ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΕΛΟΠΟΝΝΗΣΟΥ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

" ΚΑΤΑΣΚΕΥΗ ΤΕΤΡΑΕΛΙΚΟΠΤΕΡΟΥ ΚΑΙ ΑΝΑΠΤΥΞΗ ΟΛΟΚΛΗΡΩΜΕΝΟΥ ΣΥΣΤΗΜΑΤΟΣ ΕΛΕΓΧΟΥ ΠΤΗΣΗΣ ΜΕ ΠΛΑΤΦΟΡΜΑ ARDUINO "

"CONSTRUCTION OF A QUADCOPTER AND DEVELOPMENT OF INTEGRATED FLIGHT CONTROL SYSTEM BASED ON THE ARDUINO PLATFORM"

ΚΑΡΑΓΓΙΟΥΛΕΣ ΓΕΩΡΓΙΟΣ 6570
Αριθμός Πτυχιακής: 1774
ΕΠΙΒΛΕΠΩΝ:
ΠΕΤΡΟΣ ΒΛΑΧΟΠΟΥΛΟΣ
ΔΗΜΗΤΡΗΣ ΚΑΡΕΛΗΣ

ΠΑΤΡΑ 2020

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή Πάτρα, Ημερομηνία

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Πέτρος Βλαχόπουλος
2. Δημήτρης Καρέλλης
3. Μανώλης Γαλετάκης

Υπεύθυνη Δήλωση Φοιτητή

Βεβαιώνω ότι είμαι συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τη συγκεκριμένη εργασία. Η έγκριση της διπλωματικής εργασίας από το Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Πελοποννήσου δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα εκ μέρους του Τμήματος. Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Καραγγιουλέ Γεωργίου που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης ο συγγραφέας/δημιουργός εκχωρεί στο Πανεπιστήμιο Πελοποννήσου, μη αποκλειστική άδεια χρήσης του δικαιώματος αναπαραγωγής, προσαρμογής, δημόσιου δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσής τους διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος και για όλο το χρόνο διάρκειας των δικαιωμάτων πνευματικής ιδιοκτησίας. Η ανοικτή πρόσβαση στο πλήρες κείμενο για μελέτη και ανάγνωση δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, αποθήκευση, πώληση, εμπορική χρήση, μετάδοση, διανομή, έκδοση, εκτέλεση, «μεταφόρτωση» (downloading), «ανάρτηση» (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού. Ο συγγραφέας/δημιουργός διατηρεί το σύνολο των ηθικών και περιουσιακών του δικαιωμάτων.

Πίνακας Περιεχομένων

Σκοπός Εργασίας

Σκοπός της διπλωματικής αυτής είναι η κατασκευή τετραελικοπτέρου και έλεγχος της πτήσης του, χρησιμοποιώντας την πλατφόρμα ARDUINO.

Στόχος μας είναι η εξοικείωση με την πλατφόρμα Arduino και η αξιοποίηση όσων μάθαμε κατά τα διδακτικά μας εξάμηνα, από κολλήσεις πλακετών , χρήση μικροελεγκτών μέχρι έρευνα για το οποιοδήποτε αντικείμενο χρειαζόμαστε πληροφορίες.

Περίληψη Εργασίας

Το τεχνικό κομμάτι θα αποτελείται από το τετραελικόπτερο και το χειριστήριο πλοήγησης του, τα οποία θα χρησιμοποιούν την πλατφόρμα ARDUINO.
Για την διεκπεραίωση της θα χρειαστεί να ανευρεθούν τα κατάλληλα εξαρτήματα που απαιτούνται για να καλυφθούν οι ανάγκες που θέλουμε να καλύπτει το τετραελικόπτερο(π.χ. διάρκεια πτήσης, ωφέλιμο βάρος κλπ.).
Έχοντας αποφασίσει τα τεχνικά χαρακτηριστικά των εξαρτημάτων, θα γίνει έρευνα αγοράς και κατόπιν η προμήθεια τους.
Στη συνέχεια θα γίνει συναρμολόγηση των επιμέρους στοιχείων του τετραελικοπτέρου και των στοιχείων του χειριστηρίου πλοήγησής του.
Θα γίνει προγραμματισμός της πλακέτας που θα χρησιμοποιεί το τετραελικόπτερο καθώς και της πλακέτας του χειριστηρίου πλοήγησής του.
Το τετραελικόπτερο προβλέπεται να εκτελεί της λειτουργίες πλοήγησης : pitch,roll,yaw.
Η ολοκλήρωση του έργου θα επιτευχθεί με την επικοινωνιακή ζεύξη των δύο κατασκευών, με την χρήση ραδιοκυμάτων, καθώς και με τον έλεγχο του ολοκληρωμένου πλέον συστήματος ότι πληρεί τις αρχικές μας προδιάγραφες

PURPOSE & SUMMARY OF THESIS

The purpose of this thesis is the making of a quadcopter and the control of its flight, using the Arduino platform.
The technical part of this project consists of the assembly of the quadcopter and the making of its remote controller which will use the Arduino platform. For doing so we will find out the tools and parts that we need to have to cover the needs of the copter(e.g time of flight, beneficial weight). Having now known the specification that our parts need ,we will do a market research and acquire the necessary equipment. After that its assembly time of our quadcopter and its remote controller. Now its the programming phase of quadcopter's flight controller and its remote controller. The quadcopter should be able to execute the following functions : pitch, roll, yaw.
The completion of this project will be achieved through radio link communication of the two components as well with the test that our now completed system conforms on our specifications.

ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ

Το τετραελικόπτερο είναι η λύση στο πρόβλημα της κάθετης απογείωσης. Το πρώτο καταγεγραμμένο τετραελικόπτερο δημιουργήθηκε το 1908 από τον Louis Bergeut. Η εξέλιξη και η ανάπτυξη του συνέχισε με τα χρόνια ,αλλά η πραγματική της άνθιση ήρθε με την εποχή της Πληροφορίας και του Ίντερνετ, η οποία αναπτύχθηκε την δεκαετία του 1970,ξεδιπλώθηκε το 1980 και αναπτύσσετε μέχρι και σήμερα .

Πριν την Ψηφιακή Εποχή, τα μη επανδρωμένα εναέρια οχήματα (unmanned aerial vehicles, UAV) ή drones ήταν σε περιορισμένη χρήση, διαθέτοντας είτε περιορισμένη δυνατότητα καθοδήγησης είτε ευάλωτη ζεύξη ραδιοελέγχου.

Η εξέλιξη των ελαφρών και χαμηλού κόστους αισθητήρων όπως οι ψηφιακές κάμερες μαζί με τις τεχνολογίες φορητών υπολογιστών επέτρεψαν στα μη επανδρωμένα εναέρια οχήματα να εξελιχθούν και να λαμβάνουν αυτόνομες αποφάσεις κατά τη διάρκεια της πτήσης. Τα μη επανδρωμένα εναέρια οχήματα έχουν αυξανόμενη χρήση τόσο για πολιτικούς όσο και για στρατιωτικούς σκοπούς.

Τα ποιο απλά παραδείγματα είναι, η λήψη φωτογραφίας ή βίντεο από γωνίες μη προσβάσιμες με τα συνηθισμένα μέσα και στους στρατιωτικούς σκοπούς τα χειρουργικά χτυπήματα αέρος-εδάφους στο Αφγανιστάν.

ΣΤΑΔΙΑ ΔΗΜΙΟΥΡΓΙΑΣ

Κάθε αρχή και δύσκολη λένε και αυτό ισχυεί και για την δική μας περίπτωση. Αυτή τη στιγμή δεν γνωρίζουμε απολύτως τίποτα για τις τεχνικές προδιαγραφές των εξαρτημάτων που θα χρειαστούμε. Ευτυχώς θα μας λύσει τα χέρια η ιστοσελίδα ecalc.ch . Η συγκεκριμένη ιστοσελίδα είναι αφοσιωμένη στα κινητήρια συστήματα. Σε έμπειρα χέρια και γνώστες του αντικειμένου θα μπορούσε να χρησιμοποιήθει για πρόβλεψη συμπεριφοράς ενός μοντέλου, στα δικά μας χέρια μπορεί να μας πει τι ακρίβως χρειαζόμαστε για να έχουμε ένα τετρακόπτερο γνωρίζοντας μόνο το μέγεθος του και το βάρος του. Γνωρίζοντας ότι ήθελα τον σκελετό των 450mm και υπολογίζοντας στο περίπου οτι θα ήταν στα 2 κιλά,η ιστοσελίδα αυτή μας δίνει τις απαντήσεις που ψάχνουμε. Γνωρίζοντας πλέον τι χρειαζόμαστε ,είμαστε έτοιμοι να κάνουμε την έρευνα αγοράς για τα εξαρτήματα μας και την προμήθευση τους. Οι ιστοσελίδες που χρησιμοποίησα για αυτόν τον σκοπό αυτό ήταν: ebay, aliexpress, hobbyking(ειδικό για μοντελισμό) και banggood.

Η πλακέτα ελέγχου πτήσης

Υπάρχουν πολλές πλακέτες που θα μπορούσαμε να χρησιμοποιήσουμε για τον ελέγχου του τετραελικοπτέρου μας. Στην δική μας περίπτωση θα χρησιμοποιήσουμε μια CRIUS MultiWii v2.5, που χρησιμοποιεί έναν ATMELMega 328P μικροεπεξεργαστή και έχει ενσωματωμένο επιταχυνσιόμετρο, γυροσκόπιο, βαρόμετρο και μαγνητόμετρο. Ονομάστηκε από το έργο λογισμικού ανοιχτού κώδικα MultiWii, το οποίο στοχεύει στον έλεγχο ραδιοελεγχούμενων πολυκινητηρίων ιπτάμενων συστημάτων. Περισσότερες πληροφορίες για το έργο μπορούν να βρεθούν εδώ https://code.google.com/archive/p/multiwii/, επίσης σε αυτήν την ιστοσελίδα θα βρούμε τους κώδικες και τα προγράμματα που θα χρειαστούμε για την διαμόρφωση της πλακέτας μας.

ΣΥΝΑΡΜΟΛΟΓΗΣΗ

Τι θα χρειαστούμε:

Για το τετραελικόπτερο:
α) σκελετο 450mm xquad

β) 4 κινητήρες 2216 920KV(2 με δεξιοστροφο κλειδωμα και 2 με αριστεροστροφο)

γ) 4 ηλεκτρονικούς ελεγχούς ταχύτητας (ESC) favourite littlebee 20a opto

δ) μπαταρία 11.1v 3s lipo 5000 45c

ε) κεραία orange rx 2.4ghz
ζ) ελεγκτής πτήσης crius multiwii SE V2.5

Για το <u>χειριστήριο πλοήγησης:</u>

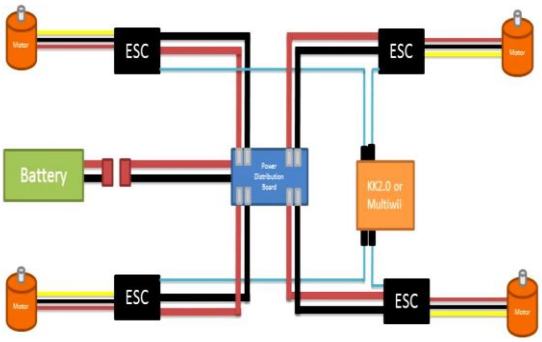
α) χειριστήριο PS3

β) arduino uno μικροελεγκτής

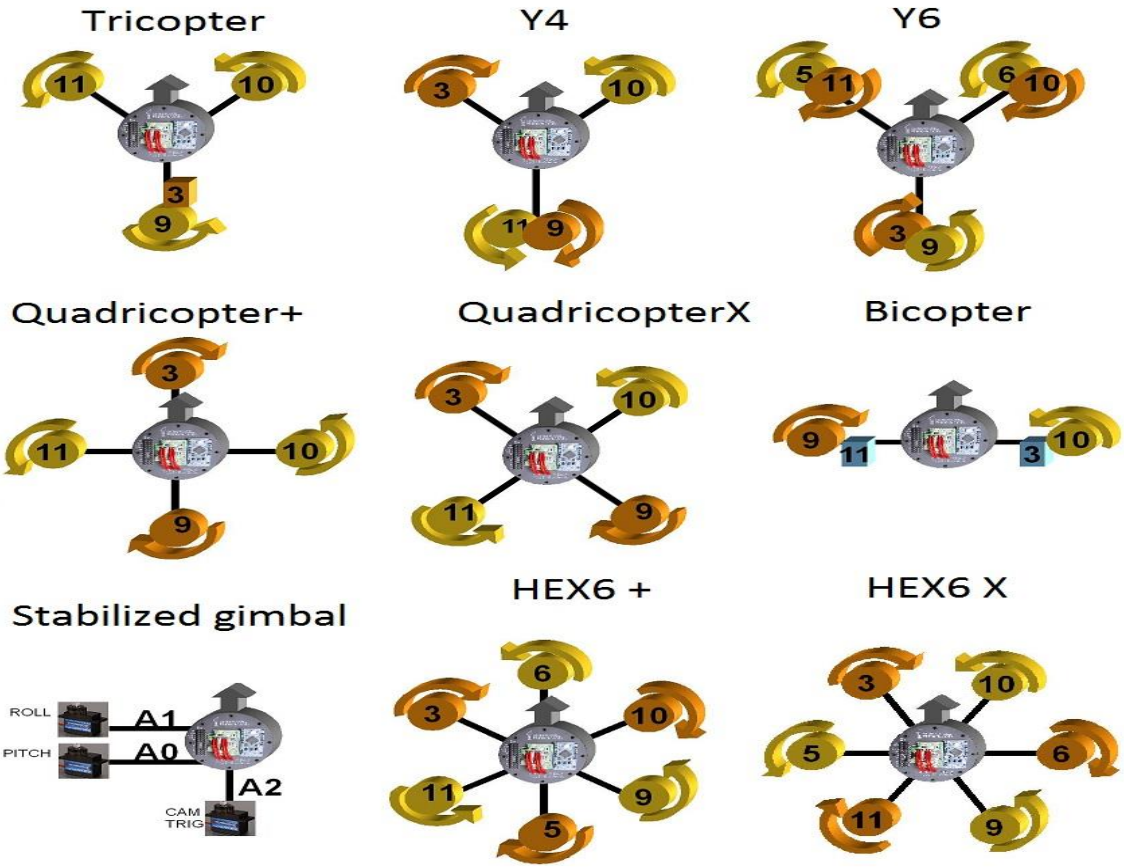
γ) arduino uno USB host shield
δ) κεραία iRangeX IRX4 Plus 2.4G CC2500 NRF24L01 A7105 CYRF6936 4 IN 1 Multiprotocol STM32 TX Module
ε) μπαταριοθήκη των 6
στ) Τα εξαρτήματα μας δεν θα έχουν όλα τα σωστά για εμάς βύσματα. Οπότε θα χρειαστούμε καλώδια, θηλυκά και αρσενικά βύσματα, κολλητήρι, καλάι, μονωτικά,κλπ.

Έχοντας μαζέψει τα υλικά που θα χρειαστούμε μπορούμε να ξεκινήσουμε. Ξεκινάμε τοποθετώντας τους κινητήρες και τα ESC στον σκελετό μας,όπως στην εικ1. Ένα από τα υπέρ του σκελετού μας είναι ότι έχει ενσωματωμένο κύκλωμα μεταφοράς ενέργειας στην βάση του. Οπότε με ένα κολλητήρι και καλάι κολλάμε τα θετικά στα θετικά και τα αρνητικά στα αρνητικά (βάση- esc). Θα κολλήσουμε και ένα θετικό και ένα αρνητικό ακόμα για να συνδέσουμε και την μπαταρία μας (εικ2). Σειρά έχει να συνδέσουμε τα ESC με την πλακέτα μας. Το πώς, εξαρτάται από τον τύπο του πολυκοπτέρου που θα χρησιμοποιήσουμε. Η εικόνα 3 μας δείχνει τον τρόπο ανάλογα με τον τύπο.

εικ1

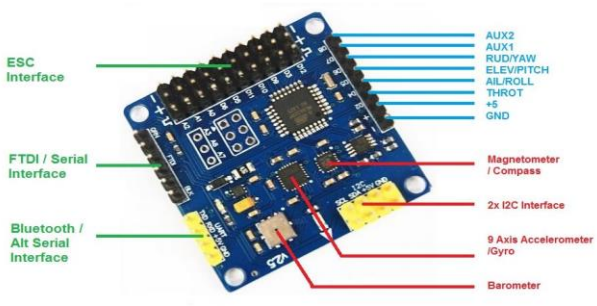


εικ2



Εμείς χρησιμοποιούμε την quad X διαμόρφωση για το τετρακόπτερο μας, οπότε απλώς χρειάζετε να δούμε στην εικόνα αυτής της διαμόρφωσης για να μάθουμε ποιο ESC πρέπει να συνδεθεί σε ποιά ακίδα της πλακέτας μας. Για παράδειγμα, εάν ο κινητήρας έχει το νούμερο 10 στην εικόνα, τότε συνδέουμε το αντίστοιχο ESC καλώδιο στην ακίδα D10 της πλακέτας μας. Προσοχή στην σειρά των καλωδίων, στα δικά μας ESC το μαύρο καλώδιο είναι η γείωση,το κόκκινο
η τάση Vcc και τα λευκά για τον τετραγωνικό παλμό. Η εικόνα με τις διαμορφώσεις μας δείχνει επίσης και την σωστή φορά περιστροφής των κινητήρων μας, για να αλλάξουμε την περιστροφή του κινητήρα πρέπει απλώς να αντιστρέψουμε 2 από τις 3 ενώσεις του. Εδώ έχει σημασία και το κλείδωμα της προπέλας του κινητήρα. Οι δύο κινητήρες που θα στρέφονται δεξιόστροφα πρέπει να έχουν αριστερόστροφο κλείδωμα και οι άλλοι δύο αντίστοιχα δεξιόστροφο.

Σειρά έχει να συνδέσουμε τον δέκτη μας στην πλακέτα. Η εικόνα 4 θα μας βοηθήσει να κατανοήσουμε καλύτερα την πλακέτα μας. Ενώ η εικόνα 5 μας δείχνει τα κανάλια του δέκτη μας.



εικ 4

εικ 5

Αν κοιτάξουμε τις ακίδες με την μπλε γραμμή στην πλακέτα μας θα δούμε ποιές ακίδες συνδέονται με ποιό κανάλι του δέκτη μας.\

Σύνοψη Συνδέσεων

Πλακέτα	Δέκτης	ESCs
+	BATT(1η σειρά)	
-	BATT(2η σειρά)	
D2	THRO	
D4	AILE	
D5	ELEV	
D6	RUDO	
D7	AUX1	
D8		
D3		ΜΠΡΟΣΤΑ ΑΡΙΣΤΕΡΑ
D9		ΠΙΣΩ ΔΕΞΙΑ
D10		ΜΠΡΟΣΤΑ ΔΕΞΙΑ
D11		ΠΙΣΩ ΑΡΙΣΤΕΡΑ

ΧΕΙΡΗΣΤΗΡΙΟ

Για το χειρηστηρίο απλώς χρειάζεται να ενώσουμε την πλακέτα arduino UNO με την πλακέτα arduino UNO USB host shield,η οποία θα μας επιτρέψει να συνδέσουμε το χειρηστήριο PS3,να το διαβάσουμε και με την βοήθεια της κεραίας μας να επικοινωνήσουμε με το τετρακόπτερο μας. Για να συνδεθεί η κεραία πρέπει η πρώτη της ακίδα να συνδεθεί με την D3 ακίδα της PWM μονάδας του arduino και μετά να τροφοδοτηθεί απο την Vcc και τη γείωση της πλακέτας.

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Διαμόρφωση ελεγκτή πτήσης

Για αυτό το βήμα θα χρειαστούμε τους MultiWii κώδικες και το γραφικό περιβάλλον διεπαφής χρήστη, που μπορεί να βρεθεί εδώ https://code.google.com/archive/p/multiwii/ . Θα χρησιμοποιήσουμε την 2.4 έκδοση που είναι και η τελευταία. Για να τροποποιήσουμε και "ανεβάσουμε" το πρόγραμμα στην πλακέτα μας θα χρησιμοποιήσουμε το arduino πρόγραμμα (Arduino IDE). Μέσα στον φάκελο"MultiWii 2.4\MultiWii", που κατεβάσαμε πριν, θα βρούμε το MultiWii.ino αρχείο, με το οποίο θα έχουμε πρόσβαση στα αρχεία κώδικα στο Arduino IDE.

Η βασική εγκατάσταση

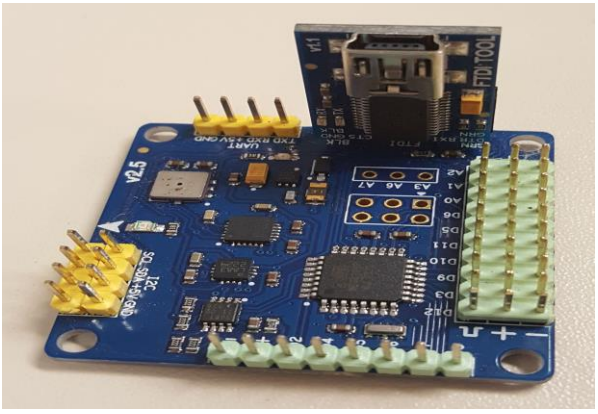
Ανοίγουμε τους κώδικες στο Arduino πρόγραμμα και ψάχνουμε την καρτέλα config.h, εκεί θα βρίσκονται οι παραμέτροι διαμόρφωσης του προγράμματος που θα ανέβει στην πλακέτα μας. Για να διαλέξουμε μία επιλογή πρέπει να την βγάλουμε από τα σχόλια,αυτό γίνετε διαγράφοντας τα "//" μπροστά απο τη γραμμή του κώδικα που θέλουμε να προσθέσουμε. Αυτές είναι οι αλλαγές που ίσως να χρειαστούμε να κάνουμε:

α) ο τύπος του πολυκόπτερου μας: εδώ προσδιορίζουμε τον τύπο του πολυκοπτέρου που θέλουμε να ελέγχουμε την πλακέτα μας. Εμείς έχουμε ένα τετρακόπτερο και επιλέξαμε να χρησιμοποιήσουμε την διαμόρφωση X. Για να την επιλέξουμε πρέπει να βγάλουμε απο τα σχόλια την γραμμή που γράφει "#define QUADX".

β) προσδιορισμός πλακέτας και αισθητήρων: η επιλογή εδώ εξαρτάται από τους αισθητήρες ελέγχου μας. Η πλακέτα μας με τα αισθητήρια της βρίσκετε ήδη στην λίστα αλλά αν αποφασίζαμε να προσθέταμε και άλλα ανεξάρτητα θα πρέπει να τα επιλέξουμε και αυτά. Η επιλογή που ταιριάζει με τον ελεγκτή πτήσης μας είναι "#define CRIUS_SE_v2_0 // Crius MultiWii SE 2.0 with MPU6050, HMC5883 and BMP085".

Για όλες τις άλλες επιλογές θα κρατήσουμε τις αρχικές τιμές, αλλά μπορούμε να τις αλλάξουμε ανάλογα τις ανάγκες μας.

Τώρα μπορούμε να πάμε στο μενού Tools, στο arduino πρόγραμμα, να επιλέξουμε την συμβατή πλακέτα και επεξεργαστή. Για το δικό μας MultiWii χρησιμοποιούμε "Arduino Pro or Pro mini" με τον "Atmega328 (5V,16MHz)". Για να συνδέσουμε το MultiWii με τον υπολογιστή θα χρησιμοποιήσουμε ένα εργαλείο που λέγετε FTDI, για να συνδεθούν σωστά πρέπει απλώς να προσέξουμε τις ενδείξεις BLK και GRN και στις 2 πλακέτες, όπως στην εικόνα 6. Έχοντας κάνει αυτό πατάμε το κουμπί "Verify" για σύνταξη του κώδικα(compile), και αν όλα είναι εντάξει τότε πατάμε το κουμπί "Upload" για να "ανεβάσουμε" τον κώδικα μας στην πλακέτα μας.



εικ 6

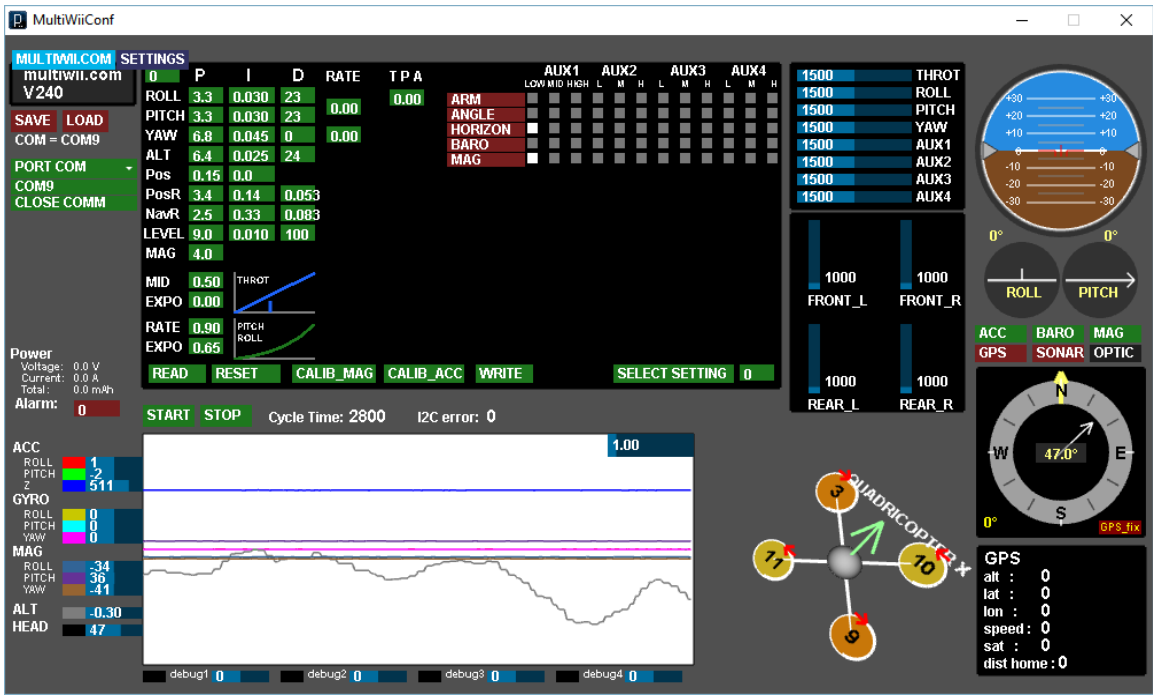
Βαθμονόμηση

Αφού ανέβηκε το πρόγραμμα μας στην πλακέτα, μπορούμε να χρησιμοποιήσουμε το MultiWiiConf GUI, που κατεβάσαμε μαζί με τους κώδικες. Στον φάκελο MultiWiiConf πρέπει να βρούμε τον φάκελο που αντιστοιχεί στο δικό μας λειτουργικό σύστημα και να τρέξουμε την εφαρμογή MultiWiiConf.
Με το MultiWiiConf GUI ανοιχτό, επιλέγουμε την θύρα(COM) στην οποία έχει συνδεθεί η πλακέτα μας(μπορούμε να το βρούμε από τον διαχειριστή συσκευών), πατάμε READ και μετά START. Όλες οι πληροφορίες μας για την πλακέτα μας θα αναπαριστώνται στο GUI, όπως στην εικόνα 7.
Τώρα πρέπει να βαθμονομήσουμε κάποιους αισθητήρες της πλακέτας μας.

- Επιταχυνσιόμετρο: βάζουμε την πλακέτα σε μία επίπεδη επιφάνεια και πατάμε το κουμπί CALIB_ACC. Δεν μετακινούμε την πλακέτα μέχρι να σβήσει το μπλε φωτάκι της.
- Πυξίδα: πατάμε το κουμπί CALIB_MAG και για τα επόμενα 30 δευτερόλεπτα γυρίζουμε την πλακέτα γύρω από όλους τους άξονες της.

Τρόποι πτήσης(Flight modes)

Το MultiWii παρέχει διάφορους τρόπους πτήσεις, ανάλογα τους αισθητήρες που χρησιμοποιούμε και τις ανάγκες που έχουμε κατά τον έλεγχο του πολυκοπτέρου μας.



εικ 7

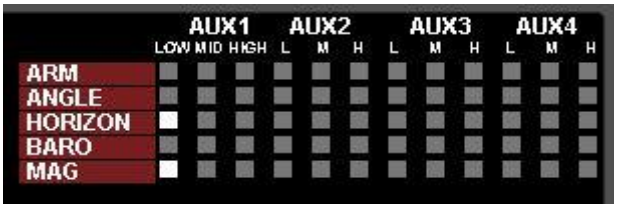
Η εικόνα 8 μας δείχνει ποιός τρόπος χρησιμοποιεί κάθε αισθητήρα (χ=αναγκαίο εξάρτημα, ο= προτεινόμενο εξάρτημα).

- Arm (Acro Mode): δείχνει αν το πολυκόπτερο είναι “οπλισμένο” ή όχι για πτήση. Εάν τίποτα άλλο δεν είναι ενεργοποιμένο,χρησιμοποιείται αυτόματα αυτός ο τρόπος. Η γωνία που αποστέλεται από τον ραδιοπομπό μάς λέει πόσο γρήγορ πρέπει το τετρακόπτερο μας να στρέφεται γύρω απο τους άξονες του και έαν αφήσουμε τον έλεγχου του θα συνεχίσει να κινείται (όχι να χαθεί επικοινωνία,αλλά έαν απλως αφήσουμε το χειρηστήριο στην άκρη).
- Angle(Self Level Mode): με αυτόν τον τρόπο ο ελεγκτής πτήσης χρησιμοποιεί τον επιταχυνσιόμετρο και το γυροσκόπιο να διατηρεί το επίπεδο του πολυκοπτέρου σταθερό όσο δεν υπάρχει έλεγχος.
- Horizon: είναι ένας συνδιασμός μεταξύ των Arm και Angle. Θα συμπεριφέρεται ως angle όσο υπάρχει μικρή χρήση του ραδιοπομπού και ως acro όταν ο πομπός βρίσκετε υπό μεγάλο φορτίο(πχ με το ίδιο drone τραβαμε σταθερά βιντεο με μικρές κινήσεις και μετατοπίσεις και μετά με το ίδο drone μπορούμε να κάνουμε αγωνιστικές πτήσεις).
- Mag: με το mag ενεργοποιημένο ο ελεγκτής πτήσης θα χρησιμοποιεί το μαγνητόμετρο για να κρατάει το πολυκόπτερο σταθερά να δείχνει προς μια κατεύθυνση, όταν δεν ελέγχεται το yaw του.
- Baro: χρησιμοποιεί το βαρόμετρο για να κρατάει το πολυκόπτερο μας σε σταθερό υψόμετρο.
- GPS hold: διατηρεί την θέση του χρησιμοποιώντας το GPS και το BARO
- GPS return home: χρησιμοποιεί το GPS και το μαγνητόμετρο για την επιστροφή του πολυκόπτερου στο αρχικό σημείο.

Για να ενεργοποιήσουμε έναν τρόπο πτήσης πρέπει να συνδέσουμε την πλακέτα μας με τον υπολογιστή, να ανοίξουμε το MultiWiiConf GUI, επιλέγουμε την θύρα μας(COM), πατάμε READ,START και τσεκάρουμε τα κουτιά που ταιριάζουν με τον επιθυμητό τρόπο πτήσης και την βοηθητική κατάσταση(AUX), όπως στην εικόνα 9. Οι τρόποι μπορούν να συνδυαστούν μεταξύ των διαφορετικών καταστάσεων για καλύψουν διαφορετικές ανάγκες (στην δική μας περίπτωση έχουμε μόνο μια βοηθητική κατάσταση AUX1).

MODE	Gyroscope	Accelerometer	Barometer	Magnetometer	GPS
Acro/Gyro only	x				
Stable/Angle	x	x			
Horizon	x	x			
HeadFree	x	x	x	x	o
Altitude Hold	x	x	x		
GPS return home	x	x	o	x	x
GPS waypoint	x	x	o	x	x
GPS position hold	x	x	o	x	x
Failsafe	x				

εικ 8



εικ 9

Συντονισμός ελέγχου PID

Ο συντονισμός ελέγχου PID γίνετε μέσω του MultiWiiConf GUI και είναι η λεπτή ρύθμιση του τετρακοπτέρου μας. Καθορίζει την συμπεριφορά του τετρακοπτέρου μας στην απογείωση,προσγείωση και κατά την διάρκεια της πτήσης του. Εδώ μπορεί να βρει κάποιος πληροφορίες για το τί είναι ο PID έλεγχος, πως λειτουργεί, πώς οι τιμές των συντελεστών του θα επηρεάσουν τη στάση του οχήματος μας και πως μπορούμε να τα συντονίσουμε. https://oscarliang.com/quadcopter-pid-explained-tuning/

Το πρόγραμμα MultiWii έχει τις δικές του στάνταρτ τιμές, όπως στην εικόνα 10. Για να αλλάξουμε μια τιμή πατάμε και κρατάμε πάνω της και μετά σέρνουμε τον δείκτη στα δεξιά ή αριστερά,για να αυξήσουμε ή να μειώσουμε αντίστοιχα την τιμή.

0	P	I	D
ROLL	3.3	0.030	23
PITCH	3.3	0.030	23
YAW	6.8	0.045	0
ALT	6.4	0.025	24
Pos	0.15	0.0	
PosR	3.4	0.14	0.053
NavR	2.5	0.33	0.083
LEVEL	9.0	0.010	100
MAG	4.0		

εικ 10

Χειριστήριο Πτήσης

Αποφάσισα ότι ήθελα να χρησιμοποιήσω χειριστήριο PS3 για την πλοήγηση του τετρακοπτέρου μας. Για να γίνει αυτό, έχοντας συνδέσει τα κομμάτια μας πηγαίνουμε στην ιστοσελίδα
https://github.com/felis/USB_Host_Shield_2.0 και κατεβάζουμε την βιβλιοθήκη σε μορφή zip.
Μετά την προσθέτουμε στο arduino IDE. Τα βήματα για να γίνει αυτό είναι ως εξής sketch->include library-> add .zip library και την επιλέγουμε από εκεί που την αποθηκεύσαμε.
Μετά κατευθυνόμαστε στην ιστοσελίδα https://github.com/DroneMesh/USB_PPM και κατεβάζουμε το αρχείο PS3_PPM και το ανοίγουμε στο arduino IDE. Επιλέγουμε θύρα, πλακέτα και μικροεπεξεργαστή, κάνουμε compile και μετά upload. Είμαστε έτοιμοι.

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΠΡΟΤΑΣΕΙΣ ΑΝΑΠΤΥΞΗΣ

Οι παραπάνω σελίδες είναι ένας οδηγός με τις βασικές οδηγίες για την υλοποιήση και ανάπτυξη ενός πολυκοπτέρου χρησιμοποιώντας MultiWii ελεγκτή πτήσης. Ανεξαρτήτως το ότι εμείς επιλέξαμε να χρησιμοποιήσουμε την διαμόρφωση X στο τετρακόπτερο μας, τα βήματα μας μπορούν εύκολα να προσαρμοστούν για να φτιάξουμε άλλου τύπου πολυκόπτερο.

Ως πτυχιακή θα μπορούσαν προστεθούν διάφοροι αισθητήρες ακόμα για μεγαλύτερο έλεγχο ή/και επιλογές πτήσεις. Ένα πολύ ωραίο παράδειγμα είναι η χρήση σόναρ, δίνοντας έτσι στο drone μας την ικανότητα να αναγνωρίζει και να απογεύγει εμπόδια. Ενώ ένα απλό παράδειγμα θα ήταν η ενσωμάτωση κάμερας με διπλή επικοινωνία εικόνας ήχου ή όπως αναφέραμε σε προηγούμενη σελίδα θα μπορούσαμε να βάζαμε GPS και να αξιοποιήσουμε τις λειτουργίες πτήσης του.

Έαν κάποιος αισθάνεται σίγουρος για τις γνώσεις του στον προγραμματισμό θα μπορούσε να εμβαθύνει και να γράψει κώδικες (υπορουτίνες). Για παράδειγμα με το πάτημα ενός κουμπιού να κάνει κάποιο κόλπο στον αέρα (flip). Επίσης με τις κατάλληλες γνώσεις και εξοπλισμό θα μπορούσε κανείς να κάνει ένα drone να κάνει χαρτογράφηση(mapping) ενός χώρου και να αναπαραστήσει το τρισδιάστατο μοντέλο του.

Τα μη επανδρωμένα εναέρια οχήματα είναι μια πάρα πολύ ωραία ιδέα, η οποία συνεχώς εξελίσσετε.
Ο κόσμος χρησιμοποιεί πλέον drones όχι απλά για φωτογραφήσεις, αλλά από αγώνες ταχύτητας μέχρι διανομή (φαγήτο ή οτιδήποτε ,πχ βλέπε amazon) μέχρι ακόμα δημιουργήθηκε drone για ψάρεμα. Αυτό που θέλω να πω είναι ότι το μόνο που μας σταματάει είναι η φαντασία μας και η όρεξη μας.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- https://ecalc.ch
- https://oscarliang.com/learn-flying-fpv-multirotors/
- https://code.google.com/archive/p/multiwii/
- http://www.multiwii.com
- https://oscarliang.com/multiwii-different-flight-modes-names-gui/
- https://www.youtube.com/watch?v=Zm8z4RLkFGU&list=PLYsWjANuAm4pD7MHpxjSxkAV6KmWYPcH7&index=1
- https://www.youtube.com/watch?v=H-vDhhyKWho&list=PLYsWjANuAm4pD7MHpxjSxkAV6KmWYPcH7&index=2
- https://www.youtube.com/watch?v=lc4ECi6JYX4&list=PLYsWjANuAm4pD7MHpxjSxkAV6KmWYPcH7&index=3
- https://www.youtube.com/watch?v=G1Pz_2tldHM&list=PLYsWjANuAm4pD7MHpxjSxkAV6KmWYPcH7&index=4
- https://www.youtube.com/watch?v=E45mxwl-jxM&list=PLYsWjANuAm4pD7MHpxjSxkAV6KmWYPcH7&index=5
- https://www.youtube.com/watch?v=Foc8lxDddHw&list=PLYsWjANuAm4pD7MHpxjSxkAV6KmWYPcH7&index=7
- https://www.youtube.com/watch?v=Foc8lxDddHw&list=PLYsWjANuAm4pD7MHpxjSxkAV6KmWYPcH7&index=7
- https://www.youtube.com/watch?v=YPtxHw3DWrg&list=PLzidsatoEzeieT03YQ6-LpO0bR1yfEZpx
- https://www.youtube.com/watch?v=4KO3FYeOZ7I&list=PLzidsatoEzeieT03YQ6-LpO0bR1yfEZpx&index=2
- https://www.youtube.com/watch?v=PA3xtLZU_s4&list=PLzidsatoEzeieT03YQ6-LpO0bR1yfEZpx&index=3
- https://www.youtube.com/watch?v=Ja5-JjGc5DU&list=PLzidsatoEzeieT03YQ6-LpO0bR1yfEZpx&index=4
- https://www.youtube.com/watch?v=_ekFXTn3GhE&list=PLzidsatoEzeieT03YQ6-LpO0bR1yfEZpx&index=5
- https://www.youtube.com/watch?v=lA3R6zLa40Y&list=PLzidsatoEzeieT03YQ6-LpO0bR1yfEZpx&index=6
- https://www.youtube.com/watch?v=9v_xaMAJIfM&list=PLzidsatoEzeieT03YQ6-LpO0bR1yfEZpx&index=7
-
- https://www.arduino.cc/en/Main/Software
- https://www.youtube.com/watch?v=4QKKIAVuhNs
- https://interestingengineering.com/a-brief-history-of-drones-the-

remote-controlled-unmanned-aerial-vehicles-uavs#:~:text=Some%20of%20the%20earliest%20military%20drones%20appeared%20in%20the%20mid-1850s&text=The%20concept%20of%20drones%20may,incendiary%20balloons%20over%20the%20city.

- https://lisha.ufsc.br/MultiWii+Quadcopters+guide
- https://www.youtube.com/watch?v=yxUEpmDe9z8&list=PLzidsatoEzeieT03YQ6-LpO0bR1yfEZpx&index=8
- https://www.youtube.com/watch?v=o-inQZn3AgU&list=PLzidsatoEzeieT03YQ6-LpO0bR1yfEZpx&index=9
- https://www.youtube.com/watch?v=NCGq8nxu3E4&list=PLzidsatoEzeieT03YQ6-LpO0bR1yfEZpx&index=10

ΠΑΡΑΡΤΗΜΑ

ΚΩΔΙΚΕΣ ΣΥΣΤΗΜΑΤΟΣ

Κώδικας MultiWii.

Καρτέλα MultiWii

```
/*
 * Welcome to MultiWii.
 *
 * If you see this message, chances are you are using the Arduino IDE. That is ok.
 * To get the MultiWii program configured for your copter, you must switch to the tab named 'config.h'.
 * Maybe that tab is not visible in the list at the top, then you must use the drop down list at the right
 * to access that tab. In that tab you must enable your baord or sensors and optionally various features.
 * For more info go to http://www.multiwii.com/wiki/index.php?title=Main_Page
 *
 * Have fun, and do not forget MultiWii is made possible and brought to you under the GPL License.
 *
 */
```

Καρτέλα Alarms.cpp

```
#include "Arduino.h"
#include "config.h"
#include "def.h"
#include "types.h"
#include "MultiWii.h"
#include "LCD.h"
#include "Sensors.h"
#include "Alarms.h"

void alarmPatternComposer();
void patternDecode(uint8_t resource,uint16_t first,uint16_t second,uint16_t third,uint16_t cyclepause, uint16_t endpause);
void setTiming(uint8_t resource, uint16_t pulse, uint16_t pause);
void turnOff(uint8_t resource);
void toggleResource(uint8_t resource, uint8_t activate);
void vario_output(uint16_t d, uint8_t up);
void inline switch_led_flasher(uint8_t on);
void inline switch_landing_lights(uint8_t on);
void PilotLampSequence(uint16_t speed, uint16_t pattern, uint8_t num_patterns);

static uint8_t cycleDone[5]={0,0,0,0,0},
               resourceIsOn[5] = {0,0,0,0,0};
static uint32_t LastToggleTime[5] ={0,0,0,0,0};
static int16_t  i2c_errors_count_old = 0;
```

```
static uint8_t SequenceActive[5]={0,0,0,0,0};

#if defined(BUZZER)
  uint8_t isBuzzerON(void) { return resourceIsOn[1]; } // returns true while buzzer is
buzzing; returns 0 for silent periods
#else
  uint8_t isBuzzerON() { return 0; }
#endif  //end of buzzer define
/*********************************************************************/
/****                    Alarm Handling                         ****/
/*********************************************************************/
/*
AlarmArray
0: toggle
1: failsafe
2: noGPS
3: beeperOn
4: pMeter
5: runtime
6: vBat
7: confirmation
8: Acc
9: I2C Error
*/
/*
Resources:
0: onboard LED
1: Buzzer
2: PL GREEN
3: PL BLUE
4: PL RED
*/
void alarmHandler(void){

  #if defined(RCOPTIONSBEEP)
    static uint8_t i = 0,firstrun = 1, last_rcOptions[CHECKBOXITEMS];

    if (last_rcOptions[i] != rcOptions[i])   alarmArray[ALRM_FAC_TOGGLE] =
ALRM_LVL_TOGGLE_1;
      last_rcOptions[i] = rcOptions[i];
      i++;
    if(i >= CHECKBOXITEMS)i=0;

    if(firstrun == 1 && alarmArray[ALRM_FAC_CONFIRM] == ALRM_LVL_OFF)
{
      alarmArray[ALRM_FAC_TOGGLE] = ALRM_LVL_OFF;    //only enable
options beep AFTER gyro init
      alarmArray[ALRM_FAC_BEEPERON] = ALRM_LVL_OFF;
    }
    else firstrun = 0;
```

```
  #endif

  #if defined(FAILSAFE)
    if ( failsafeCnt > (5*FAILSAFE_DELAY) && f.ARMED) {
      alarmArray[ALRM_FAC_FAILSAFE] = ALRM_LVL_FAILSAFE_PANIC;
//set failsafe warning level to 1 while landing
      if (failsafeCnt > 5*(FAILSAFE_DELAY+FAILSAFE_OFF_DELAY))
alarmArray[ALRM_FAC_FAILSAFE] = ALRM_LVL_FAILSAFE_FINDME;
//start "find me" signal after landing
    }
    if ( failsafeCnt > (5*FAILSAFE_DELAY) && !f.ARMED)
alarmArray[ALRM_FAC_FAILSAFE] = ALRM_LVL_FAILSAFE_FINDME;
// tx turned off while motors are off: start "find me" signal
    if ( failsafeCnt == 0) alarmArray[ALRM_FAC_FAILSAFE] = ALRM_LVL_OFF;
// turn off alarm if TX is okay
  #endif

  #if GPS
    if ((f.GPS_mode != GPS_MODE_NONE) && !f.GPS_FIX)
alarmArray[ALRM_FAC_GPS] = ALRM_LVL_GPS_NOFIX;
    else if (!f.GPS_FIX) alarmArray[ALRM_FAC_GPS] = ALRM_LVL_ON;
    else alarmArray[ALRM_FAC_GPS] = ALRM_LVL_OFF;
  #endif

  #if defined(BUZZER)
    if ( rcOptions[BOXBEEPERON] ) alarmArray[ALRM_FAC_BEEPERON] =
ALRM_LVL_ON;
    else alarmArray[ALRM_FAC_BEEPERON] = ALRM_LVL_OFF;
  #endif

  #if defined(POWERMETER)
    if ( (pMeter[PMOTOR_SUM] < pAlarm) || (pAlarm == 0) || !f.ARMED)
alarmArray[ALRM_FAC_PMETER] = ALRM_LVL_OFF;
    else if (pMeter[PMOTOR_SUM] > pAlarm) alarmArray[ALRM_FAC_PMETER]
= ALRM_LVL_ON;
  #endif

  #if defined(ARMEDTIMEWARNING)
    if (ArmedTimeWarningMicroSeconds > 0 && armedTime >=
ArmedTimeWarningMicroSeconds && f.ARMED)
alarmArray[ALRM_FAC_RUNTIME] = ALRM_LVL_ON;
    else alarmArray[ALRM_FAC_RUNTIME] = ALRM_LVL_OFF;
  #endif

  #if defined(VBAT)
    if (vbatMin < conf.vbatlevel_crit) alarmArray[ALRM_FAC_VBAT] =
ALRM_LVL_VBAT_CRIT;
    else if ( (analog.vbat > conf.vbatlevel_warn1)  || (NO_VBAT > analog.vbat))
alarmArray[ALRM_FAC_VBAT] = ALRM_LVL_OFF;
    else if (analog.vbat > conf.vbatlevel_warn2) alarmArray[ALRM_FAC_VBAT] =
```

```
ALRM_LVL_VBAT_INFO;
   else if (analog.vbat > conf.vbatlevel_crit) alarmArray[ALRM_FAC_VBAT] =
ALRM_LVL_VBAT_WARN;
   //else alarmArray[6] = 4;
  #endif

  if (i2c_errors_count > i2c_errors_count_old+100 || i2c_errors_count < -1)
alarmArray[ALRM_FAC_I2CERROR] = ALRM_LVL_ON;
  else alarmArray[ALRM_FAC_I2CERROR] = ALRM_LVL_OFF;
  #if defined(LCD_TELEMETRY) &&
!defined(SUPPRESS_TELEMETRY_PAGE_8)
   if (telemetry == 8) lcd_telemetry(); // must output the alarms states now because
alarmPatternComposer() will reset alarmArray[]
  #endif
  alarmPatternComposer();
}

void alarmPatternComposer(){
  static char resource = 0;
  // patternDecode(length1,length2,length3,beeppause,endpause,loop)
  #if defined(BUZZER)
   resource = 1;                                                    //buzzer selected
   if     ( IS_ALARM_SET(ALRM_FAC_FAILSAFE ,
ALRM_LVL_FAILSAFE_FINDME) )      patternDecode(resource,200,0,0,50,2000);
//failsafe "find me" signal
   else if ( IS_ALARM_SET(ALRM_FAC_FAILSAFE ,
ALRM_LVL_FAILSAFE_PANIC) ||
             IS_ALARM_SET(ALRM_FAC_ACC , ALRM_LVL_ON) )
patternDecode(resource,50,200,200,50,50); //failsafe "panic"  or Acc not calibrated
   else if ( IS_ALARM_SET(ALRM_FAC_TOGGLE , ALRM_LVL_TOGGLE_1) )
patternDecode(resource,50,0,0,50,0);          //toggle 1
   else if ( IS_ALARM_SET(ALRM_FAC_TOGGLE , ALRM_LVL_TOGGLE_2) )
patternDecode(resource,50,50,0,50,0);         //toggle 2
   else if ( IS_ALARM_SET(ALRM_FAC_TOGGLE ,
ALRM_LVL_TOGGLE_ELSE) )  patternDecode(resource,50,50,50,50,0);     //toggle
else
   #if GPS
    else if ( IS_ALARM_SET(ALRM_FAC_GPS , ALRM_LVL_GPS_NOFIX) )
patternDecode(resource,50,50,0,50,50);        //gps installed but no fix
   #endif
   else if ( IS_ALARM_SET(ALRM_FAC_BEEPERON , ALRM_LVL_ON) )
patternDecode(resource,50,50,50,50,50);          //BeeperOn
   #ifdef POWERMETER
    else if ( IS_ALARM_SET(ALRM_FAC_PMETER , ALRM_LVL_ON) )
patternDecode(resource,50,50,0,50,120);             //pMeter Warning
   #endif
   else if ( IS_ALARM_SET(ALRM_FAC_RUNTIME , ALRM_LVL_ON) )
patternDecode(resource,50,50,50,50,0);             //Runtime warning
   #ifdef VBAT
    else if ( IS_ALARM_SET(ALRM_FAC_VBAT , ALRM_LVL_VBAT_CRIT) )
```

```
patternDecode(resource,50,50,200,50,2000);      //vbat critical
      else if ( IS_ALARM_SET(ALRM_FAC_VBAT , ALRM_LVL_VBAT_WARN) )
patternDecode(resource,50,200,0,50,2000);       //vbat warning
      else if ( IS_ALARM_SET(ALRM_FAC_VBAT , ALRM_LVL_VBAT_INFO) )
patternDecode(resource,200,0,0,50,2000);        //vbat info
    #endif
    else if ( IS_ALARM_SET(ALRM_FAC_CONFIRM , ALRM_LVL_CONFIRM_1)
) patternDecode(resource,200,0,0,50,200);      //confirmation indicator 1x
    else if ( IS_ALARM_SET(ALRM_FAC_CONFIRM , ALRM_LVL_CONFIRM_2)
) patternDecode(resource,200,200,0,50,200);    //confirmation indicator 2x
    else if ( IS_ALARM_SET(ALRM_FAC_CONFIRM ,
ALRM_LVL_CONFIRM_ELSE) ) patternDecode(resource,200,200,200,50,200);
//confirmation indicator 3x
    else if (SequenceActive[(uint8_t)resource] == 1)
patternDecode(resource,0,0,0,0,0);                  // finish last sequence if not finished
yet
    else turnOff(resource);                                                  // turn off the
resource
    alarmArray[ALRM_FAC_ACC] = ALRM_LVL_OFF;
//reset acc not calibrated

  #endif
  #if defined(PILOTLAMP)
    if ( IS_ALARM_SET(ALRM_FAC_I2CERROR , ALRM_LVL_ON) )
PilotLampSequence(100,B000111,2);                    //I2C Error
    else if ( IS_ALARM_SET(ALRM_FAC_BEEPERON , ALRM_LVL_ON) )
PilotLampSequence(100,B0101<<8|B00010001,4);                 //BeeperOn
    else{
      resource = 2;
      if (f.ARMED && f.ANGLE_MODE)
patternDecode(resource,100,100,100,100,1000);              //Green Slow Blink-->angle
      else if (f.ARMED && f.HORIZON_MODE)
patternDecode(resource,200,200,200,100,1000);         //Green mid Blink-->horizon
      else if (f.ARMED) patternDecode(resource,100,100,0,100,1000);
//Green fast Blink-->acro
      else turnOff(resource);                                                //switch off
      resource = 3;
      #if GPS
        if (  IS_ALARM_SET(ALRM_FAC_GPS , ALRM_LVL_ON) )
patternDecode(resource,100,100,100,100,100); // blue fast blink -->no gps fix
        else if (f.GPS_mode != GPS_MODE_NONE)
patternDecode(resource,100,100,100,100,1000);          //blue slow blink --> gps
active
        else setTiming(resource,100,1000);                                          //blue
short blink -->gps fix ok
      #else
        turnOff(resource);
      #endif
      resource = 4;
      if ( IS_ALARM_SET(ALRM_FAC_FAILSAFE ,
```

```
ALRM_LVL_FAILSAFE_PANIC) ) setTiming(resource,100,100);                //Red
fast blink--> failsafe panic
     else if ( IS_ALARM_SET(ALRM_FAC_FAILSAFE ,
ALRM_LVL_FAILSAFE_FINDME) ) patternDecode(resource,1000,0,0,0,2000);
//red slow blink--> failsafe find me
     else turnOff(resource);
   }
 #endif
}

void patternDecode(uint8_t resource,uint16_t first,uint16_t second,uint16_t
third,uint16_t cyclepause, uint16_t endpause){
  static uint16_t pattern[5][5];
  static uint8_t icnt[5] = {0,0,0,0,0};

  if(SequenceActive[resource] == 0){
    SequenceActive[resource] = 1;
    pattern[resource][0] = first;
    pattern[resource][1] = second;
    pattern[resource][2] = third;
    pattern[resource][3] = endpause;
    pattern[resource][4] = cyclepause;
  }
  if(icnt[resource] <3 ){
    if (pattern[resource][icnt[resource]] != 0){
      setTiming(resource,pattern[resource][icnt[resource]],pattern[resource][4]);
     }
  }
  else if (LastToggleTime[resource] < (millis()-pattern[resource][3]))  {  //sequence is
over: reset everything
    icnt[resource]=0;
    SequenceActive[resource] = 0;                          //sequence is now done,
cycleDone sequence may begin
    alarmArray[ALRM_FAC_TOGGLE] = ALRM_LVL_OFF;
//reset toggle bit
    alarmArray[ALRM_FAC_CONFIRM] = ALRM_LVL_OFF;
//reset confirmation bit
    turnOff(resource);
    return;
  }
  if (cycleDone[resource] == 1 || pattern[resource][icnt[resource]] == 0) {
//single on off cycle is done
    if (icnt[resource] < 3) {
      icnt[resource]++;
    }
    cycleDone[resource] = 0;
    turnOff(resource);
  }
}
```

```
void turnOff(uint8_t resource){
  if (resource == 1) {
    if (resourceIsOn[1]) {
      BUZZERPIN_OFF;
      resourceIsOn[1] = 0;
    }
  }else if (resource == 0) {
    if (resourceIsOn[0]) {
      resourceIsOn[0] = 0;
      LEDPIN_OFF;
    }
  }else if (resource == 2) {
    if (resourceIsOn[2]) {
      resourceIsOn[2] = 0;
      #if defined (PILOTLAMP)
        PilotLamp(PL_GRN_OFF);
      #endif
    }
  }else if (resource == 3) {
    if (resourceIsOn[3]) {
      resourceIsOn[3] = 0;
      #if defined (PILOTLAMP)
        PilotLamp(PL_BLU_OFF);
      #endif
    }
  }else if (resource == 4) {
    if (resourceIsOn[4]) {
      resourceIsOn[4] = 0;
      #if defined (PILOTLAMP)
        PilotLamp(PL_RED_OFF);
      #endif
    }
  }
}

#if defined (PILOTLAMP)
  //original code based on mr.rc-cam and jevermeister work
  //modified by doughboy to use timer interrupt

  #define PL_BUF_SIZE 8
  volatile uint8_t queue[PL_BUF_SIZE]; //circular queue
  volatile uint8_t head = 0;
  volatile uint8_t tail = 0;

/*********************************************************************/
/****                Pilot Lamp Handling                  ****/
/*********************************************************************/


//define your light pattern by bits, 0=off 1=on
```

```
//define up to 5 patterns that cycle using 15 bits, pattern starts at bit 0 in groups of 3
//  14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
//   R  B  G  R  B G R B G R B G R B G
//parameters speed is the ms between patterns
//          pattern is the 16bit (uses only 15 bits) specifying up to 5 patterns
//          num_patterns is the number of patterns defined
//example to define sequential light where G->B->R then back to G, you need 4
patterns
//          B00000101<<8 | B00010001
//example: to define alternate all on and al off, you only need two patterns
//          B00000111
  void PilotLampSequence(uint16_t speed, uint16_t pattern, uint8_t num_patterns){
    static uint32_t lastswitch = 0;
    static uint8_t seqno = 0;
    static uint16_t lastpattern = 0;  //since variables are static, when switching patterns,
the correct pattern will start on the second sequence

    if(millis() < (lastswitch + speed))
      return; //not time to change pattern yet
    lastswitch = millis();

    for (uint8_t i=0;i<3;i++) {
      uint8_t state = (pattern >>(seqno*3+i)) & 1; //get pattern bit
      //since value is multiple of 25, we can calculate time based on pattern position
      //i=0 is green, 1=blue, 2=red, green ON ticks is calculated as 50*(0+1)-1*25 = 25
ticks
      uint8_t tick = 50*(i+1);
      if (state)
        tick -=25;
      PilotLamp(tick);
      resourceIsOn[i+2]=state;
    }
   seqno++;
   seqno%=num_patterns;
  }

  void PilotLamp(uint8_t count){
    if (((tail+1)%PL_BUF_SIZE)!=head) {
      queue[tail]=count;
      tail++;
      tail=(tail%PL_BUF_SIZE);
    }
  }
  //ISR code sensitive to changes, test thoroughly before flying
  ISR(PL_ISR) { //the interrupt service routine
   static uint8_t state = 0;
   uint8_t h = head;
   uint8_t t = tail;
   if (state==0) {
     if (h!=t) {
```

```
      uint8_t c = queue[h];
      PL_PIN_ON;
      PL_CHANNEL+=c;
      h = ((h+1) % PL_BUF_SIZE);
      head = h;
      state=1;
    }
  } else if (state==1) {
    PL_PIN_OFF;
    PL_CHANNEL+=PL_IDLE;
    state=0;
  }
 }
#endif

/****************************************************************************/
/****                      LED Handling                      ****/
/****************************************************************************/
//Beware!! Is working with delays, do not use inflight!

void blinkLED(uint8_t num, uint8_t ontime,uint8_t repeat) {
 uint8_t i,r;
 for (r=0;r<repeat;r++) {
  for(i=0;i<num;i++) {
    #if defined(LED_FLASHER)
      switch_led_flasher(1);
    #endif
    #if defined(LANDING_LIGHTS_DDR)
      switch_landing_lights(1);
    #endif
    LEDPIN_TOGGLE; // switch LEDPIN state
    delay(ontime);
    #if defined(LED_FLASHER)
      switch_led_flasher(0);
    #endif
    #if defined(LANDING_LIGHTS_DDR)
      switch_landing_lights(0);
    #endif
   }
   delay(60); //wait 60 ms
 }
}

/****************************************************************************/
/****                Global Resource Handling                ****/
/****************************************************************************/

 void setTiming(uint8_t resource, uint16_t pulse, uint16_t pause){
   if (!resourceIsOn[resource] && (millis() >= (LastToggleTime[resource] +
pause))&& pulse != 0) {
```

```
    resourceIsOn[resource] = 1;
    toggleResource(resource,1);
    LastToggleTime[resource]=millis();
   } else if ( (resourceIsOn[resource] && (millis() >= LastToggleTime[resource] +
pulse) ) || (pulse==0 && resourceIsOn[resource]) ) {
    resourceIsOn[resource] = 0;
    toggleResource(resource,0);
    LastToggleTime[resource]=millis();
    cycleDone[resource] = 1;
   }
  }

  void toggleResource(uint8_t resource, uint8_t activate){
   switch(resource) {
    #if defined (BUZZER)
     case 1:
      if (activate == 1) {BUZZERPIN_ON;}
      else BUZZERPIN_OFF;
      break;
    #endif
    #if defined (PILOTLAMP)
     case 2:
      if (activate == 1) PilotLamp(PL_GRN_ON);
      else PilotLamp(PL_GRN_OFF);
      break;
     case 3:
      if (activate == 1) PilotLamp(PL_BLU_ON);
      else PilotLamp(PL_BLU_OFF);
      break;
     case 4:
      if (activate == 1) PilotLamp(PL_RED_ON);
      else PilotLamp(PL_RED_OFF);
      break;
    #endif
    case 0:
    default:
     if (activate == 1) {LEDPIN_ON;}
     else LEDPIN_OFF;
     break;
   }
   return;
  }


/*****************************************************************/
/****                LED Ring Handling                ****/
/*****************************************************************/

  #if defined(LED_RING)
```

```
#define LED_RING_ADDRESS 0xDA //7 bits

void i2CLedRingState(void) {
  uint8_t b[10];

  b[0]='M'; // MultiwII mode
  if (f.ARMED) { // Motors running = flying
    if(!(f.ANGLE_MODE||f.HORIZON_MODE)){ //ACRO
      b[0]= 'x';
    } else if(f.GPS_mode == GPS_MODE_RTH){ //RTH
      b[0]= 'w';
    } else if(f.GPS_mode == GPS_MODE_HOLD){//Position Hold
      b[0]= 'v';
    } else if(f.HORIZON_MODE){ //HORIZON mode
      b[0]= 'y';
    } else {
      b[0]= 'u'; // ANGLE mode
    }
    i2c_rep_start(LED_RING_ADDRESS);
    i2c_write(b[0]);
    i2c_stop();
  } else if (!f.ACC_CALIBRATED) { // Multiwii not stable or uncalibrated
    b[0]= 't';
    i2c_rep_start(LED_RING_ADDRESS);
    i2c_write(b[0]);
    i2c_stop();
  } else { // Motors not running = on the ground
    b[0]= 's';
    if (f.ANGLE_MODE) b[1]=1;
    else if (f.HORIZON_MODE) b[1]=2;
    else b[1]= 0;
    if (f.BARO_MODE) b[2]=1;
    else b[2]= 0;
    if (f.MAG_MODE) b[3]=1;
    else b[3]= 0;
#if GPS
    if (rcOptions[BOXGPSHOME]) b[4]=2;
    else if (rcOptions[BOXGPSHOLD]) b[4]=1;
    else b[4]=0;
#else
    b[4]=0;
#endif
    b[5]=(180-att.heading)/2; // 1 unit = 2 degrees;
    b[6]=GPS_numSat;
    i2c_rep_start(LED_RING_ADDRESS);
    for(uint8_t i=0;i<7;i++){
      i2c_write(b[i]);
    }
    i2c_stop();
  }
```

```
  #if defined (VBAT)
    if (analog.vbat < conf.vbatlevel_warn1){ // Uh oh - battery low
      i2c_rep_start(LED_RING_ADDRESS);
      i2c_write('r');
      i2c_stop();
    }
  # endif
  }

  void blinkLedRing(void) {
    uint8_t b[3];
    b[0]='g';
    b[1]= 10;
    b[2]= 10;
    i2c_rep_start(LED_RING_ADDRESS<<1);
    for(uint8_t i=0;i<3;i++)
      i2c_write(b[i]);
    i2c_stop();
  }
#endif

/**************************************************************************/
/****                  LED flasher Handling                    ****/
/**************************************************************************/

#if defined(LED_FLASHER)
  static uint8_t led_flasher_sequence = 0;
  /* if we load a specific sequence and do not want it change, set this flag */
  static enum {
    LED_FLASHER_AUTO,
    LED_FLASHER_CUSTOM
  } led_flasher_control = LED_FLASHER_AUTO;

  void init_led_flasher() {
    #if defined(LED_FLASHER_DDR)
    LED_FLASHER_DDR |= (1<<LED_FLASHER_BIT);
    switch_led_flasher(0);
    #endif
  }

  void led_flasher_set_sequence(uint8_t s) {
    led_flasher_sequence = s;
  }

  void inline switch_led_flasher(uint8_t on) {
    #if defined(LED_FLASHER_DDR)
      #ifndef LED_FLASHER_INVERT
        if (on) {
      #else
        if (!on) {
```

```
    #endif
      LED_FLASHER_PORT |= (1<<LED_FLASHER_BIT);
    } else {
      LED_FLASHER_PORT &= ~(1<<LED_FLASHER_BIT);
    }
  #endif
}

static uint8_t inline led_flasher_on() {
  uint8_t seg = (currentTime/1000/125)%8;
  return (led_flasher_sequence & 1<<seg);
}

void auto_switch_led_flasher() {
  if (led_flasher_on()) {
    switch_led_flasher(1);
  } else {
    switch_led_flasher(0);
  }
}

/* auto-select a fitting sequence according to the
 * copter situation
 */
void led_flasher_autoselect_sequence() {
  if (led_flasher_control != LED_FLASHER_AUTO) return;

  #if defined(LED_FLASHER_SEQUENCE_MAX)
  /* do we want the complete illumination no questions asked? */
  if (rcOptions[BOXLEDMAX]) {
    led_flasher_set_sequence(LED_FLASHER_SEQUENCE_MAX);
    return;
  }
  #endif

  #if defined(LED_FLASHER_SEQUENCE_LOW)
  if (rcOptions[BOXLEDLOW]) {
    led_flasher_set_sequence(LED_FLASHER_SEQUENCE_LOW);
    return;
  }
  #endif

  #if defined(LED_FLASHER_SEQUENCE_ARMED)
  /* do we have a special sequence for armed copters? */
  if (f.ARMED) {
    led_flasher_set_sequence(LED_FLASHER_SEQUENCE_ARMED);
    return;
  }
  #endif
```

```
  /* Let's load the plain old boring sequence as a last resort */
  led_flasher_set_sequence(LED_FLASHER_SEQUENCE);
 }

 #endif

 #if defined(LANDING_LIGHTS_DDR)
 void init_landing_lights(void) {
  LANDING_LIGHTS_DDR |= 1<<LANDING_LIGHTS_BIT;
  switch_landing_lights(0);
 }

 void inline switch_landing_lights(uint8_t on) {
  #ifndef LANDING_LIGHTS_INVERT
  if (on) {
  #else
  if (!on) {
  #endif
   LANDING_LIGHTS_PORT |= 1<<LANDING_LIGHTS_BIT;
  } else {
   LANDING_LIGHTS_PORT &= ~(1<<LANDING_LIGHTS_BIT);
  }
 }

 void auto_switch_landing_lights(void) {
  if (rcOptions[BOXLLIGHTS]
  #if defined(LANDING_LIGHTS_AUTO_ALTITUDE) & SONAR
      || (sonarAlt >= 0 && sonarAlt <= LANDING_LIGHTS_AUTO_ALTITUDE
&& f.ARMED)
  #endif
  #if defined(LED_FLASHER_DDR) &
defined(LANDING_LIGHTS_ADOPT_LED_FLASHER_PATTERN)
      || (led_flasher_on())
  #endif
  ) {
    switch_landing_lights(1);
  } else {
    switch_landing_lights(0);
  }
 }
#endif


/*********************************************************************/
 /****              Variometer signaling                ****/

/*********************************************************************/
#ifdef VARIOMETER
#define TRESHOLD_UP    50          // (m1) treshhold for up velocity
#define TRESHOLD_DOWN  40          // (m1) treshhold for up velocity
```

```
#define TRESHOLD_UP_MINUS_DOWN  10  // (m1) you compute:
TRESHOLD_UP - TRESHOLD_DOWN
#define ALTITUDE_INTERVAL 400       // (m2) in calls; interval to perodically
observe altitude change
#define DELTA_ALT_TRESHOLD 200      // (m2) in cm; treshold for delta altitude
after ALTITUDE_INTERVAL
#define DELTA_T 5                   // (m2) divisor for delta_alt to compute vel
#define SIGNAL_SCALE   4       // you compute: (50ms per beep / 5*3ms cycle time)
#define SILENCE_M      200     // max duration of silence in calls
#define SILENCE_SCALE  33      // vario scale: larger -> slower decay of silence
#define SILENCE_A      6600    // you compute: SILENCE_M * SILENCE_SCALE
#define DURATION_SUP   5       // sup duration of signal
#define DURATION_SCALE 100     // vario scale: larger -> slower rise of length

  /* vario_signaling() gets called every 5th cycle (~2ms - 5ms) -> (~10ms - 25ms)
   * modulates silence duration between tones and tone duration
   * higher abs(vario) -> shorther silence & longer signal duration.
   * Utilize two methods for combined short and long term observation
   */
void vario_signaling(void) {
  static int16_t last_v = 0;
  static uint16_t silence = 0;
  static int16_t max_v = 0;
  static uint8_t max_up = 0;

  uint16_t s = 0;
  int16_t v = 0;

  /* method 1: use vario to follow short term up/down movement : */
  #if (VARIOMETER == 1) || (VARIOMETER == 12)
  {
    uint8_t up = (alt.vario > 0 ? 1 : 0 ); //, down = (vario < 0 ? 1 : 0 );
    //int16_t v = abs(vario) - up * TRESHOLD_UP - down * TRESHOLD_DOWN;
    v = abs(alt.vario) - up * (TRESHOLD_UP_MINUS_DOWN) -
TRESHOLD_DOWN;
    if (silence>0) silence--; else silence = 0;
    if (v > 0) {
      // going up or down
      if (v > last_v) {
        // current speed greater than speed for last signal,
        // so shorten the remaining silence period
        s = (SILENCE_A) / (SILENCE_SCALE + v);
        if (silence > s)  silence = s;
      }
      // remember interim max v
      if (v > max_v) {
        max_v = v;
        max_up = up;
      }
    } // end of (v>0)
```

```
  }
  #endif // end method 1
  /* method 2: use altitude to follow long term up/down movement : */
  #if (VARIOMETER == 2) || (VARIOMETER == 12)
  {
    static uint16_t t = 0;
    if (!(t++ % ALTITUDE_INTERVAL)) {
      static int32_t last_BaroAlt = 0;
      int32_t delta_BaroAlt = alt.EstAlt - last_BaroAlt;
      if (abs(delta_BaroAlt) > DELTA_ALT_TRESHOLD) {
        // inject suitable values
        max_v = abs(delta_BaroAlt / DELTA_T);
        max_up = (delta_BaroAlt > 0 ? 1 : 0);
        silence = 0;
      }
      last_BaroAlt = alt.EstAlt;
    }
  }
  #endif // end method 2
  /* something to signal now? */
  if ( (silence == 0) && (max_v > 0) ) {
    // create new signal
    uint16_t d = (DURATION_SUP * max_v)/(DURATION_SCALE + max_v);
    s = (SILENCE_A) / (SILENCE_SCALE + max_v);
    s+= d * SIGNAL_SCALE;
    vario_output(d, max_up);
    last_v = v;
    max_v = 0;
    max_up = 0;
    silence = s;
  }
} // end of vario_signaling()

void vario_output(uint16_t d, uint8_t up) {
  if (d == 0) return;
  #if defined(SUPPRESS_VARIOMETER_UP)
    if (up) return;
  #elif defined(SUPPRESS_VARIOMETER_DOWN)
    if (!up) return;
  #endif
  #ifdef VARIOMETER_SINGLE_TONE
    uint8_t s1 = 0x07;
    uint8_t d1 = d;
  #else
    uint8_t s1 = (up ? 0x05 : 0x07);
    uint8_t d1 = d/2;
  #endif
  if (d1<1) d1 = 1;
  for (uint8_t i=0; i<d1; i++) LCDprint(s1);
  #ifndef VARIOMETER_SINGLE_TONE
```

```
    uint8_t s2 = (up ? 0x07 : 0x05);
    uint8_t d2 = d-d1;
    if (d2<1) d2 = 1;
    for (uint8_t i=0; i<d2; i++) LCDprint(s2);
  #endif
}

#endif
```
καρτέλα Alarms.h
```
#ifndef ALARMS_H_
#define ALARMS_H_

void blinkLED(uint8_t num, uint8_t ontime,uint8_t repeat);
uint8_t isBuzzerON(void);
void alarmHandler(void);
void vario_signaling(void);
void i2CLedRingState(void);
void blinkLedRing(void);
void auto_switch_led_flasher();
void init_led_flasher();
void led_flasher_set_sequence(uint8_t s);
void led_flasher_autoselect_sequence();
void init_landing_lights(void);
void auto_switch_landing_lights(void);
void PilotLamp(uint8_t count);

/*
AlarmArray
0: toggle
1: failsafe
2: noGPS
3: beeperOn
4: pMeter
5: runtime
6: vBat
7: confirmation
8: Acc
9: I2C Error
*/
enum alrm_fac {
  ALRM_FAC_TOGGLE = 0,
  ALRM_FAC_FAILSAFE,
  ALRM_FAC_GPS,
  ALRM_FAC_PMETER,
  ALRM_FAC_RUNTIME,
  ALRM_FAC_VBAT,
  ALRM_FAC_CONFIRM,
  ALRM_FAC_ACC,
  ALRM_FAC_I2CERROR,
  ALRM_FAC_SIZE, // MUST be LAST - used for size of array alarmArray
```

```
 };


/*
Resources:
0: onboard LED
1: Buzzer
2: PL GREEN
3: PL BLUE
4: PL RED
*/
enum alrm_res {
  ALRM_RES_LED = 0,
  ALRM_RES_BUZZER,
  ALRM_RES_PL_GREEN,
  ALRM_RES_PL_BLUE,
  ALRM_RES_PL_RED,
  ALRM_RES_PL     ,
  ALRM_RES_ANY    ,
};

enum alrm_lvl_onoff {
 ALRM_LVL_OFF = 0,
 ALRM_LVL_ON = 1,
};
enum alrm_lvl_failsafe {
 ALRM_LVL_FAILSAFE_FINDME = 1,
 ALRM_LVL_FAILSAFE_PANIC,
 };
enum alrm_lvl_toggle {
 ALRM_LVL_TOGGLE_1  = 1,
 ALRM_LVL_TOGGLE_2    ,
 ALRM_LVL_TOGGLE_ELSE ,
};
#if GPS
  enum alrm_lvl_gps {
   ALRM_LVL_GPS_NOFIX  = 2,
  };
#endif
#ifdef VBAT
  enum alrm_lvl_vbat {
   ALRM_LVL_VBAT_INFO  = 1,
   ALRM_LVL_VBAT_WARN ,
   ALRM_LVL_VBAT_CRIT ,
  };
#endif
enum alrm_lvl_confirm {
 ALRM_LVL_CONFIRM_1 = 1,
 ALRM_LVL_CONFIRM_2    ,
 ALRM_LVL_CONFIRM_ELSE ,
```

```
};

#define SET_ALARM(fac, level) alarmArray[fac] = level
#ifdef BUZZER
  #define SET_ALARM_BUZZER(fac, level)    SET_ALARM( fac, level)
#else
  #define SET_ALARM_BUZZER(fac, level)
#endif

#define IS_ALARM_SET(fac, level) ( alarmArray[fac] == level )

#endif /* ALARMS_H_ */
```

καρτέλα EEPROM.cpp

```
#include <avr/eeprom.h>
#include "Arduino.h"
#include "config.h"
#include "def.h"
#include "types.h"
#include "EEPROM.h"
#include "MultiWii.h"
#include "Alarms.h"
#include "GPS.h"

void LoadDefaults(void);

uint8_t calculate_sum(uint8_t *cb , uint8_t siz) {
  uint8_t sum=0x55;  // checksum init
  while(--siz) sum += *cb++;  // calculate checksum (without checksum byte)
  return sum;
}

void readGlobalSet() {
  eeprom_read_block((void*)&global_conf, (void*)0, sizeof(global_conf));
  if(calculate_sum((uint8_t*)&global_conf, sizeof(global_conf)) !=
global_conf.checksum) {
    global_conf.currentSet = 0;
    global_conf.accZero[ROLL] = 5000;    // for config error signalization
  }
}

bool readEEPROM() {
  uint8_t i;
  int8_t tmp;
  uint8_t y;

  #ifdef MULTIPLE_CONFIGURATION_PROFILES
    if(global_conf.currentSet>2) global_conf.currentSet=0;
  #else
    global_conf.currentSet=0;
  #endif
```

```
  eeprom_read_block((void*)&conf, (void*)(global_conf.currentSet * sizeof(conf) +
sizeof(global_conf)), sizeof(conf));
  if(calculate_sum((uint8_t*)&conf, sizeof(conf)) != conf.checksum) {
    blinkLED(6,100,3);
    SET_ALARM_BUZZER(ALRM_FAC_CONFIRM,
ALRM_LVL_CONFIRM_ELSE);
    LoadDefaults();                // force load defaults
    return false;                  // defaults loaded, don't reload constants (EEPROM life
saving)
  }
  // 500/128 = 3.90625    3.9062 * 3.9062 = 15.259   1526*100/128 = 1192
  for(i=0;i<5;i++) {
    lookupPitchRollRC[i] = (1526+conf.rcExpo8*(i*i-
15))*i*(int32_t)conf.rcRate8/1192;
  }
  for(i=0;i<11;i++) {
    tmp = 10*i-conf.thrMid8;
    y = conf.thrMid8;
    if (tmp>0) y = 100-y;
    lookupThrottleRC[i] = 100*conf.thrMid8 + tmp*(
(int32_t)conf.thrExpo8*(tmp*tmp)/((uint16_t)y*y)+100-conf.thrExpo8 );       //
[0;10000]
    lookupThrottleRC[i] = conf.minthrottle + (uint32_t)((uint16_t)(MAXTHROTTLE-
conf.minthrottle))* lookupThrottleRC[i]/10000;  // [0;10000] ->
[conf.minthrottle;MAXTHROTTLE]
  }
  #if defined(POWERMETER)
    pAlarm = (uint32_t) conf.powerTrigger1 * (uint32_t) PLEVELSCALE * (uint32_t)
PLEVELDIV; // need to cast before multiplying
  #endif
  #if GPS
    GPS_set_pids();    // at this time we don't have info about GPS init done
    recallGPSconf();   // Load gps parameters
  #endif
  #if defined(ARMEDTIMEWARNING)
    ArmedTimeWarningMicroSeconds = (conf.armedtimewarning *1000000);
  #endif
  return true;    // setting is OK
}

void writeGlobalSet(uint8_t b) {
  global_conf.checksum = calculate_sum((uint8_t*)&global_conf,
sizeof(global_conf));
  eeprom_write_block((const void*)&global_conf, (void*)0, sizeof(global_conf));
  if (b == 1) blinkLED(15,20,1);
  SET_ALARM_BUZZER(ALRM_FAC_CONFIRM, ALRM_LVL_CONFIRM_1);

}

void writeParams(uint8_t b) {
```

```
  #ifdef MULTIPLE_CONFIGURATION_PROFILES
    if(global_conf.currentSet>2) global_conf.currentSet=0;
  #else
    global_conf.currentSet=0;
  #endif
  conf.checksum = calculate_sum((uint8_t*)&conf, sizeof(conf));
  eeprom_write_block((const void*)&conf, (void*)(global_conf.currentSet *
sizeof(conf) + sizeof(global_conf)), sizeof(conf));

#if GPS
  writeGPSconf();  //Write GPS parameters
  recallGPSconf(); //Read it to ensure correct eeprom content
#endif

  readEEPROM();
  if (b == 1) blinkLED(15,20,1);
  SET_ALARM_BUZZER(ALRM_FAC_CONFIRM, ALRM_LVL_CONFIRM_1);
}

void update_constants() {
  #if defined(GYRO_SMOOTHING)
    {
      uint8_t s[3] = GYRO_SMOOTHING;
      for(uint8_t i=0;i<3;i++) conf.Smoothing[i] = s[i];
    }
  #endif
  #if defined (FAILSAFE)
    conf.failsafe_throttle = FAILSAFE_THROTTLE;
  #endif
  #ifdef VBAT
    conf.vbatscale = VBATSCALE;
    conf.vbatlevel_warn1 = VBATLEVEL_WARN1;
    conf.vbatlevel_warn2 = VBATLEVEL_WARN2;
    conf.vbatlevel_crit = VBATLEVEL_CRIT;
  #endif
  #ifdef POWERMETER
    conf.pint2ma = PINT2mA;
  #endif
  #ifdef POWERMETER_HARD
    conf.psensornull = PSENSORNULL;
  #endif
  #ifdef MMGYRO
    conf.mmgyro = MMGYRO;
  #endif
  #if defined(ARMEDTIMEWARNING)
    conf.armedtimewarning = ARMEDTIMEWARNING;
  #endif
  conf.minthrottle = MINTHROTTLE;
  #if MAG
    conf.mag_declination = (int16_t)(MAG_DECLINATION * 10);
```

```
  #endif
  #ifdef GOVERNOR_P
    conf.governorP = GOVERNOR_P;
    conf.governorD = GOVERNOR_D;
  #endif
  #ifdef YAW_COLL_PRECOMP
    conf.yawCollPrecomp = YAW_COLL_PRECOMP;
    conf.yawCollPrecompDeadband = YAW_COLL_PRECOMP_DEADBAND;
  #endif
  #if defined(MY_PRIVATE_DEFAULTS)
    #include MY_PRIVATE_DEFAULTS
  #endif

#if GPS
  loadGPSdefaults();
#endif

  writeParams(0); // this will also (p)reset checkNewConf with the current version
number again.
}

void LoadDefaults() {
  uint8_t i;
  #ifdef SUPPRESS_DEFAULTS_FROM_GUI
    // do nothing
  #elif defined(MY_PRIVATE_DEFAULTS)
    // #include MY_PRIVATE_DEFAULTS
    // do that at the last possible moment, so we can override virtually all defaults and
constants
  #else
    #if PID_CONTROLLER == 1
      conf.pid[ROLL].P8     = 33;  conf.pid[ROLL].I8    = 30; conf.pid[ROLL].D8     =
23;
      conf.pid[PITCH].P8    = 33; conf.pid[PITCH].I8    = 30; conf.pid[PITCH].D8    =
23;
      conf.pid[PIDLEVEL].P8 = 90; conf.pid[PIDLEVEL].I8 = 10;
conf.pid[PIDLEVEL].D8 = 100;
    #elif PID_CONTROLLER == 2
      conf.pid[ROLL].P8     = 28;  conf.pid[ROLL].I8    = 10; conf.pid[ROLL].D8     =
7;
      conf.pid[PITCH].P8    = 28; conf.pid[PITCH].I8    = 10; conf.pid[PITCH].D8    =
7;
      conf.pid[PIDLEVEL].P8 = 30; conf.pid[PIDLEVEL].I8 = 32;
conf.pid[PIDLEVEL].D8 = 0;
    #endif
    conf.pid[YAW].P8      = 68;  conf.pid[YAW].I8     = 45;  conf.pid[YAW].D8     = 0;
    conf.pid[PIDALT].P8   = 64; conf.pid[PIDALT].I8   = 25; conf.pid[PIDALT].D8
= 24;

    conf.pid[PIDPOS].P8  = POSHOLD_P * 100;     conf.pid[PIDPOS].I8    =
```

```
POSHOLD_I * 100;       conf.pid[PIDPOS].D8    = 0;
   conf.pid[PIDPOSR].P8 = POSHOLD_RATE_P * 10; conf.pid[PIDPOSR].I8   =
POSHOLD_RATE_I * 100;  conf.pid[PIDPOSR].D8   = POSHOLD_RATE_D *
1000;
   conf.pid[PIDNAVR].P8 = NAV_P * 10;         conf.pid[PIDNAVR].I8   = NAV_I *
100;          conf.pid[PIDNAVR].D8   = NAV_D * 1000;

   conf.pid[PIDMAG].P8   = 40;

   conf.pid[PIDVEL].P8 = 0;      conf.pid[PIDVEL].I8 = 0;    conf.pid[PIDVEL].D8 =
0;

   conf.rcRate8 = 90; conf.rcExpo8 = 65;
   conf.rollPitchRate = 0;
   conf.yawRate = 0;
   conf.dynThrPID = 0;
   conf.thrMid8 = 50; conf.thrExpo8 = 0;
   for(i=0;i<CHECKBOXITEMS;i++) {conf.activate[i] = 0;}
   conf.angleTrim[0] = 0; conf.angleTrim[1] = 0;
   conf.powerTrigger1 = 0;
 #endif // SUPPRESS_DEFAULTS_FROM_GUI
 #if defined(SERVO)
   static int8_t sr[8] = SERVO_RATES;
   #ifdef SERVO_MIN
     static int16_t smin[8] = SERVO_MIN;
     static int16_t smax[8] = SERVO_MAX;
     static int16_t smid[8] = SERVO_MID;
   #endif
   for(i=0;i<8;i++) {
     #ifdef SERVO_MIN
       conf.servoConf[i].min = smin[i];
       conf.servoConf[i].max = smax[i];
       conf.servoConf[i].middle = smid[i];
     #else
       conf.servoConf[i].min = 1020;
       conf.servoConf[i].max = 2000;
       conf.servoConf[i].middle = 1500;
     #endif
     conf.servoConf[i].rate = sr[i];
   }
 #else                 //if no servo defined then zero out the config variables to prevent
passing false data to the gui.
//    for(i=0;i<8;i++) {
//       conf.servoConf[i].min = 0;
//       conf.servoConf[i].max = 0;
//       conf.servoConf[i].middle = 0;
//       conf.servoConf[i].rate = 0;
//     }
 #endif
 #ifdef FIXEDWING
```

```
    conf.dynThrPID = 50;
    conf.rcExpo8   =  0;
  #endif
  update_constants(); // this will also write to eeprom
}

#ifdef LOG_PERMANENT
void readPLog(void) {
  eeprom_read_block((void*)&plog, (void*)(E2END - 4 - sizeof(plog)), sizeof(plog));
  if(calculate_sum((uint8_t*)&plog, sizeof(plog)) != plog.checksum) {
    blinkLED(9,100,3);
    SET_ALARM_BUZZER(ALRM_FAC_CONFIRM,
ALRM_LVL_CONFIRM_ELSE);
    // force load defaults
    plog.arm = plog.disarm = plog.start = plog.failsafe = plog.i2c = 0;
    plog.running = 1;
    plog.lifetime = plog.armed_time = 0;
    writePLog();
  }
}
void writePLog(void) {
  plog.checksum = calculate_sum((uint8_t*)&plog, sizeof(plog));
  eeprom_write_block((const void*)&plog, (void*)(E2END - 4 - sizeof(plog)),
sizeof(plog));
}
#endif

#if GPS

//Define variables for calculations of EEPROM positions
#ifdef MULTIPLE_CONFIGURATION_PROFILES
    #define PROFILES 3
#else
    #define PROFILES 1
#endif
#ifdef LOG_PERMANENT
    #define PLOG_SIZE sizeof(plog)
#else
    #define PLOG_SIZE 0
#endif

//Store gps_config

void writeGPSconf(void) {
  GPS_conf.checksum = calculate_sum((uint8_t*)&GPS_conf, sizeof(GPS_conf));
  eeprom_write_block( (void*)&GPS_conf, (void*) (PROFILES * sizeof(conf) +
sizeof(global_conf)), sizeof(GPS_conf) );
}

//Recall gps_configuration
```

```
bool recallGPSconf(void) {
  eeprom_read_block((void*)&GPS_conf, (void*)(PROFILES * sizeof(conf) +
sizeof(global_conf)), sizeof(GPS_conf));
  if(calculate_sum((uint8_t*)&GPS_conf, sizeof(GPS_conf)) != GPS_conf.checksum)
{
    loadGPSdefaults();
    return false;
  }
  return true;
}

//Load gps_config_defaults and writes back to EEPROM just to make it sure
void loadGPSdefaults(void) {
  //zero out the conf struct
  uint8_t *ptr = (uint8_t *) &GPS_conf;
  for (int i=0;i<sizeof(GPS_conf);i++) *ptr++ = 0;

#if defined(GPS_FILTERING)
  GPS_conf.filtering = 1;
#endif
#if defined (GPS_LEAD_FILTER)
  GPS_conf.lead_filter = 1;
#endif
#if defined (DONT_RESET_HOME_AT_ARM)
  GPS_conf.dont_reset_home_at_arm = 1;
#endif
  GPS_conf.nav_controls_heading = NAV_CONTROLS_HEADING;
  GPS_conf.nav_tail_first      = NAV_TAIL_FIRST;
  GPS_conf.nav_rth_takeoff_heading = NAV_SET_TAKEOFF_HEADING;
  GPS_conf.slow_nav              = NAV_SLOW_NAV;
  GPS_conf.wait_for_rth_alt       = WAIT_FOR_RTH_ALT;

  GPS_conf.ignore_throttle        = IGNORE_THROTTLE;
  GPS_conf.takeover_baro          = NAV_TAKEOVER_BARO;

  GPS_conf.wp_radius              = GPS_WP_RADIUS;
  GPS_conf.safe_wp_distance       = SAFE_WP_DISTANCE;
  GPS_conf.nav_max_altitude       = MAX_NAV_ALTITUDE;
  GPS_conf.nav_speed_max          = NAV_SPEED_MAX;
  GPS_conf.nav_speed_min          = NAV_SPEED_MIN;
  GPS_conf.crosstrack_gain        = CROSSTRACK_GAIN * 100;
  GPS_conf.nav_bank_max           = NAV_BANK_MAX;
  GPS_conf.rth_altitude           = RTH_ALTITUDE;
  GPS_conf.fence                  = FENCE_DISTANCE;
  GPS_conf.land_speed             = LAND_SPEED;
  GPS_conf.max_wp_number          = getMaxWPNumber();
  writeGPSconf();
}
```

```
//Stores the WP data in the wp struct in the EEPROM
void storeWP() {
  if (mission_step.number >254) return;
  mission_step.checksum = calculate_sum((uint8_t*)&mission_step,
sizeof(mission_step));
  eeprom_write_block((void*)&mission_step, (void*)(PROFILES * sizeof(conf) +
sizeof(global_conf) + sizeof(GPS_conf)
+(sizeof(mission_step)*mission_step.number)),sizeof(mission_step));
}

// Read the given number of WP from the eeprom, supposedly we can use this during
flight.
// Returns true when reading is successfull and returns false if there were some error
(for example checksum)
bool recallWP(uint8_t wp_number) {
  if (wp_number > 254) return false;
  eeprom_read_block((void*)&mission_step, (void*)(PROFILES * sizeof(conf) +
sizeof(global_conf)+sizeof(GPS_conf)+(sizeof(mission_step)*wp_number)),
sizeof(mission_step));
  if(calculate_sum((uint8_t*)&mission_step, sizeof(mission_step)) !=
mission_step.checksum) return false;
  return true;
}

// Returns the maximum WP number that can be stored in the EEPROM, calculated
from conf and plog sizes, and the eeprom size
uint8_t getMaxWPNumber() {
  uint16_t first_avail = PROFILES*sizeof(conf) +
sizeof(global_conf)+sizeof(GPS_conf)+ 1; //Add one byte for addtnl separation
  uint16_t last_avail  = E2END - PLOG_SIZE - 4; //keep the last 4 bytes intakt
  uint16_t wp_num = (last_avail-first_avail)/sizeof(mission_step);
  if (wp_num>254) wp_num = 254;
  return wp_num;
}
#endif
```

καρτέλα EEPROM.h

```
#ifndef EEPROM_H_
#define EEPROM_H_

void readGlobalSet();
bool readEEPROM();
void update_constants();
void writeGlobalSet(uint8_t b);
void writeParams(uint8_t b);
void LoadDefaults();
void readPLog(void);
void writePLog(void);

#if defined(GPS)
//EEPROM functions for storing and restoring waypoints
```

```
void storeWP(void);     // Stores the WP data in the wp struct in the EEPROM
bool recallWP(uint8_t); // Read the given number of WP from the eeprom, supposedly
we can use this during flight.
                        // Returns true when reading is successfull and returns false if there
were some error (for example checksum)
uint8_t getMaxWPNumber(void);  // Returns the maximum WP number that can be
stored in the EEPROM, calculated from conf and plog sizes, and the eeprom size

void loadGPSdefaults(void);
void writeGPSconf(void) ;
bool recallGPSconf(void);
#endif

#endif /* EEPROM_H_ */
```

καρτέλα GPS.cpp

```
#include "Arduino.h"
#include "config.h"
#include "def.h"
#include "types.h"
#include "GPS.h"
#include "Serial.h"
#include "Sensors.h"
#include "MultiWii.h"
#include "EEPROM.h"
#include <math.h>

#if GPS

//Function prototypes for other GPS functions
//These perhaps could go to the gps.h file, however these are local to the gps.cpp
static void GPS_bearing(int32_t* lat1, int32_t* lon1, int32_t* lat2, int32_t* lon2,
int32_t* bearing);
static void GPS_distance_cm(int32_t* lat1, int32_t* lon1, int32_t* lat2, int32_t*
lon2,uint32_t* dist);
static void GPS_calc_velocity(void);
static void GPS_calc_location_error( int32_t* target_lat, int32_t* target_lng, int32_t*
gps_lat, int32_t* gps_lng );
static void GPS_calc_poshold(void);
static uint16_t GPS_calc_desired_speed(uint16_t max_speed, bool _slow);
static void GPS_calc_nav_rate(uint16_t max_speed);
int32_t wrap_18000(int32_t ang);
static bool check_missed_wp(void);
void GPS_calc_longitude_scaling(int32_t lat);
static void GPS_update_crosstrack(void);
int32_t wrap_36000(int32_t ang);


// Leadig filter - TODO: rewrite to normal C instead of C++
```

```
// Set up gps lag
#if defined(UBLOX) || defined (MTK_BINARY19)
#define GPS_LAG 0.5f                    //UBLOX GPS has a smaller lag than MTK
and other
#else
#define GPS_LAG 1.0f                    //We assumes that MTK GPS has a 1 sec lag
#endif

static int32_t  GPS_coord_lead[2];            // Lead filtered gps coordinates

class LeadFilter {
public:
   LeadFilter() :
      _last_velocity(0) {
   }

   // setup min and max radio values in CLI
   int32_t        get_position(int32_t pos, int16_t vel, float lag_in_seconds = 1.0);
   void           clear() { _last_velocity = 0; }

private:
   int16_t        _last_velocity;

};

int32_t LeadFilter::get_position(int32_t pos, int16_t vel, float lag_in_seconds)
{
   int16_t accel_contribution = (vel - _last_velocity) * lag_in_seconds *
lag_in_seconds;
   int16_t vel_contribution = vel * lag_in_seconds;

   // store velocity for next iteration
   _last_velocity = vel;

   return pos + vel_contribution + accel_contribution;
}


LeadFilter xLeadFilter;      // Long GPS lag filter
LeadFilter yLeadFilter;      // Lat  GPS lag filter

typedef struct PID_PARAM_ {
  float kP;
  float kI;
  float kD;
  float Imax;
  } PID_PARAM;

PID_PARAM posholdPID_PARAM;
PID_PARAM poshold_ratePID_PARAM;
```

```
PID_PARAM navPID_PARAM;

typedef struct PID_ {
  float   integrator; // integrator value
  int32_t last_input; // last input for derivative
  float   lastderivative; // last derivative for low-pass filter
  float   output;
  float   derivative;
} PID;
PID posholdPID[2];
PID poshold_ratePID[2];
PID navPID[2];

int32_t get_P(int32_t error, struct PID_PARAM_* pid) {
  return (float)error * pid->kP;
}

int32_t get_I(int32_t error, float* dt, struct PID_* pid, struct PID_PARAM_*
pid_param) {
  pid->integrator += ((float)error * pid_param->kI) * *dt;
  pid->integrator = constrain(pid->integrator,-pid_param->Imax,pid_param->Imax);
  return pid->integrator;
}

int32_t get_D(int32_t input, float* dt, struct PID_* pid, struct PID_PARAM_*
pid_param) { // dt in milliseconds
  pid->derivative = (input - pid->last_input) / *dt;

  /// Low pass filter cut frequency for derivative calculation.
  float filter = 7.9577e-3; // Set to  "1 / ( 2 * PI * f_cut )";
  // Examples for _filter:
  // f_cut = 10 Hz -> _filter = 15.9155e-3
  // f_cut = 15 Hz -> _filter = 10.6103e-3
  // f_cut = 20 Hz -> _filter =  7.9577e-3
  // f_cut = 25 Hz -> _filter =  6.3662e-3
  // f_cut = 30 Hz -> _filter =  5.3052e-3

  // discrete low pass filter, cuts out the
  // high frequency noise that can drive the controller crazy
  pid->derivative = pid->lastderivative + (*dt / ( filter + *dt)) * (pid->derivative - pid-
>lastderivative);
  // update state
  pid->last_input = input;
  pid->lastderivative    = pid->derivative;
  // add in derivative component
  return pid_param->kD * pid->derivative;
}

void reset_PID(struct PID_* pid) {
  pid->integrator = 0;
```

```
  pid->last_input = 0;
  pid->lastderivative = 0;
}

#define _X 1
#define _Y 0

#define RADX100                 0.000174532925

uint8_t land_detect;              //Detect land (extern)
static uint32_t land_settle_timer;
uint8_t GPS_Frame;           // a valid GPS_Frame was detected, and data is ready for
nav computation

static float  dTnav;           // Delta Time in milliseconds for navigation computations,
updated with every good GPS read
static int16_t actual_speed[2] = {0,0};
static float GPS_scaleLonDown; // this is used to offset the shrinking longitude as we
go towards the poles

// The difference between the desired rate of travel and the actual rate of travel
// updated after GPS read - 5-10hz
static int16_t rate_error[2];
static int32_t error[2];

static int32_t GPS_WP[2];   //Currently used WP
static int32_t GPS_FROM[2]; //the pervious waypoint for precise track following
int32_t target_bearing;     // This is the angle from the copter to the "next_WP"
location in degrees * 100
static int32_t original_target_bearing;  // deg * 100, The original angle to the next_WP
when the next_WP was set, Also used to check when we pass a WP
static int16_t crosstrack_error;     // The amount of angle correction applied to
target_bearing to bring the copter back on its optimum path
uint32_t wp_distance;               // distance between plane and next_WP in cm
static uint16_t waypoint_speed_gov;  // used for slow speed wind up when start
navigation;


////////////////////////////////////////////////////////////////////////////////
// moving average filter variables
//

#define GPS_FILTER_VECTOR_LENGTH 5

static uint8_t GPS_filter_index = 0;
static int32_t GPS_filter[2][GPS_FILTER_VECTOR_LENGTH];
static int32_t GPS_filter_sum[2];
static int32_t GPS_read[2];
static int32_t GPS_filtered[2];
static int32_t GPS_degree[2];    //the lat lon degree without any decimals (lat/10 000
```

```
000)
static uint16_t fraction3[2];

static int16_t nav_takeoff_bearing;  // saves the bearing at takeof (1deg = 1) used to
rotate to takeoff direction when arrives at home



//Main navigation processor and state engine
// TODO: add proceesing states to ease processing burden
uint8_t GPS_Compute(void) {
  unsigned char axis;
  uint32_t dist;        //temp variable to store dist to copter
  int32_t  dir;         //temp variable to store dir to copter
  static uint32_t nav_loopTimer;

  //check that we have a valid frame, if not then return immediatly
  if (GPS_Frame == 0) return 0; else GPS_Frame = 0;

  //check home position and set it if it was not set
  if (f.GPS_FIX && GPS_numSat >= 5) {
    #if !defined(DONT_RESET_HOME_AT_ARM)
      if (!f.ARMED) {f.GPS_FIX_HOME = 0;}
    #endif
    if (!f.GPS_FIX_HOME && f.ARMED) {
      GPS_reset_home_position();
    }
    //Apply moving average filter to GPS data
    if (GPS_conf.filtering) {
      GPS_filter_index = (GPS_filter_index+1) % GPS_FILTER_VECTOR_LENGTH;
      for (axis = 0; axis< 2; axis++) {
        GPS_read[axis] = GPS_coord[axis]; //latest unfiltered data is in GPS_latitude
and GPS_longitude
        GPS_degree[axis] = GPS_read[axis] / 10000000;  // get the degree to assure the
sum fits to the int32_t

        // How close we are to a degree line ? its the first three digits from the fractions
of degree
        // later we use it to Check if we are close to a degree line, if yes, disable
averaging,
        fraction3[axis] = (GPS_read[axis]- GPS_degree[axis]*10000000) / 10000;

        GPS_filter_sum[axis] -= GPS_filter[axis][GPS_filter_index];
        GPS_filter[axis][GPS_filter_index] = GPS_read[axis] -
(GPS_degree[axis]*10000000);
        GPS_filter_sum[axis] += GPS_filter[axis][GPS_filter_index];
        GPS_filtered[axis] = GPS_filter_sum[axis] /
GPS_FILTER_VECTOR_LENGTH + (GPS_degree[axis]*10000000);
        if ( NAV_state == NAV_STATE_HOLD_INFINIT || NAV_state ==
NAV_STATE_HOLD_TIMED) {      //we use gps averaging only in poshold mode...
```

```
        if ( fraction3[axis]>1 && fraction3[axis]<999 ) GPS_coord[axis] =
GPS_filtered[axis];
       }
     }
   }

   //dTnav calculation
   //Time for calculating x,y speed and navigation pids
   dTnav = (float)(millis() - nav_loopTimer)/ 1000.0;
   nav_loopTimer = millis();

   // prevent runup from bad GPS
   dTnav = min(dTnav, 1.0);

   //calculate distance and bearings for gui and other stuff continously - From home to
copter

GPS_bearing(&GPS_coord[LAT],&GPS_coord[LON],&GPS_home[LAT],&GPS_ho
me[LON],&dir);

GPS_distance_cm(&GPS_coord[LAT],&GPS_coord[LON],&GPS_home[LAT],&GP
S_home[LON],&dist);
   GPS_distanceToHome = dist/100;
   GPS_directionToHome = dir/100;

   if (!f.GPS_FIX_HOME) {     //If we don't have home set, do not display anything
     GPS_distanceToHome = 0;
     GPS_directionToHome = 0;
   }

   //Check fence setting and execute RTH if neccessary
   //TODO: autolanding
   if ((GPS_conf.fence > 0) && (GPS_conf.fence < GPS_distanceToHome) &&
(f.GPS_mode != GPS_MODE_RTH) ) {
     init_RTH();
   }

   //calculate the current velocity based on gps coordinates continously to get a valid
speed at the moment when we start navigating
   GPS_calc_velocity();

   //Navigation state engine
   if (f.GPS_mode != GPS_MODE_NONE) {   //ok we are navigating ###0002
     //do gps nav calculations here, these are common for nav and poshold

GPS_bearing(&GPS_coord[LAT],&GPS_coord[LON],&GPS_WP[LAT],&GPS_WP[
LON],&target_bearing);
     if (GPS_conf.lead_filter) {

GPS_distance_cm(&GPS_coord_lead[LAT],&GPS_coord_lead[LON],&GPS_WP[L
```

```
AT],&GPS_WP[LON],&wp_distance);

GPS_calc_location_error(&GPS_WP[LAT],&GPS_WP[LON],&GPS_coord_lead[LA
T],&GPS_coord_lead[LON]);
    } else {

GPS_distance_cm(&GPS_coord[LAT],&GPS_coord[LON],&GPS_WP[LAT],&GPS
_WP[LON],&wp_distance);

GPS_calc_location_error(&GPS_WP[LAT],&GPS_WP[LON],&GPS_coord[LAT],&
GPS_coord[LON]);
    }

    // Adjust altitude
    // if we are holding position and reached target altitude, then ignore altitude nav,
and let the user trim alt
    if ( !((NAV_state == NAV_STATE_HOLD_INFINIT) && (alt_change_flag ==
REACHED_ALT))) {
     if (!f.LAND_IN_PROGRESS) {
      alt_to_hold = get_new_altitude();
      AltHold = alt_to_hold;
     }
    }

    int16_t speed = 0;                //Desired navigation speed

    switch(NAV_state)                 //Navigation state machine
     {
     case NAV_STATE_NONE:             //Just for clarity, do nothing when nav_state
is none
      break;

     case NAV_STATE_LAND_START:
      GPS_calc_poshold();             //Land in position hold
      land_settle_timer = millis();
      NAV_state = NAV_STATE_LAND_SETTLE;
      break;

     case NAV_STATE_LAND_SETTLE:
      GPS_calc_poshold();
      if (millis()-land_settle_timer > 5000)
       NAV_state = NAV_STATE_LAND_START_DESCENT;
      break;

     case NAV_STATE_LAND_START_DESCENT:
      GPS_calc_poshold();             //Land in position hold
      f.THROTTLE_IGNORED = 1;         //Ignore Throtte stick input
      f.GPS_BARO_MODE    = 1;         //Take control of BARO mode
      land_detect = 0;                //Reset land detector
      f.LAND_COMPLETED = 0;
```

```
        f.LAND_IN_PROGRESS = 1;           // Flag land process
        NAV_state = NAV_STATE_LAND_IN_PROGRESS;
        break;

      case NAV_STATE_LAND_IN_PROGRESS:
        GPS_calc_poshold();
        check_land();  //Call land detector
        if (f.LAND_COMPLETED) {
          nav_timer_stop = millis() + 5000;
          NAV_state = NAV_STATE_LANDED;
        }
        break;

      case NAV_STATE_LANDED:
        // Disarm if THROTTLE stick is at minimum or 5sec past after land detected
        if (rcData[THROTTLE]<MINCHECK || nav_timer_stop <= millis()) {
//Throttle at minimum or 5sec passed.
          go_disarm();
          f.OK_TO_ARM = 0;              //Prevent rearming
          NAV_state = NAV_STATE_NONE;     //Disable position holding.... prevent
flippover
          f.GPS_BARO_MODE = 0;
          f.LAND_COMPLETED = 0;
          f.LAND_IN_PROGRESS = 0;
          land_detect = 0;
          f.THROTTLE_IGNORED = 0;
          GPS_reset_nav();
        }
        break;

      case NAV_STATE_HOLD_INFINIT:        //Constant position hold, no timer.
Only an rcOption change can exit from this
        GPS_calc_poshold();
        break;

      case NAV_STATE_HOLD_TIMED:
        if (nav_timer_stop == 0) {                    //We are start a timed poshold
          nav_timer_stop = millis() + 1000*nav_hold_time;  //Set when we will
continue
        } else if (nav_timer_stop <= millis()) {          //did we reach our time limit ?
          if (mission_step.flag != MISSION_FLAG_END) {
            NAV_state = NAV_STATE_PROCESS_NEXT;           //if yes then process
next mission step
          }
          NAV_error = NAV_ERROR_TIMEWAIT;
        }
        GPS_calc_poshold();                           //BTW hold position till next
command
        break;
```

```
      case NAV_STATE_RTH_START:
        if ((alt_change_flag == REACHED_ALT) || (!GPS_conf.wait_for_rth_alt)) {
//Wait until we reach RTH altitude
          GPS_set_next_wp(&GPS_home[LAT],&GPS_home[LON],
&GPS_coord[LAT], &GPS_coord[LON]); //If we reached then change mode and start
RTH
          NAV_state = NAV_STATE_RTH_ENROUTE;
          NAV_error = NAV_ERROR_NONE;
        } else {
          GPS_calc_poshold();                                    //hold position till
we reach RTH alt
          NAV_error = NAV_ERROR_WAIT_FOR_RTH_ALT;
        }
        break;

      case NAV_STATE_RTH_ENROUTE:                                //Doing
RTH navigation
        speed = GPS_calc_desired_speed(GPS_conf.nav_speed_max,
GPS_conf.slow_nav);
        GPS_calc_nav_rate(speed);
        GPS_adjust_heading();
        if ((wp_distance <= GPS_conf.wp_radius) || check_missed_wp()) {          //if
yes switch to poshold mode
          if (mission_step.parameter1 == 0) NAV_state =
NAV_STATE_HOLD_INFINIT;
          else NAV_state = NAV_STATE_LAND_START;                 // if
parameter 1 in RTH step is non 0 then land at home
          if (GPS_conf.nav_rth_takeoff_heading) { magHold = nav_takeoff_bearing; }
        }
        break;

      case NAV_STATE_WP_ENROUTE:
        speed = GPS_calc_desired_speed(GPS_conf.nav_speed_max,
GPS_conf.slow_nav);
        GPS_calc_nav_rate(speed);
        GPS_adjust_heading();

        if ((wp_distance <= GPS_conf.wp_radius) || check_missed_wp()) {
//This decides what happen when we reached the WP coordinates
          if (mission_step.action == MISSION_LAND) {
//Autoland
            NAV_state = NAV_STATE_LAND_START;                    //Start
landing
            set_new_altitude(alt.EstAlt);                        //Stop any altitude
changes
          } else if (mission_step.flag == MISSION_FLAG_END) {          //If
this was the last mission step (flag set by the mission planner), then switch to poshold
            NAV_state = NAV_STATE_HOLD_INFINIT;
            NAV_error = NAV_ERROR_FINISH;
          } else if (mission_step.action == MISSION_HOLD_UNLIM) {
```

```
//If mission_step was POSHOLD_UNLIM and we reached the position then switch to
poshold unlimited
            NAV_state = NAV_STATE_HOLD_INFINIT;
            NAV_error = NAV_ERROR_FINISH;
           } else if (mission_step.action == MISSION_HOLD_TIME) {                  //If
mission_step was a timed poshold then initiate timed poshold
            nav_hold_time = mission_step.parameter1;
            nav_timer_stop = 0;                                          //This indicates that
we are starting a timed poshold
            NAV_state = NAV_STATE_HOLD_TIMED;
           } else {
            NAV_state = NAV_STATE_PROCESS_NEXT;
//Otherwise process next step
           }
          }
          break;

        case NAV_STATE_DO_JUMP:
          if (jump_times < 0) {                          //Jump unconditionally (supposed to
be -1) -10 should not be here
           next_step = mission_step.parameter1;
           NAV_state = NAV_STATE_PROCESS_NEXT;
          }
          if (jump_times == 0) {
           jump_times = -10;                             //reset jump counter
           if (mission_step.flag == MISSION_FLAG_END) {        //If this was the last
mission step (flag set by the mission planner), then switch to poshold
            NAV_state = NAV_STATE_HOLD_INFINIT;
            NAV_error = NAV_ERROR_FINISH;
           } else
            NAV_state = NAV_STATE_PROCESS_NEXT;
          }

          if (jump_times > 0) {                          //if zero not reached do a jump
           next_step = mission_step.parameter1;
           NAV_state = NAV_STATE_PROCESS_NEXT;
           jump_times--;
          }
          break;

        case NAV_STATE_PROCESS_NEXT:                     //Processing next
mission step
          NAV_error = NAV_ERROR_NONE;
          if (!recallWP(next_step)) {
           abort_mission(NAV_ERROR_WP_CRC);
          } else {
           switch(mission_step.action)
            {
            //Waypoiny and hold commands all starts with an enroute status it includes
the LAND command too
```

```
            case MISSION_WAYPOINT:
            case MISSION_HOLD_TIME:
            case MISSION_HOLD_UNLIM:
            case MISSION_LAND:
              set_new_altitude(mission_step.altitude);
              GPS_set_next_wp(&mission_step.pos[LAT], &mission_step.pos[LON],
&GPS_prev[LAT], &GPS_prev[LON]);
              if ((wp_distance/100) >= GPS_conf.safe_wp_distance)
abort_mission(NAV_ERROR_TOOFAR);
              else NAV_state = NAV_STATE_WP_ENROUTE;
              GPS_prev[LAT] = mission_step.pos[LAT];  //Save wp coordinates for
precise route calc
              GPS_prev[LON] = mission_step.pos[LON];
              break;
            case MISSION_RTH:
              f.GPS_head_set = 0;
              if (GPS_conf.rth_altitude == 0 && mission_step.altitude == 0) //if config
and mission_step alt is zero
                set_new_altitude(alt.EstAlt);     // RTH returns at the actual altitude
              else {
                uint32_t rth_alt;
                if (mission_step.altitude == 0) rth_alt = GPS_conf.rth_altitude * 100;
//altitude in mission step has priority
                else rth_alt = mission_step.altitude;

                if (alt.EstAlt < rth_alt) set_new_altitude(rth_alt);                  //BUt only if
we are below it.
                else set_new_altitude(alt.EstAlt);
              }
              NAV_state = NAV_STATE_RTH_START;
              break;
            case MISSION_JUMP:
              if (jump_times == -10) jump_times = mission_step.parameter2;
              if (mission_step.parameter1 > 0 && mission_step.parameter1 <
mission_step.number)
                NAV_state = NAV_STATE_DO_JUMP;
              else //Error situation, invalid jump target
                abort_mission(NAV_ERROR_INVALID_JUMP);
              break;
            case MISSION_SET_HEADING:
              GPS_poi[LAT] = 0; GPS_poi[LON] = 0; // zeroing this out clears the
possible pervious set_poi
              if (mission_step.parameter1 < 0) f.GPS_head_set = 0;
              else {
                f.GPS_head_set = 1;
                GPS_directionToPoi = mission_step.parameter1;
              }
              break;
            case MISSION_SET_POI:
              GPS_poi[LAT] = mission_step.pos[LAT];
```

```
            GPS_poi[LON] = mission_step.pos[LON];
            f.GPS_head_set = 1;
            break;
           default:                    //if we got an unknown action code abort
mission and hold position
            abort_mission(NAV_ERROR_INVALID_DATA);
            break;
           }
          next_step++; //Prepare for the next step
         }
         break;
       } // switch end
    } //end of gps calcs ###0002
  }
  return 1;
} // End of GPS_compute

// Abort current mission with the given error code (switch to poshold_infinit)
void abort_mission(unsigned char error_code) {
  GPS_set_next_wp(&GPS_coord[LAT], &GPS_coord[LON],&GPS_coord[LAT],
&GPS_coord[LON]);
  NAV_error = error_code;
  NAV_state = NAV_STATE_HOLD_INFINIT;
}

//Adjusting heading according to settings - MAG mode must be enabled
void GPS_adjust_heading() {
  //TODO: Add slow windup for large heading change
  //This controls the heading
  if (f.GPS_head_set) { // We have seen a SET_POI or a SET_HEADING command
    if (GPS_poi[LAT] == 0)
      magHold = wrap_18000((GPS_directionToPoi*100))/100;
    else {

GPS_bearing(&GPS_coord[LAT],&GPS_coord[LON],&GPS_poi[LAT],&GPS_poi[
LON],&GPS_directionToPoi);

GPS_distance_cm(&GPS_coord[LAT],&GPS_coord[LON],&GPS_poi[LAT],&GPS_
poi[LON],&wp_distance);
      magHold = GPS_directionToPoi /100;
    }
  } else {                          // heading controlled by the standard defines
    if (GPS_conf.nav_controls_heading) {
      if (GPS_conf.nav_tail_first) {
        magHold = wrap_18000(target_bearing-18000)/100;
      } else {
        magHold = wrap_18000(target_bearing)/100;
      }
    }
  }
```

```
}

#define LAND_DETECT_THRESHOLD 40      //Counts of land situation
#define BAROPIDMIN          -180     //BaroPID reach this if we landed.....

//Check if we landed or not
void check_land() {
  // detect whether we have landed by watching for low climb rate and throttle control
  if ( (abs(alt.vario) < 20) && (BaroPID < BAROPIDMIN)) {
    if (!f.LAND_COMPLETED) {
      if( land_detect < LAND_DETECT_THRESHOLD) {
        land_detect++;
      } else {
        f.LAND_COMPLETED = 1;
        land_detect = 0;
      }
    }
  } else {
    // we've detected movement up or down so reset land_detector
    land_detect = 0;
    if(f.LAND_COMPLETED) {
      f.LAND_COMPLETED = 0;
    }
  }
}

int32_t get_altitude_error() {
  return alt_to_hold - alt.EstAlt;
}

void clear_new_altitude() {
  alt_change_flag = REACHED_ALT;
}

void force_new_altitude(int32_t _new_alt) {
  alt_to_hold     = _new_alt;
  target_altitude = _new_alt;
  alt_change_flag = REACHED_ALT;
}

void set_new_altitude(int32_t _new_alt) {
  //Limit maximum altitude command
  if(_new_alt > GPS_conf.nav_max_altitude*100) _new_alt =
GPS_conf.nav_max_altitude * 100;
  if(_new_alt == alt.EstAlt){
    force_new_altitude(_new_alt);
    return;
  }
  // We start at the current location altitude and gradually change alt
  alt_to_hold = alt.EstAlt;
```

```
  // for calculating the delta time
  alt_change_timer = millis();
  // save the target altitude
  target_altitude = _new_alt;
  // reset our altitude integrator
  alt_change = 0;
  // save the original altitude
  original_altitude = alt.EstAlt;
  // to decide if we have reached the target altitude
  if(target_altitude > original_altitude){
    // we are below, going up
    alt_change_flag = ASCENDING;
  } else if(target_altitude < original_altitude){
    // we are above, going down
    alt_change_flag = DESCENDING;
  } else {
    // No Change
    alt_change_flag = REACHED_ALT;
  }
}

int32_t get_new_altitude() {
  // returns a new altitude which feeded into the alt.hold controller
  if(alt_change_flag == ASCENDING) {
    // we are below, going up
    if(alt.EstAlt >=  target_altitude) alt_change_flag = REACHED_ALT;
    // we shouldn't command past our target
    if(alt_to_hold >=  target_altitude) return target_altitude;
  } else if (alt_change_flag == DESCENDING) {
    // we are above, going down
    if(alt.EstAlt <=  target_altitude) alt_change_flag = REACHED_ALT;
    // we shouldn't command past our target
    if(alt_to_hold <=  target_altitude) return target_altitude;
  }
  // if we have reached our target altitude, return the target alt
  if(alt_change_flag == REACHED_ALT) return target_altitude;

  int32_t diff  = abs(alt_to_hold - target_altitude);
  // scale is how we generate a desired rate from the elapsed time
  // a smaller scale means faster rates
  int8_t _scale = 4;

  if (alt_to_hold < target_altitude) {
    // we are below the target alt
    if(diff < 200) _scale = 4;
    else _scale = 3;
  } else {
    // we are above the target, going down
    if(diff < 400) _scale = 5;  //Slow down if only 4meters above
    if(diff < 100) _scale = 6;  //Slow down further if within 1meter
```

```
  }

  // we use the elapsed time as our altitude offset
  // 1000 = 1 sec
  // 1000 >> 4 = 64cm/s descent by default
  int32_t change = (millis() - alt_change_timer) >> _scale;

  if(alt_change_flag == ASCENDING){
    alt_change += change;
  } else {
     alt_change -= change;
  }
  // for generating delta time
  alt_change_timer = millis();

  return original_altitude + alt_change;
}

////////////////////////////////////////////////////////////////////////////////
//PID based GPS navigation functions
//Author : EOSBandi
//Based on code and ideas from the Arducopter team: Jason Short,Randy Mackay, Pat
Hickey, Jose Julio, Jani Hirvinen
//Andrew Tridgell, Justin Beech, Adam Rivera, Jean-Louis Naudin, Roberto Navoni

//original constraint does not work with variables
int16_t constrain_int16(int16_t amt, int16_t low, int16_t high) {
  return ((amt)<(low)?(low):((amt)>(high)?(high):(amt)));
}
////////////////////////////////////////////////////////////////////////////////
// this is used to offset the shrinking longitude as we go towards the poles
// It's ok to calculate this once per waypoint setting, since it changes a little within the
reach of a multicopter
//
void GPS_calc_longitude_scaling(int32_t lat) {
  GPS_scaleLonDown = cos(lat * 1.0e-7f * 0.01745329251f);
}

////////////////////////////////////////////////////////////////////////////////
// Sets the waypoint to navigate, reset neccessary variables and calculate initial values
//
void GPS_set_next_wp(int32_t* lat_to, int32_t* lon_to, int32_t* lat_from, int32_t*
lon_from) {
  GPS_WP[LAT] = *lat_to;
  GPS_WP[LON] = *lon_to;

  GPS_FROM[LAT] = *lat_from;
  GPS_FROM[LON] = *lon_from;

  GPS_calc_longitude_scaling(*lat_to);
```

```
GPS_bearing(&GPS_FROM[LAT],&GPS_FROM[LON],&GPS_WP[LAT],&GPS_
WP[LON],&target_bearing);

GPS_distance_cm(&GPS_FROM[LAT],&GPS_FROM[LON],&GPS_WP[LAT],&G
PS_WP[LON],&wp_distance);

GPS_calc_location_error(&GPS_WP[LAT],&GPS_WP[LON],&GPS_FROM[LAT],
&GPS_FROM[LON]);
  waypoint_speed_gov = GPS_conf.nav_speed_min;
  original_target_bearing = target_bearing;

}

////////////////////////////////////////////////////////////////////////////////
// Check if we missed the destination somehow
//
static bool check_missed_wp(void) {
  int32_t temp;
  temp = target_bearing - original_target_bearing;
  temp = wrap_18000(temp);
  return (abs(temp) > 10000);   // we passed the waypoint by 100 degrees
}

////////////////////////////////////////////////////////////////////////////////
// Get distance between two points in cm
// Get bearing from pos1 to pos2, returns an 1deg = 100 precision

void GPS_bearing(int32_t* lat1, int32_t* lon1, int32_t* lat2, int32_t* lon2, int32_t*
bearing) {
  int32_t off_x = *lon2 - *lon1;
  int32_t off_y = (*lat2 - *lat1) / GPS_scaleLonDown;

  *bearing = 9000 + atan2(-off_y, off_x) * 5729.57795f;      //Convert the output
redians to 100xdeg
  if (*bearing < 0) *bearing += 36000;
}

void GPS_distance_cm(int32_t* lat1, int32_t* lon1, int32_t* lat2, int32_t*
lon2,uint32_t* dist) {
  float dLat = (float)(*lat2 - *lat1);                          // difference of latitude in
1/10 000 000 degrees
  float dLon = (float)(*lon2 - *lon1) * GPS_scaleLonDown; //x
  *dist = sqrt(sq(dLat) + sq(dLon)) * 1.11318845f;
}

//****************************************************************
*********************************
// calc_velocity_and_filtered_position - velocity in lon and lat directions calculated
```

```
from GPS position
//      and accelerometer data
// lon_speed expressed in cm/s.  positive numbers mean moving east
// lat_speed expressed in cm/s.  positive numbers when moving north
// Note: we use gps locations directly to calculate velocity instead of asking gps for
velocity because
//      this is more accurate below 1.5m/s
// Note: even though the positions are projected using a lead filter, the velocities are
calculated
//      from the unaltered gps locations.  We do not want noise from our lead filter
affecting velocity
//*****************************************************************
**********************************
static void GPS_calc_velocity(void){
  static int16_t speed_old[2] = {0,0};
  static int32_t last[2] = {0,0};
  static uint8_t init = 0;

  if (init) {
    float tmp = 1.0/dTnav;
    actual_speed[_X] = (float)(GPS_coord[LON] - last[LON]) *  GPS_scaleLonDown
* tmp;
    actual_speed[_Y] = (float)(GPS_coord[LAT]  - last[LAT])  * tmp;

    //TODO: Check unrealistic speed changes and signal navigation about posibble gps
signal degradation
    if (!GPS_conf.lead_filter) {
      actual_speed[_X] = (actual_speed[_X] + speed_old[_X]) / 2;
      actual_speed[_Y] = (actual_speed[_Y] + speed_old[_Y]) / 2;

      speed_old[_X] = actual_speed[_X];
      speed_old[_Y] = actual_speed[_Y];
    }
  }
  init=1;

  last[LON] = GPS_coord[LON];
  last[LAT] = GPS_coord[LAT];

  if (GPS_conf.lead_filter) {
    GPS_coord_lead[LON] = xLeadFilter.get_position(GPS_coord[LON],
actual_speed[_X], GPS_LAG);
    GPS_coord_lead[LAT] = yLeadFilter.get_position(GPS_coord[LAT],
actual_speed[_Y], GPS_LAG);
  }
}

////////////////////////////////////////////////////////////////////////////////
// Calculate a location error between two gps coordinates
// Because we are using lat and lon to do our distance errors here's a quick chart:
```

```
//   100  = 1m
//  1000  = 11m    = 36 feet
//  1800  = 19.80m = 60 feet
//  3000  = 33m
// 10000  = 111m
//
static void GPS_calc_location_error( int32_t* target_lat, int32_t* target_lng, int32_t*
gps_lat, int32_t* gps_lng ) {
  error[LON] = (float)(*target_lng - *gps_lng) * GPS_scaleLonDown;  // X Error
  error[LAT] = *target_lat - *gps_lat; // Y Error
}

////////////////////////////////////////////////////////////////////////////////
// Calculate nav_lat and nav_lon from the x and y error and the speed
//
// TODO: check that the poshold target speed constraint can be increased for snappier
poshold lock
static void GPS_calc_poshold(void) {
  int32_t d;
  int32_t target_speed;
  uint8_t axis;

  for (axis=0;axis<2;axis++) {
    target_speed = get_P(error[axis], &posholdPID_PARAM); // calculate desired
speed from lat/lon error
    target_speed = constrain(target_speed,-100,100);      // Constrain the target speed in
poshold mode to 1m/s it helps avoid runaways..
    rate_error[axis] = target_speed - actual_speed[axis]; // calc the speed error

    nav[axis]      =
        get_P(rate_error[axis],
&poshold_ratePID_PARAM)
       +get_I(rate_error[axis] + error[axis], &dTnav, &poshold_ratePID[axis],
&poshold_ratePID_PARAM);

    d = get_D(error[axis],                  &dTnav, &poshold_ratePID[axis],
&poshold_ratePID_PARAM);

    d = constrain(d, -2000, 2000);

    // get rid of noise
    if(abs(actual_speed[axis]) < 50) d = 0;

    nav[axis] +=d;
    // nav[axis]  = constrain(nav[axis], -NAV_BANK_MAX, NAV_BANK_MAX);
    nav[axis]  = constrain_int16(nav[axis], -GPS_conf.nav_bank_max,
GPS_conf.nav_bank_max);
    navPID[axis].integrator = poshold_ratePID[axis].integrator;
  }
}
```

```
//////////////////////////////////////////////////////////////////////////////
// Calculate the desired nav_lat and nav_lon for distance flying such as RTH and WP
//
static void GPS_calc_nav_rate( uint16_t max_speed) {
  float trig[2];
  int32_t target_speed[2];
  int32_t tilt;
  uint8_t axis;

  GPS_update_crosstrack();
  int16_t cross_speed = crosstrack_error * (GPS_conf.crosstrack_gain / 100.0);
//check is it ok ?
  cross_speed = constrain(cross_speed,-200,200);
  cross_speed = -cross_speed;

  float temp = (9000l - target_bearing) * RADX100;
  trig[_X] = cos(temp);
  trig[_Y] = sin(temp);

  target_speed[_X] = max_speed * trig[_X] - cross_speed * trig[_Y];
  target_speed[_Y] = cross_speed * trig[_X] + max_speed * trig[_Y];

  for (axis=0;axis<2;axis++) {
    rate_error[axis] = target_speed[axis] - actual_speed[axis];
    rate_error[axis] = constrain(rate_error[axis],-1000,1000);
    nav[axis]      =
        get_P(rate_error[axis],                    &navPID_PARAM)
       +get_I(rate_error[axis], &dTnav, &navPID[axis], &navPID_PARAM)
       +get_D(rate_error[axis], &dTnav, &navPID[axis], &navPID_PARAM);

    // nav[axis] = constrain(nav[axis],-NAV_BANK_MAX,NAV_BANK_MAX);
    nav[axis]  = constrain_int16(nav[axis], -GPS_conf.nav_bank_max,
GPS_conf.nav_bank_max);
    poshold_ratePID[axis].integrator = navPID[axis].integrator;
  }
}

static void GPS_update_crosstrack(void) {
  // Crosstrack Error
  // ----------------
  // If we are too far off or too close we don't do track following
  float temp = (target_bearing - original_target_bearing) * RADX100;
  crosstrack_error = sin(temp) * wp_distance; // Meters we are off track line
}

//////////////////////////////////////////////////////////////////////////////
// Determine desired speed when navigating towards a waypoint, also implement slow
// speed rampup when starting a navigation
//
```

```
//      |< WP Radius
//      0  1   2   3   4   5   6   7   8m
//      ...|...|...|...|...|...|...|...|
//                100  |  200     300     400cm/s
//                 |                                +|+
//                 |< we should slow to 1 m/s as we hit the target
//
static uint16_t GPS_calc_desired_speed(uint16_t max_speed, bool _slow) {
  if(_slow){
    max_speed = min(max_speed, wp_distance / 2);
  } else {
    max_speed = min(max_speed, wp_distance);
    max_speed = max(max_speed, GPS_conf.nav_speed_min);  // go at least
nav_speed_min
  }
  // limit the ramp up of the speed
  // waypoint_speed_gov is reset to 0 at each new WP command
  if(max_speed > waypoint_speed_gov){
    waypoint_speed_gov += (int)(100.0 * dTnav); // increase at .5/ms
    max_speed = waypoint_speed_gov;
  }
  return max_speed;
}

////////////////////////////////////////////////////////////////////////////////
// Utilities
//

int32_t wrap_36000(int32_t ang) {
  if (ang > 36000) ang -= 36000;
  if (ang < 0)     ang += 36000;
  return ang;
}


/*
 * EOS increased the precision here, even if we think that the gps is not precise
enough, with 10e5 precision it has 76cm resolution
 * with 10e7 it's around 1 cm now. Increasing it further is irrelevant, since even 1cm
resolution is unrealistic, however increased
 * resolution also increased precision of nav calculations
*/

#define DIGIT_TO_VAL(_x)        (_x - '0')
uint32_t GPS_coord_to_degrees(char* s) {
  char *p, *q;
  uint8_t deg = 0, min = 0;
  unsigned int frac_min = 0;
  uint8_t i;
```

```
  // scan for decimal point or end of field
  for (p = s; isdigit(*p); p++) ;
  q = s;

  // convert degrees
  while ((p - q) > 2) {
    if (deg)
      deg *= 10;
    deg += DIGIT_TO_VAL(*q++);
  }
  // convert minutes
  while (p > q) {
    if (min)
      min *= 10;
    min += DIGIT_TO_VAL(*q++);
  }
  // convert fractional minutes
  // expect up to four digits, result is in
  // ten-thousandths of a minute
  if (*p == '.') {
    q = p + 1;
    for (i = 0; i < 4; i++) {
      frac_min *= 10;
      if (isdigit(*q))
        frac_min += *q++ - '0';
    }
  }
  return deg * 10000000UL + (min * 1000000UL + frac_min*100UL) / 6;
}

// helper functions
uint16_t grab_fields(char* src, uint8_t mult) {  // convert string to uint16
  uint8_t i;
  uint16_t tmp = 0;

  for(i=0; src[i]!=0; i++) {
    if(src[i] == '.') {
      i++;
      if(mult==0)   break;
      else  src[i+mult] = 0;
    }
    tmp *= 10;
    if(src[i] >='0' && src[i] <='9') tmp += src[i]-'0';
  }
  return tmp;
}

uint8_t hex_c(uint8_t n) {    // convert '0'..'9','A'..'F' to 0..15
  n -= '0';
  if(n>9)  n -= 7;
```

```
  n &= 0x0F;
  return n;
}

//*****************************************************************
****
// Common GPS functions
//
void init_RTH() {
  f.GPS_mode = GPS_MODE_RTH;          // Set GPS_mode to RTH
  f.GPS_BARO_MODE = true;
  GPS_hold[LAT] = GPS_coord[LAT];     //All RTH starts with a poshold
  GPS_hold[LON] = GPS_coord[LON];     //This allows to raise to rth altitude
  GPS_set_next_wp(&GPS_hold[LAT],&GPS_hold[LON], &GPS_hold[LAT],
&GPS_hold[LON]);
  NAV_paused_at = 0;
  if (GPS_conf.rth_altitude == 0) set_new_altitude(alt.EstAlt);     //Return at actual
altitude
  else {                                          // RTH altitude is defined, but we use it
only if we are below it
    if (alt.EstAlt < GPS_conf.rth_altitude * 100)
      set_new_altitude(GPS_conf.rth_altitude * 100);
    else set_new_altitude(alt.EstAlt);
  }
  f.GPS_head_set = 0;                                 //Allow the RTH ti handle heading
  NAV_state = NAV_STATE_RTH_START;                          //NAV engine status
is Starting RTH.
}

void GPS_reset_home_position(void) {
  if (f.GPS_FIX && GPS_numSat >= 5) {
    GPS_home[LAT] = GPS_coord[LAT];
    GPS_home[LON] = GPS_coord[LON];
    GPS_calc_longitude_scaling(GPS_coord[LAT]);    //need an initial value for
distance and bearing calc
    nav_takeoff_bearing = att.heading;            //save takeoff heading
    //TODO: Set ground altitude
    f.GPS_FIX_HOME = 1;
  }
}

//reset navigation (stop the navigation processor, and clear nav)
void GPS_reset_nav(void) {
  uint8_t i;

  for(i=0;i<2;i++) {
    nav[i] = 0;
    reset_PID(&posholdPID[i]);
    reset_PID(&poshold_ratePID[i]);
    reset_PID(&navPID[i]);
```

```
    NAV_state = NAV_STATE_NONE;
    //invalidate JUMP counter
    jump_times = -10;
    //reset next step counter
    next_step = 1;
    //Clear poi
    GPS_poi[LAT] = 0; GPS_poi[LON] = 0;
    f.GPS_head_set = 0;
   }
}

//Get the relevant P I D values and set the PID controllers
void GPS_set_pids(void) {
  posholdPID_PARAM.kP   = (float)conf.pid[PIDPOS].P8/100.0;
  posholdPID_PARAM.kI   = (float)conf.pid[PIDPOS].I8/100.0;
  posholdPID_PARAM.Imax = POSHOLD_RATE_IMAX * 100;

  poshold_ratePID_PARAM.kP   = (float)conf.pid[PIDPOSR].P8/10.0;
  poshold_ratePID_PARAM.kI   = (float)conf.pid[PIDPOSR].I8/100.0;
  poshold_ratePID_PARAM.kD   = (float)conf.pid[PIDPOSR].D8/1000.0;
  poshold_ratePID_PARAM.Imax = POSHOLD_RATE_IMAX * 100;

  navPID_PARAM.kP   = (float)conf.pid[PIDNAVR].P8/10.0;
  navPID_PARAM.kI   = (float)conf.pid[PIDNAVR].I8/100.0;
  navPID_PARAM.kD   = (float)conf.pid[PIDNAVR].D8/1000.0;
  navPID_PARAM.Imax = POSHOLD_RATE_IMAX * 100;
  }
//It was moved here since even i2cgps code needs it
int32_t wrap_18000(int32_t ang) {
  if (ang > 18000)  ang -= 36000;
  if (ang < -18000) ang += 36000;
  return ang;
}
```

```
/*************************************************************************
*****************/
/*************************************************************************
*****************/
```

```
/**********************       specific  GPS device section
*************************/
/*****************************************************************************
******************/
/*****************************************************************************
******************/

#if defined(GPS_SERIAL)

/*****************************************************************************
******************/
/**********************       NMEA
*************************/
/*****************************************************************************
******************/
#if defined(NMEA)
/* This is a light implementation of a GPS frame decoding
   This should work with most of modern GPS devices configured to output NMEA
frames.
   It assumes there are some NMEA GGA frames to decode on the serial bus
   Here we use only the following data :
     - latitude
     - longitude
     - GPS fix is/is not ok
     - GPS num sat (4 is enough to be +/- reliable)
     - GPS altitude
     - GPS speed
*/
#define FRAME_GGA  1
#define FRAME_RMC  2

void GPS_SerialInit(void) {
  SerialOpen(GPS_SERIAL,GPS_BAUD);
  delay(1000);
}

bool GPS_newFrame(uint8_t c) {
  uint8_t frameOK = 0;
  static uint8_t param = 0, offset = 0, parity = 0;
  static char string[15];
  static uint8_t checksum_param, frame = 0;

  if (c == '$') {
    param = 0; offset = 0; parity = 0;
  } else if (c == ',' || c == '*') {
    string[offset] = 0;
    if (param == 0) { //frame identification
      frame = 0;
      if (string[0] == 'G' && string[1] == 'P' && string[2] == 'G' && string[3] == 'G'
&& string[4] == 'A') frame = FRAME_GGA;
```

```
    if (string[0] == 'G' && string[1] == 'P' && string[2] == 'R' && string[3] == 'M'
&& string[4] == 'C') frame = FRAME_RMC;
   } else if (frame == FRAME_GGA) {
    if     (param == 2)                 {GPS_coord[LAT] =
GPS_coord_to_degrees(string);}
    else if (param == 3 && string[0] == 'S') GPS_coord[LAT] = -GPS_coord[LAT];
    else if (param == 4)                 {GPS_coord[LON] =
GPS_coord_to_degrees(string);}
    else if (param == 5 && string[0] == 'W') GPS_coord[LON] = -GPS_coord[LON];
    else if (param == 6)                 {f.GPS_FIX = (string[0]  > '0');}
    else if (param == 7)                 {GPS_numSat = grab_fields(string,0);}
    else if (param == 9)                 {GPS_altitude = grab_fields(string,0);}  //
altitude in meters added by Mis
   } else if (frame == FRAME_RMC) {
    if     (param == 7)                 {GPS_speed =
((uint32_t)grab_fields(string,1)*5144L)/1000L;}  //gps speed in cm/s will be used for
navigation
    else if (param == 8)                 {GPS_ground_course = grab_fields(string,1); }
//ground course deg*10
   }
   param++; offset = 0;
   if (c == '*') checksum_param=1;
   else parity ^= c;
  } else if (c == '\r' || c == '\n') {
   if (checksum_param) { //parity checksum
    uint8_t checksum = hex_c(string[0]);
    checksum <<= 4;
    checksum += hex_c(string[1]);
    if (checksum == parity) frameOK = 1;
   }
   checksum_param=0;
  } else {
    if (offset < 15) string[offset++] = c;
    if (!checksum_param) parity ^= c;
  }
  return frameOK && (frame==FRAME_GGA);
}
#endif //NMEA



/**************************************************************************
*****************/
/************************       UBLOX
*************************/
/**************************************************************************
*****************/
#if defined(UBLOX)
const char UBLOX_INIT[] PROGMEM = {                                    //
PROGMEM array must be outside any function !!!
```

```
  0xB5,0x62,0x06,0x01,0x03,0x00,0xF0,0x05,0x00,0xFF,0x19,
//disable all default NMEA messages
  0xB5,0x62,0x06,0x01,0x03,0x00,0xF0,0x03,0x00,0xFD,0x15,
  0xB5,0x62,0x06,0x01,0x03,0x00,0xF0,0x01,0x00,0xFB,0x11,
  0xB5,0x62,0x06,0x01,0x03,0x00,0xF0,0x00,0x00,0xFA,0x0F,
  0xB5,0x62,0x06,0x01,0x03,0x00,0xF0,0x02,0x00,0xFC,0x13,
  0xB5,0x62,0x06,0x01,0x03,0x00,0xF0,0x04,0x00,0xFE,0x17,
  0xB5,0x62,0x06,0x01,0x03,0x00,0x01,0x02,0x01,0x0E,0x47,                  //set
POSLLH MSG rate
  0xB5,0x62,0x06,0x01,0x03,0x00,0x01,0x03,0x01,0x0F,0x49,                  //set
STATUS MSG rate
  0xB5,0x62,0x06,0x01,0x03,0x00,0x01,0x06,0x01,0x12,0x4F,                  //set
SOL MSG rate
  0xB5,0x62,0x06,0x01,0x03,0x00,0x01,0x12,0x01,0x1E,0x67,                  //set
VELNED MSG rate

0xB5,0x62,0x06,0x16,0x08,0x00,0x03,0x07,0x03,0x00,0x51,0x08,0x00,0x00,0x8A,
0x41,   //set WAAS to EGNOS
  0xB5, 0x62, 0x06, 0x08, 0x06, 0x00, 0xC8, 0x00, 0x01, 0x00, 0x01, 0x00, 0xDE,
0x6A //set rate to 5Hz
};

struct ubx_header {
  uint8_t preamble1;
  uint8_t preamble2;
  uint8_t msg_class;
  uint8_t msg_id;
  uint16_t length;
};
struct ubx_nav_posllh {
  uint32_t time;  // GPS msToW
  int32_t longitude;
  int32_t latitude;
  int32_t altitude_ellipsoid;
  int32_t altitude_msl;
  uint32_t horizontal_accuracy;
  uint32_t vertical_accuracy;
};
struct ubx_nav_solution {
  uint32_t time;
  int32_t time_nsec;
  int16_t week;
  uint8_t fix_type;
  uint8_t fix_status;
  int32_t ecef_x;
  int32_t ecef_y;
  int32_t ecef_z;
  uint32_t position_accuracy_3d;
  int32_t ecef_x_velocity;
  int32_t ecef_y_velocity;
```

```
  int32_t ecef_z_velocity;
  uint32_t speed_accuracy;
  uint16_t position_DOP;
  uint8_t res;
  uint8_t satellites;
  uint32_t res2;
};
struct ubx_nav_velned {
  uint32_t time;  // GPS msToW
  int32_t ned_north;
  int32_t ned_east;
  int32_t ned_down;
  uint32_t speed_3d;
  uint32_t speed_2d;
  int32_t heading_2d;
  uint32_t speed_accuracy;
  uint32_t heading_accuracy;
};

enum ubs_protocol_bytes {
  PREAMBLE1 = 0xb5,
  PREAMBLE2 = 0x62,
  CLASS_NAV = 0x01,
  CLASS_ACK = 0x05,
  CLASS_CFG = 0x06,
  MSG_ACK_NACK = 0x00,
  MSG_ACK_ACK = 0x01,
  MSG_POSLLH = 0x2,
  MSG_STATUS = 0x3,
  MSG_SOL = 0x6,
  MSG_VELNED = 0x12,
  MSG_CFG_PRT = 0x00,
  MSG_CFG_RATE = 0x08,
  MSG_CFG_SET_RATE = 0x01,
  MSG_CFG_NAV_SETTINGS = 0x24
};
enum ubs_nav_fix_type {
  FIX_NONE = 0,
  FIX_DEAD_RECKONING = 1,
  FIX_2D = 2,
  FIX_3D = 3,
  FIX_GPS_DEAD_RECKONING = 4,
  FIX_TIME = 5
};
enum ubx_nav_status_bits {
  NAV_STATUS_FIX_VALID = 1
};

// Receive buffer
static union {
```

```
  ubx_nav_posllh posllh;
  ubx_nav_solution solution;
  ubx_nav_velned velned;
  uint8_t bytes[];
 } _buffer;

uint32_t init_speed[5] = {9600,19200,38400,57600,115200};

static void SerialGpsPrint(const char PROGMEM * str) {
  char b;
  while(str && (b = pgm_read_byte(str++))) {
    SerialWrite(GPS_SERIAL, b);
    delay(5);
  }
}

void GPS_SerialInit(void) {
  SerialOpen(GPS_SERIAL,GPS_BAUD);
  delay(1000);
  for(uint8_t i=0;i<5;i++){
    SerialOpen(GPS_SERIAL,init_speed[i]);          // switch UART speed for sending
SET BAUDRATE command (NMEA mode)
    #if (GPS_BAUD==19200)
      SerialGpsPrint(PSTR("$PUBX,41,1,0003,0001,19200,0*23\r\n"));     // 19200
baud - minimal speed for 5Hz update rate
    #endif
    #if (GPS_BAUD==38400)
      SerialGpsPrint(PSTR("$PUBX,41,1,0003,0001,38400,0*26\r\n"));     // 38400
baud
    #endif
    #if (GPS_BAUD==57600)
      SerialGpsPrint(PSTR("$PUBX,41,1,0003,0001,57600,0*2D\r\n"));     // 57600
baud
    #endif
    #if (GPS_BAUD==115200)
      SerialGpsPrint(PSTR("$PUBX,41,1,0003,0001,115200,0*1E\r\n"));    // 115200
baud
    #endif
    while(!SerialTXfree(GPS_SERIAL)) delay(10);
  }
  delay(200);
  SerialOpen(GPS_SERIAL,GPS_BAUD);
  for(uint8_t i=0; i<sizeof(UBLOX_INIT); i++) {                    // send configuration
data in UBX protocol
    SerialWrite(GPS_SERIAL, pgm_read_byte(UBLOX_INIT+i));
    delay(5); //simulating a 38400baud pace (or less), otherwise commands are not
accepted by the device.
  }
}
```

```
bool GPS_newFrame(uint8_t data){
  static uint8_t  _step = 0; // State machine state
  static uint8_t  _msg_id;
  static uint16_t _payload_length;
  static uint16_t _payload_counter;
  static uint8_t  _ck_a; // Packet checksum accumulators
  static uint8_t  _ck_b;

  uint8_t st  = _step+1;
  bool    ret = false;

  if (st == 2)
    if (PREAMBLE2 != data) st--; // in case of faillure of the 2nd header byte, still test
the first byte
  if (st == 1) {
    if(PREAMBLE1 != data) st--;
  } else if (st == 3) { // CLASS byte, not used, assume it is CLASS_NAV
    _ck_b = _ck_a = data;  // reset the checksum accumulators
  } else if (st > 3 && st < 8) {
    _ck_b += (_ck_a += data);  // checksum byte
    if (st == 4) {
      _msg_id = data;
    } else if (st == 5) {
      _payload_length = data; // payload length low byte
    } else if (st == 6) {
      _payload_length += (uint16_t)(data<<8);
      if (_payload_length > 512) st = 0;
      _payload_counter = 0;  // prepare to receive payload
    } else {
      if (_payload_counter+1 < _payload_length) st--; // stay in the same state while
data inside the frame
      if (_payload_counter < sizeof(_buffer)) _buffer.bytes[_payload_counter] = data;
      _payload_counter++;
    }
  } else if (st == 8) {
    if (_ck_a != data) st = 0;  // bad checksum
  } else if (st == 9) {
    st = 0;
    if (_ck_b == data) { // good checksum
      if (_msg_id == MSG_POSLLH) {
        if(f.GPS_FIX) {
          GPS_coord[LON] = _buffer.posllh.longitude;
          GPS_coord[LAT] = _buffer.posllh.latitude;
          GPS_altitude   = _buffer.posllh.altitude_msl / 1000; //alt in m
          //GPS_time       = _buffer.posllh.time; //not used for the moment
        }
        ret= true;        // POSLLH message received, allow blink GUI icon and LED,
frame available for nav computation
      } else if (_msg_id ==  MSG_SOL) {
        f.GPS_FIX = 0;
```

```
        if((_buffer.solution.fix_status & NAV_STATUS_FIX_VALID) &&
(_buffer.solution.fix_type == FIX_3D || _buffer.solution.fix_type == FIX_2D))
f.GPS_FIX = 1;
        GPS_numSat = _buffer.solution.satellites;
      } else if (_msg_id ==  MSG_VELNED) {
        GPS_speed         = _buffer.velned.speed_2d;  // cm/s
        GPS_ground_course = (uint16_t)(_buffer.velned.heading_2d / 10000);  //
Heading 2D deg * 100000 rescaled to deg * 10 //not used for the moment
      }
    }
  }
  _step = st;
  return ret;
}
#endif //UBLOX



/**************************************************************************
*****************/
/************************       MTK
***************************/
/**************************************************************************
*****************/
#if defined(MTK_BINARY16) || defined(MTK_BINARY19)

#define MTK_SET_BINARY         PSTR("$PGCMD,16,0,0,0,0,0*6A\r\n")
#define MTK_SET_NMEA           PSTR("$PGCMD,16,1,1,1,1,1*6B\r\n")
#define MTK_SET_NMEA_SENTENCES
PSTR("$PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*28\r\n")
#define MTK_OUTPUT_4HZ         PSTR("$PMTK220,250*29\r\n")
#define MTK_OUTPUT_5HZ         PSTR("$PMTK220,200*2C\r\n")
#define MTK_OUTPUT_10HZ        PSTR("$PMTK220,100*2F\r\n")
#define MTK_NAVTHRES_OFF       PSTR("$PMTK397,0*23\r\n") // Set Nav
Threshold (the minimum speed the GPS must be moving to update the position) to 0
m/s
#define SBAS_ON                PSTR("$PMTK313,1*2E\r\n")
#define WAAS_ON                PSTR("$PMTK301,2*2E\r\n")
#define SBAS_TEST_MODE         PSTR("$PMTK319,0*25\r\n")  //Enable test use
of sbas satelite in test mode (usually PRN124 is in test mode)

struct diyd_mtk_msg {
  int32_t  latitude;
  int32_t  longitude;
  int32_t  altitude;
  int32_t  ground_speed;
  int32_t  ground_course;
  uint8_t  satellites;
  uint8_t  fix_type;
  uint32_t utc_date;
```

```
  uint32_t utc_time;
  uint16_t hdop;
};

// #pragma pack(pop)
enum diyd_mtk_fix_type {
  FIX_NONE = 1,
  FIX_2D = 2,
  FIX_3D = 3,
  FIX_2D_SBAS = 6,
  FIX_3D_SBAS = 7
};

#if defined(MTK_BINARY16)
enum diyd_mtk_protocol_bytes {
  PREAMBLE1 = 0xd0,
  PREAMBLE2 = 0xdd,
};
#endif

#if defined(MTK_BINARY19)
enum diyd_mtk_protocol_bytes {
  PREAMBLE1 = 0xd1,
  PREAMBLE2 = 0xdd,
};
#endif

// Packet checksum accumulators
uint8_t  _ck_a;
uint8_t  _ck_b;

// State machine state
uint8_t  _step;
uint8_t  _payload_counter;

// Time from UNIX Epoch offset
long  _time_offset;
bool  _offset_calculated;

// Receive buffer
union {
  diyd_mtk_msg  msg;
  uint8_t       bytes[];
} _buffer;

inline long _swapl(const void *bytes) {
  const uint8_t *b = (const uint8_t *)bytes;
  union {
    long    v;
    uint8_t b[4];
```

```
  } u;

  u.b[0] = b[3];
  u.b[1] = b[2];
  u.b[2] = b[1];
  u.b[3] = b[0];

  return(u.v);
}

uint32_t init_speed[5] = {9600,19200,38400,57600,115200};

void SerialGpsPrint(const char PROGMEM * str) {
  char b;
  while(str && (b = pgm_read_byte(str++))) {
    SerialWrite(GPS_SERIAL, b);
  }
}

void GPS_SerialInit(void) {
  SerialOpen(GPS_SERIAL,GPS_BAUD);
  delay(1000);
  #if defined(INIT_MTK_GPS)                          // MTK GPS setup
    for(uint8_t i=0;i<5;i++){
      SerialOpen(GPS_SERIAL,init_speed[i]);              // switch UART speed for
sending SET BAUDRATE command
      #if (GPS_BAUD==19200)
        SerialGpsPrint(PSTR("$PMTK251,19200*22\r\n"));     // 19200 baud - minimal
speed for 5Hz update rate
      #endif
      #if (GPS_BAUD==38400)
        SerialGpsPrint(PSTR("$PMTK251,38400*27\r\n"));     // 38400 baud
      #endif
      #if (GPS_BAUD==57600)
        SerialGpsPrint(PSTR("$PMTK251,57600*2C\r\n"));     // 57600 baud
      #endif
      #if (GPS_BAUD==115200)
        SerialGpsPrint(PSTR("$PMTK251,115200*1F\r\n"));    // 115200 baud
      #endif
      while(!SerialTXfree(GPS_SERIAL)) delay(80);
    }
    // at this point we have GPS working at selected (via #define GPS_BAUD)
baudrate
    // So now we have to set the desired mode and update rate (which depends on the
NMEA or MTK_BINARYxx settings)
    SerialOpen(GPS_SERIAL,GPS_BAUD);

    SerialGpsPrint(MTK_NAVTHRES_OFF);
      while(!SerialTXfree(GPS_SERIAL)) delay(80);
    SerialGpsPrint(SBAS_ON);
```

```
    while(!SerialTXfree(GPS_SERIAL)) delay(80);
   SerialGpsPrint(WAAS_ON);
    while(!SerialTXfree(GPS_SERIAL)) delay(80);
   SerialGpsPrint(SBAS_TEST_MODE);
    while(!SerialTXfree(GPS_SERIAL)) delay(80);
   SerialGpsPrint(MTK_OUTPUT_5HZ);          // 5 Hz update rate

   #if defined(NMEA)
    SerialGpsPrint(MTK_SET_NMEA_SENTENCES); // only GGA and RMC
sentence
   #endif
   #if defined(MTK_BINARY19) || defined(MTK_BINARY16)
    SerialGpsPrint(MTK_SET_BINARY);
   #endif
 #endif  // init_mtk_gps
}

bool GPS_newFrame(uint8_t data) {
 bool parsed = false;

 restart:
 switch(_step) {
   // Message preamble, class, ID detection
   //
   // If we fail to match any of the expected bytes, we
   // reset the state machine and re-consider the failed
   // byte as the first byte of the preamble.  This
   // improves our chances of recovering from a mismatch
   // and makes it less likely that we will be fooled by
   // the preamble appearing as data in some other message.
   //
   case 0:
    if(PREAMBLE1 == data)
     _step++;
    break;
   case 1:
    if (PREAMBLE2 == data) {
     _step++;
     break;
    }
    _step = 0;
    goto restart;
   case 2:
    if (sizeof(_buffer) == data) {
     _step++;
     _ck_b = _ck_a = data;               // reset the checksum accumulators
     _payload_counter = 0;
    } else {
     _step = 0;                          // reset and wait for a message of the right class
     goto restart;
```

```
      }
      break;
      // Receive message data
    case 3:
      _buffer.bytes[_payload_counter++] = data;
      _ck_b += (_ck_a += data);
      if (_payload_counter == sizeof(_buffer))
        _step++;
      break;
      // Checksum and message processing
    case 4:
      _step++;
      if (_ck_a != data)
        _step = 0;
      break;
    case 5:
      _step = 0;
      if (_ck_b != data)
        break;
      f.GPS_FIX                 = ((_buffer.msg.fix_type == FIX_3D) ||
(_buffer.msg.fix_type == FIX_3D_SBAS));
      #if defined(MTK_BINARY16)
      GPS_coord[LAT]            = _buffer.msg.latitude * 10;    // XXX doc says *10e7
but device says otherwise
      GPS_coord[LON]            = _buffer.msg.longitude * 10;   // XXX doc says *10e7
but device says otherwise
      #endif
      #if defined(MTK_BINARY19)
      GPS_coord[LAT]            = _buffer.msg.latitude;         // With 1.9 now we have
real 10e7 precision
      GPS_coord[LON]            = _buffer.msg.longitude;
      #endif
      GPS_altitude              = _buffer.msg.altitude /100;    // altitude in meter
      GPS_speed                 = _buffer.msg.ground_speed;     // in m/s * 100 == in cm/s
      GPS_ground_course         = _buffer.msg.ground_course/100;  //in degrees
      GPS_numSat                = _buffer.msg.satellites;
      //GPS_time                = _buffer.msg.utc_time;
      //GPS_hdop                = _buffer.msg.hdop;
      parsed = true;
  }
  return parsed;
}
#endif //MTK

#endif //GPS SERIAL



/**************************************************************************
```

```
******************/
/***************          I2C GPS          ********************/
/*************************************************************************
******************/
#if defined(I2C_GPS)
#define I2C_GPS_ADDRESS             0x20 //7 bits

#define I2C_GPS_STATUS_00           00    //(Read only)
  #define I2C_GPS_STATUS_NEW_DATA       0x01  // New data is available (after
every GGA frame)
  #define I2C_GPS_STATUS_2DFIX          0x02  // 2dfix achieved
  #define I2C_GPS_STATUS_3DFIX          0x04  // 3dfix achieved
  #define I2C_GPS_STATUS_NUMSATS        0xF0  // Number of sats in view
#define I2C_GPS_LOCATION            07    // current location 8 byte (lat, lon)
int32_t
#define I2C_GPS_GROUND_SPEED        31    // GPS ground speed in m/s*100
(uint16_t)      (Read Only)
#define I2C_GPS_ALTITUDE            33    // GPS altitude in meters (uint16_t)
(Read Only)
#define I2C_GPS_GROUND_COURSE       35    // GPS ground course (uint16_t)
#define I2C_GPS_TIME                39    // UTC Time from GPS in hhmmss.sss * 100
(uint32_t)(unneccesary precision) (Read Only)
#define I2C_GPS_SONAR_ALT           239   // Sonar Altitude

uint8_t GPS_NewData(void) {
  uint8_t i2c_gps_status;

  i2c_gps_status = i2c_readReg(I2C_GPS_ADDRESS,I2C_GPS_STATUS_00);
//Get status register

  #if defined(I2C_GPS_SONAR)
  i2c_read_reg_to_buf(I2C_GPS_ADDRESS, I2C_GPS_SONAR_ALT,
(uint8_t*)&sonarAlt,2);
  #endif

  f.GPS_FIX = 0;
  if (i2c_gps_status & I2C_GPS_STATUS_3DFIX) {                        //Check is
we have a good 3d fix (numsats>5)
    f.GPS_FIX = 1;
    if (i2c_gps_status & I2C_GPS_STATUS_NEW_DATA) {
//Check about new data
      GPS_Frame = 1;
      if (GPS_update == 1) GPS_update = 0; else GPS_update = 1; //Blink GPS update
      GPS_numSat = i2c_gps_status >>4;
      i2c_read_reg_to_buf(I2C_GPS_ADDRESS, I2C_GPS_LOCATION,
(uint8_t*)&GPS_coord[LAT],4);
      i2c_read_reg_to_buf(I2C_GPS_ADDRESS, I2C_GPS_LOCATION+4,
(uint8_t*)&GPS_coord[LON],4);
      // note: the following vars are currently not used in nav code -- avoid retrieving it
to save time
```

```
    //i2c_read_reg_to_buf(I2C_GPS_ADDRESS, I2C_GPS_GROUND_SPEED,
(uint8_t*)&GPS_speed,2);
    //i2c_read_reg_to_buf(I2C_GPS_ADDRESS, I2C_GPS_ALTITUDE,
(uint8_t*)&GPS_altitude,2);
    //i2c_read_reg_to_buf(I2C_GPS_ADDRESS,
I2C_GPS_GROUND_COURSE,(uint8_t*)&GPS_ground_course,2);
    return 1;
   }
  }
 return 0;
}
#endif //I2C_GPS



#endif // GPS Defined
```

καρτέλα GPS.h

```
#ifndef GPS_H_
#define GPS_H_

//Function prototypes for GPS frame parsing
bool GPS_newFrame(uint8_t c);
extern uint8_t GPS_Frame;          // a valid GPS_Frame was detected, and data is
ready for nav computation

extern int32_t wrap_18000(int32_t ang);

void GPS_set_pids(void);
void GPS_SerialInit(void);
uint8_t GPS_Compute(void);
void GPS_reset_home_position(void);
void GPS_set_next_wp(int32_t* lat_to, int32_t* lon_to, int32_t* lat_from, int32_t*
lon_from);
void GPS_reset_nav(void);

int32_t get_altitude_error();
void clear_new_altitude();
void force_new_altitude(int32_t _new_alt);
void set_new_altitude(int32_t _new_alt);
int32_t get_new_altitude();
void abort_mission(unsigned char error_code);
void GPS_adjust_heading();
void init_RTH(void);
void check_land(void);

#if defined(I2C_GPS)
uint8_t GPS_NewData(void);
#endif

extern uint32_t wp_distance;
```

```
extern int32_t target_bearing;
#endif /* GPS_H_ */
```
καρτέλα IMU.cpp
```
#include "Arduino.h"
#include "config.h"
#include "def.h"
#include "types.h"
#include "MultiWii.h"
#include "IMU.h"
#include "Sensors.h"

void getEstimatedAttitude();

void computeIMU () {
  uint8_t axis;
  static int16_t gyroADCprevious[3] = {0,0,0};
  static int16_t gyroADCinter[3];

  uint16_t timeInterleave = 0;
  #if ACC
    ACC_getADC();
    getEstimatedAttitude();
  #endif
  #if GYRO
    Gyro_getADC();
  #endif
  for (axis = 0; axis < 3; axis++)
    gyroADCinter[axis] =  imu.gyroADC[axis];
  timeInterleave=micros();
  annexCode();
  uint8_t t=0;
  while((int16_t)(micros()-timeInterleave)<650) t=1; //empirical, interleaving delay
between 2 consecutive reads
  #ifdef LCD_TELEMETRY
    if (!t) annex650_overrun_count++;
  #endif
  #if GYRO
    Gyro_getADC();
  #endif
  for (axis = 0; axis < 3; axis++) {
    gyroADCinter[axis] =  imu.gyroADC[axis]+gyroADCinter[axis];
    // empirical, we take a weighted value of the current and the previous values
    imu.gyroData[axis] = (gyroADCinter[axis]+gyroADCprevious[axis])/3;
    gyroADCprevious[axis] = gyroADCinter[axis]>>1;
    if (!ACC) imu.accADC[axis]=0;
  }
  #if defined(GYRO_SMOOTHING)
    static int16_t gyroSmooth[3] = {0,0,0};
    for (axis = 0; axis < 3; axis++) {
      imu.gyroData[axis] = (int16_t) ( ( (int32_t)((int32_t)gyroSmooth[axis] *
```

```
(conf.Smoothing[axis]-1) )+imu.gyroData[axis]+1 ) / conf.Smoothing[axis]);
      gyroSmooth[axis] = imu.gyroData[axis];
    }
  #elif defined(TRI)
    static int16_t gyroYawSmooth = 0;
    imu.gyroData[YAW] = (gyroYawSmooth*2+imu.gyroData[YAW])/3;
    gyroYawSmooth = imu.gyroData[YAW];
  #endif
}

// ************************************************
// Simplified IMU based on "Complementary Filter"
// Inspired by http://starlino.com/imu_guide.html
//
// adapted by ziss_dm : http://www.multiwii.com/forum/viewtopic.php?f=8&t=198
//
// The following ideas was used in this project:
// 1) Rotation matrix: http://en.wikipedia.org/wiki/Rotation_matrix
// 2) Small-angle approximation: http://en.wikipedia.org/wiki/Small-
angle_approximation
// 3) C. Hastings approximation for atan2()
// 4) Optimization tricks: http://www.hackersdelight.org/
//
// Currently Magnetometer uses separate CF which is used only
// for heading approximation.
//
// ************************************************

//******  advanced users settings ******************
/* Set the Low Pass Filter factor for ACC
   Increasing this value would reduce ACC noise (visible in GUI), but would increase
ACC lag time
   Comment this if  you do not want filter at all.
   unit = n power of 2 */
// this one is also used for ALT HOLD calculation, should not be changed
#ifndef ACC_LPF_FACTOR
  #define ACC_LPF_FACTOR 4 // that means a LPF of 16
#endif

/* Set the Gyro Weight for Gyro/Acc complementary filter
   Increasing this value would reduce and delay Acc influence on the output of the
filter*/
#ifndef GYR_CMPF_FACTOR
  #define GYR_CMPF_FACTOR 10 //  that means a CMP_FACTOR of 1024 (2^10)
#endif

/* Set the Gyro Weight for Gyro/Magnetometer complementary filter
   Increasing this value would reduce and delay Magnetometer influence on the output
of the filter*/
#define GYR_CMPFM_FACTOR 8 // that means a CMP_FACTOR of 256 (2^8)
```

```
typedef struct  {
  int32_t X,Y,Z;
} t_int32_t_vector_def;

typedef struct  {
  uint16_t XL; int16_t X;
  uint16_t YL; int16_t Y;
  uint16_t ZL; int16_t Z;
} t_int16_t_vector_def;

// note: we use implicit first 16 MSB bits 32 -> 16 cast. ie V32.X>>16 = V16.X
typedef union {
  int32_t A32[3];
  t_int32_t_vector_def V32;
  int16_t A16[6];
  t_int16_t_vector_def V16;
} t_int32_t_vector;

//return angle , unit: 1/10 degree
int16_t _atan2(int32_t y, int32_t x){
  float z = y;
  int16_t a;
  uint8_t c;
  c = abs(y) < abs(x);
  if ( c ) {z = z / x;} else {z = x / z;}
  a = 2046.43 * (z / (3.5714 +  z * z));
  if ( c ){
   if (x<0) {
     if (y<0) a -= 1800;
     else a += 1800;
   }
  } else {
    a = 900 - a;
    if (y<0) a -= 1800;
  }
  return a;
}

float InvSqrt (float x){
  union{
    int32_t i;
    float   f;
  } conv;
  conv.f = x;
  conv.i = 0x5f1ffff9 - (conv.i >> 1);
  return conv.f * (1.68191409f - 0.703952253f * x * conv.f * conv.f);
}
```

```
// signed16 * signed16
// 22 cycles
// http://mekonik.wordpress.com/2009/03/18/arduino-avr-gcc-multiplication/
#define MultiS16X16to32(longRes, intIn1, intIn2) \
asm volatile ( \
"clr r26 \n\t" \
"mul %A1, %A2 \n\t" \
"movw %A0, r0 \n\t" \
"muls %B1, %B2 \n\t" \
"movw %C0, r0 \n\t" \
"mulsu %B2, %A1 \n\t" \
"sbc %D0, r26 \n\t" \
"add %B0, r0 \n\t" \
"adc %C0, r1 \n\t" \
"adc %D0, r26 \n\t" \
"mulsu %B1, %A2 \n\t" \
"sbc %D0, r26 \n\t" \
"add %B0, r0 \n\t" \
"adc %C0, r1 \n\t" \
"adc %D0, r26 \n\t" \
"clr r1 \n\t" \
: \
"=&r" (longRes) \
: \
"a" (intIn1), \
"a" (intIn2) \
: \
"r26" \
)

int32_t __attribute__ ((noinline)) mul(int16_t a, int16_t b) {
  int32_t r;
  MultiS16X16to32(r, a, b);
  //r = (int32_t)a*b; without asm requirement
  return r;
}

// Rotate Estimated vector(s) with small angle approximation, according to the gyro
data
void rotateV32( t_int32_t_vector *v,int16_t* delta) {
  int16_t X = v->V16.X;
  int16_t Y = v->V16.Y;
  int16_t Z = v->V16.Z;

  v->V32.Z -=  mul(delta[ROLL]  ,  X)  + mul(delta[PITCH] , Y);
  v->V32.X +=  mul(delta[ROLL]  ,  Z)  - mul(delta[YAW]   , Y);
  v->V32.Y +=  mul(delta[PITCH] ,  Z)  + mul(delta[YAW]   , X);
}

static int16_t accZ=0;
```

```
void getEstimatedAttitude(){
  uint8_t axis;
  int32_t accMag = 0;
  float scale;
  int16_t deltaGyroAngle16[3];
  static t_int32_t_vector EstG = {0,0,(int32_t)ACC_1G<<16};
  #if MAG
    static t_int32_t_vector EstM;
  #else
    static t_int32_t_vector EstM = {0,(int32_t)1<<24,0};
  #endif
  static uint32_t LPFAcc[3];
  float invG; // 1/|G|
  static int16_t accZoffset = 0;
  int32_t accZ_tmp=0;
  static uint16_t previousT;
  uint16_t currentT = micros();

  // unit: radian per bit, scaled by 2^16 for further multiplication
  // with a delta time of 3000 us, and GYRO scale of most gyros, scale = a little bit
less than 1
  scale = (currentT - previousT) * (GYRO_SCALE * 65536);
  previousT = currentT;

  // Initialization
  for (axis = 0; axis < 3; axis++) {
    // valid as long as LPF_FACTOR is less than 15
    imu.accSmooth[axis]  = LPFAcc[axis]>>ACC_LPF_FACTOR;
    LPFAcc[axis]      += imu.accADC[axis] - imu.accSmooth[axis];
    // used to calculate later the magnitude of acc vector
    accMag   += mul(imu.accSmooth[axis] , imu.accSmooth[axis]);
    // unit: radian scaled by 2^16
    // imu.gyroADC[axis] is 14 bit long, the scale factor ensure
deltaGyroAngle16[axis] is still 14 bit long
    deltaGyroAngle16[axis] = imu.gyroADC[axis]  * scale;
  }

  // we rotate the intermediate 32 bit vector with the radian vector
(deltaGyroAngle16), scaled by 2^16
  // however, only the first 16 MSB of the 32 bit vector is used to compute the result
  // it is ok to use this approximation as the 16 LSB are used only for the
complementary filter part
  rotateV32(&EstG,deltaGyroAngle16);
  rotateV32(&EstM,deltaGyroAngle16);

  // Apply complimentary filter (Gyro drift correction)
  // If accel magnitude >1.15G or <0.85G and ACC vector outside of the limit range
=> we neutralize the effect of accelerometers in the angle estimation.
  // To do that, we just skip filter, as EstV already rotated by Gyro
```

```
  for (axis = 0; axis < 3; axis++) {
    if ( (int16_t)(0.85*ACC_1G*ACC_1G/256) < (int16_t)(accMag>>8) &&
(int16_t)(accMag>>8) < (int16_t)(1.15*ACC_1G*ACC_1G/256) )
      EstG.A32[axis] += (int32_t)(imu.accSmooth[axis] - EstG.A16[2*axis+1])<<(16-
GYR_CMPF_FACTOR);
    accZ_tmp += mul(imu.accSmooth[axis] , EstG.A16[2*axis+1]);
    #if MAG
      EstM.A32[axis]  += (int32_t)(imu.magADC[axis] - EstM.A16[2*axis+1])<<(16-
GYR_CMPFM_FACTOR);
    #endif
  }

  if (EstG.V16.Z > ACCZ_25deg)
    f.SMALL_ANGLES_25 = 1;
  else
    f.SMALL_ANGLES_25 = 0;

  // Attitude of the estimated vector
  int32_t sqGX_sqGZ = mul(EstG.V16.X,EstG.V16.X) +
mul(EstG.V16.Z,EstG.V16.Z);
  invG = InvSqrt(sqGX_sqGZ + mul(EstG.V16.Y,EstG.V16.Y));
  att.angle[ROLL]  = _atan2(EstG.V16.X , EstG.V16.Z);
  att.angle[PITCH] = _atan2(EstG.V16.Y , InvSqrt(sqGX_sqGZ)*sqGX_sqGZ);

  //note on the second term: mathematically there is a risk of overflow (16*16*16=48
bits). assumed to be null with real values
  att.heading = _atan2(
    mul(EstM.V16.Z , EstG.V16.X) - mul(EstM.V16.X , EstG.V16.Z),
    (EstM.V16.Y * sqGX_sqGZ  - (mul(EstM.V16.X , EstG.V16.X) +
mul(EstM.V16.Z , EstG.V16.Z)) * EstG.V16.Y)*invG );
  #if MAG
    att.heading += conf.mag_declination; // Set from GUI
  #endif
  att.heading /= 10;

  #if defined(THROTTLE_ANGLE_CORRECTION)
    cosZ = mul(EstG.V16.Z , 100) / ACC_1G ;                                    //
cos(angleZ) * 100
    throttleAngleCorrection = THROTTLE_ANGLE_CORRECTION * constrain(100
- cosZ, 0, 100) >>3;  // 16 bit ok: 200*150 = 30000
  #endif

  // projection of ACC vector to global Z, with 1G subtructed
  // Math: accZ = A * G / |G| - 1G
  accZ = accZ_tmp *  invG;
  if (!f.ARMED) {
    accZoffset -= accZoffset>>3;
    accZoffset += accZ;
  }
  accZ -= accZoffset>>3;
```

```
}

#define UPDATE_INTERVAL 25000    // 40hz update rate (20hz LPF on acc)
#define BARO_TAB_SIZE   21

#define ACC_Z_DEADBAND (ACC_1G>>5) // was 40 instead of 32 now


#define applyDeadband(value, deadband)  \
  if(abs(value) < deadband) {           \
    value = 0;                          \
  } else if(value > 0){                 \
    value -= deadband;                  \
  } else if(value < 0){                 \
    value += deadband;                  \
  }

#if BARO
uint8_t getEstimatedAltitude(){
  int32_t  BaroAlt;
  static float baroGroundTemperatureScale,logBaroGroundPressureSum;
  static float vel = 0.0f;
  static uint16_t previousT;
  uint16_t currentT = micros();
  uint16_t dTime;

  dTime = currentT - previousT;
  if (dTime < UPDATE_INTERVAL) return 0;
  previousT = currentT;

  if(calibratingB > 0) {
    logBaroGroundPressureSum = log(baroPressureSum);
    baroGroundTemperatureScale = ((int32_t)baroTemperature + 27315) * (2 *
29.271267f); // 2 *  is included here => no need for * 2  on BaroAlt in additional LPF
    calibratingB--;
  }

  // baroGroundPressureSum is not supposed to be 0 here
  // see: https://code.google.com/p/ardupilot-
mega/source/browse/libraries/AP_Baro/AP_Baro.cpp
  BaroAlt = ( logBaroGroundPressureSum - log(baroPressureSum) ) *
baroGroundTemperatureScale;

  alt.EstAlt = (alt.EstAlt * 6 + BaroAlt ) >> 3; // additional LPF to reduce baro noise
(faster by 30 µs)
  #if (defined(VARIOMETER) && (VARIOMETER != 2)) ||
!defined(SUPPRESS_BARO_ALTHOLD)
    //P
    int16_t error16 = constrain(AltHold - alt.EstAlt, -300, 300);
    applyDeadband(error16, 10); //remove small P parametr to reduce noise near zero
```

```
position
   BaroPID = constrain((conf.pid[PIDALT].P8 * error16 >>7), -150, +150);

   //I
   errorAltitudeI += conf.pid[PIDALT].I8 * error16 >>6;
   errorAltitudeI = constrain(errorAltitudeI,-30000,30000);
   BaroPID += errorAltitudeI>>9; //I in range +/-60

   applyDeadband(accZ, ACC_Z_DEADBAND);

   static int32_t lastBaroAlt;
   // could only overflow with a difference of 320m, which is highly improbable here
   int16_t baroVel = mul((alt.EstAlt - lastBaroAlt) , (1000000 /
UPDATE_INTERVAL));

   lastBaroAlt = alt.EstAlt;

   baroVel = constrain(baroVel, -300, 300); // constrain baro velocity +/- 300cm/s
   applyDeadband(baroVel, 10); // to reduce noise near zero

   // Integrator - velocity, cm/sec
   vel += accZ * ACC_VelScale * dTime;

   // apply Complimentary Filter to keep the calculated velocity based on baro
velocity (i.e. near real velocity).
   // By using CF it's possible to correct the drift of integrated accZ (velocity) without
loosing the phase, i.e without delay
   vel = vel * 0.985f + baroVel * 0.015f;

   //D
   alt.vario = vel;
   applyDeadband(alt.vario, 5);
   BaroPID -= constrain(conf.pid[PIDALT].D8 * alt.vario >>4, -150, 150);
 #endif
 return 1;
}
#endif //BARO
```

καρτέλα IMU.h

```
#ifndef IMU_H_
#define IMU_H_

#define BARO_TAB_SIZE   21

#if BARO
uint8_t getEstimatedAltitude();
#endif

void computeIMU();
int32_t mul(int16_t a, int16_t b);
```

```
#endif /* IMU_H_ */
```
καρτέλα LCD.cpp
```
#include "Arduino.h"
#include "config.h"
#include "def.h"
#include "types.h"
#include "MultiWii.h"
#include "Alarms.h"
#include "EEPROM.h"
#include "Output.h"
#include "RX.h"
#include "Serial.h"
#include "Sensors.h"
#include "LCD.h"

void __u8Inc(void * var, int16_t inc);
void __s8Inc(void * var, int16_t inc);
void __u16Inc(void * var, int16_t inc);
void __s16Inc(void * var, int16_t inc);
void __nullInc(void * var, int16_t inc);
void __u8Fmt(void * var, uint8_t mul, uint8_t dec);
void __u16Fmt(void * var, uint8_t mul, uint8_t dec);
void __s8BitsFmt(void * var, uint8_t mul, uint8_t dec);
void __s16Fmt(void * var, uint8_t mul, uint8_t dec);
void __uAuxFmt(void * var, uint8_t mul, uint8_t dec, uint8_t aux);
void __uAuxFmt1(void * var, uint8_t mul, uint8_t dec);
void __uAuxFmt2(void * var, uint8_t mul, uint8_t dec);
void __uAuxFmt3(void * var, uint8_t mul, uint8_t dec);
void __uAuxFmt4(void * var, uint8_t mul, uint8_t dec);
void __upMFmt(void * var, uint8_t mul, uint8_t dec);

void serviceCheckPLog(void);
void i2c_clear_OLED(void);
void LCDnextline(void);
void i2c_OLED_DIGOLE_send_string(const char *string);

//
**********************************************************************
************************************
// LCD & display & monitoring
//
**********************************************************************
************************************
// in any of the following cases an LCD is required and
// the primitives for exactly one of the available types are setup
#if defined(LCD_CONF) || defined(LCD_TELEMETRY) || defined(HAS_LCD)
static char line1[17],line2[17];
static char template7[7] = " .... ";
static char template3[3] = ". ";
```

```
#ifdef DISPLAY_FONT_DSIZE
  static uint8_t line_is_valid = 0;
#endif
#if ( defined(LOG_PERMANENT) && defined(DISPLAY_MULTILINE) )
  static uint8_t lnr = 0;
#endif

#define LCD_FLUSH {/*UartSendData();*/ delay(30); }

char digit10000(uint16_t v) {return '0' + v / 10000;}
char digit1000(uint16_t v) {return '0' + v / 1000 - (v/10000) * 10;}
char digit100(uint16_t v) {return '0' + v / 100 - (v/1000) * 10;}
char digit10(uint16_t v) {return '0' + v / 10 - (v/100) * 10;}
char digit1(uint16_t v) {return '0' + v - (v/10) * 10;}

#if defined(OLED_I2C_128x64)
// ########################################
// #  i2c OLED display funtion primitives #
// ########################################
#define OLED_address   0x3C     // OLED at address 0x3C in 7bit
char LINE_FILL_STRING[] = "                     "; // Used by clear_OLED() 128 bits / 6
bytes = 21 chars per row
unsigned char CHAR_FORMAT = 0;      // use to INVERSE characters
// use INVERSE    CHAR_FORMAT = 0b01111111;
// use NORMAL     CHAR_FORMAT = 0;
static char buffer; // buffer to read bytes from ROM, using pgm_read_byte macro.
NB! avr/pgmspace.h must be included prog_uchar LOGO[] PROGMEM = {  // My
first attempt to flash a logo....
const uint8_t PROGMEM LOGO[] = { // logo....
   0x00, 0x00, 0x02, 0xFE, 0xFE, 0x0E, 0xFC, 0xF8, 0xC0, 0x00, 0xC0, 0xF8, 0xFC,
0x0E, 0xFE, 0xFE,
   0xFE, 0x02, 0x00, 0x00, 0x30, 0xF0, 0xF0, 0x00, 0x00, 0x00, 0x30, 0xF0, 0xF0,
0x00, 0x00, 0x00,
   0x02, 0xFE, 0xFE, 0x00, 0x00, 0x00, 0x30, 0xF8, 0xFE, 0x30, 0x30, 0x30, 0x00,
0x00, 0x30, 0xF6,
   0xF6, 0x00, 0x00, 0x00, 0x02, 0x06, 0x1E, 0xFE, 0xFE, 0xC2, 0x00, 0xC2, 0xFE,
0x7E, 0xFE, 0xC2,
   0x00, 0xC2, 0xFE, 0xFE, 0x3E, 0x06, 0x02, 0x00, 0x30, 0xF6, 0xF6, 0x00, 0x00,
0x00, 0x00, 0x30,
   0xF6, 0xF6, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
   0x00, 0x00, 0x10, 0x1F, 0x1F, 0x10, 0x00, 0x83, 0x9F, 0x9F, 0x9F, 0x83, 0x80,
0x90, 0x9F, 0x9F,
   0x9F, 0x10, 0x00, 0x00, 0x00, 0x0F, 0x1F, 0x18, 0x18, 0x18, 0x0C, 0x1F, 0x1F,
0x10, 0x00, 0x00,
```

0x10, 0x1F, 0x1F, 0x10, 0x00, 0x80, 0x80, 0x8F, 0x9F, 0x98, 0x9E, 0x8F, 0x80, 0x80, 0x90, 0x1F,
0x1F, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x1F, 0x1E, 0x1F, 0x03, 0x00, 0x07, 0x1F,
0x1E, 0x1F, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x1F, 0x1F, 0x10, 0x00, 0x00, 0x00, 0x10,
0x1F, 0x1F, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0xC0, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00,
0x00, 0xC0, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xE0, 0xF8, 0x3C, 0x0E, 0x07, 0x03, 0x03, 0x01, 0x81, 0xC1, 0xC1, 0xC1, 0xC1, 0x81, 0x01,
0x03, 0x03, 0x07, 0x0E, 0x3C, 0xF8, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE0,
0xF8, 0x3C, 0x0E, 0x07, 0x03, 0x03, 0x01, 0x81, 0xC1, 0xC1, 0xC1, 0xC1, 0x81, 0x01, 0x03, 0x03,
0x07, 0x0E, 0x3C, 0xF8, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x3E, 0xF8, 0xC0,
0xFC, 0x0E, 0xFC, 0xC0, 0xF8, 0x3E, 0x06, 0x00, 0xFE, 0xFE, 0x00, 0x06, 0xFF, 0xFF, 0x86, 0x86,
0x00, 0xFF, 0xFF, 0x0C, 0x06, 0x06, 0xFE, 0xFC, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x1F, 0x7F, 0xF0, 0xC0, 0x80, 0x00, 0x00, 0x00, 0x07, 0x0F, 0x0C, 0x0C, 0x0F, 0x07, 0x00,
0x00, 0x00, 0x80, 0xC0, 0xF0, 0x7F, 0x1F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1F,
0x7F, 0xF0, 0xC0, 0x80, 0x00, 0x00, 0x00, 0x07, 0x0F, 0x0C, 0x0C, 0x0F, 0x07, 0x00, 0x00, 0x00,
0x80, 0xC0, 0xF0, 0x7F, 0x1F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0xC0, 0xC0, 0xC0, 0xC1, 0xC1,
0x80, 0x80, 0x00, 0x01, 0x01, 0x00, 0x00, 0xC0, 0xC1, 0xC1, 0xC0, 0xC0, 0xC0, 0xC1, 0xC1, 0xC1,
0x80, 0x01, 0x01, 0x00, 0x00, 0x00, 0x81, 0x81, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0x80, 0x80, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x03, 0x07, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
0x07, 0x03, 0x03, 0x01, 0x03, 0x07, 0x0E, 0x1C, 0x38, 0xF0, 0xE0, 0xE0, 0xF0, 0x38, 0x1C, 0x0E,
0x07, 0x03, 0x01, 0x03, 0x03, 0x07, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x07, 0x03,

0x03, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFC, 0xFE, 0xFF, 0x07, 0x03, 0x01, 0x01, 0xE1, 0xE1, 0xE1,
0xE3, 0xE7, 0xE7, 0xE6, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0x71, 0x71, 0x71, 0x71, 0x7B, 0x7F,
0x3F, 0x1F, 0x00, 0x00, 0x00, 0x0F, 0x1F, 0x3F, 0x39, 0x71, 0x71, 0x71, 0xE3, 0xE7, 0xE7, 0x86,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x80, 0xC0, 0xE0, 0x70, 0x30, 0x38, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x38, 0x30, 0x70, 0xE0, 0xF0, 0xB8, 0x1C, 0x0E, 0x07, 0x03, 0x01, 0x01, 0x03, 0x07, 0x0E, 0x1C,
0xB8, 0xF0, 0xE0, 0x70, 0x30, 0x38, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x38, 0x30,
0x70, 0xE0, 0xC0, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x07, 0x0F, 0x0E, 0x1C, 0x1C, 0x1C, 0x1C, 0x1C,
0x0E, 0x0F, 0x0F, 0x07, 0x00, 0x00, 0x00, 0x1F, 0x1F, 0x1F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x03, 0x0F, 0x0F, 0x1E, 0x1C, 0x1C, 0x1C, 0x1C, 0x1E, 0x0F, 0x0F, 0x07,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xFE, 0xFF, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x78, 0xFC, 0xCC, 0xCC, 0xFC, 0x78, 0x00,
0x00, 0x00, 0x00, 0x00, 0x03, 0xFF, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFE,
0xFF, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x78, 0xFC, 0xCC, 0xCC, 0xFC, 0x78, 0x00, 0x00, 0x00,
0x00, 0x00, 0x03, 0xFF, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFC, 0x04,
0x04, 0x04, 0x04, 0x04, 0x04, 0x08, 0x10, 0xE0, 0x00, 0x00, 0x80, 0x40, 0x20, 0x20, 0x20, 0x40,
0x80, 0x00, 0x60, 0x80, 0x00, 0x00, 0x00, 0x80, 0x60, 0x00, 0xF0, 0x08, 0x04, 0x04, 0x04, 0x08,
0xF0, 0x00, 0x00, 0xE0, 0x5C, 0x44, 0x44, 0x44, 0x84, 0x04, 0x00, 0x00, 0x10, 0x08, 0x04, 0x04,
0x04, 0x8C, 0x70, 0x00, 0x00, 0x30, 0x48, 0x84, 0x84, 0x84, 0x48, 0x30, 0x00, 0x00, 0x00, 0x00,
0x00, 0x01, 0x07, 0x0F, 0x1C, 0x38, 0x30, 0x70, 0x60, 0x60, 0x60, 0x60, 0x60, 0x60, 0x60, 0x60,
0x70, 0x30, 0x38, 0x1C, 0x0F, 0x07, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
0x07, 0x0F, 0x1C, 0x38, 0x30, 0x70, 0x60, 0x60, 0x60, 0x60, 0x60, 0x60, 0x60, 0x70, 0x30,
0x38, 0x1C, 0x0F, 0x07, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3F, 0x20,

```
   0x20, 0x20, 0x20, 0x20, 0x20, 0x10, 0x08, 0x07, 0x00, 0x00, 0x0F, 0x12, 0x22,
0x22, 0x22, 0x12,
   0x0B, 0x00, 0x00, 0x01, 0x0E, 0x30, 0x0E, 0x01, 0x00, 0x00, 0x0F, 0x10, 0x20,
0x20, 0x20, 0x10,
   0x0F, 0x00, 0x00, 0x08, 0x10, 0x20, 0x20, 0x20, 0x10, 0x0F, 0x00, 0x00, 0x20,
0x30, 0x28, 0x24,
   0x22, 0x21, 0x20, 0x00, 0x00, 0x0E, 0x11, 0x20, 0x20, 0x20, 0x11, 0x0E, 0x00,
0x00, 0x00, 0x00

};

const uint8_t PROGMEM myFont[][5] = { // Refer to "Times New Roman" Font
Database... 5 x 7 font
  { 0x00,0x00,0x00,0x00,0x00},
  { 0x00,0x00,0x4F,0x00,0x00}, //   (  1)  ! - 0x0021 Exclamation Mark
  { 0x00,0x07,0x00,0x07,0x00}, //   (  2)  " - 0x0022 Quotation Mark
  { 0x14,0x7F,0x14,0x7F,0x14}, //   (  3)  # - 0x0023 Number Sign
  { 0x24,0x2A,0x7F,0x2A,0x12}, //   (  4)  $ - 0x0024 Dollar Sign
  { 0x23,0x13,0x08,0x64,0x62}, //   (  5)  % - 0x0025 Percent Sign
  { 0x36,0x49,0x55,0x22,0x50}, //   (  6)  & - 0x0026 Ampersand
  { 0x00,0x05,0x03,0x00,0x00}, //   (  7)  ' - 0x0027 Apostrophe
  { 0x00,0x1C,0x22,0x41,0x00}, //   (  8)  ( - 0x0028 Left Parenthesis
  { 0x00,0x41,0x22,0x1C,0x00}, //   (  9)  ) - 0x0029 Right Parenthesis
  { 0x14,0x08,0x3E,0x08,0x14}, //   ( 10)  * - 0x002A Asterisk
  { 0x08,0x08,0x3E,0x08,0x08}, //   ( 11)  + - 0x002B Plus Sign
  { 0x00,0x50,0x30,0x00,0x00}, //   ( 12)  , - 0x002C Comma
  { 0x08,0x08,0x08,0x08,0x08}, //   ( 13)  - - 0x002D Hyphen-Minus
  { 0x00,0x60,0x60,0x00,0x00}, //   ( 14)  . - 0x002E Full Stop
  { 0x20,0x10,0x08,0x04,0x02}, //   ( 15)  / - 0x002F Solidus
  { 0x3E,0x51,0x49,0x45,0x3E}, //   ( 16)  0 - 0x0030 Digit Zero
  { 0x00,0x42,0x7F,0x40,0x00}, //   ( 17)  1 - 0x0031 Digit One
  { 0x42,0x61,0x51,0x49,0x46}, //   ( 18)  2 - 0x0032 Digit Two
  { 0x21,0x41,0x45,0x4B,0x31}, //   ( 19)  3 - 0x0033 Digit Three
  { 0x18,0x14,0x12,0x7F,0x10}, //   ( 20)  4 - 0x0034 Digit Four
  { 0x27,0x45,0x45,0x45,0x39}, //   ( 21)  5 - 0x0035 Digit Five
  { 0x3C,0x4A,0x49,0x49,0x30}, //   ( 22)  6 - 0x0036 Digit Six
  { 0x01,0x71,0x09,0x05,0x03}, //   ( 23)  7 - 0x0037 Digit Seven
  { 0x36,0x49,0x49,0x49,0x36}, //   ( 24)  8 - 0x0038 Digit Eight
  { 0x06,0x49,0x49,0x29,0x1E}, //   ( 25)  9 - 0x0039 Dight Nine
  { 0x00,0x36,0x36,0x00,0x00}, //   ( 26)  : - 0x003A Colon
  { 0x00,0x56,0x36,0x00,0x00}, //   ( 27)  ; - 0x003B Semicolon
  { 0x08,0x14,0x22,0x41,0x00}, //   ( 28)  < - 0x003C Less-Than Sign
  { 0x14,0x14,0x14,0x14,0x14}, //   ( 29)  = - 0x003D Equals Sign
  { 0x00,0x41,0x22,0x14,0x08}, //   ( 30)  > - 0x003E Greater-Than Sign
  { 0x02,0x01,0x51,0x09,0x06}, //   ( 31)  ? - 0x003F Question Mark
  { 0x32,0x49,0x79,0x41,0x3E}, //   ( 32)  @ - 0x0040 Commercial At
  { 0x7E,0x11,0x11,0x11,0x7E}, //   ( 33)  A - 0x0041 Latin Capital Letter A
  { 0x7F,0x49,0x49,0x49,0x36}, //   ( 34)  B - 0x0042 Latin Capital Letter B
  { 0x3E,0x41,0x41,0x41,0x22}, //   ( 35)  C - 0x0043 Latin Capital Letter C
  { 0x7F,0x41,0x41,0x22,0x1C}, //   ( 36)  D - 0x0044 Latin Capital Letter D
```

```
{ 0x7F,0x49,0x49,0x49,0x41}, //  ( 37)  E - 0x0045 Latin Capital Letter E
{ 0x7F,0x09,0x09,0x09,0x01}, //  ( 38)  F - 0x0046 Latin Capital Letter F
{ 0x3E,0x41,0x49,0x49,0x7A}, //  ( 39)  G - 0x0047 Latin Capital Letter G
{ 0x7F,0x08,0x08,0x08,0x7F}, //  ( 40)  H - 0x0048 Latin Capital Letter H
{ 0x00,0x41,0x7F,0x41,0x00}, //  ( 41)  I - 0x0049 Latin Capital Letter I
{ 0x20,0x40,0x41,0x3F,0x01}, //  ( 42)  J - 0x004A Latin Capital Letter J
{ 0x7F,0x08,0x14,0x22,0x41}, //  ( 43)  K - 0x004B Latin Capital Letter K
{ 0x7F,0x40,0x40,0x40,0x40}, //  ( 44)  L - 0x004C Latin Capital Letter L
{ 0x7F,0x02,0x0C,0x02,0x7F}, //  ( 45)  M - 0x004D Latin Capital Letter M
{ 0x7F,0x04,0x08,0x10,0x7F}, //  ( 46)  N - 0x004E Latin Capital Letter N
{ 0x3E,0x41,0x41,0x41,0x3E}, //  ( 47)  O - 0x004F Latin Capital Letter O
{ 0x7F,0x09,0x09,0x09,0x06}, //  ( 48)  P - 0x0050 Latin Capital Letter P
{ 0x3E,0x41,0x51,0x21,0x5E}, //  ( 49)  Q - 0x0051 Latin Capital Letter Q
{ 0x7F,0x09,0x19,0x29,0x46}, //  ( 50)  R - 0x0052 Latin Capital Letter R
{ 0x46,0x49,0x49,0x49,0x31}, //  ( 51)  S - 0x0053 Latin Capital Letter S
{ 0x01,0x01,0x7F,0x01,0x01}, //  ( 52)  T - 0x0054 Latin Capital Letter T
{ 0x3F,0x40,0x40,0x40,0x3F}, //  ( 53)  U - 0x0055 Latin Capital Letter U
{ 0x1F,0x20,0x40,0x20,0x1F}, //  ( 54)  V - 0x0056 Latin Capital Letter V
{ 0x3F,0x40,0x38,0x40,0x3F}, //  ( 55)  W - 0x0057 Latin Capital Letter W
{ 0x63,0x14,0x08,0x14,0x63}, //  ( 56)  X - 0x0058 Latin Capital Letter X
{ 0x07,0x08,0x70,0x08,0x07}, //  ( 57)  Y - 0x0059 Latin Capital Letter Y
{ 0x61,0x51,0x49,0x45,0x43}, //  ( 58)  Z - 0x005A Latin Capital Letter Z
{ 0x00,0x7F,0x41,0x41,0x00}, //  ( 59)  [ - 0x005B Left Square Bracket
{ 0x02,0x04,0x08,0x10,0x20}, //  ( 60)  \ - 0x005C Reverse Solidus
{ 0x00,0x41,0x41,0x7F,0x00}, //  ( 61)  ] - 0x005D Right Square Bracket
{ 0x04,0x02,0x01,0x02,0x04}, //  ( 62)  ^ - 0x005E Circumflex Accent
{ 0x40,0x40,0x40,0x40,0x40}, //  ( 63)  _ - 0x005F Low Line
{ 0x01,0x02,0x04,0x00,0x00}, //  ( 64)  ` - 0x0060 Grave Accent
{ 0x20,0x54,0x54,0x54,0x78}, //  ( 65)  a - 0x0061 Latin Small Letter A
{ 0x7F,0x48,0x44,0x44,0x38}, //  ( 66)  b - 0x0062 Latin Small Letter B
{ 0x38,0x44,0x44,0x44,0x20}, //  ( 67)  c - 0x0063 Latin Small Letter C
{ 0x38,0x44,0x44,0x48,0x7F}, //  ( 68)  d - 0x0064 Latin Small Letter D
{ 0x38,0x54,0x54,0x54,0x18}, //  ( 69)  e - 0x0065 Latin Small Letter E
{ 0x08,0x7E,0x09,0x01,0x02}, //  ( 70)  f - 0x0066 Latin Small Letter F
{ 0x06,0x49,0x49,0x49,0x3F}, //  ( 71)  g - 0x0067 Latin Small Letter G
{ 0x7F,0x08,0x04,0x04,0x78}, //  ( 72)  h - 0x0068 Latin Small Letter H
{ 0x00,0x44,0x7D,0x40,0x00}, //  ( 73)  i - 0x0069 Latin Small Letter I
{ 0x20,0x40,0x44,0x3D,0x00}, //  ( 74)  j - 0x006A Latin Small Letter J
{ 0x7F,0x10,0x28,0x44,0x00}, //  ( 75)  k - 0x006B Latin Small Letter K
{ 0x00,0x41,0x7F,0x40,0x00}, //  ( 76)  l - 0x006C Latin Small Letter L
{ 0x7C,0x04,0x18,0x04,0x7C}, //  ( 77)  m - 0x006D Latin Small Letter M
{ 0x7C,0x08,0x04,0x04,0x78}, //  ( 78)  n - 0x006E Latin Small Letter N
{ 0x38,0x44,0x44,0x44,0x38}, //  ( 79)  o - 0x006F Latin Small Letter O
{ 0x7C,0x14,0x14,0x14,0x08}, //  ( 80)  p - 0x0070 Latin Small Letter P
{ 0x08,0x14,0x14,0x18,0x7C}, //  ( 81)  q - 0x0071 Latin Small Letter Q
{ 0x7C,0x08,0x04,0x04,0x08}, //  ( 82)  r - 0x0072 Latin Small Letter R
{ 0x48,0x54,0x54,0x54,0x20}, //  ( 83)  s - 0x0073 Latin Small Letter S
{ 0x04,0x3F,0x44,0x40,0x20}, //  ( 84)  t - 0x0074 Latin Small Letter T
{ 0x3C,0x40,0x40,0x20,0x7C}, //  ( 85)  u - 0x0075 Latin Small Letter U
{ 0x1C,0x20,0x40,0x20,0x1C}, //  ( 86)  v - 0x0076 Latin Small Letter V
```

```
  { 0x3C,0x40,0x30,0x40,0x3C}, //   ( 87)  w - 0x0077 Latin Small Letter W
  { 0x44,0x28,0x10,0x28,0x44}, //   ( 88)  x - 0x0078 Latin Small Letter X
  { 0x0C,0x50,0x50,0x50,0x3C}, //   ( 89)  y - 0x0079 Latin Small Letter Y
  { 0x44,0x64,0x54,0x4C,0x44}, //   ( 90)  z - 0x007A Latin Small Letter Z
  { 0x00,0x08,0x36,0x41,0x00}, //   ( 91)  { - 0x007B Left Curly Bracket
  { 0x00,0x00,0x7F,0x00,0x00}, //   ( 92)  | - 0x007C Vertical Line
  { 0x00,0x41,0x36,0x08,0x00}, //   ( 93)  } - 0x007D Right Curly Bracket
  { 0x02,0x01,0x02,0x04,0x02}, //   ( 94)  ~ - 0x007E Tilde
  { 0x3E,0x55,0x55,0x41,0x22}, //   ( 95)  C - 0x0080 <Control>
  { 0x00,0x00,0x00,0x00,0x00}, //   ( 96)    - 0x00A0 No-Break Space
  { 0x00,0x00,0x79,0x00,0x00}, //   ( 97)  ! - 0x00A1 Inverted Exclamation Mark
  { 0x18,0x24,0x74,0x2E,0x24}, //   ( 98)  c - 0x00A2 Cent Sign
  { 0x48,0x7E,0x49,0x42,0x40}, //   ( 99)  L - 0x00A3 Pound Sign
  { 0x5D,0x22,0x22,0x22,0x5D}, //   (100)  o - 0x00A4 Currency Sign
  { 0x15,0x16,0x7C,0x16,0x15}, //   (101)  Y - 0x00A5 Yen Sign
  { 0x00,0x00,0x77,0x00,0x00}, //   (102)  | - 0x00A6 Broken Bar
  { 0x0A,0x55,0x55,0x55,0x28}, //   (103)    - 0x00A7 Section Sign
  { 0x00,0x01,0x00,0x01,0x00}, //   (104)  " - 0x00A8 Diaeresis
  { 0x00,0x0A,0x0D,0x0A,0x04}, //   (105)    - 0x00AA Feminine Ordinal Indicator
  { 0x08,0x14,0x2A,0x14,0x22}, //   (106) << - 0x00AB Left-Pointing Double Angle
Quotation Mark
  { 0x04,0x04,0x04,0x04,0x1C}, //   (107)    - 0x00AC Not Sign
  { 0x00,0x08,0x08,0x08,0x00}, //   (108)  - - 0x00AD Soft Hyphen
  { 0x01,0x01,0x01,0x01,0x01}, //   (109)    - 0x00AF Macron
  { 0x00,0x02,0x05,0x02,0x00}, //   (110)    - 0x00B0 Degree Sign
  { 0x44,0x44,0x5F,0x44,0x44}, //   (111) +- - 0x00B1 Plus-Minus Sign
  { 0x00,0x00,0x04,0x02,0x01}, //   (112)  ` - 0x00B4 Acute Accent
  { 0x7E,0x20,0x20,0x10,0x3E}, //   (113)  u - 0x00B5 Micro Sign
  { 0x06,0x0F,0x7F,0x00,0x7F}, //   (114)    - 0x00B6 Pilcrow Sign
  { 0x00,0x18,0x18,0x00,0x00}, //   (115)  . - 0x00B7 Middle Dot
  { 0x00,0x40,0x50,0x20,0x00}, //   (116)    - 0x00B8 Cedilla
  { 0x00,0x0A,0x0D,0x0A,0x00}, //   (117)    - 0x00BA Masculine Ordinal Indicator
  { 0x22,0x14,0x2A,0x14,0x08}, //   (118) >> - 0x00BB Right-Pointing Double Angle
Quotation Mark
  { 0x17,0x08,0x34,0x2A,0x7D}, //   (119) /4 - 0x00BC Vulgar Fraction One Quarter
  { 0x17,0x08,0x04,0x6A,0x59}, //   (120) /2 - 0x00BD Vulgar Fraction One Half
  { 0x30,0x48,0x45,0x40,0x20}, //   (121)  ? - 0x00BE Inverted Question Mark
  { 0x42,0x00,0x42,0x00,0x42}, //   (122)    - 0x00BF Bargraph - 0
  { 0x7E,0x42,0x00,0x42,0x00}, //   (123)    - 0x00BF Bargraph - 1
  { 0x7E,0x7E,0x00,0x42,0x00}, //   (124)    - 0x00BF Bargraph - 2
  { 0x7E,0x7E,0x7E,0x42,0x00}, //   (125)    - 0x00BF Bargraph - 3
  { 0x7E,0x7E,0x7E,0x7E,0x00}, //   (126)    - 0x00BF Bargraph - 4
  { 0x7E,0x7E,0x7E,0x7E,0x7E}, //   (127)    - 0x00BF Bargraph - 5
};


void i2c_OLED_send_cmd(uint8_t command) {
  TWBR = ((F_CPU / 400000L) - 16) / 2; // change the I2C clock rate
  i2c_writeReg(OLED_address, 0x80, (uint8_t)command);
}
```

```
void i2c_OLED_send_byte(uint8_t val) {
  TWBR = ((F_CPU / 400000L) - 16) / 2; // change the I2C clock rate
  i2c_writeReg(OLED_address, 0x40, (uint8_t)val);
}

void  i2c_OLED_init(void){
  i2c_OLED_send_cmd(0xae);    //display off
  i2c_OLED_send_cmd(0xa4);          //SET All pixels OFF
//  i2c_OLED_send_cmd(0xa5);            //SET ALL pixels ON
  delay(50);
  i2c_OLED_send_cmd(0x20);            //Set Memory Addressing Mode
  i2c_OLED_send_cmd(0x02);            //Set Memory Addressing Mode to Page
addressing mode(RESET)
//  i2c_OLED_send_cmd(0xa0);      //colum address 0 mapped to SEG0 (POR)***
wires at bottom
  i2c_OLED_send_cmd(0xa1);    //colum address 127 mapped to SEG0 (POR) **
wires at top of board
//  i2c_OLED_send_cmd(0xC0);            // Scan from Right to Left (POR)        ***
wires at bottom
  i2c_OLED_send_cmd(0xC8);          // Scan from Left to Right              ** wires at
top
  i2c_OLED_send_cmd(0xa6);            // Set WHITE chars on BLACK backround
//  i2c_OLED_send_cmd(0xa7);            // Set BLACK chars on WHITE backround
  i2c_OLED_send_cmd(0x81);            // Setup CONTRAST CONTROL, following
byte is the contrast Value
  i2c_OLED_send_cmd(0xaf);            // contrast value between 1 ( == dull) to 256 (
== bright)
//  i2c_OLED_send_cmd(0xd3);            // Display Offset :
//  i2c_OLED_send_cmd(0x0);            // 0
//  delay(20);
//  i2c_OLED_send_cmd(0x40);            // Display start line [0;63] -> [0x40;0x7f]
//  delay(20);
  #ifdef DISPLAY_FONT_DSIZE
    i2c_OLED_send_cmd(0xd6);            // zoom
    i2c_OLED_send_cmd(0x01);            // on
  #else
//    i2c_OLED_send_cmd(0xd6);            // zoom
//    i2c_OLED_send_cmd(0x00);            // off
  #endif
  delay(20);
  i2c_OLED_send_cmd(0xaf);          //display on
  delay(20);
}

void i2c_OLED_send_char(unsigned char ascii){
  unsigned char i;
  for(i=0;i<5;i++){
    buffer = pgm_read_byte(&(myFont[ascii - 32][i])); // call the macro to read ROM
byte and put it in buffer
```

```
    buffer ^= CHAR_FORMAT;  // apply
    i2c_OLED_send_byte(buffer);
  }
  i2c_OLED_send_byte(CHAR_FORMAT);    // the gap
}

void i2c_OLED_send_string(const char *string){  // Sends a string of chars untill null
terminator
  unsigned char i=0;
  while(*string){
    for(i=0;i<5;i++){
      buffer = pgm_read_byte(&(myFont[(*string)- 32][i])); // call the macro to read the
ROM ASCII table
      buffer ^= CHAR_FORMAT;
      i2c_OLED_send_byte((unsigned char)buffer);
    }
    i2c_OLED_send_byte(CHAR_FORMAT);    // the gap
    *string++;
  }
}

#ifndef SUPPRESS_OLED_I2C_128x64LOGO       // Do we want the Logo
displayed ?
void i2c_OLED_send_logo(void){
  unsigned char i,j;
  i2c_OLED_send_cmd(0xa6);            //Set Normal Display
  i2c_OLED_send_cmd(0xae);      // Display OFF
  i2c_OLED_send_cmd(0x20);            // Set Memory Addressing Mode
  i2c_OLED_send_cmd(0x00);            // Set Memory Addressing Mode to Horizontal
addressing mode
  i2c_OLED_send_cmd(0xb0);            // set page address to 0
  i2c_OLED_send_cmd(0X40);             // Display start line register to 0
  i2c_OLED_send_cmd(0);               // Set low col address to 0
  i2c_OLED_send_cmd(0x10);            // Set high col address to 0
  for(uint16_t k=0; k<1024; k++) {         // fill the display's RAM with graphic...
128*64 pixel picture
    buffer = pgm_read_byte(&(LOGO[k]));
    i2c_OLED_send_byte(buffer);
  }
  i2c_OLED_send_cmd(0x81);            // Setup CONTRAST CONTROL, following
byte is the contrast Value... always a 2 byte instruction
  i2c_OLED_send_cmd(0x0);             // Set contrast value to 0
  i2c_OLED_send_cmd(0xaf);          // display on
  for(j=0; j<2; j++){
    for(i=0x01; i<0xff; i++){
      i2c_OLED_send_cmd(0x81);        // Setup CONTRAST CONTROL, following
byte is the contrast Value
      i2c_OLED_send_cmd(i);           // Set contrast value
      delay(1);
    }
```

```
   for(i=0xff; i>0x01; i--){
     i2c_OLED_send_cmd(0x81);        // Setup CONTRAST CONTROL, following
byte is the contrast Value
     i2c_OLED_send_cmd(i);           // Set contrast value
     delay(1);
   }
  }
  i2c_OLED_init();
  i2c_clear_OLED();
}
#if defined (OLED_I2C_128x64LOGO_PERMANENT)
void i2c_OLED_Put_Logo(void){
  unsigned char i,j;
  i2c_OLED_send_cmd(0xa6);             //Set Normal Display
  i2c_OLED_send_cmd(0xae);             // Display OFF
  i2c_OLED_send_cmd(0x20);             // Set Memory Addressing Mode
  i2c_OLED_send_cmd(0x00);             // Set Memory Addressing Mode to Horizontal
addressing mode
  i2c_OLED_send_cmd(0xb0);             // set page address to 0
  i2c_OLED_send_cmd(0X40);             // Display start line register to 0
  i2c_OLED_send_cmd(0);               // Set low col address to 0
  i2c_OLED_send_cmd(0x10);             // Set high col address to 0
  for(int i=0; i<1024; i++) {          // fill the display's RAM with graphic... 128*64
pixel picture
    buffer = pgm_read_byte(&(LOGO[i]));
    i2c_OLED_send_byte(buffer);
  }
  i2c_OLED_send_cmd(0x81);             // Setup CONTRAST CONTROL, following
byte is the contrast Value... always a 2 byte instruction
  i2c_OLED_send_cmd(250);              // Here you can set the brightness 1 = dull, 255
is very bright
  i2c_OLED_send_cmd(0xaf);             // display on
}
#endif  // OLED_I2C_128x64LOGO_PERMANENT

#endif  // SUPPRESS_OLED_I2C_128x64LOGO

void i2c_OLED_set_XY(byte col, byte row) {        //  Not used in MW V2.0 but its
here anyway!
  i2c_OLED_send_cmd(0xb0+row);              //set page address
  i2c_OLED_send_cmd(0x00+(8*col&0x0f));       //set low col address
  i2c_OLED_send_cmd(0x10+((8*col>>4)&0x0f));  //set high col address
}

void i2c_OLED_set_line(byte row) {   // goto the beginning of a single row,
compattible with LCD_CONFIG
  i2c_OLED_send_cmd(0xb0+row);   //set page address
  i2c_OLED_send_cmd(0);         //set low col address
  i2c_OLED_send_cmd(0x10);       //set high col address
}
```

```
void i2c_clear_OLED(void){
//  unsigned char i;
//  for(i=0;i<8;i++){
//    i2c_OLED_set_XY(0,i);
//    i2c_OLED_send_string(LINE_FILL_STRING);
//  }
  i2c_OLED_send_cmd(0xa6);            //Set Normal Display
  i2c_OLED_send_cmd(0xae);            // Display OFF
  i2c_OLED_send_cmd(0x20);            // Set Memory Addressing Mode
  i2c_OLED_send_cmd(0x00);            // Set Memory Addressing Mode to Horizontal
addressing mode
  i2c_OLED_send_cmd(0xb0);            // set page address to 0
  i2c_OLED_send_cmd(0X40);             // Display start line register to 0
  i2c_OLED_send_cmd(0);               // Set low col address to 0
  i2c_OLED_send_cmd(0x10);            // Set high col address to 0
  for(uint16_t i=0; i<1024; i++) {         // fill the display's RAM with graphic...
128*64 pixel picture
    i2c_OLED_send_byte(0);  // clear
  }
  i2c_OLED_send_cmd(0x81);            // Setup CONTRAST CONTROL, following
byte is the contrast Value... always a 2 byte instruction
  i2c_OLED_send_cmd(200);             // Here you can set the brightness 1 = dull, 255
is very bright
  i2c_OLED_send_cmd(0xaf);            // display on
}

#endif // OLED_I2C_128x64

#if defined(LCD_ETPP)
#define LCD_ETPP_ADDRESS 0x3B

// *********************
// i2c Eagle Tree Power Panel primitives
// *********************
void i2c_ETPP_init () {
  i2c_rep_start(LCD_ETPP_ADDRESS<<1); // LCD_ETPP i2c address: 0x3B in 7 bit
form. Shift left one bit and concatenate i2c write command bit of zero
  i2c_write(0x00);// LCD_ETPP command register
  i2c_write(0x24);// Function Set 001D0MSL D : data length for parallel interface
only; M: 0 = 1x32 , 1 = 2x16; S: 0 = 1:18 multiplex drive mode, 1x32 or 2x16
character display, 1 = 1:9 multiplex drive mode, 1x16 character display; H: 0 = basic
instruction set plus standard instruction set, 1 = basic instruction set plus extended
instruction set
  i2c_write(0x0C);// Display on   00001DCB D : 0 = Display Off, 1 = Display On; C :
0 = Underline Cursor Off, 1 = Underline Cursor On; B : 0 = Blinking Cursor Off, 1 =
Blinking Cursor On
  i2c_write(0x06);// Cursor Move  000001IS I : 0 = DDRAM or CGRAM address
decrements by 1, cursor moves to the left, 1 = DDRAM or CGRAM address
increments by 1, cursor moves to the right; S : 0 = display does not shift,  1 = display
```

```
does shifts
  LCDclear();
}
void i2c_ETPP_send_cmd (byte c) {
  i2c_rep_start(LCD_ETPP_ADDRESS<<1); // I2C write direction
  i2c_write(0x00);// LCD_ETPP command register
  i2c_write(c);
}
void i2c_ETPP_send_char (char c) {
  if (c > 0x0f) c |= 0x80; // LCD_ETPP uses character set "R", which has A->z
mapped same as ascii + high bit; don't mess with custom chars.
  i2c_rep_start(LCD_ETPP_ADDRESS<<1);// I2C write direction
  i2c_write(0x40);// LCD_ETPP data register
  i2c_write(c);
}

void i2c_ETPP_set_cursor (byte addr) {
  i2c_ETPP_send_cmd(0x80 | addr); // High bit is "Set DDRAM" command, remaing
bits are addr.
}
void i2c_ETPP_set_cursor (byte col, byte row) {
  row = min(row,1);
  col = min(col,15);
  byte addr = col + row * 0x40; // Why 0x40? RAM in this controller has many more
bytes than are displayed.  In particular, the start of the second line (line 1 char 0) is
0x40 in DDRAM. The bytes between 0x0F (last char of line 1) and 0x40 are not
displayable (unless the display is placed in marquee scroll mode)
  i2c_ETPP_set_cursor(addr);
}
void i2c_ETPP_create_char (byte idx, uint8_t* array) {
  i2c_ETPP_send_cmd(0x80); // CGRAM and DDRAM share an address register, but
you can't set certain bits with the CGRAM address command.   Use DDRAM address
command to be sure high order address bits are zero.
  i2c_ETPP_send_cmd(0x40 | byte(idx * 8));// Set CGRAM address
  i2c_rep_start(LCD_ETPP_ADDRESS<<1);// I2C write direction
  i2c_write(0x40);// LCD_ETPP data register
  for (byte i = 0; i<8; i++) {i2c_write(*array); array++;}
}

static boolean charsInitialized; // chars for servo signals are initialized
void ETPP_barGraph(byte num, int val) { // num chars in graph; percent as 1 to 100
  if (!charsInitialized) {
    charsInitialized = true;

    static byte bars[8] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x15,};
    static byte less[8] = {0x00, 0x04, 0x0C, 0x1C, 0x0C, 0x04, 0x00, 0x15,};
    static byte grt [8] = {0x00, 0x04, 0x06, 0x07, 0x06, 0x04, 0x00, 0x15,};

    byte pattern = 0x10;
    for (int8_t i = 0; i <= 5; i++) {
```

```
      for (int8_t j = 0; j < 7; j++) {
        bars[j] = pattern;
      }
      i2c_ETPP_create_char(i, bars);
      pattern >>= 1;
    }
    i2c_ETPP_create_char(6, less);
    i2c_ETPP_create_char(7, grt);
  }

  static char bar[16];
  for (int8_t i = 0; i < num; i++) {bar[i] = 5;}

  if (val < -100 || val > 100) {bar[0] = 6; bar[num] = 7;} // invalid
  else if (val < 0) {bar[0] = 6;} // <...
  else if (val >= 100) {bar[3] = 7;} // ...>
  else {bar[val/(100/num)] = (val%(100/num))/5;} // ..|.

  for (int8_t i = 0; i < num; i++) {
    i2c_ETPP_send_char(bar[i]);
  }
}
#endif //LCD_ETPP

#if defined(LCD_LCD03) // LCD_LCD03
#define LCD_LCD03_ADDRESS 0x63
// ********************
// I2C LCD03 primitives
// ********************
void i2c_LCD03_init () {
  i2c_rep_start(LCD_LCD03_ADDRESS<<1); // The LCD03 is located on the I2C
bus at address 0xC6
  i2c_write(0x00);// Command register
  i2c_write(04);// Hide cursor
  i2c_write(12);// Clear screen
  i2c_write(19);// Backlight on
}
void i2c_LCD03_send_cmd (byte c) {
  i2c_rep_start(LCD_LCD03_ADDRESS<<1);
  i2c_write(0x00);
  i2c_write(c);
}
void i2c_LCD03_send_char (char c) {
  i2c_rep_start(LCD_LCD03_ADDRESS<<1);
  i2c_write(0x00);
  i2c_write(c);
}
void i2c_LCD03_set_cursor (byte col, byte row) {
  row = min(row,1);
  col = min(col,15);
```

```
  i2c_LCD03_send_cmd(03); // set cursor (row, column)
  i2c_LCD03_send_cmd(row+1);
  i2c_LCD03_send_cmd(col+1);
}
#endif // LCD_LCD03

#if defined(LCD_LCD03S) // LCD_LCD03S
// ********************
// LCD03S serial primitives
// ********************
void serial_LCD03_init () {
  SerialWrite(LCD_SERIAL_PORT, 0x00 );// Command register
  SerialWrite(LCD_SERIAL_PORT, 04 );// Hide cursor
  SerialWrite(LCD_SERIAL_PORT, 12 );// Clear screen
  SerialWrite(LCD_SERIAL_PORT, 19 );// Backlight on
}
void serial_LCD03_send_cmd (byte c) {
  SerialWrite(LCD_SERIAL_PORT, 0x00 );// Command register
  SerialWrite(LCD_SERIAL_PORT, c );
}
void serial_LCD03_send_char (char c) {
  //SerialWrite(LCD_SERIAL_PORT, 0x00 );// Command register
  SerialWrite(LCD_SERIAL_PORT, c );
}
void serial_LCD03_set_cursor (byte col, byte row) {
  row = min(row,1);
  col = min(col,15);
  serial_LCD03_send_cmd(03); // set cursor (row, column)
  serial_LCD03_send_char(row+1);
  serial_LCD03_send_char(col+1);
}
#endif // LCD_LCD03S

#if defined(OLED_DIGOLE) // OLED_DIGOLE
#define OLED_DIGOLE_ADDRESS 0x27 // 7bit address
// ********************
// I2C DIGOLE primitives
// ********************
void i2c_OLED_DIGOLE_init () {
  i2c_OLED_DIGOLE_send_string("CL");    // clear screen
  //  delayMicroseconds(500);
  //  i2c_OLED_DIGOLE_send_string("BL");  // backlight _
  //  i2c_write(0x01);                    // _ on
  i2c_OLED_DIGOLE_send_string("CT");  // contrast _
  i2c_write(100); // contrast [0;100]
  //  i2c_write('C'); i2c_write('T'); i2c_write(100); // contrast [0;100]
  //i2c_OLED_DIGOLE_send_string("DSS");    // display start screen _
  //i2c_write(0);                       // _ off
  //i2c_OLED_DIGOLE_send_string("CS");    // show cursor _
  //i2c_write(0);                       // _ off
```

```
  //i2c_OLED_DIGOLE_printString("123456789.123456789.");
//    delayMicroseconds(500);
//    i2c_OLED_DIGOLE_send_string("MCD"); // manual command
//    i2c_write(0x81); //ssd1306: set contrast to _
//    i2c_OLED_DIGOLE_send_string("MCD"); // manual data
//    i2c_write(0xFF);  // _ value [0;255]
//    delayMicroseconds(500);
}
void i2c_OLED_DIGOLE_send_byte (byte c) {
  i2c_rep_start(OLED_DIGOLE_ADDRESS<<1);
  i2c_write(0x00);
  i2c_write(c);
}
void i2c_OLED_DIGOLE_send_string(const char *string){  // Sends a string of chars
but not the null terminator
  i2c_rep_start(OLED_DIGOLE_ADDRESS<<1);
  i2c_write(0x00);
  while(*string){
    i2c_write(*string);
    *string++;
  }
  //delayMicroseconds(10);
}
void i2c_OLED_DIGOLE_printString(const char *string){  // prints a string of chars
  //  i2c_rep_start(OLED_DIGOLE_ADDRESS<<1);
  //  i2c_write(0x00);
  //  i2c_write('T');
  //  i2c_write('T');
  i2c_OLED_DIGOLE_send_string("TT");  // type text _
  while(*string){
    i2c_write(*string);
    *string++;
  }
  i2c_write(0x00);
  //delayMicroseconds(10);
}
void i2c_OLED_DIGOLE_printChar(char c){  // prints a single char - should be
printable
  i2c_rep_start(OLED_DIGOLE_ADDRESS<<1);
  i2c_write(0x00);
  i2c_write('T');
  i2c_write('T');
  i2c_write(c);
  i2c_write(0x00);
  //delayMicroseconds(10);
}
void i2c_OLED_DIGOLE_clear(void) {
  i2c_OLED_DIGOLE_send_string("CLSF");    // set font _
  #ifdef DISPLAY_FONT_DSIZE
    i2c_write(0);                       // _ one of 6,10,18,51,120,123,0 - font 0 is 5x16
```

```
linesxrows; font 10 is 7x21
  #else
    i2c_write(10);
  #endif
}
#endif // OLED_DIGOLE
/* ------------------------------------------------------------------ */
void LCDprint(uint8_t i) {
  #ifdef DISPLAY_FONT_DSIZE
   if (! line_is_valid) return;
  #endif
  #if defined(LCD_SERIAL3W)
    // 1000000 / 9600  = 104 microseconds at 9600 baud.
    // we set it below to take some margin with the running interrupts
    #define BITDELAY 102
    LCDPIN_OFF;
    delayMicroseconds(BITDELAY);
    for (uint8_t mask = 0x01; mask; mask <<= 1) {
      if (i & mask) {LCDPIN_ON;} else {LCDPIN_OFF;} // choose bit
      delayMicroseconds(BITDELAY);
    }
    LCDPIN_ON //switch ON digital PIN 0
    delayMicroseconds(BITDELAY);
  #elif defined(LCD_TEXTSTAR) || defined(LCD_VT100) || defined(LCD_TTY)
    SerialWrite(LCD_SERIAL_PORT, i );
  #elif defined(LCD_ETPP)
    i2c_ETPP_send_char(i);
  #elif defined(LCD_LCD03)
    i2c_LCD03_send_char(i);
  #elif defined(LCD_LCD03S)
    serial_LCD03_send_char(i);
  #elif defined(OLED_I2C_128x64)
    i2c_OLED_send_char(i);
  #elif defined(OLED_DIGOLE)
    i2c_OLED_DIGOLE_printChar(i);
  #endif
}

void LCDprintChar(const char *s) {
  #ifdef OLED_DIGOLE
    #ifdef DISPLAY_FONT_DSIZE
      if (! line_is_valid) return;
    #endif
    i2c_OLED_DIGOLE_printString(s);
  #else
    while (*s) {LCDprint(*s++);}
  #endif
}

void LCDcrlf() {
```

```
  #if ( defined(OLED_I2C_128x64)|| defined(LCD_VT100) ||
defined(OLED_DIGOLE) )
    // do nothing - these displays use line positioning
  #else
    LCDprintChar("\r\n");
  #endif
}
void LCDclear() {
  #if defined(LCD_SERIAL3W)

LCDprint(0xFE);LCDprint(0x01);delay(10);LCDprint(0xFE);LCDprint(0x02);delay(
10); // clear screen, cursor line 1, pos 0 for serial LCD Sparkfun - contrib by
flyman777
  #elif defined(LCD_TEXTSTAR)
    LCDprint(0x0c);
  #elif defined(LCD_VT100)
    LCDcrlf();
    LCDprint(0x1B); LCDprint(0x5B); LCDprintChar("2J"); //ED2
    LCDcrlf();
    LCDprint(0x1B); LCDprint(0x5B); LCDprintChar("1;1H");//cursor top left
  #elif defined(LCD_TTY)
    LCDcrlf();
  #elif defined(LCD_ETPP)
    i2c_ETPP_send_cmd(0x01); // Clear display command, which does NOT clear an
Eagle Tree because character set "R" has a '>' at 0x20
    for (byte i = 0; i<80; i++) i2c_ETPP_send_char(' ');// Blanks for all 80 bytes of
RAM in the controller, not just the 2x16 display
  #elif defined(LCD_LCD03)
    i2c_LCD03_send_cmd(12); // clear screen
  #elif defined(LCD_LCD03S)
    serial_LCD03_send_cmd(12); // clear screen
  #elif defined(OLED_I2C_128x64)
    i2c_clear_OLED();
  #elif defined(OLED_DIGOLE)
    i2c_OLED_DIGOLE_clear();
  #endif
  #if ( defined(LOG_PERMANENT) && defined(DISPLAY_MULTILINE) )
    lnr = 0;
  #endif
}

void LCDsetLine(byte line) { // Line = 1 or 2 - vt100 has lines 1-99
  #ifdef DISPLAY_FONT_DSIZE
    if (line >=1 && line <= (MULTILINE_PRE+MULTILINE_POST)) {
      line_is_valid = 1;
    } else {
      line_is_valid = 0;
      return;
    }
  #endif
```

```
  #if defined(LCD_SERIAL3W)
    if (line==1) {LCDprint(0xFE);LCDprint(128);} else
{LCDprint(0xFE);LCDprint(192);}
  #elif defined(LCD_TEXTSTAR)
    LCDcrlf(); LCDprint(0xfe);LCDprint('L');LCDprint(line);
  #elif defined(LCD_VT100)
    #ifndef DEBUG // sanity check for production only. Debug runs with all possible
side effects
      if (line<1 || line>(MULTILINE_PRE+MULTILINE_POST)) line = 1;
    #endif
    LCDcrlf();
    LCDprint(0x1b); LCDprint(0x5b);
    LCDprint( digit10(line) );
    LCDprint( digit1(line) );
    LCDprintChar(";1H"); //pos line 1
    LCDprint(0x1b); LCDprint(0x5b); LCDprintChar("2K");//EL2
  #elif defined(LCD_TTY)
    LCDcrlf();
  #elif defined(LCD_ETPP)
    i2c_ETPP_set_cursor(0,line-1);
  #elif defined(LCD_LCD03)
    i2c_LCD03_set_cursor(0,line-1);
  #elif defined(LCD_LCD03S)
    serial_LCD03_set_cursor(0,line-1);
  #elif defined(OLED_I2C_128x64)
  //  #ifndef DEBUG // sanity check for production only. Debug runs with all possible
side effects
  //    if (line<1 || line>(MULTILINE_PRE+MULTILINE_POST)) line = 1;
  //  #endif
    i2c_OLED_set_line(line-1);
  #elif defined(OLED_DIGOLE)
    i2c_OLED_DIGOLE_send_string("TP");  i2c_write(0);i2c_write(line-1); // first
column is 0, line numbers start with 0
  #endif
}
#if defined(LCD_VT100)
  void LCDattributesBold() {LCDprint(0x1b); LCDprint(0x5b);
LCDprintChar("1m");}
  void LCDattributesReverse() {LCDprint(0x1b); LCDprint(0x5b);
LCDprintChar("7m");}
  void LCDattributesOff() {LCDprint(0x1b); LCDprint(0x5b); LCDprintChar("0m");}
  void LCDalarmAndReverse() {LCDattributesReverse(); if (f.ARMED) {
LCDprint(0x07); }; } // audio for errors only while armed
#elif defined(OLED_I2C_128x64)
  void LCDattributesBold() {/*CHAR_FORMAT = 0b01111111; */}
  void LCDattributesReverse() {CHAR_FORMAT = 0b01111111; }
  void LCDattributesOff() {CHAR_FORMAT = 0; }
  void LCDalarmAndReverse() {LCDattributesReverse(); }
#else
  void LCDattributesBold() {}
```

```
  void LCDattributesReverse() {}
  void LCDattributesOff() {}
  void LCDalarmAndReverse() {}
#endif

void LCDprintInt16(int16_t v) {
  uint16_t unit;
  char line[7]; // = "      ";
  if (v < 0 ) {
    unit = -v;
    line[0] = '-';
  } else {
    unit = v;
    line[0] = ' ';
  }
  line[1] = digit10000(unit);
  line[2] = digit1000(unit);
  line[3] = digit100(unit);
  line[4] = digit10(unit);
  line[5] = digit1(unit);
  line[6] = 0;
  LCDprintChar(line);
}
void lcdprint_uint32(uint32_t v) {
  static char line[14] = "-.---.---.---";
  //                     0 2 4 6 8   12
  line[0]  = '0' + v  / 1000000000;
  line[2]  = '0' + v  / 100000000 - (v/1000000000) * 10;
  line[3]  = '0' + v  / 10000000  - (v/100000000)  * 10;
  line[4]  = '0' + v  / 1000000   - (v/10000000)   * 10;
  line[6]  = '0' + v  / 100000    - (v/1000000)    * 10;
  line[7]  = '0' + v  / 10000     - (v/100000)     * 10;
  line[8]  = '0' + v  / 1000      - (v/10000)      * 10;
  line[10]  = '0' + v  / 100      - (v/1000)       * 10;
  line[11] = '0' + v  / 10        - (v/100)        * 10;
  line[12] = '0' + v              - (v/10)         * 10;
  LCDprintChar(line);
}

void initLCD() {
  blinkLED(20,30,1);
  SET_ALARM_BUZZER(ALRM_FAC_CONFIRM, ALRM_LVL_CONFIRM_1);
  #if defined(LCD_SERIAL3W)
    SerialEnd(0);
    //init LCD
    PINMODE_LCD;//TX PIN for LCD = Arduino RX PIN (more convenient to
connect a servo plug on arduino pro mini)
  #elif defined(LCD_TEXTSTAR)
    // Cat's Whisker Technologies 'TextStar' Module CW-LCD-02
    // http://cats-whisker.com/resources/documents/cw-lcd-02_datasheet.pdf
```

```
  //LCDprint(0xFE);LCDprint(0x43);LCDprint(0x02); //cursor blink mode
  LCDprint(0xFE);LCDprint('R');//reset
#elif defined(LCD_VT100)
  //LCDprint(0x1b); LCDprint('c'); //RIS
#elif defined(LCD_TTY)
  ; // do nothing special to init the serial tty device
#elif defined(LCD_ETPP)
  // Eagle Tree Power Panel - I2C & Daylight Readable LCD
  i2c_ETPP_init();
#elif defined(LCD_LCD03)
  // LCD03 - I2C LCD
  // http://www.robot-electronics.co.uk/htm/Lcd03tech.htm
  i2c_LCD03_init();
#elif defined(LCD_LCD03S)
  // LCD03 - SERIAL LCD
  // http://www.robot-electronics.co.uk/htm/Lcd03tech.htm
  serial_LCD03_init();
#elif defined(OLED_I2C_128x64)
  i2c_OLED_init();
  #ifndef SUPPRESS_OLED_I2C_128x64LOGO
    i2c_OLED_send_logo();
    #if defined (OLED_I2C_128x64LOGO_PERMANENT)
      i2c_OLED_Put_Logo();
    #endif
  #endif
#elif defined(OLED_DIGOLE)
  i2c_OLED_DIGOLE_init();
#endif
#ifndef OLED_I2C_128x64LOGO_PERMANENT
  LCDclear();
  strcpy_P(line1,PSTR( BOARD_NAME )); // user defined macro
//                  0123456789.123456
  line1[12] = digit100(VERSION);
  line1[14] = digit10(VERSION);
  line1[15] = digit1(VERSION);
  LCDattributesBold();
  LCDsetLine(1); LCDprintChar(line1);
  strcpy_P(line2,PSTR("  Unknown Modell"));
  #if defined(TRI)
    strcpy_P(line2,PSTR("  TRICopter"));
  #elif defined(QUADP)
    strcpy_P(line2,PSTR("  QUAD-P"));
  #elif defined(QUADX)
    strcpy_P(line2,PSTR("  QUAD-X"));
  #elif defined(BI)
    strcpy_P(line2,PSTR("  BICopter"));
  #elif defined(Y6)
    strcpy_P(line2,PSTR("  Y6"));
  #elif defined(HEX6)
    strcpy_P(line2,PSTR("  HEX6"));
```

```
    #elif defined(FLYING_WING)
      strcpy_P(line2,PSTR("  FLYING_WING"));
    #elif defined(Y4)
      strcpy_P(line2,PSTR("  Y4"));
    #elif defined(HEX6X)
      strcpy_P(line2,PSTR("  HEX6-X"));
    #elif defined(HEX6H)
      strcpy_P(line2,PSTR("  HEX6-H"));
    #elif defined(OCTOX8)
      strcpy_P(line2,PSTR("  OCTOX8"));
    #elif defined(OCTOFLATP)
      strcpy_P(line2,PSTR("  OCTOFLAT-P"));
    #elif defined(OCTOFLATX)
      strcpy_P(line2,PSTR("  OCTOFLAT-X"));
    #elif defined (AIRPLANE)
      strcpy_P(line2,PSTR("  AIRPLANE"));
    #elif defined (HELI_120_CCPM)
      strcpy_P(line2,PSTR("  HELI_120_CCPM"));
    #elif defined (HELI_90_DEG)
      strcpy_P(line2,PSTR("  HELI_90_DEG"));
    #elif defined(VTAIL4)
      strcpy_P(line2,PSTR("  VTAIL Quad"));
    #endif
    //LCDattributesBold();
    LCDsetLine(2); LCDprintChar(line2);
    LCDattributesOff();
    delay(2500);
  #endif // OLED_I2C_128x64LOGO_PERMANENT
  #if defined(LCD_TEXTSTAR) || defined(LCD_VT100)
    //delay(2500);
    LCDclear();
  #endif
  #if defined(OLED_I2C_128x64) &&
!(defined(OLED_I2C_128x64LOGO_PERMANENT)) &&
defined(NEW_OLED_FONT) && !(defined(LCD_TELEMETRY))
    // no need to diplay this, if LCD telemetry is enabled
    //   optional instruction on the display.....
    LCDsetLine(4); LCDprintChar("To ENTER CONFIG      ");// 21 characters on
each line
    LCDsetLine(5); LCDprintChar("YAW RIGHT & PITCH FWD");
    LCDsetLine(7); LCDprintChar("To SAVE CONFIG       ");
    LCDsetLine(8); LCDprintChar("YAW LEFT & PITCH FWD ");
  #endif
  //  if (cycleTime == 0) { //Called from Setup()
  //    strcpy_P(line1,PSTR("Ready to Fly")); LCDsetLine(2); LCDprintChar(line1);
  //  } else {
  //    strcpy_P(line1,PSTR("Config All Parms")); LCDsetLine(2);
LCDprintChar(line1);
  //  }
  #ifdef LCD_TELEMETRY_STEP
```

```
    // load the first page of the step sequence
    LCDclear();
    telemetry = telemetryStepSequence[telemetryStepIndex]; //[++telemetryStepIndex
% strlen(telemetryStepSequence)];
  #endif
}
#endif //Support functions for LCD_CONF and LCD_TELEMETRY

// -------------------- configuration menu to LCD over serial/i2c ----------------------------
------

#ifdef LCD_CONF
static uint8_t reset_to_defaults;

typedef void (*formatter_func_ptr)(void *, uint8_t, uint8_t);
typedef void (*inc_func_ptr)(void *, int16_t);

/*typedef*/struct lcd_type_desc_t {
  formatter_func_ptr fmt;
  inc_func_ptr inc;
};

static lcd_type_desc_t LTU8 = {&__u8Fmt, &__u8Inc};
static lcd_type_desc_t LTS8 = {&__s8BitsFmt, &__s8Inc};
static lcd_type_desc_t LTU16 = {&__u16Fmt, &__u16Inc};
static lcd_type_desc_t LTS16 = {&__s16Fmt, &__s16Inc};
static lcd_type_desc_t LPMM = {&__upMFmt, &__nullInc};
//static lcd_type_desc_t LPMS = {&__upSFmt, &__nullInc};
static lcd_type_desc_t LAUX1 = {&__uAuxFmt1, &__u16Inc};
static lcd_type_desc_t LAUX2 = {&__uAuxFmt2, &__u16Inc};
static lcd_type_desc_t LAUX3 = {&__uAuxFmt3, &__u16Inc};
static lcd_type_desc_t LAUX4 = {&__uAuxFmt4, &__u16Inc};

/*typedef*/struct lcd_param_def_t {
  lcd_type_desc_t * type;
  uint8_t decimal;
  uint8_t multiplier;
  uint16_t increment;
};

//typedef struct lcd_param_t{
//  char*  paramText;
//  void * var;
//  lcd_param_def_t * def;
//};

//
**********************************************************************
************************************
// LCD Layout definition
```

```
//
************************************************************************
************************************
// Descriptors
static lcd_param_def_t __P = {&LTU8, 1, 1, 1};
static lcd_param_def_t __I = {&LTU8, 3, 1, 1};
static lcd_param_def_t __D = {&LTU8, 0, 1, 1};
static lcd_param_def_t __RC = {&LTU8, 2, 1, 1};
static lcd_param_def_t __PM = {&LPMM, 1, 1, 0};
//static lcd_param_def_t __PS = {&LPMS, 1, 1, 0};
static lcd_param_def_t __PT = {&LTU8, 0, 1, 1};
static lcd_param_def_t __VB = {&LTU8, 1, 1, 0};
static lcd_param_def_t __L = {&LTU8, 0, 1, 0};
static lcd_param_def_t __FS = {&LTU8, 1, 1, 0};
static lcd_param_def_t __SE = {&LTU16, 0, 1, 10};
static lcd_param_def_t __SE1 = {&LTU16, 0, 1, 1};
static lcd_param_def_t __ST = {&LTS16, 0, 1, 10};
static lcd_param_def_t __AUX1 = {&LAUX1, 0, 1, 1};
static lcd_param_def_t __AUX2 = {&LAUX2, 0, 1, 8};
static lcd_param_def_t __AUX3 = {&LAUX3, 0, 1, 64};
static lcd_param_def_t __AUX4 = {&LAUX4, 0, 1, 512};
static lcd_param_def_t __BITS = {&LTS8, 0, 1, 1};

// Program Space Strings - These sit in program flash, not SRAM.
//                              0123456789
const char PROGMEM lcd_param_text01 [] = "Pit&Roll P";
const char PROGMEM lcd_param_text02 [] = "Roll     P";
const char PROGMEM lcd_param_text03 [] = "Roll     I";
const char PROGMEM lcd_param_text04 [] = "Roll     D";
const char PROGMEM lcd_param_text05 [] = "Pitch    P";
const char PROGMEM lcd_param_text06 [] = "Pitch    I";
const char PROGMEM lcd_param_text07 [] = "Pitch    D";
const char PROGMEM lcd_param_text08 [] = "Yaw      P";
const char PROGMEM lcd_param_text09 [] = "Yaw      I";
const char PROGMEM lcd_param_text10 [] = "Yaw      D";
#if  BARO && (!defined(SUPPRESS_BARO_ALTHOLD))
const char PROGMEM lcd_param_text11 [] = "Alt      P";
const char PROGMEM lcd_param_text12 [] = "Alt      I";
const char PROGMEM lcd_param_text13 [] = "Alt      D";
const char PROGMEM lcd_param_text14 [] = "Vel      P";
const char PROGMEM lcd_param_text15 [] = "Vel      I";
const char PROGMEM lcd_param_text16 [] = "Vel      D";
#endif
const char PROGMEM lcd_param_text17 [] = "Ang/Hor  P";
const char PROGMEM lcd_param_text18 [] = "Ang/Hor  I";
const char PROGMEM lcd_param_text188[] = "Ang/Hor  D";
const char PROGMEM lcd_param_text19 [] = "Mag      P";
const char PROGMEM lcd_param_text20 [] = "RC Rate   ";
const char PROGMEM lcd_param_text21 [] = "RC Expo   ";
const char PROGMEM lcd_param_text20t [] = "Throtт Mid";
```

```
const char PROGMEM lcd_param_text21t [] = "ThrottExpo";
const char PROGMEM lcd_param_text22 [] = "P&R Rate  ";
const char PROGMEM lcd_param_text23 [] = "Yaw Rate  ";
const char PROGMEM lcd_param_text24 [] = "Thrott PID";
#ifdef LOG_VALUES
#if (LOG_VALUES >= 3)
const char PROGMEM lcd_param_text25 [] = "pmeter m0";
const char PROGMEM lcd_param_text26 [] = "pmeter m1";
const char PROGMEM lcd_param_text27 [] = "pmeter m2";
const char PROGMEM lcd_param_text28 [] = "pmeter m3";
const char PROGMEM lcd_param_text29 [] = "pmeter m4";
const char PROGMEM lcd_param_text30 [] = "pmeter m5";
const char PROGMEM lcd_param_text31 [] = "pmeter m6";
const char PROGMEM lcd_param_text32 [] = "pmeter m7";
#endif //                              0123456789
#endif
#ifdef FLYING_WING
const char PROGMEM lcd_param_text36 [] = "SERvTRIM1";
const char PROGMEM lcd_param_text37 [] = "SERvTRIM2";
#endif
#ifdef TRI //                           0123456789
const char PROGMEM lcd_param_text38 [] = "SERvTRIMy";
const char PROGMEM lcd_param_text39 [] = "SERvINVy";
const char PROGMEM lcd_param_text152 [] = "SERvMINy";
const char PROGMEM lcd_param_text153 [] = "SERvMAXy";
#endif
//#ifdef LOG_VALUES
//const char PROGMEM lcd_param_text39 [] = "failsafes ";
//const char PROGMEM lcd_param_text40 [] = "i2c errors";
//const char PROGMEM lcd_param_text41 [] = "an overrun";
//#endif
#if defined(LCD_CONF_AUX)
const char PROGMEM lcd_param_text41 [] = "AUX angle ";
const char PROGMEM lcd_param_text42 [] = "AUX horizn";
const char PROGMEM lcd_param_text43 [] = "AUX baro  ";
const char PROGMEM lcd_param_text44 [] = "AUX mag   ";
const char PROGMEM lcd_param_text45 [] = "AUX camstb";
const char PROGMEM lcd_param_text46 [] = "AUX camtrg";
const char PROGMEM lcd_param_text47 [] = "AUX arm   ";
const char PROGMEM lcd_param_text48 [] = "AUX gpshom";
const char PROGMEM lcd_param_text49 [] = "AUX gpshld";
const char PROGMEM lcd_param_text50 [] = "AUX passth";
const char PROGMEM lcd_param_text51 [] = "AUX headfr";
const char PROGMEM lcd_param_text52 [] = "AUX buzzer";
const char PROGMEM lcd_param_text53 [] = "AUX vario ";
const char PROGMEM lcd_param_text54 [] = "AUX calib ";
const char PROGMEM lcd_param_text55 [] = "AUX govern";
const char PROGMEM lcd_param_text56 [] = "AUX osd   ";
// 53 to 61 reserved
#endif
```

```
#ifdef HELI_120_CCPM //                0123456789
const char PROGMEM lcd_param_text73 [] = "SERvTRIMn";
const char PROGMEM lcd_param_text74 [] = "SERvTRIMl";
const char PROGMEM lcd_param_text75 [] = "SERvTRIMy";
const char PROGMEM lcd_param_text76 [] = "SERvTRIMr";
const char PROGMEM lcd_param_text140 [] = "SERvINVn";
const char PROGMEM lcd_param_text141 [] = "SERvINVl";
const char PROGMEM lcd_param_text142 [] = "SERvINVy";
const char PROGMEM lcd_param_text143 [] = "SERvINVr";
#endif
#ifdef GYRO_SMOOTHING //               0123456789
const char PROGMEM lcd_param_text80 [] = "GSMOOTH R ";
const char PROGMEM lcd_param_text81 [] = "GSMOOTH P ";
const char PROGMEM lcd_param_text82 [] = "GSMOOTH Y ";
#endif
#ifdef AIRPLANE //                    0123456789
const char PROGMEM lcd_param_text83 [] = "SERVoMID3";
const char PROGMEM lcd_param_text84 [] = "SERVoMID4";
const char PROGMEM lcd_param_text85 [] = "SERVoMID5";
const char PROGMEM lcd_param_text86 [] = "SERVoMID6";
const char PROGMEM lcd_param_text87 [] = "SERVoMID7";
#endif
#if GPS
const char PROGMEM lcd_param_text91 [] = "GPS Pos. P";
const char PROGMEM lcd_param_text92 [] = "GPS Pos. I";
const char PROGMEM lcd_param_text93 [] = "Pos Rate P";
const char PROGMEM lcd_param_text94 [] = "Pos Rate I";
const char PROGMEM lcd_param_text95 [] = "Pos Rate D";
const char PROGMEM lcd_param_text96 [] = "NAV Rate P";
const char PROGMEM lcd_param_text97 [] = "NAV Rate I";
const char PROGMEM lcd_param_text98 [] = "NAV Rate D";
#endif
#if defined (FAILSAFE)
const char PROGMEM lcd_param_text101 [] = "FailThrot";
#endif
#ifdef VBAT
const char PROGMEM lcd_param_text35 [] =  "batt volt ";
const char PROGMEM lcd_param_text102 [] = "VBAT SCALE";
const char PROGMEM lcd_param_text103 [] = "BattWarn 1";
const char PROGMEM lcd_param_text104 [] = "BattWarn 2";
const char PROGMEM lcd_param_text106 [] = "BattW Crit";
//const char PROGMEM lcd_param_text107 [] = "Batt NoBat";
#endif
#ifdef POWERMETER
const char PROGMEM lcd_param_text33 [] = "pmeterSum";
const char PROGMEM lcd_param_text34 [] = "pAlarm /50"; // change text to
represent PLEVELSCALE value
const char PROGMEM lcd_param_text111 [] = "PMsENSOR0";
const char PROGMEM lcd_param_text114 [] = "PM INT2MA ";
//const char PROGMEM lcd_param_text112 [] = "PM DIVSOFT";
```

```
//const char PROGMEM lcd_param_text113 [] = "PM DIV    ";
#endif
#ifdef MMGYRO
const char PROGMEM lcd_param_text121 [] = "MMGYRO    ";
#endif
const char PROGMEM lcd_param_text131 [] = "MINTHROTL";
#if defined(ARMEDTIMEWARNING)
const char PROGMEM lcd_param_text132 [] = "ArmdTWarn";
#endif
#ifdef GOVERNOR_P
const char PROGMEM lcd_param_text133 [] = "Govern  P";
const char PROGMEM lcd_param_text134 [] = "Govern  D";
const char PROGMEM lcd_param_text135 [] = "GovernRpm";
#endif
#ifdef MULTIPLE_CONFIGURATION_PROFILES
const char PROGMEM lcd_param_text150 [] = "writeCset";
#endif
const char PROGMEM lcd_param_text151 [] = "Reset (7)";
#ifdef YAW_COLL_PRECOMP
const char PROGMEM lcd_param_text155 [] = "yawPrcomp";
const char PROGMEM lcd_param_text156 [] = "yawPrDead";
#endif
//                                012345678

PROGMEM const void * const lcd_param_ptr_table [] = {
  &lcd_param_text01, &conf.pid[ROLL].P8, &__P,
  &lcd_param_text02, &conf.pid[ROLL].P8, &__P,
  &lcd_param_text03, &conf.pid[ROLL].I8, &__I,
  &lcd_param_text04, &conf.pid[ROLL].D8, &__D,
  &lcd_param_text05, &conf.pid[PITCH].P8, &__P,
  &lcd_param_text06, &conf.pid[PITCH].I8, &__I,
  &lcd_param_text07, &conf.pid[PITCH].D8, &__D,
  &lcd_param_text08, &conf.pid[YAW].P8, &__P,
  &lcd_param_text09, &conf.pid[YAW].I8, &__I,
#if (!(PID_CONTROLLER == 1)) || (!defined(COPTER_WITH_SERVO))
  &lcd_param_text10, &conf.pid[YAW].D8, &__D,
#endif
#if BARO && (!defined(SUPPRESS_BARO_ALTHOLD))
  &lcd_param_text11, &conf.pid[PIDALT].P8, &__P,
  &lcd_param_text12, &conf.pid[PIDALT].I8, &__I,
  &lcd_param_text13, &conf.pid[PIDALT].D8, &__D,
  &lcd_param_text14, &conf.pid[PIDVEL].P8, &__P,
  &lcd_param_text15, &conf.pid[PIDVEL].I8, &__I,
  &lcd_param_text16, &conf.pid[PIDVEL].D8, &__D,
#endif
  &lcd_param_text17, &conf.pid[PIDLEVEL].P8, &__P,
  &lcd_param_text18, &conf.pid[PIDLEVEL].I8, &__I,
  &lcd_param_text188, &conf.pid[PIDLEVEL].D8, &__D,
#if MAG
  &lcd_param_text19, &conf.pid[PIDMAG].P8, &__P,
```

```
#endif
  &lcd_param_text20t, &conf.thrMid8, &__RC,
  &lcd_param_text21t, &conf.thrExpo8, &__RC,
  &lcd_param_text20, &conf.rcRate8, &__RC,
  &lcd_param_text21, &conf.rcExpo8, &__RC,
  &lcd_param_text22, &conf.rollPitchRate, &__RC,
  &lcd_param_text23, &conf.yawRate, &__RC,
  &lcd_param_text24, &conf.dynThrPID, &__RC,
#if GPS
 &lcd_param_text91, &conf.pid[PIDPOS].P8 , &__RC,
 &lcd_param_text92, &conf.pid[PIDPOS].I8 , &__I,
 &lcd_param_text93, &conf.pid[PIDPOSR].P8, &__P,
 &lcd_param_text94, &conf.pid[PIDPOSR].I8, &__I,
 &lcd_param_text95, &conf.pid[PIDPOSR].D8, &__I,
 &lcd_param_text96, &conf.pid[PIDNAVR].P8, &__P,
 &lcd_param_text97, &conf.pid[PIDNAVR].I8, &__RC,
 &lcd_param_text98, &conf.pid[PIDNAVR].D8, &__I,
#endif
#ifdef LCD_CONF_AUX
  #if ACC
    &lcd_param_text41, &conf.activate[BOXANGLE], &__AUX1,
    #ifndef SUPPRESS_LCD_CONF_AUX2
      &lcd_param_text41, &conf.activate[BOXANGLE], &__AUX2,
    #endif
    &lcd_param_text42, &conf.activate[BOXHORIZON], &__AUX1,
    #ifndef SUPPRESS_LCD_CONF_AUX2
      &lcd_param_text42, &conf.activate[BOXHORIZON], &__AUX2,
    #endif
    #ifndef SUPPRESS_LCD_CONF_AUX34
      &lcd_param_text41, &conf.activate[BOXANGLE], &__AUX3,
      &lcd_param_text41, &conf.activate[BOXANGLE], &__AUX4,
      &lcd_param_text42, &conf.activate[BOXHORIZON], &__AUX3,
      &lcd_param_text42, &conf.activate[BOXHORIZON], &__AUX4,
    #endif
  #endif
  #if BARO && (!defined(SUPPRESS_BARO_ALTHOLD))
    &lcd_param_text43, &conf.activate[BOXBARO], &__AUX1,
    #ifndef SUPPRESS_LCD_CONF_AUX2
      &lcd_param_text43, &conf.activate[BOXBARO], &__AUX2,
    #endif
    #ifndef SUPPRESS_LCD_CONF_AUX34
      &lcd_param_text43, &conf.activate[BOXBARO], &__AUX3,
      &lcd_param_text43, &conf.activate[BOXBARO], &__AUX4,
    #endif
  #endif
  &lcd_param_text44, &conf.activate[BOXMAG], &__AUX1,
  #ifndef SUPPRESS_LCD_CONF_AUX2
    &lcd_param_text44, &conf.activate[BOXMAG], &__AUX2,
  #endif
  #ifndef SUPPRESS_LCD_CONF_AUX34
```

```
  &lcd_param_text44, &conf.activate[BOXMAG], &__AUX3,
  &lcd_param_text44, &conf.activate[BOXMAG], &__AUX4,
#endif
#ifdef GIMBAL
  &lcd_param_text45, &conf.activate[BOXCAMSTAB], &__AUX1,
  #ifndef SUPPRESS_LCD_CONF_AUX2
    &lcd_param_text45, &conf.activate[BOXCAMSTAB], &__AUX2,
  #endif
  #ifndef SUPPRESS_LCD_CONF_AUX34
    &lcd_param_text45, &conf.activate[BOXCAMSTAB], &__AUX3,
    &lcd_param_text45, &conf.activate[BOXCAMSTAB], &__AUX4,
  #endif
  &lcd_param_text46, &conf.activate[BOXCAMTRIG], &__AUX1,
  #ifndef SUPPRESS_LCD_CONF_AUX2
    &lcd_param_text46, &conf.activate[BOXCAMTRIG], &__AUX2,
  #endif
  #ifndef SUPPRESS_LCD_CONF_AUX34
    &lcd_param_text46, &conf.activate[BOXCAMTRIG], &__AUX3,
    &lcd_param_text46, &conf.activate[BOXCAMTRIG], &__AUX4,
  #endif
#endif
&lcd_param_text47, &conf.activate[BOXARM], &__AUX1,
#ifndef SUPPRESS_LCD_CONF_AUX2
  &lcd_param_text47, &conf.activate[BOXARM], &__AUX2,
#endif
#ifndef SUPPRESS_LCD_CONF_AUX34
  &lcd_param_text47, &conf.activate[BOXARM], &__AUX3,
  &lcd_param_text47, &conf.activate[BOXARM], &__AUX4,
#endif
#if GPS
  &lcd_param_text48, &conf.activate[BOXGPSHOME], &__AUX1,
  #ifndef SUPPRESS_LCD_CONF_AUX2
    &lcd_param_text48, &conf.activate[BOXGPSHOME], &__AUX2,
  #endif
  #ifndef SUPPRESS_LCD_CONF_AUX34
    &lcd_param_text48, &conf.activate[BOXGPSHOME], &__AUX3,
    &lcd_param_text48, &conf.activate[BOXGPSHOME], &__AUX4,
  #endif
  &lcd_param_text49, &conf.activate[BOXGPSHOLD], &__AUX1,
  #ifndef SUPPRESS_LCD_CONF_AUX2
    &lcd_param_text49, &conf.activate[BOXGPSHOLD], &__AUX2,
  #endif
  #ifndef SUPPRESS_LCD_CONF_AUX34
    &lcd_param_text49, &conf.activate[BOXGPSHOLD], &__AUX3,
    &lcd_param_text49, &conf.activate[BOXGPSHOLD], &__AUX4,
  #endif
#endif
#if defined(FIXEDWING) || defined(HELICOPTER)
  &lcd_param_text50, &conf.activate[BOXPASSTHRU],&__AUX1,
  #ifndef SUPPRESS_LCD_CONF_AUX2
```

```
      &lcd_param_text50, &conf.activate[BOXPASSTHRU],&__AUX2,
    #endif
    #ifndef SUPPRESS_LCD_CONF_AUX34
      &lcd_param_text50, &conf.activate[BOXPASSTHRU],&__AUX3,
      &lcd_param_text50, &conf.activate[BOXPASSTHRU],&__AUX4,
    #endif
   #endif
   #if defined(HEADFREE)
    &lcd_param_text51, &conf.activate[BOXHEADFREE],&__AUX1,
    #ifndef SUPPRESS_LCD_CONF_AUX2
      &lcd_param_text51, &conf.activate[BOXHEADFREE],&__AUX2,
    #endif
    #ifndef SUPPRESS_LCD_CONF_AUX34
      &lcd_param_text51, &conf.activate[BOXHEADFREE],&__AUX3,
      &lcd_param_text51, &conf.activate[BOXHEADFREE],&__AUX4,
    #endif
   #endif
   #if defined(BUZZER)
    &lcd_param_text52, &conf.activate[BOXBEEPERON],&__AUX1,
    #ifndef SUPPRESS_LCD_CONF_AUX2
      &lcd_param_text52, &conf.activate[BOXBEEPERON],&__AUX2,
    #endif
    #ifndef SUPPRESS_LCD_CONF_AUX34
      &lcd_param_text52, &conf.activate[BOXBEEPERON],&__AUX3,
      &lcd_param_text52, &conf.activate[BOXBEEPERON],&__AUX4,
    #endif
   #endif
   #ifdef VARIOMETER
    &lcd_param_text53, &conf.activate[BOXVARIO],&__AUX1,
    #ifndef SUPPRESS_LCD_CONF_AUX2
      &lcd_param_text53, &conf.activate[BOXVARIO],&__AUX2,
    #endif
    #ifndef SUPPRESS_LCD_CONF_AUX34
      &lcd_param_text53, &conf.activate[BOXVARIO],&__AUX3,
      &lcd_param_text53, &conf.activate[BOXVARIO],&__AUX4,
    #endif
   #endif
   #ifdef INFLIGHT_ACC_CALIBRATION
    &lcd_param_text54, &conf.activate[BOXCALIB],&__AUX1,
    #ifndef SUPPRESS_LCD_CONF_AUX2
      &lcd_param_text54, &conf.activate[BOXCALIB],&__AUX2,
    #endif
    #ifndef SUPPRESS_LCD_CONF_AUX34
      &lcd_param_text54, &conf.activate[BOXCALIB],&__AUX3,
      &lcd_param_text54, &conf.activate[BOXCALIB],&__AUX4,
    #endif
   #endif
  #ifdef GOVERNOR_P
   &lcd_param_text55, &conf.activate[BOXGOV],&__AUX1,
   #ifndef SUPPRESS_LCD_CONF_AUX2
```

```
    &lcd_param_text55, &conf.activate[BOXGOV],&__AUX2,
  #endif
  #ifndef SUPPRESS_LCD_CONF_AUX34
    &lcd_param_text55, &conf.activate[BOXGOV],&__AUX3,
    &lcd_param_text55, &conf.activate[BOXGOV],&__AUX4,
  #endif
#endif
#ifdef OSD_SWITCH
  &lcd_param_text56, &conf.activate[BOXOSD],&__AUX1,
  #ifndef SUPPRESS_LCD_CONF_AUX2
    &lcd_param_text56, &conf.activate[BOXOSD],&__AUX2,
  #endif
  #ifndef SUPPRESS_LCD_CONF_AUX34
    &lcd_param_text56, &conf.activate[BOXOSD],&__AUX3,
    &lcd_param_text56, &conf.activate[BOXOSD],&__AUX4,
  #endif
#endif
#endif //lcd.conf.aux

#ifdef LOG_VALUES
#if (LOG_VALUES >= 3)
#if (NUMBER_MOTOR > 0)
  &lcd_param_text25, &pMeter[0], &__PM,
#endif
#if (NUMBER_MOTOR > 1)
  &lcd_param_text26, &pMeter[1], &__PM,
#endif
#if (NUMBER_MOTOR > 2)
  &lcd_param_text27, &pMeter[2], &__PM,
#endif
#if (NUMBER_MOTOR > 3)
  &lcd_param_text28, &pMeter[3], &__PM,
#endif
#if (NUMBER_MOTOR > 4)
  &lcd_param_text29, &pMeter[4], &__PM,
#endif
#if (NUMBER_MOTOR > 5)
  &lcd_param_text30, &pMeter[5], &__PM,
#endif
#if (NUMBER_MOTOR > 6)
  &lcd_param_text31, &pMeter[6], &__PM,
#endif
#if (NUMBER_MOTOR > 7)
  &lcd_param_text32, &pMeter[7], &__PM,
#endif
#endif
#endif
  &lcd_param_text131, &conf.minthrottle, &__ST,
#if defined (FAILSAFE)
  &lcd_param_text101, &conf.failsafe_throttle, &__ST,
```

```
#endif
#ifdef FLYING_WING
  &lcd_param_text36, &conf.servoConf[3].middle, &__SE,
  &lcd_param_text37, &conf.servoConf[4].middle, &__SE,
#endif
#ifdef TRI
  &lcd_param_text152, &conf.servoConf[5].min, &__SE,
  &lcd_param_text153, &conf.servoConf[5].max, &__SE,
  &lcd_param_text38, &conf.servoConf[5].middle, &__SE,
  &lcd_param_text39, &conf.servoConf[5].rate, &__BITS,
#endif
#ifdef HELI_120_CCPM
  &lcd_param_text73, &conf.servoConf[3].middle, &__SE,
  &lcd_param_text74, &conf.servoConf[4].middle, &__SE,
  &lcd_param_text76, &conf.servoConf[6].middle, &__SE,
  &lcd_param_text75, &conf.servoConf[5].middle, &__SE,
  &lcd_param_text140, &conf.servoConf[3].rate, &__BITS,
  &lcd_param_text141, &conf.servoConf[4].rate, &__BITS,
  &lcd_param_text143, &conf.servoConf[6].rate, &__BITS,
  &lcd_param_text142, &conf.servoConf[5].rate, &__BITS,
#endif
#ifdef GOVERNOR_P
  &lcd_param_text133, &conf.governorP, &__D,
  &lcd_param_text134, &conf.governorD, &__D,
#endif
#ifdef YAW_COLL_PRECOMP
  &lcd_param_text155, &conf.yawCollPrecomp, &__PT,
  &lcd_param_text156, &conf.yawCollPrecompDeadband, &__SE,
#endif
#ifdef GYRO_SMOOTHING
  &lcd_param_text80, &conf.Smoothing[0], &__D,
  &lcd_param_text81, &conf.Smoothing[1], &__D,
  &lcd_param_text82, &conf.Smoothing[2], &__D,
#endif
#ifdef AIRPLANE
  &lcd_param_text83, &conf.servoConf[3].middle, &__SE,
  &lcd_param_text84, &conf.servoConf[4].middle, &__SE,
  &lcd_param_text85, &conf.servoConf[5].middle, &__SE,
  &lcd_param_text86, &conf.servoConf[6].middle, &__SE,
#endif
#ifdef MMGYRO
  &lcd_param_text121, &conf.mmgyro, &__D,
#endif
#ifdef VBAT
//  &lcd_param_text35, &analog.vbat, &__VB,
  &lcd_param_text102, &conf.vbatscale, &__PT,
  &lcd_param_text103, &conf.vbatlevel_warn1, &__P,
  &lcd_param_text104, &conf.vbatlevel_warn2, &__P,
  &lcd_param_text106, &conf.vbatlevel_crit, &__P,
//  &lcd_param_text107, &conf.no_vbat, &__P,
```

```
#endif
#ifdef POWERMETER
//  &lcd_param_text33, &pMeter[PMOTOR_SUM], &__PM,
  &lcd_param_text114, &conf.pint2ma, &__PT,
  #ifdef POWERMETER_HARD
    &lcd_param_text111, &conf.psensornull, &__SE1,
  #endif
  &lcd_param_text34, &conf.powerTrigger1, &__PT,
#endif
#if defined(ARMEDTIMEWARNING)
  &lcd_param_text132, &conf.armedtimewarning, &__SE,
#endif
#ifdef MULTIPLE_CONFIGURATION_PROFILES
  &lcd_param_text150, &global_conf.currentSet, &__D,
#endif
&lcd_param_text151, &reset_to_defaults, &__D,
//#ifdef LOG_VALUES
//  &lcd_param_text39, &failsafeEvents, &__L,
//  &lcd_param_text40, &i2c_errors_count, &__L,
//  &lcd_param_text41, &annex650_overrun_count, &__L
//#endif
};
#define PARAMMAX (sizeof(lcd_param_ptr_table)/6 - 1)
//
**********************************************************************
************************************

void __u8Inc(void * var, int16_t inc) {*(uint8_t*)var += (uint8_t)inc;};
void __s8Inc(void * var, int16_t inc) {*(int8_t*)var += (int8_t)inc;};
void __u16Inc(void * var, int16_t inc) {*(uint16_t*)var += inc;};
void __s16Inc(void * var, int16_t inc) {*(int16_t*)var += inc;};
void __nullInc(void * var, int16_t inc) {};

void __u8Fmt(void * var, uint8_t mul, uint8_t dec) {
  uint16_t unit = *(uint8_t*)var;
  unit *= mul;
  char c1 = '0'+unit/100; char c2 = '0'+unit/10-(unit/100)*10; char c3 = '0'+unit-
(unit/10)*10;
  switch (dec) {
    case 0: line2[3] = c1; line2[4] = c2; line2[5] = c3; break;
    case 1: line2[2] = c1; line2[3] = c2; line2[4] = '.'; line2[5] = c3; break;
    case 2: line2[2] = c1; line2[3] = '.'; line2[4] = c2; line2[5] = c3; break;
    case 3: line2[1] = '0'; line2[2] = '.'; line2[3] = c1; line2[4] = c2; line2[5] = c3; break;
  }
}

void __u16Fmt(void * var, uint8_t mul, uint8_t dec) {
  uint16_t unit = *(uint16_t*)var;
  unit *= mul;
  line2[2] = digit10000(unit);
```

```
  line2[3] = digit1000(unit);
  line2[4] = digit100(unit);
  line2[5] = digit10(unit);
  line2[6] = digit1(unit);
}
void __s16Fmt(void * var, uint8_t mul, uint8_t dec) {
  int16_t unit = *(int16_t*)var;
  if (unit >= 0) {
    line2[1] = ' ';
  } else {
    line2[1] = '-';
    unit = -unit;
  }
  __u16Fmt(&unit, mul, dec);
}
void __uAuxFmt1(void * var, uint8_t mul, uint8_t dec) {  __uAuxFmt(var, mul, dec,
1); }
void __uAuxFmt2(void * var, uint8_t mul, uint8_t dec) {  __uAuxFmt(var, mul, dec,
2); }
void __uAuxFmt3(void * var, uint8_t mul, uint8_t dec) {  __uAuxFmt(var, mul, dec,
3); }
void __uAuxFmt4(void * var, uint8_t mul, uint8_t dec) {  __uAuxFmt(var, mul, dec,
4); }


void __uAuxFmt(void * var, uint8_t mul, uint8_t dec, uint8_t aux) {
  uint16_t unit = *(uint16_t*)var;
  line2[1] =  digit1(aux);
  line2[3] =  ( unit & 1<<(3*aux-3) ? 'L' : '.' );
  line2[4] =  ( unit & 1<<(3*aux-2) ? 'M' : '.' );
  line2[5] =  ( unit & 1<<(3*aux-1) ? 'H' : '.' );
}

void __s8BitsFmt(void * var, uint8_t mul, uint8_t dec) {
  int8_t unit = *(int8_t*)var;
  line2[1] =  ( unit & 1<<2 ? 'C' : '.' );
  line2[2] =  ( unit & 1<<1 ? 'N' : '.' );
  line2[3] =  ( unit & 1<<0 ? 'R' : '.' );
}

#ifdef POWERMETER
  void __upMFmt(void * var, uint8_t mul, uint8_t dec) {
    uint32_t unit = *(uint32_t*)var;
      unit /= PLEVELDIV;
   __u16Fmt(&unit, mul, dec);
  }

//void __upSFmt(void * var, uint8_t mul, uint8_t dec) {
//  uint32_t unit = *(uint32_t*)var;
//#if defined(POWERMETER_SOFT)
```

```
//  unit = (unit * (uint32_t)conf.pint2ma) / 100000; // was = unit / conf.pleveldivsoft;
//#elif defined(POWERMETER_HARD)
//  unit = unit / PLEVELDIV;
//#endif
//  __u16Fmt(&unit, mul, dec);
//}
#endif

static uint8_t lcdStickState[4];
#define IsLow(x)  (lcdStickState[x] & 0x1)
#define IsHigh(x) (lcdStickState[x] & 0x2)
#define IsMid(x)  (!lcdStickState[x])


/* ------------ DISPLAY_2LINES ------------------------------------*/
#ifdef DISPLAY_2LINES
void ConfigRefresh(uint8_t p) {
  blinkLED(10,20,1);
  SET_ALARM_BUZZER(ALRM_FAC_TOGGLE, ALRM_LVL_TOGGLE_1);
  strcpy_P(line1,PSTR("                "));
  strcpy(line2,line1);
  strcpy_P(line1, (char*)pgm_read_word(&(lcd_param_ptr_table[p * 3])));
  lcd_param_def_t* deft =
(lcd_param_def_t*)pgm_read_word(&(lcd_param_ptr_table[(p * 3) + 2]));
  deft->type->fmt((void*)pgm_read_word(&(lcd_param_ptr_table[(p * 3) + 1])), deft-
>multiplier, deft->decimal);
  LCDclear();
  LCDsetLine(1);LCDprintChar(line1); //refresh line 1 of LCD
  LCDsetLine(2);LCDprintChar(line2);//refresh line 2 of LCD
}
#endif // DISPLAY_2LINES
/* ------------ DISPLAY_MULTILINE ---------------------------------*/
#ifdef DISPLAY_MULTILINE
// display slice of config items prior and after current item (index p)
void ConfigRefresh(uint8_t p) {
  uint8_t j, l = 1;
  int8_t pp = (int8_t)p;
  #ifndef OLED_I2C_128x64
   blinkLED(2,4,1);
   SET_ALARM_BUZZER(ALRM_FAC_TOGGLE, ALRM_LVL_TOGGLE_1);
   LCDclear();
  #else
   delay(60);
  #endif
  for (int8_t i=pp - MULTILINE_PRE; i<pp + MULTILINE_POST; i++) {
    //j = i % (1+PARAMMAX); // why does modulo not work here?
    j = (i<0 ? i + 1 + PARAMMAX : i);
    if (j > PARAMMAX) j -= (1 + PARAMMAX);
    strcpy_P(line1,PSTR("          "));
    strcpy(line2,line1);
```

```
    strcpy_P(line1, (char*)pgm_read_word(&(lcd_param_ptr_table[j * 3])));
    lcd_param_def_t* deft =
(lcd_param_def_t*)pgm_read_word(&(lcd_param_ptr_table[(j * 3) + 2]));
    deft->type->fmt((void*)pgm_read_word(&(lcd_param_ptr_table[(j * 3) + 1])),
deft->multiplier, deft->decimal);

    LCDsetLine(l++);
    if (j == p) {
      #if ( !defined(OLED_I2C_128x64) && !defined(OLED_DIGOLE) )
        LCDprint('>');
      #endif
      #if ( defined(OLED_DIGOLE) )
        line2[0] = '*';
      #endif
     LCDattributesReverse();
    }
    LCDprintChar(line1); // the label
    if (j == p) {LCDattributesOff(); /*LCDattributesBold();*/}
    //LCDprint(' ');
    LCDprintChar(line2); // the value
    #if ( !defined(OLED_I2C_128x64) && !defined(OLED_DIGOLE) )
      if (j == p) {LCDattributesOff(); LCDprint('<');}
    #endif
    LCD_FLUSH;
  }
  LCDcrlf();
}
#endif // DISPLAY_MULTILINE
void configurationLoop() {
  uint8_t i, p;
  uint8_t LCD=1;
  uint8_t refreshLCD = 1;
  uint8_t key = 0;
  uint8_t allow_exit = 0;
  #ifdef MULTIPLE_CONFIGURATION_PROFILES
    uint8_t currentSet = global_conf.currentSet; // keep a copy for abort.case
  #endif
  initLCD();
  #if defined OLED_I2C_128x64LOGO_PERMANENT
    LCDclear();
  #endif
  reset_to_defaults = 0;
  p = 0;
  while (LCD == 1) {
    if (refreshLCD) {
      ConfigRefresh(p);
      refreshLCD = 0;
    }
    #if defined(SERIAL_RX)
      delay(10); // may help with timing for some serial receivers -1/100 second seems
```

```
non-critical here?
      if (spekFrameFlags == 0x01) readSerial_RX();
      delay(44); // For digital receivers , to ensure that an "old" frame does not cause
immediate exit at startup.
    #endif
    #if defined(LCD_TEXTSTAR) || defined(LCD_VT100) || defined(LCD_TTY) //
textstar, vt100 and tty can send keys
      key = ( SerialAvailable(LCD_SERIAL_PORT) ?
SerialRead(LCD_SERIAL_PORT) : 0 );
    #endif
    #ifdef LCD_CONF_DEBUG
      delay(1000);
      if (key == LCD_MENU_NEXT) key=LCD_VALUE_UP; else key =
LCD_MENU_NEXT;
    #endif
    for (i = ROLL; i <= THROTTLE; i++) {uint16_t Tmp = readRawRC(i);
lcdStickState[i] = (Tmp < MINCHECK) | ((Tmp > MAXCHECK) << 1);};
    if (IsMid(YAW) && IsMid(PITCH) && IsMid(ROLL)) allow_exit = 1;
    if (key == LCD_MENU_SAVE_EXIT || (IsLow(YAW) && IsHigh(PITCH) &&
allow_exit))   LCD = 0; // save and exit
    else if (key == LCD_MENU_ABORT || (IsHigh(YAW) && IsHigh(PITCH) &&
allow_exit)) LCD = 2;// exit without save: eeprom has only 100.000 write cycles
    else if (key == LCD_MENU_NEXT || (IsLow(PITCH) && IsMid(YAW))) {
//switch config param with pitch
      refreshLCD = 1; p++; if (p>PARAMMAX) p = 0;
    } else if (key == LCD_MENU_PREV || (IsHigh(PITCH) && IsMid(YAW))) {
      refreshLCD = 1; p--; if (p == 0xFF) p = PARAMMAX;
    } else if (key == LCD_VALUE_DOWN || (IsLow(ROLL))) { //+ or - param with
low and high roll
      refreshLCD = 1;
      lcd_param_def_t* deft =
(lcd_param_def_t*)pgm_read_word(&(lcd_param_ptr_table[(p * 3) + 2]));
      deft->type->inc((void*)pgm_read_word(&(lcd_param_ptr_table[(p * 3) + 1])), -
(IsHigh(THROTTLE) ? 10: 1) * deft->increment);
      if (p == 0) conf.pid[PITCH].P8 = conf.pid[ROLL].P8;
    } else if (key == LCD_VALUE_UP || (IsHigh(ROLL))) {
      refreshLCD = 1;
      lcd_param_def_t* deft =
(lcd_param_def_t*)pgm_read_word(&(lcd_param_ptr_table[(p * 3) + 2]));
      deft->type->inc((void*)pgm_read_word(&(lcd_param_ptr_table[(p * 3) + 1])),
+(IsHigh(THROTTLE) ? 10 : 1) * deft->increment);
      if (p == 0) conf.pid[PITCH].P8 = conf.pid[ROLL].P8;
    }
    #if defined(PRI_SERVO_TO)
      #define MAX_SERV 7
      #if(PRI_SERVO_TO < MAX_SERV)
        #undef  MAX_SERV
        #define MAX_SERV PRI_SERVO_TO
      #endif
      for(i=PRI_SERVO_FROM-1; i<MAX_SERV; i++) servo[i] =
```

```
conf.servoConf[i].middle;
      #if defined(HELICOPTER) && YAWMOTOR
        servo[5] =  MINCOMMAND;
      #endif
      #if defined(TRI) && defined(MEGA_HW_PWM_SERVOS) &&
defined(MEGA)
        servo[3] = servo[5];
      #endif
      writeServos();
    #endif
  } // while (LCD == 1)
  blinkLED(20,30,1);
  SET_ALARM_BUZZER(ALRM_FAC_CONFIRM, ALRM_LVL_CONFIRM_1);
  LCDclear();
  LCDsetLine(1);
  if (LCD == 0) { //     0123456789
    strcpy_P(line1,PSTR("Saving..."));
    #ifdef MULTIPLE_CONFIGURATION_PROFILES
      line1[7] = digit1(global_conf.currentSet);
    #endif
    LCDprintChar(line1);
    #ifdef MULTIPLE_CONFIGURATION_PROFILES
      writeGlobalSet(0);
    #endif
    if (reset_to_defaults == 7) {
      // magic number lucky 7 hit, then do the reset to defaults
      strcpy_P(line1,PSTR("RESET.."));
      //                0123456789.12345
      LCDprintChar(line1);
      LoadDefaults(); // this invokes writeParams()
    } else {
      writeParams(1);
    }
  } else {
    strcpy_P(line1,PSTR("Aborting"));
    LCDprintChar(line1);
    #ifdef MULTIPLE_CONFIGURATION_PROFILES
      global_conf.currentSet = currentSet; // restore
    #endif
    readEEPROM(); // roll back all values to last saved state
  }
  LCDsetLine(2);
  strcpy_P(line1,PSTR("Exit"));
  LCDprintChar(line1);
  #if defined(LCD_LCD03) || defined(LCD_LCD03S)
    delay(2000); // wait for two seconds then clear screen and show initial message
    initLCD();
  #endif
  #if defined(LCD_SERIAL3W)
    SerialOpen(0,115200);
```

```
  #endif
  #if defined(LCD_TELEMETRY) || defined(OLED_I2C_128x64)
    delay(1500); // keep exit message visible for one and one half seconds even if
(auto)telemetry continues writing in main loop
  #endif
  cycleTime = 0;
  #if defined(OLED_I2C_128x64)
    #if defined(OLED_I2C_128x64LOGO_PERMANENT)
      i2c_OLED_Put_Logo();
    #elif !defined(LOG_PERMANENT_SHOW_AFTER_CONFIG)
      LCDclear();
    #endif
  #endif
  #ifdef LOG_PERMANENT_SHOW_AFTER_CONFIG
    if (!f.ARMED) dumpPLog(0);
  #endif
}
#endif // LCD_CONF
// ------------------- telemetry output to LCD over serial/i2c ------------------------------
--

#ifdef LCD_TELEMETRY

// LCDbar(n,v) : draw a bar graph - n number of chars for width, v value in % to
display
void LCDbar(uint8_t n,uint8_t v) {
  if (v > 200) v = 0;
  else if (v > 100) v = 100;
  #if defined(LCD_SERIAL3W)
    for (uint8_t i=0; i< n; i++) LCDprint((i<n*v/100 ? '=' : '.'));
  #elif defined(LCD_TEXTSTAR)
    LCDprint(0xFE);LCDprint('b');LCDprint(n);LCDprint(v);
  #elif defined(LCD_VT100) || defined(LCD_TTY) // || defined(OLED_DIGOLE)
    uint8_t i, j = (n*v)/100;
    for (i=0; i< j; i++) LCDprint( '=' );
    for (i=j; i< n; i++) LCDprint( '.' );
  #elif defined(LCD_ETPP)
    ETPP_barGraph(n,v);
  #elif defined(LCD_LCD03) || defined(LCD_LCD03S)
    for (uint8_t i=0; i< n; i++) LCDprint((i<n*v/100 ? '=' : '.'));
  #elif defined(OLED_I2C_128x64)
    uint8_t i, j = (n*v)/100;
    for (i=0; i< j; i++) LCDprint( 159 );       // full
    if (j<n) LCDprint(154 + (v*n*5/100 - 5*j)); // partial fill
    for (i=j+1; i< n; i++) LCDprint( 154 );    // empty
  #elif defined(OLED_DIGOLE)
    uint8_t i, j = (n*v)/100;
    char l[n+1];
    for (i=0; i< j; i++) l[i] = '=';
    for (i=j; i< n; i++) l[i] = '.';
```

```
    l[n] = 0;
    LCDprintChar(l);
  #endif
}

void fill_line1_deg() {
  uint16_t unit;
  strcpy_P(line1,PSTR("Deg ---.-  ---.-"));
  // 0123456789.12345
  if (att.angle[0] < 0 ) {
    unit = -att.angle[0];
    line1[3] = '-';
  } else
  unit = att.angle[0];
  line1[4] = digit1000(unit);
  line1[5] = digit100(unit);
  line1[6] = digit10(unit);
  line1[8] = digit1(unit);
  if (att.angle[1] < 0 ) {
    unit = -att.angle[1];
    line1[10] = '-';
  } else
  unit = att.angle[1];
  line1[11] = digit1000(unit);
  line1[12] = digit100(unit);
  line1[13] = digit10(unit);
  line1[15] = digit1(unit);
}
void output_AmaxA() {
  #ifdef POWERMETER_HARD
    //uint16_t unit;
    strcpy_P(line2,PSTR("---,-A max---,-A"));
    //unit = analog.amperage; //((uint32_t)powerValue * conf.pint2ma) / 100;
    line2[0] = digit1000(analog.amperage);
    line2[1] = digit100(analog.amperage);
    line2[2] = digit10(analog.amperage);
    line2[4] = digit1(analog.amperage);
    //if (analog.amperage > powerValueMaxMAH) powerValueMaxMAH =
analog.amperage;
    line2[10] = digit1000(powerValueMaxMAH);
    line2[11] = digit100(powerValueMaxMAH);
    line2[12] = digit10(powerValueMaxMAH);
    line2[14] = digit1(powerValueMaxMAH);
    LCDprintChar(line2);
  #endif
}
#ifdef WATTS
void output_WmaxW() {
  //                0123456789.12345
  strcpy_P(line2,PSTR("----W   max----W"));
```

```
  line2[0] = digit1000(analog.watts);
  line2[1] = digit100(analog.watts);
  line2[2] = digit10(analog.watts);
  line2[3] = digit1(analog.watts);
  line2[11] = digit1000(wattsMax);
  line2[12] = digit100(wattsMax);
  line2[13] = digit10(wattsMax);
  line2[14] = digit1(wattsMax);
  LCDprintChar(line2);
}
#endif

void output_V() {
  #ifdef VBAT
    strcpy_P(line1,PSTR(" --.-V"));
    //                0123456789.12345
    line1[1] = digit100(analog.vbat);
    line1[2] = digit10(analog.vbat);
    line1[4] = digit1(analog.vbat);
    #ifndef OLED_I2C_128x64
      if (analog.vbat < conf.vbatlevel_warn1) { LCDattributesReverse(); }
    #endif
    LCDbar(DISPLAY_COLUMNS-9, (((analog.vbat -
conf.vbatlevel_warn1)*100)/(VBATNOMINAL-conf.vbatlevel_warn1)) );
    LCDattributesOff(); // turn Reverse off for rest of display
    LCDprintChar(line1);
  #endif
}

void output_Vmin() {
  #ifdef VBAT
    strcpy_P(line1,PSTR(" --.-Vmin"));
    //                0123456789.12345
    line1[1] = digit100(vbatMin);
    line1[2] = digit10(vbatMin);
    line1[4] = digit1(vbatMin);
    #ifndef OLED_I2C_128x64
      if (vbatMin < conf.vbatlevel_crit) { LCDattributesReverse(); }
    #endif
    LCDbar(DISPLAY_COLUMNS-9, (vbatMin > conf.vbatlevel_crit ? (((vbatMin -
conf.vbatlevel_crit)*100)/(VBATNOMINAL-conf.vbatlevel_crit)) : 0 ));
    LCDattributesOff();
    LCDprintChar(line1);
  #endif
}
void output_mAh() {
  #ifdef POWERMETER
    uint16_t mah = analog.intPowerMeterSum; // fallback: display consumed mAh
    if (analog.intPowerMeterSum < (uint16_t)conf.powerTrigger1 * PLEVELSCALE)
      mah = conf.powerTrigger1 * PLEVELSCALE - analog.intPowerMeterSum; //
```

```
display mah mAh
    strcpy_P(line1,PSTR(" -----mAh"));
    line1[1] = digit10000(mah);
    line1[2] = digit1000(mah);
    line1[3] = digit100(mah);
    line1[4] = digit10(mah);
    line1[5] = digit1(mah);
    if (conf.powerTrigger1) {
      int8_t v = 100 - ( analog.intPowerMeterSum/(uint16_t)conf.powerTrigger1) *2; //
bar graph powermeter (scale intPowerMeterSum/powerTrigger1 with
*100/PLEVELSCALE)
      #ifndef OLED_I2C_128x64
        if (v <= 0) { LCDattributesReverse(); } // buzzer on? then add some blink for
attention
      #endif
      LCDbar(DISPLAY_COLUMNS-9, v);
      LCDattributesOff();
    }
    LCDprintChar(line1);
  #endif
}
void output_errors_or_armedTime() {
  if (failsafeEvents || (i2c_errors_count>>10)) { // errors
    // ignore i2c==1 because of bma020-init
    LCDalarmAndReverse();
    output_fails();
    LCDattributesOff();
  } else { // ... armed time
    uint16_t ats = armedTime / 1000000;
    #ifdef ARMEDTIMEWARNING
      #ifndef OLED_I2C_128x64
        if (ats > conf.armedtimewarning) { LCDattributesReverse(); }
      #endif
      LCDbar(DISPLAY_COLUMNS-9, (ats < conf.armedtimewarning ?
(((conf.armedtimewarning-ats+1)*25)/(conf.armedtimewarning+1)*4) : 0 ));
      LCDattributesOff();
    #endif
    LCDprint(' ');
    #ifdef ARMEDTIMEWARNING
      print_uptime( (conf.armedtimewarning > ats ? conf.armedtimewarning - ats : ats)
);
    #else
      print_uptime(ats);
    #endif
  }
}
void output_altitude() {
  #if BARO
    {
      int16_t h = alt.EstAlt / 100;
```

```
      LCDprint('A'); LCDprintInt16(h); LCDprint('m');
      h = BAROaltMax / 100;
      LCDprintChar(" ("); LCDprintInt16(h);
    }
  #endif
}
void output_uptime_cset() {
  strcpy_P(line1,PSTR("Up ")); LCDprintChar(line1); print_uptime(millis() / 1000 );
  #ifdef MULTIPLE_CONFIGURATION_PROFILES
    strcpy_P(line1,PSTR("  Cset -")); line1[7] = digit1(global_conf.currentSet);
LCDprintChar(line1);
  #endif
}
void output_cycle() {
  strcpy_P(line1,PSTR("Cycle    -----us")); //uin16_t cycleTime
  // 0123456789.12345*/
  //strcpy_P(line2,PSTR("(-----, -----)us")); //uin16_t cycleTimeMax
  line1[9] = digit10000(cycleTime);
  line1[10] = digit1000(cycleTime);
  line1[11] = digit100(cycleTime);
  line1[12] = digit10(cycleTime);
  line1[13] = digit1(cycleTime);
  LCDprintChar(line1);
}
void output_cycleMinMax() {
#if (LOG_VALUES >= 2)
  strcpy_P(line2,PSTR("(-----, -----)us")); //uin16_t cycleTimeMax
  line2[1] = digit10000(cycleTimeMin );
  line2[2] = digit1000(cycleTimeMin );
  line2[3] = digit100(cycleTimeMin );
  line2[4] = digit10(cycleTimeMin );
  line2[5] = digit1(cycleTimeMin );
  line2[8] = digit10000(cycleTimeMax);
  line2[9] = digit1000(cycleTimeMax);
  line2[10] = digit100(cycleTimeMax);
  line2[11] = digit10(cycleTimeMax);
  line2[12] = digit1(cycleTimeMax);
  LCDprintChar(line2);
#endif
}
void output_fails() {
  uint16_t unit;
  //                0123456789012345
  strcpy_P(line2,PSTR("-- Fails  -- i2c"));
  unit = failsafeEvents;
  line2[0] = digit10(unit);
  line2[1] = digit1(unit);
  unit = i2c_errors_count;
  line2[10] = digit10(unit);
  line2[11] = digit1(unit);
```

```
  LCDprintChar(line2);
}
void output_annex() {
  //                  0123456789
  strcpy_P(line2,PSTR("annex --"));
  line2[6] = digit10(annex650_overrun_count);
  line2[7] = digit1(annex650_overrun_count);
  LCDprintChar(line2);
}
static char checkboxitemNames[][4] = {
    "Arm",
    #if ACC
      "Ang","Hor",
    #endif
    #if BARO && (!defined(SUPPRESS_BARO_ALTHOLD))
      "Bar",
    #endif
    #ifdef VARIOMETER
      "Var",
    #endif
    "Mag",
    #if defined(HEADFREE)
      "HFr",
      "HAd",
    #endif
    #if defined(SERVO_TILT) || defined(GIMBAL)|| defined(SERVO_MIX_TILT)
      "CSt",
    #endif
    #if defined(CAMTRIG)
      "CTr",
    #endif
    #if GPS
      "GHm",
      "GHd",
    #endif
    #if defined(FIXEDWING) || defined(HELICOPTER)
      "Pas",
    #endif
    #if defined(BUZZER)
      "Buz",
    #endif
    #if defined(LED_FLASHER)
      "LEM",
      "LEL",
    #endif
    #if defined(LANDING_LIGHTS_DDR)
      "LLs",
    #endif
    #ifdef INFLIGHT_ACC_CALIBRATION
      "Cal",
```

```
    #endif
    #ifdef GOVERNOR_P
      "Gov",
    #endif
    #ifdef OSD_SWITCH
      "OSD",
    #endif
  ""};
void output_checkboxitems() {
  for (uint8_t i=0; i<CHECKBOXITEMS; i++ ) {
    if (rcOptions[i] || ((i==BOXARM)&&(f.ARMED)) ) {
      LCDprintChar(checkboxitemNames[i]);
      LCDprint(' ');
    }
  }
}
void output_checkboxstatus() {
  #if (defined(DISPLAY_COLUMNS))
    uint8_t cntmax = DISPLAY_COLUMNS /4;
  #else
    uint8_t cntmax = 4;
  #endif
    uint8_t cnt = 0;
  #ifdef BUZZER
    if (isBuzzerON()) { LCDalarmAndReverse(); } // buzzer on? then add some blink
for attention
  #endif
  for (uint8_t i=0; (i<CHECKBOXITEMS) && (cnt<cntmax); i++ ) {
    if (rcOptions[i] || ((i==BOXARM)&&(f.ARMED)) ) {
      LCDprintChar(checkboxitemNames[i]);
      LCDprint(' ');
      cnt++;
    }
  }
  for (uint8_t i=cnt; i<cntmax; i++) LCDprintChar(".   "); // padding to EOL
  LCDattributesOff();
}

#define GYROLIMIT 60 // threshold: for larger values replace bar with dots
#define ACCLIMIT 60 // threshold: for larger values replace bar with dots
void outputSensor(uint8_t num, int16_t data, int16_t limit) {
  if (data < -limit)     {LCDprintChar("<<<<");}
  else if (data > limit) {LCDprintChar(">>>>");}
  else LCDbar(num, limit + data *50/limit);
}
void print_uptime(uint16_t sec) {
  uint16_t m, s;
  static char line[6] = "--:--";
  m = sec / 60;
  s = sec - (60 * m);
```

```
  line[0] = digit10(m);
  line[1] = digit1(m);
  line[3] = digit10(s);
  line[4] = digit1(s);
  LCDprintChar(line);
}
#if GPS
void fill_line1_gps_lat(uint8_t sat) {
  int32_t aGPS_latitude = abs(GPS_coord[LAT]);
  strcpy_P(line1,PSTR(".---.------- #--"));
  //                0123456789012345
  line1[0] = GPS_coord[LAT]<0?'S':'N';
  if (sat) {
    //line1[13] = '#';
    line1[14] = digit10(GPS_numSat);
    line1[15] = digit1(GPS_numSat);
  } //                           987654321
  line1[1]  = '0' + aGPS_latitude  / 1000000000;
  line1[2]  = '0' + aGPS_latitude  / 100000000 - (aGPS_latitude/1000000000) * 10;
  line1[3]  = '0' + aGPS_latitude  / 10000000  - (aGPS_latitude/100000000)  * 10;
  line1[5]  = '0' + aGPS_latitude  / 1000000   - (aGPS_latitude/10000000)   * 10;
  line1[6]  = '0' + aGPS_latitude  / 100000    - (aGPS_latitude/1000000)    * 10;
  line1[7]  = '0' + aGPS_latitude  / 10000     - (aGPS_latitude/100000)     * 10;
  line1[8]  = '0' + aGPS_latitude  / 1000      - (aGPS_latitude/10000)      * 10;
  line1[9]  = '0' + aGPS_latitude  / 100       - (aGPS_latitude/1000)       * 10;
  line1[10] = '0' + aGPS_latitude  / 10        - (aGPS_latitude/100)        * 10;
  line1[11] = '0' + aGPS_latitude              - (aGPS_latitude/10)         * 10;
}
void fill_line2_gps_lon(uint8_t status) {
  int32_t aGPS_longitude = abs(GPS_coord[LON]);
  strcpy_P(line2,PSTR(".---.-------    "));
  //                0123456789012345
  line2[0] = GPS_coord[LON]<0?'W':'E';
  if (status) {
    line2[13] = (GPS_update ? 'U' : '.');
    //line2[15] = (1 ? 'P' : '.');
  }
  line2[1]  = '0' + aGPS_longitude / 1000000000;
  line2[2]  = '0' + aGPS_longitude / 100000000 - (aGPS_longitude/1000000000) * 10;
  line2[3]  = '0' + aGPS_longitude / 10000000  - (aGPS_longitude/100000000)  * 10;
  line2[5]  = '0' + aGPS_longitude / 1000000   - (aGPS_longitude/10000000)   * 10;
  line2[6]  = '0' + aGPS_longitude / 100000    - (aGPS_longitude/1000000)    * 10;
  line2[7]  = '0' + aGPS_longitude / 10000     - (aGPS_longitude/100000)     * 10;
  line2[8]  = '0' + aGPS_longitude / 1000      - (aGPS_longitude/10000)      * 10;
  line2[9]  = '0' + aGPS_longitude / 100       - (aGPS_longitude/1000)       * 10;
  line2[10] = '0' + aGPS_longitude / 10        - (aGPS_longitude/100)        * 10;
  line2[11] = '0' + aGPS_longitude             - (aGPS_longitude/10)         * 10;
}
#endif
```

```
void output_gyroX() {
  LCDprintInt16(imu.gyroData[0]); LCDprint(' ');
  outputSensor(DISPLAY_COLUMNS-10, imu.gyroData[0], GYROLIMIT);
}
void output_gyroY() {
  LCDprintInt16(imu.gyroData[1]); LCDprint(' ');
  outputSensor(DISPLAY_COLUMNS-10, imu.gyroData[1], GYROLIMIT);
}
void output_gyroZ() {
  LCDprintInt16(imu.gyroData[2]); LCDprint(' ');
  outputSensor(DISPLAY_COLUMNS-10, imu.gyroData[2], GYROLIMIT);
}
void output_accX() {
  LCDprintInt16(imu.accSmooth[0]); LCDprint(' ');
  outputSensor(DISPLAY_COLUMNS-10, imu.accSmooth[0], ACCLIMIT);
}
void output_accY() {
  LCDprintInt16(imu.accSmooth[1]); LCDprint(' ');
  outputSensor(DISPLAY_COLUMNS-10, imu.accSmooth[1], ACCLIMIT);
}
void output_accZ() {
  LCDprintInt16(imu.accSmooth[2]); LCDprint(' ');
  outputSensor(DISPLAY_COLUMNS-10, imu.accSmooth[2] - ACC_1G,
ACCLIMIT);
}

void output_debug0() { LCDprintChar("D1 "); LCDprintInt16(debug[0]); }
void output_debug1() { LCDprintChar("D2 "); LCDprintInt16(debug[1]); }
void output_debug2() { LCDprintChar("D3 "); LCDprintInt16(debug[2]); }
void output_debug3() { LCDprintChar("D4 "); LCDprintInt16(debug[3]); }


#if defined(DEBUG) || defined(DEBUG_FREE)
  #define PRINT_FREE_RAM  { \
    extern unsigned int __bss_end; \
    /*extern unsigned int __heap_start;*/ \
    extern void *__brkval; \
    int free_memory; \
    if((int)__brkval == 0) \
      free_memory = ((int)&free_memory) - ((int)&__bss_end); \
    else \
      free_memory = ((int)&free_memory) - ((int)__brkval); \
    strcpy_P(line1,PSTR(" Free ----")); \
    line1[6] = digit1000( free_memory ); \
    line1[7] = digit100( free_memory ); \
    line1[8] = digit10( free_memory ); \
    line1[9] = digit1( free_memory ); \
    LCDsetLine(1); LCDprintChar(line1); \
    telemetry = 0; \
  }
```

```
  #define PRINT_FREE_RAM_v2  { \
    const uint8_t *ptr = &_end; \
    uint16_t       free_memory = 0; \
    while(*ptr == 0xa5 && ptr <= &__stack) \
    {   ptr++; free_memory++; } \
    strcpy_P(line1,PSTR(" Free ----")); \
    line1[6] = digit1000( free_memory ); \
    line1[7] = digit100( free_memory ); \
    line1[8] = digit10( free_memory ); \
    line1[9] = digit1( free_memory ); \
    LCDsetLine(1); LCDprintChar(line1); \
    telemetry = 0; \
  }
#endif

/* ------------ DISPLAY_2LINES ------------------------------------*/
#ifdef DISPLAY_2LINES
void lcd_telemetry() {
  static uint8_t linenr = 0;
  switch (telemetry) { // output telemetry data
    uint16_t unit;
    uint8_t i;
#ifndef SUPPRESS_TELEMETRY_PAGE_1
    case 1: // button A on Textstar LCD -> angles
    case '1':
    if (linenr++ % 2) {
      fill_line1_deg();
      LCDsetLine(1);
      LCDprintChar(line1);
    } else {
      #ifdef POWERMETER_HARD
        LCDsetLine(2);
        output_AmaxA();
      #endif
    }
    break;
#endif
#ifndef SUPPRESS_TELEMETRY_PAGE_2
    case 2: // button B on Textstar LCD -> Voltage, PowerSum and power alarm trigger
value
    case '2':
    if (linenr++ % 2) {
      LCDsetLine(1);
      output_V();
    } else {
      LCDsetLine(2);
      output_mAh();
    }
    break;
#endif
```

```
#ifndef SUPPRESS_TELEMETRY_PAGE_3
    case 3: // button C on Textstar LCD -> cycle time
    case '3':
    if (linenr++ % 2) {
      LCDsetLine(1);
      output_cycle();
    } else {
#if (LOG_VALUES >= 2)
      LCDsetLine(2);
      output_cycleMinMax());
#endif
    }
    break;
#endif
#ifndef SUPPRESS_TELEMETRY_PAGE_4
    case 4: // button D on Textstar LCD -> sensors
    case '4':
    if (linenr++ % 2) {
      LCDsetLine(1);LCDprintChar("G "); //refresh line 1 of LCD
      outputSensor(4, imu.gyroData[0], GYROLIMIT); LCDprint(' ');
      outputSensor(4, imu.gyroData[1], GYROLIMIT); LCDprint(' ');
      outputSensor(4, imu.gyroData[2], GYROLIMIT);
    } else {
      LCDsetLine(2);LCDprintChar("A "); //refresh line 2 of LCD
      outputSensor(4, imu.accSmooth[0], ACCLIMIT); LCDprint(' ');
      outputSensor(4, imu.accSmooth[1], ACCLIMIT); LCDprint(' ');
      outputSensor(4, imu.accSmooth[2] - ACC_1G, ACCLIMIT);
    }
    break;
#endif
#ifndef SUPPRESS_TELEMETRY_PAGE_5
    case 5:
    case '5':
    if (linenr++ % 2) {
      LCDsetLine(1);
      output_fails();
    } else {
      LCDsetLine(2);
      output_annex();
    }
      break;
#endif
#ifndef SUPPRESS_TELEMETRY_PAGE_6
    case 6: // RX inputs
    case '6':
    if (linenr++ % 2) {
      strcpy_P(line1,PSTR("Roll Pitch Throt"));
      if (f.ARMED) line2[14] = 'A'; else line2[14] = 'a';
      if (failsafeCnt > 5) line2[15] = 'F'; else line2[15] = 'f';
      LCDsetLine(1);LCDprintChar(line1);
```

```
    } else {
      // 0123456789012345
      strcpy_P(line2,PSTR("---- ---- ----xx"));
      line2[0] = digit1000( rcData[ROLL] );
      line2[1] = digit100( rcData[ROLL] );
      line2[2] = digit10( rcData[ROLL] );
      line2[3] = digit1( rcData[ROLL] );
      line2[5] = digit1000( rcData[PITCH] );
      line2[6] = digit100( rcData[PITCH] );
      line2[7] = digit10( rcData[PITCH] );
      line2[8] = digit1( rcData[PITCH] );
      line2[10] = digit1000( rcData[THROTTLE] );
      line2[11] = digit100( rcData[THROTTLE] );
      line2[12] = digit10( rcData[THROTTLE] );
      line2[13] = digit1( rcData[THROTTLE] );
      LCDsetLine(2);LCDprintChar(line2);
    }
    break;
#endif
#ifndef SUPPRESS_TELEMETRY_PAGE_7
    case 7:
    case '7':
      #if GPS
      if (linenr++ % 2) {
        fill_line1_gps_lat(1); // including #sat
        LCDsetLine(1);LCDprintChar(line1);

      } else {
        fill_line2_gps_lon(1); // including status info
        LCDsetLine(2);LCDprintChar(line2);
      }
      #endif // case 7 : GPS
      break;
#endif
#ifndef SUPPRESS_TELEMETRY_PAGE_9
    case 9:
    case '9':
      LCDsetLine(1);
      LCDprintInt16(debug[0]); LCDprint(' ');
      LCDprintInt16(debug[1]); LCDprint(' ');
      LCDprintInt16(debug[2]); LCDprint(' ');
      LCDprintInt16(debug[3]); LCDprint(' ');
      break;
#endif

#if defined(LOG_VALUES) && defined(DEBUG)
    case 'R':
    //Reset logvalues
    cycleTimeMax = 0;// reset min/max on transition on->off
    cycleTimeMin = 65535;
```

```
    telemetry = 0;// no use to repeat this forever
    break;
#endif // case R
#if defined(DEBUG) || defined(DEBUG_FREE)
    case 'F':
    PRINT_FREE_RAM;
    break;
#endif // DEBUG
    // WARNING: if you add another case here, you should also add a case: in
Serial.pde, so users can access your case via terminal input
  } // end switch (telemetry)
} // end function lcd_telemetry

#endif // DISPLAY_2LINES
/* ------------ DISPLAY_MULTILINE ------------------------------------*/
#ifdef DISPLAY_MULTILINE
#ifndef SUPPRESS_TELEMETRY_PAGE_5
void outputMotorServo(uint8_t i, uint16_t unit) {
  #ifdef HELICOPTER
    static char outputNames[16][3] = {"M1", " 2"," 3", " 4", " 5", " 6", " 7", " 8",
      "S1", "S2","S3", "SN", "SL", "ST", "SR", "SM",};
  #else
    static char outputNames[16][3] = {"M1", " 2"," 3", " 4", " 5", " 6", " 7", " 8",
      "S1", "S2","S3", "S4", "S5", "S6", "S7", "S8",};
  #endif
  LCDprintChar(outputNames[i]);
  template7[1] = digit1000(unit);
  template7[2] = digit100(unit);
  template7[3] = digit10(unit);
  template7[4] = digit1(unit);
  LCDprintChar(template7);
  unit = constrain(unit,1000,2000);
  LCDbar(DISPLAY_COLUMNS-8, (unit-1000)/10 );
  LCDcrlf();
}
#endif
#ifdef LCD_TELEMETRY_PAGE1
  void (*page1_func_ptr[]) () = LCD_TELEMETRY_PAGE1 ;
#else
  void (*page1_func_ptr[]) () = {
    #ifdef VBAT
      output_V, //1
    #endif
    #ifdef POWERMETER
      output_mAh,
    #endif
    #ifdef VBAT
      output_Vmin,
    #endif
    output_errors_or_armedTime,
```

```
    output_checkboxstatus,
    #if BARO
      output_altitude,
    #endif
    #ifdef POWERMETER_HARD
      output_AmaxA,
    #endif
    #ifdef WATTS
      output_WmaxW,
    #endif
    output_uptime_cset,
  };
#endif
#ifdef LCD_TELEMETRY_PAGE9
  void (*page9_func_ptr[]) () = LCD_TELEMETRY_PAGE9 ;
#else
  void (*page9_func_ptr[]) () = {
      output_cycle,
      #if (LOG_VALUES >= 2)
        output_cycleMinMax,
      #endif
      output_fails,
      output_annex,
      #ifdef DEBUG
        output_debug0, output_debug1, output_debug2, output_debug3,
      #endif
  };
#endif
#ifdef LCD_TELEMETRY_PAGE2
  void (*page2_func_ptr[]) () = LCD_TELEMETRY_PAGE2 ;
#else
  void (*page2_func_ptr[]) () = { output_gyroX, output_gyroY, output_gyroZ,
output_accX, output_accY, output_accZ, };
#endif

void lcd_telemetry() {
  static uint8_t linenr = 0;
  #ifdef DISPLAY_FONT_DSIZE
    uint8_t offset = 0;
    #define POSSIBLE_OFFSET offset
  #else
    #define POSSIBLE_OFFSET 0
  #endif
  switch (telemetry) { // output telemetry data
    uint16_t unit;
    uint8_t i;
    case '0': // request to turn telemetry off - workaround, cannot enter binary zero \000
into string
      telemetry = 0;
      break;
```

```
#ifndef SUPPRESS_TELEMETRY_PAGE_1
  #ifdef DISPLAY_FONT_DSIZE
    case '!':
      { offset = MULTILINE_PRE+MULTILINE_POST; }
      // no break !!
  #endif
    case 1:// overall display
    case '1':
    {
      linenr++;
      linenr %= min(MULTILINE_PRE+MULTILINE_POST,
(sizeof(page1_func_ptr)/2) - POSSIBLE_OFFSET);
      LCDsetLine(linenr+1);
      (*page1_func_ptr [linenr + POSSIBLE_OFFSET] ) (); // not really linenumbers
      LCDcrlf();
      break;
    }
#endif
#ifndef SUPPRESS_TELEMETRY_PAGE_2
  #ifdef DISPLAY_FONT_DSIZE
    case '@':
      { offset = 3; }
      // no break !!
  #endif
    case 2: // sensor readings
    case '2':
    static char sensorNames[6][3] = {"Gx", "Gy", "Gz", "Ax", "Ay", "Az"};
    i = linenr++ % min(MULTILINE_PRE+MULTILINE_POST, 6 -
POSSIBLE_OFFSET);
    LCDsetLine(i + 1);
    LCDprintChar(sensorNames[i+POSSIBLE_OFFSET]);
    LCDprint(' ');
    (*page2_func_ptr [i+POSSIBLE_OFFSET] ) (); // not really linenumbers
    LCDcrlf();
    break;
#endif
#ifndef SUPPRESS_TELEMETRY_PAGE_3
  #ifdef DISPLAY_FONT_DSIZE
    case '#':
      { offset = MULTILINE_PRE+MULTILINE_POST; }
    // no break !!
  #endif
    case 3: // checkboxes and modes
    case '3':
    {
      i = linenr++ % min(MULTILINE_PRE+MULTILINE_POST,
CHECKBOXITEMS - POSSIBLE_OFFSET);
      LCDsetLine(i + 1);
      LCDprintChar(checkboxitemNames[i+POSSIBLE_OFFSET]);
      //LCDprintChar((PGM_P)(boxnames[index]));
```

```
      LCDprint(' ');
      LCDprint( rcOptions[i+POSSIBLE_OFFSET] ? 'X' : '.');
      LCDcrlf();
      break;
    }
#endif
#ifndef SUPPRESS_TELEMETRY_PAGE_4
  #ifdef DISPLAY_FONT_DSIZE
    case '$':
      { offset = 4; }
      // no break !!
  #endif
    case 4: // RX inputs
    case '4':
    {
      static char channelNames[8][4] = {"Ail", "Ele", "Yaw", "Thr", "Ax1", "Ax2",
"Ax3", "Ax4"};
      i = linenr++ % 8; // 8 channels
      LCDsetLine((i - POSSIBLE_OFFSET)%8 + 1);
      //strcpy_P(line1,PSTR("-Thr ---- "));
      //                  0123456789.12345
      template3[0] = ( '0' + i+1);// channel numbering [1;8]
      LCDprintChar(template3);
      LCDprintChar(channelNames[i]);
      unit = rcData[i];
      template7[1] = digit1000(unit);
      template7[2] = digit100(unit);
      template7[3] = digit10(unit);
      template7[4] = digit1(unit);
      LCDprintChar(template7);
      unit = constrain(rcData[i],1000,2000);
      LCDbar(DISPLAY_COLUMNS-11, (unit-1000)/10 );
      LCDcrlf();
      break;
    }
#endif
#ifndef SUPPRESS_TELEMETRY_PAGE_5
  #ifdef DISPLAY_FONT_DSIZE
    case '%':
      { offset = MULTILINE_PRE+MULTILINE_POST; }
      // no break !!
  #endif
    case 5: // outputs motors+servos
    case '5':
    {
//      static char outputNames[16][3] = {"M1", " 2"," 3", " 4", " 5", " 6", " 7", " 8",
//          "S1", "S2","S3", "S4", "S5", "S6", "S7", "S8",};
      static uint8_t index = 0;
      i = index++ % 16;
      if (i == 0) linenr = 0; //vt100 starts linenumbering @1
```

```
      LCDsetLine((linenr - POSSIBLE_OFFSET)%8 +1);
      if (i < 8) {
       if (i < NUMBER_MOTOR) {
        outputMotorServo(i, motor[i]);
        linenr++;
       } else {
        index = 8;
       }
      } else {
       uint8_t j = i-7; // [8;15] -> [1;8]
       #if defined(PRI_SERVO_FROM) && defined(SEC_SERVO_FROM)
        if ((PRI_SERVO_FROM <= j && PRI_SERVO_TO >= j) ||
(SEC_SERVO_FROM <= j && SEC_SERVO_TO >= j))
       #elif defined(PRI_SERVO_FROM)
        if (j < PRI_SERVO_FROM) index = 7 + PRI_SERVO_FROM;
        else if (j > PRI_SERVO_TO) index = 16;
        else // (PRI_SERVO_FROM <= j && PRI_SERVO_TO >= j)
       #endif
       #if defined(PRI_SERVO_FROM) || defined(SEC_SERVO_FROM)
        {
         outputMotorServo(i, servo[j-1]);
         linenr++;
         break;
        }
       #endif

      }
      break;
    }
#endif // page 5

#ifndef SUPPRESS_TELEMETRY_PAGE_6
  #if defined(VBAT_CELLS)
    #ifdef DISPLAY_FONT_DSIZE
    case '^':
      { offset = 4; }
      // no break !!
    #endif
    case 6: // alarms states
    case '6':
      {
       i = linenr++ % VBAT_CELLS_NUM; // VBAT_CELLS_NUM cells
       LCDsetLine((i - POSSIBLE_OFFSET)% VBAT_CELLS_NUM + 1);
       strcpy_P(line1,PSTR("_:-.-V __._V"));
       //                  0123456789.12345
       line1[0] = digit1(i+1);
       uint16_t v = analog.vbatcells[i];
       if (i>0) v = (analog.vbatcells[i] > analog.vbatcells[i-1] ? analog.vbatcells[i] -
analog.vbatcells[i-1] : 0);
       line1[2] = digit10(v);
```

```
        line1[4] =  digit1(v);
        line1[7] = digit100(analog.vbatcells[i]);
        line1[8] =  digit10(analog.vbatcells[i]);
        line1[10] =   digit1(analog.vbatcells[i]);
        //      #ifndef OLED_I2C_128x64
        //        if (analog.vbat < conf.vbatlevel_warn1) { LCDattributesReverse(); }
        //      #endif
        if (v > VBATNOMINAL/VBAT_CELLS_NUM) v =
VBATNOMINAL/VBAT_CELLS_NUM;
        LCDbar(DISPLAY_COLUMNS-12,
(v*100*VBAT_CELLS_NUM)/VBATNOMINAL );
        //      LCDattributesOff(); // turn Reverse off for rest of display
        LCDprintChar(line1);
        LCDcrlf();
        break;
      }
  #endif // vbat.cells
#endif // page 6

#ifndef SUPPRESS_TELEMETRY_PAGE_7
  #if GPS
    #ifdef DISPLAY_FONT_DSIZE
    case '&':
      { offset = MULTILINE_PRE+MULTILINE_POST; }
      // no break !!
    #endif
    case 7: // GPS
    case '7':
      linenr++;
      linenr %= 6;
      LCDsetLine(linenr+1);
      switch (linenr + POSSIBLE_OFFSET) {
        case 0: // lat
          fill_line1_gps_lat(0); // skip #sat
          LCDprintChar(line1);
          break;
        case 1: // lon
          fill_line2_gps_lon(0);
          LCDprintChar(line2);
          break;
        case 2: // # Sats
          strcpy_P(line1,PSTR("-- Sats"));
          //                  0123456789012345
          line1[0] = digit10(GPS_numSat);
          line1[1] = digit1(GPS_numSat);
          LCDprintChar(line1);
          break;
//        case 3: //
//          strcpy_P(line1,PSTR("Status "));
//          //                  0123456789012345
```

```
//          LCDprintChar(line1);
//          if (1)
//            LCDprintChar("OK");
//          else {
//            LCDattributesReverse();
//            LCDprintChar("KO");
//            LCDattributesOff();
//          }
//        break;
        case 4: // gps speed
        {
          uint8_t v = (GPS_speed * 0.036f);
          strcpy_P(line1,PSTR("--km/h max--km/h"));
          //                  0123456789012345
          line1[0] = digit10(v);
          line1[1] = digit1(v);
          v = (GPS_speedMax * 0.036f);
          line1[10] = digit10(v);
          line1[11] = digit1(v);
          LCDprintChar(line1);
          break;
        }
        case 5: // vbat
          output_V();
          break;
      }
      LCDcrlf();
    break;
  #endif // gps
#endif // page 7

#ifndef SUPPRESS_TELEMETRY_PAGE_8
  #ifdef DISPLAY_FONT_DSIZE
      case '*':
      { offset = 5; }
      // no break !!
  #endif
    case 8: // alarms states
    case '8':
    //   123456789.1234567890
    static char alarmsNames[][12] = {
        "0: toggle  ",
        "1: failsafe",
        "2: noGPS   ",
        "3: beeperOn",
        "4: pMeter  ",
        "5: runtime ",
        "6: vBat    ",
        "7: confirma",
        "8: Acc     ",
```

```
        "9: I2Cerror" };
    linenr++;
    linenr %= min(MULTILINE_PRE+MULTILINE_POST, 10 -
POSSIBLE_OFFSET);
    LCDsetLine(linenr+1);
    // [linenr + POSSIBLE_OFFSET]
    LCDprintChar( alarmsNames[linenr + POSSIBLE_OFFSET] );
    LCDprint(' ');
    LCDprint( digit1( alarmArray[linenr + POSSIBLE_OFFSET] ) );
    LCDcrlf();
    break;
#endif // page 8

#ifndef SUPPRESS_TELEMETRY_PAGE_9
  #ifdef DISPLAY_FONT_DSIZE
      case '(':
      { offset = 4; }
      // no break !!
  #endif
    case 9: // diagnostics
    case '9':
    linenr++;
    linenr %= min(MULTILINE_PRE+MULTILINE_POST,
(sizeof(page9_func_ptr)/2) - POSSIBLE_OFFSET);
    LCDsetLine(linenr+1);
    (*page9_func_ptr [linenr + POSSIBLE_OFFSET] ) (); // not really linenumbers
    LCDcrlf();
    break;
#endif // page 9
#ifndef SUPPRESS_TELEMETRY_PAGE_R
    case 'R':
    {
      //Reset logvalues
      #if defined(LOG_VALUES) && (LOG_VALUES >= 2)
        cycleTimeMax = 0;
        cycleTimeMin = 65535;
      #endif
      #if BARO
        #if defined(LOG_VALUES)
          BAROaltMax = 0;
        #endif
      #endif
      #if defined(FAILSAFE)
          failsafeEvents = 0; // reset failsafe counter
      #endif
      i2c_errors_count = 0;
      f.OK_TO_ARM = 1; // allow arming again
      telemetry = 0; // no use to repeat this forever
      break;
    }
```

```
#endif // case R
#if defined(DEBUG) || defined(DEBUG_FREE)
    case 'F':
      PRINT_FREE_RAM;
      break;
#endif // DEBUG
    // WARNING: if you add another case here, you should also add a case: in
Serial.pde, so users can access your case via terminal input
  } // end switch (telemetry)
} // end function lcd_telemetry

#endif // DISPLAY_MULTILINE

void toggle_telemetry(uint8_t t) {
  if (telemetry == t) telemetry = 0;
  else {
    telemetry = t;
    #if defined( OLED_I2C_128x64)
      if (telemetry != 0) i2c_OLED_init();
    #elif defined(OLED_DIGOLE)
      if (telemetry != 0) i2c_OLED_DIGOLE_init();
    #endif
    LCDclear();
  }
}
#endif //  LCD_TELEMETRY

#ifdef LOG_PERMANENT
  void dumpPLog(uint8_t full) {
    #ifdef HAS_LCD
      /*LCDclear();*/ LCDnextline();
      LCDprintChar("LastOff   "); LCDprintChar(plog.running ? "KO" : "ok");
LCDnextline();
      LCDprintChar("#On      "); LCDprintInt16(plog.start); LCDnextline();
      LCDprintChar("Life[min]"); LCDprintInt16(plog.lifetime/60); LCDnextline();
      if (full) {
        #ifdef DEBUG
          LCDprintChar("#arm    "); LCDprintInt16(plog.arm); LCDnextline();
          LCDprintChar("#disarm"); LCDprintInt16(plog.disarm); LCDnextline();
          LCDprintChar("last[s]"); LCDprintInt16(plog.armed_time/1000000);
LCDnextline();
          LCDprintChar("#fail@dis"); LCDprintInt16(plog.failsafe); LCDnextline();
          LCDprintChar("#i2c@dis "); LCDprintInt16(plog.i2c); LCDnextline();
          //           0123456789012345
        #endif
      }
      /*strcpy_P(line2,PSTR("Fail --- i2c ---"));
      line2[5] = digit100(plog.failsafe);
      line2[6] = digit10(plog.failsafe);
      line2[7] = digit1(plog.failsafe);
```

```
    line2[13] = digit100(plog.i2c);
    line2[14] = digit10(plog.i2c);
    line2[15] = digit1(plog.i2c);
    LCDprintChar(line2); LCDnextline();*/
    delay(4000);
  #endif
  #ifdef LOG_PERMANENT_SERVICE_LIFETIME
    serviceCheckPLog();
  #endif
  #ifdef HAS_LCD
    LCDclear();
  #endif
}

void LCDnextline(void) {
  #if ( defined(DISPLAY_MULTILINE) )
    lnr++;
    if (lnr > (MULTILINE_PRE+MULTILINE_POST)) {
      lnr = 1;
      delay(4000);
      LCDclear();
    }
    LCDsetLine(lnr);
    LCD_FLUSH;
  #elif ( defined(DISPLAY_2LINES))
    #if (! (defined(LCD_TTY)  ) )
      delay(600);
    #endif
    #ifdef HAS_LCD
    LCDprintChar("\r\n");
    #endif
  #else
    // no LCD, nothing to do here
  #endif
}

#ifdef LOG_PERMANENT_SERVICE_LIFETIME
void serviceCheckPLog(void) {
  if ( (!f.ARMED) && (plog.lifetime >
LOG_PERMANENT_SERVICE_LIFETIME) ){
    for (uint8_t i = 0; i<max(1, min(9,(plog.lifetime-
LOG_PERMANENT_SERVICE_LIFETIME)>>10 )); i++) {
      #ifdef HAS_LCD
        LCDprintChar("SERVICE lifetime"); LCDnextline();
      #endif
      blinkLED(5,200,5);
      delay(5000);
    }
    SET_ALARM(ALRM_FAC_CONFIRM, ALRM_LVL_CONFIRM_ELSE);
  }
```

```
  }
  #endif // LOG_PERMANENT_SERVICE_LIFETIME

#endif // LOG_PERMANENT
```
καρτέλα LCD.h
```
#ifndef LCD_H_
#define LCD_H_

void configurationLoop();
void LCDprint(uint8_t i);
void lcd_telemetry();
void initLCD();
void i2c_OLED_DIGOLE_init ();
void i2c_OLED_init();
void LCDclear();
void toggle_telemetry(uint8_t t);
void dumpPLog(uint8_t full);

/* helper functions */
void LCDprintInt16(int16_t v);
void LCDcrlf();

void print_uptime(uint16_t sec);
void output_checkboxitems();

/* candidates for telemetry pages outputs */
void output_fails();
void output_annex();

void output_V();
void output_mAh();
void output_AmaxA();
void output_errors_or_armedTime();
void output_WmaxW();
void output_uptime_cset();
void output_altitude();
void output_checkboxstatus();
void output_cycleMinMax();
void output_cycle();
void output_gyroX();
void output_gyroY();
void output_gyroZ();
void output_accX();
void output_accY();
void output_accZ();

void output_debug0() ;
void output_debug1() ;
void output_debug2() ;
void output_debug3() ;
```

```
#endif /* LCD_H_ */
```

καρτέλα MultiWii.cpp

```
/*
MultiWiiCopter by Alexandre Dubus
www.multiwii.com
March  2015     V2.4
 This program is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 any later version. see <http://www.gnu.org/licenses/>
*/

#include <avr/io.h>

#include "Arduino.h"
#include "config.h"
#include "def.h"
#include "types.h"
#include "MultiWii.h"
#include "Alarms.h"
#include "EEPROM.h"
#include "IMU.h"
#include "LCD.h"
#include "Output.h"
#include "RX.h"
#include "Sensors.h"
#include "Serial.h"
#include "GPS.h"
#include "Protocol.h"

#include <avr/pgmspace.h>

/********** RC alias *****************/

const char pidnames[] PROGMEM =
  "ROLL;"
  "PITCH;"
  "YAW;"
  "ALT;"
  "Pos;"
  "PosR;"
  "NavR;"
  "LEVEL;"
  "MAG;"
  "VEL;"
;

const char boxnames[] PROGMEM = // names for dynamic generation of config GUI
  "ARM;"
```

```
  #if ACC
    "ANGLE;"
    "HORIZON;"
  #endif
  #if BARO && (!defined(SUPPRESS_BARO_ALTHOLD))
    "BARO;"
  #endif
  #ifdef VARIOMETER
    "VARIO;"
  #endif
  "MAG;"
  #if defined(HEADFREE)
  "HEADFREE;"
  "HEADADJ;"
#endif
#if defined(SERVO_TILT) || defined(GIMBAL)|| defined(SERVO_MIX_TILT)
  "CAMSTAB;"
#endif
#if defined(CAMTRIG)
  "CAMTRIG;"
#endif
#if GPS
  "GPS HOME;"
  "GPS HOLD;"
#endif
#if defined(FIXEDWING) || defined(HELICOPTER)
  "PASSTHRU;"
#endif
#if defined(BUZZER)
  "BEEPER;"
#endif
#if defined(LED_FLASHER)
  "LEDMAX;"
  "LEDLOW;"
#endif
#if defined(LANDING_LIGHTS_DDR)
  "LLIGHTS;"
#endif
#ifdef INFLIGHT_ACC_CALIBRATION
  "CALIB;"
#endif
#ifdef GOVERNOR_P
  "GOVERNOR;"
#endif
#ifdef OSD_SWITCH
  "OSD SW;"
#endif
#if GPS
  "MISSION;"
  "LAND;"
```

```
#endif
  ;

const uint8_t boxids[] PROGMEM = {// permanent IDs associated to boxes. This
way, you can rely on an ID number to identify a BOX function.
  0, //"ARM;"
  #if ACC
    1, //"ANGLE;"
    2, //"HORIZON;"
  #endif
  #if BARO && (!defined(SUPPRESS_BARO_ALTHOLD))
    3, //"BARO;"
  #endif
  #ifdef VARIOMETER
    4, //"VARIO;"
  #endif
  5, //"MAG;"
#if defined(HEADFREE)
  6, //"HEADFREE;"
  7, //"HEADADJ;"
#endif
#if defined(SERVO_TILT) || defined(GIMBAL)|| defined(SERVO_MIX_TILT)
  8, //"CAMSTAB;"
#endif
#if defined(CAMTRIG)
  9, //"CAMTRIG;"
#endif
#if GPS
  10, //"GPS HOME;"
  11, //"GPS HOLD;"
#endif
#if defined(FIXEDWING) || defined(HELICOPTER)
  12, //"PASSTHRU;"
#endif
#if defined(BUZZER)
  13, //"BEEPER;"
#endif
#if defined(LED_FLASHER)
  14, //"LEDMAX;"
  15, //"LEDLOW;"
#endif
#if defined(LANDING_LIGHTS_DDR)
  16, //"LLIGHTS;"
#endif
#ifdef INFLIGHT_ACC_CALIBRATION
  17, //"CALIB;"
#endif
#ifdef GOVERNOR_P
  18, //"GOVERNOR;"
#endif
```

```
#ifdef OSD_SWITCH
  19, //"OSD_SWITCH;"
#endif
#if GPS
  20, //"MISSION;"
  21, //"LAND;"
#endif
  };


uint32_t currentTime = 0;
uint16_t previousTime = 0;
uint16_t cycleTime = 0;     // this is the number in micro second to achieve a full loop, it can differ a little and is taken into account in the PID loop
uint16_t calibratingA = 0;  // the calibration is done in the main loop. Calibrating decreases at each cycle down to 0, then we enter in a normal mode.
uint16_t calibratingB = 0;  // baro calibration = get new ground pressure value
uint16_t calibratingG;
int16_t  magHold,headFreeModeHold; // [-180;+180]
uint8_t  vbatMin = VBATNOMINAL;  // lowest battery voltage in 0.1V steps
uint8_t  rcOptions[CHECKBOXITEMS];
int32_t  AltHold; // in cm
int16_t  sonarAlt;
int16_t  BaroPID = 0;
int16_t  errorAltitudeI = 0;

// **************
// gyro+acc IMU
// **************
int16_t gyroZero[3] = {0,0,0};

imu_t imu;

analog_t analog;

alt_t alt;

att_t att;

#if defined(ARMEDTIMEWARNING)
  uint32_t  ArmedTimeWarningMicroSeconds = 0;
#endif

int16_t  debug[4];

flags_struct_t f;

//for log
#if defined(LOG_VALUES) || defined(LCD_TELEMETRY)
  uint16_t cycleTimeMax = 0;       // highest ever cycle timen
```

```
  uint16_t cycleTimeMin = 65535;   // lowest ever cycle timen
  int32_t  BAROaltMax;             // maximum value
  uint16_t GPS_speedMax = 0;       // maximum speed from gps
  #ifdef POWERMETER_HARD
    uint16_t powerValueMaxMAH = 0;
  #endif
  #if defined(WATTS)
    uint16_t wattsMax = 0;
  #endif
#endif
#if defined(LOG_VALUES) || defined(LCD_TELEMETRY) ||
defined(ARMEDTIMEWARNING) || defined(LOG_PERMANENT)
  uint32_t armedTime = 0;
#endif

int16_t  i2c_errors_count = 0;


#if defined(THROTTLE_ANGLE_CORRECTION)
  int16_t throttleAngleCorrection = 0; // correction of throttle in lateral wind,
  int8_t  cosZ = 100;                  // cos(angleZ)*100
#endif



// **********************
//Automatic ACC Offset Calibration
// **********************
#if defined(INFLIGHT_ACC_CALIBRATION)
  uint16_t InflightcalibratingA = 0;
  int16_t AccInflightCalibrationArmed;
  uint16_t AccInflightCalibrationMeasurementDone = 0;
  uint16_t AccInflightCalibrationSavetoEEProm = 0;
  uint16_t AccInflightCalibrationActive = 0;
#endif

// **********************
// power meter
// **********************
#if defined(POWERMETER) || ( defined(LOG_VALUES) && (LOG_VALUES >=
3) )
  uint32_t pMeter[PMOTOR_SUM + 1];  // we use [0:7] for eight motors,one extra for
sum
  uint8_t pMeterV;                  // dummy to satisfy the paramStruct logic in
ConfigurationLoop()
  uint32_t pAlarm;                  // we scale the eeprom value from [0:255] to this value
we can directly compare to the sum in pMeter[6]
  uint16_t powerValue = 0;          // last known current
#endif
uint16_t intPowerTrigger1;
```

```
// **********************
// telemetry
// **********************
#if defined(LCD_TELEMETRY)
  uint8_t telemetry = 0;
  uint8_t telemetry_auto = 0;
  int16_t annex650_overrun_count = 0;
#endif
#ifdef LCD_TELEMETRY_STEP
  char telemetryStepSequence []  = LCD_TELEMETRY_STEP;
  uint8_t telemetryStepIndex = 0;
#endif

// ******************
// rc functions
// ******************
#define ROL_LO  (1<<(2*ROLL))
#define ROL_CE  (3<<(2*ROLL))
#define ROL_HI  (2<<(2*ROLL))
#define PIT_LO  (1<<(2*PITCH))
#define PIT_CE  (3<<(2*PITCH))
#define PIT_HI  (2<<(2*PITCH))
#define YAW_LO  (1<<(2*YAW))
#define YAW_CE  (3<<(2*YAW))
#define YAW_HI  (2<<(2*YAW))
#define THR_LO  (1<<(2*THROTTLE))
#define THR_CE  (3<<(2*THROTTLE))
#define THR_HI  (2<<(2*THROTTLE))

int16_t failsafeEvents = 0;
volatile int16_t failsafeCnt = 0;

int16_t rcData[RC_CHANS];    // interval [1000;2000]
int16_t rcSerial[8];         // interval [1000;2000] - is rcData coming from MSP
int16_t rcCommand[4];        // interval [1000;2000] for THROTTLE and [-500;+500]
for ROLL/PITCH/YAW
uint8_t rcSerialCount = 0;   // a counter to select legacy RX when there is no more
MSP rc serial data
int16_t lookupPitchRollRC[5];// lookup table for expo & RC rate PITCH+ROLL
uint16_t lookupThrottleRC[11];// lookup table for expo & mid THROTTLE

#if defined(SERIAL_RX)
  volatile uint8_t  spekFrameFlags;
  volatile uint32_t spekTimeLast;
  uint8_t  spekFrameDone;
#endif

#if defined(OPENLRSv2MULTI)
  uint8_t pot_P,pot_I; // OpenLRS onboard potentiometers for P and I trim or other
```

```
usages
#endif


// *************************
// motor and servo functions
// *************************
int16_t axisPID[3];
int16_t motor[8];
int16_t servo[8] = {1500,1500,1500,1500,1500,1500,1500,1000};

// *************************
// EEPROM Layout definition
// *************************
static uint8_t dynP8[2], dynD8[2];

global_conf_t global_conf;

conf_t conf;

#ifdef LOG_PERMANENT
  plog_t plog;
#endif

// ***********************
// GPS common variables, no need to put them in defines, since compiller will
optimize out unused variables
// ***********************
#if GPS
  gps_conf_struct GPS_conf;
#endif
  int16_t  GPS_angle[2] = { 0, 0};                  // the angles that must be applied for
GPS correction
  int32_t  GPS_coord[2];
  int32_t  GPS_home[2];
  int32_t  GPS_hold[2];
  int32_t  GPS_prev[2];                           //previous pos
  int32_t  GPS_poi[2];
  uint8_t  GPS_numSat;
  uint16_t GPS_distanceToHome;                      // distance to home  - unit: meter
  int16_t  GPS_directionToHome;                     // direction to home - unit: degree
  int32_t  GPS_directionToPoi;
  uint16_t GPS_altitude;                          // GPS altitude      - unit: meter
  uint16_t GPS_speed;                             // GPS speed         - unit: cm/s
  uint8_t  GPS_update = 0;                          // a binary toogle to distinct a GPS
position update
  uint16_t GPS_ground_course = 0;                   //                   - unit: degree*10

  //uint8_t GPS_mode  = GPS_MODE_NONE; // contains the current selected gps
flight mode --> moved to the f. structure
```

```
  uint8_t NAV_state = 0; // NAV_STATE_NONE;  /// State of the nav engine
  uint8_t NAV_error = 0; // NAV_ERROR_NONE;
  uint8_t prv_gps_modes = 0;            /// GPS_checkbox items packed into 1 byte for
checking GPS mode changes
  uint32_t nav_timer_stop = 0;          /// common timer used in navigation (contains
the desired stop time in millis()
  uint16_t nav_hold_time;               /// time in seconds to hold position
  uint8_t NAV_paused_at = 0;            // This contains the mission step where poshold
paused the runing mission.

  uint8_t next_step = 1;                /// The mission step which is upcoming it equals
with the mission_step stored in EEPROM
  int16_t jump_times = -10;
#if GPS
  mission_step_struct mission_step;
#endif

  // The desired bank towards North (Positive) or South (Negative) : latitude
  // The desired bank towards East (Positive) or West (Negative)   : longitude
  int16_t  nav[2];
  int16_t  nav_rated[2];    //Adding a rate controller to the navigation to make it
smoother

  // The orginal altitude used as base our new altitude during nav
  int32_t original_altitude;
  //This is the target what we want to reach
  int32_t target_altitude;
  //This is the interim value which is feeded into the althold controller
  int32_t alt_to_hold;

  uint32_t alt_change_timer;
  int8_t alt_change_flag;
  uint32_t alt_change;

uint8_t alarmArray[ALRM_FAC_SIZE];          // array

#if BARO
  int32_t baroPressure;
  int16_t baroTemperature;
  int32_t baroPressureSum;
#endif

void annexCode() { // this code is excetuted at each loop and won't interfere with
control loop if it lasts less than 650 microseconds
  static uint32_t calibratedAccTime;
  uint16_t tmp,tmp2;
  uint8_t axis,prop1,prop2;

  // PITCH & ROLL only dynamic PID adjustemnt,  depending on throttle value (or
collective.pitch value for heli)
```

```
  #ifdef HELICOPTER
    #define DYN_THR_PID_CHANNEL COLLECTIVE_PITCH
  #else
    #define DYN_THR_PID_CHANNEL THROTTLE
  #endif
  prop2 = 128; // prop2 was 100, is 128 now
  if (rcData[DYN_THR_PID_CHANNEL]>1500) { // breakpoint is fix: 1500
    if (rcData[DYN_THR_PID_CHANNEL]<2000) {
      prop2 -=  ((uint16_t)conf.dynThrPID*(rcData[DYN_THR_PID_CHANNEL]-
1500)>>9); //  /512 instead of /500
    } else {
      prop2 -=  conf.dynThrPID;
    }
  }

  for(axis=0;axis<3;axis++) {
    tmp = min(abs(rcData[axis]-MIDRC),500);
    #if defined(DEADBAND)
      if (tmp>DEADBAND) { tmp -= DEADBAND; }
      else { tmp=0; }
    #endif
    if(axis!=2) { //ROLL & PITCH
      tmp2 = tmp>>7; // 500/128 = 3.9  => range [0;3]
      rcCommand[axis] = lookupPitchRollRC[tmp2] + ((tmp-(tmp2<<7)) *
(lookupPitchRollRC[tmp2+1]-lookupPitchRollRC[tmp2])>>7);
      prop1 = 128-((uint16_t)conf.rollPitchRate*tmp>>9); // prop1 was 100, is 128 now
-- and /512 instead of /500
      prop1 = (uint16_t)prop1*prop2>>7; // prop1: max is 128   prop2: max is 128
result prop1: max is 128
      dynP8[axis] = (uint16_t)conf.pid[axis].P8*prop1>>7; // was /100, is /128 now
      dynD8[axis] = (uint16_t)conf.pid[axis].D8*prop1>>7; // was /100, is /128 now
    } else {      // YAW
      rcCommand[axis] = tmp;
    }
    if (rcData[axis]<MIDRC) rcCommand[axis] = -rcCommand[axis];
  }
  tmp = constrain(rcData[THROTTLE],MINCHECK,2000);
  tmp = (uint32_t)(tmp-MINCHECK)*2559/(2000-MINCHECK); //
[MINCHECK;2000] -> [0;2559]
  tmp2 = tmp/256; // range [0;9]
  rcCommand[THROTTLE] = lookupThrottleRC[tmp2] + (tmp-tmp2*256) *
(lookupThrottleRC[tmp2+1]-lookupThrottleRC[tmp2]) / 256; // [0;2559] -> expo ->
[conf.minthrottle;MAXTHROTTLE]
  #if defined(HEADFREE)
    if(f.HEADFREE_MODE) { //to optimize
      float radDiff = (att.heading - headFreeModeHold) * 0.0174533f; // where PI/180
~= 0.0174533
      float cosDiff = cos(radDiff);
      float sinDiff = sin(radDiff);
      int16_t rcCommand_PITCH = rcCommand[PITCH]*cosDiff +
```

```
rcCommand[ROLL]*sinDiff;
      rcCommand[ROLL] =  rcCommand[ROLL]*cosDiff -
rcCommand[PITCH]*sinDiff;
      rcCommand[PITCH] = rcCommand_PITCH;
    }
  #endif

  // query at most one multiplexed analog channel per MWii cycle
  static uint8_t analogReader =0;
  switch (analogReader++ % (3+VBAT_CELLS_NUM)) {
  case 0:
  {
    #if defined(POWERMETER_HARD)
      static uint32_t lastRead = currentTime;
      static uint8_t ind = 0;
      static uint16_t pvec[PSENSOR_SMOOTH], psum;
      uint16_t p =  analogRead(PSENSORPIN);
      //LCDprintInt16(p); LCDcrlf();
      //debug[0] = p;
      #if PSENSOR_SMOOTH != 1
        psum += p;
        psum -= pvec[ind];
        pvec[ind++] = p;
        ind %= PSENSOR_SMOOTH;
        p = psum / PSENSOR_SMOOTH;
      #endif
      powerValue = ( conf.psensornull > p ? conf.psensornull - p : p - conf.psensornull);
// do not use abs(), it would induce implicit cast to uint and overrun
      analog.amperage = ((uint32_t)powerValue * conf.pint2ma) / 100; // [100mA]
//old (will overflow for 65A: powerValue * conf.pint2ma; // [1mA]
      pMeter[PMOTOR_SUM] += ((currentTime-lastRead) *
(uint32_t)((uint32_t)powerValue*conf.pint2ma))/100000; // [10 mA * msec]
      lastRead = currentTime;
    #endif // POWERMETER_HARD
    break;
  }

  case 1:
  {
    #if defined(VBAT) && !defined(VBAT_CELLS)
      static uint8_t ind = 0;
      static uint16_t vvec[VBAT_SMOOTH], vsum;
      uint16_t v = analogRead(V_BATPIN);
      #if VBAT_SMOOTH == 1
        analog.vbat = (v*VBAT_PRESCALER) / conf.vbatscale + VBAT_OFFSET; //
result is Vbatt in 0.1V steps
      #else
        vsum += v;
        vsum -= vvec[ind];
        vvec[ind++] = v;
```

```
      ind %= VBAT_SMOOTH;
      #if VBAT_SMOOTH == VBAT_PRESCALER
        analog.vbat = vsum / conf.vbatscale + VBAT_OFFSET; // result is Vbatt in
0.1V steps
      #elif VBAT_SMOOTH < VBAT_PRESCALER
        analog.vbat = (vsum * (VBAT_PRESCALER/VBAT_SMOOTH)) /
conf.vbatscale + VBAT_OFFSET; // result is Vbatt in 0.1V steps
      #else
        analog.vbat = ((vsum /VBAT_SMOOTH) * VBAT_PRESCALER) /
conf.vbatscale + VBAT_OFFSET; // result is Vbatt in 0.1V steps
      #endif
    #endif
   #endif // VBAT
   break;
  }
  case 2:
  {
  #if defined(RX_RSSI)
    static uint8_t ind = 0;
    static uint16_t rvec[RSSI_SMOOTH], rsum, r;

    // http://www.multiwii.com/forum/viewtopic.php?f=8&t=5530
    #if defined(RX_RSSI_CHAN)
      uint16_t rssi_Input = constrain(rcData[RX_RSSI_CHAN],1000,2000);
      r = map((uint16_t)rssi_Input , 1000, 2000, 0, 1023);
    #else
      r = analogRead(RX_RSSI_PIN);
    #endif

    #if RSSI_SMOOTH == 1
      analog.rssi = r;
    #else
      rsum += r;
      rsum -= rvec[ind];
      rvec[ind++] = r;
      ind %= RSSI_SMOOTH;
      r = rsum / RSSI_SMOOTH;
      analog.rssi = r;
    #endif
  #endif // RX RSSI
  break;
  }
  default: // here analogReader >=4, because of ++ in switch()
  {
    #if defined(VBAT) && defined(VBAT_CELLS)
      if ( (analogReader<4) || (analogReader>4+VBAT_CELLS_NUM-1) ) break;
      uint8_t ind = analogReader-4;
      static uint16_t vbatcells_pins[VBAT_CELLS_NUM] = VBAT_CELLS_PINS;
      static uint8_t  vbatcells_offset[VBAT_CELLS_NUM] =
VBAT_CELLS_OFFSETS;
```

```
      static uint8_t  vbatcells_div[VBAT_CELLS_NUM] = VBAT_CELLS_DIVS;
      uint16_t v = analogRead(vbatcells_pins[ind]);
      analog.vbatcells[ind] = vbatcells_offset[ind] + (v << 2) / vbatcells_div[ind]; //
result is Vbatt in 0.1V steps
      if (ind == VBAT_CELLS_NUM -1) analog.vbat = analog.vbatcells[ind];
    #endif // VBAT) && defined(VBAT_CELLS)
    break;
  } // end default
  } // end of switch()

#if defined( POWERMETER_HARD ) && (defined(LOG_VALUES) ||
defined(LCD_TELEMETRY))
  if (analog.amperage > powerValueMaxMAH) powerValueMaxMAH =
analog.amperage;
#endif

#if defined(WATTS)
  analog.watts = (analog.amperage * analog.vbat) / 100; // [0.1A] * [0.1V] / 100 =
[Watt]
  #if defined(LOG_VALUES) || defined(LCD_TELEMETRY)
    if (analog.watts > wattsMax) wattsMax = analog.watts;
  #endif
#endif

  #if defined(BUZZER)
    alarmHandler(); // external buzzer routine that handles buzzer events globally now
  #endif


  if ( (calibratingA>0 && ACC ) || (calibratingG>0) ) { // Calibration phasis
    LEDPIN_TOGGLE;
  } else {
    if (f.ACC_CALIBRATED) {LEDPIN_OFF;}
    if (f.ARMED) {LEDPIN_ON;}
  }

  #if defined(LED_RING)
    static uint32_t LEDTime;
    if ( currentTime > LEDTime ) {
      LEDTime = currentTime + 50000;
      i2CLedRingState();
    }
  #endif

  #if defined(LED_FLASHER)
    auto_switch_led_flasher();
  #endif

  if ( currentTime > calibratedAccTime ) {
    if (! f.SMALL_ANGLES_25) {
```

```
      // the multi uses ACC and is not calibrated or is too much inclinated
      f.ACC_CALIBRATED = 0;
      LEDPIN_TOGGLE;
      calibratedAccTime = currentTime + 100000;
    } else {
      f.ACC_CALIBRATED = 1;
    }
  }

  #if !(defined(SERIAL_RX) && defined(PROMINI))  //Only one serial port on
ProMini.  Skip serial com if SERIAL RX in use. Note: Spek code will auto-call
serialCom if GUI data detected on serial0.
    serialCom();
  #endif

  #if defined(POWERMETER)
    analog.intPowerMeterSum = (pMeter[PMOTOR_SUM]/PLEVELDIV);
    intPowerTrigger1 = conf.powerTrigger1 * PLEVELSCALE;
  #endif

  #ifdef LCD_TELEMETRY_AUTO
    static char telemetryAutoSequence []  = LCD_TELEMETRY_AUTO;
    static uint8_t telemetryAutoIndex = 0;
    static uint16_t telemetryAutoTimer = 0;
    if ( (telemetry_auto) && (! (++telemetryAutoTimer %
LCD_TELEMETRY_AUTO_FREQ) )  ){
      telemetry = telemetryAutoSequence[++telemetryAutoIndex %
strlen(telemetryAutoSequence)];
      LCDclear(); // make sure to clear away remnants
    }
  #endif
  #ifdef LCD_TELEMETRY
    static uint16_t telemetryTimer = 0;
    if (! (++telemetryTimer % LCD_TELEMETRY_FREQ)) {
      #if (LCD_TELEMETRY_DEBUG+0 > 0)
        telemetry = LCD_TELEMETRY_DEBUG;
      #endif
      if (telemetry) lcd_telemetry();
    }
  #endif

  #if GPS & defined(GPS_LED_INDICATOR)       // modified by MIS to use
STABLEPIN LED for number of sattelites indication
    static uint32_t GPSLEDTime;              // - No GPS FIX -> LED blink at speed of
incoming GPS frames
    static uint8_t blcnt;                    // - Fix and sat no. bellow 5 -> LED off
    if(currentTime > GPSLEDTime) {           // - Fix and sat no. >= 5 -> LED blinks,
one blink for 5 sat, two blinks for 6 sat, three for 7 ...
      if(f.GPS_FIX && GPS_numSat >= 5) {
        if(++blcnt > 2*GPS_numSat) blcnt = 0;
```

```
        GPSLEDTime = currentTime + 150000;
        if(blcnt >= 10 && ((blcnt%2) == 0)) {STABLEPIN_ON;} else
{STABLEPIN_OFF;}
      }else{
        if((GPS_update == 1) && !f.GPS_FIX) {STABLEPIN_ON;} else
{STABLEPIN_OFF;}
        blcnt = 0;
      }
    }
  #endif

  #if defined(LOG_VALUES) && (LOG_VALUES >= 2)
    if (cycleTime > cycleTimeMax) cycleTimeMax = cycleTime; // remember
highscore
    if (cycleTime < cycleTimeMin) cycleTimeMin = cycleTime; // remember lowscore
  #endif
  if (f.ARMED)  {
    #if defined(LCD_TELEMETRY) || defined(ARMEDTIMEWARNING) ||
defined(LOG_PERMANENT)
      armedTime += (uint32_t)cycleTime;
    #endif
    #if defined(VBAT)
      if ( (analog.vbat > NO_VBAT) && (analog.vbat < vbatMin) ) vbatMin =
analog.vbat;
    #endif
    #ifdef LCD_TELEMETRY
      #if BARO
        if ( (alt.EstAlt > BAROaltMax) ) BAROaltMax = alt.EstAlt;
      #endif
      #if GPS
        if ( (GPS_speed > GPS_speedMax) ) GPS_speedMax = GPS_speed;
      #endif
    #endif
  }
}

void setup() {
  SerialOpen(0,SERIAL0_COM_SPEED);
  #if defined(PROMICRO)
    SerialOpen(1,SERIAL1_COM_SPEED);
  #endif
  #if defined(MEGA)
    SerialOpen(1,SERIAL1_COM_SPEED);
    SerialOpen(2,SERIAL2_COM_SPEED);
    SerialOpen(3,SERIAL3_COM_SPEED);
  #endif
  LEDPIN_PINMODE;
  POWERPIN_PINMODE;
  BUZZERPIN_PINMODE;
  STABLEPIN_PINMODE;
```

```
  POWERPIN_OFF;
  initOutput();
  readGlobalSet();
  #ifndef NO_FLASH_CHECK
    #if defined(MEGA)
      uint16_t i = 65000;                    // only first ~64K for mega board due to
pgm_read_byte limitation
    #else
      uint16_t i = 32000;
    #endif
    uint16_t flashsum = 0;
    uint8_t pbyt;
    while(i--) {
      pbyt =  pgm_read_byte(i);        // calculate flash checksum
      flashsum += pbyt;
      flashsum ^= (pbyt<<8);
    }
  #endif
  #ifdef MULTIPLE_CONFIGURATION_PROFILES
    global_conf.currentSet=2;
  #else
    global_conf.currentSet=0;
  #endif
  while(1) {                                    // check settings integrity
  #ifndef NO_FLASH_CHECK
    if(readEEPROM()) {                          // check current setting integrity
      if(flashsum != global_conf.flashsum) update_constants();  // update constants if
firmware is changed and integrity is OK
    }
  #else
    readEEPROM();                               // check current setting integrity
  #endif
    if(global_conf.currentSet == 0) break;              // all checks is done
    global_conf.currentSet--;                   // next setting for check
  }
  readGlobalSet();                      // reload global settings for get last profile
number
  #ifndef NO_FLASH_CHECK
    if(flashsum != global_conf.flashsum) {
      global_conf.flashsum = flashsum;         // new flash sum
      writeGlobalSet(1);                    // update flash sum in global config
    }
  #endif
  readEEPROM();                             // load setting data from last used profile
  blinkLED(2,40,global_conf.currentSet+1);

  #if GPS
    recallGPSconf();                        //Load GPS configuration parameteres
  #endif
```

```
  configureReceiver();
  #if defined (PILOTLAMP)
    PL_INIT;
  #endif
  #if defined(OPENLRSv2MULTI)
    initOpenLRS();
  #endif
  initSensors();
  #if GPS
    GPS_set_pids();
  #endif
  previousTime = micros();
  #if defined(GIMBAL)
   calibratingA = 512;
  #endif
  calibratingG = 512;
  calibratingB = 200;  // 10 seconds init_delay + 200 * 25 ms = 15 seconds before
ground pressure settles
  #if defined(POWERMETER)
    for(uint8_t j=0; j<=PMOTOR_SUM; j++) pMeter[j]=0;
  #endif
  /************************************/
  #if GPS
    #if defined(GPS_SERIAL)
      GPS_SerialInit();
    #endif
    GPS_conf.max_wp_number = getMaxWPNumber();
  #endif

  #if defined(LCD_ETPP) || defined(LCD_LCD03) || defined(LCD_LCD03S) ||
defined(OLED_I2C_128x64) || defined(OLED_DIGOLE) ||
defined(LCD_TELEMETRY_STEP)
    initLCD();
  #endif
  #ifdef LCD_TELEMETRY_DEBUG
    telemetry_auto = 1;
  #endif
  #ifdef LCD_CONF_DEBUG
    configurationLoop();
  #endif
  #ifdef LANDING_LIGHTS_DDR
    init_landing_lights();
  #endif
  #ifdef FASTER_ANALOG_READS
    ADCSRA |= _BV(ADPS2) ; ADCSRA &= ~_BV(ADPS1); ADCSRA &=
~_BV(ADPS0); // this speeds up analogRead without loosing too much resolution:
http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1208715493/11
  #endif
  #if defined(LED_FLASHER)
    init_led_flasher();
```

```
    led_flasher_set_sequence(LED_FLASHER_SEQUENCE);
  #endif
  f.SMALL_ANGLES_25=1; // important for gyro only conf
  #ifdef LOG_PERMANENT
    // read last stored set
    readPLog();
    plog.lifetime += plog.armed_time / 1000000;
    plog.start++;         // #powercycle/reset/initialize events
    // dump plog data to terminal
    #ifdef LOG_PERMANENT_SHOW_AT_STARTUP
      dumpPLog(0);
    #endif
    plog.armed_time = 0;   // lifetime in seconds
    //plog.running = 0;       // toggle on arm & disarm to monitor for clean shutdown vs.
powercut
  #endif
  #ifdef DEBUGMSG
    debugmsg_append_str("initialization completed\n");
  #endif
}

void go_arm() {
  if(calibratingG == 0
  #if defined(ONLYARMWHENFLAT)
    && f.ACC_CALIBRATED
  #endif
  #if defined(FAILSAFE)
    && failsafeCnt < 2
  #endif
  #if GPS && defined(ONLY_ALLOW_ARM_WITH_GPS_3DFIX)
    && (f.GPS_FIX && GPS_numSat >= 5)
  #endif
    ) {
    if(!f.ARMED && !f.BARO_MODE) { // arm now!
      f.ARMED = 1;
      #if defined(HEADFREE)
        headFreeModeHold = att.heading;
      #endif
      magHold = att.heading;
      #if defined(VBAT)
        if (analog.vbat > NO_VBAT) vbatMin = analog.vbat;
      #endif
      #ifdef ALTITUDE_RESET_ON_ARM
        #if BARO
          calibratingB = 10; // calibrate baro to new ground level (10 * 25 ms = ~250 ms
non blocking)
        #endif
      #endif
      #ifdef LCD_TELEMETRY // reset some values when arming
        #if BARO
```

```
        BAROaltMax = alt.EstAlt;
       #endif
       #if GPS
        GPS_speedMax = 0;
       #endif
       #if defined( POWERMETER_HARD ) && (defined(LOG_VALUES) ||
defined(LCD_TELEMETRY))
        powerValueMaxMAH = 0;
       #endif
       #ifdef WATTS
        wattsMax = 0;
       #endif
      #endif
      #ifdef LOG_PERMANENT
       plog.arm++;          // #arm events
       plog.running = 1;       // toggle on arm & disarm to monitor for clean shutdown
vs. powercut
       // write now.
       writePLog();
      #endif
    }
  } else if(!f.ARMED) {
    blinkLED(2,255,1);
    SET_ALARM(ALRM_FAC_ACC, ALRM_LVL_ON);
  }
}
void go_disarm() {
  if (f.ARMED) {
    f.ARMED = 0;
    #ifdef LOG_PERMANENT
      plog.disarm++;        // #disarm events
      plog.armed_time = armedTime ;   // lifetime in seconds
      if (failsafeEvents) plog.failsafe++;      // #acitve failsafe @ disarm
      if (i2c_errors_count > 10) plog.i2c++;          // #i2c errs @ disarm
      plog.running = 0;       // toggle @ arm & disarm to monitor for clean shutdown vs.
powercut
      // write now.
      writePLog();
    #endif
  }
}

// ******** Main Loop *********
void loop () {
  static uint8_t rcDelayCommand; // this indicates the number of time (multiple of RC
measurement at 50Hz) the sticks must be maintained to run or switch off motors
  static uint8_t rcSticks;       // this hold sticks position for command combos
  uint8_t axis,i;
  int16_t error,errorAngle;
  int16_t delta;
```

```
 int16_t PTerm = 0,ITerm = 0,DTerm, PTermACC, ITermACC;
 static int16_t lastGyro[2] = {0,0};
 static int16_t errorAngleI[2] = {0,0};
 #if PID_CONTROLLER == 1
 static int32_t errorGyroI_YAW;
 static int16_t delta1[2],delta2[2];
 static int16_t errorGyroI[2] = {0,0};
 #elif PID_CONTROLLER == 2
 static int16_t delta1[3],delta2[3];
 static int32_t errorGyroI[3] = {0,0,0};
 static int16_t lastError[3] = {0,0,0};
 int16_t deltaSum;
 int16_t AngleRateTmp, RateError;
 #endif
 static uint16_t rcTime  = 0;
 static int16_t initialThrottleHold;
 int16_t rc;
 int32_t prop = 0;

 #if defined(SERIAL_RX)
   if (spekFrameFlags == 0x01) readSerial_RX();
 #endif
 #if defined(OPENLRSv2MULTI)
   Read_OpenLRS_RC();
 #endif

 #if defined(SERIAL_RX)
 if ((spekFrameDone == 0x01) || ((int16_t)(currentTime-rcTime) >0 )) {
   spekFrameDone = 0x00;
 #else
 if ((int16_t)(currentTime-rcTime) >0 ) { // 50Hz
 #endif
   rcTime = currentTime + 20000;
   computeRC();
   // Failsafe routine - added by MIS
   #if defined(FAILSAFE)
     if ( failsafeCnt > (5*FAILSAFE_DELAY) && f.ARMED) {                 //
Stabilize, and set Throttle to specified level
       for(i=0; i<3; i++) rcData[i] = MIDRC;                             // after specified guard
time after RC signal is lost (in 0.1sec)
       rcData[THROTTLE] = conf.failsafe_throttle;
       if (failsafeCnt > 5*(FAILSAFE_DELAY+FAILSAFE_OFF_DELAY)) {        //
Turn OFF motors after specified Time (in 0.1sec)
         go_disarm();     // This will prevent the copter to automatically rearm if failsafe
shuts it down and prevents
         f.OK_TO_ARM = 0; // to restart accidentely by just reconnect to the tx - you
will have to switch off first to rearm
       }
       failsafeEvents++;
     }
```

```
    if ( failsafeCnt > (5*FAILSAFE_DELAY) && !f.ARMED) {  //Turn of "Ok To arm to prevent the motors from spinning after repowering the RX with low throttle and aux to arm
        go_disarm();     // This will prevent the copter to automatically rearm if failsafe shuts it down and prevents
        f.OK_TO_ARM = 0; // to restart accidentely by just reconnect to the tx - you will have to switch off first to rearm
    }
    failsafeCnt++;
  #endif
  // end of failsafe routine - next change is made with RcOptions setting

  // ------------------ STICKS COMMAND HANDLER --------------------
  // checking sticks positions
  uint8_t stTmp = 0;
  for(i=0;i<4;i++) {
    stTmp >>= 2;
    if(rcData[i] > MINCHECK) stTmp |= 0x80;      // check for MIN
    if(rcData[i] < MAXCHECK) stTmp |= 0x40;      // check for MAX
  }
  if(stTmp == rcSticks) {
    if(rcDelayCommand<250) rcDelayCommand++;
  } else rcDelayCommand = 0;
  rcSticks = stTmp;

  // perform actions
  if (rcData[THROTTLE] <= MINCHECK) {            // THROTTLE at minimum
    #if !defined(FIXEDWING)
      errorGyroI[ROLL] = 0; errorGyroI[PITCH] = 0;
      #if PID_CONTROLLER == 1
        errorGyroI_YAW = 0;
      #elif PID_CONTROLLER == 2
        errorGyroI[YAW] = 0;
      #endif
      errorAngleI[ROLL] = 0; errorAngleI[PITCH] = 0;
    #endif
    if (conf.activate[BOXARM] > 0) {             // Arming/Disarming via ARM BOX
      if ( rcOptions[BOXARM] && f.OK_TO_ARM ) go_arm(); else if (f.ARMED) go_disarm();
    }
  }
  if(rcDelayCommand == 20) {
    if(f.ARMED) {                  // actions during armed
      #ifdef ALLOW_ARM_DISARM_VIA_TX_YAW
        if (conf.activate[BOXARM] == 0 && rcSticks == THR_LO + YAW_LO + PIT_CE + ROL_CE) go_disarm();    // Disarm via YAW
      #endif
      #ifdef ALLOW_ARM_DISARM_VIA_TX_ROLL
        if (conf.activate[BOXARM] == 0 && rcSticks == THR_LO + YAW_CE + PIT_CE + ROL_LO) go_disarm();    // Disarm via ROLL
```

```
      #endif
    } else {                    // actions during not armed
      i=0;
      if (rcSticks == THR_LO + YAW_LO + PIT_LO + ROL_CE) {    // GYRO
calibration
        calibratingG=512;
        #if GPS
          GPS_reset_home_position();
        #endif
        #if BARO
          calibratingB=10;  // calibrate baro to new ground level (10 * 25 ms = ~250 ms
non blocking)
        #endif
      }
      #if defined(INFLIGHT_ACC_CALIBRATION)
       else if (rcSticks == THR_LO + YAW_LO + PIT_HI + ROL_HI) {    // Inflight
ACC calibration START/STOP
          if (AccInflightCalibrationMeasurementDone){               // trigger saving into
eeprom after landing
           AccInflightCalibrationMeasurementDone = 0;
           AccInflightCalibrationSavetoEEProm = 1;
          }else{
           AccInflightCalibrationArmed = !AccInflightCalibrationArmed;
           #if defined(BUZZER)
            if (AccInflightCalibrationArmed)
SET_ALARM_BUZZER(ALRM_FAC_TOGGLE, ALRM_LVL_TOGGLE_2);
             else     SET_ALARM_BUZZER(ALRM_FAC_TOGGLE,
ALRM_LVL_TOGGLE_ELSE);
           #endif
          }
       }
      #endif
      #ifdef MULTIPLE_CONFIGURATION_PROFILES
        if      (rcSticks == THR_LO + YAW_LO + PIT_CE + ROL_LO) i=1;    // ROLL
left  -> Profile 1
        else if (rcSticks == THR_LO + YAW_LO + PIT_HI + ROL_CE) i=2;    //
PITCH up   -> Profile 2
        else if (rcSticks == THR_LO + YAW_LO + PIT_CE + ROL_HI) i=3;    //
ROLL right -> Profile 3
        if(i) {
          global_conf.currentSet = i-1;
          writeGlobalSet(0);
          readEEPROM();
          blinkLED(2,40,i);
          SET_ALARM(ALRM_FAC_TOGGLE, i);
        }
      #endif
      if (rcSticks == THR_LO + YAW_HI + PIT_HI + ROL_CE) {           // Enter
LCD config
        #if defined(LCD_CONF)
```

```
          configurationLoop(); // beginning LCD configuration
        #endif
        previousTime = micros();
      }
      #ifdef ALLOW_ARM_DISARM_VIA_TX_YAW
        else if (conf.activate[BOXARM] == 0 && rcSticks == THR_LO + YAW_HI +
PIT_CE + ROL_CE) go_arm();      // Arm via YAW
      #endif
      #ifdef ALLOW_ARM_DISARM_VIA_TX_ROLL
        else if (conf.activate[BOXARM] == 0 && rcSticks == THR_LO + YAW_CE +
PIT_CE + ROL_HI) go_arm();      // Arm via ROLL
      #endif
      #ifdef LCD_TELEMETRY_AUTO
        else if (rcSticks == THR_LO + YAW_CE + PIT_HI + ROL_LO) {             //
Auto telemetry ON/OFF
          if (telemetry_auto) {
            telemetry_auto = 0;
            telemetry = 0;
          } else
            telemetry_auto = 1;
        }
      #endif
      #ifdef LCD_TELEMETRY_STEP
        else if (rcSticks == THR_LO + YAW_CE + PIT_HI + ROL_HI) {             //
Telemetry next step
          telemetry = telemetryStepSequence[++telemetryStepIndex %
strlen(telemetryStepSequence)];
          #if defined( OLED_I2C_128x64)
            if (telemetry != 0) i2c_OLED_init();
          #elif defined(OLED_DIGOLE)
            if (telemetry != 0) i2c_OLED_DIGOLE_init();
          #endif
          LCDclear();
        }
      #endif
      #if ACC
        else if (rcSticks == THR_HI + YAW_LO + PIT_LO + ROL_CE)
calibratingA=512;     // throttle=max, yaw=left, pitch=min
      #endif
      #if MAG
        else if (rcSticks == THR_HI + YAW_HI + PIT_LO + ROL_CE)
f.CALIBRATE_MAG = 1;  // throttle=max, yaw=right, pitch=min
      #endif
      i=0;
      if      (rcSticks == THR_HI + YAW_CE + PIT_HI + ROL_CE)
{conf.angleTrim[PITCH]+=2; i=1;}
      else if (rcSticks == THR_HI + YAW_CE + PIT_LO + ROL_CE)
{conf.angleTrim[PITCH]-=2; i=1;}
      else if (rcSticks == THR_HI + YAW_CE + PIT_CE + ROL_HI)
{conf.angleTrim[ROLL] +=2; i=1;}
```

```
      else if (rcSticks == THR_HI + YAW_CE + PIT_CE + ROL_LO)
{conf.angleTrim[ROLL] -=2; i=1;}
      if (i) {
        writeParams(1);
        rcDelayCommand = 0;    // allow autorepetition
        #if defined(LED_RING)
          blinkLedRing();
        #endif
      }
    }
  }
  #if defined(LED_FLASHER)
    led_flasher_autoselect_sequence();
  #endif

  #if defined(INFLIGHT_ACC_CALIBRATION)
    if (AccInflightCalibrationArmed && f.ARMED && rcData[THROTTLE] >
MINCHECK && !rcOptions[BOXARM] ){ // Copter is airborne and you are turning
it off via boxarm : start measurement
      InflightcalibratingA = 50;
      AccInflightCalibrationArmed = 0;
    }
    if (rcOptions[BOXCALIB]) {      // Use the Calib Option to activate : Calib =
TRUE Meausrement started, Land and Calib = 0 measurement stored
      if (!AccInflightCalibrationActive &&
!AccInflightCalibrationMeasurementDone){
        InflightcalibratingA = 50;
      }
    }else if(AccInflightCalibrationMeasurementDone && !f.ARMED){
      AccInflightCalibrationMeasurementDone = 0;
      AccInflightCalibrationSavetoEEProm = 1;
    }
  #endif

  #if defined(EXTENDED_AUX_STATES)
  uint32_t auxState = 0;
  for(i=0;i<4;i++)
    auxState |=
    (uint32_t)(rcData[AUX1+i]<1230)<<(6*i) |
    (uint32_t)(1231<rcData[AUX1+i] && rcData[AUX1+i]<1360)<<(6*i+1) |
    (uint32_t)(1361<rcData[AUX1+i] && rcData[AUX1+i]<1490)<<(6*i+2) |
    (uint32_t)(1491<rcData[AUX1+i] && rcData[AUX1+i]<1620)<<(6*i+3) |
    (uint32_t)(1621<rcData[AUX1+i] && rcData[AUX1+i]<1749)<<(6*i+4) |
    (uint32_t)(rcData[AUX1+i]>1750)<<(6*i+5);
  #else
  uint16_t auxState = 0;
  for(i=0;i<4;i++)
    auxState |= (rcData[AUX1+i]<1300)<<(3*i) | (1300<rcData[AUX1+i] &&
rcData[AUX1+i]<1700)<<(3*i+1) | (rcData[AUX1+i]>1700)<<(3*i+2);
  #endif
```

```
    for(i=0;i<CHECKBOXITEMS;i++)
      rcOptions[i] = (auxState & conf.activate[i])>0;

    // note: if FAILSAFE is disable, failsafeCnt > 5*FAILSAFE_DELAY is always
false
    #if ACC
      if ( rcOptions[BOXANGLE] || (failsafeCnt > 5*FAILSAFE_DELAY) ) {
        // bumpless transfer to Level mode
        if (!f.ANGLE_MODE) {
          errorAngleI[ROLL] = 0; errorAngleI[PITCH] = 0;
          f.ANGLE_MODE = 1;
        }
      } else {
        if(f.ANGLE_MODE){
          errorGyroI[ROLL] = 0; errorGyroI[PITCH] = 0;
        }
        f.ANGLE_MODE = 0;
      }
      if ( rcOptions[BOXHORIZON] ) {
        f.ANGLE_MODE = 0;
        if (!f.HORIZON_MODE) {
          errorAngleI[ROLL] = 0; errorAngleI[PITCH] = 0;
          f.HORIZON_MODE = 1;
        }
      } else {
        if(f.HORIZON_MODE){
          errorGyroI[ROLL] = 0;errorGyroI[PITCH] = 0;
        }
        f.HORIZON_MODE = 0;
      }
    #endif

    if (rcOptions[BOXARM] == 0) f.OK_TO_ARM = 1;
    #if !defined(GPS_LED_INDICATOR)
      if (f.ANGLE_MODE || f.HORIZON_MODE) {STABLEPIN_ON;} else
{STABLEPIN_OFF;}
    #endif

    #if BARO
      #if (!defined(SUPPRESS_BARO_ALTHOLD))
        #if GPS
        if (GPS_conf.takeover_baro) rcOptions[BOXBARO] = (rcOptions[BOXBARO]
|| f.GPS_BARO_MODE);
        #endif
        if (rcOptions[BOXBARO]) {
          if (!f.BARO_MODE) {
            f.BARO_MODE = 1;
            AltHold = alt.EstAlt;
            #if defined(ALT_HOLD_THROTTLE_MIDPOINT)
```

```
            initialThrottleHold = ALT_HOLD_THROTTLE_MIDPOINT;
           #else
            initialThrottleHold = rcCommand[THROTTLE];
           #endif
           errorAltitudeI = 0;
           BaroPID=0;
          }
         } else {
          f.BARO_MODE = 0;
         }
        #endif
        #ifdef VARIOMETER
         if (rcOptions[BOXVARIO]) {
          if (!f.VARIO_MODE) {
           f.VARIO_MODE = 1;
          }
         } else {
          f.VARIO_MODE = 0;
         }
        #endif
       #endif
       if (rcOptions[BOXMAG]) {
        if (!f.MAG_MODE) {
         f.MAG_MODE = 1;
         magHold = att.heading;
        }
       } else {
        f.MAG_MODE = 0;
       }
       #if defined(HEADFREE)
        if (rcOptions[BOXHEADFREE]) {
         if (!f.HEADFREE_MODE) {
          f.HEADFREE_MODE = 1;
         }
         #if defined(ADVANCED_HEADFREE)
          if ((f.GPS_FIX && GPS_numSat >= 5) && (GPS_distanceToHome >
ADV_HEADFREE_RANGE) ) {
           if (GPS_directionToHome < 180)  {headFreeModeHold =
GPS_directionToHome + 180;} else {headFreeModeHold = GPS_directionToHome -
180;}
          }
         #endif
        } else {
         f.HEADFREE_MODE = 0;
        }
        if (rcOptions[BOXHEADADJ]) {
         headFreeModeHold = att.heading; // acquire new heading
        }
       #endif
```

```
   #if GPS
   // This handles the three rcOptions boxes
   // unlike other parts of the multiwii code, it looks for changes and not based on flag
settings
   // by this method a priority can be established between gps option

   //Generate a packed byte of all four GPS boxes.
   uint8_t gps_modes_check = (rcOptions[BOXLAND]<< 3) +
(rcOptions[BOXGPSHOME]<< 2) + (rcOptions[BOXGPSHOLD]<<1) +
(rcOptions[BOXGPSNAV]);

   if (f.ARMED ) {                    //Check GPS status and armed
     //TODO: implement f.GPS_Trusted flag, idea from Dramida - Check for degraded
HDOP and sudden speed jumps
     if (f.GPS_FIX) {
       if (GPS_numSat >5 ) {
         if (prv_gps_modes != gps_modes_check) {                    //Check for change
since last loop
           NAV_error = NAV_ERROR_NONE;
           if (rcOptions[BOXGPSHOME]) {                             // RTH has the
priotity over everything else
             init_RTH();
           } else if (rcOptions[BOXGPSHOLD]) {                      //Position hold has
priority over mission execution  //But has less priority than RTH
             if (f.GPS_mode == GPS_MODE_NAV)
               NAV_paused_at = mission_step.number;
             f.GPS_mode = GPS_MODE_HOLD;
             f.GPS_BARO_MODE = false;
             GPS_set_next_wp(&GPS_coord[LAT],
&GPS_coord[LON],&GPS_coord[LAT], & GPS_coord[LON]); //hold at the current
position
             set_new_altitude(alt.EstAlt);                          //and current altitude
             NAV_state = NAV_STATE_HOLD_INFINIT;
           } else if (rcOptions[BOXLAND]) {                         //Land now (It has
priority over Navigation)
             f.GPS_mode = GPS_MODE_HOLD;
             f.GPS_BARO_MODE = true;
             GPS_set_next_wp(&GPS_coord[LAT],
&GPS_coord[LON],&GPS_coord[LAT], & GPS_coord[LON]);
             set_new_altitude(alt.EstAlt);
             NAV_state = NAV_STATE_LAND_START;
           } else if (rcOptions[BOXGPSNAV]) {                       //Start navigation
             f.GPS_mode = GPS_MODE_NAV;                             //Nav mode start
             f.GPS_BARO_MODE = true;
             GPS_prev[LAT] = GPS_coord[LAT];
             GPS_prev[LON] = GPS_coord[LON];
             if (NAV_paused_at != 0) {
               next_step = NAV_paused_at;
               NAV_paused_at = 0;                                   //Clear paused step
             } else {
```

```
            next_step = 1;
            jump_times = -10;                          //Reset jump counter
           }
           NAV_state = NAV_STATE_PROCESS_NEXT;
          } else {                                     //None of the GPS Boxes are
switched on
           f.GPS_mode = GPS_MODE_NONE;
           f.GPS_BARO_MODE = false;
           f.THROTTLE_IGNORED = false;
           f.LAND_IN_PROGRESS = 0;
           f.THROTTLE_IGNORED = 0;
           NAV_state = NAV_STATE_NONE;
           GPS_reset_nav();
          }
          prv_gps_modes = gps_modes_check;
         }
        } else { //numSat>5
         //numSat dropped below 5 during navigation
         if (f.GPS_mode == GPS_MODE_NAV) {
          NAV_paused_at = mission_step.number;
          f.GPS_mode = GPS_MODE_NONE;
          set_new_altitude(alt.EstAlt);               //and current altitude
          NAV_state = NAV_STATE_NONE;
          NAV_error = NAV_ERROR_SPOILED_GPS;
          prv_gps_modes = 0xff;                       //invalidates mode check, to
allow re evaluate rcOptions when numsats raised again
         }
         if (f.GPS_mode == GPS_MODE_HOLD || f.GPS_mode ==
GPS_MODE_RTH) {
          f.GPS_mode = GPS_MODE_NONE;
          NAV_state = NAV_STATE_NONE;
          NAV_error = NAV_ERROR_SPOILED_GPS;
          prv_gps_modes = 0xff;                       //invalidates mode check, to
allow re evaluate rcOptions when numsats raised again
         }
         nav[0] = 0; nav[1] = 0;
        }
      } else { //f.GPS_FIX
       // GPS Fix dissapeared, very unlikely that we will be able to regain it, abort
mission
       f.GPS_mode = GPS_MODE_NONE;
       NAV_state = NAV_STATE_NONE;
       NAV_paused_at = 0;
       NAV_error = NAV_ERROR_GPS_FIX_LOST;
       GPS_reset_nav();
       prv_gps_modes = 0xff;                          //Gives a chance to restart
mission when regain fix
      }
    } else { //copter is armed
     //copter is disarmed
```

```
      f.GPS_mode = GPS_MODE_NONE;
      f.GPS_BARO_MODE = false;
      f.THROTTLE_IGNORED = false;
      NAV_state = NAV_STATE_NONE;
      NAV_paused_at = 0;
      NAV_error = NAV_ERROR_DISARMED;
      GPS_reset_nav();
    }

    #endif //GPS

    #if defined(FIXEDWING) || defined(HELICOPTER)
      if (rcOptions[BOXPASSTHRU]) {f.PASSTHRU_MODE = 1;}
      else {f.PASSTHRU_MODE = 0;}
    #endif

  } else { // not in rc loop
    static uint8_t taskOrder=0; // never call all functions in the same loop, to avoid high
delay spikes
    switch (taskOrder) {
      case 0:
        taskOrder++;
        #if MAG
          if (Mag_getADC() != 0) break; // 320 µs
        #endif
      case 1:
        taskOrder++;
        #if BARO
          if (Baro_update() != 0) break; // for MS baro: I2C set and get: 220 us  -  presure
and temperature computation 160 us
        #endif
      case 2:
        taskOrder++;
        #if BARO
          if (getEstimatedAltitude() != 0) break; // 280 us
        #endif
      case 3:
        taskOrder++;
        #if GPS
          if (GPS_Compute() != 0) break;  // performs computation on new frame only if
present
          #if defined(I2C_GPS)
          if (GPS_NewData() != 0) break;  // 160 us with no new data / much more with
new data
          #endif
        #endif
      case 4:
        taskOrder=0;
        #if SONAR
          Sonar_update(); //debug[2] = sonarAlt;
```

```
      #endif
      #ifdef LANDING_LIGHTS_DDR
        auto_switch_landing_lights();
      #endif
      #ifdef VARIOMETER
        if (f.VARIO_MODE) vario_signaling();
      #endif
      break;
    }
  }

  while(1) {
    currentTime = micros();
    cycleTime = currentTime - previousTime;
    #if defined(LOOP_TIME)
      if (cycleTime >= LOOP_TIME) break;
    #else
      break;
    #endif
  }
  previousTime = currentTime;

  computeIMU();

  //**************************************
  //**** Experimental FlightModes *****
  //**************************************
  #if defined(ACROTRAINER_MODE)
  if(f.ANGLE_MODE){
    if (abs(rcCommand[ROLL]) + abs(rcCommand[PITCH]) >=
ACROTRAINER_MODE ) {
      f.ANGLE_MODE=0;
      f.HORIZON_MODE=0;
      f.MAG_MODE=0;
      f.BARO_MODE=0;
      GPS_mode = GPS_MODE_NONE;
      }
    }
  #endif

  //**************************************
  // THROTTLE sticks during mission and RTH
  #if GPS
  if (GPS_conf.ignore_throttle == 1) {
    if (f.GPS_mode == GPS_MODE_NAV || f.GPS_mode == GPS_MODE_RTH) {
      //rcCommand[ROLL] = 0;
      //rcCommand[PITCH] = 0;
      //rcCommand[YAW] = 0;
      f.THROTTLE_IGNORED = 1;
    } else
```

```
    f.THROTTLE_IGNORED = 0;
  }

  //Heading manipulation TODO: Do heading manipulation
  #endif

  if (abs(rcCommand[YAW]) <70 && f.MAG_MODE) {
    int16_t dif = att.heading - magHold;
    if (dif <= - 180) dif += 360;
    if (dif >= + 180) dif -= 360;
    if (f.SMALL_ANGLES_25 || (f.GPS_mode != 0)) rcCommand[YAW] -=
dif*conf.pid[PIDMAG].P8 >> 5;  //Always correct maghold in GPS mode
  } else magHold = att.heading;

  #if BARO && (!defined(SUPPRESS_BARO_ALTHOLD))
  /* Smooth alt change routine , for slow auto and aerophoto modes (in general
solution from alexmos). It's slowly increase/decrease
  * altitude proportional to stick movement (+/-100 throttle gives about +/-50 cm in 1
second with cycle time about 3-4ms)
  */
  if (f.BARO_MODE) {
    static uint8_t isAltHoldChanged = 0;
    static int16_t AltHoldCorr = 0;

    #if GPS
    if (f.LAND_IN_PROGRESS) { //If autoland is in progress then take over and
decrease alt slowly
      AltHoldCorr -= GPS_conf.land_speed;
      if(abs(AltHoldCorr) > 512) {
        AltHold += AltHoldCorr/512;
        AltHoldCorr %= 512;
      }
    }
    #endif
    //IF Throttle not ignored then allow change altitude with the stick....
    if ( (abs(rcCommand[THROTTLE]-
initialThrottleHold)>ALT_HOLD_THROTTLE_NEUTRAL_ZONE) &&
!f.THROTTLE_IGNORED) {
      // Slowly increase/decrease AltHold proportional to stick movement ( +100
throttle gives ~ +50 cm in 1 second with cycle time about 3-4ms)
      AltHoldCorr+= rcCommand[THROTTLE] - initialThrottleHold;
      if(abs(AltHoldCorr) > 512) {
        AltHold += AltHoldCorr/512;
        AltHoldCorr %= 512;
      }
      isAltHoldChanged = 1;
    } else if (isAltHoldChanged) {
      AltHold = alt.EstAlt;
      isAltHoldChanged = 0;
    }
```

```
  rcCommand[THROTTLE] = initialThrottleHold + BaroPID;
 }
 #endif //BARO



 #if defined(THROTTLE_ANGLE_CORRECTION)
 if(f.ANGLE_MODE || f.HORIZON_MODE) {
  rcCommand[THROTTLE]+= throttleAngleCorrection;
 }
 #endif

 #if GPS
 //TODO: split cos_yaw calculations into two phases (X and Y)
 if (( f.GPS_mode != GPS_MODE_NONE ) && f.GPS_FIX_HOME ) {
  float sin_yaw_y = sin(att.heading*0.0174532925f);
  float cos_yaw_x = cos(att.heading*0.0174532925f);
  GPS_angle[ROLL]   = (nav[LON]*cos_yaw_x - nav[LAT]*sin_yaw_y) /10;
  GPS_angle[PITCH]  = (nav[LON]*sin_yaw_y + nav[LAT]*cos_yaw_x) /10;
  } else {
   GPS_angle[ROLL]  = 0;
   GPS_angle[PITCH] = 0;
  }

 //Used to communicate back nav angles to the GPS simulator (for HIL testing)
 #if defined(GPS_SIMULATOR)
  SerialWrite(2,0xa5);
  SerialWrite16(2,nav[LAT]+rcCommand[PITCH]);
  SerialWrite16(2,nav[LON]+rcCommand[ROLL]);
  SerialWrite16(2,(nav[LAT]+rcCommand[PITCH])-
(nav[LON]+rcCommand[ROLL])); //check
 #endif

 #endif //GPS

 //**** PITCH & ROLL & YAW PID ****
 #if PID_CONTROLLER == 1 // evolved oldschool
 if ( f.HORIZON_MODE ) prop =
min(max(abs(rcCommand[PITCH]),abs(rcCommand[ROLL])),512);

 // PITCH & ROLL
 for(axis=0;axis<2;axis++) {
  rc = rcCommand[axis]<<1;
  error = rc - imu.gyroData[axis];
  errorGyroI[axis]  = constrain(errorGyroI[axis]+error,-16000,+16000);       //
WindUp   16 bits is ok here
  if (abs(imu.gyroData[axis])>640) errorGyroI[axis] = 0;

  ITerm = (errorGyroI[axis]>>7)*conf.pid[axis].I8>>6;                    // 16 bits is ok
here 16000/125 = 128 ; 128*250 = 32000
```

```
    PTerm = mul(rc,conf.pid[axis].P8)>>6;

    if (f.ANGLE_MODE || f.HORIZON_MODE) { // axis relying on ACC
      // 50 degrees max inclination
      errorAngle         = constrain(rc + GPS_angle[axis],-500,+500) - att.angle[axis] +
conf.angleTrim[axis]; //16 bits is ok here
      errorAngleI[axis]  = constrain(errorAngleI[axis]+errorAngle,-10000,+10000);
// WindUp     //16 bits is ok here

      PTermACC           = mul(errorAngle,conf.pid[PIDLEVEL].P8)>>7; // 32 bits is
needed for calculation: errorAngle*P8 could exceed 32768   16 bits is ok for result

      int16_t limit      = conf.pid[PIDLEVEL].D8*5;
      PTermACC           = constrain(PTermACC,-limit,+limit);

      ITermACC           = mul(errorAngleI[axis],conf.pid[PIDLEVEL].I8)>>12;   // 32
bits is needed for calculation:10000*I8 could exceed 32768   16 bits is ok for result

      ITerm              = ITermACC + ((ITerm-ITermACC)*prop>>9);
      PTerm              = PTermACC + ((PTerm-PTermACC)*prop>>9);
    }

    PTerm -= mul(imu.gyroData[axis],dynP8[axis])>>6; // 32 bits is needed for
calculation

    delta          = imu.gyroData[axis] - lastGyro[axis];  // 16 bits is ok here, the dif
between 2 consecutive gyro reads is limited to 800
    lastGyro[axis] = imu.gyroData[axis];
    DTerm          = delta1[axis]+delta2[axis]+delta;
    delta2[axis]   = delta1[axis];
    delta1[axis]   = delta;

    DTerm = mul(DTerm,dynD8[axis])>>5;        // 32 bits is needed for calculation

    axisPID[axis] =  PTerm + ITerm - DTerm;
  }

  //YAW
  #define GYRO_P_MAX 300
  #define GYRO_I_MAX 250

  rc = mul(rcCommand[YAW] , (2*conf.yawRate + 30))  >> 5;

  error = rc - imu.gyroData[YAW];
  errorGyroI_YAW  += mul(error,conf.pid[YAW].I8);
  errorGyroI_YAW  = constrain(errorGyroI_YAW, 2-((int32_t)1<<28), -
2+((int32_t)1<<28));
  if (abs(rc) > 50) errorGyroI_YAW = 0;
```

```
  PTerm = mul(error,conf.pid[YAW].P8)>>6;
  #ifndef COPTER_WITH_SERVO
    int16_t limit = GYRO_P_MAX-conf.pid[YAW].D8;
    PTerm = constrain(PTerm,-limit,+limit);
  #endif

  ITerm = constrain((int16_t)(errorGyroI_YAW>>13),-
GYRO_I_MAX,+GYRO_I_MAX);

  axisPID[YAW] =  PTerm + ITerm;

  #elif PID_CONTROLLER == 2 // alexK
  #define GYRO_I_MAX 256
  #define ACC_I_MAX 256
  prop = min(max(abs(rcCommand[PITCH]),abs(rcCommand[ROLL])),500); // range
[0;500]

  //----------PID controller----------
  for(axis=0;axis<3;axis++) {
    //-----Get the desired angle rate depending on flight mode
    if ((f.ANGLE_MODE || f.HORIZON_MODE) && axis<2 ) { // MODE relying on
ACC
      // calculate error and limit the angle to 50 degrees max inclination
      errorAngle = constrain((rcCommand[axis]<<1) + GPS_angle[axis],-500,+500) -
att.angle[axis] + conf.angleTrim[axis]; //16 bits is ok here
    }
    if (axis == 2) {//YAW is always gyro-controlled (MAG correction is applied to
rcCommand)
      AngleRateTmp = (((int32_t) (conf.yawRate + 27) * rcCommand[2]) >> 5);
    } else {
      if (!f.ANGLE_MODE) {//control is GYRO based (ACRO and HORIZON - direct
sticks control is applied to rate PID
        AngleRateTmp = ((int32_t) (conf.rollPitchRate + 27) * rcCommand[axis]) >> 4;
        if (f.HORIZON_MODE) {
          //mix up angle error to desired AngleRateTmp to add a little auto-level feel
          AngleRateTmp += ((int32_t) errorAngle * conf.pid[PIDLEVEL].I8)>>8;
        }
      } else {//it's the ANGLE mode - control is angle based, so control loop is needed
        AngleRateTmp = ((int32_t) errorAngle * conf.pid[PIDLEVEL].P8)>>4;
      }
    }

    //--------low-level gyro-based PID. ----------
    //Used in stand-alone mode for ACRO, controlled by higher level regulators in
other modes
    //-----calculate scaled error.AngleRates
    //multiplication of rcCommand corresponds to changing the sticks scaling here
    RateError = AngleRateTmp  - imu.gyroData[axis];

    //-----calculate P component
```

```
    PTerm = ((int32_t) RateError * conf.pid[axis].P8)>>7;

    //-----calculate I component
    //there should be no division before accumulating the error to integrator, because the precision would be reduced.
    //Precision is critical, as I prevents from long-time drift. Thus, 32 bits integrator is used.
    //Time correction (to avoid different I scaling for different builds based on average cycle time)
    //is normalized to cycle time = 2048.
    errorGyroI[axis]  += (((int32_t) RateError * cycleTime)>>11) * conf.pid[axis].I8;
    //limit maximum integrator value to prevent WindUp - accumulating extreme values when system is saturated.
    //I coefficient (I8) moved before integration to make limiting independent from PID settings
    errorGyroI[axis]  = constrain(errorGyroI[axis], (int32_t) -GYRO_I_MAX<<13, (int32_t) +GYRO_I_MAX<<13);
    ITerm = errorGyroI[axis]>>13;

    //-----calculate D-term
    delta          = RateError - lastError[axis];  // 16 bits is ok here, the dif between 2 consecutive gyro reads is limited to 800
    lastError[axis] = RateError;

    //Correct difference by cycle time. Cycle time is jittery (can be different 2 times), so calculated difference
    // would be scaled by different dt each time. Division by dT fixes that.
    delta = ((int32_t) delta * ((uint16_t)0xFFFF / (cycleTime>>4)))>>6;
    //add moving average here to reduce noise
    deltaSum       = delta1[axis]+delta2[axis]+delta;
    delta2[axis]   = delta1[axis];
    delta1[axis]   = delta;

    //DTerm = (deltaSum*conf.pid[axis].D8)>>8;
    //Solve overflow in calculation above...
    DTerm = ((int32_t)deltaSum*conf.pid[axis].D8)>>8;
    //-----calculate total PID output
    axisPID[axis] =  PTerm + ITerm + DTerm;
  }
  #else
    #error "*** you must set PID_CONTROLLER to one existing implementation"
  #endif
  mixTable();
  // do not update servos during unarmed calibration of sensors which are sensitive to vibration
  #if defined(DISABLE_SERVOS_WHEN_UNARMED)
  if (f.ARMED) writeServos();
  #else
  if ( (f.ARMED) || ((!calibratingG) && (!calibratingA)) ) writeServos();
  #endif
```

```
  writeMotors();
}
```
καρτέλα MultiWii.h
```
#define MINCHECK 1100
#define MAXCHECK 1900

extern volatile unsigned long timer0_overflow_count;

extern const char pidnames[];
extern const char boxnames[];
extern const uint8_t boxids[];

extern uint32_t currentTime;
extern uint16_t previousTime;
extern uint16_t cycleTime;
extern uint16_t calibratingA;
extern uint16_t calibratingB;
extern uint16_t calibratingG;
extern int16_t  magHold,headFreeModeHold;
extern uint8_t  vbatMin;
extern uint8_t  rcOptions[CHECKBOXITEMS];
extern int32_t  AltHold;
extern int16_t  sonarAlt;
extern int16_t  BaroPID;
extern int16_t  errorAltitudeI;

extern int16_t  i2c_errors_count;
extern uint8_t alarmArray[ALRM_FAC_SIZE];
extern global_conf_t global_conf;

extern imu_t imu;
extern analog_t analog;
extern alt_t alt;
extern att_t att;
#ifdef LOG_PERMANENT
extern plog_t p
#endif

extern int16_t debug[4];

extern conf_t conf;
extern int16_t  annex650_overrun_count;
extern flags_struct_t f;
extern uint16_t intPowerTrigger1;

extern int16_t gyroZero[3];
extern int16_t angle[2];


#if BARO
```

```
  extern int32_t baroPressure;
  extern int16_t baroTemperature; // temp in 0.01 deg
  extern int32_t baroPressureSum;
#endif

extern int16_t axisPID[3];
extern int16_t motor[8];
extern int16_t servo[8];

extern int16_t failsafeEvents;
extern volatile int16_t failsafeCnt;

extern int16_t rcData[RC_CHANS];
extern int16_t rcSerial[8];
extern int16_t rcCommand[4];
extern uint8_t rcSerialCount;
extern int16_t lookupPitchRollRC[5];
extern uint16_t lookupThrottleRC[11];

#if defined(POWERMETER) || ( defined(LOG_VALUES) && (LOG_VALUES >=
3) )
  #define PMOTOR_SUM 8                    // index into pMeter[] for sum
  extern uint32_t pMeter[PMOTOR_SUM + 1];  // we use [0:7] for eight motors,one
extra for sum
  extern uint8_t pMeterV;                 // dummy to satisfy the paramStruct logic in
ConfigurationLoop()
  extern uint32_t pAlarm;                 // we scale the eeprom value from [0:255] to this
value we can directly compare to the sum in pMeter[6]
  extern uint16_t powerValue;              // last known current
#endif

#if defined(LCD_TELEMETRY)
  extern uint8_t telemetry;
  extern uint8_t telemetry_auto;
#endif
#ifdef LCD_TELEMETRY_STEP
  extern char telemetryStepSequence[];
  extern uint8_t telemetryStepIndex;
#endif

#if defined(LOG_VALUES) || defined(LCD_TELEMETRY)
  extern uint16_t cycleTimeMax;       // highest ever cycle timen
  extern uint16_t cycleTimeMin;       // lowest ever cycle timen
  extern int32_t  BAROaltMax;         // maximum value
  extern uint16_t GPS_speedMax;       // maximum speed from gps
  extern uint16_t powerValueMaxMAH;
  extern uint16_t wattsMax;
#endif
#if defined(LOG_VALUES) || defined(LCD_TELEMETRY) ||
defined(ARMEDTIMEWARNING) || defined(LOG_PERMANENT)
```

```
  extern uint32_t armedTime;
#endif

#if GPS
// ************************************ begin GPS common variables and
defines
*****************************************************************

extern gps_conf_struct GPS_conf;

extern int16_t  GPS_angle[2];          // the angles that must be applied for GPS
correction
extern int32_t  GPS_coord[2];
extern int32_t  GPS_home[2];
extern int32_t  GPS_hold[2];
extern int32_t  GPS_prev[2];
extern int32_t  GPS_poi[2];            // Coordinates of the current poi
extern int32_t  GPS_directionToPoi;     // direction to the actual poi (used to set
heading to poi)
extern uint8_t  GPS_numSat;
extern uint16_t GPS_distanceToHome;     // distance to home  - unit: meter
extern int16_t  GPS_directionToHome;    // direction to home - unit: degree
extern uint16_t GPS_altitude;          // GPS altitude      - unit: meter
extern uint16_t GPS_speed;             // GPS speed         - unit: cm/s
extern uint8_t  GPS_update;            // a binary toogle to distinct a GPS position
update
extern uint16_t GPS_ground_course;      //                   - unit: degree*10
extern uint32_t GPS_time;

extern uint8_t  GPS_mode;              // contains the current selected gps flight mode

extern uint8_t NAV_error;               //Last error situation of the nav engine
extern uint8_t NAV_state;               //State of the nav engine
extern uint8_t GPS_saved_mission_state;   //The mission state saved when poshold
invoked during mission
extern uint8_t prv_gps_modes;           //GPS_checkbox items packed into 1 byte for
checking GPS mode changes
extern uint32_t nav_timer_stop;         //common timer used in navigation (contains
the desired stop time in millis()
extern uint16_t nav_hold_time;          //time in seconds to hold position
extern uint8_t NAV_paused_at;           //This contains the mission step where
poshold paused the runing mission.
extern uint8_t next_step;               //The mission step which is upcoming it equals
with the mission_step stored in EEPROM

//Altitude control state
#define ASCENDING          1
#define DESCENDING         -1
#define REACHED_ALT        0
```

```
// The orginal altitude used as base our new altitude during nav
extern int32_t original_altitude;
//This is the target what we want to reach
extern int32_t target_altitude;
//This is the interim value which is feeded into the althold controller
extern int32_t alt_to_hold;

extern uint32_t alt_change_timer;
extern int8_t   alt_change_flag;
extern uint32_t alt_change;
extern int16_t  jump_times;             //How many loops do we have to do (alt/100 from
mission step) -10 means not used jet, -1 unlimited
extern uint8_t  land_detect;            //land detector variable


// ************************
// mission step structure
// ************************
extern mission_step_struct mission_step;

//possible action codes for a mission step
#define MISSION_WAYPOINT      1   //Set waypoint
#define MISSION_HOLD_UNLIM    2   //Poshold unlimited
#define MISSION_HOLD_TIME     3   //Hold for a predetermined time
#define MISSION_RTH           4   //Return to HOME
#define MISSION_SET_POI       5   //Set POINT of interest
#define MISSION_JUMP          6   //Jump to the given step (#times)
#define MISSION_SET_HEADING   7   //Set heading to a given orientation
(parameter 1 is the waym 0-359 degree
#define MISSION_LAND          8   //Land at the given position


#define MISSION_FLAG_END        0xA5   //Flags that this is the last step
#define MISSION_FLAG_CRC_ERROR   0xFE   //Returned WP had an EEPROM
CRC error
#define MISSION_FLAG_HOME       0x01   //Returned WP is the home position
#define MISSION_FLAG_HOLD       0x02   //Returned WP is the hold position
#define MISSION_FLAG_DO_LAND    0x20   //Land when reached desired point
(used in RTH)
#define MISSION_FLAG_NAV_IN_PROG 0xff   //Navigation is in progress,
returned wp is home

#define LAT  0
#define LON  1

extern int16_t  nav[2];


#endif
```

```
// default POSHOLD control gains
#define POSHOLD_P              .15
#define POSHOLD_I              0.0
#define POSHOLD_IMAX           20       // degrees

#define POSHOLD_RATE_P         3.4
#define POSHOLD_RATE_I         0.14     // Wind control
#define POSHOLD_RATE_D         0.053    // try 2 or 3 for POSHOLD_RATE 1
#define POSHOLD_RATE_IMAX      20       // degrees

// default Navigation PID gains
#define NAV_P                  2.5
#define NAV_I                  0.33     // Wind control
#define NAV_D                  0.083    //
#define NAV_IMAX               20       // degrees


// ************************************* end GPS common variables and defines
//*****************************************************************

extern volatile uint8_t  spekFrameFlags;
extern volatile uint32_t spekTimeLast;
extern uint8_t  spekFrameDone;

#if defined(OPENLRSv2MULTI)
extern uint8_t pot_P,pot_I; // OpenLRS onboard potentiometers for P and I trim or other usages
#endif

// **********************
//Automatic ACC Offset Calibration
// **********************
#if defined(INFLIGHT_ACC_CALIBRATION)
extern uint16_t InflightcalibratingA;
extern int16_t AccInflightCalibrationArmed;
extern uint16_t AccInflightCalibrationMeasurementDone;
extern uint16_t AccInflightCalibrationSavetoEEProm;
extern uint16_t AccInflightCalibrationActive;
#endif

#if defined(ARMEDTIMEWARNING)
  extern uint32_t  ArmedTimeWarningMicroSeconds;
#endif

#if defined(THROTTLE_ANGLE_CORRECTION)
  extern int16_t throttleAngleCorrection;
  extern int8_t  cosZ;
#endif
```

```
void annexCode();
void go_disarm();
#endif /* MULTIWII_H_ */
```
καρτέλα Output.cpp
```
#include "Arduino.h"
#include "config.h"
#include "def.h"
#include "types.h"
#include "MultiWii.h"
#include "Alarms.h"

void initializeSoftPWM(void);

#if defined(SERVO)
void initializeServo();
#endif

/**************************************************************************
******************/
/***************          Motor Pin order          ********************/
/**************************************************************************
******************/
// since we are uing the PWM generation in a direct way, the pin order is just to
inizialie the right pins
// its not possible to change a PWM output pin just by changing the order
#if defined(PROMINI)
  uint8_t PWM_PIN[8] = {9,10,11,3,6,5,A2,12};   //for a quad+: rear,right,left,front
#endif
#if defined(PROMICRO)
  #if !defined(HWPWM6)
    #if defined(TEENSY20)
      uint8_t PWM_PIN[8] = {14,15,9,12,22,18,16,17};   //for a quad+:
rear,right,left,front
    #elif defined(A32U4_4_HW_PWM_SERVOS)
      uint8_t PWM_PIN[8] = {6,9,10,11,5,13,SW_PWM_P3,SW_PWM_P4};   //
    #else
    uint8_t PWM_PIN[8] = {9,10,5,6,4,A2,SW_PWM_P3,SW_PWM_P4};   //for a
quad+: rear,right,left,front
    #endif
  #else
    #if defined(TEENSY20)
      uint8_t PWM_PIN[8] = {14,15,9,12,4,10,16,17};   //for a quad+:
rear,right,left,front
    #elif defined(A32U4_4_HW_PWM_SERVOS)
      uint8_t PWM_PIN[8] = {6,9,10,11,5,13,SW_PWM_P3,SW_PWM_P4};   //
    #else
      uint8_t PWM_PIN[8] = {9,10,5,6,11,13,SW_PWM_P3,SW_PWM_P4};   //for a
quad+: rear,right,left,front
    #endif
  #endif
```

```
#endif
#if defined(MEGA)
  uint8_t PWM_PIN[8] = {3,5,6,2,7,8,9,10};      //for a quad+: rear,right,left,front   //+
for y6: 7:under right  8:under left
#endif

/************************************************************************
*****************/
/***************         Software PWM & Servo variables
*******************/
/************************************************************************
*****************/
#if defined(PROMINI) || (defined(PROMICRO) && defined(HWPWM6)) ||
(defined(MEGA) && defined(MEGA_HW_PWM_SERVOS))
  #if (NUMBER_MOTOR > 4)
    //for HEX Y6 and HEX6/HEX6X/HEX6H flat for promini
    volatile uint8_t atomicPWM_PIN5_lowState;
    volatile uint8_t atomicPWM_PIN5_highState;
    volatile uint8_t atomicPWM_PIN6_lowState;
    volatile uint8_t atomicPWM_PIN6_highState;
  #endif
  #if (NUMBER_MOTOR > 6)
    //for OCTO on promini
    volatile uint8_t atomicPWM_PINA2_lowState;
    volatile uint8_t atomicPWM_PINA2_highState;
    volatile uint8_t atomicPWM_PIN12_lowState;
    volatile uint8_t atomicPWM_PIN12_highState;
  #endif
#else
  #if (NUMBER_MOTOR > 4)
    //for HEX Y6 and HEX6/HEX6X/HEX6H and for Promicro
    volatile uint16_t atomicPWM_PIN5_lowState;
    volatile uint16_t atomicPWM_PIN5_highState;
    volatile uint16_t atomicPWM_PIN6_lowState;
    volatile uint16_t atomicPWM_PIN6_highState;
  #endif
  #if (NUMBER_MOTOR > 6)
    //for OCTO on Promicro
    volatile uint16_t atomicPWM_PINA2_lowState;
    volatile uint16_t atomicPWM_PINA2_highState;
    volatile uint16_t atomicPWM_PIN12_lowState;
    volatile uint16_t atomicPWM_PIN12_highState;
  #endif
#endif

#if defined(SERVO)
  #if defined(HW_PWM_SERVOS)
    // hw servo pwm does not need atomicServo[]
  #elif defined(PROMINI) || (defined(PROMICRO) && defined(HWPWM6))
    #if defined(AIRPLANE) || defined(HELICOPTER)
```

```
    // To prevent motor to start at reset. atomicServo[7]=5 or 249 if reversed servo
    volatile uint8_t atomicServo[8] = {125,125,125,125,125,125,125,5};
   #else
    volatile uint8_t atomicServo[8] = {125,125,125,125,125,125,125,125};
   #endif
  #else
   #if defined(AIRPLANE)|| defined(HELICOPTER)
    // To prevent motor to start at reset. atomicServo[7]=5 or 249 if reversed servo
    volatile uint16_t atomicServo[8] = {8000,8000,8000,8000,8000,8000,8000,320};
   #else
    volatile uint16_t atomicServo[8] = {8000,8000,8000,8000,8000,8000,8000,8000};
   #endif
  #endif
#endif

/**************************************************************************
******************/
/***************       Calculate first and last used servos
*********************/
/**************************************************************************
******************/
#if defined(SERVO)
 #if defined(PRI_SERVO_FROM) && defined(SEC_SERVO_FROM)
  #if PRI_SERVO_FROM < SEC_SERVO_FROM
   #define SERVO_START PRI_SERVO_FROM
  #else
   #define SERVO_START SEC_SERVO_FROM
  #endif
 #else
  #if defined(PRI_SERVO_FROM)
   #define SERVO_START PRI_SERVO_FROM
  #endif
  #if defined(SEC_SERVO_FROM)
   #define SERVO_START SEC_SERVO_FROM
  #endif
 #endif
 #if defined(PRI_SERVO_TO) && defined(SEC_SERVO_TO)
  #if PRI_SERVO_TO > SEC_SERVO_TO
   #define SERVO_END PRI_SERVO_TO
  #else
   #define SERVO_END SEC_SERVO_TO
  #endif
 #else
  #if defined(PRI_SERVO_TO)
   #define SERVO_END PRI_SERVO_TO
  #endif
  #if defined(SEC_SERVO_TO)
   #define SERVO_END SEC_SERVO_TO
  #endif
 #endif
```

```
#endif

/*********************************************************************
******************/
/***************  Writes the Servos values to the needed format
********************/
/*********************************************************************
******************/
void writeServos() {
  #if defined(SERVO)
    #if defined(PRI_SERVO_FROM) && !defined(HW_PWM_SERVOS)   // write
primary servos
      for(uint8_t i = (PRI_SERVO_FROM-1); i < PRI_SERVO_TO; i++){
        #if defined(PROMINI) || (defined(PROMICRO) && defined(HWPWM6)) ||
(defined(MEGA) && defined(MEGA_HW_PWM_SERVOS))
          atomicServo[i] = (servo[i]-1000)>>2;
        #else
          atomicServo[i] = (servo[i]-1000)<<4;
        #endif
      }
    #endif
    #if defined(SEC_SERVO_FROM) && !defined(HW_PWM_SERVOS)  // write
secundary servos
      #if (defined(SERVO_TILT)|| defined(SERVO_MIX_TILT)) &&
defined(MMSERVOGIMBAL)
        // Moving Average Servo Gimbal by Magnetron1
        static int16_t
mediaMobileServoGimbalADC[3][MMSERVOGIMBALVECTORLENGHT];
        static int32_t mediaMobileServoGimbalADCSum[3];
        static uint8_t mediaMobileServoGimbalIDX;
        uint8_t axis;

        mediaMobileServoGimbalIDX = ++mediaMobileServoGimbalIDX %
MMSERVOGIMBALVECTORLENGHT;
        for (axis=(SEC_SERVO_FROM-1); axis < SEC_SERVO_TO; axis++) {
          mediaMobileServoGimbalADCSum[axis] -=
mediaMobileServoGimbalADC[axis][mediaMobileServoGimbalIDX];
          mediaMobileServoGimbalADC[axis][mediaMobileServoGimbalIDX] =
servo[axis];
          mediaMobileServoGimbalADCSum[axis] +=
mediaMobileServoGimbalADC[axis][mediaMobileServoGimbalIDX];
          #if defined(PROMINI) || (defined(PROMICRO) && defined(HWPWM6))
            atomicServo[axis] = (mediaMobileServoGimbalADCSum[axis] /
MMSERVOGIMBALVECTORLENGHT - 1000)>>2;
          #else
            atomicServo[axis] = (mediaMobileServoGimbalADCSum[axis] /
MMSERVOGIMBALVECTORLENGHT - 1000)<<4;
          #endif
```

```
      }
    #else
      for(uint8_t i = (SEC_SERVO_FROM-1); i < SEC_SERVO_TO; i++){
       #if defined(PROMINI) || (defined(PROMICRO) && defined(HWPWM6)) ||
(defined(MEGA) && defined(MEGA_HW_PWM_SERVOS))
         atomicServo[i] = (servo[i]-1000)>>2;
       #else
         atomicServo[i] = (servo[i]-1000)<<4;
       #endif
      }
    #endif
   #endif
   // write HW PWM servos for the mega
   #if defined(MEGA) && defined(MEGA_HW_PWM_SERVOS)
    #if (PRI_SERVO_FROM == 1 || SEC_SERVO_FROM == 1)
     OCR5C = servo[0];
    #endif
    #if (PRI_SERVO_FROM <= 2 && PRI_SERVO_TO >= 2) ||
(SEC_SERVO_FROM <= 2 && SEC_SERVO_TO >= 2)
     OCR5B = servo[1];
    #endif
    #if (PRI_SERVO_FROM <= 3 && PRI_SERVO_TO >= 3) ||
(SEC_SERVO_FROM <= 3 && SEC_SERVO_TO >= 3)
     OCR5A = servo[2];
    #endif
    #if (PRI_SERVO_FROM <= 4 && PRI_SERVO_TO >= 4) ||
(SEC_SERVO_FROM <= 4 && SEC_SERVO_TO >= 4)
     OCR1A = servo[3];
    #endif
    #if (PRI_SERVO_FROM <= 5 && PRI_SERVO_TO >= 5) ||
(SEC_SERVO_FROM <= 5 && SEC_SERVO_TO >= 5)
     OCR1B = servo[4];
    #endif
    #if (PRI_SERVO_FROM <= 6 && PRI_SERVO_TO >= 6) ||
(SEC_SERVO_FROM <= 6 && SEC_SERVO_TO >= 6)
     OCR4A = servo[5];
    #endif
    #if (PRI_SERVO_FROM <= 7 && PRI_SERVO_TO >= 7) ||
(SEC_SERVO_FROM <= 7 && SEC_SERVO_TO >= 7)
     OCR4B = servo[6];
    #endif
    #if (PRI_SERVO_FROM <= 8 && PRI_SERVO_TO >= 8) ||
(SEC_SERVO_FROM <= 8 && SEC_SERVO_TO >= 8)
     OCR4C = servo[7];
    #endif
   #endif
   // write HW PWM servos for the promicro
   #if defined(PROMICRO) && defined(A32U4_4_HW_PWM_SERVOS)
    #if (PRI_SERVO_FROM <= 7 && PRI_SERVO_TO >= 7)
     OCR1A = servo[6];// Pin 9
```

```
    #endif
    #if (PRI_SERVO_FROM <= 5 && PRI_SERVO_TO >= 5)
      OCR1B = servo[4];// Pin 10
    #endif
    #if (PRI_SERVO_FROM <= 6 && PRI_SERVO_TO >= 6)
      OCR3A = servo[5];// Pin 5
    #endif
    #if (PRI_SERVO_FROM <= 4 && PRI_SERVO_TO >= 4)
      OCR1C = servo[3];// Pin 11
    #endif
   #endif
  #endif
}

/************************************************************************
******************/
/************  Writes the Motors values to the PWM compare register
******************/
/************************************************************************
******************/
void writeMotors() { // [1000;2000] => [125;250]
  /*****************  Specific PWM Timers & Registers for the MEGA's
*******************/
  #if defined(MEGA)// [1000:2000] => [8000:16000] for timer 3 & 4 for mega
    #if (NUMBER_MOTOR > 0)
      #ifndef EXT_MOTOR_RANGE
        OCR3C = motor[0]<<3; //  pin 3
      #else
        OCR3C = ((motor[0]<<4) - 16000);
      #endif
    #endif
    #if (NUMBER_MOTOR > 1)
      #ifndef EXT_MOTOR_RANGE
        OCR3A = motor[1]<<3; //  pin 5
      #else
        OCR3A = ((motor[1]<<4) - 16000);
      #endif
    #endif
    #if (NUMBER_MOTOR > 2)
      #ifndef EXT_MOTOR_RANGE
        OCR4A = motor[2]<<3; //  pin 6
      #else
        OCR4A = ((motor[2]<<4) - 16000);
      #endif
    #endif
    #if (NUMBER_MOTOR > 3)
      #ifndef EXT_MOTOR_RANGE
        OCR3B = motor[3]<<3; //  pin 2
      #else
        OCR3B = ((motor[3]<<4) - 16000);
```

```
    #endif
   #endif
   #if (NUMBER_MOTOR > 4)
    #ifndef EXT_MOTOR_RANGE
     OCR4B = motor[4]<<3; //  pin 7
     OCR4C = motor[5]<<3; //  pin 8
    #else
     OCR4B = ((motor[4]<<4) - 16000);
     OCR4C = ((motor[5]<<4) - 16000);
    #endif
   #endif
   #if (NUMBER_MOTOR > 6)
    #ifndef EXT_MOTOR_RANGE
     OCR2B = motor[6]>>3; //  pin 9
     OCR2A = motor[7]>>3; //  pin 10
    #else
     OCR2B = (motor[6]>>2) - 250;
     OCR2A = (motor[7]>>2) - 250;
    #endif
   #endif
  #endif

  /******** Specific PWM Timers & Registers for the atmega32u4 (Promicro)
**********/
  #if defined(PROMICRO)
   uint16_t Temp2;
   Temp2 = motor[3] - 1000;
   #if (NUMBER_MOTOR > 0)
    #if defined(A32U4_4_HW_PWM_SERVOS)
     // write motor0 to pin 6
     // Timer 4 A & D [1000:2000] => [1000:2000]
     #ifndef EXT_MOTOR_RANGE
      TC4H = motor[0]>>8; OCR4D = (motor[0]&0xFF); //  pin 6
     #else
      TC4H = (((motor[0]-1000)<<1)+16)>>8; OCR4D = ((((motor[0]-
1000)<<1)+16)&0xFF); //  pin 6
     #endif
    #else
     // Timer 1 A & B [1000:2000] => [8000:16000]
     #ifdef EXT_MOTOR_RANGE
      OCR1A = ((motor[0]<<4) - 16000) + 128;
     #elif defined(EXT_MOTOR_64KHZ)
      OCR1A = (motor[0] - 1000) >> 2; // max = 255
     #elif defined(EXT_MOTOR_32KHZ)
      OCR1A = (motor[0] - 1000) >> 1; // max = 511
     #elif defined(EXT_MOTOR_16KHZ)
      OCR1A = motor[0] - 1000;        //  pin 9
     #elif defined(EXT_MOTOR_8KHZ)
      OCR1A = (motor[0]-1000) << 1;   //  pin 9
     #else
```

```
      OCR1A = motor[0]<<3; //  pin 9
     #endif
    #endif
   #endif
   #if (NUMBER_MOTOR > 1)
    #ifdef EXT_MOTOR_RANGE
     OCR1B = ((motor[1]<<4) - 16000) + 128;
    #elif defined(EXT_MOTOR_64KHZ)
     OCR1B = (motor[1] - 1000) >> 2;
    #elif defined(EXT_MOTOR_32KHZ)
     OCR1B = (motor[1] - 1000) >> 1;
    #elif defined(EXT_MOTOR_16KHZ)
     OCR1B = motor[1] - 1000;       //  pin 10
    #elif defined(EXT_MOTOR_8KHZ)
     OCR1B = (motor[1]-1000) << 1;   //  pin 10
    #else
     OCR1B = motor[1]<<3; //  pin 10
    #endif
   #endif
   #if (NUMBER_MOTOR > 2) // Timer 4 A & D [1000:2000] => [1000:2000]
    #if !defined(HWPWM6)
     // to write values > 255 to timer 4 A/B we need to split the bytes
     #ifndef EXT_MOTOR_RANGE
      TC4H = (2047-motor[2])>>8; OCR4A = ((2047-motor[2])&0xFF); //  pin 5
     #else
      TC4H = 2047-(((motor[2]-1000)<<1)+16)>>8; OCR4A = (2047-(((motor[2]-
1000)<<1)+16)&0xFF); //  pin 5
     #endif
    #else
     #ifdef EXT_MOTOR_RANGE
      OCR3A = ((motor[2]<<4) - 16000) + 128;
     #elif defined(EXT_MOTOR_64KHZ)
      OCR3A = (motor[2] - 1000) >> 2;
     #elif defined(EXT_MOTOR_32KHZ)
      OCR3A = (motor[2] - 1000) >> 1;
     #elif defined(EXT_MOTOR_16KHZ)
      OCR3A = motor[2] - 1000;       //  pin 5
     #elif defined(EXT_MOTOR_8KHZ)
      OCR3A = (motor[2]-1000) << 1;   //  pin 5
     #else
      OCR3A = motor[2]<<3; //  pin 5
     #endif
    #endif
   #endif
   #if (NUMBER_MOTOR > 3)
    #ifdef EXT_MOTOR_RANGE
     TC4H = (((motor[3]-1000)<<1)+16)>>8; OCR4D = ((((motor[3]-
1000)<<1)+16)&0xFF); //  pin 6
    #elif defined(EXT_MOTOR_64KHZ)
     Temp2 = Temp2 >> 2;
```

```
      TC4H = Temp2 >> 8;
      OCR4D = Temp2 & 0xFF;          //  pin 6
    #elif defined(EXT_MOTOR_32KHZ)
      Temp2 = Temp2 >> 1;
      TC4H = Temp2 >> 8;
      OCR4D = Temp2 & 0xFF;          //  pin 6
    #elif defined(EXT_MOTOR_16KHZ)
      TC4H = Temp2 >> 8;
      OCR4D = Temp2 & 0xFF;          //  pin 6
    #elif defined(EXT_MOTOR_8KHZ)
      TC4H = Temp2 >> 8;
      OCR4D = Temp2 & 0xFF;          //  pin 6
    #else
      TC4H = motor[3]>>8; OCR4D = (motor[3]&0xFF); //  pin 6
    #endif
   #endif
   #if (NUMBER_MOTOR > 4)
    #if !defined(HWPWM6)
     #if (NUMBER_MOTOR == 6) && !defined(SERVO)
      atomicPWM_PIN5_highState = motor[4]<<3;
      atomicPWM_PIN5_lowState = 16383-atomicPWM_PIN5_highState;
      atomicPWM_PIN6_highState = motor[5]<<3;
      atomicPWM_PIN6_lowState = 16383-atomicPWM_PIN6_highState;
     #else
      atomicPWM_PIN5_highState = ((motor[4]-1000)<<4)+320;
      atomicPWM_PIN5_lowState = 15743-atomicPWM_PIN5_highState;
      atomicPWM_PIN6_highState = ((motor[5]-1000)<<4)+320;
      atomicPWM_PIN6_lowState = 15743-atomicPWM_PIN6_highState;
     #endif
    #else
     #ifndef EXT_MOTOR_RANGE
      OCR1C = motor[4]<<3; //  pin 11
      TC4H = motor[5]>>8; OCR4A = (motor[5]&0xFF); //  pin 13
     #else
      OCR1C = ((motor[4]<<4) - 16000) + 128;
      TC4H = (((motor[5]-1000)<<1)+16)>>8; OCR4A = ((((motor[5]-
1000)<<1)+16)&0xFF); //  pin 13
     #endif
    #endif
   #endif
   #if (NUMBER_MOTOR > 6)
    #if !defined(HWPWM6)
     atomicPWM_PINA2_highState = ((motor[6]-1000)<<4)+320;
     atomicPWM_PINA2_lowState = 15743-atomicPWM_PINA2_highState;
     atomicPWM_PIN12_highState = ((motor[7]-1000)<<4)+320;
     atomicPWM_PIN12_lowState = 15743-atomicPWM_PIN12_highState;
    #else
     atomicPWM_PINA2_highState = ((motor[6]-1000)>>2)+5;
     atomicPWM_PINA2_lowState = 245-atomicPWM_PINA2_highState;
     atomicPWM_PIN12_highState = ((motor[7]-1000)>>2)+5;
```

```
      atomicPWM_PIN12_lowState = 245-atomicPWM_PIN12_highState;
    #endif
   #endif
  #endif

  /******** Specific PWM Timers & Registers for the atmega328P (Promini)
************/
  #if defined(PROMINI)
   #if (NUMBER_MOTOR > 0)
    #ifdef EXT_MOTOR_RANGE          // 490Hz
     OCR1A = ((motor[0]>>2) - 250);
    #elif defined(EXT_MOTOR_32KHZ)
     OCR1A = (motor[0] - 1000) >> 2; //  pin 9
    #elif defined(EXT_MOTOR_4KHZ)
     OCR1A = (motor[0] - 1000) << 1;
    #elif defined(EXT_MOTOR_1KHZ)
     OCR1A = (motor[0] - 1000) << 3;
    #else
     OCR1A = motor[0]>>3; //  pin 9
    #endif
   #endif
   #if (NUMBER_MOTOR > 1)
    #ifdef EXT_MOTOR_RANGE          // 490Hz
     OCR1B = ((motor[1]>>2) - 250);
    #elif defined(EXT_MOTOR_32KHZ)
     OCR1B = (motor[1] - 1000) >> 2; //  pin 10
    #elif defined(EXT_MOTOR_4KHZ)
     OCR1B = (motor[1] - 1000) << 1;
    #elif defined(EXT_MOTOR_1KHZ)
     OCR1B = (motor[1] - 1000) << 3;
    #else
     OCR1B = motor[1]>>3; //  pin 10
    #endif
   #endif
   #if (NUMBER_MOTOR > 2)
    #ifdef EXT_MOTOR_RANGE          // 490Hz
     OCR2A = ((motor[2]>>2) - 250);
    #elif defined(EXT_MOTOR_32KHZ)
     OCR2A = (motor[2] - 1000) >> 2; //  pin 11
    #elif defined(EXT_MOTOR_4KHZ)
     OCR2A = (motor[2] - 1000) >> 2;
    #elif defined(EXT_MOTOR_1KHZ)
     OCR2A = (motor[2] - 1000) >> 2;
    #else
     OCR2A = motor[2]>>3; //  pin 11
    #endif
   #endif
   #if (NUMBER_MOTOR > 3)
    #ifdef EXT_MOTOR_RANGE          // 490Hz
     OCR2B = ((motor[3]>>2) - 250);
```

```
    #elif defined(EXT_MOTOR_32KHZ)
      OCR2B = (motor[3] - 1000) >> 2; //  pin 3
    #elif defined(EXT_MOTOR_4KHZ)
      OCR2B = (motor[3] - 1000) >> 2;
    #elif defined(EXT_MOTOR_1KHZ)
      OCR2B = (motor[3] - 1000) >> 2;
    #else
      OCR2B = motor[3]>>3; //  pin 3
    #endif
   #endif
   #if (NUMBER_MOTOR > 4)
    #if (NUMBER_MOTOR == 6) && !defined(SERVO)
     #ifndef EXT_MOTOR_RANGE
      atomicPWM_PIN6_highState = motor[4]>>3;
      atomicPWM_PIN5_highState = motor[5]>>3;
     #else
      atomicPWM_PIN6_highState = (motor[4]>>2) - 250;
      atomicPWM_PIN5_highState = (motor[5]>>2) - 250;
     #endif
     atomicPWM_PIN6_lowState  = 255-atomicPWM_PIN6_highState;
     atomicPWM_PIN5_lowState  = 255-atomicPWM_PIN5_highState;
    #else //note: EXT_MOTOR_RANGE not possible here
     atomicPWM_PIN6_highState = ((motor[4]-1000)>>2)+5;
     atomicPWM_PIN6_lowState  = 245-atomicPWM_PIN6_highState;
     atomicPWM_PIN5_highState = ((motor[5]-1000)>>2)+5;
     atomicPWM_PIN5_lowState  = 245-atomicPWM_PIN5_highState;
    #endif
   #endif
   #if (NUMBER_MOTOR > 6) //note: EXT_MOTOR_RANGE not possible here
    atomicPWM_PINA2_highState = ((motor[6]-1000)>>2)+5;
    atomicPWM_PINA2_lowState  = 245-atomicPWM_PINA2_highState;
    atomicPWM_PIN12_highState = ((motor[7]-1000)>>2)+5;
    atomicPWM_PIN12_lowState  = 245-atomicPWM_PIN12_highState;
   #endif
  #endif
}

/**************************************************************************
******************/
/************          Writes the mincommand to all Motors
******************/
/**************************************************************************
******************/
void writeAllMotors(int16_t mc) {   // Sends commands to all motors
  for (uint8_t i =0;i<NUMBER_MOTOR;i++) {
    motor[i]=mc;
  }
  writeMotors();
}
```

```
/************************************************************************
******************/
/************        Initialize the PWM Timers and Registers
******************/
/************************************************************************
******************/
void initOutput() {
  /****************           mark all PWM pins as Output
******************/
  for(uint8_t i=0;i<NUMBER_MOTOR;i++) {
    pinMode(PWM_PIN[i],OUTPUT);
  }

  /****************  Specific PWM Timers & Registers for the MEGA's
******************/
  #if defined(MEGA)
    #if (NUMBER_MOTOR > 0)
      // init 16bit timer 3
      TCCR3A |= (1<<WGM31); // phase correct mode
      TCCR3A &= ~(1<<WGM30);
      TCCR3B |= (1<<WGM33);
      TCCR3B &= ~(1<<CS31); // no prescaler
      ICR3   |= 0x3FFF; // TOP to 16383;

      TCCR3A |= _BV(COM3C1); // connect pin 3 to timer 3 channel C
    #endif
    #if (NUMBER_MOTOR > 1)
      TCCR3A |= _BV(COM3A1); // connect pin 5 to timer 3 channel A
    #endif
    #if (NUMBER_MOTOR > 2)
      // init 16bit timer 4
      TCCR4A |= (1<<WGM41); // phase correct mode
      TCCR4A &= ~(1<<WGM40);
      TCCR4B |= (1<<WGM43);
      TCCR4B &= ~(1<<CS41); // no prescaler
      ICR4   |= 0x3FFF; // TOP to 16383;

      TCCR4A |= _BV(COM4A1); // connect pin 6 to timer 4 channel A
    #endif
    #if (NUMBER_MOTOR > 3)
      TCCR3A |= _BV(COM3B1); // connect pin 2 to timer 3 channel B
    #endif
    #if (NUMBER_MOTOR > 4)
      TCCR4A |= _BV(COM4B1); // connect pin 7 to timer 4 channel B
      TCCR4A |= _BV(COM4C1); // connect pin 8 to timer 4 channel C
    #endif
    #if (NUMBER_MOTOR > 6)
      // timer 2 is a 8bit timer so we cant change its range
      TCCR2A |= _BV(COM2B1); // connect pin 9 to timer 2 channel B
      TCCR2A |= _BV(COM2A1); // connect pin 10 to timer 2 channel A
```

```
  #endif
 #endif

 /******** Specific PWM Timers & Registers for the atmega32u4 (Promicro)
**********/
 #if defined(PROMICRO)
  #if defined(EXT_MOTOR_64KHZ) || defined(EXT_MOTOR_32KHZ) ||
defined(EXT_MOTOR_16KHZ) || defined(EXT_MOTOR_8KHZ)
    TCCR1A = (1<<WGM11);
    TCCR1B = (1<<WGM13) | (1<<WGM12) | (1<<CS10);
    TCCR3A = (1<<WGM31);
    TCCR3B = (1<<WGM33) | (1<<WGM32) | (1<<CS30);
    #if defined(EXT_MOTOR_64KHZ)
     ICR1   = 0x00FF; // TOP to 255;
     ICR3   = 0x00FF; // TOP to 255;
     TC4H = 0x00;
     OCR4C = 0xFF; // phase and frequency correct mode & top to 255
     TCCR4B = (1<<CS40);           // prescaler to 1
    #elif defined(EXT_MOTOR_32KHZ)
     ICR1   = 0x01FF; // TOP to 511;
     ICR3   = 0x01FF; // TOP to 511;
     TC4H = 0x01;
     OCR4C = 0xFF; // phase and frequency correct mode & top to 511
     TCCR4B = (1<<CS40);           // prescaler to 1
    #elif defined(EXT_MOTOR_16KHZ)
     ICR1   = 0x03FF; // TOP to 1023;
     ICR3   = 0x03FF; // TOP to 1023;
     TC4H = 0x03;
     OCR4C = 0xFF; // phase and frequency correct mode & top to 1023
     TCCR4B = (1<<CS40);           // prescaler to 1
    #elif defined(EXT_MOTOR_8KHZ)
     ICR1   = 0x07FF; // TOP to 2046;
     ICR3   = 0x07FF; // TOP to 2046;
     TC4H = 0x3;
     OCR4C = 0xFF; // phase and frequency correct mode
     TCCR4B = (1<<CS41);           // prescaler to 2
    #endif
    TCCR1A |= _BV(COM1A1); // connect pin 9 to timer 1 channel A
    TCCR1A |= _BV(COM1B1); // connect pin 10 to timer 1 channel B
    TCCR3A |= _BV(COM3A1); // connect pin 5 to timer 3 channel A
    TCCR4D = 0;
    TCCR4C |= (1<<COM4D1)|(1<<PWM4D); // connect pin 6 to timer 4 channel D
   #else
    #if (NUMBER_MOTOR > 0) && ( !defined(A32U4_4_HW_PWM_SERVOS) )
     TCCR1A |= (1<<WGM11); // phase correct mode & no prescaler
     TCCR1A &= ~(1<<WGM10);
     TCCR1B &= ~(1<<WGM12) &  ~(1<<CS11) & ~(1<<CS12);
     TCCR1B |= (1<<WGM13) | (1<<CS10);
     ICR1   |= 0x3FFF; // TOP to 16383;
     TCCR1A |= _BV(COM1A1); // connect pin 9 to timer 1 channel A
```

```
    #endif
    #if (NUMBER_MOTOR > 1)
      TCCR1A |= _BV(COM1B1); // connect pin 10 to timer 1 channel B
    #endif
    #if (NUMBER_MOTOR > 2)
      #if !defined(HWPWM6) // timer 4A
        TCCR4E |= (1<<ENHC4); // enhanced pwm mode
        TCCR4B &= ~(1<<CS41); TCCR4B |= (1<<CS42)|(1<<CS40); // prescaler to
16
        TCCR4D |= (1<<WGM40); TC4H = 0x3; OCR4C = 0xFF; // phase and
frequency correct mode & top to 1023 but with enhanced pwm mode we have 2047
        TCCR4A |= (1<<COM4A0)|(1<<PWM4A); // connect pin 5 to timer 4 channel
A
      #else // timer 3A
        TCCR3A |= (1<<WGM31); // phase correct mode & no prescaler
        TCCR3A &= ~(1<<WGM30);
        TCCR3B &= ~(1<<WGM32) &  ~(1<<CS31) & ~(1<<CS32);
        TCCR3B |= (1<<WGM33) | (1<<CS30);
        ICR3   |= 0x3FFF; // TOP to 16383;
        TCCR3A |= _BV(COM3A1); // connect pin 5 to timer 3 channel A
      #endif
    #endif
    #if (NUMBER_MOTOR > 3) || ( (NUMBER_MOTOR > 0) &&
defined(A32U4_4_HW_PWM_SERVOS) )
      #if defined(HWPWM6)
        TCCR4E |= (1<<ENHC4); // enhanced pwm mode
        TCCR4B &= ~(1<<CS41); TCCR4B |= (1<<CS42)|(1<<CS40); // prescaler to
16
        TCCR4D |= (1<<WGM40); TC4H = 0x3; OCR4C = 0xFF; // phase and
frequency correct mode & top to 1023 but with enhanced pwm mode we have 2047
      #endif
      TCCR4C |= (1<<COM4D1)|(1<<PWM4D); // connect pin 6 to timer 4 channel
D
    #endif
    #if (NUMBER_MOTOR > 4)
      #if defined(HWPWM6)
        TCCR1A |= _BV(COM1C1); // connect pin 11 to timer 1 channel C
        TCCR4A |= (1<<COM4A1)|(1<<PWM4A); // connect pin 13 to timer 4
channel A
      #else
        initializeSoftPWM();
      #endif
    #endif
    #if (NUMBER_MOTOR > 6)
      #if defined(HWPWM6)
        initializeSoftPWM();
      #endif
    #endif
  #endif
 #endif
```

```
  /******** Specific PWM Timers & Registers for the atmega328P (Promini)
**********/
  #if defined(PROMINI)
    #if defined(EXT_MOTOR_32KHZ)
      TCCR1A = (1<<WGM11); // phase correct mode & no prescaler
      TCCR1B = (1<<WGM13) | (1<<CS10);
      ICR1   = 0x00FF; // TOP to 255;
      TCCR2B =  (1<<CS20);
    #elif defined(EXT_MOTOR_4KHZ)
      TCCR1A = (1<<WGM11); // phase correct mode & no prescaler
      TCCR1B = (1<<WGM13) | (1<<CS10);
      ICR1   = 0x07F8; // TOP to 1023;
      TCCR2B =  (1<<CS21);
    #elif defined(EXT_MOTOR_1KHZ)
      TCCR1A = (1<<WGM11); // phase correct mode & no prescaler
      TCCR1B = (1<<WGM13) | (1<<CS10);
      ICR1   = 0x1FE0; // TOP to 8184;
      TCCR2B =  (1<<CS20) | (1<<CS21);
    #endif

    #if (NUMBER_MOTOR > 0)
      TCCR1A |= _BV(COM1A1); // connect pin 9 to timer 1 channel A
    #endif
    #if (NUMBER_MOTOR > 1)
      TCCR1A |= _BV(COM1B1); // connect pin 10 to timer 1 channel B
    #endif
    #if (NUMBER_MOTOR > 2)
      TCCR2A |= _BV(COM2A1); // connect pin 11 to timer 2 channel A
    #endif
    #if (NUMBER_MOTOR > 3)
      TCCR2A |= _BV(COM2B1); // connect pin 3 to timer 2 channel B
    #endif
    #if (NUMBER_MOTOR > 4)  // PIN 5 & 6 or A0 & A1
      initializeSoftPWM();
      #if defined(A0_A1_PIN_HEX) || (NUMBER_MOTOR > 6)
        pinMode(5,INPUT);pinMode(6,INPUT);     // we reactivate the INPUT
affectation for these two PINs
        pinMode(A0,OUTPUT);pinMode(A1,OUTPUT);
      #endif
    #endif
  #endif

  /******** special version of MultiWii to calibrate all attached ESCs
**********/
  #if defined(ESC_CALIB_CANNOT_FLY)
    writeAllMotors(ESC_CALIB_HIGH);
    blinkLED(2,20, 2);
    delay(4000);
    writeAllMotors(ESC_CALIB_LOW);
```

```
    blinkLED(3,20, 2);
    while (1) {
      delay(5000);
      blinkLED(4,20, 2);
      SET_ALARM_BUZZER(ALRM_FAC_CONFIRM,
ALRM_LVL_CONFIRM_2);
    }
    exit; // statement never reached
  #endif

  writeAllMotors(MINCOMMAND);
  delay(300);
  #if defined(SERVO)
    initializeServo();
  #endif
}


#if defined(SERVO)
/**************************************************************************
*****************/
/************            Initialize the PWM Servos            ******************/
/**************************************************************************
*****************/
void initializeServo() {
  #if !defined(HW_PWM_SERVOS)
  // do pins init
    #if (PRI_SERVO_FROM == 1) || (SEC_SERVO_FROM == 1)
      SERVO_1_PINMODE;
    #endif
    #if (PRI_SERVO_FROM <= 2 && PRI_SERVO_TO >= 2) ||
(SEC_SERVO_FROM <= 2 && SEC_SERVO_TO >= 2)
      SERVO_2_PINMODE;
    #endif
    #if (PRI_SERVO_FROM <= 3 && PRI_SERVO_TO >= 3) ||
(SEC_SERVO_FROM <= 3 && SEC_SERVO_TO >= 3)
      SERVO_3_PINMODE;
    #endif
    #if (PRI_SERVO_FROM <= 4 && PRI_SERVO_TO >= 4) ||
(SEC_SERVO_FROM <= 4 && SEC_SERVO_TO >= 4)
      SERVO_4_PINMODE;
    #endif
    #if (PRI_SERVO_FROM <= 5 && PRI_SERVO_TO >= 5) ||
(SEC_SERVO_FROM <= 5 && SEC_SERVO_TO >= 5)
      SERVO_5_PINMODE;
    #endif
    #if (PRI_SERVO_FROM <= 6 && PRI_SERVO_TO >= 6) ||
(SEC_SERVO_FROM <= 6 && SEC_SERVO_TO >= 6)
      SERVO_6_PINMODE;
    #endif
```

```
    #if (PRI_SERVO_FROM <= 7 && PRI_SERVO_TO >= 7) ||
(SEC_SERVO_FROM <= 7 && SEC_SERVO_TO >= 7)
      SERVO_7_PINMODE;
    #endif
    #if (PRI_SERVO_FROM <= 8 && PRI_SERVO_TO >= 8) ||
(SEC_SERVO_FROM <= 8 && SEC_SERVO_TO >= 8)
      SERVO_8_PINMODE;
    #endif
  #endif

  #if defined(SERVO_1_HIGH)
    #if defined(PROMINI) || (defined(PROMICRO) && defined(HWPWM6)) // uses
timer 0 Comperator A (8 bit)
      TCCR0A = 0; // normal counting mode
      TIMSK0 |= (1<<OCIE0A); // Enable CTC interrupt
      #define SERVO_ISR TIMER0_COMPA_vect
      #define SERVO_CHANNEL OCR0A
      #define SERVO_1K_US 250
    #endif
    #if (defined(PROMICRO) && !defined(HWPWM6)) // uses timer 3 Comperator A
(11 bit)
      TCCR3A &= ~(1<<WGM30) & ~(1<<WGM31); //normal counting & no
prescaler
      TCCR3B &= ~(1<<WGM32) & ~(1<<CS31) & ~(1<<CS32) & ~(1<<WGM33);
      TCCR3B |= (1<<CS30);
      TIMSK3 |= (1<<OCIE3A); // Enable CTC interrupt
      #define SERVO_ISR TIMER3_COMPA_vect
      #define SERVO_CHANNEL OCR3A
      #define SERVO_1K_US 16000
    #endif
    #if defined(MEGA) // uses timer 5 Comperator A (11 bit)
      TCCR5A &= ~(1<<WGM50) & ~(1<<WGM51); //normal counting & no
prescaler
      TCCR5B &= ~(1<<WGM52) & ~(1<<CS51) & ~(1<<CS52) & ~(1<<WGM53);
      TCCR5B |= (1<<CS50);
      TIMSK5 |= (1<<OCIE5A); // Enable CTC interrupt
      #define SERVO_ISR TIMER5_COMPA_vect
      #define SERVO_CHANNEL OCR5A
      #define SERVO_1K_US 16000
    #endif
  #endif

  #if defined(MEGA) && defined(MEGA_HW_PWM_SERVOS)
    #if defined(SERVO_RFR_RATE)
      #if (SERVO_RFR_RATE < 20)
        #define SERVO_RFR_RATE 20
      #endif
      #if (SERVO_RFR_RATE > 400)
        #define SERVO_RFR_RATE 400
      #endif
```

```
    #else
      #if defined(SERVO_RFR_50HZ)
        #define SERVO_RFR_RATE 50
      #elif defined(SERVO_RFR_160HZ)
        #define SERVO_RFR_RATE 160
      #elif defined(SERVO_RFR_300HZ)
        #define SERVO_RFR_RATE 300
      #endif
    #endif
    #define SERVO_TOP_VAL (uint16_t)(1000000L / SERVO_RFR_RATE)
    // init Timer 5, 1 and 4 of the mega for hw PWM
    TIMSK5 &= ~(1<<OCIE5A); // Disable software PWM
    #if (PRI_SERVO_TO >= 1) || (SEC_SERVO_TO >= 1)
      TCCR5A |= (1<<WGM51);   // phase correct mode & prescaler to 8 = 1us
resolution
      TCCR5A &= ~(1<<WGM50);
      TCCR5B &= ~(1<<WGM52) &  ~(1<<CS50) & ~(1<<CS52);
      TCCR5B |= (1<<WGM53) | (1<<CS51);
      ICR5 = SERVO_TOP_VAL;
      #if (PRI_SERVO_FROM == 1 || SEC_SERVO_FROM == 1)
        pinMode(44,OUTPUT);
        TCCR5A |= (1<<COM5C1); // pin 44
      #endif
      #if (PRI_SERVO_FROM <= 2 && PRI_SERVO_TO >= 2) ||
(SEC_SERVO_FROM <= 2 && SEC_SERVO_TO >= 2)
        pinMode(45,OUTPUT);
        TCCR5A |= (1<<COM5B1); // pin 45
      #endif
      #if (PRI_SERVO_FROM <= 3 && PRI_SERVO_TO >= 3) ||
(SEC_SERVO_FROM <= 3 && SEC_SERVO_TO >= 3)
        pinMode(46,OUTPUT);
        TCCR5A |= (1<<COM5A1); // pin 46
      #endif
    #endif
    #if (PRI_SERVO_TO >= 4) || (SEC_SERVO_TO >= 4)
      TCCR1A |= (1<<WGM11); // phase correct mode & prescaler to 8
      TCCR1A &= ~(1<<WGM10);
      TCCR1B &= ~(1<<WGM12) &  ~(1<<CS10) & ~(1<<CS12);
      TCCR1B |= (1<<WGM13) | (1<<CS11);
      ICR1 = SERVO_TOP_VAL;
      #if (PRI_SERVO_FROM <= 4 && PRI_SERVO_TO >= 4) ||
(SEC_SERVO_FROM <= 4 && SEC_SERVO_TO >= 4)
        pinMode(11, OUTPUT);
        TCCR1A |= (1<<COM1A1); // pin 11
      #endif
      #if (PRI_SERVO_FROM <= 5 && PRI_SERVO_TO >= 5) ||
(SEC_SERVO_FROM <= 5 && SEC_SERVO_TO >= 5)
        pinMode(12,OUTPUT);
        TCCR1A |= (1<<COM1B1); // pin 12
      #endif
```

```
    #endif
    #if (PRI_SERVO_TO >= 6) || (SEC_SERVO_TO >= 6)
      // init 16bit timer 4
      TCCR4A |= (1<<WGM41); // phase correct mode
      TCCR4A &= ~(1<<WGM40);
      TCCR4B &= ~(1<<WGM42) &  ~(1<<CS40) & ~(1<<CS42);
      TCCR4B |= (1<<WGM43) | (1<<CS41);
      ICR4 = SERVO_TOP_VAL;
      #if (PRI_SERVO_FROM <= 6 && PRI_SERVO_TO >= 6) ||
(SEC_SERVO_FROM <= 6 && SEC_SERVO_TO >= 6)
        pinMode(6,OUTPUT);
        TCCR4A |= _BV(COM4A1); // connect pin 6 to timer 4 channel A
      #endif
      #if (PRI_SERVO_FROM <= 7 && PRI_SERVO_TO >= 7) ||
(SEC_SERVO_FROM <= 7 && SEC_SERVO_TO >= 7)
        pinMode(7,OUTPUT);
        TCCR4A |= _BV(COM4B1); // connect pin 7 to timer 4 channel B
      #endif
      #if (PRI_SERVO_FROM <= 8 && PRI_SERVO_TO >= 8) ||
(SEC_SERVO_FROM <= 8 && SEC_SERVO_TO >= 8)
        #if defined(AIRPLANE) || defined(HELICOPTER)
          servo[7] =  MINCOMMAND;    // Trhottle at minimum for airplane and heli
          OCR4C = MINCOMMAND;
        #endif
        pinMode(8,OUTPUT);
        TCCR4A |= _BV(COM4C1); // connect pin 8 to timer 4 channel C
      #endif
    #endif
  #endif // mega hw pwm

  #if defined(PROMICRO) && defined(A32U4_4_HW_PWM_SERVOS)
    // atm. always initialize 4 servos to pins 9, 10, 11, 5
    TIMSK1 &= ~(1<<OCIE1A) & ~(1<<OCIE1B) & ~(1<<OCIE1C);
    TCCR1A |= (1<<WGM11); // phase correct mode & prescaler to 8
    TCCR1A &= ~(1<<WGM10);
    TCCR1B &= ~(1<<WGM12) &  ~(1<<CS10) & ~(1<<CS12);
    TCCR1B |= (1<<WGM13) | (1<<CS11);
    pinMode(9,OUTPUT);
    TCCR1A |= (1<<COM1A1); // pin 9
    pinMode(10,OUTPUT);
    TCCR1A |= (1<<COM1B1); // pin 10
    pinMode(11,OUTPUT);
    TCCR1A |= (1<<COM1C1); // pin 11

    TCCR3A |= (1<<WGM31); // phase correct mode & prescaler to 8
    TCCR3A &= ~(1<<WGM30);
    TCCR3B &= ~(1<<WGM32) &  ~(1<<CS30) & ~(1<<CS32);
    TCCR3B |= (1<<WGM33) | (1<<CS31);
    pinMode(5,OUTPUT);
    TCCR3A |= (1<<COM3A1); // pin 5
```

```
    #if defined(SERVO_RFR_RATE)
      #if (SERVO_RFR_RATE < 50) || (SERVO_RFR_RATE > 400)
        #error "* invalid SERVO_RFR_RATE specified"
      #endif
      #define SERVO_TOP_VAL (uint16_t)(1000000L / SERVO_RFR_RATE)
    #elif defined(SERVO_RFR_50HZ)
      #define SERVO_TOP_VAL 16700
    #elif defined(SERVO_RFR_160HZ)
      #define SERVO_TOP_VAL 6200
    #elif defined(SERVO_RFR_300HZ)
      #define SERVO_TOP_VAL 3300
    #else
      #error "* must set SERVO_RFR_RATE or one of the fixed refresh rates of 50,
160 or 300 Hz"
    #endif
    #if defined(SERVO_PIN5_RFR_RATE)
      #if (SERVO_PIN5_RFR_RATE < 50) || (SERVO_PIN5_RFR_RATE > 400)
        #error "* invalid SERVO_PIN5_RFR_RATE specified"
      #endif
      #define SERVO_PIN5_TOP_VAL (uint16_t)(1000000L /
SERVO_PIN5_RFR_RATE)
    #else
      #define SERVO_PIN5_TOP_VAL SERVO_TOP_VAL
    #endif
    ICR1   = SERVO_TOP_VAL;      // set TOP timer 1
    ICR3   = SERVO_PIN5_TOP_VAL; // set TOP timer 3
  #endif // promicro hw pwm
}

/*****************************************************************************
******************/
/************            Servo software PWM generation
******************/
/*****************************************************************************
******************/

// prescaler is set by default to 64 on Timer0
// Duemilanove : 16MHz / 64 => 4 us
// 256 steps = 1 counter cycle = 1024 us

// for servo 2-8
// its almost the same as for servo 1
#if defined(SERVO_1_HIGH) && !defined(A32U4_4_HW_PWM_SERVOS)
  #define
SERVO_PULSE(PIN_HIGH,ACT_STATE,SERVO_NUM,LAST_PIN_LOW) \
    }else if(state == ACT_STATE){                          \
      LAST_PIN_LOW;                                    \
      PIN_HIGH;                                  \
      SERVO_CHANNEL+=SERVO_1K_US;                          \
      state++;                                     \
```

```
  }else if(state == ACT_STATE+1){                          \
    SERVO_CHANNEL+=atomicServo[SERVO_NUM];                 \
    state++;                                               \

ISR(SERVO_ISR) {
  static uint8_t state = 0; // indicates the current state of the chain
  if(state == 0){
    SERVO_1_HIGH; // set servo 1's pin high
    SERVO_CHANNEL+=SERVO_1K_US; // wait 1000us
    state++; // count up the state
  }else if(state==1){
    SERVO_CHANNEL+=atomicServo[SERVO_1_ARR_POS]; // load the servo's
value (0-1000us)
    state++; // count up the state
  #if defined(SERVO_2_HIGH)
    SERVO_PULSE(SERVO_2_HIGH,2,SERVO_2_ARR_POS,SERVO_1_LOW); //
the same here
  #endif
  #if defined(SERVO_3_HIGH)
    SERVO_PULSE(SERVO_3_HIGH,4,SERVO_3_ARR_POS,SERVO_2_LOW);
  #endif
  #if defined(SERVO_4_HIGH)
    SERVO_PULSE(SERVO_4_HIGH,6,SERVO_4_ARR_POS,SERVO_3_LOW);
  #endif
  #if defined(SERVO_5_HIGH)
    SERVO_PULSE(SERVO_5_HIGH,8,SERVO_5_ARR_POS,SERVO_4_LOW);
  #endif
  #if defined(SERVO_6_HIGH)
    SERVO_PULSE(SERVO_6_HIGH,10,SERVO_6_ARR_POS,SERVO_5_LOW);
  #endif
  #if defined(SERVO_7_HIGH)
    SERVO_PULSE(SERVO_7_HIGH,12,SERVO_7_ARR_POS,SERVO_6_LOW);
  #endif
  #if defined(SERVO_8_HIGH)
    SERVO_PULSE(SERVO_8_HIGH,14,SERVO_8_ARR_POS,SERVO_7_LOW);
  #endif
  }else{
    LAST_LOW;
    #if defined(SERVO_RFR_300HZ)
      #if defined(SERVO_3_HIGH)  // if there are 3 or more servos we dont need to
slow it down
        SERVO_CHANNEL+=(SERVO_1K_US>>3); // 0 would be better but it causes
bad jitter
        state=0;
      #else // if there are less then 3 servos we need to slow it to not go over 300Hz
(the highest working refresh rate for the digital servos for what i know..)
        SERVO_CHANNEL+=SERVO_1K_US;
        if(state<4){
          state+=2;
        }else{
```

```
          state=0;
         }
        #endif
      #endif
      #if defined(SERVO_RFR_160HZ)
       #if defined(SERVO_4_HIGH)  // if there are 4 or more servos we dont need to
slow it down
        SERVO_CHANNEL+=(SERVO_1K_US>>3); // 0 would be better but it causes
bad jitter
        state=0;
       #else // if there are less then 4 servos we need to slow it to not go over ~170Hz
(the highest working refresh rate for analog servos)
        SERVO_CHANNEL+=SERVO_1K_US;
        if(state<8){
          state+=2;
        }else{
          state=0;
        }
       #endif
      #endif
      #if defined(SERVO_RFR_50HZ) // to have ~ 50Hz for all servos
       SERVO_CHANNEL+=SERVO_1K_US;
       if(state<30){
         state+=2;
       }else{
         state=0;
       }
      #endif
    }
  }
  #endif
#endif

/************************************************************************
******************/
/************          Motor software PWM generation
******************/
/************************************************************************
******************/
// SW PWM is only used if there are not enough HW PWM pins (for exampe hexa on
a promini)

#if (NUMBER_MOTOR > 4) && (defined(PROMINI) || defined(PROMICRO))

  /****************    Pre define the used ISR's and Timerchannels
******************/
  #if !defined(PROMICRO)
    #define SOFT_PWM_ISR1 TIMER0_COMPB_vect
    #define SOFT_PWM_ISR2 TIMER0_COMPA_vect
    #define SOFT_PWM_CHANNEL1 OCR0B
```

```
  #define SOFT_PWM_CHANNEL2 OCR0A
 #elif !defined(HWPWM6)
  #define SOFT_PWM_ISR1 TIMER3_COMPB_vect
  #define SOFT_PWM_ISR2 TIMER3_COMPC_vect
  #define SOFT_PWM_CHANNEL1 OCR3B
  #define SOFT_PWM_CHANNEL2 OCR3C
 #else
  #define SOFT_PWM_ISR2 TIMER0_COMPB_vect
  #define SOFT_PWM_CHANNEL2 OCR0B
 #endif

 /****************        Initialize Timers and PWM Channels
******************/
 void initializeSoftPWM(void) {
  #if !defined(PROMICRO)
   TCCR0A = 0; // normal counting mode
   #if (NUMBER_MOTOR > 4) && !defined(HWPWM6)
    TIMSK0 |= (1<<OCIE0B); // Enable CTC interrupt
   #endif
   #if (NUMBER_MOTOR > 6) || ((NUMBER_MOTOR == 6) &&
!defined(SERVO))
    TIMSK0 |= (1<<OCIE0A);
   #endif
  #else
   #if !defined(HWPWM6)
    TCCR3A &= ~(1<<WGM30) & ~(1<<WGM31); //normal counting & no
prescaler
    TCCR3B &= ~(1<<WGM32) & ~(1<<CS31) & ~(1<<CS32) &
~(1<<WGM33);
    TCCR3B |= (1<<CS30);
    TIMSK3 |= (1<<OCIE3B); // Enable CTC interrupt
    #if (NUMBER_MOTOR > 6) || ((NUMBER_MOTOR == 6) &&
!defined(SERVO))
     TIMSK3 |= (1<<OCIE3C);
    #endif
   #else
    TCCR0A = 0; // normal counting mode
    TIMSK0 |= (1<<OCIE0B); // Enable CTC interrupt
   #endif
  #endif
 }

 /****************          Motor SW PWM ISR's
******************/
 // hexa with old but sometimes better SW PWM method
 // for setups without servos
 #if (NUMBER_MOTOR == 6) && (!defined(SERVO) && !defined(HWPWM6))
  ISR(SOFT_PWM_ISR1) {
   static uint8_t state = 0;
   if(state == 0){
```

```
      if (atomicPWM_PIN5_highState>0) SOFT_PWM_1_PIN_HIGH;
      SOFT_PWM_CHANNEL1 += atomicPWM_PIN5_highState;
      state = 1;
    }else if(state == 1){
      SOFT_PWM_CHANNEL1 += atomicPWM_PIN5_highState;
      state = 2;
    }else if(state == 2){
      SOFT_PWM_1_PIN_LOW;
      SOFT_PWM_CHANNEL1 += atomicPWM_PIN5_lowState;
      state = 3;
    }else if(state == 3){
      SOFT_PWM_CHANNEL1 += atomicPWM_PIN5_lowState;
      state = 0;
    }
  }
  ISR(SOFT_PWM_ISR2) {
    static uint8_t state = 0;
    if(state == 0){
      if (atomicPWM_PIN6_highState>0) SOFT_PWM_2_PIN_HIGH;
      SOFT_PWM_CHANNEL2 += atomicPWM_PIN6_highState;
      state = 1;
    }else if(state == 1){
      SOFT_PWM_CHANNEL2 += atomicPWM_PIN6_highState;
      state = 2;
    }else if(state == 2){
      SOFT_PWM_2_PIN_LOW;
      SOFT_PWM_CHANNEL2 += atomicPWM_PIN6_lowState;
      state = 3;
    }else if(state == 3){
      SOFT_PWM_CHANNEL2 += atomicPWM_PIN6_lowState;
      state = 0;
    }
  }
#else
  #if (NUMBER_MOTOR > 4) && !defined(HWPWM6)
    // HEXA with just OCR0B
    ISR(SOFT_PWM_ISR1) {
      static uint8_t state = 0;
      if(state == 0){
        SOFT_PWM_1_PIN_HIGH;
        SOFT_PWM_CHANNEL1 += atomicPWM_PIN5_highState;
        state = 1;
      }else if(state == 1){
        SOFT_PWM_2_PIN_LOW;
        SOFT_PWM_CHANNEL1 += atomicPWM_PIN6_lowState;
        state = 2;
      }else if(state == 2){
        SOFT_PWM_2_PIN_HIGH;
        SOFT_PWM_CHANNEL1 += atomicPWM_PIN6_highState;
        state = 3;
```

```
      }else if(state == 3){
        SOFT_PWM_1_PIN_LOW;
        SOFT_PWM_CHANNEL1 += atomicPWM_PIN5_lowState;
        state = 0;
      }
    }
  #endif
  //the same with digital PIN A2 & 12 OCR0A counter for OCTO
  #if (NUMBER_MOTOR > 6)
    ISR(SOFT_PWM_ISR2) {
      static uint8_t state = 0;
      if(state == 0){
        SOFT_PWM_3_PIN_HIGH;
        SOFT_PWM_CHANNEL2 += atomicPWM_PINA2_highState;
        state = 1;
      }else if(state == 1){
        SOFT_PWM_4_PIN_LOW;
        SOFT_PWM_CHANNEL2 += atomicPWM_PIN12_lowState;
        state = 2;
      }else if(state == 2){
        SOFT_PWM_4_PIN_HIGH;
        SOFT_PWM_CHANNEL2 += atomicPWM_PIN12_highState;
        state = 3;
      }else if(state == 3){
        SOFT_PWM_3_PIN_LOW;
        SOFT_PWM_CHANNEL2 += atomicPWM_PINA2_lowState;
        state = 0;
      }
    }
  #endif
 #endif
#endif

/************************************************************************
*****************/
/********** Mixes the Computed stabilize values to the Motors & Servos
***************/
/************************************************************************
*****************/

// get servo middle point from Config or from RC-Data
int16_t get_middle(uint8_t nr) {
  return (conf.servoConf[nr].middle < RC_CHANS) ?
rcData[conf.servoConf[nr].middle] : conf.servoConf[nr].middle;
}

// int8_t servodir(uint8_t n, uint8_t b) { return ((conf.servoConf[n].rate & b) ? -1 : 1) ;
}

void mixTable() {
```

```
  int16_t maxMotor;
  uint8_t i;
  #if defined(DYNBALANCE)
    return;
  #endif
  #define PIDMIX(X,Y,Z) rcCommand[THROTTLE] + axisPID[ROLL]*X +
axisPID[PITCH]*Y + YAW_DIRECTION * axisPID[YAW]*Z
  #define SERVODIR(n,b) ((conf.servoConf[n].rate & b) ? -1 : 1)

  /***************          main Mix Table          ******************/
  #if defined( MY_PRIVATE_MIXING )
    #include MY_PRIVATE_MIXING
  #elif defined( BI )
    motor[0] = PIDMIX(+1, 0, 0); //LEFT
    motor[1] = PIDMIX(-1, 0, 0); //RIGHT
    servo[4] = (SERVODIR(4,2) * axisPID[YAW]) + (SERVODIR(4,1) *
axisPID[PITCH]) + get_middle(4); //LEFT
    servo[5] = (SERVODIR(5,2) * axisPID[YAW]) + (SERVODIR(5,1) *
axisPID[PITCH]) + get_middle(5); //RIGHT
  #elif defined( TRI )
    motor[0] = PIDMIX( 0,+4/3, 0); //REAR
    motor[1] = PIDMIX(-1,-2/3, 0); //RIGHT
    motor[2] = PIDMIX(+1,-2/3, 0); //LEFT
    servo[5] = (SERVODIR(5, 1) * axisPID[YAW]) + get_middle(5); //REAR
  #elif defined( QUADP )
    motor[0] = PIDMIX( 0,+1,-1); //REAR
    motor[1] = PIDMIX(-1, 0,+1); //RIGHT
    motor[2] = PIDMIX(+1, 0,+1); //LEFT
    motor[3] = PIDMIX( 0,-1,-1); //FRONT
  #elif defined( QUADX )
    motor[0] = PIDMIX(-1,+1,-1); //REAR_R
    motor[1] = PIDMIX(-1,-1,+1); //FRONT_R
    motor[2] = PIDMIX(+1,+1,+1); //REAR_L
    motor[3] = PIDMIX(+1,-1,-1); //FRONT_L
  #elif defined( Y4 )
    motor[0] = PIDMIX(+0,+1,-1);   //REAR_1 CW
    motor[1] = PIDMIX(-1,-1, 0); //FRONT_R CCW
    motor[2] = PIDMIX(+0,+1,+1);   //REAR_2 CCW
    motor[3] = PIDMIX(+1,-1, 0); //FRONT_L CW
  #elif defined( Y6 )
    motor[0] = PIDMIX(+0,+4/3,+1); //REAR
    motor[1] = PIDMIX(-1,-2/3,-1); //RIGHT
    motor[2] = PIDMIX(+1,-2/3,-1); //LEFT
    motor[3] = PIDMIX(+0,+4/3,-1); //UNDER_REAR
    motor[4] = PIDMIX(-1,-2/3,+1); //UNDER_RIGHT
    motor[5] = PIDMIX(+1,-2/3,+1); //UNDER_LEFT
  #elif defined( HEX6 )
    motor[0] = PIDMIX(-7/8,+1/2,+1); //REAR_R
    motor[1] = PIDMIX(-7/8,-1/2,-1); //FRONT_R
    motor[2] = PIDMIX(+7/8,+1/2,+1); //REAR_L
```

```
  motor[3] = PIDMIX(+7/8,-1/2,-1); //FRONT_L
  motor[4] = PIDMIX(+0  ,-1  ,+1); //FRONT
  motor[5] = PIDMIX(+0  ,+1  ,-1); //REAR
#elif defined( HEX6X )
  motor[0] = PIDMIX(-1/2,+7/8,+1); //REAR_R
  motor[1] = PIDMIX(-1/2,-7/8,+1); //FRONT_R
  motor[2] = PIDMIX(+1/2,+7/8,-1); //REAR_L
  motor[3] = PIDMIX(+1/2,-7/8,-1); //FRONT_L
  motor[4] = PIDMIX(-1  ,+0  ,-1); //RIGHT
  motor[5] = PIDMIX(+1  ,+0  ,+1); //LEFT
#elif defined( HEX6H )
  motor[0] = PIDMIX(-1,+1,-1); //REAR_R
  motor[1] = PIDMIX(-1,-1,+1); //FRONT_R
  motor[2] = PIDMIX(+ 1,+1,+1); //REAR_L
  motor[3] = PIDMIX(+ 1,-1,-1); //FRONT_L
  motor[4] = PIDMIX(0 ,0 ,0); //RIGHT
  motor[5] = PIDMIX(0 ,0 ,0); //LEFT
#elif defined( OCTOX8 )
  motor[0] = PIDMIX(-1,+1,-1); //REAR_R
  motor[1] = PIDMIX(-1,-1,+1); //FRONT_R
  motor[2] = PIDMIX(+1,+1,+1); //REAR_L
  motor[3] = PIDMIX(+1,-1,-1); //FRONT_L
  motor[4] = PIDMIX(-1,+1,+1); //UNDER_REAR_R
  motor[5] = PIDMIX(-1,-1,-1); //UNDER_FRONT_R
  motor[6] = PIDMIX(+1,+1,-1); //UNDER_REAR_L
  motor[7] = PIDMIX(+1,-1,+1); //UNDER_FRONT_L
#elif defined( OCTOFLATP )
  motor[0] = PIDMIX(+7/10,-7/10,+1); //FRONT_L
  motor[1] = PIDMIX(-7/10,-7/10,+1); //FRONT_R
  motor[2] = PIDMIX(-7/10,+7/10,+1); //REAR_R
  motor[3] = PIDMIX(+7/10,+7/10,+1); //REAR_L
  motor[4] = PIDMIX(+0   ,-1   ,-1); //FRONT
  motor[5] = PIDMIX(-1   ,+0   ,-1); //RIGHT
  motor[6] = PIDMIX(+0   ,+1   ,-1); //REAR
  motor[7] = PIDMIX(+1   ,+0   ,-1); //LEFT
#elif defined( OCTOFLATX )
  motor[0] = PIDMIX(+1  ,-1/2,+1); //MIDFRONT_L
  motor[1] = PIDMIX(-1/2,-1  ,+1); //FRONT_R
  motor[2] = PIDMIX(-1  ,+1/2,+1); //MIDREAR_R
  motor[3] = PIDMIX(+1/2,+1  ,+1); //REAR_L
  motor[4] = PIDMIX(+1/2,-1  ,-1); //FRONT_L
  motor[5] = PIDMIX(-1  ,-1/2,-1); //MIDFRONT_R
  motor[6] = PIDMIX(-1/2,+1  ,-1); //REAR_R
  motor[7] = PIDMIX(+1  ,+1/2,-1); //MIDREAR_L
#elif defined( VTAIL4 )
  motor[0] = PIDMIX(+0,+1, +1); //REAR_R
  motor[1] = PIDMIX(-1, -1, +0); //FRONT_R
  motor[2] = PIDMIX(+0,+1, -1); //REAR_L
  motor[3] = PIDMIX(+1, -1, -0); //FRONT_L
#elif defined( FLYING_WING )
```

```
  /*************************          FLYING WING
**************************************/
  if (!f.ARMED) {
    servo[7] = MINCOMMAND;  // Kill throttle when disarmed
  } else {
    servo[7] = constrain(rcCommand[THROTTLE], conf.minthrottle,
MAXTHROTTLE);
  }
  motor[0] = servo[7];
  if (f.PASSTHRU_MODE) {    // do not use sensors for correction, simple 2 channel
mixing
    servo[3] = (SERVODIR(3,1) * rcCommand[PITCH]) + (SERVODIR(3,2) *
rcCommand[ROLL]);
    servo[4] = (SERVODIR(4,1) * rcCommand[PITCH]) + (SERVODIR(4,2) *
rcCommand[ROLL]);
  } else {                // use sensors to correct (gyro only or gyro+acc according to
aux1/aux2 configuration
    servo[3] = (SERVODIR(3,1) * axisPID[PITCH])   + (SERVODIR(3,2) *
axisPID[ROLL]);
    servo[4] = (SERVODIR(4,1) * axisPID[PITCH])   + (SERVODIR(4,2) *
axisPID[ROLL]);
  }
  servo[3] += get_middle(3);
  servo[4] += get_middle(4);
 #elif defined( AIRPLANE )
  /*************************          AIRPLANE
**************************************/
  // servo[7] is programmed with safty features to avoid motorstarts when ardu reset..
  // All other servos go to center at reset..  Half throttle can be dangerus
  // Only use servo[7] as motorcontrol if motor is used in the setup          */
  if (!f.ARMED) {
    servo[7] = MINCOMMAND; // Kill throttle when disarmed
  } else {
    servo[7] = constrain(rcCommand[THROTTLE], conf.minthrottle,
MAXTHROTTLE);
  }
  motor[0] = servo[7];

  // Flapperon Controll TODO - optimalisation
  int16_t flapperons[2]={0,0};
  #if  defined(FLAPPERONS) && defined(FLAPPERON_EP)
    int8_t flapinv[2] = FLAPPERON_INVERT;
    static int16_t F_Endpoint[2] = FLAPPERON_EP;
    int16_t flap =MIDRC-
constrain(rcData[FLAPPERONS],F_Endpoint[0],F_Endpoint[1]);
    static int16_t slowFlaps= flap;
    #if defined(FLAPSPEED)
      if (slowFlaps < flap ){slowFlaps+=FLAPSPEED;}else if(slowFlaps >
flap){slowFlaps-=FLAPSPEED;}
    #else
```

```
      slowFlaps = flap;
     #endif
     flap = MIDRC-(constrain(MIDRC-slowFlaps,F_Endpoint[0],F_Endpoint[1]));
    for(i=0; i<2; i++){flapperons[i] = flap * flapinv[i] ;}
    #endif

    // Traditional Flaps on SERVO3
    #if defined(FLAPS)
     // configure SERVO3 middle point in GUI to using an AUX channel for FLAPS
control
     // use servo min, servo max and servo rate for propper endpoints adjust
     int16_t lFlap = get_middle(2);
     lFlap = constrain(lFlap, conf.servoConf[2].min, conf.servoConf[2].max);
     lFlap = MIDRC - lFlap;
     static int16_t slow_LFlaps= lFlap;
     #if defined(FLAPSPEED)
      if (slow_LFlaps < lFlap ){slow_LFlaps+=FLAPSPEED;} else if(slow_LFlaps >
lFlap){slow_LFlaps-=FLAPSPEED;}
     #else
      slow_LFlaps = lFlap;
     #endif
     servo[2] = ((int32_t)conf.servoConf[2].rate * slow_LFlaps)/100L;
     servo[2] += MIDRC;
    #endif

    if(f.PASSTHRU_MODE){   // Direct passthru from RX
     servo[3] = rcCommand[ROLL] + flapperons[0];     //   Wing 1
     servo[4] = rcCommand[ROLL] + flapperons[1];     //   Wing 2
     servo[5] = rcCommand[YAW];                      //   Rudder
     servo[6] = rcCommand[PITCH];                    //   Elevator
    }else{
     // Assisted modes (gyro only or gyro+acc according to AUX configuration in Gui
     servo[3] = axisPID[ROLL] + flapperons[0];   //   Wing 1
     servo[4] = axisPID[ROLL] + flapperons[1];   //   Wing 2
     servo[5] = axisPID[YAW];                    //   Rudder
     servo[6] = axisPID[PITCH];                  //   Elevator
    }
    for(i=3;i<7;i++) {
     servo[i]  = ((int32_t)conf.servoConf[i].rate * servo[i])/100L;  // servo rates
     servo[i] += get_middle(i);
    }
  #elif defined( SINGLECOPTER )
    /*************************          Single & DualCopter
*****************************/
    // Singlecopter
    // This is a beta requested by  xuant9
    // Assisted modes (gyro only or gyro+acc according to AUX configuration in Gui
    // http://www.kkmulticopter.kr/multicopter/images/blue_single.jpg
    //servo[3]  = (axisPID[YAW]*SERVODIR(3,2)) +
(axisPID[PITCH]*SERVODIR(3,1));   //   SideServo  5  D12
```

```
    //servo[4]  = (axisPID[YAW]*SERVODIR(4,2)) +
(axisPID[PITCH]*SERVODIR(4,1));   //   SideServo  3  D11
    //servo[5]  = (axisPID[YAW]*SERVODIR(5,2)) + (axisPID[ROLL]
*SERVODIR(5,1));   //   FrontServo 2  D3
    //servo[6]  = (axisPID[YAW]*SERVODIR(6,2)) + (axisPID[ROLL]
*SERVODIR(6,1));   //   RearServo  4  D10
    for(i=3;i<7;i++) {
      servo[i]  = (axisPID[YAW] * SERVODIR(i,2)) + (axisPID[(6-i)>>1] *
SERVODIR(i,1));  // mix and setup direction
      servo[i] += get_middle(i);
    }
    motor[0] = rcCommand[THROTTLE];
  #elif defined( DUALCOPTER )
    // Dualcopter
    // This is a beta requested by  xuant9
    // Assisted modes (gyro only or gyro+acc according to AUX configuration in Gui

//http://www.kkmulticopter.kr/products_pic/index.html?sn=multicopter_v02_du&nam
e=KKMultiCopter%20Flight%20Controller%20Blackboard%3Cbr%3E
    //servo[4] = ((int32_t)conf.servoConf[4].rate * axisPID[PITCH])/100L;  //
PITCHServo   D11 => Wing2 servo
    //servo[5] = ((int32_t)conf.servoConf[5].rate * axisPID[ROLL] )/100L;   //
ROLLServo    D3  => Rudder servo
    for(i=4;i<6;i++) {
      servo[i] =  axisPID[5-i] * SERVODIR(i,1);    // mix and setup direction
      servo[i] += get_middle(i);
    }
    motor[0] = PIDMIX(0,0,-1);                          //  Pin D9
    motor[1] = PIDMIX(0,0,+1);                          //  Pin D10

  #elif defined( HELICOPTER )
    /*****************************          HELICOPTERS
************************************/
   // Common controlls for Helicopters
    int16_t heliRoll,heliNick;
    int16_t collRange[3] = COLLECTIVE_RANGE;
    static int16_t   collective;
    #ifdef GOVERNOR_P
      static int16_t last_collective = 0, delta_collective = 0, governorThrottle = 0;
    #endif
    /*************************
     * servo settings Heli.
     *************************/

    // Limit Collective range up/down
    int16_t collect = rcData[COLLECTIVE_PITCH] - (1500 + collRange[1]);
    if   (collect>0) {
      collective = collect * (collRange[2]*0.01);
    } else{
      collective = collect * (collRange[0]*0.01);
```

```
    }

    // maybe collective range can be replaced replaced by this ?
    //collective = rcData[COLLECTIVE_PITCH] - get_middle(7);              // zero
pitch offset
    //collective = ((int32_t)conf.servoConf[7].rate * collective)/100L;    // collective
range

    #ifdef GOVERNOR_P
      delta_collective = collective - last_collective;
      last_collective = collective;
      if (! f.ARMED || ! rcOptions[BOXGOV] || (rcCommand[THROTTLE] <
conf.minthrottle) )
        governorThrottle = 0; // clear subito
      else if (delta_collective > 0) {
        governorThrottle += delta_collective * conf.governorP; // attack
        // avoid overshooting governor (would result in overly long decay phase)
        if (rcCommand[THROTTLE] + governorThrottle > MAXTHROTTLE)
governorThrottle = MAXTHROTTLE - rcCommand[THROTTLE];
      } else {
        static uint8_t d = 0;
        if (! (++d % conf.governorD)) governorThrottle -= 10; // decay; signal stepsize
10 should  be smooth on most ESCs
      }
      if (governorThrottle < 0) governorThrottle = 0; // always beware of negative
impact of governor on throttle
    #endif

    if(f.PASSTHRU_MODE){ // Use Rcdata Without sensors
      heliRoll = rcCommand[ROLL] ;
      heliNick = rcCommand[PITCH];
    } else{ // Assisted modes
      heliRoll = axisPID[ROLL];
      heliNick = axisPID[PITCH];
    }

    // Limit Maximum Rates for Heli
    int16_t cRange[2] = CONTROL_RANGE;
    heliRoll*= cRange[0]*0.01;
    heliNick*= cRange[1]*0.01;

    /* Throttle & YAW
    ******************** */
    // Yaw control is common for Heli 90 & 120
    //servo[5] = (axisPID[YAW] * SERVODIR(5,1)) + conf.servoConf[5].middle;
    // tail precomp from collective
    int16_t acmd = abs(collective) - conf.yawCollPrecompDeadband; // abs collective
minus deadband
    if (acmd > 0 ){
      servo[5] = (axisPID[YAW] * SERVODIR(5,1)) + conf.servoConf[5].middle +
```

```
(acmd * conf.yawCollPrecomp)/10;
   } else {
    servo[5] = (axisPID[YAW] * SERVODIR(5,1)) + conf.servoConf[5].middle;
   }
   #if YAWMOTOR
    servo[5] = constrain(servo[5], conf.servoConf[5].min, conf.servoConf[5].max); //
limit the values
    if (rcCommand[THROTTLE]<conf.minthrottle || !f.ARMED) {servo[5] =
MINCOMMAND;} // Kill YawMotor
   #endif
   if (!f.ARMED){
    servo[7] = MINCOMMAND;          // Kill throttle when disarmed
   } else {
    servo[7] = rcCommand[THROTTLE]; //   50hz ESC or servo
    #ifdef GOVERNOR_P
     servo[7] += governorThrottle;
    #endif
    servo[7] = constrain(servo[7], conf.minthrottle, MAXTHROTTLE);   //  limit min
& max    }
   }
   #ifndef HELI_USE_SERVO_FOR_THROTTLE
    motor[0] = servo[7];     // use real motor output - ESC capable
    #if YAWMOTOR
     motor[1] = servo[5];   // use motor2 output for YAWMOTOR
    #endif
   #endif

   //            ( Collective, Pitch/Nick, Roll ) Change sign to invert

/**************************************************************************
***************************************/
   #define HeliXPIDMIX(Z,Y,X) ( (collRange[1] + collective)*Z + heliNick*Y +
heliRoll*X)/10
   #ifdef HELI_120_CCPM
    static int8_t nickMix[3] = SERVO_NICK;
    static int8_t leftMix[3] = SERVO_LEFT;
    static int8_t rightMix[3]= SERVO_RIGHT;

    servo[3]  =  HeliXPIDMIX( ( SERVODIR(3,4) * nickMix[0]), SERVODIR(3,2) *
nickMix[1], SERVODIR(3,1) * nickMix[2]);   //    NICK  servo
    servo[4]  =  HeliXPIDMIX( ( SERVODIR(4,4) * leftMix[0]), SERVODIR(4,2) *
leftMix[1], SERVODIR(4,1) * leftMix[2]);   //    LEFT  servo
    servo[6]  =  HeliXPIDMIX( ( SERVODIR(6,4) * rightMix[0]),SERVODIR(6,2) *
rightMix[1],SERVODIR(6,1) * rightMix[2]);  //    RIGHT servo
   #endif

/**************************************************************************
***************************************/
   #ifdef HELI_90_DEG
    servo[3]  = HeliXPIDMIX( +0, (conf.servoConf[3].rate/10), -0);      //     NICK
```

```
servo
     servo[4]  = HeliXPIDMIX( +0, +0, (conf.servoConf[4].rate/10));      //     ROLL
servo
     servo[6]  = HeliXPIDMIX( (conf.servoConf[6].rate/10), +0, +0);      //
COLLECTIVE  servo
   #endif
   servo[3] += get_middle(3);
   servo[4] += get_middle(4);
   servo[6] += get_middle(6);
 #elif defined( GIMBAL )
   for(i=0;i<2;i++) {
    servo[i]  = ((int32_t)conf.servoConf[i].rate * att.angle[1-i]) /50L;
    servo[i] += get_middle(i);
   }
 #else
   #error "missing coptertype mixtable entry. Either you forgot to define a copter type
or the mixing table is lacking neccessary code"
 #endif // MY_PRIVATE_MIXING


/*************************************************************************
****************************************/
 /*************************          Cam stabilize Servos
*******************************/

 #if defined(SERVO_TILT)
   servo[0] = get_middle(0);
   servo[1] = get_middle(1);
   if (rcOptions[BOXCAMSTAB]) {
    servo[0] += ((int32_t)conf.servoConf[0].rate * att.angle[PITCH]) /50L;
    servo[1] += ((int32_t)conf.servoConf[1].rate * att.angle[ROLL])  /50L;
   }
 #endif

 #ifdef SERVO_MIX_TILT
   int16_t angleP = get_middle(0) - MIDRC;
   int16_t angleR = get_middle(1) - MIDRC;
   if (rcOptions[BOXCAMSTAB]) {
    angleP += ((int32_t)conf.servoConf[0].rate * att.angle[PITCH]) /50L;
    angleR += ((int32_t)conf.servoConf[1].rate * att.angle[ROLL])  /50L;
   }
   servo[0] = MIDRC+angleP-angleR;
   servo[1] = MIDRC-angleP-angleR;
 #endif

/***************          Cam trigger Servo          *****************/
 #if defined(CAMTRIG)
   // setup MIDDLE for using as camtrig interval (in msec) or RC channel pointer for
interval control
   #define CAM_TIME_LOW  conf.servoConf[2].middle
```

```
    static uint8_t camCycle = 0;
    static uint8_t camState = 0;
    static uint32_t camTime = 0;
    static uint32_t ctLow;
    if (camCycle==1) {
     if (camState == 0) {
      camState = 1;
      camTime = millis();
     } else if (camState == 1) {
      if ( (millis() - camTime) > CAM_TIME_HIGH ) {
       camState = 2;
       camTime = millis();
       if(CAM_TIME_LOW < RC_CHANS) {
        ctLow = constrain((rcData[CAM_TIME_LOW]-1000)/4, 30, 250);
        ctLow *= ctLow;
       } else ctLow = CAM_TIME_LOW;
      }
     } else { //camState ==2
      if (((millis() - camTime) > ctLow) || !rcOptions[BOXCAMTRIG] ) {
       camState = 0;
       camCycle = 0;
      }
     }
    }
    if (rcOptions[BOXCAMTRIG]) camCycle=1;
    servo[2] =(camState==1) ? conf.servoConf[2].max : conf.servoConf[2].min;
    servo[2] = (servo[2]-1500)*SERVODIR(2,1)+1500;
  #endif

/**************************************************************************
***************************************/
  // add midpoint offset, then scale and limit servo outputs - except SERVO8 used
commonly as Moror output
  // don't add offset for camtrig servo (SERVO3)
  #if defined(SERVO)
    for(i=SERVO_START-1; i<SERVO_END; i++) {
     if(i < 2) {
       servo[i] = map(servo[i], 1020,2000, conf.servoConf[i].min,
conf.servoConf[i].max);   // servo travel scaling, only for gimbal servos
     }
    #if defined(HELICOPTER) && (YAWMOTOR)
     if(i != 5) // not limit YawMotor
    #endif
       servo[i] = constrain(servo[i], conf.servoConf[i].min, conf.servoConf[i].max); //
limit the values
    }
    #if defined(A0_A1_PIN_HEX) && (NUMBER_MOTOR == 6) &&
defined(PROMINI)
      servo[3] = servo[0];    // copy CamPitch value to propper output servo for
A0_A1_PIN_HEX
```

```
    servo[4] = servo[1];    // copy CamRoll  value to propper output servo for
A0_A1_PIN_HEX
   #endif
   #if defined(TRI) && defined(MEGA_HW_PWM_SERVOS) && defined(MEGA)
    servo[5] = constrain(servo[5], conf.servoConf[5].min, conf.servoConf[5].max); //
servo[5] is still use by gui for this config (more genereic)
    servo[3] = servo[5];    // copy TRI serwo value to propper output servo for
MEGA_HW_PWM_SERVOS
   #endif
  #endif

  /****************          compensate the Motors values
******************/
  #ifdef VOLTAGEDROP_COMPENSATION
   {
    #if (VBATNOMINAL == 84) // 2S
     #define GOV_R_NUM 24
     static int8_t g[] = {
0,4,8,12,17,21,25,30,34,39,44,49,54,59,65,70,76,81,87,93,99,106,112,119,126 };
    #elif (VBATNOMINAL == 126) // 3S
     #define GOV_R_NUM 36
     static int8_t g[] = {
0,3,5,8,11,14,17,19,22,25,28,31,34,38,41,44,47,51,54,58,61,65,68,72,76,79,83,87,91,
95,99,104,108,112,117,121,126 };
    #elif (VBATNOMINAL == 252) // 6S
     #define GOV_R_NUM 72
     static int8_t g[] = {
0,1,3,4,5,7,8,9,11,12,14,15,17,18,19,21,22,24,25,27,28,30,31,33,34,36,38,39,41,

42,44,46,47,49,51,52,54,56,58,59,61,63,65,66,68,70,72,74,76,78,79,81,83,85,87,89,9
1,93,95,97,99,
        101,104,106,108,110,112,114,117,119,121,123,126       };
    #elif (VBATNOMINAL == 255) // 6S HV
     #define GOV_R_NUM 73
     static int8_t g[] = {
0,1,3,4,5,7,8,9,11,12,14,15,16,18,19,21,22,24,25,26,28,29,31,33,34,36,37,39,40,

42,44,45,47,48,50,52,53,55,57,59,60,62,64,66,67,69,71,73,75,76,78,80,82,84,86,88,9
0,92,94,96,98,
         100,102,104,106,108,111,113,115,117,119,122,124,126       };
    #elif (VBATNOMINAL == 129) // 3S HV
     #define GOV_R_NUM 37
     static int8_t g[] = {
0,3,5,8,11,13,16,19,22,25,28,31,34,37,40,43,46,49,53,56,59,63,66,70,74,77,81,85,
         89,93,96,101,105,109,113,117,122,126       };
    #elif (VBATNOMINAL == 168) // 4S
     #define GOV_R_NUM 48
     static int8_t g[] = {
0,2,4,6,8,10,12,14,17,19,21,23,25,28,30,32,34,37,39,42,44,47,49,52,54,57,59,62,
         65,67,70,73,76,78,81,84,87,90,93,96,99,103,106,109,112,116,119,122,126
```

```
};
    #else
      #error "VOLTAGEDROP_COMPENSATION requires correction values which
fit VBATNOMINAL; not yet defined for your value of VBATNOMINAL"
    #endif
    uint8_t v = constrain( VBATNOMINAL - constrain(analog.vbat,
conf.vbatlevel_crit, VBATNOMINAL), 0, GOV_R_NUM);
    for (i = 0; i < NUMBER_MOTOR; i++) {
      motor[i] += ( ( (int32_t)(motor[i]-1000) * (int32_t)g[v] ) )/ 500;
    }
   }
 #endif
 /*****************          normalize the Motors values
******************/
   maxMotor=motor[0];
   for(i=1; i< NUMBER_MOTOR; i++)
     if (motor[i]>maxMotor) maxMotor=motor[i];
   for(i=0; i< NUMBER_MOTOR; i++) {
     if (maxMotor > MAXTHROTTLE) // this is a way to still have good gyro
corrections if at least one motor reaches its max.
       motor[i] -= maxMotor - MAXTHROTTLE;
     motor[i] = constrain(motor[i], conf.minthrottle, MAXTHROTTLE);
     if ((rcData[THROTTLE] < MINCHECK) && !f.BARO_MODE)
     #ifndef MOTOR_STOP
       motor[i] = conf.minthrottle;
     #else
       motor[i] = MINCOMMAND;
     #endif
     if (!f.ARMED)
       motor[i] = MINCOMMAND;
   }

 /*****************          Powermeter Log
******************/
 #if (LOG_VALUES >= 3) || defined(POWERMETER_SOFT)
 {
   static uint32_t lastRead = currentTime;
   uint16_t amp;
   uint32_t ampsum, ampus; // pseudo ampere * microseconds
   /* true cubic function;
    * when divided by vbat_max=126 (12.6V) for 3 cell battery this gives maximum
value of ~ 500
    * when divided by no_vbat=60 (6V) for 3 cell battery this gives maximum value of
~ 1000
    * */

   static uint16_t amperes[64] =   {   0,  2,  6, 15, 30, 52, 82,123,
                                 175,240,320,415,528,659,811,984,
                                 1181,1402,1648,1923,2226,2559,2924,3322,
                                 3755,4224,4730,5276,5861,6489,7160,7875,
```

```
                        8637 ,9446 ,10304,11213,12173,13187,14256,15381,
                        16564,17805,19108,20472,21900,23392,24951,26578,
                        28274,30041,31879,33792,35779,37843,39984,42205,
                        44507,46890,49358,51910,54549,57276,60093,63000};

   if (analog.vbat > NO_VBAT) { // by all means - must avoid division by zero
     ampsum = 0;
     for (i =0;i<NUMBER_MOTOR;i++) {
       amp = amperes[ ((motor[i] - 1000)>>4) ] / analog.vbat; // range mapped from
[1000:2000] => [0:1000]; then break that up into 64 ranges; lookup amp
       ampus = ( (currentTime-lastRead) * (uint32_t)amp * (uint32_t)conf.pint2ma ) /
PLEVELDIVSOFT;
       #if (LOG_VALUES >= 3)
         pMeter[i]+= ampus; // sum up over time the mapped ESC input
       #endif
       #if defined(POWERMETER_SOFT)
         ampsum += ampus; // total sum over all motors
       #endif
     }
     #if defined(POWERMETER_SOFT)
       pMeter[PMOTOR_SUM]+= ampsum / NUMBER_MOTOR; // total sum over all
motors
     #endif
   }
   lastRead = currentTime;
  }
  #endif
}
```

καρτέλα Output.h

```
#ifndef OUTPUT_H_
#define OUTPUT_H_

extern uint8_t PWM_PIN[8];

void initOutput();
void mixTable();
void writeServos();
void writeMotors();

#endif /* OUTPUT_H_ */
```

καρτέλα Protocol.cpp

```
#include "def.h"
#include "types.h"
#include "EEPROM.h"
#include "LCD.h"
#include "Output.h"
#include "GPS.h"
#include "MultiWii.h"
#include "Serial.h"
#include "Protocol.h"
```

```
#include "RX.h"

/********************************** MultiWii Serial Protocol **********************************************/
// Multiwii Serial Protocol 0
#define MSP_VERSION              0

//to multiwii developpers/committers : do not add new MSP messages without a proper argumentation/agreement on the forum
//range id [50-99] won't be assigned and can therefore be used for any custom multiwii fork without further MSP id conflict

#define MSP_PRIVATE              1     //in+out message      to be used for a generic framework : MSP + function code (LIST/GET/SET) + data. no code yet

#define MSP_IDENT              100   //out message         multitype + multiwii version + protocol version + capability variable
#define MSP_STATUS             101   //out message         cycletime & errors_count & sensor present & box activation & current setting number
#define MSP_RAW_IMU            102   //out message         9 DOF
#define MSP_SERVO              103   //out message         8 servos
#define MSP_MOTOR              104   //out message         8 motors
#define MSP_RC                 105   //out message         8 rc chan and more
#define MSP_RAW_GPS            106   //out message         fix, numsat, lat, lon, alt, speed, ground course
#define MSP_COMP_GPS           107   //out message         distance home, direction home
#define MSP_ATTITUDE           108   //out message         2 angles 1 heading
#define MSP_ALTITUDE           109   //out message         altitude, variometer
#define MSP_ANALOG             110   //out message         vbat, powermetersum, rssi if available on RX
#define MSP_RC_TUNING          111   //out message         rc rate, rc expo, rollpitch rate, yaw rate, dyn throttle PID
#define MSP_PID                112   //out message         P I D coeff (9 are used currently)
#define MSP_BOX                113   //out message         BOX setup (number is dependant of your setup)
#define MSP_MISC               114   //out message         powermeter trig
#define MSP_MOTOR_PINS         115   //out message         which pins are in use for motors & servos, for GUI
#define MSP_BOXNAMES           116   //out message         the aux switch names
#define MSP_PIDNAMES           117   //out message         the PID names
#define MSP_WP                 118   //out message         get a WP, WP# is in the payload, returns (WP#, lat, lon, alt, flags) WP#0-home, WP#16-poshold
#define MSP_BOXIDS             119   //out message         get the permanent IDs associated to BOXes
#define MSP_SERVO_CONF         120   //out message         Servo settings

#define MSP_NAV_STATUS         121   //out message         Returns navigation status
```

```
#define MSP_NAV_CONFIG          122   //out message         Returns navigation parameters

#define MSP_CELLS               130   //out message         FRSKY Battery Cell Voltages

#define MSP_SET_RAW_RC          200   //in message          8 rc chan
#define MSP_SET_RAW_GPS         201   //in message          fix, numsat, lat, lon, alt, speed
#define MSP_SET_PID             202   //in message          P I D coeff (9 are used currently)
#define MSP_SET_BOX             203   //in message          BOX setup (number is dependant of your setup)
#define MSP_SET_RC_TUNING       204   //in message          rc rate, rc expo, rollpitch rate, yaw rate, dyn throttle PID
#define MSP_ACC_CALIBRATION     205   //in message          no param
#define MSP_MAG_CALIBRATION     206   //in message          no param
#define MSP_SET_MISC            207   //in message          powermeter trig + 8 free for future use
#define MSP_RESET_CONF          208   //in message          no param
#define MSP_SET_WP              209   //in message          sets a given WP (WP#,lat, lon, alt, flags)
#define MSP_SELECT_SETTING      210   //in message          Select Setting Number (0-2)
#define MSP_SET_HEAD            211   //in message          define a new heading hold direction
#define MSP_SET_SERVO_CONF      212   //in message          Servo settings
#define MSP_SET_MOTOR           214   //in message          PropBalance function
#define MSP_SET_NAV_CONFIG      215   //in message          Sets nav config parameters - write to the eeprom

#define MSP_SET_ACC_TRIM        239   //in message          set acc angle trim values
#define MSP_ACC_TRIM            240   //out message         get acc angle trim values
#define MSP_BIND                241   //in message          no param

#define MSP_EEPROM_WRITE        250   //in message          no param

#define MSP_DEBUGMSG            253   //out message         debug string buffer
#define MSP_DEBUG               254   //out message         debug1,debug2,debug3,debug4

#ifdef DEBUGMSG
  #define DEBUG_MSG_BUFFER_SIZE 128
  static char debug_buf[DEBUG_MSG_BUFFER_SIZE];
  static uint8_t head_debug;
  static uint8_t tail_debug;
  static uint8_t debugmsg_available();
  static void debugmsg_serialize(uint8_t l);
#endif
```

```
static uint8_t CURRENTPORT=0;

#define INBUF_SIZE 64
static uint8_t inBuf[INBUF_SIZE][UART_NUMBER];
static uint8_t checksum[UART_NUMBER];
static uint8_t indRX[UART_NUMBER];
static uint8_t cmdMSP[UART_NUMBER];

void evaluateOtherData(uint8_t sr);
void evaluateCommand(uint8_t c);

static uint8_t read8()  {
  return inBuf[indRX[CURRENTPORT]++][CURRENTPORT]&0xff;
}
static uint16_t read16() {
  uint16_t t = read8();
  t+= (uint16_t)read8()<<8;
  return t;
}
static uint32_t read32() {
  uint32_t t = read16();
  t+= (uint32_t)read16()<<16;
  return t;
}

static void serialize8(uint8_t a) {
  SerialSerialize(CURRENTPORT,a);
  checksum[CURRENTPORT] ^= a;
}
static void serialize16(int16_t a) {
  serialize8((a   ) & 0xFF);
  serialize8((a>>8) & 0xFF);
}
static void serialize32(uint32_t a) {
  serialize8((a    ) & 0xFF);
  serialize8((a>> 8) & 0xFF);
  serialize8((a>>16) & 0xFF);
  serialize8((a>>24) & 0xFF);
}

static void headSerialResponse(uint8_t err, uint8_t s) {
  serialize8('$');
  serialize8('M');
  serialize8(err ? '!' : '>');
  checksum[CURRENTPORT] = 0; // start calculating a new checksum
  serialize8(s);
  serialize8(cmdMSP[CURRENTPORT]);
}
```

```
static void headSerialReply(uint8_t s) {
  headSerialResponse(0, s);
}

static void headSerialError() {
  headSerialResponse(1,0);
}

static void tailSerialReply() {
  serialize8(checksum[CURRENTPORT]);UartSendData(CURRENTPORT);
}

static void serializeNames(PGM_P s) {
  headSerialReply(strlen_P(s));
  for (PGM_P c = s; pgm_read_byte(c); c++)
    serialize8(pgm_read_byte(c));
  tailSerialReply();
}

static void __attribute__ ((noinline)) s_struct_w(uint8_t *cb,uint8_t siz) {
  while(siz--) *cb++ = read8();
}

static void s_struct_partial(uint8_t *cb,uint8_t siz) {
  while(siz--) serialize8(*cb++);
}

static void s_struct(uint8_t *cb,uint8_t siz) {
  headSerialReply(siz);
  s_struct_partial(cb,siz);
  tailSerialReply();
}

static void mspAck() {
  headSerialReply(0);tailSerialReply();
}

enum MSP_protocol_bytes {
  IDLE,
  HEADER_START,
  HEADER_M,
  HEADER_ARROW,
  HEADER_SIZE,
  HEADER_CMD
};

void serialCom() {
  uint8_t c,cc,port,state,bytesTXBuff;
  static uint8_t offset[UART_NUMBER];
  static uint8_t dataSize[UART_NUMBER];
```

```
  static uint8_t c_state[UART_NUMBER];
  uint32_t timeMax; // limit max time in this function in case of GPS

  timeMax = micros();
  for(port=0;port<UART_NUMBER;port++) {
    CURRENTPORT=port;
    #define RX_COND
    #if defined(SERIAL_RX) && (UART_NUMBER > 1)
      #define RX_COND && (RX_SERIAL_PORT != port)
    #endif
    cc = SerialAvailable(port);
    while (cc-- RX_COND) {
      bytesTXBuff = SerialUsedTXBuff(port); // indicates the number of occupied bytes
in TX buffer
      if (bytesTXBuff > TX_BUFFER_SIZE - 50 ) return; // ensure there is enough free
TX buffer to go further (50 bytes margin)
      c = SerialRead(port);
      #ifdef SUPPRESS_ALL_SERIAL_MSP
        evaluateOtherData(c); // no MSP handling, so go directly
      #else //SUPPRESS_ALL_SERIAL_MSP
        state = c_state[port];
        // regular data handling to detect and handle MSP and other data
        if (state == IDLE) {
          if (c=='$') state = HEADER_START;
          else evaluateOtherData(c); // evaluate all other incoming serial data
        } else if (state == HEADER_START) {
          state = (c=='M') ? HEADER_M : IDLE;
        } else if (state == HEADER_M) {
          state = (c=='<') ? HEADER_ARROW : IDLE;
        } else if (state == HEADER_ARROW) {
          if (c > INBUF_SIZE) {  // now we are expecting the payload size
            state = IDLE;
            continue;
          }
          dataSize[port] = c;
          checksum[port] = c;
          offset[port] = 0;
          indRX[port] = 0;
          state = HEADER_SIZE;  // the command is to follow
        } else if (state == HEADER_SIZE) {
          cmdMSP[port] = c;
          checksum[port] ^= c;
          state = HEADER_CMD;
        } else if (state == HEADER_CMD) {
          if (offset[port] < dataSize[port]) {
            checksum[port] ^= c;
            inBuf[offset[port]++][port] = c;
          } else {
            if (checksum[port] == c) // compare calculated and transferred checksum
              evaluateCommand(cmdMSP[port]); // we got a valid packet, evaluate it
```

```
          state = IDLE;
          cc = 0; // no more than one MSP per port and per cycle
         }
        }
        c_state[port] = state;

        // SERIAL: try to detect a new nav frame based on the current received buffer
        #if defined(GPS_SERIAL)
        if (GPS_SERIAL == port) {
          static uint32_t GPS_last_frame_seen; //Last gps frame seen at this time, used to
detect stalled gps communication
          if (GPS_newFrame(c)) {
            //We had a valid GPS data frame, so signal task scheduler to switch to
compute
            if (GPS_update == 1) GPS_update = 0; else GPS_update = 1; //Blink GPS
update
            GPS_last_frame_seen = timeMax;
            GPS_Frame = 1;
          }

          // Check for stalled GPS, if no frames seen for 1.2sec then consider it LOST
          if ((timeMax - GPS_last_frame_seen) > 1200000) {
            //No update since 1200ms clear fix...
            f.GPS_FIX = 0;
            GPS_numSat = 0;
          }
        }
        if (micros()-timeMax>250) return;  // Limit the maximum execution time of
serial decoding to avoid time spike
        #endif
      #endif // SUPPRESS_ALL_SERIAL_MSP
    } // while
  } // for
}

void evaluateCommand(uint8_t c) {
  uint32_t tmp=0;

  switch(c) {
    // adding this message as a comment will return an error status for MSP_PRIVATE
(end of switch), allowing third party tools to distinguish the implementation of this
message
    //case MSP_PRIVATE:
    //  headSerialError();tailSerialReply(); // we don't have any custom msp currently,
so tell the gui we do not use that
    //  break;
    case MSP_SET_RAW_RC:
      s_struct_w((uint8_t*)&rcSerial,16);
      rcSerialCount = 50; // 1s transition
      break;
```

```
case MSP_SET_PID:
  mspAck();
  s_struct_w((uint8_t*)&conf.pid[0].P8,3*PIDITEMS);
  break;
case MSP_SET_BOX:
  mspAck();
  #if EXTAUX
    s_struct_w((uint8_t*)&conf.activate[0],CHECKBOXITEMS*4);
  #else
    s_struct_w((uint8_t*)&conf.activate[0],CHECKBOXITEMS*2);
  #endif
  break;
case MSP_SET_RC_TUNING:
  mspAck();
  s_struct_w((uint8_t*)&conf.rcRate8,7);
  break;
#if !defined(DISABLE_SETTINGS_TAB)
case MSP_SET_MISC:
  struct {
    uint16_t a,b,c,d,e,f;
    uint32_t g;
    uint16_t h;
    uint8_t  i,j,k,l;
  } set_misc;
  mspAck();
  s_struct_w((uint8_t*)&set_misc,22);
  #if defined(POWERMETER)
    conf.powerTrigger1 = set_misc.a / PLEVELSCALE;
  #endif
  conf.minthrottle = set_misc.b;
  #ifdef FAILSAFE
    conf.failsafe_throttle = set_misc.e;
  #endif
  #if MAG
    conf.mag_declination = set_misc.h;
  #endif
  #if defined(VBAT)
    conf.vbatscale        = set_misc.i;
    conf.vbatlevel_warn1  = set_misc.j;
    conf.vbatlevel_warn2  = set_misc.k;
    conf.vbatlevel_crit   = set_misc.l;
  #endif
  break;
case MSP_MISC:
  struct {
    uint16_t a,b,c,d,e,f;
    uint32_t g;
    uint16_t h;
    uint8_t  i,j,k,l;
  } misc;
```

```
      misc.a = intPowerTrigger1;
      misc.b = conf.minthrottle;
      misc.c = MAXTHROTTLE;
      misc.d = MINCOMMAND;
      #ifdef FAILSAFE
        misc.e = conf.failsafe_throttle;
      #else
        misc.e = 0;
      #endif
      #ifdef LOG_PERMANENT
        misc.f = plog.arm;
        misc.g = plog.lifetime + (plog.armed_time / 1000000); // <- computation must be
moved outside from serial
      #else
        misc.f = 0; misc.g =0;
      #endif
      #if MAG
        misc.h = conf.mag_declination;
      #else
        misc.h = 0;
      #endif
      #ifdef VBAT
        misc.i = conf.vbatscale;
        misc.j = conf.vbatlevel_warn1;
        misc.k = conf.vbatlevel_warn2;
        misc.l = conf.vbatlevel_crit;
      #else
        misc.i = 0;misc.j = 0;misc.k = 0;misc.l = 0;
      #endif
      s_struct((uint8_t*)&misc,22);
      break;
    #endif
    #if defined (DYNBALANCE)
    case MSP_SET_MOTOR:
      mspAck();
      s_struct_w((uint8_t*)&motor,16);
    break;
    #endif
    #ifdef MULTIPLE_CONFIGURATION_PROFILES
    case MSP_SELECT_SETTING:
      if(!f.ARMED) {
        global_conf.currentSet = read8();
        if(global_conf.currentSet>2) global_conf.currentSet = 0;
        writeGlobalSet(0);
        readEEPROM();
      }
      mspAck();
      break;
    #endif
    case MSP_SET_HEAD:
```

```
      mspAck();
      s_struct_w((uint8_t*)&magHold,2);
      break;
    case MSP_IDENT:
      struct {
       uint8_t v,t,msp_v;
       uint32_t cap;
      } id;
      id.v     = VERSION;
      id.t     = MULTITYPE;
      id.msp_v = MSP_VERSION;
      id.cap   =
(0+BIND_CAPABLE)|DYNBAL<<2|FLAP<<3|NAVCAP<<4|EXTAUX<<5|((uint32
_t)NAVI_VERSION<<28); //Navi version is stored in the upper four bits;
      s_struct((uint8_t*)&id,7);
      break;
    case MSP_STATUS:
      struct {
       uint16_t cycleTime,i2c_errors_count,sensor;
       uint32_t flag;
       uint8_t set;
      } st;
      st.cycleTime        = cycleTime;
      st.i2c_errors_count = i2c_errors_count;
      st.sensor           = ACC|BARO<<1|MAG<<2|GPS<<3|SONAR<<4;
      #if ACC
       if(f.ANGLE_MODE)   tmp |= 1<<BOXANGLE;
       if(f.HORIZON_MODE) tmp |= 1<<BOXHORIZON;
      #endif
      #if BARO && (!defined(SUPPRESS_BARO_ALTHOLD))
       if(f.BARO_MODE) tmp |= 1<<BOXBARO;
      #endif
      if(f.MAG_MODE) tmp |= 1<<BOXMAG;
      #if !defined(FIXEDWING)
       #if defined(HEADFREE)
        if(f.HEADFREE_MODE)       tmp |= 1<<BOXHEADFREE;
        if(rcOptions[BOXHEADADJ]) tmp |= 1<<BOXHEADADJ;
       #endif
      #endif
      #if defined(SERVO_TILT) || defined(GIMBAL)|| defined(SERVO_MIX_TILT)
       if(rcOptions[BOXCAMSTAB]) tmp |= 1<<BOXCAMSTAB;
      #endif
      #if defined(CAMTRIG)
       if(rcOptions[BOXCAMTRIG]) tmp |= 1<<BOXCAMTRIG;
      #endif
      #if GPS
       switch (f.GPS_mode) {
        case GPS_MODE_HOLD:
         tmp |= 1<<BOXGPSHOLD;
         break;
```

```
          case GPS_MODE_RTH:
            tmp |= 1<<BOXGPSHOME;
            break;
          case GPS_MODE_NAV:
            tmp |= 1<<BOXGPSNAV;
            break;
        }
      #endif
      #if defined(FIXEDWING) || defined(HELICOPTER)
        if(f.PASSTHRU_MODE) tmp |= 1<<BOXPASSTHRU;
      #endif
      #if defined(BUZZER)
        if(rcOptions[BOXBEEPERON]) tmp |= 1<<BOXBEEPERON;
      #endif
      #if defined(LED_FLASHER)
        if(rcOptions[BOXLEDMAX]) tmp |= 1<<BOXLEDMAX;
        if(rcOptions[BOXLEDLOW]) tmp |= 1<<BOXLEDLOW;
      #endif
      #if defined(LANDING_LIGHTS_DDR)
        if(rcOptions[BOXLLIGHTS]) tmp |= 1<<BOXLLIGHTS;
      #endif
      #if defined(VARIOMETER)
        if(rcOptions[BOXVARIO]) tmp |= 1<<BOXVARIO;
      #endif
      #if defined(INFLIGHT_ACC_CALIBRATION)
        if(rcOptions[BOXCALIB]) tmp |= 1<<BOXCALIB;
      #endif
      #if defined(GOVERNOR_P)
        if(rcOptions[BOXGOV]) tmp |= 1<<BOXGOV;
      #endif
      #if defined(OSD_SWITCH)
        if(rcOptions[BOXOSD]) tmp |= 1<<BOXOSD;
      #endif
      if(f.ARMED) tmp |= 1<<BOXARM;
      st.flag            = tmp;
      st.set             = global_conf.currentSet;
      s_struct((uint8_t*)&st,11);
      break;
    case MSP_RAW_IMU:
      #if defined(DYNBALANCE)
        for(uint8_t axis=0;axis<3;axis++)
{imu.gyroData[axis]=imu.gyroADC[axis];imu.accSmooth[axis]=
imu.accADC[axis];} // Send the unfiltered Gyro & Acc values to gui.
      #endif
      s_struct((uint8_t*)&imu,18);
      break;
    case MSP_SERVO:
      s_struct((uint8_t*)&servo,16);
      break;
    case MSP_SERVO_CONF:
```

```
      s_struct((uint8_t*)&conf.servoConf[0].min,56); // struct servo_conf_ is 7 bytes
length: min:2 / max:2 / middle:2 / rate:1    ----     8 servo =>  8x7 = 56
      break;
    case MSP_SET_SERVO_CONF:
      mspAck();
      s_struct_w((uint8_t*)&conf.servoConf[0].min,56);
      break;
    case MSP_MOTOR:
      s_struct((uint8_t*)&motor,16);
      break;
    case MSP_ACC_TRIM:
      headSerialReply(4);
      s_struct_partial((uint8_t*)&conf.angleTrim[PITCH],2);
      s_struct_partial((uint8_t*)&conf.angleTrim[ROLL],2);
      tailSerialReply();
      break;
    case MSP_SET_ACC_TRIM:
      mspAck();
      s_struct_w((uint8_t*)&conf.angleTrim[PITCH],2);
      s_struct_w((uint8_t*)&conf.angleTrim[ROLL],2);
      break;
    case MSP_RC:
      s_struct((uint8_t*)&rcData,RC_CHANS*2);
      break;
    #if GPS
    case MSP_SET_RAW_GPS:
      struct {
        uint8_t a,b;
        int32_t c,d;
        int16_t e;
        uint16_t f;
      } set_set_raw_gps;
      mspAck();
      s_struct_w((uint8_t*)&set_set_raw_gps,14);
      f.GPS_FIX = set_set_raw_gps.a;
      GPS_numSat = set_set_raw_gps.b;
      GPS_coord[LAT] = set_set_raw_gps.c;
      GPS_coord[LON] = set_set_raw_gps.d;
      GPS_altitude = set_set_raw_gps.e;
      GPS_speed = set_set_raw_gps.f;
      GPS_update |= 2;              // New data signalisation to GPS functions
      break;
    case MSP_RAW_GPS:
      struct {
        uint8_t a,b;
        int32_t c,d;
        int16_t e;
        uint16_t f,g;
      } msp_raw_gps;
      msp_raw_gps.a     = f.GPS_FIX;
```

```
  msp_raw_gps.b     = GPS_numSat;
  msp_raw_gps.c     = GPS_coord[LAT];
  msp_raw_gps.d     = GPS_coord[LON];
  msp_raw_gps.e     = GPS_altitude;
  msp_raw_gps.f     = GPS_speed;
  msp_raw_gps.g     = GPS_ground_course;
  s_struct((uint8_t*)&msp_raw_gps,16);
  break;
 case MSP_COMP_GPS:
  struct {
   uint16_t a;
   int16_t b;
   uint8_t c;
  } msp_comp_gps;
  msp_comp_gps.a     = GPS_distanceToHome;
  msp_comp_gps.b     = GPS_directionToHome;
  msp_comp_gps.c     = GPS_update & 1;
  s_struct((uint8_t*)&msp_comp_gps,5);
  break;
 #if defined(USE_MSP_WP)
 case MSP_SET_NAV_CONFIG:
  mspAck();
  s_struct_w((uint8_t*)&GPS_conf,sizeof(GPS_conf));
  break;
 case MSP_NAV_CONFIG:
  s_struct((uint8_t*)&GPS_conf,sizeof(GPS_conf));
  break;
 case MSP_NAV_STATUS: // to move to struct transmission
  headSerialReply(7);
  serialize8(f.GPS_mode);
  serialize8(NAV_state);
  serialize8(mission_step.action);
  serialize8(mission_step.number);
  serialize8(NAV_error);
  serialize16( (int16_t)(target_bearing/100));
  //serialize16(magHold);
  tailSerialReply();
  break;
 case MSP_WP: // to move to struct transmission
  {
  uint8_t wp_no;
  uint8_t flag;
  bool    success;

  wp_no = read8(); //get the wp number
  headSerialReply(21);
  if (wp_no == 0) { //Get HOME coordinates
   serialize8(wp_no);
   serialize8(mission_step.action);
   serialize32(GPS_home[LAT]);
```

```
      serialize32(GPS_home[LON]);
      flag = MISSION_FLAG_HOME;
     }
     if (wp_no == 255) { //Get poshold coordinates
      serialize8(wp_no);
      serialize8(mission_step.action);
      serialize32(GPS_hold[LAT]);
      serialize32(GPS_hold[LON]);
      flag = MISSION_FLAG_HOLD;
     }
     if ((wp_no>0) && (wp_no<255)) {
      if (NAV_state == NAV_STATE_NONE) {
       success = recallWP(wp_no);
       serialize8(wp_no);
       serialize8(mission_step.action);
       serialize32(mission_step.pos[LAT]);
       serialize32(mission_step.pos[LON]);
       if (success == true) flag = mission_step.flag;
       else flag = MISSION_FLAG_CRC_ERROR; //CRC error
      } else {
       serialize8(wp_no);
       serialize8(0);
       serialize32(GPS_home[LAT]);
       serialize32(GPS_home[LON]);
       flag = MISSION_FLAG_NAV_IN_PROG;
      }
     }
     serialize32(mission_step.altitude);
     serialize16(mission_step.parameter1);
     serialize16(mission_step.parameter2);
     serialize16(mission_step.parameter3);
     serialize8(flag);
     tailSerialReply();
     }
     break;
    case MSP_SET_WP: // to move to struct transmission
     {
     uint8_t wp_no = read8(); //Get the step number

     if (NAV_state == NAV_STATE_HOLD_INFINIT && wp_no == 255) { //Special
case - during stable poshold we allow change the hold position
      mission_step.number = wp_no;
      mission_step.action = MISSION_HOLD_UNLIM;
      uint8_t temp = read8();
      mission_step.pos[LAT] =  read32();
      mission_step.pos[LON] =  read32();
      mission_step.altitude =  read32();
      mission_step.parameter1 = read16();
      mission_step.parameter2 = read16();
      mission_step.parameter3 = read16();
```

```
      mission_step.flag     =  read8();
      if (mission_step.altitude != 0) set_new_altitude(mission_step.altitude);
      GPS_set_next_wp(&mission_step.pos[LAT], &mission_step.pos[LON],
&GPS_coord[LAT], &GPS_coord[LON]);
      if ((wp_distance/100) >= GPS_conf.safe_wp_distance) NAV_state =
NAV_STATE_NONE;
      else NAV_state = NAV_STATE_WP_ENROUTE;
      break;
     }
    if (NAV_state == NAV_STATE_NONE) { // The Nav state is not zero, so
navigation is in progress, silently ignore SET_WP command)
      mission_step.number     =  wp_no;
      mission_step.action     =  read8();
      mission_step.pos[LAT]   =  read32();
      mission_step.pos[LON]   =  read32();
      mission_step.altitude   =  read32();
      mission_step.parameter1 = read16();
      mission_step.parameter2 = read16();
      mission_step.parameter3 = read16();
      mission_step.flag       =  read8();
      //It's not sure, that we want to do poshold change via mission planner so perhaps
the next if is deletable
      /*
      if (mission_step.number == 255) //Set up new hold position via mission planner,
It must set the action to MISSION_HOLD_INFINIT
      {
      if (mission_step.altitude !=0) set_new_altitude(mission_step.altitude); //Set the
altitude
      GPS_set_next_wp(&mission_step.pos[LAT], &mission_step.pos[LON],
&GPS_coord[LAT], &GPS_coord[LON]);
      NAV_state = NAV_STATE_WP_ENROUTE; //Go to that position, then it will
switch to poshold unlimited when reached
      }
      */
      if (mission_step.number == 0) { //Set new Home position
        GPS_home[LAT] = mission_step.pos[LAT];
        GPS_home[LON] = mission_step.pos[LON];
      }
      if (mission_step.number >0 && mission_step.number<255)                    //Not
home and not poshold, we are free to store it in the eprom
        if (mission_step.number <= getMaxWPNumber())                            // Do not
thrash the EEPROM with invalid wp number
          storeWP();
      mspAck();
    }
    }
    break;
  #endif //USE_MSP_WP
  #endif //GPS
  case MSP_ATTITUDE:
```

```
  s_struct((uint8_t*)&att,6);
  break;
case MSP_ALTITUDE:
  s_struct((uint8_t*)&alt,6);
  break;
case MSP_ANALOG:
  s_struct((uint8_t*)&analog,7);
  break;
case MSP_RC_TUNING:
  s_struct((uint8_t*)&conf.rcRate8,7);
  break;
case MSP_PID:
  s_struct((uint8_t*)&conf.pid[0].P8,3*PIDITEMS);
  break;
case MSP_PIDNAMES:
  serializeNames(pidnames);
  break;
case MSP_BOX:
  #if EXTAUX
    s_struct((uint8_t*)&conf.activate[0],4*CHECKBOXITEMS);
  #else
    s_struct((uint8_t*)&conf.activate[0],2*CHECKBOXITEMS);
  #endif
  break;
case MSP_BOXNAMES:
  serializeNames(boxnames);
  break;
case MSP_BOXIDS:
  headSerialReply(CHECKBOXITEMS);
  for(uint8_t i=0;i<CHECKBOXITEMS;i++)
    serialize8(pgm_read_byte(&(boxids[i])));
  tailSerialReply();
  break;
case MSP_MOTOR_PINS:
  s_struct((uint8_t*)&PWM_PIN,8);
  break;
case MSP_RESET_CONF:
  if(!f.ARMED) LoadDefaults();
  mspAck();
  break;
case MSP_ACC_CALIBRATION:
  if(!f.ARMED) calibratingA=512;
  mspAck();
  break;
#if MAG
  case MSP_MAG_CALIBRATION:
  if(!f.ARMED) f.CALIBRATE_MAG = 1;
  mspAck();
  break;
#endif
```

```
    #if defined(SPEK_BIND)
    case MSP_BIND:
      spekBind();
      mspAck();
      break;
    #endif
    case MSP_EEPROM_WRITE:
      writeParams(0);
      mspAck();
      break;
    case MSP_DEBUG:
      s_struct((uint8_t*)&debug,8);
      break;
    #ifdef DEBUGMSG
    case MSP_DEBUGMSG:
      {
      uint8_t size = debugmsg_available();
      if (size > 16) size = 16;
      headSerialReply(size);
      debugmsg_serialize(size);
      tailSerialReply();
      }
      break;
    #endif
    default:  // we do not know how to handle the (valid) message, indicate error MSP
$M!
      headSerialError();tailSerialReply();
      break;
  }
}

// evaluate all other incoming serial data
void evaluateOtherData(uint8_t sr) {
  #ifndef SUPPRESS_OTHER_SERIAL_COMMANDS
    #if GPS
      #if !defined(I2C_GPS)
        // on the GPS port, we must avoid interpreting incoming values for other
commands because there is no
        // protocol protection as is with MSP commands
        // doing so with single chars would be prone to error.
        if (CURRENTPORT == GPS_SERIAL) return;
      #endif
    #endif
    switch (sr) {
    // Note: we may receive weird characters here which could trigger unwanted
features during flight.
    //       this could lead to a crash easily.
    //       Please use if (!f.ARMED) where neccessary
      #ifdef LCD_CONF
        case 's':
```

```
      case 'S':
        if (!f.ARMED) configurationLoop();
        break;
    #endif
    #ifdef LOG_PERMANENT_SHOW_AT_L
      case 'L':
        if (!f.ARMED) dumpPLog(1);
        break;
      #endif
      #if defined(LCD_TELEMETRY) && defined(LCD_TEXTSTAR)
      case 'A': // button A press
        toggle_telemetry(1);
        break;
      case 'B': // button B press
        toggle_telemetry(2);
        break;
      case 'C': // button C press
        toggle_telemetry(3);
        break;
      case 'D': // button D press
        toggle_telemetry(4);
        break;
      case 'a': // button A release
      case 'b': // button B release
      case 'c': // button C release
      case 'd': // button D release
        break;
    #endif
    #ifdef LCD_TELEMETRY
      case '0':
      case '1':
      case '2':
      case '3':
      case '4':
      case '5':
      case '6':
      case '7':
      case '8':
      case '9':
    #ifndef SUPPRESS_TELEMETRY_PAGE_R
      case 'R':
    #endif
    #if defined(DEBUG) || defined(DEBUG_FREE)
      case 'F':
    #endif
        toggle_telemetry(sr);
        break;
    #endif // LCD_TELEMETRY
    }
  #endif // SUPPRESS_OTHER_SERIAL_COMMANDS
```

```
}

void SerialWrite16(uint8_t port, int16_t val)
{
 CURRENTPORT=port;
 serialize16(val);UartSendData(port);
}


#ifdef DEBUGMSG
void debugmsg_append_str(const char *str) {
 while(*str) {
  debug_buf[head_debug++] = *str++;
  if (head_debug == DEBUG_MSG_BUFFER_SIZE) {
   head_debug = 0;
  }
 }
}

static uint8_t debugmsg_available() {
 if (head_debug >= tail_debug) {
  return head_debug-tail_debug;
 } else {
  return head_debug + (DEBUG_MSG_BUFFER_SIZE-tail_debug);
 }
}

static void debugmsg_serialize(uint8_t l) {
 for (uint8_t i=0; i<l; i++) {
  if (head_debug != tail_debug) {
   serialize8(debug_buf[tail_debug++]);
   if (tail_debug == DEBUG_MSG_BUFFER_SIZE) {
    tail_debug = 0;
   }
  } else {
   serialize8('\0');
  }
 }
}
#else
void debugmsg_append_str(const char *str) {};
#endif
```

καρτέλα Protocol.h

```
#ifndef PROTOCOL_H_
#define PROTOCOL_H_

void serialCom();
void debugmsg_append_str(const char *str);

#endif /* PROTOCOL_H_ */
```

καρτέλα RX.cpp

```
#include "Arduino.h"
#include "config.h"
#include "def.h"
#include "types.h"
#include "Serial.h"
#include "Protocol.h"
#include "MultiWii.h"
#include "Alarms.h"

/**********************************************************************
******************/
/***************        Global RX related variables
*********************/
/**********************************************************************
******************/

#if defined(SPEKTRUM)
  #include <wiring.c>  //Auto-included by the Arduino core... but we need it sooner.
#endif

//RAW RC values will be store here
#if defined(SBUS)
  volatile uint16_t rcValue[RC_CHANS] = {1500, 1500, 1500, 1500, 1500, 1500,
1500, 1500, 1500, 1500, 1500, 1500, 1500, 1500, 1500, 1500, 1500, 1500}; // interval
[1000;2000]
#elif defined(SPEKTRUM) || defined(SERIAL_SUM_PPM)
  volatile uint16_t rcValue[RC_CHANS] = {1502, 1502, 1502, 1502, 1502, 1502,
1502, 1502, 1502, 1502, 1502, 1502}; // interval [1000;2000]
#else
  volatile uint16_t rcValue[RC_CHANS] = {1502, 1502, 1502, 1502, 1502, 1502,
1502, 1502}; // interval [1000;2000]
#endif

#if defined(SERIAL_SUM_PPM) //Channel order for PPM SUM RX Configs
  static uint8_t rcChannel[RC_CHANS] = {SERIAL_SUM_PPM};
#elif defined(SBUS) //Channel order for SBUS RX Configs
  // for 16 + 2 Channels SBUS. The 10 extra channels 8->17 are not used by MultiWii,
but it should be easy to integrate them.
  static uint8_t rcChannel[RC_CHANS] = {SBUS};
#elif defined(SUMD)
  static uint8_t rcChannel[RC_CHANS] =
{PITCH,YAW,THROTTLE,ROLL,AUX1,AUX2,AUX3,AUX4};
#elif defined(SPEKTRUM)
  static uint8_t rcChannel[RC_CHANS] =
{PITCH,YAW,THROTTLE,ROLL,AUX1,AUX2,AUX3,AUX4,8,9,10,11};
#else // Standard Channel order
  static uint8_t rcChannel[RC_CHANS]  = {ROLLPIN, PITCHPIN, YAWPIN,
THROTTLEPIN, AUX1PIN,AUX2PIN,AUX3PIN,AUX4PIN};
  static uint8_t PCInt_RX_Pins[PCINT_PIN_COUNT] = {PCINT_RX_BITS}; // if
```

```
this slowes the PCINT readings we can switch to a define for each pcint bit
#endif

void rxInt(void);

/************************************************************************
*****************/
/***************                RX Pin Setup                ********************/
/************************************************************************
*****************/
void configureReceiver() {
  /******************    Configure each rc pin for PCINT
***************************/
  #if defined(STANDARD_RX)
    #if defined(MEGA)
      DDRK = 0;  // defined PORTK as a digital port ([A8-A15] are consired as digital
PINs and not analogical)
    #endif
    // PCINT activation
    for(uint8_t i = 0; i < PCINT_PIN_COUNT; i++){ // i think a for loop is ok for the
init.
      PCINT_RX_PORT |= PCInt_RX_Pins[i];
      PCINT_RX_MASK |= PCInt_RX_Pins[i];
    }
    PCICR = PCIR_PORT_BIT;

    /*************    atmega328P's Specific Aux2 Pin Setup
**********************/
    #if defined(PROMINI)
     #if defined(RCAUXPIN)
       PCICR  |= (1 << 0) ; // PCINT activated also for PINS [D8-D13] on port B
       #if defined(RCAUXPIN8)
         PCMSK0 = (1 << 0);
       #endif
       #if defined(RCAUXPIN12)
         PCMSK0 = (1 << 4);
       #endif
     #endif
    #endif

    /***************   atmega32u4's Specific RX Pin Setup
**********************/
    #if defined(PROMICRO)
      //Trottle on pin 7
      DDRE &= ~(1 << 6); // pin 7 to input
      PORTE |= (1 << 6); // enable pullups
      EICRB |= (1 << ISC60);
      EIMSK |= (1 << INT6); // enable interuppt
      // Aux2 pin on PBO (D17/RXLED)
      #if defined(RCAUX2PIND17)
```

```
      DDRB &= ~(1 << 0); // set D17 to input
    #endif
    // Aux2 pin on PD2 (RX0)
    #if defined(RCAUX2PINRXO)
      DDRD &= ~(1 << 2); // RX to input
      PORTD |= (1 << 2); // enable pullups
      EICRA |= (1 << ISC20);
      EIMSK |= (1 << INT2); // enable interuppt
    #endif
  #endif

  /************************  Special RX Setup
*******************************/
  #endif
  #if defined(SERIAL_SUM_PPM)
    PPM_PIN_INTERRUPT;
  #endif
  #if defined (SPEKTRUM) || defined(SUMD)
    SerialOpen(RX_SERIAL_PORT,115200);
  #endif
  #if defined(SBUS)
    SerialOpen(RX_SERIAL_PORT,100000);
    switch (RX_SERIAL_PORT) { //parity
      #if defined(MEGA)
        case 0: UCSR0C |= (1<<UPM01)|(1<<USBS0); break;
        case 1: UCSR1C |= (1<<UPM11)|(1<<USBS1); break;
        case 2: UCSR2C |= (1<<UPM21)|(1<<USBS2); break;
        case 3: UCSR3C |= (1<<UPM31)|(1<<USBS3); break;
      #endif
    }
  #endif
}

/*******************************************************************************
******************/
/***************          Standard RX Pins reading
********************/
/*******************************************************************************
******************/
#if defined(STANDARD_RX)

#if defined(FAILSAFE) && !defined(PROMICRO)
  // predefined PC pin block (thanks to lianj)  - Version with failsafe
  #define RX_PIN_CHECK(pin_pos, rc_value_pos)                     \
    if (mask & PCInt_RX_Pins[pin_pos]) {                          \
      if (!(pin & PCInt_RX_Pins[pin_pos])) {                      \
        dTime = cTime-edgeTime[pin_pos];                          \
        if (900<dTime && dTime<2200) {                            \
          rcValue[rc_value_pos] = dTime;                          \
          if((rc_value_pos==THROTTLEPIN || rc_value_pos==YAWPIN ||   \
```

```
          rc_value_pos==PITCHPIN || rc_value_pos==ROLLPIN)       \
          && dTime>FAILSAFE_DETECT_TRESHOLD)                     \
            GoodPulses |= (1<<rc_value_pos);                     \
      }                                                          \
    } else edgeTime[pin_pos] = cTime;                            \
  }
#else
  // predefined PC pin block (thanks to lianj)  - Version without failsafe
  #define RX_PIN_CHECK(pin_pos, rc_value_pos)                    \
    if (mask & PCInt_RX_Pins[pin_pos]) {                         \
      if (!(pin & PCInt_RX_Pins[pin_pos])) {                     \
        dTime = cTime-edgeTime[pin_pos];                         \
        if (900<dTime && dTime<2200) {                           \
          rcValue[rc_value_pos] = dTime;                         \
        }                                                        \
      } else edgeTime[pin_pos] = cTime;                          \
    }
#endif

  // port change Interrupt
  ISR(RX_PC_INTERRUPT) { //this ISR is common to every receiver channel, it is call
everytime a change state occurs on a RX input pin
    uint8_t mask;
    uint8_t pin;
    uint16_t cTime,dTime;
    static uint16_t edgeTime[8];
    static uint8_t PCintLast;
  #if defined(FAILSAFE) && !defined(PROMICRO)
    static uint8_t GoodPulses;
  #endif

    pin = RX_PCINT_PIN_PORT; // RX_PCINT_PIN_PORT indicates the state of
each PIN for the arduino port dealing with Ports digital pins

    mask = pin ^ PCintLast;   // doing a ^ between the current interruption and the last
one indicates wich pin changed
    cTime = micros();         // micros() return a uint32_t, but it is not usefull to keep the
whole bits => we keep only 16 bits
    sei();                    // re enable other interrupts at this point, the rest of this interrupt
is not so time critical and can be interrupted safely
    PCintLast = pin;          // we memorize the current state of all PINs [D0-D7]

    #if (PCINT_PIN_COUNT > 0)
      RX_PIN_CHECK(0,2);
    #endif
    #if (PCINT_PIN_COUNT > 1)
      RX_PIN_CHECK(1,4);
    #endif
    #if (PCINT_PIN_COUNT > 2)
      RX_PIN_CHECK(2,5);
```

```
  #endif
  #if (PCINT_PIN_COUNT > 3)
    RX_PIN_CHECK(3,6);
  #endif
  #if (PCINT_PIN_COUNT > 4)
    RX_PIN_CHECK(4,7);
  #endif
  #if (PCINT_PIN_COUNT > 5)
    RX_PIN_CHECK(5,0);
  #endif
  #if (PCINT_PIN_COUNT > 6)
    RX_PIN_CHECK(6,1);
  #endif
  #if (PCINT_PIN_COUNT > 7)
    RX_PIN_CHECK(7,3);
  #endif

  #if defined(FAILSAFE) && !defined(PROMICRO)
    if
(GoodPulses==(1<<THROTTLEPIN)+(1<<YAWPIN)+(1<<ROLLPIN)+(1<<PITC
HPIN)) {  // If all main four chanells have good pulses, clear FailSafe counter
      GoodPulses = 0;
      if(failsafeCnt > 20) failsafeCnt -= 20; else failsafeCnt = 0;
    }
  #endif
 }
 /**********************      atmega328P's Aux2 Pins
************************/
 #if defined(PROMINI)
  #if defined(RCAUXPIN)
  /* this ISR is a simplification of the previous one for PROMINI on port D
     it's simplier because we know the interruption deals only with one PIN:
     bit 0 of PORT B, ie Arduino PIN 8
     or bit 4 of PORTB, ie Arduino PIN 12
    => no need to check which PIN has changed */
  ISR(PCINT0_vect) {
    uint8_t pin;
    uint16_t cTime,dTime;
    static uint16_t edgeTime;

    pin = PINB;
    cTime = micros();
    sei();
    #if defined(RCAUXPIN8)
     if (!(pin & 1<<0)) {     //indicates if the bit 0 of the arduino port [B0-B7] is not at
a high state (so that we match here only descending PPM pulse)
    #endif
    #if defined(RCAUXPIN12)
     if (!(pin & 1<<4)) {     //indicates if the bit 4 of the arduino port [B0-B7] is not at
a high state (so that we match here only descending PPM pulse)
```

```
    #endif
      dTime = cTime-edgeTime; if (900<dTime && dTime<2200) rcValue[0] = dTime;
// just a verification: the value must be in the range [1000;2000] + some margin
    } else edgeTime = cTime;    // if the bit 2 is at a high state (ascending PPM pulse),
we memorize the time
   }
   #endif
  #endif

  /*****************      atmega32u4's Throttle & Aux2 Pin
*******************/
  #if defined(PROMICRO)
   // throttle
   ISR(INT6_vect){
    static uint16_t now,diff;
    static uint16_t last = 0;
    now = micros();
    if(!(PINE & (1<<6))){
     diff = now - last;
     if(900<diff && diff<2200){
      rcValue[3] = diff;
      #if defined(FAILSAFE)
       if(diff>FAILSAFE_DETECT_TRESHOLD) {       // if Throttle value is higher
than FAILSAFE_DETECT_TRESHOLD
         if(failsafeCnt > 20) failsafeCnt -= 20; else failsafeCnt = 0;   // If pulse
present on THROTTLE pin (independent from ardu version), clear FailSafe counter  -
added by MIS
       }
      #endif
     }
    }else last = now;
   }
   // Aux 2
   #if defined(RCAUX2PINRXO)
    ISR(INT2_vect){
     static uint16_t now,diff;
     static uint16_t last = 0;
     now = micros();
     if(!(PIND & (1<<2))){
      diff = now - last;
      if(900<diff && diff<2200) rcValue[7] = diff;
     }else last = now;
    }
   #endif
  #endif
#endif


/*****************************************************************
*****************/
```

```
/***************        PPM SUM RX Pin reading
********************/
/*************************************************************************
******************/
// attachInterrupt fix for promicro
#if defined(PROMICRO) && defined(SERIAL_SUM_PPM)
  ISR(INT6_vect){rxInt();}
#endif

// PPM_SUM at THROTTLE PIN on MEGA boards
#if defined(PPM_ON_THROTTLE) && defined(MEGA) &&
defined(SERIAL_SUM_PPM)
  ISR(PCINT2_vect) { if(PINK & (1<<0)) rxInt(); }
#endif

// Read PPM SUM RX Data
#if defined(SERIAL_SUM_PPM)
  void rxInt(void) {
    uint16_t now,diff;
    static uint16_t last = 0;
    static uint8_t chan = 0;
  #if defined(FAILSAFE)
    static uint8_t GoodPulses;
  #endif

    now = micros();
    sei();
    diff = now - last;
    last = now;
    if(diff>3000) chan = 0;
    else {
      if(900<diff && diff<2200 && chan<RC_CHANS ) {   //Only if the signal is
between these values it is valid, otherwise the failsafe counter should move up
        rcValue[chan] = diff;
        #if defined(FAILSAFE)
          if(chan<4 && diff>FAILSAFE_DETECT_TRESHOLD) GoodPulses |=
(1<<chan); // if signal is valid - mark channel as OK
          if(GoodPulses==0x0F) {                                  // If first four chanells
have good pulses, clear FailSafe counter
            GoodPulses = 0;
            if(failsafeCnt > 20) failsafeCnt -= 20; else failsafeCnt = 0;
          }
        #endif
      }
    chan++;
  }
}
#endif

/*************************************************************************
```

```
******************/
/***************          SBUS RX Data                    ********************/
/*****************************************************************************
******************/
#if defined(SBUS)

#define SBUS_SYNCBYTE 0x0F // Not 100% sure: at the beginning of coding it was
0xF0 !!!
static uint16_t sbusIndex=0;
static uint16_t sbus[25]={0};

void readSerial_RX(){
  while(SerialAvailable(RX_SERIAL_PORT)){
    int val = SerialRead(RX_SERIAL_PORT);
    if(sbusIndex==0 && val != SBUS_SYNCBYTE)
      continue;
    sbus[sbusIndex++] = val;
    if(sbusIndex==25){
      sbusIndex=0;
      spekFrameFlags = 0x00;
      rcValue[0]  = ((sbus[1]|sbus[2]<< 8) & 0x07FF)/2+SBUS_MID_OFFSET;
      rcValue[1]  = ((sbus[2]>>3|sbus[3]<<5) & 0x07FF)/2+SBUS_MID_OFFSET;
      rcValue[2]  = ((sbus[3]>>6|sbus[4]<<2|sbus[5]<<10) &
0x07FF)/2+SBUS_MID_OFFSET;
      rcValue[3]  = ((sbus[5]>>1|sbus[6]<<7) & 0x07FF)/2+SBUS_MID_OFFSET;
      rcValue[4]  = ((sbus[6]>>4|sbus[7]<<4) & 0x07FF)/2+SBUS_MID_OFFSET;
      rcValue[5]  = ((sbus[7]>>7|sbus[8]<<1|sbus[9]<<9) &
0x07FF)/2+SBUS_MID_OFFSET;
      rcValue[6]  = ((sbus[9]>>2|sbus[10]<<6) & 0x07FF)/2+SBUS_MID_OFFSET;
      rcValue[7]  = ((sbus[10]>>5|sbus[11]<<3) & 0x07FF)/2+SBUS_MID_OFFSET;
// & the other 8 + 2 channels if you need them
      //The following lines: If you need more than 8 channels, max 16 analog + 2
digital. Must comment the not needed channels!
      rcValue[8]  = ((sbus[12]|sbus[13]<< 8) & 0x07FF)/2+SBUS_MID_OFFSET;
      rcValue[9]  = ((sbus[13]>>3|sbus[14]<<5) &
0x07FF)/2+SBUS_MID_OFFSET;
      rcValue[10] = ((sbus[14]>>6|sbus[15]<<2|sbus[16]<<10) &
0x07FF)/2+SBUS_MID_OFFSET;
      rcValue[11] = ((sbus[16]>>1|sbus[17]<<7) &
0x07FF)/2+SBUS_MID_OFFSET;
      rcValue[12] = ((sbus[17]>>4|sbus[18]<<4) &
0x07FF)/2+SBUS_MID_OFFSET;
      rcValue[13] = ((sbus[18]>>7|sbus[19]<<1|sbus[20]<<9) &
0x07FF)/2+SBUS_MID_OFFSET;
      rcValue[14] = ((sbus[20]>>2|sbus[21]<<6) &
0x07FF)/2+SBUS_MID_OFFSET;
      rcValue[15] = ((sbus[21]>>5|sbus[22]<<3) &
0x07FF)/2+SBUS_MID_OFFSET;
      // now the two Digital-Channels
      if ((sbus[23]) & 0x0001)      rcValue[16] = 2000; else rcValue[16] = 1000;
```

```
    if ((sbus[23] >> 1) & 0x0001)  rcValue[17] = 2000; else rcValue[17] = 1000;
    spekFrameDone = 0x01;

    // Failsafe: there is one Bit in the SBUS-protocol (Byte 25, Bit 4) whitch is the
failsafe-indicator-bit
    #if defined(FAILSAFE)
    if (!((sbus[23] >> 3) & 0x0001))
      {if(failsafeCnt > 20) failsafeCnt -= 20; else failsafeCnt = 0;}   // clear FailSafe
counter
    #endif

    // For some reason the SBUS data provides only about 75% of the actual RX
output pulse width
    // Adjust the actual value by +/-25%.  Sign determined by pulse width above or
below center
    uint8_t adj_index;
    for(adj_index=0; adj_index<16; adj_index++) {
      if (rcValue[adj_index] < MIDRC)
        rcValue[adj_index] -= (MIDRC - rcValue[adj_index]) >> 2;
      else
        rcValue[adj_index] += (rcValue[adj_index] - MIDRC) >> 2;
    }
   }
  }
}
#endif

/**************************************************************************
******************/
/*************** SUMD ********************/
/**************************************************************************
******************/

#if defined(SUMD)
#define SUMD_SYNCBYTE 0xA8
#define SUMD_MAXCHAN 8
#define SUMD_BUFFSIZE SUMD_MAXCHAN*2 + 5 // 6 channels + 5 -> 17 bytes
for 6 channels
static uint8_t sumdIndex=0;
static uint8_t sumdSize=0;
static uint8_t sumd[SUMD_BUFFSIZE]={0};

void readSerial_RX(void) {
  while (SerialAvailable(RX_SERIAL_PORT)) {
    int val = SerialRead(RX_SERIAL_PORT);
    if(sumdIndex == 0 && val != SUMD_SYNCBYTE) continue;
    if(sumdIndex == 2) sumdSize = val;
    if(sumdIndex < SUMD_BUFFSIZE) sumd[sumdIndex] = val;
    sumdIndex++;
```

```
    if(sumdIndex == sumdSize*2+5) {
      sumdIndex = 0;
      spekFrameFlags = 0x00;
      debug[1] = sumd[1];
      if (sumdSize > SUMD_MAXCHAN) sumdSize = SUMD_MAXCHAN;
      for (uint8_t b = 0; b < sumdSize; b++)
        rcValue[b] = ((sumd[2*b+3]<<8) | sumd[2*b+4])>>3;
      spekFrameDone = 0x01; // havent checked crc at all

      #if defined(FAILSAFE)
      if (sumd[1] == 0x01)
        {if(failsafeCnt > 20) failsafeCnt -= 20; else failsafeCnt = 0;} // clear FailSafe
counter
      #endif
    }
  }
}
#endif

/**************************************************************************
******************/
/***************         combine and sort the RX Datas
*********************/
/**************************************************************************
******************/
#if defined(SPEKTRUM)
void readSerial_RX(void) {
  if ((!f.ARMED) &&
      #if defined(FAILSAFE) || (RX_SERIAL_PORT != 0)
        (failsafeCnt > 5) &&
      #endif
      ( SerialPeek(RX_SERIAL_PORT) == '$')) {
    while (SerialAvailable(RX_SERIAL_PORT)) {
      serialCom();
      delay (10);
    }
    return;
  } //End of: Is it the GUI?
  while (SerialAvailable(RX_SERIAL_PORT) > SPEK_FRAME_SIZE) { // More than
a frame?  More bytes implies we weren't called for multiple frame times.  We do not
want to process 'old' frames in the buffer.
    for (uint8_t i = 0; i < SPEK_FRAME_SIZE; i++)
{SerialRead(RX_SERIAL_PORT);}  //Toss one full frame of bytes.
  }
  if (spekFrameFlags == 0x01) {   //The interrupt handler saw at least one valid frame
start since we were last here.
    if (SerialAvailable(RX_SERIAL_PORT) == SPEK_FRAME_SIZE) {  //A complete
frame? If not, we'll catch it next time we are called.
      SerialRead(RX_SERIAL_PORT); SerialRead(RX_SERIAL_PORT);        //Eat the
header bytes
```

```
    for (uint8_t b = 2; b < SPEK_FRAME_SIZE; b += 2) {
      uint8_t bh = SerialRead(RX_SERIAL_PORT);
      uint8_t bl = SerialRead(RX_SERIAL_PORT);
      uint8_t spekChannel = 0x0F & (bh >> SPEK_CHAN_SHIFT);
      if (spekChannel < RC_CHANS) rcValue[spekChannel] = 988 + ((((uint16_t)(bh
& SPEK_CHAN_MASK) << 8) + bl) SPEK_DATA_SHIFT);
    }
    spekFrameFlags = 0x00;
    spekFrameDone = 0x01;
    #if defined(FAILSAFE)
      if(failsafeCnt > 20) failsafeCnt -= 20; else failsafeCnt = 0;   // Valid frame, clear
FailSafe counter
    #endif
  } else { //Start flag is on, but not enough bytes means there is an incomplete frame
in buffer.  This could be OK, if we happened to be called in the middle of a frame.  Or
not, if it has been a while since the start flag was set.
    uint32_t spekInterval = (timer0_overflow_count << 8) * (64 /
clockCyclesPerMicrosecond()) - spekTimeLast;
    if (spekInterval > 2500) {spekFrameFlags = 0;}  //If it has been a while, make the
interrupt handler start over.
  }
 }
}
#endif

uint16_t readRawRC(uint8_t chan) {
 uint16_t data;
 #if defined(SPEKTRUM) || defined(SBUS) || defined(SUMD)
  if (chan < RC_CHANS) {
    data = rcValue[rcChannel[chan]];
  } else data = 1500;
 #else
  uint8_t oldSREG;
  oldSREG = SREG; cli(); // Let's disable interrupts
  data = rcValue[rcChannel[chan]]; // Let's copy the data Atomically
  SREG = oldSREG;        // Let's restore interrupt state
 #endif
 return data; // We return the value correctly copied when the IRQ's where disabled
}

/**************************************************************************
*****************/
/***************         compute and Filter the RX data
********************/
/**************************************************************************
*****************/
#define AVERAGING_ARRAY_LENGTH 4
void computeRC() {
 static uint16_t rcData4Values[RC_CHANS][AVERAGING_ARRAY_LENGTH-1];
 uint16_t rcDataMean,rcDataTmp;
```

```
 static uint8_t rc4ValuesIndex = 0;
 uint8_t chan,a;
 uint8_t failsafeGoodCondition = 1;

 #if !defined(OPENLRSv2MULTI)
  rc4ValuesIndex++;
  if (rc4ValuesIndex == AVERAGING_ARRAY_LENGTH-1) rc4ValuesIndex = 0;
  for (chan = 0; chan < RC_CHANS; chan++) {
   rcDataTmp = readRawRC(chan);
   #if defined(FAILSAFE)
    failsafeGoodCondition = rcDataTmp>FAILSAFE_DETECT_TRESHOLD ||
chan > 3 || !f.ARMED; // update controls channel only if pulse is above
FAILSAFE_DETECT_TRESHOLD
   #endif                                                  // In disarmed state allow
always update for easer configuration.
   #if defined(SPEKTRUM) || defined(SBUS) || defined(SUMD) // no averaging for
Spektrum & SBUS & SUMD signal
    if(failsafeGoodCondition)  rcData[chan] = rcDataTmp;
   #else
    if(failsafeGoodCondition) {
     rcDataMean = rcDataTmp;
     for (a=0;a<AVERAGING_ARRAY_LENGTH-1;a++) rcDataMean +=
rcData4Values[chan][a];
     rcDataMean =
(rcDataMean+(AVERAGING_ARRAY_LENGTH/2))/AVERAGING_ARRAY_LENGTH
;
     if ( rcDataMean < (uint16_t)rcData[chan] -3)  rcData[chan] =
rcDataMean+2;
     if ( rcDataMean > (uint16_t)rcData[chan] +3)  rcData[chan] = rcDataMean-
2;
     rcData4Values[chan][rc4ValuesIndex] = rcDataTmp;
    }
   #endif
   if (chan<8 && rcSerialCount > 0) { // rcData comes from MSP and overrides RX
Data until rcSerialCount reaches 0
    rcSerialCount --;
    #if defined(FAILSAFE)
     failsafeCnt = 0;
    #endif
    if (rcSerial[chan] >900) {rcData[chan] = rcSerial[chan];} // only relevant
channels are overridden
   }
  }
 #endif
}


/**************************************************************************
*****************/
/***************        OPENLRS          ******************/
```

```
/************************************************************************
*****************/

//note: this dont feels right in RX.pde

#if defined(OPENLRSv2MULTI)
// ******************************************************
// ******************   OpenLRS Rx Code   ******************
// ***  OpenLRS Designed by Melih Karakelle on 2010-2012  ***
// **  an Arudino based RC Rx/Tx system with extra futures **
// **       This Source code licensed under GPL            **
// ******************************************************
// Version Number     : 1.11
// Latest Code Update : 2012-03-25
// Supported Hardware : OpenLRS Rx boards (store.flytron.com)
// Project Forum      : http://forum.flytron.com/viewforum.php?f=7
// Google Code Page   : http://code.google.com/p/openlrs/
// ******************************************************
// # PROJECT DEVELOPERS #
// Melih Karakelle (http://www.flytron.com) (forum nick name: Flytron)
// Jan-Dirk Schuitemaker (http://www.schuitemaker.org/) (forum nick name:
CrashingDutchman)
// Etienne Saint-Paul (http://www.gameseed.fr) (forum nick name: Etienne)
//

//######### TRANSMISSION VARIABLES ##########
#define CARRIER_FREQUENCY 435000  // 435Mhz startup frequency
#define FREQUENCY_HOPPING 1 // 1 = Enabled  0 = Disabled

//###### HOPPING CHANNELS #######
//Select the hopping channels between 0-255
// Default values are 13,54 and 23 for all transmitters and receivers, you should
change it before your first flight for safety.
//Frequency = CARRIER_FREQUENCY + (StepSize(60khz)* Channel_Number)
static uint8_t hop_list[3] = {13,54,23};

//###### RF DEVICE ID HEADERS #######
// Change this 4 byte values for isolating your transmission, RF module accepts only
data with same header
static uint8_t RF_Header[4] = {'O','L','R','S'};

//########## Variables #################

static uint32_t last_hopping_time;
static uint8_t RF_Rx_Buffer[17];
static uint16_t temp_int;
static uint16_t Servo_Buffer[10] = {3000,3000,3000,3000,3000,3000,3000,3000};
//servo position values from RF
static uint8_t hopping_channel = 1;
```

```
// ********************************************************
// **      RFM22B/Si4432 control functions for OpenLRS     **
// **       This Source code licensed under GPL            **
// ********************************************************
// Latest Code Update : 2011-09-26
// Supported Hardware : OpenLRS Tx/Rx boards (store.flytron.com)
// Project Forum      : http://forum.flytron.com/viewforum.php?f=7
// Google Code Page   : http://code.google.com/p/openlrs/
// ********************************************************

//**********************************************************************
*********
//**********************************************************************
*********
unsigned char ItStatus1, ItStatus2;

//-------------------------------------------------------------
void Write0( void ) {
  SCK_off;
  NOP();
  SDI_off;
  NOP();
  SCK_on;
  NOP();
}
//-------------------------------------------------------------
void Write1( void ) {
  SCK_off;
  NOP();
  SDI_on;
  NOP();
  SCK_on;
  NOP();
}
//-------------------------------------------------------------
void Write8bitcommand(uint8_t command) {  // keep sel to low
  uint8_t n=8;
  nSEL_on;
  SCK_off;
  nSEL_off;
  while(n--) {
    if(command&0x80)
      Write1();
    else
      Write0();
    command = command << 1;
  }
  SCK_off;
}
```

```
//---------------------------------------------------------------
void send_read_address(uint8_t i) {
  i &= 0x7f;
  Write8bitcommand(i);
}

//---------------------------------------------------------------
void send_8bit_data(uint8_t i) {
  uint8_t n = 8;
  SCK_off;
  while(n--) {
    if(i&0x80)
      Write1();
    else
      Write0();
    i = i << 1;
  }
  SCK_off;
}
//---------------------------------------------------------------

uint8_t read_8bit_data(void) {
  uint8_t Result, i;

  SCK_off;
  Result=0;
  for(i=0;i<8;i++) {                    //read fifo data byte
    Result=Result<<1;
    SCK_on;
    NOP();
    if(SDO_1) {
      Result|=1;
    }
    SCK_off;
    NOP();
  }
  return(Result);
}

//---------------------------------------------------------------
uint8_t _spi_read(uint8_t address) {
  uint8_t result;
  send_read_address(address);
  result = read_8bit_data();
  nSEL_on;
  return(result);
}

//---------------------------------------------------------------
```

```
void _spi_write(uint8_t address, uint8_t data) {
  address |= 0x80;
  Write8bitcommand(address);
  send_8bit_data(data);
  nSEL_on;
}

//-------Defaults 38.400 baud---------------------------------------------
void RF22B_init_parameter(void) {
  ItStatus1 = _spi_read(0x03); // read status, clear interrupt
  ItStatus2 = _spi_read(0x04);
  _spi_write(0x06, 0x00);    // no wakeup up, lbd,
  _spi_write(0x07, RF22B_PWRSTATE_READY);      // disable lbd, wakeup timer, use
internal 32768,xton = 1; in ready mode
  _spi_write(0x09, 0x7f);  // c = 12.5p
  _spi_write(0x0a, 0x05);
  _spi_write(0x0b, 0x12);    // gpio0 TX State
  _spi_write(0x0c, 0x15);    // gpio1 RX State
  _spi_write(0x0d, 0xfd);    // gpio 2 micro-controller clk output
  _spi_write(0x0e, 0x00);    // gpio    0, 1,2 NO OTHER FUNCTION.
  _spi_write(0x70, 0x00);    // disable manchest

  // 57.6Kbps data rate
  _spi_write(0x1c, 0x05); // case RATE_57.6K
  _spi_write(0x20, 0x45);//  0x20 calculate from the datasheet=
500*(1+2*down3_bypass)/(2^ndec*RB*(1+enmanch))
  _spi_write(0x21, 0x01); // 0x21 , rxosr[10--8] = 0; stalltr = (default), ccoff[19:16]
= 0;
  _spi_write(0x22, 0xD7); // 0x22    ncoff =5033 = 0x13a9
  _spi_write(0x23, 0xDC); // 0x23
  _spi_write(0x24, 0x03); // 0x24
  _spi_write(0x25, 0xB8); // 0x25
  _spi_write(0x2a, 0x1e);

  _spi_write(0x6e, 0x0E); //case RATE_57.6K
  _spi_write(0x6f, 0xBF); //case RATE_57.6K

  _spi_write(0x30, 0x8c);    // enable packet handler, msb first, enable crc,

  _spi_write(0x32, 0xf3);    // 0x32address enable for headere byte 0, 1,2,3, receive
header check for byte 0, 1,2,3
  _spi_write(0x33, 0x42);    // header 3, 2, 1,0 used for head length, fixed packet
length, synchronize word length 3, 2,
  _spi_write(0x34, 0x07);    // 7 default value or   // 64 nibble = 32byte preamble
  _spi_write(0x36, 0x2d);    // synchronize word
  _spi_write(0x37, 0xd4);
  _spi_write(0x38, 0x00);
  _spi_write(0x39, 0x00);
  _spi_write(0x3a, RF_Header[0]);    // tx header
  _spi_write(0x3b, RF_Header[1]);
```

```
  _spi_write(0x3c, RF_Header[2]);
  _spi_write(0x3d, RF_Header[3]);
  _spi_write(0x3e, 17);    // total tx 17 byte

  //RX HEADER
  _spi_write(0x3f, RF_Header[0]);   // check hearder
  _spi_write(0x40, RF_Header[1]);
  _spi_write(0x41, RF_Header[2]);
  _spi_write(0x42, RF_Header[3]);
  _spi_write(0x43, 0xff);    // all the bit to be checked
  _spi_write(0x44, 0xff);    // all the bit to be checked
  _spi_write(0x45, 0xff);    // all the bit to be checked
  _spi_write(0x46, 0xff);    // all the bit to be checked

  _spi_write(0x6d, 0x07); // 7 set power max power
  _spi_write(0x79, 0x00);    // no hopping
  _spi_write(0x7a, 0x06);    // 60khz step size (10khz x value) // no hopping

  _spi_write(0x71, 0x23); // Gfsk, fd[8] =0, no invert for Tx/Rx data, fifo mode, txclk -
->gpio
  // _spi_write(0x72, 0x1F); // frequency deviation setting to 19.6khz (for 38.4kbps)
  _spi_write(0x72, 0x2E); // frequency deviation setting to 28.8khz(for 57.6kbps)
  _spi_write(0x73, 0x00);
  _spi_write(0x74, 0x00);    // no offset

  //band 435.000
  _spi_write(0x75, 0x53);
  _spi_write(0x76, 0x7D);
  _spi_write(0x77, 0x00);
}


void checkPots() {
  ////Flytron OpenLRS Multi Pots
  pot_P = analogRead(7);
  pot_I = analogRead(6);

  pot_P = pot_P - 512;
  pot_I = pot_I - 512;

  pot_P = pot_P / 25; //+-20
  pot_I = pot_I / 25; //+-20
}

void initOpenLRS(void) {
  pinMode(GREEN_LED_pin, OUTPUT);
  pinMode(RED_LED_pin, OUTPUT);

  //RF module pins
  pinMode(SDO_pin, INPUT); //SDO
```

```
  pinMode(SDI_pin, OUTPUT); //SDI
  pinMode(SCLK_pin, OUTPUT); //SCLK
  pinMode(IRQ_pin, INPUT); //IRQ
  pinMode(nSel_pin, OUTPUT); //nSEL
  checkPots(); // OpenLRS Multi board hardware pot check;
}

//--------------------------------------------------------------------
void rx_reset(void) {
  _spi_write(0x07, RF22B_PWRSTATE_READY);
  _spi_write(0x7e, 36);    // threshold for rx almost full, interrupt when 1 byte received
  _spi_write(0x08, 0x03);    //clear fifo disable multi packet
  _spi_write(0x08, 0x00);    // clear fifo, disable multi packet
  _spi_write(0x07,RF22B_PWRSTATE_RX );  // to rx mode
  _spi_write(0x05, RF22B_Rx_packet_received_interrupt);
  ItStatus1 = _spi_read(0x03);  //read the Interrupt Status1 register
  ItStatus2 = _spi_read(0x04);
}
//--------------------------------------------------------------------

//-------------------------------------------------------------
void to_ready_mode(void) {
  ItStatus1 = _spi_read(0x03);
  ItStatus2 = _spi_read(0x04);
  _spi_write(0x07, RF22B_PWRSTATE_READY);
}

void to_rx_mode(void) {
  to_ready_mode();
  delay(50);
  rx_reset();
  NOP();
}

//-------------------------------------------------------------
void to_sleep_mode(void) {
  //  TXEN = RXEN = 0;
  //LED_RED = 0;
  _spi_write(0x07, RF22B_PWRSTATE_READY);

  ItStatus1 = _spi_read(0x03);  //read the Interrupt Status1 register
  ItStatus2 = _spi_read(0x04);
  _spi_write(0x07, RF22B_PWRSTATE_POWERDOWN);
}
//-------------------------------------------------------------

void frequency_configurator(uint32_t frequency) {
  // frequency formulation from Si4432 chip's datasheet
  // original formulation is working with mHz values and floating numbers, I replaced
them with kHz values.
```

```
  frequency = frequency / 10;
  frequency = frequency - 24000;
  frequency = frequency - 19000; // 19 for 430-439.9 MHz band from datasheet
  frequency = frequency * 64; // this is the Nominal Carrier Frequency (fc) value for
register setting

  uint8_t byte0 = (uint8_t) frequency;
  uint8_t byte1 = (uint8_t) (frequency >> 8);

  _spi_write(0x76, byte1);
  _spi_write(0x77, byte0);
}

//############## FREQUENCY HOPPING FUNCTIONS ##################
#if (FREQUENCY_HOPPING==1)
void Hopping(void) {
  hopping_channel++;
  if (hopping_channel>2) hopping_channel = 0;
  _spi_write(0x79, hop_list[hopping_channel]);
}
#endif

void Config_OpenLRS() {
  RF22B_init_parameter(); // Configure the RFM22B's registers
  frequency_configurator(CARRIER_FREQUENCY); // Calibrate the RFM22B to this
frequency, frequency hopping starts from here.
  to_rx_mode();
  #if (FREQUENCY_HOPPING==1)
    Hopping(); //Hop to the next frequency
  #endif
}

//############## MAIN LOOP ###############
void Read_OpenLRS_RC() {
  uint8_t i,tx_data_length;
  uint8_t first_data = 0;

  if (_spi_read(0x0C)==0) {RF22B_init_parameter(); to_rx_mode(); }// detect the
locked module and reboot
  if ((currentTime-last_hopping_time > 25000)) {//automatic hopping for clear
channel when rf link down for 25ms.
    Red_LED_ON;
    last_hopping_time = currentTime;
    #if (FREQUENCY_HOPPING==1)
      Hopping(); //Hop to the next frequency
    #endif
  }
  if(nIRQ_0) { // RFM22B INT pin Enabled by received Data
    Red_LED_ON;
    send_read_address(0x7f); // Send the package read command
```

```
  for(i = 0; i<17; i++) {//read all buffer
    RF_Rx_Buffer[i] = read_8bit_data();
  }
  rx_reset();
  if (RF_Rx_Buffer[0] == 'S') {// servo control data
   for(i = 0; i<8; i++) {//Write into the Servo Buffer
     temp_int = (256*RF_Rx_Buffer[1+(2*i)]) + RF_Rx_Buffer[2+(2*i)];
     if ((temp_int>1500) && (temp_int<4500)) Servo_Buffer[i] = temp_int/2;
   }
   rcData[ROLL] = Servo_Buffer[0];
   rcData[PITCH] = Servo_Buffer[1];
   rcData[THROTTLE] = Servo_Buffer[2];
   rcData[YAW] = Servo_Buffer[3];
   rcData[AUX1] = Servo_Buffer[4];
   rcData[AUX2] = Servo_Buffer[5];
   rcData[AUX3] = Servo_Buffer[6];
   rcData[AUX4] = Servo_Buffer[7];
  }
  #if (FREQUENCY_HOPPING==1)
   Hopping(); //Hop to the next frequency
  #endif
  delay(1);
  last_hopping_time = currentTime;
  Red_LED_OFF;
 }
 Red_LED_OFF;
}
#endif


#if defined(SPEK_BIND)  // Bind Support
void spekBind() {
 pinMode(SPEK_BIND_DATA, INPUT);     // Data line from sat
 digitalWrite(SPEK_BIND_DATA,LOW);   // Turn off internal Pull Up resistor

 pinMode(SPEK_BIND_GROUND, INPUT);
 digitalWrite(SPEK_BIND_GROUND,LOW);
 pinMode(SPEK_BIND_GROUND, OUTPUT);
 digitalWrite(SPEK_BIND_GROUND,LOW);

 pinMode(SPEK_BIND_POWER, INPUT);
 digitalWrite(SPEK_BIND_POWER,LOW);
 pinMode(SPEK_BIND_POWER,OUTPUT);

 while(1) {  //Do not return.  User presses reset button to return to normal.
  blinkLED(4,255,1);
  digitalWrite(SPEK_BIND_POWER,LOW); // Power off sat
  pinMode(SPEK_BIND_DATA, OUTPUT);
  digitalWrite(SPEK_BIND_DATA,LOW);
  delay(1000);
```

```
    blinkLED(4,255,1);

    digitalWrite(SPEK_BIND_POWER,HIGH); // Power on sat
    delay(10);
    digitalWrite(SPEK_BIND_DATA,HIGH);
    delay(60);                // Keep data pin steady for 20 to 120ms after power up

    noInterrupts();
    for (byte i = 0; i < SPEK_BIND_PULSES; i++) {
      digitalWrite(SPEK_BIND_DATA,LOW);
        delayMicroseconds(118);
      digitalWrite(SPEK_BIND_DATA,HIGH);
        delayMicroseconds(122);
    }
    interrupts();
    delay(60000);         //Allow one full minute to bind, then try again.
  }
}
#endif
καρτέλα RX.h
#ifndef RX_H_
#define RX_H_

void configureReceiver();
void computeRC();
uint16_t readRawRC(uint8_t chan);
void readSerial_RX(void);
#if defined(OPENLRSv2MULTI)
  void initOpenLRS(void);
  void Read_OpenLRS_RC(void);
#endif
#if defined(SPEK_BIND)  // Bind Support
  void spekBind(void);
#endif

#endif /* RX_H_ */


καρτέλα Sensors.cpp
#include "Arduino.h"
#include "config.h"
#include "def.h"
#include "types.h"
#include "MultiWii.h"
#include "Alarms.h"
#include "EEPROM.h"
#include "IMU.h"
#include "LCD.h"
#include "Sensors.h"
```

```
static void Device_Mag_getADC();
static void Baro_init();
static void Mag_init();
static void ACC_init();

//
**************************************************************************
************************************
// board orientation and setup
//
**************************************************************************
************************************
//default board orientation
#if !defined(ACC_ORIENTATION)
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = X;
imu.accADC[PITCH]  = Y; imu.accADC[YAW]  = Z;}
#endif
#if !defined(GYRO_ORIENTATION)
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = X;
imu.gyroADC[PITCH] = Y; imu.gyroADC[YAW] = Z;}
#endif
#if !defined(MAG_ORIENTATION)
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  = X;
imu.magADC[PITCH]  = Y; imu.magADC[YAW]  = Z;}
#endif


//ITG3200 / ITG3205 / ITG3050 / MPU6050 / MPU3050 Gyro LPF setting
#if defined(GYRO_LPF_256HZ) || defined(GYRO_LPF_188HZ) ||
defined(GYRO_LPF_98HZ) || defined(GYRO_LPF_42HZ) ||
defined(GYRO_LPF_20HZ) || defined(GYRO_LPF_10HZ) ||
defined(GYRO_LPF_5HZ)
  #if defined(GYRO_LPF_256HZ)
    #define GYRO_DLPF_CFG   0
  #endif
  #if defined(GYRO_LPF_188HZ)
    #define GYRO_DLPF_CFG   1
  #endif
  #if defined(GYRO_LPF_98HZ)
    #define GYRO_DLPF_CFG   2
  #endif
  #if defined(GYRO_LPF_42HZ)
    #define GYRO_DLPF_CFG   3
  #endif
  #if defined(GYRO_LPF_20HZ)
    #define GYRO_DLPF_CFG   4
  #endif
  #if defined(GYRO_LPF_10HZ)
    #define GYRO_DLPF_CFG   5
  #endif
```

```
  #if defined(GYRO_LPF_5HZ)
    #define GYRO_DLPF_CFG   6
  #endif
#else
    #define GYRO_DLPF_CFG   0 //Default settings LPF 256Hz/8000Hz sample
#endif

static uint8_t rawADC[6];
#if defined(WMP)
static uint32_t neutralizeTime = 0;
#endif

//
**************************************************************************
************************************
// I2C general functions
//
**************************************************************************
************************************

void i2c_init(void) {
  #if defined(INTERNAL_I2C_PULLUPS)
    I2C_PULLUPS_ENABLE
  #else
    I2C_PULLUPS_DISABLE
  #endif
  TWSR = 0;                                  // no prescaler => prescaler = 1
  TWBR = ((F_CPU / 400000) - 16) / 2;        // set the I2C clock rate to 400kHz
  TWCR = 1<<TWEN;                            // enable twi module, no interrupt
  i2c_errors_count = 0;
}

void __attribute__ ((noinline)) waitTransmissionI2C(uint8_t twcr) {
  TWCR = twcr;
  uint8_t count = 255;
  while (!(TWCR & (1<<TWINT))) {
    count--;
    if (count==0) {             //we are in a blocking state => we don't insist
      TWCR = 0;                 //and we force a reset on TWINT register
      #if defined(WMP)
      neutralizeTime = micros(); //we take a timestamp here to neutralize the value
during a short delay
      #endif
      i2c_errors_count++;
      break;
    }
  }
}

void i2c_rep_start(uint8_t address) {
```

```
  waitTransmissionI2C((1<<TWINT) | (1<<TWSTA) | (1<<TWEN)); // send REPEAT
START condition and wait until transmission completed
  TWDR = address;                                 // send device address
  waitTransmissionI2C((1<<TWINT) | (1<<TWEN));            // wail until
transmission completed
}

void i2c_stop(void) {
  TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
  //  while(TWCR & (1<<TWSTO));               // <- can produce a blocking state with
some WMP clones
}

void i2c_write(uint8_t data ) {
  TWDR = data;                                // send data to the previously addressed device
  waitTransmissionI2C((1<<TWINT) | (1<<TWEN));
}

uint8_t i2c_readAck() {
  waitTransmissionI2C((1<<TWINT) | (1<<TWEN) | (1<<TWEA));
  return TWDR;
}

uint8_t i2c_readNak() {
  waitTransmissionI2C((1<<TWINT) | (1<<TWEN));
  uint8_t r = TWDR;
  i2c_stop();
  return r;
}

void i2c_read_reg_to_buf(uint8_t add, uint8_t reg, uint8_t *buf, uint8_t size) {
  i2c_rep_start(add<<1); // I2C write direction
  i2c_write(reg);        // register selection
  i2c_rep_start((add<<1) | 1);  // I2C read direction
  uint8_t *b = buf;
  while (--size) *b++ = i2c_readAck(); // acknowledge all but the final byte
  *b = i2c_readNak();
}

void i2c_getSixRawADC(uint8_t add, uint8_t reg) {
  i2c_read_reg_to_buf(add, reg, rawADC, 6);
}

void i2c_writeReg(uint8_t add, uint8_t reg, uint8_t val) {
  i2c_rep_start(add<<1); // I2C write direction
  i2c_write(reg);        // register selection
  i2c_write(val);        // value to write in register
  i2c_stop();
}
```

```
uint8_t i2c_readReg(uint8_t add, uint8_t reg) {
  uint8_t val;
  i2c_read_reg_to_buf(add, reg, &val, 1);
  return val;
}

// ***************
// GYRO common part
// ***************
void GYRO_Common() {
  static int16_t previousGyroADC[3] = {0,0,0};
  static int32_t g[3];
  uint8_t axis, tilt=0;

  #if defined MMGYRO
    // Moving Average Gyros by Magnetron1
    //-----------------------------------------------
    static int16_t mediaMobileGyroADC[3][MMGYROVECTORLENGTH];
    static int32_t mediaMobileGyroADCSum[3];
    static uint8_t mediaMobileGyroIDX;
    //-----------------------------------------------
  #endif

  if (calibratingG>0) {
    for (axis = 0; axis < 3; axis++) {
      if (calibratingG == 512) { // Reset g[axis] at start of calibration
        g[axis]=0;
    #if defined(GYROCALIBRATIONFAILSAFE)
        previousGyroADC[axis] = imu.gyroADC[axis];
      }
      if (calibratingG % 10 == 0) {
        if(abs(imu.gyroADC[axis] - previousGyroADC[axis]) > 8) tilt=1;
        previousGyroADC[axis] = imu.gyroADC[axis];
    #endif
      }
      g[axis] +=imu.gyroADC[axis]; // Sum up 512 readings
      gyroZero[axis]=g[axis]>>9;
      if (calibratingG == 1) {
        SET_ALARM_BUZZER(ALRM_FAC_CONFIRM,
ALRM_LVL_CONFIRM_ELSE);
      }
    }
    #if defined(GYROCALIBRATIONFAILSAFE)
      if(tilt) {
        calibratingG=1000;
        LEDPIN_ON;
      } else {
        calibratingG--;
        LEDPIN_OFF;
      }
```

```
      return;
    #else
      calibratingG--;
    #endif
  }

  #ifdef MMGYRO
  mediaMobileGyroIDX = ++mediaMobileGyroIDX % conf.mmgyro;
  for (axis = 0; axis < 3; axis++) {
    imu.gyroADC[axis]  -= gyroZero[axis];
    mediaMobileGyroADCSum[axis] -=
mediaMobileGyroADC[axis][mediaMobileGyroIDX];
    //anti gyro glitch, limit the variation between two consecutive readings
    mediaMobileGyroADC[axis][mediaMobileGyroIDX] =
constrain(imu.gyroADC[axis],previousGyroADC[axis]-
800,previousGyroADC[axis]+800);
    mediaMobileGyroADCSum[axis] +=
mediaMobileGyroADC[axis][mediaMobileGyroIDX];
    imu.gyroADC[axis] = mediaMobileGyroADCSum[axis] / conf.mmgyro;
  #else
  for (axis = 0; axis < 3; axis++) {
    imu.gyroADC[axis]  -= gyroZero[axis];
    //anti gyro glitch, limit the variation between two consecutive readings
    imu.gyroADC[axis] = constrain(imu.gyroADC[axis],previousGyroADC[axis]-
800,previousGyroADC[axis]+800);
  #endif
    previousGyroADC[axis] = imu.gyroADC[axis];
  }

  #if defined(SENSORS_TILT_45DEG_LEFT)
    int16_t temp  = ((imu.gyroADC[PITCH] - imu.gyroADC[ROLL] )*7)/10;
    imu.gyroADC[ROLL] = ((imu.gyroADC[ROLL]  +
imu.gyroADC[PITCH])*7)/10;
    imu.gyroADC[PITCH]= temp;
  #endif
  #if defined(SENSORS_TILT_45DEG_RIGHT)
    int16_t temp  = ((imu.gyroADC[PITCH] + imu.gyroADC[ROLL] )*7)/10;
    imu.gyroADC[ROLL] = ((imu.gyroADC[ROLL]  - imu.gyroADC[PITCH])*7)/10;
    imu.gyroADC[PITCH]= temp;
  #endif
}

// ***************
// ACC common part
// ***************
void ACC_Common() {
  static int32_t a[3];
  if (calibratingA>0) {
    calibratingA--;
    for (uint8_t axis = 0; axis < 3; axis++) {
```

```
      if (calibratingA == 511) a[axis]=0;   // Reset a[axis] at start of calibration
      a[axis] +=imu.accADC[axis];           // Sum up 512 readings
      global_conf.accZero[axis] = a[axis]>>9; // Calculate average, only the last
itteration where (calibratingA == 0) is relevant
    }
    if (calibratingA == 0) {
      global_conf.accZero[YAW] -= ACC_1G;   // shift Z down by ACC_1G and store
values in EEPROM at end of calibration
      conf.angleTrim[ROLL]   = 0;
      conf.angleTrim[PITCH]  = 0;
      writeGlobalSet(1); // write accZero in EEPROM
    }
  }
  #if defined(INFLIGHT_ACC_CALIBRATION)
      static int32_t b[3];
      static int16_t accZero_saved[3]  = {0,0,0};
      static int16_t  angleTrim_saved[2] = {0, 0};
      //Saving old zeropoints before measurement
      if (InflightcalibratingA==50) {
        accZero_saved[ROLL]  = global_conf.accZero[ROLL] ;
        accZero_saved[PITCH] = global_conf.accZero[PITCH];
        accZero_saved[YAW]   = global_conf.accZero[YAW] ;
        angleTrim_saved[ROLL]  = conf.angleTrim[ROLL] ;
        angleTrim_saved[PITCH] = conf.angleTrim[PITCH] ;
      }
      if (InflightcalibratingA>0) {
        for (uint8_t axis = 0; axis < 3; axis++) {
          // Reset a[axis] at start of calibration
          if (InflightcalibratingA == 50) b[axis]=0;
          // Sum up 50 readings
          b[axis] +=imu.accADC[axis];
          // Clear global variables for next reading
          imu.accADC[axis]=0;
          global_conf.accZero[axis]=0;
        }
        //all values are measured
        if (InflightcalibratingA == 1) {
          AccInflightCalibrationActive = 0;
          AccInflightCalibrationMeasurementDone = 1;
          SET_ALARM_BUZZER(ALRM_FAC_CONFIRM,
ALRM_LVL_CONFIRM_1);     //buzzer for indicatiing the end of calibration
          // recover saved values to maintain current flight behavior until new values are
transferred
          global_conf.accZero[ROLL]  = accZero_saved[ROLL] ;
          global_conf.accZero[PITCH] = accZero_saved[PITCH];
          global_conf.accZero[YAW]   = accZero_saved[YAW] ;
          conf.angleTrim[ROLL]  = angleTrim_saved[ROLL] ;
          conf.angleTrim[PITCH] = angleTrim_saved[PITCH] ;
        }
        InflightcalibratingA--;
```

```
    }
    // Calculate average, shift Z down by ACC_1G and store values in EEPROM at
end of calibration
    if (AccInflightCalibrationSavetoEEProm == 1){  //the copter is landed, disarmed
and the combo has been done again
      AccInflightCalibrationSavetoEEProm = 0;
      global_conf.accZero[ROLL]  = b[ROLL]/50;
      global_conf.accZero[PITCH] = b[PITCH]/50;
      global_conf.accZero[YAW]   = b[YAW]/50-ACC_1G;
      conf.angleTrim[ROLL]   = 0;
      conf.angleTrim[PITCH]  = 0;
      writeGlobalSet(1); // write accZero in EEPROM
    }
  #endif
  imu.accADC[ROLL]  -=  global_conf.accZero[ROLL] ;
  imu.accADC[PITCH] -=  global_conf.accZero[PITCH];
  imu.accADC[YAW]   -=  global_conf.accZero[YAW] ;

  #if defined(SENSORS_TILT_45DEG_LEFT)
    int16_t temp = ((imu.accADC[PITCH] - imu.accADC[ROLL] )*7)/10;
    imu.accADC[ROLL] = ((imu.accADC[ROLL]  + imu.accADC[PITCH])*7)/10;
    imu.accADC[PITCH] = temp;
  #endif
  #if defined(SENSORS_TILT_45DEG_RIGHT)
    int16_t temp = ((imu.accADC[PITCH] + imu.accADC[ROLL] )*7)/10;
    imu.accADC[ROLL] = ((imu.accADC[ROLL]  - imu.accADC[PITCH])*7)/10;
    imu.accADC[PITCH] = temp;
  #endif
}

//
************************************************************************
************************************
// BARO section
//
************************************************************************
************************************
#if BARO
static void Baro_Common() {
  static int32_t baroHistTab[BARO_TAB_SIZE];
  static uint8_t baroHistIdx;

  uint8_t indexplus1 = (baroHistIdx + 1);
  if (indexplus1 == BARO_TAB_SIZE) indexplus1 = 0;
  baroHistTab[baroHistIdx] = baroPressure;
  baroPressureSum += baroHistTab[baroHistIdx];
  baroPressureSum -= baroHistTab[indexplus1];
  baroHistIdx = indexplus1;
}
#endif
```

```
//
**************************************************************************
************************************
// I2C Barometer BOSCH BMP085
//
**************************************************************************
************************************
// I2C adress: 0x77 (7bit)
// principle:
//  1) read the calibration register (only once at the initialization)
//  2) read uncompensated temperature (not mandatory at every cycle)
//  3) read uncompensated pressure
//  4) raw temp + raw pressure => calculation of the adjusted pressure
//  the following code uses the maximum precision setting (oversampling setting 3)
//
**************************************************************************
************************************

#if defined(BMP085)
#define BMP085_ADDRESS 0x77

static struct {
  // sensor registers from the BOSCH BMP085 datasheet
  int16_t  ac1, ac2, ac3;
  uint16_t ac4, ac5, ac6;
  int16_t  b1, b2, mb, mc, md;
  union {uint16_t val; uint8_t raw[2]; } ut; //uncompensated T
  union {uint32_t val; uint8_t raw[4]; } up; //uncompensated P
  uint8_t  state;
  uint32_t deadline;
} bmp085_ctx;
#define OSS 3

/* transform a series of bytes from big endian to little
   endian and vice versa. */
void swap_endianness(void *buf, size_t size) {
  /* we swap in-place, so we only have to
  * place _one_ element on a temporary tray
  */
  uint8_t tray;
  uint8_t *from;
  uint8_t *to;
  /* keep swapping until the pointers have assed each other */
  for (from = (uint8_t*)buf, to = &from[size-1]; from < to; from++, to--) {
    tray = *from;
    *from = *to;
    *to = tray;
  }
}
```

```
void i2c_BMP085_readCalibration(){
  delay(10);
  //read calibration data in one go
  size_t s_bytes = (uint8_t*)&bmp085_ctx.md - (uint8_t*)&bmp085_ctx.ac1 +
sizeof(bmp085_ctx.ac1);
  i2c_read_reg_to_buf(BMP085_ADDRESS, 0xAA, (uint8_t*)&bmp085_ctx.ac1,
s_bytes);
  // now fix endianness
  int16_t *p;
  for (p = &bmp085_ctx.ac1; p <= &bmp085_ctx.md; p++) {
    swap_endianness(p, sizeof(*p));
  }
}

// read uncompensated temperature value: send command first
void i2c_BMP085_UT_Start(void) {
  i2c_writeReg(BMP085_ADDRESS,0xf4,0x2e);
  i2c_rep_start(BMP085_ADDRESS<<1);
  i2c_write(0xF6);
  i2c_stop();
}

// read uncompensated pressure value: send command first
void i2c_BMP085_UP_Start () {
  i2c_writeReg(BMP085_ADDRESS,0xf4,0x34+(OSS<<6)); // control register value
for oversampling setting 3
  i2c_rep_start(BMP085_ADDRESS<<1); //I2C write direction => 0
  i2c_write(0xF6);
  i2c_stop();
}

// read uncompensated pressure value: read result bytes
// the datasheet suggests a delay of 25.5 ms (oversampling settings 3) after the send
command
void i2c_BMP085_UP_Read () {
  i2c_rep_start((BMP085_ADDRESS<<1) | 1);//I2C read direction => 1
  bmp085_ctx.up.raw[2] = i2c_readAck();
  bmp085_ctx.up.raw[1] = i2c_readAck();
  bmp085_ctx.up.raw[0] = i2c_readNak();
}

// read uncompensated temperature value: read result bytes
// the datasheet suggests a delay of 4.5 ms after the send command
void i2c_BMP085_UT_Read() {
  i2c_rep_start((BMP085_ADDRESS<<1) | 1);//I2C read direction => 1
  bmp085_ctx.ut.raw[1] = i2c_readAck();
  bmp085_ctx.ut.raw[0] = i2c_readNak();
}
```

```
void i2c_BMP085_Calculate() {
  int32_t  x1, x2, x3, b3, b5, b6, p, tmp;
  uint32_t b4, b7;
  // Temperature calculations
  x1 = ((int32_t)bmp085_ctx.ut.val - bmp085_ctx.ac6) * bmp085_ctx.ac5 >> 15;
  x2 = ((int32_t)bmp085_ctx.mc << 11) / (x1 + bmp085_ctx.md);
  b5 = x1 + x2;
  baroTemperature = (b5 * 10 + 8) >> 4; // in 0.01 degC (same as MS561101BA
temperature)
  // Pressure calculations
  b6 = b5 - 4000;
  x1 = (bmp085_ctx.b2 * (b6 * b6 >> 12)) >> 11;
  x2 = bmp085_ctx.ac2 * b6 >> 11;
  x3 = x1 + x2;
  tmp = bmp085_ctx.ac1;
  tmp = (tmp*4 + x3) << OSS;
  b3 = (tmp+2)/4;
  x1 = bmp085_ctx.ac3 * b6 >> 13;
  x2 = (bmp085_ctx.b1 * (b6 * b6 >> 12)) >> 16;
  x3 = ((x1 + x2) + 2) >> 2;
  b4 = (bmp085_ctx.ac4 * (uint32_t)(x3 + 32768)) >> 15;
  b7 = ((uint32_t) (bmp085_ctx.up.val >> (8-OSS)) - b3) * (50000 >> OSS);
  p = b7 < 0x80000000 ? (b7 * 2) / b4 : (b7 / b4) * 2;
  x1 = (p >> 8) * (p >> 8);
  x1 = (x1 * 3038) >> 16;
  x2 = (-7357 * p) >> 16;
  baroPressure = p + ((x1 + x2 + 3791) >> 4);
}

void  Baro_init() {
  delay(10);
  i2c_BMP085_readCalibration();
  delay(5);
  i2c_BMP085_UT_Start();
  bmp085_ctx.deadline = currentTime+5000;
}

//return 0: no data available, no computation ;  1: new value available  ; 2: no new
value, but computation time
uint8_t Baro_update() {                  // first UT conversion is started in init procedure
  if (currentTime < bmp085_ctx.deadline) return 0;
  bmp085_ctx.deadline = currentTime+6000; // 1.5ms margin according to the spec
(4.5ms T convetion time)
  if (bmp085_ctx.state == 0) {
    i2c_BMP085_UT_Read();
    i2c_BMP085_UP_Start();
    bmp085_ctx.state = 1;
    Baro_Common();
    bmp085_ctx.deadline += 21000;   // 6000+21000=27000 1.5ms margin according
to the spec (25.5ms P convetion time with OSS=3)
```

```
    return 1;
  } else {
    i2c_BMP085_UP_Read();
    i2c_BMP085_UT_Start();
    i2c_BMP085_Calculate();
    bmp085_ctx.state = 0;
    return 2;
  }
}
#endif

//
**************************************************************************
************************************
// I2C Barometer MS561101BA
//
**************************************************************************
************************************
//
// specs are here: http://www.meas-spec.com/downloads/MS5611-01BA03.pdf
// useful info on pages 7 -> 12
#if defined(MS561101BA)
#if !defined(MS561101BA_ADDRESS)
  #define MS561101BA_ADDRESS 0x77 //CBR=0 0xEE I2C address when pin CSB
is connected to LOW (GND)
  //#define MS561101BA_ADDRESS 0x76 //CBR=1 0xEC I2C address when pin
CSB is connected to HIGH (VCC)
#endif

// registers of the device
#define MS561101BA_PRESSURE    0x40
#define MS561101BA_TEMPERATURE 0x50
#define MS561101BA_RESET       0x1E

// OSR (Over Sampling Ratio) constants
#define MS561101BA_OSR_256  0x00
#define MS561101BA_OSR_512  0x02
#define MS561101BA_OSR_1024 0x04
#define MS561101BA_OSR_2048 0x06
#define MS561101BA_OSR_4096 0x08

#define OSR MS561101BA_OSR_4096

static struct {
  // sensor registers from the MS561101BA datasheet
  uint16_t c[7];
  uint32_t ut; //uncompensated T
  uint32_t up; //uncompensated P
  uint8_t  state;
  uint16_t deadline;
```

```
} ms561101ba_ctx;

static void Baro_init() {
  //reset
  i2c_writeReg(MS561101BA_ADDRESS, MS561101BA_RESET, 0);
  delay(100);

  //read calibration data
  union {uint16_t val; uint8_t raw[2]; } data;
  for(uint8_t i=0;i<6;i++) {
    i2c_rep_start(MS561101BA_ADDRESS<<1);
    i2c_write(0xA2+2*i);
    i2c_rep_start((MS561101BA_ADDRESS<<1) | 1);//I2C read direction => 1
    data.raw[1] = i2c_readAck();  // read a 16 bit register
    data.raw[0] = i2c_readNak();
    ms561101ba_ctx.c[i+1] = data.val;
  }
}

// read uncompensated temperature or pressure value: send command first
static void i2c_MS561101BA_UT_or_UP_Start(uint8_t reg) {
  i2c_rep_start(MS561101BA_ADDRESS<<1);     // I2C write direction
  i2c_write(reg);  // register selection
  i2c_stop();
}

static void i2c_MS561101BA_UT_or_UP_Read(uint32_t* val) {
  union {uint32_t val; uint8_t raw[4]; } data;
  i2c_rep_start(MS561101BA_ADDRESS<<1);
  i2c_write(0);
  i2c_rep_start((MS561101BA_ADDRESS<<1) | 1);
  data.raw[2] = i2c_readAck();
  data.raw[1] = i2c_readAck();
  data.raw[0] = i2c_readNak();
  *val = data.val;
}

// use float approximation instead of int64_t intermediate values
// does not use 2nd order compensation under -15 deg
static void i2c_MS561101BA_Calculate() {
  int32_t delt;

  float dT       = (int32_t)ms561101ba_ctx.ut -
(int32_t)((uint32_t)ms561101ba_ctx.c[5] << 8);
  float off      = ((uint32_t)ms561101ba_ctx.c[2] <<16) + ((dT *
ms561101ba_ctx.c[4]) /((uint32_t)1<<7));
  float sens     = ((uint32_t)ms561101ba_ctx.c[1] <<15) + ((dT *
ms561101ba_ctx.c[3]) /((uint32_t)1<<8));
  delt           = (dT * ms561101ba_ctx.c[6])/((uint32_t)1<<23);
  baroTemperature = delt + 2000;
```

```
  if (delt < 0) { // temperature lower than 20st.C
    delt *= 5 * delt;
    off  -= delt>>1;
    sens -= delt>>2;
  }
  baroPressure     = (( (ms561101ba_ctx.up * sens ) /((uint32_t)1<<21)) -
off)/((uint32_t)1<<15);
}

//return 0: no data available, no computation ;  1: new value available   or   no new
value but computation time
uint8_t Baro_update() {                       // first UT conversion is started in init
procedure
  uint32_t* rawValPointer;
  uint8_t   commandRegister;

  if (ms561101ba_ctx.state == 2) {              // a third state is introduced here to isolate
calculate() and smooth timecycle spike
    ms561101ba_ctx.state = 0;
    i2c_MS561101BA_Calculate();
    return 1;
  }
  if ((int16_t)(currentTime - ms561101ba_ctx.deadline)<0) return 0; // the initial timer
is not initialized, but in any case, no more than 65ms to wait.
  ms561101ba_ctx.deadline = currentTime+10000;  // UT and UP conversion take
8.5ms so we do next reading after 10ms
  if (ms561101ba_ctx.state == 0) {
    Baro_Common();                            // moved here for less timecycle spike, goes
after i2c_MS561101BA_Calculate
    rawValPointer = &ms561101ba_ctx.ut;
    commandRegister = MS561101BA_PRESSURE + OSR;
  } else {
    rawValPointer = &ms561101ba_ctx.up;
    commandRegister = MS561101BA_TEMPERATURE + OSR;
  }
  ms561101ba_ctx.state++;
  i2c_MS561101BA_UT_or_UP_Read(rawValPointer);     // get the 24bit resulting
from a UP of UT command request. Nothing interresting for the first cycle because
we don't initiate a command in Baro_init()
  i2c_MS561101BA_UT_or_UP_Start(commandRegister);  // send the next command
to get UP or UT value after at least 8.5ms
  return 1;
}
#endif

//
**********************************************************************
************************************
// I2C Accelerometer MMA7455
//
```

```
//*************************************************************************
//*************************************
#if defined(MMA7455)
#if !defined(MMA7455_ADDRESS)
  #define MMA7455_ADDRESS 0x1D
#endif

void ACC_init () {
  delay(10);
  i2c_writeReg(MMA7455_ADDRESS,0x16,0x21);
}

void ACC_getADC () {
  i2c_getSixRawADC(MMA7455_ADDRESS,0x00);

  ACC_ORIENTATION( ((int8_t(rawADC[1])<<8) | int8_t(rawADC[0])) ,
                   ((int8_t(rawADC[3])<<8) | int8_t(rawADC[2])) ,
                   ((int8_t(rawADC[5])<<8) | int8_t(rawADC[4])) );
  ACC_Common();
}
#endif

//
//*************************************************************************
//*************************************
// I2C Accelerometer MMA8451Q
//
//*************************************************************************
//*************************************
#if defined(MMA8451Q)

#if !defined(MMA8451Q_ADDRESS)
  #define MMA8451Q_ADDRESS 0x1C
  //#define MMA8451Q_ADDRESS 0x1D
#endif

void ACC_init () {
  delay(10);
  i2c_writeReg(MMA8451Q_ADDRESS,0x2A,0x05); // wake up & low noise
  delay(10);
  i2c_writeReg(MMA8451Q_ADDRESS,0x0E,0x02); // full scale range
}

void ACC_getADC () {
  i2c_getSixRawADC(MMA8451Q_ADDRESS,0x00);

  ACC_ORIENTATION( ((rawADC[1]<<8) | rawADC[0])/32 ,
                   ((rawADC[3]<<8) | rawADC[2])/32 ,
                   ((rawADC[5]<<8) | rawADC[4])/32);
  ACC_Common();
```

```
}
#endif

//
**************************************************************************
************************************
// I2C Accelerometer ADXL345
//
**************************************************************************
************************************
// I2C adress: 0x3A (8bit)    0x1D (7bit)
// Resolution: 10bit (Full range - 14bit, but this is autoscaling 10bit ADC to the range
+- 16g)
// principle:
//  1) CS PIN must be linked to VCC to select the I2C mode
//  2) SD0 PIN must be linked to VCC to select the right I2C adress
//  3) bit  b00000100 must be set on register 0x2D to read data (only once at the
initialization)
//  4) bits b00001011 must be set on register 0x31 to select the data format (only once
at the initialization)
//
**************************************************************************
************************************
#if defined(ADXL345)
#if !defined(ADXL345_ADDRESS)
  #define ADXL345_ADDRESS 0x1D
  //#define ADXL345_ADDRESS 0x53   //WARNING: Conflicts with a Wii Motion
plus!
#endif

void ACC_init () {
  delay(10);
  i2c_writeReg(ADXL345_ADDRESS,0x2D,1<<3); //  register: Power CTRL  --
value: Set measure bit 3 on
  i2c_writeReg(ADXL345_ADDRESS,0x31,0x0B); //  register: DATA_FORMAT --
value: Set bits 3(full range) and 1 0 on (+/- 16g-range)
  i2c_writeReg(ADXL345_ADDRESS,0x2C,0x09); //  register: BW_RATE     --
value: rate=50hz, bw=20hz
}

void ACC_getADC () {
  i2c_getSixRawADC(ADXL345_ADDRESS,0x32);

  ACC_ORIENTATION( ((rawADC[1]<<8) | rawADC[0]) ,
                   ((rawADC[3]<<8) | rawADC[2]) ,
                   ((rawADC[5]<<8) | rawADC[4]) );
  ACC_Common();
}
#endif
```

```
//
**********************************************************************
************************************
// I2C Accelerometer BMA180
//
**********************************************************************
************************************
// I2C adress: 0x80 (8bit)    0x40 (7bit) (SDO connection to VCC)
// I2C adress: 0x82 (8bit)    0x41 (7bit) (SDO connection to VDDIO)
// Resolution: 14bit
//
// Control registers:
//
// 0x20    bw_tcs:      |                              bw<3:0> |                  tcs<3:0> |
//                 |                          150Hz |                  xxxxxxxx |
// 0x30    tco_z:       |                                tco_z<5:0>    |
mode_config<1:0> |
//                 |                                xxxxxxxxxx    |               00 |
// 0x35    offset_lsb1: |        offset_x<3:0>            |               range<2:0>      |
smp_skip |
//                 |        xxxxxxxxxxxxx            |               8G:   101       | xxxxxxxx |
//
**********************************************************************
************************************
#if defined(BMA180)
#if !defined(BMA180_ADDRESS)
  #define BMA180_ADDRESS 0x40
  //#define BMA180_ADDRESS 0x41
#endif

void ACC_init () {
  delay(10);
  //default range 2G: 1G = 4096 unit.
  i2c_writeReg(BMA180_ADDRESS,0x0D,1<<4); // register: ctrl_reg0  -- value: set
bit ee_w to 1 to enable writing
  delay(5);
  uint8_t control = i2c_readReg(BMA180_ADDRESS, 0x20);
  control = control & 0x0F;        // save tcs register
  //control = control | (0x01 << 4); // register: bw_tcs reg: bits 4-7 to set bw -- value:
set low pass filter to 20Hz
  control = control | (0x00 << 4); // set low pass filter to 10Hz (bits value = 0000xxxx)
  i2c_writeReg(BMA180_ADDRESS, 0x20, control);
  delay(5);
  control = i2c_readReg(BMA180_ADDRESS, 0x30);
  control = control & 0xFC;        // save tco_z register
  control = control | 0x00;       // set mode_config to 0
  i2c_writeReg(BMA180_ADDRESS, 0x30, control);
  delay(5);
  control = i2c_readReg(BMA180_ADDRESS, 0x35);
  control = control & 0xF1;        // save offset_x and smp_skip register
```

```
  control = control | (0x05 << 1); // set range to 8G
  i2c_writeReg(BMA180_ADDRESS, 0x35, control);
  delay(5);
}

void ACC_getADC () {
  i2c_getSixRawADC(BMA180_ADDRESS,0x02);
  //usefull info is on the 14 bits  [2-15] bits  /4 => [0-13] bits  /4 => 12 bit resolution
  ACC_ORIENTATION( ((rawADC[1]<<8) | rawADC[0])>>4 ,
                   ((rawADC[3]<<8) | rawADC[2])>>4 ,
                   ((rawADC[5]<<8) | rawADC[4])>>4 );
  ACC_Common();
}
#endif

//
**************************************************************************
************************************
// I2C Accelerometer BMA280
//
**************************************************************************
************************************
#if defined(BMA280)
#if !defined(BMA280_ADDRESS)
  #define BMA280_ADDRESS 0x18 // SDO PIN on GND
  //#define BMA280_ADDRESS 0x19  // SDO PIN on Vddio
#endif

void ACC_init () {
  delay(10);
  i2c_writeReg(BMA280_ADDRESS, 0x10, 0x09); //set BW to 15,63Hz
  delay(5);
  i2c_writeReg(BMA280_ADDRESS, 0x0F, 0x08); //set range to 8G
}

void ACC_getADC () {
  i2c_getSixRawADC(BMA280_ADDRESS,0x02);
  //usefull info is on the 14 bits  [2-15] bits  /4 => [0-13] bits  /4 => 12 bit resolution
  ACC_ORIENTATION( ((rawADC[1]<<8) | rawADC[0])>>4 ,
                   ((rawADC[3]<<8) | rawADC[2])>>4 ,
                   ((rawADC[5]<<8) | rawADC[4])>>4 );
  ACC_Common();
}
#endif

//
**************************************************************************
************************************
// I2C Accelerometer BMA020
//
```

```
//**************************************************************************
//**************************************
// I2C adress: 0x70 (8bit)
// Resolution: 10bit
// Control registers:
//
// Datasheet: After power on reset or soft reset it is recommended to set the SPI4-bit to
the correct value.
//           0x80 = SPI four-wire = Default setting
// | 0x15: | SPI4 | enable_adv_INT | new_data_INT | latch_INT | shadow_dis |
wake_up_pause<1:0> | wake_up |
// |       |    1 |              0 |            0 |         0 |          0 |                 00 |       0 |
//
// | 0x14: |                    reserved <2:0> |           range <1:0> |              bandwith
<2:0> |
// |       |                    !!Calibration!! |                  2g |                    25Hz |
//
//
//**************************************************************************
//**************************************
#if defined(BMA020)
void ACC_init(){
  i2c_writeReg(0x38,0x15,0x80);    // set SPI4 bit
  uint8_t control = i2c_readReg(0x70, 0x14);
  control = control & 0xE0;        // save bits 7,6,5
  control = control | (0x02 << 3); // Range 8G (10)
  control = control | 0x00;        // Bandwidth 25 Hz 000
  i2c_writeReg(0x38,0x14,control);
}

void ACC_getADC(){
  i2c_getSixRawADC(0x38,0x02);
  ACC_ORIENTATION( ((rawADC[1]<<8) | rawADC[0])>>6 ,
                   ((rawADC[3]<<8) | rawADC[2])>>6 ,
                   ((rawADC[5]<<8) | rawADC[4])>>6 );
  ACC_Common();
}
#endif

//
//**************************************************************************
***
// LIS3LV02 I2C Accelerometer
//
//**************************************************************************
***
#if defined(LIS3LV02)
#define LIS3A  0x1D

void ACC_init(){
```

```
  i2c_writeReg(LIS3A ,0x20 ,0xD7 ); // CTRL_REG1   1101 0111 Pwr on, 160Hz
  i2c_writeReg(LIS3A ,0x21 ,0x50 ); // CTRL_REG2   0100 0000 Littl endian, 12 Bit,
Boot
}

void ACC_getADC(){
  i2c_getSixRawADC(LIS3A,0x28+0x80);
  ACC_ORIENTATION( ((rawADC[1]<<8) | rawADC[0])>>2 ,
                   ((rawADC[3]<<8) | rawADC[2])>>2 ,
                   ((rawADC[5]<<8) | rawADC[4])>>2);
  ACC_Common();
}
#endif

//
**************************************************************************
************************************
// I2C Accelerometer LSM303DLx
//
**************************************************************************
************************************
#if defined(LSM303DLx_ACC)
void ACC_init () {
  delay(10);
  i2c_writeReg(0x18,0x20,0x27);
  i2c_writeReg(0x18,0x23,0x30);
  i2c_writeReg(0x18,0x21,0x00);
}

  void ACC_getADC () {
  i2c_getSixRawADC(0x18,0xA8);

  ACC_ORIENTATION( ((rawADC[1]<<8) | rawADC[0])>>4 ,
                   ((rawADC[3]<<8) | rawADC[2])>>4 ,
                   ((rawADC[5]<<8) | rawADC[4])>>4 );
  ACC_Common();
}
#endif

//
**************************************************************************
************************************
// ADC ACC
//
**************************************************************************
************************************
#if defined(ADCACC)
void ACC_init(){
  pinMode(A1,INPUT);
  pinMode(A2,INPUT);
```

```
  pinMode(A3,INPUT);
}

void ACC_getADC() {
  ACC_ORIENTATION(  analogRead(A1) ,
                    analogRead(A2) ,
                    analogRead(A3) );
  ACC_Common();
}
#endif

//
**************************************************************************
************************************
// I2C Gyroscope L3G4200D
//
**************************************************************************
************************************
#if defined(L3G4200D)
#define L3G4200D_ADDRESS 0x69
void Gyro_init() {
  delay(100);
  i2c_writeReg(L3G4200D_ADDRESS ,0x20 ,0x8F ); // CTRL_REG1   400Hz ODR,
20hz filter, run!
  delay(5);
  i2c_writeReg(L3G4200D_ADDRESS ,0x24 ,0x02 ); // CTRL_REG5   low pass
filter enable
  delay(5);
  i2c_writeReg(L3G4200D_ADDRESS ,0x23 ,0x30); // CTRL_REG4 Select 2000dps
}

void Gyro_getADC () {
  i2c_getSixRawADC(L3G4200D_ADDRESS,0x80|0x28);

  GYRO_ORIENTATION( ((rawADC[1]<<8) | rawADC[0])>>2  ,
                    ((rawADC[3]<<8) | rawADC[2])>>2  ,
                    ((rawADC[5]<<8) | rawADC[4])>>2  );
  GYRO_Common();
}
#endif

//
**************************************************************************
************************************
// I2C Gyroscope ITG3200 / ITG3205 / ITG3050 / MPU3050
//
**************************************************************************
************************************
// I2C adress: 0xD2 (8bit)   0x69 (7bit)
// I2C adress: 0xD0 (8bit)   0x68 (7bit)
```

```
// principle:
// 1) VIO is connected to VDD
// 2) I2C adress is set to 0x69 (AD0 PIN connected to VDD)
// or 2) I2C adress is set to 0x68 (AD0 PIN connected to GND)
// 3) sample rate = 1000Hz ( 1kHz/(div+1) )
//
************************************************************************
************************************
#if defined(ITG3200) || defined(ITG3050) || defined(MPU3050)
#if !defined(GYRO_ADDRESS)
  #define GYRO_ADDRESS 0X68
  //#define GYRO_ADDRESS 0X69
#endif

void Gyro_init() {
  i2c_writeReg(GYRO_ADDRESS, 0x3E, 0x80);              //PWR_MGMT_1    --
DEVICE_RESET 1
  delay(5);
  i2c_writeReg(GYRO_ADDRESS, 0x16, 0x18 + GYRO_DLPF_CFG); //Gyro
CONFIG   -- EXT_SYNC_SET 0 (disable input pin for data sync) ; DLPF_CFG =
GYRO_DLPF_CFG ; -- FS_SEL = 3: Full scale set to 2000 deg/sec
  delay(5);
  i2c_writeReg(GYRO_ADDRESS, 0x3E, 0x03);              //PWR_MGMT_1    --
SLEEP 0; CYCLE 0; TEMP_DIS 0; CLKSEL 3 (PLL with Z Gyro reference)
  delay(100);
}

void Gyro_getADC () {
  i2c_getSixRawADC(GYRO_ADDRESS,0X1D);
  GYRO_ORIENTATION( ((rawADC[0]<<8) | rawADC[1])>>2 , // range: +/- 8192;
+/- 2000 deg/sec
                    ((rawADC[2]<<8) | rawADC[3])>>2 ,
                    ((rawADC[4]<<8) | rawADC[5])>>2 );
  GYRO_Common();
}
#endif


//
************************************************************************
************************************
// I2C Compass common function
//
************************************************************************
************************************
#if MAG
static float magGain[3] = {1.0,1.0,1.0};  // gain for each axis, populated at sensor init

uint8_t Mag_getADC() { // return 1 when news values are available, 0 otherwise
  static uint32_t t,tCal = 0;
```

```
  static int16_t magZeroTempMin[3],magZeroTempMax[3];
  uint8_t axis;

  if ( currentTime < t ) return 0; //each read is spaced by 100ms
  t = currentTime + 100000;
  Device_Mag_getADC();

  for(axis=0;axis<3;axis++) {
    imu.magADC[axis]  = imu.magADC[axis]  * magGain[axis];
    if (!f.CALIBRATE_MAG) imu.magADC[axis]  -= global_conf.magZero[axis];
  }

  if (f.CALIBRATE_MAG) {
    if (tCal == 0) // init mag calibration
      tCal = t;
    if ((t - tCal) < 30000000) { // 30s: you have 30s to turn the multi in all directions
      LEDPIN_TOGGLE;
      for(axis=0;axis<3;axis++) {
        if(tCal == t) { // it happens only in the first step, initialize the zero
          magZeroTempMin[axis] = imu.magADC[axis];
          magZeroTempMax[axis] = imu.magADC[axis];
        }
        if (imu.magADC[axis] < magZeroTempMin[axis]) {magZeroTempMin[axis] =
imu.magADC[axis]; SET_ALARM(ALRM_FAC_TOGGLE,
ALRM_LVL_TOGGLE_1);}
        if (imu.magADC[axis] > magZeroTempMax[axis]) {magZeroTempMax[axis] =
imu.magADC[axis]; SET_ALARM(ALRM_FAC_TOGGLE,
ALRM_LVL_TOGGLE_1);}
        global_conf.magZero[axis] = (magZeroTempMin[axis] +
magZeroTempMax[axis])>>1;
      }
    } else {
      f.CALIBRATE_MAG = 0;
      tCal = 0;
      writeGlobalSet(1);
    }
  }

  #if defined(SENSORS_TILT_45DEG_LEFT)
    int16_t temp = ((imu.magADC[PITCH] - imu.magADC[ROLL] )*7)/10;
    imu.magADC[ROLL] = ((imu.magADC[ROLL]  + imu.magADC[PITCH])*7)/10;
    imu.magADC[PITCH] = temp;
  #endif
  #if defined(SENSORS_TILT_45DEG_RIGHT)
    int16_t temp = ((imu.magADC[PITCH] + imu.magADC[ROLL] )*7)/10;
    imu.magADC[ROLL] = ((imu.magADC[ROLL]  - imu.magADC[PITCH])*7)/10;
    imu.magADC[PITCH] = temp;
  #endif

  return 1;
```

```
}
#endif

//
************************************************************************
************************************
// I2C Compass MAG3110
//
************************************************************************
************************************
// I2C adress: 0x0E (7bit)
//
************************************************************************
************************************
#if defined(MAG3110)
#define MAG_ADDRESS 0x0E
#define MAG_DATA_REGISTER 0x01
#define MAG_CTRL_REG1 0x10
#define MAG_CTRL_REG2 0x11

void Mag_init() {
  delay(100);
  i2c_writeReg(MAG_ADDRESS,MAG_CTRL_REG2,0x80);  //Automatic Magnetic
Sensor Reset
  delay(100);
  i2c_writeReg(MAG_ADDRESS,MAG_CTRL_REG1,0x11); // DR = 20Hz ; OS
ratio = 64 ; mode = Active
  delay(100);
}

#if !defined(MPU6050_I2C_AUX_MASTER)
  void Device_Mag_getADC() {
    i2c_getSixRawADC(MAG_ADDRESS,MAG_DATA_REGISTER);
    MAG_ORIENTATION( ((rawADC[0]<<8) | rawADC[1]) ,
                ((rawADC[2]<<8) | rawADC[3]) ,
                ((rawADC[4]<<8) | rawADC[5]) );
  }
#endif
#endif


//
************************************************************************
************************************
// I2C Compass HMC5883
//
************************************************************************
************************************
// I2C adress: 0x3C (8bit)   0x1E (7bit)
//
```

```
/**************************************************************************
***************************************

#if defined(HMC5883)

#define HMC58X3_R_CONFA 0
#define HMC58X3_R_CONFB 1
#define HMC58X3_R_MODE 2
#define HMC58X3_X_SELF_TEST_GAUSS (+1.16)                   //!< X axis level
when bias current is applied.
#define HMC58X3_Y_SELF_TEST_GAUSS (+1.16)   //!< Y axis level when bias
current is applied.
#define HMC58X3_Z_SELF_TEST_GAUSS (+1.08)                   //!< Y axis level
when bias current is applied.
#define SELF_TEST_LOW_LIMIT  (243.0/390.0)   //!< Low limit when gain is 5.
#define SELF_TEST_HIGH_LIMIT (575.0/390.0)   //!< High limit when gain is 5.
#define HMC_POS_BIAS 1
#define HMC_NEG_BIAS 2

#define MAG_ADDRESS 0x1E
#define MAG_DATA_REGISTER 0x03

static int32_t xyz_total[3]={0,0,0};  // 32 bit totals so they won't overflow.

static void getADC() {
  i2c_getSixRawADC(MAG_ADDRESS,MAG_DATA_REGISTER);
  MAG_ORIENTATION( ((rawADC[0]<<8) | rawADC[1]) ,
                ((rawADC[4]<<8) | rawADC[5]) ,
                ((rawADC[2]<<8) | rawADC[3]) );
}

static uint8_t bias_collect(uint8_t bias) {
  int16_t abs_magADC;

  i2c_writeReg(MAG_ADDRESS, HMC58X3_R_CONFA, bias);          // Reg A
DOR=0x010 + MS1,MS0 set to pos or negative bias
  for (uint8_t i=0; i<10; i++) {                        // Collect 10 samples
    i2c_writeReg(MAG_ADDRESS,HMC58X3_R_MODE, 1);
    delay(100);
    getADC();                                   // Get the raw values in case the scales
have already been changed.
    for (uint8_t axis=0; axis<3; axis++) {
      abs_magADC =  abs(imu.magADC[axis]);
      xyz_total[axis]+= abs_magADC;                     // Since the measurements are
noisy, they should be averaged rather than taking the max.
      if ((int16_t)(1<<12) < abs_magADC) return false;        // Detect saturation.   if
false Breaks out of the for loop.  No sense in continuing if we saturated.
    }
  }
  return true;
```

```
}

static void Mag_init() {
  bool bret=true;               // Error indicator

  // Note that the  very first measurement after a gain change maintains the same gain
as the previous setting.
  // The new gain setting is effective from the second measurement and on.
  i2c_writeReg(MAG_ADDRESS, HMC58X3_R_CONFB, 2 << 5);  //Set the Gain
  i2c_writeReg(MAG_ADDRESS,HMC58X3_R_MODE, 1);
  delay(100);
  getADC();  //Get one sample, and discard it

  if (!bias_collect(0x010 + HMC_POS_BIAS)) bret = false;
  if (!bias_collect(0x010 + HMC_NEG_BIAS)) bret = false;

  if (bret) // only if no saturation detected, compute the gain. otherwise, the default 1.0
is used
    for (uint8_t axis=0; axis<3; axis++)

magGain[axis]=820.0*HMC58X3_X_SELF_TEST_GAUSS*2.0*10.0/xyz_total[axis
];  // note: xyz_total[axis] is always positive

  // leave test mode
  i2c_writeReg(MAG_ADDRESS ,HMC58X3_R_CONFA ,0x70 ); //Configuration
Register A  -- 0 11 100 00  num samples: 8 ; output rate: 15Hz ; normal measurement
mode
  i2c_writeReg(MAG_ADDRESS ,HMC58X3_R_CONFB ,0x20 ); //Configuration
Register B  -- 001 00000    configuration gain 1.3Ga
  i2c_writeReg(MAG_ADDRESS ,HMC58X3_R_MODE  ,0x00 ); //Mode register
-- 000000 00    continuous Conversion Mode
  delay(100);
}

#if !defined(MPU6050_I2C_AUX_MASTER)
static void Device_Mag_getADC() {
  getADC();
}
#endif
#endif

//
**************************************************************************
************************************
// I2C Compass HMC5843
//
**************************************************************************
************************************
// I2C adress: 0x3C (8bit)   0x1E (7bit)
//
```

```
************************************************************************
************************************
#if defined(HMC5843)
#define MAG_ADDRESS 0x1E
#define MAG_DATA_REGISTER 0x03

void getADC() {
  i2c_getSixRawADC(MAG_ADDRESS,MAG_DATA_REGISTER);
  MAG_ORIENTATION( ((rawADC[0]<<8) | rawADC[1]) ,
                   ((rawADC[2]<<8) | rawADC[3]) ,
                   ((rawADC[4]<<8) | rawADC[5]) );
}

void Mag_init() {
  delay(100);
  // force positiveBias
  i2c_writeReg(MAG_ADDRESS ,0x00 ,0x71 ); //Configuration Register A  -- 0 11
100 01  num samples: 8 ; output rate: 15Hz ; positive bias
  delay(50);
  // set gains for calibration
  i2c_writeReg(MAG_ADDRESS ,0x01 ,0x60 ); //Configuration Register B  -- 011
00000    configuration gain 2.5Ga
  i2c_writeReg(MAG_ADDRESS ,0x02 ,0x01 ); //Mode register             -- 000000 01
single Conversion Mode

  // read values from the compass -  self test operation
  // by placing the mode register into single-measurement mode (0x01), two data
acquisition cycles will be made on each magnetic vector.
  // The first acquisition values will be subtracted from the second acquisition, and the
net measurement will be placed into the data output registers
  delay(100);
  getADC();
  delay(10);
  magGain[ROLL]  =  1000.0 / abs(imu.magADC[ROLL]);
  magGain[PITCH] =  1000.0 / abs(imu.magADC[PITCH]);
  magGain[YAW]   =  1000.0 / abs(imu.magADC[YAW]);

  // leave test mode
  i2c_writeReg(MAG_ADDRESS ,0x00 ,0x70 ); //Configuration Register A  -- 0 11
100 00  num samples: 8 ; output rate: 15Hz ; normal measurement mode
  i2c_writeReg(MAG_ADDRESS ,0x01 ,0x20 ); //Configuration Register B  -- 001
00000    configuration gain 1.3Ga
  i2c_writeReg(MAG_ADDRESS ,0x02 ,0x00 ); //Mode register             -- 000000 00
continuous Conversion Mode
}

#if !defined(MPU6050_I2C_AUX_MASTER)
void Device_Mag_getADC() {
  getADC();
}
```

```
#endif
#endif


//
************************************************************************
************************************
// I2C Compass AK8975
//
************************************************************************
************************************
// I2C adress: 0x0C (7bit)
//
************************************************************************
************************************
#if defined(AK8975)
  #define MAG_ADDRESS 0x0C
  #define MAG_DATA_REGISTER 0x03

  void Mag_init() {
    delay(100);
    i2c_writeReg(MAG_ADDRESS,0x0a,0x01);  //Start the first conversion
    delay(100);
  }

  void Device_Mag_getADC() {
    i2c_getSixRawADC(MAG_ADDRESS,MAG_DATA_REGISTER);
    MAG_ORIENTATION( ((rawADC[1]<<8) | rawADC[0]) ,
                     ((rawADC[3]<<8) | rawADC[2]) ,
                     ((rawADC[5]<<8) | rawADC[4]) );
    //Start another meassurement
    i2c_writeReg(MAG_ADDRESS,0x0a,0x01);
  }
#endif

//
************************************************************************
************************************
// I2C Gyroscope and Accelerometer MPU6050
//
************************************************************************
************************************
#if defined(MPU6050)
#if !defined(MPU6050_ADDRESS)
  #define MPU6050_ADDRESS     0x68 // address pin AD0 low (GND), default for
FreeIMU v0.4 and InvenSense evaluation board
  //#define MPU6050_ADDRESS     0x69 // address pin AD0 high (VCC)
#endif

static void Gyro_init() {
```

```
  i2c_writeReg(MPU6050_ADDRESS, 0x6B, 0x80);          //PWR_MGMT_1    --
DEVICE_RESET 1
  delay(50);
  i2c_writeReg(MPU6050_ADDRESS, 0x6B, 0x03);          //PWR_MGMT_1    --
SLEEP 0; CYCLE 0; TEMP_DIS 0; CLKSEL 3 (PLL with Z Gyro reference)
  i2c_writeReg(MPU6050_ADDRESS, 0x1A, GYRO_DLPF_CFG);    //CONFIG
-- EXT_SYNC_SET 0 (disable input pin for data sync) ; default DLPF_CFG = 0 =>
ACC bandwidth = 260Hz  GYRO bandwidth = 256Hz)
  i2c_writeReg(MPU6050_ADDRESS, 0x1B, 0x18);          //GYRO_CONFIG   --
FS_SEL = 3: Full scale set to 2000 deg/sec
  // enable I2C bypass for AUX I2C
  #if defined(MAG)
    i2c_writeReg(MPU6050_ADDRESS, 0x37, 0x02);          //INT_PIN_CFG   --
INT_LEVEL=0 ; INT_OPEN=0 ; LATCH_INT_EN=0 ; INT_RD_CLEAR=0 ;
FSYNC_INT_LEVEL=0 ; FSYNC_INT_EN=0 ; I2C_BYPASS_EN=1 ;
CLKOUT_EN=0
  #endif
}

void Gyro_getADC () {
  i2c_getSixRawADC(MPU6050_ADDRESS, 0x43);
  GYRO_ORIENTATION( ((rawADC[0]<<8) | rawADC[1])>>2 , // range: +/- 8192;
+/- 2000 deg/sec
                    ((rawADC[2]<<8) | rawADC[3])>>2 ,
                    ((rawADC[4]<<8) | rawADC[5])>>2 );
  GYRO_Common();
}

static void ACC_init () {
  i2c_writeReg(MPU6050_ADDRESS, 0x1C, 0x10);          //ACCEL_CONFIG  --
AFS_SEL=2 (Full Scale = +/-8G)  ; ACCELL_HPF=0   //note something is wrong in
the spec.
  //note: something seems to be wrong in the spec here. With AFS=2 1G = 4096 but
according to my measurement: 1G=2048 (and 2048/8 = 256)
  //confirmed here:
http://www.multiwii.com/forum/viewtopic.php?f=8&t=1080&start=10#p7480

  #if defined(MPU6050_I2C_AUX_MASTER)
    //at this stage, the MAG is configured via the original MAG init function in I2C
bypass mode
    //now we configure MPU as a I2C Master device to handle the MAG via the I2C
AUX port (done here for HMC5883)
    i2c_writeReg(MPU6050_ADDRESS, 0x6A, 0b00100000);      //USER_CTRL     --
DMP_EN=0 ; FIFO_EN=0 ; I2C_MST_EN=1 (I2C master mode) ; I2C_IF_DIS=0 ;
FIFO_RESET=0 ; I2C_MST_RESET=0 ; SIG_COND_RESET=0
    i2c_writeReg(MPU6050_ADDRESS, 0x37, 0x00);          //INT_PIN_CFG   --
INT_LEVEL=0 ; INT_OPEN=0 ; LATCH_INT_EN=0 ; INT_RD_CLEAR=0 ;
FSYNC_INT_LEVEL=0 ; FSYNC_INT_EN=0 ; I2C_BYPASS_EN=0 ;
CLKOUT_EN=0
    i2c_writeReg(MPU6050_ADDRESS, 0x24, 0x0D);          //I2C_MST_CTRL  --
```

```
MULT_MST_EN=0 ; WAIT_FOR_ES=0 ; SLV_3_FIFO_EN=0 ;
I2C_MST_P_NSR=0 ; I2C_MST_CLK=13 (I2C slave speed bus = 400kHz)
   i2c_writeReg(MPU6050_ADDRESS, 0x25,
0x80|MAG_ADDRESS);//I2C_SLV0_ADDR -- I2C_SLV4_RW=1 (read operation) ;
I2C_SLV4_ADDR=MAG_ADDRESS
   i2c_writeReg(MPU6050_ADDRESS, 0x26,
MAG_DATA_REGISTER);//I2C_SLV0_REG  -- 6 data bytes of MAG are stored in 6
registers. First register address is MAG_DATA_REGISTER
   i2c_writeReg(MPU6050_ADDRESS, 0x27, 0x86);            //I2C_SLV0_CTRL --
I2C_SLV0_EN=1 ; I2C_SLV0_BYTE_SW=0 ; I2C_SLV0_REG_DIS=0 ;
I2C_SLV0_GRP=0 ; I2C_SLV0_LEN=3 (3x2 bytes)
 #endif
}

void ACC_getADC () {
 i2c_getSixRawADC(MPU6050_ADDRESS, 0x3B);
 ACC_ORIENTATION( ((rawADC[0]<<8) | rawADC[1])>>3 ,
                  ((rawADC[2]<<8) | rawADC[3])>>3 ,
                  ((rawADC[4]<<8) | rawADC[5])>>3 );
 ACC_Common();
}

//The MAG acquisition function must be replaced because we now talk to the MPU
device
 #if defined(MPU6050_I2C_AUX_MASTER)
   static void Device_Mag_getADC() {
     i2c_getSixRawADC(MPU6050_ADDRESS, 0x49);              //0x49 is the first
memory room for EXT_SENS_DATA
     #if defined(HMC5843)
       MAG_ORIENTATION( ((rawADC[0]<<8) | rawADC[1]) ,
                        ((rawADC[2]<<8) | rawADC[3]) ,
                        ((rawADC[4]<<8) | rawADC[5]) );
     #endif
     #if defined (HMC5883)
       MAG_ORIENTATION( ((rawADC[0]<<8) | rawADC[1]) ,
                        ((rawADC[4]<<8) | rawADC[5]) ,
                        ((rawADC[2]<<8) | rawADC[3]) );
     #endif
     #if defined (MAG3110)
       MAG_ORIENTATION( ((rawADC[0]<<8) | rawADC[1]) ,
                        ((rawADC[2]<<8) | rawADC[3]) ,
                        ((rawADC[4]<<8) | rawADC[5]) );
     #endif
   }
 #endif
#endif

//
**********************************************************************
************************************
```

```
// Start Of I2C Gyroscope and Accelerometer LSM330
//
************************************************************************
************************************
#if defined(LSM330)
#if !defined(LSM330_ACC_ADDRESS)
  #define LSM330_ACC_ADDRESS     0x18 // 30 >> 1 = 18  -> address pin SDO_A
low (GND)
  //#define LSM330_ACC_ADDRESS     0x19 // 32 >> 1 = 19  -> address pin SDO_A
high (VCC)
#endif
#if !defined(LSM330_GYRO_ADDRESS)
  #define LSM330_GYRO_ADDRESS     0x6A // D4 >> 1 = 6A  -> address pin
SDO_G low (GND)
  //#define LSM330_GYRO_ADDRESS     0x6B // D6 >> 1 = 6B  -> address pin
SDO_G high (VCC)
#endif

////////////////////////////////////
//          ACC start           //
////////////////////////////////////
void ACC_init () {

  delay(10);
  //i2c_writeReg(LSM330_ACC_ADDRESS ,0x20 ,0x17 ); // 1Hz
  //i2c_writeReg(LSM330_ACC_ADDRESS ,0x20 ,0x27 ); // 10Hz
  i2c_writeReg(LSM330_ACC_ADDRESS ,0x20 ,0x37 );  // 25Hz
  //i2c_writeReg(LSM330_ACC_ADDRESS ,0x20 ,0x47 ); // 50Hz
  //i2c_writeReg(LSM330_ACC_ADDRESS ,0x20 ,0x57 ); // 100Hz

  delay(5);
  //i2c_writeReg(LSM330_ACC_ADDRESS ,0x23 ,0x08 ); // 2G
  //i2c_writeReg(LSM330_ACC_ADDRESS ,0x23 ,0x18 ); // 4G
  i2c_writeReg(LSM330_ACC_ADDRESS ,0x23 ,0x28 ); // 8G
  //i2c_writeReg(LSM330_ACC_ADDRESS ,0x23 ,0x38 ); // 16G

  delay(5);
  i2c_writeReg(LSM330_ACC_ADDRESS,0x21,0x00);// no high-pass filter
}

//#define ACC_DELIMITER 5 // for 2g
#define ACC_DELIMITER 4 // for 4g
//#define ACC_DELIMITER 3 // for 8g
//#define ACC_DELIMITER 2 // for 16g

  void ACC_getADC () {
  i2c_getSixRawADC(LSM330_ACC_ADDRESS,0x80|0x28);// Start multiple read at
reg 0x28

  ACC_ORIENTATION( ((rawADC[1]<<8) | rawADC[0])>>ACC_DELIMITER ,
```

```
              ((rawADC[3]<<8) | rawADC[2])>>ACC_DELIMITER ,
              ((rawADC[5]<<8) | rawADC[4])>>ACC_DELIMITER );
  ACC_Common();
}
/////////////////////////////////////
//          ACC end            //
/////////////////////////////////////

/////////////////////////////////////
//          Gyro start         //
/////////////////////////////////////
void Gyro_init() {
  delay(100);
  i2c_writeReg(LSM330_GYRO_ADDRESS ,0x20 ,0x8F ); // CTRL_REG1   400Hz
ODR, 20hz filter, run!
  delay(5);
  i2c_writeReg(LSM330_GYRO_ADDRESS ,0x24 ,0x02 ); // CTRL_REG5   low
pass filter enable
  delay(5);
  i2c_writeReg(LSM330_GYRO_ADDRESS ,0x23 ,0x30);  // CTRL_REG4 Select
2000dps
}

void Gyro_getADC () {
  i2c_getSixRawADC(LSM330_GYRO_ADDRESS,0x80|0x28);

  GYRO_ORIENTATION( ((rawADC[1]<<8) | rawADC[0])>>2  ,
              ((rawADC[3]<<8) | rawADC[2])>>2  ,
              ((rawADC[5]<<8) | rawADC[4])>>2  );
  GYRO_Common();
}
/////////////////////////////////////
//          Gyro end           //
/////////////////////////////////////

#endif /* LSM330 */

//
******************************************************************************
************************************
// End Of I2C Gyroscope and Accelerometer LSM330
//
******************************************************************************
************************************


#if defined(WMP)
//
******************************************************************************
************************************
```

```
// I2C Wii Motion Plus
//
*************************************************************************
************************************
// I2C adress 1: 0x53 (7bit)
// I2C adress 2: 0x52 (7bit)
//
*************************************************************************
************************************
#define WMP_ADDRESS_1 0x53
#define WMP_ADDRESS_2 0x52

void Gyro_init() {
  delay(250);
  i2c_writeReg(WMP_ADDRESS_1, 0xF0, 0x55); // Initialize Extension
  delay(250);
  i2c_writeReg(WMP_ADDRESS_1, 0xFE, 0x05); // Activate Nunchuck pass-through
mode
  delay(250);
}

void Gyro_getADC() {
  uint8_t axis;
  TWBR = ((F_CPU / I2C_SPEED) - 16) / 2; // change the I2C clock rate
  i2c_getSixRawADC(WMP_ADDRESS_2,0x00);
  TWBR = ((F_CPU / 400000) - 16) / 2; // change the I2C clock rate.

  if (micros() < (neutralizeTime + NEUTRALIZE_DELAY)) {//we neutralize data in
case of blocking+hard reset state
    for (axis = 0; axis < 3; axis++) {imu.gyroADC[axis]=0;imu.accADC[axis]=0;}
    imu.accADC[YAW] = ACC_1G;
  }

  // Wii Motion Plus Data
  if ( (rawADC[5]&0x03) == 0x02 ) {
    // Assemble 14bit data
    imu.gyroADC[ROLL]  = - ( ((rawADC[5]>>2)<<8) | rawADC[2] ); //range: +/-
8192
    imu.gyroADC[PITCH] = - ( ((rawADC[4]>>2)<<8) | rawADC[1] );
    imu.gyroADC[YAW]  =  - ( ((rawADC[3]>>2)<<8) | rawADC[0] );
    GYRO_Common();
    // Check if slow bit is set and normalize to fast mode range
    imu.gyroADC[ROLL]  = (rawADC[3]&0x01)     ? imu.gyroADC[ROLL]/5  :
imu.gyroADC[ROLL];  //the ratio 1/5 is not exactly the IDG600 or ISZ650
specification
    imu.gyroADC[PITCH] = (rawADC[4]&0x02)>>1  ? imu.gyroADC[PITCH]/5 :
imu.gyroADC[PITCH]; //we detect here the slow of fast mode WMP gyros values
(see wiibrew for more details)
    imu.gyroADC[YAW]   = (rawADC[3]&0x02)>>1  ? imu.gyroADC[YAW]/5   :
imu.gyroADC[YAW];   // this step must be done after zero compensation
```

```
  }
}
#endif


//
**********************************************************************
************************************
// I2C Sonar SRF08
//
**********************************************************************
************************************
// first contribution from guru_florida (02-25-2012)
//
#if defined(SRF02) || defined(SRF08) || defined(SRF10) || defined(SRC235)

// the default address for any new sensor found on the bus
// the code will move new sonars to the next available sonar address in range of F0-FE
so that another
// sonar sensor can be added again.
// Thus, add only 1 sonar sensor at a time, poweroff, then wire the next, power on,
wait for flashing light and repeat
#if !defined(SRF08_DEFAULT_ADDRESS)
  #define SRF08_DEFAULT_ADDRESS (0xE0>>1)
#endif

#if !defined(SRF08_RANGE_WAIT)
  #define SRF08_RANGE_WAIT      70000      // delay between Ping and Range Read
commands (65ms is safe in any case)
#endif

#if !defined(SRF08_RANGE_SLEEP)
  #define SRF08_RANGE_SLEEP     5000       // sleep this long before starting
another Ping
#endif

#if !defined(SRF08_SENSOR_FIRST)
  #define SRF08_SENSOR_FIRST    (0xF0>>1)  // the first sensor i2c address (after it
has been moved)
#endif

#if !defined(SRF08_MAX_SENSORS)
  #define SRF08_MAX_SENSORS     4          // maximum number of sensors we'll
allow (can go up to 8)
#endif

//#define SONAR_MULTICAST_PING

// registers of the device
#define SRF08_REV_COMMAND    0
```

```
#define SRF08_LIGHT_GAIN     1
#define SRF08_ECHO_RANGE     2


static struct {
  // sensor registers from the MS561101BA datasheet
  int32_t  range[SRF08_MAX_SENSORS];
  int8_t   sensors;             // the number of sensors present
  int8_t   current;             // the current sensor being read
  uint8_t  state;
  uint32_t deadline;
} srf08_ctx;


// read uncompensated temperature value: send command first
void Sonar_init() {
  memset(&srf08_ctx, 0, sizeof(srf08_ctx));
  srf08_ctx.deadline = 4000000;
}

// this function works like readReg accept a failed read is a normal expectation
// use for testing the existence of sensors on the i2c bus
// a 0xffff code is returned if the read failed
uint16_t i2c_try_readReg(uint8_t add, uint8_t reg) {
  uint16_t count = 255;
  i2c_rep_start(add<<1);  // I2C write direction
  i2c_write(reg);        // register selection
  i2c_rep_start((add<<1)|1);  // I2C read direction
  TWCR = (1<<TWINT) | (1<<TWEN);
  while (!(TWCR & (1<<TWINT))) {
    count--;
    if (count==0) {             //we are in a blocking state => we don't insist
      TWCR = 0;                 //and we force a reset on TWINT register
      return 0xffff;  // return failure to read
    }
  }
  uint8_t r = TWDR;
  i2c_stop();
  return r;
}

// read a 16bit unsigned int from the i2c bus
uint16_t i2c_readReg16(int8_t addr, int8_t reg) {
  uint8_t b[2];
  i2c_read_reg_to_buf(addr, reg, (uint8_t*)&b, sizeof(b));
  return (b[0]<<8) | b[1];
}

void i2c_srf08_change_addr(int8_t current, int8_t moveto) {
  // to change a srf08 address, we must write the following sequence to the command
```

```
register
  // this sequence must occur as 4 seperate i2c transactions!!   A0 AA A5 [addr]
  i2c_writeReg(current, SRF08_REV_COMMAND, 0xA0);    delay(30);
  i2c_writeReg(current, SRF08_REV_COMMAND, 0xAA);    delay(30);
  i2c_writeReg(current, SRF08_REV_COMMAND, 0xA5);    delay(30);
  i2c_writeReg(current, SRF08_REV_COMMAND, moveto);  delay(30); // now
change i2c address
}

// discover previously known sensors and any new sensor (move new sensors to
assigned area)
void i2c_srf08_discover() {
  uint8_t addr;
  uint16_t x;

  srf08_ctx.sensors=0;                              // determine how many sensors are
plugged in
  addr = SRF08_SENSOR_FIRST;                        // using the I2C address range
we choose, starting with first one
  for(uint8_t i=0; i<SRF08_MAX_SENSORS && x!=0xff; i++) {  // 0xff means a
mesure is currently running, so try again
    x = i2c_try_readReg(addr, SRF08_REV_COMMAND);         // read the revision as
a way to check if sensor exists at this location
    if(x!=0xffff) {                                 // detected a sensor at this address
      i2c_writeReg(addr, SRF08_LIGHT_GAIN, 0x15);         // not set to max to avoid
bad echo indoor
      i2c_writeReg(addr, SRF08_ECHO_RANGE, 46);           // set to 2m max
      srf08_ctx.sensors++;
      addr += 1;                                    // 7 bit address => +1 is +2 for 8 bit address
    }
  }
  if(srf08_ctx.sensors < SRF08_MAX_SENSORS) {               // do not add sensors if
we are already maxed
    // now determine if any sensor is on the 'new sensor' address (srf08 default address)
    x = i2c_try_readReg(SRF08_DEFAULT_ADDRESS, SRF08_REV_COMMAND);
// we try to read the revision number
    if(x!=0xffff) {                                 // new sensor detected at SRF08 default
address
      i2c_srf08_change_addr(SRF08_DEFAULT_ADDRESS, addr<<1);  // move
sensor to the next address (8 bit format expected by the device)
      srf08_ctx.sensors++;
    }
  }
}

void Sonar_update() {
  if ((int32_t)(currentTime - srf08_ctx.deadline)<0) return;
  srf08_ctx.deadline = currentTime;
  switch (srf08_ctx.state) {
    case 0:
```

```
      i2c_srf08_discover();
      if(srf08_ctx.sensors>0) srf08_ctx.state++;
      else                    srf08_ctx.deadline += 5000000; // wait 5 secs before trying search
again
      break;
    case 1:
      srf08_ctx.current=0;
      srf08_ctx.state++;
      srf08_ctx.deadline += SRF08_RANGE_SLEEP;
      break;
    #if defined(SONAR_MULTICAST_PING)
    case 2:
      // send a ping via the general broadcast address
      i2c_writeReg(0, SRF08_REV_COMMAND, 0x51);  // start ranging, result in
centimeters
      srf08_ctx.state++;
      srf08_ctx.deadline += SRF08_RANGE_WAIT;
      break;
    case 3:
      srf08_ctx.range[srf08_ctx.current] = i2c_readReg16(
SRF08_SENSOR_FIRST+srf08_ctx.current, SRF08_ECHO_RANGE);
      srf08_ctx.current++;
      if(srf08_ctx.current >= srf08_ctx.sensors) srf08_ctx.state=1;
      break;
    #else
    case 2:
      // send a ping to the current sensor
      i2c_writeReg(SRF08_SENSOR_FIRST+srf08_ctx.current,
SRF08_REV_COMMAND, 0x51);  // start ranging, result in centimeters
      srf08_ctx.state++;
      srf08_ctx.deadline += SRF08_RANGE_WAIT;
      break;
    case 3:
      srf08_ctx.range[srf08_ctx.current] =
i2c_readReg16(SRF08_SENSOR_FIRST+srf08_ctx.current,
SRF08_ECHO_RANGE);
      srf08_ctx.current++;
      if(srf08_ctx.current >= srf08_ctx.sensors) srf08_ctx.state=1;
      else                                       srf08_ctx.state=2;
      break;
    #endif
  }
  sonarAlt = srf08_ctx.range[0]; // only one sensor considered for the moment
}
#else
inline void Sonar_init() {}
void Sonar_update() {}
#endif
```

```
void initS() {
  i2c_init();
  if (GYRO)  Gyro_init();
  if (BARO)  Baro_init();
  if (MAG)   Mag_init();
  if (ACC)   ACC_init();
  if (SONAR) Sonar_init();
}

void initSensors() {
  uint8_t c = 5;
  #if !defined(DISABLE_POWER_PIN)
    POWERPIN_ON;
    delay(200);
  #endif
  while(c) { // We try several times to init all sensors without any i2c errors. An I2C
error at this stage might results in a wrong sensor settings
    c--;
    initS();
    if (i2c_errors_count == 0) break; // no error during init => init ok
  }
}
```

καρτέλα Sensors.h

```
#ifndef SENSORS_H_
#define SENSORS_H_

void ACC_getADC ();
void Gyro_getADC ();
uint8_t Mag_getADC();
uint8_t Baro_update();
void Sonar_update();

void initSensors();
void i2c_rep_start(uint8_t address);
void i2c_write(uint8_t data );
void i2c_stop(void);
void i2c_write(uint8_t data );
void i2c_writeReg(uint8_t add, uint8_t reg, uint8_t val);
uint8_t i2c_readReg(uint8_t add, uint8_t reg);
uint8_t i2c_readAck();
uint8_t i2c_readNak();

void i2c_read_reg_to_buf(uint8_t add, uint8_t reg, uint8_t *buf, uint8_t size);

#if defined(MMA7455)
  #define ACC_1G 64
#endif
#if defined(MMA8451Q)
  #define ACC_1G 512
#endif
```

```
#if defined(ADXL345)
  #define ACC_1G 265
#endif
#if defined(BMA180) || defined(BMA280) || defined(LIS3LV02) ||
defined(LSM303DLx_ACC) || defined(LSM330)
  #define ACC_1G 255
#endif
#if defined(BMA020)
  #define ACC_1G 63
#endif
#if defined(ADCACC)
  #define ACC_1G 75
#endif
#if defined(MPU6050)
  #if defined(FREEIMUv04)
    #define ACC_1G 255
  #else
    #define ACC_1G 512
  #endif
#endif
#if !defined(ACC_1G)
  #define ACC_1G 256
#endif
#define ACCZ_25deg   (int16_t)(ACC_1G * 0.90631) // 0.90631 = cos(25deg)
(cos(theta) of accZ comparison)
#define ACC_VelScale (9.80665f / 10000.0f / ACC_1G)

// GYRO SCALE: we ignore the last 2 bits and convert it for rad/s
#if defined(ITG3050)
  #define GYRO_SCALE (4 / 16.0 * PI / 180.0 / 1000000.0) //16.4 LSB = 1 deg/s  --
16.0 apparently gives beter results than 16.4 (empirical)
#endif
#if defined(MPU6050) || defined(MPU3050)
  #define GYRO_SCALE (4 / 16.4 * PI / 180.0 / 1000000.0) //16.4 LSB = 1 deg/s
#endif
#if defined(ITG3200)
  #define GYRO_SCALE (4 / 14.375 * PI / 180.0 / 1000000.0) //ITG3200   14.375
LSB = 1 deg/s
#endif
#if defined(L3G4200D) || defined(LSM330)
  #define GYRO_SCALE ((4.0f * PI * 70.0f)/(1000.0f * 180.0f * 1000000.0f)) // 70
milli deg/s /digit => 1 deg/s = 1000/70 LSB
#endif
#if defined(WMP)
  #define GYRO_SCALE (1.0f/200e6f)
#endif

#endif /* SENSORS_H_ */
```

καρτέλα Serial.cpp

```
#include "Arduino.h"
```

```
#include "config.h"
#include "def.h"
#include "Serial.h"
#include "MultiWii.h"

static volatile uint8_t
serialHeadRX[UART_NUMBER],serialTailRX[UART_NUMBER];
static uint8_t serialBufferRX[RX_BUFFER_SIZE][UART_NUMBER];
static volatile uint8_t
serialHeadTX[UART_NUMBER],serialTailTX[UART_NUMBER];
static uint8_t serialBufferTX[TX_BUFFER_SIZE][UART_NUMBER];


// ******************************************************
// For Teensy 2.0, these function emulate the API used for ProMicro
// it cant have the same name as in the arduino API because it wont compile for the
promini (eaven if it will be not compiled)
// ******************************************************
#if defined(TEENSY20)
  unsigned char T_USB_Available(){
    int n = Serial.available();
    if (n > 255) n = 255;
    return n;
  }
#endif

// ******************************************************
// Interrupt driven UART transmitter - using a ring buffer
// ******************************************************


#if defined(PROMINI) || defined(MEGA)
  #if defined(PROMINI)
  ISR(USART_UDRE_vect) {  // Serial 0 on a PROMINI
  #endif
  #if defined(MEGA)
  ISR(USART0_UDRE_vect) { // Serial 0 on a MEGA
  #endif
    uint8_t t = serialTailTX[0];
    if (serialHeadTX[0] != t) {
      if (++t >= TX_BUFFER_SIZE) t = 0;
      UDR0 = serialBufferTX[t][0];  // Transmit next byte in the ring
      serialTailTX[0] = t;
    }
    if (t == serialHeadTX[0]) UCSR0B &= ~(1<<UDRIE0); // Check if all data is
transmitted . if yes disable transmitter UDRE interrupt
  }
#endif
#if defined(MEGA) || defined(PROMICRO)
  ISR(USART1_UDRE_vect) { // Serial 1 on a MEGA or on a PROMICRO
```

```
    uint8_t t = serialTailTX[1];
    if (serialHeadTX[1] != t) {
      if (++t >= TX_BUFFER_SIZE) t = 0;
      UDR1 = serialBufferTX[t][1];  // Transmit next byte in the ring
      serialTailTX[1] = t;
    }
    if (t == serialHeadTX[1]) UCSR1B &= ~(1<<UDRIE1);
  }
#endif
#if defined(MEGA)
  ISR(USART2_UDRE_vect) { // Serial 2 on a MEGA
    uint8_t t = serialTailTX[2];
    if (serialHeadTX[2] != t) {
      if (++t >= TX_BUFFER_SIZE) t = 0;
      UDR2 = serialBufferTX[t][2];
      serialTailTX[2] = t;
    }
    if (t == serialHeadTX[2]) UCSR2B &= ~(1<<UDRIE2);
  }
  ISR(USART3_UDRE_vect) { // Serial 3 on a MEGA
    uint8_t t = serialTailTX[3];
    if (serialHeadTX[3] != t) {
      if (++t >= TX_BUFFER_SIZE) t = 0;
      UDR3 = serialBufferTX[t][3];
      serialTailTX[3] = t;
    }
    if (t == serialHeadTX[3]) UCSR3B &= ~(1<<UDRIE3);
  }
#endif

void UartSendData(uint8_t port) {
  #if defined(PROMINI)
    UCSR0B |= (1<<UDRIE0);
  #endif
  #if defined(PROMICRO)
    switch (port) {
      case 0:
        while(serialHeadTX[0] != serialTailTX[0]) {
          if (++serialTailTX[0] >= TX_BUFFER_SIZE) serialTailTX[0] = 0;
          #if !defined(TEENSY20)
            USB_Send(USB_CDC_TX,serialBufferTX[serialTailTX[0]],1);
          #else
            Serial.write(serialBufferTX[serialTailTX[0]],1);
          #endif
         }
        break;
      case 1: UCSR1B |= (1<<UDRIE1); break;
    }
  #endif
  #if defined(MEGA)
```

```
    switch (port) {
      case 0: UCSR0B |= (1<<UDRIE0); break;
      case 1: UCSR1B |= (1<<UDRIE1); break;
      case 2: UCSR2B |= (1<<UDRIE2); break;
      case 3: UCSR3B |= (1<<UDRIE3); break;
    }
  #endif
}

#if defined(GPS_SERIAL)
  bool SerialTXfree(uint8_t port) {
    return (serialHeadTX[port] == serialTailTX[port]);
  }
#endif

void SerialOpen(uint8_t port, uint32_t baud) {
  uint8_t h = ((F_CPU  / 4 / baud -1) / 2) >> 8;
  uint8_t l = ((F_CPU  / 4 / baud -1) / 2);
  switch (port) {
    #if defined(PROMINI)
      case 0: UCSR0A  = (1<<U2X0); UBRR0H = h; UBRR0L = l; UCSR0B |=
(1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0); break;
    #endif
    #if defined(PROMICRO)
      #if (ARDUINO >= 100) && !defined(TEENSY20)
        case 0: UDIEN &= ~(1<<SOFE); break;// disable the USB frame interrupt of
arduino (it causes strong jitter and we dont need it)
      #endif
      case 1: UCSR1A  = (1<<U2X1); UBRR1H = h; UBRR1L = l; UCSR1B |=
(1<<RXEN1)|(1<<TXEN1)|(1<<RXCIE1); break;
    #endif
    #if defined(MEGA)
      case 0: UCSR0A  = (1<<U2X0); UBRR0H = h; UBRR0L = l; UCSR0B |=
(1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0); break;
      case 1: UCSR1A  = (1<<U2X1); UBRR1H = h; UBRR1L = l; UCSR1B |=
(1<<RXEN1)|(1<<TXEN1)|(1<<RXCIE1); break;
      case 2: UCSR2A  = (1<<U2X2); UBRR2H = h; UBRR2L = l; UCSR2B |=
(1<<RXEN2)|(1<<TXEN2)|(1<<RXCIE2); break;
      case 3: UCSR3A  = (1<<U2X3); UBRR3H = h; UBRR3L = l; UCSR3B |=
(1<<RXEN3)|(1<<TXEN3)|(1<<RXCIE3); break;
    #endif
  }
}

void SerialEnd(uint8_t port) {
  switch (port) {
    #if defined(PROMINI)
      case 0: UCSR0B &=
~((1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0)|(1<<UDRIE0)); break;
    #endif
```

```
    #if defined(PROMICRO)
      case 1: UCSR1B &=
~((1<<RXEN1)|(1<<TXEN1)|(1<<RXCIE1)|(1<<UDRIE1)); break;
    #endif
    #if defined(MEGA)
      case 0: UCSR0B &=
~((1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0)|(1<<UDRIE0)); break;
      case 1: UCSR1B &=
~((1<<RXEN1)|(1<<TXEN1)|(1<<RXCIE1)|(1<<UDRIE1)); break;
      case 2: UCSR2B &=
~((1<<RXEN2)|(1<<TXEN2)|(1<<RXCIE2)|(1<<UDRIE2)); break;
      case 3: UCSR3B &=
~((1<<RXEN3)|(1<<TXEN3)|(1<<RXCIE3)|(1<<UDRIE3)); break;
    #endif
  }
}

// we don't care about ring buffer overflow (head->tail) to avoid a test condition : data
is lost anyway if it happens
void store_uart_in_buf(uint8_t data, uint8_t portnum) {
  #if defined(SERIAL_RX)
    if (portnum == RX_SERIAL_PORT) {
      if (!spekFrameFlags) {
        sei();
        uint32_t spekTimeNow = (timer0_overflow_count << 8) * (64 /
clockCyclesPerMicrosecond()); //Move timer0_overflow_count into registers so we
don't touch a volatile twice
        uint32_t spekInterval = spekTimeNow - spekTimeLast;
//timer0_overflow_count will be slightly off because of the way the Arduino core
timer interrupt handler works; that is acceptable for this use. Using the core variable
avoids an expensive call to millis() or micros()
        spekTimeLast = spekTimeNow;
        if (spekInterval > 2500) {  //Potential start of a Spektrum frame, they arrive
every 11 or every 22 ms. Mark it, and clear the buffer.
          serialTailRX[portnum] = 0;
          serialHeadRX[portnum] = 0;
          spekFrameFlags = 0x01;
        }
        cli();
      }
    }
  #endif

  uint8_t h = serialHeadRX[portnum];
  serialBufferRX[h++][portnum] = data;
  if (h >= RX_BUFFER_SIZE) h = 0;
  serialHeadRX[portnum] = h;
}

#if defined(PROMINI)
```

```
  ISR(USART_RX_vect)  { store_uart_in_buf(UDR0, 0); }
#endif
#if defined(PROMICRO)
  ISR(USART1_RX_vect)  { store_uart_in_buf(UDR1, 1); }
#endif
#if defined(MEGA)
  ISR(USART0_RX_vect)  { store_uart_in_buf(UDR0, 0); }
  ISR(USART1_RX_vect)  { store_uart_in_buf(UDR1, 1); }
  ISR(USART2_RX_vect)  { store_uart_in_buf(UDR2, 2); }
  ISR(USART3_RX_vect)  { store_uart_in_buf(UDR3, 3); }
#endif

uint8_t SerialRead(uint8_t port) {
  #if defined(PROMICRO)
    #if defined(TEENSY20)
      if(port == 0) return Serial.read();
    #else
      #if (ARDUINO >= 100)
        if(port == 0) USB_Flush(USB_CDC_TX);
      #endif
      if(port == 0) return USB_Recv(USB_CDC_RX);
    #endif
  #endif
  uint8_t t = serialTailRX[port];
  uint8_t c = serialBufferRX[t][port];
  if (serialHeadRX[port] != t) {
    if (++t >= RX_BUFFER_SIZE) t = 0;
    serialTailRX[port] = t;
  }
  return c;
}

#if defined(SERIAL_RX)
  uint8_t SerialPeek(uint8_t port) {
    uint8_t c = serialBufferRX[serialTailRX[port]][port];
    if ((serialHeadRX[port] != serialTailRX[port])) return c; else return 0;
  }
#endif

uint8_t SerialAvailable(uint8_t port) {
  #if defined(PROMICRO)
    #if !defined(TEENSY20)
      if(port == 0) return USB_Available(USB_CDC_RX);
    #else
      if(port == 0) return T_USB_Available();
    #endif
  #endif
  return ((uint8_t)(serialHeadRX[port] - serialTailRX[port]))%RX_BUFFER_SIZE;
}
```

```
uint8_t SerialUsedTXBuff(uint8_t port) {
  return ((uint8_t)(serialHeadTX[port] - serialTailTX[port]))%TX_BUFFER_SIZE;
}

void SerialSerialize(uint8_t port,uint8_t a) {
  uint8_t t = serialHeadTX[port];
  if (++t >= TX_BUFFER_SIZE) t = 0;
  serialBufferTX[t][port] = a;
  serialHeadTX[port] = t;
}

void SerialWrite(uint8_t port,uint8_t c){
  SerialSerialize(port,c);UartSendData(port);
}
```

καρτέλα Serial.h

```
#ifndef SERIAL_H_
#define SERIAL_H_

#if defined(MEGA)
  #define UART_NUMBER 4
#elif defined(PROMICRO)
  #define UART_NUMBER 2
#else
  #define UART_NUMBER 1
#endif
#define RX_BUFFER_SIZE 256 // 256 RX buffer is needed for GPS communication
(64 or 128 was too short)
#define TX_BUFFER_SIZE 128

void    SerialOpen(uint8_t port, uint32_t baud);
uint8_t SerialRead(uint8_t port);
void    SerialWrite(uint8_t port,uint8_t c);
uint8_t SerialAvailable(uint8_t port);
void    SerialEnd(uint8_t port);
uint8_t SerialPeek(uint8_t port);
bool    SerialTXfree(uint8_t port);
uint8_t SerialUsedTXBuff(uint8_t port);
void    SerialSerialize(uint8_t port,uint8_t a);
void    UartSendData(uint8_t port);

void SerialWrite16(uint8_t port, int16_t val);

#endif /* SERIAL_H_ */
```

καρτέλα config.h

```
#ifndef CONFIG_H_
#define CONFIG_H_

/*****************************************************************
*****************************/
/****          CONFIGURABLE PARAMETERS
```

```
****/
/*****************************************************************
*****************************/

/* this file consists of several sections
 * to create a working combination you must at least make your choices in section 1.
 * 1 - BASIC SETUP - you must select an option in every block.
 *      this assumes you have 4 channels connected to your board with standard ESCs
and servos.
 * 2 - COPTER TYPE SPECIFIC OPTIONS - you likely want to check for options for
your copter type
 * 3 - RC SYSTEM SETUP
 * 4 - ALTERNATE CPUs & BOARDS - if you have
 * 5 - ALTERNATE SETUP - select alternate RX (SBUS, PPM, etc.), alternate ESC-
range, etc. here
 * 6 - OPTIONAL FEATURES - enable nice to have features here (FlightModes,
LCD, telemetry, battery monitor etc.)
 * 7 - TUNING & DEVELOPER - if you know what you are doing; you have been
warned
 *     - (ESCs calibration, Dynamic Motor/Prop Balancing, Diagnostics,Memory
savings.....)
 * 8 - DEPRECATED - these features will be removed in some future release
 */

/* Notes:
 * 1. parameters marked with (*) in the comment are stored in eeprom and can be
changed via serial monitor or LCD.
 * 2. parameters marked with (**) in the comment are stored in eeprom and can be
changed via the GUI
 */


/*****************************************************************
*****************************/
/*****************                                                 **************/
/***************  SECTION  1 - BASIC SETUP
*******/
/*****************                                                 **************/
/*****************************************************************
*****************************/

  /*************************    The type of multicopter
*****************************/
    //#define GIMBAL
    //#define BI
    //#define TRI
    //#define QUADP
    //#define QUADX
    //#define Y4
    //#define Y6
```

```
  //#define HEX6
  //#define HEX6X
  //#define HEX6H  // New Model
  //#define OCTOX8
  //#define OCTOFLATP
  //#define OCTOFLATX
  //#define FLYING_WING
  //#define VTAIL4
  //#define AIRPLANE
  //#define SINGLECOPTER
  //#define DUALCOPTER
  //#define HELI_120_CCPM
  //#define HELI_90_DEG

 /*************************    Motor minthrottle
*******************************/
  /* Set the minimum throttle command sent to the ESC (Electronic Speed
Controller)
     This is the minimum value that allow motors to run at a idle speed  */
  //#define MINTHROTTLE 1300 // for Turnigy Plush ESCs 10A
  //#define MINTHROTTLE 1120 // for Super Simple ESCs 10A
  //#define MINTHROTTLE 1064 // special ESC (simonk)
  //#define MINTHROTTLE 1050 // for brushed ESCs like ladybird
  #define MINTHROTTLE 1150 // (*) (**)

 /*************************    Motor maxthrottle
*******************************/
  /* this is the maximum value for the ESCs at full power, this value can be increased
up to 2000 */
  #define MAXTHROTTLE 1850

 /*************************    Mincommand
*******************************/
  /* this is the value for the ESCs when they are not armed
     in some cases, this value must be lowered down to 900 for some specific ESCs,
otherwise they failed to initiate */
  #define MINCOMMAND  1000

 /*********************************  I2C speed for old WMP config (useless
config for other sensors)  ************/
  #define I2C_SPEED 100000L     //100kHz normal mode, this value must be used
for a genuine WMP
  //#define I2C_SPEED 400000L   //400kHz fast mode, it works only with some
WMP clones

 /*************************    Internal i2c Pullups
*******************************/
  /* enable internal I2C pull ups (in most cases it is better to use external pullups) */
  //#define INTERNAL_I2C_PULLUPS
```

```
 /************************************  constant loop time
*****************************/
  #define LOOP_TIME 2800


/*********************************************************************
*****************/
 /*****************         boards and sensor definitions
*****************/

/*********************************************************************
*****************/

  /***************************    Combined IMU Boards
*********************************/
    /* if you use a specific sensor board:
       please submit any correction to this list.
        Note from Alex: I only own some boards, for other boards, I'm not sure, the
info was gathered via rc forums, be cautious */
    //#define FFIMUv1         // first 9DOF+baro board from Jussi, with HMC5843
<- confirmed by Alex
    //#define FFIMUv2         // second version of 9DOF+baro board from Jussi, with
HMC5883       <- confirmed by Alex
    //#define FREEIMUv1       // v0.1 & v0.2 & v0.3 version of 9DOF board from
Fabio
    //#define FREEIMUv03      // FreeIMU v0.3 and v0.3.1
    //#define FREEIMUv035     // FreeIMU v0.3.5 no baro
    //#define FREEIMUv035_MS  // FreeIMU v0.3.5_MS
<- confirmed by Alex
    //#define FREEIMUv035_BMP // FreeIMU v0.3.5_BMP
    //#define FREEIMUv04      // FreeIMU v0.4 with MPU6050, HMC5883L,
MS561101BA                <- confirmed by Alex
    //#define FREEIMUv043     // same as FREEIMUv04 with final MPU6050 (with
the right ACC scale)
    //#define NANOWII         // the smallest multiwii FC based on MPU6050 + pro
micro based proc <- confirmed by Alex
    //#define PIPO            // 9DOF board from erazz
    //#define QUADRINO        // full FC board 9DOF+baro board from witespy  with
BMP085 baro     <- confirmed by Alex
    //#define QUADRINO_ZOOM   // full FC board 9DOF+baro board from witespy
second edition
    //#define QUADRINO_ZOOM_MS// full FC board 9DOF+baro board from
witespy  second edition       <- confirmed by Alex
    //#define ALLINONE        // full FC board or standalone 9DOF+baro board from
CSG_EU
    //#define AEROQUADSHIELDv2
    //#define ATAVRSBIN1      // Atmel 9DOF (Contribution by EOSBandi). requires
3.3V power.
    //#define SIRIUS          // Sirius Navigator IMU                                  <-
confirmed by Alex
```

```
    //#define SIRIUSGPS       // Sirius Navigator IMU  using external MAG on GPS
board           <- confirmed by Alex
    //#define SIRIUS600       // Sirius Navigator IMU  using the WMP for the gyro
    //#define SIRIUS_AIR      // Sirius Navigator IMU 6050 32U4 from
MultiWiiCopter.com          <- confirmed by Alex
    //#define SIRIUS_AIR_GPS  // Sirius Navigator IMU 6050 32U4 from
MultiWiiCopter.com with GPS/MAG remote located
    //#define SIRIUS_MEGAv5_OSD //  Paris_Sirius™
ITG3050,BMA280,MS5611,HMC5883,uBlox  http://www.Multiwiicopter.com <-
confirmed by Alex
    //#define MINIWII         // Jussi's MiniWii Flight Controller                    <-
confirmed by Alex
    //#define MICROWII        // MicroWii 10DOF with ATmega32u4, MPU6050,
HMC5883L, MS561101BA from http://flyduino.net/
    //#define CITRUSv2_1      // CITRUS from qcrc.ca
    //#define CHERRY6DOFv1_0
    //#define DROTEK_10DOF    // Drotek 10DOF with ITG3200, BMA180,
HMC5883, BMP085, w or w/o LLC
    //#define DROTEK_10DOF_MS // Drotek 10DOF with ITG3200, BMA180,
HMC5883, MS5611, LLC
    //#define DROTEK_6DOFv2   // Drotek 6DOF v2
    //#define DROTEK_6DOF_MPU // Drotek 6DOF with MPU6050
    //#define DROTEK_10DOF_MPU//
    //#define MONGOOSE1_0     // mongoose 1.0    http://store.ckdevices.com/
    //#define CRIUS_LITE      // Crius MultiWii Lite
    //#define CRIUS_SE        // Crius MultiWii SE
    //#define CRIUS_SE_v2_0   // Crius MultiWii SE 2.0 with MPU6050, HMC5883
and BMP085
    //#define OPENLRSv2MULTI  // OpenLRS v2 Multi Rc Receiver board including
ITG3205 and ADXL345
    //#define BOARD_PROTO_1   // with MPU6050 + HMC5883L + MS baro
    //#define BOARD_PROTO_2   // with MPU6050 + slave  MAG3110 + MS baro
    //#define GY_80           // Chinese 10 DOF with  L3G4200D ADXL345
HMC5883L BMP085, LLC
    //#define GY_85           // Chinese 9 DOF with  ITG3205 ADXL345 HMC5883L
LLC
    //#define GY_86           // Chinese 10 DOF with  MPU6050 HMC5883L MS5611,
LLC
    //#define GY_88 // Chinese 10 DOF with MPU6050 HMC5883L BMP085, LLC
    //#define GY_521          // Chinese 6  DOF with  MPU6050, LLC
    //#define INNOVWORKS_10DOF // with ITG3200, BMA180, HMC5883,
BMP085 available here http://www.diymulticopter.com
    //#define INNOVWORKS_6DOF // with ITG3200, BMA180 available here
http://www.diymulticopter.com
    //#define MultiWiiMega    // MEGA + MPU6050+HMC5883L+MS5611 available
here http://www.diymulticopter.com
    //#define PROTO_DIY       // 10DOF mega board
    //#define IOI_MINI_MULTIWII// www.bambucopter.com
    //#define Bobs_6DOF_V1     // BobsQuads 6DOF V1 with ITG3200 & BMA180
    //#define Bobs_9DOF_V1     // BobsQuads 9DOF V1 with ITG3200, BMA180 &
```

```
HMC5883L
    //#define Bobs_10DOF_BMP_V1 // BobsQuads 10DOF V1 with ITG3200,
BMA180, HMC5883L & BMP180 - BMP180 is software compatible with BMP085
    //#define FLYDUINO_MPU       // MPU6050 Break Out onboard 3.3V reg
    //#define CRIUS_AIO_PRO
    //#define DESQUARED6DOFV2GO  // DEsquared V2 with ITG3200 only
    //#define DESQUARED6DOFV4    // DEsquared V4 with MPU6050
    //#define LADYBIRD
    //#define MEGAWAP_V2_STD     // available here: http://www.multircshop.com
<- confirmed by Alex
    //#define MEGAWAP_V2_ADV
    //#define HK_MultiWii_SE_V2  // Hobbyking board with MPU6050 +
HMC5883L + BMP085
    //#define HK_MultiWii_328P   // Also labeled "Hobbybro" on the back.  ITG3205
+ BMA180 + BMP085 + NMC5583L + DSM2 Connector (Spektrum Satellite)
    //#define RCNet_FC           // RCNet FC with MPU6050 and MS561101BA
http://www.rcnet.com
    //#define RCNet_FC_GPS       // RCNet FC with MPU6050 + MS561101BA +
HMC5883L + UBLOX GPS http://www.rcnet.com
    //#define FLYDU_ULTRA        // MEGA+10DOF+MT3339 FC
    //#define DIYFLYING_MAGE_V1  // diyflying 10DOF mega board with
MPU6050 + HMC5883L + BMP085 http://www.indoor-flying.hk
    //#define MultiWii_32U4_SE         // Hextronik MultiWii_32U4_SE
    //#define MultiWii_32U4_SE_no_baro // Hextronik MultiWii_32U4_SE without
the MS561101BA to free flash-memory for other functions
    //#define Flyduino9DOF       // Flyduino 9DOF IMU MPU6050+HMC5883l
    //#define Nano_Plane         // Multiwii Plane version with tail-front LSM330
sensor http://www.radiosait.ru/en/page_5324.html

  /**************************    independent sensors
*********************************/
    /* leave it commented if you already checked a specific board above */
    /* I2C gyroscope */
    //#define WMP
    //#define ITG3050
    //#define ITG3200
    //#define MPU3050
    //#define L3G4200D
    //#define MPU6050      //combo + ACC
    //#define LSM330       //combo + ACC

    /* I2C accelerometer */
    //#define MMA7455
    //#define ADXL345
    //#define BMA020
    //#define BMA180
    //#define BMA280
    //#define LIS3LV02
    //#define LSM303DLx_ACC
    //#define MMA8451Q
```

```
    /* I2C barometer */
    //#define BMP085
    //#define MS561101BA

    /* I2C magnetometer */
    //#define HMC5843
    //#define HMC5883
    //#define AK8975
    //#define MAG3110

    /* Sonar */ // for visualization purpose currently - no control code behind
    //#define SRF02 // use the Devantech SRF i2c sensors
    //#define SRF08
    //#define SRF10
    //#define SRF23

    /* ADC accelerometer */ // for 5DOF from sparkfun, uses analog PIN A1/A2/A3
    //#define ADCACC

    /* enforce your individual sensor orientation - even overrides board specific
defaults */
    //#define FORCE_ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  =  Y;
imu.accADC[PITCH]  = -X; imu.accADC[YAW]  = Z;}
    //#define FORCE_GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = -Y;
imu.gyroADC[PITCH] =  X; imu.gyroADC[YAW] = Z;}
    //#define FORCE_MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = Z;}

    /* Board orientation shift */
    /* If you have frame designed only for + mode and you cannot rotate FC
phisycally for flying in X mode (or vice versa)
     * you can use one of of this options for virtual sensors rotation by 45 deegres,
then set type of multicopter according to flight mode.
     * Check motors order and directions of motors rotation for matching with new
front point!  Uncomment only one option! */
    //#define SENSORS_TILT_45DEG_RIGHT       // rotate the FRONT 45 degres
clockwise
    //#define SENSORS_TILT_45DEG_LEFT        // rotate the FRONT 45 degres
counterclockwise


/*************************************************************************
*****************************/
/*****************                                              **************/
/***************  SECTION  2 - COPTER TYPE SPECIFIC OPTIONS
*******/
/*****************                                              **************/
/*************************************************************************
*****************************/
```

```
  /**********************************  PID Controller
***********************************/
   /* choose one of the alternate PID control algorithms
    * 1 = evolved oldschool algorithm (similar to v2.2)
    * 2 = new experimental algorithm from Alex Khoroshko - unsupported -
http://www.multiwii.com/forum/viewtopic.php?f=8&t=3671&start=10#p37387
    * */
   #define PID_CONTROLLER 1

   /* NEW: not used anymore for servo coptertypes  <== NEEDS FIXING - MOVE
TO WIKI */
   #define YAW_DIRECTION 1
   //#define YAW_DIRECTION -1 // if you want to reverse the yaw correction
direction

   #define ONLYARMWHENFLAT //prevent the copter from arming when the copter
is tilted

  /**********************************    ARM/DISARM
***********************************/
  /* optionally disable stick combinations to arm/disarm the motors.
    * In most cases one of the two options to arm/disarm via TX stick is sufficient */
   #define ALLOW_ARM_DISARM_VIA_TX_YAW
   //#define ALLOW_ARM_DISARM_VIA_TX_ROLL

   /**********************************    SERVOS
***********************************/
   /* info on which servos connect where and how to setup can be found here
    * http://www.multiwii.com/wiki/index.php?title=Config.h#Servos_configuration
    */

   /* Do not move servos if copter is unarmed
    * It is a quick hack to overcome feedback tail wigglight when copter has a flexibile
    * landing gear
   */
   //#define DISABLE_SERVOS_WHEN_UNARMED


   /* if you want to preset min/middle/max values for servos right after flashing,
because of limited physical
    * room for servo travel, then you must enable and set all three following options */
    //#define SERVO_MIN  {1020, 1020, 1020, 1020, 1020, 1020, 1020, 1020}
    //#define  SERVO_MAX {2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000}
    //#define  SERVO_MID {1500, 1500, 1500, 1500, 1500, 1500, 1500, 1500} // (*)
    //#define FORCE_SERVO_RATES      {30,30,100,100,100,100,100,100} // 0 =
normal, 1= reverse

  /**********************          Cam Stabilisation
***********************/
   /* The following lines apply only for a pitch/roll tilt stabilization system.
```

```
Uncomment the first or second line to activate it */
   //#define SERVO_MIX_TILT
   //#define SERVO_TILT

   /* camera trigger function : activated via Rc Options in the GUI, servo output=A2
on promini */
   // trigger interval can be changed via (*GUI*) or via AUX channel
   //#define CAMTRIG
   #define CAM_TIME_HIGH 1000   // the duration of HIGH state servo expressed
in ms

  /*********************          Airplane
**********************/
   //#define USE_THROTTLESERVO // For use of standard 50Hz servo on throttle.

   //#define FLAPPERONS    AUX4          // Mix Flaps with Aileroins.
   #define FLAPPERON_EP   { 1500, 1700 } // Endpooints for flaps on a 2 way
switch else set {1020,2000} and program in radio.
   #define FLAPPERON_INVERT { -1, 1 }    // Change direction om flapperons {
Wing1, Wing2 }

   //#define FLAPS                    // Traditional Flaps on SERVO3.
   //#define FLAPSPEED     3           // Make flaps move slowm Higher value is
Higher Speed.

  /*********************      Common for Heli & Airplane
**********************/

   /* Governor: attempts to maintain rpm through pitch and voltage changes
    * predictive approach: observe input signals and voltage and guess appropriate
corrections.
    * (the throttle curve must leave room for the governor, so 0-50-75-80-80 is ok, 0-
50-95-100-100 is _not_ ok.
    * Can be toggled via aux switch.
    */
   //#define GOVERNOR_P 7     // (*) proportional factor. Higher value -> higher
throttle increase. Must be >=1; 0 = turn off
   //#define GOVERNOR_D 4     // (*) decay timing. Higher value -> takes longer to
return throttle to normal. Must be >=1;

   /* tail precomp from collective */
   #define YAW_COLL_PRECOMP 10          // (*) proportional factor in 0.1. Higher
value -> higher precomp effect. value of 10 equals no/neutral effect
   #define YAW_COLL_PRECOMP_DEADBAND 120 // (*) deadband for collective
pitch input signal around 0-pitch input value

   //#define VOLTAGEDROP_COMPENSATION // voltage impact correction

  /*********************          Heli
**********************/
```

```
  /* Channel to control CollectivePitch */
  #define COLLECTIVE_PITCH      THROTTLE

  /* Limit the range of Collective Pitch. 100% is Full Range each way and position
for Zero Pitch */
  #define COLLECTIVE_RANGE { 80, 0, 80 }// {Min%, ZeroPitch offset from
1500, Max%}.
  #define YAWMOTOR                0       // If a motor is used as YAW Set to 1 else set
to 0.

  /* Servo mixing for heli 120
                {Coll,Nick,Roll} */
  #define SERVO_NICK   { +10, -10,  0 }
  #define SERVO_LEFT   { +10, +5, +10 }
  #define SERVO_RIGHT  { +10, +5, -10 }

  /* Limit Maximum controll for Roll & Nick  in 0-100% */
  #define CONTROL_RANGE   { 100, 100 }      //  { ROLL,PITCH }

  /* use servo code to drive the throttle output. You want this for analog servo driving
the throttle on IC engines.
     if inactive, throttle output will be treated as a motor output, so it can drive an ESC
*/
  //#define HELI_USE_SERVO_FOR_THROTTLE

 /************************      your individual mixing
***********************/
  /* if you want to override an existing entry in the mixing table, you may want to
avoid editing the
   * mixTable() function for every version again and again.
   * howto:
http://www.multiwii.com/wiki/index.php?title=Config.h#Individual_Mixing
   */
  //#define MY_PRIVATE_MIXING "filename.h"

 /************************      your individual defaults
***********************/
  /* if you want to replace the hardcoded default values with your own (e.g. from a
previous save to an .mwi file),
   * you may want to avoid editing the LoadDefaults() function for every version
again and again.
   * http://www.multiwii.com/wiki/index.php?title=Config.h#Individual_defaults
   */
  //#define MY_PRIVATE_DEFAULTS "filename.h"


/*****************************************************************************
*****************************/
/*****************                                                  **************/
/***************  SECTION  3 - RC SYSTEM SETUP
```

```
*******/
/*****************                                                    **************/
/*************************************************************************
****************************/

  /* note: no need to uncomment something in this section if you use a standard
receiver */

/****************************    EXTENDED AUX STATES
**********************************/
/* If you uncomment this line, you can use six states for each of the aux channels
(AUX1-AUX4)
to control your copter.
Channel values
1000-1230
1231-1360
1361-1490
1491-1620
1621-1749
1750-

At this moment you can use this function only with WinGUI 2.3 release.
MultiWiiConf does not support it yet
*/

//#define EXTENDED_AUX_STATES



/*************************************************************************
******************/
  /********                special receiver types          ********************/

/*************************************************************************
******************/

   /****************************    PPM Sum Reciver
**********************************/
     /* The following lines apply only for specific receiver with only one PPM sum
signal, on digital PIN 2
       Select the right line depending on your radio brand. Feel free to modify the
order in your PPM order is different */
     //#define SERIAL_SUM_PPM
PITCH,YAW,THROTTLE,ROLL,AUX1,AUX2,AUX3,AUX4,8,9,10,11 //For
Graupner/Spektrum
     //#define SERIAL_SUM_PPM
ROLL,PITCH,THROTTLE,YAW,AUX1,AUX2,AUX3,AUX4,8,9,10,11 //For
Robe/Hitec/Futaba
     //#define SERIAL_SUM_PPM
ROLL,PITCH,YAW,THROTTLE,AUX1,AUX2,AUX3,AUX4,8,9,10,11 //For
```

```
Multiplex
    //#define SERIAL_SUM_PPM
PITCH,ROLL,THROTTLE,YAW,AUX1,AUX2,AUX3,AUX4,8,9,10,11 //For some
Hitec/Sanwa/Others

    // Uncommenting following line allow to connect PPM_SUM receiver to standard
THROTTLE PIN on MEGA boards (eg. A8 in CRIUS AIO)
    //#define PPM_ON_THROTTLE

  /*********************    Spektrum Satellite Reciver
*******************************/
    /* The following lines apply only for Spektrum Satellite Receiver
       Spektrum Satellites are 3V devices.  DO NOT connect to 5V!
       For MEGA boards, attach sat grey wire to RX1, pin 19. Sat black wire to
ground. Sat orange wire to Mega board's 3.3V (or any other 3V to 3.3V source).
       For PROMINI, attach sat grey to RX0.  Attach sat black to ground. */
    //#define SPEKTRUM 1024
    //#define SPEKTRUM 2048
    //#define RX_SERIAL_PORT 1    // Forced to 0 on Pro Mini and single serial
boards; Set to your choice of 0, 1, or 2 on any Mega based board (defaults to 1 on
Mega).
    //**************************
    // Defines that allow a "Bind" of a Spektrum or Compatible Remote Receiver (aka
Satellite) via Configuration GUI.
    //   Bind mode will be same as declared above, if your TX is capable.
    //   Ground, Power, and Signal must come from three adjacent pins.
    //   By default, these are Ground=4, Power=5, Signal=6.  These pins are in a row
on most MultiWii shield boards. Pins can be overriden below.
    //   Normally use 3.3V regulator is needed on the power pin!!  If your satellite
hangs during bind (blinks, but won't complete bind with a solid light), go direct 5V on
all pins.
    //**************************
    //   For Pro Mini, the connector for the Satellite that resides on the FTDI can be
unplugged and moved to these three adjacent pins.
    //#define SPEK_BIND           //Un-Comment for Spektrum Satellie Bind
Support.  Code is ~420 bytes smaller without it.
    //#define SPEK_BIND_GROUND 4
    //#define SPEK_BIND_POWER  5
    //#define SPEK_BIND_DATA   6

  /*********************************    SBUS RECIVER
***********************************/
    /* The following line apply only for Futaba S-Bus Receiver on MEGA boards or
PROMICRO boards.
       You have to invert the S-Bus-Serial Signal e.g. with a Hex-Inverter like IC SN74
LS 04 */
    //#define SBUS
PITCH,YAW,THROTTLE,ROLL,AUX1,AUX2,AUX3,AUX4,8,9,10,11,12,13,14,15,
16,17  // dsm2 orangerx
    //#define SBUS
```

```
ROLL,PITCH,THROTTLE,YAW,AUX1,AUX2,AUX3,AUX4,8,9,10,11,12,13,14,15,
16,17  // T14SG
    //#define RX_SERIAL_PORT 1
    #define SBUS_MID_OFFSET 988 //SBUS Mid-Point at 1500

  /****************************** HOTT RECIVER
************************************/
  /* Graupner Hott HD */
  //#define SUMD PITCH,YAW,THROTTLE,ROLL,AUX1,AUX2,AUX3,AUX4
  //#define RX_SERIAL_PORT 1

/**********************************************************************
*****************************/
/*****************                                          **************/
/****************  SECTION  4 - ALTERNATE CPUs & BOARDS
*******/
/*****************                                          **************/
/**********************************************************************
*****************************/


/**********************************************************************
******************/
 /********              Promini Specifig Settings         ********************/

/**********************************************************************
******************/

  /**************************   Hexa Motor 5 & 6 Pins
*******************************/
    /* PIN A0 and A1 instead of PIN D5 & D6 for 6 motors config and promini config
      This mod allow the use of a standard receiver on a pro mini
      (no need to use a PPM sum receiver) */
    //#define A0_A1_PIN_HEX

  /*********************************   Aux 2 Pin
*********************************/
    /* possibility to use PIN8 or PIN12 as the AUX2 RC input (only one, not both)
      it deactivates in this case the POWER PIN (pin 12) or the BUZZER PIN (pin 8)
*/
    //#define RCAUXPIN8
    //#define RCAUXPIN12



/**********************************************************************
******************/
 /*****************          Teensy 2.0 Support              *****************/

/**********************************************************************
```

```
*****************/
   /* uncomment this if you use a teensy 2.0 with teensyduino
      it needs to run at 16MHz */
   //#define TEENSY20



/************************************************************************
*****************/
  /********   Settings for ProMicro, Leonardo and other Atmega32u4 Boards
***********/

/************************************************************************
*****************/

   /*********************************    pin Layout
*********************************/
     /* activate this for a better pinlayout if all pins can be used => not possible on
ProMicro */
     //#define A32U4ALLPINS

   /*********************************    PWM Setup
*********************************/
     /* activate all 6 hardware PWM outputs Motor 5 = D11 and 6 = D13.
        note: not possible on the sparkfun promicro (pin 11 & 13 are not broken out
there)
        if activated:
        Motor 1-6 = 10-bit hardware PWM
        Motor 7-8 = 8-bit Software PWM
        Servos    = 8-bit Software PWM
        if deactivated:
        Motor 1-4 = 10-bit hardware PWM
        Motor 5-8 = 10-bit Software PWM
        Servos    = 10-bit Software PWM */
     //#define HWPWM6

   /*********************************    Aux 2 Pin
*********************************/
     /* AUX2 pin on pin RXO */
     //#define RCAUX2PINRXO

     /* aux2 pin on pin D17 (RXLED) */
     //#define RCAUX2PIND17

   /*********************************    Buzzer Pin
*********************************/
     /* this moves the Buzzer pin from TXO to D8 for use with ppm sum or spectrum
sat. RX (not needed if A32U4ALLPINS is active) */
     //#define D8BUZZER
```

```
   /***********************      Promicro version related
****************************/
    /* Inverted status LED for Promicro ver 10 */
    //#define PROMICRO10



/*********************************************************************
******************/
 /********                override default pin assignments    *******************/

/*********************************************************************
******************/

 /* only enable any of this if you must change the default pin assignment, e.g. your
board does not have a specific pin */
 /* you may need to change PINx and PORTx plus #shift according to the desired
pin! */
 //#define OVERRIDE_V_BATPIN               A0 // instead of A3    // Analog PIN 3

 //#define OVERRIDE_PSENSORPIN             A1 // instead of A2    // Analog PIN
2

 //#define OVERRIDE_LEDPIN_PINMODE          pinMode (A1, OUTPUT); // use
A1 instead of d13
 //#define OVERRIDE_LEDPIN_TOGGLE           PINC |= 1<<1; // PINB |= 1<<5;
//switch LEDPIN state (digital PIN 13)
 //#define OVERRIDE_LEDPIN_OFF              PORTC &= ~(1<<1); // PORTB &=
~(1<<5);
 //#define OVERRIDE_LEDPIN_ON               PORTC |= 1<<1;    // was PORTB |=
(1<<5);

 //#define OVERRIDE_BUZZERPIN_PINMODE       pinMode (A2, OUTPUT); //
use A2 instead of d8
 //#define OVERRIDE_BUZZERPIN_ON            PORTC |= 1<<2 //PORTB |= 1;
 //#define OVERRIDE_BUZZERPIN_OFF           PORTC &= ~(1<<2); //PORTB
&= ~1;

/*********************************************************************
*****************************/
/*****************                                          ***************/
/***************  SECTION  5 - ALTERNATE SETUP
*******/
/*****************                                          ***************/
/*********************************************************************
*****************************/

 /******             Serial com speed    *********************************/
   /* This is the speed of the serial interfaces */
   #define SERIAL0_COM_SPEED 115200
```

```
  #define SERIAL1_COM_SPEED 115200
  #define SERIAL2_COM_SPEED 115200
  #define SERIAL3_COM_SPEED 115200

  /* when there is an error on I2C bus, we neutralize the values during a short time.
expressed in microseconds
     it is relevent only for a conf with at least a WMP */
  #define NEUTRALIZE_DELAY 100000


/**********************************************************************
******************/
 /********                    Gyro filters              *******************/

/**********************************************************************
******************/

  /*********************  Lowpass filter for some gyros
*****************************/
    /* ITG3200 & ITG3205 Low pass filter setting. In case you cannot eliminate all
vibrations to the Gyro, you can try
       to decrease the LPF frequency, only one step per try. As soon as twitching gone,
stick with that setting.
       It will not help on feedback wobbles, so change only when copter is randomly
twiching and all dampening and
       balancing options ran out. Uncomment only one option!
       IMPORTANT! Change low pass filter setting changes PID behaviour, so retune
your PID's after changing LPF.
       available for ITG3050, ITG3200, MPU3050, MPU6050*/
    //#define GYRO_LPF_256HZ     // This is the default setting, no need to
uncomment, just for reference
    //#define GYRO_LPF_188HZ
    //#define GYRO_LPF_98HZ
    //#define GYRO_LPF_42HZ
    //#define GYRO_LPF_20HZ
    //#define GYRO_LPF_10HZ
    //#define GYRO_LPF_5HZ       // Use this only in extreme cases, rather change
motors and/or props -- setting not available on ITG3200

  /******              Gyro smoothing    *********************************/
    /* GYRO_SMOOTHING. In case you cannot reduce vibrations _and_ _after_ you
have tried the low pass filter options, you
       may try this gyro smoothing via averaging. Not suitable for multicopters!
       Good results for helicopter, airplanes and flying wings (foamies) with lots of
vibrations.*/
    //#define GYRO_SMOOTHING {20, 20, 3}    // (*) separate averaging ranges for
roll, pitch, yaw

  /************************  Moving Average Gyros
*********************************/
```

```
    //#define MMGYRO 10                  // (*) Active Moving Average Function for
Gyros
    //#define MMGYROVECTORLENGTH 15      // Length of Moving Average
Vector (maximum value for tunable MMGYRO
    /* Moving Average ServoGimbal Signal Output */
    //#define MMSERVOGIMBAL              // Active Output Moving Average
Function for Servos Gimbal
    //#define MMSERVOGIMBALVECTORLENGHT 32   // Lenght of Moving
Average Vector

 /*********************    Analog Reads
**********************************/
  /* if you want faster analog Reads, enable this. It may result in less accurate results,
especially for more than one analog channel */
  //#define FASTER_ANALOG_READS

/*****************************************************************************
*****************************/
/*****************                                                 **************/
/***************  SECTION  6 - OPTIONAL FEATURES
*******/
/*****************                                                 **************/
/*****************************************************************************
*****************************/

 /************************       Reset Baro altitude on arm
********************/
 /* When unchecked a calibration of the baro altitude is preformed every time arming
is activated */
 //#define ALTITUDE_RESET_ON_ARM

 /************************       Angele throttle correction
********************/
 /* Automatically increase throttle based on the angle of the copter
    Original idea by Kraut Rob, first implementation HAdrian */

 //#define THROTTLE_ANGLE_CORRECTION 40

 /*** HEADFREE : the copter can be controled by an absolute stick orientation,
whatever the yaw orientation ***/
 //#define HEADFREE

/************************       Advanced Headfree Mode
********************/
/* In Advanced Headfree mode when the copter is farther than
ADV_HEADFREE_RANGE meters then
   the  bearing between home and copter position will become the control direction
   IF copter come closer than ADV_HEADFREE_RANGE meters, then the control
direction freezed to the
   bearing between home and copter at the point where it crosses the
```

```
ADV_HEADFREE_RANGE meter distance
   first implementation by HAdrian, mods by EOSBandi
 */

  //#define ADVANCED_HEADFREE      //Advanced headfree mode is enabled
when this is uncommented
  //#define ADV_HEADFREE_RANGE 15  //Range where advanced headfree mode
activated


 /************************        continuous gyro calibration
*******************/
 /* Gyrocalibration will be repeated if copter is moving during calibration. */
   //#define GYROCALIBRATIONFAILSAFE

 /************************        AP FlightMode
*********************************/
 /*** FUNCTIONALITY TEMPORARY REMOVED ***/
   /* Temporarily Disables GPS_HOLD_MODE to be able to make it possible to
adjust the Hold-position when moving the sticks.*/
   //#define AP_MODE 40  // Create a deadspan for GPS.

 /************************   Assisted AcroTrainer
***********************************/
   /* Train Acro with auto recovery. Value set the point where ANGLE_MODE takes
over.
      Remember to activate ANGLE_MODE first!...
      A Value on 200 will give a very distinct transfer */
   //#define ACROTRAINER_MODE 200   //
http://www.multiwii.com/forum/viewtopic.php?f=16&t=1944#p17437


 /********                    Failsafe settings                ********************/
   /* Failsafe check pulses on four main control channels CH1-CH4. If the pulse is
missing or bellow 985us (on any of these four channels)
      the failsafe procedure is initiated. After FAILSAFE_DELAY time from failsafe
detection, the level mode is on (if ACC is avaliable),
      PITCH, ROLL and YAW is centered and THROTTLE is set to
FAILSAFE_THROTTLE value. You must set this value to descending about 1m/s or
so
      for best results. This value is depended from your configuration, AUW and some
other params.  Next, after FAILSAFE_OFF_DELAY the copter is disarmed,
      and motors is stopped. If RC pulse coming back before reached
FAILSAFE_OFF_DELAY time, after the small quard time the RC control is returned
to normal. */
   //#define FAILSAFE                          // uncomment  to activate the failsafe
function
   #define FAILSAFE_DELAY     10                  // Guard time for failsafe activation
after signal lost. 1 step = 0.1sec - 1sec in example
   #define FAILSAFE_OFF_DELAY 200                 // Time for Landing before
```

```
motors stop in 0.1sec. 1 step = 0.1sec - 20sec in example
   #define FAILSAFE_THROTTLE  (MINTHROTTLE + 200)    // (*) Throttle level
used for landing - may be relative to MINTHROTTLE - as in this case

   #define FAILSAFE_DETECT_TRESHOLD  985


  /*****************              DFRobot LED RING
*********************************/
   /* I2C DFRobot LED RING communication */
   //#define LED_RING

  /*******************************    LED FLASHER
***********************************/
   //#define LED_FLASHER
   //#define LED_FLASHER_DDR DDRB
   //#define LED_FLASHER_PORT PORTB
   //#define LED_FLASHER_BIT PORTB4
   //#define LED_FLASHER_INVERT
   //#define LED_FLASHER_SEQUENCE       0b00000000      // leds OFF
   //#define LED_FLASHER_SEQUENCE_ARMED  0b00000101      // create
double flashes
   //#define LED_FLASHER_SEQUENCE_MAX    0b11111111      // full
illumination
   //#define LED_FLASHER_SEQUENCE_LOW    0b00000000      // no
illumination


  /*******************************    Landing lights
*********************************/
  /* Landing lights
     Use an output pin to control landing lights.
     They can be switched automatically when used in conjunction
     with altitude data from a sonar unit. */
   //#define LANDING_LIGHTS_DDR DDRC
   //#define LANDING_LIGHTS_PORT PORTC
   //#define LANDING_LIGHTS_BIT PORTC0
   //#define LANDING_LIGHTS_INVERT

   /* altitude above ground (in cm) as reported by sonar */
   //#define LANDING_LIGHTS_AUTO_ALTITUDE 50

   /* adopt the flasher pattern for landing light LEDs */
   //#define LANDING_LIGHTS_ADOPT_LED_FLASHER_PATTERN

  /*************************    INFLIGHT ACC Calibration
*****************************/
   /* This will activate the ACC-Inflight calibration if unchecked */
   //#define INFLIGHT_ACC_CALIBRATION
```

```
 /******************************    OSD Switch
************************************/
   // This adds a box that can be interpreted by OSD in activation status (to switch
on/off the overlay for instance)
 //#define OSD_SWITCH


/*************************************************************************
*****************/
 /***********************              TX-related
**************************/

/*************************************************************************
*****************/

   /* introduce a deadband around the stick center
      Must be greater than zero, comment if you dont want a deadband on roll, pitch
and yaw */
   //#define DEADBAND 6


/*************************************************************************
*****************/
 /***********************              GPS
**************************/

/*************************************************************************
*****************/

   /* ENable this for using GPS simulator (NMEA only)*/
   //#define GPS_SIMULATOR

   /* GPS using a SERIAL port
      if enabled, define here the Arduino Serial port number and the UART speed
      note: only the RX PIN is used in case of NMEA mode, the GPS is not configured
by multiwii
      in NMEA mode the GPS must be configured to output GGA and RMC NMEA
sentences (which is generally the default conf for most GPS devices)
      at least 5Hz update rate. uncomment the first line to select the GPS serial port of
the arduino */

   //#define GPS_SERIAL 2         // should be 2 for flyduino v2. It's the serial port
number on arduino MEGA
                           // must be 0 for PRO_MINI (ex GPS_PRO_MINI)
                           // note: Now a GPS can share MSP on the same port. The only
constrain is to not use it simultaneously, and use the same port speed.

   // avoid using 115200 baud because with 16MHz arduino the 115200 baudrate have
more than 2% speed error (57600 have 0.8% error)
   #define GPS_BAUD   57600       // GPS_BAUD will override
```

SERIALx_COM_SPEED for the selected port

```
  /* GPS protocol
     NMEA  - Standard NMEA protocol GGA, GSA and RMC  sentences are needed
     UBLOX - U-Blox binary protocol, use the ublox config file (u-blox-config.ublox.txt) from the source tree
     MTK_BINARY16 and MTK_BINARY19 - MTK3329 chipset based GPS with DIYDrones binary firmware (v1.6 or v1.9)
     With UBLOX and MTK_BINARY you don't have to use GPS_FILTERING in multiwii code !!! */


   //#define NMEA
   //#define UBLOX
   //#define MTK_BINARY16
   //#define MTK_BINARY19
   //#define INIT_MTK_GPS        // initialize MTK GPS for using selected speed, 5Hz update rate and GGA & RMC sentence or binary settings


   /* I2C GPS device made with an independant arduino + GPS device
      including some navigation functions
      contribution from EOSBandi   http://code.google.com/p/i2c-gps-nav/
      You have to use at least I2CGpsNav code r33 */
   /* all functionnalities allowed by SERIAL_GPS are now available for I2C_GPS: all relevant navigation computations are gathered in the main FC */

   //#define I2C_GPS

   // If your I2C GPS board has Sonar support enabled
   //#define I2C_GPS_SONAR

   /* indicate a valid GPS fix with at least 5 satellites by flashing the LED  - Modified by MIS - Using stable LED (YELLOW on CRIUS AIO) led work as sat number indicator
     - No GPS FIX -> LED blink at speed of incoming GPS frames
     - Fix and sat no. bellow 5 -> LED off
     - Fix and sat no. >= 5 -> LED blinks, one blink for 5 sat, two blinks for 6 sat, three for 7 ... */
   #define GPS_LED_INDICATOR

  //Enables the MSP_WP command set , which is used by WinGUI for displaying an setting up navigation
  //#define USE_MSP_WP

  // HOME position is reset at every arm, uncomment it to prohibit it (you can set home position with GyroCalibration)
  //#define DONT_RESET_HOME_AT_ARM

/* GPS navigation can control the heading */
```

```
// copter faces toward the navigation point, maghold must be enabled for it
#define NAV_CONTROLS_HEADING       1    //(**)
// true - copter comes in with tail first
#define NAV_TAIL_FIRST             0    //(**)
// true - when copter arrives to home position it rotates it's head to takeoff direction
#define NAV_SET_TAKEOFF_HEADING    1    //(**)

/* Get your magnetic declination from here : http://magnetic-declination.com/
Convert the degree+minutes into decimal degree by ==> degree+minutes*(1/60)
Note the sign on declination it could be negative or positive (WEST or EAST)
Also note, that maqgnetic declination changes with time, so recheck your value every
3-6 months */
#define MAG_DECLINATION  4.02f   //(**)

// Adds a forward predictive filterig to compensate gps lag. Code based on Jason
Short's lead filter implementation
#define GPS_LEAD_FILTER              //(**)

// add a 5 element moving average filter to GPS coordinates, helps eliminate gps noise
but adds latency comment out to disable
// use it with NMEA gps only
//#define GPS_FILTERING              //(**)

// if we are within this distance to a waypoint then we consider it reached (distance is
in cm)
#define GPS_WP_RADIUS              100      //(**)

// Safe WP distance, do not start mission if the first wp distance is larger than this
number (in meters)
// Also aborts mission if the next waypoint distance is more than this number
#define SAFE_WP_DISTANCE           500      //(**)

//Maximu allowable navigation altitude (in meters) automatic altitude control will not
go above this height
#define MAX_NAV_ALTITUDE           100     //(**)

// minimum speed when approach waypoint
#define NAV_SPEED_MIN              100    // cm/sec //(**)
// maximum speed to reach between waypoints
#define NAV_SPEED_MAX              400    // cm/sec //(**)
// Slow down to zero when reaching waypoint (same as NAV_SPEED_MIN = 0)
#define NAV_SLOW_NAV               0      //(**)
// Weight factor of the crosstrack error in navigation calculations (do not touch)
#define CROSSTRACK_GAIN            .4     //(**)
// Maximum allowable banking than navigation outputs
#define NAV_BANK_MAX 3000                 //(**)

//Defines the RTH altitude. 0 means keep current alt during RTH (in meters)
#define RTH_ALTITUDE               15       //(**)
```

```
//Wait to reach RTH alt before start moving to home (0-no, 1-yes)
#define WAIT_FOR_RTH_ALT            1        //(**)

//Navigation engine will takeover BARO mode control
#define NAV_TAKEOVER_BARO           1        //(**)

//Throttle stick input will be ignored  (only in BARO)
#define IGNORE_THROTTLE             1        //(**)

//If FENCE DISTANCE is larger than 0 then copter will switch to RTH when it
farther from home
//than the defined number in meters
#define FENCE_DISTANCE      600

//This governs the descent speed during landing. 100 is equals approc 50cm/sec
#define LAND_SPEED          100


   //#define ONLY_ALLOW_ARM_WITH_GPS_3DFIX      // Only allow FC arming
if GPS has a 3D fix.


/**********************************************************************
******************/
  /***********************        LCD/OLED - display settings
*********************/

/**********************************************************************
******************/

   /* http://www.multiwii.com/wiki/index.php?title=Extra_features#LCD_.2F_OLED
*/

   /*****************************   The type of LCD
*********************************/
     /* choice of LCD attached for configuration and telemetry, see notes below */
     //#define LCD_DUMMY       // No Physical LCD attached.  With this &
LCD_CONF defined, TX sticks still work to set gains, by watching LED blink.
     //#define LCD_SERIAL3W    // Alex' initial variant with 3 wires, using rx-pin for
transmission @9600 baud fixed
     //#define LCD_TEXTSTAR    // SERIAL LCD: Cat's Whisker LCD_TEXTSTAR
Module CW-LCD-02 (Which has 4 input keys for selecting menus)
     //#define LCD_VT100       // SERIAL LCD: vt100 compatible terminal emulation
(blueterm, putty, etc.)
     //#define LCD_TTY         // SERIAL LCD: useful to tweak parameters over cable
with arduino IDE 'serial monitor'
     //#define LCD_ETPP        // I2C LCD: Eagle Tree Power Panel LCD, which is i2c
(not serial)
     //#define LCD_LCD03       // I2C LCD: LCD03, which is i2c
     //#define LCD_LCD03S      // SERIAL LCD: LCD03 whit serial 9600 baud
```

```
comunication enabled.
     //#define OLED_I2C_128x64 // I2C LCD: OLED
http://www.multiwii.com/forum/viewtopic.php?f=7&t=1350
     //#define OLED_DIGOLE     // I2C OLED from
http://www.digole.com/index.php?productID=550

   /*****************************  Display settings
**********************************/
     #define LCD_SERIAL_PORT 0    // must be 0 on Pro Mini and single serial
boards; Set to your choice on any Mega based board

     //#define SUPPRESS_OLED_I2C_128x64LOGO  // suppress display of OLED
logo to save memory

   /* double font height for better readability. Reduces visible #lines by half.
    * The lower part of each page is accessible under the name of shifted keyboard
letter :
    * 1 - ! , 2 - @ , 3 - # , 4 - $ , 5 - % , 6 - ^ , 7 - & , 8 - * , 9 - (
    * You must add both to your lcd.telemetry.* sequences
    */
     //#define DISPLAY_FONT_DSIZE //currently only aplicable for
OLED_I2C_128x64 and OLED_DIGOLE

   /* style of display - AUTODETECTED via LCD_ setting - only activate to override
defaults */
     //#define DISPLAY_2LINES
     //#define DISPLAY_MULTILINE
     //#define MULTILINE_PRE 2  // multiline configMenu # pref lines
     //#define MULTILINE_POST 6 // multiline configMenu # post lines
     //#define DISPLAY_COLUMNS 16
   /*******************************    Navigation
**********************************/
   /* keys to navigate the LCD menu */
     #define LCD_MENU_PREV 'p'
     #define LCD_MENU_NEXT 'n'
     #define LCD_VALUE_UP 'u'
     #define LCD_VALUE_DOWN 'd'

     #define LCD_MENU_SAVE_EXIT 's'
     #define LCD_MENU_ABORT 'x'


/*************************************************************************
*****************/
  /***********************      LCD configuration menu
**************************/

/*************************************************************************
*****************/
```

```
   /* uncomment this line if you plan to use a LCD or OLED for tweaking parameters
    *
http://www.multiwii.com/wiki/index.php?title=Extra_features#Configuration_Menu
*/
     //#define LCD_CONF

   /* to include setting the aux switches for AUX1 -> AUX4 via LCD */
     //#define LCD_CONF_AUX

   /* optional exclude some functionality - uncomment to suppress unwanted aux
channel configuration options */
     //#define SUPPRESS_LCD_CONF_AUX2
     //#define SUPPRESS_LCD_CONF_AUX34


/**************************************************************************
******************/
  /***********************      LCD       telemetry
**************************/

/**************************************************************************
******************/

   /* to monitor system values (battery level, loop time etc. with LCD
    * http://www.multiwii.com/wiki/index.php?title=LCD_Telemetry */

   /*********************************    Activation
***********************************/
   //#define LCD_TELEMETRY

   /* to enable automatic hopping between a choice of telemetry pages uncomment
this. */
   //#define LCD_TELEMETRY_AUTO "123452679" // pages 1 to 9 in ascending
order
   //#define LCD_TELEMETRY_AUTO  "212232425262729" // strong emphasis on
page 2

   /* manual stepping sequence; first page of the sequence gets loaded at startup to
allow non-interactive display */
   //#define LCD_TELEMETRY_STEP "0123456789" // should contain a 0 to allow
switching off.

   /* optional exclude some functionality - uncomment to suppress some unwanted
telemetry pages */
   //#define SUPPRESS_TELEMETRY_PAGE_1
   //#define SUPPRESS_TELEMETRY_PAGE_2 // sensor readings
   //#define SUPPRESS_TELEMETRY_PAGE_3 // checkboxitems
   //#define SUPPRESS_TELEMETRY_PAGE_4 // rx inputs
   //#define SUPPRESS_TELEMETRY_PAGE_5 // servo&motor outputs
   //#define SUPPRESS_TELEMETRY_PAGE_6 // cells voltages
```

```
   //#define SUPPRESS_TELEMETRY_PAGE_7 // gps
   //#define SUPPRESS_TELEMETRY_PAGE_8 // alarms states
   //#define SUPPRESS_TELEMETRY_PAGE_9 // cycle & fails
   //#define SUPPRESS_TELEMETRY_PAGE_R // reset

   /* optional override default items for some telemetry pages - for complete list of
usable functions see LCD.h */
   //#define LCD_TELEMETRY_PAGE1 { output_V, output_mAh, }
   //#define LCD_TELEMETRY_PAGE2 { output_gyroX, output_gyroY,
output_accZ, }
   //#define LCD_TELEMETRY_PAGE9 { output_fails, output_annex,
output_debug0, output_debug3, }


/*************************************************************************/
 /****                    RSSI                    ****/

/*************************************************************************/
   //#define RX_RSSI
   //#define RX_RSSI_PIN A3
   //#define RX_RSSI_CHAN 8   //RSSI injection on selected channel (for PPM,
Olrs, SBUS, etc.) (Starts at 0)


/*************************************************************************/
 /****                    Buzzer                    ****/

/*************************************************************************/
   //#define BUZZER
   //#define RCOPTIONSBEEP        // uncomment this if you want the buzzer to beep
at any rcOptions change on channel Aux1 to Aux4
   //#define ARMEDTIMEWARNING 330  // (*) Trigger an alarm after a certain time
of being armed [s] to save you lipo (if your TX does not have a countdown)
   //#define PILOTLAMP           //Uncomment if you are using a X-Arcraft Pilot
Lamp


/*************************************************************************/
 /****        battery voltage monitoring                ****/

/*************************************************************************/
   /* for V BAT monitoring
      after the resistor divisor we should get [0V;5V]->[0;1023] on analog V_BATPIN
      with R1=33k and R2=51k
      vbat = [0;1023]*16/VBATSCALE
      must be associated with #define BUZZER ! */
   //#define VBAT            // uncomment this line to activate the vbat code
   #define VBATSCALE      131 // (*) (**) change this value if readed Battery
voltage is different than real voltage
   #define VBATNOMINAL    126 // 12,6V full battery nominal voltage - only used
```

```
for lcd.telemetry
   #define VBATLEVEL_WARN1 107 // (*) (**) 10,7V
   #define VBATLEVEL_WARN2  99 // (*) (**) 9.9V
   #define VBATLEVEL_CRIT   93 // (*) (**) 9.3V - critical condition: if vbat ever
goes below this value, permanent alarm is triggered
   #define NO_VBAT          16 // Avoid beeping without any battery
   #define VBAT_OFFSET       0 // offset in 0.1Volts, gets added to voltage value  -
useful for zener diodes

   /* for V BAT monitoring of individual cells
    * enable both VBAT and VBAT_CELLS
    */
   //#define VBAT_CELLS
   #define VBAT_CELLS_NUM 0 // set this to the number of cells you monitor via
analog pins
   #define VBAT_CELLS_PINS {A0, A1, A2, A3, A4, A5 } // set this to the sequence
of analog pins
   #define VBAT_CELLS_OFFSETS {0, 50, 83, 121, 149, 177 } // in 0.1 volts, gets
added to voltage value  - useful for zener diodes
   #define VBAT_CELLS_DIVS { 75, 122,  98, 18, 30, 37 } // divisor for
proportional part according to resistors - larger value here gives smaller voltage


/*********************************************************************/
 /****          powermeter (battery capacity monitoring)        ****/

/*********************************************************************/

   /* enable monitoring of the power consumption from battery (think of mAh)
      allows to set alarm value in GUI or via LCD
     Full description and howto here
http://www.multiwii.com/wiki/index.php?title=Powermeter
      Two options:
      1 - hard: - (uses hardware sensor, after configuration gives very good results)
      2 - soft: - (good results +-5% for plush and mystery ESCs @ 2S and 3S, not good
with SuperSimple ESC)    */
   //#define POWERMETER_SOFT
   //#define POWERMETER_HARD
   #define PSENSORNULL 510 /* (*) hard only: set to analogRead() value for zero
current; for I=0A my sensor
                           gives 1/2 Vss; that is approx 2.49Volt; */
   #define PINT2mA 132     /* (*) hard: one integer step on arduino analog translates
to mA (example 4.9 / 37 * 1000) ;
                           soft: use fictional value, start with 100.
                           for hard and soft: larger PINT2mA will get you larger value for
power (mAh equivalent) */
   //#define WATTS // compute and display the actual watts (=Volt*Ampere)
consumed - requires both POWERMETER_HARD and VBAT
```

```
/*********************************************************************/
 /****          altitude hold                                  ****/

/*********************************************************************/

   /* defines the neutral zone of throttle stick during altitude hold, default setting is
      +/-50 uncommend and change the value below if you want to change it. */
   #define ALT_HOLD_THROTTLE_NEUTRAL_ZONE    50
   //#define ALT_HOLD_THROTTLE_MIDPOINT        1500  // in us    - if
uncommented, this value is used in ALT_HOLD for throttle stick middle point instead
of initialThrottleHold parameter.


   /* uncomment to disable the altitude hold feature.
    * This is useful if all of the following apply
    * + you have a baro
    * + want altitude readout and/or variometer
    * + do not use altitude hold feature
    * + want to save memory space */
   //#define SUPPRESS_BARO_ALTHOLD


/*********************************************************************/
 /****          altitude variometer                            ****/

/*********************************************************************/

   /* enable to get audio feedback upon rising/falling copter/plane.
    * Requires a working baro.
    * For now, Output gets sent to an enabled vt100 terminal program over the serial
line.
    * choice of two methods (enable either one or both)
    * method 1 : use short term movement from baro ( bigger code size)
    * method 2 : use long term observation of altitude from baro (smaller code size)
    */
   //#define VARIOMETER 12           // possible values: 12 = methods 1 & 2 ; 1 =
method 1 ; 2 = method 2
   //#define SUPPRESS_VARIOMETER_UP   // if no signaling for up movement is
desired
   //#define SUPPRESS_VARIOMETER_DOWN // if no signaling for down
movement is desired
   //#define VARIOMETER_SINGLE_TONE   // use only one tone (BEL);
neccessary for non-patched vt100 terminals


/*********************************************************************/
 /****          board naming                                   ****/

/*********************************************************************/
```

```
  /*
   * this name is displayed together with the MultiWii version number
   * upon powerup on the LCD.
   * If you are without a DISPLAYD then You may enable LCD_TTY and
   * use arduino IDE's serial monitor to view the info.
   *
   * You must preserve the format of this string!
   * It must be 16 characters total,
   * The last 4 characters will be overwritten with the version number.
   */
  #define BOARD_NAME "MultiWii   V-.--"
  //                 123456789.123456

 /*************      Support multiple configuration profiles in EEPROM
************/
  //#define MULTIPLE_CONFIGURATION_PROFILES

 /*************      do no reset constants when change of flashed program is
detected ***********/
  #define NO_FLASH_CHECK

/*************************************************************************
*****************************/
/*****************                                                 **************/
/***************  SECTION  7 - TUNING & DEVELOPER
**************/
/*****************                                                 **************/
/*************************************************************************
*****************************/

 #define VBAT_PRESCALER 16 // set this to 8 if vbatscale would exceed 255


/*************************************************************************
******************/
 /********   special ESC with extended range [0-2000] microseconds
********************/

/*************************************************************************
******************/
  //#define EXT_MOTOR_RANGE // using this with wii-esc requires to change
MINCOMMAND to 1008 for promini and mega


/*************************************************************************
******************/
 /********  brushed ESC
*****************************************************************/

/*************************************************************************
```

```
******************/
   // for 328p proc
   //#define EXT_MOTOR_32KHZ
   //#define EXT_MOTOR_4KHZ
   //#define EXT_MOTOR_1KHZ

   // for 32u4 proc
   //#define EXT_MOTOR_64KHZ
   //#define EXT_MOTOR_32KHZ
   //#define EXT_MOTOR_16KHZ
   //#define EXT_MOTOR_8KHZ


/*********************************************************************
******************/
  /*********************     motor, servo and other presets
**********************/

/*********************************************************************
******************/
   /* motors will not spin when the throttle command is in low position
      this is an alternative method to stop immediately the motors */
   //#define MOTOR_STOP

   /* some radios have not a neutral point centered on 1500. can be changed here */
   #define MIDRC 1500

  /*********************        Servo Refreshrates
**********************/
   /* Default 50Hz Servo refresh rate*/
   #define SERVO_RFR_50HZ

   /* up to 160Hz servo refreshrate .. works with the most analog servos*/
   //#define SERVO_RFR_160HZ

   /* up to 300Hz refreshrate it is as fast as possible (100-300Hz depending on the
cound of used servos and the servos state).
      for use with digital servos
      dont use it with analog servos! thay may get damage. (some will work but be
careful) */
   //#define SERVO_RFR_300HZ

  /*********************          HW PWM Servos
**********************/
   /* HW PWM Servo outputs for Arduino Mega.. moves:
     Pitch   = pin 44
     Roll    = pin 45
     CamTrig = pin 46
     SERVO4  = pin 11 (aileron left for fixed wing or TRI YAW SERVO)
     SERVO5  = pin 12 (aileron right for fixed wing)
```

```
     SERVO6  = pin 6   (rudder for fixed wing)
     SERVO7  = pin 7   (elevator for fixed wing)
     SERVO8  = pin 8   (motor for fixed wing)      */

   #define MEGA_HW_PWM_SERVOS

    /* HW PWM Servo outputs for 32u4 NanoWii, MicroWii etc. - works with either the variable SERVO_RFR_RATE or
     * one of the 3 fixed servo.refresh.rates *
     * Tested only for heli_120, i.e. 1 motor + 4 servos, moves..
     * motor[0] = motor       = pin  6
     * servo[3] = nick  servo = pin 11
     * servo[4] = left  servo = pin 10
     * servo[5] = yaw   servo = pin  5
     * servo[6]  = right servo= pin  9
     */
    //#define A32U4_4_HW_PWM_SERVOS

    #define SERVO_RFR_RATE  50    // In Hz, you can set it from 20 to 400Hz, used only in HW PWM mode for mega and 32u4
    //#define SERVO_PIN5_RFR_RATE  200    // separate yaw pwm rate.
                                  // In Hz, you can set it from 20 to 400Hz, used only in HW PWM mode for 32u4



/*********************************************************************/
  /****          Memory savings                          ****/

/*********************************************************************/

    /* options to counter the general shortage of both flash and ram memory, like with leonardo m32u4 and others */

    /**** suppress handling of serial commands.***
     * This does _not_ affect handling of RXserial, Spektrum or GPS. Those will not be affected and still work the same.
     * Enable either one or both of the following options  */

      /* Remove handling of all commands of the New MultiWii Serial Protocol.
       * This will disable use of the GUI, winGUI, android apps and any other program that makes use of the MSP.
       * You must find another way (like LCD_CONF) to tune the parameters or live with the defaults.
       * If you run a LCD/OLED via i2c or serial/Bluetooth, this is safe to use */
      //#define SUPPRESS_ALL_SERIAL_MSP // saves approx 2700 bytes

      /* Remove handling of other serial commands.
       * This includes navigating via serial the lcd.configuration menu, lcd.telemetry and permanent.log .
```

```
     * Navigating via stick inputs on tx is not affected and will work the same.  */
    //#define SUPPRESS_OTHER_SERIAL_COMMANDS // saves  approx 0 to 100 bytes, depending on features enabled

   /**** suppress keeping the defaults for initial setup and reset in the code.
    * This requires a manual initial setup of the PIDs etc. or load and write from defaults.mwi;
    * reset in GUI will not work on PIDs
    */
   //#define SUPPRESS_DEFAULTS_FROM_GUI

   //#define DISABLE_SETTINGS_TAB  // Saves ~400bytes on ProMini


/*********************************************************************/
 /****          diagnostics                                  ****/
/*********************************************************************/

   /* to log values like max loop time and others to come
      logging values are visible via LCD config
      set to 1, enable 'R' option to reset values, max current, max altitude
      set to 2, adds min/max cycleTimes
      set to 3, adds additional powerconsumption on a per motor basis (this uses the big array and is a memory hog, if POWERMETER <> PM_SOFT) */
   //#define LOG_VALUES 1

   /* Permanent logging to eeprom - survives (most) upgrades and parameter resets.
    * used to track number of flights etc. over lifetime of controller board.
    * Writes to end of eeprom - should not conflict with stored parameters yet.
    * Logged values: accumulated lifetime, #powercycle/reset/initialize events, #arm events, #disarm events, last armedTime,
    *              #failsafe@disarm, #i2c_errs@disarm
    * Enable one or more options to show the log
    */
   //#define LOG_PERMANENT
   //#define LOG_PERMANENT_SHOW_AT_STARTUP // enable to display log at startup
   //#define LOG_PERMANENT_SHOW_AT_L // enable to display log when receiving 'L'
   //#define LOG_PERMANENT_SHOW_AFTER_CONFIG // enable to display log after exiting LCD config menu
   //#define LOG_PERMANENT_SERVICE_LIFETIME 36000 // in seconds; service alert at startup after 10 hours of armed time

   /* to add debugging code
      not needed and not recommended for normal operation
      will add extra code that may slow down the main loop or make copter non-flyable */
   //#define DEBUG
```

```
  //#define DEBUG_FREE // will add 'F' command to show free memory

  /* Use this to trigger LCD configuration without a TX - only for debugging - do
NOT fly with this activated */
  //#define LCD_CONF_DEBUG

  /* Use this to trigger telemetry without a TX - only for debugging - do NOT fly
with this activated */
  //#define LCD_TELEMETRY_DEBUG    //This form rolls between all screens,
LCD_TELEMETRY_AUTO must also be defined.
  //#define LCD_TELEMETRY_DEBUG 6  //This form stays on the screen
specified.

  /* Enable string transmissions from copter to GUI */
  //#define DEBUGMSG



/**************************************************************************/
 /****          ESCs calibration                          ****/

/**************************************************************************/

  /* to calibrate all ESCs connected to MWii at the same time (useful to avoid
unplugging/re-plugging each ESC)
     Warning: this creates a special version of MultiWii Code
     You cannot fly with this special version. It is only to be used for calibrating ESCs
     Read How To at http://code.google.com/p/multiwii/wiki/ESCsCalibration */
  #define ESC_CALIB_LOW  MINCOMMAND
  #define ESC_CALIB_HIGH 2000
  //#define ESC_CALIB_CANNOT_FLY  // uncomment to activate

 /****          internal frequencies                      ****/
  /* frequenies for rare cyclic actions in the main loop, depend on cycle time
     time base is main loop cycle time - a value of 6 means to trigger the action every
6th run through the main loop
     example: with cycle time of approx 3ms, do action every 6*3ms=18ms
     value must be [1; 65535] */
  #define LCD_TELEMETRY_FREQ 23       // to send telemetry data over serial 23
<=> 60ms <=> 16Hz (only sending interlaced, so 8Hz update rate)
  #define LCD_TELEMETRY_AUTO_FREQ  967// to step to next telemetry page
967 <=> 3s
  #define PSENSOR_SMOOTH 16          // len of averaging vector for smoothing
the PSENSOR readings; should be power of 2; set to 1 to disable
  #define VBAT_SMOOTH 16             // len of averaging vector for smoothing the
VBAT readings; should be power of 2; set to 1 to disable
  #define RSSI_SMOOTH 16             // len of averaging vector for smoothing the
RSSI readings; should be power of 2; set to 1 to disable
```

```
/*********************************************************************/
  /****          Dynamic Motor/Prop Balancing                   ****/

/*********************************************************************/
  /*                !!! No Fly Mode !!!                          */

   //#define DYNBALANCE   // (**) Dynamic balancing controlled from Gui


/*********************************************************************/
  /****          Regression testing                             ****/

/*********************************************************************/

   /* for development only:
      to allow for easier and reproducable config sets for test compiling, different sets
of config parameters are kept
      together. This is meant to help detecting compile time errors for various features
in a coordinated way.
      It is not meant to produce your flying firmware
      To use:
      - do not set any options in config.h,
      - enable with #define COPTERTEST 1, then compile
      - if possible, check for the size
      - repeat with other values of 2, 3, 4 etc.
       */
   //#define COPTERTEST 1

/*********************************************************************
*****************************/
/*****************                                          **************/
/***************  SECTION  8 - DEPRECATED
*******/
/*****************                                          **************/
/*********************************************************************
*****************************/

  /* these features will be removed in the unforseeable future. Do not build new
products or
   * functionality based on such features. The default for all such features is OFF.
   */

  /*************************    WMP power pin
*******************************/
  //#define D12_POWER      // Use D12 on PROMINI to power sensors. Will disable
servo[4] on D12
  /* disable use of the POWER PIN (allready done if the option RCAUXPIN12 is
selected) */
  #define DISABLE_POWER_PIN
```

```
/**************************************************************************
*****************************/
/****          END OF CONFIGURABLE PARAMETERS
****/
/**************************************************************************
*****************************/

#endif /* CONFIG_H_ */
```
καρτέλα def.h
```
#ifndef DEF_H_
#define DEF_H_

/**************************************************************************
******************/
/***************          test configurations                ********************/
/**************************************************************************
******************/
#if COPTERTEST == 1
  #define QUADP
  #define WMP
#elif COPTERTEST == 2
  #define FLYING_WING
  #define WMP
  #define BMA020
  #define FAILSAFE
  #define LCD_CONF
  #define LCD_TEXTSTAR
  #define VBAT
  #define POWERMETER_SOFT
#elif COPTERTEST == 3
  #define TRI
  #define FREEIMUv035_MS
  #define BUZZER
  #define VBAT
  #define POWERMETER_HARD
  #define LCD_CONF
  #define LCD_CONF_AUX
  #define LCD_VT100
  #define LCD_TELEMETRY
  #define LCD_TELEMETRY_STEP "01245"
  #define LOG_VALUES 1
  #define SUPPRESS_BARO_ALTHOLD
  #define VARIOMETER 12
#elif COPTERTEST == 4
  #define QUADX
  #define CRIUS_SE
  #define SPEKTRUM 2048
  #define LED_RING
  #define GPS_SERIAL 2
  #define NMEA
```

```
  #define LOG_VALUES 2
  #define LOG_PERMANENT
  #define LOG_PERMANENT_SERVICE_LIFETIME 36000
#elif COPTERTEST == 5
  #define HELI_120_CCPM
  #define CRIUS_LITE
  #undef DISABLE_POWER_PIN
  #define RCAUXPIN8
  #define OLED_I2C_128x64
  #define LCD_TELEMETRY
  #define LOG_VALUES 3
  #define DEBUG
  #undef SERVO_RFR_50HZ
  #define SERVO_RFR_160HZ
  #define VBAT
  #define POWERMETER_SOFT
  #define MMGYRO 10
  #define MMGYROVECTORLENGTH 15
  #define GYRO_SMOOTHING {45, 45, 50}
  #define INFLIGHT_ACC_CALIBRATION
  #define LOG_PERMANENT
  #define LOG_PERMANENT_SHOW_AT_STARTUP
  #define LOG_PERMANENT_SHOW_AT_L
  #define LOG_PERMANENT_SERVICE_LIFETIME 36000
  #define GOVERNOR_P 0
  #define GOVERNOR_D 10
  #define YAW_COLL_PRECOMP 15
  #define YAW_COLL_PRECOMP_DEADBAND 130
  #define VOLTAGEDROP_COMPENSATION
#elif COPTERTEST == 6
  #define HEX6H
  #define DIYFLYING_MAGE_V1
  #define BUZZER
  #define RCOPTIONSBEEP // ca. 80byte
  #define ARMEDTIMEWARNING 480 // 8 min = 480seconds
  #define VBAT
  #define VOLTAGEDROP_COMPENSATION
  #define MEGA_HW_PWM_SERVOS
  #define SERVO_RFR_RATE  300    // In Hz, you can set it from 20 to 400Hz, used
only in HW PWM mode
  #define LOG_VALUES 1
  #define DEBUG
  #define MULTIPLE_CONFIGURATION_PROFILES
  #define DISPLAY_FONT_DSIZE
  #define OLED_DIGOLE
  #define LCD_CONF
#elif COPTERTEST == 7
  #define HELI_120_CCPM
  #define YAW_COLL_PRECOMP 15
  #define YAW_COLL_PRECOMP_DEADBAND 130
```

```
  #define NANOWII
  #define FORCE_ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = X;
imu.accADC[PITCH]  =  Y; imu.accADC[YAW]  =  Z;}
  #define FORCE_GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = -Y;
imu.gyroADC[PITCH] = X; imu.gyroADC[YAW] = -Z;}
  #define A32U4_4_HW_PWM_SERVOS
  #define SERVO_RFR_RATE  200    // 200 for graupner is ok
  #define SERVO_PIN5_RFR_RATE  165    // In Hz, you can set it from 20 to 400Hz,
used only in HW PWM mode for mega and 32u4
  #define SPEKTRUM 1024
  #define BUZZER
  #define RCOPTIONSBEEP // ca. 80byte
  #define VBAT
  #define LOG_VALUES 1
  #define DISPLAY_FONT_DSIZE
  #define OLED_DIGOLE
  #define LCD_CONF
  #define LCD_TELEMETRY
  #define LCD_TELEMETRY_AUTO "1"
  #define LCD_TELEMETRY_STEP "F14$5R"
  #define LOG_PERMANENT
  #define LOG_PERMANENT_SHOW_AFTER_CONFIG
  #define SUPPRESS_OTHER_SERIAL_COMMANDS
  #define SUPPRESS_DEFAULTS_FROM_GUI
  #define NO_FLASH_CHECK
  #define DEBUG_FREE
#elif COPTERTEST == 8
  #define BI
  #define ITG3200
  #define PID_CONTROLLER 2
  #define ESC_CALIB_CANNOT_FLY
#elif COPTERTEST == 9
  #define AIRPLANE
  #define FREEIMUv035
  #define POWERMETER_HARD
  #define WATTS
  #define VBAT
  #define VBAT_CELLS
  #define VBAT_CELLS_NUM 3
  #define VBAT_CELLS_PINS {A0, A1, A2 }
  #define VBAT_CELLS_OFFSETS {0, 50, 83 }
  #define VBAT_CELLS_DIVS { 75, 122,  98 }
#elif COPTERTEST == 10
  #define Y6
  #define CRIUS_AIO_PRO
  #define LCD_LCD03S
  #define SERIAL0_COM_SPEED 9600
  #define LCD_CONF
#elif defined(COPTERTEST)
  #error "*** this test is not yet defined"
```

```
#endif


/*************************************************************************
*****************/
/***************          Proc specific definitions          ***************/
/*************************************************************************
*****************/
// Proc auto detection
#if defined(__AVR_ATmega168__) || defined(__AVR_ATmega328P__)
  #define PROMINI
#endif
#if defined(__AVR_ATmega32U4__) || defined(TEENSY20)
  #define PROMICRO
#endif
#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega1281__) ||
defined(__AVR_ATmega2560__) || defined(__AVR_ATmega2561__)
  #define MEGA
#endif


/*************************************************************************
*****************/
/***************          motor and servo numbers
********************/
/*************************************************************************
*****************/
#define SERVO_RATES      {30,30,100,100,100,100,100,100}

#if defined (AIRPLANE) || defined(FLYING_WING)
  #define FIXEDWING
#endif

#if defined(HELI_120_CCPM) || defined(HELI_90_DEG)
  #define HELICOPTER
#endif

#if defined(BI) || defined(TRI) || defined(FIXEDWING) || defined(HELICOPTER) ||
defined(SINGLECOPTER)|| defined(DUALCOPTER)
  #define COPTER_WITH_SERVO
#endif

#if defined(COPTER_WITH_SERVO) || defined(SERVO_TILT) || defined(GIMBAL)
|| defined(CAMTRIG) || defined(SERVO_MIX_TILT)
  #define SERVO
#endif

#if defined(DYNBALANCE)
  #define DYNBAL 1
#else
```

```
  #define DYNBAL 0
#endif
#if defined(FLAPS)
  #define FLAP 1
#else
  #define FLAP 0
#endif

#if defined(MEGA) && defined(MEGA_HW_PWM_SERVOS)
  #define TRI_SERVO  4
#else
  #define TRI_SERVO  6
#endif

#if defined(GIMBAL)
  #define NUMBER_MOTOR     0
  #define PRI_SERVO_FROM   1 // use servo from 1 to 2
  #define PRI_SERVO_TO     2
#elif defined(FLYING_WING)
  #define PRI_SERVO_FROM   4
  #if defined (USE_THROTTLESERVO)
    #define NUMBER_MOTOR   0
    #define PRI_SERVO_TO   8 // use servo from 4,5 and 8
  #else
    #define NUMBER_MOTOR   1
    #define PRI_SERVO_TO   5 // use servo from 4 to 5
  #endif
#elif defined(SINGLECOPTER)
  #define NUMBER_MOTOR     1
  #define PRI_SERVO_FROM   4 // use servo from 4 to 7
  #define PRI_SERVO_TO     7
#elif defined(DUALCOPTER)
  #define NUMBER_MOTOR     2
  #define PRI_SERVO_FROM   5 // use servo from 5 to 6
  #define PRI_SERVO_TO     6
#elif defined(AIRPLANE)
  #if defined (USE_THROTTLESERVO)
    #define NUMBER_MOTOR   0
    #define PRI_SERVO_TO   8
  #else
    #define NUMBER_MOTOR   1
    #define PRI_SERVO_TO   7
  #endif
  #if defined(FLAPS)
    #define PRI_SERVO_FROM   3 // use servo from 3 to 8
    #undef CAMTRIG             // Disable Camtrig on A2
  #else
    #define PRI_SERVO_FROM   4 // use servo from 4 to 8
  #endif
#elif defined(BI)
```

```
  #define NUMBER_MOTOR     2
  #define PRI_SERVO_FROM   5 // use servo from 5 to 6
  #define PRI_SERVO_TO     6
#elif defined(TRI)
  #define NUMBER_MOTOR     3
  #define PRI_SERVO_FROM   TRI_SERVO // use only servo 6 (or 4 with Mega HW
PWM)
  #define PRI_SERVO_TO     TRI_SERVO
#elif defined(QUADP) || defined(QUADX) || defined(Y4)|| defined(VTAIL4)
  #define NUMBER_MOTOR     4
#elif defined(Y6) || defined(HEX6) || defined(HEX6X) || defined(HEX6H)
  #define NUMBER_MOTOR     6
#elif defined(OCTOX8) || defined(OCTOFLATP) || defined(OCTOFLATX)
  #define NUMBER_MOTOR     8
#elif defined(HELICOPTER)
  #define PRI_SERVO_FROM   4
  #ifdef HELI_USE_SERVO_FOR_THROTTLE
    #define NUMBER_MOTOR   0 // use servo to drive throttle output
    #define PRI_SERVO_TO   8 // use servo from 4 to 8
  #else
    #define NUMBER_MOTOR   1 // use motor1 for throttle, DO  NOT SET TO 2,
OR IT WILL BURN/DESTROY SERVO7 USED FOR SWASH
    #define PRI_SERVO_TO   7 // use servo from 4 to 7
  #endif
#endif

#if (defined(SERVO_TILT)|| defined(SERVO_MIX_TILT))&& defined(CAMTRIG)
  #define SEC_SERVO_FROM   1 // use servo from 1 to 3
  #define SEC_SERVO_TO     3
#else
  #if defined(SERVO_TILT)|| defined(SERVO_MIX_TILT)
    // if A0 and A1 is taken by motors, we can use A2 and 12 for Servo tilt
    #if defined(A0_A1_PIN_HEX) && (NUMBER_MOTOR == 6) &&
defined(PROMINI)
      #define SEC_SERVO_FROM   3 // use servo from 3 to 4
      #define SEC_SERVO_TO     4
    #else
      #define SEC_SERVO_FROM   1 // use servo from 1 to 2
      #define SEC_SERVO_TO     2
    #endif
  #endif
  #if defined(CAMTRIG)
    #define SEC_SERVO_FROM   3 // use servo 3
    #define SEC_SERVO_TO     3
  #endif
#endif

#if defined(SIRIUS_AIR) || defined(SIRIUS_AIR_GPS)
  #define RCAUX2PIND17
#endif
```

```
/************************  atmega328P (Promini)
**********************************/
#if defined(PROMINI)
  #if !defined(MONGOOSE1_0)
    #define LEDPIN_PINMODE            pinMode (13, OUTPUT);
    #define LEDPIN_TOGGLE             PINB |= 1<<5;     //switch LEDPIN state
(digital PIN 13)
    #define LEDPIN_OFF                PORTB &= ~(1<<5);
    #define LEDPIN_ON                 PORTB |= (1<<5);
  #endif
  #if !defined(RCAUXPIN8)
    #if !defined(MONGOOSE1_0)
      #define BUZZERPIN_PINMODE         pinMode (8, OUTPUT);
      #if NUMBER_MOTOR >4
        #undef PILOTLAMP
      #endif
      #if defined PILOTLAMP && NUMBER_MOTOR <5
        #define    PL_PIN_ON            PORTB |= 1;
        #define    PL_PIN_OFF           PORTB &= ~1;
      #else
        #define BUZZERPIN_ON            PORTB |= 1;
        #define BUZZERPIN_OFF           PORTB &= ~1;
      #endif
    #endif
  #else
    #define BUZZERPIN_PINMODE         ;
    #define BUZZERPIN_ON              ;
    #define BUZZERPIN_OFF             ;
    #define RCAUXPIN
  #endif
  #if !defined(RCAUXPIN12) && !defined(DISABLE_POWER_PIN)
    #define POWERPIN_PINMODE          pinMode (12, OUTPUT);
    #define POWERPIN_ON               PORTB |= 1<<4;
    #define POWERPIN_OFF              PORTB &= ~(1<<4); //switch OFF WMP,
digital PIN 12
  #else
    #define POWERPIN_PINMODE          ;
    #define POWERPIN_ON               ;
    #define POWERPIN_OFF              ;
  #endif
  #if defined(RCAUXPIN12)
    #define RCAUXPIN
  #endif
  #define I2C_PULLUPS_ENABLE        PORTC |= 1<<4; PORTC |= 1<<5;   // PIN
A4&A5 (SDA&SCL)
  #define I2C_PULLUPS_DISABLE       PORTC &= ~(1<<4); PORTC &= ~(1<<5);
  #if !defined(MONGOOSE1_0)
    #define PINMODE_LCD               pinMode(0, OUTPUT);
    #define LCDPIN_OFF                PORTD &= ~1; //switch OFF digital PIN 0
```

```
  #define LCDPIN_ON                PORTD |= 1;
  #define STABLEPIN_PINMODE        ;
  #define STABLEPIN_ON             ;
  #define STABLEPIN_OFF            ;
#endif
#define PPM_PIN_INTERRUPT          attachInterrupt(0, rxInt, RISING); //PIN 0
#define RX_SERIAL_PORT             0
//RX PIN assignment inside the port //for PORTD
#define THROTTLEPIN                2
#define ROLLPIN                    4
#define PITCHPIN                   5
#define YAWPIN                     6
#define AUX1PIN                    7
#define AUX2PIN                    0 // optional PIN 8 or PIN 12
#define AUX3PIN                    1 // unused
#define AUX4PIN                    3 // unused

#define PCINT_PIN_COUNT            5
#define PCINT_RX_BITS              (1<<2),(1<<4),(1<<5),(1<<6),(1<<7)
#define PCINT_RX_PORT              PORTD
#define PCINT_RX_MASK              PCMSK2
#define PCIR_PORT_BIT              (1<<2)
#define RX_PC_INTERRUPT            PCINT2_vect
#define RX_PCINT_PIN_PORT          PIND
#define V_BATPIN                   A3    // Analog PIN 3
#define PSENSORPIN                 A2    // Analog PIN 2

#if defined(A0_A1_PIN_HEX) || (NUMBER_MOTOR > 6)
  #define SOFT_PWM_1_PIN_HIGH      PORTC |= 1<<0;
  #define SOFT_PWM_1_PIN_LOW       PORTC &= ~(1<<0);
  #define SOFT_PWM_2_PIN_HIGH      PORTC |= 1<<1;
  #define SOFT_PWM_2_PIN_LOW       PORTC &= ~(1<<1);
#else
  #define SOFT_PWM_1_PIN_HIGH      PORTD |= 1<<5;
  #define SOFT_PWM_1_PIN_LOW       PORTD &= ~(1<<5);
  #define SOFT_PWM_2_PIN_HIGH      PORTD |= 1<<6;
  #define SOFT_PWM_2_PIN_LOW       PORTD &= ~(1<<6);
#endif
#define SOFT_PWM_3_PIN_HIGH        PORTC |= 1<<2;
#define SOFT_PWM_3_PIN_LOW         PORTC &= ~(1<<2);
#define SOFT_PWM_4_PIN_HIGH        PORTB |= 1<<4;
#define SOFT_PWM_4_PIN_LOW         PORTB &= ~(1<<4);

#define SERVO_1_PINMODE            pinMode(A0,OUTPUT); // TILT_PITCH -
WING left
#define SERVO_1_PIN_HIGH           PORTC |= 1<<0;
#define SERVO_1_PIN_LOW            PORTC &= ~(1<<0);
#define SERVO_2_PINMODE            pinMode(A1,OUTPUT); // TILT_ROLL  -
WING right
#define SERVO_2_PIN_HIGH           PORTC |= 1<<1;
```

```
  #define SERVO_2_PIN_LOW          PORTC &= ~(1<<1);
  #define SERVO_3_PINMODE          pinMode(A2,OUTPUT); // CAM TRIG  - alt
TILT_PITCH
  #define SERVO_3_PIN_HIGH         PORTC |= 1<<2;
  #define SERVO_3_PIN_LOW          PORTC &= ~(1<<2);
  #if !defined(MONGOOSE1_0)
    #define SERVO_4_PINMODE          pinMode(12,OUTPUT); // new       - alt
TILT_ROLL
    #define SERVO_4_PIN_HIGH         PORTB |= 1<<4;
    #define SERVO_4_PIN_LOW          PORTB &= ~(1<<4);
  #endif
  #define SERVO_5_PINMODE          pinMode(11,OUTPUT); // BI LEFT
  #define SERVO_5_PIN_HIGH         PORTB |= 1<<3;
  #define SERVO_5_PIN_LOW          PORTB &= ~(1<<3);
  #define SERVO_6_PINMODE          pinMode(3,OUTPUT);  // TRI REAR - BI
RIGHT
  #define SERVO_6_PIN_HIGH         PORTD|= 1<<3;
  #define SERVO_6_PIN_LOW          PORTD &= ~(1<<3);
  #define SERVO_7_PINMODE          pinMode(10,OUTPUT); // new
  #define SERVO_7_PIN_HIGH         PORTB |= 1<<2;
  #define SERVO_7_PIN_LOW          PORTB &= ~(1<<2);
  #define SERVO_8_PINMODE          pinMode(9,OUTPUT); // new
  #define SERVO_8_PIN_HIGH         PORTB |= 1<<1;
  #define SERVO_8_PIN_LOW          PORTB &= ~(1<<1);
#endif

/************************  atmega32u4 (Promicro)
**********************************/
#if defined(PROMICRO)
  #if defined(MICROWII)
    #define A32U4ALLPINS
  #endif
  #if !defined(TEENSY20)
    #define LEDPIN_PINMODE           //
    #define LEDPIN_TOGGLE            PIND |= 1<<5;     //switch LEDPIN state (Port
D5)
    #if !defined(PROMICRO10)
      #define LEDPIN_OFF             PORTD |= (1<<5);
      #define LEDPIN_ON              PORTD &= ~(1<<5);
    #else
      #define LEDPIN_OFF             PORTD &= ~(1<<5);
      #define LEDPIN_ON              PORTD |= (1<<5);
    #endif
  #else
    #define LEDPIN_PINMODE           DDRD |= (1<<6);
    #define LEDPIN_OFF             PORTD &= ~(1<<6);
    #define LEDPIN_ON              PORTD |= (1<<6);
    #define LEDPIN_TOGGLE            PIND |= 1<<6;     //switch LEDPIN state (Port
D6)
  #endif
```

```
  #if defined(D8BUZZER)
    #define BUZZERPIN_PINMODE         DDRB |= (1<<4);
    #if defined PILOTLAMP
      #define    PL_PIN_ON           PORTB |= 1<<4;
      #define    PL_PIN_OFF          PORTB &= ~(1<<4);
    #else
      #define BUZZERPIN_ON            PORTB |= 1<<4;
      #define BUZZERPIN_OFF           PORTB &= ~(1<<4);
    #endif

  #elif defined(A32U4ALLPINS)
    #define BUZZERPIN_PINMODE         DDRD |= (1<<4);
    #if defined PILOTLAMP
      #define    PL_PIN_ON    PORTD |= 1<<4;
      #define    PL_PIN_OFF   PORTD &= ~(1<<4);
    #else
      #define BUZZERPIN_ON            PORTD |= 1<<4;
      #define BUZZERPIN_OFF           PORTD &= ~(1<<4);
    #endif
  #else
    #define BUZZERPIN_PINMODE         DDRD |= (1<<3);
    #if defined PILOTLAMP
      #define    PL_PIN_ON    PORTD |= 1<<3;
      #define    PL_PIN_OFF   PORTD &= ~(1<<3);
    #else
      #define BUZZERPIN_ON            PORTD |= 1<<3;
      #define BUZZERPIN_OFF           PORTD &= ~(1<<3);
    #endif
  #endif
  #define POWERPIN_PINMODE          //
  #define POWERPIN_ON               //
  #define POWERPIN_OFF              //
  #define I2C_PULLUPS_ENABLE        PORTD |= 1<<0; PORTD |= 1<<1;   // PIN
2&3 (SDA&SCL)
  #define I2C_PULLUPS_DISABLE       PORTD &= ~(1<<0); PORTD &= ~(1<<1);
  #define PINMODE_LCD               DDRD |= (1<<2);
  #define LCDPIN_OFF                PORTD &= ~1;
  #define LCDPIN_ON                 PORTD |= 1;
  #define STABLEPIN_PINMODE         ;
  #define STABLEPIN_ON              ;
  #define STABLEPIN_OFF             ;
  #define PPM_PIN_INTERRUPT         DDRE &= ~(1 << 6);PORTE |= (1 << 6);
EICRB |= (1 << ISC61)|(1 << ISC60); EIMSK |= (1 << INT6);
  #if !defined(RX_SERIAL_PORT)
    #define RX_SERIAL_PORT          1
  #endif
  #define USB_CDC_TX                3
  #define USB_CDC_RX                2

  //soft PWM Pins
```

```
#define SOFT_PWM_1_PIN_HIGH      PORTD |= 1<<4;
#define SOFT_PWM_1_PIN_LOW       PORTD &= ~(1<<4);
#define SOFT_PWM_2_PIN_HIGH      PORTF |= 1<<5;
#define SOFT_PWM_2_PIN_LOW       PORTF &= ~(1<<5);
#if !defined(A32U4ALLPINS)
  #define SOFT_PWM_3_PIN_HIGH      PORTF |= 1<<7;
  #define SOFT_PWM_3_PIN_LOW       PORTF &= ~(1<<7);
  #define SOFT_PWM_4_PIN_HIGH      PORTF |= 1<<6;
  #define SOFT_PWM_4_PIN_LOW       PORTF &= ~(1<<6);
  #define SW_PWM_P3                A1
  #define SW_PWM_P4                A0
#else
  #define SOFT_PWM_3_PIN_HIGH      PORTF |= 1<<4;
  #define SOFT_PWM_3_PIN_LOW       PORTF &= ~(1<<4);
  #define SOFT_PWM_4_PIN_HIGH      PORTF |= 1<<5;
  #define SOFT_PWM_4_PIN_LOW       PORTF &= ~(1<<5);
  #define SW_PWM_P3                A2
  #define SW_PWM_P4                A3
#endif

// Servos
#define SERVO_1_PINMODE   DDRF |= (1<<7); // A0
#define SERVO_1_PIN_HIGH  PORTF|= 1<<7;
#define SERVO_1_PIN_LOW   PORTF &= ~(1<<7);
#define SERVO_2_PINMODE   DDRF |= (1<<6); // A1
#define SERVO_2_PIN_HIGH  PORTF |= 1<<6;
#define SERVO_2_PIN_LOW   PORTF &= ~(1<<6);
#define SERVO_3_PINMODE   DDRF |= (1<<5); // A2
#define SERVO_3_PIN_HIGH  PORTF |= 1<<5;
#define SERVO_3_PIN_LOW   PORTF &= ~(1<<5);
#if !defined(A32U4ALLPINS)
  #define SERVO_4_PINMODE   DDRD |= (1<<4); // 4
  #define SERVO_4_PIN_HIGH  PORTD |= 1<<4;
  #define SERVO_4_PIN_LOW   PORTD &= ~(1<<4);
#else
  #define SERVO_4_PINMODE   DDRF |= (1<<4); // A3
  #define SERVO_4_PIN_HIGH  PORTF |= 1<<4;
  #define SERVO_4_PIN_LOW   PORTF &= ~(1<<4);
#endif
#define SERVO_5_PINMODE   DDRC |= (1<<6); // 5
#define SERVO_5_PIN_HIGH  PORTC|= 1<<6;
#define SERVO_5_PIN_LOW   PORTC &= ~(1<<6);
#define SERVO_6_PINMODE   DDRD |= (1<<7); // 6
#define SERVO_6_PIN_HIGH  PORTD |= 1<<7;
#define SERVO_6_PIN_LOW   PORTD &= ~(1<<7);
#define SERVO_7_PINMODE   DDRB |= (1<<6); // 10
#define SERVO_7_PIN_HIGH  PORTB |= 1<<6;
#define SERVO_7_PIN_LOW   PORTB &= ~(1<<6);
#define SERVO_8_PINMODE   DDRB |= (1<<5); // 9
#define SERVO_8_PIN_HIGH  PORTB |= 1<<5;
```

```
 #define SERVO_8_PIN_LOW   PORTB &= ~(1<<5);

 //Standart RX
 #define THROTTLEPIN               3
 #if defined(A32U4ALLPINS)
  #define ROLLPIN                  6
  #define PITCHPIN                 2
  #define YAWPIN                   4
  #define AUX1PIN                  5
 #else
  #define ROLLPIN                  4
  #define PITCHPIN                 5
  #define YAWPIN                   2
  #define AUX1PIN                  6
 #endif
 #define AUX2PIN                   7
 #define AUX3PIN                   1 // unused
 #define AUX4PIN                   0 // unused
 #if !defined(RCAUX2PIND17)
  #define PCINT_PIN_COUNT          4
  #define PCINT_RX_BITS            (1<<1),(1<<2),(1<<3),(1<<4)
 #else
  #define PCINT_PIN_COUNT          5 // one more bit (PB0) is added in RX code
  #define PCINT_RX_BITS            (1<<1),(1<<2),(1<<3),(1<<4),(1<<0)
 #endif
 #define PCINT_RX_PORT             PORTB
 #define PCINT_RX_MASK             PCMSK0
 #define PCIR_PORT_BIT             (1<<0)
 #define RX_PC_INTERRUPT           PCINT0_vect
 #define RX_PCINT_PIN_PORT         PINB

 #if !defined(A32U4ALLPINS) && !defined(TEENSY20)
  #define V_BATPIN                 A3    // Analog PIN 3
 #elif defined(A32U4ALLPINS)
  #define V_BATPIN                 A4    // Analog PIN 4
 #else
  #define V_BATPIN                 A2    // Analog PIN 3
 #endif
 #if !defined(TEENSY20)
  #define PSENSORPIN               A2    // Analog PIN 2
 #else
  #define PSENSORPIN               A2    // Analog PIN 2
 #endif
#endif

/*************************  all the Mega types
**********************************/
#if defined(MEGA)
 #define LEDPIN_PINMODE            pinMode (13, OUTPUT);pinMode (30,
OUTPUT);
```

```
  #define LEDPIN_TOGGLE            PINB  |= (1<<7); PINC  |= (1<<7);
  #define LEDPIN_ON                PORTB |= (1<<7); PORTC |= (1<<7);
  #define LEDPIN_OFF               PORTB &= ~(1<<7);PORTC &= ~(1<<7);
  #define BUZZERPIN_PINMODE        pinMode (32, OUTPUT);
  #if defined PILOTLAMP
    #define    PL_PIN_ON    PORTC |= 1<<5;
    #define    PL_PIN_OFF   PORTC &= ~(1<<5);
  #else
    #define BUZZERPIN_ON             PORTC |= 1<<5;
    #define BUZZERPIN_OFF            PORTC &= ~(1<<5);
  #endif

  #if !defined(DISABLE_POWER_PIN)
    #define POWERPIN_PINMODE         pinMode (37, OUTPUT);
    #define POWERPIN_ON              PORTC |= 1<<0;
    #define POWERPIN_OFF             PORTC &= ~(1<<0);
  #else
    #define POWERPIN_PINMODE         ;
    #define POWERPIN_ON              ;
    #define POWERPIN_OFF             ;
  #endif
  #define I2C_PULLUPS_ENABLE       PORTD |= 1<<0; PORTD |= 1<<1;       //
PIN 20&21 (SDA&SCL)
  #define I2C_PULLUPS_DISABLE      PORTD &= ~(1<<0); PORTD &= ~(1<<1);
  #define PINMODE_LCD              pinMode(0, OUTPUT);
  #define LCDPIN_OFF               PORTE &= ~1; //switch OFF digital PIN 0
  #define LCDPIN_ON                PORTE |= 1;
  #define STABLEPIN_PINMODE        pinMode (31, OUTPUT);
  #define STABLEPIN_ON             PORTC |= 1<<6;
  #define STABLEPIN_OFF            PORTC &= ~(1<<6);
  #if defined(PPM_ON_THROTTLE)
    //configure THROTTLE PIN (A8 pin) as input witch pullup and enabled PCINT
interrupt
    #define PPM_PIN_INTERRUPT      DDRK &= ~(1<<0); PORTK |= (1<<0);
PCICR |= (1<<2); PCMSK2 |= (1<<0);
  #else
    #define PPM_PIN_INTERRUPT      attachInterrupt(4, rxInt, RISING);  //PIN 19,
also used for Spektrum satellite option
  #endif
  #if !defined(RX_SERIAL_PORT)
    #define RX_SERIAL_PORT         1
  #endif
  //RX PIN assignment inside the port //for PORTK
  #define THROTTLEPIN              0  //PIN 62 =  PIN A8
  #define ROLLPIN                  1  //PIN 63 =  PIN A9
  #define PITCHPIN                 2  //PIN 64 =  PIN A10
  #define YAWPIN                   3  //PIN 65 =  PIN A11
  #define AUX1PIN                  4  //PIN 66 =  PIN A12
  #define AUX2PIN                  5  //PIN 67 =  PIN A13
  #define AUX3PIN                  6  //PIN 68 =  PIN A14
```

```
  #define AUX4PIN                 7  //PIN 69 =  PIN A15
  #define V_BATPIN                A0    // Analog PIN 0
  #define PSENSORPIN              A2    // Analog PIN 2
  #define PCINT_PIN_COUNT         8
  #define PCINT_RX_BITS
(1<<2),(1<<4),(1<<5),(1<<6),(1<<7),(1<<0),(1<<1),(1<<3)
  #define PCINT_RX_PORT           PORTK
  #define PCINT_RX_MASK           PCMSK2
  #define PCIR_PORT_BIT           (1<<2)
  #define RX_PC_INTERRUPT         PCINT2_vect
  #define RX_PCINT_PIN_PORT       PINK

  #define SERVO_1_PINMODE
pinMode(34,OUTPUT);pinMode(44,OUTPUT); // TILT_PITCH - WING left
  #define SERVO_1_PIN_HIGH        PORTC |= 1<<3;PORTL |= 1<<5;
  #define SERVO_1_PIN_LOW         PORTC &= ~(1<<3);PORTL &= ~(1<<5);
  #define SERVO_2_PINMODE
pinMode(35,OUTPUT);pinMode(45,OUTPUT); // TILT_ROLL  - WING right
  #define SERVO_2_PIN_HIGH        PORTC |= 1<<2;PORTL |= 1<<4;
  #define SERVO_2_PIN_LOW         PORTC &= ~(1<<2);PORTL &= ~(1<<4);
  #define SERVO_3_PINMODE         pinMode(33,OUTPUT);
pinMode(46,OUTPUT); // CAM TRIG  - alt TILT_PITCH
  #define SERVO_3_PIN_HIGH        PORTC |= 1<<4;PORTL |= 1<<3;
  #define SERVO_3_PIN_LOW         PORTC &= ~(1<<4);PORTL &= ~(1<<3);
  #define SERVO_4_PINMODE         pinMode (37,
OUTPUT);pinMode(7,OUTPUT); // new       - alt TILT_ROLL
  #define SERVO_4_PIN_HIGH        PORTC |= 1<<0; PORTH |= 1<<4;
  #define SERVO_4_PIN_LOW         PORTC &= ~(1<<0);PORTH &= ~(1<<4);

  #define SERVO_5_PINMODE         pinMode(6,OUTPUT);                  // BI
LEFT
  #define SERVO_5_PIN_HIGH        PORTH |= 1<<3;
  #define SERVO_5_PIN_LOW         PORTH &= ~(1<<3);
  #define SERVO_6_PINMODE         pinMode(2,OUTPUT);                  // TRI
REAR - BI RIGHT
  #define SERVO_6_PIN_HIGH        PORTE |= 1<<4;
  #define SERVO_6_PIN_LOW         PORTE &= ~(1<<4);
  #define SERVO_7_PINMODE         pinMode(5,OUTPUT);                  // new
  #define SERVO_7_PIN_HIGH        PORTE |= 1<<3;
  #define SERVO_7_PIN_LOW         PORTE &= ~(1<<3);
  #define SERVO_8_PINMODE         pinMode(3,OUTPUT);                  // new
  #define SERVO_8_PIN_HIGH        PORTE |= 1<<5;
  #define SERVO_8_PIN_LOW         PORTE &= ~(1<<5);
#endif


// special defines for the Mongose IMU board
// note: that may be moved to the IMU Orientations because this are board defines ..
not Proc
```

```
#if defined(MONGOOSE1_0)  // basically it's a PROMINI without some PINS =>
same code as a PROMINI board except PIN definition
                  // note: to avoid too much dubble code there are now just the
differencies
  // http://www.multiwii.com/forum/viewtopic.php?f=6&t=627
  #define LEDPIN_PINMODE            pinMode (4, OUTPUT);
  #define LEDPIN_TOGGLE             PIND |= 1<<4;     //switch LEDPIN state
(digital PIN 13)
  #define LEDPIN_OFF                PORTD &= ~(1<<4);
  #define LEDPIN_ON                 PORTD |= (1<<4);
  #define SPEK_BAUD_SET             UCSR0A  = (1<<U2X0); UBRR0H = ((F_CPU
/ 4 / 115200 -1) / 2) >> 8; UBRR0L = ((F_CPU  / 4 / 115200 -1) / 2);
  #define RX_SERIAL_PORT            0

  /* Unavailable pins on MONGOOSE1_0 */
  #define BUZZERPIN_PINMODE         ; // D8
  #define BUZZERPIN_ON              ;
  #define BUZZERPIN_OFF             ;
  #define POWERPIN_PINMODE          ; // D12
  #define POWERPIN_ON               ;
  #define POWERPIN_OFF              ;
  #define STABLEPIN_PINMODE         ; //
  #define STABLEPIN_ON              ;
  #define STABLEPIN_OFF             ;
  #define PINMODE_LCD               ; //
  #define LCDPIN_OFF                ;
  #define LCDPIN_ON                 ;


  #define SERVO_4_PINMODE           ;                 // Not available
  #define SERVO_4_PIN_HIGH          ;
  #define SERVO_4_PIN_LOW           ;
#endif


/*********************  Sort the Servos for the most ideal SW PWM
************************/
// this define block sorts the above slected servos to be in a simple order from 1 -
(count of total servos)
// its pretty fat but its the best way i found to get less compiled code and max speed in
the ISR without loosing its flexibility
#if (PRI_SERVO_FROM == 1) || (SEC_SERVO_FROM == 1)
  #define LAST_LOW SERVO_1_PIN_LOW
  #define SERVO_1_HIGH SERVO_1_PIN_HIGH
  #define SERVO_1_LOW SERVO_1_PIN_LOW
  #define SERVO_1_ARR_POS  0
#endif
#if (PRI_SERVO_FROM <= 2 && PRI_SERVO_TO >= 2) || (SEC_SERVO_FROM
<= 2 && SEC_SERVO_TO >= 2)
  #undef LAST_LOW
```

```
  #define LAST_LOW SERVO_2_PIN_LOW
  #if !defined(SERVO_1_HIGH)
    #define SERVO_1_HIGH SERVO_2_PIN_HIGH
    #define SERVO_1_LOW SERVO_2_PIN_LOW
    #define SERVO_1_ARR_POS 1
  #else
    #define SERVO_2_HIGH SERVO_2_PIN_HIGH
    #define SERVO_2_LOW SERVO_2_PIN_LOW
    #define SERVO_2_ARR_POS 1
  #endif
 #endif
 #if (PRI_SERVO_FROM <= 3 && PRI_SERVO_TO >= 3) || (SEC_SERVO_FROM
 <= 3 && SEC_SERVO_TO >= 3)
  #undef LAST_LOW
  #define LAST_LOW SERVO_3_PIN_LOW
  #if !defined(SERVO_1_HIGH)
    #define SERVO_1_HIGH SERVO_3_PIN_HIGH
    #define SERVO_1_LOW SERVO_3_PIN_LOW
    #define SERVO_1_ARR_POS 2
  #elif !defined(SERVO_2_HIGH)
    #define SERVO_2_HIGH SERVO_3_PIN_HIGH
    #define SERVO_2_LOW SERVO_3_PIN_LOW
    #define SERVO_2_ARR_POS 2
  #else
    #define SERVO_3_HIGH SERVO_3_PIN_HIGH
    #define SERVO_3_LOW SERVO_3_PIN_LOW
    #define SERVO_3_ARR_POS 2
  #endif
 #endif
 #if (PRI_SERVO_FROM <= 4 && PRI_SERVO_TO >= 4) || (SEC_SERVO_FROM
 <= 4 && SEC_SERVO_TO >= 4)
  #undef LAST_LOW
  #define LAST_LOW SERVO_4_PIN_LOW
  #if !defined(SERVO_1_HIGH)
    #define SERVO_1_HIGH SERVO_4_PIN_HIGH
    #define SERVO_1_LOW SERVO_4_PIN_LOW
    #define SERVO_1_ARR_POS 3
  #elif !defined(SERVO_2_HIGH)
    #define SERVO_2_HIGH SERVO_4_PIN_HIGH
    #define SERVO_2_LOW SERVO_4_PIN_LOW
    #define SERVO_2_ARR_POS 3
  #elif !defined(SERVO_3_HIGH)
    #define SERVO_3_HIGH SERVO_4_PIN_HIGH
    #define SERVO_3_LOW SERVO_4_PIN_LOW
    #define SERVO_3_ARR_POS 3
  #else
    #define SERVO_4_HIGH SERVO_4_PIN_HIGH
    #define SERVO_4_LOW SERVO_4_PIN_LOW
    #define SERVO_4_ARR_POS 3
  #endif
```

```
#endif
#if (PRI_SERVO_FROM <= 5 && PRI_SERVO_TO >= 5) || (SEC_SERVO_FROM
<= 5 && SEC_SERVO_TO >= 5)
  #undef LAST_LOW
  #define LAST_LOW SERVO_5_PIN_LOW
  #if !defined(SERVO_1_HIGH)
    #define SERVO_1_HIGH SERVO_5_PIN_HIGH
    #define SERVO_1_LOW SERVO_5_PIN_LOW
    #define SERVO_1_ARR_POS 4
  #elif !defined(SERVO_2_HIGH)
    #define SERVO_2_HIGH SERVO_5_PIN_HIGH
    #define SERVO_2_LOW SERVO_5_PIN_LOW
    #define SERVO_2_ARR_POS 4
  #elif !defined(SERVO_3_HIGH)
    #define SERVO_3_HIGH SERVO_5_PIN_HIGH
    #define SERVO_3_LOW SERVO_5_PIN_LOW
    #define SERVO_3_ARR_POS 4
  #elif !defined(SERVO_4_HIGH)
    #define SERVO_4_HIGH SERVO_5_PIN_HIGH
    #define SERVO_4_LOW SERVO_5_PIN_LOW
    #define SERVO_4_ARR_POS 4
  #else
    #define SERVO_5_HIGH SERVO_5_PIN_HIGH
    #define SERVO_5_LOW SERVO_5_PIN_LOW
    #define SERVO_5_ARR_POS 4
  #endif
#endif
#if (PRI_SERVO_FROM <= 6 && PRI_SERVO_TO >= 6) || (SEC_SERVO_FROM
<= 6 && SEC_SERVO_TO >= 6)
  #undef LAST_LOW
  #define LAST_LOW SERVO_6_PIN_LOW
  #if !defined(SERVO_1_HIGH)
    #define SERVO_1_HIGH SERVO_6_PIN_HIGH
    #define SERVO_1_LOW SERVO_6_PIN_LOW
    #define SERVO_1_ARR_POS 5
  #elif !defined(SERVO_2_HIGH)
    #define SERVO_2_HIGH SERVO_6_PIN_HIGH
    #define SERVO_2_LOW SERVO_6_PIN_LOW
    #define SERVO_2_ARR_POS 5
  #elif !defined(SERVO_3_HIGH)
    #define SERVO_3_HIGH SERVO_6_PIN_HIGH
    #define SERVO_3_LOW SERVO_6_PIN_LOW
    #define SERVO_3_ARR_POS 5
  #elif !defined(SERVO_4_HIGH)
    #define SERVO_4_HIGH SERVO_6_PIN_HIGH
    #define SERVO_4_LOW SERVO_6_PIN_LOW
    #define SERVO_4_ARR_POS 5
  #elif !defined(SERVO_5_HIGH)
    #define SERVO_5_HIGH SERVO_6_PIN_HIGH
    #define SERVO_5_LOW SERVO_6_PIN_LOW
```

```
    #define SERVO_5_ARR_POS 5
   #else
    #define SERVO_6_HIGH SERVO_6_PIN_HIGH
    #define SERVO_6_LOW SERVO_6_PIN_LOW
    #define SERVO_6_ARR_POS 5
   #endif
  #endif
  #if (PRI_SERVO_FROM <= 7 && PRI_SERVO_TO >= 7) || (SEC_SERVO_FROM
  <= 7 && SEC_SERVO_TO >= 7)
   #undef LAST_LOW
   #define LAST_LOW SERVO_7_PIN_LOW
   #if !defined(SERVO_1_HIGH)
    #define SERVO_1_HIGH SERVO_7_PIN_HIGH
    #define SERVO_1_LOW SERVO_7_PIN_LOW
    #define SERVO_1_ARR_POS 6
   #elif !defined(SERVO_2_HIGH)
    #define SERVO_2_HIGH SERVO_7_PIN_HIGH
    #define SERVO_2_LOW SERVO_7_PIN_LOW
    #define SERVO_2_ARR_POS 6
   #elif !defined(SERVO_3_HIGH)
    #define SERVO_3_HIGH SERVO_7_PIN_HIGH
    #define SERVO_3_LOW SERVO_7_PIN_LOW
    #define SERVO_3_ARR_POS 6
   #elif !defined(SERVO_4_HIGH)
    #define SERVO_4_HIGH SERVO_7_PIN_HIGH
    #define SERVO_4_LOW SERVO_7_PIN_LOW
    #define SERVO_4_ARR_POS 6
   #elif !defined(SERVO_5_HIGH)
    #define SERVO_5_HIGH SERVO_7_PIN_HIGH
    #define SERVO_5_LOW SERVO_7_PIN_LOW
    #define SERVO_5_ARR_POS 6
   #elif !defined(SERVO_6_HIGH)
    #define SERVO_6_HIGH SERVO_7_PIN_HIGH
    #define SERVO_6_LOW SERVO_7_PIN_LOW
    #define SERVO_6_ARR_POS 6
   #else
    #define SERVO_7_HIGH SERVO_7_PIN_HIGH
    #define SERVO_7_LOW SERVO_7_PIN_LOW
    #define SERVO_7_ARR_POS 6
   #endif
  #endif
  #if (PRI_SERVO_FROM <= 8 && PRI_SERVO_TO >= 8) || (SEC_SERVO_FROM
  <= 8 && SEC_SERVO_TO >= 8)
   #undef LAST_LOW
   #define LAST_LOW SERVO_8_PIN_LOW
   #if !defined(SERVO_1_HIGH)
    #define SERVO_1_HIGH SERVO_8_PIN_HIGH
    #define SERVO_1_LOW SERVO_8_PIN_LOW
    #define SERVO_1_ARR_POS 7
   #elif !defined(SERVO_2_HIGH)
```

```
    #define SERVO_2_HIGH SERVO_8_PIN_HIGH
    #define SERVO_2_LOW SERVO_8_PIN_LOW
    #define SERVO_2_ARR_POS 7
   #elif !defined(SERVO_3_HIGH)
    #define SERVO_3_HIGH SERVO_8_PIN_HIGH
    #define SERVO_3_LOW SERVO_8_PIN_LOW
    #define SERVO_3_ARR_POS 7
   #elif !defined(SERVO_4_HIGH)
    #define SERVO_4_HIGH SERVO_8_PIN_HIGH
    #define SERVO_4_LOW SERVO_8_PIN_LOW
    #define SERVO_4_ARR_POS 7
   #elif !defined(SERVO_5_HIGH)
    #define SERVO_5_HIGH SERVO_8_PIN_HIGH
    #define SERVO_5_LOW SERVO_8_PIN_LOW
    #define SERVO_5_ARR_POS 7
   #elif !defined(SERVO_6_HIGH)
    #define SERVO_6_HIGH SERVO_8_PIN_HIGH
    #define SERVO_6_LOW SERVO_8_PIN_LOW
    #define SERVO_6_ARR_POS 7
   #elif !defined(SERVO_7_HIGH)
    #define SERVO_7_HIGH SERVO_8_PIN_HIGH
    #define SERVO_7_LOW SERVO_8_PIN_LOW
    #define SERVO_7_ARR_POS 7
   #else
    #define SERVO_8_HIGH SERVO_8_PIN_HIGH
    #define SERVO_8_LOW SERVO_8_PIN_LOW
    #define SERVO_8_ARR_POS 7
   #endif
#endif

#if ( defined(MEGA) && defined(MEGA_HW_PWM_SERVOS) ) ||
(defined(PROMICRO) && defined(A32U4_4_HW_PWM_SERVOS))
  #undef SERVO_1_HIGH                    // No software PWM's if we use
hardware MEGA PWM or promicro hardware pwm
  #define HW_PWM_SERVOS
#endif


/************************************************************************
*****************/
/***************      IMU Orientations and Sensor definitions
*******************/
/************************************************************************
*****************/


//please submit any correction to this list.
#if defined(FFIMUv1)
  #define ITG3200
  #define BMA180
```

```
  #define BMP085
  #define HMC5843
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
#endif

#if defined(FFIMUv2)
  #define ITG3200
  #define BMA180
  #define BMP085
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  Y;
imu.magADC[PITCH]  = -X; imu.magADC[YAW]  = -Z;}
#endif

#if defined(FREEIMUv1)
  #define ITG3200
  #define ADXL345
  #define HMC5843
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH] =  Y; imu.magADC[YAW]  = -Z;}
  #define ADXL345_ADDRESS 0x53
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(FREEIMUv03)
  #define ITG3200
  #define ADXL345 // this is actually an ADXL346 but that's just the same as
ADXL345
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #define ADXL345_ADDRESS 0x53
  #undef INTERNAL_I2C_PULLUPS
```

```
#endif

#if defined(FREEIMUv035) || defined(FREEIMUv035_MS) ||
defined(FREEIMUv035_BMP)
  #define ITG3200
  #define BMA180
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #undef INTERNAL_I2C_PULLUPS
  #if defined(FREEIMUv035_MS)
    #define MS561101BA
  #elif defined(FREEIMUv035_BMP)
    #define BMP085
  #endif
#endif

#if defined(FREEIMUv04)
 #define FREEIMUv043
#endif

#if defined(MultiWiiMega)
 #define FREEIMUv043
#endif

#if defined(FREEIMUv043)  || defined(MICROWII)
  #define MPU6050
  #define HMC5883
  #define MS561101BA
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #define MPU6050_I2C_AUX_MASTER // MAG connected to the AUX I2C bus of
MPU6050
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(NANOWII)
  #define MPU6050
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -Y;
imu.accADC[PITCH]  =  X; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = -X;
imu.gyroADC[PITCH] = -Y; imu.gyroADC[YAW] = -Z;}
```

```
  #undef INTERNAL_I2C_PULLUPS
  // move motor 7 & 8 to pin 4 & A2
  #undef SOFT_PWM_3_PIN_HIGH
  #undef SOFT_PWM_3_PIN_LOW
  #undef SOFT_PWM_4_PIN_HIGH
  #undef SOFT_PWM_4_PIN_LOW
  #undef SW_PWM_P3
  #undef SW_PWM_P4
  #define SOFT_PWM_3_PIN_HIGH       PORTD |= 1<<4;
  #define SOFT_PWM_3_PIN_LOW        PORTD &= ~(1<<4);
  #define SOFT_PWM_4_PIN_HIGH       PORTF |= 1<<5;
  #define SOFT_PWM_4_PIN_LOW        PORTF &= ~(1<<5);
  #define SW_PWM_P3                 4
  #define SW_PWM_P4                 A2
  #define HWPWM6
  // move servo 3 & 4 to pin 13 & 11
  #undef SERVO_3_PINMODE
  #undef SERVO_3_PIN_HIGH
  #undef SERVO_3_PIN_LOW
  #undef SERVO_4_PINMODE
  #undef SERVO_4_PIN_HIGH
  #undef SERVO_4_PIN_LOW
  #define SERVO_3_PINMODE   DDRC |= (1<<7); // 13
  #define SERVO_3_PIN_HIGH  PORTC |= 1<<7;
  #define SERVO_3_PIN_LOW   PORTC &= ~(1<<7);
  #define SERVO_4_PINMODE   DDRB |= (1<<7); // 11
  #define SERVO_4_PIN_HIGH  PORTB |= 1<<7;
  #define SERVO_4_PIN_LOW   PORTB &= ~(1<<7);
  // use pin 4 as status LED output if we have no octo
  #if !defined(OCTOX8) && !defined(OCTOFLATP) && !defined(OCTOFLATX)
    #undef LEDPIN_PINMODE
    #undef LEDPIN_TOGGLE
    #undef LEDPIN_OFF
    #undef LEDPIN_ON
    #define LEDPIN_PINMODE          DDRD |= (1<<4);          //D4 to output
    #define LEDPIN_TOGGLE           PIND |= (1<<5)|(1<<4);    //switch LEDPIN
state (Port D5) & pin D4
    #define LEDPIN_OFF              PORTD |= (1<<5); PORTD &= ~(1<<4);
    #define LEDPIN_ON               PORTD &= ~(1<<5); PORTD |= (1<<4);
  #endif
#endif

#if defined(PIPO)
  #define L3G4200D
  #define ADXL345
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  X;
imu.gyroADC[PITCH] =  Y; imu.gyroADC[YAW] = -Z;}
```

```
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  Y;
imu.magADC[PITCH]  = -X; imu.magADC[YAW]  =  Z;}
  #define ADXL345_ADDRESS 0x53
#endif

#if defined(QUADRINO)
  #define ITG3200
  #define BMA180
  #define BMP085
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
#endif

#if defined(QUADRINO_ZOOM)
  #define ITG3200
  #define BMA180
  #define BMP085
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #define STABLEPIN_PINMODE pinMode (A2, OUTPUT);
  #define STABLEPIN_ON PORTC |= (1<<2);
  #define STABLEPIN_OFF PORTC &= ~(1<<2);
#endif

#if defined(QUADRINO_ZOOM_MS)
  #define ITG3200
  #define BMA180
  #define MS561101BA
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #define STABLEPIN_PINMODE pinMode (A2, OUTPUT);
  #define STABLEPIN_ON PORTC |= (1<<2);
  #define STABLEPIN_OFF PORTC &= ~(1<<2);
#endif
```

```
#if defined(ALLINONE)
  #define ITG3200
  #define BMA180
  #define BMP085
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #define BMA180_ADDRESS 0x41
#endif

#if defined(AEROQUADSHIELDv2) // to confirm
  #define ITG3200
  #define BMA180
  #define BMP085
  #define HMC5843
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  =  X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  X;
imu.gyroADC[PITCH] =  Y; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #define GYRO_ADDRESS 0X69
#endif

#if defined(ATAVRSBIN1)
  #define ITG3200
  #define BMA020       //Actually it's a BMA150, but this is a drop in replacement for
the discountinued BMA020
  #define AK8975
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  =  Y;
imu.accADC[PITCH]  = -X; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  X;
imu.gyroADC[PITCH] =  Y; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  = -Y;
imu.magADC[PITCH]  = -X; imu.magADC[YAW]  =  Z;}
#endif

#if defined(SIRIUS)
  #define ITG3200
  #define BMA180
  #define BMP085
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
```

```
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
#endif

#if defined(SIRIUSGPS)
  #define ITG3200
  #define BMA180
  #define BMP085
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  = -X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  =  Z;}
#endif

#if defined(SIRIUS600)
  #define WMP
  #define BMA180
  #define BMP085
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
#endif

#if defined(SIRIUS_AIR)
  #define MPU6050
  #define BMP085
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = X;
imu.accADC[PITCH]  =  Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = -Y;
imu.gyroADC[PITCH] = X; imu.gyroADC[YAW] = -Z;}
  #undef INTERNAL_I2C_PULLUPS
  #define HWPWM6
#endif

#if defined(SIRIUS_AIR_GPS)
  #define MPU6050
  #define BMP085
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = X;
imu.accADC[PITCH]  =  Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = -Y;
imu.gyroADC[PITCH] = X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  -X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = Z;}   //normal Sirius MAG on
top is X Y -Z
```

```
  #undef INTERNAL_I2C_PULLUPS
  #define HWPWM6
#endif

#if defined(SIRIUS_MEGAv5_OSD)
  #define ITG3050
  #define BMA280
  #define MS561101BA
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  =  -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = -X;
imu.gyroADC[PITCH] =  -Y; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  Y;
imu.magADC[PITCH]  =  -X; imu.magADC[YAW]  = -Z;}
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(MINIWII)
  #define ITG3200
  #define BMA180
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
#endif

#if defined(CITRUSv2_1)
  #define ITG3200
  #define ADXL345
  #define BMP085
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL] = -X;
imu.accADC[PITCH] = -Y; imu.accADC[YAW] = Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL] = X;
imu.magADC[PITCH] = Y; imu.magADC[YAW] = -Z;}
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(CHERRY6DOFv1_0)
  #define MPU6050
  #define ACC_ORIENTATION(Y, X, Z)  {imu.accADC[ROLL]  = -Y;
imu.accADC[PITCH]  = -X; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(Y, X, Z) {imu.gyroADC[ROLL] =  X;
imu.gyroADC[PITCH] = -Y; imu.gyroADC[YAW] = -Z;}
  #undef INTERNAL_I2C_PULLUPS
#endif
```

```
#if defined(DROTEK_10DOF) || defined(DROTEK_10DOF_MS)
  #define ITG3200
  #define BMA180
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #define GYRO_ADDRESS 0X69
  #if defined(DROTEK_10DOF_MS)
    #define MS561101BA
  #elif defined(DROTEK_10DOF)
    #define BMP085
  #endif
#endif

#if defined(DROTEK_6DOFv2)
  #define ITG3200
  #define BMA180
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -Y;
imu.accADC[PITCH]  =  X; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = -X;
imu.gyroADC[PITCH] = -Y; imu.gyroADC[YAW] = -Z;}
  #define GYRO_ADDRESS 0X69
#endif

#if defined(DROTEK_6DOF_MPU)
  #define MPU6050
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -Y;
imu.accADC[PITCH]  =  X; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = -X;
imu.gyroADC[PITCH] = -Y; imu.gyroADC[YAW] = -Z;}
  #define MPU6050_ADDRESS 0x69
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(DROTEK_10DOF_MPU)
  #define MPU6050
  #define HMC5883
  #define MS561101BA
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  =  Y;
imu.accADC[PITCH]  = -X; imu.accADC[YAW]  = Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  X;
imu.gyroADC[PITCH] = Y; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  -Y;
imu.magADC[PITCH]  = X; imu.magADC[YAW]  = -Z;}
  #define MPU6050_ADDRESS 0X69
  #define MPU6050_I2C_AUX_MASTER // MAG connected to the AUX I2C bus of
```

```
MPU6050
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(FLYDUINO_MPU)
  #define MPU6050
  #define ACC_ORIENTATION(X, Y, Z) {imu.accADC[ROLL] = X;
imu.accADC[PITCH] = Y; imu.accADC[YAW] = Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = -Y;
imu.gyroADC[PITCH] = X; imu.gyroADC[YAW] = -Z;}
#endif

#if defined(MONGOOSE1_0)
  #define ITG3200
  #define ADXL345
  #define BMP085
  #define HMC5883
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = -Y;
imu.gyroADC[PITCH] =  X; imu.gyroADC[YAW] = -Z;}
  #define ACC_ORIENTATION(Y, X, Z)  {imu.accADC[ROLL]  =  Y;
imu.accADC[PITCH]  =  X; imu.accADC[YAW]  =  Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  = -X;
imu.magADC[PITCH]  = -Y; imu.magADC[YAW]  = -Z;}
  #define ADXL345_ADDRESS 0x53
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(CRIUS_LITE)
  #define ITG3200
  #define ADXL345
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
#endif

#if defined(CRIUS_SE)
  #define ITG3200
  #define BMA180
  #define HMC5883
  #define BMP085
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
#endif

#if defined(CRIUS_SE_v2_0)
```

```
  #define MPU6050
  #define HMC5883
  #define BMP085
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
#endif

#if defined(BOARD_PROTO_1)
  #define MPU6050
  #define HMC5883
  #define MS561101BA
  #define ACC_ORIENTATION(Y, X, Z)  {imu.accADC[ROLL]  = -Y;
imu.accADC[PITCH]  = -X; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(Y, X, Z) {imu.gyroADC[ROLL] =  X;
imu.gyroADC[PITCH] = -Y; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #define MS561101BA_ADDRESS 0x76
  #define STABLEPIN_PINMODE pinMode (A2, OUTPUT);
  #define STABLEPIN_ON PORTC |= (1<<2);
  #define STABLEPIN_OFF PORTC &= ~(1<<2);
#endif

#if defined(BOARD_PROTO_2)
  #define MPU6050
  #define MAG3110
  #define MS561101BA
  #define ACC_ORIENTATION(Y, X, Z)  {imu.accADC[ROLL]  = -Y;
imu.accADC[PITCH]  = -X; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(Y, X, Z) {imu.gyroADC[ROLL] =  X;
imu.gyroADC[PITCH] = -Y; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  =  Z;}
  #define MPU6050_I2C_AUX_MASTER
  #define MS561101BA_ADDRESS 0x76
  #define STABLEPIN_PINMODE pinMode (A2, OUTPUT);
  #define STABLEPIN_ON PORTC |= (1<<2);
  #define STABLEPIN_OFF PORTC &= ~(1<<2);
#endif

#if defined(GY_80)
  #define L3G4200D
  #define ADXL345
  #define HMC5883
  #define BMP085
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
```

```
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #undef INTERNAL_I2C_PULLUPS
  #define ADXL345_ADDRESS 0x53
#endif

#if defined(GY_85)
  #define ITG3200
  #define ADXL345
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #undef INTERNAL_I2C_PULLUPS
  #define ADXL345_ADDRESS 0x53
#endif

#if defined(GY_86)
  #define MPU6050
  #define HMC5883
  #define MS561101BA
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #define MPU6050_I2C_AUX_MASTER // MAG connected to the AUX I2C bus of
MPU6050
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(GY_88)
  #define MPU6050
  #define HMC5883
  #define BMP085
  #define ACC_ORIENTATION(X, Y, Z) {imu.accADC[ROLL] = -X;
imu.accADC[PITCH] = -Y; imu.accADC[YAW] = Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z) {imu.magADC[ROLL] = X;
imu.magADC[PITCH] = Y; imu.magADC[YAW] = -Z;}
  #define MPU6050_I2C_AUX_MASTER // MAG connected to the AUX I2C bus of
MPU6050
```

```
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(GY_521)
  #define MPU6050
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(INNOVWORKS_10DOF)
  #define ITG3200
  #define BMA180
  #define BMP085
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  = Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  = X;
imu.magADC[PITCH]  = Y; imu.imu.[YAW]  = -Z;}
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(INNOVWORKS_6DOF)
  #define ITG3200
  #define BMA180
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  = Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(PROTO_DIY)
  #define ITG3200
  #define BMA180
  #define HMC5883
  #define MS561101BA
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = X;
imu.accADC[PITCH]  = Y; imu.accADC[YAW]  = Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = X;
imu.gyroADC[PITCH] = Y; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  = X;
imu.magADC[PITCH]  = Y; imu.magADC[YAW]  = -Z;}
  #undef INTERNAL_I2C_PULLUPS
  #define STABLEPIN_ON              PORTC &= ~(1<<6);
  #define STABLEPIN_OFF             PORTC |= 1<<6;
```

```
#endif

#if defined(IOI_MINI_MULTIWII)
  #define ITG3200
  #define BMA180
  #define HMC5883
  #define BMP085
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL] = -X;
imu.accADC[PITCH] = -Y; imu.accADC[YAW] = Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL] = -Y;
imu.magADC[PITCH] = X; imu.magADC[YAW] = -Z;}
#endif

#if defined(Bobs_6DOF_V1)
  #define ITG3200
  #define BMA180
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  Y;
imu.magADC[PITCH]  = -X; imu.magADC[YAW]  = -Z;}
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(Bobs_9DOF_V1)
  #define ITG3200
  #define BMA180
  #define BMP085
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  Y;
imu.magADC[PITCH]  = -X; imu.magADC[YAW]  = -Z;}
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(Bobs_10DOF_BMP_V1)
  #define ITG3200
  #define BMA180
  #define BMP085  // Bobs 10DOF uses the BMP180 - BMP085 and BMP180 are
software compatible
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
```

```
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  Y;
imu.magADC[PITCH]  = -X; imu.magADC[YAW]  = -Z;}
  #undef INTERNAL_IC2_PULLUPS
#endif

#if defined(HK_MultiWii_SE_V2 )
  #define MPU6050
  #define HMC5883
  #define BMP085
  #define ACC_ORIENTATION(X, Y, Z) {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z){imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z) {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #define MPU6050_EN_I2C_BYPASS // MAG connected to the AUX I2C bus of
MPU6050
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(HK_MultiWii_328P )
  #define ITG3200
  #define BMA180
  #define HMC5883
  #define BMP085
  #define ACC_ORIENTATION(X, Y, Z) {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z){imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z) {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(CRIUS_AIO_PRO)
  #define MPU6050
  #define HMC5883
  #define MS561101BA
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #define MPU6050_I2C_AUX_MASTER // MAG connected to the AUX I2C bus of
MPU6050
  #undef INTERNAL_I2C_PULLUPS
  #define I2C_SPEED 400000L         //400kHz fast mode
  //servo pins on AIO board is at pins 44,45,46, then release pins 33,34,35 for other
usage
```

```
  //eg. pin 33 on AIO can be used for LEDFLASHER output
  #define SERVO_1_PINMODE          pinMode(44,OUTPUT);       // TILT_PITCH
  #define SERVO_1_PIN_HIGH         PORTL |= 1<<5;
  #define SERVO_1_PIN_LOW          PORTL &= ~(1<<5);
  #define SERVO_2_PINMODE          pinMode(45,OUTPUT);       // TILT_ROLL
  #define SERVO_2_PIN_HIGH         PORTL |= 1<<4;
  #define SERVO_2_PIN_LOW          PORTL &= ~(1<<4);
  #define SERVO_3_PINMODE          pinMode(46,OUTPUT);       // CAM TRIG
  #define SERVO_3_PIN_HIGH         PORTL |= 1<<3;
  #define SERVO_3_PIN_LOW          PORTL &= ~(1<<3);
#endif

#if defined(LADYBIRD)
  #define MPU6050
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #undef INTERNAL_I2C_PULLUPS
  #define MINTHROTTLE 1050
  #define MAXTHROTTLE 2000
  #define EXT_MOTOR_32KHZ
  #define VBAT
  #define VBATSCALE       54
  #define VBATLEVEL_WARN1 10
  #define VBATLEVEL_WARN2 10
  #define VBATLEVEL_CRIT  10
  #define NO_VBAT         10
  #define MOTOR_STOP
#endif

#if defined(MEGAWAP_V2_STD)
  #define ITG3200
  #define BMA180
  #define HMC5883
  #define BMP085
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
#endif

#if defined(MEGAWAP_V2_ADV)
  #define MPU6050
  #define HMC5883
  #define MS561101BA
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
```

```
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #define MPU6050_EN_I2C_BYPASS // MAG connected to the AUX I2C bus of
MPU6050
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(RCNet_FC_GPS)
  #define RCNet_FC
  #define HMC5883
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  -X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = Z;}
  #define MPU6050_I2C_AUX_MASTER // MAG connected to the AUX I2C bus of
MPU6050
  #undef INTERNAL_I2C_PULLUPS
  #define GPS_SERIAL 2
  #define GPS_BAUD   115200
  #define UBLOX
#endif

#if defined(RCNet_FC)
  #define MPU6050
  #define MS561101BA
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #undef INTERNAL_I2C_PULLUPS
  //servo pins on RCNet FC board are at pins 38,39,40
  #define SERVO_1_PINMODE          pinMode(40,OUTPUT);       // TILT_PITCH
  #define SERVO_1_PIN_HIGH         PORTL |= 1<<5;
  #define SERVO_1_PIN_LOW          PORTL &= ~(1<<5);
  #define SERVO_2_PINMODE          pinMode(39,OUTPUT);       // TILT_ROLL
  #define SERVO_2_PIN_HIGH         PORTL |= 1<<4;
  #define SERVO_2_PIN_LOW          PORTL &= ~(1<<4);
  #define SERVO_3_PINMODE          pinMode(38,OUTPUT);       // CAM TRIG
  #define SERVO_3_PIN_HIGH         PORTL |= 1<<3;
  #define SERVO_3_PIN_LOW          PORTL &= ~(1<<3);
#endif

#if defined(FLYDU_ULTRA)
  #define ITG3200
  #define MMA8451Q
  #define MS561101BA
  #define MAG3110

  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  =  X;
imu.accADC[PITCH] = Y; imu.accADC[YAW]  = Z;}
```

```
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] = Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  = -Y;
imu.magADC[PITCH] = X; imu.magADC[YAW]  = Z;}

  #define GPS_SERIAL 2
  #define GPS_BAUD   57600
  #define MTK_BINARY19
  #define INIT_MTK_GPS
#endif

#if defined(MultiWii_32U4_SE)
  #define MPU6050
  #define HMC5883
  #define MPU6050_I2C_AUX_MASTER // MAG connected to the AUX I2C bus of
MPU6050
  #define MS561101BA
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(MultiWii_32U4_SE_no_baro)
  #define MPU6050
  #define HMC5883
  #define MPU6050_I2C_AUX_MASTER // MAG connected to the AUX I2C bus of
MPU6050
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(Flyduino9DOF)
  #define MPU6050
  #define HMC5883
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #define MPU6050_EN_I2C_BYPASS // MAG connected to the AUX I2C bus of
```

```
MPU6050
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(Nano_Plane)
  #define LSM330
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(OPENLRSv2MULTI)
  #define ITG3200
  #define ADXL345
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define ADXL345_ADDRESS 0x53

  #define SDO_pin A0
  #define SDI_pin A1
  #define SCLK_pin A2
  #define IRQ_pin 2
  #define nSel_pin 4
  #define IRQ_interrupt 0

  #define  nIRQ_1 (PIND & 0x04)==0x04 //D2
  #define  nIRQ_0 (PIND & 0x04)==0x00 //D2

  #define  nSEL_on PORTD |= 0x10 //D4
  #define  nSEL_off PORTD &= 0xEF //D4

  #define  SCK_on PORTC |= 0x04 //C2
  #define  SCK_off PORTC &= 0xFB //C2

  #define  SDI_on PORTC |= 0x02 //C1
  #define  SDI_off PORTC &= 0xFD //C1

  #define  SDO_1 (PINC & 0x01) == 0x01 //C0
  #define  SDO_0 (PINC & 0x01) == 0x00 //C0

  //#### Other interface pinouts ###
  #define GREEN_LED_pin 13
  #define RED_LED_pin A3

  #define Red_LED_ON  PORTC |= _BV(3);
  #define Red_LED_OFF  PORTC &= ~_BV(3);
```

```
  #define Green_LED_ON  PORTB |= _BV(5);
  #define Green_LED_OFF  PORTB &= ~_BV(5);

  #define NOP() __asm__ __volatile__("nop")

  #define RF22B_PWRSTATE_READY    01
  #define RF22B_PWRSTATE_TX       0x09
  #define RF22B_PWRSTATE_RX       05
  #define RF22B_Rx_packet_received_interrupt   0x02
  #define RF22B_PACKET_SENT_INTERRUPT  04
  #define RF22B_PWRSTATE_POWERDOWN  00

#endif

#if defined(DESQUARED6DOFV2GO)
  #define ITG3200
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(DESQUARED6DOFV4)
  #define MPU6050
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(OSEPPGYRO)
  #define MPU3050
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #undef INTERNAL_I2C_PULLUPS
#endif

#if defined(DIYFLYING_MAGE_V1)
  #define MPU6050 // gyro+acc
  #define BMP085  // baro
  #define HMC5883 // mag
  #define ACC_ORIENTATION(X, Y, Z)  {imu.accADC[ROLL]  = -X;
imu.accADC[PITCH]  = -Y; imu.accADC[YAW]  =  Z;}
  #define GYRO_ORIENTATION(X, Y, Z) {imu.gyroADC[ROLL] =  Y;
imu.gyroADC[PITCH] = -X; imu.gyroADC[YAW] = -Z;}
  #define MAG_ORIENTATION(X, Y, Z)  {imu.magADC[ROLL]  =  X;
imu.magADC[PITCH]  =  Y; imu.magADC[YAW]  = -Z;}
  #undef INTERNAL_I2C_PULLUPS
#endif
```

```
/*********************************************************************
******************/
/***************          Sensor Type definitions          ******************/
/*********************************************************************
******************/

#if defined(ADXL345) || defined(BMA020) || defined(BMA180) || defined(BMA280)
|| defined(MMA7455) || defined(ADCACC) || defined(LIS3LV02) ||
defined(LSM303DLx_ACC) || defined(MPU6050) || defined(LSM330) ||
defined(MMA8451Q)
  #define ACC 1
#else
  #define ACC 0
#endif

#if defined(HMC5883) || defined(HMC5843) || defined(AK8975) ||
defined(MAG3110)
  #define MAG 1
#else
  #define MAG 0
#endif

#if defined(ITG3200) || defined(ITG3050) || defined(L3G4200D) ||
defined(MPU6050) || defined(LSM330) || defined(MPU3050) || defined(WMP)
  #define GYRO 1
#else
  #define GYRO 0
#endif

#if defined(BMP085) || defined(MS561101BA)
  #define BARO 1
#else
  #define BARO 0
#endif

#if defined(GPS_SERIAL)  || defined(I2C_GPS)
  #define GPS 1
#else
  #define GPS 0
#endif

#if defined(USE_MSP_WP)
  #define NAVCAP 1
#else
  #define NAVCAP 0
#endif

#if defined(SRF02) || defined(SRF08) || defined(SRF10) || defined(SRC235) ||
defined(I2C_GPS_SONAR)
```

```
  #define SONAR 1
#else
  #define SONAR 0
#endif

#if defined(EXTENDED_AUX_STATES)
  #define EXTAUX 1
#else
  #define EXTAUX 0
#endif

#if defined(RX_RSSI_CHAN)
  #define RX_RSSI
#endif

/************************************************************************
******************/
/***************      Multitype decleration for the GUI's
********************/
/************************************************************************
******************/
#if defined(TRI)
  #define MULTITYPE 1
#elif defined(QUADP)
  #define MULTITYPE 2
#elif defined(QUADX)
  #define MULTITYPE 3
#elif defined(BI)
  #define MULTITYPE 4
  #define SERVO_RATES      {30,30,100,100,0,1,100,100}
#elif defined(GIMBAL)
  #define MULTITYPE 5
#elif defined(Y6)
  #define MULTITYPE 6
#elif defined(HEX6)
  #define MULTITYPE 7
#elif defined(FLYING_WING)
  #define MULTITYPE 8
  #define SERVO_RATES      {30,30,100,0,1,100,100,100}
#elif defined(Y4)
  #define MULTITYPE 9
#elif defined(HEX6X)
  #define MULTITYPE 10
#elif defined(OCTOX8)
  #define MULTITYPE 11   //the JAVA GUI is the same for all 8 motor configs
#elif defined(OCTOFLATP)
  #define MULTITYPE 12   //12  for MultiWinGui
#elif defined(OCTOFLATX)
  #define MULTITYPE 13   //13  for MultiWinGui
#elif defined(AIRPLANE)
```

```
  #define MULTITYPE 14
  #define SERVO_RATES      {30,30,100,100,-100,100,100,100}
#elif defined (HELI_120_CCPM)
  #define MULTITYPE 15
#elif defined (HELI_90_DEG)
  #define MULTITYPE 16
  #define SERVO_RATES      {30,30,100,-100,-100,100,100,100}
#elif defined(VTAIL4)
  #define MULTITYPE 17
#elif defined(HEX6H)
  #define MULTITYPE 18
#elif defined(SINGLECOPTER)
  #define MULTITYPE 21
  #define SERVO_RATES      {30,30,100,0,1,0,1,100}
#elif defined(DUALCOPTER)
  #define MULTITYPE 20
#endif

/*********************************************************************
******************/
/***************          Some unsorted "chain" defines
*********************/
/*********************************************************************
******************/

#if defined (AIRPLANE) || defined(HELICOPTER)|| defined(SINGLECOPTER)||
defined(DUALCOPTER) && defined(PROMINI)
  #if defined(D12_POWER)
    #define SERVO_4_PINMODE           ;  // D12
    #define SERVO_4_PIN_HIGH          ;
    #define SERVO_4_PIN_LOW           ;
  #else
    #undef POWERPIN_PINMODE
    #undef POWERPIN_ON
    #undef POWERPIN_OFF
    #define POWERPIN_PINMODE          ;
    #define POWERPIN_ON               ;
    #define POWERPIN_OFF              ;
  #endif
#endif

#if defined(POWERMETER_HARD) || defined(POWERMETER_SOFT)
  #define POWERMETER
  #define PLEVELSCALE 50 // step size you can use to set alarm
  #define PLEVELDIVSOFT 100000
  #define PLEVELDIV 36000
#endif

#if defined PILOTLAMP
  #define    PL_CHANNEL OCR0B  //use B since A can be used by camstab
```

```
  #define    PL_ISR TIMER0_COMPB_vect
  #define    PL_INIT
TCCR0A=0;TIMSK0|=(1<<OCIE0B);PL_CHANNEL=PL_IDLE;PilotLamp(PL_GR
N_OFF);PilotLamp(PL_BLU_OFF);PilotLamp(PL_RED_OFF);PilotLamp(PL_BZR
_OFF);
  #define    BUZZERPIN_ON PilotLamp(PL_BZR_ON);
  #define    BUZZERPIN_OFF PilotLamp(PL_BZR_OFF);
  #define    PL_GRN_ON    25    // 100us
  #define    PL_GRN_OFF   50    // 200us
  #define    PL_BLU_ON    75    // 300us
  #define    PL_BLU_OFF   100    // 400us
  #define    PL_RED_ON    125    // 500us
  #define    PL_RED_OFF   150    // 600us
  #define    PL_BZR_ON    175    // 700us
  #define    PL_BZR_OFF   200    // 800us
  #define    PL_IDLE      125    // 100us
#endif

#if defined(PILOTLAMP)
  #define BUZZER
#endif

//all new Special RX's must be added here
//this is to avoid confusion :)
#if !defined(SERIAL_SUM_PPM) && !defined(SPEKTRUM) && !defined(SBUS)
&& !defined(SUMD)
  #define STANDARD_RX
#endif

#if defined(SPEKTRUM) || defined(SBUS) || defined(SUMD)
  #define SERIAL_RX
#endif

// Spektrum Satellite
#define BIND_CAPABLE 0  //Used for Spektrum today; can be used in the future for
any RX type that needs a bind and has a MultiWii module.
#if defined(SPEKTRUM)
  #define SPEK_FRAME_SIZE 16
  #if (SPEKTRUM == 1024)
    #define SPEK_CHAN_SHIFT  2       // Assumes 10 bit frames, that is 1024 mode.
    #define SPEK_CHAN_MASK   0x03    // Assumes 10 bit frames, that is 1024
mode.
    #define SPEK_DATA_SHIFT          // Assumes 10 bit frames, that is 1024 mode.
    #define SPEK_BIND_PULSES 3
  #endif
  #if (SPEKTRUM == 2048)
    #define SPEK_CHAN_SHIFT  3       // Assumes 11 bit frames, that is 2048 mode.
    #define SPEK_CHAN_MASK   0x07    // Assumes 11 bit frames, that is 2048
mode.
    #define SPEK_DATA_SHIFT >> 1     // Assumes 11 bit frames, that is 2048 mode.
```

```
    #define SPEK_BIND_PULSES 5
  #endif
  #if defined(SPEK_BIND)
    #define BIND_CAPABLE 1
    #if !defined(SPEK_BIND_GROUND)
      #define SPEK_BIND_GROUND 4
    #endif
    #if !defined(SPEK_BIND_POWER)
      #define SPEK_BIND_POWER  5
    #endif
    #if !defined(SPEK_BIND_DATA)
      #define SPEK_BIND_DATA   6
    #endif
  #endif
#endif

#if defined(SBUS)
  #define RC_CHANS 18
#elif defined(SPEKTRUM) || defined(SERIAL_SUM_PPM)
  #define RC_CHANS 12
#else
  #define RC_CHANS 8
#endif

#if !(defined(DISPLAY_2LINES)) && !(defined(DISPLAY_MULTILINE))
  #if (defined(LCD_VT100)) || (defined(OLED_I2C_128x64) ||
defined(OLED_DIGOLE) )
    #define DISPLAY_MULTILINE
  #else
    #define DISPLAY_2LINES
  #endif
#endif

#if (defined(LCD_VT100))
  #if !(defined(MULTILINE_PRE))
    #define MULTILINE_PRE 6
  #endif
  #if !(defined(MULTILINE_POST))
    #define MULTILINE_POST 9
  #endif
  #if !(defined(DISPLAY_COLUMNS))
    #define DISPLAY_COLUMNS 40
  #endif
#elif (defined(OLED_I2C_128x64) && defined(DISPLAY_FONT_DSIZE))
  #if !(defined(MULTILINE_PRE))
    #define MULTILINE_PRE 1
  #endif
  #if !(defined(MULTILINE_POST))
    #define MULTILINE_POST 3
  #endif
```

```
  #if !(defined(DISPLAY_COLUMNS))
    #define DISPLAY_COLUMNS 21
  #endif
#elif (defined(OLED_I2C_128x64))
  #if !(defined(MULTILINE_PRE))
    #define MULTILINE_PRE 3
  #endif
  #if !(defined(MULTILINE_POST))
    #define MULTILINE_POST 5
  #endif
  #if !(defined(DISPLAY_COLUMNS))
    #define DISPLAY_COLUMNS 21
  #endif
#elif (defined(OLED_DIGOLE) && defined(DISPLAY_FONT_DSIZE))
  #if !(defined(MULTILINE_PRE))
    #define MULTILINE_PRE 2
  #endif
  #if !(defined(MULTILINE_POST))
    #define MULTILINE_POST 3
  #endif
#elif (defined(OLED_DIGOLE))
  #if !(defined(MULTILINE_PRE))
    #define MULTILINE_PRE 3
  #endif
  #if !(defined(MULTILINE_POST))
    #define MULTILINE_POST 4
  #endif
  #if !(defined(DISPLAY_COLUMNS))
    #define DISPLAY_COLUMNS 21
  #endif
#endif

#if !(defined(DISPLAY_COLUMNS))
  #define DISPLAY_COLUMNS 16
#endif


/*****************************************************************************
*****************/
/***************          override defaults          ********************/
/*****************************************************************************
*****************/

  /***************          pin assignments ?  ********************/
  #ifdef OVERRIDE_V_BATPIN
    #undef V_BATPIN
    #define V_BATPIN OVERRIDE_V_BATPIN
  #endif
  #ifdef OVERRIDE_PSENSORPIN
    #undef PSENSORPIN
```

```
  #define PSENSORPIN OVERRIDE_PSENSORPIN
 #endif
 #ifdef OVERRIDE_LEDPIN_PINMODE
  #undef LEDPIN_PINMODE
  #undef LEDPIN_TOGGLE
  #undef LEDPIN_OFF
  #undef LEDPIN_ON
  #define LEDPIN_PINMODE OVERRIDE_LEDPIN_PINMODE
  #define LEDPIN_TOGGLE  OVERRIDE_LEDPIN_TOGGLE
  #define LEDPIN_OFF     OVERRIDE_LEDPIN_OFF
  #define LEDPIN_ON      OVERRIDE_LEDPIN_ON
 #endif
 #ifdef OVERRIDE_BUZZERPIN_PINMODE
  #undef BUZZERPIN_PINMODE
  #undef BUZZERPIN_ON
  #undef BUZZERPIN_OFF
  #define BUZZERPIN_PINMODE OVERRIDE_BUZZERPIN_PINMODE
  #define BUZZERPIN_ON      OVERRIDE_BUZZERPIN_ON
  #define BUZZERPIN_OFF     OVERRIDE_BUZZERPIN_OFF
 #endif

 /*********  sensors orientation - possibly overriding board defaults  *****/
 #ifdef FORCE_GYRO_ORIENTATION
  #undef GYRO_ORIENTATION
  #define GYRO_ORIENTATION FORCE_GYRO_ORIENTATION
 #endif
 #ifdef FORCE_ACC_ORIENTATION
  #undef ACC_ORIENTATION
  #define ACC_ORIENTATION FORCE_ACC_ORIENTATION
 #endif
 #ifdef FORCE_MAG_ORIENTATION
  #undef MAG_ORIENTATION
  #define MAG_ORIENTATION FORCE_MAG_ORIENTATION
 #endif

 /*********  servo rates                                    *****/
 #ifdef FORCE_SERVO_RATES
  #undef SERVO_RATES
  #define SERVO_RATES FORCE_SERVO_RATES
 #endif
/**************************************************************************
*****************/
/***************          Error Checking Section
*******************/
/**************************************************************************
*****************/

#ifndef NUMBER_MOTOR
      #error "NUMBER_MOTOR is not set, most likely you have not defined any type
of multicopter"
```

```
#endif

#if (defined(LCD_DUMMY) || defined(LCD_SERIAL3W) ||
defined(LCD_TEXTSTAR) || defined(LCD_VT100) || defined(LCD_TTY) ||
defined(LCD_ETPP) || defined(LCD_LCD03) || defined(LCD_LCD03S) ||
defined(OLED_I2C_128x64) ) || defined(OLED_DIGOLE)
  #define HAS_LCD
#endif

#if (defined(LCD_CONF) || defined(LCD_TELEMETRY)) &&
!(defined(HAS_LCD) )
  #error "LCD_CONF or LCD_TELEMETRY defined, and choice of LCD not
defined.  Uncomment one of LCD_SERIAL3W, LCD_TEXTSTAR, LCD_VT100,
LCD_TTY or LCD_ETPP, LCD_LCD03, LCD_LCD03S, OLED_I2C_128x64,
OLED_DIGOLE"
#endif

#if defined(POWERMETER_SOFT) && !(defined(VBAT))
  #error "to use powermeter, you must also define and configure VBAT"
#endif

#if defined(WATTS) && !(defined(POWERMETER_HARD)) && !(defined(VBAT))
  #error "to compute WATTS, you must also define and configure both
POWERMETER_HARD and VBAT"
#endif

#if defined(LCD_TELEMETRY_AUTO) && !(defined(LCD_TELEMETRY))
  #error "to use automatic telemetry, you MUST also define and configure
LCD_TELEMETRY"
#endif

#if defined(LCD_TELEMETRY_STEP) && !(defined(LCD_TELEMETRY))
  #error "to use single step telemetry, you MUST also define and configure
LCD_TELEMETRY"
#endif

#if defined(A32U4_4_HW_PWM_SERVOS) && !(defined(HELI_120_CCPM))
  #error "for your protection: A32U4_4_HW_PWM_SERVOS was not tested with
your coptertype"
#endif

#if GPS && !defined(NMEA) && !defined(UBLOX) &&
!defined(MTK_BINARY16) && !defined(MTK_BINARY19) &&
!defined(INIT_MTK_GPS) && !defined(I2C_GPS)
  #error "when using GPS you must specify the protocol NMEA, UBLOX..."
#endif

#if defined(NUNCHUK) || \
    defined( MPU6050_LPF_256HZ) || defined(MPU6050_LPF_188HZ)  || defined(
MPU6050_LPF_98HZ) || defined( MPU6050_LPF_42HZ) || \
```

```
   defined( MPU6050_LPF_20HZ)  || defined( MPU6050_LPF_10HZ)  || defined(
MPU6050_LPF_5HZ)  || \
   defined( ITG3200_LPF_256HZ) || defined( ITG3200_LPF_188HZ) || defined(
ITG3200_LPF_98HZ) || defined( ITG3200_LPF_42HZ) || \
   defined( ITG3200_LPF_20HZ)  || defined( ITG3200_LPF_10HZ)
  #error "you use one feature that is no longer supported or has undergone a name
change"
#endif

#endif /* DEF_H_ */
```

καρτέλα types.h

```
#ifndef TYPES_H_
#define TYPES_H_

enum rc {
  ROLL,
  PITCH,
  YAW,
  THROTTLE,
  AUX1,
  AUX2,
  AUX3,
  AUX4,
  AUX5,
  AUX6,
  AUX7,
  AUX8
};

enum pid {
  PIDROLL,
  PIDPITCH,
  PIDYAW,
  PIDALT,
  PIDPOS,
  PIDPOSR,
  PIDNAVR,
  PIDLEVEL,
  PIDMAG,
  PIDVEL,     // not used currently
  PIDITEMS
};

enum box {
  BOXARM,
  #if ACC
    BOXANGLE,
    BOXHORIZON,
  #endif
  #if BARO && (!defined(SUPPRESS_BARO_ALTHOLD))
```

```
  BOXBARO,
 #endif
 #ifdef VARIOMETER
  BOXVARIO,
 #endif
 BOXMAG,
 #if defined(HEADFREE)
  BOXHEADFREE,
  BOXHEADADJ, // acquire heading for HEADFREE mode
 #endif
 #if defined(SERVO_TILT) || defined(GIMBAL)  || defined(SERVO_MIX_TILT)
  BOXCAMSTAB,
 #endif
 #if defined(CAMTRIG)
  BOXCAMTRIG,
 #endif
 #if GPS
  BOXGPSHOME,
  BOXGPSHOLD,
 #endif
 #if defined(FIXEDWING) || defined(HELICOPTER)
  BOXPASSTHRU,
 #endif
 #if defined(BUZZER)
  BOXBEEPERON,
 #endif
 #if defined(LED_FLASHER)
  BOXLEDMAX, // we want maximum illumination
  BOXLEDLOW, // low/no lights
 #endif
 #if defined(LANDING_LIGHTS_DDR)
  BOXLLIGHTS, // enable landing lights at any altitude
 #endif
 #ifdef INFLIGHT_ACC_CALIBRATION
  BOXCALIB,
 #endif
 #ifdef GOVERNOR_P
  BOXGOV,
 #endif
 #ifdef OSD_SWITCH
  BOXOSD,
 #endif
 #if GPS
  BOXGPSNAV,
  BOXLAND,
 #endif
 CHECKBOXITEMS
};

typedef struct {
```

```
  int16_t  accSmooth[3];
  int16_t  gyroData[3];
  int16_t  magADC[3];
  int16_t  gyroADC[3];
  int16_t  accADC[3];
} imu_t;

typedef struct {
  uint8_t  vbat;               // battery voltage in 0.1V steps
  uint16_t intPowerMeterSum;
  uint16_t rssi;              // range: [0;1023]
  uint16_t amperage;          // 1unit == 100mA
  uint16_t watts;             // 1unit == 1W
  uint16_t vbatcells[VBAT_CELLS_NUM];
} analog_t;

typedef struct {
  int32_t  EstAlt;             // in cm
  int16_t  vario;              // variometer in cm/s
} alt_t;

typedef struct {
  int16_t angle[2];            // absolute angle inclination in multiple of 0.1 degree    180
deg = 1800
  int16_t heading;             // variometer in cm/s
} att_t;

typedef struct {
  uint8_t OK_TO_ARM :1 ;
  uint8_t ARMED :1 ;
  uint8_t ACC_CALIBRATED :1 ;
  uint8_t ANGLE_MODE :1 ;
  uint8_t HORIZON_MODE :1 ;
  uint8_t MAG_MODE :1 ;
  uint8_t BARO_MODE :1 ;
#ifdef HEADFREE
  uint8_t HEADFREE_MODE :1 ;
#endif
#if defined(FIXEDWING) || defined(HELICOPTER)
  uint8_t PASSTHRU_MODE :1 ;
#endif
  uint8_t SMALL_ANGLES_25 :1 ;
#if MAG
  uint8_t CALIBRATE_MAG :1 ;
#endif
#ifdef VARIOMETER
  uint8_t VARIO_MODE :1;
#endif
  uint8_t GPS_mode: 2;              // 0-3 NONE,HOLD, HOME, NAV (see
GPS_MODE_* defines
```

```
#if BARO || GPS
  uint8_t THROTTLE_IGNORED : 1;      // If it is 1 then ignore throttle stick
movements in baro mode;
#endif
#if GPS
  uint8_t GPS_FIX :1 ;
  uint8_t GPS_FIX_HOME :1 ;
  uint8_t GPS_BARO_MODE : 1;         // This flag is used when GPS controls baro
mode instead of user (it will replace rcOptions[BARO]
  uint8_t GPS_head_set: 1;           // it is 1 if the navigation engine got commands to
control heading (SET_POI or SET_HEAD) CLEAR_HEAD will zero it
  uint8_t LAND_COMPLETED: 1;
  uint8_t LAND_IN_PROGRESS: 1;
#endif
} flags_struct_t;

typedef struct {
  uint8_t currentSet;
  int16_t accZero[3];
  int16_t magZero[3];
  uint16_t flashsum;
  uint8_t checksum;      // MUST BE ON LAST POSITION OF STRUCTURE !
} global_conf_t;

struct pid_ {
  uint8_t P8;
  uint8_t I8;
  uint8_t D8;
};

struct servo_conf_ {  // this is a generic way to configure a servo, every multi type
with a servo should use it
  int16_t min;        // minimum value, must be more than 1020 with the current
implementation
  int16_t max;        // maximum value, must be less than 2000 with the current
implementation
  int16_t middle;     // default should be 1500
  int8_t  rate;       // range [-100;+100] ; can be used to ajust a rate 0-100% and a
direction
};

typedef struct {
  pid_    pid[PIDITEMS];
  uint8_t rcRate8;
  uint8_t rcExpo8;
  uint8_t rollPitchRate;
  uint8_t yawRate;
  uint8_t dynThrPID;
  uint8_t thrMid8;
  uint8_t thrExpo8;
```

```
  int16_t angleTrim[2];
  #if defined(EXTENDED_AUX_STATES)
   uint32_t activate[CHECKBOXITEMS];  //Extended aux states define six different
aux state for each aux channel
  #else
   uint16_t activate[CHECKBOXITEMS];
  #endif
  uint8_t powerTrigger1;
  #if MAG
    int16_t mag_declination;
  #endif
  servo_conf_ servoConf[8];
  #if defined(GYRO_SMOOTHING)
    uint8_t Smoothing[3];
  #endif
  #if defined (FAILSAFE)
    int16_t failsafe_throttle;
  #endif
  #ifdef VBAT
    uint8_t vbatscale;
    uint8_t vbatlevel_warn1;
    uint8_t vbatlevel_warn2;
    uint8_t vbatlevel_crit;
  #endif
  #ifdef POWERMETER
    uint8_t pint2ma;
  #endif
  #ifdef POWERMETER_HARD
    uint16_t psensornull;
  #endif
  #ifdef MMGYRO
    uint8_t mmgyro;
  #endif
  #ifdef ARMEDTIMEWARNING
    uint16_t armedtimewarning;
  #endif
  int16_t minthrottle;
  #ifdef GOVERNOR_P
   int16_t governorP;
   int16_t governorD;
  #endif
  #ifdef YAW_COLL_PRECOMP
   uint8_t yawCollPrecomp;
   uint16_t yawCollPrecompDeadband;
  #endif
  uint8_t  checksum;      // MUST BE ON LAST POSITION OF CONF STRUCTURE
!
} conf_t;

#ifdef LOG_PERMANENT
```

```
typedef struct {
  uint16_t arm;           // #arm events
  uint16_t disarm;        // #disarm events
  uint16_t start;         // #powercycle/reset/initialize events
  uint32_t armed_time ;   // copy of armedTime @ disarm
  uint32_t lifetime;      // sum (armed) lifetime in seconds
  uint16_t failsafe;      // #failsafe state @ disarm
  uint16_t i2c;           // #i2c errs state @ disarm
  uint8_t  running;       // toggle on arm & disarm to monitor for clean shutdown vs.
powercut
  uint8_t  checksum;      // MUST BE ON LAST POSITION OF CONF STRUCTURE
!
} plog_t;
#endif

#if GPS

// TODO: cross check with I2C gps and add relevant defines

//This is the mode what is selected via the remote (NONE, HOLD, RTH and NAV
(NAV-> exectute mission)
enum gpsmode {
  GPS_MODE_NONE = 0,
  GPS_MODE_HOLD,
  GPS_MODE_RTH,
  GPS_MODE_NAV
};

enum navstate {
  NAV_STATE_NONE = 0,
  NAV_STATE_RTH_START,
  NAV_STATE_RTH_ENROUTE,
  NAV_STATE_HOLD_INFINIT,
  NAV_STATE_HOLD_TIMED,
  NAV_STATE_WP_ENROUTE,
  NAV_STATE_PROCESS_NEXT,
  NAV_STATE_DO_JUMP,
  NAV_STATE_LAND_START,
  NAV_STATE_LAND_IN_PROGRESS,
  NAV_STATE_LANDED,
  NAV_STATE_LAND_SETTLE,
  NAV_STATE_LAND_START_DESCENT
};

enum naverror {
  NAV_ERROR_NONE = 0,          //All systems clear
  NAV_ERROR_TOOFAR,            //Next waypoint distance is more than safety
distance
  NAV_ERROR_SPOILED_GPS,       //GPS reception is compromised - Nav paused
- copter is adrift !
```

```
  NAV_ERROR_WP_CRC,             //CRC error reading WP data from EEPROM -
Nav stopped
  NAV_ERROR_FINISH,             //End flag detected, navigation finished
  NAV_ERROR_TIMEWAIT,           //Waiting for poshold timer
  NAV_ERROR_INVALID_JUMP,       //Invalid jump target detected, aborting
  NAV_ERROR_INVALID_DATA,       //Invalid mission step action code, aborting,
copter is adrift
  NAV_ERROR_WAIT_FOR_RTH_ALT,   //Waiting to reach RTH Altitude
  NAV_ERROR_GPS_FIX_LOST,       //Gps fix lost, aborting mission
  NAV_ERROR_DISARMED,           //NAV engine disabled due disarm
  NAV_ERROR_LANDING             //Landing
};

typedef struct {
  uint8_t  number;     //Waypoint number
  int32_t  pos[2];     //GPS position
  uint8_t  action;     //Action to follow
  int16_t  parameter1; //Parameter for the action
  int16_t  parameter2; //Parameter for the action
  int16_t  parameter3; //Parameter for the action
  uint32_t altitude;   //Altitude in cm (AGL)
  uint8_t  flag;       //flags the last wp and other fancy things that are not yet defined
  uint8_t  checksum;   //this must be at the last position
} mission_step_struct;


typedef struct {
  //Don't forget to change the reply size in GUI when change this struct;

  // on/off flags
  // First byte
  uint8_t filtering : 1;
  uint8_t lead_filter : 1;
  uint8_t dont_reset_home_at_arm : 1;
  uint8_t nav_controls_heading : 1;
  uint8_t nav_tail_first : 1;
  uint8_t nav_rth_takeoff_heading : 1;
  uint8_t slow_nav : 1;
  uint8_t wait_for_rth_alt : 1;
  // Second byte
  uint8_t ignore_throttle: 1; // Disable stick controls during mission and RTH
  uint8_t takeover_baro: 1;



  uint16_t wp_radius;           // in cm
  uint16_t safe_wp_distance;    // in meter
  uint16_t nav_max_altitude;    // in meter
  uint16_t nav_speed_max;       // in cm/s
```

```
  uint16_t nav_speed_min;       // in cm/s
  uint8_t  crosstrack_gain;     // * 100 (0-2.56)
  uint16_t nav_bank_max;        // degree * 100; (3000 default)
  uint16_t rth_altitude;        // in meter
  uint8_t  land_speed;          // between 50 and 255 (100 approx = 50cm/sec)
  uint16_t fence;               // fence control in meters

  uint8_t  max_wp_number;

  uint8_t  checksum;
} gps_conf_struct;

#endif

#endif /* TYPES_H_ */
```

Κώδικας Arduino UNO/χειριστήριο

```
/*
   *
   * Script Was Combined and Tested By DroneMesh
   * I did not create the Library this was Found online links are below
   * I did not create the PPM function Was found online links are below
   * I merged both codes to create something usefull out of it
   *
   *
   * PPM Output is on Pin 3
   *
   * Throttle:
   * L2 button on the PS3 Controller
   *
   * Yaw:
   * The right joystick
   *
   * Pitch and Roll:
   * The left joytsitck
   *
   * Useful Note:
   * The pitch axis is inverted when you map defualt however I have inverted it to make it work
   lik a normal RC transmitter
   *
```

```
*
* Notice the last 2 numbers in the funtion this is how to invert it you will not need by defaul this should work in AETR Mode
*
* This is Roll Axis
* ppm[0] = map(PS3.getAnalogHat(RightHatX), 0 , 255, 1000, 2000);
*
* This is Pitch Axis
* ppm[1] = map(PS3.getAnalogHat(RightHatY), 0 , 255, 2000, 1000);
*
*
*
*
* Download Library First
* https://github.com/felis/USB_Host_Shield_2.0/archive/master.zip
*
*
*
* Ch1 A (Roll ) == ppm[0]
* Ch2 E (Pitch ) == ppm[1]
* Ch3 T (Throttle ) == ppm[2]
* Ch4 R (Yaw ) == ppm[3]
* Ch5 AUX1 (Arm) == ppm[4]
* Ch6 AUX2 (NOT SET) == ppm[5]
* Ch7 AUX3 (NOT SET) == ppm[6]
* Ch8 AUX4 (NOT SET) == ppm[7]
*
*/


/*
Example sketch for the PS3 USB library - developed by Kristian Lauszus
For more information visit my blog: http://blog.tkjelectronics.dk/ or
send me an e-mail: kristianl@tkjelectronics.com
*/

#include <PS3USB.h>

// Satisfy the IDE, which needs to see the include statment in the ino too.
#ifdef dobogusinclude
#include <spi4teensy3.h>
```

```
#endif
#include <SPI.h>

USB Usb;
/* You can create the instance of the class in two ways */
PS3USB PS3(&Usb); // This will just create the instance
//PS3USB PS3(&Usb,0x00,0x15,0x83,0x3D,0x0A,0x57); // This will also store the bluetooth address - this can be obtained from the dongle when running the sketch

bool printAngle;
uint8_t state = 0;

////////////
/*
* PPM generator originally written by David Hasko
* on https://code.google.com/p/generate-ppm-signal/
*/
//////////////////////CONFIGURATION///////////////////////////////
#define CHANNEL_NUMBER 8 //set the number of chanels
#define CHANNEL_DEFAULT_VALUE 1500 //set the default servo value
#define FRAME_LENGTH 22500 //set the PPM frame length in microseconds (1ms = 1000µs)
#define PULSE_LENGTH 300 //set the pulse length
#define onState 1 //set polarity of the pulses: 1 is positive, 0 is negative
#define sigPin 3 //set PPM signal output pin on the arduino
////////////////////////////////////////////////////////////////

#define SWITCH_PIN 16
#define CHANNEL_TO_MODIFY 11
#define SWITCH_STEP 100

byte previousSwitchValue;

/*this array holds the servo values for the ppm signal
change theese values in your code (usually servo values move between 1000 and 2000)*/
int ppm[CHANNEL_NUMBER];

int currentChannelStep;
/////////////

void setup() {
Serial.begin(115200);
#if !defined(__MIPSEL__)
while (!Serial); // Wait for serial port to connect - used on Leonardo, Teensy and other
```

```
boards with built-in USB CDC serial connection
#endif
if (Usb.Init() == -1) {
Serial.print(F("\r\nOSC did not start"));
while (1); //halt
}
Serial.print(F("\r\nPS3 USB Library Started"));

///////////////////////////////
previousSwitchValue = HIGH;

//initiallize default ppm values
for(int i=0; i<CHANNEL_NUMBER; i++){
if (i == 2 || i == CHANNEL_TO_MODIFY) {
ppm[i] = 1000;
} else {
ppm[i]= CHANNEL_DEFAULT_VALUE;
}
}

pinMode(sigPin, OUTPUT);
pinMode(SWITCH_PIN, INPUT_PULLUP);
digitalWrite(sigPin, !onState); //set the PPM signal pin to the default state (off)

cli();
TCCR1A = 0; // set entire TCCR1 register to 0
TCCR1B = 0;

OCR1A = 100; // compare match register, change this
TCCR1B |= (1 << WGM12); // turn on CTC mode
TCCR1B |= (1 << CS11); // 8 prescaler: 0,5 microseconds at 16mhz
TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
sei();

currentChannelStep = SWITCH_STEP;

}

/* Channel Info Currently Setup AS AETR you can change this by changing the PPM Value in the
main loop
*
Ch1 A (Roll ) == ppm[0]
Ch2 E (Pitch ) == ppm[1]
Ch3 T (Throttle ) == ppm[2]
Ch4 R (Yaw ) == ppm[3]
```

```
Ch5 AUX1 (Arm) == ppm[4]
Ch6 AUX2 (NOT SET) == ppm[5]
Ch7 AUX3 (NOT SET) == ppm[6]
Ch8 AUX4 (NOT SET) == ppm[7]
*/

void loop() {
Usb.Task();
if (PS3.PS3Connected || PS3.PS3NavigationConnected) {
// Roll Axis is on PS3 Right Joystick X Axis
ppm[0] = map(PS3.getAnalogHat(RightHatX), 0 , 255, 1000, 2000);
// Pitch Axis on PS3 Right Joystick Y Axis (Inverted to work correctly)
ppm[1] = map(PS3.getAnalogHat(RightHatY), 0 , 255, 2000, 1000);
// Yaw Axis is on PS3 Left Joystick X Axis
ppm[3] = map(PS3.getAnalogHat(LeftHatX), 0 , 255, 1000, 2000);

// IF you want Throttle on Left Joystick Enable Below and Disable the Throttle on R2
//ppm[2] = map(PS3.getAnalogHat(LeftHatY), 0 , 255, 2000, 1000);

// Analog button values can be read from almost all buttons
if (PS3.getAnalogButton(L2) || PS3.getAnalogButton(R2)) {
if (!PS3.PS3NavigationConnected) {
// Throttle is on R2 Button
ppm[2] = map(PS3.getAnalogButton(R2), 0 , 255, 800, 2000);
}
}
if (PS3.getButtonClick(SQUARE)){
// AUX2 Upper // Bound Acro Mode
ppm[5] = 2000;
}
if (PS3.getButtonClick(CROSS)){
// AUX 2 Lower Bound // Angle Mode set in Betaflight
ppm[5] = 1000;
}

if (PS3.getButtonClick(L1))
{
// L1 to Disarm if Arm is on AUX1
ppm[4] = 1000;
```

```
}

if (PS3.getButtonClick(R1))
{
// R1 is to Arm if Arm is on AUX1
ppm[4] = 2000;}

}
if (PS3.PS3Connected == false){

ppm[4] = 1000;
}
}

ISR(TIMER1_COMPA_vect){ //leave this alone
static boolean state = true;

TCNT1 = 0;

if (state) { //start pulse
digitalWrite(sigPin, onState);
OCR1A = PULSE_LENGTH * 2;
state = false;
} else{ //end pulse and calculate when to start the next pulse
static byte cur_chan_numb;
static unsigned int calc_rest;

digitalWrite(sigPin, !onState);
state = true;

if(cur_chan_numb >= CHANNEL_NUMBER){
cur_chan_numb = 0;
calc_rest = calc_rest + PULSE_LENGTH;//
OCR1A = (FRAME_LENGTH - calc_rest) * 2;
calc_rest = 0;
}
else{
OCR1A = (ppm[cur_chan_numb] - PULSE_LENGTH) * 2;
calc_rest = calc_rest + ppm[cur_chan_numb];
cur_chan_numb++;
}
}
}
```