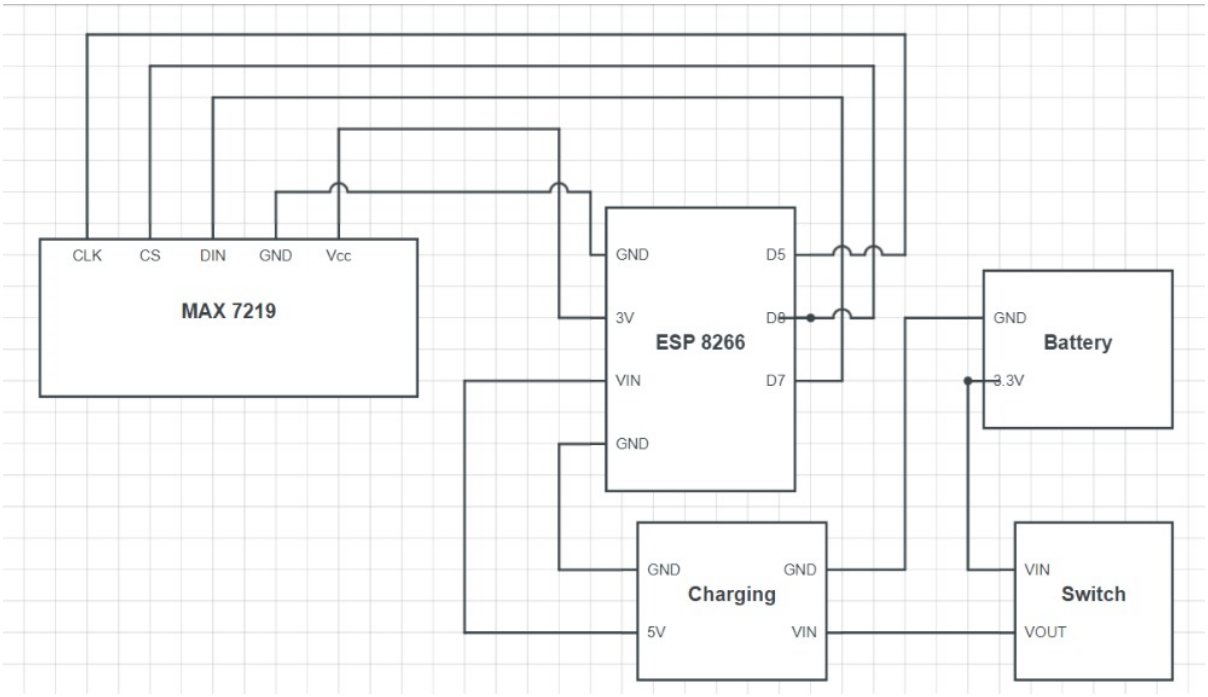
Circuit Diagram



Circuit Diagram

ESP 8266 Code

```
/*
  Sketch generated by the Arduino IoT Cloud Thing "Untitled"
  https://create.arduino.cc/cloud/things/409cac7c-e943-4e62-b010-1851e4dbb069

  Arduino IoT Cloud Variables description

  The following variables are automatically generated and updated when changes are made to th

  String data;
  String ip_address;
  int brightness;
  int speed;
  bool color;
  bool direction;

  Variables which are marked as READ/WRITE in the Cloud Thing will also have functions
  which are called when their values are changed from the Dashboard.
  These functions are generated with the Thing and added at the end of this sketch.
*/

#include "thingProperties.h"
#include <MD_Parola.h>
#include <MD_MAX72xx.h>
#include <SPI.h>

// Turn on debug statements to the serial output
```

```
#define  DEBUG  0

#if  DEBUG
#define PRINT(s, x) { Serial.print(F(s)); Serial.print(x); }
#define PRINTS(x) Serial.print(F(x))
#define PRINTX(x) Serial.println(x, HEX)
#else
#define PRINT(s, x)
#define PRINTS(x)
#define PRINTX(x)
#endif

#define HARDWARE_TYPE MD_MAX72XX::FC16_HW
#define MAX_DEVICES 8
#define CS_PIN     15 // or SS

// HARDWARE SPI
MD_Parola P = MD_Parola(HARDWARE_TYPE, CS_PIN, MAX_DEVICES);

// WiFi login parameters - network name and password
const char* ssid = "Vineet";
const char* password = "18399770";

// WiFi Server object and param eters
WiFiServer server(80);

// Scrolling parameters
uint8_t frameDelay = 25;  // default frame delay value
textEffect_t  scrollEffect = PA_SCROLL_LEFT;

// Global message buffers shared by Wifi and Scrolling functions
#define BUF_SIZE  512
char curMessage[BUF_SIZE];
char newMessage[BUF_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>" \
"<title>MajicDesigns Test Page</title>" \

"<script>" \
"strLine = \"\";" \

"function SendData()" \
"{" \
"  nocache = \"/&nocache=\" + Math.random() * 1000000;" \
"  var request = new XMLHttpRequest();" \
"  strLine = \"&MSG=\" + document.getElementById(\"data_form\").Message.value;" \
"  strLine = strLine + \"/&SD=\" + document.getElementById(\"data_form\").ScrollType.value;"
"  strLine = strLine + \"/&I=\" + document.getElementById(\"data_form\").Invert.value;" \
"  strLine = strLine + \"/&SP=\" + document.getElementById(\"data_form\").Speed.value;" \
"  request.open(\"GET\", strLine + nocache, false);" \
"  request.send(null);" \
"}" \
```

```
"</script>" \
"</head>" \

"<body>" \
"<p><b>Smart Notice Board</b></p>" \

"<form id=\"data_form\" name=\"frmText\">" \
"<label>Message:<br><input type=\"text\" name=\"Message\" maxlength=\"255\"></label>" \
"<br><br>" \
"<input type = \"radio\" name = \"Invert\" value = \"0\" checked> Normal" \
"<input type = \"radio\" name = \"Invert\" value = \"1\"> Inverse" \
"<br>" \
"<input type = \"radio\" name = \"ScrollType\" value = \"L\" checked> Left Scroll" \
"<input type = \"radio\" name = \"ScrollType\" value = \"R\"> Right Scroll" \
"<br><br>" \
"<label>Speed:<br>Fast<input type=\"range\" name=\"Speed\" min=\"10\" max=\"200\">Slow"\
"<br>" \
"</form>" \
"<br>" \
"<input type=\"submit\" value=\"Send Data\" onclick=\"SendData()\">" \
"</body>" \
"</html>";

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

void getData(char *szMesg, uint16_t len)
// Message may contain data for:
// New text (/&MSG=)
// Scroll direction (/&SD=)
// Invert (/&I=)
// Speed (/&SP=)
{
  char *pStart, *pEnd;       // pointer to start and end of text

  // check text message
  pStart = strstr(szMesg, "/&MSG=");
  if (pStart != NULL)
  {
    char *psz = newMessage;
```

```
  pStart += 6;  // skip to start of data
  pEnd = strstr(pStart, "/&");

  if (pEnd != NULL)
  {
    while (pStart != pEnd)
    {
      if ((*pStart == '%') && isxdigit(*(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
    newMessageAvailable = (strlen(newMessage) != 0);
    PRINT("\nNew Msg: ", newMessage);
  }
}

// check scroll direction
pStart = strstr(szMesg, "/&SD=");
if (pStart != NULL)
{
  pStart += 5;  // skip to start of data

  PRINT("\nScroll direction: ", *pStart);
  scrollEffect = (*pStart == 'R' ? PA_SCROLL_RIGHT : PA_SCROLL_LEFT);
  P.setTextEffect(scrollEffect, scrollEffect);
  P.displayReset();
}

// check invert
pStart = strstr(szMesg, "/&I=");
if (pStart != NULL)
{
  pStart += 4;  // skip to start of data

  PRINT("\nInvert mode: ", *pStart);
  P.setInvert(*pStart == '1');
}

// check speed
pStart = strstr(szMesg, "/&SP=");
if (pStart != NULL)
{
  pStart += 5;  // skip to start of data

  int16_t speed = atoi(pStart);
  PRINT("\nSpeed: ", P.getSpeed());
  P.setSpeed(speed);
```

```
    frameDelay = speed;
  }
}



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

  switch (state)
  {
  case S_IDLE:   // initialise
    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.
    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
    getData(szBuf, BUF_SIZE);
    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 setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is fou
  delay(1500);


  PRINTS("\n[MD_Parola WiFi Message Display]\nType a message for the scrolling display from y

  P.begin();
  P.setIntensity(15);
  P.displayClear();
  P.displaySuspend(false);

  P.displayScroll(curMessage, PA_LEFT, scrollEffect, frameDelay);

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

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

  // Start the server
```

```
  server.begin();
  PRINTS("\nServer started");

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



  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);

  /*
     The following function allows you to obtain more information
     related to the state of network and IoT Cloud connection and errors
     the higher number the more granular information you'll get.
     The default is 0 (only errors).
     Maximum is 4
 */
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();
}

void loop() {
  ArduinoCloud.update();
  // Your code here

  handleWiFi();

  if (P.displayAnimate())
  {
    if (newMessageAvailable)
    {
      strcpy(curMessage, data.c_str());
      newMessageAvailable = false;
    }
    P.displayReset();
  }
}

void onDataChange()
{
    strcpy(curMessage, data.c_str());
    P.displayReset();
}

void onSpeedChange()
{
    P.setSpeed((15-speed) * 5);
}

void onDirectionChange()
{
    scrollEffect = (direction ? PA_SCROLL_RIGHT : PA_SCROLL_LEFT);
```

```
    P.setTextEffect(scrollEffect, scrollEffect);
}

void onBrightnessChange()
{
    P.setIntensity(brightness);
}

void onColorChange()
{
    P.setInvert(color);
}
```

thingProperties.h Code

```
// Code generated by Arduino IoT Cloud, DO NOT EDIT.

#include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>

const char DEVICE_LOGIN_NAME[] = "6ea2420a-1404-48ca-a1fa-f29dc4af2bd9";

const char SSID[] = SECRET_SSID;     // Network SSID (name)
const char PASS[] = SECRET_OPTIONAL_PASS;     // Network password
const char DEVICE_KEY[] = SECRET_DEVICE_KEY;     // Secret device password

void onDataChange();
void onBrightnessChange();
void onSpeedChange();
void onColorChange();
void onDirectionChange();

String data;
String ip_address;
int brightness;
int speed;
bool color;
bool direction;

void initProperties(){

  ArduinoCloud.setBoardId(DEVICE_LOGIN_NAME);
  ArduinoCloud.setSecretDeviceKey(DEVICE_KEY);
  ArduinoCloud.addProperty(data, READWRITE, ON_CHANGE, onDataChange);
  ArduinoCloud.addProperty(ip_address, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(brightness, READWRITE, ON_CHANGE, onBrightnessChange);
  ArduinoCloud.addProperty(speed, READWRITE, ON_CHANGE, onSpeedChange);
  ArduinoCloud.addProperty(color, READWRITE, ON_CHANGE, onColorChange);
  ArduinoCloud.addProperty(direction, READWRITE, ON_CHANGE, onDirectionChange);

}

WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);
```