Prilagodba i integracija TCP/IP stoga u ERIKA okruženje na Aurix platformi

Babić, Josip

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Diplomski studij elektrotehnike

PRILAGODBA I INTEGRACIJA TCP/IP STOGA U ERIKA OKRUŽENJE NA AURIX PLATFORMI

Diplomski rad

Josip Babić

Osijek, 2017.

Sadržaj

1. UVOD

Sredinom 60-tih godina prošlog stoljeća, pojavom računalnih mreža sličnih današnjim, nastaje potreba za sigurnim sustavom komunikacijskih protokola. Kao rješenje za ovaj problem pojavio se koncept slojevitog povezivanja protokola, a najpoznatiji protokolski stog nastao za ove potrebe je TCP/IP (engl. *Transmission Control Protocol / Internet Protocol*) protokolski stog na kojemu se između ostalog zasniva i Internet. TCP/IP protokolski stog predstavlja siguran i učinkovit sustav koji zadovoljava navedene potrebe i pruža podršku brojnim protokolima viših i niži razina, koji nalaze primjenu u suvremenim tehnologijama. Postoje brojne implementacije navedenog protokolskog stoga, od kojih su neke besplatne i otvorenog koda (engl. *open source*), a neke predviđene za komercijalnu upotrebu, te su licencirane i manje pristupačne. Kao fokus proučavanja ovog diplomskog rada, odabrana je lwIP (engl. *lightweight TCP/IP*) implementacija, zbog vrste mikrokontrolera na kojemu će se vršiti testiranje, ali i zbog jednostavnosti i dostupnosti dokumentacije i podrške u vidu foruma.

Za potrebe ovog diplomskog rada koristi se mikrokontroler iz *Aurix* serije kao hardver na kojemu će se izvoditi napisano programsko rješenje. Mikrokontroleri ove vrste obično se koriste u automobilskoj industriji, a rade se u verzijama s jednom ili više procesorskih jezgara. Za sinkronizaciju programskog rješenja, efektivnije izvođenje i podizanje aplikacije na viši programski nivo, cijeli program se prilagođava radu u operacijskom sustavu realnog vremena (engl. *Real-Time Operating System*, skraćeno RTOS). RTOS odabran za sinkronizaciju je *ERIKA Enterprise* sustav, koji je posebno namijenjen autoindustriji.

U okviru ovog diplomskog rada potrebno je izraditi navedeno programsko rješenje koje će se oslanjati na navedene hardverske i softverske komponente. Da bi se što bolje opisao navedeni problem, diplomski rad je podijeljen na tri cjeline. U prvoj cjelini opisan je TCP/IP stog i implementacija korištena u ovom radu. Drugi dio opisuje korišteni mikrokontroler, operacijski sustav i način integracije cijelog sustava, a u posljednjem poglavlju prikazani su rezultati rada razvijenog rješenja.

1.1. Zadatak diplomskog rada

TCP/IP je poznati stog protokola u računalnim mrežama. U radu je potrebno prilagoditi i integrirati postojeći TCP/IP (lwIP) upravljački program. Kompletan razvoj potrebno je provesti na *Aurix Triboard* platformi, s aplikativnim scenarijem UDP poslužitelja.

2. TCP/IP

2.1. Uvod u TCP/IP i povijesni razvoj

Za uspješnu komunikaciju dvaju komunikacijskih sudionika potrebno je da navedeni članovi govore istim jezikom i da se pridržavaju dogovorenih pravila. Skup dogovorenih i jednoznačnih pravila za komunikaciju čini komunikacijski protokol. S protokolima se moguće susresti svakodnevno, a postoje brojne vrste protokola koji mogu biti manje ili više složeni, ovisno o namjeni. Za pravilnu komunikaciju među računalima također se koriste za to predviđeni protokoli. Ovaj proces razmjene informacija može biti vrlo složen, pa stoga zahtjeva i protokole više složenosti.

Složeniji protokoli se često oslanjaju na jednostavnije protokole, ili koordiniraju radom više njih. Ako se poveže više protokola koji zajedno funkcioniraju i imaju sličnu namjenu dobije se stog ili slijed protokola (engl. *protocol suite*). Jedan od najpoznatijih i najraširenijih protokolskih stogova je TCP/IP protokolski stog.

TCP/IP nastao je u sklopu istraživanja istraživačke agencije Sjedinjenih Američkih Država (engl. *Defense Advanced Research Project Agency* – DARPA) i razvijen je za mrežnu infrastrukturu nazvanu *ARPAnet* kako bi se objedinili svi prethodno korišteni protokoli.

Ideja za ovakvo rješenje seže još u drugu polovicu 60-tih godina prošlog stoljeća, a razvoj navedenog protokolskog stoga počeo je početkom 70-tih godina, tj. 1972. godine kada je Robert Elliott (Bob) Kahn, zadužen za hardverski dio *ARPAnet* mreže predstavio dizajn koji je prethodio TCP-u (engl. *Transmission Control Protocol*). Godinu dana poslije, Vinton Gray (Vint) Cerf pridružio se istraživanju koje je za rezultat imalo nastanak TCP-a (engl. *Transmission Control Program*) drugačijeg od onog kakav je danas. Ovo rješenje dokumentirano je 1974. godine pod normom RFC 675. U svojoj prvoj verziji, ovaj protokolski stog obavljao je dvije funkcije:

- funkciju slanja paketa
- funkciju usmjeravanja paketa

Za usmjeravanje paketa koristio se uređaj pod nazivom pristupnik (engl. *gateway*) koji je naknadno preimenovan u usmjerivač (engl. *router*) zbog preklapanja pojmova.

Testiranjem narednih verzija, ukazano je na propuste i preopterećenost ovakvog koncepta, te na mogućnost razdvajanja TCP-a na dvije samostalne cjeline od kojih bi svaka obavljala jednu od dvije prethodno navedene funkcije. Ova ideja dovela je do podjele na TCP

(engl. *Transmission Control Protocol*) i na IP (engl. *Internet Protocol*) u verziji 3, a taj se koncept zadržao do danas. Cjelina dobivena iz ova dva sloja čini protokolski stog nazvan TCP/IP. Ova verzija dokumentirana je 1980. godine i smatra se prvom modernom verzijom TCP/IP-a.

Daljnjim razvojem, zbog dobrih karakteristika koncepta, navedeni protokolski stog postaje standard, implementira se sve više protokola, te ga koristi sve veći broj uređaja i TCP/IP postaje osnova *ARPAnet* infrastrukture koja je prethodnik današnjeg Interneta.

2.2. Arhitektura i dizajn

Osnovna zadaća TCP/IP-a je pružanje podrške velikom broju uređaja povezanih na Internet, a uređaji mogu biti razne vrste i veličina, različitih proizvođača i različitog softvera. Pod pojmom Internet misli se na bilo koji oblik komunikacije međusobno udaljenih računala. Uređaji koji komuniciraju često su smješteni u različitim i udaljenim računalnim mrežama, te je potrebno na određeni način prenijeti informacije u točnu mrežu i na pravi uređaj. Da bi se ostvarila tražena funkcionalnost potrebno je koristiti dodatne uređaje poput usmjerivača i preklopnika, ali i uvesti dodatne mehanizme i koncepte poput paketa, datagrama, konekcije, kontrole greške, kontrole toka itd.

Jedan od najbitnijih koncepta za rad ovakve mreže jest koncept komutacije paketa. Osnovna ideja ovog koncepta je da se podatci namijenjeni prijenosu grupiraju u manje pakete i da se ti paketi šalju kroz mrežu neovisno o drugim paketima. Dolaskom na spojna mjesta mreže i uređaje za usmjeravanje, paketi se spremaju u određene spremnike, a zatim na osnovu informacija o usmjeravanju spremljenih u usmjerivačima šalju dalje. Ovaj proces usmjeravanja često se naziva multipleksiranje. Nedostatak ovog pristupa je mogućnost zagušenja zbog prevelikog broja paketa, jer je za usmjeravanje svakog paketa potrebno određeno vrijeme. Drugi pristup ovom problemu je vremensko multipleksiranje, ali i ovaj koncept ima problem u vidu slabijeg iskorištenja kapaciteta mreže.

U vremenskim multipleksima, ostvaruje se određeni oblik direktne veze slične električnom krugu u početcima telefonskih mreža, što se naziva konekcijom, a tako je i nastao jako raširen koncept konekcijski orijentirane mreže.

Razvojem ideje o paketima nastali su datagrami. Datagram predstavlja prošireni paket, koji osim podataka namijenjenih za prijenos sadrži i informacije o izvoru i odredištu paketa. Na ovaj način, informacije o usmjeravanju više nisu bile smještene u usmjerivače, te je

omogućen razvoj beskonekcijskih mreža. Nedostatak datagrama je veličina, jer se u svaki paket zapisuje relativno velik broj dodatnih informacija, što utječe na brzinu prijenosa.

Prilikom prijenosa podataka, može doći do oštećenja paketa ili gubitka njegovih dijelova. Do ovoga obično dolazi zbog hardverskih grešaka ili vanjskih utjecaja na mrežu i uređaje. Ako je izgubljen manji dio podataka, reda nekoliko bita, ovakve greške prijenosa se obično mogu otkloniti zaštitnim mehanizmima u obliku matematičkih provjera (npr. *checksum* provjera) i često ih se rješava unutar same mreže. Ako dođe do grešaka koji se ne mogu otkloniti navedenim mehanizmima, nekada je potrebno izvršiti ponovno slanje cijelog paketa. Otklanjanje ovakvih grešaka u radu mreže naziva se kontrolom greške

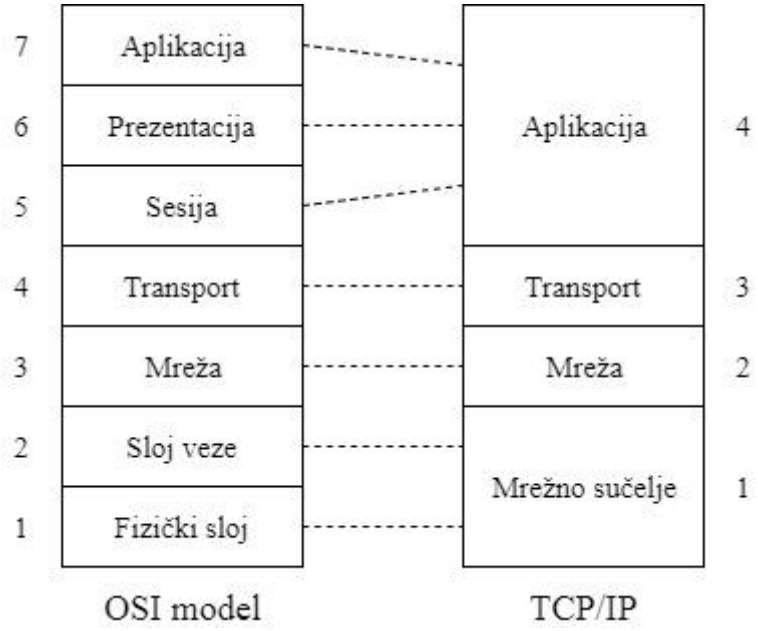
Kako ne bi došlo do pretrpavanja mreže paketima koji se ne mogu dovoljno brzo multipleksirati, potrebno je uvesti dodatni zaštitni mehanizam u vidu kontrole zagušenja. Ovaj mehanizam je obično implementiran van mreže, u protokolima više razine, te može djelovati na izvor paketa tako da uspori njihovo odašiljanje ili sličnim metodama. Osim toga, postoje i koncepti *best-effort* odlučivanja koji donose odluke kada se prestaje s procesom oporavke od greške te se paket odbacuje i traži ponovno slanje [1].

Osim navedenih mehanizama, TCP/IP se oslanja na hijerarhijsku strukturu nazvanu slojevitost (engl. *layering*). Ovakva struktura postala je klasična za protokolske stogove, a osnovna ideja je da svaki sloj obavlja neku od funkcionalnosti. Najpoznatiji model utemeljen na ovakvom pristupu je ISO-OSI (engl. *International Organization for Standardization / Open Systems Interconnection*) model, koji ima sličnosti sa TCP/IP-om. Upravo zbog korištenja ovakve slojevite arhitekture i potječe pojam protokolskog stoga. Karakteristika stogova je da se protokoli u višim slojevima oslanjaju na niže slojeve, dok ne vrijedi obratno.

U odnosu na referentni model (ISO-OSI), TCP/IP stog ima manji broj slojeva, s nešto drugačije raspoređenim funkcionalnostima:

- aplikacijski
- transportni
- mrežni
- fizički sloj/mrežno sučelje

U literaturama može pronaći i peti sloj, sloj linka podataka, ali su on i njegove funkcionalnosti većinom obuhvaćene fizičkim slojem. Na slici 2.1. prikazana je usporedba TCP/IP protokolskog stoga s referentnim ISO-OSI modelom:



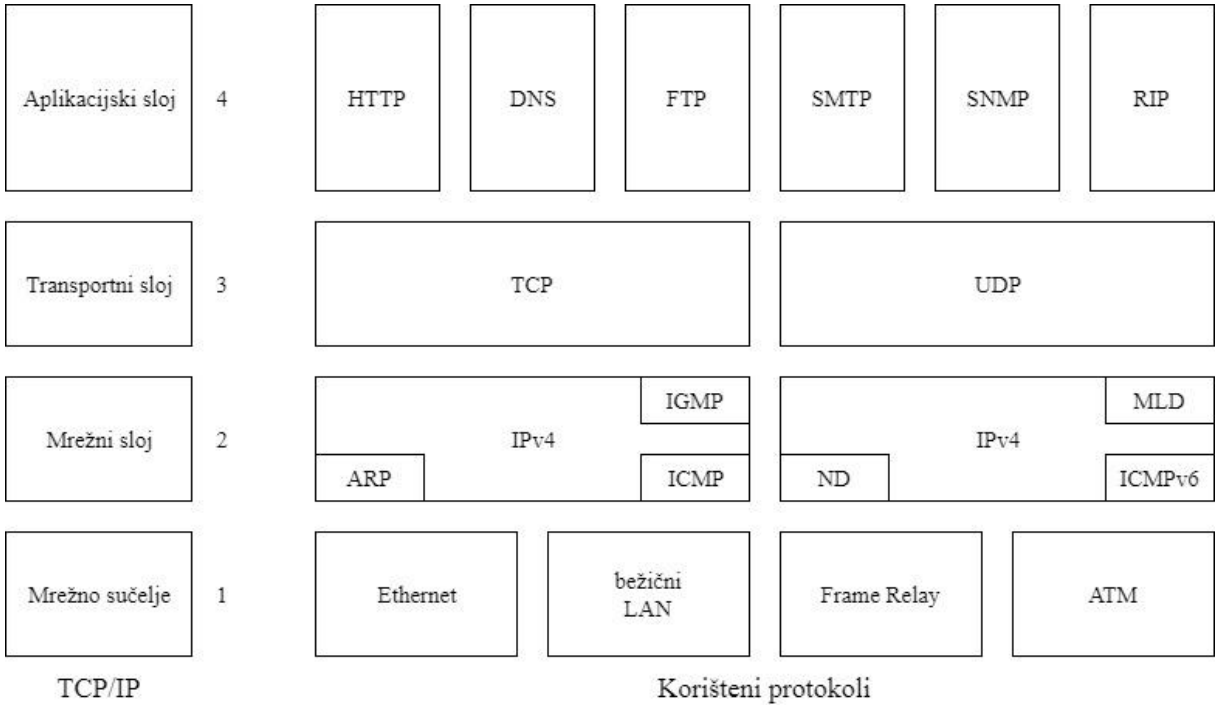
Slika 2.1. *Usporedba TCP/IP s referentnim OSI modelom* [1]

Na slici je vidljivo da se prva dva sloja OSI modela u TCP/IP-u mogu zamijeniti mrežnim sučeljem koje se često naziva i fizičkim slojem, mrežni sloj i transportni sloj ostaju isti, dok se funkcionalnosti posljednja tri sloja (sloj sesije, prezentacijski sloj i aplikacijski sloj) u TCP/IP-u zamjenjuju funkcionalnostima aplikacijskog sloja.

Fizički sloj određuje način na koji će se podatci spremni za prijenos prilagoditi prijenosnom mediju i slati dalje u mrežu i obuhvaća protokole namijenjene za obavljanje ovih funkcionalnosti. Sljedeći sloj, mrežni sloj ima zadaću adresiranja i usmjeravanja paketa kroz mrežne uređaje, odabir najkraćeg i optimalnog puta kroz mrežu. Iznad mrežnog sloja smješten je transportni sloj koji je zadužen za upravljanje tokom podataka, kontrolu zagušenja i slične mehanizme, zbog čega se često naziva najkompleksnijim slojem ovog protokolskog stoga. Povrh svega, nalazi se aplikacijski sloj koji predstavlja spojnu točku TCP/IP-a s korisnikom te mu pruža konkretne funkcionalnosti poput pretraživanja Weba posredovanjem odgovarajućih protokola namijenjenih za ovaj sloj.

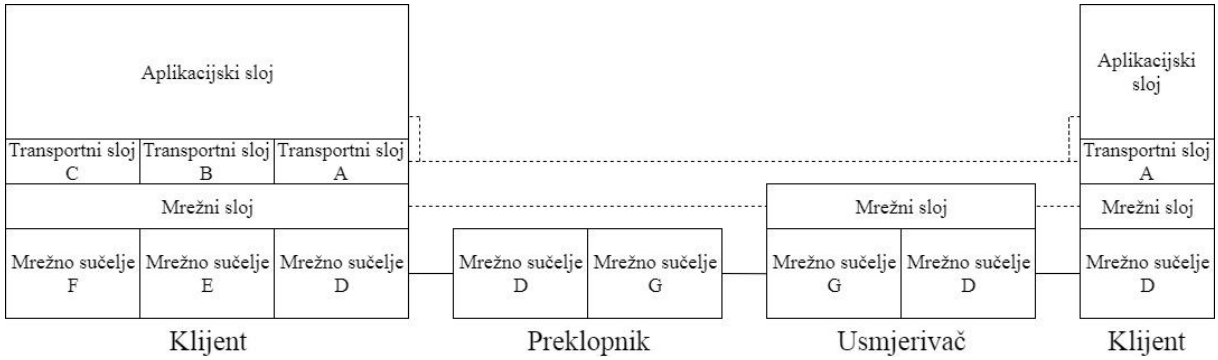
Za slojevite protokolske modele karakteristična je enkapsulacija podataka. Enkapsulacija se obično manifestira dodavanjem zaglavlja osnovnom paketu podataka kroz svaki sloj kroz koji podatak prođe. Tako npr. TCP protokol dodaje svoje zaglavlje u transportnom sloju, a na njega se u mrežnom sloju dodaje IP zaglavlje s IP adresama itd. Ovako nastali, prošireni paket, naziva se PDU (engl. *Protocol Data Unit*), a uglavnom svaki sloj protokola pravi svoj PDU, te ga prosljeđuje sljedećem sloju. Zbog svoje široke primjene,

moderni TCP/IP pruža podršku brojnim protokolima. Na slici 2.2. prikazani su najčešće korišteni protokoli za svaki sloj stoga:



Slika 2.2. *Najčešće korišteni protokoli u TCP/IP-u* [9]

Bitno je napomenuti da se implementacija TCP/IP-a u mrežnim uređajima razlikuje, te da nije potrebno u svim uređajima implementirati sve funkcionalnosti. Npr. usmjerivači i preklopnici nemaju potrebu za aplikacijskim slojem, pa je on izbačen. Potpunu implementaciju stoga najčešće zahtijevaju samo korisnici i poslužitelji. Ovakva mrežna topologija prikazana je na slici 2.3.



Slika 2.3. *Mrežna topologija predviđena za TCP/IP* [1]

2.3. Implementacije TCP/IP-a

Zbog svoje široke primjene, postoje brojne implementacije navedenog protokolskog stoga. Za potrebe ovog diplomskog rada razmatrat će se implementacije predviđene za ugradbene (engl. *embedded*) sustave s manjom količinom memorije i manjim procesorskim kapacitetom jer će se koristiti mikrokontroler namijenjen točno ovakvim uvjetima.

Implementacije TCP/IP-a mogu biti otvorenog koda ili zatvorene, namijenjene komercijalnoj upotrebi, a obično se razlikuju u širini adresne sabirnice procesora, količini potrebne memorije i performansama. Iz prve grupe, neke od najpoznatijih implementacija su:

- lwIP
- uIP
- uC/IP
- tinyIP
- WatTCP
- BSD

Od navedenih implementacija, najčešće korištene su prve dvije. LwIP implementacija namijenjena je 16-bitnim mikrokontrolerima, a za rad zahtijeva nešto više memorije od uIP-a. UIP namijenjen je 8-bitnim i 16-bitnim mikrokontrolerima, a potrebno mu je oko 10 kB ROM i 2 kB RAM memorije. U/IP obično se koristi u kombinaciji s operativnim sustavom *Contiki* i grafičkim sučeljem VNC. UC/IP zasniva se na BSD (engl. *Berkeley Software Distribution*) operacijskom sustavu u ugradbenim sustavima, te je zbog toga za njegov rad potrebno nešto više memorije nego prethodno navedenim sustavima. TinyIP, WatTCP i slične implementacije koje nisu spomenute zastarjele su te se rijetko koriste. Bitno je napomenuti da su sustavi zasnovani na BSD *Unix* izvedbi obično veliki i nepogodni za ugradbene sustave [2].

Također, postoje i brojne verzije koje su licencirane na neki način, te nisu besplatne i dostupne svima. Od takvih implementacija, neke od poznatijih su:

- *NicheStack*
- NetX
- CMX-TCP
- TargetTCP
- RTXC *Quadnet*

Sustavi poput ovih obično osim nižih slojeva protokola, obično su prošireni dodatnim protokolima aplikacijske razine (HTTP, FTP, SNMP, itd.) [2].

2.4. LwIP

LwIP je otvorena implementacija TCP/IP protokolskog stoga, a nastao je na SICS-u (engl. *Swedish Institute of Computer Science*). Navedeni projekt započeo je Adam Dunkels, osnivač današnjeg *Thingsquarea*, uIP-a, *Contiki* operacijskog sustava i koncepta *protothreadova*. Na razvoju lwIP-a se radi i danas, a na razvijaju ga brojni developeri iz cijeloga svijeta, na čelu s Kieranom Mansleyom.

Ovaj projekt, zbog svoje dostupnosti i dobrih performansi, našao se u širokoj komercijalnoj upotrebi, te je najkorišteniji besplatni TCP/IP stog. Za rad, potrebno mu je oko 40 kB ROM memorije i oko 20 kB RAM memorije, što ga čini izuzetno pogodnim za namjenske sustave. Bitno je napomenuti da se navedene brojke odnose na potpunu verziju s implementiranim svim protokolima, te se na količinu zauzete memorije može utjecati izbacivanjem pojedinih dijelova koji nisu potrebni. Smanjenjem opsega projekta, može se pozitivno utjecati na performanse. LwIP najčešće se koristi za 16-bitne, ali se može koristiti i za 32-bitne mikrokontrolere, prilagođen (portovan) je brojnim platformama i licenciran BSD licencom [4].

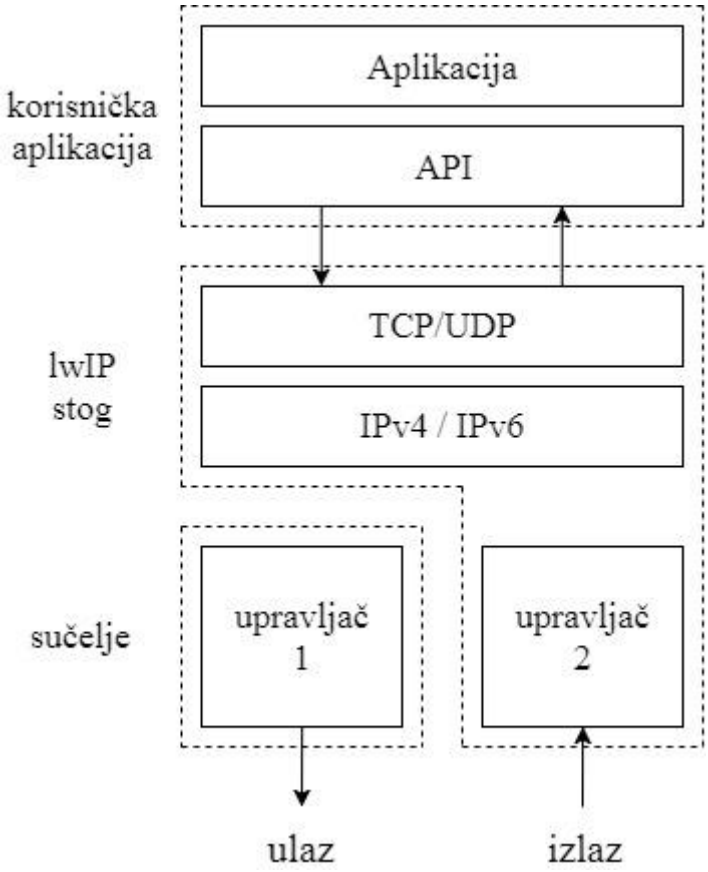
Prilikom korištenja lwIP stoga, moguće je koristiti brojne protokole s različitih slojeva TCP/IP stoga:

- Ethernet
- ARP – *Address Resolution Protocol*
- IPv4 – *Internet Protocol version 4*
- IPv6 – *Internet Protocol version 6*
- ICMP – *Internet Control Message Protocol*
- IGMP – *Internet Group Management Protocol*
- TCP – *Transmission Control Protocol*
- UDP – *User Datagram Protocol*
- DHCP – *Dynamic Host Configuration Protocol*
- PPP – *Point-to-Point Protocol*
- SNMP – *Simple Network Management Protocol*
- AUTOIP – *AUTOmatic Internet Protocol*

Za konkretnu komunikaciju sa stogom, lwIP svojim korisnicima pruža tri API (engl. *Application Programming Interface*) razine:

- *raw* API
- *netconn* API
- *socket* API

Raw API predviđen je za rad stoga bez operativnog sustava, te ne pruža mogućnost korištenja sinkronizacijskih mehanizama poput semafora i kutija za poruke (engl. *message box*). Prilikom korištenja ovakvog pristupa, pri korištenju određenih modula, potrebno je manualno paziti na sinkronizaciju u korištenju resursa. *Netconn* API omogućuje korištenje niti (engl. *thread*) i niz dodatnih proširenja u obliku *netconn* funkcija iz API modula. Ovaj API olakšava sinkronizaciju niti uvođenjem semafora i kutija za poruke. *Socket* API oslanja se na BSD *sockete* iz *Unix* sustava, a korištenjem ovog sučelja smanjuje se mogućnost pogreške i dobiva na pouzdanosti i sigurnosti. API sučelja predstavljaju spojnu točku aplikacije koju izrađuje korisnik i TCP/IP stoga. Arhitektura lwIP stoga prikazana je na sljedećoj slici:



Slika 2.4. *Arhitektura i sučelja lwIP-a* [4]

Trenutno najnovija verzija lwIP-a je verzija 2.0.2., objavljena na datum 13.3.2017. Navedena implementacija napisana je u programskom jeziku *C*, a cijeli projekt može se preuzeti besplatno, sa službene stranice. Preuzimanjem projekta, dobije se komprimirana datoteka, koja sadrži zaglavlja i izvorne datoteke za svaki modul. Svi protokoli obuhvaćeni u lwIP-u posjeduju zaseban modul, što značajno utječe na preglednost i olakšava rad.

Za rad s navedenim stogom, potrebno je preuzete module ugraditi u projekt, te dodati u pretprocesorske putanje radi povezivanja. Prije početka rada, potrebno je konfigurirati stog za potrebe korisnika. Proces konfiguracije obavlja se izmjenjivanjem datoteke *lwipopts.h*, a u navedenoj datoteci vrše se osnovne postavke stoga i mogu se uključiti ili isključiti pojedini moduli, upravljati raspodjelom memorije i brojne druge opcije. Sadržaj datoteke je mnoštvo pretprocesorskih naredbi (*#define*) koje se mijenjaju prema uputama zapisanim u komentarima, npr:

#define NO_SYS 1

Navedena linija *lwipopts.h* datoteke definira hoće li se u sustavu koristiti *raw* API ili neki od preostala dva.

Osim *lwipopts.h* datoteke, za korištenje lwIP stoga, potrebno je izvršiti konfiguraciju datoteka specifičnih za arhitekturu na kojoj će se program izvoditi. Zaglavlje *cc.h* namijenjeno je opisivanju kompajlera i procesora, te je u njemu potrebno definirati:

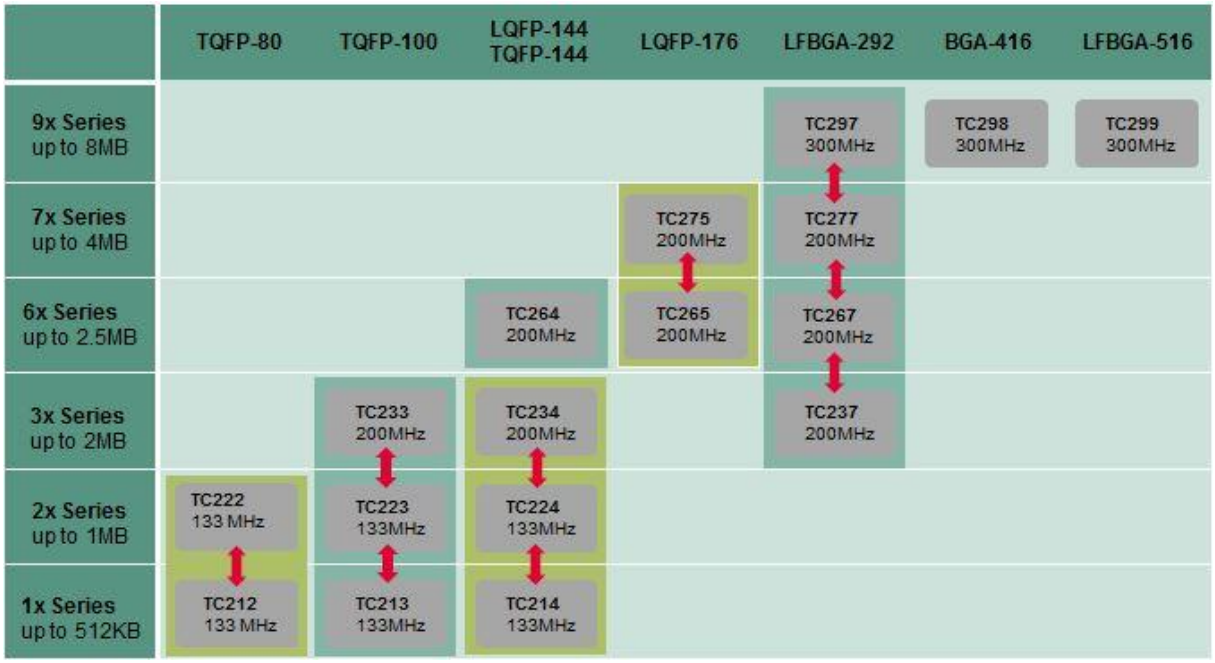
- tipove podataka
- način ispisivanja podataka
- arhitektura procesora (*little/big endian*)
- pakiranje struktura
- zaštitni checksum
- zaštitu od oduzimanja prava (*preemption*)

Potrebno je i implementirati datoteke *sys_arch.h* i *sys_arch.c*, od kojih prvi sadrži prototipe funkcija, a drugi definicije funkcija za sinkronizaciju niti pomoću semafora i kutija za poruke. Ako se koristi *raw* API, za ove funkcije dovoljno je imati null-definicije jer se tada praktički ne koriste.

3. INTEGRACIJA NA AURIX PLATFORMU I ERIKA RTOS

3.1. Aurix

Pojam *Aurix* predstavlja seriju mikrokontrolera proizvođača *Infineon Technologies AG* i radi se o mikrokontrolerima posebno namijenjenim za automobilsku industriju. Mikrokontroleri su 32-bitne, višejezgrene arhitekture i visokih performansi, a izrađuju se u kombinacijama različite veličine *flash* memorije i različitih procesorskih kapaciteta [5].Na slici 3.1. prikazane su navedene kombinacije:



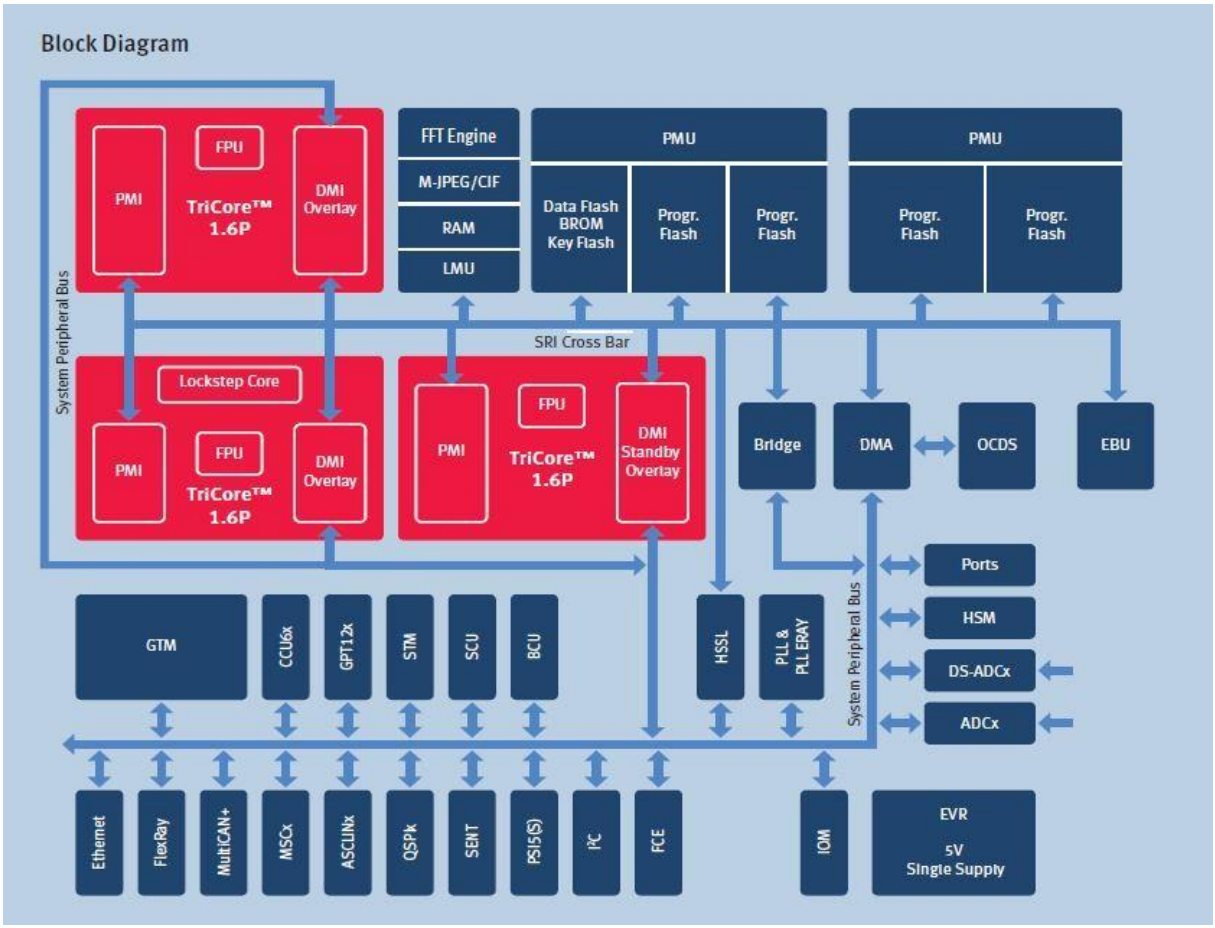
Slika 3.1. *Prikaz mogućih izvedbi Aurix mikrokontrolera* [5]

Za potrebe izrade diplomskog rada korišten je paket TC297. Navedeni paket sadrži TC297 A-step ploču, USB (engl. *Universal Serial Bus*) konektor, te adapter za napajanje. Navedena ploča posjeduje tri procesorske jezgre (engl. *TriCore*) frekvencije 300 MHz i 8 MB ROM memorije. Za napajanje koristi se 5 V adapter sa standardnim DC konektorom. Za komunikaciju s računalom može se koristiti micro USB konektor ili DAP (engl. *Device Access Port*) *miniWiggler*. Ploča osim navedenih, posjeduje i brojna druga sučelja:

- RJ45 konektor za Ethernet
- 16-pinski JTAG konektor
- 10-pinski DAP konektor

- 10-pinski LIN konektor
- 2 x 10-pinski CAN konektor
- 2 x 10-pinski *FlexRay* konektor
- 4 x 80-pinski konektor za sve ulazne i izlazne signale

Na slici 3.2. prikazan je blokovski dijagram TC297 eksperimentalne ploče:



Slika 3.2. *Blokovski dijagram TC297 eksperimentalne ploče* [6]

Na blokovskom dijagramu prikazana je arhitektura navedenog čipa zajedno sa svim prethodno navedenim modulima i načinom njihovog povezivanja. Za rad s modulima, potrebno je koristiti upravljačke programe - iLLD (engl. *Infineon Low-Level Driver*). ILLD predstavljaju spojnu točku aplikacije s hardverom, odnosno konkretnim modulima, a sadrže brojne konfiguracijske funkcije i strukture podataka koje se upisuju u odgovarajuće registre.

Slika 3.3. *Prikaz jednofunkcionalnih upravljačkih programa* [5]

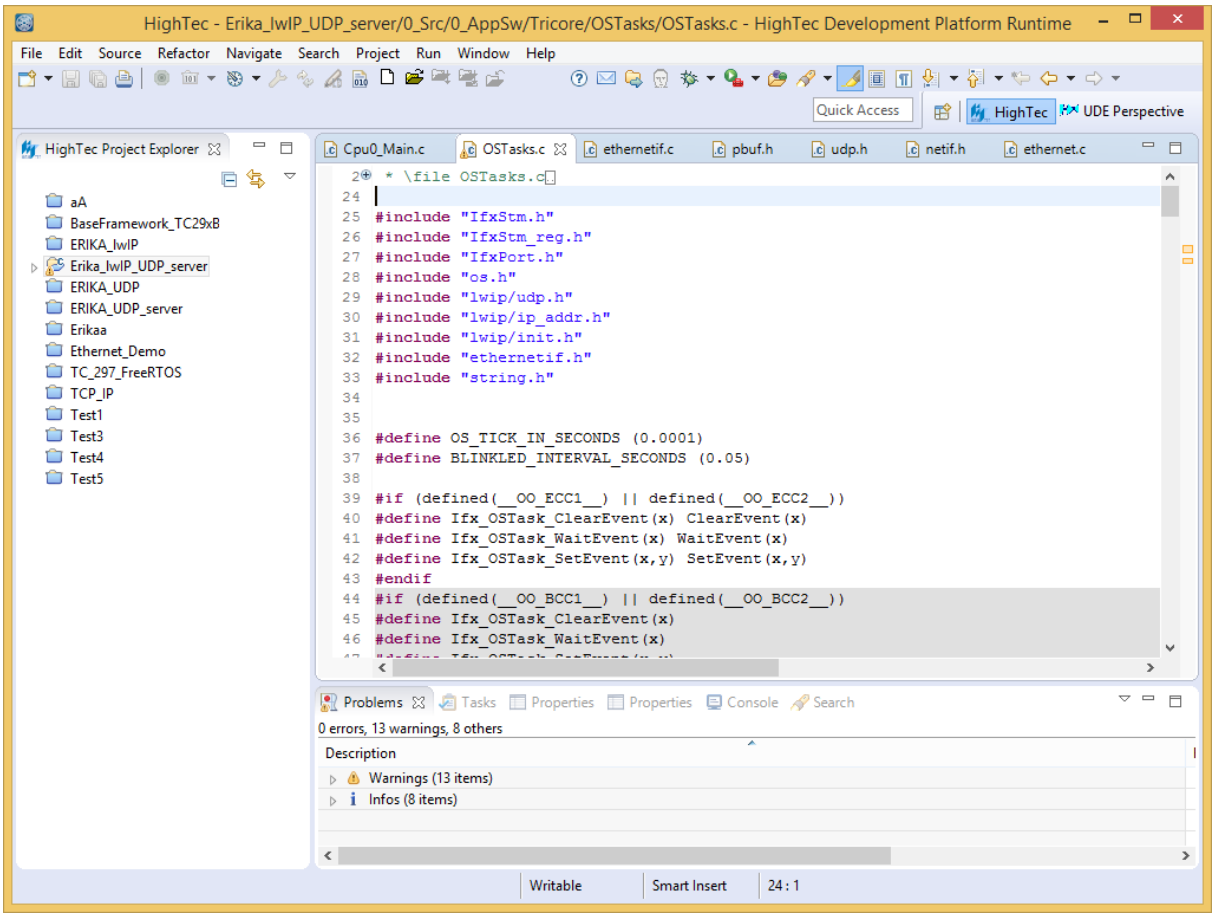
Upravljački programi mogu biti jednofunkcionalni i multifunkcionalni. Kod jednofunkcionalnih upravljačkih programa svaki iLLD paket upravlja jednim hardverskim modulom, dok je kod multifunkcionalnih moguće da više iLLD-a upravlja jednim modulom. Razlika između ove dvije vrste upravljačkih programa prikazana je na slikama 3.3. i 3.4.

Slika 3.4. *Prikaz multifunkcionalnog upravljačkog programa* [5]

Za spajanje ploče s računalom radi prenošenja programa, potrebno je na računalo proširiti dodatnim softverom. Na računalo je potrebno instalirati DAS (engl. *Device Access Server*) kako bi eksperimentalna ploča bila prepoznata od strane računala, a zatim i *Infineon Memtool* koji vrši konkretan prijenos programa (*flashovanje*) na ROM memoriju ploče. Osim navedenog, potrebno je instalirati i odgovarajuće razvojno okruženje. Korišteno razvojno okruženje je nazvano *HighTec Development Platform*.

3.2. HighTec Development Platform

HighTec je razvojno okruženje namijenjeno za više arhitektura (*Tricore*, *Power Architecture*, ARM, RH850), a raditi se može s jednom ili više procesorskih jezgara. Razvojno okruženje izrađeno je kao modifikacija razvojnog okruženja *Eclipse*, pa se zbog toga još i često naziva *Eclipse for Tricore*. Radi olakšane instalacije navedenog okruženja, moguće je preuzeti *Free Tricore Entry Tool Chain*, uz koji se dobije besplatna licenca u trajanju od šest mjeseci. *Entry Tool Chain* sadržava sve alate potrebne za razvoj softvera - kompajler, povezivač (engl. *linker*) i *debugger*, te ih nije potrebno dodatno ugrađivati i podešavati. *HighTec* koristi GNU kolekciju kompajlera napisanih za programski jezik C. Za *debuggiranje* programskog rješenja koristi se UDE (engl. *Universal Debug Engine*).



Slika 3.5. *HighTec razvojno okruženje*

Pri izradi *TriCore* projekta, preporučuje se korištenje *Infineon* softverskog okvira (engl. *Software Framework* - FW). Okvir predstavlja standard za izradu projekata i definira

strukturu projekta, tj. određuje raspored datoteka u projektu kako bi se povećala preglednost i modularnost. Na ovaj način znatno se olakšava rad na postojećim projektima zbog jednake strukture za svaki projekt. *Infineon* FW zadaje sljedeću strukturu:

Slika 3.6. *Struktura Infineon FW projekta*

Mapa *0_Src* sadrži izvorne datoteke koje služe kao objekti za kompajliranje projekta, a *1_ToolEnv* sadrži datoteke vezane za *toolchain* i razvojno okruženje. U mapu *2_Out* spremaju se izlazne i izvršne datoteke, a *3_Doc* je predviđen za dokumentaciju. Navedene mape posjeduju brojne poddirektorije, o čemu se više može pročitati u uputama za korištenje okvira.

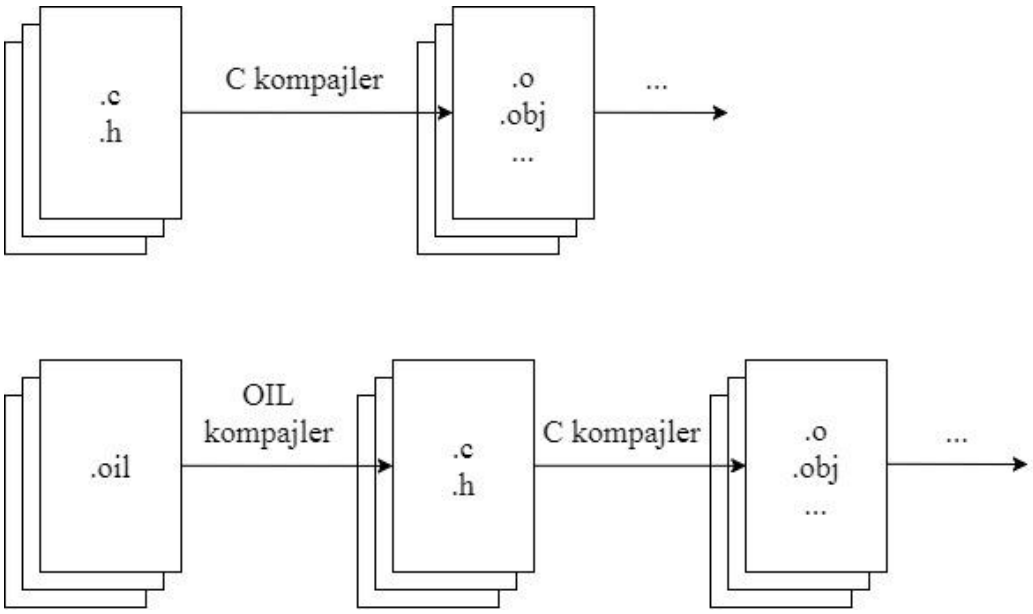
3.3. ERIKA Enterprise

ERIKA Enterprise je otvoreni RTOS koji je besplatan i prilagođen raznim standardima. Prilagođen je više jezgrenim arhitekturama i omogućuje dijeljenje memorijskog stoga među više zadataka (engl. *task*). Predviđen je za rad s 8-bitnim do 32-bitnim mikrokontrolerima i potrebno mu je 1 – 4 kB *flash* memorije, a za integraciju u *Eclipse* okruženje koristi se alat pod nazivom *RT-Druid*. [7]

Zadatak predstavlja obično manji dio koda koji se treba izvršiti jednom ili više puta (neki zadatci se izvršavaju sve dok je program aktivan) i većina RTOS-a se zasniva upravo na konceptu zadataka. Pri radu sustava, više zadataka može biti aktivno u isto vrijeme, pa im se stoga dodjeljuju prioriteti izvođenja. Redosljed izvođenja zadataka određuje centralni mehanizam RTOS-a nazvan planer (engl. *scheduler*). *ERIKA* je *hard* operativni sustav, što znači da prilikom izvođenja programa, nema promijene prioriteta zadataka. Za *ERIKA* sustav bitno je napomenuti, da se sve konfiguracije vrše prije izvođenja programa, odnosno da prilikom izvođenja programa nijedan tip objekata ne može niti nastati, niti nestati.

ERIKA sustav implementira OSEK/VDX standard. OSEK (njem. ***O****ffene* ***S****ysteme und deren Schnittstellen für die* ***E****lektronik in* ***K****raftfahrzeugen*) standard namijenjen je

automobilskoj industriji i namjenskim sustavima, a osnovan je od strane vodećih proizvođača automobila u Njemačkoj. U Francuskoj je postojao sličan projekt pod nazivom VDX (engl. *Vehicle Distributed eXecutive*) koji se pridružio prethodnom projektu, pa je stoga nastao zajednički naziv OSEK/VDX. Za implementaciju ovog standarda, dizajniran je poseban jezik nazvan OIL (engl. *OSEK Implementation Language*) i sva konfiguracija operacijskog sustava vrši se u ovom jeziku. Uvođenjem koncepta OIL jezika, promijenjen je proces kompajliranja projekta (slika 3.7.).



Slika 3.7. *Proces kompajliranja OSEK/VDX projekta* [8]

Na osnovu *.oil* datoteke vrši se generiranje *.c* i *.h* datoteka, koje se onda prosljeđuju klasičnom kompajleru. Sve postavke sustava vrše se u datoteci ekstenzije *.oil*.

ERIKA sustav se za razliku od većine sličnih RTOS-a oslanja na svega nekoliko mehanizama, te za veći prioritet pridaje performansama umjesto robusnosti [8]:

- zadatci (*task*)
- sinkronizacijski mehanizmi - događaji (engl. *event*)
- mehanizam međusobnog isključenja (engl. *resource*)
- jednokratni i periodički okidači (engl. *counter, alarm*)
- prekidi (engl. *interrupt*)
- komunikacijski servisi
- upravljanje greškama

Postoje dvije vrste zadataka – obični (engl. *basic*) i prošireni (engl. *extended*). Obični zadatak predstavlja dio programskog koda napisan u programskom jeziku *C* koji se mora izvršiti. Ako zadatak ima mogućnost čekanja na neki vanjski događaj, tada se naziva proširenim zadatkom, te u svojoj definiciji ima za to predviđeni atribut. Događajima se pristupa preko binarnih maski, gdje jedan bit registra predstavlja jedan događaj. Svaki prošireni zadatak posjeduje registar događaja koje čeka i registar događaja drugih zadataka na koje može djelovati. Mehanizam međusobnog isključenja (engl. *Mutual Exclusion*) se koristi ako je potrebno onemogućiti pristup nekom resursu od strane dvaju zadataka istovremeno. Brojači predstavljaju hardverske sklopove namijenjene brojanju vremena i obično se koriste kao okidači za neke funkcionalnosti, odnosno vrše signalizaciju nakon određenog broja otkucaja. Na brojačima se zasnivaju i alarmi, koji periodički vrše signalizaciju nakon svakih *n* otkucaja. Postoje dvije vrste prekida, jedna namijenjena običnim funkcionalnostima, a drugi za sistemske pozive (aktivacija zadatka, preuzimanje resursa, itd). Za komunikaciju između zadataka koriste se odgovarajući objekti namijenjeni za razmjenu poruka.

3.4. Integracija vlastitog rješenja

Kako bi se ostvarila tražena funkcionalnost, proces integracije svih navedenih komponenti potrebno je vršiti u nekoliko faza:

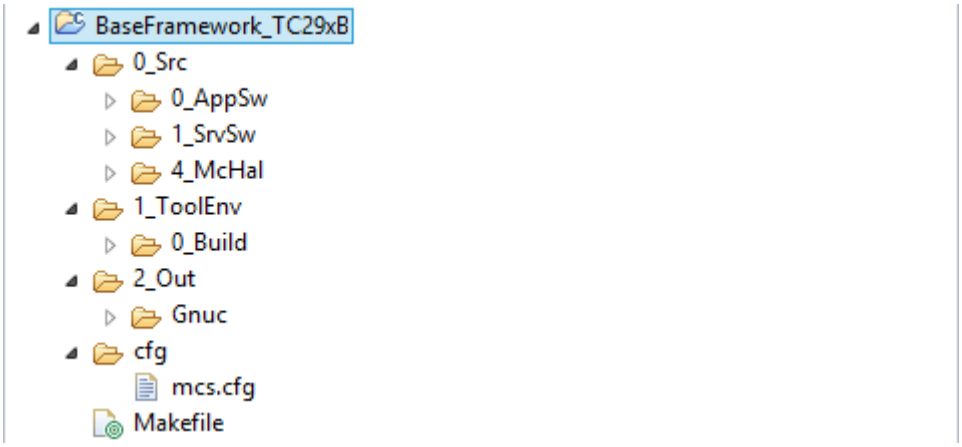
- ugradnja *Infineon* FW-a u *HighTec* razvojno okruženje
- ugradnja *ERIKA* RTOS-a u *Infineon* FW
- ugradnja lwIP-a u *ERIKA* sustav i *Infineon* FW
- prilagodba komponenti i implementacija vlastitog algoritma

Nakon što je instalirano razvojno okruženje i dodatni softver potreban za spajanje eksperimentalne ploče, može se početi s integracijom rješenja.

3.4.1. Implementacija Infineon Software Frameworka

Za korištenje ovog okvira potrebno je raspakirati njegov sadržaj i pokrenuti *install.bat* izvršnu datoteku koja će generirati ostale potrebne datoteke. Nakon što je izvršena instalacija, potrebno je izvršiti prepravke putanje koja pokazuje na razvojno okruženje u *StartFW.bat*. Kako bi se mogao koristiti navedeni okvir, razvojno okruženje je potrebno pokrenuti pomoću *StartFW.bat* datoteke.

Novi projekt s *Infineon* okvirom moguće je dobiti uvoženjem postojećeg projekta pod nazivom *BaseFramework_TC29xB* koji je preuzet s ostatkom sadržaja. Po završetku ovog koraka, dobije se projekt sljedeće strukture (slika 3.8.):

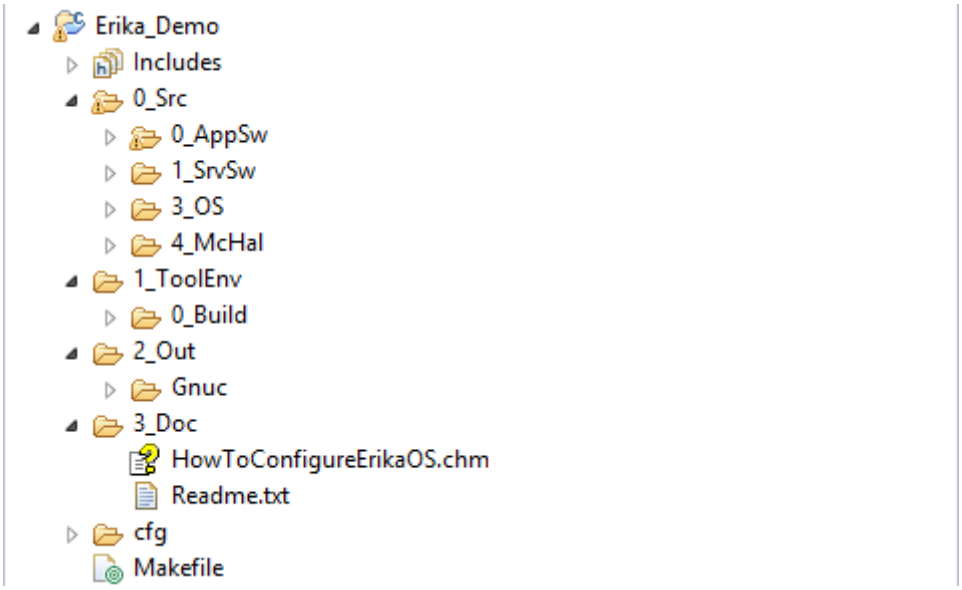


Slika 3.8. *Detaljna struktura Infineon SW projekta*

U navedenom projektu generirani su potrebni poddirektoriji za izvorne datoteke, postavke *toolchaina*, izlazne datoteke, kao i *makefileovi* potrebni za uspješno povezivanje i kompajliranje projekta. Na ovaj način uspješno je implementiran *Infineon* FW.

3.4.2. Implementacija ERIKA RTOS-a

Nakon što je implementiran potrebni programski okvir i dobivena željena struktura projekta, može se pristupiti sljedećem koraku – ugradnji operacijskog sustava. *ERIKA Enterprise* sustav prilagođen je korištenom programskom okviru, pa se na jednostavan način može ugraditi u projekt. U prethodno generiran projekt potrebno je uvesti sadržaj drugog projekta koji sadrži sve vezano za *ERIKA* sustav, a to je moguće učiniti pomoću projekta pod nazivom *ErikaOS_FrameworkSubSet*. Ovaj projekt dobije se pri preuzimanju *ERIKA* sustava. Ugradnjom novih poddirektorija, dobije se struktura kao na slici 3.9.:



Slika 3.9. *Implementacija ERIKA RTOS-a u Infineon FW*

U strukturu projekta dodan je *3_OS* poddirektorij koji sadrži datoteke potrebne za rad ovog RTOS-a. Osim toga, generiran je i *CfgErikaOS.oil*, u kojemu će se vršiti sve postavke operacijskog sustava.

U navedenoj datoteci generiran je referentni sadržaj koji opisuje korištene objekte. Sve objekte definirane od strane korisnika, pa tako i zadatke i događaje prije definiranja potrebno je deklarirati u objektu *APPMODE* . Pri definiciji zadataka, potrebno im je pridružiti događaje na koje mogu čekati, a događajima je potrebno zadati binarnu masku preko koje im se pristupa. Hardversku arhitekturu, zajedno s korištenim kompajlerom i dodatnim postavkama opisuje objekt *OS*, a svi objekti zajedno dio su centralnog objekta pod imenom *CPU*. U sljedećim poglavljima bit će prikazano kako je RTOS konfiguriran prema potrebama algoritma.

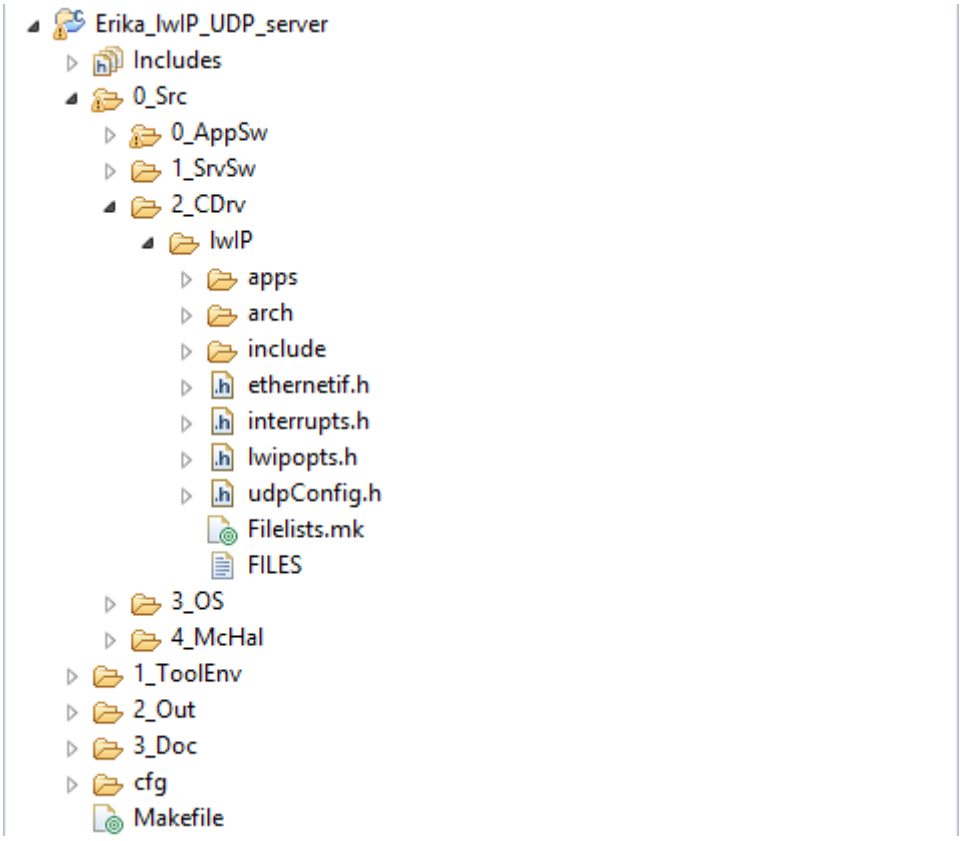
3.4.3. Implementacija lwIP-a

Za ugradnju lwIP-a u projekt, potrebno je u projekt prekopirati izvorne datoteke koje se dobiju po preuzimanju, a spremljene su u mapi *src*. Preporučuje se kopirati samo module koji će se koristiti, radi što manjeg zauzimanja ROM memorije mikrokontrolera. Za osnovne funkcionalnosti lwIP-a potrebni su sadržaji mapa *include* i *core*. Moguće je primijetiti da je posebno izdvojen najniži sloj TCP/IP stoga nazvan *netif*, a izdvojen je iz razloga jer se ovaj

dio odnosi specifično na arhitekturu koja će se koristiti, te ga je potrebno dodatno prilagoditi ako se planira konkretno korištenje fizičkog sloja (npr. Ethernet).

Nakon što su potrebni moduli kopirani u projekt, potrebno ih je povezati u pretprocesorskim putanjama kako bi se mogli uspješno koristiti. Da bi implementirani TCP/IP stog mogao raditi, potrebno je izvršiti i dodatne postavke, kako je to objašnjeno u poglavlju 2.4.

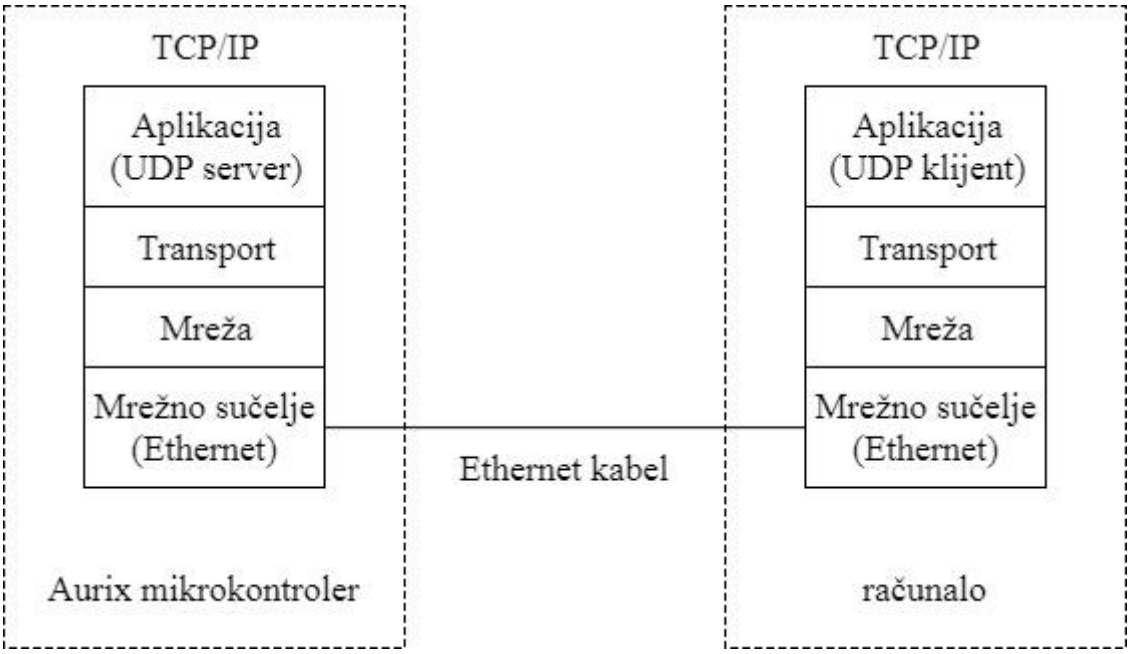
Prema preporukama *Infineon* frameworka za strukturiranje projekata, sav sadržaj vezan za lwIP preporučeno je smjestiti u mapu za kompleksne komunikacijske mehanizme – *2_CDrv*. Implementacijom lwIP stoga, u projekt su uključene sve komponente potrebne za traženu funkcionalnost, dobivena je željena struktura projekta (slika 3.10), te se može početi s realizacijom algoritma.



Slika 3.10. *Konačna struktura projekta sa svim komponentama*

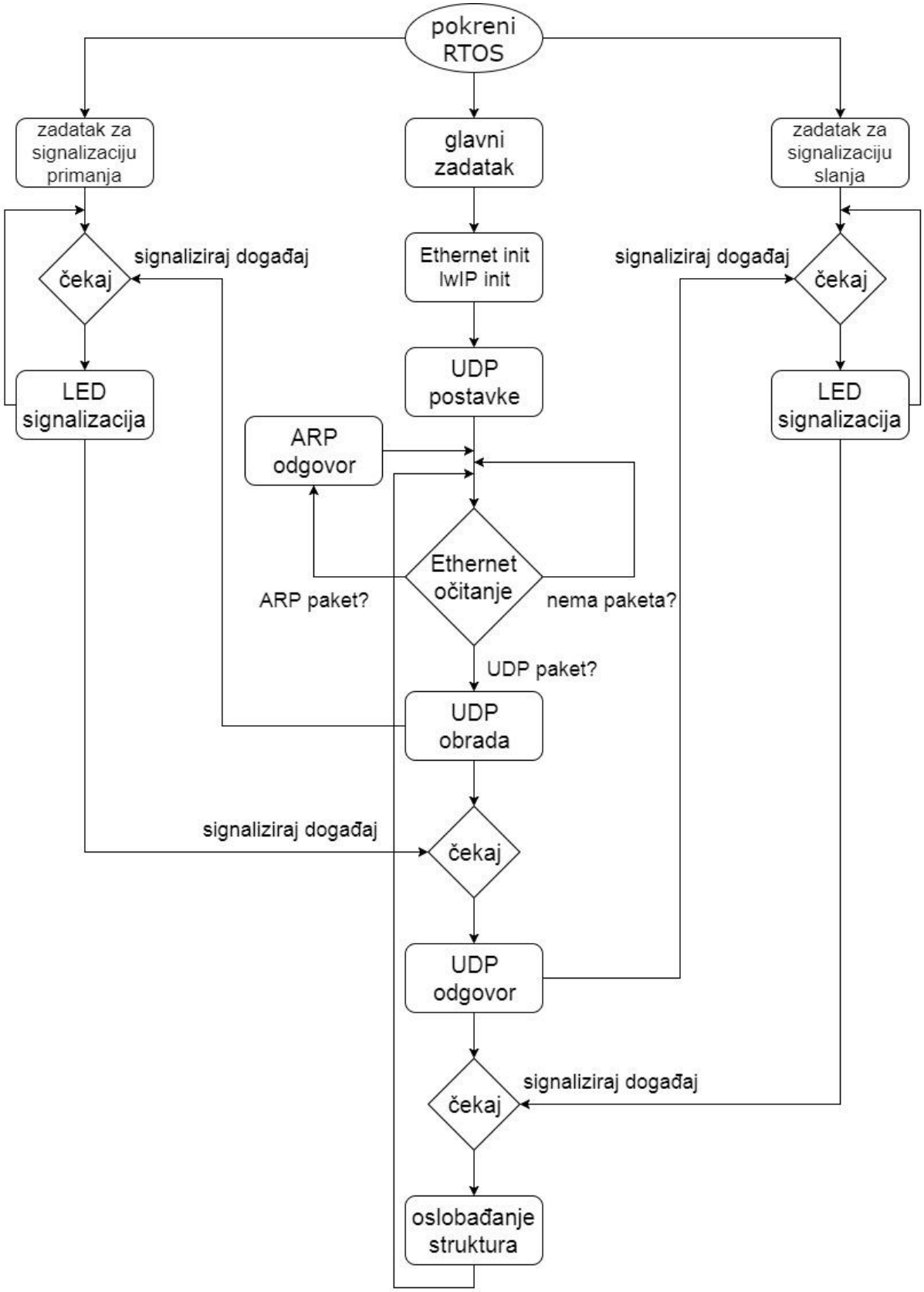
3.4.4. Prilagodba sustava i implementacija algoritma

Aplikativni scenarij koji je potrebno izraditi za potrebe diplomskog rada je funkcionalan UDP poslužitelj (engl. *server*). Za ostvarivanje tražene funkcionalnosti, potrebno je dizajnirati sustav koji ju može obaviti. Sustav koji odgovara ovim potrebama prikazan je na slici 3.11.:



Slika 3.11. *Arhitektura integriranog sustava*

Osnovna ideja je smjestiti odgovarajući program na eksperimentalnu ploču TC297A i preko Ethernet modula ju povezati s računalom na kojemu se nalazi softver s mogućnošću slanja i primanja UDP datagrama. Prije izrade rješenja, potrebno je dizajnirati algoritam za obavljanje funkcije poslužitelja, a blokovski prikaz algoritma nalazi se na slici 3.12.:



Slika 3.12. *Blokovski dijagram algoritma UDP poslužitelja*

Algoritam je zamišljen kao tri paralelna zadatka, koja su pokrenuta od početka izvođenja programa. Glavni zadatak zadužen je za direktno izvođenje algoritma, a dva sporedna zadatka koriste se za vizualnu signalizaciju na eksperimentalnoj ploči.

Glavni zadatak sastoji se iz dva bitna dijela – dio za početne postavke i petlja izvršenja. U prvom dijelu potrebno je izvršiti inicijalizaciju korištenih modula, pa su tako redom inicijalizirani Ethernet, lwIP stog i strukture potrebne za korištenje UDP-a. Nakon izvršenih postavki počinje se s izvršenjem petlje. Petlja vrši provjeru postoji li zaprimljen paket na Ethernet modulu, a onda na osnovu vrste paketa vrši neku od funkcija. Ako nema primljenog paketa, petlja ne čini ništa i vraća se na ponovnu provjeru. Ako je zaprimljen ARP paket, paket se prosljeđuje ARP modulu, koji pošiljatelja obavještava o fizičkoj, MAC (engl. *Media Access Control*) adresi mikrokontrolera. Razmjena ARP paketa je nužna prije bilo kakve komunikacije jer Ethernet zahtjeva poznavanje MAC adresa obaju sudionika komunikacije. Ako je primljen Ethernet paket s enkapsuliranim UDP datagramom, inicijaliziraju se potrebne strukture za njegovo spremanje, program obrađuje primljeni paket i aktivira zadatak za signalizaciju primanja paketa. Nakon što je obavljena signalizacija, nastavlja se izvršenje glavnog zadatka, koji zatim primljene podatke sprema u UDP datagram te isti datagram šalje nazad klijentu. Prilikom slanja aktivira se zadatak za signalizaciju slanja paketa, a njegovim završetkom, oslobađaju se korištene strukture, te se vraća na početak petlje. Izvršenjem opisanog algoritma, obavlja se funkcija *echo* UDP poslužitelja s pratećom vizualnom signalizacijom pri primanju i slanju paketa.

Nakon što je osmišljen algoritam, može se pristupiti integraciji rješenja. Za početak je potrebno prilagoditi operacijski sustav potrebama algoritma. Jedan dio postavki, vezan za postavke mikrokontrolera, generiran je prilikom ugradnje *ERIKA* RTOS-a, ali je potrebno prilagoditi dio koji se odnosi na objekte definirane od strane korisnika. Kako je već opisano u algoritmu, potrebno je definirati tri zadatka. Glavni zadatak, zadužen za funkcionalnost nazvan je *UDP_SERVER_TASK*, a pored njega koriste se još dva zadatka za signalizaciju kako je prethodno opisano. Svaki zadatak posjeduje po dva događaja, od čega je jedan namijenjen za sinkronizaciju a drugi za aktiviranje. Sinkronizacijskim događajima tijekom cijelog izvođenja programa upravlja *interrupt* modul, a aktivacijski događaji se koriste za prelazak s izvođenja jednog događaja na drugi. U sljedećem dijelu koda prikazano je kako su potrebni objekti definirani u konfiguracijskoj *.oil* datoteci:

```
APPMODE TRICORE_CPU {
 EVENT = SYNC_UDP_SERVER;
 EVENT = UDP_WAIT;

 EVENT = SYNC_RECEIVED_LED;
 EVENT = RECEIVED_WAIT;

 EVENT = SYNC_SENT_LED;
 EVENT = SENT_WAIT;

 TASK = UDP_SERVER_TASK;
 TASK = RECEIVED_LED_TASK;
 TASK = SENT_LED_TASK;

 TASK = IFX_OSTASK_BACKGROUND;
 TASK = IFX_OSTASK_INIT;

};
```

Događajima je potrebno dodijeliti binarne maske, a zatim ih navesti u deklaraciji pripadajućeg zadatka. Zadatcima je naknadno potrebno još i pridružiti odgovarajući *C* programski kod, u kojemu se točno definira što će i kako raditi. Na sljedećoj slici prikazan je način definiranja događaja i njihova ugradnja u zadatak, a ostali događaji i zadatci načinjeni na sličan način:

```
EVENT SYNC_UDP_SERVER {
    MASK = 0x01;
};

EVENT UDP_WAIT {
    MASK = 0x08;
};
TASK UDP_SERVER_TASK{
    PRIORITY = 64;
    ACTIVATION = 1;
    AUTOSTART = FALSE;
    EVENT = SYNC_UDP_SERVER;
    EVENT = UDP_WAIT;
    SCHEDULE = FULL;
    STACK = PRIVATE {
        SYS_SIZE = 256;
    };
};
```

Budući da su svi zadatci aktivni cijelo vrijeme izvođenja programa, dodijeljeni su im privatni memorijski stogovi i svojstvo preventivnosti – svojstvo da planer može izvođenje jednog zadatka zaustaviti i zamijeniti izvođenjem nekog drugog, ako je to potrebno. Ako zadatci nemaju svojstvo preventivnosti, tada planer mora čekati da se zadatak izvrši do kraja, a onda može nastaviti s drugim zadatcima. Ovo svojstvo omogućuje korištenje više zadataka s beskonačnim petljama, a upravo takvi se koriste za potrebe ovog algoritma.

Nakon što je konfiguriran RTOS koji će se koristiti, moguće je pristupiti konfiguraciji TCP/IP stoga. Postavke lwIP-a vrše se u *lwipopts.h* zaglavlju. Prilikom preuzimanja, u navedenoj datoteci već su definirane određene početne postavke:

- ARP (*LWIP_ARP*)
- IP, IP fragmentacija (*LWIP_IP*, *IP_FRAG*, *IP_REASSEMBLY*)
- *raw* IP (*LWIP_RAW*)
- UDP i UDP-Lite (*LWIP_UDP*, *LWIP_UDPLITE*)
- TCP (*LWIP_TCP*)
- statistika (*LWIP_STATS*)

Pošto je zadani protokol transportnog sloja UDP, TCP je moguće isključiti i tako bitno smanjiti količinu zauzete memorije jer je TCP najveći modul lwIP stoga. Također, koristit će se potpuna verzija UDP-a pa je moguće isključiti UDP-Lite, *raw* IP i lwIP statistiku. Pored navedenih postavki potrebno je postaviti i sljedeće opcije:

```
#define NO_SYS                          1
```

- ova opcija govori lwIP-u da niti i sinkronizacijski mehanizmi (semafori, *mutexi* i *mboxovi*) neće biti korišteni te ih nije potrebno implementirati.

```
#define MEM_ALIGNMENT                   4
```

- navedena linija definira način slaganja sadržaja programa u ROM memoriju. Za 32-bitne mikrokontrolere, vrijednost je potrebno podesiti na 4, a za 16-bitne mikrokontrolere na 2, itd.

```
#define LWIP_IPV4                       1
#define LWIP_IPV6                       0
```

- budući da računalo koristi 32-bitne IP adrese, potrebno je uključiti IPv4 modul, a isključiti IPv6 modul, kako ne bi došlo do neslaganja modula.

```
#define LWIP_DNS                        0
```

- DNS modul se ne koristi za potrebe algoritma, pa ga je potrebno isključiti.

LwIP je potrebno dodatno konfigurirati i s korisničke razine. Za prilagodbu TCP/IP stoga potrebno je definirati mrežne parametre:

- fizičku adresu mikrokontrolera
- IP adresu mikrokontrolera
- podmrežnu masku
- *gateway*
- IP adresu računala

- korištene izvorišne i odredišne portove

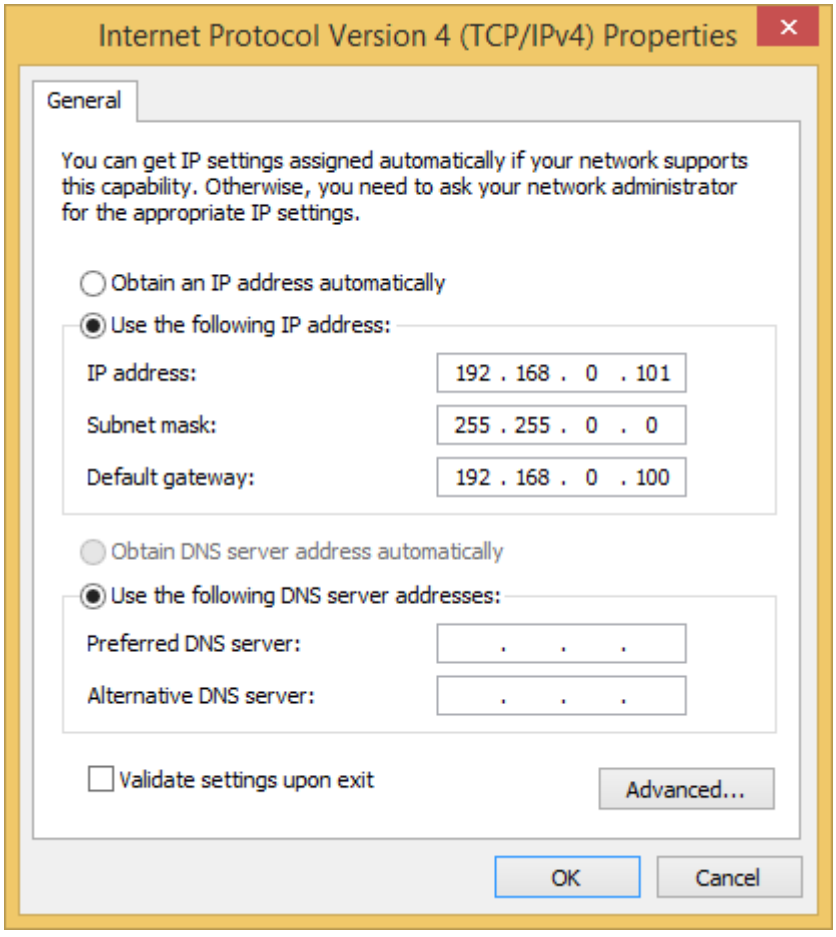
Ovi parametri definirani su u *udpConfig.h* datoteci koju je potrebno kreirati unutar projekta.

```
u8_t MAC_ADDRESS[6] = {0xAA, 0x11, 0xBB, 0x22, 0xCC, 0x33};

u8_t SOURCEADDRESS[4] = {192, 168, 0, 100};
u8_t SUBNETMASK[4] = {255, 255, 0, 0};
u8_t GATEWAY[4] = {192, 168, 0, 101};
u8_t DESTINATIONADDRESS[4] = {192, 168, 0, 101};

u16_t SOURCEPORT = 8008;
u16_t DESTINATIONPORT = 8080;
```

MAC adresa može se podesiti proizvoljno, dok se IP adresa mikrokontrolera mora slagati s IP adresom računala. Obje IP adrese moraju biti iz iste podmreže, a mikrokontroler i računalo moraju imati istu mrežnu masku, dok IP adresa mikrokontrolera treba biti *gateway* računalu i obratno. Za funkcioniranje, potrebno je podesiti mrežni uređaja računala:



Slika 3.13. *Postavke mrežnog uređaja računala*

Prije korištenja lwIP-a potrebno je još i konfigurirati najniži sloj stoga – Ethernet. Postavljanje Ethernet upravljačkog programa obavlja se u datoteci *ethernetif.c* koja u svom osnovnom obliku ima generiran približan okvir za pisanje potrebnih funkcija. Potrebno je napisati funkcije *low_level_init*, *low_level_input* i *low_level_output* prema postojećem prototipu.

U funkciju *low_level_init* prosljeđuje se struktura koja predstavlja mrežno sučelje *netif* i funkcijom potrebno je postaviti odgovarajuću MAC adresu, maksimalnu veličinu paketa i zastavice u zadanu strukturu, te pokrenuti prijemnik i predajnik Ethernet modula:

```
static void low_level_init(struct netif *netif)
{
  netif->hwaddr_len = ETHARP_HWADDR_LEN;

  netif->hwaddr[0] =  MAC_ADDRESS[0];
  netif->hwaddr[1] =  MAC_ADDRESS[1];
  netif->hwaddr[2] =  MAC_ADDRESS[2];
  netif->hwaddr[3] =  MAC_ADDRESS[3];
  netif->hwaddr[4] =  MAC_ADDRESS[4];
  netif->hwaddr[5] =  MAC_ADDRESS[5];

  netif->mtu = 1500;

  netif->flags = NETIF_FLAG_BROADCAST | NETIF_FLAG_ETHARP
          | NETIF_FLAG_LINK_UP;

  ETH_Start();

}
```

Low_level_input radi na principu provjere Ethernet *descriptora* koji signaliziraju postoji li paket na ulazu. Ako postoji, paket se kopira u predviđenu strukturu i šalje u dalje prema lwIP-u.

```
static struct pbuf * low_level_input(struct netif *netif)
{
      struct pbuf *q;
      uint16 length = 0;
      uint16 framelength = 0;

      if (ETH_pRxDescr->RDES0.A.OWN == 0)
      {
          unsigned char *pRxBuf = ETH_getReceiveBuffer(&framelength);
          s_p = pbuf_alloc(PBUF_RAW, framelength, PBUF_POOL);

          for(q = s_p; q != NULL; q = q->next)
          {
             memcpy(q->payload, &pRxBuf[length], q->len);
             length = length + q->len;
          }
          ETH_freeReceiveBuffer();
      }
      return s_p;
}
```

Low_level_output funkcija kao argumente dobija pokazivač na mrežno sučelje i strukturu s podatcima namijenjenim za prijenos, a zatim se podatci kopiraju u Ethernetov spremnik za slanje, te se poziva funkcija za slanje paketa.

```
static err_t low_level_output(struct netif *netif, struct pbuf *p)
{
  struct pbuf *q;
  uint16 framelength = 0;
  unsigned char *pTxBuf = NULL;

  while (pTxBuf == NULL)
  {
      pTxBuf = (unsigned char*)ETH_getTransmitBuffer();
  }

  for(q = p; q != NULL; q = q->next)
  {
     memcpy(&pTxBuf[framelength], q->payload, q->len);
     framelength = framelength + q->len;
  }

  ETH_sendTransmitBuffer(framelength);
  return ERR_OK;
}
```

Način i raspored pozivanja prethodno objašnjenih funkcija, već je definiran u ostatku Ethernet modula i nije potrebno voditi brigu o tome. Ključ za uspješno povezivanje mikrokontrolera na

mrežu je periodičko očitavanje zaprimljenih poruka koje mogu sadržavati ARP pakete, bez kojih povezivanje ne bi bilo moguće. Funkcija koja obavlja navedenu zadaću nazvana je *ethernetif_input*, a pozivanje ove funkcije trebao bi biti sastavni dio algoritma.

```
err_t ethernetif_input(struct netif *netif)
{
    do
    {
        s_p = low_level_input( netif );
        if (s_p)
        {
            if (ERR_OK != netif->input( s_p, netif))
            {
                pbuf_free(s_p);
                s_p = NULL;
            }
        }
    }
    while (s_p);
}
```

Proces prilagodbe Etherneta predstavlja najveći problem pri procesu prilagodbe lwIP stoga za neku arhitekturu. Nakon što je obavljena prilagodba, moguće je početi s integriranjem algoritma.

Prema blokovskom dijagramu algoritma sa slike 3.12. prvi korak pri integraciji algoritma je aktiviranje svih potrebnih zadataka. Zadatci se aktiviraju u inicijalizacijskom zadatku *IFX_OSTASK_INIT* koji je neophodan dio RTOS-a:

```
ActivateTask(UDP_SERVER_TASK);
ActivateTask(RECEIVED_LED_TASK);
ActivateTask(SENT_LED_TASK);
```

Zadatci se nakon aktivacije stavljaju u stanje čekanja i koordinirani su sinkronizacijskim *interruptom*. Gotovo cijeli algoritam implementiran je u glavni, UDP zadatak, stoga će on biti fokus objašnjavanja. Za rad s lwIP stogom, potrebno je nakon definiranja korištenih varijabli, inicijalizirati Ethernet i sam lwIP stog:

```
ETH_Init();
lwip_init();
```

Funkcija za inicijalizaciju lwIP-a, već postoji i dio je lwIP stoga, te ju potrebno mijenjati, dok je funkcija za inicijalizaciju Etherneta preuzeta iz Ethernet iLLD-a, i napravljene su manje izmjene. U *Eth_Init* funkciji vrši se postavljanje ulaznih i izlaznih pinova modula, konfiguracija registara potrebnih za rad i kreiranje prethodno spomenutih *descriptora*. Sljedeći korak u integraciji algoritma jest postavljanje mrežnog sučelja.

Mrežno sučelje prezentira najniži sloj protokolskog stoga i predstavljeno je strukturom podataka *netif*. Ova struktura podataka je neophodna za bilo kakav rad sa lwIP stogom, a sučelje se konfigurira sljedećim dijelom koda:

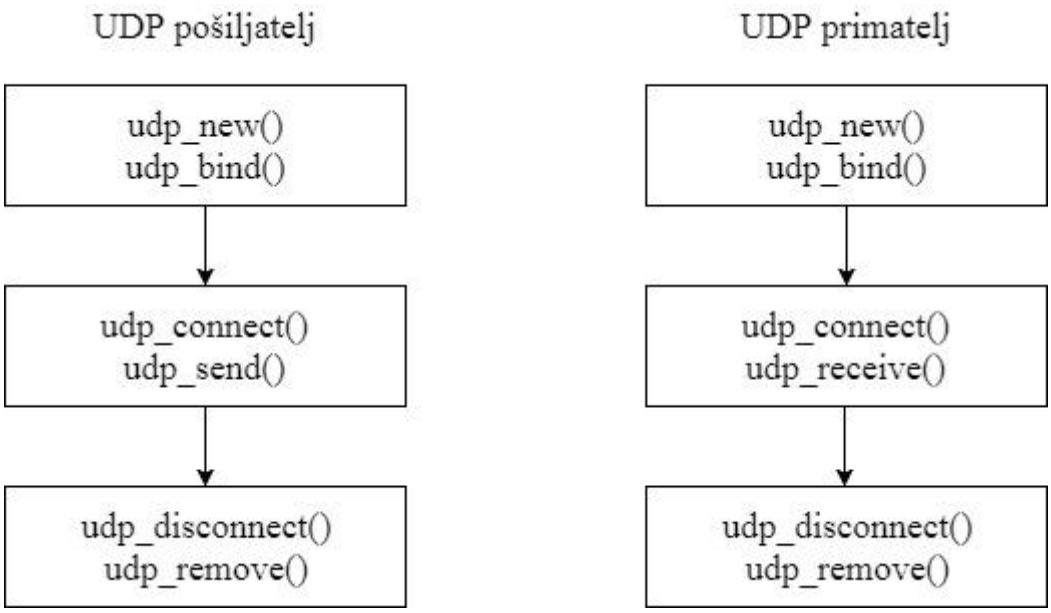
```
netif_add(&netifx, &sourceAddress, &subnetMask,
        &gateway, NULL, &ethernetif_init, &netif_input);
netif_set_default(&netifx);
netif_set_up(&netifx);
```

Strukturi se prosljeđuju mrežni parametri, funkcija kojom se inicijalizira sučelje i funkcija koja obrađuje ulazne poruke. Pošto je moguće imati više struktura ovog tipa, onu koju će se trenutno koristiti potrebno je postaviti kao zadanu, te na njoj izvršiti dodatne postavke.

Budući da će se vršiti slanje i primanje UDP datagrama, potrebno je i inicijalizirati strukture potrebne za korištenje tog protokola. Struktura korištena za potrebe UDP protokola naziva se UDP PCB (engl. *Process Control Block*). Za rad s UDP-om implementirane su sljedeće funkcije:

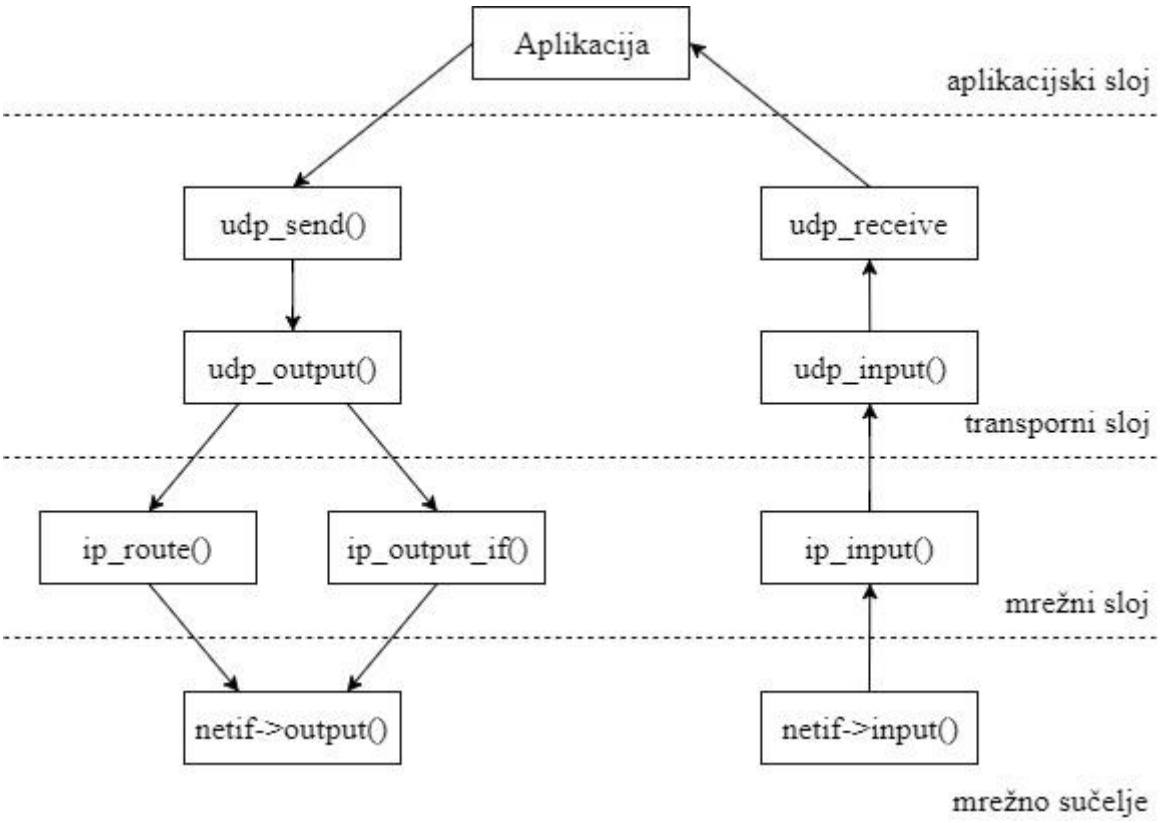
- *udp_new()* – kreiranje novog PCB bloka
- *udp_remove()* – brisanje PCB bloka
- *udp_bind()* – povezivanje PCB bloka s lokalnom IP adresom i portom
- *udp_connect()* – povezivanje s drugim UDP korisnikom
- *udp_disconnect()* – prekid veze s drugim korisnikom
- *udp_send()* – funkcija za slanje UDP paketa preko određene konekcije
- *udp_recv()* – procesuiranje primljenog UDP paketa

Pomoću ovih funkcija i njihovih implementacija moguće je ostvariti potpunu funkcionalnost UDP protokola. Slika 3.14. pokazuje pravilan redoslijed pozivanja spomenutih funkcija:



Slika 3.14. *Redoslijed pozivanja UDP API funkcija*

Za potrebe poslužitelja potrebno je koristiti funkcije *udp_send* i *udp_recv*. Ove dvije funkcije su funkcije najviše razine i cijeli algoritam se svodi na njihovu pravilnu upotrebu.



Slika 3.15. *Arhitektura lwIP-a* [3]

Na slici 3.15. prikazana je struktura lwIP-a. Poštivanjem navedenih pravila i strukture lwIP-a, može se početi s implementacijom UDP-a u centralni algoritam:

```
udp_pcbx = udp_new();
error = udp_bind(udp_pcbx, &sourceAddress, sourcePort);
error = udp_connect(udp_pcbx, &destinationAddress, destinationPort);
```

Izvršenjem ovih funkcija, gotov je inicijalizacijski dio programa te se ulazi u petlju čija je zadaća korištenje funkcija više razine i obavljanja funkcije UDP poslužitelja. U prvom dijelu petlja čita pakete s Etherneta, inicijalizira spremnike za podatke, te vrši provjeru radi li se o UDP paketu:

```
ethernetif_input(&netifx);
udp_recv(udp_pcbx, udpRecvFunction, NULL);

pbufx = pbuf_alloc(PBUF_TRANSPORT, bufferLength, PBUF_POOL);
```

Ako je primljeni paket doista UDP paket, iz njega se očitava podatak (engl. *payload*) i vrši se LED signalizacija predviđena za primanje UDP datagrama.

```
if(udpData != NULL)
{
    Ifx_OSTask_SetEvent(RECEIVED_LED_TASK, 8);
    Ifx_OSTask_ClearEvent(8);
    Ifx_OSTask_WaitEvent(8);

    MEMCPY(pbufx->payload, udpData, bufferLength);
    error = udp_send(udp_pcbx, pbufx);
```

Nakon što je primljen paket i izvršena signalizacija, primljeni podatak se sprema u spremnik za slanje i vrši se slanje identične poruke nazad prema pošiljatelju. Na ovaj način kreiran je *echo* poslužitelj.

```
    if(error == ERR_OK)
    {
        Ifx_OSTask_SetEvent(SENT_LED_TASK, 8);
        Ifx_OSTask_ClearEvent(8);
        Ifx_OSTask_WaitEvent(8);
    }
}
pbuf_free(pbufx);
udpData = NULL;
```

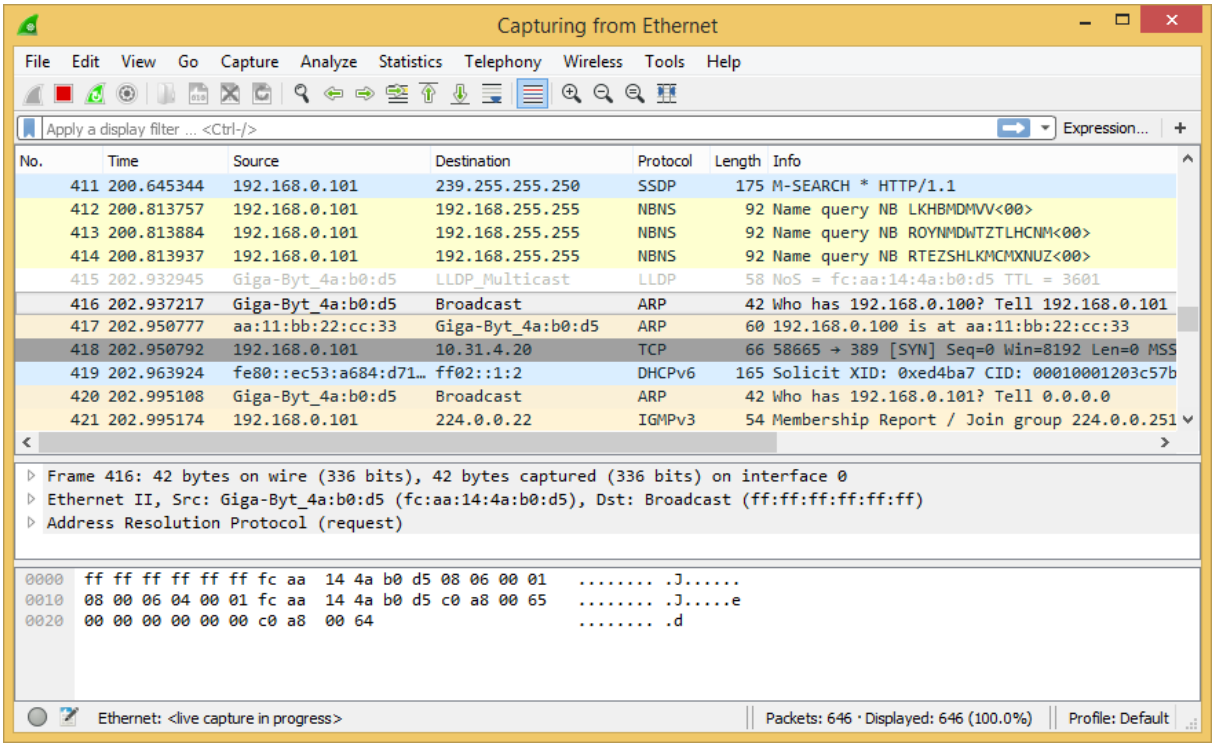
Ako je podatak uspješno poslan nazad, pokreće se signalizacijski zadatak, te oslobađaju korišteni spremnici za podatke.

```
udp_disconnect(udp_pcbx);
udp_remove(udp_pcbx);
```

Na kraju, ako dođe do prekida izvršenja petlje, vrši se proces deinicijalizacije, odnosno prekidanje veze i brisanje PCB-a. Ovako napisan program uspješno obavlja funkciju UDP *echo* poslužitelja i moguće ga je prilagoditi za različite upotrebe. Nakon što je završen proces integracije rješenja, može se pristupiti prezentaciji rezultata.

4. PREZENTACIJA RJEŠENJA

Kako bi se ispitala funkcionalnost prethodno integriranog sustava, potrebno je spojiti sustav kako je prikazano na slici 3.11. Sustav je moguće ispitati provjerom mrežnog prometa na strani računala, a za ovu potrebu korišten je otvoreni softver za analizu mrežnog prometa *Wireshark*. Na stranu poslužitelja, tj. *Aurix* mikrokontroler, potrebno je u razvojnom okruženju pomoću UDE-a ugraditi razvijeno rješenje. Pošto još uvijek nije spojena klijent strana, ovako ugrađeno rješenje prolazit će kroz prethodno objašnjenu petlju, ali budući da nema UDP datagrama na ulazu, mikrokontroler će samo mreži dati informacije o svojoj IP adresi, a korištenjem ARP poruka i informaciju o MAC adresi. Provjerom *Wireshark* sučelja, moguće je vidjeti periodičke ARP poruke od strane računala i odgovor mikrokontrolera. Ova razmjena poruka prikazana je na slici 4.1.:

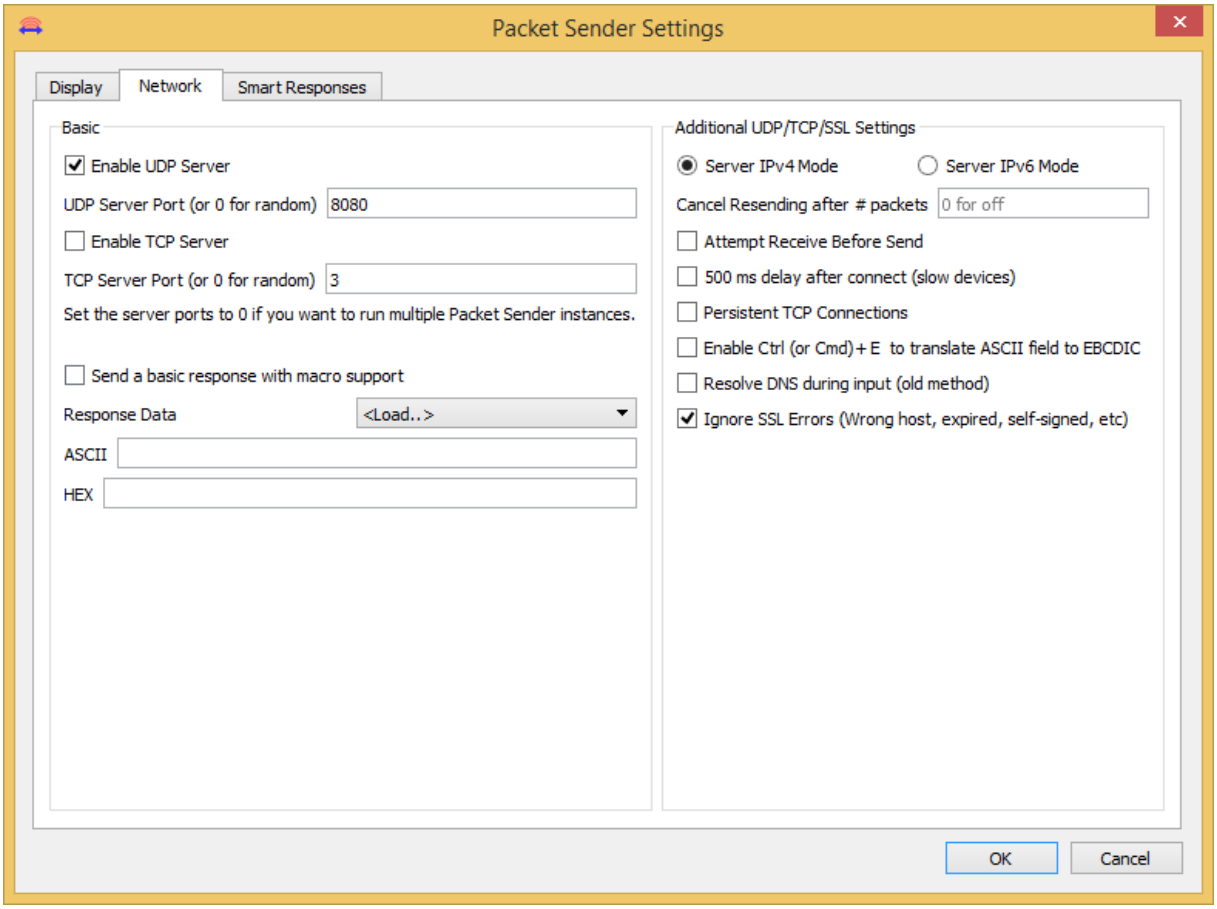


Slika 4.1. *Razmjena ARP poruka zabilježena u Wireshark sučelju*

U linijama 416 i 417 prikazana je razmjena ARP poruka. Vidljivo je da mrežna kartica računala prema svim korisnicima (*broadcast*) u mreži šalje upit o tome tko se nalazi na IP adresi 192.168.0.100, te traži da se ta informacija pošalje na sučelje s IP adresom 192.168.0.101. U sljedećoj poruci, s mikrokontrolera stiže informacija da je tražena IP adresa

povezana s navedenom MAC adresom. Ako je ova razmjena poruka uspješna, mikrokontroler je uspješno spojen na mrežu i može se početi s razmjenom UDP datagrama.

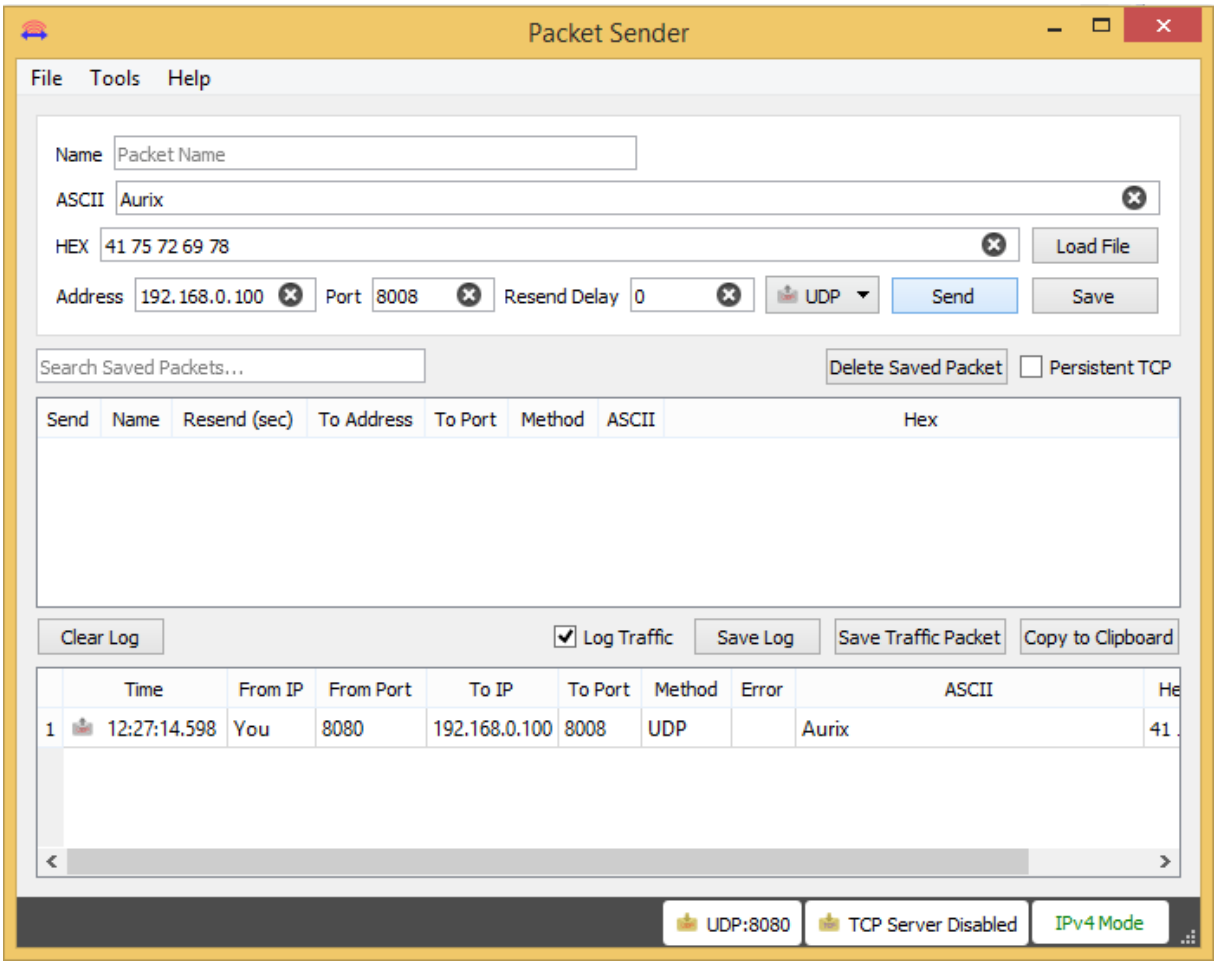
Implementirani algoritam nikada neće prvi početi sa slanjem UDP paketa, stoga je potrebno od strane računala započeti ovu komunikaciju. Za jednostavnije slanje ovakvog mrežnog prometa moguće je koristiti gotove programske pakete namijenjene za slanje različitih vrsta mrežnog prometa. Postoje brojni programi za ovu namjenu, a za operativni sustav *Windows* može se koristiti program pod nazivom *Packet Sender*. Spomenuti programski alat potrebno je prije korištenja podesiti tako da radi u IPv4 načinu, postaviti ga na odgovarajuću IP adresu i port, prema postavkama *udpConfig.h* datoteke na mikrokontroleru. Postavke *Packet Sender* sučelja prikazane su na slici 4.2.:



Slika 4.2. *Postavke Packet Sender sučelja*

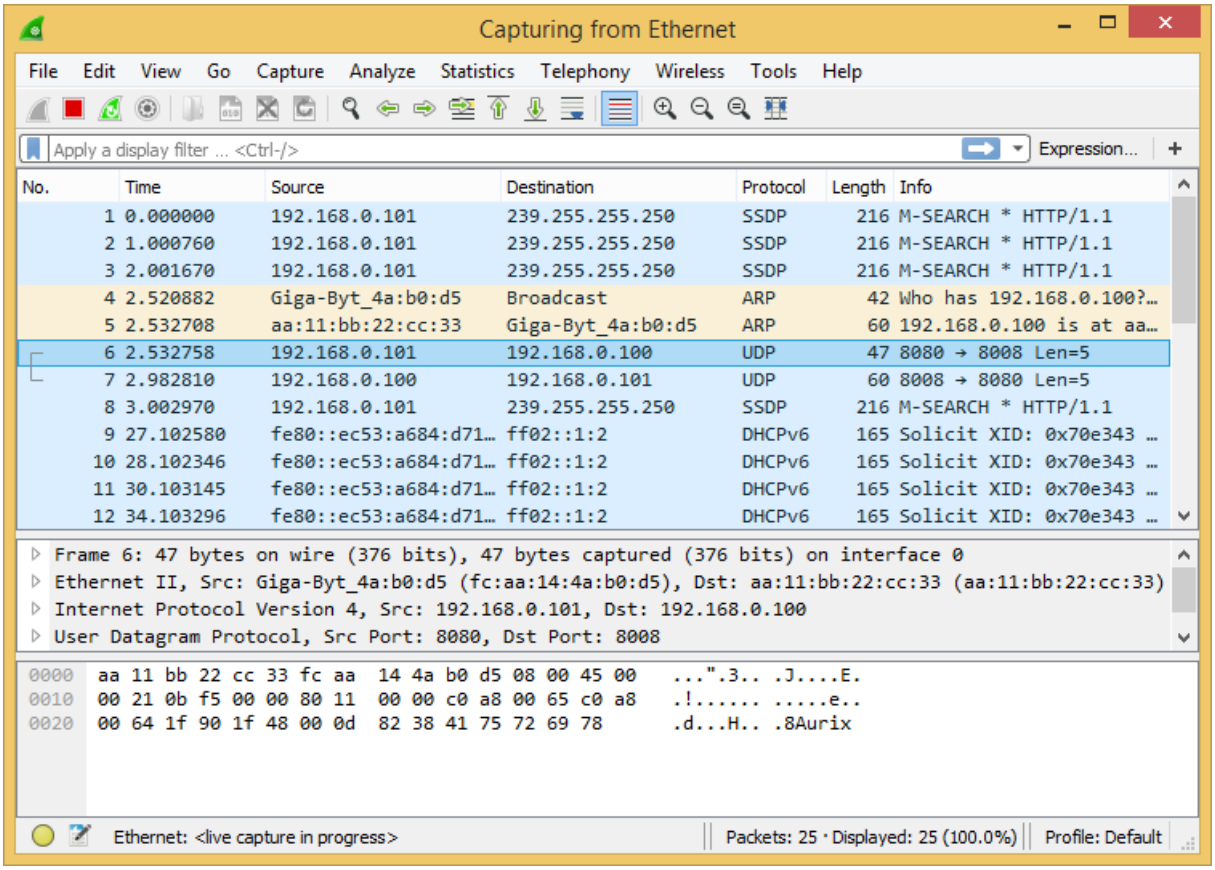
Konfiguracijom klijentske strane, završen je proces konfiguracije sustava te je moguće početi s testiranjem UDP poslužitelja smještenog na mikrokontroleru. Za slanje paketa potrebno je još i postaviti IP adresu odredišta i odredišni port, te napisati konkretnu poruku koja se

prenosi. Poruke se mogu slati jednokratno ili se periodički slati u određenim intervalima. Za potrebe testiranja moguće je poslati samo jedan paket i čekati na odgovor (slika 4.3.):



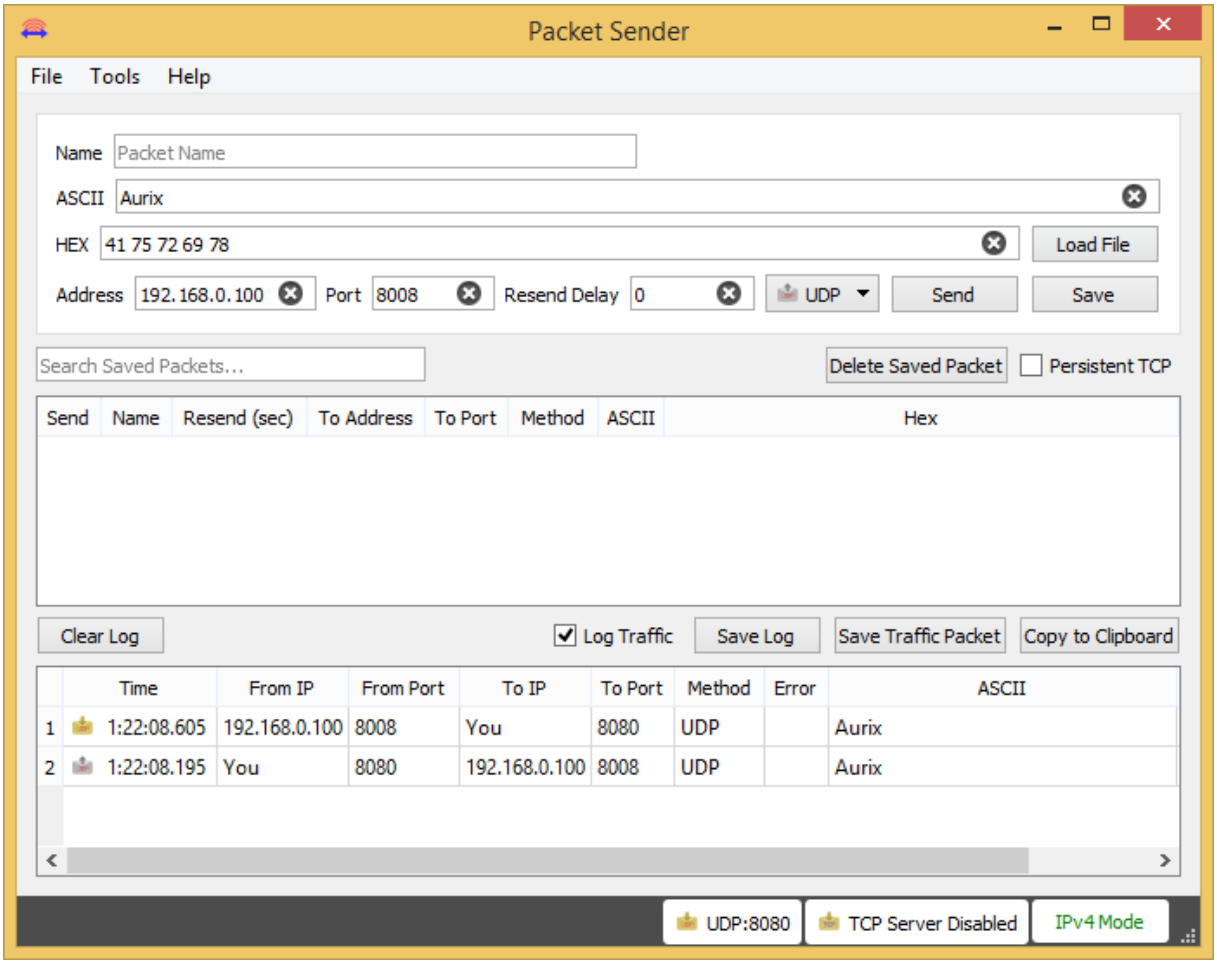
Slika 4.3. *Slanje UDP datagrama preko Packet Sendera*

U donjem prozoru moguće je pratiti ulazni i izlazni promet mrežnog sučelja koje se koristi. Pritiskom tipke *Send* poslani paket se pojavio u spomenutom prozoru, što znači da je paket ispravno poslan na mrežni uređaj. U zapisniku je prikazano vrijeme slanja, IP adrese, portovi i sama poruka koja je poslana. Za testiranje ispravnosti, prijenos paketa moguće je provjeriti u *Wireshark* sučelju. Na slici 4.4. prikazan je novonastali promet Ethernet modula:



Slika 4.4. *Zabilježena razmjena UDP paketa u Wiresharku*

U *Wiresharku* moguće je vidjeti paket o kojemu se prethodno govorilo, ali su paketi detaljnije interpretirani. Iz detaljnijeg prikaza paketa, vidljivo je da je poslani UDP paket enkapsuliran prvo u IP zaglavlje, a zatim zajedno s IP zaglavljem u Ethernet paket, baš kako je prethodno objašnjeno. Po primitku paketa, na mikrokontroleru se izvršila signalizacija implementirana u algoritmu. Nakon što je završena signalizacija, u zapisniku prometa pojavio se dolazni UDP paket poslan od strane mikrokontrolera, te je mikrokontroler počeo sa signalizacijom poslanog paketa. Analiziranjem paketa, može se vidjeti ista struktura kao i kod poslanog paketa, a jedina razlika je u duljini zaglavlja, koja je drugačije definirana u Ethernet iLLD-u. *Infineon* iLLD nakon dijela koji sadrži podatke, po potrebi dodaje dodatne bitove za popunjavanje (engl. *padding*). Duljina zaglavlja ne utječe na ispravnost prijema jer je duljina poslanog i primljenog podatka jednaka, što je jedino bitno za funkcionalnost aplikacije. Po primitku paketa, paket se sprema i u zapisnik *Packet Sendera*, kako je to i prikazano na slici 4.5.:



Slika 4.5. *Prikaz uspješno zaprimljenog odgovora u Packet Senderu*

Datagram je uspješno stigao u *Packet Sender* sučelje te se može početi s mjerenjem.

4.1. Mjerenje performansi i pouzdanosti

Wireshark i *Packet Sender* alati evidentiraju i vrijeme slanja i primanja paketa. Na ovaj način moguće je vršiti mjerenje performansi algoritma. Bitno je napomenuti da ogroman dio kašnjenja u razmjeni podataka unosi ugrađena signalizacija, pa ju je za mjerodavne rezultate pri mjerenjima potrebno izbaciti iz algoritma. Nakon što je signalizacija uklonjena iz algoritma, može se početi s testiranjem.

Kako bi se provjerila uspješnost algoritma i izmjerile performanse, izvršena su tri testa. U svakom testu vrši se periodičko slanje 50 UDP paketa, te se mjeri vrijeme odziva i postotak izgubljenih paketa. Prvo testiranje provedeno je s periodom slanja od 0.5 s, drugo s 1 s, a treće s periodom 2 s. Dobiveni rezultati prikazani su u tablicama 4.1., 4.2. i 4.3.:

Broj paketa	Vrijeme slanja [s]	Vrijeme primitka [s]	Vrijeme odziva [s]
1	42.756	42.803	0.047
2	43.152	43.162	0.010
3	43.590	/	/
4	44.027	44.060	0.033
5	44.479	44.511	0.032
6	44.917	44.948	0.031
7	45.354	45.401	0.047
8	45.777	45.808	0.031
9	46.215	46.262	0.047
10	46.652	46.684	0.032
11	47.092	47.139	0.047
12	47.531	47.578	0.047
13	47.969	48.016	0.047
14	48.409	48.456	0.047
15	48.848	48.895	0.047
16	49.288	49.335	0.047
17	49.726	49.789	0.063
18	50.165	50.212	0.047
19	50.607	50.654	0.047
20	51.046	51.109	0.063
21	51.487	/	/
22	51.931	51.994	0.063
23	52.356	52.418	0.062
24	52.794	52.857	0.063
25	53.236	53.298	0.062
26	53.673	53.736	0.063
27	54.106	54.184	0.078
28	54.544	54.607	0.063
29	54.982	55.045	0.063
30	55.423	55.502	0.079
31	55.862	55.925	0.063
32	56.302	56.380	0.078
33	56.739	56.786	0.047
34	57.179	57.242	0.063
35	57.618	57.696	0.078
36	58.058	58.120	0.062
37	58.480	58.559	0.079
38	58.919	58.997	0.078
39	59.368	59.446	0.078
40	59.813	59.891	0.078
41	0.257	0.351	0.094
42	0.696	0.774	0.078
43	1.121	1.199	0.078
44	1.562	1.641	0.079
45	2.000	2.094	0.094
46	2.438	/	/
47	2.878	/	/
48	3.316	3.394	0.078
49	3.754	3.832	0.078
50	4.183	4.277	0.094

Tablica 4.1. *Rezultati mjerenja s periodom slanja 0.5 s*

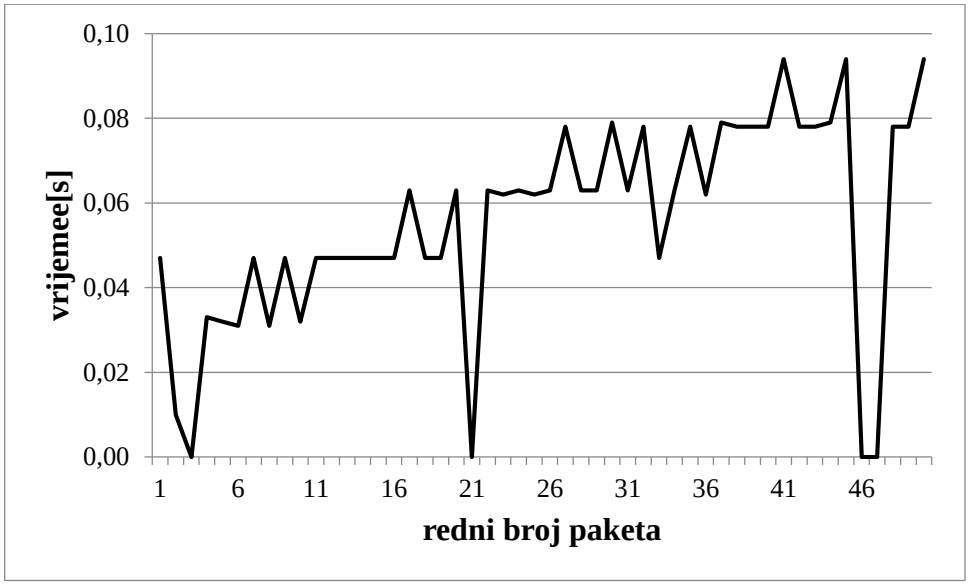
Broj paketa	Vrijeme slanja [s]	Vrijeme primitka [s]	Vrijeme odziva [s]
1	15.452	15.465	0.013
2	16.461	16.476	0.015
3	17.449	17.465	0.016
4	18.435	18.466	0.031
5	19.406	/	/
6	20.400	20.447	0.047
7	21.380	21.411	0.031
8	22.366	22.397	0.031
9	23.343	23.390	0.047
10	24.333	24.379	0.046
11	25.319	25.366	0.047
12	26.306	26.337	0.031
13	27.294	27.341	0.047
14	28.271	28.318	0.047
15	29.257	29.304	0.047
16	30.243	30.306	0.063
17	31.228	31.290	0.062
18	32.213	32.259	0.046
19	33.201	33.217	0.016
20	34.171	34.234	0.063
21	35.162	35.225	0.063
22	36.148	36.211	0.063
23	37.136	37.183	0.047
24	38.121	38.183	0.062
25	39.096	39.158	0.062
26	40.086	40.148	0.062
27	41.062	41.062	0.000
28	42.049	42.112	0.063
29	43.036	43.098	0.062
30	44.027	44.089	0.062
31	45.002	45.080	0.078
32	45.988	/	/
33	46.977	47.055	0.078
34	47.966	48.028	0.062
35	48.957	49.036	0.079
36	49.946	50.008	0.062
37	50.933	/	/
38	51.919	51.998	0.079
39	52.891	/	/
40	53.877	53.939	0.062
41	54.863	/	/
42	55.849	55.927	0.078
43	56.835	56.913	0.078
44	57.819	57.898	0.079
45	58.805	/	/
46	59.794	/	/
47	0.780	0.874	0.094
48	1.778	1.871	0.093
49	2.772	/	/
50	3.758	3.827	0.069

Tablica 4.2. *Rezultati mjerenja s periodom slanja 1 s*

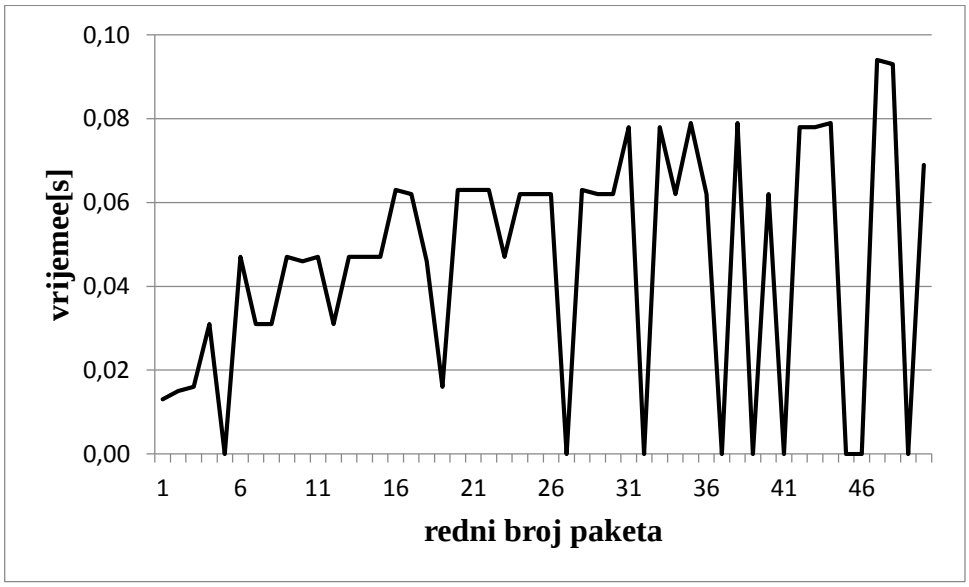
Broj paketa	Vrijeme slanja [s]	Vrijeme primitka [s]	Vrijeme odziva [s]
1	45.292	45.341	0.049
2	47.193	47.224	0.031
3	49.168	49.184	0.016
4	51.130	51.161	0.031
5	53.104	/	/
6	55.069	55.101	0.032
7	57.042	57.073	0.031
8	59.005	59.036	0.031
9	0.978	1.009	0.031
10	2.948	2.995	0.047
11	4.903	/	/
12	6.880	/	/
13	8.846	8.877	0.031
14	10.819	10.866	0.047
15	12.793	12.839	0.046
16	14.751	14.798	0.047
17	16.721	16.768	0.047
18	18.702	18.749	0.047
19	20.679	20.726	0.047
20	22.669	22.716	0.047
21	24.638	24.685	0.047
22	26.622	26.669	0.047
23	28.603	28.666	0.063
24	30.564	30.611	0.047
25	32.541	32.604	0.063
26	34.513	/	/
27	36.495	36.557	0.062
28	38.479	/	/
29	40.446	/	/
30	42.412	42.474	0.062
31	44.386	/	/
32	46.359	46.422	0.063
33	48.326	48.404	0.078
34	50.310	50.373	0.063
35	52.302	52.380	0.078
36	54.268	54.331	0.063
37	56.272	56.350	0.078
38	58.251	58.329	0.078
39	0.220	/	/
40	2.190	/	/
41	4.172	4.250	0.078
42	6.152	/	/
43	8.109	8.172	0.063
44	10.091	10.185	0.094
45	12.074	/	/
46	14.058	14.152	0.094
47	16.014	16.092	0.078
48	18.008	18.086	0.078
49	19.973	20.051	0.078
50	21.942	22.036	0.094

Tablica 4.3. *Rezultati mjerenja s periodom slanja 2 s*

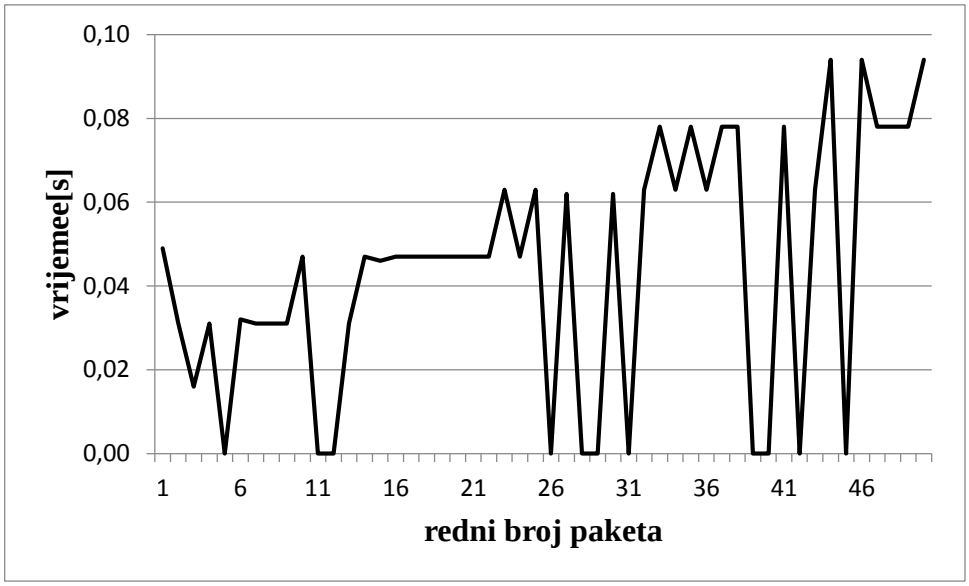
Kako bi se rezultati mjerenja lakše interpretirali i bili pregledniji, izrađeni su pripadajući grafovi za svako testiranje. Navedeni grafovi prikazani su na slikama 4.6., 4.7. i 4.8.:



Slika 4.6. *Grafički prikaz mjerenja s periodom slanja 0.5 s*



Slika 4.7. *Grafički prikaz mjerenja s periodom slanja 1 s*



Slika 4.8. *Grafički prikaz mjerenja s periodom slanja 2 s*

U dobivenim grafovima, radi lakše uočljivosti, paketi koji nisu uspješno isporučeni prikazani su kao da imaju vrijeme odziva od 0 s, a taj način svaki nagli pad vremena odziva predstavlja izgubljeni paket i ponovnu retransmisiju. Iz grafova je vidljivo da je vrijeme odziva nepredvidivo i ima velike oscilacije, a u pravilu se povećava s vremenom provedenim u testiranju.

Nakon što su dobiveni rezultati mjerenja, i iscrtani potrebni grafovi moguće je doći do određenih zaključaka. U rezultatima mjerenja je vidljivo da sve poruke nisu isporučene, a razlog toga je činjenica da je UDP protokol koji ne jamči sigurnu isporuku svih paketa. Broj neuspješno isporučenih paketa najveći je za najveći period slanja, a najmanji za najmanji period slanja. Iako je bila očekivana suprotna situacija, ovim mjerenjem je dokazano da i za najmanji odabrani period slanja, algoritam ima dovoljno vremena za izvršenje, te broj neuspješnih transmisija ne ovisi o algoritmu, već o mreži. Veći broj neuspješno primljenih datagrama u testiranju s većim periodom slanja može se interpretirati većim zagušenjem u mreži tijekom tog testiranja, a mogućnost pojave zagušenja je veća za duže testiranje. Broj izgubljenih paketa i postotak ispravno dostavljenih paketa prikazan je u tablici 4.4.:

Period slanja [s]	Izgubljenih paketa	Postotak točnog izvođenja	Prosječno vrijeme odziva [s]
0.5	4/50	92.00%	0.061
1	8/50	84.00%	0.053
2	11/50	78.00%	0.056

Tablica 4.4. *Statistički prikaz rezultata mjerenja*

Vrijeme odziva u testovima ima velika odstupanja od srednje vrijednosti i u pravilu je sve veće što se test više približava kraju. Velika odstupanja nastaju zbog opterećenosti obaju procesora drugim procesima, a pošto jezgre procesora mogu obavljati samo jedan zadatak istovremeno, moguće je da testirani program bude u stanju čekanja. Budući da je program vrlo jednostavan i nije velik, izvodi se u svega nekoliko desetaka milisekundi, pa ostali procesi kojima se procesori bave značajno utječu na duljinu izvođenja i tako unose pogrešku. Vrijeme odziva ni u jednom testiranom slučaju nije iznosilo više od 100 ms, što znači da bi program trebao imati zadovoljavajući postotak uspješnog odziva na pakete poslane u periodu većem od 200 ms.

Postotak točno dostavljenih paketa od strane algoritma i prosječno vrijeme odziva, kako se može zaključiti iz dobivenih mjerenja, ovise o vanjskim čimbenicima poput zagušenja u mreži i opterećenosti procesora. Dobiveno rješenje je stoga, u odnosu na periodiču razmjenu poruka, pogodnije za jednokratnu razmjenu poruka.

5. ZAKLJUČAK

TCP/IP protokolski stog je slojeviti model povezivanja protokola čiji početci sežu još u kraj 1960-tih. Nastao je za potrebe postojeće sigurnosne infrastrukture nazvane *ARPAnet*, a zbog dobrih svojstava postao je temelj za razvoj današnje najveće globalne mreže – Interneta. Budući da ima široku primjenu, postoje i brojne implementacije TCP/IP-a koje je moguće koristiti, posebno u namjenskim sustavima. Za potrebe izrade odgovarajućeg algoritma odabrana je lwIP implementacija TCP/IP stoga. LwIP je besplatan i otvorenog koda, a prilagođen je radu s manjim mikrokontrolerima koji su 16-bitni ili 32-bitni.

Za hardver, korišten je mikrokontroler TC297 iz serije *Aurix*, proizvođača *Infineon*, koji je 32-bitni mikrokontroler s tri jezgre, brzinom od 300 MHz i 8 MB radne memorije. Pri razvoju softvera korišteno je razvojno okruženje *HighTec* s pripadajućim alatima, a za operativni sustav koristi se *ERIKA Enterprise* RTOS.

Proces izrade konačnog rješenja izveden je u nekoliko faza, prilagodbom svih korištenih komponenti. LwIP stog prilagođen je tako da se koriste samo potrebni protokolski moduli, kako bi se zauzelo što manje memorije, a prilagodba je napravljena u *lwipopts.h* zaglavlju. Osim prilagodbe zaglavlja, budući da se za najniži sloj TCP/IP stoga koristi Ethernet, potrebno je bilo i prilagoditi cijeli Ethernet modul prema arhitekturi mikrokontrolera. *ERIKA* RTOS je u sustav ugrađen prema postojećem primjeru, u *Infineon* programski okvir i konfiguriran prema potrebama algoritma, a korištena su tri zadatka tijekom cijelog njegovog izvođenja. Nakon što su implementirane sve komponente, izrađeno je programsko rješenje koje ima funkcionalnost UDP *echo* poslužitelja.

Budući da je UDP protokol koji ne jamči sigurnu dostavu svih datagrama kroz mrežu, pri testiranju je vidljivo da su se određeni paketi izgubili. Na ovu pojavu se ne može utjecati od strane mikrokontrolera, jer gubitak paketa ovisi od zagušenja u mreži i opterećenja mrežnih uređaja. Brzina odziva mikrokontrolera, također ovisi o opterećenju procesora i zbog kratkotrajnog izvođenja, na nju se ne može utjecati.

Dobiveno rješenje zadovoljava traženu funkcionalnost, a za postotak odziva se može reći da je veći od 75 %. Integrirano rješenje obavlja funkciju razmjene UDP datagrama, pa se može zaključiti da se daljnjim razvojem na aplikacijskoj TCP/IP razini, dobiveno rješenje može iskoristiti kao razvojna podloga za brojne primjene poput HTTP (engl. *Hypertext Transfer Procotol*) poslužitelja.

LITERATURA

[1] W.R. Stevens, **TCP/IP Illustrated, Volume 1 – The protocols**, Addison-Wesley, SAD, 1994.

[2] https://www.utwente.nl/ewi/dacs/assignments/completed/bachelor/reports/B-assignment_vanderPloeg.pdf (23.8.2017.)

[3] A. Dunkels, **Design and Implementation of the lwIP TCP/IP stack**, Swedish Institute of Computer Science, Švedska, 2001.

[4] https://savannah.nongnu.org/projects/lwip/ (23.8.2017.)

[5] https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-tm-microcontroller/aurix-tm-family/channel.html?channel=db3a30433727a44301372b2eefbb48d9 (24.8.2017.)

[6] https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-tm-microcontroller/aurix-tm-family/aurix-tm-family-%E2%80%93-tc29xt/channel.html?channel=db3a304342c787030142dc92c9aa1674 (24.8.2017)

[7] http://erika.tuxfamily.org/drupal/documentation.html (26.8.2017.)

[8] http://igm.univ-mlv.fr/~masson/Teaching/IMC5-1EO/osek.pdf (26.8.2017.)

[9] https://technet.microsoft.com/en-us/library/bb726993.aspx (1.9.2017.)

SAŽETAK

Diplomski rad opisuje integraciju sustava koji za rezultat ima funkcionalnost UDP poslužitelja. Na početku su objašnjeni osnovni pojmovi vezani za TCP/IP stog, a zatim su opisane komponente korištene za izradu sustava.

Nakon što su dana potrebna objašnjenja, opisan je postupak spajanja cijelog sustava i implementacije algoritma kako bi se dobila tražena funkcionalnost.

Na dobivenom sustavu izvršena su testiranja u cilju mjerenja pouzdanosti i performansi, te je na osnovu dobivenih rezultata moguće odrediti korisnost i primjenu sustava.

Ključne riječi: TCP/IP, lwIP, Aurix, Infineon, TC297, ERIKA, RTOS, UDP, poslužitelj

ABSTRACT

This master thesis describes the integration process of the system which is used as UDP server. At the beginning, some of the basic terms about TCP/IP suite were explained, along with the explanation of components used for the system integration.

After the required explanation has been given, the process of system integration has been described, along with the algorithm implementation.

Integrated system has been used for performance and reliability measurements, which has provided the answers about the utility and the application of the system.

Keywords: TCP/IP, lwIP, Aurix, Infineon, TC297, ERIKA, RTOS, UDP, server

ŽIVOTOPIS

JOSIP BABIĆ

Rođen je 11. siječnja 1995. godine u Hanau, Njemačka. Osnovnu školu završava 2008. godine u Globarici i upisuje srednju elektrotehničku školu u Katoličkom školskom centru „Don Bosco“ u Žepču.

Srednju školu završava 2012. godine, te iste godine upisuje preddiplomski studij elektrotehnike na „Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek“.

U rujnu 2015. godine završava preddiplomski studij i upisuje diplomski studij komunikacija i informatike na istom fakultetu.

U Osijeku, rujna 2017.

_____

Josip Babić