|  |  |  |
| --- | --- | --- |
|  | UNIVERZITET U NOVOM SADU  **FAKULTET TEHNIČKIH NAUKA** |  |

Igor Šikuljak

|  |  |  |
| --- | --- | --- |
|  | **Sistem za praćenje kretanja mobilnih uređaja sa omogućenom upotrebom WiFi tehnologije** |  |

DIPLOMSKI RAD

- Osnovne akademske studije -

Novi Sad, 2021

|  |  |  |
| --- | --- | --- |
|  | UNIVERZITET U NOVOM SADU ⚫ **FAKULTET TEHNIČKIH NAUKA**  21000 NOVI SAD, Trg Dositeja Obradovića 6 | Datum: |
|  |
| **ZADATAK ZA IZRADU ZAVRŠNOG (BACHELOR) RADA** | List/Listova: |
| 1/1 |

*(Podatke unosi predmetni nastavnik - mentor)*

| Vrsta studija: | Osnovne akademske studije |
| --- | --- |
| Studijski program: | **Računarstvo i automatika** |
| Rukovodilac studijskog programa: | **prof. dr Milan Vidaković** |

|  |  |  |  |
| --- | --- | --- | --- |
| Student: | **Igor Šikuljak** | Broj indeksa: | **RA117/2017** |
| Oblast | **Računarske mreže, bezbjednost sistema, embedded sistemi** | | |
| Mentor: | **prof. dr Milan Vidaković** | | |
| NA OSNOVU PODNETE PRIJAVE, PRILOŽENE DOKUMENTACIJE I ODREDBI STATUTA FAKULTETA  IZDAJE SE ZADATAK ZA DIPLOMSKI (Bachelor) RAD, SA SLEDEĆIM ELEMENTIMA:   * problem – tema rada; * način rešavanja problema i način praktične provere rezultata rada, ako je takva provera neophodna; * literatura; | | | |

**NASLOV DIPLOMSKOG (BACHELOR) RADA:**

|  |
| --- |
| **Sistem za praćenje kretanja mobilnih uređaja sa omogućenom upotrebom WiFi tehnologije** |

**TEKST ZADATKA:**

|  |
| --- |
| Zadatak rada predstavlja kreiranje sistema za prikuplajnje podataka od značaja koji će omogućiti detekciju i praćenje kretanja uređaja koji koriste WiFi tehnologiju, bez obzira na status njihove konekcije na neku mrežu. Stoga, student treba da:   * prouči relevantne teorijske koncepte iz IEEE802.11 tehnologije te pronađe i osmisli metode koji će omogućiti prikupljanje potrebnih podataka * izabere embedded platformu na kojoj će realizovati osmišljeni sistem * implemetira osmišljeni sistem i izvrši testiranje istog * diskutuje realizovano rješenje i predloži dalja unapređenja |

|  |  |
| --- | --- |
| Rukovodilac studijskog programa: | Mentor rada: |
| **prof. dr Milan Vidaković** | **prof. dr Milan Vidaković** |

|  |
| --- |
| Primerak za:  - Studenta;  - Mentora |
|  |

**Sadržaj**

[1 Uvod 7](#_Toc83589641)

[2 Teorijske osnove 9](#_Toc83589642)

[2.1 IEEE 802.11 standard 9](#_Toc83589643)

[2.1.1 Distribucija kanala u 2,4GHz frekvencijskom spektru (EEE 802.11 b/g/n/ax) 9](#_Toc83589644)

[2.1.2 IEEE 802.11 frame 10](#_Toc83589645)

[2.1.3 Probe Request paketi 11](#_Toc83589646)

[2.1.4 Request to Send (RTS) i Clear to Send (CTS) paketi 12](#_Toc83589647)

[3 Arhitektura sistema 15](#_Toc83589648)

[3.1 Opšta struktura predloženog rješenja 15](#_Toc83589649)

[3.2 Platforma za realizaciju rješenja i njena ograničenja 15](#_Toc83589650)

[3.2.1 ESP8266 SoC i NodeMCU platforma 15](#_Toc83589651)

[3.2.2 Dodatni moduli – periferije 16](#_Toc83589652)

[3.2.3 Razvojni alati, šema sistema, platformska ograničenja i njihovo rješenja 16](#_Toc83589653)

[4 Implementacija 21](#_Toc83589654)

[4.1 SdFat biblioteka 21](#_Toc83589655)

[4.2 Konfiguracija glavnog-upravljačkog NodeMCU-a 21](#_Toc83589656)

[4.3 NTP sinhronizacija vremena 22](#_Toc83589657)

[4.4 Rad sa RTC satom 24](#_Toc83589658)

[4.4.1 Inicijalizacija RTC sata sa tačnim vremenom (u preciznosti sekunde) 24](#_Toc83589659)

[4.4.2 NodeMCU praćenje vremena u preciznosti milisekunde na osnovu sistemskih prekida 24](#_Toc83589660)

[4.5 Inicijalizacija sistema 25](#_Toc83589661)

[4.6 Aktivni režim rada glavnog NodeMCU uređaja 27](#_Toc83589662)

[4.7 Prikupljanje 802.11 paketa i njihova obrada 29](#_Toc83589663)

[4.8 Slanje komandi RTS odašiljaču 31](#_Toc83589664)

[4.9 RTS odašiljač 31](#_Toc83589665)

[5 Rezultati i diskusija 35](#_Toc83589666)

[6 Budući rad 37](#_Toc83589667)

[7 Zaključak 39](#_Toc83589668)

[8 Literatura 41](#_Toc83589669)

[Biografija 43](#_Toc83589670)

[KLJUČNA DOKUMENTACIJSKA INFORMACIJA 45](#_Toc83589671)

[KEY WORDS DOCUMENTATION 47](#_Toc83589672)

# Uvod

Mobilni uređaji koji upotrebljavaju WiFi tehnologiju [1], u svom normalnom režimu rada neizbježno odaju određene identifikacione podatke svojoj okolini. Kako je medijum kroz koji se prenose podaci zajednički za sve uređaje u okolini, tj. radi se o fizičkom prostoru u kome se podaci prenose bežično, tako jedini način za sprovođenje uspješne komunikacije jeste da svaki set prenošenih podataka sa sobom nosi informaciju/identifikaciju porijekla i destinacije. Jedinstveni identifikator svakog fizičkog uređaja u cijeloj IEEE 802 [2] grupi standarda jeste MAC (media access control) adresa.

Cilj ovog rada jeste kreiranje sistema koji će omogućiti praćenje kretanja mobilnih uređaja (WiFi stanica, WiFi station) u prostoru. Ovaj rad se ograničava na praćenje uređaja koji rade u okviru IEEE 802.11 b/g/n bežičnih mreža, što odgovara uobičajenim, opšteprisutnim verzijama 2,4Ghz WiFi tehnologije. Detekcija uređaja se postiže slušanjem komunikacije u neposrednoj okolini jedne jedinice našeg sistema te prikupljanjem relevantnih podataka za identifikaciju uređaja. Od interesa su specifični paketi u okviru komunikacije između mobilnih uređaja i WiFi pristupnih tački (Access Point, AP), a to su Probe Request paketi. Probe Request pakete šalju mobilni uređaji sa ciljem pronalaženja WiFi pristupnih tački u okolini kao i utvrđivanja kvaliteta njihovog signala.

Dodatno se razmatra i aktivna transmisija RTS (Request to Send) paketa sa ciljem povećanja broja i učestalosti slanja Probe Request paketa od strane mobilnih uređaja, što bi omogućavalo i precizniju lokaciju i praćenje istih.

# Teorijske osnove

## IEEE 802.11 standard

IEEE 802.11 je set standarda koji definiše komunikacione protokole u bežičnom aspektu mreža lokalnog nivoa (Wireless Local Area Networks, WLAN). U nastavku će biti objašnjeni relevantni dijelovi tog standarda za ovaj rad.

### Distribucija kanala u 2,4GHz frekvencijskom spektru (EEE 802.11 b/g/n/ax)

Kolizije elektromagnetnih signala narušavaju integritet poslatih podataka, te je je u tim slučajevima neophodno izvršiti retransmisiju datog podatka. Najjednostavniji primjeri za to jesu ako dva uređaja pokušaju da pošalju pakete u istom trenutku na istom frekvencijskom opsegu, ili jednostavnno prirodna ili vještačka interferencija iz okoline. Učestalost potrebe za retransmisijom paketa utiče na performanse uređaja. Ovaj problem je značajan u okolnostima kada postoji više WiFi pristupnih tački na istom mjestu, te se onda date mreže međusobno ometaju. Zbog toga, alocirani frekvencijski spektar za mreže koje prate IEEE 802.11 b/g/n ili ax standarde je podijeljen na više kanala. Svaka mreža radi na najmanje jednom kanalu, i tako se može obezbijediti potpuno nesmetana komunikacija u datoj tački prostora za najmanje tri mreže. Spektar je podijeljen na ukupno 14, međusobno djelimično preklapajućih kanala širine 22 MHz, sa idejom da ako nije moguće potpuno eliminisati koliziju između mreža, onda se ona može makar djelimično umanjiti.



Figure 1 - Kanali u 2,4GHz spektru, njihovo preklapanje i širina (Izvor: Michael Gauthier, CC BY-SA 3.0)

Kanali 1-13 su kanali koji su dozvoljeni zakonskim regulativama u većem dijelu svijeta, dok to nije slučaj za kanal 14. Zbog toga, kanal 14 nije uzet u obzir u implementaciji ovog rada.

Mreže obično funkcionišu po HT20 (High Throughput 20) principu što znači da za svoje funkcionisanje koriste jedan od pomenutih kanala u opsegu od 20Mhz (preostala 2 MHz su raspoređena na krajevima opsega i ne koriste se – služe rasipanju, slabljenju signala ka narednom neometanom signalu). IEEE 802.11n standard definiše i HT40 princip upotrebe alociranog frekvencijskog spektra i on podrazumjeva istovremenu upotrebu dva nepreklapajuća kanala od strane jedne WiFi pristupne tačke. U tom slučaju se jedan kanal koristi za prenos paketa sa podacima, a drugi za ostale pakete – ukupno 40MHz iskorišćenog frekvencijsog opsega, te otuda i naziv HT40.

### IEEE 802.11 frame

Sva komunikacija koja odvija preko WiFi mreže, bez obzira na protokol, se enkapsulira u 802.11 frame. Polja u tzv. MAC zaglavlju definišu razne osobine specifičnog paketa, kao što su npr. tip paketa, adresa porijekla, adresa destinacije, stanje paketa, kontrolni podaci… Ispod se nalazi slika na kome je prikazano MAC zaglavlje sa dužinom polja izraženom u bajtima.



Figure 2 - Polja u generičkom 802.11 frame-u [3]

Kao što se može vidjeti, postoji više adresnih polja od samo porijekla i destinacije. To je zbog toga što konfiguracija mreže može biti znatno kompleksnija od tipa stanica-pristupna tačka, i može da uključuje posrednike na obje strane te veze. Da bi se adresna polja mogla ispravno tumačiti, u okviru Frame Control polja se definiše distribucioni sistem vezan za dati paket.



Figure 3 - Polja prisutna unutar Frame Control polja u 802.11 frame-u [3]

Polja ToDS i FromDS definišu okruženje u kakvom je dati paket koji je u tranzitu. U slučaju direktne komunikacije oba bita (polja, zastavice) neće biti setovana, dok u slučajevima kada paket napušta/namijenjen je za distribucioni sistem, tada će respektivno FromDS i ToDS biti setovani. Moguća je situacija i kada su oba bita setovana.

Adresna polja mogu imati neka od sledećih značenja:

* RA (Receiver Address) – adresa primaoca
* TA (Transmitter Address) – adresa pošiljaoca
* DA (Destination Address) – adresa destinacije (krajnji primaoc paketa)
* SA (Source Address) – adresa izvora (originalni pošiljalac paketa)
* BSSID (Basic service set identifier) – adresa nadležne mreže

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **To DS** | **From DS** | **Address 1** | **Address 2** | **Address 3** | **Address 4** |
| 0 | 0 | RA = DA | TA = SA | BSSID | N/A |
| 0 | 1 | RA = DA | TA = BSSID | SA | N/A |
| 1 | 0 | RA = BSSID | TA = SA | DA | N/A |
| 1 | 1 | RA | TA | DA | SA |

Figure 4 - Značenje sadržaja adresnih polja na osnovu sadržaja DS polja [4]

### Probe Request paketi

WiFi stanice emituju Probe request pakete sa ciljem otkrivanja dostupnih pristupnih tačaka na datoj lokaciji. Paketi su obično broadcast tipa, što znači da su usmjereni ka svim pristupnim tačkama, i od svih njih koji su u dometu se očekuje odgovor. Broadcast paketi se identifikuju po jedinstvenoj adresi (MAC adresa) destinacije (DA) – FF:FF:FF:FF:FF:FF. Sa druge strane, Probe request paketi mogu biti usmjereni i ka specifičnoj pristupnoj tački. WiFi stanice emituju takve pakete uglanom sa razlogom da otkriju promjene u kvalitetu signala od određene pristupne tačke.

U skladu sa prethodno iznijetim informacijama, mobilni uređaji emituju Probe request pakete da bi otkrili pristupne tačke u blizini i povezali se na prethodno poznatu mrežu, ili da bi mogli inicirati novu konekciju. Osim toga, uređaji periodično emituju Probe request pakete bez obzira da li su u datom trenutku konektovani na određenu mrežu ili ne, sa ciljem utvrđivanja kvaliteta signala pristupnih tačaka. To omogućuje automatizovani prelazak sa jedne mreže na drugu poznatu mrežu u zavisnosti od kvaliteta signala i time pružanje bolje usluge.

Probe request paketi pripadaju Management tipu paketa (u 802.11 frame-u oznaka tipa u frame control sekciji nosi vrijednost 0) i imaju oznaku podtipa u 802.11 frame control sekciji sa vrijednošću 4.



Figure 5 - Primjer Probe request paketa

### Request to Send (RTS) i Clear to Send (CTS) paketi

RTS predstavlja mehanizam za razrješavanje problema sa kolizijama u slanju paketa. Osnovna ideja jeste da, kada se desi kolizija, umjesto ponovnog slanja samog paketa, pošiljalac primaocu prvo pošalje Request to Send (zahtjev za slanje) paket, na koji će primaoc da odgovori sa Clear to Send (dozvoljeno slanje) paketom. Nakon te razmjene, pošiljalac može da pošalje paket primaocu, a primaoc neće preduzimati nikakve akcije slanja na određeno vrijeme. Takođe, drugi uređaji na mreži koji nisu uključeni u datu komunikaciju će takođe obustaviti slanje svojih paketa odmah po primanju RTS paketa, a nastaviće svoj normalni rad tek nakon registrovanja CTS paketa, odnosno kada prođe unaprijed određeni vremenski interval.

Request to send paketi pripadaju Control tipu paketa (u 802.11 frame-u oznaka tipa u frame control sekciji nosi vrijednost 1) i imaju oznaku podtipa u 802.11 frame control sekciji sa vrijednošću 11.



Figure 6 - Primjer Request to Send paketa

Za ovaj rad je značajno je napomenuti da CTS paketi u svom 802.11 zaglavlju ne sadrže adresu pošiljaoca, već samo adresu primaoca.

Clear to send paketi pripadaju Control tipu paketa (u 802.11 frame-u oznaka tipa u frame control sekciji nosi vrijednost 1) i imaju oznaku podtipa u 802.11 frame control sekciji sa vrijednošću 12.



Figure 7 - Primjer Clear to Send paketa

# Arhitektura sistema

## Opšta struktura predloženog rješenja

Ideja za realizaciju sistema se sastoji u postavljanju više malih uređaja – jedinica sistema, raspoređenih na određenom prostoru. Svaki od tih uređaja će imati sposobnost da sluša mrežni saobraćaj iz svoje okoline, izdvaja iz njega sve Probe Request pakete i bilježi relevantne identifikacione podatke. Naknadno se okupljaju podaci od svih pojedinačnih uređaja, povezani sa informacijom o lokaciji svakog uređaja koji je prikupljao podatke i vremenskim trenutkom kada je svaki podatak registrovan. Na osnovu tih potataka je moguće rekonstruisati kretanje mobilnih uređaja prostim utvrđivanjem kada je određeni mobilni uređaj bio u okolini naše jedinice sistema – preko činjenice da li je jedinica sistema tada registrovala Probe Request-ove od datog mobilnog uređaja.

Pored praćenja Probe Request paketa, razmatra si i ciljano slanje RTS paketa. Od tog procesa se kao posljedica očekuje povećan broj Probe Requestova od ciljanog mobilnog uređaja (WiFi stanice), jer se očekuje jedan od dva scenarija. Ako je mobilni uređaj povezan na WiFi mrežu, postojanje RTS paketa može prouzrokovati potragu za novom mrežom koja će biti manje opterećena i time pružati bolju uslugu, ili, ako je mobilni uređaj nepovezan i u periodičnoj potrazi za mrežama, ubrzaće dolazak novog ciklusa emitovanja Probe Request-ova na red. Sa druge strane, takođe bi bilo moguće slušanje na CTS poruke što bi potvrdilo prisustvo mobilnog uređaja u neposrednoj blizini naše jedinice sistema. Kao što je navedeno u poglavlju 2.4.1, CTS paketi ne sadrže adresu porijekla paketa, pa to blago komplikuje pristup praćenju kretanja više uređaja, jer CTS odgovor se mora vezivati za posljednji poslati RTS paket (vezivanje za njegovu destinaciju). Pored toga, ne bi bilo moguće brzo slanje velikog broja RTS paketa namijenjenih različitim ciljnim uređajima, jer bi to moglo dovesti do preplitanja odgovora, te je potrebno praviti sitnije pauze između slanja RTS paketa različitim ciljanim WiFi stanicama.

## Platforma za realizaciju rješenja i njena ograničenja

### ESP8266 SoC i NodeMCU platforma

Za implementaciju predloženog rješenja je izabran SoC (System on a Chip) ESP8266 koji proizvodi Espressif Systems [5], odnosno ESP8266 Dev Kit platforma poznata pod nazivom NodeMCU [6]. Dati izbor je napravljen jer pruža podršku za većinu funkcionalnosti koje su potrebne za implementaciju predstavljenog rješenja, a i zbog svoje izrazito niske cijene.

### Dodatni moduli – periferije

Prikupljene podatke je potrebno potkrijepiti tačnim vremenom njihovog prikupljanja i skladištiti ih. Za precizno praćenje vremena se koristi RTC (Real Time Clock) modul sa DS3231 čipom. Bitno je napomenuti da je za ovaj projekat neophodna verzija čipa DS3231S(N) [7] jer ona ima programabilan generator signala (SQW, Square Wave Generator), dok druge verzije, kao npr. DS3231M [8], ga nemaju. Za povezivanje RTC sata na NodeMCU se koristi I2C (Inter-Integrated Circuit) protokol. Skladištenje podata se obavlja na (mikro) SD memorijsku karticu. SD kartice imaju SPI interfejs, pa se direktno povezuju na NodeMCU i sa njima se komunicira SPI (Serial Peripheral Interface) protokolom.

### Razvojni alati, šema sistema, platformska ograničenja i njihovo rješenja

Za programiranje ESP8266 postoji ESP8266 NON-OS SDK [9], ali programiranje u njemu za veći dio sistema se radi u Arduino Framework-u (ESP8266 core for Arduino) [10] koji omogućava značajno jednostavniju implementaciju. ESP8266 core for Arduino u svojoj pozadini koristi ESP8266 NON-OS SDK, pa sve njegove funkcionalnosti su dostupne i unutar samog ESP8266 core for Arduino Framework-a.

ESP8266 zajedno sa svojim ESP8266 NON-OS SDK ima podršku za tzv. promiscuous mode rada na mreži, tj. može da prati mrežne pakete koji se šalju u njegovoj okolini, a nije ograničen na samo pakete koji pripadaju mreži sa kojom je trenutno asociran. Prema tehničkoj specifikaciji [11] i API dokumentaciji [12] podržani su samo paketi u HT20 principu rada, i postoji ograničenje da se prihvataju na korektnu obradu samo paketi koji prolaze određene kriterijume. Jedan od njih jeste da MAC zaglavlje (a preko toga i 802.11 paket u cjelosti) mora imati minimalnu dužinu od 24 bajta. Međutim, kao što se to može vidjeti na primjeru za CTS paket u poglavlju 2.1.4, dužina CTS paketa je manja od toga. To znači da nije moguće registrovanje CTS paketa.

U API dokumentaciji [12] se može vidjeti da ESP8266 ima mogućnost slanja tzv. freedom paketa, tj. moguće je slanje programski definisanih 802.11 paketa bez potrebe da ESP8266 bude asociran ili u procesu asocijacije sa nekom WiFi mrežom. Nažalost, ta funkcionalnost je ograničena na slanje neenkriptovanih Data paketa, Beacon, Probe Request i Probe Response paketa. Dodatno ograničenje (koje je vrijedno pomena u ovom radu) je da poslati paket mora imati minimalnu dužinu od 24 bajta, iskučujući FCS polje. Zbog navedenog, nije moguće slanje RTS paketa. Međutim, u verziji 1.3.0 ESP8266 NON-OS SDK [13][14], se prvi put pojavila metoda za slanje freedom paketa. U tom trenutku je prisustvo te funkcije bilo nezvanično, nije se pominjala u dokumentaciji, i očigledno je da data funkcija u svojoj implementaciji za tu verziju ne vrši provjeru koji je tip (tip i podtip iz Frame Control polja) paketa u pitanju. To nam omogućava da ipak šaljemo RTS pakete, zbog čega u sistem ubacujemo još jedan NodeMCU, koji će imati ulogu RTS sendera, a komadne će dobijati od centralnog NodeMCU uređaja, preko UART (Universal asynchronous receiver-transmitter) komunikacije.

Šema jedinice sistema koja je u skladu sa prethodno napisanim je predstavljena u nastavku:



Figure 8 - Šema jedinice sistema

Raspored veza (priključaka) između komponenti i glavnog NodeMCU-a određuju karakteristike izvedenih GPIO (General Purpose Input/Output) veza sa ESP8266 na njega. U sledećoj tabeli su prikazani pinovi, njihove karakteristike, namjene, moguće upotrebe i ograničenja:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Label** | **GPIO** | **Input** | **Output** | **Notes** |
| **D0** | **GPIO16** | no interrupt | no PWM or I2C support | HIGH at boot used to wake up from deep sleep |
| **D1** | **GPIO5** | OK | OK | often used as SCL (I2C) |
| **D2** | **GPIO4** | OK | OK | often used as SDA (I2C) |
| **D3** | **GPIO0** | pulled up | OK | connected to FLASH button, boot fails if pulled LOW |
| **D4** | **GPIO2** | pulled up | OK | HIGH at boot connected to on-board LED, boot fails if pulled LOW |
| **D5** | **GPIO14** | OK | OK | SPI (SCLK) |
| **D6** | **GPIO12** | OK | OK | SPI (MISO) |
| **D7** | **GPIO13** | OK | OK | SPI (MOSI) |
| **D8** | **GPIO15** | pulled to GND | OK | SPI (CS) Boot fails if pulled HIGH |
| **RX** | **GPIO3** | OK | RX pin | HIGH at boot |
| **TX** | **GPIO1** | TX pin | OK | HIGH at boot debug output at boot, boot fails if pulled LOW |
| **A0** | **ADC0** | Analog Input | X |  |

Table 1 - Ograničenja upotrebe dostupnih ESP8266 GPIO pinova (Izvor: [15])

Veze sa modula (RTC i SD kartica) su povezani na podrazumijevane pinove (one od njih koje su morali biti neophodno vezani za određene). Izuzetak su CS veza (Chip Select) za SD karticu (koja može biti vezana za bilo koji pin koji omogućava digitalni izlazni signal) I SQW (programabilni generator periodičnog digitalnog signala) veza sa RTC sata koja će se koristi za izazivanje sistemskog prekida (interrupt). SQW žica generiše signal u svakom trenutku, pa je taj signal prisutan i u toku boot procesa na ESP8266, što može izazivati probleme. Za te dvije žice postoje sledeći slobodni pinovi sa datim ograničenjima:

* D0 – slobodno za CS, nemoguće za SQW
* D3 – slobodno za CS, invertovan signal za SQW, neuspješan boot za SQW vrijednost = logička 0
* D4 – slobodno za CS, invertovan signal za SQW, neuspješan boot za SQW vrijednost = logička 0, na liniji prikopčana dioda!
* D8 – podrazumijevano za CS, nemoguć za SQW

Na osnovu prethodnih podataka, vidno je da je nemoguće da SQW signal ponekad (očekivanje = 0.5) izazove neuspješan boot na ESP8266. Bez obzira na to, izabrana je najbolja moguća opcija, a to je CS na D8, a SQW na D3.

# Implementacija

## SdFat biblioteka

Komunikacija sa memorijskom karticom se obavlja uz pomoć SdFat [16] biblioteke, verzija 1.1.4. SdFat biblioteka u okviru sebe ima definisanu File klasu koja je u koliziji sa File klasom iz Arduino Core biblioteke FS.h [10]. FS.h je korišćena u okviru ESP8266WiFi.h [10] biblioteke koja se koristi za uspostavljanje konekcije na određenu WiFi mrežu.

Navedeni problem je razriješen tako što je SdFat biblioteka modifikovana – klasa File je preimenovana na SdFatLibFile, i sve reference na tu klasu su prepravljene [17].

## Konfiguracija glavnog-upravljačkog NodeMCU-a

Softver dozvoljava oređeni nivo parametrizacije koji se tiče omogućavanja njegovog normalnog rada i prilagođavanja specifičnim potrebama u datoj situaciji. Sve te konstante se definišu u cfg.hpp zaglavlju.

#ifndef \_CONFIG\_HPP\_

#define \_CONFIG\_HPP\_

#define MILLISECOND\_PRECISION true

#define INTERNAL\_LED\_BLINK\_INTERVAL 200

#define CHANNEL\_SWITCH\_INTERVAL 200

#define FILE\_SAVE\_INTERVAL 10000

#define RTS\_COMMAND\_SWITCHING\_INTERVAL 16000

#define SD\_CS\_PIN D8

#define SQW\_INTERRUPT\_PIN D3

#define ESP8266\_ONBOARD\_LED D4

#define WIFI\_SSID "Igor"

#define WIFI\_PASSWD ""

#endif

Sistem dobija tačno vrijeme tako što se pri pokretanju sinhronizuje sa određenim NTP (Network Time Protocol) serverom. Za tu sinhronizaciju je potrebna konekcija na internet, pa se pomoću konstanti sa WIFI\_ prefiksom definiše naziv (SSID) mreže i njena lozinka (Pre-Shared Key). Sinhronizacija i kasnije praćenje vremena je moguće u preciznosti na nivou milisekunde. U zavisnosti od potreba sistema, moguće je birati praćenje podataka na nivou milisekunde, ili, u suprotnom, na nivou sekunde, pomoću konstante MILLISECOND\_PRECISION. Sinhronizacija vremena do nivoa milisekunde se obavlja zahvaljujući biblioteci ESPNtpClient [18], a kasnije precizno praćenje protoka milisekundi se može obezbijediti pomoću ranije pomenutog programabilnog generatora signala na DS3231 RTC satu, preko njegovog SQW pina. Konfiguracija omogućava podešavanje veza za SQW pin i CS pin, koji su unaprijed podešeni na vrijednosti koje su izabrane kao optimalne u poglavlju 3.2.3.

U poglavlju 3.2.3 je takođe utvrđeno da se sistem neće pri svakom pokretanju pokrenuti ispravno, pa se iz tog razloga uvodi indikator koji će pokazivati stanje sistema. Za te potrebe je izabrana dioda koja je ugrađena na ESP8266, i povezana na D4 pin na Dev Kit-u. Moguće je signal stanja sistem preusmjeriti na neki drugi slobodni pin. Kada sistem nije u zdravom stanju, moguće ga je ručno dovesti u zdravo stanje pomoću pritiska na RST (Reset) taster (efektivno izaziva ponovno pokretanje sistema) koji je prisutan na NodeMCU ploči.

Ostale konfiguracione konstante se odnose na stanje sistema u kome je on već prešao u aktivni rad i prikupljanje podataka. Tu se definišu vremenski intervali za:

* Indikator zdravlja signala – brzina promjene stanja indikatora – lampice
* Skok na drugi kanal – zadržavanje na pojedinačnom kanalu u 2.4GHz spektru
* Čuvanje podataka na memorijskoj kartici – aktivnost oduzima mnogo sistemnskog vremena pa se radi periodično
* Promjenu ciljnog uređaja na RTS odašiljaču – interval izdavanja nove komande sekundarnom NodeMCU-u

## NTP sinhronizacija vremena

ESPNtpClient biblioteka obavlja sinhronizaciju vremena asinhronim putem. To nije željeno ponašanje u našem slučaju, jer prikupljanje podataka u našem sistemu može da počne tek kada je sinhronizacija vremena uspješno urađena. Zbog toga je bilo potrebno napraviti mehanizam koji će omogućiti čekanje na sinhronizaciju.

Kreirana je enumeracija sa osnovnim stanjima NTP klijekta koji su relevantni za naše potrebe:

typedef enum {

    TIME\_SYNCED,

    TIME\_SYNCING,

    ERROR,

    IDLE

} NTPClient\_State\_t;

U funkciji pokretanja NTP klijenta, prijavimo metodu koja će, između ostalog, pozivati metodu za procesiranje svakog događaja u ESPNtpClient biblioteci.

void startNTPClient(uint16\_t milli\_ntp\_timeout, int sync\_period,

                const char \* TZ, const char \* ntp\_server\_nullable) {

    NTP.onNTPSyncEvent ([] (NTPEvent\_t event) {

        ntpEvent = event; // hold last triggered event

        processSyncEvent(event);

    });

    ...

Metoda za procesiranje, na osnovu događaja u biblioteci definiše trenutno stanje iz prethodno predstavljenje enumeracije.

NTPClient\_State\_t ntpClientState = IDLE;

void processSyncEvent(NTPEvent\_t ntpEvent) {

    switch (ntpEvent.event) {

        case timeSyncd:

        case syncNotNeeded:

            ntpClientState = TIME\_SYNCED;

            break;

        case requestSent:

        case partlySync:

            ntpClientState = TIME\_SYNCING;

            break;

        default:

            ntpClientState = ERROR;

            break;

    }

    if(additionalEventHandler != nullptr) additionalEventHandler(ntpEvent);

}

Sada, na osnovu stanja zabilježenog u ntpClientState promjenljivoj možemo implementirati funkciju čekanja na uspješnu sinhronizaciju vremena sa NTP serverom.

#define SYNCING\_PRINT\_INTERVAL 1000

void waitNTPClientSync() {

    unsigned long time = millis();

    if(Serial) Serial.println("NTP Client Sync in progress.");

    while(getNTPClientState() != TIME\_SYNCED) {

        if(Serial && millis() - time > SYNCING\_PRINT\_INTERVAL) {

            Serial.printf(".");

            time = millis();

        }

    }

}

## Rad sa RTC satom

DS3231 ima mogućnost praćenja vremena u preciznosti sekunde, bez direktne podrške za milisekunde. Međutim, moguće postaviti generisanje pobudnog signala na frekvenciji 1024Hz (1024 pobude u toku jedne sekunde). Taj izlaz, SQW, je moguće koristiti kao izazivač sistemskih prekida. Obrađivač tog sistemskog prekida će pratiti protok vremena na osnovu prekida i tako sam čin preciznog praćenja vremena u preciznosti milisekunde prebacuje na glavni NodeMCU.

### Inicijalizacija RTC sata sa tačnim vremenom (u preciznosti sekunde)

ESPNtpClient biblioteka ne pruža dovoljnu fleksibilnost u radu sa dobijenim vremenom. Zbog toga se iz nje vremenski podaci uvoze u biblioteku TimeLib [19]. Potrebno je sačekati uspješnu sinhronizaciju TimeLib biblioteke sa vremenom uzetim iz NTP klijenta.

    // setting internal timelib helper

    setSyncProvider(NTPUnixTics);

    while (timeStatus() != timeSet);

Nakon toga se prelazi na postavljanje vremena na RTC satu, gdje se koriste funkcije i makroi iz TimeLib biblioteke.

void setDS3231Time() {

    Clock.setYear(tmYearToY2k(CalendarYrToTm(year())));

    Clock.setMonth(month());

    Clock.setDate(day());

    Clock.setDoW(weekday());

    Clock.setClockMode(false);  // set to 24h

    //setClockMode(true); // set to 12h

    Clock.setHour(hour());

    Clock.setMinute(minute());

    Clock.setSecond(second());

    ...

### NodeMCU praćenje vremena u preciznosti milisekunde na osnovu sistemskih prekida

U trenutku kada utvrdimo tačno vrijeme, imamo broj koji predstavlja broj milisekundi koje su protekle od početka epohe pa do bas tog trenutka. Početak epohe je definisan kao 01.01.1970. 00:00:00 0ms. Ako tog trenutka kada smo sinhronizovali vrijeme počnemo da pratimo broj milisekundi koje protiču kao rezultat pobudnog signala na svaku milisekundu, zbirom ta dva broja uvijek dobijamo tačno vrijeme izraženo u milisekundama od početka epohe.

Kako je frekvencija pobudnog (interrupt) signala 1024Hz, to znači da ipak nemamo pobudu tačno svake milisekunde, već svake 0.9765625ms. Taj problem rješavamo programski, tako što pri prijemu interrupt-a, funkcija koja vrši obradu sistemskog prekida (ISR, Interrupt Service Routine), odlučuje da li će dati interrupt računati kao proticanje jedne milisekunde, ili će je preskočiti. Odluka se donosi tako što se razmatra koja vrijednost će biti bliža realnoj vrijednosti, ako se doda jedna milisekunda, ili ako se preskoči.

uint16\_t int\_freq = 1024;

double milli\_period = double(int\_freq) / 1000;

uint16\_t floor\_freq = floor(milli\_period);

uint16\_t ceil\_freq = ceil(milli\_period);

volatile unsigned long long counter = 0;

volatile unsigned long long millisecond\_isr\_counter = 0;

volatile unsigned long long milli\_isr\_total = 0;

void IRAM\_ATTR isr\_func() {

    ++counter;

    ++millisecond\_isr\_counter;

    if (millisecond\_isr\_counter == ceil\_freq ||

          (millisecond\_isr\_counter == floor\_freq &&

               (double(counter) - milli\_period \* (milli\_isr\_total + 1))                      >= 0.0)) {

        millisecond\_isr\_counter = 0;

        ++milli\_isr\_total;

        ++espRTCTime;

    }

}

ISR funkcija mora biti označena sa IRAM\_ATTR, što znači da će ona biti smještena u RAM, a ne u flash memoriju ESP8266.

## Inicijalizacija sistema

U inicijalizaciji sistema se obavljaju sve pripreme dovođenje sistema u aktivno stanje prikupljanja podataka. To obuhvata:

* Inicijalizaciju veza – pripremanje pinova za komunikaciju
* Otvaranje serijske komunikacije (sa drugim NodeMCU-om)
* Povezivanje na WiFi mrežu
* Sinhronizacija vremena sa NTP serverom
* Postavljanje vremena na RTC sat i početak praćenja protoka vremena
* Povezivanje sa memorijskom karticom i otvaranje fajla za logovanje na njoj
* Registracija metode za prikupljanje podataka

void setup() {

    // setting up pin modes

    pinMode(SD\_CS\_PIN, OUTPUT);

    pinMode(SQW\_INTERRUPT\_PIN, INPUT\_PULLUP);

    pinMode(ESP8266\_ONBOARD\_LED, OUTPUT);

    digitalWrite(ESP8266\_ONBOARD\_LED, LOW);

    digitalWrite(SD\_CS\_PIN, LOW);

    // setting up serial comms

    Serial.begin(115200); while (!Serial);

    // connecting to wifi

    clientSetupWiFi();

    clientConnectWiFi(WIFI\_SSID, WIFI\_PASSWD);

    // setup I2C interface

    Wire.begin();

    //Clock.enable32kHz(true);

    if(!Clock.oscillatorCheck()) clockPresent = false;

    // interfacing with RTC oscillators

    if(clockPresent) {

        Clock.enable32kHz(false);

        Clock.enableOscillator(true, false, 1);

    }

    // synchronising time

    registerNTPEventHandler(addEHSerialPrintEvent);

    startNTPClient();

    waitNTPClientSync();

    // setting internal timelib helper

    setSyncProvider(NTPUnixTics);

    while (timeStatus() != timeSet);

    // setting RTC

    if(clockPresent) {

        attachInterrupt(digitalPinToInterrupt(SQW\_INTERRUPT\_PIN),

                            isr\_func, RISING);

        setDS3231Time();

    }

    // stopping ntp client's resync jobs

    stopNTPClient(false);

    // disconnecting from wifi

    clientDisconnectWiFi();

    // SPI interface to SD card

    sdCardPresent = SD.begin(SD\_CS\_PIN, SPI\_HALF\_SPEED);

    // open file and prepare for writing

    sprintf(filename, "%llu.txt", getPrefferedTime());

    openFile();

    // start logging

    logTime();

    rts\_switch\_command();

    sniffer\_init(channel, probe\_request\_cb\_handler);

}

Na početku se pinovi CS, SQW i pin vezan za diodu na ESP8266 ploči, pripremaju za upotrebu u skladu sa svojom namjenom. U poglavlju 3.2.3 je pokazano da pinovi D3 i D4 (SQW i onboard LED) vezani za pull up otpornik, što znači da se na njih primjenjuje invertovana logika. Zbog toga se upotreba SQW pina označava sa INPUT\_PULLUP. U poglavlju 4.2 je opisana upotreba LED diode vezane na pin D4. U toku inicijalizacione procedure ta lampica treba da bude upaljena, a s obzirom na invertovanu logiku i na tom pinu (poglavlje 3.2.3), postavljamo na nju vrijednost logičke nule (LOW).

Priprema RTC sata za upotrebu podrazumijeva gašenje 32kHz oscilatora(jer nam on nije potreban), i paljenje oscilatora sa oznakom 1, što je 1.024kHz oscilator koji je nama potreban. On postaje odmah dostupan na SQW pin-u.Uz postavljanje vremena na RTC satu, registrujemo interrupt za SQW pin tako što na njega povezujemo nasu ISR metodu (isr\_func) i definišemo da se okidanje registruje na rastućoj ivici square wave signala.

Bitno je zaustaviti NTP klijent i diskonektovati se sa WiFi mreže, jer samo u tom slučaju se smije registrovati metoda za obradu pristiglih 802.11 paketa. Zaustavljanje NTP klijenta je neophodno jer se ne smije dozvoliti pokušaj ponovni osvježavanja vremena – ponovni pokušaj pristupa internetu u toku slušanja paketa prouzrokuje fatalnu grešku.

## Aktivni režim rada glavnog NodeMCU uređaja

Slušač 802.11 paketa je prijavljen u inicijalizacionoj fazi i on funkcioniše u pozadini preko sistemskih prekida. O njemu će biti riječi tek u narednom poglavlju.

Glavna linija izvršavanja sistema se bavi periodičnim obrađivanjem poslova koji su već pomenuti u okviru poglavlja 4.1 gdje je bila riječ o konfiguraciji. Tu su bili konfigurisani periodi za periodične poslove, a to su: indikacija stanja sistema, skokovi na druge kanale, čuvanje podataka na memorijskoj kartici i izmjena komande RTS odašiljaču.

void loop() {

    probe\_request\_final\_handler();

    loop\_led\_blinking();

    channel\_hopping();

    sd\_file\_saving();

    rts\_switching();

    yield();

}

Sve četiri te funkcije su implementirane na sličan način, pomoću generičkog tajmera koji se izvršava željenu metodu samo ako je od njenog prethodnog izvršavanja prošao definisani vremenski interval. Implementacija generičkog tajmera se može vidjeti u sledećem listingu.

void generic\_timer(unsigned long \* last, unsigned long current,

                unsigned long desired\_interval, void (\*handler)()) {

    if((current - \*last) < desired\_interval) return;

    \*last = current;

    handler();

}

Upotreba tajmera se može demonstrirati na primjeru LED lampice koja je indikator zdravog stanja sistema. U toku zdravog stanja sistema ugrađena LED dioda treba da miga, što znači da se implementacija toga svodi na periodično invertovanje njenog stanja.

void led\_blinker() {

    static uint8\_t next\_state = HIGH;

    digitalWrite(ESP8266\_ONBOARD\_LED, next\_state);

    next\_state = next\_state == LOW ? HIGH : LOW;

}

void loop\_led\_blinking() {

    static unsigned long last = millis();

    generic\_timer(&last, millis(), INTERNAL\_LED\_BLINK\_INTERVAL, led\_blinker);

}

Generičkom tajmeru se predaje adresa na promjenljivu statičkog vijeka trajanja koja nosi vrijednost posljednjeg izvršavanja ciljanog posla, sledeći parametar je trenutno vrijeme, treći vremenski interval – period izvršavanja, i četvrti je pokazivač na void metodu bez parametara koja obavlja taj posao.

## Prikupljanje 802.11 paketa i njihova obrada

Obrada priimljenih paketa u promiscuous režimu se obavlja tako što se registruje funkcija koja će biti obrađivač paketa. U [11] je naglašeno da trajanje ovih funkcija ne treba biti dugo, jer to izaziva propuštanje paketa koji bi mogli biti primljeni u toku obrađivanja nekog paketa. Čuvanje izdvojenih podataka je dugotrajna operacija, tako da je svakako potrebno implementirati baferovanje podataka prije njihovog čuvanja na memorijsku karticu. Postoji opcija i baferovanja samih paketa, prije njihove obrade, ali to nije uzeto u obzir u implementaciji, jer bi baferovanje bafera paketa bilo izrazito memorijski skupocjeno, a korist je vjerovatno ograničena.

Prikupljeni podaci se čuvaju u strukturi koja sadrži sve podatke iz Probe Request paketa koji mogu biti od značaja pri praćenju i identifikaciji, a uz to sadrži i polje u koje se upisuje vrijeme prijema paketa.

struct probe\_request {

  uint64\_t millitime;

  RxControl radiotap\_header;

  uint8\_t source\_address[6];

  uint8\_t transmission\_address[6];

  uint8\_t destination\_address[6];

  uint8\_t receiver\_address[6];

  uint8\_t bssid\_address[6];

  struct frame\_sequence\_control\_t seq\_ctrl;

};

Po prijemu paketa, poziva se tzv. callback metoda koja započinje obradu paketa.

void sniffer\_handler\_func(uint8\_t \*buff, uint16\_t len) {

  // Filter unreliable packets (HT40, LDPC)

  if(len <= sizeof(struct RxControl)) return;

  const struct promiscuous\_mode\_pkt \* ppkt = (struct promiscuous\_mode\_pkt \*) buff;

  struct probe\_request pr;

  bool success = probe\_request\_filter(ppkt, len, &pr);

  if(success && sniffer\_pr\_cb != nullptr) {

    pr.radiotap\_header = ppkt->rx\_ctrl;

    sniffer\_pr\_cb(&pr);

  }

}

Bitno je napomenuti da se callback metoda poziva za sve primljene tipove 802.11 paketa, a čak i za nevalidne, pa se prvo mora pristupiti filtriranju paketa. Odmah počinje da se konstruiše probe\_request struktura i da se popunjava podacima. Ako se ispostavi da je paket validan Probe Request, tada će se pozvati metoda (u prethodnom listingu označena kao sniffer\_pr\_cb) koja je u inicijalizacionoj fazi sistema prijavljena kao funkcija koja će dovršavati izdvajanje podataka. Njene obaveze se svode na bilježenje tačnog vremena i snimanje podataka. Kao što je već navedeno, snimanje podataka se ne vrši direktno, već se baferuje (pomoćnom standardnom generičkom strukturom deque).

std::deque<struct probe\_request> prbuff;

void probe\_request\_cb\_handler(struct probe\_request \* ppr) {

    ppr->millitime = getPrefferedTime();

    prbuff.push\_back(\*ppr);

}

Od tog trenutka se nastavlja prikupljanje paketa, a krajnje čuvanje podataka se prepušta glavnoj liniji izvršavanja. U poglavlju 4.6 se u glavnoj izvršnoj funkciji loop vidi da s pri svakom pozivu iste, poziva i probe\_request\_final\_handler. On vrši finalno čuvanje podataka i prazni obrađene Probe Request-ove iz deque-a.

void probe\_request\_final\_handler() {

    if(prbuff.empty()) return;

    struct probe\_request pr = prbuff.front();

    char temp[256] = {0};

    char recv[20] = {0};

    char dest[20] = {0};

    char trans[20] = {0};

    char src[20] = {0};

    char bssid[20] = {0};

    macprint(pr.receiver\_address, recv);

    macprint(pr.destination\_address, dest);

    macprint(pr.transmission\_address, trans);

    macprint(pr.source\_address, src);

    macprint(pr.bssid\_address, bssid);

    sprintf(temp, "PR timestamp %llu channel %d

                |recv->%s|dest->%s|trans->%s|src->%s|bssid->%s|",

                pr.millitime, channel, recv, dest, trans, src, bssid);

    prbuff.pop\_front();

    logln(temp);

}

Samo ako ima Probe Request-ova na raspolaganju se pokreće čuvanje jednog (najstarijeg) od njih. Nema potrebe za čuvanjem više njih, jer se ova metoda u programskoj petlji poziva u svakoj njenoj iteraciji, a osim toga, jedinično čuvanje je bolje jer omogućuje preciznije izvršavanje periodičnih poslova.

## Slanje komandi RTS odašiljaču

Podaci se šalju periodično RTS odašiljaču. Ideja je da se u svakom period izda komanda koja će da se tiče jednog uređaja. U svakoj sledećoj periodi se ciklično prolazi kroz listu ciljanih uređaja.

{'D',0x40,0x4e,0x36,0x3a,0x45,0x1b, 'S',0x00,0x11,0x22,0x33,0x44,0x55}

Komanda se sastoji iz dva dijela. Prvi počinje sa znakom „D” i ukazuje na to da slijedi MAC adresa destinacije RTS paketa. Zatim, drugi dio započinje znakom „S” koji ukazuje da u nastavku slijedi MAC adresa porijekla RTS paketa koji treba poslati.

## RTS odašiljač

U poglavlju 3.2.3 je opisano kako je ipak moguće slati 802.11 freedom RTS pakete u određenoj implementaciji koja ne vrši provjeru tipa paketa koji se šalje. Ipak, prisutna je provjera da paket mora imati minimalnu dužinu od 24 bajta, što je znatno više od običnog RTS paketa, kao što je prikazano u 2.1.4 gdje je ukupna dužina paketa 16 bajta.

// RTS Packet buffer

uint8\_t packet[128] = { 0xb6, 0xb0, 0x00, 0x00,

                        0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

                        0xee, 0xee, 0xee, 0xee, 0xee, 0xee,

                        0xaa, 0xaa, 0xaa, 0xaa,

                        0xaa, 0xaa, 0xaa, 0xaa,

                        0xaa, 0x00};

Tu prepreku prevazilazimo tako što u paket dodajemo beznačajne podatke u obliku 0xaa bajta, kao što se može vidjeti iznad, u četvrtom i petom redu paketa. Prva dva bajta paketa predstavljaju Frame Control polje, u kome je definisan tip - RTS paket. Sledeća dva bajta predstavljaju Duration polje, koje ima vrijednost 0. Drugi red je prostor za MAC adresu primaoca, a treći je prostor za MAC adresu primaoca.

RTS odašiljač sluša serijsku komunikaciju na kojoj očekuje komande od glavnog NodeMCU-a i kada primi podatke koji su u format koji je prikazan u poglavlju 4.8, dakle počinje sa znakom „D”, on tumači te podatke i upisuje ih na odgovarajuća polja u paketu. To znači da prvu adresu kopira na lokacije počev od bajta 4 (drugi red), a drugu adresu kopira na adrese počev od reda 3 (treći red).

#define PACKET\_DEST\_IND 4

#define BUFFLEN 6

int iter = 0;

void ICACHE\_FLASH\_ATTR uart\_rx\_task(os\_event\_t \*events) {

    if (events->sig == 0) {

        // Sig 0 is a normal receive.

        // Get how many bytes have been received.

        uint8\_t rx\_len = (READ\_PERI\_REG(UART\_STATUS(UART0)) >>

                                UART\_RXFIFO\_CNT\_S) & UART\_RXFIFO\_CNT;

        uint8\_t rx\_char;

        uint8\_t ii;

        uint8\_t srcnow = 0;

        for (ii=0; ii < rx\_len; ii++) {

            rx\_char = READ\_PERI\_REG(UART\_FIFO(UART0)) & 0xFF;

            if(ii==0 && rx\_char!='D') break;

            else if(ii == 14) break;

            else if(ii == 7 && rx\_char=='S') {

                srcnow = 1;

                continue;

            }

            else if(ii==0 || ii==7) continue;

            iter = ii-1-srcnow;

            packet[PACKET\_DEST\_IND + iter] = rx\_char;

            os\_printf("iter %d SERIAL %02x = %c and datalen %d \n",

                        iter, rx\_char, rx\_char, rx\_len);

        }

        // Clear the interrupt condition flags and

        // re-enable the receive interrupt.

        WRITE\_PERI\_REG(UART\_INT\_CLR(UART0), UART\_RXFIFO\_FULL\_INT\_CLR | UART\_RXFIFO\_TOUT\_INT\_CLR);

        uart\_rx\_intr\_enable(UART0);

    }

}

U verzijama SDK u kojima je funkcija za slanje freedom paketa zvanično prisutna, postoji i mehanizam koji potvrđuje da je paket poslat. To je realizovano tako što se pomoću funkcije

int wifi\_register\_send\_pkt\_freedom\_cb(freedom\_outside\_cb\_t cb);

gdje je

typedef void (\*freedom\_outside\_cb\_t)(uint8 status);

registruje callback funkcija koja će biti pozvana tek kada je 802.11 freedom paket uspješno poslat. To je bitno, jer u slučaju kada se pokuša slanje paketa, a prethodni paket još nije poslat, posljedica je da nijedan paket neće biti poslat. Nažalost, u SDK 1.3.0, ne postoji ova funkcionalnost, tako da se između slanja mora praviti pauza da bi se osiguralo stvarno isporučivanje paketa. U ovom radu nije istraživano koliki je minimalni vremenski interval, već je izabrana vrijednost od 1250ms koja se pokazala kao više nego dovoljna da svaki paket bude isporučen.

void sender(void \*arg) {

    const char \* pmac = packet + PACKET\_DEST\_IND;

    const char \* pmac1 = packet + PACKET\_DEST\_IND + 6;

    os\_printf("Send status: %d on channel %d

                dest %02x:%02x:%02x:%02x:%02x:%02x and

                src %02x:%02x:%02x:%02x:%02x:%02x\n",

                wifi\_send\_pkt\_freedom(packet, 25, 0),

                channel, pmac[0], pmac[1], pmac[2],

                pmac[3], pmac[4], pmac[5], pmac1[0],

                pmac1[1], pmac1[2], pmac1[3],

                pmac1[4], pmac1[5]);

    next\_channel();

    light\_switch();

}

Funkcija slanja paketa se poziva periodično, sa već naglašenom pauzom, i nakon slanja paketa se odmah prelazi na sledeći kanal na kome će ponovo biti poslat isti paket, ili, ako je u međuvremenu došla nova komanda, novi paket. Zbog ovoga, bitno je da na glavnom NodeMCU interval slanja komandi RTS odašiljaču bude minimalno jednak broju kanala (u ovom radu 13) \* pauzi između slanja RTS paketa (ovdje, 1250).

# Rezultati i diskusija

# Budući rad

# Zaključak

# Literatura

1. Wi-Fi Alliance - <https://www.wi-fi.org/>
2. IEEE 802 LAN/MAN Standards Committee - <https://www.ieee802.org/>
3. Matthew S. Gast, „802.11 Wireless Networks: The Definitive Guide, 2nd Edition”, O’Reilly Media, Inc. - <https://www.oreilly.com/library/view/80211-wireless-networks/0596100523/>
4. IEEE 802.11-05/0710r 0, “WDS” Clarifications - <http://www.ieee802.org/1/files/public/802_architecture_group/802-11/4-address-format.doc>
5. Espressif Systems - <https://www.espressif.com/>
6. NodeMCU - <https://www.nodemcu.com/index_en.html>
7. DS3231S(N) Data Sheet - <https://datasheets.maximintegrated.com/en/ds/DS3231.pdf>
8. DS3231M Data Sheet - <https://datasheets.maximintegrated.com/en/ds/DS3231M.pdf>
9. ESP8266 NON-OS SDK - <https://github.com/espressif/ESP8266_NONOS_SDK>
10. ESP8266 core for Arduino - <https://github.com/esp8266/Arduino>
11. ESP8266 Technical Reference - <https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf>
12. ESP8266 Non-OS SDK API Reference - <https://www.espressif.com/sites/default/files/documentation/2c-esp8266_non_os_sdk_api_reference_en.pdf>
13. ESP8266\_NONOS\_SDK\_v1.3.0\_15\_08\_08 Release Note - [https://bbs.espressif.com/viewtopic.php?p=3092#p3092](https://bbs.espressif.com/viewtopic.php?p=3092%23p3092)
14. ESP-OPEN-SDK - <https://github.com/pfalcon/esp-open-sdk>
15. ESP8266 Pinout Reference - <https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/>
16. SdFat, Greiman - <https://github.com/greiman/SdFat>
17. Modifikovana verzija SdFat 1.4.1 biblioteke - <https://github.com/FmasterofU/SdFat>
18. ESPNtpClient, Germán Martín - <https://github.com/gmag11/ESPNtpClient>
19. TimeLib, Paul Stoffregen - <https://github.com/PaulStoffregen/Time>

Biografija

Igor Šikuljak je rođen u 22.07.1998. godine u Sokocu, Bosna i Hercegovina. Osnovnu školu „Vuk Karadžić” u Vlasenici je završio 2013. godine kao učenik generacije. Potom upisuje gimaziju u SŠC „Milorad Vlačić” u Vlasenici koju završava 2017. godine, takođe kao učenik generacije. Uporedo sa srednjom školom, polaznik je na seminarima računarstva u IS Petnica u Valjevu. Nakon završene srednje škole, upisuje Fakultet tehničkih nauka u Novom Sadu, smijer Računarstvo i automatika, a postaje i asistent na programu računarstva u IS Petnica, gdje je i dalje angažovan.

Položio je sve ispite predviđene planom i programom svog smijera, na modulu Primenjene računarske nauke i informatika i studijskoj grupi Internet i elektronsko poslovanje.

|  |  |  |  |
| --- | --- | --- | --- |
|  | UNIVERZITET U NOVOM SADU ⚫ **FAKULTET TEHNIČKIH NAUKA**  21000 NOVI SAD, Trg Dositeja Obradovića 6 | | |
| KLJUČNA DOKUMENTACIJSKA INFORMACIJA | | |
| Redni broj, **RBR**: | |  | |
| Identifikacioni broj, **IBR**: | |  | |
| Tip dokumentacije, **TD**: | | monografska publikacija | |
| Tip zapisa, **TZ**: | | tekstualni štampani dokument | |
| Vrsta rada, **VR**: | | diplomski rad | |
| Autor, **AU**: | | Igor Šikuljak | |
| Mentor, **MN**: | | prof. dr Milan Vidaković, FTN Novi Sad | |
| Naslov rada, **NR**: | | Sistem za praćenje kretanja mobilnih uređaja sa omogućenom upotrebom WiFi tehnologije | |
| Jezik publikacije, **JP**: | | srpski | |
| Jezik izvoda, **JI**: | | srpski / engleski | |
| Zemlja publikovanja, **ZP**: | | Srbija | |
| Uže geografsko područje, **UGP**: | | Vojvodina | |
| Godina, **GO**: | | 2021 | |
| Izdavač, **IZ**: | | autorski reprint | |
| Mesto i adresa, **MA**: | | Novi Sad, Fakultet tehničkih nauka, Trg Dositeja Obradovića 6 | |
| Fizički opis rada, **FO**: | | br. poglavlja: 8 / stranica: 64 / citata: 42 / slika: 34 / listinga: 17 / priloga: 0 / tabela: 1 / grafikona: 0 | |
| Naučna oblast, **NO**: | | Računarske nauke i informatika | |
| Naučna disciplina, **ND**: | | Računarske mreže, bezbjednost sistema, embedded sistemi | |
| Predmetna odrednica /  ključne reči, **PO**: | | Praćenje mobilnih uređaja, MAC adrese, 802.11 Probe Request paketi, 802.11 RTS/CTS paketi, Espressif ESP8266 | |
| **UDK** | |  | |
| Čuva se, **ČU**: | | Biblioteka Fakulteta tehničkih nauka, Trg Dositeja Obradovića 6, Novi Sad | |
| Važna napomena, **VN**: | |  | |
| Izvod, **IZ**: | | OVDE IDE APSRAKT U NEKOLIKO REDOVA | |
| Datum prihvatanja teme, **DP**: | |  | |
| Datum odbrane, **DO**: | |  | |
| Članovi komisije, **KO**: | |  | |
|  | Predsednik: | dr Željko Vuković, FTN Novi Sad | |
|  | Član: | dr Miroslav Zarić, vanredni profesor, FTN Novi Sad | Potpis mentora: |
|  | Mentor: | prof. dr Milan Vidaković, FTN Novi Sad |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | UNIVERSITY OF NOVI SAD ⚫ **FACULTY OF TECHNICAL SCIENCES**  21000 NOVI SAD, Trg Dositeja Obradovića 6 | | |
| KEY WORDS DOCUMENTATION | | |
| Accession number, **ANO**: | | |  | |
| Identification number, **INO**: | | |  | |
| Document type, **DT**: | | | monographic publication | |
| Type of record, **TR**: | | | textual material | |
| Contents code, **CC**: | | | BSc thesis | |
| Author, **AU**: | | | Igor Šikuljak | |
| Mentor, **MN**: | | | Milan Vidaković, PhD assoc. prof., FTN Novi Sad | |
| Title, **TI**: | | | A system for spatial movement tracking of mobile devices with enabled WiFi technology | |
| Language of text, **LT**: | | | Serbian | |
| Language of abstract, **LA**: | | | serbian / english | |
| Country of publication, **CP**: | | | Serbia | |
| Locality of publication, **LP**: | | | Vojvodina | |
| Publication year, **PY**: | | | 2021 | |
| Publisher, **PB**: | | | author’s reprint | |
| Publication place, **PP**: | | | Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6 | |
| Physical description, **PD**: | | | no. of chapters: 8 / pages: 64 / citations: 42 / images: 34 / listings: 17 / appendices: 0 / tables: 1 / graphs: 0 | |
| Scientific field, **SF**: | | | Computer science | |
| Scientific discipline, **SD**: | | | Computer networks, security, embedded systems | |
| Subject / Keywords, **S/KW**: | | | Mobile device tracking, MAC addresses, 802.11 Probe Request packets, 802.11 RTS/CTS packets, Espressif ESP8266 | |
| **UDC** | | |  | |
| Holding data, **HD**: | | | Library of the Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad | |
| Note, **N**: | | |  | |
| Abstract, **AB**: | | | ABSTRACT HERE | |
| Accepted by sci. board on, **ASB**: | | |  | |
| Defended on, **DE**: | | |  | |
| Defense board, **DB**: | | |  | |
|  | President: | | Dr Željko Vuković, PhD, FTN Novi Sad | |
|  | Member: | | Miroslav Zarić, PhD, assoc. prof., FTN Novi Sad | Mentor’s signature |
|  | Mentor: | | Milan Vidaković, PhD, prof., FTN Novi Sad |  |