

# ESP32forth 帳本

版本 **1.3** - 2023 年 **11** 月 **7** 日



## 作者

- Marc PETREMANN      [petremann@arduino-forth.com](mailto:petremann@arduino-forth.com)

## 合作者

- Bob EDWARDS
- Vaclav POSELT
- Thomas SCHREIN

# 內容

作者.....	1
合作者.....	1
介紹.....	<b>5</b>
翻譯幫助.....	5
<b>ESP32 卡的發現.....</b>	<b>6</b>
推介會.....	6
優點.....	6
ESP32 上的 GPIO 輸入/輸出.....	7
ESP32 週邊設備.....	9
安裝 <b>ESP32Forth</b> .....	<b>11</b>
下載 ESP32forth.....	11
編譯並安裝 ESP32forth.....	11
ESP32 WROOM 的設置.....	13
開始編譯.....	13
修復上傳連線錯誤.....	<b>15</b>
為什麼在 <b>ESP32</b> 上使用 <b>FORTH</b> 語言程式設計？ .....	<b>18</b>
前言.....	18
語言與應用之間的界限.....	18
FORTH 字是什麼？ .....	19
一個字就是一個函數？ .....	19
FORTH 語言與 C 語言的比較.....	20
與 C 語言相比，FORTH 可以讓您做什麼.....	20
但為什麼是堆疊而不是變數呢？ .....	21
你確信嗎？ .....	21
有沒有用 FORTH 寫的專業應用程式？ .....	21
使用 <b>ESP32Forth</b> 的真正 <b>32 位元 FORTH</b> .....	<b>23</b>
資料堆疊上的值.....	23
記憶體中的值.....	23
根據資料大小或類型進行文字處理.....	24
結論.....	25
字典/堆疊/變數/常數.....	<b>26</b>
展開字典.....	26
字典管理.....	26
堆疊和逆波蘭表示法.....	26
處理參數堆疊.....	27
返回堆疊及其用途.....	28
記憶體使用情況.....	28
變數.....	28
常數.....	29
偽常數值.....	29
記憶體分配的基本工具.....	29

終端機上的文字顏色和顯示位置.....	31
端子的 ANSI 編碼.....	31
文字著色.....	31
顯示位置.....	33
<b>ESP32Forth 的局部變數.....</b>	<b>35</b>
介紹.....	35
假堆疊註釋.....	35
對局部變數的操作.....	36
<b>ESP32forth 的實數.....</b>	<b>38</b>
真正的 ESP32forth.....	38
ESP32forth 的實數精度.....	38
實數常數和變數.....	39
實數的算術運算符.....	39
實數的數學運算符.....	39
實數上的邏輯運算符.....	40
整數 ↔ 實數轉換.....	40
將麵包板適應 <b>ESP32</b> 板.....	<b>42</b>
ESP32 測試板.....	42
建造適合 ESP32 板的麵包板.....	42
新增 <b>SPI</b> 庫.....	<b>45</b>
ESP32forth.ino 檔案的更改.....	46
第一次修改.....	46
第二次修改.....	46
第三次修改.....	46
第四次修改.....	47
與 MAX7219 顯示模組通信.....	47
找到 ESP32 板上的 SPI 端口.....	48
MAX7219 顯示模組上的 SPI 連接器.....	48
SPI 埠軟體層.....	49
安裝 <b>HTTP</b> 用戶端.....	<b>51</b>
編輯 ESP32forth.ino 文件.....	51
HTTP 用戶端測試.....	52
<b>ESP32forth 詞彙詳細內容.....</b>	<b>55</b>
Version v 7.0.7.15.....	55
FORTH.....	55
asm.....	56
bluetooth.....	56
editor.....	57
ESP.....	57
httpd.....	57
insides.....	57
internals.....	57
interrupts.....	58
ledc.....	58
oled.....	58

registers.....	58
riscv.....	58
rtos.....	59
SD.....	59
SD_MMC.....	59
Serial.....	59
sockets.....	59
spi.....	59
SPIFFS.....	59
streams.....	59
structures.....	59
tasks.....	59
telnetd.....	60
visual.....	60
web-interface.....	60
WiFi.....	60
xtensa.....	60
資源.....	<b>62</b>
用英語.....	62
法語.....	62
GitHub.....	62

# 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

谷歌翻譯可以让你輕鬆翻譯文本，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](mailto:petremann@arduino-forth.com)

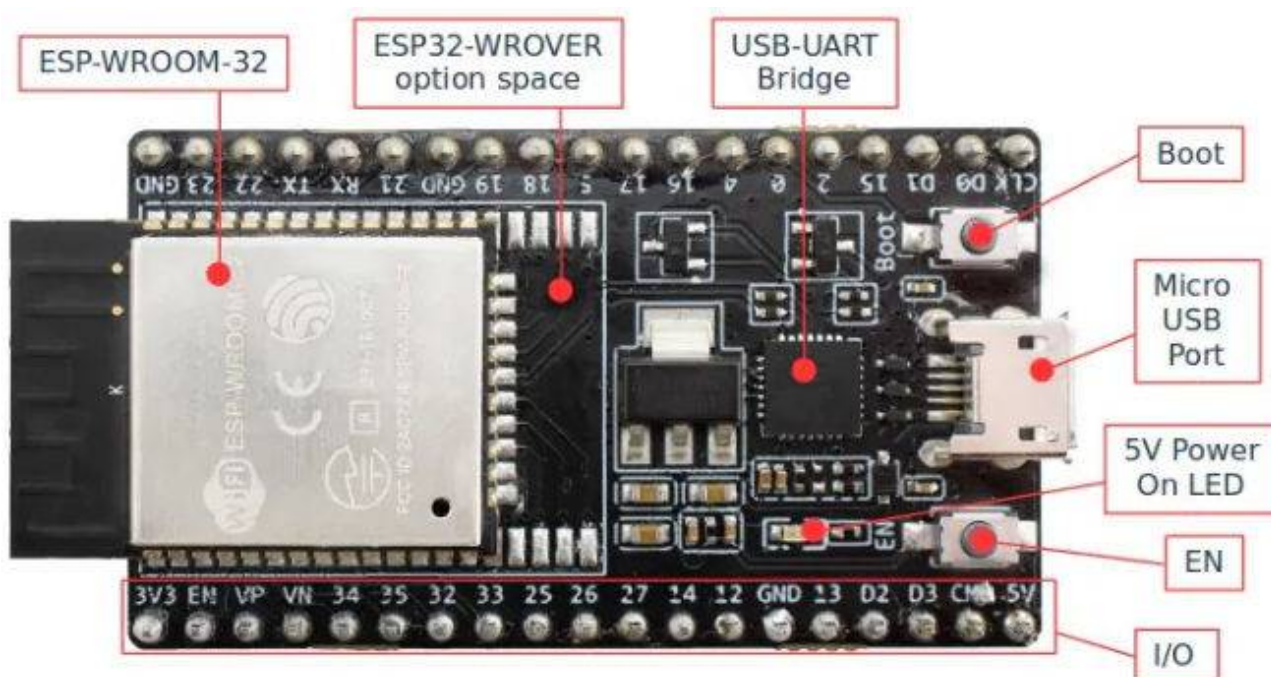
# ESP32 卡的發現

## 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

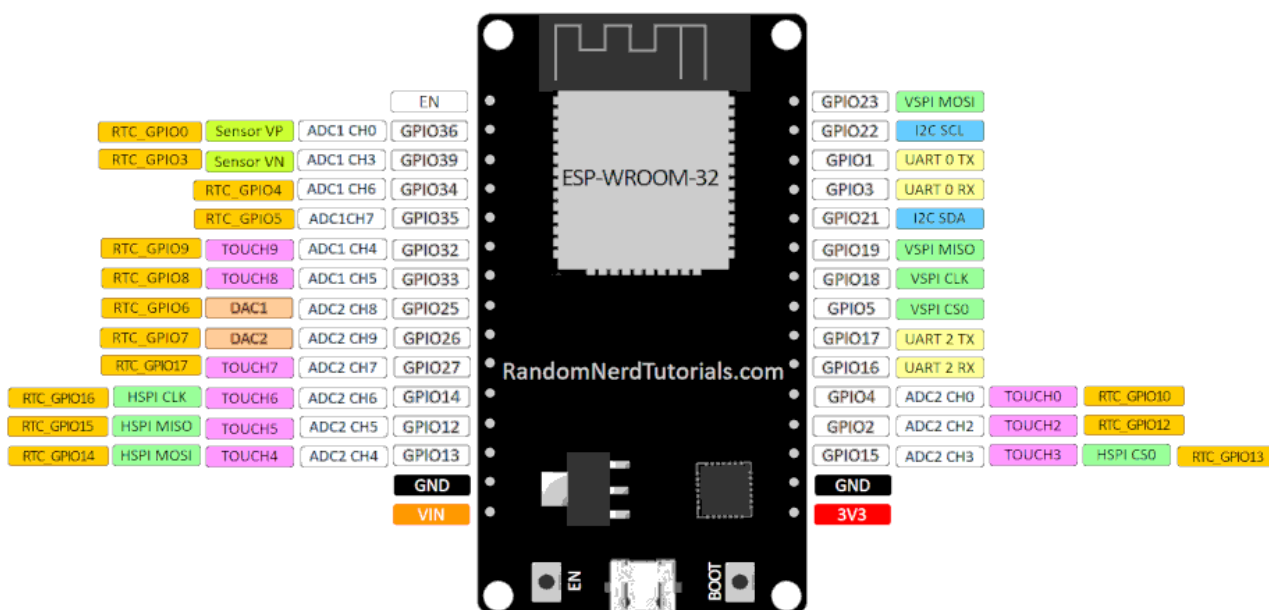
以下是 ESP32 卡的照片，我們將從中解釋不同 GPIO 輸入/輸出的作用：



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。

# 安裝 ESP32Forth

## 下載 ESP32forth

第一步包括恢復 ESP32forth 的 C 語言原始碼。最好使用最新版本：

<https://esp32forth.appspot.com/ESP32forth.html>

下載檔案的內容：

```
ESP32forth-7.0.xx
  ESP32forth
    自述文件.txt
    esp32forth.ino
    選修的
      SPI-flash.h
      串行藍牙.h
      ... ETC...
```

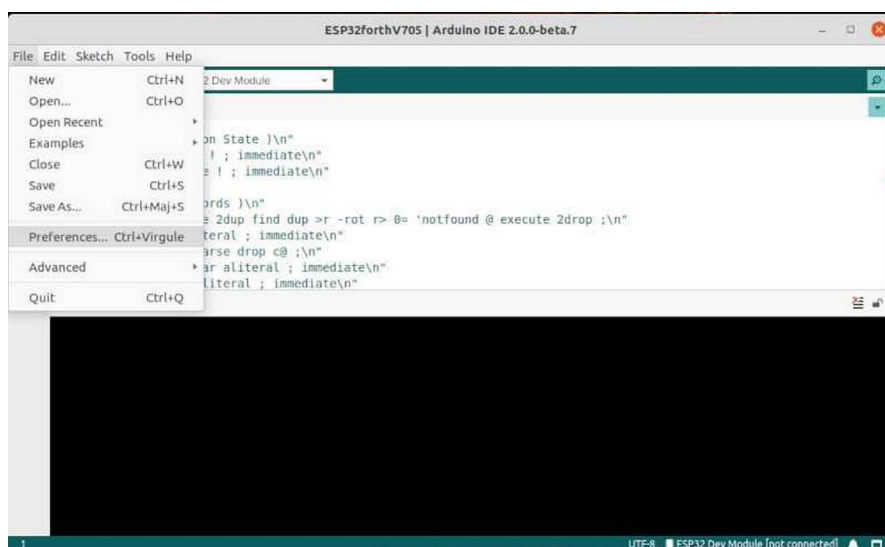
## 編譯並安裝 ESP32forth

**esp32forth.ino** 檔案複製到工作目錄中。可選目錄包含允許擴展 ESP32forth 的檔案。對於我們第一次建置和上傳 ESP32forth，不需要這些檔案。

要編譯 ESP32forth，您的電腦上必須已安裝 ARDUINO IDE：

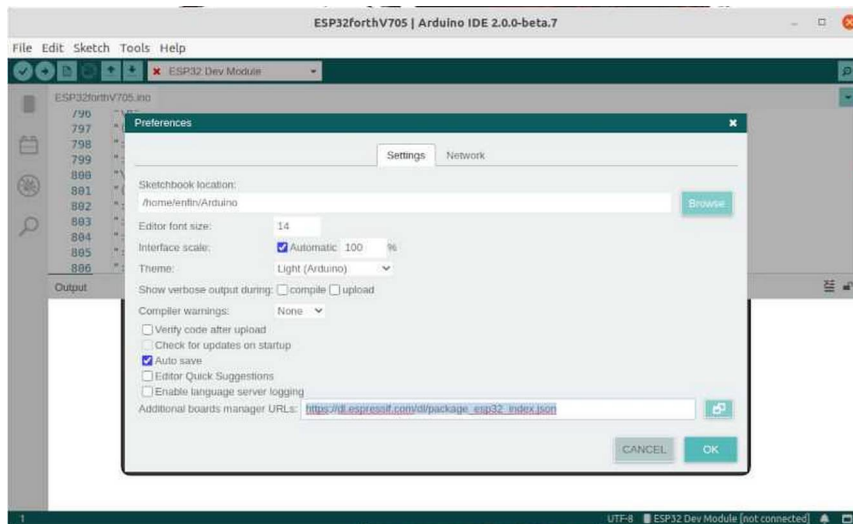
<https://docs.arduino.cc/software/ide-v2>

安裝 ARDUINO IDE 後，啟動它。ARDUINO IDE 已經開放，這裡是 2.0 版本。點擊檔案並選擇首選項：

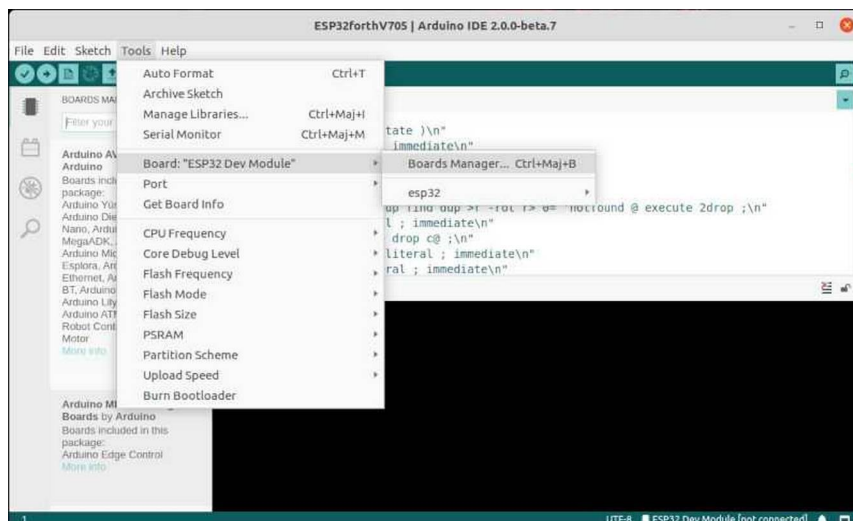


在出現的視窗中，轉到標記為「附加板管理器 URL」的輸入框，然後輸入以下行：

```
https://dl.espressif.com/dl/package\_esp32\_index.json
```



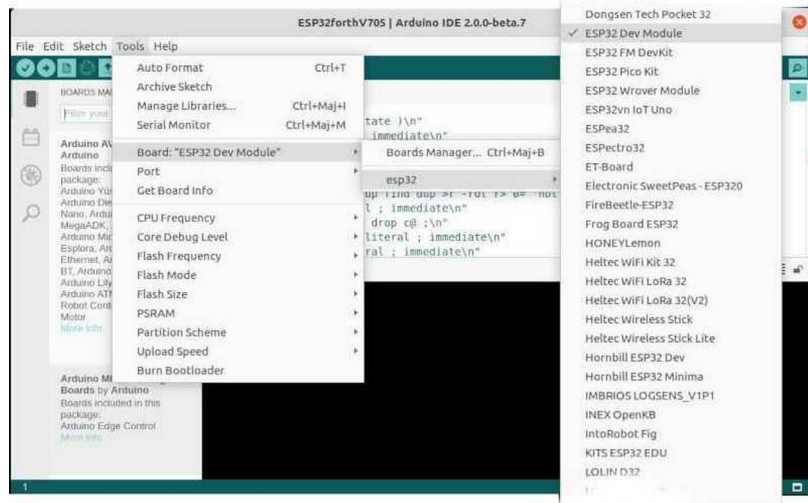
接下來，點擊“工具”並選擇“板：”。



此選擇將為您提供 ESP32 軟體包的安裝。接受此安裝。

然後您應該能夠存取 ESP32 卡的選擇：

**ESP32 開發模組板選型：**



## ESP32 WROOM 的設置

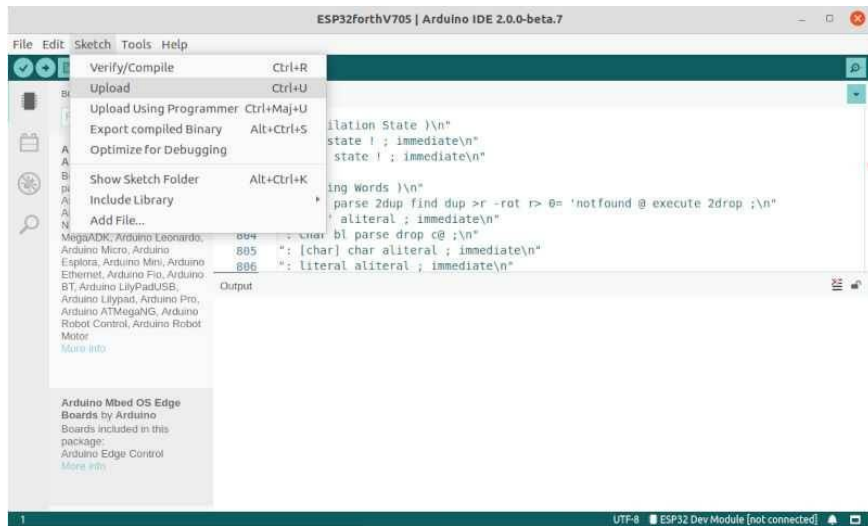
以下是編譯 ESP32forth 之前所需的其他設定。再次點選 “工具” 存取設定：

```
-- 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
```

## 開始編譯

剩下的就是編譯 ESP32forth。透過 *File* 和 *Open* 載入原始碼。

假設您的 ESP32 板已連接到 USB 連接埠。點選 *Sketch* 並選擇 *Upload* 開始編譯：



如果一切正常，您應該將二進位代碼自動傳輸到 ESP32 板中。如果編譯沒有錯誤，但出現傳輸錯誤，請重新編譯 **esp32forth.ino** 檔案。傳輸時，按下 ESP32 板上標示 **BOOT** 的按鈕。這將使該卡可用於傳輸 ESP32forth 二進位代碼。

影片中 ARDUINO IDE 的安裝與配置：

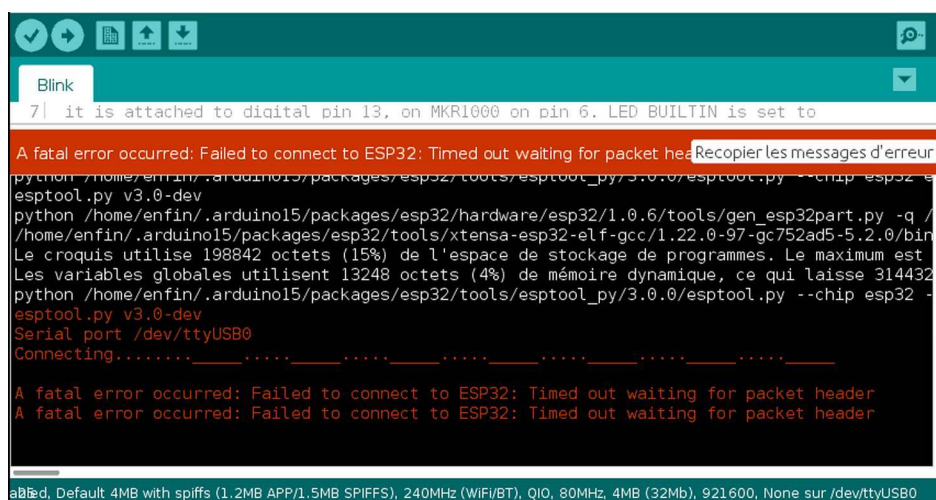
- Windows: <https://www.youtube.com/watch?v=2AZQfieHv9g>
- Linux: [https://www.youtube.com/watch?v=JeD3nz0\\_nc](https://www.youtube.com/watch?v=JeD3nz0_nc)

## • 修復上傳連線錯誤

了解如何修復在嘗試一次將新程式碼上傳到 ESP32 卡時發生的致命錯誤：「無法連接到 ESP32：等待資料包標頭逾時」。

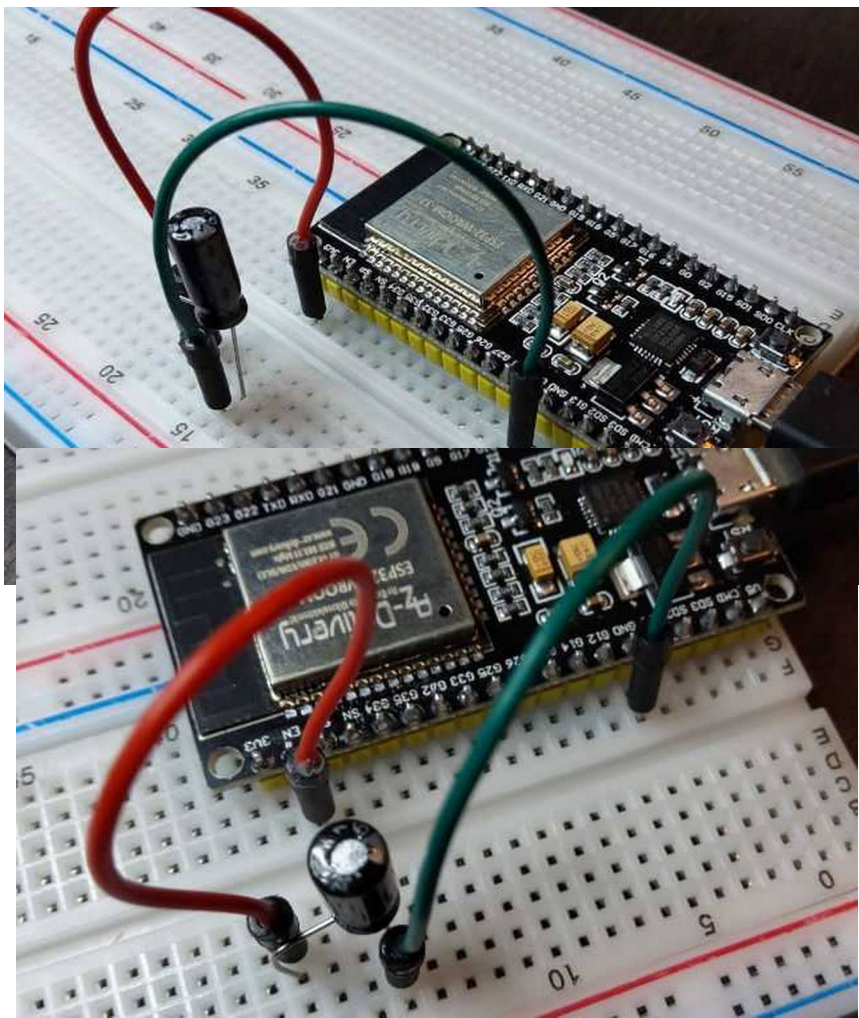
下載新程式碼時，某些 ESP32 開發板（請參閱最佳 ESP32 板）不會自動進入快閃記憶體/上傳模式。

這表示當您嘗試將新草圖上傳到 ESP32 開發板時，ARDUINO IDE 無法連線到您的開發板，並且您會收到以下錯誤訊息：



```
7| it is attached to digital pin 13, on MKR1000 on pin 6. LED BUILTIN is set to  
A fatal error occurred: Failed to connect to ESP32: Timed out waiting for packet header  
python /home/enfin/.arduino15/packages/esp32/tools/esptool_py/3.0.0/esptool.py --chip esp32  
esptool.py v3.0-dev  
python /home/enfin/.arduino15/packages/esp32/hardware/esp32/1.0.6/tools/gen_esp32part.py -q /  
/home/enfin/.arduino15/packages/esp32/tools/xtensa-esp32-elf-gcc/1.22.0-97-gc752ad5-5.2.0/bin  
Le croquis utilise 198842 octets (15%) de l'espace de stockage de programmes. Le maximum est  
Les variables globales utilisent 13248 octets (4%) de mémoire dynamique, ce qui laisse 314432  
python /home/enfin/.arduino15/packages/esp32/tools/esptool_py/3.0.0/esptool.py --chip esp32 -  
esptool.py v3.0-dev  
Serial port /dev/ttyUSB0  
Connecting.....  
A fatal error occurred: Failed to connect to ESP32: Timed out waiting for packet header  
A fatal error occurred: Failed to connect to ESP32: Timed out waiting for packet header  
added, Default 4MB with spiiffs (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (32Mb), 921600, None sur /dev/ttyUSB0
```

為了讓 ESP32 板自動切換到 flash/下載模式，我們可以在 EN 和 GND 接腳之間連接 10uF 電解電容：



只有當您處於從 **ARDUINO IDE** 上傳 **ESP32forth** 的階段時，才需要進行此操作。一旦 **ESP32forth** 安裝在 **ESP32** 板上，就不再需要使用該電容器。





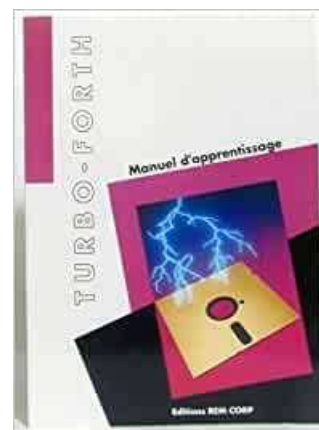
# 為什麼在 ESP32 上使用 FORTH 語言程式設計？

## 前言

我自 1983 年以來一直使用 FORTH 進程式設計。我於 1996 年停止使用 FORTH 進程式設計。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 Loisetech - 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。有些編譯器生成機器碼，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 寫的专业應用程式？

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。

# 使用 ESP32Forth 的真正 32 位元 FORTH

ESP32Forth 是真正的 32 位元 FORTH。這是什麼意思？

FORTH 語言支援整數值的運算。這些值可以是文字值、記憶體位址、暫存器內容等。

## 資料堆疊上的值

當 ESP32Forth 啟動時，FORTH 解釋器可用。如果輸入任何數字，它將作為 32 位元整數放入堆疊中：

```
35
```

如果我們堆疊另一個值，它也會被堆疊。前一個值將被下推一位：

```
45
```

為了添加這兩個值，我們使用一個詞，這裡 **+**：

```
+
```

我們將兩個 32 位元整數值相加，並將結果放入堆疊中。為了顯示這個結果，我們將使用單字 **. \**：

```
. \ 顯示 80
```

在 FORTH 語言中，我們可以將所有這些操作集中在一行中：

```
35 45 + . \ 顯示 80
```

與 C 語言不同，我們沒有定義 **int8** 或 **int16** 或 **int32** 型別。

對於 ESP32Forth，ASCII 字元將由 32 位元整數指定，但其值將受到 [32..256]。例：

```
67 emit \顯示 C
```

## 記憶體中的值

ESP32Forth 允許您定義常數和變數。它們的內容將始終採用 32 位元格式。但有些情況下這不一定適合我們。讓我們舉一個簡單的例子，定義摩斯電碼字母表。我們只需要幾個位元組：

- 1 定義莫爾斯電碼符號的數量
- 摩斯電碼的每個字母一個或多個位元組

```
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,
```

這裡我們只定義了 3 個字，**mA**、**mB** 和 **mC**。每個字中儲存幾個位元組。問題是：我們如何檢索這些單字中的信息？

這些字之一的執行會存入一個 32 位元值，對應到我們儲存摩斯電碼資訊的記憶體位址。我們將使用單字 **c@** 從每個字母中提取莫爾斯電碼：

```
mA c@ . \ 顯示 2
mB c@ . \ 顯示 4
```

像這樣提取的第一個位元組將用於管理循環以顯示字母的莫爾斯電碼：

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

當然還有很多更優雅的例子。在這裡，它展示了一種操作 8 位元值（我們的位元組）的方法，同時我們在 32 位元堆疊上使用這些位元組。

## 根據資料大小或類型進行文字處理

在所有其他語言中，我們都有一個通用詞，例如 **echo**（在 **PHP** 中），它可以顯示任何類型的資料。無論是整數、實數、字串，我們總是使用同一個字。**PHP** 語言範例：

```
$bread = "backed bread";
$price = 2.30;
echo $bread . " : " . $price;
// display   backed bread: 2.30
```

對於所有程式設計師來說，這種做事方式就是標準！那麼 **FORTH** 將如何在 **PHP** 中完成這個範例呢？

```
: bread s" backed bread" ;
: price s" 2.30" ;
bread type    s" : " type    price type
\ display    backed bread: 2.30
```

這裡，單字 **type** 告訴我們我們剛剛處理了一個字串。

**PHP**（或任何其他語言）具有通用函數和解析器，而 **FORTH** 使用單一資料類型進行補償，但採用了適應的處理方法，這些方法告訴我們所處理資料的性質。

這是 **FORTH** 的一個絕對簡單的例子，以 **HH:MM:SS** 格式顯示秒數：

```

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

```

我喜歡這個例子，因為到目前為止，**沒有其他程式語言**能夠如此優雅而簡潔地執行這種 HH:MM:SS 轉換。

您已經明白，FORTH 的秘密就在於它的詞彙。

## 結論

FORTH 沒有資料型別。所有資料都透過資料堆疊。堆疊中的每個位置始終是 32 位元整數！

這就是所有需要知道的。

超結構化和冗長語言（例如 C 或 Java）的純粹主義者肯定會大喊異端。在這裡，我將允許自己回答這些問題：為什麼需要輸入資料？

因為 FORTH 的強大之處就在於這種簡單性：具有非類型化格式和非常簡單的操作的單一資料堆疊。

我將向您展示許多其他程式語言無法做到的事情，定義新的定義詞：

```

: 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   .- -... -.-.

```

在這裡，莫爾斯電碼這個詞：已經成為一個定義詞，就像**常數**或**變數**一樣...

因為 FORTH 不只是一種程式語言。它是一種元語言，也就是說一種建構你自己的程式語言的語言...

# 字典/堆疊/變數/常數

## 展開字典

Forth 屬於編織解釋語言類別。這意味著它可以解釋在控制台上鍵入的命令，以及編譯新的子程式和程式。

Forth 編譯器是語言的一部分，特殊單字用於建立新的字典條目（即單字）。最重要的是：**:**（開始一個新定義）和**;**（完成定義）。讓我們輸入以下命令來嘗試：

```
: *+ * + ;
```

發生了什麼事？的動作是建立一個名為**\*+**的**新字典條目**，並從解釋模式切換到編譯模式。在編譯模式下，解釋器會搜尋單字，而不是執行它們，而是安裝指向其程式碼的指標。如果文字是數字，則 ESP32forth 不會將其壓入堆疊，而是在為新單字分配的字典空間中建構數字，並遵循每次執行該單字時將儲存的數字放入堆疊的特殊程式碼。因此，**\*+** 的執行動作是順序執行先前定義的字**\***和**+**。

這個詞 **:** 很特別。它是一個立即字，即使系統處於編譯模式，它也總是被執行。什麼是**;**是雙重的。首先，它安裝將控制項返回到解釋器的下一個外部層級的程式碼，其次，它從編譯模式返回到解釋模式。現在試試你的新字：

```
decimal 5 6 7 *+ . \ 顯示 47
```

此範例說明了 Forth 中的兩個主要工作活動：將新單字新增到字典中，並在定義後立即嘗試它。

## 字典管理

單字 **forget** 後面跟著「要刪除」的單字將刪除自該單字以來您所做的所有字典條目：

```
: test1 ;  
: test2 ;  
: test3 ;  
忘記 test2 \ 從字典中刪除 test2 和 test3
```

## 堆疊和逆波蘭表示法

Forth 有一個明確可見的堆疊，用於在單字（命令）之間傳遞數字。使用 Forth 可以有效地迫使您從堆疊的角度進行思考。一開始這可能會很困難，但就像任何事情一樣，透過練習它會變得容易得多。

在 FORTH 中，這堆類似於一堆寫有數字的卡片。數字總是添加到堆疊頂部並從堆疊頂部刪除。

ESP32forth 整合了兩個堆疊：參數堆疊和回饋堆疊，每個堆疊由多個可容納 16 位元數字的單元組成。

第四條輸入線：

```
decimal 2 5 73 -16
```

保留參數堆疊不變

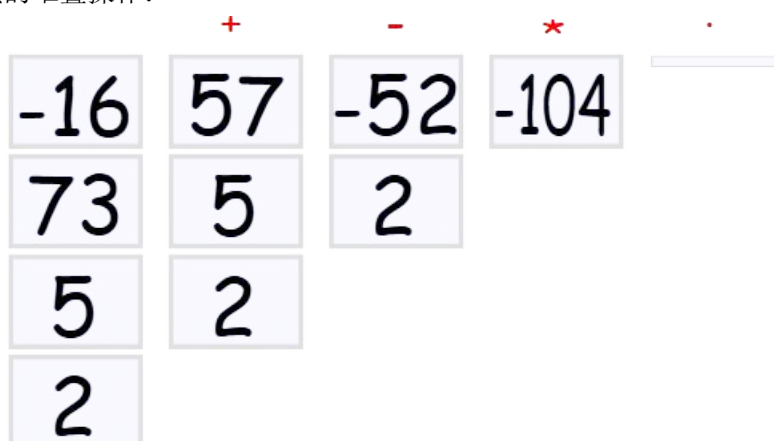
	細胞	內容	評論
0		-16	(TOS) 右上角
1		73	(NOS) 下一個
2		5	
3		2	

我們通常會在 **Forth** 資料結構（例如堆疊、陣列和表格）中使用從零開始的相對編號。請注意，當像這樣輸入數字序列時，最右邊的數字將成為 **TOS**，最左邊的數字位於堆疊的底部。

假設我們使用以下行跟隨原始輸入行

```
+ - * .
```

這些操作將產生連續的堆疊操作：



兩行之後，控制台顯示：

```
decimal 2 5 73 -16 \ 顯示: 2 5 73 -16 ok
+ - * . \ 顯示: -104 ok
```

請注意，**ESP32forth** 在解釋每一行時會方便地顯示堆疊元素，並且 **-16** 的值顯示為 **32** 位元無符號整數。此外，這個字。消耗資料值**-104**，使堆疊為空。如果我們執行。在現在為空的堆疊上，外部解釋器將中止並出現堆疊指標錯誤 **STACK UNDERFLOW ERROR**。

先出現運算元，後跟運算子的程式表示法稱為逆波蘭表示法 (**RPN**)。

## 處理參數堆疊

作為一個基於堆疊的系統，**ESP32forth** 必須提供將數字放入堆疊、刪除它們以及重新排列它們的順序的方法。我們已經看到，只需鍵入數字即可將數字放入堆疊中。我們也可以將數字整合到 **FORTH** 單字的定義中。

**drop** 這個字從堆疊頂部刪除一個數字，從而將下一個數字放在頂部。單字**交換**交換前 **2** 個數字。 **dup** 複製頂部的數字，將所有其他數字向下推。 **rot** 旋轉前 **3** 個數字。這些行動如下所示。



## 返回堆疊及其用途

編譯新單字時，ESP32forth 會在調用單字和之前定義的單字之間建立鏈接，這些單字將由新單字的執行呼叫。這種連結機制在運行時使用 **rstack**。要呼叫的下一個字的位址被放置在返回堆疊中，以便噹噹前字執行完畢時，系統知道從哪裡移動到下一個字。由於單字可以嵌套，因此必須有一個這些返回位址的堆疊。

除了充當傳回位址的儲存庫之外，使用者還可以儲存和檢索返回堆疊，但必須小心操作，因為返回堆疊對於程式執行至關重要。如果您使用回電電池進行臨時存儲，則必須將其恢復到原始狀態，否則您可能會導致 ESP32forth 系統崩潰。儘管存在危險，但有時使用 **backstack** 作為臨時儲存可以降低程式碼的複雜性。

要儲存在堆疊上，請使用 **>r** 將參數堆疊的頂部移動到返回堆疊的頂部。要檢索值，**r>** 將堆疊頂部的值移回參數堆疊的頂部。為了簡單地從堆疊頂部刪除一個值，可以使用 **rdrop** 一詞。單字 **r@** 將堆疊頂部複製回參數堆疊。

## 記憶體使用情況

**@ (fetch)** 從記憶體中取出到堆疊，並透過字 **!** 從頂部儲存到記憶體。（瞎的）。**@** 期望堆疊上的地址並用其內容替換該地址。**!** 需要一個數字和一個地址來儲存它。它將數字放置在位址引用的記憶體位置中，同時消耗該過程中的兩個參數。

表示 8 位元（位元組）值的無符號數字可以放置在字元大小的字元中。使用 **c@** 和 **c!** 的儲存單元。

```
create testVar
  cell allot
$f7 testVar c!
testVar c@ . \ 顯示 247
```

## 變數

變數是記憶體中的命名位置，可以將數字（例如計算的中間結果）儲存在堆疊之外。例如：

```
variable x
```

建立一個名為 **x** 的儲存位置，執行時將其儲存位置的位址保留在堆疊頂部：

```
x . \ 顯示位址
```

然後我們可以在這個地址收集或儲存：

```
variable x
3 x !
x @ . \ 顯示: 3
```

## 常數

常數是程式運行時您不希望更改的數字。執行與常數關聯的字的結果是保留在堆疊上的資料的值。

```
\ 定義 VSPI 腳
19 constant VSPI_MISO
```

```

23 constant VSPI_MOSI
18 constant VSPI_SCLK
05 constant VSPI_CS

\ 設定 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
: init.VSPI ( -- )
    VSPI_CS OUTPUT pinMode
    VSPI_SCLK VSPI_MISO VSPI_MOSI VSPI_CS SPI.begin
    SPI_FREQ SPI.setFrequency
;

```

## 偽常數值

用 **value** 定義的值是變數和常數的混合型別。我們設定並初始化一個值，然後像呼叫常數一樣呼叫它。我們也可以像更改變數一樣更改值。

```

decimal
13 value thirteen
thirteen . \顯示: 13
47 to thirteen
thirteen . \顯示: 47

```

單字 **to** 也適用於單字定義，將其後面的值替換為當前堆疊頂部的值。您需要注意的是，**to** 後面是由 **value** 定義的值，而不是其他內容。

## 記憶體分配的基本工具

**創建**和**分配**這兩個詞是保留記憶體空間並為其附加標籤的基本工具。例如，以下轉錄顯示了一個新的**圖形數組**字典條目：

```

create graphic-array ( --- addr )
    %00000000 c,
    %00000010 c,
    %00000100 c,
    %00001000 c,
    %00010000 c,
    %00100000 c,
    %01000000 c,
    %10000000 c,

```

執行時，**圖形數組**字將推送第一個條目的位址。

使用前面解釋的獲取和儲存字來存取分配給**圖形陣列**的記憶體。要計算分配給**圖形數組**的第三個位元組的位址，我們可以編寫**圖形數組** **graphic-array 2 +**，記住索引從 **0** 開始。

```

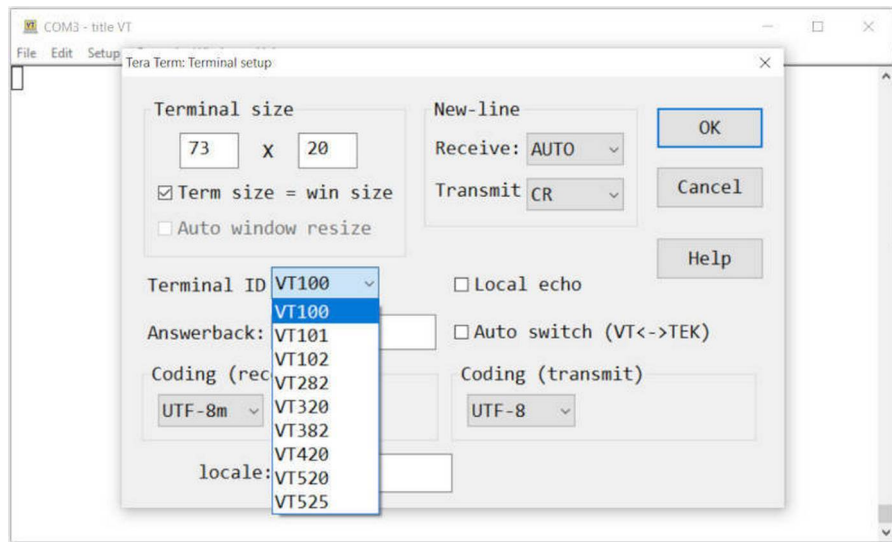
30 graphic-array 2 + c!
graphic-array 2 + c@ . \ 顯示 30

```

# 終端機上的文字顏色和顯示位置

## 端子的 ANSI 編碼

如果您使用終端軟體與 ESP32forth 進行通信，則該終端很可能模擬 VT 類型終端或同等終端。此處，TeraTerm 配置為模擬 VT100 終端：



這些終端有兩個有趣的功能：

- 為頁面背景和要顯示的文字著色
- 定位顯示遊標

這兩個功能均由 ESC（轉義）序列控制。這就是 ESP32forth 中 **bg** 和 **fg** 的定義方式：

```
forth definitions ansi
: fg ( n -- ) esc ." [38;5;" n. ." m" ;
: bg ( n -- ) esc ." [48;5;" n. ." m" ;
: normal esc ." [0m" ;
: at-xy ( x y -- ) esc ." [" 1+ n. ." ;" 1+ n. ." H" ;
: page esc ." [2J" esc ." [H" ;
```

單字 **normal**「正常」會覆蓋 **bg** 和 **fg** 定義的著色序列。

單字 **page** 頁面清除終端螢幕並將遊標定位在螢幕的左上角。

## 文字著色

讓我們先看看如何為文字著色：

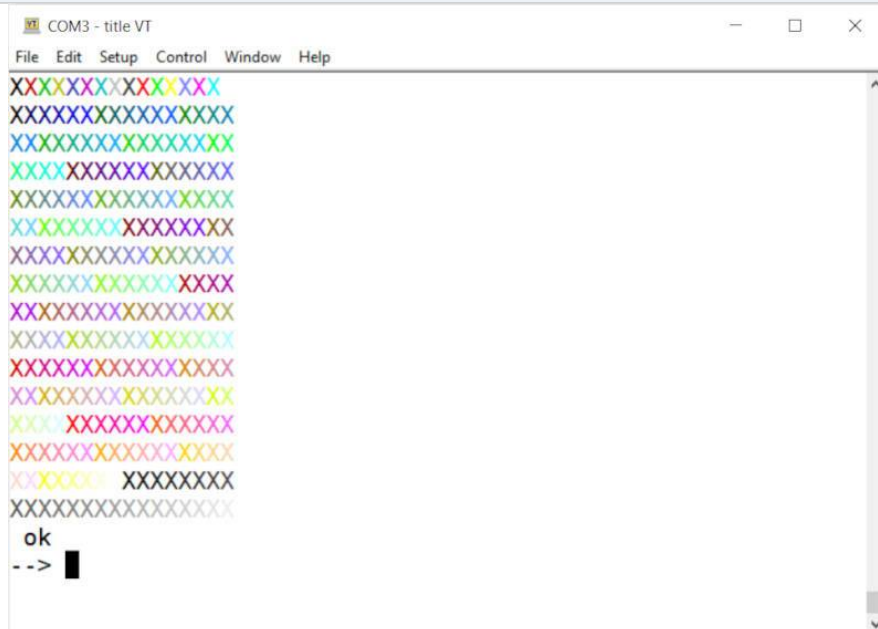
```
: testFG ( -- )
  page
```



```

16 0 do
  16 0 do
    j 16 * i + fg
    ." X"
  loop
cr
loop
normal
;

```



運行 **testFG** 會顯示以下內容：

要測試背景顏色，我們將按以下步驟進行：

```

: testBG ( -- )
  page
  16 0 do
    16 0 do
      j 16 * i + bg
      space space
    loop
  cr
  loop
normal
;

```

執行 **testBG** 會顯示以下內容：



## 顯示位置

該終端是與 ESP32forth 通訊的最簡單的解決方案。使用 ANSI 轉義序列可以輕鬆改進資料的表示。

```
09 constant red
11 constant yellow
14 constant cyan
15 constant whyte
: box { x0 y0 xn yn color -- }
  color bg
  yn y0 - 1+ \ 決定高度
  0 do
    x0 y0 i + at-xy
    xn x0 - spaces
  loop
  normal
;

: 3boxes ( -- )
  page
  2 4 20 6 cyan box
  8 6 28 8 red box
  14 8 36 10 yellow box
  0 0 at-xy
;
```

運行 **3boxes** 顯示如下：



現在您可以建立簡單有效的介面，允許與 **ESP32forth** 編譯的 **FORTH** 定義進行互動。

# ESP32Forth 的局部變數

## 介紹

FORTH 語言主要透過資料棧來處理資料。這種非常簡單的機制提供了無與倫比的性能。相反，追蹤資料流很快就會變得複雜。局部變數提供了一個有趣的選擇。

## 假堆疊註釋

如果您遵循第四個不同的範例，您將注意到由 ( 和 ) 構成的堆疊註解。例子：

```
\ 將兩個無符號值相加，將和和進位留在堆疊上
: um+ ( u1 u2 -- sum carry )
  \ here the definition
;
```

在這裡，註釋 ( **u1 u2 -- sum carry** ) 對 FORTH 程式碼的其餘部分絕對沒有任何作用。這是純粹的評論。

當準備複雜的定義時，解決方案是使用由 { 和 } 構成的局部變數。例：

```
: 2OVER { a b c d }
  a b c d a b
;
```

我們定義了四個局部變數 **a b c** 和 **d**。

單字 { 和 } 與單字 ( 和 ) 類似，但效果完全不同。位於 { 和 } 之間的程式碼是局部變數。唯一的限制：不要使用可能是 FORTH 字典中的 FORTH 單字的變數名稱。我們不妨這樣寫我們的例子：

```
: 2OVER { varA varB varC varD }
  varA varB varC varD varA varB
;
```

每個變數都會按照其在資料堆疊上的存放順序取得資料堆疊的值。這裡，**1** 進入 **varA**，**2** 進入 **varB**，以此類推：

```
--> 1 2 3 4
ok
1 2 3 4 --> 2over
ok
1 2 3 4 1 2 -->
```

我們的假堆疊註解可以這樣完成：

```
: 2OVER { varA varB varC varD -- varA varB varC varD varA varB }
```

-- 以下字元無效。唯一的一點是讓我們的假評論看起來像真正的堆疊評論。

## 對局部變數的操作

**value** 定義的偽變數完全相同。例：

```
: 3x+1 { var -- sum }
      var 3 * 1 +
      ;
```

的效果：

```
0 value var
: 3x+1 ( var -- sum )
  to var
  var 3 * 1 +
  ;
```

在此範例中，**var** 由 **value** 明確定義。

**to** 或 **+to** 為局部變數賦值，以增加局部變數的內容。在此範例中，我們在單字程式碼中加入一個初始化為零的局部變數 **result**：

```
: a+bEXP2 { varA varB -- (a+b)EXP2 }
  0 { result }
  varA varA *      to result
  varB varB *      +to result
  varA varB * 2 * +to result
  result
  ;
```

不是比這個更具可讀性嗎？

```
: a+bEXP2 ( varA varB -- result )
  2dup
  * 2 * >r
  dup *
  swap dup * +
  r> +
  ;
```

**um+** 的定義，它將兩個無符號整數相加，並將總和以及該總和的溢出值保留在資料堆疊上：

```
\ 將兩個無符號整數相加，將和和進位留在堆疊上
: um+ { u1 u2 -- 進位與 }
: um+ { u1 u2 -- sum carry }
  0 { sum }
  cell for
    aft
      u1 $100 /mod to u1
      u2 $100 /mod to u2
      +
      cell 1- i - 8 * lshift +to sum
    then
  next
  sum
  u1 u2 + abs
  ;
```

以下是一個更複雜的例子，使用局部變數重寫 **DUMP**：

```

\ 轉儲中的局部變數:
\ START_ADDR    \ 轉儲的首位址
\ END_ADDR      \ 轉儲的最後位址
\ OSTART_ADDR   \ 轉儲中循環的第一個位址
\ LINES         \ 轉儲循環的行數
\ myBASE        \ 目前數值基數
: dump ( start len -- )
  cr cr ." --addr--- "
  ." 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  -----chars-----"
  2dup + { END_ADDR }          \ store latest address to dump
  swap { START_ADDR }          \ store START address to dump
  START_ADDR 16 / 16 * { OSTART_ADDR } \ calc. addr for loop start
  16 / 1+ { LINES }
  base @ { myBASE }            \ save current base
  hex
  \ outer loop
  LINES 0 do
    OSTART_ADDR i 16 * +        \ calc start address for current line
    cr <# # # # # [char] - hold # # # # #> type
    space space                \ and display address
    \ first inner loop, display bytes
    16 0 do
      \ calculate real address
      OSTART_ADDR j 16 * i + +
      ca@ <# # # #> type space \ display byte in format: NN
    loop
    space
    \ second inner loop, display chars
    16 0 do
      \ calculate real address
      OSTART_ADDR j 16 * i + +
      \ display char if code in interval 32-127
      ca@      dup 32 < over 127 > or
      if      drop [char] . emit
      else    emit
      then
    loop
    loop
    myBASE base !              \ restore current base
    cr cr
  ;
forth

```

局部變數的使用極大地簡化了堆疊上的資料操作。程式碼更具可讀性。請注意，不需要預先聲明這些局部變量，在使用它們時指定它們就足夠了，例如：**base @ { myBASE }**。

**>r** 和 **r>** 一詞，否則您將面臨破壞局部變數管理的風險。只要看一下這個版本的 **DUMP** 的反編譯就可以明白這個警告的原因：

```

: dump cr cr s" --addr--- " type
  s" 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  -----chars-----" type
  2dup + >R SWAP >R -4 local@ 16 / 16 * >R 16 / 1+ >R base @ >R
  hex -8 local@ 0 (do) -20 local@ R@ 16 * + cr
  <# # # # # 45 hold # # # # #> type space space
  16 0 (do) -28 local@ j 16 * R@ + + CA@ <# # # #> type space 1 (+loop)
  0BRANCH rdrop rdrop space 16 0 (do) -28 local@ j 16 * R@ + + CA@ DUP 32 < OVER 127 > OR
  0BRANCH DROP 46 emit BRANCH emit 1 (+loop) 0BRANCH rdrop rdrop 1 (+loop)
  0BRANCH rdrop rdrop -4 local@ base ! cr cr rdrop rdrop rdrop rdrop rdrop ;

```

## 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  $r1 \leq r2$ ，則 fl 為真。
- **f<>** (r1 r2 -- fl) 如果  $r1 \neq r2$ ，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  $r1 \geq r2$ ，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

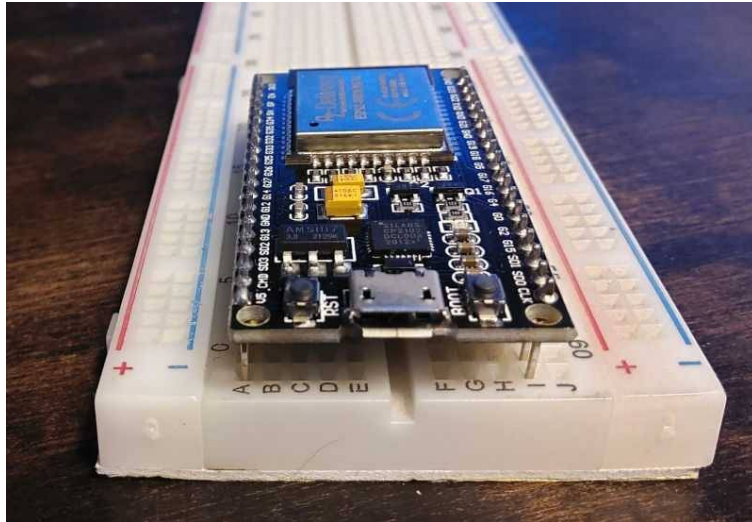
3.5e F>S .    \ display 3
```



# 將麵包板適應 ESP32 板

## ESP32 測試板

您剛剛收到 ESP32 卡。第一個糟糕的驚喜是，這張卡在測試板上非常不適合：

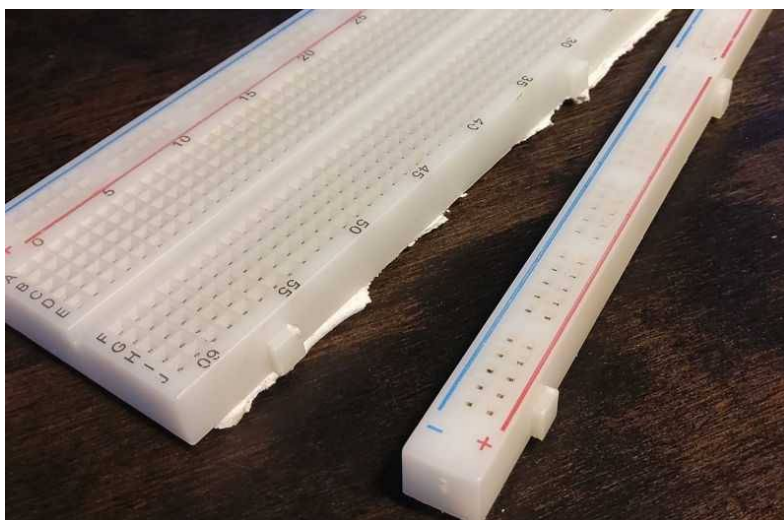


沒有專門適合 ESP32 板的麵包板。

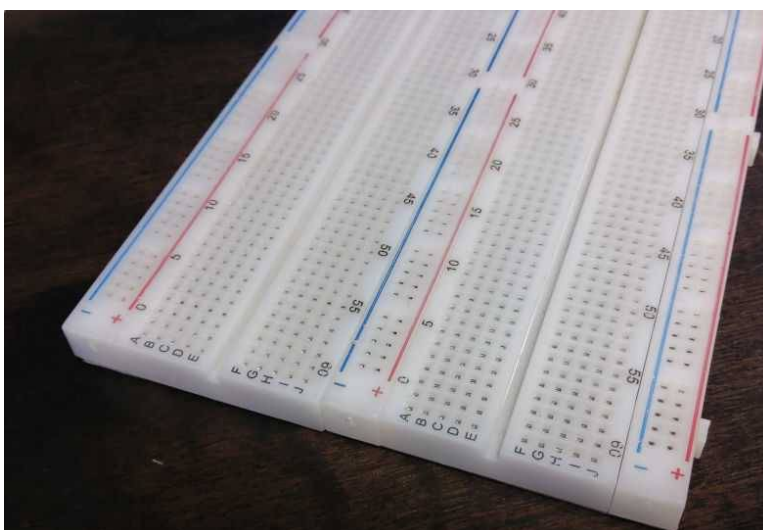
## 建造適合 ESP32 板的麵包板

我們將建立自己的測試板。為此，您必須有兩個相同的測試板。

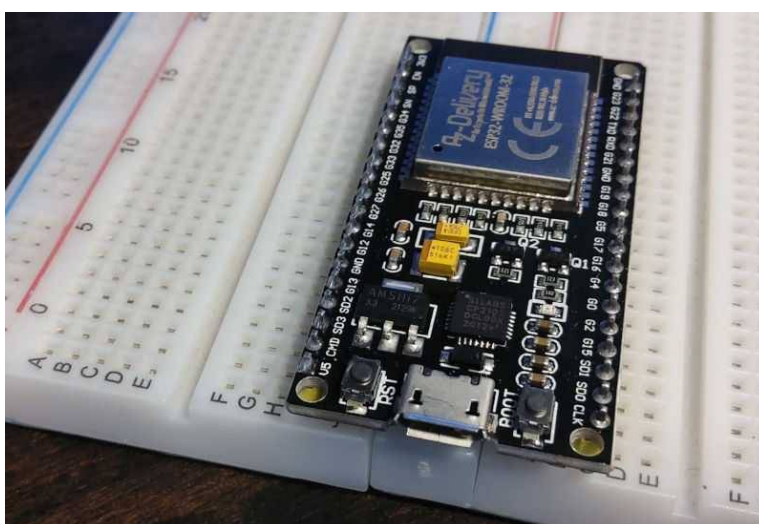
在其中一個板上，我們將拆下一條電源線。為此，請使用刀具從下方切割。您應該能夠像這樣分開這條電源線：



然後我們可以用這張地圖重新組裝整個地圖。測試板的側面有橡子將它們連接在一起：



就這樣吧！現在我們可以放置 ESP32 卡：



I/O 埠可以毫無困難地擴展。



## 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/MPETREMANNN11/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

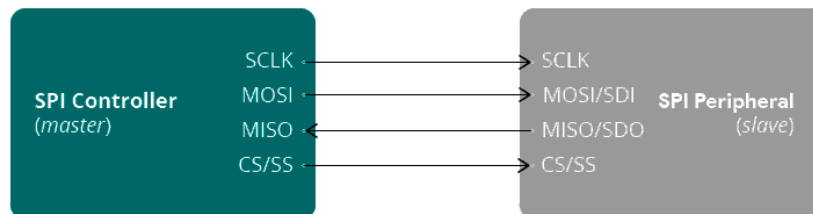
**spi** 詞彙表中定義的單字：

```
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.writel6
SPI.write32 SPI.writeBytes SPI.writePixels SPI.writePattern spi-builtins
```

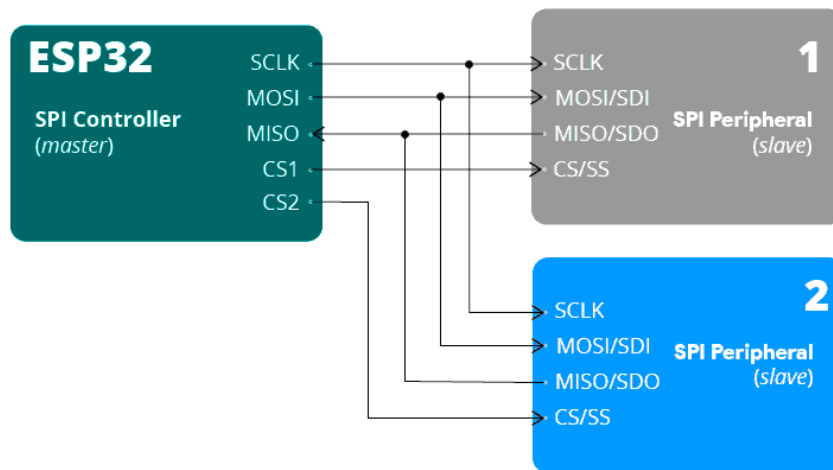
現在您可以透過 SPI 連接埠驅動擴展，例如 MAX7219 LED 顯示器。

## 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。



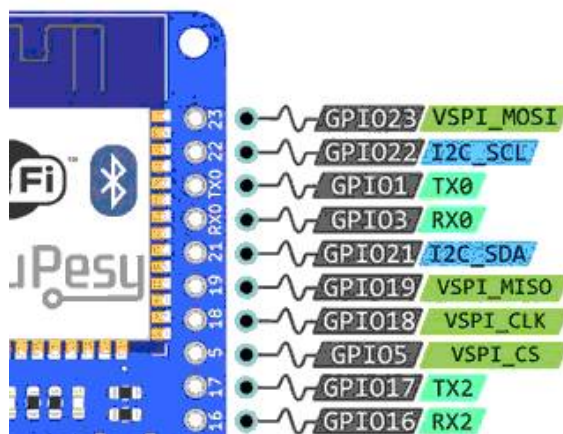
你可以有幾個奴隸。從設備可以是感測器、顯示器、microSD 卡等，或其他微控制器。這意味著您可以將 ESP32 連接到多個裝置。



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：



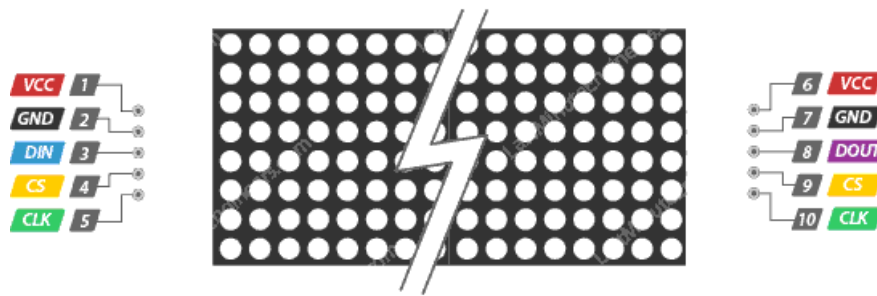
因此，使用 ESP32forth，我們可以定義指向這些 VSPI 引腳的常數：

```
\ 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

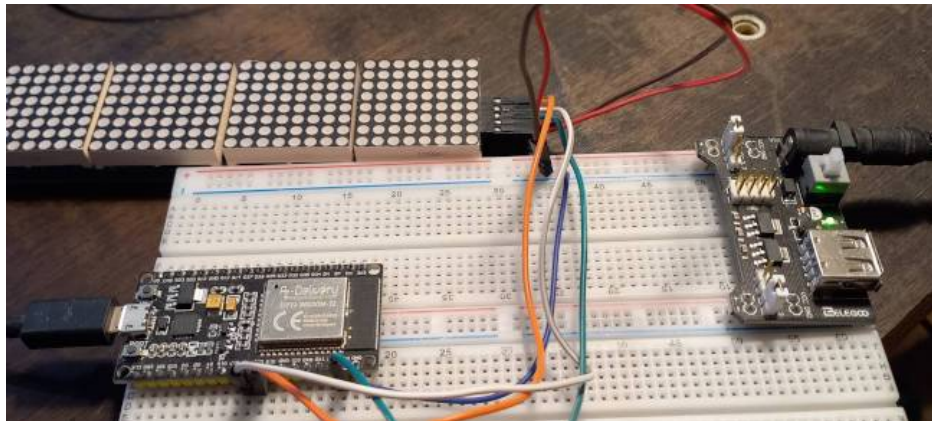
以下是 MAX7219 模組上的 SPI 連接埠連接器圖：



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。



# 安裝 HTTP 用戶端

## 編輯 ESP32forth.ino 文件

ESP32Forth 以原始檔案形式提供，由 C 語言編寫，該檔案必須使用 ARDUINO IDE 或任何其他與 ARDUINO 開發環境相容的 C 編譯器進行編譯。

以下是要修改的程式碼部分。第一部分修改：

```
#define ENABLE_SD_SUPPORT
#define ENABLE_SPI_FLASH_SUPPORT
#define ENABLE_HTTP_SUPPORT
// #define ENABLE_HTTPS_SUPPORT
```

第二部分修改：

```
// .....
#define VOCABULARY_LIST \
  V(forth) V(internals) \
  V(rtos) V(SPIFFS) V(serial) V(SD) V(SD_MMC) V(ESP) \
  V(ledc) V(http) V(Wire) V(WiFi) V(bluetooth) V(sockets) V(oled) \
  V(rmt) V(interrupts) V(spi_flash) V(camera) V(timers)
```

第三部分修改：

```
OPTIONAL_RMT_SUPPORT \
OPTIONAL_OLED_SUPPORT \
OPTIONAL_SPI_FLASH_SUPPORT \
OPTIONAL_HTTP_SUPPORT \
FLOATING_POINT_LIST

#ifndef ENABLE_HTTP_SUPPORT
# define OPTIONAL_HTTP_SUPPORT
#else

# include <HTTPClient.h>
  HTTPClient http;

# define OPTIONAL_HTTP_SUPPORT \
  XV(http, "HTTP.begin", HTTP_BEGIN, tos = http.begin(c0)) \
  XV(http, "HTTP.doGet", HTTP_DOGET, PUSH http.GET()) \
  XV(http, "HTTP.getPayload", HTTP_GETPL, String s = http.getString(); \
    memcpy((void *) n1, (void *) s.c_str(), n0); DROPn(2)) \
  XV(http, "HTTP.end", HTTP_END, http.end())
#endif
```

第四部分修改：

```
vocabulary ledc ledc definitions
transfer ledc-builtins
forth definitions
```

```

vocabulary http  http definitions
transfer http-builtins
forth definitions

vocabulary Serial  Serial definitions
transfer Serial-builtins
forth definitions

```

**ESP32forth.ino** 檔案後，您可以對其進行編譯並將其上傳到 ESP32 開發板。如果一切順利，你應該會有一個新的 **http** 詞彙：

```

http
vlist      \ displays :
HTTP.begin HTTP.doGet HTTP.getPayload HTTP.end http-builtins

```

## HTTP 用戶端測試

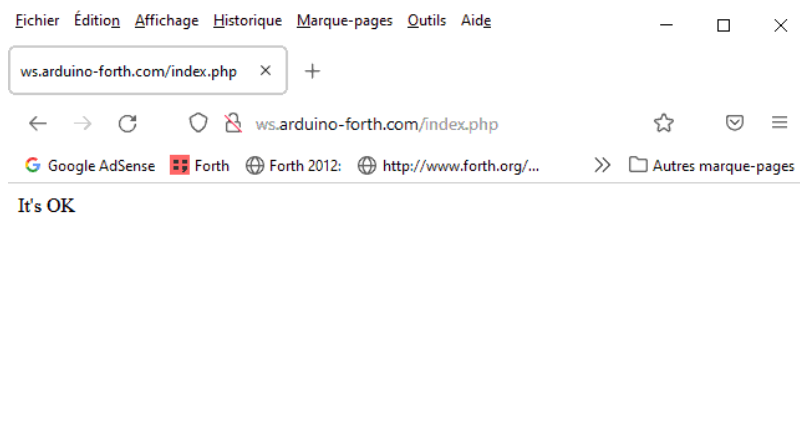
為了測試我們的 HTTP 用戶端，我們可以透過查詢任何 Web 伺服器來完成。但對於我們稍後要考慮的，您需要有一個個人網頁伺服器。在此伺服器上，我們建立一個子網域：

- 我們的伺服器是 [arduino-forth.com](http://arduino-forth.com)
- **ws** 子網域
- 我們使用 URL <http://ws.arduino-forth.com> 來訪問該子網域

正在建立此子網域，它不包含任何要執行的腳本。我們創建 **index.php** 頁面並將以下程式碼放在那裡：

```
It's OK
```

要檢查我們的子網域是否正常運行，只需從我們最喜歡的網頁瀏覽器中查詢它：



在我們最喜歡的網頁瀏覽器中顯示「**It's OK**」文字。現在讓我們看看如何從 ESP32Forth 執行相同的伺服器查詢...

以下是快速編寫的用於執行 HTTP 客戶端測試的 FORTH 程式碼：

```
WiFi
```

```

\ connection to local WiFi LAN
: myWiFiConnect
  z" mySSID"
  z" myWiFiCode"
  login
;

Forth

create httpBuffer 700 allot
  httpBuffer 700 erase

HTTP

: run
  cr
  z" http://ws.arduino-forth.com/" HTTP.begin
  if
    HTTP.doGet dup ." Get results: " . cr 0 >
    if
      httpBuffer 700 HTTP.getPayload
      httpBuffer z>s dup . cr type
    then
  then
  HTTP.end
;

```

**myWiFiConnect** 來啟動 Wifi 連接，然後運行：

```

--> myWiFiConnect
192.168.1.23
MDNS started
ok
--> run

Get results: 200
8
It's OK
ok

```

我們的 HTTP 用戶端完美地查詢了 Web 伺服器，顯示了與從 Web 瀏覽器檢索到的文字相同的文字。

這個小小的成功測試為巨大的可能性開闢了道路。





# ESP32forth 詞彙詳細內容

ESP32forth 提供了大量詞彙：

- **FORTH** 是主要詞彙；
- 某些詞彙用於 ESP32Forth 的內部機制，例如 **insides** 、 **asm...**
- 許多詞彙允許管理特定連接埠或配件，例如 **藍牙**、 **oled** 、 **spi** 、 **wifi** 、 **wire...**

在這裡您將找到這些不同詞彙表中定義的所有單字的清單。有些單字帶有彩色連結：

**align** 是一個普通的 FORTH 字；

**CONSTANT** 是定義詞；

**begin** 標記控制結構；

**key** 是延遲執行字；

**LED** 常數、變數或值定義的詞；

**registers** 標記一個詞彙。

**FORTH** 詞彙依字母順序顯示。對於其他詞彙表，單字會以其顯示順序呈現。

## Version v 7.0.7.15

### FORTH

<a href="#">=</a>	<a href="#">-rot</a>	<a href="#">└</a>	<a href="#">:</a>	<a href="#">:</a>	<a href="#">:noname</a>	<a href="#">!</a>
<a href="#">?</a>	<a href="#">?do</a>	<a href="#">?dup</a>	<a href="#">└</a>	<a href="#">└"</a>	<a href="#">.s</a>	<a href="#">└</a>
<a href="#">(local)</a>	<a href="#">[</a>	<a href="#">[']</a>	<a href="#">[char]</a>	<a href="#">[ELSE]</a>	<a href="#">[IF]</a>	<a href="#">[THEN]</a>
<a href="#">l</a>	<a href="#">f</a>	<a href="#">f</a>	<a href="#">}transfer</a>	<a href="#">@</a>	<a href="#">*</a>	<a href="#">*/</a>
<a href="#">*/MOD</a>	<a href="#">/</a>	<a href="#">/mod</a>	<a href="#">#</a>	<a href="#">#!</a>	<a href="#">#&gt;</a>	<a href="#">#fs</a>
<a href="#">#s</a>	<a href="#">#tib</a>	<a href="#">+</a>	<a href="#">+!</a>	<a href="#">+loop</a>	<a href="#">+to</a>	<a href="#">&lt;</a>
<a href="#">&lt;#</a>	<a href="#">&lt;=</a>	<a href="#">&lt;&gt;</a>	<a href="#">=</a>	<a href="#">&gt;</a>	<a href="#">&gt;=</a>	<a href="#">&gt;BODY</a>
<a href="#">&gt;flags</a>	<a href="#">&gt;flags&amp;</a>	<a href="#">&gt;in</a>	<a href="#">&gt;link</a>	<a href="#">&gt;link&amp;</a>	<a href="#">&gt;name</a>	<a href="#">&gt;params</a>
<a href="#">&gt;R</a>	<a href="#">&gt;size</a>	<a href="#">0&lt;</a>	<a href="#">0&lt;&gt;</a>	<a href="#">0=</a>	<a href="#">1-</a>	<a href="#">1/F</a>
<a href="#">1+</a>	<a href="#">2!</a>	<a href="#">2@</a>	<a href="#">2*</a>	<a href="#">2/</a>	<a href="#">2drop</a>	<a href="#">2dup</a>
<a href="#">4*</a>	<a href="#">4/</a>	<a href="#">abort</a>	<a href="#">abort"</a>	<a href="#">abs</a>	<a href="#">accept</a>	<a href="#">adc</a>
<a href="#">afliteral</a>	<a href="#">aft</a>	<a href="#">again</a>	<a href="#">ahead</a>	<a href="#">align</a>	<a href="#">aligned</a>	<a href="#">allocate</a>
<a href="#">allot</a>	<a href="#">also</a>	<a href="#">analogRead</a>	<a href="#">AND</a>	<a href="#">ansi</a>	<a href="#">ARSHIFT</a>	<a href="#">asm</a>
<a href="#">assert</a>	<a href="#">at-xy</a>	<a href="#">base</a>	<a href="#">begin</a>	<a href="#">bq</a>	<a href="#">BIN</a>	<a href="#">binary</a>
<a href="#">bl</a>	<a href="#">blank</a>	<a href="#">block</a>	<a href="#">block-fid</a>	<a href="#">block-id</a>	<a href="#">buffer</a>	<a href="#">bye</a>
<a href="#">c,</a>	<a href="#">C!</a>	<a href="#">C@</a>	<a href="#">CASE</a>	<a href="#">cat</a>	<a href="#">catch</a>	<a href="#">CELL</a>
<a href="#">cell/</a>	<a href="#">cell+</a>	<a href="#">cells</a>	<a href="#">char</a>	<a href="#">CLOSE-DIR</a>	<a href="#">CLOSE-FILE</a>	<a href="#">cmove</a>
<a href="#">cmove&gt;</a>	<a href="#">CONSTANT</a>	<a href="#">context</a>	<a href="#">copy</a>	<a href="#">cp</a>	<a href="#">cr</a>	<a href="#">CREATE</a>
<a href="#">CREATE-FILE</a>	<a href="#">current</a>	<a href="#">dacWrite</a>	<a href="#">decimal</a>	<a href="#">default-key</a>	<a href="#">default-key?</a>	
<a href="#">default-type</a>		<a href="#">default-use</a>	<a href="#">defer</a>	<a href="#">DEFINED?</a>	<a href="#">definitions</a>	<a href="#">DELETE-FILE</a>
<a href="#">depth</a>	<a href="#">digitalRead</a>	<a href="#">digitalWrite</a>		<a href="#">do</a>	<a href="#">DOES&gt;</a>	<a href="#">DROP</a>
<a href="#">dump</a>	<a href="#">dump-file</a>	<a href="#">DUP</a>	<a href="#">duty</a>	<a href="#">echo</a>	<a href="#">editor</a>	<a href="#">else</a>
<a href="#">emit</a>	<a href="#">empty-buffers</a>		<a href="#">ENDCASE</a>	<a href="#">ENDOF</a>	<a href="#">erase</a>	<a href="#">ESP</a>

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

## 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 OBRANCH  
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  
SO\_L\_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.beginTransaction 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,  
S32CI, 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

1/F.....	37	insides.....	55	spi.....	57
asm.....	54	internals.....	55	SPI.....	43
bluetooth.....	54	interrupts.....	56	SPIFFS.....	57
drop.....	26	ledc.....	56	streams.....	57
dup.....	26	normal.....	29	structures.....	57
editor.....	55	oled.....	56	tasks.....	57
ESP.....	55	page.....	29	telnetd.....	58
F-.....	37	registers.....	56	to.....	28, 34
F*.....	37	riscv.....	56	value.....	28
F/.....	37	rot.....	26	variable.....	27
F+.....	37	rtos.....	57	visual.....	58
fconstant.....	37	SD.....	57	web-interface.....	58
FORTH.....	53	SD_MMC.....	57	WiFi.....	58
fsqrt.....	37	Serial.....	57	xtensa.....	58
fvariable.....	37	set-precision.....	36	+to.....	34
httpd.....	55	sockets.....	57		