СЕКРЕТЫ SI473X

ДЕЛАЕМ ПРИЕМНИК И ИЩЕМ СКРЫТЫЕ ВОЗМОЖНОСТИ МИКРОСХЕМЫ SDR

Сегодня я расскажу о том, как устроены современные вещательные приемники, на примере SI473X — семейства однокристальных SDR-приемников. Заодно напишем собственную библиотеку для управления этими чипами. Спросишь, зачем нам еще один приемник? Чтобы поупражняться в их создании, опробовать интересную микросхему и, конечно же, узнать много нового!

Candidum
duospirit@gmail.com

КАК РАДИОПРИЕМНИКИ СТАЛИ ШИРПОТРЕБОМ

Прогресс не стоит на месте: в течение предыдущего столетия стоимость радиоприемников снижалась, при этом их характеристики становились все лучше. Так, в 20-е годы XX века основной вклад в стоимость вносили радиолампы — вспомни хотя бы первый супергетеродин Армстронга, который мы уже упоминали, говоря об истории супергетеродина.

На момент своего появления он казался совершенно безумным, так как содержал восемь ламп — огромное количество для того времени. А ведь ему нужны были еще батарейки общим размером с небольшой чемодан!

В 1930-х подобный приемник уже был вполне реален и даже производился серийно, а кроме того, появились лампы косвенного накала, которые можно было запитать от сети. Да и цены стали не такие заоблачные. В итоге приемник стоил примерно как сейчас айфон, и его уже можно было поставить на стол, не рискуя сломать последний.

Следующий этап удешевления и миниатюризации проходил достаточно медленно, лампы дешевели и уменьшались в размерах, совершенствовалась схемотехника. Продолжалось это вплоть до 1960-х годов. А прорыв случился в начале пятидесятых, когда появились первые серийные транзисторы и на них построили первый серийный приемник Regency TR-1.

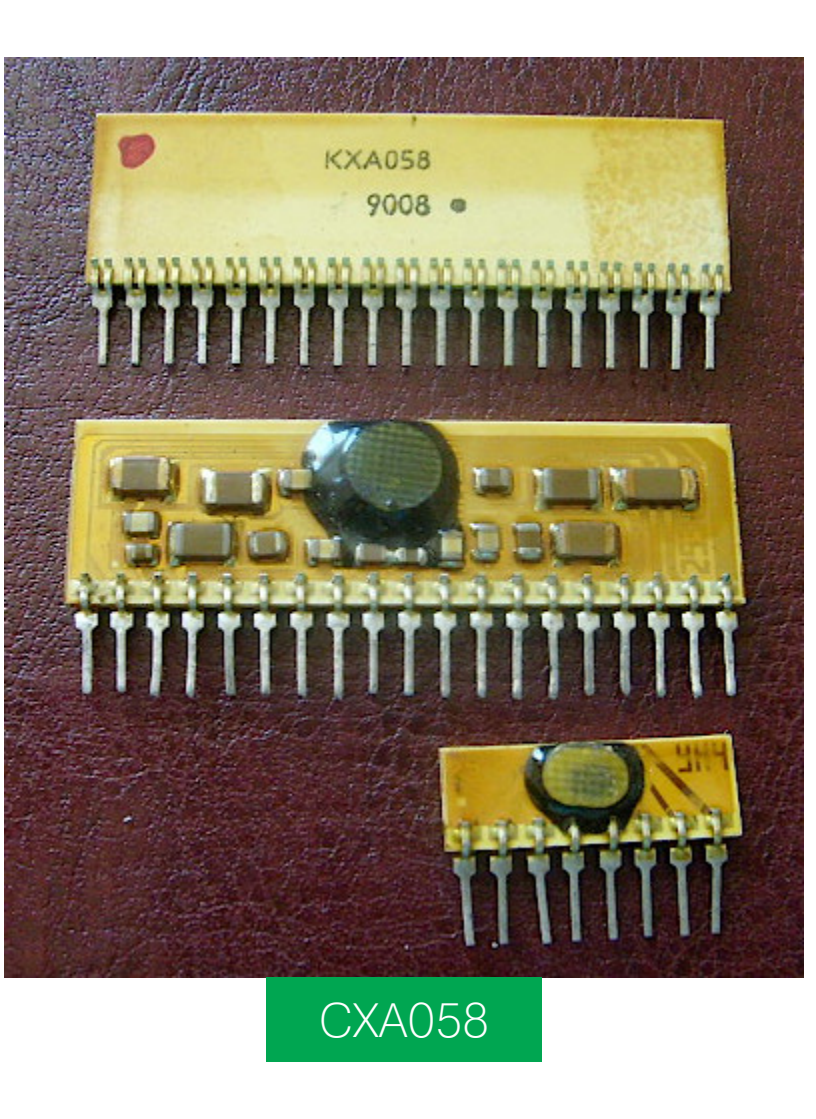
По характеристикам он уступал ламповым того времени и стоил заметно дороже, но его уже можно было положить в карман. А дальше транзисторы потихоньку дешевели, их параметры улучшались, а вместе с ними становились меньше и экономичнее приемники. Появились интегральные схемы, и где-то к 1970-м годам количество транзисторов в устройстве перестало существенно влиять на цену. Все больший вклад в размер и цену стали вносить контуры промежуточной частоты и входные перестраиваемые цепи.

Очередной рывок произошел в начале восьмидесятых, когда инженерам фирмы Philips удалось уместить весь радиочастотный тракт в одну микросхему. А кроме того, за счет схемотехнических ухищрений избавиться от всех контуров, кроме гетеродинного. Микросхема получила название TDA7000, а прототип приемника, представленный в рекламных целях, выглядел довольно-таки необычно.

Прототип приемника на TDA7000

Штука получилась на редкость удачная, поэтому вскоре появились TDA7021 (PDF) с поддержкой стереокодирования и TDA7088 (PDF), где добавилась возможность автопоиска станций. В последней микросхеме использовалась небольшая цифровая часть, которая за этот самый поиск отвечала. Впрочем, там все было устроено достаточно примитивно, но продержалась такая конструкция достаточно долго. Это именно те приемники, которые встраивали чуть ли не в зажигалки в начале 2000-х.

Российские разработчики хоть и отставали, но переняли опыт, в результате чего появилась знаменитые К174ХА34 (TDA7021), К174ХА42 (TDA7000) и очень забавная гибридная схема СХА058.



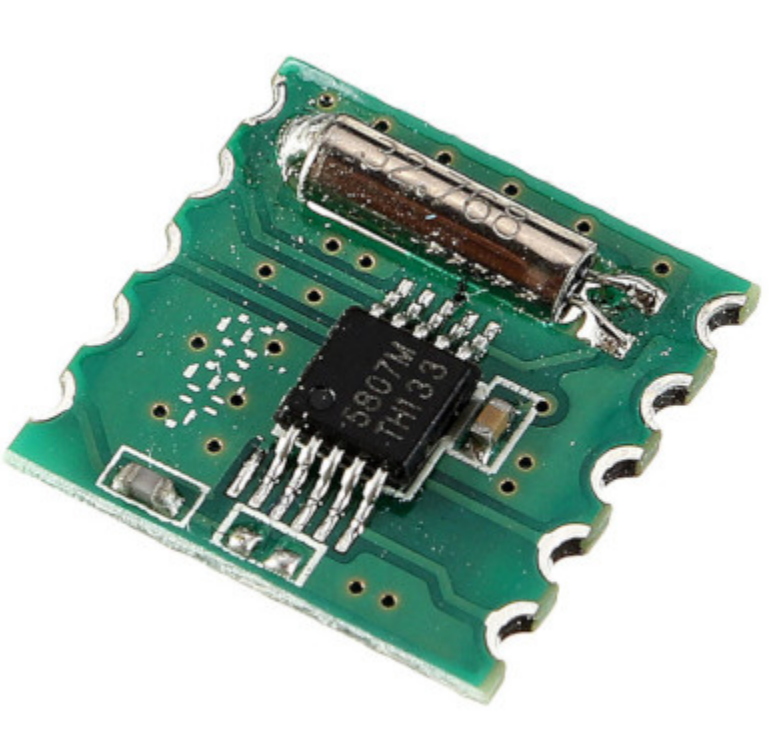
СХА058

А вот на создание аналога TDA7088 ресурсов у отечественного производителя уже не хватило, или, скорее, стало не до того. В любом случае, сейчас все эти чипы считаются устаревшими и не производятся, за исключением клонов TDA7088, но и ему, видать, недолго осталось.

Сегодня наступила эра SDR/DSP-приемников, в которых основная обработка сигнала выполняется математически на оцифрованных данных, мы это уже обсуждали, когда собирали ZetaSDR. Но там обработка оцифрованного сигнала происходила на ПК. А можно ли обойтись без компьютера? Да легко: в 2001 году Philips выпустила чип TEA5767 (PDF), представляющий собой однокристальный цифровой приемник. Этот чип требовал минимум обвязки, имел цифровое управление и позиционировался (PDF) как удобный вариант для встраивания в различные гаджеты типа MP3-плееров и мобильных телефонов. Среди его достоинств — кварцевая стабилизация частоты и возможность декодировать стерео.

TEA5767 с полной обвязкой

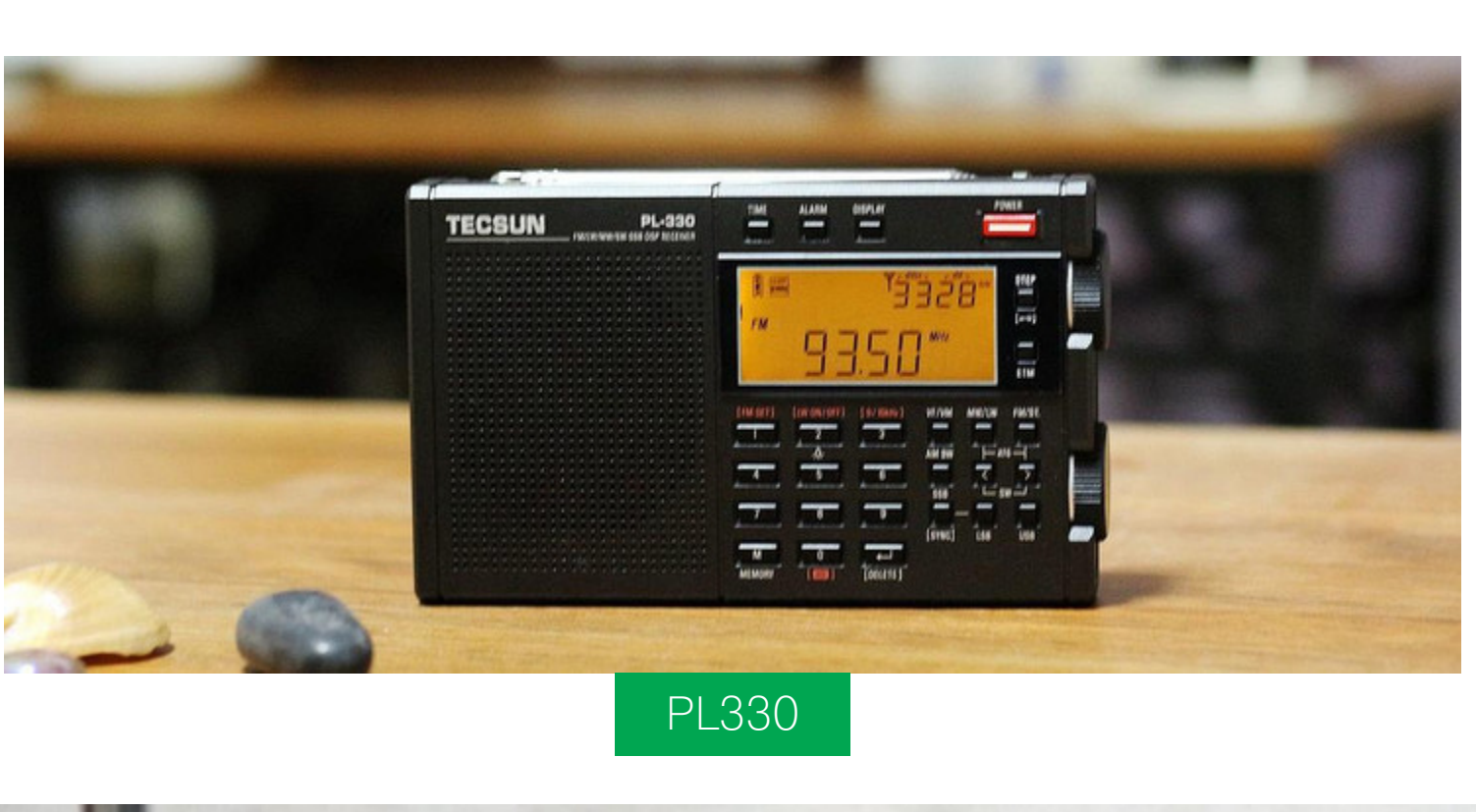
Чуть позже появился более совершенный чип RDA5807. Он избавился от последнего колебательного контура в обвязке. Собственно, там и обвязки-то не осталось, при этом принимаемый диапазон был заметно расширен (64–108 МГц), появилась поддержка RDS. Чувствительность стала повыше, качество звука тоже, и, что самое удивительное, эта кроха способна тянуть 32-омные наушники без дополнительного усилителя. И все это меньше чем за десять рублей! А сверх того, чип имеет обратную совместимость с RDA5807, да и вообще способен работать без управляющего микроконтроллера. Но с контроллером все же веселее.



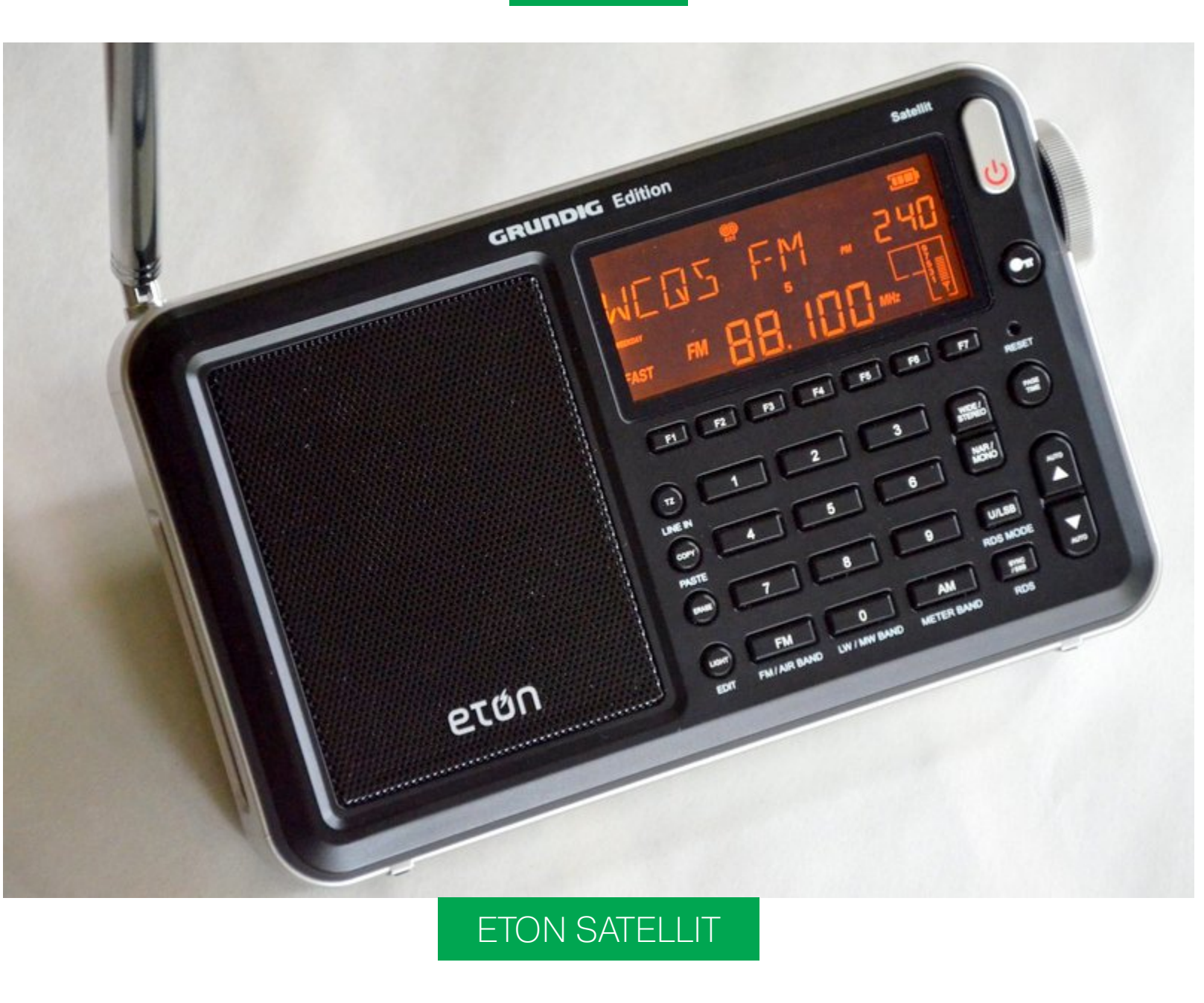
RDA5807 с обвязкой

Но даже все перечисленное не предел: в чип можно запихнуть еще и ДВ/СВ/КВ-приемник, как это сделано в KT0915 (PDF), AKC6951 (PDF) (тут еще и первые несколько каналов TV принимать можно) и SI473X, о которых мы и будем говорить дальше.

Мы создадим современный радиоприемник, подобный современным коммерческим образцам, таким как PL330 и ETON SATELLIT. Но наше изделие будет при этом максимально простым и эффективным.



PL330



ETON SATELLIT

ПОЧЕМУ SI4734

SI4735 отличается от других упомянутых чипов тем, что поддерживает патчи прошивки, а это открывает доступ к дополнительным функциям. Так, в сети есть патч, который позволяет принимать сигналы с SSB-модуляцией. Что в ней такого, спросишь ты? Да в общем, ничего особенного, просто на ней работают любители в КВ-диапазонах, и их порой интересно послушать. И это, наверное, самый простой вариант такого приемника.

Хорошо, с SI4735 разобрались, а почему в заголовке значится SI4734? Дело в том, что все микросхемы SI473X совместимы «pin в pin» и отличаются только набором функций. Младшие модели (SI4730, SI4731) поддерживают длинные волны и FM, а старшие модели (SI4732, SI4735) поддерживают еще и короткие волны и RDS. SI4734 поддерживает КВ, но не умеет RDS. Кроме всего прочего, они здорово различаются по цене: SI4730 стоит примерно 100 рублей, SI4734 — 150, SI4735 — порядка 500 рублей. Правда, всего год назад они были минимум в три раза дешевле, ну да это известная сейчас проблема.

Патч официально поддерживает только SI4735, на ней я и хотел экспериментировать. Но купленный мною экземпляр оказался нерабочим, поэтому я поставил SI4734-D60, который имелся в загашнике. А заодно попробовал скормить этому чипу патч, и, к моему удивлению, он сработал. Так что, если тебе не нужен RDS, можно сэкономить.

Обрадовавшись такому успеху, я попробовал поковырять SI4730-D60, тем более что в сети проскальзывала информация, будто некоторые из этих чипов могут работать на КВ. Однако у меня они не заработали и патч на них тоже не встал. Очень вероятно, что патч сработает и на SI4732, поскольку китайцы часто добавляют эту микросхему в наборы своих приемников и заявляют о поддержке SSB.

СХЕМОТЕХНИКА

Для наших экспериментов мы соберем относительно несложную конструкцию, состоящую из двух блоков: блока управления и блока приемника. Блок управления соберем на STM32F030, добавим к нему энкодер, дисплей OLED и восемь кнопок. От кнопок можно вовсе отказаться, но с ними управлять приемником намного удобнее. За клавиатуру будет отвечать PCF8574, очень удобная микросхема — расширитель портов с I2C-интерфейсом. Введение расширителя портов хоть и усложняет схему, но упрощает разводку платы и опрос кнопок. Питать все это дело удобно с помощью LiPO-аккумулятора, поэтому добавим туда еще контроллер заряда и DC/DC-преобразователь на RT9136 для питания контроллера. Использование активного преобразователя целесообразно в плане повышения КПД.

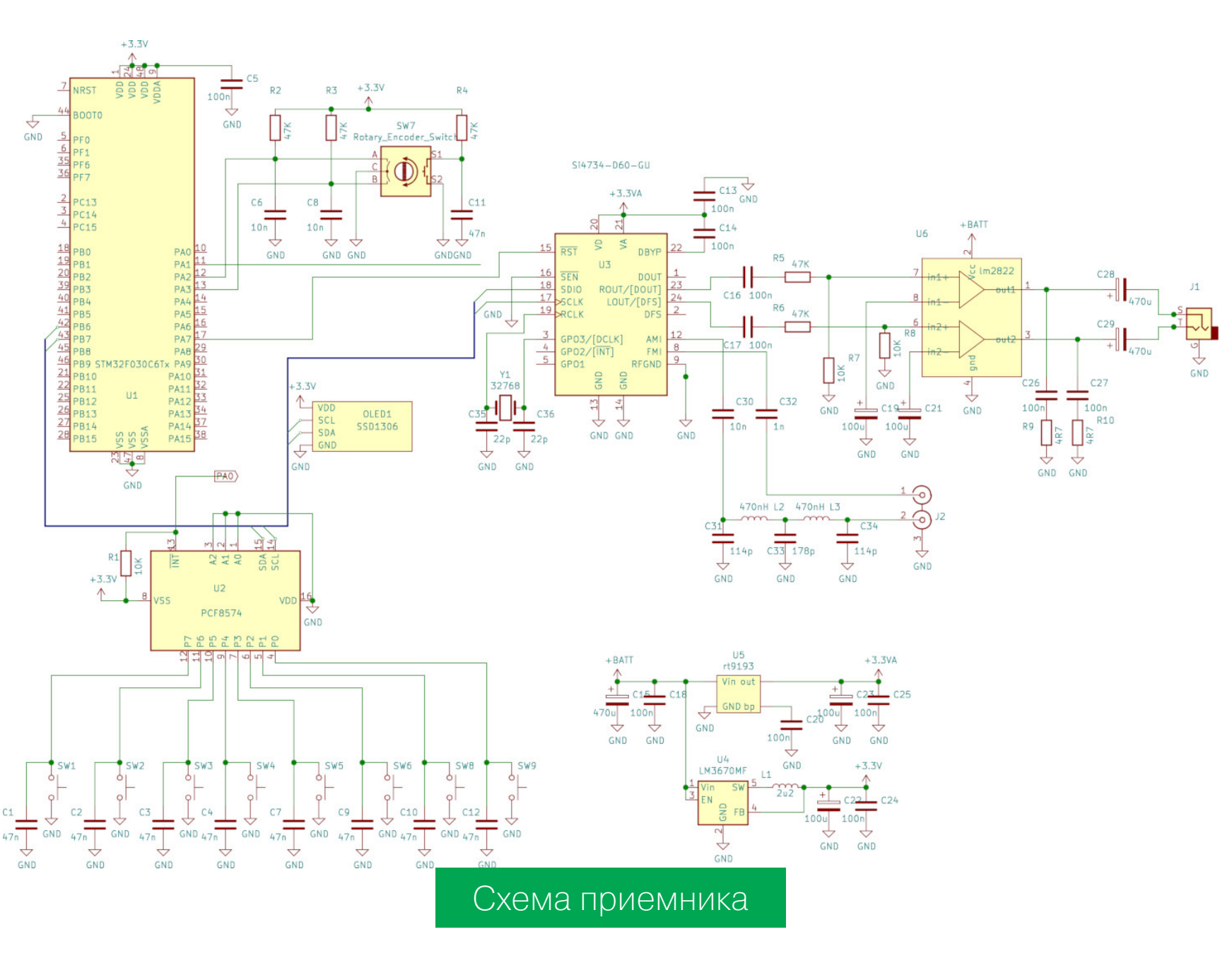


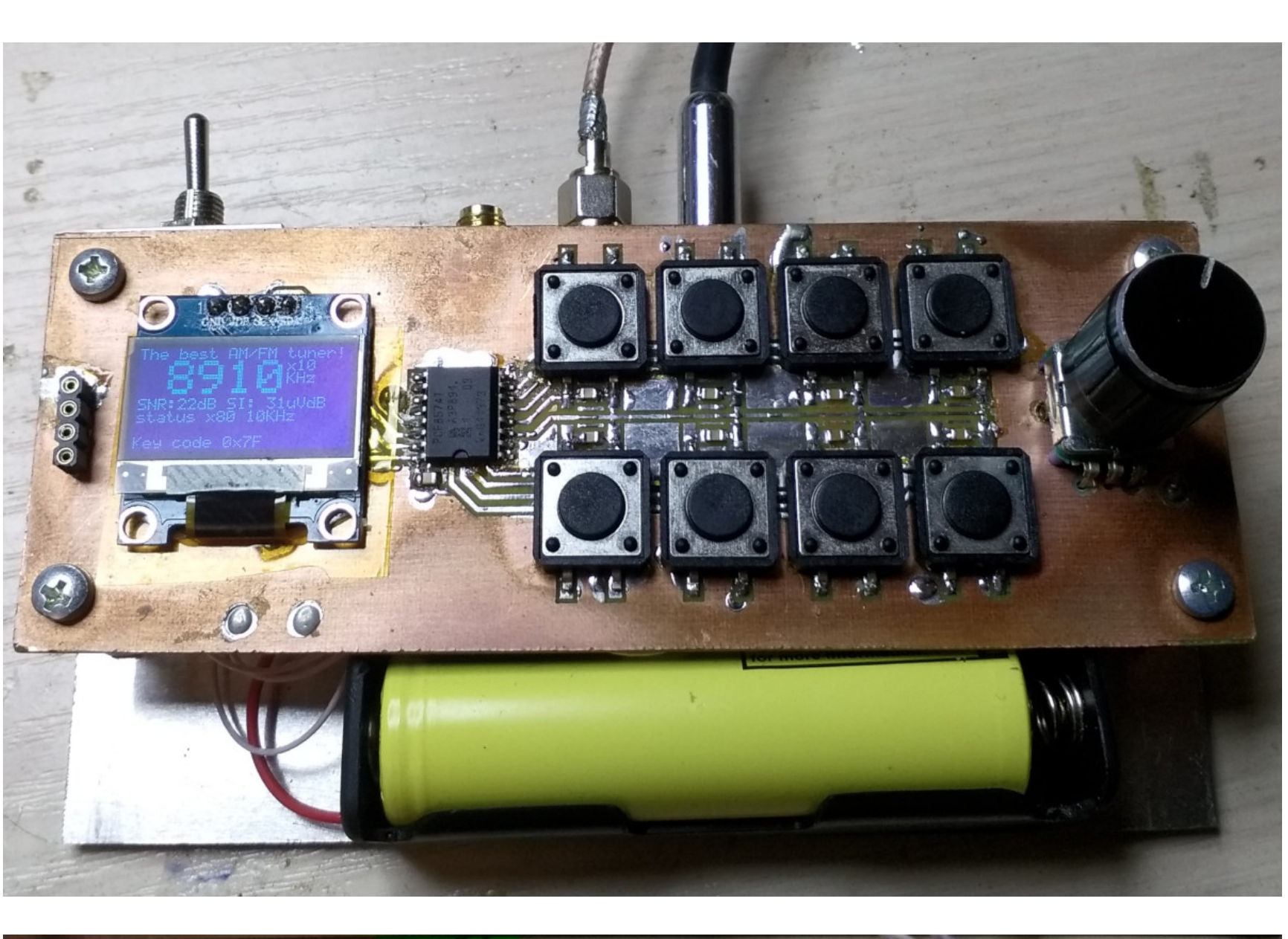
Схема приемника

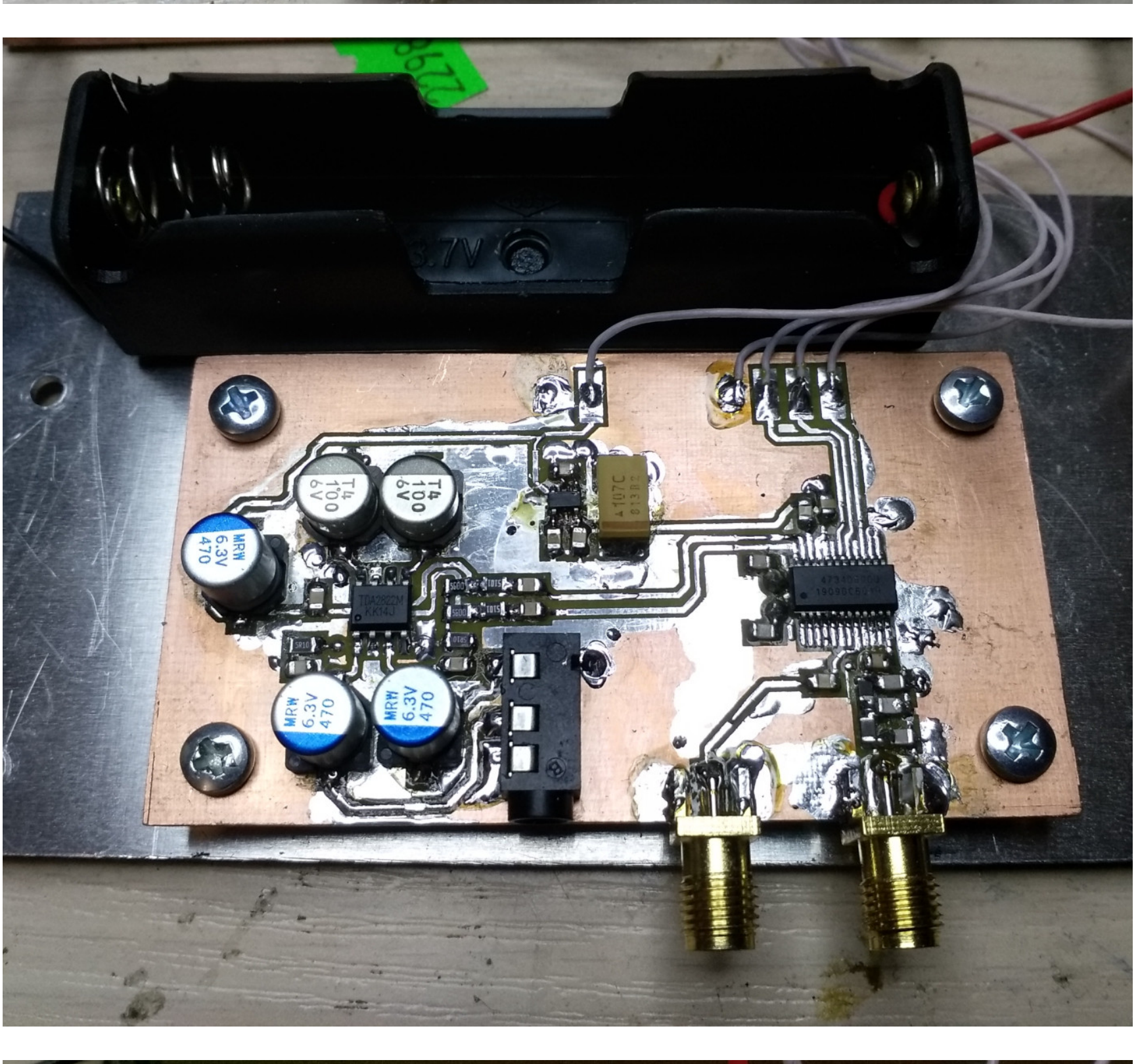
Выходной мощности SI4735 недостаточно для раскачки стандартных 32-омных наушников, поэтому нужен аудиоусилитель, даже два, так как у нас стерео. В качестве усилителя использована микросхема TDA2822 (PDF) в стандартном включении. Это не лучший вариант по двум причинам: во-первых, у нее слишком высок коэффициент усиления, а во-вторых, на мой вкус, она слишком шумит. Лучше на эту роль подойдет LM4863 (PDF), но у меня ее не оказалось под рукой. Тем не менее TDA2822 недурно справляется со своей задачей.

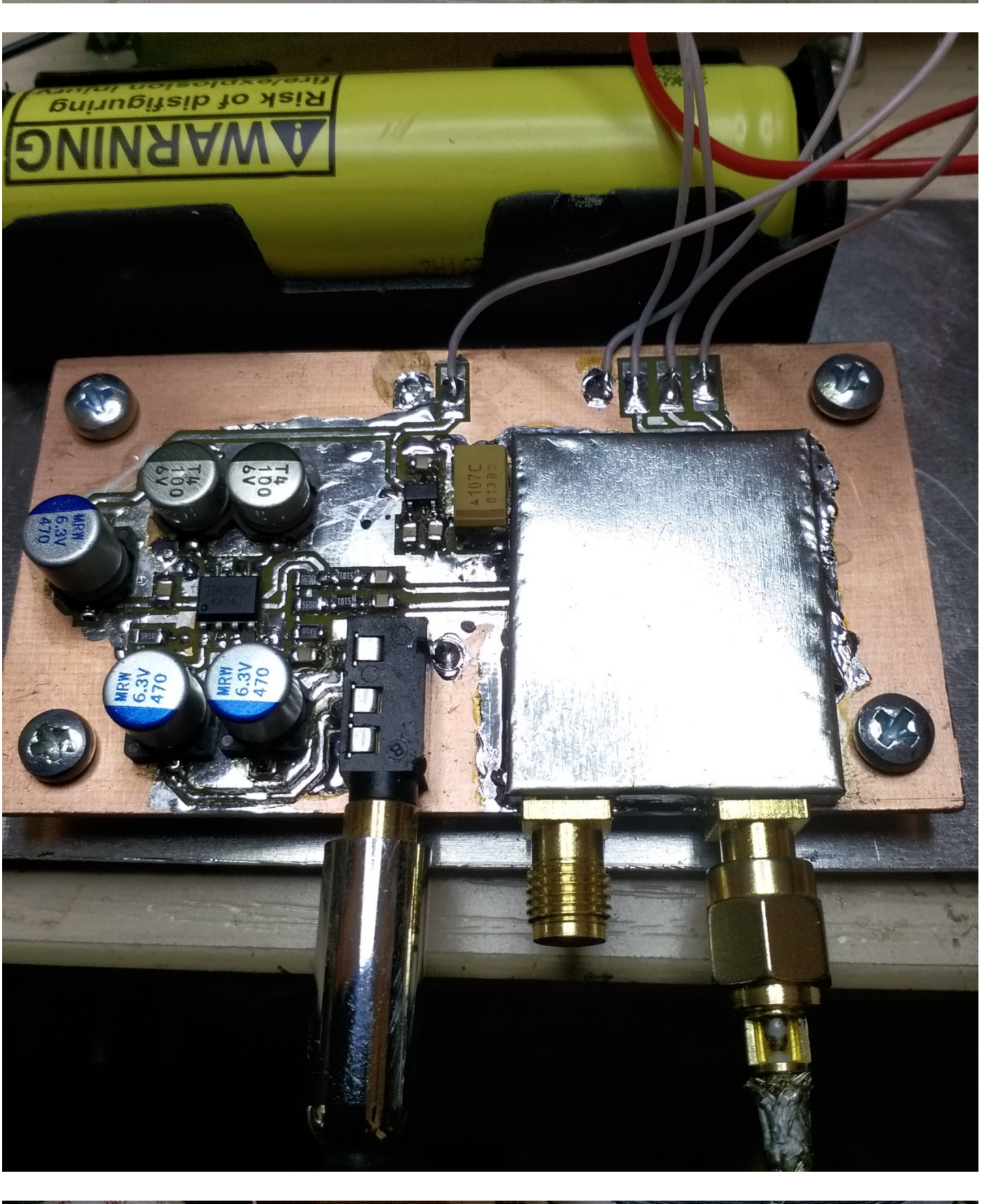
В заводских решениях обычно используется УВЧ и магнитная антенна, мы же поступим проще: поставим на вход фильтр 5-го порядка с частотой среза и будем использовать полноразмерную антенну — все равно на штырь в квартире можно ловить только помехи, FM и пару китайских станций в хороший день. Что же касается FM-входа, то ему комфортно и без входных цепей. Кроме того, саму SI4734 вместе со входными цепями мы поместим в экран из жести (плата двухсторонняя, вторая сторона — сплошная медь), благо это совсем не сложно. Использование внешней полноразмерной антенны сильно снизит наводки от цифровой части и избавит от УВЧ.

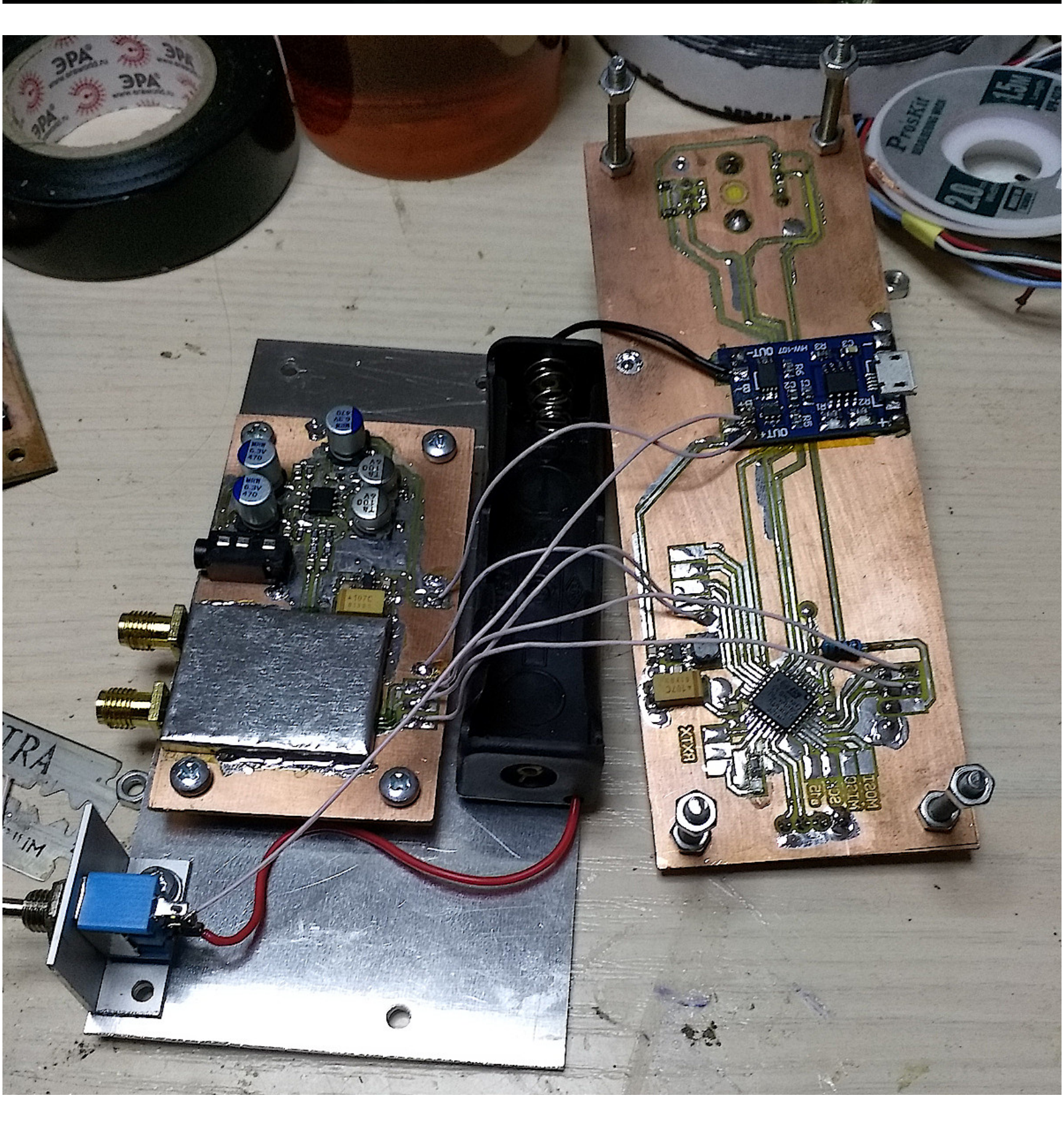
Что касается этой самой цифровой части, то тут каких-либо особенностей нет. Схема, платы и прочее лежат на GitHub. Вешать постоянно обновляющийся дисплей и клавиатуру на одну шину с SI4734 — не очень хорошая идея из-за возможных помех, однако остановка контроллера и выключение дисплея на слух не вносит изменений. Отсюда можно сделать вывод, что в городе гораздо больший вклад в качество приема вносит зашумленность эфира.

Оформлено это в достаточно минималистичном стиле, впрочем, корпуса я делать никогда не любил. У меня получилось что-то среднее между макетом и законченным устройством, но транспортировку и полевое использование приемник пережил не поморщившись.









INFO

Предвидя вопросы, скажу сразу, что управляющий блок можно собрать и на Blue Pill, и на ARDUINO, в последнем случае на Али можно купить уже собранную плату. Обойдется это примерно в 3000 рублей. А за дополнительные деньги к этому делу можно докупить корпус. Но это не наш метод, мы же собрались поковыряться с SI4734!

Продолжение статьи →

СЕКРЕТЫ SI473X

ДЕЛАЕМ ПРИЕМНИК И ИЩЕМ СКРЫТЫЕ ВОЗМОЖНОСТИ МИКРОСХЕМЫ SDR

ПРОШИВКА

В сети достаточно руководств по сборке приемников на SI4735, однако большинство авторов делают акцент на схемотехнику и сборку на макете, после чего туда заливают один из вариантов готовой прошивки. Мы же попробуем разобраться, как написать такую прошивку самостоятельно почти с нуля, поэтому все нижесказанное достаточно легко перенести на любой другой микроконтроллер, лишь бы у него хватало памяти для хранения патча.

Итак, что же за зверь SI4734 и с чем его едят? Этот чип управляется по шине I2C, и каждая посылка представляет собой адрес микросхемы (с битом переключения запись/чтение), 1 байт команды и до 7 байт аргументов. У каждой команды свое количество аргументов, впрочем, даташит говорит, что посылки можно сделать и фиксированной длины, если вместо неиспользуемых аргументов слать `0x00`. Для наших целей понадобится не так много команд, поэтому мы можем позволить себе написать для каждой свою функцию. Результатом выполнения команды можно считать ответ, состоящий из байта статуса и до 7 байт собственно ответа, причем и здесь допускается унификация длины: можно читать по 8 байт, все неиспользуемые будут `0x00`.

Но тут есть нюанс: команда выполняется не мгновенно, а с задержкой, до истечения которой микросхема будет отвечать только нулями. Поэтому, когда нам необходим ответ, мы с некоторой периодичностью будем его считывать, пока первый байт ответа не будет равен `0x80`, что свидетельствует о завершении исполнения команды. Следом можно считать байты ответа и/или отправлять следующую команду.

Для отправки и чтения пакетов по I2C мы будем использовать уже известную нам команду библиотеки LibopenCM3 `i2c_transfer7(SI4734I2C, SI4734ADR ...)`, где `SI4734I2C` — используемая шина I2C (I2C1), а `SI4734ADR` — семибитный адрес `SI4734 0x11`. О бите записи/чтения за нас позаботится библиотека. В итоге работа с микросхемой вкратце будет представлять собой следующую последовательность действий: инициализация, настройка режима работы, настройка на нужную частоту. Все описанное ниже опирается на содержание документов AN332 «Si47XX Programming Guide» и AN332SSB.

Инициализация

Прежде всего SI4734 нужно инициализировать. Сделать это можно в одном из трех режимов: AM, FM или SSB. Перед началом инициализации документация рекомендует выполнить сброс. Делается это тривиально: надо ненадолго подтянуть к земле REST-пин SI4734. Для задержки используется совершенно ленивая функция, благо точность тут не имеет особого значения.

```
#define SI4734D60_RSTPORT GPIOA
#define SI4734D60_RSTPIN GPIO7
#define SI4734_RST_CLR() gpio_clear(SI4734D60_RSTPORT,
SI4734D60_RSTPIN)
#define SI4734_RST_SET() gpio_set(SI4734D60_RSTPORT,
SI4734D60_RSTPIN)

void delay(uint16_t ms){
  uint64_t temp;
  temp=ms<<10;
  while(temp--)__asm__("nop");
}

void si4734_reset(){
  SI4734_RST_CLR();
  delay(10);
  SI4734_RST_SET();
  delay(10);
}
```

Для инициализации используется команда `POWER_UP 0x01`, которая требует два параметра. Первый включает тактирование и определяет режим работы, а второй настраивает аудиовыходы. Мы используем часовой кварц и аналоговые выходы, поэтому для FM применяются параметры `0x10`, `0x05`, а для AM — `0x11`, `0x05`. После отправки команды, опрашивая чип, дожидаемся ответа `0x80`. Обычно на это уходит один-два запроса.

```
#define SI4734I2C I2C1
#define SI4734ADR 0x11

uint8_t si4734_fm_mode(){
  // ARG1 (1<<4)|0 AN322 p130
  // ARG2 00000101
  uint8_t cmd[3]={POWER_UP,0x10,0x05};
  uint8_t status, tray=0;
  i2c_transfer7(SI4734I2C,SI4734ADR,cmd,3,0,0);
  delay(1000);
  do{ i2c_transfer7(SI4734I2C,SI4734ADR,0,0,&status,1);
    tray++;
    if(tray==255) return 0xff;
    delay(50);
  }while(status!=0x80);
  return status;
}
uint8_t si4734_am_mode(){
  // ARG1 (1<<4)|1 AN322 p130
  // ARG2 00000101
  uint8_t cmd[3]={POWER_UP,0x11,0x05};
  uint8_t status, tray=0;
  i2c_transfer7(SI4734I2C,SI4734ADR,cmd,3,0,0);
  delay(1000);
  do{ i2c_transfer7(SI4734I2C,SI4734ADR,0,0,&status,1);
    tray++;
    if(tray==255) return 0xff;
    delay(50);
  }while(status!=0x80);
  return status;
}
```

В ответ на команду `POWER_UP` чип может выдать еще 8 байт, которые даташит рекомендует проверять, однако на это можно забить и даже их не считывать. На данном этапе уже можно проверить качество работы микросхемы: исправная вернет ответ `0x80` и запустит кварцевый генератор, что проверяется осциллографом. Если команды отправлены верно, а генератор не запустился, то, вероятно, чип битый.

А что там с SSB? С однополосной модуляцией все хитрее, так как это не стандартная функция. Однако благодаря Вадиму Афонькину нам доступны патчи и мануал к ним, позволяющие принимать SSB и NBFM. Именно их и используют библиотеки, выложенные на GitHub для работы с SI4735, и мы также ими воспользуемся. Впрочем, мы позаимствуем этот патч из библиотеки Arduino, там он уже оформлен в виде массива C, что нам и нужно.

Для загрузки патча в память у SI4734 есть две команды: `0x15` и `0x16`. У каждой по 7 байт аргументов, `0x15` отмечает адрес в памяти SI4734, куда будет писаться патч, а `0x16` отвечает за загрузку самого патча по 7 байт за один заход. Само собой, после каждой такой команды надо дождаться ответа `0x80`. Патч представляет собой массив, содержащий упомянутые выше команды и необходимые аргументы, то есть от нас требуется читать из массива по 8 байт, отправлять в чип и ждать, пока он их переварит. Учитывая изрядный размер патча, на это уходит около секунды. В целом весь процесс напоминает описанную выше процедуру инициализации AM, только в первом аргументе выставлен бит, отвечающий за прием патча: `RST POWER_UP 0x31, 0x05`. А дальше заливаем патч.

```
const uint8_t ssb_patch_content[] =
  {0x15, 0x00, 0x03, 0x74, 0x0B, 0xD4, 0x84, 0x60,
  ...
  0x15, 0x00, 0x00, 0x00, 0x00, 0x00, 0xA9, 0x02};// 0x451 строк

uint8_t si4734_ssb_patch_mode(uint8_t *patch){
  // ARG1 (1<<5)|(1<<4)|1 AN322SSB p7
  // ARG2 00000101
  uint8_t cmd[3]={POWER_UP,0x31,0x05};
  uint8_t status, tray=0;
  uint16_t count,iterate=0;
  i2c_transfer7(SI4734I2C,SI4734ADR,cmd,3,0,0);
  delay(1000);
  do{
    i2c_transfer7(SI4734I2C,SI4734ADR,0,0,&status,1);
    tray++;
    if(tray==255) return 0xff;
    delay(50);
  }while(status!=0x80);
  if(status!=0x80) return 0x1;
  count=0x451;
  while(count--){
    tray=0;
    i2c_transfer7(SI4734I2C,SI4734ADR,patch+iterate,8,0,0);
    iterate+=8;
    delay(2);
    do{
      i2c_transfer7(SI4734I2C,SI4734ADR,0,0,&status,1);
      tray++;
      if(tray==255) return 0x02;
      delay(1);
    }while(status!=0x80);
  }
  return status;
}

si4734_ssb_patch_mode(ssb_patch_content);
```

INFO

Если у тебя есть одна из реализаций подобного приемника, можно посмотреть, какой патч она использует, слушая шину I2C и вылавливая посылки, начинающиеся на `0x11`, `0x15` и `0x11`, `0x16`. Это несложно сделать логическим анализатором.

Сразу после заливки патча раздается характерный «хрюк» из динамиков. Окирпичить чип некорректным патчем, судя по всему, невозможно, так как при сбросе восстанавливается заводская прошивка, а некорректный патч просто не заработает. Во всяком случае, те экземпляры SI4730, которые тестировал я, после ресета работали штатно.

Настройка режима работы

После инициализации устанавливаются прошитые на заводе дефолтные параметры. Если хочется побыстрее запустить эту штуку, то этот шаг можно пропустить. Особенно учитывая, что настраиваемых параметров у SI4734 несколько десятков и для каждого режима они свои, а на заводских настройках чип уже будет вполне себе работать. Однако некоторые параметры имеет смысл подкрутить, особенно это касается SSB.

Начнем с простого: с FM. В SI4734 по умолчанию выставлен режим коррекции предыскажений по американскому стандарту, его стоит поменять на европейский. Для этого нам понадобится команда `SET_PROPERTY 0x12`, у нее пять аргументов. Первый — `0x00`, затем два байта номера параметра (старший и младший) и два байта — значение, которое будет записано (старший и младший).

```
uint8_t si4734_set_prop(uint16_t prop, uint16_t val){
  uint8_t cmd[6]={SET_PROPERTY,0,(prop>>8),
                (prop&0xff),(val>>8),(val&0xff)};
  uint8_t status;
  i2c_transfer7(SI4734I2C,SI4734ADR,cmd,6,0,0);
  delay(100);
  i2c_transfer7(SI4734I2C,SI4734ADR,0,0,&status,1);
  return status;
}
```

Здесь регулярный опрос чипа в ожидании ответа заменен фиксированной задержкой, но ее надо подобрать. Теперь с помощью этой функции поменяем интересующий параметр, а заодно пропишем заданную громкость.

```
#define FM_DEEMPHASIS 0x1100
#define RX_VOLUME    0x4000
si4734_set_prop(FM_DEEMPHASIS,0x0001);// 01 = 50 µs. Used in Europe,
Australia, Japan
si4734_set_prop(RX_VOLUME, vol);
```

С AM параметров побольше: расширим полосу пропускания до 6 кГц, без этого вещательные станции звучат непривычно глухо, а ведь там и музыку крутят. Заодно включим подавление помехи 100 Гц по питанию. Не знаю, насколько это эффективно, но хуже не становится. За эти два параметра отвечает команда `AM_CHANNEL_FILTER 0x3102`. На этом можно и остановиться, однако, чтобы лучше слышать слабые станции, стоит отключить шумодав и добавить усиления в тракт НЧ, ну и выставить громкость. В общем виде настройка выглядит так:

```
#define AM_CHANNEL_FILTER 0x3102
#define AM_SOFT_MUTE_MAX_ATTENUATION 0x3302
#define AM_AUTOMATIC_VOLUME_CONTROL_MAX_GAIN 0x3103

si4734_set_prop(AM_CHANNEL_FILTER, 0x0100);
si4734_set_prop(AM_SOFT_MUTE_MAX_ATTENUATION, 0);// soft mute off
si4734_set_prop(AM_AUTOMATIC_VOLUME_CONTROL_MAX_GAIN, 0x5000); // 60
дБ
si4734_set_prop(RX_VOLUME, vol);
```

И наконец, SSB. Тут покрутить придется побольше: надо настроить режим работы параметром `SSB_MODE 0x0101` и выставить полосу пропускания 3 кГц (ориентируемся на телефонные сигналы). Выключаем автоподстройку частоты и включаем автонастройку громкости. Затем добавляем усиления в аудиотракт и выключаем шумодав. Коды параметров тут те же, что и в AM. Подобным же образом выставляем громкость.

```
#define SSB_MODE 0x0101
si4734_set_prop(SSB_MODE,((1<<15)|(1<<12)|(1<<4)|2));// ssb man page
24
si4734_set_prop(AM_SOFT_MUTE_MAX_ATTENUATION, 0);// soft mute off
si4734_set_prop(AM_AUTOMATIC_VOLUME_CONTROL_MAX_GAIN, 0x7000); // 84
дБ
si4734_set_prop(RX_VOLUME, vol);
```

Вот и вся предварительная настройка, не так страшно, в общем-то. Осталось задать частоту.

Настройка частоты

Настройка частоты в разных режимах работы немного отличается: проще всего в FM и AM, несколько хитрее в SSB. Начнем с FM. Для настройки на заданную частоту существует специальная команда `FM_TUNE_FREQ 0x20`, у которой четыре параметра. Первый отвечает за скорость настройки в ущерб точности, зададим его как `0x00`. В FM это не очень принципиально. Два следующих — старший и младший байты частоты в десятках килогерц, а последний — это подстройка входной емкости, его рекомендуют выставлять в `0x00` (автонастройка), если прием ведется на короткий штырь. Так и поступим. В общем, все так же, как и раньше: шлем команду с параметрами, ждем ответа `0x80`.

```
uint8_t si4734_fm_set_freq(uint16_t freq_10khz){
  uint8_t fast,freq_h,freq_l,status,tray=0;
  fast=0;
  freq_h=freq_10khz>>8;
  freq_l=freq_10khz&0xff;
  uint8_t cmd[6]={FM_TUNE_FREQ,fast,freq_h,freq_l,0,0};
  i2c_transfer7(SI4734I2C,SI4734ADR,cmd,6,0,0);
  delay(20);
  //do{
  //  status=si4734_get_int_status();
  //  delay(50);
  //} while(!status || status&1);
  do{
    i2c_transfer7(SI4734I2C,SI4734ADR,0,0,&status,1);
    tray++;
    if(tray==255) return 0xff;
    delay(20);
  }while(status!=0x80);
  return status;
}
```

Доступные значения частоты от 6300 до 10 900 чуть шире даташита, дальше чип выдает ошибку. С AM все почти так же, только код команды другой — `AM_TUNE_FREQ 0x40` и параметров уже пять. Первый — ускоренная настройка в ущерб точности (`0x00` и `0x01`, выкл/вкл соответственно). Далее два байта частоты в килогерцах, старший и младший, а за ними два байта настройки входной емкости (`0x00`, автоподстройка).

Автоподстройка хороша на длинных/средних волнах, когда ко входу AMI подключена магнитная антенна. Это позволяет настроить ее в резонанс. Но так как мы используем внешнюю полноразмерную антенну, а на входе у нас стоит еще и ФНЧ, мы отключим эту подстройку, выставив минимальное значение двух байт `0x00 0x01`. Вообще, интересно было бы поставить на вход резонансную цепь и покрутить эти параметры, но это усложнило бы конструкцию и вряд ли дало бы существенный выигрыш.

Что же касается первого параметра, то вопрос дискуссионный — включать или нет ускоренную настройку? С одной стороны, на AM настройка нужна поточнее, с другой — у этих микросхем есть неприятное свойство, которое в интернете назвали «чух-чух», — характерные звуки при перестройке частоты. Это довольно сильно раздражает, особенно с непривычки. Так вот, включение ускоренной настройки чуть уменьшает этот эффект, а недостаток точности нам компенсирует автоподстройка частоты, включенная по умолчанию. Так что тут можешь сам попробовать и выбрать, что больше понравится. В итоге имеем такую функцию.

```
uint8_t si4734_am_set_freq(uint16_t freq_khz){
  uint8_t fast,freq_h,freq_l,status,tray=0;
  fast=1;
  freq_h=freq_khz>>8;
  freq_l=freq_khz&0xff;
  uint8_t cmd[6]={AM_TUNE_FREQ,fast,freq_h,freq_l,0,1};
  i2c_transfer7(SI4734I2C,SI4734ADR,cmd,6,0,0);
  delay(20);
  do{
    i2c_transfer7(SI4734I2C,SI4734ADR,0,0,&status,1);
    tray++;
    if(tray==255) return 0xff;
    delay(20);
  }while(status!=0x80);
  return status;
}
```

Теперь SSB. Команда та же, что для AM, и параметров столько же, только теперь первый параметр отвечает за то, какую боковую полосу мы будем слушать — LSB (`0b01000000`) или USB (`0b10000000`). Существует соглашение, согласно которому на частотах до 10 МГц используют LSB, а выше — USB. Из этого правила имеются исключения, но это детали, поэтому мы сделаем привязку первого аргумента к частоте. Остальные четыре параметра те же, что и в AM. В итоге получаем вот такую функцию.

```
uint8_t si4734_ssb_set_freq(uint16_t freq_khz){
  uint8_t mode,freq_h,freq_l,status,tray=0;
  if(freq_khz>10000)mode=0b10000000;else mode=0b01000000;
  freq_h=freq_khz>>8;
  freq_l=freq_khz&0xff;
  uint8_t cmd[6]={AM_TUNE_FREQ,mode,freq_h,freq_l,0,1};
  i2c_transfer7(SI4734I2C,SI4734ADR,cmd,6,0,0);
  delay(20);
  do{
    i2c_transfer7(SI4734I2C,SI4734ADR,0,0,&status,1);
    tray++;
    if(tray==255) return 0xff;
    delay(20);
  }while(status!=0x80);
  return status;
}
```

Искушенный читатель заметит, что точность настройки 1 кГц для SSB недостаточна, так как от величины расстройки зависит тон сигнала и всего несколько сотен герц могут сделать голос неразборчивым. А кварц у нас неидеальный, да и на передающей стороне такое тоже может иметь место.

Поэтому нужно предусмотреть возможность точно подстроить частоту. Для этого есть параметр `SSB_BFO 0x0100`, принимающий значения от –16 383 до +16 383 и означающий отстройку частоты в герцах от той, что задана командой `AM_TUNE_FREQ`. Даташит говорит, что истинная частота настройки будет `AM_TUNE_FREQ (кГц) + SSB_BFO (Гц)`, однако мои эксперименты показали, что для SI4734 BFO нужно вычитать из основной частоты, то есть `AM_TUNE_FREQ (кГц) - SSB_BFO (Гц)`. То ли это особенность именно SI4734, то ли ошибка в даташите, впрочем, это не так важно. В итоге BFO мы задаем командой `si4734_set_prop(SSB_BFO, bfo)`, где `bfo` представляет собой желаемую отстройку в герцах (16-битное целое число со знаком).

Собираем воедино

Теперь у нас есть все минимально необходимые функции для запуска SI4734 в любом из трех режимов. Соберем все в одну функцию, принимающую на вход желаемый режим. А заодно добавим функцию `si4734_powerdown()`, которая позволит не только стартовать из выключенного состояния, но и переключать уже установленные режимы. Как понятно из названия, команда программно выключает SI4734.

```
#define AM_MODE 0
#define FM_MODE 1
#define SSB_MODE 2
uint16_t encoder=15200;
uint16_t pwm1=750;
uint16_t coef=5;
int16_t bfo=0;
int16_t vol=0x1a;

uint8_t si4734_powerdown(){
  uint8_t cmd=POWER_DOWN,status;
  i2c_transfer7(SI4734I2C,SI4734ADR,&cmd,1,0,0);
  delay(200);
  i2c_transfer7(SI4734I2C,SI4734ADR,0,0,&status,1);
  return status;
}

void reciver_set_mode(uint8_t rec_mod){
  static uint16_t amfreq=15200,fmfreq=8910; // Запоминаем старое
значение
  si4734_powerdown(); // частоты
  if(reciver_mode==FM_MODE)fmfreq=encoder; else amfreq=encoder;
  if(rec_mod==AM_MODE){
    reciver_mode=AM_MODE;
    si4734_am_mode();
    si4734_set_prop(AM_CHANNEL_FILTER, 0x0100);
    si4734_set_prop(AM_SOFT_MUTE_MAX_ATTENUATION, 0); // soft mute
off
    si4734_set_prop(AM_AUTOMATIC_VOLUME_CONTROL_MAX_GAIN, 0x5000);
// 60 дБ
    si4734_set_prop(RX_VOLUME, vol);
    encoder=amfreq-bfo/1000; // Поправка на BFO
    si4734_am_set_freq(encoder);
    coef=1;
    encoder_mode=0;
  }else if(rec_mod==FM_MODE){
    reciver_mode=FM_MODE;
    si4734_fm_mode();
    si4734_set_prop(FM_DEEMPHASIS,0x0001); // 01 = 50 µs. Used in
Europe, Australia, Japan
    si4734_set_prop(RX_VOLUME, vol);
    MIN_LIMIT=6000;
    MAX_LIMIT=11100;
    coef=10;
    encoder=fmfreq;
    si4734_fm_set_freq(encoder);
    encoder_mode=0;
  }else{
    reciver_mode=SSB_MODE;
    si4734_ssb_patch_mode(ssb_patch_content);
    si4734_set_prop(0x0101,((1<<15)|(1<<12)|(1<<4)|2)); // ssb man
page 24
    si4734_set_prop(AM_SOFT_MUTE_MAX_ATTENUATION, 0); // soft mute
off
    si4734_set_prop(AM_AUTOMATIC_VOLUME_CONTROL_MAX_GAIN, 0x7000);
// 84 дБ
    si4734_set_prop(RX_VOLUME, vol);
    encoder=amfreq;
    si4734_ssb_set_freq(encoder);
    coef=1;
    encoder_mode=0;
  }
}
```

Таким образом, для запуска чипа, например, в режиме FM на частоте `freq` достаточно вызвать три функции:

```
si4734_reset();
for(uint32_t i=0;i<0x5ff;i++)__asm__("nop"); // Задержка, требуемая
даташитом
reciver_set_mode(FM_MODE);
si4734_fm_set_freq(freq);
```

Продолжение статьи →

СЕКРЕТЫ SI473X

ДЕЛАЕМ ПРИЕМНИК И ИЩЕМ СКРЫТЫЕ ВОЗМОЖНОСТИ МИКРОСХЕМЫ SDR

Чтобы переключиться в другой режим, достаточно вызвать две последние, так как сброс рекомендуют делать только при включении. Для перестройки же частоты и вовсе достаточно последней функции. Наш приемник уже работоспособен, осталось налепить красивостей и функций.

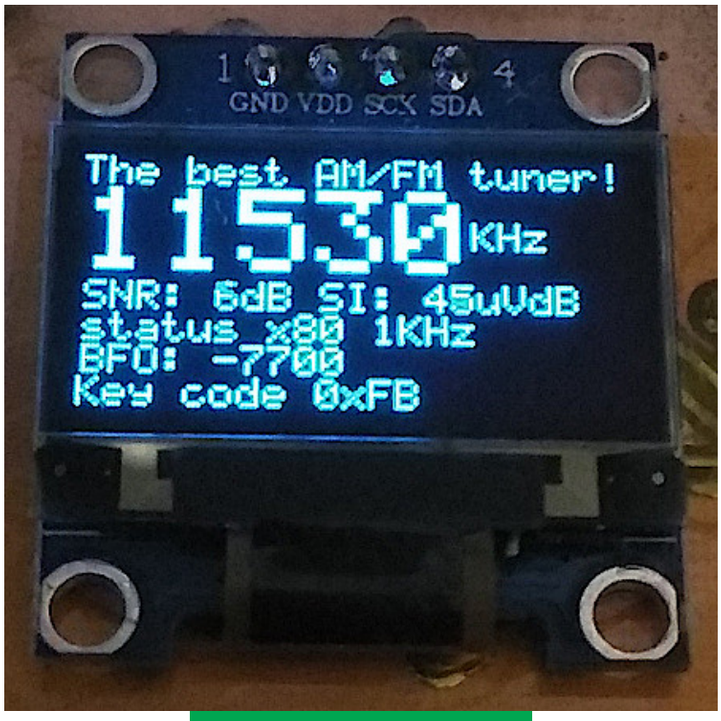
Добавим красивости и функции

Для вывода я взял самописную библиотеку, которую использовал в предыдущих проектах. Она умеет выводить форматные строки заданного размера в заданной точке дисплея, и этого текстового интерфейса нам хватит за глаза. Из настроенного чипа можно узнать некоторые параметры сигнала — текущую частоту настройки, интенсивность сигнала, соотношение сигнал/шум и другие. Но поскольку мы используем прямую установку частоты, то особой необходимости выспрашивать частоту у чипа нет, а вот интенсивность и зашумленность полезно видеть. Кроме того, в режиме SSB есть еще BFO, его тоже неплохо бы отобразить. Причем в случае последнего в SSB-режиме мы сделаем поправку отображаемой частоты. Итак, функция индикации будет выглядеть следующим образом.

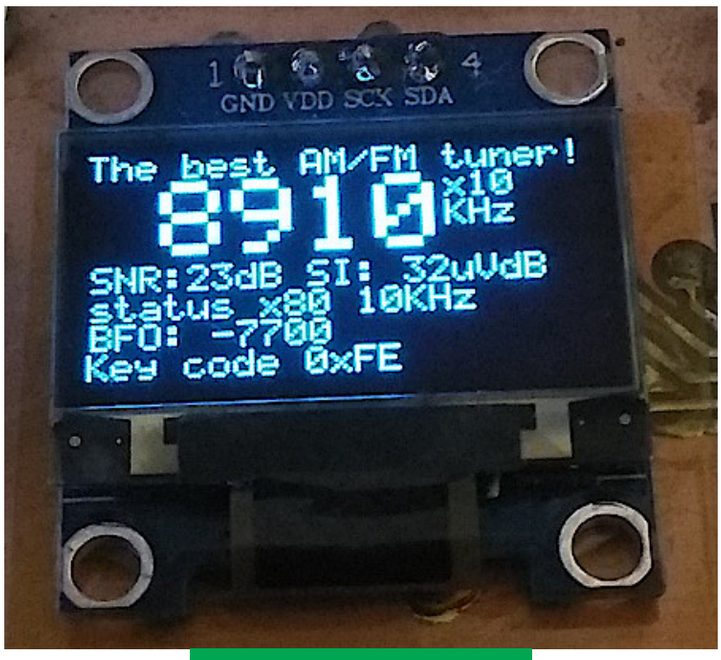
```
void show_freq(uint16_t freq, int16_t offset){
  uint16_t offset_hz;
  uint16_t freq_khz;
  if(reciver_mode==FM_MODE)o_printf_at(18*5,1,1,0,"x10");
    else o_printf_at(18*5,1,1,0,"   ");
  // f=f-bfo
```

```
  if(reciver_mode==SSB_MODE){
    offset_hz=(1000-offset%1000)%1000;
    freq_khz=freq-offset/1000;
    if(offset%1000>0)freq_khz--;
    o_printf_at(18*5,3,1,0,"%03d",offset_hz);
    o_printf_at(0,1,3,0,"%5d",freq_khz);
  }else{
    o_printf_at(18*5,3,1,0,"   ");
    o_printf_at(0,1,3,0,"%5d",freq);
  }
  o_printf_at(18*5,2,1,0,"KHz");
}

void show_reciver_status(uint8_t snr, uint8_t rssi, uint8_t status){
  uint8_t n=1;
  o_printf_at(1,4,1,0,"SNR:%2ddB SI: %2duVdB",snr,rssi);
  // coef – глобальная переменная
  // n – поправочный коэффициент шага
  if(reciver_mode==FM_MODE)n=10;
  o_printf_at(1,5,1,0,"status x%x %dKHz   ",status,coef*n);
}

void show_reciver_full_status(uint16_t freq, int16_t offset,
                              uint8_t snr, uint8_t rssi, uint8_t
status){
  show_freq(freq, offset);
  show_reciver_status(snr,rssi,status);
}
```

Здесь **reciver_mode** — глобальная переменная, содержащая код текущего режима, от которого немного меняется интерфейс, **freq** — текущая частота настройки, **offset** — отстройка частоты в SSB-режиме, **snr** — соотношение сигнал/шум в децибелах, **rssi** — интенсивность сигнала в децибел-микровольтах, **status** — статус-байт, возвращенный командой установки частоты. Последний помогает при отладке. Глобальная переменная **coef** отражает шаг перестройки частоты энкодером, причем в случае AM/SSB это килогерцы, а для FM — десятки килогерц. Целочисленная математика — это поправка на отстройку частоты. А выглядит это все вот так.



AM-интерфейс



FM-интерфейс

Узнать параметры **rssi** и **snr** мы можем с помощью команды **AM_RSQ_STATUS 0x43** или **FM_RSQ_STATUS 0x23**, причем команды для SSB и AM идентичны:

```
uint8_t si4734_am_signal_status(uint8_t *resp1,uint8_t *resp2,uint8_t
*rssi,uint8_t *snr){
  uint8_t cmd[3]={AM_RSQ_STATUS,0x1};
  uint8_t tray=0;
  uint8_t answer[6];
  i2c_transfer7(SI4734I2C,SI4734ADR,cmd,2,0,0);
  delay(50);
  answer[0]=0;
  while(answer[0]==0){
    i2c_transfer7(SI4734I2C,SI4734ADR,0,0,answer,6);
    tray++;
    if(tray==255) return 0xff;
    delay(50);
  }
  *resp1=answer[1];
  *resp2=answer[2];
  *rssi=answer[4];
  *snr=answer[5];
  return answer[0];
}

uint8_t si4734_fm_signal_status(uint8_t *rssi,uint8_t *snr,int8_t *
freq_of){
  uint8_t cmd[3]={FM_RSQ_STATUS,0x1};
  uint8_t tray=0;
  uint8_t answer[8];
  i2c_transfer7(SI4734I2C,SI4734ADR,cmd,2,0,0);
  delay(50);
  answer[0]=0;
  while(answer[0]==0){
    i2c_transfer7(SI4734I2C,SI4734ADR,0,0,answer,8);
    tray++;
    if(tray==255) return 0xff;
    delay(50);
  }
  *rssi=answer[4];
  *snr=answer[5];
  *freq_of=answer[7];
  return answer[0];
}
```

```
uint8_t get_recivier_signal_status(uint8_t *snr,uint8_t *rssi,uint8_t
*freq_of){
  uint8_t status,resp1,resp2;
  switch(reciver_mode){
    case AM_MODE: status=si4734_am_signal_status(&resp1,&resp2,rssi,
snr);
      break;
    case FM_MODE: status=si4734_fm_signal_status(rssi,snr,freq_of);
      break;
    case SSB_MODE: status=si4734_am_signal_status(&resp1,&resp2,rssi,
snr);
      break;
  }
  return status;
}
```

Теперь вернемся к проблеме перестройки. Шаг перестройки мы вольны задавать сами, коли уж мы можем задавать частоту с точностью до килогерца в AM/SSB и до 10 кГц в FM, на практике удобны значения 1, 5 и 10 кГц, а для FM их удобно умножать на 10.

В случае же с SSB можно использовать небольшой хак и почти решить проблему «чух-чух». Для этого мы используем отстройку частоты, которая не дает щелчков. К несчастью, ее величина ограничивается +/–16 кГц, но если мы будем следить за BFO и по достижении +/–16 000 сбрасывать BFO, а затем поправлять основную настройку, то мы получим возможность непрерывной настройки с точностью около 10 Гц. Заявленная в даташите точность отстройки — 8 Гц, впрочем, это не так важно, за такие деньги лишь бы не плавала. На практике удобно взять шаг 100 Гц. С этим шагом все еще можно достаточно точно настроиться и при этом не заколебаться крутить ручку. Кроме того, возможность настройки основной частоты с произвольным шагом никто не отменял. А выглядит этот хак вот так:

```
if(old_bfo!=bfo && reciver_mode==2){
  si4734_set_prop(SSB_BFO, bfo);
  o_printf_at(1,6,1,0,"BFO: %d  ",-bfo);
  if(bfo>16000||bfo<-16000){
    encoder=encoder-bfo/1000;
    bfo=bfo%1000;
```

```
  }
  old_bfo=bfo;
}
if(old_encoder!=encoder){
  old_encoder=encoder;
  if(reciver_mode==1) status=si4734_fm_set_freq(encoder);
  else if(reciver_mode==2) status=si4734_ssb_set_freq(encoder);
  else status=si4734_am_set_freq(encoder);
}
```

И последняя штука, о которой стоит упомянуть, — это автопоиск станций. Очень удобная функция, когда лень крутить ручку. Я его реализовал только для АМ, так как на УКВ у меня станции идут подряд с шагом примерно 400 кГц — чего там, спрашивается, искать?

Поиск возможен в двух направлениях — вверх и вниз соответственно, а из параметров он требует задания шага поиска, причем поддерживаются только фиксированные значения 1, 5 и 10 кГц. 1 кГц надежнее и медленнее, 5 может проскочить, но быстрее. Я остановился на 5 кГц. Далее нужно выбрать критерий поиска. Можно искать по RSSI и по SNR. Учитывая, что у меня очень зашумленный эфир, искать по интенсивности — пустое занятие, поэтому ищем по SNR и выставляем порог в минимум, то есть в 1 дБ. В любом случае, если станция не дает и 1 дБ SNR, ее особо не послушаешь.

Далее задаем границы поиска. Текущая частота должна быть между верхней и нижней границами поиска, иначе фокус не сработает. В тех примерах, что я видел, в памяти хранились все поддиапазоны КВ и в них велся поиск. Однако это, на мой вкус, неудобно, так как, во-первых, вещалки бывают и за пределами поддиапазонов (китайцы не очень блюдут стандарты), а во-вторых, поиск по всему поддиапазону может быть достаточно долгим. Поэтому каждый раз при вызове поиска я буду задавать границы как +/–200 кГц от текущей частоты. Осталось задать поведение при неудачном поиске — остановиться на соответствующей границе.

После окончания поиска надо узнать частоту, на которую настроен чип, и поправить значение частоты в управляющем контроллере, чтобы она правильно отображалась. В этот раз мы будем ждать от чипа ответ **0x81**, сигнализирующий об окончании поиска. Строго говоря, **0x81** — это **0x80|0x01**, где **0x80** сигнализирует о готовности дальше принимать команды, а **0x01** говорит, что настройка частоты завершена. В рассмотренных выше командах перестройки частоты, если дальше продолжить опрос статуса чипа после приема **0x80**, через некоторое время чип будет выдавать **0x81**. В коде все описанное выглядит так.

```
...
```

```
// Эта часть привязана к кнопке
if(reciver_mode==AM_MODE){
  si4734_am_seek(encoder,1); // Поиск вверх
  si4734_get_freq_v2(&encoder);
}
...
// Эта часть привязана к кнопке
if(reciver_mode==AM_MODE){
  si4734_am_seek(encoder,0); // Поиск вниз
  si4734_get_freq_v2(&encoder);
}
...
void si4734_am_seek(uint16_t freq, uint8_t up){
  uint8_t cmd[6]={AM_SEEK_START,(1<<3),0,0,0,1}; // AN332 p138
  uint16_t top,bottom;
  // Чтобы поиск надолго не вешал приемник, ограничим диапазон до
400 кГц
  top=freq+200;
  if(top>30000)top=30000;
  if(freq<400) bottom=200;else bottom=freq-200;
  si4734_set_prop(AM_SEEK_BAND_TOP,top);
  si4734_set_prop(AM_SEEK_BAND_BOTTOM,bottom);
  si4734_set_prop(AM_SEEK_FREQ_SPACING,5); // Шаг поиска
  si4734_set_prop(AM_SEEK_SNR_THRESHOLD,1); // Порог 1 дБ
  // uint16_t freq;
  // uint8_t rssi,snr;
  if(!up) cmd[1]&=~(1<<3);
  i2c_transfer7(SI4734I2C,SI4734ADR,cmd,6,0,0);
  // delay(500);
  while (si4734_get_int_status()!=0x81)delay(100);
}

uint8_t si4734_get_freq_v2(uint16_t *freq){ // Возвращает только
частоту
  uint8_t cmd[2]={AM_TUNE_STATUS,0x0};
  uint8_t tray=0;
  uint8_t answer[8];
  i2c_transfer7(SI4734I2C,SI4734ADR,cmd,2,0,0);
  delay(50);
  answer[0]=0;
  while(answer[0]==0){ // Если все нормально, ответ будет 0x81
    i2c_transfer7(SI4734I2C,SI4734ADR,0,0,answer,8);
```

```
    tray++;
    if(tray==255) return 0xff;
    delay(50);
  }
  *freq=((answer[2]<<8)|answer[3]);
  return answer[0];
}
```

Любопытно: если во время поиска выспрашивать у «сишки» текущую частоту, то можно увидеть, как она шагает в заданном интервале, пока не найдет станцию или не дойдет до границы поиска. Прочие подробности реализации можно посмотреть на GitHub, там весь код, схемы, платы и полезные даташиты.

ВПЕЧАТЛЕНИЯ

Впечатления от SI4734 у меня остались положительные, и по большому счету претензия только одна — «чух-чух» при настройке AM/SSB. Аналоговый приемник с двойным преобразованием настраивать гораздо приятнее, однако если принять во внимание простоту конструкции и очень демократичную цену «сишки», то ей можно простить этот недостаток.

При этом FM работает просто превосходно, и тут никаких нареканий. С чувствительностью у него тоже все в порядке, в режиме SSB отчетливо слышен сигнал тестового генератора при амплитуде около 0,5 мкВ. Поэтому если включить его вдали от источников помех, например за городом или хотя бы в парке, то на AM/SSB-диапазонах всегда будет что послушать. Но и в городе все не так плохо, правда, тут результат уже целиком зависит от антенны.

Но самое забавное — этот приемник в диапазоне CB (27 МГц AM) работает лучше, чем RTL-SDR-v3. Вероятно, это связано с внутренними помехами RTL-SDR-v3, в остальном последний лучше. Однако если хочется послушать эфир на природе, а ноут тащить лень, то SI4734 — то, что надо!