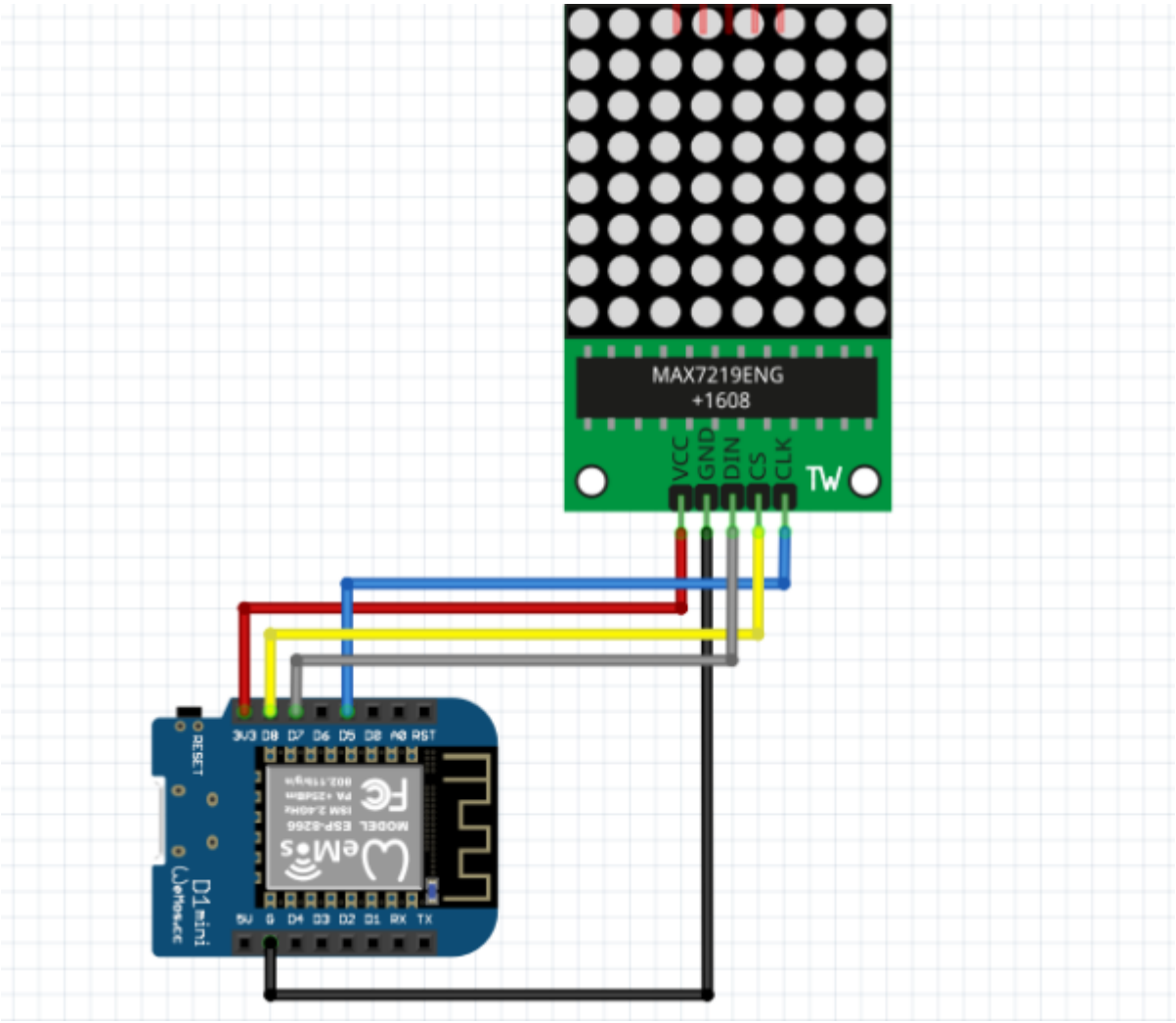
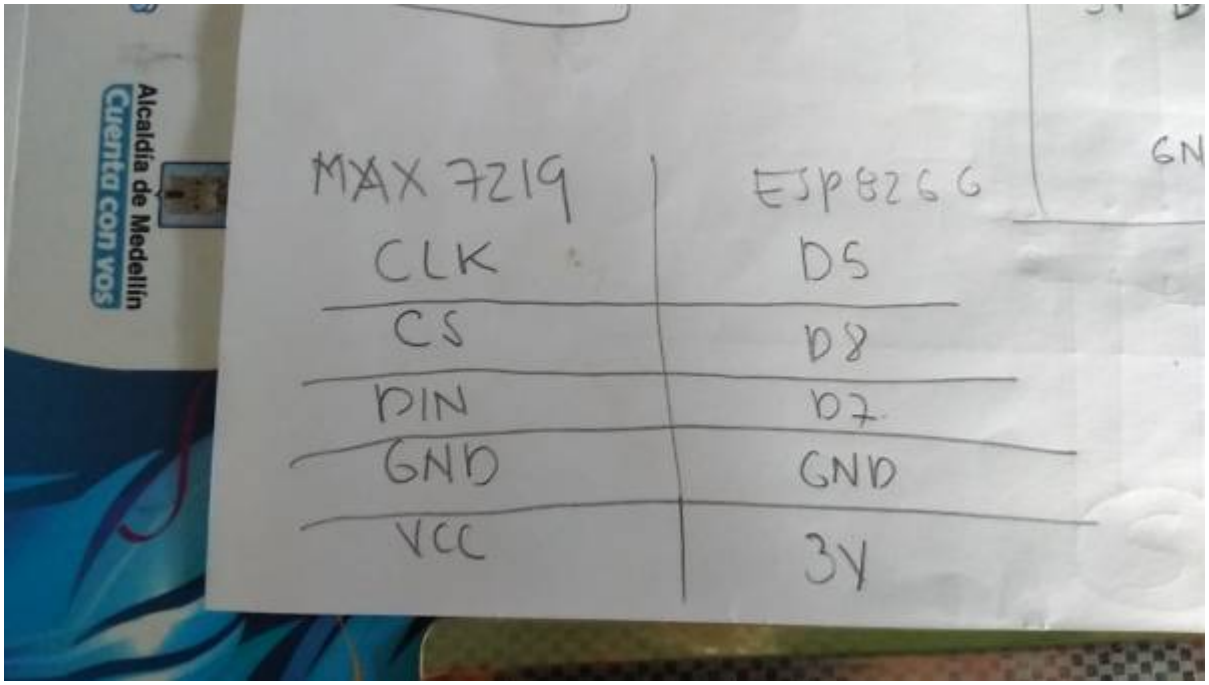
Modulo matrix max 7219

CONEXIÓN





Code

```
#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 MESG_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>" \

"<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;

#if 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, MESG_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
{
#if 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()
{
#if 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();
}
```