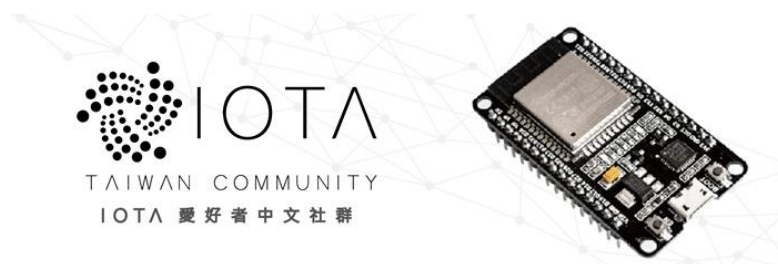


深入淺出 ESP32 硬體開發

本教學章節旨在指導用戶建置 ESP32 硬體開發的軟體環境，通過一個簡單的範例展示如何使用 **ESP-IDF** (Espressif IoT Development Framework) 配置選單，並編譯、下載韌體至 ESP32 開發板等步驟。



■ 概述

ESP32 SoC 芯片支援以下功能：

- 2.4 GHz Wi-Fi
- 藍牙 4.2 標準
- 高性能雙核
- 超低功耗協處理器
- 支援多種外接設備

ESP32 採用 40 nm 工藝製成，具有最佳的功耗性能、射頻性能、穩定性、通用性和可靠性，適用於各種應用場景和不同功耗需求。

樂鑫為用戶提供完整的軟、硬體資源，進行 ESP32 硬體設備的開發。其中，樂鑫的軟體開發環境 ESP-IDF 旨在協助用戶快速開發物聯網(IoT) 應用，可滿足用戶對 Wi-Fi、藍牙、低功耗等方面的要求。

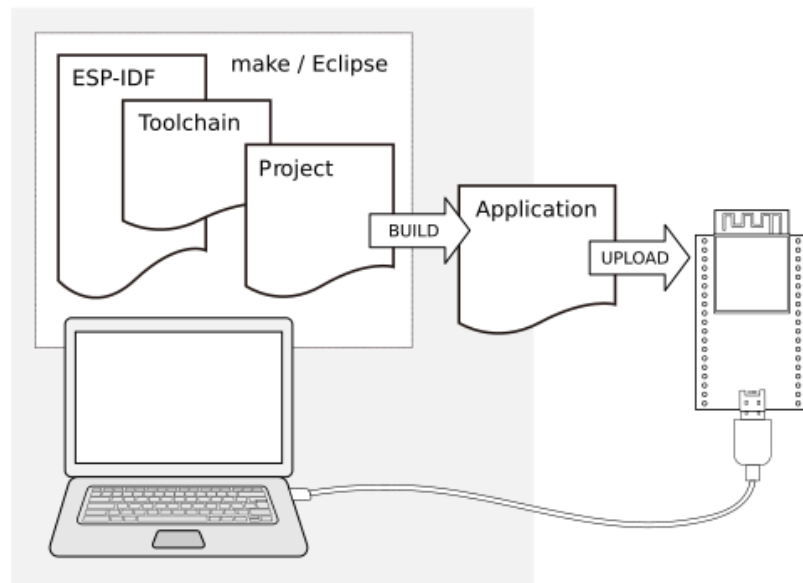
■ 準備工作

硬體：

- 一款 **ESP32** 開發板
- **USB 數據線** (USB A/Micro USB B)
- PC (Windows、Linux 或 Mac OS)

軟體：

- 設置**工具鏈(Toolchain)** - 用於編譯 ESP32 程式碼。
- **編譯工具** - CMake 和 Ninja 編譯工具 - 用於編譯 ESP32 應用程式。
- 獲取 **ESP-IDF** 軟體開發框架。該框架已經基本包含 ESP32 使用的 API（軟體庫和來源程式碼）和執行**工具鏈**的腳本。
- 安裝 C 語言編程的**文本編輯器**，例如 [Eclipse](#),或自己熟悉的文字編輯器工具,vi/vim/nano/Notepad++...等。



ESP32 應用程式開發

■ 詳細安裝步驟

請根據下方詳細步驟，完成安裝過程。

設置開發環境


- 第一步：設置工具鏈
- 第二步：獲取 [ESP-IDF](#)
- 第三步：設置環境變數
- 第四步：安裝 [Python](#) 軟體

建立您的第一個應用

- 第五步：開始建立工程
- 第六步：連接設備
- 第七步：配置
- 第八步：編譯
- 第九步：燒錄到設備
- 第十步：監視器

第一步：設置工具鏈 (Toolchain)

工具鏈是一套用於編譯程式碼和應用程式的程式。為了加快開發進度，您可以直接使用樂鑫提供的預製工具鏈。請根據您的操作系統，點擊下方對應的鏈接，並按照鏈接中的指導進行安裝。

		
<p>Windows</p> <p>https://docs.espressif.com/projects/esp-idf/zh_CN/latest/get-started/windows-setup.html</p>	<p>Linux</p> <p>https://docs.espressif.com/projects/esp-idf/zh_CN/latest/get-started/linux-setup.html</p>	<p>Mac OS</p> <p>https://docs.espressif.com/projects/esp-idf/zh_CN/latest/get-started/macos-setup.html</p>

Windows 平台工具鏈的標準設置

ESP-IDF 需要安裝必要的工具，以編譯 ESP32 韌體，包括：Git，交叉編譯器，以及 CMake 構建工具。本文將對這些工具一一說明。

小提醒：

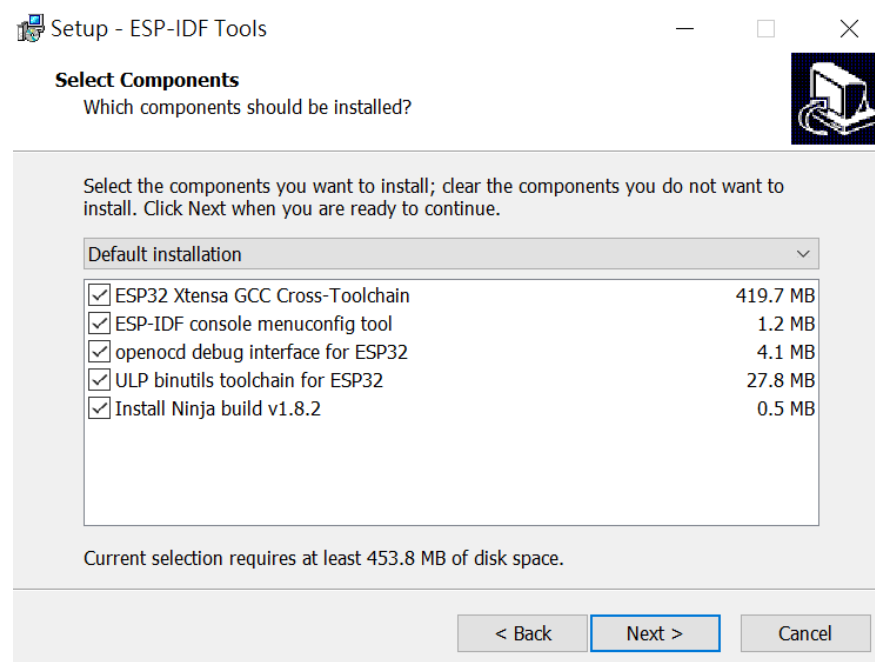
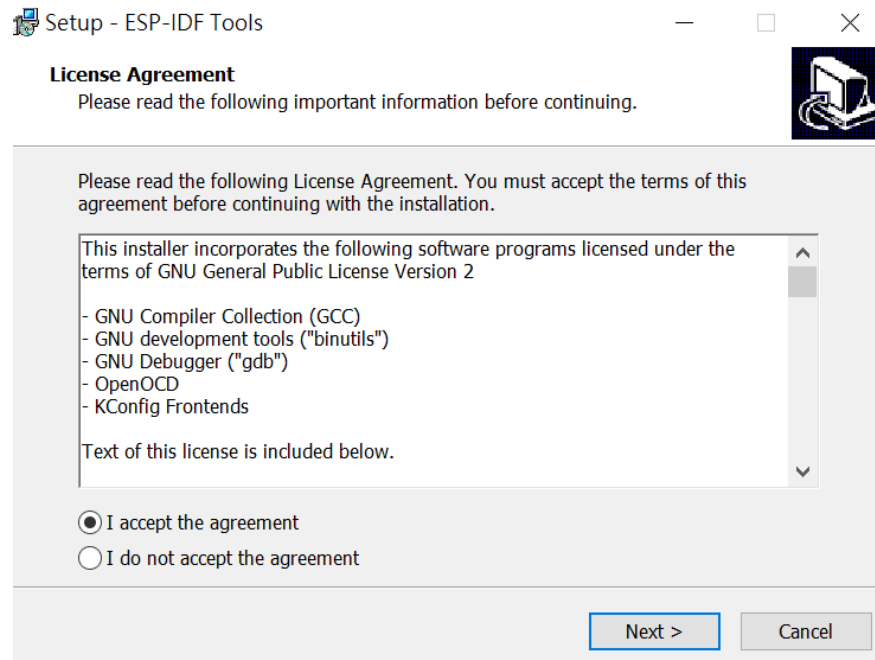
基於 CMake 的構建系統僅支持 64 位版本 Windows。

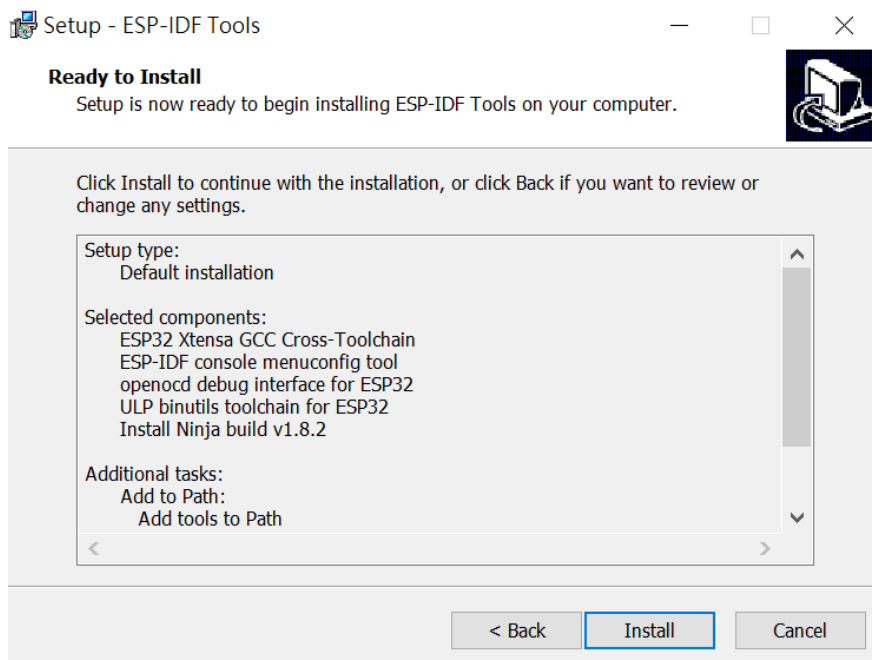
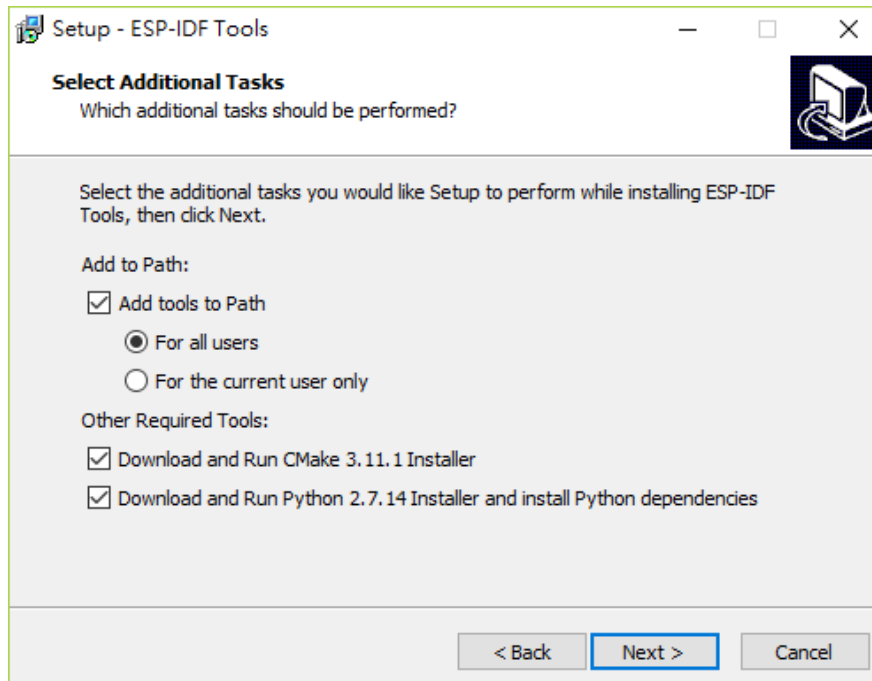
ESP-IDF 工具安裝器

安裝 ESP-IDF 必備工具最簡易的方式是下載 ESP-IDF 工具安裝器，下載網址：
<https://dl.espressif.com/dl/esp-idf-tools-setup-1.2.exe>

安裝器會自動安裝 ESP32 Xtensa gcc 工具鏈，Ninja 編譯工具，以及名為 `mconf-idf` 的配置工具。此外，如果你的電腦還未安裝有關 CMake 和 Python2.7，它還會自動下載和執行與之對應的安裝器。安裝器預設會更新 Windows `Path` 環境變數，因而上

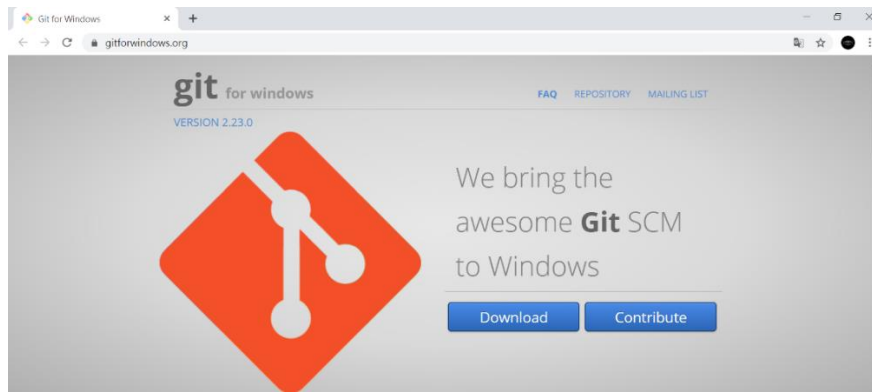
工具也可在其他環境中執行。如果禁止該選項，則需自行設置 ESP-IDF 所使用的環境（終端或所選 IDE），並配置正確的路徑。





下載和安裝 Git

ESP-IDF 工具安裝器並不會安裝 Git 套件。你可以通過 [Git For Windows](https://gitforwindows.org/) (<https://gitforwindows.org/>) 下載和安裝 Windows 平台的命令列 Git 工具（包括“Git Bash”終端）。下載後直接行安裝,安裝過程都以預設值下一步即可。



git 下載畫面

第二步：ESP-IDF 下載

ESP-IDF 全名稱 *Espressif IoT Development Framework* 是樂鑫 ESP32 推出的新一代 SDK，基於 FreeRTOS 系統，在上一代 SDK 基礎上做了眾多更新和改進，整合了眾多元件。支援在 Windows、Linux 和 MacOS 下基於 ESP-IDF 開發。ESP-IDF 原始程式碼已託管在 github 上，可在 <https://github.com/espressif/esp-idf> 下載，在命令列輸入 `git clone --recursive https://github.com/espressif/esp-idf` 下載，注意 `git clone` 必須新增 `--recursive`，否則無法拉取 components 目錄下的全部模組。

請將 ESP-IDF 下載到您的本地。打開終端，切換到你要存放 ESP-IDF 的工作目錄，使用命令 `git clone`

➤ Linux 和 MacOS 操作系統

打開終端，執行以下命令：

```
cd ~/esp
git clone --recursive https://github.com/espressif/esp-idf.git
```

ESP-IDF 將下載至 `~/esp/esp-idf`。

➤ Windows 操作系統

打開命令提示字元(CMD)，執行以下命令：

```
mkdir %userprofile%\esp
cd %userprofile%\esp
git clone -b v3.2.2 --recursive https://github.com/espressif/esp-idf.git
```

ESP-IDF 將下載至 `%userprofile%\esp\目錄裡`。-b v3.2.2 表示下載 3.2.2 版本，記得 `--recursive` 拉取 components 目錄下的全部模組。

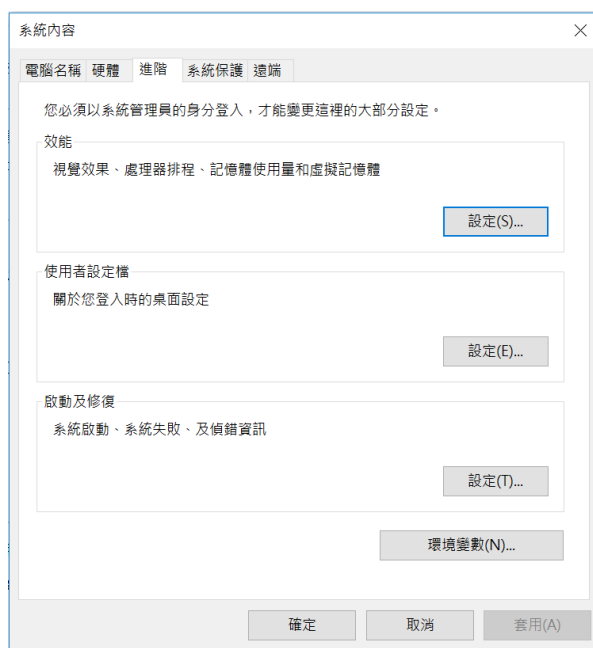
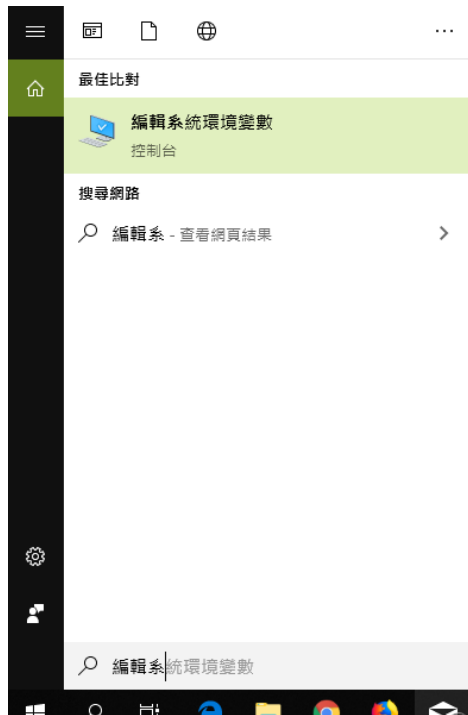
第三步：設置環境變數

請在您的 PC 系統上設配置增加 **IDF_PATH** 和 **idf.py PATH** 環境變數，否則無法順利編譯。使用基於 CMake 的構建系統和 idf.py 工具，用戶需修改兩處系統環境變數：

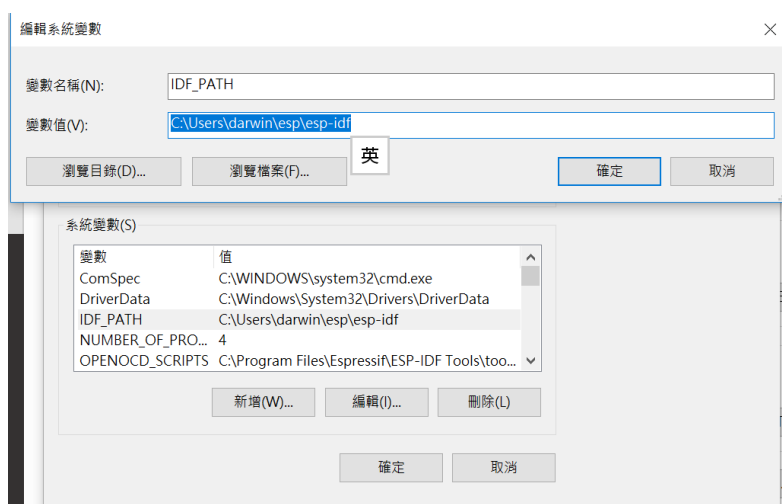
- `IDF_PATH` 應設置為 ESP-IDF 根目錄的路徑。
- `PATH` 應包括同一 `IDF_PATH` 目錄下的 `tools` 目錄路徑。

➤ Windows 操作系統

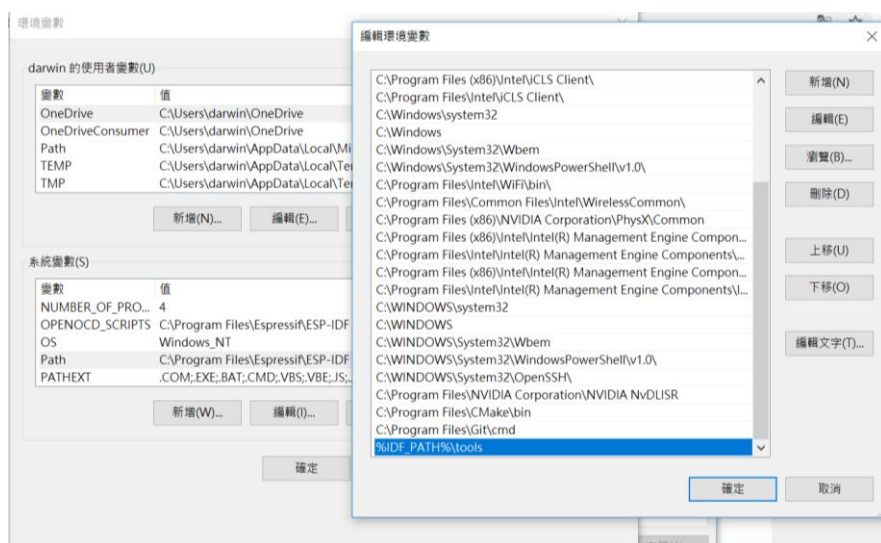
在 Windows 10 操作系統下設置環境變數，用戶應在開始選單下搜索 【編輯系統環境變數】。在系統內容→選取【進階】標籤→【環境變數(N)..】按鈕。



- 在系統變數(S)中點擊【新增(W)】，新增系統變數名稱為 `IDF_PATH`，具體設置為包含 ESP-IDF 的目錄，例如，`C:\Users\<user-name>\esp\esp-idf`。



接下來一樣在系統變數(S)中找到 `Path` 環境變數，雙擊進行編輯。在末尾增加 `%IDF_PATH%\tools`，這樣你就可以通過 Windows 命令窗口執行 `idf.py` 等其他工具了。



小提醒：

如果之前命令字元視窗有開著，建議關掉重開，剛剛所設定的環境變數才會生效。

➤ Linux 和 MacOS 操作系統

要設置 `IDF_PATH`，並在 `PATH` 中增加 `idf.py`，請將以下兩行程式碼增加至你的 `~/.profile` 文件中：

```
export IDF_PATH=~/.esp/esp-idf
export PATH="$IDF_PATH/tools:$PATH"
```


執行以下命令來檢查 `IDF_PATH` 設置是否正確:

```
printenv IDF_PATH
```

此處應打印出此前在 `~/.profile` 文件中輸入（或手動設置）的路徑。

為確認 `idf.py` 目前是否在 `PATH` 中，你可以執行以下命令:

```
which idf.py
```

這裡，應打印出類似 `${IDF_PATH}/tools/idf.py` 的路徑。

如果不想修改 `IDF_PATH` 或 `PATH`，你可以在每次重啟或退出後在終端中手動輸入:

```
export IDF_PATH=~/.esp/esp-idf
export PATH="${IDF_PATH}/tools:$PATH"
```

第四步：安裝 Python 軟體

ESP-IDF 所需的 Python 軟體套件位於 `IDF_PATH/requirements.txt` 中。您可以執行以下命令進行安裝：

```
cd %userprofile%\esp\esp-idf
python -m pip install --user -r requirements.txt
python -m pip install --upgrade pip
```

```
Collecting pip
  Downloading https://files.pythonhosted.org/packages/4a/08/6ca123073af4ebc4c5488a5bc8a010ac57aa39ce4d3c8a931ad504de4185/pip-19.3-py2.py3-none-any.whl (1.4MB)
    100% |#####| 1.4MB 544kB/s
Installing collected packages: pip
  Found existing installation: pip 9.0.1
    Uninstalling pip-9.0.1:
      Successfully uninstalled pip-9.0.1
Successfully installed pip-19.3
```

第五步：開始建立小程序 (Hello-World)

現在，您可以開始準備開發 ESP32 應用程式了。您可以直接從 ESP-IDF 中 `examples` 目錄下的 `get-started/hello_world` 開始。將 `get-started/hello_world` 複製至您本地的 `~/esp` 目錄下：

➤ Linux 和 MacOS 操作系統

```
cd ~/esp
cp -r $IDF_PATH/examples/get-started/hello_world .
```

➤ Windows 操作系統

```
cd %userprofile%\esp
xcopy /e /i %IDF_PATH%\examples\get-started\hello_world hello_world
```

❏ 選取 命令提示字元

```
C:\Users\darwin\esp>xcopy /e /i %IDF_PATH%\examples\get-started\hello_world hello_world
C:\Users\darwin\esp\esp-idf\examples\get-started\hello_world\CMakeLists.txt
C:\Users\darwin\esp\esp-idf\examples\get-started\hello_world\Makefile
C:\Users\darwin\esp\esp-idf\examples\get-started\hello_world\README.md
C:\Users\darwin\esp\esp-idf\examples\get-started\hello_world\main\CMakeLists.txt
C:\Users\darwin\esp\esp-idf\examples\get-started\hello_world\main\component.mk
C:\Users\darwin\esp\esp-idf\examples\get-started\hello_world\main\hello_world_main.c
已複製 6 個檔案
```

ESP-IDF 的 **examples** 目錄下有一系列範例，都可以按照上面的方法進行建立。您可以按照上述方法複製並執行其中的任何範例，也可以直接編譯範例，無需進行複製。

重要：

ESP-IDF 編譯系統不支援帶有空格的路徑。

第六步：連接設備

現在，請將您的 ESP32 開發板連接到 PC，並查看開發板使用的序列埠。通常，序列埠在不同操作系統下顯示的名稱有所不同：

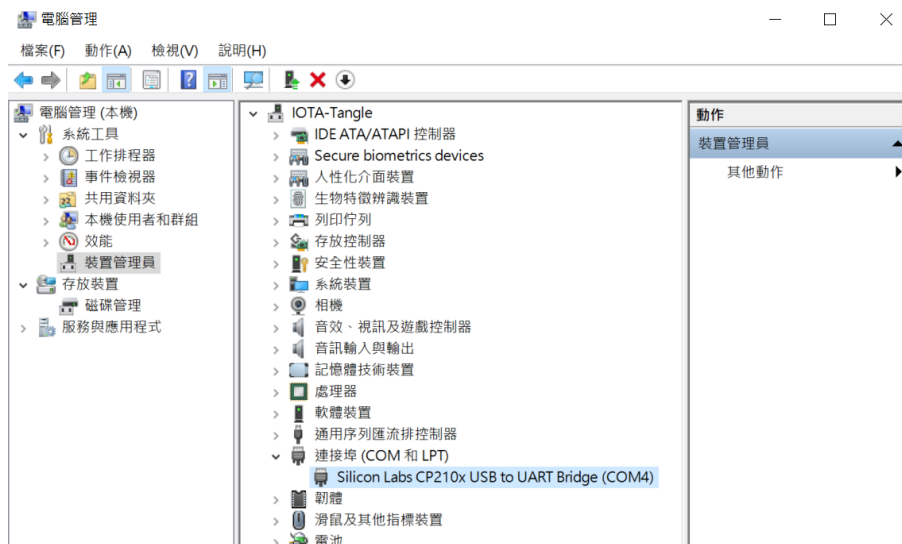
- **Windows** 操作系統：COM1 等
- **Linux** 操作系統：以 /dev/tty* 開始
- **MacOS** 操作系統：以 /dev/tty* 或 /dev/cu 開始

小提醒：

請記住序列埠名稱，您會在下面的步驟中用到。

在 Windows 上查看序列埠

檢查 Windows 設備的 COM 埠列表。拔除 ESP32 與 PC 的連接，然後重新連接，查看哪個序列埠從列表中消失，然後再次出現。以下為 ESP32 DevKitC 序列埠（這裡是 COM4）：



設備管理中 ESP32-DevKitC 的 USB 至 UART Bridge

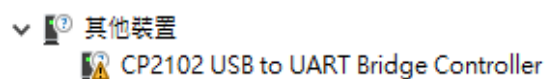
確認序列埠內容

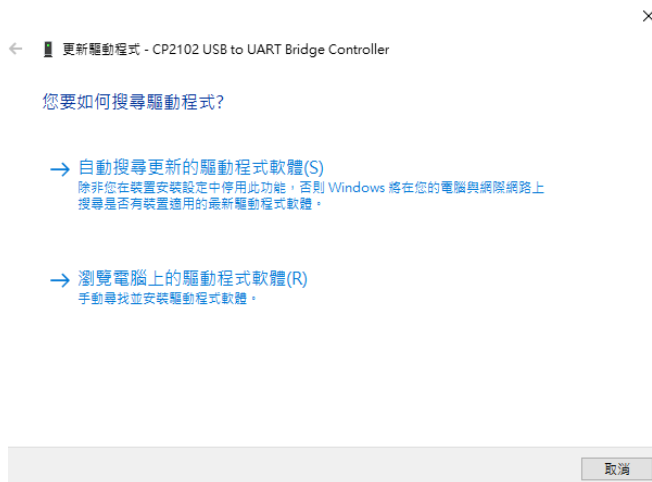
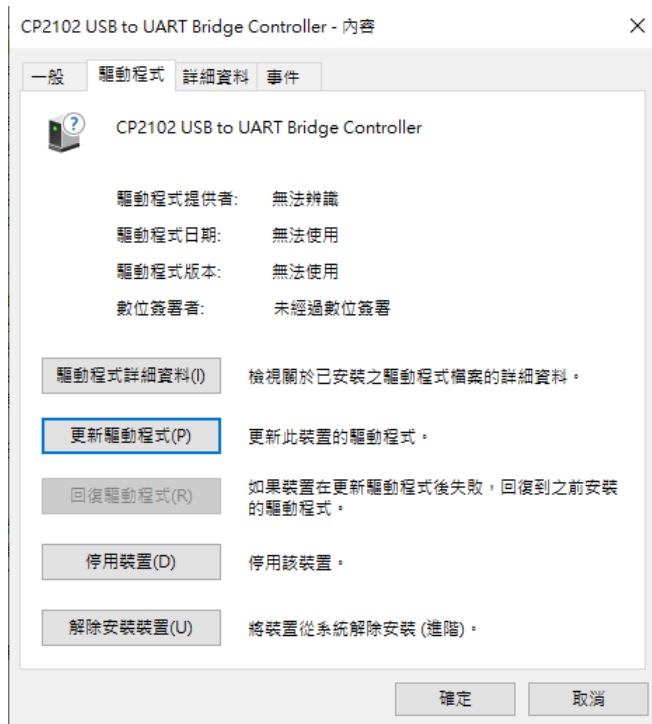
配置序列埠：鮑率= 115200，數據位= 8，停止位= 1，奇偶校驗= N。



如果沒偵測到介面, 如出現下圖所示的話, 可以指示線上自動去安裝更新驅動程式, 或者手動至官方網站下載安裝驅動程式。下載網址:

<https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers>





第七步：配置

完成以上配置後，切換至 `hello_world` 目錄，並執行配置工具命令 `idf.py menuconfig`，預設會進入 `configuration` 介面。我們可以使用預設設定直接退出。

➤ Linux 和 MacOS 操作系統

```
cd ~/esp/hello_world
idf.py menuconfig
```

如果您的預設 Python 版本為 3.0 以上，可能需要執行 `python2 idf.py`

➤ Windows 操作系統

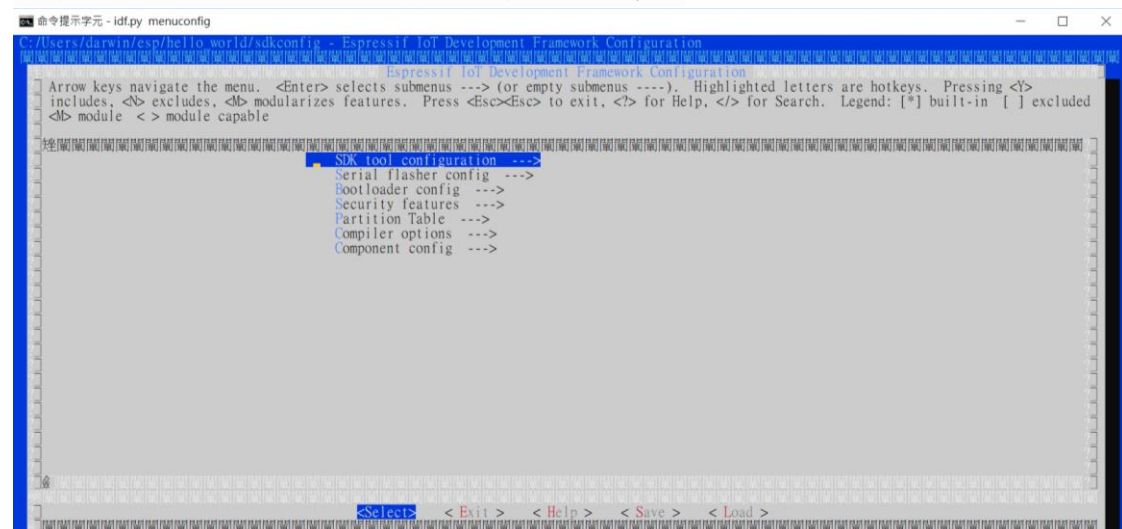
```
cd %userprofile%\esp\hello_world
idf.py menuconfig
```

如出現要求 python 安裝 `requirements.txt` 的話, 依指令執行安裝(直接複製/貼上)

```
C:\Python27\python.exe -m pip install --user -r C:\Users\ITS_User\esp\esp-idf\requirements.txt
```

```
The following Python requirements are not satisfied:
click==5.0
pyserial==3.0
future==0.15.2
cryptography==2.1.4
pyparsing==2.0.3,<2.4.0
pyelftools==0.22
Please refer to the Get Started section of the ESP-IDF Programming Guide for setting up the required packages.
Alternatively, you can run "C:\Python27\python.exe -m pip install --user -r C:\Users\ITS_User\esp\esp-idf\requirements.txt" for resolving the issue.
ESP-IDF v4.1-dev-437-gd2ad0f077
```

如果之前的步驟都正確，則會顯示下面的選單：



IDF 配置主選單

menuconfig 工具的常見操作如下。

上下箭頭：移動

Enter 鍵：進入子選單

ESC 鍵：返回上層選單或退出

英文問號：調出幫助選單（退出幫助選單，請按 Enter 鍵）。

空格、：啟用/禁用配置選項 Y 鍵或 N 鍵[*]

/ 鍵：尋找配置項目

Python 2.7 安裝程式將嘗試配置 Windows，將 `.py` 文件與 Python 2 關聯起來。如果其他程式（比如 Visual Studio Python 工具）曾關聯了其他版本 Python，則 `idf.py` 可能無法正常執行（文件將在 Visual Studio 中打開）。這種情況下，您可以選擇每次都執行一遍，或更改 Windows 的關聯檔案設置。`C:\Python27\python idf.py.py`

注意

如果您使用的是 ESP32-DevKitC（板載 ESP32-SOLO-1 模組），請在燒寫範例程式前，前往 `menuconfig` 中使能單核模式（`CONFIG_FREERTOS_UNICORE`）。

第八步：編譯二進制執行檔案(Binary)

請使用以下命令，進行編譯：

```
idf.py build
```

執行以上命令可以編譯應用程式和所有 ESP-IDF 元件，接著生成 bootloader、分區表和應用程式二進制檔案。

```
C:\Users\darwin\esp\hello_world>idf.py build
```

```
Checking Python dependencies...
```

```
Python requirements from C:\Users\darwin\esp\esp-idf\requirements.txt are satisfied.
```

```
Running ninja in directory C:\Users\darwin\esp\hello_world\build
```

```
Executing "ninja all"...
```

```
[0/1] Re-running CMake...
```

```
.....
```

```
.....
```

```
[43/43] Generating bootloader.bin
```

```
esptool.py v2.6
```

```
[736/736] Generating hello-world.bin
```

```
esptool.py v2.6
```

```
Project build complete. To flash, run this command:
```

```
..\esp-idf\components\esptool_py\esptool\esptool.py -p (PORT) -b 460800 write_flash --  
flash_mode dio --flash_size detect --flash_freq 40m 0x1000 build\bootloader\bootloader.bin  
0x8000 build\partition_table\partition-table.bin 0x10000 build\hello-world.bin  
or run 'idf.py -p (PORT) flash'
```

如果一切正常，編譯完成後將產生三個檔案 bin 檔案分別為：

位於 build/bootloader 目錄下的 bootloader.bin

位於 build/partition_table 目錄下的 partition-table.bin

位於 build 目錄下的 hello-world.bin

第九步：燒錄(上傳)到 ESP32 設備

請使用以下命令，將剛剛生成的二進制檔案燒錄至您的 ESP32 開發板：

```
idf.py [-p PORT] [-b BAUD] flash
```

請將 PORT 替換為 ESP32 開發板的序列埠名稱，例如 COM4。您還可以將 BAUD 替換為您希望的燒錄速率。預設速率為 **460800**。如果您只有一個序列埠在使用可省略不指定系統會自動識別 COM Port，如下所示

```
C:\Users\darwin\esp\hello_world>idf.py flash  
Checking Python dependencies...  
Python requirements from C:\Users\darwin\esp\esp-idf\requirements.txt are satisfied.  
Running ninja in directory C:\Users\darwin\esp\hello_world\build  
Executing "ninja all"...  
[1/3] Performing build step for 'bootloader'  
esptool.py -p COM4 -b 460800 write_flash --flash_mode dio --flash_size detect --flash_freq  
40m 0x1000 bootloader/bootloader.bin 0x8000 partition_table/partition-table.bin 0x10000  
hello-world.bin  
esptool.py v2.6  
Serial port COM4  
Connecting.....  
Detecting chip type... ESP32  
...  
...  
Leaving...  
Hard resetting via RTS pin...  
Done
```

如果一切順利，燒錄完成後，開發板將會自動復位，應用程式“hello_world”開始執行 (每 10 秒 say Hello world!)。

```

(158) esp_image: segment 5: paddr=0x00032578 vaddr=0x40087048 size=0x02518 ( 9496) load
0x40087048: multi_heap_internal_unlock at C:/Users/darwin/esp/esp-idf/components/heap/multi_heap.c:380
(inlined by) multi_heap_free_impl at C:/Users/darwin/esp/esp-idf/components/heap/multi_heap.c:505
(168) boot: Loaded app from partition at offset 0x10000
(168) boot: Disabling RNG early entropy source...
(170) cpu_start: Pro cpu up.
(174) cpu_start: Starting app cpu, entry point is 0x40080e8c
0x40080e8c: call_start_cpu1 at C:/Users/darwin/esp/esp-idf/components/esp32/cpu_start.c:246
(0) cpu_start: App cpu up.
(184) heap_init: Initializing. RAM available for dynamic allocation:
(191) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
(197) heap_init: At 3FFB2EF8 len 0002D108 (180 KiB): DRAM
(203) heap_init: At 3FFB0440 len 00003AE0 (14 KiB): D/IRAM
(210) heap_init: At 3FFB4350 len 0001BCB0 (111 KiB): D/IRAM
(216) heap_init: At 40089560 len 00016AA0 (90 KiB): IRAM
(222) cpu_start: Pro cpu start user code
(240) cpu_start: Starting scheduler on PRO CPU.
(0) cpu_start: Starting scheduler on APP CPU.
Hello world!
This is ESP32 chip with 2 CPU cores, WiFi/BT/BLE, silicon revision 1, 4MB external flash
Restarting in 10 seconds...
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...
Restarting in 6 seconds...
Restarting in 5 seconds...
Restarting in 4 seconds...
Restarting in 3 seconds...
Restarting in 2 seconds...
Restarting in 1 seconds...

```

第十步：監視器(monitor)

您可以透過序列埠輸出查看 “hello_world” 的執行情況。注意，不要忘記將 **port** 替換為您的序列埠名稱。idf.py monitor 執行該命令後，IDF 監視器應用程式將啟動：

```

C:\Users\darwin\esp\hello_world>idf.py monitor

Running idf_monitor in directory [...]/esp/hello_world/build
Executing "python [...]/esp-idf/tools/idf_monitor.py -b 115200
[...]/esp/hello_world/build/hello-world.elf"...
--- idf_monitor on /dev/ttyUSB0 115200 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
ets Jun  8 2016 00:22:57
...

```

此時，您就可以在啟動日誌和診斷日誌之後，看到打印的“Hello world!”了。

```

...
Hello world!
Restarting in 10 seconds...
I (211) cpu_start: Starting scheduler on APP CPU.
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...

```

Linux/MAC 系統您可使用快捷鍵 **Ctrl+]** ，退出 IDF 監視器。

Windows 系統則使用 **Ctrl+c** ，退出 IDF 監視器。

小提醒：

您也可以執行以下命令，一次性執行編譯、燒錄和查看過程：

```
idf.py build flash monitor
```

恭喜，您已完成 **ESP32** 的入門學習！

現在，您可以嘗試一些其他 [examples](#)，或者直接開發自己的應用程式。

<https://github.com/espressif/esp-idf/tree/4dac7c7df/examples>

專案 2 – 自動傳送溫濕度訊息

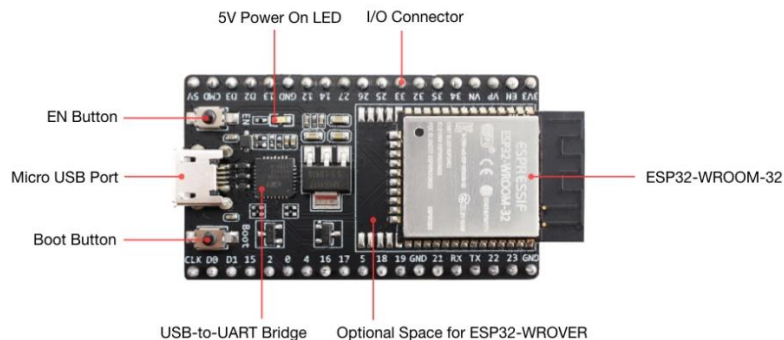
自動傳送溫濕度訊息程式改良自 TanglePigeon 程式。這一個範例項目，展示如何將溫濕度訊息自動定時發送到 Tangle 網路上，可以透過 GPIO（BOOT 按鈕）或 Timer 觸發訊息。該訊息採用 JSON 格式，如下所示：

```
Message "TanglePigeon": "v0.0.1"
  ○ Trytes "Device": "ESP32-WROOM32D"
  ○ Text "Trigger": "POWER_ON"
  ☑ JSON "LocalTime": "2020-01-11 19:02:56"
      "TimeZone": "CST-8"
      "Data": "Temperature=23.8*C humidity=66%"
```

注意：由於 IRI 快照是零值交易，因此將在 IRI 快照後刪除消息。

■ 準備材料

- ESP32 Wroom 32D 開發板



規格細項可參考 ESP 官方文件說明：

https://docs.espressif.com/projects/esp-idf/zh_CN/latest/hw-reference/modules-and-boards.html#wroomsolowrover-pico

- DHT11 感測器模組 (DHT = Digital Humidity Temperature)

DHT11 是一個數位溫濕度感測模組。主要用來量測周遭環境的溫溼度狀態,將所量測到的溫溼度數據轉換為數位訊號,再由 data pin 接腳將數據傳送出去, 具有極高的可靠性和卓越長期穩定性。該模組具有體積小、功耗低、反應快及抗干擾能力強等優點。

規格：

供電電壓：3.3~5.5V DC

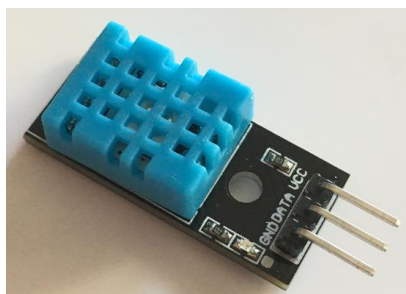
測量範圍：濕度 20-90%RH，溫度 0~50°C

精度：濕度 $\pm 5\%$ RH，溫度 $\pm 2^{\circ}\text{C}$

分辨率：濕度 1%RH，溫度 1°C

長期穩定性： $<\pm 1\%$ RH /年

傳輸距離可長達 20m 以上



DHT11 感測器模組

■ 安裝 DHT11 感測器

使用 3 根跳線將 DHT11 感測器連接到 ESP32 開發板。

DHT11 感測器接地腳位(GND) - ESP32 開發板接地腳位(GND)

DHT11 感測器資料腳位(DATA) - ESP32 開發板 GPIO 16 腳位

DHT11 感測器電壓腳位(VCC) - ESP32 開發板 5V 電壓腳位

■ 建置

請根據下方詳細步驟，完成安裝過程。

第一步：下載 `tangle_pigeon_dht11`

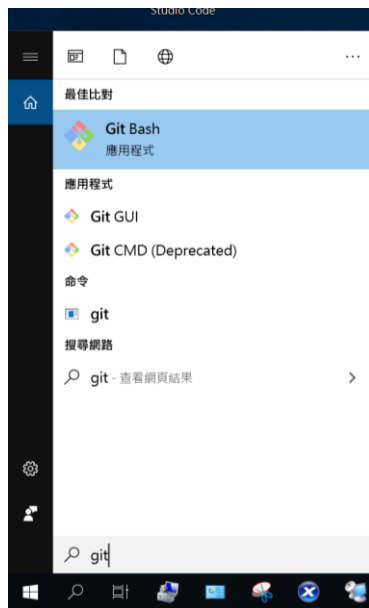
➤ Windows 操作系統

打開命令提示字元(CMD)，執行以下命令：

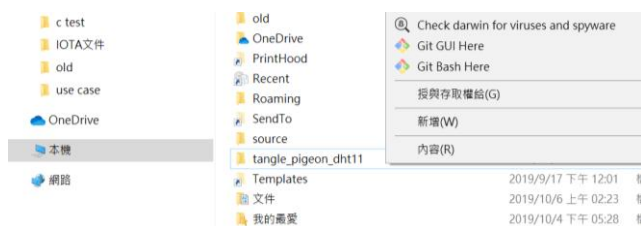
```
cd %userprofile%\  
git clone --recursive https://github.com/gotangle/tangle_pigeon_dht11.git
```

將 `tangle_pigeon_dht11` 下載至`%userprofile%\`目錄，記得要多加`--recursive` 參數這樣才會完整下載 `components` 目錄下的全部模組。

下載完成後，因該專案需要一開始執行初始化動作，故先開啟 `Git Bash` 視窗，切換至該專案目錄裡並執行 `init.sh shell script` 檔案，執行完後關閉視窗即可。



或者以目錄方式按右鍵點選 **Git Bash Here** 直接執行 **Git Bash** 並切換至目錄。



執行 `./init.sh` 檔案; 執行完後關閉視窗即可。

```
MINGW64:/c/Users/darwin/tangle_pigeon_dht11
darwin@IOTA-Tangle MINGW64 ~/tangle_pigeon_dht11 (master)
$ pwd
/c/Users/darwin/tangle_pigeon_dht11
darwin@IOTA-Tangle MINGW64 ~/tangle_pigeon_dht11 (master)
$ ./init.sh
```

第二步：設定

切回到 CMD 提示字元視窗，在此步驟中，您需要主要設置 WiFi、IRI 節點和喚醒時間等選項，其它預設值即可。

```
cd tangle_pigeon_dht11
C:\Users\darwin\tangle_pigeon_dht11>idf.py menuconfig
```

- 選取 **Tangle Pigeon** 選項。

```

SDK tool configuration --->
Tangle Pigeon --->
Serial flasher config --->
Bootloader config --->
Security features --->
Partition Table --->
Compiler options --->
Component config --->

```

- WiFi 選項→輸入您的無線連線資訊，如 SSID 及 WIFI 密碼。

```

WiFi --->
SNTP --->
IRI Node --->
(5) Wake up time (minutes)
flex_trit encoding (3 trits per byte) --->
[ ] Enable DEBUG in CClient

```

- IRI Node 選項→在這裡重點在於接收位址(The receiver address)這裡輸入您的訊息想要接收的位址(81 字元)，其它維持預設值即可。

```

(nodes.thetangle.org) IRI Node URI
(443) Port Number of IRI Node
[*] Use HTTPS
(HPAMGYV1PVHVESXPXQFEKWVUQVG9GJBFWSFYT9SY9J SWNXFTV9MJFDFM
(3) Depth for tips selection
(14) Minimum Weight Magnitude for PoW

```

- Wake up time 選項→設定喚醒時間,這裡預設是 5 分鐘。

```

WiFi --->
SNTP --->
IRI Node --->
(5) Wake up time (minutes)
(16) DHT11 data pin
flex_trit encoding (3 trits per byte) --->
[ ] Enable DEBUG in CClient

```

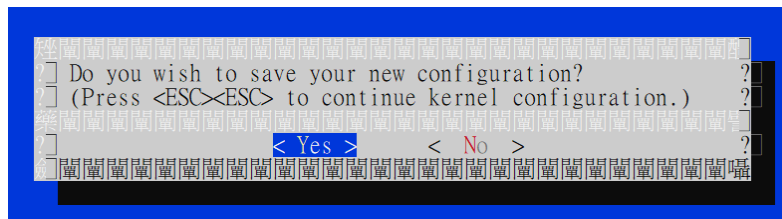
- DHT11 感測器選項→預設值 16 是 DHT11 Data 接腳(pin 腳位)接至開發板 16 編號即可。

```

WiFi --->
SNTP --->
IRI Node --->
(5) Wake up time (minutes)
(16) DHT11 data pin
flex_trit encoding (3 trits per byte) --->
[ ] Enable DEBUG in CClient

```

設定完畢後按<Yes>存檔退出。



您可以檢查 `sdkconfig` 檔案中的配置。

請確保您設定了訊息的接收位址（`CONFIG_MSG_RECEIVER`）：

```
CONFIG_WIFI_SSID="darwin43"
CONFIG_WIFI_PASSWORD="wifi pwd"
CONFIG_SNTP_SERVER="pool.ntp.org"
CONFIG_SNTP_TZ="CST-8"
CONFIG_IRI_NODE_URI="nodes.thetangle.org"
CONFIG_IRI_NODE_PORT=443
CONFIG_ENABLE_HTTPS=y
CONFIG_MSG_RECEIVER="HPAMGYVIPVHDEVSPXQFEKWVUQVG9GJBFWSFYT9SY9JSWNXFTV9MJFDF
MSJB9OKNVNTJPCGZNOVAXFEIS9"
CONFIG_IOTA_DEPTH=3
CONFIG_IOTA_MWM=14
CONFIG_WAKE_UP_TIME=5
CONFIG_DHT11_DATA_PIN=16
```

第三步：編譯二進制執行檔(Binary)

```
C:\Users\darwin\tangle_pigeon_dht11>idf.py build
```

第四步：燒錄(上傳)到 ESP32 設備

```
C:\Users\darwin\tangle_pigeon_dht11>idf.py flash
```

第五步：監視器(monitor)

```
C:\Users\darwin\tangle_pigeon_dht11>idf.py monitor
```

```
(4867) event: sta ip: 192.168.1.109, mask: 255.255.255.0, gw: 192.168.1.1
(4867) TanglePigeon: Connected to AP
(4867) TanglePigeon: IRI Node: nodes.thetangle.org, port: 443, HTTPS:True
(4877) TanglePigeon: Initializing SNTP: pool.ntp.org, Timezone: CST-8
(4887) TanglePigeon: Waiting for system time to be set... (1/10)
(6887) TanglePigeon: The current date/time is: Sat Jan 11 19:02:56 2020
(6887) TanglePigeon: Msg Address: HPAMGYVIPVHDEVSPXQFEKWVUQVG9GJBFWSFYT9SY9JSWNXFTV9MJFDFMSJB9OKNVNTJPCGZNOVAXFEIS9
(6887) TanglePigeon: Temperature:23.8°C humidity:66%
Power on reset
(30517) message-builder: transaction sent: OK
(31517) TanglePigeon: Enabling timer wakeup, 5 minutes
(31517) TanglePigeon: Entering deep sleep
```

Windows 系統則使用 `Ctrl+c`，退出 IDF 監視器。

打開瀏覽器輸入接收位址可看到會有定時訊息被傳送至 Tangle 網路中並被確認!

<https://thetangle.org/>

Transaction

VSHTRJJOAQWUGJE9NNAVIXYSCODSVRPIEACHCJNHBL9OJUGKXL9BZRYDOQCDCLVFMOGJWOUVCTUA9999 </> 
January 11, 2020 19:03:14 - 2 minutes and 2 seconds ago

Value	01	Confirmed	on 2020-01-11 at 19:04:23
Conversion	0 USD	Index in bundle	0 / 0
Tag	TANGLEPIGEON99999999999999999999	Weight magnitude	14

Address HPAMGYVIPVHDVESPXQFEKWVUQVG9GJBFWSFY9SY9JSWNXFTV9MJDFMSJB90KNVNTJPCGZNOVAXFEIS9YQNZCAQB

Bundle VRCVY9HVHECNUBWDXYEYWBYHBIRSDIWBGTGUYKKLDYNDUW9KQJDB9PSHEVIFFKETEXNBVKCWFZUYQ9EW

Nonce FFEARJFQQAUGNOIINUJBB99NUBM

Message "TanglePigeon": "v0.0.1"
○ Trytes "Device": "ESP32-WROOM32D"
○ Text "Trigger": "POWER_ON"
○ JSON "LocalTime": "2020-01-11 19:02:56"
"TimeZone": "CST-8"
"Data": "Temperature=23.8°C humidity=66%"

■ 部份原始程式(main.c):

```
#include "dht11_rmt.h" //匯入 dht11_rmt.h 標頭檔
```

...

```
char sensor_data[40]; //定義全域變數；傳送傳感器資料
```

...

在主程式 app_main() 中增加一段 dht11 傳感器監測資料

```
const int gpio_pin = CONFIG_DHT11_DATA_PIN;
```

```
const int rmt_channel = 0;
```

```
// Set up the RMT_RX module
```

```
vTaskDelay(1000 / portTICK_PERIOD_MS);
```

```
dht11_rmt_rx_init(gpio_pin, rmt_channel);
```

```
if(dht11_rmt_rx(gpio_pin, rmt_channel, &humidity, &temp_x10)) {
```

```
/* Clamp the inputs to ensure we don't blow the buffer out */
```

```
if(temp_x10 > 999) temp_x10 = 999;
```

```
if(temp_x10 < -999) temp_x10 = -999;
```

```
if(humidity > 1000) humidity = 100;
```

```
if(humidity < 0) humidity = 0;
```

```

/* Build the message */
const char template[] = "Temperature=%i.%iC humidity=%i%%\n";
sprintf(sensor_data, template, temp_x10/10, temp_x10%10, humidity);

} else {
    ESP_LOGI(TAG, "Sensor failure - retrying\n");
    sleep(5);
    if(dht11_rmt_rx(gpio_pin, rmt_channel, &humidity, &temp_x10)) {
        ESP_LOGI(TAG, "Temperature:%i.%iC humidity:%i%%\n", temp_x10/10,
temp_x10%10, humidity);

    } else {
        ESP_LOGI(TAG, "Sensor failure\n");
    }
}
}

```

■ 部份原始程式(message_builder.c):

```

extern char sensor_data[]; //定義外部變數；接收來自 main.c 程式傳送過來的傳感器資料
...
// data element
cJSON_AddItemToObject(json_root, "Data", cJSON_CreateString(sensor_data));

```

資料來源: IOTA 基金會的 SAM 提供

https://github.com/oopsmonk/tangle_pigeon_esp32