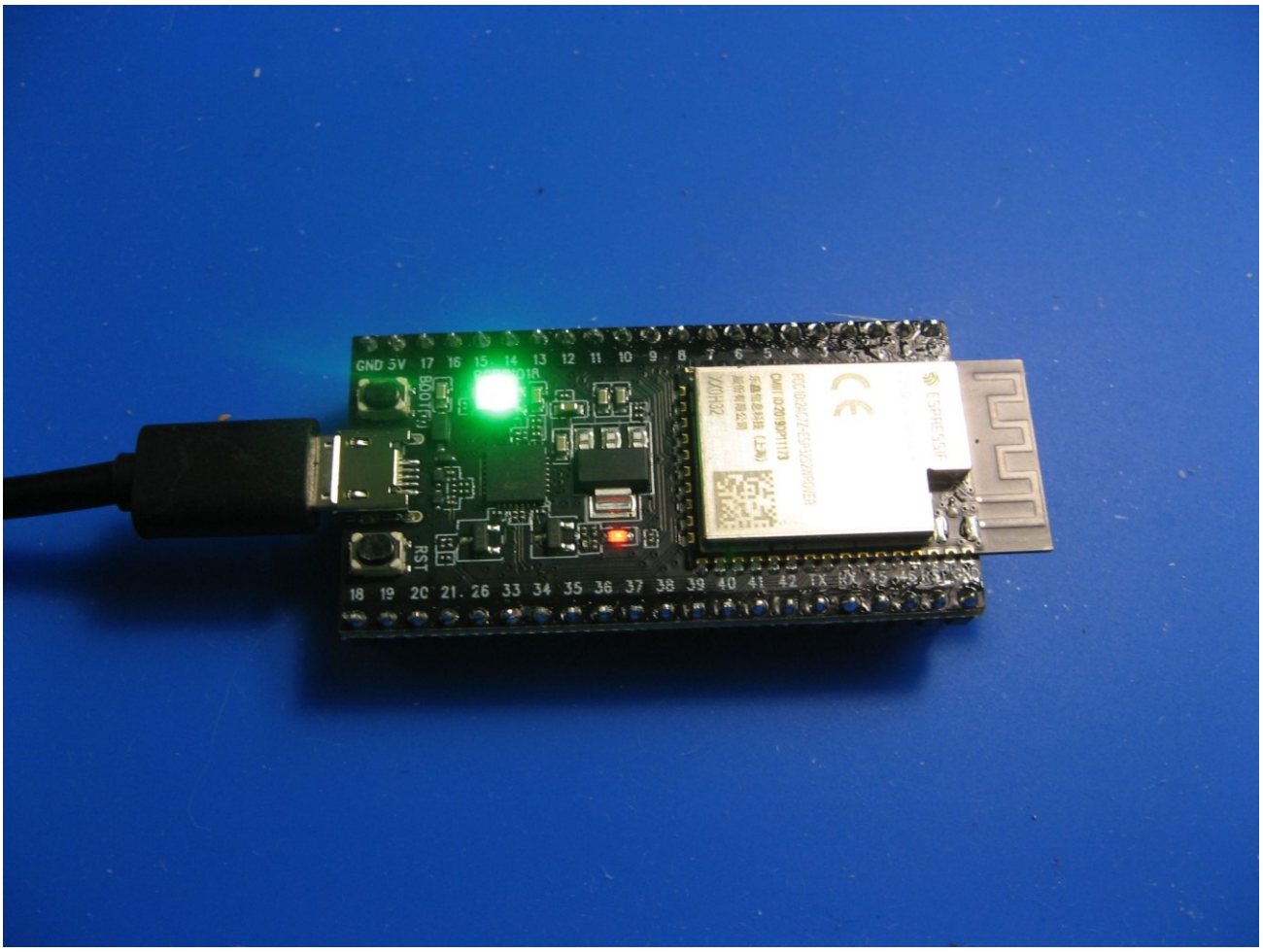
Супер-чип от Espressif: ESP32-S2

Николай Варанкин, заметки на LinkedIn, год 2021-й

Часть 1: Встреча

Сегодня прибыл новый процессор из Китая. Это точно не GeForce, но вот и посмотрим, на что он способен. Кстати, сам чип на Mouser Electronics продается всего за $1.61. 😊 То есть на эквивалентной системе можно впаять 300 чипов. По числу ядер это раз в восемь меньше, чем у моего GeForce.

Возьму короткую паузу, но продолжение последует обязательно... (Главное, не спалить чип в порыве фантазий. 😊



Вы точно согласитесь, что самый волнительный момент – это первое включение. К тому же он у меня – первый. Особые чувства. Вот и я замер на мгновение, сделал вдох-выдох, и воткнул кабель. Зажегся диодик питания, короткая пауза, и начал моргать «трехцветник». Это значит, что работает программа. Облегченный выдох и море радости! Модуль живой, процессор-память-коммуникация есть. Можно начинать работу.

Кстати, для чипа заявлена поддержка памяти до 1 ГБ SPI flash и до 1 ГБ SPI PSRAM. Это подразумеваются внешние микросхемы по порту SPI2. Как это осуществимо, для меня пока непонятно, но старшие 2,5 ГБ адресного пространства совершенно свободны.

Лимиты в 1 ГБ — это технические лимиты Memory Management Unit (#MMU). Окно адресного доступа к этой памяти через #ICache и #DCache примерно 10 МБ. И эта супер-пупер память сидит на одном SPI0, то есть в один момент времени доступна либо PSRAM, либо FLASH. Старшие 2,5 ГБ адресов на самом деле ждут лучших времен.

Чип конкретно этого модуля скорее всего не так дешев, как я написал выше. По догадке, а вскрытие модуля сейчас нежелательно, там установлен ESP32-S2FN4R2 (4 МБ flash RAM + 2 МБ PSRAM). И это чудо стоит на Mouser Electronics уже $3.33. Чуток дешевле, чем модуль WROVER.

Часть 2: Документация

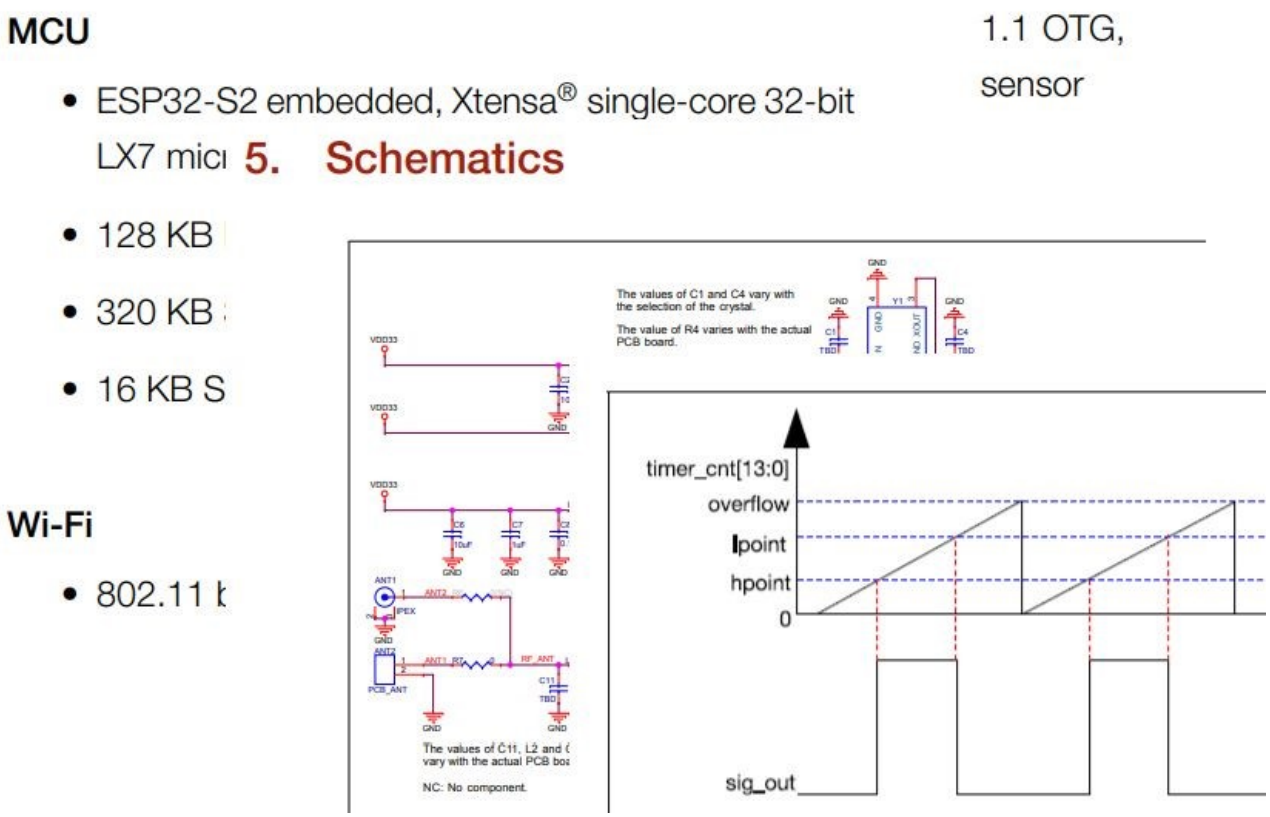
Первое впечатление от #IoT - это гораздо больший объем документации. На плату размером с кредитку главный документ на чип содержит 900+ страниц. Раньше я считал, что книга для программистов на 500 страниц — это для dummies, зачем лить столько воды! Документ на чип достаточно краткий, скорее конспективный. По моему мнению, текста надо в два раза больше. Минимум. И большое спасибо, что с гиперссылками, и что уже на английском, а не на китайском. И еще предстоит знакомство с API от производителя, но это другой документ и будет попозже.

Второе впечатление — это ощущение триумфа цифровых технологий. Чтобы посадить второе дерево, надо снова добыть саженец, выкопать яму, вставить саженец, засыпать землей и полить. То есть в точности повторить всю работу для первого. А чтобы добавить второй канал I/O, не надо повторять месяцы разработки. Простой copy-paste сделает основную "работу", затем изменить базовый адрес, добавить контакты - и voilà, дело сделано (почти)! Не сравнимо совсем с лесо-посадкой.

"Technical Reference Manual" на 849 страниц прочтен! Но остались вопросы.

- Ни слова про Wi-Fi модуль. Наверное, всё рассказано в другом документе.
- Бравурно заявлено о поддержке флэш и памяти на 1 ГБ каждая, но как это реализуется, тоже ни слова.

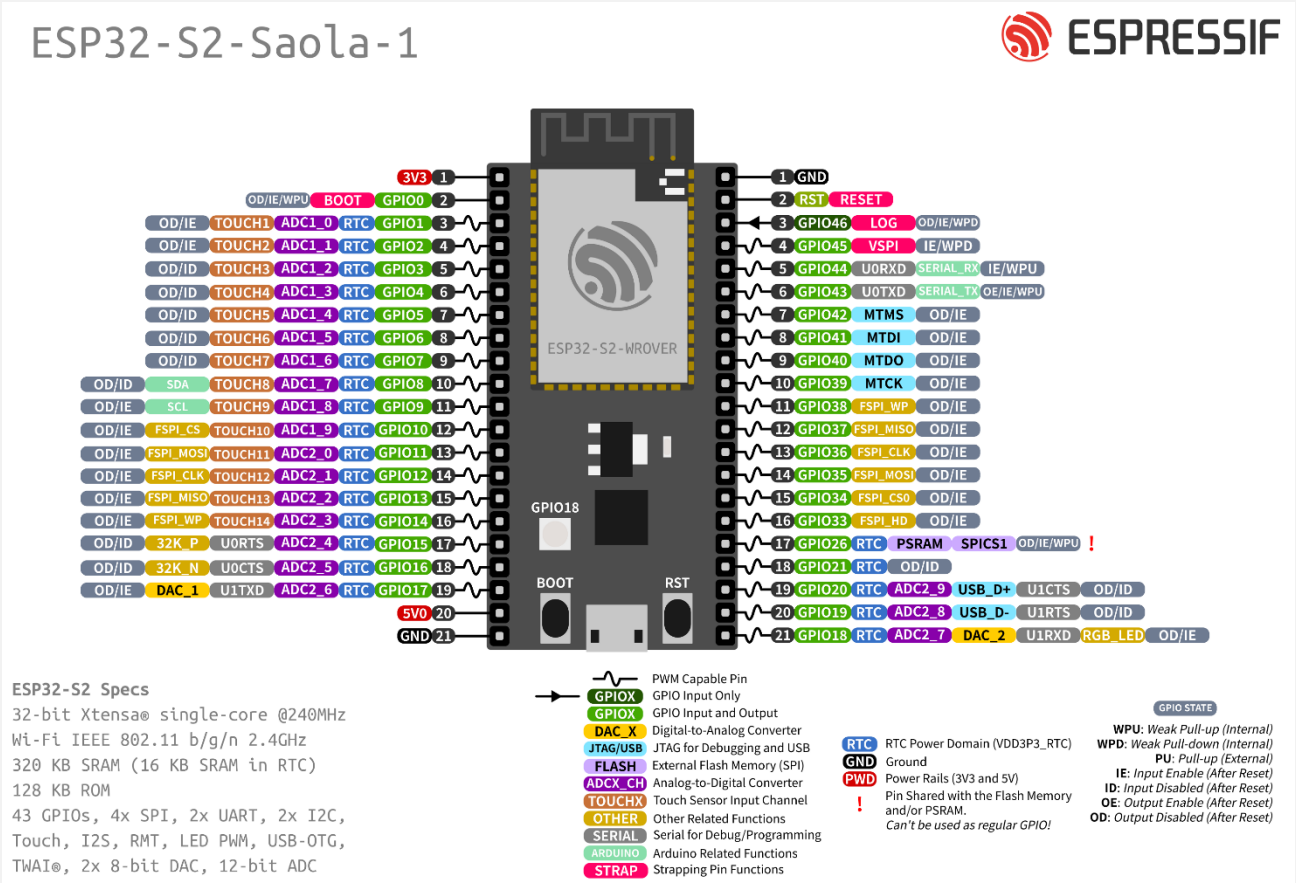
Хотя это - скорее реклама, так как столько памяти в реальных чипах потянет тока от 3А по питанию и это уже не #IoT на батарейке. А у производителей #PSRAM максимум объема памяти мелькает как 128 (16M*8) МБит на чип. Даже если поставить на SPI2 с 5-ю линиями, это 80МБ максимум. Популизм. Но для будущего это огромный плюс. Достаточно вспомнить стоны и плачи при переходе архитектур с 8 бит на 16, затем на 32, потом на 64. И, в конце концов, на горизонте 128 вроде многие активисты уже угомонились. 😊



"Technical Reference Manual", похоже, еще не закончен! :) Внимательное рассмотрение списка ревизий позволяет заключить, что документ "неспешно" пишется с момента начала продажи чипов. Ждем продолжения. Кстати, в этом документе описаны только собственные компоненты Espressif Systems. На компоненты сторонних производителей (CPU Xtensa, Wi-Fi) есть только примеры использования высокоуровневого API. Не густо.

Всё познаётся в сравнении. Описание SDK в PDF-варианте тянет на 1500 страниц. Как славно, когда не надо тратить бумагу и мучить совесть лимитами на размер фолианта и масштабом вырубленных деревьев. Проблема только в том, что весь этот текст надо прочитать. И желательно что-то оставить в голове.

Схема контактов #saola-1 приведена на рисунке. Это - самая удобная и потому наиболее популярная форма представления. Контактов много, в отличие от некоторых модулей на том же чипе. По факту все внешние выводы чипа идут на контакты модуля и затем на контакты платы.

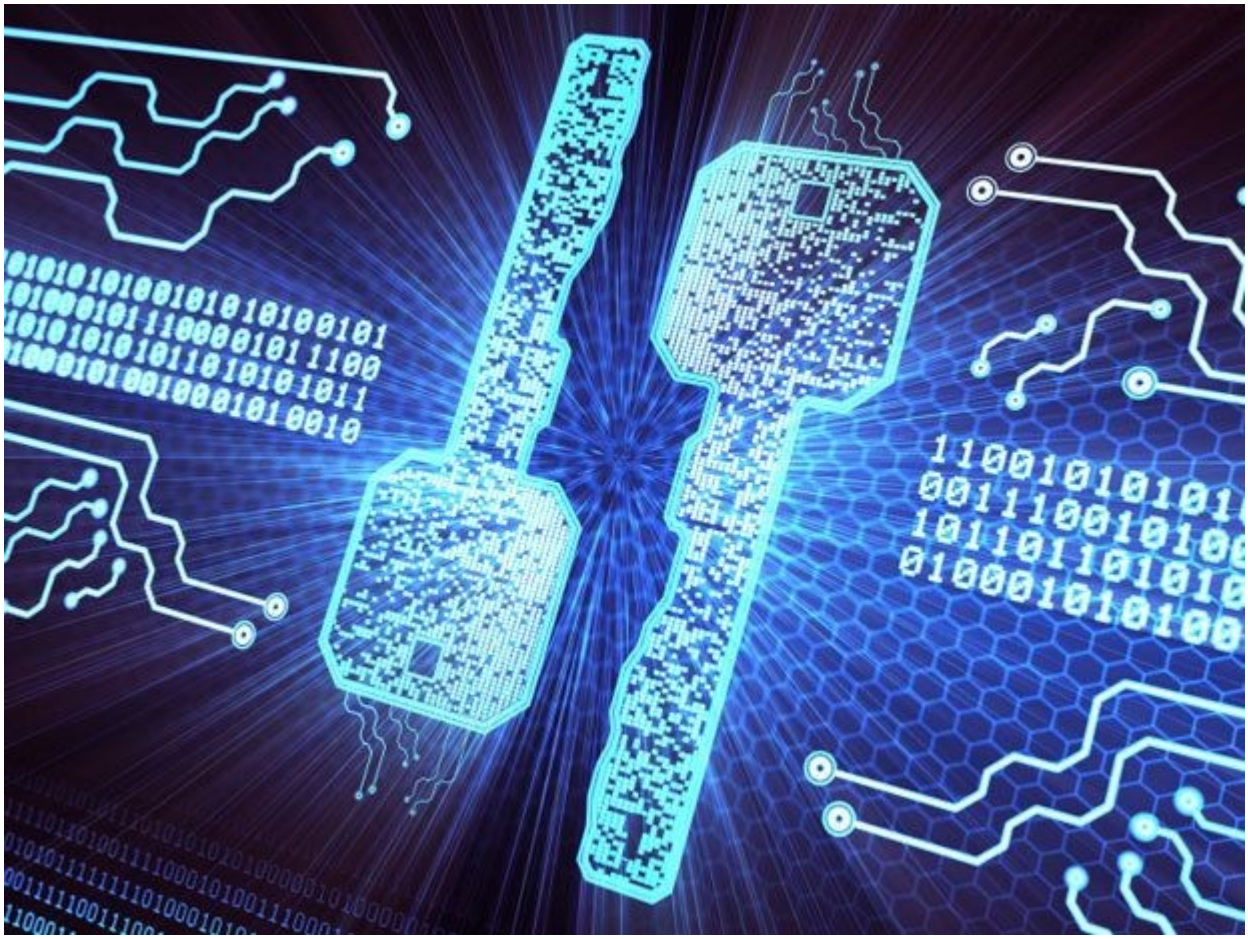


Часть 3: Безопасность

Эта крошка, ESP32-S2, имеет на борту аппаратные ускорители для вычисления современных функций шифрования. Можно подписать пакет данных цифровой подписью, можно применить #AES, можно по #RSA, а можно и просто посчитать #SHA-256.

Конечно, RSA обсчитывается неприлично долго, около 0,3 секунды. Можно применить упрощения, чтобы получить ускорение в 150 раз, но качество такого решения пока непонятно. Для большинства ускорителей есть два варианта поставки и возврата данных: через FIFO или блок регистров, или пакетами через DMA.

Схема безопасности достаточно проста. Несколько (немного) секретных (private) ключей хранятся на чипе в специальной памяти типа ROM с однократной записью. Перезапись не предусмотрена. Эта память может быть закрыта от процессора и это тоже одноразовая операция. Доступ к ключам есть только у ускорителей. Добавляем сертификат открытого (public) ключа на host-систему и можно выполнять защищенную и авторизованную передачу данных.



"Враг не пройдёт!" - написано на ветхом плакате. Глядя на него каждый день, можно стать параноиком безопасности и шифровать абсолютно всё: содержимое флэшки, сетевой трафик, каналы I2C и SPI и всё прочее. А на самом деле это всё лишнее? Отнюдь. Интеллектуальная собственность создаётся годами и конкретными почти незаменимыми людьми, а потому и дорого стоит.

Вдруг стало понятно, почему в версии S2 линейки ESP32 у процессора только одно ядро. Второе убрали и на его место поставили ускорители. Соблюден баланс интересов мажорных покупателей, параметров чипа и возможностей фабрики. Но это исключительно моя версия.

Часть 4: Экономичность

Процессоры #IoT имеют одну экстремальную особенность, в отличие от "нормальных" процессоров: они практически не потребляют энергию в спящем режиме, но при этом что-то на чипе всё еще функционирует.

Если взять ESP32-S2, то в режиме глубокого сна и мониторинга сенсоров он сможет работать на "часовой" батарейке CR2032 целых 9600 часов (13 месяцев). Если включится специальное низко-

потребляющее ядро процессора, то можно даже выполнять небольшую программу килобайт на 8, и той же батареи хватит на 1000 часов (40 дней). При этом процессору доступен канал внешнего ввода-вывода типа #I2C.

Что это означает для устройств на базе такого процессора: если устройство снабжено резервным питанием в виде батареи-пуговки или аккумулятора, то такое устройство будет работать "вечно". Ну то есть ровно столько, насколько хватит ресурса батареи. То есть годы без сервисного обслуживания. Это как батарея RTC в обычном компьютере - лет пять точно. Кстати, вы знали об этом? Или компьютер меняется раньше, чем разрядится батарейка? 😊

На рисунке - типичный режим работы, как это «видит» источник питания.

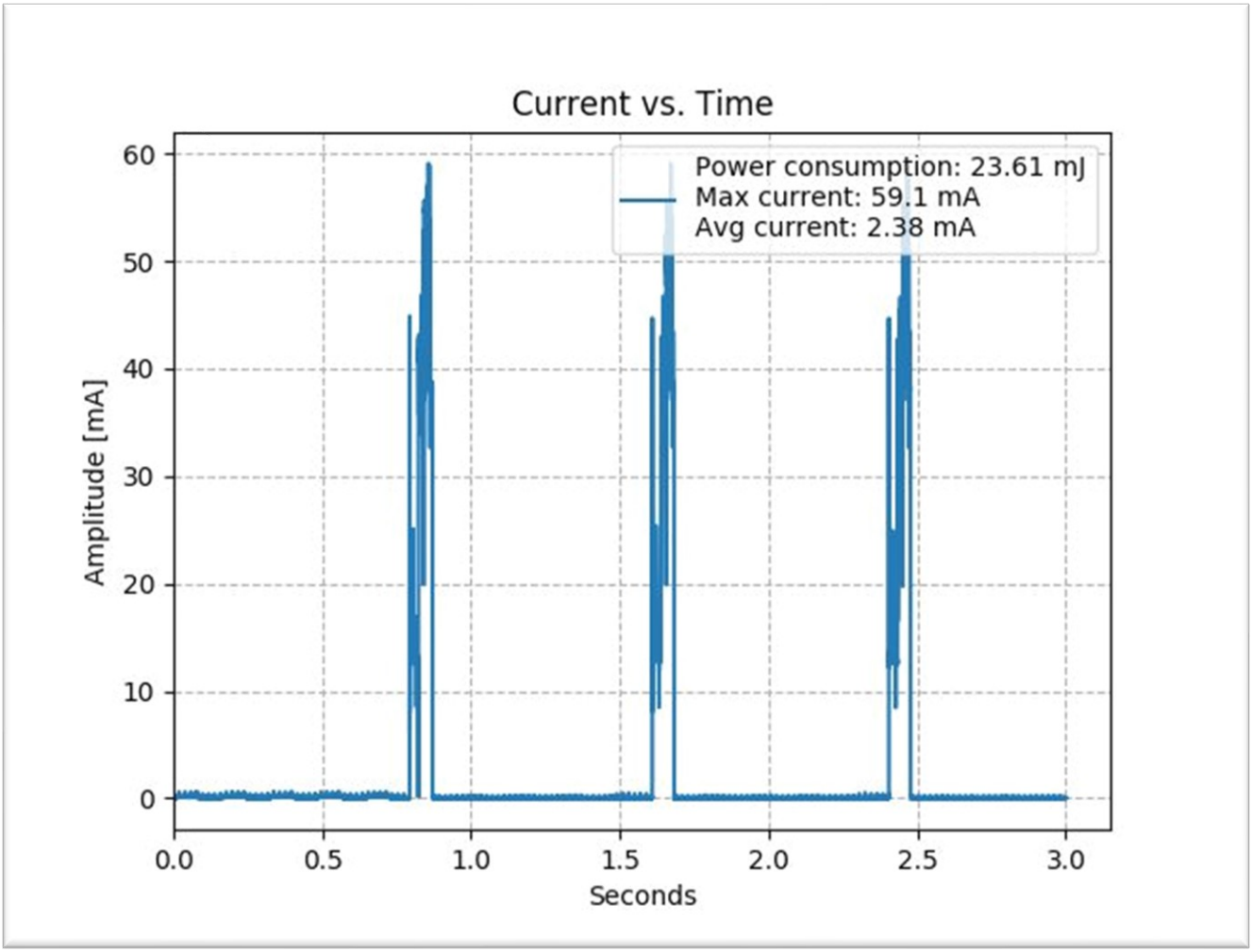


График © Espressif Systems, взят из GitHub, examples/system/ulp/image/ulp_power_graph.png

Комментарий к "Max current: 59.1 mA". Это, скорее всего, режим работы основного процессорного ядра Xtensa®, без Wi-Fi. Чип ESP32-S2 потребляет на себя в этом режиме экономии максимум 19 мА, остальное приходится на внешнюю память FLASH и PSRAM.

Вот ссылка на очень полезную статью с разъяснением режимов спячки предыдущего процессора, ESP32 с ядрами LX6. Новый ESP32-S2 на одном ядре LX7 оказывается более экономен, раза в 2, почти во всех режимах. Но эта экономия не относится к Saola-1 в полной мере. На плате есть шикарный светодиод-трехцветник и чипы коммуникации. При питании от линии 5 В чип питания может выдать до 1 А в линию основного питания 3,3 В. Хотя это не точно. Моя Saola – это китайский клон оригинального китайского производителя. 😊 На нём из оригинального – только WROVER. Точные параметры других чипов у меня под сомнением.

Программирование - тонкая штука. Проблема пришла откуда не ждали. Штатный пример от #Espressif для "ассемблерного" #ULP процессора работает для ESP32, но не для ESP32-S2. 🙃 Даже не компилируется. После первых правок кода теперь можно загружать прошивку, но при внимательном рассмотрении оказалось, что ULP процессор даже не стартует. 😆 Так что битва продолжается. 💻

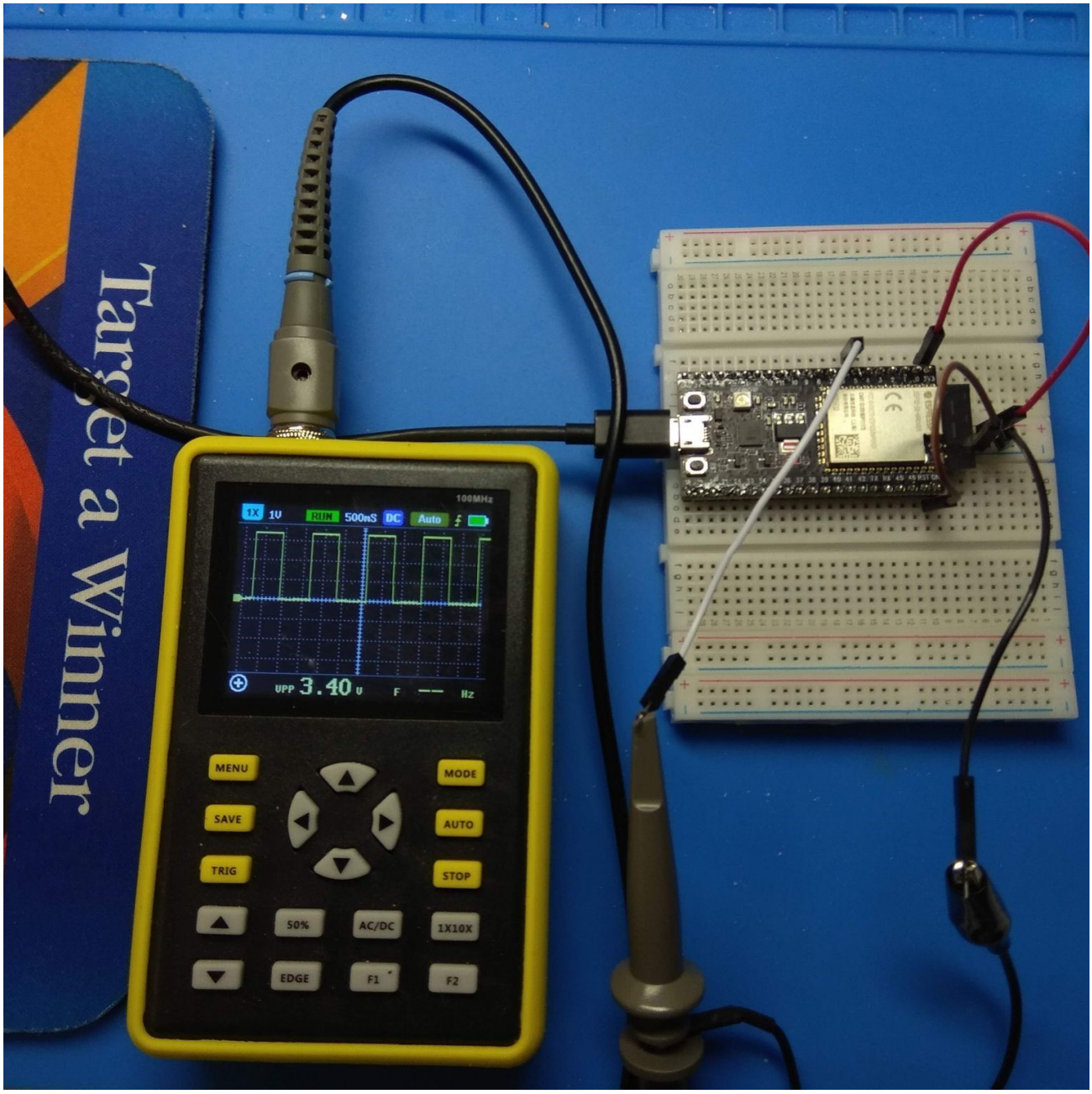
Хорошо ходить в ресторан на обед: приносят уже готовую курицу, а весь кулинарный процесс остается "за кадром". Так же приятно тыкать кнопки банкомата и водить пальцем по смартфону, потому что весь процесс программирования сложных систем где-то "там" остался и никак не ощущается. Мой #ULP процессор по-прежнему отказывается работать, но из хороших новостей - можно программировать для #ESP32 и на #C++. Получается очень компактный и понятный код. Но сам C++ — это еще та "штучка". Надо, при удобном случае, об этом процессе написать поподробней.

Часть 5. Битва за ULP

Инженерный труд — это победы и поражения. О победах говорим пафосно, о поражениях вспоминаем с тихой грустью. Битва за низкопотребляющий сопроцессор #ESP32-S2 пока не закончилась. То есть пока без победы. Есть предчувствия, что системный API от #Espressif (ESP-IDF) еще не доделан в этой части. Запрошена "помощь зала", как говорили в одной популярной телеигре. Жду результат с форума. 🏚

А пока — о лёгких победах. Есть примеры для стабильной части оборудования. Они запускаются на раз-два-три. И работают, как и задумано. Что и видно на фото. Теперь можно внимательно почитать код и поковырять его на экстремумы. ✨

"Терпение и труд всё перетрут". Эта фраза из советского детства по-прежнему работает. Удалось запустить сопроцессор. Родимый API от Espressif Systems точно не поддерживает эту возможность, но оказалось возможным запустить #FSM "ручками". То есть простыми обращениями к регистрам. Да, пока таймер не участвует. И повторный запуск тоже не получается. Но главное в том, что сопроцессор живой. В чём появлялись некоторые сомнения. Плата была куплена как Saola-1, но есть подозрения в её оригинальности. Прошивка сообщает, что она - "release candidate 4". Чуток стрёмно от "кандидата". Но для обучения пойдёт и такая.



Есть аппаратные трюки, которые не доходят до документации. 😁 Чтобы запустить процессор, надо закинуть в некий регистр единицу, читаем мы английским по белому. Если забыть написать в документе, что сначала там должен быть ноль, то можно годами искать причину "нерабочего" блока #FSM. Или продавать "секретное" know-how. ⚛️

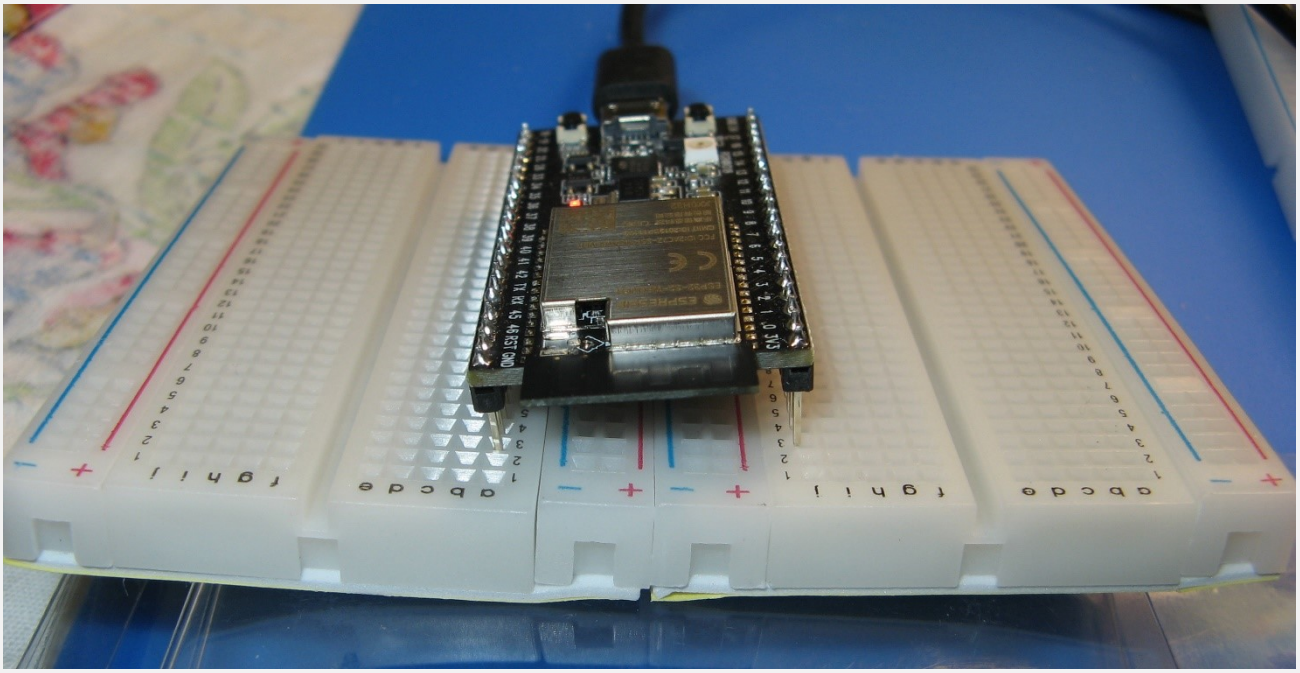
"В чём сила, брат?" - спрашивает ключевой вопрос герой известного фильма. Сила в Интернете, в который раз убеждаюсь я на многих примерах. 🌐

Вот и сейчас, сработала "помощь зала". Пришел совет незнакомого мне человека, даже откуда - не знаю, с готовым частным решением моей проблемы. Оно мне не совсем подходило, но в нём оказалась ключевая информация, которой мне не хватало. И вот - победа! Мой тест для #FSM заработал, как и было задумано. Ура! 🎺

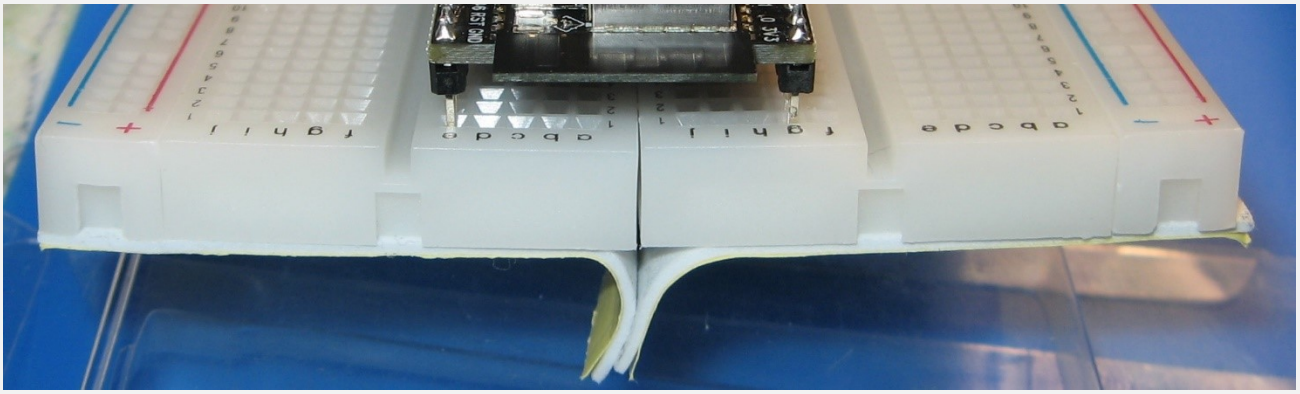
А в фильме ответ на вопрос звучит так: "Сила - в правде". Это так, я в этом уверен. "Правда" — это математическая функция, и у нее есть формула. А люди через Интернет помогают наполнить эту функцию точностью. Но это уже другая история, про #AI и #AGI, и об этом - в следующий раз, при подходящем случае. 🖨️

Кстати, по отвлеченному поводу. Мир разделился на два лагеря: дюймовый и систему единиц ISO. Ножки этой китайской платы и монтажной доски идут с "дюймовым" шагом 2,54 мм. В наших краях чаще принят шаг 1 мм и его кратные. Если, конечно, вы не работаете исключительно с американо-китайским импортом комплектующих. 😁

Встречаются обидные инженерные ляпы. Вот на этой монтажной доске (на фото), замечательно задуманной для стыковки по всем сторонам, нарушается дюймовый шаг при переходе с одной доски на другую. 😕 Поэтому - без рекламы, но с иллюстрацией.



Но нас так просто не возьмешь, голыми ручками! 😊 Немного смекалки, и проблема решена. Обойдемся без линий питания посередине. Острым ножом подрежу подложку-липучку и доска будет участником всей фотосессии.



Кстати, липучка – одноразовая. Непонятный концепт для многоразовой комбинаторики.

Часть 6. Линейка продуктов

Линейка процессорных продуктов от Espressif Systems немного путает названиями. Я бы предложил такое понятное разделение, основанное на принципе многопоточности.

ESP8266 --> ESP32-S2

Это одноядерные вычислители. Ну то есть почти, у второго есть дополнительный процессор для экономного режима. Эти машинки подойдут для конфигураций, где они занимаются своей главной

работой и только по возможности общаются с внешним миром. Если на них организовать искусственную многозадачность, то получим известные болячки одноядерных ЦП.

🎯 ESP32 --> ESP32-S3

Это вычислители с двумя основными ядрами. Это значит, что одно ядро делает главную работу, а второе ведет полноценную коммуникацию с внешним миром, без "тормозов" и пауз.

🎯 ESP32-C3

Это тоже одноядерный вычислитель, но в отличие от первой линейки, он нацелен на экономию энергии. Батарея — это жизнь. Чем больше остаток заряда - тем жизнь длится дольше.

С момента первоначальной публикации не прошло и полгода, а Espressif Systems уже добавил пару продуктов в линейку ESP32. Вот ESP32-C6 идёт вдогонку ESP32-C3, с Wi-Fi 6-й версии. Зайдите на сайт компании за последними новостями. Не все продукты продаются массово, поэтому читайте, смотрите и спрашивайте образец на тестирование. Как говорил один мой хороший коллега, “asking is free”.

Сила продуктов возникает из коллективной разработки. Моя плата Saola-1 клонирована неизвестной компанией из более дешевых компонентов. Это легко, электрическая схема находится в открытом доступе. Модуль, который я изучаю, построен на базе микросхемы #ESP32-S2 от Espressif Systems, а та, в свою очередь, использует основное процессорное ядро #Xtensa от Cadence Design Systems. Сайт компании Cadence огромен, поэтому вот прямая ссылка.

Очень надеюсь, что и эти заметки добавят силы в коллективный стек.

Часть 7. Программирование

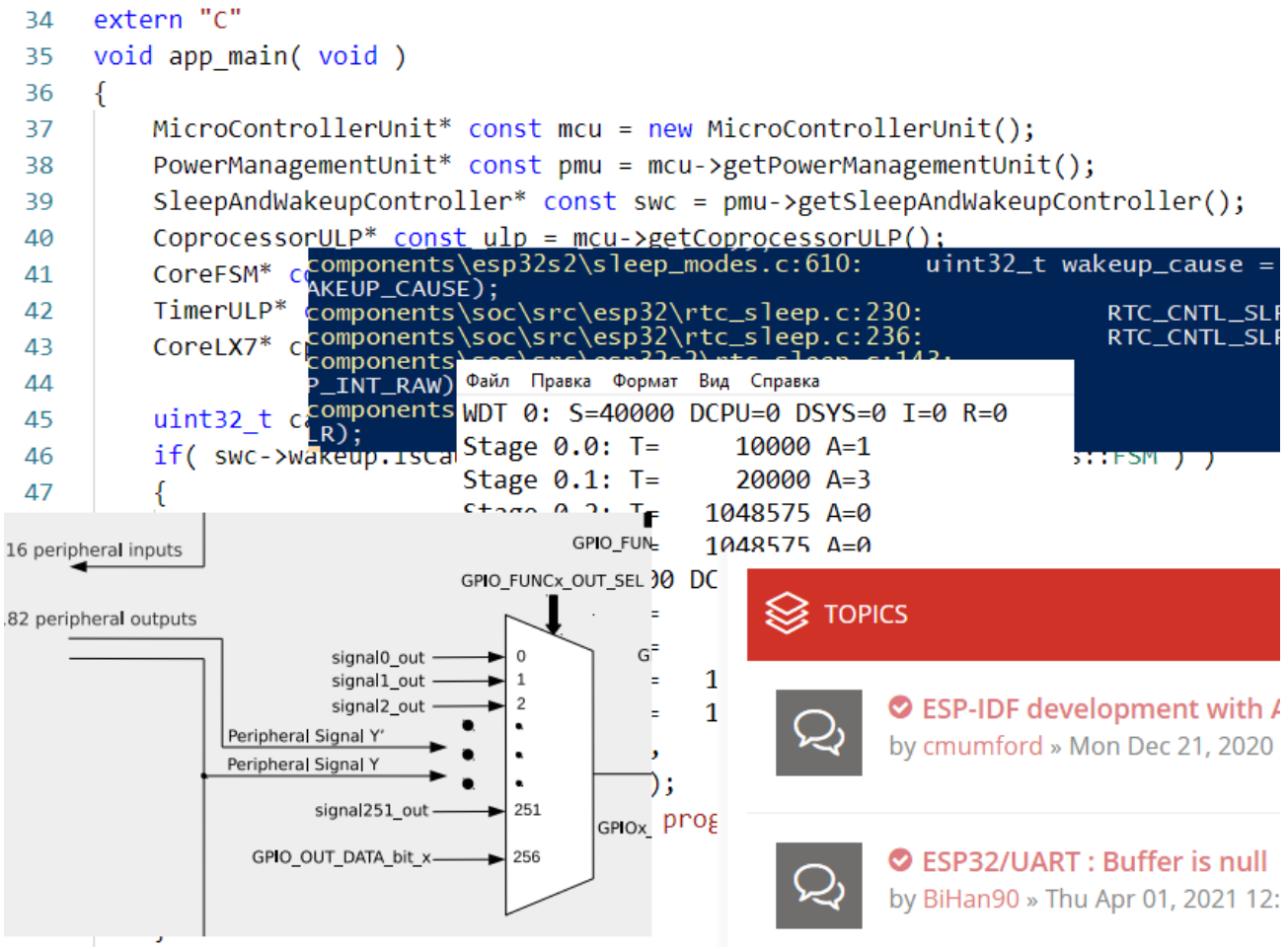
Программирование - увлекательная профессия. 🤓 Нескучная точно. Это если различать её от кодирования. По аналогии с другими профессиями есть инженеры-конструкторы и есть инженеры-технологи. Кодировщики - они больше технологи. Они уверены в результате. Конструкторы - не всегда на 100%. 😬 В моём случае - я больше конструктор и на 100% исследователь. Вот так.

Система программирования для ESP32 базируется на адаптированной FreeRTOS и собственном API "ESP-IDF" от Espressif Systems. Автоматизация постройки программного кода использует CMake. Но это только базис и всё, что вы получите, будет набор команд для консоли и простенький настройщик конфигурации в стиле программ для MS-DOS прошлого века. 😂 Так жить точно нельзя, поэтому я перевел весь процесс в Visual Studio Code и пока удовлетворён таким решением. Выбор Studio был спонтанным. Есть и другие варианты. Кто ищет – тот всегда найдет!

В VS Code есть странная особенность. Я такую встречал в Embarcadero тоже. Начинаешь новый проект, вроде и пути к компонентам уже настроены, а IDE в упор не видит ни локальные классы, ни библиотечные. Но стоит сделать хотя бы один build, как что-то "щелкает" и классы становятся узнаваемы, появляются подсказки к классам, методам, полям и переменным.

"Щелкает" только после перезагрузки проекта. Но и это не совсем так. Везде свои секреты и заморочки. 😛

Это так непривычно в сравнении с Java! Там если подключил библиотеку, то сразу получил по ней всю информацию. Немедленно. Ну то есть почти немедленно. Это IDE немедленно начинает сканирование библиотеки и кэширование информации в фоновом процессе. К хорошему сервису привыкаешь быстро.



Вообще-то надо попробовать CLion. JetBrains вырос в мощную и зрелую компанию, которая как никто другой понимает потребности программистов и делает инструменты, которые de facto становятся стандартом в отрасли.

Ну вот, неожиданно Visual Studio Code поломал интеграцию с Python в окне Terminal в недавнем обновлении. Да и установку ESP-IDF помял немного. Последнее починилось переустановкой. Так что при удобном случае надо попробовать альтернативные IDE. Выбор есть. А Python – он теперь везде по чуть-чуть. Куда ж без него! 😀

Вообще-то мир C/C++ суров. Еще со времен 90-х прошлого века. Там всё идет сложно. Поэтому я по-настоящему люблю только Java. И знаете за что? Не только за сам язык, а еще и за JCP - Java Community Process. Этот, по сути, публичный механизм принёс в Java силу, стабильность и веру в будущее.

Я сообщил о проблеме. Это было не очень просто технически и заняло много времени. Но когда рушится налаженный процесс, стоит идти на такие затраты. Потому что наладить процесс под другой IDE может стать дороже. Но дела пошли не очень гладко. Дефект закрыли без рассмотрения. Я живой человек, эмоциональный, разрядил эмоции, написав текст, что выше.

А теперь о чудесах. Дефект починили. Только что. Пары дней не прошло. Может быть, заодно с чем-нибудь еще сломанным. Спасибо, #Microsoft.

Что для меня на самом деле важно, это то, как люди и компании решают проблемы, возникающие в процессе взаимоотношений. Мир не идеален, проблемы есть и будут. Но если есть желание их решать, жизнь будет продолжаться, со взаимным успехом.

На рисунке просвечивает код на #C++. Это мой собственный #API по управлению #MCU, без участия FreeRTOS, напрямую к регистрам периферии и RTC. В данный момент — это что-то типа иерархического набора helper классов. Писать на нем легче и код понятней. О скорости кода напишу отдельно. В финальной перспективе хотелось бы совсем убрать FreeRTOS. На этом чипе он непонятно зачем.

Предлагаемый API заточен на язык C. И поддерживает C++, хоть и не весь стандарт. Что немного смущает, отказаться от FreeRTOS совсем не просто. Её польза понятна для двухъядерного базового чипа ESP32, но для одноядерного ESP32-S2 преимущества не так очевидны. FreeRTOS съедает некоторые критические ресурсы и периферию, а его виртуальная многозадачность - спорная концепция. Он занимает 160 кБ памяти (это неточно) из 320 имеющейся. Зато в ней имеются драйверы на сложную периферию типа WiFi, и это - несомненный плюс. 🤦

⛑️ 💲 🍾 Если кому интересно, я могу обсудить бизнес-варианты по дальнейшей целевой разработке. Дело за вами! Welcome, comrades!

Ниже еще один фрагмент кода на C++. О преимуществах ООП расскажу в следующий раз. Потому что C++ — это просто. Если писать код грамотно. И в сумме - экономно для проекта, по времени и деньгам.

Полная компиляция простенькой программы на C++, вместе с линковкой, длится примерно 4 минуты на i7-4510U+SSD. При этом ESP-IDF строит граф зависимостей конкретного проекта, включая свои библиотечные компоненты. Метод хорош тем, что последующие изменения моего кода компилируются за секунды, потому что инкрементно.

```
PulseCountController* pcnt = mcu->periphery.getPulseCountController();
pcnt->reset->on(); // strangely it was needed once
pcnt->reset->off();
pcnt->clock->on();
pcnt->enable->on();

PulseCountUnit* pcu = pcnt->getUnit( 0 ); // 1..3 are good fit too
pcu->getFilter()->enable->off(); // otherwise fast ticks will be lost
printf( pcu, "initially" );

PulseCountComparator* pcuc = pcu->getComparator();
pcuc->interrupt.enable->off();
pcuc->interrupt.clear->on();
pcuc->high.enable->off(); // no reset on limit
// pcuc->high.threshold->set( pcuc->high.threshold->max() >> 1 ); // +32767
pcuc->low.enable ->off(); // no reset on limit-
// pcuc->low.threshold->set( ~( pcuc->high.threshold->max() >> 1 ) ); // -32768
pcuc->thr0.enable->off();
pcuc->thr1.enable->off();
pcuc->zero.enable->off();

PulseCountChannel* pcc0 = pcu->getChannel( 0 ); // all inputs in IGNORE
printf( pcc0 );
```

В build входит примерно 1000 объектов. Это код ESP-IDF поверх FreeRTOS. По текущей практике, в большинстве случаев CMake компилирует только несколько файлов за секунды, линковка и сборка bin - чуть подольше и столько же идет прошивка. То есть в целом процесс не напрягает "задумчивостью". Самый задумчивый элемент в этой системе — это я. 😀

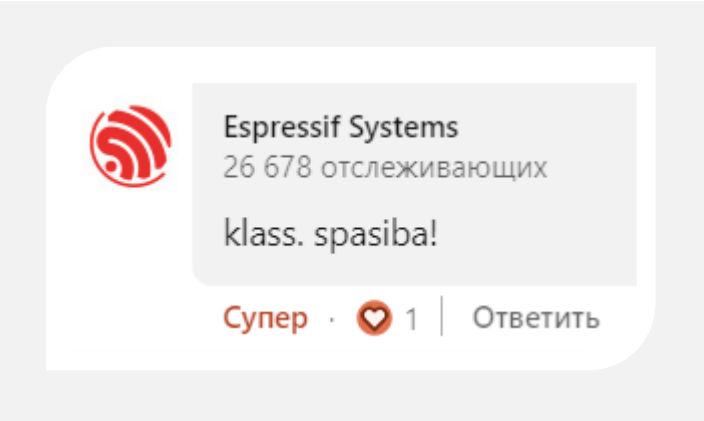
Но есть и проблемы, если одновременно идет разработка общего кода, который сидит в других директориях, а не в файловом дереве текущего проекта. После добавления нового общего файла проект приходится реконфигурировать, иначе новый файл оказывается неизвестен. Возможно, это есть приколы #CMake в моей конфигурации.

CMake, кроме того, что он – работяга, был замечен в двух положительных моментах: реконфигурация проекта инициируется автоматически по коммитам Git и по изменению файлов "CMakeLists.txt", участвующих в текущей конфигурации.

Размер программы, вернее образа для загрузки на микросхему FLASH, около 167 кБ и пока почти не зависит от размера моих программ. Они крошечные, потому что учебно-пробные. Я так понимаю, весь объем кода — это практически только #FreeRTOS.

У моей платы Saola-1 нет дисплея. Совсем. Точнее — пока. Есть порт microUSB для питания и коммуникации. Поэтому наблюдение за поведением программы весьма рудиментарно. Я использую отладочную печать на stdout в консоль Studio и осциллограф. Каменный век, точно. 😢 Про отладчик расскажу позже, когда попробую.

Всегда приятно получать подарки и благодарности. Да хотя бы просто заметную статистику просмотров. Вот один из них:



Часть 8. Мягкий генератор

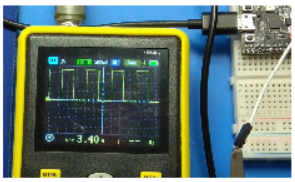
Так хочется верить в чудо! Но их в природе не бывает. Разве что, если прогулял всю физику, химию и математику. Потому что весна, футбол и многое другое.

```
Macro...   7996998 Hz
Inline 32 bit...   5452432 Hz
Simple 64 bit...   1211644 Hz
Write and read 64 bit...    754397 Hz
Test write and read 64 bit...    644897 Hz
Log write and read 64 bit...
                0 v4=1 v8=1
                ...
               49 v4=0 v8=0
        282 Hz
Channel 1 bit...   2695549 Hz
Channel write and read...   1310945 Hz
Test channel write and read...   1193556 Hz
Log channel write and read...
                0 v4=1 v8=1
                ...
               79 v4=0 v8=0
        271 Hz
```

```
void runChWrRead( const uint32_t max, SystemTimer* stm,
{
    printf( "Channel write and read..." ); fflush( stdou
    vTaskDelay( 1 ); //esp_task_wdt_reset();
    uint64_t t0 = stm->getValidValue( TIME_ATTEMPTS );
    bool b4 = true;
    for( uint32_t i = 0; i < max; i++ )
    {
        ch4->write( b4 );
        ch8->read();
        b4 = !b4;
    }
    uint64_t t1 = stm->getValidValue( TIME_ATTEMPTS );
    uint64_t t2 = stm->getValidValue( TIME_ATTEMPTS );
    printf( " %9.0f Hz\n", FREQ * max / ( t1 - t0 - (t2-
}
```

CPU and Memory

- Xtensa® single-core 32-bit LX7 microprocessor, up to 240 MHz

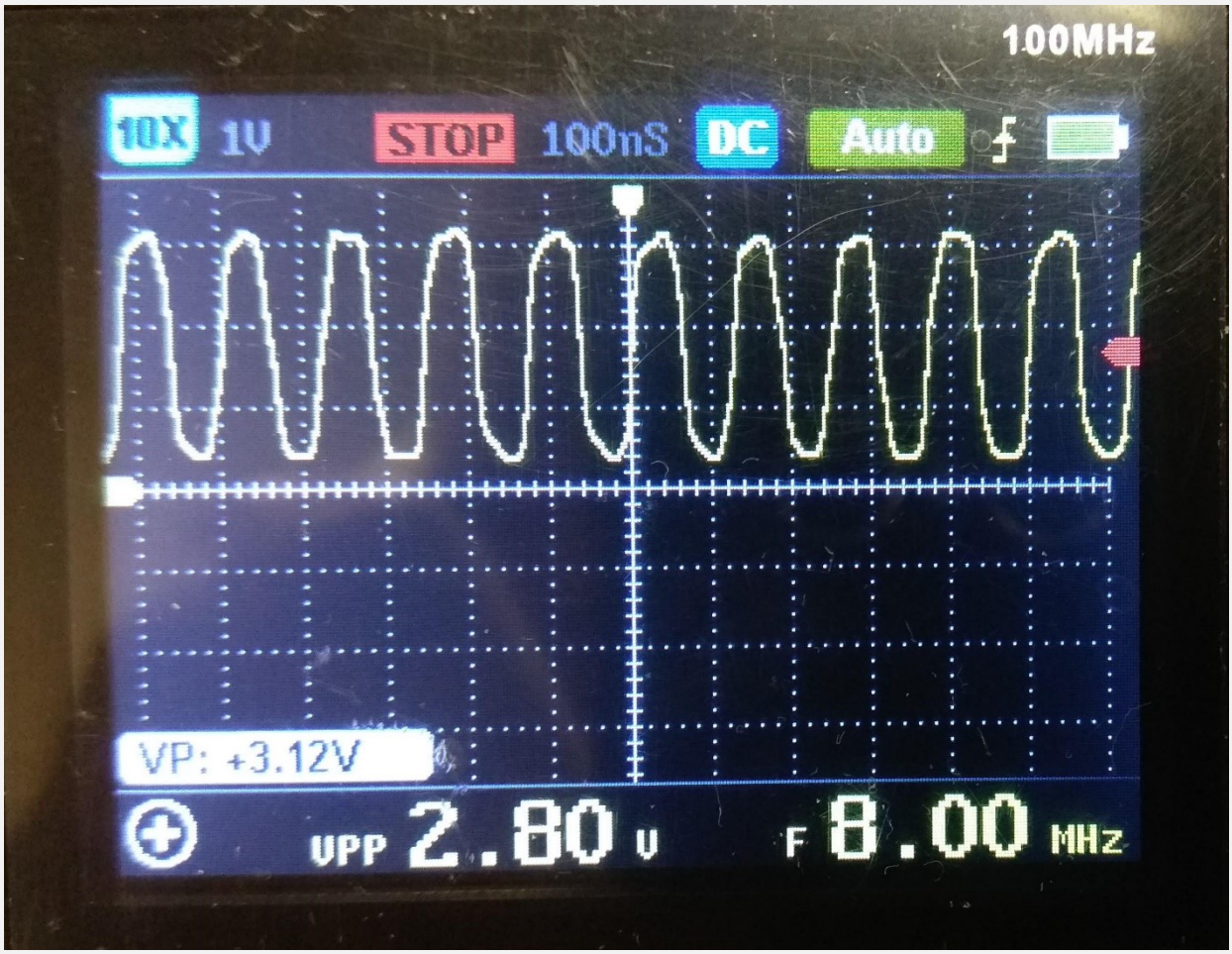


Закончен первый тест производительности для ESP32-S2. Это не мой целевой тест, но всё же результат важен. Суть теста - измерить скорость программного потока путём простого инвертирования значения на внешнем цифровом выводе.

Максимально, что удалось получить — это 8 МГц. На процессоре с частотой 240. Навскидку. на каждый полученный цикл приходится 30 тактов ЦП. Программный цикл потребляет 15 тактов (в электрическом цикле два фронта сигнала, + и -). Вроде бы немного тактов, но эмоциональная разница между 240 и 8 столь велика, что чудо не случилось.

А теперь о реалиях, то есть о прочих тестах. Если перейти на "быстрый" C++ (inline и т. п.), частота снижается до 5,5 МГц (терпимо). Если использовать "однобитный" вывод - уже 2,7 МГц. Если хочется применять идеальную модель буфера из 64 бит, получаем 1,2 МГц (тоска..., но процессор-то 32-битный). При желании проверить выведенное через другой канал частота ожидаемо снижается вдвое до 0,65 МГц. Это почти упс. 😀

А теперь немного о паранойях: если хочется проверить каждый такт, получаем частоту - 270 Гц! Медленнее только паровозом. 😁 Это происходит потому, что идет отладочная печать через UART 115 кбод. Кстати, его можно разогнать (он же на USB по факту), но это потом, потом...



В отличие от “мягкого” генератора "твердый" позволяет формировать более стабильную цепочку импульсов. На рисунке пример модулированного сигнала. Кстати, его можно демодулировать обратно другим каналом. Обо всём этом расскажу подробнее на этой неделе. Не торопите!

Эта картинка, как и все остальные, получена на осциллографе FNIRSI-5012H. Если захочется себе любимому, ищите на AliExpress. Это не профи-прибор (хочу Tektronix! но он дорог для тренировок). Реально в "цифре" работает до 20-30 МГц (у китайцев свои мегагерцы 😁), обещано было целых 100 в "аналоге". Без синхро-входа и USB интерфейса, поэтому снимки сделаны фотокамерой. Но прибор стоит недорого, работает от аккумулятора честные 4 часа и прост в управлении. Меня устраивает.

Часть 9. Твердый генератор

Гонг! "Мягкий" против "твердого" - кто победит? Недавно я рассказывал про программный генератор импульсов. Было доблестно получено 8 МГц стандартного сигнала. Однако сложилось впечатление, что это - предел выбранного метода. Сегодня речь пойдет о его конкуренте - программируемом генераторе импульсов, называемом в #esp32s2 как RMT.

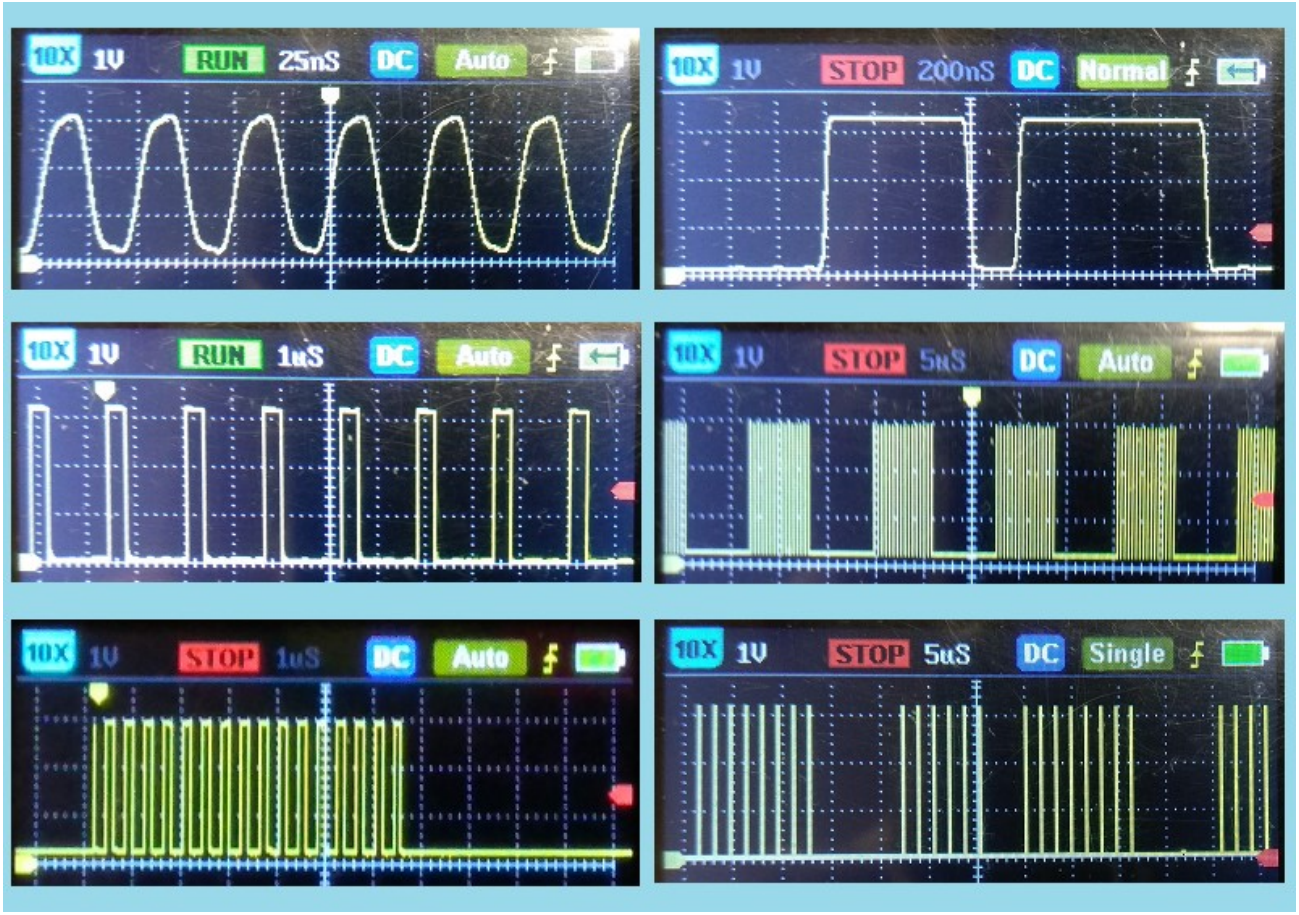
Чем же так хорош RMT? Самое главное - он реализован в самом чипе. Как только он будет запущен, цепочка выданных импульсов никак не будет зависеть от того, чем занят процессор. Второе преимущество - большая частота. Теоретически можно получить 40 МГц. Это сложно показать, на первом снимке - всего 20 МГц и не похоже на цифру. 😁 Не волнуйтесь, всё на самом деле хорошо. Просто для таких частот карманный осциллограф уже не подходит.

Генерация импульсов идет по шаблону, что хранится в памяти. Шаблон скромненький, на 64 импульса, но можно "украсть" память у соседних каналов и получится на все 256. Можно жить честно, но тогда процессор должен подкачивать остаток шаблона импульсов по мере их выхода. Морока, зато длинный сигнал идет четко.

И вот - вишенка на тортик! Выходные импульсы можно заменить генерацией высшей частоты. Если есть запас. Это видно на правом нижнем рисунке. И чтобы продлить wow, у #RMT есть обратный канал, который понимает всё, что сгенерировано.

Помните бородатый анекдот про "демо"? 😄 Так вот, получить эти импульсы - это 10% работы и это - демо. Остальные 90% уйдут на решение проблем. Типа той, что на левом нижнем рисунке: первый импульс сгенерирован невовремя, его остаток сгенерирован в конце. Схема выдала на один импульс больше и поэтому приёмник без фильтра встретится с проблемой.

Это не кляуза, не «фотошоп», а суровая бытность инженерной работы. Осциллограф делает 500 мегасэмплов в секунду, поэтому проблема реальна, а не есть дефект картинки. Кстати, именно для таких целей осциллограф и нужен. Лучше один раз увидеть, чем сто раз перечитывать протокол отладки.



Немного по поводу "синусоидной" генерации. Компьютерные гигагерцы играют прикольную психологическую шутку: мы к ним привыкли. И перестали воспринимать адекватно. 👮

Вот ноутбук работает на 2,8 ГГц в пике нагрузки. Но сравните, вот совсем рядом приборы на 2,4 ГГц - это диапазон Wi-Fi, Bluetooth и остальной компании публичного радиодиапазона! Где энергия излучается в пространство с любого провода, гуляет по помещениям и окрестностям. 🌊

Даже когда я сообщил о 40 МГц генерации на внешний контакт, это тоже много и тоже излучает. Помните, 87,5-108 МГц - это диапазон FM-радио (ловится на проводные наушники смартфона), а частотами ниже - тоже радио, но других диапазонов. Там требуются антенны больших габаритов, но ничто не мешает энергии утекать в пространство и с коротких проводов. Ловите! ⚡

Поэтому нет никаких претензий к маленькому осциллографу, подключенному несогласованными проводами. Видит, что может. И на том спасибо. 😁

Но я отвлёкся. Слышали про «хакер поневоле»? 👹 Звучит странно? Но в программистской жизни это - рядовое явление. Вот и здесь. Для #esp32s2, громко заявлялось, что ввод-вывод для #RMT возможен двумя путями: через FIFO-порт (но это старомодно, забудьте!) и напрямую через память.

Замечательно! 📢 Только в документации от #Espressif забыли написать, какой адрес у этой памяти. Мило.

Это затравка. Отсюда начинается азартное хакерство. 🛠️ Скан по открытому коду дал значение адреса для чипа #esp32. Ха-ха-ха, у этого чипа S2 адрес другой, безрезультатная находка. Делается dump памяти, как в старые добрые времена. И что же - сюрприз! Оказывается, RMT состоит из 4-х блоков на 4 канала, а не из одного. Аналогично счетчику импульсов. Но для пользователей открыт только один блок. ⛔

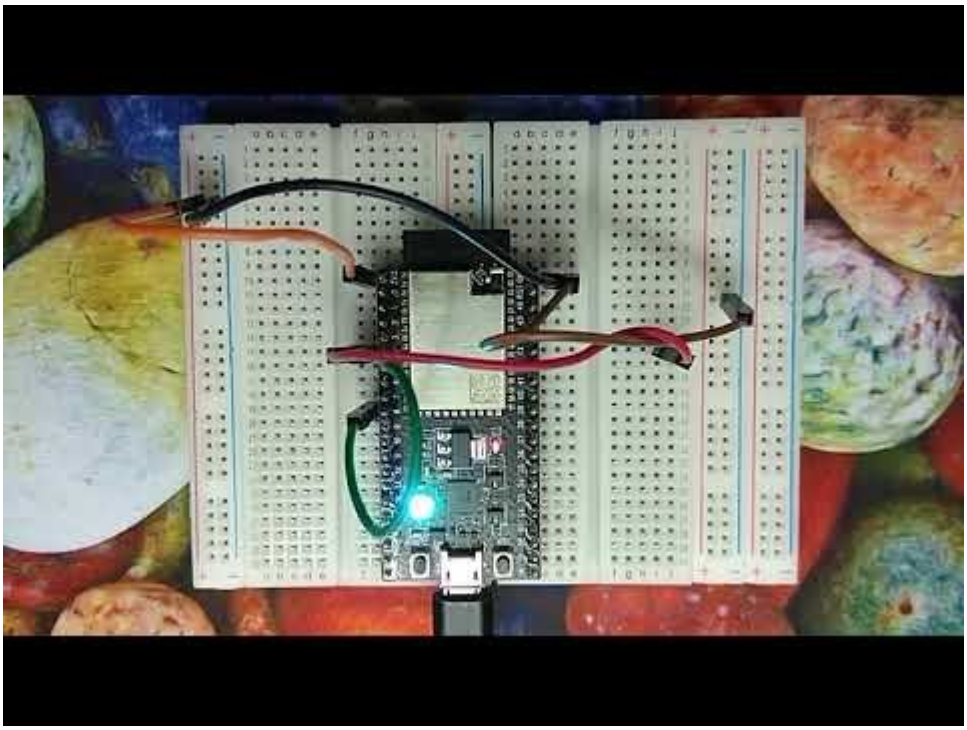
А адресок нашелся. 🎷 Пара-другая походов по коду - и всё обнаружено. Всё работает. Не совсем и секретно, но сделано исключительно для собственного API. Жизнь продолжается... 🌞 🌙

Победа достигнута, но я плачу, салфетки заканчиваются. 😢 Столько времени ушло... Форум у Espressif есть, но он пока скромный, ему до размаха StackOverflow пока далеко. А жаль.

Часть 10. Крошка-триколор

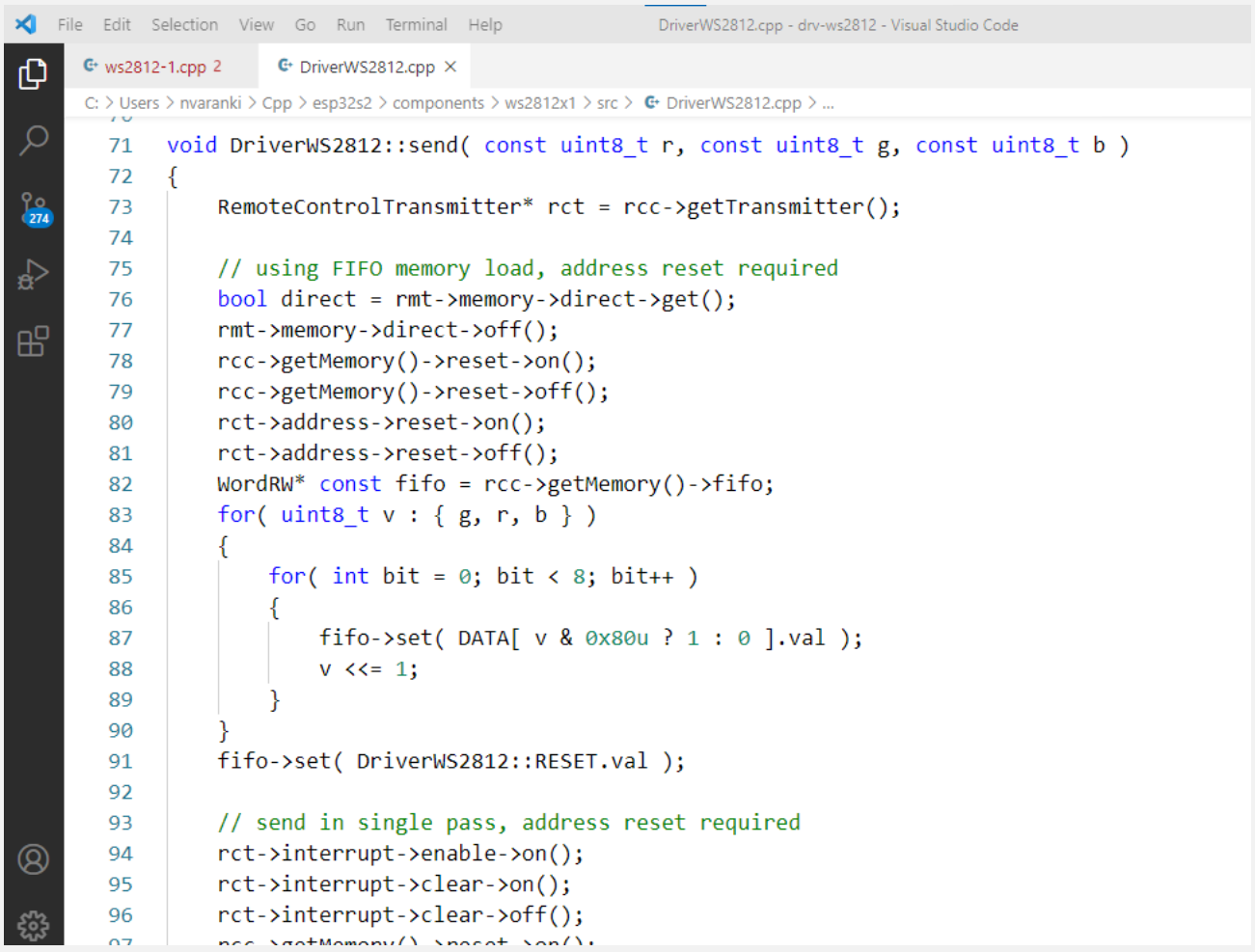
В дополнение к рассказу о "твердом" генераторе сигналов, снял краткое видео о работе драйвера трехцветного светодиода #WS2812. Драйвер работает на этом самом генераторе #RMT.

Благодаря C++, код краток и понятен, смотрите примеры далее. Иерархия программного кода несложная: комплексный объект для работы с #MCU <--> объекты-драйверы типовых протоколов и интерфейсов взаимодействия <--> код прикладной программы. Задача каждого уровня - спрятать сложность своей реализации за простыми интерфейсами управления.

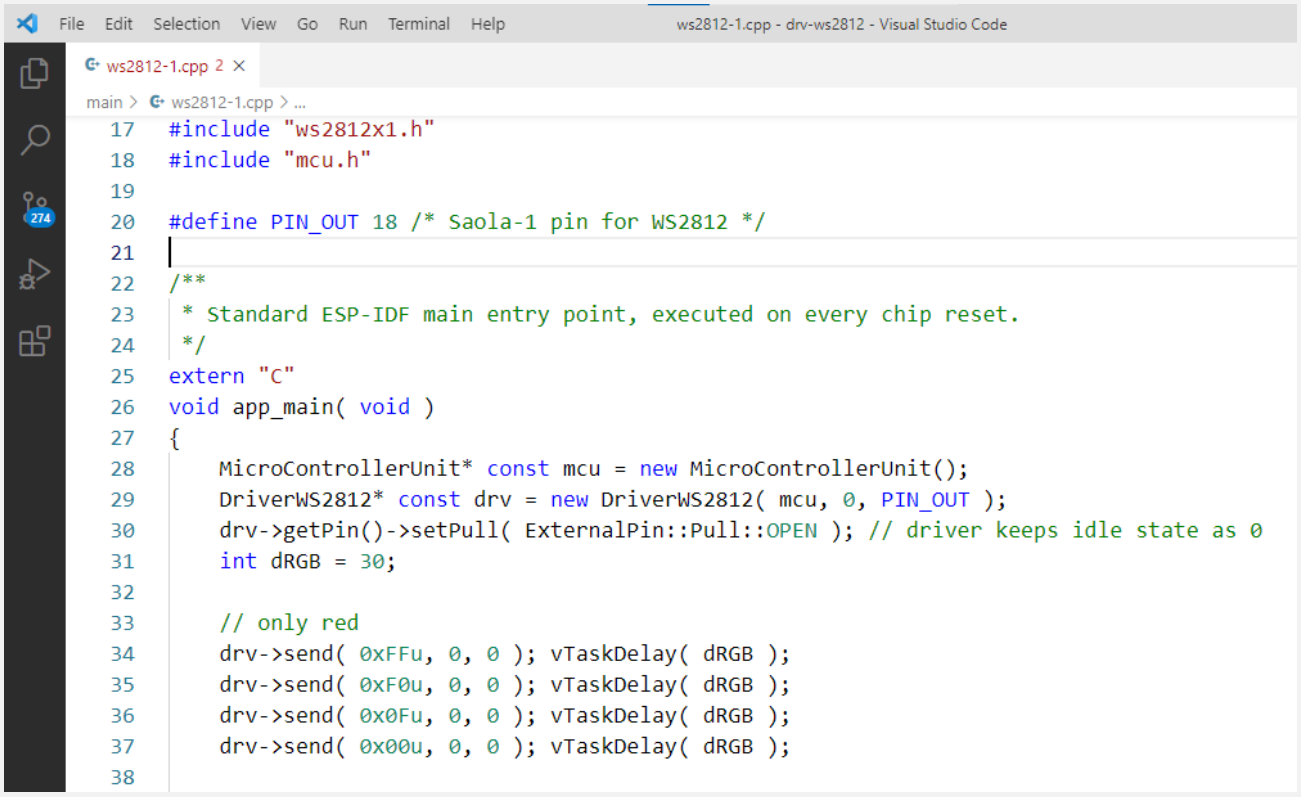


Это легко сказать. А на самом деле, вот спрашивают меня: «Ничего не понял. Что драйвер то делает?» Отвечаю. Базовый API работает с регистрами RMT. На нем построен драйвер протокола NZR (но это не тот протокол, про который вы сейчас подумали), он посылает bitstream. На этом драйвере построен драйвер семейства NeoPixel, он посылает массивы GBR троек на диоды. И в самом верху драйвер конкретного диода, он по факту только задает параметры сигналов 0/1/сброс по протоколу NZR.

А это - половина кода драйвера. Вот так работает C++. 🖉 Для более продвинутой версии (в перспективе), добавится только динамическое заполнение памяти. Сильно больше код не станет. 😃



И, наконец, код, который делает "мигалку". Все секреты раскрыты! Сначала тесты каждого канала RGB, а потом проход по всему пространству яркостей, в логарифмическом режиме. А если выводить каждую точку, будет минут 20.



📢 "Сильно больше код не станет." Обещания давать легко, если за них не отвечать. 😮 Профи-версия драйвера (почему я работаю только из-за любопытства?) - это уже не 50, а 450+ строк кода C++. Но — это код для "человеков". С комментариями, читабельным форматированием и осмысленными именованиями, в добавление к структурированному, функциональному и надежному коду. 💻 Однако бездушный компилятор отбрасывает это всё ему ненужное, как и линковщик. В результате код для флэшки занимает те же пресловутые 163 кБ. Ха-ха. 😆

```
Total sizes:
 DRAM .data size:    7544 bytes
 DRAM .bss  size:    2016 bytes
Used static DRAM:       0 bytes (      0 available, nan% used)
Used static IRAM:       0 bytes (      0 available, nan% used)
Used stat D/IRAM:   44579 bytes ( 315869 available, 12.4% used)
      Flash code:   95667 bytes
    Flash rodata:   23620 bytes
Total image size:~ 163866 bytes (.bin may be padded larger)
```

На следующем видео уже работает профи-драйвер. Так как профи, то никаких фрагментов кода (даром) не будет, не просите. 😉 Только за деньги.

Часть 11. Телевизор для космоса

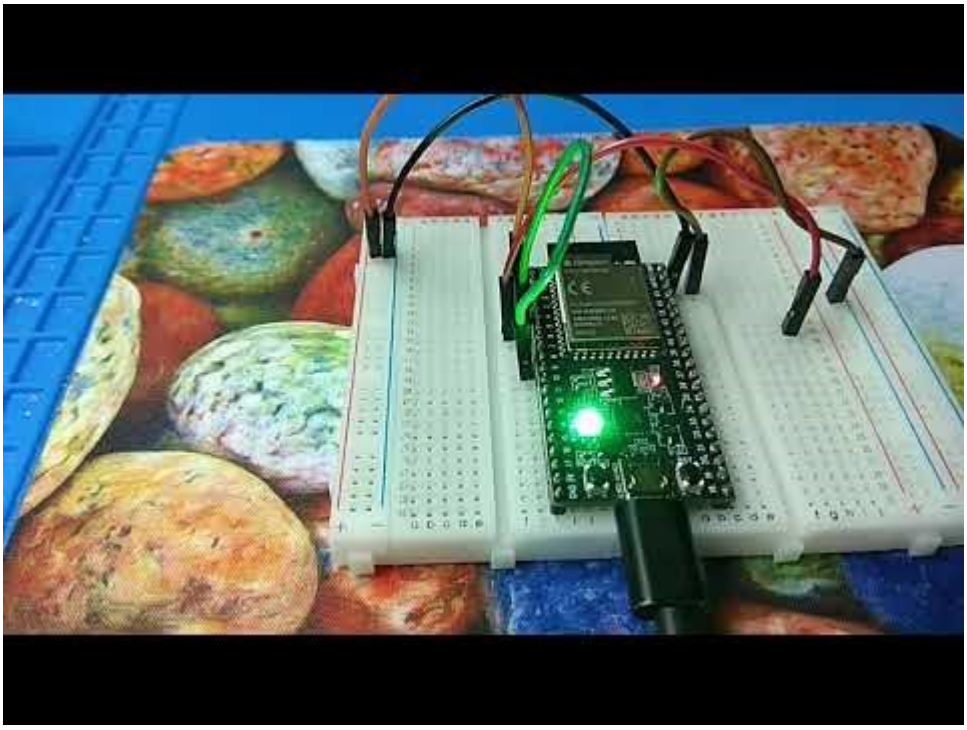
Есть разные идеи. Одни будоражат ум и чувства, а другим удается ещё и реализоваться. Вот сегодня обратил внимание на светодиодный щит, подошел поближе. В нём стоят трехцветные диоды с шагом миллиметров 10. Управление таким щитом можно сделать на #esp32s2. Если картинка будет 400x225 пикселей (это щит размером 4x2,25 м!), понадобится 57 или 28 mcu, работающих параллельно. Приблизительно, 1 mcu с 4-я каналами RMT обслужит 8 строк при 30 fps. У базового ESP32 на борту 8 каналов. Для него надо в 2 раза меньше. Но это всё - предварительный расчет. Вот щит на картинке построен из модулей, там расчет точно другой. Было б на чипе еще больше каналов, понадобилось бы меньше чипов.

Это не тот щит, что на нашей улице, картинка взята с video-ekran.ru. Кстати, а где в России пальмы растут?

Такое количество контроллеров не должно смущать совсем. Стоят они по 1,5 евро за штуку. А светодиоды стоят примерно 50 евро за строку, а их надо 225 (оптом дешевле, да?). И по сравнению со счетом за электричество такого шита (по нашим ценам будет 450 евро в месяц при круглосуточной работе), в итоге MCU оказывается самым дешевым пунктом в калькуляции.

Зато получится "телевизор", который будет видно из космоса! Диоды очень яркие. Но если космос не в фаворе, можно установить щит в торговом центре. Тогда электричество не пропадет даром в холодное время. Всё-таки щит — это не только видео-реклама, а и обогреватель на 8 кВт.

А на видео работает мой профессиональный драйвер для таких диодов и ESP32-S2. Это уже не 50 строчек кода, а 450+. Зато не "демо", хоть ролики и похожи. А размер бинарного кода - по-прежнему около 163 кБ. Пиши - не пиши, ESP-IDF выкинет всё лишнее.

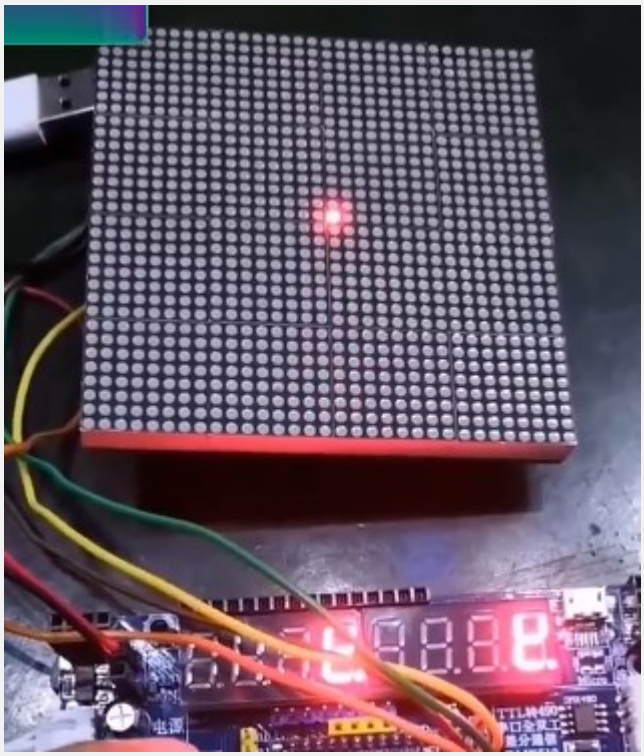


Почему нужен драйвер? 🤦 Главная причина - это возможность работать с объектами своей задачи, а не возиться с тонкостями конкретного оборудования. Для данного примера #RMT работает с потоком шаблонов импульсов. А драйвер светодиода уже использует GBR-код для цвета. Про биты и регистры речь уже не идет. Надо зажечь диод красным - задаем яркость для красного понятным числом и посылаем на диод.

Вторая причина - диоды разные, что и видно на фото от Тима. Моя Saola-1 — это клон. 😑 На ней стоит оригинальный модуль #WROVER, но всё остальное - другое. Вот светодиод в оригинале - WS2812. Я его так и программировал, по его документации. Пока не возникли заморочки. Начался "разбор полётов" и проверки с пристрастием. Выяснилось, что тип диода скорее всего SK6812mini. Потому что размеры другие и число ножек тоже. 🤣 Всё бы ничего (работает же!), но RESET у оригинала короче, да и другие параметры отличаются. Ему достаточно 50 мкс на сброс, а моему надо 80. Вот и заморочки. Очередная "пачка" приходит на 30 мкс раньше (RMT работает скрупулёзно, такт 12,5 нс). Её контроллер не ждёт, он пока занят предыдущей пачкой. Такие дела.

Этот замечательный светодиод в мире Arduino зовут #NeoPixel. На AliExpress их можно купить штуками, лентами, кольцами, прямоугольниками и много еще как. Управление простое, средняя частота сигнала 800 кГц. Но у разных производителей параметры импульсов немного отличаются.

А вот что такое полезное можно сделать на базе матрицы #NeoPixel? Видно, что быстродействие - адекватное. Вот навскидку первый пример, видео от XIXI W, найден на LinkedIn.



Вот еще один проект на #NeoPixel. Найдено у Nikunj Panchal. На другом MCU, правда, и не такой масштабный, как Космический Телевизор. Просто прикольно. Часики можно сделать на таком колечке, они есть на 12 и 24 диода, если поискать на AliExpress. По Wi-Fi соединиться с местным сервером NTP и будет вам точное время круглый год.

Так что дерзайте, и успех вам обеспечен!

Часть 12. Тест на AI

Нами двигают мечты, но вокруг нас - реальность. Такая, как есть на самом деле. Эта парочка противоположностей при совместной работе создаёт динамически стабильный процесс, название которому - жизнь. Здесь - персонально сложный для меня текст, потому что речь идёт о развенчании моих иллюзий. Но всё по порядку.

Этот чип - #esp32s2 - привлёк меня поддержкой #float32, расширяемой памятью, интерфейсом #SPI и заметной частотой процессора. И конечно, смешной ценой за "всё в одной коробке".

Я давно веду исследования по #AGI. По ряду параметров этот чип стал претендовать на роль "движка" в перспективной мобильной системе. Я сделал расчёт. #WROVER с 4 мБ флэш и 2 мБ PSI RAM обещал расчет гиперкуба 50*64*64*2. А 300 чипов (мечтать никогда не вредно) уже составляют грозную силу.

Но это всего лишь мои фантазии. Что я получу на самом деле, в моём примере, могло выясниться только на практике. И я купил #Saola-1, скромная инвестиция на 15 евро. Но это был волнительный момент. Последний раз я терзал начинку компьютеров в 1980-х (клоны PDP-11).

Если вы ещё не прочитали предыдущие части, там подробно про этот увлекательный процесс. И вот я подошел к главному тесту гиперкуба. Он сделан. Результат на картинке. Он неоднозначный. Мне ещё потребуется время его осознать. Идёт процесс развенчания иллюзий. Чтобы подготовить почву для новых проектов.

слоёв	входы	выходы	размерность	устройство	всего, нс	память, нс	матрица, нс	слой, нс	выход., нс		затраты/выход, мкДж
5	64	64	4 float	ESP32-S2	67 618 925	20 928 375	46 690 550	9 338 110	145 908,0	CPU 240 MHz, SPI RAM 40 MHz	
50	64	64	4 float	ESP32-S2	676 764 000	210 248 600	466 515 400	9 330 308	145 786,1	CPU 240 MHz, SPI RAM 40 MHz	
5	64	64	4 float	ESP32-S2	60 739 975	15 731 200	45 008 775	9 001 755	140 652,4	CPU 240 MHz, SPI RAM 80 MHz	
50	64	64	4 float	ESP32-S2	606 427 075	156 639 450	449 787 625	8 995 753	140 558,6	CPU 240 MHz, SPI RAM 80 MHz	
5	64	64	4 float	ESP32-S2	58 568 375	15 718 275	42 850 100	8 570 020	133 906,6	CPU 240 MHz, SRAM, ic=8k dc=8k	
5	64	64	4 float	ESP32-S2	58 539 850	15 704 600	42 835 250	8 567 050	133 860,2	CPU 240 MHz, SRAM, ic=16k	
5	64	64	4 float	ESP32-S2	58 557 525	15 708 050	42 849 475	8 569 895	133 904,6	CPU 240 MHz, SRAM, dc=16k	
5	64	64	4 float	ESP32-S2	37 203 925	15 701 625	21 502 300	4 300 460	67 194,7	CPU 240 MHz, SRAM, code enh.	
7	64	64	4 float	ESP32-S2	82 033 950	22 059 000	59 974 950	8 567 850	133 872,7	CPU 240 MHz, SRAM max	
											26,51
50000	64	64	4 float	1070 Ti	370 275 688	358 650 936	11 624 752	232	3,6	мин.	0,91
50000	64	64	4 float	840M	1 610 978 210	1 467 190 510	143 232 276	2 865	44,8	мин.	1,34
50000	64	64	4 float	i7-6700K×8	664 152 739	591 620 420	71 788 591	1 436	22,4	мин.	2,92
50000	64	64	4 float	i7-4510U×4	943 533 870	629 503 394	306 911 811	6 138	95,9	мин.	2,88

Метка "мин." в таблице означает "минимум". Числа – в наносекундах. Вычисления на ESP32-S2 идут почти с нулевой дисперсией результатов, на Nvidia — это мелкие проценты, а на Intel/Windows есть заметный разброс, потому что система занимается своими делами тоже и некоторые процессы не только спонтанны, но и имеют высокий приоритет.

Немного про тест. Saola-1 - пятый участник соревнования. На желтом фоне - два Intel, на зеленом - два Nvidia (#CUDA). Колонки начинаются с размерностей куба, далее время расчета в наносекундах и энергозатраты. Последние две колонки - самые важные. По ним можно строить прогнозы на кубы иных размерностей.

Intel и ESP обсчитывали идентичный код. Для Intel на Java, для ESP на C++. Код ядра для Nvidia на C, вспомогательный на Java/C++/C. Он другой, но рассчитываемая функция и числа-параметры в кубе - идентичные. 8-я строчка для ESP (ячейка в обводке) - исключение, в том коде есть заметные изменения.

Все участники "мололи" одинаковые числа в формате #float32, исключительно для сравнимости результатов. Это - расточительный формат для #AI. От него ESP пострадал больше всех. А #float16 есть только для CUDA. На самом деле хватит и 8 бит на параметр, но требуются преобразования во float, что портит показатели по скорости для всех.

Мой рейтинг, результат кратко: Nvidia - лидер по скорости и энергозатратам на 1 выход, Intel - лидер по размеру куба, ESP - лидер по суммарным энергозатратам на единицу оборудования. Никого не обидел? 😆

А "хватит и 8 бит на параметр" — это абстрактно. На практике многое зависит от скорости выполнения трансформаций из беззнаковых байтов во #float32 (мне #float16 нравится, но это не совсем то, что мне нужно). Эта операция проходит быстро на современных Nvidia и Intel. На ESP она неэффективна. Во многом потому, что диапазон значений 0..255 плохо умножается по верхней границе, некрасиво. Диапазон 0..256 - прекрасно, но он вылазит за пределы байта. Использовать 0..128 можно, но вертятся мысли, что будет очень грубо. Будут реактивности в вычисленных движениях и прочих переменных управления. Сгодится для детской игрушки, но не для промышленного робота.

То есть медленно, но верно, приходим к 16 бит на параметр и 16-и битной арифметике. Для AI это не стыдно. Там на каждом шагу стоят нормировочные функции «активации», которые нивелируют всю точность расчётов. Ну и мы же не адронный коллайдер собираемся обслуживать этой малышкой.

Цифры от ESP в сравнении с конкурентами великоваты. Я перепроверил результаты несколько раз, но похоже на то, что тест не врёт. На картинке ниже результат по другому тесту. Он подтверждает тест на AI. "if-break" дают 20-30% надбавки к операциям над элементами массива, так что в целом цифры обоих тестов примерно сходятся. Также я перепроверил частоту процессора по регистровым параметрам. Это точно 240 МГц (PLL @480 МГц / 2).

```
if(    ! x[i]      ) break; // 130 ns
if( y[0] == x[i]   ) break; // 129 ns
if( y[i] == x[i]   ) break; // 133 ns
if( x[i]+y[i] < 0. ) break; // 370 ns
if( y[i]-x[i] < 0. ) break; // 445 ns
if( y[i]*x[i] < 0. ) break; // 409 ns
if( y[i]/x[i] < 0. ) break; // 1381 ns
```

Какие же мои ожидания от ESP32-S2? Это 8-10 мкс/выход матрицы (см. цветную таблицу выше). Такая скорость сделает WROVER полезным для моей AI. Я имею в виду скорость обработки матрицы предельного размера, которая упадёт до разумных 50 мс. 📉

По таблице, с учетом поправки на частоту процессора и потребляемую энергию, такая удельная производительность становится конкурентоспособной против i7-4510U. 📊 Заметьте, у Intel есть многоуровневый кэш и быстрая память. Но это по-прежнему плата заметных размеров с кучей внешней периферии и охлаждением, а у ESP32 это всё ещё только один модуль размером в монету

2 евро. И бригада таких модулей свободно разместится в детской игрушке-роботе, вместе с батареей, моторами, лампочками, динамиками и кучей разнообразных сенсоров. 😈

Ах уж эта инженерная работа! Никогда не знаешь результат заранее. Всю дорогу (к успеху) только тем и занимаешься, что решаешь проблемы (спускаешься с небес на землю). 😉 Похоже на то, что с плавающей арифметикой на этом чипе есть нюансы. Люди говорят, что нет на этом чипе FPU. То есть он есть в системе проектирования LX7 Xtensa, даже тензорные команды, но по факту... проверю ассемблером. А пока, вот критический "разбор плавающих полётов" 📘

Если информация подтвердится, то мой AI будет хранить байты и считать быстрыми целочисленными функциями. Как и 30 лет назад. История пошла по спирали. ⭕ В общем, паники нет. Есть варианты.

Xtensa выполняет аналогичные операции умножения и деления над #integer32 за 25-30 нс. Мне уже встречалась такая ситуация в теперь уже сильно далёком 1993-м, когда Intel очень медленно считал "плавающую" арифметику на 386-м процессоре. Спасали целочисленные операции. Может, и здесь надо поступить аналогично?

А получить ассемблерный код пока не удалось. Надо лезть в дебри кода CMake/Python, где гасится опция "-S" для компилятора. Просто так она не прикладывается и по факту не работает. По разговорам на форуме есть сложности правового характера по ассемблеру. Но это совсем другая история...

Часть 13. Антенна Wi-Fi

Wi-Fi... как много в этом звуке
Для сердца нашего слилось!
Как много в нем отозвалось!

Я думаю, Александр Сергеевич сегодня написал бы именно так, спасибо ему за гладкий слог о Москве. Но сегодня речь идет о славном маленьком компьютере с антенной.

Wi-Fi, как и другие протоколы связи публичного диапазона 2,4 ГГц, - это ключевая "фишка" процессорной линейки от Espressif Systems. Чем это хорошо для #IoT? Не надо прокладывать провода, не надо думать про разъёмы. Конечно, придётся подумать о чистоте радиоэфира от помех и конкурентов, но ведь ничего не бывает задаром, не так ли?

Вместо "печатной" антенны можно подключить внешнюю, "единорог" ловит и передаёт лучше. На картинке видно только место для разъёма антенны. Зоркий глаз и твёрдая рука установят разъём и переставят "нулевой" резистор. Но проще стразу заказать модуль с "печаткой" или разъемом, разница в цене - ноль.

Для первого знакомства я запустил стандартный пример для быстрого соединения с точкой доступа. Связь установилась мгновенно. Программа сообщила об использовании протокола WPA2-PSK,

упомянула bgn и номер канала, и сообщила о мощности сигнала rssi=-72 дБм. Процессор получил свой адрес через протокол DHCP, узнал маску сети и адрес маршрутизатора.

А в чём интрига? В том, что код флэшки занимает 600+ кБ! Это почти в два раза больше памяти SRAM на «борту». И это означает, что исполняемый код почти на половину будет выполняться из памяти FLASH. Фанфары замолкли.

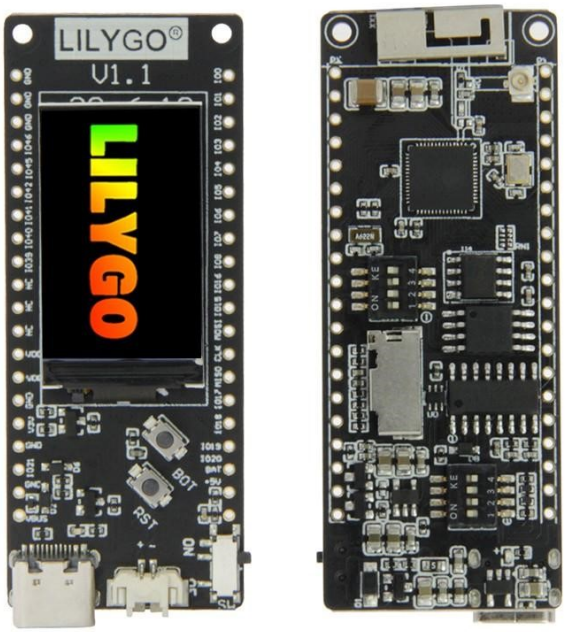
Учебный пример просканировал ближайшие точки доступа Wi-Fi. Кстати, ваш смартфон делает то же самое, как только вы вернулись домой, приехали на работу или просто идёте по улице. 📱 В эфире - толпа и шум, как на восточном базаре. Тест засёк 22 станции. Это на 11 (13) доступных частотных каналов! 🗼 И что удивительно, какие разнообразные региональные настройки замечены. Даже венгерские. 🤣 Но самые прикольные станции - без региональных настроек. По моей версии, это дает возможность устанавливать максимальную мощность передатчика. Наверное для того, чтобы соседям через улицу тоже хватило. ⚡

Диапазон 5 ГГц дает больший простор для связи. Но в ESP32-S2 он не поддерживается. У кого-то может возникнуть чувство, аналогичное мироощущению водителя "копейки" 🚗 : ты едешь по

городу, и довольно неплохо. Но не так быстро, как все остальные. Даже очередная встреча с той же компанией на следующем светофоре не сильно повышает самооценку.

Вот поэтому в новых чипах от ведущих производителей речь идёт о 5G и Wi-Fi версии 6. Меньше уже становится неприличным, точнее - немодным. А рынок MCU тесен и суров, как и остальной High-tech. 🤼 Ждём ответ от новых продуктов Espressif Systems.

Задать пароль к точке доступа в учебном примере легко. Заходим в "menuconfig" и впечатываем нужные буковки. 🤦 🤦 А сейчас представьте, что поменялся пароль Wi-Fi на цеховой точке доступа и сотня-другая маленьких коробочек оказалась в недоумении, куда подевался любимый сервер. Всё бы ничего, но у коробочек нет ни дисплея, ни клавиатуры, чтобы провести изменение привычным способом. Только сеть. Но к ней уже нет доступа. Упс... 😕 Повод для раздумий.



Картинка с сайта известного производителя

Даже если на устройстве есть touch screen, смена пароля требует ловкости рук и внимательности. Пользователи сетевых Wi-Fi принтеров знакомы с такой ситуацией. Помните, вас просили задать сложный пароль сети из 50 букв разного регистра, цифр и специальных знаков, всё в незапоминаемой комбинации. Чтобы вы не выдали секрет даже под средневековыми пытками. Нет, я совсем не кровожадный. 😈

Как вариант решения, в состав устройства, как на картинке, можно включить слот для SD карты, на которой будут храниться зашифрованные параметры доступа. Но что делать, если ключи шифрования MCU окажутся скомпрометированы? Заколдованный круг... 🤔

🎬 Wi-Fi этой малышки не быстр. В стандартных примерах нашелся проект для измерения скорости, со шлюзом на программу iperf. В таблице цифры по работе MCU как клиента. В среднем данные летят со скоростью 10 МБит/с. В режиме сервера - на 20% медленнее. На близком расстоянии, метра три от роутера, был замечен пик в 15 МБит/с, при том же среднем значении.

Interval, second	Transfer, Mbytes	Bandwidth, Mbits/sec
0,0- 3,0	3,34	9,3
3,0- 6,0	3,19	8,9
6,0- 9,0	3,34	9,3
9,0-12,0	3,62	10,1
12,0-15,0	3,71	10,4
15,0-18,0	3,61	10,1
18,0-21,0	3,25	9,1
21,0-24,0	3,53	9,9
24,0-27,0	3,36	9,4
27,0-30,0	3,33	9,3
30,0-33,0	3,49	9,8
33,0-36,0	3,60	10,1
36,0-39,0	3,53	9,9
39,0-42,0	3,48	9,7
42,0-45,0	3,59	10,0
45,0-48,0	3,44	9,6
48,0-51,0	3,56	10,0
51,0-54,0	3,47	9,7
54,0-57,0	3,33	9,3
57,0-60,0	3,47	9,7
0,0-60,0	69,20	9,7

📊 Много это или мало? С точки зрения гигабитного роутера — это до смешного медленно. С точки зрения устройства с пиковой потребляемой мощностью 1 Вт — это нормально. Почему?

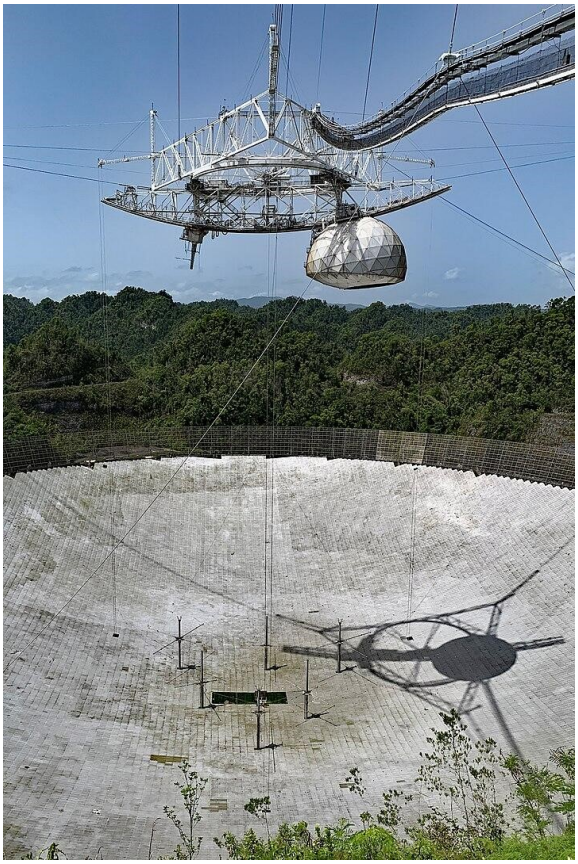
📡 Этот класс чипов изначально не задумывался как скоростной шлюз многоканального сбора данных. Для этой цели делают специальную аппаратуру. А на ESP32-S2 всего 46 внешних каналов, причем количество каналов одного типа - порядка 10. Сбор данных в принципе не может идти с частотой выше 240 МГц.

🤦 В идеале можно оцифровывать пяток каналов. Фильтровать грубые помехи, например вычислять среднее по 10 измерениям и дисперсию. и передавать уже сокращенный поток значимой информации. А быстрый сетевой сервер сможет принимать сотни таких Wi-Fi каналов.

Вообще-то роутер оценивает Saola-1 как устройство с TX rate 72 Мбит/с и RX rate 30 Мбит/с во время другого стандартного теста для протокола TCP, с участием эхо-сервера, быстро написанного 😉 на Java (это вам не на C++ пыхтеть и потеть!). При этом трафик на мелких объемах оценивается как 2 кБ/с (биты или байты - всегда загадка, если сокращения не расшифровываются).

Ирония обоих моих тестов в том, что у старенького (но бодренького) ноутбука, на котором трудится сервер, модуль Wi-Fi взят из бюджетной комплектации и больше 75 Мбит/с не даст никогда, проси не проси. Но цифры в 10-15 Мбит/с из первого теста всё равно маловаты для бросания бейсболок вверх. 🤸

Кстати, как-то при случае смотрел ролик по замене модуля Wi-Fi на материнской плате. Сие действие восхищает размахом AliExpress, мастерством твёрдой руки с паяльником и авантюрной смелостью автора. Но у него получилось! Как сегодня цитировали китайцы своего древнего мыслителя на CNC или CGTN, кто не мечтает круто, тот ничего и не достигает. Нашу версию про шампанское, я уверен, все знают тоже. 🍾



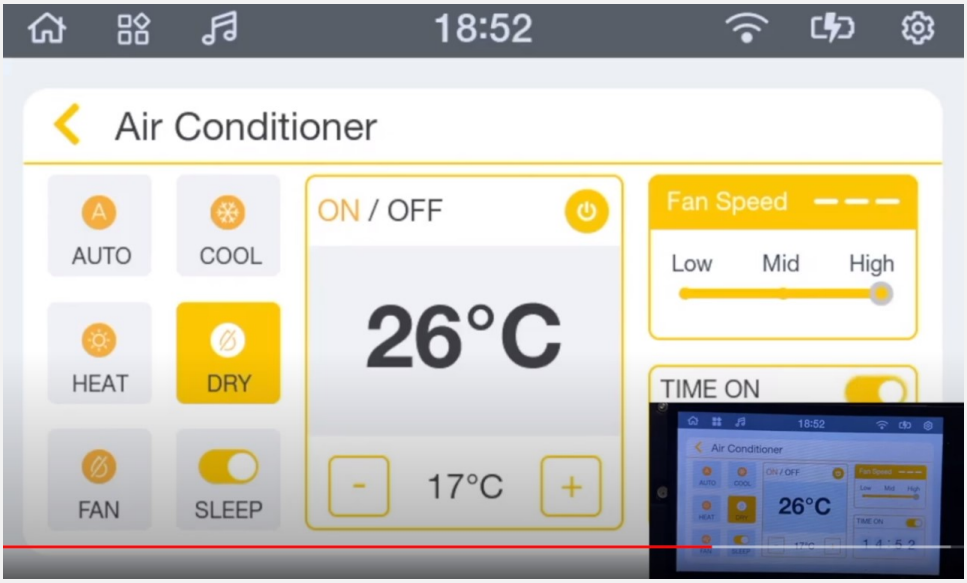
Глядя на "печатную" антенну, возникает закономерный вопрос: как далеко эти странные "проводочки" излучают сигнал? На память приходит антенна обсерватории Аресибо для прослушивания дальнего космоса, размером с небольшой вулкан, где-то в Латинской Америке. 🔔 Это - сила! А здесь? Устройство всего на 1 Вт потребляемой мощности.

Оказывается, надо только спросить. Добрые талантливые люди уже давно всё исследовали и выложили результаты в Интернет. На видео от «atomic14», по ссылке, - инструментальный тест модуля от Espressif Systems, в реальных условиях загородного дома. Если кратко, то "малышка" эффективно работает на 300 м. Lilygo, у них другая антенна, обещает в спецификациях те же самые дистанции. Я приятно удивлён, я ожидал раза в три меньше.

Но есть и сюрприз в мешке Деда Мороза! 🎅 Если покопаться с настройками и не гнаться за рекордами скорости передачи, то можно соединяться на 500 метров! Я считаю, это круто! 🏅 А что будет, если вещать через "единорога"? В общем, смотрите видео, там много ценной информации.

Часть 14. Окно во внутренний мир (неоконченное)

Большой экран и функции касания — это то, что сдвинуло с точки замирания всю отрасль мобильников, прямо революционно. Вот и здесь, в IoT, на примере для ESP32-S2, большой сенсорный экран - ключевой элемент #HDI.



Для меня только один вопрос в данном проекте — это скорость потока видеоданных, и что после этого остается для ЦП (он же один!) на непосредственно коммуникацию и датчики. Для примера, посчитайте пиксельную площадь телевизора в гостиной, умножьте на 24 бита и частоту обновления экрана. Не буду пугать жуткими цифрами, но именно они не дают стоять стандарту #HDMI на месте. Здесь экран меньше, но и процессор - не Xeon/GeForce.

Ну вот и всё на сегодня. Я думаю, продолжение последует. Поступают TFT и OLED мониторы. Да и новый AGI на подходе. Тьфу-тьфу-тьфу, что б не сглазить!

До встречи!

Опытный программист на Java, да и в C++ тоже не промах 😉
http://varankin.com/