**iliasam** вчера в 09:23

# SDR приемник GPS на микроконтроллере

Средний 28 мин 9.5K

Глобальные системы позиционирования*, Программирование микроконтроллеров*, DIY или Сделай сам, Электроника для начинающих

Кейс

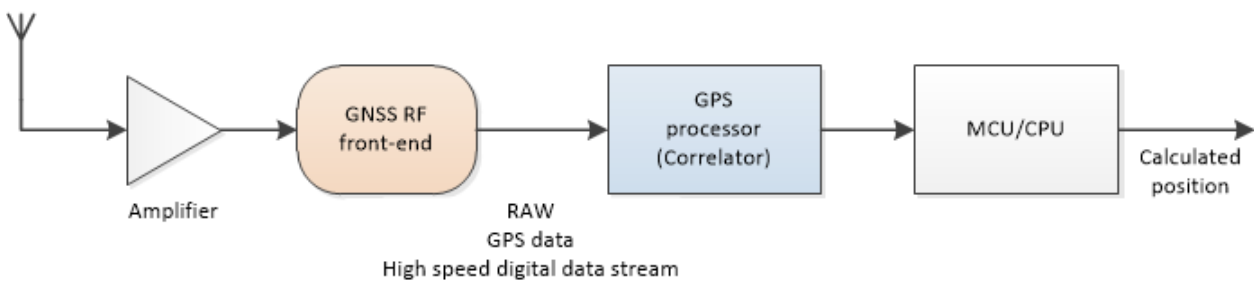
В этой статье я расскажу о том, как я делал самодельный SDR GPS приемник на микроконтроллере. SDR в данном случае означает, что приемник не содержит готовых GPS-модулей или специализированных микросхем для обработки GPS сигналов - вся обработка "сырых" данных выполняется в реальном времени на микроконтроллере (STM32 или ESP32). Зачем я это сделал — просто Just for fun, плюс - получение опыта.

## Введение

Эта статья во многом является продолжением моей предыдущей статьи: Новая жизнь старого GPS-приёмника. В этой статье я кратко описал принципы работы GPS приемников, и рассказал, как я смог получить "сырые" GPS-данные и обработать их на ПК. Принципы, описанные там, используются во всех типах GPS-приемников, основная разница - в способах обработки сигнала. Обработка данных на ПК - очень экзотический случай, обычно потребителю нужно, чтобы приемник был максимально компактным и малопотребляющим.

С учетом того, что обработка GPS-данных требует выполнения достаточно большого числа математических операций в реальном времени и для нескольких каналов сразу - то использование заказных микросхем и ASIC выглядит ожидаемым решением для этой задачи. Именно так и обстоит дела в реальности - в подавляющем большинстве современных GPS-приемников используются именно специализированные микросхемы. Причем такой принцип используется уже довольно давно.

Какой-нибудь старый GPS приемник (вроде Sony Pyxis IPS-360) мог иметь такую структурную схему:




Структурная схема GPS приемника

Отдельные блоки в то время были отдельными микросхемами.

Первым делом в такой конструкции идет усилитель и фильтры.
Затем GNSS RFfront-end - он переносит радиосигнал на более низкую частоту и оцифровывает его (АЦП тут имеет разрядность 1-2 бита).
Далее как раз - вышеупомянутая специализированная микросхема для обработки данных GPS, иногда называется коррелятор, так как ее основная задача - поиск максимума корреляции для каждого принимаемого спутника. Пример такой микросхемы - GP2021.
Далее - классический микроконтроллер или микропроцессор, который обрабатывает относительно медленно меняющиеся данные с коррелятора и вычисляет положение приемника.
С развитием технологий все эти блоки удалось совместить в одной микросхеме. Такие микросхемы, в том числе, используются в модулях GPS приемников, к которым любители DIY уже привычны.

А что делать, если не хочется брать готовую микросхему, а хочется разобраться с обработкой сырых данных? Проще всего использовать связку RF front-end + ПЛИС, в которой реализовать всю ресурсоемкую обработку данных с АЦП, координаты пользователя можно считать на той же ПЛИС (используя софтовый процессор) или используя внешний процессор. Такой подход можно увидеть в этом DIY проекте [1].

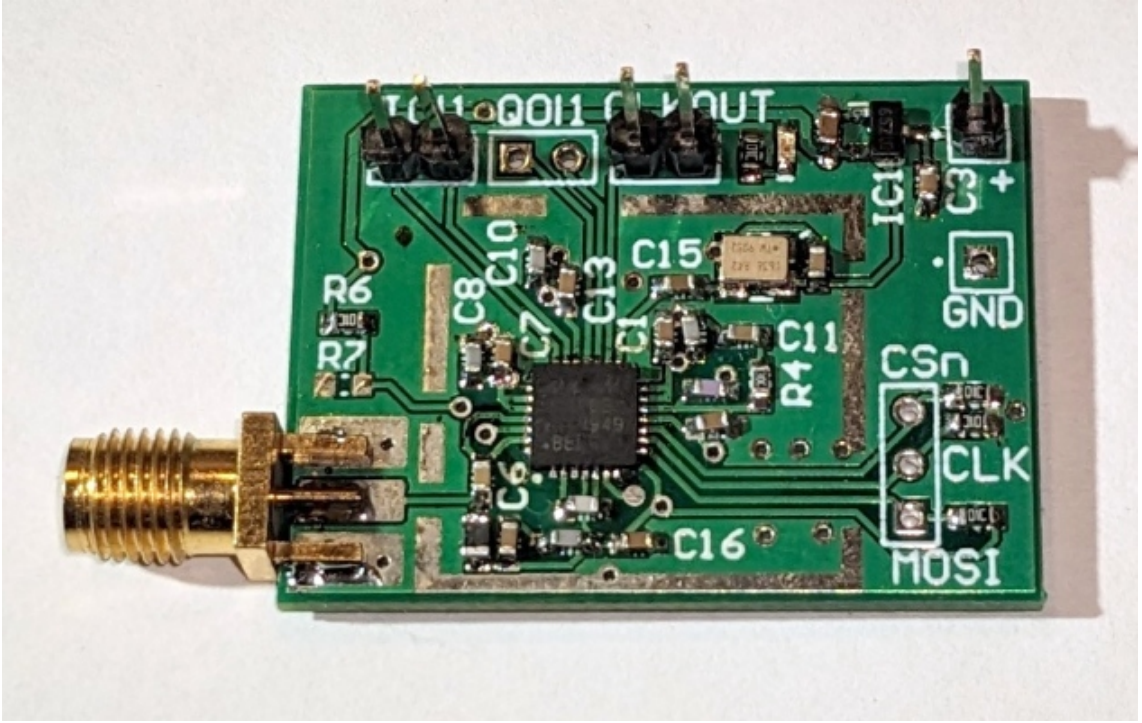
А можно ли обойтись без ПЛИС и ПК, и обрабатывать сырые данные в реальном времени прямо на микроконтроллере, вроде STM32? Честно сказать, я не смог найти подобных проектов. Для DSP такие проекты попадались, но там авторы использовали мощные процессоры с серьезным объемом ОЗУ.

Есть вот такой экзотический и сложный проект микропотребляющего GPS приемника - (сайт производителя). Разработчики, используя некоторые трюки, добились возможности определения координат, включая приемник на единицы миллисекунд с периодом в сотни секунд. При этом они не используют такие традиционные для GPS-приемников вещи, как tracking (слежение за сигналом от спутника). Нет даже приема навигационных данных! Вся обработка данных производится на микроконтроллере MAX32632 (Cortex-M4).
Но есть и недостатки - насколько я понял из описания, устройство нужно предварительно инициализировать (указать текущее время и примерные координаты), и загружать туда данные эфемерид (их предоставляет производитель, и их хватает на 28 дней). И, конечно же, все полностью закрытое.

Так что выяснять, насколько эта задача вообще реализуема, мне пришлось самостоятельно.

## Получение "сырых" GPS данных

Об этом я довольно много рассказал в своей предыдущей статье. В этот раз я решил использовать уже имевшуюся у меня отладочную плату GNSS RF front-end, собранную на микросхеме MAX2769.




MAX2769 GNSS RF front-end

Эта микросхема удобна тем, что имеет специальный режим - *Preconfigured Device State,* в который ее можно переключить, подав единицу на вход PGM. Таким образом можно получить данные с микросхемы, не конфигурируя ее по SPI. Я использовал вариант конфигурации - 2.

**Table 3. Preconfigured Device States**

| DEVICE STATE | DEVICE ELECTRICAL CHARACTERISTICS | | | | | | | | 3-WIRE CONTROL PINS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | REFERENCE FREQUENCY (MHz) | REFERENCE DIVISION RATIO | MAIN DIVISION RATIO | I AND Q OR I ONLY | NUMBER OF I Q BITS | I AND Q LOGIC LEVEL | IF CENTER FREQUENCY (MHz) | IF FILTER ORDER | SCLK | DATA | $\overline{CS}$ |
| 0 | 16.368 | 16 | 1536 | I | 1 | Differential | 4.092 | 5th | 0 | 0 | 0 |
| 1 | 16.368 | 16 | 1536 | I | 1 | Differential | 4.092 | 3rd | 0 | 0 | 1 |
| 2 | 16.368 | 16 | 1536 | I | 2 | CMOS | 4.092 | 5th | 0 | 1 | 0 |
| 3 | 32.736 | 32 | 1536 | I | 2 | CMOS | 4.092 | 5th | 0 | 1 | 1 |
| 4 | 19.2 | 96 | 7857 | I | 2 | CMOS | 4.092 | 5th | 1 | 0 | 0 |
| 5 | 18.414 | 18 | 1539 | I | 2 | CMOS | 1.023* | 5th | 1 | 0 | 1 |
| 6 | 13 | 65 | 7857 | I | 2 | CMOS | 4.092 | 5th | 1 | 1 | 0 |
| 7 | 16.368 | 16 | 1536 | I | 1 | CMOS | 4.092 | 5th | 1 | 1 | 1 |

**If the IF center frequency is programmed to 1.023MHz, the filter passband extends from 0.1MHz to 2.6MHz.*

▸ Схема RF front-end

Фактически, вся плата состоит из MAX2769, TCXO - кварцевого генератора на 16.368 МГц и линейного источника напряжения 3.0В.
В разъему J3 подключаемся активная антенна, с разъема P5 снимается сигнал тактирования - тоже частотой 16.368 МГц. На разъем P3 плата выдает оцифрованные данные - "сырые" данные GPS.

## Захват данных на STM32

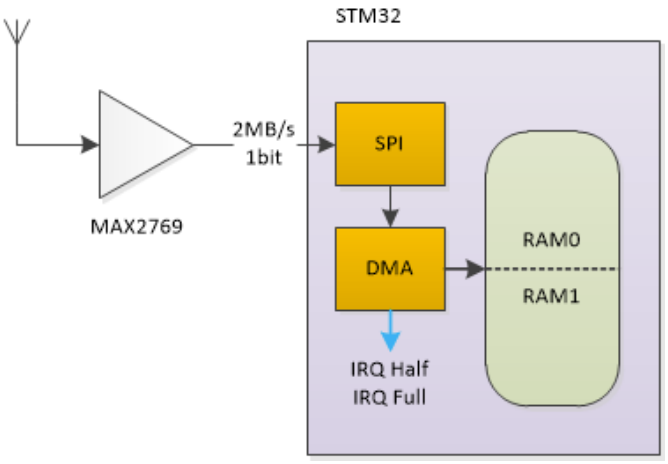
В итоге мы имеем на выходе микросхемы MAX2769 двухбитный поток данных. Однако для

работы приемника достаточно и однобитных данных, так что для упрощения конструкции, я использовал этот режим. Данные берутся с вывода I0 MAX2769.

В своих экспериментах я решил использовать старую отладочную плату STM32F4-Discovery. Чего только я не ней не делал с 2012 года, а родной микроконтроллер все еще жив)
Характеристики MCU:

- Ядро Cortex-M4, 32-бит
- Тактовая частота 168МГц
- Flash - 1 Мбайт
- ОЗУ - 192 Кбайт

Получается, что MCU должен захватывать последовательный синхронный поток данных со скоростью ~16Mbit/s (2 Mbyte/s). Немало.
Лучше всего для этой подходит интерфейс SPI, работающий в связке с DMA. DMA настроен в режим Circular, т.е. дойдя до конца выделенной памяти, DMA автоматически начнет запись в ее начало. В итоге мы получаем в памяти набор данных, в которых один бит соответствует одной выборке АЦП MAX2769.




Для хранения принятых данных в ОЗУ выделен один буфер, но DMA может формировать прерывания, когда счетчик принятых данных доходит до середины и конца памяти. Таким образом можно обеспечить двойную буферизацию - она часть памяти заполняется, вторая обрабатывается.

Обращаю внимание, что в процессе приема, недопустимо останавливать прием или терять даже одиночные байты - это приведет к потере синхронизации. В начале приема приходится выполнять достаточно длительные операции, так что перед выполнением таких операций новые данные нужно максимально быстро сохранить в дополнительный буфер.

**Частоты, данные и время**

Частота тактового сигнала 16.368 МГц выбрана разработчиками MAX2769 совершенно неслучайно. Напомню, что в системе GPS используется повторяющийся *дальномерный код (PRN)*, который спутники передают с периодом 1 мс. За время 1мс микросхема MAX2769 выдаст 16368 выборок АЦП.
В то же время, один период PRN состоит из 1023 *чипов* (единичных участков, где фаза модуляции сигнала не изменяется).
16368 / 1023 = 16, то есть одному чипу будут соответствовать ровно 16 выборок АЦП. В моем

случае, это ровно 16 бит = 2 байта. Получается, один байт = 0.5 *чипа*, что в дальнейшем сильно пригодится.

Получается, что для хранения 1мс данных (1 PRN) нужно выделить 16368/8= 2046 байт (~2 Кбайт). Для двойной буферизации нужно ~4 Кбайт, плюс нужен еще один буфер ~2 Кбайт для длительной обработки данных.

В выбранном режиме MAX2769 выдает данные на промежуточной частоте (IF), равной 4.092 МГц. Это ровно 1/4 от тактовой частоты. Получается, что один *чип* при полном отсутствии помех будет выглядеть как 0b1100110011001100 (это один из 4 существующих вариантов сдвига фазы).

## Обработка данных

В предыдущей статье я уже описывал методы, используемые для обработки данных на ПК. Очевидно, что MCU слишком слаб, чтобы переносить алгоритмы с ПК напрямую - не хватает производительности ядра, объемов ОЗУ и даже специальных инструкцией, которые есть на ПК.

Что обычно делают на ПК, получив поток данных с GNSS RF front-end? Переводят каждую выборку АЦП в форму целых чисел со знаком, или даже в формат с плавающей запятой. Такой подход означает, что каждую секунду процессор должен обрабатывать 16368000 выборок. Это много, так что я даже не стал проверять, насколько это вообще реалистично, даже с учетом того, что STM32F4 тоже считается DSP и имеет FPU.

Вместо этого я использовал другой подход, увиденный в этом проекте: A homemade receiver for GPS & GLONASS satellites [2]. Там автор смог сделать GPS приемник на обычных цифровых микросхемах. Конечно, использование там методы невозможно напрямую перенести на MCU, но кое-что взять можно. Главное - часть операций с цифровыми данными там идет на уровне одиночных битов. Ключевая операция ЦОС - умножение, заменяется логической операцией "Исключающее ИЛИ" (XOR).

| A | B | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**XOR**

| A | B | Q |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

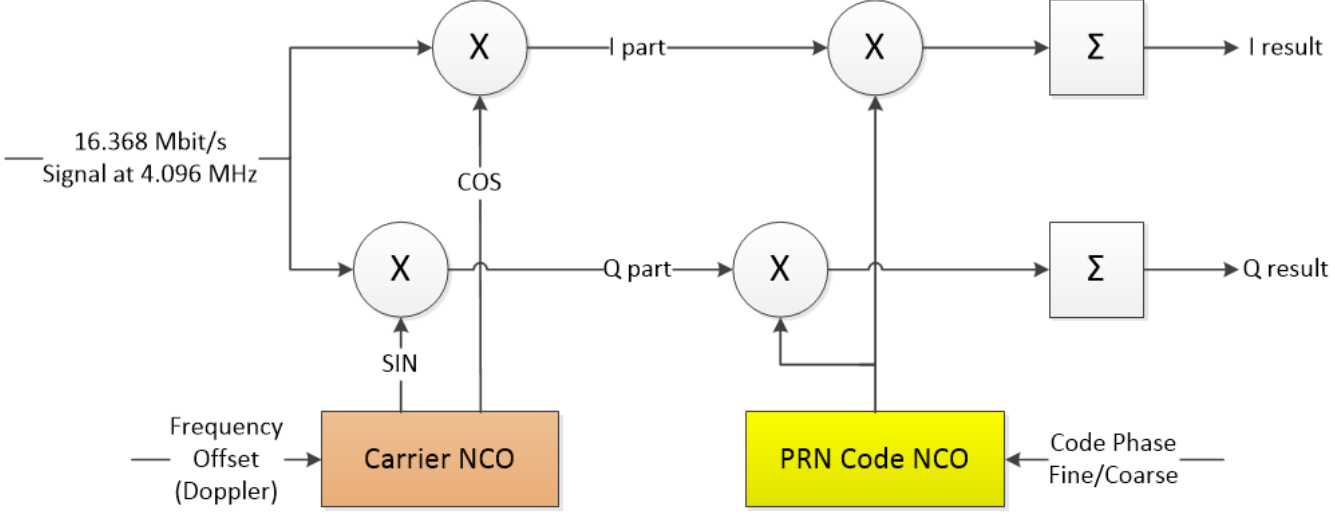
**XNOR**

Взято отсюда: https://homepages.inf.ed.ac.uk/rbf/HIPR2/xor.htm

Если посмотреть на таблицу истинности инвертированной операции XOR (XNOR), и представить, что значениям "0" соответствует негативный знаковый сигнал "-1", то получается, что XNOR действительно может заменить умножение (без переносов). Два последовательно идущих XOR своей инверсией компенсируют друг друга, так что в таком случае XNOR не требуется.

Операция XOR хороша тем, что ее можно выполнять сразу для большого числа выборок-битов, производя операции над 16/32 битными словами, и выполняется она очень быстро.

Какие же операции нужно производить в GPS приемнике? Вот так выглядит классическая структурная схема цепочки операций:

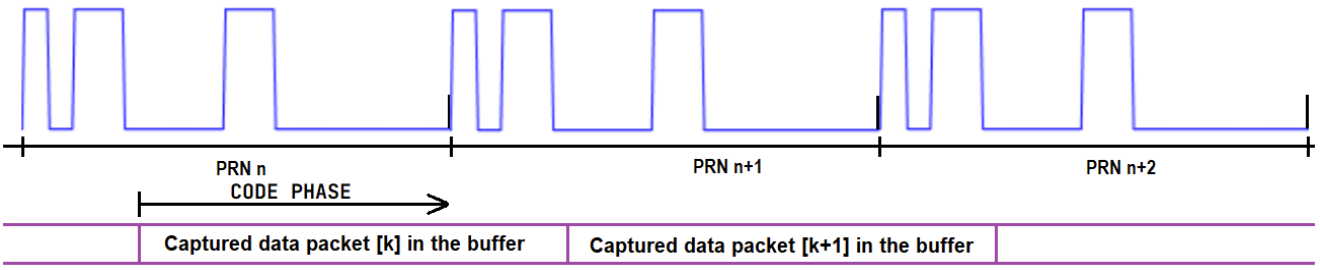



Левая половина - классический перенос сигнала с одной частоты на другую. Напомню, что сигнал, принятый со спутника, имеет доплеровское смещение частоты, у каждого спутника оно будет свое. Кроме того, тактовый генератор приемника тоже может иметь свою ошибку частоты, а MAX2769 переносит сигнал не на нулевую, а на промежуточную частоту.
Перемножив исходный сигнал на сформированный нами гармонический сигнал нужной частоты (Carrier NCO), мы получим на выходе умножителя новый сигнал - входной сигнал окажется перенесен на более низкую частоту. Обычно в GPS производят перенос сразу на нулевую частоту - т.е. после умножителя мы получаем сигнал, передаваемый передатчиком.
Вот только и сигналы, находящиеся по частоте выше/ниже нашего генератора, тоже оказываются перенесенными вниз, и отделить их друг от друга нельзя.
Чтобы решить эту проблему, используют два смесителя (умножители) и формируют два гармонических сигнала, смещенные на 90 градусов. Фактически, в таком случае происходит не только перенос сигнала на другую частоту, но и перевод его в комплексную форму, что дает дополнительную информацию о фазе сигнала. Получается квадратурный сигнал, его составляющие обозначаются буквами I и Q.

Следующий этап - формирование локального PRN кода. У каждого спутника он свой, его можно сформировать при инициализации приемника. Сам код состоит из 1023 чипов/бит.
Однако напрямую сформированные данные использовать нельзя, перед перемножением нужно преобразовать скорость данных, чтобы она соответствовала той скорости, с которой идут данные от SPI.

Следующий этап - окончательное перемножение данных. Получается, что с одной стороны мы имеем данные за 1мс, полученные со спутника, и перенесенные на нулевую частоту, а с другой стороны - локальные "идеальные" данные PRN (local replica code). Перемножив данные поэлементно, а потом их просуммировав, мы получаем *меру подобия* сигналов, или результат взаимной корреляции. Как видно из схемы, для каждого из каналов I/Q нужно выполнять свои вычисления.

Тут стоит ввести часто упоминаемое понятие - *фаза PRN кода*. Фактически, это просто величина задержки, на которую нужно сдвинуть локальные данные PRN, чтобы получить максимум корреляции. Спутники постоянно двигаются, и получается, что фаза кода будет тоже постоянно меняться.




Фаза кода - Code Phase

Если помехи отсутствуют, частота и фаза генератора промежуточной частоты Carrier NCO полностью совпадают с частотой и фазой принимаемого сигнала, и фаза кода принятых данных равна фазе локально сформированного PRN, то получается, что во время передачи спутником установленного в "1" бита навигационных данных, мы будем иметь на выходе коррелятора максимум, при передаче "0" - минимум.
Напомню, что один бит данных передается в течении 20мс, в за это время передается 20 кодов PRN подряд, во время передачи "0" передаваемые *чипы* инвертируются.

**Особенности реализации алгоритмов на практике**

Как я уже писал выше, в моей реализации все приходящие данные обрабатываются блоками по 1мс длиной, при этом для слежения (tracking) данные должны обрабатываться непрерывно. Это означает, что все вышеупомянутые вычисления нужно успеть сделать за 1мс.

Формирование несущей частоты (Carrier NCO)
Очевидно, что использовать тригонометрические функции, чтобы вычислить значения одиночных битов, в данном случае ужасно расточительно.
Можно посмотреть, как формирование частот сделано в DDS генераторах. А там все довольно просто, и может быть реализовано в целочисленном виде. Так как формируются однобитные данные, то даже таблица поиска (LUT) не нужна - достаточно просто проверять старший бит переменной-аккумулятора фазы. Однако, как оказалось, даже такой простой способ требует слишком много времени. Для вычисления 16368*2 значений бит требовалось 1280мкс.

Поэтому я решил использовать несколько другой подход, основанный на том, то формируемая частота очень близка к 1/4 от тактовой частоты. Предельное доплеровское смещение в этом проекте я установил в 7кГц, что означает, что необходимое предельное смещение частоты составляет около 0.2%. Получается, что при этом период формируемого сигнала составляет 4 выборки (бита), а отклонение частоты создает только фазовой сдвиг формируемого сигнала.
В итоге, чтобы получить буфер, заполненный данными нужной частоты, достаточно разбить его на 32-битные слова, и записывать в них значения из таблицы:
*const uint32_t sin_buf32[4] = { 0x33333333, 0x9999999, 0xCCCCCCCC, 0x66666666 };*
Индекс в таблице - как раз значение фазы, его нужно периодически менять, а методика расчета фазы та же, что и в DDS генераторе.

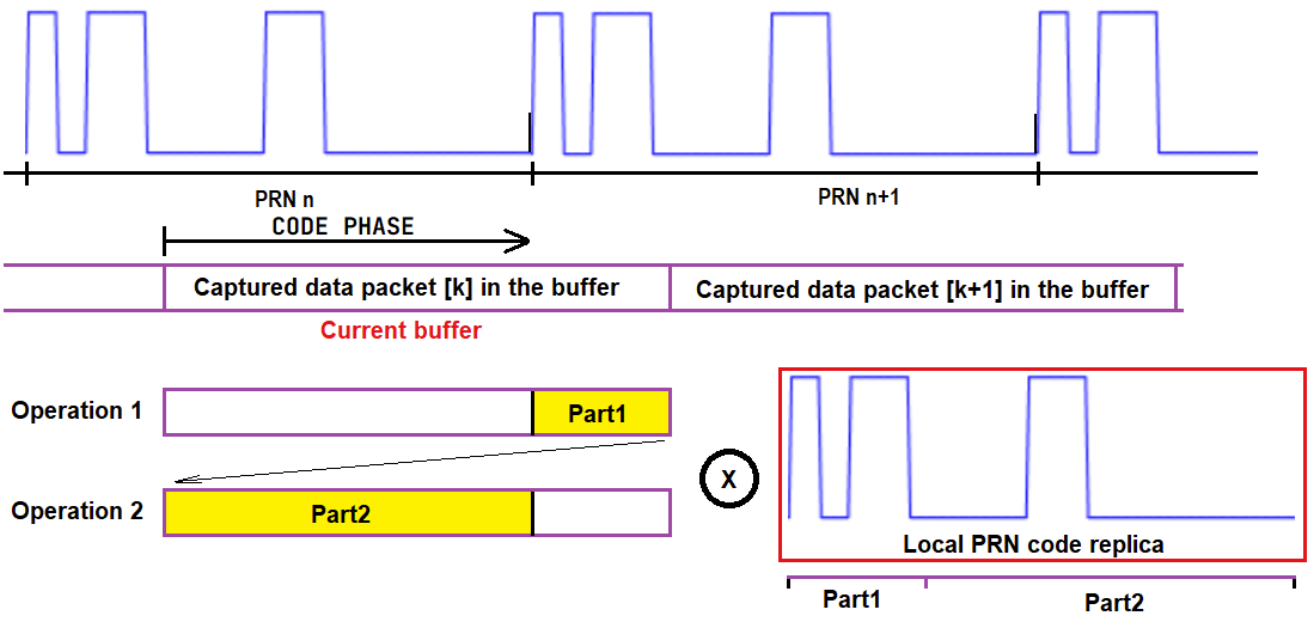
Если посмотреть на структурную схему алгоритма приема данных выше, то можно увидеть, что данные генератора Carrier NCO далее больше нигде не используются. Получается, что нет смысла заполнять специальный буфер данными - можно "на лету" перемножать через XOR данные от SPI на элементы *sin_buf32,* и уже этот результат сохранять в промежуточный буфер. С второй частью квадратурных данных все аналогично.
Получившаяся у меня функция для I/Q данных выполняется за 44 мкс.

Формирование данных PRN (локальная реплика)
Тут все заметно проще. Данные PRN уже сгенерированы во время инициализации, нужно только

привести их к скорости SPI. Один *чип* PRN длится 16 выборок (бит), так что достаточно разбить буфер на 16-битные слова, и заполнить его значениями 0x0/0xFFFF, используя исходные данные PRN. Еще на этом этапе можно сместить записываемые данные с шагом 1 бит = 1/16 *чипа*, но в таком случае нужно будет использовать уже 32-битные слова.
Получившаяся функция выполняется за 70 мкс.

Перемножение и суммирование

Вот в этот момент появляется важный нюанс - что делать с фазой кода?
Коррелятор выдает значительный результат только в том случае, если фаза кода локальной реплики PRN строго совпадает с фазой кода принятых данных. Получается, что нужно иметь возможность управлять сдвигом данных. Я решил выполнять сдвиг именно в момент умножения, сдвигая при этом просто указатель на обработанные данные. Указатель можно сдвинуть минимум на байт, то есть точность установки фазы кода - 0.5 *чипа*.
Вот только буфер данных - всего на 1мс, а при сдвиге точно возникает выход за границы массивов с данными, как это решать? Более правильный вариант - изменять подход к обработке данных, но я решил пойти по более простому пути. Как я уже упоминал, передача одного навигационного бита занимает 20мс, то есть данные PRN повторяются 20 раз подряд. Это позволяет закольцевать обработку данных:




Закольцованная обработка принятых данных

Получается, что в таком случае обрабатываются сначала данные новой PRN последовательности, потом - старой. В случае, если передаваемые в этот момент данные не меняются, то алгоритм работает без проблем. А вот в момент смены знака навигационных данных результат корреляции будет напрямую зависеть от знаков данных и значения фазы кода. Это усложняет определение точного момента времени, когда произошло такое изменение знака навигационных данных и сильно мешает в процессе захвата сигнала (Acquisition).

Остается операция суммирования. Чтобы вычислить результат корреляции, нужно просуммировать все биты, которые получились в результате умножения. А их много - 16368.
К сожалению, используемый MCU не имеет команд ядра, позволяющих посчитать сумму установленных бит (в x86 это popcount). Нужно реализовывать суммирование полностью программно. Я рассмотрел разные методы, и остановился на простой таблице поиска на 16 бит. Она хранится в ОЗУ (так быстрее) и занимает 64 Кбайт. Заполняется таблица во время инициализации. Так как суммирование 16-битное, то и операции умножения XOR сделаны 16-битными. Переход к 32-битным вычислениям не дает значительного прироста по скорости, так

как именно операции суммирования требуют больше времени.
В итоге получившаяся функция умножения и сложения для I/Q данных выполняется за 112 мкс.

## Управление приемом

Для того, чтобы вышеописанный алгоритм приема работал, нужно знать следующие параметры сигнала:

- Номер спутника, чтобы сгенерировать PRN данные
- Доплеровское смещение частоты спутника, чтобы правильно перенести принятый сигнала на нулевую частоту
- Фаза кода
- Фаза несущей (фаза Carrier NCO)

Номера спутников в своем проекте я указываю вручную, в готовых устройствах каналов корреляции настолько много, что в случае "холодного старта" можно без проблем проверить наличие сигнала от всех спутников за небольшое время.

Чтобы определить текущие частоту и фазу кода сигнала, в приемниках обычно имеется специальный режим - захват сигнала (Acquisition). Так как у MCU слишком мало ресурсов, именно этот режим занимает больше всего времени. У себя я разбил его на несколько этапов - поиск частоты и несколько итераций поиска фазы кода. Acquisition в моем случае работает не с данными реального времени, а со скопированными в дополнительный буфер.

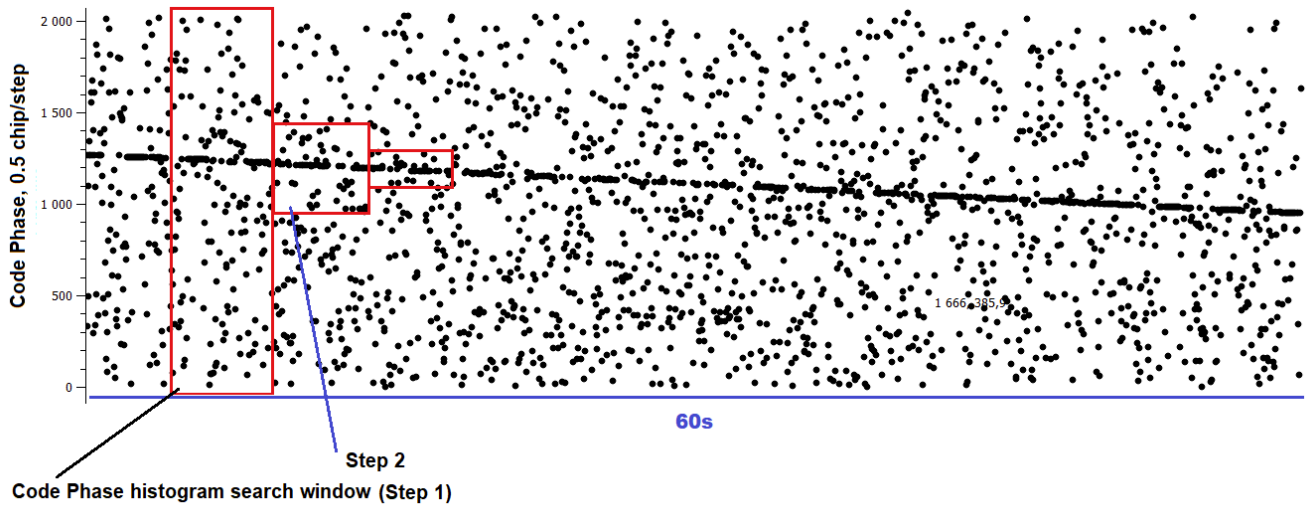
**Поиск частоты**
Самая длительная часть Acquisition. В предыдущей статье я упоминал, что есть специальный метод поиска частоты и фазы, основанный на FFT. Однако для MCU FFT на 16368 точек - довольно ресурсоёмкая вещь, каких-то специализированных однобитных FFT я не нашел. Так что в этом проекте я реализовал поиск напрямую, через перебор всех параметров.

Основная операция - поиск максимума корреляции. Нужно перебрать все 2046 вариантов фазы кода, и найти максимум (в идеале - перебрать все 16368 семплов, но это слишком долго). Такой поиск занимает около 0.23с. Однако, входной сигнал довольно шумный, плюс с моим кольцевым коррелятором есть риск попасть на момент смены навигационных битов - такие данные анализировать бесполезно, но обнаружить такое событие во время acquisition проблематично. Поэтому я запускаю поиск 10 раз подряд (каждый раз с последними актуальными данными), данные о найденных фазах кода сохраняю в массив. Проверив, если ли в этом массиве часто встречающееся значение, можно судить о том, есть ли на данной частоте сигнал от спутника. У этого подхода есть проблема - если спутник быстро движется относительно наблюдателя, то фаза кода быстро меняется, и такие данные сложно анализировать.

Перебирая частоты по завершению вышеописанного процесса, и анализируя получающуюся гистограмму, можно опередить частоту, на которой идет передача. В своем проекте я произвожу сканирование диапазона частот [-7000..+7000 Гц] с шагом 500 Гц. Получается 29 шагов на весь диапазон, для его полного сканирования нужно около 60 секунд. Если сигнал от спутника слабый, то нужны повторные сканирования. Замечу, что все это касается одного спутника, так что каждый используемый спутник также требует такого долгого поиска. В итоге поиск частот 4 спутников может занять 4-10 минут.

**Поиск фазы кода**

На этом этапе приемник должен определить текущую фазу кода. Принцип поиска здесь похож на тот, что был использован ранее, но найденная фаза кода сразу же добавляется в гистограмму, которая тут же анализируется.
Если в гистограмме обнаруживается крупный пик, то поиск можно считать оконченным. Проблема в том, что перебор всех фаз занимает много времени, а фаза успевает "убежать" за то время, когда происходит поиск фазы других спутников. Вот так это выглядит для не очень сильного сигнала (наклонная линия - правильно найденное значение фазы кода, остальные значения - шум):



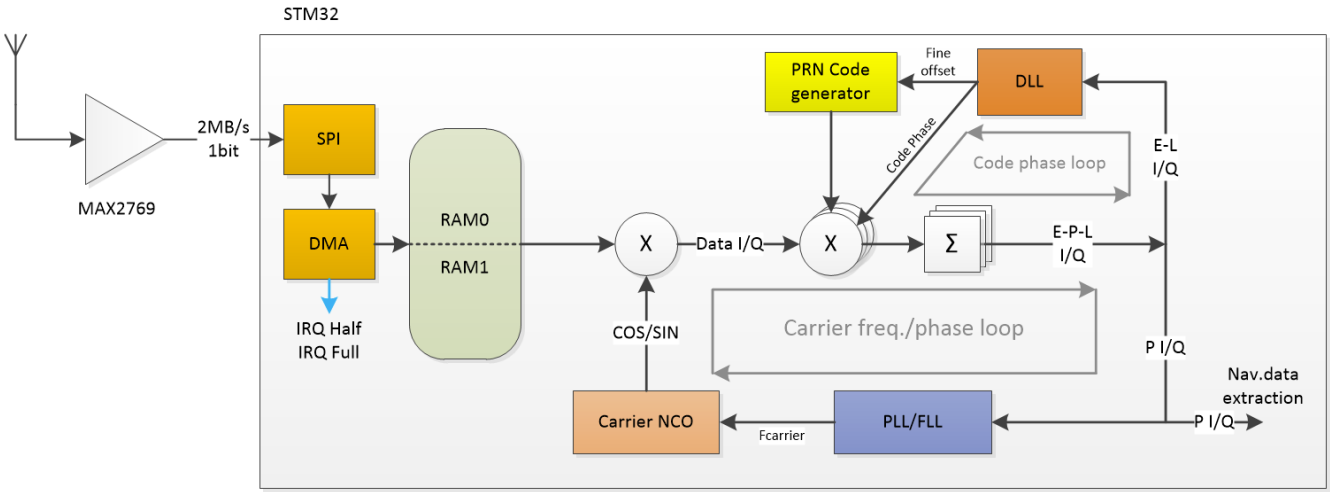

Изменение фазы кода, обработка "сырых" данных на ПК.
Также показаны этапы обработки данных.

Вполне возможно, что можно было бы предсказывать скорость ухода фазы кода, используя данные о смещении частоты (между этими параметрами есть связь), но я прошел более простым путем - использовал итеративный подход. То есть на первом этапе сканируются все фазы (2046 шагов), на следующем - +-250 шагов (отсчитанные от найденной фазы), далее - 60 шагов. По мере уменьшения числа шагов скорость поиска пропорционально возрастает.
Важно иметь таймаут на первых этапах - если за определенное время в гистограмме не обнаружился пик, ее нужно сбросить, иначе сдвигающаяся фаза "размажется" по гистограмме.
В итоге удается найти фазу кода для всех спутников, но она все еще не найдена с нужной точностью.

**Слежение за сигналом (Tracking)**
Эта часть алгоритма уже выполняется строго в реальном времени. Так так в моем случае фаза кода изначально найдена недостаточно точно, я добавил сюда еще один этап - pre-tracking. Он тоже должен выполняться в реальном времени, в отличие от процессов Acquisition.
Алгоритм pre-tracking тоже основан на поиске максимума корреляции, только число операций поиска ограничено - чтобы уложится в 1мс. Принцип анализа данных здесь такой же, как и в Acquisition - новые значения фазы кода добавляются в массив, который в процессе анализируется на наличие одинаковых значений. Несколько одинаковых значений с высокой степенью вероятности являются именно правильным значением фазы кода. Таким образом получается значение фазы кода с точностью до 0.5 *чипа*, полученное в реальном времени, после чего можно переходить к настоящим операциям слежения за сигналом.
Во многом этот процесс я также описал в предыдущей статье.

На момент запуска трекинга частота сигнала известна грубо, фаза несущей частоты неизвестна, а принимаемая фаза кода может "убежать" в любой момент. Именно поэтому за всеми этими

параметрами приемник должен "следить".




Структурная схема механизма слежения

На приведенной схеме присутствуют ранее упомянутые элементы обработки сигнала - Carrier NCO, генератор PRN кода, умножители и сумматоры, только теперь они управляются данными, полученными в результате вычисления корреляции.
Можно видеть, что для управления параметрами здесь используются два контура обратной связи - DLL (Delay Locked Loop) и PLL/FLL (Phase Locked Loop/Frequency Locked Loop).
Обе реализации алгоритма я взял из программы GNSS-SDRLIB.

Внимательный читатель мог заметить, что корреляторов на картинке выше три. Они обрабатывают одни и те же данные, но на них подаются разные величины фазы кода, отличающиеся на 0.5 чипа (-1/2; 0; +1/2). Такой подход называется (*Early*, *Prompt*, *Late*), благодаря нему можно оценивать, насколько текущее значение фазы кода программы отличается от принимаемого, включая знак ошибки.

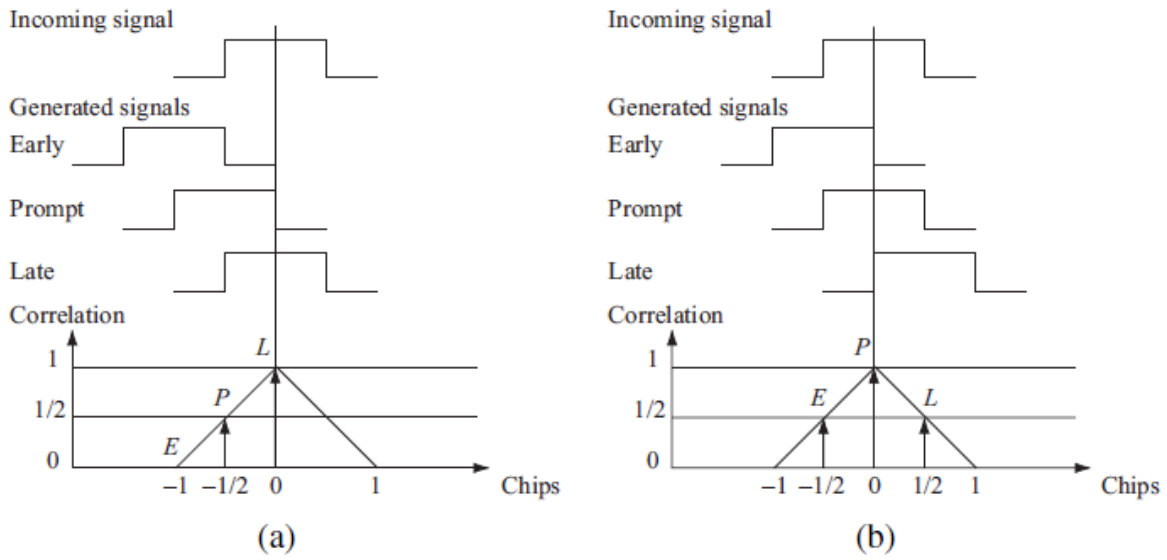



FIGURE 7.11. Code tracking. Three local codes are generated and correlated with the incoming signal. (a) The late replica has the highest correlation so the code phase must be decreased, i.e., the code sequence must be delayed. (b) The prompt code has the highest correlation, and the early and late have similar correlation. The loop is perfectly tuned in.

Изображение из книги [3], хорошо объясняющая происходящее.

Так как в моем проекте 1 байту данных соответствует как раз 0.5 *чипа*, то три варианта сдвигов

реализуются довольно просто.
Однако, у такого метода есть и недостаток - если по каким-то причинам ошибка станет больше 0.5 чипа, то механизм обратной связи может навсегда "потерять" сигнал.

Задача контура DLL - слежение за фазой кода сигнала, используя данные E-P-L корреляторов. Конкретно у меня, так же, как и в GNSS-SDRLIB, используются данные только E-L корреляторов. Ошибка вычисляется по такой формуле:

| | | |
|---|---|---|
| Noncoherent | $\frac{(I_E^2 + Q_E^2) - (I_L^2 + Q_L^2)}{(I_E^2 + Q_E^2) + (I_L^2 + Q_L^2)}$ | Normalized early minus late power. The discriminator has a great property when the chip error is larger than a $\frac{1}{2}$ chip; this will help the DLL to keep track in noisy signals. |

Изображение из книги [3], TABLE 7.2

Далее значение ошибки фильтруется, умножается на дополнительный коэффициент и используется для изменения текущего значения кода фазы.

Корреляторы, основанные на XOR умножении, позволяют изменять фазу кода с точностью до 1 байта=0.5 чипов. А если хочется точнее? Повышение точности настройки фазы кода позволяет улучшить результат корреляции и точней определять расстояние. Вот тут-то и используется возможность генератора данных PRN сдвигать данные на один бит.

**PLL/FLL** - используются для управления частотой генератора несущей частоты (Carrier NCO). Изменяя частоту, можно управлять и фазой сигнала этого генератора. Слежение за фазой сигнала необходимо для извлечения из сигнала навигационных данных. В GPS для слежения за фазой его часто реализуют, используя реализацию Costas Loop. Этот регулятор удобен тем, что не чувствителен к изменению фазы сигнала на 180 градусов, а она как раз появляется при смене знака передаваемых навигационных данных. Задачей этого регулятора является удержание амплитуды канала I максимальной, а Q - минимальной.
Исходными данными являются результаты, выдаваемые коррелятором P (prompt).
Ошибка фазы частоты вычисляется по формуле:

| Discriminator | Description |
|---|---|
| $D = \tan^{-1}\left(\frac{Q^k}{I^k}\right)$ | The discriminator output is the phase error. |

Изображение из книги [3], TABLE 7.1

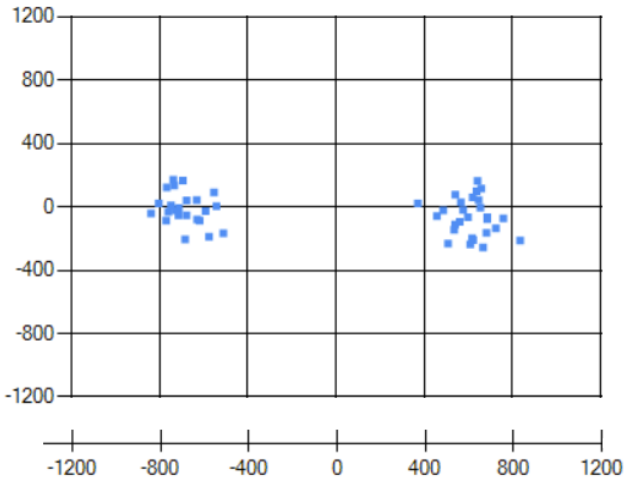
Значение ошибки также фильтруется, умножается на дополнительный коэффициент и используется для изменения текущего значения смещения частоты приема.

Подстройка частоты **FLL** основана на производной ошибки фазы частоты по времени.
Я в своей реализации алгоритма заметил, что в некоторых случаях PLL/FLL "захватывает" ошибочную частоту. Чтобы справится с этой проблемой, я добавил в код проверку принимаемых данных. Если данные некорректные, частота автоматически перестраивается на случайную, а дальше уже FLL автоматически пытается подстроить частоту. Вполне возможно, что у меня где-то просто ошибка в коде, которая приводит к такой ситуации.

Для того, чтобы оценивать состояние работы обратных связей в процессе разработки, хорошо

иметь какой-нибуть инструмент, позволяющий визуализировать состояние трекинга. С моей точки зрения, неплохим решением может являться диаграмма распределения результатов работы коррелятора P (prompt). Коррелятор выдает значения I/Q, значения I откладываются по оси X, Q - по Y. Работу алгоритмов я отлаживал на ПК, так что тут не проблема сделать отображение такой диаграммы.

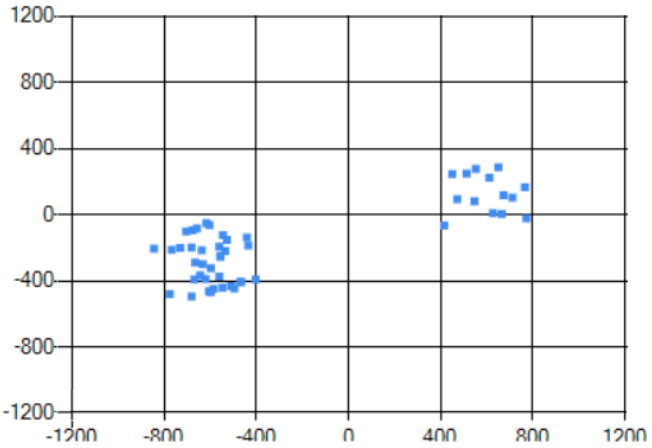
При работающих DLL/PLL после выхода на режим получается такая картина (хорошо видно сигнальное созвездие с модуляцией BPSK):




Работающие DLL/PLL обратные связи

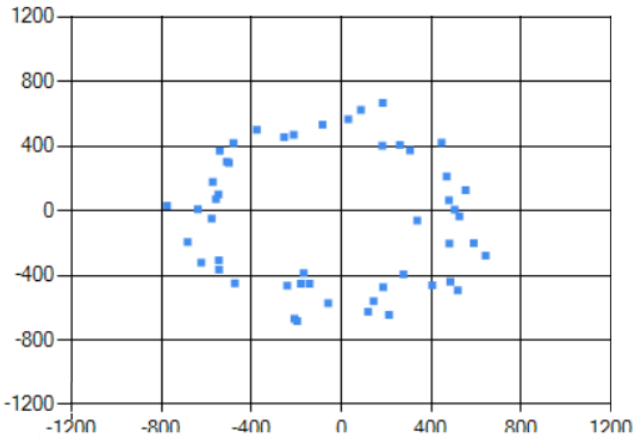
Модуляция тут - та, что связана с передачей навигационных бит.

При наличии ошибки фазы PLL возникает поворот созвездия:




Ошибка по фазе PLL, DLL работает

В случае, если есть сильная ошибка по фазе (PLL не работает или не смог достичь стабилизации фазы), а DLL работает нормально, то изображение выглядит как окружность:




Радиус окружности зависит от уровня сигнала.

Ошибка по фазе генератора велика, DLL работает

Если же DLL работает неправильно, и фаза **кода** ушла слишком сильно - то в центре диаграммы будет просто одиночное пятно, размер которого определяется только шумом.

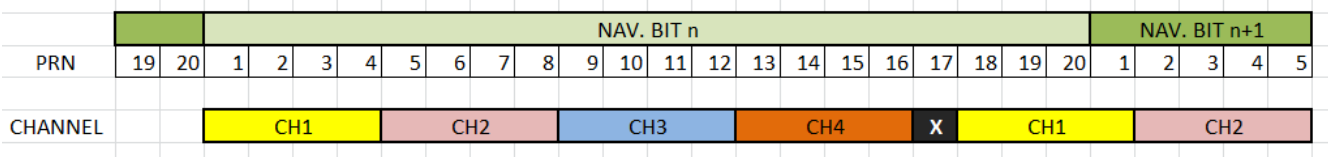
**Мультиплексирование**

Если посчитать суммарное время, необходимое для обработки нового пакета данных от одного спутника, то выйдет: перенос частоты (44 мкс) + формирование PRN (70 мкс) + корреляторы (112 мкс x 3) = 450 мкс. Видно, что на два спутника этого с трудом хватит, но на 4 - никак. Что же делать, если обработку данных прерывать нельзя?
Решение нашлось в статье [2] - там все соответствующие узлы были аппаратные, а приемник был одноканальным. Там автор использовал мультиплексирование каналов - приемник периодически переключался между отдельными спутниками. Важно, чтобы выполнялись следующие условия:

1. Синхронизацию (прием данных по SPI) нарушать никак нельзя.
2. Одиночному спутнику периодически должно доставаться несколько последовательно идущих "окон" в 1мс для анализа навигационных данных. Для определения координат важно знать точные моменты смены знака навигационных данных, а для этого нужно анализировать несколько подряд идущих PRN.
3. Нельзя выделять одному спутнику слишком много времени - все навигационные биты должны быть приняты, а один бит длится 20мс. Кроме того, если переключать каналы слишком редко, то без трекинга параметры принимаемого сигнала могут сильно уйти.
4. Период передачи навигационных бит 20мс **не** должен делится нацело на период мультиплексирования - иначе может сложится такая ситуация, что в определенном канале перестанет попадаться смена знаков.

В проекте [2] у автора были свои сложности - из-за аппаратных особенностей приёмника, переключение каналов само по себе занимало 1 мс, так что переключать каналы слишком часто было нельзя. В результате ему пришлось делать достаточно сложный алгоритм приема данных с привилегированным и тремя "простыми" каналами. Только от привилегированного спутника можно принимать навигационные данные, и периодически нужно менять привилегированный канал.
В моем случае переключение каналов происходит полностью программно, так что оно практически не занимает времени.

После различных прикидок, я остановился на следующем подходе: на 4 спутника выделяются 4 канала, каждому каналу приема выделяется 4 мс, после получения данных со всех спутников (16 мс) формируется пауза в 1 мс. С одной стороны, она нужна для того, чтобы выполнялось правило #4, а с другой стороны - она нужна для проведения длительных навигационных вычислений.




Методика переключения каналов

Здесь хорошо видно, что каналу номер 1 "повезло" получить 7 кодов PRN из 20, и в этом канале

можно определить момент смены знака навигационных данных (если он там есть).

Замечу, что использование переключения каналов явно усложняет трекинг, так как данные в одном канале принимаются "рывками". Например, я так и не смог настроить PLL для работы в таком режиме, и он использует только данные одного PRN из четырех за период сканирования.

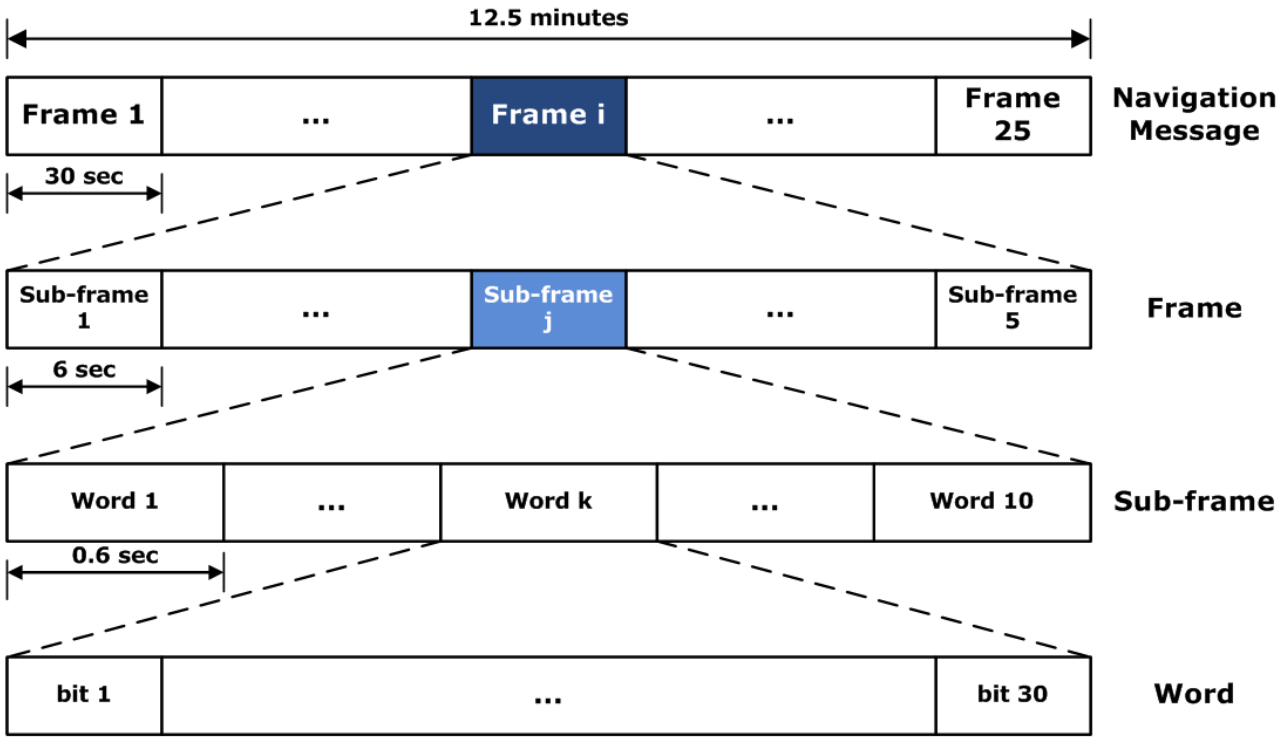
## Навигационные данные

Теперь, когда прием данных во всех четырех каналах реализован, можно поговорить про прием навигационных данных. Их источником являются результаты, выдаваемые каналом I коррелятора P (prompt). В зависимости от полярности результата, можно считать, что приняты единица или ноль данных.
Переключение каналов определенным образом усложняет выделение навигационных данных. Главная задача тут - отследить (хотя бы не очень точно) моменты смены знака навигационных бит. Зная, что период их смены строго кратен 20 мс, можно бороться с вероятным шумом. Обнаружив присутствие периодической смены бит, программа определяет момент системного времени, когда смена происходит. Таким образом, в дальнейшем, анализируя отдельные результаты трекинга отдельных PRN и зная текущее системное время, можно определить к какому именно навигационному биту они относятся.
После того, как все данные собраны и навигационный бит закончился, можно окончательно определить его знак и добавить его в массив принятых данных.

Так как для определения координат нужно знать точное время смены навигационных бит, то для его вычисления я сделал отдельный алгоритм. Напомню, что неточность связана с особенностью работы "закольцованного" коррелятора, который выдает неточные значения в моменты смены знака навигационного бита. Алгоритм дожидается, когда в собранных четырех значениях корреляции смена знака будет приходится ровно на середину массива. А далее, анализируя все четыре значения и значение текущего кода фазы, алгоритм определяет, относится ли истинный момент времени смены знака ко второму или третьему значению. Не могу сказать, что что алгоритм является абсолютно надежным - наличие сильного шума может приводить к ошибкам в определении времени.

**Декодирование навигационных данных**
Формат навигационных данных подробно описан в документе IS-GPS-200M [4]. Описание начинается в разделе "20.3.2 Message Structure".
Основные положения кратко:

- Данные передаются отдельными битами, один бит длится 20мс.
- 30 бит образуют *слово* (word). Слово всегда заканчивается контрольной суммой, которая занимает 6 бит.
- 10 слов образуют subframe. Они делятся на 5 типов, каждый тип содержит свои данные. Длительность передачи одного subframe - 6 секунд.
- 5 subframe образуют один кадр (frame). Длительность передачи одного кадра - 30 секунд. Этого достаточно, чтобы передать текущее время и эфемериды (данные об орбите одиночного спутника).
- 25 кадров образуют полное навигационное сообщение. Оно длится 12.5 мин и содержит данные альманаха для всех спутников.




Изображение отсюда: https://gssc.esa.int/navipedia/index.php/GPS_Navigation_Message

Каждый subframe начинается со специального *слова* TLM, начинающегося с 8 бит преамбулы (заголовка). Ее можно использовать для обнаружения начала subframe, но обязательно в дальнейшем проверять контрольную сумму отдельных слов. Контрольная сумма считается несколько запутанно, для ее проверки нужно знать еще и 29/30 биты предыдущего слова.

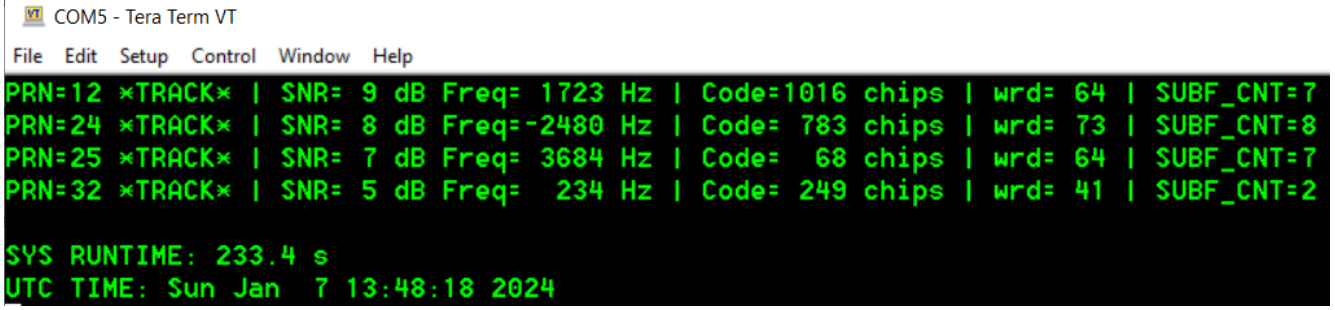
После того, как subframe целиком собран, можно провести его декодирование. Тут я тоже использовал часть кода из проекта GNSS-SDRLIB, а там используется часть кода RTKLIB.
В subframe 1 содержится информация о спутнике, некоторые коэффициенты коррекции времени. Также тут передается номер текущей GPS недели.

В subframe 2/3 передаются данные эфемерид.
В В subframe 4/5 передаются данные альманаха и некоторые дополнительные коэффициенты. В моем проекте номера используемых спутников вводит пользователь, поэтому данные альманаха не нужны. Из этих subframe, так же, как и из остальных, программа выделяет только значение текущего времени TOW.

Как я уже писал выше, использованный регулятор Costas Loop нечувствителен к изменению фазы на 180 градусов. Это означает, что полярность принимаемых данных может оказаться инвертированной (это зависит только от везения). Более того, при низком уровне сигнала, PLL может не удержать фазу, и полярность принимаемых данных поменяется "на лету". Поэтому приходится несколько усложнять подход к декодированию данных, и проверять, не инвертированы ли принимаемые данные.

**Отображение текущего состояния приемника**

Так как приемник "одновременно" обрабатывает данные сразу с четырех спутников, и число важных параметров достаточно большое, то я решил сделать отображение основных текущих параметров приема. Тут все просто - микроконтроллер формирует в памяти текстовые данные и отравляет их в UART, использую я DMA. ПК принимает их и выводит в эмулятор терминала. Выглядит это так:




Состояние приема в окне эмулятора терминала

Можно видеть номера спутников, режим приема, SNR (он вычисляется просто как отношение I/Q), частоту приема, фазу кода, количество принятых *слов* и subframe.

## Псевдодальности

Вот уже и навигационные данные приняты, и приемник отслеживает сигналы от четырех спутников. А что же с определением расстояния до спутников?

В системе GPS все спутники начинают передачу subframe строго одновременно, но из-за разницы в расстояниях от спутников до приемника они доходят о приемника немного в разное время. Задержка обычно лежит в диапазоне 65–83 мс. Напрямую величину задержки измерить невозможно, поэтому используют относительные измерения. Время получения subframe для ближайшего спутника (оно будет минимальным из всех) принимают за ноль, от него отсчитывают остальные задержки (такой спутник считается "опорным"):

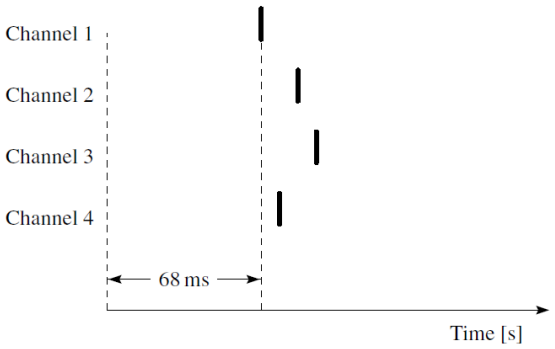



FIGURE 8.8. The transmit time and start of the subframe for four channels.

Изображение из книги [3]

Дополнительно к полученным значениям можно прибавить 68.802мс для удобства - так значения времени и расстояний будут более похожи на реальные.
Умножив полученные значения времени на скорость света - получаем значения расстояний до спутников (псевдодальности/pseudorange). Псевдодальностями они называются потому, что содержат различные ошибки. Компенсацией этих ошибок должен заниматься уже алгоритм локализации.

Системное время в этом проекте отсчитывается дискретно, с шагом 1 мс (по приходу данных от SPI). Это слишком мало для адекватной локализации - 1 мс соответствует расстоянию около 300 км. Поэтому для увеличения точности используются данные фазы кода - она измеряется с точностью 1/16 *чипа* - это около 18м.

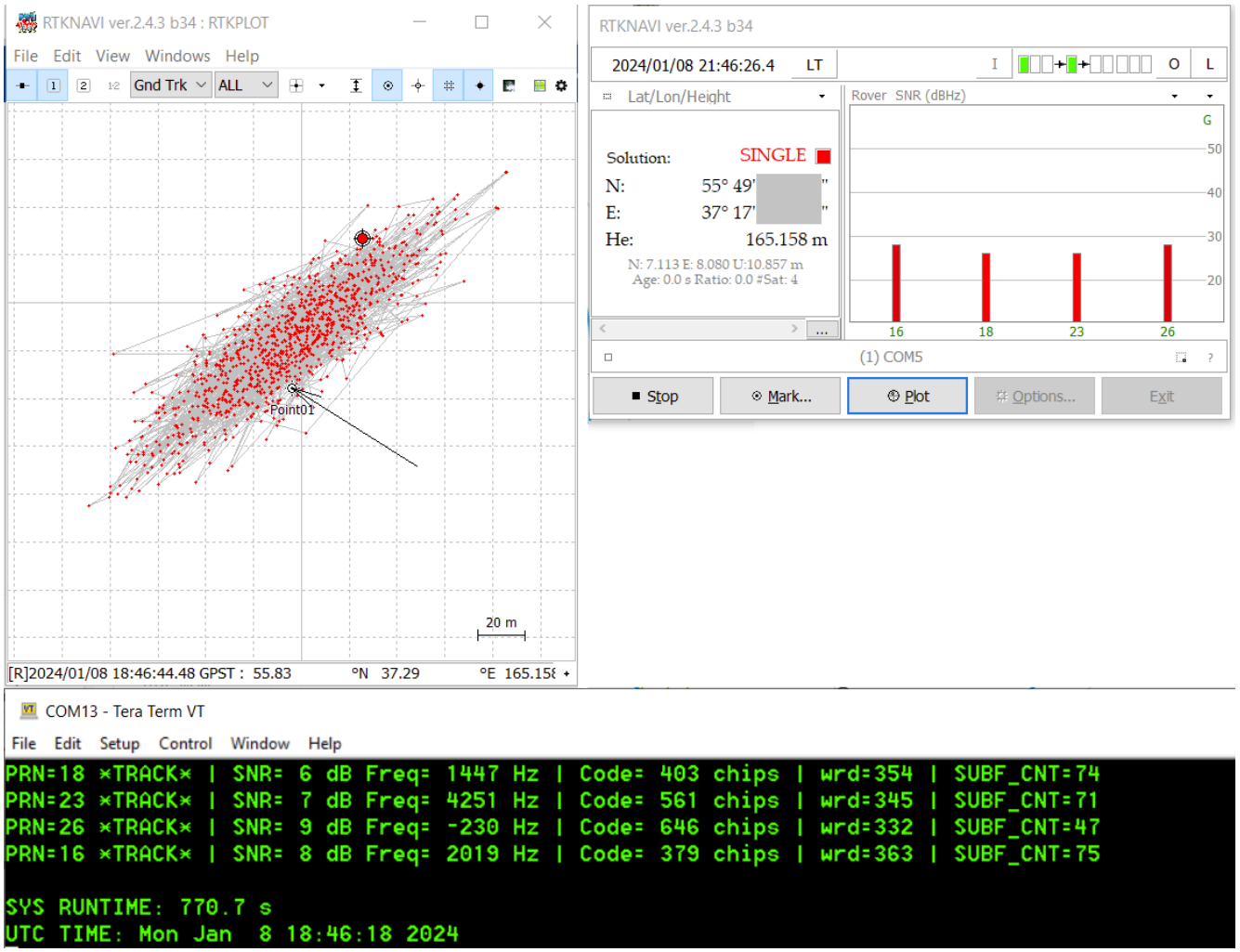
Дополнительную информацию про вычисление псевдодальностей можно найти здесь и в книге [3].

## Передача измерений на ПК

После того, как я получил от приемника вычисленные текущие псевдодальности и время, которому они соответствуют, для проверки его работы я решил передать эти данные на ПК - так же, как это сделано в GNSS-SDRLIB. Для передачи этих данных я использовал протокол RTCM 3. Он, в первую очередь, предназначен для передачи корректирующей информации для повышения точности навигационных систем, но подходит и для передачи данных о текущих измерениях приемника.

В этом проекте для формирования RTCM сообщений я использовал код из проекта RTKLIB. Для сокращения его объема пришлось значительно пройтись по библиотеке.
Передаются два вида сообщений - с данными измерений псевдодальностей и времени, которому они соответствуют (observations) - ID 1075, и с данными эфемерид - ID 1019. Благодаря тому, что декодирование навигационных данных тоже основана на коде RTKLIB, перекодирование данных эфемерид внутри программы не требуется. Упакованные данные микроконтроллер отправляет в UART при помощи DMA.

Далее эти данные можно обрабатывать на ПК так же, как я делал в предыдущей статье - при помощи RTKLIB можно определить свои текущие координаты.
В итоге получается такой результат:

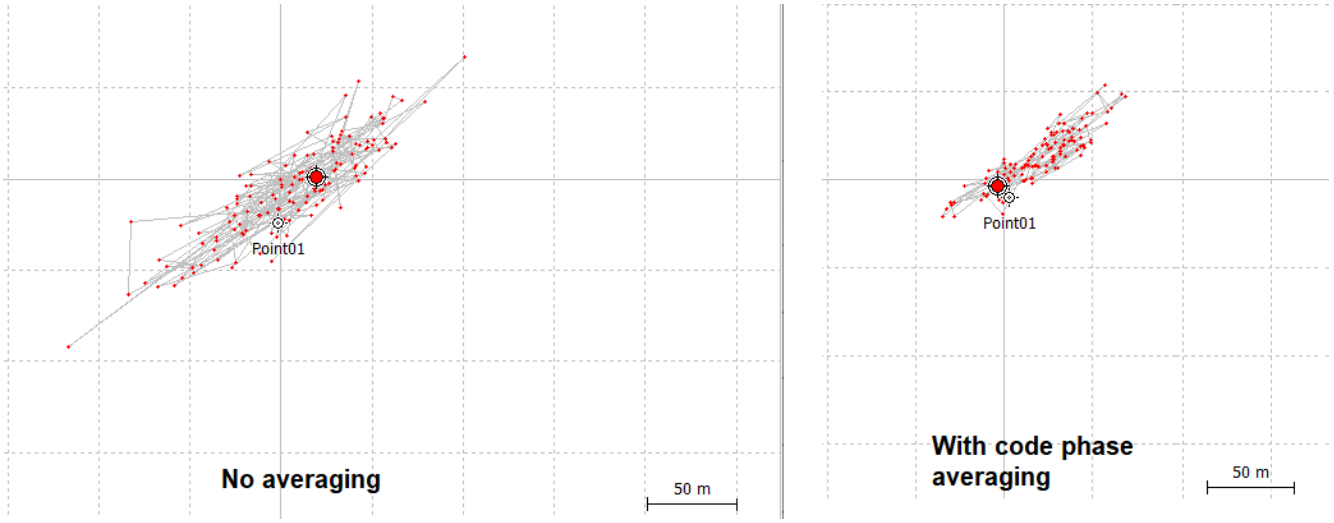



Результат работы RTKLIB. Стрелкой показано реальное расположение антенны.

Видно, что разброс координат довольно большой: ±80м. В чем-то это похоже на результаты GNSS-SDR, которые я получал ранее. К сожалению, у меня нет возможности расположить антенну под открытым небом - ее приходится устанавливать на окне балкона (внутри помещения). А на горизонте при этом - дома.
Приемник получился довольно требовательным к уровню сигнала - при слабом уровне сигнала

он не может пройти этап захвата сигнала (Acquisition). Это приводит к тому, что нужно дожидаться, когда спутники попадут в сектор, где антенна обеспечивает прием сигнала. В итоге - значение HDOP постоянно получается высоким.

С текущим алгоритмом есть возможность получать псевдодальности каждые 17мс. А что, если усреднять данные измерений фазы кода? Я попробовал, получилась такая разница (усреднение по 100 точкам, это около 400 мс):




Слева - без обработки, справа - с усреднением

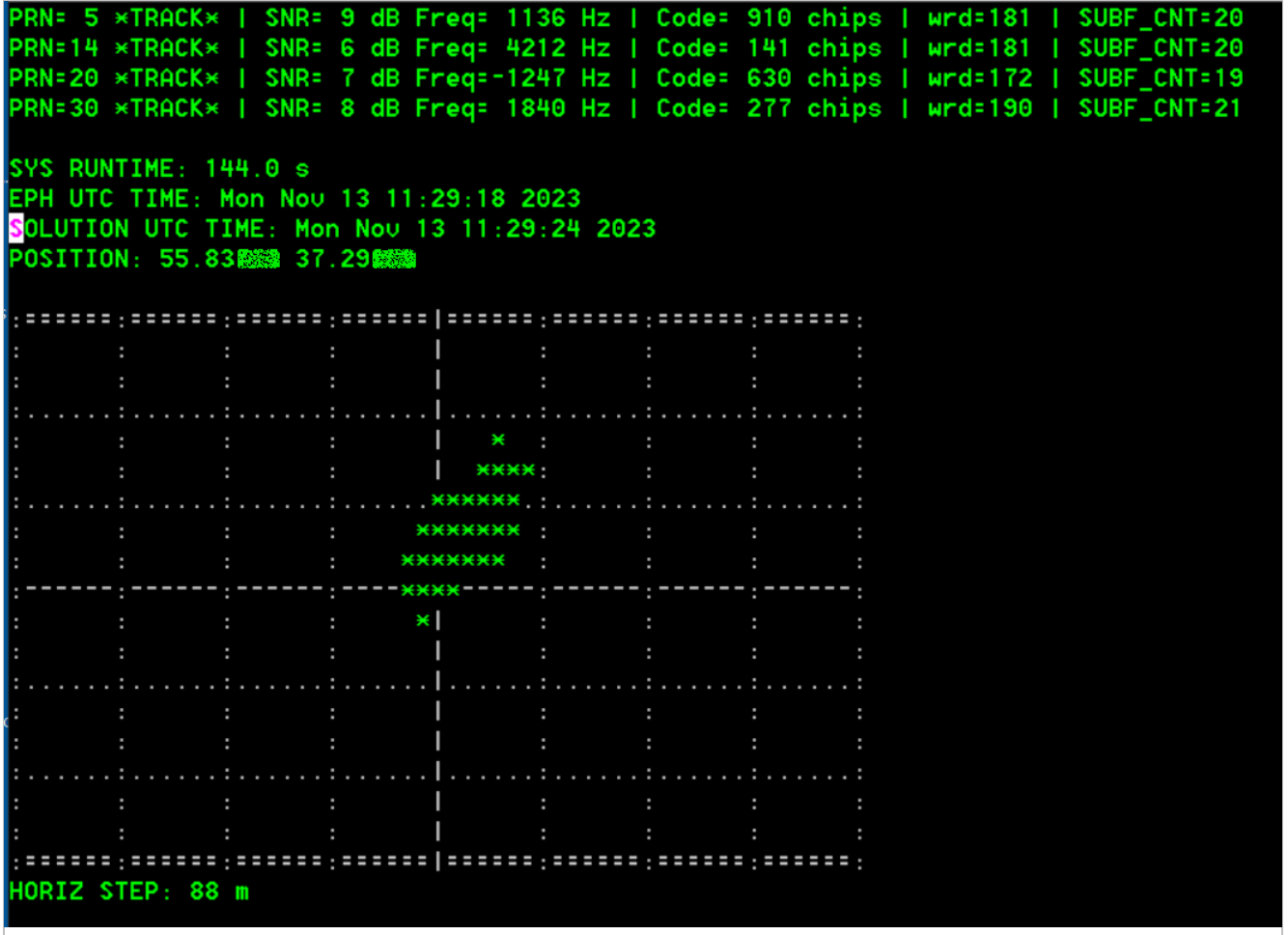
Можно видеть, что усреднение дает уменьшение уровня шума примерно в 2 раза.
Во всех случаях видно, что центр пятна явно смещен относительно истинных координат антенны (обычно это несколько десятков метров). С чем это связано - я не разбирался.

## Определение координат на микроконтроллере

Чтобы можно было назвать получившуюся конструкцию полноценным GPS приемником, нужно было перенести вычисление координат на микроконтроллер. Я рассмотрел несколько вариантов видов готовых алгоритмов, но в итоге решил использовать ту же RTKLIB.
Про то, как именно приемник производит расчет координат, можно почитать здесь и в [3].

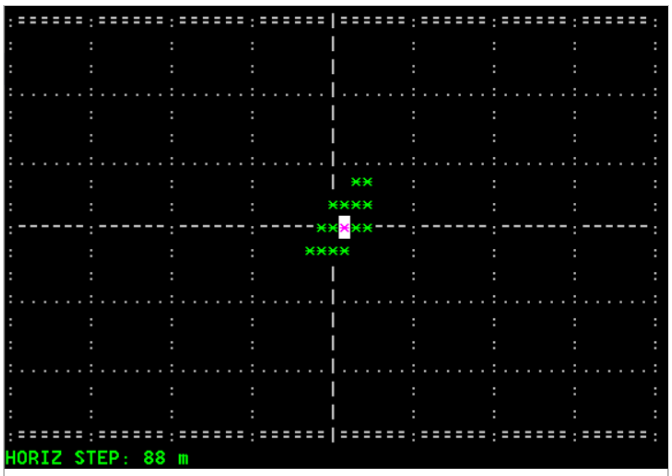
В первую очередь, я перенес все необходимые для локализации функции в один файл и убрал поддержу других навигационных систем и частот. Далее я проверил работу библиотеки на микроконтроллере. Как и ожидалось, вычисление вышло довольно долгим - до 30 мс (тестовый запуск, без работы самого приемника), так что мне пришлось заметно переделать некоторые используемые функции. Я сделал их итеративными, что было возможным за счет того, что очень часто длительные операции были связаны с вычислениями, касающимися только одного спутника, и не связаны с другими данными. Кроме того, сам по себе алгоритм вычисления координат на конечном этапе работает итеративно. В итоге удалось добиться ситуации, когда любая из функций локализации занимала меньше 1мс. Так как "окно" для вычислений появляется каждые 17 мс, одно определение координат занимает 350-700 мс.

Вычисленные координаты и текущее время выводятся вместе с данными о состоянии приема. Для большей наглядности я сделал отображение графика разброса координат при помощи псевдографики:



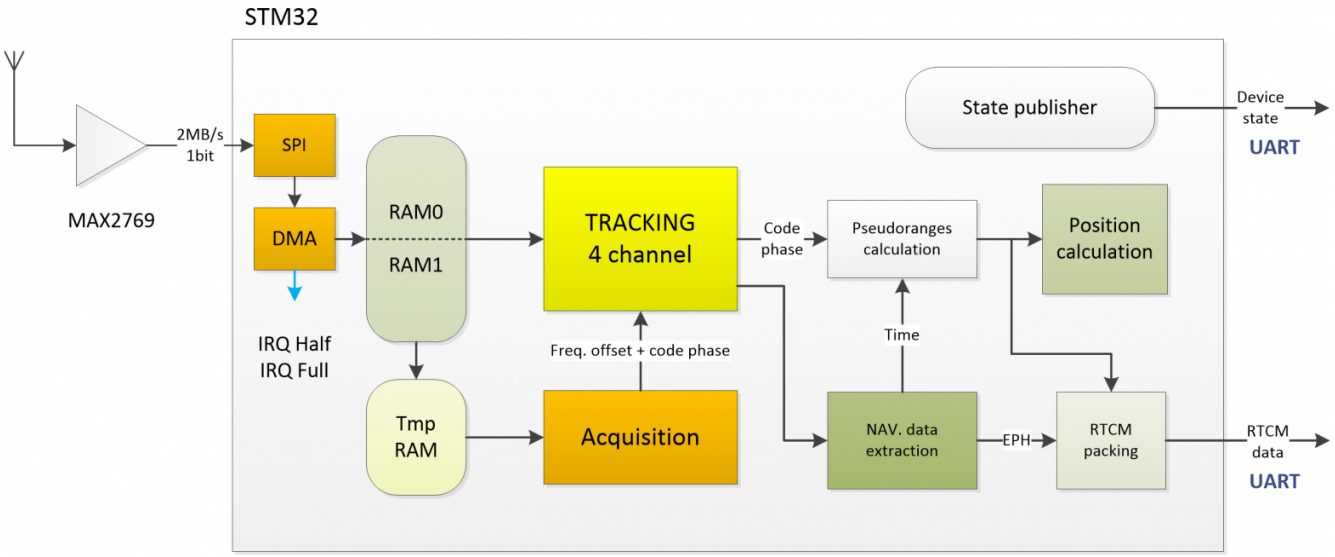

Окончательный результат работы приемника

С включенным усреднением фазы кода разброс координат выглядит так:



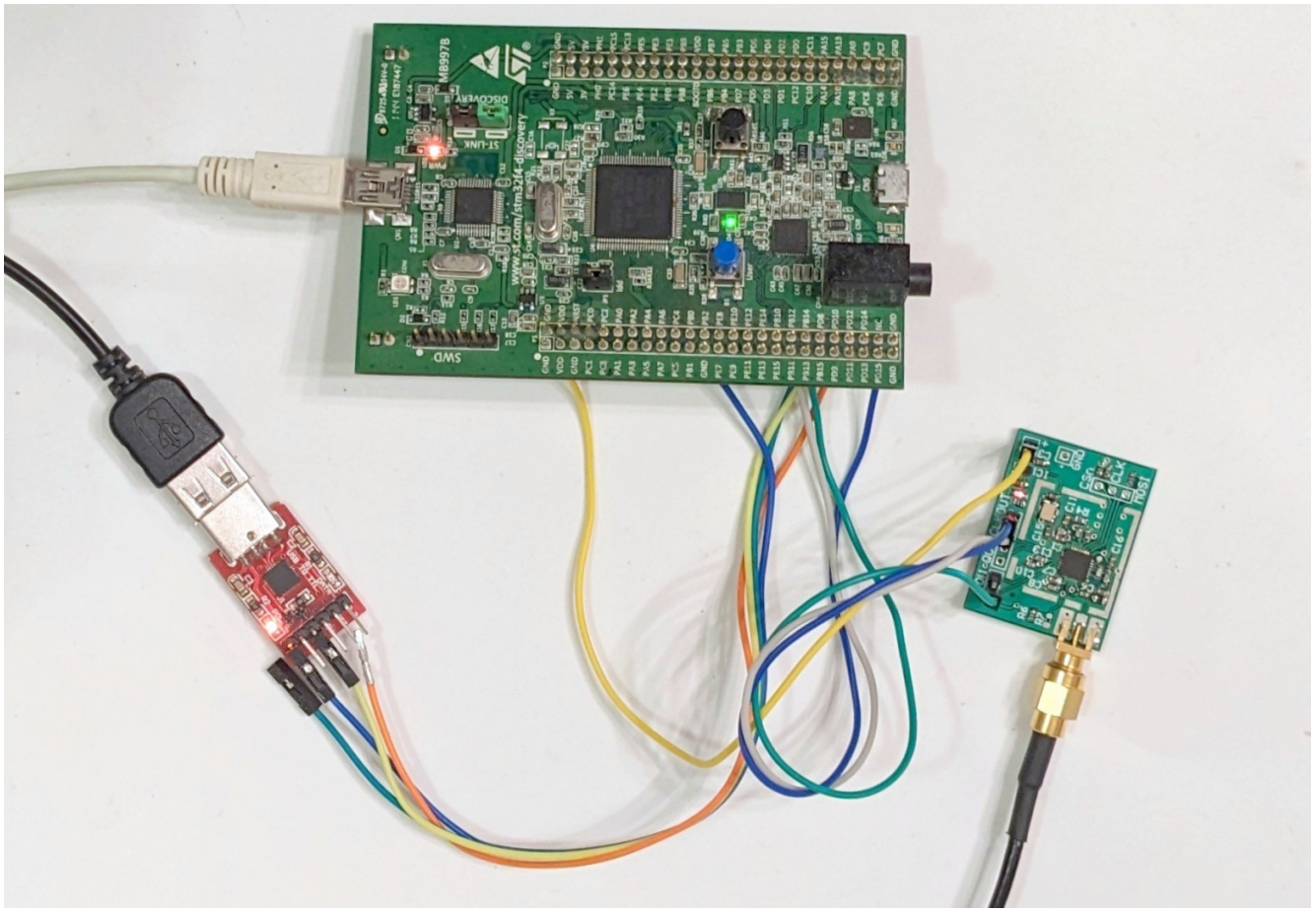

Разброс координат, те же условия, что и выше

Окончательная структурная схема программы получается такая:




Структурная схема программы STM32

Сам по себе получившийся приемник выглядит неказисто:

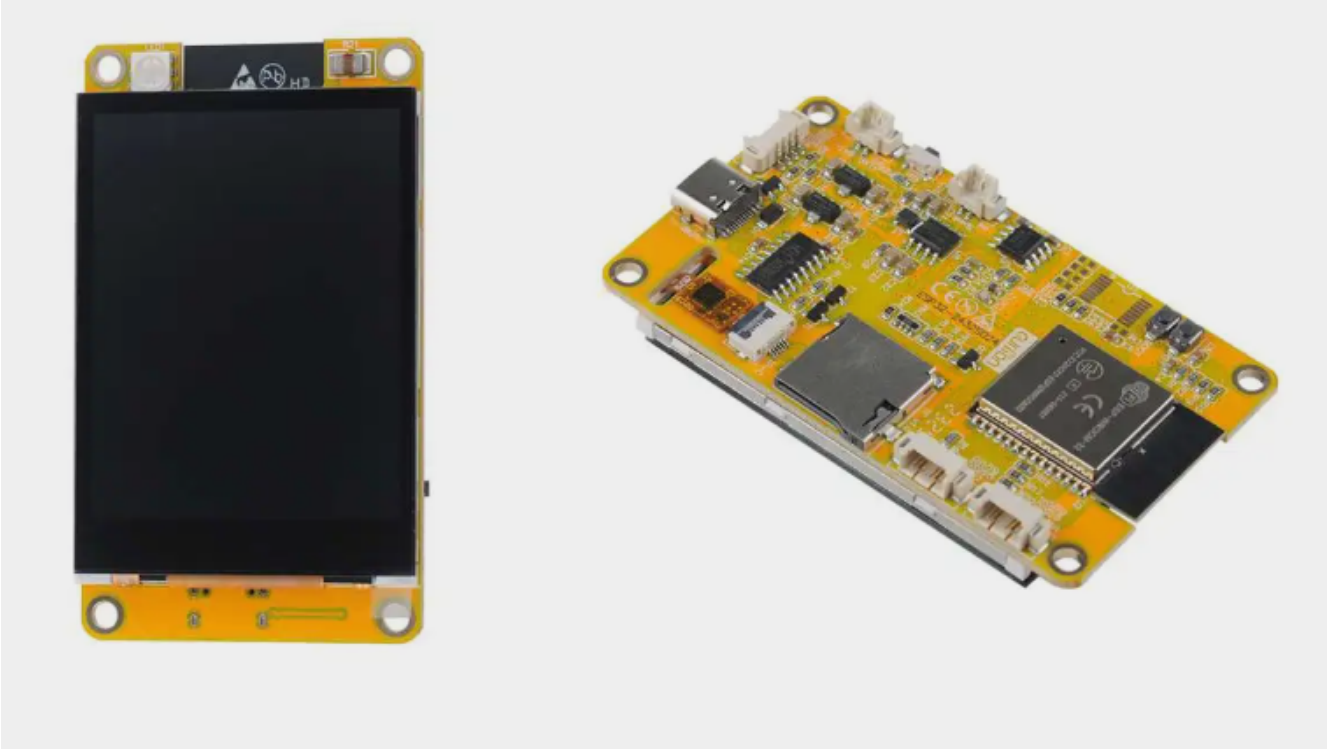


STM32F4-DISCOVERY + GPS RF Frontend

---

## GPS приемник на ESP32

У меня имелась отладочная плата, собранная на базе на EPS32 - ESP32-2432S024C.
Плата имеет LCD 320x240 с диагональю 2.4 дюйма и емкостной тачскрин. Стоит около 10$.
Выглядит она вот так:



Плата ESP32-2432S024C

Это один из вариантов "Cheap Yellow Display".
У меня возникла идея попробовать сделать GPS приемник на этой плате.

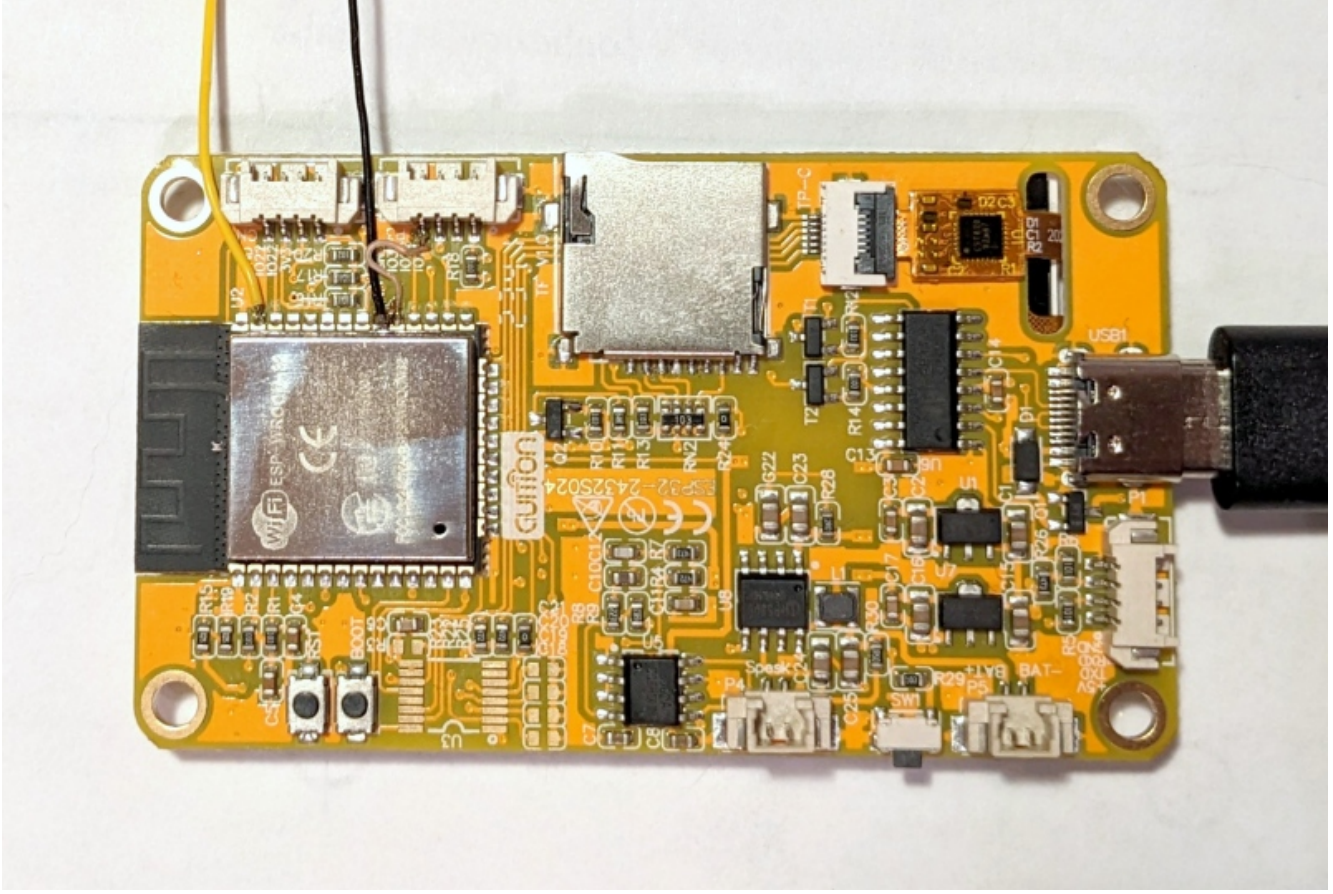
Характеристики MCU (модуль ESP32-WROOM-32):

- 2 ядра Tensilica Xtensa LX6, 32-бит
- Тактовая частота 240МГц
- Flash - 4 Мбайт
- ОЗУ - 520 Кбайт (но пользователю доступны только 320 Кбайт DRAM)

Благодаря тому, что микроконтроллер - двухъядерный, одно из ядер можно полностью отдать под графику.
Для программирования микроконтроллера я использовал фреймворк ESP-IDF вместе с плагином для VS Code.

**Получение данных от SPI**

К сожалению, у вышеупомянутой платы слишком мало свободных выводов, доступных для подключения внешних устройств - их всего 3. Судя по документации, SPI можно переназначить на любые выводы, а мне хватило бы и трех (CLK/DATA/CSn), однако этот режим может плохо работать на высоких частотах. Так что я решил использовать рекомендуемые в документации линии SPI. На этой плате они подключены к держателю microSD карты. Если карту не использовать, то их можно использовать без проблем. Я припаял нужные провода к площадкам модуля ESP32. Важно помнить про линию CSn - она должна быть подключена к земле, чтобы SPI Slave мог принимать данные. Выглядит это так:



Подключение линий SPI

Именно запуск приема данных от SPI занял у меня больше всего времени при портировании проекта на STM32. Этот микроконтроллер, также как и STM32, позволяет использовать SPI вместе с DMA. Однако имеющаяся документация на периферийные модули несколько скудная. Основной упор в документации производителя сделан на использовании готовых библиотек высокого уровня, входящих в состав ESP-IDF. И вот тут возникает проблема - библиотека для работы со Slave SPI рассчитана на получение конечного объема данных. Насколько я понял, на неразрывное получение данных ее настроить невозможно. Пришлось очень сильно переписать библиотеку.

Используемый тут контроллер DMA заметно отличается от того, что используется в STM32. Для начала, он позволяет передать за раз всего 4090 единиц данных. Тут мне повезло - SPI работает в 8-битном режиме, таким образом нужно сделать 16368/8=2046 передач для приема 1мс данных. Если бы ограничение было бы меньшим, это условие могло бы привести к усложнению программы.

Также, в отличие от STM32, часть настроек DMA хранятся в ОЗУ пользователя - в специальных структурах, называемых дескрипторами. В частности, каждый дескриптор содержит информацию о том, по какому адресу нужно записывать данные, каков объем данных, и указатель на следующий дескриптор. Завершив обработку одного дескриптора, DMA перегодит к следующему. В этом проекте программа создает два дескриптора, которые ссылаются друг на друга. Таким образом и получается организовать кольцевую запись данных.

Также в документации упомянуто: "If DMA is enabled, the rx buffer should be word-aligned (starting from a 32-bit boundary and having a length of multiples of 4 bytes). Otherwise, DMA may write incorrectly or not in a boundary aligned manner." У меня размер данных не кратен 4 байтам, но данные принимаются нормально. Если бы пришлось выполнять это условие, то пришлось бы заметно переделать способ приема данных.

В итоге я смог написать библиотеку, работающую примерно так же, как и в STM32 - два буфера

по 1 мс, по заполнению буфера DMA генерирует прерывание.

**Обработка данных**

Так как FreeRTOS является частью ESP-IDF, я использовал возможности RTOS для обработки принимаемых данных. Для приема данных GPS созданы две задачи - более приоритетная и менее приоритетная. Более приоритетная задача ждет прерывания от DMA, и получив его, занимается копированием данных и Tracking.
Менее приоритетная задача отвечает за алгоритмы Acquisition и вычисление координат приемника. За счет вытесняющей многозадачности уже не нужно разбивать вычисление координат на итерации, за счёт чего читаемость этой части кода заметно повысилась.
Быстродействие тоже заметно возросло - вычисление координат занимает около 20-40 мс (хотя, думаю, более высокая частота MCU тут тоже помогает).
В итоге код SDR, перенесенный с STM32, заработал сразу без доработок.

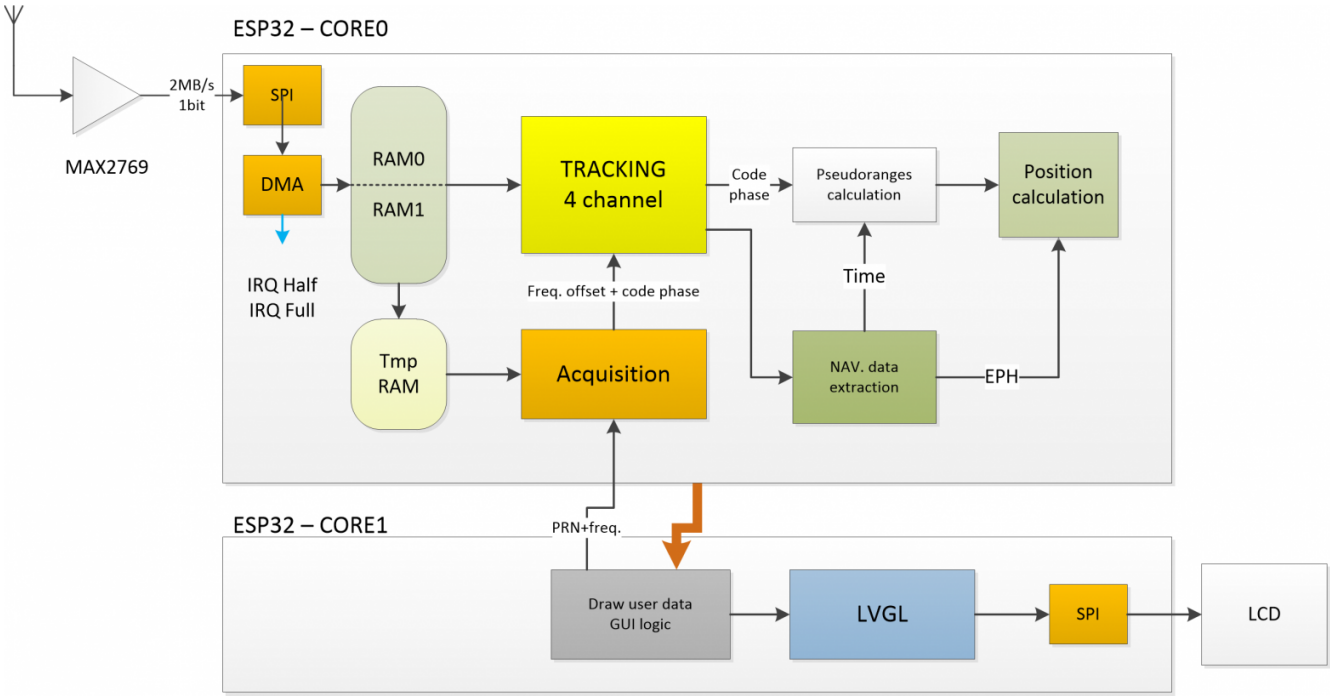
**GUI**

GUI в этом проекте я сделал на LVGL. Мне уже приходилось иметь дело с LVGL, так что это не составило большой сложности. Пришлось немного поковыряться с библиотеками - выбранная связка версий ESP-IDF / LVGL / драйвера дисплея и тачскрина имели небольшие проблемы с совместимостью. Дизайн GUI я набросал в SquareLine Studio.

Вся информация отображается на четырех элементах Screen, которые можно перелистывать жестами.

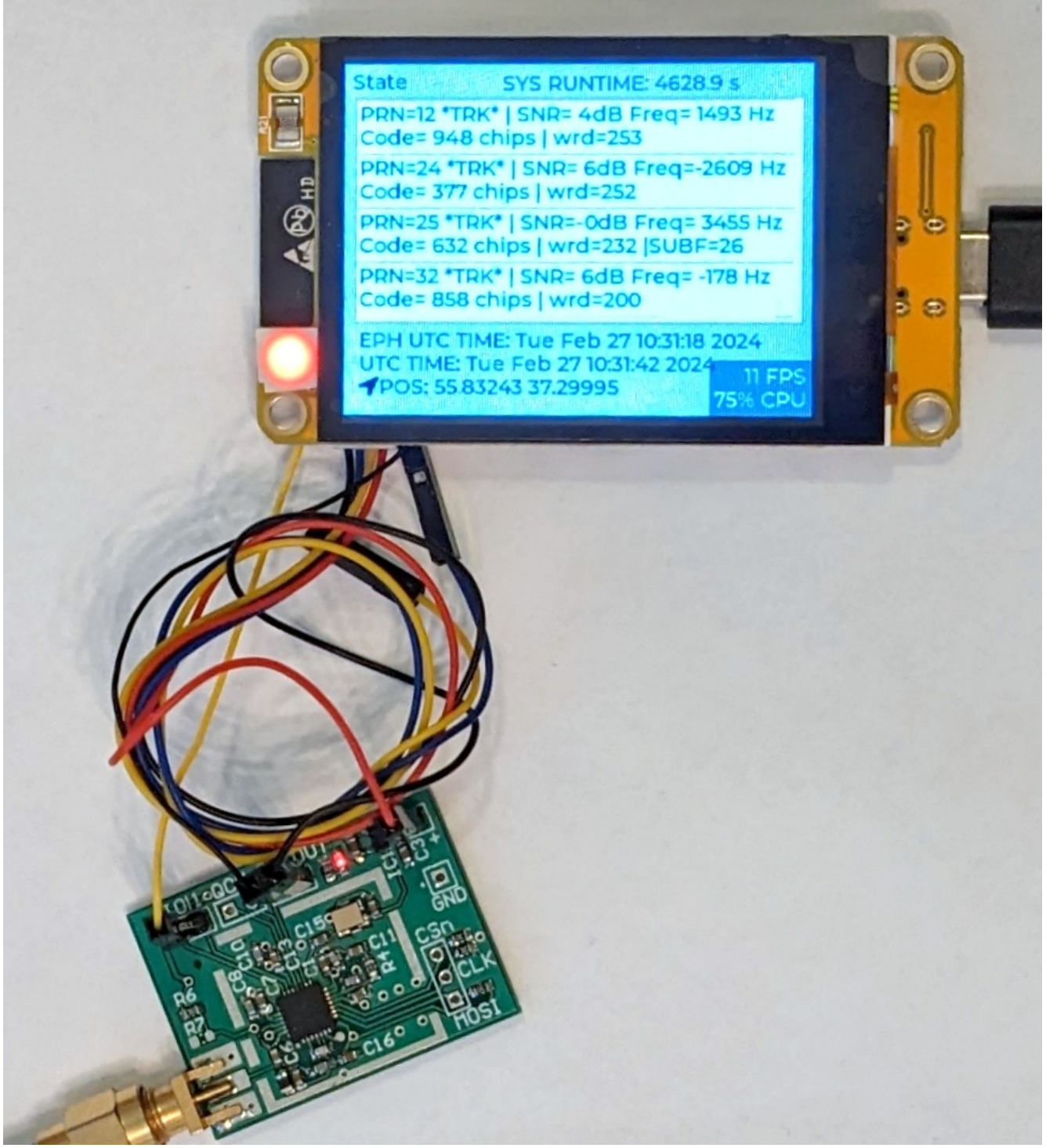
Первый экран - ввод настроек приемника (номер PRN спутника, и, опционально, смещение частоты).
Второй экран - отображение текущего состояния приемника, тут выводятся примерно те же параметры, что и в консоль в случае STM32.
Третий экран - отображение диаграммы распределения значений I/Q.
Четвертый экран - отображение диаграммы разброса определенны координат.

Структурная схема программы:




Структурная схема программы ESP32

Фото получившейся конструкции:




Экран отображения состояния приема.

Видео с демонстрацией работы приемника - на базе STM32 и ESP32:

## Воспроизведение записанных данных GPS

Для тестирования работы приемника очень полезно иметь возможность воспроизводить заранее записанные сигналы - чтобы не нужно было выставлять антенну на окно, волноваться о качестве сигнала, ждать, пока будет определена частота каждого из спутников. На ПК можно просто считывать данные с диска. Для проверки кода на MCU нужно было иметь возможность передавать записанные данные с ПК на микроконтроллер в виде потока бит. Я рассматривал разные варианты реализации этой задачи, и в итоге остановился на использовании воспроизведении данных при помощи микросхемы FT232H.
Нашелся даже готовый проект SpiLight, который мог воспроизводить данные по SPI.
Но у него был недостаток - данные передавались слишком медленно, а максимальный объем передаваемых данных был ограничен 64 КБ. Так что мне пришлось немного доработать код SpiLight, результат выложен в репозитории для STM32 ниже.
Максимальная получившаяся частота SPI - 15 МГц, это не совпадает с частотой 16.368 МГц, на которой записаны данные, но приемнику проблем не создает - просто системное время приемника идет несколько медленнее. Из-за ограничений FT232H между каждыми 64 КБ данных возникает небольшая задержка, но она тоже не мешает приемнику - так как сами данные не повреждаются.

Если использовать ранее записанные данные для формирования потока RTCM, можно обнаружить, что RTKLIB (программа rtknavi.exe) не может его обработать. Насколько я понял, это связано с тем, что в данных RTCM не содержится информация о текущей GPS-неделе, и rtknavi вычисляет ее значение по системным часам ПК. Так что для тестирования обработки данных RTCM я дополнительно использовал программу RunAsDate, которая подменяла информацию о текущем времени для rtknavi.

## Итоги

В итоге я смог сделать GPS-приемник на микроконтроллере. Приемник может принимать сигналы

от 4 спутников, обрабатывать их в реальном времени и определять собственные координаты.
Но это - чисто демонстрационный проект, для практического применения у него слишком много недостатков - низкая точность определения координат, необходимость указывать номера спутников вручную, слишком долгий поиск частоты спутников, acquisition требует, чтобы сигнал от спутника был достаточно сильным.
И все же приемник работает, мне удалось обойдясь без ПЛИС или мощных DSP. Изначально я боялся, что на STM32F4 приемник сделать не удастся, потребуется куда более мощный STM32H7.

Исходные коды проекта:
https://github.com/iliasam/STM32F4_SDR_GPS

https://github.com/iliasam/ESP32_SDR_GPS

---

Проект самодельного GPS приемника на ПЛИС [1]: Homemade GPS Receiver. RF front-end здесь тоже самодельный.

Очень крутой проект GPS приемника из 90-х, собранного на аналоговых микросхемах, распространенных цифровых микросхемах и MC68010 в качестве CPU [2]: A homemade receiver for GPS & GLONASS satellites. Также тут подробно описана теория работы GNSS систем.

Книга про построение программных GNSS приемников [3]: A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach

Документ IS-GPS-200M [4], описывающий данные, передаваемые системой GPS.

**Теги:** GPS, stm32, sdr

**Хабы:** Глобальные системы позиционирования, Программирование микроконтроллеров, DIY или Сделай сам, Электроника для начинающих



290 Карма 98.2 Рейтинг

@iliasam

## Комментарии 29

**Raegdan** вчера в 11:13

~~Вопрос~~ крик души автору как специалисту в GNSS, пусть и не по данному проекту. Насколько реально при помощи ЦОС отфильтровать сигнал навигации от отходов жизнедеятельности вояк, чтобы в крупных городах снова можно было нормально пользоваться навигатором...?

Ответить

**iliasam** 23 часа назад

Так вояки тоже не дураки, там люди явно постарались, чтобы отфильтровать помехи было сильно проблематично.
Так что думаю, что без системы направленных антенн и многоканальных приемников не обойтись.

Ответить

**Khabrovchanin** 23 часа назад

Сигнал губят не совсем вояки, а переотраженки от зданий и сооружений и иные помехи. Есть специальные антенны, снижающие такие факторы, но они нифига не компактные, называются Choke Ring. Если же речь именно о зонах, попадающих под целенаправленное искажение, то оно для того и сделано, чтоб его не обходили ) Можно сделать невысокое кольцо из меди или иного металла, вокруг антенны, снизив боковые помехи, но при этом уменьшится видимое созвездие, если высокая точность не нужна, то частично поможет. Но это не для карманных решений. Я живу не в очень крупном городе, и по роду профессиональной деятельности работаю с GNSS оборудованием, но еще не встречался с полным отказом обработки данных приёмником. Хотя в Москве и Питере народ жалуется, на систематические высокоскоростные перемещения в пространстве по мнению их навигатора )

Ответить

**iliasam** 23 часа назад

В Москве точно глушат, причем не просто генератор шума ставят, а имитируют сигнал спутников (видел ситуацию, когда телефон показывает кучу спутников с последовательно идущими номерами 1-12, уровень сигнала высокий, но координаты приемник определить не может).
Где-то пишут, подмена сигнала сложней - там приемник к аэропортам "перебрасывает".

Ответить

**Raegdan** 23 часа назад

Именно что. Едешь на машине, попадаешь под говнилку, стрелка начинает рандомно прыгать по району, навигатор судорожно перестраивает маршрут и говорит полную ересь, несколько раз из-за этого пропускал повороты и перестроения. Это большими кусками внутри ТТК и практически сплошняком внутри Садового. Как полагаю, излучается с крыш. А телепортация во Внуково это точечно возле Кремля, она там давно и в принципе особо не мешала.

Ответить

**Raegdan** 23 часа назад

Понятно что не карманное, но в площадь автомобильной крыши думаю ведь реально вписать? Сделать ФАР, чтобы сделать ДН более острой в небо, и еще окружить ее таким экраном.

Ответить

**iliasam** 23 часа назад

"Сделать ФАР"
Подозреваю, что это будет стоить >2000$, а то и больше ...

Ответить

**Raegdan** 22 часа назад

Ну, пассивная ФАР в принципе ведь вещь не космической сложности, особенно при наличии единичных антенн промышленного производства (а для GPS они есть). Разместить их матрицей с точно выверенным шагом и завести в сумматор кабелями выверенной длины. Нарезать в размер и качественно оконцевать кабеля тоже могут хоть на алике. Сумматор, у которого 1536 МГц попадало бы в рабочий диапазон, можно попробовать поискать в оборудовании для спутникового ТВ или сотовых репитеров. Разве нет?

Ответить

**VO_Obsidian** 22 часа назад

Думаю можно уложиться в 500-600. В целом nullsteering антенну можно сделать на основе того же KrakenSDR и ПК с N100. Правда с krakensdr я не совсем уверен что сигнал и помеха влезут в его динамический диапазон, там же АЦП всего 8 бит. Но сделать свой приемник на 6-7 когерентных каналов задача не то чтобы плёвая, но выполнимая. Как вариант взять железо от RSP1, там и L-диапазон и 12 бит. Единственное, такой приемник будет работать только по GPS L1C и Galileo E1. ГЛОНАСС с его частотным разделением банально не влезет в дешевый SDR, а кодовое на других частотах чем у GPS.

Ответить

**Khabrovchanin** 21 час назад

Мой текущий приемник принимает около 40 спутников при открытом небе, надо сразу десяток ФАР ставить, чтоб получить нормальное решение, ибо они не по центру, а размазаны по всему небосводу. Я еще молчу, что при этом используется 3 частоты сигнала. В общем на крыше ведерко металлическое, невысокое, в центр антенну, дешево и сердито, большую часть помех отсеет, ценой снижения качества навигационного решения. Но для бытовых целей пойдёт.

Ответить

**VO_Obsidian** 18 часов назад

Пробовали такой эксперимент, в итоге выяснили что даже в теории не получится. Внутри ТТК помеха примерно так на 40-50 дБ мощнее сигнала. И это не имитационная помеха, а просто белый шум на всю полосу L1. Никакая антенна не даст такой разницы между главным и боковыми лепестками, при условии что главный лепесток широкий, а не 1-2 градуса.

Ответить

**Khabrovchanin** 18 часов назад

Ну да, мое предложение не панацея на все случаи, но в некоторых случаях помогало. Но там были именно боковые помехи и не такие мощные, но работать не давали нормально, решение портилось сильно. Всё зависит от помехи и её мощности. Ну и глушилка может стоять на здании, тогда ничего не поможет, сигнал же также сверху, как и спутниковый, не экранируешь.

Ответить

**acc0unt** 3 часа назад

Обычно такого ада с "помеха на 40-50 дБ мощнее сигнала" по физическим причинам не происходит. Слишком много надо в такое всаживать энергии. Но вот сигнал GPS - это как раз исключение из правил.

Спутники очень далеко, и сигнал от них невыносимо слабый. Поэтому любой мудозвон с дешёвой глушилкой может убить GPS практически наглухо в радиусе пары сотен метров. А если глушением занимается правительство, то дело вообще швах.

Ответить

**chnav** 1 час назад

>> любой мудозвон с дешёвой глушилкой может убить GPS практически наглухо в радиусе пары сотен метров

С этим действительно беда, дальнобойщики часто глушат трекеры ПЛАТОНа, а заодно всех соседей по дороге и дорожную технику - грейдеры, бульдозеры и прочих геодезистов.

Ответить

**almaz1c** 22 часа назад

Спасибо за статью!

Возможно ли это устройство доработать до RTK модема сантиметровой точности позиционирования?

Ответить

**iliasam** 20 часов назад

Не думаю, точность тут получилась довольно низкая.

Ответить

**melodictsk** 21 час назад

Кстати в мск вполне сносно работает gps в диапазоне l5, почти без телепортации. Без него боль.

Ответить

**chnav** 19 часов назад

Посмотрел свой смартфон - к сожалению не умеет L5 ((

Вообще удивляет нежелание производителей делать многочастотные GPS-приёмники. Тому же гражданскому сигналу L2C уже сто лет в обед, имеется на всех спутниках, запущенных после 2005.
Источник: New Civil Signals

Ответить

**vadimk91** 19 часов назад

Так вроде если приемник умеет работать с разными системами (GPS, Глонасс, Beidou и проч.) - он уже по определению многочастотный, нет?
А кстати, где вы смотрели в телефоне поддерживаемые диапазоны? у меня ничего подобного не нашлось, GPS Test тоже такого не знает.

Ответить

**chnav** 18 часов назад

Как раз в GPS Test и смотрел. Однако это интересный вопрос, поищу спецификацию чипа.

>> так вроде если приемник умеет работать с разными системами (GPS, Глонасс, Beidou и проч.)

Это "мультисистемные приёмники". Многочастотные это когда несколько диапазонов L1=>1.5-1.6 GHz, L2=>1.2 GHz и др.

Ответить

**Gudd-Head** 21 час назад

> SDR в данном случае означает, что приемник не содержит готовых GPS-модулей или специализированных микросхем для обработки GPS сигналов

MAX2769 - universal GPS reciever.

Ответить

**iliasam** 20 часов назад

"для **обработки** GPS сигналов"
MAX2769 просто переносит частоту принятого сигнала вниз, и оцифровывает сигнал.
Как таковой, обработкой я этот процесс назвать не могу.

Ответить

**BarsMonster** 17 часов назад

Илья, это огонь! Тысячу лет назад гонял поиск в матлабе по данным с SDR - а теперь в МК влезло )

Систематическая ошибка - может быть из-за того, что из псевдодальности нужно вычесть время распространения сигнала в кабеле? Ну или подобрать задержку эксперементально пока ошибка не снизится.

А насчет шума - если в коммерческих приемниках усреднение убирать - они тоже шумят ппц. Потому фильтруют дико, секунд за 5-10 с моделью движения чтобы в глаза сильно не бросалось.

Ответить

**chnav** 16 часов назад

>> может быть из-за того, что из псевдодальности нужно вычесть время распространения сигнала в кабеле?

Категорическое нет. Для всех спутников (каналов) длина RF-тракта одинакова и она автоматически попадёт в dT (расхождение часов). В этом и прелесть систем GNSS в отличие от старых дальномерных систем от радиогеодезии.

PS: проблема с задержкой сигнала имеется у ГЛОГАСС т.к. у каждого спутника своя частота. Вот там нужны поканальные калибровки.

Ответить

**1CHer** 16 часов назад

Статья огонь. Респект и уважение! Но у меня даже повторить за автором не получится я думаю. Были намётки кстати делать подобное с помощью нейросети: сгенерировать набор и скормить, далее пускай находит похожее и говорит координаты но дальше идеи не пошло.

Ответить

**Prokop1977** 15 часов назад

Большое спасибо за статью! За обе части. Четко и подробно изложено. Особенно хочу отметить то, что Вы не пропускали те моменты, где у Вас что-то не получалось, а подробно излагали. Идущим за Вами это поможет. Удачи в последующих проектах!

Ответить

**VBKesha** 13 часов назад

Спасибо за статью!

Ответить

**VO_Obsidian** 13 часов назад

Статья отличная! У меня появился вопрос по цифровому квадратурном детектору. А что, такая схема будет работать? Там же во всех схемах, работающих во временной области, нужны ФНЧ в том или ином виде. Согласно таблице здесь. Или такая схема как в статье будет работать исключительно с сигналами ГНСС?

Ответить

**iliasam** 13 часов назад

Здесь данные после переноса частоты подаются сразу на коррелятор. А процесс корреляции моно рассматривать как процесс фильтрации: https://ru.wikipedia.org/wiki/Корреляционный_фильтр
А благодаря тому, что данные PRN - имеют широкую полосу, фильтр получается мало чувствительным к помехам.

Ответить