ESP32forth 帳本

版本 1.0 - 2023 年 10 月 24 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

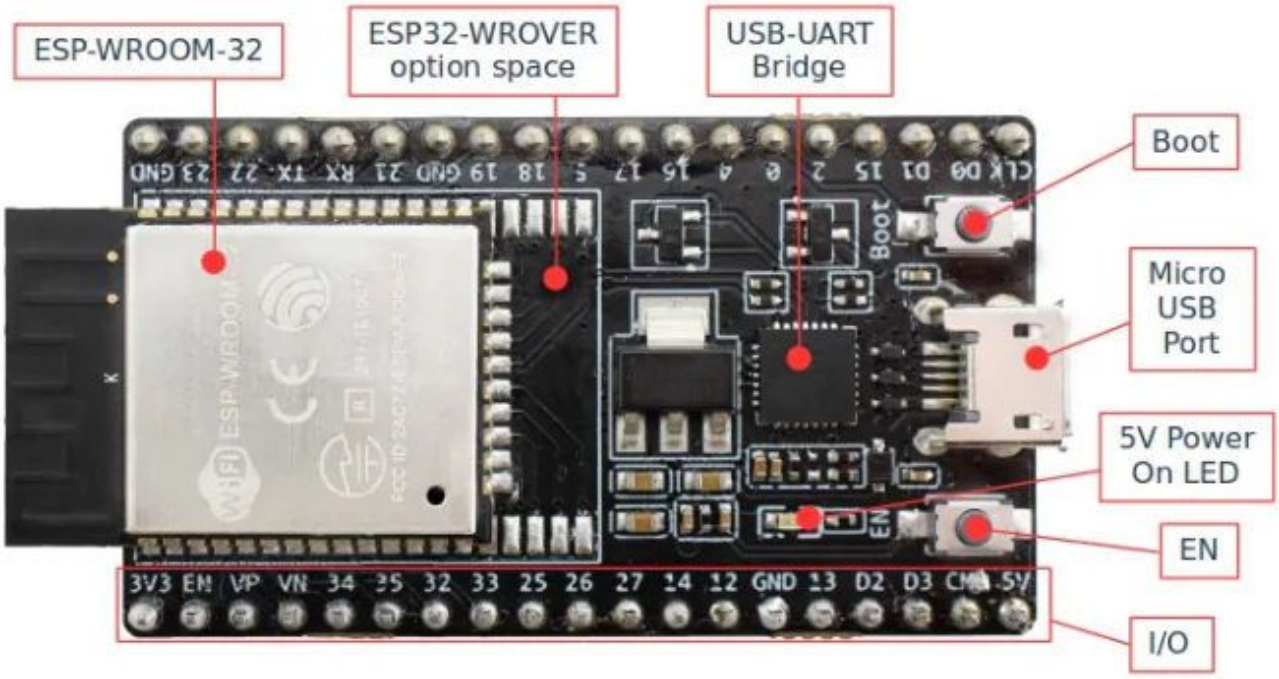
ESP32 卡的發現

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。

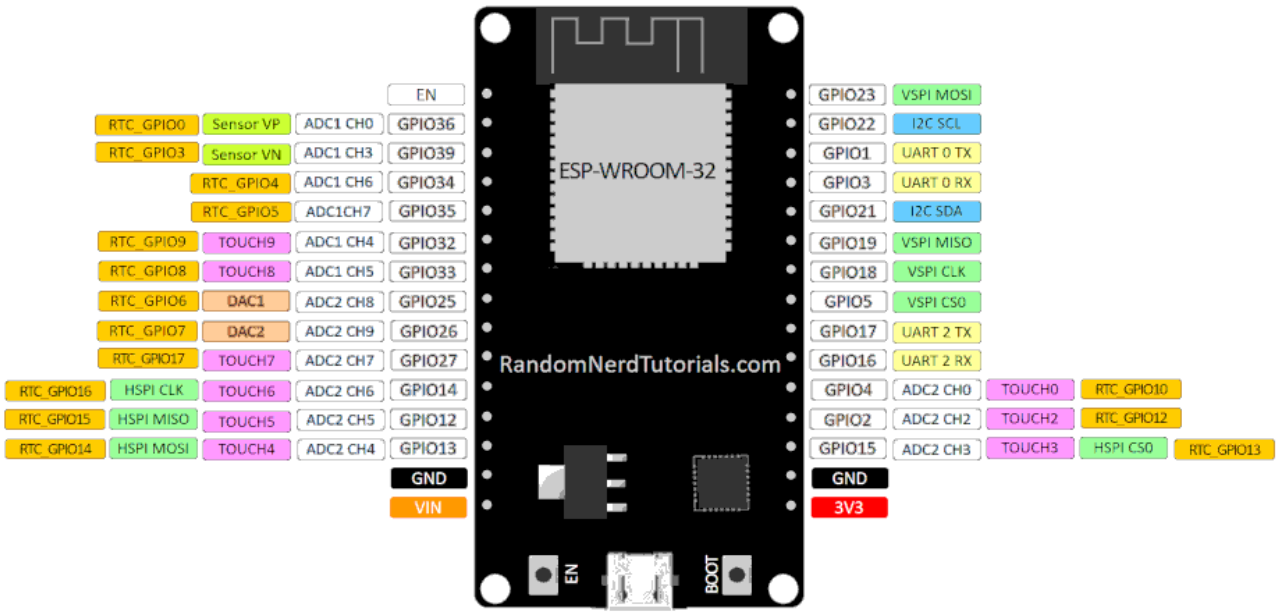
大多數連接器都有多種用途。

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。

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
```

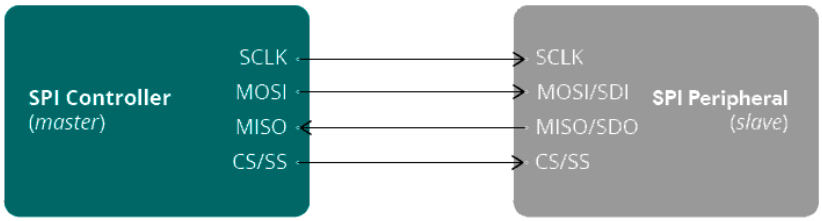
spi 詞彙表中定義的單字：

```
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
```

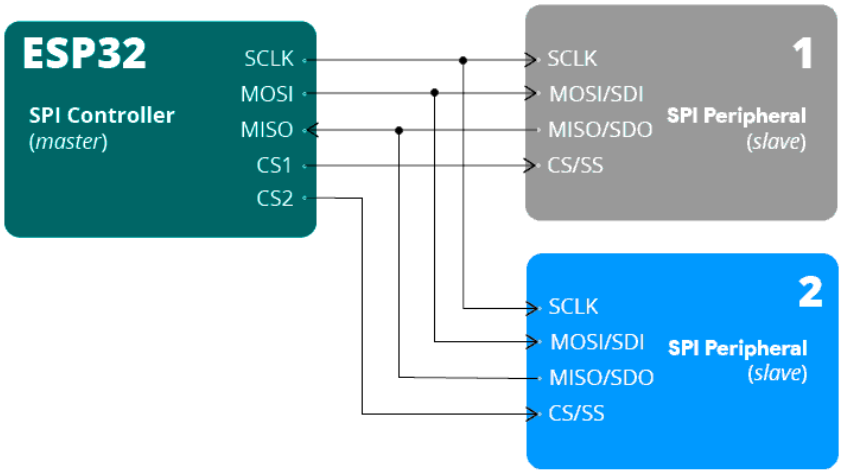
現在您可以透過 SPI 連接埠驅動擴展，例如 MAX7219 LED 顯示器。

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。



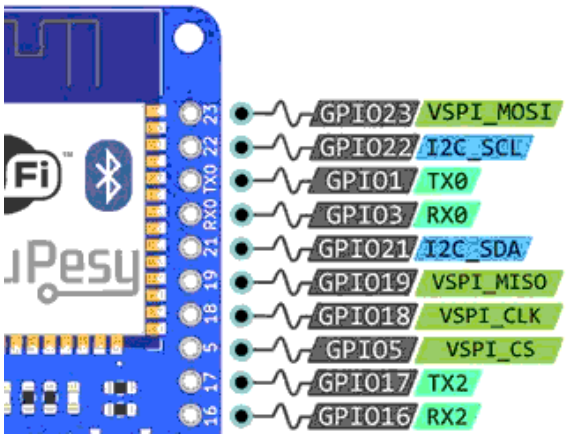
你可以有幾個奴隸。從設備可以是感測器、顯示器、microSD 卡等，或其他微控制器。這意味著您可以將 ESP32 連接到**多個裝置**。



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：



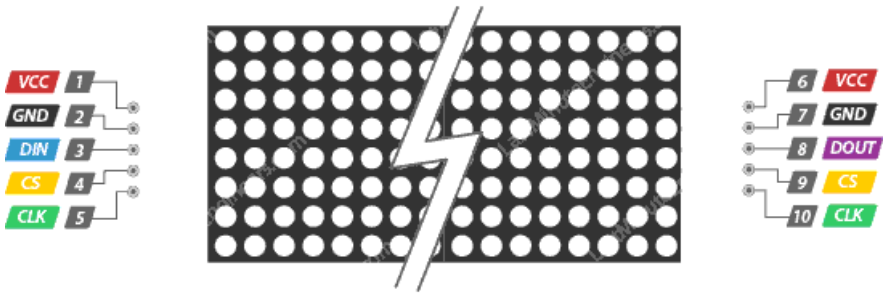
因此，使用 ESP32forth，我們可以定義指向這些 VSPI 引腳的常數：

```
\ 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

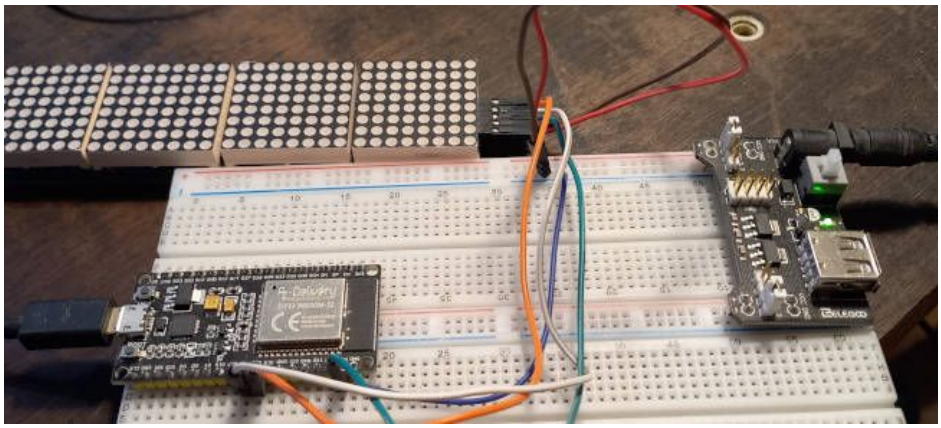
以下是 MAX7219 模組上的 SPI 連接埠連接器圖：



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。

詞彙索引