# R3PB AIR TRANSPONDER

Ссылки на полную документацию на github.com: https://github.com/R3PB/AIR-TRANSPONDER

Идея проекта возникла, как говорится, не на пустом месте. Так уж вышло, что планеризмом, стал заниматься с 2001 года. По началу подлётывал на разных аэродромах, пока в 2007-ом году, по иронии судьбы прям под боком, где я теперь проживаю, обнаружил самый известный в нашей стране аэроклуб «2-ой МАК». С тех пор, можно сказать, осел там же.

Авиационный парк нашего аэроклуба простой, всё как у всех. Остатки былой славы Советского прошлого: Вильги, Бланики, Яки, Янтари которые бережём, своими руками восстанавливаем, и при надобности ремонтируем. Благо коллектив собрался очень дружный и толковый.

Рисунок 1 Бланик L13

Учимся летать, а точнее парить на планёрах. Тут тоже как в любом другом АУЦ, программа обучения – прежде всего буксировка и полёты по кругам до «тошнотиков», несложные пилотажные зоны включая штопорные, и самое главное ради чего всё это – парение. Это когда у тебя немного перья появляются, а инструктор тебя научил восходящие потоки пятой точкой чувствовать при этом самостоятельно можешь лететь. Но увы, летать далеко не отпускают. И это понятно, так как парение — это целая наука! Не рассчитал – не долетел. Сел в поле, хорошо если не на лес. Если что напомню, планер – это всё же как самолёт, но только без мотора. L13 Бланик весит 295 кг без пилота. Разложить об кочки при посадке очень не хочется. Потому то руководитель полётов(РП) внимание уделяет немалое тому, где мы парим, как парим, на какой высоте, в каком потоке, на каком удалении, в каком месте – за стартом или впереди старта… Однако, не всегда есть возможность визуального контроля за бортами. Это бывает по различным причинам. Зона-парение, площадка, у кого маршрут и планёр скрывается с глаз. Есть конечно радиосвязь. Регламент обмена, доклад, и всё такое… Хорошо, но всё же, необходимо больше. Иногда, казалось бы уже «всезнающие» планеристы, всё же увлёкшись потоком забываются, и уходят далеко за старт, да и другие причины так же бывают - азарт во чтобы то ни стало, найти тот самый термик... Как раз из-за подобных ситуаций, родилась идея того, чтобы можно было как-то следить за бортами всех пилотов, особенно курсантов, не обладающих необходимым опытом. Что бы РП мог всегда увидеть, подсказать, дать команду на выполнение. В общем дополнительно обезопасить весь процесс обучения. Так и появилась идея, установить на борт что-то, что может передавать параметры.

Конечно же первое что пришло в голову – это система ADSB и прочие xDSB. Всё это круто, но нереально дорого для нас, да и места нет для такой сложно аппаратуры, ко всему прочему имеющей вес. Другие коммерческие решения, так же оказывались дорогостоящими, либо не соответствующими нашей специфике, да и попросту не удобными.

Ну чтож. Пришлось, применить скромные радиолюбительские навыки.

Первое что было сделано – определено техническое задание от которого и оттолкнулся.

Итак, ТЗ:

1. Первое и самое важное – **никаких сторонних операторов связи**, никакого интернет, GSM, 5G и каких-либо других сетей связи, включая радиолюбительскую APRS. Только своя базовая станция(БС) на земле и транспондер в воздухе, за исключением GPS/ГЛОНАС приёмника – тут уж никак.
2. Определено **количество транспондеров** в системе. Так как авиапарк не большой, то и количество было определено по количеству бортов – **10 транспондеров в системе** (хотя можно и больше).
3. Определён максимальный **радиус действия** системы. Основываясь на опыте полётов наших старожилов, и учитывая требования безопасности полётов, за основу было взят круг радиусом **35 км** (думаю можно больше в раза два). Это расстояние, в которое попадают всё значимые населённые пункты на карте, дальше которых на парение мы не улетаем. Ну и примерно 100 километровый треугольный маршрут для зачёта. Если учесть, что речь идёт о обучении то новички редко за 8 км от старта уходят, то более чем достаточно.
4. Транспондер летательного аппарата должен в автоматическом режиме **передавать**:
   * **Место**
   * **Высоту**
   * **Скорость**
   * **Курс**
   * **Скороподъёмность**
   * **Номер борта**
5. Транспондер должен **передавать данные** не реже **1 раза в 4 секунды**.
6. Мощность транспондера в пределах 1 Ватта. Питание автономное.
7. **Базовая станция**(БС) должна **принимать данные** со всех транспондеров **одновременно**.
8. **Данные** полученные БС должны быть **записаны в базу данных**.
9. **Данные** полученные с транспондеров должны **отображаться на карте в режиме реального времени**.
10. **Отображать данные** можно на компьютере планшете смартфоне в **WEB браузере**. Повторюсь, НИКАКОГО ИНТЕРНЕТА!!! Всё оффлайн.
11. **Базовая станция автономна**: батарея, плюс антенна на СКП. **Подключение** компьютера к БС **с помощью Ethernet или WIFI**.
12. Решение должно быть максимально бюджетным, при этом габариты транспондеров должны быть такими чтоб ничему и никому не мешали.

Первый пункт многое перечеркнул. Если бы не этот пункт то, готовых решений, на просторах интернета, сходу найти можно с дюжину. Всяческих GPS трекеров любительских. Про коммерческие и вовсе молчу. Но всё работает в сетях операторов связи. Это минус. (*Хотя на будущее включить использование такого вида связи в качестве неосновного аварийного всё же стоит*).

Так же не стал привлекательным вариант с использованием APRS системы, а именно с развёртыванием своей сети не включенной в общую сеть. Слишком громоздко. Да и требования пунктов 2 и 4 исключили возможность использования этого протокола. Представьте 10 бортов должны циклично каждые три секунды передавать данные в которых кодировано 6 параметров. При этом они не должны мешать друг другу.

Изучено было много материала. Камнем преткновения стал протокол передачи данных. Конечно, я много смотрел в сторону AX25. Но как по нормальному совместить передачу данных через радиоэфир на малой мощности, от постоянно движущегося объекта, на условно большое расстояние, да так чтоб гарантированно, при этом объектов сразу 10. И чтоб данные отображались действительные - актуальность 4 секунды?

Пришло понимание того что передаваемый пакет должен быть коротким. Исходя из п. 10 и 5 получается:

4сек / 10бортов = 400мсек 1борт

Пункт 12 так же определил выбор используемых частот - 433 или 144 МГц.

Передать необходимо:

1. МЕСТО с определённой точностью - это данные GPS широта и долгота. Полный передаваемый параметр системой GPS — это два 9-тизначных числа с плавающей запятой!!!
2. ВЫСОТА с определённой точностью. Полный параметр, передаваемый системой GPS 6 знаков
3. Скорость. Закладываю 3 знака
4. Курс. Так же 3 знака
5. Скороподъёмность. 3 знака
6. Номер борта. 2 знака
7. Незабываем что необходимо хоть какое-то избыточное кодирование, плюс преамбулы начала передачи пакета, контрольные суммы в конце передачи. Назовём это «Транспорт» + 50% ресурса – это примерно 15 знаков сверху.

В итоге что имеем: Необходимо кодировать и передавать в эфир 50 байт, на частоте (433) 144МГц малой мощностью, со скоростью 400 мсек (примерная скорость 120 байт в секунду).

На помощь пришёл его величество случай. Как-то в один из зимних вечеров, пришёл ко мне товарищ и принёс мне простенькую самодельную метеостанцию, сделанную на ESP1866. Платка ESP и датчик BME280 всё без корпуса. Просто незатейливо, но всё работало. Идея со сбором метеоданных с того момента начала развиваться. Хотелось сделать больше, собирать больше различных данных. Явно не хватало анемометра и уровня осадков. Стал копать в этом направлении что есть. Перебрав опять же кучу материала, набрёл на статью об использовании таких датчиков - OREGON SCIENTIFIC WRT810 и PCR800. И надо же, эти датчики есть в достаточном количестве в продаже, и даже не дорого! Заказал сразу два комплекта себе и товарищу :). Но вот дальше интересно. Проблема в том, что к ним нет базы. Они беспроводные, работают по радиоканалу 433 MHz. Нужно было строить. В итоге был построен радио-модуль для приёма данных с этих датчиков, но самое главное не это, а то что для этих датчиков не было поддержки в коде. На просторах интернет наткнулся на пост автора Invandy, в котором он описывает работу различных коммерческих датчиков, а также какие-то свои собственные разработки датчиков. На мой взгляд, это лучшая статья, и лучшая любительская работа. Ему спасибо! Начал изучать, и как водится допиливать этот проект под свои задачи. Всё получилось. Датчики были встроены успешно в проект и по сей день работают. Увидеть результат можно на narodmon.ru ссылка на проект git-lira.net. Результат впечатлил. Меня тут осенило. Стало понятно то, что идея применима к тому что я искал столько времени. Просто, дёшево и сердито. Главное повторяемо!

Итак, за основу был взят код из проекта метеостанции INVANDY https://github.com/invandy/Oregon\_NR. Существенно переработанный. Но за идею ему ещё раз спасибо! Общая концепция проекта приобрела вот такой вид:

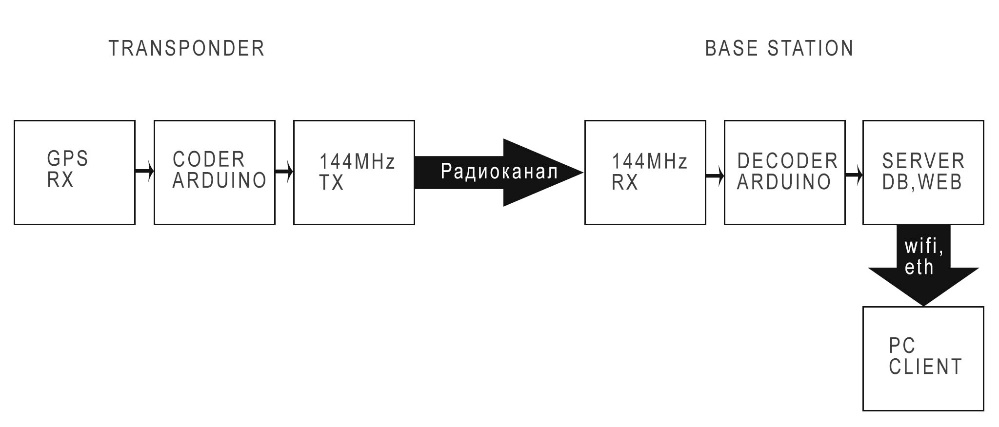


Рисунок 2 Блок-схема проекта.

Как вы понимаете, исходя из схемы, транспондер только передаёт, а БС только слушает. Между собой они не договариваются. Это минус, но зато время передачи пакета существенно сокращается, да и потом, становится ниже вероятность того, что кто-то кого-то перестал слышать и в итоге пакет либо не дошёл, либо был отвергнут из-за квитанции… Остаётся только обрабатывать возможные ошибки на стороне БС и проверять контрольные суммы приходящих пакетов. На стороне БС ресурс производительности условно можно и не жалеть.

Прошу принять во внимание тот факт, что речь далее пойдёт о передаче данных на разрешённых радиолюбительских частотах диапазона 2 метра, с мощностью не превышаемой разрешённую. То есть, в соответствии с радиолюбительским регламентом. Следовательно, для использования этого диапазона вы должны иметь радиолюбительский позывной(лицензию). Альтернатива - 433 MHz гражданский диапазон. Так же хочу отметить что согласно того же регламента, запрещено шифрование данных передаваемых радиолюбителями в эфир. Поэтому не стоит путать термины КОДИРОВАНИЕ и ШИФРОВАНИЕ. **В данной статье речь идёт только о кодировании.**

## Формирование пакета.

Ну что же всё есть. Вооружившись начальными данными, и небольшим опытом обрисуем схему формирования пакетов.

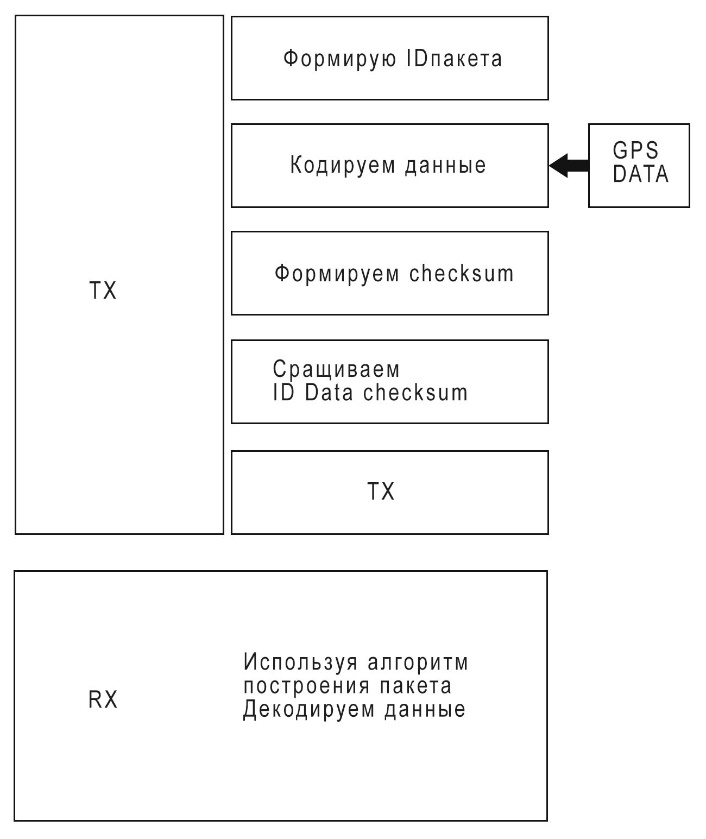


Рисунок 3 Формирование пакета.

Следующее что можно было оптимизировать это код самого транспондера. Слишком велико значение lan и lon получаемые приёмником GPS. Для нашей специфики полётов такая высокая точность просто избыточна. Поэтому принято решения сократить кодируемые координаты до 3х знаков после запятой, да и саму запятую необходимо бы убрать избавившись тем самым от неудобного числа с плавающей запятой. Сократили lat lon до 10 символов, помимо символов «+» и « – » их кодировать достаточно одним битом 1/0 .

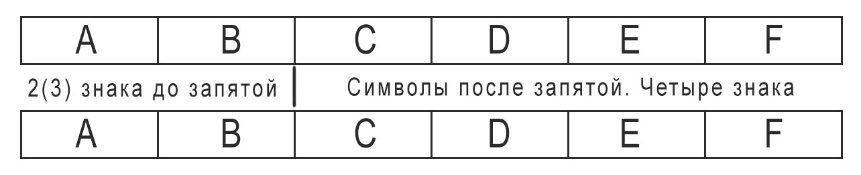


Рисунок 4 Формирование данных lat, lon

Подобные оптимизации касаются и других переменных значений. Подумав еще немного пришёл к выводу, что курс можно не передавать, а рассчитывать его математически на стороне БС, то же самое и сделать с курсовой скоростью, ибо скорость в потоке всё равно взять в Бланике просто нечем. Итак, получилась вот такая расчётная схема кодирования разрядов:

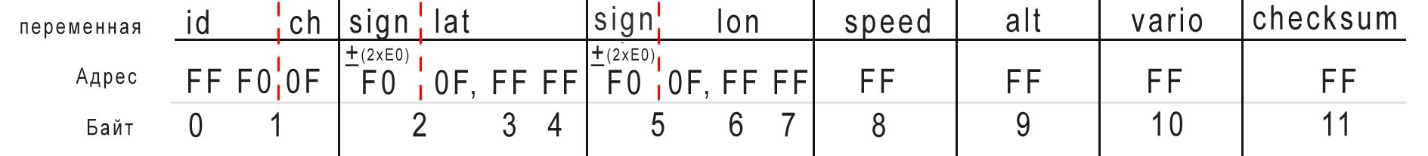


Рисунок 5 Схема пакета V2.

Можно сэкономить байт если использовать пустую часть нибла в переменной lat. В этом начальном нибле можно хранить флаги значений (+,-) для параметров lat, lon, vario. Использовать можно только последних три MSB.

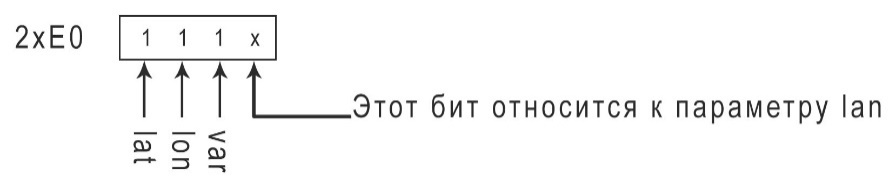


Рисунок 6 Использование свободного бита.

Основываясь на параметрах полёта планёра Бланик L13 так же ограничил остальные параметры транспондера:

|  |  |  |
| --- | --- | --- |
| Параметр | Предел измерений | Адресация/размер |
| Lat | 179,999 | 0F FFFF |
| Lon | 89,999 | 0F FFFF |
| Speed | 255 km/h | FF |
| Alt | 2550 m  (255x10 - множитель 10) | FF |
| Vario | 15 m | 0F |

Если кодировать знаки + - в конец переменных lat, lon – а именно: использовать по одному биту значения 0, 1 как флаги знаков + -. Пространство alt должно быть FF, так как не хватает места для указания параметра alt. Можно взять ещё один свободный бит из переменной lon.

Тесты и отладки показали то, что можно экономить байтовое пространство, и задействовать все свободные биты которые не использовались бы при заполнении. В таблице параметры lat, lon представлены адресным пространством начиная с 0F Т.к. при максимальном числе, например, lat = 179 я могу использовать всего три разряда этого нибла. Нулевой бит этого нибла(2x80) остаётся незадействованным. По этому я могу использовать его как флаг: 1 это «-» , 0 останется «+». Напомню, что в системе GPS, в зависимости от полушария, значения могут быть и минусовыми.

Так же поступил и с ещё одним параметром var (vario) указание скороподъемности так же требует разных знаков. В данном случае параметр lon ещё более короткий чем lat. И в старшем байте остаются целых два свободных бита (4x08, 4x04), чем я и воспользовался. В итоге родился пакет версии 6.

На заметку: Для передачи координат lat/lon, необходимо было избавиться от запятой, так как тип float занимает большое пространство, что увеличивает размер пакета, тем самым снижая его надёжность, а также снижается общая производительность системы. Поэтому я кодировал значение целого числа отдельно от дроби(мантиссы).



Рисунок 7 Пакет v6.

Ужимал пространство пакета, как только мог. Как можно увидеть из этой схемы, параметры curs и vario так же разделили между собой бит. Параметры alt кодируется значением от 0 до 255. На стороне БС этот параметр умножается на 10. Точность измерения в этом случае +/- 10 метров. Не очень то, но пришлось сэкономить. Параметр vario так же ограничен значением +/- 15 метров секунду. Для планеристов такие показатели во время полёта бывают крайне редко, как правило критические. Например, попал в грозовое облако, а вариометр с пределом всего 10 метров в секунду. Напомню, что полёты в облаках запрещены :).

### Идентификаторы пакета

ID – идентификатор пакета. Содержит версию протокола. Так же ID содержит номер канала(timeslot) адрес 1x0F.

Небольшое пояснение по принципу формирования каналов транспондеров. Использован timeslot – то есть каждому транспондеру задаётся интервал и своё время передачи, в общем цикле равном 4 секундам. Например, первый транспондер начинает передачу на нулевой секунде, продолжительность 0,4 сек. Второй на 0,5-ой секунде, продолжительность 0,4 сек. Третий на 1-ой секунде и т.д…



Рисунок 8 ID пакета v6. Определяется 4-мя ниблами.

Можно начать идентификацию с адресного пространства AAA + номер канала(ch). Тогда получаем:

1. AAA1 – протокол X AIR1 (канал1)
2. AAA2 – протокол X AIR2 (канал2)
3. AAA3 – протокол X AIR3 (канал3)
4. AAA4 – протокол X AIR4 (канал4)

И т.д..

## Стендовые испытания

Считаю нужным сказать пару слов об стендовых испытаниях, так как ошибки допущенные на этапе разработки, больше помогу понять то, как устроен пакет. В стендовых испытаниях уже учувствовал декодер. Для этих целей поначалу я использовал БС от проекта метеостанции, доработал протоколы, допилил что нужно и, пакеты различных версий я уже мог отлаживать. И так дело дошло до очередных стендовых испытаний. Выявлены следующие проблемы:

Неверно был расшифрован пакет

* Неверно была определена длинна пакета
* Неверно была определена CRC checksum

В итоге оказалось, что длинна пакета составляла 17 ниблов, а это 8 байт и 1 полубайт. Перепутал количество ниблов с длинной – нумерация начинается с нуля. То есть первый нибл под номером НОЛЬ. Кроме этого сократилось пространство для полезной информации из-за неправильно подсчитанной мной длинны checksum. Длинна checksum составляла 5 ниблов, то есть 2,5 байта. Это особенность работы протокола OREGON V3. И Так как я в своей изначальной работе с метеодатчиками имел дело с допиливанием хорошо изученного протокола для PCR810, то результатом такой ошибки стало то, что в пакет не вошло 4 параметра:

* Speed FF
* ALT FF
* VARIO F0
* CURS 0F

Всего три байта.

Частично решено

1. Сократил CRC до одного байта, занявшего пространство FF тем самым высвободил один байт.
2. Увеличил длину пакета на 1 нибл (полбайта). И сдвинул CRC вправо (окончание пакета).

Высвободившееся пространство использовал для параметра alt – 0XFF. Ещё осталось полбайта vario. (При необходимости можно будет сдвинуть CRC ещё на полбайта вправо для параметра curs).

## Пакет версии 7.

Результатом отладки стал пакет версии 7. Всего 19 ниблов. Если считать от 0-го, то последним будет 18-ый нибл.

Вид пакета v7. Пример: AAA10D5C909206170F5

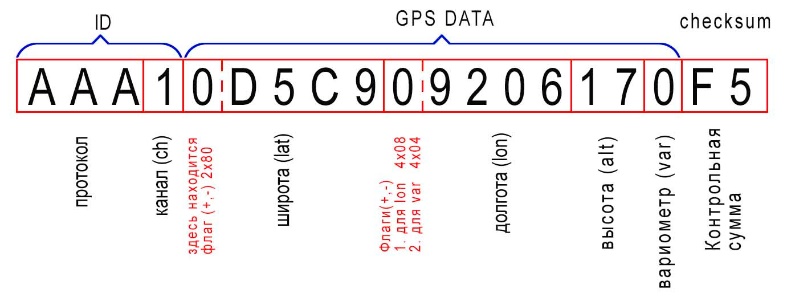


Рисунок 9 Пакет v7

## Полевые испытания

Наконец наступил долгожданный лётный сезон. И можно приступать к полевым испытаниям. Сразу оговорюсь, что зимой-весной, било подготовлено и отлажено всё необходимое железо и ПО, но об этом потом…



Рисунок 10 СКП

Развёрнута базовая станция Старенький Macbook в качестве сервера. Приёмник - их было два, тестировались по очереди: Radoshack и YAESU FT-11r В качестве преобразователя уровней двухканальный ОУ от оптического привода. Декодер – их тоже было два: ESP8266 для беспроводного соединения с сервером, и Arduino UNO с ETHERNET шильдом для проводного соединения с сервером. Вариант на ESP8266 мне понравился больше, так как от него не было цифрового шума, и как ни странно обработка пакетов на ESP8266 была быстрее! Для беспроводного соединения рабочих станций с сервером, использовался WIFI роутер TP-LINK. Ну и в качестве рабочей станции – ноутбук, или смартфон, да что под рукой было…:) Да и самое главное – антенна. Внешняя двух диапазонная антенна OMNI 144/433 MHz, с круговой диаграммой, и с коэффициентом усиления, по паспорту, 3,5/4 db.



Рисунок 11 Транспондер на борту Бланик L13

Транспондер был установлен на один из планёров. Кодер на Arduino Nano. В качестве трансмиттера YAESU FT-11r. Антенна была штатная резинка. На фотографии справа можно увидеть батарею, FT-11r, и беленькая коробочка. Вот оно в таком виде отлетало. Обзору и пилотированию не мешало, приём был хорошим :).

В результате полевых испытаний была обнаружена ошибка передачи параметра alt. При достижении высоты 255 метров, счётчик (указатель высоты), обнулялся. После чего отсчёт вновь продолжался с нуля. И так каждый раз до 255 и вновь сброс счётчика…

Ошибка была на стороне приёмника (в библиотеке get\_altitude) были установлены неверные типы данных – byte. Для корректного приёма этого было достаточно, но для дальнейшей передачи параметров больше 255, нужно было увеличить полученное число на один разряд путём перемножения полученного значения на 10. В результате разрядность числа увеличивалась, превышая размеры указанного типа данных – byte (всего 8 бит), не вмещающиеся в 8 бит старшие разряды отсекались. Так счётчик обнулялся через каждые 255 метров.

Решение было таким: Изменён тип данных всей библиотеки get\_altitude до 16 бит – тип данных word.

## Итоги полевых испытаний

Итоги первых полевых испытаний порадовали. При излучаемой мощности 0.1 Ватт (не опечатка, да всего 100 милливатт) уверенны приём был в радиусе 8 км от точки. Жаль в те дни не было парящих маршрутов. Ходили вокруг да около старта те самые 8 км. От экрана я не отрывался. Следил за всеми перемещениями метки.

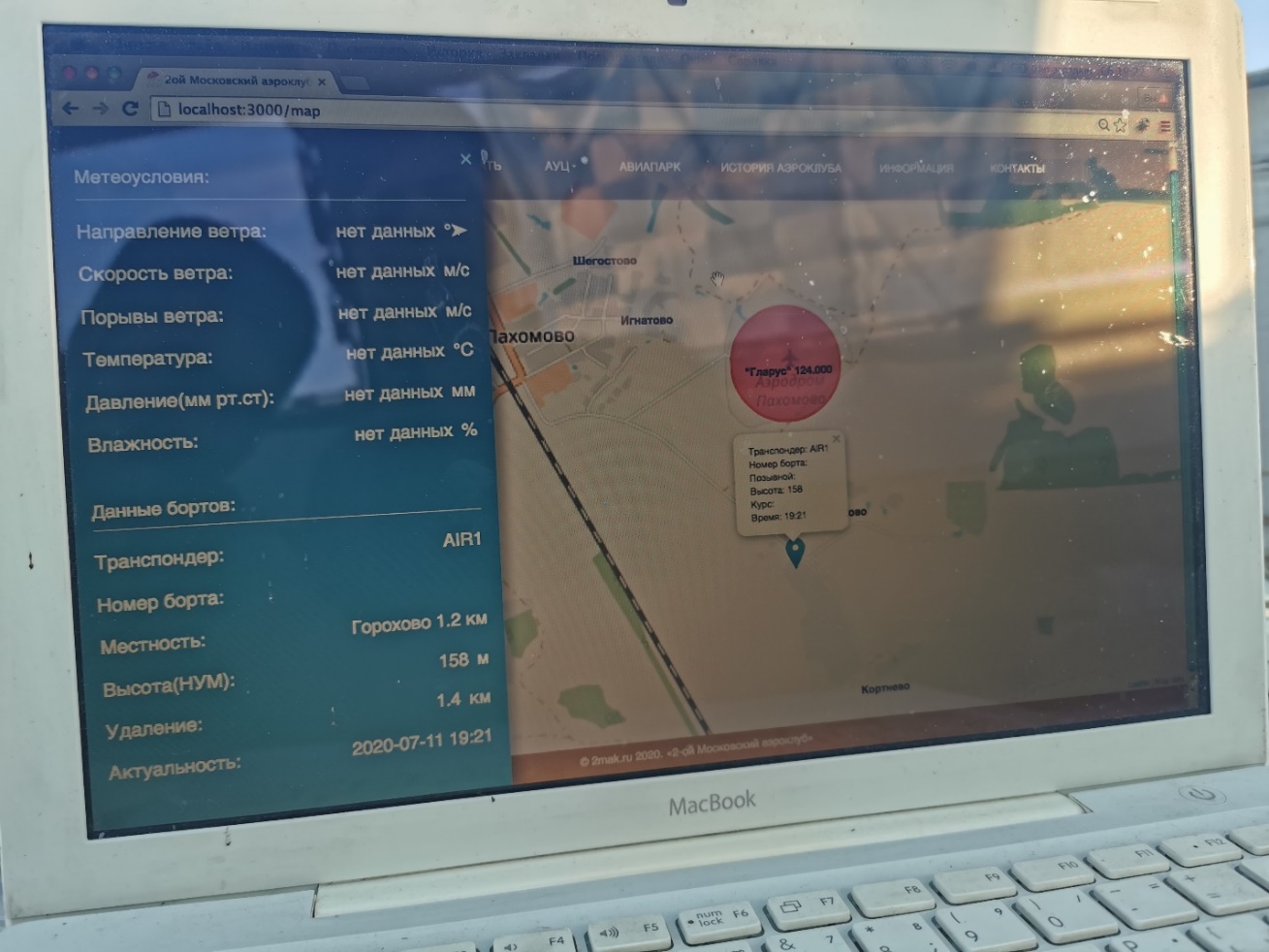


Рисунок 12 WEB интерфейс базовой станции. Отображение на карте и вывод параметров полёта.

В интерфейсе, если обратили внимание, я сделал так же метеосводку. Эти данные берутся с тех самых датчиков OREGON и BME280 которые устанавливались так же на аэродроме. На этой фотографии данные отсутствуют, но все датчики я так же оттестировал в полевых испытаниях. Тут проблем не было.

Тестирование проводилось на частоте 144.810 MHz мощностью 100 милливатт. Активные тесты начинаются с конца мая. Несмотря на небольшую мощность, высОты на которых работают транспондеры, позволяют излучать на приличные расстояния. Так что коллеги радиолюбители, на частоте 144.810 MHz с субботы по воскресенье каждую неделю до октября, могут послушать сигналы транспондера и при желании, декодировать.

Надо пояснить почему всё-таки 100 милливатт. Минимальная мощность излучения FT-11r составляет 300 милливатт, максимальная 5 Ватт. Но батареи моих Yaesu давно вышли из строя. Разобрав корпуса батарей я установил в них li-ion батареи на 3,7 вольт. При таком напряжении эти радиостанции не выдают более 100 милливатт. В планах использование других источников питания и тестирование на 500 милливатт и на 1 Ватт.

По итогам полевых, боевых, лётных (как угодно...) испытаний были сделаны доработки:

15.07.20 Дополнен код приёмника. Теперь приёмник одновременно может принимать несколько транспондеров. Префиксы транспондеров AAA0 – AAA9

22.07.20 Исправлен код транспондера. Транспондер может передавать, соответственно кодировать пакеты AAA0 – AAA9

Замечание: Указатель префикса(заголовок) можно только в файле transponder\_air.h Необходимо вынести define transponder в пользовательский \*.ino файл – скетч.

## Каналы транспондеров или time\_slot function

Задача: Описать функцию time\_slot которая будет определять время начала передачи пакета, каждого транспондера, в зависимости от времени (префикс транспондера содержит номер канала – номер таймслот).

Реешние:

Исходя из того, что имеем:

qTX = 10 - кол-во транспондеров AIRx (AAAx) – Начальное условие

iTX = 4 сек - интервал передачи – расчётное условие, смещение времени не более 4 сек.

dTX = 0,4 сек - длительность передачи 1-го транспондера. Рассчитано и оптимизировано.

Тогда очевидно:

iTX / qTX = dTX;

dTX = 0,4 сек

Таким образом необходимо добиться что бы каждый транспондер начинал передачу в своё заданное время с циклом dTX – 4 секунды. Для этого необходимо синхронизировать таймеры транспондеров с GPS time с точностью не ниже 0,1 сек. После чего отсчитывается 0-я секунда каждой минуты реального времени, от которой начинается отсчёт таймера. Например,

|  |  |  |
| --- | --- | --- |
| Транспондер | Начало передачи | Смещение (offset\_time) |
| AAA0 | 0-я секунда | 0,0 сек |
| AAA1 | 0,4 сек |
| AAA2 | 0,8 сек |
| И т.д. | … |
| AAA9 | 3,6 сек |

Рисунок 13 Формирование каналов (Time slot).

Порядок инструкций:

1. Синхронизация с GPS
2. Получение GPS\_time\_seconds
3. Определение нулевой секунды любой минуты.
4. Начало отсчёта от нулевой секунды
5. Начало передачи пакета, по достижению счётчика.

Да. Скорее всего, при такой реализации, мы получим пару проблем:

1-я проблема. Из-за особенности работы Arduino возникнет разность нулевой отметки времени. Отсчёт millis у всех ардуин хоть немного, но отличаются друг от друга. Из-за этого, сразу после включения, различные сигналы транспондеров будут накладываться друг на друга в начальном минутном интервале даже после синхронизации с GPS\_time.

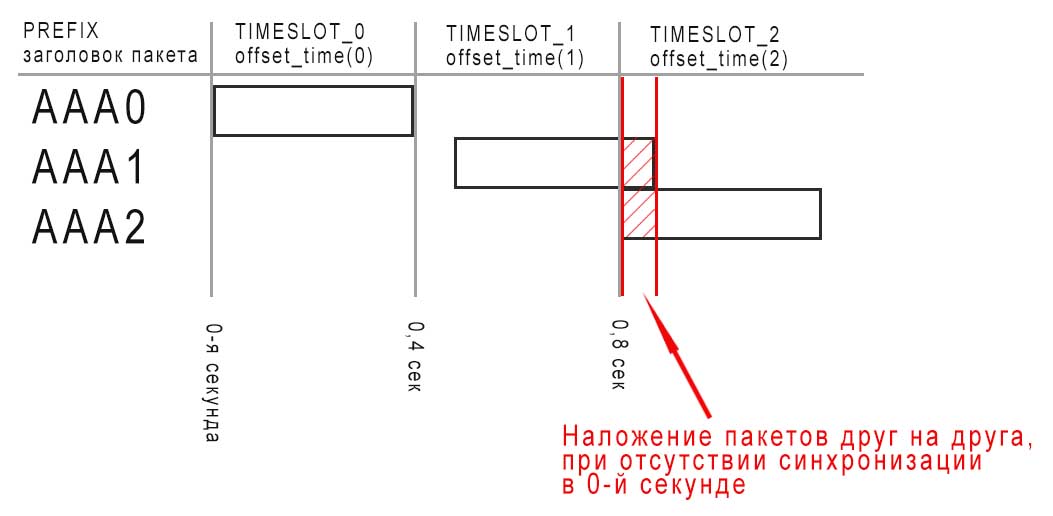


Рисунок 14 Timeslot. Не синхронизированный пакет.

2-я проблема. При выполнении различных инструкций таймер счётчика, может периодически теряться.

1: Инструкция 50% ресурса

2: Инструкция 50% ресурса

3: Инструкция offset\_time = doit\_now – возвратит FAIL

Потому что из таймера в память не пришло то значение счётчика секунд, которое было нужно, из-за того, что система была занята в этот момент другой процедурой. И попросту пропустила это время.

Решение по второй проблеме, оказалось простым. Можно было использовать допуски +- 0,2 сек. Но и этого не понадобилось. Опасения не подтвердились. Тесты показали, что заданные отметки времени своевременно отрабатывались инструкциями без пропусков.

А вот по первой проблеме решение было таким:

1. Получить время с точностью до 0,001 секунды
2. Представить время в разрядной десятичной форме
3. Определить разряды отвечающие за минуты, секунды.
4. Старшие разряды обнулить и не использовать. Таким образом, полученное переменное значение, повторяющееся в цикле, будет синхронно с GPS\_time.
5. Привязать millis counter к полученному значению.
6. Задать смещение от нулевой секунды (offset\_time)

Внимание, тут нас поджидает небольшая проблема! :) Дело в том, что значение time (параметр, отдаваемый библиотекой TYNY GPS и вообще приёмником GPS NEO M6), представляется в таком виде: hh min sec 0.00. То есть, миллисекунды всегда равны 00!!! В библиотеке они есть, но GPS приёмник их не может отдать из-за своей ограниченной функциональности.



Рисунок 15 Tiny GPS. Пустое значение millisec.

Применил, как мне кажется, весьма элегантное решение.

Для наглядности кусок из скетча:

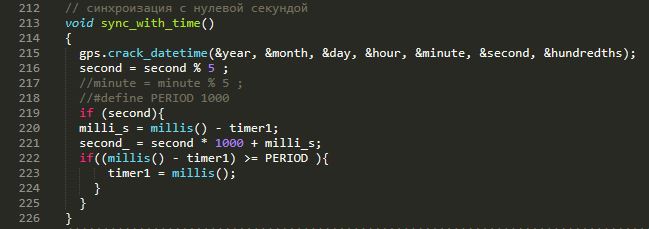


Рисунок 16. Синхронизация с нулевой секундой.

Так как преобразование из DEC в HEX с сохранением вида числа (например, 59 DEC -> 0x59 HEX) затратное по количеству процедур и бла-бла-бла (сорри), было решено использовать такую функцию - % (процент). Напомню, этот оператор заносит только остаток деления в переменную. Поэтому решение получилось очень простым. Взять параметр second, передаваемый gps.crack\_datetime, и поделить его на 5 с сохранением остатка в ту же переменную second.

Так мы получаем цикличность параметра second, получаемого от GPS time, не от 0 до 60, а от 0 до 5-ой секунды с шагом 0,2 сек. То есть 0, 2, 4, 6, 8, 0 - каждые 5(пять) секунд! И так по кругу.

Таким образом получим прохождение через 0-ю секунду каждые 5 секунд. А так же, синхронизация пакетов происходит относительно быстро, за один цикл передачи транспондера - 5 секунд.

Замечу, что на период отладки, для удобства, я увеличил интервал отправки данных до 5 секунд. Но ничего не мешает вновь установить значение % 4 или даже % 3;

Еще одно важное замечание по поводу работы GPS NEO M6 и синхронизации времени gps. Синхронизация наступает только в том случае если есть приём данных приёмника GPS со спутников.

1. У меня эти модули начинают принимать данные (лампочка начинает моргать), только на подоконнике, внутри комнаты. Ито не всегда. Если облачная погода, то приходится к стеклу прислонять.
2. Если приёмник впервые включается в вашей геолокации, то первоначальный расчёт данных, самим GPS приёмником при первом включении в ваших координатах, может занять до 20 минут при плохом приёме спутников!!!

Эти два фактора необходимо учитывать при построении своей системы.

Так же напомню про специфику – полёты под открытым небом. Проблем с приёмом нет:)

## Скетч транспондера

Весь материал по данному проекту доступен на github.com: <https://github.com/R3PB/AIR-TRANSPONDER>

#include <Oregon\_TX\_AIR.h>

#include <SoftwareSerial.h>

#include <TinyGPS.h>

//#define AIRx 0xAAA5

int offset\_t = 3600; // Миллисекунды. Время смещения(таймслот транспондера) указывать в интервале от 0 до 4500 шагом 500.

//byte AIRx = AIR2;

// GPS модуль

//float lat, lon; // создать переменную для объекта широты и долготы

SoftwareSerial gpsSerial(5, 4);//rx,tx

//SoftwareSerial gpsSerial(14, 13);//rx,tx

TinyGPS gps; // создать gps объект

bool newdata = false;

unsigned long start;

long lat, lon;

float falt;

unsigned long fix\_age, time, date, Second;

int year;

byte month, day, hour, minute, second, hundredths;

uint32\_t mysecond, second\_, milli\_s, timer1;

#define PERIOD 1000 // период отсчёта миллисек. при синхронизации от 00 сек

// static byte GPS\_LED = 16;

// END

//Имитатор датчиков температуры и влажности Oregon Scientific

//В данном примере иммитируются пакеты следующих типов датчиков:

//THGN132N (тип 1D20h, протокол v2, 3 канала)

//RTGN318 (тип ХСС3h, протокол v2, 5 каналов)

//THGR810 (тип F824h, протокол v3, 10 каналов)

///////////////////////////////////////////////////

//Вывод передатчика один. Указывается только при создании первого объекта. В данном примере передатчик подключен к D6

//Oregon\_TM transmitter(6), transmitter2, transmitter3, transmitter4;

Oregon\_TM transmitter, transmitter2, transmitter3, transmitter4;

//tr\_pin = 16;

// Массив режимов работы ЗЕЛЁНОГО светодиода(ИНДИКАТОР ПРИЁМА GPS)

byte modes[] = {

0B00000000, //Светодиод выключен

0B11111111, //Горит постоянно

0B00001111, //Мигание по 0.5 сек

0B00000001, //Короткая вспышка раз в секунду

0B00000101, //Две короткие вспышки раз в секунду

0B00010101, //Три короткие вспышки раз в секунду

0B01010101 //Частые короткие вспышки (4 раза в секунду)

};

uint32\_t ms, ms1 = 0, ms2 = 0;

uint8\_t blink\_loop = 0;

uint8\_t blink\_mode = 0;

uint8\_t modes\_count = 0;

//////////////////////////////////////////////

void setup()

{

Serial.begin(115200);

Serial.println("Waiting data of GPS...");

gpsSerial.begin(9600); // подключаем gps датчик

//Первый передатчик

transmitter.setType(THGN132);

transmitter.setChannel(3); // Номер канала для THGN132 - 1...3

transmitter.setBatteryFlag(1); // Флаг разряженной батарейки

transmitter.setTemperature(24.2); // -50C...+70C

transmitter.setHumidity(13); // 2...98%

transmitter.setComfort(24.2, 13); // Расчёт передаваемого индекса комфорта

//Второй передатчик

transmitter2.setType(THGR810);

transmitter2.setChannel(1); // Номер канала для THGR810 - 1...10

transmitter2.setBatteryFlag(1);

transmitter2.setTemperature(+49.9);

transmitter2.setHumidity(98);

transmitter2.setComfort(+49.9, 98);

//Третий передатчик

transmitter3.setType(RTGN318);

transmitter3.setChannel(5); // Номер канала для RTGN318 - 1...5.

transmitter3.setBatteryFlag(0);

transmitter3.setTemperature(-31);

transmitter3.setHumidity(50);

transmitter3.setComfort(-31,50);

/// ИНДИКАЦИЯ GPS ПРИЁМА - ЗЕЛЁНЫЙ СВЕТОДИОД /////

pinMode(GPS\_LED, OUTPUT);

digitalWrite(GPS\_LED, HIGH);

modes\_count = 1;

blink\_mode = modes[modes\_count];

}

///////////////////////////////////////////////

void loop()

{

newdata = readgps();

// GPS mr.Tech

if (millis() - start > 900) //установка задержки в одну секунду между обновлением данных

{

if (newdata)

{

start = millis();

gps.get\_position(&lat, &lon);

gps.get\_datetime(&date, &time, &fix\_age);

// second = second % 5; // %10 - эта процедура превращает секунды 00 - 59,

// полученные от tinyGPS в десятичном виде, в секунды от 0 до 9 (%5 => 0 ... 4),

// путем деления с остатком который и заносится в переменную.

// Важное замечание! переменную second необходимо представить типом byte!

// Это самый короткий путь чтобы отбросить старший разряд,

// то есть отбросить десятки секунд и оставить только цикл от 0 до 9

falt = gps.f\_altitude(); // +/- altitude in meters

green\_led();

Serial.print("Lat: "); Serial.print(lat);

Serial.print(" Long: "); Serial.print(lon);

Serial.print(" Altitude: "); Serial.print(falt);

Serial.print(" Date: "); Serial.print(date);

Serial.print(" Time: "); Serial.println(time);

Serial.print(" Real\_Time секунды: ");Serial.println(second);

Serial.print(" Offset\_Time: ");Serial.println(offset\_t);

//Serial.print(" Real\_Time минуты: "); Serial.println(minute);

Serial.print(" MILLI\_S: "); Serial.println(milli\_s);

Serial.print(" SECOND\_: "); Serial.println(second\_);

}

}

/////////////////////

//передача осуществляется по таймеру, который определяется типом датчика и номером канала

transmitter4.setType(AIRx);

transmitter4.setChannel(1); // timeslot. формируется номер транспондера

// (Сейчас этот параметр формирует номер канала - timeslot.

// 27.07.20 - флаг, пока, не принимает участия в формировании номера тайм-слота)

transmitter4.setLatitude(lat);

transmitter4.setLongitude(lon);

transmitter4.setAltitude(falt);

// if (transmitter.transmit()) PrintSentData(transmitter.SendBuffer);

// if (transmitter2.transmit())PrintSentData(transmitter2.SendBuffer);

// if (transmitter3.transmit()) PrintSentData(transmitter3.SendBuffer);

if (transmitter4.transmit(second\_, offset\_t)) PrintSentData(transmitter4.SendBuffer);

// параметры (second, offset\_t) - текущее время(только секунды), и время смещения(таймслот)

// Serial.print(" MILLI\_S: "); Serial.println(milli\_s);

// green\_led();

}

//////////////////// ИНДИКАЦИЯ GPS ЗЕЛЁНЫЙ СВЕТОДИОД ////////////////////

void green\_led(){

// blink\_mode = modes[3];

ms = millis();

// Событие срабатывающее каждые 125 мс

// if( ( ms - ms1 ) > 125|| ms < ms1 ){

ms1 = ms;

// Режим светодиода ищем по битовой маске

if( blink\_mode & 1<<(blink\_loop&0x07) ) digitalWrite(GPS\_LED, LOW);

else digitalWrite(GPS\_LED, HIGH);

blink\_loop++;

// }

blink\_mode = modes[2];

// Этот код служит для демонстрации переключения режимов

// Один раз в 5 секунд меняем эффект

// if( ( ms - ms2 ) > 1000|| ms < ms2 ){

// ms2 = ms;

// blink\_mode = modes[3];

// blink\_mode = modes[modes\_count++];

// if( modes\_count >= 7 )modes\_count = 1;

// }

}

// mr.Tech GPS void #########################

// проверка наличия данных

bool readgps()

{

while (gpsSerial.available())

{

int b = gpsSerial.read();

// в библиотеке TinyGPS имеется ошибка: не обрабатываются данные с \r и \n

if('\r' != b)

{

if (gps.encode(b))

return true;

}

}

return false;

}

// END

// синхроизация с нулевой секундой

void sync\_with\_time()

{

gps.crack\_datetime(&year, &month, &day, &hour, &minute, &second, &hundredths);

second = second % 5 ;

if (second){

milli\_s = millis() - timer1;

second\_ = second \* 1000 + milli\_s;

if((millis() - timer1) >= PERIOD ){

timer1 = millis();

}

}

}

////////////////////////////////////////////////////////////////

void PrintSentData(byte\* buf)

{

Serial.print(millis() / 1000);

Serial.print("s \t\t");

for (byte i = 0; i < 10; ++i)

{

byte trmbuf = \*buf;

Serial.print(trmbuf >> 4, HEX);

if (i < (10 - 1) )

Serial.print(trmbuf & 0x0F, HEX);

buf++;

}

Serial.println();

}