|  |  |
| --- | --- |
| |  | | --- | | **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: Филип Јашић**  **Број индекса: РА46/2014**  **Тема рада: TODO**  **Ментор рада: доц. др Богдан Павковић**  **Нови Сад, јул, 2019.** | |

|  |  |  |  |
| --- | --- | --- | --- |
| 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**: | | **Филип Јашић** | |
| Ментор, **МН**: | | **доц. др Богдан Павковић** | |
| Наслов рада, **НР**: | | TODO | |
| Језик публикације, **ЈП**: | | Српски / ћирилица | |
| Језик извода, **ЈИ**: | | Српски | |
| Земља публиковања, **ЗП**: | | Република Србија | |
| Уже географско подручје, **УГП**: | | Војводина | |
| Година, **ГО**: | | **2019.** | |
| 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) | | **7/58/14/3/38/0/0** | |
| Научна област, **НО**: | | Електротехника и рачунарство | |
| Научна дисциплина, **НД**: | | Рачунарска техника | |
| Предметна одредница/Кључне речи, **ПО**: | |  | |
| **УДК** | |  | |
| Чува се, **ЧУ**: | | У библиотеци Факултета техничких наука, Нови Сад | |
| Важна напомена, **ВН**: | |  | |
| Извод, **ИЗ**: | | TODO | |
| Датум прихватања теме, **ДП**: | |  | |
| Датум одбране, **ДО**: | |  | |
| Чланови комисије, **КО**: | Председник: | проф. др |  |
|  | Члан: | доц. др | Потпис ментора |
|  | Члан, ментор: | доц. др |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 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**: | | **Filip Jašić** | |
| Mentor, **MN**: | | **Bogdan Pavkovic, PhD** | |
| Title, **TI**: | | TODO | |
| 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**: | | **2019.** | |
| Publisher, **PB**: | | Author’s reprint | |
| Publication place, **PP**: | | Novi Sad, Dositeja Obradovica sq. 6 | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | **7/48/14/0/12/0/0** | |
| Scientific field, **SF**: | | Electrical Engineering | |
| Scientific discipline, **SD**: | | Computer Engineering, Engineering of Computer Based Systems | |
| Subject/Key words, **S**/**KW**: | |  | |
| **UC** | |  | |
| Holding data, **HD**: | | The Library of Faculty of Technical Sciences, Novi Sad, Serbia | |
| Note, **N**: | |  | |
| Abstract, **AB**: | | TODO | |
| Accepted by the Scientific Board on, **ASB**: | |  | |
| Defended on, **DE**: | |  | |
| Defended Board, **DB**: | President: | , PhD |  |
|  | Member: | , PhD | Menthor's sign |
|  | Member, Mentor: | ,PhD |  |

**Захвалност**

**Садржај**

[1. Увод 7](#_Toc12369613)

[2. Теоријске основе 9](#_Toc12369614)

[*2.1* *CAN*  9](#_Toc12369615)

[2.1.1 Физички слој *CAN* – а 10](#_Toc12369616)

[2.1.2 Електрична својства, каблови и утичнице 12](#_Toc12369617)

[2.1.3 Порука, формат оквира и пренос података 13](#_Toc12369618)

[2.1.4 Арбитража 17](#_Toc12369619)

[2.1.5 Убацивање бита, препознавање грешака, сигурност 18](#_Toc12369620)

[*2.2* *LIN*  20](#_Toc12369621)

[2.2.1 Топологија и физички слој 21](#_Toc12369622)

[2.2.2 *LIN* оквир 22](#_Toc12369623)

[*2.3* *SPI* 25](#_Toc12369624)

[2.3.1 Пренос података, фаза и поларитет 26](#_Toc12369625)

[*2.4* *I²C* 29](#_Toc12369626)

[2.4.1 Начин функционисања 30](#_Toc12369627)

[2.5 Упоредно поређење протокола 31](#_Toc12369628)

[2.6 Осцилоскоп 32](#_Toc12369629)

[3. Концепт решења 34](#_Toc12369630)

[3.1 Повезивање *CAN* – и мерење са осцилоскопом 34](#_Toc12369631)

[3.2 Повезивање *LIN* – а и мерење са осцилоскопом 36](#_Toc12369632)

[3.3 Повезивање *SPI* – а и мерење са осцилоскопом 39](#_Toc12369633)

[3.4 Повезивање *I2C* – а и мерење са осцилоскопом 40](#_Toc12369634)

[4. Програмско решење 42](#_Toc12369635)

[4.1 Комуникација са осцилоскопом, мерење и обрада сигнала 42](#_Toc12369636)

[*4.2* *CANoe* 43](#_Toc12369637)

[4.3 Декодовање *CAN* – а 45](#_Toc12369638)

[4.4 Декодовање *LIN* – а 47](#_Toc12369639)

[4.5 Декодовање *SPI* – а 49](#_Toc12369640)

[4.6 Декодовање *I2C* – а 50](#_Toc12369641)

[5. Тестирање и резултати 52](#_Toc12369642)

[6. Закључак 54](#_Toc12369643)

[7. Литература 55](#_Toc12369644)

**Списак слика**

[Слика 2.1 Брза *CAN* мрежа 10](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369573)

[Слика 2.2 Доминантни и рецесивни бит *CAN* мреже 11](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369574)

[Слика 2.3 Омеђавање линија за брзу *CAN* линију 12](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369575)

[Слика 2.4 Омеђавање линија за спору *CAN* линију 12](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369576)

[Слика 2.5 Утичница *DE-9* 13](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369577)

[Слика 2.6 Илустративни пример *CAN 2.0 A* оквира без уметања бита и његова корелација са напоном на *CAN-H* и *CAN-L* 13](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369578)

[Слика 2.7 Илустративно објашњење како фунционише *DLC* 15](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369579)

[Слика 2.8 Илустративни приказ *LIN* оквира за подаке 15](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369580)

[Слика 2.9 Пример арбитраже 17](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369581)

[Слика 2.10 Пример убацивања бита 18](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369582)

[Слика 2.11 Топологија типичне *LIN* мреже 21](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369583)

[Слика 2.12 Илустрација мапирања идентификатора и парности унутар поља за заштићени идентификатор 23](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369584)

[Слика 2.13 Пример *LIN* оквира 24](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369585)

[Слика 2.14 Пример повезивања *SPI* руководиоца и подређеног уређаја 27](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369586)

[Слика 2.15 Пример преноса података између *SPI* руководиоца и подређеног 27](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369587)

[Слика 2.16 Режими рада *SPI* 28](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369588)

[Слика 2.17 Пример фазе и поларитета за *SPI* 29](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369589)

[Слика 2.18 Временски дијаграм *I²C* 31](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369590)

[Слика 2.19 *Tektronix DPO 4104B* 33](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369591)

[Слика 3.1 Приказ екрана осцилоскопа приликом снимања *CAN* сигнала 35](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369592)

[Слика 3.2 Дијаграм повезивања *CAN* 35](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369593)

[Слика 3.3 *Raspberry Pi 2* модел Б,преглед пинова 36](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369594)

[Слика 3.4 Илустровани приказ главне утичнице за *Carberry* модул 37](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369595)

[Слика 3.5 Приказ екрана осцилоскопа приликом снимања *LIN* сигнала 38](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369596)

[Слика 3.6 Дијаграм повезивања *LIN*-a 38](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369597)

[Слика 3.7 Приказ екрана осцилоскопа приликом снимања *SPI* сигнала 39](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369598)

[Слика 3.8 Дијаграм повезивања *SPI* 40](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369599)

[Слика 3.9 Приказ екрана осцилоскопа приликом снимања *I2C* сигнала 41](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369600)

[Слика 3.10 Дијаграм повезивања *I2C* 41](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369601)

[Слика 4.1 Приказ *Simulation* *Setup* прозора унутар *CANoe* - а 44](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369602)

[Слика 4.2 Приказ *I-Generator* прозора 44](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369603)

[Слика 4.3 Приказ *Canoe CANdb++ Editor* - а 45](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369604)

[Слика 4.4 Приказ ухваћеног необрађеног *CAN* сигнала са осцилоскопа 46](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369605)

[Слика 4.5 Приказ ухваћеног *CAN* сигнала са осцилоскопа након обраде 47](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369606)

[Слика 4.6 Приказ обрађеног LIN сигнала 49](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369607)

[Слика 4.7 Приказ обрађеног LIN сигнала 49](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369608)

[Слика 4.8 Пример необрађеног *I2C* сигнала 51](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369609)

[Слика 4.9 Приказ обрађеног *I2C* сигнала 51](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369610)

[Слика 5.1 Пример излаза успешног декодовања 52](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369611)

[Слика 5.2 Кашњење између *CAN* оквира 53](file:///C:\Users\fjasic\Downloads\Filip_Jasic_RA46-2014-v2.docx#_Toc12369612)

**Списак табела**

[Табела 1 Разлика између *CAN 2.0 A* и *CAN 2.0 B* 11](#_Toc11928133)

[Табела 2 Упоредно поређење протокола 26](#_Toc11928134)

[Табела 3 Опис жица главне утичнице за *Carberry* модул 31](#_Toc11928135)

[Табела 4 *Carberry* команде за *LIN* 42](#_Toc11928136)

**Скраћенице**

CAN – **C**ontroller **A**rea **N**etwork

LIN – **L**ocal **I**nterconnect **N**etwork

SPI – **S**erial **P**eripheral **I**nterface

I²C – **I**nter-**I**ntegrated **C**ircuit

ISO - **I**nternational **O**rganization for **S**tandardization

Mb/s – **m**ega**b**yte per **s**econd

kb/s – **k**ilobytes **p**er **s**econd

CRC - **C**yclic **r**edundancy **c**heck

ECU - **E**lectronic **C**ontrol **U**nit

RTR – **R**emote **T**ransmission **R**equest

MSB **- M**ost **S**ignificant **B**it

LSB – **L**east **S**ignificant **B**it

SRR – **S**ubstitute **r**emote **r**equest

IDE - **I**dentifier **e**xtension **b**it

UART – **U**niversal **a**synchronous **r**eceiver-**t**ransmitter

ASIC - **A**pplication**-s**pecific **i**ntegrated **c**ircuit

USB – **U**niversal **S**erial **B**us

# Увод

Аутомобилска индустрија [1] је грана индустрије која се бави пројектовањем, развојем, производњом, маркетингом и продајом мотроних возила. Од самог почетка развоја аутомобила, због великих трошкова производње, јавила се потреба за стандардизацијом компоненти које се уграђују у аутомобил. Главни покретач развоја мрежних технологија у аутомобилима су били напреци остварени у електронској индустрији, нарочито у Сједињеним Америчким Државама. Са стриктном контролом штетних гасова од стране закона америчке владе, морао се направити механизам за већи степен контроле самог аутомобила. Ово се постигло уграђивањем рачунара који не само да су побољшали контролу штетних емисија, него су повећали и перформансе аутомобила, сигурност, комфор али су и допринели смањењу трошкова производње. Током година се број електронских модула у аутомобилу усложњавао (*ABS, BCM, ECU, TCU*).

Електронски контролни модул добија своје податке од сензора које користи за израчунавање и конверзију у одређену јединицу, ти израчунати и процесирани подаци се шаљу актуаторима који извршавају одређене функције. Такође понекад је потребно да ови модули међусобно комуницирају између себе (нпр. мотор мора да обавести мењач која је тренутна брзина мотора, а мењач мора даље да обавести остале модуле када се та брзина промени). Овај пренос података треба да буде брз и поуздан. Како се повећавао број свих ових компоненти, дошло је до велике комплексности ожичавања појединих модула. Ово ожичење би морало да буде другачије за сваки модел аутомобила, што би додатно повећавало трошкове производње.

Одговор аутомобилске индустрије на све ове проблеме је да се створи центрлна мрежа у возилу. Модули би били само „прикачени“ на ове мреже и могли би да лакше међусобно комуницирају. Овај дизајн је био лакши за производњу, одржавање и дао је додатну флексибилност за надограђивање додатних модула без промене читаве архитектуре. Сваки модул, који представља чвор на аутомобилској мрежи контролише специфичну компоненту и комуницира са другим модулима по потреби, коришћењем стандардних протокола.

Циљеви аутомобилске мреже и њених протокола су да :

* смање трошкове производње
* имају отпорност на екстерне сметње
* поседују функционалност и у отежаним условима
* робустност и поузданост

Иако су тренутни захтеви аутомобилске мреже за брзином смањени у односу на мреже за неке друге системе, због убрзаног развоја технологија и све већих могућности аутомобила, постојаће и потреба за бржим магистралама.

Задатак овог рада јесте имплементација анализирања и декодовања сигнала који се користе у аутомобилској индустрији, чиме би се олакшала анализа целокупног система.

Рад се састоји из пет целина:

1. **Теоријске основе –** опис протокола и осцилоскопа који се користи за анализу и декодовање сигнала.

2. **Концепт решења –** имплементација повезивања протокола.

3. **Програмско решење –** имплементација декодовања и анализирања сигнала.

4. **Тестирање и резултати –** опис тестирања програмског решења.

5. **Закључак** **–** анализа правца за даљи развој програмског решења.

# Теоријске основе

У овом поглављу су дата објашњења протокола који се користе у аутомобилској индустрији са акцентом на начином повезивања и изгледом оквира сваког од сигнала протокола. Након тога је дат упоредни преглед протокола, као и кратaк опис како функционише осцилоскоп на којем је урађено мерење.

## *CAN*

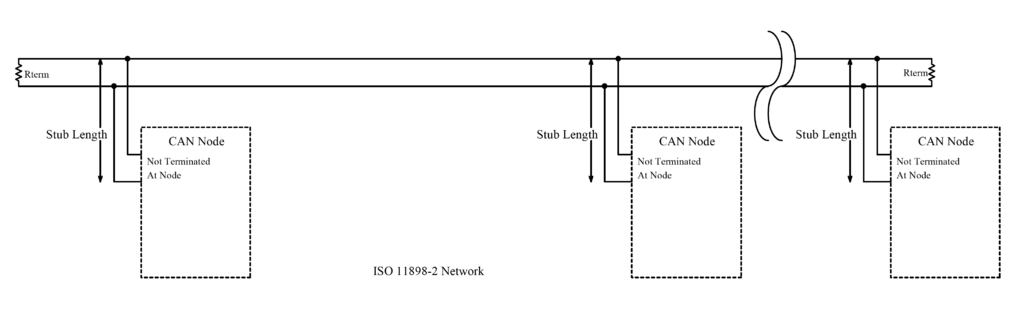
*CAN* [2] протокол је најпознатији аутомобилски протокол. Развијен од стране *Robert Bosch GmbH* 1983. године, док је званично пуштен у употребу 1986. године на *Society of Automotive Engineers (SAE)* конферецији у Детроиту. *Bosch* је објавио више верзија CAN спецификације, а најновија је *CAN 2.0*, објављена 1991. и на тој верзији спецификације се базира декодовање *CAN* сигнала урађено у овом дипломском раду. Ова верзија спецификације се састоји из 2 дела:

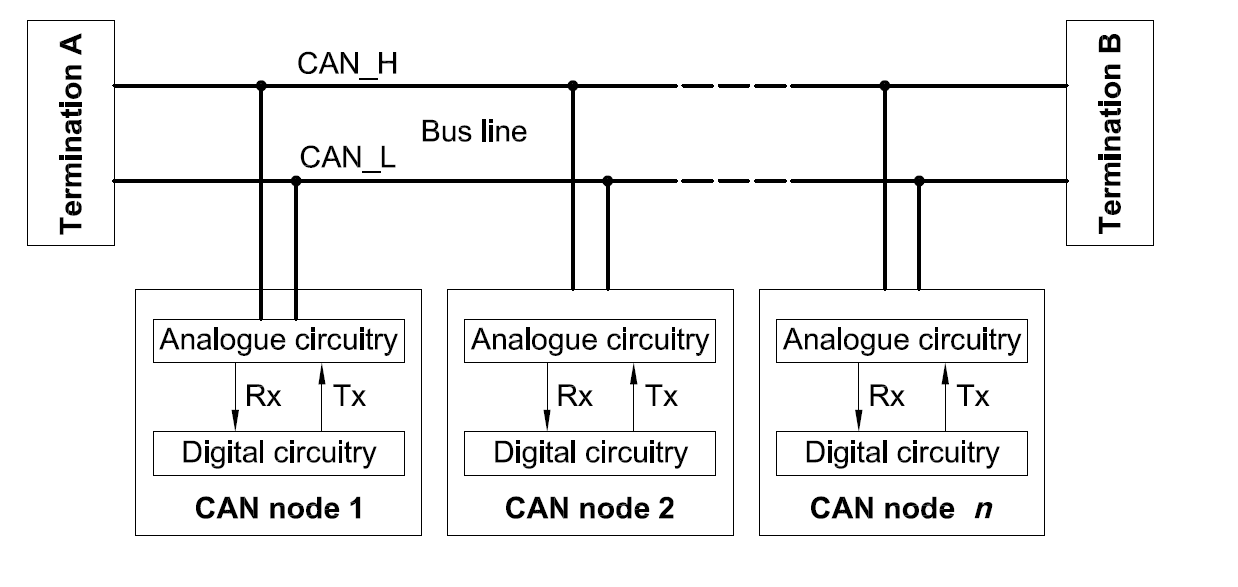
* А (покривен *ISO* 11519 стандардом) – овај део спецификације описује стандардни формат са једанестобитним идентификатором. Мрежа CAN где уређај који користи овакав стандардни идентификатор се често назива *CAN* *2.0 А* и тако ћемо га звати надаље.
* Б (покривен *ISO* 11898 стандардом) – овај део спецификације описује проширени формат са дватесетдеветобитним идентификатором. Мрежа *CAN* где уређај који користи овај продужени идентификатор се често назива *CAN 2.0 В* и тако ћемо га звати надаље.

Неке од карактеристика *CAN* продокола [3] су:

* Брза, серијска, заједничка магистрала (брзине до *1Mbs*) за више уређаја (до 32 уређаја по *ISO* 11898; либералан приступ – било ко може да приступа када је медијум слободан), коришћење упредене парице (домет до 40m)
* Асинхрона комуникација (окидање на догађаје).
* Величина оквира: 0-8 октета.
* Филтрирање на пријему, препознавање грешке (коришшењем цикличне провере редудансе), мере за ограничавање грешака (гашење физички оштећених).
* Одзив у реалном времену.
* Недеструктивна арбитража (100% искоришћеност пропусне моћи).
* Приоритет на основу идентификатора.
* Већа кашњења за поруке ниског приоритета.
* Мала кашњења за поруке високог приоритета

### Физички слој *CAN* – а





Слика . Брза *CAN* мрежа

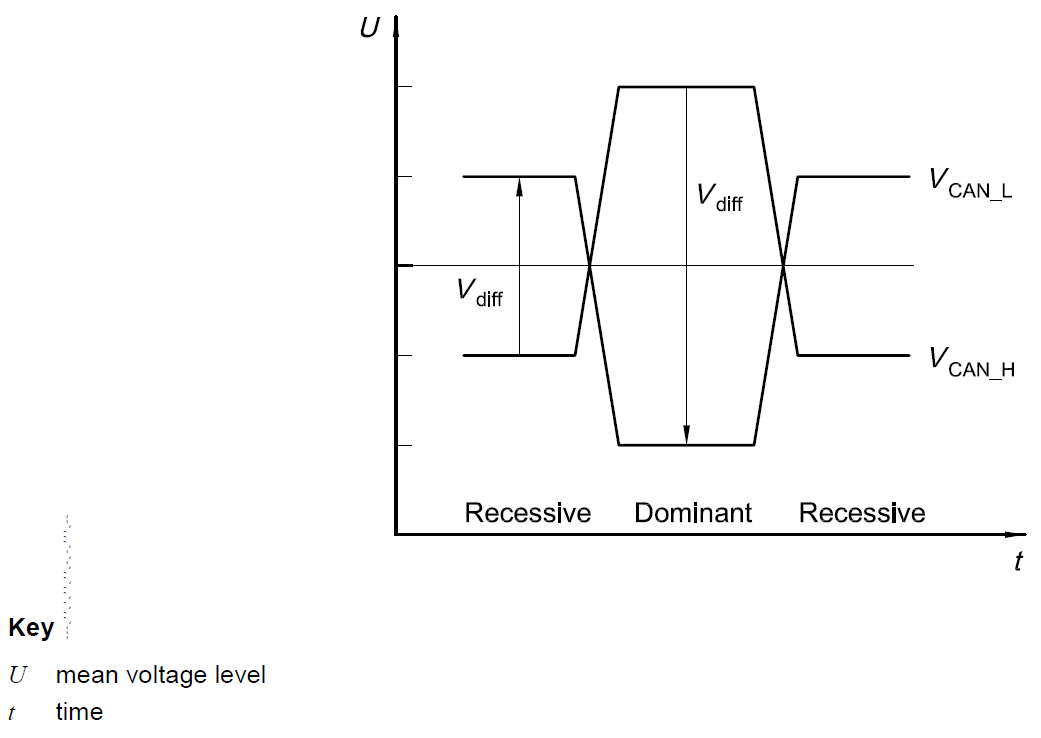
Ц

C

*CAN* је мрежа са могућношћу да има више власника (енгл. *master*). Све електронске управљачке јединице се међусобно повезују на магистралу са 2 жице (једна се зове *CAN-H*, друга *CAN-L*). Ове 2 жице су упрадена парица (оклопљена или неоклопљена) и потребно је омеђити их отпорником са номиналном карактеристичном импеданцом од 120 Ω у случају брзе *CAN* магистрале, док у случају споре *CAN* магистрале се користи отпорник од бар 100 Ω (отпорност се индивидуално израчунава).

Логичка стања се процењују на основу разлике напона. Напонски нивои зависе од конкретног сучеља :

* За брзу *CAN* магистралу (*ISO* 11898-2) [4] - Слика 2.1 Брза *CAN* мрежа :
  + Логичка 1 представља разлику од 0V (у теорији) између *CAN-H* и *CAN-L* жице, у пракси се та разлика креће од 0.5V. Напон на *CAN-H* жици се креће ка 5V док на *CAN-L* жици се креће ка 0V. Логичка 1 представља рецесивни бит (високо стање).
  + Логичка 0 представља разлику од 2V (мора биту у распону између 1.5V и 3.5V, разлика напона иде чак и преко 0.9V уколико је напајање +/-12V) између *CAN-H* и *CAN-L* жице. Напон *CAN-H* жице иде ка 5V док напон *CAN-L* жице иде ка 0V. Логичка 0 представља доминантни бит (ниско стање). Представљање доминатног бита нулом се ради да би електронске контрлне јединице са најнижим идентификационим бројем имали највећи приоритет (Слика 2.2 ).
* За спору *CAN* магистралу (*ISO* 11898-3):
  + Логичка 1 представља разлику од бар 0.6V између *CAN-H* и *CAN-L* жице. Напон *CAN-H* жице иде ка 5V, док напон жице *CAN-L* иде при преносу логичке 0. Она представља рецесивни бит.
  + Логичка 0 представља разлику од бар 2.3V између *CAN-H* и *CAN-L* жице. Напон *CAN-H* жице иде ка 5V, док напон жице *CAN-L* иде при преносу логичке 0. Она представља доминантни бит.

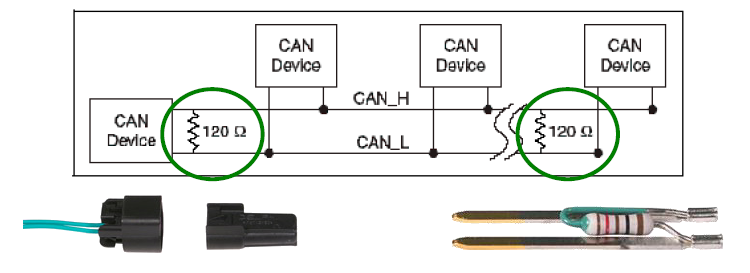


Слика . Доминантни и рецесивни бит *CAN* мреже

Слика 2.2 Физичка репрезентација бита у брзој CAN мрежи,ISO 11898-2

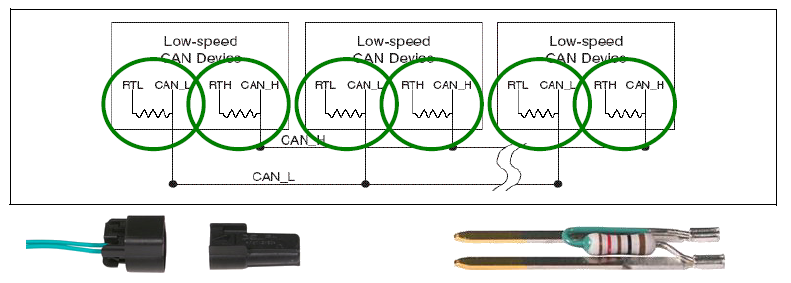
Разлике у ограђивању брзе и споре CAN магистрале су следеће:

* За брзу *CAN* магистралу је потребно да се оба краја две линије (*CAN-H* и *CAN-L*) ограде, док је средина слободна. Начин повезивања за овај тип магистале је илустрован на слици 2.3.
* За спору *CAN* магистралу нема потребе за ограђивањем крајева линије. Омеђавају се уређаји и свака линија посебно: *RTH-CAN-H* и *RTH-CAN-L*. Начин повезивања за овај тип магистрале је илустрован на слици 2.4.

****

Слика . Омеђавање линија за брзу *CAN* линију

Слика 2.3 Омеђавање линије за брзу CAN линију

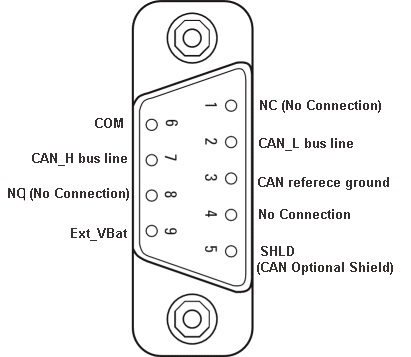


Слика . Омеђавање линија за спору *CAN* линију

### Електрична својства, каблови и утичнице

Брзина транзиције је већа када се деси промена из рецесивног у доминантни бит, без обзира да ли је у питању брза или спора *CAN* мрежа. Брзина транзиције из доминантног у рецесивни бит зависи од дужине саме *CAN* мреже и капацитивности коришћене жице. Спецификације захтевају да магистрала буде унутар минималног и максималног заједничког напона, али не дефинишу саме вредности тих граница.

Најчешће се за каблове користе оклољене упредене парице. Користи се један кабел, осим у случају када постоји потреба за додатним напајањем. Сам *CAN* не одређује врсту физичког медијума међутим, типично се користе деветопинске *D-sub* и петопинске *mini style* утичице. Пример утичнице која се најчешће користи у аутомобилској индустрији са *CAN* протоколом (*DE-9* тип *D-sub* утичнице) је приказан нана слици 2.5 .



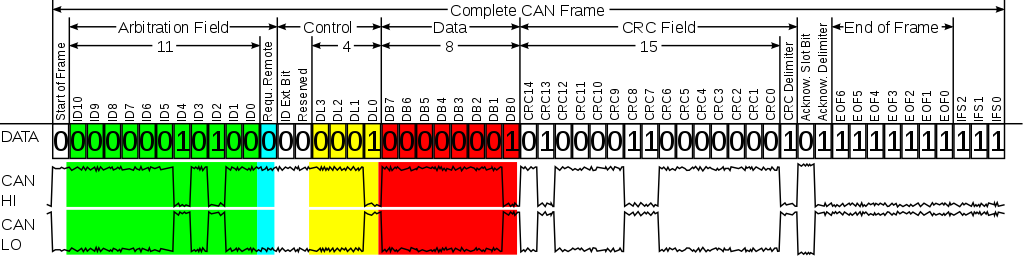
Слика . Утичница *DE-9*

### Порука, формат оквира и пренос података

Уређаји повезани на *CAN* мрежу комуницирају слањем порука. Пренос порука је контролисан са 4 типа оквира:

* Оквир за податке (енгл. *Data frame*) носи податке од предајника до пријемника.
* Оквир за даљинско управљање (енгл. *Remote frame*) се шаље на магистралу као захтев за оквиром за податке са одређеним идентификатором.
* Оквир за грешке (енгл. *Error frame*) се шаље од било ког чвора уређаја који детектује грешку на магистрали.
* Оквир за пренатрпаност (енгл. *Overload frame*) се користи да омогући кашњење између 2 послата оквира за податке или оквира за грешке.

Илустративни пример *CAN 2.0 A* без уметања бита је приказан нана .



Слика . Илустративни пример *CAN 2.0 A* оквира без уметања бита и његова корелација са напоном на *CAN-H* и *CAN-L*

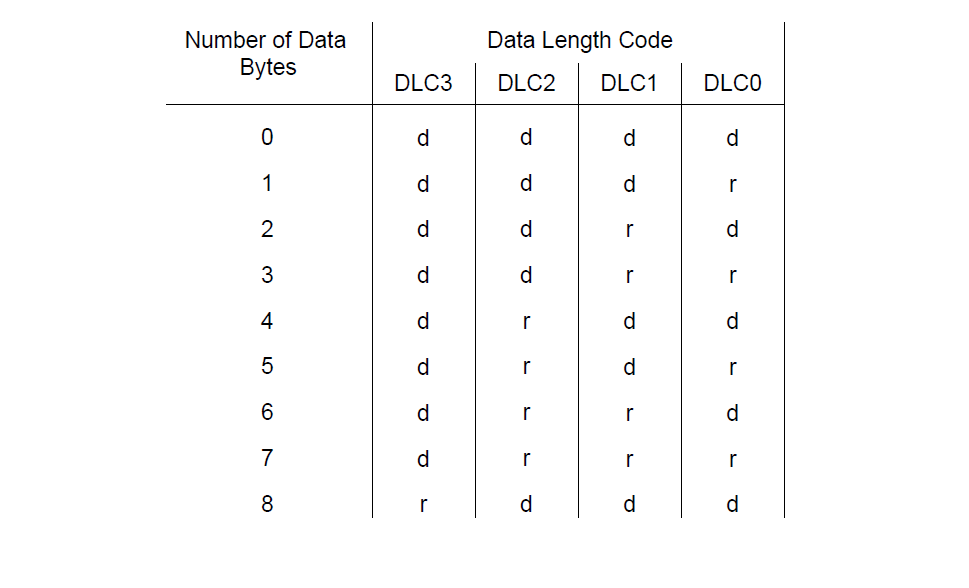
Оквир за податке се састоји од 7 поља (разлике између *CAN 2.0* *A* и *CAN 2.0 B* ћемо посебно навести где је потребно, иначе ћемо подразумевати одређена поља имају исту структуру):

* Почетак оквира (енгл. *Start of frame*) означава почетак поља за оквир за податке или оквира за даљинско управљање. Састоји се од једног доминантног бита (логичка 0). Уређај на магистрали може да почне да шаље када је магистрала у мирном режиму рада (енгл. *Bus Idle*). Сви уређаји морају да се синхронизују са водећом ивицом која је проузрокована од стране овог поља од уређаја који је први започео пренос.
* Поље за арбитражу (енгл. *Arbitration field*). За ово поље постоје разлике између *CAN 2.0 A* и *CAN 2.0 B*:
  + Ово поље се састоји од идентификатора и *RTR* бита у случају *CAN 2.0 A* при чему идентификатор се састоји од 11 бита, ови бити се гледају у *MSB* редоследу при чему свих 7 (дакле од *ID-10* до *ID-4*) најзначајних бита не смеју бити рецесивни. *RTR* бит мора да буде доминантан када се шаље оквир за податке, иначе када се шаље оквир за даљинско управљање онда је тај бит рецесиван.
  + У случају *CAN 2.0 B* идентификатор се састоји од 29 бита, подељен на 2 дела: идентификатор А (11 бита) и идентификатор Б(18 бита). Између ова 2 дела продуженог идентификатора се налази *SRR* (1 рецесиван бит) и *IDE* (1 рецесиван бит).
* Контролно поље (енгл. *Control field*) . За ово поље такође постоје разлике између 2 подврсте протокола:
  + Код бржег типа протокола, контролно поље се састоји од 6 бита. Оно укључује 4 бита који означавају дужину послатих података (енгл. *Data length code*) у бајтима и 2 бита који су резервисани за будућа проширења протокола, *IDE* бит(мора бити доминантан) и резервисани *r0* бит (може бити или доминантан или рецесиван).
  + Код споријег типа протокола ово поље се такође састоји од 6 бита, при чему прво стижу 2 резервисана бита – *r0*, *r1* (могу бити или доминанти или рецесивни). Након њих у оквиру се налазе 14 бита који означавају дужину послатих података и фунцкионишу на исти начин као у бржем протоколу.

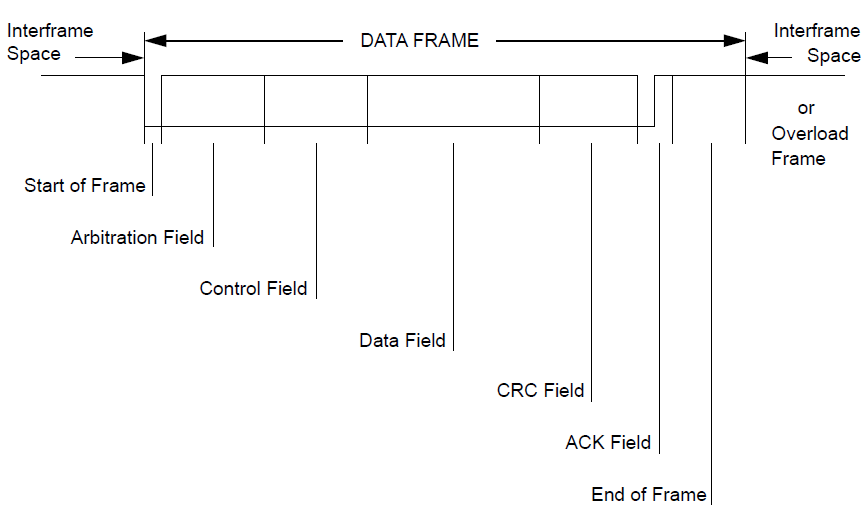
Илустративно објашњење како фунционише део котнролног поља које означава дужину података је приказано на при чему су са *d* обележени доминантни бити док су са *r* обележени рецесивни бити.

* Поље за податке (енгл. *Data field*) се састоји од 0 до 8 бајта, при чему 1 бајт има 8 бита. Подаци се шаљу у MSB редоследу.
* Поље за цикличну проверу редудансе (енгл. *CRC field*) се састоји од 15 бита који заправо представљају израчунату цикличну проверу редудансе и делимитера који заузима 1 бит, при чему је тај бит рецесиван.
* Поље за потврду о пријему (енгл. *ACK field*) се састоји од *ACK slot*, *ACK* делимитера, сваки заузима по 1 бит, при чему пошиљаоц шаље рецесивни бит у *ACK slot*, а примаоц поруке може да нареди слање доминантног бита. *ACK* делимитер „раздваја“ ово поље са крајем оквира и оно садржи 1 рецесиван бит.
* Крај оквира (*End of frame*) се састоји од 7 рецесивних бита.

Илустративни приказ оквира за податке је приказан на .



Слика . Илустративно објашњење како фунционише *DLC*



Слика . Илустративни приказ *LIN* оквира за подаке

Разлика у оквирима две врсте CAN протокола је приказан нау табели 1 .

|  |  |  |  |
| --- | --- | --- | --- |
| *CAN 2.0 А* (једанаестобитни идентификатор) | | *CAN 2.0 B* (двадесетдеветобитни идентификатор) | |
| Назив поља | Дужина поља | Назив поља | Дужина поља |
| Почетак оквира | 1 бит | Почетак оквира | 1 бит |
| Идентификатор | 11 bita | Идентификатор А | 11 бита |
| *RTR* | 1 бит | *SRR* | 1 бит |
| *IDE* | 1 бита | *IDE* | 1 бит |
| Резервисан бит | 1 бита | Идентификатор Б | 18 бита |
| *DLC* | 4 бита | *RTR* | 1 бит |
| Подаци | 0-64 бита | Резервисани бити | 2 бита |
| *CRC* | 15 бита | *DLC* | 4 бита |
| *CRC* делимитер | 1 бит | Подаци | 0-64 бита |
| *ACK* | 1 бит | *CRC* | 15 бита |
| *ACK* делимитер | 1 бит | *CRC* делимитер | 1 бит |
| *EOF* | 1 бит | *ACK* | 1 бит |
|  |  | *ACK* делимитер | 1 бит |
|  |  | *EOF* | 1 бит |

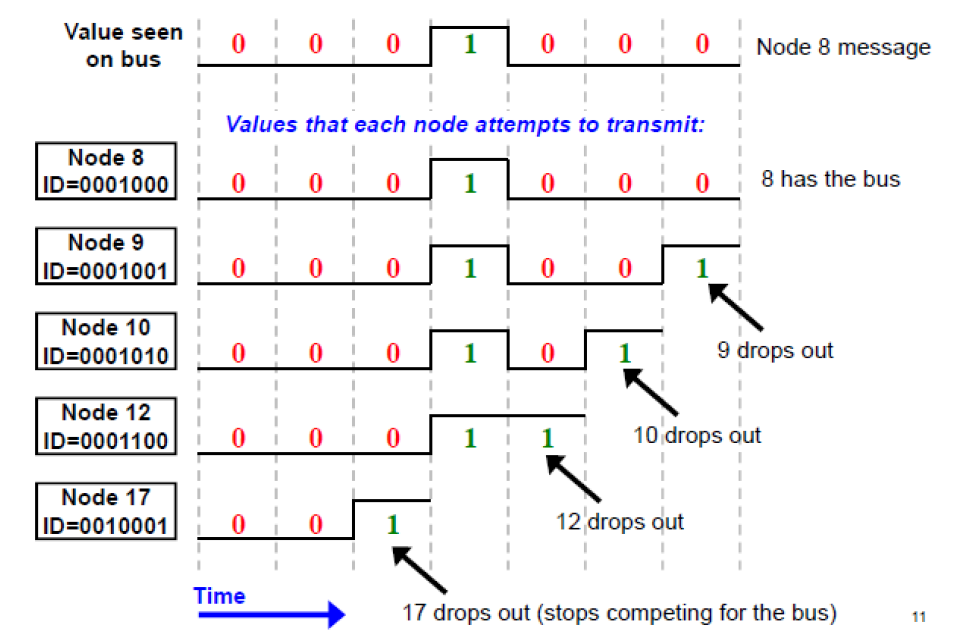
Табела Разлика између *CAN 2.0 A* и *CAN 2.0 B*

* Поље за почетак оквира
* Поље за арбитражу
* Контролно поље
* Поље за податке
* Поље за цикличну проверу редудансе
* Поље за потврду о пријему
* Крај оквира

### Арбитража

Када год је магистрала слободна, било која електронска управљачка јединица може да почне са слањем поруке. У случају да 2 или више ових управљачких јединица почне са слањем порука у исто време, настаје конфликт који се решава механизмом арбитраже за појединачне бите коришћењем идентификатора. Овај механизам обезбеђује да се не губи на времемену, а такође спречава губитак података. Ако оквир за пренос података и оквир за даљинско управљање имају исти идентификатор и ако су иницијализовани у исто време, предност има оквир за слање података.

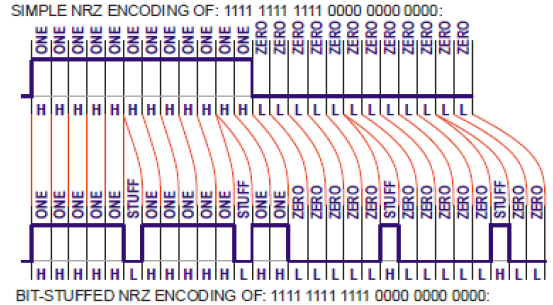
Током арбитраже, сваки пошиљаоц пореди на ком нивоу се налази бит идентификатора који је он послао, наспрам нивоа бита идентификатора који се налази на магистрали. Ако су ови бити на истим нивоима једнаки, пошиљаоц може да настави са слањем. Када је рецесивни бит на истом нивоу послат и доминантни бит на истом нивоу се налази на магистрали, тада је пошиљаоц „изгубио“ арбитражу и мора да се повуче, при чему се прекида пренос преосталих бита у оквиру. Односно, побеђују уређаји са најмањим идентификатором. Идентификатор се шаље у *MSB* формату, да би се брже уочила арбитража и да би медијум био ефикаснији за слање. Илустративни пример арбитраже је приказан нана слици 2.9 .



Слика . Пример арбитраже

### Убацивање бита, препознавање грешака, сигурност

Убацивање бита се врши на следећи начин: након 5 идентичних бита се убацује супротан бит, ово побољшава усклађивање и додати бити не мењају податке пошто је убацивање урађено хардверски. Оно генерално није потребно, пошто врло ретко се појављују истоветне вредности на магистрали. Проблем са убацивањем бита је зато што инверотовани бити се тешко могу препознати, али и зато што могу да се јаве такозвани „лавински“ бити (нпр. 5 рецисних бита које прати 5 доминантих бита) који могу да збуне механизам за препознавање убачених бита. Понекад цичклична провера редудансе може да помогне али уз ограничења. Једноставан пример убацивања бита је приказан на слици 2.10 .



Слика . Пример убацивања бита

Статистичка учестаност грешака *CAN* протокола зависи од укупног броја уређаја, физичког ожичења и распореда, спољашњих електромагнетних сметњи. Учестаност непрепознатих грешака је 1 у 1000 година, и то је један од разлога зашто је ово најкоришћенији протокол у аутомобилској индустрији која захтева дуготрајну поузданост.

Ток препознавања грешака између пошиљаоца (*Tx*) и примаоца (*Rx*) је независан од филтрирања и маскирања на пријему и се састоји из 5 механизама:

1. Надгледање појединачних бита (*Tx*) - препознавање локалних и глобалних грешака код пошиљаоца, упоређује се стање на магистрали са послатим битом,

не примењује се на поље за арбитражу .

1. Провера структуре – типична поља: *CRC* делимитер, *ACK* делимитер и *EOF* –увек рецесивни бити.
2. Провера кодовања (уметање бита) - пријемник проверава ток бита, на сваких 5 узастопних истоветних бита мора да дође убачена промена,

укључује све бите од *SOF* do краја *CRC*.

1. Провера потврде (*Tx*) је обавеза пошиљаоца -

потребно бар да пристигне једна потврда, пошиљалац подеси рецесивни бит, потребно да пријем обори са доминантним.

1. Циклична провера редудансе - пошиљалац рачуна пре слања, пријемник проверава подударање при пријему.

Циклична провера редудансе се рачуна на следећи начин:

полином, чији су коефицијенти представљени комбинацијом претходних поља (при чему је претходно урађено избацивање бита) и за 15 најнижих коефицијената, са 0. Овај полином се потом дели (коефицијенти су израчунати модулом двојке) са генератором-полиномом :

*x15 + x14 + x10 + x8 + x7 + x4 + x3 + 1*

Oстатак овог дељења представља петнаестобитну вредност *CRC* секвенце која се шаље на магистралу. Да би се применила ова функција, 15 – битни померачки регистар *CRC\_RG* (14:0) се може користити. Псеудо-код ове функције се описује са:

CRC\_RG = 0; //иницијализација померачког регистра

REPEAT

CRCNXT = NXTBIT EXOR CRC\_RG(14); //NXTBIT означава следећи бит у низу биза

CRC\_RG(14:1) = CRC\_RG(13:0); //померај у лево за једно место

CRC\_RG(0) = 0;

IF CRCNXT THEN

CRC\_RG(14:0) = CRC\_RG(14:0) EXOR (4599hex);

ENDIF

UNTIL (CRC SEQUENCE starts or there is an ERROR condition)

Након слања/пријема последњег бита од поља за податке, *CRC\_RG* садржи *CRC* секвенцу.

Сам протокол не подржава енкрипцију у стандардној имплементацији, у већини примена, од уређаја који користе *CAN* се очекује да имају своје сигурносне механизме. Ако ово није урађено, уређаји су подобни раличитим врстама малициозних напада, нпр. ако неко пошаље своје поруке на магуистралу отворен је приступ за манипулацију са уређајем. Постоје шифровани системи за одређене фунцкионалности које су од критичног значаја за сигурност, као што су модификовање системског софтвера, програмерски кључеви или контролисање кочница. Међутим, ови системи нису универзално имплементирани.

## *LIN*

LIN [5] је серијски мрежни протокол, развијен од стране LIN конзорцијума (5 произвођача аутомобила - BMW, Volkswagen Group, Audi Group, Volvo Cars, Mercedes-Benz; 1 добављач полупроводиника – Motorola; 1 добављач алата - VCT) основаног 1998. године, док је прва, у потпуности имплементирана верзија LIN спецификације (LIN 1.3) објављена 2002. године. Иницијално је развијена као CAN подмрежа са циљем да се смањи оптерећење, међутим, данас има широку употребу у раличитим индустријама: у аутомобилима, медицинској опреми, белој техници...

Најважније одлике *LIN* прокола су:

* Брзина до 20*kb/s* на дужинама до 40 метара.
* Руководилац контролише медијум, због тога постоји гаранција кашњења и нема судара. Усклађивање такта је одговорност зависних уређаја (због тога нема потребе за кварцним или керамичким резонаторима).
* Варијабилност у дужини послатих података (2,4 или 8 бајта, при чему је 1 бајт 8 бита).
* Флексибилност у конфигурацији и динамичка топологија.
* Величина мреже је углавном до 12 чворова (могуће је и до 64, али уз низак проток).
* Детекција дефектних чворова, детекција грешке и контролни збир података.
* Оперативни напон од 12V.
* Ниска цена, силиконска имплементација заснована на *UART* стандарду.
* Предности *LIN* су: лак за коришћење, велика доступност компоненти, јефтинија него *CAN* и друге магистрале за комуникацију, возила су поузданија, продужвање магистрале је лако за имплементацију, не постоје трошкови за лиценцирање.

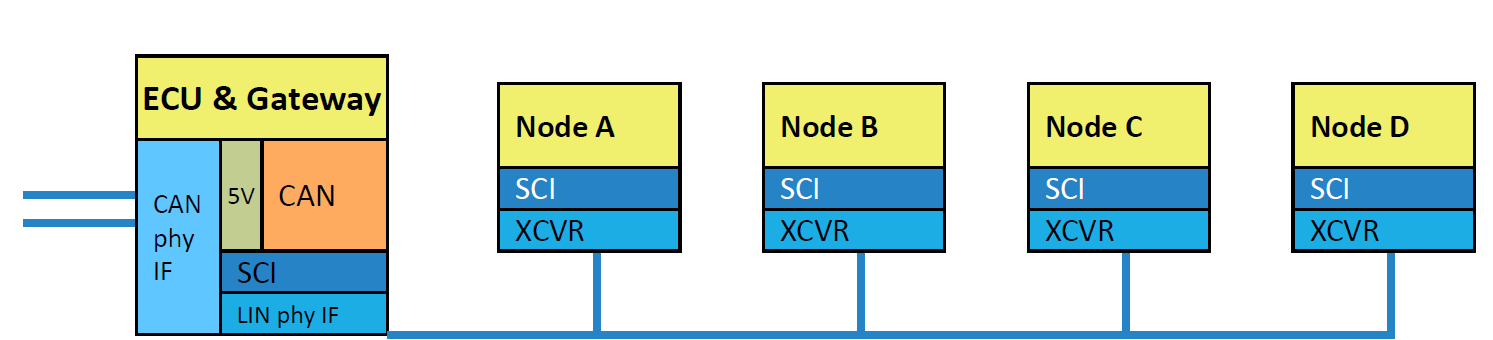
Постоји више верзија *LIN* протокола, две су нам релевантне надаље – 1.3 и 2.0 и неке од унапређења новије верзије су :

* Низови бајтова су подржани, тако да омогућавају дужину сигнала до 8 бајта.
* Групе сигнала се бришу.
* Проширени контролни збир (који укључује и заштићени идентификатор) је унапређење у односу а класични контролни збир код *LIN* *1*.*3*.
* Спорадични оквири су дефинисани.
* Управљање мрежом је мерено у секундама, не у битима.
* Управљање статусом је поједностављено и обавештавање мрежи и апликацијама је стандардизовано.

### Топологија и физички слој

*LIN* [6] је мрежа са дифузним емитовањем која се састоји од најчешће 16 мрежних чворова (1 руководилац и до 15 подређених). Све поруке су иницијализоване од стране руководиоца са највише једним подређеним који одговара на тренутно послату поруку.

Руководилац такође може да се понаша као подређени тако што одговара на своје поруке. Пошто је целокупна комуникација иницијализована од стране руководиоца, није потребна имплементација детекције судара. Уређаји повезани на *LIN* мрежу су углавном микроконтролери, али могу бити имплементирани у специјализованом хардверу или *ASIC* у намери да се додатно смање трошкови, површина или потребна стурја. Једна типична *LIN* мрежа је приказана на слици 2.11 .



Слика . Топологија типичне *LIN* мреже

Физички слој се представља као коло са отвореним колектором, магистала је терминирана руководиоцом (1 kOhm). Напон је између 7V и 18V на прикључцима. Постоје строги захтеви за симетријом и нагибима ивица, а мрежа је осетљива због велике толеранције осцилатора. Радни циклус има минимум од 39.6% а максимум од 58.1%.

### *LIN* оквир

Време које је потребно да се пренесе *LIN* оквир на магистралу је сума свих времена за које је потребно да се пренесе сваки бајт, времена за одговор и времена између 2 оквира. Време између 2 бајта је период између краја једног бита и времена почетка наредног бита.

Структура оквира се састоји од следећих делова:

* Заглавље – унутар кога се налази:
  + Поље за почетак (енгл. *Break*) се користи за сигнализацију почетка оквира. Оно се увек генерише од стране главног, управљачког чвора (енгл. *master node*) и мора да има величину од бар 13 доминантних бита, укључујући почетни бит и након њих, делимитер за почетак поља.
  + Поље за синхронизацију (енгл. *Synch*) увек има вредност која у хексадецималном формату износи 0x55, да би уређаји на мрежи увек могли да препознају ово поље. Уређаји којима се управља (енгл. slave) увек имају могућност да деткују секвенцу поља за почетак и овог поља, чак иако очекују поље за податке.
  + Заштићени идентификатор (енгл. *Protected identifier*) се састоји од 5 под-поља : идентификатор и парност идентификатора. Вредност од 0. до 5. бита означава идентификатор док од 6. до 7. бита је та вредност парност идентификатора.

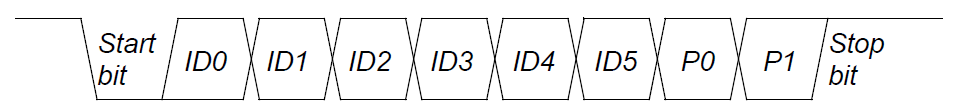
Вредности идентификатора имају и функцију одређивања примене послате *LIN* поруке па тако:

* + - 1. Вредности од 0 до 59 (*0x3b* у хексадецималном облику) се користе за пренос података.
      2. Вредности између 60 (*0x3c* у хексадецималном облику) и 61 (0x3d у хексадецималном облику) се користе за слање дијагностичких података.
      3. Вредност 62 (*0x3e* у хексадецималном облику) је резервисана са употребе дефинисане од стране корисника.
      4. 63 (*0x3f* у хексадецималном облику) је резервисано за будућа побољшања.

Парност бита се израчунава на следећи начин (при чему су вредности у једначинама објашњене у илустрацији мапирања овог поља на слици 2.12 ) :

P0 = ID0 ⊕ ID1 ⊕ ID2 ⊕ ID4 (1)

P1 = ¬ (ID1 ⊕ ID3 ⊕ ID4 ⊕ ID5)



Слика . Илустрација мапирања идентификатора и парности унутар поља за заштићени идентификатор

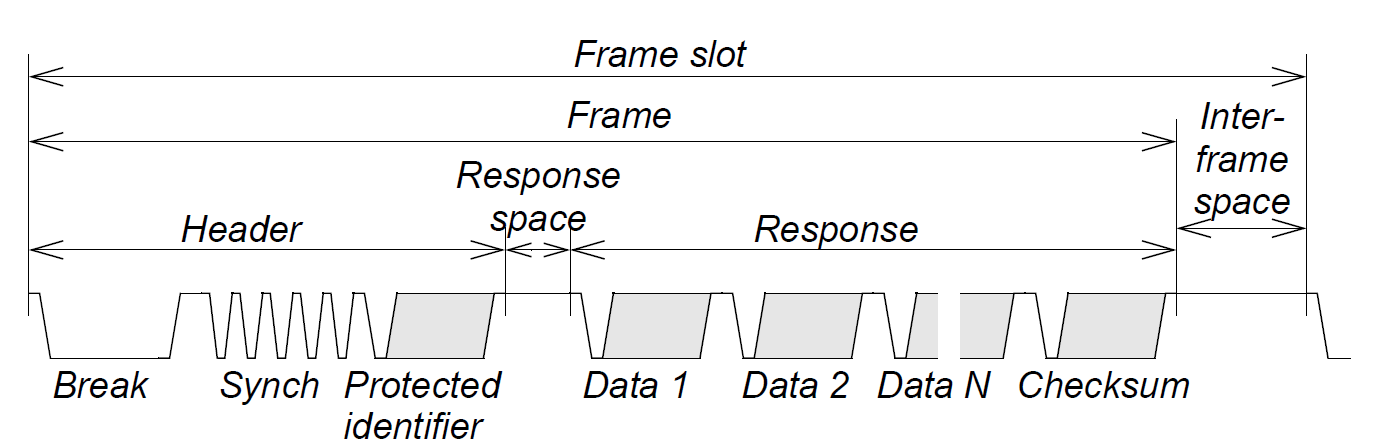
* Размак за одговор (енгл. *Response space*).
* Одговор (енгл. *Response*) – унутар кога се налазе подаци и поље за проверу контролног збира. Поље за податке се састоји од 1 до 8 бајта. Након ових поља за податке долази поље за проверу контролног збира. Оно садржи осмобитну инвертовану суму са преносом за све податке унутар поруке (у случају *LIN 1.3* – и тада се оно назива класично поље за проверу контролног збира, енгл. *classic checksum*) или инвертовану суму са преносом за све податке и заштићени идентификатор унутар поруке (такозвано побољшано поље за проверу контролног збира, енгл. *enhanced checksum*, користи се у верзији 2.0). Идентификатори са вредношћу 60 и 63 ће увек користити класичну контролну суму.

Врсте одговора су :

1. Безусловни оквир (енгл. *Unconditional Frames*) (идентфикатор 0-59) – тачно један дефинисани подређени шаље одговор, сваки захтев се шаље у посебном пакету. Потребно је обезбедити довољно велики пакет за одговор. Пакет се углавном шаље сваког циклуса распореда. Могуће је слати више пута у току истог циклуса, то зависи од апликације. Пријем одговора је доступан свим подређенима и руководиоцу.
2. Дијагностички оквир (енгл. *Diagnostic Frames*) (идентификатор 60-61). Могуће је слати сегментирано, односно више оквира заредом без заглавља. Постоје 2 врсте овог оквира:
   1. Захтев (енгл. *Master Request Frame*) (идентификатор 60) – заглавље и одговор шаље руководилац. Употребљава се кад је потребан захтев за дијагностиком или подешавањем.
   2. Одговор (енгл. *Slave Response Frame*) (идентификатор 61) – заглавље шаље руководилац а одговор одговарајући подређени. Употребљава се као одговор на захтев за дијагностиком или подешавањем.
3. Временски диригован оквир (енгл. *Event triggered frame*) (идентификатор 0-59). Служи за груписање више повремених безусловних оквира. Овим се добија на уштеди у циклусу – кратики одговори који не заузимају распоред. Последица овога је да долази до ретких појава судара, који се додатно избегавају добрим осмишљавањем мреже. Сударе решава руководилац.
4. Спорадични оквир (енгл. *Sporadic frame*) (идентификатор 0-59). Логика овог оквира је слична претходном оквиру.
5. Остали оквири (енгл. *Other frames*) (идентификатор 62-63).

* Размак између оквира.

Илустративни приказ оквира је приказан на слици 2.13 .



Слика . Пример *LIN* оквира

Слика 3.2 Пример LIN оквира

Сваки оквир чије се слање планира, алоцира одређени временски интервал на магистрали. Трајање овог временског интервала мора бити довољно дугачко да омогући пренос оквира чак и у најгорем могућем случају (због лошег квалитета подређених компоненти се стварају кашњења). Номинална вредност за пренос оквира је једнако броју послатих податак у битима. Рачунање овог временског интервала се израчунава на следећи начин:

**Тноминално\_време\_заглавља = 34 \*Tбита**

**Тноминално\_време\_одговора = 10 \* (Број\_бајта\_података + 1) \* Тбита**

**Тноминално\_време\_оквира = Тноминално\_време\_заглавља + Тноминално\_време\_одговора**

**Тбит = 1 / брзина такта**

**Тбита** представља време потребно за пренос једног бита. Максимално време оквира је још 40% у односу на **Тноминално\_време\_оквира** .Такт је до 20 kb/s.

## *SPI*

*SPI* [7] је синхрони, серијски комуникациони интерфејс који се користи за комуникацју на кратким дистанцама, првенствено у уграђеним системима. Интерфејс је осмишљен од стране Motorola средином 80 – тих година 20. века. Неке од карактеристика овог интерфејса су:

* Једноставност повезивања : са 4 жице – *SCLK* (*Serial Clock*) - такт за усклађивање преноса; *MOSI* (*Master Output Slave Input*) – преноси податке од руководиоца до подређеног; *MISO* (*Master Input Slave Output*) – преноси податке од подређеног до руководиоца ; *SS* (*Slave Select*) – посебна линија за одабир сваког подређеног), један надређени и више подређених. Надређени активира подређене ниским сигналом. И *MISO* и *MOSI* линије су активне током сваког преноса.
* Брже од *UART*-а : 250*kb/s* до 2*Mb/s*.
* Кратак домет: ~10цм на сваки чвор.
* Могућност повезивања процесора и широког спектра подржаних периферија, 2 процесора. Могуће је и повезивање више зависних подређених уређаја (један руководилац и више нанизаних подређених – енгл. *Daisy chained*).
* Групе бита података (4,8,16), непостојање стартног и стоп бита. Кодовање *NRZ*-L. Одвојене линије за пренос у оба смера (дуплекс, али је потребна подршка на чипу).
* Синхронизација – посебна линија за такт (неактивна кад се постави на 1 или 0 на дужи период). Нема потребе за усклађивањем такта.
* Читање на опадајућу или растућу ивицу.

*SPI* поседује 4 регистра :

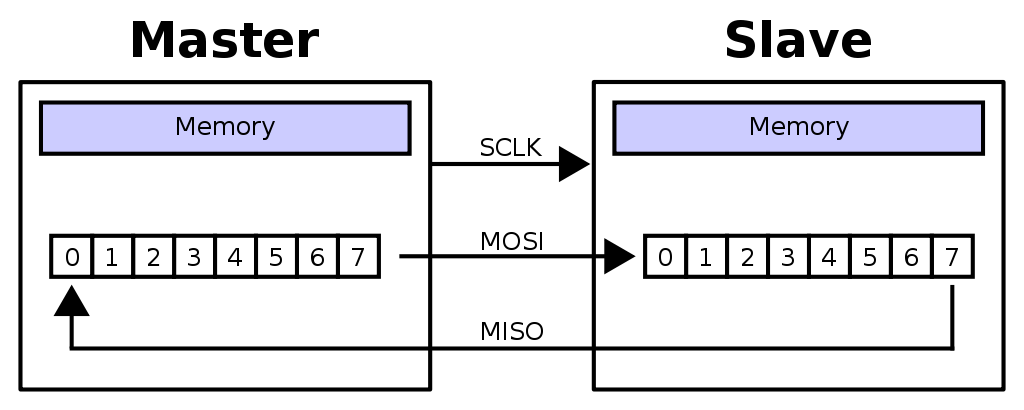
1. *SDPR* (*transferred Data read/write Register*) – смештање примо-предајних података.
2. *SPCR* (*Control Register*) – смештање контролних подешавања.
3. *SPSR* (*Status Register*) – смештање статусних информација.
4. *SPTCI* (*Transfer Complete Interrupt*) – смештање прекидачке рутине за готов пренос.

### Пренос података, фаза и поларитет

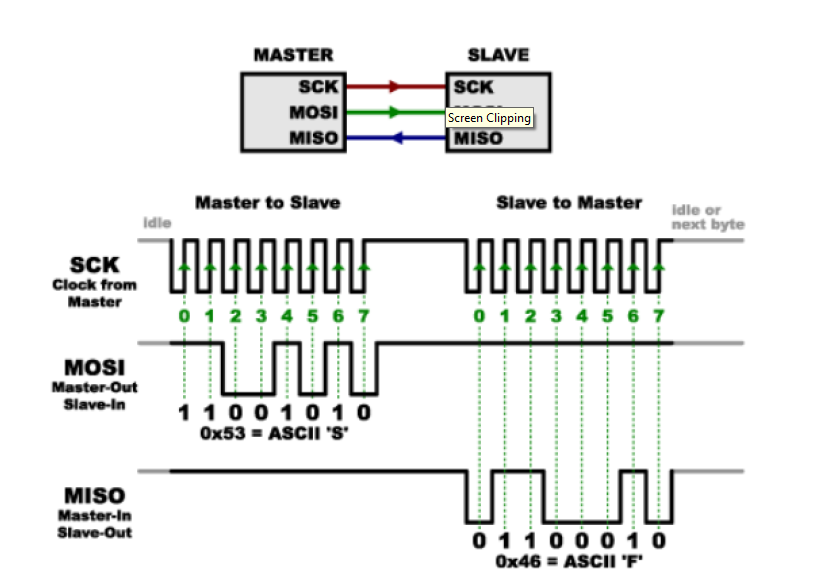
Да започне комуникацију, руководилац магистрале конфигурише радни такт, користећи фреквенцију подржану од стране подређеног уређаја, углавном до пар *MHz*. Руководилац потом бира подређени уређај са логичком 0 на *SS* линији. Ако се захтева период чекања, руководилац мора да чека тај период пре него што пошаље радни такт.

Током сваког радног такта, пуни дуплекс, односно пренос у оба смера, се деси. Руководилац шаље бит на *MOSI* линији и потом га подређени прочита, док је на *MISO* линији обрнуто. Ова процес се дешава чак и када се подаци шаљу у једном смеру.

Пренос углавном укључује 2 померачка регистра, дужине података, један се налази код руководиоца, други код подређеног. Подаци се шаљу користећи *MSB* организацију. На ивицу такта оба уређаја померају по 1 бит из регистра и њега шаљу даље свом одговарајућем уређају. На следећој ивици такта, сваки од уређаја прими бит са линије са пријем и тај бит се поставља као *LSB* померачког регистра. Након што су се сви бити померили и након што су послати, два уређаја су разменили податке. Када се трансмисија у потпуности обавила, руководилац престаје да генерише ивице такта. Ако треба да се пошаље још података, померачки регистри се опет испочетка „напуне“ битима и пренос опет може да започне. Пример повезивања и слања податак 2 уређаја на овакав начин је приказан на слици 2.14 и слици 2.15 . Трансмисија се углавном састоји од 8 бита.



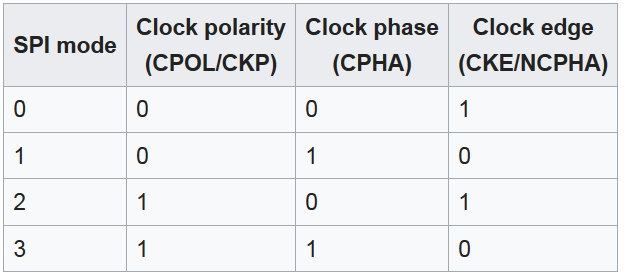
Слика . Пример повезивања *SPI* руководиоца и подређеног уређаја



Слика . Пример преноса података између *SPI* руководиоца и подређеног

Постоје 4 режима као комбинација (илустровано на слици 2.16 ):

* 2 стања фазе такта (енгл. *CPHA*)
* 2 стања поларитета такта (енгл. *CKP/CPOL*)



Слика . Режими рада *SPI*

\

Фаза (енгл. *CPHA*) дефинише значење предње и задње ивице такта. Пресликавање узорковања и прелаза на нови бит некој од ивица. Различите ивице имају различито значење. Поларитет (енгл. *CKP/CPOL*) дефинише вредност активног и стања мировања. Пресликавање 0 и 1 на активност и мировање. Мировање као почетак и референца на ивице.

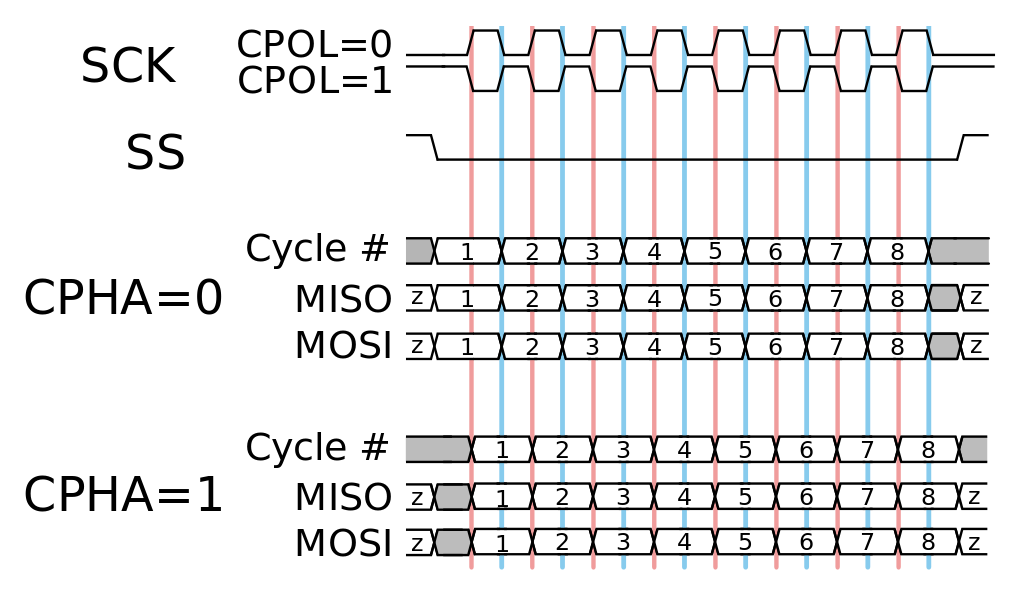
Када је *CPOL*=0 мировање је 0 а активност 1:

* За *CPHA*=0, узорковање на предњу (узлазну) ивицу (0→1) а прелаз на нови бит на задњу (силазну) (1→0).
* За *CPHA*=1, узорковање на задњу(силазну) ивицу (1→0) а прелаз на нови бит на предњу (узлазну) (0→1).

Када је *CPOL*=1 мировање је 1 а активност 0 (инверзија за *CPOL*=0) :

* За *CPHA*=0, узорковање на предњу (силазну) ивицу (1→0) а прелаз на нови бит на задњу (узлазну) (0→1).
* За *CPHA*=1, узорковање на задњу (узлазну) ивицу (0→1) а прелаз на нови бит на предњу (силазну) (1→0).

*MOSI* и *MISO* сигнали су углавном стабилни за пола циклуса до следеће промене такта. *SPI* руководилац и подређени могу да одабирају сигнале на различитим одбирцима у том полу-циклусу. Ово даје додатну флексибилност комуникационом каналу између руководиоца и подређеног. Пример фунционисања фаза и поларитета је дат на слици 2.17 .



Слика . Пример фазе и поларитета за *SPI*

## *I²C*

*I²C* [8] је синхрона магистрала са могућношћу да има више руководиоца и више подређених створена од *Philips Semiconductor* 1982. године. Има широку употребу за повезивање спорих периферија за процесоре и микроконтроле на малим дистанцама.Од октобра 2006. , не постоје трошкови лиценцирања за употребу овог протокола. *SMBus* је подскуп овог протола,створе од стране *Intel* – а 1995. године за дефинисаном стриктнијом употребом са циљем да дода робустност и интер-опребилност.

Модерни *I2C* системи инкорпорирају нека правила из *SMBus* – a , при чему је потребна само мала количина реконфигурације. Оригинално употребљиван у телевизорима, сада подржава велике количине периферија(наменски системи – *EEPROM* – ови , *Flash* и *RAM* меморија, *watchdog* тајмери, микроконтролери, персонални рачунари). *I²C* подржава више брзина комуникације :

* Стандардни режим (енгл. *Standard* *Mode*) : 100*kb/s* , оригинално од прве верзије у употреби.
* Брзи режим (енгл. *Fast Mode*) : 400*kb/s* , додат у верзији 1 овог протокола 1992. године, додато је и десетобитно адресирање, због чега се максималан број мрежих чворова повећао на 1008. Ово је прва стандардизована верзија.
* Брзи режим плус (енгл. *Fast Mode plus*) : *1Mb/s* , додат у верзији 3 2007. године.
* Режим велике брзине (енгл. *High Speed Mode*) : 3.4*Mb/s* додат у верзији 2 1998. године.
* Ултра-брзи режим (енгл. *Ultra Fast Mode*) : 5*Mb/s* додат у верзији 4 2012. године.

### Начин функционисања

*I²C* користи 2 двосмерне линије – *SDA* (*serial data*) и *SCL* (*serial clock*). Имплементација преко отвореног колектора омогућава :

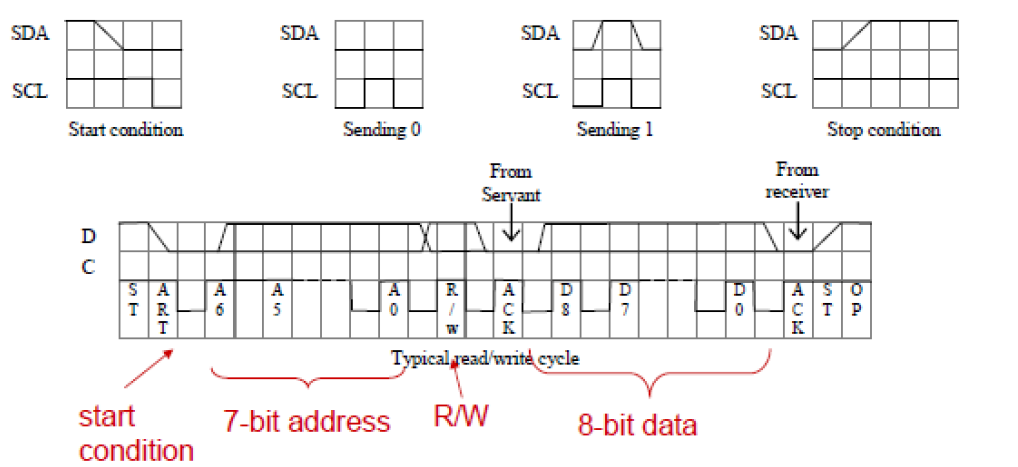
* Подршку за више уређаја
* Једноставно спајање за различите напоне
* Понашање као ожичено логичко И
* Подржава посредовање и решавање судара

Руководилац контролише, а не генерише такт и иницира комуникацију, док подређени прима такт и одговара на прозивку руководиоца, али уколико му је потребно више времена може да обори такт. Улоге се могу заменити након стоп секвенце. Сваки уређај повезан на магистралу има јединствену адресу. Формат адресе се састоји од адресног поља :

* Изворно – 7 бита + 1 најнижи бит у адреси за статус уписа или читања (1 означава читање, 0 упис).
* Проширено – 10 бита. Водећи октет се састоји из петобитне контролне секвенце (11110), 2 горња бита адресе и бит за статус уписа или читања (исти начин фунцкионисања као и за претходни тип адресе). Нижи октет представља 8 доњих бита адресе.

Максималан број уређаја на магистрали је ограничен бројем расположивих адреса и укупном капацитивношћу магистрале (максимално 400pF). Максимална дужина линије је неколико метара. Пошиљаоц поставља податке на *SDA* линију, прималац их потврди. Руководилац започиње комуникацију старт секвенцом – поставља *SDA* линију на ниску ивицу док је *SCL* линија на високој ивици. Редовне промене *SDA* се једино дешавају када је *SCL* на ниској ивици. Пренос података се завршава стоп секвенцом. Након старт секвенце се шаље адресно поље,прозвани зависни потврђује,након потврде иду подаци које такође прати потврда. Могуће је слање више података одједном. Оквир изгледа исто приликом слања и читања, једино је бит који је статус за упис читања и писања различит.

Такође могућ је и комбиновани пренос који служи за промену смера трансакције, где руководилац уместо стоп секвенце шаље продужени старт, затим шаље ново адресно поље са битом за писање или читање. Подешавање бита података се врши за време ниског нивоа такта, а узорковање бита података се врши за време високог нивоа такта. Да би се избегло лажно препознавање пожељно је радити узорковање и подешавање на ивицу такта. Временски дијаграм *I²C* је приказан нана слици 2.18 .



Слика . Временски дијаграм *I²C*

## Упоредно поређење протокола

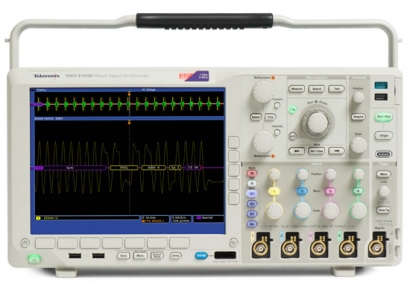
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *CAN* | *LIN* | *SPI* | *I²C* |
| Брзина | До 1 *Mb/s* | До 20 *kb/s* | До 2 *Mb/s* | До 5 *Mb/s* |
| Број линија | 2 | 1 | 4 | 2 |
| Предности | Прилагођеност за асинхрони саобраћај.  Висока пропусна моћ под малим оптерећењем.  Ефикасно коришћење пропусног опсега.  Могуће дефинисати локалне и глобалне приорите.  Константан премашај: битска арбитрација део поруке. | Проширење CAN: подмреже.  Смањење цене ожичења.  Побољшава ЕМК.  Нема потребе за арбитражом.  Без спољњег кристала.  Смањује ризик од доступности.  Детерминизам и предвидљивост . | Велика брзина за тачка на тачку преносе.  Подршка за симултани улазно/излазни ток података.  Не постоји адресирање: олакшана имплементација.  Широко подржан. | Bише брзине, дуплекс, једноставни померачки регистри. |
| Мане | Потребно доминантно битско стање: немогућност трансформаторског одвајања  немогућност окидања на време.  Брзина простирања ограничава дужину магистрале.  Неправедан приступ: високи приоритет може да окупира мрежу.  Повећана кашњења за низак приоритет: изгладњивање  Потребно филтрирање да се не би преливали спремнике. | Уређаји су подобни раличитим врстама малициозних напада. | Руковање више подређених: усложњава линије.  Немогућност комуникације између подређених.  Нема потврде пријема: такт могуће да одлута  Не постоји решавање колизија.  Не постоји контрола тока (потребно знати брзину подређених). | Ограничења за проширења: број уређаја ограничен новим линијама.  Тешко имплементирати само у софтверу.  Већи број линија. |

Као резиме претходних поглавља у табели 2 је приказан упоредни приказ протокола.

Табела Упоредно поређење протокола

## Осцилоскоп

Приликом израде овог дипломског рада користио сам *Tektronix DPO 4104B* осцилоскоп, приказан на слици 2.18 . Овај осцилоскоп има 4 независна канала. Са сондама су мерени напонски нивои који су одређени сигнали протокола имали, чије сам жице претходно повезао на протоборд. Окидачи су потом подешени за нивое измереног напона, хоризонталну и вертикалну скалу. На самом почетку ово подешавање је било мануелно, праћењем упутства за употребу овог осцилоскопа [9], потом аутоматски – комуникацијом преко *USB*-а.



Слика . *Tektronix DPO 4104B*

# Концепт решења

## Повезивање *CAN* – и мерење са осцилоскопом

За генерисање сигнала за *CAN* магистралу коришћен је *Vector VN8912A* модул и алат *CANoe* .

*CANoe* алат је напредни алат за развој, тестирање и анализу како појединачних електронских управљачких јединица, тако и комплетне мреже. Овим алатом могуће је заокружити комплетан процес од планирања мреже до тестова на системском нивоу.

Основне предности овог алата су:

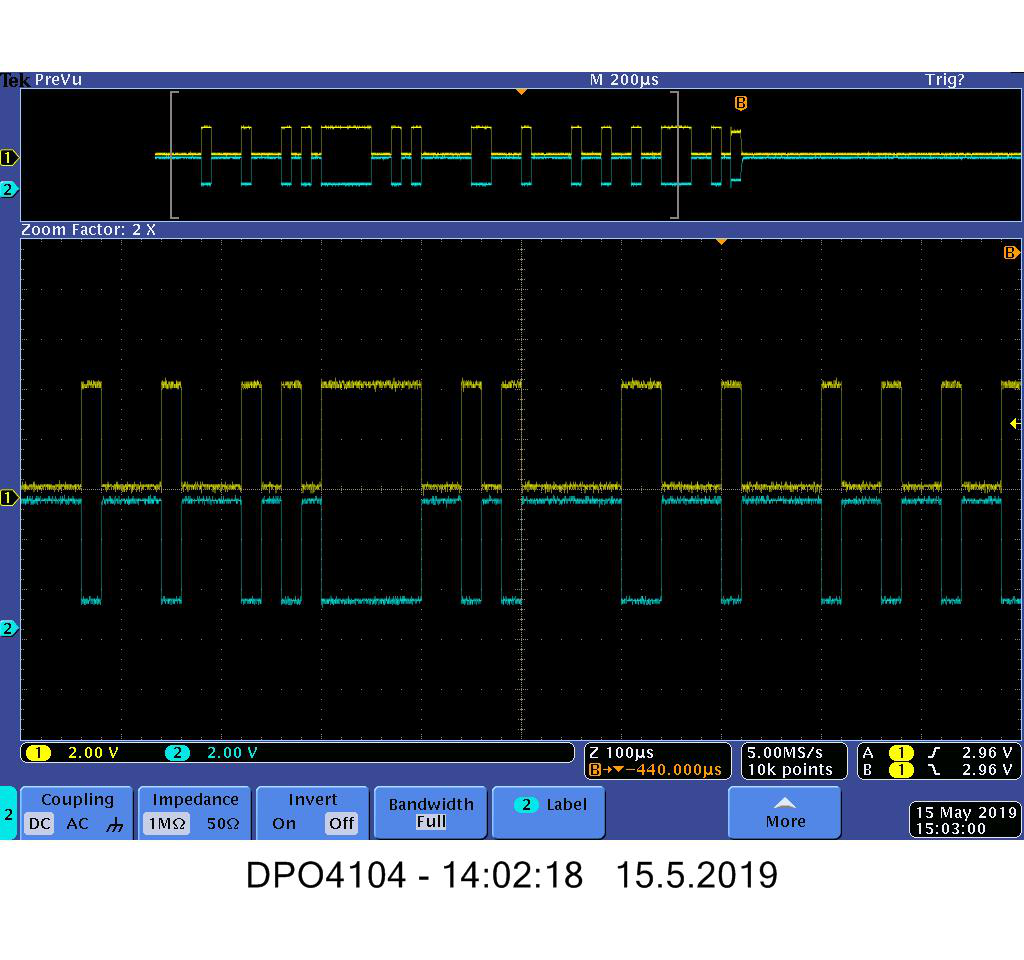
* Један заједнички алат и за тестирање и за развој.
* Лако аутоматско тестирање.
* Могућност симулирања и тестирања електронских управљачких јединица преко дијагностике.
* Откривање и исправљање грешака у раним фазама развоја софтвера.
* Интуитивно графичко окружење и евалуација резултата базирана на тексту.

Корисници *CANoe*-а могу анализирати више магистрала као и цео систем на свом столу.

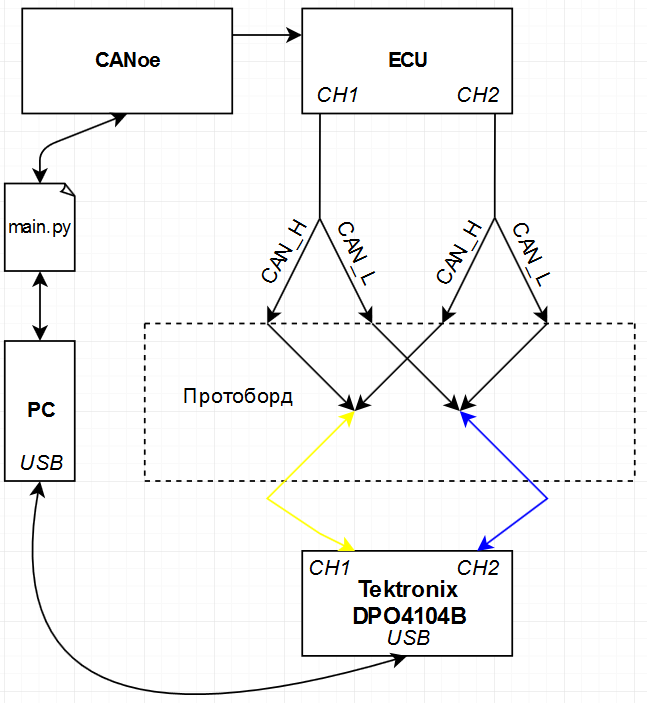
У *CANdb++ Editor* - у је креирана база података (формат .*dbc*) која описује целокупну CAN мрежу. Свака *CAN* мрежа је дефинисана са:

* мрежом
* контролоном јединицом
* променљивама окружења
* мрежним чворовима
* порукама
* сигналима

Само креирање ове базе података ће бити детаљније објашњено у програмском решењу. Након подешавања конфигурације у *CANoe* - у и укључивања креиране базе података, урађено је повезивање модула *VN8912A* и рачунар *USB* каблом, покренута је симулација док су са друге стране повезане 2 пробе осцилоскопа на *CAN-H* и *CAN-L* канал чије су жице изведене и прикључене на протоборд. Окидач за напон је подешен на 2.96 V, скала на 100 микросекунди, док је сваки од 2 канала скалиран на 2V. Екран осцилоскопа приликом снимљеног *CAN* сигнала је приказан на слици 3.1 . Дијаграм која објашњава повезивање *CAN*-а је приказан на слици 3.2 .



Слика . Приказ екрана осцилоскопа приликом снимања *CAN* сигнала

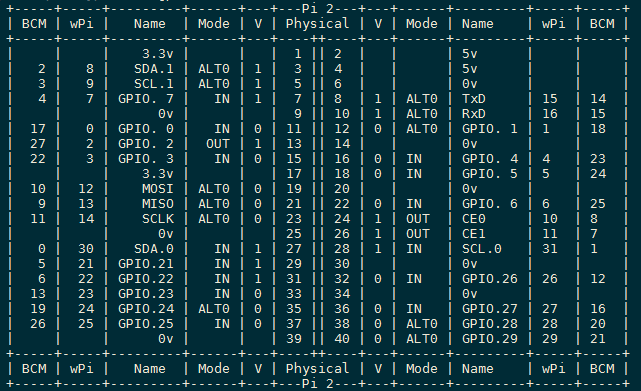


Слика . Дијаграм повезивања *CAN*

## Повезивање *LIN* – а и мерење са осцилоскопом

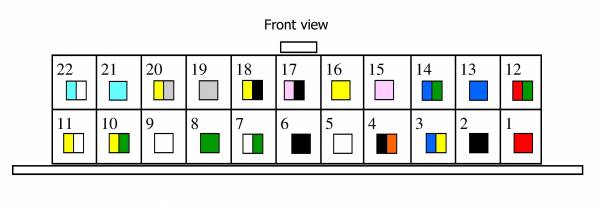
Carberry је аутомобилска плочица за проширење рачунара *Raspberry Pi*. Идеја је да се понуди спона ка електроници и комуникационим каналима у аутомобилима и самим тим омогући интеракција, али и развој апликација за крајњег корисника као што су забавно-информациони системи (медија центар, дијагностика аутомобила, логовање података, управљање флотом возила, праћење возила), алармни системи, пружање Интернета и слично. Такође *Carberry* може да послужи и у образовне сврхе да би се проучавали аутомобилске магистрале и протоколи.

На самом почетку је *Carberry* плочица повезана преко 40-пинског *GPIO* интерфејса на *Raspberry Pi 2* модел Б рачунар. Преглед пинова *Raspberry Pi 2* модел Б рачунарa (добијено куцањем команде *gpio readall*) је приказан на слици 3.3 . *Carberry* и *Raspberry Pi* комуницирају преко *GPIO* интерфејса.



Слика . *Raspberry Pi 2* модел Б,преглед пинова

Следећи корак је био повезивање жице са главне утичнице *Carberry* модула на протоборд. Илустровани приказ главне утичнице се налази на слици 3.4 , а табела са описом свих сигнала се налази у табели 3 где су црвеном бојом означени каблови за напајање и уземљење, а зеленом каблови за *LIN* (остали каблови ми нису релевантни тако да њихову улогу нисам описао). Након што су повезана 2 оваква модула на протоборд, комуникација је остварена преко серијског порта коришћењем *microUSBТоUSB* кабла. Након тога је започета комуникација са сваким од ових модула. То је урађено коришћењем команди које су стандардизоване од стране произвођача *Carberry*-ја.

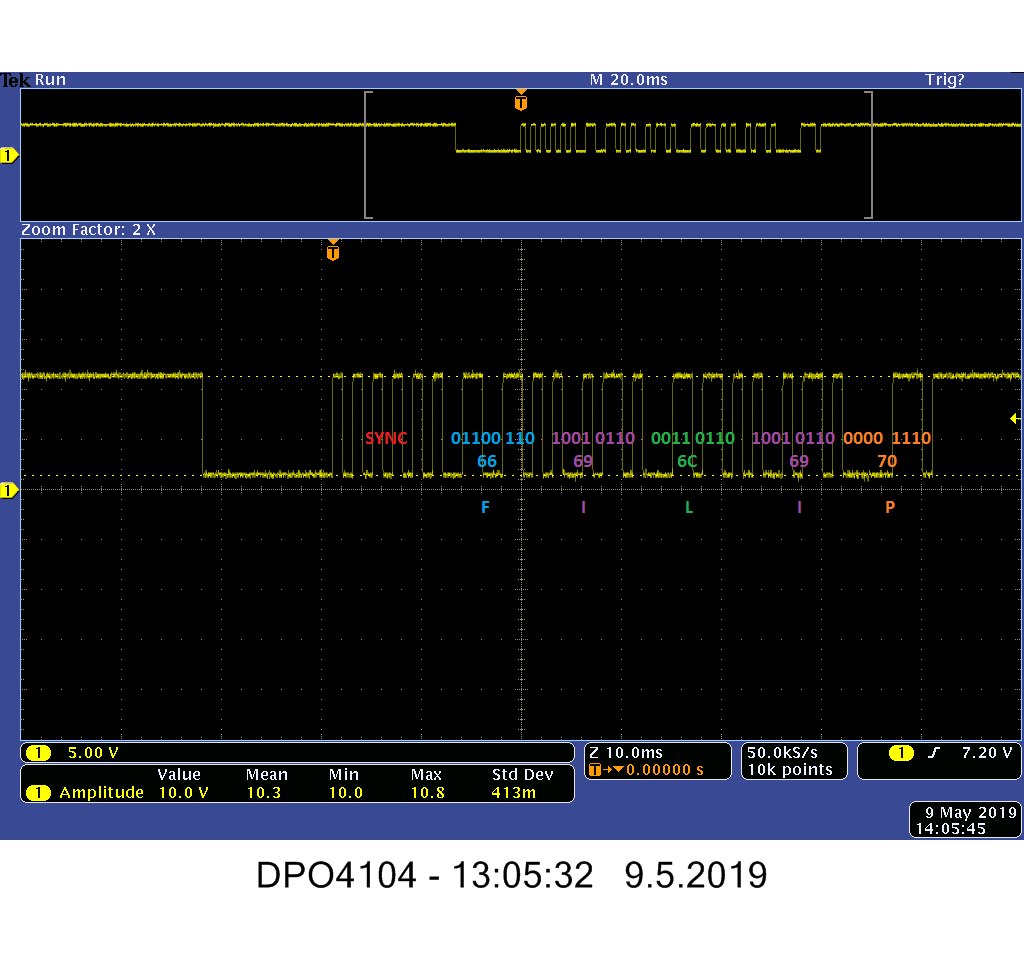


Слика . Илустровани приказ главне утичнице за *Carberry* модул

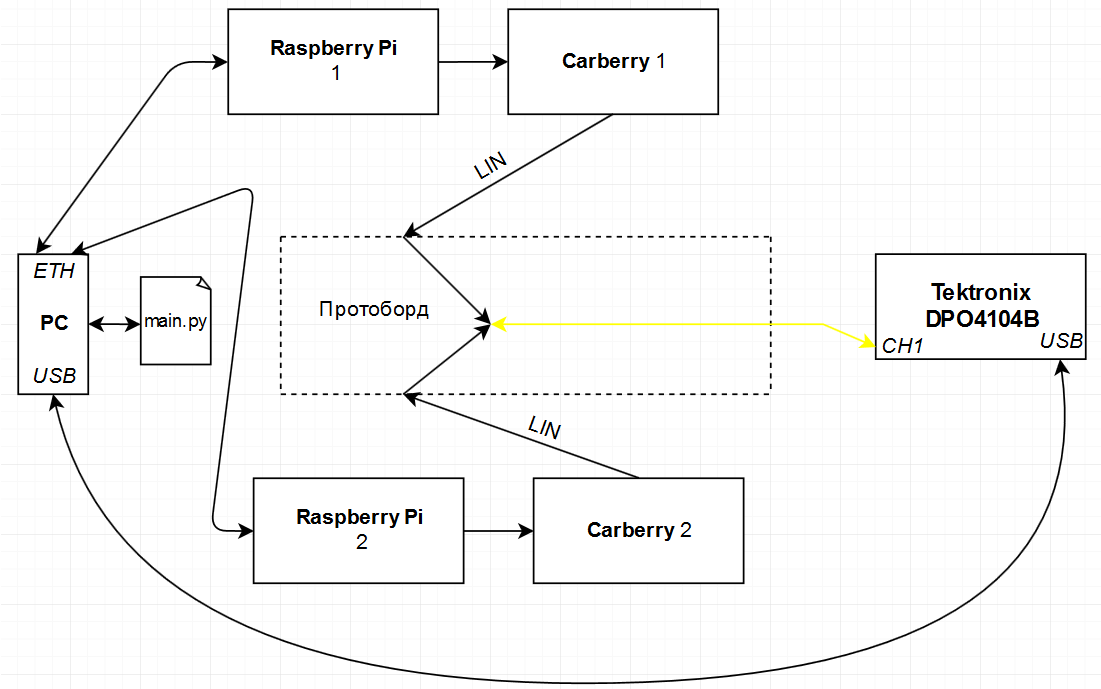
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Позиција** | **Боја жице** | **Опис** | **Позиција** | **Боја жице** | **Опис** |
| **1** | **Црвена** | **+12V Напон** | **12** | **Црвено**  **/Зелена** | **LIN** |
| **2** | **Црна** | **Узмемљење** | 13 | Плава | / |
| **3** | **Плаво**  **/Жута** | **+12V Улаз за паљење** | 14 | Плаво  /Зелена | / |
| 4 | Црно  /Наран. | / | 15 | Ружичаста | / |
| 5 | Бела | / | 16 | Жута | / |
| 6 | Црна | / | 17 | Ружичасто  /Бела | / |
| 7 | Бело  /Зелена | / | 18 | Жуто  /Црна | / |
| 8 | Зелена | / | 19 | Сива | / |
| 9 | Бела | / | 20 | Жуто  /Сива | / |
| 10 | Жуто  /Зелена | / | 21 | Светло плава | / |
| 11 | Жуто  /Бела | / | 22 | Светло-плава/  Бела | / |

Табела Опис жица главне утичнице за *Carberry* модул

Ове команде ће детаљније бити објашњене у програмском решењу. Оба уређаја се морају подесити на истој брзини комуникације. Након тога се могу слати команде за генерисање сигнала. На осцилоскопу је скала подешена на 10ms, окидач на 7.2V а скала самог сигнала на 5V. Пример сигнала који је снимљен осцилоскопом је приказан на слици 3.5 . Дијаграм повезивања за *LIN* је приказан на слици 3.6 .



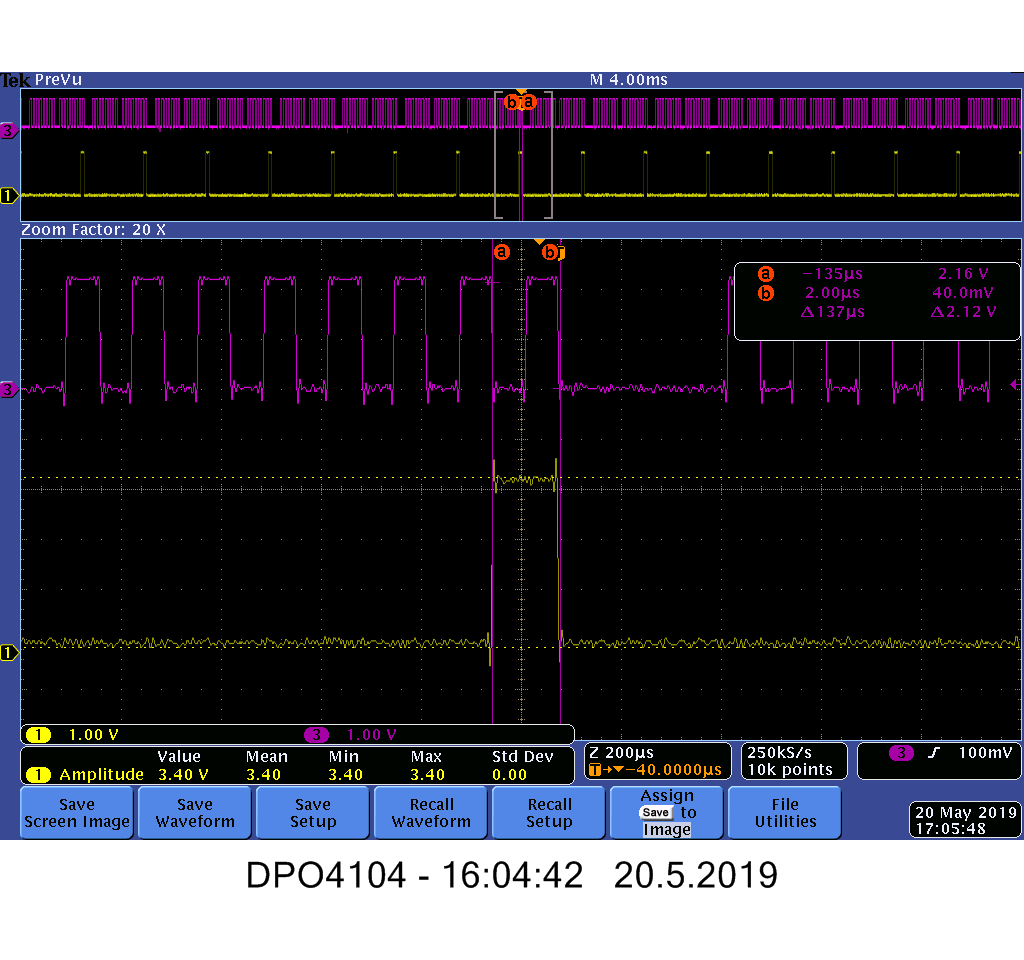
Слика . Приказ екрана осцилоскопа приликом снимања *LIN* сигнала



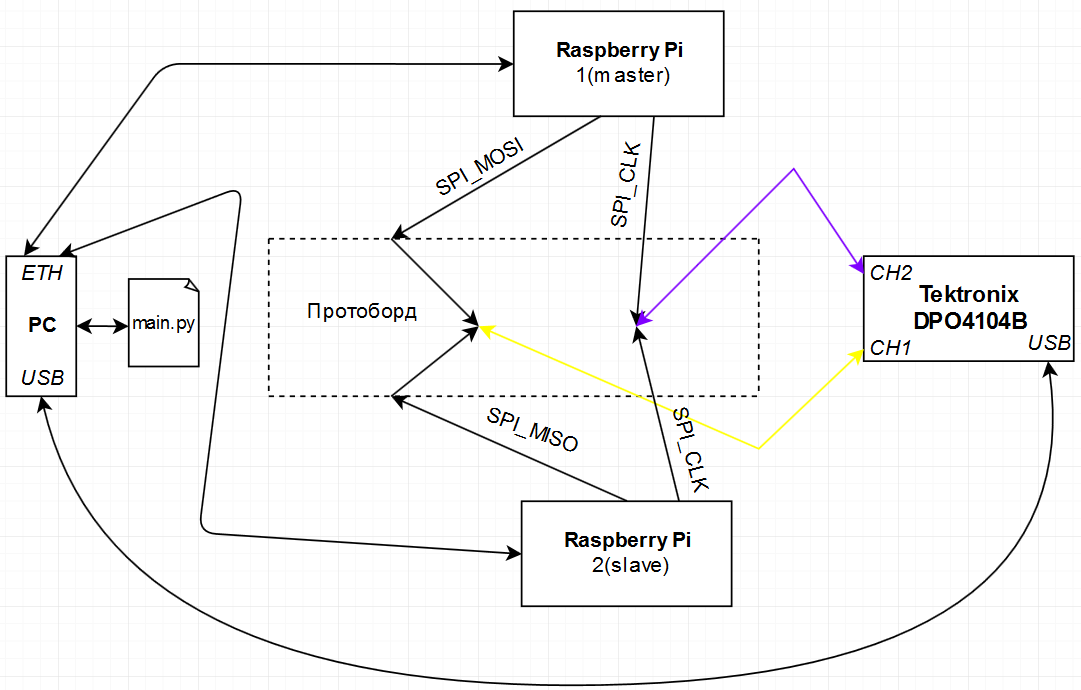
Слика . Дијаграм повезивања *LIN*-a

## Повезивање *SPI* – а и мерење са осцилоскопом

Повезивање *SPI* на протоборд се врши праћењем пинова за *GPIO* интерфејс *Raspberry* *Pi*  рачунара као на слици 3.3 , при чему су коришћени само пинови *GPIO* 9 (*SPI\_MOSI*), *GPIO* 10 (*SPI\_MISO*) и 11 (*SPI\_CLK*). Један *Raspberry Pi* треба да се понаша као надређени, док је други подређени. Међусобно на протоборду је повезана *SPI\_MOSI* линија надређеног уређаја са *SPI\_MISO* линијом подређеног уређаја и *SPI\_CLK* линија једног уређаја са *SPI\_CLK* линијом другог уређаја. Потом се коришћењем *SSH* протокола комуницира са *Raspberry Pi* (надређеним уређајем) и покретањем *Python* скрипте (која ће детаљније бити објашњена у програмском решењу) се генерише секвенца *SPI* сигнала. Пример снимљеног *SPI* сигнала је приказан на слици 3.7 . Окидач је подешен на 100 mV док је скала подешена на 200 микросекунди. Скала сваког од канала се подешава на 1V. Дијаграм повезивања *SPI* је приказан на слици 3.8 .



Слика . Приказ екрана осцилоскопа приликом снимања *SPI* сигнала



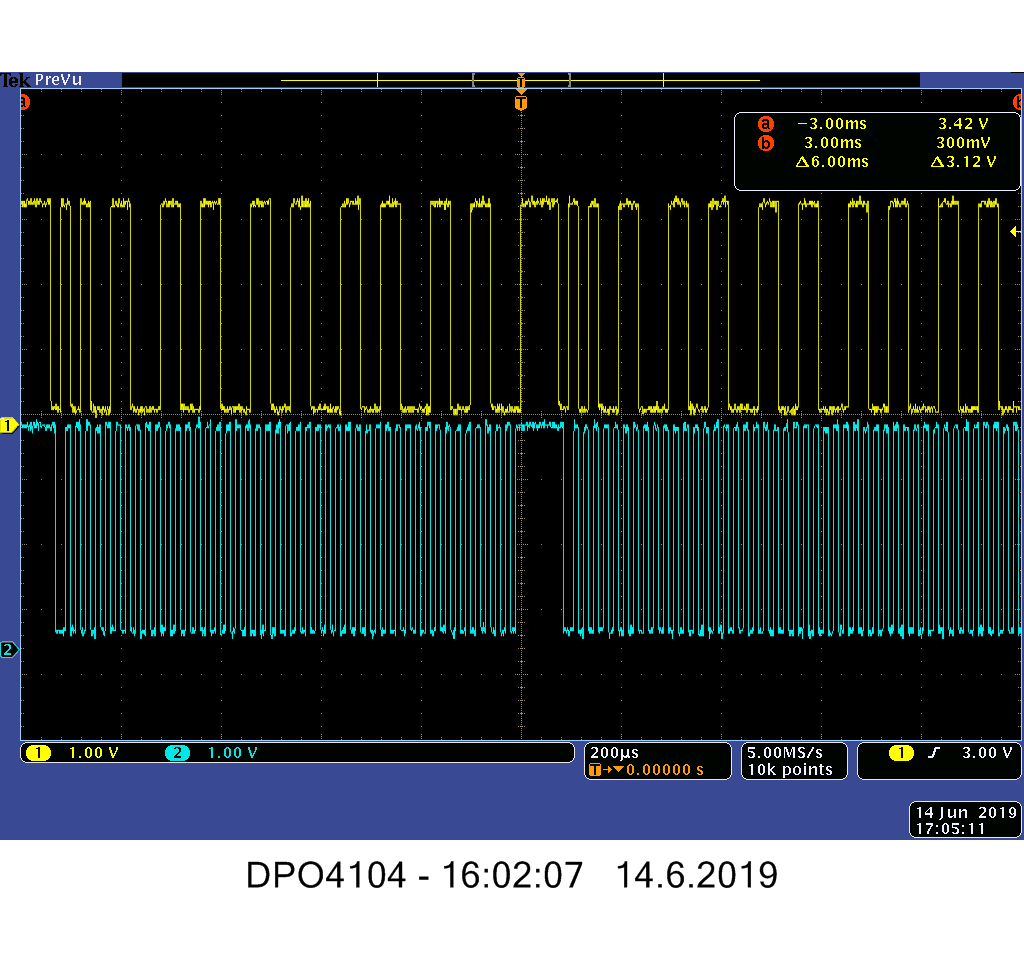
Слика . Дијаграм повезивања *SPI*

## Повезивање *I2C* – а и мерење са осцилоскопом

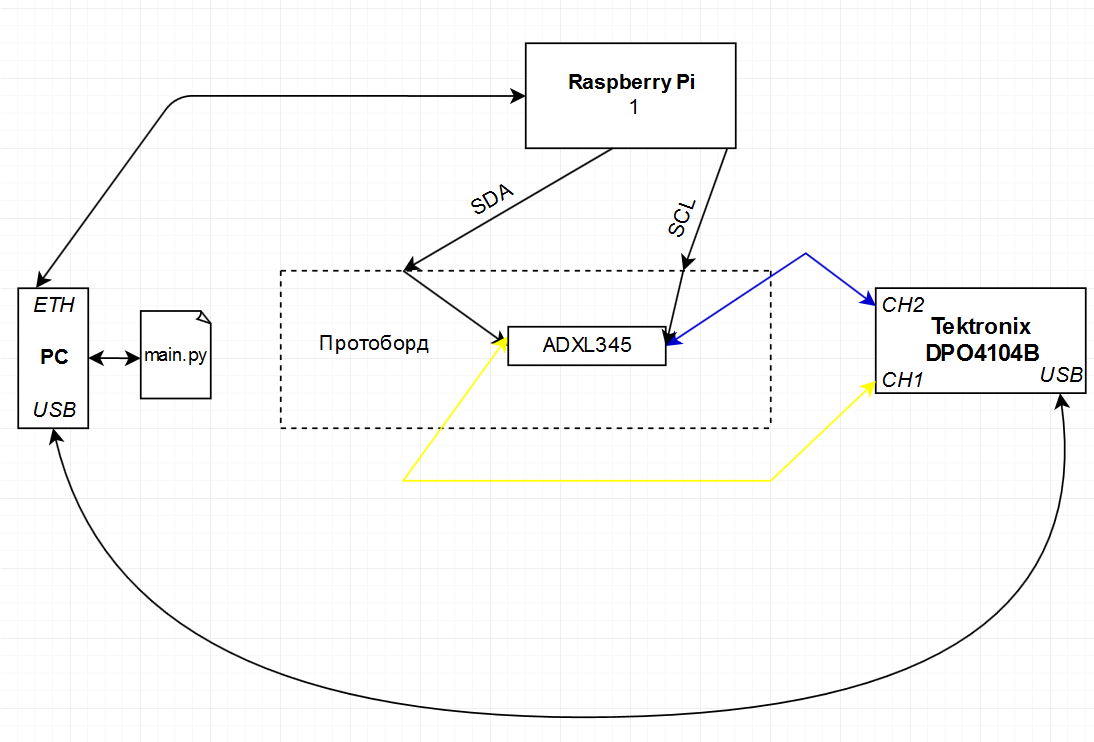
Повезивање *I2C* се врши коришћењем пинова 1 (*DC Power*), 3 (*SDA*), 5 (*SCL*) и 30 (*Ground*). Због тога што овај протокол на *Raspberry Pi* захтева уређај повезан на ове пинове за успешну детекцију саобраћаја на мрежи, користи се *ADXL345* сензор за акцелерометар произведен од стране *Sunfouder* компаније.

Потом сам коришћењем 2 сонде, при чему је намена сонде на каналу 1 осцилоскопа била да прикупи сигнал са подацима (*SDA*), а друга сонда да прикупи сигнал са радним тактом (*SCL*). Пример снимања ова 2 сигнала осцилоскопом је приказан на слици 3.9 .

Окидач за сваки од ова 2 канала је подешен на 3V док је скала подешена на 200 микросекунди. Дијаграм повезивања *I2C* је приказан на слици 3.10 .



Слика . Приказ екрана осцилоскопа приликом снимања *I2C* сигнала



Слика . Дијаграм повезивања *I2C*

# Програмско решење

## Комуникација са осцилоскопом, мерење и обрада сигнала

Комуникација са осцилоскопом је урађена коришћењем *Python PyVisa* модула. Овај модул омогућава иницијализацију осцилоскопа као објеката класе *ResourceManager*. Функцији *open\_resource* овог објекта као параметар се шаље идентификациони број осцилоскопа који је приказан у апликацији *OpenChoiceDesktop*. Размењивање података са осцилоскопом се своди на 2 методе објекта - *write()* и *ask().* Праћењем програмерског приручника *Tektronix*-а [10] за овај модел осцилоскопа послате су команде као параметар једној од две претходно описане фунције, у зависности од намене. Послате су команде за одабир канала, постављање хоризонталне и вертикалне скале, а потом за прикупљање података. Део кода за подешавање и прикупљање података са осцилоскопа:

scope = visa.ResourceManager().open\_resource(instrument\_id)

scope.write('DATA:SOU CH' + str(channel))

scope.write("HORizontal:SCAle" + str(scale))

scale\_readout = float(scope.ask("HORizontal:SCAle?"))

scope.write("CH"+channel\_num+":SCAle " + str(ch\_scale))

scope.write("TRIGger:A SETLevel")

scope.write('DATA:WIDTH 1')

scope.write('DATA:ENC RPB')

ymult = float(scope.ask('WFMPRE:YMULT?'))

yzero = float(scope.ask('WFMPRE:YZERO?'))

yoff = float(scope.ask('WFMPRE:YOFF?'))

xincr = float(scope.ask('WFMPRE:XINCR?'))

scope.write('CURVE?')

data = scope.read\_raw()

headerlen = 2 + int(data[1])

header = data[:headerlen]

ADC\_wave = data[headerlen:-1]

ADC\_wave = np.array(unpack('%sB' % len(ADC\_wave), ADC\_wave))

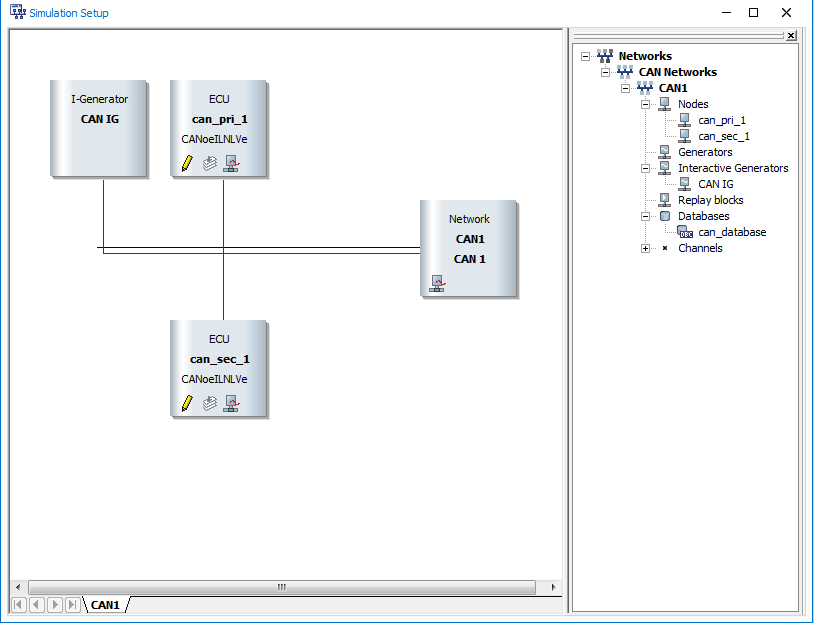
volts = (ADC\_wave - yoff)\*ymult + yzero

time = np.arange(0, xincr\*len(volts), xincr)

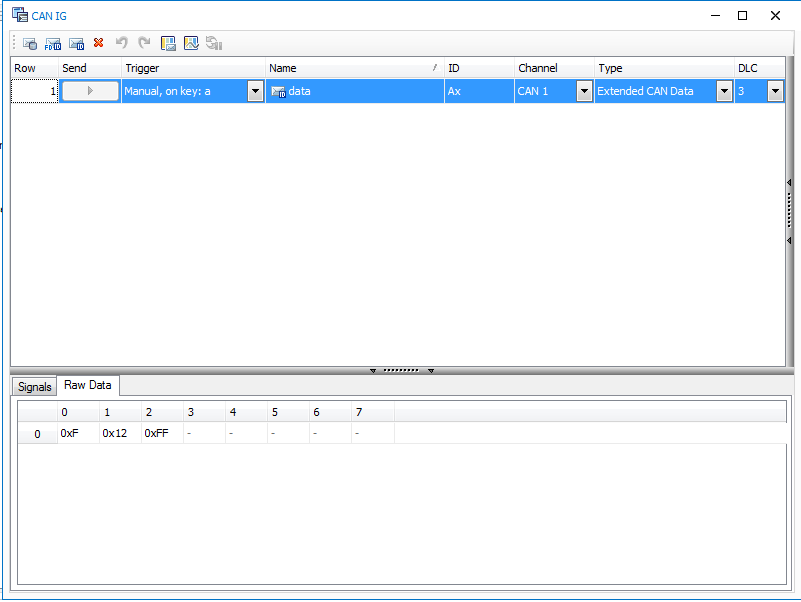
## *CANoe*

У *CANoe* је прво креирана одговарајућа конфигурација. Потом је у *CANoe* Options менију под *Measurment* опцијом и у *General* прозору *Channel Usage* подешен на одговарајући број *CAN* канала (2). У *Simulation Setup* прозору су потом креирана 2 *ECU* чвора и један *I-Generator* који су повезани на претходно направљену *CAN* мрежу. *Simulation Setup* је приказан на . *I-Generator* прозор је приказан на .  
У менију *Hardware* под опцијом *Network Hardware* је намештена одговарајућа бодна брзина која мора да одговара бодној брзини која се користи за декодовање. Ако ово није правилно подешено, декодовање неће бити исправно. Потом је креирана база података за *CAN* мрежу у *CANoe Candb++*, унутар које су креирана 2 чвора (примарни и секундарни).

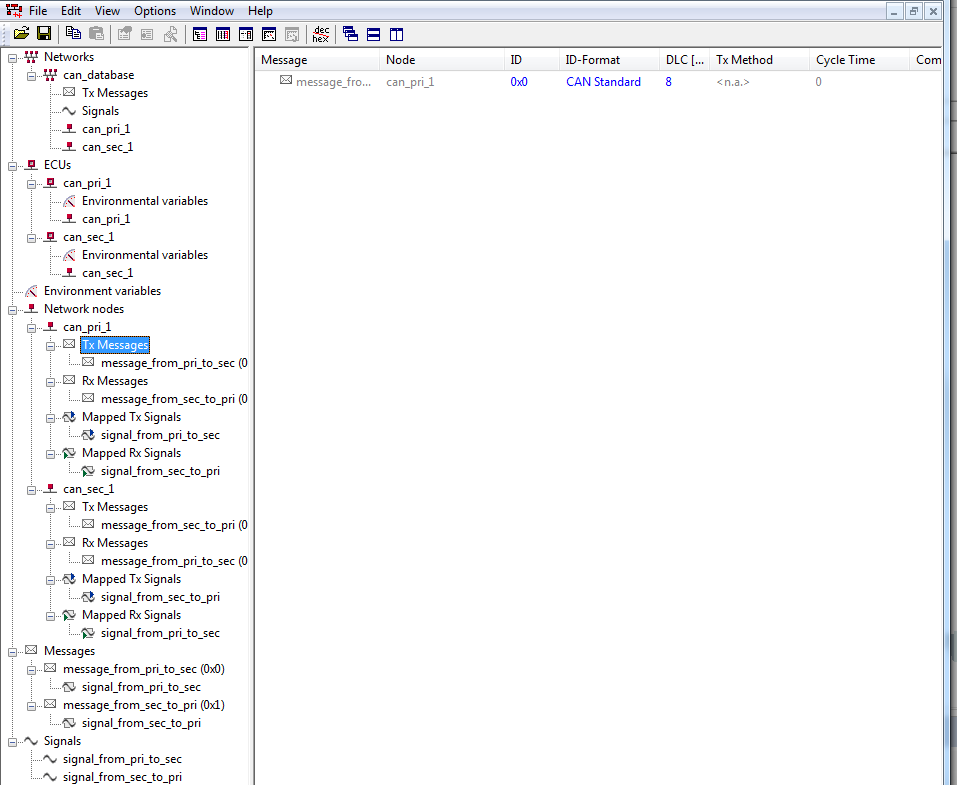
Након тога су креирана 2 сигнала при чему је један резервисан за слање података од примарног чвора до секундарног, а други за слање обрнутим смером. Онда су ови сигнали мапирани за одговарајућу поруку која је претходно креирана. На крају су ове поруке мапиране на сваки од коресподентних чвора и овим се завршава креирање базе података. *Canoe CANdb++* *Editor* је приказан на слици 4.3 . Унутар *I-Generator* **-** а су креирани сигнали који ће се слати од примарног до секундарног чвора. За креацију сигнала се мора одабрати канал, *ID*, тип сигнала (да ли је са стандардним идентификатором или продуженим), дужина података а унутар *Raw Data* се конфигурише сваки од октета за слање. Пре самог покретања *.cfg* конфигурације, мора се потврдити да унутар *Home* прозора буде подешено да је конфигурација у *Online Mode* и да се користи *Real Bus*, пошто ће се сигнал генерисати хардверски, коришћењем модула *VN8912A*. Праћење сигнала у времену се врши унутар *Data* прозора из *Analysis* менија.



Слика . Приказ *Simulation* *Setup* прозора унутар *CANoe* - а



Слика . Приказ *I-Generator* прозора



Слика . Приказ *Canoe CANdb++ Editor* - а

## Декодовање *CAN* – а

Декодовање *CAN* сигнала се врши поштујући *CAN 2.0* протокол. Напони прикупљеног сигнала се представљају у облику 1 и 0, при чему су сви делови сигнала са напоном изнад претходног постављеног окидача постављени на 1, док су остали постављени на 0. Након тога су избачени из сигнала сви убачени и лавински бити. Коришћењем *Knuth-Morris-Pratt* алгоритма се препознавају лавински бити и 5 истих доминантних или рецесивних бита, и за свако место у низу где су се појављивали ови бити се ти подаци бришу. Само тада се могло наставити са декодовањем при чему се гледа 13. бит сигнала који указује на то да ли је коришћен *CAN 2.0 A* или *CAN 2.0 B*.

Потом се издвајају сви релевантни оквири и израчунава се циклична провера редудансе. Ако се ова израчуната провера поклапа са цикличном провером редудансе издвојеном из сигнала, главном програму се враћа декодован *CAN* сигнал са назнаком да је провера тачна (оквири су означени зеленом бојом), у супротном је враћен декодован *CAN* сигнал али означен црвеном бојом. Део кода који објашњава декодовање претходно обрађеног сигнала:

*# Bit stuffing removal.*

for s in KnuthMorrisPratt(start\_decoding\_voltage\_high, [0, 0, 0, 0, 0, 1]):

for\_deletion.append(s)

for s in KnuthMorrisPratt(start\_decoding\_voltage\_high, [1, 1, 1, 1, 1, 0]):

for\_deletion.append(s)

iteration = 0

while iteration < len(for\_deletion):

del start\_decoding\_voltage\_high[for\_deletion[iteration]+5-iteration]

iteration += 1

ide = start\_decoding\_voltage\_high[13]

if ide == 0:

sof = start\_decoding\_voltage\_high[0]

id\_can = start\_decoding\_voltage\_high[1:12]

rtr = start\_decoding\_voltage\_high[12]

reserved\_bits = start\_decoding\_voltage\_high[14]

length\_can = int(

"".join(map(str, start\_decoding\_voltage\_high[15:19])), 2)

data\_can = start\_decoding\_voltage\_high[19:19+8\*length\_can]

data\_can\_hex = []

for i in range(length\_can):

data\_can\_hex.append("{0:0>2X}".format(

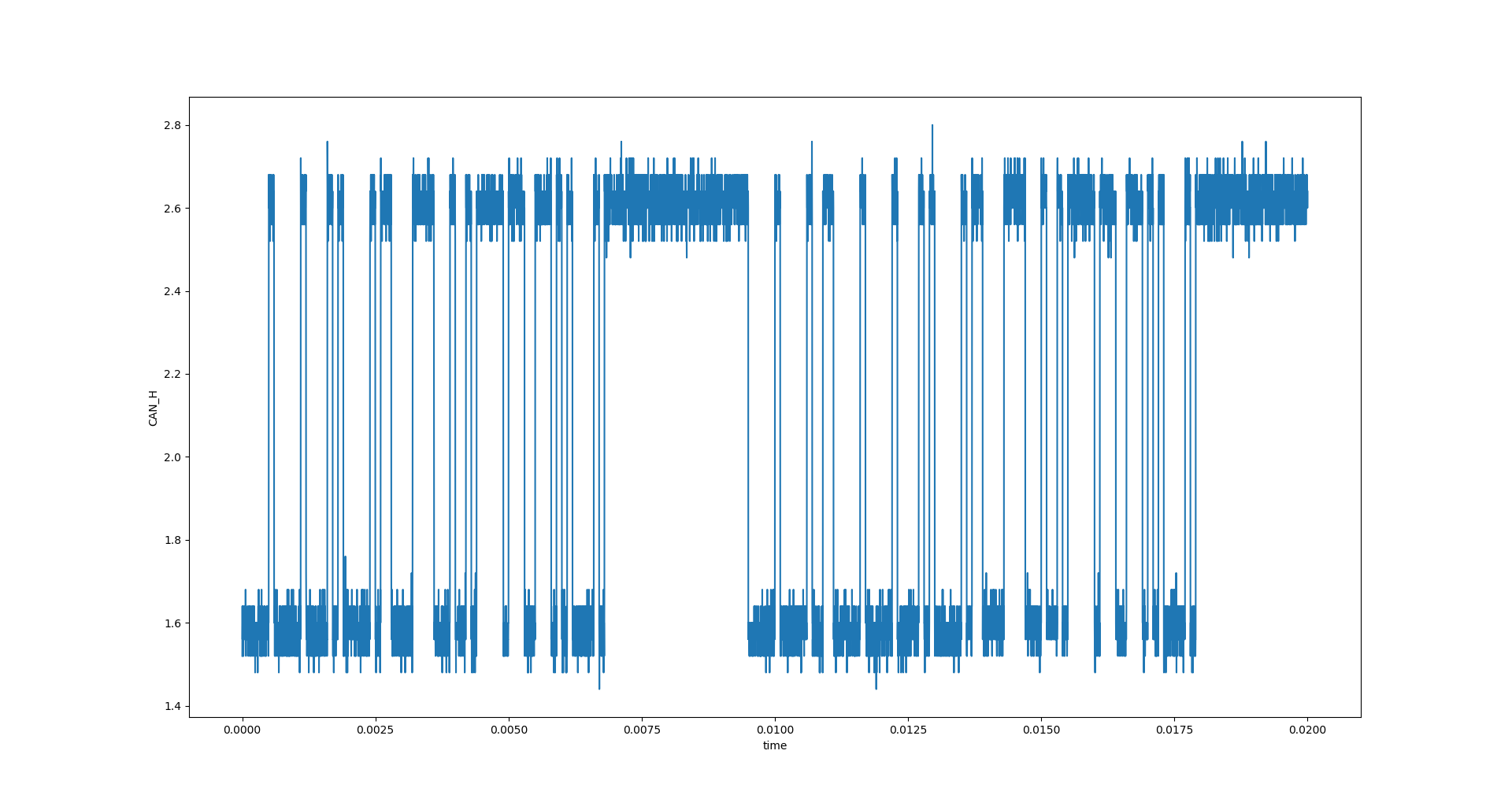
int("".join(map(str, start\_decoding\_voltage\_high[19+i\*8:19+i\*8+8])), 2)))

crc\_can = start\_decoding\_voltage\_high[19 +

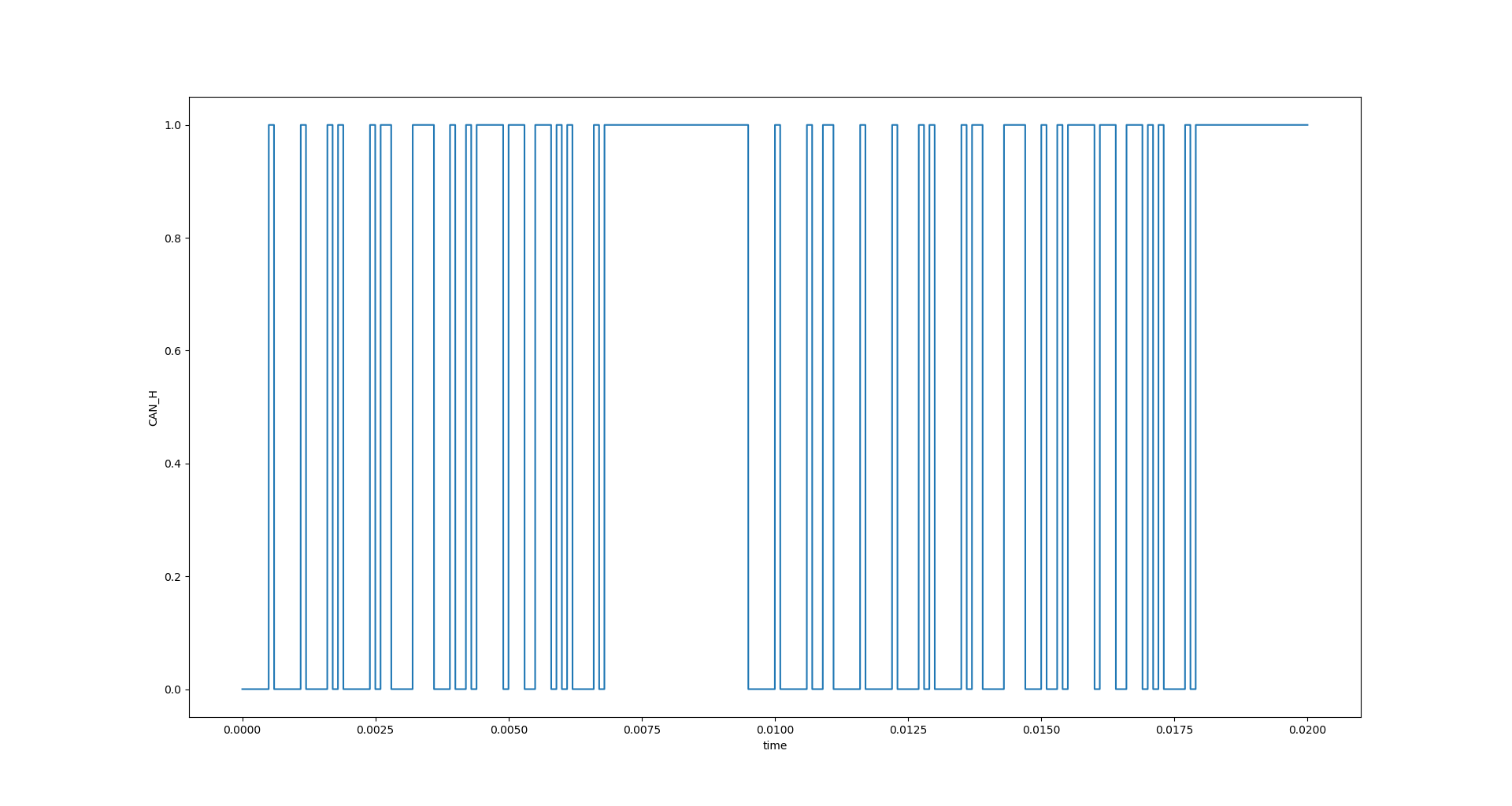
8\*length\_can:19+8\*length\_can+15]

for\_crc = start\_decoding\_voltage\_high[:19+8\*length\_can]

Ухваћен *CAN* сигнал, необрађен и обрађен је приказан на слици 4.4 и слици 4.5 , респективно.



Слика . Приказ ухваћеног необрађеног *CAN* сигнала са осцилоскопа



Слика . Приказ ухваћеног *CAN* сигнала са осцилоскопа након обраде

## Декодовање *LIN* – а

Као што је наведено у концепту решења, у овом поглављу ће бити објашњене стандардне команде за *Carberry* за *LIN* протокол. Прво преко серијског порта се мора отворити конекција за сваки од *Carberry* модула, коришћењем *LIN OPEN* команде. Постоји 5 различитих режима рада које су подржане од стране *Carberry*-ја (Табела 4 )

| **Mод** | **Команда** | **Опис команде** |
| --- | --- | --- |
| *Master 1x* | *LIN OPEN MASTER1X* <бодна брзина> | Отвара канал при чему аутоматски препозанаје брзину,понаша се као надређени,стандардна контролна сума |
| *Master 2x* | *LIN OPEN MASTER2X* <бодна брзина> | Отвара канал при чему аутоматски препозанаје брзину,понаша се као надређени,продужена контролна сума |
| *Slave* *1x* | *LIN OPEN SLAVE1X* | Отвара канал при чему аутоматски препозанаје брзину,понаша се као подрећени,стандардна контролна сума |
| *Slave* *2x* | *LIN OPEN SLAVE2X* | Отвара канал при чему аутоматски препозанаје брзину,понаша се као подрећени,продужена контролна сума |
| *Free* | *LIN OPEN FREE* <бодна брзина> | Отвара канал на брзини<бодна брзина> за паритет и конролну суму је одговоран корисник |

Табела *Carberry* команде за *LIN*

Коришћен је последњи, „слободни“ режим због флексибилности. Након слања сваке команде, ако је она исправна, добија се одговор *OK*. Слање *LIN* оквира у слободном режиму се ради употребљавањем *LIN TX* <заштићени идентификатор> [*D0…D7*] [контролна сума] команде. Да би се проверило да ли је контролна сума исправна, израчуната је независно праћењем упутства из *LIN* спецификације. Потом се пореди да ли израчуната контролна сума одговара контролној суми у оквиру. Такође се мора проверити паритет за сваки од послатих оквира. У зависности од тога да ли су подаци послати у оквиру одговарали израчунатим подацима, на стандардном излазу екрана се шаље обавештење да ли има грешке у послатом оквиру. Код који објашњава конекцију са сваким од Carberry модула и слање *LIN* оквира:

data = checksum(lin\_data, lin\_id, lin\_version)

com3 = serial.Serial(port="COM3", baudrate=115200)

com4 = serial.Serial(port="COM4", baudrate=115200)

com3.write("LIN CLOSE\r\n")

com4.write("LIN CLOSE\r\n")

com3.write("LIN OPEN FREE 1000\r\n")

com4.write("LIN OPEN FREE 1000\r\n")

com4.write("LIN TX %s\r\n" % (data))

Декодовање *LIN* се заснива спецификацији *LIN 2.0*, а само декодовање оквира је објашњено у коду :

sync\_break = decoded\_lin[0:13]

sync\_field = decoded\_lin[15:22]

sync\_field\_to\_hex = ""

for i in range(len(sync\_field)):

sync\_field\_to\_hex += str(sync\_field[i])

sync\_field\_hex = "{0:0>2X}".format(int(sync\_field\_to\_hex, 2))

rest\_of\_lin = decoded\_lin[25:len(decoded\_lin)]

id\_field = ""

parity\_bits = ""

data\_field = []

id\_field = "{0:0>2X}".format(

int("".join(map(str, decoded\_lin[25:33][::-1])), 2))

parity\_bits = "{0:b}".format(

int("".join(map(str, decoded\_lin[25:33][::-1])), 2))[0:2]

if id\_field >= 0x00 and id\_field < 0x1f:

length = 2

elif id\_field >= 0x20 and id\_field < 0x2f:

length = 4

else:

length = 8

length = length

for x in range(length):

data\_field.append("{0:0>2X}".format(

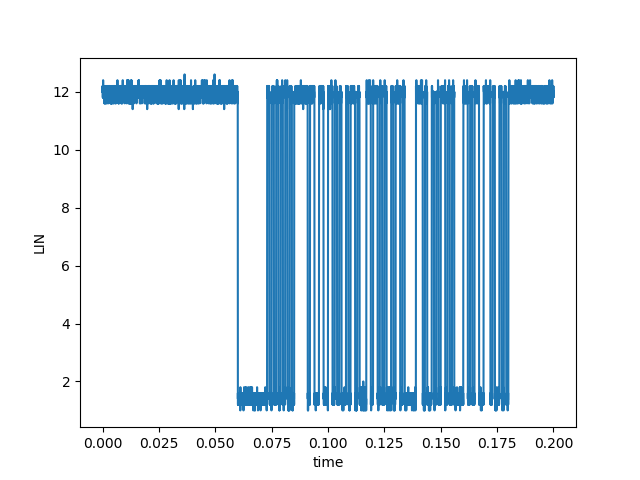
int("".join(map(str, decoded\_lin[(35 + (x \* 10)):(43 + (x \* 10))][::-1])), 2)))

checksum = int(("{0:0>2X}".format(

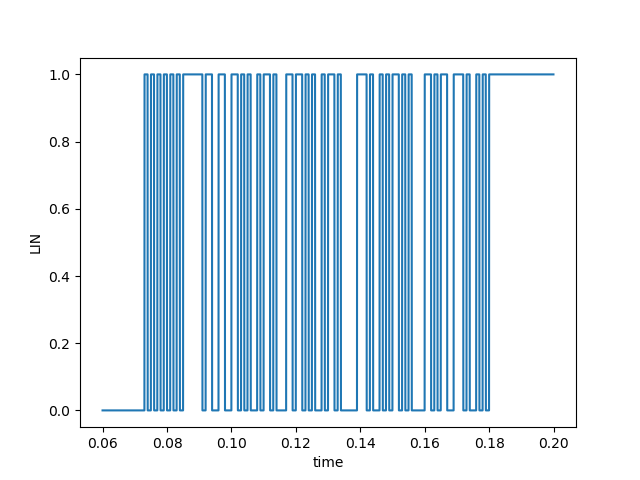
int("".join(map(str, decoded\_lin[(35 + (length \* 10)):(43 + (length \* 10))][::-1])), 2))))

return id\_field, int(parity\_bits), data\_field, checksum

Пример ухваћеног LIN сигнала, необрађеног и обрађеног је приказан на сликама, респективно.



Слика . Приказ обрађеног LIN сигнала



Слика . Приказ обрађеног LIN сигнала

## Декодовање *SPI* – а

Да би се препознали подаци који треба да се декодују, прво су се морали издвојити податаци на *MOSI* линији који се шаљу у исто време када се појави одређена секвенца на *SCLK* линији. За претрагу секвенце на *SCLK* линији се користи *Knuth-Morris-Pratt* алгоритам. Након што су издвојени корисни подаци, декодовани су и представљени у хексадецималном формату. Декодовање *SPI* сигнала је објешњено кодом:

for s in KnuthMorrisPratt(start\_decoding\_spi\_data, [0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]):

data\_final.append("{0:0>2X}".format(

int("".join(map(str, star\_decodig\_spi\_data[s:s+16:2])), 2)))

return data\_final

## Декодовање *I2C* – а

Декодовање *I2C* сигнала је започето претрагом почетне секвенце за сваки од канала (*SDA*, *SCL*) – почетна секвеца канала за пренос радног такта је за један бит већа. За претрагу ове две секвенце је коришћен *Knuth-Morris-Pratt* алгоритам. Декодовање је објашњено кодом:

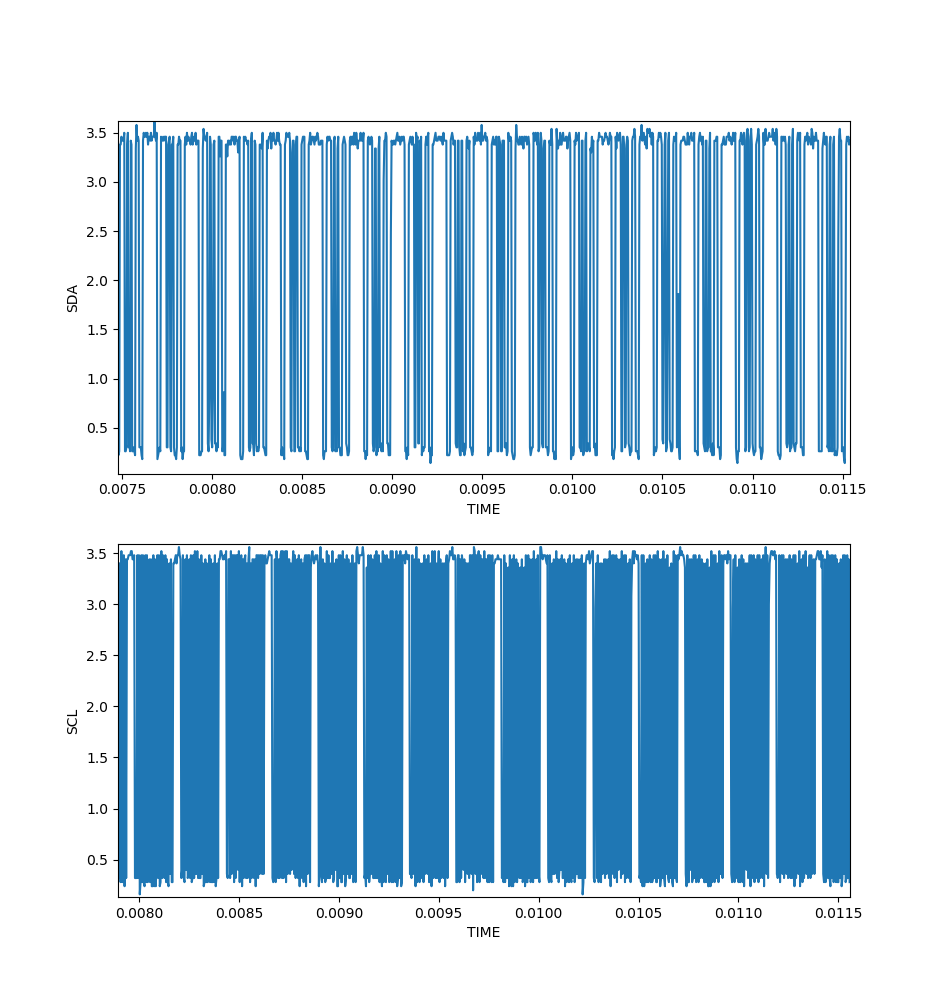
for s in KnuthMorrisPratt(start\_decoding\_spi\_data, [0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]):

data\_final.append("{0:0>2X}".format(

int("".join(map(str, star\_decodig\_spi\_data[s:s+16:2])), 2)))

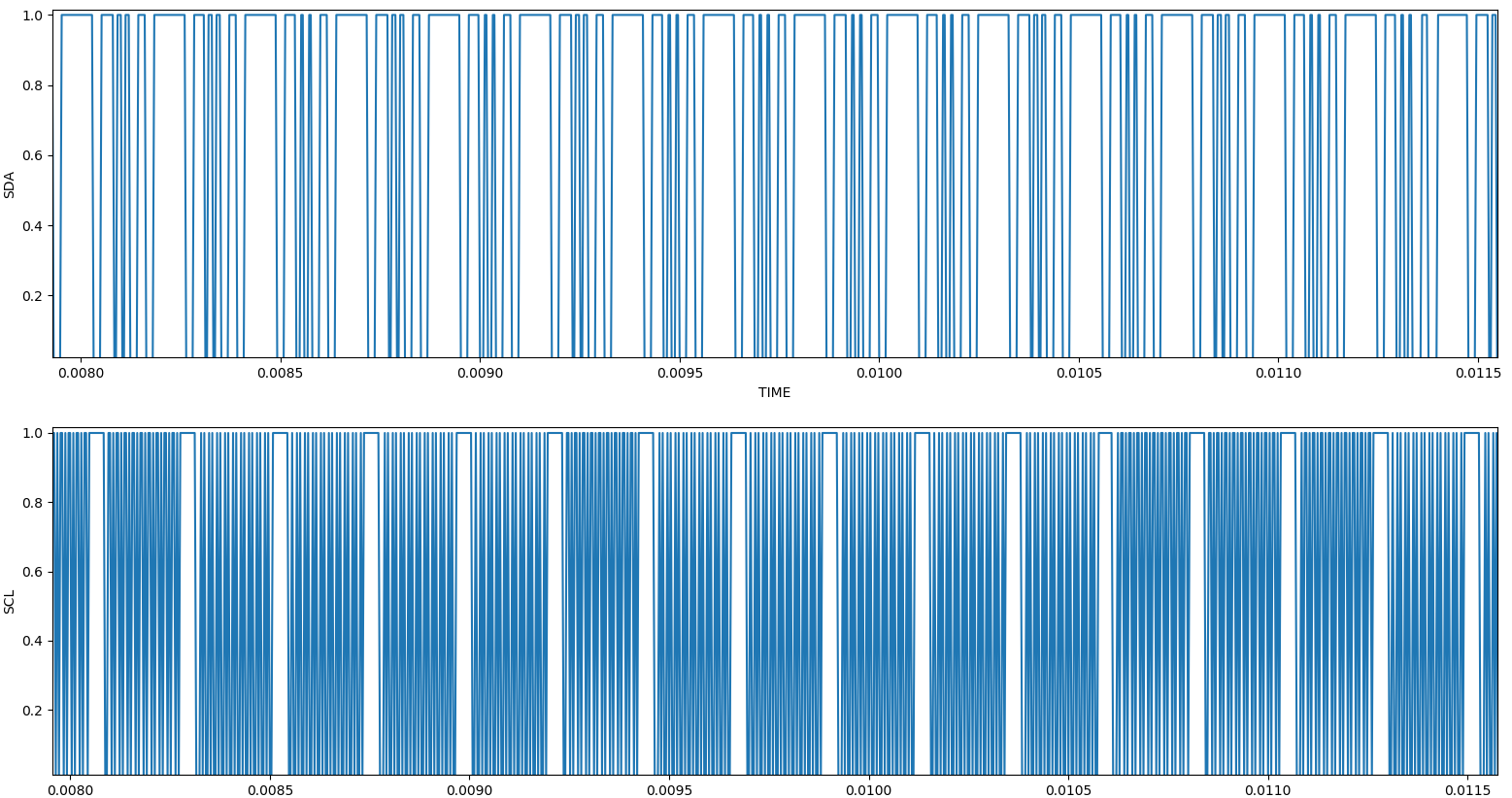
return data\_final

Приказ необрађеног сигнала је на слици 4.8 . Приказ ухваћеног сигнала који је обрађен и спреман за декодовање је приказан на слици 4.9 .



Слика . Пример необрађеног *I2C* сигнала

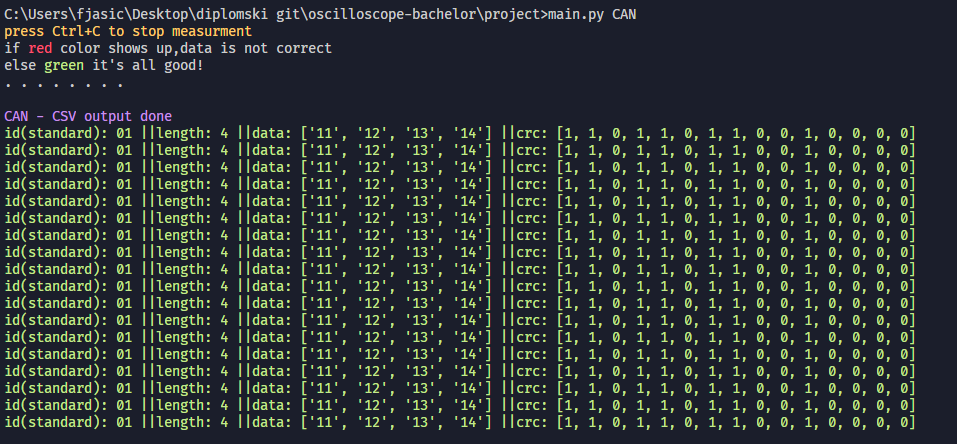
Слика 4.10 Пример необрађеног I2C сигнала



Слика . Приказ обрађеног *I2C* сигнала

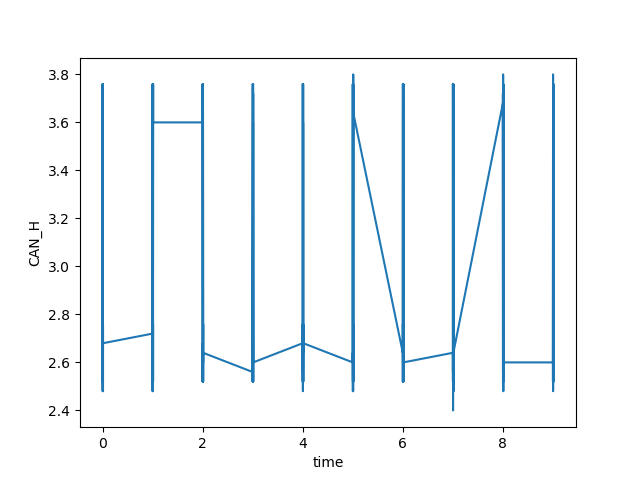
# Тестирање и резултати

Провера исправности имплементације декодера за протоколе је урађена покретањем генерисањем *CAN, LIN, SPI и I2C* оквира и потом посматрања излаза покренуте апликације. Приликом покретања апликације за декодовање, потребно је као аргумент унети тип протокола (у случају *LIN* – a, потребно је унети и коју врсту контролне суме желимо да декодујемо), као и режим у којем се извршава(*online* или *offline*). Генерисање *CAN* оквира је урађено у *CANoe* - у на начин описан у претходном поглављу. Генерисање *LIN* оквира је урађено у *serial\_lin.py*. Генерисање *SPI* оквира је урађено у *sendSPI.py*, док је генерисање *I2C* оквира урађено у *sendI2C.py*. Пример излаза успешног декодовања је дат на слици 5.1 .



Слика . Пример излаза успешног декодовања

Потешкоће приликом израде су настале при самом мерењу и обради података, јер при континуалном прикупљању података долази до кашњења између 2 оквира. Ово је проузроковано коришћењем *Python* **-** а који иако олакшава имплементацију програма, уводи проблеме кашњења и нетачности приликом мерења. Пример кашњења између 2 *CAN* оквира може се видети на слици 5.2 . Кашњење између оквира је око једне секунде.



Слика . Кашњење између *CAN* оквира

# Закључак

У овом раду је реализовано декодовање *CAN, LIN, SPI* и *I2C* оквира. Функционалност имплементираног декодера олакшава анализирање сигнала који се често користе у аутомобилској индустрији, тиме што корисник овог програма не мора да константно води рачуна о подешавању самог осцилоскопа, већ је подешавање и анализа оквира аутоматски урађена.

Даљи развој ће се фокусирати на аутоматској детекцији тренутног проткола (комплетна аутоматизација програмског решења) и на додавање још функционалности као што су: детекција максималне и минималне амплитуде сигнала, фазе сигнала.

# Литература

[1] Automotive industry. [Online].

Доступно: [https://en.wikipedia.org/wiki/Automotive\_industry](https://en.wikipedia.org/wiki/Automotive_industry%20) [приступљено: јул 2019.]

[2] CAN bus. [Online].

Доступно: <https://en.wikipedia.org/wiki/CAN_bus> [приступљено: јул 2019.]

[3] “BOSCH CAN Specification Version 2.0”, 1991. [Online].

Доступно: [https://www.academia.edu/9998014/BOSCH\_CAN\_Specification](https://www.academia.edu/9998014/BOSCH_CAN_Specification%20) [приступљено: јул 2019.]

[4] ISO 11898-1 “Road vehicles — Controller area network (CAN) — Part 1:

Data link layer and physical signalling”, First edition, Geneva, Switzerland, ISO, 2003.

[5] Local Interconnect Network. [Online]

Доступно: <https://en.wikipedia.org/wiki/Local_Interconnect_Network> [приступљено: јул 2019.]

[6] “LIN Specification Package Revision 2.0”, 2003. [Online].

Доступно: <https://forums.ni.com/attachments/ni/30/3619/1/LIN.pdf> [приступљено: јул 2019.]

[7] Serial Peripheral Interface. [Online].

Доступно: <https://en.wikipedia.org/wiki/Serial_Peripheral_Interface> [приступљено: јул 2019.]

[8] I²C. [Online].

Доступно: [https://en.wikipedia.org/wiki/I%C2%B2C](https://en.wikipedia.org/wiki/I%C2%B2C%20) [приступљено: јул 2019.]

[9] MSO4000B and DPO4000B Series Digital Phosphor Oscilloscopes User Manual. [Online]

Доступно: [https://download.tek.com/manual/071281002web.pdf](https://download.tek.com/manual/071281002web.pdf%20) [приступљено: јул 2019.]

[10] MSO4000 and DPO4000 Series Digital Phosphor Oscilloscopes Programmer Manual. [Online] Доступно: [https://download.tek.com/manual/077024801web.pdf](https://download.tek.com/manual/077024801web.pdf%20) [приступљено: јул 2019.]