Laborator 8

WiFi

Obiective

- Modurile de funcționare ale modulului de WiFi pentru ESP8266
- Configurarea unei rețele WiFi și a unui server

Cuprins

ESP8266 WiFi

La momentul actual, datorită evoluției microcontrollerelor, au evoluat și perifericele care le însoțesc. Astfel, a crescut capacitatea de stocare a acestora, viteza de operare (frecvența) și dimensiunea magistralelor de date. Datorită acestor peformanțe crescute, putem, deci, utiliza moduri avansate de comunicare pentru a lărgi aria aplicațiilor.

Un astfel de modul îl reprezintă modulul de WiFi, integrat în placa noastră de dezvoltare. Acesta permite comunicarea wireless cu alte dispozitive și gestionarea clienților conectați în cazul în care configurăm un server local.

În cazul plăcilor de dezvoltare cu ESP8266 putem utiliza unul din cele 3 moduri de funcționare ale modulului de WiFI: STA, AP și STA-AP. În modul de funcționare STA (Station), placa de dezvoltare se conectează la o rețea WiFi deja existentă. În modul de funcționare AP (Access Point), placa de dezvoltare își creeaza propria rețea WiFI independentă fără a fi necesară conectarea la altă rețea deja existentă. Modul de funcționare STA-AP reprezintă o combinație dintre cele 2 moduri descrise mai sus în care se creează o rețea wireless ca punct de acces la o altă rețea deja existentă la care este conectată placa de dezvoltare.

Configurarea propriei noastre rețele de WiFi

Pentru a putea realiza o rețea wireless trebuie efectuați câțiva pași inițiali care ne vor configura rețeaua așa cum ne dorim.

Pentru început o să inițializăm adaptorul de TCP-IP inclus deja în SDK care ne va gestiona IP-urile din rețea, inclusiv IP-ul modulului intern de WiFi.

Urmatorul pas este să creăm o structură de date în care o să stocăm o configurație de bază cu ajutorul căreia o sa inițializam modulul wireless, urmând ca utilizând funcțiile din cadrul SDK să alocăm și memorie pentru acest modul.

În continuare, o să creăm o structură de date în care vom stoca configurarea rețelei pentru modul de funcționare AP. Aici o să setăm numele rețelei (SSID) așa cum va fi vizibil pentru alte dispozitive, parola, numărul maxim de conexiuni disponibile, modul de criptare al rețelei și

vizibilitatea ei. În cazul în care setăm modul de criptare WPA2-PSK, trebuie să fim atenți să setăm o parolă de minim 8 caractere, altfel rețeaua nu va funcționa și sistemul de operare va intra într-un boot-loop fără a afișa vreo eroare.

În ultimul pas setăm modul de funcționare, configurarea descrisă mai sus care trebuie să corespundă cu modul ales (AP în cazul nostru) și pornim modulul de WiFi.

În acest moment ar trebui să avem o rețea funcțională care va fi vizibilă pe celelalte dispozitive și la care ne putem conecta. Pentru a ne putea conecta la aceasta, din setările rețelei de pe dispozitiv, vom selecta modul de conectare “Static” și vom introduce manual adresa “192.168.4.2”.

Crearea propriului nostru server

Un server este un program care are rolul de a furniza servicii altor aplicații (clienți) aflate pe același dispozitiv sau pe alte dispozitive în momentul în care are loc o anumită cerere.

În cazul nostru, o să utilizăm un server pentru a găzdui un site web (interfață) pe care vom afișa diverse informații primite de la placa de dezvoltare (și periferice) și pentru a efectua anumite acțiuni ca urmare a interacțiunii utilizatorului cu interfața web.

Pentru a crea un server va fi nevoie să utilizăm un serviciu de comunicare, numit HTTP (Hypertext Transfer Protocol). Comparativ cu serviciul HTTPS, care este securizat, criptând astfel cererile, HTTP transmite cererile într-un format vizibil pentru oricine este conectat la rețea. Dar, pentru a ne ușura munca în scopul acestui laborator și deoarece nu suntem conectați la internet datorită modului AP pentru a exista probleme de securitate, o să utilizăm serviciul HTTP.

În acest protocol de transfer se pot utiliza mai multe tipuri de cereri între clienți și server, dar noi o să utilizăm în special cererile de tip GET și POST. În momentul în care un client face un „request” (cerere) de tipul GET, acesta îi transmite serverului că are nevoie să primească date de la o anumită adresă. În cazul cererii de tip POST, clientul anunță serverul că vrea să modifice anumite date efectuând o transmisie către server la o anumită adresă. Toate aceste date sunt transmise utilizând HTTP sub forma unor șiruri de caractere ce au un anumit format.

Pentru aplicația noastră vom găzdui site-ul web la adresa „ip-server/index.html” și ne vom folosi de link-ul „ip-server/data.txt” pentru a recepționa anumite date de la server, unde „ip-server” reprezintă adresa ip de bază a modulului (de obicei „192.168.4.1”).

În momentul în care un client se va conecta la server și va accesa adresa „ip-server/index.html”, se va face un request GET automat către server, iar acesta va transmite înapoi pagina html care va putea fi interpretată de browserul folosit. De asemenea, utilizând funcțiile javascript din această pagină, se vor efectua cereri automat către server la anumite intervale de timp utlizând linkul de la adresa „data.txt”, datele primite fiind prelucrate și afișate tot cu ajutorul javascript.

La apăsarea butoanelor de pe interfață, se vor face anumite cereri de tip POST cu anumite texte prestabilite în codul paginii web și în funcție de aceste mesaje primite, se vor efectua anumite acțiuni la nivelul plăcii de dezvoltare.

Pentru a crea un server, vom folosi mai întai o funcționalitate din SDK prin care la compilare, prin modul de configurare al proiectului, se va face automat o conversie a fișierului „index.html” în cod binar pe care îl vom stoca sub formă de șiruri de caractere.

```
7 COMPONENT_EMBED_FILES := index.html
```

```
23 /* Import the html binary files to be used for the request */
24 extern const char index_html_start[] asm("_binary_index_html_start");
25 extern const char index_html_end[] asm("_binary_index_html_end");
```

Pentru a putea rula acest server, ne vom folosi de câteva funcționalitați specifice SDK-ului bazat pe FreeRTOS și anume, „event handlers”. Utilizând aceste rutine, vom putea rula serverul în fundal fără a fi nevoiți să verificăm manual conexiunile de la clienți și cererile acestora. Prin urmare, vom crea 3 structuri de date specifice fiecărei interacțiuni client-server.

```
33 /* Page configuration structure */
34 httpd_uri_t myPage_get =
35 { .uri = "/index.html", .method = HTTP_GET, .handler = WebPageGetHandler,
36         .user_ctx = NULL };
37
38 /* Page configuration structure */
39 httpd_uri_t myData_get =
40 { .uri = "/data.txt", .method = HTTP_GET, .handler = DataGetHandler, .user_ctx =
41 NULL };
42
43 httpd_uri_t myPage_post =
44 { .uri = "/index.html", .method = HTTP_POST, .handler = WebPagePostHandler,
45         .user_ctx = NULL };
46
```

Folosind rutina „WebPageGetHandler”, serverul va returna pagina web de la adresa „index.html” de fiecare dată când clientul face o cerere de tip GET la această adresă.

```
47 esp_err_t WebPageGetHandler(httpd_req_t *req)
48 {
49     /* Send the request response to the user */
50     ESP_ERROR_CHECK(
51             httpd_resp_send(req, index_html_start,
52                     index_html_end - index_html_start));
53 
54     return ESP_OK;
55 }
```

Rutina „DataGetHandler” va rula în momentul în care clientul va face o cerere GET la adresa „data.txt”, server-ul returnând prin intermediul HTTP un șir de caractere ce va conține anumite date selectate de programator. Acestea vor fi prelucrate la nivelul server-ului prin concatenarea mai multor șiruri de caractere, cu ajutorul caracterului „\n” ca delimitator, într-un singur sir.

```
57 esp_err_t DataGetHandler(httpd_req_t *req)
58 {
59     g_bGETRequestInProcess = true;
60 
61     /* Wait for the process to complete */
62     while (g_bGETRequestInProcess)
63     {
64         taskYIELD();
65     }
66 
67     /* Send the request response to the user */
68     ESP_ERROR_CHECK(httpd_resp_send(req, g_cGETBuffer, sizeof(g_cGETBuffer)));
69 
70     return ESP_OK;
71 }
```

Cea de-a treia rutină, „WebPagePostHandler” va rula în momentul în care clientul va face o cerere de tip POST, șirul de caractere recepționat prin intermediul HTTP urmând sa fie pe urmă prelucrat de catre server/program.

```
73  esp_err_t WebPagePostHandler(httpd_req_t *req)
74  {
75      int ret, remaining = req->content_len;
76
77      uint8_t u8Index = 0;
78
79      /* Clear the buffer */
80      for (; u8Index < 100; u8Index++)
81      {
82          g_cPOSTBuffer[u8Index] = 0;
83      }
84
85      while (remaining > 0)
86      {
87          /* Read the data for the request */
88          if ((ret = httpd_req_recv(req, g_cPOSTBuffer,
89                  MIN(remaining, sizeof(g_cPOSTBuffer)))) <= 0)
90          {
91              if (ret == HTTPD_SOCK_ERR_TIMEOUT)
92              {
93                  /* Retry receiving if timeout occurred */
94                  continue;
95              }
96              return ESP_FAIL;
97          }
98
99          /* Send back the same data */
100         ESP_ERROR_CHECK(
101                 httpd_resp_send(req, index_html_start,
102                         index_html_end - index_html_start));
103         remaining -= ret;
104
105         for (; ret < 100; ret++)
106         {
107             g_cPOSTBuffer[ret] = 0;
108         }
109     }
110
111     g_bPOSTRequestInProcess = true;
112
113     /* Wait for the process to complete */
114     while (g_bPOSTRequestInProcess)
115     {
116         taskYIELD();
117     }
118
119     return ESP_OK;
120 }
```

Pentru a putea porni severul, va fi necesară deschiderea unui port (de obicei 80) și înregistrarea link-urilor precizate mai sus „index.html”, respectiv „data.txt”.

```
122 httpd_handle_t WebPageStart(void)
123 {
124     httpd_handle_t server = NULL;
125     httpd_config_t config = HTTPD_DEFAULT_CONFIG();
126 
127     // Start the HTTPD server
128     ESP_LOGI(TAG, "Starting server on port: '%d'", config.server_port);
129 
130     if (httpd_start(&server, &config) == ESP_OK)
131     {
132         // Set URI handlers
133         ESP_LOGI(TAG, "Registering URI handlers");
134         ESP_ERROR_CHECK(httpd_register_uri_handler(server, &myPage_get));
135         ESP_ERROR_CHECK(httpd_register_uri_handler(server, &myData_get));
136         ESP_ERROR_CHECK(httpd_register_uri_handler(server, &myPage_post));
137 
138         ESP_LOGI(TAG, "URI handlers registered");
139 
140         return server;
141     }
142 
143     ESP_LOGI(TAG, "Error starting server!");
144 
145     return NULL;
146 }
```

```
148 static esp_err_t event_handler(void *ctx, system_event_t *event)
149 {
150     httpd_handle_t *server = (httpd_handle_t*) ctx;
151 
152     /* Start the web server */
153     if (*server == NULL)
154     {
155         *server = WebPageStart();
156     }
157 
158     return ESP_OK;
159 }
160 
```

Ultimul pas va fi crearea unei rutine de tipul „event_loop” prin care sistemul de operare va rula automat cererile efectuate de către clienții conectați.

```
161 void WIFI_vInit(void *arg)
162 {
163     /* TCPIP stack to handle IPs */
164     tcpip_adapter_init();
165 
166     /* Initialize the server events */
167     ESP_ERROR_CHECK(esp_event_loop_init(event_handler, arg));
168 
169     /* WIFI structure */
170     wifi_init_config_t wifi_config = WIFI_INIT_CONFIG_DEFAULT()
171     ;
172 
173     /* Default configuration */
174     ESP_ERROR_CHECK(esp_wifi_init(&wifi_config));
175     ESP_ERROR_CHECK(esp_wifi_set_storage(WIFI_STORAGE_RAM));
176 
177     /* Configure AP structure */
178     wifi_config_t wifi_ap_config =
179     { .ap =
180     { .ssid = ESP_SSID_AP, .password = ESP_PASS_AP, .ssid_len = strlen(
181     ESP_SSID_AP), .channel = 1, .authmode = WIFI_AUTH_WPA2_PSK,
182             .ssid_hidden = 0, .max_connection = 1, .beacon_interval = 100 }, };
183 
184     /* Password validation */
185     if (strlen(ESP_PASS_AP) == 0)
186     {
187         wifi_ap_config.ap.authmode = WIFI_AUTH_OPEN;
188     }
189     if (strlen(ESP_PASS_AP) < 8)
190     {
191         ESP_LOGI(TAG, "Password %s is too short", ESP_PASS_AP);
192     }
193 
194     /* Set mode to AP */
195     ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));
196     /* Apply AP settings */
197     ESP_ERROR_CHECK(esp_wifi_set_config(ESP_IF_WIFI_AP, &wifi_ap_config));
198     /* Start the WIFI protocol */
199     ESP_ERROR_CHECK(esp_wifi_start());
200 
201     ESP_LOGI(TAG, "WiFi initialize AP finished. SSID: %s password: %s",
202             ESP_SSID_AP, ESP_PASS_AP);
203 }
204 
```

Această inițializare va putea fi apelată utilizând ca parametru un element de tip httpd_handle_t ca parametru.

```
37 static httpd_handle_t server = NULL;
```

```
56     WIFI_vInit(&server);
```

În acest moment vom avea o pagină web care va fi disponibilă pe dispozitiv la adresa “192.168.4.1/index.html”.