δεδομένων υλοποιείται με τα παρακάτω βήματα.

```
 int putU2(int c)

{

        while(CTS);  //wait for !CTS, clear to send
```

```
        while(U2STAbits.UTXBF);  //wait while Tx buffer full

        U2TXREG=c;

        return c;

}
```

Στο παραπάνω κομμάτι περιμένουμε το clear to send να είναι έτοιμο, είναι σε active low και περιμένουμε να γίνει high. Ακόμη περιμένουμε μέχρι η είσοδος του buffer να είναι (full-πλήρης) και παραπάνω το U2TXREG=C παίρνει την τιμή του C και την επιστρέφει.

```
char getU2(void)

{

        RTS=0; //assert Request To Send !RTS

        while(U2STAbits.URXDA==0); //wait for a new char to arrive

        RTS=1;

        return U2RXREG; //read char from receive buffer

}
```

Ειδοποιούμε το τερματικό ότι είμαστε έτοιμοι να λάβουμε το σήμα RTS (active low) RTS=0, στον κώδικα επιβεβαιώνεται το αίτημα για να στείλει. Μετά περιμένει να φτάσει ο νέος χαρακτήρας στον buffer λήψης και γίνεται έλεγχος της σημαίας URXDA στο εσωτερικό της UART2 του καταχωρητή Status έπειτα το RTS=1 γίνεται 1 και διαβάζει και επιστρέφει τον χαρακτήρα που έχει λάβει από την buffer.

Παρακάτω μέσα στην main γίνεται ένα test για την σειριακή θύρα και συγκεκριμένα για την αρχικοποίησή της. Ο κώδικας παρακάτω αρχικοποιεί την σειριακή θύρα και στέλνει άμεσα τον χαρακτήρα. Μπαίνει στην loop περιμένει για τον χαρακτήρα όπου στέλνεται άμεσα και πληκτρολογώντας στο τερματικό φαίνεται ο κάθε χαρακτήρας πίσω στην οθόνη.

```
main()

{



char c;
```

```
        //1.init the UART2 serial port
        initU2();
        //2.prompt
        putU2('>');
        //3.main loop
while(1)
        {
        //3.1 wait for a character
        c=getU2();
        //3.2 echo the character
        putU2(c);
        }
}
```

Παρακάτω είναι ο κώδικας ολοκληρωμένος

```
#include <p32xxxx.h>
//External Crystal Frequency=8MHz
//Configuration Bits are set in code:
#pragma config POSCMOD=XT //Primary Oscillator Mode=Crystal Oscillator (XT)
#pragma config FNOSC=PRIPLL //Select Primary Oscillator with PLL Circuit
#pragma config FPLLIDIV=DIV_2 //PLL Input Divider 1:2
#pragma config FPLLMUL=MUL_18 //PLL Multiplier x18
#pragma config FPLLODIV=DIV_1 //System PLL output clock divider 1:1 System Clock=72MHz
#pragma config FPBDIV=DIV_2 //Periferal Clock divisor 1:2. Periferal Bus Clock=36MHz
```

```
#pragma config FWDTEN=OFF //Watchdog Timer disable
#pragma config CP=OFF //Code Protection OFF
#pragma config BWP=OFF //Boot Flash Write Protect is OFF
//#include "test_new28.h"
#define CTS _RF12
#define RTS _RF13
#define TRTS TRISFbits.TRISF13
#define BRATE 77 //115200 bps
#define U_ENABLE 0x8008
#define U_RX_TX 0x1400 //Enable Receive and Transimt module
void initU2(void)
{
	U2BRG=BRATE; //initialize the baud rate generator
	U2MODE=U_ENABLE;//initialize the UART module
	U2STA=U_RX_TX; //enable the trasmitter
	TRTS=0; //make RTS an output pin
	RTS=1; //set RTS default status (not ready)
}
int putU2(int c)
{
	while(CTS); //wait for !CTS, clear to send
	while(U2STAbits.UTXBF); //wait while Tx buffer full
	U2TXREG=c;
	return c;
}
```

```
char getU2(void)
{
	RTS=0; //assert Request To Send !RTS
	while(U2STAbits.URXDA==0); //wait for a new char to arrive
	RTS=1;
	return U2RXREG; //read char from receive buffer
}
main()
{

char c;
	//1.init the UART2 serial port
	initU2();
	//2.prompt
	putU2('>');
	//3.main loop
while(1)
	{	//3.1 wait for a character
	c=getU2();
	//3.2 echo the character
	putU2(c);
	}
}
```

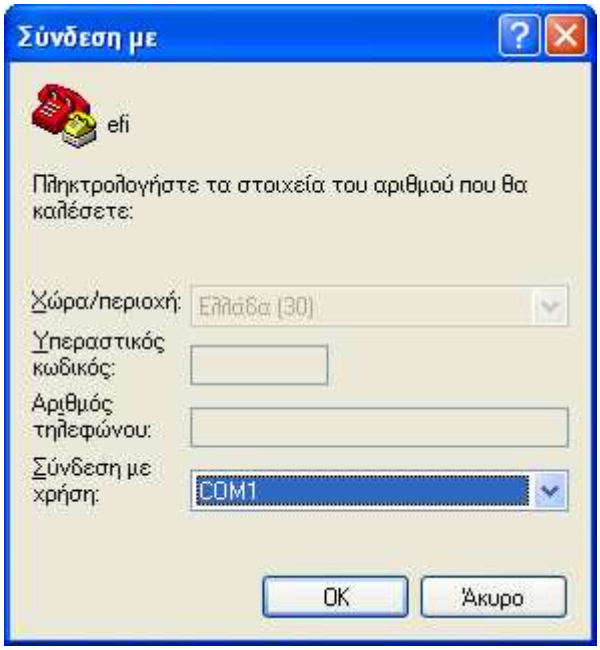
Το παραπάνω κομμάτι κώδικα αναφέρεται στον τρόπο αποστολής και λήψης δεδομένων μέσω της ασύγχρονης επικοινωνίας. Στο θεωρητικό κομμάτι αναφέρθηκε τι ακριβώς συμβαίνει και εδώ είναι η σύνθεση του κώδικα και ο τρόπος που λειτουργεί χρησιμοποιώντας τον Hyper Terminal. Για την

αποστολή και την λήψη των δεδομένων θα δούμε τον τρόπο λειτουργίας του Hyper Terminal δίνοντας είσοδο από το πληκτρολόγιο και τα αποτελέσματα φαίνονται στην οθόνη, χρησιμοποιώντας ταυτόχρονα και τον Explorer 16.

Ανοίγουμε τον Hyper Terminal από Έναρξη->Προγράμματα->Βοηθήματα->Επικοινωνίες->Hyper Terminal.

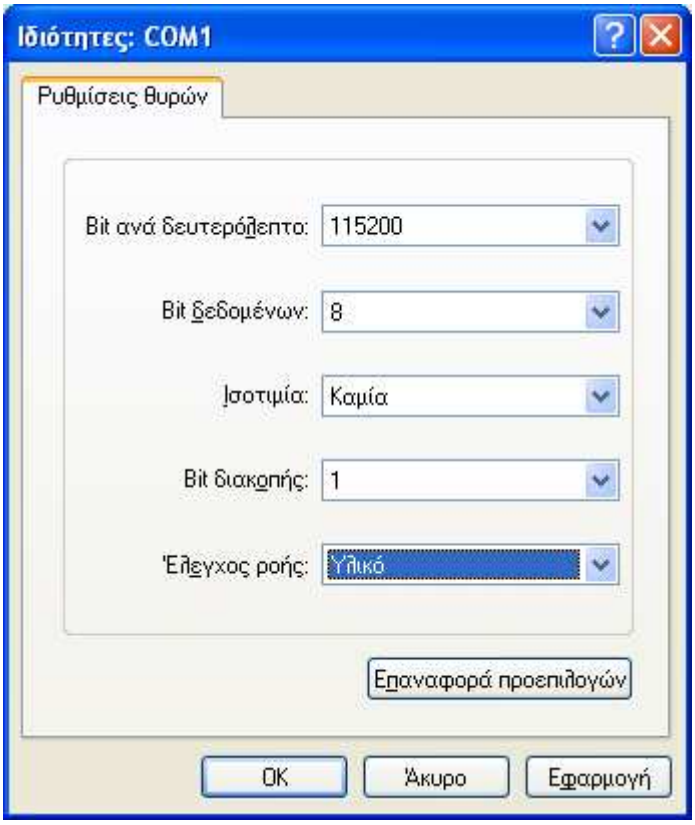
Εικόνα 6.1 Παράθυρο Περιγραφής σύνδεσης

Γράφουμε ένα όνομα και επιλέγουμε το πρώτο εικονίδιο και πατάμε OK και εμφανίζεται το παρακάτω παράθυρο.



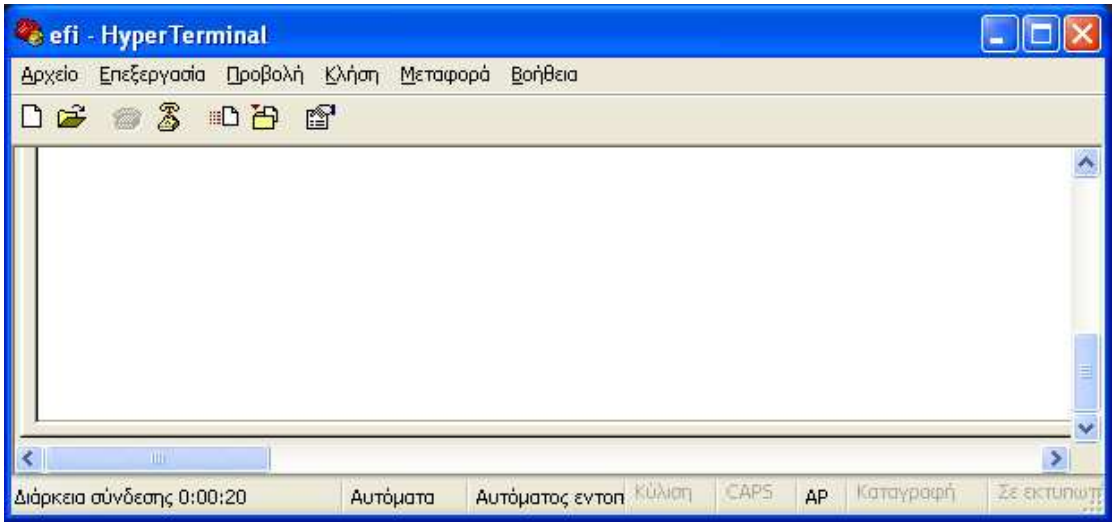
Εικόνα 6.2 Παράθυρο Σύνδεσης

Παρακάτω πατάμε OK και εμφανίζεται ένα ακόμη παράθυρο και κάνουμε τις ρυθμίσεις που είναι απαραίτητες.



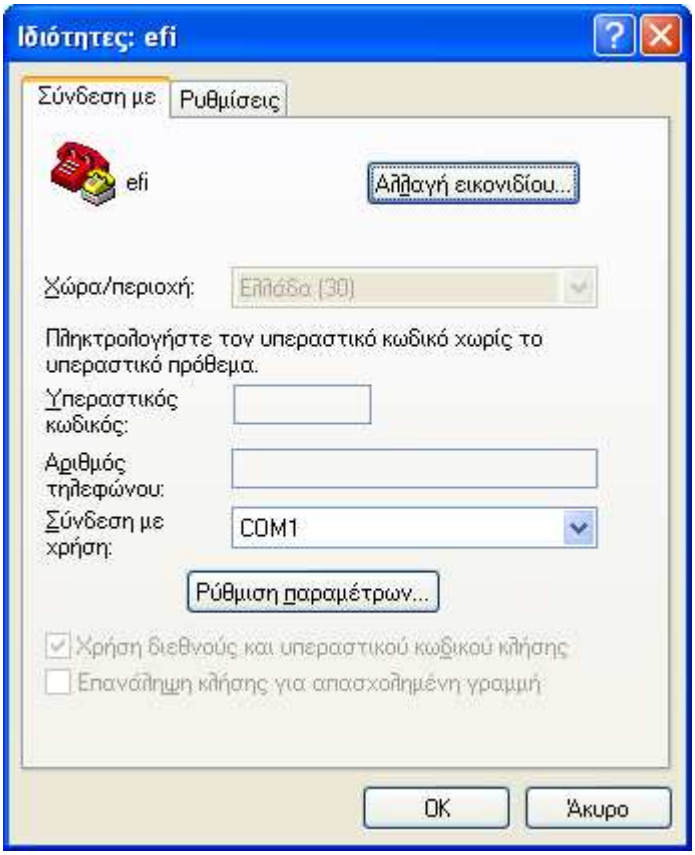
Εικόνα 6.3 Παράθυρο Ιδιότητες COM1

Πατάμε εφαρμογή και OK και έχουμε το παράθυρο που θα φαίνονται τα δεδομένα που στέλνουμε και λαμβάνουμε.



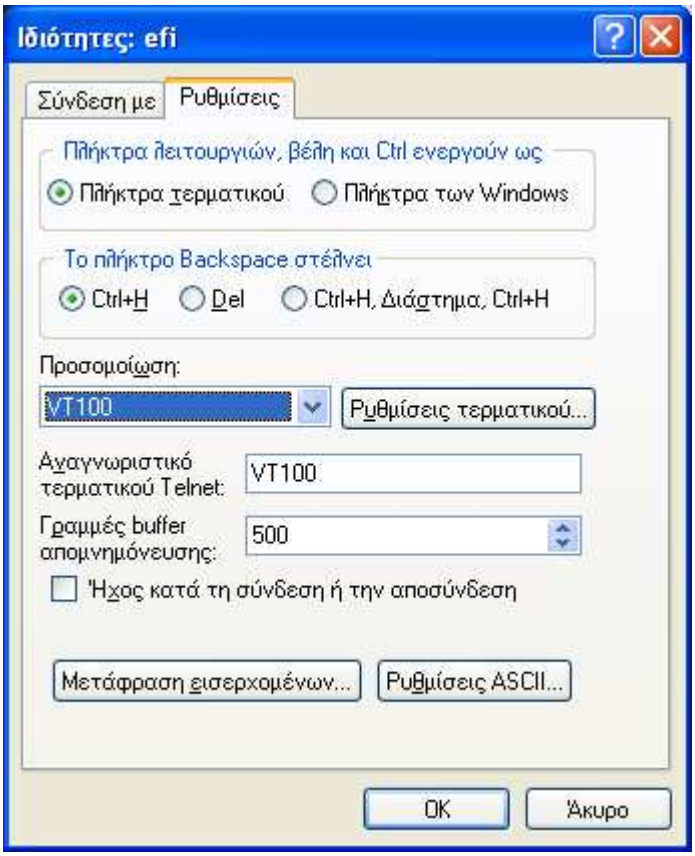
Εικόνα 6.4 Παράθυρο Hyper Terminal

Μετά πατάμε στο παραπάνω παράθυρο Αρχείο->Ιδιότητες και μου εμφανίζει το παρακάτω.



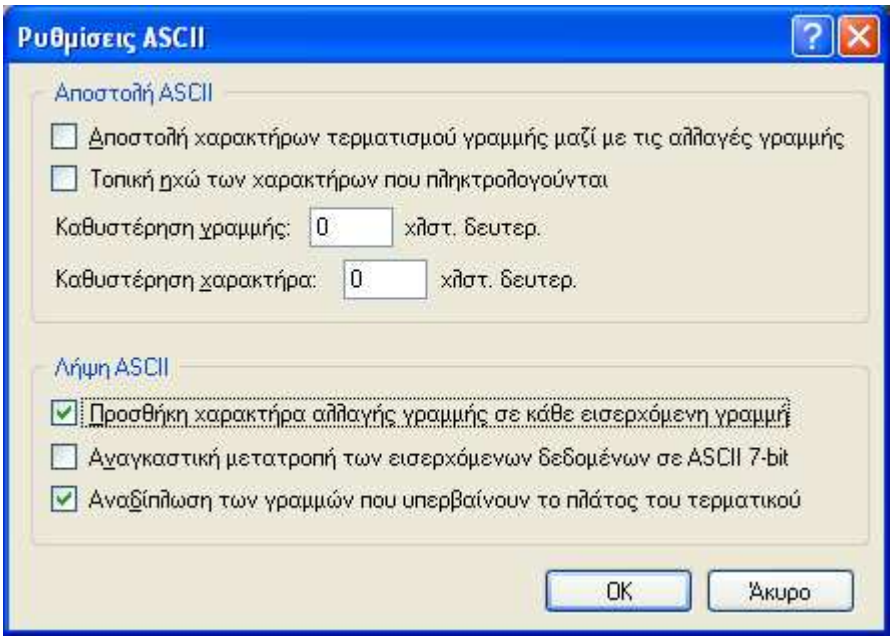
Εικόνα 6.5 Παράθυρο Ιδιότητες

Και επιλέγουμε την δεύτερη καρτέλα και κάνουμε τις παρακάτω ρυθμίσεις.



Εικόνα 6.6 Παράθυρο Ρυθμίσεις

Παρακάτω πατάμε Ρυθμίσεις ASCII στην συγκεκριμένη καρτέλα και έχουμε τις εξής ρυθμίσεις.



Εικόνα 6.7 Παράθυρο Ρυθμίσεις ASCII

Και τέλος πατάμε το OK και στο παράθυρο που έχουμε αν δώσουμε χαρακτήρα θα δώσει χαρακτήρα, ανάλογα τι έχουμε ορίσει στον κώδικα.

Με αυτό τον τρόπο γίνεται η ασύγχρονη επικοινωνία.

ΚΕΦΑΛΑΙΟ 7ο

ΣΥΜΠΕΡΑΣΜΑΤΑ-ΠΡΟΟΠΤΙΚΕΣ

Έχοντας δημιουργήσει τις παραπάνω εφαρμογές και περιγράψει το σύστημά μας καταλήγουμε στο συμπέρασμα ότι είναι ένα σύστημα εύχρηστο με πολλές δυνατότητες.

Ο μικροελεγκτής PIC32MX795F512L της Microchip είναι ένα σύστημα που παρέχει πολλές δυνατότητες στον χρήστη μαζί με την χρήση του MPLAB που είναι ένα ευρέως διαδεδομένο λογισμικό, με πλήθος συναρτήσεων για πολλά είδη επεξεργασίας.

Η κατανομή του χώρου της μνήμης του συστήματος είναι σημαντικό γιατί προσφέρει την δυνατότητα για τον τρόπο που θα τοποθετηθούν οι μεταβλητές στην μνήμη και η εύρεση μιας φυσικής διεύθυνσης η οποία διατίθενται για κάθε αντικείμενο στο χώρο μνήμης. Επίσης, σημαντικό ρόλο έπαιξε και η μνήμη του προγράμματος. Επιπλέον, τα Configuration Bits που χρησιμοποιούμε στις εφαρμογές διευκόλυναν στην αρχική ρύθμιση του καταχωρητή OSCCON και περιέχουν και την αρχική διαμόρφωση της συσκευής.

Ένα μέρος της ευελιξίας του μικροελεγκτή βρίσκεται στα περιφερειακά και στο μεγάλο πλήθος ακροδεκτών για είσοδο και έξοδο δεδομένων. Το σύστημα έχει την δυνατότητα να διαχειρίζεται την είσοδο και την έξοδο των δεδομένων μέσω της PORTD και να υλοποιεί καθυστερήσεις μέσω των timer1 και του timer2. Επιπλέον, ο PIC32 προσφέρει έναν εναλλακτικό μηχανισμό που χρησιμοποιεί διακοπές vectored και σύνολα multiple register. Η οικογένεια PIC32MX προσφέρει 64-vector table και 2 πλήρεις σειρές από working registers που, μπορούν να απαλλαχθούν αυτόματα.

Ας σημειωθεί ότι η οικογένεια PIC32MX προσφέρει 7 περιφερειακά επικοινωνίας που έχουν σχεδιαστεί για να βοηθούν όλες τις εφαρμογές ελέγχου. Μέχρι 6 από αυτά είναι συσκευές σειριακής επικοινωνίας. Μεταδίδουν και λαμβάνουν κάθε φορά ένα κομμάτι των πληροφοριών. Η UART που είναι ένα κομμάτι της επικοινωνίας κάνει τον μικροελεγκτή δυνατό σε εφαρμογές επικοινωνίας με άλλες συσκευές. Ακόμη ο μικροελεγκτής επικοινωνεί με τον Η/Υ μέσω θύρας USB και μέσω Ethernet σε σχέση με παλαιότερους μικροελεγκτές.

Τέλος, μία προοπτική επέκτασης της εργασίας είναι η επικοινωνία WiFi ανάμεσα στον μικροελεγκτή και σε host υπολογιστή καθώς και επικοινωνία με οθόνη γραφικών.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Lucio Di Jasio , Programming 32-bit Microcontrollers in c.

2. John L. Hennessy & David A. Patterson, Αρχιτεκτονική Υπολογιστών.

3. Ιωάννης Ν.Έλληνας, Αρχιτεκτονική Υπολογιστών.

4. Σ. Καζαρλής, Αρχιτεκτονική Υπολογιστών-Σημειώσεις για το ΤΕΙ Σερρών(2004).

5. Ι. Καλόμοιρος, Υλικό και Λογισμικό για ΣΥΣΤΗΜΑΤΑ ΠΡΑΓΜΑΤΙΚΟΥ ΧΡΟΝΟΥ -Σημειώσεις για το ΤΕΙ Σερρών(2008).

6. Σ.Μπουλταδάκης, Γ. Πατουλίδης, Ν. Ασημόπουλος, ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ ΜΙΚΡΟΕΛΕΓΚΤΩΝ, FPGA και CPLD: ΕΠΙΛΕΓΜΕΝΕΣ ΕΦΑΡΜΟΓΕΣ.

7. Ιστοσελίδα της εταιρίας Microchip: www.microchip.com

8. Ιστοσελίδα της Wikipedia: www.wikipedia.org