

Elektrotehnički fakultet

Univerzitet u Banjoj Luci

**IZVJEŠTAJ PROJEKTNOG ZADATKA**

iz predmeta

**Industrijske komunikacione mreže**

Tema: Proširenje funkcionalnosti UART drajvera

Student: Mentor:

Dejan Milojica 1150/16 Profesor: doc. dr Mladen Knežić

Jul, 2020. godine

Sadržaj:

[Uvod 2](#_Toc47080870)

[Uvodna razmatranja problema 2](#_Toc47080871)

[Razmatranje rješenja 3](#_Toc47080872)

[Realizacija rješenja 3](#_Toc47080873)

[Proširenje UART drajvera 3](#_Toc47080874)

[Proširenje GPIO drajvera 5](#_Toc47080875)

[Korisnička aplikacija 6](#_Toc47080876)

[Literatura 7](#_Toc47080877)

# Uvod

## Uvodna razmatranja problema

Korišćenjem standardne komunikacione mreže, sa jednim Klijentom (Raspberry Pi), te jednim serverom (Sklop sa četiri releja), pri čemu je za komunikaciju zadužen RS-485 transiver, što se obavlja preko prenosnog medijuma sačinjenog od dvije upredene žice, pri čemu svaka od njih obezbjeđuje prenos u jednu stranu, dolazimo do problema koji se bazira na “tromosti” izlaznog GPIO pina Raspberry Pi-a, čija je osnovna funkcionalnost definisanje, preko RS-485 transivera, smjera prenosa podataka. Vrijednost pina, definiše smijer prenosa, odnosno, defineše koja od dvije žice će biti aktivna, čime će prenos kroz nju biti omogućen u jednu stranu, pri čemu je njihova aktivacija komplementarna, tako da će u jednom trenutku samo jedna biti aktivna. Ukoliko je vrijednost izlaznog pina “1”, aktivira se žica koja vrši prenos podataka ka relejnim izlazima, čime se odgovarajući relejni izlaz uključuje/isključuje, što definiše poruka koja se prenosi ka izlazu, odnosno ukoliko je “0”, vrši se čitanje podataka sa relejnog izlaza, odnosno prihvatanje odgovora(response) sa releja.

Za prvobitnu realizaciju C programa koji se izvršava na Raspberry Pi platformi, te obezbjeđuje korektnu komunikaciju u vidu Modbus mreže između Klijenta I Servera, koristi se posebna biblioteka **<modbus.h>**, koja pored funkcija za uspostavljanje veze između pojedinačnih strana, te naravno, prenosa podataka, pruža i kontrolu GPIO pina za definisanje smjera prenosa, usljed čega dolazi do prvobitnih problema sa tromosti pina, jer se po završetku prenosa podataka sa RPI-a ka relejnim izlazima, očekuje prihvatanje potvrde i odgovora od strane servera, a to zahtjeva prebacivanje pina u stanje „0“, odnosno stanje prihvatanja podataka i aktiviranje žice koja služi za prenos podataka ka RPI-u. Usljed navedene tromosti, podaci sa servera počinju da pristižu prije nego je vrijednost pina ažurirana, i prenosni medijum aktivan za prenos, što dovodi do „narušavanja“ određenog dijela dolaznih podataka. Takođe, ono što predstavlja dodatni problem, je zahtjevanje poznavanja vremena transfera podataka od klijenta ka serveru, kako bi stanje izlaznog pina ostalo u stanju „1“, sve dok prenos traje, te po njegovom završetku, prebacivanje u stanje prihvatanja podataka. Potencijalno, kao još jedan vid nedostatka prethodne realizacije, možemo posmatrati i to da je korisniku preko navedene biblioteke omogućen pristup ne samo za kontrolu vrijednosti odgovarajućeg pina, već i njegovo eksportovanje u /sys/class, podešavanje direction-a i slično, odnosno potpuna kontrola nad GPIO pinovima RPI platforme u kernel prostoru.

# Razmatranje rješenja

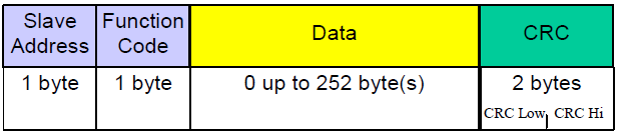
Ideja rješenja kojom bismo poboljšali funkcionalnost, i u najboljem slučaju riješili prethodno razmotrene probleme, bazira se na tome, da upravljanje pinovima Raspberry Pi platforme, iz korisničkog prostora(**User space**), prebacimo u kernel prostor(**Kernel space**). Na taj način, vrijeme odziva prilikom definisanja vrijednosti izlaznog pina, kao I bilo kakvo upravljanje nad pinom, biće značajno kraće, čime bismo obezbjedili efikasniji rad Sistema, koji će uz pravovremene promjene stanja pina, obezbjediti korektnu komunikaciju. Pored toga, iz korisničke aplikacije, korisnik ne bi imao direktan pristup pinovima, već bi taj pristup bio indirektan preko odgovarajućeg Linux kernel modula, koji će nam prethodno I omogućiti. U skladu sa prethodnim, ideja se bazira na tome da realizujemo kernel modul, koji će obezbjeđivati komunikaciju, kao i standardni modul za serijsku komunikaciju, te pružiti dodatnu mogućnost upravljanja nad GPIO pinovima RPI platforme. Kako se iz korisničkog prostora ne pristupa pinovima direktno, neophodno će biti I obezbjediti njihovo eksportovanje u /sys/class, mogućnost promjene stanja, kao I potencijalnu zamjenu pina(Kernel modul ćemo jednom učitati, dok korisnički program možemo pokretati više puta, uz različite parametre, među kojima je I pin za komunikaciju, tako da ne bismo za svaku promjenu parametara C programa, morali svaki put isčitavati/učitavati kernel modul).

# Realizacija rješenja

## Proširenje UART drajvera

Prenos datih podataka obavlja se serijski, korišćenjem standardnog serijskog UART prenosa, odnosno TX I RX linije Raspberry Pi-a. U skladu sa tim, rješavanje uvodno definisanih problema, bazira se na korišćenju već postojećeg UART drajvera, te njegovom proširenju, odnosno “Proširenom UART drajveru”. Dati drajver, pored standardnog, treba da na adekvatan način omogući indirektno upravljanje, preko proširenog GPIO drajvera, sa odgovarajućim pinom, odnosno da omogući pravovremen upis odgovarajućeg izlaznog bita, u zavisnosti od toga da li vršimo slanje ili prihvatanje podataka, kao i njegovu zamjenu. U skladu sa tim, realizacija drajvera se bazira na tome, da unutar funkcije za inicijalizaciju drajvera, vršimo kreiranje klase uređaja „chardev“(karakterski uređaj), te na osnovu dinamički dodjeljenih Major i Minor brojeva, korišćenjem funkcije „**device\_create**“, formiramo odgovarajući fajl koji reprezentuje dati drajver, „/dev/prosireni\_uart“. Kako dati drajver treba da pruži funkcjonalnosti koje podržava standardni UART drajver, najlakše ćemo to postići tako što otvorimo originali UART fajl „/dev/ttyAMA0“, i sve što se upiše u naš kreirani fajl, se zapravo prosljeđuje u originalni, čime se ostvaruje potrebna komunikacija. Za otvaranje fajlova unutar kernel prostora, koristimo funkciju filp\_open, koja je definisana unutar biblioteke **<fs.h>**, te kao rezultat izvršavanja, vraća file descriptor otvorene datoteke, ili kod greške. Za zatvaranje prethodno otvorene datoteke, koristimo funkciju filp\_close, koja kao argument zahtjeva file descriptor otvorene datoteke.

Ono što karakteriše serijski prenos, su parametri serijskog prenosa, kao što su Baud rate, Parity, Start bit, Stop bit i slično, te se za korektan prenos, zahtjeva njihovo podešavanje. U skladu sa tim, to ćemo obavljati korišćenjem IOCTL funkcije originalnog UART-a, kome ćemo kao argument proslijediti strukturu “termios”, koja je definisana unutar biblioteke **<termios.h>**, te sadrži sve neophodne parametre. Data IOCTL funkcija, zahtjeva I komandu za izvšavanje, što može da bude TCGETS koji predstavlja čitanje date structure, ili TCSETS, što reprezentuje podešavanje parametara prenosa na osnovu proslijeđene strukture. Unutar inicijalizacije modula, vršimo postavljanje parametara serijskog prenosa na unaprijed definisane vrijednosti, što predstavlja default-ne vrijednosti, dok će se po uspostvljanju izvršavanja korisničke aplikacije, na osnovu korisničkih parametara I IOCTL funkcije proširenog UART drajvera, izvšiti njihovo ažuriranje. Preklopljene funkcije za upis, odnosno čitanje podataka (device\_ write / device\_read), kao što je navedeno, imaju zadatak da sve što se upiše u prošireni UART, najčešće iz korisničkog prostora preko funkcije “**copy\_from\_user**”, se prebaci u otvoreni fajl originalnog UART drajvera, odnosno u slučaju čitanja, vrši se čitanje sadržaja originalnog UART, te njihovo prebacivanje u korisnički prostor. Međutim, ono što se dodatno zahtjeva, je kontrola GPIO pina, kojim definišemo operaciju. Prije nego se vrši upis podataka, zahtjeva se postavljanje stanja pina u logičku 1, čime sugerišemo RS-485 transiveru da se podaci šalju ka relejnom sklopu. Po završetku slanja podataka, pristiže odgovor sa relejnog sklopa, koji treba da se pročita, što zahtjeva prebacivanje vrijednosti pina u logičku 0. Za kontrolu pina, koristimo funkciju “**gpio\_set\_configure**”, koja pored što ima mogućnost kontrole stanja, ima mogućnost promjene pina za korišćenje. Data funkcija komunicira sa GPIO proširenim modulom, preko IOCTL funkcije, čime se izvršava prethodno zahtjevano. U skladu sa tim, pored otvaranja fajla originalnog UART drajvera, zahtjeva se I otvaranje fajla proširenog GPIO drajvera, kojim vršimo adekvatne GPIO konfiguracije. Ono što u prethodnoj situaciji kod upisa, a zatim čitanja, predstavlja problem, je to što vrijednost pina treba da bude “1” sve dok se podaci ne pošalju, odnosno da se uspostavi stanje “0”, po zavšetku slanja, a prije početka prijema. U skladu sa tim, zahtjeva se poznavanje vremena transfera, slanja podataka ka relejnim izlazima, što određujemo na osnovu izlazne brzine slanja originalnog UART drajvera, broja bita po karakteru, te broja bajtova koji se šalje. Proračun vremena “kašnjenja” promjene stanja pina, vršimo unutar funkcije “**time\_of\_data\_transfer**”, koja vraća vrijeme izraženo u [µs].

**

*Slika 1.1: Format modbus poruke.*

Obezbjeđivanje datog kašnjenja promjene stanja pina, vršimo pomoću funkcije „**usleep\_range(usec min, usec max)**“, za koju se pokazuje da je od funkcija koje obezbjeđuju kašnjenje, najbolja ukoliko se kašnjenje kreće u opsegu [10us-10ms]. Za baud rate 9600, pokazuje se na osnovu prethodne formule, da je vrijeme prenosa 8328us, što je 8.3ms, čime je zadovoljen prethodni uslov za korišćenje funkcije, odnosno za sve ostale veće baud rate-ove, vrijeme prenosa će biti manje, ali unutar datog opsega.

U slučaju čitanja podataka sa proširenog UART drajvera, pri čemu je GPIO pin postavljen u stanje „0“, vršimo čitanje podataka sa originalnog UART drajvera, te njihovo prosljeđivanje iz kernel prostora, u korisnički prostor, za šta koristimo funkciju „**copy\_to\_user**“, nakon čega vršimo ažuriranje stanja pina u „1“, jer se očekuje ponovno slanje podataka preko RS-485 transivera.

Po završetku korišćenja datog proširenog UART modula, vrši se njegovo uklanjanje, što zahtjeva dealokaciju svih zauzetih resursa, odnosno,

* Zatvaranje otvorenih datoteka (originalni UART, te prošireni GPIO).
* Uklanjanje uredjaja iz sistema, nije vise aktivan za koriscenje. (cdev\_del).
* Uklanjanje odgovarajuce datoteke uredjaja ( device\_destroy).
* Uklanjanje odgovarajuce klase uredjaja u /sys/class (class\_destroy).
* Dealokacija Major I Minor brojeva uredjaja (unregister\_chrdev\_region).

## Proširenje GPIO drajvera

Za konfiguraciju, I rad sa odgovarajućim pinovima Raspberry Pi platforme, koristimo prošireni GPIO drajver. Za inicijalizaciju drajvera, koristimo iste funkcije kao I u prethodnom slučaju, čime ostvarujemo dinamičku alokaciju brojeva (Major I Minor), kreiranje klase uređaja (class), kreiranje odgovarajuće datoteke uređaja (/dev/gpio\_driver), te povezivanje uredjaja sa operacijama, odnosno njegovo aktiviranje. Pošto osnovni zadatak datog drajvera, treba da bude pristupanje odgovarajućen PINu, odnosno definisanje njegovog smjera, kao i upisane vrijednosti, unutar inicijalizacije, vršimo “request-ovanje” odgovarajućeg pina, njegovo eksportovanje u /sys/class/gpio čime će postati vidljiv, te postavljanje smjera u “izlaz/out”, odnosno početne vrijednosti u “1”. U slučaju greške prilikom izvršavanja neke od pomenutih funkcija, zahtjeva se dealokacija svega alociranog.

Konfiguracija odgovarajućeg pina, vrši se korišćenjem IOCTL funkcije “**gpio\_ioctl**”, koja pruža prethodne dvije navedene funkcije (Promjena vrijednosti/ Promjena Pina). U slučaju promjene vrijednosti, smjer ostaje I dalje OUT, ali je vrijednost definisana sa ulaznim argumentum funkcije, što može da bude “1” ili “0”. U slučaju druge komande, zahtjeva se upotreba nekog drugog pina, što zahtjeva dealociranje prethodno zauzetog pina, te alociranje novog pina.

Po zavšetku korišćenja, vršimo dealokaciju zauzetih resursa, kao što smo to radili kod prethodno definisanog drajvera, s tim što se ovdje dodatno zahtjeva dealokacija zauzetog PIN-a!

## Korisnička aplikacija

Korišćenje aplikacije “modbus-master.c”, kao što je navedeno, stvaralo je određenu grešku prilikom prenosa podataka, koja se ogledala u tromosti, I kašnjenju GPIO pina. U skladu sa tim, uvođenjem prethodno definisanih proširenja, ostvaruje se modifikacija, I formira struktura korisničkog programa, na sljedeći način.

Kao I u originalnom programu, imamo stvaranje modbus konteksta, putem funkcije ” **modbus\_new\_rtu**”, koja zahtjeva kao ulazne argumente, naziv serijskog fajla, što je fajl proširenog UART drajvera, te parametre serijskog prenosa, odnosno Baud rate, Data bits, Parity te Stop bit. Dati parametri, kao I svi parametri koji će biti korišćeni u sljedećim funkcija, definisani su unutar header fajla “parametri.h”. Unutar date funkcije, vrši se otvaranje datog fajla sa adekvatno proslijeđenim parametrima prenosa, što će se reprezentovati pozivom IOCTL funkcije proširenog UART drajvera, koji će kao što je navedeno, pozvati IOCTL funkciju sa datim parametrima I datom komandom, originalnog UART drajvera, I na taj način izvršiti podešavanje serijskog prenosa. Unutar date funkcije, pored prethodnog, kreira se element strukture “modbus\_t”, koja sadrži file descriptor otvorenog fajla, pokazivače na funkcije koje se obavljaju nad datim elementom, te niz dodatnih stvari. Među datim funkcijama, nalaze se funkcije:

* **modbus\_set\_slave(modbus\_t, SERVER\_ID))** – postavljanje adrese slejva, što je “SERVER\_ID”.
* **modbus\_set\_debug(modbus\_t, TRUE)** – omogućenje prikazivanja debug informacija.
* **modbus\_connect(modbus\_t)** – uspostavljanje konekcije sa slejvom.

Modifikacija datog korisničkog programa, u odnosu na originalni, bazira se na tome da se PINu za omogućenje, kao i upravljanje prenosom podataka, ne pristupa iz korisničkog prostora, već se sve to obavlja u kernel prostoru, na način koji je prethodno pojašnjen. U skladu sa tim, funkcije za omogućenje RPI pina, njegovu konfiguraciju, te eksportovanje/uneksportovanje više nisu neophodne, ioako njihovo korišćenje, zapravo ne bi trebalo uticati na rad datih modula, kao I samog izvšavanja. Kako se definisanje PINa za kontrolu prenosa obavlja u korisničkom prostoru, od strane korisnika, zahtjeva se adekvatno prosljeđivanje u kernel prostor izabranog pina, što se obavlja korišćenjem funkcije “**bcm\_pin\_de\_configuration**”, koja obezbjeđuje IOCTL poziv prosirenog UART drajvera, uz komandu za promjenu PINa, te nudi provjeru uspješnosti konfiguracije pina. Pored prethodne, imamo I funkciju “**uart\_serial\_param\_data\_transfera**”, čiji je osnovni zadatak proslijeđivanje parametara serijskog prenosa, ali samo onih koji su neophodni za proračunavanje vremena transfera podataka između Klijenta I Servera, odnosno RPIa I Relejnog sklopa. Takođe, proslijeđivanje se obavlja preko IOCTL funkcije, uz adekvatnu komandu za prenos. Kod svih prenosa iz korisničkog, u kernel prostor, unutar odgovarajućih modula, koristi se funkcija “**copy\_to\_user**”. Funkcija “**modbus\_write\_bit**”, kao I u originalnom slučaju, vriši adekvatno slanje podataka, što reprezentuje poruku kojom se uključuje ili isključuje odgovarajući relejni izlaz, koji je definisan preko parametra “COIL\_ADDRESS”.

# Literatura

1. A. Šabić, *Proširenje funkcionalnosti UART drajvera na Linux baziranim sistemima*, Master rad April 2020, pristupljeno Jul 2020 godine
2. *Bootlin*, adresa: <https://bootlin.com/>, pristupljeno Jul 2020 godine
3. *Driving Me Nuts – Things You Never Should Do in the Kernel*, adresa: <https://www.linuxjournal.com/article/8110>, pristupljeno Jul 2020. godine
4. *The Linux Kernel Timers*, adresa: https://www.kernel.org/doc/html/latest/timers , pristupljeno Jul 2020 godine
5. *RS-485 Max*, adresa: <https://www.alldatasheet.com/view.jsp?Searchword=RS485&sField=4>, pristupljeno Jul 2020 godine
6. *The Modbus Organization*, adresa: https://www.modbus.org/, pristupljeno Jul 2020 godine
7. *Libmodbus*, adresa: <https://docs.ros.org/api/libmodbus/html/modbus_8h_source.html>, pristupljeno Jul 2020 godine