SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD

Algoritmi sinkronizacije u raspodijeljenim sustavima s velikim brojem uređaja

Robert Medvedec

Mentor: izv. prof. dr. sc. Igor Čavrak

Zagreb, lipanj 2023.

Sažetak

Sinkronizacija prijenosnih uređaja korištenjem bežičnih tehnologija jedan je od još uvijek aktivnih problema u *IoT* svijetu. S obzirom na ograničenja količine energije i razine performansi mikrokontrolera, potrebno je implementirati različite načine kako bi se postigla optimalna razina sinkronizacije sa što manje resursa.

U radu su detaljno proučeni i implementirani razni sinkronizacijski algoritmi bazirani na BLE komunikaciji radi sinkronizacije više ESP32 uređaja u stvarnom vremenu.

Zbog limita same tehnologije zaključak je da je moguće postići konzistentnu sinkronizaciju u razini nekoliko milisekundi uz minimalnu potrošnju energije.

Sadržaj

[1. Uvod 8](#_Toc137995921)

[2. Osnovno o mikrokontrolerima 9](#_Toc137995922)

[2.1. ESP32 9](#_Toc137995923)

[2.2. FreeRTOS 12](#_Toc137995924)

[2.3. Bežićne tehnologije povezivanja mikrokontrolera 13](#_Toc137995925)

[2.3.1. Bluetooth 13](#_Toc137995926)

[2.3.2. Wi-Fi 19](#_Toc137995927)

[2.3.3. ESP-Now 19](#_Toc137995928)

[2.3.4. Ostali protokoli i standardi 21](#_Toc137995929)

[2.3.5. Povezivanje uređaja iz ESP-32 serije 22](#_Toc137995930)

[3. Sinkronizacija 27](#_Toc137995931)

[3.1. Direktna komunikacija 27](#_Toc137995932)

[3.1.1. Sinkronizacija pomoću signala 27](#_Toc137995933)

[3.1.2. Sinkronizacija unutarnjeg brojača 30](#_Toc137995934)

[3.2. Sinkronizacija sata 33](#_Toc137995935)

[3.2.1. Network Time Protocol (NTP) 33](#_Toc137995936)

[3.2.2. Simplified Network Time Protocol (SNTP) 36](#_Toc137995937)

[3.2.3. Precision Time Protocol (PTP) 37](#_Toc137995938)

[3.3. Problemi kod sinkronizacije 38](#_Toc137995939)

[4. Implementacija, spajanje i sinkroniziranje mikrokontrolera 40](#_Toc137995940)

[4.1. Protokoli i sučelja 40](#_Toc137995941)

[4.1.1. ESP-IDF 40](#_Toc137995942)

[4.1.2. Bluedroid 41](#_Toc137995943)

[4.1.3. NimBLE 41](#_Toc137995944)

[4.1.4. Mjerenje vremena s ESP32 42](#_Toc137995945)

[4.1.5. Prekidne rutine 43](#_Toc137995946)

[4.2. Sinkronizacija uređaja 44](#_Toc137995947)

[4.2.1. Korišteno sučelje i postavke okoline 44](#_Toc137995948)

[4.2.2. Sinkronizacija bez odmaka 48](#_Toc137995949)

[4.2.3. Sinkronizacija s fiksnim odmakom 49](#_Toc137995950)

[4.2.4. Sinkronizacija s dinamičnim odmakom 51](#_Toc137995951)

[4.2.5. Sinkronizacija pomoću brojača 52](#_Toc137995952)

[4.3. Mjerenje odmaka 55](#_Toc137995953)

[4.3.1. Računski 55](#_Toc137995954)

[4.3.2. Kamera 58](#_Toc137995955)

[5. Daljnji rad i razmatranje 63](#_Toc137995956)

[6. Zaključak 65](#_Toc137995957)

[7. Literatura 66](#_Toc137995958)

[DODATAK A: Programska potpora 70](#_Toc137995959)

[ESP-IDF 70](#_Toc137995960)

[Konstrukcija koda 71](#_Toc137995961)

Popis oznaka i kratica

LED *light-emitting diode*

BLE *Bluetooth Low-Energy*

BR/EDR *Basic rate/enhanced data rate*

HCI *Host controller interface*

GATT *Generic Attribute Profile*

GAP *General Access Profile*

UUID *Universally Unique IDentifier*

IoT *Internet of Things*

GPIO *General Purpose Input/Output*

Popis tablica

[Tablica 2.1 Usporedba uređaja iz ESP32 obitelji [2] 11](#_Toc137996986)

[Tablica 2.2 Razlike između BLE i Bluetooth Classic protokola [5] 18](#_Toc137996987)

[Tablica 4.1 Rezultati preciznosti sinkronizacije pomoću video mjerenja 58](#_Toc137996988)

Popis slika

[Slika 2.1 Preslikavanje BLE stoga na OSI referentni model 13](#_Toc138002425)

[Slika 2.2 BLE stog 14](#_Toc138002426)

[Slika 2.3 BLE ATT atributi 15](#_Toc138002427)

[Slika 2.4 ESP-Now 19](#_Toc138002428)

[Slika 2.5 ESP32 Wi-Fi and Bluetooth coexsistence 22](#_Toc138002429)

[Slika 3.1 UML sekvencijski dijagram - Komunikacija bez sinkronizacije 27](#_Toc138002430)

[Slika 3.2 UML sekvencijski dijagram - Sinkronizacija s fiksnim odmakom 28](#_Toc138002431)

[Slika 3.3 UML sekvencijski dijagram - Sinkronizacija s dinamičnim odmakom 29](#_Toc138002432)

[Slika 3.4 UML sekvencijski dijagram - Periodička sinkronizacija na zahtjev 30](#_Toc138002433)

[Slika 3.5 UML sekvencijski dijagram - Periodička resinkronizacija 31](#_Toc138002434)

[Slika 3.6 Mreža uređaja i satova u NTP protokolu 33](#_Toc138002435)

[Slika 4.1 Postavke *Bluedroid* stoga kroz ESP-IDF sučelje 40](#_Toc138002436)

[Slika 4.2 Osnovna komunikacija ESP32 servera i klijenta 44](#_Toc138002437)

[Slika 4.3 Vremenski dijagram slanja poruke 46](#_Toc138002438)

[Slika 4.4 Skica odmaka slanja poruke 47](#_Toc138002439)

[Slika 4.5 Sinkronizacija bez odmaka 48](#_Toc138002440)

[Slika 4.6 Sinkronizacija s fiksnim odmakom 49](#_Toc138002441)

[Slika 4.7 Sinkronizacija s izračunatim dinamičnim odmakom 50](#_Toc138002442)

[Slika 4.8 Sinkronizacija pomoću brojača 52](#_Toc138002443)

[Slika 4.9 Resinkronizacija pomoću brojača 53](#_Toc138002444)

[Slika 4.10 Kvantilni graf odmaka bez sinkronizacije 56](#_Toc138002445)

[Slika 4.11 Kvantilni graf odmaka sinkronizacije s fiksnom vrijednošću 56](#_Toc138002446)

[Slika 4.12 Kvantilni graf odmaka sinkronizacije s dinamičnom vrijednošću 56](#_Toc138002447)

[Slika 4.13 Pojedinačni okviri snimke bez sinkronizacije 59](#_Toc138002448)

[Slika 4.14 Pojedinačni okviri snimke s dinamičnom sinkronizacijom 60](#_Toc138002449)

[Slika 4.15 Pojedinačni okviri snimke s fiksnom sinkronizacijom 61](#_Toc138002450)

# Uvod

Sinkronizacija vremena i podataka povezanih uređaja jedan je od problema koji se proteže od samih početaka računala. Dok se problem memorijske sinkronizacije rješavao napretkom tehnologije koristeći različite pristupe i metode, problem vremenske sinkronizacije, ponajviše zbog neograničene preciznosti, još je uvijek aktivna tema kod mnogih sustava. Povećanjem broja prijenosnih uređaja, koji su osim performansama limitirani i dostupnom energijom, sinkronizacija u raznim uvjetima postala je problematika visokog prioriteta. Svojstvo sinkronizacije, tj. vremenske i podatkovne ujednačenosti od izuzetne je važnosti u kritičnim sustavima te sustavima visoke preciznosti.

Glavna tema ovog rada je povezanost i sinkronizacija prijenosnih sustava baziranih na ESP32 seriji mikrokontrolera, koji su među najkorištenijima na tržištu. S ciljem postizanja optimalne razine sinkronizacije koristeći razne protokole i algoritme, u radu se problemu pristupa na nekoliko različitih načina te pokušava doći do zadovoljavajućeg rješenja uz nisku razinu kompleksnosti te minimalnu potrošnju energije.

# Osnovno o mikrokontrolerima

* 1. ESP32

ESP32 serija je nisko-budžetnih potrošača male snage sistema na čipu (*SoC*) mikrokontrolera s integriranim *Wi-Fi* i *Bluetooth* protokolima za komunikaciju. Kreiran je od strane kineske kompanije *Espressif Systems* kao direktan nasljednik serije ESP8266 mikrokontrolera sa sličnim mogućnostima.

Prvi modeli izašli su na tržište 2016. godine te su koristili *Tensilica Xtensa* LX6 procesor. U obitelji mikrokontrolera postoji nekoliko različitih serija te više desetaka modela, koji su svi bazirani na LX6, LX7 ili RISC-V mikroprocesorima. Jedna od glavnih prednosti ovih mikrokontrolera, osim male potrošnje, malog formata, niske cijene i velike brzine, jest velik broj integriranih modula poput antene, pojačala, filtera, sustava za kontrolu potrošnje te velikog broja ulaznih i izlaznih jedinica (GPIO). Osnovni modeli dolaze s 320 KiB RAM-a te rade na voltaži od 3.3V. *Flash* (ROM) memorija uređaja kreće se između 512 KiB i 32 MiB, ovisno o modelu.

Od ostale periferije na uređajima se mogu naći sljedeće komponente:

* DAC (*Digital-Analog Converter*)
* ADC (*Analog-Digital Converter*)
* Senzori na dodir
* Sučelja za komunikacijske protokole SPI, I2S, I2C, UART
* Sučelja za memorijske (SD) kartice
* *Ethernet* MAC sučelje
* CAN sabirnica
* PWM
* LED svjetla
* Wi-Fi modul
* Bluetooth modul
* Procesor za kriptografiju
* Sklopovlje za prekide, *watchdog*, brojači

ESP32 može se naći na tržištu u mnogo različitih formata i oblika kao i već spremnih pločica za razvoj koje sadrže dodatne komponente. U sljedećoj tablici prikazane su glavne razlike i modeli ESP32 serije mikroprocesora te njihove mogućnosti. Neke od promjena u modelima tijekom godina relevantne su za ovaj rad radi poboljšanja sposobnosti povezivanja uređaja te podrške za novije verzije protokola *Wi-Fi* i *Bluetooth*.

Glavne serije ESP32 obitelji su ESP32-S, ESP32-C i ESP32-H. [1]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Povezivost** | **Periferija** | **Procesor** | **Karakteristike** | **Modeli** |
| **ESP32** | Bluetooth LE (4.2), Bluetooth Classic, Wi-Fi 4 (2.4GHz) | 34 GPIO, kapacitivni senzori na dodir, Hall senzor, sučelje za SD karticu, Ethernet, SPI, UART, I2S, I2C, DAC, antena | 1 ili 2 Xtensa 32-bit LX6 mikroprocesora s promjenjivom frekvencijom između 80 i 240 MHz | Vrlo niska potrošnja energije tijekom aktivnog rada i u tihom načinu rada | WROOM, WROVER, MINI, PICO, DU1906, SOLO, ZERO |
| **ESP32-S2** | Wi-Fi 4 (2.4 GHz) | 43 GPIO, USB OTG, SPI, I2S, I2C, UART, LED PWM, sučelje za LCD, sučelje za kameru, 2x ADC, DAC, senzori na dodir, senzor za temperaturu, antena | Xtensa jednojezgreni 32-bit LX7 mikroprocesor s promjenjivom frekvencijom između 80 i 240 MHz, 320 KiB SRAM, 128 KiB ROM, 16 KiB RTC SRAM | Izvedba na jako niskom načinu rada, vrlo fino podešavanje sata, dinamična voltaža i skaliranje frekvencije; sigurnost - AES, SHA, RSA, sigurni boot, enkripcija | MINI (1, 2, 2U); SOLO (1, 2, U, 2U);  WROVER (1, I); WROOM (1, I) |
| **ESP32-S3** | Bluetooth LE (5.0), Bluetooth Classic, Wi-Fi 4 (2.4 GHz) | 45 GPIO, 12x ADC, SPI, I2S, I2C, PWM, RMT, DAC, UART, SD/MMC sučelje, TWAITM, antena | Xtensa dvojezgreni 32-bit LX7 mikroprocesor s promjenjivom frekvencijom između 80 i 240 MHz, 512 KiB SRAM, 384 KiB ROM, 8 KiB RTC SRAM | Dodatna podrška za vektorske instrukcije u MCU-u, SPI podrška, sigurnosne karakteristike sigurnog boots, AES-XTS enkripcija flash memorije, digitalno potpisivanje | MINI (1, 1U); WROOM (1, 1U, 2) |
| **ESP32-C2** | Bluetooth LE (5.0), Bluetooth Classic, Wi-Fi 4 (2.4 GHz) | 14 GPIO, SPI, UART, I2C, LED PWM kontroler, DMA kontroler, SAR ADC, senzor za temperaturu | RISC-V jednojezgreni 32-bitni procesor s frekvencijom od 120 MHz | Jako niska potrošnja, visoka kvaliteta RF modula | MINI (1, 1U); WROOM (01C, 02C, 02UC, 03, 04C, 05, 06C, 07 |
| **ESP32-C3** | Bluetooth LE (5.0), Bluetooth Classic, Wi-Fi 4 (2.4 GHz) | 22 GPIO, SPI, UART, I2C, LED PWM kontroler, DMA kontroler, SAR ADC, senzor za temperaturu, integrirana antena | RISC-V jednojezgreni 32-bitni procesor s frekvencijom do 160 MHz | Jako niska potrošnja, visoka kvaliteta RF modula, visoka razina enkripcije i dodatni sustav za kriptografske algoritma | MINI (1, 1U); WROOM (02, 02U) |
| **ESP32-C6** | Bluetooth LE (5.0), Bluetooth Classic, Wi-Fi 6 (2.4 GHz), Thread, Zigbee | 30 GPIO, SPI, UART, I2C, LED PWM kontroler, DMA kontroler, SAR ADC, senzor za temperaturu, integrirana antena | RISC-V jednojezgreni 32-bitni procesor s frekvencijom do 160 MHz | Jako niska potrošnja, visoka kvaliteta RF modula | MINI (1, 1U);  WROOM (1, 1U) |
| **ESP8266** | Wi-Fi 4 (2.4 GHz) | GPIO, UART, I2C, I2S, SDIO, PWM, ADC, SPI | Xtensa 32-bit LX6 mikroprocesor s frekvencijom do 160 MHz | Vrlo niska potrošnja u tihom načinu rada | WROOM (02D, 02U, 02, S2) |

Tablica 2.1 Usporedba uređaja iz ESP32 obitelji [2]

Dugo najavljivanja verzija mikrokontrolera s podrškom za *Bluetooth* 5.2, što bi omogućilo istovremenu povezanost i komunikaciju s više drugih uređaja, bila je ESP32-H2. Čip je izašao početkom 5. Mjeseca 2023. godine u eksperimentalnoj verziji bez podrške za 5.2, samo s osnovnom *Bluetooth* 5.0 podrškom. [3]

* 1. FreeRTOS

*FreeRTOS* je sustav za rad u stvarnom vremenu za ugradbene uređaje. Podržan je na 35 različitih platformi mikrokontrolera i najkorišteniji je operacijski sustav među malim i prijenosnim računalima. Kod ovog sustava je otvoren i dostupan svima pod MIT licencom. Prva verzija izašla je 2003. godine od strane kompanije *Real Time Engineers Ltd.* da bi u 2017. prešla u vlasništvo tvrtke *Amazon* pod AWS (*Amazon Web Services)* projektom.

Neke od glavnih podržanih platformi mikrokontrolera su ARM (ARM7, ARM9, *Cortex-M* serija), *MicroBlaze*, *Cortus*, *Espressif* ESP32 i RISC-V. Glavne prednosti ovog OS-a su vrlo mala količina memorije i procesorske snage potrebne za njegovo pokretanje kao i visoka razina kontrole nad radom. Uglavnom je pisan u C programskom jeziku radi lakšeg održavanja i performansi.

*FreeRTOS* uključuje podršku za višestrukim brojem dretvi, zadataka, semafora i brojača, kao i nekoliko načina rada niske potrošnje i prioritet izvršavanja dretvi. Osim toga nudi vrlo detaljnu statičku i dinamičku alokaciju memorije. Komunikacija s ostalom periferijom preko I/O i UART sučelja je moguća.

Zbog svih ovih karakteristika FreeRTOS je baza gotovo svakog *IoT* i prijenosnog uređaja kojemu je cilj mala potrošnja energije i brzo izvođenje. Neke od naprednih funkcija operativnih sustava kao što su *Linux* i *Windows* nisu dostupne, ali izostankom istih smanjena je kompleksnost i veličina cijelog sustava.

* 1. Bežićne tehnologije povezivanja mikrokontrolera

### Bluetooth

*Bluetooth* je bežična komunikacijska tehnologija kratkog dometa koja se koristi za razmjenu podataka između uređaja. Uz *Wi-Fi*, najkorištenija je metoda prijenosa podataka na tržištu te se uglavnom koristi u prijenosnim uređajima, računalima, pametnim satovima, kontrolerima, audio uređajima i ugradbenim uređajima.

Određeni dijelovi ovog poglavlju odnose se isključivo na rad *Bluetootha* na mikrokontrolerima i specifično na ESP32 obitelji uređaja. Zbog velike količine podataka i potrošnje baterije, *Bluetooth* se u ugradbenim sustavima uglavnom koristi u *Low Energy* načinu. U nastavku će detaljno biti objašnjen stog i način rada tog načina, dok će standardni *Bluetooth* način biti objašnjen samo u kratkim crtama.

#### Bluetooth Classic (BR/EDR)

Standardni *Bluetooth* način prijenosa je radio niske snage koji prenosi podatke preko 79 kanala u 2.4GHz-nom ISM (*Industrial, scientific, medical*) pojasu. Način komunikacije između uređaja je isključivo direktna povezanost uređaja (*point-to-point*) te se pretežito koristi radi prijenosa zvuka visoke kvalitete ili velikih podataka. Omogućava veliku propusnost i zahtjeva popriličnu količinu snage za rad.

*Bluetooth* BR protokol prvi je i osnovni način rada te se bazirao na već spomenutoj 2.4GHz frekvenciji koja ne zahtjeva licencu te je besplatna za korištenje. Osnovna brzina prijenosa podataka korištenjem ovog načina je oko 1 MB/s.

*Bluetooth* 2.0 specifikacija uvela je novi način prijenosa podataka kroz *Bluetooth* EDR koji je omogućavao trostruke brzine osnovnog Bluetootha. Oba načina se sada podrazumijevaju kao jedan, osnovni način prijenosa te ih se često referencira kao *Bluetooth Classic*.

#### Bluetooth Low Energy (BLE)

S izlaskom specifikacije *Bluetooth* 4.0 predstavljen je i *Low Energy* način rada. Baziran je na rješenju kompanije *Nokia* zvanom *Wibree* iz 2006. godine, koji je nakon integracije u *Bluetooth* protokol 2009. dobio novo ime *Bluetooth Low Energy*. Optimiziran je za korištenje u aplikacijama s kratkim ciklusom rada, tj. gdje je komunikacija povremena s minimalnim slanjem podataka. Tako se ciklus rada aktivira u određenim periodima (1 sekunda, svakih nekoliko sekundi ili rjeđe) te sam aktivan rad traje puno manje zbog malog prijenosa podataka što omogućava minimalnu potrošnju energije.

BLE koristi se i u mnogim drugim prijenosima podataka kao što su određivanje i slanje lokacije, prijenos podataka kod medicinske i sportske opreme te komunikacija sustava za nadgledanje i kontrolu. [4]

Stog protokola podijeljen je na 3 glavna dijela. Najniži fizički sloj *Bluetooth* LE PHY direktno se preslikava na OSI referentni model fizičkog sloja. Sloj podataka (MAC + LLC) mapiran je na dva sloja BLE-a - *Link layer* (LL) i L2CAP (*Logical link control and adaptation protocol*). Ostatak OSI slojeva preslikava se aplikacijske profile i druge protokole koji upravljaju komunikacijom.

A picture containing text, screenshot, font, number

Description automatically generated

Slika 2.1 Preslikavanje BLE stoga na OSI referentni model

Sama funkcionalnost protokola podijeljena je u tri sloja - kontroler, domaćin te profili i servisi aplikacije. Preslikavanje slojeva u podskupine nešto je drugačije nego ono na OSI referentni model te je prikazano na idućoj ilustraciji.

A picture containing text, screenshot, number, diagram

Description automatically generated

Slika 2.2 BLE stog

Sloj kontrolera obuhvaća Bluetooth LE PHY i LL slojeve kao i HCI (*Host controller interface)* koji služi za komunikaciju sa slojem domaćina. Fizički sloj dijeli istu 2.4 GHz frekvenciju kao i *Bluetooth Classic* te *Wi-Fi*. Podijeljen je u 40 različitih kanala, od kojih se prvih 37 koristi za slanje podataka korisnika, propusnosti 2 MHz koji se nalaze između 2.400 GHz i 2.4835 GHz. Brzina prijenosa podataka za nekodiranu transmisiju iznosi 1 MB/s, dok sve inačice Bluetooth 5.1 i više imaju podršku za nekodiranu transmisiju od 2MB/s te kodiranu transmisiju od 125 KB/s do 500 KB/s, ovisno o razini zaštite.

LL sloj vrši direktnu komunikaciju s fizičkim slojem i preko HCI-ja se spaja na domaćina, preko kojeg se onda vrši komunikacija na višoj razini (pakiranje, transmisija, …).

Sloj domaćina sastoji se od HCI-ja (sa strane domaćina), L2CAP, ATT, GATT, SMP i GAP protokola. HCI komunicira direktno s kontrolerom te definira set komandi i događanja za transmisiju u primanje paketa, prilikom čega pretvara sve u sirove podatke prije slanja u kontroler. L2CAP inkapsulira sirove podatke iz kontrolera (ili raspakirava ukoliko idu prema kontroleru) s obzirom na konfiguraciju specificiranu u gornjim slojevima. ATT (*Attribute protocol*) definira uloge u arhitekturi klijent-server te određuje centralnu figuru, kao i organizaciju podataka u atribute kao na ilustraciji.

A picture containing text, screenshot, line, font

Description automatically generated

Slika 2.3 BLE ATT atributi

Navedeni atributi koriste se za sljedeće:

* *Attribute handle* - 16-bitni identifikator pridijeljen klijentima od strane servera
* *Attribute type* - UUID (*Universally unique identifier*) određenog servisa kojim se definira konkretna uloga u sustavu (npr. 0x2A37 predstavlja sustav mjerenja otkucaja srca)
* *Attribute value* - glavni podatci prijenosa, duljina i sadržaj ovog polja promjenjiv je ovisno o vrsti atributa
* *Attribute permissions* - set dopuštenja povezanih uz svaki atribut kao što su mogućnost čitanja i pisanja te razina sigurnosti potrebna za akcije nad podatcima

GATT (*generic attribute profile*) pruža referencu za sve GATT profile. Služi za omatanje ATT-a i koordinaciju razmjene profila u BLE stogu. Profili uključuju informacije i podatke o tome kako ih obraditi, UUID i listu dopuštenja. GATT koristi arhitekturu klijent-server, te osim klijenta i servera, sastoji se od servisa, karakteristike i opisa karakteristike. SMP pruža zaštitu u vidu sigurnosnih algoritama te enkripcije/dekripcije podataka, dok GAP komunicira direktno s uređajem te osigurava povezanost i sigurnost za vrijeme prijenosa. Kroz taj sloj komunicira se s aplikacijom koja se nalazi na uređaju i koja služi za direktan prijenos i komunikaciju u cijelom protokolu kao i komunikaciju s korisnikom kroz UI (ukoliko isti postoji).

|  |  |  |
| --- | --- | --- |
|  | **Bluetooth Low Energy (LE)** | **Bluetooth Classic** |
| **Frekvencije** | 2.4GHz ISM Band (2.402 – 2.480 GHz korišteno) | 2.4GHz ISM Band (2.402 – 2.480 GHz Utilized) |
| **Kanali** | 40 kanala s razmakom od 2 MHz (3 reklamirajuća kanala / 37 podatkovnih kanala) | 79 kanala s razmakom od 1 MHz |
| **Korištenje kanala** | *Frequency-Hopping Spread Spectrum* (FHSS) | *Frequency-Hopping Spread Spectrum* (FHSS) |
| **Modulacija** | GFSK | GFSK, π/4 DQPSK, 8DPSK |
| **Brzina prijenosa podataka** | LE 2M PHY: 2 Mb/s LE 1M PHY: 1 Mb/s LE Coded PHY (S=2): 500 Kb/s LE Coded PHY (S=8): 125 Kb/s | EDR PHY (8DPSK): 3 Mb/s EDR PHY (π/4 DQPSK): 2 Mb/s BR PHY (GFSK): 1 Mb/s |
| **Tx snaga** | ≤ 100 mW (+20 dBm) | ≤ 100 mW (+20 dBm) |
| **Rx osjetljivost** | LE 2M PHY: ≤-70 dBm LE 1M PHY: ≤-70 dBm LE Coded PHY (S=2): ≤-75 dBm LE Coded PHY (S=8): ≤-82 dBm | ≤-70 dBm |
| **Transport podataka** | Asinkrono, sinkrono, izosinkrono | Asinkrono, sinkrono |
| **Komunikacijske topologije** | *Point-to-Point, Broadcast, Mesh* | *Point-to.Point* |

Tablica 2.2 Razlike između BLE i Bluetooth Classic protokola [5]

#### Bluetooth 4.2

S *Bluetooth* 4.0 verzijom napravljen je veliki iskorak uvevši BLE kao način rada te su sljedećih nekoliko verzija predstavljale minorna poboljšanja. *Bluetooth* 4.2 verzija podržana je na prvoj verziji ESP32 te se sama ne razlikuje mnogo od osnovne 4.0 verzije. S 4.1 verzijom poboljšali su se sigurnosni protokoli te prijenos audio podataka dok je 4.2 udvostručila brzinu prijenosa podataka kao i proširila domet veze. U toj verziji dodala se i nova podrška za sigurnosne protokole kroz privatne ključeve što je omogućavalo sigurno i stabilnu vezu u prostoru s više uređaja bez prevelikih smetnji.

#### Bluetooth 5.0

Veliki iskorak napravljen je u novoj major verziji 5.0. Brzine prijenosa udvostručile su se s obzirom na verziju 4.2 te u teoriji dosežu i do 2 MB/s (u praksi to iznosi oko 1.4 MB/s) [6]. Domet se učetverostručio na velikih 200 metara u otvorenom prostoru što se postiže smanjenjem stvarne brzine prijenosa. 5.0 je i po prvi puta omogućio domaćinu spajanje i komunikaciju s dva uređaja u isto vrijeme. Verzija 5.1 donijela je inkrementalne pomake od kojih je najznačajniji određivanje smjera veze između dva uređaja i jednostavnije otkrivanje novih uređaja. U verziji 5.2 predstavljen je novi audio kodek LC3 koji značajno povećava propusnost i poboljšava performanse baterije u LE modu. Ostatak promjena u 5.2 i u 5.3 uglavnom se odnosi na poboljšanja u prijenosu audio podataka primarno zbog zahtijevanja tržišta i naglog rasta udjela bežičnih slušalica i zvučnika na tržištu. Prvi uređaj iz ESP obitelji koji je trebao podržavati Bluetooth 5.2 i prijenos audio podataka u isto vrijeme na dva uređaja je ESP32-H2, što se ipak nije dogodilo te je *Espressif* odlučio objaviti proizvod samo s podrškom za Bluetooth 5.0. [7] [8]

### Wi-Fi

*Wi-Fi* kao protokol vrlo je raširen i opće poznat, stoga ovo poglavlje ne sadrži previše informacija o osnovnim karakteristikama istog.

Većina ESP32 uređaja, izuzev ESP32-C6 sadrži podršku za *Wi-Fi* 4 koji je trenutno najrašireniji *Wi-Fi* standard. Prve 4 verzije *Wi-Fi* protokola po IEEE standardu poznatije kao 802.11, 802.11b, 802.11a i 802.11g nisu koristile imena verzije. Sve verzije radile su na 2.4 GHz pojasu, osim verzije 802.11a koja je radila na 5 GHz. Propusnost je rasla kroz godine, ali nije prelazila 54 Mbit/s.

2008. godine na tržište je izašla verzija *Wi-Fi* 4 (IEEE 802.11n) s velikim napretkom u propusnošću koja je sezala do 600 Mbit/s s velikim povećanjem u dometu. Podrška za starije verzije protokola bila je prisutna te su se mogli koristiti interoperabilno u 2.4 i 5 GHz pojasu. Upravo zbog te interoperabilnosti, jednostavnosti i ne prevelikom zahtjevnošću, većina uređaja u *IoT* svijetu koriste *Wi-Fi* 4.

2014. godine izlazi *Wi-Fi* 5 (IEEE 802.11ac) kojim se maksimalna propusnost povećala na 6933 Mbit/s te koji je radio isključivo na 5 GHz pojasu.

2019. godine izlazi *Wi-Fi* 6 (IEEE 802.11ax) kojim se propusnost povećava na 9608 Mbit/s te kojim se ponovno omogućava interoperabilnost sa starijim standardima kroz podršku za 2.4 i 5 GHz pojas. U verziji *Wi-Fi* 6E dodana je podrška za 6 GHz pojas.

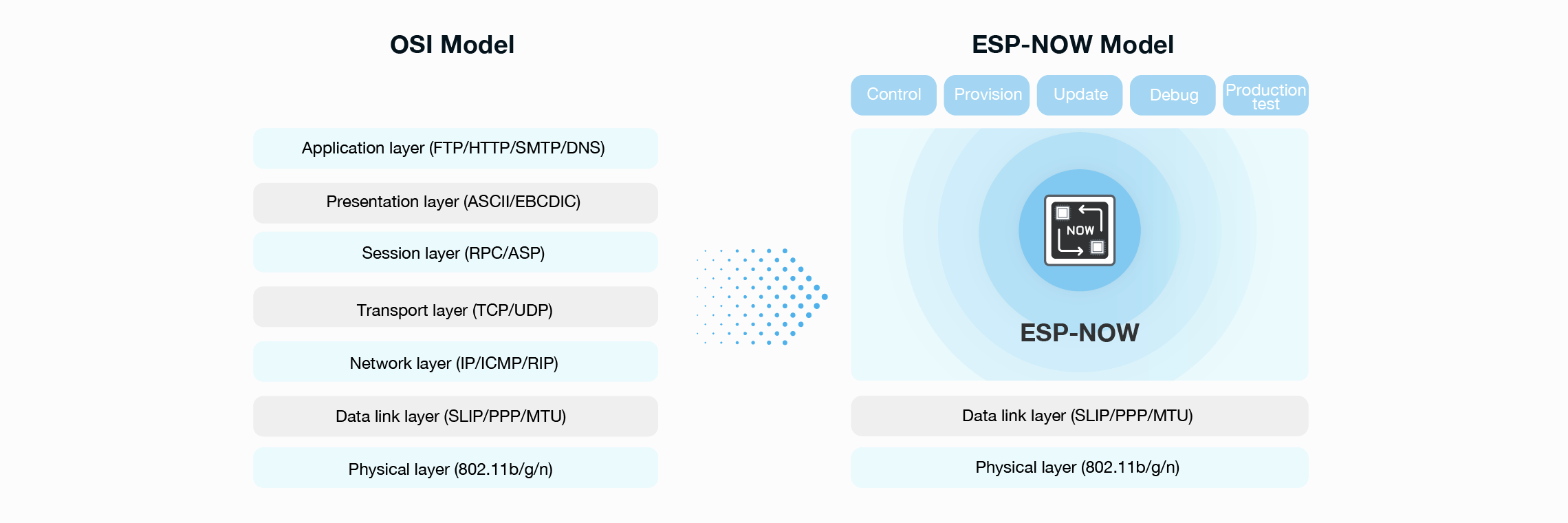
ESP32 uređaji mogu koristiti *Wi-Fi* u klijentskom načinu rada (STA) ili u serverskom načinu rada (AP - *access point*) te imaju podršku za enkripciju i sigurnosne algoritme poput WPA2 i WPA3. [9]

### ESP-NOW

Prvi nativni bežični komunikacijski protokol za ESP32, u potpunosti razvijen od strane kompanije *Espressif*, naziva *ESP-NOW*, omogućuje komunikaciju između više ESP uređaja bez korištenja *Wi-Fi* tehnologije. Princip rada i povezivanja vrlo je sličan onome koji koristi *Wi-Fi*, no uz znatno manju potrošnju energije, te je pogodniji za korištenje prijenosnih uređaja s manjim baterijama. Podatci se razmjenjuju između dva uređaja bez stalne veze uz osnovnu brzinu prijenosa od 1 MB/s.

ESP-NOW može raditi zajedno s *Bluetooth* i *Wi-Fi* protokolima te podržava sve uređaje iz serija ESP8266, ESP32, ESP32-S i ESP32-C.

Sam protokol baziran je na slojevima iznad sloja podataka te tako inkapsulira mrežni, transportni, sesijski, prezentacijski i aplikacijski sloj iz OSI modela. Podržani načini povezivanja su “*one-to-many*” i “*many-to-many*” što omogućava brzu razmjenu podataka i kontrolu uređaja. Korištenjem ECDH i AES algoritama osigurava se sigurnost prijenosa i podataka te je vrlo lagan i zahtjeva malo resursa procesora tijekom korištenja.



Slika 2.4 ESP-Now

Jedan od glavnih aduta ovog protokola je nedostatak ‘rukovanja’ između uređaja te držanja stalne veze. Ta karakteristika omogućava znatno produljenje trajanja baterije. Dodatna prednost je i istovremena komunikacija u oba smjera, veći broj spojenih uređaja od osnovne verzije BLE te domet koji u različitim načinima rada doseže i nekoliko stotina metara.

### Ostali protokoli i standardi

U zadnjih nekoliko godina zbog naglog povećanja *IoT* uređaja i potrebe za brzim i efikasnim bežičnim protokolima, pojavilo se nekoliko standarda koji su pokušali, i još uvijek pokušavaju, preuzeti dominaciju na tržištu.

Protokoli navedeni u ovom poglavlju neće biti detaljnije razmatrani zbog njihove rijetke korištenosti kao i činjenice da nisu podržani od većine uređaja ESP32 obitelji. Samo ESP32-H2 i uređaji iz ESP32-C serije imaju djelomičnu ili potpunu podršku za neke od ovih protokola. Zbog njihove važnosti u ovom području vrijedi ih spomenuti.

#### Thread

Baziran na IPv6 Internet protokolu, *Thread* je protokol niske potrošnje namijenjen mrežama *IoT* uređaja. Komunikacija se odvija preko 6LoWPAN mreže baziranoj na IEEE 802.15.4 standardu za prijenos male količine podataka u mrežama na 2.4 GHz frekvenciji. *Thread* podržava enkripciju te direktno IP adresiranje. Glavne prednosti ovog protokola su niska potrošnja i latencija, domet, sigurnost i sposobnost samo-otklanjanja kvara.

Glavna ideja *Threada* bila je slobodno korištenje bez licence i činjenica da se bazira na već postojećim tehnologijama i slojevima. Grupa koja radi na ovom protokolu osnovana je 2014. Godine te se sastoji od nekih od najvećih tehnoloških firmi kao što su ARM, NXP, *Google*, *Samsung*, *Qualcomm* te *Apple*, koji se pridružio 2018. godine.

#### Zigbee

*Zigbee* je osnovno vrlo sličan protokol *Thread*, također baziran na IEEE 802.15.4 standardu koji se primarno koristi za kreaciju malih PAN mreža. Koristi se u kućnim uređajima i medicinskoj opremi s bez velikih potreba za propusnost, energiju i domet. Jednostavnost hardwarea i softwarea naspram ostalih protokola, kao što su *Bluetooth* i *Wi-Fi,* čine *Zigbee* lakšim i jeftinijim za implementaciju.

*Zigbee* se prvi put pojavljuje na tržištu 2005. Godine te je u međuvremenu izašlo nekoliko verzija i poboljšanja. Nakon kreacije i izlaska *Thread* protokola na tržište, potvrđeno je da će oba protokola biti međusobno podržana te da će moći jednostavno komunicirati.

#### Matter

U listopadu 2022. godine na tržište i službeno izlazi prva verzija standarda zvanog *Matter*. Riječ je o standardu kojim se pokušava ujediniti komunikacija *IoT* uređaja bez obzira čijeg su proizvođača i koje protokole koriste, s ciljem da je moguća sigurna i stabilna komunikacija sa ili bez internetske konekcije.

Standard je začet 2019. godine od strane tri najveće tehnološke tvrtke - Amazona, Applea i Googlea, kao i tvoraca *Zigbee* protokola, među kojima se nalazi i Samsung. Standard je besplatan i otvorenog koda te baziran na IP protokolu kao i prethodno spomenuti protokoli.

### Povezivanje uređaja iz ESP-32 serije

Prve verzije ESP32 uređaja izašle su na tržište 2016. godine kao direktan nasljednik ESP8266 koji je izašao nekoliko godina ranije. Verzija Bluetootha 4.2 podržana je od prve verzije čipa te je ista omogućavala korištenje i standardne *Bluetooth* konekcije kao i *Bluetooth Low Energy* (BLE). Drugo izdanje čipa u nešto jednostavnijoj kombinaciji, ESP32-S2, nije imalo podršku za *Bluetooth*. Sva kasnija izdanja čipa (ESP32-C3, ESP32-S3 i ESP32-C6) počevši od kraja 2020. godine podržavaju Bluetooth 5.0 čime se značajno poboljšavaju mogućnosti i performanse u komunikaciji između uređaja. Svi čipovi iz serije imaju po jedan RF (*radio frequency*) modul što znači da istovremena komunikacija putem *Wi-Fi* i *Bluetooth* kanala nije moguća. [10]

ESP32 može biti istovremeno povezan i na *Wi-Fi* i na *Bluetooth* prilikom čega poseban modul odabire komunikaciju jednim od ta dva načina pomoću prioriteta. Taj način rada se naziva načinom koegzistencije.

A white rectangle with black text

Description automatically generated with medium confidence

Slika 2.5 ESP32 Wi-Fi and Bluetooth coexsistence

Vrijeme korištenja RF modula se tako dijeli na *Wi-Fi* i BLE s obzirom na određene faze. Vremena kojima raspolažu ovise o statusu *Wi-Fi* konekcije, koja se djelomično prioritizira te dinamično povećava i smanjuje. Postoje četiri faze statusa konekcije:

* *IDLE* (BESPOSLEN) - Bluetooth u potpunosti kontrolira i koristi RF modul
* *CONNECTED* (SPOJEN) - period koegzistencije je veći od 100 milisekundi
* *SCAN* (SKENIRANJE) - vrijeme rezervirano za Wi-F i koegzistenciju je dulje nego u statusu SPOJEN, vrijeme rezervirano za Bluetooth je smanjeno
* *CONNECTING* ( SPAJANJE) - vrijeme rezervirano za Wi-Fi je dulje nego u statusu SPOJEN, vrijeme rezervirano za Bluetooth je smanjeno

Odabir statusa koegzistencije događa se automatski, s mogućnošću ručnog odabira. Za vrijeme rada *Bluetooth* načina komunikacije, omogućeno je korištenje dvostruke *Bluetooth* konekcije istovremeno, u običnom načinu i u *Low Energy* načinu. Kada se koristi individualno, prijenos podataka običnog načina je oko 230 KB/s, a *Low Energy* načina 40 KB/s (s mogućnošću optimizacije do 90 KB/s). Korištenjem simultanog načina, brzine prijenosa padaju i do 50%. [11]

#### Stog protokola Bluetooth

Kontrola Bluetootha odrađuje se preko *Host Controller Interface* (HCI) sučelja koje se nalazi na stogu kontrolera kojega direktno povezuje sa slojem domaćina. Ovaj standard omogućava domaćinu stoga ili kontrolera da izmjenjuju uloge uz minimalnu prilagodbu. Postoji nekoliko standarda transportnog sloja preko kojeg se odvija komunikacija od kojih su najčešći USB i UART.

Osnovna verzija ESP32 ima podršku za tri vrste HCI-a - UART, SPI i VHCI. Samo se jedno sučelje može koristiti istovremeno te je UART postavljen kao zadani.

Razvojno okruženje ESP-IDF pruža podršku za dva različita stoga domaćina - *Bluedroid* i *Apache* *NimBLE*.

*Bluedroid* podržava rad običnog *Bluetootha* kao i BLE, dok *NimBLE*, sukladno imenu, podržava rad isključivo BLE tehnologije. *Bluedroid* je nešto kompliciraniji i memorijski zahtjevniji te je preporučeno koristiti *NimBLE* ukoliko se radi isključivo s BLE. [12]

#### Usporedba protokola na ESP32 uređajima

Iako je fokus ovog rada na komunikaciji preko Bluetooth *Low-energy* protokola zbog minimalne potrošnje energije, nekoliko je razloga zašto bi se razmatrale i drugi već spomenuti protokoli za povezivanje.

Jedna bitna stvar je brz razvoj tehnologije i novih protokola. Kao što je i prikazano u prethodnom poglavlju, mnogi protokoli dobivaju nove mogućnosti i nove verzije svakih nekoliko godina te su njihove tehničke specifikacije i mogućnosti promjenjive. Standardi su isto nešto što se mijenja i što može varirati iz godine u godinu te se razlike u karakteristikama određenih protokola svake godine sve više smanjuje pošto se svi protokoli usavršavaju.

Druga bitna stavka je činjenica da su ESP32 uređaji tek nekoliko godina na tržištu te da su ponašanja i karakteristike različitih modela još uvijek neistražene. Mnogi novi modeli s novim mogućnostima i podrškama za novije protokole i standarde dodatno kompliciraju analize, tako da je teško dati definitivan odgovor na pitanje koji je protokol najbolje koristiti kod uređaja s baterijom kako bi se potrošilo što manje energije.

Fokus u ovom poglavlju ipak će biti na usporedbi *Bluetooth* 4.2 i *Wi-Fi* 4 (IEEE 802.11n) verzijama protokola, kao i na *ESP-NOW*, zbog široke rasprostranjenosti tih tehnologija na gotovo svim uređajima iz ESP32 obitelji. S obzirom da je glavna tema rada sinkronizacija kod prijenosnih uređaja, glavni parametri koji su promatrani su latencija i potrošnja energije. Brzina prijenosa (propusnost), stabilnost signala i domet nisu podjednako bitni.

U istraživanju provedenom 2021. godine [13] testirana su sva tri navedena protokola. Osim potrošnje energije i latencije, analiziran je domet, stabilnost signala, komunikacija sa smetnjama, propusnost i jačina signala sa i bez antene. Izmjereno je da *Bluetooth* protokol ima najnižu potrošnju energije, u prosjeku oko 40% manju od *Wi-Fi* protokola te nekoliko puta manju od *ESP-NOW* protokola, ovisno o tome radi li se samo o spajanju ili aktivnoj transmisiji. Bluetooth također ima najveću latenciju, u prosjeku duplo veću od Wi-Fi i 5 puta veću od *ESP-NOW* protokola. Ti rezultati su očekivani s obzirom na proces rukovanja koji je potreban odraditi kod Bluetootha. *ESP-NOW* pruža najveći domet i otpornost signala, no i najmanju propusnost, dok je domet i otpornost signala Bluetootha najlošija s obzirom na druge protokole.

*Bluetooth* je korišten u standardnom načinu, a ne u *Low Energy* načinu rada. S obzirom na karakteristike i napretke koje taj način rada nudi, može se pretpostaviti da bi potrošnja energije bila značajno manja te tako razlika naspram ostalih protokola još veća, dok se latencija ne bi značajnije mijenjala.

Slični rezultati dobiveni su i u još jednom istraživanju. [14] Uspoređivali su se potrošnja energije i propusnost na BLE i *ESP-NOW* protokolima također koristeći ESP32. Slični rezultati su dobiveni kao i u prethodnom istraživanju gdje je BLE protokol trošio dvostruko manje energije za iste zadatke.

Ono što nije zanemarivo kod implementacije ovih protokola, pogotovo kod manjih i jednostavnijih projekata je količina koda i potrebno vrijeme postavljanja. *ESP-NOW* i *Wi-Fi* pružaju puno jednostavnije osnovne implementacije, dok je kod *Bluetooth* protokola potrebno vrlo detaljno proučiti i postaviti servise i karakteristike komunikacije kako bi se dobili željeni rezultati. U službenim osnovnim primjerima implementacije ovih protokola od strane *Espressif*-a, razlike u broju linija koda za primjer servera su sljedeće:

* BLE - *NimBLE* [15] - 472 linije
* BLE - *Bluedroid* [16] - 699 linija
* *Bluetooth Classic* [17] - 487 linija
* *ESP*-*NOW* [18] - 128 linija
* *Wi-Fi* [19] - 90 linija

S obzirom na sve ove parametre i činjenicu da je ipak limitirana potrošnja energije najbitnija stavka u komunikaciji ovakvih uređaja, BLE još uvijek se smatra najboljim protokolom za jednostavnu, brzu i efikasnu komunikaciju.

# Sinkronizacija

Vremenska sinkronizacija prijenosnih uređaja poznat je problem kojeg se pokušava riješiti s nekoliko različitih pristup. Glavne prepreke u sinkronizaciji su potrošnja energije i latencija u komunikaciji. Bežična komunikacija najčešći je način komunikacije i sinkronizacije kod prijenosnih i *IoT* uređaja zbog manjka žica, velikog dometa i male potrošnje energije. Postoji nekoliko pristupa ovom problemu s obzirom na potrošnju energije, robusnost, stabilnost i preciznost sinkronizacije.

Sinkronizacija može biti postignuta razini stvarnog vremena ili na razini vremenske oznake. Kod stvarnog vremena cilj je postići izvršavanje iste ili slične akcije na više različitih uređaja u isto ili približno isto vrijeme te se generalno koristi kod uređaja koji upravljaju drugim uređajima ili nekom drugom periferijom. Sinkronizacija na razini vremenske oznake koristi se pri skupljanju podataka, kada sam trenutak izvedbe radnje nije u prvom planu već je potrebno skupiti podatke s više različitih uređaja. Pomoću sinkronizacije osigurava se da su podatci skupljeni u okvirno isto vrijeme te da se usporedno mogu analizirati vremenske oznake.

Sinkronizacija na razini stvarnog vremena kompleksniji je problem zbog nepredvidivosti vremena izvođenja akcija na uređajima i trajanja prijenosa poruke prilikom komunikacije. Fokus u sljedećim poglavljima je sinkronizacija u stvarnom vremenu.

* 1. Direktna komunikacija

### Sinkronizacija pomoću signala

Jedan od najjednostavnijih načina sinkronizacije jest direktno slanje signala. Nakon što glavni uređaj (server) pošalje signal klijentu ili klijentima, jedini vremenski odmak između odrađivanja radnji bit će vrijeme putovanja poruke, koje varira ovisno o korištenom protokolu.

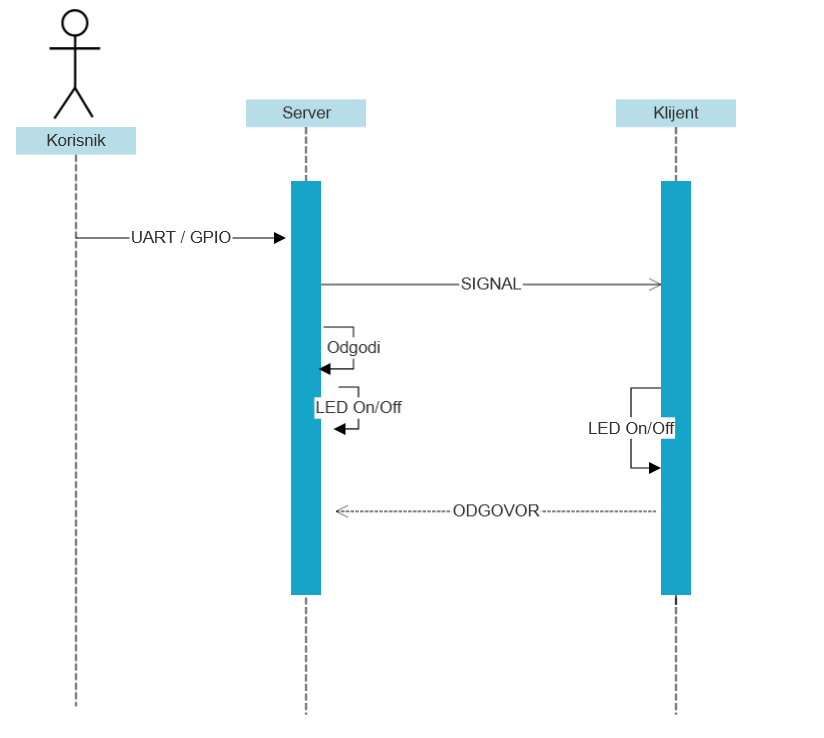
Niska latencija je u stvarnim sustavima veoma bitna stavka, no nije nužna za postizanje visokog stupnja sinkronizacije. Ukoliko se vrijeme putovanja signala može izračunati s visokom razinom točnosti, vrijeme latencije ne utječe na samu sinkronizaciju koja se postigne s odmakom izvršavanja radnje na glavnom uređaju. U prijenosnim sustavima trajanje baterije često je bitnija stavka od same brzine, dok sustavi s manjom latencijom često zahtijevaju robusnije i zahtjevnije radnje koje troše više energije od jednostavnijih sustava.

U radu su implementirani sljedeći primjeri ovog oblika sinkronizacije tijekom kojih su korištena dva ESP32 uređaja i dva LED svjetla s ciljem kontrole LED svjetala u okvirno istom trenutku. Pristup i algoritmi korišteni u nastavku rada su ovdje opisani u kratkim crtama sljedećim dijagramima. A picture containing text, diagram, line, parallel

Description automatically generated

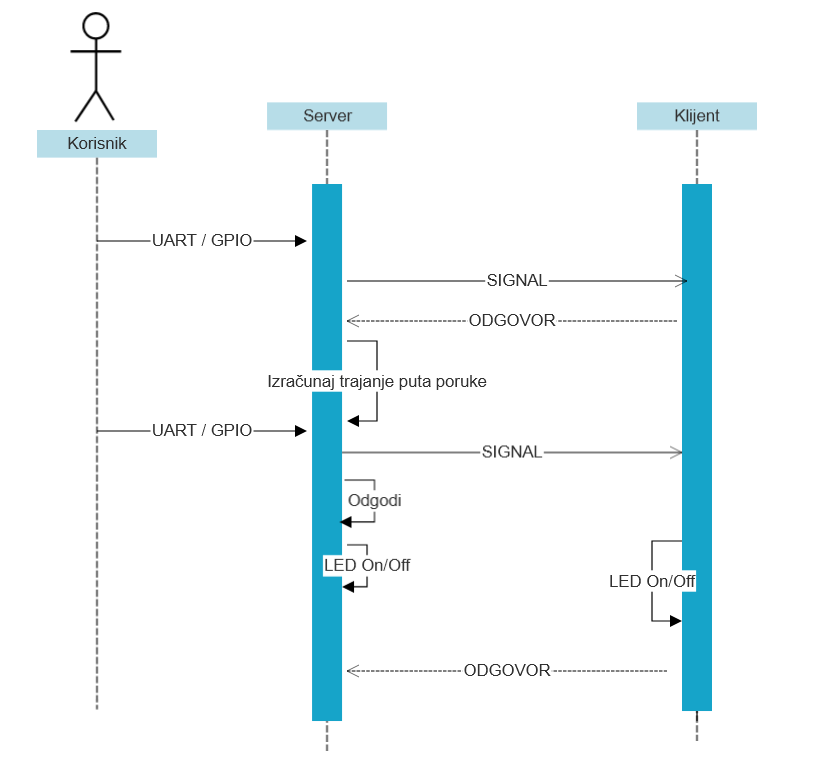
Slika 3.1 UML sekvencijski dijagram - Komunikacija bez sinkronizacije

Osnovni primjer koristi se kao baza sinkronizacijskih algoritama. Na dijagramu je prikazana komunikacija bez ikakvog implementiranog oblika sinkronizacije i očekivano ponašanje uređaja u tom slučaju.



Slika 3.2 UML sekvencijski dijagram - Sinkronizacija s fiksnim odmakom

Poboljšani pristup koristi fiksni odmak akcije prvog uređaja nakon slanja poruke kako bi se postigla prividna sinkronizacija. Vrijeme odgode unaprijed je izračunato i njime se odgađaju sve radnje do približnog vremena akcije na drugom uređaju.

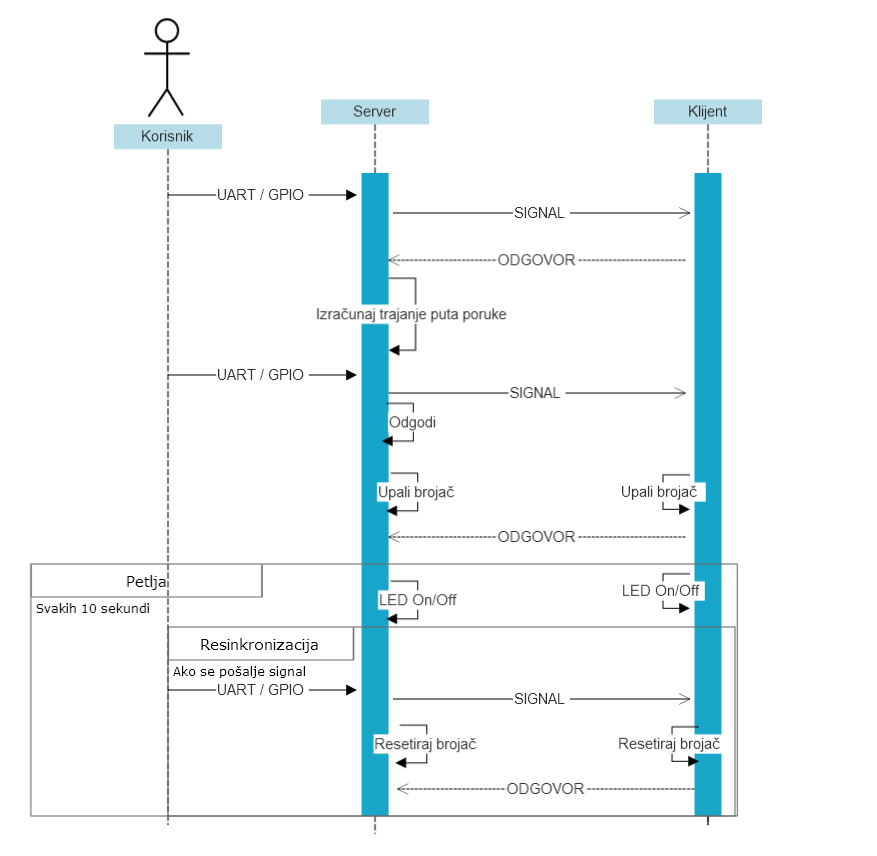


Slika 3.3 UML sekvencijski dijagram - Sinkronizacija s dinamičnim odmakom

Unaprijeđeni algoritam prethodnog načina u kojem se prvo pošalje poruka kako bi se izračunalo trajanje puta i obrade poruke te se ta vrijednost koristi za odmak akcije prilikom slanja. Ovim načinom sinkronizacije postiže se konzistentnost točnosti odgode kod svakog slanja poruke.

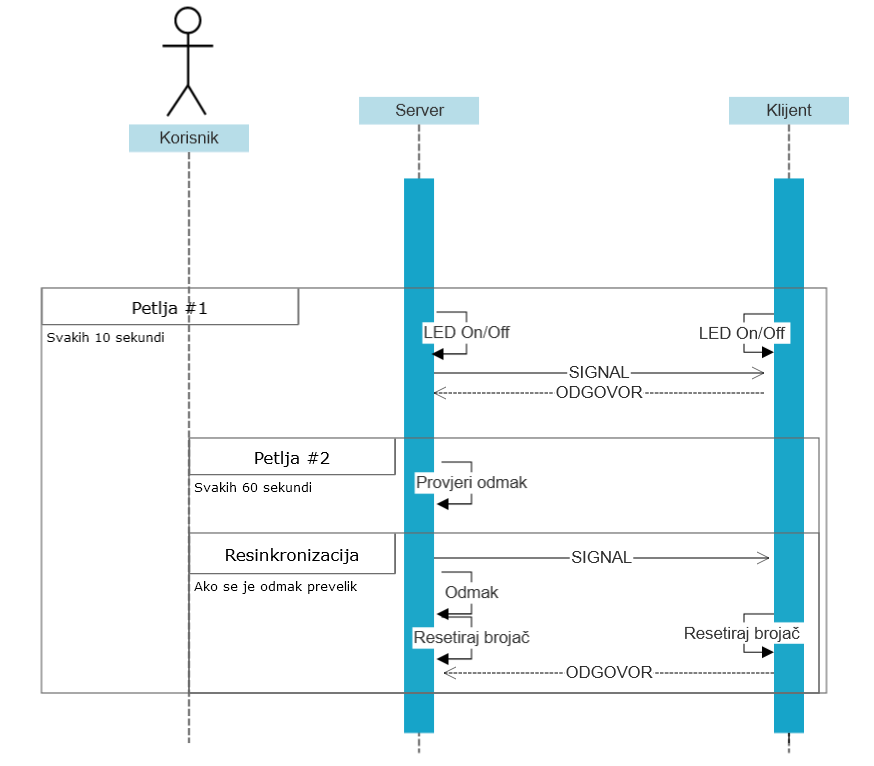
### Sinkronizacija unutarnjeg brojača

ESP uređaji temelje sve svoje radnje na unutarnjem satu koji je opisan u sljedećim poglavljima. Preciznost i konzistentnost tog sata ovisi o korištenom mehanizmu. Čak i najjednostavniji unutarnji RTC ESP32 uređaja dovoljno je precizan za osiguravanje visoke razine točnosti u normalnom načinu rada, uz poneka odstupanja u tihom načinu rada i pri ekstremnim temperaturama.

Određeni događaji aktiviraju se pomoću prekidne rutine. Pomoćni brojač se pokrene te svaki put kad njegova vrijednost dođe do 0, dogodi se prekid i pokrene nova funkcija. Pritom se brojač ponovno može pokrenuti i tako napraviti cikličnost prekidnih funkcija. Brojač također koristi isti oscilator te radi na istoj frekvenciji kao i odabrani unutarnji sat. 

Slika 3.4 UML sekvencijski dijagram - Periodička sinkronizacija na zahtjev

Jedan od načina da se zadrži sinkronizacija unutarnjih satova (brojača) i nakon nekog vremena je pomoću povremene provjere istih. Brojači se mogu ponovno pokrenuti kod svake komunikacije i tako osigurati da će signali u skorijoj budućnosti biti sinkronizirani. Ukoliko njihova trenutna razlika ne odstupa više od njihove početne razlike, brojač nastavlja s radom, a ukoliko se povećala, brojač se resetira. Sustav se može i dodatno zakomplicirati dodavanjem drugog brojača koji će raditi periodičku provjeru, a ne provjeru na zahtjev.



Slika 3.5 UML sekvencijski dijagram - Periodička resinkronizacija

* 1. Sinkronizacija sata

### Network Time Protocol (NTP)

Jedan od najstarijih protokola za sinkronizaciju vremena, i Internetskih protokola općenito, NTP je službeno objavljen 1985. Koristi se pretežito u mrežama u kojima se koristi slanje paketa i u kojima je latencija varijabilna. Glavni cilj protokola je sinkronizacija svih povezanih uređaja sa sistemskim satom (UTC) s maksimalnim odmakom od nekoliko milisekundi.

Za postizanje tog cilja koristi se algoritam presjeka koji je baziran na *Marzullovom* algoritmu. Algoritam bira izvore iz kolekcije izvora sa šumovima kako bi procijenio točno vrijeme kašnjenja i sa što manjim odmakom sinkronizirao ostale uređaje. Razlika s obzirom na *Marzullov* algoritam je što izračunati interval odmaka uređaja od “referentne točke” ne mora biti nužno u odnosu na centralnu točku. NTP tako izabire referentnu točku s kojom će utjecaj podrhtavanja (*jitter*) biti najmanji.

NTP pokreće se od strane *User Datagram Protocola* (UDP) komunicirajući kroz port 123. Mreža dobiva vrijeme iz prethodno određenog izvora vremena te prosljeđuje to vrijeme putem mreže. Sinkronizacija se uobičajeno odvija u razdoblju između 64 i 1204 sekunde, ovisno o broju povezanih uređaja i njihovoj konfiguraciji. Komunikacija se odvija pomoću razina tj. mrežne udaljenosti određenih računala od glavnog izvora. [20]

A picture containing screenshot, diagram, circle, line

Description automatically generated

Slika 3.6 Mreža uređaja i satova u NTP protokolu

Kreacija stabla NTP klijenata kreće od izvora tj. Sata s kojim se sinkroniziraju ostali uređaji. NTP se ne sinkronizira sa sustavima koji nemaju internu sinkronizaciju niti sa sustavima čije se vrijeme znatno razlikuje od većine drugih sustava. Protokol koristi nedeterminističku duljinu paketa podataka. Nominalna preciznost osigurana ovim protokolom je do 10 milisekundi u WAN (*wide-area networks*) do 2000 km te 1 milisekundi na LAN (*local-area networks*) mrežama. Što je korisnik udaljeniji od referentnog sata to je preciznost više određena latencijom veze, dok je pri manjim udaljenostima u hijerarhiji za preciznost najbitnija točnost referentnog sata. [21]

Postoje tri načina povezivanja u NTP protokolu, a to su klijent-server, *peer-to-peer* te *broadcast/multicast*.

Klijent-server način omogućava sinkronizaciju klijenata sa serverom te je najjednostavnija konfiguracija. Korisnik šalje zahtjev serveru te čeka na odgovor, nakon čega dobiva odgovor o preporučenom referentnom vremenu s kojim se treba ravnati te o njegovoj pouzdanosti. Serveri rade u grupi od 3 ili više servera što povećava robusnost ukoliko neki od servera ne radi ispravno ili izgubi vezu.

*Peer-to-peer* način koristi se tako da međusobno ravnopravni i povezani serveri na istoj razini si uzajamno služe kao sinkronizacija i zaštita. Svaki od njih je povezan s jednim ili više referentnih satova te ukoliko neki od njih izgubi vezu, ostali mu pružaju mogućnost oporavka. Koristi se u konfiguracijama u kojima je više servera na istoj razini sa sličnom ulogom.

*Broadcas*t način se koristi gdje zahtjevi za točnost nisu na visokoj razini. Potrebna je konfiguracija istog načina rada na svakoj pod mreži. Prednost ovog načina je da korisnici ne trebaju imati posebnu konfiguraciju za komunikaciju sa serverom, već se mogu konfigurirati osluškivanjem. Struktura povezanih uređaja može biti jednorazinska, hijerarhijska ili zvjezdasta.

Tipični NTP klijent u određenim intervalima provjerava vrijeme kod jednog ili više NTP servera. Klijent pritom računa vrijeme pomaka i vrijeme povratnog putovanja paketa.

Vrijeme odmaka računa se prema sljedećoj formuli:

()

gdje su:

t0 odlazno vrijeme na klijentu,

t1 dolazno vrijeme na serveru,

t2 odlazno vrijeme na klijentu,

t3 dolazno vrijeme na klijentu.

Vrijeme povratnog putovanja paketa računa se prema sljedećoj formuli:

()

gdje su oznake iste kao i na prethodnom primjeru.

### Simplified Network Time Protocol (SNTP)

Pojednostavljena verzija NTP-a puštena je u javnost nekoliko godina kasnije, da bi se u potpunosti spojila i ujedinila s NTP-om u njegovoj četvrtoj iteraciji (NTPv5, RFC 5905, 2010). Glavna razlika između ova dva protokola je način na koji SNTP obrađuje i čuva podatke te preciznost. [22]

SNTP koristi jednostavnije metode sinkronizacije te zamjenjuje matematičke izračune i statističke metode NTP-a čime osigurava znatno jednostavniju i lakšu implementaciju, čime se ostvaruje manja preciznost. SNTP protokol može se koristiti na uređajima unutar NTP mreže na mjestima gdje preciznost nije najbitnija karakteristika te na čvoru o kojem drugi sustavi u mreži ne ovise. SNTP također nema implementiranu provjeru za konzistentnost i točnost izvora, kao ni mehanizam za trajno čuvanje prethodnih podataka koje NTP koristi za retrospektivnu korekciju.

Verzije Windowsa 2000 i XP koristile su isključivo SNTP za sinkronizaciju vremena.

Jedan od problema SNTP i NTP protokola je u prelijevanju koje će se dogoditi u budućnosti zbog limitiranog broja bitova. Vrijeme koje se prikazuje kroz ove protokole prikazuje se u nepotpisanih (*unsigned*) 64 bita, gdje prva 32 bita prikazuju cjelobrojni dio, a drugih 32 bita realni dio, a vrijednost se mjeri u broju sekundi od 1.1.1900. godine. Upravo zato bi se 7.2.2036. Godine u 6 sati i 28 minuta trebao dogoditi preljev te resetirati brojač na 0. Novija verzija protokola zato koristi najvažniji bit (u ovom slučaju bit 0), kao oznaku radi li se o vremenu prije 2036. godine ili nakon. Tako se produžilo vrijeme mjerenja za još 68 godina, sve do 2104. godine.

### Precision Time Protocol (PTP)

PTP je nešto moderniji protokol, prvi put pušten u javnost 2002. godine, koji se aktivno koristi za sinkronizaciju satova u mreži računala. Na lokalnoj mreži (LAN) postiže preciznost u razini mikro sekunde što ga čini dobrim odabirom za kontrolne sustave gdje je preciznost vremena najbitnija stavka. Upravo iz tog razloga se koristi za sinkronizaciju financijskih transakcija, mobilnih tornjeva, podmorskih akcija i svih ostalih sustava koji nemaju stalni pristup satelitima.

Arhitektura samog protokola bazirana je na klijent-server načinu rada. Sustav distribucije sastoji se od jednog ili više elemenata za komunikaciju te jednog ili više sata. Običan sat (*Ordinary clock*) je uređaj s jednom vezom na mrežu te se koristi kao izvor ili kao odredište u sinkronizaciji. Granični sat (*Boundary clock*) ima više veza na mrežu i koristi se za sinkronizaciju između više elemenata mreže. Svaki element mreže ima svoj server preko kojeg se sinkronizira. Glavni izvor vremena naziva se *grandmaster* tj. glavni server, preko kojeg kreće sinkronizacija svih ostalih server elemenata. Jednostavnija verzija ovog algoritma ne koristi granične satove već su svi satovi direktno povezani na glavni sat i ostale elemente mreže.

U kasnijim verzijama algoritma (PTPv2) uvedene su i oznake vremena koje se šalju zajedno s porukama, koje modificiraju transparentni satovi (*Transparent clocks*), čije je vrijeme prilagođeno tako da je uračunato i vrijeme putovanja između elemenata mreže.

PTP koristi isto mjerenje vremena kao i *Unix* sustavi, koji kao referentnu točku i vrijednost 0 uzimaju 1. Siječanj 1970. Godine. Jedina razlika je što je PTP algoritam baziran na TAI (*International atomic time*), a ne na UTC-u (*Universal time*). Glavni server tako komunicira i izračunava razliku između ta dva sata (zbog prijestupnih sekundi) tako da se i UTC vrijeme može koristiti u PTP-u.

* 1. Problemi kod sinkronizacije

Sinkronizacija pomoću *Bluetootha* i sličnih bežičnih protokola predstavlja neke od problema te otežava potpunu točnost satova, za razliku od žičnog prijenosa gdje se analogni signal sata dvaju uređaja može uskladiti s velikom preciznošću. Najveći problem kod bežične sinkronizacije predstavljaju latencija i neodređeno trajanje određenih funkcija.

Iako su neki od prethodno navedenih protokola za komunikaciju višestruko puta brži od drugih i dalje se ne mogu mjeriti s gotovo trenutnim prijenosom pomoću žice. Samo pakiranje paketa i njihova priprema za slanje po točno određenim pravilima kao i sam proces putovanja koji raste s obzirom na udaljenost uređaja i samu okolinu uvijek ima određeno trajanje te se ne može spustiti na dovoljno malu vrijednost koja se može ignorirati. Kako bi potpuno uskladili satove potrebno je nadoknaditi to vrijeme putovanja i napraviti ‘odgodu’ sata uređaja koji šalje paket kako bi se akcije izvršile u isto vrijeme.

Drugi najveći problem, neodređeno trajanje određenih funkcija predstavlja gotovo nerješivu zapreku. Samo trajanje puta, kao i obrada ulaznog signala tj. poruke te izvršavanje određenih funkcija često varira zbog različitih stanja u kojima se mikrokontroler može nalaziti te zbog drugih zadataka koje trenutno izvršava. Nikad se u potpunosti ne može znati koliko će trajati iduća funkcija već samo pretpostaviti. Kod nešto složenijih sustava i programa često nailazi se na ostale zadatke s višim ili podjednakim prioritetom od trenutnog što može dodatno uvesti faktor nepredvidljivosti trajanja određenih funkcija.

Mnogi protokoli imaju i određenu propusnost te brzinu prijenosa. Ukoliko je potrebno poslati veću količinu podataka i sam prijenos će duže trajati. Unutarnji satovi različitih mikrokontrolera mogu raditi na različitim frekvencijama, što direktno znači i razlike u samoj točnosti unutarnjih satova i potencijalnom odstupanju istih. Čak i satovi čiji oscilatori rade na istim frekvencijama i koji imaju identične karakteristike mogu imati različita odstupanja s obzirom na okolinu, temperaturu i unutarnje faktore. U pravilu je vrlo teško postići jako precizna mjerenja bez korištenja vanjskih modula i veoma skupocjene opreme koja u većini slučajeva korištenja mikrokontrolera ili nije dostupna ili troši previše energije da bi se opravdalo njeno korištenje.

# Implementacija, spajanje i sinkroniziranje mikrokontrolera

* 1. Protokoli i sučelja

### ESP-IDF

Programiranje i rad s mikrokontrolerima ostvaruje se na nekoliko različitih načina. Mnogi proizvođači nude ekstenzije i dodatke za neke od najpoznatijih sučelja za programiranje.

Najkorišteniji su *Arduino* IDE i *PlatformIO* IDE (za *VSCode*) koji su polako kroz godine postali standardizirani zbog velikog broja podržanih mikrokontrolera i široke dokumentacije.

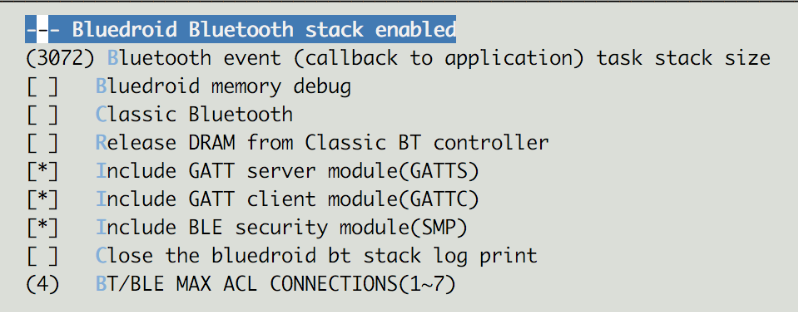
*Espressif* je uz podršku za već navedene platforme odlučio napraviti i svoje sučelje. Riječ je o ESP-IDF, službenom *IoT* sučelju za programiranje uređaja iz ESP32 obitelji (ESP32, ESP32-S, ESP32-C i ESP32-H). Uz podršku za programske jezike C i C++ te kroz Python skripte, ESP-IDF osigurava podršku za najjednostavnije i najkompleksnije aplikacije. Nije preporučeno za potpune početnike zbog relativne kompleksnosti i nedostatka “pravog” sučelja, jer se većina naredbi i opcija pozivaju kroz terminal. Vrlo je dobro dokumentiran alat s jako velikim brojem već gotovih primjera koji se odmah mogu primijeniti na stvarne uređaje.

Integrirana je i podrška za sve najkorištenije knjižnice i tehnologije kao što su RTOS-a, sučelja za komunikaciju, *Wi-Fi*, *Bluetooth*, mrežna sučelja, memorija, sigurnost i mnogi drugi. [23] Sučelje također dolazi kao dodatak za već spomenuti *VSCode* kao i za *Eclipse* IDE, što značajno olakšava njegovo korištenje. ESP-IDF pruža podršku za dva domaćinska stoga koja pružaju usluge uređaju, a to su *Bluedroid* i *Apache* *NimBLE*. Ove tehnologije pobliže su objašnjene u sljedećim poglavljima.

### Bluedroid

*Bluedroid* je implementacija Bluetooth tehnologije bazirana na Androidu te služi kao sučelje za nativno korištenje Bluetootha na Android uređajima, kao i ostalim uređajima koji dijele sličnu arhitekturu. Ono što je implementirano u ESP-IDF je zapravo omotač oko *Bluedroid* stoga adaptiran tako da ga mogu koristiti i uređaji ESP32 obitelji. Podržava cijeli set različitih funkcija te mnoge često korištene standarde. Ovo je standardni stog korišten kroz ESP-IDF.

*Bluedroid* kao HCI koristi VHCI, koji kroz ESP-IDF nudi sljedeće postavke.



Slika 4.1 Postavke *Bluedroid* stoga kroz ESP-IDF sučelje

### NimBLE

*NimBLE* je potpuno drugačiji pristup korištenju BLE tehnologije baziran na puno jednostavnijem i bržem stogu. Razvijen od tvrtke *Apache* prvi put se pojavljuje na tržištu 2018. godine te je u međuvremenu dobio i podršku za ESP32 uređaje od strane *Espressifa*. Kompatibilan je s *Bluetooth* 5 specifikacijama i podržava rad s mrežom BLE uređaja. Kao što je i spomenuto u prethodnim poglavljima, ne podržava rad *Bluetooth* *Classic* protokola.

Iako su potencijalne prednosti *NimBLE*-a naspram *Bluedroida* velike, kao što su brzina prijenosa i manja količina potrebnog prostora, za ovaj rad one nisu od presudne važnosti te se i kao baza koristio *Bluedroid* koji još uvijek pruža više funkcionalnosti i stabilniju vezu.

### Mjerenje vremena s ESP32

ESP32 uređaji imaju dva brojača koja služe za mjerenje vremena. Sustav omogućava simultano korištenje oba brojača ukoliko je potrebna veća preciznost i pouzdanost, no moguće je koristiti i samo jedan. Radi se o RTC (*Real-time clock*) brojaču i brojaču visoke rezolucije. RTC se koristi za mjerenje vremena za cijelo vrijeme rada uređaja, čak i u tihom načinu rada. Resetiranje uređaja i potpuni gubitak energije će resetirati ovaj brojač. Njegova preciznost je 6.67 mikro sekundi. RTC može mjeriti vrijeme na nekoliko izvora koji se koriste ovisno o načinu radu mikrokontrolera.

Unutarnji 150 kHz RC oscilator osnovni je izvor koji je ovisi ostalim komponentama i pruža najnižu potrošnju. Stabilnost frekvencije ovisi o temperaturi te jesu moguća odstupanja za vrijeme tihog načina rada.

Vanjski 32 kHz kristal koristi vanjski kristal kojeg je potrebno spojiti na dva pina. Njime se postiže stabilna frekvencija uz nešto malo veću potrošnju u tihom načinu rada (1 mikro amper). Ukupna potrošnja koristeći ovaj način vrlo je niska s obzirom na dobivenu preciznost. [24]

Vanjski 32 kHz oscilator omogućava uređaju da koristi signal generiran od nekog vanjskog sklopovlja po kojem onda ravna svoji sat. Interni 8.5 oscilator pruža stabilniju frekvenciju od osnovnog načina rada, ne zahtjeva vanjske komponente, ali troši nešto više energije u tihom načnu rada (5 mikro ampera).

Brojač visoke rezolucije nije moguće koristiti za vrijeme načina tihog rada, no njime se postiže viša preciznost od 1 mikro sekunde. [25]

Osim već spomenutog problema prelijevanja NTP/SNTP protokola, računanje *Unix* vremena također ima svoje implementacije. Zbog korištenja 32-bitne reprezentacije broja, prelijevanje tog brojača trebalo bi se dogoditi 2038. godine., poznatije kao Y2K38 problem. [26] ESP-IDF je u verziji v5.0 riješio problem tako da je povećao varijablu dohvata *Unix* vremena na 64 bita, što produžuje mogućnost mjerenja vremena za još 292 milijarde godina.

### Prekidne rutine

Kako bi ostvarili željenu sinkronizaciju potrebno je poznavati ponašanje prekida na ESP32 uređajima te kako ih postaviti. Većina uređaja ima 2 jezgre od kojih svaka ima 32 prekidna polja od kojih svako ima određeni prioritet obrade. Reakcija prekida na neki događaj ili signal izuzetno je brza s određenim postavkama može se spustiti ispod 1 mikro sekunde. [27]

Prekide po uzrocima dijele se na one vanjske, izazvane od periferije, i na unutarnje, izazvane od strane samog procesora poput isteka brojača. Postoje minimalne razlike u obradi tih prekida. Svi dostupni GPIO pinovi na ESP32 uređajima mogu se koristiti kao prekidni pinovi.

Prekidi se definiraju s uzrokom prekida, prekidnom funkcijom i načinom rada. Način rada razlikuje se za vanjske prekide koji se pozivaju direktno preko pinova za sve moguće kombinacije promjena stanja pina (visoko, nisko, na promjenu, padajući, rastući).

Sve prekidne funkcije izvode se bez konteksta i argumenata te su smještene u internu RAM memoriju sustava kako bi njihovo izvođenje bilo što brže. Prekidne funkcije bi također trebale obavljati posao u najkraćem mogućem roku te vratiti pravo glavnom programu.

Korištenjem FreeRTOS operativnog sustava prekidne rutine, kao i mnoge druge pod funkcije sustava, obrađuje se kroz zadatke (*task*). Samim pokretanjem sustava kreira se 5 osnovnih grupa zadataka:

Besposleni zadatci (*Idle Tasks*) - po jedan IDLEx zadatak za svaki CPU

*FreeRTOS* zadatak brojača (*FreeRTOS Timer Task*) - zadatci koji se izvršavaju u slučaju korištenja brojača

Glavni zadatak (*Main task*) - zadatak koji zove glavnu funkciju programa app\_main

IPC zadatci (*IPC Tasks*) - koriste se za zvanje unutar procesorskih naredbi kada je omogućena dvojezgrenost sustava

ESP zadatak brojača (*ESP Timer Task*) - zadatci korišteni za povratne pozive ESP internih brojača

Nužno je osigurati siguran i brz povratak u glavni zadatak kao i brisanje konteksta svih zadataka koji se više neće koristiti. [28]

Dodatni zadatci mogu se dodavati i micati po potrebi te im se može odrediti određeni prioritet. Broj prioriteta nije ograničen i može se postaviti kroz varijablu *configMAX\_PRIORITIES*, no svaki definirani prioritet zauzima dio RAM memorije te je preporučeno koristiti samo onoliko prioriteta koliko je stvarno potrebno. Na svaki prioritet može se staviti neograničen broj zadataka. [29] Besposleni zadatak ima početni prioritet 0 koji je ujedno i najniži prioritet. Zadatci s najvišim prioritetom uvijek se prvi izvršavaju. Ukoliko je više zadataka postavljeno na isti prioritet, dijele procesorsko vrijeme ukoliko drugačije nije definirano.

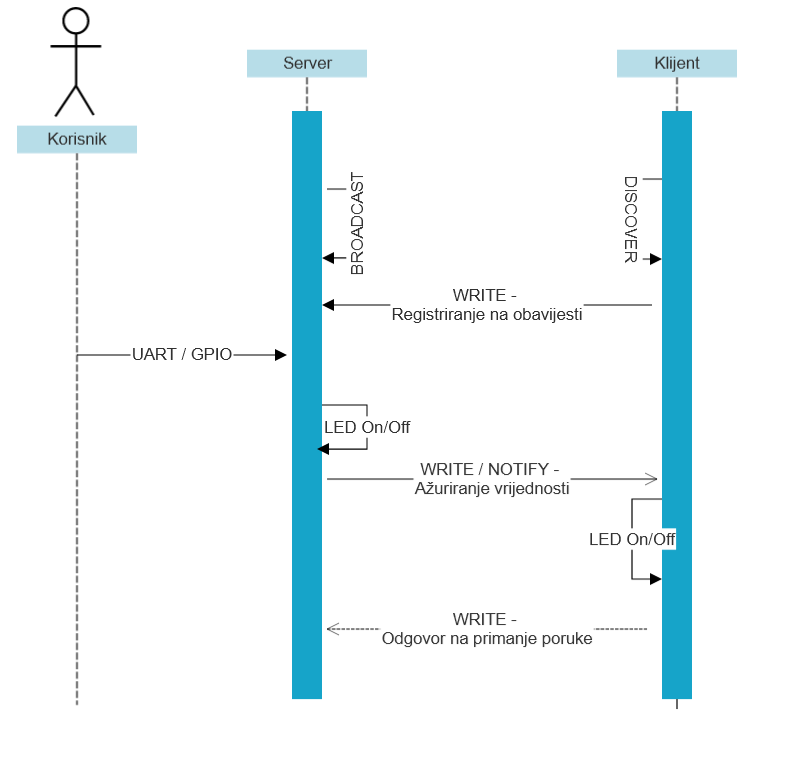
Osnovne vrijednosti prioriteta predefiniranih zadataka su sljedeće:

* Glavni zadatak (*app\_main*) – 1
* Brojač visoke rezolucije (*High resolution timer*) – 22
* F*reeRTOS* zadatak brojača – 1
* Događaji (*Event Handling*) – 20
* *Wi-Fi* – 23
* Bluetooth kontroler – 23
* *NimBLE Bluetooth* – 21
* *Bluedroid Bluetooth* – 22
  1. Sinkronizacija uređaja

### Korišteno sučelje i postavke okoline

Za samu sinkronizaciju unutar koda korišteni su isključivo algoritmi direktne sinkronizacije preko BLE koristeći *Bluedroid Bluetooth* stog. Sva komunikacija inicirana je od strane servera prema klijentu koji vraća određene informacije prema serveru u obliku teksta. Za razvoj je korišteno ESP-IDF sučelje.

Server prilikom promjene vrijednosti karakteristike koristi NOTIFY događaj čime obavještava sve spojene uređaje o promjeni. Klijent odgovara WRITE događajem i povratnom porukom prema serveru. Kako bi se lakše upravljalo događajima koristi se tipkovnica preko koje se kontrolira serverom i klijentom različitim tipkama. Svaki pritisak tipke označava se kao prekid visokog prioriteta unutar programa. Za testiranje ovog koda potrebno je uređaje spojiti na računalo radi napajanja i povezanosti s tipkovnicom. Moguće je izbjeći povezivanje s računalom spajanjem uređaja na vanjsko napajanje i dodavanjem nekoliko prekidača koji će na isti način upravljati uređajima. U **Dodatku A** nalazi se detaljno objašnjenje cjelokupnog koda i naredbi koje se mogu pozivati preko tipkovnice. Sinkronizacija se prikazuje pomoću LED svjetla na svakom uređaju koja služe kao simulacija za neke druge radnje poput prikupljanja dinamičnih podataka ili aktiviranje nekog složenijeg sklopovlja koje je primjerenije stvarnim situacijama.



Slika 4.2 Osnovna komunikacija ESP32 servera i klijenta

U poruci koja se distribuira sa servera na klijent nalazi se informacija o slovu koje je pritisnuto na tipkovnici. Klijent na temelju te informacije radi određene akcije te vraća sljedeće informacije:

* Vrijeme obrade poruke - tp = tsp - tss
* Trenutno vrijeme globalnog brojača klijenta - vrijeme od pokretanja uređaja u mikro sekundama
* Trenutno vrijeme lokalnog brojača za periodičko paljenje LED svjetala - vraća vrijednost paljenja LED svjetla kod odgovora na poruku ukoliko brojač još nije pokrenut

Za vrijeme obrade poruke koristi se formula:

()

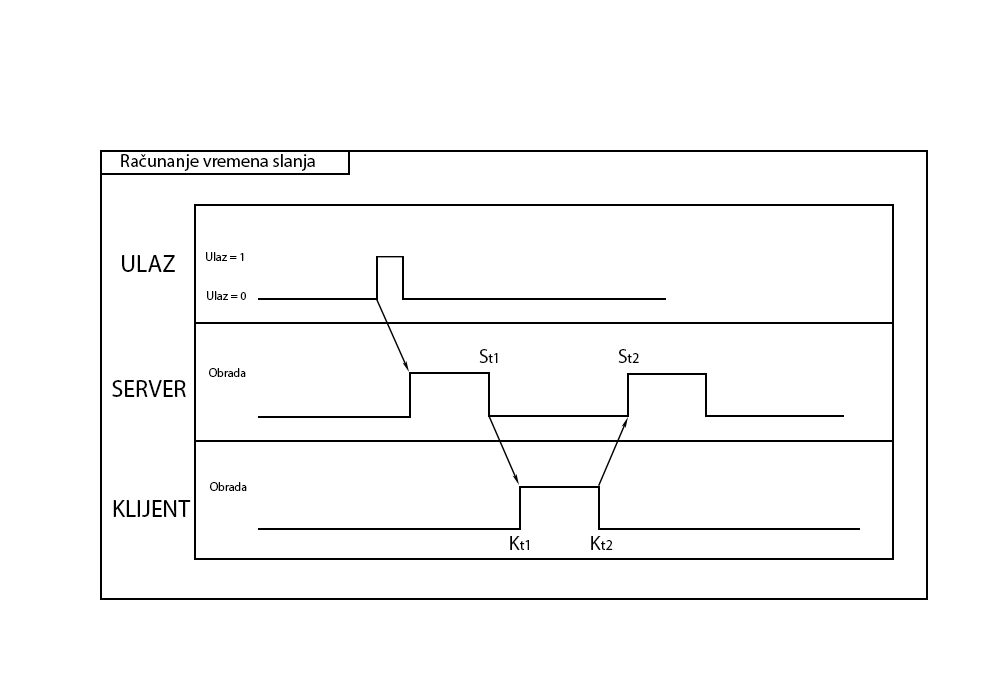
gdje su:

*Tp* vrijeme obrade poruke na klijentu,

*Tsp* vrijeme pristizanja poruke sa servera na klijent,

*Tss* vrijeme slanja odgovora s klijenta na server.

Pomoću ovih informacija na serveru računa se vrijednost trajanja putovanja (od slanja poruke prema klijentu do primanja poruke s klijenta) i razlike između dva globalna sata. Ukoliko su uključeni brojači za periodičko upravljanje LED svjetlima, računa se i njihova razlika.



Slika 4.3 Vremenski dijagram slanja poruke

Ukupno trajanje puta poruke, pod pretpostavkom da je put simetričan, računa se po sljedećoj formuli:

()

gdje su:

*St1* odlazno vrijeme na serveru,

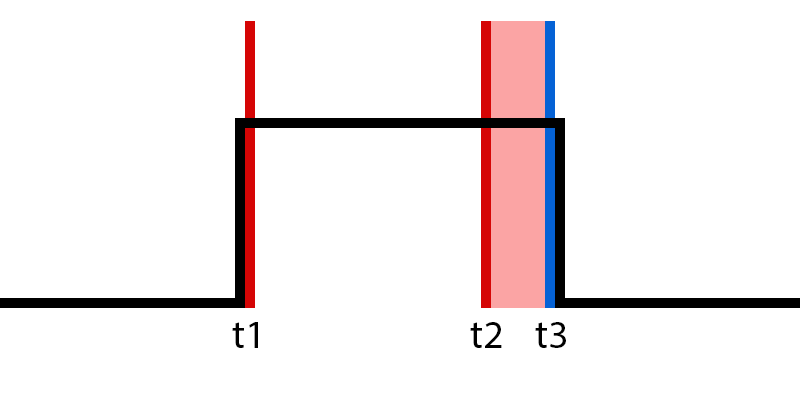
*St2*dolazno vrijeme na serveru,

*Kt1* dolazno vrijeme na klijentu,

*Kt2*odlazno vrijeme na klijentu.

Server dobiva vrijednost trajanja obrade na klijentu te oduzima tu vrijednost od vrijednosti ukupnog trajanja puta. Ovu se vrijednost koristi u odmaku kod sinkronizacije. Ukoliko je poznata informacija koliko je vremena potrebno od slanja poruke sa servera do odrađivanja određene akcije na klijentu, za tu vrijednost može se odgoditi odrađivanje akcije na serveru.

Vrijednosti brojača koje se šalju s klijenta na server potrebno je pretvoriti u tekstualni oblik kako bi mogle biti poslane. Konverzija broja visoke preciznosti (*int64\_t*) u polje znakova (*character array*) ima određeno trajanje. Kako bi se nadoknadilo izgubljeno vrijeme od uzimanja vrijednosti brojača, potrebno je unaprijed izračunati prosječno vrijeme trajanja funkcije konverzije te ga nadodati na pretvoreno vrijeme.



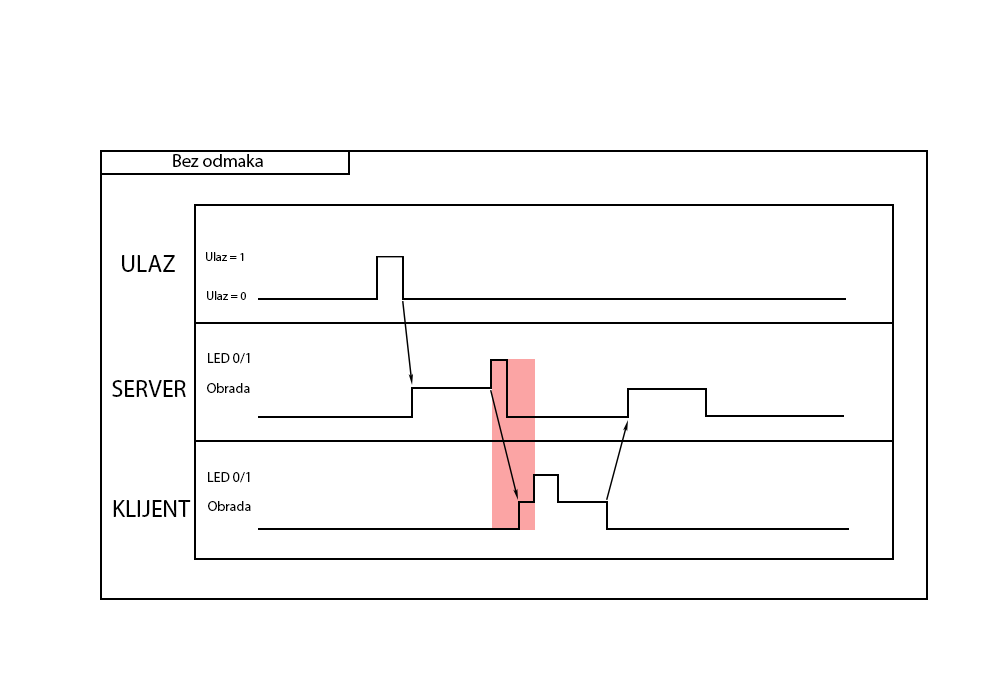
Slika 4.4 Skica odmaka slanja poruke

Dohvaćanje vrijednosti brojača uređaja funkcijom *esp\_timer\_get\_time()* iznosi oko 2 mikro sekundi što je u zanemariva vrijednost s obzirom na činjenicu da je okvirni cilj postizanje sinkronizacije u razini nekoliko milisekundi.

Vrijeme pretvorbe jedne brojčane vrijednosti u polje znakova iznosi između 15 i 20 mikro sekundi, a vrijeme postavljanja poruke s informacijama brojača oko 90 mikro sekundi.

### Sinkronizacija bez odmaka

Radi osnovnog postavljanja i rješavanja problema, napravljen je primjer u kojem uređaji nemaju implementiran nikakav sustav sinkronizacije. Radi se o jednostavnoj komunikaciji prilikom koje na signal server pošalje poruku prema klijentu i neposredno nakon upali/ugasi LED svjetlo. Klijent prilikom primanja poruke odmah napravi isto. Time je dobivena osnova slanja i primanja poruka.

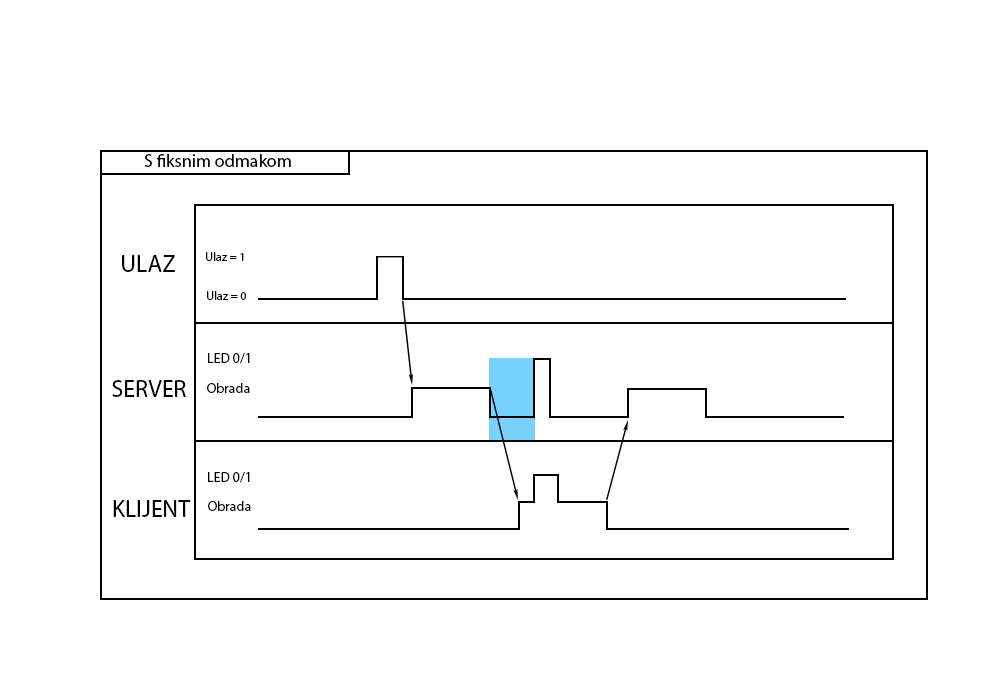


Slika 4.5 Sinkronizacija bez odmaka

U postavljenom okruženju u kojem uređaji nemaju nikakve druge zadatke osim međusobne komunikacije preko BLE protokola ustanovljen je odmak paljenja LED svjetala od 20 do 30 milisekundi. Cijelo putovanje poruke traje oko 50 milisekundi. Vrijednosti su okvirne te nisu uvijek iste - ponekad samo putovanje traje puno duže i odmak je mnogo duži zbog konteksta i raznih drugih procesa koji se odvijaju u uređajima (npr. punjenje memorije, veće vrijednosti internih brojača, …).

### Sinkronizacija s fiksnim odmakom

Najjednostavniji način prividne sinkronizacije ostvaruje se ubacivanjem fiksnog odmaka prilikom aktiviranja akcije na serveru. Prethodno izračunata okvirna vrijednost trajanja putovanja i aktiviranja iste akcije na klijentu uzima se kao referentna vrijednost i služi za stvaranje prividne sinkronizacije.

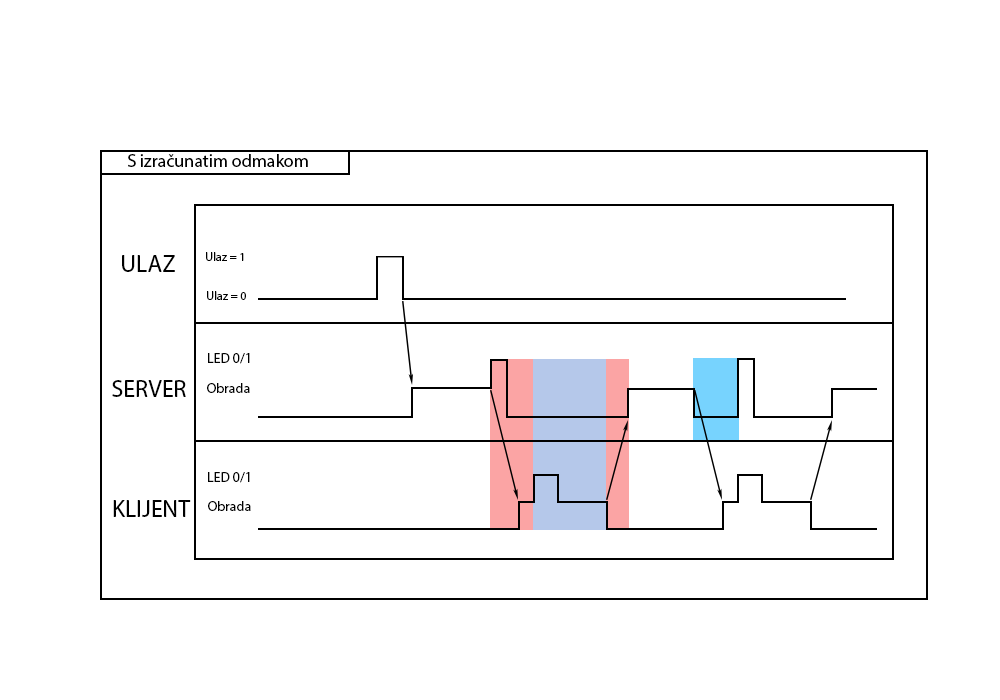


Slika 4.6 Sinkronizacija s fiksnim odmakom

Glavni problem ove metode je što je trajanje slanja poruke kao i obrade na uređajima često dinamičan proces čije trajanje ovisi o nekoliko faktora i nije konstantno. Vrlo lako se može dogoditi da je pogođena gotovo savršena sinkronizacija, ali isto tako i da se akcija na klijentu izvede prije akcije na serveru, što je manje vjerojatno zbog činjenice da trajanje slanja poruke ima minimalno trajanje. Češći slučaj je da se slanje poruke ili njena obrada oduži te da uređaji tj. akcije uopće nisu sinkronizirane već imaju poveći odmak. Ova metoda je korisna kad nam dinamičko računanje trajanja nije opcija ili kada je sigurno da se vrijeme trajanja puta i obrade značajno ne mijenja, što može biti slučaj kod nešto jednostavnijih sustava koji nemaju druge zahtjevne poslove u istovremenom izvođenju. Prednost ove metode je manjak potrebe za dinamičnim računanjem i smanjivanje kontrolnih poruka između uređaja.

### Sinkronizacija s dinamičnim odmakom

Na sličan način može se koristiti i metoda s dinamičkim odmakom. Što se tiče samog odmaka i razine točnosti sinkronizacije, barem inicijalno, ova dva pristupa su gotovo identična. Glavnu razlika je u činjenici da se pri svakoj komunikaciji između servera i klijenta trajanje putovanja i obrade dinamično računa te se uvijek koriste posljednje vrijednost trajanja kao referentna vrijednost odmaka akcije na serveru.



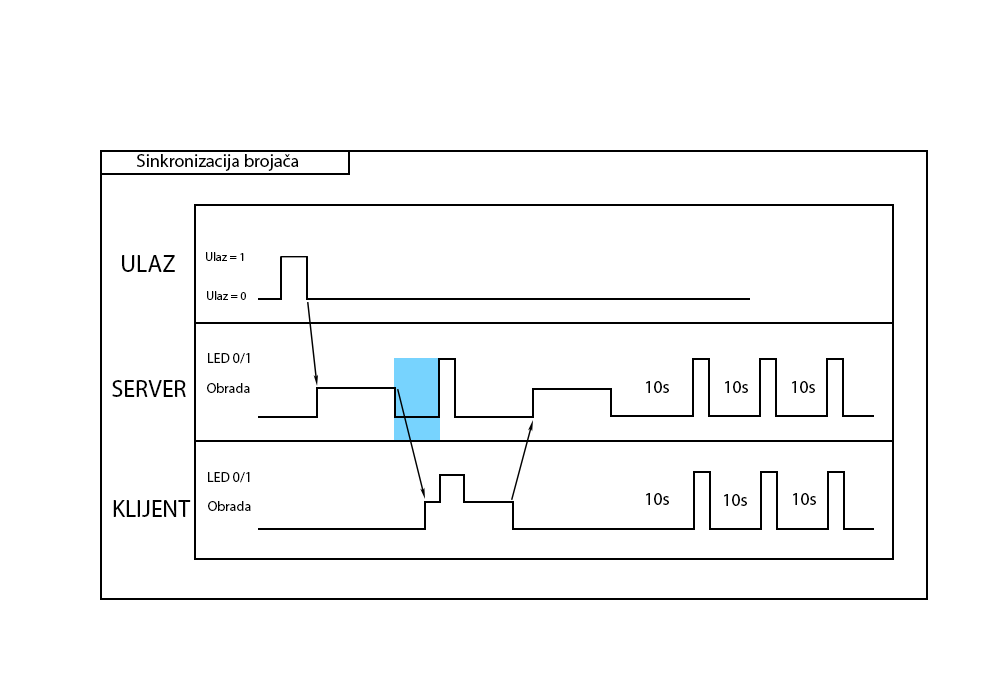
Slika 4.7 Sinkronizacija s izračunatim dinamičnim odmakom

Time se postiže realnija vrijednost odmaka te ukoliko uređaj radi sporije zbog drugih aktivnosti u tom vremenskom periodu, omogućuje da je preciznost sinkronizacije veća. Nedostatak ove metode je da u slučajnu privremenog ili trenutnog usporenja još uvijek je moguće dobiti neprecizni odmak. Pošto se odmak računa isključivo preko posljednje poruke, ukoliko je prošlo puno vremena između dvije poruke, mogućnost dobivanja krive vrijednosti je veća. Taj problem se može riješiti slanjem kontrolne poruke neposredno prije slanja prave poruke čime se ponovno računa odmak i koristi novija vrijednost za odmak. Nedostatak ove metode je puno veći broj poruka i kontrolnih poruka naspram prethodnog slučaja, što znači veću potrošnju energije i druge potencijalne probleme veze. Glavna prednost je što bi u većini slučajeva, pogotovo kod kompliciranijih sustava, sinkronizacija trebala biti puno preciznija.

### Sinkronizacija pomoću brojača

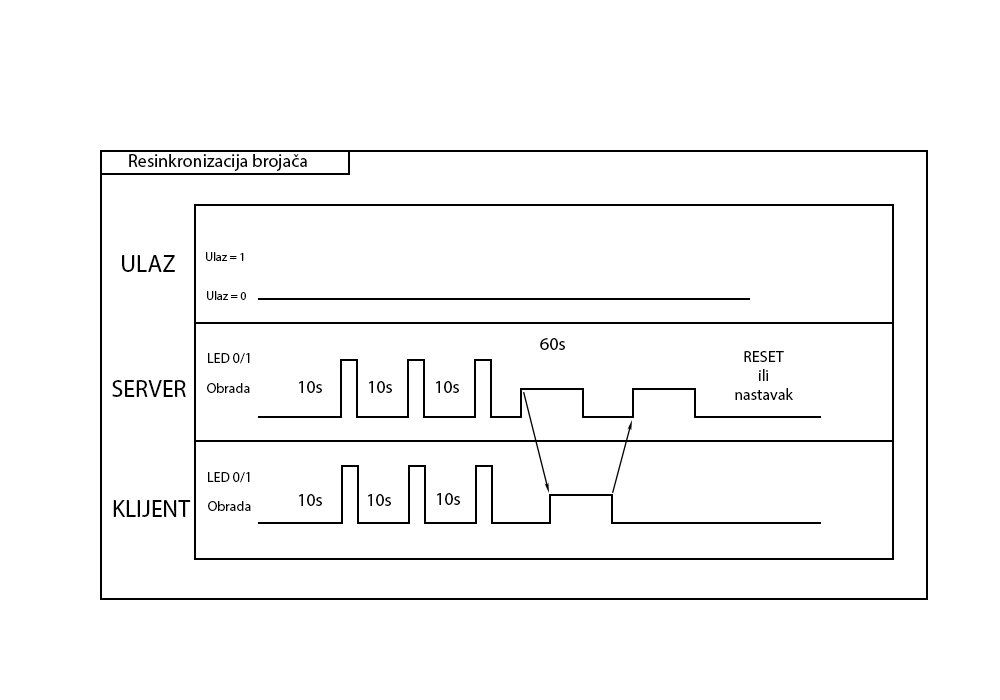
Glavni način sinkronizacije uređaja, ukoliko je korisno imati akcije u određenim intervalima, je pomoću internih brojača uređaja. Koristeći prethodno definiranu formulu za računanje odmaka, server šalje poruku klijentu te aktivira brojač nakon definiranog odmaka. Time se interni brojači na svim uređajima pokreću u okvirno isto vrijeme. Ukoliko se na uređajima koriste satovi iste preciznosti i frekvencije, njihovo odbrojavanje i izvršavanje trebalo bi biti u gotovo identičnim intervalima. Zbog visokog stupnja prioriteta prekidnih potprograma brojača moguće je ostvariti sinkronizaciju vrlo visoke preciznosti, pogotovo u sustavima s manjim brojem drugih zadataka visokog prioriteta.

Brojač na ESP32 uređaju moguće je definirati ili kao pravi brojač ili preko zadatka koji odgađa svoje izvršavanje pomoću funkcije vTaskDelay() u određenim intervalima. Pravi brojač koristi prekidnu rutinu visokog prioriteta po isteku te automatski izvodi određenu funkciju (ili zadatak). Zadatcima je s druge strane moguće odrediti određeni stupanj prioriteta i zadržati određeni kontekst kojeg prekidne rutine nemaju. Mijenjanje intervala odbrojavanja tj. čekanja moguće je odraditi u oba slučaja.



Slika 4.8 Sinkronizacija pomoću brojača

Glavni nedostatak ove metode je što brojači odbrojavaju samo po unaprijed definiranoj vrijednosti. Prije pokretanja svakog brojača potrebno je odredit interval prekida. Zadatci se konstantno izvršavaju te je njihovo izvršavanje prekinuto određenim vremenskim intervalom ili postavljanjem zastavica. Kako bi se koristile prave prednosti ovog pristupa, što su mali broj poruka sinkronizacije između uređaja i mogućnost njihove nepovezanosti, potrebno je unaprijed definirati te intervale. Ovom metodom uređaji se mogu koristiti izvan dometa komunikacije čime će se još uvijek zadržati visok stupanj sinkronizacije.



Slika 4.9 Resinkronizacija pomoću brojača

Sustav za sinkronizaciju uveden je i pokreće se u određenim intervalima (npr. svakih 60 sekundi) na isti način kao i u prethodnim primjerima. Klijent kao odgovor u poruci šalje vrijednost posljednjeg vremena internog brojača u trenutku određene akcije (paljenja/gašenja LED svjetla) čije se razlika s internim brojačem servera uspoređuje s njihovom razlikom prilikom pokretanja brojača. Ukoliko ta razlika nije ista ili odstupa više od očekivanog, brojači se zaustavljaju i ponovno pokreću nakon ponovnog računanja odmaka. Tijekom dugoročnog testiranja na jednostavnijem programu brojači su ostali unutar nekoliko mikro sekundi te nikad nisu odstupali više od 50 mikro sekundi, što se događalo u jako rijetkim slučajevima. Korištenje brojača mnogo je konzistentnije i sigurnije od mjerenja vremena preko poruke.

Ukoliko je potrebno svesti komunikaciju na minimum ili ju u potpunosti ukloniti, postoji mogućnost aktiviranja provjere brojača i resinkronizacije samo na signal tj. kada je komunikacija moguća za razliku od određenih intervala. Time je komunikacija i potrošnja energije svedena na apsolutni minimum dok je žrtvovana provjera brojača u čestim intervalima.

* 1. Mjerenje odmaka

### Računski

Najjednostavniji način mjerenja stvarnog odmaka je programski. Korištenjem knjižnice brojača visoke preciznosti (ESP *Timer)* [31] prati se vrijeme od samog pokretanja uređaja s *double double* preciznošću (int64\_t) te razine mikro sekundi. Pozivom na funkciju esp\_timer\_get\_time dohvaća se trenutna vrijednost glavnog brojača. Sam poziv funkcije traje nekoliko mikro sekundi te je zanemariv s obzirom na željenu razinu preciznosti koja je 1000 puta veća.

Dohvaćanjem vrijednosti brojača u određenim trenutcima moguće je relativno precizno odrediti duljine trajanja pojedinih radnji. Glavni problem u ovom obliku mjerenja, kao što je već i prije napomenuto u radu, je samo vrijeme pripremanja poruke te duljina trajanja obrade prilikom slanja i primanja tj. prije nego se uspije izmjeriti ikakva vrijednost brojača. Te vrijednosti mogu se zanemariti ili unaprijed izračunati pa dodati izmjerenoj vrijednosti brojača, no s obzirom na njihovu kratkotrajnost, razlike u završnim mjerenjima gotovo i nema.

Još jedan vrijedan mehanizam procjene preciznosti sinkronizacije je mjerenje odmaka unutarnjih satova povezanih uređaja. Na taj način lakše se može pratiti kolika je zapravo njihova vremenska udaljenost te koliko je blizu razina vremenske sinkronizacije. U tim mjerenjima teško je izbjeći male odmake i nekonzistentnosti trajanja određenih funkcija.

Najtočnija metoda mjerenja ovim pristupom je prilikom pozivanja periodičkog brojača. Ukoliko je poznata okvirna vrijednost udaljenosti između dva brojača, svaka sljedeća aktivacija istih može se usporediti po lokalnoj vrijednosti te tako vidjeti je li se razlika između njihove aktivacije smanjila ili povećala. Za svako mjerenje uzet je uzorak od 20 izmjerenih podataka.

Rezultati mjerenja po različitim parametrima su sljedeći (sve vrijednosti su u milisekundama):

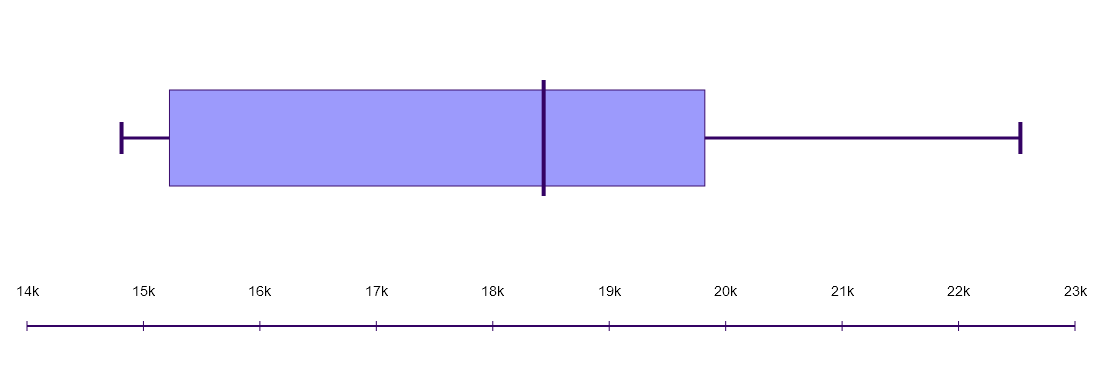
|  |  |  |  |
| --- | --- | --- | --- |
| Vrijednosti su u milisekundama | **Sinkronizacija bez odmaka** | **Sinkronizacija s fiksnim odmakom** | **Sinkronizacija s dinamičnim odmakom** |
| **Prosjek** | 17.959 | 5.138 | 5.069 |
| **Medijan** | 18.437 | 4.817 | 4.479 |
| **Standardna devijacija** | 2.446 | 3.170 | 1.973 |
| **MIN** | 14.812 | 527 | 2.401 |
| **MAX** | 22.531 | 9.910 | 8.785 |

Tablica 4.1 Prosječne vrijednosti odmaka sinkronizacije pomoću video mjerenja

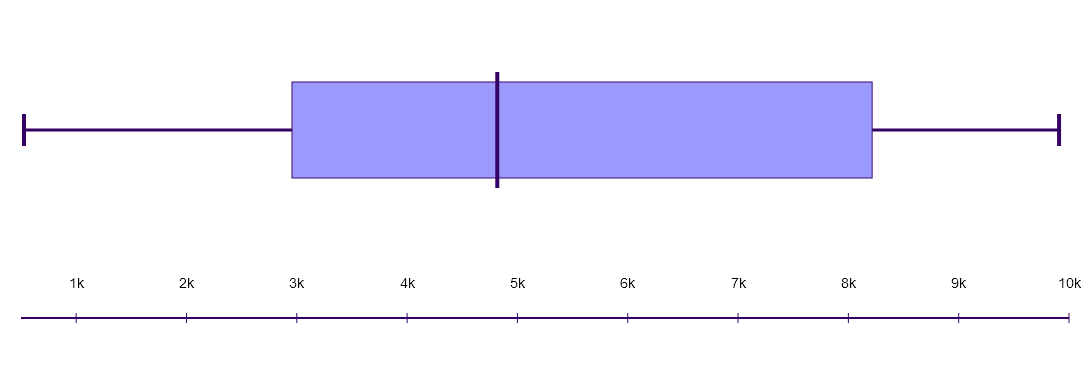
Prilikom korištenja sinkronizacije pomoću brojača, koristi se dinamički odmak pri inicijalnoj sinkronizaciji. Sva daljnja provjera sinkronizacije sastoji se od provjera vremena u kojem su brojači aktivirali akciju na svim uređajima. Usporedba se radi s referentnom tj. početnom vrijednošću, koja je u istim intervalima kao i vrijednosti dinamičnog odmaka u tablici.

Interni brojači su izuzetno precizni i tijekom testiranja nije dolazilo do resinkronizacije u vrijednosti tolerancije iznad 500 mikro sekundi. Češća resinkronizacija (u više od 50% slučajeva) događala se tek kada bi se taj interval spustio na stupanj razine 10 mikro sekundi ili manje

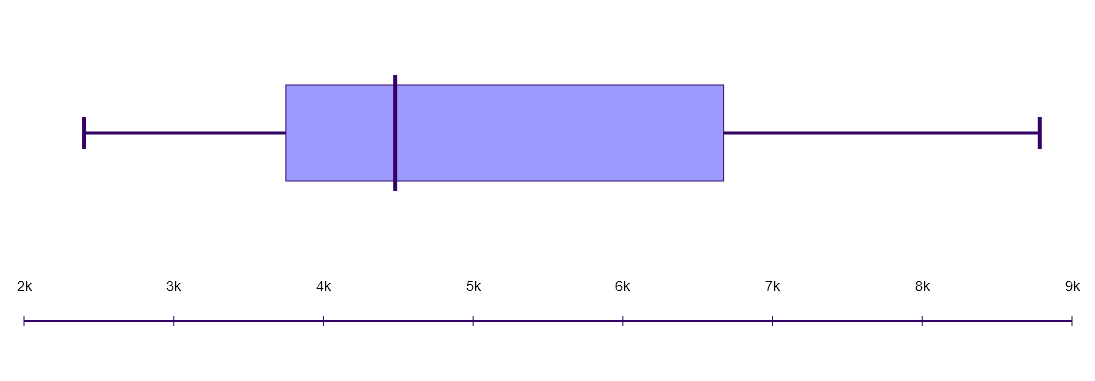
Treba uzeti u obzir da mikrokontroleri nisu obavljali nikakve druge radnje te da bi rezultati potencijalno bili nešto drugačiji s kompleksnijim zadatcima gdje su očekivana i odstupanja samih brojača. Preciznost sinkronizacije se ne bi promijenila, već bi se ponovno primijenio algoritam za dinamično računanje odmaka pri ponovnom pokretanju brojača. Mehanizam brojača osigurava da se ta vrijednost pretjerano ne udaljava od referentne. Ukoliko bi početni signal sinkronizacije bio visoke preciznosti između 0,5 i 1 milisekundi, s gotovo 100%-tnom sigurnošću moglo bi se garantirat očuvanje te razine sinkronizacije.



Slika 4.10 Kvantilni graf odmaka bez sinkronizacije



Slika 4.11 Kvantilni graf odmaka sinkronizacije s fiksnom vrijednošću



Slika 4.12 Kvantilni graf odmaka sinkronizacije s dinamičnom vrijednošću

Iz prethodnih grafova koji predstavljaju kvantile prosječnih vrijednosti, mogu se iščitati određeni podatci. Sva mjerenja rađena su u istim uvjetima s uzorkom od 20 ponavljanja. Kvantili predstavljaju najčešće izmjerene vrijednosti. Vanjske oznake označuju ekstreme od srednje vrijednosti (vertikalna linija), dok je osjenčano područje oznaka 50% svih vrijednosti, od kojih je 25% u negativnom i 25% u pozitivnom od navedene srednje vrijednosti.

Primjećuje se kako je prosječan raspon najčešćih vrijednosti kašnjenja gotovo dvostruko veći kod fiksnog odmaka nego kod dinamičnog, što je u skladu s podatcima standardne devijacije. Može se zaključiti kako iako je moguće postići približno podjednaku razinu sinkronizacije koristeći oba načina, dinamičnim računanjem udaljenosti osigurava se konzistentnost i manje odstupanje od prosječne vrijednosti. Postignuta razina sinkronizacije u prosjeku iznosi oko 5 milisekundi odmaka jednog uređaja od drugog.

### Kamera

Kao jedan od načina provjera latencije korištena je video snimka u velikom broju sličica u sekundi. *Google Pixel* 6 korišten je kao uređaj za snimanje u *super slow-motion* načinu rada kamere u 240 sličica po sekundi. Svaka sličica tako označava 4.16 milisekundi stvarne snimke te pomoću gledanja individualnih sličica može se izračunati približna latenciju s varijancom pogreške od 1 sličice tj. 4.16 milisekundi.

Vrijednosti broja sličica i vremena su okvirni zbog nedefiniranosti stanja “upaljenog” LED svjetla - na nekoliko ekrana primjećuje se kako je svjetlo u među stanju te je teško napraviti preciznu procjenu kad je akcija paljenja/gašenja točno započela.

Iako mjerenje pomoću video snimke nije uobičajeno i ne daje nam dovoljno precizne rezultate, služi kao dobar način potvrde rezultata mjerenja unutar uređaja te je primjerenije stvarnim slučajevima sinkronizacije.

Rezultati video mjerenja su sljedeći:

|  |  |  |
| --- | --- | --- |
| Algoritam | **Broj sličica između paljenja dva LED svjetla** | **Vremenski razmak između paljenja dva LED svjetla** |
| **Sinkronizacija bez odmaka** | 6 | 16.64 – 25 ms |
| **Sinkronizacija s fiksnim odmakom [[1]](#footnote-1)** | 2 | 0 – 8.32 ms |
| **Sinkronizacija s dinamičnim odmakom** | 3 | 4.16 – 12.48 ms |
| **Sinkronizacija pomoću brojača [[2]](#footnote-2)** | 3 | 4.16 – 12.48 ms |

Tablica 4.1 Rezultati preciznosti sinkronizacije pomoću video mjerenja

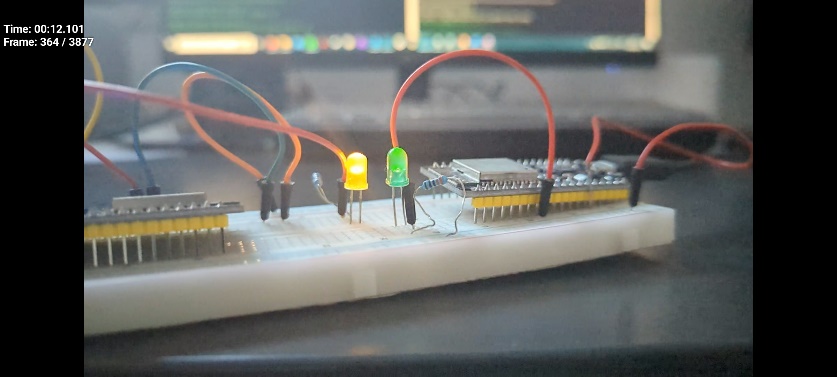
Uzimajući u obzir da jedna sličica predstavlja interval od 4.16 milisekundi, moguće je da se promjena dogodi u bilo kojem trenutku tog intervala, stoga je nemoguće definitivno odrediti vrijednost točniju od tog intervala. S obzirom da je očekivano kašnjenje otprilike u istoj razini preciznosti, ova metoda služi kao potvrda prethodno provedenih mjerenja putem direktnog zapisivanja vrijednosti iz mikrokontrolera. Jedan od izazova prilikom mjerenja je i teško određivanje trenutka završetka paljenja/gašenja LED svjetla zbog među razina tih stanja koje se mogu postići.

A circuit board with wires connected to it

Description automatically generated with low confidenceA picture containing electronics, electrical wiring, electronic engineering, cable

Description automatically generatedA circuit board with wires connected to it

Description automatically generated with low confidenceA circuit board with wires connected to it

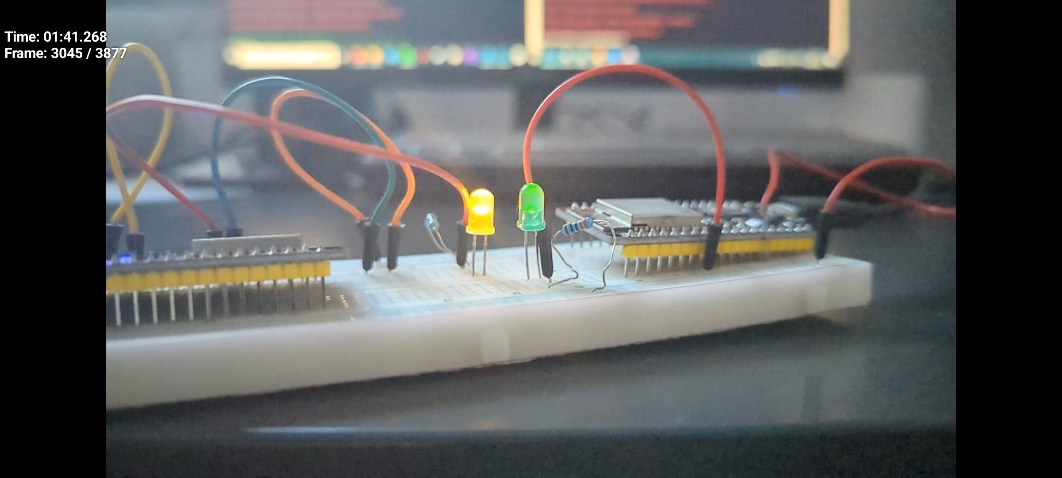
Description automatically generated with low confidence

Slika 4.13 Pojedinačni okviri snimke bez sinkronizacije

A picture containing electronics, electrical wiring, cable, electronic engineering

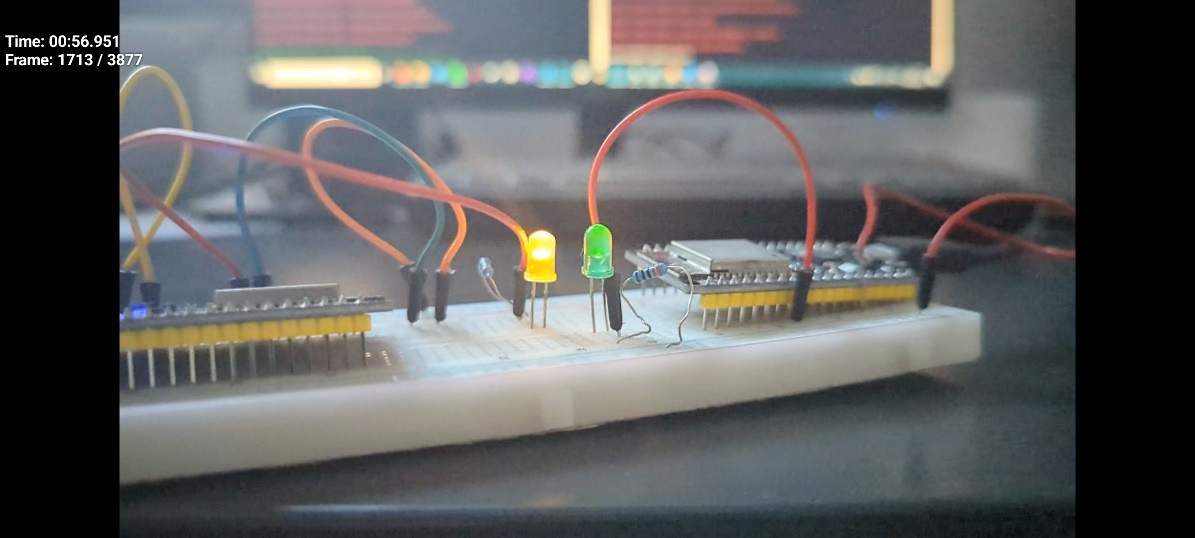
Description automatically generatedA circuit board with wires connected to it

Description automatically generated with low confidence



Slika 4.14 Pojedinačni okviri snimke s dinamičnom sinkronizacijom

A close up of a circuit board

Description automatically generated with low confidence

Slika 4.15 Pojedinačni okviri snimke s fiksnom sinkronizacijom

# Daljnji rad i razmatranje

Glavni problemi primijećeni tijekom razvijanja i testiranja odabranog rješenja nemogućnost su preciznijeg određivanja trajanja komunikacije i određenih funkcija obrade poruke. S obzirom na određenu fluktuaciju navedenih vrijednosti, nije bilo moguće konstruirati točniji algoritam koji bi se temeljio na preciznijim vrijednostima.

S obzirom na nedostatak preciznije opreme za mjerenje, točna udaljenost od savršene sinkronizacije nije mogla biti izmjerena. Metode korištene umjesto osciloskopa pružaju dovoljno dobar indikator preciznosti, pogotovo ako se uzme u obzir da se ta vrijednost nekonzistentna s obzirom na prethodno navedena odstupanja.

Kako bi se pobliže istražio ovaj problem sinkronizacije, u budućim radovima moguće je nadograditi ovo rješenje na nekoliko načina. Umjesto *Bluedroid* stoga BLE protokola moguće je koristiti *NimBLE* stog, čime bi se potencijalno smanjila latencija i potrošnja same komunikacije. Uzimajući u obzir da se i dalje radi o istom protokolu, teško je za očekivati da bi se mogla postići puno bolja preciznost, ali bi se mogle uočiti prednosti i mane korištenja ova dva stoga u različitim okolnostima. Isto tako moguće je implementirati i *ESP-NOW* protokol koji je mnogo jednostavniji i brži od BLE protokola. Glavna mana ovog protokola je potrošnja energije, no s obzirom da se radi o relativno novijem protokolu, korištenje istog u stvarnim slučajevima još nije dovoljno istraženo da bi se moglo definitivno zaključiti kako troši previše energije s obzirom na potencijalno unaprjeđenje preciznosti.

Sama preciznost brojača i potrošnja energije također ovisi o različitim načinima rada uređaja kao što su način rada niske i super niske potrošnje, čime bi se mogao dobiti dodatan uvid u razinu preciznosti koju je moguće ostvariti s još nižom potrošnjom energije. Povezivanje više od dva uređaja u praksi bi trebalo donijeti iste rezultate zbog načina na koji *Bluetooth* propagira poruke. Bez obzira na to, istraživanje u tom smjeru pružilo bi nešto stvarniji pristup pošto se u praksi često koriste više od dva povezana ESP32 uređaja.

Jedno od poglavlja u radu je i sinkronizacija uređaja protokolima sinkronizacije sata kao što su NTP, SNTP i PTP. Ti protokoli nisu implementirani u radu zbog kompleksnosti i veće potrošnje energije zbog potrebe za stalnom ili povremenom internetskom vezom. Uzme li se u obzir vrlo visoka preciznost takvih satova, u razini mikro sekunde ili bolje, moguće je postići mnogo veći stupanj preciznosti. Potrebno bi bilo istražiti koliko su ti algoritmi zahtjevni na sam rad uređaja i koliko ograničuju ostale aspekte njegova rada.

Za kraj, jedno od područja koje bi se direktno moglo detaljnije implementirati u rješenje predstavljeno u ovom radu je statistička analiza i filtriranje vremena kašnjenja. Implementacijom raznih algoritama filtriranja podataka i više referentnih vrijednosti vremena slanja poruke, potencijalno je moguće dobiti precizniji odmak i predvidjeti okvirno vrijeme trajanja slanja iduće poruke.

# Zaključak

Sinkronizaciju više uređaja iz ESP32 serije koristeći *Bluetooth Low Energy* protokol moguće je postići do određene razine. S obzirom na trajanje samog procesa slanja poruke koje u jednom smjeru iznosi nekoliko desetaka milisekundi, ovisno o postavkama, te koje nije konstantno, gotovo je nemoguće osigurati potpunu sinkronizaciju kod kontinuirane komunikacije.

Proces obrade poruke i sinkronizacije može se postići u vrijeme ispod jedne milisekunde. Taj proces nešto je konzistentnijeg trajanja od procesa slanja poruke, no još uvijek predstavlja mogućnost odmaka s obzirom na druge procese koji se odvijaju na uređajima koji imaju podjednake ili više prioritete, kao i memorijsko stanje.

Maksimalna konzistentno postignuta preciznost koristeći razne algoritme je u rangu od nekoliko milisekundi. Korištenje internih brojača i alarma nakon inicijalne sinkronizacije najpouzdaniji je način zadržavanja originalno postignutog stupnja sinkronizacije zbog visokog prioriteta istih i konzistentnog vremena izvođenja čak i s osnovnim oscilatorom koji ne pruža najveću razinu preciznosti.

Uzimajući u obzir potrošnju energije, latenciju, konzistentnost trajanja i ostale parametre nekoliko različitih bežičnih protokola za komunikaciju, može se zaključiti kako BLE pruža optimalan omjer fleksibilnosti i preciznosti, pogotovo uzimajući u obzir njegovu rasprostranjenost na velik broj različitih platformi i podršku za iste.

Ukoliko je zahtjev sinkronizacija u razini mikro sekunde ili mikro sekunde, potrebno je koristiti neke energetski zahtjevnije bežične protokole ili povezivanje uređaja pomoću žice radi finijeg usklađivanja satova.

# Literatura

|  |  |
| --- | --- |
| [1] | Espressif, »Products - Modules,« [Mrežno]. Available: https://www.espressif.com/en/products/modules. |
| [2] | F. Riccardi, »Comparison Espressif ESP,« [Mrežno]. Available: https://gist.github.com/sekcompsci/2bf39e715d5fe47579fa184fa819f421. |
| [3] | Macnica, »ESP32-H2 Bluetooth 5.2 support,« [Mrežno]. Available: https://www.macnica.eu/news-archive/2021-08-18-announcing-esp32-h2-ieee-802154-bluetooth-52-le-risc-v-soc. |
| [4] | Mathworks, »Comparison Bluetooth,« [Mrežno]. Available: https://www.mathworks.com/help/bluetooth/gs/comparison-of-bluetooth-bredr-and-bluetooth-le.html. |
| [5] | Bluetooth, »Comparison BLE vs Classic,« [Mrežno]. Available: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/#:~:text=The%20Bluetooth%20Classic%20radio%2C%20also,medical%20(ISM)%20frequency%20band. |
| [6] | S. electronics, »Bluetooth 4.2 vs 5.0 Comparison,« [Mrežno]. Available: https://www.symmetryelectronics.com/blog/bluetooth-5-versus-bluetooth-4-2-what-s-the-difference/ . |
| [7] | Espressif, »ESP32-H2,« [Mrežno]. Available: https://www.espressif.com/en/news/ESP32\_H2. |
| [8] | Espressif, »Espressif ESP32 BLE 5.0 Certification,« [Mrežno]. Available: https://www.espressif.com/en/news/BLE\_5.0\_Certification . |
| [9] | Espressif, »ESP32 Wi-Fi,« [Mrežno]. Available: https://espressif-docs.readthedocs-hosted.com/projects/arduino-esp32/en/latest/api/wifi.html. |
| [10] | Espressif, »ESP-32 RF coexsistence,« [Mrežno]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32c3/api-guides/coexist.html#:~:text=ESP32%2DC3%20has%20only%20one,to%20receive%20and%20transmit%20packets. |
| [11] | Espressif, »BLE Dual mode,« [Mrežno]. Available: https://www.espressif.com/sites/default/files/documentation/btble\_coexistence\_demo\_en.pdf. |
| [12] | Espressif, »Bluetooth for ESP32 General,« [Mrežno]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/bluetooth/index.html. |
| [13] | A. R. F. C. D. Eridani, »Comparative Performance Study of ESP-NOW, Wi-Fi, Bluetooth Protocols based on Range, Transmission Speed, Latency, Energy Usage and Barrier Resistance,« u *Conference: 2021 International Seminar on Application for Technology of Information and Communication*, 2021. |
| [14] | M. E. A. G. S. A. K. M.I. Labib, »An efficient networking solution for extending and controlling wireless sensor networks using low-energy technologies,« *PeerJ Comput Sci.,* 2021. |
| [15] | Espressif, »GitHub - ESP32 NimBLE Code example,« [Mrežno]. Available: https://github.com/espressif/esp-idf/blob/master/examples/bluetooth/nimble/ble\_spp/spp\_server/main/main.c . |
| [16] | Espressif, »GitHub - ESP32 Bluedroid Code example,« [Mrežno]. Available: https://github.com/espressif/esp-idf/blob/master/examples/bluetooth/bluedroid/ble/ble\_spp\_server/main/ble\_spp\_server\_demo.c . |
| [17] | Espressif, »GitHub - ESP32 Bluetooth Classic Code example,« [Mrežno]. Available: https://github.com/espressif/esp-idf/blob/master/examples/bluetooth/blufi/main/blufi\_example\_main.c. |
| [18] | Espressif, »GitHub - ESP32 ESP-Now Code example,« [Mrežno]. Available: https://github.com/espressif/esp-now/blob/master/examples/get-started/main/app\_main.c . |
| [19] | Espressif, »GitHub - ESP32 Wi-Fi Code example,« [Mrežno]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/esp\_http\_server.html . |
| [20] | K. Danijel, »Network Time Protocol,« 2005. [Mrežno]. Available: http://spvp.zesoi.fer.hr/seminari/2005/KlasicekDanijel\_NetworkTimeProtocol. |
| [21] | T. Ltd, »NTP,« [Mrežno]. Available: https://timetoolsltd.com/ntp/ntp-timing-accuracy/. |
| [22] | M. Global, »SNTP Simplified,« [Mrežno]. Available: https://www.meinbergglobal.com/english/faq/faq\_37.htm#:~:text=SNTP%20. |
| [23] | Espressif, »ESP32 Supported SDKs and Libraries,« [Mrežno]. Available: https://www.espressif.com/en/products/sdks/esp-idf . |
| [24] | »GitHub - ESP32 power consumption,« [Mrežno]. Available: https://github.com/espressif/esp-idf/issues/947#issuecomment-500312453 . |
| [25] | Espressif, »ESP32 System Time,« [Mrežno]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/system\_time.html . |
| [26] | Wikipedia, »Y2K38 problem,« [Mrežno]. Available: https://en.wikipedia.org/wiki/Year\_2038\_problem . |
| [27] | H. Dekker, »ESP32 interrupts can only do 200 khz,« [Mrežno]. Available: https://haydendekker.medium.com/esp32-interrupts-can-only-do-200khz-56f8dbb6a61c. |
| [28] | Espressif, »FreeRTOS with ESP32,« [Mrežno]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html. |
| [29] | FreeRTOS, »FreeRTOS details,« [Mrežno]. Available: https://www.freertos.org/a00110.html . |
| [30] | Espressif, »Improving performance speed,« [Mrežno]. Available: https://docs.espressif.com/projects/esp-idf/en/v4.4/esp32/api-guides/performance/speed.html. |
| [31] | Espressif, »ESP Timer,« [Mrežno]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/esp\_timer.html. |

# DODATAK A: Programska potpora

## ESP-IDF

**Instalacija**

* *Linux* i *MacOS*
  + <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/linux-macos-setup.html>
* *Windows*
  + <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/windows-setup.html>

**Naredbe**

**idf.py build** - izgradnja programa u kojem se dohvaćaju sve potrebne datoteke i povezivaju komponente - prvo pokretanje traje nekoliko puta više od sljedećih

**idf.py set-target esp32** - postavlja ciljani uređaj kompilacije na ESP32

**idf.py menuconfig** - otvara meni s postavkama

**idf.py -p (PORT) flash** - zapisivanje programa na ciljani uređaj (COM# na Windowsima)

**idf.py -p (PORT) erase-flash** - brisanje zapisanog programa iz *flash* memorije

**idf.py -p (PORT) monitor** - nadgledanje ispisa ciljanog uređaja

mode - prikazuje povezane USB uređaje

**Ctrl+]** - izlazak iz IDF monitora

**Greške**

* ‘Idf.py ne može biti pronađen’ - dodati %DISK:\Espressif\frameworks\esp-idf-v5.0.1-2\tools u PATH - (<https://docs.espressif.com/projects/esp-idf/en/v3.3/get-started-cmake/add-idf_path-to-profile.html> )

**Uređaj ESP32-WROOM**

* <https://www.upesy.com/blogs/tutorials/esp32-pinout-reference-gpio-pins-ultimate-guide>

## Konstrukcija koda

* **Link:** [**https://github.com/robert-fer98/ESP32\_Connections**](https://github.com/robert-fer98/ESP32_Connections)
* Detaljnija dokumentacija koda i objašnjenje pojedinih varijabli i funkcija nalazi se unutar koda

**Baza koda**

* Server - <https://github.com/espressif/esp-idf/tree/master/examples/bluetooth/bluedroid/ble/ble_spp_server>
* Klijent - <https://github.com/espressif/esp-idf/tree/master/examples/bluetooth/bluedroid/ble/ble_spp_client>

**Linkovi dokumentacije**

* Brojači
  + <https://docs.espressif.com/projects/esp-idf/en/v4.4/esp32/api-reference/peripherals/timer.html>
  + <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/esp_timer.html>
  + <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/system_time.html>
* Klijent
  + <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/bluetooth/esp_gattc.html>
* Server
  + <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/bluetooth/esp_gatts.html>
* FreeRTOS
  + <https://docs.espressif.com/projects/esp-idf/en/v4.3/esp32/api-reference/system/freertos.html>
* ESP-IDF
  + <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html#installation>
  + <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/tools/idf-monitor.html>
* Bluetooth i BLE
* <https://espressif-docs.readthedocs-hosted.com/projects/espressif-esp-faq/en/latest/software-framework/ble-bt.html>
* GPIO Pinovi:
  + 0-19, 21-23, 25-27, 32-39
  + <https://github.com/craftmetrics/esp32-button>

* **Funkcije postavljanja**
* **led\_setup()**
* **keyboard\_button\_handles\_setup()**
* **button\_setup()**
* **server\_message\_setup()**
* **timer\_setup()**
* **peripheral\_setup()**
* **client\_message\_setup()**
* **Pomoćne funkcije**
  + char\* **response\_extract\_clock**(char\* response)
  + char\* **response\_extract\_duration**(char\* response)
  + char\* **response\_extract\_last\_client\_timer\_local\_Time**(char\* response)
  + int64\_t **calculate\_server\_client\_clock\_difference**(**int64\_t** clientClock, **int64\_t** clientDuration)
  + Int64\_t **calculate\_client\_response\_time**(**int64\_t** incomingTimer, **int64\_t** clientStart)
  + int64\_t **calculate\_printf\_duration()**
  + int64\_t **calculate\_loge\_duration()**
  + int64\_t **calculate\_logi\_duration()**
  + int64\_t **calculate\_uart\_write\_bytes\_duration()**
  + int64\_t **calculate\_led\_control\_duration()**
  + char\* **timer\_value\_to\_char\_array(int64\_t** currentTime, **bool** addFunctionTime)
  + char\* **timer\_and\_response\_message\_reply(**)
  + int64\_t **char\_array\_to\_timer\_value(**char\* arrayValue)
  + void **timer\_callback(**void\* param**)**
  + void **timer\_start()**
  + void **timer\_stop()**
  + void **timer\_resync\_start()**
* Funkcije zadataka:
  + **void LED\_control\_task(void\*** ledPin**)**
  + **void uart\_task(void\*** pvParameters**)**
  + **void timer\_resync\_task(void\*** params**)**

* **Strukture**
  + struct **function\_durations**
  + struct **server\_send\_message**
  + struct **client\_send\_message**
  + struct **server\_basic\_message**
  + struct **client\_basic\_message**
  + struct **keyboard\_button\_handles**
* **Varijable**
  + **bool** server\_initiated\_message
  + **bool** client\_initiated\_message
  + **esp\_timer\_handle\_t** timer handler
  + **esp\_timer\_handle\_t** timer\_resync\_handler
  + **TaskHandle\_t** syncTask
  + **TaskHandle\_t** resyncTask
  + **button\_event\_t** ev
  + **QueueHandle\_t** button\_events
  + **int64\_t** server\_client\_clock\_difference
  + **int64\_t** last\_server\_timer\_local\_time
  + **int64\_t** last\_server\_LED\_local\_time
  + **int64\_t** last\_client\_timer\_local\_time
  + **int64\_t** server\_client\_clock\_difference
  + **int64\_t** initial\_server\_client\_clock\_difference
  + **int64\_t** last\_timer\_local\_time
  + **int** delay\_value
  + **bool** timer\_resync\_enable
  + **bool** timer\_resync\_allow
  + **bool** console\_logging
  + **bool** static\_delay\_value

* **Naredbe upravljanja pomoću tipkovnice**
* ‘W’ - paljenje/gašenje LED-ice nakon ručno izračunate vrijednosti x - vrijednost x je otprilike jednaka polovici trajanja putovanja poruke te se LED-ice na klijentu i serveru pale/gase bliže nego inače
* ‘A’ - paljenje/gašenje LED-ice nakon automatski izračunate vrijednosti x - vrijednost x jednaka je izračunatoj polovici vrijednosti trajanja putovanja poruke kroz prethodno slanje poruke
* ‘T’ - pokretanje brojača koji pali/gasi LED-icu svakih 10 sekundi - ako je brojač već pokrenut, ispisat će se vrijeme do sljedećeg paljenja/gašenja - pokreće se i sinkronizacija koja se odvija svakih 12 sekundi (smanjeno radi testiranja)
* ‘S’ - računanje vremena trajanja slanja poruke i razlike između lokalnih satova servera i klijenta
* ‘U’ - zaustavljanje brojača koji pali/gasi LED-icu svakih 10 sekundi
* Sve ostale tipke - pale/gase LED-icu na serveru koja bi se trebala upaliti/ugasiti 40-50 ms prije one na klijentu (trajanje putovanja + obrada) - ista komanda može se koristiti i na klijentu
* ‘O’ - gašenje/paljenje ispisivanja dodatnih logova u CMD kako bi se ubrzao program - ista komanda može se koristiti i na klijentu
* ‘L’ i ‘C’ koriste se za sinkronizaciju te ih je bolje ne koristiti

1. Slučaj je nekonzistentan [↑](#footnote-ref-1)
2. Slučaj je konzistentan i nakon nekoliko desetaka ciklusa [↑](#footnote-ref-2)