

IoT amb D1 mini (ESP8266) i codi Arduino



Jordi Orts

Gener 2019

Índex de continguts

| | |
|--|----|
| Pròleg..... | 4 |
| Introducció..... | 5 |
| El kit D1 mini R1..... | 6 |
| Contingut del kit..... | 6 |
| Connexions i compatibilitat dels shields..... | 12 |
| Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit..... | 13 |
| 1-button shield..... | 17 |
| Relay shield V2.0.0..... | 18 |
| RGB shield..... | 20 |
| Matrix led shield..... | 22 |
| Buzzer shield..... | 24 |
| DHT shield..... | 26 |
| OLED Shield..... | 28 |
| RTC shield..... | 30 |
| PIR shield..... | 32 |
| IR controller Shield..... | 34 |
| Part II. Connectats amb xarxa..... | 36 |
| Connexió com a estació..... | 37 |
| Un servidor web. Controlant el relé via WiFi..... | 38 |
| Connexió com a AP. Control WiFi d'un semàfor..... | 41 |
| AP + estació..... | 44 |
| Estació o AP?..... | 45 |
| ThingSpeak: enregistrar les nostres dades al núvol..... | 46 |
| Part III. Utilitzant components externs..... | 48 |
| Mòdul 6 LEDs..... | 50 |
| Mòdul LED RGB..... | 52 |
| Microservo 3,7g..... | 53 |
| Mòdul 4 polsadors..... | 55 |
| Mòdul potenciòmetre..... | 57 |
| Mòdul sensor temperatura DS18B20..... | 60 |
| Mòdul sensor de llum I2C BH1750FVI..... | 64 |

| | |
|--|-----|
| Mòdul acceleròmetre + giroscopi I2C GY-521..... | 65 |
| Arduino Pro mini com a esclau I2C..... | 68 |
| Part IV. El sistema de fitxers SPIFFS. Utilització a un servidor web. | |
| | 73 |
| Afegint fitxers al nostre servidor web. La funció serveStatic().... | 75 |
| Enregistrar dades a la SD virtual i poder recuperar-les via WiFi. | |
| Generació de fitxers compatibles amb fulls de càcul..... | 78 |
| Llegint dades de la SD virtual..... | 81 |
| FSManager, la navalla suïssa per a la gestió de fitxers a la SD virtual via WiFi..... | 83 |
| Part V. Altres shields D1 mini..... | 90 |
| Part VI. HTML5, javaScript i AJAX..... | 91 |
| Part VII. Projectes..... | 92 |
| Part VIII. Prototips comercials..... | 93 |
| Part IX. Actualització WiFi del programa (OTA)..... | 94 |
| Annexos..... | 95 |
| Annex : Instal·lació Arduino per a ESP8266..... | 96 |
| Annex : Taula de colors RGB..... | 99 |
| Annex: Taula de freqüències per a les notes musicals..... | 101 |
| Annex: Firmware de l'arduino pro mini..... | 102 |

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.

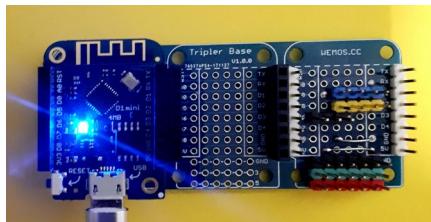
El kit D1 mini R1

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

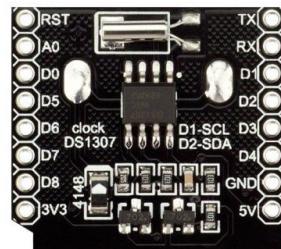
Matrix led shield

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

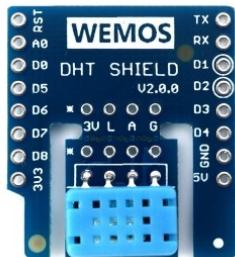
Reconfigurat al pin D7



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

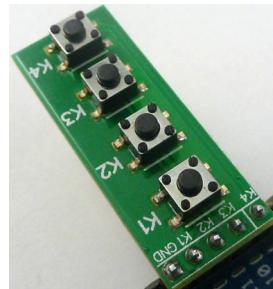


IoT amb D1 mini (ESP8266) i codi Arduino

mòdul 6 LEDs



mòdul 4 polsadors



mòdul sensor de llum I2C
BH1750FVI



mòdul acceleròmetre + giroscopi
I2C GY-521

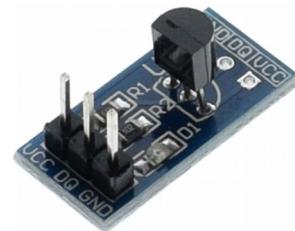


El kit D1 mini R1

mòdul potenciòmetre



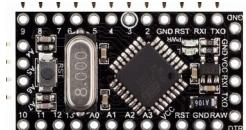
mòdul sensor temperatura
DS18B20



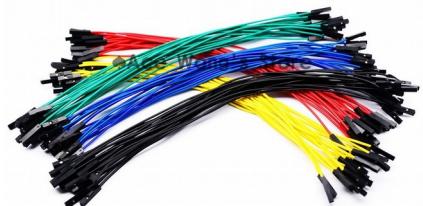
microservo 3,7g



Arduino Pro mini com a esclau
I2C



10 cables Dupont F-F



Connexions i compatibilitat dels shields

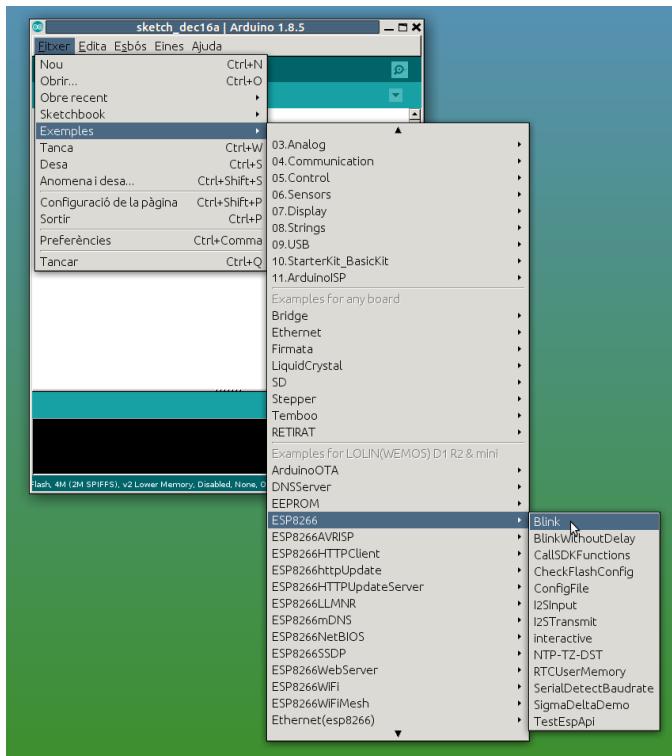
| Shield | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | A0 |
|------------|----|-----|-----|----|----|----|----|----|----|----|
| OLED | | I2C | I2C | x | x | | | | | |
| Matrix led | | | | | | x | | x | | |
| RTC | | I2C | I2C | | | | | | | |
| Buzzer | | | | | | x | | | | |
| 1 button | | | | x | | | | | | |
| 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. The main window displays the code for the "Blink" sketch. The code uses the LED_BUILTIN pin (D1) to control an LED. It initializes the pin as an output in setup() and then enters a loop where it alternates between LOW and HIGH states every second. 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
}

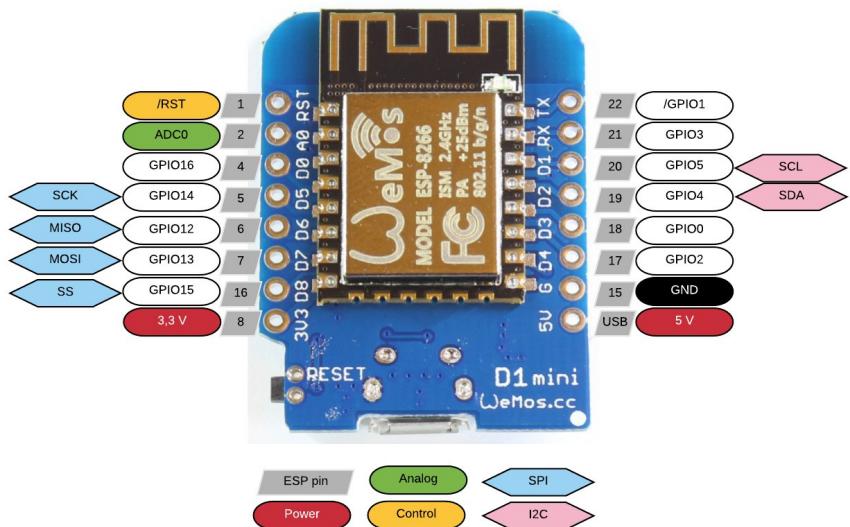
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, LOW);    // Turn the LED on (Note that LOW is the voltage level
  // but actually the LED is on; this is because
  // it is active low on the ESP-01)
  delay(1000);                    // Wait for a second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

At the bottom of the code editor, there is a status bar with the text "LOLIN(WEMOS) D1 R2 & mini, 80 MHz, Flash, 4M (2M SPIFFS), v2 Lower Memory, Disabled, None, Only Sketch, 921600 en /dev/ttyUSB0".

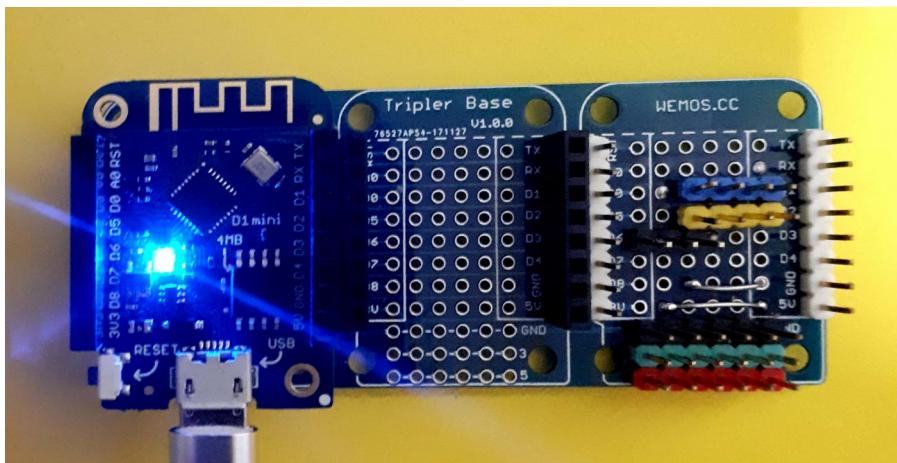
Fixeu-vos com al programa s'utilitza el pin *LED_BUILTIN*. És un sinònim de *D4*, nom del pin serigrafiat a la placa del D1 mini, que en realitat és el pin *GPIO2*. És a dir, si substituïm *LED_BUILTIN* al programa anterior per *D4* o *2*, el programa farà exactament el mateix.

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.



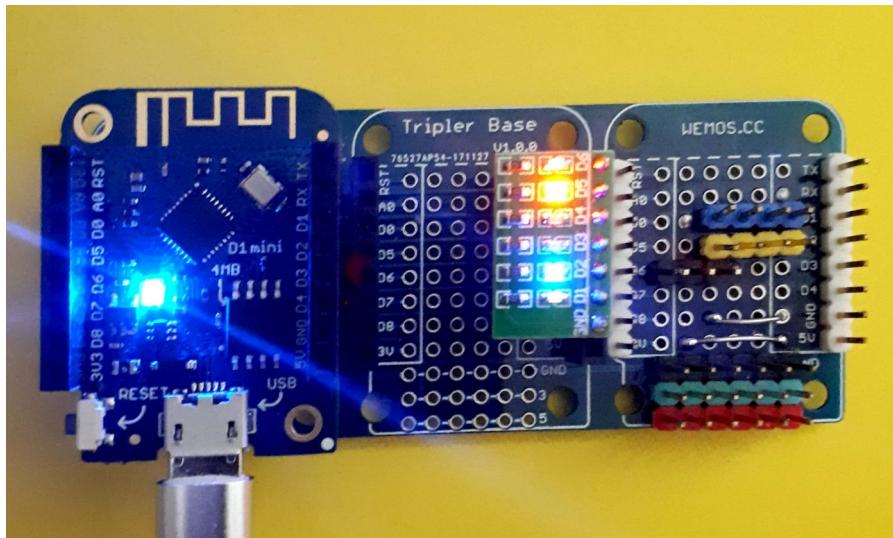
Ja podeu enviar el programa al D1 mini. El led blau integrat farà pampallugues.



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
    }
}
```

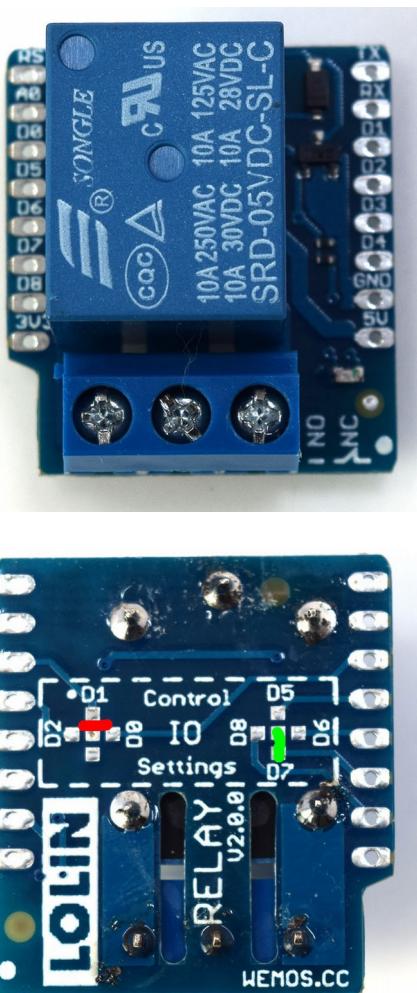


Relay shield V2.0.0

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:

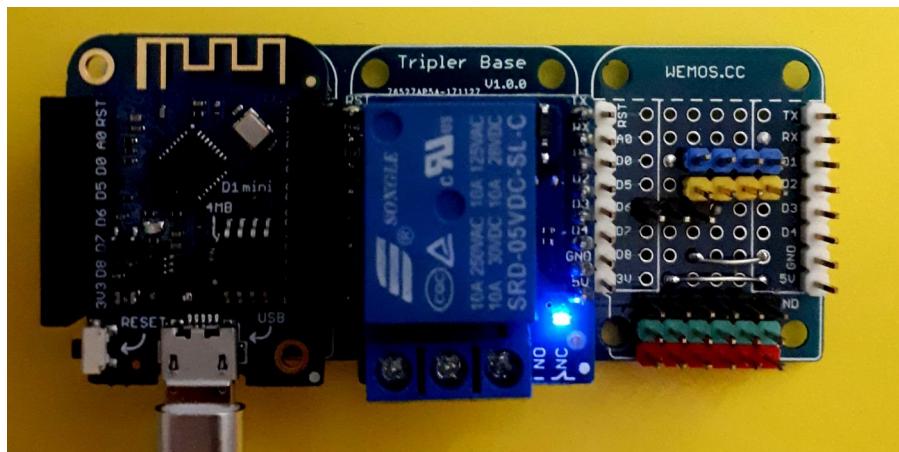


Part I. Programant el D1 mini com si fos un Arduino. Utilitzant els shields del kit.

```
//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
}
```

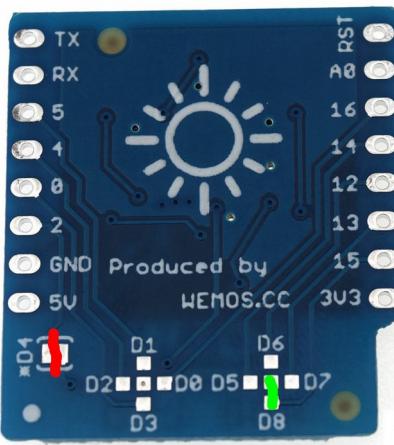


RGB shield

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).
    }
}
```

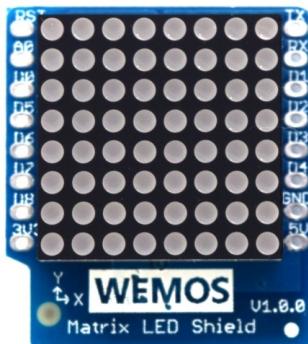


Matrix led shield

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 for 'Sofata d'ent.', 'Classes', 'Institut Priv...', 'Recursos-Tri...', 'Public-Drop...', 'LED Matrix', and a '+' button. Below the header, there's a toolbar with color swatches and a 'Select LED+ color' dropdown. The main area has three sections: a 'Library' section with a list of tools like 'Set N01: Digits / Letters / Signs', 'Set N02: Digits / Letters / Signs / Others', and 'Set N03: Digits / Icons'; a 'Free Writing Tool' section with a text input field and a 'Grammarly' link; and an 'Arduino/C code' editor with a scrollable code block containing hex values. At the bottom, there's a preview area showing a sequence of 8x8 matrices and a 'Use Drag-and-Drop to reorder matrices' note.

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[] = {
  0x00080cfffe0c08,
  0x0004067f7f7f0604,
  0x0002033f3f3f0302,
  0x0001011f1f1f0101,
  0x00000000f0f0f0000,
  0x00000000707070000,
  0x00000000303030000,
  0x00000000101010000,
  0x000000000000000000
};

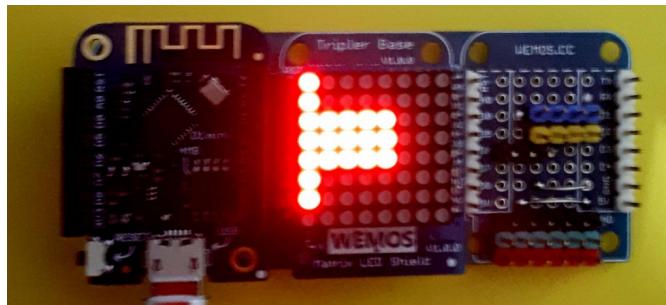
const int IMAGES_LEN = sizeof(IMAGES)/8;

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);
}
```

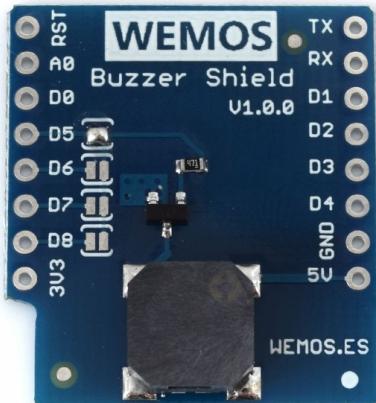


Buzzer shield

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.



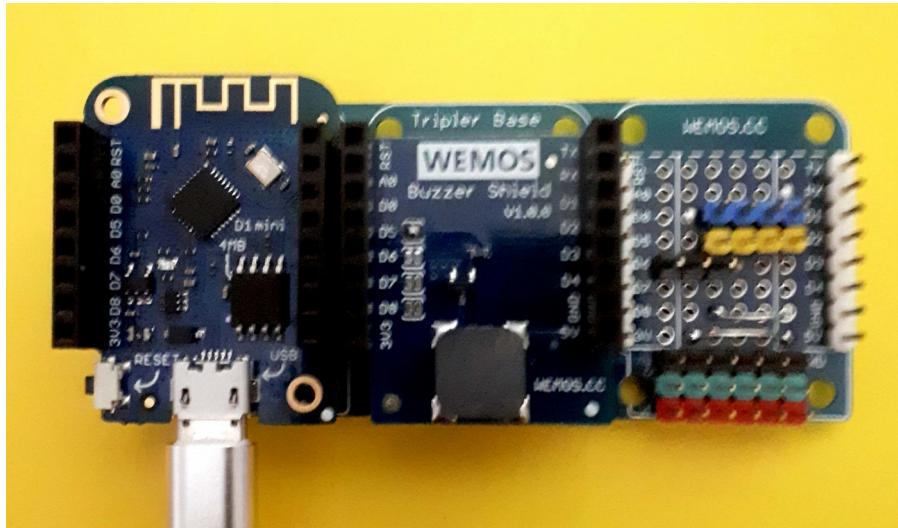
Al meu exemple *sirena* podeu veure com utilitzar aquesta funció:

```
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.



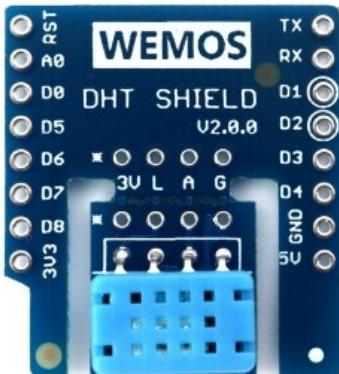
Podeu trobar informació molt útil a l'Annex: Taula de freqüències per a les notes musicals.

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
```



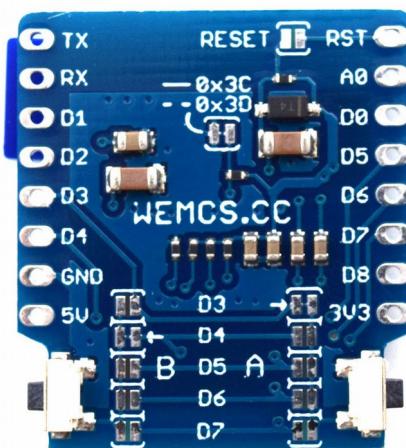
OLED Shield

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.



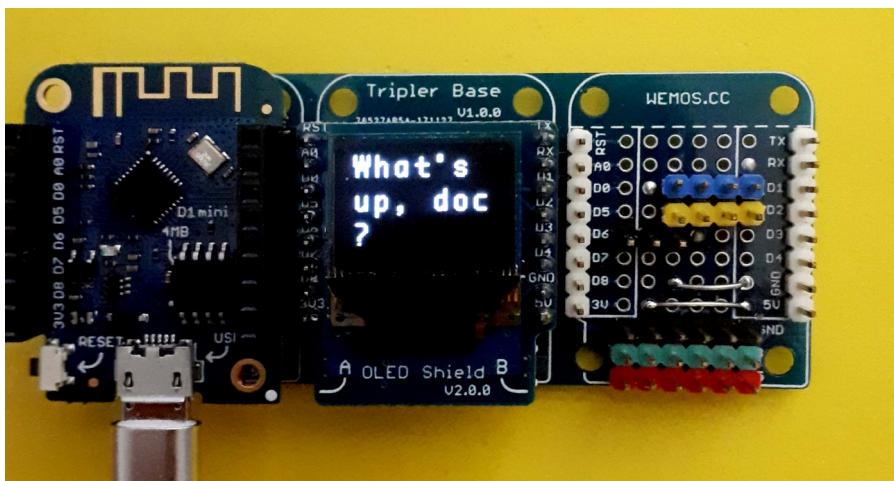
El meu exemple *OLED* inclou tot tipus de funcions. Però per començar crec que serà més pràctic el meu exemple *OLED_test*:

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.
}

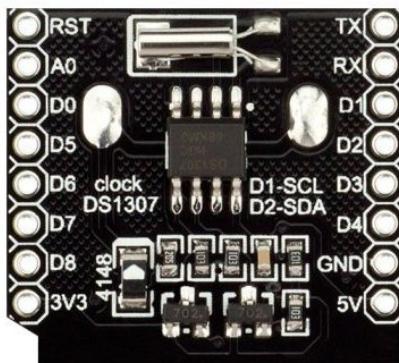
void loop()
{
    oled.clear(PAGE);
    oled.setFontType(1);
    oled.setCursor(0,0);
    oled.println("What's up, doc?");
    oled.display();
    delay(1500);
}
```



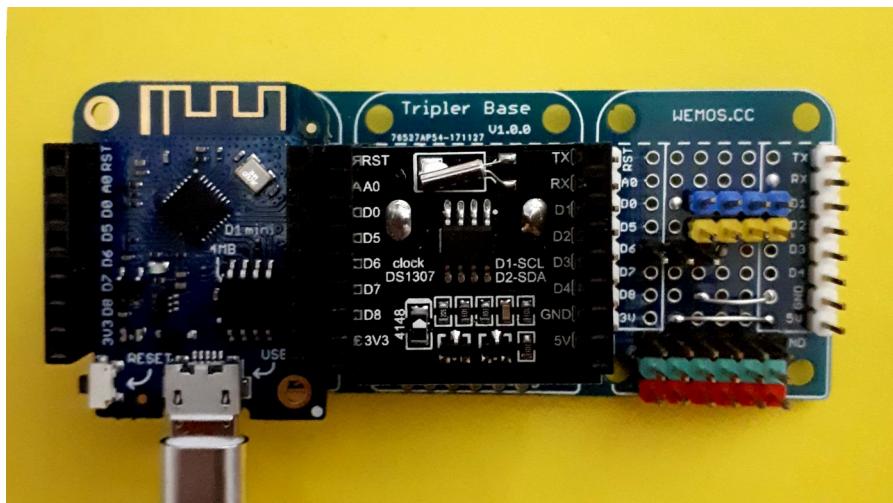
RTC shield

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);
}
```



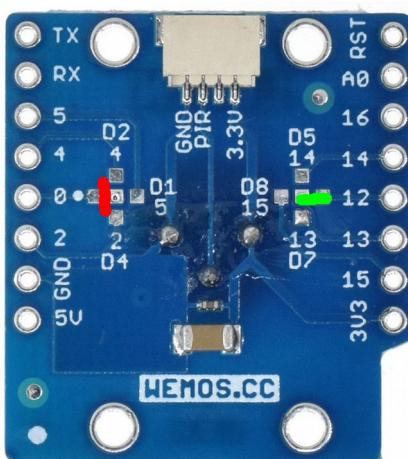
PIR shield

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
  }
}
```



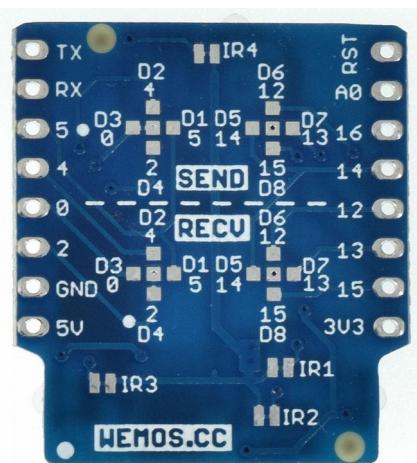
IR controller Shield

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>
```

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.

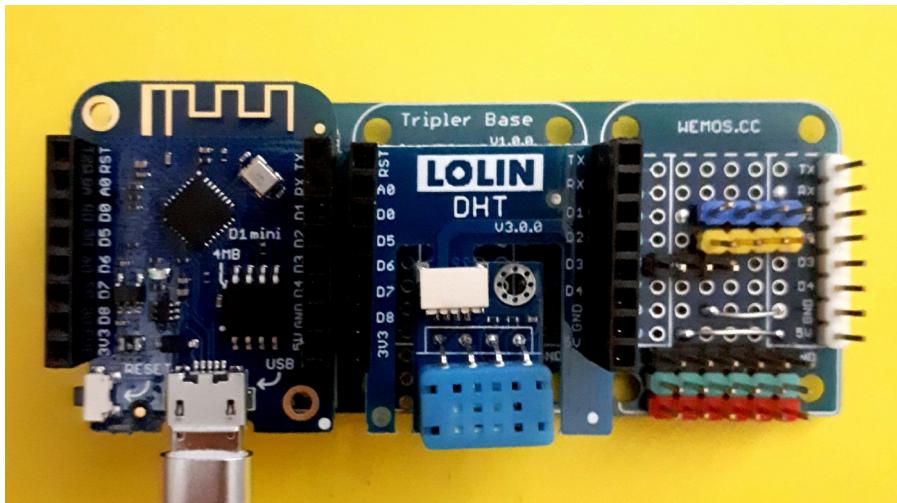
Podeu provar-lo amb el meu exemple *thingspeak*:

```
#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:

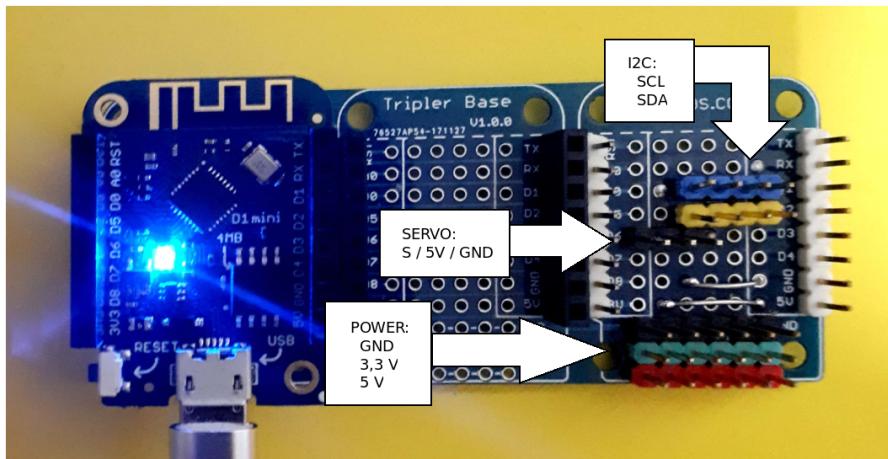
```
#include <ESP8266WiFi.h>
#include "ThingSpeak.h"
#include <WEMOS_DHT12.h>
DHT12 dht12;
const char *ssid = "Nom de la xarxa WiFi aquí";
const char *password = "Contrasenya de la WiFi aquí";
WiFiClient client;
unsigned long myChannelNumber = 185812; //canvia pel teu canal
const char * myWriteAPIKey = "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");
}

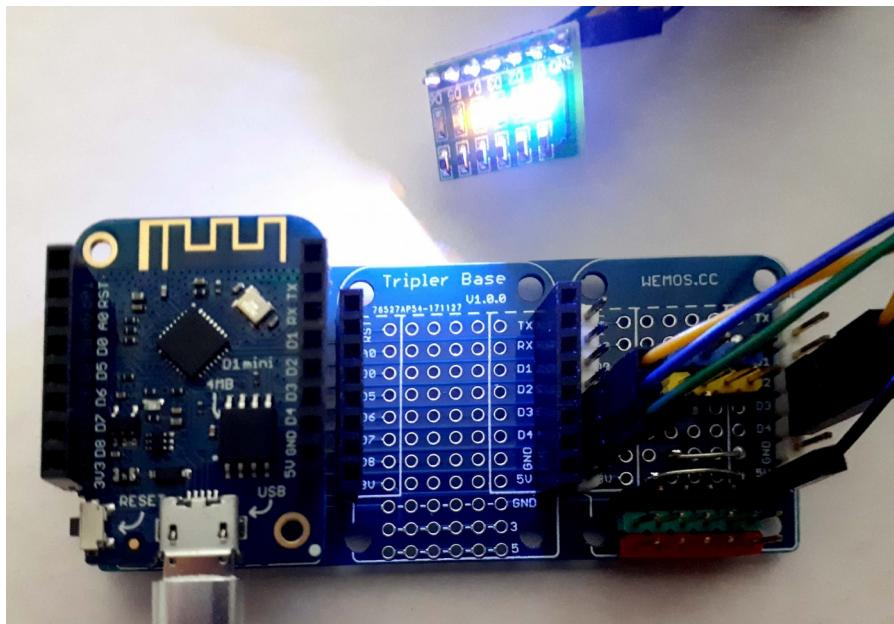
void loop() {
    byte error, address;
    int nDevices;
    Serial.println("Scanning...");
    nDevices = 0;
    for(address = 1; address < 127; address++ )
    {
        Wire.beginTransmission(address);
        error = Wire.endTransmission();
        if (error == 0)
        {
            Serial.print("I2C 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:

```
oid 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);  
}
```

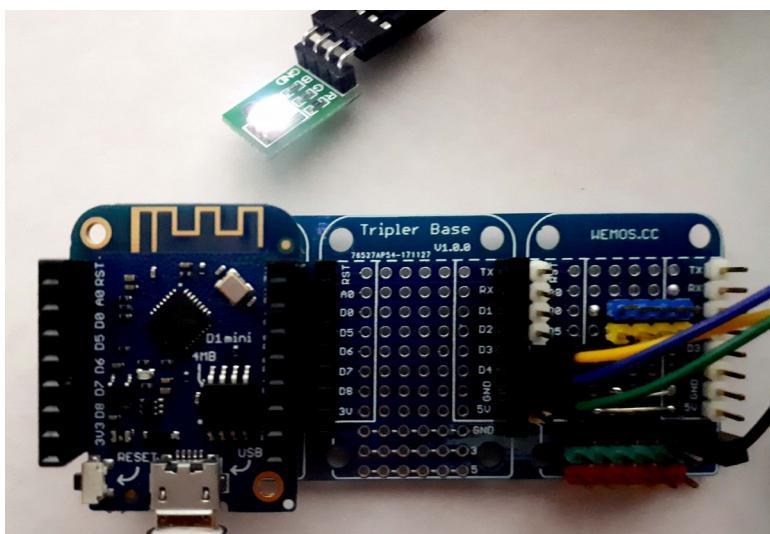
No us ha d'estranyar que quan el led connectat a D4 estigui apagat s'encengui el led blau integrat a l'ESP8266. Recordeu que aquest led està polaritzat a l'inrevés.

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);  
            }  
        }  
    }  
}
```

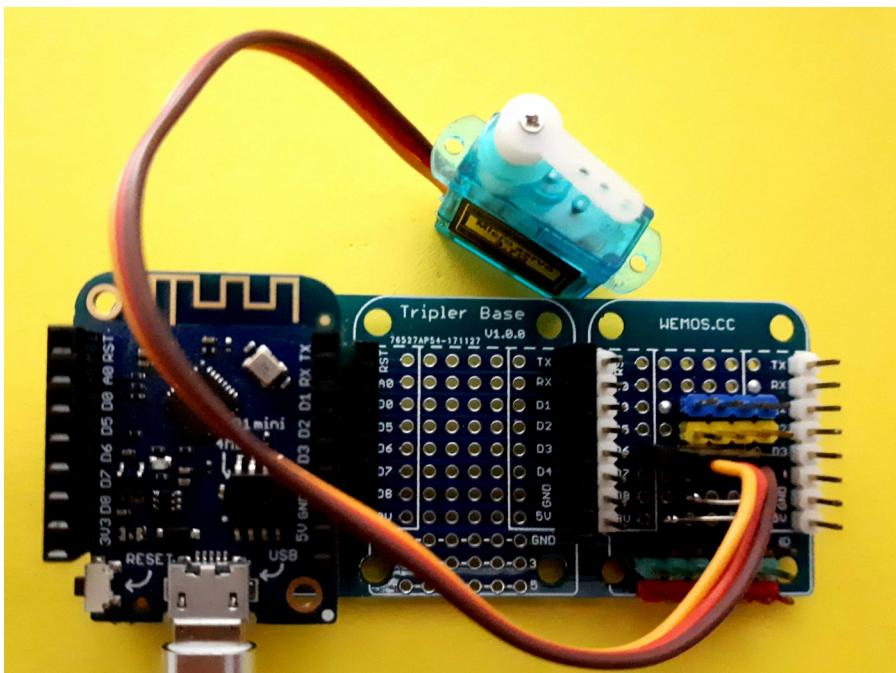


Microservo 3,7g

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);
```

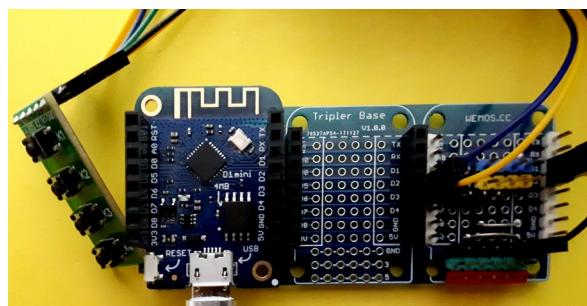
però no es recomanable fer servir aquestes entrades ja que si es fa un RST amb l'entrada a LOW no s'inicia el programa.

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, D2, D5, and D6 all at HIGH. The second set shows pins D1, D2, D5, and D6 all at HIGH. The third set shows pins D1, D2, D5, and D6 all at 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 endings), 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);
}

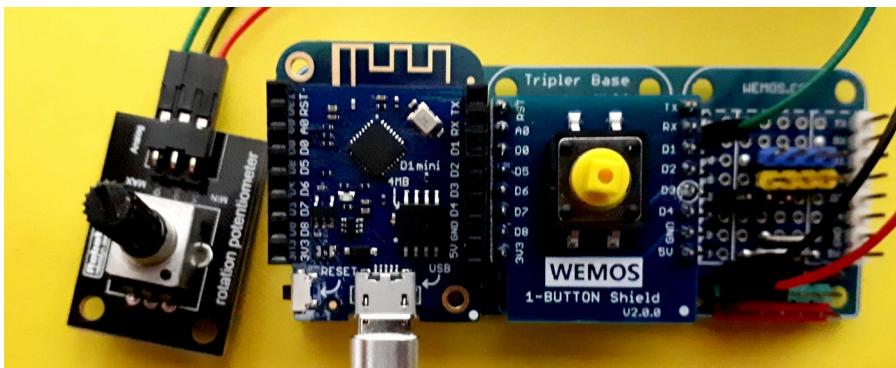
void loop() {
    Serial.print("D1: ");
    if (digitalRead(D1)) {
        Serial.println("HIGH");
    }
    else {
        Serial.println("LOW");
    }
    Serial.print("D2: ");
    if (digitalRead(D2)) {
        Serial.println("HIGH");
    }
    else {
        Serial.println("LOW");
    }
    Serial.print("D5: ");
    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 SVG¹:



1 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);
}

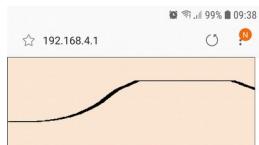
void prova() {
  digitalWrite(ledPin, LOW);
  WiFi.softAP(ssid, password);
  server.on("/", handleRoot);
  server.on("/graf.svg", grafica);
  server.on("/rst", reinicia);
  server.begin();
  while (true) {
    server.handleClient();
  }
}

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

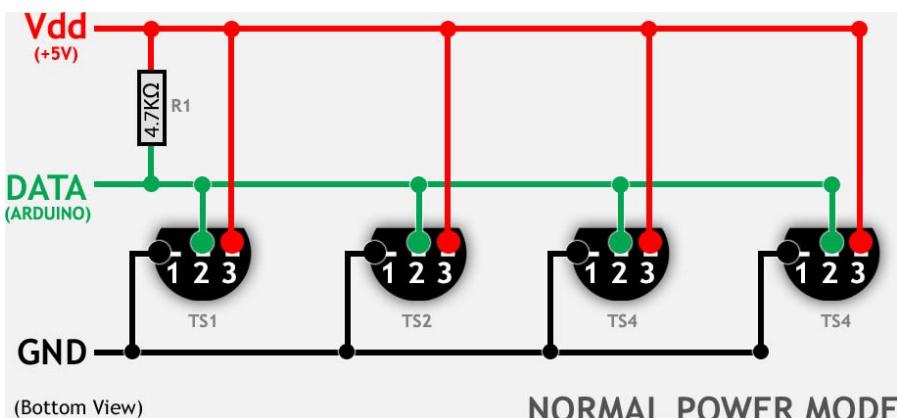
```
    out += temp;
    y = y2;
}
out += "</g>\n</svg>\n";
server.send ( 200, "image/svg+xml", out);
}

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.



Existeix un shield pel D1 mini amb aquest sensor. Malauradament l'han 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.

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

Part III. Utilitzant components externs

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

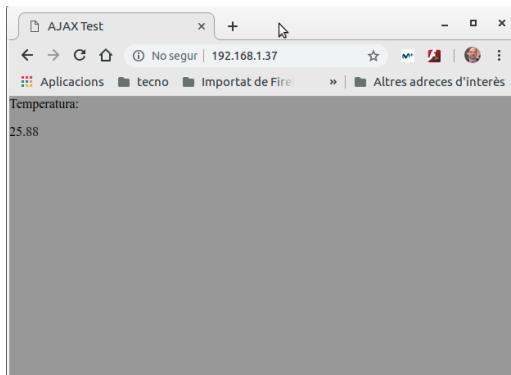
IoT amb D1 mini (ESP8266) i codi Arduino

```
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);
    sprintf(temp, 100, "%d", t1);
    out += temp;
    out += ".";
    t2 = int((t+0.005)*100)-100*t1;
    sprintf(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" );
}

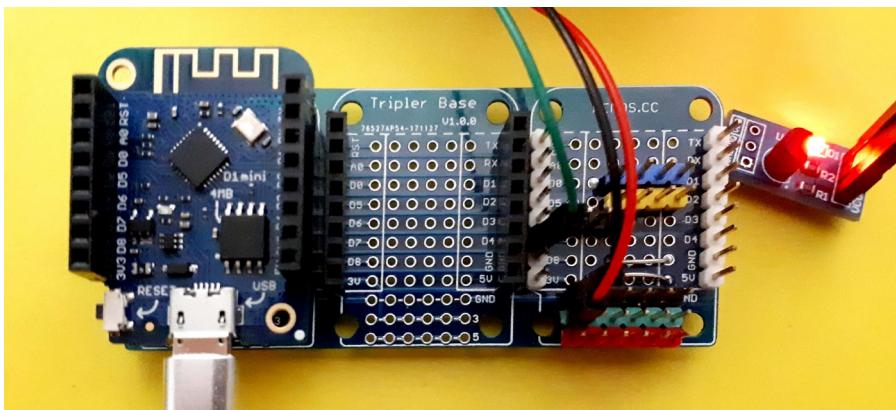
void loop ( void ) {
    server.handleClient();
}
```



Part III. Utilitzant components externs

Fixeu-vos com amb AJAX² 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 periòdicament³ 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).



-
- 2 Podeu trobar més informació sobre com utilitzar AJAX al vostre codi javascript a https://www.w3schools.com/js/js_ajax_intro.asp
 - 3 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:

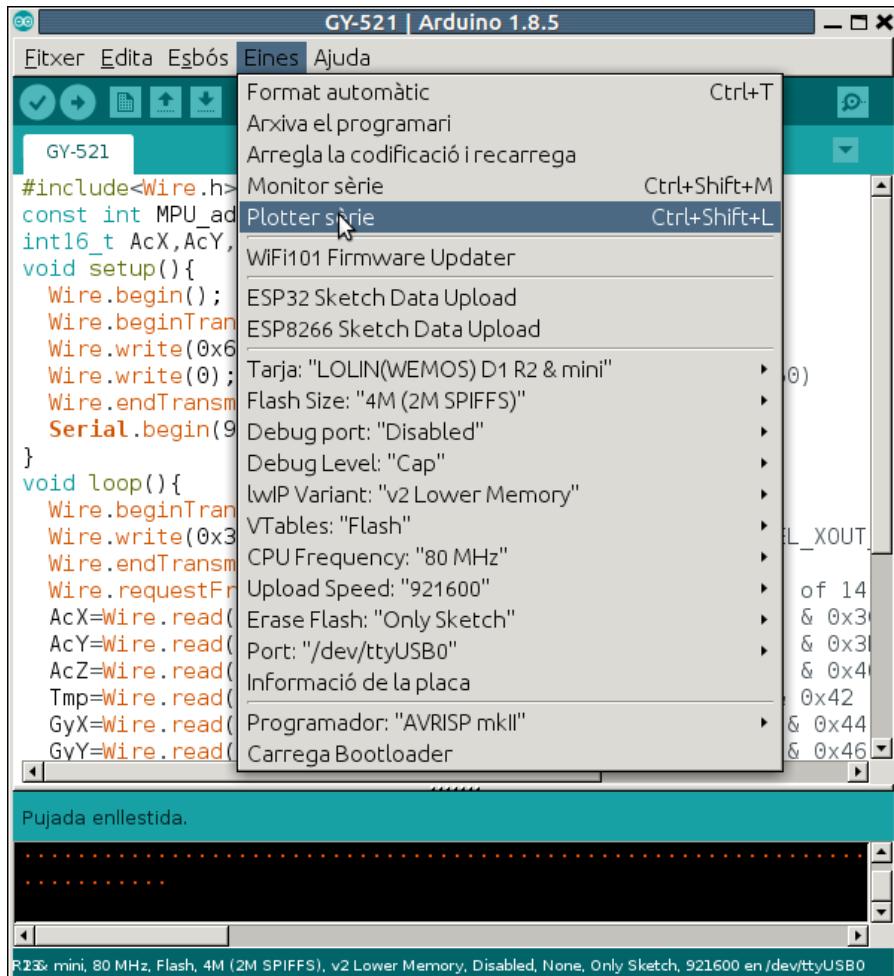
```
//Install [claws/BH1750 Library](https://github.com/claws/BH1750) first.  
#include <Wire.h>  
#include <BH1750.h>  
BH1750 lightMeter(0x23);  
  
void setup(){  
    Serial.begin(9600);  
    Wire.begin();  
    if (lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE)) {  
        Serial.println(F("BH1750 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);  
}
```



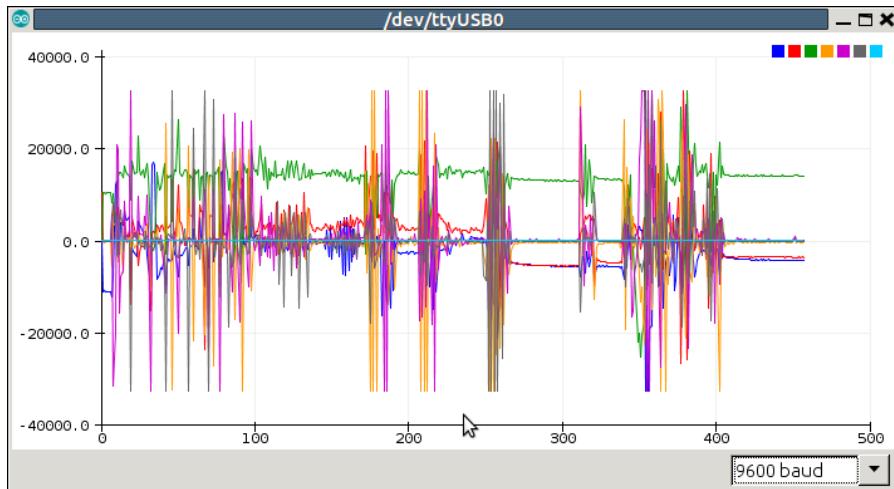
Mòdul acceleròmetre + giroscopi I2C GY-521

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



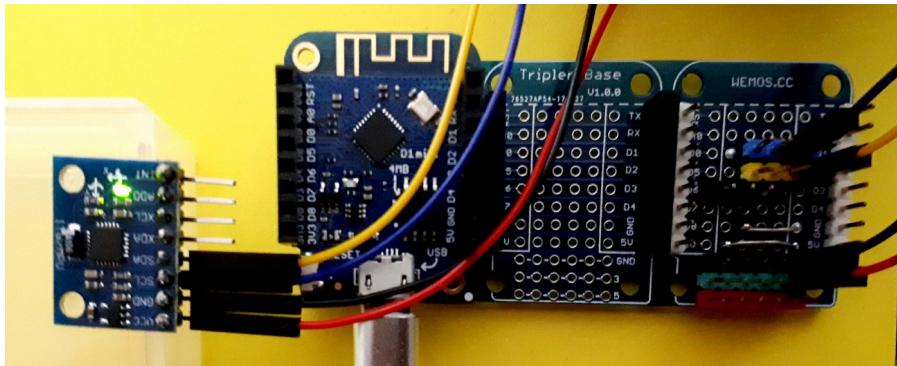
```
#include<Wire.h>
const int MPU_addr=0x68; // I2C address of the MPU-6050
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
void setup(){
    Wire.begin();
    Wire.beginTransmission(MPU_addr);
    Wire.write(0x6B); // PWR_MGMT_1 register
    Wire.write(0); // set to zero (wakes up the MPU-6050)
    Wire.endTransmission(true);
    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!

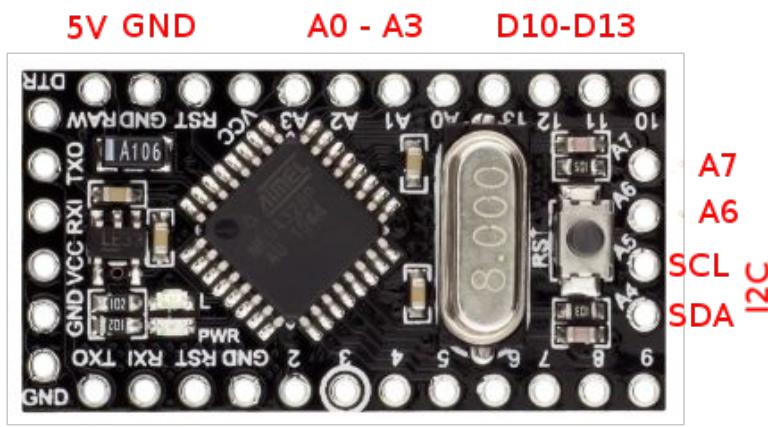
Però existeix una llibreria per a aquest mòdul. Es diu MPU6050, per si voleu provar-la.



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

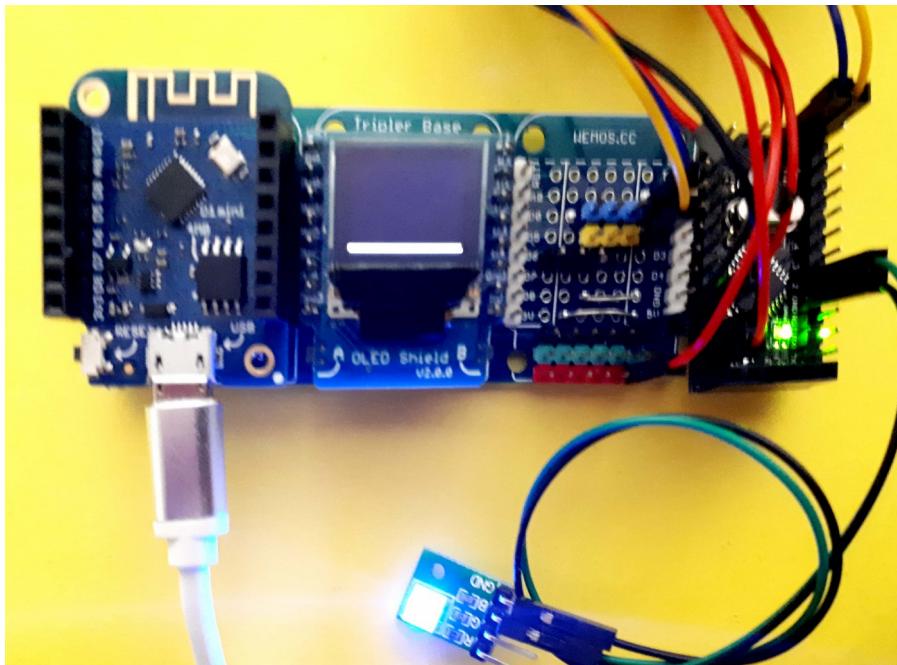
```
#include "I2CsapmR1.h"
#include <Wire.h>

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);
}
```

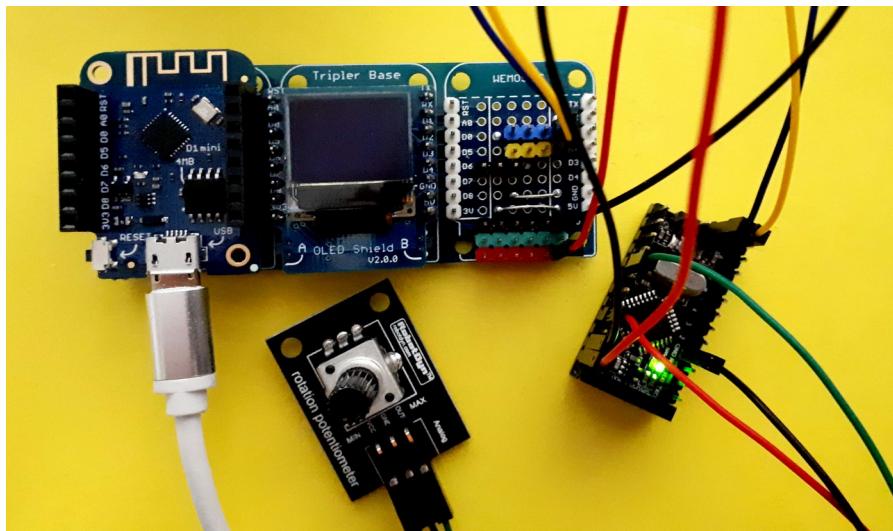
Si connectem un led al pin 2 de l'arduino pro mini veurem que fa pampallugues⁴:



4 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):

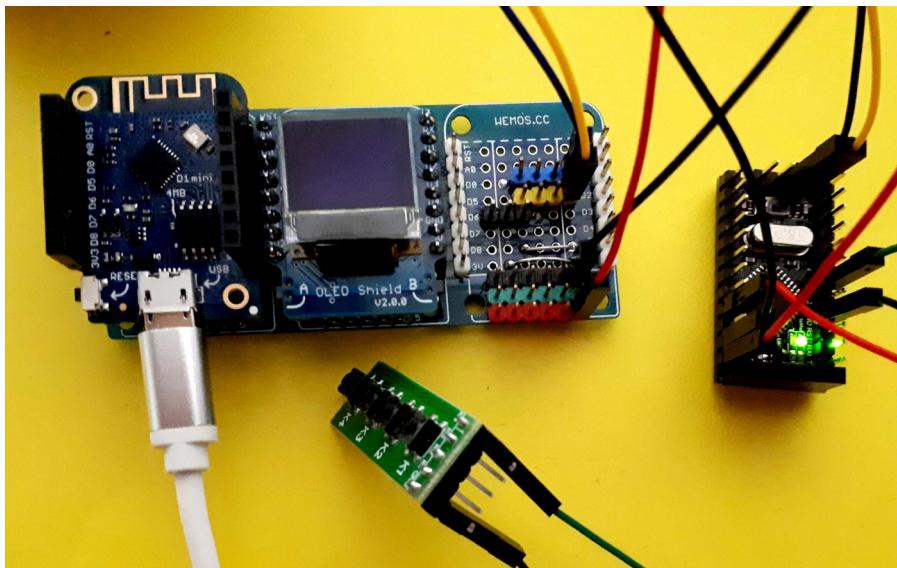


```
/dev/ttyUSB0
703 / 0
566 / 0
461 / 0
397 / 0
354 / 0
338 / 0
412 / 0
552 / 0
651 / 0
731 / 0
775 / 0
852 / 0
838 / 0
686 / 0
564 / 0
```

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



```
00 /dev/ttyUSB0
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 firmware⁵ 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

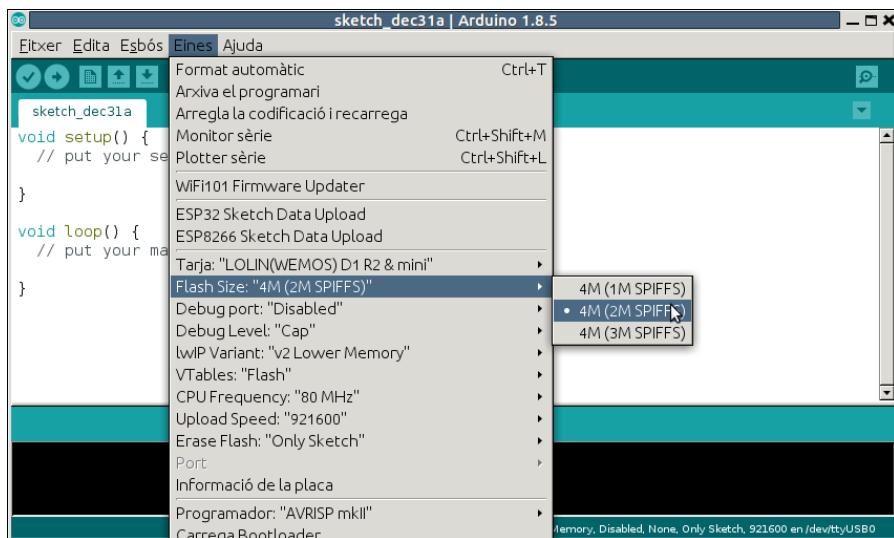
A més a més en podem tenir tants arduino pro mini al bus I2C com vulguem. Només cal canviar l'adreça I2C dels dispositius al seu firmware.

⁵ 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)⁶.



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 Upload⁷, que haurem d'utilitzar

⁶ Veure Part IX. Actualització WiFi del programa (OTA)

⁷ 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 );
}

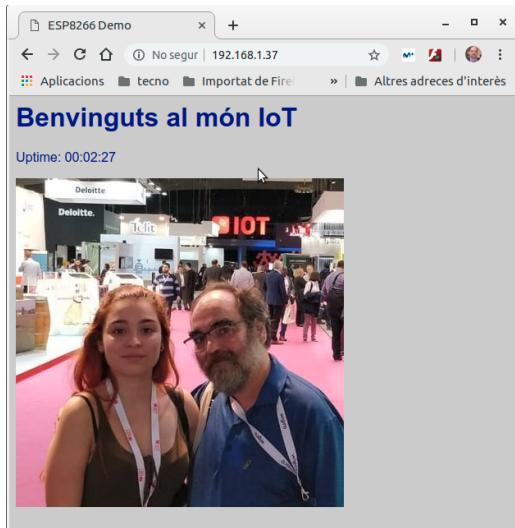
void handleNotFound() {
    String message = "File Not Found\n\n";
    message += "URI: ";
    message += server.uri();
```

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 ofimàtic⁸.

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) {
```

8 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");
}

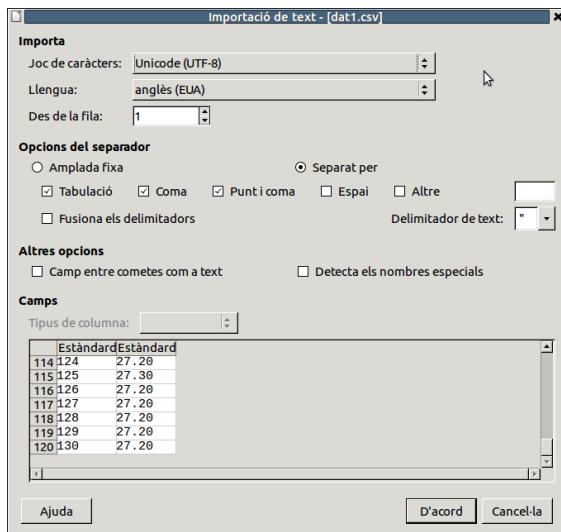
void loop() {
    File f = SPIFFS.open("/dades/dat1.csv","a");
    for(int i=1; i<=100; i++){
        ara = millis();
        if(dht12.get()==0){
            f.print(ara/1000);
            f.print(",");
            f.println(dht12.cTemp);
            espera(1000);
        }
    }
    f.close();
}
```

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.

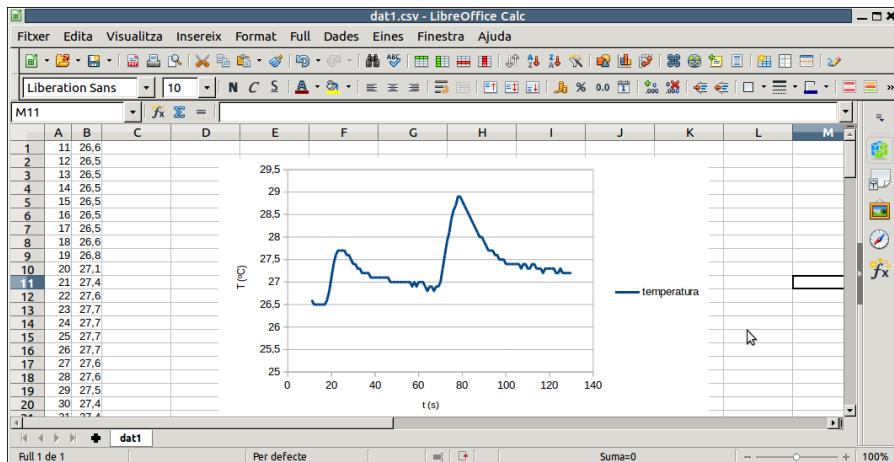
Una vegada enregistrades les dades podem baixar el fitxer amb un navegador apuntant a <http://192.168.1.37/dades/dat1.csv> (en el meu cas va agafar la IP 192.168.1.37).

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*⁹ 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
```

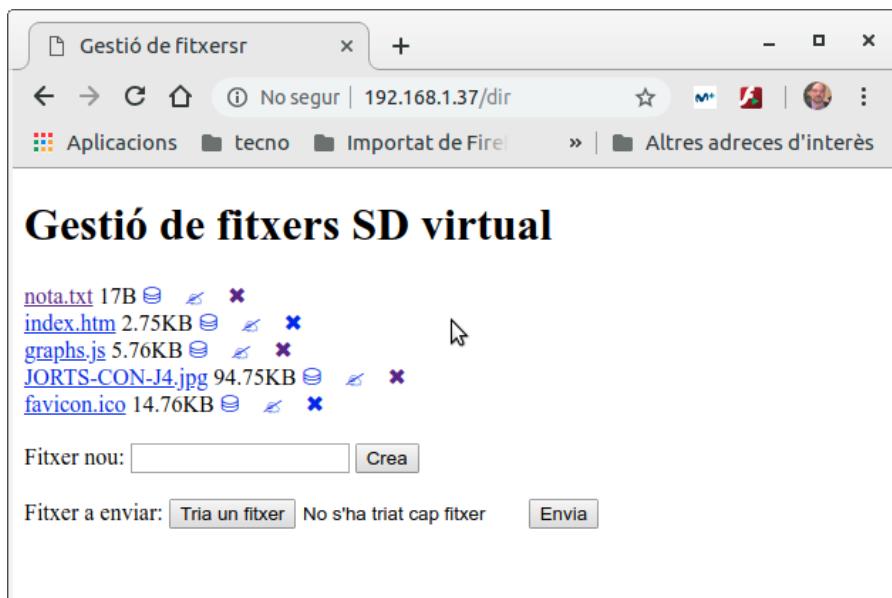
⁹ Disponible a <http://javl.github.io/image2cpp/>. Recordeu fer servir la codificació vertical i una mida de 64 x 48 px.

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

Part VI. HTML5, javaScript i AJAX

Part VII. Projectes

Part VIII. Prototips comercials

Part VIII. Prototips comercials

Part IX. Actualització WiFi del programa (OTA)

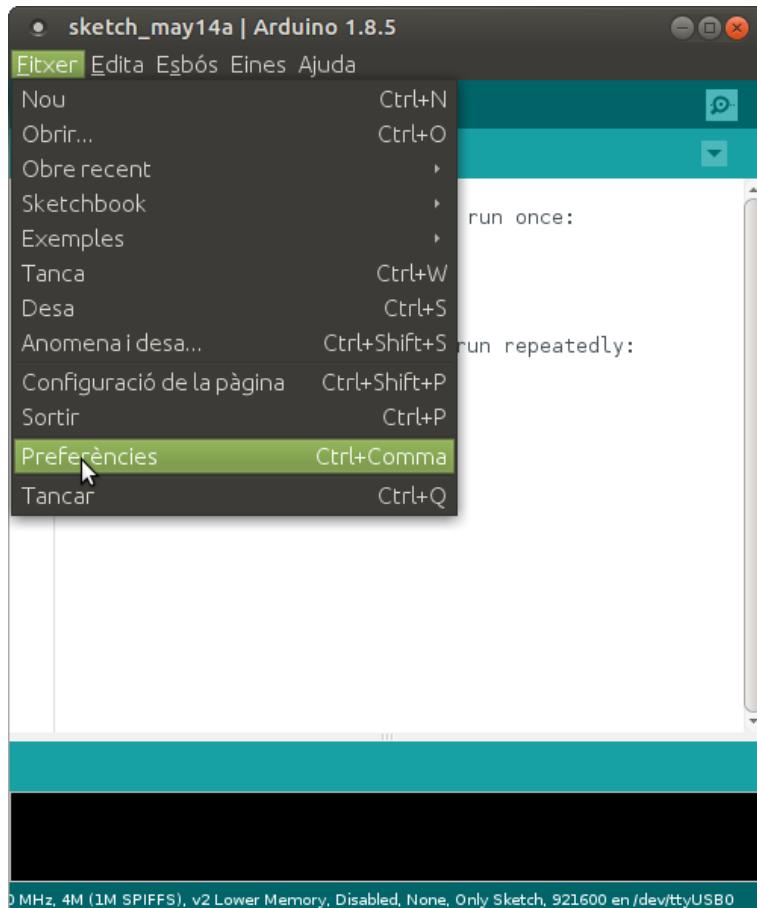
Annexos

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.

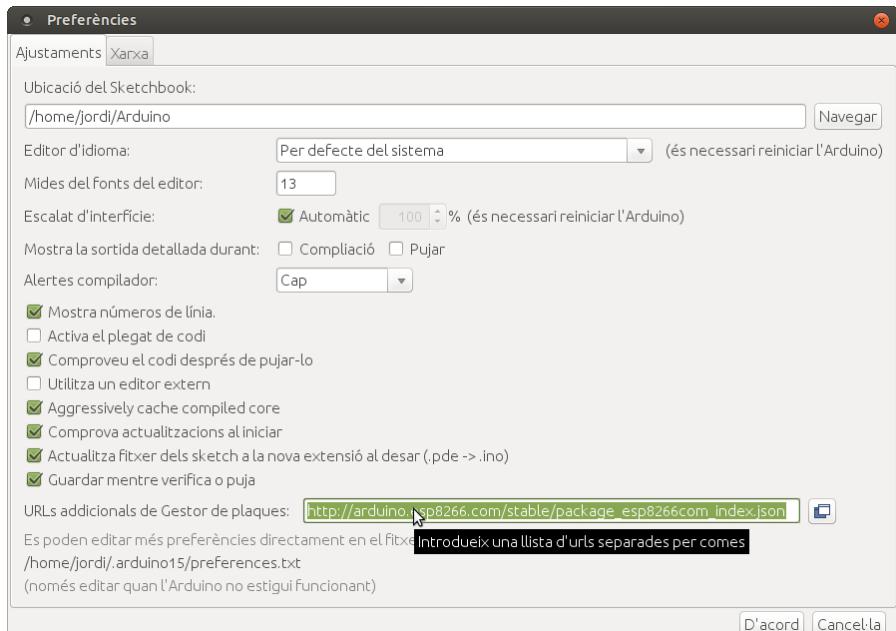
Una vegada instal·lat, hem d'anar a *Fitxer->Preferències*



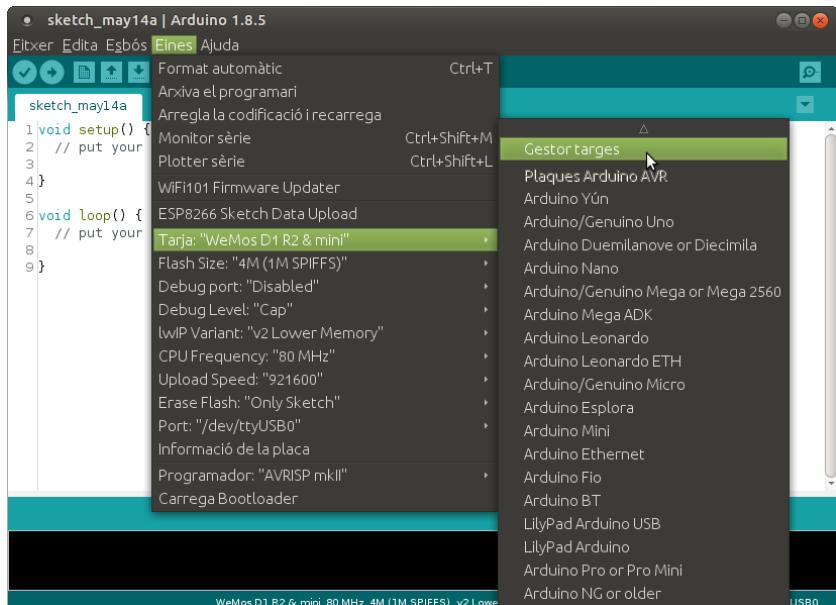
I omplir l'opció URLs adiccionals de Gestors de plaques amb

http://arduino.esp8266.com/stable/package_esp8266com_index.json

Annexos



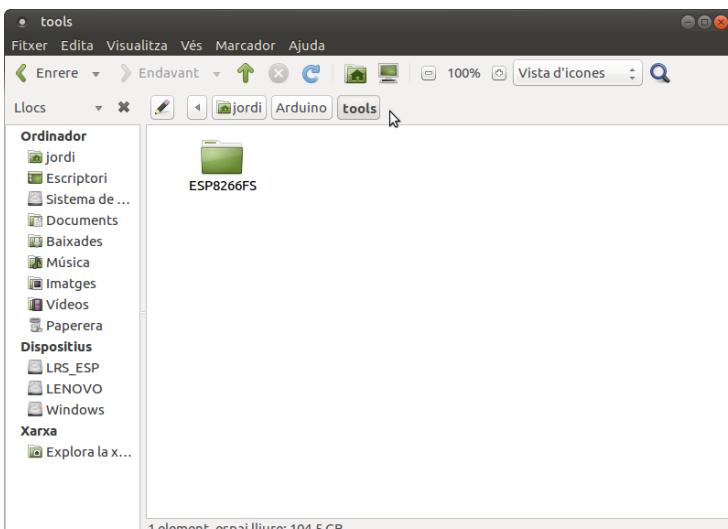
Ara podem anar a *Eines-> Tarja* -> *Gestor targes* i escollir *esp8266* per instal·lar-la



IoT amb D1 mini (ESP8266) i codi Arduino



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
    0xFFFFB6C1, // 1 LightPink
    0xFFFFB4B4, // 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
    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
    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
    0xDA8A520, // 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
    0x6B8E23, // 50 OliveDrab
    0x9ACD32, // 51 YellowGreen
    0x32CD32, // 52 LimeGreen
    0x00FF00, // 53 Lime
    0x7CFC00, // 54 LawnGreen
```

IoT amb D1 mini (ESP8266) i codi Arduino

```
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  
0x66CDAA, // 67 MediumAquamarine  
0x00FFFF, // 68 Aqua  
0x00FFFF, // 69 Cyan  
0xE0FFFF, // 70 LightCyan  
0xAFEEEE, // 71 PaleTurquoise  
0x7FFFD4, // 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  
0xD8BFD8, // 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  
0xFFFFFFF, // 112 White  
0xF5F5DC, // 113 Beige  
0xFAEBD7, // 114 AntiqueWhite  
0xFFE4E1, // 115 MistyRose  
0x808080, // 116 Gray  
0x708090 // 117 SlateGray  
};
```

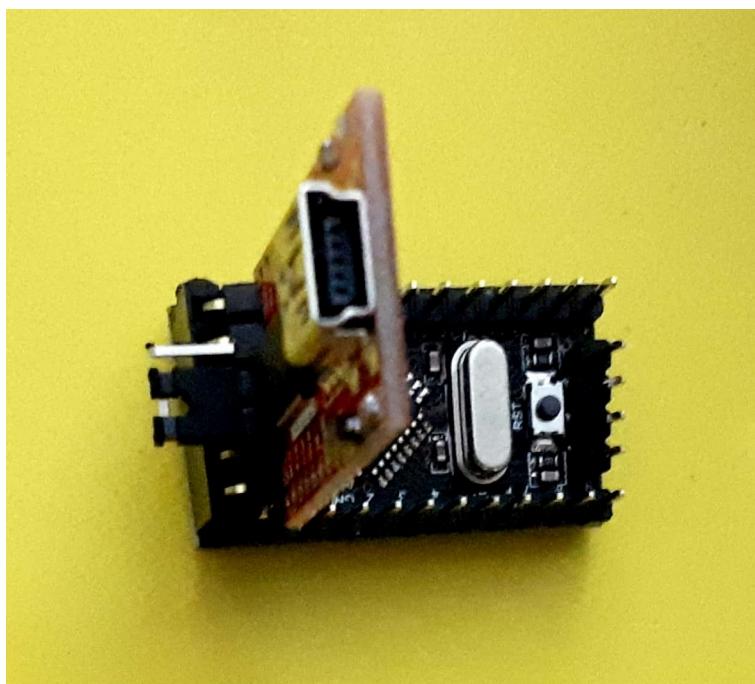
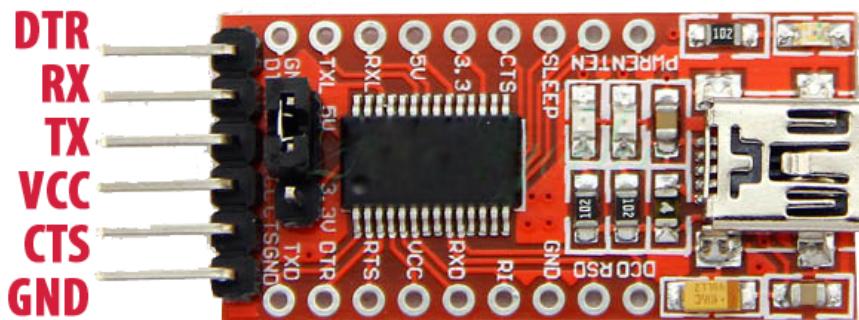
Annex: Taula de freqüències per a les notes musicals

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:



Annexos

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);
```

IoT amb D1 mini (ESP8266) i codi Arduino

```
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;
        }
    }
}
```

Annexos

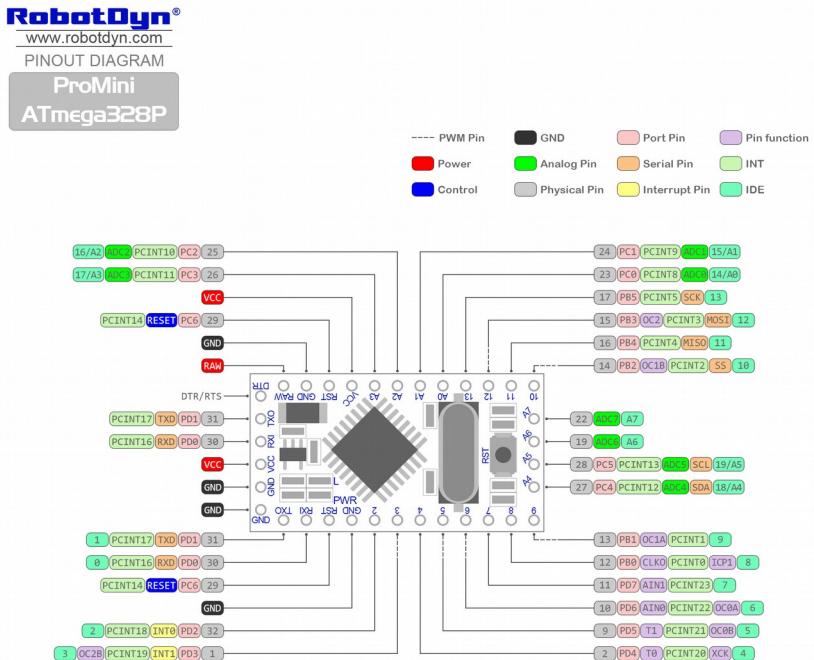
```
case I2CsapmR1_D3:  
    digitalWrite(3,dada);  
    break;  
case I2CsapmR1_D4:  
    digitalWrite(4,dada);  
    break;  
case I2CsapmR1_D5:  
    digitalWrite(5,dada);  
    break;  
case I2CsapmR1_D6:  
    digitalWrite(6,dada);  
    break;  
case I2CsapmR1_D7:  
    digitalWrite(7,dada);  
    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;
```

IoT amb D1 mini (ESP8266) i codi Arduino

```
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!

Annexos



RobotDyn®
04 Aug 2017