УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Драган Радановић

Једно решење програмске подршке за трајно чување података у аутомобилској индустрији

МАСТЕР РАД

Нови Сад, 2017

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**:		Дипломски – мастер рад	
Аутор, **АУ**:		Драган Радановић	
Ментор, **МН**:		Доц. др Иван Каштелан	
Наслов рада, **НР**:		Једно решење програмске подршке за трајно чување података у аутомобилској индустрији	
Језик публикације, **ЈП**:		Српски / латиница	
Језик извода, **ЈИ**:		Српски	
Земља публиковања, **ЗП**:		Република Србија	
Уже географско подручје, **УГП**:		Војводина	
Година, **ГО**:		2016	
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)		8/53/30/11/6/0/0	
Научна област, **НО**:		Електротехника и рачунарство	
Научна дисциплина, **НД**:		Рачунарска техника	
Предметна одредница/Кључне речи, **ПО**:		Трајно чување меморије,	
УДК			
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:			
Извод, **ИЗ**:		У овом раду је приказано једно решење програмске подршке за трајно чување садржаја меморије у аутомобилској индустрији, развијено по AUTOSAR и ISO26262 стандардима.	
Датум прихватања теме, **ДП**:			
Датум одбране, **ДО**:			
Чланови комисије, **КО**:	Председник:	Др. Илија Башичевић	
	Члан:	Др. Платон Совиљ	Потпис ментора
	Члан, ментор:	Доц. др Иван Каштелан	

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**:		Master Thesis	
Author, **AU**:		Dragan Radanović	
Mentor, **MN**:		Doc. PhD Ivan Kaštelan	
Title, **TI**:		One solution for persistent data storage in automotive industry	
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**:		2016	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		8/53/30/11/6/0/0	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S/KW**:		Persistent data storage	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		This paper describes one solution for permanent data storage in automotive industry, developed according to AUTOSAR and ISO 26262 standards.	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	Phd. Ilija Bašičević	
	Member:	Phd. Platon Sovilj	Menthor's sign
	Member, Mentor:	Doc. PhD Ivan Kaštelan	

Zahvalnost

Srdačno se zahvaljujem svom mentoru doc. dr Ivanu Kaštlanu na stručnoj pomoći, razumevanju i savetovanju tokom izrade master rada.

Takodje se zahvaljujem asistentu dr. Zeljku Lukaču na ukazanoj pomoći, stalnoj podršci, i razumevanju tokom izrade MASTER rada.

Posebno se zahvaljujem rukovodstvu firme RT-RK na ukazanoj prilici da se bolje upoznam sa načinom rada u inženjerskom okruženju i budem uključen u proces razvoja novih programskih rešenja.

Na kraju se zahvaljujem svima onima koji su na bilo koji način doprineli izradi ovog master rada.

SADRŽAJ

SPISAK SLIKA

SPISAK TABELA

SKRAĆENICE

ADAS - *Advanced driver assistance systems,* Napredni sistemi za pomć vozaču

NVM - *NonVolotile Memory***,** Memorija sa stalnim sadržajem

RAM - *Random Access Memory,* Memorije sa nasumičnim pristupom

ROM - *Read Only Memory,* Memorija samo za čitanje

CRC - *Cyclic redundancy check,* Ciklična redudantna provera

NAND - Logička operacija inverzne konjukcije

NOR - Logička operacija inverzne disjunkcije

OEM - *Original Equipment Manufacturer*, Orginalni proizvođač opreme

ECU - *Electronic control unit,* Elektronska kontrolna jedinica

VFB - Virtual Functional Bus, Virtuelna funkcionalna magistrala

SW-C - *SoftWare Component,* Komponenta programske podrške, aplikacija

BSW - *Basic SoftWare,* Osnovna programska podrška

RTE - *RunTime Enviroment,* Okruženje izvršavanja

IDE - *Integrated development environment,* Integrisano razvojno okruženje

CAN - *Controller Area Network,* Mrežni kontroler

MISRA - *Motor Industry Software Reliability Association,* Asocijacija pouzdanosti programske podrške industrije motora

HIS - *Herstellerinitiative Software,* Inicijativa proizvođača programske podrške

XML - *eXtensible Markup Language*, Proširivi jezik za označavanje

XSD - *XML Schema Definition*, Definicija XML šeme

IEC - *International Electrotechnical Commission,* Međunarodna elektrotehnička komisija

ISO - *International Organization for Standardization,* Međunarodna organizacija za standarde

MDA **-** *Model Driven Engineering,* Inženjerstvo upravljano modelima

MDSD **-** *Model-driven software development,* Razvoj programske podrške zasnovan na modelu

ASIL - *Automotive Software Integrity Level,* Nivo Integriteta Automotiv programske podrške

QM - *Quality Management,* Upravljanje kvalitetom

API - *Application programming interface,* Programski interfejs za aplikacije

SSD **-** *Solid State Drive,* Poluprovodnički disk

BBM - *Bad Block Management,* Upravljanje neispravnim blokovima

POSIX **-** *Portable Operating System Interface,* Interfejs za prenosive operativne sisteme

CSV **-** *Comma Separated Values,* Vrednosti razdvojene zarezom

1. Uvod

Kompanije koje se bave proizvodnjom automobilske opreme i vozila imaju stalnu želju za inovacijom novih, kao i unapređenje već postojećih usluga koje nude u svojim vozilima i uređajima. Konstantno unapređenje programske podrške i fizičkih arhitektura računara i sistema zasnovanih na računaru omogućava razvoj novih funkcionalnosti, različitih nivoa kompleksnosti, od kontrole svetla u kabini, preko elektronske kontrole stabilnosti, pa sve do autopilota, i autonomne vožnje. Savremeni računarski sistemi koji se koriste u automobilskim sistemima, po performansama, procesorskoj snazi ne zaostaju za savremenim PC računarima.

Modernim automobilima se postavljaju visoki zahtevi vezani za sigurnost, ekonomičnost i udobnost [1].

Razvoj programske podrške za automobilsku industriju se veoma razlikuje od razvoja za potrošačku industriju. Nepouzdan rad programske podrške u potrošačkoj industriji može izazvati nezadovoljstvo korisnika, ali takve situacije u automobilskoj industriji mogu da prouzrokuju veliku materijalnu štetu, teške telesne povrede, u najgorem slučaju čak i smrt.

Trajno održanje sadržaja memorije u automobilskim sistemima može se trivijalno posmatrati kao funkcija za pamćenje omiljenih radio stanica, podešavanje sedišta, volana, i retrovizora u zavisnosti od trenutnog vozača, pređene kilometraže, prosečne potrošnje goriva, kalibracionih podataka, do bezbednosno kritičnih podataka, kao što su slike sa kamera, radara, i ostalih senzora u trenutcima pre udesa automobila (crna kutija).

U savremenim automobilima se mogu nalaziti i više desetina, pa čak i stotine elektronskih kontrolnih jedinica, koje međusobno komuniciraju preko automobilskih mreža (CAN, FlexRay, u poslednje vreme sve više koristi i Ethernet).

Sa stanovništa bezbednosti od neželjenog pristupa, sigurnost automobilskih mreža nije na nivou kao i u potrošačkoj industriji. Sa ovog aspekta potrebno je obezbediti dodatnu zaštitu, kao i šifrovanje podataka, kako bi se onemogućilo, ili makar znatno otežala manipulacija podacima

koje se trajno čuvaju u automobilu, kao što su na primer dijagnostički podaci, podaci o izvršenim popravkama i servisima, ukupna pređena kilometraža vozila, i slično.

Napredni sistemu za pomoć vozaču su sistemi razvijeni sa namerom da automatizuju, prilagode, poboljšaju sisteme za bezbednost i sigurniju vožnju. Neki od bezbednosnih sistema su sistemi za izbegavanja sudara vozila i saobraćajnih nesreća, sistemi za rano upozorenje vozača, koji obezbeđuju zaštite, i preuzimaju kontrolu nad vozilom u slučaju procene da će doći do saobraćajne nesreće. Dodatni sistemi mogu biti automatsko osvetljenje, tempomat, automatsko kočenje, adaptivno kočenje, GPS servis, upozorenje o stanju na putevima, povezivanje sa pametnim telefonima, upozorenje o promeni vozne trake, parking senzori, sistemi za detekciju umora i pospanosti kod vozača. Postoje mnogi različiti sistemi za pomoć vozaču, koji svakim danom postaju sve brojniji i kompleksniji. Napredni sistemi za pomoć vozaču su najbrže rastući segment u automobilskoj industriji [2] [3].

Želje vodečih kompanija da u svojim vozilima pruže nove usluge zahtevaju ubrzan razvoj naprednih sistema za pomoć vozaču. Sa druge strane potrebno je napraviti pouzdane sisteme sa minimalnom verovatnoćom nepredvidivog ponašanja.

Programska podrška koja se koristi u ovakvim sistemim se razvija po najvišim bezbednostnim standardima, radi obezbeđenja korektne funkcionalnosti i ranom otkrivanju kvarova, smanjujući mogućnost nepredviđenog ponašanja na prihvatljiv nivo.

U ovom radu je prikazan dizajn i implementacija jednog rešenja programske podrške za trajno čuvanje sadržaja memorije u programskom jeziku C, u AUTOSAR okruženju. Cilj rada je implementacija rešenja koje zadovoljava sledeće zahteve:

- Programska podrška pruža funkcionalnost trajnog čuvanja sadržaja memorije.
- Zaštitu integriteta sadržaja trajno čuvane memorije.
- Rešenje razviti po AUTOSAR standardu.
- Programski C kod se kompajlira bez grešaka i upozorenje pod ISO C 99 standardu [4].
- Programski C kod ne narušava MISRA C:2004 pravila i smernice.
- Programsku podršku je moguće integrisati u AUTOSAR sistem.
- Realizovati proveru ulaznih podataka i parametara.
- Rešenje omogućava nadogradnju i dalji razvoj programske podrške.
- Rešenje omogućava iterativni i inkrementalni proces razvoja programske podrške.

Ovaj rad je sačinjen od osam celina. Prvo poglavlje sadrži opis rada.

Drugom poglavlje sadrži osnovne informacije o standardima koji se primenjuju u automobilskoj industriji.

U trećem poglavlju su opisane teorijske osnove neophodne za razumevanje osnovnih koncepata u problemu trajnog održanja sadržaja memorije, i razvoja programske podrške u automobilskoj industriji.

U četvrtom i petom poglavlju su detaljno opisani koncept i implementacija programskog rešenje za trajno očuvanje sadržaja memorije.

U šestom poglavlju je predstavljeni postupci ispitivanja, rezultati verifikacije programskog rešenja, kao i proveru zadovoljenja zahteva zadatka.

Sedmo poglavlje sadrži kratak pregled postignutih rezultata i moguće pravce daljeg razvoja.

Osmo poglavlje navodi korišćenu literaturu prilikom izrade ovog rada.

2. Standardi u automobilskoj industriji

U okviru ovog poglavlja opisani su automotiv standardi ISO 26262, AUTOSAR i MISRA C. Predstavljena je kratka istorija standarda i date su osnovne informacije o standardima koji su značajni za ovaj rad.

2.1 ISO 26262

Nazvan "Road vehicles – Functional safety", (Drumska vozila - Funkcionalna bezbednost) je međunarodni standard funkcionalne bezbednosti električnih i elektronskih sistema za proizvodnju drumskih vozila definisan od strane Međunarodne organizacije za standardizaciju (International Organization for Standardization - ISO) [5].

Funkcionalna bezbednost igra važnu ulogu u svim fazama razvoja autmobilskog proizvoda, od projektovanja, specifikacija, dizajna, implementacije, integracije, verifikacije, validacije , sve do završnog proizvoda.

ISO 26262 standard je adaptacija IEC 61508 standarda Međunarodne Elektrotehničke Komisije. (International Electrotechnical Commission - IEC). ISO 26262 definiše funkcionalnu bezbednost za automobilske električne-elektronske sisteme kroz ceo životni ciklus svih sistema koji su od značaja za bezbednost putnika.

Standard se odnosi samo na električne i elektronske sisteme, ne odnosi se na sisteme kao celine, niti na njihove mehaničke pod-sisteme. Cilj standarda je da se otkriju moguće opasnosti, kao i smanjienje mogućnosti za neispravno ponašanje koje bi moglo da prouzrokuje nepredvidljivo posašanje

Kao i IEC 61508, ISO 26262 je takođe bezbednosni standard zasnovan na riziku, gde su rizici neispravnog funkcionisanja sistema kvalitativno procenjeni i definisane su bezbednosne

mere radi izbegavanja ili kontrolisanja kvarova, otkrivanje ili kontrola nepredvidivih i nasumičnih grešaka u električnim-elektronskim sistemima, kao i da onemogući propagiranje grešaka do drugih sistema.

Glavne ciljevi ovog standarda su:

- Obezbeđuje životni ciklus proizvoda koji je značajan sa aspekta bezbednosti (rukovodstvo, razvoj, proizvodnju, rad, rukovanje, servisiranje, prestanak proizvodnje)
- Pokriva funkcionalne bezbednosne aspekte tokom celog razvojnog procesa
- Obezbeđuje specifičan pristup rizicima u automobilskoj industriji (ASILs - Automotive Safety Integrity Levels)
- Koristeći ASIL nivoe za specificiranje neophodnih sigurnosnih zahteva za postizanje prihvatljivog nivoa rizika
- Obezbeđuje zahteve za validaciju i mere provere koje obezbeđuju da je dovoljan i prihvatljiv nivo bezbednosti postignut.

ASIL nivoi su šema klasifikacije rizika definisana standardon ISO 26262. Ova klasifikacija pomaže u definisanju bezbednosnih zahteva u skladu sa standardom. ASIL nivo je ustanovljen analizom opasnosti i procenom potencijalnog rizika gledajući sa aspekata ozbiljnosti, izlaganja i mogućnosti kontrole.

Standard definiše četiri ASIL nivoa i jedan QM nivo:

- ASIL D
- ASIL C
- ASIL B
- ASIL A
- QM

ASIL D predstavlja najviši zahtev za integritet bezbednosti na proizvodu, dok ASIL A predstavlja najniži. Opasnosti koje su identifikovane kao QM nivo ne diktiraju zahteve za bezbednost.

U kontekstu standarda ISO 26262, opasnost je procenjuje na osnovu relativnog uticaja opasnih efekata na dati sistem, i na koji način će se opasnost ispoljiti. Svaki rizik se procenjuje u

odnosu na ozbiljnost, vreme kojem je vozilo izloženo opasnosti, i verovatnoći da će prosečan vozač biti u stanju da spreči moguće povreda vozača i drugih učesnika u saobraćaju .

Rizik se generalno može izraziti kao:

*Rizik = (Očekivan gubitak u slučaju nezgode) * (Verovatnoća nesreće)*

ili kao:

*Rizik = Ozbiljnost * (Izloženost * Verovatnoća)*

Slično, ASIL nivo može biti izražen kao:

*ASIL = Ozbiljnost * (Izloženost * Mogućnost kontrole)*

ASIL D je najviši nivo klasifikacije rizika standarda ISO 26262. ASIL D predstavlja verovatnoću za opasne po život ili kobne povrede u slučaju kvara i zahteva najviši mogući stepen sigurnosti i uverenje da su bezbednosni zahtevi dostignuti i zadovoljeni.

ASIL D je vredan pomena ne samo zbog povećanog rizika koji zahtevaju izuzetnu strogosti zahteva tokom razvoja, već i zbog toga toga što proizvođači koji tvrde da su njihovi proizvodi razvijeni, certifikovani, ili akreditovani kao ASIL D, olakšavaju razvoj drugih ASIL D komponenti. Bilo koji proizvod koji je saglasan sa ASIL D zahtevima, takođe je u skladu i sa nižim ASIL nivoima.

QM nivo znači da rizik povezan sa opasnim događajem nije nerazuman i stoga ne zahteva mere bezbednosti u skladu sa standardom.

2.2 AUTOSAR

2.2.1 Osnovne informacije

"Saradnja na standardima, takmičenje u implementaciji" - AUTOSAR moto

Tokom prethodnih decenija, razvoj programske podrške u automobilskoj industriji konstantno postaje sve kompleksniji. Trenutno, programska podrška se koristi u kontroli velikog broja funckija koje komuniciraju putem preko povezanih mreža u automobilu. Kompleksna interakcija ovih funkcija zahteva standardizovano i kontrolisano okruženje za razvoj programske podrške. Brz razvoj, i povećanje broja funkcija i elektronskih kontrolnih jedinica u automobilskim sistemima predstavljaju sve veći izazov za proizvođače. Industrijski standard za razvoj elektronskih kontrolnih jedinica (ECU) nudi efikasan način za kontrolu rastuće kompleksnosti sistema , smanji troškove i obezbedi lakši razvoj u budućnosti.

AUTOSAR (AUTomotive Open System ARchitecture) je standardizovana i otvorena arhitektura za automobilske elektronske kontrolne jedinice (ECU). AUTOSAR je kooperacja proizvođača automobila, automobilske opreme, alata, i proizvođača poluprovodničkih komponenti.

AUTOSAR standard sastoji se od specifikacija koje opisuju arhitekturu programske podrške, interfejse aplikacija, i metodologiju. AUTOSAR je slojevita arhitektura koja omogućava upotrebu aplikacija programskih podrški nezavisnih proizvođača. Na taj način programske podrške mogu biti iskorišćene u automobilima i elektronskim komponentama drugih proizvođača, kroz više generacija uređaja. Rezultat je visoka pouzdanost celog sistema, sa značajnim smanjenjem troškova razvoja. AUTOSAR moto: "Saradnja na standardima, takmičenje u implementaciji", jasno prikazuje sa kojim ciljem je standard razvijan. Standard nudi razne pogodnosti za proizvođače opreme, alata, kao i novim kompanijama koje ulaze na tržište automobilske industrije.

Vođeni pojavom inovativnih funkcija u vozilima, savremeni računarski sistemi u automobilskoj industriji su dostigli nivo kompleksnosti koji zahteva značajan tehnološki napredak kako bi se ispunili zahtevi zakonskih regulativa, kao i putnika. Ovo je posebno tačno za vodeće proizvođače automobila i automobilske opreme, koje se često sreću sa konfliktnim zahtevima od strane:

- Zakonskih regulativa - askpeki zaštite životne sredine, kao i bezbednosni zahtevi.
- Udobnost vozača i putnika, kao i dodatni zahtevi, poput infotaiment-a.
- Sistemi za pomoć vozaču (Driver Assistance), sistemi za dijagnostiku i otkrivanje kvarova.

Vodeći proizvođači orginalne opreme (OEMs) i "Tier 1" dobavljači su prepoznali ove zahteve kao zahteve za celu industriju, i odlučili na zajedničku saradnju kako bi odgovorili na ove izazove.

"Tier 1" dobavljači su kompanije koje direktno snabdevaju orginalne proizvođače opreme. Ovaj sistem rangova je posebno rasprostranjen u automotiv, vazduhoplovnoj i računarskoj industriji, gde se konačni proizvod sastoji od mnoštva kompleksnih komponenti. Orginalni proizvodjači opreme su kompanije koje prave proizvod za prodaju krajnjim korisnicima. "Tier 1" dobavljač snabdeva orginalnog proizvođača sa komponentama od kojih se pravi završni proizvod, dok "Tier 2", "Tier 3", itd. dobavljači snabdevaju samo dobavljače koji su na višem nivou u hijerarhiji.

Njihov zajednički cilj je da kreiraju osnovu za industrijsku saradnju i razvoj na osnovnim funkcijama, pružajuci platformu za razvoj koja ohrabruje i podstče nadmetanje-takmičenje u razvijanju inovativnih funkcija. Da bi se ovo postiglo formirano je partnerstvo AUTOSAR sa ciljevima:

- Standardizacija osnove funkcionalnosti programske podrške za elektronske kontrolne jedinice u automobilskoj industriji (Automotive ECUs).
- Skalabilnost i primenjivost na različita vozila i platformske varijante.
- Laku ponovnu upotrebu programske podrške.
- Podrška za različite funkcionalne domene.
- Definicija otvorene arhitekture.
- Saradnja između različitih partnera.
- Razvoj pouzdanih sistema.
- Podrška za međunarodne automotiv standarde, kao i za napredne tehnolgije.

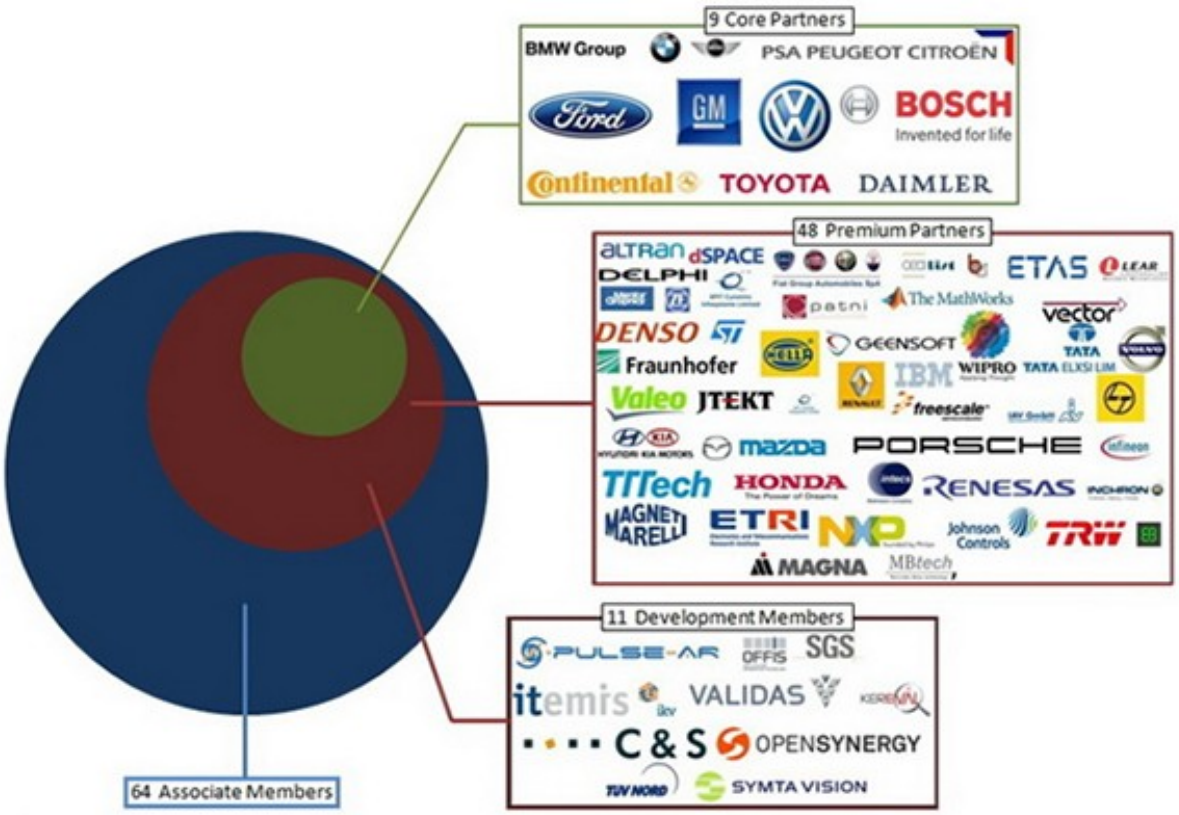
AUTOSAR standard služi kao platformska osnova za dalji razvoj programske podrške koja je implementirana u vozilima. Standard se takođe koristi da minimizira prepreke između različitih funkcionalnih domena.

2.2.2 Istorija

AUTOSAR partnerstvo je alijansa (savez) proizvođača opreme (OEMs) i "Tier 1" proizvođača automobila, koji sarađuju na razvoju, kreiranju i uspostavljanju standarda otvorenog tipa za automotiv električne - elektronske arihtekture koje služe kao osnovna infrastrukura za upravljanjem razvojem, obezbeđujući standardne module programske podrške.

Inicijalne diskusije o zajedničkim izazovima i ciljevima su vodile kompanije: BMW, Bosch, Continental, DaimleChrysler i Volkswagen avgusta 2002 godine. Novembra 2002 godine je sastavljen zajednički tehnički tim, radi sastavljanje strategije implementacije. Partnerstvo između članova formalno je potpisano Jula 2003 godine. Glavnim partnerima alijanse su se kasnije pridruzile i kompanije: Ford , Peugeot Citroën, Toyota i General Motors [6].

Od januara 2014.više od 180 kompanija učestvuje u AUTOSAR partnerstvu. [7]



Slika 2.1 Članovi AUTOSAR alijanse [8]

2.2.3 Motivacija i ciljevi

Motivacija:

- Upravljanje sve većom kompleknošću električnih i elektronskih automobilskih sistema , povezanih sa porastom broja i složenosti funkcija.
- Fleksibilnost za modifikacije proizvoda, nadogradnju i ažuriranje.
- Skalabilnost rešenja.
- Unapređen kvalitet i pouzadnost elektronskih i električnih sistema.

Ciljevi:

- Ispunjenje sadašnjih i budućih zahteva u razvoju vozila, kao što su dostupnost i bezbednost, nadogradnja i ažuriranje programske podrške, kao i generalno održavanje.
- Povećana skalabilnost i fleksibilnost za integraciju i transfer funkcionalnosti na druge platforme, povećana ponovna upotrebljivost vec razvijenih programskih podrški.
- Viša upotreba "sa police" programske podrške, kao i fizičke arhitekture u celom proizvodnom procesu.
- Unapređenje kontrole faktora rizika proizvoda i procesa.
- Optimizacija troškova.

2.2.4 Ključne osobine

Modularnost i konfigurabilnost:

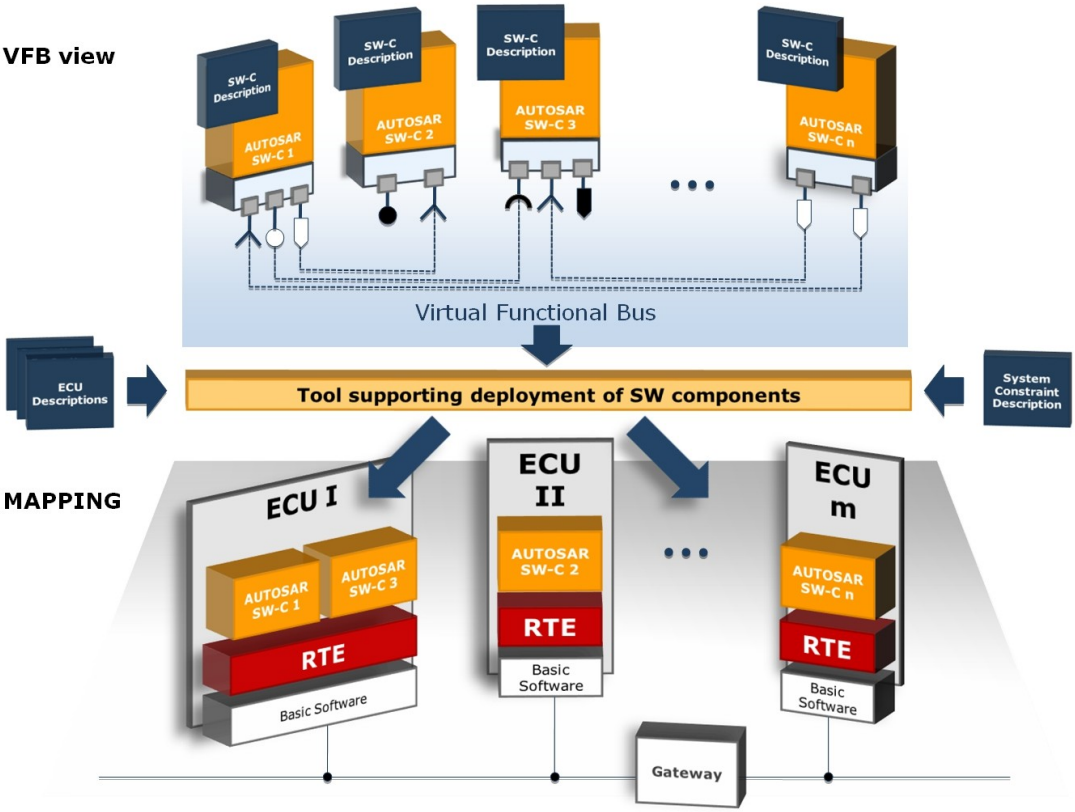
- Definicija slojevite arihtekture osnovne programske podrške za elektronske kontrolne jedinice kako bi se obuhvatile i ogradile zavisnosti fizičke arhitekture.
- Integracija osnovnih modula programske podrške razvijenih od strane različitih kompanija, povećavajući ponovnu upotrebu programske podrške.
- Omogućava transfer funkcionalnih komponenti programske podrške određenog električno elektronskog sistema u drugi sistem.
- Resursno optimizovana konfiguracija infrastrukutre programske podrške svake elektronske kontrolne jedinice u zavsnosti od rasporeda funkcionalnosti.
- Skalabilnost i proširivost elektronsko električnih sistema u svim delovima proizvodnih linija vozila.

Standardizovani interfejsi:

- Standardizacija različitih interfejsa za razdvajanje slojeva AUTOSAR programske podrške.
- Enkapsulacija funkcionalnih komponenti programske podrške.
- Definicija tipova podataka funkcionalnih komponenti programske podrške.
- Standardizacija interfejsa osnovnih modula infrastrukture.

Okurženje izvršenja (RTE):

- Omogućava komunikaciju unutar jednog i između različitih elektronskih kontrolnih jedinica preko svih čvorova mreže u vozilu.
- Nalazi se između funkcionalnih modula i osnovnih modula programske podrške.
- Svi entiteti povezani na AUTOSAR RTE moraju biti u skladu sa AUTOSAR specifikacijama.
- Omogućava laku integraciju specifičnih funkcionalnih modula programske podrške.
- Standardizacija testnih specifikacija radi provere osnovne programske podrške [9].



Slika 2.2 Pregled osnovnog pristupa AUTOSAR [10]

2.2.5 Budućnost AUTOSAR-a

Partnerstvo je u potpunosti prihvatilo 'Razvojni Koncept' u glavnu specifikaciju AUTOSAR procesa, kontinualnu evaluaciju i razvoj novih koncepata, validirajući ih pre integracije u standard. Naredne verzije standarda sadržaće nove koncepte, zahtevane od strane AUTOSAR partnera. Funkcionalna bezbednost, Ethernet TCP/IP komunikacija, podrška za procerore sa više jezgara, bezbednost, dijagnostika, kao i podrška za specifična regionalna tržišta su samo neki od fokusnih tačaka za razvoj u narednim godinama.

Aktivna je saradnja sa drugim standardizacionim telima kao sto su GENIVI, Car-to-X, i Ethernet, radi obezbeđivanja infrastrukturne podrške za više standarda.

2.3 MISRA C

MISRA C je skup smernica za razvoj programske podrške u C jeziku, za upotrebu u automobilskoj inustriji. Razvijen je od strane MISRA (Motor Industry Software Reliability Association) organizacije. Njeni ciljevi su unapređenje pouzdanosti, portabilnosti, i sigurnosti programskog koda, u kontekstu računarskih sistema, posebno onih koji su programirani po ISO C-C90-C99 standardu. MISRA C je široko prihvaćen standard i model za najbolje prakse u radu. Koristi se u razvoju programske podrške za automotiv, avio, telekomunikacione, medicinske industrije, kao i drugih [11].

MISRA C standard trenutno je trećoj iteraciji. Prva iteracija standarda je MISRA-C:1998, i imala je 127 smernica, od čega su 93 obavezne, a preostale 34 saveti [12]. Druga edicija je nazvana "Guidelines for the use of the C language in critical systems" ili MISRA-C:2004, i sadrži 142 pravila, od kojih su 122 obavezna. MISRA-C:2012 proširuje podršku na C99 verziju C programskog jezika. Sadrži 143 pravila i 16 direktiva, koja su podeljena u tri grupe: obavezna, potrebna, i saveti. U aprilu 2016, MISRA je objavila *MISRA Compliance:2016* [13] , koji obezeđuje poboljšane smernice za postizanje usaglašenosti sa standardom.

2.3.1 MISRA C smernice

Na početku novog projekta, najnoviji MISRA standard bi trebalo koristiti [14]. Zastareli standardi se koriste za projekte koji su započeti pre nastanka nove iteracije standarda. Svaka smernica pripada jednoj klasi smernica: Obavezno, Potrebno, i Saveti

Klasifikacija:

.

- Obavezne smernice koje uvek moraju biti zadovoljenje (ispoštovane).
- Potrebne smernice koje uvek moraju biti zadovoljene, osim ako nije predmet odstupanja.
- Saveti se smatraju dobrom praksom, ali ne moraju biti zadovoljeni.

Pravila se mogu logički podeliti u kategorije:

- Izbegavanje mogućnosti razlika u kompajleru, npr veličina celobrojne promenjive u C može da varira u zavisnosti od korišćenog kompajlera, ali celobrojna promenjiva od 16 bita je uvek 16 bita (C99 standardizacija `int16_t`).
- Izbegavanje korišćenja funckija koje su sklone greškama, npr funkcija `malloc()` može neuspešno da se izvrši.
- Pisati održiv i lako čitljiv kod, u kome se lako pronalaze greške.
- Koristeći pravila i smernice koja su se dobro pokazali u praksi.
- Izbegavanje pravljenja previše kompleksnih rešenja.

Mnoga MISRA C pravila se mogu okarakterizovati kao smernice zato što pod određenim uslovima inženjer može odstupiti od pravila, a da se smatra da je u saglasnosti sa standardom. Odstupanja moraju biti dokumentovana, ili u samom programskom kodu, ili u posebnom dokumentu. Pored dokaza da je inženjer razmotrio da bezbednost sistema nije ugrožena odstupanjem od pravila, zahtev za odstupanje od pravila mora da sadrži:

- Pravilo od kog se odstupa,
- Razlog odstupanje,
- Objašnjenje zašto to odstupanje neće narušiti bezbednost sistema [15]

Većina pravila i smernica se može proverit koristeci programske alate koji vrše statičku analizu koda. Ostala pravila zahtevaju dinamičku analizu koda. Neki od programskih alata koji se koriste za proveru usaglašenosti sa standardom: Astrée, Coverity, ECLAIR, Goanna, Rational Test RealTime, Polyspace, QA-C, SonarQube , Understand i drugi.

3. Teorijske osnove

U narednom poglavlju biće opisane teorijske osnove i metodologije razvoja programske podrške koje je potrebno izložiti pre samog predstavljanja implementacije programskog rešenja. Predstavljene metodologije značajno olakšavaju proces razvijanja programske podrške, pogotovo u modernim sistemima visoke kompleksnosti.

Date su informacije o razvoju programske podrške zasnovane na modelu, generatorima programskog koda, memorijskim medijumima, i matematičke osnove ciklične redudantne provere (CRC).

Programska podrška se može koristiti u vozilu tek kada se utvrdi da je razvijena po odgovarajućim standardima, i da zadovoljava bezbednosne kriterijume. Izuzetak od ovog pravila su razvojna vozila koja služe za ispitivanje funkcionalnosti sistema, kojima upravljaju posebno obučeni vožači.

3.1 Razvoj programske podrške zasnovan na modelu

3.1.1 Model

Model predstavlja opis sistema ili njegovog dela zapisan korišćenjem dobro definisanog jezika koji podržava automatsku interpretaciju od strane računara. Dobro definisan jezik mora da poseduje apstraktnu sintaksu (meta model), konkretnu sintaksu (notacija, prezentacija), i semantiku. Meta model definiše osnovne koncepte jezika i njegove međusobne relacije. Drugim rečima, model predsavlja apstrakciju realnog sistema, a meta model je eksplicitna specifikacija apstrakcije koja definiše načine na koji se apstrakcija obavlja.

3.1.2 Inženjerstvo upravljano modelima (MDE)

Metodologije razvoja programske podrške zasnovane na modelu fokusiraju se na pravljenju i eksploataciji konceptualnih modela svih tema koje se odnose na određeni problem. Cilj je napraviti apstraktnu predstavu problema, znanja, aktivnosti i tehnologija koje se koriste u rešavanju postavljenog problema, povećanje produktivnosti i kvaliteta, nezavisnost od korištenih tehnologija, kao i kvalitetna dokumentacija i jednostavnije održavanje.

Jedna od tehnika je inženjerstvo upravljano modelima (MDE). Osnovna ideja je razmatranje modela na visokom nivou apstrakcije, bez detalja implementacije. Podizanjem nivoa apstrakcije smanjuje se složenost modela. Povećanje produktivnosti se postiže maksimiziranjem kompatibilnosti između sistema, korišćenjem standardizovanih modela, uproštavanjem procesa dizajniranja, olakšavanjem komunikacije između različitih timova koji rade na sistemu, korišćenjem standardizovane terminologije, i korišćenjem tehnika i tehnologija koje su se pokazale kao najbolje u praksi (*Best practice*).

Paradigma modela se smatra efikasnom ako model ima smisla sa stanovništva korisnika koji je upoznat sa sistemom, i ako on može poslužiti kao osnova za implementaciju sistema.

Obećavajuci pristup za rešavanje problema kompleksnih fizičkih arhitektura i programskih podrški, usled nemogućnosti programskih jezika treće generacije da smanje kompleksnost, i izraze koncepte problema određenih domena, je razvoj programske podrške korišćenjem inženjerstva zasnovanog na modelima [16].

3.2 Generatori izvornog koda

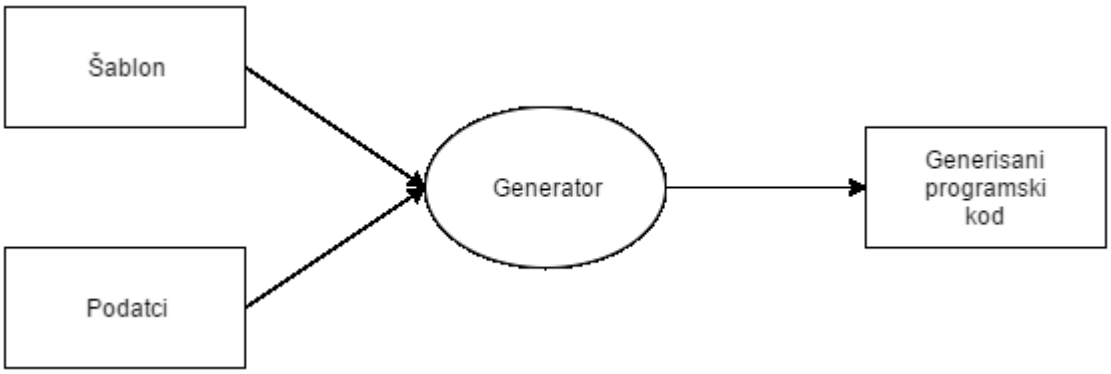
Generisanje izvornog koda je proces izrade programskog izvornog koda na osnovu ontološkog modela upotrebom šablona i postiže se programskim alatima za obradu šablona ili korišćenjem integrisanih razvojnih okruženja (IDE) [17].

Integrisana razvojna okruženja se mogu posmatrati kao generatori programskog koda, imajuću u vidu da takvi alati značajno olakšavaju proces pisanja programskog koda.

U okviru ovog rada, pod generatorima programskog koda, i generisanjem koda misli se na programske alate koji se koriste za obradu šablona. Upotreba ovakvih alata je korisna u slučajevima gde se slični delovi koda pojavljuju više puta, ili kada je se zahtevi i specifikacije često menjaju. U situacijama kada se ulazne specifikacije i zahtevi često menjaju, pogodno je

koristiti generatore koda. U slučaju promene ulaznih specifikacija, potrebo je samo ponovo izgenerisati programski kod.

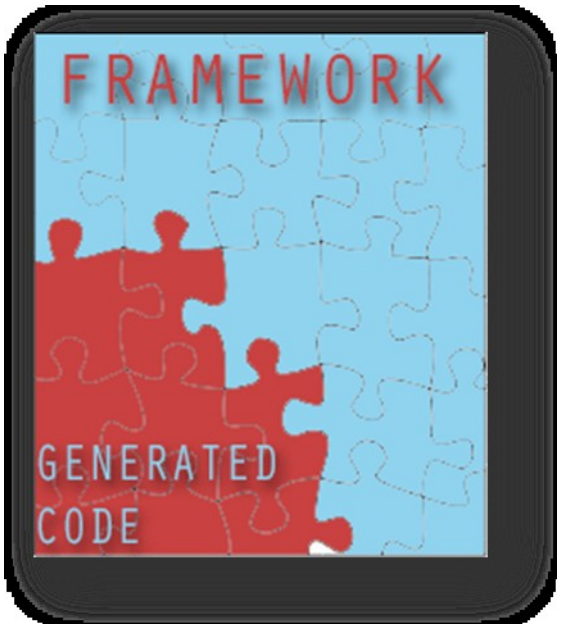
Ovakav način rada značajno smanjuje broj grešaka i olakšava pronalazak grešaka u programskom kodu, omogućava brzu i automatsku promenu programskog koda u zavisnosti od promena u zahtevima i ulaznim specifikacijama.



Slika 3.1 Princip rada generatora koda

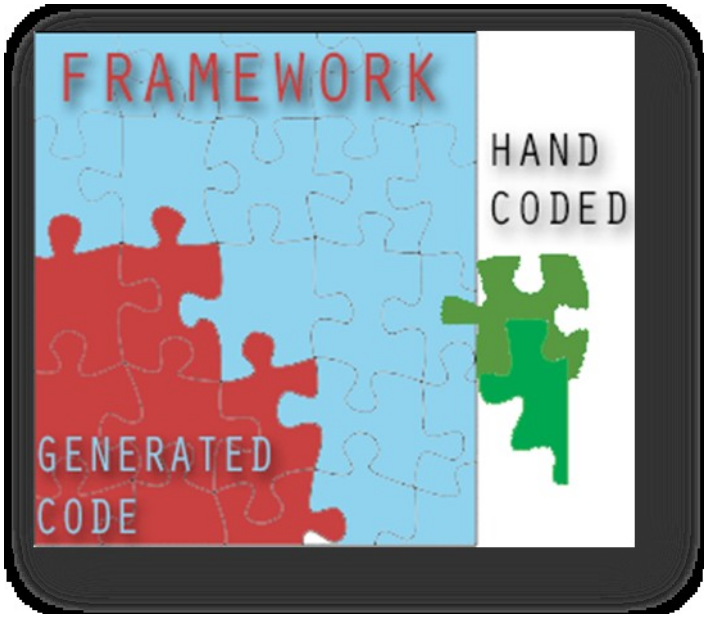
Podaci za generisanje koda se mogu dobaviti na različite načine. Neki od često korištenih načina su parsiranjem datoteka (XML, CSV ili neki drugi format), direktnim pristupom podacima u operativnoj memoriji računara, čitanjem baze podataka.

U idealnim slučajevima struktura generisanog programskog koda se sastoji od osnove i generisanog programskog koda.



Slika 3.2 Prikaz idealne strukture generisane aplikacije

U realnim situacijama, mnogo češće se struktura sastoji od osnove, generisanog dela i dodatnog, ručno pisanog programskog koda.



Slika 3.3 Prikaz realne strukture generisane aplikacije

Dodavanje ručno pisanog programskog koda u strukturu generisane aplikacije je često posledica različitih fizičkih arhitektura na kojoj se izvršava programski kod.

Pri projektovanju integrisanja ručno pisanog sa generisanim programskim kodom neophodno je obezbediti podršku za inkrementalni i iterativni način rada. U razvoju, podrazumevaju se česte izmene modela. Potrebno je omogućiti da nakon promene modela ponovno generisanje koda ne ugrožava ručno unete izmene. To je moguće postići definisanjem zaštićenih zona u programskom kodu, ili fizičkim odvajanjem datoteka generisanog i ručno pisanog programsko koda.

Model se nalazi na višem nivou apstrakcije nego programski kod koji je generisan na osnovu modela. Zbog toga je nije moguće održati model u konzistentom stanju nakon ručne promene u generisanom programskom kodu. Zbog ovoga generisani programski kod ne treba ručno menjati. Generisani kod koji je ručno menjan se više ne smatra generisanim kodom. Neophodna je precizna definicija koji delovi su generisani, a koji su ručno pisani [18].

Preporučuje se da generisani programski kod bude lako čitljiv, dodavanje komentara u generisanom programskom kodu, ispravno korišćenje tabulatora (uvlačenje koda), dodeljivanje smislenih naziva promenjivima i funkcijama, generisanja dodatnih informacija radi lakšeg

održavanja, kao sto su nazivi korištenih šablona, datum i vreme generisanja programskog koda, i slično. Ukoliko je potrebno pružiti podršku za različite fizičke arhitekture, preporuka je pisati šablone tako da ne zavise od fizičke arhitekture, i implementirati podršku za specifične fizičke arhitekture u generatoru.

Neki od često korištenih alata za obradu šablona su:

- Jinja (Python)
- Jinja2 (Python)
- Đango (Python)
- FreeMaker (Java)
- Twig (PHP)
- Jwtig (Java)
- Amber (C++)
- AutoGen (C)
- Mako (Python)
- mTemplate (PHP)

3.3 Trajno čuvanje memorije

Memorije sa stalnim sadržajem, nepromenjive memorije, ili stalne memorije su vrsta memorija koje mogu da održe svoj sadržaj i po prestanku napajanja. Koriste se za trajno čuvanje podataka. Danas postoji više tipova memorija sa stalnim sadržajem. Primeri ovakvih memorija su: ROM, fleš memorije, feroelektrični RAM, većina magnetnih medijuma (tvrdi diskovi, flopi diskovi, magnetne trake), optički diskovi kao i zastarele metode čuvanja podataka, papirne trake i bušene karte [19].

Podatke je moguće trajno sačuvati i u memorijama sa nasumičnim pristupom, ako se obezbedi konstantno napajanje memorijskog uređaja i u slučaju prekida spoljnog napajanja. To se najčešće postiže korišćenjem baterije, koja se puni za vreme normalnog rada uređaja, a koristi se za održanje radnog napona same memorije kada je uređaj isključen. Ovakav pristup nije pogodan za upotrebu u automobilskoj industriji, zbog životnog veka baterije, i zbog mogućnosti trajnog gubitka podataka u slučaju prestanka rada baterije.

U savremenjim računarskim sistemima primarna je uglavnom RAM (memorije sa nasumičnim pristupom). Memorije sa stalnim sadržajem su uglavnom skuplje, imaju lošije performanse, i kraći životni vek od memorija sa nasumičnim pristupom.

U okviru ovog rada, od posebnog interesa su polu-provodničke fleš memorije zbog dostupnosti, cene i kapaciteta. Magnetni medijumi su osetljiviji na fizičke potrese i udare, pa zbog toga nisu pogodni za upotrebu u automobilskoj industriji.

Polu-provodničke memorije se izrađuju u dve tehnologije, NOR i NAND, zasnovane na NI i NILI logičkim kolima. Poluprovodničke memorije NOR tipa dozvoljavaju brisanje i upis jednog bajta (najmanja adresabilna lokacija, dužina mašinske reči), dok NAND memorije moraju biti upisivane po blokovima (ili stranicama). NAND memorije se koriste u memoriskim karticama (SD, MMC, microSD), USB fleš memorijama i popuprovodničkim diskovima (SSD). Mana poluprovodničkih memorija je relativno mali broj upisa u određeni memorijski blok. Ovaj nedostatak se može značajno smanjiti korišćenjem memorijskih uređaja koji imaju mehanizam za nivelisanje broja upisa u blokove (wear leveling), ili korišćenjem sistema datoteka koji obavljaju sličnu funkciju. Životni vek poluprovodničke memorije se može produžiti verifikovanjem upisa, i u slučaju detektovanja greske, blok se označava kao pokvaren, i više se ne koristi. (Bad Block Management - BBM) [20].

3.4 CRC

CRC je izumeo William Wesley Peterson 1961 godine [21]. Ciklična redudantna provera je kod za detekciju greške koji se koristi u računarskim mrežama i uređajima za skladištanje podataka. Podacima se dodaje kratak redudantni deo, koji se izračunava na osnovu podataka. Prilikom prijema podataka, vrednost se ponovo izračunava i poredi sa primljenom. U slučaju da se vrednosti razlikuju smatra se da je došlo do greške prilikom prenosa podataka i korektivne mere se mogu preduzeti. Redudantna vrednost se izračunava na osnovu ostatka deljenja sa polinomom. Ciklična redudantna provera je popularna zbog lake implementacije fizičke arhitekture, lake matematičke analize, i dobrih osobina u otkrivanju najčešćih grešaka koje se događaju u prenosu podataka.

3.4.1 Matematičke osnove ciklične redudantne provere

Ciklična provera redudantnosti je zasnovana na aritmetici po modulu 2, i deljenju polinoma. Niz bitova koji se prenosi se posmatra kao niz koeficijenata polinoma. Niz $a_n a_{n-1} \ldots a_1 a_0$ odgovara polinomu $M(x)=a_n x^n+a_{n-1}x^{n-1}+\ldots+a_1 x+a_0$.

Postupak kodiranja se svodi na odabir polinom generatora G(x) stepena k. Od dužine odabranog polinom generatora direktno zavisi dužina kontrolne vrednosti. Izračuna se $x^k M(x)/G(x)$, i dobije ostatak pri deljenju se označava kao R(x) .

Koeficijenti ostatka deljenja se dodaje na kraj poruke. Postupak dekodiranja poruke se svodi na deljenje kontrolne vrednosti sa polinom generatora Ako je ostatak pri deljenju nula, nema grešaka u prenosu podataka. Ako ostatak nije nula, postoje greške pri prenosu.

Postoje greške koje nije moguće detektovati, ali se dobrim izborom polinom generatora one smanjuju.

3.4.2 Greške u radu sa memorijom

Osim u prenosu, postoji i mogućnost da dođe do greške prilikom upisa ili čitanja podataka. Greške u radu sa memorijom se mogu podeliti u dve grupe: stalno prisutne defekte i privremene (prolazne) defekte. Stalno pristutni defekti se pojavljuju usled neispravnosti memoriske celije, koja nije u stanju da pouzdano čuva podatke, ili ih menja bez spoljašnjeg uzroka. Prolazni defekti su najčešće posledica smetnji u napajanju, elektromagnetnih talasa, ili radioaktivnog zračenja.

U nekim slučajevima, čak i ako je detektovana greška u prenosu podataka, ponovno slanje podataka nije moguće, jel često ti podaci više ne postoje. U situacijama gde je pored detekcije greške neophodno omogućiti i oporavak od grešaka potrebno je koristiti naprednije metode koje omogućavaju ne samo otkrivanje, nego i korekciju pronađene greške. U takvim situacijama je potrebno koristiti memorije sa mogućnošću ispravljanja greške (ECC memory).

3.4.3 Primena algoritma ciklične redudantne provere

Ciklična redudantna provera se koristi za izračunavanje CRC vrednosti, kratke sekvence fiksne dužine za svaki blok podataka koji se šalje ili upisuje u medijum za trajno čuvanje. Izračunata vrednost se naziva i redudantna, zato što ne nosi korisne informacije. Kada se blok podataka primi ili pročita, CRC vrednost se ponovo izračunava i poredi sa primljenom. Ako se vrednosti razlikuju, blok podatak sadrži grešku. U suporotnom, predpostavlja se se da su podaci ispravni, i da su podaci preneti bez greške. Postoji mala mogućnost da greška nije detektovana, ako orginalni podaci i poslati podaci imaju istu CRC vrednost [22].

Algoritmi ciklične redudantne provere su dizajnirani za zaštitu protiv tipičnih grešaka nastalih u komunikaciji i prenosu podataka, za brzu proveru integriteta podataka. Međutim, nisu

pogodni za zaštitu protiv namerne promene podataka. Na osnovu podataka se relativno lako može otkriti koji CRC algoritam se koristi, sto ovu proveru čini nepogodnom za upotrebu u digitalnim potpisima i u sistemima od kritične važnosti [23].

Izbor generatora polinoma za je najvažniji deo implementacije algoritma. Potrebno je izabrati polinom koji maksimalno povećava verovatnoću pronalaska greške. Najvažnija osobina polinoma je njegova dužina, zbog direktnog uticaja na dužinu izračunate CRC vrednosti, vrednosti koja se dodaje na kraj poruke.

Dizajn polinoma zavisi od maksimalne duzine bloka podataka nad kojim će se računati CRC vrednost, resursa fizičke arhitekture na kojoj se implementira zaštita, kao i željene performanse.

Najčešće koričćene dužine polinoma su:

- 2 bita (CRC-1) - Provera parnosti
- 9 bita (CRC-8)
- 17 bita (CRC-16)
- 33 bita (CRC-32)
- 65 bita (CRC-64)

Naziv	Upotreba	Polinom
CRC-1	Fizičke arhitekture, poznatije kao bit parnosti	0x1
CRC-5-USB	USB token paketi	0x05
CRC-8	DVB-S2 [24]	0xD5
CRC-8-AUTOSAR	Primena u automotiv integraciji [25]	0x2F
CRC-11	FlexRay	0x385
CRC-24	FlexRay	0x5D6DCB
CRC-32	Ethernet, PNG datoteke [26]	0x04C11DB7
CRC-32 (Castagnoli)	Ext4 sistem datoteka	0x1EDC6F41

Tabela 3.1 Polinomi korišteni u računanju CRC vrednosti

U praksi, računanje CRC vrednosti najviše podseća na operaciju binarnog deljenja. Jedina razlika je da operacije oduzimanja ne pozajmljuju od značajnijih cifara. Takva operacija se naziva logička operaciji ekskluzivno ILI.

Ulazi		Izlaz
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 3.2 Logička operacija ekskluzivno ILI

3.4.4 Detekcija greške

Detekcija greške je osobina CRC koja zavisi od stepena korištenog polinoma i specifičnosti izabranog polinoma. Greška ce biti neopažena od strane CRC algoritma ako i samo ako je CRC vrednost orginalnih podataka i podataka sa greškom identična. Greška neće biti detektovana ako su podaci i kontrolna vrednosti namerno promenuti.

Nije moguća detekcija grešaka koje se sastoje od niza nula dodatih na podatke, ili nedostajućih vodećih nula zato što je CRC kalkulacija zasnovana na deljenju polinoma.

Sve greške u jednom bitu će biti detektovane sa bilo kojim generator polinomom, pod uslovom da barem dva člana plinoma imaju ne nulte koeficijente.

Sve greške u dva bita će biti detektovane ako su udaljene manje od reda veličine korištenog generator polinoma.

Sve rafalne greške dužine manje od stepena generator polinoma ce biti detektovane ako generator polinom ima ne nulti prvi član.

4. Koncept rešenja

Rešenje prikazano u ovom radu se sastoji iz dva dela: ručno pisanog dela programskog koda, i generisanog dela programskog koda. Ideja za ovakvu realizaciju se javlja iz razloga primećenih u radu na komercijalnim projektima. Ručno pisani deo sadrži deo implementacije koji je nazavisan od ulaznih podataka, pružajući osnovne funkcionalnosti, dok u generisanom delu se nalaze samo delovi koda koji su zavisni od ulaznih podataka.

Prikazano rešenje se integriše u AUTOSAR sistem, i pruža mogućnosti trajnog očuvanja sadržaja RTE. AUTOSAR sistem koristi funkcionalnosti modula za trajno čuvanje memorije prilikom pokretanja, i prilikom gašenja sistema. Kontrola modula se odvija u AUTOSAR sistemu.

ASIL nivo prikazanog rešenja zavisi od ASIL nivoa korišćenog sistema datoteka (file system) korišćene u fizičkoj arhitekturi. Često korišteni sistemi datoteka uglavnom nemaju dovoljno visok nivo pouzdanosti da bi se klasifikovali kao ASIL. Postoje specijalizovani operativni sistemi čiji sistemi datoteka mogu da se klasifikuju kao ASIL nivo. Ovakvi operativni sistemi su skoro uvek operativni sistemi koji rade u realnom vremenu, sa čvrstim vremenskim ograničenjima.

4.1 Negenerisani deo

Obezbeđuje funkcionalnost očuvanja blokova podataka, korišćenjem stalne memorije. Obezbeđuje funkcije čitanja i pisanja glavne i pomoćne datoteke bloka podataka. Blok podataka čine dva datoteke, glavna i pomoćna. Pomoćna datoteka se koristi u slučaju da je glavna datoteka oštećena, ili ako je utvrđeno da podaci u glavnoj datoteci nisu validni, ili da integritet nije

očuvan. Integritet može narušiti u slučaju kvara memoriskog uređaja, ili u slučaju neovlašćene promene memorije.

Očuvanost integriteta blokova podataka se proverava koristeći algoritme za izračunavanje CRC vrednosti.

Modul koristi brzu CRC kalkulaciju, koja zahteva unapred izračunatu referentu tabelu, koja značajno smanjuje broj potrebnih procesorskih instrukcija za izračunanje CRC vrednosti. Ovo ubrzanje je posebno značajno u sistemima sa ograničenim resursima.

U nekim fizičkim arhitekturama postoje posebni koprocesori za računanje CRC. Takvi procesori su u mogućnosti da izračunaju CRC vrednost mnogo brže nego računanje vrednosti koristeći matematičke operaciji u programskoj podršci. Izbor između računanja CRC u programskoj podršci ili u fizičkoj arhitekturi zavisi od željenih performansi. Računanje u fizičkoj arhitekturi je značajno brže, ali fizičke arhitekture koje poseduju takve funkcionalnosti su kompleskniji i skuplji.

4.2 Generisani deo

Obezbeđuje funkcionalost generisanja izvornog programskog koda na osnovu ulazne datoteke. Ulazna datoteka je u formatu XML (eXtensible Markup Language). Ulazna datoteka se proverava XSD šemom, proveravajući da li je ulazna datoteka validna, i da će generator izvornog koda napraviti validan izvršni kod. Ukoliko ulazna datoteka nije korektna u odnosu na shemu, prijavljuje se greška, i generisanje programskog koda se prekida.

Generisani deo programskog rešenja obezbeđuje funkcije trajnog čuvanja sadržaja memorije celom AUTOSAR sistemu. Funkcionalnosti koje se obezbeđuju sistemu se mogu podeliti u dve grupe: trajno čuvanje sadržaja memorije pre gašenja uređaja, i povratak sadržaja memorije po uključenju.

Ovakav pristup omgućava interativni i inkrementalni razvoj programske podrške. U slučaju promene ulaznih parametara, dovoljno je samo ponovo izgenerisati deo programske podrške koji zavisi od ulaznih parametara modela, bez potreba dodatnih ručnih izmena u programskom kodu. Na ovaj način se omogućava brz odgovor na promenu ulaznih podataka (XML datoteke).

5. Programsko rešenje

U ovom poglavlju dat je detaljan opis realizovanih programskih modula, odnosno realizovanih funkcija u okviru programske podrške ručno pisanog dela i u okviru programske podrške generisanog dela izvornog programskog koda.

5.1 Negenerisani deo

Negenerisani deo je pisanu u programskom jeziku C. Razvijen je po ISO C 99, ISO 26262, MISRA C, i AUTOSAR standardima.

Sastoji se od dva modula: NvmManager, i CRC modula.

NvmManager modul pruža funkcionalnosti čitanja i pisanja blokova podataka, datoteka, kao i proveru očuvanosti integriteta blokova podataka, koristeći CRC proveru.

CRC modul pruza funckionalnost brzog računanja CRC vrednosti u programskoj podršci, koristeći unapred izračunate referentne tabele. Ukoliko se na zadatoj fizičkoj arhitekturi nalazi koprocesor za računanje CRC, koristi se računanje CRC vrednosti u fizičkoj arhitekturi.

Datoteke koje spadaju u negenerisani deo:

Naziv datoteke	Opis
NvmManager.c	Sadrži implementaciju funkcija za operacije nad blokovima podataka
NvmManager.h	Sadrži deklaracije interfejs funkcija koje se koriste u generisanom delu
crc.c	Sadrži implementaciju računanja CRC kontrolne vrednosti
crc.h	Sadrži deklaraciju funckije za računanje CRC kontrolne vrednosti
Rte_Types.h	Sadrži deklaracije AUTOSAR tipova podataka.

Tabela 5.1 Spisak datoteka negenerisanog dela programskog rešenja

5.1.1 NvmManager

Modul služi za trajno čuvanje podataka u stalnoj memoriji. Sastoji od sledećih funkcija.

Naziv funkcije	Kratak opis
NvmReadBlock()	Funckija za čitanje jednog bloka podatak
NvmWriteBlock()	Funkcija za upis jednog bloka podataka
NvmReadFile()	Funkcija za čitanje jedne datoteke
NvmWriteFile()	Funkcija za upis jednoe datoteke
NvmCalcCrc()	Funkcija za računanje CRC
NvmCrcCheck()	Funkcija za proveru CRC

Tabela 5.2 Funkcije programske podrške negenerisanog dela

5.1.1.1 Std_ReturnType NvmReadBlock (blockId_t blockId, dataPtr_t dataPtr , NvM_Decriptor_SSH_t * descriptor);

Pruža funkcionalnost čitanja bloka podataka koji se sastoji od glavne i pomoćne datoteke. Proverava se da li je identifikator bloka koji se čita identičan ulaznom parametru funkcije.

Prvo se iz stalne memorije čita glavna datoteka koristeći funckiju NvmReadFile() i proverava se integritet koristeći NvmCrcCheck() funkciju.

Ukoliko je čitanje neuspešno završeno , ili je utvrđeno da je integritet datoteke narušen, iz stalne memorije se čita pomoćna datoteka i proverava se integritet pomoćne datoteke. Ukoliko je čitanje pomoćne datoteke neuspešno završeno, ili je utvrđeno da je integritet narušen, prijavljuje se greška, i upisuje se odgovarajući status datog bloka podataka u RTE.

5.1.1.2 Std_ReturnType NvmWriteBlock (blockId_t writeBlock , dataPtr_t dataPtr , NvM_Decriptor_SSH_t * descriptor);

Pruža funkcionalnost upisa jednog bloka podataka u stalnu memoriju koji se sastoji od glavne i pomoćne datoteke.

Proverava se da li je identifikator bloka koji se upisuje identičan onome koji je prosleđen kao ulazni argument funkcije.

Pre upisivanja podataka izračunava se kontrolna CRC vrednosti koristeći funkciju NvmCalcCrc(). Izračunata vrednost se dodaje na kraj podataka. Tako pripremljeni blok podataka se upisuje u stalnu memoriju koristeći dva poziva funkcije NvmWriteFile().

U slučaju greške tokom poziva funkcije NvmWriteFile(), greška se prijavljuje.

5.1.1.3 static Std_ReturnType NvmReadFile (fileName_t fileName , dataPtr_t data , size_t dataSize);

Pruža funkcionalnost čitanja jedne datoteke bloka podatak iz stalne memorije. Čita sadržaj stalne memorije koristeći sistemske pozive niskog nivoa , kao što je na primer POSIX UNIX funkicje open(), read(), close().

U slučaju neuspešnog čitanja prijavljuje se greška.

5.1.1.4 static Std_ReturnType NvmWriteFile (fileName_t fileName , dataPtr_t data , size_t dataSize);

Pruža funkcionalnost upisa jedne datoteke bloka podatka u stalnu memoriju. Upisuje podatke koristeći sistemske pozive operativnog sistema, kao što su na primer POSIX UNIX funkcije open(), write(), close().

U slučaju greške tokom upisa datoteke, prijavljuje se greška.

5.1.1.5 static void NvmCrcCalc (void * data, CRCfield_t * crc, NvM_Decriptor_SSH_t * descriptor);

Pruža funkcionalnost računanja kontrolne CRC vrednosti. Funkcija se poziva se pre upisa bloka podataka u stalnu memoriju. Izračunata vrednost se koristi za proveru očuvanosti integriteta podataka.

Koristi se brza implementacija u programskoj podršci.

5.1.1.6 static void NvmCrcCheck (void * data, CRCfield_t * crc, NvM_Decriptor_SSH_t * descriptor);

Pruža funkcionalnost ispitivanja očuvanosti integriteta bloka podataka koji je pročitan iz stalne memorije. Izračunava vrednost CRC na pročitanim podacima, i upoređuje je sa pročitanom vrednošću. Ukoliko su izračunata i pročitana vrednost identične, zaključuje se da je

integritet bloka podataka očuvan. Ako su vrednosti različite, integritet je narušen, i ne može se garantovati validnost pročitanih podataka.

5.1.2 CRC - Modul za računanje CRC

Modul za računanje CRC vrednosti nad podacima je preuzet sa interneta, razvijen je od strane *Michael Barr* [27]. CRC zaštita je implementirana korišcenjem 32-bitnog Ethernet polinoma (0x04C11DB7), koji se dobro pokazao u praksi. U prikazanom rešenju CRC vrednost se računa u programskoj podršci, korištenjem unapred izračunate referentne tabele, koja značajno ubrzava računaje CRC vrednosti.

Referentna tabela je dvodimenzionalni niz celobrojniih neoznačenih brojeva. Veličina niza je 1024 (4 * 256) neoznačenih celobrojnih brojeva dužine 32 bita, zauzimajući ukupno 4 kilobajta memorije.Vrednosti u referentnoj table se računaju samo jednom, na osnovu izabranog polinom generatora.

5.2 Generisani deo

Generator koda je razvijen u Python programskom jeziku, koristeći Jinja parser šablona. Jinja je jedan od najčešće korišćenih alata za parsiranje šablona za programski jezik Python [28]. Izlaz iz generatora koda su izvorni datoteke u programskom jeziku C, koje su usaglašene sa AUTOSAR i MISRA C standardima. Takođe je korištena Python biblioteka "lxml" za parsiranje XML ulazne datoteke.

Ulazni podaci za generator koda su datoteke "block_shema.xsd" i "ref_instance.xml". Datoteka "ref_instance.xml" sadrži informacije o blokovima podataka, njihovim nazivima, i veličine. Datoteka "block_shema.xsd" se koristi za validaciju "ref_instance.xml". U slučaju nevalidnih ulaznih datoteka, proces generisanja programskog koda se prekida. Ako je utvrđeno da ulazna datoteka validna u odnosu na šemu, podaci iz parsiranog modela se obrađuju, i prosleđuju se alatu za obradu šablona.

Generator koda je realizovan u datotekama:

Naziv datoteke	Opis
generator.py	Sadrži implementaciju parsiranja ulaznih datoteka, proveru validnosti ulaznih podataka, i prosleđivanje podataka alatu za obradu šablona.
run_generation.py	Skripta za pokretanje procesa generisanja programskog koda.
main.template	Šablon za generisanje "NvmGen.c" datoteke.
header.template	Šablon za generisanje "NvmGen.h" datoteke.

Tabela 5.3 Spisak datoteka korištenih za izradu generisanog dela

Datoteke "generator.py" i "run_generation.py" su Python skripte. Datoteke "main.template" i "header.template" su šabloni za Jinja parser. Šablon "main.template" se koristi za generisanje "NvmGen.c" datoteke, "header.template" šablon se koristi za generisanje "NvmGen.h" datoteke.

Čitanje i parsiranje ulaznih datoteka, provera validnosti, priprema podataka za alat za obradu šablona. Jinja parser se izvršava u "generator.py" skripti. Pokretanje skripte za generisanje se vrši pozivom "run_generation.py". Skripta se takođe može pozvati i iz komande datoteke (.bat, .sh), i tako integristi u proces generisanja porgramskog koda celoukupnog sistema, sa drugim generatorima programskog koda.

"generator.py" takođe sadrži implementaciju klase "CodeGen" sa sledećim metodama:

Naziv metode	Opis metode
def __init__(self, author)	Konstruktor klase
def generate(self, source_fileName, header_fileName , functions):	Prosleđe podatke alatu za obradu šablona.

Tabela 5.4 Spisak metoda u generatoru koda

Izlazi iz generatora koda su sledeće izvorne datoteke programskog jezika C:

Naziv	Kratak opis
NvmGen.c	Sadrži implementacije generisanih funkcije
NvmGen.h	Sadrži deklaracije generisanih funkcija

Tabela 5.5 Spisak izlaznih datoteka generisanog dela programskog rešenja

5.2.1 Provera i parsiranje ulaznih vrednosti

Izvorni programski kod se generiše na osnovu ulazne datoteke. Ulazna datoteka je u XML formatu. Ulazna datoteka se parsira korišćenjem Python lxml biblioteke [29]. Validnost ulazne datoteke se proverava na osnovu XML sheme (XSD). Ako ulazna datoteka nije validna, izvorni programski kod se ne generiše. Validna ulazna XML datoteka se parsira, i parsirane informacije se koriste kao ulazne podatke za šablone.

Provera i parsiranje ulaznih vrednosti se sastoji od sledecih funkcija:

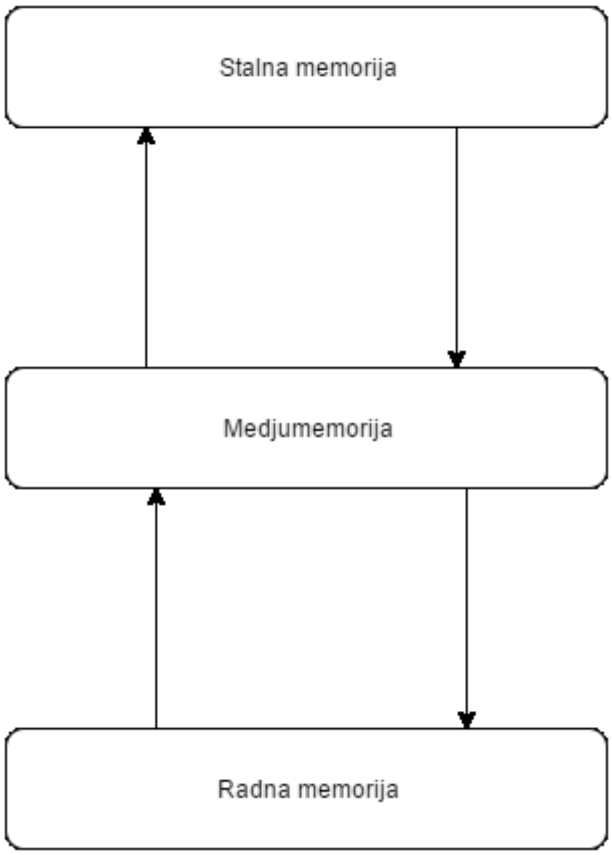
Naziv funkcije	Opis
def validate_xml(schema, modxml):	Validacija XML datoteke u odnosu na XSD datoteku.
def CreateDectriptorDict(inputDict):	Pravi rečnik vrednosti koji se prosleđuju alatu za obradu šablona

Tabela 5.6 Spisak funkcija u generisanom delu

Generisanje izvornih datoteka se vrši instaciranjem objekta *CodeGen* klase.

5.2.2 Šabloni za generisanje programskog koda

Koriste se dva šablona za generisanje, *main.template* šablona za generisanje NvmGen.c datoteke, i *header.template* šablona za generisanje NvmGen.h datoteke.



Slika 5.1 Prikaz toka podataka

Prilikom pokretanja sistema, iz stalne memorije podaci se učitavaju u međumemoriju, a iz međumemorije se kopiraju u radnu memoriju. Prilikom gašenja sistema, tok podataka je suprotan. Iz radne memorije podaci se kopiraju u međumemoriju, a iz međumemorije se smeštaju u stalnu memoriju.

5.2.2.1 Šablon za generisanje NvmGen.c datoteke

Šablon se može logički podeliti na sledeće delove:

- Generisanje zaglavlja izvornog programskog koda
- Generisanje deskriptor tabele
- Generisanje deklaracija promenjivih za međumemoriju
- Generisanje deklaracija internih funkcija za čitanje jednog bloka podataka iz stalne memorije
- Generisanje deklaracija internih funkcija za pisanje jednog bloka podataka u stalnu memoriju
- Generisanje implementacija internih funkcija za čitanje jednog bloka podataka iz stalne memorije
- Generisanje implementacija internih funkcija za pisanje jednog bloka podataka u stalnu memoriju.
- Generisanje deklaracije interfejs funkcije za čitanje svih blokova podataka iz stalne memorije
- Generisanje deklaracije interfejs funkcije za pisanje svih blokova podataka u stalnu memoriju
- Generisanje implementacije interfejs funkcije za čitanje svih blokova podataka iz stalne memorije
- Generisanje implementacije interfejs funkcije za pisanje svih blokova podataka u stalnu memoriju
- Generisanje implementacija interfejs funkcija za upis svih blokova podataka u radnu memoriju (RTE)
- Generisanje implementacije interfjes funkcija za čitanje svih blokova podataka iz radne memorije (RTE)

U zaglavlju izvornog programskog koda se generišu informacije o nazivu korišćenog šablona. vremenu generisanja, naziv generisane datoteke izvornog progamskog koda.

Deskriptor tabela sadrži informacije o svim blokova podataka, identifikator bloka podatka nazive glavne i pomoćne datoteke bloga podataka, veličinu bloka podataka, i kontrolnu CRC vrednost.

Promenjive za međumemoriju se koriste kao dodatna prihvatna memorija između stalne (NvM) i radne (RTE) memorije. Prilikom pokretanja celog sistema, blokovi podataka se učitavaju iz stalne memorije u međumemoriju. Nakon provere validnosti, iz međumemorije se upisuje u stalnu memoriju (RTE), odakle podatke mogu da koriste druge komponente sistema.

Funkcije za čitanje i upis jednog bloka podataka iz stalne memorije u međumemoriju. Generišu se na osnovu validne ulazne XML datoteke. Prilikom čitanja proveravaju validnost pročitanih blokova podataka, u slučaju da podaci u glavnoj datoteci bloka podaka nisu validni, čitaju se podaci iz pomoćne datoteke. U slučaju da su podaci trajno izgubljeni prijavljuje se greška. Prilikom upisa izračunava se kontrolna CRC vrednost, i blok podataka se upisje u stalnu memoriju.

Interfjes funkcija za čitanje svih blokova podataka iz stalne memorije obuhvataju sve generisane pojedinačne funkcije za čitanje jednog bloka podataka iz stalne memorije. Podaci se privremeno smeštaju u međumemoriju, odakle se upisuju u radnu memoriju (RTE)

Interfejs funkcija za upis svih blokova podataka u stlanu memoriju obuhvataju sve generisane funkcije za upis pojedinačnog bloka podataka u stalnu memoriju. Podaci se nalaze u međumemoriji, i odatle se upisuju u stalnu memoriju(NvM)

Interfejs funkcija za čitanje svih blokova podataka iz radne memorije (RTE) obuhvataju čitanje svih blokova podataka iz radne memorije (RTE), i kopiranje njihovog sadržaja u međumemoriju.

Interfejs funkcija za upis svih blokova podataka u radnu memoriju (RTE) obuhvataju kopiranje sadržaja svih blokova podataka iz međumemorije, i upis u radnu memoriju (RTE).

5.2.2.2 Šablon za generisanje NvmGen.h datoteke

Šablon se može podeliti na sledeće logičke delove:

- Generisanje zaglavlja izvornog programskog koda
- Generisanje definicija identifikatora blokova podataka
- Generisanje struktura za tipove podataka
- Generisanje prototipa funkcije za čitanje svih blokova podatka iz stalne memorije
- Generisanje prototipa funkcija za pisanje svih blokova podataka u stalnu memoriju
- Generisanje prototipa funkcije za čitanje svih blokova podataka iz radne memorije
- Generisanje prototipa funkcije za upis svih blokova podataka u radnu memorij

U zaglavlju izvornog programskog koda se generišu informacije o nazivu korišćenog šablona, vremenu generisanja, naziv generisane datoteke izvornog progamskog koda.

Identifikatori su celobrojni brojevi koji se koriste za proveru korektnosti prilikom čitanja i uspisa u stalnu memorijue (NvM).

Instancirane strukture tipova podataka se koriste kao međumemorija između stalne i radne memorije, kao dodatni nivo sigurnosti i provere validnosti podataka.

Prototipovi generisanih funkcija se stavljaju na raspolaganje sistemu. Sistem se brine o pravovremenom pozivanju generisanih funkcija.

6. Ispitivanje i verifikacija

Česte promene zahteva su sastavni deo tokom razvoja programske podrške. Zahtevi koje rešenje mora da zadovolji se često menjaju, i samim tim potrebne su odgovarajuće promene u izvornom programskom kodu. Cilj ovih promena je stabilnije programsko rešenje, u praksi česte promene, proces izmena i dodavanja novih funkcionalnosti je čest uzrok nastanka grešaka i nepouzdanog izvršenja programske podrške. Cilj testiranja je da se utvrdi ispravna funkcionalnost i potvrdi zadovoljenost zahteva.

Prilikom izrade rešenja ispitivanje je izvršeno na platformi sa Nvidia Tegra K1 procesorom [30], nad Linux distribucijom Jetson TK1 R21.5, sa kernel verzijom 3.10.40 [31]

Prilikom ispitivanja rešenja utvrdjeno je da u slučaju validnih ulaznih parametara, generator koda generiše ispravan C programski kod, koji je moguće iskompajlirati.

U slučaju invalidnih ulaznih parametara, generator prijavljuje grešku, i prekida se proces generisanja.

Programskim alatom za statičku analizu PRQA QA C koda je utvrdjeno da programsko rešenje ne narušava MISRA C:2004 smernice.

Izvršene su tri grupe funkcionalnih testova: grupa testova koji verifikuju ispravno čitanje podataka iz stalne memorije, grupa testova koji verifikuju ispravno upisivanje podataka u stalnu memoriju i grupa testova koji verifikuju detekciju greške u stalnoj memoriji. Opis i rezultati testova su dati u tabelama.

Naziv testa	Čitanje bloka podataka iz stalne memorije
Opis testa	Test verifikuje ispravnost pročitanih podataka iz stalne memorije nakon uključenja uredjaja
Očekivani rezultat	Podaci su uspešno pročitani
Ostvareni rezultat	Testni slučaj je zadovoljen

Tabela 6.1 Test čitanja podataka iz stalne memorije

Naziv testa	Upis bloka podataka u stalnu memoriju
Opis testa	Test verifikuje ispravnost upisanih podataka u stalnu memoriju
Očekivani rezultat	Podaci su uspešno zapisani
Ostvareni rezultat	Testni slučaj je zadovoljen

Tabela 6.2 Test upisa podataka u stalnu memoriju

Naziv testa	Čitanje invalidnih podataka iz stalne memorije
Opis testa	Test verifikuje da je greška detektovana u slučaju invalidnih podataka u stalnoj memoriji
Očekivani rezultat	Detektovana je greška, i podaci iz stalne memorije nisu prosledjeni u radnu memoriju
Ostvareni rezultat	Testni slučaj je zadovoljen

Tabela 6.3 Test detekcije greške prilikom čitanja iz stalne memorije

Takodje su izvršena ispitivanja performansi. S obzirom da u trenutku izvršenja testova nije bilo dodatnih programskih podrški koje bi pristupale stalnoj memoriji, postignute brzine čitanja i pisanja u stalnu memoriju bile su bliske maksimalnim brzinama čitanja i pisanja za dati memorijski uređaj.

Izmerene brzine čitanja i pisanja značajno se razlikuju od načina izvodjenja testova. Ukoliko su samo pozvane POSIX funkcije, rezultati merenja su značajno brži od maksimalne moguće specificirane brzine za sam memorijski uređaj. Razlog se nalazi u načinu na koji operativni sistem rukuje stalnom memorijom. Operativni sistem ne smešta podatke stalnu memoriju odmah po primljenom zahtevu, podaci se privremeno smeštaju u keš memoriju, pa cu u pozadinskom zadatku biti stvarno smeštene u stalnu memoriju. Na ovaj način operativni sistem mnogo brže funkcioniše, i izbegava se čekanje na spore ulazno izlazne operacije.

U slučaju da se od operativnog sistema eksplicitno zahteva da podatke smesti u stalnu memoriju, dobijaju se brzine bliske maksimalnim mogućim za dati memorijski uređaj.

Vreme izvršenja takođe zavisi od izračunavanja CRC vrednosti. Vreme potrebno za izračunavanje kontrolne CRC vrednosti direktno je zavisno od količine podataka. U sledećoj tabeli su prikazani rezultati merenja vremea potrebnog za izračunavanje CRC za blokove podataka različitih veličina.

Veličina bloka [byte]	Vreme [us]
512	15
20000	27
40000	41
128000	92

Tabela 6.4 Tabela sa vremenima izračunavanja CRC vrednosti

7. Zaključak

Prikazano rešenje programske podrške za trajno čuvanje sadržaja memorije je razvijeno po AUTOSAR i MISRA C:2004 standardima, i se može lako integrisati u AUTOSAR projekat, ili kao deo posebnog ECU-a.

Rešenje je moguće nadograditi dodatnim funkcionalnostima u zavisnosti od potrebe. Dodatne funkcionalnosti je potrebno implementirati u negenerisanom delu programske podrške. Sve promene u generisanom delu zavise isključivo od promene ulaznih podataka.

Provera očuvanosti integriteta blokova podataka je realizovana korišćenjem algoritama ciklične provere redudantnosti (CRC). CRC provera omogućava detekciju najčešćih grešaka koje se dogadjaju u prenosu, čitanju i pisanju podataka. Algoritmi CRC pružaju zaštitu od nepouzdanog rada izazvanih kvarom memorijskog uredjaja.

Razdvajanjem programskog rešenja na generisani i ručno pisani (negenerisani) deo se omogućava brza i laka promena izvornog programskog koda na osnovu zadatih ulaza, kao i podrška za inkrementalni i iterativni razvoj programske podrške. Generator programskog koda proverava validnost ulaznih podataka i parametara radi sprečavanja generisanja programskog koda koji nije moguće prevesti.

Ispitivanjem i verifikacijam programskog rešenja pokazano je da prikazano rešenje zadovoljava početne zahteve.

Dalji razvoj može biti u pravcu implementacije dodatnih funkcionalnosti kao što su enkripcija podataka, dijagnostičke informacije o stanju memorijskog uređaja, brojač upisa u memorijski uređaj i slično.

8. Literatura

[1] M. Jovanović *et al*, "Programska implementacija sistema za prikupljanje i obradu signala sa elektronskih kontrolnih jedinica u automobilskoj mreži," Telekomunikacioni forum TELFOR, Beograd, Srbija, 2015

[2] I. Riches, "Strategy Analytics: Automotive Ethernet: Market Growth Outlook," *IEEE SA: Ethernet & IP @ Automotive Technology Day*. Oct. 23, 2014. [Online]. Dostupno: IEEE, http://standards.ieee.org/events/automotive/2014/00_Automotive_Ethernet_Market_Growth_Outlook.pdf. [Preuzeto: Dec. 17, 2016].

[3] "Global Automotive Supplier Study 2016", Jul. 2016. [Online]. Dostupno: https://www.rolandberger.com/publications/publication_pdf/roland_berger_global_automotive_supplier_2016_final.pdf. [Preuzeto: Jan. 14, 2017].

[4] ISO/IEC 9899:1999: Programming languages - C, Dec. 1, 1999. JTC 1/SC 22/WG 14

[5] ISO 26262:2011, "Road vehicles-Functional safety," International Standardization Organization.

[6] "AUTOSAR Background", 2017. [Online]. Dostupno: http://www.autosar.org/about/basics/background/. [Preuzeto: Jan. 19, 2017].

[7] "AUTOSAR Basic Information Short Version," Dec. 17, 2016. [Online]. Dostupno: https://www.autosar.org/fileadmin/files/basic_information/AUTOSARBasicInformationShortVersion_EN.pdf. [Preuzeto: Dec. 17, 2016].

[8] Balaji Kulkarni, "AUTOSAR – Automotive Open Systems Architecture", 2012. [Online]. Dostupno: http://www.engineersgarage.com/articles/autosar-automotive-open-systems-architecture. [Preuzeto: Dec. 10, 2016].

[9] AUTOSAR , (AUTomotive Open System ARchitecture), Dec. 10, 2016. [Online]. Dostupno: http://www.autosar.org. [Preuzeto: Dec. 10, 2016].

[10] AUTOSAR Technical overview, Dec. 10, 2016. [Online]. Dostupno: http://www.autosar.org/about/technical-overview/. [Preuezeto: Dec. 10, 2016].

[11] *"MISRA C and MISRA C++ Compliance,"* 2016. [Online]. Dostupno: http://www.programmingresearch.com/coding-standards/misra/. [Preuzeto: Dec. 18, 2016].

[12] "A brief history of MISRA C," 2016. [Online]. Dostupno: https://www.misra.org.uk/Activities/MISRAC/tabid/160/Default.aspx. [Preuzeto: Dec. 17, 2016].

[13] "MISRA Compliance: 2016," Apr. 2016. [Online]. Dostupno: https://misra.org.uk/LinkClick.aspx?fileticket=w_Syhpkf7xA%3d&tabid=57. [Preuzeto: Nov. 26, 2016].

[14] "MISRA publications," 2016. [Online]. Dostupno: https://www.misra.org.uk/Publications/tabid/57/Default.aspx. [Preuzeto: Nov. 26, 2016]

[15] "Achieving MISRA C:2012 Compliance," 2016. [Online] Dostupno: https://alm.parasoft.com/achieving-misra-c-2012-compliance-with-a-development-testing-platform. [Preuzeto: Dec. 18,2016].

[16] D.C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," IEEE Computer, vol 39, no., pp. 25-31, Feb. 2006.

[17] J. Wilcox, "Paying Too Much for Custom Application Development," *blog.edgewater.com,* Mar. 18, 2011. [Online]. Dostupno: https://blog.edgewater.com/2011/03/11/paying-too-much-for-custom-application-implementation-code-generation/. [Preuzeto: Dec. 18, 2016].

[18] T. Stahl, M. Voelter, *et al.* "Code Generation Techniques" u *Model-Driven Software Development: Technology, Engineering, Management,* 2006, ch. 9, pp. 181-201.

[19] S. Mittal, J. S. Vetter, "A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems," IEEE Transactions on Parallel and Distributed Systems, vol 27, issue 5, May, 2016

[20] S. D. Lowe, "A Flash Storage Technical and Economic Primer," *flashstorage.com.* 2017. [Online]. Dostupno: http://www.flashstorage.com/flash-storage-technical-economic-primer. [Preuzeto: Jan. 7, 2017]

[21] W. W Peterson, D. T. Brown, "Cyclic Codes for Error Detection", Proceedings of the IRE, vol: 49, issue 1, Jan. 1961.

[22] T. Ritter, "The Great CRC Mystery," *Dr. Dobb's Journal of Software Tools,* Feb. 1986. pp 26-34, 76-83. [Online]. Dostupno: http://www.ciphersbyritter.com/ARTS/CRCMYST.HTM. [Preuzeto: Jan. 8, 2017].

[23] M. Stigge *et al.* "Reversing CRC – Theory and Practice," Berlin: Humboldt University, 2006. [Online]. Dostupno: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.173.2551. [Preuzeto: Jan. 8, 2017].

[24] Evropski Institut za Telekomunikacione Standarde, "Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; Part 1 (DVB-S2) - CRC-8 encoder (for packetized streams only)," *European Telecommunications Standards Institute,* EN 302 307-1 V1.4.1, 2014. [Online]. Dostupno: http://www.etsi.org/deliver/etsi_en/302300_302399/30230701/01.04.01_60/en_30230701v010401p.pdf. [Preuzeto: Jan. 14, 2017].

[25] "Specification of CRC Routines," 2016. [Online]. Dostupno: https://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/safety-and-security/standard/AUTOSAR_SWS_CRCLibrary.pdf. [Preuzeto: Nov, 5 2016]. AUTOSAR. 22 July 2015.

[26] T. Boutell, G. Randers-Pehrson, et al. *"PNG (Portable Network Graphics) Specification, Version 1.2 "*, Jul. 14, 1998. [Online]. Dostupno: https://www.w3.org/TR/PNG/. [Preuzeto: Dec. 3, 2016].

[27] M. Barr, "CRC Series, Part 3: CRC Implementation Code in C/C++", *barrgroup.com,* Jan. 1, 2000. [Online]. Dostupno: http://www.barrgroup.com/Embedded-Systems/How-To/CRC-Calculation-C-Code [Preuzeto: Nov. 19, 2016].

[28] "Jinja2," 2014. [Online]. Dostupno: http://jinja.pocoo.org/ [Preuzeto: Aug. 17, 2016]

[29] "Processing XML and HTML with Python," Jan. 8, 2017 [Online]. Dostupno: http://lxml.de/. [Preuzeto: Jan. 8, 2017].

[30] "Tegra K1 Next-Gen Mobile Processor", 2017. [Online]. Dostupno: http://www.nvidia.com/object/tegra-k1-processor.html. [Preuzeto: Jan. 15, 2017].

[31] "Linux For Tegra R21.5", 2017. [Online]. Dostupno: https://developer.nvidia.com/linux-tegra-r215. [Preuzeto Jan. 15, 2017].