SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINARSKI RAD

ESP-NOW PROTOKOL

Fran Fodor

Voditelj: Doc. dr. sc. Leonardo Jelenković

Zagreb, lipanj, 2025.

ESP-NOW protokol

Fran Fodor

Sažetak

U ovom seminaru opisan je ESP-NOW protokol i prikazana je primjena istog uz web server koji služi da preko web sučelja šaljemo podatke/zahtjeve na udaljeni mikroupravljač.

Ključne riječi: ESP; ESP-NOW

Sadržaj

1. Uvod

Komunikacija između uređaja je neizostavni dio današnjeg (IoT) svijeta. Jedan od standardnih pristupa bi bila komunikacija pomoću WiFi-ja, no takav način uvodi dodatnu potrošnju i latenciju u sustav. Kao rješenje za taj problem razvijen je ESP-NOW protokol koji služi za bežičnu komunikaciju između ESP32 i ESP8266 mikrokontrolera. ESP-NOW koristi već postojeći hardver za WiFi, samo što pojednostavljuje način komunikacije da bi se postigla manja potrošnja i latencija.

U ovom seminaru opisan je ESP-NOW protokol i prikazana je primjena istog uz web server koji služi da preko web sučelja šaljemo podatke/zahtjeve na udaljeni mikroupravljač.

2. ESP-NOW

ESP-NOW je P2P (engl. peer-to-peer) protokol koji omogućava jednostavnu bežičnu komunikaciju između ESP mikrokontrolera koristeći 2.4GHz pojas (isto kao i Wi-Fi). Komunikacija može biti jednostrana (engl. single-duplex) ili dvostrana (engl. full-duplex) i moguće je povezati od 2 do 20 mikrokontrolera. Paketi su limitirani na 250 bajtova.

2.1. Zašto (i zašto ne) koristiti ESP-NOW?

2.1.1. Usporedba ESP-NOW protokola s WiFi protokolom

Na sljedećoj tablici prikazana je usporedbe dometa i brzine WiFi i ESP-NOW protokola. Brzina je uspoređivana tako da je mjereno vrijeme da stigne poruka od 94 bajtova, a domet je mjeren tako da je jedan ESP ostao stacionaran a drugi se kretao od njega sve dok nije došlo do prekida konekcije[1] .

-	**ESP-NOW**	**WiFi**
Vrijeme potrebno da stigne poruka	37.25 ms	71.4 ms
Domet u zatvorenom prostoru	∼20 m	∼16 m
Domet u otvorenom prostoru	∼120 m	∼130 m

Tablica 2.1. Usporedba protokola bežične komunikacije dostupnih na ESP-u

ESP-NOW protokol je otprilike dva puta brži od WiFi-ja s više manje istim dometom.

2.1.2. Zašto?

Najočitija stvar jest to da nam ne treba nekakva WiFi infrastruktura (ruteri) koji sa sobom povlače problem malog dometa kao i potrebne uspostave konekcije (upis SSID-a

[1]Stvarni domet u otvorenom prostoru je teoretski veći (∼450 m) ako se koriste idealni uvjeti (nema nikakvih prepreka, interferencije, mikrokontroleri postavljeni visoko u zrak i sl.).

i lozinke) i održavanje iste, te ne ovisimo o pružatelju usluge. Također, ostvarit ćemo manju potrošnju energije jer se ne mora održavati konekcija.

Praktično je za male projekte gdje bi korištenje nekog kompleksnijeg protokola uvelo dodatan nepotreban *overhead* (dodatno vrijeme).

2.1.3. Zašto ne?

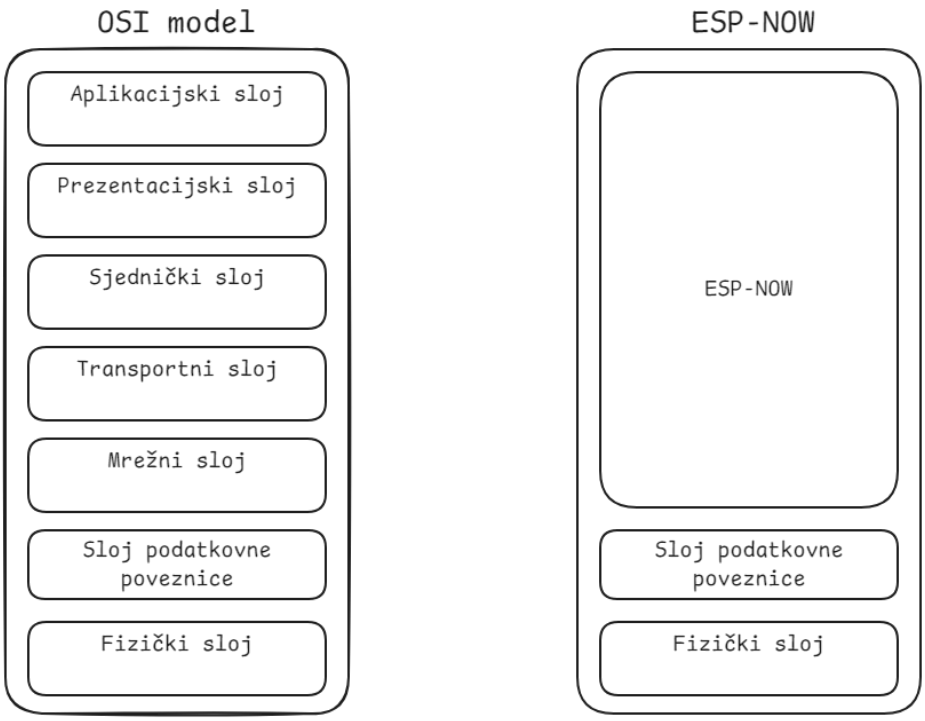
S obzirom na to da su paketi limitirani na samo 250 bajtova, ne možemo koristiti ESP-NOW ako želimo slati video, zvuk i sl. stvari (iako postoje načini kako da ih šaljemo, npr. da video razdvojimo na više paketa ali nije baš praktično). Također, ne možemo dohvaćati podatke s interneta, npr. temperatura mjesta.

2.2. Primjene ESP-NOW protokola

Gledajući prednosti i nedostatke protokola, možemo zaključiti da bi najbolje bilo koristiti protokol za očitanje podataka sa senzora ili slanje nekakvih naredbi drugom uređaju npr. pametna kuća, mreže senzora, sustavima daljinskog upravljanja, pametna poljoprivreda.

3. Kako radi?

ESP-NOW protokol smanjuje klasični OSI model koji se sastoji od sedam slojeva na samo tri (sl 3.1.). Time smanjuje *overhead* potreban za razmjenu paketa između slojeva i samo vrijeme potrebno za poslati isti.



Slika 3.1. Usporedba klasičnog OSI modela s modelom ESP-NOW protkola

Svaki uređaj ima ulogu pošiljatelja i/ili primatelja poruka (obje uloge ako se koristi dvostrani način komunikacije). Uređaji se povezuju s pomoću MAC adresa.

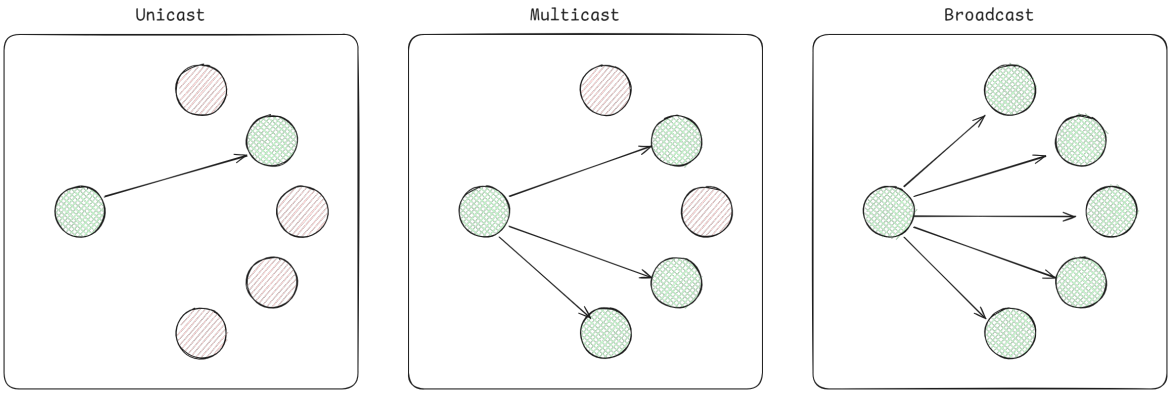
3.1. Načini komunikacije

Usporedba prikazana na slici (sl 3.2.)

Jedan na jedan (engl. Unicast One-to-One)
Međusobna komunikacija dva uređaja.

Jedan na više (engl. Multicast One-to-Many)
Jedan uređaj šalje na više uređaja poruku.

Više na jedan (engl. Broadcast Many-to-One)
Ovaj način podrazumijeva slanje paketa svim uređajima. Mogu ali i ne moraju biti povezani (stavljanjem FF:FF:FF:FF kao MAC adrese poruka se šalje svima).



Slika 3.2. Usporedba različitih načina komunikacije

3.2. ESP-NOW API

Detaljnije se može naći na ovom linku, a u nastavku će biti opisane funkcije koje će se koristiti u kodu.

3.2.1. Funkcije inicijalizacije

Funkcije u ovom dijelu služe za inicijalizaciju potrebnog sklopovlja za komunikaciju.

```
esp_err_t nvs_flash_init(void)
```

Funkcija služi za inicijalizaciju NVS-a (engl. Non-Volatile Storage) koji je potreban za WiFi (spremanje WiFi konfiguracije, ključeva i sl).

```
esp_err_t esp_netif_init(void)
```

Funkcija služi za inicijalizaciju TCP/IP stoga.

`esp_err_t esp_event_loop_create_default(void)`

Funkcija napravi petlju događaja koja omogućava drugim komponentama da registriraju *handle*.

`esp_netif_t *esp_netif_create_default_wifi_ap(void)`

Funkcija napravi *defaultni* AP (engl. Access Point - *hotspot*).

`esp_err_t esp_wifi_init(const wifi_init_config_t *config)`

Inicijalizacija WiFi *drivera* i pokretanje WiFi zadatka. Potrebno je pozvati prije bilo koje druge `wifi` naredbe.

`esp_err_t esp_wifi_set_mode(wifi_mode_t mode)`

Postavljenje željenog načina rada (AP, STA ili APSTA).

`esp_err_t esp_wifi_set_config(wifi_interface_t interface, wifi_config_t *conf)`

Postavljenje željene konfiguracije specificirane u `*conf` varijabli koja je za AP način definirana:

```
typedef struct {
    uint8_t peer_addr[ESP_NOW_ETH_ALEN];
    esp_now_role_t role;
    uint8_t channel;
    uint8_t encrypt;
    uint8_t lmk[ESP_NOW_KEY_LEN];
} esp_now_peer_info_t;
```

`esp_err_t esp_wifi_start(void)`

Pokretanje WiFi sučelja s zadanom konfiguracijom.

`esp_err_t esp_now_init(void)`

Inicijalizira potrebno za ESP-NOW protokol. Potrebno je pokrenuti inicijalizaciju WiFi-ja prije poziva ove funkcije jer ESP-NOW koristi WiFi.

`esp_err_t esp_now_add_peer(const esp_now_peer_info_t *peer)`

Dodaje uređaj za komunikaciju koji je definiran sljedećom strukturom:

```
typedef struct {
    uint8_t peer_addr[ESP_NOW_ETH_ALEN];
    esp_now_role_t role;
    uint8_t channel;
    uint8_t encrypt;
    uint8_t lmk[ESP_NOW_KEY_LEN];
} esp_now_peer_info_t;
```

3.2.2. Funkcije za web server

Funkcije u ovom dijelu služe za upravljanje web serverom (prihvaćanje i obrada zahtjeva).

`esp_err_t httpd_start(httpd_handle_t *handle, const httpd_config_t *config)`
Pokreće web server i alocira memoriju i resurse potrebne za isti kako je zadano u konfiguraciji.

`esp_err_t httpd_register_uri_handler(httpd_handle_t handle, const httpd_uri_t *uri_handler)`
Registriraj URI handler.

`esp_err_t httpd_req_get_url_query_str(httpd_req_t *r, char *buf, size_t buf_len)`
Dohvati string iz URL-a

`esp_err_t httpd_query_key_value(const char *query, const char *key, char *val, size_t val_size);`
Dohvaćanje vrijednosti iz niza oblika: `param1=val1&param2=val2`.

`esp_err_t httpd_resp_send(httpd_req_t *req, const char *buf, ssize_t buf_len);`
Služi za slanje HTTP odgovora. Podrazumijeva da se cijeli odgovor nalazi u *bufferu*, ako želimo slati u dijelovima koristiti `httpd_resp_send_chunk()`

`esp_err_t httpd_resp_set_type(httpd_req_t *r, const char *type)`

Postavlja HTTP content type polje odgovora, podrazumijevana vrijednost je `text/html`.

3.2.3. Funkcije za komunikaciju

Funkcije u ovom dijelu služe za komunikaciju ESP-NOW protokolom.

```
esp_err_t esp_now_send(const uint8_t *peer_addr,
const uint8_t *data, size_t len)
```

Slanje poruke ESP-NOW protokolom.

```
esp_err_t esp_now_register_recv_cb(esp_now_recv_cb_t cb)
```

Specificiranje callback funkcije (funkcija koja se automatski poziva kada se primi poruka) za primanje ESP-NOW protokolom.

```
esp_err_t esp_now_register_send_cb(esp_now_send_cb_t cb)
```

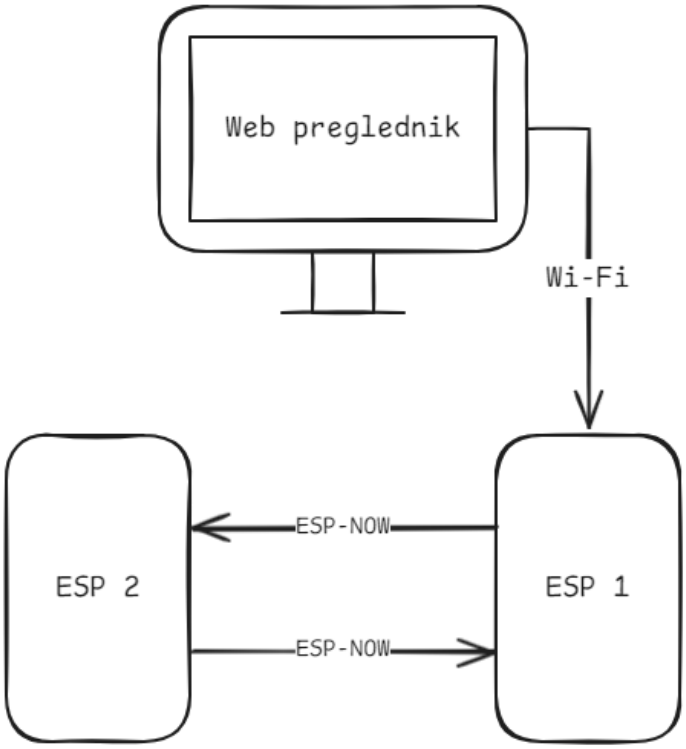
Isto kao i prethodna samo za slanje poruke.

3.3. Sigurnost

ESP-NOW protokol ima mogućnost slanja enkriptiranih poruka između uređaja. Za enkripciju koristi CCMP metodu uz primarni i lokalni ključ. Primarni ključ se koristi da bi enkriptirao lokalni ključ s AES-128 algoritmom, dok se lokalni koristi za enkripciju poruke CCMP metodom.

4. Primjer sustava

Sustav se sastoji od dva ESP mikroupravljača. Na jednom od njih se vrti web server na koji se možemo spojiti s pomoću računala i on šalje ESP-NOW poruke drugom mikroupravljaču koji prima poruke i obrađuje ih i povremeno šalje poruke prvom upravljaču. Osnovna shema sustava je prikazana na slici 4.1



Slika 4.1. Osnovna shema sustava

4.1. Povezivanje mikrokontrolera i komunikacija s ESP-NOW

Budući da ESP-NOW protokol koristi MAC adresu uređaja za međusobno povezivanje, prvo moramo saznati MAC adresu što možemo napraviti funkcijom `esp_wifi_get_mac`.

Kada znamo MAC adresu možemo uspostaviti ESP-NOW vezu i slati podatke međusobno. Prvo je potrebno inicijalizirati ESP-NOW protokol i registrirati drugi mikrokontroler što je prikazano sljedećim isječkom koda:

```
nvs_flash_init();

// inicijalizacija WiFi-ja
esp_netif_init();
esp_event_loop_create_default();

wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
esp_wifi_init(&cfg);
esp_wifi_set_mode(WIFI_MODE_APSTA);
esp_wifi_start();

// inicijalizacija ESP-NOW
if (esp_now_init() != ESP_OK) {
    ESP_LOGE(TAG, "ESP-NOW Init Failed");
    return;
}

// registriranje callback funckija za primanje i slanje podataka
esp_now_register_send_cb(on_data_sent);
esp_now_register_recv_cb(on_data_recv);

// spajanje s drugim uredjajem
esp_now_peer_info_t peerInfo = {};
memcpy(peerInfo.peer_addr, sender_mac, 6);
peerInfo.channel = 0;
peerInfo.ifidx = ESP_IF_WIFI_AP;
peerInfo.encrypt = false;

if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    ESP_LOGE(TAG, "Failed to add peer");
    return;
}

ESP_LOGI(TAG, "ESP-NOW Initialized Successfully");
```

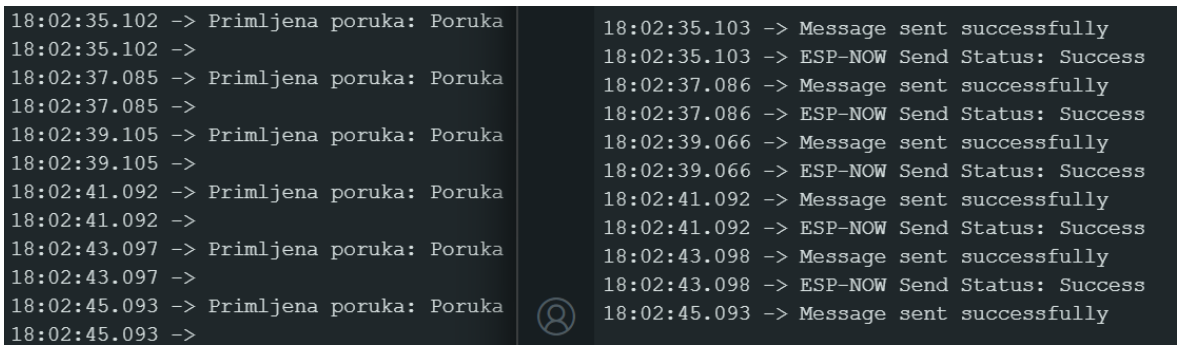
Potrebno je inicijalizirati i WiFi sučelje jer ESP-NOW koristi isto. Poruku možemo poslati sljedećim isječkom:

```
char message[] = "Poruka";

esp_err_t result = esp_now_send(receiver_mac,
    (uint8_t *)message,
    sizeof(message));

if (result == ESP_OK) {
    ESP_LOGI(TAG, "Message sent successfully");
} else {
    ESP_LOGE(TAG, "ESP-NOW send failed: %s", esp_err_to_name(result));
}
```

Primjer komunikacije možemo vidjeti na sljedećem ispisu:



Slika 4.2. Primjer ispisa ESP-NOW protokolom

Ako dodatno želimo i mi komunicirati sa sustavom, najpraktičnije bi bilo pomoću nekakvog web sučelja na koji se možemo spojiti pomoću web preglednika. Radi toga, možemo na jednom mikroupravljaču *vrtjeti* i web server na koji se povezujemo. Za to nam je potreban WiFi:

```
// inicijalizacija WiFi
esp_netif_init();
esp_event_loop_create_default();

esp_netif_t *esp_netif = esp_netif_create_default_wifi_ap();
wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
esp_wifi_init(&cfg);

esp_wifi_set_mode(WIFI_MODE_AP);
```

```
wifi_config_t ap_config = {
    .ap = {
        .ssid = "ESP32_AP",
        .ssid_len = strlen("ESP32_AP"),
        .channel = 1,
        .password = "12345678",
        .max_connection = 4,
        .authmode = WIFI_AUTH_WPA_WPA2_PSK
    }
};
esp_wifi_set_config(WIFI_IF_AP, &ap_config);
esp_wifi_start();
```

Odsječak *handlera* za http zahtjev:

```
char query[128];
char message[64] = {0};

if (httpd_req_get_url_query_str(req, query, sizeof(query)) == ESP_OK)
{
    ESP_LOGI(TAG, "Received Query: %s", query);
    if (httpd_query_key_value(query, "message", message,
        sizeof(message)) == ESP_OK) {
        ESP_LOGI(TAG, "Extracted Message: %s", message);

        // slanje poruke esp now protokolom
        esp_err_t result = esp_now_send(receiver_mac,
            (uint8_t *)message, strlen(message));
        if (result == ESP_OK) {
            httpd_resp_send(req, "ESP-NOW Message Sent!",
                HTTPD_RESP_USE_STRLEN);
            return ESP_OK;
        } else {
            httpd_resp_send(req, "ESP-NOW Message Failed!",
                HTTPD_RESP_USE_STRLEN);
            return ESP_OK;
        }
    }
```

```
23 }
```

Uz pomoć `httpd_req_get_url_query_str` iz zahtjeva izvlačimo parametar koji smo upisali, npr. ako pošaljemo zahtjev `http://192.168.4.1/send?message=hello` (IP adresu ispiše ESP kada je spreman), `httpd_req_get_url_query_str` funkcija će nam vratiti `message=hello`, a potom uz pomoć funkcije `httpd_query_key_value` dobivamo `hello`. `hello`, koji smo pohranili u varijablu `message[]`, šaljemo ESP-NOW protokolom na drugi mikroupravljač.

5. Zaključak

TODO

Literatura

Privitak A: Kod

```
1  //dohvacanje MAC
2  void print_mac_address() {
3      uint8_t mac[6];  // Array to store MAC address
4
5      // Get the MAC address for the WiFi station (WIFI_IF_STA)
       interface
6      esp_err_t ret = esp_wifi_get_mac(WIFI_IF_STA, mac);
7      if (ret == ESP_OK) {
8          ESP_LOGI(TAG, "ESP32 MAC Address: %02X:%02X:%02X:%02X:%02X:%02
           X",
9                   mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);
10     } else {
11         ESP_LOGE(TAG, "Failed to get MAC Address!");
12     }
13 }
```

```c
#include <stdio.h>
#include <string.h>
#include "esp_wifi.h"
#include "esp_event.h"
#include "esp_log.h"
#include "esp_now.h"
#include "nvs_flash.h"
#include "esp_netif.h"
#include "esp_http_server.h"

#define TAG "ESP_NOW_SENDER"

uint8_t receiver_mac[] = {0x34, 0x86, 0x5D, 0xDC, 0x20, 0xD0};

/*
 * Callback funkcija za slanje podataka
 *
 * mac_addr:    MAC adresa uredjaja sa kojeg saljemo podatke
 * status:      status poslane poruke
 *
 * return:      void
 */
void on_data_sent(const uint8_t *mac_addr, esp_now_send_status_t
status) {
    ESP_LOGI(TAG, "ESP-NOW Send Status: %s",
        (status == ESP_NOW_SEND_SUCCESS) ? "Success" : "Fail");
}

/*
 * Callback funkcija za slanje podataka
 *
 * recv_info:   podatci o uredjaju sa kojeg smo primili poruku
 * data:        podatci koje primamo
 * len:         velicina polja data
 *
 * return:      void
 */
void on_data_recv(const esp_now_recv_info_t *recv_info,
```

```
38      const uint8_t *data, int len) {
39      //const uint8_t *mac_addr = recv_info->src_addr;
40
41      //ESP_LOGI("ESP-NOW", "Received data from: %02X:%02X:%02X:%02X:%02
        X:%02X",
42      //         mac_addr[0], mac_addr[1], mac_addr[2],
43      //         mac_addr[3], mac_addr[4], mac_addr[5]);
44
45      ESP_LOGI("ESP-NOW", "Data: %.*s", len, data);
46  }
47
48  /*
49   * Funkcija za inicijalizaciju WiFi-ja u AP nacinu rada kako bi se
50   * mogli povezati preko browsera i slati poruke
51   *
52   * return:      void
53   */
54  void init_wifi_ap() {
55      esp_netif_init();
56      esp_event_loop_create_default();
57
58      esp_netif_t *esp_netif = esp_netif_create_default_wifi_ap();
59      wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
60      esp_wifi_init(&cfg);
61
62      esp_wifi_set_mode(WIFI_MODE_AP);
63      wifi_config_t ap_config = {
64          .ap = {
65              .ssid = "ESP32_AP",
66              .ssid_len = strlen("ESP32_AP"),
67              .channel = 1,
68              .password = "12345678",
69              .max_connection = 4,
70              .authmode = WIFI_AUTH_WPA_WPA2_PSK
71          }
72      };
73      esp_wifi_set_config(WIFI_IF_AP, &ap_config);
74      esp_wifi_start();
75
```

```
76      esp_netif_ip_info_t ip_info;
77      esp_netif_get_ip_info(esp_netif, &ip_info);
78      ESP_LOGI(TAG, "ESP32 AP IP Address: " IPSTR, IP2STR(&ip_info.ip));
79  }
80
81  /*
82   * Funkcija za inicijalizaciju ESPNOW protokola
83   *
84   * return:      void
85   */
86  void init_espnow() {
87      // inicijalizacija
88      if (esp_now_init() != ESP_OK) {
89          ESP_LOGE(TAG, "ESP-NOW Init Failed");
90          return;
91      }
92
93      // registriranje callback funckija za primanje i slanje podataka
94      esp_now_register_send_cb(on_data_sent);
95      esp_now_register_recv_cb(on_data_recv);
96
97      // spajanje s drugim uredjajem
98      esp_now_peer_info_t peerInfo = {};
99      memcpy(peerInfo.peer_addr, receiver_mac, 6);
100     peerInfo.channel = 0;
101     peerInfo.ifidx = ESP_IF_WIFI_AP;
102     peerInfo.encrypt = false;
103
104     if (esp_now_add_peer(&peerInfo) != ESP_OK) {
105         ESP_LOGE(TAG, "Failed to add peer");
106         return;
107     }
108
109     ESP_LOGI(TAG, "ESP-NOW Initialized Successfully");
110 }
111
112 esp_err_t get_handler(httpd_req_t *req) {
113     char query[128];
114     char message[64] = {0};
```

```
115
116     if (httpd_req_get_url_query_str(req, query, sizeof(query)) ==
        ESP_OK) {
117         ESP_LOGI(TAG, "Received Query: %s", query);
118         if (httpd_query_key_value(query, "message", message,
119             sizeof(message)) == ESP_OK) {
120             ESP_LOGI(TAG, "Extracted Message: %s", message);
121
122             // slanje poruke esp now protokolom
123             esp_err_t result = esp_now_send(receiver_mac,
124                 (uint8_t *)message, strlen(message));
125             if (result == ESP_OK) {
126                 httpd_resp_send(req, "ESP-NOW Message Sent!",
127                     HTTPD_RESP_USE_STRLEN);
128                 return ESP_OK;
129             } else {
130                 httpd_resp_send(req, "ESP-NOW Message Failed!",
131                     HTTPD_RESP_USE_STRLEN);
132                 return ESP_OK;
133             }
134         }
135     }
136
137     // html stranica
138     const char *html_response =
139         "<html>"
140         "<head><title>ESP32 Sender</title></head>"
141         "<body>"
142         "<h2>Send a Message via ESP-NOW</h2>"
143         "<form action='/send' method='get'>"
144         "  <input type='text' name='message' placeholder='Poruka'
        required>"
145         "  <button type='submit'>Send</button>"
146         "</form>"
147         "</body>"
148         "</html>";
149
150     httpd_resp_set_type(req, "text/html");
151     httpd_resp_send(req, html_response, HTTPD_RESP_USE_STRLEN);
```

```
152     return ESP_OK;
153 }
154 
155 httpd_handle_t start_server() {
156     httpd_config_t config = HTTPD_DEFAULT_CONFIG();
157     httpd_handle_t server = NULL;
158 
159     if (httpd_start(&server, &config) == ESP_OK) {
160         httpd_uri_t uri_get = {
161             .uri       = "/send",
162             .method    = HTTP_GET,
163             .handler   = get_handler,
164             .user_ctx  = NULL
165         };
166         httpd_register_uri_handler(server, &uri_get);
167     }
168     return server;
169 }
170 
171 /*
172  * Glavna funkcija iz koje se pozivaju sve potrebne inicijalizacije
173  *
174  * return:      void
175  */
176 void app_main() {
177     nvs_flash_init();
178     init_wifi_ap();
179     init_espnow();
180 
181     // Message to send
182     char message[] = "Hello from ESP-NOW Sender!";
183 
184     // Send data
185     esp_err_t result = esp_now_send(receiver_mac,
186         (uint8_t *)message,
187         sizeof(message));
188 
189     if (result == ESP_OK) {
190         ESP_LOGI(TAG, "Message sent successfully");
```

```
191     } else {
192         ESP_LOGE(TAG, "ESP-NOW send failed: %s", esp_err_to_name(
            result));
193     }
194 
195     start_server();
196     ESP_LOGI(TAG, "ESP-NOW Sender with Web Server Ready!");
197 }
```

```c
#include <stdio.h>
#include <string.h>
#include "esp_now.h"
#include "esp_wifi.h"
#include "esp_event.h"
#include "esp_log.h"
#include "nvs_flash.h"

#define TAG "ESP_NOW_RECEIVER"

uint8_t sender_mac[] = {0x7C, 0x87, 0xCE, 0xF8, 0xC9, 0x59};

/*
 * Callback funkcija za slanje podataka
 *
 * mac_addr:    MAC adresa uredjaja sa kojeg saljemo podatke
 * status:      status poslane poruke
 *
 * return:      void
 */
void on_data_sent(const uint8_t *mac_addr, esp_now_send_status_t
status) {
    ESP_LOGI(TAG, "ESP-NOW Send Status: %s",
        (status == ESP_NOW_SEND_SUCCESS) ? "Success" : "Fail");
}

/*
 * Callback funkcija za slanje podataka
 *
 * recv_info:   podatci o uredjaju sa kojeg smo primili poruku
 * data:        podatci koje primamo
 * len:         velicina polja data
 *
 * return:      void
 */
void on_data_recv(const esp_now_recv_info_t *recv_info,
    const uint8_t *data, int len) {
    //const uint8_t *mac_addr = recv_info->src_addr;
```

```
38
39      //ESP_LOGI("ESP-NOW", "Received data from: %02X:%02X:%02X:%02X:%02
        X:%02X",
40      //          mac_addr[0], mac_addr[1], mac_addr[2],
41      //          mac_addr[3], mac_addr[4], mac_addr[5]);
42
43      ESP_LOGI("ESP-NOW", "Data: %.*s", len, data);
44  }
45
46  /*
47   * Funkcija za inicijalizaciju WiFi-ja
48   *
49   * Potrebna jer ESPNOW protokol koristi WiFi pa je potrebno
     inicijalizirati
50   * hardver zasluzen za WiFi
51   *
52   * return:      void
53   */
54  void init_wifi() {
55      esp_netif_init();
56      esp_event_loop_create_default();
57
58      wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
59      esp_wifi_init(&cfg);
60      esp_wifi_set_mode(WIFI_MODE_APSTA);
61      esp_wifi_start();
62  }
63
64  /*
65   * Funkcija za inicijalizaciju ESPNOW protokola
66   *
67   * return:      void
68   */
69  void init_espnow() {
70      // inicijalizacija
71      if (esp_now_init() != ESP_OK) {
72          ESP_LOGE(TAG, "ESP-NOW Init Failed");
73          return;
74      }
```

```
75
76      // registriranje callback funckija za primanje i slanje podataka
77      esp_now_register_send_cb(on_data_sent);
78      esp_now_register_recv_cb(on_data_recv);
79
80      // spajanje s drugim uredjajem
81      esp_now_peer_info_t peerInfo = {};
82      memcpy(peerInfo.peer_addr, sender_mac, 6);
83      peerInfo.channel = 0;
84      peerInfo.ifidx = ESP_IF_WIFI_AP;
85      peerInfo.encrypt = false;
86
87      if (esp_now_add_peer(&peerInfo) != ESP_OK) {
88          ESP_LOGE(TAG, "Failed to add peer");
89          return;
90      }
91
92      ESP_LOGI(TAG, "ESP-NOW Initialized Successfully");
93  }
94
95  /*
96   * Glavna funkcija iz koje se pozivaju sve potrebne inicijalizacije
97   *
98   * return:      void
99   */
100 void app_main() {
101     // nvs potreban za WiFi
102     nvs_flash_init();
103     init_wifi();
104     init_espnow();
105
106     // testiranje ESPNOW konekcije
107     /* char message[] = "Poruka iz recievera";
108
109     esp_err_t result = esp_now_send(receiver_mac,
110         (uint8_t *)message,
111         sizeof(message));
112
113     if (result == ESP_OK) {
```

```
114             ESP_LOGI(TAG, "Message sent successfully");
115         } else {
116             ESP_LOGE(TAG, "ESP-NOW send failed: %s", esp_err_to_name(
                result));
117         } */
118
119         ESP_LOGI(TAG, "ESP-NOW Receiver Ready!");
120 }
```