ESP32forth 帳本

版本 1.1 - 2023 年 10 月 26 日

作者

- Marc PETREMANN petremann@arduino-forth.com

合作者

- Vaclav POSELT
- Thomas SCHREIN

內容

介紹

自 2019 年以來，我管理了幾個致力於 ARDUINO 和 ESP32 卡的 FORTH 語言開發的網站，以及 eForth 網頁版：

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

這些網站有兩種語言版本：法語和英語。每年我都會支付主網站 **arduino-forth.com 的託管費用**。

遲早——而且盡可能晚——我將不再能夠確保這些網站的可持續性。其後果將是這些網站傳播的訊息消失。

這本書是我網站內容的彙編。它是從 Github 儲存庫免費分發的。這種分發方法將比網站具有更大的可持續性。

順便說一句，如果這些頁面的一些讀者希望做出貢獻，我們歡迎：

- 建議章節；
- 報告錯誤或建議更改；
- 幫忙翻譯**...**

翻譯幫助

谷歌翻譯可以讓你輕鬆翻譯文本，但會出現錯誤。所以我請求幫助來糾正翻譯。

在實務中，我提供了已經翻譯成 LibreOffice 格式的章節。如果您想幫助完成這些翻譯，您的角色只是更正並返回這些翻譯。

修改一章只需要很少的時間，從一個到幾個小時不等。

聯絡我： petremann@arduino-forth.com

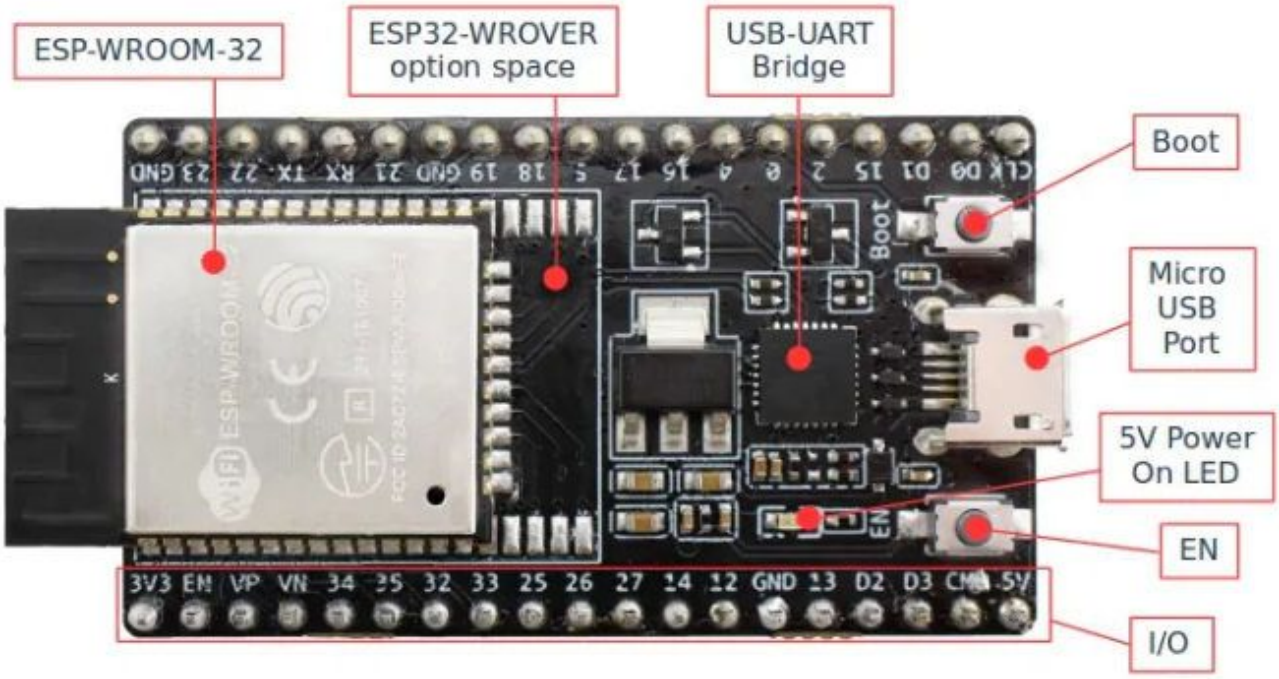
ESP32 卡的發現

推介會

ESP32 板不是 ARDUINO 板。然而，開發工具利用了 ARDUINO 生態系統的某些元素，例如 ARDUINO IDE。

優點

就可用連接埠數量而言，ESP32 卡位於 ARDUINO NANO 和 ARDUINO UNO 之間。基本型號有 38 個連接器：



ESP32 設備包括：

- 18 通道類比數位轉換器 (ADC)
- 3 個 SPI 接口
- 3 個 UART 接口
- 2 個 I2C 接口
- 16 個 PWM 輸出通道
- 2 個數位類比轉換器 (DAC)
- 2 個 I2S 接口

- 10 個電容感應 GPIO

ADC（類比數位轉換器）和 DAC（數位類比轉換器）功能被指派給特定的靜態引腳。但是，您可以決定哪些引腳是 UART、I2C、SPI、PWM 等。您只需在程式碼中分配它們即可。這要歸功於 ESP32 晶片的複用功能。

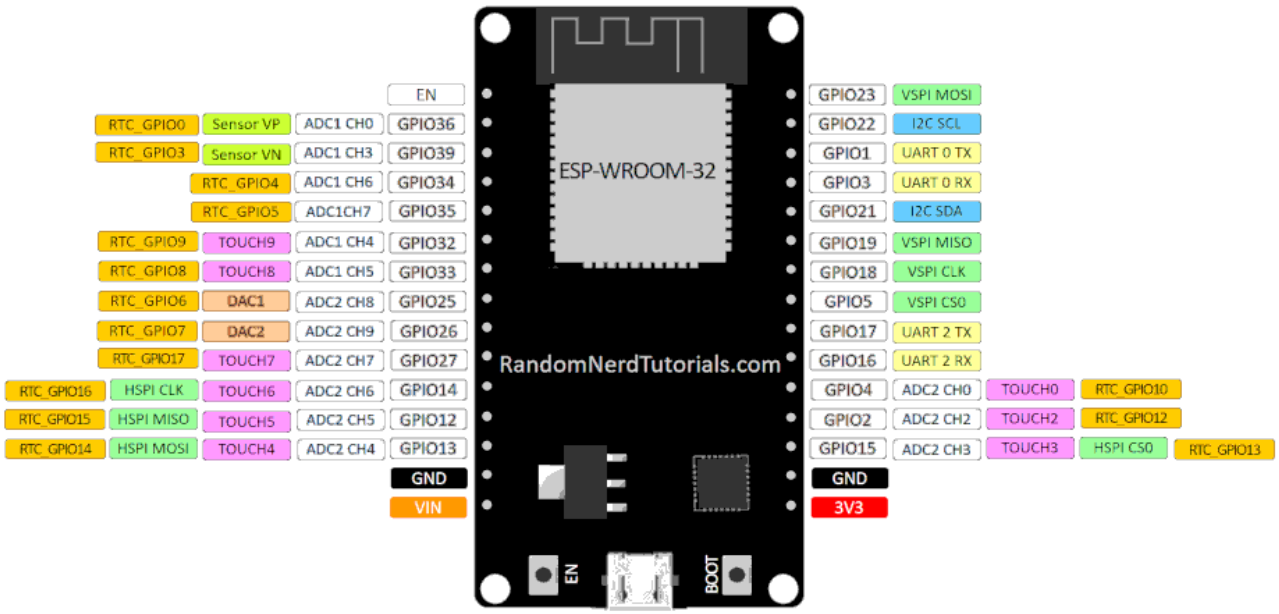
大多數連接器都有多種用途。

但 ESP32 板的與眾不同之處在於，它標配了 WiFi 和藍牙支持，而 ARDUINO 板僅以擴展的形式提供這些功能。

ESP32 上的 GPIO 輸入/輸出

以下是 ESP32 卡的照片，我們將從中解釋不同 GPIO 輸入/輸出的作用：

GPIO I/O 的位置和數量可能會根據卡品牌而變化。如果是這樣的話，那麼只有實體圖上出現的指示才是真實的。如圖，底行由左至右：
CLK、SD0、SD1、G15、G2、G0、G4、G16.....G22、G23、GND。



在此圖中，我們看到底部行以 3V3 開頭，而在照片中，此 I/O 位於頂部行的末尾。因此，不要依賴圖表，而是仔細檢查實體 ESP32 卡上的周邊設備和組件的正確連接，這一點非常重要。

基於 ESP32 的開發板除了電源接腳外，一般還有 33 個接腳。一些 GPIO 引腳具有一些特殊的功能：

通用輸入輸出介面	可能的名稱
6	SCK/CLK
7	SCK/CLK
8	SDO/SD0
9	SDI/SD1
10	SHD/SD2
11	CSC/CMD

如果你的 ESP32 卡有 I/O GPIO6、GPIO7、GPIO8、GPIO9、GPIO10、GPIO11，你絕對不應該使用它們，因為它們連接到 ESP32 的快閃記憶體。如果您使用它們，ESP32 將無法運作。

GPIO1(TX0) 和 GPIO3(RX0) I/O 用於透過 USB 連接埠與電腦進行 UART 通訊。如果您使用它們，您將無法再與該卡通訊。

GPIO36(VP)、GPIO39(VN)、GPIO34、GPIO35 I/O 只能用作輸入。它們也沒有內建的內部上拉和下拉電阻。

EN 端子可讓您透過外部導線控制 ESP32 的點火狀態。它連接到卡上的 EN 按鈕。當 ESP32 開啟時，電壓為 3.3V。如果我們將該引腳接地，ESP32 將關閉。當 ESP32 位於盒子中並且您希望能夠透過開關打開/關閉它時，您可以使用它。

ESP32 週邊設備

為了與模組、感測器或電子電路交互，ESP32 與任何微控制器一樣，擁有大量週邊設備。它們的數量比經典 Arduino 板上的數量還要多。

ESP32 有以下週邊：

- 3 個 UART 接口
- 2 個 I2C 接口
- 3 個 SPI 接口
- 16 個 PWM 輸出
- 10 個電容式感測器
- 18 個類比輸入 (ADC)
- 2 個 DAC 輸出

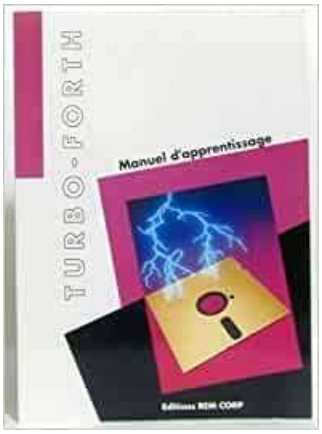
ESP32 在其基本操作過程中已經使用了一些週邊設備。因此，每個設備可能的介面較少。

為什麼在 ESP32 上使用 FORTH 語言程式設計?

前言

我自 1983 年以來一直使用 FORTH 進行程式設計。我於 1996 年停止使用 FORTH 進行程式設計。但我從未停止監視這種語言的演變。我於 2019 年重新開始在 ARDUINO 上使用 FlashForth 和 ESP32forth 進行程式設計。

我是幾本有關 FORTH 語言的書籍的合著者：



- 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)

使用 FORTH 語言程式設計一直是我的愛好，直到 1992 年，一家汽車行業分包商公司的經理聯繫了我。他們對 C 語言軟體開發關心，需要訂購一台工業自動機。

該公司的兩位軟體設計人員以 C 語言進行程式設計：準確地說是來自 Borland 的 TURBO-C。而且他們的程式碼不夠緊湊和快速，無法適應 64 KB 的 RAM 記憶體。那是 1992 年，快閃記憶體類型擴充還不存在。在這 64 KB RAM 中，我們必須容納 MS-DOS 3.0 和應用程式！

一個月來，C 語言開發人員一直在各個方向上扭轉這個問題，甚至使用 SOURCER（反彙編器）進行逆向工程，以消除可執行程式碼中非必要的部分。

我分析了向我提出的問題。從頭開始，我獨自一人在一周內創建了一個符合規範、完美運行的原型。從 1992 年到 1995 年的三年時間裡，我創建了該應用程式的多個版本，並在多家汽車製造商的組裝線上使用。

語言與應用之間的界限

所有程式語言共享如下：

- 解釋器和可執行原始程式碼：BASIC、PHP、MySQL、JavaScript 等...該應用程式包含在一個或多個檔案中，必要時將對其進行解釋。系統必須永久託管執行原始碼的解釋器；
- 編譯器和/或組譯器：C、Java 等。有些編譯器生成本機程式碼，也就是說可以在系統上專門執行。其他的，像是 Java，在 Java 虛擬機器上編譯可執行程式碼。

FORTH 語言是個例外。這包括：

- 能夠執行 FORTH 語言中任何單字的解釋器
- 能夠擴展 FORTH 單字字典的編譯器。

FORTH 字是什麼?

FORTH 單字指定由 ASCII 字元組成並可用於解釋和/或編譯的任何字典表達式：單字可讓您列出 FORTH 字典中的所有單字。

某些 FORTH 單字只能在編譯中使用：例如 `if else then` 。

對於 FORTH 語言，基本原則是我們不創建應用程式。在 FORTH 中，我們正在擴展字典！您定義的每個新單字都會與 FORTH 啟動時預先定義的所有單字一樣成為 FORTH 字典的一部分。例 ：

```
: typeToLoRa ( -- )
    0 echo !    \ desactive l'echo d'affichage du terminal
    ['] serial2-type is type
  ;
: typeToTerm ( -- )
    ['] default-type is type
    -1 echo !   \ active l'echo d'affichage du terminal
  ;
```

我們建立兩個新單字： `typeToLoRa` 和 `typeToTerm` ，這將完成預先定義單字的字典。

一個字就是一個函數?

是和不是。事實上，一個單字可以是一個常數、一個變數、一個函數…在我們的例子中，有以下序列：

```
: typeToLoRa ...代碼... ;
```

在 C 語言中會有等價的：

```
void typeToLoRa() { ...程式碼... }
```

在 FORTH 語言中，語言和應用之間沒有限制。

在 FORTH 中，與 C 語言一樣，您可以使用新單字定義中已定義的任何單字。

是的，但是為什麼是 FORTH 而不是 C?

我正期待著這個問題。

在 C 語言中，一個函數只能透過主函數 `main()` 來存取。如果該功能整合了多個附加功能，則在程式發生故障時很難發現參數錯誤。

相反，使用 FORTH 可以透過解釋器執行任何預先定義或由您定義的單詞，而無需遍歷程式的主單字。

FORTH 解譯器可透過終端機類型程式以及 ESP32 卡與 PC 之間的 USB 連結在 ESP32 卡上立即存取。

用 FORTH 語言編寫的程式的編譯是在 ESP32 卡中進行的，而不是在 PC 上進行的。沒有上傳。例 ：

```
: >gray ( n -- n' )
    dup 2/ xor      \ n' = n xor ( 1 right logical shift )
  ;
```

此定義透過複製/貼上的方式傳輸到終端中。 FORTH 解釋器/編譯器將解析流並編譯新單字 **>gray** 。

>gray 的定義中，我們看到序列 **dup 2/ xor** 。要測試此序列，只需在終端機中鍵入它即可。要運行 **>gray** ，只需在終端機中輸入該單詞，前面加上要轉換的數字。

FORTH 語言與 C 語言的比較

這是我最不喜歡的部分。我不喜歡將 FORTH 語言與 C 語言進行比較。但由於幾乎所有開發人員都使用 C 語言，所以我將嘗試這個練習。

用 C 語言 **if()** 進行的測試：

```
if(j > 13){                 // 如果接收到所有位
    rc5_ok = 1;             // 解碼過程正常
    detachInterrupt(0);     // 禁止外部中斷（INT0）
    return;
}
```

用 FORTH 語言的 **if** 進行測試（程式碼片段）：

```
var-j @ 13 >            \ 如果接收到所有位
    if
        1 rc5_ok !      \ 解碼過程正常
        di              \ 禁用外部中斷（INT0）
        exit
    then
```

下面是 C 語言中暫存器的初始化：

```
void setup() {
  //  Configuration du module Timer1
  TCCR1A = 0;
  TCCR1B = 0;           // Desactive le module Timer1
  TCNT1  = 0;           // Definit valeur préchargement Timer1 sur 0
(reset)
  TIMSK1 = 1;           // activer interruption de debordement Timer1
}
```

FORTH 語言中的相同定義：

```
: setup ( -- )
  \ Configuration du module Timer1
  0 TCCR1A !
  0 TCCR1B !      \ Desactive le module Timer1
  0 TCNT1  !      \ Définit valeur préchargement Timer1 sur 0 (reset)
  1 TIMSK1 !      \ activer interruption de debordement Timer1
;
```

與 C 語言相比，FORTH 可以讓您做什麼

我們知道 FORTH 可以立即存取字典中的所有單詞，但不僅限於此。透過解釋器，我們還可以存取 ESP32 卡的整個記憶體。連接到安裝了 FlashForth 的 ARDUINO 板，然後只需鍵入：

```
hex here 100 dump
```

您應該在終端螢幕上找到它：

```
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
```

這對應於閃存的內容。

而 C 語言卻做不到這一點？

是的，但不像 FORTH 語言那樣簡單和互動。

讓我們來看另一個案例，凸顯 FORTH 語言非凡的緊湊性...

但為什麼是堆疊而不是變數呢？

堆疊是幾乎所有微控制器和微處理器上實現的機制。即使 C 語言也利用堆疊，但您無權存取它。

只有 FORTH 語言才能完全存取資料堆疊。例如，要進行加法，我們將兩個值堆疊起來，執行加法，然後顯示結果： `2 5 + .` 顯示 `7` 。

這有點不穩定，但是當您了解資料堆疊的機制時，您會非常欣賞它的強大效率。

資料堆疊允許資料在 FORTH 個字之間傳遞，比透過 C 語言或使用變數的任何其他語言中處理變數要快得多。

你確信嗎？

就我個人而言，我懷疑這一章是否會不可挽回地讓您轉向使用 FORTH 語言進行程式設計。當您嘗試掌握 ESP32 卡時，您有兩種選擇：

- 使用 C 語言進行程式設計並使用大量可用的程式庫。但您仍將無法使用這些函式庫的功能。將程式碼改編為 C 語言需要真正的 C 語言程式設計知識並掌握 ESP32 卡的架構。開發複雜的程式永遠是個問題。
- 嘗試第四次冒險，探索一個新的、令人興奮的世界。當然，這並不容易。您需要深入了解 ESP32 卡的架構、暫存器、暫存器標誌。作為回報，您將能夠獲得完全適合您的專案的程式設計。

有沒有用 FORTH 寫的專業應用程式?

哦是的！從哈伯太空望遠鏡開始，其某些組件是用 FORTH 語言編寫的。

德國 TGV ICE（Intercity Express）使用 RTX2000 處理器透過功率半導體控制馬達。 RTX2000 處理器的機器語言是 FORTH 語言。

試圖降落在彗星上的菲萊探測器也使用了相同的 RTX2000 處理器。

如果我們將每個單字視為一個黑盒子，那麼為專業應用程式選擇 FORTH 語言就會變得很有趣。每個單字必須很簡單，因此具有相當短的定義並且依賴很少的參數。

在偵錯階段，測試該字處理的所有可能值變得容易。一旦變得完全可靠，這個詞就變成了一個黑盒子，也就是說，我們對其正常運作擁有無限信心的功能。從字到字，用 FORTH 比用任何其他程式語言更容易使複雜的程式變得可靠。

但如果我們缺乏嚴謹性，如果我們建造天然氣工廠，也很容易得到一個運作不佳的應用程序，甚至完全崩潰！

最後，可以用 FORTH 語言編寫您用任何人類語言定義的單字。然而，可用的字元僅限於 33 到 127 之間的 ASCII 字元集。以下是我們如何象徵性地重寫單字 **high** 和 **low** ：

```
\ Active broche de port, ne changez pas les autres.
: __/ ( pinmask portadr -- )
    mset
  ;
\ Desactivez une broche de port, ne change pas les autres.
: \__ ( pinmask portadr -- )
    mclr
  ;
```

從此時起，要開啟 LED，您可以輸入：

```
_O_  __/     \ LED on
```

是的！序列 **_O_ __/** 採用 FORTH 語言！

對於 ESP32forth，以下是您可以使用的可以組成 FORTH 單字的所有字元：

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

良好的編程。

ESP32forth 的實數

如果我們用 FORTH 語言測試運算 `1 3 /` ，結果將為 0。

這並不奇怪。基本上，ESP32forth 僅透過資料堆疊使用 32 位元整數。整數具有某些優點：

- 處理速度；
- 計算結果在迭代時沒有漂移風險；
- 幾乎適用於所有情況。

即使在三角計算中，我們也可以使用整數表。只需建立一個包含 90 個值的表，其中每個值對應於角度的正弦值乘以 1000。

但整數也有限制：

- 簡單除法計算的不可能結果，例如我們的 1/3 範例；
- 需要複雜的操作來應用物理公式。

從 7.0.6.5 版本開始，ESP32forth 包含了處理實數的運算子。

實數又稱浮點數。

真正的 ESP32forth

為了區分實數，它們必須以字母“e”結尾：

```
3           \ push 3 on the normal stack
3e          \ push 3 on the real stack
5.21e f.    \ display 5.210000
```

就是這個詞。它允許您顯示位於實數堆疊頂部的實數。

ESP32forth 的實數精度

`set- precision` 一詞可讓您指示小數點後顯示的小數位數。讓我們用常數 `pi` 來看看：

```
pi f.       \ display 3.141592
4 set-precision
pi f.       \ display 3.1415
```

ESP32forth 處理實數的極限精度為小數點後六位：

```
12 set-precision
1.987654321e f.       \ display 1.987654668777
```

如果我們將實數的顯示精度降低到 6 以下，計算仍然會以小數點後 6 位的精度進行。

實數常數和變數

實數常數用單字 **fconstant** 定義：

```
0.693147e fconstant ln2   \ natural logarithm of 2
```

實數變數用單字 **fvariable** 定義：

```
fvariable intensity
170e 12e F/ intensity SF!   \ I=P/U   ---  P=170w   U=12V
intensity SF@ f.            \ display 14.166669
```

注意：所有實數都通過**實數棧**。對於實數變量，只有指向實數值的位址才會通過資料堆疊。

這個字**!** 將實際值儲存在其記憶體位址指向的位址或變數中。執行實數變數會將記憶體位址放置在經典資料堆疊上。

字 **SF@**堆疊其記憶體位址指向的實際值。

實數的算術運算符

ESP32Forth 有四個算術運算子 **F+ F- F* F/** ：

```
1.23e 4.56e F+ f.   \ display 5.790000      1.23-4.56
1.23e 4.56e F- f.   \ display -3.330000     1.23-4.56
1.23e 4.56e F* f.   \ display 5.608800      1.23*4.56
1.23e 4.56e F/ f.   \ display 0.269736      1.23/4.56
```

ESP32forth 還有這樣的話：

- **1/F** 計算實數的倒數；
- **fsqrt** 計算實數的平方根。

```
5e 1/F f.       \ display 0.200000      1/5
5e fsqrt f.     \ display 2.236068      sqrt(5)
```

實數的數學運算符

ESP32forth 有幾個數學運算子：

- **F****計算實數 r_val 的 r_exp 次方
- **FATAN2** 從切線計算弧度角。
- **FCOS** (r1 -- r2) 計算以弧度表示的角度的餘弦。
- **FEXP** (ln-r -- r) 計算 e EXP r 對應的實數
- **FLN** (r -- ln-r) 計算實數的自然對數。
- **FSIN** (r1 - r2) 計算以弧度表示的角度的正弦值。

- **FSINCOS** (r1 -- rcos rsin) 計算以弧度表示的角度的餘弦和正弦。

一些例子 ：

```
  2e 3e f** f.     \ display 8.000000
  2e 4e f** f.     \ display 16.000000
 10e 1.5e f** f.   \ display 31.622776

4.605170e FEXP F.     \ display 100.000018

pi 4e f/
FSINCOS f. f.    \ display 0.707106 0.707106
pi 2e f/
FSINCOS f. f.    \ display 0.000000 1.000000
```

實數上的邏輯運算符

ESP32forth 還允許您對真實資料進行邏輯測試：

- **F0<** (r -- fl) 測試實數是否小於零。
- **F0=** (r -- fl) 表示實數為零時為真。
- **f<** (r1 r2 -- fl) 若 r1 < r2，則 fl 為真。
- **f<=** (r1 r2 -- fl) 如果 r1 <= r2，則 fl 為真。
- **f<>** (r1 r2 -- fl) 如果 r1 <> r2，則 fl 為真。
- **f=** (r1 r2 -- fl) 若 r1 = r2，則 fl 為真。
- **f>** (r1 r2 -- fl) 如果 r1 > r2，則 fl 為真。
- **f>=** (r1 r2 -- fl) 如果 r1 >= r2，則 fl 為真。

整數 ↔ 實數轉換

ESP32forth 有兩個字用於將整數轉換為實數，反之亦然：

- **F>S** (r -- n) 將實數轉換為整數。如果實數有小數部分，則將整數部分保留在資料堆疊上。
- **S>F** (n -- r: r) 將整數轉換為實數並將該實數傳送到實數堆疊。

例 ：

```
35 S>F
F.   \ display 35.000000

3.5e F>S .   \ display 3
```

新增 SPI 庫

ESP32forth 中並未原生實作 SPI 函式庫。要安裝它，您必須先建立 **spi.h 文件**，該文件必須安裝在與包含 **ESP42forth.ino 文件的資料夾相同的資料夾中**。

spi.h 文件內容（C 語言）：

```
# include <SPI.h>

#define OPTIONAL_SPI_VOCABULARY V(spi)
#define OPTIONAL_SPI_SUPPORT \
  XV(internals, "spi-source", SPI_SOURCE, \
      PUSH spi_source; PUSH sizeof(spi_source) - 1) \
  XV(spi, "SPI.begin", SPI_BEGIN, SPI.begin((int8_t) n3, (int8_t) n2, (int8_t) n1, (int8_t)
n0); DROPn(4)) \
  XV(spi, "SPI.end", SPI_END, SPI.end();) \
  XV(spi, "SPI.setHwCs", SPI_SETHWCS, SPI.setHwCs((boolean) n0); DROP) \
  XV(spi, "SPI.setBitOrder", SPI_SETBITORDER, SPI.setBitOrder((uint8_t) n0); DROP) \
  XV(spi, "SPI.setDataMode", SPI_SETDATAMODE, SPI.setDataMode((uint8_t) n0); DROP) \
  XV(spi, "SPI.setFrequency", SPI_SETFREQUENCY, SPI.setFrequency((uint32_t) n0); DROP) \
  XV(spi, "SPI.setClockDivider", SPI_SETCLOCKDIVIDER, SPI.setClockDivider((uint32_t) n0);
DROP) \
  XV(spi, "SPI.getClockDivider", SPI_GETCLOCKDIVIDER, PUSH SPI.getClockDivider();) \
  XV(spi, "SPI.transfer",   SPI_TRANSFER, SPI.transfer((uint8_t *) n1, (uint32_t) n0);
DROPn(2)) \
  XV(spi, "SPI.transfer8",  SPI_TRANSFER_8,  PUSH (uint8_t)  SPI.transfer((uint8_t) n0);
NIP) \
  XV(spi, "SPI.transfer16", SPI_TRANSFER_16, PUSH (uint16_t) SPI.transfer16((uint16_t) n0);
NIP) \
  XV(spi, "SPI.transfer32", SPI_TRANSFER_32, PUSH (uint32_t) SPI.transfer32((uint32_t) n0);
NIP) \
  XV(spi, "SPI.transferBytes", SPI_TRANSFER_BYTES, SPI.transferBytes((const uint8_t *) n2,
(uint8_t *) n1, (uint32_t) n0); DROPn(3)) \
  XV(spi, "SPI.transferBits", SPI_TRANSFER_BITES, SPI.transferBits((uint32_t) n2, (uint32_t *)
n1, (uint8_t) n0); DROPn(3)) \
  XV(spi, "SPI.write", SPI_WRITE, SPI.write((uint8_t) n0); DROP) \
  XV(spi, "SPI.write16", SPI_WRITE16, SPI.write16((uint16_t) n0); DROP) \
  XV(spi, "SPI.write32", SPI_WRITE32, SPI.write32((uint32_t) n0); DROP) \
  XV(spi, "SPI.writeBytes", SPI_WRITE_BYTES, SPI.writeBytes((const uint8_t *) n1, (uint32_t)
n0); DROPn(2)) \
  XV(spi, "SPI.writePixels", SPI_WRITE_PIXELS, SPI.writePixels((const void *) n1, (uint32_t)
n0); DROPn(2)) \
  XV(spi, "SPI.writePattern", SPI_WRITE_PATTERN, SPI.writePattern((const uint8_t *) n2,
(uint8_t) n1, (uint32_t) n0); DROPn(3))

const char spi_source[] = R"""(
vocabulary spi   spi definitions
transfer spi-builtins
forth definitions
)""";
```

完整文件也可以在這裡找到： https://github.com/MPETREMANN11/ESP32forth/blob/main/optional/spi.h

ESP32forth.ino 檔案的更改

ESP32forth.ino 檔案進行一些更改，則 spi.h 檔案的內容無法整合到 ESP32forth 中。以下是對此文件進行的一些修改。這些變更是在版本 7.0.7.15 上進行的，但應該適用於其他最近或未來的版本。

第一次修改

新增紅色代碼：

```
#define VOCABULARY_LIST \
  V(forth) V(internals) \
  V(rtos) V(SPIFFS) V(serial) V(SD) V(SD_MMC) V(ESP) \
  V(ledc) V(Wire) V(WiFi) V(sockets) \
  OPTIONAL_CAMERA_VOCABULARY \
  OPTIONAL_BLUETOOTH_VOCABULARY \
  OPTIONAL_INTERRUPTS_VOCABULARIES \
  OPTIONAL_OLED_VOCABULARY \
  OPTIONAL_SPI_VOCABULARY \
  OPTIONAL_RMT_VOCABULARY \
  OPTIONAL_SPI_FLASH_VOCABULARY \
  USER_VOCABULARIES
```

第二次修改

這段程式碼後面加上紅色：

```
// Hook to pull in optional Oled support.
# if __has_include("oled.h")
#  include "oled.h"
# else
#  define OPTIONAL_OLED_VOCABULARY
#  define OPTIONAL_OLED_SUPPORT
# endif

// Hook to pull in optional SPI support.
# if __has_include("spi.h")
#  include "spi.h"
# else
#  define OPTIONAL_SPI_VOCABULARY
#  define OPTIONAL_SPI_SUPPORT
# endif
```

第三次修改

添加紅色：

```
#define EXTERNAL_OPTIONAL_MODULE_SUPPORT \
  OPTIONAL_ASSEMBLERS_SUPPORT \
  OPTIONAL_CAMERA_SUPPORT \
  OPTIONAL_INTERRUPTS_SUPPORT \
  OPTIONAL_OLED_SUPPORT \
  OPTIONAL_SPI_SUPPORT \
  OPTIONAL_RMT_SUPPORT \
  OPTIONAL_SERIAL_BLUETOOTH_SUPPORT \
```

```
  OPTIONAL_SPI_FLASH_SUPPORT
```

第四次修改

添加紅色：

```
internals DEFINED? oled-source [IF]
  oled-source evaluate
[THEN] forth

internals DEFINED? spi-source [IF]
  spi-source evaluate
[THEN] forth
```

如果您仔細遵循這些說明，您將能夠使用 ARDUINO IDE 編譯 ESP32forth 並將其上傳到 ESP32 開發板。完成這些操作後，啟動終端。您需要找到 ESP32forth 歡迎提示。類型：

```
大三角帆列表
```

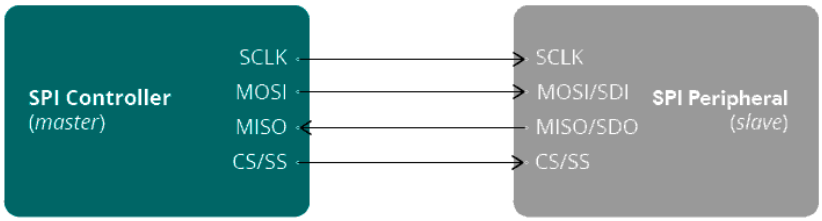
spi 詞彙表中定義的單字：

```
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
```

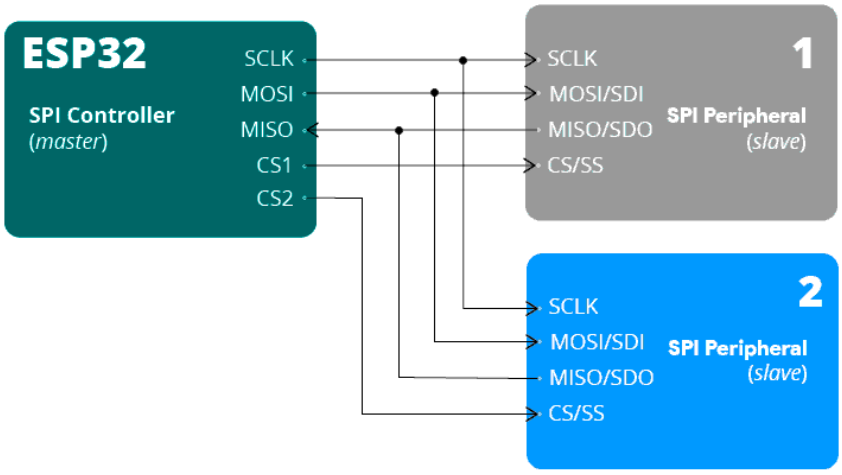
現在您可以透過 SPI 連接埠驅動擴展，例如 MAX7219 LED 顯示器。

與 MAX7219 顯示模組通信

在 SPI 通訊中，總有一個控制外設的主機（也稱為*從機*）。資料可以同時發送和接收。這意味著主機可以向從機發送數據，並且從機可以同時向主機發送數據。



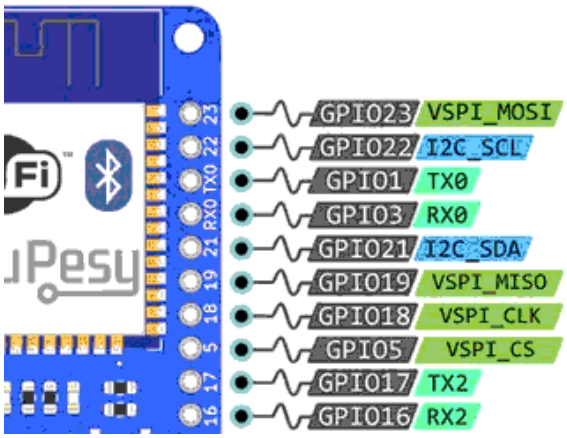
你可以有幾個奴隸。從設備可以是感測器、顯示器、microSD 卡等，或其他微控制器。這意味著您可以將 ESP32 連接到**多個裝置**。



透過將 CS1 或 CS2 選擇器設定為低電位來選擇從機。有多少從屬設備需要管理，就需要多少個 CS 選擇器。

找到 ESP32 板上的 SPI 端口

ESP32 板上有兩個 SPI 連接埠：HSPI 和 VSPI。我們將管理的 SPI 連接埠是引腳前綴為 VSPI 的連接埠：



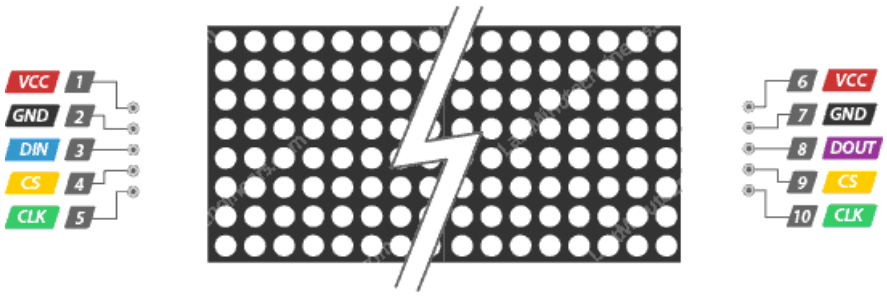
因此，使用 ESP32forth，我們可以定義指向這些 VSPI 引腳的常數：

```
\ 定義 VSPI 腳
19 constant VSPI_MISO
23 constant VSPI_MOSI
18 constant VSPI_SCLK
05 constant VSPI_CS
```

為了與 MAX7219 顯示模組通信，我們只需連接 VSPI_MOSI、VSPI_SCLK 和 VSPI_CS 引腳。

MAX7219 顯示模組上的 SPI 連接器

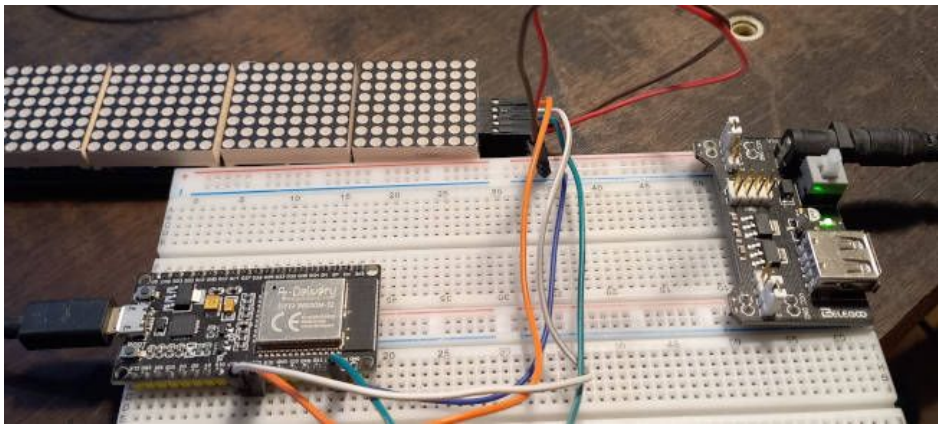
以下是 MAX7219 模組上的 SPI 連接埠連接器圖：



MAX7219 模組與 ESP32 卡之間的連接：

```
            MAX7219                ESP32
              DIN     <---->    VSPI_MOSI
              CS      <---->    VSPI_CS
              CLK     <---->    VSPI_SCLK
```

VCC 和 GND 連接器連接到外部電源：



此外部電源的 GND 部分與 ESP32 卡的 GND 引腳共用。

SPI 埠軟體層

所有用於管理 SPI 連接埠的單字都已在 **spi** 詞彙表中可用。

唯一需要定義的是 SPI 埠的初始化：

```
\ 定義 SPI 連接埠頻率
4000000 constant SPI_FREQ

\選擇 SPI 詞彙
only FORTH  SPI also

\ 初始化 SPI 端口
: 初始化.VSPI ( -- )
: init.VSPI ( -- )
    VSPI_CS OUTPUT pinMode
    VSPI_SCLK VSPI_MISO VSPI_MOSI VSPI_CS SPI.begin
    SPI_FREQ SPI.setFrequency
  ;
```

現在我們可以使用 MAX7219 顯示模組了。

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
```

資源

用英語

- **ESP32forth** 頁面由 ESP32forth 的創建者 Brad NELSON 維護。您將在那裡找到所有版本（ESP32、Windows、Web、Linux...）
  https://esp32forth.appspot.com/ESP32forth.html
-

法語

- **ESP32 Forth** 網站有兩種語言（法語、英語），有許多範例
  https://esp32.arduino-forth.com/

GitHub

- **尤福斯** 資源由 Brad NELSON 維護。包含 ESP32forth 的所有 Forth 和 C 語言原始檔
  https://github.com/flagxor/ueforth
- **ESP32forth** ESP32forth 的原始碼和文件。 Marc PETREMANN 維護的資源
  https://github.com/MPETREMANN11/ESP32forth
- **ESP32forthStation** 資源由 Ulrich HOFFMAN 維護。帶有 LillyGo TTGO VGA32 單板計算機和 ESP32forth 的獨立 Forth 計算機。
  https://github.com/uho/ESP32forthStation
- **ESP32Forth** FJ RUSSO 維護的資源
  https://github.com/FJRusso53/ESP32Forth
- **esp32forth 插件** Peter FORTH 維護的資源
  https://github.com/PeterForth/esp32forth-addons
- **Esp32forth-org** Forth2020 和 ESp32forth 組成員的程式碼儲存庫
  https://github.com/Esp32forth-org
-

詞彙索引