Design with Microprocessors

Lecture 12

The ESP32 Microcontroller - WiFi and files

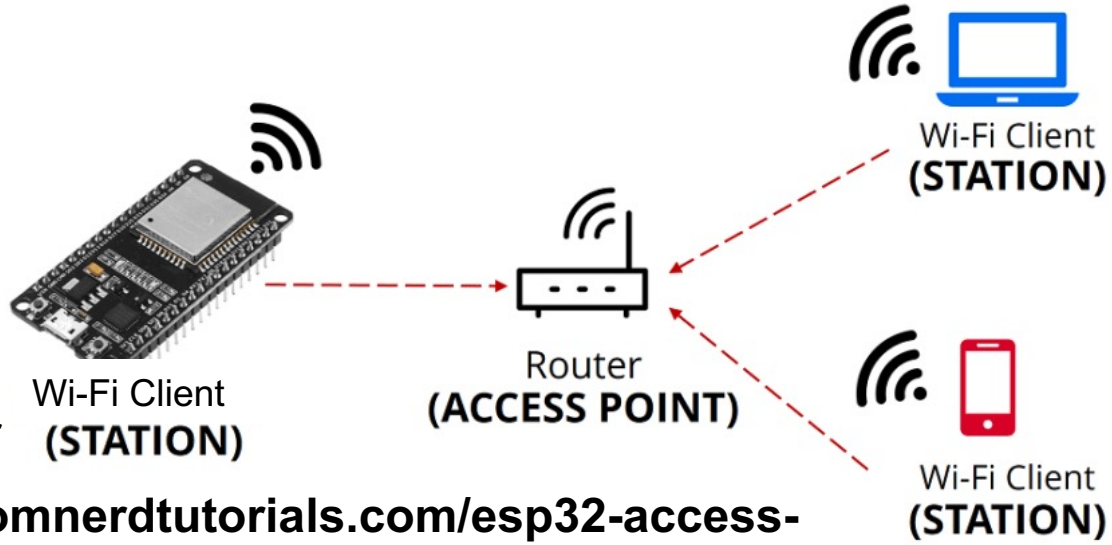
Year 3 CS
Academic year 2023/2024
1st Semester

Lecturer: Radu Dănescu

ESP32 WiFi modes

Access Point (AP)
- Creates its own WiFi network
- Defines the SSID - Service Set Identifier (name of the network)
- Defines the password (optional)
- Usually acts as a server

Station (Client)
- Connects to an existing WiFi network
- Must know the SSID and password
- Can act as a client or as a server



Source for images: https://randomnerdtutorials.com/esp32-access-point-ap-web-server/

ESP32 WiFi modes

Setting up an Access Point (AP)

WiFi.softAP(ssid, password);

WiFi.softAP(const char* ssid, const char* password, int channel, int ssid_hidden, int max_connection)

- SSID: network name, maximum of 63 characters;
- password: minimum of 8 characters; NULL means no password is required
- channel: Wi-Fi channel number (1-13)
- ssid_hidden: (0 = broadcast SSID, 1 = hide SSID)
- max_connection: maximum simultaneous connected clients (1-4)

Get the IP address of the Access Point:

```
IPAddress IP = WiFi.softAPIP();
Serial.print("AP IP address: ");
Serial.println(IP);
```

ESP32 WiFi modes

Connecting to an Access Point - automatic configuration

```
WiFi.begin(ssid, password);
Serial.println("Connecting");
while(WiFi.status() != WL_CONNECTED) { delay(500); Serial.print("."); }

Serial.println("");
Serial.print("Connected to WiFi network with IP Address: ");
```

Connecting to an Access Point - manual configuration

```
IPAddress local_IP(192, 168, 1, 184); // Set your Static IP address
IPAddress gateway(192, 168, 1, 1); // Set your Gateway IP address
IPAddress subnet(255, 255, 0, 0); // Subnet mask
IPAddress primaryDNS(8, 8, 8, 8); //DNS, optional
IPAddress secondaryDNS(8, 8, 4, 4); //DNS 2, optional

if (!WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS)) {
Serial.println("STA Failed to configure"); }

WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) { delay(500); Serial.print("."); }
```

TCP Servers and clients

TCP Server

```
WiFiServer server (port) // creates a TCP server on specified port

server.begin();   // Starts listening for incoming connections

WiFiClient client = server.available();   // returns connected client or NULL if no
client wants to connect
```

TCP Client

```
WiFiClient client = server.available();   // on the server side


WiFiClient client;                        // on the client side
client.connect (servername, port);
Or
client.connect (IP, port);

client.stop()                                 // disconnects from server
```

TCP Data Transfer

Data transfer is handled by the WiFiClient class, on the client side and on the server side

client.available() // Returns the number of bytes available for reading.

client.read() // Read the next available byte received after the last call to read()

client.write (byte b) // writes a byte

client.write (byte *buf, size) // writes a byte buffer

client.print(data) // prints data in a human-readable form

client.print(data, BASE) // optional, you can specify the base (DEC, HEX, OCT)

client.println (data) // Print data, followed by a carriage return and newline

client.flush() // Discard any bytes that have been written to the client but not yet read.

Simple WiFi AP and Web Server

```
#include <WiFi.h>
#include <WiFiAP.h>
#include <WiFiClient.h>

// MESSAGE STRINGS
const String SETUP_INIT = "SETUP: Initializing ESP32 dev board";
const String SETUP_ERROR = "!!ERROR!! SETUP: Unable to start SoftAP mode";
const String SETUP_SERVER_START = "SETUP: HTTP server started --> IP addr: ";
const String SETUP_SERVER_PORT = " on port: ";
const String INFO_NEW_CLIENT = "New client connected";
const String INFO_DISCONNECT_CLIENT = "Client disconnected";

// HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
// and a content-type so the client knows what's coming, then a blank line:
const String HTTP_HEADER = "HTTP/1.1 200 OK\r\nContent-type:text/html\r\n\r\n";
const String HTML_WELCOME = "<h1>Welcome to your ESP32 Web Server!</h1>";

// BASIC WIFI CONFIGURATION CONSTANTS
// The SSID (Service Set IDentifier), in other words, the network's name
const char *SSID = "<your_unique_SSID>";
// Password for the network
// By default the ESP32 uses WPA / WPA2-Personal security, therefore the
// the password MUST be between 8 and 63 ASCII characters
const char *PASS = "<your_password_here>";
// The default port (both TCP & UDP) of a WWW HTTP server number according to
// RFC1340 is 80
const int HTTP_PORT_NO = 80;
```

Simple WiFi AP and Web Server

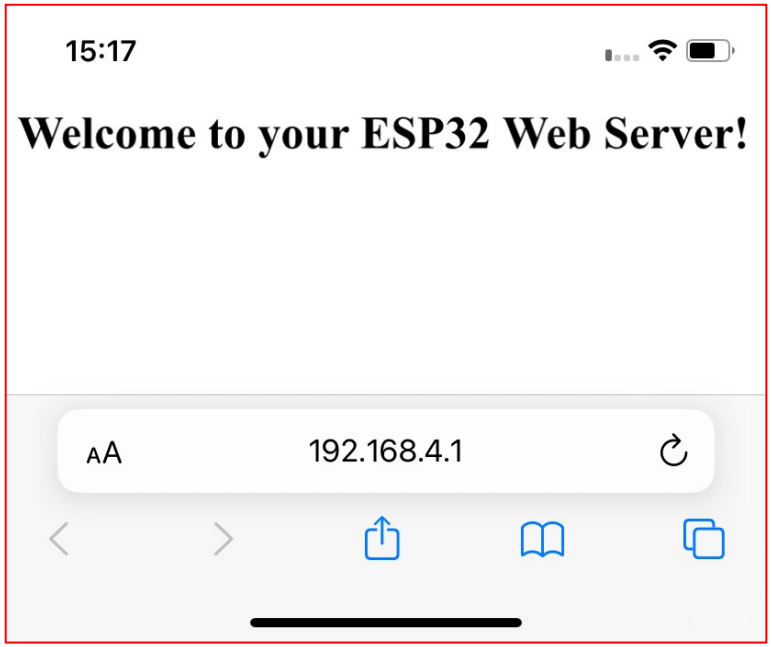
```
// ADDITIONAL GLOBALS
// Initialize the HTTP server on the ESP32 board
WiFiServer HttpServer(HTTP_PORT_NO);

void setup() {
 Serial.begin(9600);

 if (!WiFi.softAP(SSID, PASS)) {
   Serial.println(SETUP_ERROR);
   // Lock system in infinite loop in order to prevent further execution
   while (1)
     ;
 }

 // Get AP's IP address for info message
 const IPAddress accessPointIP = WiFi.softAPIP();
 const String webServerInfoMessage = SETUP_SERVER_START +
accessPointIP.toString()  + SETUP_SERVER_PORT + HTTP_PORT_NO;

 // Start the HTTP server
 HttpServer.begin();
 Serial.println(webServerInfoMessage);
}
```

Simple WiFi AP and Web Server

```
void loop() {
 WiFiClient client = HttpServer.available();  // listen for incoming clients
 if (client) {                               // if you get a client,
   Serial.println(INFO_NEW_CLIENT);          // print a message out the serial port
   String currentLine = ""; // make a String to hold request from the client
   while (client.connected()) {              // loop while the client's connected
     if (client.available()) {             // if there's bytes to read from the client,
       const char c = client.read();       // read a byte, then
       Serial.write(c);                    // print it out the serial monitor
       if (c == '\n') {  // if the byte is a newline character
         // if the current line is blank, that's the end of the client request
         if (currentLine.length() == 0) {
           // Send welcome page to the client
           printWelcomePage(client);
           break;
         } else currentLine = "";
       } else if (c != '\r') {  // if not carriage return
         currentLine += c;      // add it to the end of the currentLine
       }
     }
   }
   // close the connection:
   client.stop();
   Serial.println(INFO_DISCONNECT_CLIENT);
   Serial.println();
 }
}
```

Simple WiFi AP and Web Server

```
void printWelcomePage(WiFiClient client) {
 // Always start the response to the client with the proper headers
 client.println(HTTP_HEADER);


 // Send the relevant HTML
 client.print(HTML_WELCOME);


 // The HTTP response ends with another blank line
 client.println();
}
```



Web page requests - sample index page

http://192.168.4.1

Welcome to the index page!

This will show the ESP32 filesystem capabilities, and the web server capabilities.

The File System

The Web Server

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=ut
  <meta http-equiv="Content-Style-Type" content="text/css">
  <title></title>
  <meta name="Generator" content="Cocoa HTML Writer">
  <meta name="CocoaVersion" content="2022.6">
  <style type="text/css">
    p.p1 {margin: 0.0px 0.0px 0.0px 0.0px; font: 12.0px Helvetic
    p.p2 {margin: 0.0px 0.0px 0.0px 0.0px; font: 12.0px Helvetic
  </style>
</head>
<body>
<p class="p1"><b>Welcome to the index page!</b></p>
<p class="p2"><br></p>
<p class="p1">This will show the ESP32 filesystem capabilities,
<p class="p2"><br></p>
<p class="p1"><a href="filesystem.html">The File System</a></p>
<p class="p2"><br></p>
<p class="p1"><a href="webserver.html">The Web Server</a></p>
<p class="p2"><br></p>
<p class="p2"><br></p>
<p class="p2"><br></p>
</body>
</html>

```

Web page requests - no file name

GET / HTTP/1.1
Host: 192.168.4.1
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Mobile Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: ro-RO,ro;q=0.9,en-US;q=0.8,en;q=0.7,pt;q=0.6,de;q=0.5

Web page requests - specific file name

GET /filesystem.html HTTP/1.1
Host: 192.168.4.1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Mobile Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://192.168.4.1/
Accept-Encoding: gzip, deflate
Accept-Language: ro-RO,ro;q=0.9,en-US;q=0.8,en;q=0.7,pt;q=0.6,de;q=0.5

The SPIFFS file system

SPIFFS - SPI Flash File Storage

- A file system for the ESP32 flash

A part of the ESP32's flash memory can be used for storing files

- You can find the address and the size of the file area of the flash in the ESP32 folder of the Arduino environment

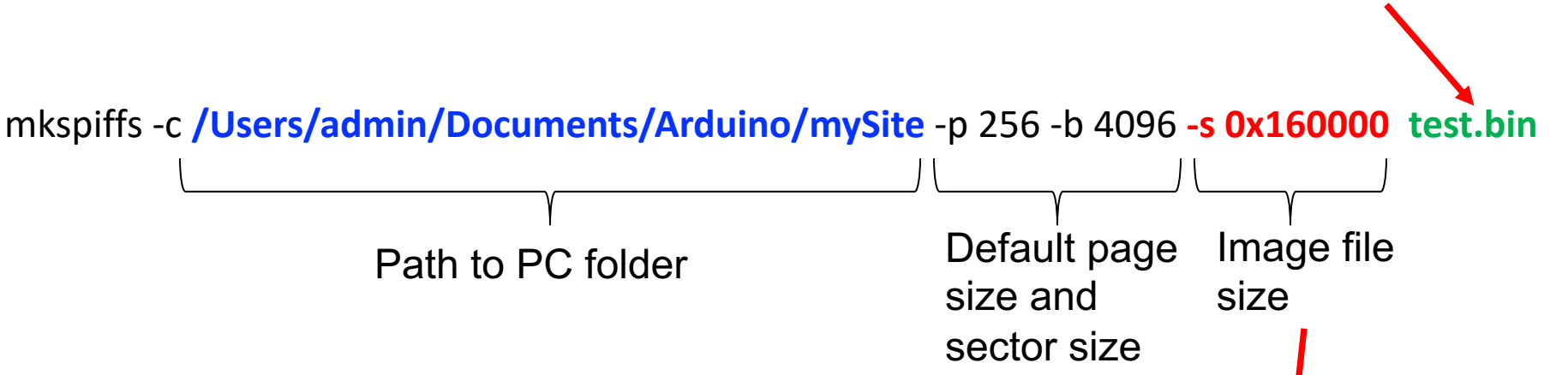
/System/Volumes/Data/Users/admin/Library/Arduino15/packages/esp32/hardware/esp32/2.0.11/tools/partitions/default.csv

```
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x5000,
otadata,  data, ota,     0xe000,  0x2000,
app0,     app,  ota_0,   0x10000, 0x140000,
app1,     app,  ota_1,   0x150000,0x140000,
spiffs,   data, spiffs,  0x290000,0x160000,
coredump, data, coredump,0x3F0000,0x10000,
```

The SPIFFS file system

You can create the files from your ESP32 program, or **you can move files from the PC!**

To move the files from a folder (your "website"), first you have to create an **image file**:




```
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x5000,
otadata,  data, ota,     0xe000,  0x2000,
app0,     app,  ota_0,   0x10000, 0x140000,
app1,     app,  ota_1,   0x150000,0x140000,
spiffs,   data, spiffs,  0x290000,0x160000,
coredump, data, coredump,0x3F0000,0x10000,
```

The SPIFFS file system

After the **image file** is created, write it to ESP32 using **esptool**:

esptool --chip esp32 --port **/dev/cu.usbserial-0001** --baud 921600 --before default_reset --after hard_reset write_flash --flash_freq 80m --flash_size 4MB **0x290000** **test.bin**

The serial port

Address to write to

```
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x5000,
otadata,  data, ota,     0xe000,  0x2000,
app0,     app,  ota_0,   0x10000, 0x140000,
app1,     app,  ota_1,   0x150000,0x140000,
spiffs,   data, spiffs,  0x290000,0x160000,
coredump, data, coredump,0x3F0000,0x10000,
```

Working with the SPIFFS file system

```
// list the files in the directory
void listDir(fs::FS &fs, const char * dirname, uint8_t levels){

    File root = fs.open(dirname);
    if(!root){
        Serial.println("- failed to open directory");
        return;
    }

  if(!root.isDirectory()){
        Serial.println(" - not a directory");
        return;
    }

   File file = root.openNextFile();
    while(file){
        if(file.isDirectory()){
            Serial.print("  DIR : "); Serial.println(file.name());
            if(levels){
                listDir(fs, file.name(), levels -1);
            }
        } else {
            Serial.print("  FILE: "); Serial.print(file.name());
Serial.print("\tSIZE: "); Serial.println(file.size());
        }
        file = root.openNextFile();
    }
}
```

https://www.tutorialspoint.com/esp32_for_iot/esp32_for_iot_spiffs_storage.htm

Working with the SPIFFS file system

```
// read the content of a file into a string
String readFileString(fs::FS &fs, const char * path){

  String s;
  File file = fs.open(path);

   if(!file || file.isDirectory()){
        return "";
    }

    while(file.available()){
        s = s + (char)file.read();
    }
    file.close();

    return s;
}
```

Working with the SPIFFS file system

```
// write a String to a file
void writeFile(fs::FS &fs, const char * path, String message){

    Serial.printf("Writing file: %s\r\n", path);
    File file = fs.open(path, FILE_WRITE);
    if(!file){
        Serial.println("- failed to open file for writing");
        return;
    }

    if(file.print(message)){
        Serial.println("- file written");
    } else {
        Serial.println("- write failed");
    }

    file.close();

}
```

Web server using SPIFFS

```
#include "FS.h"
#include "SPIFFS.h"


#include <WiFi.h>
#include <WiFiAP.h>
#include <WiFiClient.h>


// HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
// and a content-type so the client knows what's coming, then a blank line:
const String HTTP_HEADER = "HTTP/1.1 200 OK\r\nContent-type:text/html\r\n\r\n";


// BASIC WIFI CONFIGURATION CONSTANTS
// The SSID (Service Set IDentifier), in other words, the network's name
const char *SSID = "ESP32 web server";

// The Server object
WiFiServer HttpServer(80);
```

Web server using SPIFFS

```
// File system setting - do not format if fail
#define FORMAT_SPIFFS_IF_FAILED false
void setup(){
    Serial.begin(115200);

    // Initialize SPIFFS file system
    if(!SPIFFS.begin(FORMAT_SPIFFS_IF_FAILED)){
        Serial.println("SPIFFS Mount Failed");
                while (1) ; // Freeze here
    } else Serial.println ("SPIFFS Mount OK");

   // set up the access point
   if (!WiFi.softAP(SSID)) {
     Serial.println("Failed to set up access point");
        while (1) ;      // Freeze here
   }

   // Get AP's IP address for info message
   const IPAddress accessPointIP = WiFi.softAPIP();
   // Start the HTTP server
   HttpServer.begin();
   const String webServerInfoMessage = "Access point set up " +
accessPointIP.toString() + " and web server on port 80";

   Serial.println(webServerInfoMessage);
}
```

Web server using SPIFFS

```
void printPage(WiFiClient client, String page) {
 // Always start the response to the client with the proper headers
 client.println(HTTP_HEADER);

  String fname = "/index.html"; // default file name if none specified

  if (page.indexOf("htm")>=0)
      fname = "/"+page;            // real file name if specified

 // Send the relevant HTML

 String HTML_WELCOME = readFileString (SPIFFS, fname.c_str());
 client.print(HTML_WELCOME);


 // The HTTP response ends with another blank line
 client.println();
}
```

Web server using SPIFFS

```
String getRequestFileName (String& request)
{

          String fname = "";
            // parse request
          int g = request.indexOf ("GET /");
          if (g>=0) {
             String afterGet = request.substring(g+5);

             int h = afterGet.indexOf (" HTTP/");

             if (h>=0)
               fname = afterGet.substring(0, h);
          }

      return fname;
}
```

GET / HTTP/1.1

GET /filesystem.html HTTP/1.1

Web server using SPIFFS

```
void loop(){
 WiFiClient client = HttpServer.available();  // listen for incoming clients
 if (client) {                    // if you get a client,
 String request;                 // this contains the request from the client
 Serial.println("New client connected"); // print a message out the serial port
 String currentLine = "";    // make a String to hold incoming data from the client
  while (client.connected()) {          // loop while the client's connected
     if (client.available()) {           // if there's bytes to read from the client,
       const char c = client.read();   // read a byte, then
       request = request + c;           // add it to the request string
       Serial.write(c);                 // print it out the serial monitor
       if (c == '\n') {  // if the byte is a newline character
         // if the current line is blank, end of request, send response.
          if (currentLine.length() == 0) {
            String fname = getRequestFileName(request);
            Serial.println ("The client wants the file: "+fname);
           printPage(client, fname); // Send welcome page to the client
            break;
         } else currentLine = "";
       } else if (c != '\r') {
           currentLine += c;  // add read character to current line
       }
     }
   }
   client.stop(); // close the connection:
   Serial.println("Client disconnected."); Serial.println();
 }
}
```

Adding CSS capabilities

```html
<!DOCTYPE html>
<html>
<head>
  <title>ESP32 Web Server</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" href="data:,">
  <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
  <h1>ESP32 Web Server</h1>
  <p>GPIO state: <strong> %STATE%</strong></p>
  <p><a href="/on"><button class="button">ON</button></a></p>
  <p><a href="/off"><button class="button button2">OFF</button></a></p>
</body>
</html>
```

index.html

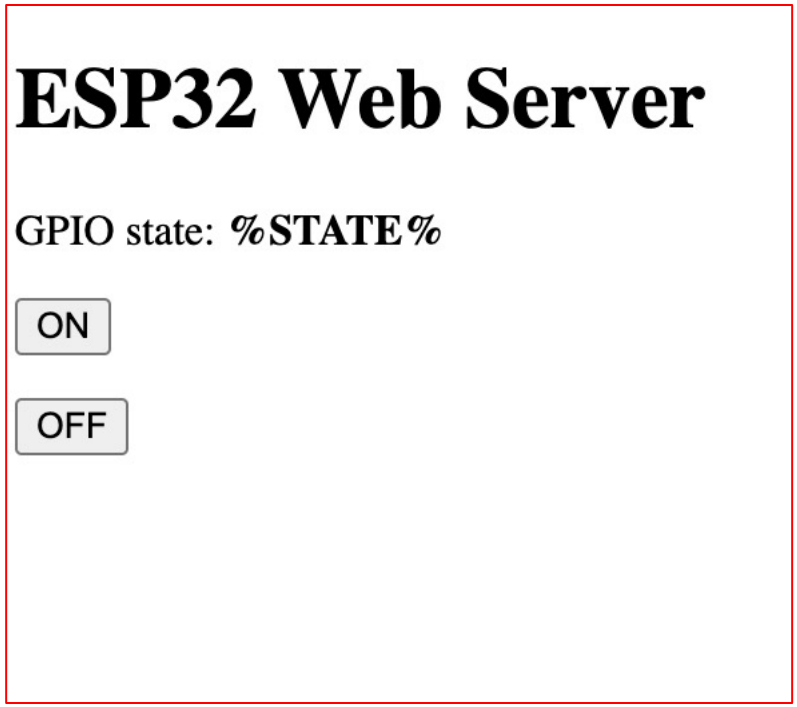
```css
html {
  font-family: Helvetica;
  display: inline-block;
  margin: 0px auto;
  text-align: center;
}
h1{
  color: #0F3376;
  padding: 2vh;
}
p{
  font-size: 1.5rem;
}
.button {
  display: inline-block;
  background-color: #008CBA;
  border: none;
  border-radius: 4px;
  color: white;
  padding: 16px 40px;
  text-decoration: none;
  font-size: 30px;
  margin: 2px;
  cursor: pointer;
}
.button2 {
  background-color: #f44336;
}
```

style.css

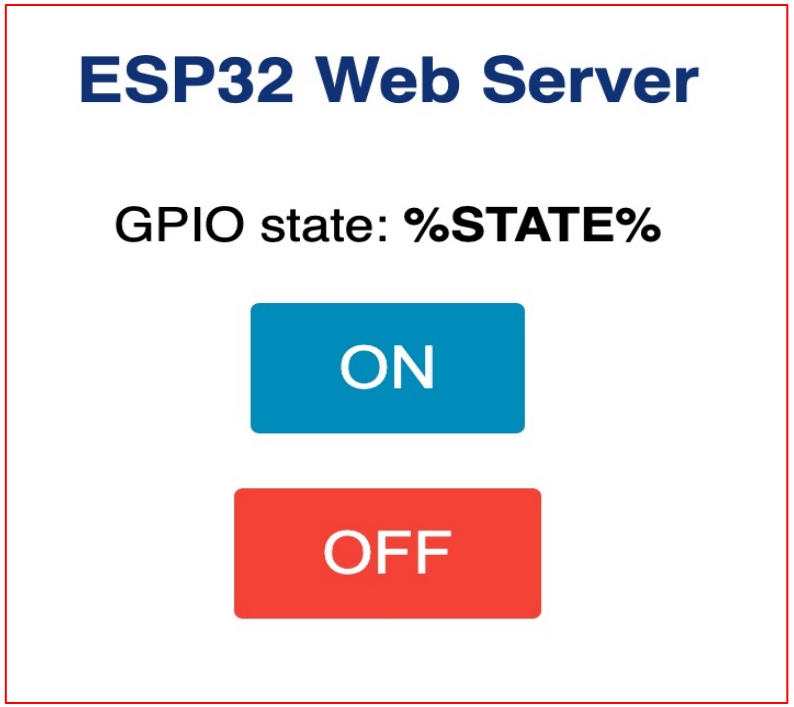
Adding CSS capabilities

```
const String CSS_HEADER = "HTTP/1.1 200 OK\r\nContent-type:text/css\r\n\r\n";

void printPage(WiFiClient client, String page) {
 // Always start the response to the client with the proper headers

 if (page.indexOf("css")>0)
 client.println(CSS_HEADER);
 else
 client.println(HTTP_HEADER);

  String fname = "/index.html";

  if (page.indexOf("htm")>=0 || page.indexOf("css")>=0)
      fname = "/"+page;

 // Send the relevant HTML

 String HTML_WELCOME = readFileString (SPIFFS, fname.c_str());
 client.print(HTML_WELCOME);


 // The HTTP response ends with another blank line
 client.println();
}
```

Adding CSS capabilities



Without CSS



With CSS

Using a Web Server Library

ESP Async Web Server

- Listens for connections
- Wraps the new clients into Request
- Keeps track of clients and cleans memory
- The server can handle new connections while serving requests in the background
- High speed
- Easy to use API, HTTP Basic and Digest MD5 Authentication
- Easily extendible to handle any type of content

https://github.com/me-no-dev/ESPAsyncWebServer

https://github.com/me-no-dev/AsyncTCP

Using a Web Server Library

```
// Import required libraries
#include "WiFi.h"
#include "ESPAsyncWebServer.h"
#include "SPIFFS.h"

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";

// Set LED GPIO
const int ledPin = 2;
// Stores LED state
String ledState;

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);
```

https://randomnerdtutorials.com/esp32-web-server-spiffs-spi-flash-file-system/

Using a Web Server Library

```
void setup(){
  Serial.begin(115200); // Serial port for debugging purposes
  pinMode(ledPin, OUTPUT);
  // Initialize SPIFFS
  if(!SPIFFS.begin(true)){
    Serial.println("An Error has occurred while mounting SPIFFS"); return; }
 // set up the access point
   if (!WiFi.softAP(ssid)) {
     Serial.println("Failed to set up access point");
     while (1);}

// Route for root / web page
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/index.html", String(), false, processor); });
  // Route to load style.css file
  server.on("/style.css", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/style.css", "text/css"); });
  // Route to set GPIO to HIGH
  server.on("/on", HTTP_GET, [](AsyncWebServerRequest *request){
    digitalWrite(ledPin, HIGH);
    request->send(SPIFFS, "/index.html", String(), false, processor); });
  // Route to set GPIO to LOW
  server.on("/off", HTTP_GET, [](AsyncWebServerRequest *request){
    digitalWrite(ledPin, LOW);
    request->send(SPIFFS, "/index.html", String(), false, processor); });
  // Start server
  server.begin();
}
```

Using a Web Server Library

```
// The processor() function is what will attribute a value to the placeholder
// we've created on the HTML file. It accepts as argument the placeholder and
// should return a String that will replace the placeholder.

String processor(const String& var){
  Serial.println(var);
  if(var == "STATE"){
    if(digitalRead(ledPin)){
      ledState = "ON";
    }
    else{
      ledState = "OFF";
    }
    Serial.print(ledState);
    return ledState;
  }
  return String();
}


void loop ()
{
  // nothing in the loop
}
```

Summary

- **ESP32 can act as WiFi access point as well as WiFi client**
- **A TCP server can be set up at a specified port number**
- **A TCP client connects to an IP address and a port number**
- **The communication is handled by the WiFiClient class**
- **The SPIFFS file system can store files, allowing us to create more complex servers or applications**
- **The ESP Async Web Server library can be used to set up complex web pages and service multiple simultaneous requests**