

IoT amb D1 mini (ESP8266) i codi Arduino



Jordi Orts

Gener 2019

Taula de continguts

Pròleg.....	5
Introducció.....	6
El kit D1 mini R1.....	8
Contingut del kit.....	8
Connexions i compatibilitat dels shields.....	14
Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit.....	15
1-button shield.....	19
Relay shield V2.0.0.....	20
RGB shield.....	22
Matrix led shield.....	24
Buzzer shield.....	26
DHT shield.....	28
OLED Shield.....	30
RTC shield.....	32
PIR shield.....	34
IR controller Shield.....	36
Part II. Connectats amb xarxa.....	38
Connexió com a estació.....	39
Un servidor web. Controlant el relé via WiFi.....	40
Connexió com a AP. Control WiFi d'un semàfor.....	43
AP + estació.....	46
Estació o AP?.....	47
ThingSpeak: enregistrar les nostres dades al núvol.....	48
Part III. Utilitzant components externs.....	50
Mòdul 6 LEDs.....	52
Mòdul LED RGB.....	54
Microservo 3,7g.....	55
Mòdul 4 polsadors.....	57
Mòdul potenciòmetre.....	59
Mòdul sensor temperatura DS18B20.....	62
Mòdul sensor de llum I2C BH1750FVI.....	66

Mòdul acceleròmetre + giroscopi I2C GY-521.....	67
Arduino Pro mini com a esclau I2C.....	70
Part IV. El sistema de fitxers SPIFFS. Utilització a un servidor web.	75
Afegint fitxers al nostre servidor web. La funció serveStatic()....	77
Enregistrar dades a la SD virtual i poder recuperar-les via WiFi.	
Generació de fitxers compatibles amb fulls de càlcul.....	80
Llegint dades de la SD virtual.....	83
FSManager, la navalla suïssa per a la gestió de fitxers a la SD virtual via WiFi.....	85
Part V. Altres shields D1 mini.....	92
Shield de connexions (JORTS).....	93
Shield CON1.....	95
Shield CON2.....	96
Dual base.....	97
Protoboard shield.....	98
WS2812B RGB Shield.....	99
Micro SD card shield.....	103
Datalogger shield.....	106
Ambient light Shield (BH1750).....	109
SHT30 Shield.....	110
Barometric Pressure Shield.....	112
TFT 1.4 Shield.....	114
TFT 2.4 Touch Shield.....	119
TFT I2C Connector Shield.....	122
Motor shield.....	123
DS18B20 shield.....	124
Battery Shield.....	125
DC Power Shield.....	126
Part VI. HTML5, javaScript i AJAX.....	127
HTML5 input type='range'. Creació de sliders.....	128
HTML5 input type='color'	131
Incloure eines HTML5 a la SD virtual.....	133
Utilitzant AJAX.....	135
La potència de javascript. Gràfics HTML canvas.....	139
La llibreria gràfica gauge.js.....	148
Part VII. Projectes.....	154

IoT amb D1 mini (ESP8266) i codi Arduino

Part VIII. Prototips comercials.....	155
Part IX. Actualització WiFi del programa (OTA).....	156
Part X. Altres plaques basades en l'ESP8266.....	157
Els mòduls ESP12 V1 i V3. La placa NODEMCU Base v1.0....	158
Els mòduls ESP12 V2. La placa NODEMCU motor shield.....	159
Els mòduls ESP-01. Plaques compatibles.....	160
Part XI. Altres components externs interessants.....	161
Part XII. Wearables.....	162
Annexos.....	164
Annex : Instal·lació Arduino per a ESP8266.....	165
Annex : Taula de colors RGB.....	168
Annex: Taula de freqüències per a les notes musicals.....	170
Annex: Firmware de l'arduino pro mini.....	171
Annex : Afegint la funcionalitat de FSManager als nostres projectes.....	177

Pròleg

Pròleg

Introducció

Aquest llibre està destinat a tots aquells que, tenint un coneixement bàsic de programació en Arduino i el seu IDE, volen treballar amb els ESP8266 en aquest entorn.

Porto treballant amb l'ESP8266 des de març del 2016. Es a dir, gairebé 3 anys. Si bé a l'estiu d'aquell any vaig explorar intensament les plaques NodeMCU, i vaig quedar meravellat, especialment amb la utilització de l'arduino IDE per a aquestes plaques , vaig descobrir tot el potencial d'aquestes joies un parell de mesos després, amb el sistema modular D1 mini.

Aquest sistema reuneix tres avantatges importants:

- Es treballa en un entorn conegut, l'arduino IDE, de forma que els alumnes poden fer una transició suau des de la programació tradicional amb arduino. L'aparició en aquest entorn de la gestió de targetes externes amb la versió 1.6.4 de l'IDE va facilitar molt l'adaptació d'aquest.
- Aquesta adaptació de l'IDE d'arduino introduceix noves prestacions com la interfície WiFi integrada, la creació de SDs virtuals amb el sistema de fitxers SPIFFS o la generació de freqüències variables de sortida. Realment s'ha fet un gran treball en aquest sentit.
- El sistema de shields del D1 mini permet un disseny realment modular. Amb un arduino tradicional podem tenir problemes per treballar amb 2 shields i algun component extern. Amb el D1 mini podem treballar fàcilment amb 4 o 5 shields i afegir components externs. I a un preu molt raonable.

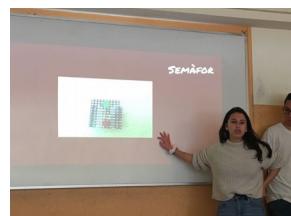
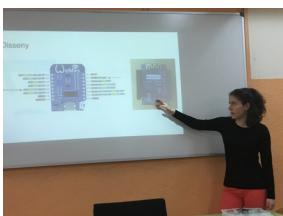
Introducció

Un any després els meus alumnes de 4t d'ESO ja utilitzaven aquests mòduls en els seus projectes de recerca, amb resultats sorprenents. Al curs següent vaig dissenyar la primera versió del kit que aquí us presentem, gràcies a l'aparició del primer distribuïdor a Espanya d'aquests mòduls. Amb aquest kit es va treballar a 1r de Batxillerat, amb un rotund èxit al Maker Faire BNC 2018, on els meus alumnes van ser premiats amb un *Maker of Merit*.

Al setembre de 2018 el meu nebot Javi Orts i jo vam dissenyar i fabricar el nostre primer shield pel D1 mini.

A la jornada «Engrescant el jovent cap a la tecno» que es va realitzar a la ESEIAAT-UPC de Terrassa l'1 de desembre de 2018 vaig tenir l'ocasió de presentar en un taller el kit, amb gran expectació i satisfacció per part dels professors assistents (va haver overbooking al meu taller!). I allà mateix va sortir la idea de crear un grup de treball de professors per aprofundir i difondre el tema.

En tres anys vaig fer innombrables proves, vaig tutoritzar diversos PREs, TRs, TFM, projectes al taller de Batxillerat ... Aprendent cada vegada més amb els meus alumnes¹.



Naturalment, ja em coneixeu: calia fer un llibre per transmetre tots els coneixements adquirits al llarg d'aquest temps, a la vegada que ordenava les meves idees. I aquí el teniu!

1 Podeu veure una selecció d'aquests treballs a <https://github.com/jorts64/kit-D1-mini/tree/master/projects>

El kit D1 mini R1

Fruit dels resultats de l'anterior kit, que em va permetre reduir el temps de muntatge de prototips al taller considerablement, i amb la intenció de guanyar compatibilitat entre els mòduls i millorar les prestacions del kit, a l'octubre del 2018 vaig dissenyar una revisió d'aquest on no em vaig limitar als productes disponibles al distribuïdor espanyol.

Amb aquest nou conjunt de materials és poden fer fàcilment multitud de prototips comercials.

Contingut del kit

Tripler base amb D1 mini, terminals mascle a la tercera columna per I/O i alimentació (GND-3,3V-5V), connector 3 pins sortida de servo a 5V (connectat a D6) i connexions I2C



OLED shield V2.1.0

Terminals M



El kit D1 mini R1

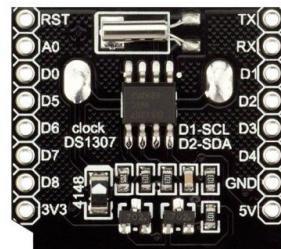
Matrix led shield

Terminals M



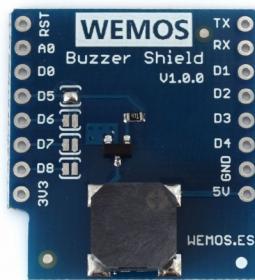
RTC shield

Terminals M-F



Buzzer shield

Terminals M-F



IoT amb D1 mini (ESP8266) i codi Arduino

1 button shield

Terminals M



RGB shield

Terminals M

Reconfigurat al pin D8



Relay shield V2.0.0

Terminals M

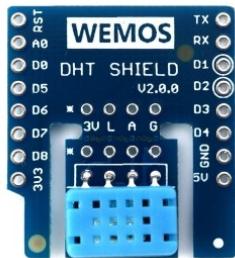
Reconfigurat al pin D7



El kit D1 mini R1

DHT shield

Terminals M-F



IR controller Shield

Terminals M



PIR shield

Terminals M

Reconfigurat al pin D6



mòdul LED RGB

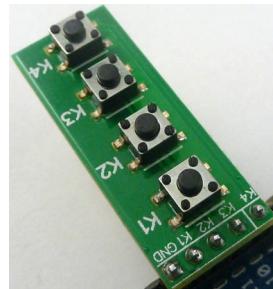


IoT amb D1 mini (ESP8266) i codi Arduino

mòdul 6 LEDs



mòdul 4 polsadors



mòdul sensor de llum I2C
BH1750FVI



mòdul acceleròmetre + giroscopi
I2C GY-521

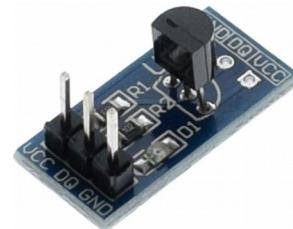


El kit D1 mini R1

mòdul potenciòmetre



mòdul sensor temperatura
DS18B20



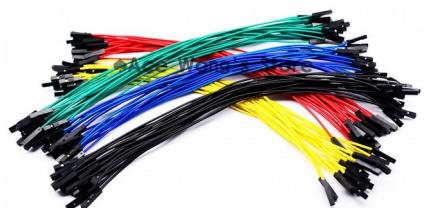
microservo 3,7g



Arduino Pro mini com a esclau
I2C



10 cables Dupont F-F



Connexions i compatibilitat dels shields

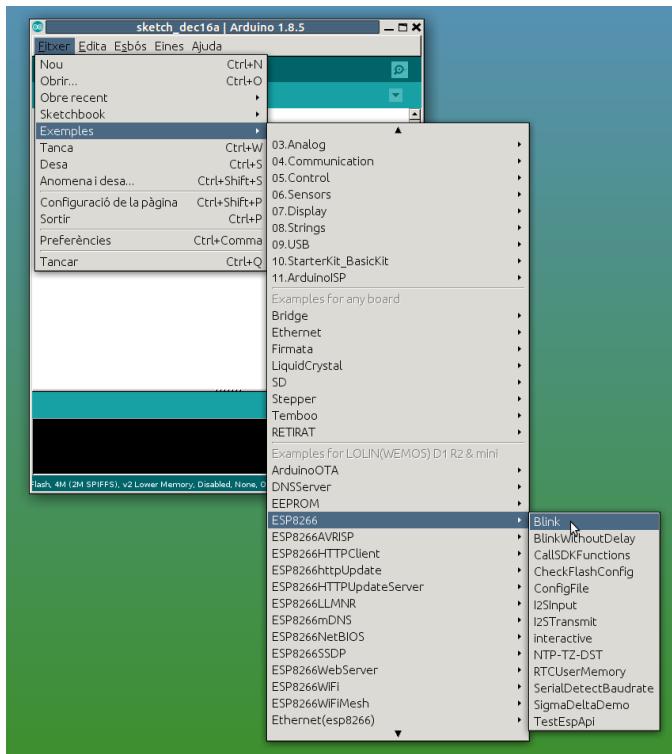
Shield	D0	D1	D2	D3	D4	D5	D6	D7	D8	A0
OLED		I2C	I2C	x	x					
Matrix led						x		x		
RTC		I2C	I2C							
Buzzer						x				
1 button				x						
RGB									x	
Relay								x		
DHT		I2C	I2C		x					
IR				x	x					
PIR							x			

Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit.

Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit.

És típic quan un comença a treballar amb Arduino carregar l'exemple Blink, que fa pampallugues al led integrat en la placa Arduino.

Nosaltres podem començar igual amb el D1 mini. Però haurem de fer servir l'exemple creat especialment pel ESP8266:



IoT amb D1 mini (ESP8266) i codi Arduino

The screenshot shows the Arduino IDE interface with the title bar "Blink | Arduino 1.8.5". The menu bar includes "Fitxer", "Edita", "Esbós", "Eines", and "Ajuda". Below the menu is a toolbar with icons for file operations like Open, Save, and Print. A dropdown menu shows "Blink" selected. The code editor contains the "Blink" sketch for ESP8266. The code is as follows:

```
/*
ESP8266 Blink by Simon Peter
Blink the blue LED on the ESP-01 module
This example code is in the public domain

The blue LED on the ESP-01 module is connected to GPIO1
(which is also the TXD pin; so we cannot use Serial.print() at the same time)

Note that this sketch uses LED_BUILTIN to find the pin with the internal LED
*/

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);      // Initialize the LED_BUILTIN pin as an output
}

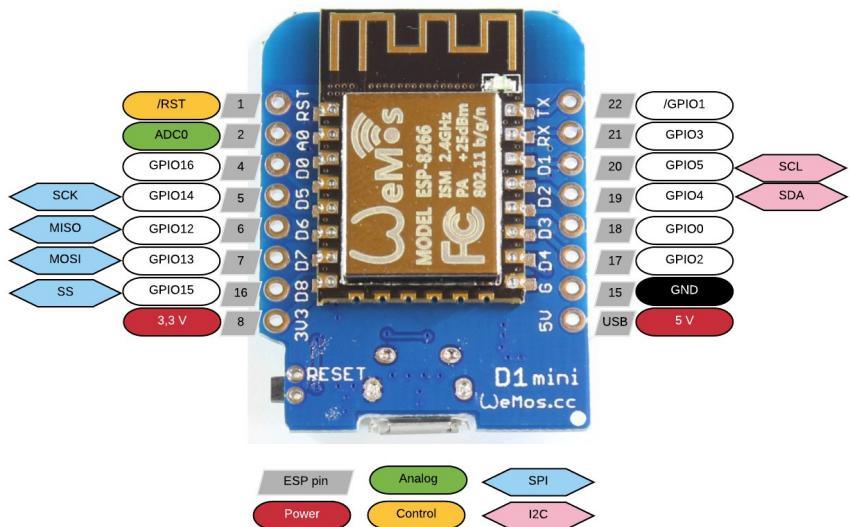
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, LOW);    // Turn the LED on (Note that LOW is the voltage level
  // but actually the LED is on; this is because
  // it is active low on the ESP-01)
  delay(1000);                   // Wait for a second
  digitalWrite(LED_BUILTIN, HIGH);   // Turn the LED off by making the voltage HIGH
  delay(2000);                   // Wait for two seconds (to demonstrate the active low LED)
}
```

The status bar at the bottom indicates the board is "LOLIN(WEMOS) D1 R2 & mini, 80 MHz, Flash, 4M (2M SPIFFS), v2 Lower Memory, Disabled, None, Only Sketch, 921600 en /dev/ttyUSB0".

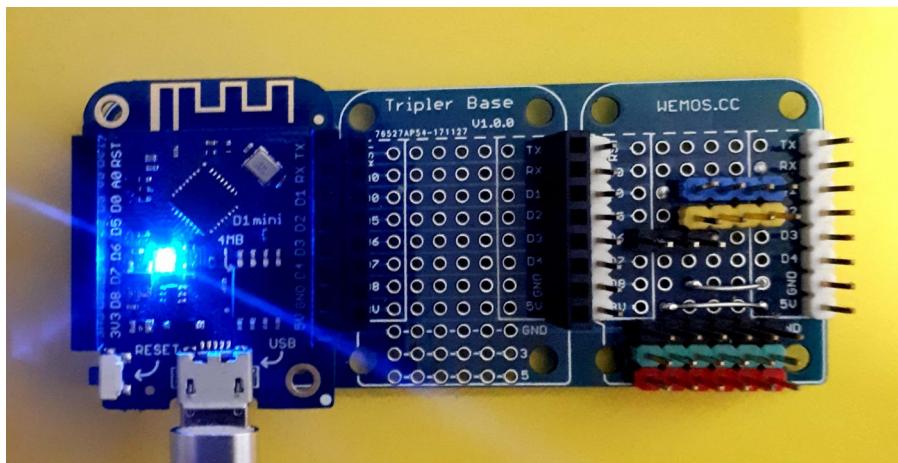
Fixeu-vos com al programa s'utilitza el pin *LED_BUILTIN*. És un sinònim de *D4*, nom del pin serigrafiat a la placa del D1 mini, que en realitat és el pin *GPIO2*. És a dir, si substituïm *LED_BUILTIN* al programa anterior per *D4* o *2*, el programa farà exactament el mateix.

Un error típic al programar el D1 mini és posar només el número de la pota serigrafiada. No és el mateix. Cal posar la *D* abans, ja que el número de pota serigrafiat no coincideix amb la pota física del xip:

Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit.



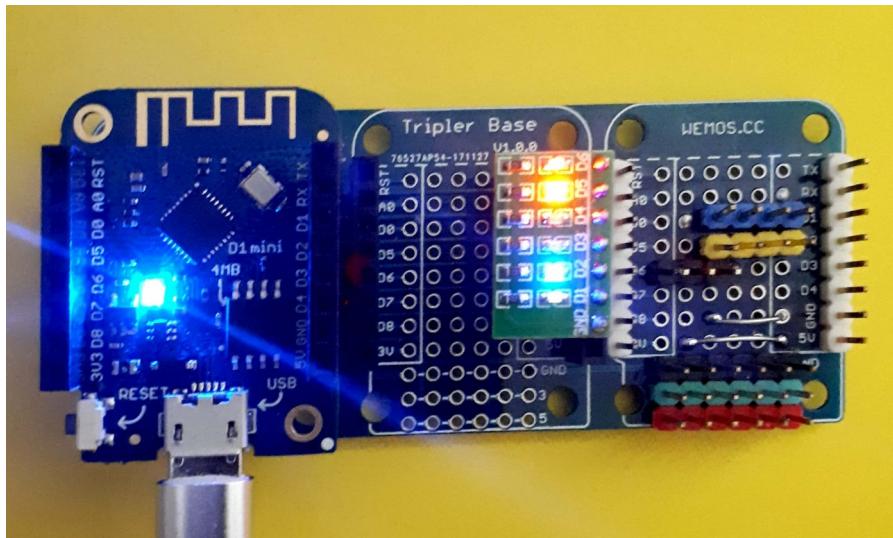
Ja podeu enviar el programa al D1 mini. El led blau integrat farà pampallugues.



IoT amb D1 mini (ESP8266) i codi Arduino

Per altra banda, si us fixeu bé, aquest led reacciona a l'inrevés del que un espera. Quan posem un *LOW*, el led s'encén (durant 1s), i quan posem un *HIGH* s'apaga (durant 2s).

Aquest comportament només el trobem en aquest led blau, no afecta als leds connectats exteriorment. Ho podeu comprovar connectant un led extern entre la pata *D4* i *GND*.



Fixeu-vos com he col·locat el mòdul de leds, amb *GND* coincidint amb el *GND* de la base triple. Veureu com quan s'apaga el led integrat s'encén el led blanc, i a l'inrevés.

Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit.

1-button shield

Aquest shield ens ofereix un polsador connectat a la porta *D3*. Quan premem el polsador, aquest connecta la entrada a *GND*. Es a dir, obtenim un *HIGH* si no prenem el polsador, i un *LOW* si ho fem.

Podeu comprovar el seu funcionament amb el meu exemple *button*:



```
const int buttonPin = D3;
const int ledPin = BUILTIN_LED;

void setup() {
    pinMode(buttonPin, INPUT);
    pinMode(ledPin, OUTPUT);
}

void loop() {
    if (digitalRead(buttonPin) == HIGH) {
        digitalWrite(ledPin, HIGH); // button released, LED off
    } else {
        digitalWrite(ledPin, LOW); // button pressed, LED on
    }
}
```

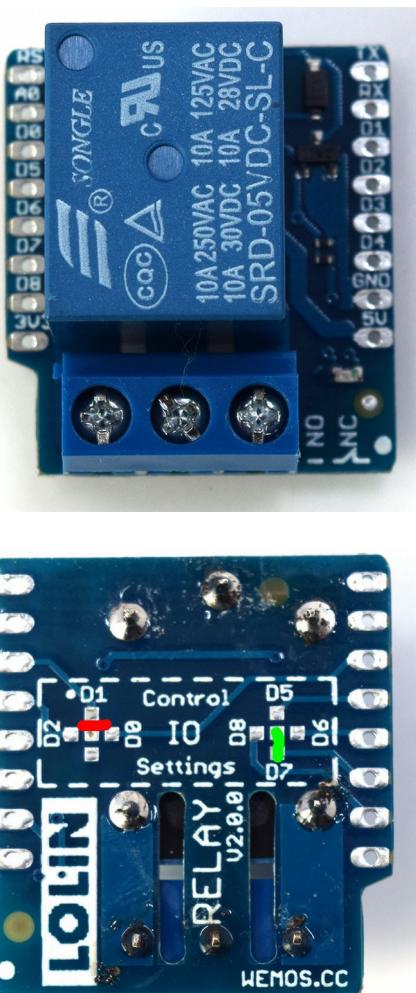


Relay shield V2.0.0

Aquest mòdul porta un relé (nosaltres el farem servir connectat a D7) amb el qual podem controlar fins i tot circuits a CA de 220V. Fixeu-vos que els límits de càrrega depenen de si aquesta la connectem a NO o a NC:

- NO:
5A(250VAC/30VDC),
10A(125VAC)
MAX:1250VA/150W
- NC:
3A(250VAC/30VDC)
MAX:750VA/90W

Podeu comprovar el seu funcionament amb el meu exemple *relay*. No oblideu canviar el *relayPin* a *D7*, tal com es mostra al llistat següent:



Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit.

```
//const int relayPin = D1; // comment for kit D1 mini R1
const int relayPin = D7; // uncomment for kit D1 mini R1
const long interval = 2000; // pause for two seconds

void setup() {
    pinMode(relayPin, OUTPUT);
}

void loop() {
    digitalWrite(relayPin, HIGH); // turn on relay
    delay(interval);           // pause
    digitalWrite(relayPin, LOW); // turn off relay
    delay(interval);           // pause
}
```

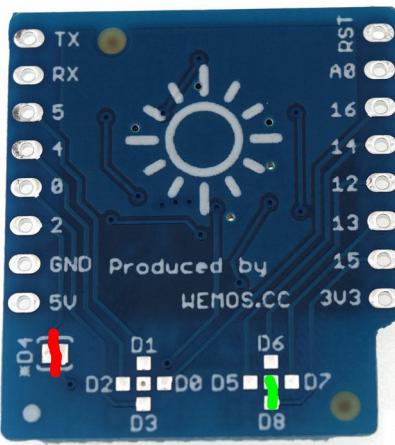


RGB shield

Aquest mòdul porta 7 leds RGB WS2812B, que fan servir un protocol d'un sol fil (per defecte connectat a D4). Nosaltres el fem servir connectat a D8.

La forma més senzilla d'utilitzar aquest shield és amb la llibreria Adafruit NeoPixel, que podem instal·lar des del gestor de llibreries de l'Arduino IDE.

Utilitzant la llibreria Adafruit NeoPixel, que podem instal·lar des del gestor de llibreries, podem provar aquest senzill exemple:

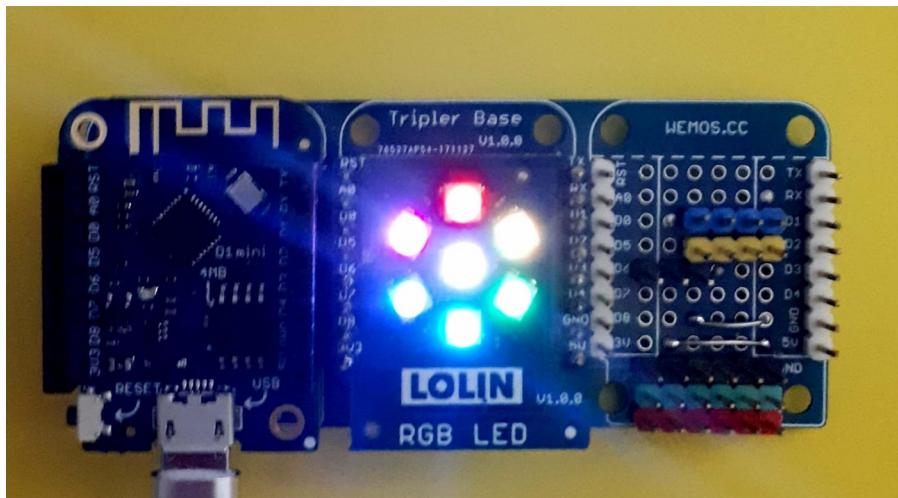


Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit.

```
#include <Adafruit_NeoPixel.h>
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(7, D8, NEO_GRB + NEO_KHZ800);

void setup() {
    pixels.begin(); // This initializes the NeoPixel library.
}

void loop() {
    for (int i=0; i<=255;i++){
        pixels.setPixelColor(0, pixels.Color(i,i,i )); // blanc.
        pixels.setPixelColor(1, pixels.Color(i,0,0 )); // vermell.
        pixels.setPixelColor(2, pixels.Color(i,i,0 )); // groc.
        pixels.setPixelColor(3, pixels.Color(0,i,0 )); // verd.
        pixels.setPixelColor(4, pixels.Color(0,i,i )); // cyan.
        pixels.setPixelColor(5, pixels.Color(0,0,i )); // blau.
        pixels.setPixelColor(6, pixels.Color(i,0,i )); // magenta.
        pixels.show(); // This sends the updated pixel color to the hardware.
        delay(10); // Delay for a period of time (in milliseconds).
    }
}
```

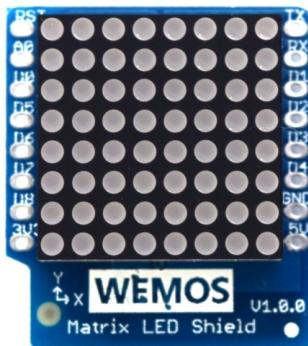


Matrix led shield

Matriu de 8x8 leds vermells amb 8 nivells d'intensitat. Utilitza D5 (CLK) i D7 (DIN)

Amb la llibreria *WEMOS Matrix LED Shield* és relativament senzill utilitzar aquest shield.

Amb la funció `dot(x,y,valor)` fixem els punts encesos (`valor=1`) i apagats (`valor=0`). Una vegada fixats, amb la funció `display()` s'actualitza la matriu de leds.



Gràcies a un editor online (<https://xantorohara.github.io/led-matrix-editor/>) podem fer animacions i definir pantalles:

The screenshot shows the LED Matrix Editor interface. At the top, there's a browser header with tabs like 'Sofata d'ent.', 'Classes', 'Institut Priv.', 'Recursos-Tri', 'Public-Drop', and 'LED Matrix'. Below the header, there's a toolbar with color swatches and a 'Select LED+color' dropdown. The main area has three main sections: a 'Library' section with options like 'Set No1: Digits / Letters / Signs', 'Set No2: Digits / Letters / Signs / Others', and 'Set No3: Digits / Icons'; a 'Free Writing Tool' section with a text input field and a 'Grammarly' button; and an 'Arduino/C code' editor with a scrollable code block containing hex values. At the bottom, there's a preview area showing a 8x8 grid of red dots and a 'HEX' section with a scrollable list of hex values.

Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit.

Al meu exemple *matrix_LED_animation* hem fet servir aquest editor, i us pot servir com a plantilla. Només cal canviar l'array *IMAGES[]*:

```
#include <WEMOS_Matrix_LED.h>
MLED mled(5); //set intensity=5

const uint64_t IMAGES[] = {
    0x000080cfeffffe0c08,
    0x00004067f7f7f0604,
    0x00002033f3f3f0302,
    0x00001011f1f1f0101,
    0x00000000f0f0f0000,
    0x00000000707070000,
    0x00000000303030000,
    0x00000000101010000,
    0x000000000000000000
};

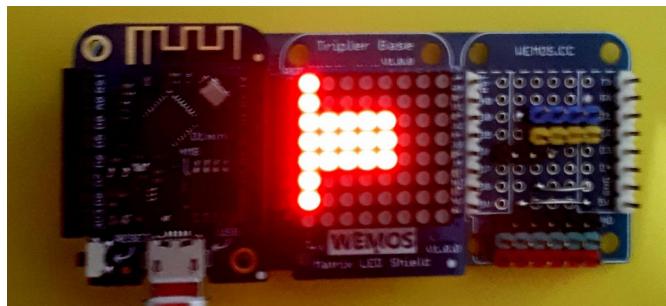
const int IMAGES_LEN = sizeof(IMAGES)/8;

void setup() {
}

void displayImage(uint64_t image) {
    for (int i = 0; i < 8; i++) {
        byte row = (image >> i * 8) & 0xFF;
        for (int j = 0; j < 8; j++) {
            mled.dot(j, 7-i, bitRead(row, j));
        }
    }
    mled.display();
}

int i = 0;

void loop() {
    displayImage(IMAGES[i]);
    if (++i >= IMAGES_LEN ) {
        i = 0;
    }
    delay(100);
}
```

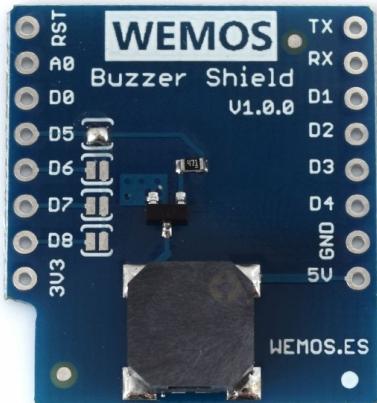


Buzzer shield

Afegeix un petit buzzer al nostre sistema.

Nosaltres treballem amb la connexió amb el pin *D5* (per defecte).

La implementació per al IDE Arduino del ESP8266 introduceix una funció molt útil per a aquest shield, *analogWriteFreq(freq)*, que permet fixar la freqüència de la sortida i ens permet fer melodies.



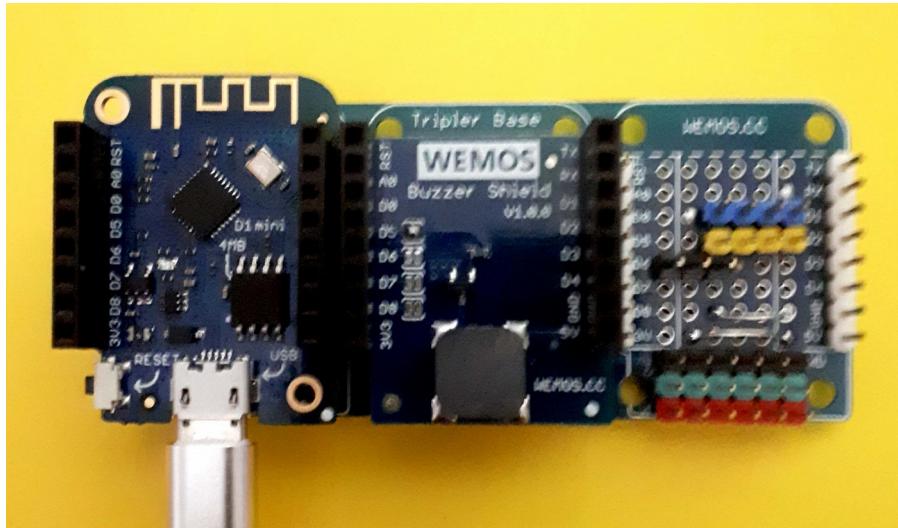
Al meu exemple *sirena* podeu veure com utilitzar aquesta funció:

```
int buzzer=D5; //Buzzer control port, default D5

void setup() {
  pinMode(buzzer, OUTPUT);
  digitalWrite(buzzer, LOW);
}

void loop() {
  for (int i=0;i<5;i++){
    analogWriteFreq(988);      // 988 Hz, nota Do
    analogWrite(buzzer, 512);   // ona simètrica
    delay(700);                // 0,7 s durada
    analogWrite(buzzer, 0);     // silenci
    delay(50);                 // 0,05 s
    analogWriteFreq(587);       // 587 Hz, nota Si
    analogWrite(buzzer, 512);
    delay(700);
    analogWrite(buzzer, 0);
    delay(50);
  }
  pinMode(buzzer, OUTPUT);
  digitalWrite(buzzer, LOW);    // silenci
  delay(5000);
}
```

Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit.



Podeu trobar informació molt útil a l'Annex: Taula de freqüències per a les notes musicals.

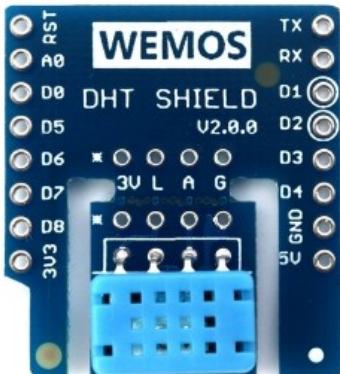
DHT shield

Sensor de temperatura i humitat
DHT11 amb protocol I2C (potes D1 i D2).

Temperatura: -20~60°C ($\pm 0.5^\circ\text{C}$)

Humitat: 20-95%RH ($\pm 5\%$ RH)

És molt fàcil el seu ús amb la *WEMOS_DHT12_Arduino_Library*.
Al meu exemple *DHT* es llegeix la temperatura i la humitat i es mostren pel canal sèrie



```
#include <WEMOS_DHT12.h>
DHT12 dht12;

void setup() {
  Serial.begin(115200);
}

void loop() {
  if(dht12.get()==0){
    Serial.print("Temperature in Celsius : ");
    Serial.println(dht12.cTemp);
    Serial.print("Temperature in Fahrenheit : ");
    Serial.println(dht12.fTemp);
    Serial.print("Relative Humidity : ");
    Serial.println(dht12.humidity);
    Serial.println();
  }
  delay(1000);
}
```

Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit.

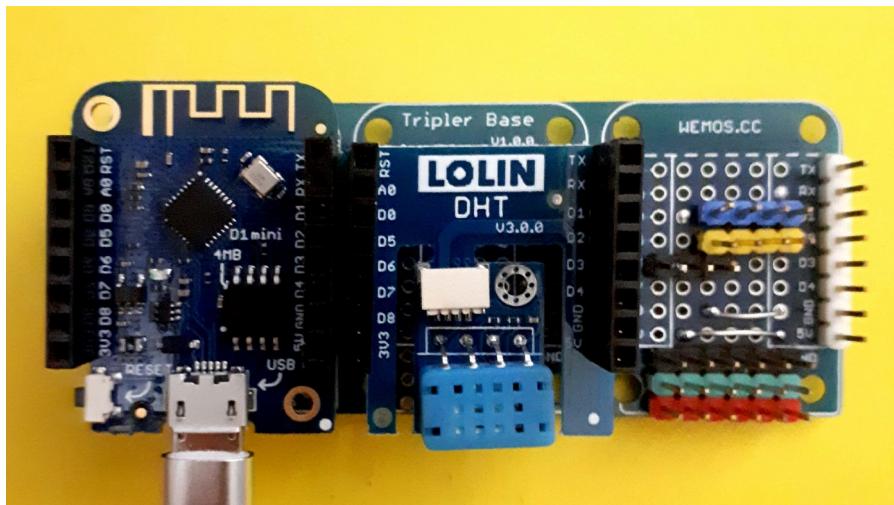
```
/dev/ttyUSB1
Temperature in Fahrenheit : 80.06
Relative Humidity : 33.20

Temperature in Celsius : 26.70
Temperature in Fahrenheit : 80.06
Relative Humidity : 33.20

Temperature in Celsius : 26.80
Temperature in Fahrenheit : 80.24
Relative Humidity : 33.20

Temperature in Celsius : 26.70
Temperature in Fahrenheit : 80.06
Relative Humidity : 33.10

 Desplaçament automàtic Sense salts de línia ▾ 115200 baud ▾ Clear output
```



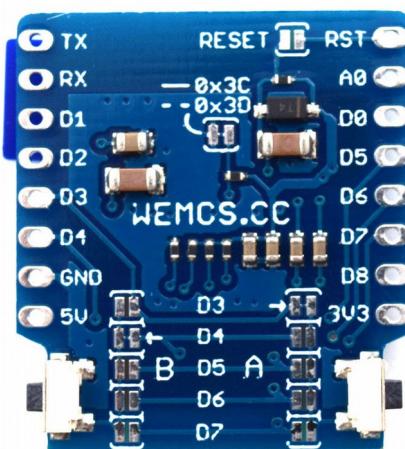
OLED Shield

Aquest mòdul porta un display gràfic OLED de 64x48 pixels mitjançant el protocol I2C (D1 SCL, D2 SDA).

Els nostres kits fan servir la versió 2.0.0, que inclou dos polsadors de connexió configurable (per defecte D3 i D4, que són els que fem servir).

Els millors resultats els he aconseguit amb la llibreria sparkfun/Micro_OLED_Breakout, encara que ha calgut un exemple modificat (*OLED*) configurat per a I2C i amb alguns canvis per a evitar problemes amb el WatchDog.

Tenim moltíssimes funcions, i inclouen funcions gràfiques molt avançades. Les més importants per mostrar un text a pantalla son *begin()*, *clear()*, *setFontType()*, *setCursor()*, *print()* i *println()*. Recordeu fer sempre un *display()* al final, o no es mostrerà res per pantalla.



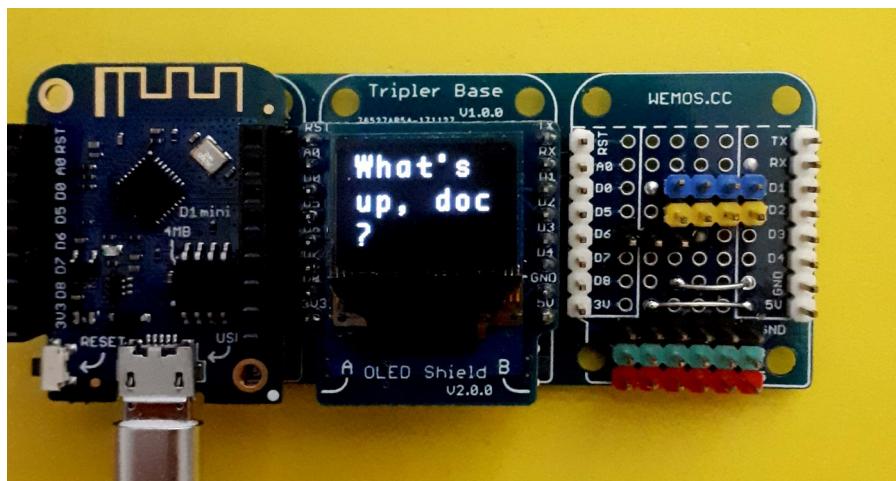
El meu exemple *OLED* inclou tot tipus de funcions. Però per començar crec que serà més pràctic el meu exemple *OLED_test*:

Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit.

```
#include <Wire.h> // Include Wire if you're using I2C
#include <SFE_MicroOLED.h> // Include the SFE_MicroOLED library
#define PIN_RESET 255 // Connect RST to pin 9
#define DC_JUMPER 0
MicroOLED oled(PIN_RESET, DC_JUMPER); // I2C declaration

void setup()
{
    oled.begin(); // Initialize the OLED
    oled.clear(ALL); // Clear the display's internal memory
    oled.display(); // Display what's in the buffer (splashscreen)
    delay(1000); // Delay 1000 ms
    oled.clear(PAGE); // Clear the buffer.
}

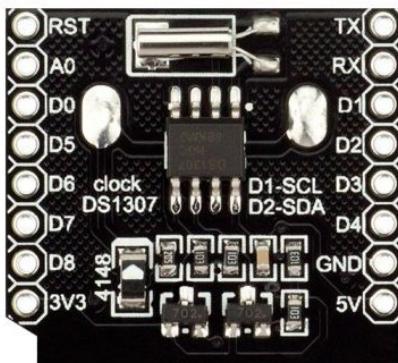
void loop()
{
    oled.clear(PAGE);
    oled.setFontType(1);
    oled.setCursor(0,0);
    oled.println("What's up, doc?");
    oled.display();
    delay(1500);
}
```



RTC shield

Aquest shield porta un rellotge en temps real (Real Time Clock) amb connexió I2C.

Aquest és un dels pocs shields que no fabrica Wemos / Lolin, si no RobotDyn, fabricant que també comercialitza una versió datalogger que, a més del rellotge I2C, inclou un lector microSD.



La llibreria *RTCLib* (Adafruit) incorporada al Library manager funciona perfectament amb l'ESP8266, com podeu comprovar amb el meu exemple RTC_test.



Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit.

```
#include <Wire.h>
#include "RTClib.h"
RTC_DS1307 RTC;

void setup () {
    Serial.begin(9600);
    Wire.begin();
    RTC.begin();
    // Check to see if the RTC is keeping time. If it is, load the time from your computer.
    if (! RTC.isrunning()) {
        Serial.println("RTC is NOT running!");
        // This will reflect the time that your sketch was compiled
        RTC.adjust(DateTime(__DATE__, __TIME__));
    }
}

void loop () {
    DateTime now = RTC.now();
    Serial.print(now.month(), DEC);
    Serial.print('/');
    Serial.print(now.day(), DEC);
    Serial.print('/');
    Serial.print(now.year(), DEC);
    Serial.print(' ');
    Serial.print(now.hour(), DEC);
    Serial.print(':');
    Serial.print(now.minute(), DEC);
    Serial.print(':');
    Serial.print(now.second(), DEC);
    Serial.println();
    delay(1000);
}
```



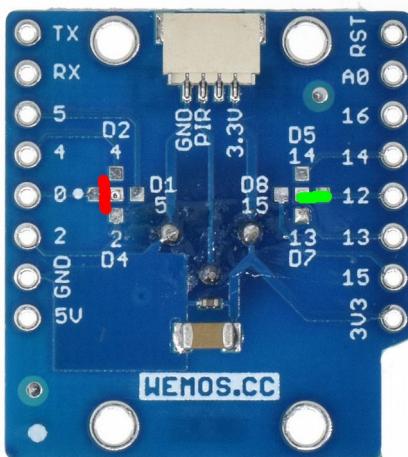
PIR shield

Aquest shield ens dona un detector de presència per calor (PIR). Quan detecta una persona, dona una sortida HIGH.

Nosaltres el tenim connectat a D6.

A tots els efectes el podem utilitzar com un interruptor d'entrada connectat a la pista D6, però heu de pensar que una vegada que es dispara, triga uns 3 s des de que no detecta gent a tornar la sortida a LOW.

Podeu provar el meu exemple *PIR_test* per comprovar el seu funcionament.

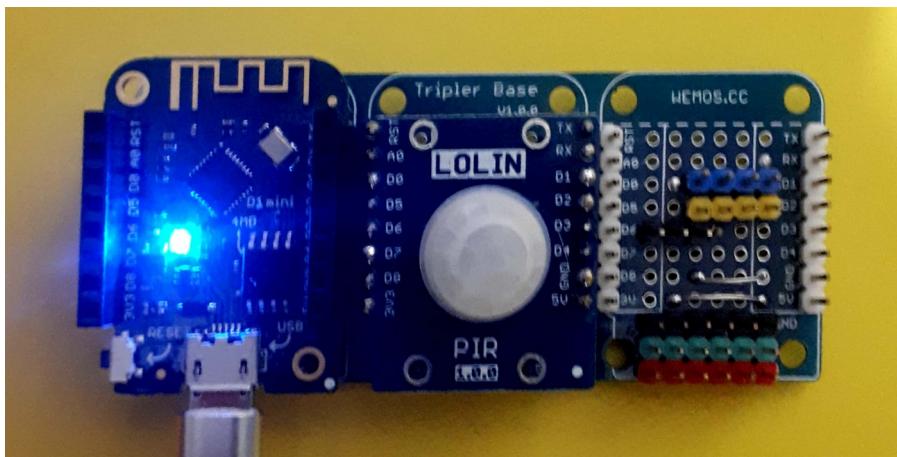


Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit.

```
const int buttonPin = D6;
const int ledPin = BUILTIN_LED;

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH); // LED off
}

void loop() {
  if (digitalRead(buttonPin) == HIGH) {
    digitalWrite(ledPin, LOW); // people detected, LED on
  } else {
    digitalWrite(ledPin, HIGH); // LED off
  }
}
```



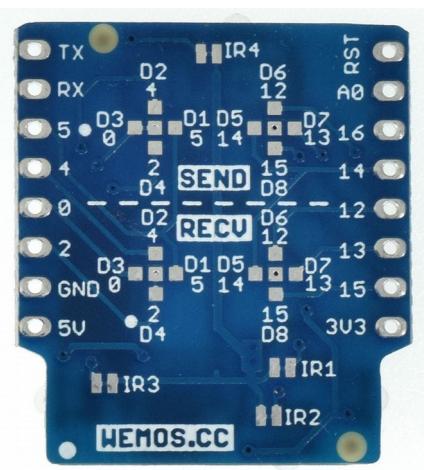
IR controller Shield

Aquest shield incorpora 4 emissors i 1 receptor de codis IR. Per defecte els emissors estan connectats a D3 i el receptor a D4. Nosaltres mantindrem aquestes connexions.

Amb la llibreria *IRremote ESP8266 Library* (es pot instal·lar des del gestor de llibreries) tenim diferents exemples per rebre i enviar codis IR.

Us deixo dos exemples:

- *IRrx*, que rep i decodifica codis IR
- *IRtx*, que envia codis IR



Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit.

```
#include <IRremoteESP8266.h>
#include <IRrecv.h>
#include <IRutils.h>

IRrecv irrecv(D4);
decode_results results;

void setup() {
  Serial.begin(115200);
  irrecv.enableIRIn(); // Start the receiver
}

void loop() {
  if (irrecv.decode(&results)) {
    // print() & println() can't handle printing long longs. (uint64_t)
    serialPrintInt64(results.value, HEX);
    Serial.println("");
    irrecv.resume(); // Receive the next value
  }
  delay(100);
}
```



```
#include <IRremoteESP8266.h>
#include <IRsend.h>
IRsend irsend(D3);

void setup() {
  irsend.begin();
}

void loop() {
  irsend.sendSony(0xa90, 12, 2);
  delay(2000);
}
```

Part II. Connectats amb xarxa

Fins ara no hem utilitzat la potència WiFi del nostre ESP8266. Ha arribat el moment de fer-la servir.

```
#include <ESP8266WiFi.h>
```

Aquesta senzilla línia dona sentit al títol d'aquest llibre. IoT. Internet of Things. Internet de les coses. I com podem dir IoT sense connexió a Internet? La veritat, n'estic fart de tallers, cursos, llibres i webs d'Arduino que s'anuncien com a IoT i la connexió a Internet no apareix enllloc.

Aquest, en canvi, és un llibre orientat a IoT. I amb aquest capítol veureu que va de debò!

Connexió com a estació

Per connectar-nos a una xarxa existent, com fariem amb qualsevol ordinador, hem de fer servir la funció

```
WiFi.begin("network-name", "pass-to-network");
```

on cal canviar *network-name* pel nom de la nostre xarxa i *pass-to-network* per la contrasenya, que deixarem en blanc si es tracta d'una xarxa oberta.

Cal comprovar que hem aconseguit connectar-nos amb la funció WiFi.status(). Veiem un exemple:

```
#include <ESP8266WiFi.h>

void setup()
{
    Serial.begin(115200);
    Serial.println();

    WiFi.begin("network-name", "pass-to-network");

    Serial.print("Connecting");
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print(".");
    }
    Serial.println();

    Serial.print("Connected, IP address: ");
    Serial.println(WiFi.localIP());
}

void loop() {}
```

Un servidor web. Controlant el relé via WiFi

La majoria de vegades el que volem es connectar-nos al nostre ESP8266 i recollir dades o executar ordres. La millor forma és amb un servidor web. Per això caldrà afegir al principi

```
#include <ESP8266WebServer.h>
```

i definir un servidor al port 80:

```
ESP8266WebServer server(80);
```

Caldrà definir, típicament al *setup()* com reacciona el servidor a les diferents pàgines adjudicant a cada pàgina una funció amb

```
server.on("PAGINA", FUNCIO);
```

i que fer en cas de no trobar la pàgina amb

```
server.onNotFound(handleNotFound);
```

Una vegada definides les pàgines i la funció per quan es demani una no contemplada cal engegar el servidor amb

```
server.begin();
```

i estar pendent de les peticions entrants, típicament al *loop()* amb

```
server.handleClient();
```

A les funcions definides podem generar una resposta amb

```
server.send(codi-resposta, "text/plain", message);
```

Podem veure tot això funcionant al meu exemple *RelayRemote*:



Part II. Connectats amb xarxa

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

const char* ssid = "la teva xarxa aqui";
const char* password = "contrasenya de xarxa aqui";

ESP8266WebServer server(80);
int estat;

void handleRoot() {
    server.send(200, "text/plain", "hello from esp8266!");
}

void engegar() {
    digitalWrite(D1, true);
    estat=1;
    server.send(200, "text/plain", "ON");
}

void apagar() {
    estat=0;
    server.send(200, "text/plain", "OFF");
    digitalWrite(D1, false);
}

void canviar() {
    if(estat==0){
        digitalWrite(D1, true);
        estat=1;
    }
    else{
        digitalWrite(D1, false);
        estat=0;
    }
    server.send(200, "text/plain", "CHANGED");
}

void handleNotFound(){
    String message = "File Not Found\n\n";
    message += "URI: ";
    message += server.uri();
    message += "\nMethod: ";
    message += (server.method() == HTTP_GET)?"GET":"POST";
    message += "\nArguments: ";message += server.args();
    message += "\n";
    for (uint8_t i=0; i<server.args(); i++){
        message += " " + server.argName(i) + ":" + server.arg(i) + "\n";
    }
    server.send(404, "text/plain", message);
}

void setup(void){
    pinMode(D1, OUTPUT);
    Serial.begin(115200);
    WiFi.begin(ssid, password);
    Serial.println("");
    // Wait for connection
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
}
```

IoT amb D1 mini (ESP8266) i codi Arduino

```
}

Serial.println("");
Serial.print("Connected to ");
Serial.println(ssid);
Serial.print("IP address: ");
Serial.println(WiFi.localIP());

server.on("/", handleRoot);
server.on("/on", engegar);
server.on("/off", apagar);
server.on("/change", canviar);
server.on("/inline", [](){
    server.send(200, "text/plain", "this works as well");
});
server.onNotFound(handleNotFound);

server.begin();
Serial.println("HTTP server started");estat=0;
digitalWrite(D1, false);
}

void loop(void){
    server.handleClient();
}
```

Connexió com a AP. Control WiFi d'un semàfor

El nostre ESP8266 pot crear una xarxa pròpia, a la que podem connectar-nos amb el nostre ordinador o mòbil.

En lloc del *WiFi.begin()* que fèiem servir la connexió com a estació, farem servir

```
WiFi.mode(WIFI_AP);  
WiFi.softAP(ssid, password);
```

i ja el podrem fer servir.

Veiem com funciona amb el meu exemple *semafor_RGB_Wifi.ino*:



```
#include <ESP8266WiFi.h>  
#include <WiFiClient.h>  
#include <ESP8266WebServer.h>  
  
const char *ssid = "Semaforo";  
const char *password = "robotica";  
  
ESP8266WebServer server ( 80 );  
  
#include <Adafruit_NeoPixel.h>  
  
#define PIN D8
```

IoT amb D1 mini (ESP8266) i codi Arduino

```
#define LED_NUM 7

Adafruit_NeoPixel leds = Adafruit_NeoPixel(LED_NUM, PIN, NEO_GRB + NEO_KHZ800);

int tr=2000;
int ty=500;
int tg=1500;

String mensaje = "";

-----CODO HTML PAGINA DE CONFIGURACION-----
String pagina = "<!DOCTYPE html>
"<html>
"<head>
<title>Control de temps semafor</title>
"<meta charset='UTF-8'>
"<meta name='viewport' content='width=device-width' />
"</head>
"<body>
"<table style='width:100%><form action='canviar' method='get'>
"<tr><td>Temps vermell:</td><td><input type='range' name='fr' min='1' max='60'
value='3' step='1'></td></tr>
"<tr><td>Temps groc:</td><td><input type='range' name='fy' min='1' max='60'
value='1' step='1'></td></tr>
"<tr><td>Temps verd:</td><td><input type='range' name='fg' min='1' max='60'
value='2' step='1'></td></tr>
"<tr><td></td><td><input type='submit' value='CANVIAR' /></td></tr>
"</form></table>
"</body>
"</html>";

void espera(int temps) {
    unsigned long ara = millis();
    unsigned long seguent = ara + temps;
    delay(1); //refresh watchdog
    while (millis() < seguent){
        server.handleClient();
    }
}

void paginacanvi() {
    server.send(200, "text/html", pagina);
}

void canviar_temps() {
    tr=server.arg("fr").toInt()*1000;
    ty=server.arg("fy").toInt()*1000;
    tg=server.arg("fg").toInt()*1000;
    paginacanvi();
}

void led_set(uint8 R, uint8 G, uint8 B) {
    for (int i = 0; i < LED_NUM; i++) {
        leds.setPixelColor(i, leds.Color(R, G, B));
        leds.show();
        delay(50);
    }
}

void red_set() {
    leds.setPixelColor(1, leds.Color(100, 0, 0));
    leds.setPixelColor(0, leds.Color(10, 10, 0));
    leds.setPixelColor(4, leds.Color(0, 10, 0));
```

Part II. Connectats amb xarxa

```
leds.show();
espera(tr);
}

void yellow_set() {
    leds.setPixelColor(1, leds.Color(10, 0, 0));
    leds.setPixelColor(0, leds.Color(50, 50, 0));
    leds.setPixelColor(4, leds.Color(0, 10, 0));
    leds.show();
    espera(ty);
}

void green_set() {
    leds.setPixelColor(1, leds.Color(10, 0, 0));
    leds.setPixelColor(0, leds.Color(10, 10, 0));
    leds.setPixelColor(4, leds.Color(0, 100, 0));
    leds.show();
    espera(tg);
}

void setup() {
    WiFi.mode(WIFI_AP);
    WiFi.softAP(ssid, password);
    server.on("/", paginacanvi);
    server.on("/canviar", canviar_temps);
    server.begin();
    leds.begin();
    led_set(0, 0, 0);
}

void loop() {
    green_set();
    yellow_set();
    red_set();
}
```

Fixeu-vos que al programa en lloc de *delay()* faig servir *espera()*, una funció que he definit i que mentre espera el temps marcat en ms està pendent de les peticions web amb *handleClient()*.

AP + estació

És possible generar una xarxa com a AP i, a la vegada, estar connectat a una altra xarxa com a estació:

```
WiFi.mode(WIFI_AP_STA);  
WiFi.softAP(ssid1, password1);  
WiFi.begin(ssid2, password2);
```

Encara que són dues xarxes diferents, farà servir un únic canal: el que utilitza la xarxa a la que es connecta com a estació.

Una aplicació interessant d'aquesta doble connexió és la creació de xarxes de malla. Saida Sillo va fer un TR que vaig tutoritzar amb el títol «Xarxa detectora d'incendis forestals» on utilitzava aquestes xarxes per cobrir tot un bosc amb sensors i permetre enviar una alerta a internet. Cada sensor és, a la vegada, AP i estació dels seus veïns. Estació per transmetre l'alerta, AP per rebre les alertes dels seus veïns i passar la informació a la resta com a estació.

Una altra aplicació freqüent és utilitzar l'accés AP com a porta del darrera per a configurar el dispositiu quan perd la connexió a la xarxa com a estació. D'aquesta forma podem canviar el nom de la xarxa i contrasenya amb que es connecta.

També és freqüent aquesta doble connexió en un dispositiu mòbil (al que accedirem com a AP des del mòbil o tablet fora de casa) al que volem accedir des d'un ordinador fix a casa (és farragós haver de canviar la configuració d'una torre per accedir al nostre dispositiu, impossible si la torre es connecta per cable a la xarxa). Un exemple seria una càmera digital o un enregistrator de dades, controlable des del mòbil i amb unes fotografies i dades que volem descarregar després a l'ordinador de casa.

Estació o AP?

És interessant no canviar el *WiFi.mode()* si és possible. El nom de xarxa i la contrasenya s'emmagatzemen a la memòria *RTC* del ESP8266, de forma que no es perden amb un *RST* o apagat del sistema. Sempre i quan no canviem el *WiFi.mode()*. Aquesta estratègia evita haver de guardar nosaltres aquestes dades.

Utilitzar *WIFI_AP* és interessant en entorns hostils: tallers, conferències, demos, on de vegades l'accés a la xarxa és complicat. Penseu que els tipus de criptografia implementats són limitats. O les xarxes tenen limitada l'accés a IPs o ports no autoritzats.

Però amb *WIFI_AP* no tindrem accés a internet, limitant l'ús de llibreries javascript ubicades al núvol o l'enviament de dades a servidors. En aquest cas ens interessa *WIFI_STA*, encara que sigui generant una xarxa amb el nostre mòbil. Per cert, si la xarxa a la que ens connectem no té contrasenya hem de posar

```
WiFi.begin(ssid, "");
```

Finalment, recordeu que també podem apagar la WiFi (i estalviar bateria) amb

```
WiFi.mode(WIFI_OFF);
```

Al meu projecte «Càmera tèrmica» teniu un exemple de com engegar amb botons la WiFi, tant com *WIFI_AP* com a *WIFI_STA*. Fixeu-vos en la funció *initWifi()* al meu codi.

ThingSpeak: enregistrar les nostres dades al núvol

ThingSpeak és un servei gratuït que ens permet publicar al núvol les nostres dades i visualitzar-les de forma gràfica. A més a més permet fer l'anàlisi amb la potència de MATLAB.

Primer de tot crearem un compte a <http://thingspeak.com> i un canal on centralitzar les nostres mesures. Prenem nota del canal i del APIKEY associat per escriure'n dades.

Haurem d'instal·lar la llibreria *thinkspeak* a l'Arduino IDE (<https://github.com/mathworks/thingspeak-arduino>). Ho podeu fer amb el gestor de llibreries de l'IDE.

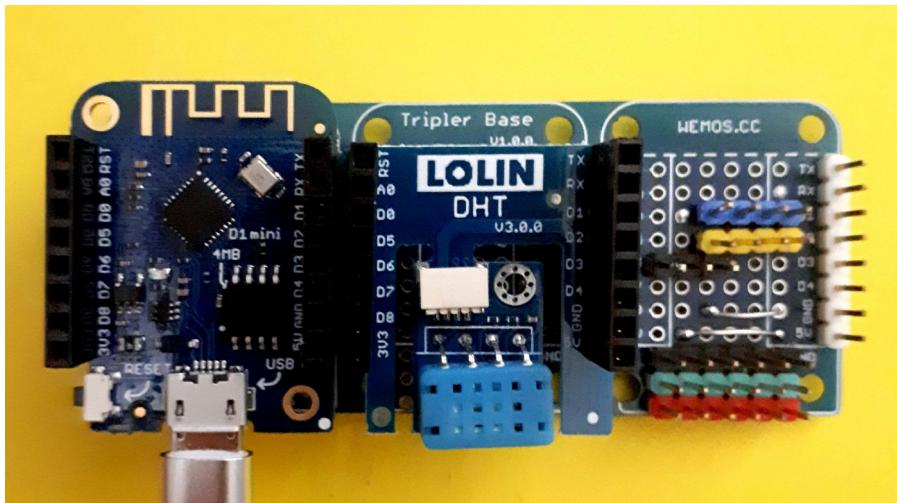
Podeu provar-lo amb el meu exemple *thingspeak*:

```
#include <ESP8266WiFi.h>
#include "ThingSpeak.h"
#include <WEMOS_DHT12.h>
DHT12 dht12;
const char *ssid = "Nom de la xarxa WiFi aqui";
const char *password = "Contrasenya de la WiFi aqui";
WiFiClient client;
unsigned long myChannelNumber = 135405; //canvia pel teu canal
const char *myWriteAPIKey = "API key aqui";
const int led = BUILTIN_LED; // internal blue led

void setup () {
  pinMode ( led, OUTPUT );
  digitalWrite ( led, HIGH );
  WiFi.begin ( ssid, password );
  while ( WiFi.status() != WL_CONNECTED ) {
    delay ( 500 );
  }
  ThingSpeak.begin(client);
}

void loop () {
  digitalWrite ( led, LOW );
  dht12.get();
  ThingSpeak.writeField(myChannelNumber, 1, dht12.cTemp, myWriteAPIKey);
  digitalWrite ( led, HIGH );
  delay(60000); // ThingSpeak will only accept updates every 15 seconds.
}
```

Part II. Connectats amb xarxa



Quant tenim més d'una dada el mecanisme és lleugerament diferent. Primer preparam les dades i després les enviem de cop. Veieu el meu exemple thinspeak2c:

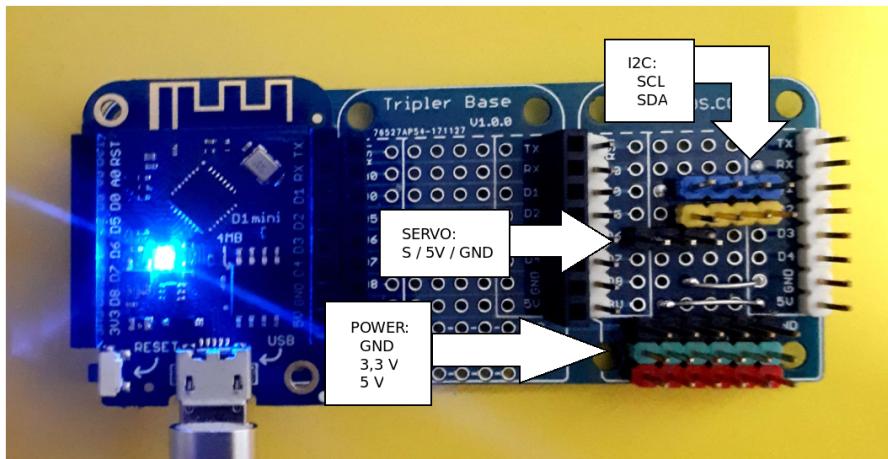
```
#include <ESP8266WiFi.h>
#include "ThingSpeak.h"
#include <WEMOS_DHT12.h>
DHT12 dht12;
const char *ssid = "Nom de la xarxa WiFi aquí";
const char *password = "Contrasenya de la WiFi aquí";
WiFiClient client;
unsigned long myChannelNumber = 185812; //canvia pel teu canal
const char * myWriteAPIKey = "API key aquí";
const int led = BUILTIN_LED; // internal blue led
int t;

void setup () {
  pinMode ( led, OUTPUT );
  digitalWrite ( led, HIGH );
  WiFi.begin ( ssid, password );
  while ( WiFi.status() != WL_CONNECTED ) {
    delay ( 500 );
  }
  ThingSpeak.begin(client);
}

void loop () {
  digitalWrite ( led, LOW );
  dht12.get();
  ThingSpeak.setField(1,dht12.cTemp);
  ThingSpeak.setField(2,dht12.humidity);
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  digitalWrite ( led, HIGH );
  delay(60000); // ThingSpeak will only accept updates every 15 seconds.
}
```

Part III. Utilitzant components externs

Hem sacrificat una de les 3 columnes de la triple base per posar-hi connectors mascle. Ara veurem la seva utilitat.



Les sortides no haurien de donar cap problema, però les entrades poden ser molt complicades si oblidem els pull-ups i pull-downs que porta la placa integrats, o el divisor de tensió integrat a l'entrada analògica i que pel meu gust es de massa baixa impedància.

Recordeu que quan treballem amb components I2C cal que siguin alimentats a 3,3V. Si necessitem treballar amb un component I2C a 5V, ens caldrà un adaptador de nivells pels senyals SCL i SDA. Connectar un component I2C de 5V sense aquest adaptador no només fa que la senyal no sigui fiable (el senyal de 3,3V de l'ESP8266 quedaria fora de les especificacions del bus I2C a 5V), a més a més podem fer malbé els components I2C que funcionen a 3,3V.

També recordeu que al bus I2C calen pull-ups a 3,3V a SCL i SDA. La majoria de shields i mòduls I2C porten aquests pull-ups integrats.

Part III. Utilitzant components externs

Per tant quan tenim un mòdul I2C farem les següent connexions:

GND → GND	(connector negre de 6 pins)
VCC → 3,3V	(connector verd de 6 pins)
SCL → D1	(connecor blau de 4 pins)
SDA → D2	(connector groc de 4 pins)

Si treballem amb un component I2C i no estem segurs de la seva adreça, podem utilitzar el programa *I2Cscanner*, que en farà un llistat dels dispositius I2C trobats:

```
#include <Wire.h>
void setup() {
    Wire.begin();
    Serial.begin(9600);
    while (!Serial);           // Leonardo: wait for serial monitor
    Serial.println("\nI2C Scanner");
}

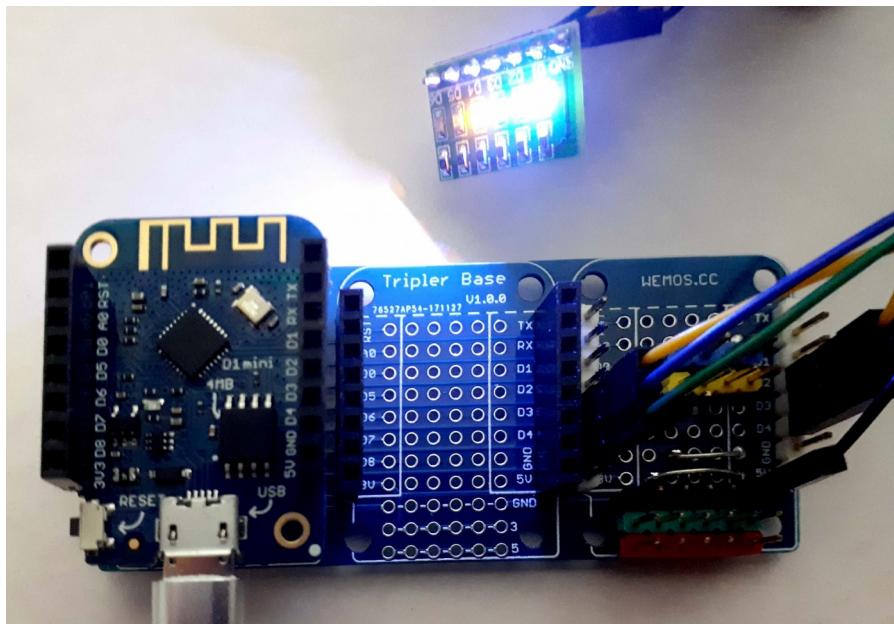
void loop() {
    byte error, address;
    int nDevices;
    Serial.println("Scanning...");
    nDevices = 0;
    for(address = 1; address < 127; address++ )
    {
        Wire.beginTransmission(address);
        error = Wire.endTransmission();
        if (error == 0)
        {
            Serial.print("I2C device found at address 0x");
            if (address<16)
                Serial.print("0");
            Serial.print(address,HEX);
            Serial.println(" !");
            nDevices++;
        }
        else if (error==4)
        {
            Serial.print("Unknown error at address 0x");
            if (address<16)
                Serial.print("0");
            Serial.println(address,HEX);
        }
    }
    if (nDevices == 0)
        Serial.println("No I2C devices found\n");
    else
        Serial.println("done\n");
    delay(5000);               // wait 5 seconds for next scan
}
```

Mòdul 6 LEDs

Ers tracta d'un mòdul molt econòmic i compacte, de 6 leds de diversos colors, cadascú amb la seva resistència de $1\text{ k}\Omega$ i que comparteixen negatiu.

Tots els pins D0-D8 funcionen perfectament com a sortida. Això si, donen 3,3 V en estat *HIGH*. Recordeu que l'ESP8266 funciona a aquest voltatge.

Utilitzarem els cables Dupont femella-femella per connectar el mòdul.



Part III. Utilitzant components externs

Veiem un programa senzill:

```
void setup() {  
    pinMode(D2,OUTPUT);  
    pinMode(D3,OUTPUT);  
    pinMode(D4,OUTPUT);  
    pinMode(D6,OUTPUT);  
    pinMode(D7,OUTPUT);  
    pinMode(D8,OUTPUT);  
}  
  
void loop() {  
    digitalWrite(D2,LOW);  
    digitalWrite(D3,LOW);  
    digitalWrite(D4,LOW);  
    digitalWrite(D6,LOW);  
    digitalWrite(D7,LOW);  
    digitalWrite(D8,LOW);  
    delay(1000);  
    digitalWrite(D2,HIGH);  
    delay(1000);  
    digitalWrite(D3,HIGH);  
    delay(1000);  
    digitalWrite(D4,HIGH);  
    delay(1000);  
    digitalWrite(D6,HIGH);  
    delay(1000);  
    digitalWrite(D7,HIGH);  
    delay(1000);  
    digitalWrite(D8,HIGH);  
    delay(1000);  
}
```

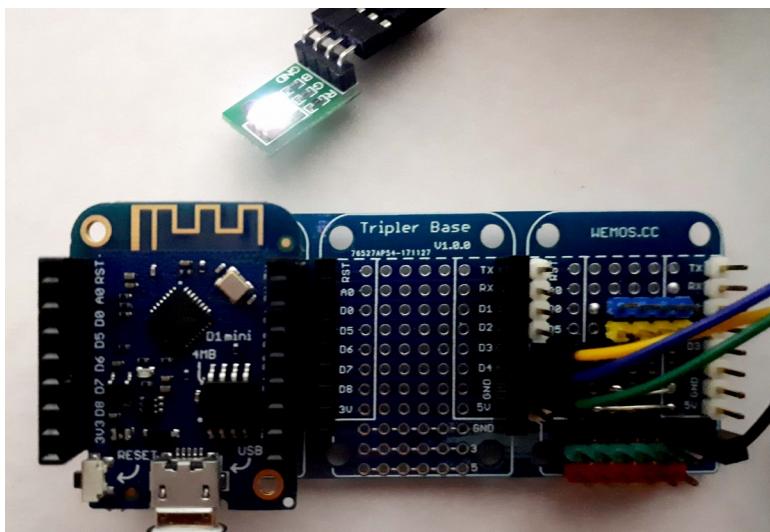
No us ha d'estranyar que quan el led connectat a D4 estigui apagat s'encengui el led blau integrat a l'ESP8266. Recordeu que aquest led està polaritzat a l'inrevés.

Mòdul LED RGB

Molt semblant al mòdul de 6 leds, cada entrada d'aquest led porta la seva resistència limitadora (270 Ω per verd i blau, 470 Ω pel vermell) i comparteixen negatiu.

Tots els pins D0-D8 permeten PWM. Ho comprovem amb un senzill programa:

```
void setup() {  
    pinMode(D6,OUTPUT);  
    pinMode(D7,OUTPUT);  
    pinMode(D8,OUTPUT);  
}  
  
void loop() {  
    int i,j,k;  
    for (i=0;i<256;i+=10) {  
        analogWrite(D6,i);  
        for (j=0;j<256;j+=10){  
            analogWrite(D7,j);  
            for (k=0;k<256;k+=10) {  
                analogWrite(D8,k);  
                delay(1);  
            }  
        }  
    }  
}
```

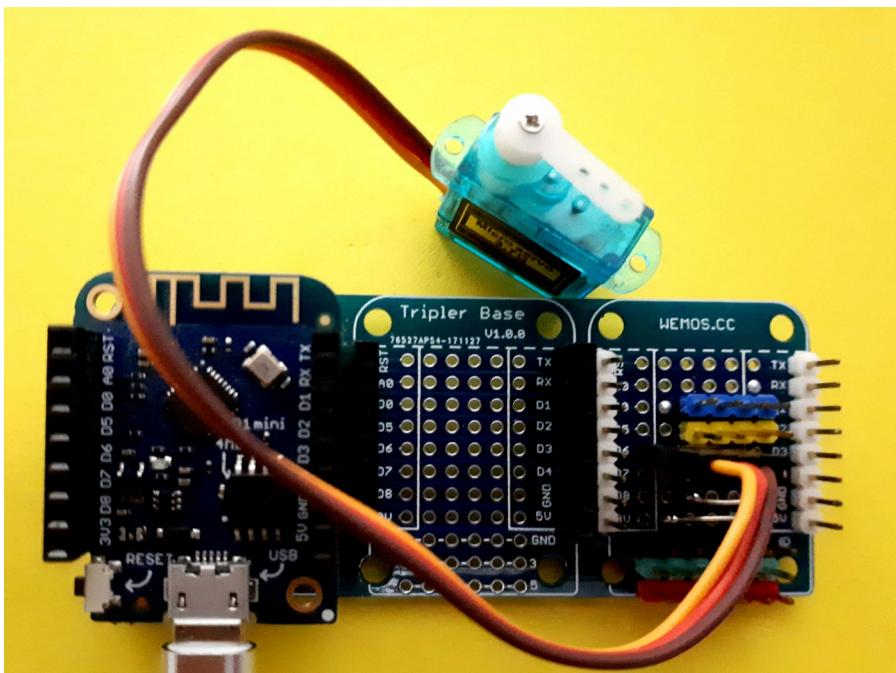


Microservo 3,7g

Aquest servo, econòmic (no tant com el de 9g, però ocupa menys a la capsula del kit) ens mostra com utilitzar el connector estàndard de servo (S / 5V / GND) que hem creat a la triple base.

Un dels avantatges dels servos és que agafen l'alimentació de forma independent. Per això el connector que hem fet té al mig 5V, encara que el microcontrolador ESP8266 funcioni a 3,3V.

Fixeu-vos que el cable taronja (senyal de control) ha de quedar a l'esquerra.



Podem provar-lo amb l'exemple de la llibreria servo específica pel ESP8266. Caldrà canviar el pin per D6.

IoT amb D1 mini (ESP8266) i codi Arduino

```
/* Sweep
by BARRAGAN <http://barraganstudio.com>
This example code is in the public domain.

modified 28 May 2015
by Michael C. Miller
modified 8 Nov 2013
by Scott Fitzgerald

http://arduino.cc/en/Tutorial/Sweep
*/

#include <Servo.h>

Servo myservo; // create servo object to control a servo
// twelve servo objects can be created on most boards

void setup() {
  myservo.attach(D6); // attaches the servo on GIO2 to the servo object
}

void loop() {
  int pos;

  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos); // tell servo to go to position in
variable 'pos'
    delay(15); // waits 15ms for the servo to reach the
position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos); // tell servo to go to position in
variable 'pos'
    delay(15); // waits 15ms for the servo to reach the
position
  }
}
```

Mòdul 4 polsadors

Mòdul compacte i econòmic, porta quatre polsadors que connecten a negatiu.

Com D8 porta un pull-up a negatiu, millor no utilitzar aquest pin.

D3 i D4 porten pull-ups a positiu. En principi els podem fer servir amb

```
pinMode(D3, INPUT);  
pinMode(D4, INPUT);
```

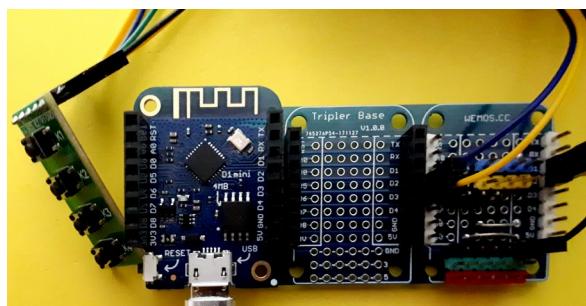
però no es recomanable fer servir aquestes entrades ja que si es fa un RST amb l'entrada a LOW no s'inicia el programa.

D0 no porta pull-up intern configurable, per tant tampoc el farem servir.

Finalment, D1, D2, D5, D6 i D7 permeten configurar un pull-up intern. Són les entrades que farem servir amb aquest mòdul amb les ordres

```
pinMode(D1, INPUT_PULLUP);  
pinMode(D2, INPUT_PULLUP);  
pinMode(D5, INPUT_PULLUP);  
pinMode(D6, INPUT_PULLUP);  
pinMode(D7, INPUT_PULLUP);
```

Veiem un petit exemple que ens mostra l'estat de les entrades pel canal sèrie:



IoT amb D1 mini (ESP8266) i codi Arduino

The screenshot shows a terminal window titled '/dev/ttyUSB0'. The window displays three sets of digital pin status prints from an Arduino sketch. The first set shows pins D1-HIGH, D2-HIGH, D5-HIGH, D6-LOW. The second set shows pins D1-HIGH, D2-HIGH, D5-HIGH, D6-LOW. The third set shows pins D1-HIGH, D2-HIGH, D5-HIGH, D6-LOW. Below the terminal window, there are several control buttons: a checkbox for 'Desplaçament automàtic' (Automatic scroll), a dropdown menu for 'Sense salts de línia' (No line breaks), a dropdown menu for '9600 baud' (baud rate), and a 'Clear output' button.

```
D1: HIGH
D2: HIGH
D5: HIGH
D6: LOW
-----
D1: HIGH
D2: HIGH
D5: HIGH
D6: LOW
-----
D1: HIGH
D2: HIGH
D5: HIGH
D6: LOW
```

Desplaçament automàtic Sense salts de línia ▾ 9600 baud ▾ Clear output

```
void setup() {
    pinMode(D1, INPUT_PULLUP);
    pinMode(D2, INPUT_PULLUP);
    pinMode(D5, INPUT_PULLUP);
    pinMode(D6, INPUT_PULLUP);
    Serial.begin(9600);
}

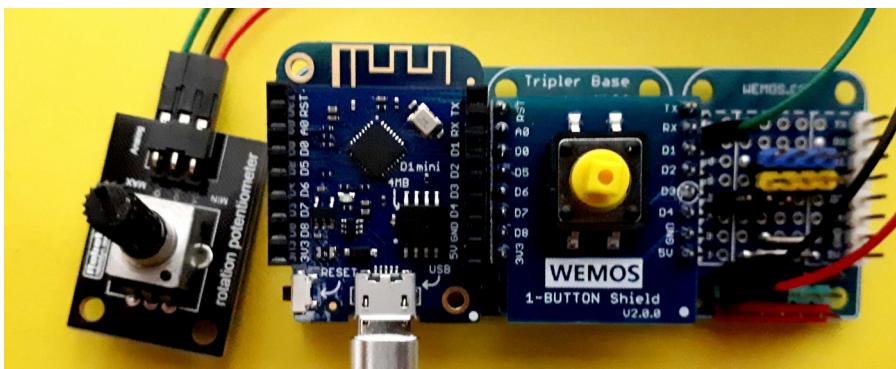
void loop() {
    Serial.print("D1: ");
    if (digitalRead(D1)) {
        Serial.println("HIGH");
    }
    else {
        Serial.println("LOW");
    }
    Serial.print("D2: ");
    if (digitalRead(D2)) {
        Serial.println("HIGH");
    }
    else {
        Serial.println("LOW");
    }
    Serial.print("D5: ");
    if (digitalRead(D5)) {
        Serial.println("HIGH");
    }
    else {
        Serial.println("LOW");
    }
    Serial.print("D6: ");
    if (digitalRead(D6)) {
        Serial.println("HIGH");
    }
    else {
        Serial.println("LOW");
    }
    Serial.println("-----");
    delay(2000);
}
```

Mòdul potenciómetre

Aquest mòdul porta un potenciómetre de $10\text{ k}\Omega$ que podem utilitzar com a divisor de tensió i llegir el seu valor amb l'entrada analògica A0.

Les entrades analògiques són un problema en l'ESP8266. Només en tenim una, i internament treballa en el rang 0V:1V. A la placa D1 mini porta un divisor de tensió integrat per agafar 1/3 com a mostra de la tensió d'entrada, amb la qual cosa el rang passa a ser de 0V : 3,3V. El problema és que el divisor de tensió està fet amb resistències de valor molt petit pel meu gust: $100\text{ k}\Omega$ i $220\text{ k}\Omega$, deixant una impedància d'entrada baixa. Si bé podem llegir valors de consigna amb aquest potenciómetre, el fa inservible per a altres aplicacions, com ara la utilització de leds com a sensors de llum.

Amb el meu exemple *Analog2WiFi* podem llegir el valor del potenciómetre i comparar-lo amb un valor de referència, de forma que en superar-lo encengui el led integrat. A més a més, si activem el pulsador del shield 1-button genera una xarxa WiFi i mostra una senzilla gràfica de 400 lectures amb codi SVG²:



2 Podeu conèixer més sobre el codi SVG (Scalable Vector Graphics) a HTML5 a https://www.w3schools.com/html/html5_svg.asp

IoT amb D1 mini (ESP8266) i codi Arduino

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
const char *ssid = "ESPaP";
const char *password = "robotica";
ESP8266WebServer server(80);

void handleRoot() {
  server.send(200, "text/html", "<html><head></head><body><a href='graf.svg'>Grafica</a><br><a href='rst'>Reset</a></body></html>");
}

const int buttonPin = D3;
const int ledPin = BUILTIN_LED;

void setup() {
  delay(1000);
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);
}

void loop() {
  if (digitalRead(buttonPin)==LOW) {
    prova();
  }
  else {
    if (analogRead(A0)>512) {
      digitalWrite(ledPin, LOW);
      delay(100);
      digitalWrite(ledPin, HIGH);
      delay(100);
    }
  }
  delay(1);
}

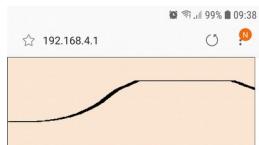
void prova() {
  digitalWrite(ledPin, LOW);
  WiFi.softAP(ssid, password);
  server.on("/", handleRoot);
  server.on("/graf.svg", grafica);
  server.on("/rst", reinicia);
  server.begin();
  while (true) {
    server.handleClient();
  }
}

void grafica() {
  String out = "";
  char temp[1000];
  out += "<svg xmlns=\"http://www.w3.org/2000/svg\" version=\"1.1\" width=\"400\" height=\"150\">\n";
  out += "<rect width=\"400\" height=\"150\" fill=\"rgb(250, 230, 210)\" stroke-width=\"1\" stroke=\"rgb(0, 0, 0)\" />\n";
  out += "<g stroke=\"black\">\n";
  int y = analogRead(A0)/10;
  delay(5);
  for (int x = 1; x < 399; x++) {
    int y2 = analogRead(A0)/10;
    delay(5);
    sprintf(temp, "<line x1=\"%d\" y1=\"%d\" x2=\"%d\" y2=\"%d\" stroke-width=\"1\" />\n", x, 140 - y, x + 10, 140 - y2);
  }
}
```

Part III. Utilitzant components externs

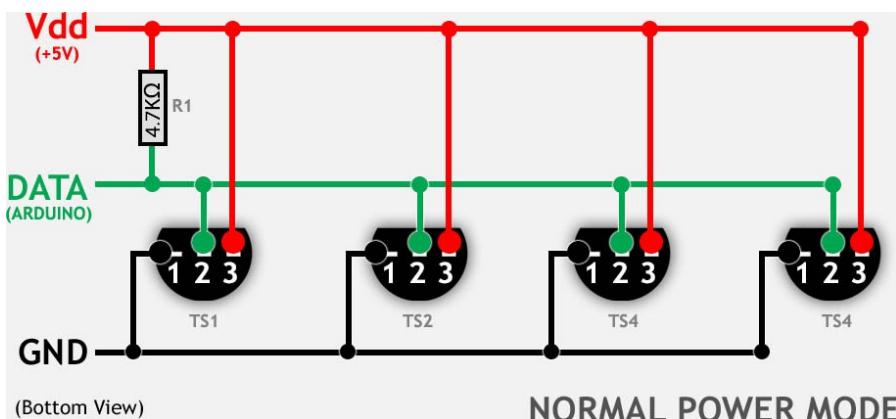
```
    out += temp;
    y = y2;
}
out += "</g>\n</svg>\n";
server.send ( 200, "image/svg+xml", out);
}

void reinicia() {
    ESP.restart();
}
```



Mòdul sensor temperatura DS18B20

Aquest sensor és molt interessant. De fet, un dels meus favorits. Dona la temperatura amb una precisió ($\pm 0,5^{\circ}\text{C}$) i resolució (de 9 a 12 bits) altíssimes en un rang molt ampli de temperatures (de -55°C a $+85^{\circ}\text{C}$). Essent digital fa servir només 3 fils (Vcc, GND i senyal) amb un bus propi on podem connectar diversos sensors. Com cada sensor porta gravat amb làser un codi únic de 64 bits a la seva ROM en teoria podríem connectar un nombre il·limitat de sensors al mateix pin.



Com el bus porta un pull-up a 3,3 V, no podem fer servir els pins D3 ni D4 per a aquest tipus de sensor.

Existeix un shield pel D1 mini amb aquest sensor (veure pàg. 124). Malauradament va connectat al pin D2, amb la qual cosa col·lisiona amb el bus I2C, i no porta un connector per afegir altres sensors externs. Per això el kit porta un mòdul extern, que permet soldar un segon connector de 3 pins per connectar més sensors al mateix bus.

Per utilitzar aquest sensor cal tenir les llibreries *OneWire* i *DallasTemperature*, que podeu afegir amb el gestor de llibreries.

Part III. Utilitzant components externs

A l'exemple DS18B20_AJAX utilitzem aquest sensor per mostrar via web la temperatura cada segon:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <OneWire.h>
#include <DallasTemperature.h>

const char *ssid = "nom de la xarxa";
const char *password = "contrasenya";

ESP8266WebServer server ( 80 );
#define ONE_WIRE_BUS D7 // DS18B20 pin
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature DS18B20(&oneWire);
float t;

const char *pagina = "<html> \
<head> \
    <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'> \
    <meta name='viewport' content='width=device-width'> \
    <title>AJAX Test</title> \
    <style>body{padding:0;margin:0;background:#999}</style> \
    <script> \
function loadTemp() { \
    var xhttp = new XMLHttpRequest(); \
    var temp = 0; \
    xhttp.open('GET', 'ajax_info.txt', false); \
    xhttp.send(); \
    temp = parseFloat(xhttp.responseText); \
    return (temp); \
} \
</script> \
</head> \
<body> \
<p>Temperatura:</p> \
<p id='demo'></p> \
<script> \
var myVar = setInterval(myTimer, 1000); \
function myTimer() { \
    document.getElementById('demo').innerHTML = loadTemp(); \
} \
</script> \
</body> \
</html>"; 

void handleRoot() {
    server.send ( 200, "text/html", pagina );
}

void handleNotFound() {
    String message = "File Not Found\n\n";
    message += "URI: ";
    message += server.uri();
    message += "\nMethod: ";
    message += ( server.method() == HTTP_GET ) ? "GET" : "POST";
    message += "\nArguments: ";
    message += server.args();
    message += "\n";
    for ( uint8_t i = 0; i < server.args(); i++ ) {
        message += " " + server.argName ( i ) + ": " + server.arg ( i ) + "\n";
    }
}
```

IoT amb D1 mini (ESP8266) i codi Arduino

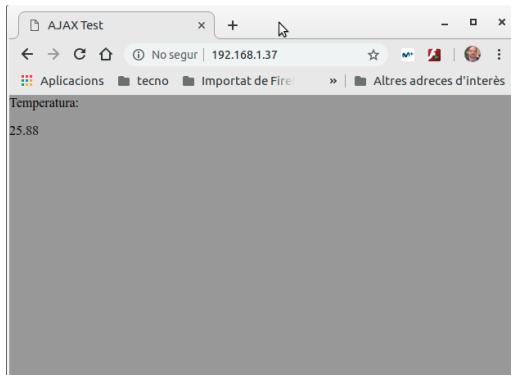
```
}

server.send ( 404, "text/plain", message );
}

void handleAjax(){
DS18B20.requestTemperatures();
t = DS18B20.getTempCByIndex(0);
int t1,t2;
Serial.print("Temperature: ");
Serial.println(t);
String out = "";
char temp[100];
t1 = int(t);
snprintf(temp, 100, "%d", t1);
out += temp;
out += ".";
t2 = int((t+0.005)*100)-100*t1;
snprintf(temp, 100, "%d", t2);
out += temp;
server.send ( 200, "text/txt", out);
}

void setup ( void ) {
Serial.begin ( 115200 );
WiFi.begin ( ssid, password );
DS18B20.begin();
Serial.println ( "" );
while ( WiFi.status() != WL_CONNECTED ) {
delay ( 500 );
Serial.print ( "." );
}
Serial.println ( "" );
Serial.print ( "Connected to " );
Serial.println ( ssid );
Serial.print ( "IP address: " );
Serial.println ( WiFi.localIP() );
server.on ( "/", handleRoot );
server.on ( "/ajax_info.txt", handleAjax );
server.onNotFound ( handleNotFound );
server.begin();
Serial.println ( "HTTP server started" );
}

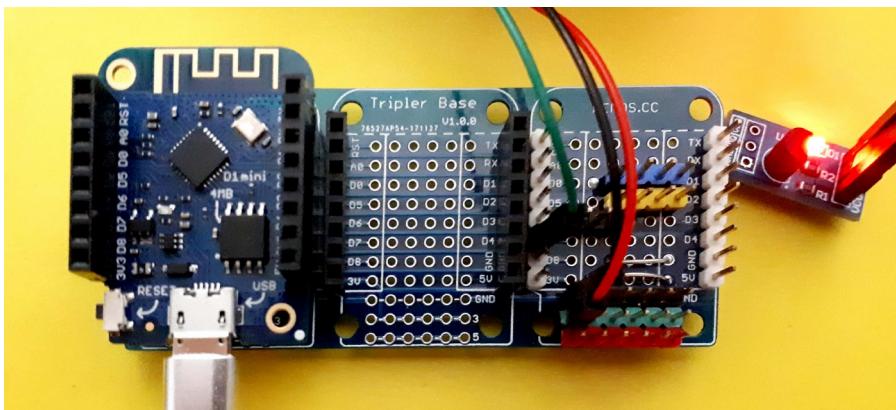
void loop ( void ) {
server.handleClient();
}
```



Part III. Utilitzant components externs

Fixeu-vos com amb AJAX³ podem actualitzar una pàgina web minimitzant el trànsit de dades. A la part HTML5 bàsicament ens cal una funció javascript per agafar les dades (*loadTemp()* en el nostre cas) i trucar-la periòdicament⁴ per actualitzar la pàgina (funció *myTimer()* activada per *setInterval(myTimer, temps en ms)*).

Al codi Arduino ens cal una funció (*handleAjax()* en el nostre cas) que llegeixi la temperatura i la lliuri en el format adequat, activada quan demanem una pàgina en concret (*ajax_info.txt* en el nostre cas).



-
- 3 Podeu trobar més informació sobre com utilitzar AJAX al vostre codi javascript a https://www.w3schools.com/js/js_ajax_intro.asp
 - 4 Podeu trobar més informació sobre *setInterval()* i els seus paràmetres a https://www.w3schools.com/jsref/met_win_setinterval.asp

Mòdul sensor de llum I2C BH1750FVI

Aquest sensor pot treballar amb 3,3V. També està disponible com a shield del D1 mini.

El principal avantatge d'aquest sensor és que ens dona la llum directament en lux. La precisió (que pot arribar a ser de $\pm 0,5$ lx) depèn del mode d'operació, i repercutex en el temps de mesura. L'exemple BH1750 utilitza el mode d'alta resolució continu, amb un temps de mesura de 120 ms i una precisió de ± 1 lx:

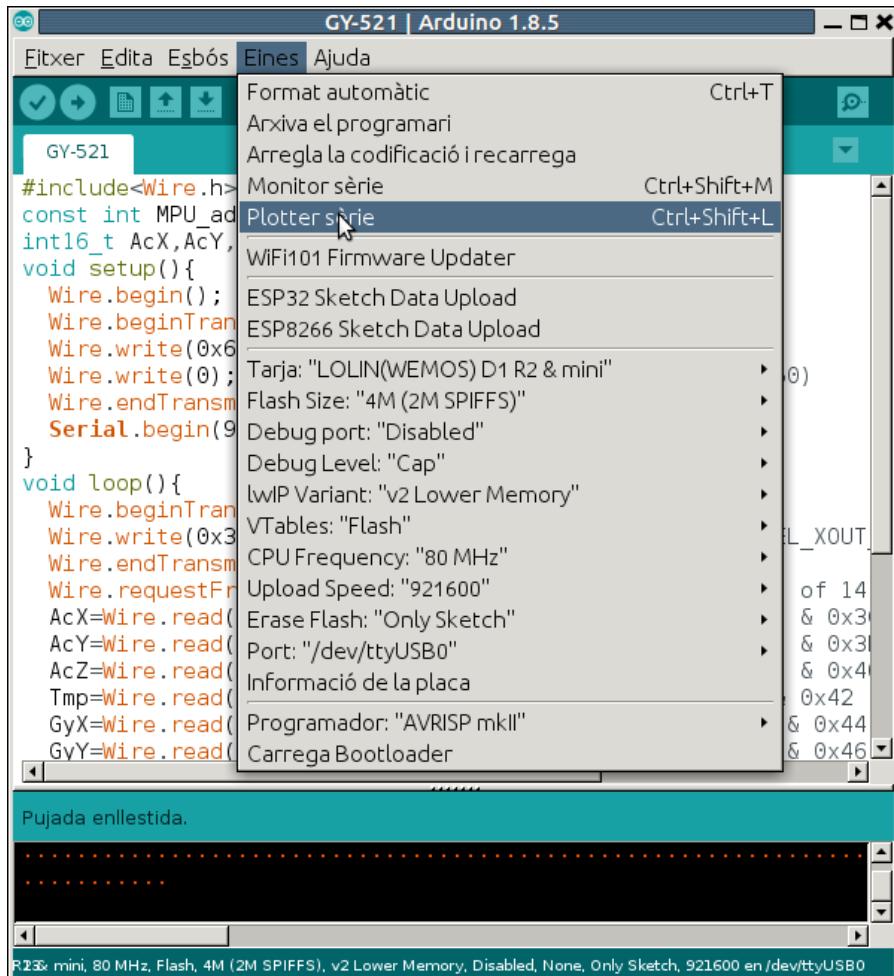
```
//Install [claws/BH1750 Library](https://github.com/claws/BH1750) first.  
#include <Wire.h>  
#include <BH1750.h>  
BH1750 lightMeter(0x23);  
  
void setup(){  
    Serial.begin(9600);  
    Wire.begin();  
    if (lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE)) {  
        Serial.println(F("BH1750 Advanced begin"));  
    }  
    else {  
        Serial.println(F("Error initialising BH1750"));  
    }  
}  
  
void loop() {  
    uint16_t lux = lightMeter.readLightLevel();  
    Serial.print("Light: ");  
    Serial.print(lux);  
    Serial.println(" lx");  
    delay(1000);  
}
```



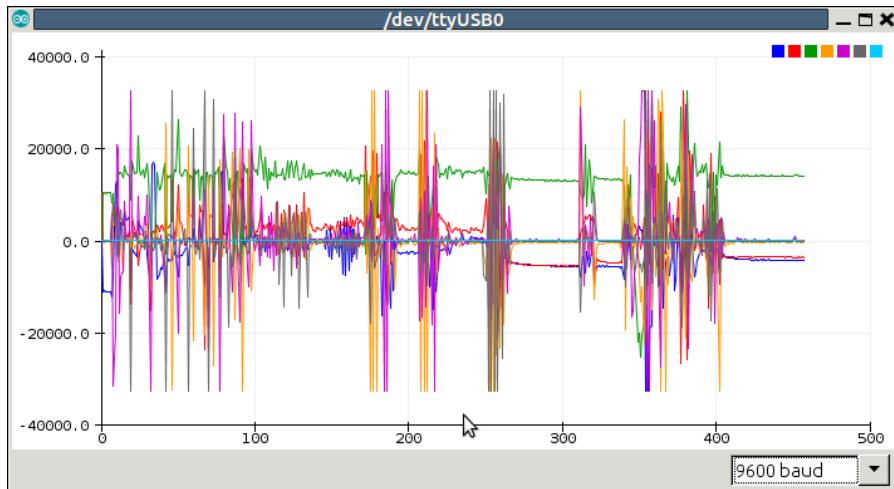
Mòdul acceleròmetre + giroscopi I2C GY-521

Aquest mòdul I2C, que pot treballar a 3,3V i és molt econòmic, ens dona la temperatura i les acceleracions lineals i angulars segons els tres eixos X, Y i Z.

A l'exemple GY-521 utilitzarem el plotter sèrie per visualitzar els canvis en aquests valors:



IoT amb D1 mini (ESP8266) i codi Arduino



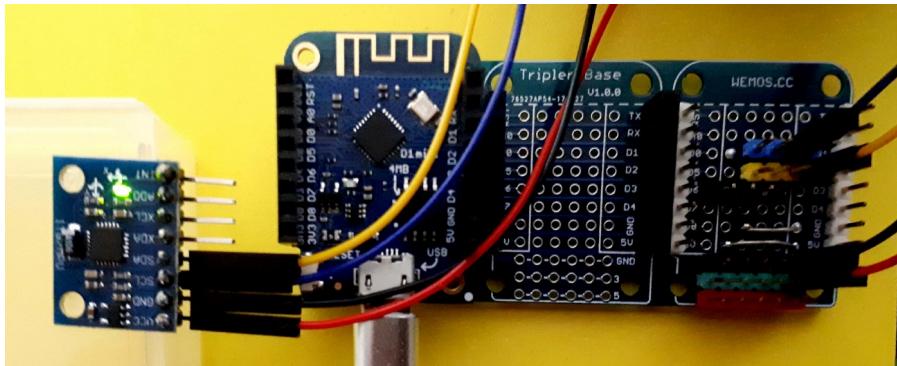
```
#include<Wire.h>
const int MPU_addr=0x68; // I2C address of the MPU-6050
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
void setup(){
    Wire.begin();
    Wire.beginTransmission(MPU_addr);
    Wire.write(0x6B); // PWR_MGMT_1 register
    Wire.write(0); // set to zero (wakes up the MPU-6050)
    Wire.endTransmission(true);
    Serial.begin(9600);
}
void loop(){
    Wire.beginTransmission(MPU_addr);
    Wire.write(0x3B); // starting with register 0x3B (ACCEL_XOUT_H)
    Wire.endTransmission(false);
    Wire.requestFrom(MPU_addr,14,true); // request a total of 14 registers
    AcX=Wire.read()<<8|Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
    AcY=Wire.read()<<8|Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
    AcZ=Wire.read()<<8|Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
    Tmp=Wire.read()<<8|Wire.read(); // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
    GyX=Wire.read()<<8|Wire.read(); // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
    GyY=Wire.read()<<8|Wire.read(); // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
    GyZ=Wire.read()<<8|Wire.read(); // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
    Serial.print(AcX); Serial.print(",");
    Serial.print(AcY); Serial.print(",");
    Serial.print(AcZ); Serial.print(",");
    Serial.print(GyX); Serial.print(",");
    Serial.print(GyY); Serial.print(",");
    Serial.print(GyZ); Serial.print(",");
    Serial.println(Tmp/340.00+36.53); //equation for temperature in degrees C
    from datasheet
    delay(333);
}
```

Fixeu-vos que en aquest cas no he fet servir cap llibreria, sinó comandes I2C pures per llegir els registres del dispositiu. En realitat, les llibreries de components I2C el que fan es facilitar aquestes

Part III. Utilitzant components externs

comandes sota el nom d'una funció. El cor d'aquest mòdul, el MPU6050, té 118 registres!

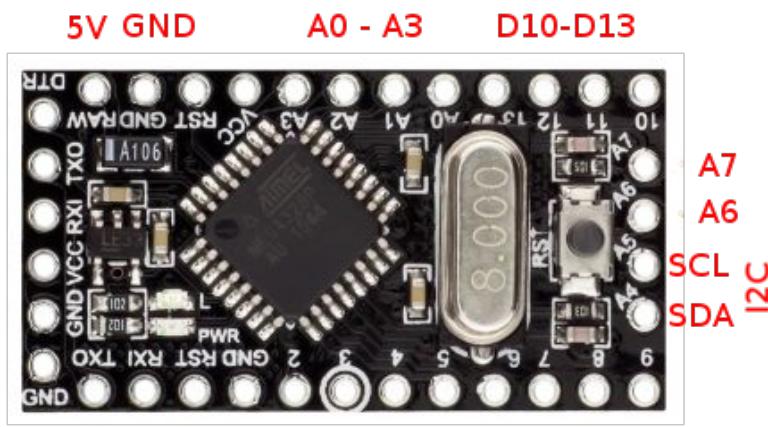
Però existeix una llibreria per a aquest mòdul. Es diu MPU6050, per si voleu provar-la.



Arduino Pro mini com a esclau I2C

Com heu vist les dues limitacions més importants de l'ESP8266 són l'existència de només una entrada analògica (i, a més a més, de petita impedància) i el reduït nombre d'entrades digitals disponibles (especialment si fem servir el bus I2C).

Per això al kit trobareu un arduino pro mini a 8 MHz i 3,3 V configurat com a esclau I2C. Amb això afegim 6 entrades analògiques (d'alta impedància) i 12 ports I/O.



Aquest mòdul funciona a 3,3 V, que pot generar amb un regulador de tensió propi. Per això el connectarem als 5 V de la placa triple (d'aquesta manera repartim el consum a 3,3 V entre els reguladors del D1 mini i del arduino pro mini).

He preparat una llibreria (I2CsapmR1) per utilitzar fàcilment aquest dispositiu. La trobareu a (<https://github.com/jorts64/I2CsapmR1>). Veiem l'exemple *simple* que incorpora:

Part III. Utilitzant components externs

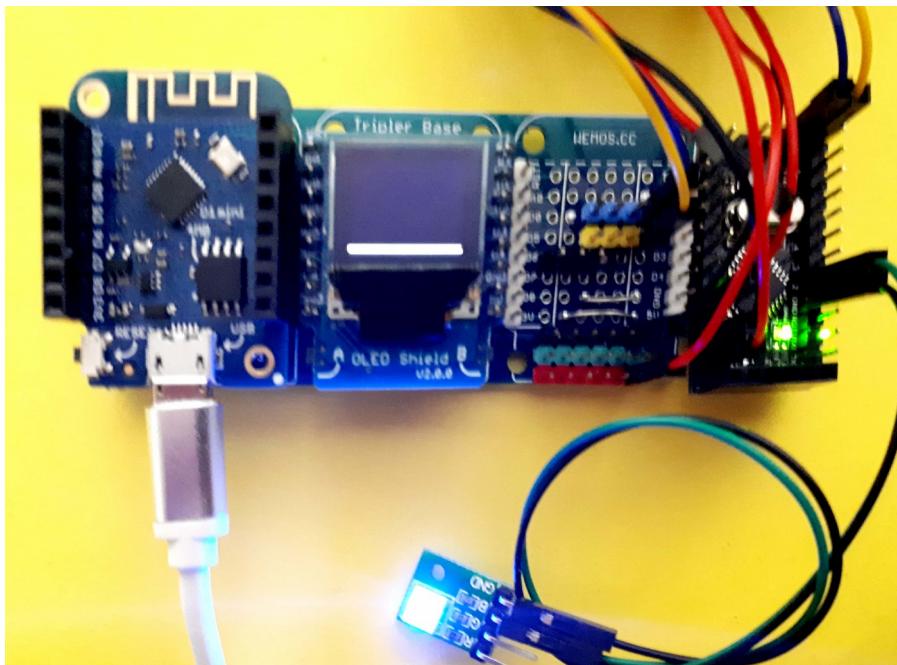
```
#include "I2CsapmR1.h"
#include <Wire.h>

I2CsapmR1 SAPM(2);

void setup() {
    Serial.begin(9600);
    SAPM.begin();
    SAPM.write(I2CsapmR1_M2,OUTPUT);
    SAPM.write(I2CsapmR1_M3,INPUT_PULLUP);
}

void loop() {
    SAPM.write(I2CsapmR1_D2,HIGH);
    int sensorValue = SAPM.read(I2CsapmR1_A0);
    int digValue = SAPM.read(I2CsapmR1_D3);
    Serial.print(sensorValue);
    Serial.print(" / ");
    Serial.println(digValue);
    delay(100);
    SAPM.write(I2CsapmR1_D2,LOW);
    delay(100);
}
```

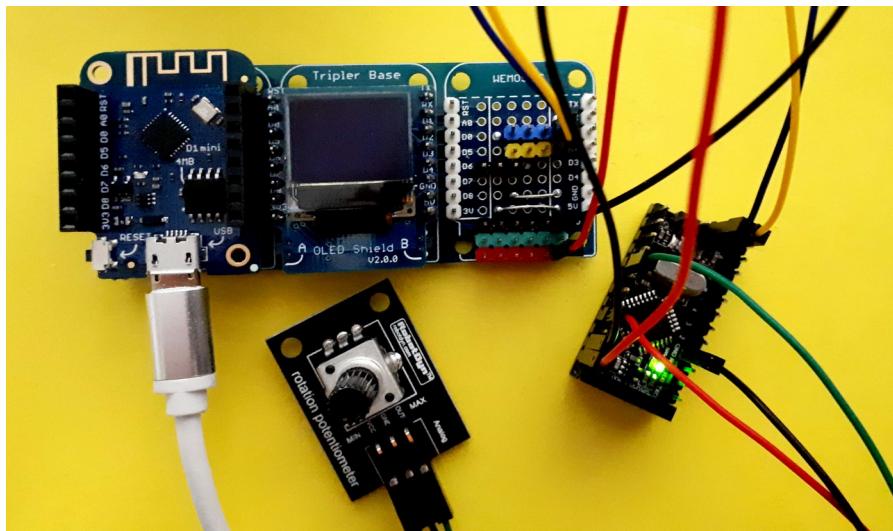
Si connectem un led al pin 2 de l'arduino pro mini veurem que fa pampallugues⁵:



⁵ Encara que aquest exemple no fa servir el display OLED l'he posat per tenir al menys un dispositiu I2C al bus que posi els seus pull-ups a SCL i SDA.

IoT amb D1 mini (ESP8266) i codi Arduino

Si ara connectem el potenciómetre a l'entrada A0 de l'arduino pro mini i obrim el monitor sèrie llegirem el seu valor (primer nombre de cada línia):

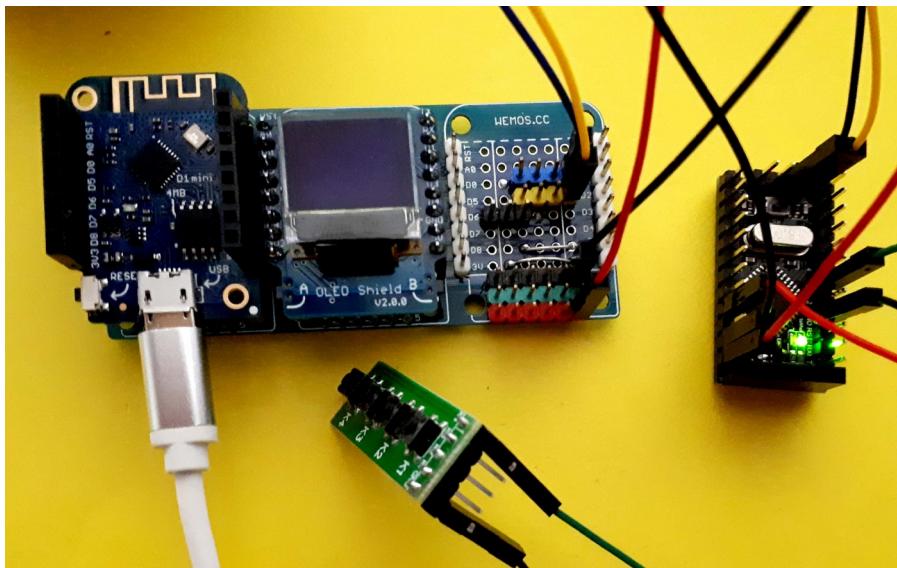


```
/dev/ttyUSB0
703 / 0
566 / 0
461 / 0
397 / 0
354 / 0
338 / 0
412 / 0
552 / 0
651 / 0
731 / 0
775 / 0
852 / 0
838 / 0
686 / 0
564 / 0
```

Desplaçament automàtic Sense salts de línia ▾ 9600 baud ▾ Clear output

Ara connectem un pulsador a l'entrada 3 de l'arduino pro mini i obrim el monitor sèrie. Veurem els canvis a l'entrada (segon nombre de cada línia):

Part III. Utilitzant components externs



```
/dev/ttyUSB0
Envia
31 / 0
38 / 0
-1 / 0
0 / 0
5 / 1
33 / 1
32 / 1
0 / 1
0 / 1
17 / 1
35 / 1
6 / 1
0 / 1
17 / 1
38 / 0

 Desplaçament automàtic  Sense salts de línia  9600 baud  Clear output
```

Analitzem una mica el codi:

Al setup() hem configurat les entrades digitals tal com ho fariem amb pinMode():

```
SAPM.write(I2CsapmR1_M2,OUTPUT);
SAPM.write(I2CsapmR1_M3,INPUT_PULLUP);
```

IoT amb D1 mini (ESP8266) i codi Arduino

Al loop hem llegit les entrades digitals i analògiques i canviat les sortides digitals accedint al seu registre:

```
SAPM.write(I2CsapmR1_D2,HIGH);
int sensorValue = SAPM.read(I2CsapmR1_A0);
int digValue = SAPM.read(I2CsapmR1_D3);
```

Com els registres són consecutius podem utilitzar bucles per configurar o accedir a un conjunt de pins:

```
#include "I2CsapmR1.h"
I2CsapmR1 SAPM(2);
int diginp[8];
int aninp[6];
void setup() {
    SAPM.begin();
    for (int i=I2CsapmR1_M2;i<=I2CsapmR1_M9;i++)
        SAPM.write(i,INPUT_PULLUP);
}

void loop() {
    for (int i=I2CsapmR1_M2;i<=I2CsapmR1_M9;i++)
        diginp[i-I2CsapmR1_M2]=SAPM.read(i);
    for (int i=I2CsapmR1_A0;i<=I2CsapmR1_A7;i++)
        aninp[i-I2CsapmR1_A0]=SAPM.read(i);
}
```

L'avantatge d'aquest dispositiu és el seu preu, molt econòmic, i la seva flexibilitat. Alterant el firmware⁶ podem utilitzar-lo com a una navalla suïssa per a qualsevol aplicació:

- sortides amb pwm
- control de servos
- control de motors pas a pas
- lectura d'encoders rotatoris
- lectura de teclats matricials

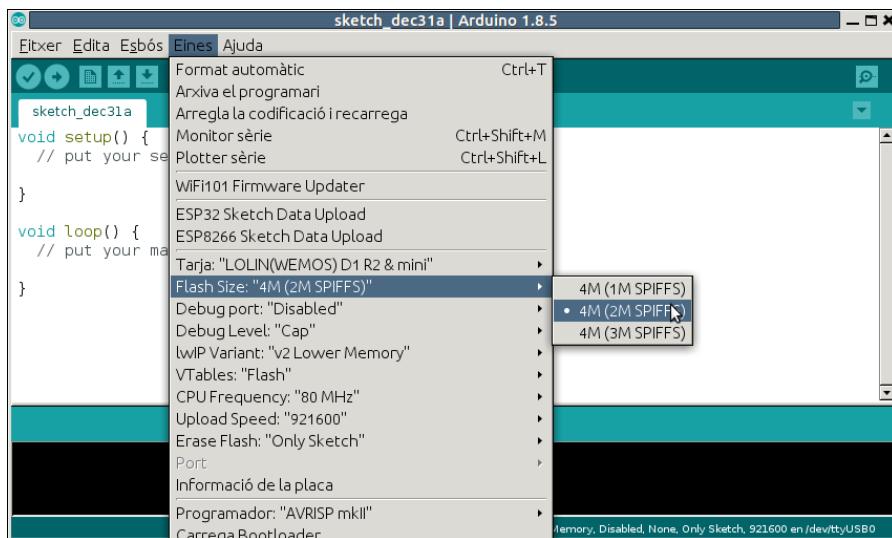
A més a més en podem tenir tants arduino pro mini al bus I2C com vulguem. Només cal canviar l'adreça I2C dels dispositius al seu firmware.

⁶ Veure Annex: Firmware de l'arduino pro mini

Part IV. El sistema de fitxers SPIFFS. Utilització a un servidor web.

Una de les joies de l'ESP8266 és el seu sistema de fitxers SPIFFS, que simula una SD. Com l'arduino IDE només aprofita 1 MB de memòria i el nostre D1 mini en té 4 MB de flash, podem dedicar algun MB a aquesta SD virtual.

Per utilitzar aquesta opció cal escollir a l'IDE la mida de la SD virtual. Encara que podríem dedicar els 3 MB restants, jo prefereixo 2 MB, reservant 1 MB per l'actualització WiFi del programa (OTA)⁷.



Cal crear a la carpeta del programa una subcarpeta amb el nom data, on posarem els fitxers amb que volem carregar inicialment la SD virtual. La imatge d'aquesta SD virtual l'enviarem al nostre D1 mini amb l'opció ESP8266 Sketch Data Upload⁸, que haurem d'utilitzar

7 Veure Part IX. Actualització WiFi del programa (OTA)

8 Cal instal·lar de forma separada aquesta eina. Veure Annex : Instal·lació Arduino per a ESP8266.

IoT amb D1 mini (ESP8266) i codi Arduino
amb el monitor sèrie tancat (si el tenim obert tindrem problemes per connectar-nos al D1 mini per enviar aquesta imatge de la SD).

Al programa cal posar la línia

```
#include <FS.h>
```

Us pot semblar que 2 MB són molt poc, especialment tenint en compte la capacitat de les SDs actuals, però al llarg d'aquest capítol veureu que en podem treure un bon profit d'aquests 2 MB.

Afegint fitxers al nostre servidor web. La funció `serveStatic()`.

De vegades l'únic que ens interessa es poder tenir fitxers a la nostra SD virtual amb els quals millorar l'aspecte de les pàgines webs generades: imatges, codi javascript o estils css, manuals en PDF ...

La funció `serveStatic()` fa justament això: quan es demana un fitxer al servidor web que comença amb un cert patró el busca a la SD virtual i l'envia directament. Així de simple.

Veiem el meu exemple `serveStatic`:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <FS.h>

const char *ssid = "Nom de la xarxa";
const char *password = "contrasenya";

ESP8266WebServer server ( 80 );

void handleRoot() {
    char temp[400];
    int sec = millis() / 1000;
    int min = sec / 60;
    int hr = min / 60;
    sprintf ( temp, 400,
"<html>\n<head>\n    <meta http-equiv='refresh' content='5' />\n    <title>ESP8266 Demo</title>\n    <style>\n        body { background-color: #cccccc; font-family: Arial, Helvetica, Sans-\nSerif; Color: #000088; }\n    </style>\n</head>\n<body>\n    <h1>Benvinguts al m&oacute;n IoT</h1>\n    <p>Uptime: %02d:%02d:%02d</p>\n    <img src='/img/IoT.jpg'>\n</body>\n</html>",
        hr, min % 60, sec % 60
    );
    server.send ( 200, "text/html", temp );
}

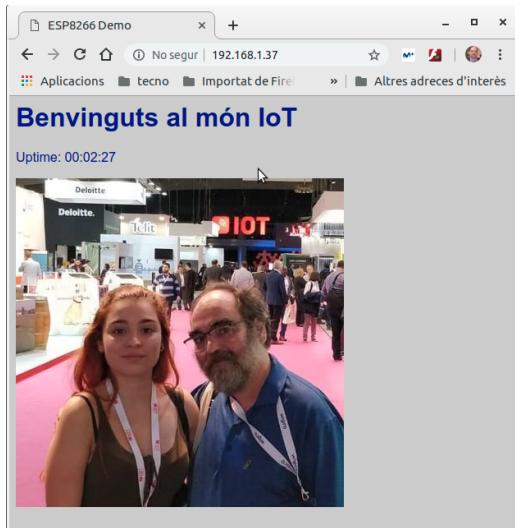
void handleNotFound() {
    String message = "File Not Found\n\n";
    message += "URI: ";
    message += server.uri();
```

IoT amb D1 mini (ESP8266) i codi Arduino

```
message += "\nMethod: ";
message += ( server.method() == HTTP_GET ) ? "GET" : "POST";
message += "\nArguments: ";
message += server.args();
message += "\n";
for ( uint8_t i = 0; i < server.args(); i++ ) {
    message += " " + server.argName ( i ) + ":" + server.arg ( i ) + "\n";
}
server.send ( 404, "text/plain", message );
}

void setup ( void ) {
Serial.begin(9600);
WiFi.begin ( ssid, password );
SPIFFS.begin();
while ( WiFi.status() != WL_CONNECTED ) {
    delay ( 500 );
}
Serial.println ( "" );
Serial.print ( "Connected to " );
Serial.println ( ssid );
Serial.print ( "IP address: " );
Serial.println ( WiFi.localIP() );
server.on ( "/", handleRoot );
server.serveStatic("/img", SPIFFS, "/img"); // imatges dir
server.onNotFound ( handleNotFound );
server.begin();
Serial.println ( "HTTP server started" );
}

void loop ( void ) {
    server.handleClient();
}
```



Com podreu suposar analitzant el codi a la carpeta *data* he creat una subcarpeta *img* on he posat una fotografia amb el nom *IoT.jpg* (que

Part IV. El sistema de fitxers SPIFFS. Utilització a un servidor web.
millor fotografia que una amb la meva filla Ingrid visitant la *IOTSWC18* el passat 17 d'octubre!).

Amb la línia

```
server.serveStatic("/img", SPIFFS, "/img"); // imatges dir
```

fem tota la màgia. Ara quan el servidor rep una petició d'un fitxer que comença per *img* el busca a la carpeta *img* de la SD virtual i l'envia directament.

Enregistrar dades a la SD virtual i poder recuperar-les via WiFi. Generació de fitxers compatibles amb fulls de càlcul

Aquesta és una tasca molt comú: un datalogger. Si emmagatzemem les dades a la SD en un fitxer CSV després el podrem recuperar mitjançant el servidor web i obrir-lo amb qualsevol paquet ofimàtic⁹.

Veiem el meu exemple *datalogger* que utilitza el DHT shield per mesurar la temperatura:



```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <FS.h>
#include <WEMOS_DHT12.h>

DHT12 dht12;
ESP8266WebServer server ( 80 );
unsigned long ara;

const char *ssid = "jortsnet";
const char *password = "9periodico";

void espera (int n) {
```

⁹ Si voleu un ànalisi més profund de les dades us recomano el programari QtiPlot, disponible com a GPLv2 a <https://sourceforge.net/projects/qtiplot.berlios/>

Part IV. El sistema de fitxers SPIFFS. Utilització a un servidor web.

```
while (millis()<ara+n) {
    delay(1);
    server.handleClient();
}

void setup() {
    Serial.begin(9600);
    SPIFFS.begin();
    WiFi.begin ( ssid, password );
    while ( WiFi.status() != WL_CONNECTED ) {
        delay ( 500 );
    }
    Serial.println ( "" );
    Serial.print ( "Connected to " );
    Serial.println ( ssid );
    Serial.print ( "IP address: " );
    Serial.println ( WiFi.localIP() );
    server.serveStatic("/dades", SPIFFS, "/dades"); //  gauge files dir
    server.begin();
    SPIFFS.remove("/dades/dat1.csv");
}

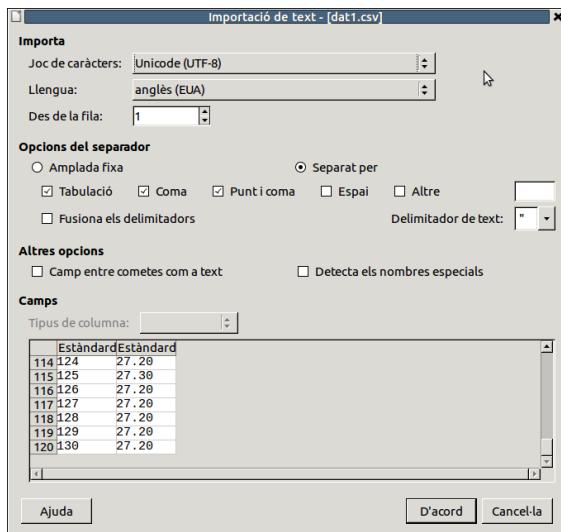
void loop() {
    File f = SPIFFS.open("/dades/dat1.csv","a");
    for(int i=1; i<=100; i++){
        ara = millis();
        if(dht12.get()==0){
            f.print(ara/1000);
            f.print(",");
            f.println(dht12.cTemp);
            espera(1000);
        }
    }
    f.close();
}
```

Fixeu-vos al programa com fem servir la funció *f.print()* per desar les dades a la SD virtual i com el *serveStatic()* ens permet accedir a elles via WiFi.

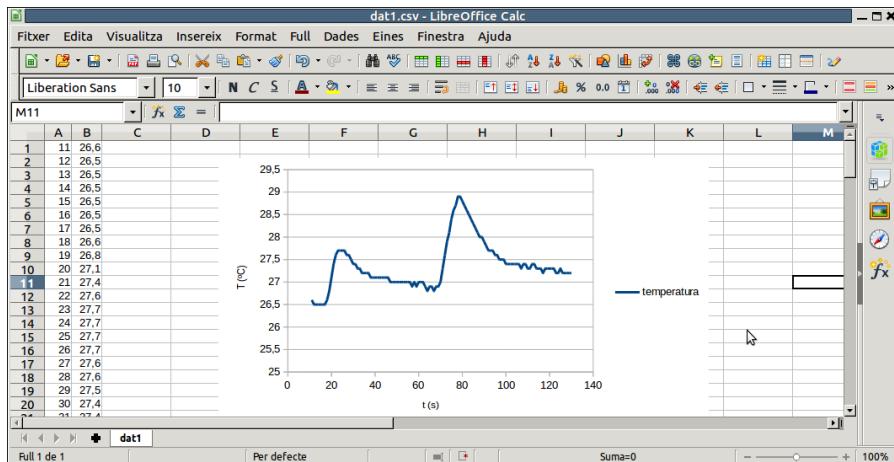
Una vegada enregistrades les dades podem baixar el fitxer amb un navegador apuntant a <http://192.168.1.37/dades/dat1.csv> (en el meu cas va agafar la IP 192.168.1.37).

Quan obrim el fitxer recomano triar idioma anglès (EUA) per interpretar correctament el separador decimal:

IoT amb D1 mini (ESP8266) i codi Arduino



I ja podem treballar amb les dades:



Llegint dades de la SD virtual

La SD virtual és el lloc ideal per desar dades com ara definicions de bitmaps o fragments de pàgines web. El problema és interpretar correctament aquestes dades.

El meu exemple *bitmapSPIFFS* carreguem la definició d'un bitmap per a la pantalla OLED. Com aquest l'hem obtingut mitjançant l'eina *image2cpp*¹⁰ la tenim en un array amb valor hexadecimals. Caldrà trobar cada element de la matriu i convertir el format de text hexadecimal:

```
#include <Wire.h>
#include <SFE_MicroOLED.h>
#include <FS.h>

#define PIN_RESET 255 // 
#define DC_JUMPER 0 // I2C Addres: 0 - 0x3C, 1 - 0x3D
MicroOLED oled(PIN_RESET, DC_JUMPER); // I2C Example
File f;
uint8_t pixels[16*24];

void setup()
{
    SPIFFS.begin();
    oled.begin();
    oled.clear(ALL);
    oled.display();
    delay(1000);
    oled.clear(PAGE);
    loadbitmap();
    oled.drawBitmap(pixels);
    oled.display();
}

void loop() {
}

String getLine() {
    String S = "" ;
    char c = f.read();
    while ( c != '\n') {
        S = S + c ;
        c = f.read();
    }
    return(S) ;
}

int convertFromHex(int ascii){
    if(ascii > 0x39) ascii -= 7; // adjust for hex letters upper or lower case
```

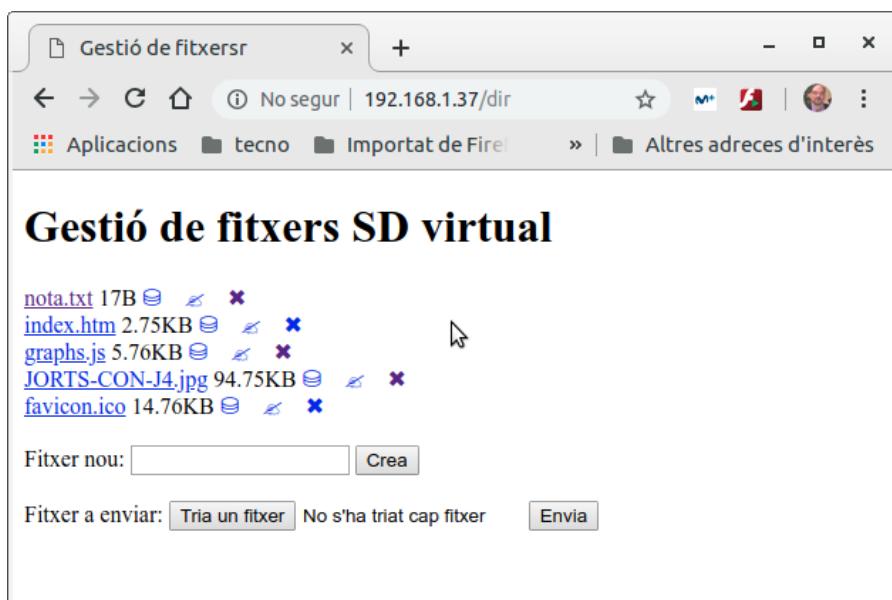
10 Disponible a <http://javl.github.io/image2cpp/>. Recordeu fer servir la codificació vertical i una mida de 64 x 48 px.

FSManager, la navalla suïssa per a la gestió de fitxers a la SD virtual via WiFi

Ja haureu vist que carregar la imatge de la SD virtual és lent. Especialment si estem modificant arxius d'estil CSS o pàgines HTML i codi javascript per millorar l'aspecte de la nostra interfície web.

El meu exemple *FSManager* es pot utilitzar com a una plantilla per als vostres programes. Es fàcil d'adaptar i millorar, i permet enviar, crear, editar, veure, baixar i esborrar tot tipus de fitxers. Això si, està limitat a gestionar un sol directori, que per defecte és l'arrel.

A més a més facilita la connexió WiFi i pot treballar tant en mode AP com STA. Només cal canviar el nom de xarxa i contrasenya, així com el tipus de connexió, si s'escau. No utilitza fitxers externs ni llibreries al núvol.



IoT amb D1 mini (ESP8266) i codi Arduino

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <FS.h>

#define DBG_OUTPUT_PORT Serial

const char* ssid = "esp8266AP";
const char* password = "robotica";
const char* host = "esp8266fs";
// tria el tipus de connexió. Client WIFI_STA, Access Point WIFI_AP
#define MODE_WIFI WIFI_AP

ESP8266WebServer server(80);
File fsUploadFile;

// Helper functions prototypes
void OkRetorn();
void printDirectory();
void handleFileCreate();
void handleFileDelete();
void handleFileUpload();
String formatBytes(size_t bytes);
String getContentType(String filename);
bool handleFileRead(String path);
void handleFileEdit();
void handleFileSave();
void initHelper();
void initWifi();
void pageHead();

void setup(void) {
    DBG_OUTPUT_PORT.begin(115200);
    SPIFFS.begin();

    initHelper(); //inclou connexió Wifi

    //get heap status, analog input value and all GPIO statuses in one json call
    server.on("/all", HTTP_GET, []() {
        String json = "{";
        json += "\"heap\":" + String(ESP.getFreeHeap());
        json += ", \"analog\":" + String(analogRead(A0));
        json += ", \"gpio\":" + String(uint32_t)((GPI | GPO) & 0xFFFF) | ((GP16I
& 0x01) << 16));
        json += "}";
        server.send(200, "text/json", json);
        json = String();
    });

    server.begin();
    DBG_OUTPUT_PORT.println("HTTP server started");
}

void loop(void) {
    server.handleClient();
}

//----- Helper functions
-----

void pageHead(){
    server.setContentLength(CONTENT_LENGTH_UNKNOWN);
    server.send(200, "text/html", "");
```

Part IV. El sistema de fitxers SPIFFS. Utilització a un servidor web.

```
server.sendContent("<!DOCTYPE html><html><head><title>Gesti&oacute; de  
fitxers</title><meta charset='UTF-8'><meta name='viewport'  
content='width=device-width'/></head><body>");  
}  
  
void initWifi(){  
    if (MODE_WIFI==WIFI_STA) {  
        WiFi.mode(WIFI_STA);  
        WiFi.begin(ssid, password);  
        DBG_OUTPUT_PORT.printf("Connecting to %s\n", ssid);  
        while (WiFi.status() != WL_CONNECTED) {  
            delay(500);  
            DBG_OUTPUT_PORT.print(".");  
        }  
        DBG_OUTPUT_PORT.println("");  
        DBG_OUTPUT_PORT.print("Connected! IP address: ");  
        DBG_OUTPUT_PORT.println(WiFi.localIP());  
  
        DBG_OUTPUT_PORT.print("Open http://");  
        DBG_OUTPUT_PORT.print(WiFi.localIP());  
        DBG_OUTPUT_PORT.println("/dir to see the file browser");  
    }  
    else {  
        WiFi.mode(WIFI_AP);  
        WiFi.softAP(ssid, password);  
        DBG_OUTPUT_PORT.print("Open http://192.168.4.1/dir to see the file  
browser");  
    }  
}  
  
void initHelper(){  
    DBG_OUTPUT_PORT.print("\n");  
    DBG_OUTPUT_PORT.setDebugOutput(true);  
    Dir dir = SPIFFS.openDir("/");  
    while (dir.next()) {  
        String fileName = dir.fileName();  
        size_t fileSize = dir.fileSize();  
        DBG_OUTPUT_PORT.printf("FS File: %s, size: %s\n", fileName.c_str(),  
formatBytes(fileSize).c_str());  
        DBG_OUTPUT_PORT.printf("\n");  
    }  
  
    initWifi();  
  
    //SERVER INIT  
    server.on("/dir", HTTP_GET, printDirectory);  
    server.on("/create", HTTP_GET, handleFileCreate);  
    server.on("/delete", HTTP_GET, handleFileDelete);  
    //first callback is called after the request has ended with all parsed  
arguments  
    //second callback handles file uploads at that location  
    server.on("/upload", HTTP_POST, []() {  
        OkReturn();  
    }, handleFileUpload);  
    server.on("/edit", HTTP_GET, handleFileEdit);  
    server.on("/save", HTTP_POST, handleFileSave);  
    //called when the url is not defined here  
    //use it to load content from SPIFFS  
    server.onNotFound([]() {  
        if (!handleFileRead(server.uri())) {  
            server.send(404, "text/plain", "FileNotFoundException");  
        }  
    });
```

IoT amb D1 mini (ESP8266) i codi Arduino

```
}

void OkReturn() {
    pageHead();
    server.sendContent("Operació realitzada amb èxit<br/><a href='/dir'>Tornar a gestió de fitxers</a></body></html>");
}

void printDirectory(){
    Dir dir = SPIFFS.openDir("/");
    pageHead();
    server.sendContent("<h1>Gestió de fitxers SD virtual</h1>");
    while (dir.next()) {
        String fileName = dir.fileName();
        size_t fileSize = dir.fileSize();
        DBG_OUTPUT_PORT.printf("FS File: %s, size: %s\n", fileName.c_str(),
formatBytes(fileSize).c_str());
        String output;
        output += "<a href='/";
        output += fileName.substring(1);
        output += '>"';
        output += fileName.substring(1);
        output += "</a> ";
        output += formatBytes(fileSize).c_str();
        output += " <a href='";
        output += fileName;
        output += "?download=1' style='text-decoration: none;'> &#9921;
</a>&nbsp;&nbsp;";
        output += " <a href='/edit?fe=";
        output += fileName;
        output += "' style='text-decoration: none;'> &#9997; </a>&nbsp;&nbsp;";
        output += " <a href='/delete?killfitxer=";
        output += fileName;
        output += "' style='text-decoration: none;'> &#10006; </a><br/>";
        server.sendContent(output);
        DBG_OUTPUT_PORT.println(output);
    }

    server.sendContent("<br/><form action='/create'>Fitxer nou: <input type='text' name='noufitxer' value='> <input type='submit' value='Crea'></form>");
    server.sendContent("<br/><form method='post' enctype='multipart/form-data' action='/upload'>Fitxer a enviar: <input type='file' name='myFile'><input type='submit' value='Envia'></form>");
    server.sendContent("</body></html>");
}

void handleFileCreate() {
    if (server.args() == 0) {
        return server.send(500, "text/plain", "BAD ARGS");
    }
    String path = "/" + server.arg("noufitxer");
    DBG_OUTPUT_PORT.println("handleFileCreate: " + path);
    if (path == "/") {
        return server.send(500, "text/plain", "BAD PATH");
    }
    if (SPIFFS.exists(path)) {
        return server.send(500, "text/plain", "FILE EXISTS");
    }
    File file = SPIFFS.open(path, "w");
    if (file) {
        file.close();
    } else {
```

Part IV. El sistema de fitxers SPIFFS. Utilització a un servidor web.

```
    return server.send(500, "text/plain", "CREATE FAILED");
}
OkRetorn();
path = String();
}

void handleFileDelete() {
    if (server.args() == 0) {
        return server.send(500, "text/plain", "BAD ARGS");
    }
    String path = server.arg("killfitxer");
    DBG_OUTPUT_PORT.println("handleFileDelete: " + path);
    if (path == "/") {
        return server.send(500, "text/plain", "BAD PATH");
    }
    if (!SPIFFS.exists(path)) {
        return server.send(404, "text/plain", "FileNotFoundException");
    }
    SPIFFS.remove(path);
    OkRetorn();
    path = String();
}

void handleFileUpload() {
    if (server.uri() != "/upload") {
        return;
    }
    HTTPUpload& upload = server.upload();
    if (upload.status == UPLOAD_FILE_START) {
        String filename = upload.filename;
        if (!filename.startsWith("/")) {
            filename = "/" + filename;
        }
        DBG_OUTPUT_PORT.print("handleFileUpload Name: ");
        DBG_OUTPUT_PORT.println(filename);
        fsUploadFile = SPIFFS.open(filename, "w");
        filename = String();
    } else if (upload.status == UPLOAD_FILE_WRITE) {
        //DBG_OUTPUT_PORT.print("handleFileUpload Data: ");
        DBG_OUTPUT_PORT.println(upload.currentSize);
        if (fsUploadFile) {
            fsUploadFile.write(upload.buf, upload.currentSize);
        }
    } else if (upload.status == UPLOAD_FILE_END) {
        if (fsUploadFile) {
            fsUploadFile.close();
        }
        DBG_OUTPUT_PORT.print("handleFileUpload Size: ");
        DBG_OUTPUT_PORT.println(upload.totalSize);
    }
}

String formatBytes(size_t bytes) {
    if (bytes < 1024) {
        return String(bytes) + "B";
    } else if (bytes < (1024 * 1024)) {
        return String(bytes / 1024.0) + "KB";
    } else if (bytes < (1024 * 1024 * 1024)) {
        return String(bytes / 1024.0 / 1024.0) + "MB";
    } else {
        return String(bytes / 1024.0 / 1024.0 / 1024.0) + "GB";
    }
}
```

IoT amb D1 mini (ESP8266) i codi Arduino

```
String getContentType(String filename) {
    if (server.hasArg("download")) {
        return "application/octet-stream";
    } else if (filename.endsWith(".htm")) {
        return "text/html";
    } else if (filename.endsWith(".html")) {
        return "text/html";
    } else if (filename.endsWith(".css")) {
        return "text/css";
    } else if (filename.endsWith(".js")) {
        return "application/javascript";
    } else if (filename.endsWith(".png")) {
        return "image/png";
    } else if (filename.endsWith(".gif")) {
        return "image/gif";
    } else if (filename.endsWith(".jpg")) {
        return "image/jpeg";
    } else if (filename.endsWith(".ico")) {
        return "image/x-icon";
    } else if (filename.endsWith(".xml")) {
        return "text/xml";
    } else if (filename.endsWith(".pdf")) {
        return "application/x-pdf";
    } else if (filename.endsWith(".zip")) {
        return "application/x-zip";
    } else if (filename.endsWith(".gz")) {
        return "application/x-gzip";
    }
    return "text/plain";
}

bool handleFileRead(String path) {
    DBG_OUTPUT_PORT.println("handleFileRead: " + path);
    if (!path.endsWith("/")) {
        path += "index.htm";
    }
    String contentType = getContentType(path);
    String pathWithGz = path + ".gz";
    if (SPIFFS.exists(pathWithGz) || SPIFFS.exists(path)) {
        if (SPIFFS.exists(pathWithGz)) {
            path += ".gz";
        }
        File file = SPIFFS.open(path, "r");
        server.streamFile(file, contentType);
        file.close();
        return true;
    }
    return false;
}

void handleFileEdit() {
    if (server.args() == 0) {
        return server.send(500, "text/plain", "BAD ARGS");
    }
    String path = server.arg("fe");
    DBG_OUTPUT_PORT.println("handleFileEdit: " + path);
    if (path == "/") {
        return server.send(500, "text/plain", "BAD PATH");
    }
    if (!SPIFFS.exists(path)) {
        DBG_OUTPUT_PORT.print("handleFileEdit Name: ");
        DBG_OUTPUT_PORT.print(path);DBG_OUTPUT_PORT.println(" FileNotFound");
        return server.send(404, "text/plain", "FileNotFound");
    }
}
```

Part IV. El sistema de fitxers SPIFFS. Utilització a un servidor web.

```
pageHead();
server.sendContent("<form action='/save' method='POST'>Fitxer: <input
type='text' name='nomfitxer' value=''");
server.sendContent(path);
server.sendContent("<br/><textarea name='cos' rows='30' cols='40'
wrap='off'>");
File file = SPIFFS.open(path,"r");
char buffer[2];
buffer[1] = 0;
// server.streamFile(file, "text/plain");
while (file.readBytes(buffer, 1)!=0 ) {
    server.sendContent(buffer);
}
file.close();
server.sendContent("</textarea><br><input type='submit'
value='Desa'></form></body></html>");
path = String();
}

void handleFileSave() {
String path = server.arg("nomfitxer");
String text = server.arg("cos");
DBG_OUTPUT_PORT.println("handleFileSave: " + path);
DBG_OUTPUT_PORT.println("text: " + text);
File file = SPIFFS.open(path,"w");
if (!file) {
    DBG_OUTPUT_PORT.println("file open failed");
}
file.print(text);
file.close();
OkRetorn();
path = String();
text = String();
}
```

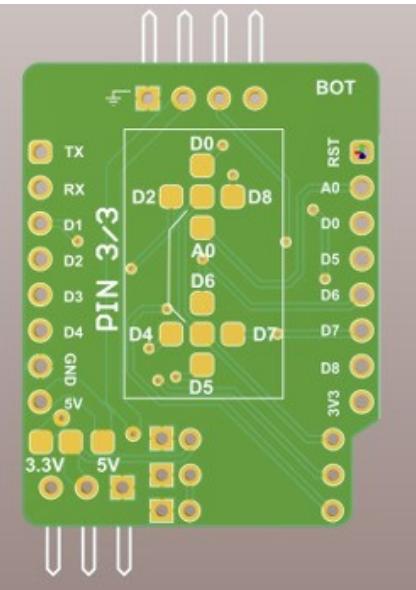
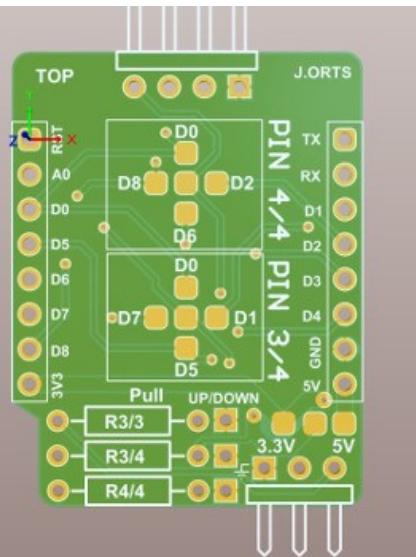
Part V. Altres shields D1 mini

Shield de connexions (JORTS)

A l'hora de passar del prototip de desenvolupament o viabilitat al prototip de comercialització un dels problemes que em vaig trobar al llarg del curs 2017-18 va ser la connexió de maquinari extern. La placa triple amb els seus pins mascle al 3r bloc facilita molt aquesta connexió al prototip de desenvolupament, en el qual els alumnes comproven la distribució correcta de pins i desenvolupen els programes de control.

Però al passar al prototip de comercialització, on els alumnes minimitzen el cost del producte i dissenyen una capsà amb la impressora 3D, l'única solució a l'abast era l'ús de plaques de prototips per posar els connectors al maquinari extern. Aquesta és una solució lenta i, de vegades, complicada, ja que cal dissenyar aquesta placa i soldar-la correctament.

Per tot això a l'estiu del 2018 el meu nebot Javier i jo vam dissenyar i enviar a fabricar a Xina un shield de connexions configurable.



IoT amb D1 mini (ESP8266) i codi Arduino

A la cara superior (TOP) podem configurar el connector J4 de 4 pins, i a la cara inferior (BOT) el J3 de 3 pins. Per a cada senyal podem, a la cara superior, afegir una resistència de pull-up o pull-down.

Connector J4 (TOP)

- pin 1: GND (indicat amb un quadrat)
- pin 2: VCC. Cal fer un pont entre 3.3V o 5V i el pad central per seleccionar aquest voltatge
- pin 3: podem escollir entre D0, D1, D5 o D7. Cal fer un pont entre el pad corresponent i el pad central
- pin 4: podem escolir entre D0, D2, D6 o D8. Cal fer un pont entre el pad corresponent i el pad central

A la cara superior (TOP) podem posar una resistència per a cada senyal si ens interessa fixar un valor per defecte a aquest pin.

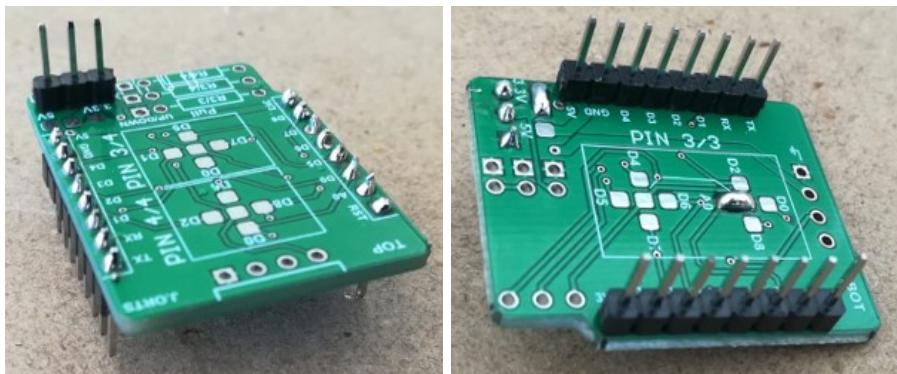
- Si la resistència arriba al forat indicat per un quadrat serà un pull-down (valor per defecte 0V)
- Si la resistència és una mica més curta i connecta al forat indicat per un cercle serà un pull-up (valor per defecte VCC per a aquest connector)

Connector J3 (BOT)

- pin 1: GND (indicat amb un quadrat)
- pin 2: VCC. Cal fer un pont entre 3.3V o 5V i el pad central per seleccionar aquest voltatge
- pin 3: podem escollir entre A0, D0, D2, D4, D5, D6, D7 o D8. Cal fer un pont entre el pad corresponent i el pad central

A continuació podeu veure un exemple real d'aplicació. Uns alumnes meus necessitaven per a un prototip de comercialització del seu treball de recerca un connector per a l'entrada analògica A0. Amb aquest shield el van tenir funcionant en 5 minuts:

Part V. Altres shields D1 mini



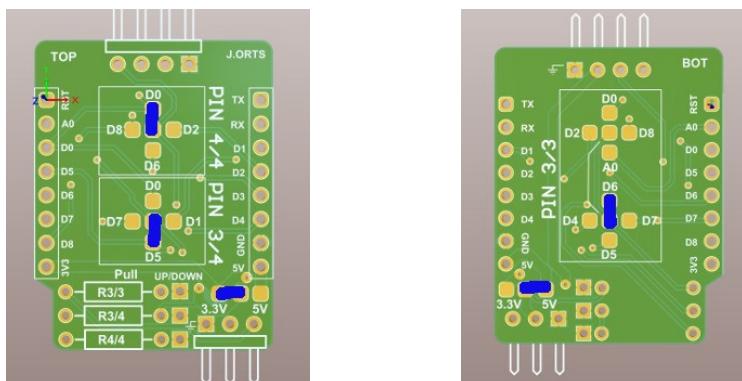
Pel desenvolupament de prototips comercials amb el nostre kit utilitzarem 2 configuracions típiques, CON1 i CON2.

Shield CON1

Caldrà fer alguns ponts a la placa, de forma que

- J4 té connexions a D5 (3/4) i D0 (4/4) amb VCC = 3,3V (2/4)
- J3 té connexió a D6 (3/3) i VCC = 5 V (2/3)

D'aquesta manera podem connectar un servo a J3 i mòduls amb entrades o sortides digitals a J4. S'han escollit D5 i D0 per a J4 per la seva baixa col·lisió amb altres shields en els projectes considerats.

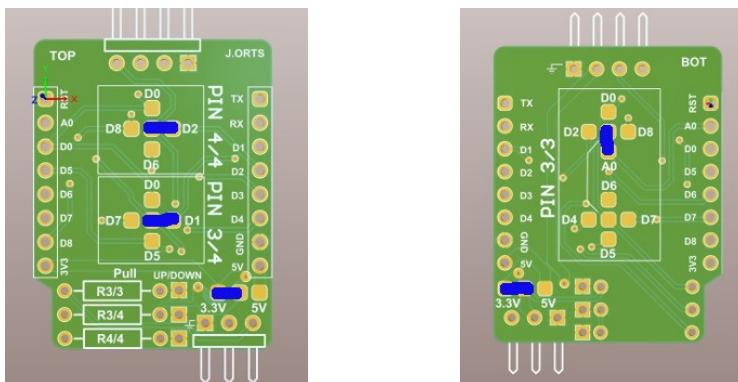


Shield CON2

Caldrà fer alguns ponts a la placa, de forma que

- J4 té connexions a D1 (3/4) i D2 (4/4) amb VCC = 3,3V (2/4)
- J3 té connexió a A0 (3/3) i VCC = 3,3 V (2/3)

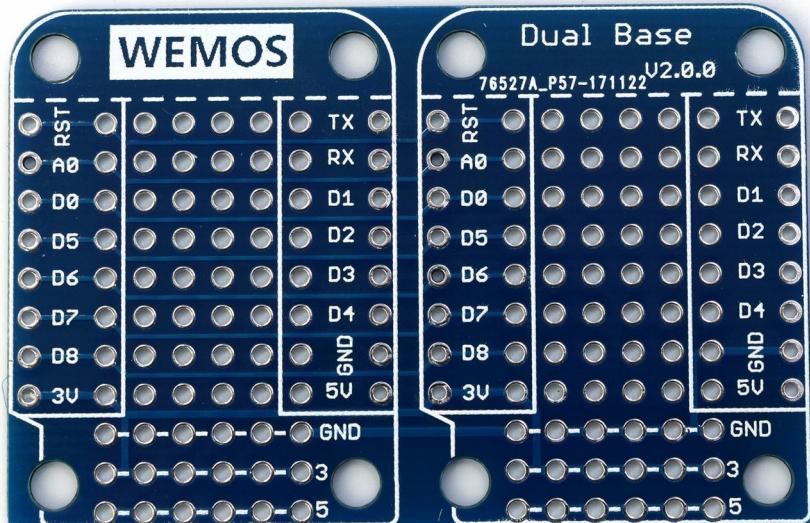
D'aquesta manera podem connectar mòduls I2C a J4 (SCL/D1 a 3/4 i SDA/D2 a 4/4, 3,3V a 2/4, GND a 1/4) i sensors analògics a J3 (A0 a 3/3, 3,3V a 2/3, GND a 1/3)



Dual base

Amb només 2 columnes de shields, aquesta base és molt útil per prototips comercials.

L'antiga versió només portava els connectors per a cada columna. L'actual (V2.0.0) és semblant a la triple base, amb línies de potència i zona de prototipatge.



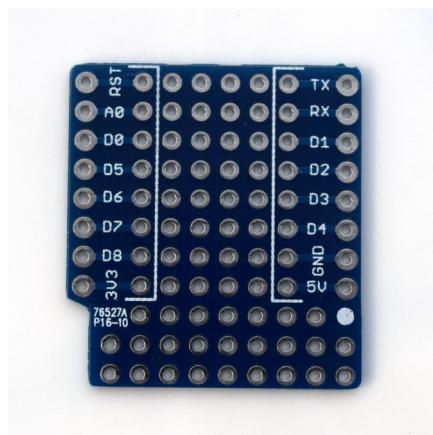
Amb la nova versió podem reproduir l'estrucció de la nostra 3a columna per a projectes amb nombrosos components externs, o incloure algun d'ells a la zona de prototipatge.

Protoboard shield

Molt útil per afegir altres components o connectors.

La primera i darrera columna de topos estan connectades a les respectives senyals del bus.

La versió antiga del kit (abril 2018) portava un semàfor de dos leds fet amb aquest shield.



WS2812B RGB Shield

Aquest mòdul porta un led RGB WS2812B, que fa servir un protocol d'un sol fil (connectat a D2).

Malauradament D2 s'utilitza típicament com SDA al bus I2C. Es a dir, si utilitzem aquest mòdul, no podrem fer servir cap shield amb I2C.

Aquest shield estava inclòs a la versió antiga del kit (abril 2018), i ha estat substituït pel RGB shield, amb 7 leds i connexió configurable.

Aquest shield el podem fer servir per a wearables, on podem canviar amb el fil el pin on va connectat i tenir-ne diversos leds connectats, o en aplicacions on la resta de mòduls no utilitzen I2C o només s'utilitza aquest shield. Un exemple d'aquest darrer cas és el meu exemple *peana*, on il·luminem amb diversos colors una escultura feta amb impressora 3D des de la base amb aquest shield:

```
//Install [Adafruit_NeoPixel_Library]
(https://github.com/adafruit/Adafruit_NeoPixel) first.

#include <Adafruit_NeoPixel.h>

#define PIN          D2

Adafruit_NeoPixel pixels = Adafruit_NeoPixel(1, PIN, NEO_BRG + NEO_KHZ800);

// colors RGB values from Viquipedia: https://ca.wikipedia.org/wiki/Colors_HTML

uint32_t color[]={
  0xFFC0CB, // 0 Pink
  0xFFB6C1, // 1 LightPink
  0xFF69B4, // 2 HotPink
  0xFF1493, // 3 DeepPink
  0xDB7093, // 4 PaleVioletRed
  0xC71585, // 5 MediumVioletRed
  0xFFA07A, // 6 LightSalmon
  0xFA8072, // 7 Salmon
  0xE9967A, // 8 DarkSalmon
```



IoT amb D1 mini (ESP8266) i codi Arduino

```
0xF08080, // 9 LightCoral
0xCD5C5C, // 10 IndianRed
0xDC143C, // 11 Crimson
0xB22222, // 12 FireBrick
0x8B0000, // 13 DarkRed
0xFF0000, // 14 Red
0xFF4500, // 15 OrangeRed
0xFF6347, // 16 Tomato
0xFF7F50, // 17 Coral
0xFF8C00, // 18 DarkOrange
0xFFA500, // 19 Orange
0xFFD700, // 20 Gold
0xFFFF00, // 21 Yellow
0xFFFFE0, // 22 LightYellow
0xFFFFACD, // 23 LemonChiffon
0xFAFAD2, // 24 LightGoldenrodYellow
0xFFEFD5, // 25 PapayaWhip
0xFFE4B5, // 26 Moccasin
0xFFDAB9, // 27 PeachPuff
0xEEE8AA, // 28 PaleGoldenrod
0xF0E68C, // 29 Khaki
0xBDDB76B, // 30 DarkKhaki
0xFFFF8DC, // 31 Cornsilk
0xFFEBBC, // 32 BlanchedAlmond
0FFE4C4, // 33 Bisque
0xFFDEAD, // 34 NavajoWhite
0xF5DEB3, // 35 Wheat
0xDEB887, // 36 BurlyWood
0xD2B48C, // 37 Tan
0xBC8F8F, // 38 RosyBrown
0xF4A460, // 39 SandyBrown
0xDA5220, // 40 Goldenrod
0xB8860B, // 41 DarkGoldenrod
0xCD853F, // 42 Peru
0xD2691E, // 43 Chocolate
0x884513, // 44 SaddleBrown
0xA0522D, // 45 Sienna
0xA52A2A, // 46 Brown
0x800000, // 47 Maroon
0x556B2F, // 48 DarkOliveGreen
0x808000, // 49 Olive
0x6B8E23, // 50 OliveDrab
0x9ACD32, // 51 YellowGreen
0x32CD32, // 52 LimeGreen
0x00FF00, // 53 Lime
0x7CFC00, // 54 LawnGreen
0x7FFF00, // 55 Chartreuse
0xADFF2F, // 56 GreenYellow
0x00FF7F, // 57 SpringGreen
0x00FA9A, // 58 MediumSpringGreen
0x90EE90, // 59 LightGreen
0x98FB98, // 60 PaleGreen
0x8FBC8F, // 61 DarkSeaGreen
0x3CB371, // 62 MediumSeaGreen
0x2E8B57, // 63 SeaGreen
0x228B22, // 64 ForestGreen
0x008000, // 65 Green
0x006400, // 66 DarkGreen
0x66CDAE, // 67 MediumAquamarine
0x00FFFF, // 68 Aqua
0x00FFFF, // 69 Cyan
0xE0FFFF, // 70 LightCyan
0xAFEEEE, // 71 PaleTurquoise
0x7FFF04, // 72 Aquamarine
```

Part V. Altres shields D1 mini

```
0x40E0D0, // 73 Turquoise
0x48D1CC, // 74 MediumTurquoise
0x00CED1, // 75 DarkTurquoise
0x20B2AA, // 76 LightSeaGreen
0x5F9EA0, // 77 CadetBlue
0x00888B, // 78 DarkCyan
0x008080, // 79 Teal
0xB0C4DE, // 80 LightSteelBlue
0xB0E0E6, // 81 PowderBlue
0xAD8E6, // 82 LightBlue
0x87CEEB, // 83 SkyBlue
0x87CEFA, // 84 LightSkyBlue
0x00BFFF, // 85 DeepSkyBlue
0x1E90FF, // 86 DodgerBlue
0x6495ED, // 87 CornflowerBlue
0x4682B4, // 88 SteelBlue
0x4169E1, // 89 RoyalBlue
0x0000FF, // 90 Blue
0x0000CD, // 91 MediumBlue
0x00008B, // 92 DarkBlue
0x000080, // 93 Navy
0x191970, // 94 MidnightBlue
0xE6E6FA, // 95 Lavender
0xD8BF08, // 96 Thistle
0xDDA0DD, // 97 Plum
0xEE82EE, // 98 Violet
0xDA70D6, // 99 Orchid
0xFF00FF, // 100 Magenta
0xBA55D3, // 101 MediumOrchid
0x9370DB, // 102 MediumPurple
0x8A2BE2, // 103 BlueViolet
0x9400D3, // 104 DarkViolet
0x9932CC, // 105 DarkOrchid
0x8B008B, // 106 DarkMagenta
0x800080, // 107 Purple
0x4B0082, // 108 Indigo
0x483D8B, // 109 DarkSlateBlue
0x6A5ACD, // 110 SlateBlue
0x7B68EE, // 111 MediumSlateBlue
0xFFFFF, // 112 White
0xF5F5DC, // 113 Beige
0xFAEBD7, // 114 AntiqueWhite
0xFFE4E1, // 115 MistyRose
0x808080, // 116 Gray
0x708090 // 117 SlateGray
};

int indexcolor;
uint32_t oldcolor, newcolor;

void setup() {
    pixels.begin(); // This initializes the NeoPixel library.
    indexcolor = random(118);
    oldcolor=color[indexcolor];
    pixels.setPixelColor(0, oldcolor);
    pixels.show(); // This sends the updated pixel color to the hardware.
    delay(400); // Delay for a period of time (in milliseconds).
}

void loop() {

    indexcolor = random(118);
    newcolor=color[indexcolor];
    while (oldcolor != newcolor) {
```

IoT amb D1 mini (ESP8266) i codi Arduino

```
uint32_t oldR = oldcolor & 0xFF0000;
uint32_t newR = newcolor & 0xFF0000;
uint32_t oldG = oldcolor & 0x00FF00;
uint32_t newG = newcolor & 0x00FF00;
uint32_t oldB = oldcolor & 0x0000FF;
uint32_t newB = newcolor & 0x0000FF;
if (newR > oldR) {
    oldcolor += 0x10000;
}
if (newR < oldR) {
    oldcolor -= 0x10000;
}
if (newG > oldG) {
    oldcolor += 0x100;
}
if (newG < oldG) {
    oldcolor -= 0x100;
}
if (newB > oldB) {
    oldcolor += 0x1;
}
if (newB < oldB) {
    oldcolor -= 0x1;
}

pixels.setPixelColor(0, oldcolor);
pixels.show(); // This sends the updated pixel color to the hardware.
delay(10); // Delay for a period of time (in milliseconds).
}
pixels.setPixelColor(0, oldcolor);
pixels.show(); // This sends the updated pixel color to the hardware.
delay(1000); // Delay for a period of time (in milliseconds).
}
```

Micro SD card shield

Aquest shield ens permet treballar amb SDs reals, de forma molt semblant a com hem utilitzat la SD virtuals del sistema de fitxers SPIFFS.

Si en fet meravelles amb els 2 MB de les SDs virtuals, imagineu-vos amb els GB de les SDs reals.

Malauradament la funció *serveStatic()* no funciona amb SDs reals. Haurem de crear codi per servir nosaltres els fitxers.

El meu exemple *SdserverAP* ens permet llistar els fitxers d'un directori i accedir a qualsevol fitxer de la SD. Fixeu-vos que tota la màgia del codi està en la funció *handleNotFound()*:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <SPI.h>
#include <SD.h>

#define DBG_OUTPUT_PORT Serial

const char* ssid = "ESPserver";
const char* password = "robotica";

ESP8266WebServer server(80);

static bool hasSD = false;
File uploadFile;

void returnOK() {
    server.send(200, "text/plain", "");
}

void returnFail(String msg) {
    server.send(500, "text/plain", msg + "\r\n");
}

bool loadFromSdCard(String path) {
    String dataType = "text/plain";
    if (path.endsWith("/")) {
        path += "index.htm";
    }
}
```



IoT amb D1 mini (ESP8266) i codi Arduino

```
}

if (path.endsWith(".src")) {
    path = path.substring(0, path.lastIndexOf("."));
} else if (path.endsWith(".htm")) {
    dataType = "text/html";
} else if (path.endsWith(".css")) {
    dataType = "text/css";
} else if (path.endsWith(".js")) {
    dataType = "application/javascript";
} else if (path.endsWith(".png")) {
    dataType = "image/png";
} else if (path.endsWith(".gif")) {
    dataType = "image/gif";
} else if (path.endsWith(".jpg")) {
    dataType = "image/jpeg";
} else if (path.endsWith(".ico")) {
    dataType = "image/x-icon";
} else if (path.endsWith(".xml")) {
    dataType = "text/xml";
} else if (path.endsWith(".pdf")) {
    dataType = "application/pdf";
} else if (path.endsWith(".zip")) {
    dataType = "application/zip";
}

File dataFile = SD.open(path.c_str());
if (dataFile.isDirectory()) {
    path += "/index.htm";
    dataType = "text/html";
    dataFile = SD.open(path.c_str());
}

if (!dataFile) {
    return false;
}

if (server.hasArg("download")) {
    dataType = "application/octet-stream";
}

if (server.streamFile(dataFile, dataType) != dataFile.size()) {
    DBG_OUTPUT_PORT.println("Sent less data than expected!");
}

dataFile.close();
return true;
}

void handleNotFound() {
    if (hasSD && loadFromSdCard(server.uri())) {
        return;
    }
    String message = "SDCARD Not Detected\n\n";
    message += "URI: ";
    message += server.uri();
    message += "\nMethod: ";
    message += (server.method() == HTTP_GET) ? "GET" : "POST";
    message += "\nArguments: ";
    message += server.args();
    message += "\n";
    for (uint8_t i = 0; i < server.args(); i++) {
        message += " NAME:" + server.argName(i) + "\n VALUE:" + server.arg(i) +
    "\n";
}
```

Part V. Altres shields D1 mini

```
}

server.send(404, "text/plain", message);
DBG_OUTPUT_PORT.print(message);
}

void printDirectory(){
    File dir = SD.open("/files");
    dir.rewindDirectory();
    server.setContentLength(CONTENT_LENGTH_UNKNOWN);
    server.send(200, "text/html", "");
    server.sendContent("<html><head></head><body>");
    while(true) {
        File entry = dir.openNextFile();
        if (! entry) {
            // no more files
            break;
        }
        String output;
        output += "<a href='/files/";
        output += entry.name();
        output += "'>";
        output += entry.name();
        output += "</a><br/>";
        server.sendContent(output);
        entry.close();
    }
    server.sendContent("</body></html>");
    dir.close();
}

void setup(void) {
    DBG_OUTPUT_PORT.begin(115200);
    DBG_OUTPUT_PORT.setDebugOutput(true);
    DBG_OUTPUT_PORT.print("\n");
    WiFi.mode(WIFI_AP);
    WiFi.softAP(ssid, password);

    server.on("/list", HTTP_GET, printDirectory);
    server.onNotFound(handleNotFound);

    server.begin();
    DBG_OUTPUT_PORT.println("HTTP server started");

    if (SD.begin(SS)) {
        DBG_OUTPUT_PORT.println("SD Card initialized.");
        hasSD = true;
    }
}

void loop(void) {
    server.handleClient();
}
```

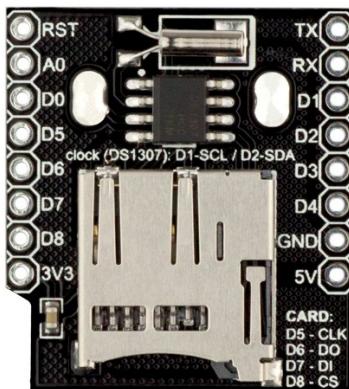
Datalogger shield

Aquest mòdul, comercialitzat pel fabricant RobotDyn, combina el lector microSD amb el RTC, facilitant molt el disseny de dataloggers.

Encara que inclòs al kit inicial d'abril de 2018, es va substituir pel RTC shield. El problema és l'alt nombre de pins utilitzats. A més a més del bus I2C (D1 i D2) pel RTC, utilitza quatre pins més (D5, D6, D7 i D8) per la SD. Això podria provocar col·lisions amb altres mòduls del kit.

És a dir, només ens queden lliures els pins D3 i D4, A0, i el bus I2C.

Per tant és un mòdul interessant quan fem servir sensors I2C, o ampliem el nombre d'entrades amb un dispositiu I2C (com l'arduino pro mini inclòs al kit).



A l'exemple *SDdatalogger* fem servir aquest shield per enregistrar les dades de temperatura d'un shield DHT juntament amb l'hora. Mitjançant un servidor web que s'activa amb el polsador del 1-button shield. Cal tenir paciència: potser caldrà esperar més de 30 s polsant el botó per engegar el servidor!

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <SD.h>
#include <WEMOS_DHT12.h>
#include <Wire.h>
```

Part V. Altres shields D1 mini

```
#include "RTClib.h"

RTC_DS1307 RTC;
DateTime now;
DHT12 dht12;
ESP8266WebServer server ( 80 );
unsigned long ara;

const char *ssid = "jortsnet";
const char *password = "9periodico";

void espera (int n) {
    while (millis()<ara+n) {
        delay(1);
    }
}

void sendFile() {

    File dataFile = SD.open("/dat1.csv");
    server.streamFile(dataFile, "application/octet-stream");
    dataFile.close();
}

void setup() {
    pinMode(D3,INPUT);
    pinMode(D4,OUTPUT);
    digitalWrite(D4,HIGH);
    Serial.begin(9600);
    SD.begin();
    SD.remove("/dat1.csv");
    Wire.begin();
    RTC.begin();
    if (! RTC.isrunning()) {
        Serial.println("RTC is NOT running!");
        RTC.adjust(DateTime(__DATE__, __TIME__));
    }
}

void loop() {
    if (digitalRead(D3)==true){
        File f = SD.open("/dat1.csv",FILE_WRITE);
        for(int i=1; i<=30; i++){
            ara = millis();
            now = RTC.now();
            if(dht12.get()==0){
                f.print(ara/1000);
                f.print(",");
                f.print(now.day());
                f.print("/");
                f.print(now.month());
                f.print("/");
                f.print(now.year()%2000);
                f.print(" ");
                f.print(now.hour());
                f.print(":");
                f.print(now.minute());
                f.print(":");
                f.print(now.second());
                f.print(",");
                f.println(dht12.cTemp);
                espera(1000);
            }
        }
    }
}
```

IoT amb D1 mini (ESP8266) i codi Arduino

```
f.close();
}
else {
    digitalWrite(D4,LOW);
    WiFi.begin ( ssid, password );
    while ( WiFi.status() != WL_CONNECTED ) {
        delay ( 500 );
    }
    Serial.println ( "" );
    Serial.print ( "Connected to " );
    Serial.println ( ssid );
    Serial.print ( "IP address: " );
    Serial.println ( WiFi.localIP() );
    server.on("/dat1.csv", HTTP_GET, sendFile);
    server.begin();
    while(true){
        server.handleClient();
    }
}
```

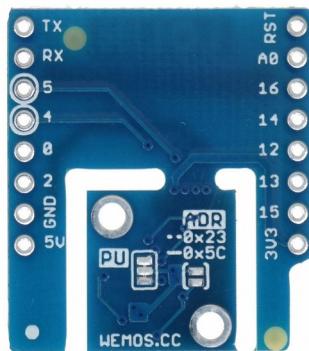
Ambient light Shield (BH1750)

Aquest shield és equivalent al nostre Mòdul sensor de llum I2C BH1750FVI. Podem fer servir directament el codi d'aquell mòdul.

Quan es tracta de mesurar la llum cal tenir molt clar com col·loquem el sensor. El sensor en si es pot separar del shield, però ens caldrà un altre shield (TFT I2C Connector Shield) per connectar-lo, ja que el connector que porta és especial. Tan especial que encara no he trobat un conjunt de connectors en bus per utilitzar una única sortida per diversos dispositius I2C.

M'estimo més fer servir el nostre mòdul: connexió a la triple base o al shield CON2 amb cables Dupont estàndard, ja està separat i en puc fer un bus de connexions I2C amb una placa de pistes i unes tires de pins.

De totes formes és interessant conèixer l'existència d'aquest shield, per si us va bé per algun projecte.



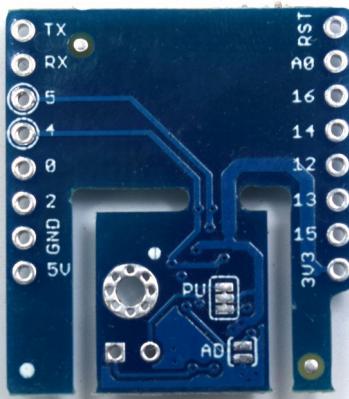
SHT30 Shield

Aquest shield és semblant al nostre DHT shield, però amb mesures més precises: $\pm 3\%$ per la humitat relativa i $\pm 0,3\text{ }^{\circ}\text{C}$ i amb una resolució de $\pm 0,05\text{ }^{\circ}\text{C}$ amb un rang entre $-40\text{ }^{\circ}\text{C}$ i $+125\text{ }^{\circ}\text{C}$. També funciona per I2C, i porta una sortida d'alarma programable.

El seu ús és tan senzill com el del DHT gràcies a la llibreria *WEMOS_SHT3x*¹¹.

De fet, és una bona alternativa al DHT pel kit, i més potent.

A l'exemple *SHT30_OLED* mostrem al display la temperatura i la humitat cada segon:



```
#include <Wire.h>
#include <WEMOS_SHT3X.h>
#include <SFE_MicroOLED.h> // Include the SFE_MicroOLED library
#define PIN_RESET 255 // Connect RST to pin 9
#define DC_JUMPER 0

MicroOLED oled(PIN_RESET, DC_JUMPER); // I2C declaration
SHT3X sht30(0x45);

void setup() {
    oled.begin(); // Initialize the OLED
```

11 Disponible a https://github.com/wemos/WEMOS_SHT3x_Arduino_Library

Part V. Altres shields D1 mini

```
oled.clear(ALL); // Clear the display's internal memory
oled.display(); // Display what's in the buffer (splashscreen)
delay(1000); // Delay 1000 ms
oled.clear(PAGE); // Clear the buffer.

Serial.begin(115200);
}

void loop() {
    sht30.get();
    oled.clear(PAGE);
    oled.setFontType(1);
    oled.setCursor(0,0);
    oled.print("T:");
    oled.print(sht30.cTemp);
    oled.setCursor(0,24);
    oled.print("H:");
    oled.print(sht30.humidity);
    oled.display();
    delay(1000);
}
```

Barometric Pressure Shield

Aquest shield I2C porta un sensor de pressió i temperatura HP303B.

Pressió

Rang: 300 : 1200 hPa

Precisió: $\pm 0,005$ hPa

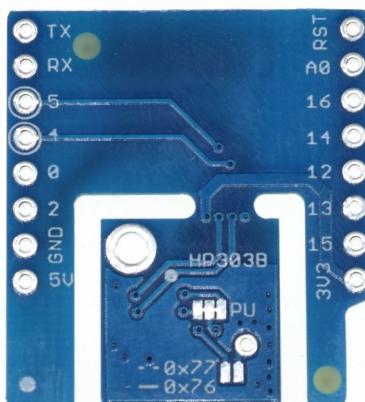
Exactitud: ± 1 hPa

Temperatura

Rang: -40 °C : +85 °C

Exactitud: $\pm 0,5$ °C

Com veieu, una sonda perfecta per a una estació meteorològica de butxaca.



```
#include <LOLIN_HP303B.h>
LOLIN_HP303B HP303BPressureSensor;

void setup()
{
    Serial.begin(9600);
    HP303BPressureSensor.begin();
    Serial.println("Barometric test");
}
```

Part V. Altres shields D1 mini

```
void loop()
{
    int32_t temperature;
    int32_t pressure;
    int16_t oversampling = 7;
    int16_t ret;
    ret = HP303BPressureSensor.measureTempOnce(temperature, oversampling);
    Serial.print("T=");
    Serial.print(temperature);
    Serial.print(" °C, ");
    ret = HP303BPressureSensor.measurePressureOnce(pressure, oversampling);
    Serial.print("P=");
    Serial.print(pressure);
    Serial.println(" Pa");
    delay(500);
}
```

TFT 1.4 Shield

Aquest shield porta una pantalla TFT color de 128x128 px, 1,4". Utilitza un protocol SPI, amb els pins D3, D4, D5 i D7 per defecte. També afecta al pin D6. Amb els seus 262,144 colors aquesta pantalla és ideal per mostrar gràfiques, imatges ... I al mateix preu que el nostre display OLED monocrom!

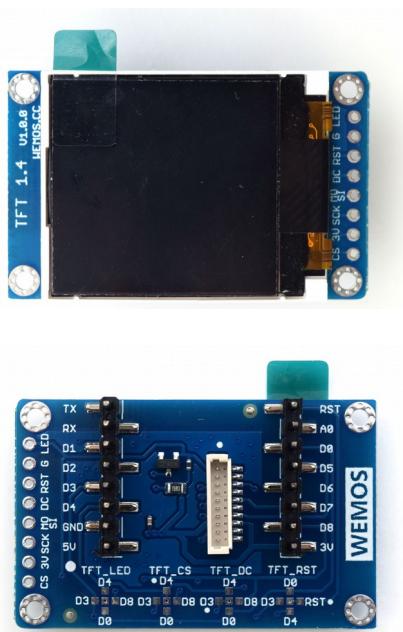
És més ampla que una columna, amb la qual cosa haurem de veure com distribuir els shields per alçades si utilitzem una base. Aquest ha estat el motiu, juntament amb el nombre de pins utilitzats, pel que vaig escollir l'OLED pel nostre kit.

Es pot connectar de 3 formes diferents:

- a una columna d'una base
- amb una tira de pins
- amb un connector específic TFT al TFT I2C Connector Shield

Utilitza les llibreries *Adafruit_GFX* i *Adafruit_ST7735*, que podeu instal·lar amb el gestor.

Com veieu a la imatge alguns pins es poden canviar. Fins i tot controlar més coses (RST independent, LED), incrementant la despesa de pins.



Part V. Altres shields D1 mini

Cal dir que els pins D5 (SCK), D6 (MISO) i D7 (MOSI) formen la part fixa del bus SPI, i per tant es poden compartir amb altres dispositius SPI, com ara el datalogger shield. Amb D4 (TFT_CS) el bus accedeix a la pantalla TFT, i amb D8 (CS) a la SD. En canvi, com el microSD shield també fa servir D4 (TF-CS) per accedir a la SD, caldria canviar el CS per defecte d'un dels dos shields per utilitzar-los a la vegada.

L'exemple *TFT_1.4_Bitmap* , adaptat de la web d'Adafruit¹² pel nostre shield, carrega una imatge de la SD virtual i la mostra en pantalla:



```
#include <Adafruit_GFX.h>      // Core graphics library
#include <Adafruit_ST7735.h> // Hardware-specific library
#include <SPI.h>
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <FS.h>

#define TFT_RST -1 //for TFT I2C Connector Shield V1.0.0 and TFT 1.4 Shield V1.0.0
#define TFT_CS D4 //for TFT I2C Connector Shield V1.0.0 and TFT 1.4 Shield V1.0.0
#define TFT_DC D3 //for TFT I2C Connector Shield V1.0.0 and TFT 1.4 Shield V1.0.0

// #define TFT_RST -1    //for TFT I2C Connector Shield (V1.1.0 or later)
// #define TFT_CS D0    //for TFT I2C Connector Shield (V1.1.0 or later)
// #define TFT_DC D8    //for TFT I2C Connector Shield (V1.1.0 or later)

Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST);

void setup(void)
```

12 Veure <https://learn.adafruit.com/adafruit-1-44-color-tft-with-micro-sd-socket/drawing-bitmaps>

IoT amb D1 mini (ESP8266) i codi Arduino

```
{  
    SPIFFS.begin();  
    tft.initR(INITR_144GREENTAB);  
    tft.setTextWrap(false); // Allow text to run off right edge  
    tft.fillScreen(ST7735_BLACK);  
    Serial.begin(9600);  
}  
  
void loop(void)  
{  
    tft.fillScreen(ST7735_BLACK);  
    // change the name here!  
    bmpDraw("lily128.bmp", 0, 0);  
    // wait 5 seconds  
    delay(5000);  
}  
  
// This function opens a Windows Bitmap (BMP) file and  
// displays it at the given coordinates. It's sped up  
// by reading many pixels worth of data at a time  
// (rather than pixel by pixel). Increasing the buffer  
// size takes more of the Arduino's precious RAM but  
// makes loading a little faster. 20 pixels seems a  
// good balance.  
#define BUFFPIXEL 20  
  
void bmpDraw(char *filename, uint8_t x, uint16_t y) {  
    File bmpfile;  
    int bmpWidth, bmpHeight; // W+H in pixels  
    uint8_t bmpDepth; // Bit depth (currently must be 24)  
    uint32_t bmpImageoffset; // Start of image data in file  
    uint32_t rowSize; // Not always = bmpWidth; may have padding  
    uint8_t sdbuf[3*BUFFPIXEL]; // pixel buffer (R+G+B per pixel)  
    uint8_t buffidx = sizeof(sdbuf); // Current position in sdbuf  
    boolean goodBmp = false; // Set to true on valid header parse  
    boolean flip = true; // BMP is stored bottom-to-top  
    int w, h, row, col;  
    uint8_t r, g, b;  
    uint32_t pos = 0, startTime = millis();  
    if((x >= tft.width()) || (y >= tft.height())) return;  
    Serial.println();  
    Serial.print(F("Loading image "));  
    Serial.print(filename);  
    Serial.println('\\');  
    // Open requested file on SD card  
    if ((bmpFile = SPIFFS.open("/lily128.bmp","r")) == NULL) {  
        Serial.print(F("File not found"));  
        return;  
    }  
    // Parse BMP header  
    if(read16(bmpFile) == 0x4D42) { // BMP signature  
        Serial.print(F("File size: ")); Serial.println(read32(bmpFile));  
        (void)read32(bmpFile); // Read & ignore creator bytes  
        bmpImageoffset = read32(bmpFile); // Start of image data  
        Serial.print(F("Image Offset: ")); Serial.println(bmpImageoffset, DEC);  
        // Read DIB header  
        Serial.print(F("Header size: ")); Serial.println(read32(bmpFile));  
        bmpWidth = read32(bmpFile);  
        bmpHeight = read32(bmpFile);  
        if(read16(bmpFile) == 1) { // # planes -- must be '1'  
            bmpDepth = read16(bmpFile); // bits per pixel  
            Serial.print(F("Bit Depth: ")); Serial.println(bmpDepth);  
            if((bmpDepth == 24) && (read32(bmpFile) == 0)) { // 0 = uncompressed  
                goodBmp = true; // Supported BMP format -- proceed!  
            }  
        }  
    }  
}
```

Part V. Altres shields D1 mini

```
Serial.print(F("Image size: "));  
Serial.print(bmpWidth);  
Serial.print('x');  
Serial.println(bmpHeight);  
// BMP rows are padded (if needed) to 4-byte boundary  
rowSize = (bmpWidth * 3 + 3) & ~3;  
// If bmpHeight is negative, image is in top-down order.  
// This is not canon but has been observed in the wild.  
if(bmpHeight < 0) {  
    bmpHeight = -bmpHeight;  
    flip      = false;  
}  
// Crop area to be loaded  
w = bmpWidth;  
h = bmpHeight;  
if((x+w-1) >= tft.width()) w = tft.width() - x;  
if((y+h-1) >= tft.height()) h = tft.height() - y;  
// Set TFT address window to clipped image bounds  
tft.startWrite();  
tft.setAddrWindow(x, y, w, h);  
for (row=0; row<h; row++) { // For each scanline...  
    // Seek to start of scan line. It might seem labor-  
    // intensive to be doing this on every line, but this  
    // method covers a lot of gritty details like cropping  
    // and scanline padding. Also, the seek only takes  
    // place if the file position actually needs to change  
    // (avoids a lot of cluster math in SD library).  
    if(flip) // Bitmap is stored bottom-to-top order (normal BMP)  
        pos = bmpImageOffset + (bmpHeight - 1 - row) * rowSize;  
    else    // Bitmap is stored top-to-bottom  
        pos = bmpImageOffset + row * rowSize;  
    if(bmpFile.position() != pos) { // Need seek?  
        tft.endWrite();  
        bmpFile.seek(pos);  
        buffidx = sizeof(sdbuf); // Force buffer reload  
    }  
    for (col=0; col<w; col++) { // For each pixel...  
        // Time to read more pixel data?  
        if (buffidx >= sizeof(sdbuf)) { // Indeed  
            bmpFile.read(sdbuf, sizeof(sdbuf));  
            buffidx = 0; // Set index to beginning  
            tft.startWrite();  
        }  
        // Convert pixel from BMP to TFT format, push to display  
        b = sdbuf[buffidx++];  
        g = sdbuf[buffidx++];  
        r = sdbuf[buffidx++];  
        tft.pushColor(tft.color565(r,g,b));  
    } // end pixel  
} // end scanline  
tft.endWrite();  
Serial.print(F("Loaded in_ "));  
Serial.print(millis() - startTime);  
Serial.println(" ms");  
} // end goodBmp  
}  
}  
}  
}
```

// These read 16- and 32-bit types from the SD card file.
// BMP data is stored little-endian, Arduino is little-endian too.

IoT amb D1 mini (ESP8266) i codi Arduino

```
// May need to reverse subscript order if porting elsewhere.

uint16_t read16(File f) {
    uint16_t result;
    ((uint8_t *)&result)[0] = f.read(); // LSB
    ((uint8_t *)&result)[1] = f.read(); // MSB
    return result;
}

uint32_t read32(File f) {
    uint32_t result;
    ((uint8_t *)&result)[0] = f.read(); // LSB
    ((uint8_t *)&result)[1] = f.read();
    ((uint8_t *)&result)[2] = f.read();
    ((uint8_t *)&result)[3] = f.read(); // MSB
    return result;
}
```

TFT 2.4 Touch Shield

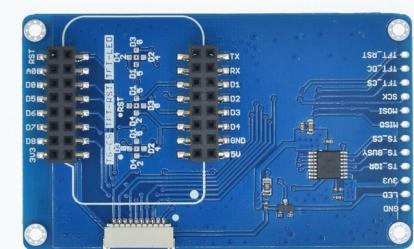
Aquest shield ofereix una pantalla 320x240 TFT de 2,4" sensible al tacte.

Utilitza les biblioteques *Adafruit_GFX*, *Adafruit_ILI9341* i *XPT2046_Touchscreen*.

La connexió del bus de columna es femella. També es pot connectar amb una tira de pins o amb el TFT I2C Connector Shield.

Utilitza els pins D0, D3, D5, D6, D7 i D8, amb un bus SPI.

Especialment interessant per a menús interactius, nsimulacions de joystick, signatures ...



L'exemple *TFT2p4_touchpaint*, adaptat de la web d'Adafruit¹³ per a aquest shield, permet dibuixar en la pantalla amb el dit:



¹³ Veure <https://learn.adafruit.com/adafruit-2-4-tft-touch-screen-featherwing/resistive-touch-screen>

IoT amb D1 mini (ESP8266) i codi Arduino

```
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_ILI9341.h>
#include <XPT2046_Touchscreen.h>

#define TFT_CS D0 //for D1 mini or TFT I2C Connector Shield (V1.1.0 or later)
#define TFT_DC D8 //for D1 mini or TFT I2C Connector Shield (V1.1.0 or later)
#define TFT_RST -1 //for D1 mini or TFT I2C Connector Shield (V1.1.0 or later)
#define TS_CS D3 //for D1 mini or TFT I2C Connector Shield (V1.1.0 or later)

Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_RST);
XPT2046_Touchscreen ts(TS_CS);

// This is calibration data for the raw touch data to the screen coordinates
#define TS_MINX 3800
#define TS_MAXX 100
#define TS_MINY 100
#define TS_MAXY 3750

// Size of the color selection boxes and the paintbrush size
#define BOXSIZE 40
#define PENRADIUS 3
int oldcolor, currentcolor;

void setup(void) {
    Serial.begin(115200);
    delay(10);
    Serial.println("FeatherWing TFT");
    if (!ts.begin()) {
        Serial.println("Couldn't start touchscreen controller");
        while (1);
    }
    ts.setRotation(4);
    Serial.println("Touchscreen started");
    tft.begin();
    tft.setRotation(2);
    tft.fillScreen(ILI9341_BLACK);

    // make the color selection boxes
    tft.fillRect(0, 0, BOXSIZE, BOXSIZE, ILI9341_RED);
    tft.fillRect(BOXSIZE, 0, BOXSIZE, BOXSIZE, ILI9341_YELLOW);
    tft.fillRect(BOXSIZE*2, 0, BOXSIZE, BOXSIZE, ILI9341_GREEN);
    tft.fillRect(BOXSIZE*3, 0, BOXSIZE, BOXSIZE, ILI9341_CYAN);
    tft.fillRect(BOXSIZE*4, 0, BOXSIZE, BOXSIZE, ILI9341_BLUE);
    tft.fillRect(BOXSIZE*5, 0, BOXSIZE, BOXSIZE, ILI9341_MAGENTA);

    // select the current color 'red'
    tft.drawRect(0, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);
    currentcolor = ILI9341_RED;
}

void loop() {
    // Retrieve a point
    TS_Point p = ts.getPoint();
    Serial.print("X = "); Serial.print(p.x);
    Serial.print("\tY = "); Serial.print(p.y);
    Serial.print("\tPressure = "); Serial.println(p.z);

    // Scale from ~0->4000 to tft.width using the calibration #'s
    p.x = map(p.x, TS_MINX, TS_MAXX, 0, tft.width());
    p.y = map(p.y, TS_MINY, TS_MAXY, 0, tft.height());

    if (p.y < BOXSIZE) {
        oldcolor = currentcolor;
```

Part V. Altres shields D1 mini

```
if (p.x < BOXSIZE) {  
    currentcolor = ILI9341_RED;  
    tft.drawRect(0, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);  
} else if (p.x < BOXSIZE*2) {  
    currentcolor = ILI9341_YELLOW;  
    tft.drawRect(BOXSIZE, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);  
} else if (p.x < BOXSIZE*3) {  
    currentcolor = ILI9341_GREEN;  
    tft.drawRect(BOXSIZE*2, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);  
} else if (p.x < BOXSIZE*4) {  
    currentcolor = ILI9341_CYAN;  
    tft.drawRect(BOXSIZE*3, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);  
} else if (p.x < BOXSIZE*5) {  
    currentcolor = ILI9341_BLUE;  
    tft.drawRect(BOXSIZE*4, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);  
} else if (p.x < BOXSIZE*6) {  
    currentcolor = ILI9341_MAGENTA;  
    tft.drawRect(BOXSIZE*5, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);  
}  
  
if (oldcolor != currentcolor) {  
    if (oldcolor == ILI9341_RED)  
        tft.fillRect(0, 0, BOXSIZE, BOXSIZE, ILI9341_RED);  
    if (oldcolor == ILI9341_YELLOW)  
        tft.fillRect(BOXSIZE, 0, BOXSIZE, BOXSIZE, ILI9341_YELLOW);  
    if (oldcolor == ILI9341_GREEN)  
        tft.fillRect(BOXSIZE*2, 0, BOXSIZE, BOXSIZE, ILI9341_GREEN);  
    if (oldcolor == ILI9341_CYAN)  
        tft.fillRect(BOXSIZE*3, 0, BOXSIZE, BOXSIZE, ILI9341_CYAN);  
    if (oldcolor == ILI9341_BLUE)  
        tft.fillRect(BOXSIZE*4, 0, BOXSIZE, BOXSIZE, ILI9341_BLUE);  
    if (oldcolor == ILI9341_MAGENTA)  
        tft.fillRect(BOXSIZE*5, 0, BOXSIZE, BOXSIZE, ILI9341_MAGENTA);  
}  
}  
  
if (((p.y-PENRADIUS) > 0) && ((p.y+PENRADIUS) < tft.height())) {  
    tft.fillCircle(p.x, p.y, PENRADIUS, currentcolor);  
}  
}
```

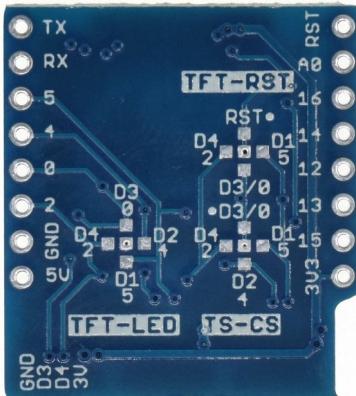
TFT I2C Connector Shield

Aquest shield porta uns connectors especials pels shields TFT i pels sensors I2C.

Especialment útil quan el display TFT el volem separar de la base del D1 mini.

Aneu amb compte amb la versió del connector. Han canviats coses. La V1.0.0, pensada amb el TFT 1.4 Shield, portava un selector per TFT_CS, per defecte connectat a D4. La versió actual V1.1.0, pensada amb el TFT 2.4 Touch Shield, no porta aquest selector, TFT_CS va connectat ara sempre a D0. El mateix passa amb TFT_DC. Abans portava un selector, per defecte a D3. Ara va fixe a D8.

A més a més la V1.1.0 porta un nou connector I/O amb D3 i D4. Aquesta darrera connexió amb D3 no té sentit si no s'utilitza el TFT 1.4 Shield, que no té sensor tàctil.



Motor shield

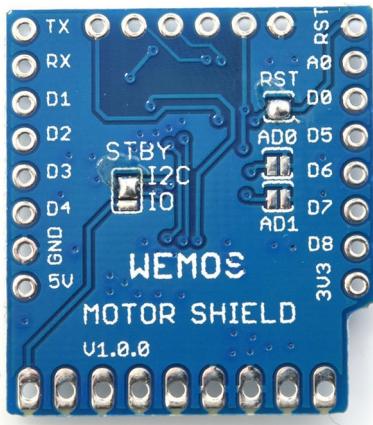
Aquest shield porta un pont H controlat pel seu propi microcontroladors al que s'accedeix amb I2C.

D'aquesta manera podem controlar 2 motors fàcilment. Podem aturar-los o engegar-los, canviar la velocitat, el sentit de gir, frenar-los o deixar-los en stand-by de forma independent

Utilitza la llibreria pròpia WEMOS_Motor_Shield¹⁴, i el firmware¹⁵ es pot modificar.

Pot donar una intensitat promig de 1,2 A i 3,2 A de pic, amb una alimentació independent pels motors de fins a 15 V.

Personalment, per a aplicacions de tipus robots m'estimo més utilitzar Els mòduls ESP12 V2. La placa NODEMCU motor shield . Aquesta altra placa porta interruptor i regulador de tensió pel mòdul ESP8266, així com connectors per a sensors externs.



14 Disponible a https://github.com/wemos/WEMOS_Motor_Shield_Arduino_Library

15 Disponible a https://github.com/wemos/Motor_Shield_Firmware

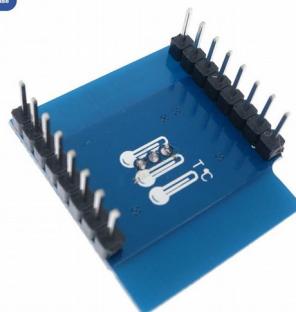
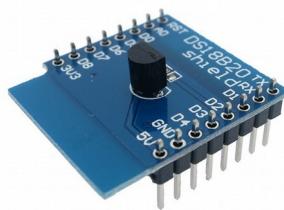
DS18B20 shield

Aquest shield porta un sensor de temperatura DS18B20, essent equivalent al nostre Mòdul sensor temperatura DS18B20 .

El problema és que va connectat al pin D2, inutilitzant el bus I2C. A més a més, no porta connector per altres sensors al mateix bus OneWire.

Pot ser interessant quan no fem servir el bus I2C, o per wearables on podem canviar la connexió amb el fil.

Podeu fer servir els mateixos programes que tenim pel nostre mòdul. Això si, cal canviar el pin per D2.



Battery Shield

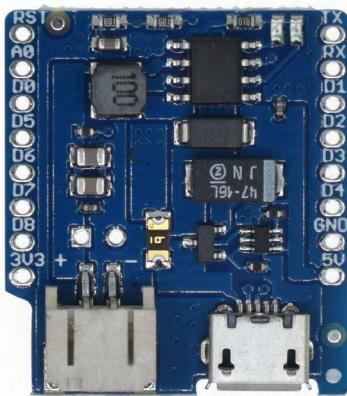
Es tracta d'un shield al qual podem connectar una bateria de Liti (3,3 V : 4,2 V) amb un connector PH2-2.0MM o bé soldada a la placa.

La bateria es carrega amb el seu propi connector microUSB. Podem configurar el shield per carregar amb un màxim de 0,5 A (per defecte) o 1 A.

El shield genera 5 V / 1 A amb la bateria i els injecta a l'alimentació del bus del D1 mini.

Porta dos leds: vermell mentre carrega i verd quan la càrrega és completa.

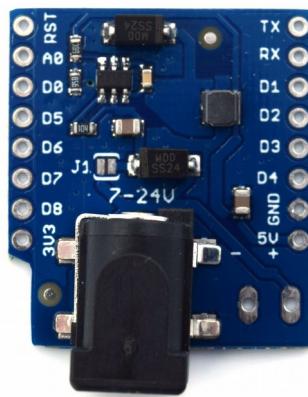
Podem configurar el shield per connectar 'entrada analògica A0 a la bateria i així monitoritzar el seu estat.



DC Power Shield

Aquest shield porta un regulador de tensió de 5 V / 1 A que injecta al bus D1 mini.

Podem alimentar el sistema amb una entrada de 7 V : 24 V connectats amb un connector de barril, soldant a la placa o amb una regleta



Part VI. HTML5, javaScript i AJAX

El llenguatge HTML va ser dissenyat al CERN per Tim Berners-Lee i Robert Caillau al 1991 per interconnectar estacions de treball gràfiques X-Window. Per compartir informació entre ordinadors diferents.

En la evolució de la web, empreses amb programari propietari han corromput aquesta filosofia: extensions del navegador de Microsoft, plugins flash i java ... han dificultat crear continguts amb aquest estàndard multiplataforma.

HTML5 torna a posar les coses al seu lloc. Amb la seva potència no hi ha excusa per altres extensions o plugins sinó és per ànsia de poder de certes companyies.

HTML5 va estretament lligat a javascript i CSS. El primer per fer codi que s'executa al navegador del client, estalviant recursos al servidor i transit de dades a la xarxa, i accelerant la resposta de la pàgina. Objectius prioritaris quan el nostre servidor és un modest ESP8266, que, a més a més, és possible que hagi generat la seva pròpria xarxa.

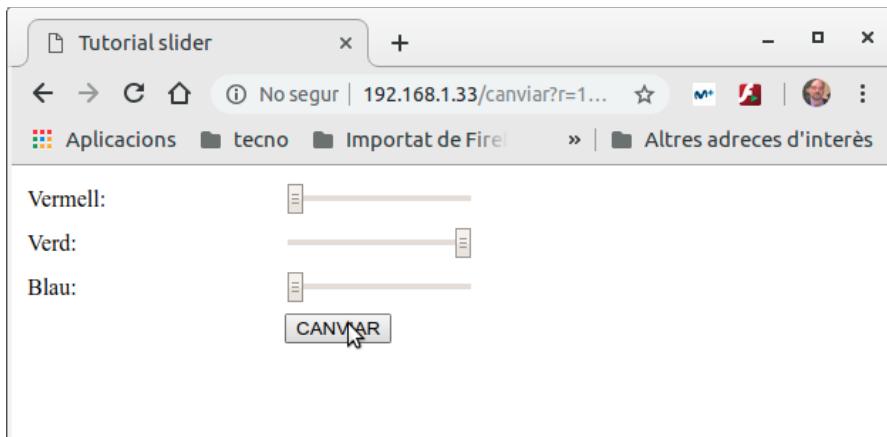
Al llarg d'aquest capítol veurem alguns dels recursos que HTML5, javascript i CSS ens ofereixen per millorar les nostres interfícies de xarxa. Però recordeu que això és només un tasteig, les possibilitats¹⁶ són il·limitades!

¹⁶ Us recomano la fantàstica web <https://www.w3schools.com/>. Els seus tutorials, referències i exemples són genials!

HTML5 input type='range'. Creació de sliders.

HTML5 introduceix nous tipus d'entrades als formularis¹⁷. Comencem amb el tipus *range*, que genera una barra de selecció o slider.

Veiem un exemple (*slider*) on fem servir tres sliders per modificar les components de color del led central del nostre shield RGB:



```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

const char *ssid = "Aquí el nom de la teva xarxa";
const char *password = "Aquí la contrasenya de la xarxa";

ESP8266WebServer server ( 80 );

#include <Adafruit_NeoPixel.h>
#define PIN          D8
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(1, PIN, NEO_GRB + NEO_KHZ800);

String mensaje = "";

-----CODIGO HTML PAGINA DE CONFIGURACION-----
String pagina = "<!DOCTYPE html>
"<html>"
"<head>"
"<title>Tutorial slider</title>"
"<meta charset='UTF-8'>"
"<meta name='viewport' content='width=device-width' />"
"</head>"
```

17 Veure https://www.w3schools.com/html/html_form_input_types.asp

Part VI. HTML5, javaScript i AJAX

```
<body>
<table style='width:100%'><form action='canviar' method='get'>
<tr><td>Vermell:</td><td><input type='range' name='r' min='0'
max='255'></td></tr>
<tr><td>Verd:</td><td><input type='range' name='g' min='0'
max='255'></td></tr>
<tr><td>Blau:</td><td><input type='range' name='b' min='0'
max='255'></td></tr>
<tr><td></td><td><input type='submit' value='CANVIAR' /></td></tr>
</form></table>
</body>
</html>;

void paginacanvi() {
    server.send(200, "text/html", pagina);
}

void canviar_colors() {
    pixels.setPixelColor(0, server.arg("r").toInt(), server.arg("g").toInt(),
server.arg("b").toInt());
    pixels.show(); // This sends the updated pixel color to the hardware.
    paginacanvi();
}

void setup() {
    WiFi.begin ( ssid, password );
    pixels.begin();
    server.on("/", paginacanvi);
    server.on("/canviar", canviar_colors);
    server.begin();

    pixels.setPixelColor(0, pixels.Color(0, 63, 0));
    pixels.show();
}

void loop() {
    server.handleClient();
}
```

Primer de tot fixeu-vos que he canviat el D2 original per D8 a la línia

```
#define PIN          D8
```

ja que aquest exemple inicialment va ser fet pel WS2812B RGB Shield, inclòs a la primera versió del kit.

Us recomano afegir sempre al <head> la línia

```
<meta name='viewport' content='width=device-width' />
```

que adapta la visualització de la pàgina a l'amplada del dispositiu. Imprescindible per mostrar correctament la pàgina a dispositius mòbils.

IoT amb D1 mini (ESP8266) i codi Arduino

Fixeu-vos que al codi HTML sempre faig servir la cometa simple ('). Això permet incloure el codi HTML al codi arduino amb cometes dobles (").

Per crear al formulari el slider fem servir el codi

```
<input type='range' name='g' min='0' max='255'>
```

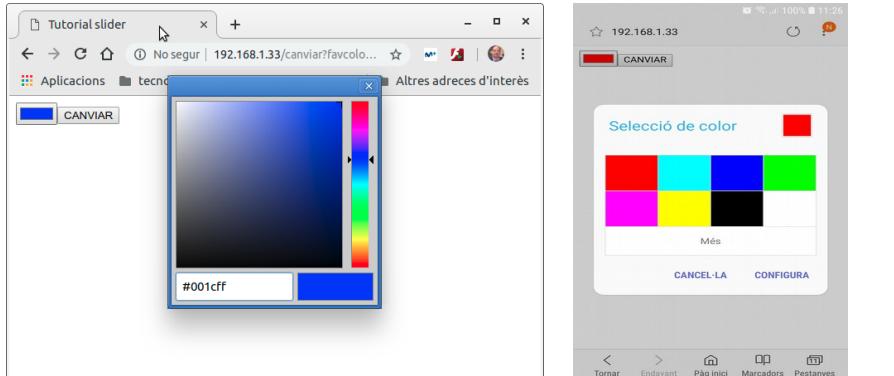
Aquest valor el recuperem i interpretarem com a nombre amb el codi

```
server.arg("g").toInt()
```

Això ho fem pels tres colors RGB, cadascú amb el seu propi paràmetre.

HTML5 input type='color'

A l'exemple *Color_Picker* seleccionem el color del nostre RGB shield amb un selector de color. La seva aparença canvia segons el dispositiu:



```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

const char *ssid = "Aquí el nom de la teva xarxa";
const char *password = "Aquí la contrasenya de la xarxa";

ESP8266WebServer server ( 80 );
#include <Adafruit_NeoPixel.h>
#define PIN D8
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(1, PIN, NEO_GRB + NEO_KHZ800);

String mensaje = "";

-----CODIGO HTML PAGINA DE CONFIGURACION-----
String pagina = "<!DOCTYPE html>
<html>
<head>
<title>Tutorial slider</title>
<meta charset='UTF-8'>
<meta name='viewport' content='width=device-width'/>
</head>
<body>
<form action='canviar' method='get'>
<input type='color' name='favcolor' value='#ff0000'>
<input type='submit' value='CANVIAR' /><br>
</form>
</body>
</html>";

void paginacanvi() {
    server.send(200, "text/html", pagina);
}
```

IoT amb D1 mini (ESP8266) i codi Arduino

```
int hex2dec(byte c) {
    if (c >= '0' && c <= '9') {
        return c - '0';
    } else if (c >= 'A' && c <= 'F') {
        return c - 'A' + 10;
    } else if (c >= 'a' && c <= 'f') {
        return c - 'a' + 10;
    }
}

void canviar_colors() {
    int r,g,b;
    String c;
    c = server.arg("favcolor");
    Serial.print("c= ");
    Serial.println(c);
    r=hex2dec(c.charAt(1))*16+hex2dec(c.charAt(2));
    g=hex2dec(c.charAt(3))*16+hex2dec(c.charAt(4));
    b=hex2dec(c.charAt(5))*16+hex2dec(c.charAt(6));
    Serial.print("r= ");
    Serial.println(r);
    Serial.print("g= ");
    Serial.println(g);
    Serial.print("b= ");
    Serial.println(b);
    pixels.setPixelColor(0, r, g, b);
    pixels.show(); // This sends the updated pixel color to the hardware.
    paginacanvi();
}

void setup() {
    Serial.begin(115200);
    WiFi.begin ( ssid, password );
    pixels.begin(); // This initializes the NeoPixel library.
    server.on("/", paginacanvi);
    server.on("/canviar", canviar_colors);
    server.begin();
    pixels.setPixelColor(0, pixels.Color(0, 63, 0));
    pixels.show(); // This sends the updated pixel color to the hardware.
}

void loop() {
    server.handleClient();
}
```

Amb el codi HTML5

```
<input type='color' name='favcolor' value='#ff0000'>
```

seleccionem un color en format RGB. Separem les components amb codi arduino del tipus

```
r=hex2dec(c.charAt(1))*16+hex2dec(c.charAt(2));
```

per poder cridar a la funció *setPixelColor()* amb les tres components de color separades.

Incloure eines HTML5 a la SD virtual

A la comunitat del programari lliure hi ha una quantitat immensa de programadors que han creat eines HTML5 fantàstiques que es poden aprofitar desant uns quants fitxers a la nostra SD virtual. Podem fer servir *FSManager*, la navalla suïssa per a la gestió de fitxers a la SD virtual via WiFi per desar aquests fitxers i accedir-hi.

A l'exemple *carrusel* he modificat l'exemple *bitmapSPIFFS* afegint-hi la funcionalitat de *FSManager* i fent un escanejat de la SD virtual per trobar arxius amb l'extensió *.dat* per mostrar els bitmaps que contenen. Per simplificar el seu ús, la funcionalitat del *FSManager* l'he posada en dos fitxers (*FSManager.cpp* i *FSManager.h*) a la mateixa carpeta del programa¹⁸.

I, el més important, he afegit a la carpeta *data* l'eina *image2cpp*¹⁹ que permet codificar nous bitmaps per afegir-ne.

Aquí teniu el contingut de *carrusel.ino*

```
#include <Wire.h>
#include <SFE_MicroOLED.h>
#include <FS.h>
#include "FSManager.h"
extern ESP8266WebServer server;

#define PIN_RESET 255 // 
#define DC_JUMPER 0 // I2C Addres: 0 - 0x3C, 1 - 0x3D
MicroOLED oled(PIN_RESET, DC_JUMPER); // I2C Example
File f;
uint8_t pixels[16*24];

void loadbitmap(String fitxer);

void setup() {
    SPIFFS.begin();
    initHelper(); //inclou connexió Wifi i server
                  // Aquí podem afegir altres funcions server.on()
    server.begin();

    oled.begin();
    oled.clear(ALL);
```

18 Veure Annex : Afegint la funcionalitat de *FSManager* als nostres projectes

19 Disponible a <http://javl.github.io/image2cpp/>

IoT amb D1 mini (ESP8266) i codi Arduino

```
oled.display();
delay(1000);
}

void loop() {
  Dir dir = SPIFFS.openDir("/");
  while (dir.next()) {
    String fileName = dir.fileName();
    if(fileName.endsWith(".dat")){
      oled.clear(PAGE);
      loadbitmap(fileName);
      oled.drawBitmap(pixels);
      oled.display();
      espera(1000);
    }
  }
}

String getLine() {
  String S = "" ;
  char c = f.read();
  while ( c != '\n') {
    S = S + c ;
    c = f.read();
  }
  return(S) ;
}

int convertFromHex(int ascii){
  if(ascii > 0x39) ascii -= 7; // adjust for hex letters upper or lower case
  return(ascii & 0xf);
}

void loadbitmap(String fitxer){
  uint8_t sa[16];
  String txt;  int i,j,k;
  f = SPIFFS.open(fitxer,"r");
  for (i=0;i<24;i++){
    txt=getLine();
    k=0;
    for (j=0; j<16; j++) {
      while (txt.charAt(k) != 'x') k++;
      sa[j] = convertFromHex(txt[k+1])*16;
      sa[j] += convertFromHex(txt[k+2]);
      k++;
    }
    for (j=0;j<16;j++) {
      pixels[i*16+j]=sa[j];
    }
  }
  f.close();
}
```

Utilitzant AJAX

Si volem mostrar dades a la nostra interfície web de forma eficient AJAX és la solució.

Només hem de generar pàgines de text amb el nostre ESP8266 amb les dades i una bonica pàgina web amb codi javascript a la nostre servidor web. Millor fer la pàgina a la SD virtual i modificar el seu aspecte amb el nostre *FSManager* incrustat.

Veiem l'exemple *DHT_ajax*:

```
#include <FS.h>
#include "FSManager.h"
extern ESP8266WebServer server;

#include <WEMOS_DHT12.h>
DHT12 dht12;

void setup() {
    SPIFFS.begin();
    pinMode(D4,OUTPUT);
    initHelper(); //inclou connexió Wifi i server
    server.on("/ajax_info.txt", HTTP_GET, []() {
        dht12.get();
        String txt = String(dht12.cTemp);
        server.send(200, "text/plain", txt);
        txt = String();
    });
    server.begin();
}

void loop() {
    digitalWrite(D4,LOW);
    espera(100);
    digitalWrite(D4,HIGH);
    espera(100);
}
```

Senzill i breu, oi?

Recordeu que a la mateixa carpeta tenim *FSManager.cpp* i *FSManager.h* que s'encarreguen de la gestió web de la SD virtual.

Ara s'entén perquè *server.begin()* no està inclòs dintre de *initHelper()*. Abans d'iniciar el servidor web cal definir les pàgines pròpies de la nostra aplicació.

IoT amb D1 mini (ESP8266) i codi Arduino

Veiem el fitxer *index.htm* creat a la SD virtual:

```
<html>
<head>
    <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>
    <meta name='viewport' content='width=device-width'>
    <title>AJAX Test</title>
    <style>
        body {
            background-color: Cornsilk;
        }
    </style>
    <script>
        function loadTemp() {
            var xhttp = new XMLHttpRequest();
            var temp = 0;
            xhttp.open('GET', 'ajax_info.txt', false);
            xhttp.send();
            temp = parseFloat(xhttp.responseText);
            return (temp);
        }
    </script>
    <style>
        #valor {
            border-radius: 25px;
            background: #73AD21;
            padding: 20px;
            width: 55px;
            height: 12px;
            text-align: center;
        }
    </style>
</head>
<body>
<h1>Temperatura</h1>
<div style="padding-left:40px;padding-bottom:5px;">
<p id='valor'></p>
</div>
<p><img id='icona' src='cloud.svg'></p>
<script>
var myVar = setInterval(myTimer, 1000);
function myTimer() {
    var t= loadTemp();
    document.getElementById('valor').innerHTML = t+' °C';
    if (t>27) {
        document.getElementById('icona').src = 'sun.svg';
    }
    else {
        document.getElementById('icona').src = 'snowflake.svg';
    }
}
</script>
</body>
</html>
```

Tota la màgia està en la funció *myTimer()*. No només actualitza el valor de la temperatura amb

```
var t= loadTemp();
document.getElementById('valor').innerHTML = t+' °C';
```

Part VI. HTML5, javaScript i AJAX

A més a més canvia la imatge de sota en funció del valor de la temperatura:

```
if (t>27) {  
    document.getElementById('icona').src = 'sun.svg';  
}  
else {  
    document.getElementById('icona').src = 'snowflake.svg';  
}
```

El truc és utilitzar *id* al codi HTML:

```
<p id='valor'></p>
```

que és el mateix *id* que fem servir per canviar el contingut en aquell paràgraf:

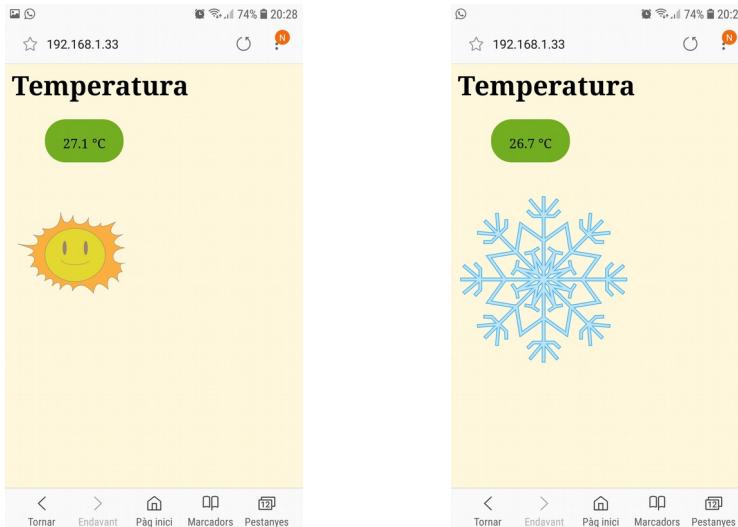
```
document.getElementById('valor').innerHTML = t+ ' °C';
```

Amb un altre *id* identifiquem la imatge mostrada:

```
<p><img id='icona' src='cloud.svg'></p>
```

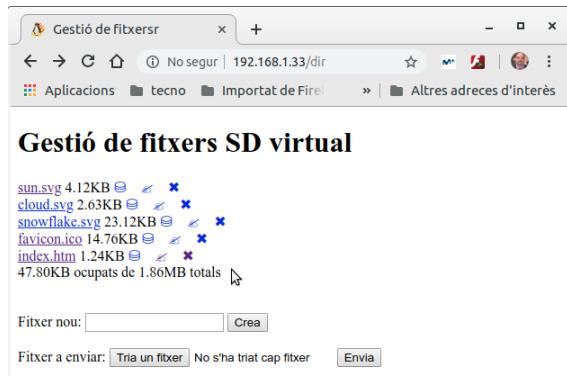
i la modifiquem:

```
document.getElementById('icona').src = 'sun.svg';
```



Naturalment a la nostra SD virtual tenim les tres imatges SVG:

IoT amb D1 mini (ESP8266) i codi Arduino



I tot això amb només 48 kB de la nostra SD virtual!

Fixeu-vos que poc ocupen les imatges. Són gràfics vectorials, re-escalables i molt eficients.

Fixeu-vos també com el codi CSS millora l'aspecte de la interfície. Aquest codi dona l'aparença d'un botó arrodonit a la temperatura:

```
<style>
#valor {
    border-radius: 25px;
    background: #73AD21;
    padding: 20px;
    width: 55px;
    height: 12px;
    text-align: center;
}
</style>
```

Fixeu-vos que valor és l'*id* del paràgraf on posem la temperatura, i va precedit pel signe sostingut (#).

La potència de javascript. Gràfics HTML canvas

Javascript és un gran llenguatge, que se executa al navegador del nostre ordinador o mòbil, amb una potència de càlcul molt superior al nostre modest ESP8266. Calaprofitar aquest avantatge a les nostres interfícies HTML.

HTML5 permet dos tipus de gràfics. SVG i canvas. SVG és vectorial, canvas orientat a bitmap. Cadascú té els seus avantatges i desavantatges.

En el meu cas m'interessa treballar amb canvas, ja que permet baixar directament la imatge com un arxiu png, i SVG no.

Fa poc em vaig fer una càmera tèrmica basada en el nostre kit, de forma que porta FSManager incrustat en el codi per servir els fitxers de les imatges capturades, que consisteixen en una matriu 8x8 de valors de la temperatura. Vull baixar a l'ordinador les imatges amb els colors mapejats tèrmicament per fer un tractament digital de la imatge i utilitzar-les en altres treballs.

Veiem l'exemple *visor_foto_term*:

```
#include <FS.h>
#include "FSManager.h"
extern ESP8266WebServer server;

void setup() {
  SPIFFS.begin();
  pinMode(D4,OUTPUT);
  initHelper(); //inclou connexió Wifi i server
  server.begin();
}

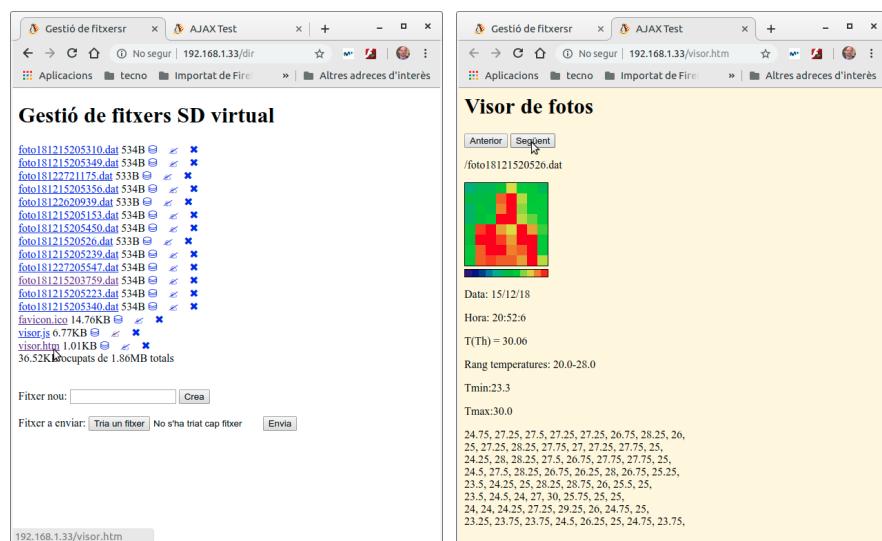
void loop() {
  digitalWrite(D4,LOW);
  espera(100);
  digitalWrite(D4,HIGH);
  espera(100);
}
```

Si. Només fa un blink. O això sembla ...

IoT amb D1 mini (ESP8266) i codi Arduino

A veure. En realitat el que m'interessa es posar a les vostres mans un sistema que funcioni i us permeti seguir l'exemple. Un ESP8266 amb el FSManager incrustat i amb imatges tèrmiques reals desades a la SD virtual. La meva càmera té un codi més llarg²⁰, ja que ha de fer les captures d'imatge, mostrar les imatges al TFT 1.4 Shield que porta i les dades al OLED Shield ... I, naturalment, crear el servidor amb el FSManager, que és la part que treballarem, on el veritable codi és a la interfície web, concretament als arxius *visor.htm* i *visor.js*.

Veiem primer el resultat:



Quan visualitzo el fitxer *visor.htm* aconsegueixo una interfície web que em permet desplaçar-me pels diferents arxius .dat de la SD virtual i visualitzar el mapa tèrmic i les diferents dades capturades. Tot utilitzant menys de 8 kB de la SD virtual!

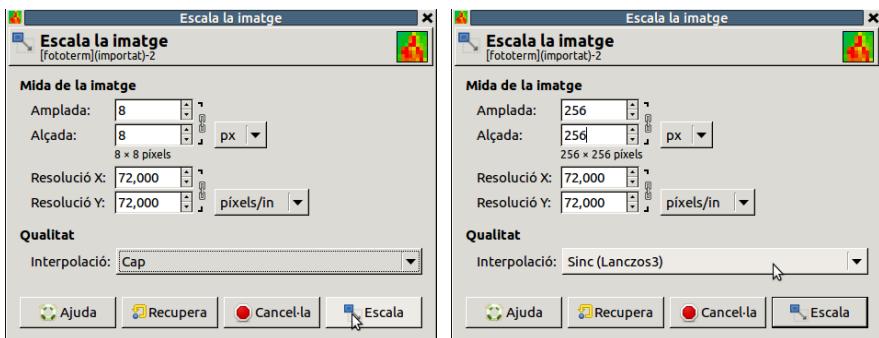
20 El codi de la càmera el teniu disponible a <https://github.com/jorts64/kit-D1-mini/tree/master/projectes/jorts/camera%20termica>

Part VI. HTML5, javaScript i AJAX

Fixeu-vos la reduïda mida dels fitxers d'imatge. A continuació teniu el contingut del fitxer .dat de la imatge mostrada:

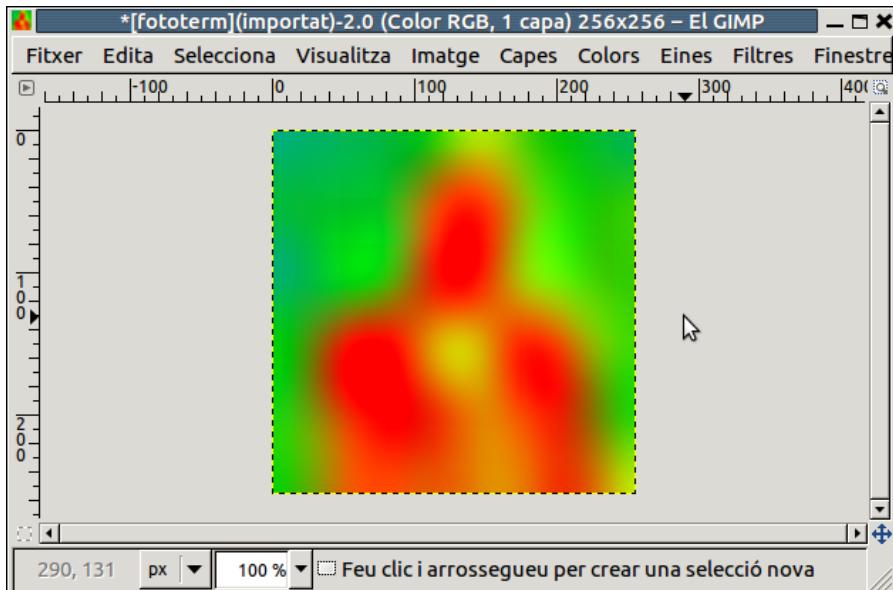
```
15/12/18
20:52:6
T(Th) = 30.06
20.0-28.0
Tmin:23.3
Tmax:30.0
24.75, 27.25, 27.50, 27.25, 27.25, 26.75, 28.25, 26.00,
25.00, 27.25, 28.25, 27.75, 27.00, 27.25, 27.75, 25.00,
24.25, 28.00, 28.25, 27.50, 26.75, 27.75, 27.75, 25.00,
24.50, 27.50, 28.25, 26.75, 26.25, 28.00, 26.75, 25.25,
23.50, 24.25, 25.00, 28.25, 28.75, 26.00, 25.50, 25.00,
23.50, 24.50, 24.00, 27.00, 30.00, 25.75, 25.00, 25.00,
24.00, 24.00, 24.25, 27.25, 29.25, 26.00, 24.75, 25.00,
23.25, 23.75, 23.75, 24.50, 26.25, 25.00, 24.75, 23.75,
```

Clicant amb el botó dret sobre la imatge la puc baixar com a arxiu .png. Si amb el GIMP redueixo la imatge a les dades originals (8x8) i faig una interpolació *Lanczos3* per aconseguir una imatge 256x256 :



aconsegueix una imatge millorada digitalment que difícilment podria obtenir amb l'ESP8266, i amb una transmissió de dades molt petita comparada amb el resultat:

IoT amb D1 mini (ESP8266) i codi Arduino



Impressionant, oi?

Al codi HTML carreguem les funcions javascript del fitxer *visor.js* al *<head>* i definim els diferents espais gràfics *<canvas>* i textuais *<p>*, cadascú amb el seu *id*, així com el botons per poder moure'ns entre les imatges. Finalment llegim el directori amb AJAX i visualitzem la primera fotografia. Aquí teniu el contingut de *visor.htm*:

```
<html>
<head>
  <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>
  <meta name='viewport' content='width=device-width'>
  <title>AJAX Test</title>
  <style>
    body {
      background-color: Cornsilk;
    }
  </style>
  <script src="visor.js"></script>
</head>
<body>
  <h1>Visor de fotos</h1>
  <div>
    <button onclick="anterior()">Anterior</button>
    <button onclick="seguent()">Següent</button>
  </div>

```


IoT amb D1 mini (ESP8266) i codi Arduino

```
xhttp.send();
dadesAJAX = xhttp.responseText;
return (dadesAJAX);
}
function readArray(fname) {
    var xhttp = new XMLHttpRequest();
    var dadesAJAX = "buit";
    xhttp.open('GET', fname, false);
    xhttp.send();
    dadesAJAX = xhttp.responseText;
    return (dadesAJAX);
}
function filterFile(value, index, array) {
    return value.endsWith(".dat");
}
function getDir() {
    var dirstr = readDir();
    var dirarr = dirstr.split(",");
    var dirdat = dirarr.filter(filterFile);
    nfotos = dirdat.length;
    return dirdat;
}
function seguent() {
    if (fotonum < nfotos-1) fotonum++;
    var myVar = loadArray(arxius[fotonum]);
}
function anterior() {
    if (fotonum > 0) fotonum--;
    var myVar = loadArray(arxius[fotonum]);
}
function pescala(ctx) {
    for (i=0; i<12;i++) {
        x=10*i;
        y=0;
        var tempi=(maxtemp - mintemp)/12*(i+0.5)+mintemp;
        var colorindexi = arduino_map(tempi, mintemp, maxtemp, 0, 255);
        var colori=camColors[colorindexi]; ctx.fillStyle = "#0000FF";
        ctx.fillStyle = colorRGBhex(colori);
        ctx.fillRect(x, y, 15, 20)
    }
}
function pfoto(ctx, boxSize) {
    var colorTemp;
    for (y=0; y<8; y++) {
        for (x=0; x<8; x++) {
            var val = pixels[(7-y)*8+x];
            colorTemp = val;
            if(val >= maxtemp) colorTemp = maxtemp;
            if(val <= mintemp) colorTemp = mintemp;
            var colorIndex = arduino_map(colorTemp, mintemp, maxtemp, 0, 255);
            colorIndex = arduino constrain(colorIndex, 0, 255);
            //draw the pixels!
            var color = camColors[colorIndex];
            ctx.fillStyle = colorRGBhex(color);
            ctx.fillRect(boxSize * x, boxSize * y, boxSize, boxSize);
        }
    }
}
function loadArray(fotarx) {
    document.getElementById('nomf').innerHTML = fotarx;
    var txt= readArray(fotarx);
//    document.getElementById('crues').innerHTML = txt;
    var r = 0; var s = 0;
    r = txt.indexOf("\n");
```

Part VI. HTML5, javaScript i AJAX

```
var fecha = txt.substring(0,r);
document.getElementById('fecha').innerHTML = "Data: " + fecha;
s = r+1;
r = txt.indexOf("\n",s+1);
var hora = txt.substring(s,r);
document.getElementById('hora').innerHTML = "Hora: " + hora;
s = r+1;
r = txt.indexOf("\n",s+1);
var tth = txt.substring(s,r);
document.getElementById('tth').innerHTML = tth;
s = r+1;
r = txt.indexOf("\n",s+1);
var rangt = txt.substring(s,r);
document.getElementById('rangt').innerHTML = "Rang temperatures: " + rangt;
s = r+1;
r = txt.indexOf("\n",s+1);
var tmin = txt.substring(s,r);
document.getElementById('tmin').innerHTML = tmin;
s = r+1;
r = txt.indexOf("\n",s+1);
var tmax = txt.substring(s,r);
document.getElementById('tmax').innerHTML = tmax;
for (i=0;i<64;i++) {
    s = r+1;
    r = txt.indexOf(", ",s+1);
    pixels[i]= parseFloat(txt.substring(s,r));
}
r = rangt.indexOf("-");
mintemp = parseFloat(rangt.substring(0,r));
maxtemp = parseFloat(rangt.substring(r+1));
// document.getElementById('mintemp').innerHTML = mintemp;
// document.getElementById('maxtemp').innerHTML = maxtemp;
var valors = "";
for (i=0;i<8;i++) {
    for (j=0;j<8;j++) {
        valors = " " + valors + pixels[i*8+j] + ", ";
    }
    valors = valors + "<br/>";
}
document.getElementById('array').innerHTML = valors;

var c = document.getElementById("escala");
var ctx1 = c.getContext("2d");
var esc1 = pescala(ctx1);

var c = document.getElementById("foto");
var ctx2 = c.getContext("2d");
var pf1 = pfoto(ctx2, 15);
}

function arduino_map(valinp, mininp, maxinp, minout, maxout) {
    var valout = (valinp - mininp) * (maxout - minout) / (maxinp - mininp) + minout;
    valout = Math.round(valout);
    return valout;
}
function arduino_constrain(valinp, minout, maxout) {
    var valout = valinp;
    if (valinp < minout) valout = minout;
    if (valinp > maxout) valout = maxout;
    return valout;
}
function colorRGB (col565) {
    var colout = (col565 & 0xF800) * 256;
    colout = colout + (col565 & 0x07E0) * 32;
    colout = colout + (col565 & 0x001F) * 8;
```

IoT amb D1 mini (ESP8266) i codi Arduino

```
    return colout;
}
function componentToHex(c) {
  var hex = c.toString(16);
  return hex.length == 1 ? "0" + hex : hex;
}
function rgbToHex(r, g, b) {
  return "#" + componentToHex(r) + componentToHex(g) +
componentToHex(b);
}
function colorR (col565) {
  var colout = (col565 & 0xF800) / 256;
  return colout;
}
function colorG (col565) {
  var colout = (col565 & 0x07E0) / 8;
  return colout;
}
function colorB (col565) {
  var colout = (col565 & 0x001F)*8;
  return colout;
}
function colorRGBhex (col565) {
  return rgbToHex(colorR(col565), colorG(col565), colorB(col565));
  return colout;
}
```

Cal dir que el programa podria haver estat molt més senzill, però he partit del mapejat de colors que faig servir al TFT 1.4 Shield de la càmera que utilitza coordenades de color especials²¹ ²² (5 bits pel vermell, 6 pel verd i 5 pel blau, en total 16 bits) mentre que <canvas> fa servir 8 bits per cada component RGB.

Comentem alguns punts interessants del codi javascript:

- Les variables locals porten el prefix *var* quan es declaren:

```
var r = 0; var s = 0;
```

- Si inicialitzem una variable sense el prefix *var* l'estem declarant com a variable global, accessible des de qualsevol funció:

²¹ Podeu obtenir més informació sobre aquestes coordenades de color i la llibreria gràfica GFX d'Adafruit a <https://learn.adafruit.com/adafruit-gfx-graphics-library/coordinate-system-and-units>

²² El sistema de mapejat de temperatura a colors és part dels exemples de la llibreria Adafruit pel sensor de càmera tèrmica AMG8833, ho podeu veure a <https://learn.adafruit.com/adafruit-amg8833-8x8-thermal-camera-sensor/arduino-thermal-camera>

Part VI. HTML5, javaScript i AJAX

```
pixels = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

- Com veieu, no és fàcil declarar arrays amb una mida fixa. Per això és tan llarga la declaració anterior. En canvi és fàcil afegir nous elements a un array amb el mètode *push*.
- És molt senzill generar un array a partir d'un text on els valors estan separats per un delimitador (el directori AJAX dona els noms d'arxius separats per comes):

```
var dirarr = dirstr.split(",");
```

- I podem eliminar els valors que no compleixen certa condició (jo només vull noms d'arxius de fotografies) definida en una funció pròpia:

```
var dirdat = dirarr.filter(filterFile);
```

- Podem modificar el *canvas*²³ des de qualsevol funció amb el seu *id*:

```
var c = document.getElementById("foto");
var ctx2 = c.getContext("2d");
```

- O amb el descriptor obtingut amb *getContext()*:

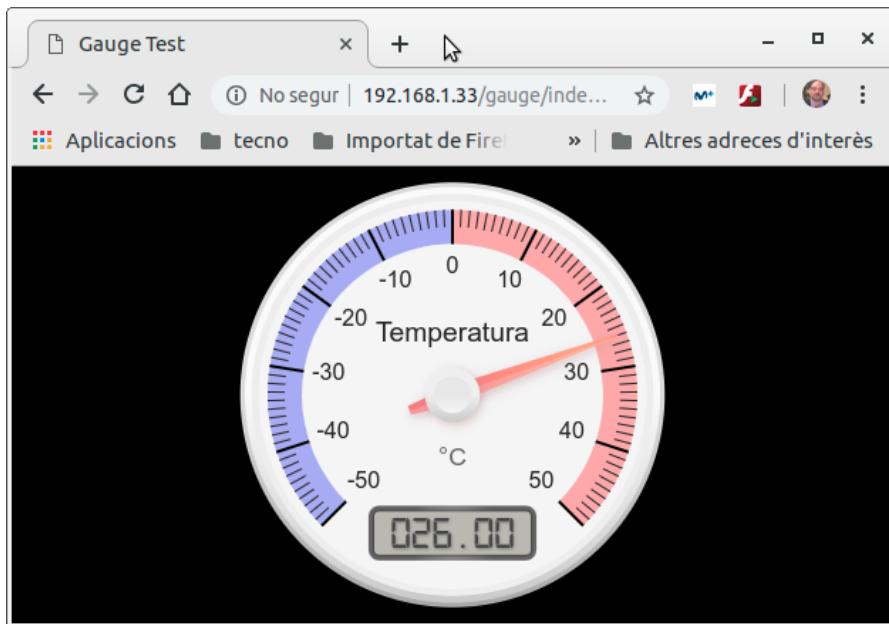
```
ctx.fillStyle = colorRGBHex(color);
ctx.fillRect(boxSize * x, boxSize * y, boxSize, boxSize);
```

23 Trobareu més informació sobre *canvas* i les seves funcions a https://www.w3schools.com/graphics/canvas_intro.asp

La llibreria gràfica gauge.js

Es tracta d'una llibreria gràfica per mostrar valors d'un sensor. Té una llicència MIT²⁴ que en permet desar-la a la nostra SD virtual i adaptar-la a les nostres necessitats.

La darrera versió de la llibreria sembla més fàcil de configurar i utilitzar, però a mi m'agrada molt una versió més antiga pel seu aspecte més formal:



A l'exemple *gauge* la faig servir amb el DHT shield:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <FS.h>
#include <WEMOS_DHT12.h>
DHT12 dht12;
```

24 Més informació sobre la llibreria gauge.js a <http://bernii.github.io/gauge.js/> on fins-i-tot podeu provar en línia variacions de paràmetres a la darrera versió de la llibreria

Part VI. HTML5, javaScript i AJAX

```
const char *ssid = "nom de la xarxa";
const char *password = "contrasenya";

ESP8266WebServer server ( 80 );
const int led = BUILTIN_LED;
int t;

void handleRoot() {
    digitalWrite ( led, LOW );
    char temp[400];
    int sec = millis() / 1000;
    int min = sec / 60;
    int hr = min / 60;
    sprintf ( temp, 400,
"<html> \
<head> \
    <meta http-equiv='refresh' content='5' /> \
    <title>ESP8266 Demo</title> \
    <style> \
        body { background-color: #cccccc; font-family: Arial, Helvetica, Sans-Serif; Color: #000088; } \
    </style> \
</head> \
<body> \
    <h1>Sonda de temperatura amb ESP8266</h1> \
    <p><a href='gauge/index.html'>gauge</a><p/> \
    <p>Uptime: %02d:%02d:%02d</p> \
</body> \
</html>",
    hr, min % 60, sec % 60
);
    server.send ( 200, "text/html", temp );
    digitalWrite ( led, HIGH );
}

void handleNotFound() {
    digitalWrite ( led, LOW );
    String message = "File Not Found\n\n";
    message += "URI: ";
    message += server.uri();
    message += "\nMethod: ";
    message += ( server.method() == HTTP_GET ) ? "GET" : "POST";
    message += "\nArguments: ";
    message += server.args();
    message += "\n";
    for ( uint8_t i = 0; i < server.args(); i++ ) {
        message += " " + server.argName ( i ) + ":" + server.arg ( i ) + "\n";
    }
    server.send ( 404, "text/plain", message );
    digitalWrite ( led, HIGH );
}

void handleAjax(){
    digitalWrite ( led, LOW );
    dht12.get();
    t=dht12.cTemp;
    Serial.print("Temperature: ");
    Serial.println(t);
    String out = "";
    char temp[100];
    sprintf(temp, 100, "%d", t);
    out += temp;
    server.send ( 200, "text/txt", out );
    digitalWrite ( led, HIGH );
}
```

IoT amb D1 mini (ESP8266) i codi Arduino

```
}  
  
void setup ( void ) {  
    pinMode ( led, OUTPUT );  
    digitalWrite ( led, HIGH );  
    Serial.begin ( 115200 );  
    WiFi.begin ( ssid, password );  
    SPIFFS.begin();  
    Serial.println ( "" );  
    // Wait for connection  
    while ( WiFi.status() != WL_CONNECTED ) {  
        delay ( 500 );  
        Serial.print ( "." );  
    }  
    Serial.println ( "" );  
    Serial.print ( "Connected to " );  
    Serial.println ( ssid );  
    Serial.print ( "IP address: " );  
    Serial.println ( WiFi.localIP() );  
    server.on ( "/", handleRoot );  
    server.serveStatic("/gauge", SPIFFS, "/gauge"); // gauge files dir  
    server.on ( "/ajax_info.txt", handleAjax );  
    server.onNotFound ( handleNotFound );  
    server.begin();  
    Serial.println ( "HTTP server started" );  
}  
void loop ( void ) {  
    server.handleClient();  
}
```

Podem configurar el control al fitxer */gauge/index.html*:

```
<html>  
<head>  
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
    <meta name="viewport" content="width=device-width">  
    <title>Gauge Test</title>  
    <script src="gauge.js"></script>  
    <style>body{padding:0;margin:0;background:#222}</style>  
    <script>  
function loadTemp() {  
    var xhttp = new XMLHttpRequest();  
    var temp = 0;  
    xhttp.open("GET", "../ajax_info.txt", false);  
    xhttp.send();  
    temp = parseFloat(xhttp.responseText);  
    return (temp);  
}  
function DrawGauge()  
{  
    gauge = new Gauge({  
        renderTo : 'gauge',  
        width : document.body.clientWidth,  
        height : document.body.clientHeight,  
        glow : true,  
        units : '°C',  
        title : 'Temperatura',  
        minValue : -50,  
        maxValue : 50,  
        majorTicks : ['-50', '-40', '-30', '-20', '-10', '0', '10', '20', '30', '40', '50'],  
        minorTicks : 10,  
        strokeTicks : false,  
        highlights : [  
    }
```

Part VI. HTML5, javaScript i AJAX

```
        { from : -50, to : 0, color : 'rgba(0, 0, 255, .3)' },
        { from : 0, to : 50, color : 'rgba(255, 0, 0, .3)' }
    ],
colors : {
    plate      : '#f5f5f5',
    majorTicks : '#000',
    minorTicks : '#222',
    title      : '#222',
    units      : '#666',
    numbers    : '#222',
    needle     : { start : 'rgba(240, 128, 128, 1)', end : 'rgba(255, 160, 122, .9)' }
},
animation : {
    delay : 25,
    duration: 500,
    fn : 'bounce'
}
});
gauge.onready = function() {
    setInterval( function() {
        gauge.setValue( loadTemp());
    }, 1000);
};
    gauge.draw();
};

</script>
</head>
<body style="background-color:black;">
    <canvas id="gauge"></canvas>
    <div id="console"></div>
    <script>
        DrawGauge();
    </script>
</html>
```

Com veieu és molt fàcil canviar

```
units      : '°C',
title      : 'Temperatura',
minValue   : -50,
maxValue   : 50,
```

o d'altres paràmetres.

La funció que dona les dades AJAX es podria reduir a la seva expressió mínima:

```
void handleAjax(){
    dht12.get();
    t=dht12.cTemp;
    String out = "";
    char temp[100];
    sprintf(temp, 100, "%d", t);
    out += temp;
    server.send ( 200, "text/txt", out);
}
```

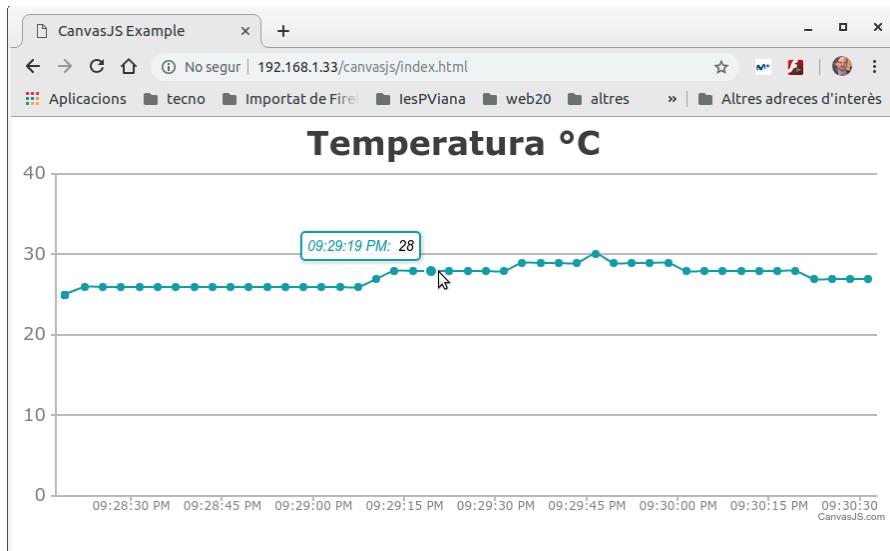
IoT amb D1 mini (ESP8266) i codi Arduino

La llibreria gràfica *canvasjs*. Gràfiques de dades.

Aquesta llibreria²⁵ la podeu fer servir amb una llicència CC BY-NC.

A la carpeta *canvasjs/examples* trobareu diferents exemples de gràfiques que es poden fer amb aquesta llibreria.

El fitxer *canvas/index.html* és una versió modificada de l'exemple *spline*, que actualitza les dades del nostre DHT shield amb tecnologia AJAX:



El codi arduino és el mateix que el del capítol La llibreria gràfica *gauge.js*, ja que el vaig fer compatible amb els dos exemples de les llibreries. El que canvia en aquest exemple *canvasjs* és el contingut de la SD virtual. No només hi ha aquest tipus de gràfica. A la seva web podem trobar més de 30 exemples: àrees, rangs, acumulats, columnes, barres, sectors, bombolles, capses ... El problema és que per fer servir qualsevol gràfic ha d'utilitzar codi extern al núvol i, per tant, tenir connexió a internet. No funciona em mode AP!

25 Disponible a <http://canvasjs.com>

Part VI. HTML5, javaScript i AJAX

Veiem el contingut de *canvas/index.html*:

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
window.onload = function () {
var chart = new CanvasJS.Chart("chartContainer", {
    title: {
        text: "Temperatura °C"
    },
    data: [
        {
            type: "spline",
            dataPoints: [
                { x: new Date() , y: 25 },
                { x: new Date() , y: 25 }
            ]
        }
    );
chart.render();

function loadTemp() {
    var xhttp = new XMLHttpRequest();
    var temp = 0;
    xhttp.open("GET", "../ajax_info.txt", false);
    xhttp.send();
    temp = parseFloat(xhttp.responseText);
    chart.options.data[0].dataPoints.push({ x: new Date() , y: temp});
    chart.render();
}
setInterval( function() {
    loadTemp();
}, 3000);
}
</script>
<script src="canvasjs.min.js"></script>
<title>CanvasJS Example</title>
</head>
<body>
    <div id="chartContainer" style="height: 400px; width: 100%;"></div>
</body>
</html>
```

Part VII. Projectes

Part VIII. Prototips comercials

Part IX. Actualització WiFi del programa (OTA)

Part X. Altres plaques basades en l'ESP8266

Els mòduls ESP12 V1 i V3. La placa NODEMCU Base v1.0

Els mòduls ESP12 V2. La placa NODEMCU motor shield

Els mòduls ESP-01. Plaques compatibles

Part XI. Altres components externs interessants

Part XII. Wearables

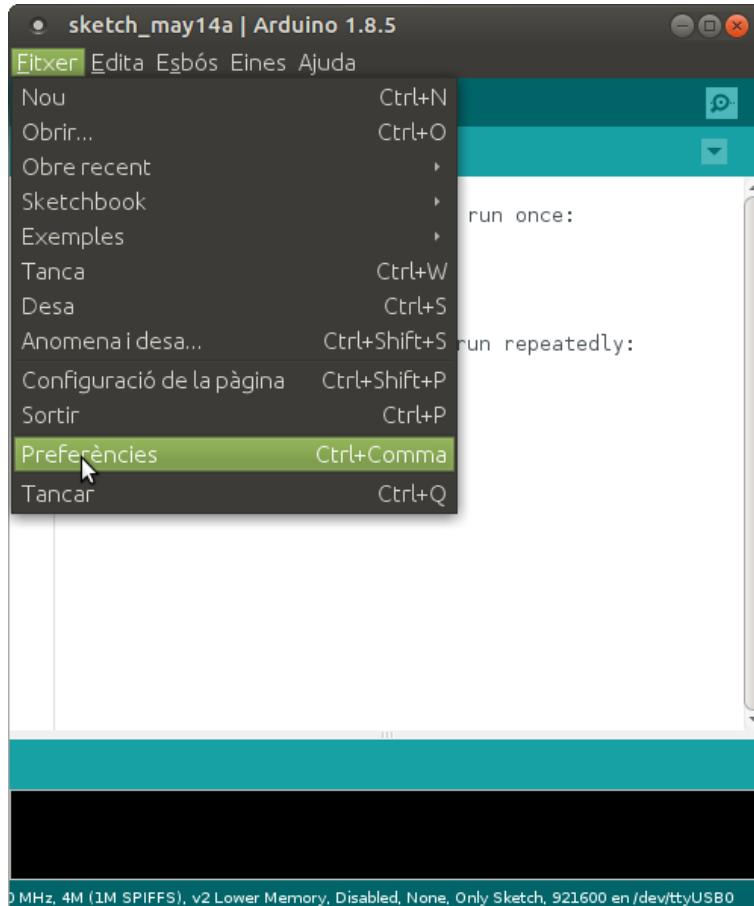
Part XII. Wearables

Annexos

Annex : Instal·lació Arduino per a ESP8266

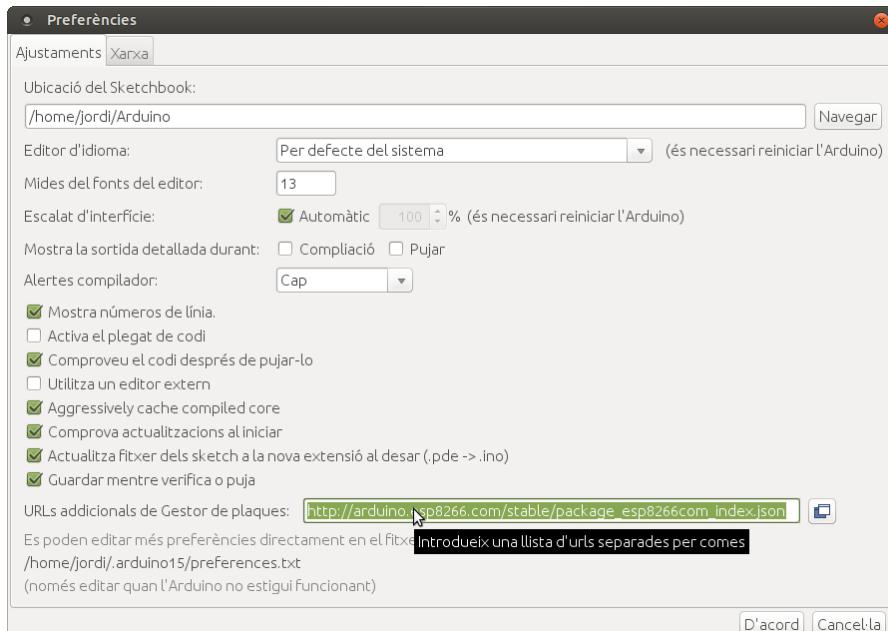
Ens caldrà un arduino IDE modern que permeti l'extensió a nous xips amb el Gestor targes. Nosaltres actualment fem servir la versió 1.8.5, però també hem fet servir la versió 1.6.8 sense cap problema.

Una vegada instal·lat, hem d'anar a *Fitxer->Preferències*

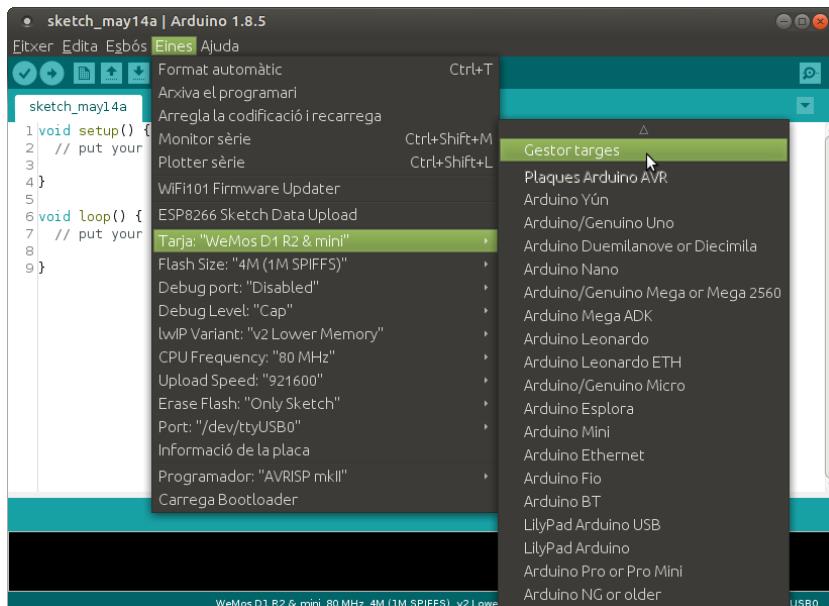


I omplir l'opció URLs adiccionals de Gestors de plaques amb
http://arduino.esp8266.com/stable/package_esp8266com_index.json

IoT amb D1 mini (ESP8266) i codi Arduino



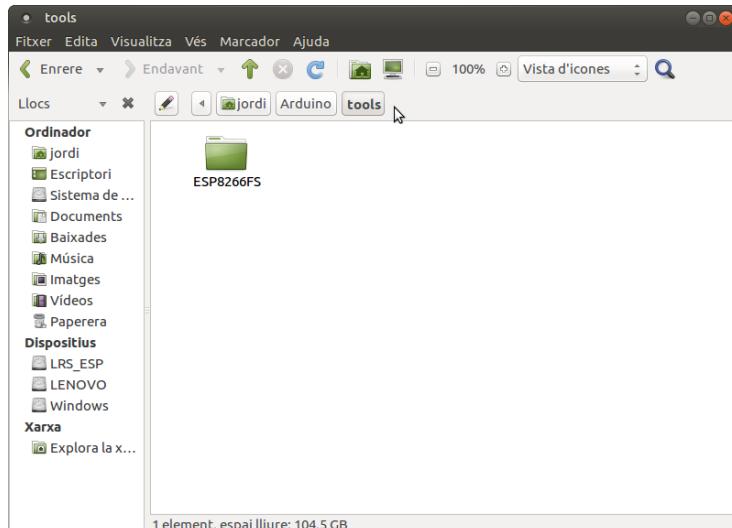
Ara podem anar a *Eines-> Tarja -> Gestor targes* i escollir *esp8266* per instal·lar-la



Annexos



Finalment baixem l'arxiu <https://github.com/esp8266/arduino-esp8266fs-plugin/releases/download/0.3.0/ESP8266FS-0.3.0.zip> i el descomprimiu a la carpeta tools (potser haureu de crear aquesta carpeta) de la carpeta Arduino del vostre espai personal



Podeu trobar més informació a <https://github.com/esp8266/Arduino>

Annex : Taula de colors RGB

Aquest array el faig servir al meu exemple Peana per canviar de colors un led RGB amb la funció *setPixelColor()*:

```
uint32_t color[] = {
    0xFFC0CB, // 0 Pink
    0xFFFFC1, // 1 LightPink
    0xFFFFB4, // 2 HotPink
    0xFF1493, // 3 DeepPink
    0xDB7093, // 4 PaleVioletRed
    0xC71585, // 5 MediumVioletRed
    0xFFA07A, // 6 LightSalmon
    0xFA8072, // 7 Salmon
    0xE9967A, // 8 DarkSalmon
    0xF08080, // 9 LightCoral
    0xCD5C5C, // 10 IndianRed
    0xDC143C, // 11 Crimson
    0xB22222, // 12 FireBrick
    0x800000, // 13 DarkRed
    0xFF0000, // 14 Red
    0xFF4500, // 15 OrangeRed
    0xFF6347, // 16 Tomato
    0xFF7F50, // 17 Coral
    0xFF8C00, // 18 DarkOrange
    0xFFA500, // 19 Orange
    0xFFD700, // 20 Gold
    0xFFFF00, // 21 Yellow
    0xFFFFE0, // 22 LightYellow
    0xFFFFACD, // 23 LemonChiffon
    0xFFAFAD2, // 24 LightGoldenrodYellow
    0xFFEFD5, // 25 PapayaWhip
    0xFFE4B5, // 26 Moccasin
    0xFFDAB9, // 27 PeachPuff
    0xEEE8AA, // 28 PaleGoldenrod
    0xF0E68C, // 29 Khaki
    0xBDB76B, // 30 DarkKhaki
    0xFFFF8DC, // 31 Cornsilk
    0xFFEBBC, // 32 BlanchedAlmond
    0xFFE4C4, // 33 Bisque
    0xFFDEAD, // 34 NavajoWhite
    0xF5DEB3, // 35 Wheat
    0xDEB887, // 36 BurlyWood
    0xD2B48C, // 37 Tan
    0xBC8F8F, // 38 RosyBrown
    0xF4A460, // 39 SandyBrown
    0xDAA520, // 40 Goldenrod
    0xB8860B, // 41 DarkGoldenrod
    0xCD853F, // 42 Peru
    0xD2691E, // 43 Chocolate
    0x8B4513, // 44 SaddleBrown
    0xA0522D, // 45 Sienna
    0xA52A2A, // 46 Brown
    0x800000, // 47 Maroon
    0x556B2F, // 48 DarkOliveGreen
    0x808000, // 49 Olive
    0x6B88E23, // 50 OliveDrab
    0x9ACD32, // 51 YellowGreen
    0x32CD32, // 52 LimeGreen
    0x00FF00, // 53 Lime
    0x7FC00, // 54 LawnGreen
}
```

Annexos

```
0x7FFF00, // 55 Chartreuse
0xADFF2F, // 56 GreenYellow
0x00FF7F, // 57 SpringGreen
0x00FA9A, // 58 MediumSpringGreen
0x90EE90, // 59 LightGreen
0x98FB98, // 60 PaleGreen
0x8FBC8F, // 61 DarkSeaGreen
0x3CB371, // 62 MediumSeaGreen
0x2E8B57, // 63 SeaGreen
0x228B22, // 64 ForestGreen
0x008000, // 65 Green
0x006400, // 66 DarkGreen
0x66CDA, // 67 MediumAquamarine
0x00FFFF, // 68 Aqua
0x00FFFF, // 69 Cyan
0xE0FFFF, // 70 LightCyan
0xAFEEEE, // 71 PaleTurquoise
0x7FFF0D4, // 72 Aquamarine
0x40E0D0, // 73 Turquoise
0x48D1CC, // 74 MediumTurquoise
0x00CED1, // 75 DarkTurquoise
0x20B2AA, // 76 LightSeaGreen
0x5F9EA0, // 77 CadetBlue
0x008B8B, // 78 DarkCyan
0x008080, // 79 Teal
0xB0C4DE, // 80 LightSteelBlue
0xB0E0E6, // 81 PowderBlue
0xADD8E6, // 82 LightBlue
0x87CEEB, // 83 SkyBlue
0x87CEFA, // 84 LightSkyBlue
0x00BFFF, // 85 DeepSkyBlue
0x1E90FF, // 86 DodgerBlue
0x6495ED, // 87 CornflowerBlue
0x4682B4, // 88 SteelBlue
0x4169E1, // 89 RoyalBlue
0x0000FF, // 90 Blue
0x0000CD, // 91 MediumBlue
0x00008B, // 92 DarkBlue
0x000080, // 93 Navy
0x191970, // 94 MidnightBlue
0xE6E6FA, // 95 Lavender
0xD8BF0D8, // 96 Thistle
0xDDA0DD, // 97 Plum
0xEE82EE, // 98 Violet
0xDA70D6, // 99 Orchid
0xFF00FF, // 100 Magenta
0xBA55D3, // 101 MediumOrchid
0x9370DB, // 102 MediumPurple
0x8A2BE2, // 103 BlueViolet
0x9400D3, // 104 DarkViolet
0x9932CC, // 105 DarkOrchid
0x8B008B, // 106 DarkMagenta
0x800080, // 107 Purple
0x4B0082, // 108 Indigo
0x483D8B, // 109 DarkSlateBlue
0x6A5ACD, // 110 SlateBlue
0x7B68EE, // 111 MediumSlateBlue
0xFFFF00, // 112 White
0xF5F5DC, // 113 Beige
0xFAEBD7, // 114 AntiqueWhite
0xFFE4E1, // 115 MistyRose
0x808080, // 116 Gray
0x708090 // 117 SlateGray
};
```

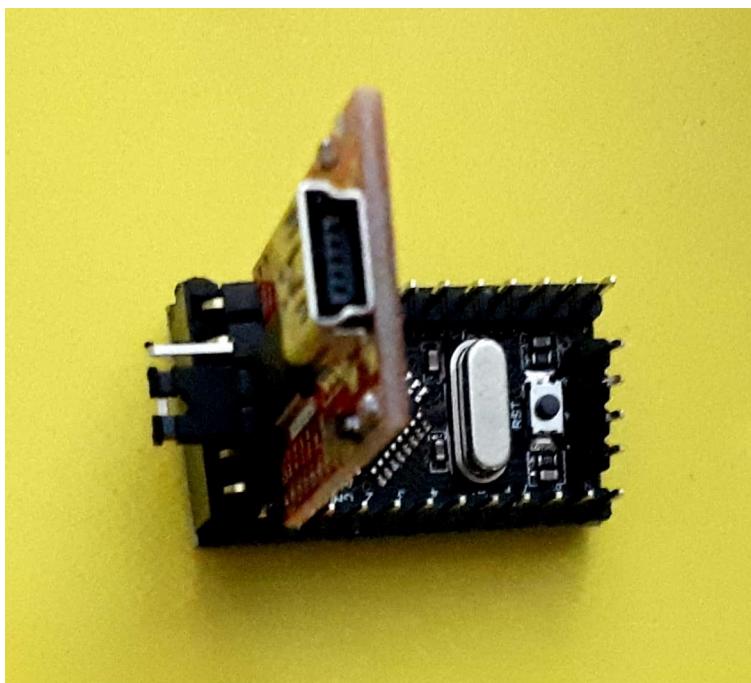
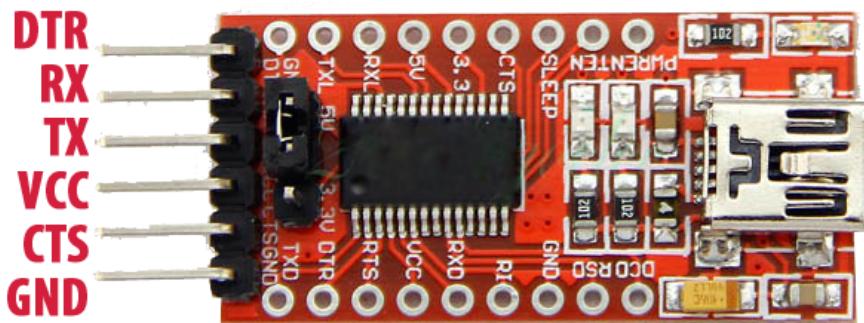
Annex: Taula de freqüències per a les notes musicals

Aquest array el faig servir al meu exemple musica:

```
int notes [36][3] =
{
{'8',B00100000,262}, // C5
{(')',B00100001,277}, // C5#
{'9',B00100010,294}, // D5
{(')',B00100011,311}, // D5#
{'0',B00100100,330}, // E5
{'q',B00100101,349}, // F5
{'Q',B00100110,370}, // F5#
{'w',B00100111,392}, // G5
{'W',B00101000,415}, // G5#
{'e',B00101001,440}, // A5
{'E',B00101010,466}, // A5#
{'r',B00101011,494}, // B5
{'t',B00000000,523}, // C6
{'T',B00000001,554}, // C6#
{'y',B00000010,587}, // D6
{'Y',B00000011,622}, // D6#
{'u',B00000100,669}, // E6
{'i',B00000101,698}, // F6
{'I',B00000110,740}, // F6#
{'o',B00000111,784}, // G6
{'O',B00001000,831}, // G6#
{'p',B00001001,880}, // A6
{'P',B00001010,932}, // A6#
{'a',B00001011,988}, // B6
{'s',B00010000,1047}, // C7
{'S',B00010001,1109}, // C7#
{'d',B00010010,1175}, // D7
{'D',B00010011,1245}, // D7#
{'f',B00010100,1318}, // E7
{'g',B00010101,1396}, // F7
{'G',B00010110,1480}, // F7#
{'h',B00010111,1568}, // G7
{'H',B00011000,1661}, // G7#
{'j',B00011001,1760}, // A7
{'J',B00011010,1865}, // A7#
{'k',B00011011,1975}, // B7
};
```

Annex: Firmware de l'arduino pro mini

Podem utilitzar un mòdul FTDI232 endollant-lo directament a l'arduino pro mini per carregar el firmware:



IoT amb D1 mini (ESP8266) i codi Arduino

I2CsapmR1_firmware.h (V1.0)

```
#define I2CsapmR1_A0 0
#define I2CsapmR1_A1 1
#define I2CsapmR1_A2 2
#define I2CsapmR1_A3 3
#define I2CsapmR1_A6 4
#define I2CsapmR1_A7 5
#define I2CsapmR1_D2 6
#define I2CsapmR1_D3 7
#define I2CsapmR1_D4 8
#define I2CsapmR1_D5 9
#define I2CsapmR1_D6 10
#define I2CsapmR1_D7 11
#define I2CsapmR1_D8 12
#define I2CsapmR1_D9 13
#define I2CsapmR1_D10 14
#define I2CsapmR1_D11 15
#define I2CsapmR1_D12 16
#define I2CsapmR1_D13 17
#define I2CsapmR1_M2 18
#define I2CsapmR1_M3 19
#define I2CsapmR1_M4 20
#define I2CsapmR1_M5 21
#define I2CsapmR1_M6 22
#define I2CsapmR1_M7 23
#define I2CsapmR1_M8 24
#define I2CsapmR1_M9 25
#define I2CsapmR1_M10 26
#define I2CsapmR1_M11 27
#define I2CsapmR1_M12 28
#define I2CsapmR1_M13 29
```

I2CsapmR1_firmware.cpp (V1.0)

```
#include "I2CsapmR1_firmware.h"
#include <Wire.h>
#define I2CADDR 2

byte ordre = 0;
byte dada = 0;

int analin[6];
int digm[12];
int digd[12];

int i;

void setup() {
  pinMode(2, INPUT);
  pinMode(3, INPUT);
  pinMode(4, INPUT);
  pinMode(5, INPUT);
  pinMode(6, INPUT);
  pinMode(7, INPUT);
  pinMode(8, INPUT);
  pinMode(9, INPUT);
  pinMode(10, INPUT);
  pinMode(11, INPUT);
  pinMode(12, INPUT);
  pinMode(13, INPUT);
  for (i=0;i<12;i++) digm[i]=INPUT;
  Wire.begin(I2CADDR);
```

Annexos

```
Wire.onReceive(escolta);
Wire.onRequest(respon);
}

void loop() {
    analin[0]=analogRead(A0);
    analin[1]=analogRead(A1);
    analin[2]=analogRead(A2);
    analin[3]=analogRead(A3);
    analin[4]=analogRead(A6);
    analin[5]=analogRead(A7);
    for (i=0;i<12;i++){
        if ((digm[i]==INPUT) || (digm[i]==INPUT_PULLUP)){
            digd[i]=digitalRead(i+2);
        }
    }
}

void escolta(int ordrelen) {
    ordre = Wire.read();
    if (ordrelen>1) {
        dada = Wire.read();
        switch(ordre) {
            case I2CsapmR1_M2:
                pinMode(2,dada);
                break;
            case I2CsapmR1_M3:
                pinMode(3,dada);
                break;
            case I2CsapmR1_M4:
                pinMode(4,dada);
                break;
            case I2CsapmR1_M5:
                pinMode(5,dada);
                break;
            case I2CsapmR1_M6:
                pinMode(6,dada);
                break;
            case I2CsapmR1_M7:
                pinMode(7,dada);
                break;
            case I2CsapmR1_M8:
                pinMode(8,dada);
                break;
            case I2CsapmR1_M9:
                pinMode(9,dada);
                break;
            case I2CsapmR1_M10:
                pinMode(10,dada);
                break;
            case I2CsapmR1_M11:
                pinMode(11,dada);
                break;
            case I2CsapmR1_M12:
                pinMode(12,dada);
                break;
            case I2CsapmR1_M13:
                pinMode(13,dada);
                break;
            case I2CsapmR1_D2:
                digitalWrite(2,dada);
                break;
        }
    }
}
```

IoT amb D1 mini (ESP8266) i codi Arduino

```
case I2CsapmR1_D3:
    digitalWrite(3,dada);
    break;
case I2CsapmR1_D4:
    digitalWrite(4,dada);
    break;
case I2CsapmR1_D5:
    digitalWrite(5,dada);
    break;
case I2CsapmR1_D6:
    digitalWrite(6,dada);
    break;
case I2CsapmR1_D7:
    digitalWrite(7,dada);
    break;
case I2CsapmR1_D8:
    digitalWrite(8,dada);
    break;
case I2CsapmR1_D9:
    digitalWrite(9,dada);
    break;
case I2CsapmR1_D10:
    digitalWrite(10,dada);
    break;
case I2CsapmR1_D11:
    digitalWrite(11,dada);
    break;
case I2CsapmR1_D12:
    digitalWrite(12,dada);
    break;
case I2CsapmR1_D13:
    digitalWrite(13,dada);
    break;
}
}

void respon() {
    int resposta;
    switch(ordre) {
        case I2CsapmR1_A0:
            resposta = analin[0];
            break;
        case I2CsapmR1_A1:
            resposta = analin[1];
            break;
        case I2CsapmR1_A2:
            resposta = analin[2];
            break;
        case I2CsapmR1_A3:
            resposta = analin[3];
            break;
        case I2CsapmR1_A6:
            resposta = analin[4];
            break;
        case I2CsapmR1_A7:
            resposta = analin[5];
            break;
        case I2CsapmR1_D2:
            resposta = digd[0];
            break;
        case I2CsapmR1_D3:
            resposta = digd[1];
            break;
    }
}
```

Annexos

```
case I2CsapmR1_D4:  
    resposta = digd[2];  
    break;  
case I2CsapmR1_D5:  
    resposta = digd[3];  
    break;  
case I2CsapmR1_D6:  
    resposta = digd[4];  
    break;  
case I2CsapmR1_D7:  
    resposta = digd[5];  
    break;  
case I2CsapmR1_D8:  
    resposta = digd[6];  
    break;  
case I2CsapmR1_D9:  
    resposta = digd[7];  
    break;  
case I2CsapmR1_D10:  
    resposta = digd[8];  
    break;  
case I2CsapmR1_D11:  
    resposta = digd[9];  
    break;  
case I2CsapmR1_D12:  
    resposta = digd[10];  
    break;  
case I2CsapmR1_D13:  
    resposta = digd[11];  
    break;  
default:  
    resposta = -1;  
    break;  
}  
byte buffer[2];  
buffer[0]=resposta >> 8;  
buffer[1]=resposta & 0xff;  
Wire.write(buffer,2);  
ordre=0;  
}
```

Podem adaptar aquest firmware a les nostres necessitats. Les possibilitats són infinites!

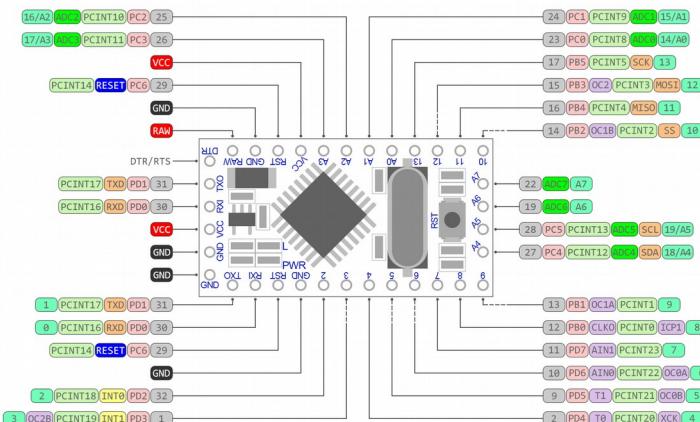
IoT amb D1 mini (ESP8266) i codi Arduino

RobotDyn®

www.robotdyn.com

PINOUT DIAGRAM

**ProMini
ATmega328P**



RobotDyn®
04 Aug 2017

Annex : Afegint la funcionalitat de FSManager als nostres projectes

Com vau veure a Incloure eines HTML5 a la SD virtual si voleu afegir FSManager al vostre programa només cal copiar els arxius FSManager.cpp i FSManager.h a la carpeta del vostre programa i afegir al principi les línies:

```
#include <FS.h>
#include "FSManager.h"
extern ESP8266WebServer server;
```

i al *setup()*

```
SPIFFS.begin();
initHelper(); //inclou connexió Wifi i server
             // Aquí podem afegir altres funcions server.on()
server.begin();
```

I fer servir la funció *espera()* al *loop()* en lloc de *delay()* per poder atendre les peticions web.

Recordeu canviar a l'inici de FSManager.cpp el nom de la vostre xarxa i contrasenya i, si s'escau, el mode WiFi.

Contingut de *FSManager.h*:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#define DBG_OUTPUT_PORT Serial
#include <FS.h>

// Helper functions prototypes
void OkRetorn();
void printDirectory();
void ajaxDirectory();
void handleFileCreate();
void handlefileDelete();
void handleFileUpload();
String formatBytes(size_t bytes);
String getContentType(String filename);
bool handlefileRead(String path);
void handleFileEdit();
void handlefileSave();
void initHelper();
void initWifi();
void pageHead();
void espera(int temps);
```

IoT amb D1 mini (ESP8266) i codi Arduino

Contingut de *FSManager.cpp*:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#define DBG_OUTPUT_PORT Serial
#include <FS.h>

// Helper functions prototypes
void OKRetorn();
void printDirectory();
void ajaxDirectory();
void handleFileCreate();
void handleFileDelete();
void handleFileUpload();
String formatBytes(size_t bytes);
String getContentType(String filename);
bool handleFileRead(String path);
void handleFileEdit();
void handleFileSave();
void initHelper();
void initWifi();
void pageHead();
void espera(int temps);

const char* ssid = "nom de la xarxa";
const char* password = "contrasenya";
const char* host = "esp8266fs";
// tria el tipus de connexió. Client WIFI_STA, Access Point WIFI_AP
#define MODE_WIFI WIFI_STA

ESP8266WebServer server(80);
File fsUploadFile;

void espera(int temps) {
    unsigned long ara = millis();
    unsigned long seguent = ara + temps;
    delay(1); //refresh watchdog
    while (millis()<seguent){
        server.handleClient();
    }
}

void pageHead(){
    server.setContentLength(CONTENT_LENGTH_UNKNOWN);
    server.send(200, "text/html", "");
    server.sendContent("<!DOCTYPE html><html><head><title>Gestió de fitxers</title><meta charset='UTF-8'><meta name='viewport' content='width=device-width'/></head><body>");
}

void initWifi(){
    DBG_OUTPUT_PORT.begin(115200);
    if (MODE_WIFI==WIFI_STA) {
        WiFi.mode(WIFI_STA);
        WiFi.begin(ssid, password);
        DBG_OUTPUT_PORT.printf("Connecting to %s\n", ssid);
        while (WiFi.status() != WL_CONNECTED) {
            delay(500);
            DBG_OUTPUT_PORT.print(".");
        }
        DBG_OUTPUT_PORT.println("");
        DBG_OUTPUT_PORT.print("Connected! IP address: ");
    }
}
```

Annexos

```
DBG_OUTPUT_PORT.println(WiFi.localIP());  
  
DBG_OUTPUT_PORT.print("Open http://");  
DBG_OUTPUT_PORT.print(WiFi.localIP());  
DBG_OUTPUT_PORT.println("/dir to see the file browser");  
}  
else {  
    WiFi.mode(WIFI_AP);  
    WiFi.softAP(ssid, password);  
    DBG_OUTPUT_PORT.print("Open http://192.168.4.1/dir to see the file  
browser");  
}  
}  
  
void initHelper(){  
    DBG_OUTPUT_PORT.print("\n");  
    DBG_OUTPUT_PORT.setDebugOutput(true);  
    Dir dir = SPIFFS.openDir("/");  
    while (dir.next()) {  
        String fileName = dir.fileName();  
        size_t fileSize = dir.fileSize();  
        DBG_OUTPUT_PORT.print("FS File: %s, size: %s\n", fileName.c_str(),  
formatBytes(fileSize).c_str());  
        DBG_OUTPUT_PORT.printf("\n");  
    }  
  
    initWifi();  
  
    //SERVER INIT  
    server.on("/dir", HTTP_GET, printDirectory);  
    server.on("/ajaxdir", HTTP_GET, ajaxDirectory);  
    server.on("/create", HTTP_GET, handleFileCreate);  
    server.on("/delete", HTTP_GET, handleFileDelete);  
    //first callback is called after the request has ended with all parsed  
arguments  
    //second callback handles file uploads at that location  
    server.on("/upload", HTTP_POST, []() {  
        OkRetorn();  
    }, handleFileUpload);  
    server.on("/edit", HTTP_GET, handleFileEdit);  
    server.on("/save", HTTP_POST, handleFileSave);  
    //called when the url is not defined here  
    //use it to load content from SPIFFS  
    server.onNotFound([]() {  
        if (!handleFileRead(server.uri())) {  
            server.send(404, "text/plain", "FileNotFoundException");  
        }  
    });  
    //    server.begin();  
  
}  
  
void OkRetorn() {  
    pageHead();  
    server.sendContent("Operaci&acute; realitzada amb &egrave;xit<br/><a  
href='/dir'>Tornar a gesti&acute; de fitxers</a></body></html>");  
}  
  
void printDirectory(){  
    Dir dir = SPIFFS.openDir("/");  
    pageHead();  
    server.sendContent("<h1>Gesti&acute; de fitxers SD virtual</h1>");  
    while (dir.next()) {
```

IoT amb D1 mini (ESP8266) i codi Arduino

```
String fileName = dir.fileName();
size_t fileSize = dir.fileSize();
DBG_OUTPUT_PORT.printf("FS File: %s, size: %s\n", fileName.c_str(),
formatBytes(fileSize).c_str());
String output;
output += "<a href='/";
output += fileName.substring(1);
output += ">";
output += fileName.substring(1);
output += "</a> ";
output += formatBytes(fileSize).c_str();
output += " <a href='";
output += fileName;
output += "?download=1 style='text-decoration: none;'> #9921;
</a>&nbsp;&nbsp;";
output += " <a href='/edit?fe=";
output += fileName;
output += "' style='text-decoration: none;'> #9997; </a>&nbsp;&nbsp;";
output += " <a href='/delete?killfitxer=";
output += fileName;
output += "' style='text-decoration: none;'> #10006; </a><br/>";
server.sendContent(output);
DBG_OUTPUT_PORT.println(output);

}

FSInfo fs_info;
SPIFFS.info(fs_info);
String output;
output += formatBytes(fs_info.usedBytes).c_str();
output += " ocupats de ";
output += formatBytes(fs_info.totalBytes).c_str();
output += " totals";
output += "<br/><br/>";
server.sendContent(output);

server.sendContent("<br/><form action='/create'>Fitxer nou: <input
type='text' name='noufitxer' value=' '> <input type='submit'
value='Crea'></form>");
server.sendContent("<br/><form method='post' enctype='multipart/form-data'
action='/upload'>Fitxer a enviar: <input type='file' name='myFile'><input
type='submit' value='Envia'></form>");
server.sendContent("</body></html>");
}

void ajaxDirectory(){
server.setContentLength(CONTENT_LENGTH_UNKNOWN);
server.send(200, "text/plain", "");
Dir dir = SPIFFS.openDir("/");
while (dir.next()) {
    String fileName = dir.fileName();
    String output;
    output += fileName;
    output += ",";
    server.sendContent(output);
}
}

void handleFileCreate() {
if (server.args() == 0) {
    return server.send(500, "text/plain", "BAD ARGS");
}
String path = "/" + server.arg("noufitxer");
DBG_OUTPUT_PORT.println("handleFileCreate: " + path);
```

Annexos

```
if (path == "/") {
    return server.send(500, "text/plain", "BAD PATH");
}
if (SPIFFS.exists(path)) {
    return server.send(500, "text/plain", "FILE EXISTS");
}
File file = SPIFFS.open(path, "w");
if (file) {
    file.close();
} else {
    return server.send(500, "text/plain", "CREATE FAILED");
}
OkReturn();
path = String();
}

void handleFileDelete() {
    if (server.args() == 0) {
        return server.send(500, "text/plain", "BAD ARGS");
    }
    String path = server.arg("killfitxer");
    DBG_OUTPUT_PORT.println("handleFileDelete: " + path);
    if (path == "/") {
        return server.send(500, "text/plain", "BAD PATH");
    }
    if (!SPIFFS.exists(path)) {
        return server.send(404, "text/plain", "FileNotFoundException");
    }
    SPIFFS.remove(path);
    OkReturn();
    path = String();
}

void handleFileUpload() {
    if (server.uri() != "/upload") {
        return;
    }
    HTTPUpload& upload = server.upload();
    if (upload.status == UPLOAD_FILE_START) {
        String filename = upload.filename;
        if (!filename.startsWith("/")) {
            filename = "/" + filename;
        }
        DBG_OUTPUT_PORT.print("handleFileUpload Name: ");
        DBG_OUTPUT_PORT.println(filename);
        fsUploadFile = SPIFFS.open(filename, "w");
        filename = String();
    } else if (upload.status == UPLOAD_FILE_WRITE) {
        //DBG_OUTPUT_PORT.print("handleFileUpload Data: ");
        DBG_OUTPUT_PORT.println(upload.currentSize);
        if (fsUploadFile) {
            fsUploadFile.write(upload.buf, upload.currentSize);
        }
    } else if (upload.status == UPLOAD_FILE_END) {
        if (fsUploadFile) {
            fsUploadFile.close();
        }
        DBG_OUTPUT_PORT.print("handleFileUpload Size: ");
        DBG_OUTPUT_PORT.println(upload.totalSize);
    }
}

String formatBytes(size_t bytes) {
    if (bytes < 1024) {
```

IoT amb D1 mini (ESP8266) i codi Arduino

```
    return String(bytes) + "B";
} else if (bytes < (1024 * 1024)) {
    return String(bytes / 1024.0) + "KB";
} else if (bytes < (1024 * 1024 * 1024)) {
    return String(bytes / 1024.0 / 1024.0) + "MB";
} else {
    return String(bytes / 1024.0 / 1024.0 / 1024.0) + "GB";
}
}

String getContentType(String filename) {
    if (server.hasArg("download")) {
        return "application/octet-stream";
    } else if (filename.endsWith(".htm")) {
        return "text/html";
    } else if (filename.endsWith(".html")) {
        return "text/html";
    } else if (filename.endsWith(".css")) {
        return "text/css";
    } else if (filename.endsWith(".js")) {
        return "application/javascript";
    } else if (filename.endsWith(".png")) {
        return "image/png";
    } else if (filename.endsWith(".gif")) {
        return "image/gif";
    } else if (filename.endsWith(".jpg")) {
        return "image/jpeg";
    } else if (filename.endsWith(".svg")) {
        return "image/svg+xml";
    } else if (filename.endsWith(".ico")) {
        return "image/x-icon";
    } else if (filename.endsWith(".xml")) {
        return "text/xml";
    } else if (filename.endsWith(".pdf")) {
        return "application/x-pdf";
    } else if (filename.endsWith(".zip")) {
        return "application/x-zip";
    } else if (filename.endsWith(".gz")) {
        return "application/x-gzip";
    }
    return "text/plain";
}

bool handleFileRead(String path) {
    DBG_OUTPUT_PORT.println("handleFileRead: " + path);
    if (path.endsWith("/")) {
        path += "index.htm";
    }
    String contentType = getContentType(path);
    String pathWithGz = path + ".gz";
    if (SPIFFS.exists(pathWithGz) || SPIFFS.exists(path)) {
        if (SPIFFS.exists(pathWithGz)) {
            path += ".gz";
        }
        File file = SPIFFS.open(path, "r");
        server.streamFile(file, contentType);
        file.close();
        return true;
    }
    return false;
}

void handleFileEdit() {
    if (server.args() == 0) {
```

Annexos

```
        return server.send(500, "text/plain", "BAD ARGS");
    }
    String path = server.arg("fe");
    DBG_OUTPUT_PORT.println("handleFileEdit: " + path);
    if (path == "/") {
        return server.send(500, "text/plain", "BAD PATH");
    }
    if (!SPIFFS.exists(path)) {
        DBG_OUTPUT_PORT.print("handleFileEdit Name: ");
        DBG_OUTPUT_PORT.print(path);DBG_OUTPUT_PORT.println(" FileNotFound");
        return server.send(404, "text/plain", "FileNotFound");
    }
    pageHead();
    server.sendContent("<form action='/save' method='POST'>Fitxer: <input type='text' name='nomfitxer' value=''");
    server.sendContent(path);
    server.sendContent("<br><textarea name='cos' rows='30' cols='40' wrap='off'>");
    File file = SPIFFS.open(path,"r");
    char buffer[2];
    buffer[1] = 0;
//    server.streamFile(file, "text/plain");
    while (file.readBytes(buffer, 1)!=0 ) {
        server.sendContent(buffer);
    }
    file.close();
    server.sendContent("</textarea><br><input type='submit' value='Desa'></form></body></html>");
    path = String();
}

void handleFileSave() {
    String path = server.arg("nomfitxer");
    String text = server.arg("cos");
    DBG_OUTPUT_PORT.println("handleFileSave: " + path);
    DBG_OUTPUT_PORT.println("text: " + text);
    File file = SPIFFS.open(path,"w");
    if (!file) {
        DBG_OUTPUT_PORT.println("file open failed");
    }
    file.print(text);
    file.close();
    OkRetorn();
    path = String();
    text = String();
}
```