УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Филип Јашић

Једно решење примене осцилоскопа *Tektronix DPO 4104B* у аутоматским тестовима

ДИПЛОМСКИ РАД
- Основне академске студије -

Нови Сад, септембар, 2019.

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

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**:			
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)			
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**:			
Датум прихватања теме, **ДП**:			
Датум одбране, **ДО**:			
Чланови комисије, **КО**:	Председник:	проф. др	
	Члан:	доц. др	Потпис ментора
	Члан, ментор:	доц. др	

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

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**:			
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)			
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**:			
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	PhD	
	Member:	PhD	Menthor's sign
	Member, Mentor:	PhD	

Захвалност

Захваљујем се породици и пријатељима на пруженој подршци током читавог школовања.

Захваљујем се и ментору доц. др Богдану Павковићу, као и Слободану Рачановићу и Јовици Кричковићу на стручној помоћи и саветима током израде овог рада.

САДРЖАЈ

Списак слика

Списак табела

Скраћенице

API – **A**pplication **p**rogramming **i**nterface, програмска спрега апликације

ASIC – **A**pplication-**s**pecific **i**ntegrated **c**ircuit, интегрисано коло специфичне намене

CAN – **C**ontroller **A**rea **N**etwork, управљачка мрежа за подручје

CRC – **C**yclic **r**edundancy **c**heck, циклична провера редундансе

ECU – **E**lectronic **C**ontrol **U**nit, електронска управљачка јединица

I²C – **I**nter-**I**ntegrated **C**ircuit, међу-интегрисано коло

ISO – **I**nternational **O**rganization for **S**tandardization, Међународна организација за стандардизацију

LIN – **L**ocal **I**nterconnect **N**etwork, локална међу-повезана мрежа

LSB – **L**east **S**ignificant **B**it, бит најмање важности

MSB – **M**ost **S**ignificant **B**it, бит највеће важности

SPI – **S**erial **P**eripheral **I**nterface, серијска периферна спрега

UART – **U**niversal **a**synchronous **r**eceiver-**t**ransmitter, универзални асинхрони пријемник-предајник

USB – **U**niversal **S**erial **B**us, универзална серијска магистрала

1. Увод

Главни покретач развоја мрежних технологија у аутомобилима су били напреци остварени у електронској индустрији, нарочито у Сједињеним Америчким Државама. Због стриктне контроле 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 управљање емисијом штетних гасова, већ су повећали и перформансе аутомобила, сигурност, комфор, али су и допринели смањењу трошкова производње. Током година се број електронских модула у аутомобилу повећавао. Електронска управљачка јединица (енгл. *ECU*) представља уграђени систем који управља једним или више електронских модула. У модерним возилима може да се нађе и до 80 електронских управљачких јединица. *ECU* добија податке од сензора који се конвертују у одређену јединицу, где се тако обрађени подаци даље шаљу актуаторима који извршавају одређене функције. Понекад је потребно да ови модули међусобно комуницирају (нпр. модул мотора мора да обавести модул мењача која је тренутна брзина мотора, који опет мора даље да обавести остале модуле када се промени степен преноса). Овај пренос података треба да буде брз и поуздан. Како се временом повећавао број ових компоненти, дошло је до велике комплексности у ожичавању појединих модула. Штавише, ожичавања аутомобила су се разликовала од модела до модела, што је додатно повећавало трошкове производње.

Одговор аутомобилске индустрије на претходно описане проблеме је био да се створи централна мрежа у возилу. Идеја је била да модули буду само „прикачени“ на ове мреже како би могли да лакше међусобно комуницирају. Овај дизајн је био лакши за производњу, одржавање и дао је додатну флексибилност за надограђивање додатних модула без промене целокупне 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.
- Задрже функционалност и у отежаним условима.
- Робусност и поузданост.

Иако су тренутни захтеви аутомобилске мреже за брзином смањени у односу на захтеве мрежа неких других система, због убрзаног развоја технологија и све већих перформанси аутомобила, постојање и потреба за бржим магистралама.

Један од задатака овог рада јесте имплементација декодовања сигнала протокола чија је употреба честа у аутомобилској индустрији, чиме би се олакшала анализа целокупног система. Фреквентну примену у дијагностици аутомобилских мрежа има осцилоскоп. Пошто сваки протокол има своје карактеристике које га разликују од осталих протокола, када се ручно ради дијагностика осцилоскопом, за сваки од протокола се осцилоскоп мора подесити на одговарајући начин. Један од циљева овог рада јесте и аутоматизација подешавања осцилоскопа, а декодовањем сигнала протокола се добија читљивији приказ евентуалних грешака или проблема на мрежи, што додатно убрзава процес дијагностике.

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** – ретроспектива урађеног и анализа правца за даљи развој програмског решења.

2. Теоријске основе

У овом поглављу су дата објашњења протокола који се користе у аутомобилској индустрији са акцентом на начин повезивања и изгледом оквира сваког од сигнала протокола.

2.1 *I²C* протокол

I²C [1] је синхрона магистрала са могућношћу да има више руководиоца и више подређених створена од *Philips Semiconductor*-а 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 протокола, створен од стране *Intel*-а 1995. године као одговор на потребу за дефинисанијом, стриктнијом употребом и циљем да се дода робусност и интер-опребилност.

Модерни I^2C системи подржавају нека правила из *SMBus*-а, при чему је потреба за реконфигурацијом мала. Оригинално употребљаван у телевизорима, сада подржава разне врсте периферија (наменски системи – *EEPROM*, *Flash* и *RAM* меморија, *watchdog* тајмери, микроконтролери, персонални рачунари). *I²C* подржава више брзина комуникације.

2.1.1 Начин функционисања

I²C користи две 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 управља тактом (не генерише га), и иницира комуникацију, док подређени уређај прима такт и одговара на прозивку руководиоца, али може да обори такт уколико му је потребно више времена за одговор. 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 функционисања 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 400 pF). Максимална дужина линије је неколико метара. Пошиљалац поставља податке на *SDA* линију, а прималац их потврђује. Руководилац започиње комуникацију почетном 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 шаље се адресно поље, прозвани подређени уређај потврђује; након потврде иду подаци које такође прати потврда. Могуће је слање више података истовремено. Оквир изгледа исто приликом слања и читања, где је статусни бит читања и писања једина разлика.

Такође, могућ је и комбиновани пренос који служи за промену смера трансакције, где руководиоц уместо стоп секвенце шаље продужени старт, а затим и 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 података пожељно је радити узорковање и подешавање на ивицу такта.

2.2 *SPI* протокол

SPI [2] је синхроно, серијско комуникационо окружење које се користи за комуникацију на кратким дистанцама, првенствено у уграђеним системима. Овај

протокол је осмишљен од стране *Motorola*-е средином 80-тих година 20. века. Неке од карактеристика овог окружења 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, могућност постојања једног надређеног и више подређених уређаја. Надређени активира подређене ниским сигналом. Обе, *MISO* и *MOSI*, линије су активне током сваког преноса.
- Брже од *UART*-а: 250 *kb/s* до 2 *Mb/s*.
- Кратак домет: ~10 *cm* на сваки чвор.
- Могућност повезивања процесора и широког спектра подржаних периферија. Могуће је и повезивање више зависних подређених уређаја (један руководиоц и више нанизаних подређених – енгл. *Daisy chained*).
- Групе бита података (4, 8, 16); непостојање почетног и крајњег бита. Одвојене линије за пренос у оба смера (могућ 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 четири регистра:

- *SDPR* (*Transferred data read/write register*) – смештање примопредајних података.
- *SPCR* (*Control Register*) – смештање контролних подешавања.
- *SPSR* (*Status Register*) – смештање статусних информација.
- *SPTCI* (*Transfer Complete Interrupt*) – смештање прекидачке рутине за готов пренос.

2.2.1 Пренос података, фаза и поларитет

Да би започео комуникацију, руководилац магистрале прво 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 од стране подређеног, руководилац мора да чека тај период пре него што пошаље радни такт.

Након сваке периоде радног такта, долази до преноса у оба смера – пуни дуплекс. Руководилац шаље бит на *MOSI* линији и потом га подређени прочита, док је на *MISO* линији обрнуто. Овај процес се дешава чак и када се подаци шаљу у једном смеру.

Пренос углавном укључује два 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 шаљу га даље свом одговарајућем уређају. На следећој ивици такта, сваки од уређаја прими бит са линије за 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, руководиоц престаје да генерише ивице такта. Ако треба да се пошаље још података, померачки регистри се опет испочетка „напуне“ подацима и пренос опет може да започне. Трансмисија се углавном састоји од 8 бита.

Постоје четири режима као последица комбинација:

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

Фаза дефинише значење предње и задње ивице такта; пресликавање узорковања и прелаза на нови бит некој од ивица. Различите ивице имају различито значење. Поларитет 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 се врши на узлазну ивицу (0→1), а прелаз на нови бит је на силазну ивицу (1→0).
- За *CPHA*=1, узорковање се врши на силазну ивицу (1→0), а прелаз на нови бит је на узлазну ивицу (0→1).

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

- За *CPHA*=0, узорковање се врши на силазну ивицу (1→0), а прелаз на нови бит је на узлазну ивицу (0→1).
- За *CPHA*=1, узорковање се врши на узлазну ивицу (0→1), а прелаз на нови бит је на силазну ивицу (1→0).

MOSI и *MISO* сигнали су углавном стабилни за пола циклуса до следећег прелаза на нови бит. *SPI* руководиоц и подређени могу да одабирају сигнале на различитим одбирцима у том полу-циклусу. Ово даје додатну флексибилност комуникационом каналу између руководиоца и подређеног уређаја.

2.3 *LIN* протокол

LIN [3] је серијски мрежни протокол, развијен од стране *LIN* конзорцијума (5 произвођача аутомобила – *BMW*-а, *Volkswagen Group*-е, *Audi Group*-е, *Volvo Cars*-а, *Mercedes-Benz*-а; једног добављача полупроводника – *Motorola*-е; и једног добављача алата – *VCT*-а) 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 различитим индустријама (у аутомобилима, медицинској опреми, белој техници...).

Најважније одлике *LIN* прокола су:

- Брзина до 20 *kb/s* на дужинама до 40 метара.
- Руководилац управља медијумом, због чега постоји гаранција кашњења и нема судара. Усклађивање такта је одговорност зависних уређаја .
- Варијабилност у дужини послатих података (8, 16 или 64 бита).
- Флексибилност у конфигурацији и динамичка топологија.
- Величина мреже је углавном до 16 чворова (могуће је и до 64, али уз низак проток).
- Детекција дефектних чворова, детекција грешке и контролни збир података.
- Оперативни напон од 12 *V*.
- Лак за коришћење, велика доступност компоненти, јефтинији него *CAN* и друге магистрале за комуникацију, возила су поузданија, продуживање магистрале је лако за имплементацију, не постоје трошкови за лиценцирање.

Постоји више верзија *LIN* протокола, од којих су у овом раду две релевантне надаље – 1.3 и 2.0. Нека од унапређења новије верзије су:

- Низови података су подржани тако да омогућавају дужину сигнала до 64 бита.
- Проширени контролни збир (који укључује и заштићени идентификатор) је унапређење у односу на класични контролни збир код *LIN 1.3*.
- Спорадични оквири су дефинисани.
- Управљање мрежом је мерено у секундама, не у битима.
- Управљање статусом је поједностављено, а обавештавање мреже је стандардизовано.

LIN је мрежа са дифузним емитовањем која се састоји од најчешће 16 мрежних чворова (један руководиоц и до петнаест 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 и у специјализованом хардверу или *ASIC*-у у намери да се додатно смање трошкови, површина или потребна струја. Физички слој се представља као коло са отвореним колектором, магистрала је терминирана руководиоцем (1000 Ω). Напон је између 7 V и 18 V на прикључцима.

2.3.1 Оквир

Време које је потребно да се пренесе *LIN* оквир на магистралу је сума свих времена за које је потребно да се пренесе сваки бит, времена за одговор и времена између два оквира. Време између 2 бита 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[1] бита, укључујући почетни бит и након њих, размак за почетак поља.
    - Поље за синхронизацију (енгл. *Sync*) увек има вредност која у хексадецималном формату износи 55, да би уређаји на мрежи могли да препознају ово поље.
    - Заштићени идентификатор (енгл. *Protected identifier*) се састоји од идентификатора и парности идентификатора. Вредност од 0. до 5. бита означава идентификатор док вредност од 6. до 7. бита представља парност идентификатора. Вредности идентификатора имају и функцију одређивања примене послате *LIN* поруке па тако:

[1] Доминантни бит представља логичку нулу. Представљање логичке нуле као доминантног бита се врши да би електронске управљачке јединице са најнижим идентификационим бројем имале највећи приоритет.

1. Вредности од 0 до 59 (*3b* у хексадецималном облику) се користе за пренос података.
2. Вредности 60 (*3c* у хексадецималном облику) и 61 (*3d* у хексадецималном облику) се користе за слање дијагностичких података.
3. 62 (*3e* у хексадецималном облику) је резервисано за употребе дефинисане од стране корисника.
4. 63 (*3f* у хексадецималном облику) је резервисано за будућа побољшања.

Парност бита се израчунава на следећи начин (при чему су вредности у једначинама: *ID* – представља индекс идентификатора, док су *P0* и *P1* бити паритета):

$$P0 = ID0 \oplus ID1 \oplus ID2 \oplus ID4$$
$$P1 = \neg (ID1 \oplus ID3 \oplus ID4 \oplus ID5)$$

- Размак за одговор (енгл. *Response space*).
- Поље за одговор (енгл. *Response*) – унутар кога се налази поље за податке и поље за проверу контролног збира. Поље за податке се састоји од 8 до 64 бита. Након ових поља за податке долази поље за проверу контролног збира. Оно садржи 8-битну инвертовану суму са преносом за све податке унутар поруке (у случају *LIN* 1.3 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 (побољшано поље за проверу контролног збира, енгл. *enhanced checksum*, користи се у верзији *LIN* 2.0). Идентификатори са вредностима 60 и 63 ће увек користити класичну контролну суму. Врсте одговора су:
  1. Безусловни оквир (енгл. *Unconditional Frames*) (идентификатор 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 током сваког циклуса распореда. Могуће је слати више пута у току истог циклуса. 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 две врсте дијагностичког оквира:

- Оквир захтева (енгл. *Master Request Frame*) (идентификатор 60) – заглавље и одговор шаље руководиоц. Употребљава се кад је потребан захтев за дијагностиком или подешавањем.
- Оквир одговора (енгл. *Slave Response Frame*) (идентификатор 61) – заглавље шаље надређени, 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 – кратки одговори који не заузимају распоред. Последица овога је да долази до ретких појава судара, који се додатно избегавају добром архитектуром мреже. Сударе решава руководилац.
4. Спорадични оквир (енгл. *Sporadic frame*) (идентификатор 0-59). Логика овог оквира је слична претходно описаном оквиру.
5. Остали оквири (енгл. *Other frames*) (идентификатор 62-63).

- Размак између 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 (нпр. због лошег квалитета подређених компоненти се стварају кашњења). Номинална вредност за пренос оквира је једнака броју послатих података у битима. Рачунање овог временског интервала се врши на следећи начин:

$$\mathbf{T_{номинално\_време\_заглавља} = 34 * T_{бит}}$$

$$\mathbf{T_{номинално\_време\_одговора} = 10 * (ББП + 1) * T_{бит}}$$

$$\mathbf{T_{номинално\_време\_оквира} = T_{номинално\_време\_заглавља} + T_{номинално\_време\_одговора}}$$

$$\mathbf{T_{бит} = 1 / брзина\ такта}$$

ББП представља број бајта података (1 бајт је овде 8 бита). $\mathbf{T_{бит}}$ представља време потребно за пренос једног бита. Максимално време трајања оквира је још 40% веће у односу на $\mathbf{T_{номинално\_време\_оквира}}$.

2.4 *CAN* протокол

CAN [4] протокол је најзаступљенији протокол који се користи у аутомобилској индустрији. Развијен од стране *Robert Bosch GmbH*-а 1983. године, док је званично

пуштен у употребу 1986. године на *Society of Automotive Engineers (SAE)* конференцији у Детроиту.

Верзија спецификације се састоји из два дела:

- А (покривен *ISO* 11519 стандардом) – овај део спецификације описује стандардни формат са 11-битним идентификатором. Мрежа где уређај користи овакав стандардни идентификатор често узима назив *CAN 2.0 A*, коришћен надаље у раду.
- Б (покривен *ISO* 11898 стандардом) – овај део спецификације описује проширени формат са 29-битним идентификатором. Мрежа где уређај користи овај продужени идентификатор се често назива и *CAN 2.0 B* – назив коришћен надаље у раду.

Неке од карактеристика *CAN* протокола су:

- Брза, серијска, заједничка магистрала (брзине до *1Mbps*) за више уређаја (до 32 уређаја по *ISO* 11898; слободан приступ – било ко може да приступа када медијум није заузет), коришћење упредене парице (домет до 40m).
- Асинхрона комуникација (окидање на догађаје).
- Величина оквира података: 0-64 бита.
- Филтрирање на пријему, препознавање грешке (коришћењем цикличне провере редудансе), мере за ограничавање грешака.
- Одзив у реалном времену.
- Већа кашњења за поруке ниског приоритета, мала кашњења за поруке високог приоритета.

2.4.1 Физички слој

CAN мрежа поседује могућност да има више власника (енгл. *masters*). Све електронске управљачке јединице се међусобно повезују на магистралу са две жице (једна се зове *CAN-H*, друга *CAN-L*). Ове две жице чине упредену парицу (оклопљена или неоклопљена) и потребно их је омеђити отпорником са номиналном карактеристичном импеданцом од 120 Ω у случају брзе *CAN* мреже, док у случају споре *CAN* мреже се користи отпорник од бар 100 Ω (отпорност се засебно израчунава).

Логичка стања се процењују на основу разлике напона:

- За брзу *CAN* магистралу (*ISO* 11898-2) [5]:

- Рецесивни бит[2] представља разлику од 0 *V* (у теорији) између *CAN-H* и *CAN-L* жице; у пракси се та разлика креће од 0.5 *V*. Напон на *CAN-H* жици се креће ка 5 *V*, док се на *CAN-L* жици креће ка 0 *V*.
- Доминантни бит представља разлику од око 2 *V* (мора биту у распону између 1.5 *V* и 3.5 *V*) између *CAN-H* и *CAN-L* жице. Напон на *CAN-H* жици се креће ка 5 *V* док напон на *CAN-L* жици се креће ка 0 *V*.

- За спору *CAN* магистралу (*ISO* 11898-3):
  - Рецесивни бит представља разлику од бар 0.6 *V* између *CAN-H* и *CAN-L* жице. Напон на *CAN-H* жици се креће ка 0 *V*, док напон на жици *CAN-L* иде ка 5 *V*.
  - Доминантни бит представља разлику од бар 2.3 *V* између *CAN-H* и *CAN-L* жице. Напон на *CAN-H* жици се креће ка 5 *V*, док напон жице *CAN-L* иде ка 0 *V*.

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.
- За спору *CAN* магистралу нема потребе за ограђивањем крајева линије. Ограђују се уређаји и свака линија посебно: *RTH-CAN-H* и *RTH-CAN-L*.

2.4.2 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 оклопљене 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 9-пинске *D-sub,* и 5-пинске *mini style* утичнице.

2.4.3 Оквири поруке

Пренос порука је контролисан са четири типа оквира:

[2] Рецесивни бит представља логичку јединицу.

- Оквир за податке (енгл. *Data frame*) – носи податке од предајника до пријемника. Оквир за податке се састоји од седам поља (разлике између *CAN* 2.0 *A* и *CAN* 2.0 *B* су по потреби посебно наведени, док се подразумева да остала поља имају исту структуру – табела 1):
    - Почетак оквира (енгл. *Start of frame*) означава почетак поља за оквир за податке или оквира за даљинско управљање. Састоји се од једног доминантног бита. 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 најзначајних бита (од *ID-10* до *ID-4*) не смеју бити рецесивни. *RTR* бит мора да буде доминантан када се шаље оквир за податке, иначе, када се шаље оквир за даљинско управљање овај је бит рецесиван.
        - У случају *CAN* 2.0 *B* идентификатор се састоји од 29 бита, подељених на два дела: идентификатор А (11 бита) и идентификатор Б (18 бита). Између ова два дела продуженог идентификатора се налази *SRR* бит (1 рецесиван бит) и *IDE* бит (1 рецесиван бит).
    - Контролно поље (енгл. *Control field*). За ово поље такође постоје разлике између две врсте 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 прва 2 бита су резервисани бити – *r0*, *r1* (могу бити или доминанти или рецесивни). Након њих у оквиру се налазе 14 бита који означавају дужину послатих података и функционишу на исти начин као у бржем протоколу.

  - Поље за податке (енгл. *Data field*) се састоји од 0 до 64 бита. Подаци се шаљу у MSB редоследу.
  - Поље за цикличну проверу редундансе (енгл. *CRC field*) се састоји од 15 бита који представљају израчунату цикличну проверу редундансе и делимитера који заузима 1 бит, при чему је тај бит рецесиван.
  - Поље за потврду о пријему (енгл. *ACK field*) се састоји од *ACK* временског интервала (енгл. *slot*) и *ACK* делимитера где сваки од поља заузима по 1 бит и при чему пошиљаоц поставља рецесивни бит у *ACK* временском интервалу, док 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.
- 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 преклапање (енгл. *Overload frame*) се користи да омогући кашњење између два послата оквира за податке или оквира за грешке.

***CAN* 2.0 *A* (11-битни идентификатор)**		***CAN* 2.0 *B* (29-битни идентификатор)**	
Назив поља	**Дужина поља**	**Назив поља**	**Дужина поља**
Почетак оквира	1 бит	Почетак оквира	1 бит
Идентификатор	11 бита	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 1 Разлике између *CAN* 2.0 *A* и *CAN* 2.0 *B*.

2.4.4 Арбитража

Када год је магистрала слободна, било која електронска управљачка јединица може да почне са слањем поруке. У случају да две или више 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 времену, 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 пошиљалац проверава на ком нивоу се налази бит идентификатора који је он послао поредећи га наспрам нивоа бита идентификатора који се налази на магистрали. Ако су ови бити на истим нивоима једнаки, пошиљалац 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 пошиљалац „изгубио“ арбитражу и мора да се повуче, при чему се прекида пренос преосталих бита у оквиру. Другим речима, 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.

2.4.5 Убацивање бита, препознавање грешака, контролна сума

Убацивање бита се врши на следећи начин: након 5 идентичних бита се убацује супротан бит; ово побољшава усклађивање, а додатни бити не мењају податке пошто је убацивање урађено хардверски. Ова техника генерално није потребна, пошто се врло ретко појављују истоветне вредности на магистрали. Проблем са убацивањем бита је појава такозваних „лавинских“ битова (нпр. 5 рецесивних бита које прати 5 доминантних бита) који могу да збуне механизам за препознавање убачених бита.

Статистичка учестаност грешака *CAN* протокола зависи од укупног броја уређаја, физичког ожичења и распореда, као и спољашњих електромагнетних сметњи. Ток препознавања грешака између пошиљаоца и примаоца је независан од филтрирања и маскирања на пријему и састоји се из пет засебних механизама:

1. Надгледање појединачних бита – препознавање локалних и глобалних грешака код пошиљаоца; упоређује се стање на магистрали са послатим битом; не примењује се на поље за арбитражу.
2. Провера структуре – типична поља су *CRC* делимитер, *ACK* делимитер и *EOF* и чине их увек рецесивни бити.
3. Провера кодовања (уметање бита) – пријемник проверава ток бита, на сваких 5 узастопних истоветних бита мора да дође убачена промена; укључује све бите од *SOF* до краја *CRC*.
4. Провера потврде је обавеза пошиљаоца – потребно је бар да пристигне једна потврда па да пошиљалац подеси рецесивни бит; такође, потребно је да пријем обори са доминантним.
5. Циклична провера редундансе – пошиљалац прорачунава пре слања, пријемник проверава подударање при пријему. Циклична провера редундансе се рачуна на следећи начин: полином, чији су коефицијенти представљени комбинацијом вредности претходних поља (поље за почетак оквира, поље за арбитражу, контролно поље, поље за податке), при чему је претходно урађено избацивање убачених бита, а коефицијенти 15 најнижих бита чине нуле. Овај полином се потом дели (коефицијенти су израчунати модулом двојке) са генератор-полиномом:

$$x^{15} + x^{14} + x^{10} + x^{8} + x^{7} + x^{4} + x^{3} + 1$$

Остатак овог дељења представља 15-битну вредност *CRC* секвенце која се шаље на магистралу као додатак оригиналне поруке. Да би се применила ова функција, 15-битни померачки регистар *CRC_RG(14:0)* се користи. *NXTBIT* означава следећи бит у низу. Псеудо-код ове функције се описује са:

```
CRC_RG = 0;
REPEAT
      CRCNXT = NXTBIT EXOR CRC_RG(14);
      CRC_RG(14 :1) = CRC_RG(13 :0);
      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 поља за податке, *CRC_RG* садржи *CRC* секвенцу.

3. Концепт решења

3.1 Осцилоскоп

Осцилоскоп је мерни уређај који се користи за графички приказ напона као функције времена једног или више сигнала у облику дводимензионалног дијаграма. Може имати један или више канала за прикупљање сигнала. На улаз за сваки од канала је повезана сонда. Постоји више врста осцилоскопа, али их све карактерише подешавање хоризонталне скале (временске базе – део времена по хоризонталном подеоку – изражено у секундама, милисекундима или микросекундима) и вертикалне скале (напонске резолуције – величина напона по вертикалном подеоку – изражено у *V* или *mV*). Сви осцилоскопи имају и подешавање окидача који служи за синхронизацију. Пошто се у електроници јављају различити сигнали, синхронизација може да буде веома комплексна. Потенциометром *LEVEL* се подешава напонски ниво улазног сигнала на коме ће се „окинути“ коло за синхронизацију. Избором начина повезивања сигнала на коло за синхронизацију (*AC* – без једносмерне компоненте, *DC* – директно спрегнут сигнал, *HF* – преко високофреквентног филтера, *LF* – преко нискофреквентног филтера) се може омогућити квалитетнија синхронизација. Померањем овог потенциометра може се изабрати напонски ниво при коме је приказ сигнала на екрану стабилан. Ако сигнал није стабилан, доћи ће до варирања синхронизације и до „дрхтања“ сигнала на екрану.

Приликом израде овог рада коришћен је *Tektronix DPO 4104B* осцилоскоп. Овај осцилоскоп има четири независна канала. Сондама су мерени напонски нивои који су одређени сигнали протокола имали, чије су жице претходно повезане на прототипску плочу. У овом раду је коришћен један или два канала у зависности од протокола који се користи (једино *LIN* протокол користи један канал, остали протоколи користе по два

канала). Окидачи су потом подешени за нивое измереног напона, хоризонталну и вертикалну скалу. На самом почетку ово подешавање је било ручно, праћењем упутства за употребу овог осцилоскопа [6], потом аутоматски – комуникацијом преко *USB*-а употребом *API*-ја.

3.2 Повезивање *I²C*-а и мерење са осцилоскопом

На тржишту се почетком 2012. године појавио рачунар *Raspberry Pi* који задовољава све критеријуме рачунара фон Нојмановог типа. Поседује чип *BCM2835* са *ARM11* процесором и *RAM* меморију, као и могућност повезивања са осталим помоћним компонентама, чак и оним нестандардним преко *GPIO* порта. Оно што га чини интересантним је да је у питању рачунар опште намене приступачан по цени, малих димензија, са могућношћу прикључивања нестандардне опреме.

Raspberry Pi 2 модел Б представља другу генерацију платформе која се појавила у фебруару 2015. Нови модел је донео и значајно побољшање перформанси. *Raspberry Pi* 2 је чак 6 пута бржи од свог претходника. Нови *Raspberry Pi* 2 Модел Б је истог формата као претходни *Raspberry Pi* модел Б+ али са дупло више *RAM* меморије и знатно бржим процесором.

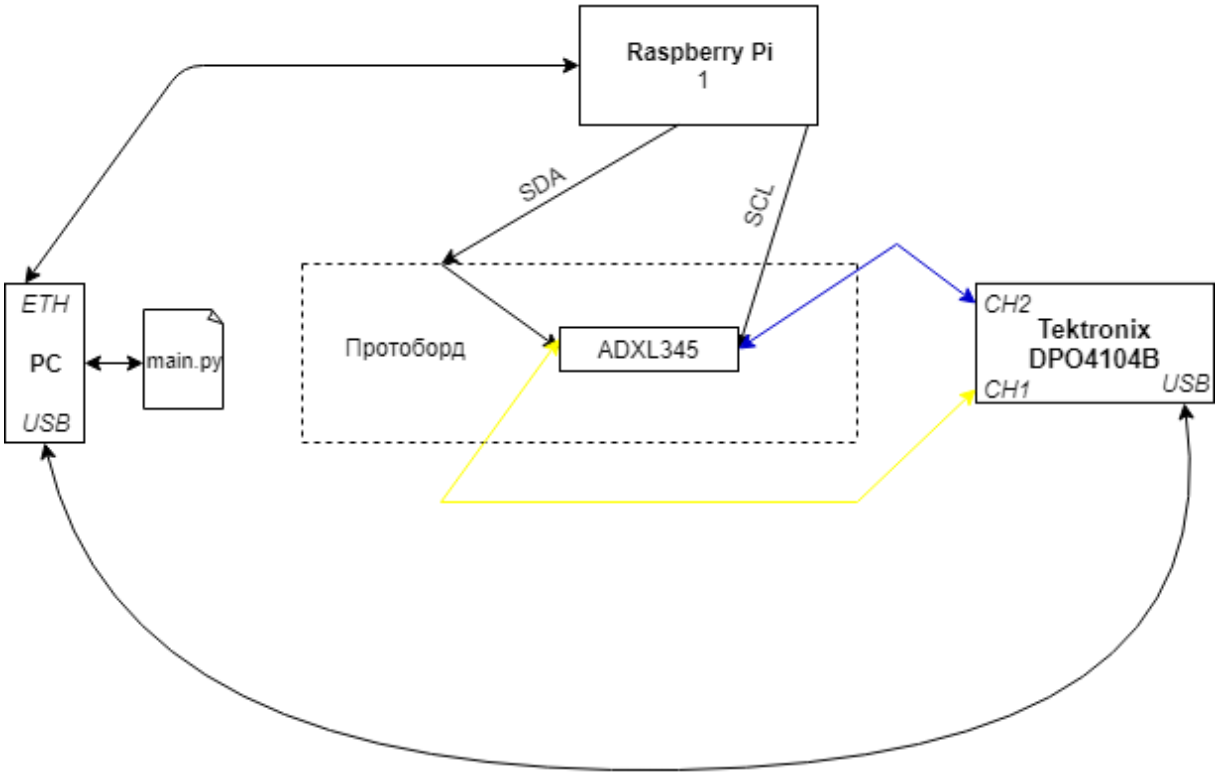
Преглед пинова *Raspberry Pi* 2 модел Б рачунара се добија куцањем команде *gpio readall* у терминалу (слика 3.1).

BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
		Pi 2								
		3.3v			1 || 2			5v		
2	8	SDA.1	ALT0	1	3 || 4			5v		
3	9	SCL.1	ALT0	1	5 || 6			0v		
4	7	GPIO. 7	IN	1	7 || 8	1	ALT0	TxD	15	14
		0v			9 || 10	1	ALT0	RxD	16	15
17	0	GPIO. 0	IN	0	11 || 12	0	ALT0	GPIO. 1	1	18
27	2	GPIO. 2	OUT	1	13 || 14			0v		
22	3	GPIO. 3	IN	0	15 || 16	0	IN	GPIO. 4	4	23
		3.3v			17 || 18	0	IN	GPIO. 5	5	24
10	12	MOSI	ALT0	0	19 || 20			0v		
9	13	MISO	ALT0	0	21 || 22	0	IN	GPIO. 6	6	25
11	14	SCLK	ALT0	0	23 || 24	1	OUT	CE0	10	8
		0v			25 || 26	1	OUT	CE1	11	7
0	30	SDA.0	IN	1	27 || 28	1	IN	SCL.0	31	1
5	21	GPIO.21	IN	1	29 || 30			0v		
6	22	GPIO.22	IN	1	31 || 32	0	IN	GPIO.26	26	12
13	23	GPIO.23	IN	0	33 || 34			0v		
19	24	GPIO.24	ALT0	0	35 || 36	0	IN	GPIO.27	27	16
26	25	GPIO.25	IN	0	37 || 38	0	ALT0	GPIO.28	28	20
		0v			39 || 40	0	ALT0	GPIO.29	29	21
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
					Pi 2					

Слика 3.1 *Raspberry Pi* 2 модел Б, преглед пинова.

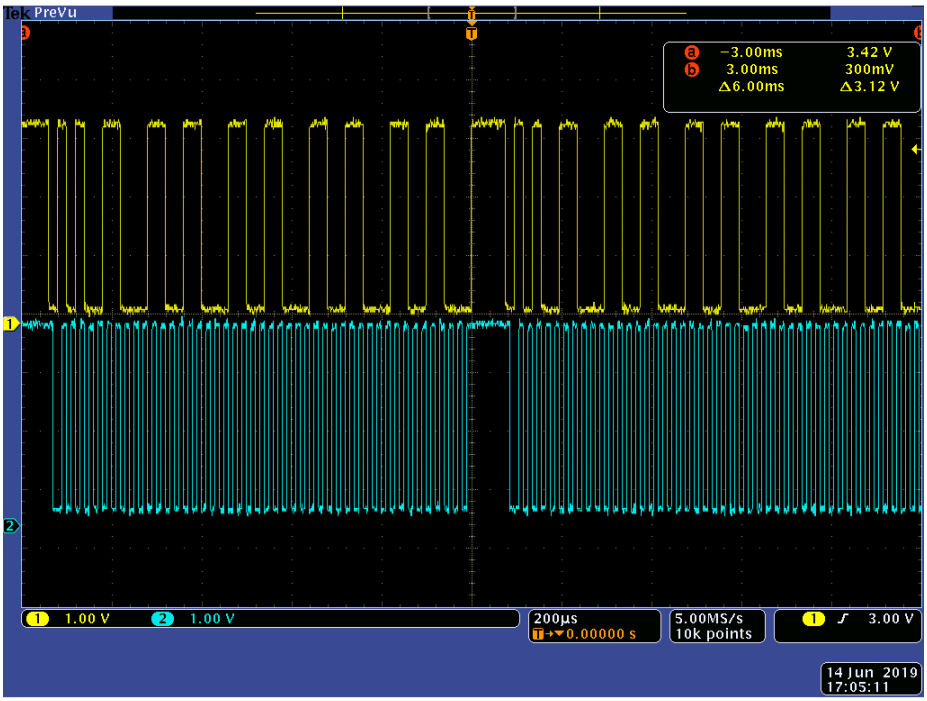
Повезивање I^2C [7] се врши прегледом пинова – 1 (*DC Power*), 3 (*SDA*), 5 (*SCL*) и 30 (*Ground*). Због тога што овај протокол на *Raspberry Pi*-у захтева уређај повезан на

ове пинове за успешну детекцију саобраћаја на мрежи, користио се *ADXL345* сензор за акцелерометар произведен од стране *Sunfouder* компаније.



Слика 3.2 Дијаграм повезивања I^2C

Потом, коришћењем две сонде вршено је очитавање сигнала, при чему је намена сонде на каналу 1 осцилоскопа била да прикупи сигнал са подацима (*SDA*), а сонде на другом каналу да прикупи сигнал са радним тактом (*SCL*). На дијаграму (слика 3.2) повезивања сонди осцилоскопа са жицама које су повезане између *Raspberry Pi*-а и *ADXL345* може да се види и како се повезује рачунар са осцилоскопом и *Raspberry Pi*-ем (они комуницирају употребом *SSH*). Окидач за сваки од канала је подешен на 3 *V*, хоризонтална скала је подешена на 200 *ms*, вертикална на 1 *V*. Након овог подешавања се могао видети сигнал I^2C протокола (слика 3.3 – жутом бојом је означен сигнал који преноси податке, а плавом бојом сигнал који преноси такт).

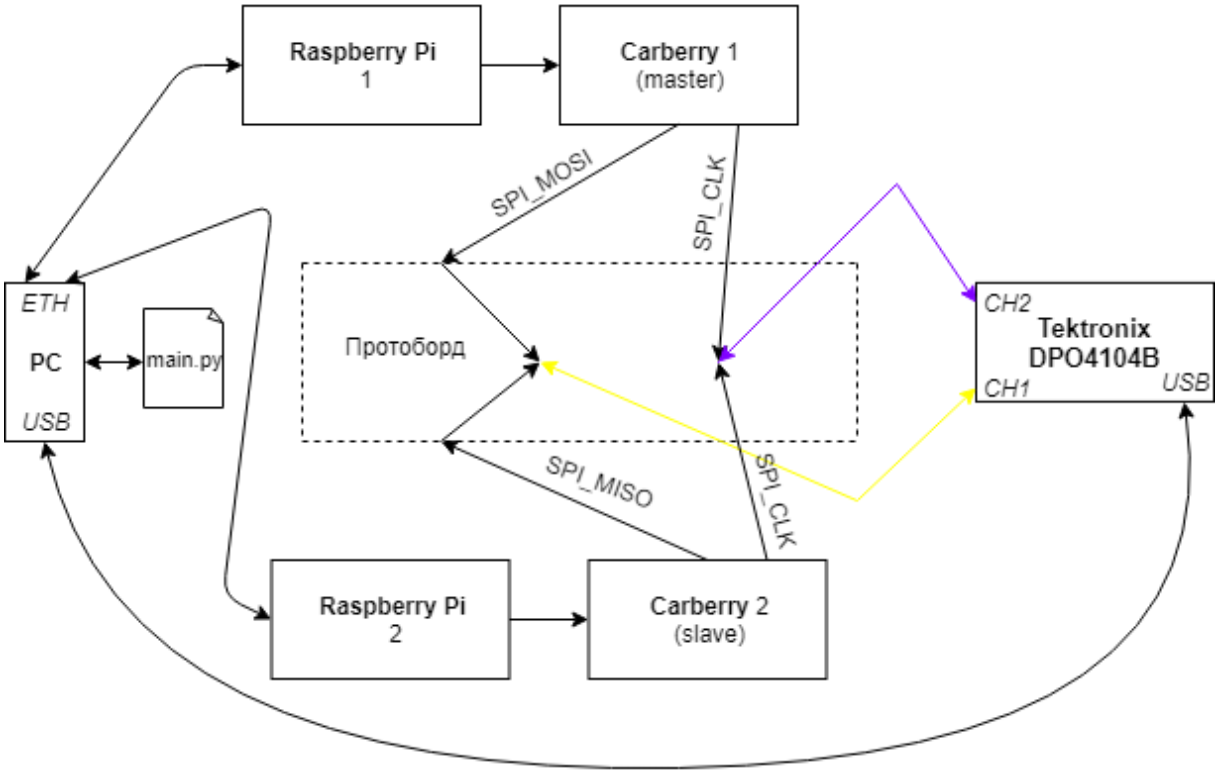


Слика 3.3 Приказ екрана осцилоскопа приликом снимања I^2C сигнала.

3.3 Повезивање *SPI*-а и мерење са осцилоскопом

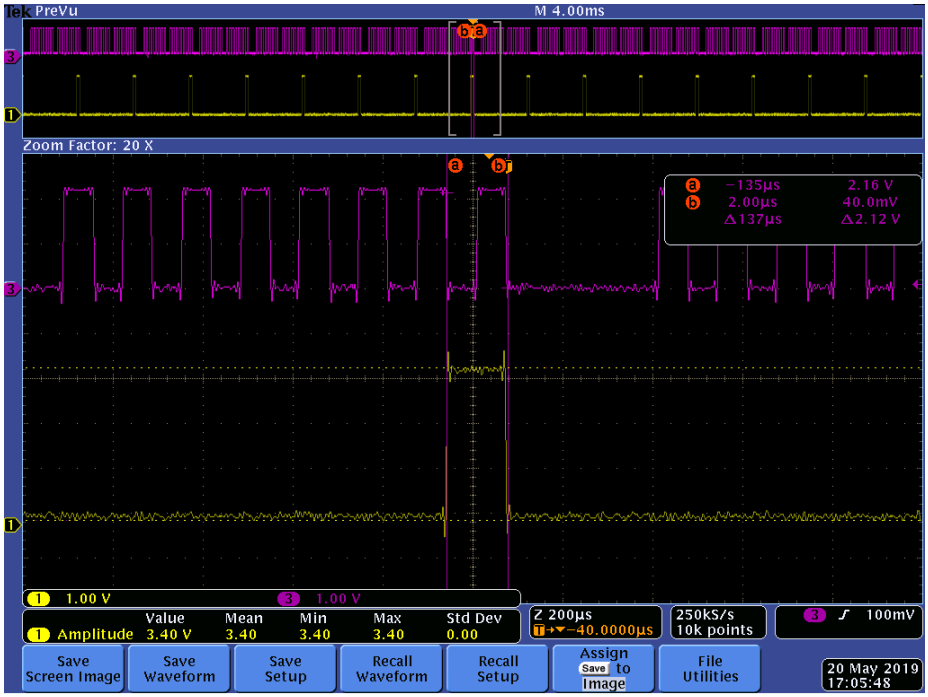
Повезивање *SPI* на прототипску плочу се врши праћењем пинова за *GPIO* интерфејс *Raspberry Pi* рачунара, при чему су коришћени само пинови *GPIO* 9 (*SPI_MOSI*), *GPIO* 10 (*SPI_MISO*) и 11 (*SPI_CLK*). Један *Raspberry Pi* треба да се понаша као надређени, док је други подређени.

Међусобно на прототипску плочу 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.



Слика 3.4 Дијаграм повезивања *SPI*.

На дијаграму (слика 3.4) повезивања сонди осцилоскопа са жицама које су повезане између надређеног и подређеног *Raspberry Pi*-а може да се види и како се повезује рачунар са осцилоскопом и *Raspberry Pi*-ом.



Слика 3.5 Приказ екрана осцилоскопа приликом снимања *SPI* сигнала.

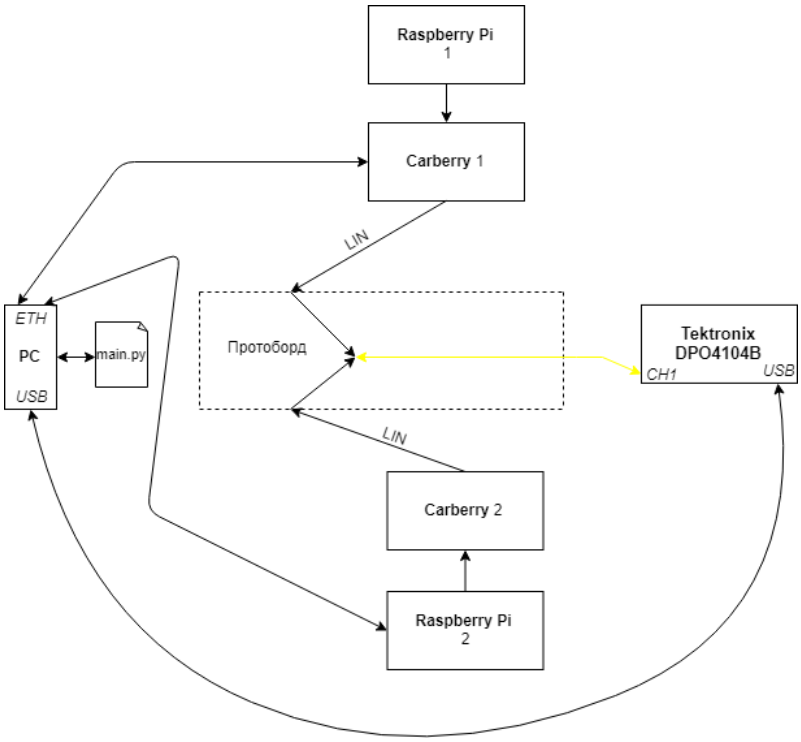
Употребом *SSH* протокола се врши комуникација рачунара са надређеним *Raspberry Pi*-ем. Покретањем програмског решења се генерише секвенца *SPI* сигнал, а

сигнал се снима са осцилоскопа на рачунар. Окидач је подешен на 3 *V* док је хоризонтална скала подешена на 200 *ms*. Вертикална скала сваког од канала је подешена на 1 *V*. Након овог подешавања се могао видети сигнал *SPI* протокола (слика 3.5 – где је љубичастом бојом означен канал на којем се преноси такт, а жутом канал на којем се преноси подаци).

3.4 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.

Carberry плочица је повезана преко 40-пинског *GPIO* интерфејса на *Raspberry Pi* 2 модел Б рачунар. *Carberry* комуницира са *Raspberry Pi* употребом пинова 8 (за слање података), 10 (за пријем података), 12 (за даљинско управљање), 13 (за гашење *Carberry*-ја) на *Raspberry Pi*-у.

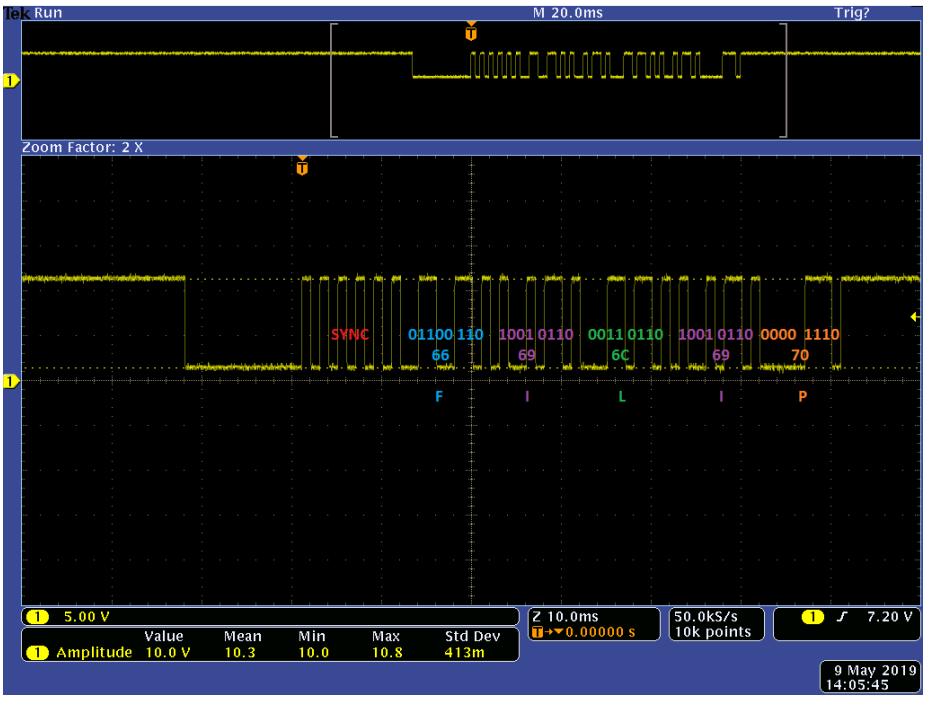


Слика 3.6 Дијаграм повезивања *LIN*-а.

Следећи корак је био повезивање жице са главне утичнице *Carberry* модула на прототипску плочу. Табела 2 описује све жице главне утичнице, где су црвеном бојом означене жице за напајање и уземљење *Carberry*-ја, а зеленом бојом жица за *LIN* (остали каблови нису релевантни тако да њихова улога није описана). Након што су повезана два оваква модула (слика 3.6) на прототипску плочу, комуникација рачунара са сваким од *Carberry* модула је остварена преко серијског порта брзином од 9600 бита у секунди коришћењем *microUSBToUSB* кабла и употребом *Putty* програма. Комуникација је урађена коришћењем команди које су стандардизоване од стране произвођача *Carberry*-ја.

Позиција	**Боја жице**	**Опис**	**Позиција**	**Боја жице**	**Опис**
1	**Црвена**	**+12 *V* Напон**	**12**	**Црвено/Зелена**	**LIN**
2	**Црна**	**Уземљење**	13	Плава	
3	**Плаво/Жута**	**+12 *V* 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 2 Опис жица главне утичнице за *Carberry* модул.



Слика 3.7 Приказ екрана осцилоскопа приликом снимања *LIN* сигнала

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 хоризонтала скала подешена на 4 *ms*, окидач на 3 *V* а вертикална скала сигнала на 5 *V*. Након овог подешавања се могао видети сигнал *LIN* протокола (слика 3.7 – на слици су растуће ивице представљене 1 у бинарном облику – напон око 3 *V,* а напон око 0 *V* представља 0 у бинарном облику – конкретно на овој слици је означено како би требало да изгледа декодован испис сигнала).

3.5 Повезивање *CAN*-а и мерење са осцилоскопом

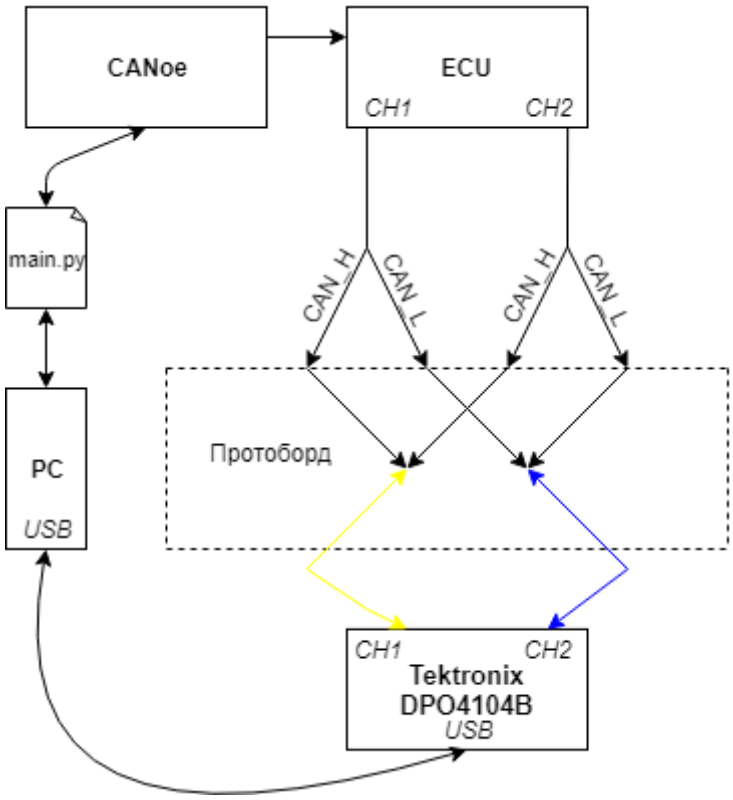
За генерисање сигнала за *CAN* магистралу коришћена је електронска управљачка јединица и алат *CANoe*. *CANoe* алат је напредни алат за развој, тестирање и анализу како појединачних електронских управљачких јединица, тако и комплетне мреже.

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.

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

- Мрежом.
- Управљачком јединицом.
- Променљивама окружења.
- Мрежним чворовима.
- Порукама.
- Сигналима.



Слика 3.8 Дијаграм повезивања *CAN*.

Слика 3.9 Приказ екрана осцилоскопа приликом снимања *CAN* сигнала.

Само креирање ове базе података ће бити детаљније објашњено у програмском решењу (4.6). Након подешавања конфигурације у *CANoe*-у и укључивања направљене базе података, урађено је повезивање електронске управљачке јединице и рачунара *USB* каблом, а затим је покренута симулација док су са друге стране повезане две пробе осцилоскопа на *CAN-H* и *CAN-L* канал чије су жице изведене и прикључене на прототипску плочу (слика 3.8). Окидач је подешен на 3 *V*, хоризонтална скала на 100

ms, док је вертикална скала оба канала подешена на 2 *V*. Након овог подешавања се могао видети сигнал *CAN* протокола (слика 3.9 – жутом бојом је представљен сигнал на *CAN-H* линије док је плавом представљен сигнал на *CAN-L* линији).

4. Програмско решење

Програмско решење се односи на прикупљање, обраду и декодовање сигнала са осцилоскопа. При изради овог програмског решења коришћен је програмски језик Пајтон. Код је писан праћењем *PEP8* [8] конвенције.

Реализација програмског решења је груписана у 14 програмских модула (табела 3). Документација програмског решења је урађена употребом *Doxygen* алата. Комуникација са *Raspberry Pi*-ем употребом *SSH* протокола је урађена коришћењем *paramiko* Пајтон модула.

Назив програмског модула	**Функционалност програмског модула**
main.py	Прикупљање података и обрада сигнала са осцилоскопа.
kmp.py	*KnuthMorrisPratt* алгоритам за претрагу.
csv_everything.py	Снимање сигнала за каснију употребу.
serial_can.py	Серијска комуникација са *Carberry*, слање *CAN* оквира.
serial_lin.py	Серијска комуникација са *Carberry*, слање *LIN* оквира.
ssh_i2c.py	*SSH* комуникација са *Raspberry Pi,* покреће *send_i2c.py*.
ssh_spi.py	*SSH* комуникација са *Raspberry Pi,* покреће *send_spi.py*.
send_i2c.py	Слање *I2C* оквира.
send_spi.py	Слање *SPI* оквира.
can_decoding.py	Декодовање *CAN* оквира.
lin_decoding.py	Декодовање *LIN* оквира.
spi_decoding.py	Декодовање *SPI* оквира.
i2c_decoding.py	Декодовање *I2C* оквира.
checksum.py	Израчунавање контролне суме за *LIN*.

Табела 3 Опис програмских модула.

4.1 Комуникација са осцилоскопом, мерење и обрада сигнала

Комуникација са осцилоскопом је урађена употребом *PyVisa* модула у *main.py*. *PyVisa* омогућава иницијализацију осцилоскопа као објеката класе *ResourceManager*. Функцији *open_resource* овог објекта као параметар се шаље идентификациони број осцилоскопа који се може видети у апликацији *OpenChoiceDesktop*. Размењивање података са осцилоскопом се своди на две методе класе – *write()* и *ask()*. Праћењем приручника за програмере од *Tektronix*-а [9] за овај модел осцилоскопа послате су команде као параметар једној од две претходно описане методе, у зависности од намене (да ли се жели подешавање осцилоскопа или прикупљање података). Послате су команде за одабир канала (`DATA:SOU CH`), постављање хоризонталне (`HORizontal:SCAle`) и вертикалне скале (`VERtical:SCAle`), а потом за прикупљање података (`DATA:WIDTH 1`, `DATA:ENC RPB`, `WFMPRE:YMULT?`, `WFMPRE:YZERO?`, `WFMPRE:YOFF?`, `WFMPRE:XINCR?`, `CURVE?`) након сваке детекције окидача (`TRIGger:A SETLevel`, `ACQ:STATE ON`, `ACQ:STATE?`). Ови снимљени подаци (време и напон за сваки од изабраних канала) се прослеђују објекту класе *Signal* као аргументи. За сваки канал који је потребан за одређени протокол, се посебно иницијализује објекат класе *Signal*.

Унутар класе *Signal* се налазе методе:

- **plotting_1_ch** – Ова метода служи за графички приказ сигнала који има 1 канал (нема одвојен канал за податке и такт, већ само канал за податке).
- **plotting_2_ch** – Ова метода служи за графички приказ сигнала који има 2 канала (одвојен канал за податке и такт).
- **level_out_signal** – Ова метода служи за постављање сваког од одбирка сигнала на 1 или 0 у зависности од тога да ли прелази одређену вредност напона која представља окидач. Ова обрада сигнала се врши ради лакшег декодовања сигнала.

Унутар *main.py* се налазе функције:

- **main** – Ова функција се користи за слање команди осцилоскопу за подешавање хоризонталне и вертикалне скале, као и за прикупљање напона и времена за сваки од одабраних канала.
- **set_channel** – Ова функција се користи за постављање канала осцилоскопа на којима се прате сигнали.

- **can_processing** – Ова функција се користи за рекурзивну проверу размака између сваког од оквира *CAN* сигнала. За ову проверу коришћена је *Knuth-Morris-Pratt* алгоритам која се налази унутар *kmp.py*. Ову проверу је било потребно урадити због могућих интерференција које се јављају приликом снимања сигнала осцилоскопом. Ова функција даље позива **can_decoded** функцију за декодовање која се налази унутар *can_decoding.py*.
- **lin_processing** – Ова функција позива **lin_decoded** функцију која се налази унутар *lin_decoding.py* и проверава исправност измерене контролне суме и измереног паритета.
- **i2c_online_processing** – Ова функција снима сигнал директно са осцилоскопа, потом га обрађује и позива **i2c_decoded** функцију за декодовање која се налази унутар *i2c_decoding.py*.
- **i2c_offline_processing** – Ова функција снима сигнал из *i2c-capture.csv*[3] датотеке, потом га обрађује и позива **i2c_decoded** функцију за декодовање која се налази унутар *i2c_decoding.py*.
- **spi_online_processing** – Ова функција снима сигнал директно са осцилоскопа, потом га обрађује и позива **spi_decoded** функцију за декодовање која се налази унутар *spi_decoding.py*.
- **spi_offline_processing** – Ова функција снима сигнал из *spi-capture.csv* датотеке, потом га обрађује и позива **spi_decoded** функцију за декодовање која се налази унутар *spi_decoding.py*.

4.2 *Knuth-Morris-Pratt* алгоритам

Knuth-Morris-Pratt [10] алгоритам за претрагу низова тражи појаву обрасца унутар текста, али када дође до неслагања образац се помера за неки оптималан број карактера, тако заобилазећи преиспитивање раније усклађених карактера. Следећи пример ће појаснити функционисање овог алгоритма:

T=	T_i	T_{i+1}	T_{i+2}	T_{i+3}	T_{i+4}	T_{i+5}	T_{i+6}	T_{i+7}
O=			O_0	O_1	O_2	O_3	O_4	O_5
			√	√	√	√	X	

У примеру смо утврдили да је карактер O_0 обрасца поравнат са карактером T_{i+2} текста, карактер обрасца O_1 са карактером T_{i+3} текста, итд. На позицији T_{i+6} текста смо уочили неслагање. Како се O_0 слаже са T_{i+2}, O_1 са T_{i+3}, итд., бесмислено је померати

[3] Енгл. *comma-separated values* – представља текстуалну датотеку са граничницима, смештају се табеларни подаци у отворен текст.

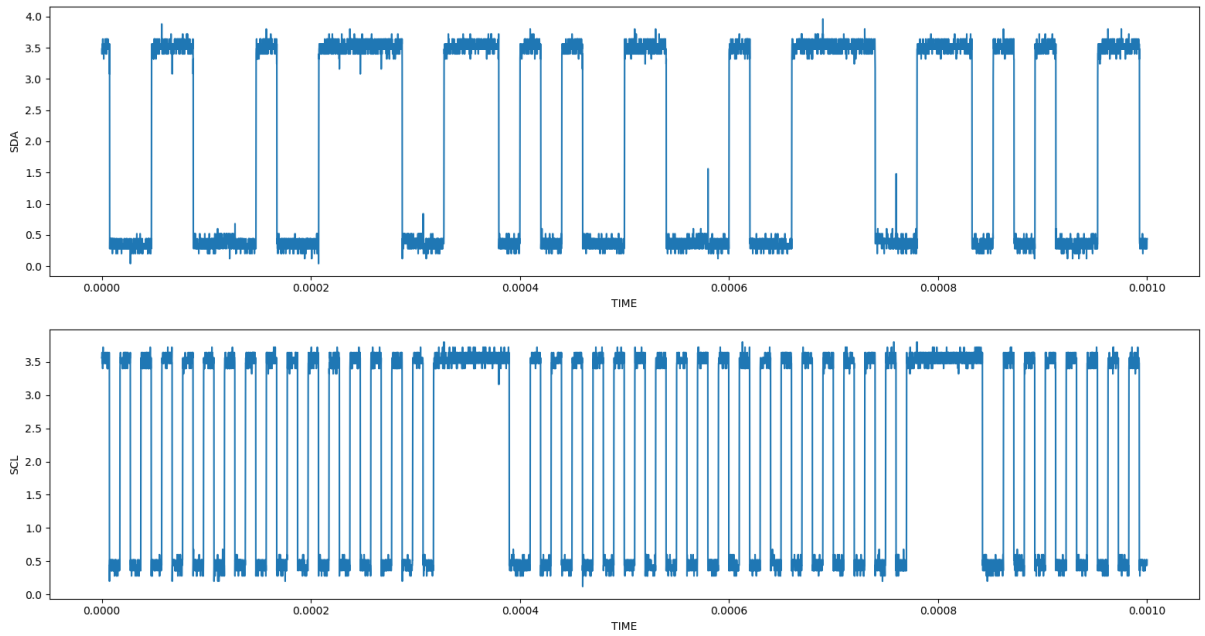
образац за једну позицију у односу на текст јер ни тада не може да дође до слагања. Ако у обрасцу нема понављања одмах треба прећи на следећу позицију:

T=	T_i	T_{i+1}	T_{i+2}	T_{i+3}	T_{i+4}	T_{i+5}	T_{i+6}	T_{i+7}
O=			O_0	O_1	O_2	O_3	O_4	O_5
			√	√	√	√	X	
							O_0	O_1

Овај поступак се понавља све док се не дође до крајње позиције текста над којим се врши претрага. Посебна предност овакве процедуре је што она ради са текстом идући с лева у десно и униформно врши померање обрасца, а може и паралелно да проверава више образаца. У најгорем случају број поређења је линеаран.

4.3 Декодовање *I²C*-а

Декодовање *I²C* сигнала је урађено у *i2c_decoding.py* у функцији **i2c_decoded**. На почетку се врши претрага почетне секвенце за сваки од канала (*SDA*, *SCL*) – почетна секвенца канала за пренос радног такта је за један бит већа. За претрагу ове две секвенце је коришћен *Knuth-Morris-Pratt* алгоритам. Слика 4.1 приказује пример снимљеног *I²C* сигнала са осцилоскопа употребом програмског решења – свако узорковање које је око 0 *V* представља 0 у бинарном облику, а напон који је око 3 *V* (напоном изнад претходног постављеног окидача) представља 1 у бинарном облику. Након што су сигнали снимљени узоркују се на вредности временског интервала које смо послали осцилоскопу. Потом се узорковање података са *SDA* линије врши на растућој ивици такта са *SCL* линије. Подаци се читају у *MSB*-у. Први бит представља почетак оквира – почетна секвенца, наредних 7 бита представља адресу уређаја са ког се шаљу подаци и након бита потврде подређеног уређаја, потом долази 8-битни садржај који представља податке које се преносе на магистрали и 1 бит потврде од руководиоца (могуће је слати више оваквих пакета за податке унутар једног оквира поруке). Након што су издвојени корисни подаци, исти се представљају у хексадецималном формату.

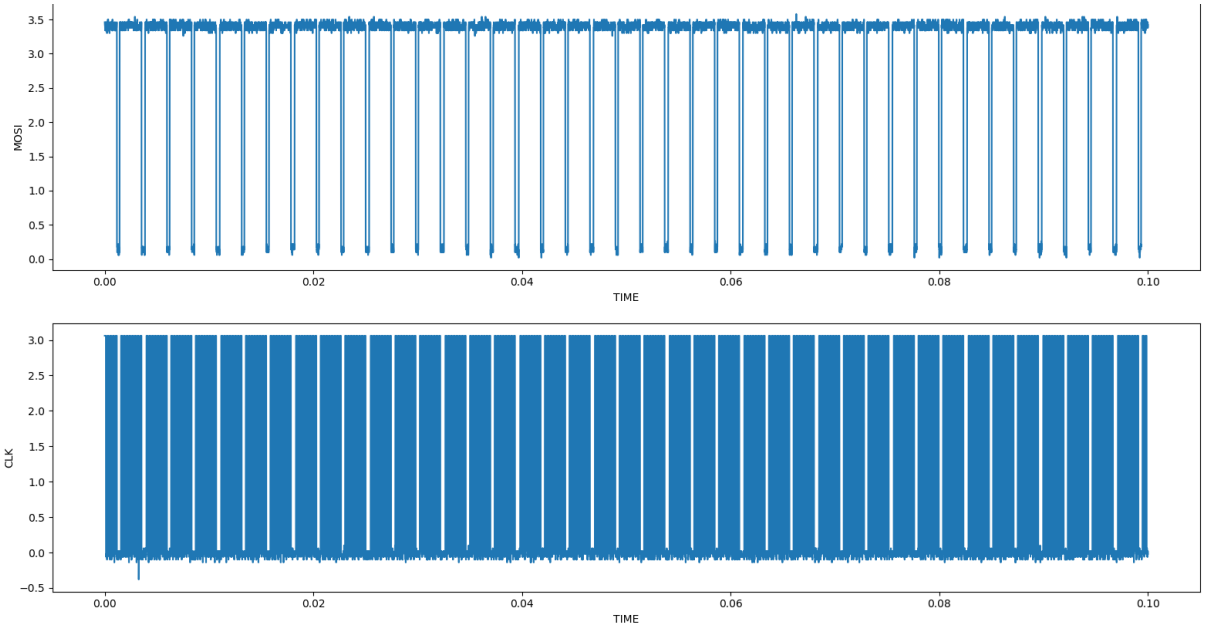


Слика 4.1 Пример необрађеног I^2C сигнала.

4.4 Декодовање *SPI*-а

Декодовање је урађено у *spi_decoding.py* у функцији **spi_decoded**. Слика 4.2 приказује пример снимљеног *SPI* сигнала са осцилоскопа употребом програмског решења. Након што су сигнали снимљени узоркују се на вредности временског интервала које смо послали осцилоскопу. Подаци се читају у *MSB*-у као да су бинарни бројеви – свако узорковање које је око 0 *V* представља 0 у бинарном облику, а напон који је око 3 *V* (напоном изнад претходног постављеног окидача) представља 1 у бинарном облику. Узорковање података са *SPI_MOSI* или *SPI_MISO* линије се потом врши на растућој ивици такта са *SPI_CLK* линије у случају да је *CPHA* 0, док се узоркују на опадајућој ивици у случају да је *CPHA* 1. Свака растућа (улазна) ивица сигнала такта представља да је пренос података активан, дужи периоди мирног сигнала (0 *V*) представља стање да се тренутно не шаљу подаци. Да би се препознали подаци који треба да се декодују, прво су се морали издвојити корисни подаци на *SPI_MOSI* линији који се шаљу у исто време када се појави секвенца од 8 растућих ивица на *SPI_CLK* линији. За претрагу секвенце на *SPI_CLK* линији се користи *Knuth-Morris-Pratt* алгоритам. Уколико секвенца није правилно пронађена (ако није правилно подешена хоризонтална или вертикална скала) добиће се неисправни подаци. Уколико је секвенца пронађена потребно је посматрати податке који имају дужину ове секвенце. Корисне податке можемо да издвојимо тек након што је утврђено да је дужина секвенце правилна. Пошто унутар сваког интервала слања има 8 растућих ивица такта

то значи да је дужина података унутар једног интервала слања максимално осмобитна. Након што су издвојени корисни подаци, исти се представљају у хексадецималном формату.



Слика 4.2 Пример необрађеног *SPI* сигнала.

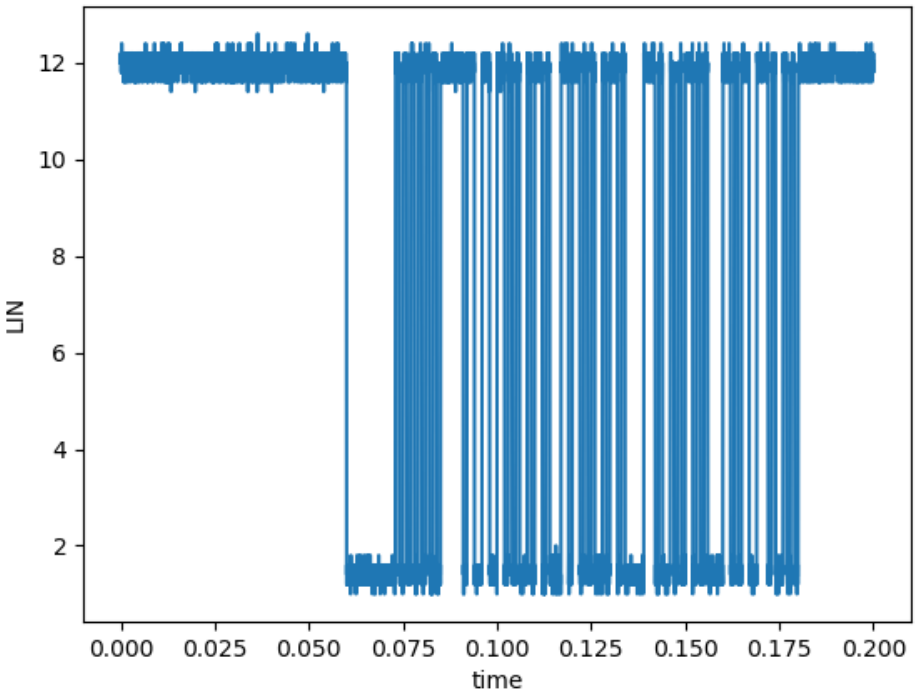
4.5 Декодовање *LIN*-а

Као што је наведено у концепту решења, у овом поглављу биће дато објашњење стандардних команди за *Carberry* за *LIN* протокол. Прво, преко серијског порта, потребно је успоставити везу за сваки од *Carberry* модула коришћењем *LIN OPEN* команде. Постоји пет различитих режима рада које су подржане од стране *Carberry*-ја (табела 4).

Мод	Команда	Опис команде
Master 1x	*LIN OPEN MASTER1X* <брзина преноса>	Отвара канал при чему аутоматски препознаје брзину на магистрали, уређај се понаша као надређени, користи се стандардна контролна сума.
Master 2x	*LIN OPEN MASTER2X* <брзина преноса>	Отвара канал при чему аутоматски препознаје брзину, уређај се понаша као надређени, користи се продужена контролна сума.
Slave 1x	*LIN OPEN SLAVE1X*	Отвара канал при чему аутоматски препознаје брзину, уређај се понаша као подређени, користи се стандардна контролна сума.
Slave 2x	*LIN OPEN SLAVE2X*	Отвара канал при чему аутоматски препознаје брзину, уређај се понаша као подређени, користи се продужена контролна сума.
Free	*LIN OPEN FREE* <брзина преноса>	Отвара канал и задаје брзину; за паритет и контролну суму је одговоран корисник.

Табела 4 *Carberry* команде за *LIN*.

У изради рада коришћен је последњи, „слободни“ режим за слање пакета, због флексибилности. Након слања сваке команде, уколико је исправна, добија се одговор *OK*. Слање *LIN* оквира у слободном режиму се ради употребљавањем *LIN TX* <заштићени идентификатор> [хексадецимални приказ података] [контролна сума] команде.



Слика 4.3 Приказ необрађеног *LIN* сигнала.

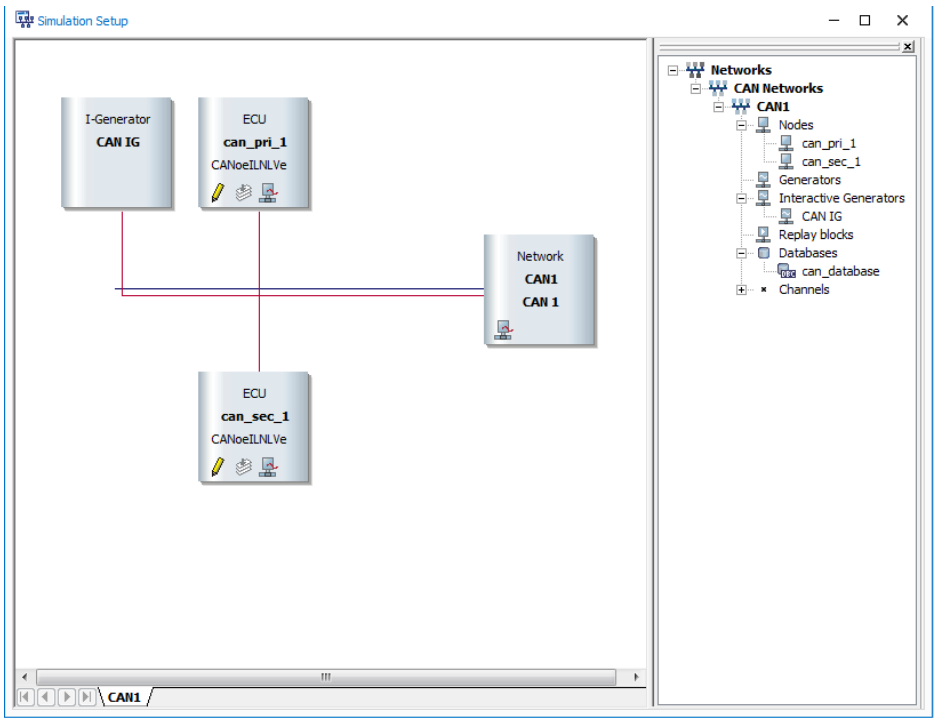
Слика 4.3 приказује пример снимљеног *LIN* сигнала са осцилоскопа употребом програмског решења. Након што је сигнал снимљен узоркује се на вредност временског интервала које смо послали осцилоскопу – свако узорковање које је око 0 *V* представља 0 у бинарном облику, а напон који је око 3 *V* (напоном изнад претходног постављеног окидача) представља 1 у бинарном облику. Почетак самог пакета се представља у облику првих 13 бита са вредношћу 0, бити од 15. до 22. позиције представљају синхронизационо поље (2 бита између овог поља и претходног представљају 2 бита са вредношћу 0 који означавају размак између поља и овај размак ће се појављивати након сваког поља, тако да надаље неће бити спомињан, већ ће се подразумевати његово присуство), потом долази идентификационо поље – 25. до 33. бит, након овог поља стижу поље са подацима (величине 2, 4 или 8 бита) и 8-битно поље са контролном сумом. Сва ова издвојена поља се потом представљају у хексадецималном облику.

Декодовање је урађено у *lin_decoding.py* у функцији **lin decoded**. Да би се проверило да ли је контролна сума исправна, израчуната је праћењем упутства из *LIN* спецификације. Потом се пореди да ли израчуната контролна сума одговара измереној

контролној суми у оквиру. Такође се мора проверити валидност паритета 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.

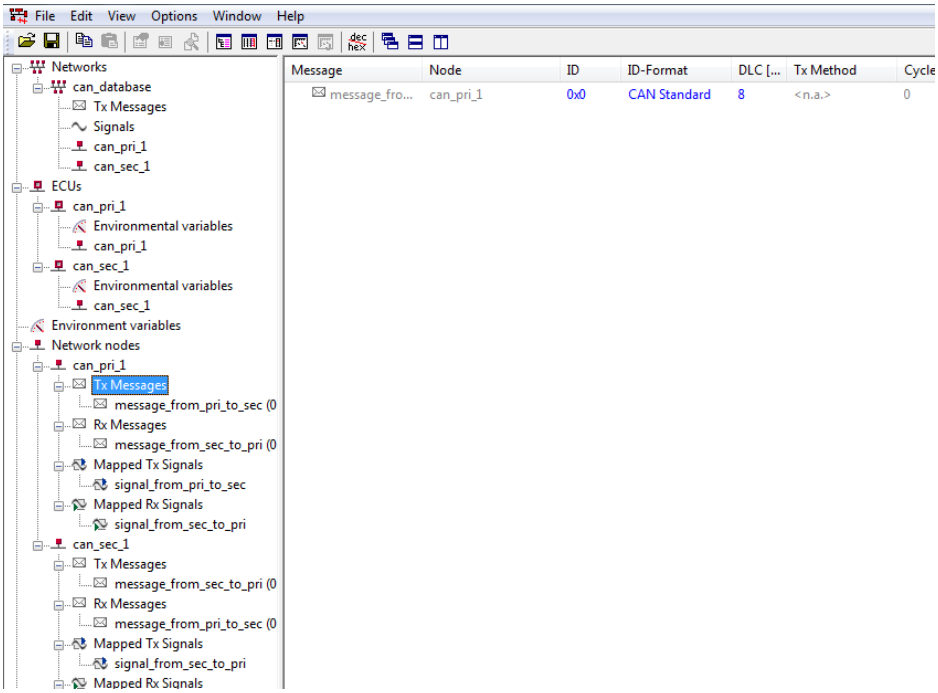
4.6 Декодовање *CAN*-а

У *CANoe* је прво креирана одговарајућа конфигурација и сачувана у конфигурационој датотеци са наставком *.cfg*. Потом је у *CANoe Options* менију под *Measurment* опцијом, и у *General* прозору *Channel Usage,* подешен одговарајући број *CAN* канала (2). У *Simulation Setup* (у овом прозору је укупан систем приказан графички са мрежама, уређајима, и свим мрежним чворовима) прозору (слика 4.4) су потом креирана два *ECU* чвора и један *I-Generator* (користи се за генерацију *CAN* сигнала) који су повезани на претходно направљену *CAN* мрежу. У менију *Hardware* под опцијом *Network Hardware* је намештена одговарајућа брзина преноса која мора да одговара брзини преноса 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++ Editor*-у (слика 4.5), унутар које су креирана 2 чвора (примарни и секундарни).



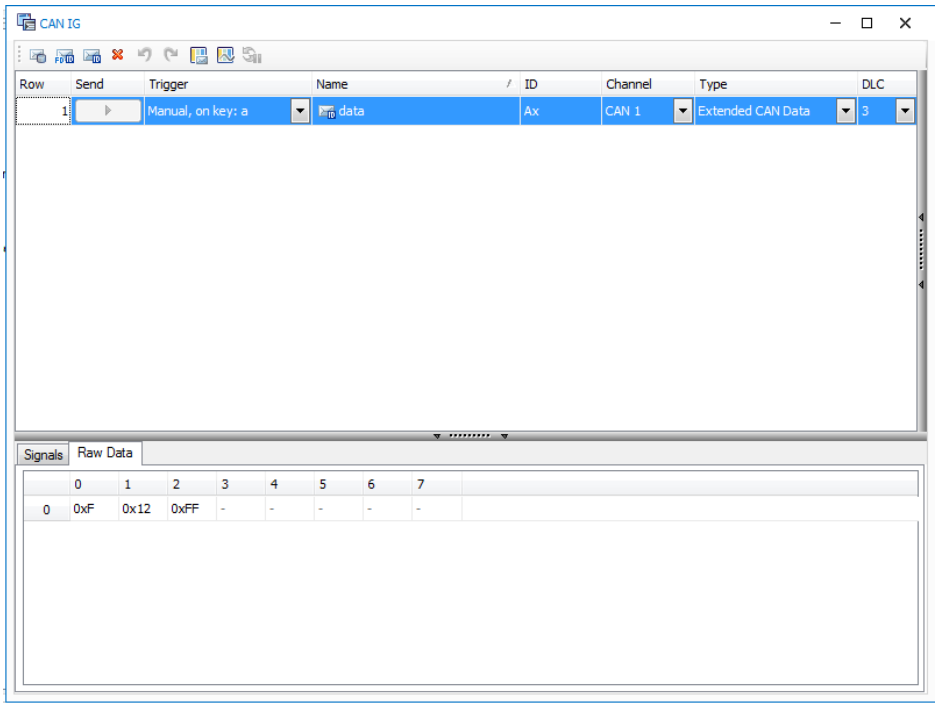
Слика 4.4 Приказ *Simulation Setup*-а прозора унутар *CANoe*-а.

Након тога су креирана два сигнала, при чему је један резервисан за слање података од примарног чвора до секундарног, а други за слање обрнутим смером. Затим су ови сигнали мапирани за одговарајућу поруку која је претходно креирана. На крају су ове поруке мапиране на сваки од одговарајућих чворова, чиме се завршава креирање базе података.



Слика 4.5 Приказ *Canoe CANdb++ Editor*-а.

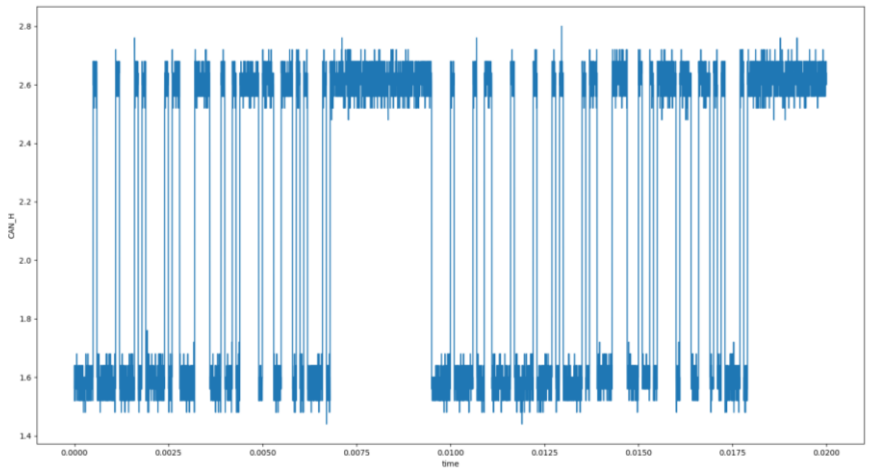
Унутар *I-Generator*-а (слика 4.6) су креирани сигнали који ће се слати од примарног до секундарног чвора. За прављење сигнала мора се одабрати канал, идентификатор, тип сигнала (да ли је са стандардним идентификатором или продуженим), дужина података, а унутар *Raw Data* прозора се конфигурише сваки од октета за слање.



Слика 4.6 Приказ *I-Generator* прозора.

Пре самог покретања конфигурације, мора се потврдити да унутар *Home* прозора да је конфигурација подешена у *Online Mode* (када се подаци са тренутног мерења користе – хардверски или симулацијом, за разлику када се користе подаци из једног од фајлова за евидентирање) и да се користи *Real Bus* (за разлику од симулиране магистрале), пошто ће се сигнал генерисати употребом електронске управљачке јединице. Праћење сигнала у времену се врши унутар *Data* прозора из *Analysis* менија.

Декодовање *CAN* сигнала урађено је у *can_decoding.py* у функцији **can_decoded** поштујући *CAN 2.0* протокол. Стања прикупљеног сигнала се представљају у облику 1 и 0, при чему су сви делови сигнала са напоном изнад претходног постављеног окидача (*3V*) постављени на 1, док су остали постављени на 0. Након тога су избачени из сигнала сви убачени и лавински бити. Употребом *Knuth-Morris-Pratt* алгоритма се препознавају лавински бити и 5 истих доминантних или рецесивних бита. Тек након овога се могло наставити са декодовањем при чему се гледа бит 13 сигнала који указује на то да ли је коришћен *CAN 2.0 A* или *CAN 2.0 B*.



Слика 4.7 Приказ ухваћеног необрађеног *CAN* сигнала са осцилоскопа

Слика 4.7 приказује пример снимљеног *CAN* сигнала са осцилоскопа употребом програмског решења. Након што је сигнал снимљен узоркује се на вредност временског интервала које смо послали осцилоскопу. Потом се издвајају сви релевантни оквири, врши се конверзија добијених бинарних бројева у хексадецималну вредност и израчунава се циклична провера редудансе. Ако се ова израчуната провера поклапа са цикличном провером редудансе издвојеном из сигнала, главном програму се прослеђује декодован *CAN* сигнал са назнаком да је провера тачна, у супротном, прослеђује се декодован *CAN* сигнал са назнаком да провера није тачна.

5. Тестирање и резултати

Циљ тестирања је провера тачности имплементације програмског решења. Тестирање програмског решења је започето повезивањем сваког од протокола као у концепту решења (слика 3.2, слика 3.4, слика 3.6, слика 3.8), а потом генерисањем оквира за сваки од протокола на следећи начин:

- Генерисање I^2C оквира урађено у *sendI2C.py* (употребом Пајтон модула *smbus* – `smbus.SMBus(1).write_byte(adresa, podaci)`).
- Генерисање *SPI* оквира је урађено у *sendSPI.py* (употребом Пајтон модула *spidev* – `spidev.SpiDev().open(magistrala, adresa)` потом `spi.xfer(podaci)`).
- Генерисање *LIN* оквира је урађено у *serial_lin.py* (употребом *serial* Пајтон модула – `serial.Serial(ulaz, brzina prenosa).write(podaci)`).
- Генерисање *CAN* оквира је урађено у *CANoe*-у на начин описан у претходном поглављу (4.6).

Након што су генерисани оквири, покретањем апликације за декодовање, потребно је као аргумент унети тип протокола који жели да се декодује (у случају *LIN*-а, потребно је унети и коју врсту контролне суме желимо да декодујемо *c* – за класичну контролну суму, *e* – за побољшану контролну суму), као и режим у којем се извршава (*online* – када се жели очитавање директно са осцилоскопа или *offline* – када се жели очитавање из једне од датотека са наставком *.csv* – име је креирано у зависности од тога који се протокол користи, чији се садржај ствара при сваком покретању апликације у *online* режиму употребом једне од 4 функција унутар *csv_everything.py*). Потом се упоредном методологијом (поређењем излаза програмског решења и генерисаног

сигнала) утврдила исправност имплементираног декодера. У случају *CAN* и *LIN* протокола, поређењем декодоване контролне суме снимљене са осцилоскопа и израчунате контролне суме се додатно гарантује исправност имплементираног програмског решења. Код I^2C-а и *SPI*-а исправност програмског решења се гарантује синхронизацијом бита података са битима такта – уколико очитавање бита података није на узорковању бита такта када је активно стање преноса података, подаци неће бити добро очитани са осцилоскопа а самим тим неће се моћи ни исправно декодовати. Тестирање је обављено са великом количином различитих сигнала (различити оквири како у варијацији садржаја оквира идентификатора и оквира података тако и у величини оквира података), при чему се посебно повело рачуна да се шаљу оквири за тестирање механизма за детекцију „лавинских“ битова у случају декодовања *CAN* сигнала. И у овом случају је тестирање успешно извршено.

Слика 5.1 представља пример успешног декодовања I^2C сигнала, слика 5.2 представља пример успешног декодовања *SPI* сигнала, слика 5.3 представља пример излаза успешног декодовања *LIN* сигнала, а слика 5.4 представља пример излаза успешног декодовања *CAN* сигнала, на сликама су зеленом бојом означени исписи успешно декодованих оквира сигнала који су нам битни за препознавање исправности имплементираног решења (само подаци у случају I^2C-а и *SPI*-а, а за преостала 2 протокола – идентификатор, дужина, подаци и контролна сума), у случају да верзија контролне суме унета као један од аргумената програмског решења не одговара контролној суми која се користила при генерисању *LIN* сигнала, тај испис би био црвене боје означавајући неуспешно декодовање.

```
C:\Users\jasic\Documents\bsc\project>main.py I2C offline
press Ctrl+C to stop measurment
if red color shows up,data is not correct
else green it's all good!
decoded i2c data: ['4F']
```

Слика 5.1 Пример излаза успешног декодовања I^2C-а.

```
C:\Users\jasic\Documents\bsc\project>main.py SPI offline
press Ctrl+C to stop measurment
if red color shows up,data is not correct
else green it's all good!
data: ['FF', 'FF', 'FF', 'FF', 'FF', 'FF', 'FF', 'FF', 'F
F', 'FF', 'FF', 'FF', 'FF', 'FF', 'FF']
```

Слика 5.2 Пример излаза успешног декодовања *SPI*-а.

```
C:\Users\fjasic\Desktop\diplomski git\oscilloscope-bachelor\project>main.py LIN -e
press Ctrl+C to stop measurment
average voltage for CH1:8.511360000000025
scale for CH1:0.02
LIN - CSV output done
ID: BF||PARITY BITS: 10||DATA: ['66', '69', '6C', '69', '70', '6A', '61', '73']||CHECKSUM: EA
```

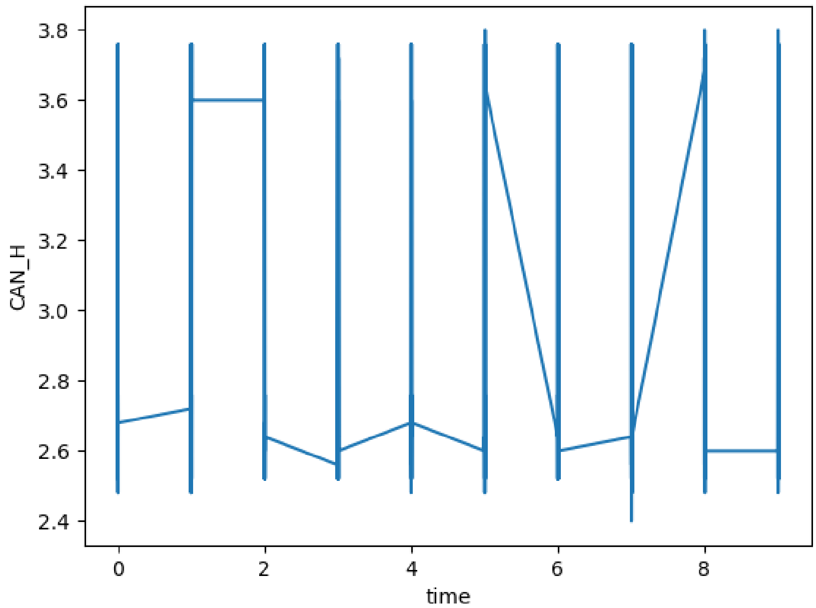
Слика 5.3 Пример излаза успешног декодовања *LIN*-а.

```
C:\Users\fjasic\Desktop\diplomski git\oscilloscope-bachelor\project>main.py CAN
press Ctrl+C to stop measurment
if red color shows up,data is not correct
else green it's all good!
. . . . . . . .

CAN - CSV output done
id(standard): 01 ||length: 4 ||data: ['11', '12', '13', '14'] ||crc: [1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0]
id(standard): 01 ||length: 4 ||data: ['11', '12', '13', '14'] ||crc: [1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0]
id(standard): 01 ||length: 4 ||data: ['11', '12', '13', '14'] ||crc: [1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0]
id(standard): 01 ||length: 4 ||data: ['11', '12', '13', '14'] ||crc: [1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0]
id(standard): 01 ||length: 4 ||data: ['11', '12', '13', '14'] ||crc: [1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0]
id(standard): 01 ||length: 4 ||data: ['11', '12', '13', '14'] ||crc: [1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0]
id(standard): 01 ||length: 4 ||data: ['11', '12', '13', '14'] ||crc: [1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0]
id(standard): 01 ||length: 4 ||data: ['11', '12', '13', '14'] ||crc: [1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0]
id(standard): 01 ||length: 4 ||data: ['11', '12', '13', '14'] ||crc: [1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0]
id(standard): 01 ||length: 4 ||data: ['11', '12', '13', '14'] ||crc: [1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0]
```

Слика 5.4 Пример излаза успешног декодовања *CAN*-а.

Потешкоће приликом израде су настале при снимању података, јер при континуалном прикупљању података долази до кашњења између два оквира. Ово је проузроковано коришћењем програмског језика Пајтона који, иако олакшава имплементацију програмског решења, уводи проблеме кашњења и нетачности приликом мерења. Кашњење између оквира је око једне секунде (слика 5.5 – вертикалне црте представљају по један *CAN* оквир, а остале црте на графику представљају кашњење). Овај проблем је решен очекивањем овог кашњења и „померањем“ сигнала на временског оси за вредност кашњења.



Слика 5.5 Кашњење између *CAN* оквира.

Такође, проблем је настао при очитавању сигнала протокола који је више канални – I^2C и *SPI. CAN* иако има 2 канала, та два канала су фактички негативи један другог (инверзија сигнала једног од канала даје сигнал другог канала), тако да овај проблем се није појавио код овог протокола као ни код *LIN*-а. У једном тренутку израде овог дипломског рада пробан је паралелизам слања команди употребом Пајтон модула *multiprocessing*. Ово се показало неуспешним због немогућности контролисања редоследа слања команди. Команде послате једном каналу могле су за своје одредиште да имају други канал, пошто тип осцилоскопа коришћен у овом раду не подржава индивидуално слање команди за сваки од канала у реалном времену. Због немогућности паралелног слања команди осцилоскопу употребом *API*-ја (команде су објашњене у програмском решењу – 4.1) за сваки од канала, решавање проблема је урађено наизменичним смењивањем канала са којима се комуницира и на који се шаљу команде за постављање хоризонталне и вертикалне скале, ниво окидача, прикупљање снимљених података. Са овим се јавио проблем кашњења између 2 сигнала. Поготово што се овај проблем јавио на протоколима код којих је важна међусобна синхронизација сигнала такта и података (да би се правило очитали послати подаци), било је веома важно да се он реши. То је урађено мерењем кашњења и потом „померањем“ сигнала канала на који се тренутно шаљу команде (након комуникације са првим каналом, обавља се комуникација са другим каналом) на временској оси за измерену вредност кашњења.

6. Закључак

У овом раду је реализовано једно решење декодовања *CAN, LIN, SPI* и I^2C оквира. У ту сврху било је неопходно прво изучити начин функционисања сваког од поменутих протокола, па потом их измерити осцилоскопом. Да би се осцилоскоп правилно употребљавао морало се приступити изучавању ручног подешавања параметара за спецификацију сваког од протокола да би се сигнали успешно снимили. Након тога је започето повезивање осцилоскопа, генерисање, мерење и разумевање измереног сигнала. Потом се приступило креирању програмског решења коришћењем Пајтон програмског језика, праћењем документације од стране произвођача осцилоскопа коришћеног у овом раду.

Резултати показују да је успешно реализовано програмско решење за сва 4 протокола.

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 сигнала аутоматски урађено.

Даљи развој ће се фокусирати на реализацију аутоматске детекције коришћеног протокола и реализацију аутоматске детекције коришћене врсте контролне суме за *LIN* протокол.

7. Литература

[1] "Basics of working with I2C buses", 2018. [Online]. Доступно: https://www.testandmeasurementtips.com/basics-working-i2c-buses [приступљено: јул 2019.]

[2] "Analyzing the Serial Peripheral Interface", 2017. [Online]. Доступно: https://www.testandmeasurementtips.com/analyzing-serial-peripheral-interface-spi-bus [приступљено: јул 2019.]

[3] "LIN Specification Package Revision 2.0", 2003. [Online]. Доступно: https://forums.ni.com/attachments/ni/30/3619/1/LIN.pdf [приступљено: јун 2019.]

[4] "BOSCH CAN Specification Version 2.0", 1991. [Online]. Доступно: https://www.academia.edu/9998014/BOSCH_CAN_Specification [приступљено: јун 2019.]

[5] ISO 11898-1, "Road vehicles — Controller area network (CAN) — Part 1:
Data link layer and physical signalling", First edition, Geneva, Switzerland, ISO, 2003.

[6] MSO4000B and DPO4000B Series Digital Phosphor Oscilloscopes User Manual. [Online] Доступно: https://download.tek.com/manual/071281002web.pdf [приступљено: јун 2019.]

[7] "Testing and debugging the I2C digital data bus". [Online]. Доступно: https://www.testandmeasurementtips.com/testing-and-debugging-the-i2c-digital-data-bus [приступљено: јул 2019.]

[8] "PEP 8 -- Style Guide for Python Code", 2001. [Online]. Доступно: https://www.python.org/dev/peps/pep-0008 [приступљено: август 2019.]

[9] MSO4000 and DPO4000 Series Digital Phosphor Oscilloscopes Programmer Manual. [Online] Доступно: https://download.tek.com/manual/077024801web.pdf [приступљено: јун 2019.]

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, "Introduction to Algorithms", Third Edition, The MIT Press, London, England, 2010.