**Что делается в функции void setup().**

1. Происходит открытие и инициализация всех аппаратных последовательных портов платы. В MultiWii на базе Arduino pro mini один аппаратный последовательный порт. Также, в реализации SchoolWii для лодок и роверов применяется еще один программный последовательный порт – модифицированный из стандартного SoftwareSerial под названием MwiiSSerial. Модификация заключается в увеличении буфера приема до 256 байт от стандартных 64 байт. А также добавление в функцию обработки прерывания PCINT запрета аппаратного прерывания от приемника радиоаппаратуры (см. раздел «Работа с GPS). Это необходимо для приема пакетом NMEA от приемника GPS. Его инициализация происходит чуть позже, на шаге 11.
2. Настраивается пин светодиода (13) как выход (#define LEDPIN\_PINMODE DDRB |= 1<<5;)
3. Настраивается STABLEPIN – пин светодиода, который загорается если плату сильно наклонить. В SchoolWii этот функционал удален.
4. Инициализируются таймеры и регистры микроконтроллера, отвечающие за генерацию сигналов ШИМ на выходах платы (вызов функции initOutput()).
5. Считываются глобальные настройки из EEPROM (readGlobalSet()).
6. blinkLED
7. recallGPSconf(); //Load GPS configuration parameteres
8. Настраивается приемник радиосигнала (configureReceiver();)
9. Инициализируются сенсоры initSensors();
10. Задаются коэффициенты ПИД регулятров GPS (GPS\_set\_pids();)
11. Открытие и инициализация последовательного порта, к которому подключен приемник GPS. В реализации SchoolWii это программный последовательный порт, упомянутый на шаге 1.
12. Настраиваются регистры микроконтроллера, отвечающие за работу встроенного АЦП. АЦП используется для считывания значение напряжения и тока от аккумулятора. В SchoolWii считывается и контролируется только напряжение батареи с делителя напряжения, подключенного к пину A3 (см. схему).
13. Настраивается LED FLASHER – сигнальный индикатор, который можно подключить к плате, чтобы наблюдать ее состояние в более удобной форме, нежели по встроенному светодиоду на пине 13. Его функционал описан (реализован) в модуле Alarms.
14. В самом конце в версии SchoolWii добавлена строчка активации при загрузке режима PASSTHRU:

conf.activate[BOXPASSTHRU] = 1;

**Что делается в void loop()**

1. Вызывается функция computeRC(). Эта функция вычисляет и фильтрует данные с радиоприемника. На выходе эта функция заполняет массив int16\_t rcData[RC\_CHANS]. Длина RC\_CHANS массива равна максимальному количеству каналов, определенному в def.h (по умолчанию 12, но в случае Arduino pro mini и SchoolWii заполняются не все, а только 6 каналов). Диапазон каждого значение от 900 до примерно 2000 (диапазон значений серво). По умолчанию в этом массиве содержаться значения 1502 во всех ячейках.
2. Выполняется ряд процедур, связанных с Failsafe (если он определен). Вот как описана логика работы Failsafe в MultiWii:

«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.»

1. Выполняется обработка команд со стиков пульта. Пример логики этого участка:

Проверяется, что стиг тяги (throttle) в положении «минимум». Если на минимуме, то сбрасываются интегральные составляющие ПИД-контроллеров. А также проверяется состояние BOX-а отвечающего за ARM (conf.activate[BOXARM] > 0). Если контроллер находится в состоянии ARM (взведен), то происходит DISARM. Если же контроллер наоборот находится в состоянии DISARM, то происходит ARM.

*В MultiWii применяется своеобразная концепция хранения конфигурации контроллера через «боксы» (BOX). Управлять состоянием отдельных BOX с пульта радиоаппаратуры можно привязав состояние интересующих BOX к значениям AUX-каналов пульта.*

Проверяется, если стики находятся в каком-то ненулевом положении (20\*1/50) секунд (rcDelayCommand == 20), то выполняются различные действия с контроллером в зависимости от положения стиков (ARM / DISARM, калибровка акселерометра и пр).

Сбрасывается счетчик rcDelayCommand = 0;

Ну и дальше по коду там все понятно.

1. Обрабатывается состояние LEDFLASHER.
2. Запоминается состояние AUX (auxState) и сравнивается с текущей конфигурацией:

rcOptions[i] = (auxState & conf.activate[i])>0;

На основе rcOptions выполняются действия:

Если Установлен акселерометр (ACC):

Если изменился BOXANGLE (отвечает за переход в ANGLE\_MODE), то режим ANGLE\_MODE включается, если был выключен, либо выключается, если был включен.

Аналогично для BOXHORIZON.

Если rcOptions[BOXARM] == 0, то подготовить контроллер к ARM (f.OK\_TO\_ARM = 1;)

Если rcOptions[BOXMAG], то перейти в MAG\_MODE и в качестве азимута на удержание установить текущий азимут magHold = att.heading; В противном случае – выключить f.MAG\_MODE = 0;

Если установлен GPS:

Этот участок кода организован следующим образом. Есть четыре BOX, которые отвечают за режимы, связанные с GPS. Состояние BOX этих режимов заносится в переменную:

//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]);

И дальше уже разбирается – что делать в каждом режиме в зависимости от текущих состояний контроллера и обозначенных rcOptions. В целом этот участок кода достаточно прост для понимания.

Специально для самолетов проверяется rcOptions[BOXPASSTHRU] – режим, при котором сигналы с радиоприемника напрямую передаются на выход, без какой-либо стабилизации и обработки.

1. На этом rc\_loop заканчивается. В том случае, если на этом шаге главного цикла не произошло итерации rc\_loop (не прошло 20мс), то выполняется другая задача из списка диспетчера задач.

Диспетчер задач выполняет по одной задачке за цикл, самая долгая задача из того, что я видно по комментариям к коду MultiWii: Mag\_getADC() - 320мкс.

Хотя там также есть функции GPS\_Compute() и GPS\_NewData() без описания длительности (могу предположить, что их длительность явно больше).

1. После выполнения функции из диспетчера, вызывается функция расчёта IMU: computeIMU();
2. После нее проверяется не находится ли контроллер в состоянии, когда нужно игнорировать данные со стика тяги. Если находится – устанавливается соответствующий флаг: f.THROTTLE\_IGNORED = 1;
3. Происходит регулирование курса для малых отклонений стика курса (<70). Почему именно 70 – неизвестно, но скорее всего это значение подобрано эмпирически.

Но только если нет критических углов наклона по осям:

The mag orientation is not accurate if the tilt is too high, MAG MODE correction is deactivated after 25deg inclination.

За это отвечает флаг SMALL\_ANGLES\_25. Если он 1, то все норм, если 0 – контроллер (и носитель) сильно наклонены.

Стабилизация курса происходит при любом из условий: либо контроллер не наклонен, либо он в GPS mode.

1. Вычисляются GPS\_angle[PITCH] и GPS\_angle[ROLL] - // the angles that must be applied for GPS correction

Пока не разобрался что это, но вся суть скрыта в формулах ниже:

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;

1. Далее идет PITCH & ROLL & YAW PID

В MultiWii можно использовать любой из двух ПИД-регуляторов: стандартный и за авторством некоего Alex.

1. Вызывается функция mixTable() – формирование значений управления (моторами, серво) на выводах Arduino pro mini. Эту функцию, наверное, придётся переписать, т.к. она написана для самолета. А может нет. Но на выходе эта функция изменяет значения в массиве motor[i] и servo[i].
2. Вызываются writeServos() и writeMotors().

**Режимы контроллера (MODES)**

В оригинальной MultiWii имеется множество режимов полета. Подробно про них можно прочитать, например, здесь: <https://oscarliang.com/multiwii-different-flight-modes-names-gui/>

В SchoolWii оставлены только следующие режимы:

ACRO

PASSTHROU

MAG

GPS HOLD

GPS WP (MISSION)

GPS RTH

По умолчанию при выполнение ARM активируется режим ACRO. Этот режим даже не имеет никакого специального обозначения в структурах и коде.

За состояние активацию режимов отвечают массивы rcOptions[CHECKBOXITEMS], conf.activate[CHECKBOXITEMS], а также переменные auxState и переменная флагов f (flags\_struct\_t f;).

В цикле void loop() в массив rcOptions[CHECKBOXITEMS] заносятся 1 или 0:

for (i = 0; i < CHECKBOXITEMS; i++)

rcOptions[i] = (auxState & conf.activate[i]) > 0;

И уже на основе значения в rcOptions активируется тот или иной режим. При этом, значение режима записывается в переменную флагов состояния (пример для режима PASSTHRU):

if (rcOptions[BOXPASSTHRU]) {

f.PASSTHRU\_MODE = 1;

}

else {

f.PASSTHRU\_MODE = 0;

}

**Работа с GPS.**

В SchoolWii оставлен только функционал работы с приемниками GPS по протоколу NMEA, подключающимися к последовательному порту (UART). Причем, в силу того, что на плате Arduino pro mini имеется всего один аппаратный последовательный порт, который уже занят модулем беспроводной связи на базе ESP8266, то модуль GPS подключается к программному последовательному порту на пинах 3 (SRX) и 4 (STX) – см. схему.

Самой важной модификацией SchoolWii по сравнению с MultiWii является как раз возможности подключения GPS по программному последовательному порту. Для этого пришлось решить такую задачу. Сигнал от приемника радиоаппаратуры представляет собой последовательность импульсов, длительностью примерно от 900 до 2200мкс. Количество импульсов соответствует числу каналов ПРИЕМНИКА. Для шестиканального приемника импульсов будет соответственно 6. За импульсами следует пауза, необходимая для синхронизации. Длительность паузы такова, чтобы вся последовательность «импульсы+пауза» занимали примерно 20мс. В обработчике аппаратного прерывания от приемника (файл RX.cpp) можно увидеть принцип регистрации сигналов от приемника PPM. В обработчике фиксируется момент вызова обработчика и вычисляется разница между во времени между текущим и предыдущим вызовами функции обработки этого прерывания. Если эта разница больше 3000мкс – то фиксируется пауза в PPM сигнале и номер канала chan обнуляется. Если разница лежит в диапазоне между 900 и 2200 мкс, то этот сигнал соответствует каналу chan, значение записывается rcValue[chan], а сам номер канала chan инкрементируется (при приходе следующего импульса новое значение длительности будет соответствовать следующему по счету каналу).

Одновременно с этим, примерно 1 раз в секунду, от приемника GPS приходят пакеты GPRMC и GPGGA протокола NMEA. На скорости 38400 бод, длительность пары этих пакетов составляет чуть менее 40мс. Для дальнейшего понимания происходящего, необходимо вспомнить, как работают прерывания в AVR. Достаточно подробно и в тоже время доступно про это рассказано, например здесь: <http://easyelectronics.ru/avr-uchebnyj-kurs-programmirovanie-na-si-chast-3.html>

Важным выводом из полученных знаний является то, что в момент прихода битов пакетов от GPS мы не сможем достоверно регистрировать сигналы приемника PPM. И наоборот – обработка сигналов приемника PPM может «испортить» прием данных от GPS. Произойдет своеобразное «наложение прерываний» - процессор будет переключаться из основной программы то на один обработчик, то на другой, в зависимости от приоритета и от того, какое прерывание возникло первым. Проверено, что без дополнительных ухищрений это приводит именно к невозможности в момент прихода данных от GPS принять данные от приемника PPM, либо от самого GPS. Чтобы разрешить задачу, необходимо прежде всего выбрать какими данными можно пожертвовать в этот момент – от модуля GPS или от приемника PPM. Разработчики SchoolWii приносят в жертву данные от приемника PPM – потому что это единственный способ в принципе получить данные от GPS (а иначе зачем это все), а также потому, что данные от приемника PPM следуют постоянно, и уже через 40мс, потеряв данным PPM за эти 40мс, контроллер получит новые актуальные данные от приемника. В SchoolWii это реализуется в два шага:

1. Отключение аппаратного прерывания от PPM при вызове обработчика PCINT программного последовательного порта (MwiiSSerial.cpp):

inline void MwiiSSerial::handle\_interrupt()

{

if (active\_object)

{

PPM\_PIN\_ISR\_DISABLE;

active\_object->recv();

}

}

1. Включение аппаратного прерывания от PPM при в основном цикле loop после приема двух пакетов GPS - GPRMC и GPGGA (MultiWii.cpp, в функции void loop()):

#if (GPS)

{

sserialCom();

if (GPS\_Frame) PPM\_PIN\_ISR\_ENABLE;

}

#endif

В функции sserialCom() как раз и происходит установление факта завершения получение нового пакета от GPS, о чем сообщает флаг GPS\_Frame = 1. Т. е. при приходе первого бита пакета от GPS мы запрещаем прерывания от приемника PPM, а по завершению приема сообщений GPRMC и GPGGA – снова разрешаем. Данные от приемника PPM между этими событиями безвозвратно теряются, но для нас это не критично.

Для определения позиции по GPS применяется несложный фильтр, реализованный классом LeadFilter в GPS.cpp.

Фильтр достаточно прост. Он пытается «предсказать» положение объекта навигации через интегрирование ускорения и скорости и прибавлении этих «приращений» к текущей позиции объекта (метод get\_position()). Длительность навигационного цикла GPS (интервал следования пакетов NMEA приемника) составляет 1 секунду. Благодаря фильтру можно получить достаточно близкую оценку мгновенного положения объекта в интервале между пакетами NMEA.

Функции get\_P, get\_I, get\_D возвращает уже вычисленные от ошибки и коэффициентов значения соответствующей составляющей функции регулирования (пропорциональную, интегральную и диференциальную). Причем при вычислении D применяется еще и фильтр нижних частот (по умолчанию полоса пропускания 20Гц).

Функция reset\_PID() обнуляет составляющие.

**Что делается в функции GPS\_Compute() (Main navigation processor and state engine).**

1. Проверяется получение корректного пакета GPS (GPS frame). Под фреймом в данном случае понимается отдельное сообщение NMEA, начинающееся с $GPGGA или $GPRMC. В SchoolWii используются только эти два сообщения NMEA. Поэтому целесообразно настроить приемник GPS на выдачу только этих двух сообщений. Если с момента последнего вызова GPS\_Compute() не было нового фрейма – выходим из функции. Если был – переходим к пункту 2.
2. Проверяем наличие состояние gps.fix и регистрацию пяти (5) спутников. Если условие выполнено, то переходим к пункту 3.
3. Check home position and set it if it was not set.
4. Apply moving average filter to GPS data. Глубина фильтра задается #define GPS\_FILTER\_VECTOR\_LENGTH 5. Этот фильтр работает примерно так:
   1. Считываем последние координаты широты и долготы, делим каждую координату на 10 000 000 –для того, чтобы взять от всей координаты только старшие три разряда – градусы. Потому как MultiWii после чтения координат от приемника GPS преобразует их в целочисленное представление (функция uint32\_t GPS\_coord\_to\_degrees(char\* s)), преобразуя (градусы, минуты, десятитысячные доли минут – все в одном целом 32-х битном числе).
   2. В массив fraction3[axis] помещается разница между точным значением координат и координат, определенных только градусами, с отброшенными минутами и долями минут.
   3. Далее фильтр работает очень просто и хитро одновременно. Буфер фильтра циклический - из него вначале удаляется первое значение из последних пяти (#define GPS\_FILTER\_VECTOR\_LENGTH 5) помещенных в него значений. Потом на место этого первого помещается новое значение равное разницы между точным значением координат и координат, определенных только градусами, с отброшенными минутами и долями минут.
   4. Отфильтрованное значение координат получается, как частное суммы значений в буфере фильтра и глубины буфера (5), плюс значение координат с отброшенными минутами и долями минут. Т.о. при вычислении в фильтре участвуют только минуты и доли минут, а градусы просто вначале вычитаются и после фильтрации снова прибавляются – очень логично ведь ни широта, ни долгота у реального для нас объекта (дрона) не может столь существенно измениться за секунду. А сумма при фильтрации при использовании полных координат может и не поместиться в разрядную сетку 32 битного целого числа.
   5. И дальше, если мы находимся в режиме poshold mode (NAV\_state == NAV\_STATE\_HOLD\_INFINIT || NAV\_state == NAV\_STATE\_HOLD\_TIMED) , и (fraction3[axis]>1 && fraction3[axis]<999), то GPS\_coord[axis] = GPS\_filtered[axis]

*Надо отметить, что описанная фильтрация координат используется только если старшие три разряда разницы между точным значением координат и координат, определенных только градусами, с отброшенными минутами и долями минут больше 1 и меньше 999. Дело вот в чем. Поскольку старшие значения отбрасываются, поэтому при близком нахождении к линии меридиана/параллели есть опасность, что в какой-то момент объект навигации может перескочить этот меридиан/параллель, и произойдет ошибка: были, допустим, на (12).00000005, а скакнули на (11).23599997. Но для навигации это равнозначно смещению позиции на хх.23599997 (он же не учитывает старшие значения), что заставит объект навигации устремиться далеко-далеко на хх.00000005.*

1. Вычисляется последний период навигации:

//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);

Your bearing is calculated to the nearest degree, and it is typically the most direct route from Point A to Point B. Bearing is calculated as an angle measured in degrees in a clockwise direction from true north.

1. Вычисляется bearing и заносится в переменную dir (в 1 град = 100 значений, т. .е Х град\*100).
2. Вычисляется дистанция до ДОМА и заносится в переменную dist (в сантиметрах).
3. Если нет позиции ДОМ, то дистанции и градусу (bearing) до ДОМА присваивается значение 0.
4. Проверяется, если установлена зона fence, если позиция объекта навигации от ДОМА дальше границы этой зоны и объект навигации не в режима RTH (возврат домой), то выполняется Init\_RTH() – запуск процедуры возврата домой.
5. Вычисляется текущая скорость объекта навигации.
6. Обрабатывается автомат состояний навигации.

11.1 Проверяется, что GPS\_mode в состоянии GPS\_MODE\_NONE. Если условие выполнено:

- вычисляется bearing на waypoint;

- вычисляется дистанция до waypoint;

- сalculate a location error between two gps coordinates – это по сути просто разница между соответствующими координатами двух точек. Результат записывается в массив error (error[LAT] и error[LON]).

*Там правда при вычислении error[LON] применяется какой-то масштабирующий коэффициент:*

*// 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*

*GPS\_scaleLonDown = cos(lat \* 1.0e-7f \* 0.01745329251f);*

*Пока не разобрался что это.*

* 1. Далее, проверяется если мы не достигли нужной позиции по координатам и высоте

(if ( !((NAV\_state == NAV\_STATE\_HOLD\_INFINIT) && (alt\_change\_flag == REACHED\_ALT)))), и не находимся в состоянии выполнения посадки (if (!f.LAND\_IN\_PROGRESS)), то

alt\_to\_hold = get\_new\_altitude(); //возвращает промежуточную высоту.

Эта высота может быть целевой, а может быть и «на пути» к целевой.

AltHold = alt\_to\_hold;

*Всю эту часть про высоту для лодок и роверов я выпилил и в SchoolWii ее нет.*

* 1. И дальше идет уже собственно автомат навигации. У автомата следующие состояния:
     1. NAV\_STATE\_NONE: //Just for clarity, do nothing when nav\_state is none
     2. NAV\_STATE\_LAND\_START:

Начало процедуры посадки. Инициализация таймера посадки, вычисление координат удержания позиции (GPS\_calc\_poshold(); //Calculate nav\_lat and nav\_lon from the x and y error and the speed) и переход к состоянию NAV\_STATE\_LAND\_SETTLE.

* + 1. NAV\_STATE\_LAND\_SETTLE:

GPS\_calc\_poshold() и ждем в этом состоянии 5 секунд и потом переходим к снижению NAV\_STATE\_LAND\_START\_DESCENT

* + 1. NAV\_STATE\_LAND\_START\_DESCENT:

В этом состоянии все понятно из кода, единственное тут ставится не до конца понятный флаг

GPS\_BARO\_MODE: 1; // This flag is used when GPS controls baro mode instead of user (it will replace rcOptions[BARO]

…и переход к состоянию NAV\_STATE\_LAND\_IN\_PROGRESS

* + 1. NAV\_STATE\_LAND\_IN\_PROGRESS:

Проверяется не достигли ли земли (check\_land()). Если достигли, то переходим в состояние NAV\_STATE\_LANDED

* + 1. NAV\_STATE\_LANDED: // Disarm if THROTTLE stick is at minimum or 5sec past after land detected

Проверяется, что тяга пульта на минимуме и прошло 5 секунд с момента приземления, то происходит DISARM, все флаги в 0, сброс навигации. Переход в состояние NAV\_STATE\_NONE.

* + 1. NAV\_STATE\_HOLD\_INFINIT: //Constant position hold, no timer. Only an rcOption change can exit from this

В этом состоянии только вызывается одна функция - GPS\_calc\_poshold()

* + 1. NAV\_STATE\_HOLD\_TIMED:

В этом состоянии вызывается GPS\_calc\_poshold() до тех пор, пока таймер не достигнет 1000\*nav\_hold\_time. После этого произойдет переход либо в NAV\_STATE\_PROCESS\_NEXT, либо в окончание миссии с NAV\_error = NAV\_ERROR\_TIMEWAIT.

* + 1. NAV\_STATE\_RTH\_START:

Проверяется на нужной ли высоте объект навигации. Если на нужно, то в качестве следующего waypoint задается ДОМ, и происходит переход в NAV\_STATE\_RTH\_ENROUTE и NAV\_error = NAV\_ERROR\_NONE. Если высота не достигнута, то

GPS\_calc\_poshold(); //hold position till we reach RTH alt

NAV\_error = NAV\_ERROR\_WAIT\_FOR\_RTH\_ALT;

* + 1. NAV\_STATE\_RTH\_ENROUTE: //Doing RTH navigation

Вычисляется скорость и еще ряд параметров и автомат находится в этом состоянии пока не прилетит к waypoint. В этом состоянии проверяется, что объект в радиусе waypoint. И если была команда на посадку – то начинается посадка (переход в состояние NAV\_STATE\_LAND\_START). Если команды на посадку не было (а была команда «зависнуть»), то переход в состояние NAV\_STATE\_HOLD\_INFINIT.

* + 1. NAV\_STATE\_WP\_ENROUTE:

Практически тоже самое, что и NAV\_STATE\_RTH\_ENROUTE, только посложнее. В зависимости от того, что делать в этом WP согласно миссии, в то состояние и производится в последствии переход. В большинстве случаев – просто в NAV\_STATE\_PROCESS\_NEXT.

* + 1. NAV\_STATE\_DO\_JUMP:
    2. NAV\_STATE\_PROCESS\_NEXT: //Processing next mission step

Итогом работы GPSCompute() являются:

Новое состояние NAV\_state, NAV\_error возможно, меняются некоторые флаги в переменной f, ответственные за GPS и навигацию. Многое из этого списка:

extern uint32\_t wp\_distance;

extern int32\_t target\_bearing;

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

// 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)

extern uint8\_t land\_detect; //land detector variable

extern mission\_step\_struct mission\_step;

extern int16\_t nav[2];

**Как происходит работа с waypoint-ами (WP, точками пути).**

WP хранятся в EEPROM (вид энергнонезависимой памяти в микроконтроллере Arduino pro mini). Для всех плат предельное число WP равно 255 или 254 (8 бит и там 1 или 2 зарезервированы). Но для каждой разновидности платы (в нашем случае - Arduino pro mini) реальное число определяется размером свободного места в EEPROM. Это считает функция uint8\_t getMaxWPNumber() модуля EEPROM.h. Функции storeWP() и recallWP() соответственно сохраняют в EEPROM и считывают из него очередной WP. storeWP() вызывается например, когда приходит команда MSP\_SET\_WP по протоколу MSP (MultiWii Serial protocol). Т. е. вы в программе на ПК рисуете точки миссии, нажимаете на кнопку, и программа отправляет точки миссии со всеми параметрами на контроллер MultiWii, используя протокол MSP и конкретно команды MSP\_SET\_WP. Контроллер принимает эти команды, расшифровывает содержимое пакета и помещает WP в EEPROM. И когда вы нажимаете на кнопку "запустить миссию", или подаете команду с пульта за запуск миссии, контроллер считывает точки миссии уже из EEPROM. В принципе так сделано во всех подобных контроллерах. даже если потеряется связь с постом оператора – объект управления (машина или лодка в нашем случае) продолжат исполнять миссию без каких-либо проблем.

Каждая точка WP описывается структурой «mission\_step\_struct» (types.h):

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;

Возможные значения поля «action» (MultiWii.h):

//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

Возможные значения поля «flag» (MultiWii.h):

#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

**Как работает IMU.**

Главная функция модуля IMU – ComputeIME().

1. Считываются данные акселерометра – последовательный вызов функций ACC\_getADC() и getEstimatedAttitude().
2. Считываются данные гироскопа – вызов функции Gyro\_getADC()
3. Для всех трех осей (axis) значения gyroADC[axis] копируются в gyroADCinter[axis].
4. Вызывается функция annexCode()
5. Выдерживается пауза 650 мкс с момента последнего считывания Gyro\_getADC() и снова производится чтение Gyro\_getADC(). В комментарии к коду сказано, что значение 650мкс определено эмпирически. Теперь в массиве gyroADC[axis] новые значения, а в gyroADCinter[axis] – старые (с первого вызова Gyro\_getADC()).
6. Для всех трех осей производится приращение: gyroADCinter[axis] = gyroADCinter[axis] + gyroADC[axis].
7. Вычисляется взвешенное значение

imu.gyroData[axis] = (gyroADCinter[axis]+gyroADCprevious[axis])/3;

В знаменателе стоит 3 по той причине, что по факту мы складываем три значения от гироскопа: одно значение в самом начале функции ComputeIMU(), второе через 650 мкс и третье – значение из предыдущего вызова функции ComputeIMU().

1. В gyroADCprevious[axis] сохраняются текущие значения gyroADCinter[axis] деленные на 2.

*По пунктам 5-8. Логика такого усреднения мне достоверно не известна. Скорее всего это какие-то эмпирические исследования разработчиков MultiWii.*

1. Если включена опция GYRO\_SMOOTHING (по умолчанию выключена), то данные в imu.gyroData[axis] дополнительно фильтруются сглаживающим фильтром. Вот описание этой опции из комментария к коду:

*«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.»*

Могу предположить, что для лодки и для ровера эта опция может быть полезной.

Функция annexCode() на шаге 4. Это относительно небольшая функция, в описании которой сказано:

*«This code is excetuted at each loop and won't interfere with control loop if it lasts less than 650 microseconds.»*

Подробный анализ пока затруднен, но бегло можно сказать, что эта функция вообще не связана с IMU напрямую. Разработчики просто удобно запихали ее тело функции ComputeIMU() для вызова не чаще, чем в 650 мкс. Через диспетчер это могло бы не получиться сделать – любая из функций диспетчера может выполнится быстрее или медленее и тогда происходил бы недетерминированный вызов этого кода. Что в ней делается:

1. Динамическая подстройка коэффициентов ПИД-регуляторов для PITCH и ROLL.
2. Считывание и обработка значения напряжения аккумулятора.
3. Обработка состояния светодиода состояния на пине 13.
4. Обработка состояния внешнего LED FLASHER.
5. Обработка состояния калибровки акселерометра (только обработка состояния – не сама калибровка).
6. Обработка коммуникаций по аппаратному и программному последовательным портам (чтение пакетов спорта и их разбор) – вызов функций serialCom() (для связи на базе ESP8266) и sserialCom() (для приемника GPS).

**Как работает обрабатываются сигналы от радиоприемника, подключенного по протоколу PPM**

Радиоприемник PPM подключен к пину 2 Arduino pro mini. И обработка сигналов приемника производится с использование внешнего аппаратного прерывания с вектором 0 (EXTERNAL ISR).

Обработчик прерываний просто считает импульсы и их длительность. Никакие счетчики микроконтроллера или иные ресурсы для этого не применяются.

На выходе обработчика прерываний массив rcValue[chan]. Значения в нем от 900 до 2200.

**Как выполняется формирование выходных сигналов ШИМ.**

Всего в MultiWii на базе платы Arduino pro mini 8 выходов PWM (Output.cpp). По умолчанию для самолета, ровера и лодки установлен один выход на мотор и 7 выходов для сервоприводов (def.h).

16-и битный счетчик TCNT1 и 8-и битный TCNT2 применяется для формирования выходного сигнала PWM на моторы.

Управление сервоприводами осуществляется с использованием таймера TCNT0 и прерывания CTC этого таймера.

На Arduino pro mini у нас есть только software servo.

Моторы 1 и 2 на пинах 9 и 10 соответственно. Сервоприводы на пинах А2, 12, 11, 8 и 3. Для самолета распиновка такая:

Wing1: 12

Wing2: 11

Elev: 10

Throttle: 9

Rudd: 8