Wielka księga
dla ESP32forth

wersja 1.2 - 30 października 2023

Autor

- Marc PETREMANN petremann@arduino-forth.com

Pracownicy

- Vaclav POSSELT

Treść

Wstęp

Od 2019 roku zarządzałem kilkoma stronami internetowymi poświęconymi rozwojowi języka FORTH dla płytek ARDUINO i ESP32, a także wersją webową eForth :

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

Strony te są dostępne w dwóch językach: francuskim i angielskim. Co roku płacę za hosting strony głównej **arduino-forth.com**.

Prędzej czy później i tak późno, jak to możliwe, stanie się tak, że nie będę już w stanie zapewnić trwałości tych obiektów. Konsekwencją będzie zniknięcie informacji rozpowszechnianych na tych stronach.

Ta książka jest kompilacją treści z moich stron internetowych. Jest dystrybuowany bezpłatnie z repozytorium Github. Ta metoda dystrybucji zapewni większą trwałość niż strony internetowe.

Nawiasem mówiąc, jeśli niektórzy czytelnicy tych stron chcą wnieść swój wkład, są mile widziani :

- sugerowanie rozdziałów
- zgłaszania błędów lub sugerowania zmian;
- pomoc w tłumaczeniu…

Pomoc w tłumaczeniu

Tłumacz Google umożliwia łatwe tłumaczenie tekstów, ale z błędami. Dlatego proszę o pomoc w poprawieniu tłumaczeń.

W praktyce udostępniam rozdziały już przetłumaczone w formacie LibreOffice. Jeśli chcesz pomóc przy tych tłumaczeniach, Twoja rola będzie polegać po prostu na poprawieniu i zwróceniu tych tłumaczeń.

Korekta rozdziału zajmuje niewiele czasu, od jednej do kilku godzin.

Aby się ze mną skontaktować : petremann@arduino-forth.com

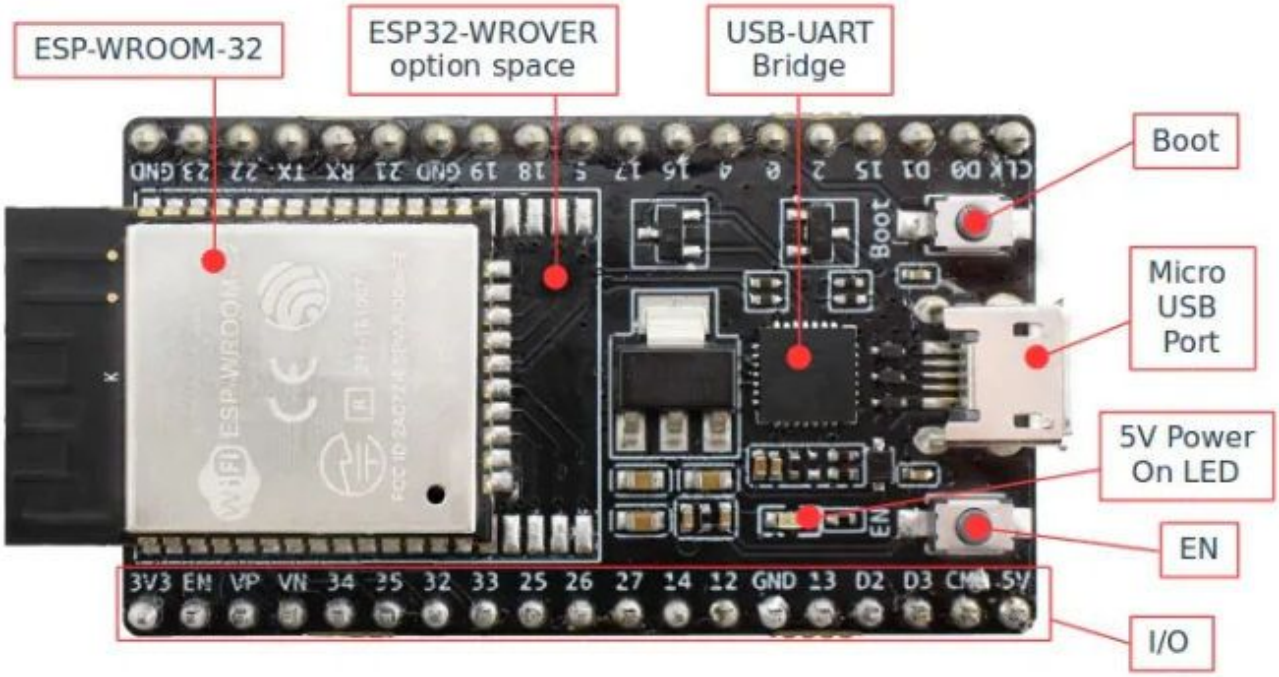
Odkrycie karty ESP32

Prezentacja

Płyta ESP32 nie jest płytą ARDUINO. Jednakże narzędzia programistyczne wykorzystują pewne elementy ekosystemu ARDUINO, takie jak ARDUINO IDE.

Mocne strony

Pod względem liczby dostępnych portów karta ESP32 plasuje się pomiędzy ARDUINO NANO i ARDUINO UNO. Podstawowy model posiada 38 złączy:



Urządzenia ESP32 obejmują:

- 18 kanałów przetwornika analogowo-cyfrowego (ADC)
- 3 interfejsy SPI
- 3 interfejsy UART
- 2 interfejsy I2C
- 16 kanałów wyjściowych PWM
- 2 przetworniki cyfrowo-analogowe (DAC)
- 2 interfejsy I2S
- 10 GPIO z czujnikiem pojemnościowym

Funkcje ADC (przetwornik analogowo-cyfrowy) i DAC (przetwornik cyfrowo-analogowy) są przypisane do określonych pinów statycznych. Możesz jednak zdecydować, które piny to UART, I2C, SPI, PWM itp. Wystarczy przypisać je w kodzie. Jest to możliwe dzięki funkcji multipleksowania układu ESP32.

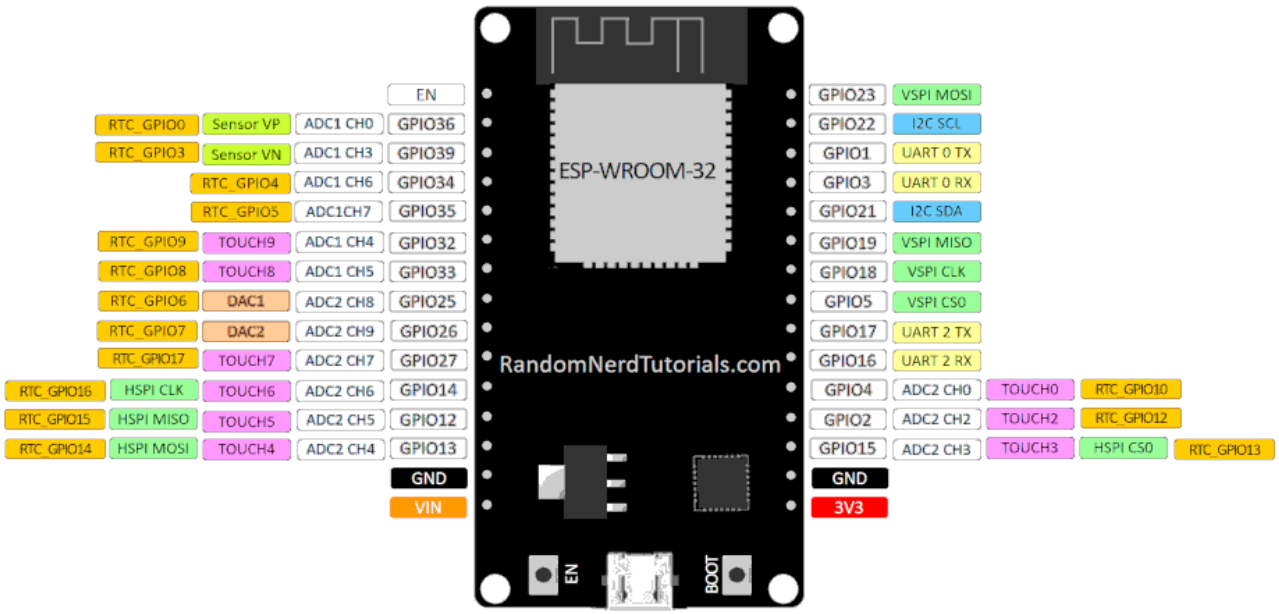
Większość złączy ma wiele zastosowań.

Jednak tym, co wyróżnia płytkę ESP32, jest to, że jest ona standardowo wyposażona w obsługę Wi-Fi i Bluetooth, coś, co płyty ARDUINO oferują jedynie w formie rozszerzeń.

Wejścia/wyjścia GPIO na ESP32

Tutaj na zdjęciu karta ESP32, z której wyjaśnimy rolę poszczególnych wejść/wyjść GPIO:

Położenie i liczba wejść/wyjść GPIO może się zmieniać w zależności od marki karty. W takim przypadku autentyczne są jedynie wskazania znajdujące się na mapie fizycznej. Na zdjęciu dolny rząd, od lewej do prawej: CLK, SD0, SD1, G15, G2, G0, G4, G16.....G22, G23, GND.



Na tym schemacie widzimy, że dolny rząd zaczyna się od 3V3, podczas gdy na zdjęciu to wejście/wyjście znajduje się na końcu górnego rzędu. Dlatego bardzo ważne jest, aby nie polegać na schemacie, a zamiast tego dokładnie sprawdzić poprawność podłączenia urządzeń peryferyjnych i komponentów na fizycznej karcie ESP32.

Płytki rozwojowe oparte na ESP32 mają zazwyczaj 33 piny oprócz tych przeznaczonych do zasilania. Niektóre piny GPIO mają nieco szczególne funkcje:

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

Jeśli Twoja karta ESP32 ma wejścia/wyjścia GPIO6, GPIO7, GPIO8, GPIO9, GPIO10, GPIO11, zdecydowanie nie powinieneś ich używać, ponieważ są one podłączone do pamięci flash ESP32. Jeśli ich użyjesz, ESP32 nie będzie działać.

Wejścia/wyjścia GPIO1(TX0) i GPIO3(RX0) służą do komunikacji z komputerem w trybie UART poprzez port USB. Jeśli z nich skorzystasz, nie będziesz już mógł komunikować się z kartą.

GPIO36(VP), GPIO39(VN), GPIO34, GPIO35 We/Wy mogą być używane tylko jako wejścia. Nie mają również wbudowanych wewnętrznych rezystorów podciągających i rozwijających.

Terminal EN umożliwia kontrolę stanu zapłonu ESP32 za pomocą przewodu zewnętrznego. Podłącza się go do przycisku EN na karcie. Gdy ESP32 jest włączony, ma napięcie 3,3 V. Jeżeli podłączymy ten pin do masy to ESP32 zostanie wyłączony. Można go używać, gdy ESP32 jest w pudełku i chcesz mieć możliwość jego włączania/wyłączania za pomocą przełącznika.

Urządzenia peryferyjne ESP32

Aby współdziałać z modułami, czujnikami lub obwodami elektronicznymi, ESP32, jak każdy mikrokontroler, posiada wiele urządzeń peryferyjnych. Jest ich więcej niż na klasycznej płytce Arduino.

ESP32 ma następujące urządzenia peryferyjne:

- 3 interfejsy UART
- 2 interfejsy I2C
- 3 interfejsy SPI
- 16 wyjść PWM
- 10 czujników pojemnościowych
- 18 wejść analogowych (ADC)
- 2 wyjścia DAC

Niektóre urządzenia peryferyjne są już wykorzystywane przez ESP32 podczas jego podstawowej pracy. Dlatego jest mniej możliwych interfejsów dla każdego urządzenia.

Zainstaluj ESP32Forth

Pobierz ESP32forth

Pierwszy krok polega na odzyskaniu kodu źródłowego ESP32forth w języku C. Najlepiej użyj najnowszej wersji:
https://esp32forth.appspot.com/ESP32forth.html

Zawartość pobranego pliku:

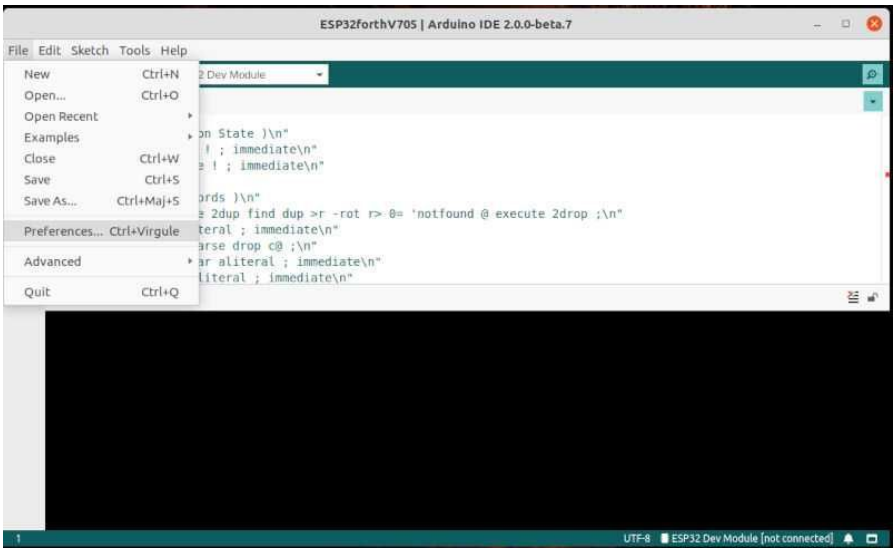
```
ESP32forth-7.0.x.x
     ESP32forth
          readme.txt
          esp32forth.ino
          optional
               SPI-flash.h
               serial-blueooth.h
               ... itd....
```

Kompilacja i instalacja ESP32forth

plik **esp32forth.ino** do katalogu roboczego. Opcjonalny katalog zawiera pliki umożliwiające rozbudowę ESP32forth. W przypadku naszej pierwszej kompilacji i przesyłania ESP32forward te pliki nie są potrzebne.

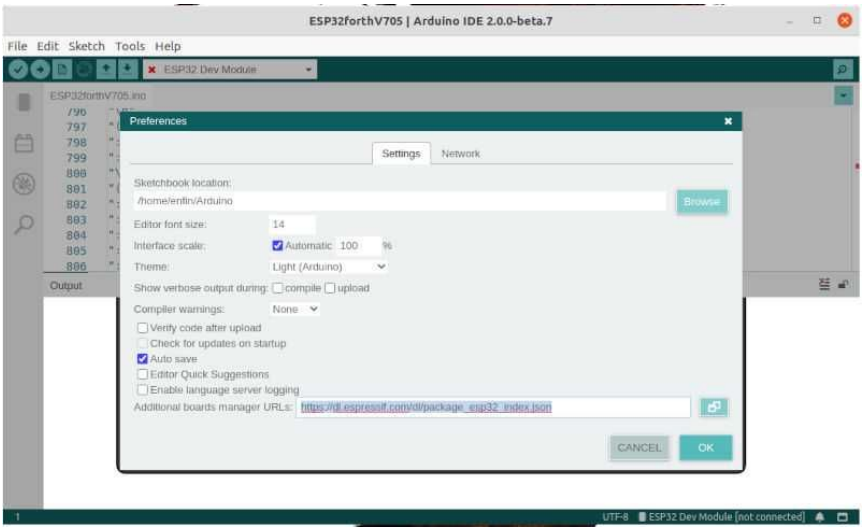
Aby skompilować ESP32forth, musisz mieć już zainstalowany ARDUINO IDE na swoim komputerze:
https://docs.arduino.cc/software/ide-v2

Po zainstalowaniu ARDUINO IDE uruchom je. ARDUINO IDE jest otwarte, tutaj wersja 2.0. Kliknij *file* i wybierz *Preferences* :

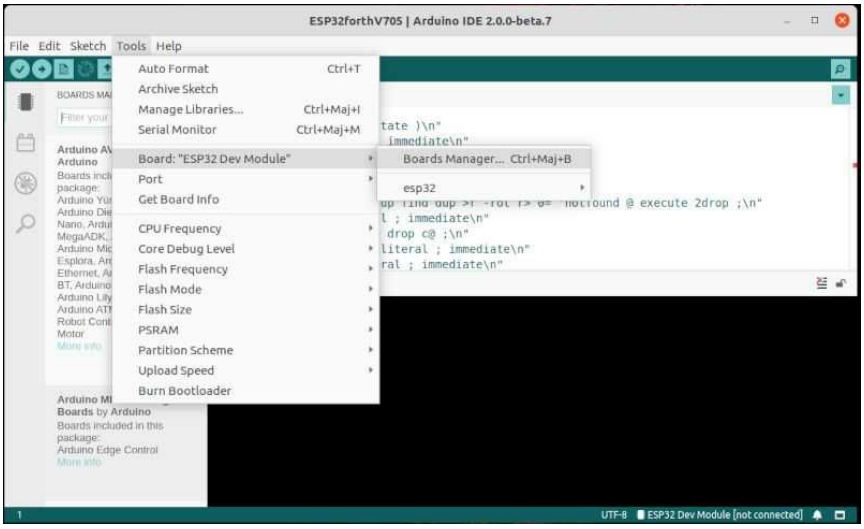


W wyświetlonym oknie przejdź do pola tekstowego oznaczonego *Additional boards manager URLs*: i wpisz następującą linię:

```
https://dl.espressif.com/dl/package_esp32_index.json
```



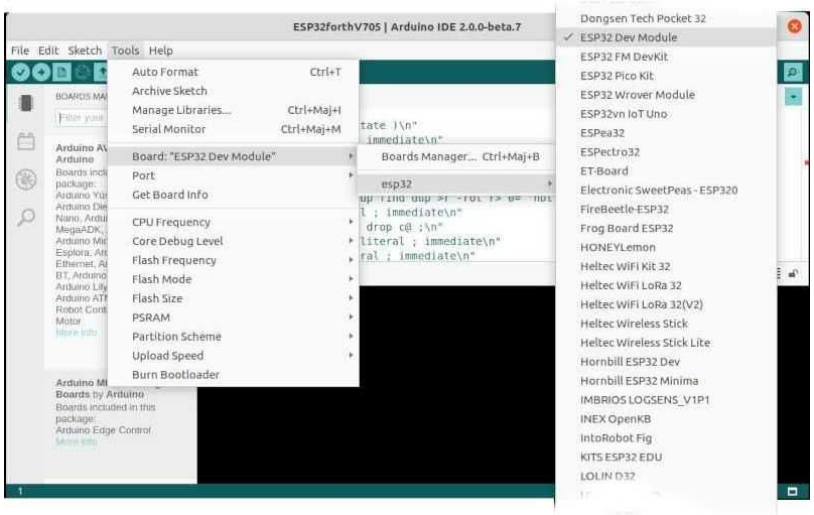
Następnie kliknij *Tools* i wybierz *Board* :.



Ten wybór powinien umożliwić instalację pakietów dla ESP32. Zaakceptuj tę instalację.

Powinieneś wtedy mieć dostęp do wyboru kart ESP32:

Wybór płyty **ESP32 Dev Module** :



Ustawienia dla ESP32 WROOM

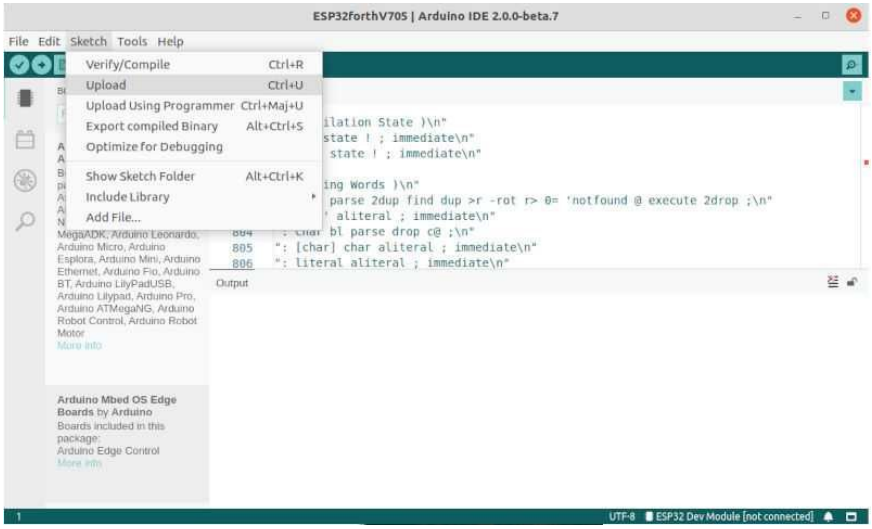
Oto inne ustawienia potrzebne przed kompilacją ESP32forth. Uzyskaj dostęp do ustawień, klikając ponownie *Narzędzia* :

```
-- TOOLS----+-- BOARD    ----+-- ESP32 ------+-- ESP32 Dev Module
            +-- Port: -------+-- COMx
            |
            +-- CPU Frequency ---------+-- 240 Mhz
            +-- Core Debug Level ------+-- None
            +-- Erase All Flash...-----+-- Disabled
            +-- Events Run On ---------+-- Core 1
            +-- Flash Frequency -------+-- 80 Mhz
            +-- Flash Mode ------------+-- QIO
            +-- Flash Size ------------+-- 4MB
            +-- JTAG Adapter ----------+-- FTDI Adapter
            +-- Arduino Runs on -------+-- Core 1
            +-- PSRAM -----------------+-- Disabled
            +-- Partition Scheme ------+-- Default 4MB with SPIFFS
            +-- Upload Speed ----------+-- 921600
```

Rozpocznij kompilację

Pozostaje tylko skompilować ESP32forth. Załaduj kod źródłowy poprzez *File* i *Open* .

Zakłada się, że płyta ESP32 jest podłączona do portu USB. Rozpocznij kompilację, klikając *Sketch* i wybierając *Upload* :



Jeżeli wszystko przebiegło poprawnie należy automatycznie przenieść kod binarny do płytki ESP32. Jeśli kompilacja przebiegnie bez błędów, ale wystąpi błąd transferu, skompiluj ponownie plik **esp32forth.ino** . W momencie transferu naciśnij przycisk oznaczony **BOOT** na płycie ESP32. Powinno to udostępnić kartę do przesyłania kodu binarnego ESP32forth.

Instalacja i konfiguracja ARDUINO IDE na filmie:

- Windows: https://www.youtube.com/watch?v=2AZQfieHv9g
- Linux: https://www.youtube.com/watch?v=JeD3nz0__nc

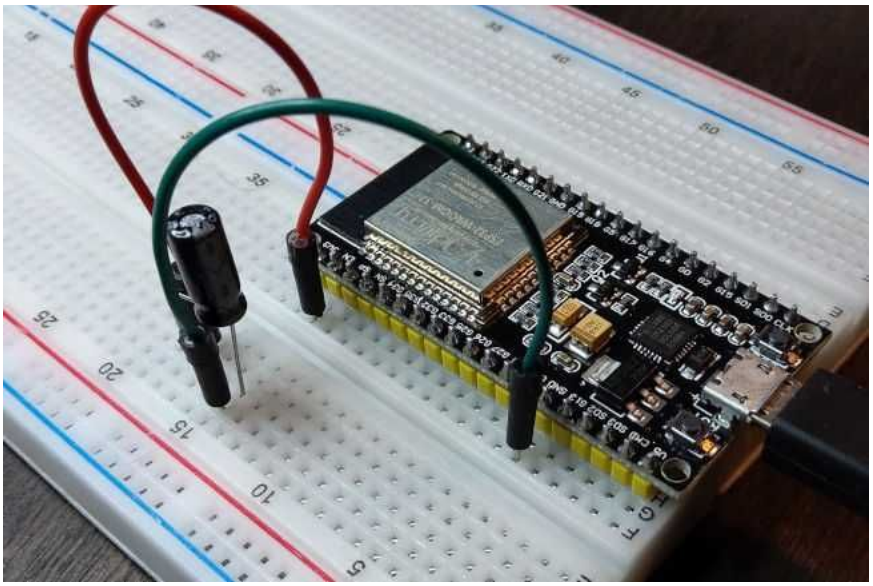
Napraw błąd połączenia przesyłania

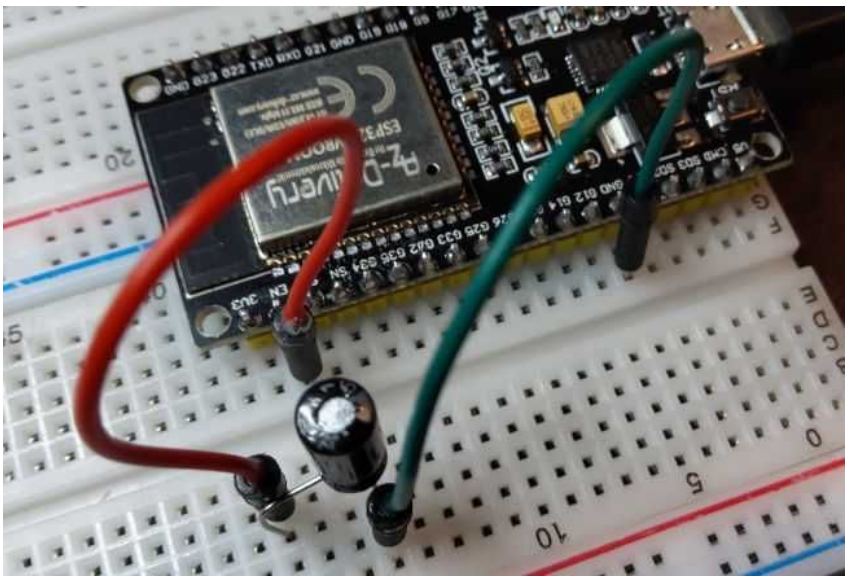
Dowiedz się, jak naprawić błąd krytyczny, który wystąpił: „Failed to connect to ESP32: Timed out waiting for packet header” podczas próby jednorazowego przesłania nowego kodu na kartę ESP32.

Niektóre płytki rozwojowe ESP32 (czytaj najlepsze płyty ESP32) nie przechodzą automatycznie w tryb flashowania/przesyłania podczas pobierania nowego kodu.

Oznacza to, że przy próbie przesłania nowego szkicu na płytkę ESP32 ARDUINO IDE nie może połączyć się z płytką i pojawia się następujący komunikat o błędzie:

Aby płytka ESP32 automatycznie przełączała się w tryb flash/download, możemy podłączyć kondensator elektrolityczny 10uF pomiędzy pinami EN i GND:





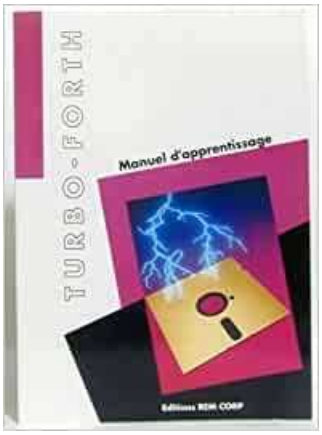
Ta manipulacja jest konieczna tylko wtedy, gdy jesteś w fazie przesyłania ESP32forth z ARDUINO IDE. Po zainstalowaniu ESP32forth na płycie ESP32 użycie tego kondensatora nie jest już konieczne.

Po co programować w języku FORTH na ESP32?

Preambuła

Programuję w FORTH od 1983 roku. Przestałem programować w FORTH w 1996 roku. Ale nigdy nie przestałem monitorować ewolucji tego języka. W 2019 roku wznowiłem programowanie na ARDUINO z FlashForth, a następnie ESP32forth.

Jestem współautorem kilku książek dotyczących języka 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)

Programowanie w języku FORTH było zawsze moim hobby aż do roku 1992, kiedy to skontaktował się ze mną menadżer firmy działającej jako podwykonawca dla przemysłu samochodowego. Zależało im na stworzeniu oprogramowania w języku C. Musieli zamówić automat przemysłowy.

Dwóch projektantów oprogramowania tej firmy programowało w języku C, a dokładniej TURBO-C firmy Borland. A ich kod nie mógł być wystarczająco zwarty i szybki, aby zmieścić się w 64 kilobajtach pamięci RAM. Był rok 1992 i nie istniały jeszcze rozszerzenia typu flash. W tych 64 KB RAM-u musieliśmy zmieścić MS-DOS 3.0 i aplikację!

Przez miesiąc programiści języka C przekręcali problem we wszystkich kierunkach, nawet stosując inżynierię wsteczną za pomocą SOURCER (dezasemblera), aby wyeliminować nieistotne części kodu wykonywalnego.

Przeanalizowałem przedstawiony mi problem. Zaczynając od zera, samodzielnie w ciągu tygodnia stworzyłem doskonale działający prototyp, który spełniał specyfikacje. Przez trzy lata, od 1992 do 1995, stworzyłem wiele wersji tej aplikacji, która była używana na liniach montażowych kilku producentów samochodów.

Granice między językiem a zastosowaniem

Wszystkie języki programowania są udostępniane w następujący sposób :

- interpreter i wykonywalny kod źródłowy: BASIC, PHP, MySQL, JavaScript, itp... Aplikacja zawarta jest w jednym lub większej liczbie plików, które będą interpretowane w razie potrzeby. System musi na stałe hostować interpreter wykonujący kod źródłowy;
- kompilator i/lub asembler: C, Java itp. Niektóre kompilatory generują kod natywny, to znaczy wykonywalny specjalnie w systemie. Inne, jak Java, kompilują kod wykonywalny na wirtualnej maszynie Java.

Język FORTH jest wyjątkiem. Obejmuje :

- interpreter zdolny do wykonania dowolnego słowa w języku FORTH
- kompilator umożliwiający rozszerzenie słownika słów FORTH.

Co to jest słowo FORTH?

Słowo FORTH oznacza dowolne wyrażenie słownikowe składające się ze znaków ASCII i nadające się do interpretacji i/lub kompilacji: `words` umożliwiają wyświetlenie listy wszystkich słów w słowniku FORTH.

Niektórych słów FORTH można używać tylko w kompilacji: na przykład `if else then` .

W przypadku języka FORTH podstawową zasadą jest to, że nie tworzymy aplikacji. W FORTH rozbudowujemy słownik! Każde nowe słowo, które zdefiniujesz, będzie w takim samym stopniu częścią słownika FORTH, jak wszystkie słowa zdefiniowane wcześniej na początku FORTH. Przykład :

```
: typeToLoRa ( -- )
    0 echo !    \ wyłącza echo wyświetlacza terminala
    ['] serial2-type is type
  ;
: typeToTerm ( -- )
    ['] default-type is type
    -1 echo !   \ aktywuje echo wyświetlacza terminala
  ;
```

Tworzymy dwa nowe słowa: `typeToLoRa` i `typeToTerm` , które uzupełnią słownik predefiniowanych słów.

Słowo jest funkcją?

Tak i nie. Tak naprawdę słowo może być stałą, zmienną, funkcją... Tutaj w naszym przykładzie następująca sekwencja:

```
: typeToLoRa ...kod... ;
```

miałby swój odpowiednik w języku C:

```
typ pustyToLoRa() { ...kod... }
```

W języku FORTH nie ma ograniczeń pomiędzy językiem a zastosowaniem.

W FORTH, podobnie jak w języku C, w definicji nowego słowa można użyć dowolnego słowa już zdefiniowanego.

Tak, ale dlaczego w takim razie FORTH, a nie C?

Spodziewałem się tego pytania.

W języku C dostęp do funkcji można uzyskać jedynie poprzez główną funkcję `main()` . Jeśli ta funkcja integruje kilka dodatkowych funkcji, znalezienie błędu parametru w przypadku nieprawidłowego działania programu staje się trudne.

Wręcz przeciwnie, za pomocą FORTH możliwe jest wykonanie - za pośrednictwem interpretera - dowolnego słowa predefiniowanego lub zdefiniowanego przez Ciebie, bez konieczności przechodzenia przez główne słowo programu.

Interpreter FORTH jest natychmiast dostępny na karcie ESP32 poprzez program typu terminal i łącze USB pomiędzy kartą ESP32 a komputerem.

Kompilacja programów napisanych w języku FORTH odbywa się na karcie ESP32, a nie na komputerze PC. Nie ma przesyłania. Przykład :

```
: >gray ( n -- n' )
    dup 2/ xor      \ n' = n xor (1 logiczne przesunięcie w prawo)
  ;
```

Definicja ta jest przesyłana do terminala metodą kopiuj/wklej. Interpreter/kompilator FORTH przeanalizuje strumień i skompiluje nowe słowo `>gray` .

W definicji `>gray` widzimy sekwencję `dup 2/ xor` . Aby przetestować tę sekwencję, po prostu wpisz ją w terminalu. Aby uruchomić `>gray` , po prostu wpisz to słowo w terminalu, poprzedzone liczbą do przekształcenia.

Język FORTH w porównaniu do języka C

To moja najmniej ulubiona część. Nie lubię porównywać języka FORTH do języka C. Ponieważ jednak prawie wszyscy programiści używają języka C, spróbuję tego ćwiczenia.

Oto test z `if()` w języku C:

```
if(j > 13){              // Jeśli odebrane zostaną wszystkie bity
    rc5_ok = 1;          // Proces dekodowania przebiegł prawidłowo
    detachInterrupt(0);  // Wyłącz przerwanie zewnętrzne (INT0)
    return;
}
```

Przetestuj za pomocą `if` w języku FORTH (fragment kodu):

```
var-j @ 13 >        \ Jeśli odebrane zostaną wszystkie bity
    if
        1 rc5_ok !  \ Proces dekodowania przebiega prawidłowo
        di          \ Wyłącz przerwanie zewnętrzne (INT0)
        exit
```

```
    then
```

Oto inicjalizacja rejestrów w języku C:

```
void setup() {
  //  Konfiguracja modułu Timer1
  TCCR1A = 0;
  TCCR1B = 0;          // Wyłącz moduł Timer1
  TCNT1  = 0;          // Ustaw wartość wstępnego ładowania Timera1
na 0
  TIMSK1 = 1;          // włącz przerwanie przepełnienia Timera1
}
```

Ta sama definicja w języku FORTH:

```
: setup ( -- )
  \ Konfiguracja modułu Timer1
  0 TCCR1A !
  0 TCCR1B !     \ Wyłącza moduł Timer1
  0 TCNT1  !     \ Ustawia wartość wstępnego ładowania Timera 1 na 0
(reset)
  1 TIMSK1 !     \ włącz przerwanie przepełnienia Timera 1
;
```

Co pozwala FORTH w porównaniu z językiem C

Rozumiemy, że FORTH od razu daje dostęp do wszystkich słów w słowniku, ale nie tylko. Za pośrednictwem interpretera mamy także dostęp do całej pamięci karty ESP32. Połącz się z płytą ARDUINO, na której zainstalowany jest FlashForth, a następnie po prostu wpisz:

```
hex here 100 dump
```

Powinieneś znaleźć to na ekranie terminala:

```
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
```

Odpowiada to zawartości pamięci flash.

A język C nie mógłby tego zrobić?

Tak, ale nie tak proste i interaktywne jak w języku FORTH.

Zobaczmy inny przypadek podkreślający niezwykłą zwartość języka FORTH...

Ale dlaczego stos, a nie zmienne?

Stos to mechanizm zaimplementowany w prawie wszystkich mikrokontrolerach i mikroprocesorach. Nawet język C wykorzystuje stos, ale nie masz do niego dostępu.

Pełny dostęp do stosu danych daje dopiero język FORTH. Na przykład, aby dokonać dodania, układamy dwie wartości w stos, wykonujemy dodawanie i wyświetlamy wynik: `2 5 + .` wyświetla `7` .

To trochę destabilizujące, ale kiedy zrozumiesz mechanizm stosu danych, bardzo docenisz jego niesamowitą wydajność.

Stos danych umożliwia przesyłanie danych pomiędzy słowami FORTH znacznie szybciej niż poprzez przetwarzanie zmiennych, jak w języku C lub jakimkolwiek innym języku wykorzystującym zmienne.

Czy jesteś przekonany?

Osobiście wątpię, czy ten pojedynczy rozdział nieodwracalnie nawróci Cię do programowania w języku FORTH. Próbując opanować karty ESP32, masz dwie możliwości:

- programować w języku C i korzystać z licznych dostępnych bibliotek. Jednak pozostaniesz zamknięty w możliwościach tych bibliotek. Dostosowanie kodów do języka C wymaga prawdziwej wiedzy z programowania w języku C i opanowania architektury kart ESP32. Tworzenie złożonych programów zawsze będzie stanowić problem.
- wypróbuj przygodę FORTH i odkryj nowy, ekscytujący świat. Oczywiście nie będzie to łatwe. Będziesz musiał dogłębnie zrozumieć architekturę kart ESP32, rejestry i flagi rejestrów. W zamian będziesz miał dostęp do programowania idealnie dopasowanego do Twoich projektów.

Czy są jakieś profesjonalne aplikacje napisane w FORTH?

O tak! Począwszy od teleskopu kosmicznego HUBBLE, którego niektóre elementy zostały napisane w języku FORTH.

Niemiecki TGV ICE (Intercit y Express) wykorzystuje procesory RTX2000 do sterowania silnikami za pomocą półprzewodników mocy. Językiem maszynowym procesora RTX2000 jest język FORTH.

Ten sam procesor RTX2000 został wykorzystany w sondzie Philae, która próbowała wylądować na komecie.

Wybór języka FORTH do zastosowań profesjonalnych okazuje się interesujący, jeśli każde słowo potraktujemy jak czarną skrzynkę. Każde słowo musi być proste, dlatego ma dość krótką definicję i zależy od kilku parametrów.

Podczas fazy debugowania łatwo jest przetestować wszystkie możliwe wartości przetwarzane przez to słowo. Gdy słowo to stanie się całkowicie niezawodne, staje się czarną skrzynką, czyli funkcją, co do której mamy nieograniczone zaufanie co do jego prawidłowego funkcjonowania. Od słowa do słowa łatwiej jest sprawić, by złożony program był niezawodny w FORTH niż w jakimkolwiek innym języku programowania.

Ale jeśli brakuje nam dyscypliny, jeśli budujemy elektrownie gazowe, bardzo łatwo jest też trafić na aplikację, która będzie słabo działać, a nawet całkowicie się załamie!

Wreszcie, w języku FORTH możliwe jest zapisanie zdefiniowanych przez Ciebie słów w dowolnym ludzkim języku. Jednak liczba możliwych do użycia znaków jest ograniczona do zestawu znaków ASCII od 33 do 127. Oto jak moglibyśmy symbolicznie przepisać słowa **high** i **low** :

```
\ Aktywny pin portu, innych nie zmieniaj.
: __/ ( pinmask portadr -- )
    mset
  ;
\ Wyłącz jeden pin portu, nie zmieniaj pozostałych.
: \__ ( pinmask portadr -- )
    mclr
  ;
```

Od tego momentu, aby włączyć diodę LED, możesz wpisać:

```
_O_ __/ \ Światła LED
```

Tak! Sekwencja **_O_ __/** jest w FORTH języku!

Dzięki ESP32forth masz do dyspozycji wszystkie znaki, które mogą utworzyć słowo FORTH :

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

Dobre programowanie.

Prawdziwy 32-bitowy FORTH z ESP32Forth

ESP32Forth to prawdziwy 32-bitowy FORTH. Co to znaczy ?

Język FORTH sprzyja manipulacji wartościami całkowitymi. Wartościami tymi mogą być wartości literalne, adresy pamięci, zawartość rejestrów itp.

Wartości na stosie danych

Po uruchomieniu ESP32Forth dostępny jest interpreter FORTH. Jeśli wpiszesz dowolną liczbę, zostanie ona upuszczona na stos jako 32-bitowa liczba całkowita:

```
35
```

Jeśli ułożymy inną wartość, ona również zostanie ułożona. Poprzednia wartość zostanie przesunięta o jedną pozycję w dół:

```
45
```

Aby dodać te dwie wartości, używamy tutaj słowa **+** :

```
+
```

Nasze dwie 32-bitowe wartości całkowite są sumowane i wynik jest upuszczany na stos. Aby wyświetlić ten wynik, użyjemy słowa **.** :

```
. \ wyświetla 80
```

W języku FORTH możemy skoncentrować wszystkie te operacje w jednym wierszu:

```
35 45 +.   \ wyświetlacz 80
```

W przeciwieństwie do języka C, nie definiujemy typu **int8** , **int16** lub **int32** .

W ESP32Forth znak ASCII będzie oznaczony 32-bitową liczbą całkowitą, ale której wartość będzie ograniczona [32..256[. Przykład :

```
67 emituj   \  wyświetlacz C
```

Wartości w pamięci

ESP32Forth umożliwia definiowanie stałych i zmiennych. Ich zawartość będzie zawsze w formacie 32-bitowym. Są jednak sytuacje, w których niekoniecznie nam to odpowiada. Weźmy prosty przykład definiujący alfabet Morse'a. Potrzebujemy tylko kilku bajtów:

- jeden określający liczbę znaków alfabetu Morse'a
- jeden lub więcej bajtów na każdą literę alfabetu Morse'a

```
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,
```

Tutaj definiujemy tylko 3 słowa, `mA` , `mB` i `mC` . W każdym słowie przechowywanych jest kilka bajtów. Pytanie brzmi: w jaki sposób odzyskamy informacje zawarte w tych słowach?

Wykonanie jednego z tych słów powoduje złożenie 32-bitowej wartości, która odpowiada adresowi pamięci, w którym przechowujemy informacje alfabetu Morse'a. To jest słowo `c@` , którego użyjemy do wyodrębnienia alfabetu Morse'a z każdej litery:

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

Pierwszy wyodrębniony w ten sposób bajt zostanie użyty do zarządzania pętlą wyświetlającą alfabet Morse'a litery:

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

Z pewnością jest mnóstwo bardziej eleganckich przykładów. Tutaj chcemy pokazać sposób manipulowania wartościami 8-bitowymi, czyli naszymi bajtami, podczas gdy my używamy tych bajtów na stosie 32-bitowym.

Przetwarzanie tekstu w zależności od rozmiaru lub typu danych

We wszystkich innych językach mamy słowo ogólne, takie jak `echo` (w PHP), które wyświetla dowolny typ danych. Niezależnie od tego, czy jest to liczba całkowita, rzeczywista, czy ciąg znaków, zawsze używamy tego samego słowa. Przykład w języku PHP :

```
$bread = "Pieczony chleb";
```

```
$price = 2.30;
echo $bread . " : " . $price;
// affiche   Pieczony chleb: 2.30
```

Dla wszystkich programistów taki sposób działania to STANDARD! Jak więc FORTH zrobiłby ten przykład w PHP ?

```
: chleb s" Chleb gotowany" ;
: cena s" 2.30" ;
chleb type   s" : " type    cena type
\ wyświetla   Chleb gotowany: 2.30
```

`type` słowa mówi nam, że właśnie przetworzyliśmy ciąg znaków.

Tam, gdzie PHP (lub jakikolwiek inny język) ma funkcję ogólną i parser, FORTH kompensuje jednym typem danych, ale dostosowanymi metodami przetwarzania, które informują nas o naturze przetwarzanych danych.

Oto zupełnie trywialny przypadek FORTH, wyświetlający liczbę sekund w formacie GG:MM:SS :

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

Podoba mi się ten przykład, ponieważ jak dotąd **ŻADEN INNY JĘZYK PROGRAMOWANIA nie** jest w stanie wykonać konwersji HH:MM:SS tak elegancko i zwięźle.

Zrozumiałeś, sekret FORTH tkwi w jego słownictwie.

Wniosek

FORTH nie ma możliwości wpisywania danych. Wszystkie dane przechodzą przez stos danych. Każda pozycja na stosie jest ZAWSZE 32-bitową liczbą całkowitą !

To wszystko, co warto wiedzieć.

Puryści języków hiperstrukturalnych i pełnych gadatliwości, takich jak C czy Java, z pewnością okrzykną herezją. I tutaj pozwolę sobie odpowiedzieć na nie: po co wpisywać swoje dane ?

Ponieważ w tej prostocie kryje się siła FORTH: pojedynczy stos danych o nietypowanym formacie i bardzo prostych operacjach.

Pokażę Ci to, czego nie potrafi wiele innych języków programowania, zdefiniuję nowe słowa definicyjne :

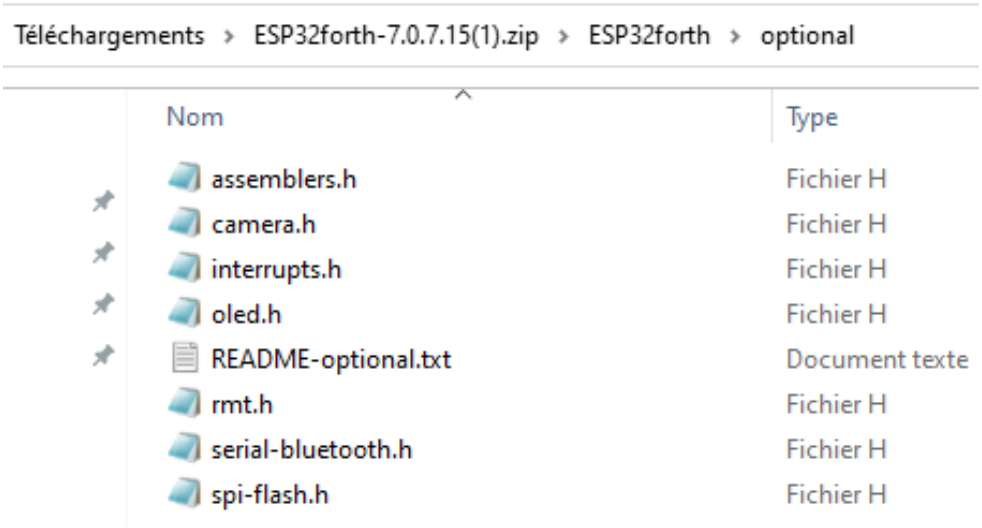
```
: morse: ( comp: c -- | exec -- )
    create
        c,
    does>
        dup 1+ swap c@ 0 do
            dup i + c@ emit
        loop
        drop space
  ;
2 morse: mA     char . c,   char - c,
4 morse: mB     char - c,   char . c,   char . c,   char . c,
4 morse: mC     char - c,   char . c, 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

Tutaj słowo **Morse:** stało się słowem definiującym, w taki sam sposób, jak **constant** lub **variable** ...

Ponieważ FORTH to coś więcej niż język programowania. Jest to metajęzyk, czyli język umożliwiający zbudowanie własnego języka programowania....

Instalowanie biblioteki OLED dla dysku SSD1306

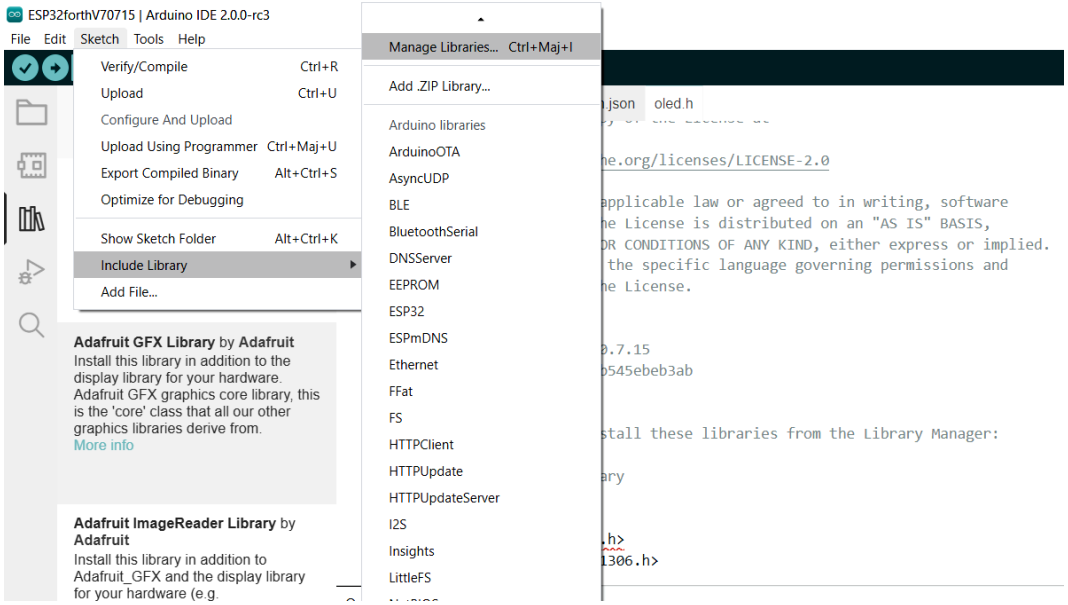
Od wersji ESP32forth 7.0.7.15 opcje są dostępne w folderze **optional** :



Aby mieć słownik `oled , skopiuj` plik **oled.h** do folderu zawierającego plik **ESP32forth.ino**.

Następnie uruchom ARDUINO IDE i wybierz najnowszy plik **ESP32forth.ino .**

Jeżeli biblioteka OLED nie została zainstalowana, w ARDUINO IDE kliknij *Sketch* i wybierz *Include Library*, a następnie wybierz *Manage Libraries*.



Na lewym pasku bocznym znajdź bibliotekę **Adafruit SSD1306 by Adafruit**.

Możesz teraz rozpocząć kompilację szkicu, klikając *Sketch* i wybierając *Upload*.

Po przesłaniu szkicu na płytkę ESP32 uruchom terminal TeraTerm. Sprawdź, czy obecne jest słownictwo **OLED** :

```
oled vlist   \ displays:
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
```

Generator liczb losowych

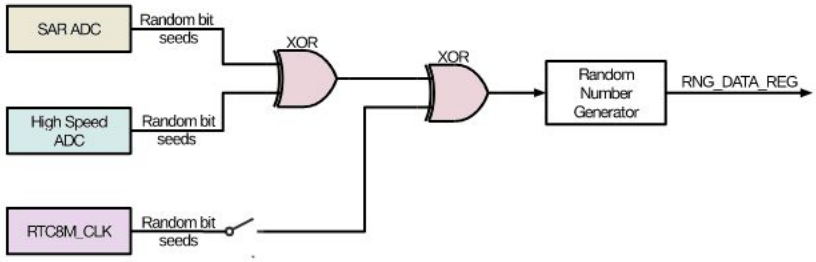
Charakterystyka

Generator liczb losowych generuje prawdziwe liczby losowe, czyli liczbę losową wygenerowaną w procesie fizycznym, a nie za pomocą algorytmu. Żadna liczba wygenerowana w określonym zakresie nie ma większego lub mniejszego prawdopodobieństwa pojawienia się niż jakakolwiek inna liczba.

Każda 32-bitowa wartość, którą system odczytuje z rejestru RNG_DATA_REG generatora liczb losowych, jest prawdziwą liczbą losową. Te prawdziwe liczby losowe są generowane na podstawie szumu termicznego w systemie i asynchronicznego przesunięcia zegara.

Szum termiczny pochodzi z szybkiego przetwornika ADC, przetwornika ADC SAR lub obu. Za każdym razem, gdy zostanie aktywowany szybki przetwornik ADC lub SAR ADC, strumienie bitów będą generowane i wprowadzane do generatora liczb losowych poprzez bramkę logiczną XOR jako losowe nasiona.

Gdy zegar RTC8M_CLK jest włączony dla rdzenia cyfrowego, generator liczb losowych będzie również próbkować RTC8M_CLK (8 MHz) jako losowy zarodek binarny. RTC8M_CLK jest asynchronicznym źródłem zegara i zwiększa entropię RNG poprzez wprowadzenie metastabilności obwodu. Jednakże, aby zapewnić maksymalną entropię, zaleca się również, aby zawsze włączać źródło ADC.



Gdy przetwornik ADC SAR generuje szum, generator liczb losowych zasilany jest entropią 2 bitów w cyklu zegara RTC8M_CLK (8 MHz), która jest generowana przez wewnętrzny oscylator RC (więcej szczegółów można znaleźć w rozdziale Resetowanie i zegar). Dlatego zaleca się odczytywanie rejestru **RNG_DATA_REG** z maksymalną częstotliwością 500 kHz, aby uzyskać maksymalną entropię.

Gdy szybki przetwornik ADC generuje szum, generator liczb losowych otrzymuje 2-bitową entropię w cyklu zegara APB, który zwykle wynosi 80 MHz. Dlatego zaleca się odczytywanie rejestru **RNG_DATA_REG** z maksymalną częstotliwością 5 MHz, aby uzyskać maksymalną entropię.

Próbkę danych o wielkości 2 GB, która jest odczytywana z generatora liczb losowych z częstotliwością 5 MHz przy włączonym tylko szybkim przetworniku ADC, została przetestowana przy użyciu zestawu testów Dieharder Random Number (wersja 3.31.1). Próbka przeszła wszystkie testy.

Procedura programowania

Korzystając z generatora liczb losowych, upewnij się, że dozwolone jest co najmniej SAR ADC, High Speed ADC lub RTC8M_CLK. W przeciwnym razie zwrócone zostaną liczby pseudolosowe.

- SAR ADC można aktywować za pomocą sterownika DIG ADC.
- Szybki ADC jest włączany automatycznie po włączeniu modułów Wi-Fi lub Bluetooth.
- RTC8M_CLK jest włączany poprzez ustawienie bitu RTC_CNTL_DIG_CLK8M_EN w rejestrze RTC_CNTL_CLK_CONF_REG.

Korzystając z generatora liczb losowych, należy kilkakrotnie przeczytać rejestr **RNG_DATA_REG** , aż wygenerowana zostanie wystarczająca liczba liczb losowych.

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

```
\ Dane liczbowe losowe
$3FF75144 constant RNG_DATA_REG

\ pobierz 32-bitowy losowy b=liczba
: rnd  ( -- x )
    RNG_DATA_REG L@
  ;

\ pobierz losową liczbę w przedziale [0..n-1]
: random ( n -- 0..n-1 )
    rnd swap mod
  ;
```

Funkcja RND w asemblerze XTENSA

Od wersji 7.0.7.4 ESP32forth posiada asembler XTENSA. Możliwe jest przepisanie słowa **RND** w asemblerze 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
```

Szczegółowa zawartość słowników ESP32forth

ESP32forth udostępnia liczne słowniki:

- **FORTH** to główne słownictwo;
- pewne słowniki są używane w mechanice wewnętrznej ESP32Forth, np. **internals** , **asm…**
- wiele słowników umożliwia zarządzanie określonymi portami lub akcesoriami, takimi jak **bluetooth** , **oled** , **spi** , **wifi** , **wire…**

Tutaj znajdziesz listę wszystkich słów zdefiniowanych w tych różnych słownikach. Niektóre słowa są oznaczone kolorowym linkiem:

align jest zwykłym słowem FORTH;

CONSTANT jest słowem definicyjnym;

begin oznacza strukturę kontrolną;

key jest słowem o odroczonym wykonaniu;

LED jest słowem zdefiniowanym przez **stałą** , **zmienną** lub **wartość** ;

registers oznacza słownictwo.

Słowa słownikowe **FORTH** są wyświetlane w kolejności alfabetycznej. W przypadku innych słowników słowa są prezentowane w kolejności wyświetlania.

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 l 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
```

Zasoby

Po angielsku

- **ESP32forth** strona prowadzona przez Brad NELSON, twórcę ESP32forth. Znajdziesz tam wszystkie wersje (ESP32, Windows, Web, Linux...)
  https://esp32forth.appspot.com/ESP32forth.html
-

Po francusku

- **ESP32 Forth** strona w dwóch językach (francuski, angielski) z dużą ilością przykładów
  https://esp32.arduino-forth.com/

GitHub

- **Ueforth** zasoby utrzymywane przez Brad NELSON. Zawiera wszystkie pliki źródłowe języków Forth i C dla ESP32forth
  https://github.com/flagxor/ueforth
- **ESP32forth** kody źródłowe i dokumentacja dla ESP32forter. Zasoby utrzymywane przez Marc PETREMANN
  https://github.com/MPETREMANN11/ESP32forth
- **ESP32forthStation** zasoby utrzymywane przez Ulrich HOFFMAN. Samodzielny komputer Forth z komputerem jednopłytkowym LillyGo TTGO VGA32 i ESP32forth.
  https://github.com/uho/ESP32forthStation
- **ESP32Forth** zasoby utrzymywane przez F. J. RUSSO
  https://github.com/FJRusso53/ESP32Forth
- **esp32forth-addons** zasoby utrzymywane przez Peter FORTH
  https://github.com/PeterForth/esp32forth-addons
- **Esp32forth-org** Repozytorium kodu dla członków grup Forth2020 i ESP32forth
  https://github.com/Esp32forth-org
-

Index