Большая книга
из ESP32forth

версия 1.2 - 27 октября, 2023

Автор

- Marc PETREMANN petremann@arduino-forth.com

Соавтор

- Vaclav POSELT
- Thomas SCHREIN

Содержание

Введение

С 2019 года я управляю несколькими веб-сайтами, посвященными разработке языка FORTH для плат ARDUINO и ESP32, а также веб-версией eForth :

- ARDUINO : https://arduino-forth.com/
- ESP32 : https://esp32.arduino-forth.com/
- eForth web : https://eforth.arduino-forth.com/

Эти сайты доступны на двух языках: французском и английском. Каждый год я плачу за хостинг основного сайта **arduino-forth.com**.

Рано или поздно – и как можно позже – произойдет, что я больше не смогу обеспечивать устойчивость этих объектов. Последствием будет то, что информация, распространяемая этими сайтами, исчезнет.

Эта книга представляет собой компиляцию материалов с моих веб-сайтов. Он распространяется бесплатно из репозитория Github. Этот метод распространения обеспечит большую устойчивость, чем веб-сайты.

Кстати, если кто-то из читателей этих страниц захочет внести свой вклад, мы будем рады этому. :

- предлагать главы ;
- сообщать об ошибках или предлагать изменения ;
- помочь с переводом…

Помощь в переводе

Google Translate позволяет легко переводить тексты, но с ошибками. Вот и прошу помощи в исправлении переводов.

На практике я предоставляю уже переведенные главы в формате LibreOffice. Если вы хотите помочь с этими переводами, вашей ролью будет просто исправить и вернуть эти переводы.

Исправление главы занимает мало времени, от одного до нескольких часов.

Чтобы связаться со мной : petremann@arduino-forth.com

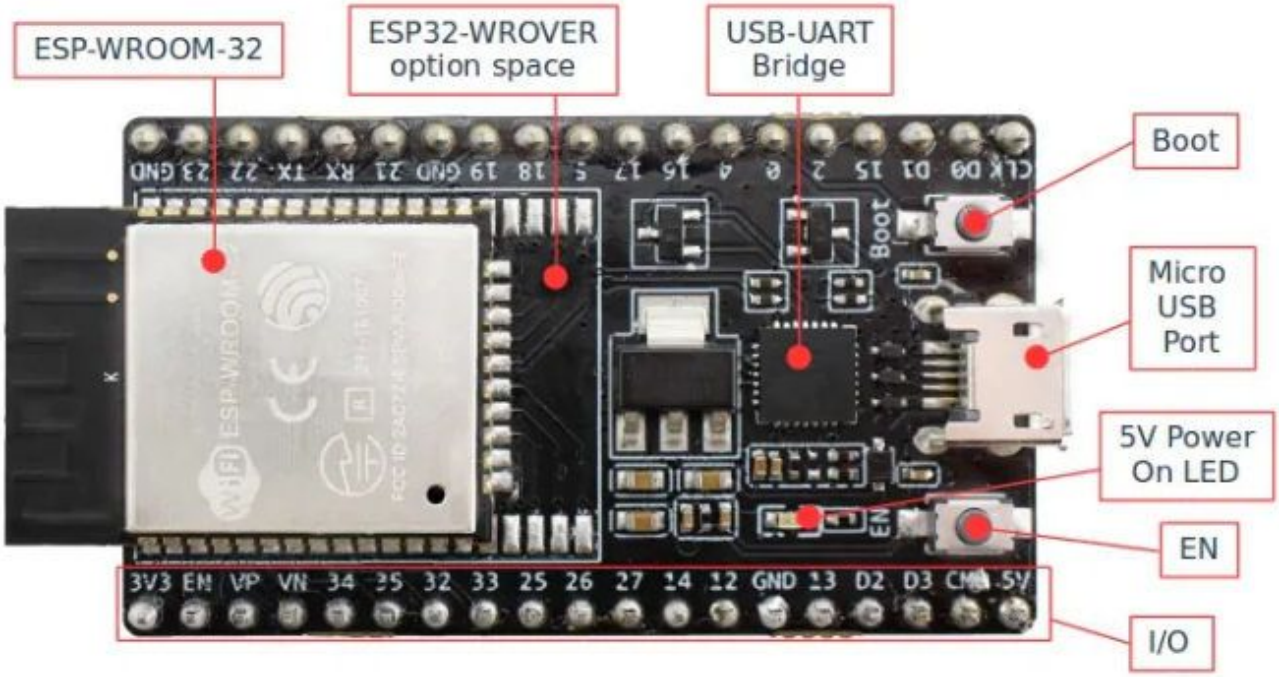
Обнаружение карты ESP32

Презентация

Плата ESP32 не является платой ARDUINO. Однако инструменты разработки используют определенные элементы экосистемы ARDUINO, такие как ARDUINO IDE.

Сильные стороны

По количеству доступных портов карта ESP32 находится между ARDUINO NANO и ARDUINO UNO. Базовая модель имеет 38 разъемов:



Устройства ESP32 включают в себя:

- 18 каналов аналого-цифрового преобразователя (АЦП)
- 3 интерфейса SPI
- 3 интерфейса UART
- 2 интерфейса I2C
- 16 выходных каналов ШИМ
- 2 цифро-аналоговых преобразователя (ЦАП)
- 2 интерфейса I2S
- 10 емкостных сенсорных GPIO

Функции ADC (аналогово-цифрового преобразователя) и ADC (цифро-аналогового преобразователя) назначены на определенные статические контакты. Однако вы можете решить, какие контакты будут UART, I2C, SPI, PWM и т. д. Вам просто нужно назначить их в коде. Это возможно благодаря функции мультиплексирования чипа ESP32.

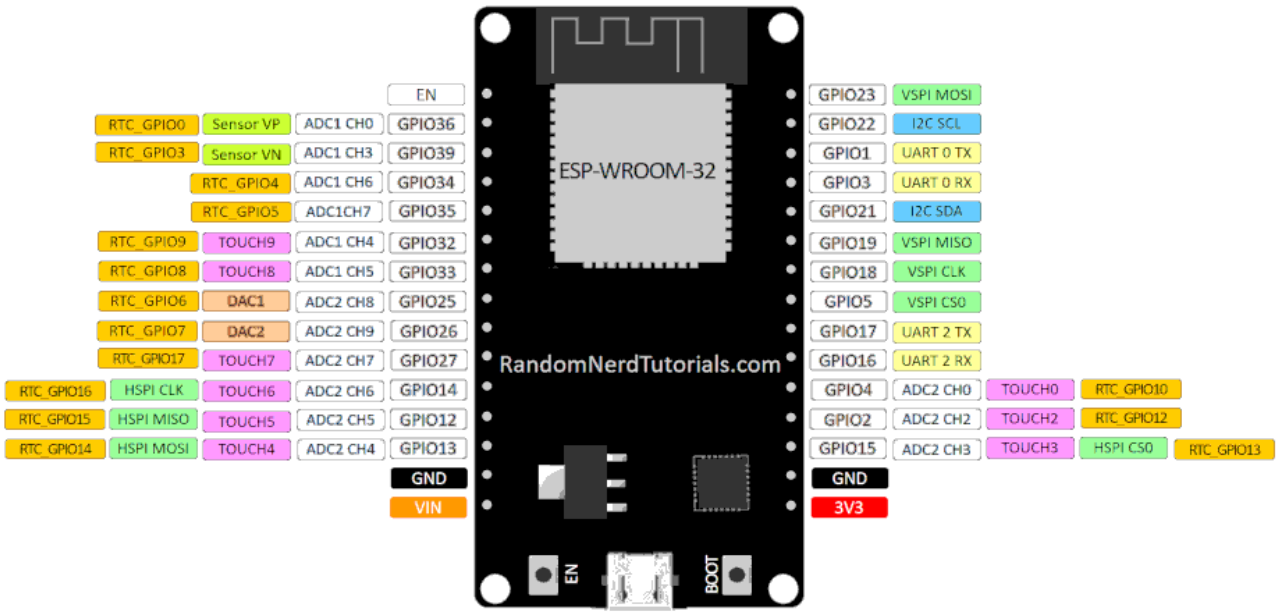
Большинство разъемов имеют многократное использование.

Но что отличает плату ESP32, так это то, что она в стандартной комплектации оснащена поддержкой Wi-Fi и Bluetooth, что платы ARDUINO предлагают только в виде расширений.

Входы/выходы GPIO на ESP32

Здесь на фотографии карта ESP32, на которой мы объясним роль различных входов/выходов GPIO:

Положение и количество входов/выходов GPIO могут меняться в зависимости от марки карты. В этом случае подлинными являются только указания, представленные на физической карте. На фото нижний ряд слева направо: CLK, SD0, SD1, G15, G2, G0, G4, G16.....G22, G23, GND.



На этой диаграмме мы видим, что нижний ряд начинается с 3V3, а на фотографии этот ввод-вывод находится в конце верхнего ряда. Поэтому очень важно не полагаться на схему, а дважды проверить правильность подключения периферийных устройств и компонентов на физической карте ESP32.

Платы разработки на базе ESP32 обычно имеют 33 контакта, кроме контактов источника питания. Некоторые контакты GPIO имеют несколько специфических функций:

GPIO	Возможные имена
6	SCK/CLK
7	SCK/CLK
8	SDO/SD0
9	SDI/SD1
10	SHD/SD2
11	CSC/CMD

Если ваша карта ESP32 имеет входы/выходы GPIO6, GPIO7, GPIO8, GPIO9, GPIO10, GPIO11, вам определенно не следует их использовать, поскольку они подключены к флэш-памяти ESP32. Если вы их используете, ESP32 не будет работать.

GPIO1(TX0) и GPIO3(RX0) ввода-вывода используются для связи с компьютером по UART через порт USB. Если вы ими воспользуетесь, то больше не сможете общаться с картой.

GPIO36(VP), GPIO39(VN), GPIO34, GPIO35 ввода-вывода можно использовать только как вход. У них также нет встроенных внутренних подтягивающих и понижающих резисторов.

Разъем EN позволяет контролировать состояние зажигания ESP32 через внешний провод. Он подключен к кнопке EN на карте. Когда ESP32 включен, он составляет 3,3 В. Если мы подключим этот контакт к земле, ESP32 выключится. Вы можете использовать его, когда ESP32 находится в коробке и вы хотите иметь возможность включать/выключать его с помощью переключателя.

Периферийные устройства ESP32

Для взаимодействия с модулями, датчиками или электронными схемами ESP32, как и любой микроконтроллер, имеет множество периферийных устройств. Их больше, чем на классической плате Arduino.

ESP32 имеет следующие периферийные устройства:

- 3 интерфейса UART
- 2 интерфейса I2C
- 3 интерфейса SPI
- 16 ШИМ-выходов
- 10 емкостных датчиков
- 18 аналоговых входов (АЦП)
- 2 выхода ЦАП

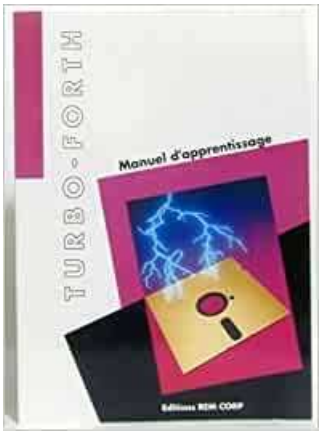
Некоторые периферийные устройства уже используются ESP32 во время его основной работы. Таким образом, для каждого устройства существует меньше возможных интерфейсов.

Зачем программировать на языке FORTH на ESP32?

Преамбула

Я программирую на FORTH с 1983 года. Я прекратил программировать на FORTH в 1996 году. Но я никогда не переставал следить за развитием этого языка. В 2019 году я возобновил программирование на ARDUINO с помощью FlashForth, затем ESP32forth.

Я являюсь соавтором нескольких книг о языке FORTH:



- Introduction au ZX-FORTH (ed Eyrolles - 1984 - ASIN:B0014IGOXO)
- Tours de FORTH (ed Eyrolles - 1985 - ISBN-13: 978-2212082258)
- FORTH pour CP/M et MSDOS (ed Loisitech - 1986)
- TURBO-Forth, manuel d'apprentissage (ed Rem CORP - 1990)
- TURBO-Forth, guide de référence (ed Rem CORP - 1991)

Программирование на языке FORTH всегда было хобби, пока в 1992 году со мной не связался менеджер компании, работавшей субподрядчиком в автомобильной промышленности. У них была забота о разработке программного обеспечения на языке C. Им нужно было заказать промышленный автомат.

Два разработчика программного обеспечения этой компании программировали на языке C: точнее, TURBO-C от Borland. И их код не мог быть достаточно компактным и быстрым, чтобы уместиться в 64 килобайта оперативной памяти. Это был 1992 год, и расширения типа флэш-памяти не существовало. В эти 64 КБ ОЗУ нам предстояло уместить MS-DOS 3.0 и приложение!

В течение месяца разработчики языка C решали проблему во всех направлениях, даже занимаясь обратным проектированием с помощью SOURCER (дизассемблера), чтобы исключить несущественные части исполняемого кода.

Я проанализировал поставленную передо мной проблему. Начав с нуля, я в одиночку за неделю создал идеально работающий прототип, соответствующий спецификациям. За три года, с 1992 по 1995 год, я создал множество версий этого приложения, которые использовались на сборочных конвейерах нескольких автопроизводителей.

Границы между языком и применением

Все языки программирования делятся следующим образом:

- интерпретатор и исполняемый исходный код: BASIC, PHP, MySQL, JavaScript и т. д. Приложение содержится в одном или нескольких файлах, которые будут интерпретироваться при необходимости. В системе должен постоянно находиться интерпретатор, выполняющий исходный код;
- компилятор и/или ассемблер: C, Java и т. д. Некоторые компиляторы генерируют собственный код, то есть исполняемый конкретно в системе. Другие, например Java, компилируют исполняемый код на виртуальной Java-машине.

Язык FORTH является исключением. Оно включает :

- интерпретатор, способный выполнить любое слово языка FORTH
- компилятор, способный расширять словарь слов FORTH.

Что такое ФОРТ-слово?

Слово FORTH обозначает любое словарное выражение, состоящее из символов ASCII и пригодное для интерпретации и/или компиляции: **`слова`** позволяют перечислить все слова в словаре FORTH.

Определенные слова FORTH можно использовать только при компиляции: например, **`if else then`**.

Основной принцип языка FORTH заключается в том, что мы не создаем приложение. В FORTH мы расширяем словарь! Каждое новое слово, которое вы определяете, будет такой же частью словаря FORTH, как и все слова, заранее определенные при запуске FORTH. Пример :

```
: typeToLoRa ( -- )
    0 echo !    \ отключает эхо дисплея терминала
    ['] serial2-type is type
  ;
: typeToTerm ( -- )
    ['] default-type is type
    -1 echo !   \ активирует эхо дисплея терминала
  ;
```

Мы создаем два новых слова: **`typeToLoRa`** и **`typeToTerm`** , которые дополнят словарь заранее определенных слов.

Слово — это функция?

Да и нет. На самом деле слово может быть константой, переменной, функцией... Вот в нашем примере следующая последовательность:

```
: typeToLoRa ...код... ;
```

будет иметь эквивалент на языке C:

```
void typeToLoRa() { ...код... }
```

В языке FORTH нет границ между языком и приложением.

В FORTH, как и в языке C, вы можете использовать любое слово, уже определенное в определении нового слова.

Да, но тогда почему FORTH, а не C?

Я ожидал этого вопроса.

В языке C доступ к функции возможен только через основную функцию `main()` . Если эта функция объединяет несколько дополнительных функций, становится сложно найти ошибку параметра в случае неисправности программы.

Напротив, с помощью FORTH можно выполнить - через интерпретатор - любое слово, заранее определенное или определенное вами, без необходимости проходить через главное слово программы.

Интерпретатор FORTH сразу доступен на карте ESP32 через программу терминального типа и соединение USB между картой ESP32 и ПК.

Компиляция программ, написанных на языке FORTH, осуществляется на карте ESP32, а не на ПК. Нет загрузки. Пример :

```
: >gray ( n -- n' )
    dup 2/ xor      \ n' = n xor (1 логический сдвиг вправо)
  ;
```

Это определение передается путем копирования/вставки в терминал. Интерпретатор/компилятор FORTH проанализирует поток и скомпилирует новое слово `>gray` .

В определении `>gray` мы видим последовательность `dup 2/ xor` . Чтобы проверить эту последовательность, просто введите ее в терминал. Чтобы запустить `>gray` , просто введите это слово в терминале, которому предшествует число для преобразования.

Язык FORTH по сравнению с языком C

Это моя наименее любимая часть. Мне не нравится сравнивать язык FORTH с языком C. Но, поскольку почти все разработчики используют язык C, я попробую это упражнение.

Вот тест с `if()` на языке C:

```
if(j > 13){                 // Если все биты получены
    rc5_ok = 1;             // Процесс декодирования в порядке
```

```
    detachInterrupt(0);  // Запретить внешнее прерывание (INT0)
    return;
}
```

Проверьте **if** на языке FORTH (фрагмент кода) :

```
var-j @ 13 >        \ Если все биты получены
    if
        1 rc5_ok !  \ Процесс декодирования в порядке
        di          \ Отключить внешнее прерывание (INT0)
        exit
    then
```

Вот инициализация регистров на языке C:

```
void setup() {
  //  Конфигурация модуля Timer1
  TCCR1A = 0;
  TCCR1B = 0;          // Отключаем модуль Timer1
  TCNT1  = 0;          // Устанавливаем значение предварительной
                       // загрузки Timer1 на 0 (сброс)
  TIMSK1 = 1;          // включаем прерывание переполнения Timer1
}
```

То же определение на языке FORTH :

```
: setup ( -- )
  \ Configuration du module Timer1
  0 TCCR1A !
  0 TCCR1B !     \ Отключаем модуль Timer1
  0 TCNT1  !     \ Устанавливаем значение предварительной
                 \ загрузки Timer1 на 0 (сброс)
  1 TIMSK1 !     \ включаем прерывание переполнения Timer1
;
```

Что FORTH позволяет делать по сравнению с языком C

Мы понимаем, что FORTH сразу дает доступ ко всем словам словаря, но не только. Через интерпретатор мы также получаем доступ ко всей памяти карты ESP32. Подключитесь к плате ARDUINO, на которой установлен FlashForth, затем просто введите:

```
hex here 100 dump
```

Вы должны найти это на экране терминала:

```
3FFEE964                        DF DF 29 27 6F 59 2B 42 FA CF 9B 84
3FFEE970              39 4E 35 F7 78 FB D2 2C A0 AD 5A AF 7C 14 E3 52
3FFEE980              77 0C 67 CE 53 DE E9 9F 9A 49 AB F7 BC 64 AE E6
3FFEE990              3A DF 1C BB FE B7 C2 73 18 A6 A5 3F A4 68 B5 69
3FFEE9A0              F9 54 68 D9 4D 7C 96 4D 66 9A 02 BF 33 46 46 45
3FFEE9B0              45 39 33 33 2F 0D 08 18 BF 95 AF 87 AC D0 C7 5D
3FFEE9C0              F2 99 B6 43 DF 19 C9 74 10 BD 8C AE 5A 7F 13 F1
3FFEE9D0              9E 00 3D 6F 7F 74 2A 2B 52 2D F4 01 2D 7D B5 1C
3FFEE9E0              4A 88 88 B5 2D BE B1 38 57 79 B2 66 11 2D A1 76
```

```
3FFEE9F0              F6 68 1F 71 37 9E C1 82 43 A6 A4 9A 57 5D AC 9A
3FFEEA00              4C AD 03 F1 F8 AF 2E 1A B4 67 9C 71 25 98 E1 A0
3FFEEA10              E6 29 EE 2D EF 6F C7 06 10 E0 33 4A E1 57 58 60
3FFEEA20              08 74 C6 70 BD 70 FE 01 5D 9D 00 9E F7 B7 E0 CA
3FFEEA30              72 6E 49 16 0E 7C 3F 23 11 8D 66 55 EC F6 18 01
3FFEEA40              20 E7 48 63 D1 FB 56 77 3E 9A 53 7D B6 A7 A5 AB
3FFEEA50              EA 65 F8 21 3D BA 54 10 06 16 E6 9E 23 CA 87 25
3FFEEA60              E7 D7 C4 45
```

Это соответствует содержимому флэш-памяти.

А язык C не мог этого сделать?

Да, но не так просто и интерактивно, как в языке FORTH.

Давайте рассмотрим еще один случай, подчеркивающий необычайную компактность языка FORTH...

Но почему именно стек, а не переменные?

Стек — это механизм, реализованный практически во всех микроконтроллерах и микропроцессорах. Даже язык C использует стек, но у вас нет к нему доступа.

Только язык FORTH дает полный доступ к стеку данных. Например, чтобы сложить, мы складываем два значения, выполняем сложение, отображаем результат: `2 5 +` . отображает `7` .

Это немного дестабилизирует, но когда вы поймете механизм стека данных, вы высоко оцените его огромную эффективность.

Стек данных позволяет передавать данные между словами FORTH гораздо быстрее, чем при обработке переменных, как в языке C или любом другом языке, использующем переменные.

Вы убеждены?

Лично я сомневаюсь, что эта единственная глава безвозвратно вернет вас к программированию на языке FORTH. При попытке освоить карты ESP32 у вас есть два варианта:

- программировать на языке C и использовать многочисленные доступные библиотеки. Но вы останетесь привязанными к возможностям этих библиотек. Адаптация кодов к языку C требует реальных знаний программирования на языке C и владения архитектурой карт ESP32. Разработка сложных программ всегда будет вызывать беспокойство.
- попробуйте приключение FORTH и исследуйте новый захватывающий мир. Конечно, это будет непросто. Вам нужно будет глубоко понять архитектуру карт ESP32, регистры и флаги регистров. Взамен вы получите доступ к программам, идеально подходящим для ваших проектов.

Существуют ли профессиональные приложения, написанные на FORTH?

О, да! Начиная с космического телескопа ХАББЛ, некоторые компоненты которого были написаны на языке FORTH.

Немецкая компания TGV ICE (Intercity y Express) использует процессоры RTX2000 для управления двигателями через силовые полупроводники. Машинным языком процессора RTX2000 является язык FORTH.

Тот же процессор RTX2000 использовался для зонда Philae, который пытался приземлиться на комету.

Выбор языка FORTH для профессиональных приложений оказывается интересным, если рассматривать каждое слово как черный ящик. Каждое слово должно быть простым, поэтому иметь достаточно краткое определение и зависеть от нескольких параметров.

На этапе отладки становится легко протестировать все возможные значения, обрабатываемые этим словом. Однажды сделанное абсолютно надежным, это слово становится черным ящиком, то есть функцией, в правильном функционировании которой мы имеем неограниченную уверенность. От слова к слову, на FORTH легче сделать сложную программу надежной, чем на любом другом языке программирования.

Но если нам не хватает строгости, если мы построим газовые заводы, то также очень легко получить приложение, которое будет работать плохо, а то и вовсе провалиться ФОРТ!

Наконец, на языке FORTH можно писать определяемые вами слова на любом человеческом языке. Однако используемые символы ограничены набором символов ASCII от 33 до 127. Вот как мы могли бы символически переписать слова **high** и **low** :

```
\Активный контакт порта, не меняйте другие.
: __/ ( pinmask portadr -- )
    mset
  ;
\ Отключите один вывод порта, не меняйте остальные.
: \__ ( pinmask portadr -- )
    mclr
  ;
```

С этого момента для включения светодиода можно набрать:

```
_0_ __/             \ Светодиодные фонари
```

Да! Последовательность `_0_ __/` написана на языке FORTH!

С ESP32forth в вашем распоряжении все символы, которые могут составить слово FORTH:

```
~}|{zyxwvutsrqponmlkjihgfedcba`_
^]\[ZYXWVUTSRQPONMLKJIHGFEDCBA@?
>=<;:9876543210/.-,+*)('&%$#"!
```

Хорошее программирование.

Настоящий 32-битный FORTH с ESP32Forth.

ESP32Forth — это настоящий 32-битный FORTH. Что это значит ?

Язык FORTH благоприятствует манипулированию целочисленными значениями. Этими значениями могут быть литеральные значения, адреса памяти, содержимое регистров и т. д.

Значения в стеке данных

Когда ESP32Forth запускается, интерпретатор FORTH доступен. Если вы введете любое число, оно будет помещено в стек как 32-битное целое число:

```
35
```

Если мы сложим другое значение, оно тоже будет сложено. Предыдущее значение будет сдвинуто на одну позицию вниз:

```
45
```

Чтобы сложить эти два значения, мы используем слово здесь `+` :

```
+
```

Наши два 32-битных целочисленных значения складываются и результат сбрасывается в стек. Чтобы отобразить этот результат, мы будем использовать слово `.` :

```
. \ отображает 80
```

В языке FORTH мы можем сконцентрировать все эти операции в одной строке:

```
35 45 +. \ показать 80
```

В отличие от языка C, мы не определяем тип **int8** , **int16** или **int32** .

В ESP32Forth символ ASCII будет обозначаться 32-битным целым числом, но значение которого будет ограничено [32..256[. Пример :

```
67 emit \ дисплей C
```

Значения в памяти

ESP32Forth позволяет определять константы и переменные. Их контент всегда будет в 32-битном формате. Но бывают ситуации, когда это нас не обязательно устраивает. Давайте рассмотрим простой пример: определение алфавита азбуки Морзе. Нам нужно всего несколько байт:

- один для определения количества знаков азбуки Морзе
- один или несколько байтов для каждой буквы азбуки Морзе

```
create mA ( -- addr )
    2 c,
    char . c,   char - c,

create mB ( -- addr )
    4 c,
    char - c,   char . c,   char . c,   char . c,

create mC ( -- addr )
    4 c,
    char - c,   char . c,   char - c,   char . c,
```

Здесь мы определяем только 3 слова: **mA** , **mB** и **mC** . В каждом слове хранится несколько байт. Вопрос в том, как мы получим информацию, содержащуюся в этих словах?

Выполнение одного из этих слов сохраняет 32-битное значение, значение, которое соответствует адресу памяти, где мы сохранили информацию азбуки Морзе. Именно слово **c@** мы будем использовать для извлечения азбуки Морзе из каждой буквы:

```
mA c@ .  \ отображает 2
mB c@ .  \ отображает 4
```

Первый извлеченный таким образом байт будет использоваться для управления циклом отображения азбуки Морзе буквы:

```
: .morse ( addr -- )
    dup 1+ swap c@ 0 do
        dup i + c@ emit
    loop
    drop
  ;
mA .morse   \ отображает .-
mB .morse   \ отображает -...
mC .morse   \ отображает -.-.
```

Есть множество, конечно, более элегантных примеров. Здесь показано, как манипулировать 8-битными значениями, нашими байтами, пока мы используем эти байты в 32-битном стеке.

Обработка текста в зависимости от размера или типа данных

Во всех других языках есть общее слово, такое как **echo** (в PHP), которое отображает любой тип данных. Будь то целое, вещественное или строковое значение, мы всегда используем одно и то же слово. Пример на языке PHP:

```
$bread = "Pain cuit";
$price = 2.30;
echo $bread . " : " . $price;
// отображает   Pain cuit: 2.30
```

Для всех программистов такой способ работы является СТАНДАРТОМ! Так как же FORTH реализовал бы этот пример на PHP?

```
: pain s" Pain cuit" ;
: prix s" 2.30" ;
pain type   s" : " type    prix type
\ отображает   Pain cuit: 2.30
```

тип слова сообщает нам, что мы только что обработали строку символов.

Там, где PHP (или любой другой язык) имеет общую функцию и анализатор, FORTH компенсирует это одним типом данных, но адаптированными методами обработки, которые сообщают нам о природе обрабатываемых данных.

Вот совершенно тривиальный случай для FORTH, отображающий количество секунд в формате ЧЧ:ММ:СС:

```
: :##
    #  6 base !
    #  decimal
    [char] : hold
  ;
: .hms ( n -- )
    <# :## :## # # #>  type
  ;
4225 .hms  \ отображает: 01:10:25
```

Мне нравится этот пример, потому что на сегодняшний день **НИ ОДИН ДРУГОЙ ЯЗЫК ПРОГРАММИРОВАНИЯ** не способен выполнить преобразование ЧЧ:ММ:СС так элегантно и лаконично.

Вы поняли, секрет ФОРТА – в его словаре.

Заключение

В FORTH нет типизации данных. Все данные проходят через стек данных. Каждая позиция в стеке ВСЕГДА представляет собой 32-битное целое число!

Это все, что нужно знать.

Сторонники гиперструктурированных и многословных языков, таких как C или Java, наверняка будут кричать о ереси. И здесь я позволю себе на них ответить: зачем вам вводить свои данные?

Потому что именно в этой простоте и заключается сила FORTH: единый стек данных в нетипизированном формате и очень простые операции.

И я собираюсь показать вам то, чего не могут сделать многие другие языки программирования, — определить новые слова определения:

```
: morse: ( comp: c -- | exec -- )
    create
        c,
    does>
        dup 1+ swap c@ 0 do
            dup i + c@ emit
        loop
        drop space
  ;
2 morse: mA     char . c,   char - c,
4 morse: mB     char - c,   char . c,   char . c,   char . c,
4 morse: mC     char - c,   char . c,   char - c,   char . c,
mA mB mC    \ отображает .- -... -.-.
```

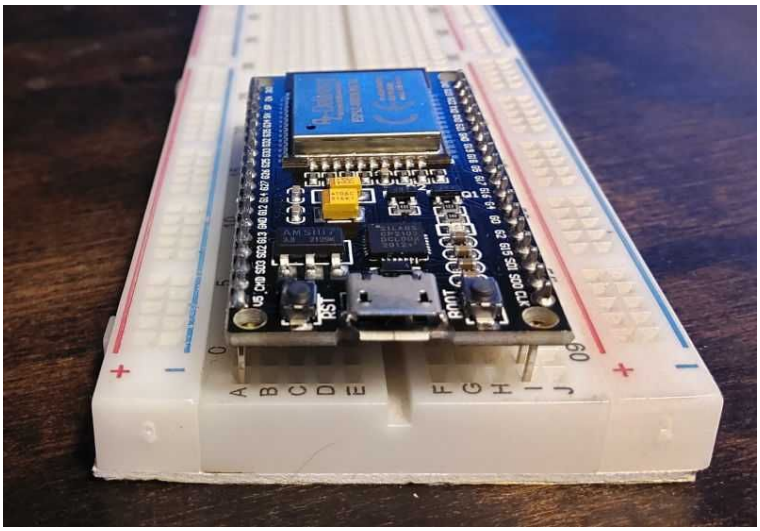
Здесь слово **morse:** стало определяющим словом, точно так же, как **constant** или **variable** ...

Потому что FORTH — это больше, чем язык программирования. Это метаязык, то есть язык для создания собственного языка программирования....

Адаптируйте макеты к плате ESP32.

Тестовые пластины для ESP32

Вы только что получили карты ESP32. И первый неприятный сюрприз: эта карта очень плохо помещается на тестовой плате:

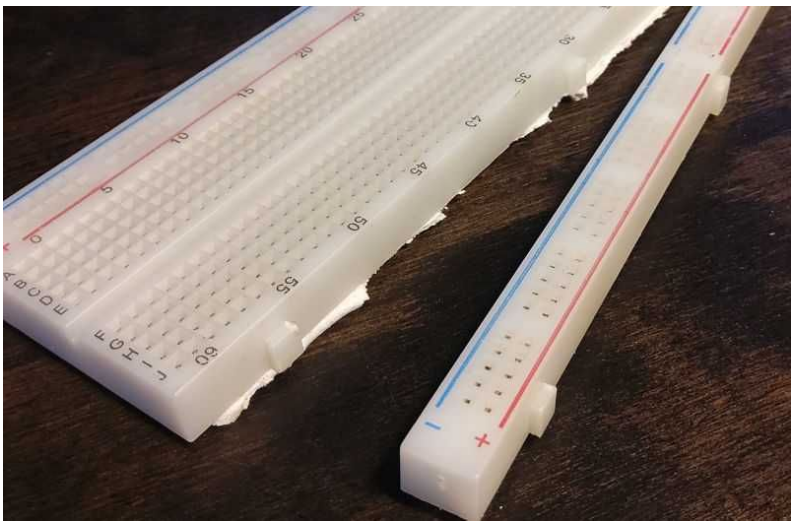


Не существует макета, специально подходящего для плат ESP32.

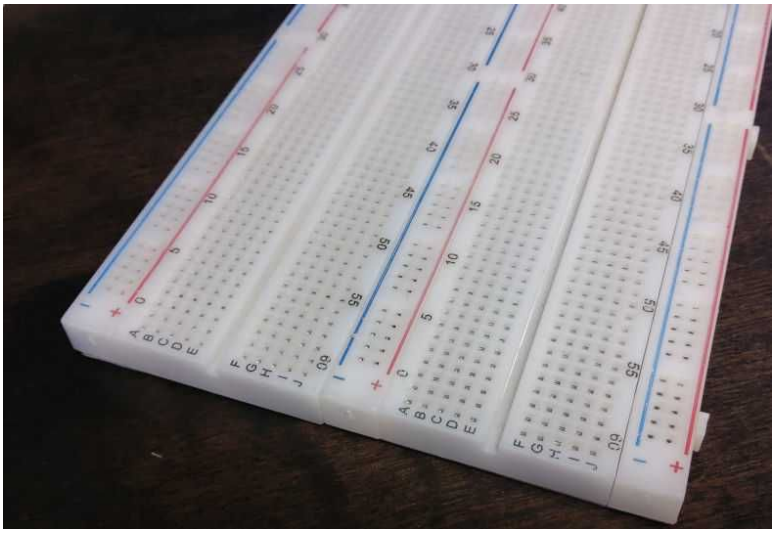
Создайте макет, подходящий для платы ESP32.

Мы собираемся построить свою собственную тестовую пластину. Для этого необходимо иметь две одинаковые тестовые пластины.

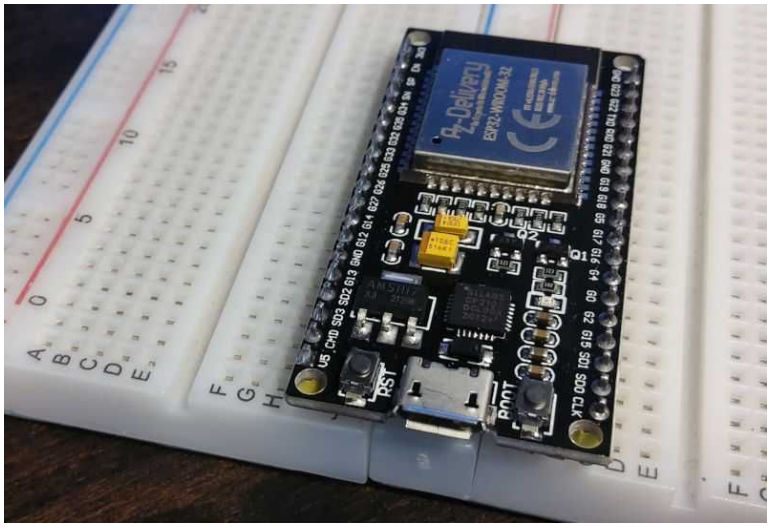
На одной из пластин снимем линию электропередачи. Для этого воспользуйтесь резаком и разрежьте снизу. Вы сможете отделить эту линию электропередачи следующим образом:



Затем мы можем собрать всю карту заново с помощью этой карты. У вас есть стропила по бокам тестовых пластин, чтобы соединить их вместе:



И вот! Теперь мы можем разместить нашу карту ESP32:



Порты ввода-вывода можно без труда расширить.

Генератор случайных чисел

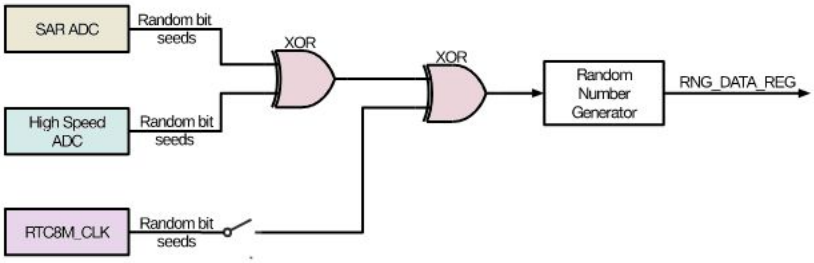
Характеристика

Генератор случайных чисел генерирует настоящие случайные числа, то есть случайное число, сгенерированное в результате физического процесса, а не с помощью алгоритма. Ни одно число, сгенерированное в указанном диапазоне, не может появиться с большей или меньшей вероятностью, чем любое другое число.

Каждое 32-битное значение, которое система считывает из регистра RNG_DATA_REG генератора случайных чисел, является истинным случайным числом. Эти истинные случайные числа генерируются на основе теплового шума в системе и асинхронного перекоса часов.

Тепловой шум исходит от высокоскоростного АЦП или АЦП SAR, или от того и другого. Всякий раз, когда активируется высокоскоростной АЦП или АЦП SAR, потоки битов будут генерироваться и поступать в генератор случайных чисел через логический элемент XOR в качестве случайных начальных значений.

Когда для цифрового ядра включена тактовая частота RTC8M_CLK, генератор случайных чисел также будет выбирать RTC8M_CLK (8 МГц) в качестве случайного двоичного начального числа. RTC8M_CLK — это асинхронный источник синхронизации, который увеличивает энтропию ГСЧ за счет метастабильности схемы. Однако для обеспечения максимальной энтропии также рекомендуется всегда включать источник АЦП.



При наличии шума от АЦП SAR на генератор случайных чисел подается энтропия размером 2 бита в такте RTC8M_CLK (8 МГц), которая генерируется внутренним RC-генератором (более подробную информацию см. в главе «Сброс и синхронизация»).). Таким образом, для получения максимальной энтропии чтение регистра **RNG_DATA_REG желательно производить** с максимальной частотой 500 кГц.

При появлении шума от высокоскоростного АЦП в генератор случайных чисел подается 2-битная энтропия в тактовом цикле APB, который обычно составляет 80

МГц. Таким образом, для получения максимальной энтропии чтение регистра `RNG_DATA_REG` желательно производить с максимальной скоростью 5 МГц.

Образец данных объемом 2 ГБ, который считывается из генератора случайных чисел на частоте 5 МГц при включенном только высокоскоростном АЦП, был протестирован с использованием набора тестов случайных чисел Dieharder (версия 3.31.1). Образец прошел все испытания.

Процедура программирования

При использовании генератора случайных чисел убедитесь, что разрешен как минимум SAR ADC, High Speed ADC или RTC8M_CLK. В противном случае будут возвращены псевдослучайные числа.

- SAR ADC можно активировать с помощью контроллера DIG ADC.
- Высокоскоростной АЦП автоматически включается при включении модулей Wi-Fi или Bluetooth.
- RTC8M_CLK включается установкой бита RTC_CNTL_DIG_CLK8M_EN в регистре RTC_CNTL_CLK_CONF_REG.

При использовании генератора случайных чисел читайте регистр `RNG_DATA_REG` несколько раз, пока не будет сгенерировано достаточно случайных чисел.

Name	Description	Address	Access
RNG_DATA_REG	Random number data	$3FF75144	RO

```
\ Данные случайных чисел
$3FF75144 constant RNG_DATA_REG

\ получаем 32-битное случайное число b=число
: rnd  ( -- x )
    RNG_DATA_REG L@
  ;

\ получить случайное число в интервале [0..n-1]
: random ( n -- 0..n-1 )
    rnd swap mod
  ;
```

Функция RND в ассемблере XTENSA

Начиная с версии 7.0.7.4, ESP32forth имеет ассемблер XTENSA. Наше `rnd` слово можно переписать на ассемблере XTENSA:

```
forth definitions
asm xtensa
$3FF75144 constant RNG_DATA_REG
```

```
code myRND ( -- [addr] )
    a1 32           ENTRY,
    a8 RNG_DATA_REG L32R,       \ a8 = RNG_DATA_REG
    a9 a8 0         L32I.N,     \ a9 = [a8]
    a9          arPUSH,         \ push a9 on stack
                    RETW.N,
end-code
```

Подробное содержание словарей ESP32forth

ESP32forth предоставляет множество словарей:

- **FORTH** – основной словарь;
- определенные словари используются для внутренней механики ESP32Forth, например, **Internals** , **asm…**
- многие словари позволяют управлять определенными портами или аксессуарами, такими как **Bluetooth** , **oled** , **SPI** , **Wi-Fi** , **провод…**

Здесь вы найдете список всех слов, определенных в этих разных словарях.
Некоторые слова представлены цветной ссылкой:

align — обычное слово FORTH;

CONSTANT – определяющее слово;

begin отмечает структуру управления;

key — слово отложенного выполнения;

LED это слово, определяемое **константой** , **переменной** или **значением** ;

registers отмечает словарный запас.

Слова словаря **FORTH** отображаются в алфавитном порядке. В других словарях слова представлены в порядке их отображения.

Version v 7.0.7.15

FORTH

-	-rot	,	;	:	:noname	!
?	?do	?dup	.	."	.s	'
(local)	[	[']	[char]	[ELSE]	[IF]	[THEN]
]	{	{	}transfer	@	*	*/
*/MOD	/	/mod	#	#!	#>	#fs
#s	#tib	+	+!	+loop	+to	<
<#	<=	<>	=	>	>=	>BODY
>flags	>flags&	>in	>link	>link&	>name	>params
>R	>size	0<	0<>	0=	1-	1/F
1+	2!	2@	2*	2/	2drop	2dup
4*	4/	abort	abort"	abs	accept	adc
afliteral	aft	again	ahead	align	aligned	allocate
allot	also	analogRead	AND	ansi	ARSHIFT	asm
assert	at-xy	base	begin	bg	BIN	binary
bl	blank	block	block-fid	block-id	buffer	bye
c,	C!	C@	CASE	cat	catch	CELL
cell/	cell+	cells	char	CLOSE-DIR	CLOSE-FILE	cmove

cmove>	CONSTANT	context	copy	cp	cr	CREATE
CREATE-FILE	current	dacWrite	decimal	default-key	default-key?	
default-type		default-use	defer	DEFINED?	definitions	DELETE-FILE
depth	digitalRead	digitalWrite		do	DOES>	DROP
dump	dump-file	DUP	duty	echo	editor	else
emit	empty-buffers		ENDCASE	ENDOF	erase	ESP
ESP32-C3?	ESP32-S2?	ESP32-S3?	ESP32?	evaluate	EXECUTE	exit
extract	F-	f.	f.s	F*	F**	F/
F+	F<	F<=	F<>	F=	F>	F>=
F>S	F0<	F0=	FABS	FATAN2	fconstant	FCOS
fdepth	FDROP	FDUP	FEXP	fg	file-exists?	
FILE-POSITION		FILE-SIZE	fill	FIND	fliteral	FLN
FLOOR	flush	FLUSH-FILE	FMAX	FMIN	FNEGATE	FNIP
for	forget	FORTH	forth-builtins		FOVER	FP!
FP@	fp0	free	freq	FROT	FSIN	FSINCOS
FSQRT	FSWAP	fvariable	handler	here	hex	HIGH
hld	hold	httpd	I	if	IMMEDIATE	include
included	included?	INPUT	internals	invert	is	J
K	key	key?	L!	latestxt	leave	LED
ledc	list	literal	load	login	loop	LOW
ls	LSHIFT	max	MDNS.begin	min	mod	ms
MS-TICKS	mv	n.	needs	negate	nest-depth	next
nip	nl	NON-BLOCK	normal	octal	OF	ok
only	open-blocks	OPEN-DIR	OPEN-FILE	OR	order	OUTPUT
OVER	pad	page	PARSE	pause	PI	pin
pinMode	postpone	precision	previous	prompt	PSRAM?	pulseIn
quit	r"	R@	R/O	R/W	R>	r|
r~	rdrop	read-dir	READ-FILE	recurse	refill	registers
remaining	remember	RENAME-FILE	repeat	REPOSITION-FILE		required
reset	resize	RESIZE-FILE	restore	revive	RISC-V?	rm
rot	RP!	RP@	rp0	RSHIFT	rtos	s"
S>F	s>z	save	save-buffers		scr	SD
SD_MMC	sealed	see	Serial	set-precision		set-title
sf,	SF!	SF@	SFLOAT	SFLOAT+	SFLOATS	sign
SL@	sockets	SP!	SP@	sp0	space	spaces
SPIFFS	start-task	startswith?	startup:	state	str	str=
streams	structures	SW@	SWAP	task	tasks	telnetd
terminate	then	throw	thru	tib	to	tone
touch	transfer	transfer	type	u.	U/MOD	UL@
UNLOOP	until	update	use	used	UW@	value
VARIABLE	visual	vlist	vocabulary	W!	W/O	web-interface
webui	while	WiFi	Wire	words	WRITE-FILE	XOR
Xtensa?	z"	z>s				

asm

xtensa disasm disasm1 matchit address istep sextend m. m@ for-ops op >operands >mask >pattern >length >xt op-snap opcodes coden, names operand 1 o bits bit skip advance advance-operand reset reset-operand for-operands operands >printop >inop >next >opmask& bit! mask pattern length demask enmask >>1 odd? high-bit end-code code, code4, code3, code2, code1, callot chere reserve

code-at code-start

bluetooth

SerialBT.new SerialBT.delete SerialBT.begin SerialBT.end SerialBT.available
SerialBT.readBytes SerialBT.write SerialBT.flush SerialBT.hasClient
SerialBT.enableSSP SerialBT.setPin SerialBT.unpairDevice SerialBT.connect
SerialBT.connectAddr SerialBT.disconnect SerialBT.connected
SerialBT.isReady bluetooth-builtins

editor

a r d e wipe p n l

ESP

getHeapSize getFreeHeap getMaxAllocHeap getChipModel getChipCores getFlashChipSize
getCpuFreqMHz getSketchSize deepSleep getEfuseMac esp_log_level_set ESP-builtins

httpd

notfound-response bad-response ok-response response send path method **hasHeader**
handleClient read-headers completed? body content-length header crnl= eat
skipover skipto in@<> end< **goal#** **goal** strcase= upper server client-cr client-emit
client-read client-type **client-len** client httpd-port clientfd **sockfd** body-read
body-1st-read body-chunk body-chunk-size **chunk-filled** chunk **chunk-size**
max-connections

insides

run normal-mode raw-mode step ground handle-key quit-edit save load backspace
delete handle-esc insert update crtype cremit ndown down nup up caret length
capacity text start-size fileh filename# filename max-path

internals

assembler-source xtensa-assembler-source MALLOC SYSFREE REALLOC heap_caps_malloc
heap_caps_free heap_caps_realloc heap_caps_get_total_size heap_caps_get_free_size
heap_caps_get_minimum_free_size heap_caps_get_largest_free_block RAW-YIELD
RAW-TERMINATE READDIR CALLCODE CALL0 CALL1 CALL2 CALL3 CALL4 CALL5 CALL6
CALL7 CALL8 CALL9 CALL10 CALL11 CALL12 CALL13 CALL14 CALL15 DOFLIT S>FLOAT?
fill32 'heap 'context 'latestxt 'notfound 'heap-start 'heap-size 'stack-cells
'boot 'boot-size 'tib 'argc 'argv 'runner 'throw-handler NOP BRANCH 0BRANCH
DONEXT DOLIT DOSET DOCOL DOCON DOVAR DOCREATE DODOES ALITERAL LONG-SIZE
S>NUMBER? 'SYS YIELD EVALUATE1 'builtins internals-builtins autoexec
arduino-remember-filename
arduino-default-use esp32-stats serial-key? serial-key serial-type yield-task
yield-step e' @line grow-blocks use?! common-default-use block-data **block-dirty**
clobber clobber-line include+ path-join **included-files** raw-included include-file
sourcedirname sourcefilename! sourcefilename **sourcefilename#** **sourcefilename&**
starts../ starts./ dirname ends/ default-remember-filename *remember-filename*

restore-name save-name forth-wordlist setup-saving-base 'cold park-forth
park-heap saving-base crtype cremit cases (+to) (to) --? }? ?room scope-create
do-local scope-clear scope-exit local-op scope-depth local+! local! local@
<>locals locals-here locals-area locals-gap locals-capacity ?ins. ins.
vins. onlines line-pos line-width size-all size-vocabulary vocs. voc. voclist
voclist-from see-all >vocnext see-vocabulary nonvoc? see-xt ?see-flags
see-loop see-one indent+! icr see. indent mem= ARGS_MARK -TAB +TAB NONAMED
BUILTIN_FORK SMUDGE IMMEDIATE_MARK relinquish dump-line ca@ cell-shift
cell-base cell-mask MALLOC_CAP_RTCRAM MALLOC_CAP_RETENTION MALLOC_CAP_IRAM_8BIT
MALLOC_CAP_DEFAULT MALLOC_CAP_INTERNAL MALLOC_CAP_SPIRAM MALLOC_CAP_DMA
MALLOC_CAP_8BIT MALLOC_CAP_32BIT MALLOC_CAP_EXEC #f+s internalized BUILTIN_MARK
zplace $place free. boot-prompt raw-ok [SKIP]' [SKIP] ?stack sp-limit input-limit
tib-setup raw.s $@ digit parse-quote leaving, leaving)leaving leaving(
value-bind evaluate&fill evaluate-buffer arrow ?arrow. ?echo input-buffer
immediate? eat-till-cr wascr *emit *key notfound last-vocabulary voc-stack-end
xt-transfer xt-hide xt-find& scope

interrupts

pinchange #GPIO_INTR_HIGH_LEVEL #GPIO_INTR_LOW_LEVEL #GPIO_INTR_ANYEDGE
#GPIO_INTR_NEGEDGE #GPIO_INTR_POSEDGE #GPIO_INTR_DISABLE ESP_INTR_FLAG_INTRDISABLED
ESP_INTR_FLAG_IRAM ESP_INTR_FLAG_EDGE ESP_INTR_FLAG_SHARED ESP_INTR_FLAG_NMI
ESP_INTR_FLAG_LEVELn ESP_INTR_FLAG_DEFAULT gpio_config gpio_reset_pin gpio_set_intr_type
gpio_intr_enable gpio_intr_disable gpio_set_level gpio_get_level gpio_set_direction
gpio_set_pull_mode gpio_wakeup_enable gpio_wakeup_disable gpio_pullup_en
gpio_pulldown_en gpio_pulldown_dis gpio_hold_en gpio_hold_dis
gpio_deep_sleep_hold_en gpio_deep_sleep_hold_dis gpio_install_isr_service
gpio_isr_handler_add gpio_isr_handler_remove
gpio_set_drive_capability gpio_get_drive_capability esp_intr_alloc esp_intr_free
interrupts-builtins

ledc

ledcSetup ledcAttachPin ledcDetachPin ledcRead ledcReadFreq ledcWrite ledcWriteTone
ledcWriteNote ledc-builtins

oled

OledInit SSD1306_SWITCHCAPVCC SSD1306_EXTERNALVCC WHITE BLACK OledReset HEIGHT
WIDTH OledAddr OledNew OledDelete OledBegin OledHOME OledCLS OledTextc
OledPrintln OledNumln OledNum OledDisplay OledPrint OledInvert OledTextsize
OledSetCursor OledPixel OledDrawL OledCirc OledCircF OledRect OledRectF
OledRectR OledRectRF oled-builtins

registers

m@ m!

riscv

C.FSWSP, C.SWSP, C.FSDSP, C.ADD, C.JALR, C.EBREAK, C.MV, C.JR, C.FLWSP,
C.LWSP, C.FLDSP, C.SLLI, BNEZ, BEQZ, C.J, C.ADDW, C.SUBW, C.AND, C.OR,
C.XOR, C.SUB, C.ANDI, C.SRAI, C.SRLI, C.LUI, C.LI, C.JAL, C.ADDI, C.NOP,
C.FSW, C.SW, C.FSD, C.FLW, C.LW, C.FLD, C.ADDI4SP, C.ILL, EBREAK, ECALL,
AND, OR, SRA, SRL, XOR, SLTU, SLT, SLL, SUB, ADD, SRAI, SRLI, SLLI, ANDI,

ORI, XORI, SLTIU, SLTI, ADDI, SW, SH, SB, LHU, LBU, LW, LH, LB, BGEU, BLTU, BGE, BLT, BNE, BEQ, JALR, JAL, AUIPC, LUI, J-TYPE U-TYPE B-TYPE S-TYPE I-TYPE R-TYPE rs2' rs2#' rs2 rs2# rs1' rs1#' rs1 rs1# rd' rd#' rd rd# offset ofs ofs. >ofs iiii i numeric register' reg'. reg>reg' register reg. nop x31 x30 x29 x28 x27 x26 x25 x24 x23 x22 x21 x20 x19 x18 x17 x16 x15 x14 x13 x12 x11 x10 x9 x8 x7 x6 x5 x4 x3 x2 x1 zero

rtos

vTaskDelete xTaskCreatePinnedToCore xPortGetCoreID rtos-builtins

SD

SD.begin SD.beginFull SD.beginDefaults SD.end SD.cardType SD.totalBytes SD.usedBytes SD-builtins

SD_MMC

SD_MMC.begin SD_MMC.beginFull SD_MMC.beginDefaults SD_MMC.end SD_MMC.cardType SD_MMC.totalBytes SD_MMC.usedBytes SD_MMC-builtins

Serial

Serial.begin Serial.end Serial.available Serial.readBytes Serial.write Serial.flush Serial.setDebugOutput Serial2.begin Serial2.end Serial2.available Serial2.readBytes Serial2.write Serial2.flush Serial2.setDebugOutput serial-builtins

sockets

ip. ip# ->h_addr ->addr! ->addr@ ->port! ->port@ **sockaddr** l, s, bs, **SO_REUSEADDR** **SOL_SOCKET** **sizeof(sockaddr_in)** **AF_INET** **SOCK_RAW** **SOCK_DGRAM** **SOCK_STREAM** socket setsockopt bind listen connect sockaccept select poll send sendto sendmsg recv recvfrom recvmsg gethostbyname errno sockets-builtins

spi

SPI.begin SPI.end SPI.setHwCs SPI.setBitOrder SPI.setDataMode SPI.setFrequency SPI.setClockDivider SPI.getClockDivider SPI.transfer SPI.transfer8 SPI.transfer16 SPI.transfer32 SPI.transferBytes SPI.transferBits SPI.write SPI.write16 SPI.write32 SPI.writeBytes SPI.writePixels SPI.writePattern SPI-builtins

SPIFFS

SPIFFS.begin SPIFFS.end SPIFFS.format SPIFFS.totalBytes SPIFFS.usedBytes SPIFFS-builtins

streams

stream> >stream stream>ch ch>stream wait-read wait-write empty? full? stream# >offset >read >write **stream**

structures

field struct-align align-by last-struct struct long ptr i64 i32 i16 i8
typer last-align

tasks

.tasks main-task task-list

telnetd

server broker-connection wait-for-connection connection telnet-key telnet-type
telnet-emit broker client-len client telnet-port clientfd sockfd

visual

edit insides

web-interface

server webserver-task do-serve handle1 serve-key serve-type handle-input handle-index out-string output-stream input-stream out-size webserver index-html index-html#

WiFi

Wire.begin Wire.setClock Wire.getClock Wire.setTimeout Wire.getTimeout Wire.beginTransmission Wire.endTransmission Wire.requestFrom Wire.write Wire.available Wire.read Wire.peek Wire.flush Wire-builtins

xtensa

WUR, WSR, WITLB, WER, WDTLB, WAITI, SSXU, SSX, SSR, SSL, SSIU, SSI, SSAI,
SSA8L, SSA8B, SRLI, SRL, SRC, SRAI, SRA, SLLI, SLL, SICW, SICT, SEXT, SDCT,
RUR, RSR, RSIL, RFI, ROTW, RITLB1, RITLB0, RER, RDTLB1, RDTLB0, PITLB,
PDTLB, NSAU, NSA, MULA.DD.HH, MULA.DD.LH, MULA.DD.HL, MULA.DD.LL, MULS.DD
MULA.DA.HH, MULA.DA.LH, MULA.DA.HL, MULA.DA.LL, MULS.DA MULA.AD.HH, MULA.AD.LH,
MULA.AD.HL, MULA.AD.LL, MULS.AD MULA.AA.HH, MULA.AA.LH, MULA.AA.HL, MULA.AA.LL,
MULS.AA MULA.DD.HH.LDINC, MULA.DD.LH.LDINC, MULA.DD.HL.LDINC, MULA.DD.LL.LDINC,
MULA.DD.LDINC MULA.DD.HH.LDDEC, MULA.DD.LH.LDDEC, MULA.DD.HL.LDDEC,
MULA.DD.LL.LDDEC,
MULA.DD.LDDEC MULA.DD.HH, MULA.DD.LH, MULA.DD.HL, MULA.DD.LL, MULA.DD
MULA.DA.HH.LDINC,
MULA.DA.LH.LDINC, MULA.DA.HL.LDINC, MULA.DA.LL.LDINC, MULA.DA.LDINC
MULA.DA.HH.LDDEC,
MULA.DA.LH.LDDEC, MULA.DA.HL.LDDEC, MULA.DA.LL.LDDEC, MULA.DA.LDDEC MULA.DA.HH,
MULA.DA.LH, MULA.DA.HL, MULA.DA.LL, MULA.DA MULA.AD.HH, MULA.AD.LH, MULA.AD.HL,
MULA.AD.LL, MULA.AD MULA.AA.HH, MULA.AA.LH, MULA.AA.HL, MULA.AA.LL, MULA.AA
MUL16U, MUL16S, MUL.DD.HH, MUL.DD.LH, MUL.DD.HL, MUL.DD.LL, MUL.DD MUL.DA.HH,
MUL.DA.LH, MUL.DA.HL, MUL.DA.LL, MUL.DA MUL.AD.HH, MUL.AD.LH, MUL.AD.HL,
MUL.AD.LL, MUL.AD MUL.AA.HH, MUL.AA.LH, MUL.AA.HL, MUL.AA.LL, MUL.AA MOVT,
MOVSP, MOVT.S, MOVF.S, MOVGEZ.S, MOVLTZ.S, MOVNEZ.S, MOVEQZ.S, ULE.S, OLE.S,
ULT.S, OLT.S, UEQ.S, OEQ.S, UN.S, CMPSOP NEG.S, WFR, RFR, ABS.S, MOV.S,
ALU2.S UTRUNC.S, UFLOAT.S, FLOAT.S, CEIL.S, FLOOR.S, TRUNC.S, ROUND.S,

```
MSUB.S, MADD.S, MUL.S, SUB.S, ADD.S, ALU.S MOVF, MOVGEZ, MOVLTZ, MOVNEZ,
MOVEQZ, MAXU, MINU, MAX, MIN, CONDOP MOV, LSXU, LSX, L32E, LICW, LICT,
LDCT, JX, IITLB, IDTLB, LSIU, LSI, LDINC, LDDEC, L32R, EXTUI, S32E, S32RI,
S32C1I, ADDMI, ADDI, L32AI, L16SI, S32I, S16I, S8I, L32I, L16UI, L8UI,
LDSTORE MOVI, IIU, IHU, IPFL, DIWBI, DIWB, DIU, DHU, DPFL, CACHING2 III,
IHI, IPF, DII, DHI, DHWBI, DHWB, DPFWO, DPFRO, DPFW, DPFR, CACHING1 CLAMPS,
BREAK, CALLX12, CALLX8, CALLX4, CALLX0, CALLXOP CALL12, CALL8, CALL4, CALL0,
CALLOP LOOPGTZ, LOOPNEZ, LOOP, BT, BF, BRANCH2b J, BGEUI, BGEI, BGEZ, BLTUI,
BLTI, BLTZ, BNEI, BNEZ, ENTRY, BEQI, BEQZ, BRANCH2e BRANCH2a BRANCH2 BBSI,
BBS, BNALL, BGEU, BGE, BNE, BANY, BBCI, BBC, BALL, BLTU, BLT, BEQ, BNONE,
BRANCH1 REMS, REMU, QUOS, QUOU, MULSH, MULUH, MULL, XORB, ORBC, ORB, ANDBC,
ANDB, ALU2 ALL8, ANY8, ALL4, ANY4, ANYALL SUBX8, SUBX4, SUBX2, SUB, ADDX8,
ADDX4, ADDX2, ADD, XOR, OR, AND, ALU XSR, ABS, NEG, RFDO, RFDD, SIMCALL,
SYSCALL, RFWU, RFWO, RFDE, RFUE, RFME, RFE, NOP, EXTW, MEMW, EXCW, DSYNC,
ESYNC, RSYNC, ISYNC, RETW, RET, ILL, ILL.N, NOP.N, RETW.N, RET.N, BREAK.N,
MOV.N, MOVI.N, BNEZ.N, BEQZ.N, ADDI.N, ADD.N, S32I.N, L32I.N, tttt t ssss
s rrrr r bbbb b y w iiii i xxxx x sa sa. >sa entry12 entry12' entry12.
>entry12 coffset18 cofs cofs. >cofs offset18 offset12 offset8 ofs18 ofs12
ofs8 ofs18. ofs12. ofs8. >ofs sr imm16 imm8 imm4 im numeric register reg.
nop a15 a14 a13 a12 a11 a10 a9 a8 a7 a6 a5 a4 a3 a2 a1 a0
```

Ресурсы

по-английски

- **ESP32forth** страницу поддерживает Брэд НЕЛЬСОН, создатель ESP32forth. Там вы найдете все версии (ESP32, Windows, Web, Linux...).
  https://esp32forth.appspot.com/ESP32forth.html
-

На французском

- **ESP32 Forth** сайт на двух языках (французский, английский) с множеством примеров
  https://esp32.arduino-forth.com/

GitHub

- **Ueforth** ресурсы, поддерживаемые Брэдом НЕЛЬСОНОМ. Содержит все исходные файлы языков Forth и C для ESP32forth.
  https://github.com/flagxor/ueforth
- **ESP32forth** исходные коды и документация для ESP32forth. Ресурсы поддерживаются Марком ПЕТРЕМАНОМ.
  https://github.com/MPETREMANN11/ESP32forth
- **ESP32forthStation** ресурсы, поддерживаемые Ульрихом Хоффманом. Автономный компьютер Forth с одноплатным компьютером LillyGo TTGO VGA32 и ESP32forth.
  https://github.com/uho/ESP32forthStation
- **ESP32Forth** ресурсы, поддерживаемые FJ RUSSO
  https://github.com/FJRusso53/ESP32Forth
- **esp32forth-addons** ресурсы, поддерживаемые Питером ФОРТОМ
  https://github.com/PeterForth/esp32forth-addons
- **Esp32forth-org** Репозиторий кода для членов групп Forth2020 и ESp32forth.
  https://github.com/Esp32forth-org
-

Лексический индекс