ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΗΛΕΚΤΡΙΚΗΣ ΙΣΧΥΟΣ

Εφαρμογή των DSP σε διατάξεις ηλεκτρονικών ισχύος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Στυλιανός Φ. Ορφανός

Επιβλέπων: Στέφανος Ν. Μανιάς

Καθηγητής Ε.Μ.Π.

Αθήνα, Μάιος 2015

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΗΛΕΚΤΡΙΚΗΣ ΙΣΧΥΟΣ

Εφαρμογή των DSP σε διατάξεις ηλεκτρονικών ισχύος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Στυλιανός Φ. Ορφανός

Επιβλέπων: Στέφανος Ν. Μανιάς
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 5η Μαΐου 2015

……	……	……
Στ. Ν. Μανιάς	Α. Κλαδάς	Στ. Παπαθανασίου
Καθηγητής	Καθηγητής	Αναπληρωτής Καθηγητής

Αθήνα, Μάιος 2015

……

Στυλιανός Φ. Ορφανός

Διπλωματούχος Ηλεκτρολόγος Μηχανικός & Μηχανικός Υπολογιστών Ε.Μ.Π.

ΠΕΡΙΛΗΨΗ

Τα ενσωματωμένα συστήματα αποτελούν σημαντικό βοήθημα στις περισσότερες εκφάνσεις της σύγχρονης βιομηχανίας. Συγκεκριμένα, στις διατάξεις Ηλεκτρονικών Ισχύος χρησιμοποιούνται με τη μορφή των Ψηφιακών Ελεγκτών Σήματος (DSC) και των συσκευών προγραμματιζόμενης λογικής (FPGA). Η παρούσα διπλωματική αποτελεί εγχειρίδιο για τον προγραμματισμό του DSC TMS320F28335 της Texas Instruments στη C γλώσσα προγραμματισμού. Κατά την πορεία των κεφαλαίων παρουσιάζονται οι δυνατότητες και οι διαφορετικές μονάδες του DSC, θεωρητικά αλλά και με την δημιουργία πειραματικών εφαρμογών. Τα συμπεράσματα και οι παρατηρήσεις που προκύπτουν είναι χρήσιμα και για τον προγραμματισμό DSC άλλων κατασκευαστών. Στο τέλος της διπλωματικής, αυτές οι εφαρμογές συνδυάζονται για την κατασκευή των αλγορίθμων της Διαμόρφωσης Ημιτονοειδούς Εύρους Παλμών (SPWM) για μονοφασική διάταξη ανοιχτού βρόχου και της Διαμόρφωσης Εύρους Παλμών με βάση τα Χωρικά Διανύσματα Τάσης του Αντιστροφέα (Space Vector Pulse width modulation-SVPWM) για τριφασική διάταξη ανοιχτού βρόχου.

<u>Λέξεις κλειδιά:</u>

Ενσωματωμένα συστήματα, DSP, DSC, TMS320F28335, Texas Instruments, SPWM, SVPWM, εγχειρίδιο προγραμματισμού στη C

ABSTRACT

Embedded systems are a significant asset to the contemporary industry. In power electronics they are utilized mostly in systems with Digital Signal Controllers (DSCs) or Field-Programmable Gate Arrays (FPGAs) as the heart of these systems. This diploma thesis is a programming manual for the Texas Instrument's DSC, TMS320F28335, in C programming language. The properties and the various modules of the DSC are explained throughout the diploma thesis, both theoretically and with experimental results. The programming method, which is followed in this diloma thesis can be used to program DSC's, that are built from other manufacturers. At the end of the diploma thesis, all aforementioned applications are combined to construct the algorithm for open-loop, single-phase, Sinusoidal Pulse Width Modulation (SPWM) and for open-loop, three-phase, Space Vector Pulse width modulation(SVPWM).

Key Words:

Embedded Systems, DSP, DSC, TMS320F28335, Texas Instruments, SPWM, SVPWM, programming manual in C

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να εκφράσω τις ευχαριστίες μου στον επιβλέποντα καθηγητή κ. Στέφανο Μανιά για τη δυνατότητα που μου έδωσε να ασχοληθώ και να φέρω εις πέρας ένα τόσο ενδιαφέρον θέμα. Επίσης, ευχαριστώ θερμά τον υποψήφιο διδάκτορα Γεώργιο Καμπίτση για την πολύτιμη βοήθειά του κατά την εκπόνηση της παρούσας διπλωματικής. Τέλος, θέλω να ευχαριστήσω την οικογένειά μου και ιδιαίτερα τους γονείς μου για τη στήριξή τους όλα αυτά τα χρόνια.

ΕΙΣΑΓΩΓΗ

Στόχος Διπλωματικής

Η παρούσα διπλωματική έχει ως σκοπό να εκπαιδεύσει προπτυχιακούς και μεταπτυχιακούς φοιτητές των Ηλεκτρονικών ισχύος στον προγραμματισμό του DSC TMS320F28335. Αυτό επιτυγχάνεται μέσω πολλαπλών εφαρμογών, οι οποίες σταδιακά βοηθούν το φοιτητή να κατασκευάσει προγράμματα που υλοποιούν γνωστές τεχνικές διαμόρφωσης εύρους παλμών. Παρά το γεγονός ότι αφορά το συγκεκριμένο DSC, η προγραμματιστική λογική είναι κοινή και για άλλα DSC από διαφορετικούς κατασκευαστές. Το μόνο που διαφοροποιείται είναι κάποιες ειδικές εντολές.

Δομή Κεφαλαίων

Τα κεφάλαια 1 έως 3 έχουν θεωρητικό χαρακτήρα και λειτουργούν εισαγωγικά στο αντικείμενο των ενσωματωμένων συστημάτων, αλλά και για τις δύο τεχνικές διαμόρφωσης εύρους παλμών που θα μελετηθούν στη συνέχεια (SPWM και SVPWM).

Τα κεφάλαια 4 έως 10 αποτελούν τον πυρήνα της διπλωματικής και εισάγουν τον αναγνώστη σταδιακά στον προγραμματισμό, στα περιφερειακά και στις λειτουργίες του DSC. Τελικά αυτή η γνώση θα συνδυαστεί για την κατασκευή των αλγορίθμων του SPWM και SVPWM στα κεφάλαια 11 και 12 αντίστοιχα. Κάθε κεφάλαιο αυτής της ομάδας αρχίζει με τη θεωρητική μελέτη του εκάστοτε αντικειμένου. Αυτό περιλαμβάνει κυρίως την επεξήγηση της λειτουργίας των καταχωρητών της κάθε μονάδας. Στη συνέχεια, υπάρχει εφαρμογή, η οποία αξιοποιεί τη θεωρία του κεφαλαίου πρακτικά. Τα πειραματικά αποτελέσματα και ο κώδικας της εφαρμογής παρατίθενται στο τέλος του κεφαλαίου. Θεωρείται σκόπιμο ο λυμένος κώδικας να χρησιμοποιείται ως επιβεβαίωση και όχι ώς πηγή της λύσης.

Συνοπτική παρουσίαση κεφαλαίων

1. **Εισαγωγή στα ενσωματωμένα υπολογιστικά συστήματα:** Παρουσίαση των ενσωματωμένων υπολογιστικών συστημάτων από τον Μικροεπεξεργαστή (μP) μέχρι και τον Ψηφιακό ελεγκτή σήματος (DSC), ο οποίος είναι και το αντικείμενο της παρούσας διπλωματικής. Στο τέλος του κεφαλαίου γίνεται μία μικρή αναφορά στις συσκευές προγραμματιζόμενης λογικής (FPGA) και ποιοτική σύγκριση μεταξύ τους και των DSC.

2. **Αρχιτεκτονική και περιφερειακές συσκευές του DSC TMS320F28335:** Θεωρητική μελέτη της αρχιτεκτονικής του DSC που χρησιμοποιείται στα πλαίσια της διπλωματικής.

3. Θεωρητική παρουσίαση των τεχνικών διαμόρφωσης SPWM και SVPWM: Το κεφάλαιο αυτό παρουσιάζει θεωρητικά τον τελικό στόχο της διπλωματικής.

4. Ψηφιακή Είσοδος/Εξοδος του DSC TMS320F28335: Σε αυτό το σημείο αρχίζει η επαφή του αναγνώστη με τον DSC. Αρχικά παρουσιάζεται το κομμάτι της ψηφιακής Εισόδου/Εξόδου, η οποία είναι πολύ σημαντική για τη δημιουργία ενός λειτουργικού κώδικα. Με αυτό μπορούμε να δώσουμε σήματα στον DSC μέσω μπουτόν, ή να στείλουμε σήματα από τον DSC σε led με σκοπό την αποσφαλμάτωση του κώδικα. Στο τέλος του κεφαλαίου υπάρχει εφαρμογή, η οποία αξιοποιεί την είσοδο και την έξοδο του DSC για τις ανάγκες ενός ψηφιακού χρονομετρητή.

5. Σύστημα διακοπών διεργασίας του DSC TMS320F28335: Το σύστημα διακοπών είναι η δυνατότητα του επεξεργαστή να διακόψει την κανονική ροή κώδικα και να τρέξει μία ανεξάρτητη διαδικασία. Στο κεφάλαιο ο αναγνώστης ενημερώνεται για το σύστημα διακοπών του F28335 και παρουσιάζονται, μέσω παραδειγμάτων, δύο σημαντικές πηγές διακοπών. Η περιοδική διακοπή από εσωτερικό χρονιστή του DSC και η εξωτερική διακοπή από εξωτερικό ψηφιακό σήμα.

6. ePWM module - Μονάδα Βάσης Χρόνου: Τα κεφάλαιο 6 έως 8 αφορούν τη μονάδα ePWM του DSC. Αρχικά παρουσιάζεται το μέρος που είναι υπεύθυνο για τη συχνότητα και τη μορφή του φορέα, ενώ στην πορεία του κεφαλαίο λύνονται κάποιες τεχνικές δυσκολίες που εντοπίστηκαν στην πορεία της μελέτης. Το κεφάλαιο κλείνει με τη δημιουργία συνάρτησης, η οποία αντιστοιχίζει τις παραμέτρους του ePWM module σε μία επιλεγμένη συχνότητα.

7. ePWM module - Μονάδα σύγκρισης και διακοπής διεργασιών: Στη συνέχεια της μελέτης της μονάδας ePWM υπάρχει το κομμάτι, στο οποίο καθορίζονται τα σημεία τομής του φορέα με την αναφορά, καθώς και η επίδραση που θα έχουν αυτά στην έξοδο του DSC. Ο υπολογισμός των σημείων γίνεται on the fly και για αυτό το λόγο ο DSC επιτρέπει τον ορισμό διακοπών εντός του κύκλου του φορέα, ώστε να υπολογιστεί το επόμενο σημείο τομής. Στην εφαρμογή, στο τέλος του κεφαλαίου, παράγεται μία τυχαία ακολουθία παλμών και η συμπληρωματική της. Σε πραγματική εφαρμογή οι παλμοί θα οδηγούσαν ημιαγωγικούς διακόπτες της ίδιας ημιγέφυρας.

8. ePWM module - Προστασία διατάξεων των ημιαγωγικών διακοπτών ηλεκτρονικών ισχύος: Σύντομο κεφάλαιο για τις μονάδες του DSC, οι οποίες συνδράμουν στην προστασία των διατάξεων Ηλεκτρονικών Ισχύος. Το Dead Band εισάγει μία μικρή καθυστέρηση στις ακμές των παλμοσειρών, ώστε να εξασφαλιστεί ότι δύο διακόπτες της ίδιας ημιγέφυρας δε θα άγουν την ίδια χρονική στιγμή. Το Trip Zone είναι μια μονάδα, η οποία σταματάει τους παλμούς ελέγχου προς τους διακόπτες όταν παρουσιαστεί σφάλμα. Στην ουσία αποτελεί μία εξειδικευμένη διακοπή με πηγή ένα εξωτερικό σήμα trip.

9. eCAP module - Μονάδα σύλληψης ψηφιακών σημάτων: Το eCAP (enhanced capture) είναι η μονάδα του F28335, η οποία επιτρέπει τη χρησιμοποίηση του DSC ως ψηφιακό παλμογράφο. Αποτελεί σημαντικό εργαλείο για την αποσφαλμάτωση ρων προγραμμάτων. Στο τέλος του κεφαλαίο υπάρχει σχετική εφαρμογή, η οποία αναδεικνύει τις δυνατότητες του eCAP στη μελέτη περιοδικών παλμοσειρών.

10. Μετατροπέας Αναλογικού σήματος σε Ψηφιακό (ADC module): Ο F28335 επιτρέπει τη ψηφιοποίηση αναλογικών τιμών. Η μονάδα μετατροπής Αναλογικού σήματος σε Ψηφιακό είναι χρήσιμη στην περίπτωση πραγματικών διατάξεων ηλεκτρονικών ισχύος όταν εισάγεται σε αυτά ανάδραση. Το μονοφασικό SPWM που υλοποιείται στο Κεφάλαιο 11, αξιοποιεί την μονάδα ADC για τον καθορισμό του πλάτους του σήματος αναφοράς. Στο τέλος του παρόντος κεφαλαίου παρουσιάζεται μία απλή εφαρμογή, στην οποία σήματα της γεννήτριας του εργαστηρίου αναπαρίστανται μέσω του DSC στο Code Composer Studio.

11. Υλοποίηση μονοφασικού SPWM στον F28335: Σύνδεση όλων των προηγούμενων εφαρμογών και δημιουργία προγράμματος που παράγει παλμούς σύμφωνα με την τεχνική διαμόρφωσης Ημιτονοειδούς Εύρους Παλμών (SPWM) για μονοφασική διάταξη ανοιχτού βρόχου.

12. Υλοποίηση Space Vector PWM (SVPWM) στον F28335: Σύνδεση όλων των προηγούμενων εφαρμογών και δημιουργία προγράμματος που παράγει παλμούς σύμφωνα με την τεχνική διαμόρφωσης Εύρους Παλμών με βάση τα Χωρικά Διανύσματα Τάσης του Αντιστροφέα (Space Vector Pulse width modulation-SVPWM) για τριφασική διάταξη ανοιχτού βρόχου.

13. Συμπεράσματα: Κλείσιμο της διπλωματικής με συμπεράσματα και παρατηρήσεις.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ ΣΤΑ ΕΝΣΩΜΑΤΩΜΕΝΑ ΥΠΟΛΟΓΙΣΤΙΚΑ ΣΥΣΤΗΜΑΤΑ

1.1 Εισαγωγή

Ενσωματωμένο σύστημα (embedded system) είναι ένα υπολογιστικό σύστημα, το οποίο επιτελεί μία συγκεκριμένη λειτουργία και μπορεί να είναι μέρος ενός μεγαλύτερου ηλεκτρικού ή μηχανικού συστήματος. Τα ενσωματωμένα συστήματα αξιοποιούνται τόσο σε απλές φορητές εφαρμογές (πχ. ψηφιακά ρολόγια) όσο και σε πολύπλοκα συστήματα (πχ. υβριδικά αυτοκίνητα)

Στο πρώτο μέρος του κεφαλαίου (τμήμα 1.2 έως 1.6) παρουσιάζεται η εξέλιξη των ενσωματωμένων συστημάτων από τον μικρο-επεξεργαστή μέχρι τους ευρέως χρησιμοποιούμενους, σε εφαρμογές πραγματικού χρόνου, ψηφιακούς επεξεργαστές σήματος (DSC). Στη συνέχεια (τμήμα 1.7) εκθέτονται συνοπτικά κάποιες ηλεκτρολογικές εφαρμογές των DSC.

Στο δεύτερο μέρος του κεφαλαίου (τμήματα 1.8, 1.9) εξετάζεται συνοπτικά ο ανταγωνιστής των DSC, οι Συσκευές Προγραμματιζόμενης λογικής FPGA (Field Programmable Gate Arrays). Πρόκειται για συσκευές οι οποίες επιτρέπουν καλύτερες επιδόσεις στην εκτέλεση προγραμμάτων, αλλά με τελείως διαφορετική φιλοσοφία στον προγραμματισμό τους. Στο τμήμα 1.9 γίνεται μια σύντομη σύγκριση μεταξύ των δύο συστημάτων.

1.2 Μικροεπεξεργαστής (microprocessor - μP) [1]

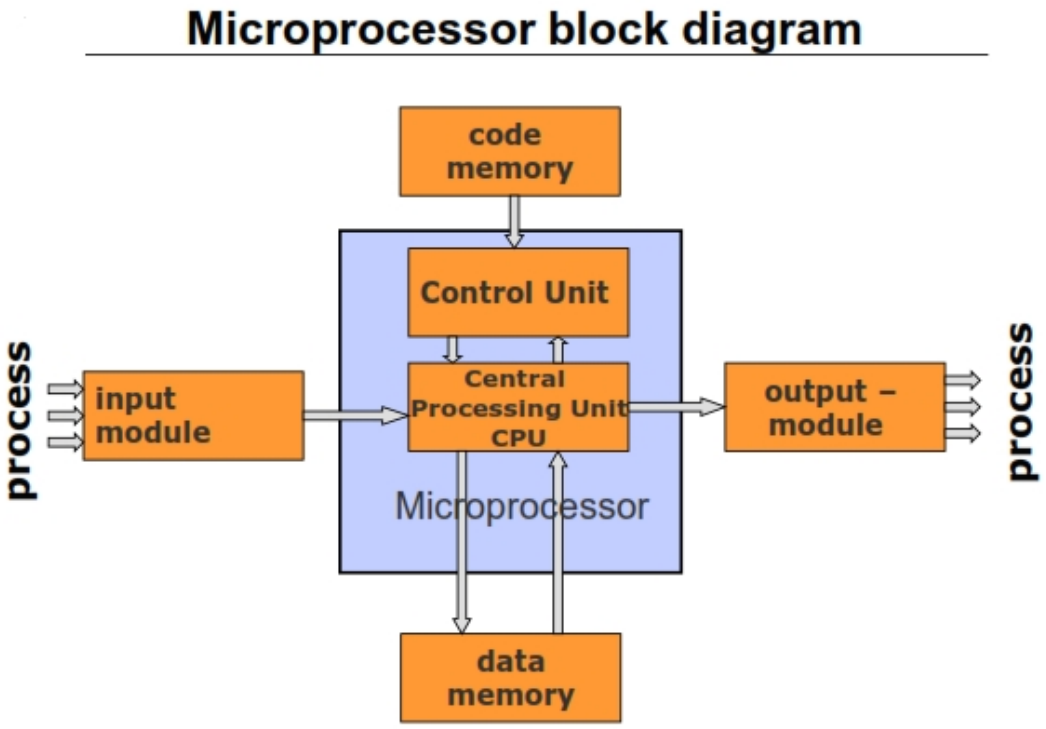
Ένας μικροεπεξεργαστής βασίζεται στην εξής απλή αλληλουχία εργασιών:

1. Διάβασμα εντολής από την μνήμη εντολών
2. Αποκωδικοποίηση εντολής
3. Διάβασμα τελεστών από τη μνήμη δεδομένων
4. Εκτέλεση εντολής
5. Αποθήκευση των αποτελεσμάτων

Μετά το τέλος αυτού του κύκλου τα βήματα επαναλαμβάνονται, αλλά για την επόμενη εντολή. Ο μP αποτελεί τη βάση οποιουδήποτε ενσωματωμένου συστήματος, ωστόσο μόνος του δεν μπορεί να εκτελέσει καμία λειτουργία. Απαιτείται τουλάχιστον μνήμη και κάποιες εξωτερικές συσκευές, οι οποίες του επιτρέπουν την επικοινωνία με άλλα συστήματα.

Μία διάκριση των μP γίνεται κατά την αρχιτεκτονική σύμφωνα με την οποία σχεδιάστηκαν. Οι δύο επικρατέστερες αρχιτεκτονικές είναι οι Von Neumann και Harvard. Χαρακτηριστικό της **Von Neumann** είναι ότι αξιοποιεί τον ίδιο χώρο και

διαύλους (buses) για την αποθήκευση και διαχείριση του κώδικα και των δεδομένων. Χαρακτηριστικό παράδειγμα αυτής της αρχιτεκτονικής είναι η οικογένεια x86 Pentium της Intel χρησιμοποιεί αυτή την αρχιτεκτονική. Αντίθετα στην αρχιτεκτονική **Harvard** αυτά τα δύο διαχωρίζονται. Ο μP του σχήματος 1.1 είναι σχεδιασμένος με αρχιτεκτονική Harvard.



Σχήμα 1.1: Τυπικό διάγραμμα μικροεπεξεργαστή αρχιτεκτονικής Harvard [3]

Πυρήνας ενός μP είναι η Κεντρική Μονάδα Επεξεργασίας (Central Processing Unit στο εξής CPU). Αυτή αποτελείται από:

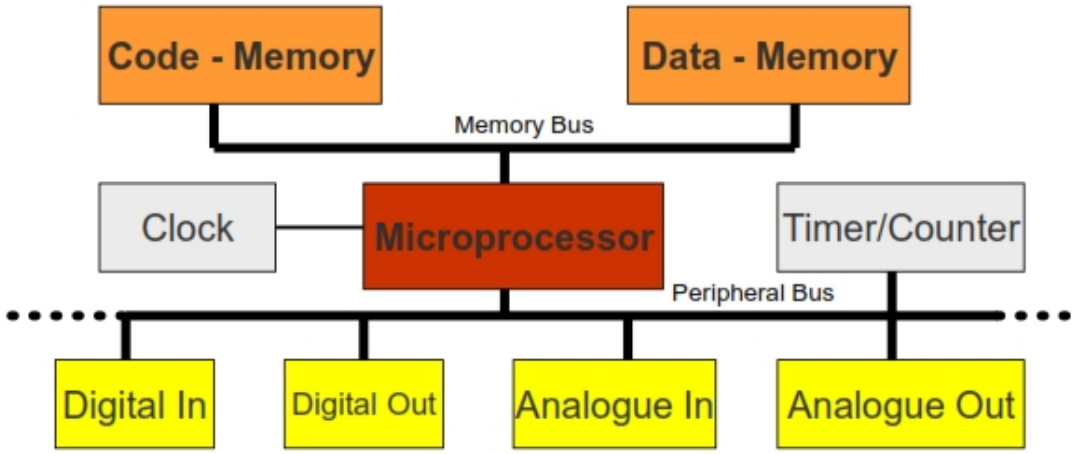
- Καταχωρητές (Registers): κύτταρα μνήμης
- Αριθμητική και Λογική Μονάδα (Arithmetic and Logic Unit στο εξής ALU)
- Καταχωρητής διεύθυνσης εντολών και αποκωδικοποιητής διεύθυνσης
- Μονάδα διευθύνσεων

*Η **μονάδα διευθύνσεων** διαβάζει από τη μνήμη δεδομένων και εντολών και γράφει τα αποτελέσματα στη μνήμη δεδομένων.*

1.3 Μικροϋπολογιστής (microcomputer - μPC) [1]

Ο μικροϋπολογιστής αποτελείται από έναν μP στον οποίον έχουν προστεθεί εξωτερική μνήμη και διάφορες περιφερειακές συσκευές. Οι περιφερειακές συσκευές

είναι υπεύθυνες για την επικοινωνία του συστήματος με τον εξωτερικό κόσμο μέσω αισθητήρων.



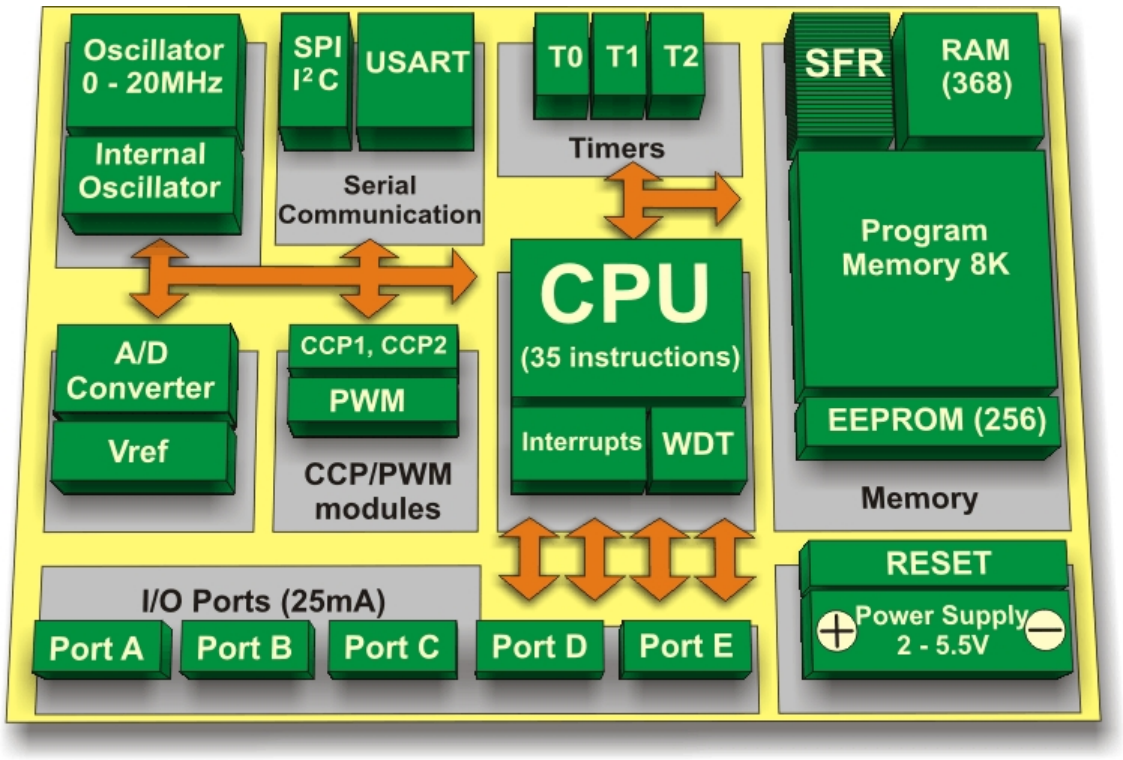
Σχήμα 1.2: Τυπικό διάγραμμα μικροϋπολογιστή [3]

Παραδείγματα τέτοιων συσκευών είναι:

- Ψηφιακή είσοδος/Εξοδος
- Μετατροπέας Αναλογικού σήματος σε Ψηφιακό (Analogue to Digital Converter- ADC)
- Μετατροπέας Ψηφιακού σήματος σε Αναλογικό (Digital to Analogue Converter- DAC)
- Χρονιστές / Απαριθμητές
- Διαμόρφωση Εύρους παλμών (PWM) σε ψηφιακές εξόδους
- Μονάδα σύλληψης ψηφιακών παλμοσειρών (Capture Unit)
- Μονάδες Διεπαφής δικτύου:
    - Serial Communication Interface (SCI) - UART
    - Serial Peripheral Interface (SPI)
    - Inter Integrated Circuit (I^2C)- Bus
    - Controller Area Network (CAN)
    - Local Interconnect Network (LIN)
    - Universal Serial Bus (USB)
    - Local / Wide Area Networks (LAN, WAN)
- Συσκευές Γραφικών

1.4 Μικροελεγκτής (microcontroller-μC) [1]

Ο Μικροελεγκτής εκτελεί τις τυπικές λειτουργίες ενός μPC (όπως αυτές περιγράφηκαν στην προηγούμενη παράγραφο) με τη διαφορά ότι αποτελείται από ένα σύστημα υλοποιημένο σε ένα ενιαίο κομμάτι πυριτίου (System on Chip- SoC). Τα συστήματα που χρησιμοποιούν μικροελεγκτές ονομάζονται ενσωματωμένα συστήματα (embedded systems).



Σχήμα 1.3: Το γενικό διάγραμμα βαθμίδων ενός ενσωματωμένου συστήματος [www.mikroe.com]

Προφανώς, είναι άστοχο και δαπανηρό να συμπεριλάβουμε σε ένα ενσωματωμένο σύστημα, όλες τις διαθέσιμες λειτουργίες και περιφερειακά. Οι μικροελεγκτές μπορούν να έχουν διάφορα μεγέθη μνήμης, να λειτουργούν σε διαφορετικές συχνότητες και να έχουν διαφορετικά περιφερειακά ανάλογα με την εκάστοτε εφαρμογή. Για παράδειγμα ένας μικροελεγκτής που ελέγχει το σύστημα πρόσφυσης ενός αυτοκινήτου, έχει διαφορετικές προδιαγραφές από έναν μικροελεγκτή που ελέγχει τη στάθμη μιας δεξαμενής.

Οι μικροελεγκτές κατατάσσονται σε κατηγορίες ανάλογα με τον αριθμό bit από τον οποίο αποτελείται ο δίαυλος επικοινωνίας μνήμης - επεξεργαστή. Κατά κανόνα περισσότερα bits σημαίνουν πολυπλοκότερο και ακριβότερο μικροελεγκτή.

Στον παρακάτω πίνακα παραθέτουμε τα χαρακτηριστικά δύο εμπορικών οικογενειών μικροελεγκτών και τις αντίστοιχες εφαρμογές τους:

Πίνακας 1.1: Χαρακτηριστικά δύο οικογενειών μC που είναι κατάλληλοι για διαφορετικές εφαρμογές

Όνομα	MSP430	UC3 C-series
Εταιρία	Texas Instruments	Atmel
Μέγεθος διαύλου	16 bit	32 bit
Μέγιστη συχνότητα	25MHz	66MHz
Μνήμα	0,5 έως 512KB ανάλογα με το μοντέλο	64 έως 512KB ανάλογα με το μοντέλο

Λοιπά χαρακτηριστικά	FRAM	• Δυνατότητα PWM με dead band • 16 κανάλια ADC • 4 κανάλια DAC
Εφαρμογές	• Βιο-ιατρική • Έξυπνο δίκτυο • Αισθητήρες	Υψηλής επίδοσης Βιομηχανικές εφαρμογές

1.5 Ψηφιακός Επεξεργαστής Σήματος (Digital Signal Processor- DSP) [3]

Ένας ψηφιακός επεξεργαστής σήματος είναι παρόμοιος σε λειτουργία με τον μP. Ωστόσο, σε αυτόν έχουν προστεθεί κατάλληλα στοιχεία υλικού (hardware) τα οποία επιταχύνουν τις αριθμητικές πράξεις. Στόχος είναι η επεξεργασία των τιμών εισόδου και η παραγωγή αποτελεσμάτων ή και εξόδων σε πραγματικό χρόνο.

Οι επιπλέον αριθμητικές μονάδες μπορεί να είναι:

- Πολλαπλασιαστές
- Σύστημα διαύλων για παράλληλη επεξεργασία δεδομένων
- Ολισθητές (εκτελούν πολλαπλασιασμό και διαίρεση με τους αριθμούς 2^n, όπου n είναι ο αριθμός τον bit που ολίσθησαν)

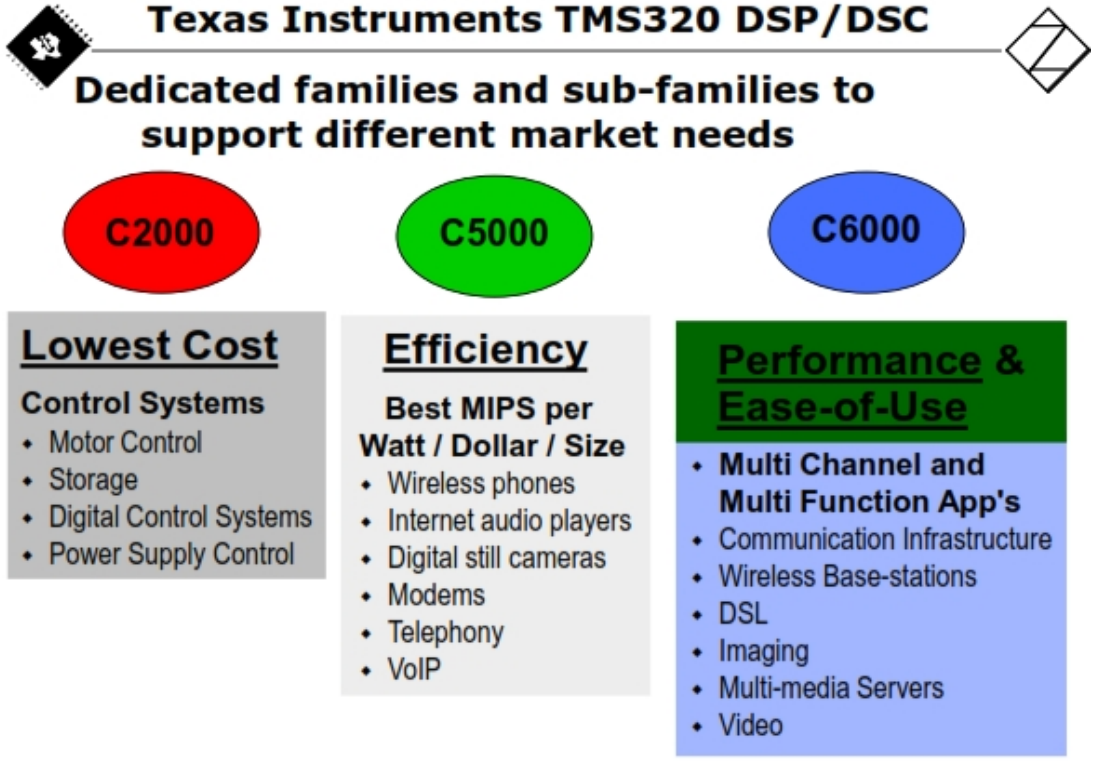
Οι επιπλέον αριθμητικές μονάδες του F28335 αναφέρονται στο κεφάλαιο 2.

1.6 Ψηφιακός Ελεγκτής Σήματος (Digital Signal Controller- DSC) [3]

Ο ψηφιακός ελεγκτής σήματος αποτελεί έναν νέο τύπο μC, ο οποίος στον πυρήνα του αντί για μικροεπεξεργαστή έχει έναν DSP. Οπότε το DSC είναι ένα ενσωματωμένο σύστημα το οποίο συνδυάζει τα υπολογιστικά οφέλη ενός DSP με τα περιφερειακά ενός μικροελεγκτή.

Σε εφαρμογές πραγματικού χρόνου, στις οποίες απαιτείται ταχύτητα στον υπολογισμό δεδομένων, τα DSC είναι καταλληλότερα σε σχέση με τους πιο αργούς μικροελεγκτές.Είναι αρκετά ενδιαφέρον να σημειωθεί ότι πολλά DSC εμπορικά αναφέρονται ως μικροελεγκτές. Αυτό συμβαίνει για εμπορικούς καθαρά λόγους, καθώς ο μέσος μηχανικός δεν είναι εξοικειωμένος με τον όρο DSC.

Στο σχήμα 1.4 περιγράφονται τα χαρακτηριστικά των διάφορων οικογενειών DSC της εταιρίας Texas Instruments. Το DSC F28335 ανήκει στην οικογένεια C2000.



Σχήμα 1.4: Οι οικογένειες DSC της Texas Instruments και οι εφαρμογές τους [3]

1.7 Εφαρμογές των μικροελεγκτών και των DSC

Τα ενσωματωμένα συστήματα, τα οποία αποτελούνται είτε από μικροελεγκτές είτε από DSC, μπορούν να ικανοποιήσουν τις προδιαγραφές πολλών εφαρμογών. Ιδιαίτερη ανάπτυξη υπάρχει στον κλάδο των ηλεκτρονικών ισχύος, καθώς οι σύγχρονες διατάξεις οδηγούνται από ψηφιακά σήματα υψηλών διακοπτικών συχνοτήτων. Η διπλωματική αυτή εξετάζει σε βάθος τη χρήση των DSC στον έλεγχο τέτοιων διατάξεων. Ωστόσο, τα DSC και οι μικροελεγκτές έχουν πολλαπλές εφαρμογές σε όλο το εύρος δραστηριότητας ενός Ηλεκτρολόγου Μηχανικού. Στις επόμενες σελίδες αναφέρονται επιγραμματικά τέτοιες εφαρμογές.

- **Έξυπνο δίκτυο [4]**

Το έξυπνο δίκτυο (smart grid) αναπτύσσεται τα τελευταία χρόνια λόγω της αυξανόμενης διείσδυσης των ΑΠΕ στο δίκτυο ηλεκτρικής ενέργειας. Βασική ιδέα του έξυπνου δικτύου είναι η **αμφίδρομη ροή ενέργειας και πληροφορίας**. Αυτό θα έχει σαν αποτέλεσμα το σύστημα ηλεκτρικής ενέργειας να είναι παντού **παρατηρήσιμο** (ικανό να υπολογιστεί και να απεικονιστεί), **ελέγξιμο** (διαχειρίσιμο και ικανό να βελτιστοποιηθεί), **αυτοματοποιημένο** (ικανό να προσαρμοστεί και να αυτό-θεραπευτεί), **πλήρως διασυνδεδεμένο** (πλήρως διαλειτουργικό με τα υπάρχοντα συστήματα και με την ικανότητα να ενσωματώσει ένα διαφορετικό σύνολο πηγών ενέργειας).

Πλεονεκτήματα του έξυπνου δικτύου έναντι του παραδοσιακού είναι:

- Η αυξημένη αξιοπιστία του δικτύου.
- Η βελτιστοποίηση στην αξιοποίηση εγκαταστάσεων παραγωγής ηλεκτρικής ενέργειας.
- Η διευκόλυνση στην ανάπτυξη των ΑΠΕ.
- Η καλύτερη αξιοποίηση των κατανεμημένων πηγών ενέργειας.

Ο τρόπος με τον οποίο μπορεί να υλοποιηθεί ένα τέτοιο δίκτυο είναι η ύπαρξη **πληθώρας αισθητήρων** σε επιλεγμένα σημεία των γραμμών, οι οποίοι θα δίνουν σαν ανάδραση **τάση** (γωνία και μέτρο) **και ρεύμα** (γωνία και μέτρο). Αυτοί οι αισθητήρες θα μπορούσαν να αποτελούν ένα ενσωματωμένο σύστημα με 2 εισόδους ADC και με κατάλληλη διεπαφή δικτύου ώστε η ανάδραση να επιστρέφει στον απομακρυσμένο δέκτη για επεξεργασία.

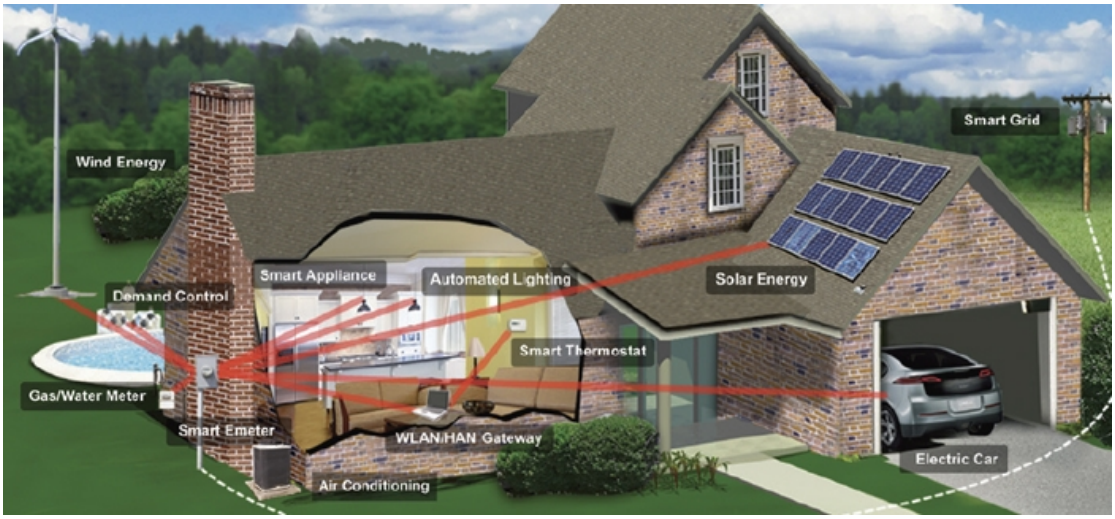
Τα πρωτόκολλα ασύρματης επικοινωνίας που χρησιμοποιούνται στην τεχνολογία smart grid είναι: ασύρματα τοπικά δίκτυα (WLAN), WiMAX, Κυψελωτές επικοινωνίες (Cellular Network Communication), δορυφορικές επικοινωνίες, κα. Με κατάλληλα περιφερειακά μπορεί να σχεδιαστεί ένα ενσωματωμένο σύστημα που να πληροί τις προϋποθέσεις για την επικοινωνία με το απομακρυσμένο κέντρο.

- **Έξυπνο σπίτι [4]**

Αποτελεί εφαρμογή παρόμοια με αυτή του Έξυπνου Δικτύου αλλά σε μικρότερη κλίμακα. Στο έξυπνο σπίτι ο φωτισμός, η θέρμανση και η ροή ηλεκτρικής ενέργειας (στην περίπτωση που στο σπίτι είναι εγκατεστημένη κάποια ΑΠΕ) μπορούν να ελέγχονται μέσω διακοπτών οι οποίοι είναι συμβατοί με ένα πρωτόκολλο ασύρματης επικοινωνίας.

Το πρωτόκολλο που χρησιμοποιείται σε τέτοιες εφαρμογές ονομάζεται ZigBee. Χαρακτηριστικά του είναι η χαμηλή κατανάλωση ισχύος, το μικρό κόστος εφαρμογής και η μικρή πολυπλοκότητα. Η εμβέλειά του ποικίλει μεταξύ 10 και 100 μέτρων. Χρησιμοποιώντας, λοιπόν, έναν μικροελεγκτή που μπορεί να επικοινωνήσει με τους διακόπτες ZigBee, έχουμε στη διάθεσή μας ένα σύστημα πλήρως ελέγξιμο.

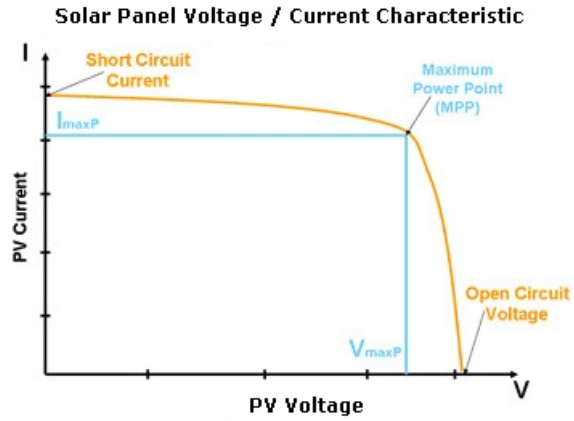
Είναι σημαντικό να σημειωθεί ότι πολλοί μικροελεγκτές έχουν τη δυνατότητα αξιοποίησης και δικτύου WiFi. Οπότε το επόμενο βήμα στην υλοποίηση του έξυπνου σπιτιού είναι ο έλεγχος της κατάστασης του οικιακού μικροδυκτίου από μεγάλες αποστάσεις μέσω του διαδικτύου.



Σχήμα 1.5: Έξυπνο σπίτι [www.ti.com]

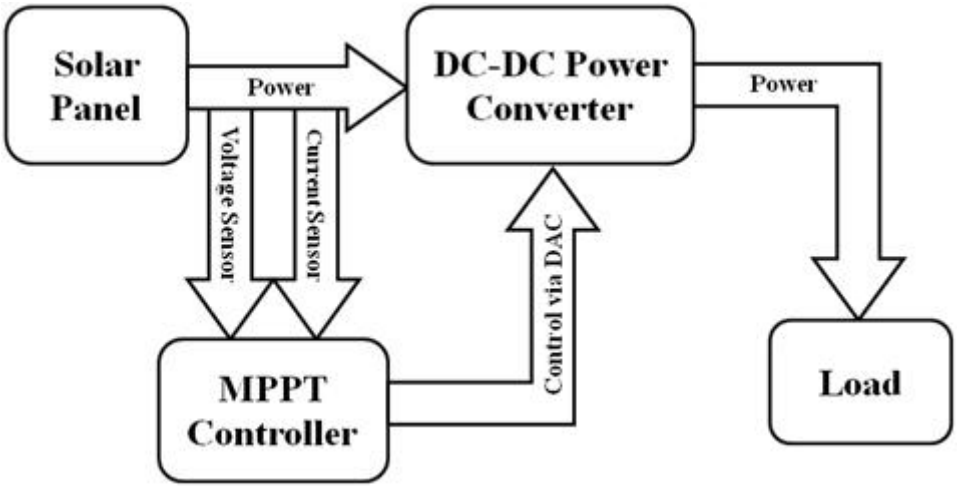
- **Φωτοβολταϊκά με Ιχνηλάτη Σημείου Μέγιστης Ισχύος (MPPT) [5]**

Στο σχήμα 1.6 παρουσιάζεται η χαρακτηριστική I-V ενός τυπικού φωτοβολταϊκού κυττάρου. Το σημείο λειτουργίας στο οποίο παράγεται η μέγιστη ηλεκτρικής ισχύς σημειώνεται πάνω στην χαρακτηριστική και βρίσκεται στο «γόνατο» της καμπύλης. Το σημείο αυτό ονομάζεται Σημείο Μέγιστης Ισχύος (Maximum Power Point- MPP) και η γνώση του ανά πάσα στιγμή είναι πολύ σημαντική για την απόδοση των φωτοβολταϊκών συστημάτων.



Σχήμα 1.6: Ποιοτική Χαρακτηριστική I-V ενός τυπικού Φωτοβολταϊκού κυττάρου. [www.mpoweruk.com]

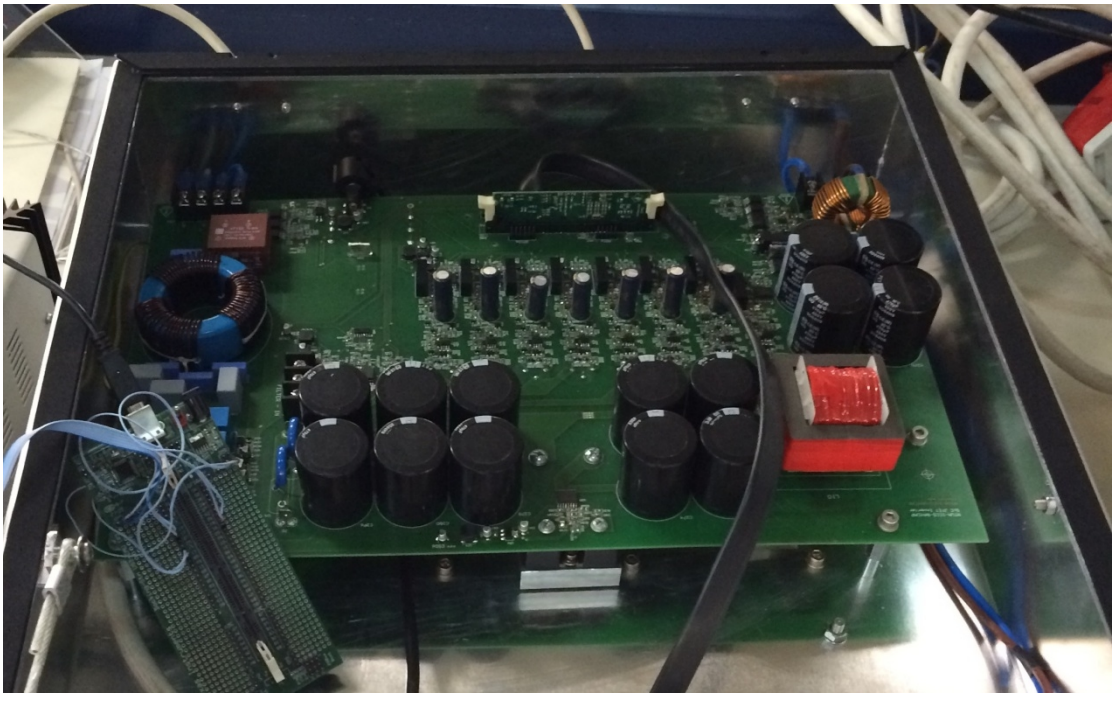
Για την επίτευξη της μέγιστης ισχύς εξόδου από τη συστοιχία φ/β χρησιμοποιούνται διατάξεις ψηφιακού ελέγχου που περιγράφονται από το παρακάτω δομικό διάγραμμα:



Σχήμα 1.7: Δομικό διάγραμμα ελέγχου φ/β πλαισίων με MPPT [www.mpoweruk.com]

Οι αλγόριθμοι MPPT ελέγχουν το κύκλο λειτουργίας του μετατροπέα και μεταβάλλουν με αυτόν τον τρόπο την αντίσταση που «βλέπει» η φ/β συστοιχία. Ο μικροελεγκτής αναπροσαρμόζει τον κύκλο λειτουργίας ανάλογα με τις μεταβολές στις συνθήκες περιβάλλοντος αλλά και του φορτίου.

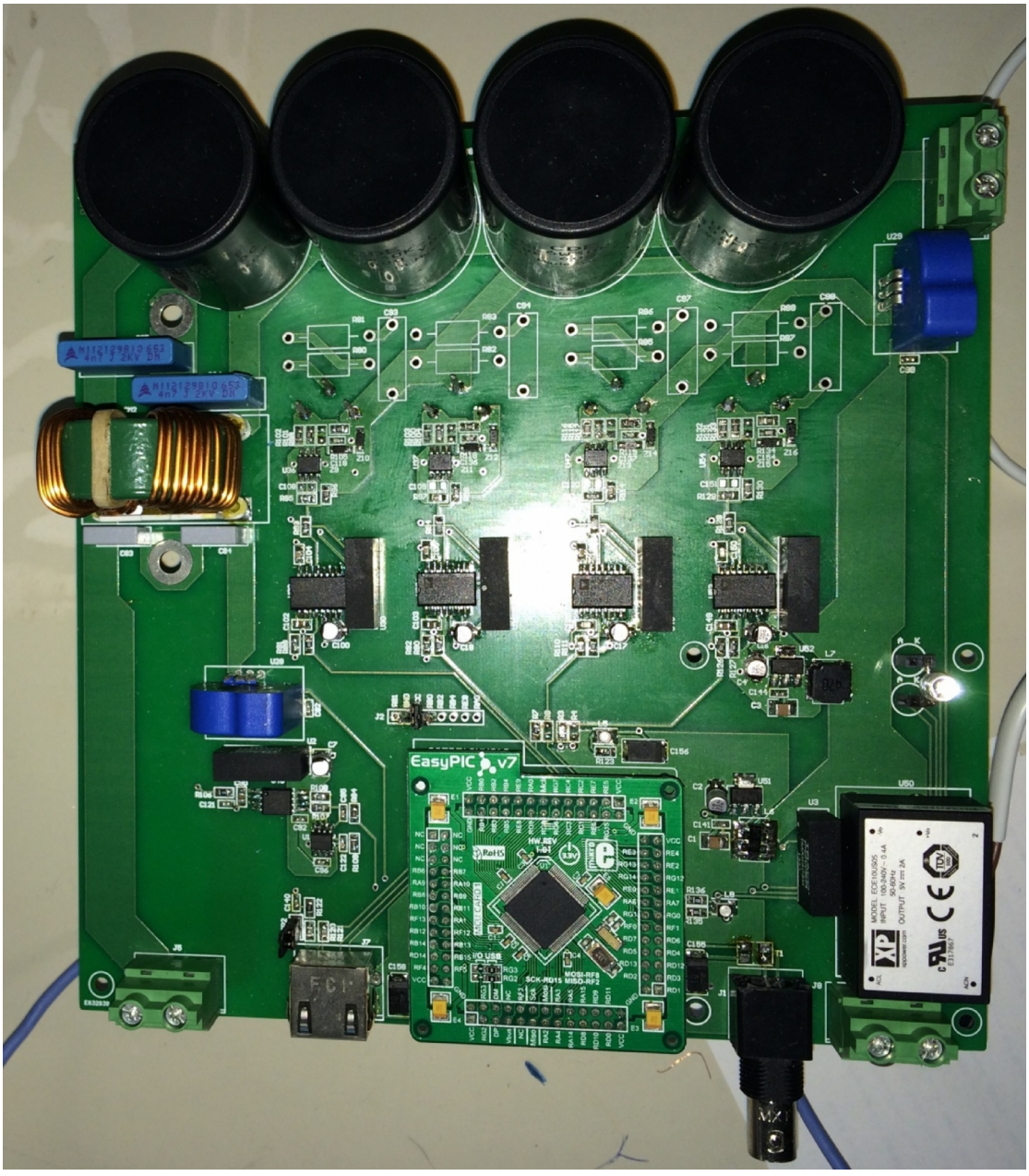
Στο σχήμα 1.8 υπάρχει ένα ενσωματωμένο σύστημα αντιστροφέα, ο οποίος χρησιμοποιείται για διατάξεις φωτοβολταϊκών στοιχείων.



Σχήμα 1.8: Ενσωματωμένο σύστημα αντιστροφέα φωτοβολταϊκών, με χρήση του DSC TMS320F28335

Οι παραπάνω εφαρμογές είναι μόνο ένα δείγμα των δυνατοτήτων των μικροελεγκτών και των DSC στον τομέα της ηλεκτρολογίας. Άλλες εξ' ίσου σημαντικές εφαρμογές είναι:

- Έλεγχος ανεμογεννητριών
- Υβριδικά οχήματα
- Φωτισμός δρόμων, κτιρίων
- Αυτοματοποιημένη παραγωγή, γραμμές παραγωγής
- Φόρτιση συσσωρευτών (Σχήμα 1.9)



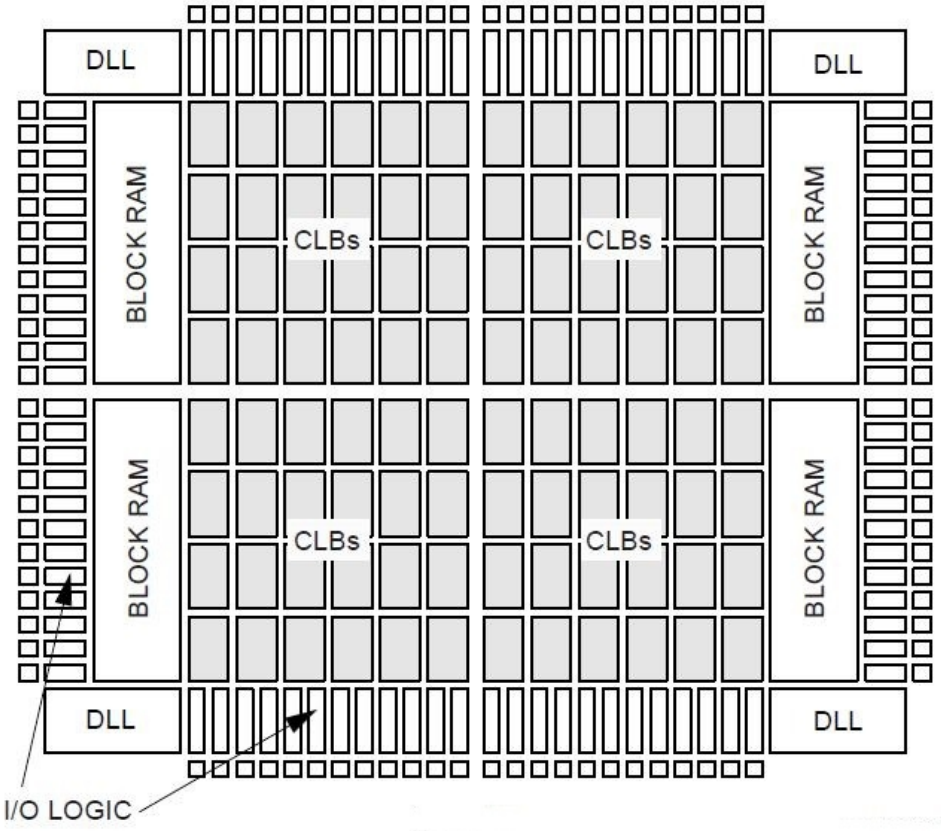
Σχήμα 1.9: Ενσωματωμένο σύστημα φόρτισης συσσωρευτών

1.8 Συσκευές προγραμματιζόμενης Λογικής (FPGAs) [1]

Τα τελευταία χρόνια οι εφαρμογές των ηλεκτρονικών ισχύος απαιτούν πολύπλοκους αριθμητικούς υπολογισμούς για την οδήγηση διατάξεων. Υπάρχει, δηλαδή, ανάγκη για συστήματα τα οποία έχουν καλύτερες επίδοσης και υψηλότερη υπολογιστική ισχύ. Για αυτό το λόγο σε πολλές εφαρμογές, πλέον, προτιμούνται τα FPGA έναντι των DSC.

Τα FPGA (Field Programmable Gate Arrays) είναι ψηφιακά ολοκληρωμένα κυκλώματα που περιέχουν προγραμματιζόμενα μπλοκ λογικής τα οποία διασυνδέονται με επίσης προγραμματιζόμενες συνδέσεις. Ο όρος field (πεδίο) καταδεικνύει την δυνατότητα αλλαγής στο χώρο λειτουργίας, σε αντίθεση με άλλα ολοκληρωμένα (όλες οι τεχνολογίες που έχουν αναφερθεί μέχρι τώρα) στα οποία το υλικό δε μεταβάλλεται.

Στο σχήμα 1.10 είναι το δομικό διάγραμμα της οικογένειας FPGA Spartan II. Αν και το μοντέλο είναι σχετικά παλιό (το 2009 εισήχθη στην αγορά η 6^{η} γενιά Spartan) το διάγραμμα είναι κατάλληλο για την κατανόηση λειτουργίας ενός FPGA.



Σχήμα 1.10: Δομικό διάγραμμα FPGA της οικογένειας Spartan II, εταιρία Xilinx [www.xilinx.com]

Στο εσωτερικό του διαγράμματος υπάρχουν τα **Μπλοκ λογικής (Configurable Logic Blocks)**. Κάθε CLB αποτελείται από 4 slices και κάθε slice από 2 κύτταρα μνήμης τα οποία είναι και η στοιχειώδης μονάδα προγραμματισμού. Ανάλογα με το σχεδιασμό του προγραμματιστή το CLB μπορεί να είναι ένας ολισθητής bit, μία λογική πράξη ή μία μνήμη. Διασυνδέσεις μεταξύ διαφορετικών CLB επιτρέπουν την υλοποίηση πιο σύνθετων λογικών διαδικασιών.

Στις τέσσερις άκρες είναι τοποθετημένα τα **Delay-Locked Loops (DLL)** τα οποία διανέμουν το ρολόι του ολοκληρωμένου στην πλακέτα και αντισταθμίζουν όποιες καθυστερήσεις εμφανιστούν. Τέλος, το διάγραμμα ολοκληρώνεται από τα **κύτταρα μνήμης RAM** και από τις θέσεις **εισόδου/εξόδου** οι οποίες διασυνδέουν την εσωτερική λογική των CLB με τα εκάστοτε περιφερειακά. Στον χώρο των FPGA κυρίαρχες εταιρίες είναι η Xilinx και η Altera.

1.9 Σύγκριση DSC με FPGA

- **Ταχύτητα επεξεργασίας**

Τα DSC σε προγράμματα τα οποία απαιτούν υψηλό ρυθμό στην επεξεργασία δεδομένων, μπορεί να εμφανίσουν προβλήματα. Αυτό οφείλεται στο γεγονός ότι οι πόροι του συστήματος (αριθμητικές μονάδες, δίαυλοι) είναι πεπερασμένοι. Αντίθετα σε ένα FPGA μπορούν να σχεδιαστούν διατάξεις με πολλαπλές μονάδες υπολογισμού και μεταφοράς δεδομένων, οι οποίες θα λειτουργούν παράλληλα[6]. Ένα επιπρόσθετο πλεονέκτημα των FPGA είναι ότι οι μονάδες υλικού μπορούν να προσαρμοστούν ανάλογα με τον αριθμό bit των δεδομένων. Οπότε, αν μια εφαρμογή εκτελεί πολλαπλασιασμούς μεταξύ αριθμών με 8 bit, αυτόματα οι θα είναι κατάλληλα διαστασιολογημένοι επιτρέποντας στο σχεδιαστή να κάνει οικονομία στους πόρους. Ένας άλλος λόγος είναι ότι τα DSC έχουν δεδομένο τρόπο με τον οποίον εναλλάσσονται οι εντολές. Για κάθε εντολή εκτελούνται τα πέντε στάδια που αναφέρονται στο τμήμα 1.2 εισάγοντας καθυστέρηση μεταξύ εισόδου και αποτελέσματος. Αντίθετα στα FPGA σε κάθε κύκλο ρολογιού μπορούν να εκτελούνται πολλαπλές πράξεις παράλληλα.

- **Εντολές αποφάσεων/κλάδων**

Εντολές της C όπως η:

if (condition **)** action_if_true **else** action_if_false

αφορούν προγραμματιστικές τεχνικές, οι οποίες χωρίζουν το πρόγραμμα σε κλάδους. Η αλληλουχία των εντολών που ακολουθούν τη συνθήκη εξαρτάται άμεσα από το αν αυτή επαληθεύεται ή όχι. Σε αυτόν τον τομέα τα DSC υπερέχουν[7]. Τα FPGA δεν παράγουν απαραίτητα χειρότερα αποτελέσματα, ωστόσο η υλοποίηση τέτοιου τύπου προγράμματος είναι αρκετά δυσκολότερη.

- **Διαδικασία ανάπτυξης εφαρμογής**

Τα FPGA έχουν τελείως διαφορετική φιλοσοφία προγραμματισμού από τα DSC. Οι γλώσσες προγραμματισμού που χρησιμοποιούν (VHDL, Verilog) χαρακτηρίζονται ως «γλώσσες περιγραφής υλικού» και έχουν μικρή συνάφεια με τις γλώσσες ακολουθιακού προγραμματισμού (πχ. C). Επίσης ένα DSC κατά κανόνα έχει έτοιμες μονάδες υλικού, οι οποίες διευκολύνουν το έργο των μηχανικών λογισμικού. Για παράδειγμα ο DSC της Atmel στον πίνακα 1.1 υποστηρίζει dead band στα κανάλια PWM, ενώ σε ένα FPGA θα έπρεπε να σχεδιάσουμε το υλικό κατά κατάλληλο τρόπο ώστε να επιτελεί αυτή τη λειτουργία.

Συμπερασματικά, η επιλογή μεταξύ ενός FPGA ή ενός DSC εξαρτάται από την εκάστοτε εφαρμογή. Εάν απαιτείται υψηλή ταχύτητα επεξεργασίας, τότε το FPGA είναι η ενδεδειγμένη λύση. Εάν η εφαρμογή περιλαμβάνει πληθώρα διακλαδώσεων και αποφάσεων ή εάν δεν υπάρχει απαίτηση σε ταχύτητα και όγκο επεξεργασίας τότε το DSC είναι η καταλληλότερη επιλογή.

Στα πλαίσια αυτής της διπλωματικής επιλέχθηκε η λύση του DSC για τους λόγους που αναφέρονται παραπάνω. Οι εφαρμογές που σχεδιάστηκαν δεν έχουν μεγάλες απαιτήσεις σε υπολογισμούς πραγματικού χρόνου, ενώ και από άποψη εκμάθησης, η C είναι αρκετά φιλικότερη προς το χρήστη σε σχέση με την Verilog ή την VHDL. Το DSC που επιλέχθηκε είναι το TMS230F28335 της Texas Instruments του οποίου τα χαρακτηριστικά και η αρχιτεκτονική παρουσιάζονται στο επόμενο κεφάλαιο.

1.10 Βιβλιογραφία

[1] Πεκμετζή Κ. «Συστήματα Μικροϋπολογιστών, Τόμος Ι: Μικροεπεξεργαστές Intel 80x86, Pentium και ARM», 2009.

[2] Weste N.H.E. and Harris D.M. , «CMOS VLSI», 2011.

[3] «F2833x tutorial – Module 1» available at Texas Instruments(http://www.ti.com.)

[4] Ζώτου Θ.Ε. «Σύγχρονες Τεχνολογίες Πρόσβασης και Διαδικτύου σε Έξυπνα Δίκτυα (Smart Grids)» Διπλωματική εργασία, Οκτώβριος 2012

[5] Μαρμαρινός Κ.Α. «Τεχνικές Ιχνηλάτησης Σημείου Μέγιστης Ισχύος Φ/Β Συστοιχίας και Προσομοίωση Με Το MATLAB/SIMULINK» Φεβρουάριος 2011.

[6] «FPGAs vs. DSPs: A look at the unanswered questions» available at Berkeley Design Technology, Inc (www.BDTI.com.)

[7] Bamdad Afra and Amit Kapadiya, «DSP or FPGA? How to choose the right device» available at Electrical Engineering Times (http://www.eetimes.com), 2008.

[8] Reg Zatrepalek, «DSP versus FPGA» available at Electronic Weekly (http://www.electronicsweekly.com), 2012.

ΚΕΦΑΛΑΙΟ 2

ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΚΑΙ ΠΕΡΙΦΕΡΕΙΑΚΕΣ ΣΥΣΚΕΥΕΣ ΤΟΥ DSC TMS320F28335

2.1 Εισαγωγή

Πρώτο βήμα για την υλοποίηση μιας εφαρμογής ηλεκτρονικών ισχύος είναι η επιλογή του κατάλληλου μικροεπεξεργαστή. Οι δυνατότητές του δεν πρέπει να είναι ούτε λιγότερες από ότι θέλουμε, αλλά ούτε και κατά πολύ μεγαλύτερες, γιατί αυξάνεται ανούσια το κόστος του τελικού προϊόντος. Για αυτό το λόγο είναι σημαντικό να γνωρίζουμε την **αρχιτεκτονική** του μικροεπεξεργαστή, αλλά και τα **περιφερειακά** που τον συνοδεύουν.

Για τη συγκεκριμένη εργασία επιλέχθηκε ψηφιακός επεξεργαστής σήματος (DSP) **TMS320F28335**. Ο επεξεργαστής αυτός μαζί με τα περιφερειακά της πλακέτας του, αποτελούν τον ψηφιακό **ελεγκτή** σήματος (DSC). Ωστόσο, χάριν απλότητας, όταν αναφερόμαστε σε αυτόν θα χρησιμοποιούμε τον όρο DSP ή F28335 (ο κωδικός TMS320 αναφέρεται σε ολόκληρη οικογένεια DSP) . Ο όρος DSC θα χρησιμοποιείται όταν αναφερόμαστε σε ολόκληρο το σύστημα (επεξεργαστής μαζί με τα περιφερειακά)

Τα περιφερειακά και η αρχιτεκτονική του DSP είναι τα στοιχεία που καθορίζουν τις δυνατότητές του. Στην αρχή του κεφαλαίου (τμήμα 2.2) παραθέτουμε σε συγκεντρωτικό πίνακα (Πίνακας 2.1) τα **χαρακτηριστικά του F28335**.

Στη συνέχεια (τμήμα 2.3) παρουσιάζεται και αναλύεται το **δομικό διάγραμμα του DSP**. Η οργάνωση της κεντρικής μονάδας επεξεργασίας (τμήμα 2.3.1) καθώς και της μνήμης (τμήματα 2.3.2, 2.3.3) βοηθούν στην κατανόηση των διαδικασιών που λαμβάνουν χώρα κατά την εκτέλεση των εντολών ενός προγράμματος. Αυτό το κομμάτι είναι σημαντικό για εφαρμογές πραγματικού χρόνου που απαιτούν ακαριαίες αποκρίσεις από το σύστημα.

Τα επόμενα τμήματα αφορούν τις **περιφερειακές συσκευές** τις πλακέτας. Αυτές συμπεριλαμβάνουν περιφερειακά, όπως τη μονάδα διαμόρφωσης εύρους παλμών (PWM unit). Τα περιφερειακά είναι τα σημαντικότερα στοιχεία ενός DSP, καθώς κάθε εφαρμογή αξιοποιεί ένα ή περισσότερα από αυτά.

Στις περισσότερες εφαρμογές ο DSP λαμβάνει δεδομένα από εξωτερικές συσκευές, τα επεξεργάζεται και παράγει κατάλληλες εξόδους. Αυτό επιτυγχάνεται με **περιφερειακά** που επιτρέπουν στον F28335 να **επικοινωνήσει με εξωτερικές** συσκευές (τμήμα 2.3.5). Για αυτό το σκοπό στη διπλωματική χρησιμοποιείται η μονάδα ψηφιακής εισόδου/εξόδου (GPIO), η οποία παρουσιάζεται λεπτομερώς στο κεφάλαιο 4.

2.2 Χαρακτηριστικά του TMS320F28335 [1]

Τα χαρακτηριστικά του TMS320F28335 καταγράφονται στον παρακάτω πίνακα:

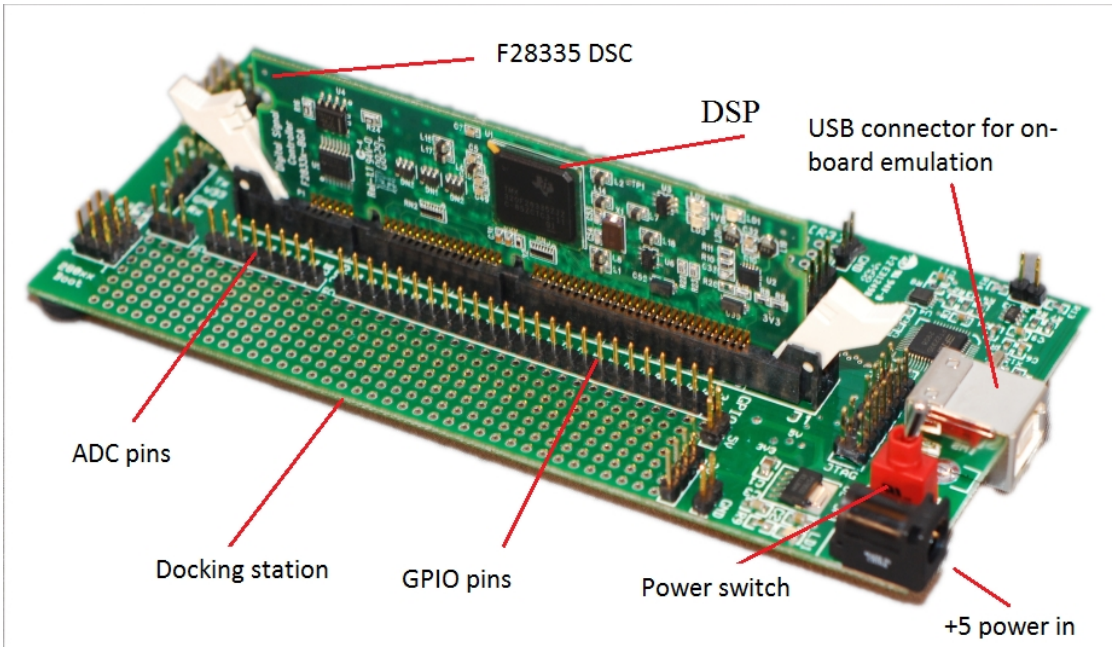
Πίνακας 2.1: Χαρακτηριστικά του TMS320F28335

Χαρακτηριστικά του TMS320F28335		
Μέγιστη συχνότητα	150 MHz	
Μνήμη RAM	68 KB	Μνήμη προγράμματος
Μνήμη Flash	512 KB	Μνήμη δεδομένων
Peak mmacs	1.000.000	Multiply and accumulate cycles per second(mmacs)
DMA	6 κανάλια	Direct memory access (DMA)
PWM	18 κανάλια (12 κύρια και 6 βοηθητικά)	Pulse width modulation (PWM)
CAP	6 κανάλια	Capture module= μονάδα σύλληψης ψηφιακών κυματομορφών
ADC	16 κανάλια 12 bit ανάλυση	Analogue to digital converter (ADC)
McBSP	2 κανάλια	Multi channeled Buffered Serial Port (McBSP)
SCI	3 κανάλια	Serial Communication Interface (SCI)
I2C	1 κανάλι	Inter-integrated circuit (I2C)
SPI	1 κανάλι	Serial Peripheral Interface (SPI)
CAN	2 κανάλια	Controller Area Network (CAN)
GPIO	88	General Purpose Input/Output (GPIO)
Χρονιστές	3 των 32 bit Watchdog	Ο Watchdog είναι ένας χρονιστής υπεύθυνος για την ομαλή λειτουργία του DSP
Θερμοκρασία λειτουργίας	-40 έως 85	

Το κάθε ένα από τα παραπάνω στοιχεία θα αναλυθούν με περισσότερη λεπτομέρεια στη συνέχεια αυτού του κεφαλαίου.

Στο σχήμα 2.1 υπάρχει η φωτογραφία του F28335 με την αναπτυξιακή πλακέτα που χρησιμοποιείται στο εργαστήριο. Να σημειωθεί ότι οι GPIO ακίδες είναι πολλαπλής χρήσης. Όπως θα αναφερθεί και στο κεφάλαιο για το GPIO, κάθε ακίδα έχει έως και 4 λειτουργίες ανάλογα με το πώς θα προγραμματιστεί (πχ. μια ακίδα μπορεί να χρησιμοποιείται σαν PWM έξοδος ή GPIO). Αυτό σημαίνει ότι τα περισσότερα περιφερειακά επικοινωνούν με εξωτερικές συσκευές μέσω των ακίδων GPIO, όταν αυτές έχουν προγραμματιστεί κατάλληλα. Για αυτό το λόγο δεν

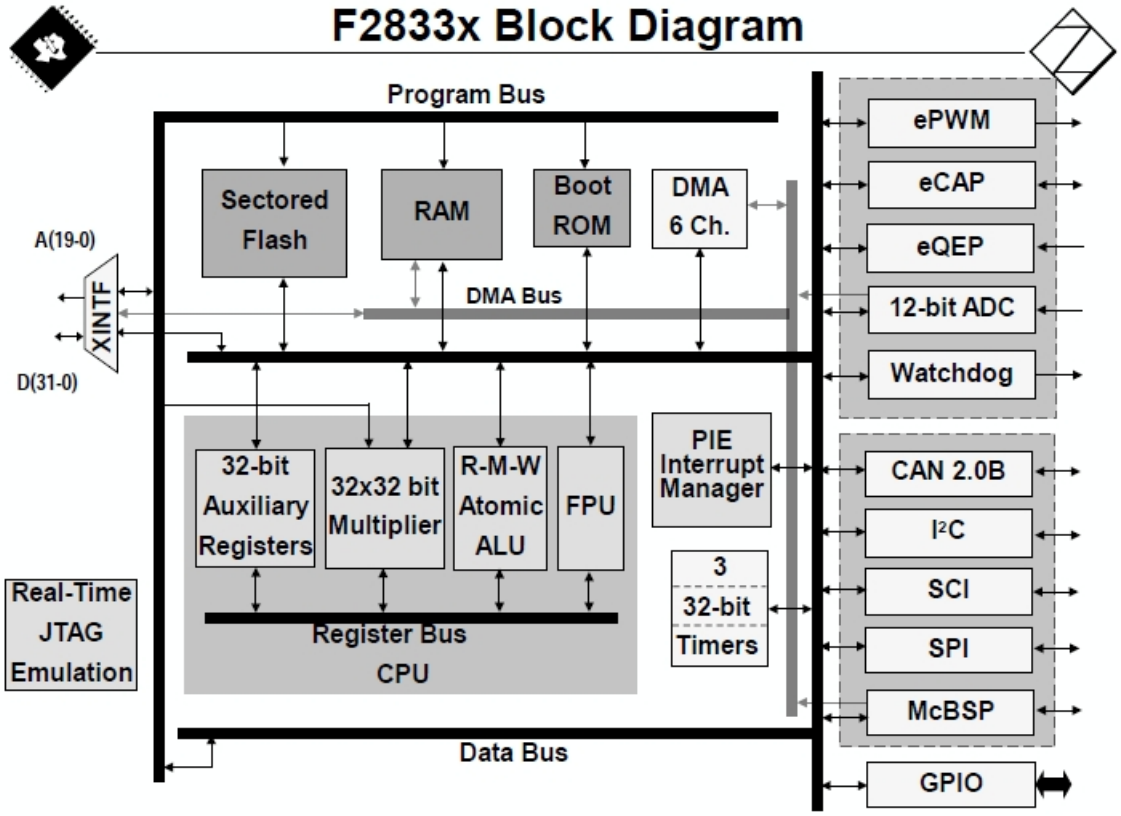
εμφανίζονται στην πλακέτα ακίδες άλλων περιφερειακών εκτός του Analogue to Digital module.



Σχήμα 2.1: F28335 μαζί με την αναπτυξιακή πλακέτα του εργαστηρίου

2.3 Δομικό διάγραμμα του F28335 και ανάλυση των περιφερειακών συσκευών [1]

Στο σχήμα 2.2 υπάρχει το δομικό διάγραμμα του F28335. Σε αυτό το κεφάλαιο θα αναλυθούν τα περισσότερα μέρη του διαγράμματος αυτού, κυρίως εκείνα που ενδιαφέρουν από τη σκοπιά του Ηλεκτρολόγου Μηχανικού. Στα πλαίσια της διπλωματικής δεν κρίνεται σκόπιμο να εισέλθουμε σε μεγάλες λεπτομέρειες αρχιτεκτονικής.

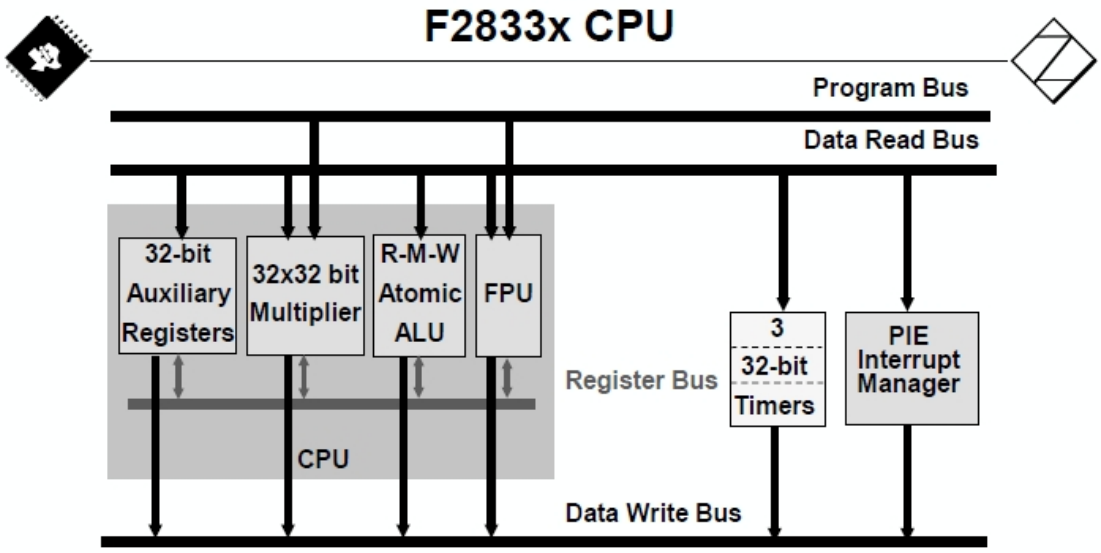


Σχήμα 2.2: Δομικό διάγραμμα των DSC της οικογένειας F2833x [1]

Όπως όλα τα DSC, ο F28335 έχει στον πυρήνα του την Κεντρική Μονάδα επεξεργασίας (CPU). Χάρη στο σχεδιασμένο κατά Harvard σύστημα διαύλων (βλ. Κεφάλαιο 1) η CPU επικοινωνεί με στοιχεία μνήμης (program bus) και με περιφερειακές συσκευές (data bus). Στο διάγραμμα φαίνεται και ένας τρίτος δίαυλος, ο δίαυλος Άμεσης Πρόσβασης Μνήμης (Direct Memory Access- DMA). Ο δίαυλος DMA επιτρέπει στη μονάδα DMA να επικοινωνεί αυτόνομα (χωρίς τη μεσολάβηση της CPU) με κομμάτια της μνήμης.

2.3.1 Κεντρική μονάδα επεξεργασίας (CPU) [1]

Η CPU (Σχήμα 2.3) του F28335 αποτελείται από 4 αριθμητικές μονάδες οι οποίες μπορούν να λειτουργούν και παράλληλα, εξοικονομώντας χρόνο στην εκτέλεση του προγράμματος. Η επικοινωνία μεταξύ των διαφορετικών μονάδων της CPU επιτυγχάνεται με το δίαυλο καταχωρητών, ο οποίος είναι και μέρος της αρχιτεκτονικής της.



Σχήμα 2.3: Κεντρική μονάδα επεξεργασίας [2]

Οι διαφορετικές αριθμητικές μονάδες είναι:

- Πολλαπλασιαστής 32 bit σταθερής υποδιαστολής
- Τυπική αριθμητική και λογική μονάδα (ALU)
- Μονάδα για πράξεις κινητής υποδιαστολής (FPU) απλής ακρίβειας
- Βοηθητικοί καταχωρητές 32 bit

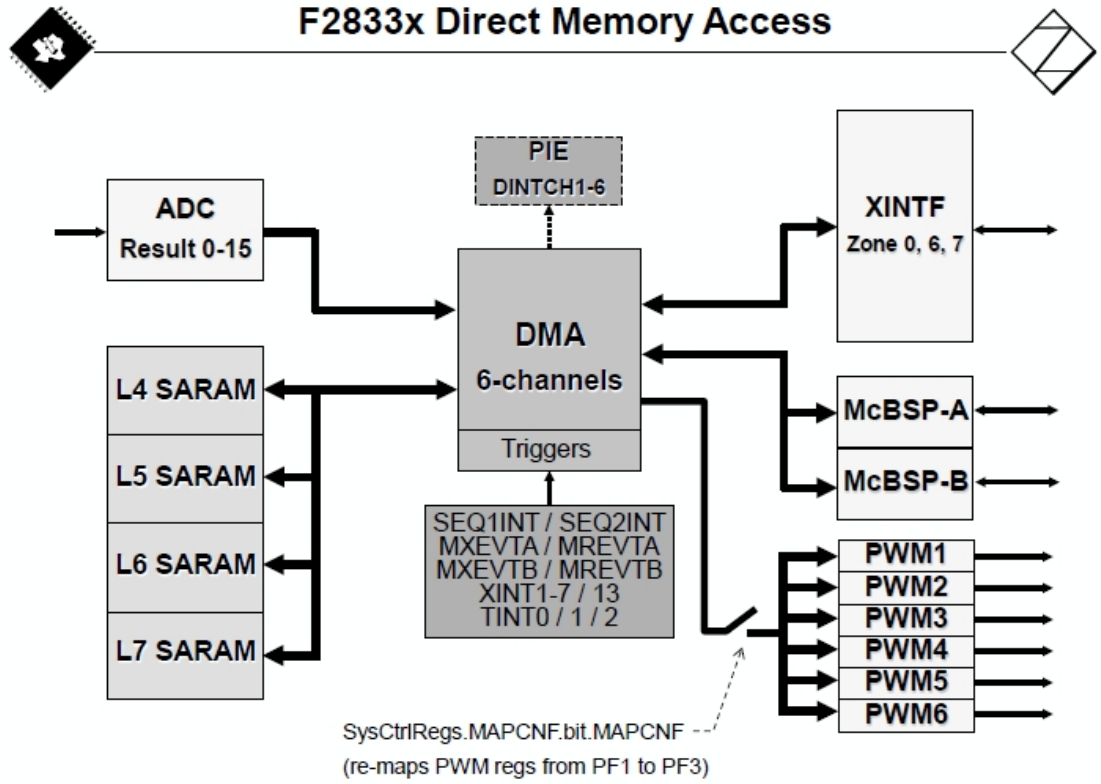
Οι βοηθητικοί καταχωρητές έχουν τη δική τους ALU (ονομάζεται ARAU) και μπορούν να εκτελέσουν αριθμητικές πράξεις δεικτών σε παραλληλία με τις άλλες μονάδες.

Εκτός από τις αριθμητικές μονάδες στον επεξεργαστή υπάρχουν και τρεις χρονιστές των 32 bit και ο Peripheral Interrupt Expansion Manager. Οι διακοπές είναι κάτι στο οποίο θα αναφερθούμε σε επόμενο κεφάλαιο, ωστόσο συνοπτικά οι διακοπές δίνουν τη δυνατότητα στον επεξεργαστή να επικοινωνήσει με τα περιφερειακά συστήματα. Η μονάδα PIE Manager συντελεί στην αμεσότερη επικοινωνία των περιφερειακών συσκευών με τον επεξεργαστή μειώνοντας την «Καθυστέρηση Διακοπής» (Interrupt Latency) ή πιο απλά, την απόκριση του συστήματος.

2.3.2 Ελεγκτής Άμεσης Πρόσβασης Μνήμης (DMA) [1]

Η απόδοση ενός συστήματος DSC δεν εξαρτάται μόνο από την συχνότητα του επεξεργαστή του, αλλά από το συνολικό αριθμό πράξεων που μπορεί να εκτελέσει σε δεδομένο χρονικό διάστημα. Ο Ελεγκτής DMA δίνει τη δυνατότητα στο σύστημα να μετακινεί δεδομένα χωρίς την μεσολάβηση της CPU. Η μεταφορά δεδομένων μπορεί να γίνει είτε από μία περιοχή της μνήμης σε μια άλλη είτε από επικοινωνία περιφερειακών (πχ ADC module/PWM) με τη μνήμη (πχ RAM).

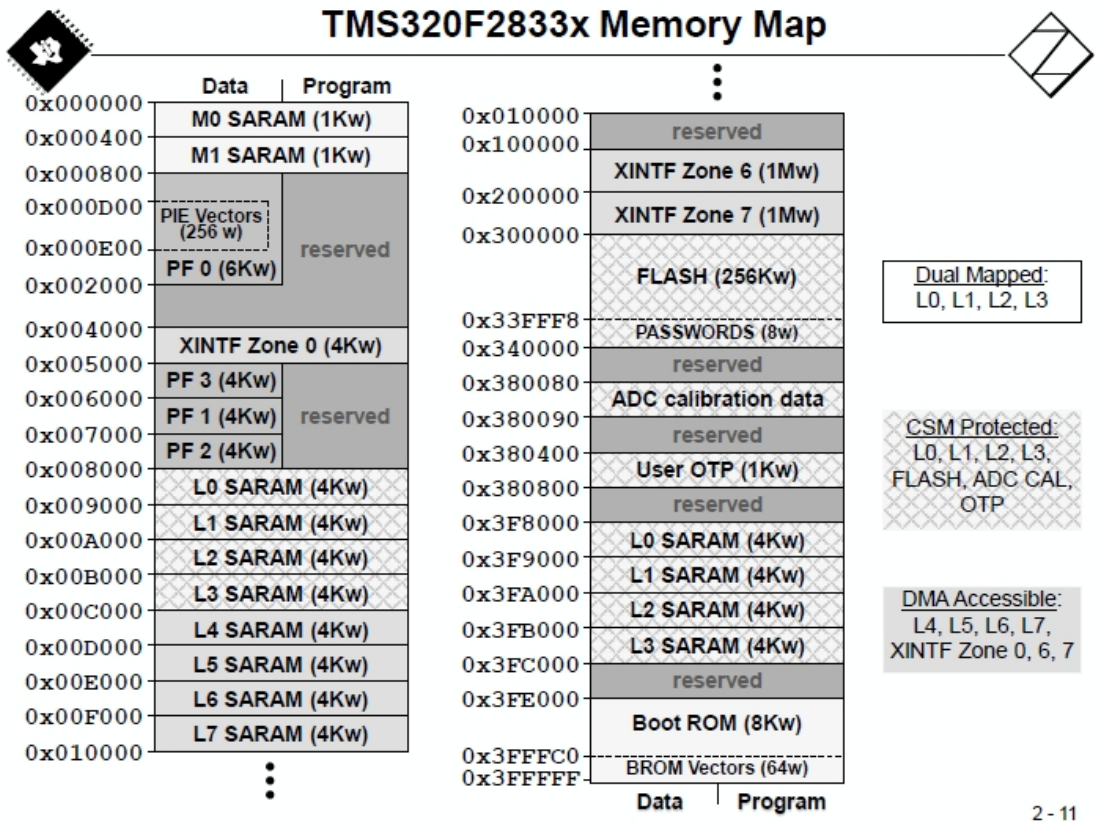
Στο παρακάτω σχήμα (2.4) παρουσιάζεται εποπτικά η λειτουργία του DMA controller με τα διάφορα κομμάτια της RAM (Lx SARAM) καθώς και με τις περιφερειακές συσκευές.



Σχήμα 2.4: Διάγραμμα λειτουργίας του DMA controller [2]

2.3.3 Χάρτης Μνήμης [1]

Η μνήμη του F28335 χωρίζεται σε **μνήμη προγράμματος** και **μνήμη δεδομένων**. Όπως φαίνεται στο σχήμα 2.5 υπάρχουν αρκετά κομμάτια της μνήμης που χρησιμοποιούνται και για τις δύο λειτουργίες. Αυτό συμπεριλαμβάνει κάποια ανεξάρτητα κομμάτια της Flash, την RAM (single access RAM- SARAM) και την boot ROM. Η boot ROM είναι μνήμη εργοστασιακά προγραμματισμένη με λογισμικό, ρουτίνες εκκίνησης και τριγωνομετρικούς πίνακες. Το εύρος της μνήμης είναι 16bit. Οι τιμές μνήμης είναι σε λέξεις (1 word= 16 bit)



Σχήμα 2.5: Χάρτης μνήμης για τον F28335 [2]

Στον F28335 μπορεί να προστεθεί μνήμη off chip. Συγκεκριμένα μπορεί να έχει έως και 4G words μνήμη δεδομένων και 4M words μνήμη προγράμματος. Αυτός ο περιορισμός προκύπτει από το μήκος των διευθύνσεων της εκάστοτε μνήμης. Για τα δεδομένα οι διευθύνσεις είναι μήκους 32-bit ενώ για το πρόγραμμα είναι 22-bit.

2.3.4 Περιφερειακές Συσκευές [1]

Ο F28335 προσφέρει πληθώρα περιφερειακών συσκευών, πολλές από τις οποίες είναι ειδικά σχεδιασμένες για τις εφαρμογές των ηλεκτρονικών ισχύος.

ePWM (enhanced PWM unit): Με τη βοήθεια αυτής της μονάδας ο DSP μπορεί να παράγει σε συγκεκριμένες ψηφιακές εξόδους σήματα PWM. Ανάλογα με την μαθηματική επεξεργασία μπορούμε να παράγουμε SPWM, SVPWM και γενικότερα οποιαδήποτε περιοδική παλμοσειρά επιθυμούμε. Υπάρχουν 12 κύρια κανάλια διαθέσιμα και 6 βοηθητικά.

eCAP (enhanced Capture Unit): Ο F28335 παρέχει τη δυνατότητα «ανάγνωσης» ενός ψηφιακού σήματος. Αυτό επιτυγχάνεται μέσω του εντοπισμού των ακμών του σήματος και την αντιστοίχηση της εκάστοτε ακμής με την τιμή ενός χρονιστή. Όπως θα δούμε στο αντίστοιχο κεφάλαιο, με κατάλληλο πρόγραμμα μπορούμε να αναγνώσουμε και να αναπαράγουμε οποιαδήποτε παλμοσειρά αρκεί να γνωρίζουμε την περιοδικότητά της.

Analogue to Digital (ADC): Η μονάδα μετατροπής σήματος από αναλογικό σε ψηφιακό απαιτείται στις περισσότερες εφαρμογές με ανάδραση (closed loop). Μετατρέπει τάση εισόδου από 0-3V σε έναν ακέραιο αριθμό μεταξύ 0-4095 (12 bit ακρίβειας).

Watchdog: Μετρητής ο οποίος εγγυάται την ασφάλεια του προγράμματος. Συγκεκριμένα ο Watchdog εκκινεί με την αρχή του προγράμματος, και αν περάσσουν 4.37ms χωρίς να μηδενιστεί τότε το πρόγραμμα σταματάει να τρέχει. Όπως αναφέρεται παραπάνω αυτό αποτελεί μέτρο προστασίας για την CPU και εγγυάται την ορθή λειτουργία των προγραμμάτων. Ο μηδενισμός (και η επανεκκίνηση του WD γίνεται με συγκεκριμένες εντολές

```
SysCtrlRegs.WDCR= 0x00AF;   // Re-enable the watchdog
```

Αυτή η εντολή ενεργοποιεί τον WD και συμπεριλαμβάνεται μία φορά στην αρχή του εκάστοτε προγράμματος

```
SysCtrlRegs.WDKEY = 0x55;    // WD counter will be reset on the next
0xAA write
SysCtrlRegs.WDKEY = 0xAA;   // WD is reset
```

Το ζευγάρι εντολών πρέπει να εμφανίζεται περιοδικά στον κώδικα (το πολύ ανα 4.37ms) ώστε να μην υπερχειλίζει ο WD. Αν για κάποιο λόγο κάτι τέτοιο δε γίνεται σημαίνει ότι υπάρχει λάθος στον προγραμματισμό (για παράδειγμα το πρόγραμμα μπορεί να έχει παγιδευτεί σε έναν ατέρμονο βρόχο)

eQEP (enhanced Quadrature Encoder Positioning)[5]: Το συγκεκριμένο περιφερειακό χρησιμοποιείται σε εφαρμογές που συμπεριλαμβάνουν κυκλική κίνηση ενός άξονα. Το eQEP μας επιτρέπει να παρατηρήσουμε αυτή την κίνηση. Πιθανή εφαρμογή του παραπάνω είναι ο έλεγχος της ταχύτητας ενός οχήματος.

2.3.5 Περιφερειακά επικοινωνίας F28335 με άλλες συσκευές

Serial Communication Interface (SCI) [1]: ασύγχρονη σειριακή επικοινωνία. Είναι ευρύτερα γνωστή σαν UART και χρησιμοποιείται συχνά σύμφωνα με το RS232 πρωτόκολλο.

Serial Peripheral Interface (SPI)[1]: σύγχρονη σειριακή επικοινωνία παρόμοιας μορφής με το SCI.

CAN 2.0B (controller area network) [3]: αποτελεί πρωτόκολλο επικοινωνίας που χρησιμοποιείται σε οχήματα για την επικοινωνία μικροϋπολογιστών χωρίς τη μεσολάβηση ενός υπολογιστικού συστήματος.

Multi Channel Buffered Serial Port (MsBSP)[1]: είναι σύγχρονη σειριακή επικοινωνία με υψηλή μετάδοση δεδομένων. Χρησιμοποιείται συχνά για τη σύνδεση αποκωδικοποιητών εικόνας ή ήχου στον F28335.

I2C (Inter- Integrated Circuit) [4]: μέθοδος σειριακής επικοινωνίας που δημιουργήθηκε από την εταιρία Philips και χρησιμοποιείται για ηλεκτρονικές συσκευές χαμηλής συχνότητας. Το συγκεκριμένο DSC είναι συμβατό με την έκδοση 2.1. Χαρακτηριστικό αυτού του διαύλου είναι ότι απαιτεί μόνο δύο γραμμές, μία για τα σειριακά δεδομένα και μία για το σειριακό ρολόι.

GPIO (General Purpose Input/Output)[1]: Οι ακίδες της αναπτυξιακής πλακέτας (Εικόνα 2.1) λειτουργούν κυρίως σαν ψηφιακές είσοδοι/έξοδοι 0-3,3V. Μπορούν να αξιοποιηθούν ως σήματα ενεργοποίησης σε λογικά κυκλώματα ή σαν είσοδοι εξωτερικών σημάτων. Τέλος κατά την δημιουργία προγραμμάτων τα GPIO λειτουργούν ως μέσα αποσφαλμάτωσης.

2.4 Βιβλιογραφία

[1] «TMS320F28335 Digital Signal Controllers (DSCs) Data Manual» available at http://www.ti.com/lit/ds/sprs439m/sprs439m.pdf

[2] «F2833x tutorial – Module 2» available at Texas Instruments(http://www.ti.com.)

[3] Robert I. Davis, Alan Burns, Reinder J. Bril, Johan J. Lukkien «Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised» 30 January 2007

[4] «I2C-bus specification and user manual» available at http://www.nxp.com/documents/other/UM10204_v5.pdf

[5] «Enhanced Quadrature Encoder Pulse (eQEP) Module» available at Texas Instruments(http://www.ti.com/lit/ug/sprug05a/sprug05a.pdf .)

ΚΕΦΑΛΑΙΟ 3

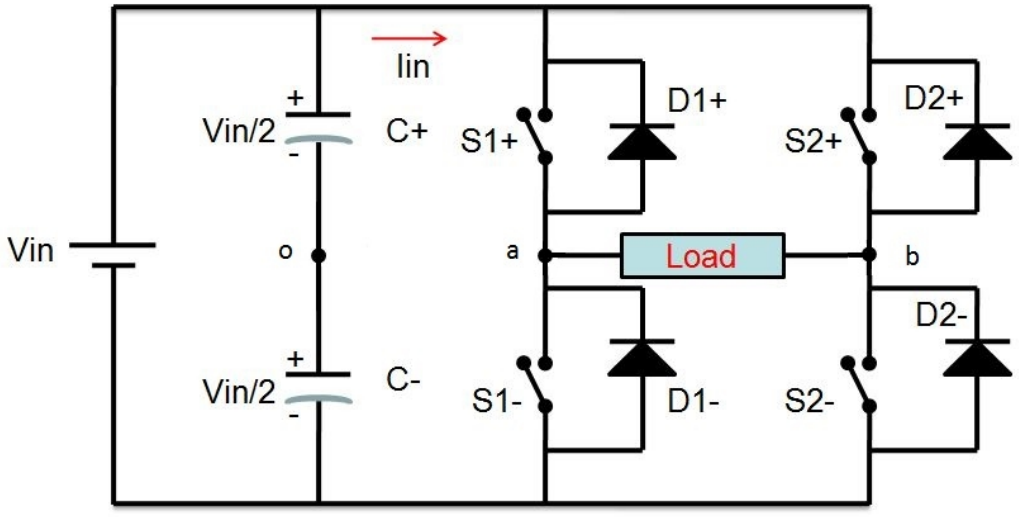
ΘΕΩΡΗΤΙΚΗ ΠΑΡΟΥΣΙΑΣΗ ΤΩΝ ΤΕΧΝΙΚΩΝ ΔΙΑΜΟΡΦΩΣΗΣ SPWM ΚΑΙ SVPWM

3.1 Εισαγωγή

Σε αυτό το κεφάλαιο καταγράφονται οι βασικές αρχές της Ημιτονοειδούς Διαμόρφωσης Εύρους Παλμών (SPWM) για μονοφασική διάταξη και της Διαμόρφωσης Εύρους Παλμών βάση των Χωρικών Διανυσμάτων Τάσης του αντιστροφέα (SVPWM) για τριφασική διάταξη. Η εφαρμογή, η οποία θα παρουσιαστεί στα τελευταία κεφάλαια της διπλωματικής, αφορά διάταξη ανοιχτού βρόχου. Στόχος αυτής της εφαρμογής είναι να αποτελέσει βάση στην οποία θα στηριχθούν μελλοντικές εργασίες που θα συμπεριλαμβάνουν και ανάδραση.

3.2 Ημιτονοειδής Διαμόρφωση Εύρους Παλμών (SPWM) για μονοφασική διάταξη ανοιχτού βρόχου

Στο σχήμα 3.1 φαίνεται η διάταξη στην οποία εφαρμόζεται η τεχνική SPWM. Η είσοδος της διάταξης είναι η τάση εισόδου V_{in} ενώ η έξοδος εντοπίζεται στα άκρα του φορτίου. Οι διακόπτες $S1_{+/-}$ και $S2_{+/-}$ μπορούν να είναι οποιασδήποτε τεχνολογίας (IGBT, SiC κτλ) χωρίς να αλλάζουν οι βασικές αρχές της μεθόδου.



Σχήμα 3.1: Μονοφασική διάταξη αντιστροφέα διπλής γέφυρας

Σε επίπεδο κυκλώματος ελέγχου δημιουργούνται δύο ημιτονοειδή σήματα αναφοράς τα οποία έχουν μεταξύ τους διαφορά 180 μοιρών και αντιστοιχούν στις δύο

γέφυρες του κυκλώματος. Η ημιτονική τάση που θα παραχθεί στην έξοδο θα είναι συμφασική με το ημίτονο αναφοράς της γέφυρας $S1_{+/-}$. Εδώ να σημειωθεί ότι ο συμβολισμός +/- δεν είναι τυχαίος καθώς το ψηφιακό σήμα ελέγχου που φτάνει στον $S1_{+}$ είναι αντίστροφο του σήματος που φτάνει στον $S1_{-}$. Αυτό πρέπει να αληθεύει κάθε στιγμή να μην βραχυκυκλώνονται ποτέ τα άκρα της εισόδου.

Για να παραχθεί η παλμοσειρά ελέγχου των ημιαγωγών S τα σήματα αναφοράς συγκρίνονται με ένα τρίτο σήμα τριγωνικής μορφής το οποίο ονομάζεται φέρον. Εάν η αναφορά είναι μεγαλύτερη του φέροντος τότε οι διακόπτες S_{+} είναι σε κατάσταση ON. Στην αντίθετη περίπτωση είναι OFF. Οι διακόπτες S_{-} ακολουθούν αντίστροφη συμπεριφορά.

Οι περιορισμοί για το φέρον και τον φορέα είναι:

1) Τα σήματα πρέπει να είναι συγχρονισμένα μεταξύ τους.
2) Η συχνότητα του φέροντος πρέπει να είναι ακέραιο πολλαπλάσιο της συχνότητας αναφοράς. (πχ. για σήμα αναφοράς 50Hz αποδεκτή συχνότητα φέροντος είναι τα 400Hz αλλά όχι τα 320Hz)

Παρακάτω ορίζονται τα βασικά μεγέθη αυτής της μεθόδου:

A_r = πλάτος κυματομορφής αναφοράς

A_c = πλάτος κυματομορφής φέροντος

$T_r = 1/f_r$ = περίοδος κυματομορφής αναφοράς

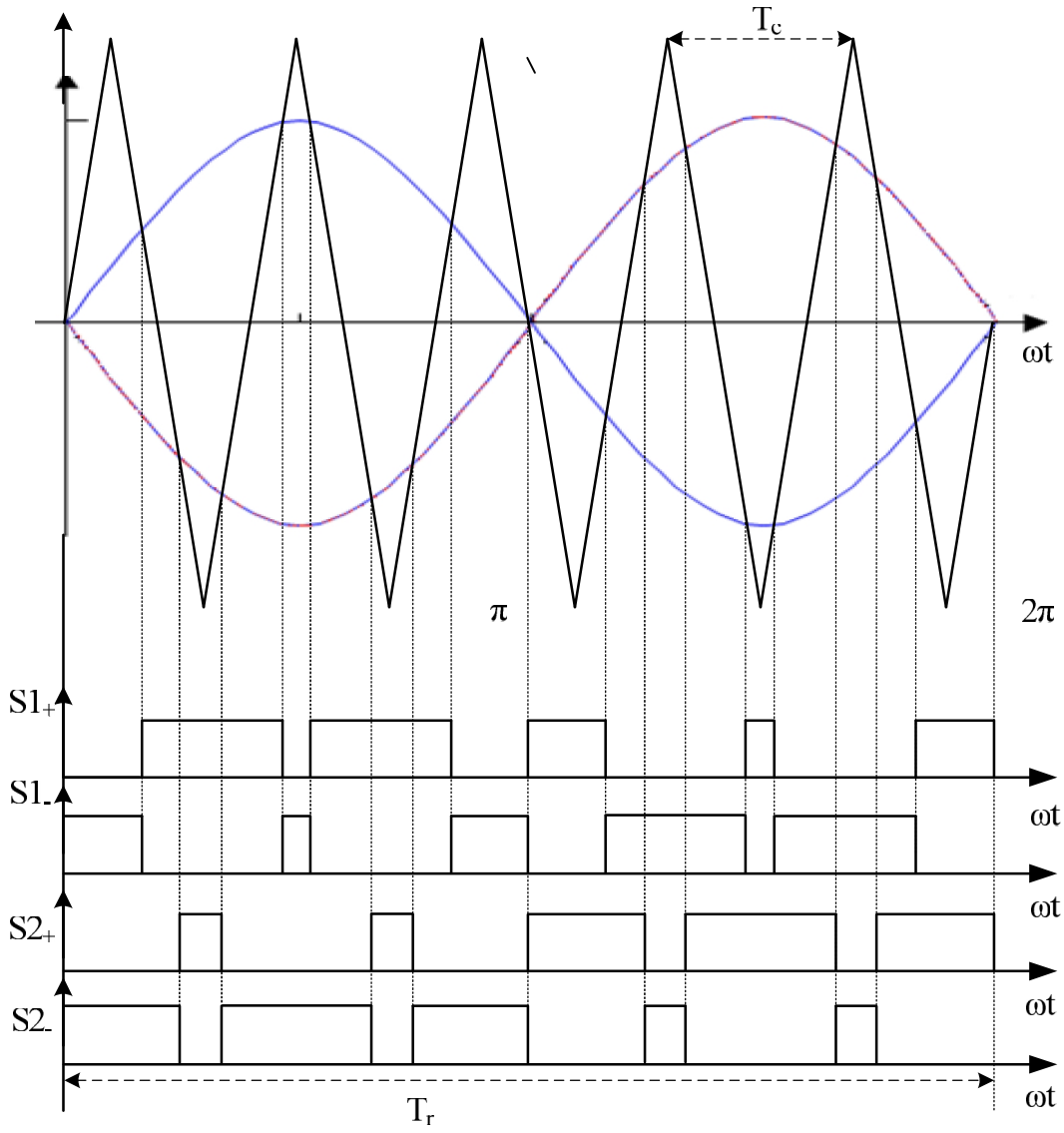
$T_c = 1/f_c$ = περίοδος κυματομορφής φέροντος

$m_a = A_r/A_c$ = συντελεστής διαμόρφωσης πλάτους ($0<m_a<1$)

$m_f = f_c/f_r$ =συντελεστής διαμόρφωσης συχνότητας

Η συχνότητα του σήματος αναφοράς καθορίζει τη συχνότητα του σήματος εξόδου, ενώ η συχνότητα φέροντος καθορίζει τη διακοπτική συχνότητα των ημιαγωγικών διακοπτών του αντιστροφέα και τη συχνότητα των ανωτέρων αρμονικών του σήματος εξόδου. Η παραπάνω τεχνική αναπαρίσταται στο σχήμα 3.2.

Στην περίπτωση που ο έλεγχος παράγεται από DSC δεν απαιτούνται επιπλέον συγκριτές. Η εξαγωγή των παλμοσειρών γίνεται απευθείας από τα pin του DSC.

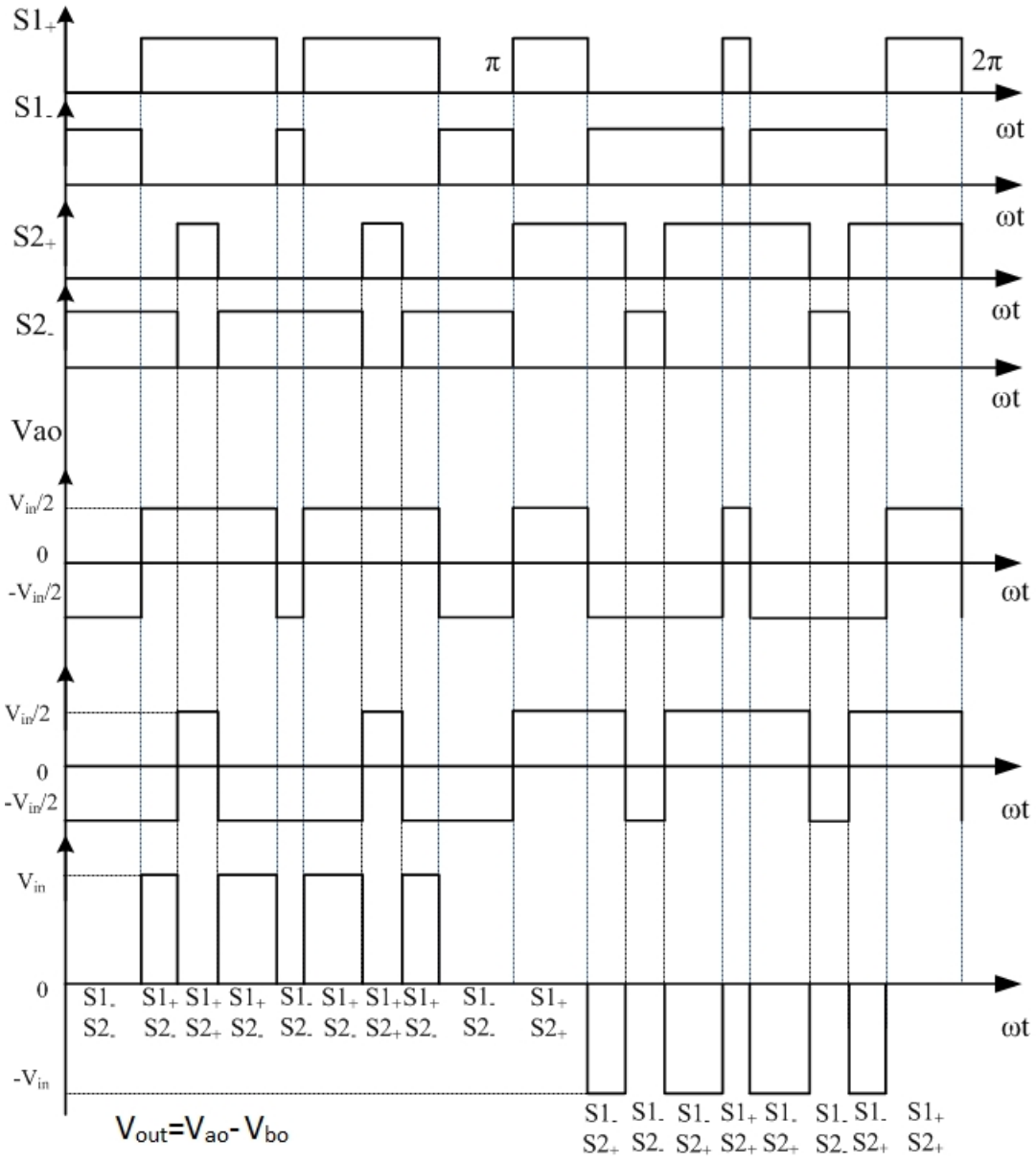


Σχήμα 3.2: Αναπαράσταση παλμών ενεργοποίησης των διακοπτών σε τεχνική SPWM για m_f=5 και m_a=0,7

Στο σχήμα 3.3 παρουσιάζονται οι φασικές τάσεις v_{ao} και v_{bo} και η τάση εξόδου v_{out}.

Η σχέση που συνδέει αυτές τις τάσεις είναι:

$$V_{out} = V_{ab} = V_{ao} - V_{bo} \tag{3.1}$$



Σχήμα 3.3: Μορφή τάσης εξόδου σε τεχνική SPWM με m_f=5 και ma=0,7

Τα παραπάνω στοιχεία επαρκούν για να κατασκευαστεί ένα πρόγραμμα που θα παρέχει στους διακόπτες τις κατάλληλες παλμοσειρές ώστε να παραχθεί ημιτονοειδής έξοδος στο φορτίο με τεχνική SPWM.

3.3 Διαμόρφωση Εύρους Παλμών με βάση τα Χωρικά Διανύσματα Τάσης του Αντιστροφέα (Space Vector Pulse width modulation-SVPWM) για τριφασική διάταξη ανοιχτού βρόχου

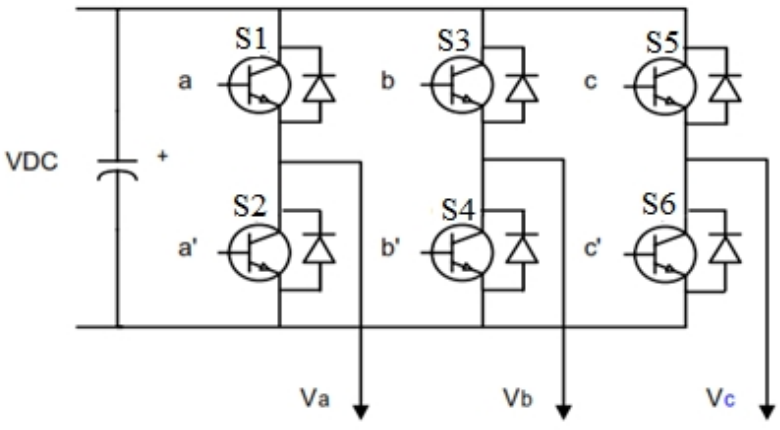
Η τεχνική διαμόρφωσης Εύρους Παλμών με βάση τα Χωρικά Διανύσματα Τάσης του Αντιστροφέα (SVPWM) έχει ξεχωρίσει τα τελευταία χρόνια στις εφαρμογές των τριφασικών αντιστροφέων. Τα κύρια πλεονεκτήματά της είναι:

- Το μεγαλύτερο πλάτος της βασικής συνιστώσας της τάσης
- Η καλύτερη ποιότητα στην τάση εξόδου, δηλαδή μικρότερο THD (total harmonic distortion) σε σχέση με την SPWM

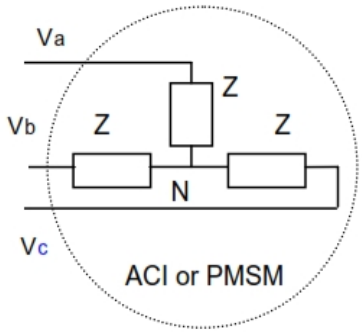
Στο κεφάλαιο αυτό παρουσιάζονται συνοπτικά τα **βασικά στοιχεία θεωρίας της SVPWM-dq** και στη συνέχεια ακολουθεί ο **αλγόριθμος παραγωγής παλμών για ένα τριφασικό αντιστροφέα**. Η συγκεκριμένη εφαρμογή είναι ανοιχτού βρόχου, ωστόσο ο ίδιος κώδικας μπορεί να χρησιμοποιηθεί και για συστήματα με ανάδραση.

3.3.1 Θεωρία του SVPWM [1,2,3]

Για την διάταξη του τριφασικού αντιστροφέα (Σχήμα 3.4), και έχοντας στο μυαλό μας ότι οι διακόπτες του ίδιου κλάδου έχουν υποχρεωτικά συμπληρωματική λειτουργία, υπάρχουν **8 δυνατές διακοπτικές καταστάσεις**. Οπότε, αν το φορτίο που συνδέεται στους ακροδέκτες του αντίστροφέα είναι συμμετρικό και σε διάταξη αστέρα (Σχήμα 3.5) μπορούμε να σχηματίσουμε τον παρακάτω πίνακα για τα πιθανά ισοδύναμα κυκλώματα (Πίνακας 3.1):



Σχήμα 3.4: Τριφασικός αντιστροφέας [2]

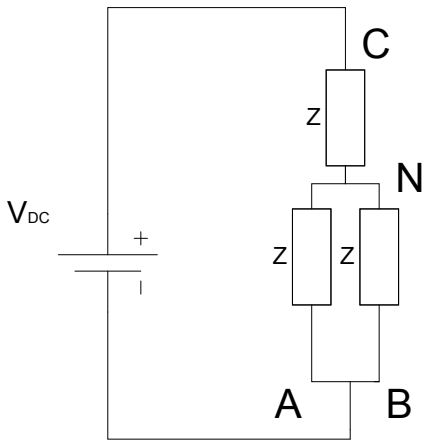


Σχήμα 3.5: Τριφασικό φορτίο [2]

Πίνακας 3.1: Οι οκτώ πιθανοί διακοπτικοί συνδυασμοί και οι αντίστοιχες φασικές και πολικές τάσεις του αντιστροφέα

Διακοπτικές καταστάσεις			**Φασικές τάσεις εξόδου**			**Πολικές τάσεις εξόδου**		
c	**b**	**a**	V_{AN}	V_{BN}	V_{CN}	V_{AB}	V_{BC}	V_{CA}
0	0	0	0	0	0	0	0	0
0	0	1	$2V_{DC}/3$	$-V_{DC}/3$	$-V_{DC}/3$	V_{DC}	0	$-V_{DC}$
0	1	0	$-V_{DC}/3$	$2V_{DC}/3$	$-V_{DC}/3$	$-V_{DC}$	V_{DC}	0
0	1	1	$V_{DC}/3$	$V_{DC}/3$	$-2V_{DC}/3$	0	V_{DC}	$-V_{DC}$
1	0	0	$-V_{DC}/3$	$-V_{DC}/3$	$2V_{DC}/3$	0	$-V_{DC}$	V_{DC}
1	0	1	$V_{DC}/3$	$-2V_{DC}/3$	$V_{DC}/3$	V_{DC}	$-V_{DC}$	0
1	1	0	$-2V_{DC}/3$	$V_{DC}/3$	$V_{DC}/3$	$-V_{DC}$	0	V_{DC}
1	1	1	0	0	0	0	0	0

Παράδειγμα: Σε αυτό το παράδειγμα θα αναλύσουμε τις τάσεις (φασικές και πολικές) του αντιστροφέα για τη διακοπτική κατάσταση 100.



Σχήμα 3.6: Ισοδύναμο κύκλωμα για τη διακοπτική κατάσταση 100

$$Z_{(A\equiv B)N} = Z//Z = \frac{Z}{2} \tag{3.2}$$

$$V_N = V_{DC}\frac{Z/2}{Z/2+Z} = V_{DC}\frac{Z/2}{3Z/2} = \frac{V_{DC}}{3} \tag{3.3}$$

Άρα:

$$V_{CN} = V_C - V_N = V_{DC} - \frac{V_{DC}}{3} = 2\frac{V_{DC}}{3} \tag{3.4}$$

$$V_{AN} = V_{BN} = V_B - V_N = 0 - \frac{V_{DC}}{3} = -\frac{V_{DC}}{3} \tag{3.5}$$

$$V_{AB} = V_A - V_B = 0 - 0 = 0 \tag{3.6}$$

$$V_{BC} = V_B - V_C = 0 - V_{DC} = -V_{DC} \tag{3.7}$$

$$V_{CA} = V_C - V_A = V_{DC} - 0 = V_{DC} \tag{3.8}$$

Εάν στις φασικές τιμές εξόδου εφαρμόσουμε τον μετασχηματισμό Clarke (3.9) μπορούμε να κατασκευάσουμε τον Πίνακας 3.2. Υπενθυμίζουμε τις σχέσεις που συνδέουν το σύστημα Vabc με Vαβ (ή dq):

$$\begin{aligned} V_{alpha} &= V_{\mathrm{a}} \\ V_{beta} &= \frac{V_{\mathrm{a}} + 2V_b}{\sqrt{3}} \end{aligned} \tag{3.9}$$

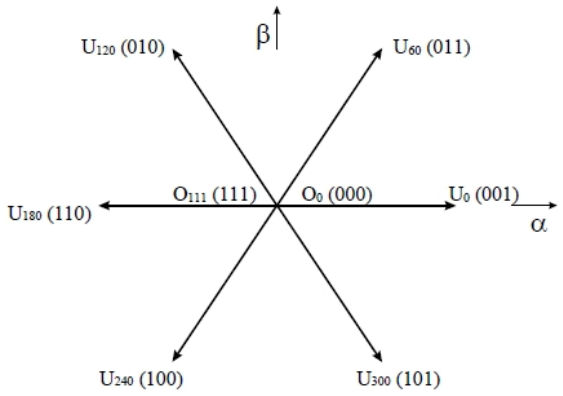
(Προφανώς το σύστημα είναι συμμετρικό οπότε η τρίτη συνιστώσα είναι μηδενική)

Πίνακας 3.2: Φασικές τάσεις εξόδου για όλες τις διακοπτικές καταστάσεις

Διακοπτικές καταστάσεις			**Φασικές τάσεις εξόδου μετασχηματισμένες κατά Clarke**		**Διάνυσμα (Vector)**
c	**b**	**a**	$\mathbf{V_{sα}}$	$\mathbf{V_{sβ}}$	
0	0	0	0	0	O_0
0	0	1	$2V_{DC}/3$	0	V_0
0	1	0	$-V_{DC}/3$	$V_{DC}/\sqrt{3}$	V_{120}
0	1	1	$V_{DC}/3$	$V_{DC}/\sqrt{3}$	V_{60}
1	0	0	$-V_{DC}/3$	$-V_{DC}/\sqrt{3}$	V_{240}

1	0	1	$V_{DC}/3$	$-V_{DC}/\sqrt{3}$	V_{300}
1	1	0	$-2V_{DC}/3$	0	V_{180}
1	1	1	0	0	O_{111}

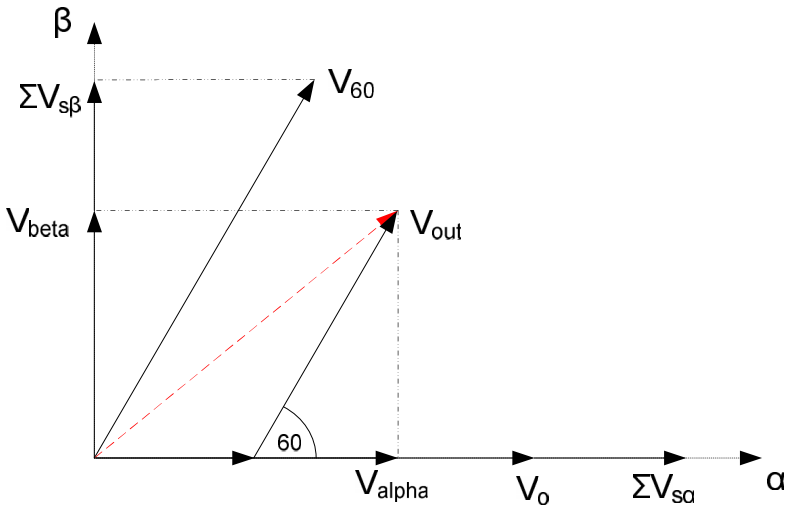
Οι τάσεις Vsα και Vsβ είναι οι (α,β) συνιστώσες των βασικών διανυσμάτων χώρου (space vectors) και αντιστοιχούν σε μία συγκεκριμένη διακοπτική κατάσταση των ημιαγωγών (c,b,a). Εποπτικά μπορούμε να δούμε τους τομείς που ορίζονται από τα διανύσματα του Πίνακας 3.2 στο παρακάτω σχήμα (Σχήμα 3.7):



Σχήμα 3.7: Τα βασικά διανίσματα χώρου σχετικά με τους άξονες α,β [2]

Στόχος της μεθόδου SVPWM είναι **να προσεγγίσει στην έξοδο μία δεδομένη τάση αναφοράς (V_{ref}).** Η αναφορά αυτή θα είναι κάποιο διάνυσμα εντός του κύκλου και η προσέγγιση γίνεται με την εφαρμογή **των παράπλευρων βασικών** (V_{60} και V_0 για το διπλανό σχήμα) και των **μηδενικών διανυσμάτων (O_0 και O_{111})** για συγκεκριμένους χρόνους. Στην παραπάνω φράση τα «βασικά διανύσματα» είναι νοηματικά ισοδύναμα με τις «διακοπτικές καταστάσεις».

Πιο συγκεκριμένα ας δούμε πως αναλύουμε τη συγκεκριμένη τάση αναφοράς σε συνιστώσες βασικών διανυσμάτων και κατ' επέκταση σε χρόνους εφαρμογής διακοπτικών καταστάσεων:



Σχήμα 3.8: Ανάλυση του διανύσματος αναφοράς (ταυτόσημο με το Uout) σε βασικά διανύσματα

Το $\Sigma V_{s\beta}$ και το $\Sigma V_{s\alpha}$ αποτελούν τις συνιστώσες (α,β) του αθροίσματος $\vec{V}_0 + \vec{V}_{60}$. Μπορούμε να γράψουμε τα εξής για την U_{out}:

$$U_{out} = \frac{T_1}{T} V_0 + \frac{T_3}{T} V_{60} \tag{3.10}$$

$$T = T_1 + T_0 + T_3 \tag{3.11}$$

,όπου T= περίοδος παλμών ,

T_1= χρόνος εφαρμογής του V_0,

T_3= χρόνος εφαρμογής του V_{60},

T_0= χρόνος εφαρμογής των $O_{0/111}$

Ο υπολογισμός αυτών των χρόνων γίνεται παρακάτω:

$$\left.\begin{aligned} V_{beta} &= \frac{T_3}{T}\left|V_{60}\right|\sin(60^\circ) \\ V_{alpha} &= \frac{T_1}{T}\left|V_0\right| + \frac{T_3}{T}\left|V_{60}\right|\cos(60^\circ) \end{aligned}\right\} \xrightarrow{\frac{|V_x|}{V_{DC}/\sqrt{3}} = 2/\sqrt{3}} \left.\begin{aligned} T_1 &= \frac{T}{2}(\sqrt{3}V_{alpha} - V_{beta}) \\ T_3 &= TV_{beta} \end{aligned}\right\} \tag{3.12}$$

Το μέτρο των βασικών διανυσμάτων είναι ίσο με $2V_{DC}/3$ (οι υπολογισμοί είναι τετριμμένοι). Κανονικοποιώντας τα με την τιμή $V_{DC}/\sqrt{3}$ (μέγιστη φασική τάση) το μέτρο γίνεται ίσο με $2/\sqrt{3}$. Επίσης πρέπει να σημειωθεί ότι και τα V_{alpha} και V_{beta} θεωρούνται κανονικοποιημένα στην τιμή $V_{DC}/\sqrt{3}$. Διαιρώντας τα T_1,T_3 με τη συνολική περίοδο T έχουμε τις ποσοστιαίες τιμές εφαρμογής της κάθε διακοπτικής κατάστασης t_1,t_2 (στην ουσία το duty cycle). Ο χρόνος που περισσεύει προφανώς αφορά τα μηδενικά διανύσματα O_0,O_{111}. Οπότε:

$$\begin{aligned} t_1 &= \frac{T_1}{T} = \frac{1}{2}(\sqrt{3}V_{alpha} - V_{beta}) \\ t_2 &= \frac{T_3}{T} = V_{beta} \end{aligned} \tag{3.13}$$

Μέσω της ίδιας διαδικασίας, αν η αναφορά ήταν στον τομέα V_{240} και V_{300} τότε οι χρόνοι t_1,t_2 θα είχαν την παρακάτω μορφή:

$$\begin{aligned} t_1 &= \frac{T_5}{T} = -\frac{1}{2}(\sqrt{3}V_{alpha} + V_{beta}) \\ t_2 &= \frac{T_6}{T} = \frac{1}{2}(\sqrt{3}V_{alpha} - V_{beta}) \end{aligned} \tag{3.14}$$

Επαναλαμβάνοντας την παραπάνω διαδικασία για όλους τους τομείς καταλήγουμε στο συμπέρασμα ότι οι δυνατές τιμές των t_1,t_2 είναι 3 οπότε δηλώνουμε τις εξής μεταβλητές:

$$
\begin{aligned}
X &= V_{beta} \\
Y &= \frac{1}{2}(\sqrt{3}V_{alpha} + V_{beta}) \\
Z &= \frac{1}{2}(-\sqrt{3}V_{alpha} + V_{beta})
\end{aligned}
\tag{3.15}
$$

Και στη συνέχεια σχηματίζουμε τον παρακάτω συγκεντρωτικό πίνακα:

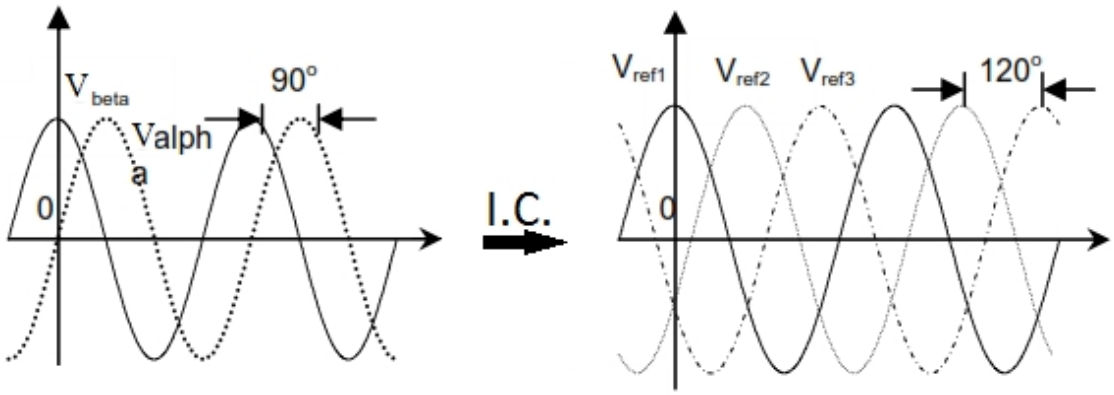
Πίνακας 3.3: Αντιστοίχιση τομέα με τιμές των t_1, t_2

Τομέας	$\mathbf{V_0,V_{60}}$	$\mathbf{V_{60},V_{120}}$	$\mathbf{V_{120},V_{180}}$	$\mathbf{V_{180},V_{240}}$	$\mathbf{V_{240},V_{300}}$	$\mathbf{V_{300},V_0}$
t_1	-Z	Z	X	-X	-Y	Y
t_2	X	Y	-Y	Z	-Z	-X

Για να γνωρίζουμε σε ποιο τομέα είναι το διάνυσμα αναφοράς μας μετατρέπουμε το V_{out} από (α,β) σε ένα τριφασικό συμμετρικό σύστημα. Στην περίπτωση αυτή, δε χρησιμοποιούμε το συμβατικό αντίστροφο μετασχηματισμό Clarke αλλά μία παραλλαγή του:

$$
\begin{cases}
V_{ref1} = V_{beta} \\
V_{ref2} = \dfrac{\sqrt{3}V_{alpha} - V_{beta}}{2} \\
V_{ref3} = \dfrac{-\sqrt{3}V_{alpha} - V_{beta}}{2}
\end{cases}
\tag{3.16}
$$

Πρέπει να παρατηρηθεί ότι σε σχέση με τον κανονικό μετασχηματισμό, αυτός προηγείται κατά 90^o. Άρα σχηματικά γίνεται ο παρακάτω μετασχηματισμός:



Με αυτόν τον μετασχηματισμό είμαστε έτοιμοι να αποφασίσουμε σε ποιο τομέα είμαστε κάθε φορά (μέσω της εκάστοτε τιμής των V_{ref}).

Αυτό υπολογίζεται από τον τύπο:

$$P = 4c + 2b + a \tag{3.17}$$

όπου:

- Αν $V_{ref1}>0$ a=1 αλλιώς a=0
- Αν $V_{ref2}>0$ b=1 αλλιώς b=0
- Αν $V_{ref3}>0$ c=1 αλλιώς c=0

Η αντιστοιχία του αριθμού P με τον εκάστοτε τομέα είναι:

Πίνακας 3.4: Αντιστοίχιση P με αριθμό τομέα

P	1	2	3	4	5	6
Τομέας	II	VI	I	IV	III	V

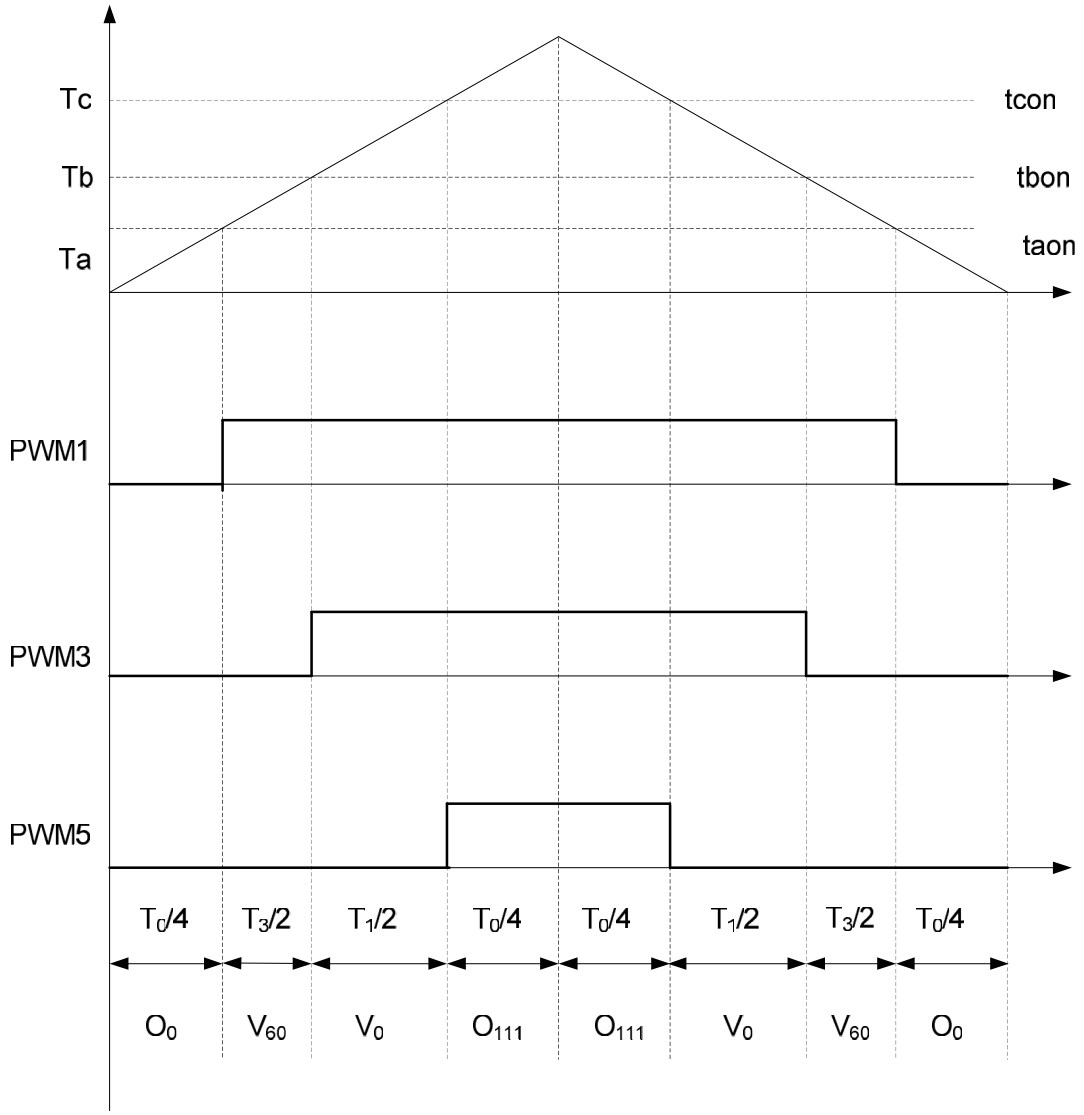
Η σχέση (3.18) και ο πίνακας 3.4 καθορίζουν τους χρόνους αγωγής του κάθε διακοπτικού στοιχείου:

$$\begin{cases} t_{aon} = \dfrac{PWMPRD - t_1 - t_2}{2} \\ t_{bon} = t_{aon} + t_1 \\ t_{con} = t_{bon} + t_2 \end{cases} \tag{3.18}$$

Πίνακας 3.5:Αντιστοίχηση Duty Cycles σε ημιγέφυρες (Ta→Q1, Tb→Q3, Tc→Q5)

Τομέας	U_0,U_{60}	U_{60},U_{120}	U_{120},U_{180}	U_{180},U_{240}	U_{240},U_{300}	U_{300},U_0
Ta	taon	tbon	tcon	tcon	tbon	taon
Tb	tbon	taon	taon	tbon	tcon	tcon
Tc	tcon	tcon	tbon	taon	taon	tbon

Παρακάτω φαίνεται ένα παράδειγμα για διάνυσμα αναφοράς στον πρώτο τομέα (V_0,V_{60})



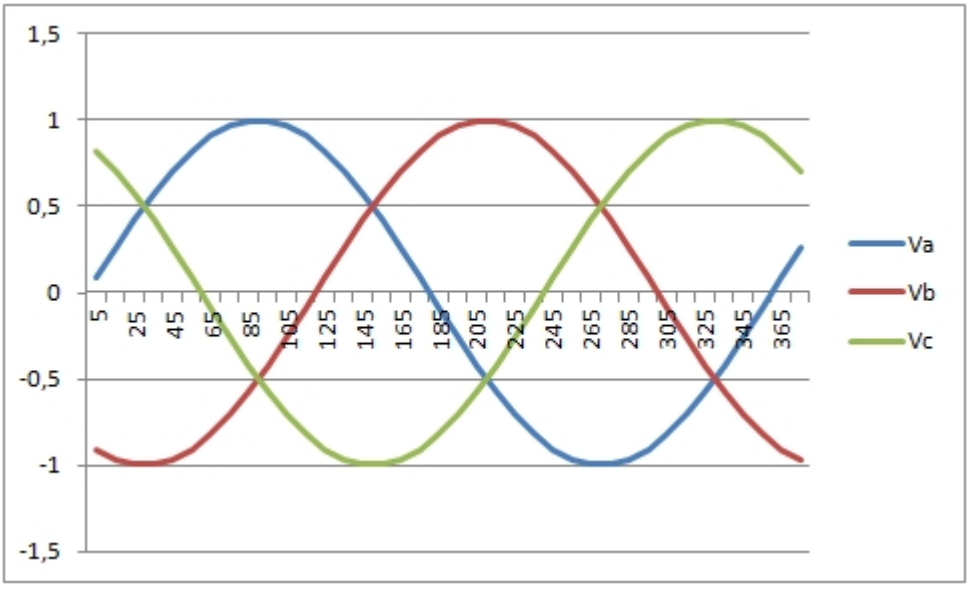
Σχήμα 3.9: Ποιοτικό παράδειγμα σημάτων ελέγχου για διάνυσμα αναφοράς σε τυχαία θέση του πρώτου τομέα

Σε αυτό το σημείο είναι διαθέσιμες όλες οι απαραίτητες πληροφορίες ώστε να δημιουργηθεί ένα πρόγραμμα που να παράγει παλμούς SVPWM και να οδηγεί τριφασικό αντιστροφέα.

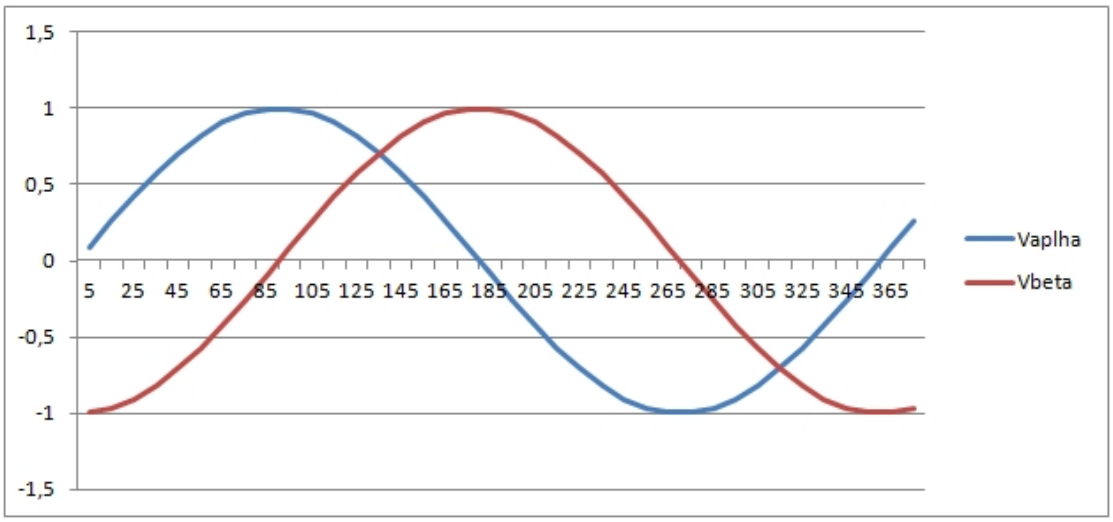
Συνοψίζοντας, η διαδικασία που ακολουθείται είναι:

- Ανάκτηση τιμών ημιτόνων αναφοράς, είτε από έτοιμο πίνακα της μνήμης (ανοιχτού βρόχου πρόγραμμα) είτε από αναδράσεις (κλειστός βρόχος)
- Μετασχηματισμός των τιμών αυτών κατά Clarke στο (α,β) σύστημα (σχέση (3.9)).
- Υπολογισμός των X,Y,Z (σχέση (3.15)).
- Υπολογισμός των τιμών V_{ref} (σχέση (3.16)).
- Καθορισμός του τομέα στον οποίον βρίσκεται το διάνυσμα αναφοράς μέσω της σχέσης (3.17) και του πίνακα 3.4.
- Υπολογισμός των t_{aon}, t_{bon},t_{con} (σχέση (3.18)).
- Αντιστοίχιση των t_{xon} με τους συντελεστές χρήσης των διακοπτών (Ta, Tb, Tc-Πίνακας 3.5)

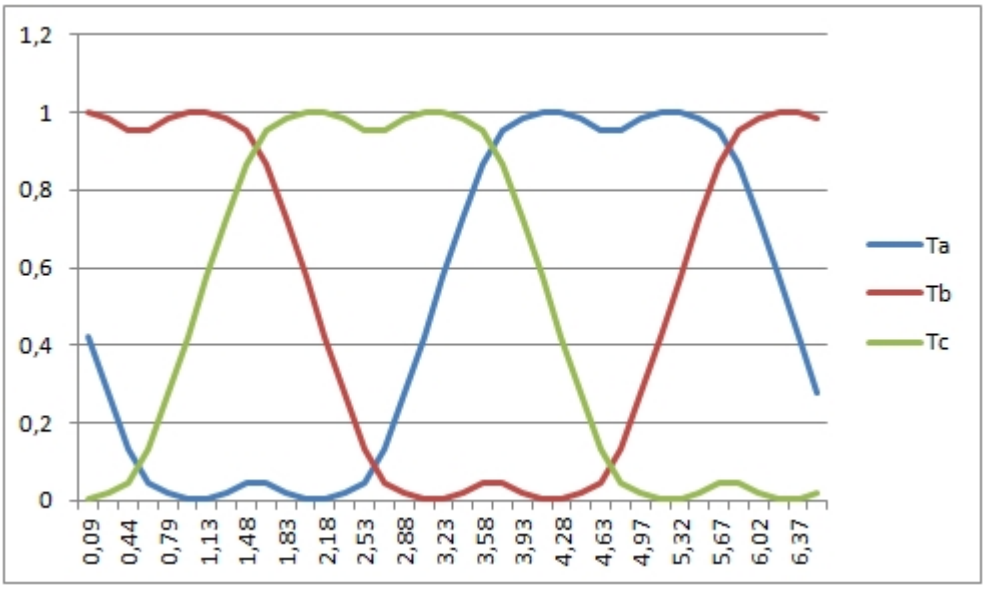
Ακολουθώντας την παραπάνω διαδικασία και με τη βοήθεια του excel δημιουργήσαμε τις παρακάτω γραφικές παραστάσεις. Οι τιμές για τα διανύσματα αναφοράς ήταν 36 (βήμα 5 μοιρών) αρχίζοντας από τις 5 μοίρες, το οποίο αποτελεί μία πλήρη περιστροφή του διανύσματος χώρου. Ο πίνακας 3.6 περιέχει το υπολογιστικό φύλλο της διαδικασίας.



Σχήμα 3.10: Τριφασικό σύστημα αναφοράς



Σχήμα 3.11: Μετασχηματισμός της αναφοράς κατά Clarke.



Σχήμα 3.12: Συντελεστές χρήσης των διακοπτών S1 (Ta), S3(Tb), S5(Tc)

Για να γίνει κατανοητή η μέθοδος ακολουθεί ένα αριθμητικό παράδειγμα για μία τυχαία γωνία:

- Τιμές τάσεων τριφασικού συστήματος για 135 μοίρες:

$$a(rad) = a(\deg) \cdot \frac{\pi}{180^o} \Rightarrow a(rad) = 135 \cdot \frac{\pi}{180} = 2,36rad \tag{3.19}$$

$$V_a = \sin(a(rad)) = \sin(2,35) = 0,7 \tag{3.20}$$

$$V_b = \sin(a(rad) - 2\frac{\pi}{3}) = 0,26 \tag{3.21}$$

$$V_c = \sin(a(rad) + 2\frac{\pi}{3}) = -0,97 \tag{3.22}$$

- Μετασχηματισμός των τιμών αυτών κατά Clarke στο (α,β) σύστημα

$$\begin{aligned} V_{alpha} &= V_{\mathrm{a}} \Rightarrow V_a = 0,7 \\ V_{beta} &= \frac{V_{\mathrm{a}} + 2V_b}{\sqrt{3}} \Rightarrow V_b = \frac{0,7 + 2 \cdot 0,26}{\sqrt{3}} = 0,7 \end{aligned} \tag{3.23}$$

- Υπολογισμός των X,Y,Z

$$\begin{aligned} X &= V_{beta} \Rightarrow X = 0,7 \\ Y &= \frac{1}{2}(\sqrt{3}V_{alpha} + V_{beta}) = \frac{1}{2}(\sqrt{3} \cdot 0,7 + 0,7) \Rightarrow Y = 0,97 \\ Z &= \frac{1}{2}(-\sqrt{3}V_{alpha} + V_{beta}) = \frac{1}{2}(-\sqrt{3} \cdot 0,7 + 0,7) \Rightarrow Z = -0,26 \end{aligned} \tag{3.24}$$

- Υπολογισμός των τιμών V_{ref}

$$\begin{cases} V_{ref1} = V_{beta} \\ V_{ref2} = \dfrac{\sqrt{3}V_{alpha} - V_{beta}}{2} \\ V_{ref3} = \dfrac{-\sqrt{3}V_{alpha} - V_{beta}}{2} \end{cases} \Rightarrow \begin{cases} V_{ref1} = 0,7 \\ V_{ref2} = 0,26 \\ V_{ref3} = -0,97 \end{cases} \tag{3.25}$$

- Καθορισμός του τομέα στον οποίον βρίσκεται το διάνυσμα αναφοράς

$V_{ref1}>0$ άρα a=1
$V_{ref2}>0$ άρα b=1
$V_{ref3}<0$ άρα c=0

Άρα:

$$P = 4c + 2b + a = 2 + 1 = 3 \tag{3.26}$$

Σύμφωνα με τον Πίνακα 3.3 για P=1 το διάνυσμα αναφοράς βρίσκεται στον τομέα I.

Για τον πρώτο τομέα ισχύει ότι **t1=-Z** και **t2=X.**

- Υπολογισμός των t_{aon}, t_{bon},t_{con} .

$$\begin{cases} t_{aon} = \dfrac{1-t_1-t_2}{2} \\ t_{bon} = t_{aon} + t_1 \\ t_{con} = t_{bon} + t_2 \end{cases} \Rightarrow \begin{cases} t_{aon} = \dfrac{1+Z-X}{2} \\ t_{bon} = t_{aon} - Z \\ t_{con} = t_{bon} + X \end{cases} \Rightarrow \begin{cases} t_{aon} = \dfrac{1-0,26-0,7}{2} \\ t_{bon} = t_{aon} + 0,26 \\ t_{con} = t_{bon} + 0,7 \end{cases} \Rightarrow \begin{cases} t_{aon} = 0,017 \\ t_{bon} = 0,277 \\ t_{con} = 0,983 \end{cases}$$

(3.27)

Πίνακας 3.6: Φύλλο υπολογισμού τιμών για τα σχήματα 3.10, 3.11, 3.12

α(°)	α(rad)	Va	Vb	Vc	Valpha	Vbeta	X	Y	Z
5	0,09	0,087156	-0,906308	0,819152	0,087156	-0,99619	-0,99619	-0,42262	-0,57358
15	0,26	0,258819	-0,965926	0,707107	0,258819	-0,96593	-0,96593	-0,25882	-0,70711
25	0,44	0,422618	-0,996195	0,573576	0,422618	-0,90631	-0,90631	-0,08716	-0,81915
35	0,61	0,573576	-0,996195	0,422618	0,573576	-0,81915	-0,81915	0,087156	-0,90631
45	0,79	0,707107	-0,965926	0,258819	0,707107	-0,70711	-0,70711	0,258819	-0,96593
55	0,96	0,819152	-0,906308	0,087156	0,819152	-0,57358	-0,57358	0,422618	-0,99619
65	1,13	0,906308	-0,819152	-0,08716	0,906308	-0,42262	-0,42262	0,573576	-0,99619
75	1,31	0,965926	-0,707107	-0,25882	0,965926	-0,25882	-0,25882	0,707107	-0,96593
85	1,48	0,996195	-0,573576	-0,42262	0,996195	-0,08716	-0,08716	0,819152	-0,90631
95	1,66	0,996195	-0,422618	-0,57358	0,996195	0,087156	0,087156	0,906308	-0,81915
105	1,83	0,965926	-0,258819	-0,70711	0,965926	0,258819	0,258819	0,965926	-0,70711
115	2,01	0,906308	-0,087156	-0,81915	0,906308	0,422618	0,422618	0,996195	-0,57358
125	2,18	0,819152	0,0871557	-0,90631	0,819152	0,573576	0,573576	0,996195	-0,42262
135	2,36	0,707107	0,258819	-0,96593	0,707107	0,707107	0,707107	0,965926	-0,25882
145	2,53	0,573576	0,4226183	-0,99619	0,573576	0,819152	0,819152	0,906308	-0,08716
155	2,71	0,422618	0,5735764	-0,99619	0,422618	0,906308	0,906308	0,819152	0,087156
165	2,88	0,258819	0,7071068	-0,96593	0,258819	0,965926	0,965926	0,707107	0,258819
175	3,05	0,087156	0,819152	-0,90631	0,087156	0,996195	0,996195	0,573576	0,422618
185	3,23	-0,08716	0,9063078	-0,81915	-0,08716	0,996195	0,996195	0,422618	0,573576
195	3,40	-0,25882	0,9659258	-0,70711	-0,25882	0,965926	0,965926	0,258819	0,707107
205	3,58	-0,42262	0,9961947	-0,57358	-0,42262	0,906308	0,906308	0,087156	0,819152
215	3,75	-0,57358	0,9961947	-0,42262	-0,57358	0,819152	0,819152	-0,08716	0,906308
225	3,93	-0,70711	0,9659258	-0,25882	-0,70711	0,707107	0,707107	-0,25882	0,965926
235	4,10	-0,81915	0,9063078	-0,08716	-0,81915	0,573576	0,573576	-0,42262	0,996195
245	4,28	-0,90631	0,819152	0,087156	-0,90631	0,422618	0,422618	-0,57358	0,996195
255	4,45	-0,96593	0,7071068	0,258819	-0,96593	0,258819	0,258819	-0,70711	0,965926
265	4,63	-0,99619	0,5735764	0,422618	-0,99619	0,087156	0,087156	-0,81915	0,906308
275	4,80	-0,99619	0,4226183	0,573576	-0,99619	-0,08716	-0,08716	-0,90631	0,819152
285	4,97	-0,96593	0,258819	0,707107	-0,96593	-0,25882	-0,25882	-0,96593	0,707107

295	5,15	-0,90631	0,0871557	0,819152	-0,90631	-0,42262	-0,42262	-0,99619	0,573576
305	5,32	-0,81915	-0,087156	0,906308	-0,81915	-0,57358	-0,57358	-0,99619	0,422618
315	5,50	-0,70711	-0,258819	0,965926	-0,70711	-0,70711	-0,70711	-0,96593	0,258819
325	5,67	-0,57358	-0,422618	0,996195	-0,57358	-0,81915	-0,81915	-0,90631	0,087156
335	5,85	-0,42262	-0,573576	0,996195	-0,42262	-0,90631	-0,90631	-0,81915	-0,08716
345	6,02	-0,25882	-0,707107	0,965926	-0,25882	-0,96593	-0,96593	-0,70711	-0,25882
355	6,20	-0,08716	-0,819152	0,906308	-0,08716	-0,99619	-0,99619	-0,57358	-0,42262
365	6,37	0,087156	-0,906308	0,819152	0,087156	-0,99619	-0,99619	-0,42262	-0,57358
375	6,54	0,258819	-0,965926	0,707107	0,258819	-0,96593	-0,96593	-0,25882	-0,70711

Vref1	Vref2	Vref3	P	Sector	T1	T2	Taon	Tbon	Tcon	Ta	Tb	Tc
-0,996	0,574	0,423	6,000	5,000	0,423	0,574	0,002	0,425	0,998	0,425	0,998	0,002
-0,966	0,707	0,259	6,000	5,000	0,259	0,707	0,017	0,276	0,983	0,276	0,983	0,017
-0,906	0,819	0,087	6,000	5,000	0,087	0,819	0,047	0,134	0,953	0,134	0,953	0,047
-0,819	0,906	-0,087	2,000	6,000	0,087	0,819	0,047	0,134	0,953	0,047	0,953	0,134
-0,707	0,966	-0,259	2,000	6,000	0,259	0,707	0,017	0,276	0,983	0,017	0,983	0,276
-0,574	0,996	-0,423	2,000	6,000	0,423	0,574	0,002	0,425	0,998	0,002	0,998	0,425
-0,423	0,996	-0,574	2,000	6,000	0,574	0,423	0,002	0,575	0,998	0,002	0,998	0,575
-0,259	0,966	-0,707	2,000	6,000	0,707	0,259	0,017	0,724	0,983	0,017	0,983	0,724
-0,087	0,906	-0,819	2,000	6,000	0,819	0,087	0,047	0,866	0,953	0,047	0,953	0,866
0,087	0,819	-0,906	3,000	1,000	0,819	0,087	0,047	0,866	0,953	0,047	0,866	0,953
0,259	0,707	-0,966	3,000	1,000	0,707	0,259	0,017	0,724	0,983	0,017	0,724	0,983
0,423	0,574	-0,996	3,000	1,000	0,574	0,423	0,002	0,575	0,998	0,002	0,575	0,998
0,574	0,423	-0,996	3,000	1,000	0,423	0,574	0,002	0,425	0,998	0,002	0,425	0,998
0,707	0,259	-0,966	3,000	1,000	0,259	0,707	0,017	0,276	0,983	0,017	0,276	0,983
0,819	0,087	-0,906	3,000	1,000	0,087	0,819	0,047	0,134	0,953	0,047	0,134	0,953
0,906	-0,087	-0,819	1,000	2,000	0,087	0,819	0,047	0,134	0,953	0,134	0,047	0,953
0,966	-0,259	-0,707	1,000	2,000	0,259	0,707	0,017	0,276	0,983	0,276	0,017	0,983
0,996	-0,423	-0,574	1,000	2,000	0,423	0,574	0,002	0,425	0,998	0,425	0,002	0,998
0,996	-0,574	-0,423	1,000	2,000	0,574	0,423	0,002	0,575	0,998	0,575	0,002	0,998
0,966	-0,707	-0,259	1,000	2,000	0,707	0,259	0,017	0,724	0,983	0,724	0,017	0,983
0,906	-0,819	-0,087	1,000	2,000	0,819	0,087	0,047	0,866	0,953	0,866	0,047	0,953
0,819	-0,906	0,087	5,000	3,000	0,819	0,087	0,047	0,866	0,953	0,953	0,047	0,866
0,707	-0,966	0,259	5,000	3,000	0,707	0,259	0,017	0,724	0,983	0,983	0,017	0,724
0,574	-0,996	0,423	5,000	3,000	0,574	0,423	0,002	0,575	0,998	0,998	0,002	0,575
0,423	-0,996	0,574	5,000	3,000	0,423	0,574	0,002	0,425	0,998	0,998	0,002	0,425
0,259	-0,966	0,707	5,000	3,000	0,259	0,707	0,017	0,276	0,983	0,983	0,017	0,276
0,087	-0,906	0,819	5,000	3,000	0,087	0,819	0,047	0,134	0,953	0,953	0,047	0,134
-0,087	-0,819	0,906	4,000	4,000	0,087	0,819	0,047	0,134	0,953	0,953	0,134	0,047
-0,259	-0,707	0,966	4,000	4,000	0,259	0,707	0,017	0,276	0,983	0,983	0,276	0,017
-0,423	-0,574	0,996	4,000	4,000	0,423	0,574	0,002	0,425	0,998	0,998	0,425	0,002
-0,574	-0,423	0,996	4,000	4,000	0,574	0,423	0,002	0,575	0,998	0,998	0,575	0,002
-0,707	-0,259	0,966	4,000	4,000	0,707	0,259	0,017	0,724	0,983	0,983	0,724	0,017

-0,819	-0,087	0,906	4,000	4,000	0,819	0,087	0,047	0,866	0,953	0,953	0,866	0,047
-0,906	0,087	0,819	6,000	5,000	0,819	0,087	0,047	0,866	0,953	0,866	0,953	0,047
-0,966	0,259	0,707	6,000	5,000	0,707	0,259	0,017	0,724	0,983	0,724	0,983	0,017
-0,996	0,423	0,574	6,000	5,000	0,574	0,423	0,002	0,575	0,998	0,575	0,998	0,002
-0,996	0,574	0,423	6,000	5,000	0,423	0,574	0,002	0,425	0,998	0,425	0,998	0,002
-0,966	0,707	0,259	6,000	5,000	0,259	0,707	0,017	0,276	0,983	0,276	0,983	0,017

3.4 Βιβλιογραφία

[1] Στέφανος Ν. Μανιάς, «Ηλεκτρονικά ισχύος», Αθήνα 2012

[2] «Space Vector Generator with Quadrature Control» available at Texas Instruments(http://www.ti.com.)

[3] Κωνσταντίνος Δ. Κορακίτης, «Εφαρμογή Μοντελοποιημένου Προβλεπτικού PQ Ελέγχου για τη Διασύνδεση Τριφασικού Αντιστροφέα στο Δίκτυο Ηλεκτρικής Ενέργειας», Διπλωματική εργασία, Μάρτιος 2011

[4] Tolle Sutinko, Auzani Jidin, Mohd Farriz Basar, «Simple Realization of 5-Segment Discontinuous SVPWM Based on FPGA», International Journal of Computer and Electrical Engineering, February 2010

ΚΕΦΑΛΑΙΟ 4

ΨΗΦΙΑΚΗ ΕΙΣΟΔΟΣ ΕΞΟΔΟΣ ΤΟΥ DSC TMS320F28335

4.1 Εισαγωγή

Η **ψηφιακή είσοδος/έξοδος (General Purpose Input/Output - GPIO)** είναι ένα από τα περιφερειακά του F28335, το οποίο επιτρέπει την επικοινωνία του DSP με εξωτερικές συσκευές. Λόγω της απλότητας χρήσης της, η μονάδα GPIO θα αποτελέσει το βασικό εργαλείο αποσφαλμάτωσης του κώδικα (debugging), αλλά και της παραγωγής ειδικών παλμοσειρών.

Στην αρχή του κεφαλαίου (τμήμα 4.2) παρουσιάζεται το **δομικό διάγραμμα του hardware** που συνοδεύει κάθε ακίδα. Σε αυτό το σημείο εξηγείται ο τρόπος με τον οποίον επιτυγχάνεται η πολύπλεξη πολλαπλών λειτουργιών (έως και τέσσερις) για κάθε ακίδα. Επιπλέον, επισημαίνεται μία ιδιαίτερη περίπτωση κατά την οποία η ψηφιακή έξοδος παρουσιάζει προβληματική συμπεριφορά και πως μπορεί να αποφευχθεί αυτή.

Στη συνέχεια, εξετάζεται η λειτουργία του **παραθύρου εγκυρότητας (qualification window)**. Αυτό αποτελεί, στην ουσία, ένα βαθυπερατό φίλτρο στην είσοδο κάθε GPIO ακίδας και επιτρέπει στον επεξεργαστή να αγνοήσει σπινθηρισμούς σε ψηφιακά σήματα εισόδου. Οι καταχωρητές που ενεργοποιούν και διαμορφώνουν το εύρος του παραθύρου εγκυρότητας παρουσιάζονται στο τμήμα 4.3.1. Στο τέλος του υποκεφαλαίου (τμήμα 4.3.3), βρίσκεται η επιβεβαίωση της ορθής λειτουργίας του παραθύρου εγκυρότητας μέσω μιας απλής πειραματικής διάταξης με παλμογράφο.

Στο τμήμα 4.4 παραθέτονται οι καταχωρητές βάσει των οποίων τα GPIO μπορούν να **αρχικοποιηθούν**, να **αναγνωστούν** και να **εγγραφούν**. Μετέπειτα (τμήμα 4.5) υπάρχουν πίνακες με τις **εναλλακτικές λειτουργίες των ακίδων**. Αυτοί οι πίνακες θα φανούν ιδιαίτερα χρήσιμοι σε επόμενα κεφάλαια, όταν και θα αξιοποιηθούν περισσότερα περιφερειακά του F28335. Τέλος, το κεφάλαιο κλείνει (τμήμα 4.6) με μία απλή **εφαρμογή** ενός **ψηφιακού χρονομετρητή**, ώστε ο αναγνώστης να συμφιλιωθεί με βασικές εντολές των GPIO.

4.2 Ανάλυση δομικού διαγράμματος των ακίδων GPIO [1]

Στην αναπτυξιακή πλακέτα του εργαστηρίου οι 88 ακίδες GPIO περιβάλλουν την κάρτα ελέγχου F28335. Χωρίζονται σε τρεις θύρες:

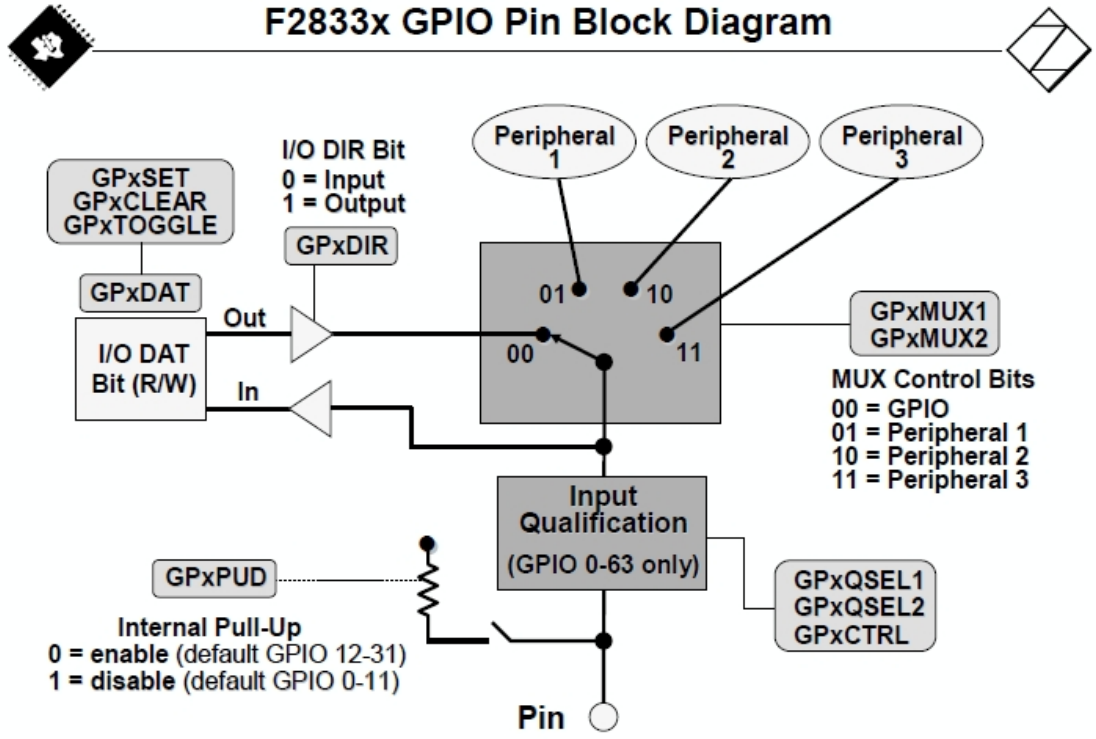
- Port A για τα GPIO0 έως 31
- Port B για τα GPIO32 έως 63
- Port C για τα GPIO64 έως 87

Όπως αναφέρθηκε και στο κεφάλαιο 2 κάθε GPIO pin της πλακέτας μπορεί να επιτελέσει έως και τέσσερις διαφορετικές λειτουργίες. Η επιλογή λειτουργίας ελέγχεται από την τιμή του πολυπλέκτη GPxMUX1/2 (όπου αντί για x τοποθετούμε το όνομα της θύρας και ο αριθμός 1 ή 2 εξαρτάται από το pin όπως φαίνεται στον Πίνακα 4.1).

Πίνακας 4.1: Αντιστοίχηση pins σε θύρες και πολυπλέκτες

Pin	Port	MUX
0-15	A	1
16-31	A	2
32-47	B	1
48-63	B	2
64-79	C	1
80-87	C	2

Οπότε, το pin 35 ανήκει στην θύρα B και η λειτουργία του ελέγχεται από τον GPBMUX1. Σε κάθε περίπτωση ο συνδυασμός 00 (βλ. Σχήμα 4.1) επιβάλει στο pin τη λειτουργία GPIO ενώ οι υπόλοιποι τρεις συνδέουν το pin με ένα από τα περιφερειακά του DSC (βλ. τμήμα 4.5).

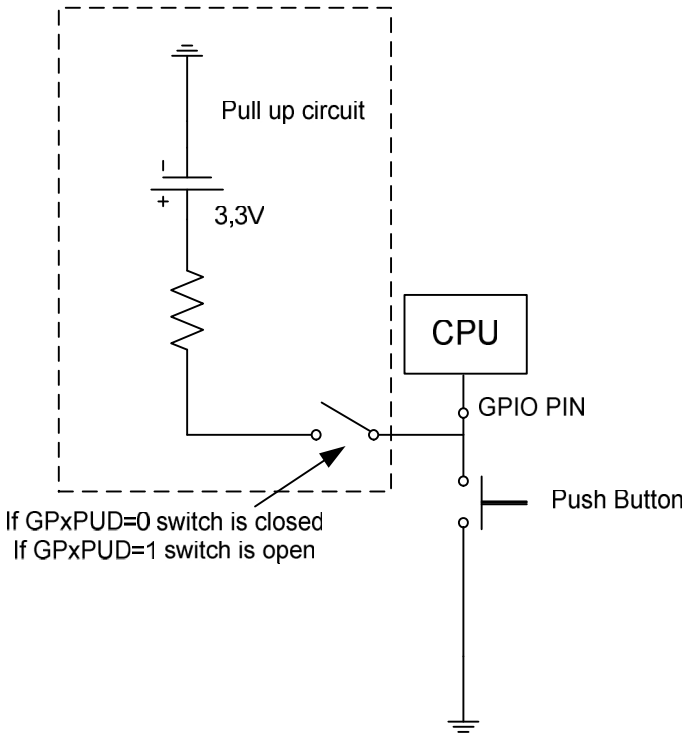


Σχήμα 4.1:Δομικό διάγραμμα των GPIO pin [1]

Οι καταχωρητές GPxDIR καθορίζουν εάν το pin θα αποτελεί είσοδο ή έξοδο. Για GPxDIR=0 το pin λειτουργεί ως είσοδος ενώ για GPxDIR=1 ως έξοδο.

Για ανάγνωση της τιμής του pin ελέγχουμε το πεδίο GPxDAT. Για εγγραφή στο pin έχουμε διαθέσιμες τις εντολές GPxSET, GPxCLEAR και GPxTOGGLE, που θέτουν (τιμή 1), καθαρίζουν (τιμή 0) η αντιστρέφουν την τιμή του pin αντίστοιχα.

Στο κάτω αριστερά μέρος του σχήματος 4.1 υπάρχει ο καταχωρητής GPxPUD. Εάν η τιμή του είναι 0 ο διακόπτης κλείνει και το pin εξαναγκάζεται σε υψηλό δυναμικό (pull-up) σε αντίθετη περίπτωση το pin είναι αιωρούμενο. Το συγκεκριμένο ζήτημα χρειάζεται ιδιαίτερη προσοχή όταν χειριζόμαστε pin εισόδου. Εργαστηριακά παρατηρήθηκε ότι η ενδεδειγμένη πορεία είναι **πάντοτε να ενεργοποιούμε την επιλογή Pullup (GPxPUD=0) για τη λειτουργία εισόδου**. Αυτό συμβαίνει, γιατί εάν σε κάποιες περιπτώσεις δεν παρέχει το εξωτερικό κύκλωμα το επίπεδο τάσης του pin, η τιμή που διαβάζει ο επεξεργαστής είναι τυχαία και μεταβάλλεται με το χρόνο. Αυτή η περίπτωση μπορεί να συμβεί όταν στην είσοδο του pin έχει συνδεθεί ένα push button (Σχήμα 4.2).



Σχήμα 4.2: Διάταξη κυκλώματος εισόδου η οποία μπορεί να παρουσιάσει προβληματική συμπεριφορά

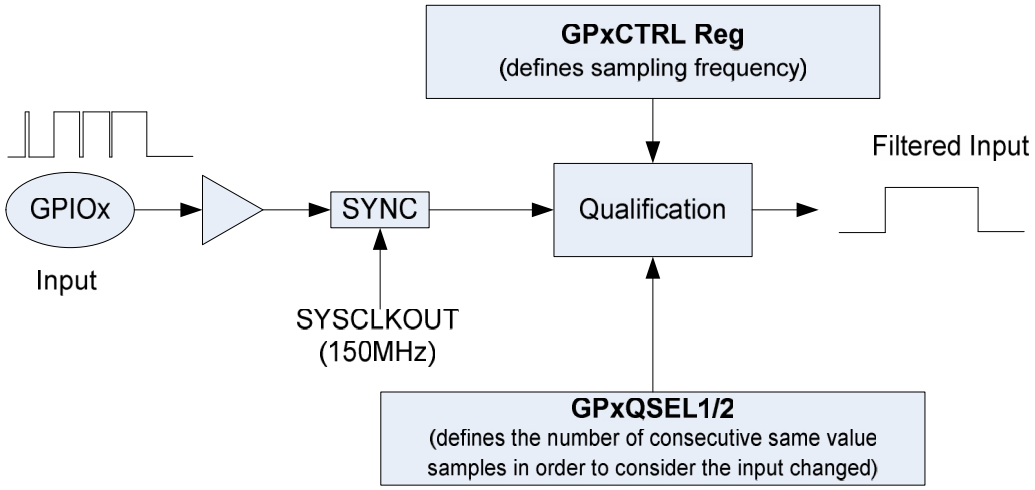
Η τελευταία παράμετρος του σχήματος 4.1 που μένει να αναλυθεί είναι το Παράθυρο Εγκυρότητας (Qualification Window). Τα σήματα συχνά περιέχουν συνιστώσες θορύβου. Χάρη στο Παράθυρο Εγκυρότητας μπορούμε να επιλέξουμε να αγνοούμε εναλλαγές στο σήμα οι οποίες διαρκούν μικρό χρονικό διάστημα. Το εύρος,

κάτω από το οποίο τα σήματα αγνοούνται, μπορεί να μεταβληθεί. Στη συνέχεια του κεφαλαίου το ζήτημα αυτό θα αναλυθεί περαιτέρω.

4.3 Παράθυρο Εγκυρότητας (Qualification Window) [1]

Ένα σήμα που φτάνει στην είσοδο ενός GPIO pin μπορεί να έχει παρεμβολές, σπινθηρισμούς και πολλά άλλα παράσιτα, τα οποία θέλουμε ο επεξεργαστής να αγνοήσει. Για αυτό το λόγο μας παρέχεται ένα φίλτρο με τη δυνατότητα να μεταβάλουμε τη συχνότητα αποκοπής του ανάλογα με τις προδιαγραφές της εφαρμογής.

Το φίλτρο αυτό ονομάζεται **παράθυρο εγκυρότητας (qualification window)** και είναι, ουσιαστικά, ένα χρονικό κατώφλι. Ο επεξεργαστής αγνοεί κάθε μεταβολή με χρονική διάρκεια μικρότερη από αυτό. Το **εύρος του "παραθύρου"** καθορίζεται από δύο μεγέθη. Το πρώτο είναι η **συχνότητα των δειγματοληψίας** και το δεύτερο είναι ο **αριθμός των δειγμάτων ανά παράθυρο**. Το σχήμα 4.3 αναπαριστά σε δομικό διάγραμμα τη λειτουργία του qualification window.



Σχήμα 4.3: Διάγραμμα λειτουργίας του Qualification Window

Το σήμα εισέρχεται στην ακίδα GPIOx. Στη συνέχεια δειγματοληπτείται με τη συχνότητα του εσωτερικού ρολογιού του DSP (SYSCLKOUT), η οποία είναι 150MHz. Ο καταχωρητής GPxCTRL καθορίζει την τελική συχνότητα δειγματοληψίας της εισόδου, η οποία είναι πάντα υποπολλαπλάσιο της συχνότητας SYSCLKOUT.

Για να καθοριστεί το παράθυρο δειγματοληψίας απαιτείται και ο καθορισμός του καταχωρητή GPxQSEL1/2. Αυτός ο καταχωρητής εκφράζει τον αριθμό των

δειγμάτων, τα οποία πρέπει να είναι ίδια, ώστε να θεωρηθεί η αλλαγή λογικού επιπέδου έγκυρη και να μην αγνοηθεί ως θόρυβος.

Το χρονικό εύρος ($\mathbf{T_{QW}}$) του παραθύρου καθορίζεται από τον τύπο (4.1):

$$T_{QW} = T_S(\text{samples} - 1) \tag{4.1}$$

όπου $\mathbf{T_S}$ είναι η περίοδος δειγματοληψίας και **samples** είναι ο αριθμός δειγμάτων. Στη συνέχεια του κεφαλαίου εξηγείται πως μπορεί να γίνει η διαμόρφωση του T_{QW}.

Τέλος, πρέπει να σημειωθεί ότι το φίλτρο αυτό καθυστερεί την είσοδο κατά χρονικό διάστημα ίσο με το εύρος του παραθύρου εγκυρότητας (T_{QW}). Αν και αυτή η καθυστέρηση μπορεί να είναι πολύ μικρή, σε κάποιες εφαρμογές πραγματικού χρόνου είναι κρίσιμη και για αυτό το λόγο υπάρχει τρόπος να απενεργοποιηθεί.

4.3.1 Καταχωρητές που ενεργοποιούν και διαμορφώνουν το εύρος του παραθύρου εγκυρότητας

- **Καταχωρητής ελέγχου της συχνότητας δειγματοληψίας**

Ο καταχωρητής ***GpioCtrlRegs.GPxCTRL, όπου x=A ή B*** ελέγχει το χρονικό διάστημα που μεσολαβεί ανάμεσα στα δείγματα του παράθυρου εγκυρότητας. Χωρίζεται σε τέσσερις αριθμούς των 8 bit και κάθε τέτοιος αριθμός ελέγχει μία ομάδα GPIO εισόδων, όπως φαίνεται στους παρακάτω πίνακες:

GPIO Port x Qualification Control (GPx CTRL) Register

31	24	23	16
QUALPRD3		QUALPRD2	
R/W-0		R/W-0	

15	8	7	0
QUALPRD1		QUALPRD0	
R/W-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset x= A or B

Σχήμα 4.4: Καταχωρητής ελέγχου της συχνότητας δειγματοληψίας[1]

Πίνακας 4.2: Έλεγχος της συχνότητας δειγματοληψίας για τα pin της θύρας A

GPIO Port A Qualification Control (GPACTRL) Register Field Description			
Bits	**Πεδίο**	**Τιμή**	**GPIO**
31 έως 24	QUALPRD3	0x00 έως 0xFF	GPIO24-31
23 έως 16	QUALPRD2	0x00 έως 0xFF	GPIO16-23
15 έως 8	QUALPRD1	0x00 έως 0xFF	GPIO8-15
7 έως 0	QUALPRD0	0x00 έως 0xFF	GPIO0-7

Σημ.: Ο καταχωρητής αυτός είναι προστατευμένος. Απαιτείται η EALLOW για να γράψουμε σε αυτόν.

Ο τύπος που καθορίζει τη συχνότητα (f_s) και την περίοδο (T_s) δειγματοληψίας είναι:

$$T_s = 2 \cdot T_{SYSCLKOUT} \cdot GPxCTRL[QUALPRDy] \tag{4.2}$$

$$f_s = \frac{1}{T_s} \tag{4.3}$$

,όπου $\mathbf{T_{SYSCLKOUT}}$ είναι η περίοδος του ρολογιού του επεξεγαστή και για το συγκεκριμένο επεξεργαστή είναι **6,67ns** (η συχνότητά του είναι 150MHz).

Για πληρότητα ακολουθεί και ο πίνακας για τη θύρα B. Τα GPIO pins της θύρας C δεν έχουν δυνατότητα φιλτραρίσματος του σήματος μέσω Qualification Window.

Πίνακας 4.3: Έλεγχος της συχνότητας δειγματοληψίας για τα pin της θύρας B

GPIO Port B Qualification Control (GPBCTRL) Register Field Description			
Bits	**Πεδίο**	**Τιμή**	**GPIO**
31 έως 24	QUALPRD3	0x00 έως 0xFF	GPIO56-73
23 έως 16	QUALPRD2	0x00 έως 0xFF	GPIO48-55
15 έως 8	QUALPRD1	0x00 έως 0xFF	GPIO40-47
7 έως 0	QUALPRD0	0x00 έως 0xFF	GPIO32-29

Οπότε αν δώσουμε τις εντολές:

```
EALLOW;
GpioCtrlRegs.GPACTRL.bit.QUALPRD0 = 0x32;
EDIS;
```

σημαίνει ότι για τα **GPIO bits 7-0** η δειγματοληψία της εισόδου θα έχει περίοδο και συχνότητα:

$$T_s = 2 \cdot T_{SYSCLKOUT} \cdot GPACTRL[QUALPRD0] = 2 \cdot 6,67 \cdot 50 = 667ns \quad (4.4)$$

$$f_s = \frac{1}{T_s} = \frac{1}{667} = 1,4992MHz \quad (4.5)$$

- **Καταχωρητής Μεθόδου Δειγματοληψίας**

Ο καταχωρητής αυτός ελέγχει πόσα δείγματα θα λαμβάνονται ώστε να θεωρηθεί μία είσοδος έγκυρη. Παράλληλα, λειτουργεί και ως ενεργοποιητής, καθώς μία από τις τέσσερις καταστάσεις απενεργοποιεί το παράθυρο εγκυρότητας. Στον πίνακα 4.4 παρουσιάζονται οι καταστάσεις αυτές.

GPIO Port A Qualification Select 1 (GPAQSEL1) Register

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16
GPIO15	GPIO14	GPIO13	GPIO12	GPIO11	GPIO10	GPIO9	GPIO8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
GPIO7	GPIO6	GPIO5	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Εικόνα 4.5: Καταχωρητής μεθόδου δειγματοληψίας[1]

Πίνακας 4.4: Επεξήγηση του τρόπου λειτουργίας του καταχωρητή μεθόδου δειγματοληψίας

Bits	Πεδίο	Τιμή	Περιγραφή	Αριθμός δειγμάτων
31-0 (τα εξετάζουμε ανά 2)	GPIO15 (bits 31-30) έως GPIO0 (bits 1-0)	00	Συγχρονισμένο στο SYSCLKOUT. Η είσοδος σαρώνεται και αλλάζει την τιμή της ανά 6,67ns (το μικρότερο δυνατό παράθυρο εγκυρότητας)	-
		01	Περιμένει 3 δείγματα με την ίδια τιμή για να θεωρήσει έγκυρη μία είσοδο. Η απόσταση μεταξύ των δειγμάτων καθορίζεται από τον GPACTRL καταχωρητή (αναφέρεται στην προηγούμενη σελίδα)	3
		10	Όμοια με την περίπτωση 01 αλλά για 6 δείγματα.	6
		11	Ασύγχρονη λειτουργία, κατάργηση	-

			του παραθύρου εγκυρότητας. Χρησιμοποιείται σε περιπτώσεις που απαιτείται ακαριαία δράση (πχ. ανίχνευση σφαλμάτων)	

Όπως είναι αναμενόμενο υπάρχουν άλλοι 3 τέτοιοι καταχωρητές που ελέγχουν τα εναπομείναντα GPIO pins. Οπότε συνολικά οι GPxQSELy καταχωρητές είναι:

Πίνακας 4.5: Καταχωρητής επιλογής δειγμάτων για όλα τα GPIO pins στα οποία είναι διαθέσιμος

GPIO Port Qualification Select Register and their respective pins	
'Ονομα καταχωρητή	**GPIO**
GpioCtrlRegs.GPAQSEL1	GPIO 15-0
GpioCtrlRegs.GPAQSEL2	GPIO 31-16
GpioCtrlRegs.GPBQSEL1	GPIO 47-32
GpioCtrlRegs.GPBQSEL2	GPIO 63-48

- **Εφαρμογή για τη διαμόρφωση μέγιστου παράθυρου εγκυρότητας**

Ποιο είναι το μέγιστο qualification window (παράθυρο εγκυρότητας) για το GPIO17? Ποιες εντολές το εξασφαλίζουν?

Απάντηση:

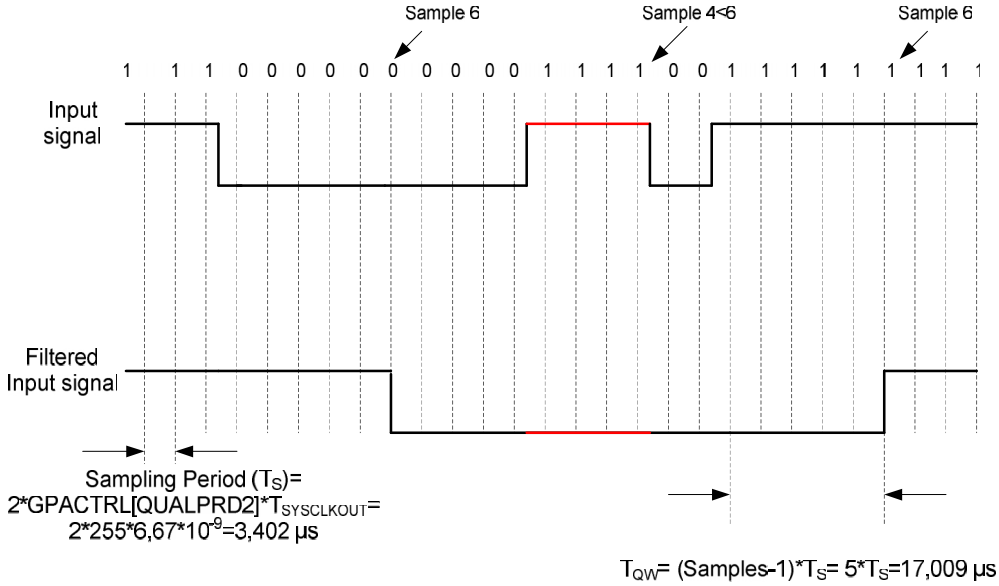
Το GPIO17 είναι pin της θύρας A (QUALPRD2) και ανήκει στην ομάδα του Qualification Select Register 2. Το μέγιστο παράθυρο το εξασφαλίζει η ρύθμιση για **6 δείγματα** και για **0xFF** συχνότητα δειγματοληψίας. Οπότε οι εντολές που πρέπει να γράψουμε είναι οι εξής:

```
GpioCtrlRegs.GPAQSEL2.bit.GPIO17 = 2; //6 sample qualification window
for GPIO17
GpioCtrlRegs.GPACTRL.bit.QUALPRD2 = 0xFF; // sampling 510*Tsysclk
GPIO16-23
```

Το εύρος του παραθύρου εγκυρότητας υπολογίζεται με τη βοήθεια των τύπων (4.1) και (4.2):

$$T_{QW} = T_S(\text{samples}-1) = 2 \cdot T_{SYSCLKOUT} \cdot GPACTRL[QUALPRD2](\text{samples}-1) =$$
$$= 2 \cdot 6,67 \cdot 10^{-9} \cdot 255 \cdot (6-1) = 17,0085 \mu s \tag{4.6}$$

Παρακάτω φαίνεται το διάγραμμα χρονισμού μεταξύ της πραγματικής τιμής του σήματος (αυθεντικό σήμα που έρχεται σαν είσοδος) και της τιμής του, μετά τον qualifier, για τις παραμέτρους που έχουν τεθεί στη συγκεκριμένη εφαρμογή:



Σχήμα 4.6: Παράδειγμα λειτουργίας του Qualification Window

Το πραγματικό σήμα εισόδου έχει έναν σπινθηρισμό που διαρκεί περίπου 4*T_S (τονισμένο με κόκκινο χρώμα). Όπως είναι αναμενόμενο αυτή η μικρή χρονικά μεταβολή "δεν περνάει" από τον Qualifier και ως εκ τούτου αγνοείται. Εξαρτάται από την εφαρμογή το αν θέλουμε ή όχι να δούμε αυτή τη μεταβολή.

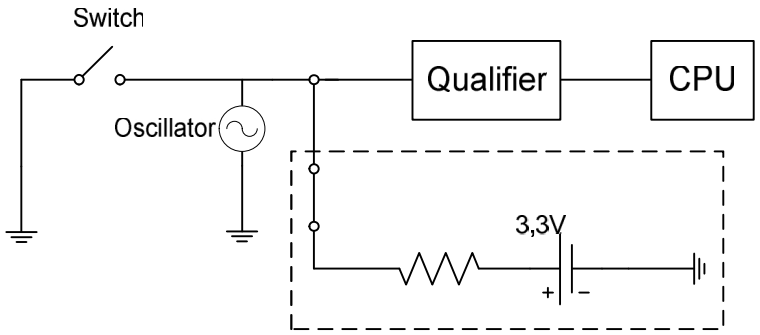
Παρατηρούμε, ότι το παράθυρο εγκυρότητας προκαλεί καθυστέρηση στο σήμα εισόδου κατά χρονικό διάστημα περίπου ίσο με T_{QW}. Η χρησιμοποίησης της λέξης "περίπου" οφείλεται στο γεγονός ότι το σήμα δεν είναι απαραίτητα συγχρονισμένο με τη δειγματοληψία του Qualifier. Οπότε, η συνολική καθυστέρηση που εισάγει το παράθυρο εγκυρότητας (T_{delay}) κυμαίνεται στην περιοχή:

$$T_{QW} \le T_{delay} \le T_{QW} + T_s \tag{4.7}$$

4.3.2 Έλεγχος παραθύρου εγκυρότητας με παλμογράφο

Τη σημασία του Qualification window σε σχέση με το θόρυβο μπορούμε να την καταλάβουμε καλύτερα μέσω ενός απλού πειράματος. Η διάταξη του πειράματος

φαίνεται στο σχήμα 4.7. Αποτελείται από έναν διακόπτη ο οποίος στο ένα του άκρο είναι γειωμένος, και στο άλλο είναι ενωμένος με την ακίδα GPIO 17 και με το probe ενός παλμογράφου. Το GPIO17 έχει δηλωθεί σαν είσοδος και έχει τεθεί σε υψηλό δυναμικό (χρησιμοποιείται το pull-up κύκλωμα). Με τη χρήση του διακόπτη μπορούμε να θέσουμε το pin σε χαμηλό δυναμικό (κλειστός διακόπτης) ή σε υψηλό (ανοικτός διακόπτης).

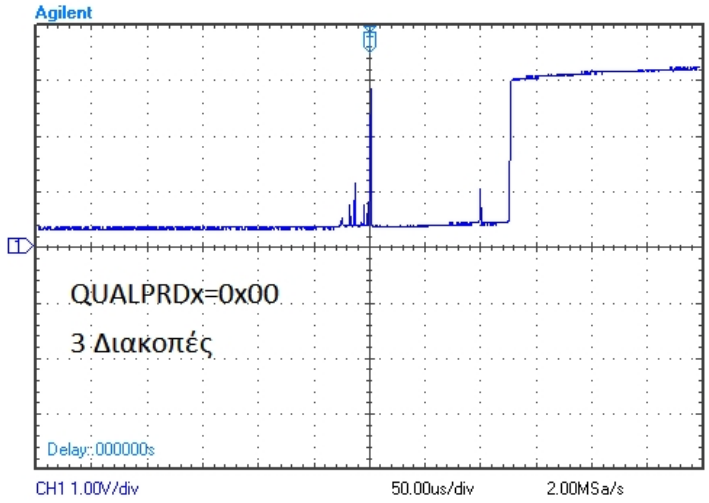


Σχήμα 4.7: Πειραματική διάταξη για τον έλεγχο του παράθυρου εγκυρότητας

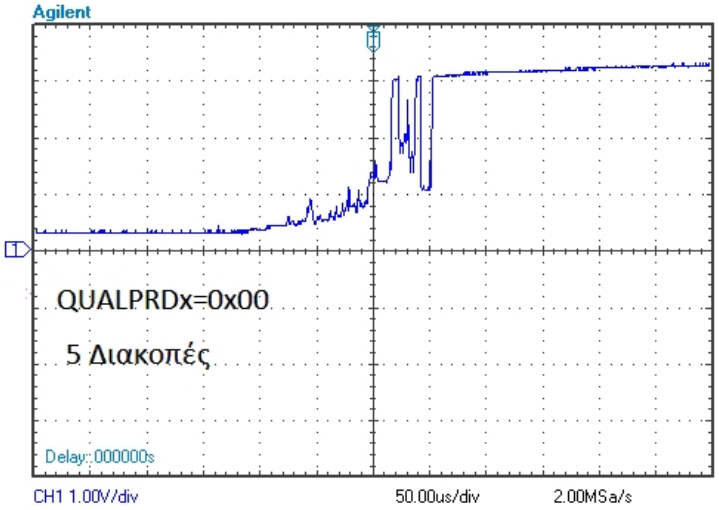
Το πρόγραμμα που τρέχει στον DSP μετράει τις συνολικές μεταβάσεις από χαμηλό σε υψηλό δυναμικό (0 σε 3,3V). Αυτό επιτυγχάνεται με τη τεχνική των διακοπών. Κάθε φορά που ανιχνεύεται θετική ακμή στο GPIO17, το πρόγραμμα διακόπτει την κανονική ροή εντολών και εκτελεί μία ειδική διαδικασία, τη ρουτίνα εξυπηρέτησης διακοπής (ΡΕΔ). Στη συγκεκριμένη εφαρμογή, αυτή η ρουτίνα αυξάνει κατά "1" την τιμή μιας μεταβλητής. Λεπτομερής παρουσίαση του συστήματος διακοπών του F28335 γίνεται στο επόμενο κεφάλαιο.

Εάν ο διακόπτης ήταν ιδανικός, θα μετρούσαμε μία διακοπή ανά χειρισμό (άνοιγμα του διακόπτη). Ωστόσο, κατά τον χειρισμό δημιουργούνται αρκετοί σπινθηρισμοί οι οποίοι ανιχνεύονται από το πρόγραμμα ανάλογα με το εύρος του παράθυρου εγκυρότητας. **Όσο μικρότερο είναι το παράθυρο τόσο μεγαλύτερη είναι η ευαισθησία του προγράμματος σε σπινθηρισμούς**.

Στα σχήματα 4.8 και 4.9 παρουσιάζονται δύο χειρισμοί του διακόπτη. Το εύρος του παραθύρου εγκυρότητας είναι το μικρότερο δυνατό και όπως είναι φυσικό, ο παραμικρός σπινθηρισμός καταγράφεται από το πρόγραμμα.

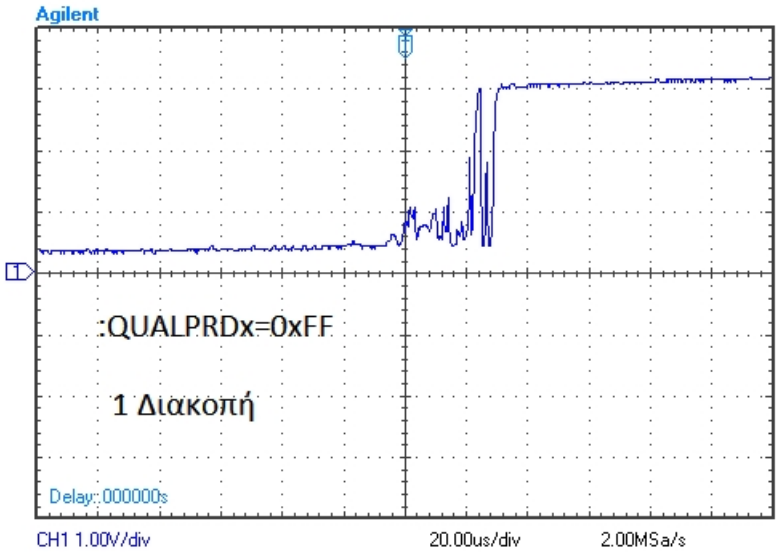


Σχήμα 4.8: Για το μικρότερο δυνατό παράθυρο εγκυρότητας το πρόγραμμα ανίχνευσε 3 ακμές



Σχήμα 4.9 : Για το μικρότερο δυνατό παράθυρο εγκυρότητας το πρόγραμμα ανίχνευσε 5 ακμές

Αντίθετα, όταν χρησιμοποιούμε το μέγιστη δυνατή τιμή για το παράθυρο εγκυρότητας, εξίσου θορυβώδη σήματα αγνοούνται από το πρόγραμμα και εκτελείται μία διακοπή (αντί για πολλαπλές).



Σχήμα 4.10: Για το μεγαλύτερο δυνατό παράθυρο εγκυρότητας το πρόγραμμα ανίχνευσε 1 ακμή, παρά τον έντονο θόρυβο που υπάρχει στο σήμα

4.4 Καταχωρητές GPIO [1]

Στους πίνακες 4.6 και 4.7 υπάρχουν όλοι οι καταχωρητές οι οποίοι χρησιμοποιούνται για τη λειτουργία των GPIO:

Πίνακας 4.6 :Καταχωρητές ελέγχου GPIO (GpioCtrlRegs)

Καταχωρητής	Περιγραφή
GPACTRL	GPIO A control Register (GPIO0-31)
GPAQSEL1	GPIO A Qualifier Select 1 Register (GPIO0-15)
GPAQSEL2	GPIO A Qualifier Select 2 Register (GPIO16-31)
GPAMUX1	GPIO A MUX1 Register (GPIO0-15)
GPAMUX2	GPIO A MUX2 Register (GPIO16-31)
GPADIR	GPIO A Direction Register (GPIO0-31)
GPAPUD	GPIO A Pull Up Disable Register (GPIO0-31)
GPBCTRL	GPIO B control Register (GPIO32-63)
GPBQSEL1	GPIO B Qualifier Select 1 Register (GPIO32-47)
GPBQSEL2	GPIO B Qualifier Select 2 Register (GPIO48-63)
GPBMUX1	GPIO B MUX1 Register (GPIO32-47)
GPBMUX2	GPIO B MUX2 Register (GPIO48-63)
GPBDIR	GPIO B Direction Register (GPIO32-63)

GPBPUD	GPIO B Pull Up Disable Register (GPIO32-63)
GPCMUX1	GPIO C MUX1 Register (GPIO64-79)
GPCMUX1	GPIO C MUX2 Register (GPIO80-87)
GPCDIR	GPIO C Direction Register (GPIO64-87)
GPCPUD	GPIO C Pull Up Disable Register (GPIO64-87)

Πίνακας 4.7: Καταχωρητές δεδομένων GPIO (GpioDataRegs)

Καταχωρητής	Περιγραφή
GPADAT	GPIO A Data Register GPIO(0-31)
GPASET	GPIO A Data Set Register GPIO(0-31)
GPACLEAR	GPIO A Data Clear Register GPIO(0-31)
GPATOGGLE	GPIO A Data Toggle Register GPIO(0-31)
GPBDAT	GPIO B Data Register GPIO(32-63)
GPBSET	GPIO B Data Set Register GPIO(32-63)
GPBCLEAR	GPIO B Data Clear Register GPIO(32-63)
GPBTOGGLE	GPIO B Data Toggle Register GPIO(32-63)
GPCDAT	GPIO C Data Register GPIO(64-87)
GPCSET	GPIO C Data Set Register GPIO(64-87)
GPCCLEAR	GPIO C Data Clear Register GPIO(64-87)
GPCTOGGLE	GPIO C Data Toggle Register GPIO(64-87)

Να σημειωθεί ότι οι καταχωρητές του πίνακα 4.6 είναι προστατευμένοι. Αυτό σημαίνει ότι για να μεταβληθούν οι τιμές τους χρειάζεται η εντολή **EALLOW**. Όταν τελειώσει η αρχικοποίηση των GPIO για να αποτραπεί περαιτέρω πρόσβαση δίνεται η εντολή **EDIS**. Παραδείγματα με επιμέρους εντολές που αξιοποιούν αυτούς τους καταχωρητές δίνονται στην εφαρμογή στο τέλος του κεφαλαίου.

4.5 Εναλλακτικές λειτουργίες των ακίδων GPIO [1]

Όπως αναφέρεται και στην αρχή του κεφαλαίου τα 88 GPIO pin μπορούν με κατάλληλες τιμές στους πολυπλέκτες (GPxMUX1/2) του σχήματος 4.1 να επιτελέσουν διαφορετικές λειτουργίες. Οι παρακάτω πίνακες παρουσιάζουν όλες τις δυνατές περιπτώσεις.

Οι σημαντικότερες από αυτές τις λειτουργίες είναι:

- EPWMxx : Αυτά τα pin χρησιμοποιούνται για την παραγωγή PWM παλμών.
- ECAPx: Περιφερειακό που χρησιμοποιείται για την ανάλυση ψηφιακών παλμοσειρών.
- ADCSOCxx: Pin που χρησιμοποιούνται για τη μετατροπή αναλογικών τιμών σε ψηφιακές.

- CAN/SCI/SPI: Pin τα οποία μπορούν να χρησιμοποιηθούν για την επικοινωνία του DSC με εξωτερικές συσκευές, μέσω των αντιστοίχων πρωτοκόλλων επικοινωνίας (βλ. κεφάλαιο 1)

Πίνακας 4.8: Εναλλακτικές λειτουργίες των Pin 0-15 [1]

GPIO - A Multiplex Register GPAMUX1

GPAMUX1 - Bits	00	01	10	11
1,0	GPIO0	EPWM1A	-	-
3,2	GPIO1	EPWM1B	ECAP6	MFSRB
5,4	GPIO2	EPWM2A	-	-
7,6	GPIO3	EPWM2B	ECAP5	MCLKRB
9,8	GPIO4	EPWM3A	-	-
11,10	GPIO5	EPWM3B	MFSRA	ECAP1
13,12	GPIO6	EPWM4A	EPWMSYNCI	EPWMSYNC0
15,14	GPIO7	EPWM4B	MCLKRA	ECAP2
17,16	GPIO8	EPWM5A	CANTXB	/ADCSOCA0
19,18	GPIO9	EPWM5B	SCITXDB	ECAP3
21,20	GPIO10	EPWM6A	CANRXB	/ADCSOCB0
23,22	GPIO11	EPWM6B	SCIRXDB	ECAP4
25,24	GPIO12	/TZ1	CANTXB	SPISIMOB
27,26	GPIO13	/TZ2	CANRXB	SPISOMIB
29,28	GPIO14	/TZ3_/XHOLD	SCITXDB	SPICLKB
31,30	GPIO15	/TZ4_/XHOLDA	SCIRXDB	/SPISTEB

Πίνακας 4.9: Εναλλακτικές λειτουργίες των Pin 15-31 [1]

GPIO - A Multiplex Register GPAMUX2

GPAMUX2 - Bits	00	01	10	11
1,0	GPIO16	SPISIMOA	CANTXB	/TZ5
3,2	GPIO17	SPISOMIA	CANRXB	/TZ6
5,4	GPIO18	SPICLKA	SCITXDB	CANRXA
7,6	GPIO19	/SPISTEA	SCIRXDB	CANTXA
9,8	GPIO20	EQEP1A	MDXA	CANTXB
11,10	GPIO21	EQEP1B	MDRA	CANRXB
13,12	GPIO22	EQEP1S	MCLKXA	SCITXDB
15,14	GPIO23	EQEP1I	MFSXA	SCIRXDB
17,16	GPIO24	ECAP1	EQEP2A	MDXB
19,18	GPIO25	ECAP2	EQEP2B	MDRB
21,20	GPIO26	ECAP3	EQEP2I	MCLKXB
23,22	GPIO27	ECAP4	EQEP2S	MFSXB
25,24	GPIO28	SCIRXDA	/XZCS6	/XZCS6
27,26	GPIO29	SCITXDA	XA19	XA19
29,28	GPIO30	CANRXA	XA18	XA18
31,30	GPIO31	CANTXA	XA17	XA17

Πίνακας 4.10: Εναλλακτικές λειτουργίες των Pin 32-47 [1]

GPIO - B Multiplex Register GPBMUX1

GPBMUX1 - Bits	00	01	10	11
1,0	GPIO32	SDAA	EPWMSYNCI	/ADCSOCA0
3,2	GPIO33	SCLA	EPWMSYNCO	/ADCSOCB0
5,4	GPIO34	ECAP1	XREADY	XREADY
7,6	GPIO35	SCITXDA	XR/W	XR/W
9,8	GPIO36	SCIRXDA	/XZCS0	/XZCS0
11,10	GPIO37	ECAP2	/XZCS7	/XZCS7
13,12	GPIO38	-	/XWE0	/XWE0
15,14	GPIO39	-	XA16	XA16
17,16	GPIO40	-	XA0/XWE1	XA0/XWE1
19,18	GPIO41	-	XA1	XA1
21,20	GPIO42	-	XA2	XA2
23,22	GPIO43	-	XA3	XA3
25,24	GPIO44	-	XA4	XA4
27,26	GPIO45	-	XA5	XA6
29,28	GPIO46	-	XA6	XA6
31,30	GPIO47	-	XA7	XA7

Πίνακας 4.11: Εναλλακτικές λειτουργίες των Pin 48-63 [1]

GPIO - B Multiplex Register GPBMUX2

GPBMUX2 - Bits	00	01	10	11
1,0	GPIO48	ECAP5	XD31	XD31
3,2	GPIO49	ECAP6	XD30	XD30
5,4	GPIO50	EQEP1A	XD29	XD29
7,6	GPIO51	EQEP1B	XD28	XD28
9,8	GPIO52	EQEP1S	XD27	XD27
11,10	GPIO53	EQEP1I	XD26	XD26
13,12	GPIO54	SPISIMOA	XD25	XD25
15,14	GPIO55	SPISOMIA	XD24	XD24
17,16	GPIO56	SPICLKA	XD23	XD23
19,18	GPIO57	/SPISTEA	XD22	XD22
21,20	GPIO58	MCLKRA	XD21	XD21
23,22	GPIO59	MFSRA	XD20	XD20
25,24	GPIO60	MCLKRB	XD19	XD19
27,26	GPIO61	MFSRB	XD18	XD18
29,28	GPIO62	SCIRXDC	XD17	XD17
31,30	GPIO63	SCITXDC	XD16	XD16

Πίνακας 4.12: Εναλλακτικές λειτουργίες των Pin 64-87 [1]

GPIO - C Multiplex Register

GPCMUX1 - Bits	00 or 01	10 or 11
1,0	GPIO64	XD15
3,2	GPIO65	XD14
5,4	GPIO66	XD13
7,6	GPIO67	XD12
9,8	GPIO68	XD11
11,10	GPIO69	XD10
13,12	GPIO70	XD9
15,14	GPIO71	XD8
17,16	GPIO72	XD7
19,18	GPIO73	XD6
21,20	GPIO74	XD5
23,22	GPIO75	XD4
25,24	GPIO76	XD3
27,26	GPIO77	XD2
29,28	GPIO78	XD1
31,30	GPIO79	XD0

GPCMUX2 - Bits	00 or 01	10 or 11
1,0	GPIO80	XA8
3,2	GPIO81	XA9
5,4	GPIO82	XA10
7,6	GPIO83	XA11
9,8	GPIO84	XA12
11,10	GPIO85	XA13
13,12	GPIO86	XA14
15,14	GPIO87	XA15
17,16	-	-
19,18	-	-
21,20	-	-
23,22	-	-
25,24	-	-
27,26	-	-
29,28	-	-
31,30	-	-

4.6 Εφαρμογή ψηφιακού χρονομετρητή

Σε αυτή την εφαρμογή θα δημιουργήσουμε ένα χρονόμετρο με την εξής λειτουργία:

- Τα GPIO9/11/34/49 αποτελούν το χρονομέτρης με εύρος από $(0000)_2$ έως $(1111)_2$. Το Least Significant Bit (LSB) είναι το GPIO9 ενώ το Most Significant Bit(MSB) είναι το GPIO49.
- Τα χρονικό διάστημα για την αύξηση του χρονομέτρου κατά 1 είναι 100ms.
- Το GPIO 48 λειτουργεί σαν START.
- Το GPIO 17 λειτουργεί σαν STOP.

Σχήμα 4.11: Παρουσίαση στιγμιοτύπου λειτουργίας του χρονομετρητή και αντιστοίχιση GPIO pins με τις λειτουργίες του.

Τα μπουτόν και τα led που φαίνονται στο σχήμα 4.11 δε συμπεριλαμβάνονται στην αναπτυξιακή πλακέτα. Θα ήταν αρκετά βοηθητικό (εάν ασχοληθείτε σε βάθος με το συγκεκριμένο DSC) να κατασκευάσετε μία πλακέτα με κάποιους διακόπτες,

μπουτόν και led. Μπορεί να μην είναι απευθείας χρήσιμο για τις εφαρμογές, αλλά μπορούν να βοηθήσουν αρκετά στην κατασκευή κώδικα. Η εναλλακτική, είναι η αποσφαλμάτωση πραγματικού χρόνου (real time debugging), η οποία περιγράφεται στο παράρτημα. Είναι εξίσου χρήσιμη, αλλά πιο πολύπλοκη.

Ας εξετάσουμε ξεχωριστά τα διαφορετικά κομμάτια του κώδικα που απαιτούνται:

- Αρχικά πρέπει επιλέξουμε τη λειτουργία GPIO για τα pin που θα χρησιμοποιήσουμε και να τα δηλώσουμε σαν είσοδο ή έξοδο. Θα δείξουμε τη διαδικασία για τα pin 48(port B, mux 2) και 9 (port A, mux 1):

```
EALLOW;
GpioCtrlRegs.GPAMUX1.bit.GPIO9 = 0;// GPIO9 = General Purpose I/O
GpioCtrlRegs.GPADIR.bit.GPIO9 = 1; // GPIO9= output
GpioCtrlRegs.GPBMUX2.bit.GPIO48 = 0;// GPIO48 = General Purpose I/O
GpioCtrlRegs.GPBDIR.bit.GPIO48 = 1; // GPIO48= input
GpioCtrlRegs.GPBPUD.bit.GPIO48 = 0; // enable internal Pull-Up
EDIS;
```

Στον κώδικά δεν κάνουμε ξεχωριστή δήλωση για το pin 48, καθώς στην διαδικασία **`Gpio_select()`** αρχικοποιούμε όλα τα pins σαν εισόδους GPIO. Επίσης αυτό το pin έχει ενεργοποιημένο το εσωτερικό pull up κύκλωμα, ωστόσο για να είμαστε ασφαλείς μπορούμε να συμπεριλάβουμε και την εντολή που το ενεργοποιεί.

- Οι παρακάτω ορισμοί βοηθούν ώστε να είναι ευανάγνωστο το πρόγραμμα

```
#define STOP      GpioDataRegs.GPADAT.bit.GPIO17
#define START     GpioDataRegs.GPBDAT.bit.GPIO48
```

- Η καθυστέρηση των 100ms υλοποιείται με τη συνάρτηση delay_loop(1000000) η οποία τρέχει ένα απλό κομμάτι κώδικα (artificial latency). Αυτό που παρουσιάζει ενδιαφέρον είναι ο τρόπος με τον οποίον υλοποιείται το START και STOP. Παρακάτω υπάρχει ο κώδικας με σχόλια:

```
void delay_loop(long end)
{
    long i;
    static unsigned int run = 0;    // control flag

    for (i = 0; i < end; i++)
    {
        do
        {
            EALLOW;
            SysCtrlRegs.WDKEY = 0x55;
            SysCtrlRegs.WDKEY = 0xAA;       // service watchdog
            EDIS;
            if(START == 0 && STOP == 1) run = 1;
// if START button is pushed GPIO48's value is zero and run becomes 1
```

```
        }
        while(!run);//while run=0 (STOP pushed) CPU stays in the loop
                    // if run=1 (START pushed) it gets out

        if(STOP == 0) run = 0;
//if STOP is pushed run becomes 0 and the CPU gets stuck in the while
//loop abovestop execution
    }
}
```

Η συνάρτηση InitSystem() αρχικοποιεί τον Watchdog και το ρολόι του DSP. Εκτός αν ζητηθεί κάτι διαφορετικό αυτή η συνάρτηση δε θα αλλάξει στα πλαίσια αυτής της διπλωματικής. Υπενθυμίζουμε ότι για να μην υπερχειλίσει ο Watchdog απαιτείται περιοδικά στο πρόγραμμα να εκτελείται το παρακάτω ζεύγος εντολών. Για αυτό το λόγο έχει συμπεριληφθεί στη διαδικασία **delay_loop.**

```
EALLOW;
SysCtrlRegs.WDKEY = 0x55;
SysCtrlRegs.WDKEY = 0xAA;       // service watchdog
EDIS;
```

- Παρακάτω υπάρχει ο κώδικας της main, ο οποίος χρησιμοποιείται για την έξοδο του χρονομέτρου. Αποτελείται από έναν ατέρμονα βρόχο ο οποίος εκτελείται κάθε 100ms και αυξάνει τη μεταβλητή **counter** κατά 1.

```
while(1)
    {
            if(counter&1) GpioDataRegs.GPASET.bit.GPIO9 = 1;
                else GpioDataRegs.GPACLEAR.bit.GPIO9 = 1;
            if(counter&2) GpioDataRegs.GPASET.bit.GPIO11 = 1;
                else GpioDataRegs.GPACLEAR.bit.GPIO11 = 1;
            if(counter&4) GpioDataRegs.GPBSET.bit.GPIO34 = 1;
                else GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1;
            if(counter&8) GpioDataRegs.GPBSET.bit.GPIO49 = 1;
                else GpioDataRegs.GPBCLEAR.bit.GPIO49 = 1;

            delay_loop(1000000);
            counter++;
    }
```

Τα τέσσερα **"if ...else"** αποφασίζουν αν η εκάστοτε έξοδος θα είναι ON η OFF ανάλογα με την τιμή του counter. Το GPIO9 είναι το least significant bit του χρονομετρητή. Οπότε, θέλουμε να είναι ON κάθε φορά που ο counter έχει τιμή xxx1 και OFF κάθε φορά που ο counter έχει τιμή xxx0. Αυτό επιτυγχάνεται με τον bit προς bit πολλαπλασιασμό του counter με τον αριθμό 0001. Ο αριθμός 0001 ονομάζεται μάσκα, στον προγραμματισμό, καθώς ο πολλαπλασιασμός του με κάποιον άλλον αριθμό 4-bit μας επιτρέπει να απομονώσουμε την πληροφορία του πρώτου bit.

$$(0001)_2 \,\&\, (xxxx)_2 = (000x)_2 \tag{4.8}$$

,όπου x=0 ή 1

Οπότε εάν το αποτέλεσμα της (counter&1) είναι 1 τότε το GPIO9 τίθεται σε υψηλό δυναμικό, ενώ σε αντίθετη περίπτωση σε χαμηλό.Αντίστοιχες μάσκες χρησιμοποιούνται και στα επόμενα ψηφία.

4.6.1 Απαραίτητα αρχεία για την εφαρμογή

Για αυτήν αλλά και για τις επόμενες εφαρμογές χρειάζονται τα παρακάτω αρχεία από την Texas instruments. Υπάρχουν διαθέσιμα στο διαδίκτυο στην ιστοσελίδα της εταιρίας.

- Από C:\tidcs\c28\dsp2833x\v131\DSP2833x_headers\source προσθέστε:
  DSP2833x_GlobalVariableDefs.c

Αυτό το αρχείο ορίζει όλα τα ονόματα των καθολικών μεταβλητών.

- Από C:\tidcs\c28\dsp2833x\v131\DSP2833x_common\source προσθέστε:
  DSP2833x_CodeStartBranch.asm

Αρχείο assembly που λειτουργεί σαν αρχή του προγράμματος. Ο linker θα ενώσει στο τέλος αυτού του κώδικα τον δικό μας

- Από C:\tidcs\c28\dsp2833x\v131\DSP2833x_common\cmd προσθέστε:
  28335_RAM_lnk.cmd

Ενώνει όλες τις εντολές του προγράμματος με τη φυσική μνήμη. Σε μελλοντικές εφαρμογές μπορεί να χρειαστεί να επεξεργαστούμε αυτό το αρχείο ώστε να επεκτείνουμε τη διαθέσιμη μνήμη για ορισμένα κομμάτια του κώδικα.

- Από C:\tidcs\c28\dsp2833x\v131\DSP2833x_headers\cmd προσθέστε:
  DSP2833x_Headers_nonBIOS.cmd

Ενώνει όλες τις καθολικές μεταβλητές σε φυσική μνήμη.

- Από C:\CCStudio_v3.3\c2000\cgtools\lib προσθέστε:
  rts2800_fpu32.lib

Η βιβλιοθήκη είναι απαραίτητη για κάθε πρόγραμμα C. Η συγκεκριμένη παρέχει υποστήριξη και για πράξεις κινητής υποδιαστολής.

Απαιτούνται και οι παρακάτω ρυθμίσεις:

- Πρέπει να παρέχουμε στον Μεταγλωττιστή τη διεύθυνση των φακέλων που συμπεριλάβαμε:
  Project → Build Options → Compiler tab

Και στο Include Search Path συμπληρώνουμε:

C:\tidcs\C28\dsp2833x\v131\DSP2833x_headers\include

- Για την υποστήριξη πράξεων κινητής υποδιαστολής συμπληρώνουμε στο:

  Project → Build Options → Compiler tab → Advanced

Και στο πεδίο Floating point support επιλέγουμε Fpu32.

- Πάμε στο: Project → Build Options → Linker tab

Και στο πεδίο stack size συμπληρώνουμε 400.

4.6.2 Κώδικας με σχόλια

```
#include "DSP2833x_Device.h"
#define STOP      GpioDataRegs.GPADAT.bit.GPIO17
#define START     GpioDataRegs.GPBDAT.bit.GPIO48
// Prototype statements for functions found within this file.
void Gpio_select(void);
void InitSystem(void);
void delay_loop(long);
//###########################################################################
//                       main code
//###########################################################################
void main(void)
{
    int counter=0;      // binary counter for digital output
    long delay;
    InitSystem();       // Basic Core Initialization
    DINT;               // Disable all interrupts
    Gpio_select();      // GPIO9, GPIO11, GPIO34 and GPIO49 as output


    while(1)
    {
            if(counter&1) GpioDataRegs.GPASET.bit.GPIO9 = 1;
                else GpioDataRegs.GPACLEAR.bit.GPIO9 = 1;
            if(counter&2) GpioDataRegs.GPASET.bit.GPIO11 = 1;
                else GpioDataRegs.GPACLEAR.bit.GPIO11 = 1;
            if(counter&4) GpioDataRegs.GPBSET.bit.GPIO34 = 1;
                else GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1;
            if(counter&8) GpioDataRegs.GPBSET.bit.GPIO49 = 1;
                else GpioDataRegs.GPBCLEAR.bit.GPIO49 = 1;

            delay_loop(1000000);
            counter++;
    }
}

void delay_loop(long end)
{
    long i;
    static unsigned int run = 0;    // control flag

    for (i = 0; i < end; i++)
    {
        do
        {
            EALLOW;
            SysCtrlRegs.WDKEY = 0x55;
            SysCtrlRegs.WDKEY = 0xAA;       // service watchdog
```

```
            EDIS;
            if(START == 0 && STOP == 1) run = 1;
        }
        while(!run);

        if(STOP == 0) run = 0;
    }
}


void Gpio_select(void)
{
EALLOW;
GpioCtrlRegs.GPAMUX1.all = 0;// GPIO15 ... GPIO0
GpioCtrlRegs.GPAMUX2.all = 0;// GPIO31 ... GPIO16
GpioCtrlRegs.GPBMUX1.all = 0;// GPIO47 ... GPIO32
GpioCtrlRegs.GPBMUX2.all = 0;// GPIO63 ... GPIO48
GpioCtrlRegs.GPCMUX1.all = 0;// GPIO79 ... GPIO64
GpioCtrlRegs.GPCMUX2.all = 0;// GPIO87 ... GPIO80
    GpioCtrlRegs.GPADIR.all = 0;          // GPIO31-0 as inputs
GpioCtrlRegs.GPADIR.bit.GPIO9 = 1;
GpioCtrlRegs.GPADIR.bit.GPIO11 = 1;
    GpioCtrlRegs.GPBDIR.all = 0;          // GPIO63-32 as inputs
GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1;
GpioCtrlRegs.GPBDIR.bit.GPIO49 = 1;
    GpioCtrlRegs.GPCDIR.all = 0;          // GPIO87-64 as inputs
    EDIS;
}

void InitSystem(void)
{
    EALLOW;
    SysCtrlRegs.WDCR = 0x0028;          // Watchdog enabled, 4.3
milliseconds
    SysCtrlRegs.SCSR = 0x0000;          // Watchdog will cause a
reset
    SysCtrlRegs.PLLSTS.bit.DIVSEL = 2;
    SysCtrlRegs.PLLCR.bit.DIV = 10;     // 30MHz * 10 / 2 = 150 MHz
SYSCLK
    SysCtrlRegs.HISPCP.all = 0x0001;    // SYSCLK / 2
    SysCtrlRegs.LOSPCP.all = 0x0002;    // SYSCLK / 4
    SysCtrlRegs.PCLKCR0.all = 0x0000;
    SysCtrlRegs.PCLKCR1.all = 0x0000;
    SysCtrlRegs.PCLKCR3.all = 0x0000;
    SysCtrlRegs.PCLKCR3.bit.GPIOINENCLK = 1;
    EDIS;
}
// End of SourceCode.
```

4.7 Βιβλιογραφία

[1] «TMS320F28335 Digital Signal Controllers (DSCs) Data Manual» available at http://www.ti.com/lit/ds/sprs439m/sprs439m.pdf

[2] «TMS320F28335 System controls and interrupts» available at http://www.ti.com

ΚΕΦΑΛΑΙΟ 5

ΣΥΣΤΗΜΑ ΔΙΑΚΟΠΩΝ ΔΙΕΡΓΑΣΙΑΣ ΤΟΥ DSC TMS320F28335

5.1 Εισαγωγή [2]

Το κεφάλαιο αυτό έχει ως στόχο τη γνωριμία με το σύστημα διακοπών διεργασίας του F28335. Στο εξής θα αναφέρεται απλά ως "σύστημα διακοπών" για λόγους συντομίας. Για να κατανοήσουμε τι αποτελεί μια διακοπή ας φανταστούμε το επόμενο παράδειγμα. Ένας φοιτητής ακούει μουσική από τον προσωπικό του υπολογιστή ενώ παράλληλα χρησιμοποιεί κάποιον κειμενογράφο. Όταν πατάει κάποιο πλήκτρο τότε στην CPU του υπολογιστή συμβαίνουν τα εξής:

- Διακόπτεται η διεργασία που επιτελούσε όταν κλήθηκε η διακοπή (στο παράδειγμά μας η μουσική)
- Η CPU εκτελεί μία ειδική διαδικασία, την Ρουτίνα Εξυπηρέτησης Διακοπής (ΡΕΔ) του κειμενογράφου ώστε να τυπώσει στην οθόνη το γράμμα.
- Η CPU επιστρέφει στην προηγούμενη διεργασία η οποία ήταν η αναπαραγωγή της μουσικής.

Παρόλα αυτά, ο χρήστης δεν αντιλαμβάνεται τίποτα από τα παραπάνω, καθώς η ταχύτητα με την οποία συμβαίνουν είναι αρκετή ώστε η μουσική να μην διακοπεί ποτέ. Ωστόσο, εάν υπήρχε ένας υπολογιστής με πολύ χαμηλή συχνότητα επεξεργασίας, η μουσική θα έπαυε για αρκετά δευτερόλεπτα κάθε φορά που θα πιεζόταν ένα πλήκτρο.

Με τεχνικούς όρους, διακοπή σε ένα υπολογιστικό σύστημα είναι ένα **ασύγχρονο γεγονός που δημιουργείται από το hardware του συστήματος**. Το γεγονός αυτό εξαναγκάζει την CPU να διακόψει τη διεργασία στην οποία βρισκόταν και να μεταφερθεί σε μία νέα διεργασία η οποία εξυπηρετεί τη διακοπή. Η διεργασία αυτή ονομάζεται Ρουτίνα Εξυπηρέτησης Διακοπής ή ΡΕΔ (Interrupt Service routine-ISR). Αφού τελειώσει η εκτέλεση της ΡΕΔ , η CPU επιστρέφει στην προηγούμενη διεργασία.

Η τεχνική αυτή χρησιμοποιείται για τη δημιουργία περιοδικών γεγονότων (πχ. σε PWM εφαρμογές) αλλά και για την αντιμετώπιση βλαβών σε συστήματα (πχ. σήματα βλάβης-trip). Αρχικά (τμήμα 5.2), παρουσιάζονται όλες οι πιθανές πηγές που μπορούν να πυροδοτήσουν διακοπές στον F28335. Στο τμήμα 5.3 παρατίθεται ο τρόπος ενεργοποίησης μιας συγκεκριμένης πηγής διακοπών. Επιπλέον εξηγείται ποιες είναι οι διαθέσιμες εσωτερικές διακοπές που αφορούν τις περιφερειακές συσκευές του συστήματος.

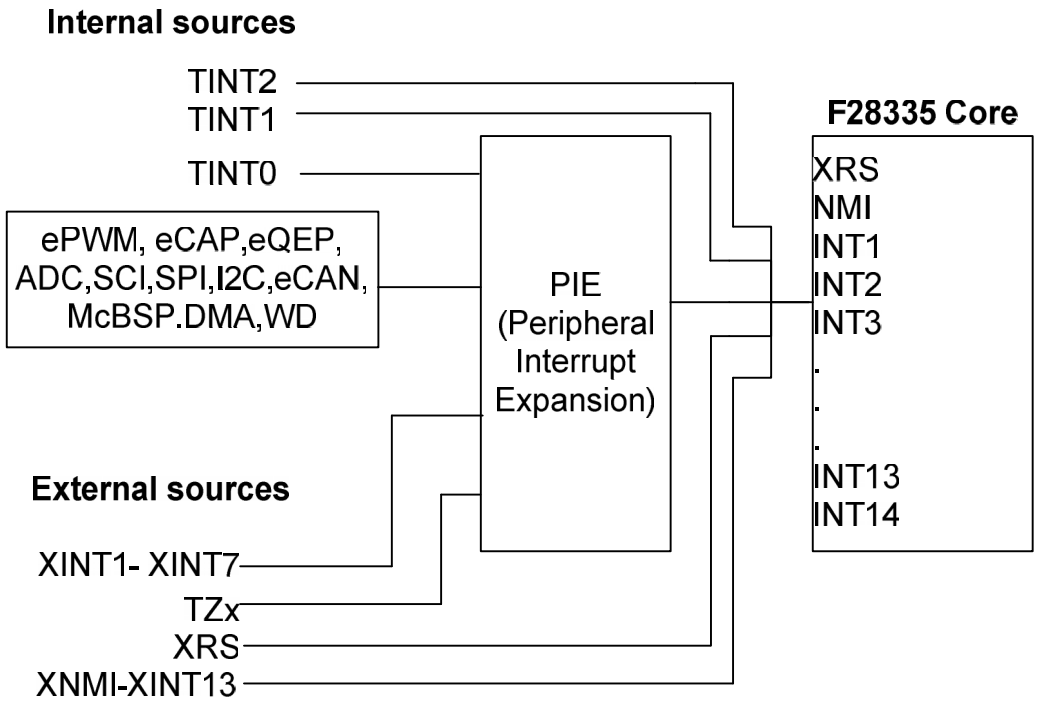
Στο τέλος του κεφαλαίου υπάρχουν δύο εφαρμογές για δύο διαφορετικούς τύπους διακοπών. Η πρώτη είναι μία εσωτερική περιοδική διακοπή με τη χρήση χρονιστή (τμήμα 5.5) και η δεύτερη είναι μία εξωτερική διακοπή, με πηγή ένα σήμα εισόδου σε ένα GPIO pin (τμήμα 5.6).

5.2 Πηγές διακοπών [1]

Οι διακοπές που μπορεί να δεχθεί ο F28335 χωρίζονται σε maskable (απενεργοποιήσιμες) και non-maskable (μη απενεργοποιήσιμες). Η διαφορά τους είναι ότι οι maskable μπορούν να απενεργοποιηθούν ανεξαρτήτως αν το hardware δέχεται σήματα διακοπής ή όχι. Οι unmaskable διακοπές είναι 2 και αφορούν γεγονότα υψηλής προτεραιότητας και ασφάλειας.

Οι maskable διακοπές είναι 96, ωστόσο όπως φαίνεται στο σχήμα 5.1 οι φυσικές γραμμές διακοπών είναι μόνο δεκατέσσερις (INT1 έως INT14). Όπως γίνεται κατανοητό πρέπει μέσω διεργασίας λογισμικού να αντιστοιχιστούν σε μία γραμμή διακοπής INTx περισσότερες από μία διακοπές.

Ο τρόπος με τον οποίον επιτυγχάνεται αυτό στον F28335 είναι μέσω της μονάδας Επέκτασης Διακοπών από Περιφερειακές Συσκευές (Peripheral Interrupt Expansion- PIE). Η PIE επεκτείνει τα ήδη υπάρχοντα διανύσματα των INTx γραμμών στην πτητική μνήμη, παρέχοντας έτσι 96 νέα διανύσματα των 32 bit για όλες τις πιθανές διακοπές. Η θέση μνήμης από την οποία αρχίζει η PIE είναι η 0x00 0D00.

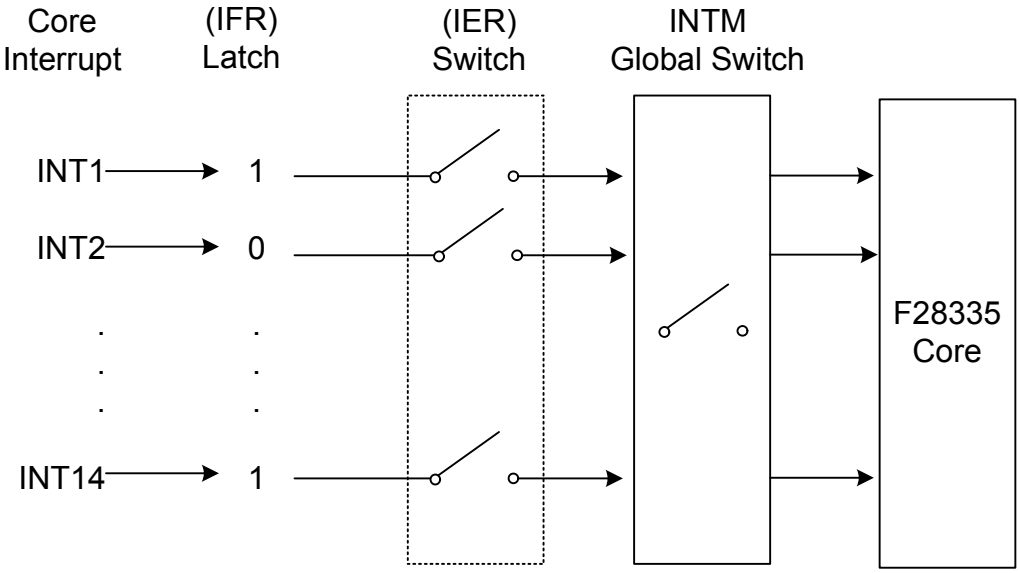


Σχήμα 5.1: Πηγές maskable διακοπών για τον F28335

Οι διακοπές χωρίζονται σε εσωτερικές και εξωτερικές. Οι εσωτερικές προέρχονται από τα περιφερειακά της πλακέτας ή από τους τρεις χρονιστές της CPU (TIN0 έως TINT2). Οι εξωτερικές διακοπές προέρχονται από σήματα GPIO (XINT1 έως 7), ή από σήματα σφαλμάτων (TZx, trip zone)

5.3 Διαδικασία ενεργοποίησης διακοπών [1]

Όπως αναφέρθηκε στην προηγούμενη σελίδα, οι maskable διακοπές πρέπει να ενεργοποιηθούν για να επικοινωνήσουν το αίτημά τους στον επεξεργαστή του DSC. Για αυτό το σκοπό το σήμα πρέπει να περάσει από δύο «διακόπτες». Ο πρώτος είναι ο καταχωρητής ενεργοποίησης διακοπής (Interrupt Enable Register –IER, Σχήμα 5.4), ο οποίος έχει πεδία για κάθε γραμμή διακοπής INTx, και ο δεύτερος είναι Καθολικός Καταχωρητής Επίτρεψης Διακοπών (Interrupt Global Mask- INTM, Σχήμα 5.5) ο οποίος είτε επιτρέπει είτε αποτρέπει όλες τις διακοπές. Καταχωρητής IFR (Interrupt Flag Register, Σχήμα 5.3) περιέχει το σήμα της διακοπής το οποίο διαδίδεται μέχρι τον επεξεργαστή.



Σχήμα 5.2: Δομικό διάγραμμα διαδικασίας ενεργοποίησης διακοπών

Interrupt Flag Register (IFR)

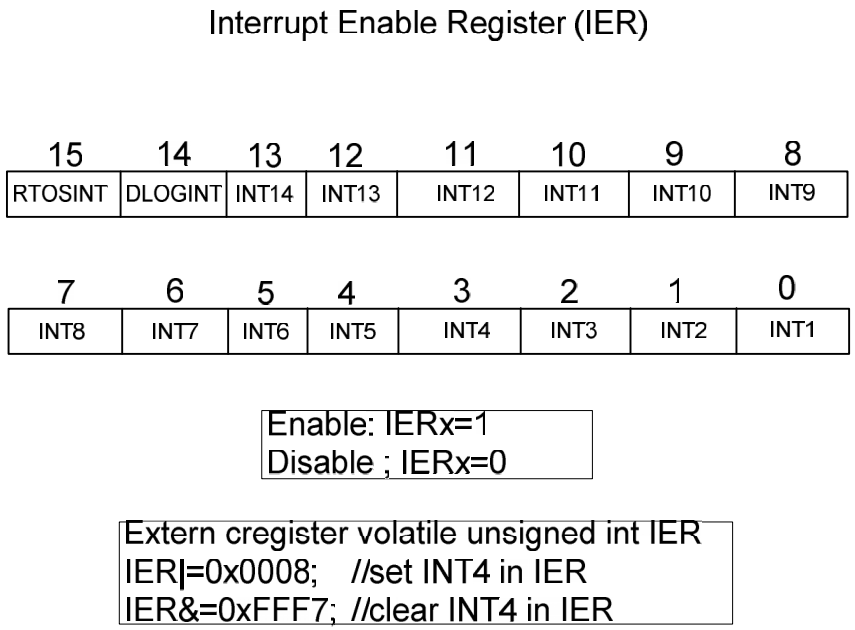
15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9

7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1

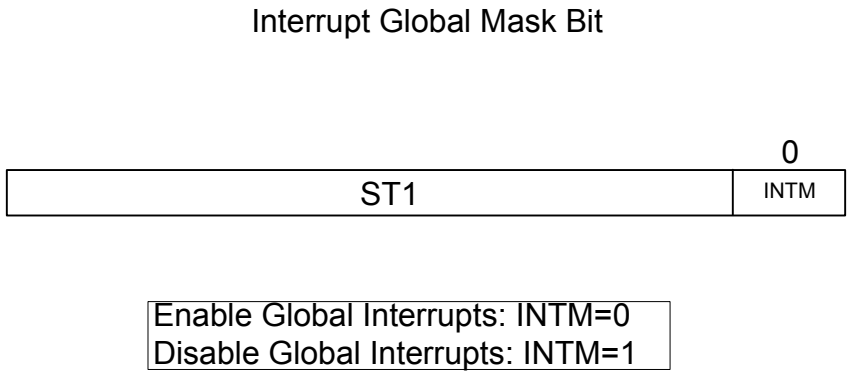
Pending: IFRx=1
Absent ; IFRx=0

```
Extern cregister volatile unsigned int IFR
IFR|=0x0008;    //set INT4 in IFR
IFR&=0xFFF7;  //clear INT4 in IFR
```

Σχήμα 5.3: Καταχωρητής σημαίας Διακοπών



Σχήμα 5.4: Καταχωρητής ενεργοποίησης Διακοπών



Σχήμα 5.5: Καθολικός Καταχωρητής Επίτρεψης Διακοπών

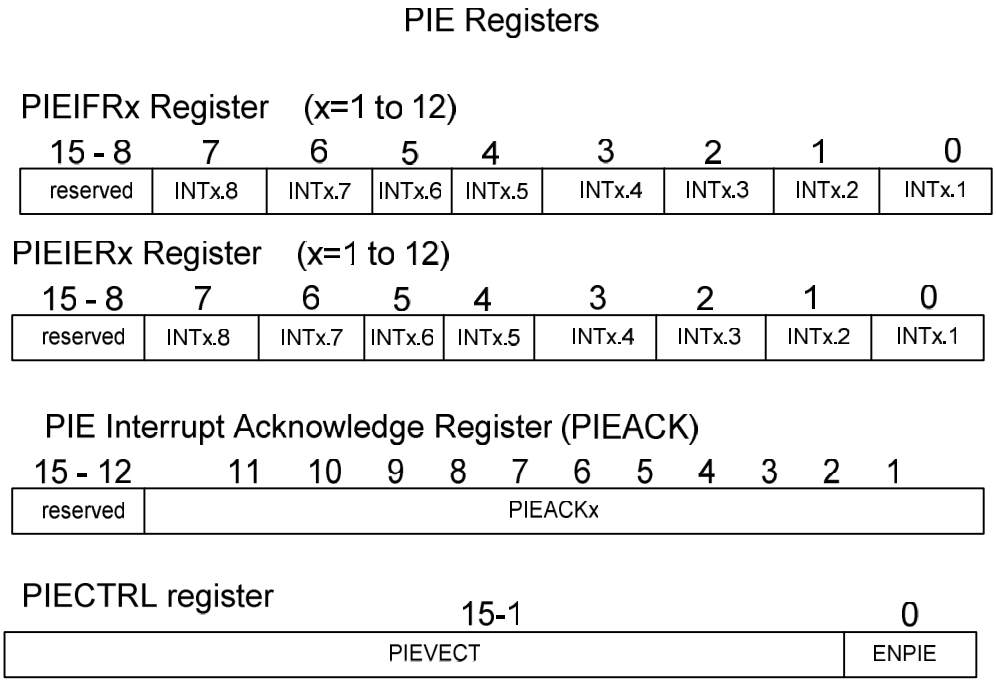
Οπότε για να ενεργοποιηθεί μια διακοπή της γραμμής INT5 πρέπει να χρησιμοποιηθούν οι εξής εντολές:

```
EDIS; // Global interrupt switch disable
IER |=x0010; // INT 5 enable
EINT; // Global interrupt switch enable
```

5.3.1 Μονάδα Επέκτασης Διακοπών από Περιφερειακές Συσκευές (PIE)

Μέχρι τώρα έχουμε καταφέρει να ενεργοποιήσουμε μία γραμμή διακοπών INTx. Ωστόσο, με τη συμβολή του PIE κάθε γραμμή αντιστοιχεί σε έως και 8 διαφορετικές διακοπές.

Στο σχήμα 5.6 αναγράφονται οι καταχωρητές PIE. Παρατηρούμε ότι κάθε βασική γραμμή INTx κρύβει (λόγω του PIE) ένα σύστημα παρόμοιο του σχήματος 5.2 αλλά αφορά κάθε μία από τις 96 διακοπές ξεχωριστά.



Σχήμα 5.6: Καταχωρητές PIE

Ο καταχωρητής PIEACK (τρίτος κατά σειρά στο σχήμα 5.6) χρησιμοποιείται στο τέλος της ΡΕΔ για να μπορεί να δεχθεί το σύστημα νέα διακοπή από τη συγκεκριμένη γραμμή διακοπών.

Ο πίνακας του σχήματος 5.7 αντιστοιχίζει τις γραμμές διακοπών INTx με τις πολυπλεμγένες σε αυτές διακοπές από περιφερειακά. Οι πιο σημαντικές από αυτές είναι:

- TINT0: διακοπή μέσω του χρονιστή της CPU
- ADCINT: διακοπή από τη ADC μονάδα
- EPWMx_INT: διακοπές στον κύκλο του ePWM
- EPWM_TZINT: διακοπές από σήματα trip του EPWM
- ECAPx_INT: διακοπές από τη μονάδα σύλληψης παλμοσειρών
- XINTx: Εξωτερικές διακοπές από GPIO

F2833x PIE Interrupt Assignment Table

	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
INT1	WAKEINT	TINT0	ADCINT	XINT2	XINT1		SEQ2INT	SEQ1INT
INT2			EPWM6_TZINT	EPWM5_TZINT	EPWM4_TZINT	EPWM3_TZINT	EPWM2_TZINT	EPWM1_TZINT
INT3			EPWM6_INT	EPWM5_INT	EPWM4_INT	EPWM3_INT	EPWM2_INT	EPWM1_INT
INT4			ECAP6_INT	ECAP5_INT	ECAP4_INT	ECAP3_INT	ECAP2_INT	ECAP1_INT
INT5							EQEP2_INT	EQEP1_INT
INT6			MXINTA	MRINTA	MXINTB	MRINTB	SPITXINTA	SPIRXINTA
INT7			DINTCH6	DINTCH5	DINTCH4	DINTCH3	DINTCH2	DINTCH1
INT8			SCITXINTC	SCIRXINTC			I2CINT2A	I2CINT1A
INT9	ECAN1_INTB	ECAN0_INTB	ECAN1_INTA	ECAN0_INTA	SCITXINTB	SCIRXINTB	SCITXINTA	SCIRXINTA
INT10								
INT11								
INT12	LUF	LVF		XINT7	XINT6	XINT5	XINT4	XINT3

Σχήμα 5.7: Πίνακας αντιστοίχισης διακοπών τους καταχωρητές PIE [1]

Παράδειγμα: Ποιες είναι οι εντολές που απαιτούνται για να δηλωθεί μια διακοπή (ADCINT στο συγκεκριμένο παράδειγμα)? Τι μορφή θα έχει η ΡΕΔ?

Απάντηση:

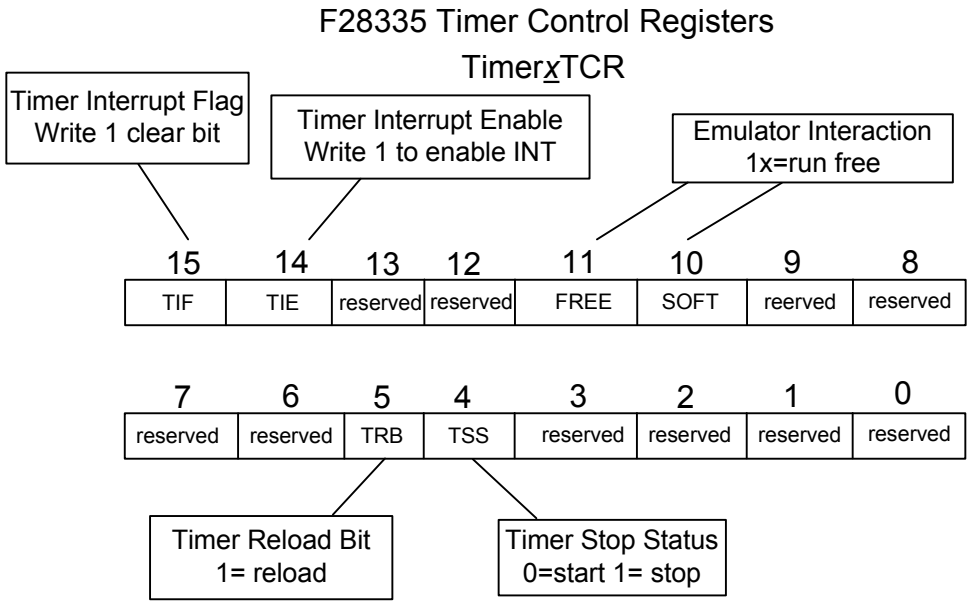
```
DINT; // Global interrupt switch disable
InitPieCtrl();// basic setup of PIE table; from DSP2833x_PieCtrl.c
InitPieVectTable();// default ISR's in PIE
EALLOW;
PieVectTable.ADCINT = &diakopi_adc; // Custom name of the ISR, it is
EALLOW protected
EDIS; // Global interrupt switch disable
PieCtrlRegs.PIEIER1.bit.INTx6 = 1; // PIEIER enable
IER |=1; // IER enable
EINT; // Global interrupt switch enable

interrupt void  diakopi_adc (void)
{
      /***our code***/
      AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;     // Clear INT SEQ1 bit
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP1; // Acknowledge
interrupt to PIE
}
```

Η ταχύτερη απόκριση μίας διακοπής (για τον F28335) είναι 14-16 κύκλοι.

5.4 Χρονιστές του F28335 [1]

Ο F28335 έχει τρεις ανεξάρτητους χρονιστές των 32-bit. Στην εφαρμογή θα χειριστούμε τον Timer 0, ο οποίος αξιοποιεί και τη μονάδα PIE (Σχήμα 5.1).

Στο σχήμα 5.8 παρατίθεται ο καταχωρητής ελέγχου των χρονιστών:



Σχήμα 5.8: Καταχωρητής ελέγχου χρονιστών

Για να οριστεί και να χρησιμοποιηθεί ο χρονιστής και η ΡΕΔ του πρέπει να γίνουν οι γραφτούν οι παρακάτω εντολές:

```
InitPieCtrl();// basic setup of PIE table; from DSP2833x_PieCtrl.c
InitPieVectTable(); // default ISR's in PIE

EALLOW;
PieVectTable.TINT0 = &cpu_timer0_isr;
EDIS;
InitCpuTimers();     // basic setup CPU Timer0, 1 and 2
ConfigCpuTimer(&CpuTimer0,150,100000); // CPU - Timer0 at 100
milliseconds
PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
CpuTimer0Regs.TCR.bit.TSS = 0;  // start timer0

interrupt void cpu_timer0_isr(void)
{
    /***code***/
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

5.4.1 Εφαρμογή Διακοπής χρονιστή

Σε αυτό το απλό παράδειγμα θα αξιοποιήσουμε έναν από τους τρεις χρονιστές του F28335 για να παράγουμε μία περιοδική διακοπή. Ως βάση θα χρησιμοποιήσουμε την εφαρμογή του προηγούμενο κεφαλαίου (χωρίς το start και stop) και θα

αντικαταστήσουμε την delay_loop που παρήγαγε καθυστέρηση απασχολώντας όμως τον επεξεργαστή. Στην εφαρμογή αυτή θα χρησιμοποιήσουμε και το Peripheral Interrupt Expansion (PIE) unit.

5.4.2 Σημαντικά χωρία Κώδικα

- ```
  EALLOW;
  PieVectTable.TINT0 = &cpu_timer0_isr;
  EDIS;
  ```

Η εντολή αυτή αντιστοιχίζει το διάνυσμα της διακοπής του Timer 0 με το όνομα της ρουτίνας εξυπηρέτησης διακοπής που εμείς επιθυμούμε.

- ```
  InitCpuTimers();     // basic setup CPU Timer0, 1 and 2
  ConfigCpuTimer(&CpuTimer0,150,100000); // CPU - Timer0 at 100
  milliseconds
  ```

Εντολές οι οποίες αρχικοποιούν τον χρονιστή. Το δεύτερο όρισμα της ConfigCpuTimer είναι η συχνότητα του ρολογιού (150MHz) και το τρίτο όρισμα είναι η περίοδος της διακοπής σε μs. Το τρίτο όρισμα είναι η περίοδος της διακοπής σε ns.

- ```
  PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
  IER |=1;
  ```

Ενεργοποίηση της διακοπής timer_interrupt 0 βάσει των στοιχείων του σχήματος 5.7

Πέρα από τα αρχεία που προστέθηκαν στο προηγούμενο παράδειγμα αυτή η εφαρμογή απαιτεί και τα παρακάτω. Όπως και τα προηγούμενα, τα αρχεία αυτά υπάρχουν διαθέσιμα στο διαδίκτυο στην ιστοσελίδα της εταιρίας.

- Από C:\tidcs\c28\dsp2833x\v131\DSP2833x_common\source προσθέστε:
  DSP2833x_SysCtrl.c
  DSP2833x_ADC_cal.asm
  DSP2833x_usDelay.asm

Τα αρχεία αυτά περιέχουν συναρτήσεις οι οποίες χρησιμοποιούνται στο παράδειγμα. Οι δύο τελευταίες δεν είναι απαραίτητες για τις εφαρμογές των επόμενων κεφαλαίων (δεν έχουν αρνητικό αντίκτυπο στο project, απλά είναι περιττές)

- Από C:\tidcs\c28\dsp2833x\v131\DSP2833x_common\source προσθέστε:
  DSP2833x_PieCtrl.c
  DSP2833x_PieVect.c
  ```

DSP2833x_DefaultIsr.c
DSP2833x_CpuTimers.c

Απαιτούνται και οι παρακάτω ρυθμίσεις:

- Πρέπει να παρέχουμε στον Μεταγλωττιστή τη διεύθυνση των φακέλων που συμπεριλάβαμε:

  Project → Build Options → Compiler tab

Και στο Include Search Path συμπληρώνουμε:

C:\tidcs\C28\dsp2833x\v131\DSP2833x_headers\include;
C:\tidcs\C28\dsp2833x\v131\DSP2833x_common\include

Όλες οι υπόλοιπες ρυθμίσεις είναι ίδιες με το προηγούμενο παράδειγμα

5.4.3 Κώδικας με σχόλια

```
#include "DSP2833x_Device.h"
// external function prototypes
extern void InitSysCtrl(void);
extern void InitPieCtrl(void);
extern void InitPieVectTable(void);
extern void InitCpuTimers(void);
extern void ConfigCpuTimer(struct CPUTIMER_VARS *, float, float);
// Prototype statements for functions found within this file.
void Gpio_select(void);
interrupt void cpu_timer0_isr(void);
//###########################################################################
//                       main code
//###########################################################################
void main(void)
 {
    int counter=0;  // binary counter for digital output
    InitSysCtrl();  // Basic Core Init from DSP2833x_SysCtrl.c
    EALLOW;
    SysCtrlRegs.WDCR= 0x00AF;   // Re-enable the watchdog
    EDIS;           // 0x00AF  to NOT disable the Watchdog, Prescaler
= 64
    DINT;               // Disable all interrupts
    Gpio_select();      // GPIO9, GPIO11, GPIO34 and GPIO49 as output
    InitPieCtrl();      // basic setup of PIE table; from
DSP2833x_PieCtrl.c
    InitPieVectTable(); // default ISR's in PIE

    EALLOW;
    PieVectTable.TINT0 = &cpu_timer0_isr;
    EDIS;

    InitCpuTimers();    // basic setup CPU Timer0, 1 and 2
```

```
    ConfigCpuTimer(&CpuTimer0,150,100000); // CPU - Timer0 at 100
milliseconds
    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;

    IER |=1;

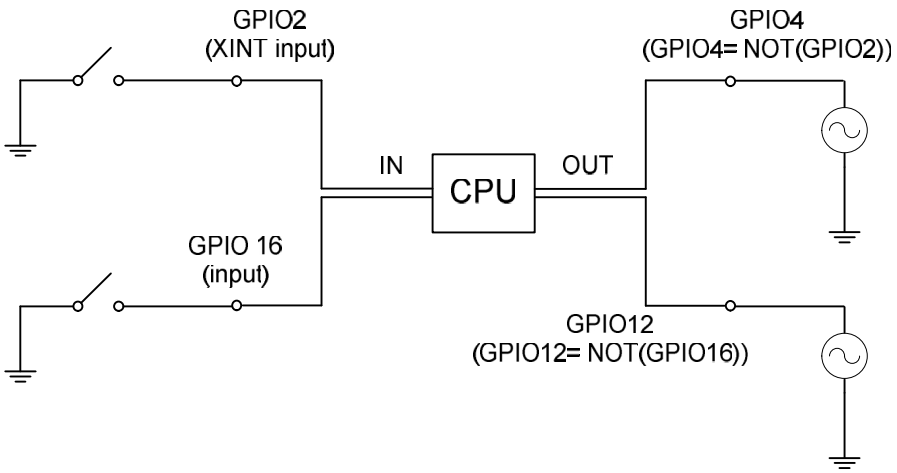
    EINT;
    ERTM;
    CpuTimer0Regs.TCR.bit.TSS = 0;  // start timer0

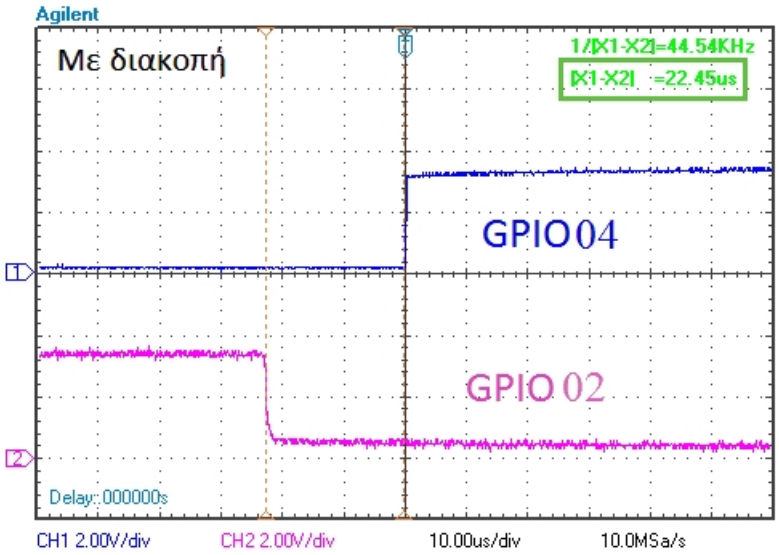
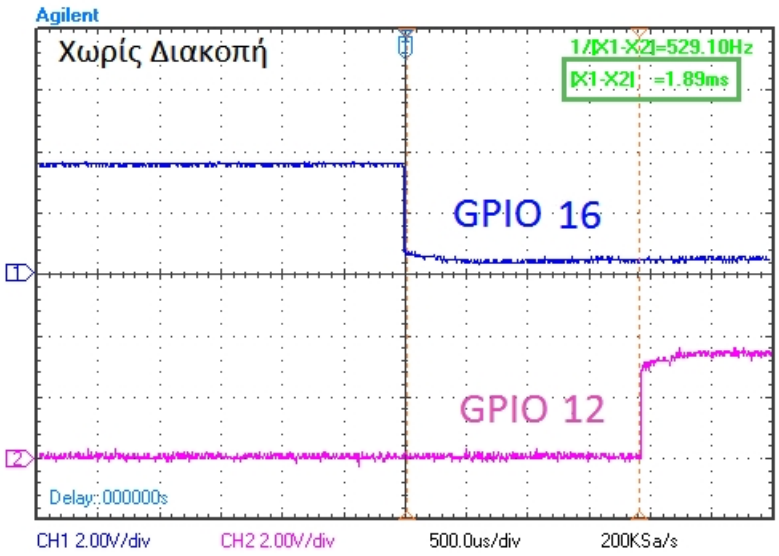
    while(1)
    {
            while(CpuTimer0.InterruptCount == 0);
            CpuTimer0.InterruptCount = 0;

            EALLOW;
            SysCtrlRegs.WDKEY = 0x55;   // service WD #1
            EDIS;

            counter++;
            if(counter&1) GpioDataRegs.GPASET.bit.GPIO9 = 1;
                else GpioDataRegs.GPACLEAR.bit.GPIO9 = 1;
            if(counter&2) GpioDataRegs.GPASET.bit.GPIO11 = 1;
                else GpioDataRegs.GPACLEAR.bit.GPIO11 = 1;
            if(counter&4) GpioDataRegs.GPBSET.bit.GPIO34 = 1;
                else GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1;
            if(counter&8) GpioDataRegs.GPBSET.bit.GPIO49 = 1;
                else GpioDataRegs.GPBCLEAR.bit.GPIO49 = 1;
    }
}

void Gpio_select(void)
{
EALLOW;
GpioCtrlRegs.GPAMUX1.all = 0;        // GPIO15 ... GPIO0
GpioCtrlRegs.GPAMUX2.all = 0;        // GPIO31 ... GPIO16
GpioCtrlRegs.GPBMUX1.all = 0;        // GPIO47 ... GPIO32
GpioCtrlRegs.GPBMUX2.all = 0;        // GPIO63 ... GPIO48
GpioCtrlRegs.GPCMUX1.all = 0;        // GPIO79 ... GPIO64
GpioCtrlRegs.GPCMUX2.all = 0;        // GPIO87 ... GPIO80
    GpioCtrlRegs.GPADIR.all = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO9 = 1;  // GPIO9 as output
GpioCtrlRegs.GPADIR.bit.GPIO11 = 1; // GPIO11 as output
    GpioCtrlRegs.GPBDIR.all = 0;         // GPIO63-32 as inputs
    GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1; // GPIO34 as output
GpioCtrlRegs.GPBDIR.bit.GPIO49 = 1; // GPIO49 as output
GpioCtrlRegs.GPCDIR.all = 0;         // GPIO87-64 as inputs
    EDIS;
}

interrupt void cpu_timer0_isr(void)
{
    CpuTimer0.InterruptCount++;
    EALLOW;
    SysCtrlRegs.WDKEY = 0xAA;   // service WD #2
    EDIS;
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
// End of SourceCode.
```

5.5 Εξωτερικές διακοπές από GPIO [1]

Σε ένα απλό κύκλωμα ελέγχου για ηλεκτρονικά ισχύος, όταν λάβουμε σήμα σφάλματος, θέλουμε ακαριαία να βγάλουμε εκτός ένα το κυκλώματα ισχύος. Το παραπάνω μπορεί να υλοποιηθεί με δύο τρόπους σε κώδικα. Ο ένας είναι να ελέγχουμε περιοδικά την τιμή εισόδου του pin σφάλματος, ενώ ο άλλος είναι να προκαλούμε διακοπή τη στιγμή που εντοπίζεται το σφάλμα μέσω των XINT (external interrupt)

Οι XINT στην ουσία είναι διακοπές οι οποίες πυροδοτούνται από τις ακμές (θετικές, αρνητικές ή και τις δύο) που εντοπίζονται στα pins της αναπτυξιακής πλακέτας (GPIO pins). Ο f28335 υποστηρίζει 7 διαφορετικές XINT. Στο παρακάτω χάρτη φαίνονται σε ποιες γραμμές και στήλες αντιστοιχεί η κάθε μία.

F2833x PIE Interrupt Assignment Table

	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
INT1	WAKEINT	TINT0	ADCINT	XINT2	XINT1		SEQ2INT	SEQ1INT
INT2			EPWM6_TZINT	EPWM5_TZINT	EPWM4_TZINT	EPWM3_TZINT	EPWM2_TZINT	EPWM1_TZINT
INT3			EPWM6_INT	EPWM5_INT	EPWM4_INT	EPWM3_INT	EPWM2_INT	EPWM1_INT
INT4			ECAP6_INT	ECAP5_INT	ECAP4_INT	ECAP3_INT	ECAP2_INT	ECAP1_INT
INT5							EQEP2_INT	EQEP1_INT
INT6			MXINTA	MRINTA	MXINTB	MRINTB	SPITXINTA	SPIRXINTA
INT7			DINTCH6	DINTCH5	DINTCH4	DINTCH3	DINTCH2	DINTCH1
INT8			SCITXINTC	SCIRXINTC			I2CINT2A	I2CINT1A
INT9	ECAN1_INTB	ECAN0_INTB	ECAN1_INTA	ECAN0_INTA	SCITXINTB	SCIRXINTB	SCITXINTA	SCIRXINTA
INT10								
INT11								
INT12	LUF	LVF		XINT7	XINT6	XINT5	XINT4	XINT3

6 - 16

Σχήμα 5.9: Χάρτης PIE διακοπών

Οπότε για να ενεργοποιηθεί η XINT7 πρέπει να υπάρχουν στον κώδικα οι παρακάτω εντολές:

```
PieCtrlRegs.PIEIER12.bit.INTx5 = 1; // XINT7 enable
IER =0x0800; // line INT12 enable
```

Η αντιστοίχιση των GPIO pins στις διακοπές (XINT1-7) είναι ελεύθερη αλλά με κάποιους περιορισμούς. Οι **XINT1/2** μπορούν να συσχετιστούν μόνο με **GPIO της portA (0-31)** ενώ οι **XINT3/4/5/6/7** με **GPIO της portB (32-63)**. Οι παρακάτω πίνακες κατατοπίζουν σχετικά με το πώς γίνεται η επιλογή αυτή:

Πίνακας 5.1: Επιλογή GPIO πηγής για τις XINT 1 και XINT2

Bits	Πεδίο	Τιμή	Περιγραφή
15-5	reserved		
4-0	GPIOXINTxSEL	00000	Επιλογή GPIO0 σαν πηγή της XINTn διακοπής
		00001	Επιλογή GPIO1 σαν πηγή της XINTn διακοπής
	(n=1 ή 2)	……	
		11110	Επιλογή GPIO30 σαν πηγή της XINTn διακοπής
		11111	Επιλογή GPIO31 σαν πηγή της XINTn διακοπής

Πίνακας 5.2: Επιλογή GPIO πηγής για τις XINT 3 εώς XINT7

Bits	Πεδίο	Τιμή	Περιγραφή
15-5	reserved		
4-0	GPIOXINTxSEL	00000	Επιλογή GPIO32 σαν πηγή της XINTn διακοπής
		00001	Επιλογή GPIO33 σαν πηγή της XINTn διακοπής
	(n=1 ή 2)	……	
		11110	Επιλογή GPIO62 σαν πηγή της XINTn διακοπής
		11111	Επιλογή GPIO63 σαν πηγή της XINTn διακοπής

Οπότε για να αντιστοιχίσουμε την διακοπή XINT3 στο GPIO 48 πρέπει κατά σειρά να κάνουμε τα εξής:

1. Να θέσουμε το GPIO 48 σε λειτουργία GPIO
2. Να το ορίσουμε σαν είσοδο
3. Να δώσουμε τη τιμή 1000 (16) στον GPIOXINT3SEL

Στη συνέχεια υπάρχει μία εφαρμογή που συγκρίνει την απόκριση της διακοπής XINT σε σύγκριση με τον περιοδικό έλεγχο ενός pin.

5.5.1 Εφαρμογή διακοπής από GPIO

Για να τονίσουμε τη διαφορά των δύο προσεγγίσεων (περιοδικός έλεγχος και έλεγχος με διακοπές) ελέγχου δημιουργήσαμε ένα κυρίως πρόγραμμα με μεγάλο χρόνο εκτέλεσης:

```
for (i=0;i<10000;i++)
      {
      float_temp=3.2/35.4;
      float_temp=45.45/34.0;
      float_temp=45.45/34.0;
      float_temp=3.2/35.4;
      float_temp=45.45/34.0;
      float_temp=3.2/35.4;
      float_temp=45.45/34.0;
      float_temp=3.2/35.4;
```

```
float_temp=45.45/34.0;
float_temp=3.2/35.4;
float_temp=45.45/34.0;
float_temp=3.2/35.4;
float_temp=45.45/34.0;
}
```

Στη συνέχεια συνδέσαμε σε δύο GPIO του DSC δύο διακόπτες . Η είσοδος GPIO2 έχει δηλωθεί σαν είσοδος εξωτερικής διακοπής (από την οποία περιμένουμε ταχύτερη απόκριση) και η GPIO16 είναι μια απλή είσοδος. Τα GPIO4 και GPIO12 έχουν ρυθμιστεί ώστε να έχουν πάντα αντίθετη τιμή από τα GPIO2 και GPIO16 αντίστοιχα. Οπότε, αφού κλείσει ένας διακόπτης (σήμα σφάλματος) μπορούμε να δούμε πόσο γρήγορα αποκρίνεται η CPU σε κάθε περίπτωση, παρατηρώντας την τιμή των δύο GPIO εξόδων.



Σχήμα 5.10: Πειραματική διάταξη

- **1ος τρόπος (περιοδικός έλεγχος της τιμής του GPIO16)**

```
while(1)
      {
      for (i=0;i<10000;i++){/* main for loop*/ }

      GpioDataRegs.GPADAT.bit.GPIO12=!(GpioDataRegs.GPADAT.bit.GPIO16
      ); /* GPIO12= NOT(GPIO16)*/
      }
```

Κάθε φορά πρέπει να εκτελεστεί το for ώστε να εξεταστεί η τιμή της εισόδου 16.

- **2ος τρόπος (έλεγχος μέσω διακοπής XINT)**

```
interrupt void GPIO16_isr(void)
{
GPIO4_edge_counter++;
GPIO4_flag=GPIO4_edge_counter%2; // 1 if positive edge, 0 if
negative
GpioDataRegs.GPADAT.bit.GPIO4= (!GPIO4_flag);
PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

Η διακοπή συμβαίνει ασύγχρονα σε οποιαδήποτε φάση εκτέλεσης του προγράμματος.

Τα αποτελέσματα αυτών των δύο προσεγγίσεων φαίνονται στις παρακάτω εικόνες. Η διαφορά είναι της τάξεως του 1,8ms. Προφανώς το παράδειγμα είναι ακραίο (λόγω του μεγάλου χρόνου εκτέλεσης της main) ωστόσο εάν η εφαρμογή απαιτεί συγκεκριμένες προδιαγραφές απόκρισης, η μέθοδος των διακοπών είναι η πιο ασφαλής.



Σχήμα 5.11: Απόκριση σφάλματος με διακοπή. Το GPIO 02 είναι το σφάλμα και το GPIO04 είναι η απόκριση.



Σχήμα 5.12: Απόκριση σφάλματος χωρίς διακοπή. Το GPIO16 είναι το σφάλμα και το GPIO12 είναι η απόκριση.

5.5.2 Κώδικας με σχόλια

```
#include "DSP2833x_Device.h"
#include "IQmathLib.h"

// external function prototypes
extern void InitSysCtrl(void);
extern void InitPieCtrl(void);
extern void InitPieVectTable(void);
extern void InitGpio(void);
// Prototype statements for functions found within this file.
void Setup_XINT(void);
interrupt void xint_error1_isr(void);
// ***********Global Variables**********
int
i,start_edge_counter=0,error1_edge_counter=0,start_enable=0,start_3s=
0,error1_flag=0,start_initialized=0;
int sw_reset1=0,sw_reset2=0;
float float_temp=0;
//############################main code###################################

void main(void)
{
    InitSysCtrl();  //  Basic Core Init from DSP2833x_SysCtrl.c
    EALLOW;
    SysCtrlRegs.WDCR= 0x00AF;   // Re-enable the watchdog
    EDIS;

    DINT;//Disable all interrupts
    InitGpio(); //GPIO16/2 as inputs GPIO4/12 as outputs

    InitPieCtrl();
    InitPieVectTable();

    EALLOW;
    PieVectTable.XINT2=&xint_error1_isr;
    EDIS;
    Setup_XINT();

    IER =0x0001;
    PieCtrlRegs.PIEIER1.bit.INTx5 = 1;      //XINT 4
    EINT;
    ERTM;

    error1_edge_counter=0;
    error1_flag=0;
    while(1)
        {
        EALLOW;
        SysCtrlRegs.WDKEY = 0x55;   // service WD #1
        EDIS;

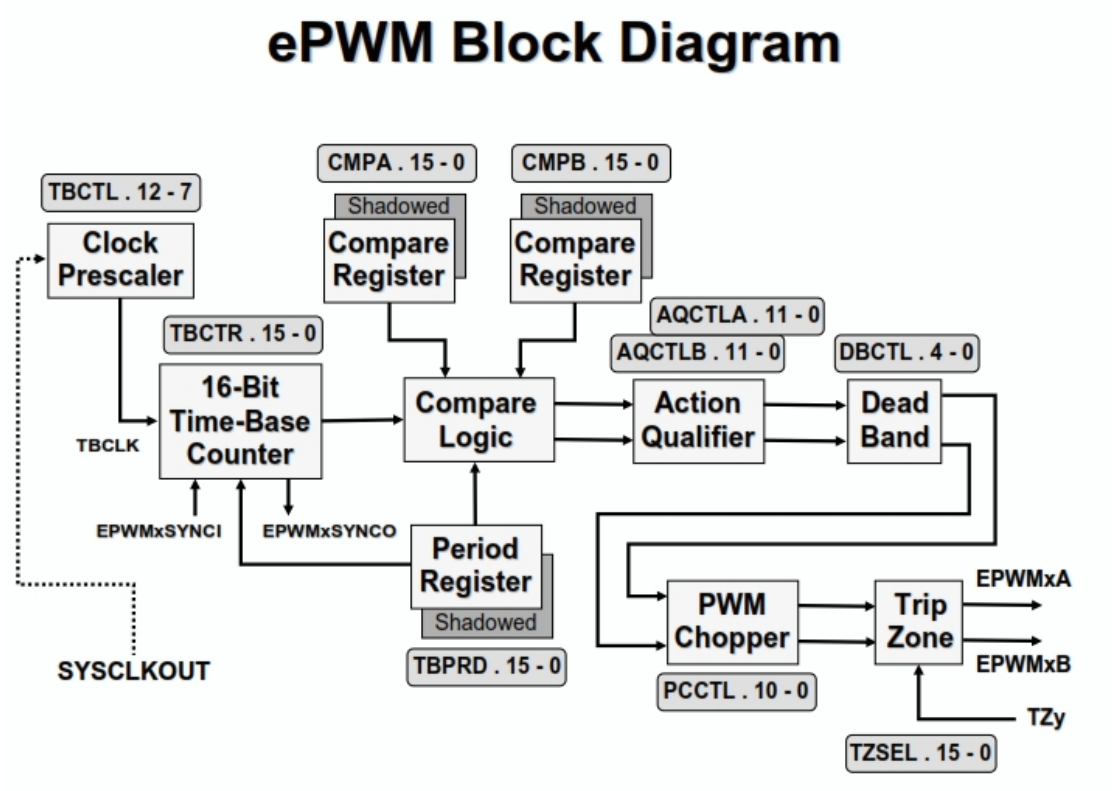
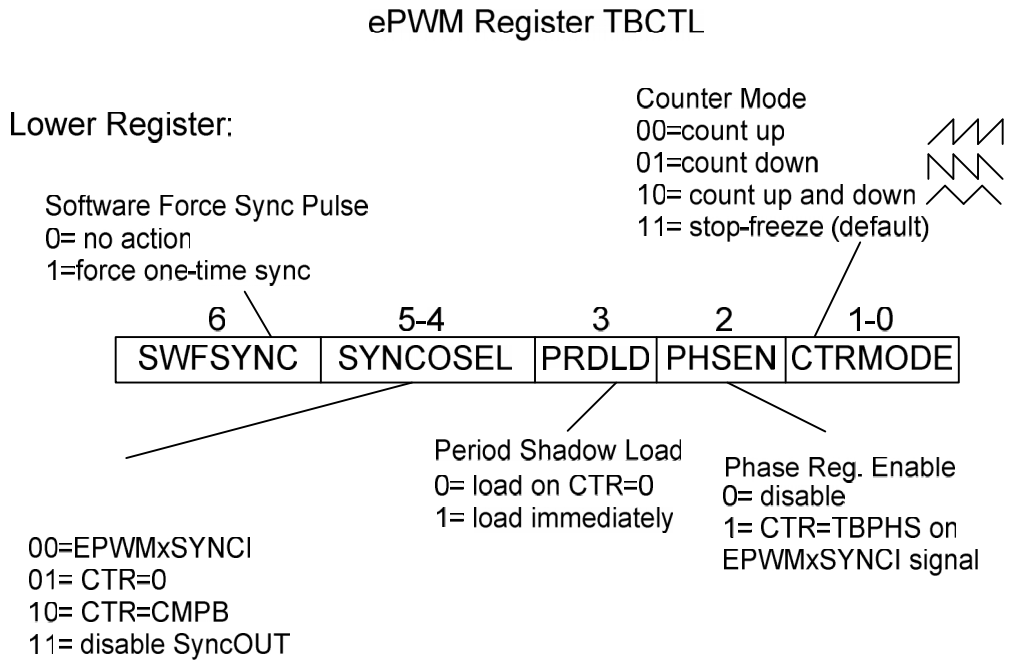
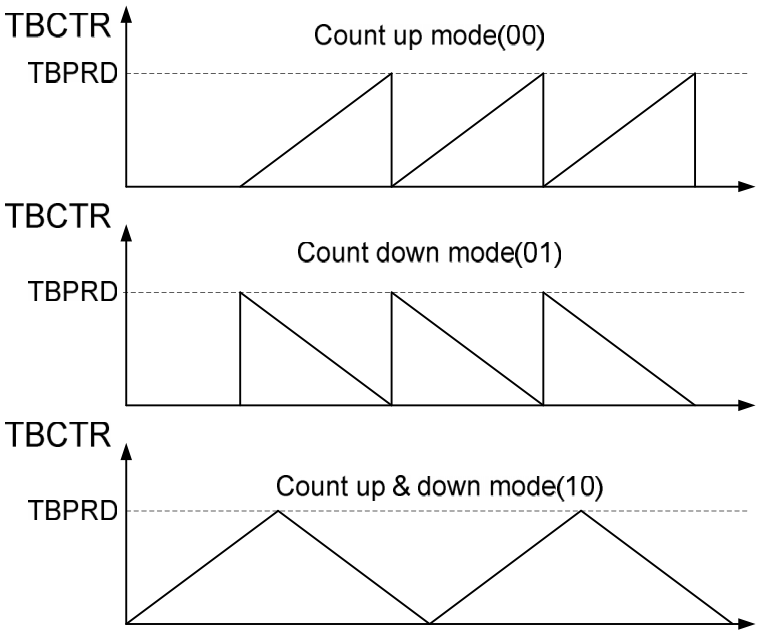
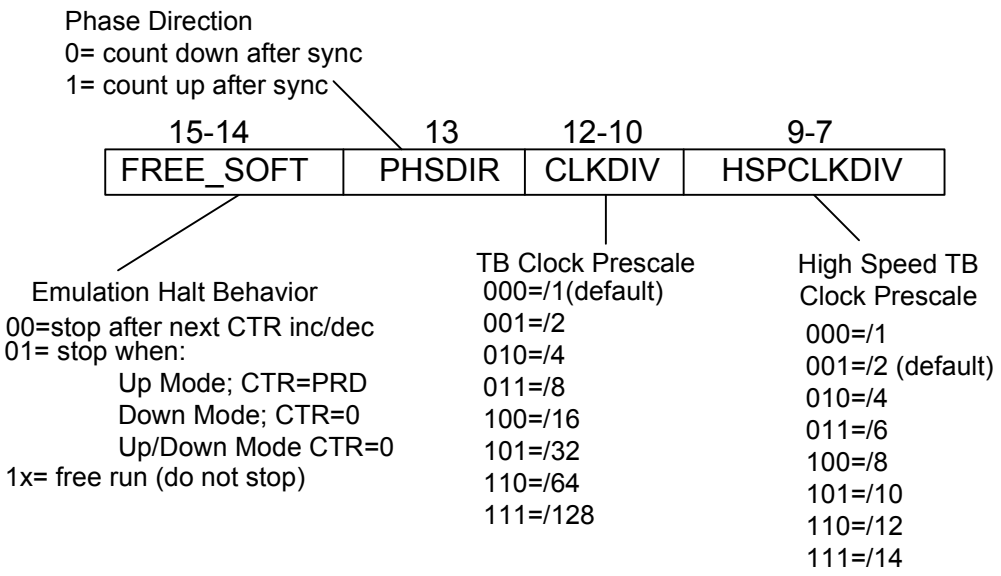
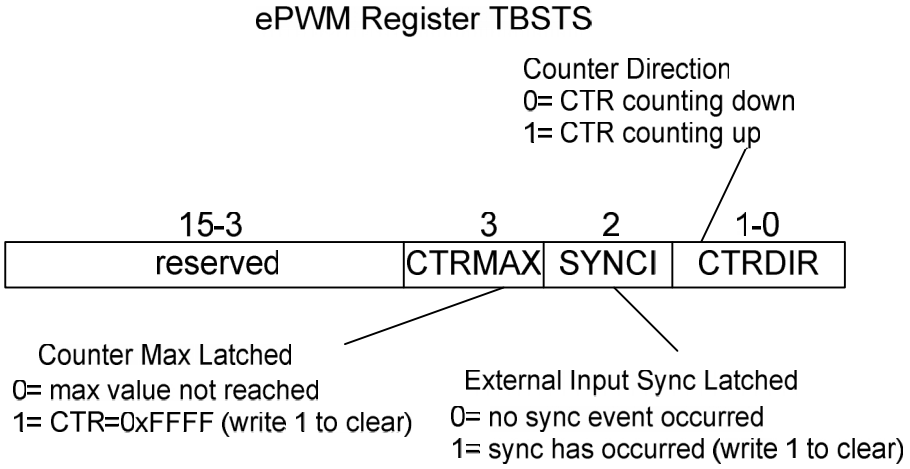
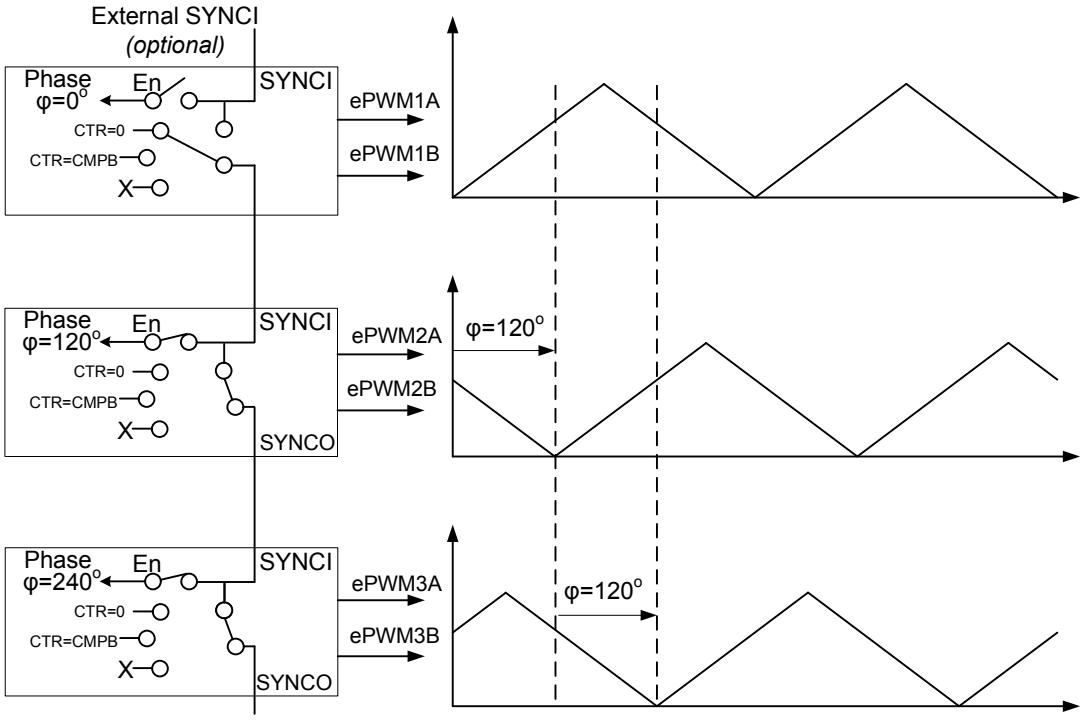
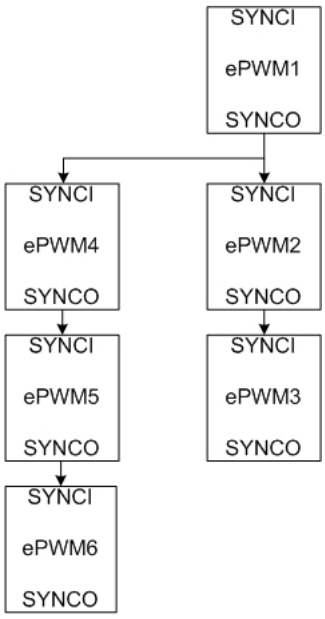
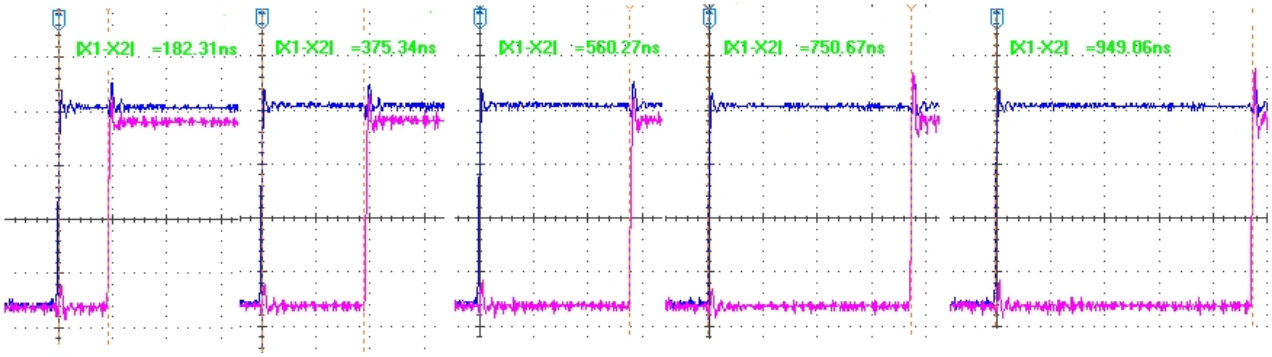
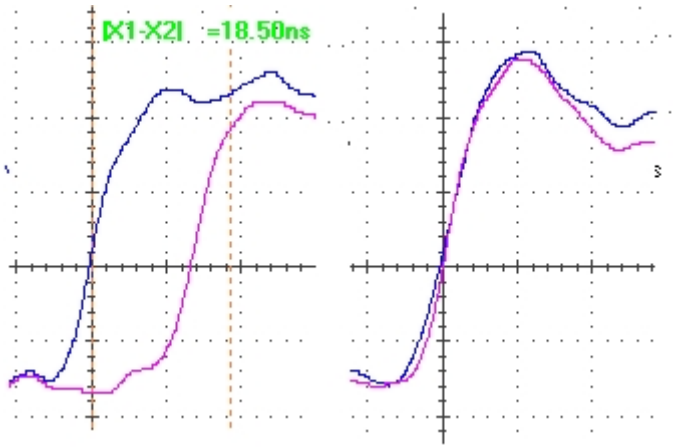
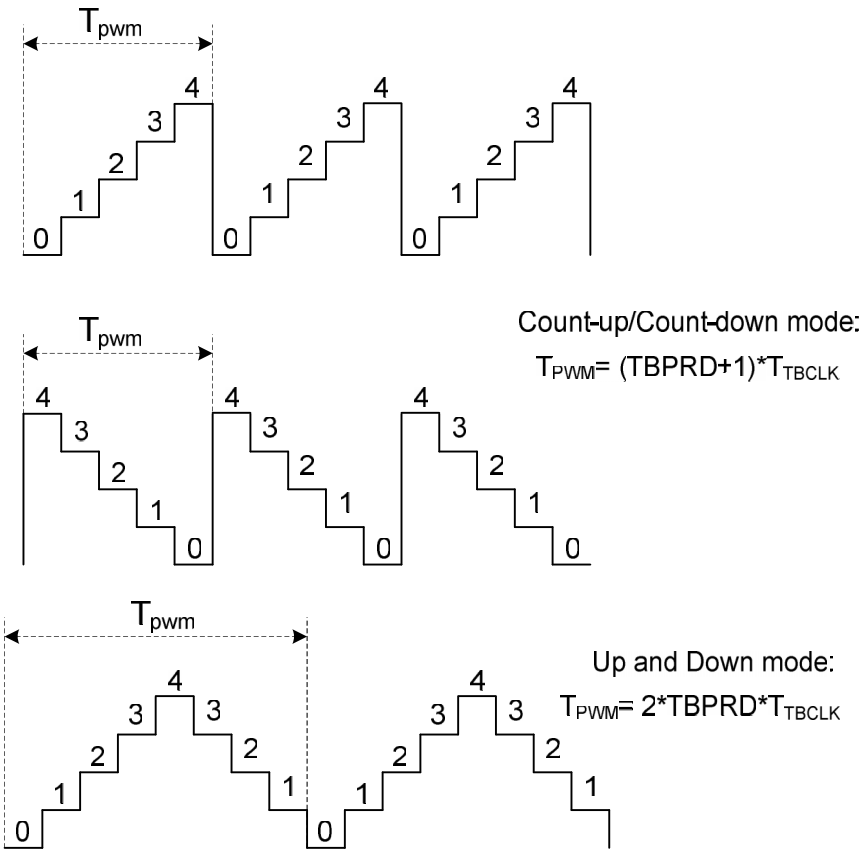
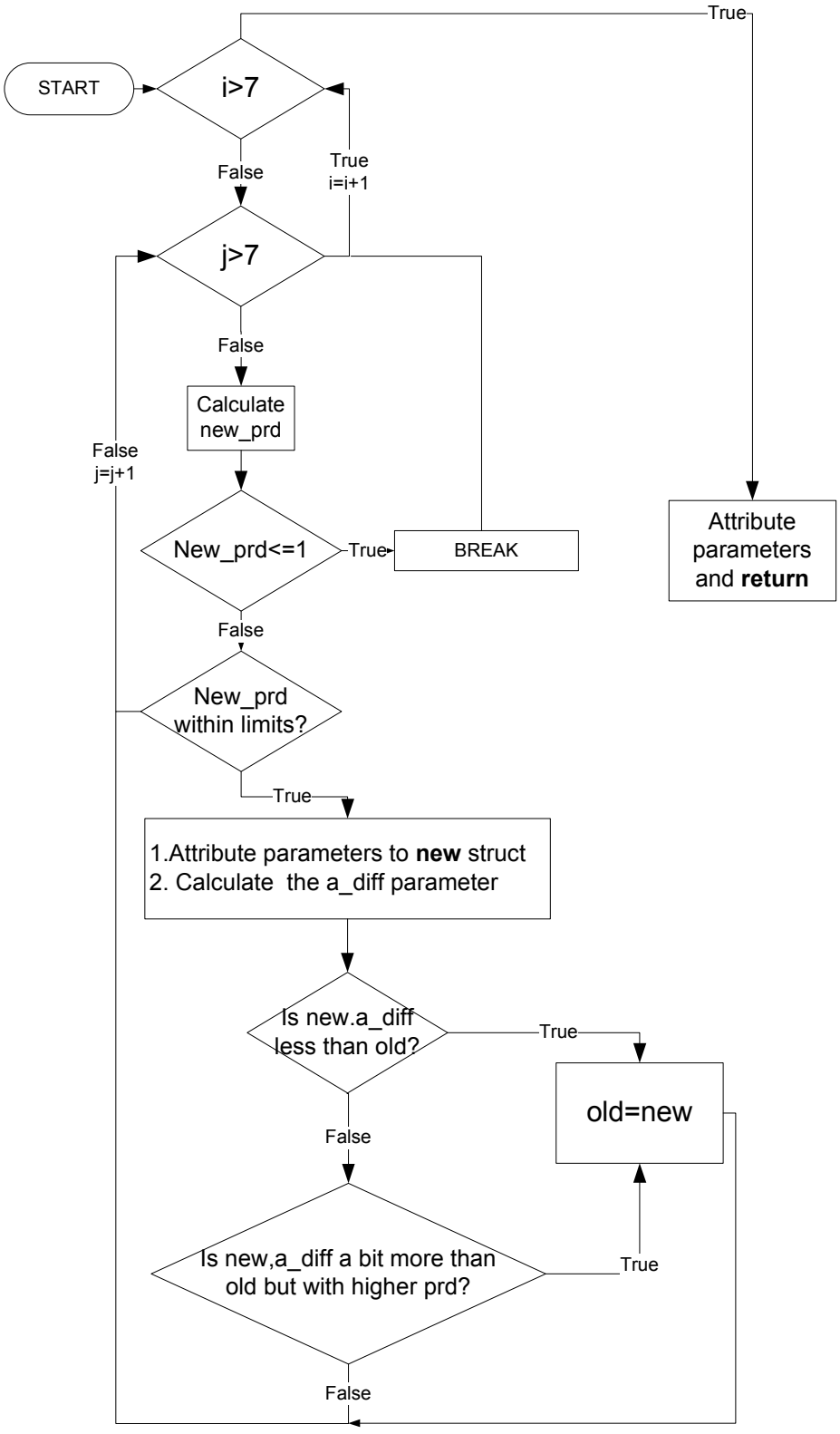
    for (i=0;i<10000;i++){
        float_temp=3.2/35.4;
        float_temp=45.45/34.0;
        float_temp=45.45/34.0;
        float_temp=3.2/35.4;
        float_temp=45.45/34.0;
        float_temp=3.2/35.4;
        float_temp=45.45/34.0;
        float_temp=3.2/35.4;
```

```
        float_temp=45.45/34.0;
        float_temp=3.2/35.4;
        float_temp=45.45/34.0;
        float_temp=3.2/35.4;
        float_temp=45.45/34.0;
        }
 GpioDataRegs.GPADAT.bit.GPIO12= !(GpioDataRegs.GPADAT.bit.GPIO16);
        }
}


void Setup_XINT (void)
{
XIntruptRegs.XINT2CR.bit.POLARITY = 3;  // Both Falling and Rising
edge interrupt
XIntruptRegs.XINT2CR.bit.ENABLE = 1;    // Enable XINT2
EALLOW;
GpioIntRegs.GPIOXINT2SEL.bit.GPIOSEL = 2; // Choice GPIO2 for Xint2
EDIS;
}


interrupt void xint_error1_isr(void)

{
    error1_edge_counter++;
    error1_flag=error1_edge_counter%2;


    //if error pin=1 error flag=0 cause its a pull up input
    GpioDataRegs.GPADAT.bit.GPIO4= (!error1_flag);
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

// End of SourceCode.
```

5.6 Βιβλιογραφία

[1] «TMS320F28335 System controls and interrupts» available at http://www.ti.com

[2] Patterson D.A. and Hennessy J.L.«Οργάνωση και σχεδίαση υπολογιστών » , 2006

[3] Πεκμετζή Κ. «Συστήματα Μικροϋπολογιστών, Τόμος Ι: Μικροεπεξεργαστές Intel 80x86, Pentium και ARM», 2009.

ΚΕΦΑΛΑΙΟ 6

EPWM MODULE - ΜΟΝΑΔΑ ΒΑΣΗΣ ΧΡΟΝΟΥ

6.1 Εισαγωγή

Η διαμόρφωση εύρους παλμών (pulse width modulation) είναι κοινός τόπος στην οδήγηση διατάξεων ηλεκτρονικών ισχύος. Στα κεφάλαια 6 έως 8 παρουσιάζονται οι δυνατότητες του F28335 για τη δημιουργία παλμοσειρών ελέγχου διατάξεων ηλεκτρονικών ισχύος, όπως SPWM και SVPWM.

Στο τμήμα 6.2 παρουσιάζονται οι καταχωρητές οι οποίοι είναι υπεύθυνοι για τη συχνότητα και τη φάση των παραγόμενων PWM παλμών. Στη συνέχεια παρουσιάζονται δύο σημαντικά θέματα για το συγχρονισμό μεταξύ δύο καναλιών ePWM και την επιλογή συχνότητας. Στο τέλος του κεφαλαίου υπάρχει μία πολύ σημαντική εφαρμογή για τη διαμόρφωση της συχνότητας των ePWM παλμών.



Σχήμα 6.1: Δομικό διάγραμμα ePWM [1]

Στο σχήμα 6.1 απεικονίζεται ολοκληρωμένο το **δομικό διάγραμμα** του ePWM για τον F28335. Αν και δεν έχει αυτή τη στιγμή καμία ουσιαστική χρησιμότητα, παρουσιάζει εποπτικά τη διαδρομή που θα ακολουθήσουμε. Ο αναγνώστης θα κληθεί πολλές φορές στη διάρκεια του κεφαλαίου να ανατρέξει πίσω σε αυτό το διάγραμμα. Αυτή τη στιγμή είμαστε στο πρώτο βήμα για τη δημιουργία παλμών για τις τεχνικές SPWM και SVPWM.

6.2 Μονάδα Βάσης Χρόνου ePWM (Time Base Unit) [1]

Στον πυρήνα της ePWM μονάδας είναι ένας **χρονιστής(timer) 16-bit** (TBCTR-time base counter) ο οποίος αυξάνεται με βασική συχνότητα SYSCLKOUT. Η συχνότητα του πυρήνα (για τον F28335) είναι στα **150MHz**. Προφανώς αυτή η συχνότητα αύξησης του TBCTR δεν είναι σταθερή, ενώ και πολλές άλλες παράμετροι του μπορούν να μεταβληθούν ανάλογα με τις ανάγκες μας, όπως η **μορφή** του φορέα, ο **συγχρονισμός** κ.α. Στην ενότητα αυτή θα εξετάσουμε όλες αυτές τις εναλλακτικές που μας παρέχουν οι **Time Base καταχωρητές** της ePWM μονάδας.

Πίνακας 6.1: Συνοπτική περιγραφή των Time Base καταχωρητών

Όνομα	Περιγραφή	Δομή
TBCTR	Time Base Counter	EPwm*x*Regs.TBCTL
TBPRD	Time Base Period	EPwm*x*Regs.TBPRD
TBPHS	Time Base Phase	EPwm*x*Regs.TBPHS
TBCTL	Time Base Control	EPwm*x*Regs.TBCTL.all
TBSTS	Time Base Status	EPwm*x*Regs.TBSTS.all

Time Base Counter (TBCTR): έχει την **παρούσα τιμή** του τριγώνου του ePWM. Το μέγεθός του είναι 16 bit και η τιμή του κυμαίνεται από 0 έως TBPRD.

Time Base Period (TBPRD): καθορίζει ποιο θα είναι το **μέγιστο** μέχρι το οποίο θα μετρήσει ο TBCTR. Είναι προφανές ότι και αυτός ο καταχωρητής είναι 16-bit.

Time Base Phase (TBPHS): είναι η τιμή που φορτώνεται στον TBCTR όταν παράγεται ένα σήμα συγχρονισμού (SYNCIN). Και αυτός με τη σειρά του έχει μέγεθος 16bit και τιμή μεταξύ 0 και TBPRD.Περισσότερες λεπτομέρειες για το πώς λειτουργεί ο συγχρονισμός μεταξύ των διαφορετικών καναλιών ePWM στις επόμενες σελίδες.

6.2.1 Time Base Control Register (TBCTL)



Σχήμα 6.2: Least Significant Byte (LSB) του καταχωρητή ελέγχου του PWM

Ο TBCTL είναι ένας καταχωρητής 16bit (στο Σχήμα 6.2 βλέπουμε το κάτω μισό του) ο οποίος ελέγχει τη **μορφή**, τη **συχνότητα** και άλλες παραμέτρους του ePWM. Συγκεκριμένα:

CTRMODE

Αυτό το πεδίο επηρεάζει τον τρόπο με τον οποίον μεταβάλλεται ο TBCTR, ή, με ορολογία Ηλεκτρονικών Ισχύος, επηρεάζει τη **μορφή του φορέα** (Σχήμα 6.3). Ανάλογα με τη μορφή που επιλέγουμε μεταβάλλεται και ο τρόπος υπολογισμού της συχνότητας του PWM, όπως θα δούμε και παρακάτω. Η πιο συχνά χρησιμοποιούμενη μορφή είναι η **τριγωνική**. Όταν η τιμή είναι 11, ο TBCTR μένει σταθερός.



Σχήμα 6.3: Δυνατές μορφές φορέα ανάλογα με το CTRMODE

PHEN

Ενεργοποιεί τη δυνατότητα ολίσθησης φάσης μεταξύ διαφορετικών καναλιών ePWM. Αυτό υλοποιείται με τη φόρτωση της τιμής του καταχωρητή TBPHS στον TBCTR όταν δοθεί ο παλμός συγχρονισμού EPWMxSYNCI. Περισσότερα για το συγχρονισμού μεταξύ ePWM στη συνέχεια του κεφαλαίου.

PRDLD

Ενεργοποιεί (0) ή απενεργοποιεί (1) τη λειτουργία shadow στη φόρτωση νέας τιμής στον καταχωρητή TBPRD. Αν είναι ενεργοποιημένη (*PRDLD=0*) τότε η απόδοση νέας τιμής στον καταχωρητή TBPRD θα γίνει όταν TBCTR=0. Σε άλλη περίπτωση (*PRDLD=1*) θα γίνει αμέσως.

SYNCOSEL

Επιλέγεται πότε θα δοθεί ο παλμός συγχρονισμού SYNCO από ένα κανάλι ePWM. Οι δυνατές περιπτώσεις είναι

1. SYNCO=SYNCI (00)
2. Όταν TBCTR=0 (01)
3. Όταν TBCTR=CMPB (10)
4. Απενεργοποίηση του SYNCO (11)

SWFSYNC:

Αν αυτό το bit γίνει 1, θα αναγκάσει το ePWM κανάλι να παράγει παλμό SYNCO. Ένας άλλος τρόπος να εκφράσουμε το παραπάνω θα ήταν «η θετική ακμή αυτού του πεδίου παράγει ένα SYNCO παλμό».

ePWM Register TBCTL

Upper Register:



Σχήμα 6.4: Most Significant Byte (MSB) του καταχωρητή ελέγχου του PWM

CLKDIV και **HSPCLKDIV**

Παράγοντες με τους οποίους διαιρείται η συχνότητα SYSCLKOUT (150MHz). Εάν ο ρυθμός που αυξάνεται ο TBCTR είναι TBCLK, τότε:

$$TBCLK = \frac{SYSCLKOUT}{HSPCLKDIV \cdot CLKDIV} \tag{6.1}$$

Οπότε η συχνότητα του φορέα (στην περίπτωση πριονωτών-ασύμμετρων σημάτων) είναι ίση με:

$$F_{asym\_carrier} = \frac{TBCLK}{TBPRD + 1} \rightarrow TBPRD = \frac{SYSCLKOUT}{F_{asym\_carrier} \cdot HSPCLKDIV \cdot CLKDIV} - 1 \quad (6.2)$$

Στην περίπτωση του τριγωνικού-συμμετρικού φορέα έχουμε το μετρητή να μετράει 2 φορές μέχρι το TBPRD σε μία περίοδο φορέα. Οπότε η σχέση (6.1) μεταβάλλεται σε:

$$F_{sym\_carrier} = \frac{TBCLK}{TBPRD} \rightarrow TBPRD = \frac{SYSCLKOUT}{2 \cdot F_{sym\_carrier} \cdot HSPCLKDIV \cdot CLKDIV} \quad (6.3)$$

Με τους τύπους (6.1) και (6.2) μπορούμε να βρούμε τον **κατάλληλο συνδυασμό TBPRD,CLKDIV και HSPCLKDIV** για δεδομένη $\mathbf{F_{carrier}}$. Το γιατί παρατηρείται αυτή η διαφορά στη μορφή τους θα εξηγηθεί αργότερα.

PHSDIR

Κατεύθυνση μέτρησης μετά την αναγνώριση παλμού SYNCIN. Αν έχουμε απενεργοποιήσει τη το συγχρονισμό το bit αυτό είναι αδιάφορο.

FREE_SOFT

Ελέγχει την αλληλεπίδραση του DSC με το JTAG-Emulator. Αν ο κώδικας βρει ένα breakpoint καθορίζουμε τι θα γίνει με το συγκεκριμένο ePWM.

6.2.2 Time Base Status Register (TBSTS)



Σχήμα 6.5: Καταχωρητής Κατάστασης

Καταχωρητής που χρησιμοποιείται για να εξετάζουμε την κατάσταση του ePWM module. Η εξήγηση της λειτουργίας των πεδίων είναι τετριμμένη και για αυτό δεν παρουσιάζεται.

Το μόνο σημείο που αξίζει προσοχής, είναι ότι δύο από αυτά τα πεδία **μανδαλώνουν (latching fields).** Η τιμή τους, δηλαδή, **«κλειδώνει»,** όταν συμβεί το γεγονός που αναμένουν. Αν θέλουμε να εντοπίσουμε το επόμενο γεγονός πρέπει **«χειροκίνητα» να καθαρίσουμε το εν λόγω πεδίο.**

6.3 Προχωρημένα Θέματα [2]

6.3.1 Συγχρονισμός μεταξύ διαφορετικών καναλιών ePWM

Για την οδήγηση τριφασικών διατάξεων χρειάζονται τρία σήματα PWM συγχρονισμένα ή με διαφορά φάσης μεταξύ τους.



Σχήμα 6.6: Παράδειγμα συγχρονισμού φορέων

Υπάρχουν δύο σήματα hardware, SYNCI (synch in) και SYNCO(synch out), τα οποία μας επιτρέπουν να συγχρονίζουμε μεταξύ τους τα ePWM. Στο παραπάνω σχήμα (Σχήμα 6.6) υπάρχουν τρεις φορείς στους οποίους το ePWM1 είναι master και τα άλλα δύο (ePWM2/3) συγχρονίζονται σύμφωνα με αυτό (slaves). Χρήζει προσοχής ότι το ePWM3 συγχρονίζεται με το 1 μέσω του 2. Είναι δηλαδή **απαραίτητη η ύπαρξη Sync_out_2** για να μπορούμε να συγχρονίσουμε το ePWM3.



Σχήμα 6.7: Διάγραμμα χρονισμού
των ePWM για το F28335

Για το συγκεκριμένο DSP (F28335) το διάγραμμα συγχρονισμού είναι αυτό που φαίνεται στο Σχήμα 6.7.

Το οποίο πρακτικά σημαίνει ότι δε χρειάζεται να θέσουμε τα SYNCI/O των ePWM2/3 για να ορίσουμε διαφορά φάσης των ePWM1/4.

Ένα ζήτημα το οποίο δεν αναφέρεται στη βιβλιογραφία είναι ότι το PHEN είναι απενεργοποιημένο, τα 6 PWM δεν είναι ακριβώς συγχρονισμένα, αλλά παρουσιάζουν μία καθυστέρηση (Σχήμα 6.8).



Σχήμα 6.8: Η έξοδος του PWM 1 σε σχέση με τα PWM2-6 όταν δεν έχουμε εισάγει ρυθμίσεις συγχρονισμού των καναλιών PWM.

Για να γίνει ακριβής συγχρονισμός είναι **απαραίτητο** να ενεργοποιηθεί ο συγχρονισμός φάσης και να τεθεί η διαφορά φάσης ίση με το μηδέν. Αυτό υλοποιείται ως εξής με κώδικα:

```
EPwm1Regs.TBCTL.bit.SYNCOSEL =1;           //SYNCOUT when CTR=0
EPwm2Regs.TBCTL.bit.PHSEN =  1;
EPwm2Regs.TBPHS.half.TBPHS =0x0000;
EPwm2Regs.TBCTL.bit.PHSDIR =  1;           // count-up after sync
EPwm2Regs.TBCTL.bit.SYNCOSEL = 0;          // SYNCOUT= SYNCIN in order
//to drive ePWM 3. Those lines iterate for the rest ePWM modules
```

Το αποτέλεσμα φαίνονται στο Σχήμα 6.9. Είναι σημαντικό να παρατηρηθεί ότι το PWM1 θα έχει σταθερά **18,5ns διαφορά** με οποιοδήποτε άλλο PWM. Οπότε αν για την εφαρμογή μας τα 18ns είναι σημαντικό μέγεθος θα αναγκαστούμε να χρησιμοποιήσουμε ένα συνδυασμό των άλλων καναλιών(2-6) των οποίων η διαφορά είναι της τάξης του 1 ns.



Σχήμα 6.9: Αριστερά: PWM1(master)-PWMx(slave).
Δεξιά: PWMx(slave)-PWMy(slave)

6.3.2 Καθορισμός συχνότητας ePWM.



Σχήμα 6.10: Υπολογισμός συχνότητας ανάλογα με τη μορφή του φορέα

Στο σχήμα 6.10 επεξηγείται εποπτικά η διαφορά των τύπων (6.2) και (6.3). Ο **συντελεστής ½** για το **συμμετρικό φορέα** είναι προφανής καθώς όταν PRD=x σε up & down mode, η περίοδος είναι 2*PRD. Το **+1** που απαιτείται για την περίπτωση **ασύμμετρου φορέα** οφείλεται στο ότι ο CTR μετράει από το 0 έως **και** το PRD. Οπότε μετράει PRD+1 βήματα.

Υπενθυμίζεται ότι το TBCLK καθορίζεται από τον τύπο (6.1):

$$TBCLK = \frac{SYSCLKOUT}{HSPCLKDIV \cdot CLKDIV} \tag{6.4}$$

6.4 Εφαρμογή : Συνάρτηση επιλογής συχνότητας

Στο παράδειγμα αυτό θα δημιουργήσουμε μία συνάρτηση, η οποία έχει σαν είσοδο τη **συχνότητα σε Hz** και σαν έξοδο παράγει τις **παραμέτρους tb_prd, clkdiv και hspclkdiv** για το ePWM module.

Πυρήνας του προγράμματός θα είναι ο τύπος (6.3) καθώς θα χρησιμοποιηθεί τριγωνικός φορέας. Για να εξεταστούν όλοι οι πιθανοί συνδυασμοί clkdiv και hspclkdiv πρέπει να κατασκευάσουμε **δύο for loops** με το ένα να είναι φωλιασμένο στο πρώτο. Υφίσταται, βέβαια, η πιθανότητα να υπάρχουν δύο ή περισσότεροι συνδυασμοί που να παράγουν την επιθυμητή συχνότητα. Σε αυτή την περίπτωση επινοήθηκαν δύο κριτήρια βάσει των οποίων επιλέγουμε τη **βέλτιστη περίπτωση**.

Το **πρώτο κριτήριο** αφορά την **απόκλιση** της θεωρητικής από την πραγματική συχνότητα. Η απόκλιση αυτή οφείλεται στο ότι η διαίρεση (6.3) μπορεί να έχει αποτέλεσμα με δεκαδικά ψηφία, ωστόσο ο αριθμός TBPRD είναι αυστηρά ακέραιος. Οπότε με τη στρογγυλοποίηση δημιουργούνται και οι αποκλίσεις. Στη συγκεκριμένη υλοποίηση συγκρίνονται οι λύσεις γύρω από την **απόλυτη διαφορά** τους με την επιθυμητή συχνότητα.

Το **δεύτερο κριτήριο** αφορά την ανάλυση **της PWM παλμοσειράς**. Αυτή η παράμετρος είναι σημαντική κυρίως για τις υλοποιήσεις των SPWM και SVM τεχνικών. **Μεγαλύτερη ανάλυση έχει σαν αποτέλεσμα μεγαλύτερη ακρίβεια στη θέση των ακμών της παλμοσειράς**. Σε αυτό το κριτήριο έχει δοθεί μικρότερη προτεραιότητα αν και μέσω της μεταβλητής tolerance μπορούμε να «ανταλλάξουμε» ακρίβεια στη συχνότητα για μεγαλύτερη ανάλυση. Το αν θα χρησιμοποιηθεί αυτό το στοιχείο εξαρτάται από την εφαρμογή.

Πρέπει να τονιστεί ότι για εφαρμογές που είναι σημαντική η **σχετική θέση δύο κυματομορφών** (πχ SPWM) η συχνότητα πρέπει να είναι **απόλυτα ακριβής**, καθώς ακόμα και μια μικρή απόκλιση θα προκαλέσει (αθροιστικά) μεγάλα προβλήματα. Σε αυτήν την περίπτωση απορρίπτονται όλες τις λύσεις που έχουν απόκλιση, και από αυτές που μένουν προτιμάται εκείνη με το μεγαλύτερο TBPRD. Αυτό μπορεί να ρυθμιστεί με τη μεταβλητή tolerance της συνάρτησης. Για **tolerance=0.0** η συνάρτηση θα κρατήσει μόνο τις λύσεις με το μικρότερο σφάλμα (ιδανικά 0) και θα τις κατατάξει βάσει του TBPRD.

Στην επόμενη σελίδα ακολουθεί το **διάγραμμα ροής** της συνάρτησης:



Σχήμα 6.11: Διάγραμμα ροής συνάρτησης επιλογής παραμέτρων PWM

Ένα επιπλέον στοιχείο της συνάρτησης είναι τα **όρια ανάλυσης** που μπορούμε να θέσουμε. Έτσι αν η εφαρμογή έχει συγκεκριμένες προδιαγραφές σε ανάλυση πχ. τουλάχιστον TBPRD=1000, μπορεί να περαστεί σαν παράμετρος low της συνάρτησης και έτσι οι λύσεις με tbprd≤1000 θα απορριφθούν αυτόματα.

6.4.1 Συνάρτηση επιλογής παραμέτρων ePWM για δεδομένη συχνότητα (.c file)

```
#include "DSP2833x_Device.h"
#include "ePWM_function.h"
#include "math_function.h"
#include <stdlib.h>
#include <math.h>

char  scan_for_PWM_solutions(long  int  high,int  low,long  int  fpwm,
PWM_parameters* p)
{
    float pre_prd,tolerance=0.0;
    int i,j;
    int cdiv[8]={1,2,4,8,16,32,64,128};
    int hspdiv[8]={1,2,4,6,8,10,12,14};
    PWM_parameters new=PWM_parameters_def, old=PWM_parameters_def;

    for (i=0;i<=7;i++)
        {
        for (j=0;j<=7;j++)
            {
            pre_prd=
            (150000000.0)/2.0/(fpwm*1.0)/cdiv[i]*1.0/hspdiv[j]*1.0;
            if (pre_prd<=1.0)                          break;
            //You can't have a proper PWM with tb_prd=1
            if ((pre_prd<=high*1.0)&&(pre_prd>=low*1.0))
                {
                  new.clk_div=i;
                  new.hsp_clk_div=j;
                  new.tb_prd=round(pre_prd);
                  new.a_diff=fabs(fpwm*1.0-150000000.0/2.0
                  /(new.tb_prd*1.0)/cdiv[i]*1.0/hspdiv[j]*1.0);
                  if (new.a_diff<old.a_diff) old=new;
//1st criterion: better convergence to the target frequency
                  if
(new.a_diff<=(old.a_diff+tolerance)&&new.tb_prd>old.tb_prd) old=new;
//2nd criterion:  higher PWM resolution (higher PRD) with a tolerance
//to the accuracy. We opted for "1.0" tolerance for no particular
//reason. You can modify it based on your application specification
                }
            }
        }
*p=old; //cast the values of old struct to
return 0;
}
```

Συνάρτηση επιλογής παραμέτρων ePWM για δεδομένη συχνότητα (.h file)

```
#ifndef EPWM_FUNCTION_H
#define EPWM_FUNCTION_H

typedef struct {  int       clk_div;
                  int       hsp_clk_div;
                  int       tb_prd;
                  float     a_diff;
                } PWM_parameters;

#define PWM_parameters_def { 0, \
                             0, \
                             1000, \
                             1000.0, \
                           }

char scan_for_PWM_solutions(long int high,int low,long int fpwm,
PWM_parameters* p);
#endif
```

Παρατήρηση 1: Στο ePWM_function.h είναι δηλωμένο η δομή (struct) PWM_parameters. Για αυτό το λόγο συμπεριλαμβάνεται σαν header file στην αρχή του ePWM_function.c αρχείου.

Παρατήρηση 2: Είναι ενδιαφέρον ότι η βιβλιοθήκη math.h δεν περιέχει τη συνάρτηση στρογγυλοποίησης. Αν λοιπόν αποδοθεί η τιμή ενός float σε έναν int τότε αυτό που θα γίνει είναι να κοπούν από τον float τα δεκαδικά ψηφία (η διαδικασία αυτή είναι γνωστή ως **truncate**). Για να επιτευχθεί μεγαλύτερη ακρίβεια στην προσέγγιση της επιθυμητής συχνότητας χρησιμοποιούμε την παρακάτω συνάρτηση:

```
int round(double number)
//double precision floating-point type
{
    return (number >= 0) ? (int)(number + 0.5) : (int)(number - 0.5);
    //variable = condition ? value_if_true : value_if_false
}
```

Πχ.
number=4.2 → round(number)=4.2+0.5=4.7 → int_number=round(number)= 4.~~7~~ = 4
number=5.8 → round(number)=5.8+0.5=6.3 → int_number=round(number)= 6.~~3~~ = 6

Παρατήρηση 3: Θα μπορούσε να γίνει βελτιστοποίηση της παραπάνω συνάρτησης με τη χρήση IQ-math. Δεν κρίθηκε απαραίτητο γιατί, για τη συγκεκριμένη λειτουργία δεν απαιτείται άμεση απόκριση (real time)

6.4.2 Κώδικας main με σχόλια

```
#include "DSP2833x_Device.h"
#include "ePWM_function.h"
#include <stdlib.h>

// external function prototypes
extern void InitSysCtrl(void);
extern void InitPieCtrl(void);
extern void InitPieVectTable(void);
// Prototype statements for functions found within this file.
void Gpio_select(void);
void Setup_ePWM(void);
//Global Variables
PWM_parameters par=PWM_parameters_def;
int counter=0,setup_aknowledged=0;
long int freq_carrier=2345,freq_carrier_old,freq_carrier_new;

void main(void)
{
    InitSysCtrl();  // Basic Core Init from DSP2833x_SysCtrl.c
    EALLOW;
    SysCtrlRegs.WDCR= 0x00AF;   // Re-enable the watchdog
    EDIS;  // 0x00AF  to NOT disable the Watchdog, Prescaler = 64
    DINT;               // Disable all interrupts
    Gpio_select();      // enable ePWM1A pin
    Setup_ePWM();       // init of ePWM1A
    InitPieCtrl();// basic setup of PIE table
    InitPieVectTable(); // default ISR's in PIE
    while(1)
    {
            EALLOW;
            SysCtrlRegs.WDKEY = 0x55;   // service WD #1
            SysCtrlRegs.WDKEY = 0xAA;   // service WD #2
            EDIS;

            counter++; // segment that enables real-time
            freq_carrier_new=freq_carrier;
            if (counter==3000)
                {counter=0;
                freq_carrier_old=freq_carrier;
                setup_aknowledged=0; }
            if
(freq_carrier_old!=freq_carrier_new&&setup_aknowledged==0)
                {setup_aknowledged=1;
                Setup_ePWM();}

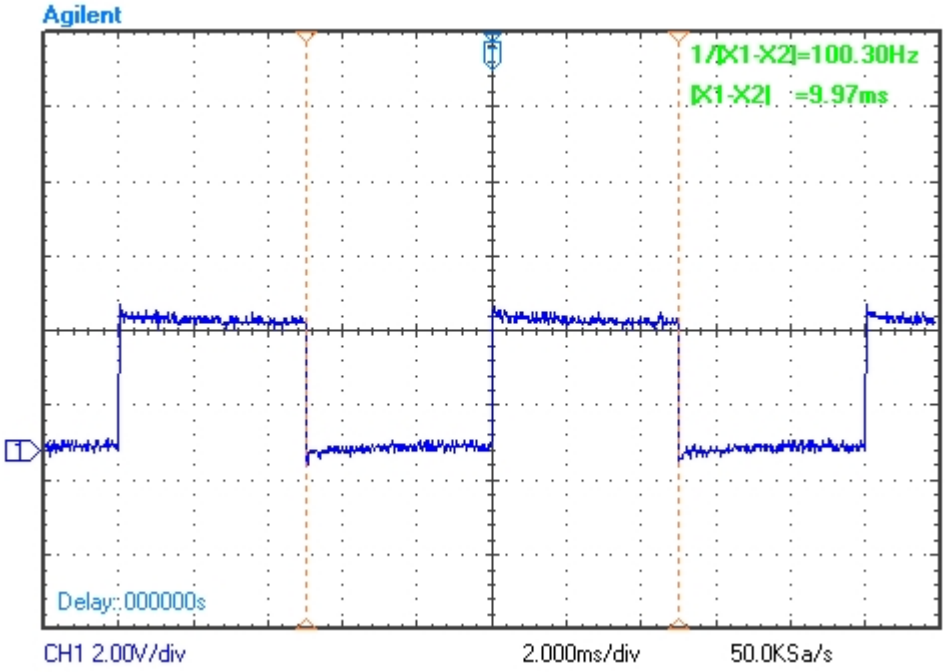
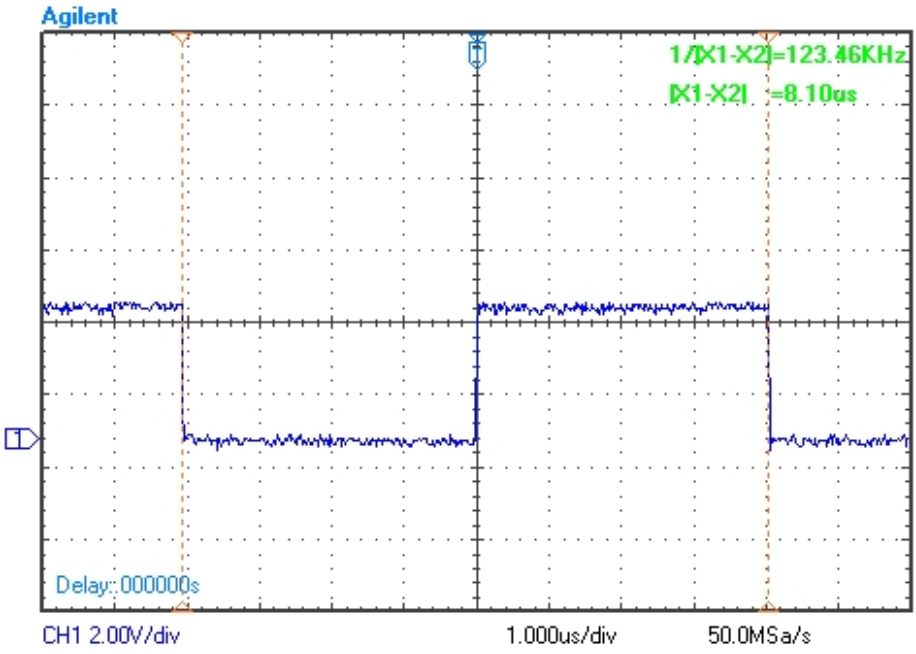
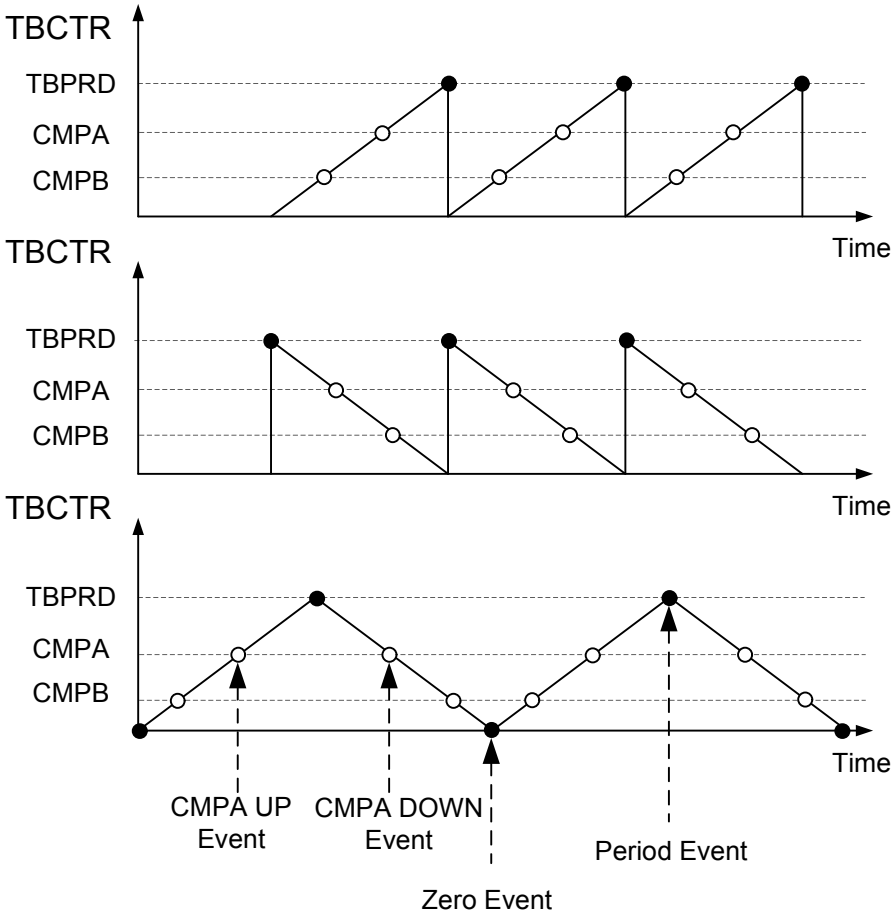
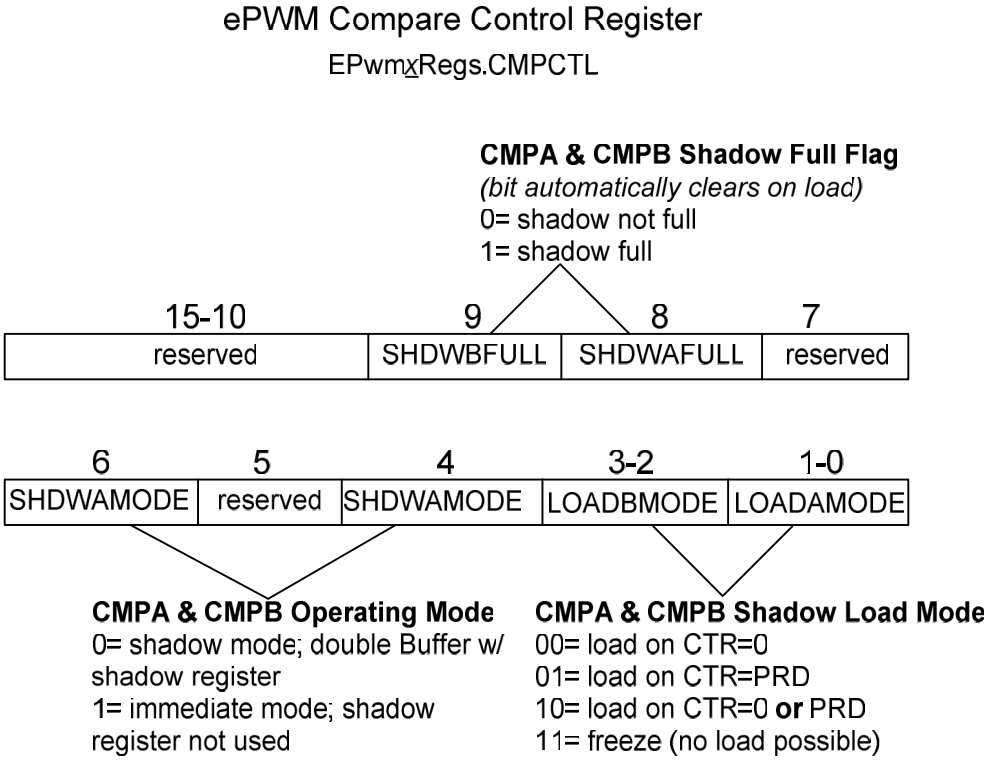
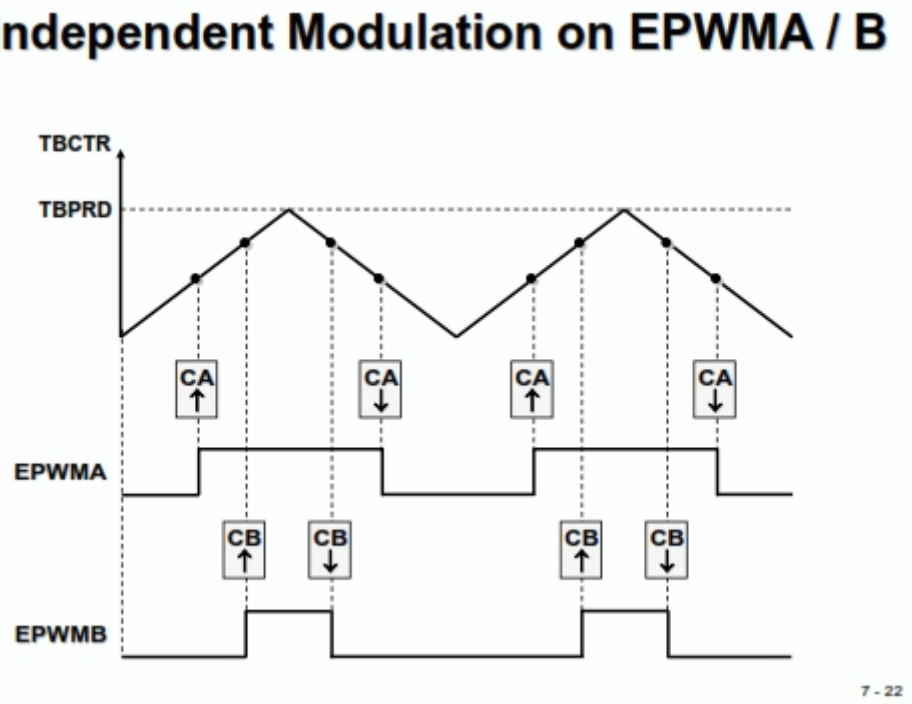
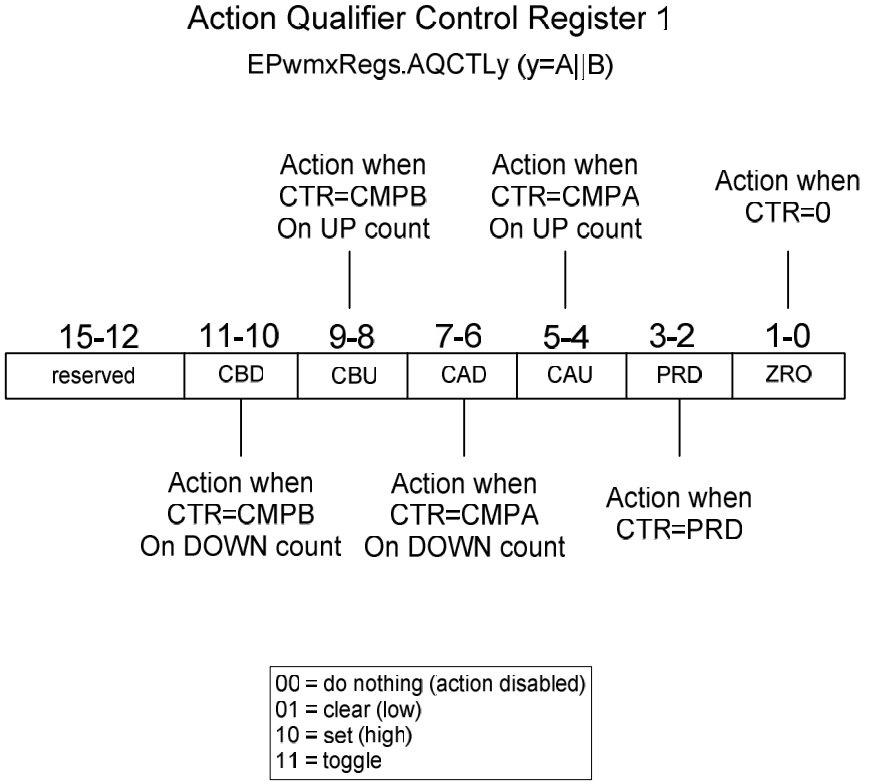
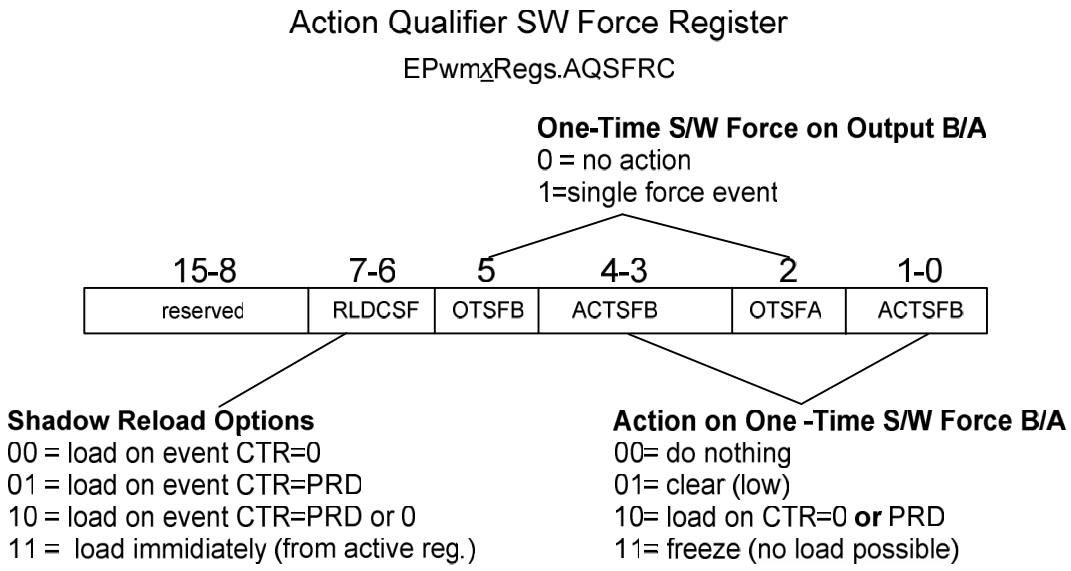
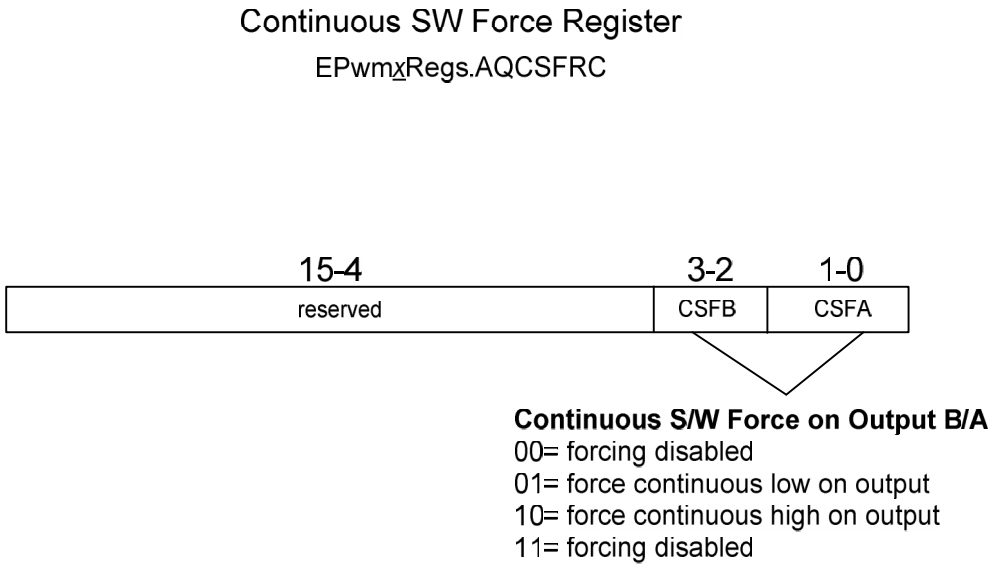
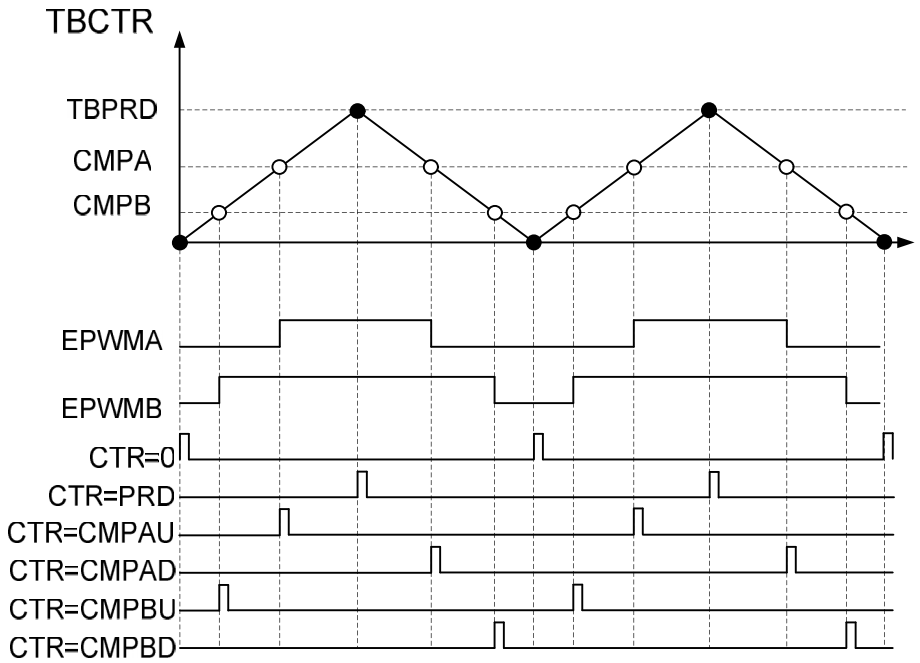
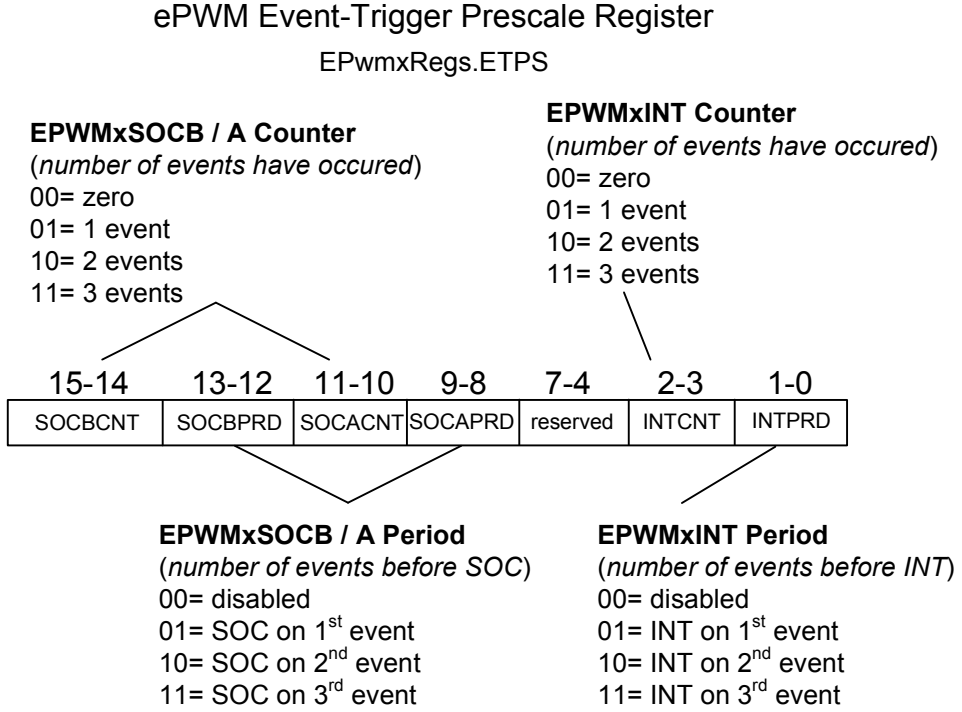
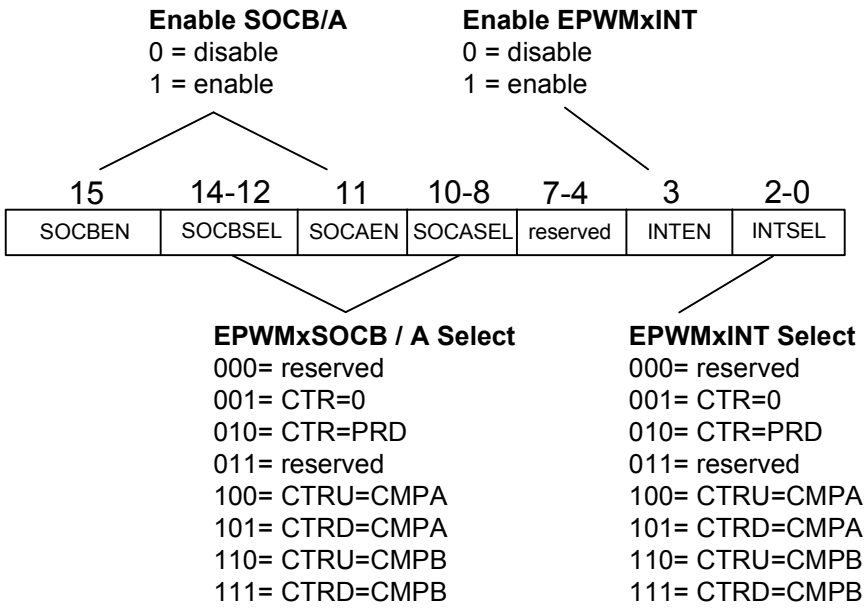
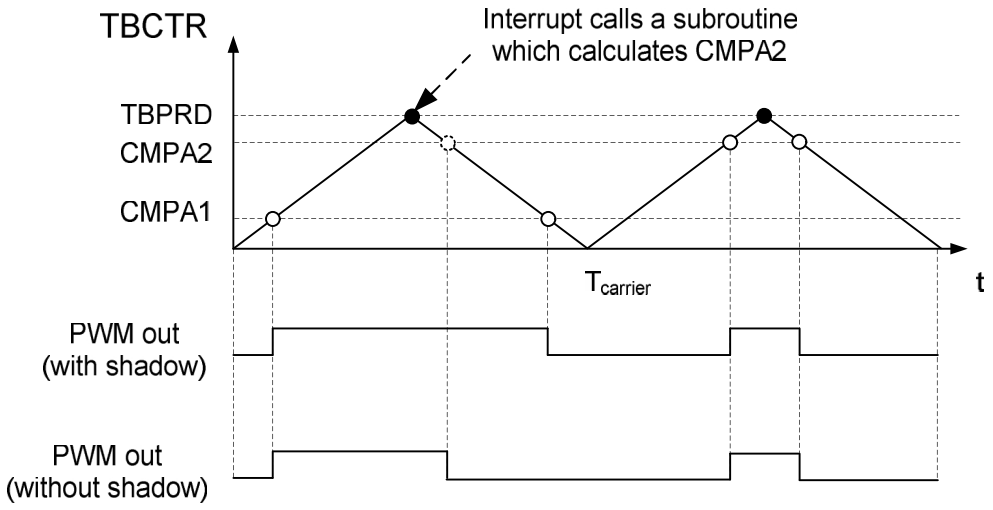
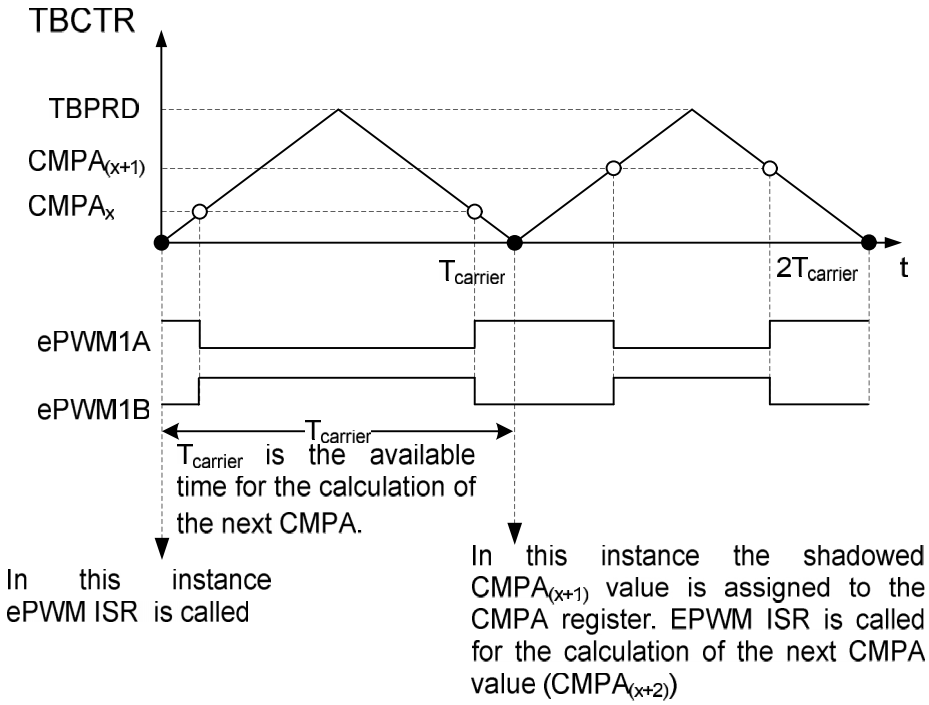
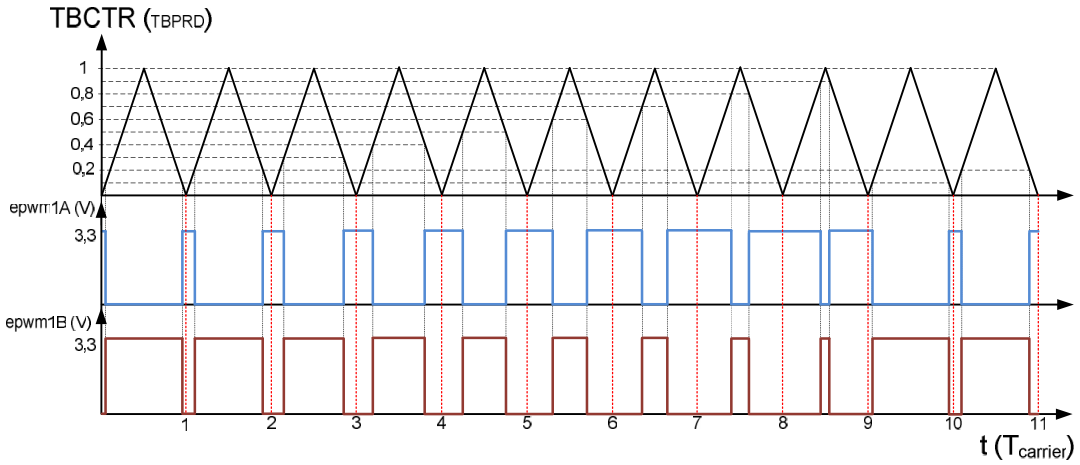
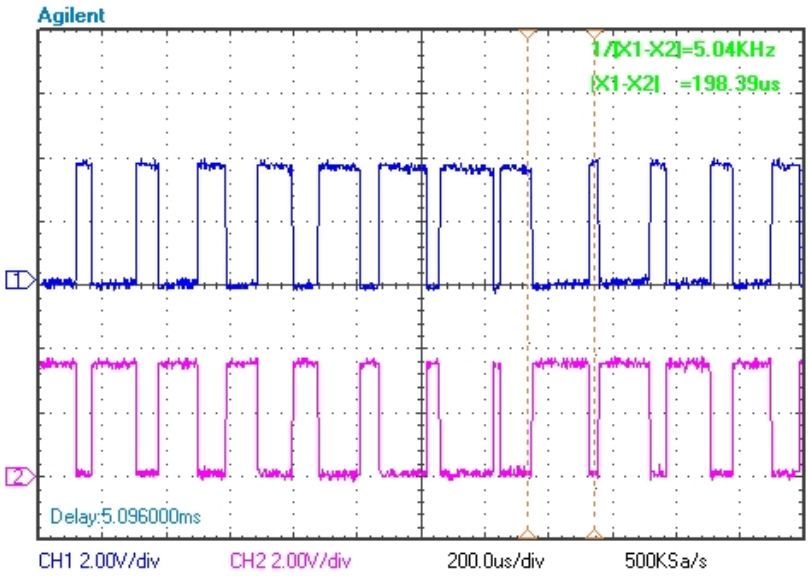

    }
}
```

```
void Gpio_select(void)
{
EALLOW;
GpioCtrlRegs.GPAMUX1.all = 0; // GPIO15 ... GPIO0 = General Purpose
I/O
GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1; // ePWM1A active

GpioCtrlRegs.GPAMUX2.all = 0;             //   GPIO31   ...   GPIO16
GpioCtrlRegs.GPBMUX1.all = 0;             //   GPIO47   ...   GPIO32
GpioCtrlRegs.GPBMUX2.all = 0;             //   GPIO63   ...   GPIO48
GpioCtrlRegs.GPCMUX1.all = 0;             //   GPIO79   ...   GPIO64
GpioCtrlRegs.GPCMUX2.all = 0;             // GPIO87 ... GPIO80
GpioCtrlRegs.GPADIR.all = 0;
GpioCtrlRegs.GPBDIR.all = 0;        // GPIO63-32 as inputs
GpioCtrlRegs.GPCDIR.all = 0;        // GPIO87-64 as inputs
EDIS;
}

void Setup_ePWM(void)
{
    scan_for_PWM_solutions(63000,10,freq_carrier,&par);
    EPwm1Regs.TBCTL.bit.SYNCOSEL =1;  //SYNCIN when CTR=0

    EPwm1Regs.TBCTL.bit.CLKDIV = par.clk_div;
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = par.hsp_clk_div;
    EPwm1Regs.TBCTL.bit.CTRMODE = 2;    // up - down mode
    EPwm1Regs.AQCTLA.all = 0x0006;      // ZRO = set, PRD = clear

    EPwm1Regs.TBPRD = par.tb_prd;

}
```

Παρατήρηση 1: Το κομμάτι του κώδικα που βρίσκεται μέσα στο ατέρμονο βρόχο while εξυπηρετεί στην **απόδοση νέων τιμών συχνότητας** παράλληλα με την εκτέλεση του προγράμματος. Επιλέγουμε **View→Watch Window** και στο παράθυρο που ανοίγει πληκτρολογούμε το όνομα της μεταβλητής (**`freq_carrier`**). Με διπλό κλικ πάνω στο πεδίο τιμών μπορούμε να μεταβάλλουμε την τιμή της συχνότητας.

6.4.3 Πειραματικά Αποτελέσματα

Συχνότητα 100Hz: Για τη συγκεκριμένη περίπτωση βρέθηκε ακριβής λύση. Η μικρή διαφορά που παρατηρείται στη μέτρηση του παλμογράφου οφείλεται στην ανάλυση του άξονα του χρόνου.



Σχήμα 6.12: PWM συχνότητας 100Hz

Name	Value
freq_carrier	100
par	{...}
clk_div	1
hsp_clk_div	6
tb_prd	31250
a_diff	0.0

Σχήμα 6.13: Παράμετροι ePWM για f=100Hz

Συχνότητα 123452Hz: Επιλέχθηκε μία τυχαία συχνότητα ώστε να παρουσιαστεί η δυνατότητα του αλγόριθμου να διαχειριστεί «δύσκολες» περιπτώσεις. Το σφάλμα, ωστόσο, είναι αρκετά μεγάλο σαν απόλυτο μέγεθος, αλλά ποσοστιαία είναι μικρό.



Σχήμα 6.14:PWM συχνότητας 123,452KHz

Name	Value
freq_carrier	123452
par	{...}
clk_div	0
hsp_clk_div	0
tb_prd	608
a_diff	96.74219

Σχήμα 6.15: Παράμετροι ePWM για f=123,452KHz

6.5 Βιβλιογραφία

[1] «TMS320F28335 Enhanced Pulse width modulation module» available at http://www.ti.com /

[2] «TMS320F28335 Data Manual» available at http://www.ti.com /

ΚΕΦΑΛΑΙΟ 7

EPWM MODULE - ΜΟΝΑΔΑ ΣΥΓΚΡΙΣΗΣ ΚΑΙ ΔΙΑΚΟΠΗΣ ΔΙΕΡΓΑΣΙΩΝ

7.1 Εισαγωγή

Μέχρι τώρα έχουμε αναφερθεί στον τρόπο με τον οποίο παράγουμε παλμοσειρές **σταθερού πλάτους** και **διαφόρων συχνοτήτων**. Ωστόσο, αυτό δεν αρκεί καθώς οι ανάγκες των ηλεκτρονικών ισχύος επιτάσσουν και **συνεχή μεταβολή στο εύρος των παλμών**. Σε αυτό το κεφάλαιο παρέχονται στον αναγνώστη τα εφόδια για να μπορεί να κατασκευάζει παλμοσειρές μεταβλητού εύρους (για την εκάστοτε συχνότητα). Στο **τμήμα 7.2** περιγράφονται οι **καταχωρητές που διαμορφώνουν το συντελεστή χρησιμοποίησης των παλμών** που παράγει η ePWM μονάδα.

Στις περισσότερες εφαρμογές με ανάδραση, ο υπολογισμός του συντελεστή χρησιμοποίησης για κάθε παλμό, υπολογίζεται σχεδόν ταυτόχρονα με την παραγωγή του (on the fly). Για αυτό το λόγο υπάρχουν πολλαπλά **γεγονότα** κατά τον κύκλο του ePWM, στα οποία μπορούμε να δημιουργήσουμε **διακοπές**, ώστε να υπολογιστεί το επιθυμητό εύρος παλμών. Η παράθεση αυτών των γεγονότων, καθώς και οι καταχωρητές που καθορίζουν την επιλογή τους γίνεται στο **τμήμα 7.3** .

Στη συνέχεια, (**τμήμα 7.4**) παρουσιάζεται με μεγαλύτερη λεπτομέρεια μία μέθοδος ώστε να γίνει πιο στιβαρό το πρόγραμμα υπολογισμού και απόδοσης των νέων τιμών εύρους παλμών. Η μέθοδος αυτή ονομάζεται **σκιώδης λειτουργία** (**shadow mode**).

Τέλος, στο **τμήμα 7.5** βρίσκεται η εφαρμογή του κεφαλαίου, στην οποία υλοποιείται μία απλή παλμοσειρά συχνότητας **5KHz**. Η παλμοσειρά αυτή αποτελείται από **εννιά(9) παλμούς**, με **συντελεστή χρησιμοποίησης από 10% έως 90%**. Αν και η παλμοσειρά είναι περιοδική, ο υπολογισμός και η απόδοση της τιμής του εύρους για κάθε παλμό γίνεται σε πραγματικό χρόνο, μέσω των μεθόδων που αναλύθηκαν κατά την πορεία του παρόντος κεφαλαίου.

7.2 Μονάδα σύγκρισης [1]

7.2.1 Καταχωρητές ελέγχου (ePWM Compare Control Register)

Είναι το κομμάτι του ePWM module που ελέγχει το **συντελεστή χρησιμοποίησης των παλμών**.Η λειτουργία του βασίζεται σε δύο καταχωρητές (Compare Register A και B- CMPA και CMPB στο εξής) οι οποίοι δημιουργούν «γεγονότα» στα οποία μπορούμε να πάρουμε αποφάσεις (να ανοίξουμε ή να κλείσουμε έναν διακόπτη). Στο σχήμα 7.1 φαίνεται η αλληλεπίδραση των καταχωρητών CMPA|B με τους διαφορετικούς τύπους φορέων. Σημειώνεται ότι στην

τριγωνική μορφή φορέα δημιουργούνται δύο γεγονότα για κάθε καταχωρητή (CMPA count up και CMPA count down).



Σχήμα 7.1: Τα διαθέσιμα γεγονότα διακοπής για τους τρεις τύπους φορέων

Στον πίνακα 7.1 παραθέτουμε τους καταχωρητές οι οποίοι χρησιμοποιούνται στη συγκεκριμένη ενότητα:

Πίνακας 7.1: Καταχωρητές ελέγχου μονάδας σύγκρισης

Όνομα	Περιγραφή	Δομή
CMPCTL	Compare Control	EPwmxRegs.CMPCTL.all
CMPA	Compare A	EPwmxRegs.CMPA
CMPB	Compare B	EPwmxRegs.CMPB

Οι καταχωρητές *EPwmxRegs.CMPA* και *EPwmxRegs.CMPB* είναι αριθμοί 16bit. Ο καταχωρητής *EPwmxRegs.CMPCTL* είναι υπεύθυνος για το πώς μεταβάλλεται η τιμή αυτών των καταχωρητών. Αυτός ο καταχωρητής παρουσιάζεται λεπτομερώς στο σχήμα 7.2 .



Σχήμα 7.2: Καταχωρητής Ελέγχου της μονάδας Σύγκρισης

- **SHDWA|BFULL (CMPA|B shadow full flag)**

Το πεδίο αυτό είναι μόνο για ανάγνωση. Εάν η λειτουργία shadow είναι πλήρης (1) αντιγράφεται η τιμή του CMPA|B στο προσκήνιο. Το bit καθαρίζεται αυτόματα.

- **SHDWA|BMODE (CMPA|B operating mode)**

Ενεργοποίηση (0) ή απενεργοποίηση (1) της shadow φόρτωσης των καταχωρητών CMPA|CMPB. Επιγραμματικά, όταν αυτή η λειτουργία είναι ενεργοποιημένη, η νέα τιμή του CMPA|B φορτώνεται σε προκαθορισμένα σημεία του κύκλου λειτουργίας του PWM. Αλλιώς, όταν η λειτουργία αυτή είναι απενεργοποιημένη, φορτώνεται άμεσα. Περισσότερες λεπτομέρειες υπάρχουν στο τμήμα 7.4 .

- **LOADA|BMODE (CMPA|B Shadow load mode)**

Αφορά την περίπτωση στην οποία η shadow λειτουργία είναι ενεργοποιημένη. Συγκεκριμένα αφορά τη στιγμή κατά την οποία φορτώνεται η νέα τιμή των καταχωρητών CMPA|B.

7.2.2 Καταχωρητής δράσης (ePWM Action Qualifier Unit)

Το δεύτερο κομμάτι της δημιουργίας παλμοσειρών είναι ο καθορισμός ενεργειών στο εκάστοτε γεγονός. Οι πιθανές ενέργειες για κάθε γεγονός είναι:

- Θέση γραμμής ePWM σε υψηλό λογικό επίπεδο (3,3V)
- Θέση γραμμής ePWM σε χαμηλό λογικό επίπεδο (0V)
- Αντιστροφή της προηγούμενης κατάστασης (toggle)
- Καμία ενέργεια

Υπενθυμίζεται ότι υπάρχουν 6 πιθανά γεγονότα για τριγωνικής μορφής φορέα (Μηδέν, CMPA-up, CMPB-up, ημι-περίοδος, CMPA-down, CMPB-down).

Στο παρακάτω σχήμα βλέπουμε ένα παράδειγμα χρησιμοποίησης του Action Qualifier (AQ) με ξεχωριστό τρόπο για τη γραμμή A και B του ePWM1. Επίσης, στο φορέα σημειώνονται όλα τα πιθανά γεγονότα που μπορούν να χρησιμοποιηθούν για τη μεταβολή της εξόδου.



Σχήμα 7.3: Ανεξάρτητη Διαμόρφωση καναλιού A και B [1]

Στο συγκεκριμένο παράδειγμα η γραμμή 1A έχει τέτοιες ρυθμίσεις στον AQ ώστε να γίνεται 1 στο γεγονός CMPA UP και μηδέν στο CMPA DOWN. Η γραμμή 1B είναι ρυθμισμένη να έχει την ίδια συμπεριφορά, αλλά χρησιμοποιεί σαν αναφορά το CMPB.

Στον παρακάτω πίνακα παραθέτουμε τους καταχωρητές οι οποίοι χρησιμοποιούνται για τη συγκεκριμένη μονάδα.

Πίνακας 7.2: Καταχωρητές της μονάδας δράσης

Όνομα	Περιγραφή	Δομή
AQCTLA	AQ Control Output A	EPwmxRegs.AQCTLA.all
AQCTLB	AQ Control Output B	EPwmxRegs.AQCTLB.all
AQSFRC	AQ S/W Force	EPwmxRegs.AQSFRC.all
AQCSFRC	AQ Cont. S/W Force	EPwmxRegs.AQCSFRC.all

Στη συνέχεια παρουσιάζονται λεπτομερώς οι εν λόγω καταχωρητές:



Σχήμα 7.4: Καταχωρητής ελέγχου του Action Qualifier

Ο καταχωρητής τους παραπάνω σχήματος καθορίζει τις ενέργειες που θα γίνουν στην έξοδο για κάθε γεγονός. Υπάρχουν 6 γεγονότα διαθέσιμα και για κάθε ξεχωριστό γεγονός υπάρχουν τέσσερις πιθανές ενέργειες. Για τις εφαρμογές που απασχολούν τα ηλεκτρονικά ισχύος αξιοποιούμε συνήθως δύο από τα πεδία αυτά.

Για την υλοποίηση του σχήματος 7.3 απαιτούνται οι παρακάτω γραμμές κώδικα:

```
EPwmxRegs.AQCTLA.bit.CAU=2; //set on CMPA UP
EPwmxRegs.AQCTLA.bit.CAD=1; //clear on CMPA DOWN
EPwmxRegs.AQCTLB.bit.CBU=2; //set on CMPB UP
EPwmxRegs.AQCTLB.bit.CBD=1; //clear on CMPB DOWN
```



Σχήμα 7.5: Καταχωρητής εξαναγκασμού τιμής του Action qualifier

Ο καταχωρητής AQSFRC χρησιμοποιείται για να εξαναγκάσει μία γραμμή εξόδου σε μία συγκεκριμένη κατάσταση. Ο όρος άπαξ (one time) σημαίνει ότι η τιμή αυτή παραμένει για μία περίοδο του PWM.



Σχήμα 7.6: Καταχωρητής συνεχούς εξαναγκασμού τιμής του Action qualifier

Στον αντίποδα ο καταχωρητής AQCSFRC εξαναγκάζει διαρκώς (Continuous) τη γραμμή εξόδου σε συγκεκριμένο λογικό επίπεδο.

7.3 Πηγές διακοπών [1]

Το τελευταίο στοιχείο που απαιτείται για να έχουμε "on-the-fly" παραγωγή ψηφιακών σημάτων (όπως είναι το SPWM) είναι ένα σύστημα που να επιτρέπει την πρόκληση διακοπών σε επίπεδο PWM περιόδου. Έτσι ο συντελεστής χρησιμοποίησης του κάθε παλμού υπολογίζεται στην προηγούμενη περίοδο. Αυτός είναι ο μόνος τρόπος για να εισάγουμε **ανάδραση στον έλεγχο**, γεγονός το οποίο είναι ζητούμενο στις περισσότερες εφαρμογές ηλεκτρονικών ισχύος.



Σχήμα 7.7: Πιθανά σημεία διακοπών

Στο σχήμα 7.7 φαίνονται τα πιθανά σημεία στα οποία μπορεί να προκληθεί μία διακοπή. Να σημειώσουμε ότι οι γραμμές ePWM μπορούν να χρησιμοποιηθούν και για την πυροδότηση του Analog-to-Digital module του F28335. Στην επεξήγηση των καταχωρητών θα συναντήσουμε τον όρο SoC (start of conversion) το οποίο σημαίνει «αρχή μετατροπής» και αναφέρεται στην μετατροπή μία αναλογικής τάσης (0-3,3V) σε ψηφιακή στάθμη. Οι καταχωρητές που θα μας απασχολήσουν στη συγκεκριμένη ενότητα είναι οι εξής:

Πίνακας 7.3: Καταχωρητές Διακοπών ePWM

Όνομα	Περιγραφή	Δομή
ETSEL	Event-Trigger Selection	EPwmxRegs.ETSEL.all
ETPS	Event-Trigger Pre-Scale	EPwmxRegs.ETPS.all
ETFLG	Event-Trigger Flag	EPwmxRegs.ETFLG.all
ETCLR	Event-Trigger Clear	EPwmxRegs.ETCLR.all
ETFRC	Event-Trigger Force	EPwmxRegs.ETFRC.all

Οι καταχωρητές ETFLG, ETCLR, ETFRC αφορούν τη σημαία που ενεργοποιείται όταν καλείται η διακοπή ePWM. Αυτή τη σημαία μπορούμε να την διαβάσουμε (ETFLG), να την καθαρίσουμε (ETCLR) ή να εξαναγκάσουμε σε ένα λογικό επίπεδο (ETFRC).



Σχήμα 7.8: Καταχωρητής event prescale

- **EWPMxSOCB|A Counter**

Αφορά την Analog-to-Digital λειτουργία. Πεδίο μόνο για ανάγνωση, το οποίο καταγράφει τον αριθμό γεγονότων από την τελευταία Αρχή Μετατροπής (SoC).

- **EWPMxSOCB|A Period**

Αφορά την Analog-to-Digital λειτουργία. Η αρχή μετατροπής (SoC) ενεργοποιείται κάθε 1, 2 ή 3 γεγονότα.

- **EWPMxINT Counter**

Πεδίο μόνο για ανάγνωση, το οποίο καταγράφει τον αριθμό γεγονότων από την τελευταία διακοπή.

- **EWPMxINT Period**

Πεδίο το οποίο δίνει τη δυνατότητα στον προγραμματιστή να ενεργοποιήσει διακοπή κάθε 1, 2 ή 3 γεγονότα.

ePWM Event-Trigger Selection Register

EPwmxRegs.ETSEL



Σχήμα 7.9: Καταχωρητής επιλογής διακοπής

- **SOCB|AEN (Enable SoC A or B)**

Αφορά την Analog-to-Digital λειτουργία. Ενεργοποιεί τη λειτουργία Analog to Digital με πυροδότηση μέσω του ePWM καναλιού.

- **SOCB|ASEL (Select SoC A or B)**

Αφορά την Analog-to-Digital λειτουργία. Επιλογή συγκεκριμένου γεγονότος που πυροδοτεί την αρχή μετατροπής (SoC).

- **INTEN (EWPMxINT Enable)**

Ενεργοποίηση της δυνατότητας διακοπών σε γεγονότα της PWM περιόδου.

- **INTSEL (EWPMxINT Select)**

Επιλογή γεγονότος που πυροδοτεί τη διακοπή.

7.4 Προχωρημένα θέματα [2]

- **Επεξήγηση λειτουργίας shadow**

Η λειτουργία shadow είναι ένας ασφαλής τρόπος να φορτώνουμε στους CMP καταχωρητές τη νέα τους τιμή. Για αυτό το λόγο υπάρχει ένα σύμπλεγμα καταχωρητών buffer, οι οποίοι **αποθηκεύουν στο παρασκήνιο** τη νέα τιμή του σημείου τομής του φορέα με την αναφορά, και την αποδίδουν στους CMP καταχωρητές στον επόμενο κύκλο.

Τα **οφέλη** αυτής της λειτουργίας είναι ότι:

1. Μειώνει την αναγκαιότητα για πρόσβαση στον καταχωρητή CMP, μειώνοντας έτσι την κατανάλωση επεξεργαστικής ισχύος.
2. Επιτρέπει στον προγραμματιστή να καθορίσει ένα αυστηρό πλαίσιο μέσα στο οποίο μπορεί να γίνει ο υπολογισμός της νέας τιμής CMP.

Το **μειονέκτημα** είναι ότι η όποια διόρθωση εκτελεί το πρόγραμμα λόγω ανάδρασης καθυστερεί για χρόνο ανάλογο της περιόδου PWM ($T_{carrier}$). Αυτός ο χρόνος στις περισσότερες εφαρμογές είναι αμελητέος.

Προς επέκταση του δεύτερου πλεονεκτήματος και τι προβλήματα μπορεί να προκαλέσει η απουσία του, θα μελετήσουμε το παρακάτω παράδειγμα:



Σχήμα 7.10: Περίπτωση στην οποία η έλλειψη shadow λειτουργίας προκαλεί ελαττωματική έξοδο

Για το σχήμα 7.10 έχει επιλεγεί η διακοπή για τον υπολογισμό του επόμενου CMPA να γίνεται στη μέση της περιόδου του τριγωνικού φορέα. Στόχος είναι η νέα τιμή να εφαρμόζεται στην **επόμενη περίοδο PWM** και να δημιουργήσει στην έξοδο την πρώτη παλμοσειρά (with shadow). Εάν χρησιμοποιήσουμε τη shadow λειτουργία για τη φόρτωση των νέων τιμών στους CMPA καταχωρητές έχουμε σαν αποτέλεσμα **πάντα** την ίδια παλμοσειρά.

Στην περίπτωση που απενεργοποιήσουμε τη shadow λειτουργία φόρτωσης, υπάρχει **πιθανότητα** να εμφανιστεί την έξοδο η δεύτερη παλμοσειρά (without shadow). Αυτό το γεγονός θα παρατηρηθεί εάν ο χρόνος εκτέλεσης της ΡΕΔ είναι πολύ μικρότερος από την περίοδο PWM.

Ακόμα και αν έχει ληφθεί μέριμνα κάτι τέτοιο να μη συμβαίνει, είναι **επισφαλές** να στηρίξουμε την ορθή λειτουργία του προγράμματος σε κάτι τόσο

απρόβλεπτο, όπως ο χρόνος εκτέλεσης μιας υπό-ρουτίνας. **Συμπερασματικά, είναι πιο ασφαλές να έχουμε την shadow λειτουργία φόρτωσης πάντα ενεργή**.

7.5 Εφαρμογή : Παραγωγή PWM παλμοσειράς και της συμπληρωματικής της

Σε αυτή την εφαρμογή θα δημιουργήσουμε κώδικα, ο οποίος παράγει μία παλμοσειρά PWM, στην οποία ο συντελεστής χρησιμοποίησης των παλμών μεταβάλλεται από 10% έως 90%, με βήμα 10%. Παρότι στα πλαίσια της συγκεκριμένης εφαρμογής, η παλμοσειρά παραμένει αμετάβλητη (δεν εξαρτάται από κάποια ανάδραση) το ίδιο πρόγραμμα θα μπορούσε να χρησιμοποιηθεί για την παραγωγή **δυναμικών παλμοσειρών**, όπως θα δούμε παρακάτω στην εφαρμογή του SPWM.

Στη συγκεκριμένη εφαρμογή χρησιμοποιούνται δύο γραμμές ePWM. Η ePWM1B (GPIO1) ακολουθεί τις ενέργειες του σχήματος 7.4. Δηλαδή, εξαναγκάζεται σε υψηλό δυναμικό (3,3V) για το γεγονός CMPA-up και σε χαμηλό (0V) για CPMA-down. Η γραμμή ePWM1A ακολουθεί την αντίστροφη διαδικασία, χρησιμοποιώντας και αυτή σαν αναφορά την τιμή του καταχωρητή CPMA. Η παραπάνω διαδικασία, καθώς και οι διεργασίες που εκτελεί η CPU για τον καθορισμό των σημείων σύγκρισης (CMPA) περιγράφονται στο σχήμα 7.11.



Σχήμα 7.11: Σχηματική αναπαράσταση της δυναμικής μεθόδου υπολογισμού των σημείων τομής

Στην αρχή κάθε περιόδου του φορέα καλείται η Ρουτίνα Εξυπηρέτησης Διακοπής (ΡΕΔ) για το PWM1. Η επιλογή να καλείται στην αρχή κάθε περιόδου μας

παρέχει το **μέγιστο δυνατό χρόνο υπολογισμού** ο οποίος είναι ίσος με $\mathbf{T_{carrier}}$. Ακόμα και αν δεν απαιτείται κάτι τέτοιο στην παρούσα εφαρμογή, γενικότερα είναι καλή πρακτική και θα φανεί ιδιαίτερα χρήσιμο στην περίπτωση **εισαγωγής αναδράσεων** που απαιτούν αρκετό υπολογιστικό χρόνο. Σε κάθε περίοδο της αναφοράς (T_r) περιέχονται 9 παλμοί, οι οποίοι απεικονίζονται στο σχήμα 7.12.



Σχήμα 7.12: Θεωρητική απεικόνιση της εξόδου ePWM1A (GPIO0) και ePWM1B (GPIO1) για διάστημα $11*T_c$ ($T_r=9*T_c$)

Υπενθυμίζουμε ότι ο καταχωρητής CMPA είναι shadowed το οποίο σημαίνει ότι η ανάθεση νέας τιμής γίνεται στην αρχή κάθε νέας περιόδου του φορέα.

7.5.1 Σημαντικά χωρία κώδικα

- ```
  EPwm1Regs.AQCTLA.all = 0x0060;
  // CMPA-UP=>clearA, CMPA-DOWN=>setA
  EPwm1Regs.AQCTLB.all = 0x0090;
  // CMPA-UP=>setB ,  CMPA-DOWN=>clearB
  EPwm1Regs.CMPCTL.all = 0x0000;
  // 0x0000 shadow mode ON, load on CTR=0
  ```

Σημαντικές ρυθμίσεις για το ePWM. Παρατηρούμε ότι δε χρειάζεται η χρησιμοποίηση του καταχωρητή CMPB για την παραγωγή της δεύτερης παλμοσειράς. Η αρχικοποίηση του καταχωρητή CMPCTL δεν είναι απαραίτητη (καθώς αυτή είναι η default τιμή του) απλά υπάρχει για αυξημένη ασφάλεια.

- ```
  index_percentage=_IQ(((index)*1.0)/10.0);
  EPwm1Regs.CMPA.half.CMPA= _IQmpy(index_percentage,
  EPwm1Regs.TBPRD);
  ```

Αυτές οι γραμμές περιέχονται στην ΡΕΔ που καλείται όταν έχουμε διακοπή PWM. Στη συγκεκριμένη εφαρμογή η ΡΕΔ καλείται σε κάθε αρχή της περιόδου T_c. Ο αριθμός index αυξάνεται από 1 έως 9 ώστε να σχηματιστεί στην έξοδο το επιθυμητή

παλμοσειρά. Στην περίπτωση που υπήρχε κάποια ανάδραση αυτό είναι το σημείο στο οποίο θα αξιοποιούταν.

```
•       counter++;
              freq_carrier_new=freq_carrier;
              if (counter==3000)
                  {counter=0;
                  freq_carrier_old=freq_carrier;
                  setup_aknowledged=0; }
              if
(freq_carrier_old!=freq_carrier_new&&setup_aknowledged==0)
                  {setup_aknowledged=1;
                  Setup_ePWM();}
```

Το κομμάτι αυτό βρίσκεται εντός του ατέρμονα βρόχου που τρέχει στην κύρια συνάρτηση του προγράμματος. Ανά 3000 εκτελέσεις του βρόχου ελέγχει τη μεταβλητή **`freq_carrier_new`** η οποία μεταβάλλεται χειροκίνητα μέσα από το παράθυρο μεταβλητών (Watch Window) του Code Composer Studio, όταν το πρόγραμμα τρέχει με αποσφαλμάτωση πραγματικού χρόνου (Real Time Debugging). Λεπτομέρειες για τη μέθοδο αυτή βρίσκονται στο παράρτημα.

Στη συνέχεια ελέγχεται αν η τιμή της **`freq_carrier_new`** είναι διαφορετική από την προηγούμενη, η οποία είναι αποθηκευμένη στη μεταβλητή **`freq_carrier_old.`** Αν είναι ίδια, η εκτέλεση του προγράμματος συνεχίζει κανονικά, σε αντίθετη περίπτωση καλείται η **`Setup_ePWM()`** η οποία μεταβάλλει την συχνότητα φορέα του ePWM.

Συμπερασματικά, αυτός ο κώδικας μας δίνει τη δυνατότητα να μεταβάλλουμε τη συχνότητα φορέα του ePWM **κατά τη διάρκεια εκτέλεσης του προγράμματος**, γεγονός το οποία συμβάλλει στην εξοικονόμηση χρόνου κατά τη διενέργεια δοκιμών.

7.5.2 Πειραματικά Αποτελέσματα

Στο σχήμα 7.13 υπάρχει η έξοδος των GPIO0 (ePWM1A - μπλε παλμοσειρά) και GPIO1 (ePWM1B - ροζ παλμοσειρά), όπως αυτή απεικονίζεται στον παλμογράφο του εργαστηρίου. Παρατηρούμε ότι η μορφή τους είναι όμοια με τη θεωρητική απεικόνιση στο σχήμα 7.12. Το διάστημα [X1-X2] αποτελεί το χρονικό διάστημα μεταξύ των πορτοκαλί διακεκομμένων γραμμών, οι οποίες συμπεριλαμβάνουν την έξοδο των GPIO για την πρώτη περίοδο του τριγωνικού φορέα (T_c=200μs).



Σχήμα 7.13: Η έξοδος των GPIO0 (ePWM1A-μπλε) και GPIO1 (ePWM1B-ροζ). Σε μια τυπική διάταξη Ηλεκτρονικών Ισχύος οι δύο παλμοσειρές θα ήλεγχαν την αγωγή δύο ημιαγωγικών διακοπτών της ίδιας ημιγέφυρας.

7.5.3 Κώδικας με σχόλια

```
#include "DSP2833x_Device.h"
#include "ePWM_function.h"
#include "IQmathLib.h"
#include <math.h>
#include <stdlib.h>

// external function prototypes
extern void InitSysCtrl(void);
extern void InitPieCtrl(void);
extern void InitPieVectTable(void);
// Prototype statements for functions found within this file.
void Gpio_select(void);
void Setup_ePWM(void);
interrupt void ePWM1A_isr(void);

//Global Variables
PWM_parameters par=PWM_parameters_def;
unsigned int counter=0,setup_aknowledged=0,index=0,CMPA_buff[10];
long int freq_carrier=1000,freq_carrier_old,freq_carrier_new;
_iq index_percentage=0.0;
//###########################################################################
//                       main code
//###########################################################################
void main(void)
{
    InitSysCtrl();  // Basic Core Init from DSP2833x_SysCtrl.c

    EALLOW;
    SysCtrlRegs.WDCR= 0x00AF;   // Re-enable the watchdog
    EDIS;
    DINT;               // Disable all interrupts

    Gpio_select();      // GPIO9, GPIO11, GPIO34 and GPIO49 as output
```

```
                        // to 4 LEDs at Peripheral Explorer)
    Setup_ePWM();       // init of ePWM1A
    EPwm1Regs.CMPA.half.CMPA=EPwm1Regs.TBPRD/2;
    InitPieCtrl();        // basic setup of PIE table; from
                          // DSP2833x_PieCtrl.c

    InitPieVectTable(); // default ISR's in PIE

    EALLOW;
    PieVectTable.EPWM1_INT=&ePWM1A_isr;
    EDIS;
    PieCtrlRegs.PIEIER3.bit.INTx1=1; //ePWM1 IER activation
        IER |=4;
    EINT; //enable global interrupt switch
    ERTM;
    while(1)
    {
            EALLOW;
            SysCtrlRegs.WDKEY = 0x55;   // service WD #1
            SysCtrlRegs.WDKEY = 0xAA;   // service WD #2
            EDIS;

            counter++;
            freq_carrier_new=freq_carrier;
            if (counter==3000)
                {counter=0;
                freq_carrier_old=freq_carrier;
                setup_aknowledged=0; }
            if
(freq_carrier_old!=freq_carrier_new&&setup_aknowledged==0)
                {setup_aknowledged=1;
                Setup_ePWM();}

//With the following command we are able to produce a pulse series
//that has a positive edge when CTR=0 and a negative edge when
//CTR=PRD. This is needed for syncing the oscilloscope to the
//individual pulses. It serves no other purpose
    GpioDataRegs.GPADAT.bit.GPIO9 = EPwm2Regs.TBSTS.bit.SYNCI;
    }
}


void Gpio_select(void)
{
    EALLOW;
    GpioCtrlRegs.GPAMUX1.all = 0
      GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1;     // ePWM1A active
      GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 1;     // ePWM1B active
    GpioCtrlRegs.GPAMUX2.all = 0;
    GpioCtrlRegs.GPBMUX1.all = 0;
    GpioCtrlRegs.GPBMUX2.all = 0;
    GpioCtrlRegs.GPCMUX1.all = 0;
    GpioCtrlRegs.GPCMUX2.all = 0;
    GpioCtrlRegs.GPADIR.all = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO9 = 1;  // GPIO09 as output
    GpioCtrlRegs.GPBDIR.all = 0;        // GPIO63-32 as inputs
    GpioCtrlRegs.GPCDIR.all = 0;        // GPIO87-64 as inputs
    EDIS;
}

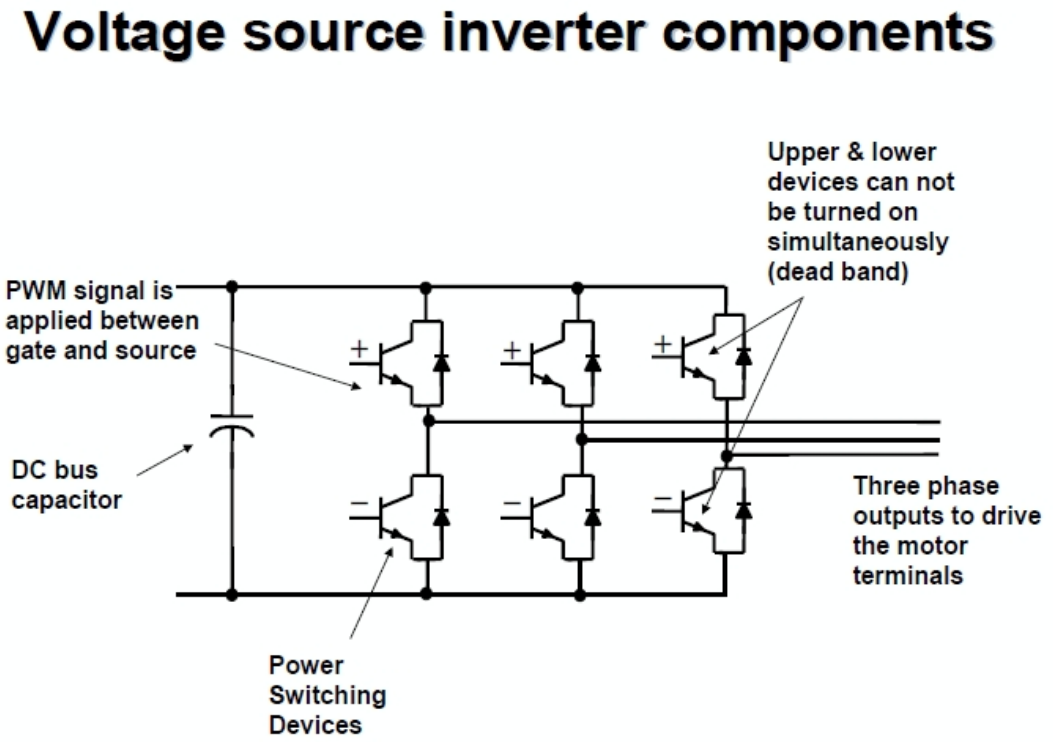
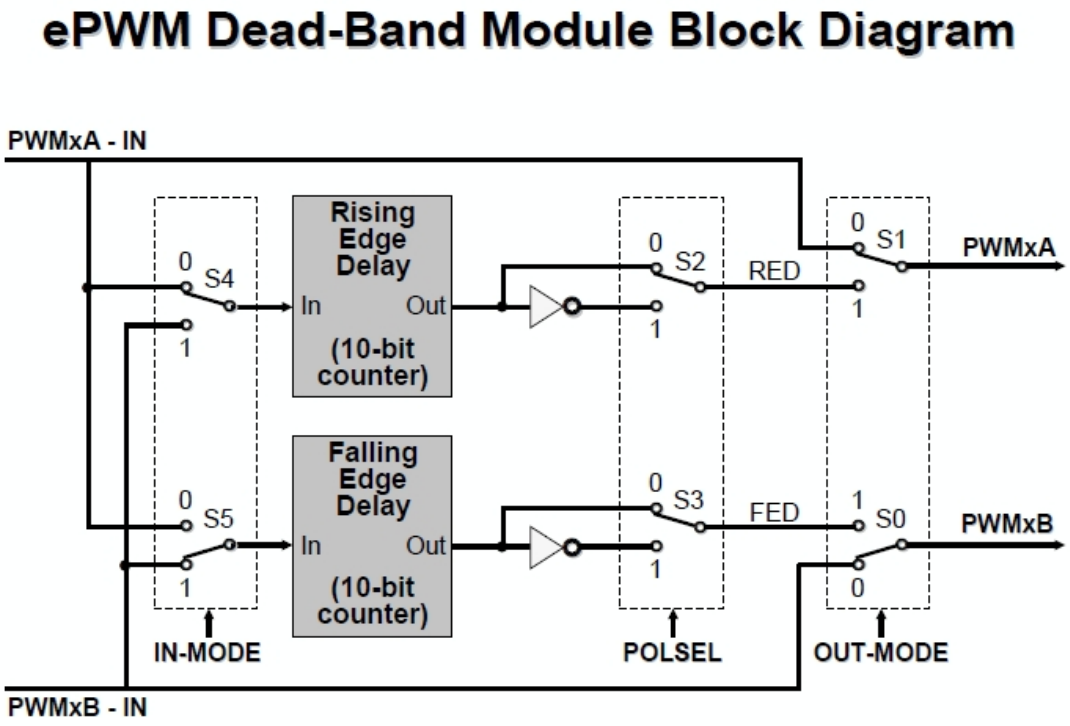
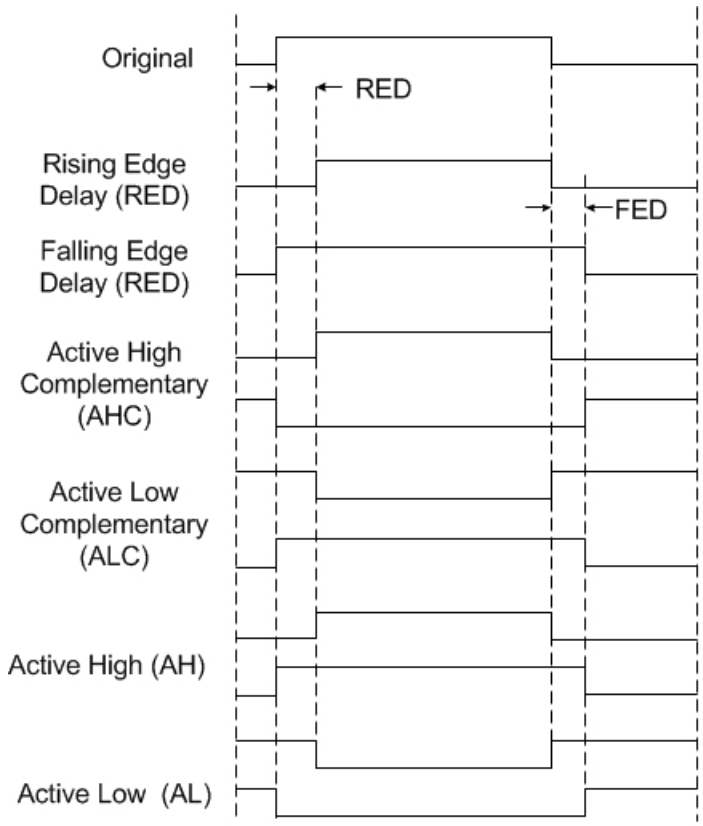
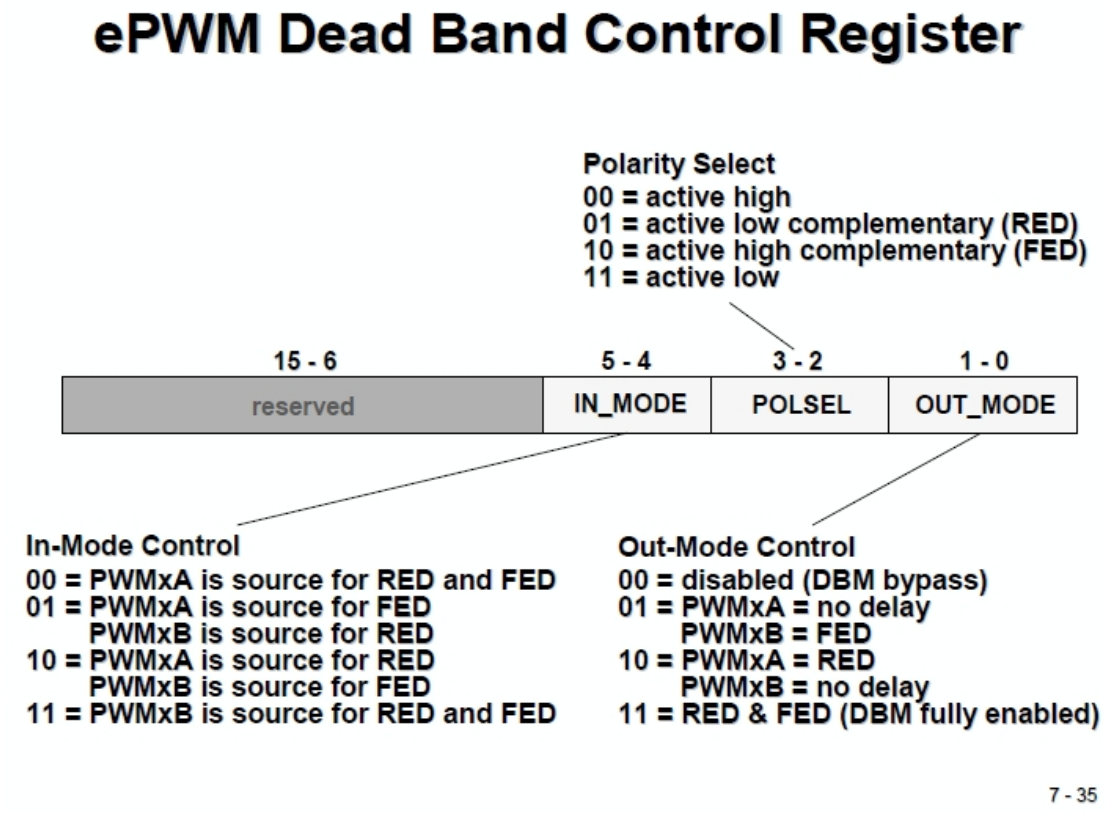
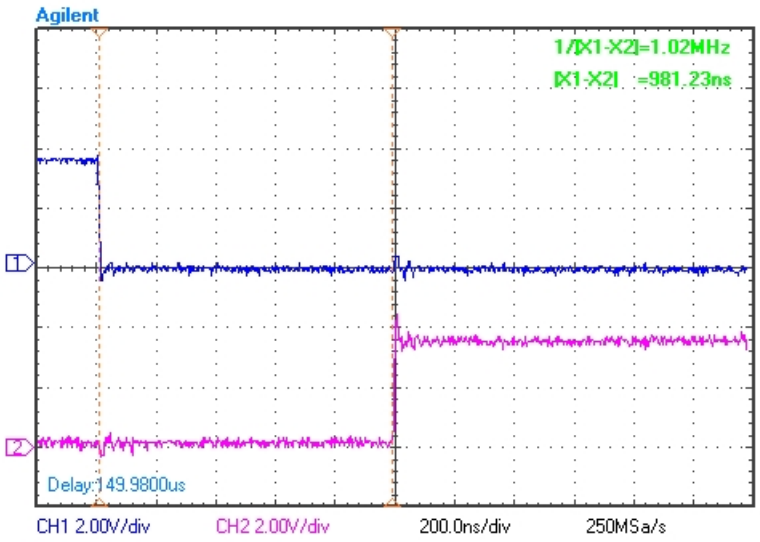
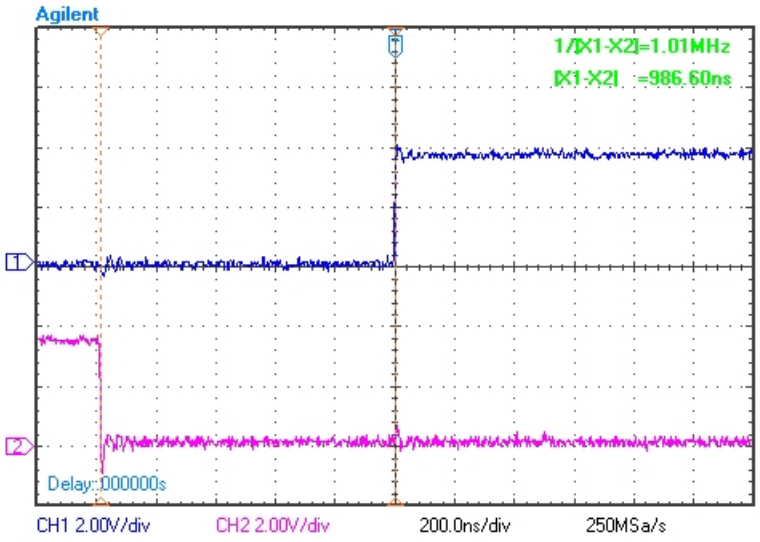
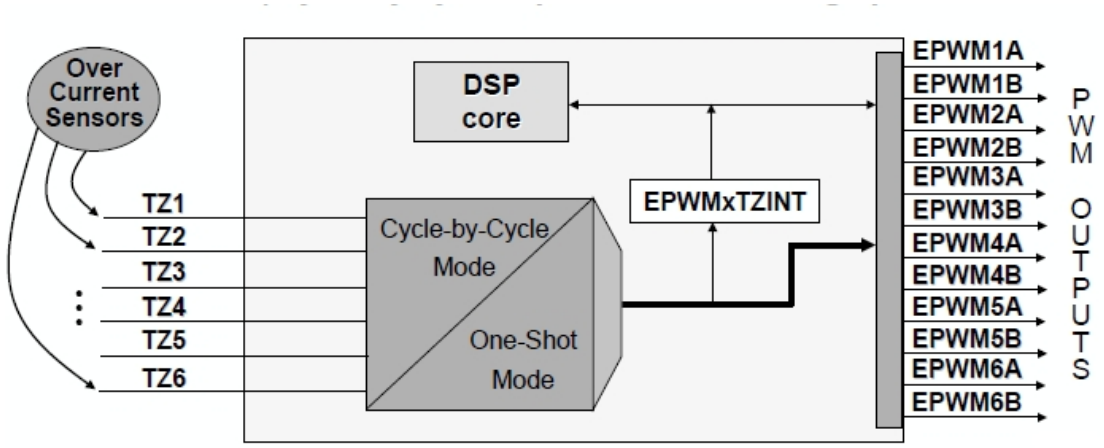
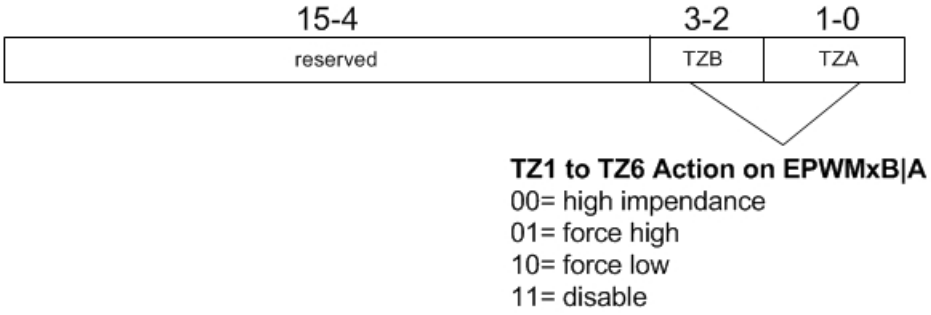
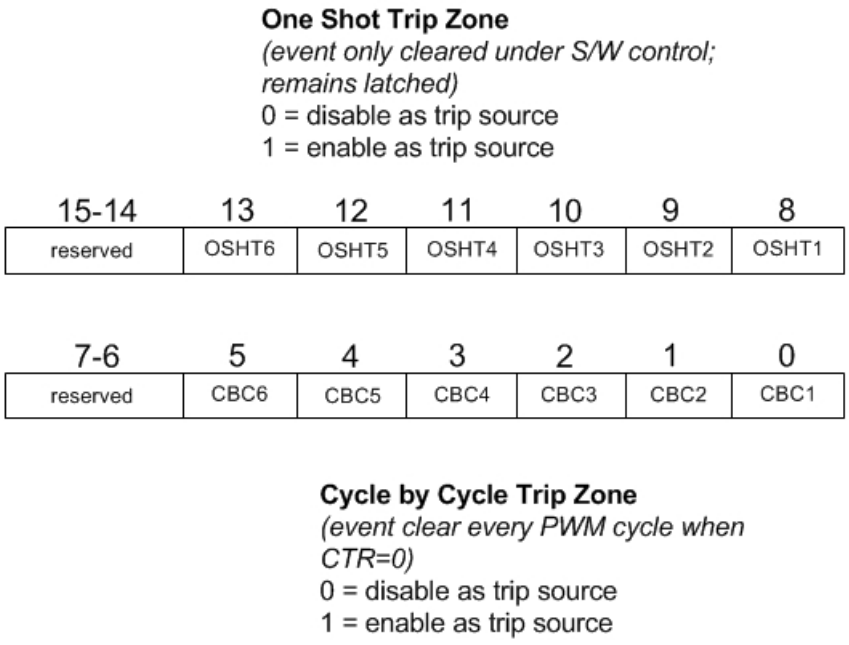
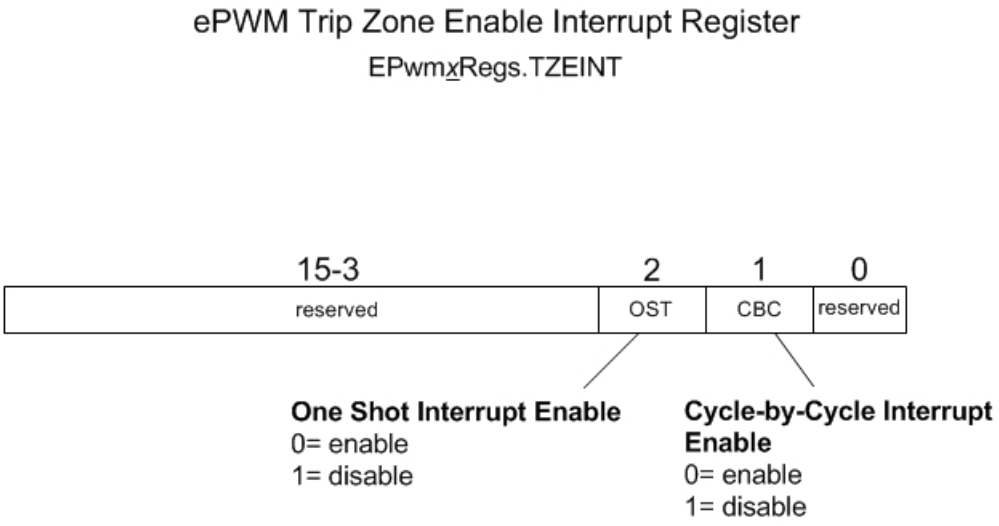
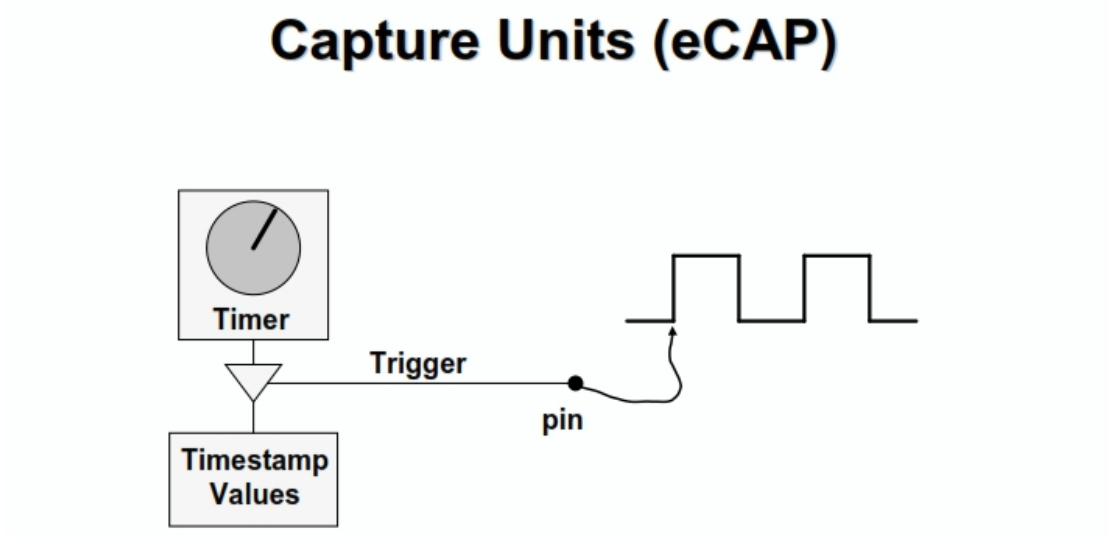
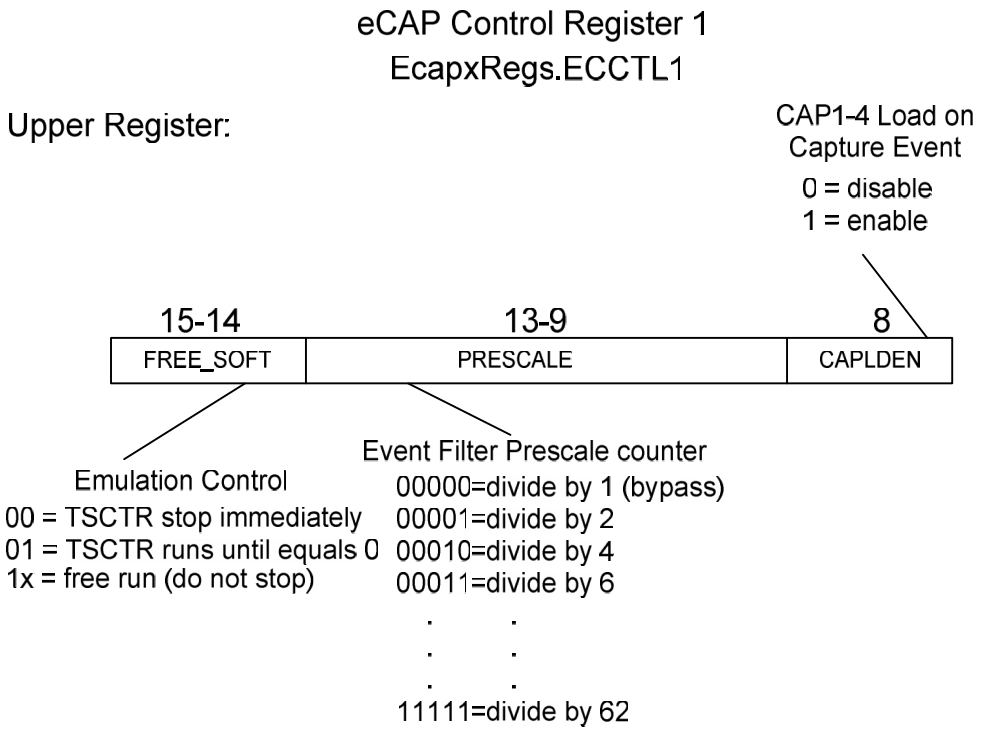
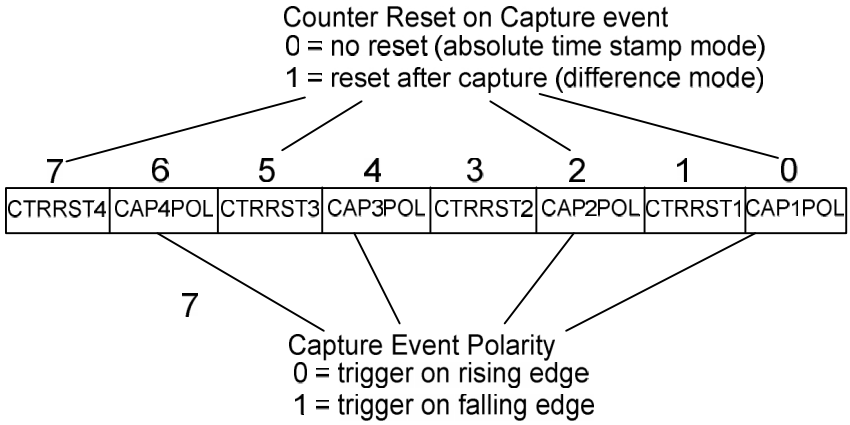
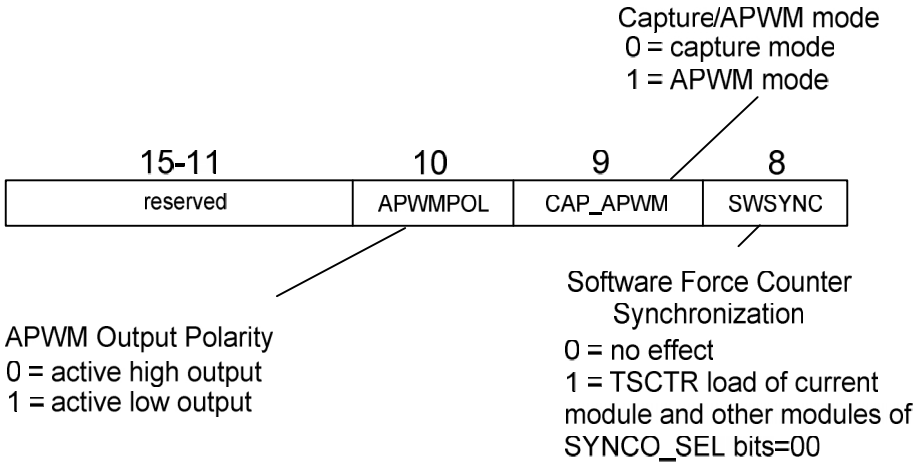
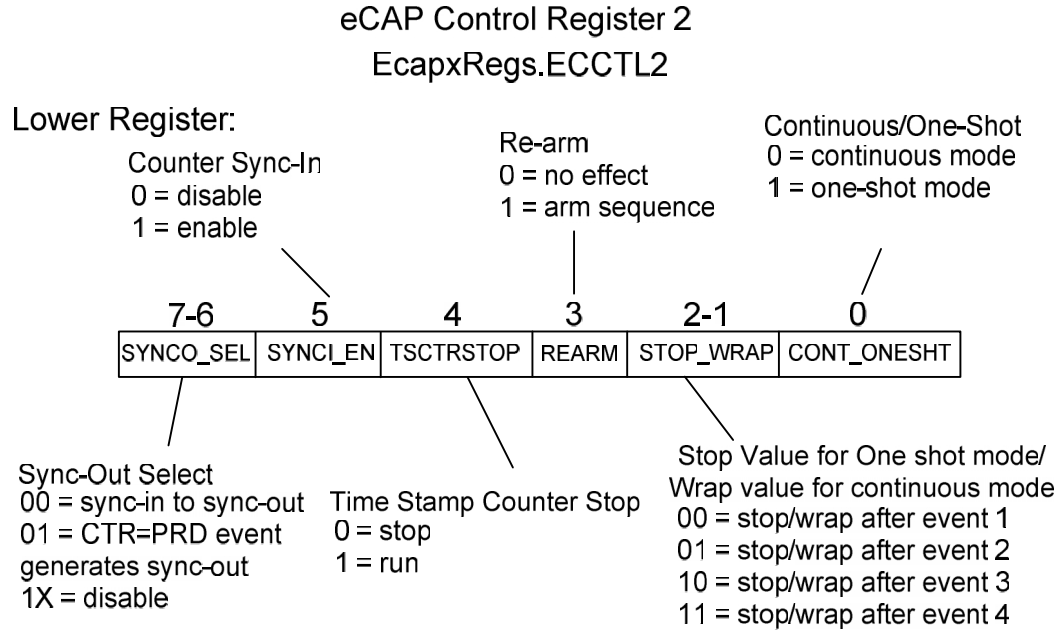
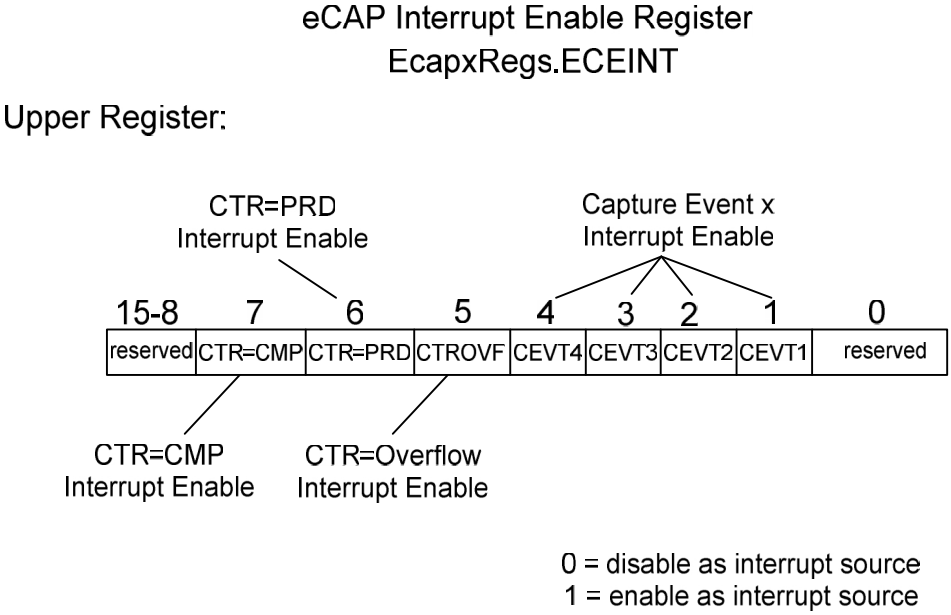
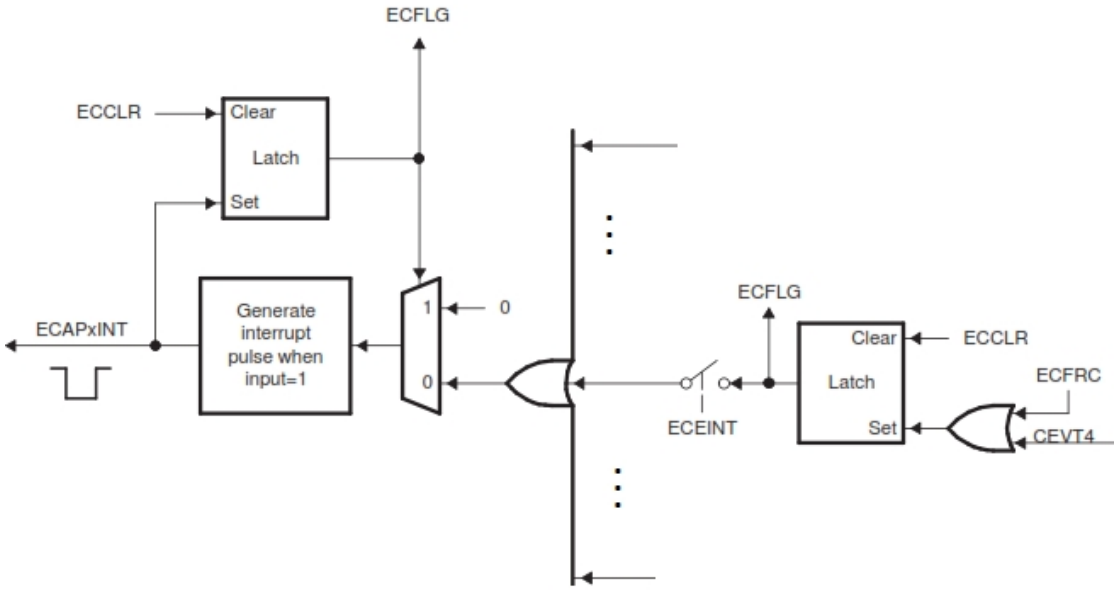
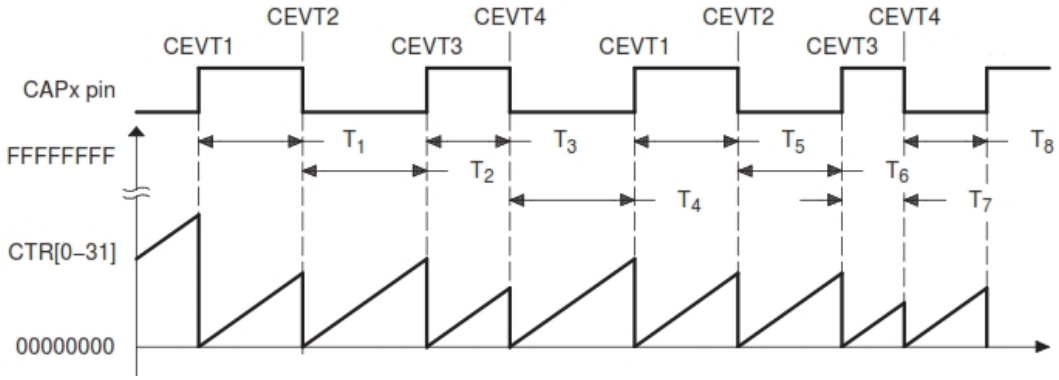
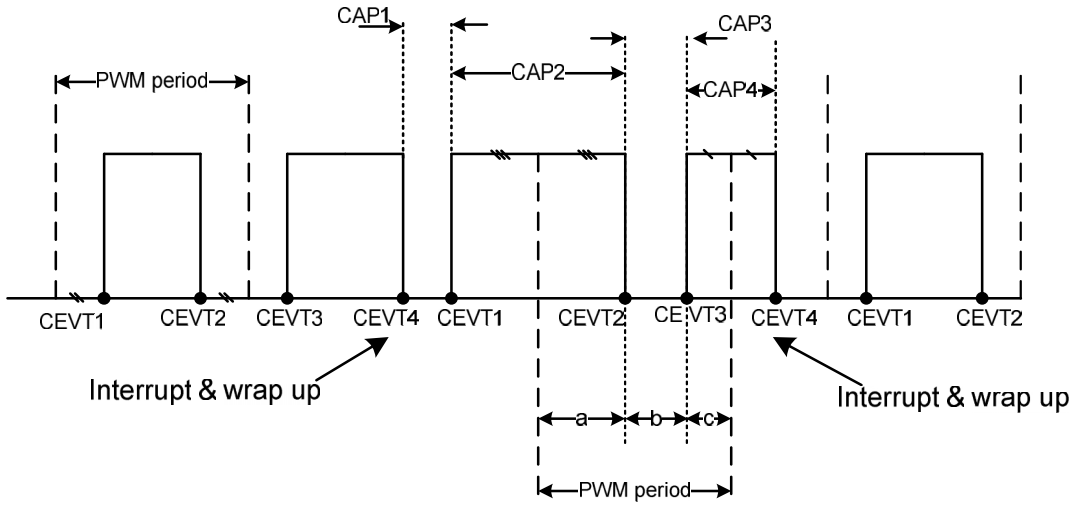
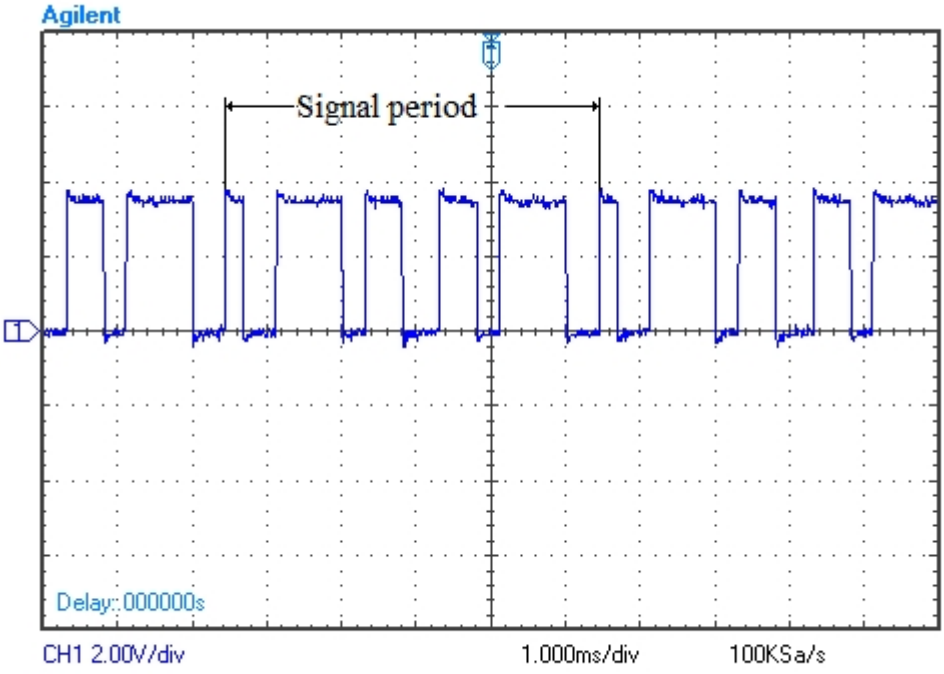
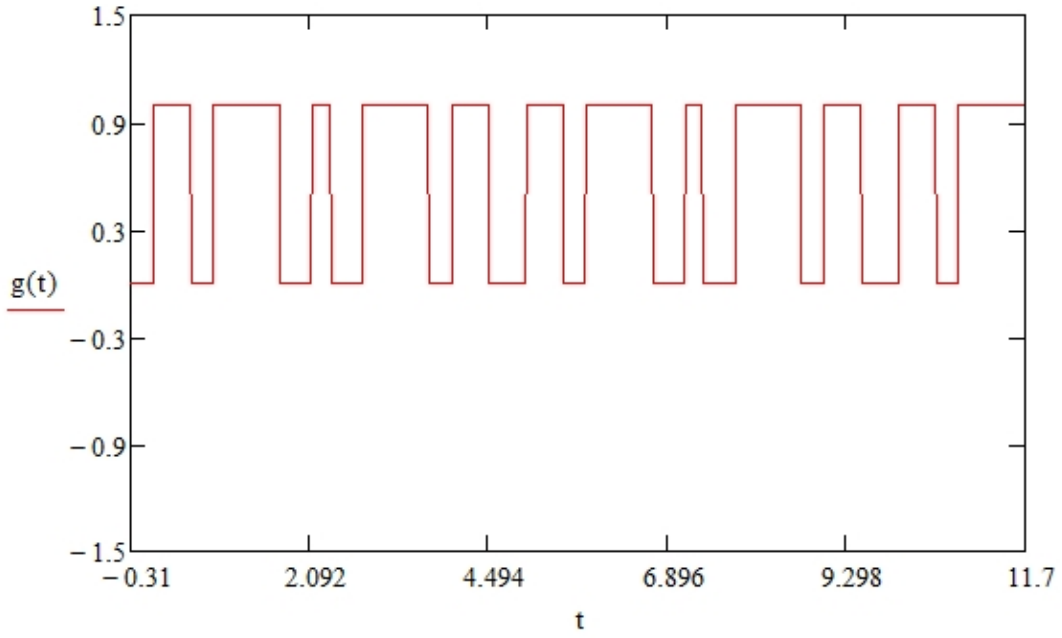
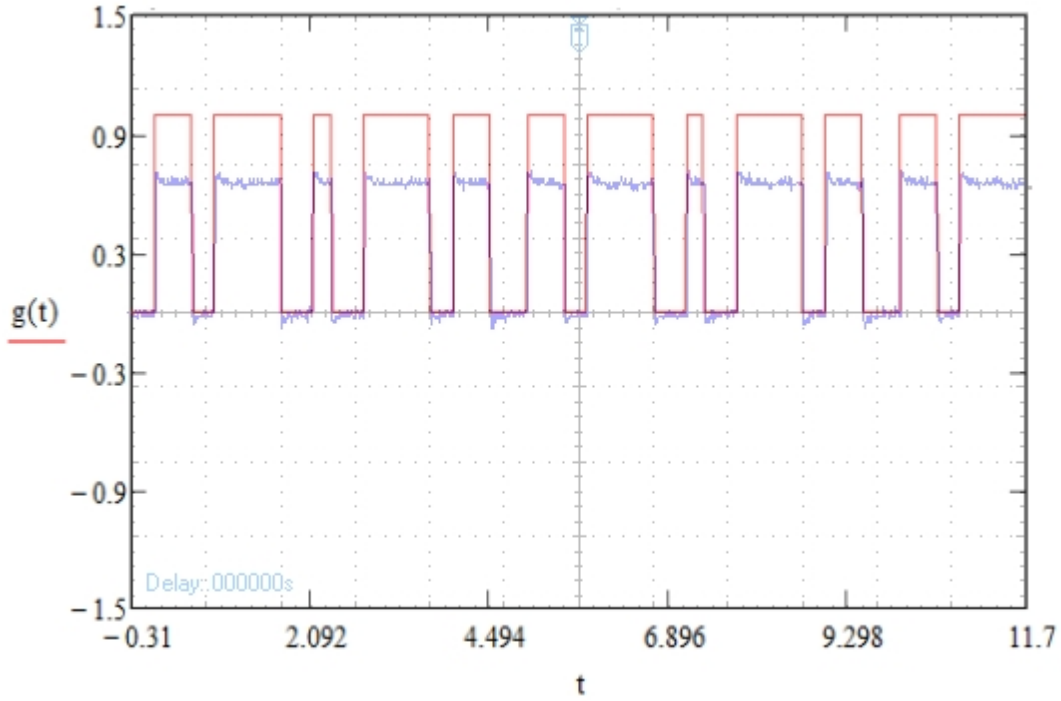
void Setup_ePWM(void)
```

```
{
    scan_for_PWM_solutions(63000,10,freq_carrier,&par);
    EPwm1Regs.TBCTL.bit.SYNCOSEL =1;  //SYNCOUT when CTR=0


    EPwm1Regs.TBCTL.bit.CLKDIV = par.clk_div;
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = par.hsp_clk_div;
    EPwm1Regs.TBCTL.bit.CTRMODE = 2;     // up - down mode
    EPwm1Regs.AQCTLA.all = 0x0060;       // CMPA-UP=>clearA, CMPA-
DOWN=>setA
    EPwm1Regs.AQCTLB.all = 0x0090;       // CMPA-UP=>setB ,  CMPA-
DOWN=>clearB
    EPwm1Regs.TBPRD = par.tb_prd;
    EPwm1Regs.CMPCTL.all = 0x0000;  // 0x0000 shadow mode ON, load on
CTR=0
                                    // 0x0050 shadow mode OFF
    EPwm1Regs.ETSEL.all = 0;
    EPwm1Regs.ETSEL.bit.INTEN = 1;  //interrupt enable for ePWM1
    EPwm1Regs.ETSEL.bit.INTSEL = 2; //interrupt on CTR=PRD
    EPwm1Regs.ETPS.bit.INTPRD = 1;  //interrupt on 1st event
}

interrupt void ePWM1A_isr (void)
{
static unsigned int index2=0;
GpioDataRegs.GPATOGGLE.bit.GPIO9 = 1;
EPwm2Regs.TBSTS.bit.SYNCI=1; //Clear SYNCI bit for ePWM2
index++;

if (index==10) index=1;

index_percentage=_IQ(((index)*1.0)/10.0);
EPwm1Regs.CMPA.half.CMPA= _IQmpy(index_percentage,  EPwm1Regs.TBPRD);
CMPA_buff[index]=EPwm1Regs.CMPA.half.CMPA;

EPwm1Regs.ETCLR.bit.INT=1; //Clear ePWM1 Interrupt flag
PieCtrlRegs.PIEACK.all=4; //Acknowledge this interrupt to receive
more interrupts for group3
}
//===========================================================================
// End of SourceCode
//===========================================================================
```

7.6 Βιβλιογραφία

[1] «TMS320F28335 Enhanced Pulse width modulation module» available at http://www.ti.com /

[2] «TMS320F28335 Data Manual» available at http://www.ti.com /

ΚΕΦΑΛΑΙΟ 8

EPWM MODULE - ΠΡΟΣΤΑΣΙΑ ΤΩΝ ΗΜΙΑΓΙΚΩΝ ΔΙΑΚΟΠΤΩΝ ΔΙΑΤΑΞΕΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΙΣΧΥΟΣ

8.1 Εισαγωγή

Για τον έλεγχο διατάξεων Ηλεκτρονικών ισχύος, εκτός από τις τυπικές λειτουργίες που προσφέρει ένας μικροεπεξεργαστής, υπάρχει αναγκαιότητα για λειτουργίες οι οποίες αποσκοπούν **στην ασφάλεια των ημιαγωγικών διακοπτών της εν λόγω διάταξης**. Ο F28335 καλύπτει αυτή την αναγκαιότητα με τις μονάδες του Dead Band και του Trip Zone.

8.2 Dead Band Sub-Module [1]

Στο σχήμα 8.1 παρουσιάζεται μια τυπική διάταξη ηλεκτρονικών ισχύος η οποία οδηγεί μία τριφασική μηχανή. Όπως είναι γνωστό δύο διακόπτες του ίδιου κλάδου δεν επιτρέπεται να άγουν την ίδια χρονική στιγμή, διότι σε αυτή την περίπτωση δημιουργείται βραχυκύκλωμα στα άκρα του πυκνωτή της DC πλευρά του κυκλώματος. Το ρεύμα που διέρχεται μέσω των ημιαγωγών σε αυτή την περίπτωση είναι δυνατό να τους καταστρέψει.



Σχήμα 8.1: Τυπική διάταξη οδήγησης τριφασικού κινητήρα[1]

Το πρόβλημα πηγάζει από το γεγονός ότι οι ημιαγωγοί διακόπτες συνήθως ανοίγουν γρηγορότερα από ότι κλείνουν (ειδικά οι ημιαγωγοί τεχνολογίας FET). Οπότε υπάρχει πιθανότητα επικάλυψης του χρόνου αγωγής μεταξύ διακοπτών του ίδιου κλάδου (shoot-through).

Για την αντιμετώπιση του παραπάνω προβλήματος πρέπει να εισαχθεί ένα χρονικό διάστημα καθυστέρησης στην αλλαγή κατάστασης των ημιαγωγών (από αγωγή σε μη αγωγή, ή αντίστροφα).

Ένας τρόπος για τη δημιουργία αυτής της καθυστέρησης είναι η χρήση ενός βαθυπερατού φίλτρου στην πύλη των ημιαγωγών. Ωστόσο, αυτή η μέθοδος εξαρτάται από την εκάστοτε διάταξη και για να μεταβληθούν οι παράμετροι του φίλτρου πρέπει να μεταβληθούν οι αντιστάσεις και οι πυκνωτές που το συνθέτουν.

Η δεύτερη λύση είναι η καθυστέρηση να υλοποιηθεί μέσω λογισμικού. Ο F28335 προσφέρει αυτή τη δυνατότητα με κατάλληλο on-chip υλικό, το οποίο δεν επιβαρύνει επιπλέον CPU. Πρόσθετο πλεονέκτημα αυτής της μεθόδου είναι η δυνατότητα ακριβούς καθορισμού χρονικής διάρκειας της καθυστέρησης.



Σχήμα 8.2: Διάγραμμα λειτουργίας της μονάδας Dead Band[1]

Η θέση του dead band μεταξύ των καναλιών PWMxA και PWMxB εξαρτάται από την τιμή των bit S0-5. Αν και επιτρέπονται όλοι οι δυνατοί συνδυασμοί πρακτικά προτιμούμε S4=0 και S5=0 (δηλαδή η πηγή είναι το PWMxA κανάλι) και για αυτές τις τιμές παρουσιάζονται στον παρακάτω πίνακα οι δυνατές περιπτώσεις:

Πίνακας 8.1: Διαθέσιμοι συνδυασμοί των S0-S3 για τη δημιουργία Dead Band

Mode	Mode Description	S3	S2	S1	S0
1	No delay	x	x	0	0
2	Active High Complementary (AHC)	1	0	1	1
3	Active Low Complementary (ALC)	0	1	1	1
4	Active High (AH)	0	0	1	1
5	Active Low (AL)	1	1	1	1
6	EPWMxA Out= EPWMxA In (no delay) EPWMxB Out= EPWMxA In with Falling Edge Delay	0/1	0/1	0	1
7	EPWMxA Out= EPWMxA In with Rising Edge Delay EPWMxB Out= EPWMxB In (no delay)	0/1	0/1	1	0

Στο σχήμα 8.3 παρουσιάζονται ποιοτικά οι παραπάνω τρόποι λειτουργίας:



Σχήμα 8.3: Υλοποιήσεις Dead Band του Πίνακα 8.1

Στον πίνακα 8.2 αναγράφονται οι καταχωρητές οι οποίοι χρησιμεύουν σε αυτή την ενότητα:

Πίνακας 8.2: Καταχωρητές Dead Band

Όνομα	Περιγραφή	Δομή
DBCTL	Dead Band Control	EPwnxRegs.DBCTL.all
DBRED	10-bit Rising Edge Delay	EPwnxRegs.DBRED
DBFED	10-bit Falling Edge Delay	EPwnxRegs.DBFED

Οι καθυστερήσεις FED και RED υπολογίζονται από τον τύπο:

$$RED = T_{TBCLK} \cdot DBRED$$
$$FED = T_{TBCLK} \cdot DBFED \quad (8.1)$$

,όπου

$$T_{TBCLK} = 6,67 \cdot CLKDIV \cdot HSPCLKDIV \quad (8.2)$$

Στο σχήμα 8.4 παρουσιάζεται λεπτομερώς ο καταχωρητής ελέγχου του Dead Band. Τα bit S0-S5 που αναφέρθηκαν προηγουμένως αντιστοιχούν στα bit 0-5 του εν λόγω καταχωρητή.



Σχήμα 8.4: Καταχωρητής Ελέγχου Dead Band [1]

8.2.1 Παράδειγμα

Ζητούμενο για το συγκεκριμένο παράδειγμα είναι να σχηματιστεί dead band τύπου Active High Complementary (AHC), και η διάρκειά του να είναι παραμετροποιήσιμη και ανεξάρτητη από τη συχνότητα του PWM. Σαν βάση θα χρησιμοποιηθεί η εφαρμογή του Κεφαλαίου 7.

Για την επίτευξη AHC Dead Band χρειάζονται οι παρακάτω εντολές:

```
EPwm1Regs.DBCTL.bit.IN_MODE=0;  //  PWM1A is source for RED & FED
EPwm1Regs.DBCTL.bit.POLSEL=2;   //  Active high complementary (FED)
EPwm1Regs.DBCTL.bit.OUT_MODE=3; //  RED & FED (DMB fully enabled)
```

Ενώ για τον καθορισμό της διάρκειας του Dead Band και έχοντας υπ' όψιν τον τύπο (8.1), γράφουμε τις παρακάτω εντολές:

```
EPwm1Regs.DBRED=round(RED_in_ns/par.Ttb_clk); //6.67ns*15.000=1000us
EPwm1Regs.DBFED=round(FED_in_ns/par.Ttb_clk);
```

Η μεταβλητής `RED|FED_in_ns` είναι η επιθυμητή διάρκεια του Dead Band σε ns, ενώ η μεταβλητή `par.Ttb_clk` είναι ένα πεδίο float το οποίο προστέθηκε στη δομή `PWM_parameters` και περιέχει σε ns την περίοδο ρολογιού για το ePWM. Η τιμή του υπολογίζεται από τον τύπο (8.1) και γίνεται εντός της συνάρτησης `scan_for_PWM_solutions` όπως αυτή εμφανίζεται στο κεφάλαιο 6Α. Για να λειτουργήσει το πρόγραμμα σωστά πρέπει να προστεθεί η παρακάτω γραμμή κώδικα στη συνάρτηση:

```
old.Ttb_clk=6.67*cdiv[old.clk_div]*hspdiv[old.hsp_clk_div];
*p=old;
return 0;
```

Οι άλλες δύο γραμμές προστέθηκαν για την επισήμανση του σημείου του κώδικα στο οποίο γίνεται η προσθήκη.

Name	Value	Type	Radix
par	{...}	PWM_...	hex
clk_div	0	int	dec
hsp_clk_div	2	int	dec
tb_prd	18750	int	dec
a_diff	0.0	float	float
Ttb_clk	26.68	float	float
EPwm1Regs.DBRED	37	Uint16	unsigned
freq_carrier	1000	long	dec

Σχήμα 8.5: Παράμετροι προγράμματος για 1μs επιθυμητό Dead Band στο 1KHz



Σχήμα 8.6: Καθυστέρηση αρνητικής ακμής (FED)



Σχήμα 8.7: Καθυστέρηση θετικής ακμής (RED)

Η μικρή απόκλιση που παρατηρείται σε σχέση με την επιθυμητή τιμή (1μs) οφείλεται στην ανάλυση του παλμογράφου για το συγκεκριμένο ns/div.

8.3 Trip Zone Sub-Module [1]

Τα σήματα Trip δημιουργούνται από αισθητήρες (συνήθως υπερέντασης) οι οποίοι στέλνουν ένα ψηφιακό σήμα όταν ξεπεραστεί κάποιο κατώφλι. Τα χαρακτηριστικά της μονάδας αυτής είναι:

- 6 σήματα trip (TZ1 έως TZ6) τα οποία μπορούν να συσχετιστούν με οποιοδήποτε από τα 6 κανάλια PWM
- Όταν ανιχνευθεί σφάλμα, τότε τα PWMxA|B κανάλια μπορούν να εξαναγκαστούν στις εξής καταστάσεις:
    - Υψηλό λογικό επίπεδο
    - Χαμηλό λογικό επίπεδο
    - Υψηλή αντίσταση εξόδου
    - Καμία δράση
- Υποστηρίζεται άπαξ δράση (one-shot) ή δράση κύκλο με κύκλο (cycle-by-cycle)
- Trip μπορεί να προκληθεί και μέσω λογισμικού
- Αν δε χρειάζεται το Trip Zone module μπορεί να αγνοηθεί



Σχήμα 8.8: Διάγραμμα λειτουργίας Trip Zone

Στον παρακάτω πίνακα παρουσιάζονται οι καταχωρητές της μονάδας:

Πίνακας 8.3: Καταχωρητές Trip Zone

Όνομα	Περιγραφή	Δομή
TZCTL	Trip Zone Control	EPwnxRegs.TZCTL.all
TZSEL	Trip Zone Select	EPwnxRegs.TZSEL.all
TZEINT	Enable Interrupt	EPwnxRegs.TZEINT.all
TZFLG	Trip Zone Flag	EPwnxRegs.TZFLG.all
TZCLR	Trip Zone Clear	EPwnxRegs.TZCLR.all
TZFRC	Trip Zone Force	EPwnxRegs.TZFRC.all

Να σημειωθεί ότι όλοι οι καταχωρητές της μονάδας είναι προστατευμένοι. Για να έχουμε πρόσβαση πρέπει να γράψουμε την εντολή EALLOW, και αφού τελειώσουμε EDIS.

ePWM Trip Zone Control Register

EPwmxRegs.TZCTL



Σχήμα 8.9: Καταχωρητής ελέγχου Trip Zone

Ο καταχωρητής αυτός καθορίζει την κατάσταση των καναλιών ePWM στην περίπτωση που ανιχνευθεί σφάλμα.

ePWM Trip Zone Select Register

EPwmxRegs.TZSEL



Σχήμα 8.10: Καταχωρητής επιλογής διακοπής trip zone

Με τον TZSEL καθορίζουμε ποιο από τα σήματα trip είναι ενεργά, και με τι τρόπο (one-shot ή cycle-by-cycle)



Σχήμα 8.11: Καταχωρητής ενεργοποίησης διακοπής trip zone

Τέλος ο καταχωρητής του σχήματος 8.11 είναι υπεύθυνος για το αν θα προκληθεί διακοπή μετά την ανίχνευση του σήματος trip.

8.4 Βιβλιογραφία

[1] «TMS320F28335 Enhanced Pulse width modulation module» available at http://www.ti.com /

[2] «TMS320F28335 Data Manual» available at http://www.ti.com /

ΚΕΦΑΛΑΙΟ 9

ECAP MODULE - ΜΟΝΑΔΑ ΣΥΛΛΗΨΗΣ ΨΗΦΙΑΚΩΝ ΣΗΜΑΤΩΝ

9.1 Εισαγωγή

Η μονάδα σύλληψης ψηφιακών σημάτων (capture unit) αποτελεί, ίσως, το σημαντικότερο στοιχείο του F28335. Χάρη σε αυτή τη μονάδα έχουμε τη δυνατότητα να ελέγχουμε πλήρως παλμοσειρές, ακόμα και αν αυτές παράγονται από το ίδιο το DSC. Με αυτόν τον τρόπο δεν απαιτείται παλμογράφος για την επαλήθευση της ορθότητας ενός προγράμματος παραγωγής παλμοσειρών.

Η αρχή λειτουργίας του είναι αρκετά απλή και φαίνεται στο σχήμα 9.1. Η βάση του eCAP είναι ένα **μετρητής 32 bit** ο οποίος μετράει από τη στιγμή που ενεργοποιείται (μέσα στο πρόγραμμα) μέχρι τη στιγμή που θα τον **μηδενίσουμε** ή μέχρι να **υπερχειλίσει**. Όταν στο σήμα εισόδου ανιχνευθεί κατάλληλη διέγερσή (θετική η αρνητική ακμή) τότε καταχωρείται, σε έναν από τους τέσσερις καταχωρητές(CAP1-4) του eCAP, η τιμή του μετρητή εκείνη τη στιγμή (time-stamp).



Σχήμα 9.1: Δομικό διάγραμμα της μονάδας eCAP [1]

Η επεξεργασία του αποτελέσματος αυτού είναι στην ευχέρεια του προγραμματιστή και το εγχείρημα της μετατροπής των τιμών του μετρητή σε αναγνώσιμα αποτελέσματα είναι θέμα που χρίζει προσοχής. Ωστόσο, στο τέλος του κεφαλαίου υπάρχει εφαρμογή στην οποία λύνεται ένα τέτοιο πρόβλημα.

Όπως θα φανεί και στην ανάλυση των καταχωρητών, το eCAP unit προσφέρει και τη δυνατότητα λειτουργίας σαν **Auxiliary(βοηθητικό) PWM**. Συνολικά υπάρχουν **4 pin** τα οποία μπορούν να λειτουργήσουν σαν eCAP ή APWM.

9.2 Καταχωρητές του eCAP module [1]

Ο πίνακας 9.1 παρουσιάζει το σύνολο των καταχωρητών για τη μονάδα σύλληψης κυματομορφών:

Πίνακας 9.1: Καταχωρητές eCAP

Name	Description	Structure
ECCTL1	Capture Control 1	ECapxRegs.ECCTL1.all
ECCTL2	Capture Control 2	ECapxRegs.ECCTL2.all
TSCTR	Time Stamp Counter	ECapxRegs.TSCTR
CTRPHS	Counter Phase Offset	ECapxRegs. CTRPHS
CAP1	Capture 1	ECapxRegs.CAP1
CAP2	Capture 2	ECapxRegs.CAP2
CAP3	Capture 3	ECapxRegs.CAP3
CAP4	Capture 4	ECapxRegs.CAP4
ECEINT	Enable Interrupt	ECapxRegs.ECEINT.all
ECFLG	Interrupt Flag	ECapxRegs. ECFLG.all
ECCLR	Interrupt Clear	ECapxRegs.ECCLR.all
ECFRC	Interrupt Force	ECapxRegs.ECFRC.all

Στις επόμενες σελίδες αναλύονται επιλεγμένοι καταχωρητές του πίνακα 9.1 με μεγαλύτερη λεπτομέρεια.



Σχήμα 9.2: Καταχωρητής ελέγχου eCAP 1 (15-8 bit)

FREE_SOFT

Ελέγχει την αλληλεπίδραση του DSC με το JTAG-Emulator. Αν ο κώδικας συναντήσει ένα breakpoint καθορίζουμε τι θα γίνει με το συγκεκριμένο APWM.

PRESCALE

Λειτουργεί σαν φίλτρο της εισόδου του σήματος προς σύλληψη. Εάν θέλουμε να εντοπίσουμε κάθε γεγονός θέτουμε «00000», αν θέλουμε κάθε δεύτερο γεγονός «00001» κοκ.

CAPLDEN (Capture Load Enable)

«Διακόπτης» για την εκάστοτε μονάδα σύλληψης (capture unit)

eCAP Control Register 1
EcapxRegs.ECCTL1

Lower Register:



Σχήμα 9.3: Καταχωρητής ελέγχου eCAP 1 (7-0 bit)

CAPxPOL (Capture Polarity)

Καθορισμός θετικής ή αρνητικής ακμοπυροδότησης για τα γεγονότα CEVT1-4 (capture event).

CTRRSTx (Counter Reset)

Καθορισμός της σχετικής (1) ή απόλυτης (0) μέτρησης των γεγονότων. Εάν κάποιο από τα πεδία αυτά είναι 1 τότε ο μετρητής TSCTR θα μηδενιστεί μετά από το αντίστοιχο γεγονός. Είναι δυνατόν αυτό να γίνεται ακόμα και μετά από κάθε γεγονός (όπως θα δούμε και στην εφαρμογή στο τέλος του κεφαλαίου)

eCAP Control Register 2
EcapxRegs.ECCTL2

Upper Register:



Σχήμα 9.4: Καταχωρητής ελέγχου eCAP 2 (15-8 bit)

APWMPOL (Auxiliary PWM Output polarity)

Αφορά τη λειτουργία PWM του συγκεκριμένου pin. Αν το πεδίο είναι 0 τότε το σήμα θα είναι ενεργό στα 3,3V, ενώ στην αντίθετη περίπτωση στα 0V.

CAP_APWM (Capture/APWM mode)

Επιλογή μεταξύ λειτουργίας Capture και APWM.

SWSYNC (Software Synchronization)

Χρησιμοποιείται για το συγχρονισμό μεταξύ διαφορετικών καναλιών του APWM. Στην περίπτωση ενεργού σήματος συγχρονισμού θα φορτωθεί στον TSCTR η αρχική τιμή CTRPHS (αντίστοιχο του καταχωρητή Phase για το ePWM unit)



Σχήμα 9.5: Καταχωρητής ελέγχου eCAP 2 (7-0 bit)

SYNCO_SEL (Sync Out Select)

Αφορά τη λειτουργία APWM. Χρησιμοποιείται για να καθορίσει το σήμα συγχρονισμού εξόδου. Ο μηχανισμός είναι παρόμοιος με το ePWM και δε θα αναλυθεί περαιτέρω.

SYNCI_EN (Sync In Enable)

Ενεργοποιεί τη λήψη σήματος συγχρονισμού για το APWM.

REARM [3], STOP_WRAP[2-1] και CONT_ONESHT[0]

Τα πεδία αυτά είναι αλληλένδετα, οπότε θα εξεταστούν μαζί. Συγκεκριμένα από αυτά καθορίζεται αν η λειτουργία της σύλληψης θα γίνει άπαξ ή συνεχώς.

Λειτουργία άπαξ (one shot):

Όταν οπλιστεί, το eCAP περιμένει για 1 έως 4 γεγονότα (ο αριθμός καθορίζεται από στο πεδίο STOP_WRAP) πριν παγώσει τους καταχωρητές CAP1-4. Αν επανα-οπλίσουμε το eCAP (rearm) τότε και μόνο τότε είναι δυνατόν να λάβουμε την επόμενη αλληλουχία γεγονότων.

Συνεχής λειτουργία (continuous):

Στη συνεχή λειτουργία τα γεγονότα διαδέχονται κυκλικά το ένα το άλλο (1→2→3→4→1→2→…) και οι καταχωρητές CAPx ανανεώνονται επίσης κυκλικά. Η τιμή περιτύλιξης (wrap) καθορίζει τον αριθμό των καταχωρητών που εναλλάσσονται (CAP1-CAPx)



Σχήμα 9.6: Καταχωρητής ενεργοποίησης διακοπών

Ο καταχωρητής **ECEINT** καθορίζει πότε θα συμβεί η διακοπή η οποία θα επιτρέψει την επεξεργασία των σημείων χρόνου (time stamps) CAP1-4. Τα **bit 6-7** αφορούν το σημείο διακοπής στην περίπτωση APWM λειτουργίας.

9.3 Προχωρημένα θέματα

9.3.1 Καθαρισμός Σημαίας Διακοπής [2]

Υπενθυμίζουμε ότι όταν καλείται μία διακοπή *συνήθως* μία σημαία (flag) τίθεται. Για να μπορούμε να εντοπίσουμε και δεύτερη διακοπή πρέπει αυτή η σημαία να καθαριστεί.

Η διακοπή που προκαλείται από το eCAP module χρειάζεται ιδιαίτερη προσοχή στον καθαρισμό της σημαίας της γιατί ελέγχεται (πρακτικά) από δύο bit. Όπως φαίνεται στο σχήμα 9.7 το bit **ECFLG (Interrupt Flag)** είναι η έξοδος ενός μανδαλωτή. Το **CLEAR του μανδαλωτή ελέγχεται από το ECCLR του eCAP1** ενώ το **SET από το ECCLR του εκάστοτε γεγονότος** που προκαλεί τη διακοπή. Οπότε, πρέπει **πρώτα** να θέσουμε το SET=0 και **μετά** να θέσουμε το CLEAR=1. Η αλλιώς με κώδικα C:

```
ECap1Regs.ECCLR.bit.CEVT4 = 1; // Clear the CEVT4 flag
ECap1Regs.ECCLR.bit.INT = 1;   // Clear the ECAP1 interrupt flag
```



Σχήμα 9.7: Δομικό διάγραμμα ελέγχου διακοπών eCAP [1]

Αν η σειρά αναστραφεί το πρόγραμμα θα «μπαίνει» στη ΡΕΔ του eCAP μία φορά επιπλέον ανά περίοδο γεγονός καταστροφικό για την εκτέλεση του προγράμματος. Η συγκεκριμένη λεπτομέρεια δεν αναφέρεται στη βιβλιογραφία του F28335 και για αυτό το λόγο επισημαίνεται.

9.4 Εφαρμογή : Ψηφιακός Παλμογράφος.

Όπως έχει αναφερθεί και παραπάνω το Capture module του F28335 παρέχει στον προγραμματιστή τη δυνατότητα να εξετάζει με μεγάλη ακρίβεια παλμοσειρές. Σε αυτή την εφαρμογή υλοποιείται ένας «ψηφιακός παλμογράφος», ο οποίος απαιτεί από το χρήστη σαν **είσοδο** την **περιοδικότητα της εξεταζόμενης παλμοσειράς** (παλμοί ανά περίοδο) και σαν **έξοδο** παρέχει το **εύρος του κάθε παλμού** (σαν ποσοστό επί της συχνότητας του φορέα).

Ο τρόπος μέτρησης της παλμοσειράς φαίνεται στο σχήμα 9.8. Συγκεκριμένα υπολογίζεται το εύρος των διαστημάτων μεταξύ των ακμών, καθώς ο μετρητής TSCTR (Time Stamp Counter) **μηδενίζεται μετά από κάθε γεγονός σύλληψης** (capture event). Η διακοπή για την καταχώρηση των τιμών του εκάστοτε μετρητή (CAP1 έως CAP4) πραγματοποιείται με το γεγονός CAP4.



Σχήμα 9.8: Ο τρόπος μέτρησης που χρησιμοποιήθηκε στο δεδομένο παράδειγμα [1]

Το περιεχόμενο, ωστόσο, των καταχωρητών CAP1-4 μας είναι άχρηστο αν δεν υπάρχει κάποια **σταθερά** βάσει της οποίας θα συγκριθούν αυτά τα μεγέθη, ώστε να εξαχθεί **ο κύκλος λειτουργίας κάθε παλμού**. Αυτή η σταθερά θα είναι σαφώς η περίοδος του PWM που έχει επιλεχθεί για την παλμοσειρά. Στο σχήμα 9.9 φαίνεται εύκολα ότι η περίοδος είναι ίση με:

$$\text{Period} = a + b + c = \frac{\text{CAP4}}{2} + \frac{\text{CAP2}}{2} + \text{CAP3} \qquad (9.1)$$



Σχήμα 9.9: Σχηματική επεξήγηση του τύπου (9.1)

9.4.1 Σημαντικά χωρία κώδικα

- `DC_table= (float*)malloc(2*period_pulse_count*sizeof(float));`

Δυναμική δέσμευση του DC_table (πίνακας που περιέχει όλους τους κύκλους λειτουργίας ανά περίοδο) καθώς η περιοδικότητα της παλμοσειράς δεν είναι εκ των προτέρων γνωστή.

- `if (index==(period_pulse_count-2)) ECap1Regs.ECEINT.all = 0x0010;`

Η συγκεκριμένη εντολή βρίσκεται στην ΡΕΔ του ePWM και έχει ως στόχο την επιλογή του πρώτου παλμού της παλμοσειράς (όπως αυτός θα αποθηκευθεί στον DC_table).

- `ECap1Regs.ECCTL2.all = 0x0096;`

Εντολή αρχικοποίησης του καταχωρητή ECCTL2. Στο δυαδικό σύστημα η τιμή του καταχωρητή είναι: $(0000\ 0000\ 1001\ 0110)_2$. Αναλύοντας τα πεδία σύμφωνα με τα σχήματα 9.4 και 9.5, έχουμε:

- **Bit [15-8]= $(0000\ 0000)_2$:** Δε χρησιμοποιείται η λειτουργία APWM και οι λειτουργίες που τη συνοδεύουν.
- **Bit [7-6]= $(10)_2$:** Απενεργοποίηση παλμού SYNCOUT. ο Αφορά λειτουργία APWM.
- **Bit [5]= $(0)_2$:** Απενεργοποίηση συγχρονισμού. Αφορά λειτουργία APWM.
- **Bit [4]= $(0)_2$:** Δε γίνεται επανοπλισμός.
- **Bit [2-1]= $(11)_2$:** Αναδίπλωση μετά το $4^{ο}$ γεγονός.

- **Bit [0]= $(0)_2$**: Συνεχής λειτουργία.

- `ECap1Regs.ECCTL1.all = 0x01EE;`

Εντολή αρχικοποίησης του καταχωρητή ECCTL1. Στο δυαδικό σύστημα η τιμή του καταχωρητή είναι: $(0000\ 0001\ 1110\ 1110)_2$. Αναλύοντας τα πεδία σύμφωνα με τα σχήματα 9.2 και 9.3, έχουμε:

- **Bit [15-14]= $(00)_2$:** Αδιάφορο πεδίο για την εφαρμογή.
- **Bit [13-9]= $(00000)_2$:** Η μονάδα eCAP δεν αγνοεί κανένα γεγονός.
- **Bit [8]= $(1)_2$:** Ενεργοποίηση των καταχωρητών CAP1-4.
- **Bit [7-0]= $(1110\ 1110)_2$:** Αυτό το πεδίο χρειάζεται για να προσομοιώσουμε το Σχήμα 9.9. Ο μετρητής μηδενίζεται μετά από κάθε γεγονός. Τα γεγονότα 1 και 3 περιμένουν θετική ακμή και τα 2,4 αρνητική.

- `interrupt void eCAP1_isr(void)`

```
{     // Rest ISR code
     switch (2*period_pulse_count-index2-3-1)
         {
         case -1 :    *(DC_table+index2)=DC_1st_Zero;
                      *(DC_table+index2+1)=DC_1st_Pulse;
                      *(DC_table+index2+2)=DC_2nd_Zero;
                      *(DC_table+0)=DC_2nd_Pulse;
                      index2=1;
                      break;
         case -2 :    *(DC_table+index2)=DC_1st_Zero;
                      *(DC_table+index2+1)=DC_1st_Pulse;
                      *(DC_table+0)=DC_2nd_Zero;
                      *(DC_table+1)=DC_2nd_Pulse;
                      index2=2;
                      break;
         case -3 :    *(DC_table+index2)=DC_1st_Zero;
                      *(DC_table+0)=DC_1st_Pulse;
                      *(DC_table+1)=DC_2nd_Zero;
                      *(DC_table+2)=DC_2nd_Pulse;
                      index2=3;
                      break;
         case 0  :    *(DC_table+index2)=DC_1st_Zero;
                      *(DC_table+index2+1)=DC_1st_Pulse;
                      *(DC_table+index2+2)=DC_2nd_Zero;
                      *(DC_table+index2+3)=DC_2nd_Pulse;
                      index2=0;
                      break;
         default :    *(DC_table+index2)=DC_1st_Zero;
                      *(DC_table+index2+1)=DC_1st_Pulse;
                      *(DC_table+index2+2)=DC_2nd_Zero;
                      *(DC_table+index2+3)=DC_2nd_Pulse;
                      index2+=4;
         }// Rest ISR code
}
```

Τέλος, παρουσιάζεται το switch-case το οποίο αποδίδει τις τέσσερις τιμές του Capture unit (CAP1-4) ανηγμένες στην περίοδο του φορέα, όπως αυτή υπολογίστηκε

από τον τύπο (9.1). Το πρόβλημα που παρουσιάζεται αφορά το τέλος της εκάστοτε παλμοσειράς. Αν η παλμοσειρά περιέχει αριθμό παλμών μη πολλαπλάσιο του 4, στην ανακύκλωση του DC_table (δηλαδή όταν θα πρέπει να ξαναγραφτεί το στοιχείο DC_table[0]) θα υπάρχει αντίφαση μεταξύ της παλιάς και της νέας τιμής. Για να αποφευχθεί αυτή η διένεξη ήταν απαραίτητο να δημιουργηθούν τα πρώτα 4 case. Η περίπτωση default αφορά την αρχή και το μέσο της παλμοσειράς.

9.4.2 Πειραματικά Αποτελέσματα

Στο σχήμα 9.10 υπάρχει η έξοδος του GPIO 0 (ePWM 1) όπως φαίνεται στον παλμογράφο. Το δεδομένο στιγμιότυπο περιλαμβάνει περίπου δύο περιόδους. Η παλμοσειρά αποτελείται από παλμούς τυχαίου πλάτους, όπως φαίνεται και στον κώδικα του παραρτήματος



Σχήμα 9.10: Απεικόνιση της παλμοσειράς προς μελέτη στον παλμογράφο

Στο σχήμα 9.11 απεικονίζει το μέρος της μνήμης που είναι αποθηκευμένο το διάνυσμα DC_table (θέση μνήμης 0x592 έως 0x5A4). Όπως είναι φανερό οι τιμές παραμένουν ίδιες κατά τη διάρκεια εκτέλεσης του προγράμματος (εάν μεταβάλλονταν θα ήταν κόκκινες).

&(DC_table[0])	F28335 XDS100 USB Emulator/TMS320C2800_0			
0x00000592	0.4250018	0.2499983	0.4374954	0.8750241
0x0000059A	0.3124929	0.5000033	0.5000033	0.5000033
0x000005A2	0.2999993	0.9000113	1.401298e-44	0.25
0x000005AA	0.8750133	0.5	0.5	0.9
0x000005B2	-1.#QNAN	5.87747e-39	-1.220219e+28	1.486772e+32

Σχήμα 9.11: Ο χάρτης μνήμης του DSC στη θέση DC_table

Τέλος, μετά από επεξεργασία των δεδομένων με το πρόγραμμα MathCAD, σχεδιάστηκε η παλμοσειρά στο σχήμα 9.12. Για την επιβεβαίωση της ορθότητας των αποτελεσμάτων, τοποθετήθηκε η παλμοσειρά του MathCAD «πάνω» σε αυτή του παλμογράφου (Σχήμα 9.13). **Είναι φανερό ότι οι ακμές συμπίπτουν σε ικανοποιητικό βαθμό.** Οι όποιες διαφορές υπάρχουν οφείλονται στην αδυναμία του παλμογράφου να αναπαραστήσει με ακρίβεια το σήμα για τη δεδομένη ανάλυση.



Σχήμα 9.12: Η παλμοσειρά από το MathCAD



Σχήμα 9.13: Συγχώνευση των δύο παλμοσειρών

9.4.3 Κώδικας με σχόλια

```
#include "DSP2833x_Device.h"
#include <stdlib.h>

// external function prototypes
extern void InitSysCtrl(void);
extern void InitPieCtrl(void);
extern void InitPieVectTable(void);
// Prototype statements for functions found within this file.
void Gpio_select(void);
void Setup_ePWM1A(void);
void Setup_eCAP1(void);
interrupt void ePWM_isr(void);
interrupt void eCAP1_isr(void);
// Global Variables
Uint32 PWM_Duty;
Uint32 PWM_Period, Pulse_1, Pulse_2, old_cap1;
float DC_1st_Pulse,DC_2nd_Pulse,DC_2nd_Zero,DC_1st_Zero,* DC_table,*
CMPA_buff;
int index=0, index2=0, period_pulse_count=5;

//###########################################################################
//                         main code
//###########################################################################
void main(void)
{
    InitSysCtrl();  // Basic Core Init from DSP2833x_SysCtrl.c
    EALLOW;
    SysCtrlRegs.WDCR= 0x00AF;   // Re-enable the watchdog
    EDIS;
    DINT;               // Disable all interrupts
```

```
    Gpio_select();
    Setup_ePWM1A();     // init  ePWM1A
    Setup_eCAP1();      // init  eCAP1
    InitPieCtrl();      // basic setup of PIE table; from
DSP2833x_PieCtrl.c
    InitPieVectTable(); // default ISR's in PIE

    EALLOW;
    PieVectTable.EPWM1_INT= &ePWM_isr;
    PieVectTable.ECAP1_INT= &eCAP1_isr;
    EDIS;

    PieCtrlRegs.PIEIER3.bit.INTx1 = 1;  // Enable EPWM1_INT in PIE
group 3
    PieCtrlRegs.PIEIER4.bit.INTx1 = 1;  // Enable ECAP1_INT in PIE
group 4
    IER |= 0x000C;                      // Enable INT1

    EINT;
    ERTM;

    DC_table= (float*)malloc(2*period_pulse_count*sizeof(float));
    //Each period has an OFF and an ON duty cycles
    CMPA_buff= (float*)malloc(period_pulse_count*sizeof(float));
    //Each period has one CMPA
    while(1)
    {
            EALLOW;
            SysCtrlRegs.WDKEY = 0x55;   // service WD #1
            SysCtrlRegs.WDKEY = 0xAA;
            EDIS;}} // end of main

void Gpio_select(void)
{
    EALLOW;
    GpioCtrlRegs.GPAMUX1.all = 0
    GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1; // ePWM1A active
    GpioCtrlRegs.GPAMUX2.all = 0;
    GpioCtrlRegs.GPAMUX2.bit.GPIO24= 1; // eCAP1 active
    GpioCtrlRegs.GPBMUX1.all = 0;
    GpioCtrlRegs.GPBMUX2.all = 0;
    GpioCtrlRegs.GPCMUX1.all = 0;
    GpioCtrlRegs.GPCMUX2.all = 0;

    GpioCtrlRegs.GPADIR.all = 0;
    GpioCtrlRegs.GPBDIR.all = 0;        // GPIO63-32 as inputs
    GpioCtrlRegs.GPCDIR.all = 0;        // GPIO87-64 as inputs
    EDIS;
}

void Setup_ePWM1A(void)
{
    EPwm1Regs.TZCTL.all =  0;             //high impedance
    EPwm1Regs.TBCTL.bit.CLKDIV =  0;      // CLKDIV = 1
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = 1;    // HSPCLKDIV = 2
    EPwm1Regs.TBCTL.bit.CTRMODE = 2;      // up - down mode
    EPwm1Regs.AQCTLA.all = 0x0060;        // CTR=CMPA up=> set
                                          //CTR=CMPA down=> clear
    EPwm1Regs.TBPRD = 37500;              // 1KHz - PWM signal
    EPwm1Regs.CMPA.half.CMPA=EPwm1Regs.TBPRD/2;
```

```
    EPwm1Regs.CMPCTL.all = 0;
    EPwm1Regs.ETSEL.all = 0;
    EPwm1Regs.ETSEL.bit.INTEN = 1;      // interrupt enable for ePWM1
    EPwm1Regs.ETSEL.bit.INTSEL = 1;     // interrupt on CTR=0
    EPwm1Regs.ETPS.bit.INTPRD = 1;      // interrupt on first event

}

void Setup_eCAP1(void)
{
//--- Configure eCAP1 unit for capture
    ECap1Regs.ECEINT.all = 0;        // Disable all eCAP interrupts
    ECap1Regs.ECCTL1.bit.CAPLDEN = 0;// Disabled loading of capture
results
    ECap1Regs.ECCTL2.bit.TSCTRSTOP = 0;// Stop the counter
    ECap1Regs.TSCTR = 0;               // Clear the counter
    ECap1Regs.CTRPHS = 0;        // Clear the counter phase register
    ECap1Regs.ECCTL2.all = 0x0096;// ECAP control register 2
    ECap1Regs.ECCTL1.all = 0x01EE;// ECAP control register 1
    ECap1Regs.ECEINT.all = 0x0010;              // Enable desired
eCAP interrupts
}

interrupt void ePWM_isr(void)
{

    // Service watchdog every interrupt
    EALLOW;
    SysCtrlRegs.WDKEY = 0xAA;       // Service watchdog 1
    EDIS;

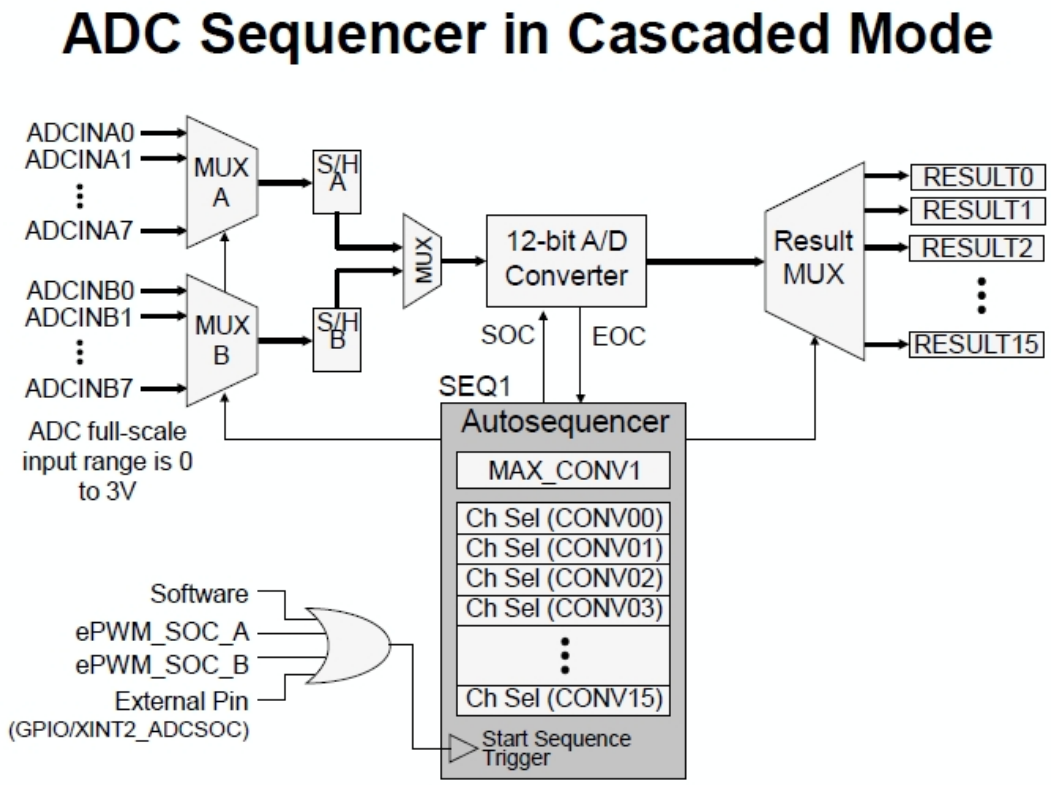
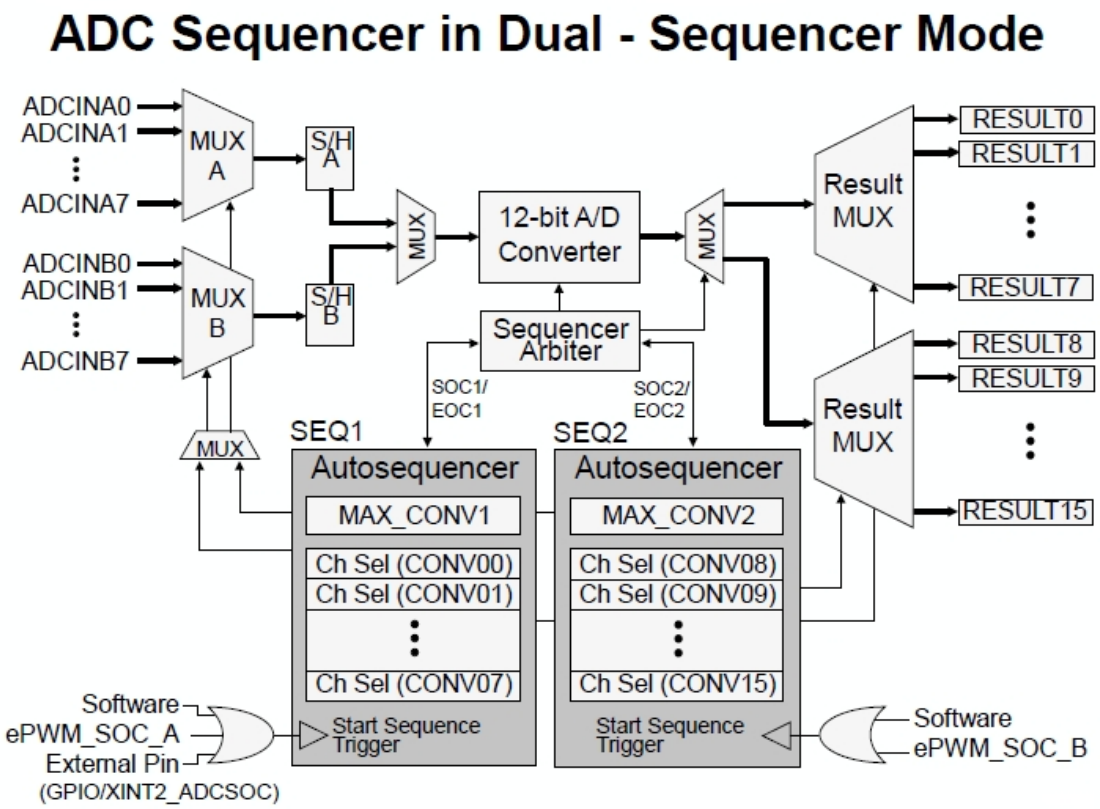
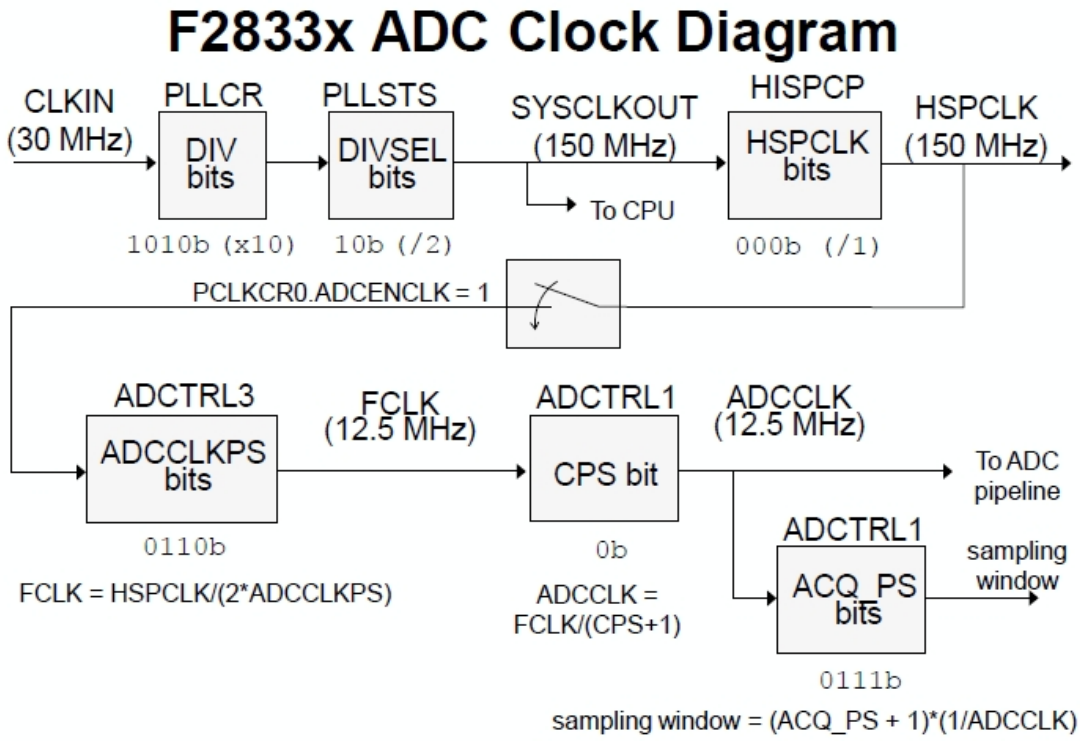
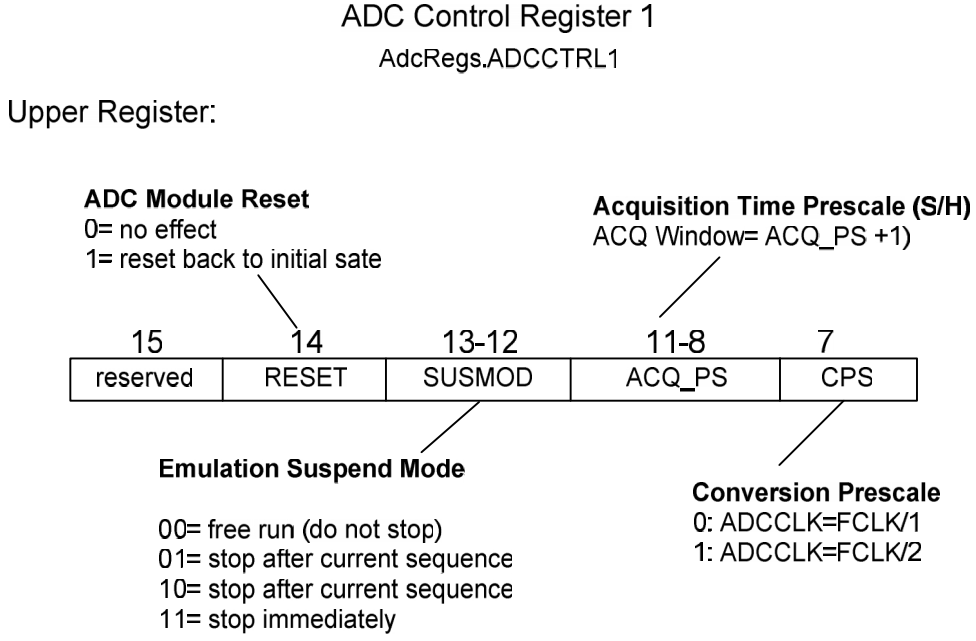
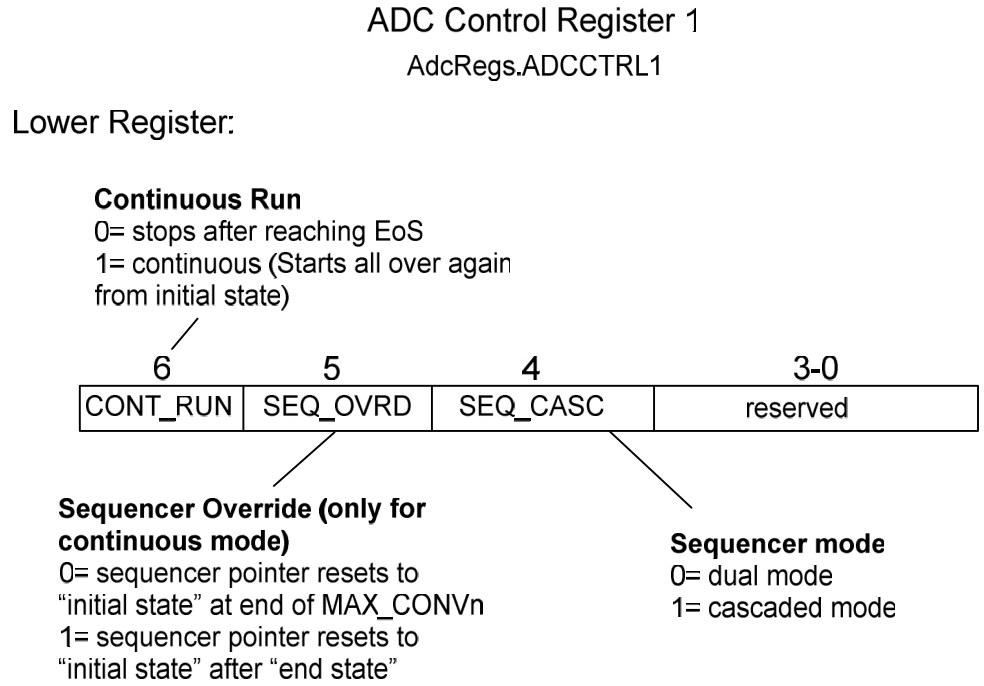
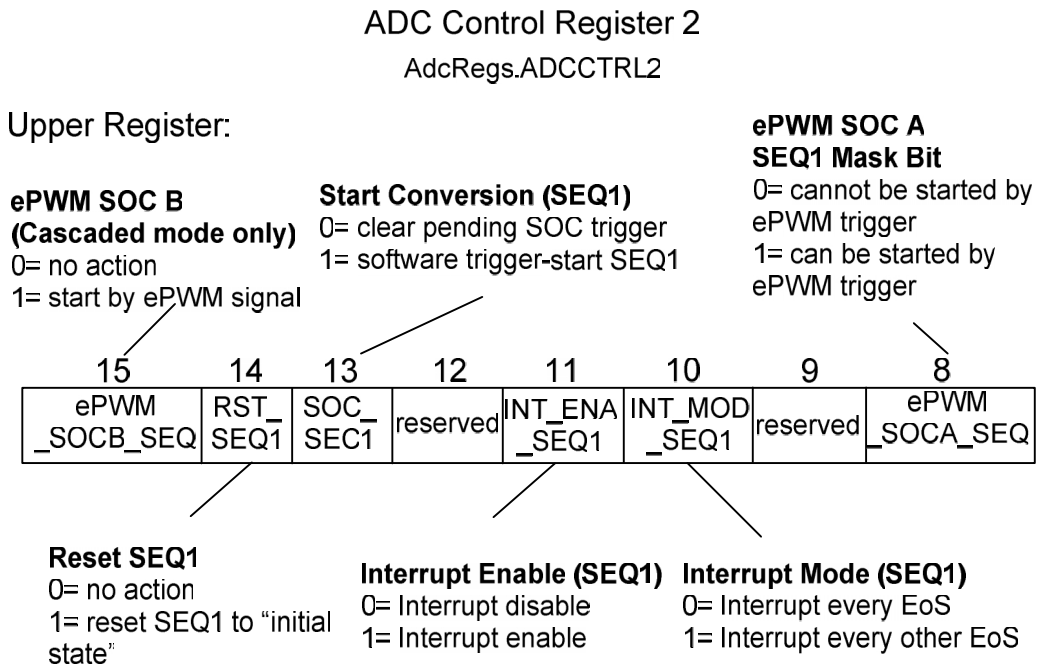
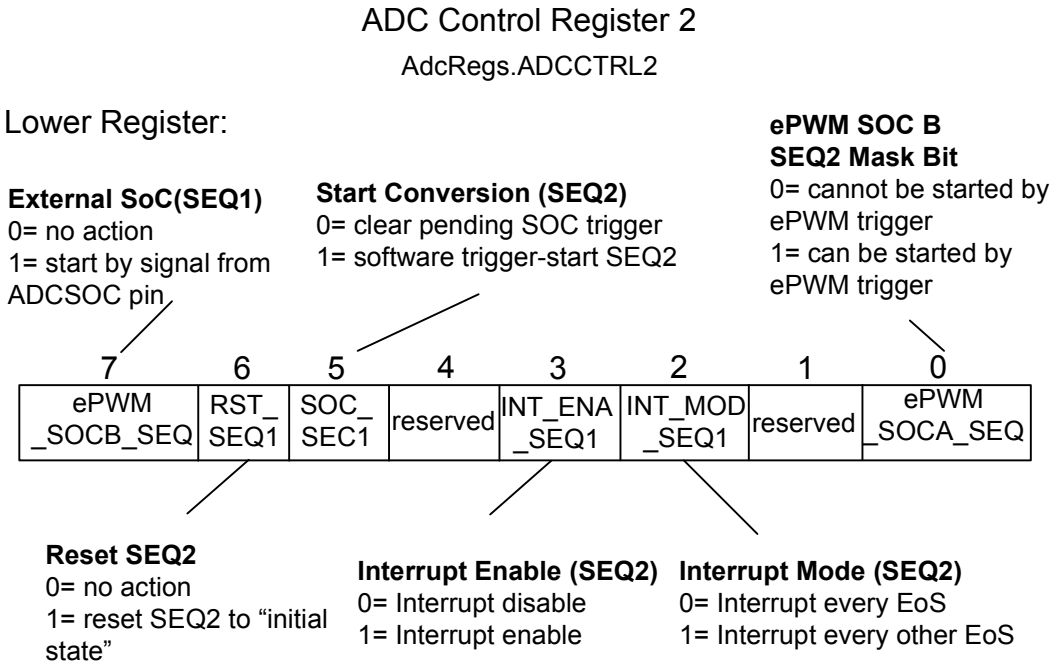
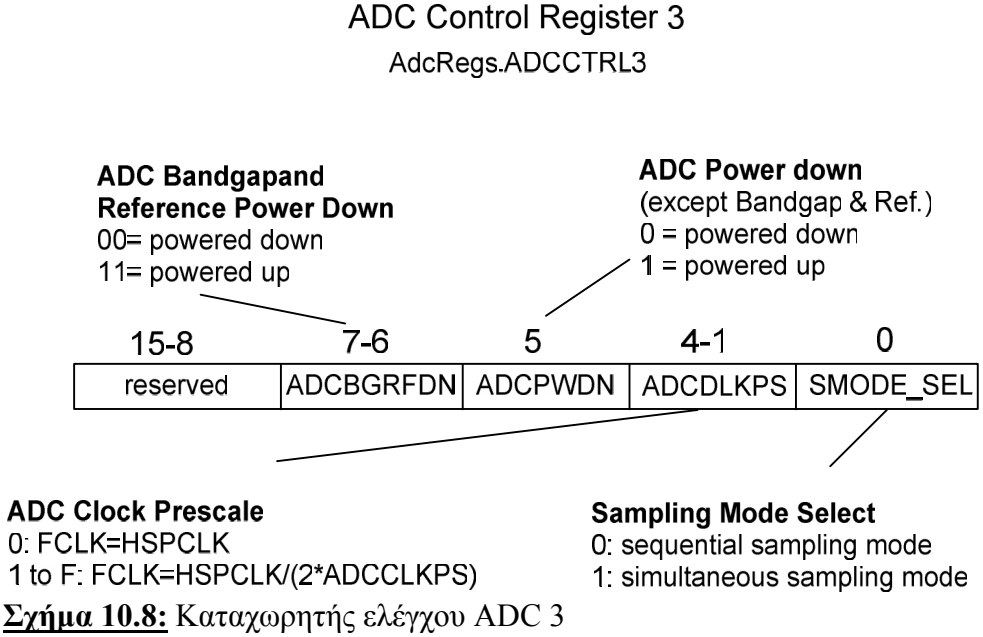
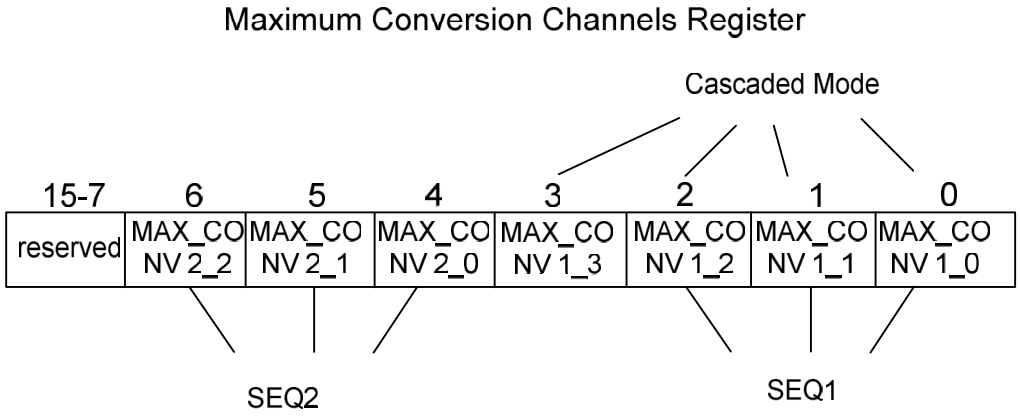
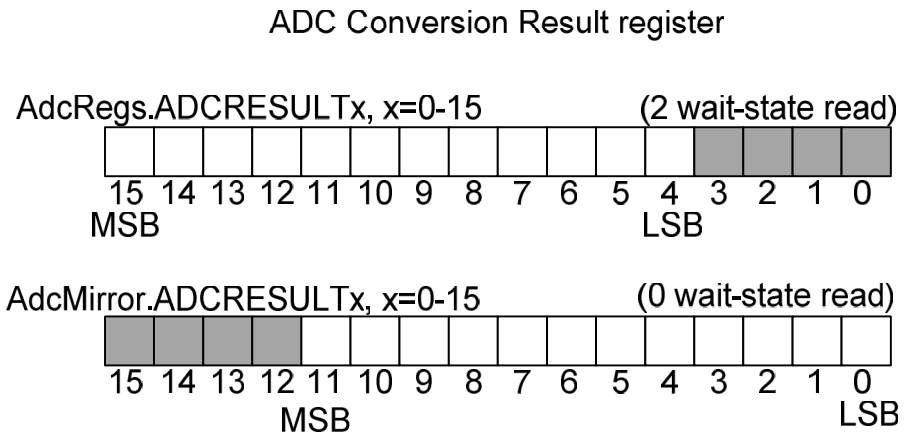
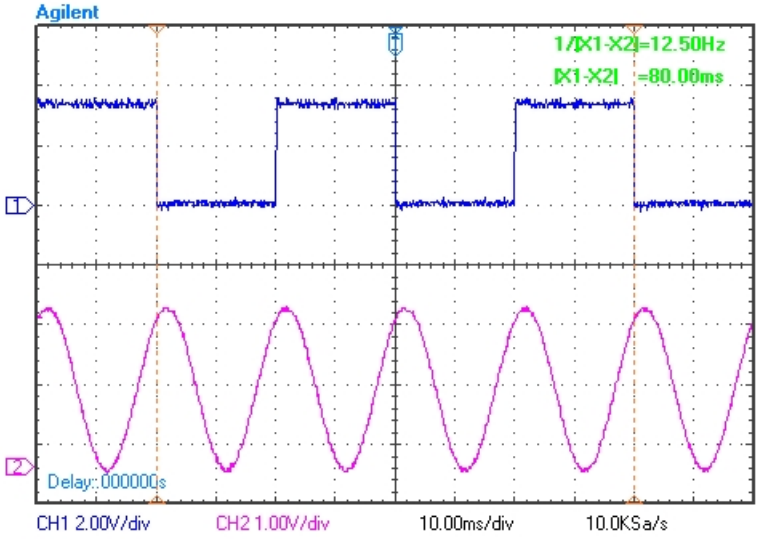
    CMPA_buff[index]=(EPwm1Regs.TBPRD*1.0-
EPwm1Regs.CMPA.half.CMPA*1.0)/(EPwm1Regs.TBPRD*1.0);

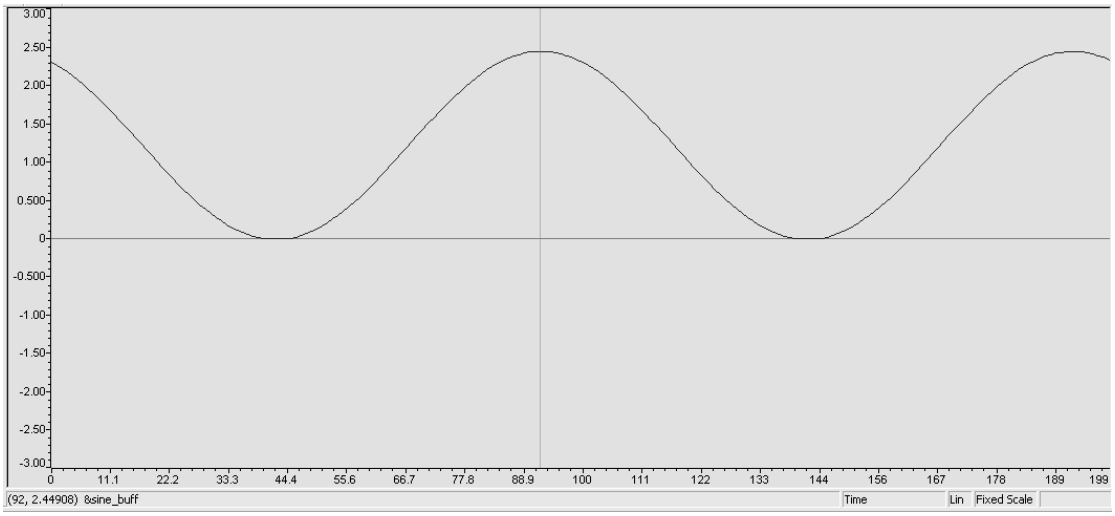
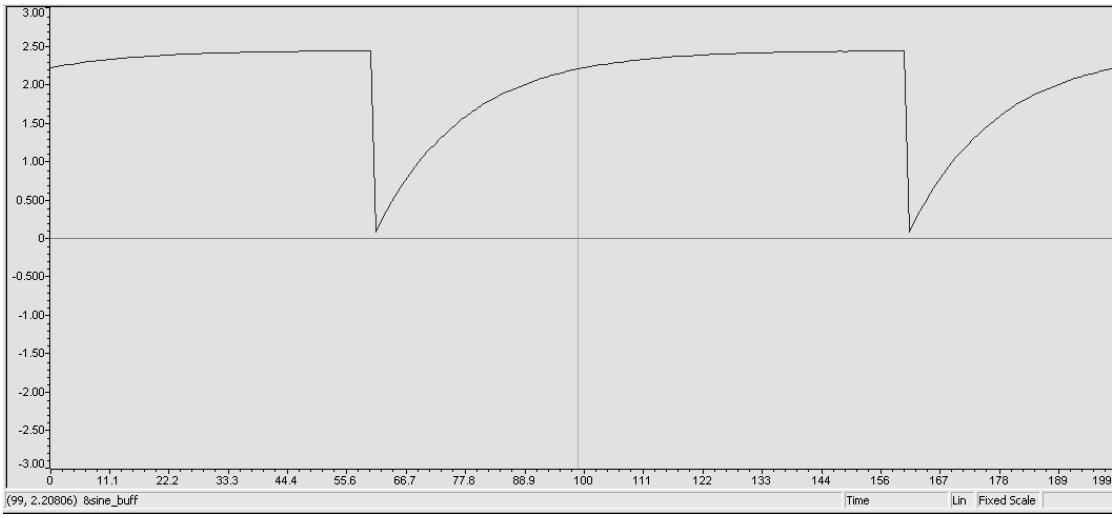
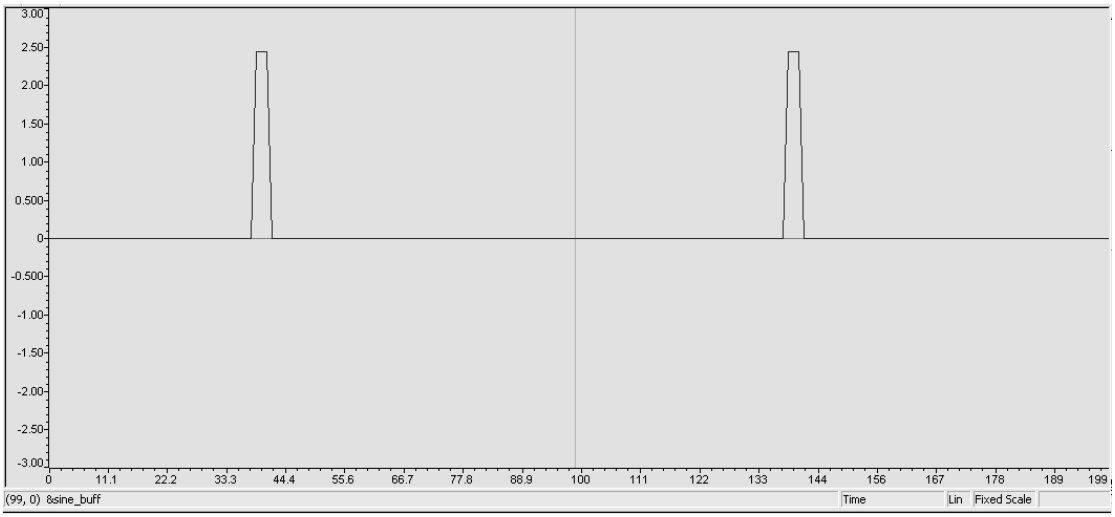
    switch (index)//PWM random sequence
        {
        case 0: EPwm1Regs.CMPA.half.CMPA=EPwm1Regs.TBPRD*0.125;
break;
        case 1: EPwm1Regs.CMPA.half.CMPA=EPwm1Regs.TBPRD*0.5; break;
        case 2: EPwm1Regs.CMPA.half.CMPA=EPwm1Regs.TBPRD*0.5; break;
        case 3: EPwm1Regs.CMPA.half.CMPA=EPwm1Regs.TBPRD*0.1; break;
        case 4: EPwm1Regs.CMPA.half.CMPA=EPwm1Regs.TBPRD*0.75;
index=-1; break;
        }

    index++;
    if (index==(period_pulse_count-2)) ECap1Regs.ECEINT.all = 0x0010;
    EPwm1Regs.ETCLR.bit.INT = 1;        // Clear ePWM1 Interrupt flag
    // Acknowledge this interrupt to receive more interrupts from
group 3
    PieCtrlRegs.PIEACK.all = 4;
}

interrupt void eCAP1_isr(void)
{
    ECap1Regs.ECCLR.bit.CEVT4 = 1;              // Clear the CEVT4
flag
    ECap1Regs.ECCLR.bit.INT = 1;                // Clear the ECAP1
interrupt flag
    // The sequence of the clearance is of outmost importance! For
further
```

```
    //research check eCAP datasheet page 16. We must first disable
the SET in the last
    //latch and THEN produce a CLEAR signal.
    //index2=index;


    PWM_Period = (int32)ECap1Regs.CAP4/2 +
(int32)ECap1Regs.CAP2/2+(int32)ECap1Regs.CAP3;
    DC_1st_Zero=(int32)ECap1Regs.CAP1*1.0/(PWM_Period*1.0);
    DC_1st_Pulse=(int32)ECap1Regs.CAP2*1.0/(PWM_Period*1.0);
    DC_2nd_Zero=(int32)ECap1Regs.CAP3*1.0/(PWM_Period*1.0);
    DC_2nd_Pulse=(int32)ECap1Regs.CAP4*1.0/(PWM_Period*1.0);

    //Attribute values to the right vector position. That's essential
if you have signals
    //which have periods not multiplied by 4.
    switch (2*period_pulse_count-index2-3-1)
        {
        case -1 :   *(DC_table+index2)=DC_1st_Zero;
                    *(DC_table+index2+1)=DC_1st_Pulse;
                    *(DC_table+index2+2)=DC_2nd_Zero;
                    *(DC_table+0)=DC_2nd_Pulse;
                    index2=1;
                    break;
        case -2 :   *(DC_table+index2)=DC_1st_Zero;
                    *(DC_table+index2+1)=DC_1st_Pulse;
                    *(DC_table+0)=DC_2nd_Zero;
                    *(DC_table+1)=DC_2nd_Pulse;
                    index2=2;
                    break;
        case -3 :   *(DC_table+index2)=DC_1st_Zero;
                    *(DC_table+0)=DC_1st_Pulse;
                    *(DC_table+1)=DC_2nd_Zero;
                    *(DC_table+2)=DC_2nd_Pulse;
                    index2=3;
                    break;
        case 0  :   *(DC_table+index2)=DC_1st_Zero;
                    *(DC_table+index2+1)=DC_1st_Pulse;
                    *(DC_table+index2+2)=DC_2nd_Zero;
                    *(DC_table+index2+3)=DC_2nd_Pulse;
                    index2=0;
                    break;
        default :   *(DC_table+index2)=DC_1st_Zero;
                    *(DC_table+index2+1)=DC_1st_Pulse;
                    *(DC_table+index2+2)=DC_2nd_Zero;
                    *(DC_table+index2+3)=DC_2nd_Pulse;
                    index2+=4;
        }
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP4; // Must acknowledge the
PIE group 4
}
//===========================================================================
// End of SourceCode.
//===========================================================================
```

ΚΕΦΑΛΑΙΟ 10

ΜΕΤΑΤΡΟΠΕΑΣ ΑΝΑΛΟΓΙΚΟΥ ΣΗΜΑΤΟΣ ΣΕ ΨΗΦΙΑΚΟ ΤΟΥ DSC TMS320F28335

10.1 Εισαγωγή

Ο F28335 διαθέτει ως περιφερειακή συσκευή τον **Μετατροπέα σήματος Αναλογικό-σε-Ψηφιακό**(Analogue-to-Digital Converter, ADC στο εξής). Τα περισσότερα φυσικά μεγέθη, όπως η θερμοκρασία, η ταχύτητα, το ρεύμα, η πίεση κ.α. μπορούν να μετρηθούν και να μετατραπούν σε επίπεδο τάσης μέσω κατάλληλων μορφοτροπέων (transducers). Το ADC module μετατρέπει την τάση εισόδου από έναν μορφοτροπέα (0 έως 3V) σε έναν ψηφιακό αριθμό 12-bit (0 έως 4095).

Ο τύπος που χρησιμοποιούμε για να μετατρέψουμε την ψηφιακή ένδειξη στην πραγματική τάση εισόδου είναι:

$$V_{in} = \frac{V^+ - V^- \cdot D}{2^n - 1} \Rightarrow V_{in} = \frac{3 \cdot D}{4095} \qquad (10.1)$$

, όπου $\mathbf{V^+}$ η μέγιστη τάση εισόδου (3V),
$\mathbf{V^-}$ η ελάχιστη (0V),
D η ψηφιακή στάθμη (η τάση εισόδου σε bit),
n η ανάλυση (12 bit)

Ο συγκεκριμένος μικροεπεξεργαστής υποστηρίζει **16 αναλογικές εισόδους**, αλλά μόνο δύο μπορούν να αναγνωστούν ταυτόχρονα. Το υλικό που είναι υπεύθυνο για την ανάγνωση και τη μετατροπή των εισόδων ονομάζεται **sample and hold unit** (S/H στο εξής). Ο μικρότερος χρόνος δειγματοληψίας που υποστηρίζεται είναι 80ns, ωστόσο η πρώτη ανάγνωση θα διαρκέσει 160ns.

Για την εκκίνηση της ανάγνωσης και μετατροπής των αναλογικών εισόδων είναι απαραίτητο να δοθεί πρώτα το κατάλληλο σήμα. Υπάρχουν τρεις τρόποι που μπορούν να ενεργοποιήσουν την αρχή της μετατροπής (Start-of-Conversion, στο εξής SoC):

1. Σήμα μέσω λογισμικού
2. Εξωτερικό σήμα σε GPIO
3. Μέσω ενός γεγονότος σε ένα από τα 6 κανάλια PWM

Στην εφαρμογή θα εξετάσουμε την τρίτη μόνο περίπτωση, καθώς αυτή επιτρέπει μεγάλη ακρίβεια όταν γνωρίζουμε εκ των προτέρων τη χρονική στιγμή που επιθυμούμε να πραγματοποιηθεί η δειγματοληψία.

10.2 Καταστάσεις λειτουργίας του ADC [1]

Το ADC module μπορεί να λειτουργεί σε πολλές διαφορετικές καταστάσεις. Μία κατάσταση λειτουργίας είναι συνδυασμός των παρακάτω τριών παραμέτρων:

Τρόπος Εναλλαγής:

- Κασκοδική εναλλαγή: Σχηματίζεται ένα αυτόματο* 16 καταστάσεων.
- Διπλή εναλλαγή: Σχηματίζονται δύο ανεξάρτητα αυτόματα 8 καταστάσεων έκαστο.

Τρόπος Δειγματοληψίας:

- Σειριακή: Μόνο μία μέτρηση λαμβάνεται ανά κάθε ανάγνωση.
- Παράλληλη: Στην παράλληλη δειγματοληψία χρησιμοποιούνται ταυτόχρονα οι δύο S/H μονάδες.

Τρόπος Εκκίνησης:

- Άπαξ: Όταν τελειώσει η ανάγνωση όλων των εισόδων, απαιτείται εκ νέου πυροδότηση του ADC.
- Συνεχόμενη: Όταν τελειώσει η ανάγνωση των εισόδων, η επόμενη ανάγνωση αρχίζει αυτόματα.

**Με τον όρο αυτόματο αναφερόμαστε στο αυτόματο πεπερασμένων καταστάσεων, όπως αυτό ορίζεται στη λογική σχεδίαση.*

10.2.1 Επεξήγηση της Κασκοδικής Εναλλαγής του ADC



Σχήμα 10.1: Δομικό διάγραμμα ADC module σε λειτουργία κασκοδικής εναλλαγής[1]

Στο Σχήμα 10.1 φαίνεται το διάγραμμα του ADC module όταν αυτό λειτουργεί με **κασκοδική εναλλαγή**. Πάνω αριστερά είναι οι **αναλογικές είσοδοι** οι οποίες οδηγούνται μέσω δύο πολυπλεκτών στον μετατροπέα 12bit. Το γκρι ορθογώνιο είναι η **ακολουθία** βάση της οποίας επιλέγεται ποιες είσοδοι διαβάζονται και πότε. Το **σήμα για την εκκίνηση** της διαδικασίας δίνεται από τα σήματα που βλέπουμε κάτω αριστερά. Στη δεξιά πλευρά του διαγράμματος είναι οι καταχωρητές στους οποίους οδηγούνται τα **ψηφιοποιημένα αποτελέσματα** και είναι αναγνώσιμα αφού δοθεί το σήμα «Τέλος μετατροπής» (End of Conversion, στο εξής EoC).

Μπορούμε να επιλέξουμε μεταξύ **Παράλληλης** και **Σειριακής** δειγματοληψίας. Στην περίπτωση της σειριακής, κάθε είσοδος διαβάζεται και μετατρέπεται ξεχωριστά από τις άλλες στο δικό της στάδιο στην ακολουθία, ενώ στην παράλληλη το ίδιο στάδιο της ακολουθίας εξυπηρετεί 2 εισόδους (πχ. οι είσοδοι ADCINA3 και ADCINB3 εξυπηρετούνται από το CON03). Για την **ανάγνωση των αποτελεσμάτων** καλούμε μία ειδική διακοπή (ADC interrupt) μετά το τέλος της μετατροπής (EoC).

10.2.2 Επεξήγηση της Διπλής Εναλλαγής του ADC



Σχήμα 10.2: Δομικό διάγραμμα ADC module σε λειτουργία διπλής εναλλαγής[1]

Ουσιαστικά, το Σχήμα 10.2 είναι ίδιο με το προηγούμενο διάγραμμα. Η διαφορά έγκειται στο ότι εδώ υπάρχουν δύο ακολουθίες. Η μία ελέγχει τις αναλογικές εισόδους ADCINAx ενώ η άλλη τις αναλογικές εισόδους ADCINBx. Οι ακολουθίες μεταξύ τους είναι ανεξάρτητες και μπορούν να ενεργοποιηθούν σε **ξεχωριστές χρονικές στιγμές**. Στην περίπτωση στην οποία συμπέσουν χρονικά η ακολουθία SEQ1 έχει προτεραιότητα σε σχέση με την ακολουθία SEQ2.

10.2.3 Χρόνος μετατροπής [1]



Note: Maximum F2833x ADCCLK is 25 MHz, but INL (integral nonlinearity error) is greater above 12.5 MHz. See the device datasheet (SPRU812A) for more information.

Σχήμα 10.3: Δομικό διάγραμμα συχνότητας για το ADC module [1]

Οι περιορισμοί για το χρόνο μετατροπής του ADC module είναι τρεις:

Ο πρώτος είναι το η συχνότητα που έχει ο **κρύσταλλος της πλακέτας**. Η συχνότητά του είναι δεδομένη και μετά από τα στάδια που φαίνονται στο Σχήμα 10.3 έχουμε SYSCLKOUT=150MHz.

Ο δεύτερος περιορισμός είναι η **μέγιστη συχνότητα για το FCLK**, η οποία είναι η βάση για την εξαγωγή της συχνότητας ADCCLK (συχνότητα με την οποία λειτουργεί η διοχέτευση ή pipeline του ADC module) και του παραθύρου δειγματοληψίας. Η FCCLK μπορεί να φτάσει μέχρι και τα 25MHz, αλλά στην πραγματικότητα μπορούμε να φτάσουμε μέχρι τα **12,5MHz** ώστε να βρισκόμαστε εντός της περιοχής στην οποία τα αποτελέσματα είναι **γραμμικά**.

Τέλος, ο τρίτος περιορισμός, είναι το **παράθυρο δειγματοληψίας** το οποίο ελέγχεται από το πεδίο bit “ACQ_PS”. Το πεδίο αυτό καθορίζει το χρονικό διάστημα μεταξύ της αλλαγής του πολυπλέκτη (από τη μία είσοδο στην επόμενη) μέχρι τη στιγμή που η είσοδος «παγώνει» και το δείγμα εισέρχεται στον μετατροπέα. Η προσαρμογή της τιμής αυτού του πεδίου εξαρτάται από την εκάστοτε διάταξη. Στα πλαίσια των εφαρμογών της διπλωματικής το παράθυρο δειγματοληψίας είναι αδιάφορο.

10.3 Καταχωρητές της μονάδας ADC [1]

Οι καταχωρητές του ADC module φαίνονται επιγραμματικά στον παρακάτω πίνακα:

Πίνακας 10.1: Καταχωρητές ADC module

Όνομα	Περιγραφή
ADCCTRL1	ADC control register 1
ADCCTRL2	ADC control register 2
ADCCTRL3	ADC control register 3
ADCMAXCONV	ADC maximum Conversion Channel Register
ADCCHSELSEQ1	ADC Channel Select Sequencing Control Register 1
ADCCHSELSEQ2	ADC Channel Select Sequencing Control Register 2
ADCCHSELSEQ3	ADC Channel Select Sequencing Control Register 3
ADCCHSELSEQ4	ADC Channel Select Sequencing Control Register 4
ADCASEQSR	ADC Auto sequencer Status Register
ADCRESULT0	ADC Conversion Result Buffer Register 0
ADCRESULT1	ADC Conversion Result Buffer Register 0
ADCRESULT2	ADC Conversion Result Buffer Register 0
…	…
ADCRESULT15	ADC Conversion Result Buffer Register 0
ADCREFSEL	ADC Reference Select Register
ADCOFFTRIM	ADC Offset Trim Register
ADCST	ADC Status Flag Register

Παρακάτω παρουσιάζονται τα πεδία των καταχωρητών και επεξηγούνται, όπου αυτό κρίνεται απαραίτητο:



Σχήμα 10.4: Καταχωρητής ελέγχου ADC 1 (15-7 bit)

- **RESET**

Μπορεί να χρησιμοποιηθεί για να επαναφέρει το αυτόματο του ADC module στην αρχική του κατάσταση. Να σημειωθεί ότι δεν γίνεται η αρχικοποίηση του καταχωρητή αυτού στην ίδια εντολή με την οποία γίνεται το reset.

- **SUSMOD (Emulation Suspend Mode)**

Καθορίζει την αλληλεπίδραση του ADC με τον JTAG εξομοιωτή, όπως έχει σημειωθεί και στο κεφάλαιο που αφορά το ePWM.

- **ACQ_PS (Acquisition Time Prescale)**

Καθορίζει το μήκος του παράθυρου δειγματοληψίας.

- **CPS (Conversion Prescale)**

Χρησιμοποιείται για τη διαίρεση της συχνότητας FCLK με 1 ή 2.



Σχήμα 10.5: Καταχωρητής ελέγχου ADC 1 (6-0 bit)

- **CONT_RUN (Continuous Run)**

Καθορίζει εάν η ακολουθία επαναληφθεί μετά το τέλος της (1), ή αν απαιτείται νέα διέγερση για να επαναληφθεί (0).



Σχήμα 10.6: Καταχωρητής ελέγχου ADC 2 (15-8 bit)

Το πάνω μισό του καταχωρητή Control Register 2 ελέγχει τη λειτουργία της ακολουθίας 1 (SEQ 1).

- **ePWM_SOCB_SEQ (Cascaded mode only)**

Επιτρέπει την έναρξη της μετατροπής μέσω σήματος από ePWM. Είναι διαθέσιμο μόνο για την κασκοδική εναλλαγή.

- **RST_SEQ1 (Reset SEQ1)**

Επαναφέρει άμεσα την Ακολουθία 1 στην αρχική κατάσταση.

- **SOC_SEQ1 (Start of Conversion SEQ1)**

Αρχή μετατροπής μέσω λογισμικού.

- **INT_ENA_SEQ1 (interrupt enable) INT_MOD_SEQ1 (Interrrupt mode)**

Ρύθμιση των διακοπών ADC. Αυτές είναι απαραίτητες για την ανάκτηση των εισόδων σε ψηφιακή μορφή. Μπορούμε να επιλέξουμε να καλούμε τη διακοπή κάθε EoS ή κάθε δεύτερη EoS.

- **ePWM_SOCA_SEQ1**

Επιτρέπει στο σήμα SOCA από το ePWM να αποτελέσει διέγερση για την αρχή μετατροπής.



Σχήμα 10.7: Καταχωρητής ελέγχου ADC 2 (7-0 bit)

- **EXT_SOC_SEQ1**

Επιτρέπει την ενεργοποίηση της Αρχής Μετατροπής (SoC) να γίνει μέσω εξωτερικού σήματος σε ένα από τα GPIO pin.

Τα υπόλοιπα πεδία είναι αντίστοιχα του πάνω μισού, αλλά για τη δεύτερη ακολουθία (SEQ2).



Σχήμα 10.8: Καταχωρητής ελέγχου ADC 3

- **SMODE_SEL (Sampling Mode Select)**

Πεδίο που επιλέγει μεταξύ παράλληλης ή σειριακής δειγματοληψίας. Για παράδειγμα εάν θέλουμε στο πρόγραμμά μας να μετατρέψουμε ταυτόχρονα το σήμα στις θύρες A4 και B4, κάνουμε την εξής αρχικοποίηση:

```
SMODE_SEL=1;        // simultaneous sampling
MAXCONV=0;          // One conversion
CONV00=4;           // Channel number for ADCINA4
```

Μετά το τέλος της μετατροπής το αποτέλεσμα για την A4 βρίσκεται στον καταχωρητή RESULT0 και της B4 στον RESULT1.

- **ADCCLKPS (ADC Clock Prescale)**
  Αρχικοποίηση του FCLK.

- **ADCPWDN (ADC Power Down)**
  Διακόπτης λειτουργίας του αναλογικού κυκλώματος της πλακέτας.

- **ADCBGRFDN (ADC Bandgap and Reference Power Down)**
  Ρυθμίσεις που αφορούν το εσωτερικό σύστημα τάσης αναφοράς.

10.3.1 ADC MAXCONV Καταχωρητής

Ο συγκεκριμένος καταχωρητής ορίζει τον αριθμό των σταδίων της ακολουθίας μετατροπής. Για παράδειγμα το MACCONV=4 σημαίνει ότι θα γίνουν 5 μετατροπές και τα στάδια της ακολουθίας θα είναι αντίστοιχα τα CONV00 έως CONV04. Στην **κασκοδική** μετατροπή χρησιμοποιούνται τα 4 πρώτα bit του καταχωρητή.



Σχήμα10.9: Καταχωρητής Maximum Conversion

Στην περίπτωση που έχει επιλεγεί **διπλή εναλλαγή**, μεταβάλλεται η λειτουργία του καταχωρητή. Όπως φαίνεται και στο σχήμα ο μέγιστος αριθμός μετατροπών για την Ακολουθία 1 καθορίζεται από τα bit 0-2 και για την Ακολουθία 2 από τα bit 4-6. Και για αυτή την περίπτωση ο τελικός αριθμός μετατροπών ισούται με (δυαδικός+1)

Στον παρακάτω πίνακα φαίνονται οι αρχικές και τελικές καταστάσεις για τους δυο διαφορετικούς τρόπους εναλλαγής καταστάσεων.

Πίνακας 10.2: Αρχικές και τελικές καταστάσεις για τους δύο τρόπους εναλλαγής

	SEQ1	SEQ2	Cascaded
Initial State	CONV00	CONV08	CONV00
End State	CONV07	CONV15	CONV15

ADC Input Channel Select Sequencing Control Register

	15-7	11-8	7-4	3-0
ADCCHSELSEQ1	CONV03	CONV02	CONV01	CONV00
ADCCHSELSEQ2	CONV07	CONV06	CONV05	CONV04
ADCCHSELSEQ3	CONV11	CONV10	CONV09	CONV08
ADCCHSELSEQ4	CONV15	CONV14	CONV13	CONV12

ADC input channels are binary counted	
ADCINA0 = 0000	ADCINB0 = 1000
ADCINA1 = 0001	...
…	ADCINB7 = 1111

Σχήμα 10.10 : Καταχωρητής ελέγχου καναλιών εισόδου

Οι καταχωρητές ADCCHSELSEQx χρησιμοποιούνται για την αντιστοίχιση εισόδων (ADCINx) σε στάδια της ακολουθίας μετατροπής. Οπότε, αν για παράδειγμα θέλουμε να μετατρέψουμε κατά σειρά τις εξής εισόδους ADCINA3, ADCINB4, ADCINB5, ADCINA0 πρέπει να γίνουν οι εξής αναθέσεις:

```
AdcRegs.ADCCHSELSEQ1.bit.CONV00=4;
AdcRegs.ADCCHSELSEQ1.bit.CONV01=12;
AdcRegs.ADCCHSELSEQ1.bit.CONV01=13;
AdcRegs.ADCCHSELSEQ1.bit.CONV01=0;
```



Σχήμα 10.11: Καταχωρητής Αποτελεσμάτων

Η μορφή που έχουν τα αποτελέσματα όταν αποθηκεύονται στους καταχωρητές RESULT έχουν τη μορφή του παραπάνω σχήματος. Συνήθως χρησιμοποιείται η δεύτερη μορφή (mirror) καθώς δεν απαιτεί ολίσθηση 4 bit σε σχέση με την πρώτη.

10.4 Εφαρμογή: Αναλογικός Παλμογράφος

Στόχος αυτής της εφαρμογής είναι η αξιοποίηση του ADC module του F28335 με σκοπό την απεικόνιση τυχαίων κυματομορφών τάσεως οι οποίες εισάγονται σαν είσοδο στα pin αναλογικής εισόδου της αναπτυξιακής πλακέτας.

Ο τρόπος διέγερσης της SoC γίνεται μέσω του ePWM2. Όπως τονίσαμε παραπάνω, αυτός είναι ο πιο αξιόπιστος τρόπος, όταν γνωρίζουμε επακριβώς τις χρονικές στιγμές στις οποίες επιθυμούμε να δειγματοληπτήσουμε την είσοδο.

Για την παραγωγή του σήματος εισόδου, χρησιμοποιήθηκε μια γεννήτρια αναλογικού σήματος του εργαστηρίου. Το εύρος του περιορίστηκε στην εμβέλεια των 0-3V. Ωστόσο, ο ίδιος κώδικας θα μπορούσε να αποτυπώσει και τάσεις υψηλότερες (πχ του δικτύου ΧΤ) αλλά θα ήταν απαραίτητη κατάλληλη διάταξη υποβιβασμού της τάσης.

Η δειγματοληψία της εισόδου ADCINA1 καθορίζεται από τον τύπο:

$$sampling\_frequency = F_{input} \cdot samples\_per\_period \qquad (10.2)$$

Στο τέλος κάθε διαδικασίας μετατροπής προκαλείται διακοπή ADC και έχουμε τη δυνατότητα ανάγνωσης της νέας τιμής ADCRESULT0. Η τιμή αυτή **μετατρέπεται** σε μορφή αντίστοιχη της πραγματικής τάσης μέσω του (10.1) και **αποθηκεύεται** στις θέσεις ενός πίνακα μεγέθους samples_per_period ώστε η ανανέωσή του να είναι περιοδική. Στη συνέχεια με τη λειτουργία graph αποτυπώνουμε στην οθόνη τα δεδομένα του πίνακα.

Να σημειωθεί ότι η επεξεργασία και αποθήκευση των δεδομένων γίνεται σε διαφορετική ΡΕΔ. Κάτι τέτοιο δεν είναι απαραίτητο, ωστόσο θα χρησιμοποιηθεί στη συνέχεια και συγκεκριμένα στην εφαρμογή του SPWM. Επίσης στο συγκεκριμένο παράδειγμα έχουν ενεργοποιηθεί δύο είσοδοι (ADCINA1 και ADCINA2). Η δεύτερη δε χρησιμοποιείται, απλά υπάρχει για χάριν παραδείγματος.

10.4.1 Πειραματικά Αποτελέσματα



Σχήμα 10.12: Έλεγχος συχνότητας γεννήτριας σημάτων στον παλμογράφο

Η ροζ κυματομορφή είναι το ημίτονο εισόδου με πλάτος από 0 έως 2,5Vκαι με συχνότητα 50Hz. Η μπλε παλμοσειρά είναι μία GPIO έξοδος η οποία αναστρέφεται κάθε 100 κλήσεις της ΡΕΔ της ePWM1 γραμμής. Σκοπός αυτής της εξόδου είναι να επιβεβαιώσει τη συμφωνία στη συχνότητα μεταξύ γεννήτριας και DSP. Το κομμάτι κώδικα που χρησιμοποιήθηκε είναι το εξής:

```
interrupt void epwm2_isr(void)

      {…
      index++;
      if (index==100) index=0;
      if (index==50) GpioDataRegs.GPATOGGLE.bit.GPIO9=1;
      ...}
```

Οι παρακάτω εικόνες δημιουργήθηκαν στο CCS (Code Composer Studio) και αντιστοιχούν σε πραγματική τάση.



Σχήμα 10.13: Ημίτονο 50Hz όπως αποτυπώθηκε στο Code Composer Studio (CCS)

Προφανώς οποιαδήποτε κυματομορφή με συχνότητα 50Hz μπορεί να αναπαρασταθεί από τον «παλμογράφο».



Σχήμα 10.14: Εκθετική αύξηση όπως αποτυπώθηκε στο Code Composer Studio (CCS)



Σχήμα 10.15: Παλμοί εύρους 600us όπως αποτυπώθηκε στο Code Composer Studio (CCS)

Για το Σχήμα 10.15 τα 600us ήταν η μικρότερη δυνατή διάρκεια παλμών την οποία μπορεί να ανιχνεύσει η συγκεκριμένη έκδοση του προγράμματος. Για την ανίχνευση παλμών μικρότερης διάρκειας πρέπει να αυξηθεί η δειγματοληψία.

10.4.2 Κώδικας με σχόλια

```
#include "DSP2833x_Device.h"
#include "ePWM_function.h"
#include "IQmathLib.h"
#include <math.h>
#include <stdlib.h>
// external function prototypes
extern void InitAdc(void);
extern void InitSysCtrl(void);
extern void InitPieCtrl(void);
extern void InitPieVectTable(void);
extern void InitCpuTimers(void);
extern void ConfigCpuTimer(struct CPUTIMER_VARS *, float, float);
extern void display_ADC(unsigned int);
// Prototype statements for functions found within this file.
void Gpio_select(void);
void ADC_EPWM_setup(void);
interrupt void cpu_timer0_isr(void);
interrupt void adc_isr(void);        // ADC  End of Sequence ISR
interrupt void epwm2_isr(void);
// Global Variables
unsigned int Voltage_VR1,sine_int_buff[100];
unsigned int Voltage_VR2;
float float_adc=0.0,sine_buff[100];
int index=0, samples_per_period=100;
long int sampling_frequency=0, input_signal_frequency=50;
PWM_parameters par=PWM_parameters_def;

//###########################################################################
//                         main code
//###########################################################################
void main(void)
```

```
{
    InitSysCtrl();  // Basic Core Init from DSP2833x_SysCtrl.c
    EALLOW;
    SysCtrlRegs.WDCR= 0x00AF;   // Re-enable the watchdog
    EDIS;
    DINT;               // Disable all interrupts
    Gpio_select();
    InitPieCtrl();            //  basic  setup  of  PIE  table;  from
DSP2833x_PieCtrl.c
    InitPieVectTable(); // default ISR's in PIE
    InitAdc();          // Basic ADC setup, incl. calibration

    ADC_EPWM_setup();
    EALLOW;
    PieVectTable.ADCINT = &adc_isr;
    PieVectTable.EPWM2_INT=&epwm2_isr;
    EDIS;

    PieCtrlRegs.PIEIER1.bit.INTx6 = 1;// ADC, enable adc interrupt
    PieCtrlRegs.PIEIER3.bit.INTx2=1; //ePWM2 IER activation
    IER |=5;

    EINT;
    ERTM;
    while(1)
    {

        {
            EALLOW;
            SysCtrlRegs.WDKEY = 0x55;
            EDIS;
        }

    }
}

void Gpio_select(void)
{
      EALLOW;
      GpioCtrlRegs.GPAMUX1.all = 0
      GpioCtrlRegs.GPAMUX2.all = 0;
      GpioCtrlRegs.GPBMUX1.all = 0;
      GpioCtrlRegs.GPBMUX2.all = 0;
      GpioCtrlRegs.GPCMUX1.all = 0;
      GpioCtrlRegs.GPCMUX2.all = 0;
      GpioCtrlRegs.GPADIR.all = 0;
      GpioCtrlRegs.GPADIR.bit.GPIO9 = 1;
      GpioCtrlRegs.GPADIR.bit.GPIO11 = 1;
      GpioCtrlRegs.GPBDIR.all = 0;
      GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1;
      GpioCtrlRegs.GPBDIR.bit.GPIO49 = 1;
      GpioCtrlRegs.GPCDIR.all = 0;
      EDIS;
}

void ADC_EPWM_setup(void)
{

    AdcRegs.ADCTRL1.all = 0;
    AdcRegs.ADCTRL1.bit.ACQ_PS = 7;
```

```
 // 7 = 8 x ADCCLK, Acquisition time prescale (length of sample
window)
    AdcRegs.ADCTRL1.bit.SEQ_CASC =1;
 // 1=cascaded sequencer 2*8 state machine. This is the feature that
dictates 2 inputs
    AdcRegs.ADCTRL1.bit.CPS = 0;           //divide by 1   ("faster"
input frequency)
    AdcRegs.ADCTRL1.bit.CONT_RUN = 0;     // single run mode, waits
another trigger at the end of the sequence

    AdcRegs.ADCTRL2.all = 0;
    AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;     // 1=enable   SEQ1 ADC
interrupt
    AdcRegs.ADCTRL2.bit.EPWM_SOCA_SEQ1 =1;    // 1=SEQ1 start from
ePWM_SOCA trigger for input A
    AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1 = 0;     // 0= interrupt after
every end of sequence

    AdcRegs.ADCTRL3.bit.ADCCLKPS = 3;   // ADC clock: FCLK = HSPCLK /
2 * ADCCLKPS
// HSPCLK = 75MHz (see DSP2833x_SysCtrl.c)
// FCLK = 12.5 MHz

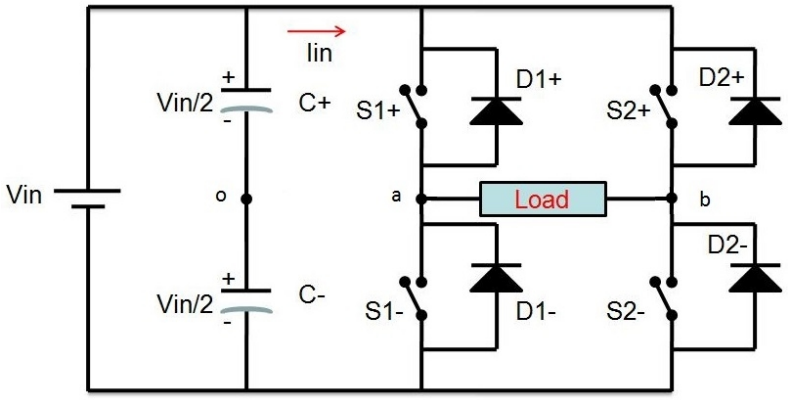
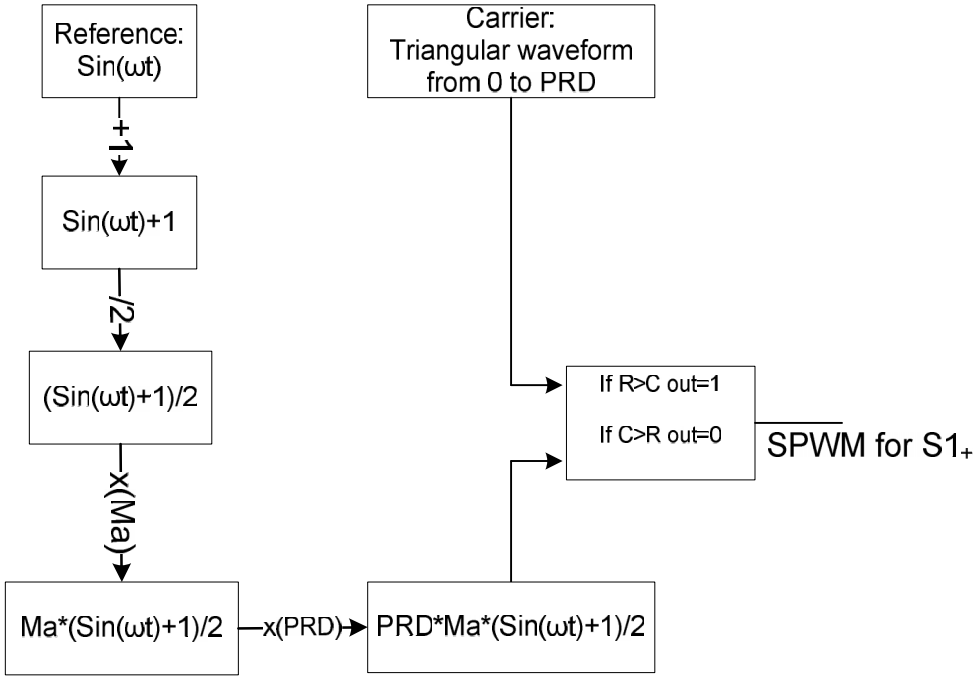
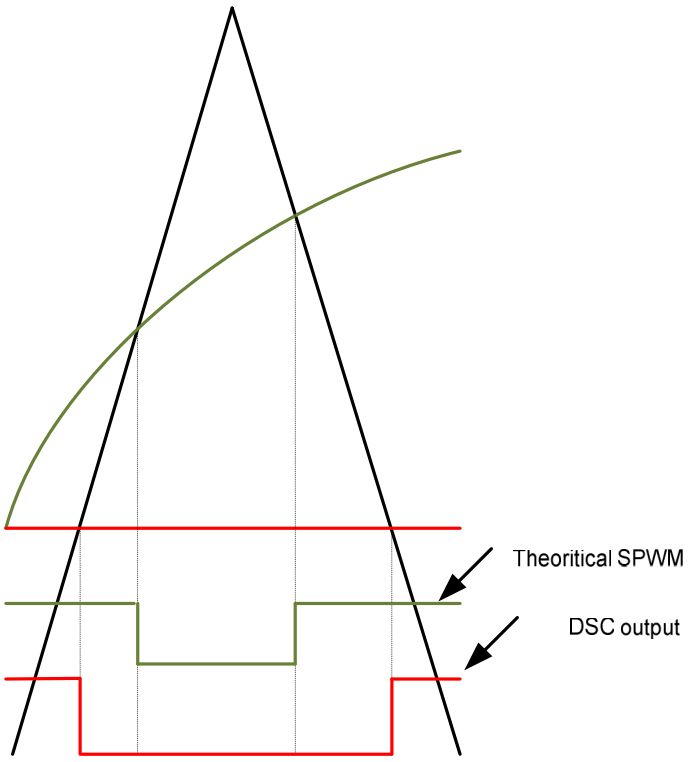
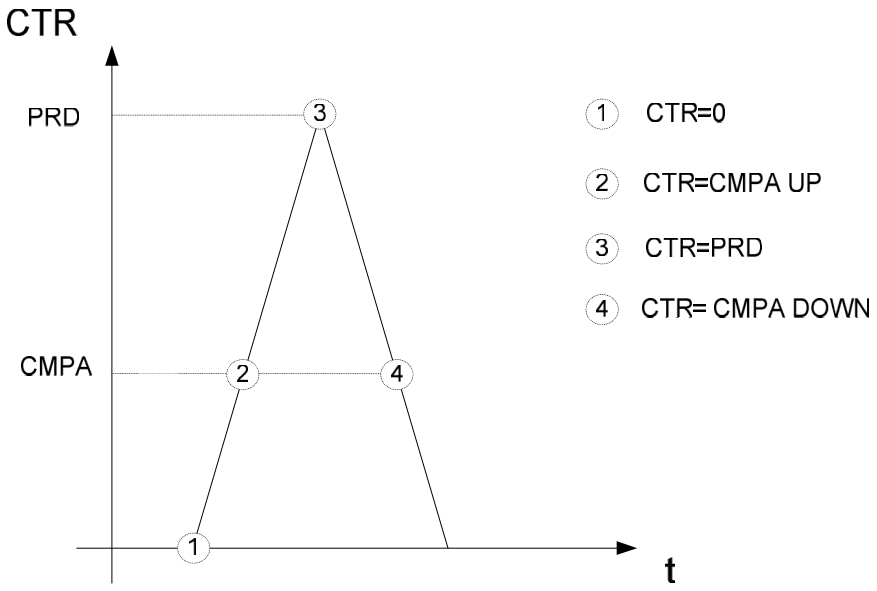
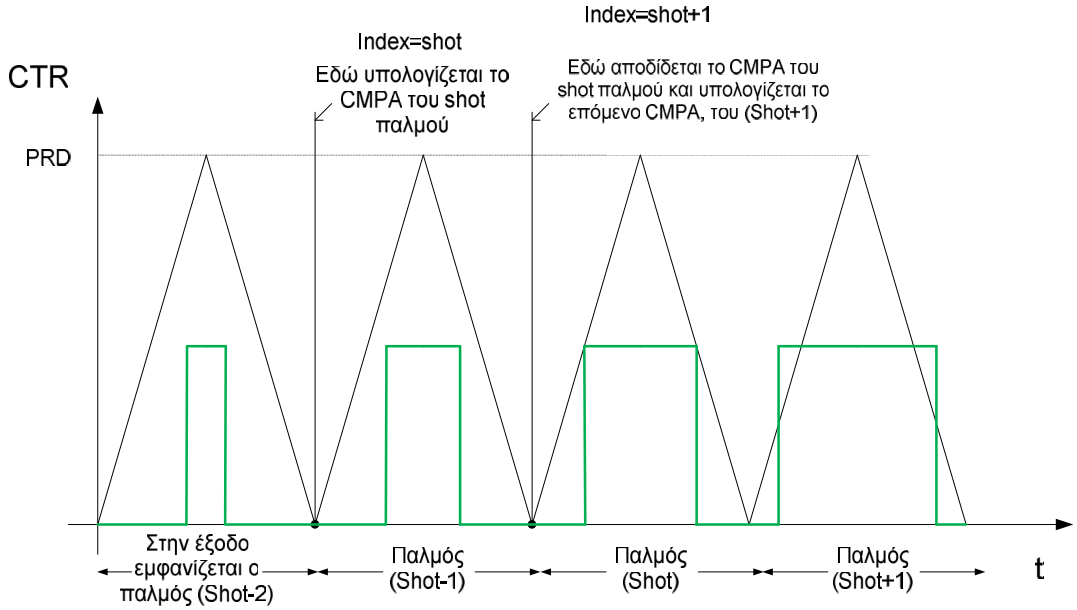
    AdcRegs.ADCMAXCONV.all = 0x0001;       // 2 conversions from
Sequencer 1
    AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0; // Setup ADCINA0 as 1st SEQ1
conv. Here is the initialization of ADCINs
    AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 1; // Setup ADCINA1 as 2nd SEQ1
conv.
    EPwm2Regs.TBCTL.all = 0xC030;     // Configure timer control
register
    /*
     bit 15-14     11:     FREE/SOFT, 11 = ignore emulation suspend
     bit 13        0:      PHSDIR, 0 = count down after sync event
     bit 12-10     000:    CLKDIV, 000 => TBCLK = HSPCLK/1
     bit 9-7       000:    HSPCLKDIV, 000 => HSPCLK = SYSCLKOUT/1
     bit 6         0:      SWFSYNC, 0 = no software sync produced
     bit 5-4       11:     SYNCOSEL, 11 = sync-out disabled
     bit 3         0:      PRDLD, 0 = reload PRD on counter=0
     bit 2         0:      PHSEN, 0 = phase control disabled
     bit 1-0       00:     CTRMODE, 00 = count up mode
    */

    sampling_frequency=samples_per_period*input_signal_frequency;
    scan_for_PWM_solutions(63000,10,sampling_frequency,&par);

    EPwm2Regs.TBCTL.bit.CLKDIV = par.clk_div;
    EPwm2Regs.TBCTL.bit.HSPCLKDIV = par.hsp_clk_div;
    EPwm2Regs.TBPRD = par.tb_prd;
    EPwm2Regs.ETPS.all = 0x0101;           // Configure ADC start by
ePWM2
    /*
     bit 15-14     00:     EPWMxSOCB, read-only
     bit 13-12     00:     SOCBPRD, don't care
     bit 11-10     00:     EPWMxSOCA, read-only
     bit 9-8       01:     SOCAPRD, 01 = generate SOCA on first event
     bit 7-4       0000:   reserved
     bit 3-2       00:     INTCNT, don't care
     bit 1-0       01:     INTPRD, interrupt on first event
    */
```

```
    EPwm2Regs.ETSEL.all = 0x0A09;           // Enable SOCA to ADC
    /*                                      //Enable Interrupt
     bit 15        0:      SOCBEN, 0 = disable SOCB
     bit 14-12     000:    SOCBSEL, don't care
     bit 11        1:      SOCAEN, 1 = enable SOCA
     bit 10-8      010:    SOCASEL, 010 = SOCA on PRD event
     bit 7-4       0000:   reserved
     bit 3         1:      INTEN, 0 = enable interrupt
     bit 2-0       001:    INTSEL, PRD=0
    */
}
interrupt void epwm2_isr(void)
{
    EALLOW;
    SysCtrlRegs.WDKEY = 0xAA;   // service WD #2
    EDIS;

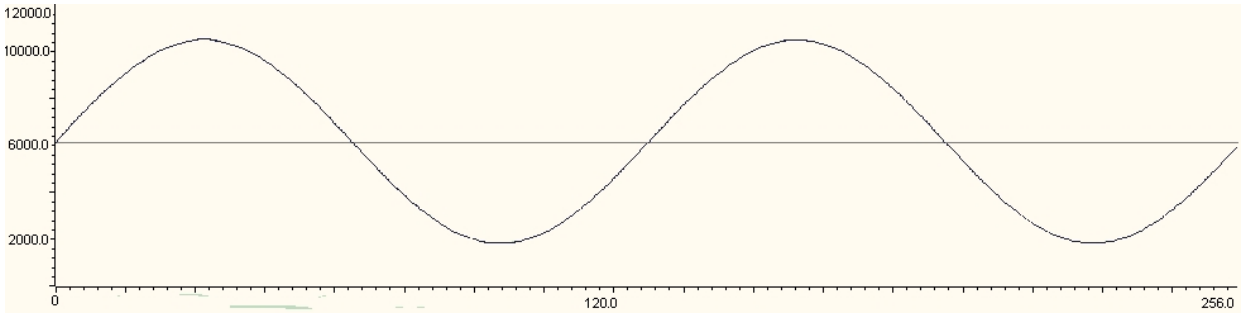
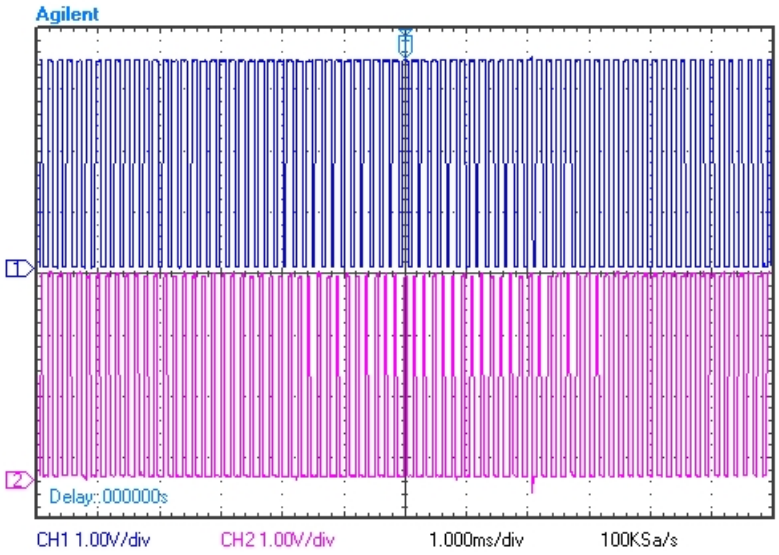
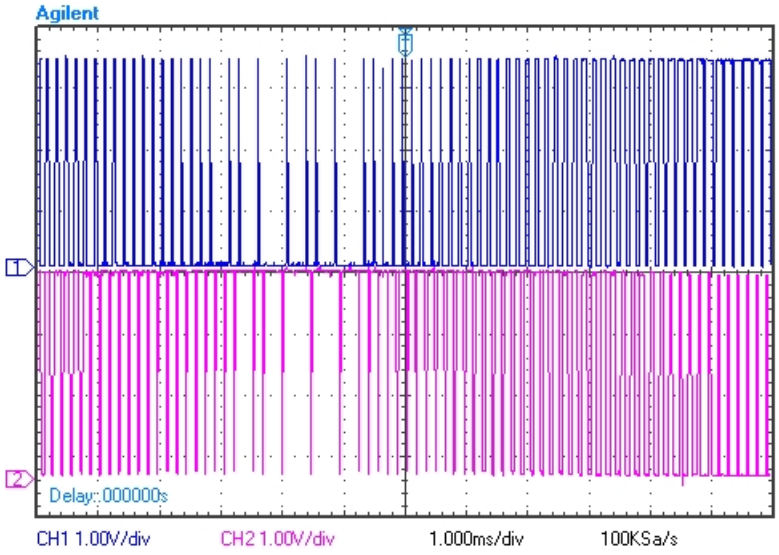
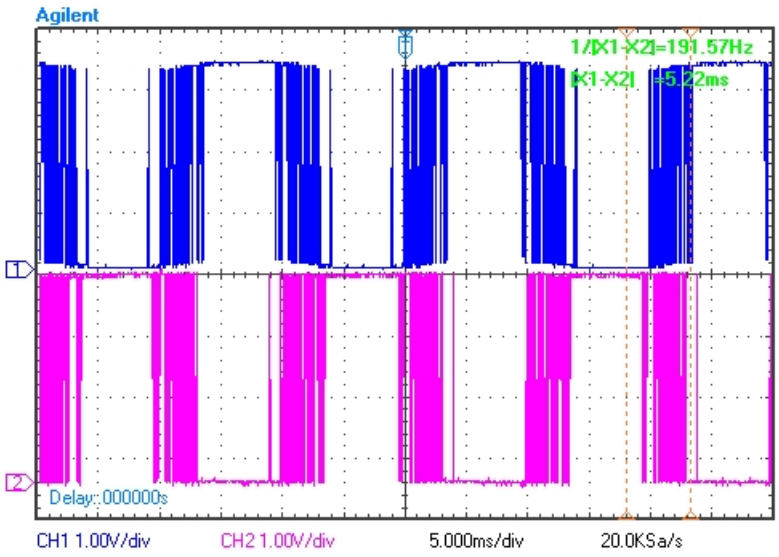
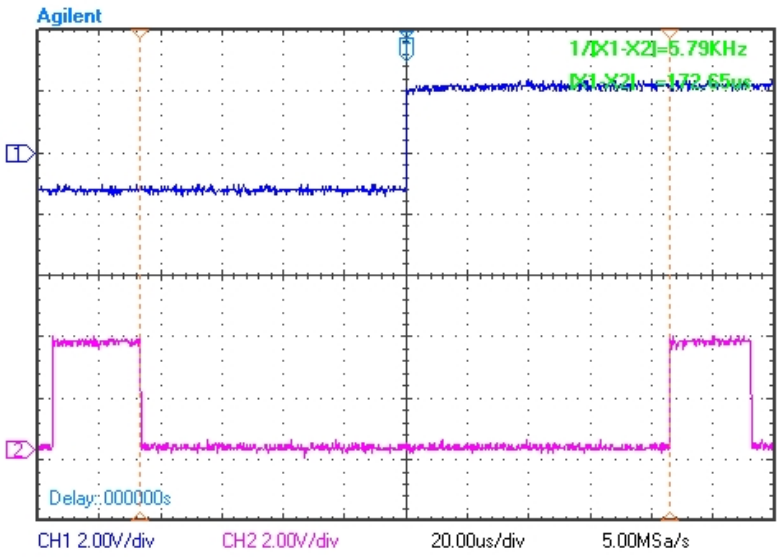
    float_adc=(Voltage_VR1*1.0/4095.0)*3.0;
    sine_buff[index]=float_adc;
    sine_int_buff[index]=Voltage_VR1;
    index++;
    if (index==100) index=0;
    if (index==50) GpioDataRegs.GPATOGGLE.bit.GPIO9=1;
    EPwm2Regs.ETCLR.bit.INT=1; //Clear ePWM2 Interrupt flag
    PieCtrlRegs.PIEACK.all=4; //Acknowledge this interrupt to receive
more interrupts for group3
}

interrupt void  adc_isr(void) // ADC interrupt service routine
{
    Voltage_VR1 = AdcMirror.ADCRESULT0; // store results global
    Voltage_VR2 = AdcMirror.ADCRESULT1;// It saves the mirror so in
reality the  right shift 8 (in adc display) means that we
    //display the 0000 xxxx(max15) number
    // Reinitialize for next ADC sequence
    AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;          // Reset SEQ1 state
machine to its initial state
    AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;     // Clear INT SEQ1 bit
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1; // Acknowledge interrupt
to PIE
}

//===========================================================================
// End of SourceCode.
//===========================================================================
```

10.5 Βιβλιογραφία

[1] «TMS320F28335 Analog to digital converter (ADC) Module» available at http://www.ti.com /

[2] «TMS320F28335 Data Manual» available at http://www.ti.com /

ΚΕΦΑΛΑΙΟ 11

ΥΛΟΠΟΙΗΣΗ ΜΟΝΟΦΑΣΙΚΟΥ SPWM ΣΤΟΝ TMS320F28335

11.1 Εισαγωγή

Στο κεφάλαιο 3 αναλύθηκε η θεωρία της τεχνικής SPWM. Με αυτό το θεωρητικό υπόβαθρο και συνδυάζοντας τις προγραμματιστικές γνώσεις των προηγούμενων κεφαλαίων, θα αναπτυχθεί κώδικας που θα παράγει SPWM με μεταβλητό συντελεστή διαμόρφωσης πλάτους m_a.



Σχήμα 11.1: Διάταξη στην οποία εφαρμόζεται μονοφασικό SPWM

Το δομικό διάγραμμα του προγράμματος για τους παλμούς οδήγησης του ημιαγωγού $S1_+$ είναι το ακόλουθο:



Σχήμα 11.2: Δημιουργία παλμών για την οδήγηση του $S1_+$

Παρατηρούμε ότι σε σχέση με το κανονική τεχνική SPWM υπάρχει μία διαφοροποίηση. Το σήμα αναφοράς και το σήμα φέροντος μεταβάλλονται από 0 έως PRD (για m_a=1). Η ανάγκη για αυτή την τροποποίηση πηγάζει από τον τρόπο με τον οποίο υλοποιείται το τριγωνικό σήμα φέροντος στον DSC. Όπως έχει εξηγηθεί και στα κεφάλαια του ePWM, ο τριγωνικός φορέας είναι στην ουσία ένας μετρητής το εύρος του οποίου εκτείνεται στους μη μηδενικούς αριθμούς. Πιο συγκεκριμένα για μετρητή 16 bit το εύρος του είναι 0 έως 65535 (2^{16}-1). Οπότε οι παραπάνω τροποποιήσεις στην αναφορά είναι απαραίτητες για να έχουμε τα ίδια αποτελέσματα.

Όπως και στη θεωρία, οι πύλες «-» δέχονται τους αντίστροφους παλμούς από τις πύλες των «+» ημιαγωγών, ενώ η πύλη του ημιαγωγού $S2_+$ δέχεται σήμα, το οποίο εξάγεται από το σχήμα 11.2 αλλά με αρχική αναφορά:

$$V_{ref}=Sin(\omega t-180^o)$$

Το m_a δίνεται σαν τάση εισόδου στο A0 pin (ADC pin) του DSC. Η τάση είναι από 0 έως 3V και με πράξεις ανάγεται σε πραγματικό αριθμό από 0 έως 1.

Η άλλη διαφοροποίηση του θεωρητικού SPWM με αυτό που πρακτικά παράγεται από τα DSC είναι η εξής:



Σχήμα 11.3: Διαφορά θεωρητικού με πραγματικού SPWM

Εξ' ανάγκης παρατηρείται μία διαφοροποίηση στα σημεία τομής του σήματος φορέα με την αναφορά. Αυτό οφείλεται στο ότι ο καταχωρητής CMPA αλλάζει την τιμή του στην αρχή κάθε περιόδου του σήματος φέροντος (βλ. Κεφ. 7). Σε κάθε περίπτωση το σφάλμα που εν τέλει δημιουργείται είναι αμελητέο.

11.2 Η θέση της διακοπής σε on-the-fly εφαρμογές PWM

Όπως έχουμε αναφέρει και στο κεφάλαιο 7 του PWM τα διαθέσιμα σημεία για διακοπή είναι τέσσερα και φαίνονται στο παρακάτω σχήμα (Σχήμα 11.4).



Σχήμα 11.4: Διαθέσιμα σημεία Διακοπής PWM

Προφανώς το βέλτιστο σημείο για να γίνει ο υπολογισμός της επόμενης τιμής είναι η αρχή του προηγούμενο κύκλου (Σχήμα 11.4). Έτσι η ΡΕΔ έχει τη δυνατότητα να τρέχει για χρονικό διάστημα ίσο με την περίοδο του φέροντος. Δίνοντας τη δυνατότητα στον προγραμματιστή να συμπεριλάβει διαδικασίες με μεγάλο υπολογιστικό κόστος, ή να ανεβεί σε υψηλότερες συχνότητες.



Σχήμα 11.5: Επεξήγηση της διαφοράς υπολογισμού/απόδοσης της τιμής του CMPA για ένα δεδομένο παλμό (shot)

Το σημείο το οποίο χρίζει προσοχής είναι η λειτουργία shadow του PWM. Όταν η λειτουργία αυτή είναι ενεργοποιημένη η απόδοση της νέας τιμής CMPA γίνεται σε προκαθορισμένα γεγονότα του PWM κύκλου και όχι άμεσα. Οπότε παρότι ο υπολογισμός γίνεται όταν CTR=0, η απόδοση γίνεται στην αρχή του επόμενου τριγώνου (Σχήμα 11.5).

Οπότε σε αυτή αλλά και στην επόμενη εφαρμογή θα επιλέξουμε η διακοπή ePWM να συμβαίνει όταν CTR=0.

11.3 Σημαντικά χωρία Κώδικα

- **Δημιουργία των σημάτων αναφοράς**

Υπάρχουν δύο τρόποι προσέγγισης σε αυτό το ζήτημα. Ο ένας είναι η δημιουργία ενός look-up table του οποίου το μέγιστο μέγεθος δηλώνεται στην αρχή και δε μεταβάλλεται. Ο δεύτερος τρόπος χρησιμοποιεί μεθόδους δυναμικής δέσμευσης μνήμης κατά την αρχή του προγράμματος. Το πλεονέκτημά του είναι ότι χρησιμοποιεί ακριβώς τη μνήμη που χρειάζεται η εφαρμογή ωστόσο προγραμματιστικά η υλοποίησή του είναι πιο απαιτητική:

Δυναμική δέσμευση μνήμης

```
sine_sampling=freq_carrier/freq_ref;
    wave= malloc(sine_sampling*sizeof(*wave)); //memory allocation of
wave
    for (i=0;i<sine_sampling;i++)
    {
      temp_float=i/(sine_sampling*1.0);
      temp_iq=_IQ28mpy(_IQ28mpy(_IQ28(2.0),_IQ28(PI)),_IQ28(temp_floa
      t));
      *(wave+i)=_IQ28sin(temp_iq);
    }
```

Δημιουργία look-up table

```
sine_sampling=freq_carrier/freq_ref;
    for (i=0;i<sine_sampling;i++)
        {
            temp_float=i/(sine_sampling*1.0);

            in1=_IQ28mpy(_IQ28mpy(_IQ28(2.0),_IQ28(PI)),_IQ28(temp_fl
            oat));
            in2

             sine_table2[i]=_IQ28sin(in1);
        }
```

Η τιμή του sine_sampling είναι ο συνολικός αριθμός τιμών που χρειάζονται από την αναφορά. Στην πρακτική υλοποίηση του SPWM θα χρειάζονταν 2*sine_sampling, κάτι που όπως φαίνεται από το σχήμα 11.3 δεν χρειάζεται για την υλοποίηση του προγράμματος.

- Υπολογισμός m_a.

```
temp_adc=_IQ4div(Voltage_VR1,4095);
float_VR1=_IQ28(_IQ4toF(temp_adc));
if (float_VR1<_IQ28(0.05))  float_VR1=_IQ28(0.05);
ma=float_VR1;
```

Οι εντολές αυτές βρίσκονται εντός της ΡΕΔ του ePWM πριν τον υπολογισμό των χρόνων χρησιμοποίησης. Η μεταβλητή Voltage_VR1 περιέχει την τιμή της τάσης εισόδου. Μετά από πράξεις μετατρέπεται σε πραγματικό αριθμό από 0 έως 1. Η τιμή αυτή αποδίδεται στην "ma" για πρακτικούς λόγους.

- **Υπολογισμός χρόνου χρησιμοποίησης ημιαγωγών**

Το παραπάνω κομμάτι κώδικα υλοποιεί ένα διάνυσμα τιμών που περιέχει τις τιμές του: sin(ωt). Η επεξεργασία της τιμής αυτής ώστε να συμβαδίζει με τον τελικό τύπο του σχήματος 11.2 γίνεται με τον παρακάτω κώδικα, ο οποίος βρίσκεται εντός της ΡΕΔ του ePWM:

```
    EPwm1Regs.CMPA.half.CMPA =
_IQsat(_IQ28mpy((_IQ28mpy(ma,*(wave+index))+_IQ28(1.0))/2,EPwm1Regs.T
BPRD),EPwm1Regs.TBPRD,0);
```

Αυτό ωστόσο, αποτελεί το σημείο τομής της αναφοράς με το φορέα για την ημιγέφυρα 1. Πολλαπλασιάζοντας την τιμή του αρχικού διανύσματος με «-1» έχουμε την αναφορά για την ημιγέφυρα 2:

```
   in4=_IQ28mpy(_IQ28(-1.0),*(wave+index));

   EPwm3Regs.CMPB =
_IQsat(_IQ28mpy((_IQ28mpy(ma,in4)+_IQ28(1.0))/2,EPwm1Regs.TBPRD),EPwm
1Regs.TBPRD,0);
```

Οι παλμοί για τους «-» δεν παράγονται από το DSC, αλλά προκύπτουν από τα σήματα εξόδου των GPIO0 ($S1_+$) και GPIO5($S2_+$) αν αυτά οδηγηθούν σε μία πύλη NOT.

- **Έλεγχος εύρους παλμών**

```
if (index==shot) GpioDataRegs.GPATOGGLE.bit.GPIO9 = 1;
```

Αυτή η εντολή βρίσκεται εντός της ΡΕΔ του ePWM. Αλλάζει την κατάσταση του GPIO9 κατά τον παλμό shot. Με αυτό τον τρόπο (και με κατάλληλες ρυθμίσεις του παλμογράφου) μπορούμε να εστιάσουμε σε έναν συγκεκριμένο παλμό της εξόδου ώστε να μετρήσουμε το πλάτος του. Να σημειωθεί ότι το παράδειγμα περιλαμβάνει συνάρτηση σύλληψης κυματομορφής η οποία όμως δεν είναι απαραίτητη για τη λειτουργία του SPWM.

Επιπλέον των αρχείων που έχουμε συμπεριλάβει στις ως τώρα εργασίες, για να χρησιμοποιήσουμε IQ_math αριθμούς πρέπει να προσθέσουμε το παρακάτω αρχείο:

C:\tidcs\C28\IQmath\v15a\lib\IQmath_fpu32.lib;

Και να προβούμε στην παρακάτω ρύθμιση:

- Πρέπει να παρέχουμε στον Μεταγλωττιστή τη διεύθυνση των φακέλων που συμπεριλάβαμε:

Project → Build Options → Compiler tab

Και στο Include Search Path συμπληρώνουμε:

C:\tidcs\C28\IQmath\v15a\include;

11.4 Πειραματικά Αποτελέσματα

- **Συχνότητα φέροντος 6,4KHz**

Στις παρακάτω εικόνες παρουσιάζονται τα αποτελέσματα του κώδικα SPWM για συχνότητα φέροντος ίσης με 6,4KHz

Στο σχήμα 11.6 απεικονίζεται για δύο περιόδους των 50Hz οι Χρόνοι χρησιμοποίησης του ημιαγωγού $S1_+$ και για m_a=0,72. Για τη συχνότητα των 6,4KHz το tbprd είναι ίσο με 11719 (στην εικόνα φαίνεται η τιμή 12000). Οπότε για m_a=0,72 ο η μέγιστη τιμή που θα πάρει ο CMPA καταχωρητής είναι 10078, κάτι το οποίο επιβεβαιώνεται ποιοτικά από το διάγραμμα



Σχήμα 11.6: Συντελεστές χρησιμοποίησης για το $S1_+$ με m_a=0,72

Τα σχήματα 11.7, 11.8 και 11.9 είναι οι παλμοσειρές ελέγχου των $S1_+$ και $S2_+$ για την ίδια συχνότητα και για διάφορα m_a.



Σχήμα 11.7: Παλμοσειρές οδήγησης $S1_+$ και $S2_+$ m_a= 0,42



Σχήμα 11.8: Παλμοσειρές οδήγησης $S1_+$ και $S2_+$ m_a =0,95



Σχήμα 11.9: Παλμοσειρές οδήγησης $S1_+$ και $S2_+$ m_a =1,35

11.5 Συχνότητα φέροντος 5KHz και επαλήθευση ορθότητας

Εργαστηριακά η συχνότητα αυτή ήταν προβληματική καθώς παρουσιαζόταν μία παλινδρομική κίνηση κατά τη μελέτη των εξόδων. Για αυτό το λόγο έπρεπε να εξεταστούν όλοι οι παραγόμενοι παλμοί ώστε να διαπιστωθεί αν υπήρχε κάποιο λάθος λογικής στο πρόγραμμα. Η μέθοδος που χρησιμοποιήθηκε περιγράφηκε παραπάνω στην επεξήγηση του κώδικα και εξετάστηκε ξεχωριστά κάθε ένας από τους 100 παλμούς της εξόδου που οδηγεί τον ημιαγωγό $S1_+$.

Το συμπέρασμα που εξήχθη από αυτή τη διαδικασία ήταν ότι η πλευρά του προγράμματος ήταν ορθή. Ο λόγος για τον οποίον προκαλείται αυτή η ανωμαλία δεν έχει εξακριβωθεί, αλλά μπορούμε να είμαστε βέβαιοι ότι το σήμα που εξέρχεται από τον DSC είναι απόλυτα σωστό.

Για τη συγκεκριμένη μελέτη μορφοποιήσαμε τον κώδικα ώστε η διακοπή για τον υπολογισμό και την απόδοση του νέου CMPA να γίνεται για CTR=PRD, δηλαδή στο μέσο του τριγώνου. Επίσης, για τον διακόπτη $S1_+$ εφαρμόζεται πρακτική αντίθετη του σχήματος 11.5, δηλαδή η κορυφή του τριγωνικού φορέα βρίσκεται στο μέσον του αρνητικού παλμού.

Στο σχήμα 11.10 υπάρχει ο εξηκοστός ένατος παλμός του SPWM. Η ακμή του μπλε παλμού (GPIO9), σηματοδοτεί το μέσον του τριγώνου, καθώς μέσα στη ΡΕΔ (η οποία καλείται για CTR=PRD) υπάρχει εντολή toggle για το GPIO9. Αυτό που ενδιαφέρει είναι το εύρος του αρνητικού παλμού της ροζ παλμοσειράς. Με τα μετρητικά μέσα του παλμογράφου βλέπουμε ότι το εύρος του παλμού είναι 172,65μs. Η ίδια διαδικασία συνεχίστηκε για τους υπόλοιπους 99 παλμούς.



Σχήμα 11.10: Μελέτη μεμονωμένου παλμού του SPWM

11.5.1 Υπολογισμοί SPWM 5KHz

Για τη συγκεκριμένη συχνότητα η περίοδος φέροντος είναι 200μs. Το οποίο μεταφράζεται σε TBPRD=15000. Αποτελεί τη μέγιστη τιμή που μπορεί να λάβει το CMPA.Τα αποτελέσματα από την μελέτη των 100 παλμών φαίνονται στον παρακάτω πίνακα. Το m_a είναι ίσο με 0,9.

Πίνακας 11.1: Σύγκριση θεωρητικών και πρακτικών τιμών του συντελεστή χρησιμοποίησης κάθε παλμού για συχνότητα φέροντος 5KHz

Triangle	Shot	CMPA value	Theoretical toff (us)	Real toff (us)	Variance	error (%)
0	1	7500	100	100	0,1	0
1	2	7876	94,98666667	95	0,1	-0,01404
2	3	8251	89,98666667	90	0,1	-0,01482
3	4	8624	85,01333333	85	0,1	0,015684
4	5	8992	80,10666667	80,1	0,1	0,008322
5	6	9354	75,28	75,3	0,1	-0,02657
6	7	9708	70,56	70,6	-0,1	-0,05669
7	8	10054	65,94666667	66	-0,1	-0,08087
8	9	10390	61,46666667	61,5	-0,1	-0,05423
9	10	10714	57,14666667	57,1	0,1	0,081661
10	11	11026	52,98666667	53	0	-0,02516
11	12	11324	49,01333333	49	0,1	0,027203
12	13	11607	45,24	45,2	0,1	0,088417
13	14	11873	41,69333333	41,7	0,1	-0,01599
14	15	12123	38,36	38,4	-0,1	-0,10428
15	16	12354	35,28	35,3	-0,1	-0,05669
16	17	12565	32,46666667	32,5	-0,1	-0,10267

17	18	12757	29,90666667	29,9	0,1	0,022292
18	19	12928	27,62666667	27,6	0,1	0,096525
19	20	13078	25,62666667	25,6	0,1	0,104058
20	21	13206	23,92	23,9	0,05	0,083612
21	22	13311	22,52	22,5	0,05	0,08881
22	23	13393	21,42666667	21,4	0,05	0,124456
23	24	13452	20,64	20,65	-0,05	-0,04845
24	25	13488	20,16	20,15	0,05	0,049603
25	26	13500	20	20	0,02	0
26	27	13488	20,16	20,16	0,02	0
27	28	13452	20,64	20,64	0,02	0
28	29	13393	21,42666667	21,42	0,02	0,031114
29	30	13311	22,52	22,52	0,02	0
30	31	13206	23,92	23,9	0,05	0,083612
31	32	13078	25,62666667	25,6	0,05	0,104058
32	33	12928	27,62666667	27,65	-0,05	-0,08446
33	34	12757	29,90666667	29,9	0,05	0,022292
34	35	12565	32,46666667	32,5	-0,05	-0,10267
35	36	12354	35,28	35,3	-0,05	-0,05669
36	37	12123	38,36	38,35	0,05	0,026069
37	38	11873	41,69333333	41,7		-0,01599
38	39	11607	45,24	45,25		-0,0221
39	40	11324	49,01333333	49	0,05	0,027203
40	41	11026	52,98666667	53	-0,05	-0,02516
41	42	10714	57,14666667	57,15	0,05	-0,00583
42	43	10390	61,46666667	61,5	-0,1	-0,05423
43	44	10054	65,94666667	66	-0,1	-0,08087
44	45	9708	70,56	70,6	-0,1	-0,05669
45	46	9354	75,28	75,3	-0,1	-0,02657
46	47	8992	80,10666667	80,1	0,1	0,008322
47	48	8624	85,01333333	85	0,1	0,015684
48	49	8251	89,98666667	90	0,1	-0,01482
49	50	7876	94,98666667	95	0,1	-0,01404
50	51	7499	100,0133333	100	0,1	0,013332
51	52	7123	105,0266667	105	0,1	0,02539
52	53	6748	110,0266667	110	0,1	0,024237
53	54	6375	115	115	0,2	-1,2E-14
54	55	6007	119,9066667	120	-0,2	-0,07784
55	56	5645	124,7333333	124,8	-0,2	-0,05345
56	57	5291	129,4533333	129,4	0,2	0,041199
57	58	4945	134,0666667	134,2	-0,2	-0,09945
58	59	4609	138,5466667	138,6	-0,2	-0,03849
59	60	4285	142,8666667	142,8	0,2	0,046664
60	61	3973	147,0266667	147	0,2	0,018137

61	62	3675	151	151	0,2	0
62	63	3392	154,7733333	154,8		-0,01723
63	64	3126	158,32	158,4	-0,2	-0,05053
64	65	2876	161,6533333	161,6	0,2	0,032992
65	66	2645	164,7333333	164,8	-0,2	-0,04047
66	67	2434	167,5466667	167,6	-0,2	-0,03183
67	68	2242	170,1066667	170,2	-0,2	-0,05487
68	69	2071	172,3866667	172,4	0,2	-0,00773
69	70	1921	174,3866667	174,4	0,2	-0,00765
70	71	1793	176,0933333	176	0,2	0,053002
71	72	1688	177,4933333	177,4	0,2	0,052584
72	73	1606	178,5866667	178,6	0,2	-0,00747
73	74	1547	179,3733333	179,4	+	-0,01487
74	75	1511	179,8533333	179,8	0,2	0,029654
75	76	1499	180,0133333	180	0,2	0,007407
76	77	1511	179,8533333	179,8	0,2	0,029654
77	78	1547	179,3733333	179,4	+	-0,01487
78	79	1606	178,5866667	178,6	0,2	-0,00747
79	80	1688	177,4933333	177,6	-0,2	-0,0601
80	81	1793	176,0933333	176	0,2	0,053002
81	82	1921	174,3866667	174,4	0,2	-0,00765
82	83	2071	172,3866667	172,4		-0,00773
83	84	2242	170,1066667	170	0,2	0,062706
84	85	2434	167,5466667	167,6	-0,2	-0,03183
85	86	2645	164,7333333	164,8	-0,2	-0,04047
86	87	2876	161,6533333	161,6	0,2	0,032992
87	88	3126	158,32	158,4	-0,2	-0,05053
88	89	3392	154,7733333	154,8		-0,01723
89	90	3675	151	151		0
90	91	3973	147,0266667	147	0,2	0,018137
91	92	4285	142,8666667	142,8	0,2	0,046664
92	93	4609	138,5466667	138,6	-0,2	-0,03849
93	94	4945	134,0666667	134	0,2	0,049727
94	95	5291	129,4533333	129,4	0,2	0,041199
95	96	5645	124,7333333	124,8	-0,2	-0,05345
96	97	6007	119,9066667	120	-0,2	-0,07784
97	98	6375	115	115	0,2	-1,2E-14
98	99	6748	110,0266667	110	0,2	0,024237
99	0	7123	105,0266667	105	0,2	0,02539

11.6 Κώδικας με σχόλια

```
#include "DSP2833x_Device.h"
#include "ePWM_function.h"
```

```
#include "IQmathLib.h"
#include <math.h>
#include <stdlib.h>
#define PI 3.14159265358979323846

_iq28 ma,in4;
_iq28 in1, in2;
_iq28 *wave;
_iq28 temp_iq,Va;
_iq4 temp_adc,temp;
// external function prototypes
extern void InitSysCtrl(void);
extern void InitPieCtrl(void);
extern void InitPieVectTable(void);
extern void InitAdc(void);
// Prototype statements for functions found within this file.
void Gpio_select(void);
void Setup_ePWM1(void);
void Setup_eCAP1(void);
void ADC_EPWM_setup(void);
interrupt void ePWM1A_compare_isr(void);
interrupt void adc_isr(void);
interrupt void eCAP1_isr(void);
int CMPA=0, CMPA_loc=0,index=0,counter=0, counter2=0;

long int ratio_loc,int1,int2, int3, int4=0;
int cdiv[8]={1,2,4,8,16,32,64,128};
int hspdiv[8]={1,2,4,6,8,10,12,14};
int CMPA_buff[512];
long int freq1= 25600;
long int freq_carrier=6400;
int freq_ref=50;
int clk_div=0,hsp_clk_div=0;
long int tb_prd=11719;
 _iq28 float_VR1=0.8;
float cpu_timer_interval;
float fl_1,fl_2,fl_3,temp_float;
long int i=0;
int sine_sampling,shot,period_pulse_count=5,index2=0;
Uint32 PWM_Duty;
Uint32 PWM_Period, Pulse_1, Pulse_2, old_cap1;
float DC_1st_Pulse,DC_2nd_Pulse,DC_2nd_Zero,DC_1st_Zero,* DC_table;
Uint16 Voltage_VR1,Voltage_VR2;
int samples_per_period=100;
long int sampling_frequency=0, input_signal_frequency=50;
PWM_parameters par=PWM_parameters_def;
//###########################################################################
//                         main code
//###########################################################################
void main(void)
{
    InitSysCtrl();  // Basic Core Init from DSP2833x_SysCtrl.c ,.,.

    EALLOW;
    SysCtrlRegs.WDCR= 0x00AF;   // Re-enable the watchdog
    EDIS;     // 0x00AF  to NOT disable the Watchdog, Prescaler = 64
    DINT;               // Disable all interrupts
    Gpio_select();
    InitAdc();
    Setup_ePWM1();      // init of ePWM1A
    EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD / 2;
```

```
    EPwm3Regs.CMPB = EPwm3Regs.TBPRD / 2;//EPwm3Regs.CMPA.half.CMPA =
EPwm3Regs.TBPRD / 2;
    EPwm1Regs.TBCTL.bit.SYNCOSEL = 1;   // generate a syncout if CTR
= 0

    EPwm2Regs.TBCTL.bit.PHSEN = 1;      // enable phase shift for
ePWM2
    EPwm2Regs.TBCTL.bit.SYNCOSEL = 0;   // syncin = syncout
    EPwm2Regs.TBPHS.half.TBPHS =0;  // 1/3 phase shift
    EPwm3Regs.TBCTL.bit.PHSEN = 1;      // enable phase shift for
ePWM3
    EPwm3Regs.TBPHS.half.TBPHS = 0; // 2/3 phase shift

    InitAdc();          // Basic ADC setup, incl. calibration
    ADC_EPWM_setup();

    InitPieCtrl();      // basic setup of PIE table; from
DSP2833x_PieCtrl.c

    InitPieVectTable(); // default ISR's in PIE
    EALLOW;
    PieVectTable.EPWM1_INT = &ePWM1A_compare_isr;
    PieVectTable.ADCINT = &adc_isr;
    PieVectTable.ECAP1_INT= &eCAP1_isr;
    EDIS;

       PieCtrlRegs.PIEIER3.bit.INTx1 = 1;  // Enable EPWM1_INT in PIE
group 3
    PieCtrlRegs.PIEIER4.bit.INTx1 = 1;  // Enable ECAP1_INT in PIE
group 4
    PieCtrlRegs.PIEIER1.bit.INTx6 = 1;  // Enable ADC_INT in PIE
group 4
    IER |=0x0005;           // enable INT3 for ePWM1

    EINT;
    ERTM;


    sine_sampling=freq_carrier/freq_ref;
    wave= malloc(sine_sampling*sizeof(*wave)); //memory allocation of
wave

    for (i=0;i<sine_sampling;i++)
    {
    temp_float=i/(sine_sampling*1.0);

temp_iq=_IQ28mpy(_IQ28mpy(_IQ28(2.0),_IQ28(PI)),_IQ28(temp_float));
    *(wave+i)=_IQ28sin(temp_iq);
    }


    while(1)
    {
        EALLOW;
        SysCtrlRegs.WDKEY = 0x55;   // service WD #1
        EDIS;
    }
}

void Gpio_select(void)
{
```

```
    EALLOW;
    GpioCtrlRegs.GPAMUX1.all = 0;       // GPIO15 ... GPIO0 = General
Purpose I/O
    GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1; // ePWM1A active
    GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 1; // ePWM3B active
    GpioCtrlRegs.GPAMUX2.all = 0;       // GPIO31 ... GPIO16 =
General Purpose I/O
    GpioCtrlRegs.GPAMUX2.bit.GPIO24= 1; // eCAP1 active
    GpioCtrlRegs.GPBMUX1.all = 0;       // GPIO47 ... GPIO32 =
General Purpose I/O
    GpioCtrlRegs.GPBMUX2.all = 0;       // GPIO63 ... GPIO48 =
General Purpose I/O
    GpioCtrlRegs.GPCMUX1.all = 0;       // GPIO79 ... GPIO64 =
General Purpose I/O
    GpioCtrlRegs.GPCMUX2.all = 0;       // GPIO87 ... GPIO80 =
General Purpose I/O

    GpioCtrlRegs.GPADIR.all = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO9 = 1;
GpioCtrlRegs.GPADIR.bit.GPIO11 = 1;

    GpioCtrlRegs.GPBDIR.all = 0;        // GPIO63-32 as inputs
    GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1;
    GpioCtrlRegs.GPBDIR.bit.GPIO49 = 1;    GpioCtrlRegs.GPCDIR.all =
0;        // GPIO87-64 as inputs
    EDIS;
}

void Setup_ePWM1(void)
{
    scan_for_PWM_solutions(63000,10,freq_carrier,&par);

    EPwm1Regs.TBCTL.bit.CLKDIV =  par.clk_div;  // CLKDIV = 1
    EPwm3Regs.TBCTL.bit.CLKDIV = par.clk_div ;
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = par.hsp_clk_div;    // HSPCLKDIV
= 1
    EPwm3Regs.TBCTL.bit.HSPCLKDIV = par.hsp_clk_div;
    EPwm1Regs.TBCTL.bit.CTRMODE = 2;    // up - down mode
    EPwm3Regs.TBCTL.bit.CTRMODE = 2;
    EPwm1Regs.AQCTLA.all = 0x0090;      // clear ePWM1A on CMPA up

    EPwm3Regs.AQCTLB.all = 0x0900;      // set ePWM1A on CMPA down
    EPwm1Regs.TBPRD = tb_prd;               // timer period for 500
KHz
    EPwm3Regs.TBPRD = tb_prd

    EPwm1Regs.CMPCTL.all=0x0000;
    EPwm3Regs.CMPCTL.all=0x0000;      //shadow mode ON
    EPwm1Regs.ETSEL.all = 0;
    EPwm1Regs.ETSEL.bit.INTEN = 1;      // interrupt enable for ePWM1
    EPwm1Regs.ETSEL.bit.INTSEL = 1;     // interrupt on CTR=PRD
    EPwm1Regs.ETPS.bit.INTPRD = 1;      // interrupt on first event

}
void Setup_eCAP1(void)
{
//-----------------------------------------------------------------
//--- Configure eCAP1 unit for capture
//-----------------------------------------------------------------
    ECap1Regs.ECEINT.all = 0;                   // Disable all eCAP
interrupts
```

```
    ECap1Regs.ECCTL1.bit.CAPLDEN = 0;            // Disabled loading
of capture results
    ECap1Regs.ECCTL2.bit.TSCTRSTOP = 0;          // Stop the counter

    ECap1Regs.TSCTR = 0;                         // Clear the counter
    ECap1Regs.CTRPHS = 0;                        // Clear the counter
phase register

    ECap1Regs.ECCTL2.all = 0x0096;
    ECap1Regs.ECCTL1.all = 0x01EE;
    ECap1Regs.ECEINT.all = 0x0010;               // Enable desired
eCAP interrupts
    }

void ADC_EPWM_setup(void)
{

    AdcRegs.ADCTRL1.all = 0;
    AdcRegs.ADCTRL1.bit.ACQ_PS = 7;      // 7 = 8 x ADCCLK
//Acquisition time prescale (length of sample window)
    AdcRegs.ADCTRL1.bit.SEQ_CASC =1;     // 1=cascaded sequencer
//2*8 state machine. This is the feature that dictates 2 inputs
    AdcRegs.ADCTRL1.bit.CPS = 0;         // divide by 1  ("faster"
input frequency)
    AdcRegs.ADCTRL1.bit.CONT_RUN = 0;    // single run mode, waits
another trigger at the end of the sequence

    AdcRegs.ADCTRL2.all = 0;
    AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;    // =enable  SEQ1 ADC
interrupt
    AdcRegs.ADCTRL2.bit.EPWM_SOCA_SEQ1 =1;  // 1=SEQ1 start from
ePWM_SOCA trigger for input A
    AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1 = 0;    // 0= interrupt after
every end of sequence

    AdcRegs.ADCTRL3.bit.ADCCLKPS = 3;    // ADC clock: FCLK = HSPCLK /
2 * ADCCLKPS
                                         // HSPCLK = 75MHz (see
DSP2833x_SysCtrl.c)
                                         // FCLK = 12.5 MHz

    AdcRegs.ADCMAXCONV.all = 0x0001;     // 2 conversions from
Sequencer 1

    AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0; // Setup ADCINA0 as 1st SEQ1
conv.
    AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 1; // Setup ADCINA1 as 2nd SEQ1
conv.

    EPwm2Regs.TBCTL.all = 0xC030;   // Configure timer control
register


    sampling_frequency=samples_per_period*input_signal_frequency;
    scan_for_PWM_asym_solutions(63000,10,sampling_frequency,&par);

    EPwm2Regs.TBCTL.bit.CLKDIV = par.clk_div;
    EPwm2Regs.TBCTL.bit.HSPCLKDIV = par.hsp_clk_div;
    EPwm2Regs.TBPRD = par.tb_prd;
```

```
    EPwm2Regs.ETPS.all = 0x0101;            // Configure ADC start by
ePWM2


    EPwm2Regs.ETSEL.all = 0x0A09;           // Enable SOCA to ADC
}


interrupt void ePWM1A_compare_isr(void)
{
    EALLOW;
    SysCtrlRegs.WDKEY = 0xAA;       // Service watchdog 1
    EDIS;
    if (index==shot) GpioDataRegs.GPATOGGLE.bit.GPIO9 = 1;

    temp_adc=_IQ4div(Voltage_VR1,4095);
    float_VR1=_IQ28(_IQ4toF(temp_adc));
    //if (float_VR1>_IQ28(0.95))    float_VR1=_IQ28(0.95);
    if (float_VR1<_IQ28(0.05))  float_VR1=_IQ28(0.05);

    ma=float_VR1;
      //ma=_IQ28(0.8); this line has been added, if we want to omit
      //the ADC calculation of magnitude.

    EPwm1Regs.CMPA.half.CMPA =

_IQsat(_IQ28mpy((_IQ28mpy(ma,*(wave+index))+_IQ28(1.0))/2,EPwm1Regs.T
BPRD),EPwm1Regs.TBPRD,0);


    in4=_IQ28mpy(_IQ28(-1.0),*(wave+index));
    EPwm3Regs.CMPB =
_IQsat(_IQ28mpy((_IQ28mpy(ma,in4)+_IQ28(1.0))/2,EPwm1Regs.TBPRD),EPwm
1Regs.TBPRD,0);


    CMPA_buff[index]=EPwm1Regs.CMPA.half.CMPA;

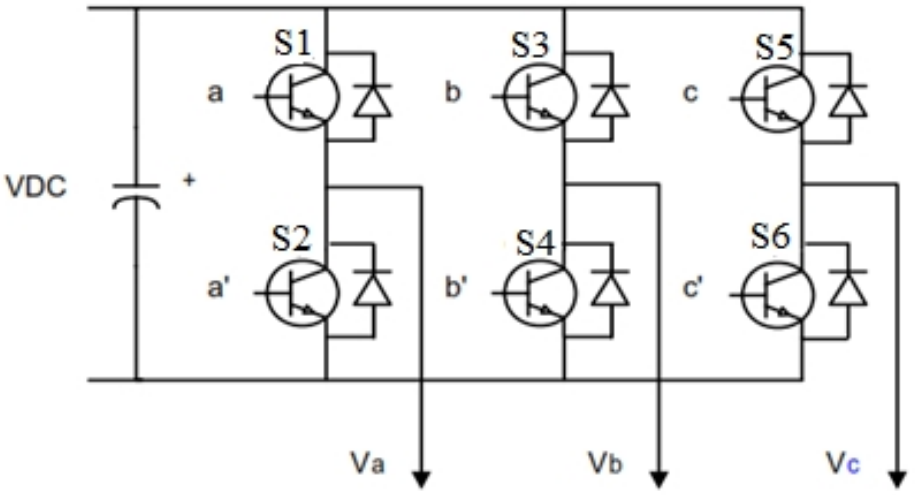
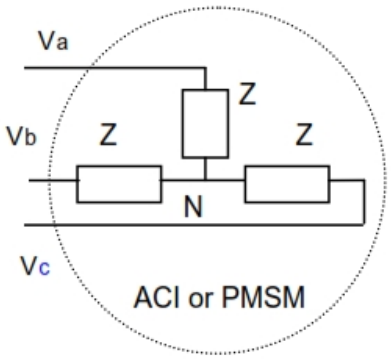
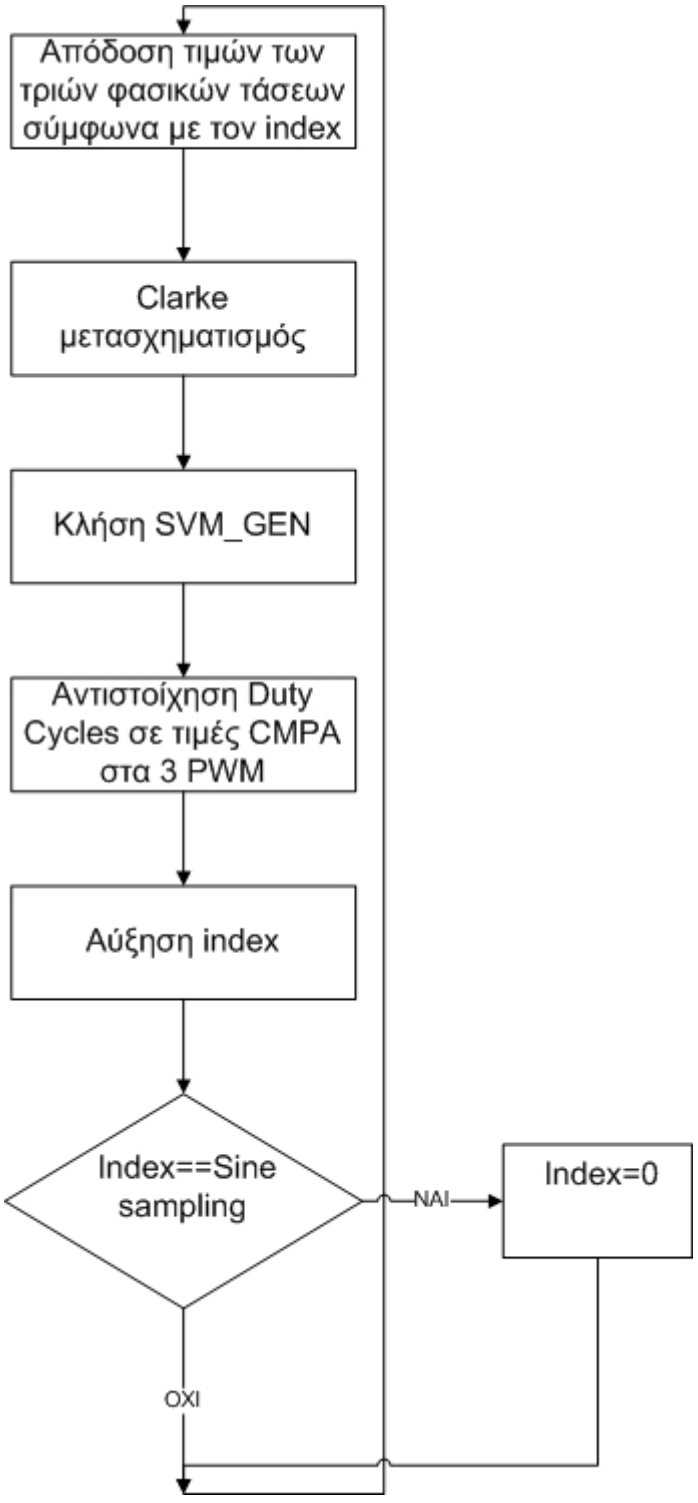
    index +=1;          // use next element out of lookup table
    if (index >(sine_sampling-1)) index = 0;

    EPwm1Regs.ETCLR.bit.INT = 1;        // Clear ePWM1 Interrupt flag

    // Acknowledge this interrupt to receive more interrupts from
group 3
    PieCtrlRegs.PIEACK.all = 4;

}

interrupt void  adc_isr(void) //ADC interrupt service routine
{
    Voltage_VR1 = AdcMirror.ADCRESULT0; // store results global
    Voltage_VR2 = AdcMirror.ADCRESULT1
    AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;       // Reset SEQ1, state
machine to its initial state
    AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;     // Clear INT SEQ1 bit
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1; // Acknowledge interrupt
to PIE
}

interrupt void eCAP1_isr(void)
```

```
{
    ECap1Regs.ECCLR.bit.CEVT4 = 1;              // Clear the CEVT4
flag
    ECap1Regs.ECCLR.bit.INT = 1;                // Clear the ECAP1
interrupt flag
        PWM_Period = (int32)ECap1Regs.CAP4/2 +
(int32)ECap1Regs.CAP2/2+(int32)ECap1Regs.CAP3;
    DC_1st_Zero=(int32)ECap1Regs.CAP1*1.0/(PWM_Period*1.0);
    DC_1st_Pulse=(int32)ECap1Regs.CAP2*1.0/(PWM_Period*1.0);
    DC_2nd_Zero=(int32)ECap1Regs.CAP3*1.0/(PWM_Period*1.0);
    DC_2nd_Pulse=(int32)ECap1Regs.CAP4*1.0/(PWM_Period*1.0);
     //Attribute values to the right vector position. That's
essential if you have signals
    //which have periods not multiplied by 4.
    switch (2*period_pulse_count-index2-3-1)
        {
        case -1 :   *(DC_table+index2)=DC_1st_Zero;
                    *(DC_table+index2+1)=DC_1st_Pulse;
                    *(DC_table+index2+2)=DC_2nd_Zero;
                    *(DC_table+0)=DC_2nd_Pulse;
                    index2=1;
                    break;
        case -2 :   *(DC_table+index2)=DC_1st_Zero;
                    *(DC_table+index2+1)=DC_1st_Pulse;
                    *(DC_table+0)=DC_2nd_Zero;
                    *(DC_table+1)=DC_2nd_Pulse;
                    index2=2;
                    break;
        case -3 :   *(DC_table+index2)=DC_1st_Zero;
                    *(DC_table+0)=DC_1st_Pulse;
                    *(DC_table+1)=DC_2nd_Zero;
                    *(DC_table+2)=DC_2nd_Pulse;
                    index2=3;
                    break;
        case 0  :   *(DC_table+index2)=DC_1st_Zero;
                    *(DC_table+index2+1)=DC_1st_Pulse;
                    *(DC_table+index2+2)=DC_2nd_Zero;
                    *(DC_table+index2+3)=DC_2nd_Pulse;
                    index2=0;
                    break;
        default :   *(DC_table+index2)=DC_1st_Zero;
                    *(DC_table+index2+1)=DC_1st_Pulse;
                    *(DC_table+index2+2)=DC_2nd_Zero;
                    *(DC_table+index2+3)=DC_2nd_Pulse;
                    index2+=4;
        }
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP4; // Must acknowledge the
PIE group 4
}
//===========================================================================
// End of SourceCode.
//===========================================================================
```

ΚΕΦΑΛΑΙΟ 12

ΥΛΟΠΟΙΗΣΗ SPACE VECTOR PWM (SVPWM) ΣΤΟΝ TMS320F28335

12.1 Εισαγωγή

Η τεχνική διαμόρφωσης Εύρους Παλμών με βάση τα Χωρικά Διανύσματα Τάσης του Αντιστροφέα (SVPWM) έχει ξεχωρίσει τα τελευταία χρόνια στις εφαρμογές των τριφασικών αντιστροφέων. Τα κύρια πλεονεκτήματά της είναι:

- Το μεγαλύτερο πλάτος της βασικής συνιστώσας της τάσης
- Η καλύτερη ποιότητα στην τάση εξόδου, δηλαδή μικρότερο THD (total harmonic distortion) σε σχέση με την SPWM

Σε συνέχεια του Κεφαλαίου 3, στο οποία θεμελιώθηκαν οι βασικές αρχές της SVPWM, ακολουθεί ο **αλγόριθμος παραγωγής παλμών για ένα τριφασικό αντιστροφέα**. Η συγκεκριμένη εφαρμογή είναι ανοιχτού βρόχου, ωστόσο ο ίδιος κώδικας μπορεί να χρησιμοποιηθεί σαν βάση για συστήματα με ανάδραση.



Σχήμα 12.1: Τριφασικός αντιστροφέας



Σχήμα 12.2: Τριφασικό συμμετρικό φορτίο

12.2 Διάγραμμα Ροής

Να σημειωθεί ότι το συγκεκριμένο διάγραμμα ροής αφορά την **ΡΕΔ της PWM διακοπής**. Πριν από αυτό προηγούνται στην main αρχικοποιήσεις για τα διάφορα module που χρησιμοποιούμε.

Συγκεκριμένα καθορίζεται τη λειτουργία των **GPIO**, του **PWM** και το διάνυσμα με τις τιμές του ημιτόνου αναφοράς (διάνυσμα μεγέθους sine_sampling). Στη συνέχεια το πρόγραμμα, οδηγείται σε έναν **ατέρμονο βρόχο while** και η **SVPWM παράγεται από τις περιοδικές διακοπές** στο κυρίως πρόγραμμα.

Η συνάρτηση SVPWM_gen είναι υπεύθυνη για τον **υπολογισμό των duty cycles των τριών ημιγεφυρών**. Συγκεκριμένα, δέχεται ως είσοδο τις τάσεις $\mathbf{U_{alpha}}$ **και** $\mathbf{U_{beta}}$ από το Μ/Σ Clarke, στη συνέχεια «αποφασίζει» σε ποιον τομέα βρίσκεται η αναφορά και αποδίδει τα κατάλληλα duty cycle ανά φάση. (Λεπτομερή σχόλια βρίσκονται στο παράρτημα)

Η συγκεκριμένη εφαρμογή είναι **ανοιχτού βρόχου** και ανά 50Hz πραγματοποιείται ένας πλήρης κύκλος του διανύσματος αναφοράς. Για λόγους πληρότητας έχει επιλεχθεί να δημιουργείται πρώτα το τριφασικό σύστημα αναφοράς και μετά αυτό να μετατρέπεται σε σύστημα (α,β) μέσω του Μ/Σ Clarke.

Στην περίπτωση που υπάρχει η ανάγκη για ανάδραση , τότε μπορεί να παρακαμφθεί τελείως η δημιουργία και η μετατροπή της τριφασικής αναφοράς και να εισάγονται στο πρόγραμμα σαν είσοδοι οι συνιστώσες της τάσεως αναφοράς U_{alpha}, U_{beta}.



12.3 Σημαντικά χωρία κώδικα

- ```
  sine_sampling=freq_carrier/freq_ref;
  period_carrier=1000000.0/(freq_carrier*1.0);
  phase_shift_240=sine_sampling*2/3;
  phase_shift_120=sine_sampling/3;
  for (i=0;i<sine_sampling;i++)
    {
    temp_float=i/(sine_sampling*1.0);

in1=_IQ28mpy(_IQ28mpy(_IQ28(2.0),_IQ28(PI)),_IQ28(temp_float));
    wave[i]=_IQ28sin(in1);
    }
  ```

**Συμπλήρωση του διανύσματος με τις τιμές του ημιτόνου**. Η χρησιμοποιούμενη ακρίβεια είναι IQ28 γιατί στον παραπάνω κώδικα υπάρχουν μερικά αθροίσματα μεγαλύτερα του 3.9999 τα οποία δεν μπορούν να αναπαρασταθούν με το IQ29 (βλ. Παράρτημα Β). Οι αριθμοί phase_shift_240 και phase_shift_120 υπολογίζονται για την εισαγωγή διαφοράς φάσης στον υπολογισμό των τιμών του ημιτόνου για τη φάση Β και C. Έτσι αποφεύγουμε τη δημιουργία επιπλέον διανυσμάτων.

Χρησιμοποιείται δυναμική δέσμευση μνήμης γιατί το μέγεθος των διανυσμάτων δεν είναι εκ των προτέρων γνωστό αλλά είναι ανάλογο της συχνότητας φορέα (sine_sampling). Με αυτόν τον τρόπο γίνεται πιο **αποδοτική χρησιμοποίηση** της , περιορισμένης, μνήμης

- ```
  EPwm1Regs.CMPCTL.all=0x0000; //CMPCTL=0x0000 => shadow mode ON,
  load on CTR=0
  ```

Ενεργοποίηση της λειτουργίας shadow και φόρτωση της νέας τιμής όταν CTR=0. Αυτό το βήμα είναι απαραίτητο ώστε να μη δημιουργηθούν προβλήματα πρόωρης ανάθεσης. Λεπτομερή αναφορά στο συγκεκριμένο θέμα γίνεται στο κεφάλαιο 7.

- ```
  Vb=wave[(index+phase_shift_240)%sine_sampling];
  ```

Δημιουργία της Β φάσης χρησιμοποιώντας το phase_shift_240 και τη συνάρτηση modulo.

- ```
  if (index ==sine_sampling) index = 0;
  ```

Συνθήκη μηδενισμού του index. Πρέπει να σημειωθεί ότι εφόσον χρησιμοποιείται παραπάνω το modulo για να αποφασιστεί σε ποια θέση του διανύσματος wave είμαστε, ο μηδενισμός αυτός πλεονάζει. Στην πράξη όμως παρατηρήθηκε ότι η συμπεριφορά του `index%sine_sampling` είναι ιδιόρρυθμη όταν το index πλησιάζει στην τιμή που θα κάνει overflow και θα επανέλθει στο μηδέν.

- ```
  clarke1.As=Va;
  clarke1.Bs=Vb;
  clarke1.calc(&clarke1);        //Input: Va,Vb Output: Vα,Vβ
  svgen_dq1.Ualpha=clarke1.Alpha;
  ```
  ```

```
      svgen_dq1.Ubeta=clarke1.Beta;
      svgen_dq1.calc(&svgen_dq1);          //Input: Vα,Vβ Output:
Ta,Tb,Tc(duty cycles of the three phases)
```

Καλούμε τις συναρτήσεις που εκτελούν Clarke, Park και τη διαδικασία που παρουσιάστηκε στο κεφάλαιο 3 (επιλογή τομέα του διανύσματος αναφοράς και έξοδος σε duty cycle ανα φάση). Τα δομές δεδομένων (struct) `clarke1.xxxx`, `svgen_dq1.xxxxx` είναι μεταβλητές οι οποίες αφορούν τις αντίστοιχες συναρτήσεις και δηλώνονται στα αντίστοιχα αρχεία .h, όπως φαίνονται παρακάτω.

```
typedef struct {  _iq  As;           // Input: phase-a variable
                  _iq  Bs;           // Input: phase-b variable
                  _iq  Alpha;        // Output: stationary d-axis
variable
                  _iq  Beta;         // Output: stationary q-axis
variable
                  void  (*calc)();   // Pointer to calculation
function
                } CLARKE;

typedef CLARKE *CLARKE_handle;

typedef struct  { _iq  Ualpha; // Input: reference alpha-axis phase
voltage
                  _iq  Ubeta;  // Input: reference beta-axis phase
voltage
                  _iq  Ta;     // Output: reference phase-a switching
function
                  _iq  Tb;     // Output: reference phase-b switching
function
                  _iq  Tc;     // Output: reference phase-c switching
function
                  void (*calc)();       // Pointer to calculation
function
                } SVGENDQ;

typedef SVGENDQ *SVGENDQ_handle;
```

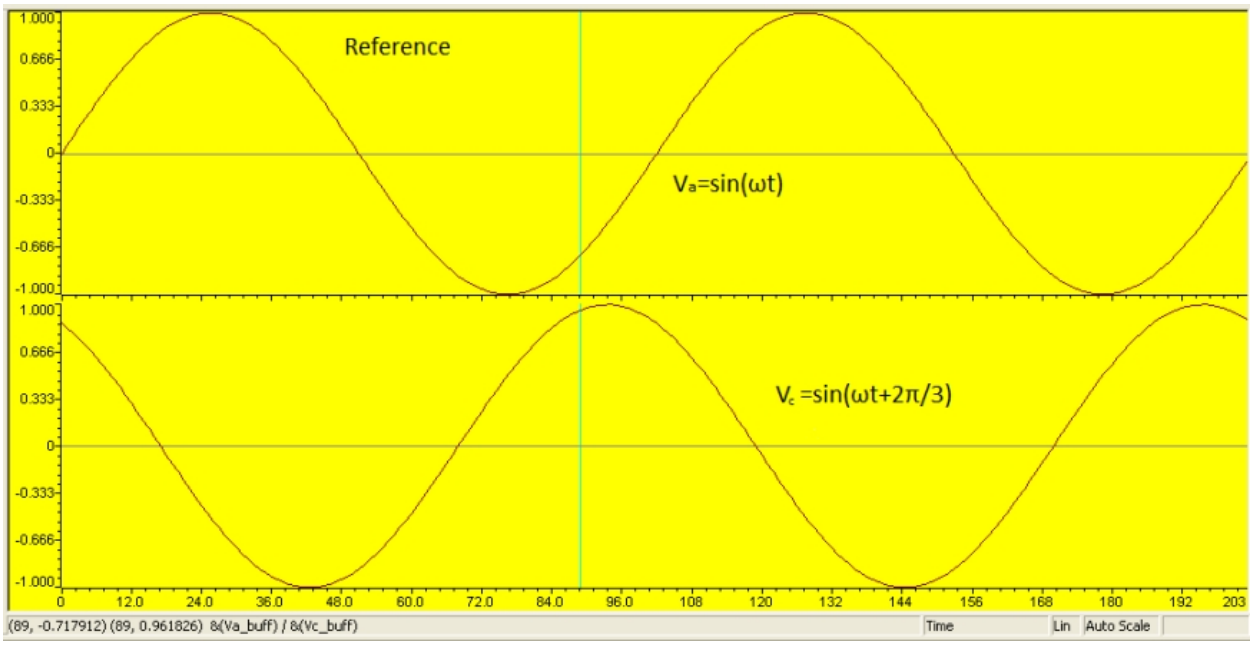
- `EPwm1Regs.CMPA.half.CMPA= _IQsat(_IQmpy(svgen_dq1.Ta,EPwm1Regs.TBPRD),EPwm1Regs.TBPRD,0);`

Μετατροπή του Ta σε CMPA για τη φάση 1 του PWM. Όλα τα μεγέθη είναι (εκτός από τους ακεραίους) είναι σε IQ ώστε οι πράξεις να είναι οικονομικές (σε χρόνο εκτέλεσης και σε εντολές assembly). Υπενθυμίζουμε ότι αυτό το κομμάτι εκτελείται σε κάθε κύκλο του φέροντος (real time), οπότε ενδιαφερόμαστε να είναι όσο το δυνατόν ταχύτερο.

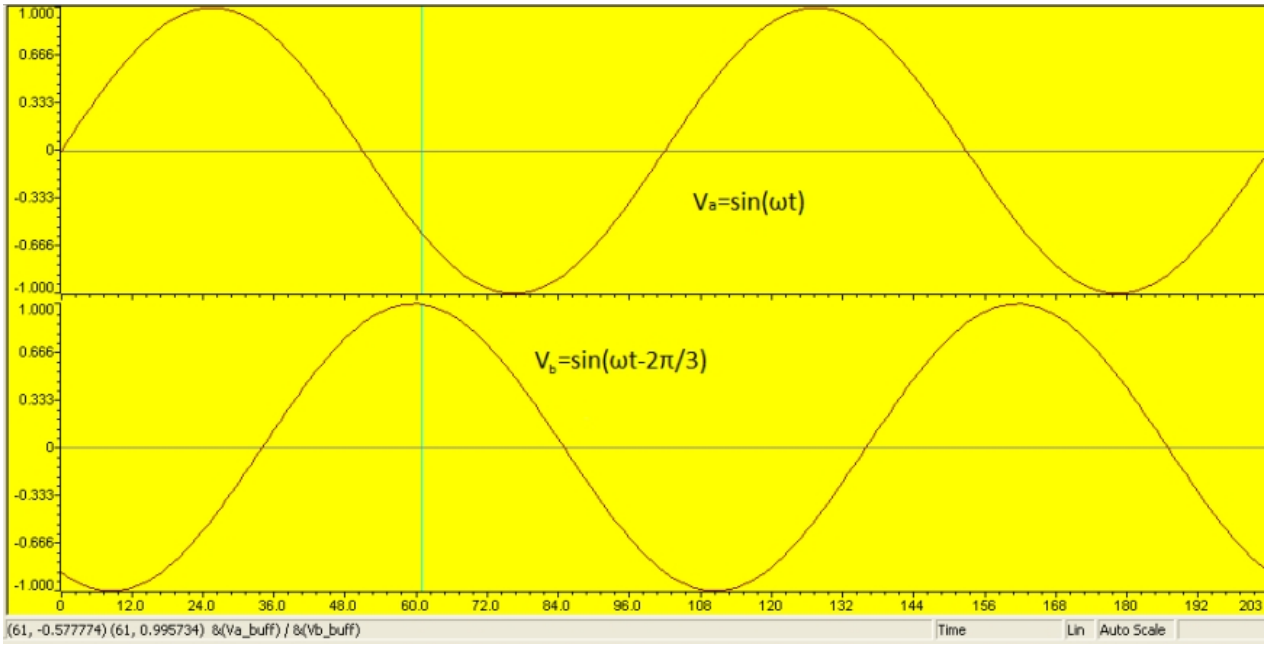
Ο κώδικας των αρχείων SVPWM.c και Clarke.c περιέχονται με πλήρη σχολιασμό στο παράρτημα.

12.4 Πειραματικά Αποτελέσματα

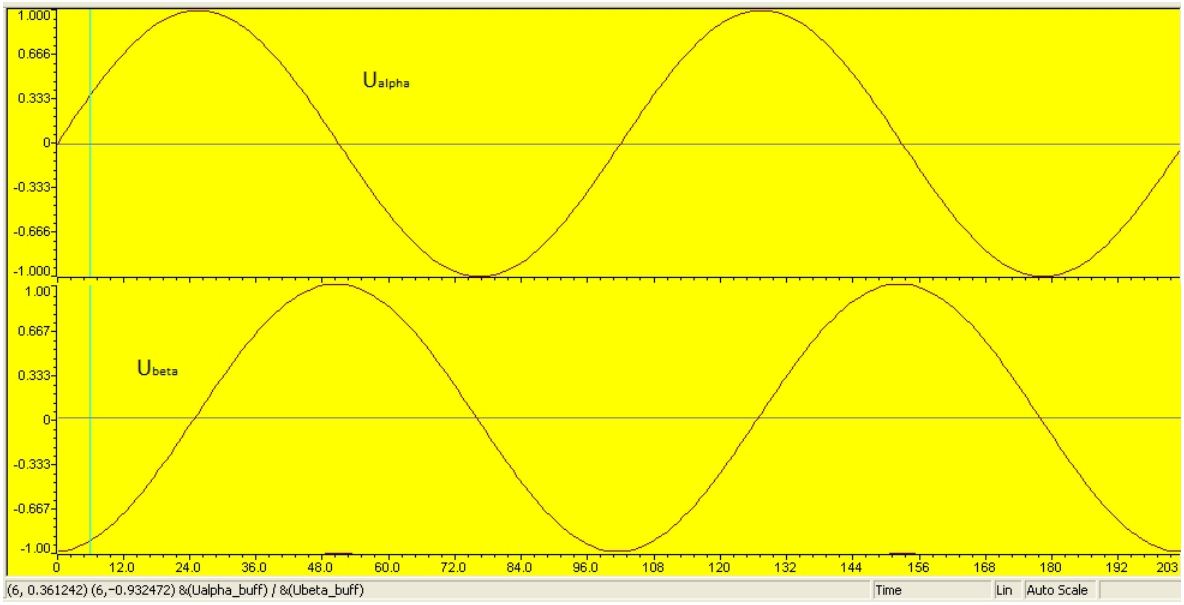
Παρακάτω ακολουθούν στιγμιότυπα από τον παλμογράφο και από τη λειτουργία graph του code composer studio που επιβεβαιώνουν την ορθή λειτουργία του προγράμματος. Σε όλες τις περιπτώσεις η συχνότητα της αναφοράς είναι 50Hz.



Σχήμα 12.3: Το τριφασικό σύστημα αναφοράς (δύο κύκλοι) στο CCS (Code Composer Studio)

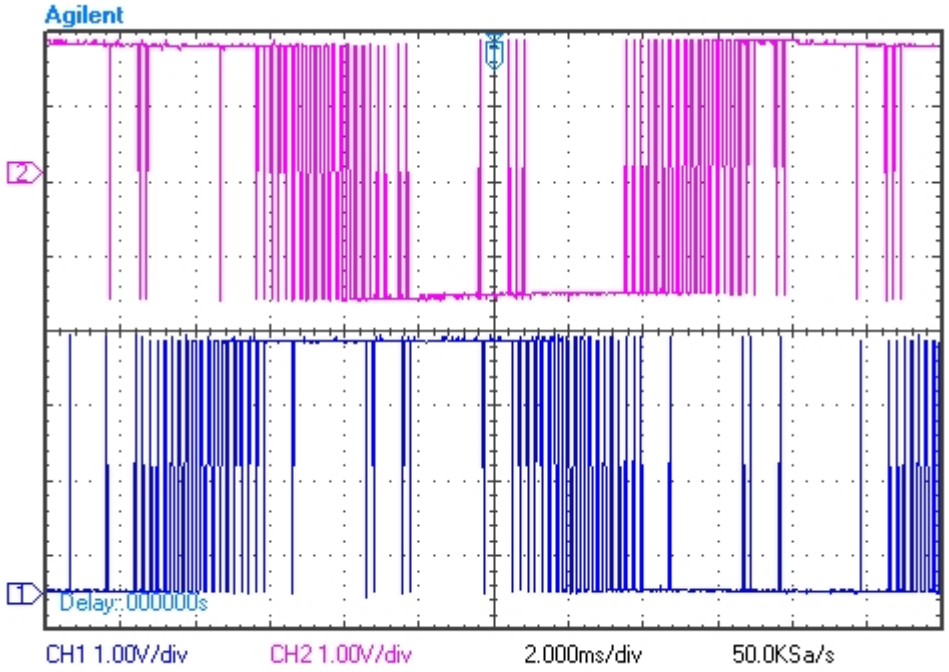


Σχήμα 12.4: Το τριφασικό σύστημα αναφοράς (δύο κύκλοι) στο CCS

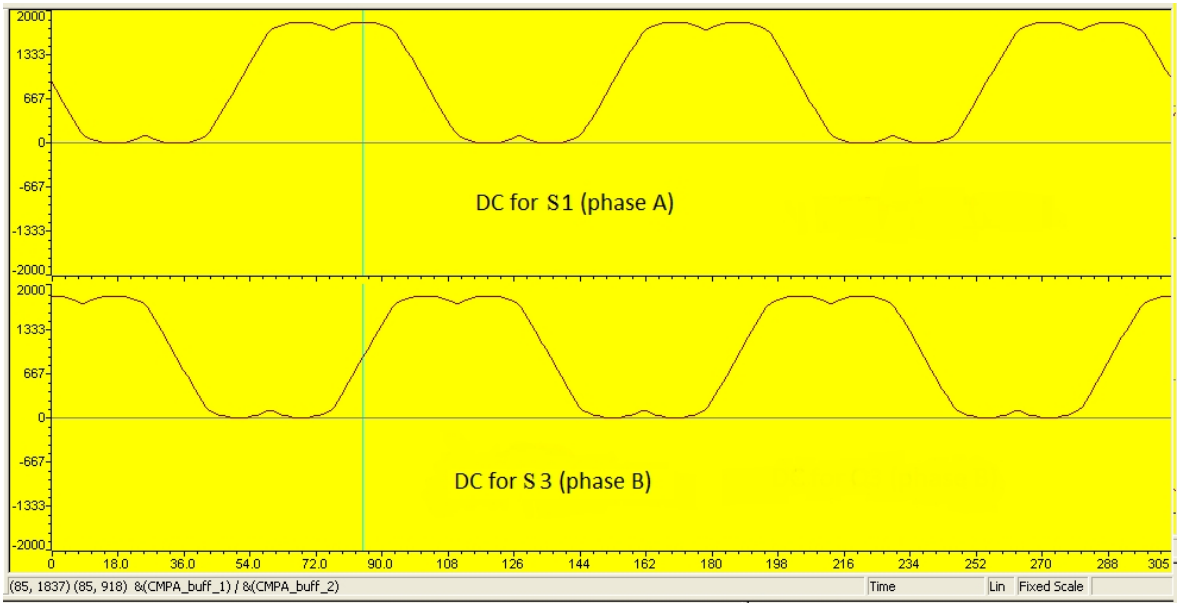


Σχήμα 12.5: Ο μετασχηματισμός Clarke (U_α,U_β) στο CCS

Συχνότητα φέροντος 5,1KHz

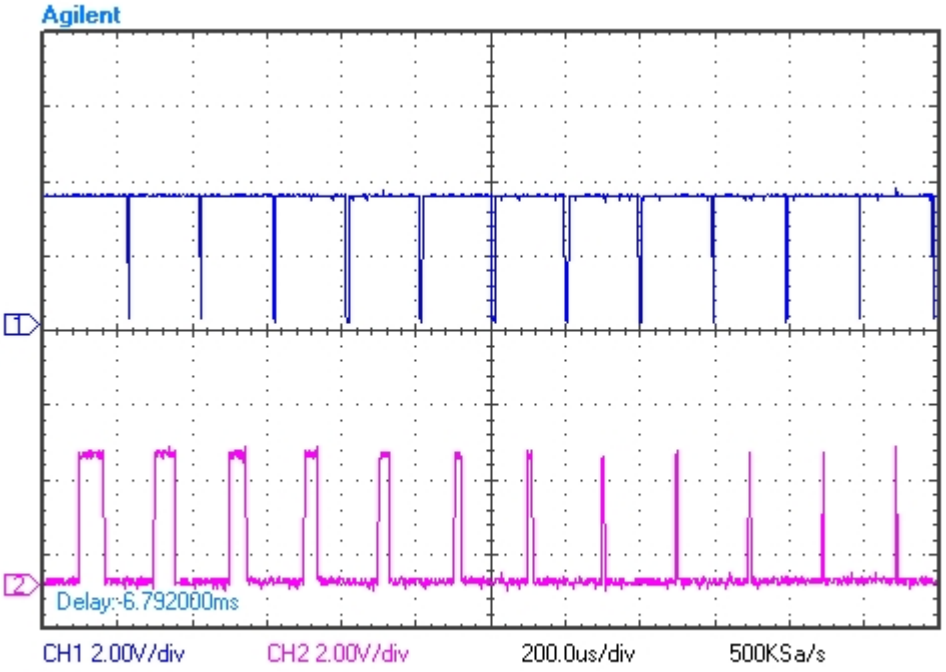


Σχήμα 12.6: Παλμοί οδήγησης διακόπτη S1 (ροζ-φάση A) και S3 (μπλε-φάση B)

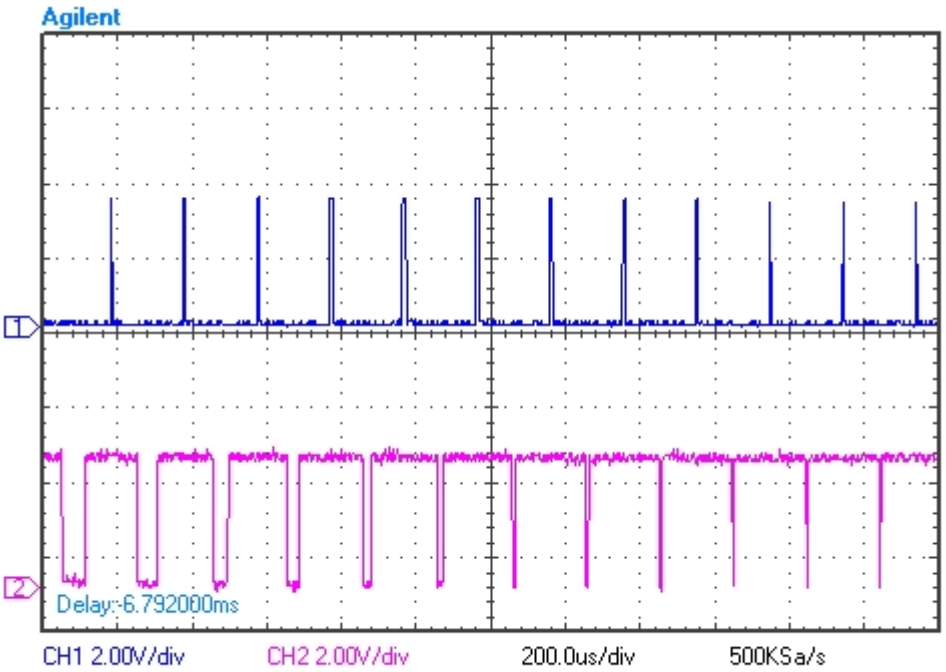


Σχήμα 12.7: Συντελεστές χρησιμοποίησης των διακοπτών S1 και S3 (τρεις κύκλοι) στο CCS

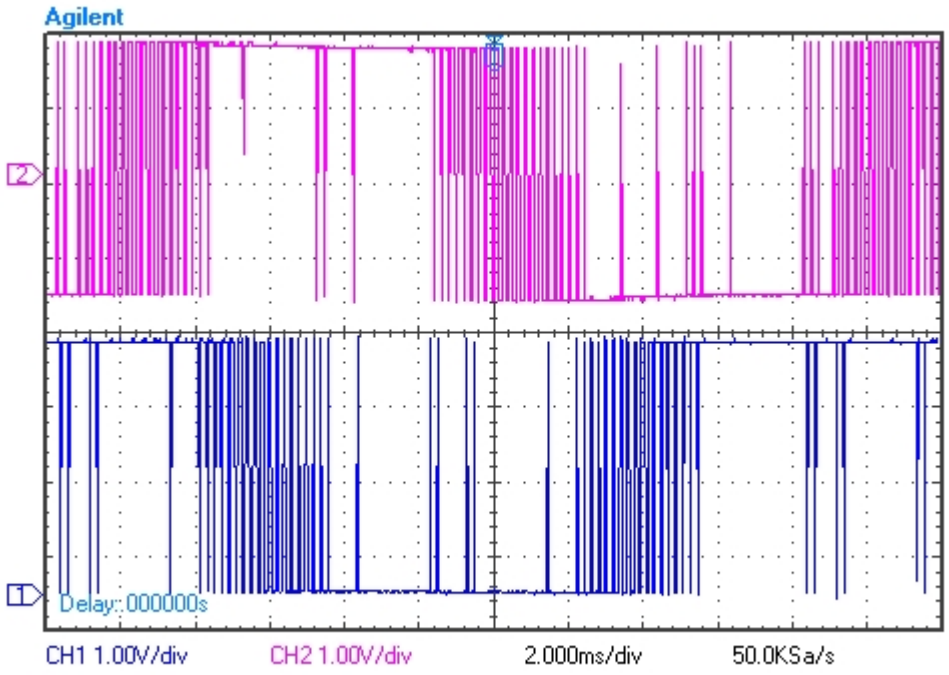
Στο Σχήμα 12.6 δεν είναι ξεκάθαρη η μορφή των παλμών όταν είναι πολύ κοντά στο 1 ή αντίστοιχα στο 0. Αυτό οφείλεται στο ότι ο παλμογράφος για τη δεδομένη ανάλυση δεν μπορεί να αναπαραστήσει με ακρίβεια τόσο μικρά χρονικά διαστήματα. Για αυτό το λόγο παραθέτουμε τα σχήματα 12.8 και 12.9 που δείχνουν σε μεγέθυνση τις περιοχές αυτές.



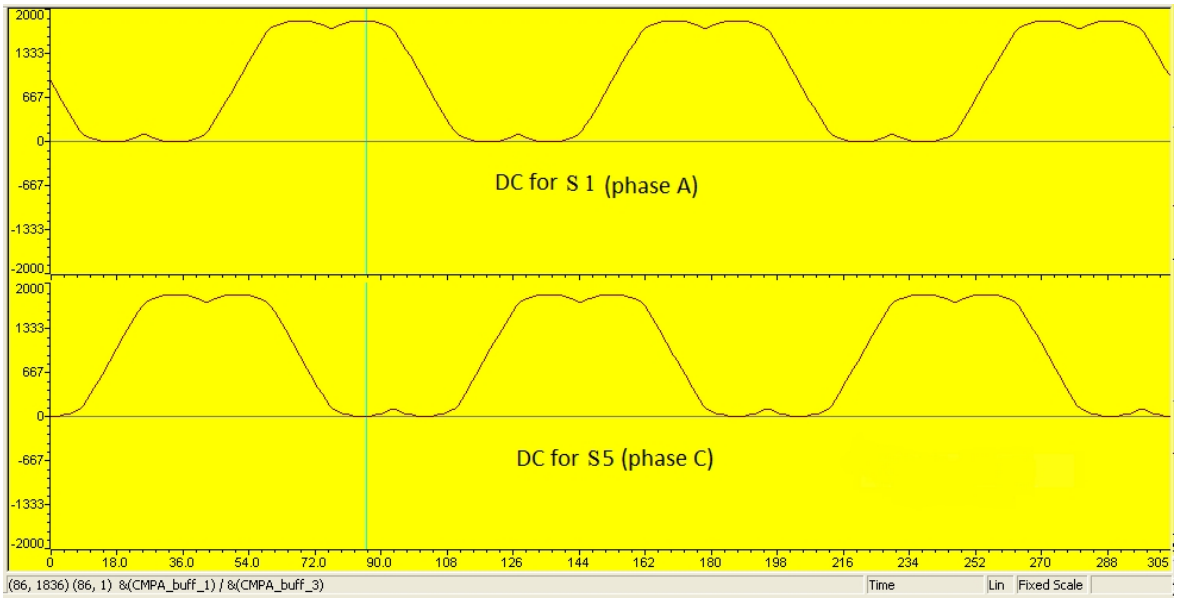
Σχήμα 12.8: Παλμοί οδήγησης διακόπτη S1 (μπλε-φάση A) εστίαση στο $+V_{DC}$ στον παλμογράφο



Σχήμα 12.9: Παλμοί οδήγησης διακόπτη S1 (μπλε-φάση A) εστίαση εστίαση στα μηδενικά στον παλμογράφο



Σχήμα 12.10: Παλμοί οδήγησης διακόπτη S1 (ροζ-φάση A) και S5 (μπλε-φάση C)

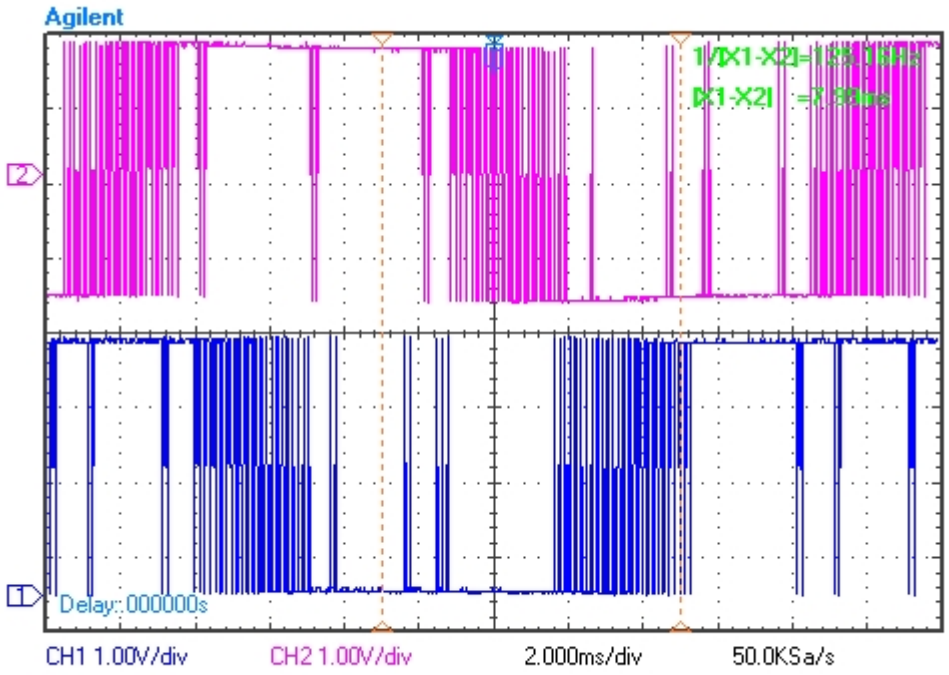


Σχήμα 12.11: Συντελεστές χρησιμοποίησης των διακοπτών S1 και S5 (τρεις κύκλοι) στο CCS

Παρατηρούμε ότι οι συντελεστές χρησιμοποίησης για την κάθε φάση A,B,C έχει ίδια μορφή με εκείνη που προέκυψε από την αριθμητική μελέτη της μεθόδου SVPWM στο κεφάλαιο 3. Αυτό αποτελεί τεκμήριο ορθής λειτουργίας του κώδικα.

Συχνότητα φέροντος 7KHz

Η παρουσίαση αυτής της συχνότητας γίνεται μόνο για να αποδειχθεί ότι μπορούμε να αναπαραστήσουμε συχνότητες στις οποίες το **sine_sampling≠3κ**. Αυτό δεν έχει καμία επίδραση στην περιοδικότητα της παλμοσειράς, αλλά υπάρχει στρογγυλοποίηση στη διαφορά φάσης της αναφοράς (η φάση Α με τη φάση Β δεν έχουν ακριβώς 120^o)



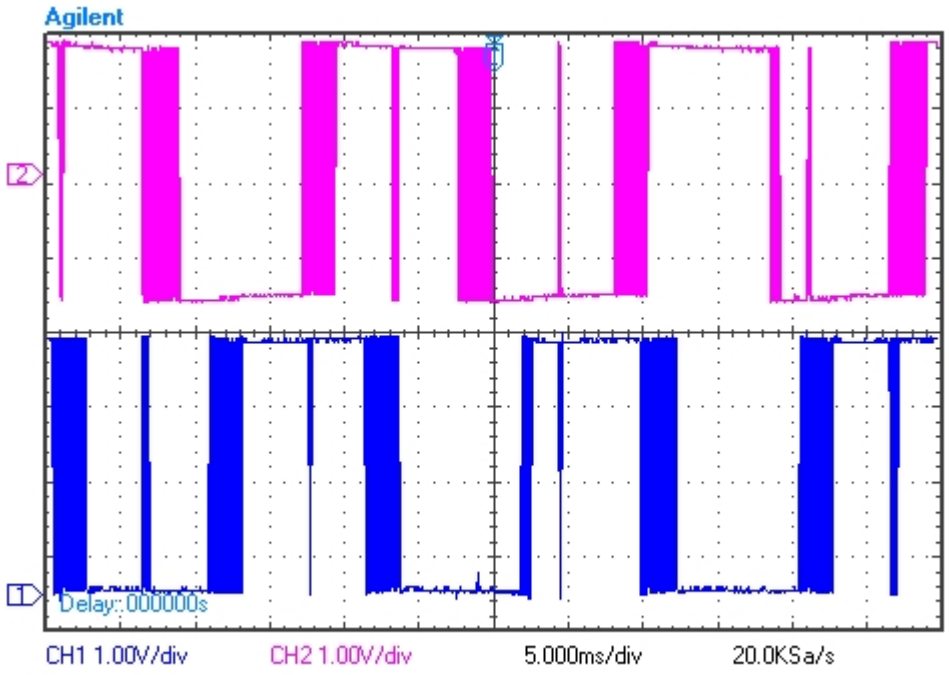
Σχήμα 12.12: Παλμοί οδήγησης διακόπτη S1 (ροζ-φάση Α) και S3 (μπλε-φάση Β) στα 7KHz στον παλμογράφο

freq_carrier	7000	long	dec
sine_sampling	140	int	dec
phase_shift_120	46	long	dec
phase_shift_240	93	long	dec

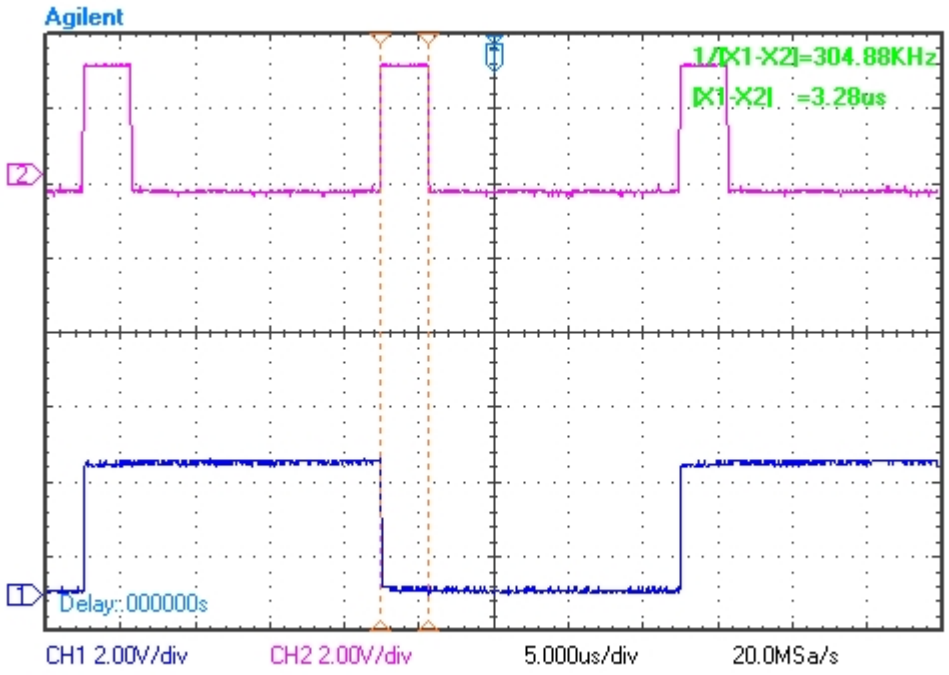
$$140/3 = 46,\overline{6} \rightarrow 46$$
$$2 \cdot 140/3 = 93,\overline{3} \rightarrow 93$$

Σχήμα 12.13: Στοιχεία της υλοποίησης των 7KHz

Συχνότητα φέροντος 50,1 KHz



Σχήμα 12.15: Παλμοί οδήγησης διακόπτη S1 (ροζ-φάση A) και S3 (μπλε-φάση B) στα 50,1KHz στον παλμογράφο



Σχήμα 12.17: Σύγκριση χρόνου εκτέλεσης (ροζ) με διαθέσιμο διάστημα υπολογισμού (μπλε) στον παλμογράφο

Για το σχήμα 12.17 η πάνω γραφική παράσταση είναι ο χρόνος εκτέλεσης της ΡΕΔ

για την εύρεση των CMPA ενώ η κάτω παλμοσειρά αναπαριστά το φορέα (κάθε ακμή αντιστοιχεί σε ένα μηδενισμό του τριγώνου). Παρατηρούμε ότι το διαθέσιμο διάστημα είναι αρκετά μεγάλο, το οποίο μας επιτρέπει να ανεβούμε και ακόμα και στα 150KHz στη συχνότητα του φέροντος.

12.5 Κώδικας με σχόλια

```
//  GLOBAL IQ is 28 for this programm
//      (c) Stelios Orfanos
//
//###########################################################################
// TITLE:   Space Vector Generator, open loop with fixed 3 phase
sinusoidal vectors
//          ePWM pin out 0,4,8
//###########################################################################
//###########################################################################
#include "DSP2833x_Device.h"
#include "IQmathLib.h"
#include "clarke.h"
#include "svgen_dq.h"
#define PI 3.14159265358979323846

_iq28
in1,wave[1050],Va_buff[512],Vb_buff[1050],Vc_buff[512],Ualpha_buff[10
50],
Ubeta_buff[512],Va,Vb,Vc;
// external function prototypes
extern void InitSysCtrl(void);
extern void InitPieCtrl(void);
extern void InitPieVectTable(void);
extern void InitCpuTimers(void);
extern void ConfigCpuTimer(struct CPUTIMER_VARS *, float, float);
// Prototype statements for functions found within this file.
char scan_for_PWM_solutions(long int,int,long int);
void find_parameters_PWM(long int);
void Gpio_select(void);
void Setup_ePWM1(void);
interrupt void ePWM1A_compare_isr(void);
interrupt void cpu_timer0_isr(void);

int clk_div=0,hsp_clk_div=0,tb_prd=0,CMPA=0,
CMPA_loc=0,index=0,counter=0, counter2=0;
int shot=0,shot_CMPA=0;
long int ratio_loc,int1,int2, int3, int4=0;
int cdiv[8]={1,2,4,8,16,32,64,128};
int hspdiv[8]={1,2,4,6,8,10,12,14};
int sine_sampling,i;
Uint16 CMPA_buff_1[1050],CMPA_buff_2[512],CMPA_buff_3[512];
long int freq_carrier=5100;
int freq_ref=50;
float float_VR1=0.0,period_carrier,
dc=0,pos_width=0,neg_width_half=0;
float temp_float;

CLARKE clarke1 = CLARKE_DEFAULTS;
SVGENDQ svgen_dq1 = SVGENDQ_DEFAULTS;



//###########################################################################
//                         main code
//###########################################################################
void main(void)
{
    InitSysCtrl();  // Basic Core Init from DSP2833x_SysCtrl.c
```

```
    EALLOW;
    SysCtrlRegs.WDCR= 0x00AF;   // Re-enable the watchdog
    EDIS;     // 0x00AF  to NOT disable the Watchdog, Prescaler = 64

    DINT;               // Disable all interrupts

    Gpio_select();      // GPIO9, GPIO11, GPIO34 and GPIO49 as output
                        // to 4 LEDs at Peripheral Explorer Board

    Setup_ePWM1();      // init of ePWM1A
    EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD / 2;
    EPwm3Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD / 2;
    EPwm5Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD / 2;

    InitPieCtrl();      // basic setup of PIE table; from
                        //DSP2833x_PieCtrl.c

    InitPieVectTable(); // default ISR's in PIE

    EALLOW;
    PieVectTable.EPWM1_INT = &ePWM1A_compare_isr;
    EDIS;


    // Enable EPWM1A INT in the PIE: Group 3 interrupt 1
    PieCtrlRegs.PIEIER3.bit.INTx1 = 1;
    PieCtrlRegs.PIEIER1.bit.INTx6 = 1;
    IER |=5;            // enable INT3 for ePWM1

    EINT;
    ERTM;



    sine_sampling=freq_carrier/freq_ref;
    period_carrier=1000000.0/(freq_carrier*1.0);
    int1=sine_sampling*2/3;
    int2=sine_sampling/3;
    for (i=0;i<sine_sampling;i++)
        {
        temp_float=i/(sine_sampling*1.0);

in1=_IQ28mpy(_IQ28mpy(_IQ28(2.0),_IQ28(PI)),_IQ28(temp_float));
        wave[i]=_IQ28sin(in1);
        }


    while(1)
    {

        EALLOW;
        SysCtrlRegs.WDKEY = 0x55;   // service WD #1
            //SysCtrlRegs.WDKEY = 0xAA;
        EDIS;
    }
}

void Gpio_select(void)
{
    EALLOW;
    GpioCtrlRegs.GPAMUX1.all = 0;       // GPIO15 ... GPIO0 = General
```

```
                                        //Puropse I/O
    GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1; // ePWM1A active
    GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 1; // ePWM3A active
    GpioCtrlRegs.GPAMUX1.bit.GPIO8 = 1; // ePWM5A active
    GpioCtrlRegs.GPAMUX2.all = 0;       // GPIO31 ... GPIO16 =
                                        //General Purpose I/O
    GpioCtrlRegs.GPBMUX1.all = 0;       // GPIO47 ... GPIO32 =
                                        //General Purpose I/O
    GpioCtrlRegs.GPBMUX2.all = 0;       // GPIO63 ... GPIO48 =
                                        //General Purpose I/O
    GpioCtrlRegs.GPCMUX1.all = 0;       // GPIO79 ... GPIO64 =
                                        //General Purpose I/O
    GpioCtrlRegs.GPCMUX2.all = 0;       // GPIO87 ... GPIO80 =
                                        //General Purpose I/O

    GpioCtrlRegs.GPADIR.all = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO9 = 1;
    GpioCtrlRegs.GPADIR.bit.GPIO11 = 1;

    GpioCtrlRegs.GPBDIR.all = 0;        // GPIO63-32 as inputs
    GpioCtrlRegs.GPBDIR.bit.GPIO59 = 1;
    GpioCtrlRegs.GPBDIR.bit.GPIO49 = 1;
    GpioCtrlRegs.GPCDIR.all = 0;        // GPIO87-64 as inputs
    EDIS;
}

char scan_for_PWM_solutions(long int high,int low,long int fpwm)
{
    int i,j,found=0, pre_prd;

    for (i=7;i>=0;i--)//(i=0;i<=7;i++)
        {
        for (j=7;j>=0;j--)//(j=0;j<=7;j++)
            {
            pre_prd=150000000/2/fpwm/cdiv[i]/hspdiv[j];
            if ((pre_prd<=high)&&(pre_prd>=low))
                {found=1; break;}
            };
        if (found==1) break;
        };

    if (found==1)    {
                     clk_div=i;
                     hsp_clk_div=j;
                     tb_prd=150000000/2/fpwm/cdiv[i]/hspdiv[j];
                     return 1;
                     }
        else return 0;

}


void find_parameters_PWM(long int fpwm)
{
    while (1)
    {
        if (scan_for_PWM_solutions(63000,1000,fpwm)==1) break;
        if (scan_for_PWM_solutions(63000,100,fpwm)==1) break;
        if (scan_for_PWM_solutions(63000,10,fpwm)==1) break;
        if (scan_for_PWM_solutions(63000,3,fpwm)==1) break;
        break;
```

```
    }
}


void Setup_ePWM1(void)
{
    find_parameters_PWM(freq_carrier);
    EPwm1Regs.TBCTL.bit.CLKDIV =  clk_div;
        EPwm3Regs.TBCTL.bit.CLKDIV =  clk_div;
        EPwm5Regs.TBCTL.bit.CLKDIV =  clk_div;
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = hsp_clk_div;
        EPwm3Regs.TBCTL.bit.HSPCLKDIV = hsp_clk_div;
        EPwm5Regs.TBCTL.bit.HSPCLKDIV = hsp_clk_div;
    EPwm1Regs.TBCTL.bit.CTRMODE = 2;    // up - down mode
        EPwm3Regs.TBCTL.bit.CTRMODE = 2;    // up - down mode
        EPwm5Regs.TBCTL.bit.CTRMODE = 2;    // up - down mode
    EPwm1Regs.AQCTLA.all = 0x0060;      // set ePWM1A on CMPA up
        EPwm3Regs.AQCTLA.all = 0x0060;  // clear ePWM1A on CMPA down
        EPwm5Regs.AQCTLA.all = 0x0060;
    EPwm1Regs.TBPRD =tb_prd;
        EPwm3Regs.TBPRD =tb_prd;
        EPwm5Regs.TBPRD =tb_prd;

    EPwm1Regs.CMPCTL.all=0x0000; //CMPCTL=0x0000 => shadow mode ON,
                                 //load on PRD=0
    EPwm3Regs.CMPCTL.all=0x0000;
    EPwm5Regs.CMPCTL.all=0x0000;

    EPwm1Regs.ETSEL.all = 0;
    EPwm1Regs.ETSEL.bit.INTEN = 1;      // interrupt enable for ePWM1
    EPwm1Regs.ETSEL.bit.INTSEL = 1;     // interrupt on PRD=0 TESTING
    EPwm1Regs.ETPS.bit.INTPRD = 1;      // interrupt on first event

}


interrupt void ePWM1A_compare_isr(void)
{   //Worst case: 4400 ns
     GpioDataRegs.GPASET.bit.GPIO11 = 1;

    // Service watchdog every interrupt

    EALLOW;
    SysCtrlRegs.WDKEY = 0xAA;       // Service watchdog #2
    EDIS;
    if (index==shot) GpioDataRegs.GPASET.bit.GPIO9 = 1;
    if (index==shot+1) GpioDataRegs.GPACLEAR.bit.GPIO9 = 1;

    shot_CMPA=CMPA_buff_1[shot-1];    //Calculation of
                                      //expected pulse widths
    dc=shot_CMPA*1.0/(tb_prd*1.0);
    neg_width_half=period_carrier*dc/2;
    pos_width=period_carrier-neg_width_half*2;

    Va=wave[index];
    Vb=wave[(index+sine_sampling*2/3)%sine_sampling];
    //512/3~171. We take
    //mod cause it would be not beneficial to check index overflow
    //for every Vx.
    //RE EVALUATION: if we allow index to grow beyond measure
      //it creates some overflow problems so we make it zero every
```

```
      //sine sampling
    Vc=wave[(index+sine_sampling/3)%sine_sampling];

    clarke1.As=Va;
    clarke1.Bs=Vb;
    clarke1.calc(&clarke1);
    svgen_dq1.Ualpha=clarke1.Alpha;
    svgen_dq1.Ubeta=clarke1.Beta;
    //defined in cpu timer. See notes of 11/10
    svgen_dq1.calc(&svgen_dq1);
    EPwm1Regs.CMPA.half.CMPA =
_IQsat(_IQmpy(svgen_dq1.Ta,EPwm1Regs.TBPRD),EPwm1Regs.TBPRD,0);
    EPwm3Regs.CMPA.half.CMPA =
_IQsat(_IQmpy(svgen_dq1.Tb,EPwm3Regs.TBPRD),EPwm3Regs.TBPRD,0);
    EPwm5Regs.CMPA.half.CMPA =
_IQsat(_IQmpy(svgen_dq1.Tc,EPwm5Regs.TBPRD),EPwm5Regs.TBPRD,0);


    CMPA_buff_1[index]  =EPwm1Regs.CMPA.half.CMPA;
    CMPA_buff_2[index]  =EPwm3Regs.CMPA.half.CMPA;
    CMPA_buff_3[index]  =EPwm5Regs.CMPA.half.CMPA;
    Va_buff[index]      =Va;
    Vb_buff[index]      =Vb;
    Vc_buff[index]      =Vc;
    Ualpha_buff[index]  =clarke1.Alpha;
    Ubeta_buff[index]   =clarke1.Beta;

    index ++;
    if (index ==sine_sampling) index = 0;
    EPwm1Regs.ETCLR.bit.INT = 1;        // Clear ePWM1 Interrupt flag
      // Acknowledge this interrupt to receive more interrupts from
      //group 3
    PieCtrlRegs.PIEACK.all = 4;
    GpioDataRegs.GPACLEAR.bit.GPIO11 = 1;
}

//===========================================================================
// End of SourceCode.
//===========================================================================
```

- **Space Vector generator:**

```
#include "IQmathLib.h"          // Include header for IQmath library
// Don't forget to set a proper GLOBAL_Q in "IQmathLib.h" file
#include "dmctype.h"
#include "svgen_dq.h"

void svgendq_calc(SVGENDQ *v)
{

    _iq Va,Vb,Vc,t1,t2;
    Uint32 Sector = 0;  // Sector is treated as Q0 - independently
                        //with global Q

// Inverse clarke transformation
    Va = v->Ubeta;
    Vb = _IQmpy(_IQ(-0.5),v->Ubeta) + _IQmpy(_IQ(0.8660254),v-
>Ualpha);  // 0.8660254 = sqrt(3)/2
    Vc = _IQmpy(_IQ(-0.5),v->Ubeta) - _IQmpy(_IQ(0.8660254),v-
>Ualpha);  // 0.8660254 = sqrt(3)/2

// 60 degree Sector determination
    if (Va>_IQ(0))
       Sector = 1;
    if (Vb>_IQ(0))
       Sector = Sector + 2;
    if (Vc>_IQ(0))
       Sector = Sector + 4;

// X,Y,Z (Va,Vb,Vc) calculations
    Va = v->Ubeta;
// X = Va
    Vb = _IQmpy(_IQ(0.5),v->Ubeta) + _IQmpy(_IQ(0.8660254),v-
>Ualpha);   // Y = Vb
    Vc = _IQmpy(_IQ(0.5),v->Ubeta) - _IQmpy(_IQ(0.8660254),v-
>Ualpha);   // Z = Vc

    if (Sector==0)  // Sector 0: this is special case for
                    //(Ualpha,Ubeta) = (0,0)
    {
       v->Ta = _IQ(0.5);
       v->Tb = _IQ(0.5);
       v->Tc = _IQ(0.5);
    }
    if (Sector==1)  // Sector 1: t1=Z and t2=Y (abc ---> Tb,Ta,Tc)
    {
       t1 = Vc;
       t2 = Vb;
       v->Tb = _IQmpy(_IQ(0.5),(_IQ(1)-t1-t2));      // tbon = (1-t1-
                                                     //t2)/2
       v->Ta = v->Tb+t1;                             // taon =
                                                     //tbon+t1
       v->Tc = v->Ta+t2;                             // tcon =
                                                     //taon+t2
    }
    else if (Sector==2)  // Sector 2: t1=Y and t2=-X (abc --->
                         //Ta,Tc,Tb)
    {
       t1 = Vb;
       t2 = -Va;
       v->Ta = _IQmpy(_IQ(0.5),(_IQ(1)-t1-t2));      // taon = (1-t1-
                                                     //t2)/2
```

```
        v->Tc = v->Ta+t1;                                   // tcon =
                                                            //taon+t1
        v->Tb = v->Tc+t2;                                   // tbon =
                                                            //tcon+t2
    }
    else if (Sector==3)  // Sector 3: t1=-Z and t2=X (abc --->
                         //Ta,Tb,Tc)
    {
        t1 = -Vc;
        t2 = Va;
        v->Ta = _IQmpy(_IQ(0.5),(_IQ(1)-t1-t2));        // taon = (1-t1-
                                                            //t2)/2
        v->Tb = v->Ta+t1;                                   // tbon =
                                                            //taon+t1
        v->Tc = v->Tb+t2;                                   // tcon =
                                                            //tbon+t2
    }
    else if (Sector==4)  // Sector 4: t1=-X and t2=Z (abc --->
                         //Tc,Tb,Ta)
    {
        t1 = -Va;
        t2 = Vc;
        v->Tc = _IQmpy(_IQ(0.5),(_IQ(1)-t1-t2));        // tcon = (1-t1-
                                                            //t2)/2
        v->Tb = v->Tc+t1;                                   // tbon =
                                                            //tcon+t1
        v->Ta = v->Tb+t2;                                   // taon =
                                                            //tbon+t2
    }
    else if (Sector==5)  // Sector 5: t1=X and t2=-Y (abc --->
                         //Tb,Tc,Ta)
    {
        t1 = Va;
        t2 = -Vb;
        v->Tb = _IQmpy(_IQ(0.5),(_IQ(1)-t1-t2));        // tbon = (1-t1-
                                                            //t2)/2
        v->Tc = v->Tb+t1;                                   // tcon =
                                                            //tbon+t1
        v->Ta = v->Tc+t2;                                   // taon =
                                                            //tcon+t2
    }
    else if (Sector==6)  // Sector 6: t1=-Y and t2=-Z (abc --->
                         //Tc,Ta,Tb)
    {
        t1 = -Vb;
        t2 = -Vc;
        v->Tc = _IQmpy(_IQ(0.5),(_IQ(1)-t1-t2));        // tcon = (1-t1-
                                                            //t2)/2
        v->Ta = v->Tc+t1;                                   // taon =
                                                            //tcon+t1
        v->Tb = v->Ta+t2;                                   // tbon =
                                                            //taon+t2
    }

// Convert the unsigned GLOBAL_Q format (ranged (0,1)) -> signed
//GLOBAL_Q format (ranged (-1,1))
    v->Ta = _IQmpy(_IQ(2.0),(v->Ta-_IQ(0.5)));
    v->Tb = _IQmpy(_IQ(2.0),(v->Tb-_IQ(0.5)));
    v->Tc = _IQmpy(_IQ(2.0),(v->Tc-_IQ(0.5)));

}
```

- **Clarke transformation:**

```
#include "IQmathLib.h"          // Include header for IQmath library
// Don't forget to set a proper GLOBAL_Q in "IQmathLib.h" file
#include "dmctype.h"
#include "clarke.h"

void clarke_calc(CLARKE *v)
{

   v->Alpha = v->As;

   v->Beta = _IQmpy((v->As + _IQmpy(_IQ(2),v->Bs)),
_IQ(0.57735026918963)); // 1/sqrt(3) = 0.57735026918963

}
```

ΚΕΦΑΛΑΙΟ 13

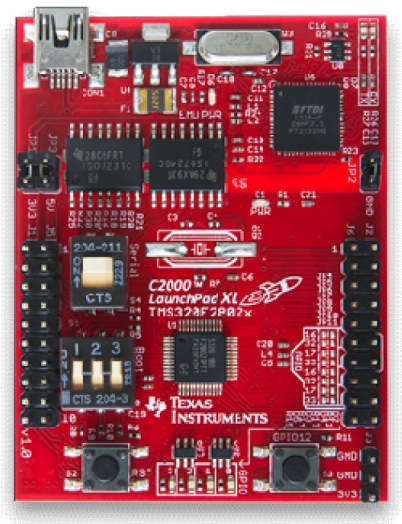
ΣΥΜΠΕΡΑΣΜΑΤΑ

Όπως ειπώθηκε και στην εισαγωγή, ο σκοπός της διπλωματικής είναι αμιγώς εκπαιδευτικός. Κατά την πορεία, ωστόσο προέκυψαν τα παρακάτω χρήσιμα συμπεράσματα:

1. Ο DSC TMS320F28335 είναι αρκετά ισχυρός για να υποστηρίξει εφαρμογές ηλεκτρονικών ισχύος, οι οποίες έχουν συχνότητα φορέα μέχρι και 5KHz. Ακόμα και στην περίπτωση του SVPWM παρατηρούμε ότι ο χρόνος εκτέλεσης των πράξεων που απαιτούνται ανά κύκλο, είναι σημαντικά μικρότερος από το συνολικό χρόνο που έχουμε διαθέσιμο ανά κύκλο του φορέα. Αυτό σημαίνει ότι σε πραγματικές εφαρμογές μπορούμε να χρησιμοποιήσουμε κάποιον επεξεργαστή μικρότερων δυνατοτήτων και κατά συνέπεια να μειώσουμε δραστικά το συνολικό κόστος του ενσωματωμένου συστήματος που θέλουμε να σχεδιάσουμε.

2. Ο προγραμματισμός σε χαμηλό επίπεδο (γλώσσα προγραμματισμού C) επιτρέπει στον μηχανικό την καλύτερη και πιο αξιόπιστη αντιμετώπιση τυχών προβλημάτων που προκύπτουν στον κώδικα. Όπως, φάνηκε και στην εφαρμογή του μονοφασικού SPWM, είναι αρκετά εύκολο να απομονωθεί και να εξεταστεί ξεχωριστά το κάθε σημείο του κώδικα, ώστε να εντοπιστεί και να διορθωθεί τυχών σφάλμα. Μειονέκτημα της μεθόδου αυτής είναι ο περισσότερος χρόνος που απαιτείται σε σύγκριση με τον προγραμματισμό μέσω έτοιμων συναρτήσεων (πχ. Matlab simulink)

3. Προς επέκταση του πρώτου συμπεράσματος, μπορούν να χρησιμοποιηθούν μικροελεγκτές μικρότερων δυνατοτήτων για εκπαιδευτικούς σκοπούς σε προπτυχιακό επίπεδο. Ένας τέτοιος μικροελεγκτής παρουσιάζεται στο σχήμα 13.1.

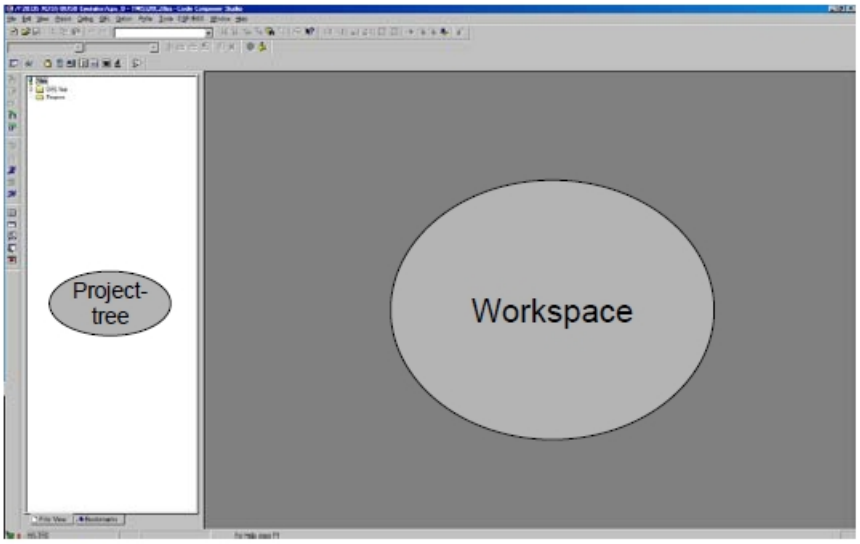


Σχήμα 13.1: Μικροελεγκτής C2000 LAUNCHXL-F28027 της Texas Instruments με ταχύτητα επεξεργαστή 60MHz, 12 -bit ADC και κανάλια PWM υψηλής ανάλυσης.

Αυτός ο μικροελεγκτής συνδυάζει όλα τα επιθυμητά στοιχεία (για διατάξεις Ηλεκτρονικών ισχύος) του F28335 με χαμηλό κόστος. Ο LAUNCHXL-F28027 είναι απλά ένα δείγμα από δεκάδες παρόμοιους ελεγκτές που είναι διαθέσιμοι στην αγορά.

<u>Παράρτημα Α:</u> Γνωριμία με το Code Composer Studio

Το πρόγραμμα που χρησιμοποιούν τα προϊόντα της Texas Instruments για την ανάπτυξη εφαρμογών είναι το Code Composer Studio (CCS). Αποτελεί έναν τυπικό μεταγλωττιστή C ο οποίος είναι εξειδικευμένος για τη λειτουργία ενσωματωμένων συστημάτων. Στις επόμενες σελίδες θα αναφερθούν βασικές διαδικασίες, ώστε ο αναγνώστης να μπορέσει να εκτελέσει κάποια βασικά προγράμματα.



<u>Σχήμα Α.1:</u> CCS interface

1) **Δημιουργία project**
   - Project→New: Δημιουργείται ένα project
   - file → New→ Source file: ανοίγουμε τον text editor του CCS και μπορούμε να γράψουμε κατευθείαν πηγαίο κώδικα C.
   - Filer → Save as: Αποθήκευση αρχείου
   - Project→Add files to project: Με αυτόν τον τρόπο μπορούμε να συνδέσουμε αποθηκευμένα αρχεία C με το παρόν project
   - Project→Compile file: Μεταγλώττιση του αρχείου.

   - Απαραίτητες διαδικασίες για να μπορούμε να δημιουργήσουμε απλά project είναι:

     Να προσθέσουμε τη C-runtime-library και το RAM_linker

     C:\CCStudio_v3.3\C2000\cgtools\lib\rts2800_ml.lib

     C:\tidcs\c28\DSP2833x\v131\DSP2833x_common\cmd\28335_RAM_lnk.cmd

- Και να θέσουμε το stack-size στο 0x400:

Project→Build Options→Linker→Stack size: 0x400

- Project→Build (compile, assemble and link)

2) Φόρτωση του project στον DSC

- Debug→Connect : Σύνδεση του DSP με τον υπολογιστή
- File→Load Program→Debug\myprogram.out : Φόρτωση του προγράμματος στον DSP.
- Τρέξιμο του προγράμματος μέχρι την αρχή της main()

Debug→Reset CPU	<CTRL+R>
Debug→Restart	<CTRL+SHIFT+F5>
Debug→Go main	<CTRL+M>

- Τρέξιμο προγράμματος σε πραγματικό χρόνο

Debug→Run	<F5>

- Σταμάτημα προγράμματος

Debug→Halt	<SHIFT+F5>

3) Παρατήρηση Μεταβλητών

Κατά τη διαδικασία προγραμματισμού είναι χρήσιμο να έχουμε πρόσβαση στις τιμές που λαμβάνουν οι διάφορες μεταβλητές κατά τη διάρκεια εκτέλεσης του προγράμματος. Το watch window μας δίνει τη δυνατότητα να παρατηρούμε τόσο τοπικές όσο και καθολικές μεταβλητές.

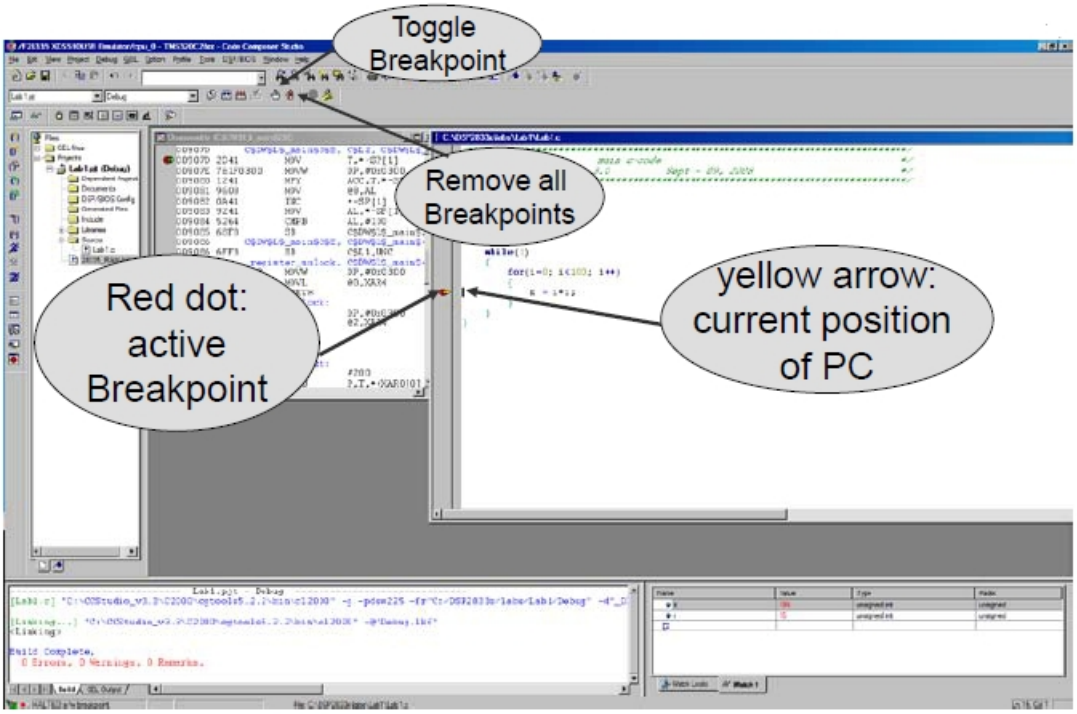
View→Watch Window για την ενεργοποίηση του παραθύρου

Ωστόσο, σε κανονική ροή προγράμματος δεν δυνατόν να παρατηρήσουμε την αλλαγή των μεταβλητών. Υπάρχουν 3 μέθοδοι με τις οποίες μπορούμε να αξιοποιήσουμε το watch window.

- Βηματική αποσφαλμάτωση (single step debugging)

Debug → Step Into (αντί για run) <F11>

- Δημιουργία Break point



Σχήμα Α.2: Δημιουργία Break Point

Ο κώδικας σταματά στην εντολή που έχουμε τοποθετήσει το breakpoint και έτσι ανανεώνονται οι τιμές στο Watch Window.

- Real Time Debugging

Ο πιο χρήσιμος τρόπος αποσφαλμάτωσης ενός προγράμματος είναι το Real time debugging. Χάρη σε εσωτερικούς JTAG καταχωρητές μπορούμε να παρατηρούμε τις μεταβλητές του προγράμματος χωρίς να επηρεάζεται η πορεία της εκτέλεσης του.

Για να χρησιμοποιήσουμε Real Time Debugging εκτελούμε:

Debug → Reset CPU
GEL → Watchdog → Disable Watchdog
Debug→ Realtime Emulation Control → Run realtime with Restart

Προσοχή: Όταν θέλουμε να σταματήσουμε το πρόγραμμα εκτελούμε αυστηρά

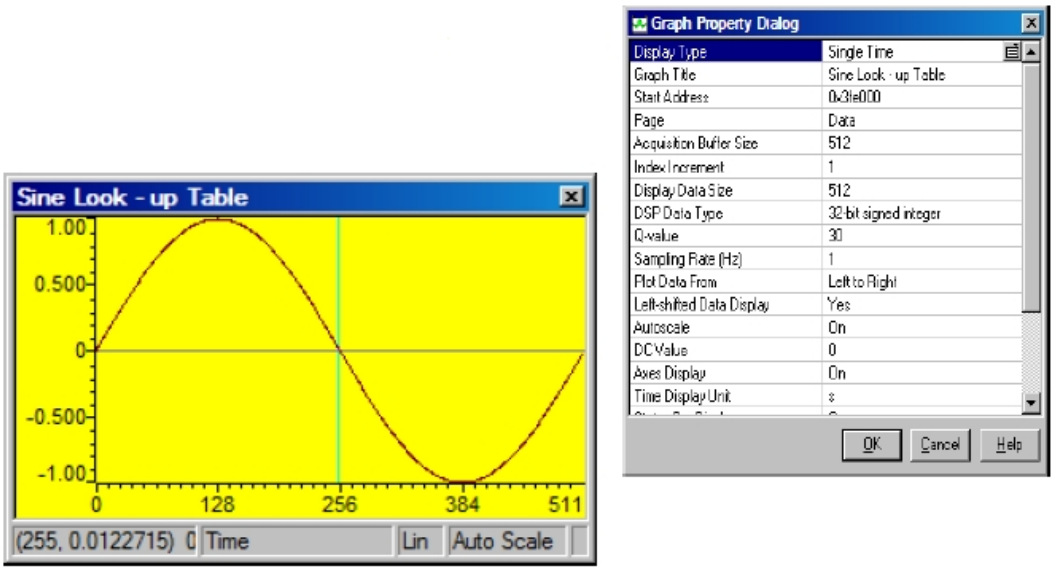
GEL→ Realtime Emulation Control →Full Halt

4) **Άλλες χρήσιμες λειτουργίες του CCS**
   - View→ Memory

Παρόμοιο με το Watch Window αλλά με θέσεις μνήμης. Με αυτόν τον τρόπο παρατηρούνται ευκολότερα διανύσματα και πίνακες.

   - View→ Graph → Time/Frequency

Με τη λειτουργία αυτή μπορούμε να αναπαραστήσουμε σε διάγραμμα δεδομένα που βρίσκονται αποθηκευμένα σε πίνακες. Στην παρακάτω εικόνα έχει σχεδιαστεί το γράφημα περιοχής μνήμης στην οποία είναι αποθηκευμένες 512 τιμές ημιτόνου.



<u>Σχήμα Α.3:</u> Γραφική αναπαράσταση περιοχής μνήμης του DSP

Η χρηστικότητα του CCS δεν εξαντλούνται στις βασικές αυτές λειτουργίες. Ωστόσο, είναι αρκετά για τα πλαίσια της παρούσας διπλωματικής.

Παράρτημα Β: Εισαγωγή στην αριθμητική IQ_math

Το IQ_math είναι μία μορφή πραγματικών αριθμών σταθερής υποδιαστολής την οποία επινόησε η Texas instruments. Παρότι ο F28335 υποστηρίζει πράξεις με αριθμούς κινητής υποδιαστολής η αρχιτεκτονική του υποστηρίζει και πράξεις IQ αριθμών. Ο λόγος είναι ότι σε εφαρμογές με μεγάλο υπολογιστικό φορτίο είναι γενικά ενδεδειγμένο να χρησιμοποιούνται αριθμοί σταθερής υποδιαστολής. Το μειονέκτημά τους είναι ότι ο χρόνος ανάπτυξης προγραμμάτων με IQ αριθμητική είναι αρκετά μεγαλύτερος από αυτόν της floating point λογική. Στην εργασία αξιοποιείται αρκετά η IQ αριθμητική για λόγους υπολογιστικής απόδοσης.

Data Type	Range		Resolution/Precision
	Min	Max	
_iq30	-2	1.999 999 999	0.000 000 001
_iq29	-4	3.999 999 998	0.000 000 002
_iq28	-8	7.999 999 996	0.000 000 004
_iq27	-16	15.999 999 993	0.000 000 007
_iq26	-32	31.999 999 985	0.000 000 015
_iq25	-64	63.999 999 970	0.000 000 030
_iq24	-128	127.999 999 940	0.000 000 060
_iq23	-256	255.999 999 981	0.000 000 119
_iq22	-512	511.999 999 762	0.000 000 238
_iq21	-1024	1023.999 999 523	0.000 000 477
_iq20	-2048	2047.999 999 046	0.000 000 954
_iq19	-4096	4095.999 998 093	0.000 001 907
_iq18	-8192	8191.999 996 185	0.000 003 815
_iq17	-16384	16383.999 992 371	0.000 007 629
_iq16	-32768	32767.999 984 741	0.000 015 259
_iq15	-65536	65535.999 969 482	0.000 030 518
_iq14	-131072	131071.999 938 965	0.000 061 035
_iq13	-262144	262143.999 877 930	0.000 122 070
_iq12	-524288	524287.999 755 859	0.000 244 141
_iq11	-1048576	1048575.999 511 719	0.000 488 281
_iq10	-2097152	2097151.999 023 437	0.000 976 563
_iq9	-4194304	4194303.998 046 875	0.001 953 125
_iq8	-8388608	8388607.996 093 750	0.003 906 250
_iq7	-16777216	16777215.992 187 500	0.007 812 500
_iq6	-33554432	33554431.984 375 000	0.015 625 000
_iq5	-67108864	67108863.968 750 000	0.031 250 000
_iq4	-134217728	134217727.937 500 000	0.062 500 000
_iq3	-268435456	268435455.875 000 000	0.125 000 000
_iq2	-536870912	536870911.750 000 000	0.250 000 000
_iq1	-1073741824	1 073741823.500 000 000	0.500 000 000

Σχήμα Β.1: Το εύρος τιμών και η ακρίβεια για κάθε τύπο IQn

Στις επόμενες εικόνες, παρατίθενται οι αριθμητικές συναρτήσεις και οι συναρτήσεις μετατροπής της βιβλιοθήκης IQ:

IQmath Library: Math & Trig Functions

Operation	Floating-Point	"IQmath" in C	"IQmath" in C++
type	float A, B;	_iq A, B;	iq A, B;
constant	A = 1.2345	A = _IQ(1.2345)	A = IQ(1.2345)
multiply	A * B	_IQmpy(A , B)	A * B
divide	A / B	_IQdiv (A , B)	A / B
add	A + B	A + B	A + B
substract	A - B	A - B	A – B
boolean	>, >=, <, <=, ==, |=, &&, ||	>, >=, <, <=, ==, |=, &&, ||	>, >=, <, <=, ==, |=, &&, ||
trig and power functions	sin(A),cos(A) sin(A*2pi),cos(A*2pi) asin(A),acos(A) atan(A),atan2(A,B) atan2(A,B)/2pi sqrt(A),1/sqrt(A) sqrt(A*A + B*B) exp(A)	_IQsin(A), _IQcos(A) _IQsinPU(A), _IQcosPU(A) _IQasin(A),_IQacos(A) _IQatan(A), _IQatan2(A,B) _IQatan2PU(A,B) _IQsqrt(A), _IQisqrt(A) _IQmag(A,B) _IQexp(A)	IQsin(A),IQcos(A) IQsinPU(A),IQcosPU(A) IQasin(A),IQacos(A) IQatan(A),IQatan2(A,B) IQatan2PU(A,B) IQsqrt(A),IQisqrt(A) IQmag(A,B) IQexp(A)
saturation	if(A > Pos) A = Pos if(A < Neg) A = Neg	_IQsat(A,Pos,Neg)	IQsat(A,Pos,Neg)

Σχήμα Β.2: Αριθμητικές συναρτήσεις IQmath

IQmath Library: Conversion Functions

Operation	Floating-Point	"IQmath" in C	"IQmath" in C++
iq to iqN	A	_IQtoIQN(A)	IQtoIQN(A)
iqN to iq	A	_IQNtoIQ(A)	IQNtoIQ(A)
integer(iq)	(long) A	_IQint(A)	IQint(A)
fraction(iq)	A – (long) A	_IQfrac(A)	IQfrac(A)
iq = iq*long	A * (float) B	_IQmpyl32(A,B)	IQmpyl32(A,B)
integer(iq*long)	(long) (A * (float) B)	_IQmpyl32int(A,B)	IQmpyl32int(A,B)
fraction(iq*long)	A - (long) (A * (float) B)	_IQmpyl32frac(A,B)	IQmpyl32frac(A,B)
qN to iq	A	_QNtoIQ(A)	QNtoIQ(A)
iq to qN	A	_IQtoQN(A)	IQtoQN(A)
string to iq	atof(char)	_atoIQ(char)	atoIQ(char)
IQ to float	A	_IQtoF(A)	IQtoF(A)
IQ to ASCII	sprintf(A,B,C)	_IQtoA(A,B,C)	IQtoA(A,B,C)

Σχήμα Β.3: Συναρτήσεις μετατροπής IQmath