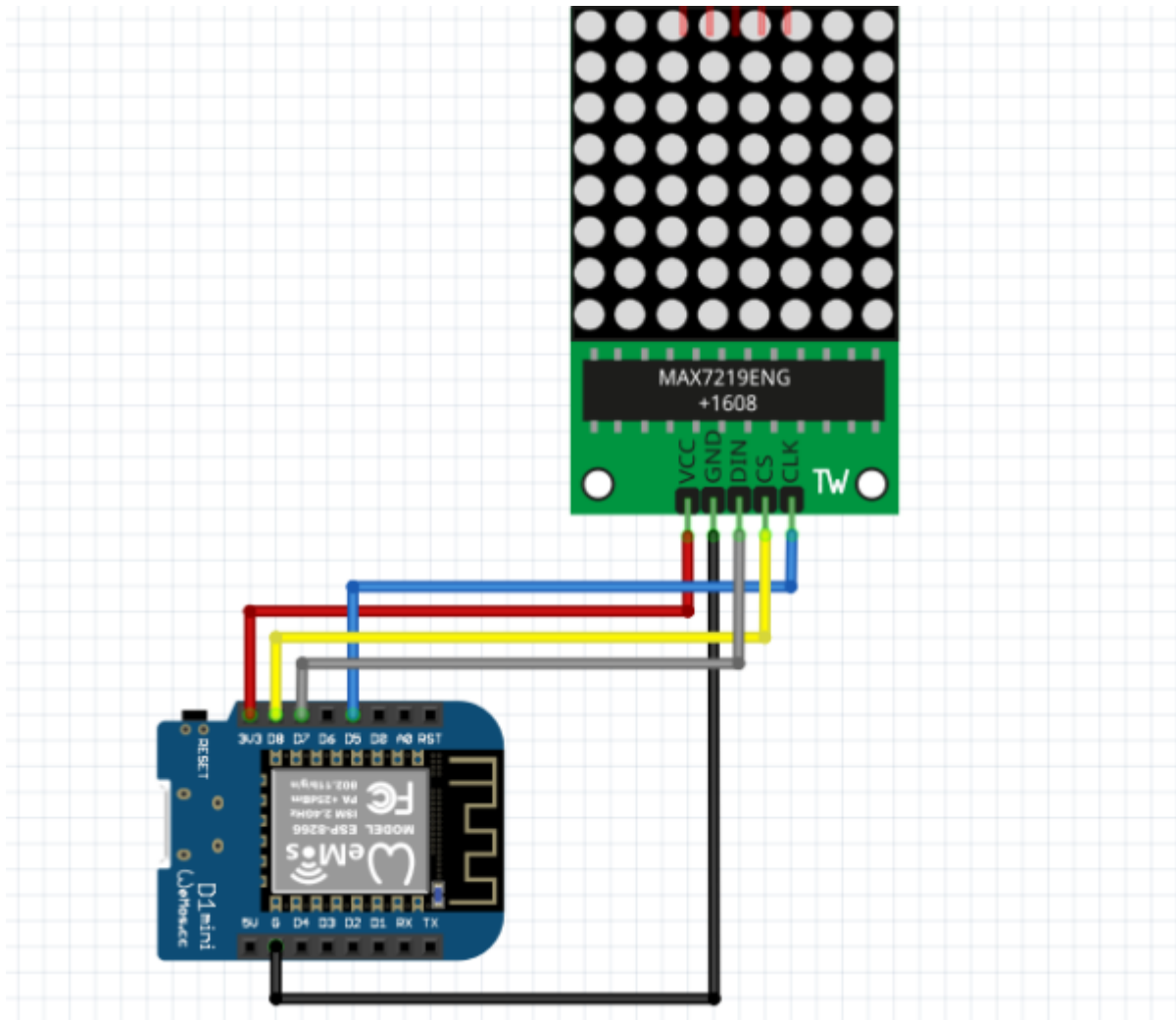


Modulo matrix max 7219

CONEXIÓN





MAX 7219	ESP8266
CLK	D5
CS	D8
DIN	D7
GND	GND
VCC	3V

Code

Este código hace que el esp sirva una página web, donde se escribe un mensaje y se presenta en la pantalla

```
#include <ESP8266WiFi.h>
#include <MD_MAX72xx.h>
#include <SPI.h>

#define PRINT_CALLBACK 0
#define DEBUG 0
#define LED_HEARTBEAT 0

#if DEBUG
#define PRINT(s, v) { Serial.print(F(s)); Serial.print(v); }
#define PRINTS(s) { Serial.print(F(s)); }
#else
#define PRINT(s, v)
#define PRINTS(s)
#endif

#if LED_HEARTBEAT
#define HB_LED D2
#define HB_LED_TIME 500 // in milliseconds
#endif

// Define the number of devices we have in the chain and the hardware
interface
// NOTE: These pin numbers will probably not work with your hardware and may
// need to be adapted
#define HARDWARE_TYPE MD_MAX72XX::FC16_HW
#define MAX_DEVICES 4

#define CLK_PIN D5 // or SCK
#define DATA_PIN D7 // or MOSI
#define CS_PIN D8 // or SS

// SPI hardware interface
MD_MAX72XX mx = MD_MAX72XX(HARDWARE_TYPE, CS_PIN, MAX_DEVICES);
// Arbitrary pins
//MD_MAX72XX mx = MD_MAX72XX(HARDWARE_TYPE, DATA_PIN, CLK_PIN, CS_PIN,
MAX_DEVICES);

// WiFi login parameters - network name and password
const char* ssid = "name wifi";
const char* password = "clave wifi";

// WiFi Server object and parameters
WiFiServer server(80);

// Global message buffers shared by Wifi and Scrolling functions
const uint8_t MMSG_SIZE = 255;
const uint8_t CHAR_SPACING = 1;
const uint8_t SCROLL_DELAY = 75;
```

```
char curMessage[MESG_SIZE];
char newMessage[MESG_SIZE];
bool newMessageAvailable = false;

const char WebResponse[] = "HTTP/1.1 200 OK\nContent-Type: text/html\n\n";

const char WebPage[] =
"<!DOCTYPE html>" \
"<html>" \
"<head><meta name=\"viewport\" content=\"width=device-width, initial-  
scale=1\">" \
"<title>MYTECTUTOR ESP8266 AND MAX7219</title>" \
"<style>" \
"html, body" \
"{ " \
"font-family: Helvetica; " \
"display: block;" \
"margin: 0px auto;" \
"text-align: center;" \
"background-color: #cad9c5;" \
}" \
"#container " \
"{ " \
"width: 100%;" \
"height: 100%;" \
"margin-left: 5px;" \
"margin-top: 20px;" \
"border: solid 2px;" \
"padding: 10px;" \
"background-color: #2dfa53;" \
}" \
"</style>" \
"<script>" \
"strLine = \"\";" \
"function SendText()" \
"{ " \
"  nocache = \"/&nocache=\" + Math.random() * 1000000;" \
"  var request = new XMLHttpRequest();" \
"  strLine = \"&MSG=\" +  
document.getElementById(\"txt_form\").Message.value;" \
"  request.open(\"GET\", strLine + nocache, false);" \
"  request.send(null);" \
}" \
"</script>" \
"</head>" \
"<body>" \
"<H1><b>ESP8266 and MAX7219 LED Matrix WiFi Control</b></H1>" \
<br>
"<div id=\"container\">" \
"<form id=\"txt_form\" name=\"frmText\">" \
"<label>Message:<input type=\"text\" name=\"Message\"
```

```

maxlength=\"255\"></label><br>\" \
\"</form>\" \
\"<br>\" \
\"<input type=\"submit\" value=\"Send Text\" onclick=\"SendText()\">\" \
\"</div>\" \
\"</body>\" \
\"</html>\";

const char *err2Str(wl_status_t code)
{
    switch (code)
    {
        case WL_IDLE_STATUS:    return("IDLE");           break; // WiFi is in
process of changing between statuses
        case WL_NO_SSID_AVAIL:  return("NO_SSID_AVAIL");  break; // case
configured SSID cannot be reached
        case WL_CONNECTED:     return("CONNECTED");      break; // successful
connection is established
        case WL_CONNECT_FAILED: return("CONNECT_FAILED"); break; // password is
incorrect
        case WL_DISCONNECTED:   return("CONNECT_FAILED"); break; // module is not
configured in station mode
        default: return("??");
    }
}

uint8_t htoi(char c)
{
    c = toupper(c);
    if ((c >= '0') && (c <= '9')) return(c - '0');
    if ((c >= 'A') && (c <= 'F')) return(c - 'A' + 0xa);
    return(0);
}

boolean getText(char *szMesg, char *psz, uint8_t len)
{
    boolean isValid = false; // text received flag
    char *pStart, *pEnd;     // pointer to start and end of text

    // get pointer to the beginning of the text
    pStart = strstr(szMesg, "/&MSG=");

    if (pStart != NULL)
    {
        pStart += 6; // skip to start of data
        pEnd = strstr(pStart, "/&");

        if (pEnd != NULL)
        {
            while (pStart != pEnd)
            {

```

```
    if ((*pStart == '%') && isdigit(*(pStart+1)))
    {
        // replace %xx hex code with the ASCII character
        char c = 0;
        pStart++;
        c += (htoi(*pStart++) << 4);
        c += htoi(*pStart++);
        *psz++ = c;
    }
    else
        *psz++ = *pStart++;
}

*psz = '\0'; // terminate the string
isValid = true;
}
}

return(isValid);
}

void handleWiFi(void)
{
    static enum { S_IDLE, S_WAIT_CONN, S_READ, S_EXTRACT, S_RESPONSE,
S_DISCONN } state = S_IDLE;
    static char szBuf[1024];
    static uint16_t idxBuf = 0;
    static WiFiClient client;
    static uint32_t timeStart;

    switch (state)
    {
    case S_IDLE: // initialize
        PRINTS("\nS_IDLE");
        idxBuf = 0;
        state = S_WAIT_CONN;
        break;

    case S_WAIT_CONN: // waiting for connection
        {
            client = server.available();
            if (!client) break;
            if (!client.connected()) break;

#ifdef DEBUG
            char szTxt[20];
            sprintf(szTxt, "%03d:%03d:%03d:%03d", client.remoteIP()[0],
client.remoteIP()[1], client.remoteIP()[2], client.remoteIP()[3]);
            PRINT("\nNew client @ ", szTxt);
#endif
        }
    }
}
```

```
    timeStart = millis();
    state = S_READ;
}
break;

case S_READ: // get the first line of data
PRINTS("\nS_READ");
while (client.available())
{
    char c = client.read();
    if ((c == '\r') || (c == '\n'))
    {
        szBuf[idxBuf] = '\0';
        client.flush();
        PRINT("\nRecv: ", szBuf);
        state = S_EXTRACT;
    }
    else
        szBuf[idxBuf++] = (char)c;
}
if (millis() - timeStart > 1000)
{
    PRINTS("\nWait timeout");
    state = S_DISCONN;
}
break;

case S_EXTRACT: // extract data
PRINTS("\nS_EXTRACT");
// Extract the string from the message if there is one
newMessageAvailable = getText(szBuf, newMessage, MESSG_SIZE);
PRINT("\nNew Msg: ", newMessage);
state = S_RESPONSE;
break;

case S_RESPONSE: // send the response to the client
PRINTS("\nS_RESPONSE");
// Return the response to the client (web page)
client.print(WebResponse);
client.print(WebPage);
state = S_DISCONN;
break;

case S_DISCONN: // disconnect client
PRINTS("\nS_DISCONN");
client.flush();
client.stop();
state = S_IDLE;
break;
```

```
    default: state = S_IDLE;
  }
}

void scrollDataSink(uint8_t dev, MD_MAX72XX::transformType_t t, uint8_t col)
// Callback function for data that is being scrolled off the display
{
#ifdef PRINT_CALLBACK
  Serial.print("\n cb ");
  Serial.print(dev);
  Serial.print(' ');
  Serial.print(t);
  Serial.print(' ');
  Serial.println(col);
#endif
}

uint8_t scrollDataSource(uint8_t dev, MD_MAX72XX::transformType_t t)
// Callback function for data that is required for scrolling into the
display
{
  static enum { S_IDLE, S_NEXT_CHAR, S_SHOW_CHAR, S_SHOW_SPACE } state =
S_IDLE;
  static char *p;
  static uint16_t curLen, showLen;
  static uint8_t cBuf[8];
  uint8_t colData = 0;

  // finite state machine to control what we do on the callback
  switch (state)
  {
  case S_IDLE: // reset the message pointer and check for new message to
load
    PRINTS("\nS_IDLE");
    p = curMessage; // reset the pointer to start of message
    if (newMessageAvailable) // there is a new message waiting
    {
      strcpy(curMessage, newMessage); // copy it in
      newMessageAvailable = false;
    }
    state = S_NEXT_CHAR;
    break;

  case S_NEXT_CHAR: // Load the next character from the font table
    PRINTS("\nS_NEXT_CHAR");
    if (*p == '\0')
      state = S_IDLE;
    else
    {
      showLen = mx.getChar(*p++, sizeof(cBuf) / sizeof(cBuf[0]), cBuf);
      curLen = 0;
    }
  }
}
```

```

    state = S_SHOW_CHAR;
}
break;

case S_SHOW_CHAR: // display the next part of the character
PRINTS("\nS_SHOW_CHAR");
colData = cBuf[curLen++];
if (curLen < showLen)
    break;

// set up the inter character spacing
showLen = (*p != '\0' ? CHAR_SPACING : (MAX_DEVICES*COL_SIZE)/2);
curLen = 0;
state = S_SHOW_SPACE;
// fall through

case S_SHOW_SPACE: // display inter-character spacing (blank column)
PRINT("\nS_ICSPACE: ", curLen);
PRINT("/", showLen);
curLen++;
if (curLen == showLen)
    state = S_NEXT_CHAR;
break;

default:
    state = S_IDLE;
}

return(colData);
}

void scrollText(void)
{
    static uint32_t prevTime = 0;

    // Is it time to scroll the text?
    if (millis() - prevTime >= SCROLL_DELAY)
    {
        mx.transform(MD_MAX72XX::TSL); // scroll along - the callback will load
all the data
        prevTime = millis(); // starting point for next time
    }
}

void setup()
{
#ifdef DEBUG
    Serial.begin(115200);
    PRINTS("\n[MD_MAX72XX WiFi Message Display]\nType a message for the
scrolling display from your internet browser");
#endif
}

```

```
#if LED_HEARTBEAT
  pinMode(HB_LED, OUTPUT);
  digitalWrite(HB_LED, LOW);
#endif

// Display initialization
mx.begin();
mx.setShiftDataInCallback(scrollDataSource);
mx.setShiftDataOutCallback(scrollDataSink);

curMessage[0] = newMessage[0] = '\0';

// Connect to and initialize WiFi network
PRINT("\nConnecting to ", ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED)
{
  PRINT("\n", err2Str(WiFi.status()));
  delay(500);
}
PRINTS("\nWiFi connected");

// Start the server
server.begin();
PRINTS("\nServer started");

// Set up first message as the IP address
sprintf(curMessage, "%03d:%03d:%03d:%03d", WiFi.localIP()[0],
WiFi.localIP()[1], WiFi.localIP()[2], WiFi.localIP()[3]);
PRINT("\nAssigned IP ", curMessage);
}

void loop()
{
#if LED_HEARTBEAT
  static uint32_t timeLast = 0;

  if (millis() - timeLast >= HB_LED_TIME)
  {
    digitalWrite(HB_LED, digitalRead(HB_LED) == LOW ? HIGH : LOW);
    timeLast = millis();
  }
#endif

  handleWiFi();
  scrollText();
}
```

Este otro código es tomado de internet del señor [humberto higinio](#).

```

/*
Programa: Wifi controlled LED matrix display
Autor: Humberto Higinio
Web: www.humbertohiginio.com
Canal de Youtube: https://www.youtube.com/user/HHSolis
Video Exclusivo para mi canal de Youtube
Todos los Derechos Reservados - 2018
Código de Dominio Público

Wemos D1 Mini o NodeMCU pines    -> Matrix pines
MOSI-D7-GPI013    -> DIN
CLK-D5-GPI014    -> Clk
GPI00-D3          -> CS o LOAD

*/

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Max72xxPanel.h>

#define SSID "XXXXXX"           // insert your SSID
#define PASS "XXXXXX"          // insert your password
// ***** String form to sent to the client-browser
// *****

String form =
  "<p>"
  "<center>"
  "<h1>Humberto Higinio Web Server</h1>"
  "<form action='msg'><p>Tipee su mensaje <input type='text' name='msg'"
  "size=100 autofocus> <input type='submit' value='Enviar'></form>"
  "</center>";

ESP8266WebServer server(80);           // HTTP server will
listen at port 80
long period;
int offset=1,refresh=0;
int pinCS = 0; // Attach CS to this pin, DIN to MOSI and CLK to SCK (cf
http://arduino.cc/en/Reference/SPI )
int numberOfHorizontalDisplays = 8;
int numberOfVerticalDisplays = 1;
String decodedMsg;
Max72xxPanel matrix = Max72xxPanel(pinCS, numberOfHorizontalDisplays,
numberOfVerticalDisplays);

String tape = "Arduino";
int wait = 20; // In milliseconds

```

```
int spacer = 2;
int width = 5 + spacer; // The font width is 5 pixels

/*
  handles the messages coming from the webbrowser, restores a few special
  characters and
  constructs the strings that can be sent to the oled display
*/
void handle_msg() {

  matrix.fillScreen(Low);
  server.send(200, "text/html", form); // Send same page so they can send
another msg
  refresh=1;
  // Display msg on Oled
  String msg = server.arg("msg");
  Serial.println(msg);
  decodedMsg = msg;
  // Restore special characters that are misformed to %char by the client
browser
  decodedMsg.replace("+", " ");
  decodedMsg.replace("%21", "!");
  decodedMsg.replace("%22", "");
  decodedMsg.replace("%23", "#");
  decodedMsg.replace("%24", "$");
  decodedMsg.replace("%25", "%");
  decodedMsg.replace("%26", "&");
  decodedMsg.replace("%27", "'");
  decodedMsg.replace("%28", "(");
  decodedMsg.replace("%29", ")");
  decodedMsg.replace("%2A", "*");
  decodedMsg.replace("%2B", "+");
  decodedMsg.replace("%2C", ",");
  decodedMsg.replace("%2F", "/");
  decodedMsg.replace("%3A", ":");
  decodedMsg.replace("%3B", ";");
  decodedMsg.replace("%3C", "<");
  decodedMsg.replace("%3D", "=");
  decodedMsg.replace("%3E", ">");
  decodedMsg.replace("%3F", "?");
  decodedMsg.replace("%40", "@");
  //Serial.println(decodedMsg); // print original string
to monitor

  //Serial.println(' '); // new line in monitor
}

void setup(void) {
matrix.setIntensity(10); // Use a value between 0 and 15 for brightness
```

```

// Adjust to your own needs
// matrix.setPosition(0, 1, 0); // The first display is at <0, 0>
// matrix.setPosition(1, 0, 0); // The second display is at <1, 0>

// Adjust to your own needs
matrix.setPosition(0, 7, 0); // The first display is at <0, 7>
matrix.setPosition(1, 6, 0); // The second display is at <1, 0>
matrix.setPosition(2, 5, 0); // The third display is at <2, 0>
matrix.setPosition(3, 4, 0); // And the last display is at <3, 0>
matrix.setPosition(4, 3, 0); // The first display is at <0, 0>
matrix.setPosition(5, 2, 0); // The second display is at <1, 0>
matrix.setPosition(6, 1, 0); // The third display is at <2, 0>
matrix.setPosition(7, 0, 0); // And the last display is at <3, 0>

matrix.setRotation(0, 3); // The first display is position upside down
matrix.setRotation(1, 3); // The first display is position upside down
matrix.setRotation(2, 3); // The first display is position upside down
matrix.setRotation(3, 3); // The first display is position upside down
matrix.setRotation(4, 3); // The first display is position upside down
matrix.setRotation(5, 3); // The first display is position upside down
matrix.setRotation(6, 3); // The first display is position upside down
matrix.setRotation(7, 3); // The first display is position upside down

//ESP.wdtDisable(); // used to debug, disable
wachdog timer,
Serial.begin(115200); // full speed to monitor

WiFi.begin(SSID, PASS); // Connect to WiFi network
while (WiFi.status() != WL_CONNECTED) { // Wait for connection
  delay(500);
  Serial.print(".");
}
// Set up the endpoints for HTTP server, Endpoints can be written as
inline functions:
server.on("/", []() {
  server.send(200, "text/html", form);
});
server.on("/msg", handle_msg); // And as regular external
functions:
server.begin(); // Start the server

Serial.print("SSID : "); // prints SSID in monitor
Serial.println(SSID); // to monitor

char result[16];

```

```
    sprintf(result, "%3d.%3d.%1d.%3d", WiFi.localIP()[0], WiFi.localIP()[1],
WiFi.localIP()[2], WiFi.localIP()[3]);
    Serial.println();
    Serial.println(result);
    decodedMsg = result;
    Serial.println("WebServer ready!  ");

    Serial.println(WiFi.localIP()); // Serial monitor prints
localIP
    Serial.print(analogRead(A0));
}

void loop(void) {

    for ( int i = 0 ; i < width * decodedMsg.length() + matrix.width() - 1 -
spacer; i++ ) {
        server.handleClient(); // checks for incoming
messages
        if (refresh==1) i=0;
        refresh=0;
        matrix.fillScreen(L0W);

        int letter = i / width;
        int x = (matrix.width() - 1) - i % width;
        int y = (matrix.height() - 8) / 2; // center the text vertically

        while ( x + width - spacer >= 0 && letter >= 0 ) {
            if ( letter < decodedMsg.length() ) {
                matrix.drawChar(x, y, decodedMsg[letter], HIGH, LOW, 1);
            }

            letter--;
            x -= width;
        }

        matrix.write(); // Send bitmap to display

        delay(wait);
    }
}
```

- Otros enlaces: [video](#)
- [github](#)

From:

<https://wiki.unloquer.org/> -

Permanent link:

<https://wiki.unloquer.org/personas/johnny/proyectos/matrices-led>

Last update: **2021/10/16 18:54**

