|  |  |  |
| --- | --- | --- |
| |  | | --- | | **УНИВЕРЗИТЕТ У НОВОМ САДУ**  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  **НОВИ САД**  **Департман за рачунарство и аутоматику**  **Одсек за рачунарску технику и рачунарске комуникације**  **ЗАВРШНИ (BACHELOR) РАД**  **Кандидат: Масалушић Стефан**  **Број индекса: РА193 - 2014**  **Тема рада: ПРИКУПЉАЊЕ ДИЈАГНОСТИЧКИХ ПОДАТАКА О РАДУ**  **ДВОЖИЛНОГ ЕТЕРНЕТ КОМУНИКАЦИОНОГ ИНТЕРФЕЈСА – Broadr-Reach**  **Ментор рада: Доц.Др Богдан Павковић**  **Нови Сад, јун, 2018.** | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| Редни број, **РБР**: | |  | |
| Идентификациони број, **ИБР**: | |  | |
| Тип документације, **ТД**: | | Монографска документација | |
| Тип записа, **ТЗ**: | | Текстуални штампани материјал | |
| Врста рада, **ВР**: | | Завршни (Bachelor) рад | |
| Аутор, **АУ**: | | Стефан Масалушић | |
| Ментор, **МН**: | | Доц. Др Богдан Павковић | |
| Наслов рада, **НР**: | | Прикупљање дијагностичких података о раду двожилног етернет  комуникационог интерфејса – BroadR-Reach | |
| Језик публикације, **ЈП**: | | Српски / латиница | |
| Језик извода, **ЈИ**: | | Српски | |
| Земља публиковања, **ЗП**: | | Република Србија | |
| Уже географско подручје, **УГП**: | | Војводина | |
| Година, **ГО**: | | 2018 | |
| Издавач, **ИЗ**: | | Ауторски репринт | |
| Место и адреса, **МА**: | | Нови Сад; трг Доситеја Обрадовића 6 | |
| Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога) | |  | |
| Научна област, **НО**: | | Електротехника и рачунарство | |
| Научна дисциплина, **НД**: | | Рачунарска техника | |
| Предметна одредница/Кqучне речи, **ПО**: | | RTOS, VxWorks, етернет, комуникација, BroadR-Reach, дијагностика, RTP | |
| **УДК** | |  | |
| Чува се, **ЧУ**: | | У библиотеци Факултета техничких наука, Нови Сад | |
| Важна напомена, **ВН**: | |  | |
| Извод, **ИЗ**: | | Написати код који прикупља циљане дијагностичке податке о току етернет комуникације преко „BroadR-Reach“ чипа, на начин да се подаци прикупљају у кернелу RTOS-a, затим прослеђују „RT“ процесу који обрађује примљене податке по предвиђеној логици. Поред тога направити апликацију на страни персоналног рачунара која визуелизује прикупљене податке. | |
| Датум прихватања теме, **ДП**: | |  | |
| Датум одбране, **ДО**: | |  | |
| Чланови комисије, **КО**: | Председник: |  |
|  | Члан: |  | Потпис ментора |
|  | Члан, ментор: |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Accession number, **ANO**: | |  | |
| Identification number, **INO**: | |  | |
| Document type, **DT**: | | Monographic publication | |
| Type of record, **TR**: | | Textual printed material | |
| Contents code, **CC**: | | Bachelor Thesis | |
| Author, **AU**: | | Stefan Masalušić | |
| Mentor, **MN**: | | Bogdan Pavković, Phd | |
| Title, **TI**: | | Collecting diagnostic data over one UTP cable ethernet communication interface – BroadR-Reach | |
| Language of text, **LT**: | | Serbian | |
| Language of abstract, **LA**: | | Serbian | |
| Country of publication, **CP**: | | Republic of Serbia | |
| Locality of publication, **LP**: | | Vojvodina | |
| Publication year, **PY**: | |  | |
| Publisher, **PB**: | | Author’s reprint | |
| Publication place, **PP**: | | Novi Sad, Dositeja Obradovica sq. 6 | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | |  | |
| Scientific field, **SF**: | | Electrical Engineering | |
| Scientific discipline, **SD**: | | Computer Engineering, Engineering of Computer Based Systems | |
| Subject/Key words, **S**/**KW**: | | RTOS, VxWorks, ethernet, communication, BroadR-Reach, diagnostics, RTP | |
| **UC** | |  | |
| Holding data, **HD**: | | The Library of Faculty of Technical Sciences, Novi Sad, Serbia | |
| Note, **N**: | |  | |
| Abstract, **AB**: | | Write code that collects desired diagnostic data about Ethernet communication via BroadR-Reach chip, in sense that data is collected in RTOS kernel, then passed to RT module which process it by predefined logic. Also, make PC application that visualize collected data. | |
| Accepted by the Scientific Board on, **ASB**: | |  | |
| Defended on, **DE**: | |  | |
| Defended Board, **DB**: | President: |  |
|  | Member: |  | Menthor's sign |
|  | Member, Mentor: |  |  |

**Zahvalnost**

Srdačno se zahvaljujem mentoru Doc. Dr Bogdanu Pavkoviću na stručnoj pomoći

tokom izrade rada.

Posebno se zahvaljujem Savi Šunjki na pomoći, utrošenom vremenu, razumevanju i savetovanju tokom izrade rada.

Takođe se zahvaljujem Institutu RT-RK na ukazanoj prilici da se bolje upoznam sa načinom rada u inžinjerskom okruženju i na omogućenom usavršavanju znanja iz date oblasti. Na kraju, hvala roditeljima, Slađanu i Gabrijeli, bratu Andreju, kao i svima onima koji su mi bili stalna podrška u toku školovanja.

**Sadržaj**

[1. Uvod 1](#_Toc519249347)

[2. Teorijske osnove 3](#_Toc519249348)

[2.1 BroadR-Reach 3](#_Toc519249349)

[2.1.1 TJA1100 4](#_Toc519249350)

[2.1.2 MDIO 5](#_Toc519249351)

[2.1.2.1 Odredbe 22 i 45 5](#_Toc519249352)

[2.2 AUTOSAR 6](#_Toc519249353)

[2.2.1 ISO 26262 8](#_Toc519249354)

[2.2.2 ASIL 9](#_Toc519249355)

[2.2.3 MISRA 10](#_Toc519249356)

[2.3 Eternet 10](#_Toc519249357)

[2.3.1 TCP/IP 11](#_Toc519249358)

[2.3.2 IPv6 14](#_Toc519249359)

[2.4 RTOS i Qt 16](#_Toc519249360)

[2.4.1 VxWorks 16](#_Toc519249361)

[2.4.2 Qt 17](#_Toc519249362)

[3. Koncept rešenja 18](#_Toc519249363)

[4. Programsko rešenje 20](#_Toc519249364)

[4.1 Modul za čitanje podataka sa ETH PHY integrisanog kola 20](#_Toc519249365)

[4.1.1 Funkcionalnost datoteke moduleOneStartup.c 20](#_Toc519249366)

[ void module1\_InitPhy(void) 21](#_Toc519249367)

[ LOCAL STATUS \_module1\_CreateMsgQueues (void) 22](#_Toc519249368)

[ void module1\_ConfigureEthInterface(void) 22](#_Toc519249369)

[ void module1\_ConfigureVLAN(void) 22](#_Toc519249370)

[ void module1\_StartTasks(void) 23](#_Toc519249371)

[ STATUS rtpModule(void) 23](#_Toc519249372)

[4.1.2 Funkcionalnost datoteke moduleOneReadEthPhy.c 24](#_Toc519249373)

[4.1.3 Funkcionalnost datoteke moduleOneHandleRoutines.c 29](#_Toc519249374)

[4.2 Modul za logičku obradu prikupljenih podataka 31](#_Toc519249375)

[4.2.1 Funkcionalnost datoteke moduleTwoRTP.c 31](#_Toc519249376)

[4.2.2 Funkcionalnost datoteke moduleTwoServer.c 33](#_Toc519249377)

[4.2.3 Funkcionalnost datoteke moduleTwoCommunication.c 34](#_Toc519249378)

[4.2.4 Funkcionalnost datoteke moduleTwoHandleRoutines.c 35](#_Toc519249379)

[4.2.5 Funkcionalnost datoteke moduleTwoFileSending.c 36](#_Toc519249380)

[4.3 Modul za interaktivnu vizuelizaciju podataka 38](#_Toc519249381)

[4.3.1 Funkcionalnost datoteke main.cpp 38](#_Toc519249382)

[4.3.2 Funkcionalnost datoteke mainwindow.cpp 38](#_Toc519249383)

[4.3.3 Funkcionalnost datoteke module3.cpp 43](#_Toc519249384)

[5. Rezultati 46](#_Toc519249385)

[6. Zaključak 51](#_Toc519249386)

[7. Literatura 52](#_Toc519249387)

**Spisak slika**

Slika 1.1 Eternet u automobilskoj industriji 2

Slika 2.1 BroadR-Reach dupleks komunikacija putem UTP kabla 3

Slika 2.2 TJA1100 integrisano kolo 4

Slika 2.3 Logika odredbe 22 dodata na odredbu 45 5

Slika 2.4 AUTOSAR kao sprega između fizičke arh. i programske podrške 6

Slika 2.5 AUTOSAR slojevi 7

Slika 2.6 Pregled ISO 26262 standarda u odnosu na V-model 8

Slika 2.7 Primer komponenata sa različitim ASIL nivoima 9

Slika 2.8 ASIL klasifikacija 10

Slika 2.9 Eternet frejm 11

Slika 2.10 Uprošćen dijagram stanja TCP protokola 12

Slika 2.11 TCP uspostava veze 13

Slika 2.12 TCP prekid veze 13

Slika 2.13 TCP segment 14

Slika 2.14 Razlike između IPv4 i IPv6 zaglavlja 15

Slika 2.15 VxWorks logo 16

Slika 2.16 Qt logo 17

Slika 4.1 *Altera Cyclone V* integisano kolo 22

Slika 4.2 Spisak registara TJA1100 integrisanog kola 27

Slika 4.3 Izgled *log* datoteke 27

Slika 4.4 Izgled klijentske aplikacije 39

Slika 4.5 Primer *ping* zahteva i odgovora 40

Slika 4.6 Primer dobavljanja *link\_fail* brojača 41

Slika 4.7 Primer uspešnog preuzimanja dijagnostičkih podataka 42

Slika 4.8 Primer preuzimanja *errorLog.txt* datoteke 44

Slika 5.1 Uspešno pokretanje aplikacije (gore) i prihvatanje veze (dole) 46

Slika 5.2 Preuzimanje *errorLog.txt* datoteke 47

Slika 5.3 Preuzimanje *message queue* dijagnostičkih podataka 48

Slika 5.4 Preuzimanje deljene memorije 48

Slika 5.5 Veza uspostavljena 49

Slika 5.6 Veza prekinuta i provera konekcije bezuspešna 49

Slika 5.7 Pokrenuta testna rutina 4 50

Slika 5.8 Izgled deljene memorije nakon fizičog prekida veze dva puta 50

**Spisak tabela**

Tabela 4.1 Funkcije datoteke moduleOneStartup.c 21

Tabela 4.2 Funkcije datoteke moduleOneReadEthPhy.c 24

Tabela 4.3 Polja strukture \_diag\_data\_struct 25

Tabela 4.4 Polja strukture \_diag\_shm\_ptr 26

Tabela 4.5 Funkcije datoteke moduleOneHandleRoutines.c 29

Tabela 4.6 Funkcije datoteke moduleTwoRTP.c 31

Tabela 4.7 Funkcije datoteke moduleTwoServer.c 33

Tabela 4.8 Funkcije datoteke moduleTwoCommunication.c 34

Tabela 4.9 Funkcije datoteke moduleTwoHandleRoutines.c 35

Tabela 4.10 Funkcije datoteke moduleTwoFileSending.c 36

**Skraćenice**

**IVI -** *in-vehicle-infotainment,* sistem za informacije i multimediju

**CAN** - *controller area network*

**LiDAR -** *light detection and ranging*

**PHY** - *physical layer,* fizički nivo OSI modela

**MAC -** *media access control layer,* sloj veze OSI modela

**ADAS** - *advanced driver-assistance systems*

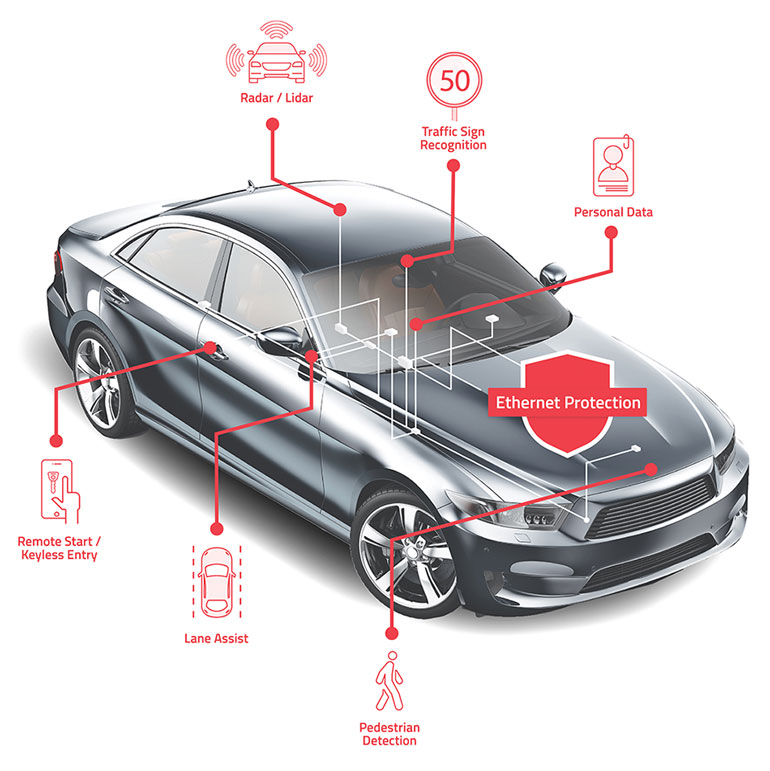
**LVDS -** *low-voltage diffeential signaling*

# Uvod

Automatizacija i povezivanje, odnosno umrežavanje, guraju automobilsku industriju (eng. *Automotive)* napred. Zbog sve veće potrebe za protokom (eng. *Bandwidth)*, fleksibilnosti i smanjenju troškova, umrežavanje, kako komponenata u samom vozilu, tako i između vozila evoluira, a eternet (eng. *Ethernet)* postaje izbor za realizaciju tih potreba.

Iako eternet nije bio u upotrebi u automobilskoj industriji do pre nekoliko godina, koristio se u mnogim drugim oblastima povezivanja i komunikacija. Razvijen 1970ih postao je standard za umrežavanje računara širom sveta.

Eternet u automobilskoj industriji je nešto drugačiji od standardnog i optimizovan je za upotrebu u vozilima. Do sada, uglavnom se koristio za dijagnostiku, IVI sisteme i povezivanje senzora. Ovi sistemi, zahtevaju veći protok podataka da bi dovoljno velikim brzinama mogli preneti podatke i istovremeno održati bezbednost vozača i putnika, brzinama koje komunikacioni protokoli poput CAN-a ili FlexRay-a ne omogućavaju.



Slika 1.1 Eternet u automobilskoj industriji

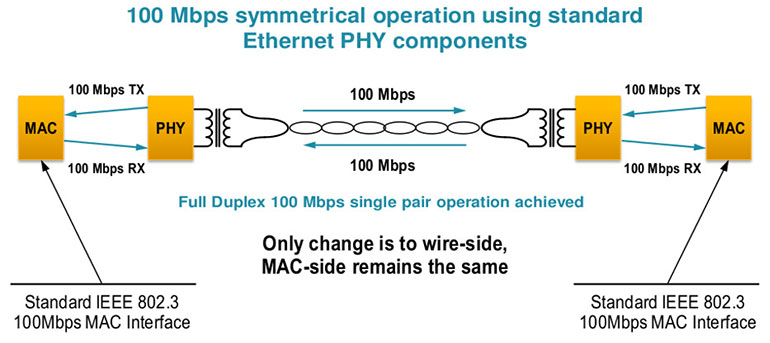
Zbog sve većeg razvoja različitih funkcionalnosti, poput kamera, LiDAR sistema i prepoznavanja saobraćajnih znakova i pešaka, dosadašnjih 100 Mb/s postaje nedovoljno. IEEE802.3 radi na razvoju mnogo bržeg standarda koji će mogućnost protoka od više gigabita po sekundi.

Eternet nudi veliki potencijal za budućnost. Napravljen je za veliki protok, veliki broj ljudi ga koristi i radi na njegovom usavršavanju. Takođe je ekonomično rešenje, korišćenjem jednog TP kabla bez omotača. Broadcom, kompanija odgovorna za uvođenje trenutnog standarda, BroadR-Reach, procenjuje da mogu smanjiti troškove povezivanja za 80% i težinu kablova u vozilima za 30%.

# Teorijske osnove

## BroadR-Reach

Konfiguracija BroadR-Reach veze pokazuje da protokol nudi dupleks komunikacija brzinom od 100Mb/s preko jednog TP kabla. Na obe strane se nalaze BroadR-Reach PHY integrisana kola koji šalju i primaju podatke u oba smera istovremeno. Iznad PHY nivoa nalazi se MAC sprega, čija topologija se u potpunosti oslanja na IEEE 802.3 eternet. Razlika između BroadR-Reach i standardnog eterneta je u međusobnoj komunikaciji PHY nivoa.



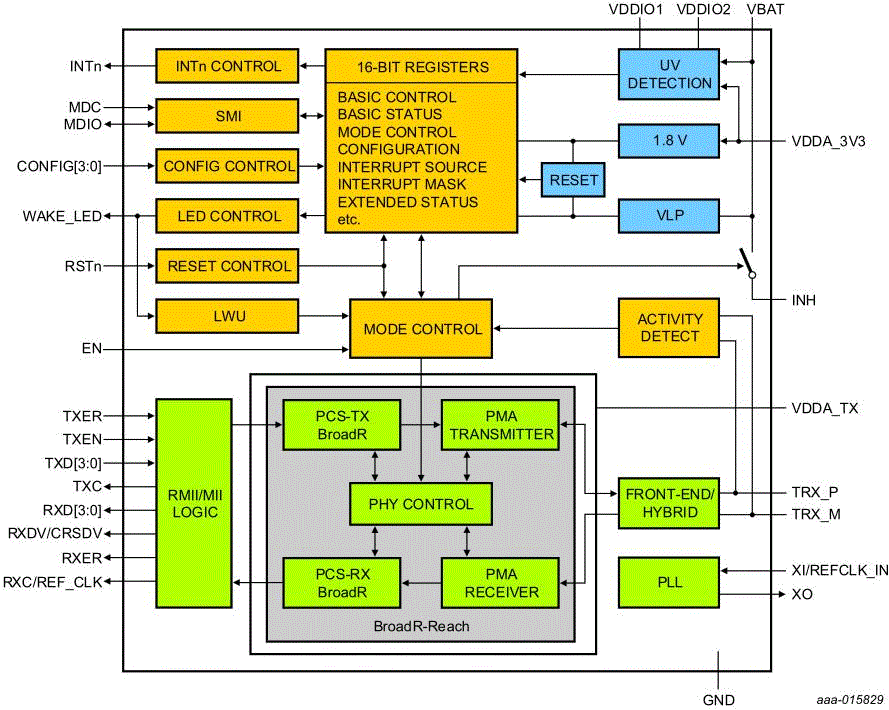
Slika 2.1 BroadR-Reach dupleks komunikacija putem UTP kabla

Jedan od najvažnijih razloga zbog kojih se koristi ovaj protokol jeste ADAS, koji može biti pasivan ili aktivan. Pasivan ADAS sistem pruža vozačima samo upozorenja ukoliko, na primer, izađu van svoje trake. S druge strane, aktivan ADAS sistem ne samo da upozorava vozača, već i preuzima kontrolu nad vozilom u smislu da vrati vozilo u traku ili izvrši automatsko kočenje u slučaju potencijalne nezgode.

BroadR-Reach je najpogodniji za pasivne ADAS primene, jer aktivni ADAS zahteva video bez gubitaka kako bi algoritmi za prepoznavanje oblika mogli normalno funkcionisati. Koristi se i u prenosu informacija i multimedija u automobilu jer je jeftiniji i efikasniji od LVDS standarda.

### TJA1100

TJA1100 je integrisano kolo koje koristi 100BASE-T1 eternet standard optimizovan za upotrebu u automobilskoj industriji. Uređaj omogućava brzinu slanja i prijema od 100Mb/s putem jednog UTP kabla maksimalne dužine od 15m. Kolo je dizajnirano da umanji potrošnju energije, istovremeno pružajući sigurnost koju zahteva standard.



Slika 2.2 TJA1100 integrisano kolo

### MDIO

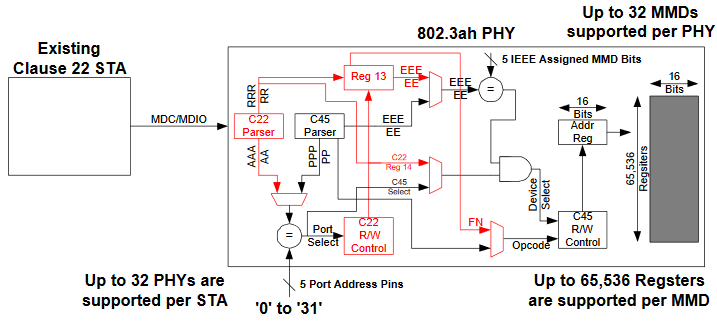
MDIO (eng. *Management Data Input / Output*) je dvožilna serijska sprega koja povezuje upravljački entitet i upravljani PHY sa svrhom kontrolisanja PHY nivoa i dobavljanja statusa istog.

#### Odredbe 22 i 45

Odredba 22 (eng. *Clause 22*) se koristi za upravljanje PHY nivoa koji rade na 10Mbps, 100Mbps i 1000Mbps. Podržava do 32 PHY elementa po jednoj MDIO sprezi. Postoje dva tipa MDIO okvira, okvir za čitanje i okvir za pisanje koji rade na MDC (eng. *Management Data Clock*) brzini od 2.5MHz. Samo dva 16-bitna registra ostaju neiskorišćena. Problem je što ne podržava PHY elemente koji se sastoje od više uređaja.

Odredba 45 (eng. *Clause 45*) se koristi za 10Gbps PHY elemente i ima indirektnu adresnu šemu. Dva okvira su potrebna da bi se pristupilo registru, prvo adresni, a zatim okvir operacije. Podržava 32 MMD (eng. *MDIO Manageable Devices*) po PHY elementu, a 65.536 registara po MMD.

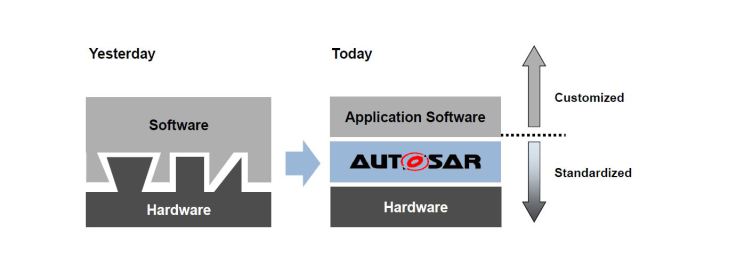
Problem prelaska sa odredbe 22 na odredbu 45 je što većina uređaja radi sa postojećim 10/100Mbps koristeći MII (eng. *Media Independent Interface*) za okvire podataka i MDC/MDIO za pristupanje registrima. Rešenje ovog problema se pronašlo u dva neiskorišćena registra (13 i 14) odredbe 22. Registar 13 se koristi kao komandni regista, a registar 14 kao adresni registar ili registar podataka odredbe 45.



Slika 2.3 Logika odredbe 22 dodata na odredbu 45

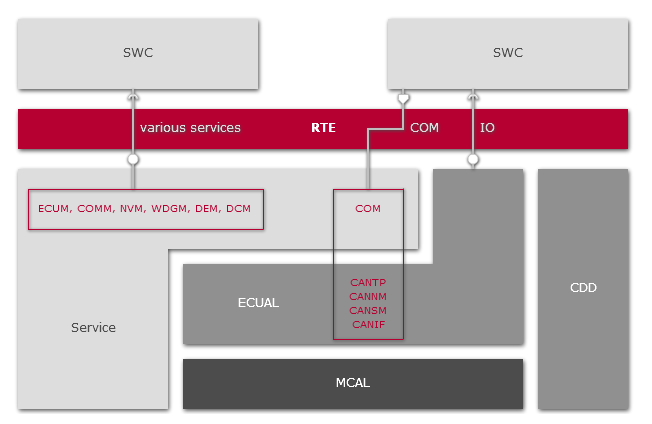
## AUTOSAR

AUTOSAR (eng. *AUTomotive Open System ARchitecture*) je standardizovana programska podrška, razvijena zajednički od strane proizvođača automobila, programskih alata, poluprovodnika i elektronske industrije. Ona predstavlja vezu između fizičke arhitekture i programske podrške putem virtualne magistrale (eng. *Virtual function bus*).



Slika 2.4 AUTOSAR kao sprega između fizičke arh. i programske podrške

Formalno definisane komponente programske podrške (SWCs, eng. *Software component*) sa jasnom spregom ka osnovnoj programskoj podršci (BSW, eng. *Basic software*) formiraju jedan od elemenata AUTOSAR arhitekture. Moduli osnovne programske podrške pružaju usluge kao što je komunikacija putem magistrala, upravljanje memorijom, pristup U/I uređajima i dijagnostičke usluge. Izvršno okruženje (eng. *Runtime environment)* upravlja vezom između SWCs i vezom od SWC ka BSW. Virtualna magistrala (VFB) predstavlja vezu između programskih komponenti i zbog toga su te komponente nezavisne od elektonske upravljačke jedinice (ECU, eng. *Electronic control unit*).



Slika 2.5 AUTOSAR slojevi

Programska podrška za jednu elektronsku upravljačku jedinicu je podeljena na osnovnu programsku podršku, izvršno okruženje i aplikativni sloj. Osnovna programska podrška se sastoji od više potprograma grupisanih u slojeve. Razlikujemo sloj za apstrakciju mikrokontrolera (MCAL, eng. *The Microcontroller abstraction layer),* sloj za apstrakciju elektronske upravljačke jedinice (ECUAL, eng. *ECU Abstraction Layer)* i sloj sistemskih usluga (eng. *Service Layer).*

RTE vrši apstrakciju aplikativnog sloja od osnovne programske podrške. Kontroliše ponašanje izvršavanja aplikativnog sloja i realizuje razmenu podataka. U aplikativnom sloju, sama namena ECU je realizovana u obliku zasebnih programskih komponenti.

Ovakav model pojednostavljuje prenosivost programske podrške na različite fizičke arhitekture. Sve što treba uraditi je zameniti upravljačke programe u MCAL sloju i podesiti ECUAL sloj, čime se značajno štedi na vremenu prilikom izrade i testiranja samih programa.

### ISO 26262

ISO 26262 je međunarodni standard za bezbednost električnih i elektronskih uređaja u proizvodnji automobila, težine do 3500kg, definisan od strane Međunarodne organizacije za standardizaciju (ISO, eng. *International Organization for Standardization*) 2011. godine.

Predstavlja prilagođenje, odnosno produžetak, IEC 61508 bezbednosnog standarda koji definiše celovitost bezbednosnih nivoa (SILs, eng. *Safety Integrity Levels)* i primenjuje ih u automobilskoj industriji (ASILs, eng. *Automotive SILs*). Bezbednost sistema se postiže brojnim sigurnosnim merama, koje su realizovane na gotovo svim sistemima u vozilu (hidraulika, mehanika, pneumatika, elektronika, itd.). Ne odnosi se na potencijale opasnosti vezane za električni udar, požar, dim, toplota, radijaciju, zapaljivost, koroziju, oslobađanje energije i slično koje nisu direktno vezane za otkazivanje elektronskih uređaja u vozilu.

Ciljevi ISO 26262 standarda su:

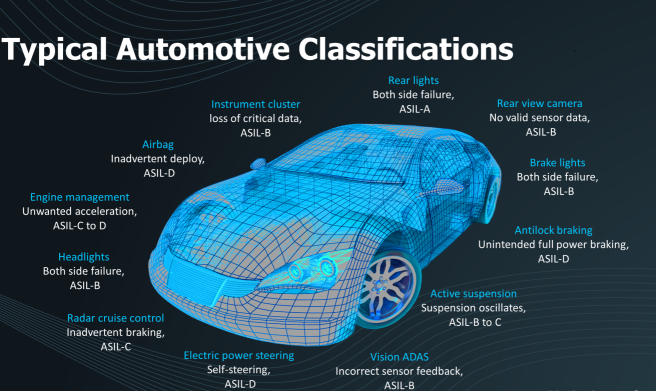
* Sigurnost u životnom ciklusu razvoja vozila
* Kontrola rizika
* Izbegavanje rizika korišćenjem ASILa
* Zahtevanje provere ispravnosti da bi se ustanovilo da li je određen nivo bezbednosti zadovoljen
* Pruža zahteve za odnose sa dobavljačima



Slika 2.6 Pregled ISO 26262 standarda u odnosu na V-model

### ASIL

ASIL, definisan ISO 26262 standardom je šema za kategorizaciju rizika pomoću koje se određuju sigurnosni zahtevi. Razvoj svake komponente u automobilu započinje svoj životni ciklus definisanjem ASIL nivoa koji ona treba da zadovolji. Svaka komponenta ima sigurnosni cilj koji treba da ispuni da bi funkcionisala normalno, bez da predstavlja opasnost po vozilo ili putnike u njemu, i taj cilj se određuje dodeljivanjem odgovarajućeg QM (eng. *Quality Management*) ili ASIL nivoa po standardu.



Slika 2.7 Primer komponenata sa različitim ASIL nivoima

Razlikujemo ASIL A,B,C i D, gde ASIL D predstavlja najveći stepen opasnosti. Postoje tri sigurnosna nivoa bazirana na sledećim parametrima:

* Izloženost (E) – mera mogućnosti da se vozilo nađe u opasnoj situaciji koja može prouzrokovati štetu ljudima i imovini. Postoje 4 nivoa, E1, E2, E3 i E4, sortirani rastuće po stepenu verovatnoće da će se dogoditi.
* Mogućnost upravljanja (C) – određuje do kojeg nivoa vozač može kontrolisati vozilo ukoliko dođe do kvara na nekoj od komponenata. Postoje 3 nivoa, C1, C2 i C3, sortirani rastuće po težini upravljanja u slučaju kvara.
* Opasnost (S) – definiše ozbiljnost štete ili posledica na ljudski život i imovinu. S1 predstavlja lake i srednje teške povrede, S2 ozbiljne povrede koje ugrožavaju život i S3 teški incidenti i sudari u kojima može doći do gubitka života.



Slika 2.8 ASIL klasifikacija

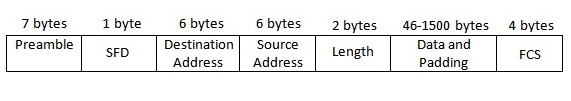
### MISRA

MISRA C (*Motor Industry Software Reliability Association*) je skup smernica za razvoj programske podrške u C programskom jeziku. Cilj joj je da poboljša sigurnost, prenosivost i pouzdanost koda u ugrađenim sistemima posebno u onima koji su napisani po ISO C / C90 / C99 standardima.

## Eternet

Eternet predstavlja najčešće korišćenu tehnologiju u žičnim lokalnim mrežama. Opisana je 802.2 i 802.3 standardima, koji definišu protokole na nivou mreže i na fizičkom nivou. Struktura eternet frejma se sastoji iz sledećih polja:

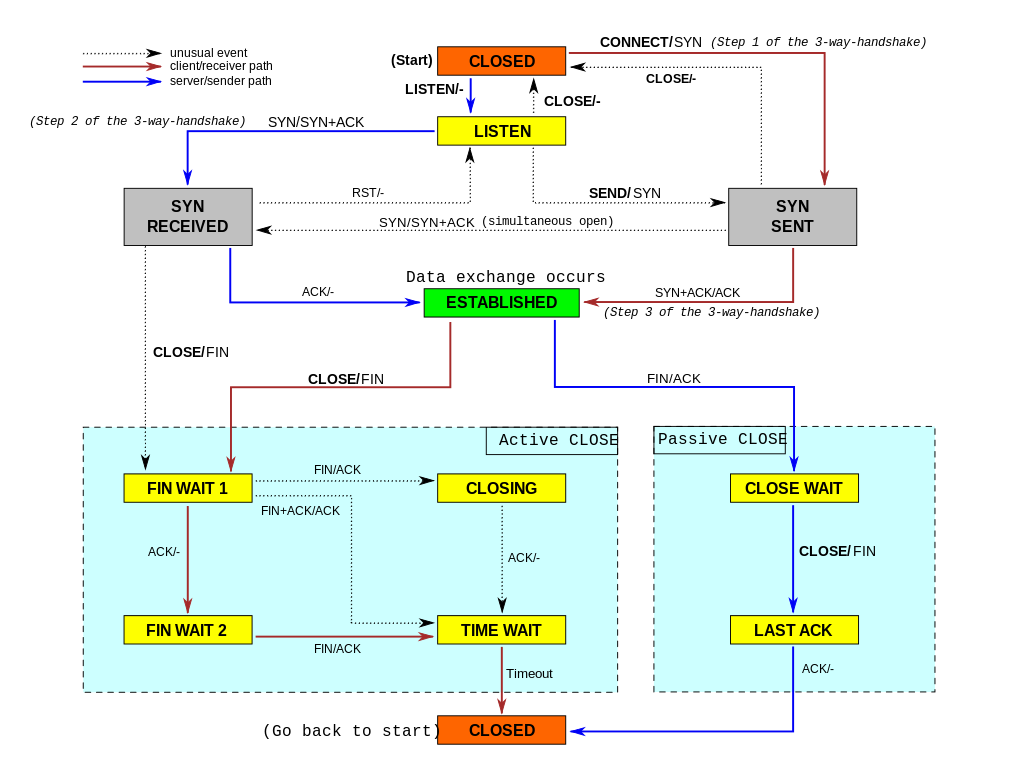
* Preambula (8 bajta) – ima ulogu u sinhronizaciji između pošiljaoca i primaoca. Sastoji se od sekvence nula i jedinica, koje se naizmenično šalju.
* Destinaciona adresa (6 bajta) – fizička (MAC) adresa primaoca.
* Izvorišna adresa (6 bajta) – fizička (MAC) adresa pošiljaoca.
* Tip (2 bajta) – Tip enkapsuliranih podataka (IP, ARP, VLAN...).
* Podaci (46-1500 bajta) – Enkapsulirani podaci viših nivoa, najčešće IP protokola. Frejmovi koji nisu propisane veličine se smatraju nevalidnim i odbacuju se. Ukoliko je enkapsulirana veličina paketa manja od 46 bajta, ona se dopunjuje dodatnim bitima. U slučaju da su enkapsulirani podaci veći od 1500 bajtva, datagram se deli u više frejmova.
* Sekvenca provere frejma (eng. *Frame Check Sequence*) – veličine 4 bajta, omogućava prepoznavanje oštećenih frejmova pri fizičkom prenosu. Na osnovu njene vrednosti prijemnik određuje da li se desila greška pri prenosu.



Slika 2.9 Eternet frejm

### TCP/IP

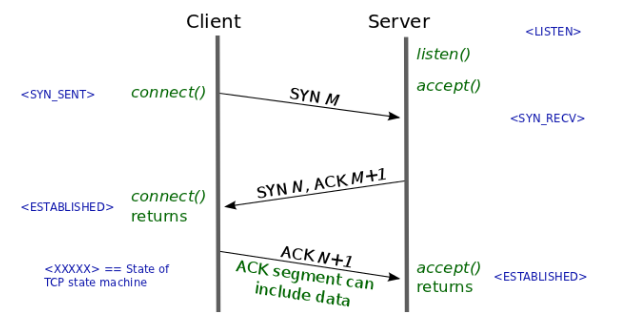
TCP/IP (eng. *Transmission Control Protocol / Internet Protocol*) koristi klijent/server model komunikacije gde klijent biva opslužen od strane servera. Komunikacija je dvosmerna i realizovana u vidu konekcije koja se uspostavlja pomoću metode rukovanja (eng. *Handshaking*). TCP omogućava praćenje poslatih paketa, kontrolu toka komunikacije, kao i redosleda pristiglih segmenata.



Slika 2.10 Uprošćen dijagram stanja TCP protokola

Rad TCP protokola se može razdvojiti u tri dela. Konekcija se uspostavlja tako što se između predajne i prijemne strane iz tri puta razmene poruke sa podešenim odgovarajućim kontrolnim bitima (TCP *three-way handshake*):

* Predajna strana šalje poruku sa podešenim kontrolnim bitom SYN = 1, ostali su podešeni na 0, pri čemu nasumično odabere redni broj segmenta (SEQa).
* Prijemna strana odgovara porukom sa kontrolnim bitima SYN i ACK = 1, takođe nasumično bira broj segmenata (SEQb), a za ACK uzima broj ACK = (SEQa) + 1. Na ovaj način je uspostavljena veza od predajne ka prijemnoj strani.
* Slanjem poruke sa podešenim kontrolnim bitom SYN = 1 od prijemne strane, ona zahteva da predajna strana potvrdi uspostavljanje veze od prijemne ka predajnoj strani. Predajna strana to čini slanjem poruke sa podešenim kontrolnim bitom ACK = 1 i uzima vrednost ACK broja ACK = (SEQb) + 1. Na ovaj način uspostavljena je veza između klijenta i servera.

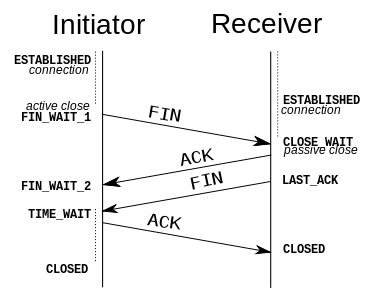


Slika 2.11 TCP uspostava veze

Sledeća faza je faza prenosa podataka. TCP definiše uslugu pouzdane isporuke toka (eng. *Stream*) korisničkih podataka gde je osnovna jedinica prenosa segment podataka. Format segmenta je izabran tako da je moguće potvrđivanje podataka iz jednog smera, njihovim uključivanjem u zaglavlje segmenata koji se šalju u drugom smeru. Kontrola toka je realizovana tako što prijemnik oglašava količinu podataka koju je spreman da primi.

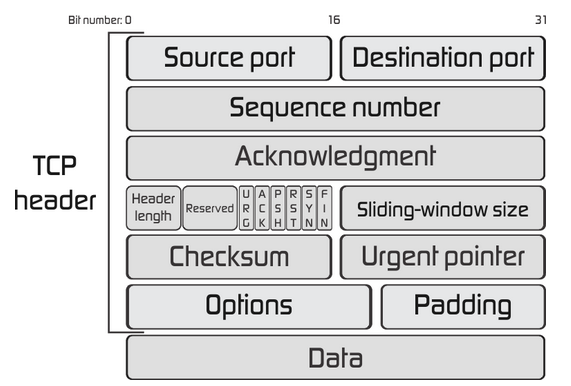
Poslednja faza je faza prekida veze koja ima 4 faze:

* Klijent inicira prekid klijent-server veze slanjem bita FIN = 1
* Server potvrđuje zahtev za prekid slanjem ACK = 1
* Server, potom, šalje zahtev za prekid server-klijent veze slanjem FIN = 1
* Na kraju, klijent odgovara na zahtev za prekid slanjem ACK = 1



Slika 2.12 TCP prekid veze

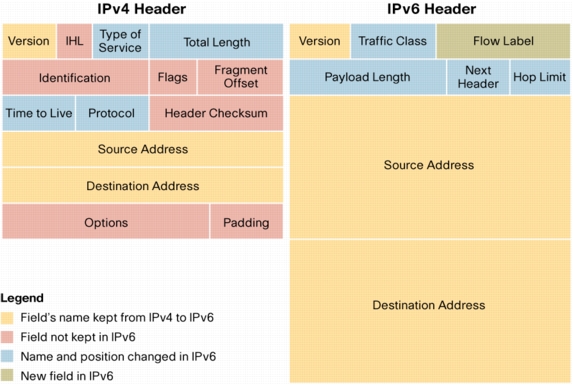
TCP segment se sastoji iz zaglavlja i dela u kome se nalaze podaci. U zaglavlje se smeštaju izvorišni port, odredišni port, broj prvog bajta segmenta u okviru toka podataka, broj sekvence narednog očekivanog segmenta, dužinu zaglavlja, rezervisane bite, kontrolne bite, veličinu klizajućeg prozora, kontrolnu sumu i neke dodatne opcije. TCP segment se enkapsulira u IP datagram, koji se dalje enkapsulira u okvir nivoa veze.



Slika 2.13 TCP segment

### IPv6

IPv6 je najnovija verzija IP protokola, koji vrši identifikaciju i lokaciju sistema za računare na mreži i preusmerava saobraćaj na internetu. Razvijen je da reši dugoročni problem manjka adresa kod IPv4 protokola, jer koristi 128-bitne adrese. IPv6 adrese se predstavljaju u osam grupa po četiri heksadecimalne cifre razdvojene dvotačkom (npr. 2001:0db8:0000:0042:0000:8a2e:0370:7334).



Slika 2.14 Razlike između IPv4 i IPv6 zaglavlja

IPv6 zaglavlje se sastoji od sledećih polja:

* Verzija (4 bita) – konstanta sekvenca bita 0110.
* Klasa saobraćaja (eng. *Traffic Class*) – veličine 6+2 bita, gde se u prvih 6 bita nalazi DS (eng. *Differentiated Services*)polje koje se koristi za klasifikaciju paketa. Druga dva bita predstavljaju ECN (eng. *Explicit Congestion Notification*).
* Oznaka toka (20 bita)
* Dužina podataka (16 bita) – veličina podataka u oktetima, uključujući sva dodatna zaglavlja.
* Sledeće zaglavlje (8 bita) – tip sledećeg zaglavlja. Obično određuje koji se transportni protocol koristi.
* „Hop” ograničenje (8 bita) – vrednost ovog polja se umanjuje za jedan na svakom međučvoru koji paket poseti. Kada brojač dođe do nula paket se odbacuje.
* Izvorišna adresa (128 bita)
* Odredišna adresa (128 bita)

## RTOS i Qt

RTOS (*Real-time operating system*) je operativni sistem čija je svrha da opsluži aplikacije u realnom vremenu i obradi podatke bez odlaganja. RTOS ima napredan algoritam za raspored izvršavanja (eng. *Scheduler*) zadataka. U tipičnom slučaju, zadatak (eng. *Task*) ima tri stanja, aktivan, spreman i blokiran. U najvećem broju slučajeva zadaci su ili blokirani, ili spremni, jer samo jedan zadatak može da se izvršava na procesoru u jednom trenutku.

### VxWorks

VxWorks je RTOS razvijen kao vlasnička programska podrška (eng. *Proprietary Sowtware*) od strane Wind River-a. Namena mu je da se koristi u ugrađenim sistemima koji zahtevaju rad u realnom vremenu, determinističko ponašanje i u velikom broju slučajeva sigurnosne sertifikate, za industrije poput automobilske, avionske, vojne, medicinske, itd. Podržava Intel, POWER i ARM arhitekture, simetrično i asimetrično multiprocesiranje na 32-bitnim i 64-bitnim procesorima. Prevodioci koji postoje unutar VxWorks jezgra su Diab, GNU i Intelov C++ kompajler (ICC).



Slika 2.15 VxWorks logo

Razvoj programa obavlja se na računaru „domaćinu“ unakrsnim prevođenjem (eng. *Cross Compiling*), korišćenjem integrisanog radnog okruženja (IDE) koje dolazi sa programom za obradu teksta, prevodiocem, programom za otklanjanje grešaka (eng. *Debugger*), i emulatorom. Sam IDE je baziran na Eklipsu (eng. *Eclipse*).

### Qt



Slika 2.16 Qt logo

Qt je razvojno okruženje za kreiranje grafičkih aplikacija (GUI, eng. *Graphical User Interface*) koja funckionišu na različitim fizičkim arhitekturama i programima sa jako malo izmena. Podržava različite prevodioce uključujući GCC C++ prevodilac i kompatibilan je sa Visual Studio okruženjem. Ključni koncept Qt-a jeste koncept signala i slotova, koji omogućava komunikaciju između objekata, kao i slanje objekata korišćenjem ovog mehanizma.

# Koncept rešenja

Koncept rešenja se svodi na opisivanje tri jedinstvene celine. Prva celina, odnosno modul se izvršava direktno u *kernel* prostoru namenske platforme. Drugi modul se izvršava u korisničkom prostoru namenske platforme, dok se treća celina izvršava na računaru. Sve tri strane su realizovane u programskom jeziku C, s tim što je treći modul potpomognut Qt razvojnim okruženjem zbog realizacije korisničke aplikacije. Sem Qt, radna okruženja koja su korišćena su *Visual Studio 2017 Professional* i *Wind River Workbench.* Operativni sistemi koji na kojima su realizovani pomenuti moduli su *Windows 10,* na računaru, i *RTOS VxWorks Wind River* na namenskoj platformi.

Ideja rešenja jeste da modul jedan, modul za čitanje registara ETH PHY integrisanog kola, vrši mapiranje *FPGA Cyclone V* integrisanog kola i čitanje registara istog, čime se dolazi do informacija o radu dvožilnog komunikacionog *BroadR-Reach* eternet interfejsa. Čitanje registara se vrši u posebnom zadatku na svakih 250 milisekundi. Pročitane vrednosti se pohranjuju u prethodno inicijalizovane deljenu memoriju (eng. *Shared Memory*) i redove poruka (eng. *Message Queues*). Putem ovih memorija modul jedan šalje podatke modulu dva na dalju obradu i kreira spregu za primanje naredbi koje zadaje korisnik. Naredbe koje se realizuju su preuzimanje podataka iz jedne ili druge memorije i testne rutine. Neke od testnih rutina služe za proveru da li je komunikacija između namenske platforme i računara moguća ili koliko puta je došlo do prekida komunikacije (fizički prekinuta konekcija, itd). Takodje, modul jedan izvršava podešavanje *hmi0* eternet interfejsa za komunikaciju sa računarom i vrši upisivanje potencijalih grešaka u *log* datoteku koja se briše svaki put kada je preuzeta.

Zadatak modula dva, modula za logičku obradu podataka, koji predstavlja SWC (eng. *Software Component*), jeste da pročita i obradi podatke pristigle iz modula jedan i kreira sprege za komunikaciju sa modulom jedan i modulom tri, odnosno korisničkom aplikacijom. Funkcionalnost ovog modula predstavlja automat sa konačnim brojem stanja gde automat čeka na naredbu koju zadaje korisnik. Unutrašnja komunikacija u modulu, odnosno automatu, obavlja se putem *message queue*-a upisivanjem naredbe u isti. Naredba se potom izvršava i automat se vraća u početno stanje gde čeka na novu naredbu. Modul dva komunikaciju sa modulom jedan obavlja otvaranjem deljene memorije i *message queue*-a i predstavlja serversku stranu u komunikaciji sa računarom putem TCP/IP protokola. Omogućava slanje datoteka ili dijagnostičkih podataka putem *socket*-a na zahtev iz korisničke aplikacije.

Treći modul, modul na računarskoj strani, jeste klijentska aplikacija koja pruža korisniku, na zahtev, dijagnostičke podatke o radu *BroadR-Reach* integrisanog kola. Preuzimanje *log* datoteke koja sadrži podatke o vrsti greške i sistemsko vreme kada se greška dogodila vrši se periodično u zasebnom *thread-*u. Aplikacija je realizovana u QT programskom okruženju pomoću signala i slotova. Funkcionalnost aplikacije se pokreće pritiskom na dugme „*Start Client“*. Korisnik pritiskom na odgovarajuće dugme može da zatraži dijagnostičke podatke koji se prikazuju u interfejsu aplikacije ili unosom broja rutine da zatraži izvršavanje iste. Pritiskom na dugme „*Exit“* aplikacija prestaje sa radom.

# Programsko rešenje

U ovom poglavlju biće opisana sama funkcionalnost svakog od tri modula.

## Modul za čitanje podataka sa ETH PHY integrisanog kola

Realizacija modula 1 je smeštena u tri datoteke:

1. **moduleOneStartup.c**
2. **moduleOneReadEthPhy.c**
3. **moduleOneHandleRoutines.c**

### Funkcionalnost datoteke moduleOneStartup.c

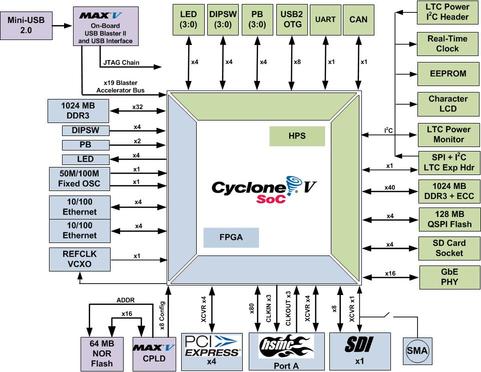
Datoteka moduleOneStartup.c je početna datoteka u kojoj se nalaze potrebne funkcije za inicijalizaciju integrisanog kola, memorije, zadataka kao i Eternet i VLAN interfejsa.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prototip funkcije | Vidljivost | Povratna vrednost | Poziva | Poziva je |
| void module1\_InitPhy() | globalna | / | \_module1\_CreateMsgQueues() | tttechBRInit()  (*tttechBroadRReach.c*) |
| STATUS rtpModule() | globalna | OK /  ERROR | / | \_BRDiagTask\_entry()  (*tttechBroadRReach.c*) |
| void module1\_StartTasks() | globalna | / | module1\_ReadChipRegistersTask()  (*moduleOneReadEthPhy.c*)  module1\_GetRoutineNum()  (*moduleOneHandleRoutines.c*) | \_ttechStartBRDiagTask()  (*tttechBroadRReach.c*) |
| void module1\_ConfigureEthInterface() | globalna | / | / | \_attach\_if\_to\_device()  (*tttechBroadRReach.c*) |
| void module1\_ConfigureVLAN() | globalna | / | / | \_attach\_if\_to\_device()  (*tttechBroadRReach.c*) |
| LOCAL STATUS \_module1\_CreateMsgQueues() | lokalna | OK /  ERROR | / | module1\_InitPhy() |

Tabela 4.1 Funkcije datoteke moduleOneStartup.c

### void module1\_InitPhy(void)

Izvršavanje modula počinje pozivanjem ove funkcije iz datoteke tttechBroadRReach.c prilikom pokretanja sprege za očitavanje dijagnostike. Ova funkcija vrši mapiranje adrese integrisanog kola *Cyclone V* u korisnički prostor i mapirana adresa se smešta u promenljivu *ethIf\_macVirtAddr*. U globalne promenljive *phyGmiiAddress* i *phyGmiiData* se smeštaju mapirane adrese registara GMII\_Address i GMII\_Data dodavanjem odgovarajućeg *offset*-a na promenljivu *ethIf\_macVirtAddr*. Funkcija otvara i *log* datoteku za upis potencijalnih grešaka i poziva funkciju za kreiranje redova poruka (eng. *Message Queues*).



Slika 4.1 *Altera Cyclone V* integisano kolo

### LOCAL STATUS \_module1\_CreateMsgQueues (void)

Ova funkcija vrši otvaranje javnih redova poruka za komunikaciju sa modulom dva. *DiagMsgQ* se koristi za slanje structure koja sadrži dijagnostičke podatke, dok *routinesMsgQ* služi za primanje zahteva za pokretanje testnih slučajeva. Povratna vrednost funkcije je OK ili ERROR u slučaju da je došlo do greške prilikom otvaranja nekog od ova dva reda poruka.

### void module1\_ConfigureEthInterface(void)

Funkcija obavlja konfigurisanje i podizanje eternet interfejsa *hmi0* sa IPv6 adresom namenske platforme upotrebom funkcije *ifconfig().*

### void module1\_ConfigureVLAN(void)

Funkcija obavlja konfigurisanje i podizanje VLAN interfejsa *vlan10* sa IPv4 i IPv6 adresom namenske platforme upotrebom funkcije *ifconfig().*

### void module1\_StartTasks(void)

Ova funkcija kreira zadatak (eng. *Task*) *“readDiag”,* čija funkcionalnost se izvršava na svakih 250ms i zadatak *“getRoutine”* sa prioritetima 202 i 204. Ovako nizak prioritet u sistemu znači da će se funckionalnost ovih zadataka izvršavati tek onda kada processor obavi ostale aktivnosti važnije za rad sistema. Zadatak *“readDiag”* poziva funkciju *module1\_ReadChipRegistersTask(),* a *“getRoutine” module1\_GetRoutineNumber().* U slučaju da je došlo do greške prilikom pokretanja nekog od zadataka, red poruka koji je vezan za taj zadatak se zatvara.

### STATUS rtpModule(void)

Funkcija za pokretanje RT procesa, odnosno softverske komponente, koja predstavlja modul za logičku obradu prikupljenih podataka – modul dva. Povratna vrednost funkcije je OK ili ERROR ukoliko je došlo do greške prilikom pokretanja RT procesa.

### Funkcionalnost datoteke moduleOneReadEthPhy.c

Ova datoteka sadrži funkcije za čitanje registara TJA1100 integrisanog kola. Pročitana stanja smešta u globalne strukture *\_diag\_data\_struct* i *\_diag\_shm\_ptr* koje šalje modulu dva kroz deljenu red poruka, odnosno deljenu memoriju.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prototip funckije | Vidljivost | Povratna vrednost | Poziva | Poziva je |
| uint32\_t mdio\_read\_br(uint32\_t regNumber) | globalna | Vrednost pročitanog registra | / | Više funkcija |
| void mdio\_write\_br(uint32\_t regNumber, uint16\_t dataWrite) | globalna | / | / | Više funkcija |
| void module1\_ReadChipRegistersTask(void) | globalna | / | **\_**sharedMemAlloc() \_module1\_ReadChipRegisters() | module1\_StartTasks()  (*moduleOneStartup.c*) |
| LOCAL void \_module1\_ReadChipRegisters(void); | lokalna | / | \_module1\_getErrorTime() | module1\_ReadChipRegistersTask() |
| LOCAL STATUS \_sharedMemAlloc(void) | lokalna | OK /  ERROR | \_module1\_shMem\_Alloc() | module1\_ReadChipRegistersTask() |
| LOCAL void \* \_module1\_shMem\_Alloc(const char \* fname, size\_t size) | lokalna | NULL /  adresa alocirane memorije | / | \_sharedMemAlloc() |
| LOCAL void \_module1\_getErrorTime(struct tm const \* time\_info, uint8\_t pos) | lokalna | / | / | \_module1\_ReadChipRegisters() |

Tabela 4.2 Funkcije datoteke moduleOneReadEthPhy.c

* **void module1\_ReadChipRegistersTask(void)**

Funkcija koja se izvršava kao zaseban zadatak na svakih 250ms. Pre nego što uđe u beskonačnu petlju poziva funkciju \_*sharedMemAlloc()* čime zauzima deljenu memoriju za upisivanje dijagnostičkih podataka pre nego što se isti pročitaju. U beskonačnoj petlji

poziva funckiju *\_module1\_ReadChipRegisters()* koja obavlja čitanje registara i smešta vrednosti u globalnu strukturu *\_diag\_data\_struct*.

|  |  |
| --- | --- |
| Polje strukture | Opis |
| link\_status | Da li je veza uključena ili ne. |
| link\_control | Omogućavanje kontrole veze (eng. *Link).* |
| power\_mode | Biranje režima rada. |
| loopback\_mode | Biranje režima rada loopback-a. |
| phy\_init\_fail | Da li je došlo do greške prilikom inicijalizacije PHY sprege. |
| wakeup | Da li je došlo do zahteva za prelazak u normalni režim rada. |
| link\_status\_fail | Pad veze usled smetnji. |
| link\_status\_up | Vraćanje veze nakon pada. |
| link\_up | Da li je došlo do pada ili ne. |
| tx\_mode | Režim rada predajnika. |
| loc\_rcvr\_status | Lokalni prijemnik OK ili ne. |
| rem\_rcvr\_status | Udaljeni prijemnik OK ili ne. |
| signal\_quality | Kvalitet signala po SQI klasama. |
| errors\_array | Struktura koja sadrži informacije o potencijalnim greškama. Polja ove strukture su broj greške, vreme kada se dogodila i opis greške. |
| ping\_result | Rezultat provere komunikacije između namenske platforme i računara. |
| routine\_status | Status rutine u trenutku čitanja. |
| routine\_result | Rezultat prethodno pokrenute rutine. |

Tabela 4.3 Polja strukture \_diag\_data\_struct

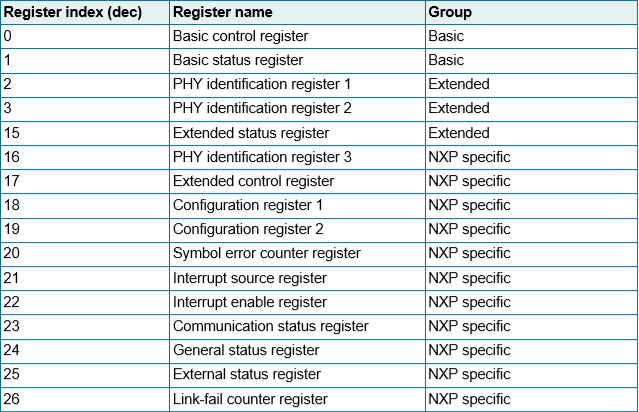
Pozivom ugrađene funkcije *msgQSend()* ova struktura se prosleđuje u deljenu memoriju ka modulu dva. Takođe poziva funkciju *\_module1\_FillSharedMem()* koja manje bitne podatke o radu BroadR-Reach sprege smešta u globalnu strukturu *\_diag\_shm\_ptr*.

|  |  |
| --- | --- |
| Polje strukture | Opis |
| phy\_id\_reg1 | Biti [3:18] OUI. |
| phy\_id\_reg2 | Biti [19:24] OUI. |
| type\_no | Šestobitni fabrički broj. |
| revision\_no | Četvorobitni broj revizije proizvođača. |
| jabber\_detect | Da li je došlo do zagušenja. |
| int\_status | Status prekida. |
| loc\_rcvr\_cnt | Uvećava se kada je lokalni prijemnik u stanju NOT\_OK. Vrednost se resetuje kada se registar pročita. |
| rem\_rcvr\_cnt | Uvećava se kada je udaljeni prijemnik u stanju NOT\_OK. Vrednost se resetuje kada se registar pročita. |
| link\_fail\_cnt | Uvećava se kada konekcija nije moguća. Vrednost se resetuje kada se registar pročita. |

Tabela 4.4 Polja strukture \_diag\_shm\_ptr

* **LOCAL void \_module1\_ReadChipRegisters(void)**

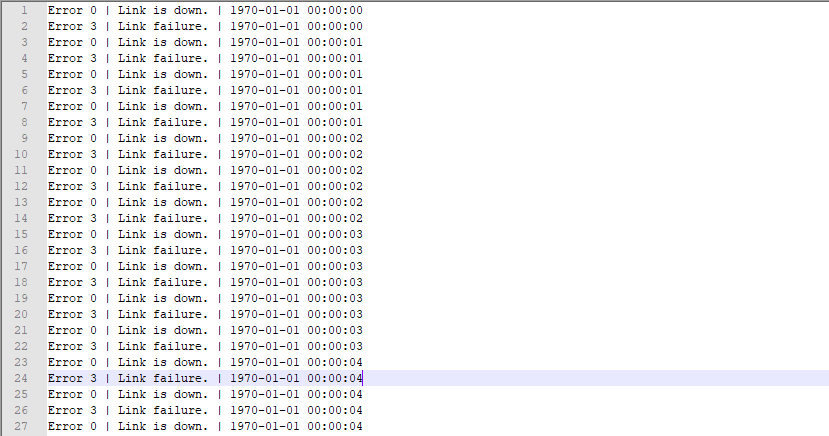
Ova funkcija redom čita stanja bita opisanih polja strukture *\_diag\_data\_struct* pomoću funkcije *mdio\_read\_br()* i u slučajevima ako je došlo do pada linka ili PHY nije dobro podešen upisuje grešku u polje strukture *errors\_array*. Nakon svakog prolaska kroz ovu funkciju brojač *\_delete\_cnt* se uvećava. Kada dostigne vrednost 255 poziva se ugrađena funkcija *remove()* čime se briše *log* datoteka. Brisanje datoteke periodično se obavlja da ne bi došlo do zauzimanja memorije na namenskoj platformi. Potencijalni problem u ovom načinu realizacije čuvanja memorije jeste gubljenje podataka.



Slika 4.2 Spisak registara TJA1100 integrisanog kola

* **LOCAL void \_module\_getErrorTime (struct tm const \* time\_info, uint8\_t pos)**

Funkcija koja dobavlja vreme greške, otvara *log* datoteku *errorLog.txt* i upisuje greške u nju.



Slika 4.3 Izgled *log* datoteke

* **LOCAL STATUS \_sharedMemAlloc(void)**

Alocira memoriju pozivom funkcije *\_module1\_shMem\_Alloc()* čime uzima pokazivač na tu memoriju i dobija mogućnost pristupa, odnosno upisa u strukturu *\_diag\_shm\_ptr*. U slučaju da funkcija *\_module1\_shMem\_Alloc()* vrati NULL pokazivač povratna vrednost je ERROR, inače OK.

* **LOCAL void \* \_module1\_shMem\_Alloc (const char \* fname, size\_t size)**

Parametri ove funkcije su ime deljene memorije i njena veličina. Funkcionalnost se ogleda u otvaranju deljene memorije, odnosno pravljenju objekta, postavljanju veličine objekta i mapiranja memorije u adresni prostor procesa. Povratna vrednost je pokazivač na mapiranu adresu ili NULL pokazivač u slučaju greške.

* **LOCAL void \_module\_FillSharedMem(void)**

Slično kao i funkcija *\_module1\_readChipRegisters(),* obavlja čitanje stanja bita opisanih polja strukture *\_diag\_shm\_ptr* pomoću funkcije *mdio\_read\_br().*

* **uint32\_t mdio\_read\_br (uint32\_t regNumber)**

Parametar ove funkcije je broj registra koji želimo pročitati. Gornja četiri bajta mapirane adrese *phyGmiiAddress* se nuliraju. Pomera broj registra za 6 bita levo i dodaje broj jedan jer pretposlednji bit predstavlja bit dozvole čitanja ili pisanja a poslednji je bit zauzetosti. Bit zauzetosti se podiže na 1 sve dok traje čitanje, a automatski se vraća na 0 kada se čitanje registra završi ili kada istekne brojač. Povratna vrednost funkcije je 32-bitna vrednost sa *phyGmiiData* adrese.

* **void mdio\_write\_br (uint32\_t regNumber, uint16\_t dataWrite)**

Slično kao kod prethodne funckije parametri su broj registra u koji želimo upisati i

16-bitna vrednost koju želimo upisati. Postupak je sličan, s razlikom da se bit čitanja/pisanja postavlja na 1 zbog dozvole upisa. Bit zauzetosti se podiže na 1 sve dok traje pisanje, a automatski se vraća na 0 kada se upisivanje u registar završi ili kad istekne brojač.

### Funkcionalnost datoteke moduleOneHandleRoutines.c

Ova datoteka sadrži funkcije za upravljanje rutinama i testnim slučajevima.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prototip funkcije | Vidljivost | Povratna vrednost | Poziva | Poziva je |
| void module1\_GetRoutineNum(void) | globalna | / | **\_**testModes() | module1\_StartTasks()  (*moduleOneStartup.c*) |
| LOCAL void \_testModes(int routine\_trigger) | lokalna | / | \_normalOperationTest() \_pingRoutine() | module1\_GetRoutineNum() |
| LOCAL void \_normalOperationTest(uint16\_t mode); | lokalna | / | / | **\_**testModes() |
| LOCAL STATUS \_pingRoutine(void); | lokalna | OK /  ERROR | / | **\_**testModes() |

Tabela 4.5 Funkcije datoteke moduleOneHandleRoutines.c

* **void module1\_GetRoutineNum(void)**

Ova funkcija prestavlja još jedan zaseban zadatak koji se izvršava u beskonačnoj petlji. Pozivom ugrađene funkcije *msgQReceive()* sa poslednjim parametrom WAIT\_FOREVER funkcija čeka sve dok korisnik se u red poruke ne upiše broj rutine koju treba pokrenuti. Kada pristigne broj rutine poziva se funckija *\_testModes()* sa parametrom *routineNum* i pokreće se odgovarajuča rutina.

* **LOCAL void \_testModes (int routine\_trigger)**

Parametar ove funkcije je broj rutine koju je korisnik zatražio. Prvo se omogući podešavanje, odosno menjanje stanja bita registara, tako što se u polje CONFIG\_EN proširenog upravljačkog registra upiše jedinica. Postoji osam testnih rutina. Ukoliko korisnik unese broj 0 pokreće se testiranje normalnog režima rada, dok ukoliko unese broj 6 pokreće se rutina za proveru da li je komunikacija sa računarom moguća preko *hmi0* BroadR-Reach IPv6 sprege. Unošenjem broja 7 čita se polje *link\_fail\_cnt* opšteg statusnog registra, kaoi polja registra koji čita koliko puta je lokalni ili udaljeni prijemnik uzrokovao grešku u komunikaciji Brojevima 1-5 se pokreće neka od testnih rutina čiji je rad opisan u TJA1100 dokumentaciji. Nakon pokretanja svake od ovih pet rutina potrebno je pokrenuti rutinu za testiranje noramlnog režima rada, jer ove rutine dovode do prekida komunikacije između namenske platforme i računara. Stanja i rezultat rutine se smešta u polja routine\_status i *routine\_result* *\_diag\_data\_struct* strukture.

Stanja rutine:

1. ROUTINE\_IDLE – početno stanje
2. ROUTINE\_ACTIVE – rutina pokrenuta, kada se pokrene bilo koja od pet testnih rutina
3. ROUTINE\_ABORTED – rutina prekinuta, kada se pozove rutina za testiranje normalnog režima rada
4. ROUTINE\_FINISHED – rutina završena, samo kod ping rutine

Vrednosti rutine:

1. ROUTINE\_NORESULT – početno stanje, nema rezultata rutine
2. ROUTINE\_CORRECTRESULT – rutina vratila tačan rezultat, samo kod ping rutine

* **LOCAL void \_normalOperationTest (uint16\_t mode)**

Funkcija za testiranje normalnog režima rada TJA1100 namenskog integrisanog kola upisivanjem jedinice u AUTO\_OP polje konfiguracionog registra jedan, čime se integrisano kolo vraća u automatski režim.

* **LOCAL STATUS \_pingRoutine(void)**

Rutina koja poziva *ping6()* funkciju za proveru veze sa računarom. U slučaju da je veza uspostavljena vraća OK, u suprotnom ERROR.

## Modul za logičku obradu prikupljenih podataka

Modul za logičku obradu podataka realizovan je kao SWC koja se nalazi u korisničkom prostoru, ima zasebnu memoriju i u potpunosti je nezavisna od modula za čitanje registara. Sastoji se od datoteka:

* **moduleTwoRTP.c**
* **moduleTwoServer.c**
* **moduleTwoCommunication.c**
* **moduleTwoHandleRoutines.c**
* **moduleTwoFileSending.c**

Komunikacija između modula jedan i dva realizovana je u datoteci moduleTwoCommunication.c, a komunikacija između modula dva i tri u datoteci moduleTwoServer.c gde se pokreće serverska strana TCP/IP veze.

### Funkcionalnost datoteke moduleTwoRTP.c

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prototip funkcije | Vidljivost | Povratna vrednost | Poziva | Poziva je |
| STATUS module2\_SetRoutineNum(int routineNum); | globalna | OK /  ERROR | / | processMessage()  (*moduleTwoHandleRoutines.c*) |
| int main (void) | globalna | / | \_module2\_init()  \_module2\_ReadDiagMsgQ() | / |
| LOCAL void \_module2\_init(void) | lokalna | / | sharedMemAlloc\_module2()  (*moduleTwoCommunication.c*) initCommunication()  (*moduleTwoServer.c*)  \_backgroundTask() | main() |
| LOCAL void \_backgroundTask(void) | lokalna | / | processMessage()  (*moduleTwoHandleRoutines.c*) | \_module2\_init() |

Tabela 4.6 Funkcije datoteke moduleTwoRTP.c

* **int main(void)**

Svaki RT proces, odnosno svaka SWC, mora imati *main()* funkciju koja se ponaša kao ulazna tačka za zadatak (eng. *Task).* Poziva funkciju *\_module2\_init()* koja izvršava sve inicijalizacije i podešavanja modula. Potom u beskonačnoj petlji pozivanjem funkcije *\_module2\_ReadDiagMsgQ()* čita *message queue* sa dijagnostičkim podacima na svakih 250ms.

* **LOCAL void \_module2\_init(void)**

Ova funkcija ima zadatak da pozove odgovarajuće funkcije i tako obavi podešavanje modula. Prvo se poziva funkcija *sharedMemAlloc\_module2(),* koja se nalazi u datoteci *moduleTwoCommunication.c* čime se zauzme prostor za deljenu memoriju. Zatim se napravi *message queue* koji će služiti za slanje naredbi u automatu sa konačnim brojem stanja. Promenljiva *\_changeState* se postavlja na 1 što predstavlja stanje STATEMACHINE\_WAIT\_FOR\_COMMAND i šalje se u *message queue*. Ovo je početno stanje automata. Funkcionisanje automata biće objašnjeno kada budu opisivane ostale datoteke koje pripadaju modulu dva.

Pozivom funkcije *initCommunication()*, koja je realizovana u datoteci *moduleTwoServer.c* kreira se serverska strana veze između namenske platforme i računara.

Konačno, kreira se novi zadatak, *“bgTask”,* koji je pozadinski zadatak u kom je realizovan rad automata.

* **LOCAL void \_backgroundTask(void)**

Predstavlja funkciju pozadinskog zadatka. U beskonačnoj petlji prima promenljivu *\_msg* u kojoj se nalazi stanje u koje automat treba da uđe i poziva funkciju *processMessage(),* koja se nalazi u datoteci *moduleTwoHandleRoutines.c.* Potom se u *message queue* ponovo upisuje 1 čime se automat vraća u početno stanje. U slučaju da je primljeno stanje STATEMACHINE\_EXIT\_BG\_TASK automat prestaje sa radom i serverska strana veze se gasi kao i čitav RT proces. Na kraju, promenljivu *\_changeState* ponovo upisuje u *message queue* čime se automat vraća u početno stanje.

* **STATUS module2\_SetRoutineNum (int routineNum)**

Ova funkcija kao parametar prima broj rutine koju je korisnik zatražio. Otvara odgovarajući javni *message queue* sa identifikacionim brojem *routinesMsgQId* šalje broj rutine u modul jedan koji izvršava rutinu. Povratna vrednost funkcije je OK, odnosno ERROR ukoliko je došlo do greške prilikom otvaranja ili slanja broja rutine kroz *message queue*.

### Funkcionalnost datoteke moduleTwoServer.c

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prototip funkcije | Vidljivost | Povratna vrednost | Poziva | Poziva je |
| void initCommunication(void) | globalna | / | \_establishCommunication() | \_module2\_init()  (*moduleTwoRTP.c*) |
| LOCAL void \_establishCommunication(void) | lokalna | / | / | initCommunication() |

Tabela 4.7 Funkcije datoteke moduleTwoServer.c

* **void initCommunication(void)**

Funkcija koja kreira *socket* za TCP/IP vezu sa računarom. Postavlja odgovarajuće parametre u strukturu *sockaddr\_in* i poziva *bind()* čime se povezuje adresa na priključak (eng. *Port).* Vrši pozivanje funkcije *\_establishCommunication().*

* **LOCAL void \_establishCommunication(void)**

Funkcija poziva *listen(),* koja je blokirajuća funkcija, i čeka na klijenta da zatraži povezivanje. Nakon što dobije zahtev od klijenta poziva *accept()* čime prihvata taj zahtev za povezivanje i obavlja rukovanje (eng. *Three-Way Handshake*). U zavisnosti od toga da li je povezivanje bilo uspešno obaveštava klijenta odgovarajućom porukom.

### Funkcionalnost datoteke moduleTwoCommunication.c

Obavlja komunikaciju sa modulom jedan preko deljene memorije i *message queue*-a.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prototip funckije | Vidljivost | Povratna vrednost | Poziva | Poziva je |
| STATUS \_module2\_ReadDiagMsgQ(void) | globalna | OK / ERROR | \_module2\_shMem\_Check() | main()  (*moduleTwoRTP.c*) |
| STATUS sharedMemAlloc\_module2(void) | globalna | OK /  ERROR | \_module2\_shMem\_open() | \_module2\_init()  (*moduleTwoRTP.c*) |
| LOCAL void \* \_module2\_shMem\_open(const char \* fname, size\_t size) | lokalna | / | / | sharedMemAlloc\_module2() |

Tabela 4.8 Funkcije datoteke moduleTwoCommunication.c

* **STATUS \_module2\_ReadDiagMsgQ**

Ova funkcija otvara *message queue* sa dijagnostičkim podacima i prima pokazivač na strukturu u kojoj se oni nalaze. U slučaju greške prilikom otvaranja ili čitanja *message queue*-a vraća ERROR, inače OK.

* **STATUS sharedMemAlloc\_module2(void)**

Poziva funkciju *\_module2\_shMem\_open()* koja vraća pokazivač na deljenu memoriju sa dijagnostičkim podacima. Vraća OK, ili ERROR u slučaju da *\_module2\_shMem\_open()* vrati NULL pokazivač.

* **LOCAL void \* \_module2\_shMem\_open (const char \* fname, size\_t size)**

Parametri ove funkcije su ime deljene memorije i njena veličina. Funkcionalnost se ogleda u otvaranju deljene memorije koja je kreirana u modulu jedan, odnosno pravljenju objekta, postavljanju veličine objekta i mapiranja memorije u adresni prostor zadatka. Povratna vrednost je pokazivač na mapiranu adresu ili NULL pokazivač u slučaju greške.

### Funkcionalnost datoteke moduleTwoHandleRoutines.c

Datoteka u kojij se nalazi realizacija automata sa konačnim brojem stanja.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prototip funkcije | Vidljivost | Povratna vrednost | Poziva | Poziva je |
| void processMessage(void) | globalna | / | \_receiveCommand()  uploadFile()  (*moduleTwoFileSending.c*) module2\_SetRoutineNum()  (*moduleTwoRTP.c*) | \_backgroundTask()  (*moduleTwoRTP.c*) |
| LOCAL void \_receiveCommand(void) | lokalna | / | / | / |

Tabela 4.9 Funkcije datoteke moduleTwoHandleRoutines.c

* **void processMessage(void)**

Jednostavna *switch-case* naredba koja u zavisnosti od globalne promenljive *\_msg* ulazi u odgovarajuće stanje. Postoji ukupno šest stanja automata:

1. STATEMACHINE\_WAIT\_FOR\_COMMAND

Poziva funkciju \_*receiveCommand()* koja prihvata naredbu od korisnika putem TCP/IP veze.

1. STATEMACHINE\_SEND\_FILE

Poziva funkciju *uploadFile()* koja se nalazi u datoteci *moduleTwoFileSending.c.*

Ova funkcija šalje *log* datoteku na računar putem TCP/IP veze.

1. STATEMACHINE\_START\_ROUTINE

Poziva funkciju *module2\_SetRoutineNum()* kojoj prosleđuje broj rutine koju je korisnik zatražio i vraća automat u početno stanje.

1. STATEMACHINE\_SEND\_MSG\_QUEUE

Šalje dijagnostičke podatke iz *message queue*-a korisniku na zahtev putem *socket*-a i vraća automat u početno stanje.

1. STATEMACHINE\_SEND\_SHARED\_MEMORY

Šalje dijagnostičke podatke iz deljene memorijekorisniku na zahtev putem *socket*-a i vraća automat u početno stanje.

1. Stanje kada korisnik unese nulu, zatvara se konekcija i RT proces prestaje sa radom.

* **LOCAL void \_receiveCommand(void)**

Funkcija koja prihvata naredbu od korisnika i postavlja automat u odgovarajuće stanje.

### Funkcionalnost datoteke moduleTwoFileSending.c

Datoteka u kojoj je realizovano slanje *log* datoteke na računar putem TCP/IP protokola.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prototip funkcije | Vidljivost | Povratna vrednost | Poziva | Poziva je |
| void uploadFile(void) | globalna | / | \_sendFile() | processMessage()  (*moduleTwoHandleRoutines.c*) |
| LOCAL void \_sendFile(char fs\_name[]) | lokalna | / | \_processFileName()  \_processFileSize()  \_processFileSending() | uploadFile() |
| LOCAL void \_processFileName(char fs\_name[]); | lokalna | / | / | \_sendFile() |
| LOCAL void \_processFileSize(void); | lokalna | / | / | \_sendFile() |
| LOCAL void \_processFileSending(void); | lokalna | / | / | \_sendFile() |

Tabela 4.10 Funkcije datoteke moduleTwoFileSending.c

* **void uploadFile(void)**

Početna funkcija slanja *log* datoteke. Stavlja automat u početno stanje i šalje broj fajlova koje klijent treba da primi, što je u ovom slučaju jedan. Otvara direktorijum koji se nalazi na namenskoj platformi na putanji *“/mmc0:1/err”* i smešta naziv *log* datoteke u promenjivu *tempStr.* Ovu promenljivu prosleđuje funkciji *\_sendFile()* kao parametar.

* **LOCAL void \_sendFile(char fs\_name[])**

Funkcija koja kao parametar prima naziv *log* datoteke i poziva funkcije \_*processFileName(),* koja kao parametar prima isti taj naziv datoteke, zatim *\_processFileSize() i \_processFileSending(),* nakon čega briše datoteku sa namenske platforme.

* **LOCAL void \_processFileName(char fs\_name[])**

Funkcija koja šalje naziv datoteke klijentu i spaja naziv datoteke sa putanjom na kojoj se ona nalazi.

* **LOCAL void \_processFileSize(void)**

Otvara datoteku ako ne postoji i dobavlja veličinu iste pomoću ugrađenih funkcija *fseek() i ftell().* Veličinu datoteke zatim šalje klijentu.

* **LOCAL void \_processFileSending(void)**

Postavlja pokazivač na početak datoteke pomoću ugrađene funkcije *fseek().* U petlji funkcijom *fread()* čita komade datoteke od 512 bajtova i šalje ih na klijentsku stranu sve dok ne dođe do kraja datoteke.

## Modul za interaktivnu vizuelizaciju podataka

Ovaj modul je realizovan u Qt programskom okruženju kombinacijom programskog jezika C i Qt mehanizma signala i slotova. Predstavlja klijentsku stranu u TCP/IP komunikaciji između računara i namenske platforme. Pritiskom na dugme „Start Client“ aplikacija se povezuje sa serverom, odnosno modulom dva. Korisnik može da zatraži čitanje deljene memorije ili *message queue*-a čiji se rezultati ispisuju u interfejsu aplikacije. Takođe može da zatraži pokretanje neke od rutina unošenjem broja 1-7 u polje pored dugmeta „Start Routine“. Ukoliko se pokrene neka od testnih rutina unošenjem broja 1-5 potrebno je vratiti se u normalan režim rada unošenjem broja 0. Nakon pokretanja rutine za proveru komunikacije unošenjem broja 6 rezultati se mogu proveriti pritiskom na dugme „Get MsgQ“. Rezultati pokretanja rutine sa brojem 7, koja proverava koliko puta je došlo do pada komunikacije od poslednje provere, mogu se videti pritiskom na dugme „Get ShMem“. Preuzimanje *log* datoteke se obavlja periodično na svakih 15s u posebnom *thread*-u. Izlaz iz aplikacije se vrši pritiskom na dugme „Exit“.

### Funkcionalnost datoteke main.cpp

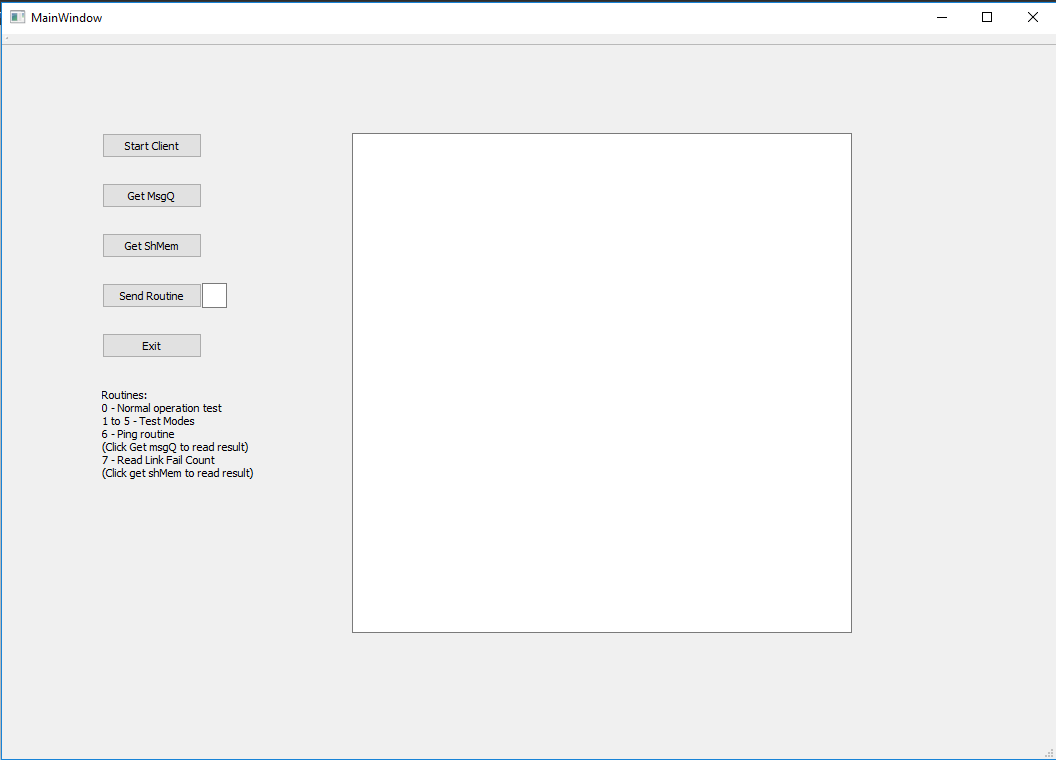
Ova datoteka ima samo jednu funkciju **int main(int argc, char \*argv[])**. Ona stvara objekat klase *MainWindow* i pokreće aplikaciju.

### Funkcionalnost datoteke mainwindow.cpp

U ovoj datoteci je realizovan Qt mehanizam signala i slotova. Pritiskom na dugme u aplikaciji generiše se signal koji je vezan za odgovarajući slot i aktivira se tražena funkcija. Funkcije **handleMsgQ\_button(), handleShMem\_button(), handleRoutineClick() i handleDisconnectClick()** služe kao sprega između akcije korisnika i reakcije u vidu poziva odgovarajuće funkcije.

* **MainWindow::MainWindow (Qwidget \*parent) : QMainWindow (parent), ui (new Ui::MainWindow)**

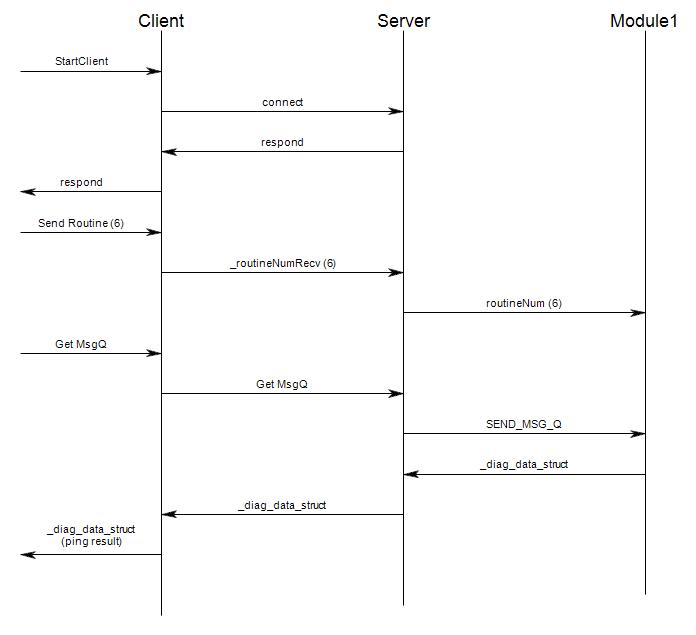
Funkcija koja kreira izgled aplikacije i objekte poput dugmića i mesta za unos kao i prikaz podataka. Vrši otvaranje datoteke u koju se preusmerava sav ispis koji ide na standardni izlaz i povezuje signale sa odgovarajućim slotovima.



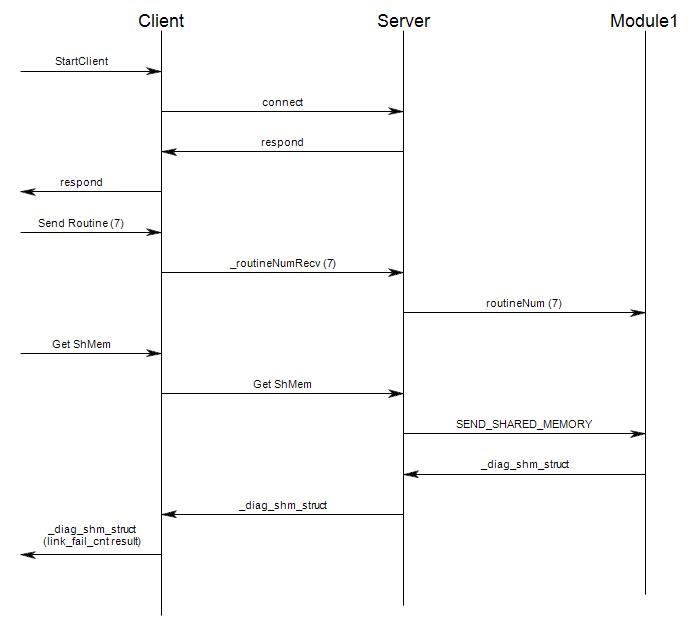
Slika 4.4 Izgled klijentske aplikacije

* **void MainWindow::insertRoutine()**

Aktivira se pritiskom na dugme *„Send routine“* i unosom broja željene rutine iz opisa na dnu aplikacije. Čita broj naredbe koju je korisnik uneo i prosleđuje broj 2 funkciji *startModule()*, a uneti broj prosleđuje funkciji *sendRoutineNumber().* Dvojka se šalje iz razloga što se server prvo mora obavestiti da korisnik želi pokrenuti neku od rutina, a potom se šalje i sam broj rutine. Standardni izlaz se čisti i upisani podaci se čitaju iz datoteke na koju je preusmeren. Na kraju, i sadržaj datoteke se briše.



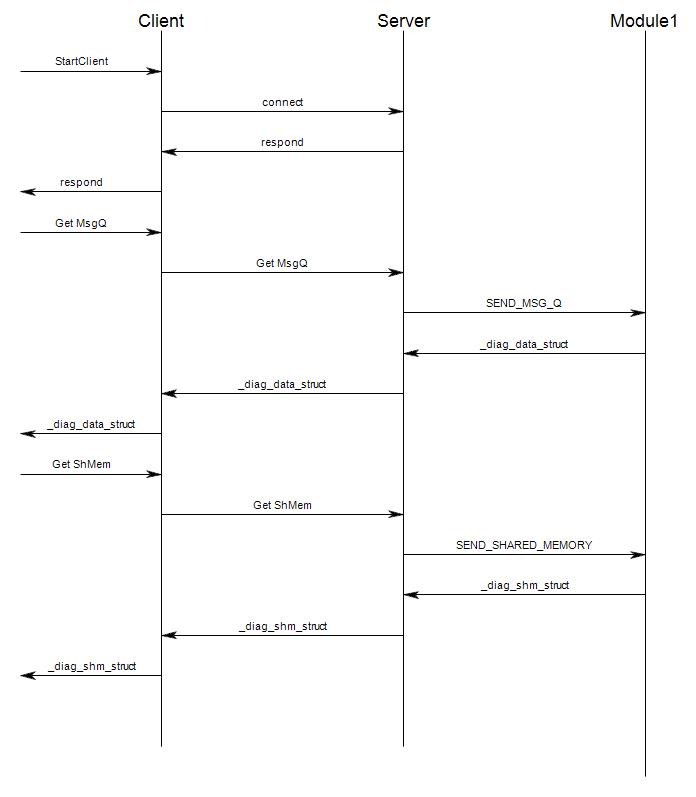
Slika 4.5 Primer *ping* zahteva i odgovora



Slika 4.6 Primer dobavljanja *link\_fail* brojača

* **void MainWindow::getShMem()**

Funkcija se pokreće pritiskom na dugme *„Get ShMem“* i poziva *startModule()* sa parametrom 4 čime se dobavlja i prikazuje sadržaj deljene memorije. Standardni izlaz se čisti i upisani podaci se čitaju iz datoteke na koju je preusmeren. Na kraju, i sadržaj datoteke se briše. Sličnu ulogu ima i funkcija **void MainWindow::getMsgQ()** koja čita sadržaj *message queue*-a.



Slika 4.7 Primer uspešnog preuzimanja dijagnostičkih podataka

* **void MainWindow::initComm()**

Pokreće se pritiskom na dugme *„Start Client“.* Poziva funkciju *initCommunication()* koja uspostavlja vezu sa namenskom platformom preko TCP/IP protokola. Standardni izlaz se čisti i upisani podaci se čitaju iz datoteke na koju je preusmeren. Na kraju, i sadržaj datoteke se briše.

* **void MainWindow::terminateProgram()**

Pritiskom na dugme *„Exit“* poziva funkciju *startModule()* sa parametrom 0 čime započinje prekidanje veze sa serverom, odnosno namenskom platformom, i gašenje aplikacije. Standardni izlaz se čisti i upisani podaci se čitaju iz datoteke na koju je preusmeren. Na kraju, i sadržaj datoteke se briše.

* **void MainWindow::readFile()**

Otvara datoteku *output\_file.txt* u koju je preusmeren standardni izlaz i čita liniju po liniju datoteke koju ispisuje u prozoru aplikacije.

* **void MainWindow::clearFile()**

Briše sadržaj datoteke koji je preusmeren sa standardnog izlaza.

### Funkcionalnost datoteke module3.cpp

Ova datoteka sadrži funkcije napisane u C programskom jeziku u kojima je realizovana funkcionalnost modula.

* **void Module3::initCommunication(void)**

Ova funkcija pokreće *thread* koji služi za periodično preuzimanje *log* datoteke sa namenske platforme. Sem toga, ona vrši podešavanje i pokretanje klijentske strane kreiranjem *socket*-a. U strukturu *sockaddr\_in* se upisuje broj porta i IPv4 adresa namenske platforme, i potom pozivom funkcije *connect()* vrši povezivanje na server. Nakon toga, vrši se procedura rukovanja (eng. *Three-Way Handshake*) kojom se utvrđuje da li je veza uspostavljena.

* **int Module3::startModule (int insertedNumber)**

Parametar funkcije je broj koji zavisi od akcije korisnika. Taj broj se prosleđuje serveru i čeka se na odgovor ukoliko su to naredbe dobavljanja deljene memorije, *message queue*-a ili naredba za završetak programa.

* **void Module3::sendRoutineNum (int rNum)**

Parametar funckije je broj rutine koju je uneo korisnik u aplikaciju. Ovaj broj se prosleđuje serverskoj strani čime se pokreće tražena rutina.

* **void Module3::printStates(void)**

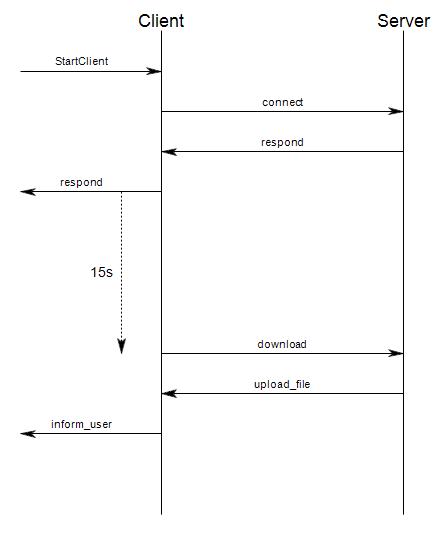
Funkcija koja prikazuje dijagnostičke podatke.

* **void \*dequeueLoopWrapper (void \*arg)**

Funkcija koja *pthread* metodu za pokretanje *thread*-a prebacuje u Qt *thread* metodu.

* **void \*Module3::threadfunc(void \*arg)**

*Thread* funkcija koja se izvršava na svakih 15s i preuzima *log* datoteku pozivom funkcije *receiveFile().* Pre njenog poziva postavlja broj naredbe na 1 i obaveštava serversku stranu da treba da pošalje datoteku. Takođe prima i broj datoteka koje će server poslati, što je u ovom slučaju 1.



Slika 4.8 Primer preuzimanja *errorLog.txt* datoteke

* **void Module3::receiveFile()**

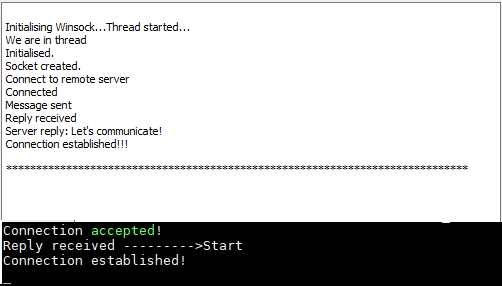
Obavlja funkcionalnost preuzimanja *log* datoteke. Prihvata ime datoteke „errorLog.txt“ i menja ga u *„errorLog\_n.txt“* gde n počinje od 0 i uvećava se svaki sledeći put kad se datoteka preuzima. Potom prihvata veličinu datoteke i računa broj paketa koje treba da primi kao i ostatak. U beskonačnoj petlji prihvata paket po paket i upisuje u datoteku na računaru dok ne dođe do kraja.

# Rezultati

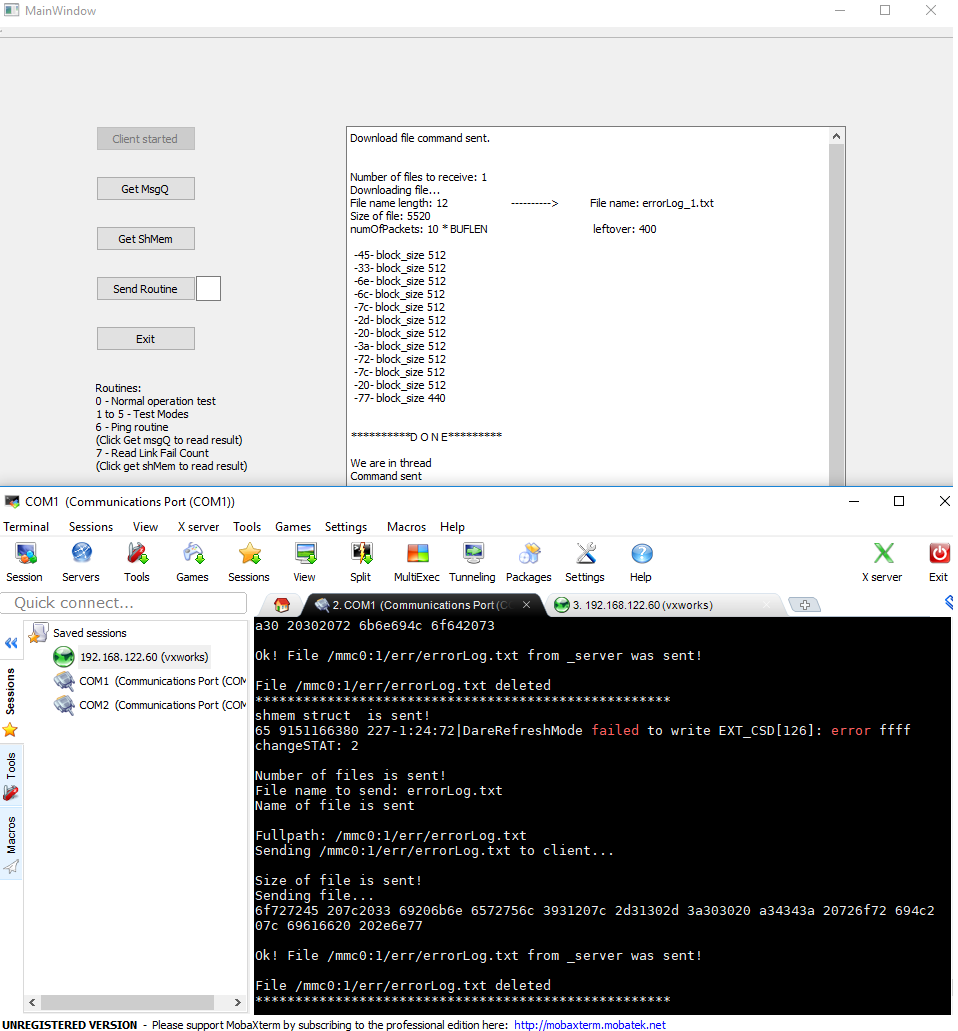
Testiranja su obavljena u tri faze. Prva faza obuhvata testiranje modula jedan, odnosno provere da li su adrese dobro mapirane, da li su sve potrebne memorije inicijalizovane na pravi način i izvršena je provera rezultata čitanja registara ispisivanjem istih putem UART sprege. Validnost pročitanih registara potvrđena je upoređivanjem dobijenih vrednosti i vrednosti koje se nalaze u dokumentaciji za TJA1100 integrisano kolo. Rad Eternet i VLAN sprege proverena je unošenjem komande *ping6 „pcIPv6address“* u komandni prozor namenske platforme, kao i proverom komunikacije sa strane računara komandom *ping -6 „brIPv6address“.*

Druga faza obuhvatila je testiranje modula dva, odnosno otvaranje *message-queue* poruka i deljene memorije i utvrđivanje poklapanja dijagnostičkih podataka sa podacima iz modula jedan. Proveren je rad TCP/IP veze sa računarom, kao i rad automata sa konačnim brojem stanja.

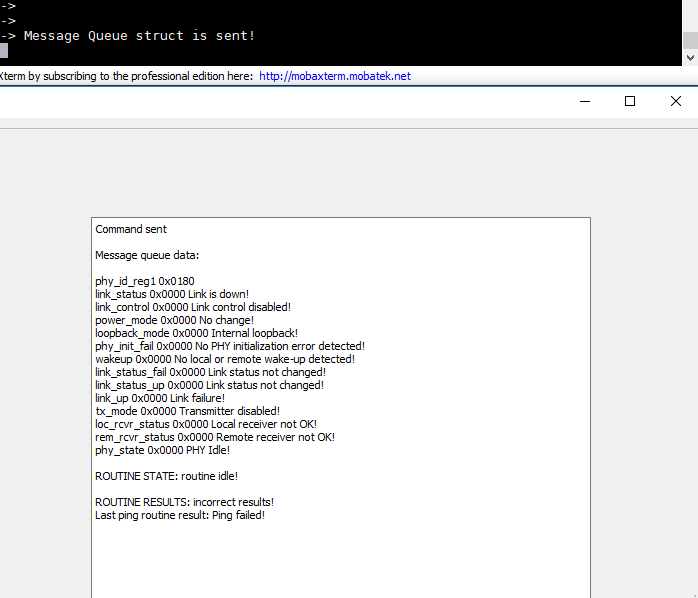
Treća faza obuhvatila je proveru ispravnosti rada čitavog sistema.



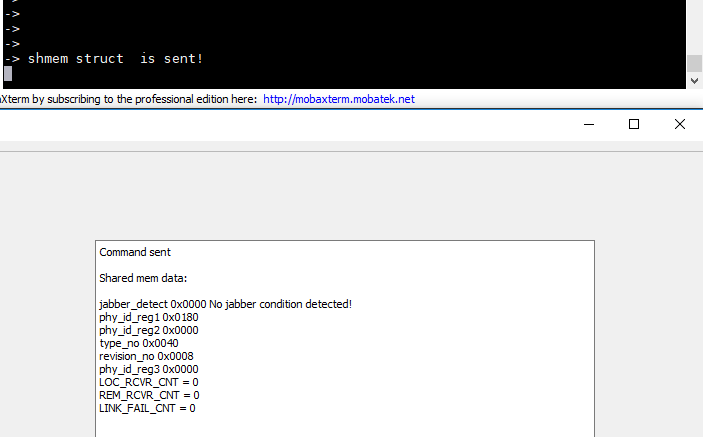
Slika 5.1 Uspešno pokretanje aplikacije (gore) i prihvatanje veze (dole)



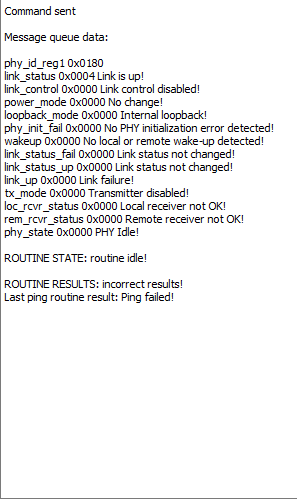
Slika 5.2 Preuzimanje *errorLog.txt* datoteke



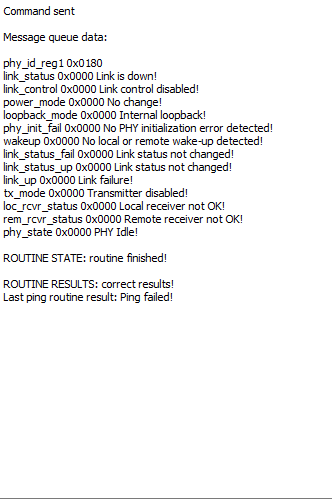
Slika 5.3 Preuzimanje *message queue* dijagnostičkih podataka



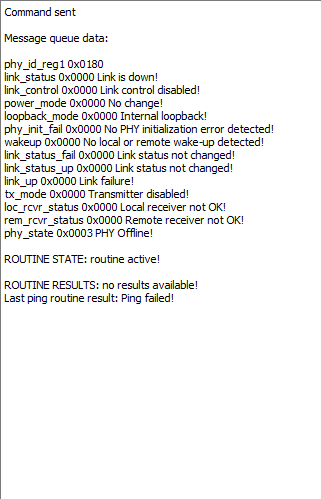
Slika 5.4 Preuzimanje deljene memorije



Slika 5.5 Veza uspostavljena

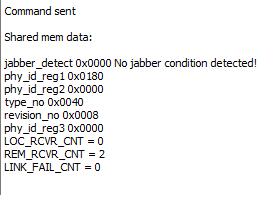


Slika 5.6 Veza prekinuta i provera konekcije bezuspešna



Slika 5.7 Pokrenuta testna rutina 4

Rutina za proveru koliko puta je došlo do prekida veze testirana je fizičkim prekidanjem veze nekoliko puta i očitavanjem deljene memorije nakon toga.



Slika 5.8 Izgled deljene memorije nakon fizičog prekida veze dva puta

# Zaključak

Smisao ovog zadatka bio je upoznavanje sa mini V modelom i sa ISO-26262 sigurnosnim konceptom za rad u *automotive* projektima. Takođe, bilo je potrebno upoznati se sa konceptom dvožilne *BroadR-Reach* eternet komunikacije, podacima eternet veze koji su bitni za dijagnostiku i način kako ih čitati sa TJA1100 integrisanog kola. Zatim bilo je potrebno upoznati se sa procesom formiranja SWC komponente u RTOS operativnom sistemu i vezama između RT procesa po AUTOSAR RTE okruženju. Na kraju provera realizacije zadatka obavljena je upotrebom QAC alata koji poštuje MISRA standard. Tokom realizacije programskog rešenja, sem MISRA standard, bilo je potrebno ispoštovati *coding guideline* zajedno sa HIS metrikom.

Realizacija ovog zadatka mogla je biti poboljšana na više načina, prvenstveno otvaranjem još jednog *socket*-a u modulu dva i još jednog *thread*-a u modulu tri čime bi se uz manje izmene dobilo dobavljanje podataka u aplikaciju u realnom vremenu. Takodje mogla se u modulu dva uvesti neka logika sortiranja dijagnostičkih podataka po važnosti i eventualno uvesti rukovanje greškama. Međutim, kako nisam bio upoznat sa većinom koncepata koji su bili korišćeni i kako sam učio ove koncepte u toku izrade programskog rešenja, tek pred kraj sam uvideo ove mogućnosti poboljšanja.

# Literatura

1. ……...
2. ………
3. ………
4. ……….
5. *………*