Modeliranje i implementacija prilagodljivog sustava za upravljanje svjetlima vozila

Radoš, Ante

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

MODELIRANJE I IMPLEMENTACIJA PRILAGODLJIVOG SUSTAVA ZA UPRAVLJANJE SVJETLIMA VOZILA

Diplomski rad

Ante Radoš

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 21.09.2020.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Ante Radoš
Studij, smjer:	Diplomski sveučilišni studij Elektrotehnika, smjer Komunikacije i informatika'
Mat. br. studenta, godina upisa:	D-1197, 23.09.2019.
OIB studenta:	76010933713
Mentor:	Doc.dr.sc. Zdravko Krpić
Sumentor:	Dr.sc. Ivan Vidović
Sumentor iz tvrtke:	Luka Petrinšak
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Mario Vranješ
Član Povjerenstva 1:	Doc.dr.sc. Zdravko Krpić
Član Povjerenstva 2:	Izv. prof. dr. sc. Marijan Herceg
Naslov diplomskog rada:	Modeliranje i implementacija prilagodljivog sustava za upravljanje svjetlima vozila
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U okviru diplomskog rada potrebno je modelirati i realizirati prilagodljivi sustav za upravljanje svjetlima vozila. Sustav treba biti zasnovan na AUTOSAR arhitekturi što znači da sadrži odgovarajući model koji predstavlja sve gradivne jedinice podsustava, njihovu povezanost (međusobne veze) kao i sve veze ka drugim sustavima u okviru suvremenog vozila. Podsustav za prilagodljivo upravljanje svjetlima vozila treba se sastojati od komponenata za obradu senzorskih signala, centralne upravljačke komponente (koja predstavlja kontroler sa regulacijom za svjetla vozila) i aktuatorskih komponenti (s ulogom za prilagođavanje upravljačkih signala k aktuatorskim jedinicama). Realizaciju podsustava treba započeti s osmišljavanjem strukture
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	21.09.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 04.10.2020.

Ime i prezime studenta:	Ante Radoš
Studij:	Diplomski sveučilišni studij Elektrotehnika, smjer Komunikacije i informatika'
Mat. br. studenta, godina upisa:	D-1197, 23.09.2019.
Turnitin podudaranje [%]:	1

Ovom izjavom izjavljujem da je rad pod nazivom**: Modeliranje i implementacija prilagodljivog sustava za upravljanje svjetlima vozila**

izrađen pod vodstvom mentora Doc.dr.sc. Zdravko Krpić

i sumentora Dr.sc. Ivan Vidović

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj:

1. UVOD

Sustavi prilagodljivih svjetala ili AFS-i (engl. *Adaptive Front-lighting System*) koriste se u svrhu poboljšanja uvjeta vidljivosti u prometu pri noćnim uvjetima i uvjetima smanjene vidljivosti. Klasična, „stacionarna“ automobilska svjetla imaju ograničenja koja nastaju zbog same prirode njihove izvedbe. Takva svjetla se ne mogu dovoljno dobro prilagoditi dinamičkim uvjetima koji nastaju zbog kretanja vozila u prostoru ili kretanja objekata u okolini vozila, što se prvenstveno odnosi na druge sudionike u prometu. Različite izvedbe AFS-a bave se problemima poput slabog osvjetljenja cestovnih zavoja ili skretanja u raskrižju, zatim uvjeta smanjene vidljivosti (magla, kiša, snijeg), „zasljepljenja“ drugih vozila itd. AFS-i su dizajnirani tako da budu prilagodljivi, što znači da reagiraju na promjene u uvjetima u prometu, čak i kada se ti uvjeti naglo mijenjaju.

Današnji AFS-i su napredniji od nekadašnjih zbog razvoja novih tehnologija, naprednijih senzora i mogućnosti obrade veće količine podataka. Realiziraju se pomoću namjenskih ugradbenih računalnih sustava koji prikupljaju podatke iz automobila i okoline pomoću senzora te na osnovu tih podataka upravljaju radom svjetala. Aktuatori AFS-a, posebice „glavna svjetla“, također su najčešće namjenski, pa se tako pojavljuju rješenja poput pomične glave svjetla ili LED matrice svjetala. Pritom je esencijalna i uloga programske podrške putem koje se implementiraju funkcionalnosti ovakvih sustava. Cilj ovog rada je izrada AFS-a koji će implementirati najčešće korištene funkcionalnosti sličnih sustava koje će biti istražene u sljedećem poglavlju.

Radom će biti obuhvaćen cjelokupan proces razvoja programske podrške ovakvog sustava, od modeliranja programske arhitekture, povezivanja komponenti sustava i njihove konfiguracije, sve do faze simulacije i testiranja sustava. Drugo poglavlje rada obuhvaća pregled odabranih radova u ovom području, radi boljeg upoznavanja sa tematikom. U trećem poglavlju bit će data kratka teorijska podloga kako bi se pobliže prikazala tehnička rješenja koja će se koristiti. Nakon toga, u četvrtom poglavlju, prikazane su faze modeliranja i konfiguracije sustava gdje se realiziraju potrebne programske komponente i uspostavlja njihova međusobna komunikacija. Peto poglavlje uključuje fazu implementacije funkcionalnosti u model u obliku programskog koda, čime se sustav „zaokružuje“ u upotrebljivu cjelinu. Rad takvog cjelokupnog sustava će potom biti demonstriran grafički pomoću CANoe alata a ispravnost implementiranih funkcija će biti provjerena primjenom prikladnih testova, što će također biti prikazano u petom poglavlju.

2. POSTOJEĆE IZVEDBE SUSTAVA PRILAGODLJIVIH SVJETALA

Prilagodljivi sustavi za upravljanje svjetlima se razlikuju po funkcionalnostima koje nude i načinu izvedbe tih funkcionalnosti. Te razlike se odnose na odabir senzorskih i aktuatorskih komponenti sustava, ali i na izvedbu pripadajuće programske podrške. U okviru ovog poglavlja bit će predstavljene neke od postojećih izvedbi ovakvih sustava, prikazane kroz znanstvene radove i konkretna rješenja iz prakse.

2.1 Izvedba funkcionalnosti

Jedna od osnovnih funkcionalnosti svih AFS-a je horizontalna prilagodba snopova glavnog svjetla čime se osigurava bolja osvijetljenost ceste u raznim uvjetima. Dva najraširenija načina izvedbe su korištenje koračnog motora (engl. *stepper motor*) i korištenje matrice LED (engl. *Light Emitting Diode*) svjetala. U izvedbama s motorom, motor ima ulogu aktuatora za pomicanje glave svjetla koja sadrži žarulju, pri čemu se pomak računa s obzirom na senzorske parametre dobivene iz okoline vozila. U [1], [2] i [3] prikazani su primjeri izvedbe takve funkcionalnosti. Zajedničko im je korištenje koračnog motora kao aktuatora, no razlikuju se u načinu prikupljanja parametara iz okoline. U [1] koristi se ulaz sa kamere koja daje slikovni prikaz okoline vozila. Slike se obrađuju i analiziraju kako bi se dobili podaci o krivinama na cesti na osnovu kojih se može izračunati odgovarajući stupanj zakretanja koračnog motora. Ovakav način rada je funkcionalan ali zahtijeva točnu obradu velike količine podataka u kratkom vremenu što ga čini računalno „skupim“ zadatkom. U [2] kao ulazni parametar koristi se podatak o zakretanju upravljača automobila. Sličan senzorski ulaz će se koristiti i u ovome radu, samo se neće koristiti kut zakretanja upravljača, već izravno kut zakretanja kotača. Na ovaj način prima se jednostavan parametar o kretanju vozila u okolini na osnovu kojeg se može računati pomak svjetala. Rad [3] također koristi parametar kuta zakretanja upravljača, a značajan je i po tome što koristi komunikaciju putem CAN (engl. *Controller Area Network*) sabirnice, o čemu će kasnije biti riječ. Osim toga, u [3] se govori i o vertikalnom pomaku svjetala nazvanom „*pitch*“, a koji se također realizira pomoću koračnog motora kao i horizontalni pomak.

Vertikalni pomak u [3] se računa pomoću parametra nagiba vozila kojeg mjeri žiroskop vozila. Sličan princip će se koristiti i u sustavu koji će biti predstavljen u ovom radu, a služit će za prilagodbu vertikalnog kuta u odnosu na cestu pri kretanju na nizbrdicama i uzbrdicama. U tim uvjetima dolazi do pomicanja centra mase vozila pa tako fiksna svjetla na nizbrdici osvjetljavaju nedovoljnu

udaljenost jer su usmjerena prenisko, a na uzbrdici su previše usmjerena prema gore. U [4] se osim ovog razloga za korištenje vertikalne prilagodbe svjetla navodi i smanjenje odsjaja od ceste prema drugim vozačima pri različitim vremenskim uvjetima. Ta funkcionalnost neće biti obrađena u okviru ovog rada, ali treba napomenuti da je nju moguće programski dodati, dok je sama sklopovska komponenta već implementirana u sustav.

Osim dinamičkog pomaka glavnog svjetla, u AFS-e su često implementirana i statička svjetla za skretanje u oštrim zavojima (engl. *cornering lights* ili *bend lights*). U sustavu koji će biti prikazan u radu, ova funkcionalnost će biti implementirana na način da se iskorištavaju postojeća svjetla za maglu. U [5] je ova funkcionalnost izrađena na način da se intenzitet svjetla postepeno povećava kada se ono pali, odnosno smanjuje postepeno kada se ono gasi, što je kvalitetno rješenje sa sigurnosne i estetske strane zbog laganog prijelaza između stanja. U ovom radu će biti implementirana nešto jednostavnija inačica ovakvog svjetla, ali bez prilagodljivog intenziteta svjetala.

2.2 Primjeri arhitekture

AFS-i se u osnovi sastoje od senzora, aktuatora i jedinice za obradu podataka – ECU-a (engl. *Electronic Control Unit*). Senzori prikupljaju relevantne podatke iz okoline te ih prosljeđuju ECU-u koji na osnovu tih podataka upravlja aktuatorima. Razmjena podataka u elektronskim sustavima vozila se u pravilu odvija putem žičane veze (sabirnice), nekim od posebnih protokola od kojih su najčešći CAN, FlexRay, LIN i MOST. U [3] i [6] je prikazan princip komunikacije preko CAN/LIN hibridne mreže, a u ovom radu će se koristiti CAN protokol za razmjenu podataka s korisničkim sučeljem (naredbe vozača). CAN protokol je odabran jer ga odlikuje veća brzina prijenosa podataka od, primjerice, LIN-a, što je važno za sigurnosno kritičan sustav poput osvjetljenja ceste.

Slično postojećim rješenjima, fizička arhitektura sustava se sastoji od nekoliko senzorskih ulaza koji su povezani preko sabirnice sa priključcima (engl. *port*) ECU-a. ECU je preko svojih portova sabirnicom povezan s aktuatorima. Sličan sustav prikazan je odgovarajućim blokovima u [1]. Ovaj rad se prvenstveno dotiče programske arhitekture ECU-a, pa će tako njegov naglasak biti na razvoju programske podrške sustava. Arhitektura programske podrške ECU-a je na neki način preslika fizičke arhitekture, jer je unutar nje predstavljena svaka fizička komponenta svojom programskom apstrakcijom, kako je opisano u [7].

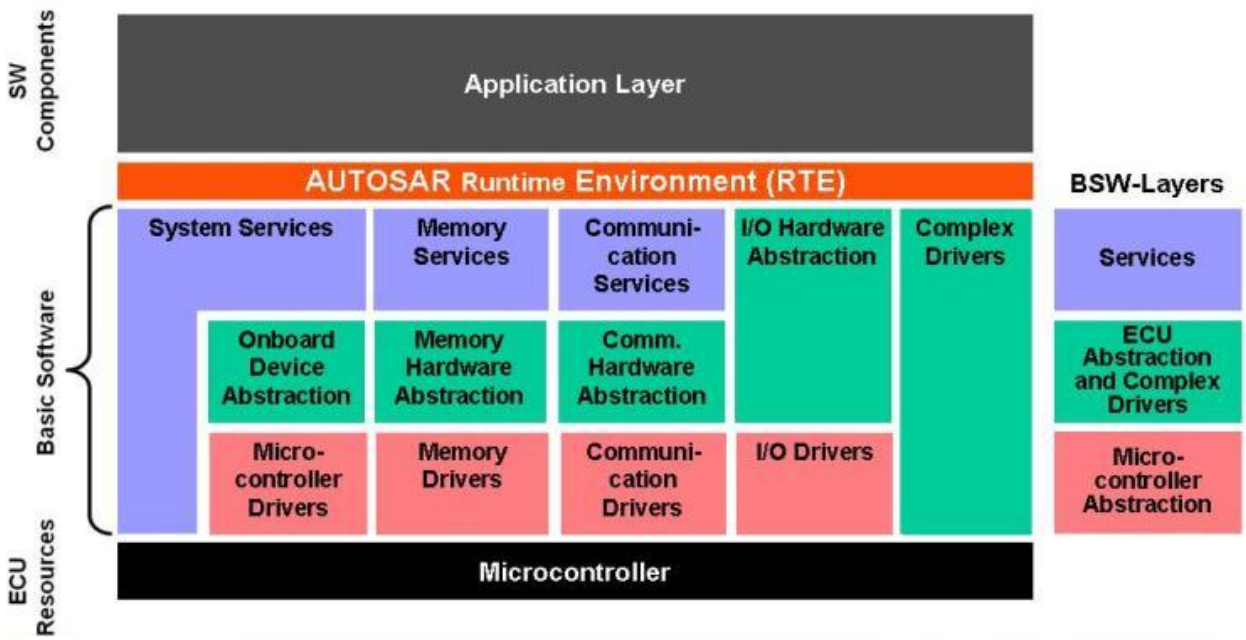
3. STANDARDI I ALATI ZA REALIZACIJU SUSTAVA

Proces izrade sustava koji će biti prikazan ovim radom zasniva se na određenim općeprihvaćenim tehnikama i pravilima prakse. Osim toga, sama izrada sustava oslanja se korištenje namjenskih programskih alata za različite faze izrade koje će biti prikazane u kasnijim poglavljima. Ovo poglavlje daje kratak pregled najvažnijih tehničkih pojmova vezanih za problematiku rada, kao i pregled osnovnih alata korištenih za njegovu izradu.

3.1 AUTOSAR

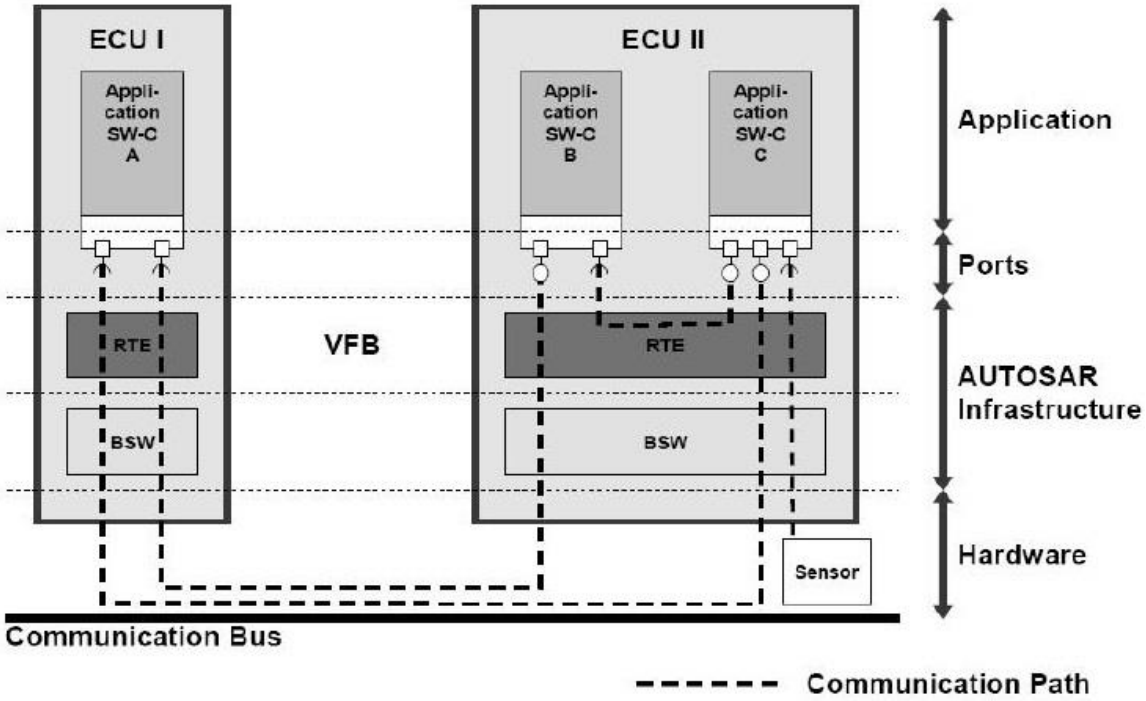
AUTOSAR (engl. *Automotive Open Systems Architecture*) je standard zamišljen od strane vodećih kompanija u automobilskoj te srodnim industrijama, započet 2003. godine. Glavni cilj AUTOSAR-a je standardizacija programske arhitekture u ugradbenim računalnim sustavima korištenim u autoindustriji i kao takav je danas široko prihvaćen u relevantnim granama industrije. AUTOSAR-om su definirana sučelja i metodologije rada vođene idejama modularnosti, skalabilnosti i odvojenosti programske podrške od sklopovlja čime se omogućava lakši i brži razvoj programske podrške.

U ovom radu su primijenjene tehnike i alati u skladu s AUTOSAR-om za izradu sustava prilagodljivih svjetala. Pojam sustav odnosi se na fizičku ugradbenu komponentu koja se u automobilskoj industriji naziva ECU, a ima funkciju upravljanja određenim podsustavom u automobilu (npr. pogon, svjetla, „*infotainment*“ itd.) uz pomoć senzora i aktuatora. Sklopovlje ECU-a ima i svoju programsku podršku. U kontekstu ovog rada izraz ECU će se uglavnom odnositi na programski dio ECU-a. AUTOSAR definira slojevitu programsku arhitekturu (Slika 3.1.) u okviru koje svaki sloj predstavlja apstrakciju nižeg sloja te se na taj način ostvaruje mogućnost razvoja programske podrške u višim slojevima neovisno o arhitekturi fizičkih komponenti.



Slika 3.1. Slojevita AUTOSAR arhitektura, preuzeto sa [8]

ECU, kako je već rečeno, predstavlja sustav zadužen za obavljanje određenih funkcija vozila, a on se, po AUTOSAR-u, sastoji od programskih komponenti SWC (engl. *Software component* - SWC). Programske komponente predstavljaju dijelove programskog koda koji obavljaju specifične zadatke koji su potrebni za izvršavanje funkcionalnosti sustava. S obzirom na to AUTOSAR definira i više tipova SWC-a, ovisno o njihovim primjenama. Važna odlika SWC-a je da su nezavisni od infrastrukture na kojoj se izvode, pa se kao takvi mogu izvršavati na različitim arhitekturama sklopovlja. Uz to, SWC-i su modularni, tj. mogu se koristiti i ponovno, primjerice u novom modelu automobila, modelima drugog proizvođača itd.. Važna značajka SWC-a je mogućnost međusobne komunikacije, kako unutar jednog ECU-a tako i s SWC-ima drugog ECU-a, a u svrhu razmjene potrebnih podataka. Ta razmjena se odvija posredstvom izvršnog okruženja (engl. *Runtime environment* - RTE), sloja koji ima ulogu programskog međusloja (engl. *middleware*) koji omogućuje komunikaciju unutar ECU-a, između ECU-ova, kao i komunikaciju SWC-a s osnovnom programskom podrškom (engl. *Basic Software* – BSW), kako je prikazano na Slika 3.2. Sva komunikacija između SWC-a se odvija putem namjenskih portova koji su podešeni da obavljaju primanje ili odašiljanje potrebnih tipova podataka.



Slika 3.2. Intra ECU i inter ECU komunikacija, preuzeto sa [9]

3.2 Programski alati i tehnike razvoja programske podrške

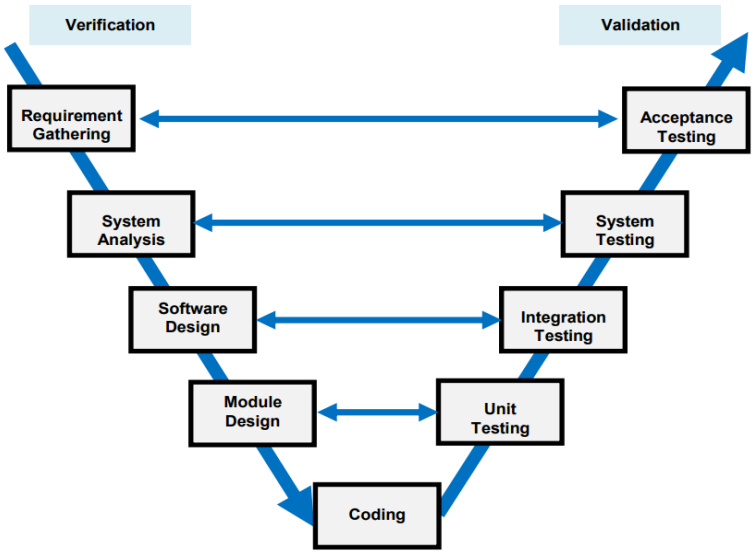
Programska podrška automobilskih ugradbenih računalnih sustava se u osnovi sastoji od različitih tipova datoteka koje se zajedno koriste za upravljanje sustavom. Teoretski je za navedene sustave moguće u cijelosti napisati programsku podršku koristeći samo alat za obradu teksta, programski prevoditelj (engl. *compiler*) i povezivač (engl. *linker*), no povećanjem složenosti sustava, kao i složenosti programske podrške, takav način rada postaje nepraktičan. Iz tog razloga su razvijeni specijalni programski alati, tzv. razvojna okruženja (engl. *Integrated Development Environment* - IDE) koja pružaju sve potrebne funkcionalnosti za razvoj sustava. Osim za razvoj programske podrške, postoje i razvojna okruženja za modeliranje arhitekture sustava, generiranje programskih datoteka, simulaciju, verifikaciju, itd.

Jedna od tehnika razvoja koja donosi poboljšanja u načinu rada naziva se MDD (eng. *Model Driven Development*). To je pristup razvoju programske podrške kojim se nastoji podijeliti programski kod na elementarne funkcionalne dijelove, odnosno modele. Na taj način se omogućava apstrakcija promatranog sustava na visokoj razini, čime se olakšava vizualizacija koda, dizajn sustava,

te djelomična automatizacija izrade programskog koda [10]. Prednost MDD-a je i mogućnost korištenja standardiziranih modela, pri čemu se mogu generirati dijelovi programskog koda koji se univerzalno koriste. Ovakav pristup je izrazito koristan u razvoju modernih ugradbenih sustava koje odlikuje visok stupanj složenosti. Modeliranje sustava se realizira pomoću programskih jezika kao što je, primjerice, XML. AUTOSAR standard definira svoju inačicu ovog jezika koja se naziva ARXML (*AUTOSAR XML*).

Uz MDD, postoji i CBD (eng. *Component-Based Developmnet*) razvojna tehnika programske podrške kojoj je glavna odlika ponovna iskoristivost programskog koda. Sustavi se dijele na nezavisne komponente koje se potom implementiraju i povezuju po potrebi. Ponovnim korištenjem programskog koda u srodnim aplikacijama ostvaruju se uštede u vremenu i novcu, kao i učinkovitost proizvodnje, jer se razvojni procesi mogu više usredotočiti na značajke samog sustava [11]. CBD se može primjenjivati na različitim razinama apstrakcije sustava, a za modeliranje se često koriste alati zasnovani na UML (engl. *Unified Modeling Language*) jeziku. Ovakvim načinom rada odlikuje se proces izrade SWC komponenti u AUTOSAR standardu.

Razvoj automobilske programske podrške je proces koji se sastoji od više faza kao što su, primjerice, definiranje zahtjeva, modeliranje sustava, generiranje koda, simulacija i testiranje. Postoji više vrsta pristupa razvojnom procesu, a u autoindustriji se najčešće koristi V-model (Slika 3.3.). V-modelom je opisan redoslijed faza razvoja programske podrške od definiranja zahtjeva do implementacije, a odlikuje ga fokusiranost na testiranje svih faza razvoja.



Slika 3.3. V-model u autoindustriji, preuzeto iz [12]

U kontekstu razvoja automobilskih ugradbenih sustava, prethodno navedeni alati i tehnike se koriste u većoj ili manjoj mjeri, ovisno proizvođaču i složenosti sustava. AUTOSAR standard nastoji iskoristiti najbolje elemente tih tehnika kako bi pojednostavio procese u autoindustriji i omogućio bolju suradnju proizvođača. Proizvođači programskih alata su prepoznali ulogu AUTOSAR-a, pa pružaju alate koji omogućuju razvoj programske podrške sukladno tom standardu. Jedan od tih proizvođača je Vector Informatik GmbH, koji je razvio programske alate koji pokrivaju cjelokupan proces razvoja programske podrške u ugradbenim sustavima u autoindustriji. Neki od tih alata su korišteni u izradi ovog diplomskog rada, a to su: DaVinci Developer, DaVinci Configurator, i CANoe.

DaVinci Developer je razvojno okruženje za dizajniranje SWC-a sukladno s AUTOSAR-om. Pruža grafičko sučelje za modeliranje i povezivanje SWC-a unutar ECU-a čime se kompletno definira programska arhitektura sustava. SWC-ima se mogu dodijeliti ulazno/izlazni portovi, i „*Runnable*“ funkcije. Nad izrađenim modelom moguće je izvršiti provjeru sukladnosti s AUTOSAR-om, kao i generiranje „kostura“ programskog koda. Rezultat rada u ovom okruženju su programske datoteke pojedinih SWC-a (.c i .h), te ARXML datoteke kojima je opisan model ECU-a.

DaVinci Configurator je alat za konfiguriranje BSW-a i RTE-a za ECU-e po AUTOSAR standardu. Pomoću njega se postavljaju parametri potrebni za povezivanje SWC-a izrađenih u DaVinci Developeru ili sličnim alatima u funkcionalan model. Unutar njega se obavlja postavljanje

značajki SWC-a poput komunikacije, dijagnostike, upravljanja memorijom itd.. Configurator izvršava validaciju sustava kako bi se pronašle eventualne greške u konfiguraciji pri čemu postoji mogućnost automatskog podešavanja pojedinih parametara. Ispravno konfiguriran ECU model moguće je generirati, čime nastaju datoteke koje povezuju SWC-e sa BSW-om, odnosno omogućuju međusobnu komunikaciju SWC-a na željeni način.

CANoe je programski alat za razvoj, testiranje i analizu individualnih ECU-a, ili mreža ECU-a [13]. Njime je moguće nadzirati sve aspekte komunikacije promatranog sustava. Omogućava praćenje poruka i signala koji se razmjenjuju u sustavu, njihovo mjerenje i grafički prikaz. Pojedine poruke i signale moguće je podesiti i simulirati po želji te promatrati ponašanje sustava. Uz to, važna je i mogućnost simulacije cjelokupnog sustava što se može realizirati unutar grafičkog okruženja pomoću interaktivnih panela. Panelima se dodijele komponente po potrebi, te se njima dodijele odgovarajući signali koji predstavljaju stvarne signale unutar sustava. Na taj način moguće je mijenjati ulazne parametre te promatrati izlaze iz sustava.

4. MODELIRANJE SUSTAVA PRILAGODLJIVIH SVJETALA

Ovo poglavlje prikazuje proces modeliranja programske arhitekture sustava. Sustav je podijeljen na komponente koje predstavljaju senzore i aktuatore, kao i dodatne komponente za obradu podataka. Opisan je način izrade komponenata te njihovo međusobno povezivanje.

4.1 Definiranje zahtjeva

U sklopu ovog diplomskog rada izrađen je prilagodljivi sustav za upravljanje svjetlima vozila, a njegove funkcionalnosti su odabrane po uzoru na postojeća rješenja. Zadatak nalaže da sustav treba biti zasnovan na AUTOSAR arhitekturi, te su navedene glavne komponente sustava – jedinice za obradu senzorskih/aktuatorskih signala i centralna upravljačka komponenta. Osim toga, zadano je i radno okruženje koje je potrebno koristiti, a ono se sastoji od programskih alata DaVinci Developer i DaVinci Configurator. Implementaciju je potrebno obaviti koristeći C programski jezik unutar Microsoft Visual Studio 2013 razvojnog okruženja, a za simulaciju je predviđen programski alat CANoe. DaVinci Developer, DaVinci Configurator i CANoe su specijalizirani alati proizvođača Vector Informatik GmbH, a služe za razvoj, testiranje i simulaciju programskih komponenti zasnovanih na AUTOSAR platformi.

Za glavne funkcionalnosti sustava odabrane su one opcije koje se u praksi najčešće koriste, a zajedno čine smislenu cjelinu koja bi mogla biti primijenjena u stvarnim situacijama sudjelovanja vozila u prometu. To je prije svega funkcionalnost prilagodljivog „dugog“ svjetla u ovisnosti o nekoliko parametara (brzina, skretanje vozila, vožnja po nagibu, položaj drugih vozila). Osim toga, odabrana je i funkcionalnost paljenja svjetala za maglu („maglenki“) pri skretanju (engl. *cornering lights*). Uz to, dodana je i mogućnost paljenja/gašenja adaptivnog sustava svjetala, odnosno dijela njegovih funkcionalnosti, prema zahtjevu vozača,. Za implementaciju navedenih funkcionalnosti utvrđeni su potrebni ulazni parametri, zamišljeni kao ulazi sa različitih senzora, a predstavljeni su unutar modela pomoću nekoliko namjenskih senzorskih komponenata. To su komponente za brzinu kretanja vozila, kut zakretanja kotača, kut nagiba vozila, instrukcije vozača te komponenta za parametre ostalih vozila u blizini, a sve su detaljnije opisane u narednom poglavlju. Osim ulaznih komponenata potrebno je bilo definirati i izlazne, odnosno one koje će predstavljati upravljanje aktuatorima. Te komponente su lijeva i desna maglenka te lijevo i desno „dugo“ svjetlo, a također su opisane u nastavku.

Sve navedene komponente izrađene su i povezane u DaVinci Developeru a u DaVinci Configuratoru podešene su njihove postavke, kao i postavke cjelokupnog sustava.

4.2 Modeliranje komponenata sustava

Senzorske komponente predstavljaju ulaze sustava, odnosno vanjske parametre mjerene fizičkim senzorima, a aktuatorske komponente predstavljaju stvarne fizičke komponente koje će obavljati određene radnje. Modeliranjem tih komponenti u DaVinci Developeru opisuju se njihove osnovne značajke, pa je svaka komponenta s pripadnim parametrima predstavljena jednim SWC-om. Prije svega je potrebno stvoriti aplikacijski tip SWC-a (engl. *Application Component Type*) za sve pojedine komponente sustava, a zatim se tako stvorene vrste instanciraju pomoću prototipa. Pri stvaranju tipa SWC-a definiraju se njegov naziv, vrsta i ostale potrebne značajke. Prikaz svih izrađenih SWC-a sustava nalazi se na Tablica 4.1.

Naziv komponente:	**Uloga:**	**Tip komponente:**
CtSaSpeed	senzor brzine vozila	*SensorActuator*
CtSaSteeringAngle	senzor kuta zakretanja kotača	*SensorActuator*
CtSaPitch	senzor kuta nagiba vozila	*SensorActuator*
CtSaOutPrm	senzor parametara okolnih vozila	*SensorActuator*
CtSaHeadlight	aktuator - „dugo“ svjetlo	*SensorActuator*
CtSaCorneringLight	aktuator - „maglenka“	*SensorActuator*
CtCddIoHwAb	apstrakcija podataka	*Complex Driver*
CtApMySwc	centralna upravljačka komponenta	*Application*

Tablica 4.1. Izrađeni tipovi SWC-a

Za sve senzorske i aktuatorske SWC-e je odabrana postavka „*Atomic*“ te tip „*SensorActuator*“ koji je u DaVinci Developeru predviđen za modeliranje senzorskih i aktuatorskih komponenti. Oznaka „*Atomic*“ predstavlja najjednostavniji SWC koji se ne može dalje razlagati [14], dok se „*Composition*“ tip sastoji od više „*Atomic*“ SWC-a te njihovih međuveza. Za aktuatorske komponente odabrana je i mogućnost višestrukog instanciranja, pa će se na taj način jednom izrađen SWC moći koristiti više puta, kao primjerice za module lijevog i desnog svjetla.

Nakon tipova SWC-a, potrebno je definirati nove aplikacijske tipove podataka, koji će biti primjenjivani za razmjenu podataka između SWC-a na aplikacijskoj razini [15]. Nazivi tipova podataka su također odabrani u skladu s AUTOSAR-om, a oznaka „*Adt*“ označava aplikacijski tip podataka (engl. *Application Data Type*). Tipovi podataka izrađeni za potrebe senzorskih SWC-a su: *AdtSpeedState, AdtAngleState, AdtPitchState, AdtIncDistState, AdtPrecDistState, AdtIncWidthState,* i *AdtPrecWidthState*. Tipovi podataka koji će biti korišteni u aktuatorskim SWC-ima su: *AdtCorneringState, AdtHorizontalState, AdtVerticalState* i *AdtLightState*.

Svaki SWC za potrebe komunikacije mora imati definirane portove, odnosno njihova sučelja. Sučelja služe za razmjenu informacija pomoću pripadajućih tipova podataka, a definiraju se kao aplikacijska sučelja priključaka (engl. *Application Port Interface*). Potrebno je odabrati tip sučelja koje se izrađuje, dodijeliti mu naziv te mu dodati prototip podatkovnog elementa (engl. *Data Element Prototype*) s pripadajućim nazivom i aplikacijskim tipom podatka koji je prethodno izrađen. Sučelja portova odgovaraju fizičkim portovima, a opisuju tip podatka koji se prenosi pojedinim portom, kao i neke dodatne informacije. Odabrani način komunikacije senzorskih i aktuatorskih SWC-a sa centralnim SWC-om je „*sender-receiver*“, pa je tako odabran tip sučelja „S/R“. Sva izrađena sučelja portova s odgovarajućim tipovima podataka prikazana su u Tablica 4.2.

Naziv sučelja	Prototipi tipova podataka	Odgovarajući "*Adt*" tipovi
PiSpeedState	*DeSpeedState*	*AdtSpeedState*
PiAngleState	*DeAngleState*	*AdtAngleState*
PiPitchState	*DePitchState*	*AdtPitchState*
PiPrmState	*DeIncDistState*	*AdtIncDistState*
	DeIncWidthState	*AdtIncWidthState*
	DePrecDistState	*AdtPrecDistState*
	DePrecWidthState	*AdtPrecWidthState*
PiCorneringState	*DeCorneringState*	*AdtCorneringState*
PiLightState	*DeLightState*	*AdtLightState*
	DeHorizontalState	*AdtHorizontalState*
	DeVerticalState	*AdtVerticalState*

Tablica 4.2. Izrađena sučelja portova

Pojedinim SWC-ima sada je potrebno dodijeliti odgovarajuće portove. Izrađena sučelja portova se instanciraju unutar svakog od SWC-a kao prototipi portova (engl. *Port prototype*) pa je, primjerice, u SWC „*CtSaSpeed"* dodana instanca sučelja „*PiSpeedState"*. Navedena instanca je nazvana „*PpSpeedState*". Svakom prototipu porta je potrebno odrediti smjer komunikacije, pri čemu se određuje da li taj port šalje podatke, prima ih, ili služi za dvosmjernu komunikaciju. Svim instancama portova u senzorskim komponentama dodijeljen je atribut „*Sender*" jer će se putem njih odašiljati podaci prema centralnom SWC-u, dok je za portove aktuatora taj atribut postavljen na „*Reciever*" zbog toga što će služiti za prijam podataka od centralnog SWC-a. Također, potrebno je podesiti inicijalne vrijednosti svim tipovima podataka koje će taj port prenositi. Inicijalne vrijednosti ovdje mogu biti zadane na više načina. Odabran je referentni tip, gdje je vrijednost zadana vanjskim konstantama koje su definirane za svaki tip podataka. U ovom slučaju to je bio tip podatka „*DeSpeedState*" kojem je dodijeljena inicijalna vrijednost 0. Ovakav postupak odrađen je na isti način za sve senzorske i aktuatorske SWC-e. Sučelja portova dodana pojedinim SWC-ima prikazana su u **Error! Reference source not found.**.

Naziv SWC-a	**Instance (prototipi) sučelja**	**Tip sučelja**
CtSaSpeed	*PpSpeedState, PpSpeedInHwAb*	***sender, client***
CtSaSteeringAngle	*PpAngleState, PpAngleStateInHwAb*	***sender, client***
CtSaPitch	*PpPitchState, PpPitchStateInHwAb*	***sender, client***
CtSaOutPrm	*PpPrmState, PpPrmStateInHwAb*	***sender, client***
CtSaCorneringLight	*PpCorneringState, PpCornerInHwAb*	***reciever, client***
CtSaHeadlight	*PpLightState, PpLightInHwAb*	***reciever, client***
CtCddHwIoAb	*PpSpeedHwAb, PpAngleStateHwAb, PpPitchStateHwAb, PpPrmStateHwAb, PpCornerLeftHwAb, PpCornerRightHwAb, PpLightLeftHwAb, PpLightRightHwAb*	***server***

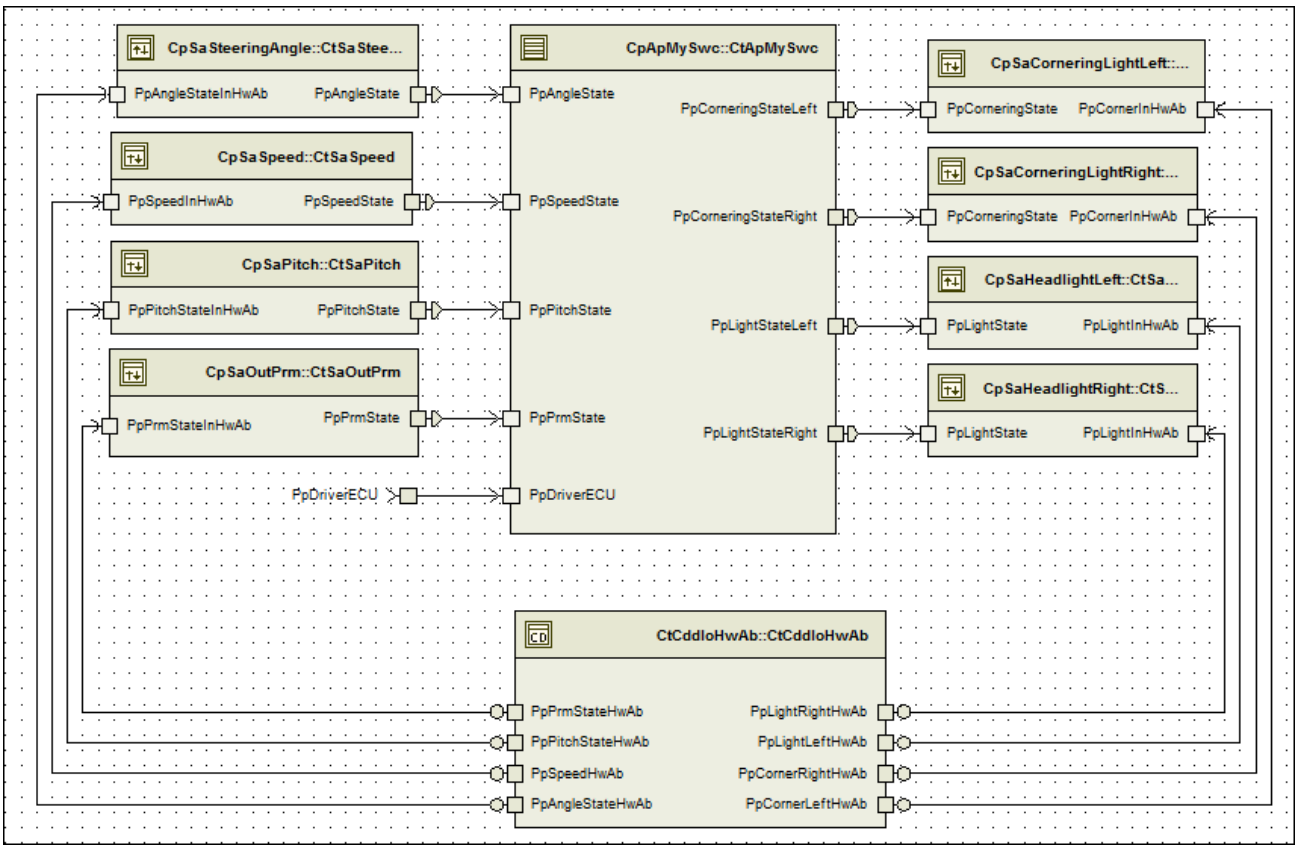
Tablica 4.3. Sučelja dodijeljena SWC-ima

Osim senzorskih i aktuatorskih SWC-a konfiguriran je još jedan SWC koji će služiti za apstrakciju podataka kojima rukuju senzori i aktuatori. Pomoću tog SWC-a će se moći simulirati ulazi

i izlazi sustava jer će pružiti funkcije za čitanje i zapisivanje podataka. Taj SWC sa popratnim portovima i tipovima podataka se konfigurira slično kao i prethodni SWC-i, no postoje neke bitne razlike. SWC je nazvan *CtCddIoHwAb*. „Cdd“ u nazivu ukazuje na to da se radi o komponenti složenog upravljačkog programa (engl. *Complex Device Driver*). Takve komponente između ostalog služe za implementaciju senzorskih i aktuatorskih signala [16], a mogu vršiti direktnu komunikaciju između RTE-a i mikroupravljača nad kojim su implementirane (Slika 3.1.).

Glavna odlika koja razlikuje *CtCddIoHwAb* od prethodno opisanih SWC-a je tip sučelja portova koje će biti korišteni. To su portovi tipa klijent/poslužitelj („C/S“) pa se i komunikacija između SWC-a u kojima su instancirani odvija po principu da jedan poslužitelj prema jednom ili više klijenata. Izrađena su dodatna sučelja portova C/S tipa, po jedno za svaki tip prethodno izrađenih „*SensorActuator*“ sučelja portova. Ta se sučelja moraju konfigurirati na način da se unutar njih definiraju programske operacije koje klijent može zatražiti od poslužitelja. Unutar C/S portova koji odgovaraju senzorskim portovima su definirane operacije čitanja („*Read*“), a za one koji odgovaraju aktuatorskim portovima definirane su operacije pisanja („*Write*“). Kako će se sve instance C/S portova koje su tipa poslužitelj nalaziti u *CtCddIoHwAb*-u, a instance tipa klijent u SWC-ima senzora i aktuatora, sada je vidljiva struktura klijent-poslužitelj komunikacije. Senzorski SWC-i će tražiti operaciju čitanja podataka iz *CtCddIoHwAb*-a, a aktuatorski će tražiti operacije zapisivanja. Na taj način će senzori i aktuatori slati i primati potrebne podatke, što će biti opisano kasnije u radu. Operacije u C/S sučeljima je potrebno detaljnije definirati na način da se odredi smjer komunikacije, tip podatka i naziv varijable koja će biti korištena za taj tip podatka u operaciji. Osim toga, definirana je i poruka pogreške sa vrijednošću koja će biti vraćena u slučaju nepravilnog izvođenja operacije. Tip podatka koji je zadan za izvršenje operacije je posebnog, implementacijskog tipa podataka (engl. *Implementation Data Type*), koji je objašnjen kasnije u radu.

Nakon što su definirani svi tipovi varijabli i sučelja portova, sučelja se instanciraju kao prototipi unutar odgovarajućih tipova SWC-a (**Error! Reference source not found.**) čime su SWC-i dovršeni i spremni za povezivanje. Povezivanje je izvršeno unutar zasebne „složene“ komponente tipa „*Composition*“, koji je već prije spomenut. Ovdje se instanciraju svi izrađeni SWC-i onoliko puta koliko je to potrebno te se njihovi portovi spajaju kako bi se dobio funkcionalan model zamišljenog ECU-a. Na Slika 4.1. prikazana je konačna shema sustava sa svim navedenim dijelovima, uz dodatak centralne upravljačke komponente *CtApMySwc* koja je opisana u nastavku.



Slika 4.1. Sastav ECU-a, SWC shema

Centralna upravljačka komponenta je SWC koji prima ulazne parametre od senzorskih SWC-a, provjerava ih, obrađuje te generira izlazne parametre koje predaje aktuatorskim SWC-ima. Ovaj SWC je nazvan *CtApMySwc,* a tipa je „*Application*“. To je aplikacijski tip SWC-a koji se za razliku od, primjerice, senzor/aktuator SWC-a, ne može ponovno koristiti u drugom ECU-u. Centralni SWC sadrži instance svih primopredajnih portova za potrebe komunikacije sa senzorima odnosno aktuatorima. Unutar centralnog SWC-a se implementiraju „Runnable“ funkcije koje obrađuju ulazne i daju izlazne parametre.

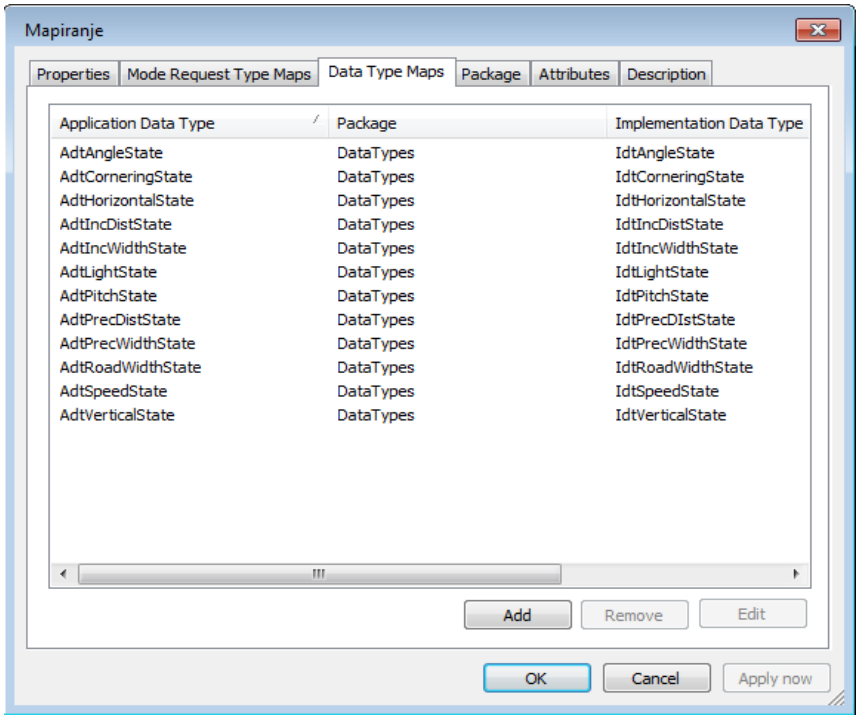
Posljednji ulaz u centralni SWC je *PpDriverECU* port. Pomoću njega je moguće slati upravljačke podatke vozača direktno u centralni SWC preko CAN sabirnice. Realiziran je kao i prethodno opisani portovi, svojim tipom i tipom podatka koji prenosi. Poseban je i po tome što se razlikuje načinom implementacije u model. Ovaj port nije povezan s nekim od senzorskih tj. ulaznih SWC-a, već postoji zasebno. To je takozvani delegirani priključak (engl. *Delegation Port*), a

implementira se na dvije razine. Instanciran je u *CtApMySwc* kao i ostali portovi, ali i u višoj razini *CtCoApplication* zbog toga što komunicira s entitetima izvan trenutno promatranog ECU-a. *PpDriverECU* je instanciran u oba slučaja kao „*reciever"*, a kako nema svoj „*sender"* par, potrebno je za obje instance desnim klikom na SWC odabrati „*Complete Ports"* i opciju „*Delegation Ports only"*. Na taj način se stvara odgovarajući par porta koji predstavlja ulaz izvan ECU-a.

4.3 Podešavanje komponenata modela

Prethodno izrađeni model opisuje strukturu sustava i međuveze komponenata, kao i korištene tipove podataka. Takav model potrebno je proširiti dodavanjem funkcionalnosti, odnosno određenih pravila ponašanja modela. Prije svega, potrebno je točnije opisati tipove podataka koji će biti korišteni u implementaciji funkcionalnosti. Do sada korišteni aplikacijski tipovi podataka su samo apstrakcije tipova podataka za upotrebu na aplikacijskom sloju i omogućuju odvojenost modela od njegove konkretne implementacije. Za implementaciju modela potrebno je definirati „*Implementation Data Type*-ove" koji se nalaze na nižoj razini apstrakcije, odnosno bliže programskom jeziku poput, primjerice, C-a [15]. Za svaki „*Adt*" tip podatka definiran je tako odgovarajući „*Idt*" tip podatka, s dodijeljenim oznakama tipa podatka (npr. *sint8, float32, ...)*. Osim oznaka tipa podatka, svakom „*Idt*" tipu podatka dodana je i „*CompuMethod*" metoda koja opisuje vezu između aplikacijskih tipova podataka i stvarnih fizičkih vrijednosti.

Kako bi model mogao biti interpretiran na nižim razinama apstrakcije, potrebno je povezati „*Application Data Type*-ove" i „*Implementation Data Type*-ove". To se čini mapiranjem, tako da se izrađuje novi „*Type Mapping Set*" koji sadržava veze između ove dvije vrste podataka. Ovaj skup je nazvan „Mapiranje", a parovi „*Adt*" - „*Idt*" tipova podataka prikazani su na Slika 4.2. Set je potom u postavkama dodan svim SWC-ima osim *CtCddIoHwAb,* kako bi bio primijenjen u implementaciji.

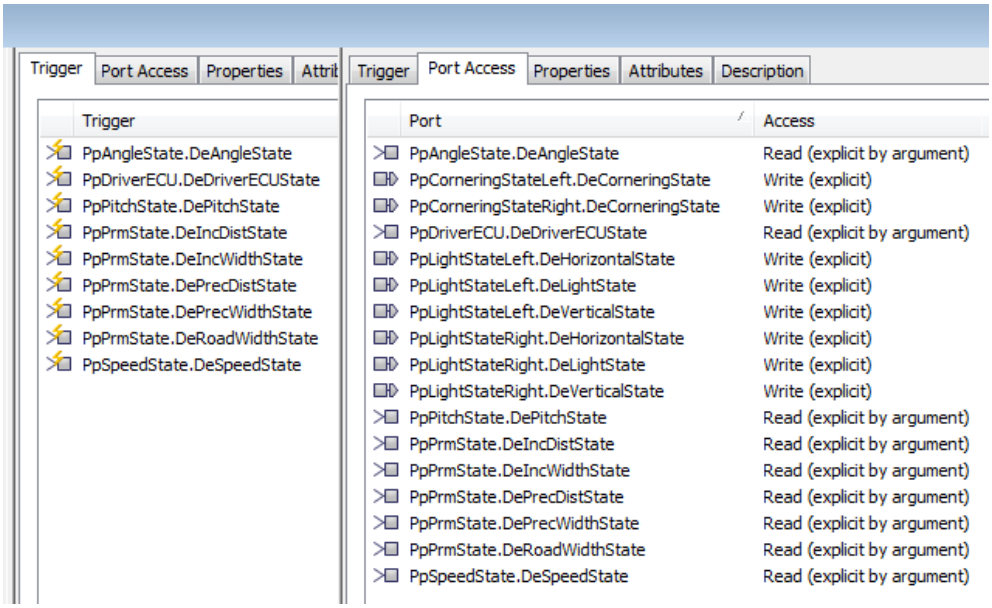


Slika 4.2. Type Mapping Set

Funkcionalnosti SWC-a predstavljene su „*Runnable*“ entitetima. *Runnable*-i su sljedovi naredbi koji izvršavaju nekakvu radnju, i svaka *Atomic* komponenta mora imati barem jedan *Runnable*. *Runnable*-e je najlakše opisati kao funkcije SWC-a koje se izvršavaju na RTE-u, međusobno su nezavisne, a aktiviraju se pomoću okidača (engl. *trigger*). Pri izradi *Runnable*-a mora mu se dodijeliti pristup portovima, odnosno tipovima podataka s tih portova koji će biti korišteni unutar *Runnable*-a. Ovisno o vrsti porta, taj pristup je za čitanje ili pisanje, a može biti implicitan ili eksplicitan. Osim toga, *Runnable* mora imati i zadan uvjet okidanja koji određuje kada će *Runnable* biti pozvan. Ti uvjeti mogu biti primitak podataka sa porta, primitak pogreške, periodičko okidanje itd.. Kod klijentskih portova, okidanje se vrši na zahtjev prema poslužitelju.

Svim SWC-ima u modelu dodijeljeni su *Runnable*-i na odgovarajući način. Na primjer, *Runnable*-i senzorskih SWC-a se okidaju svakih 100 milisekundi što je procijenjena dovoljna učestalost provjere parametra, a kod aktuatorskih SWC-a se okidaju po primitku podataka. Na taj način se svakih 100 milisekundi osvježavaju podaci sa senzora te se šalju na obradu u centralni SWC, a pri slanju tako obrađenih podataka aktuatorskim SWC-ima, okidaju se njihovi *Runnable*-i. Pri izradi *Runnable*-a u *CtCddIoHwAb* SWC-u važno je odabrati vrstu *Server Runnable* jer oni rade po nešto

drugačijem principu, jer ih se poziva po potrebi umjesto da ih pokreću okidači. Primjer postavki za *RCtSaMySwcRunnable* prikazan je na Slika 4.3.

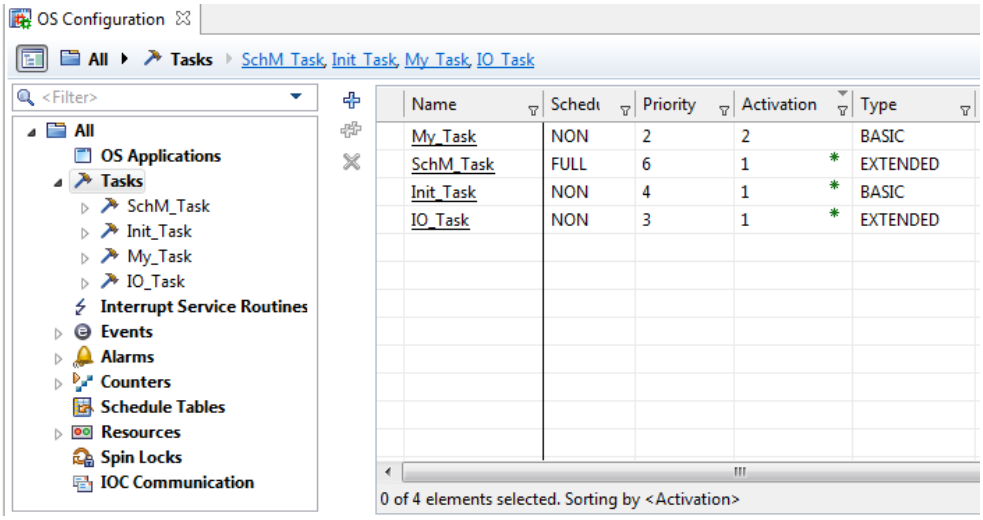


Slika 4.3. Postavke okidača i pristupa portovima

4.4 Konfiguracija modela i generiranje programskog koda

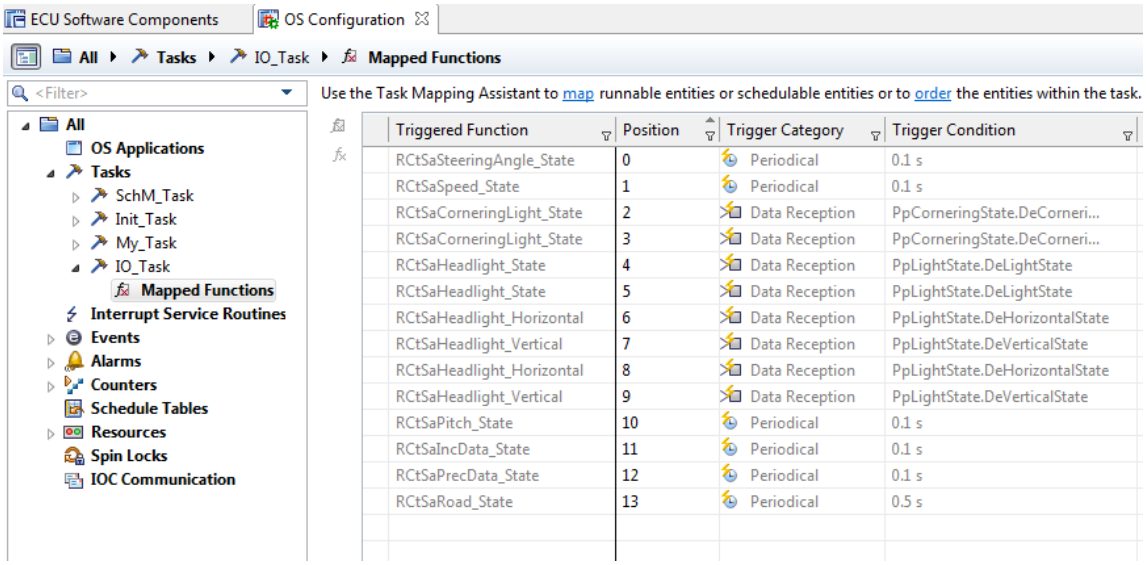
Dizajnirani ECU-u sada je potrebno dodatno konfigurirati odnosno dodijeliti mu parametre komunikacije (BSW i RTE), a to je odrađeno pomoću alata DaVinci Configurator. U njega se učitava prethodno napravljeni model ECU-a, te se vrši sinkronizacija uvezenog modela s Configurator sučeljem. Potrebno je u Configurator uvesti sve BSW module potrebne za realizaciju zamišljene komunikacije. BSW moduli su gotovi paketi koji pružaju funkcionalnosti određenog bloka AUTOSAR arhitekture, a dodaju se u „*Project Settings*“. U ovom slučaju su uvezeni moduli: *BswM, Can, CanIf, CanNm, CanSM, Com, ComM, Crc, Dem, Det, Dio, EcuC, EcuM, Fee, Fls, Mcu, MemIf, Nm, NvM, Os, PduR, VTTCan, VTTCntrl, VTTDio, VTTEcuC, VTTFls, VTTMcu* i *VTTOs*. Za rad sustava je potrebno izraditi zadatke (engl. *tasks*). Zadaci su najmanje jedinice koje operacijski sustav (engl. Operating system) može pokretati te se u okviru njih izvršavaju SWC *Runnable*-i. Pod opcijom „*Runtime System*“ su uz postojeće zadatke *Init_Task* i *SchM_Task* dodana još dva zadatka – *IO_Task* i *My_Task*. Tim zadacima su podešena svojstva koja se tiču redoslijeda njihova izvršavanja (*schedule*,

priority, *activation*), kako je prikazano na Slika 4.4. IO_Task će služiti za izvršavanje *Runnable*-a senzorskih i aktuatorskih SWC-a, dok će My_Task biti zadužen za *Runnable*-e centralnog SWC-a.



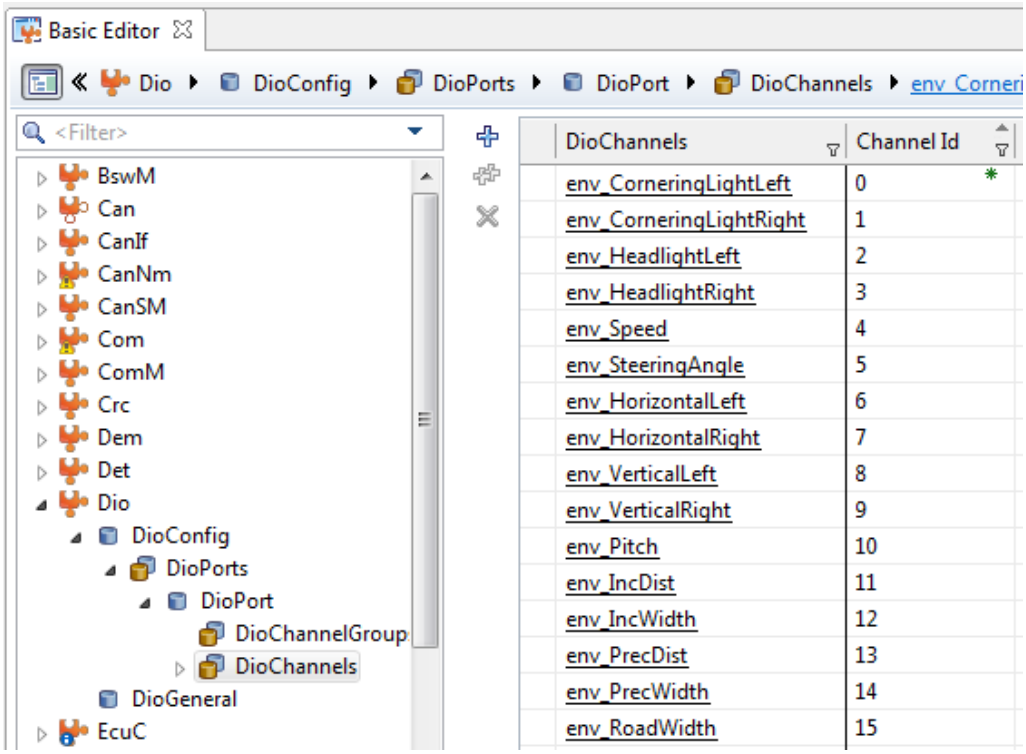
Slika 4.4. Parametri zadataka

Prethodno izrađene zadatke sada je potrebno mapirati tj. dodijeliti ih svim instanciranim prototipovima SWC-a sustava kako bi OS mogao pokretati odgovarajuće *Runnable*-e u odgovarajućim zadacima. Ovo mapiranje se postavlja pod „*Runtime System*“ opcijom za svaki *Runnable* unutar SWC-a zasebno. U ovom slučaju su svi *Runnable*-i osim onih iz *CpApMySwc* SWC-a mapirani u *IO_Task*, dok su potonji mapirani u *My_Task*. Nakon što su funkcije dodijeljene zadacima još je potrebno zadati pozicije koje uvjetuju redoslijed njihovog izvođenja u slučaju periodičkog okidanja. Prikaz mapiranih funkcija i njihovih pozicija za IO_Task nalazi se na Slika 4.5.



Slika 4.5. IO_Task

Za potrebe komunikacije *CtCddIoHwAb* SWC-a potrebno je stvoriti DIO (engl. *Digital Input/Output*) kanale. DIO kanali predstavljaju digitalne ulazno/izlazne pinove s kojih se čitaju ili pišu podaci [17]. Funkcionalnosti DIO kanala spadaju pod „*DIO driver*“ koji je jedan od modula MCAL (engl. *Microcontroller Abstraction Layer*) sloja AUTOSAR arhitekture. Ovi kanali će kasnije biti povezani sa simulatorom koji će preko njih slati i primati podatke za primjenu na senzorima i aktuatorima. Kanali se grupiraju u DIO portove, a za potrebe rada korišten je jedan port, *DioPort*, koji obuhvaća sve korištene kanale. U okviru tog porta, unutar „*Basic Editor*-a“ stvoreni su kanali za sve potrebne podatke, a definirani su svojim imenom i „*Id*“ oznakom, kako prikazuje Slika 4.6.



Slika 4.6. DIO kanali

Podešavanjem navedenih postavki omogućena je unutarnja i vanjska komunikacija ECU-a te se sada mogu generirati datoteke koje opisuju model. Te datoteke se automatski generiraju iz danog modela naredbom unutar DaVinci Configuratora.

5. IMPLEMENTACIJA FUNKCIJA I TESTIRANJE SUSTAVA

Da bi model bio u potpunosti funkcionalan potrebno je definirati funkcije koje će se izvršavati u SWC-ima. Te funkcije su napisane u C programskom jeziku unutar .c datoteka koje su prethodno generirane unutar MyECU.sln projekta. Generirane datoteke u sebi sadrže „kosture“ *Runnable*-a koje je potrebno ispuniti odgovarajućim programskim kodom. Tipovi podataka i funkcije (npr. za čitanje i pisanje) su također generirani iz izrađenog modela, a ovdje se koriste u izgradnji programske logike sustava odnosno funkcionalnosti unutar *Runnable*-a. Po implementaciji ove logike sustav će moći izvršavati sve zamišljene funkcionalnosti. Rad gotovog sustava bit će prikazan unutar simulacijskog okruženja koje će biti izrađeno i prilagođeno baš za tu svrhu. Uz to, sustav će biti testiran pomoću *unit* testova kako bi se dokazala točnost dobivenih rezultata.

5.1 Implementacija programskog koda

Za svaki tip SWC-a generirana je .c datoteka istog naziva unutar koje se nalaze njegove prazne *Runnable* funkcije. Te funkcije je potrebno popuniti odgovarajućim programskim kodom kako bi se unutar njih, odnosno na pripadajućim SWC-ima, izvršavale predviđene funkcionalnosti. Osim .c datoteka generirane su i datoteke zaglavlja (.h datoteke) u kojima se nalaze definicije gotovih funkcija koje će biti korištene u izradi koda. Programski kod i njegovo objašnjenje za svaki od SWC-a prikazani su u sljedećim točkama:

a) CtSaSpeed.c

```
1.  FUNC(void, CtSaSpeed_CODE) RCtSaSpeed_State(void) /* PRQA S 0850 */ /* MD_MSR_19.8 *
2.  {
3.      IdtDio_float32 localSpeedState;
4.      (void)Rte_Call_PpSpeedInHwAb_ReadChannel(&localSpeedState);
5.      (void)Rte_Write_PpSpeedState_DeSpeedState(localSpeedState);
6.  }
```

Slika 5.1. Funkcija *RctSaSpeed_State*

Runnable RCtSaSpeed_State (kod na Slika 5.1.) sadrži definiciju lokalne varijable *localSpeedState* tipa *IdtDio_float32* u koju se sprema vrijednost iz funkcije *Rte_Call_PpSpeedInHwAb_ReadChannel*. Ta funkcija čita vrijednost sa senzora brzine koja dolazi iz *CtCddIoHwAb* komponente. Zatim funkcija *Rte_Write_PpSpeedState_DeSpeedState* vrijednost iz *localSpeedState* prosljeđuje prema *CtApMySwc* gdje će ju moći pročitati odgovarajući *Runnable*.

b) CtSaSteeringAngle.c

Ovaj *Runnable* funkcionira kao i prethodni, na način da se u definiranu varijablu odgovarajućeg tipa sprema vrijednost kuta zakretanja kotača vozila iz funkcije koja poziva *CtCddIoHwAb* da zapiše vrijednost. Zatim se ta vrijednost prosljeđuje u *Runnable* centralnog SWC-a.

c) CtSaPitch.c

Ovo je također senzorski *Runnable*, pa obavlja radnje po uzoru na prethodna dva. Služi za dohvaćanje vrijednosti nagiba vozila sa senzora i prosljeđivanje te vrijednosti centralnom SWC-u.

d) CtSaOutPrm.c

```
1.  FUNC(void, CtSaOutPrm_CODE) RCtSaIncData_State(void) /* PRQA S 0850 */ /* MD_MSR_19.8 *
    /
2.  {
3.      IdtIncWidthState localIncWidth;
4.      IdtIncDistState localIncDist;
5.
6.      (void)Rte_Call_PpPrmStateInHwAb_ReadChannel_INCWIDTH(&localIncWidth);
7.      (void)Rte_Call_PpPrmStateInHwAb_ReadChannel_INCDIST(&localIncDist);
8.
9.      (void)Rte_Write_CtSaOutPrm_PpPrmState_DeIncWidthState(localIncWidth);
10.     (void)Rte_Write_CtSaOutPrm_PpPrmState_DeIncDistState(localIncDist);
11. }
```

Slika 5.2. - Funkcija *RctSaIncData_State*

SWC *CtSaOutPrm* sadrži više *Runnable* funkcija, a prikazan je primjer *RCtSaIncData_State Runnable*-a (Slika 5.2.). Taj *Runnable* dobavlja senzorske parametre o udaljenosti nadolazećih vozila od senzora te o udaljenosti daljeg kraja nadolazećeg vozila od osi središta ceste, nakon čega ih prosljeđuje centralnom SWC-u.

e) CtSaHeadlight.c

```
1.  FUNC(void, CtSaHeadlight_CODE) RCtSaHeadlight_Horizontal(Rte_Instance self) /* PRQA S 0
    850 */ /* MD_MSR_19.8 */
2.  {
3.      IdtHorizontalState localHorizontalState;
4.      (void)Rte_Read_PpLightState_DeHorizontalState(self, &localHorizontalState);
5.
6.      if (localHorizontalState !=0)
7.      (void)Rte_Call_PpLightInHwAb_WriteChannel_HORIZONTAL(self, (IdtDio_float32)localHor
    izontalState);
8.  }
9.
10. FUNC(void, CtSaHeadlight_CODE) RCtSaHeadlight_State(Rte_Instance self) /* PRQA S 0850 *
    / /* MD_MSR_19.8 */
11. {
12.     IdtLightState localLightState;
13.     (void)Rte_Read_PpLightState_DeLightState(self, &localLightState);
14.
15.     if (localLightState == CMHEADLIGHTSTATE_1)
16.     {
17.         (void)Rte_Call_PpLightInHwAb_WriteChannel(self, (IdtDio_sint8)localLightState);

18.     }
19.     else if (localLightState == CMHEADLIGHTSTATE_2)
20.     {
21.         (void)Rte_Call_PpLightInHwAb_WriteChannel(self, (IdtDio_sint8)localLightState);

22.     }
23.     else if (localLightState == CMHEADLIGHTSTATE_3)
24.     {
25.         (void)Rte_Call_PpLightInHwAb_WriteChannel(self, (IdtDio_sint8)localLightState);

26.     }
27.     else
28.     {
29.         (void)Rte_Call_PpLightInHwAb_WriteChannel(self, localLightState);
30.     }
31. }
32.
33. FUNC(void, CtSaHeadlight_CODE) RCtSaHeadlight_Vertical(Rte_Instance self) /* PRQA S 085
    0 */ /* MD_MSR_19.8 */
34. {
35.     IdtVerticalState localVerticalState;
36.     (void)Rte_Read_PpLightState_DeVerticalState(self, &localVerticalState);
37.
38.     if (localVerticalState != 0)
39.         (void)Rte_Call_PpLightInHwAb_WriteChannelVERTICAL(self, (IdtDio_float32)localVe
    rticalState);
40. }
```

Slika 5.3. Funkcije *RCtSaHeadlight_State Runnable*-a

U aktuatorskom SWC-u *CtSaHeadlight* nalaze se tri *Runnable*-a, prikazana Slika 5.3. *Runnable*-i *RCtSaHeadlight_Horizontal* i *RCtSaHeadlight_Vertical* čitaju parametre za horizontalni odnosno vertikalni otklon svjetla koji dolaze iz centralnog SWC-a. Te parametre predaju SWC-u *CtCddIoHwAb* koji predstavlja apstakciju sklopovlja, a u ovom slučaju će biti korišteni za potrebe

simulacije rada aktuatora. *RCtSaHeadlight_State* čita vrijednost stanja svjetla, pri čemu se misli na jedan od četiri predviđena moda rada. Ti modovi su: ugašeno (0), kratko (1), srednje (2) i dugo svjetlo (3). Nakon što od centralnog SWC-a primi vrijednost, *RCtSaHeadlight_State* pomoću naredbe uvjetnog grananja *if-else* provjerava koji od modova je potrebno postaviti, uspoređujući vrijednost s odgovarajućim konstantama. Uvjetnom naredbom se određuje koja vrijednost će se poslati prema *CtCddIoHwAb*.

f) CtSaCorneringLight.c

U ovom slučaju funkcija *RCtSaCorneringLight_State* čita i sprema parametar s ulaza u *localCorneringState* varijablu. Ako je vrijednost varijable jednaka nuli prosljeđuje „FALSE“ što označava ugašeno svjetlo (maglenku), a ako je vrijednost jednaka jedan prosljeđuje se vrijednost „TRUE“, što je vrijedost upaljenog svjetla.

g) CtCddIoHwAb.c

```
1.  FUNC(void, CtCddIoHwAb_CODE) RCtCddIoHwAb_PpAngleStateHwAb_ReadChannel(P2VAR(IdtDio_flo
    at32, AUTOMATIC, RTE_CTCDDIOHWAB_APPL_VAR) value) /* PRQA S 0850 */ /* MD_MSR_19.8 */
2.  {
3.      *value = Dio_ReadChannel(DioConf_DioChannel_env_SteeringAngle);
4.  }
5.
6.  FUNC(void, CtCddIoHwAb_CODE) RCtCddIoHwAb_PpCornerLeftHwAb_WriteChannel(IdtDio_sint8 va
    lue) /* PRQA S 0850 */ /* MD_MSR_19.8 */
7.  {
8.      Dio_WriteChannel(DioConf_DioChannel_env_CorneringLightLeft, value);
9.  }
10.
```

Slika 5.4. Primjer funkcije iz *CtCddIoHwAb*

SWC *CtCddIoHwAb* ima 16 *Runnable*-a koji se dijele na dvije vrste – za čitanje („*ReadChannel*“) i za pisanje („*WriteChannel*“). Postoji po jedan *Runnable* za svaki od tipova podataka koji se koriste na portovima ovog SWC-a. Na Slika 5.4. se nalazi po jedan primjer za svaku vrstu funkcije. Funkcije za čitanje u varijablu *value* spremaju vrijednost iz funkcije *Dio_ReadChannel*. To je funkcija iz „Dio.h“ biblioteke, a njoj se kao parametar predaje željeni DIO kanal – jedan od prethodno izrađeni kanala unutar Configuratora. Na sličan način, funkcijom Dio_WriteChannel kojoj se predaje kanal i vrijednost (*value*), se zapisuju parametri u željeni kanal. Ova komponenta putem DIO kanala komunicira sa okolinom, odnosno putem njih je moguće povezati model sa simulacijom. Preko DIO kanala se potom zadaju ulazni parametri i očitavaju izlazni.

h) CtApMySwc.c

U *CtApMySwc*-u, kao centralnom SWC-u, se nalazi glavni dio logike za obradu podataka. Parametri dohvaćeni sa senzorskih SWC-a se ovdje koriste u funkcijama za izračunavanje odgovarajućih izlaznih parametara koji se prosljeđuju aktuatorima. Sve funkcije su opisane unutar *RCtSaMySwcRunnable*-a, prikazanog na Slika 5.5. Unutar njega se deklariraju potrebne varijable i popunjavaju se pozivom *Rte_Read* tipa unkcija koje čitaju parametre s ulaznih portova.

Modovi svjetala se računaju unutar funkcije *modoviSvjetla* kojoj se predaje parametar brzine vozila. Funkcija *modoviSvjetla* kao i neke druge funkcije će biti opisane kasnije. Rezultat ove funkcije se predaje u funkciju tipa *Rte_Write* za zapisivanje stanja svjetla u aktuator.

Uvjet paljenja maglenki pri skretanju se izračunava unutar funkcije *stateCornering* koja prima parametre brzine i kuta zakretanja kotača, kao i dva dodatna parametra (pokazivači) pomoću kojih se vraćaju vrijednosti za lijevo i desno svjetlo (*stateLeft* i *stateRight*).

Horizontalni i vertikalni pomak glavnog svjetla se izračunava ukoliko je ta opcija omogućena od strane vozača („*ext_DriverECU* == 1“). U tom slučaju se pozivaju funkcije *pomakHorizontal* i *pomakVertical*. Funkcija *pomakHorizontal* prima parametre brzine, kuta zakretanja kotača, te parametre drugih vozila koja dolaze prema vozilu ili kojima se vozilo približava, a osim toga prima i dva pokazivača za spremanje povratnih vrijednosti (*stateHorLeft* i *stateHorRight*). Funkcija *pomakVertical* prima samo informaciju o nagibu vozila. Rezultati ovih funkcija se predaju funkcijama tipa *Rte_Write* kako bi se parametri dostavili aktuatorima.

U slučaju da je AFS sustav isključen („*ext_DriverECU* == 0“), parametri horizontalnog i vertikalnog pomaka se postavljaju na inicijalne vrijednosti (nema horizontalnog i vertikalnog otklona).

```
1.  FUNC(void, CtApMySwc_CODE) RCtSaMySwcRunnable(void) /* PRQA S 0850 */ /* MD_MSR_19.8 *
    /
2.  {
3.      IdtAngleState localAngleState;
4.      IdtSpeedState localSpeedState;
5.      IdtPitchState localPitchState;
6.      IdtIncDistState localIncDistState;
7.      IdtIncWidthState localIncWidthState;
8.      IdtPrecDIstState localPrecDistState;
9.      IdtPrecWidthState localPrecWidthState;
10.     uint8 ext_DriverECU;
11.     (void)Rte_Read_PpAngleState_DeAngleState(&localAngleState);
12.     (void)Rte_Read_PpSpeedState_DeSpeedState(&localSpeedState);
13.     (void)Rte_Read_PpPitchState_DePitchState(&localPitchState);
14.     (void)Rte_Read_PpPrmState_DeIncDistState(&localIncDistState);
15.     (void)Rte_Read_PpPrmState_DeIncWidthState(&localIncWidthState);
16.     (void)Rte_Read_PpPrmState_DePrecDistState(&localPrecDistState);
17.     (void)Rte_Read_PpPrmState_DePrecWidthState(&localPrecWidthState);
18.     (void)Rte_Read_PpDriverECU_DeDriverECUState(&ext_DriverECU);
19.     //***MODOVI SVJETALA***//
20.     (void)Rte_Write_PpLightStateLeft_DeLightState(modoviSvjetla(localSpeedState));
21.     (void)Rte_Write_PpLightStateRight_DeLightState(modoviSvjetla(localSpeedState));
22.     //***CORNERING LIGHTS***//
23.     IdtCorneringState stateLeft;
24.     IdtCorneringState stateRight;
25.     stateCornering(localSpeedState, localAngleState, &stateLeft, &stateRight);
26.     (void)Rte_Write_PpCorneringStateLeft_DeCorneringState(stateLeft);
27.     (void)Rte_Write_PpCorneringStateRight_DeCorneringState(stateRight);
28.     //AFS sustav
29.     if (ext_DriverECU == 1)
30.     {
31.         //***HORIZONTALNI POMAK***//
32.         IdtHorizontalState stateHorLeft;
33.         IdtHorizontalState stateHorRight;
34.         pomakHorizontal(localAngleState, &stateHorLeft, &stateHorRight, localIncDistSt
    ate, localPrecDistState, localIncWidthState, localPrecWidthState);
35.
36.         (void)Rte_Write_PpLightStateLeft_DeHorizontalState(stateHorLeft);
37.         (void)Rte_Write_PpLightStateRight_DeHorizontalState(stateHorRight);
38.
39.         //***VERTIKALNI POMAK***//
40.         (void)Rte_Write_PpLightStateLeft_DeVerticalState(pomakVertical(localPitchState
    ));
41.         (void)Rte_Write_PpLightStateRight_DeVerticalState(pomakVertical(localPitchStat
    e));
42.     }
43.     else if (ext_DriverECU == 0)
44.     {
45.         //***DEFAULT POSTAVKE***//
46.         (void)Rte_Write_PpLightStateLeft_DeHorizontalState(20);
47.         (void)Rte_Write_PpLightStateRight_DeHorizontalState(10);
48.         (void)Rte_Write_PpLightStateLeft_DeVerticalState(5);
49.         (void)Rte_Write_PpLightStateRight_DeVerticalState(5);
50.     }
51. }
```

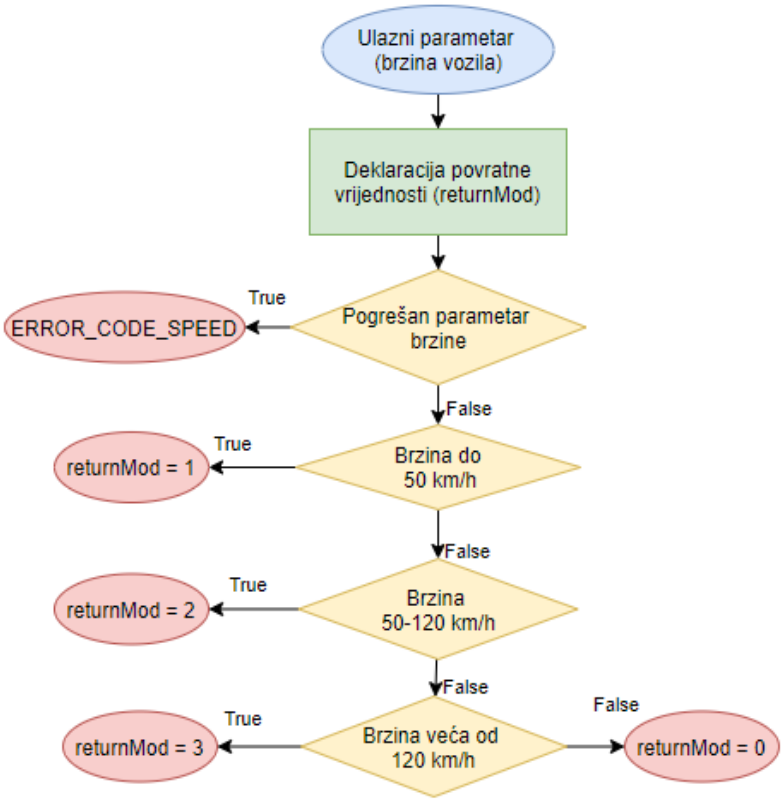
Slika 5.5. Funkcije unutar *RCtSaMySwcRunnable*-a

5.2 Funkcije za izračunavanje izlaznih parametara

Ove funkcije su korisnički definirane i pozivaju se unutar *RCtSaMySwcRunnable*-a, a to su funkcije: *modoviSvjetla, stateCornering, pomakHorizontal* i *pomakVertical*. Uz njih, izrađena je i pomoćna funkcija *incprecCoef*. Ove funkcije se nalaze unutar Functions.c datoteke za koju je izrađena odgovarajuća biblioteka Funkcije.h. Unutar funkcija se obrađuju ulazni parametri sa senzora kako bi se dobili izlazni parametri koji će uvjetovati djelovanje aktuatora u skladu sa zamišljenim funkcionalnostima.

a) modoviSvjetla

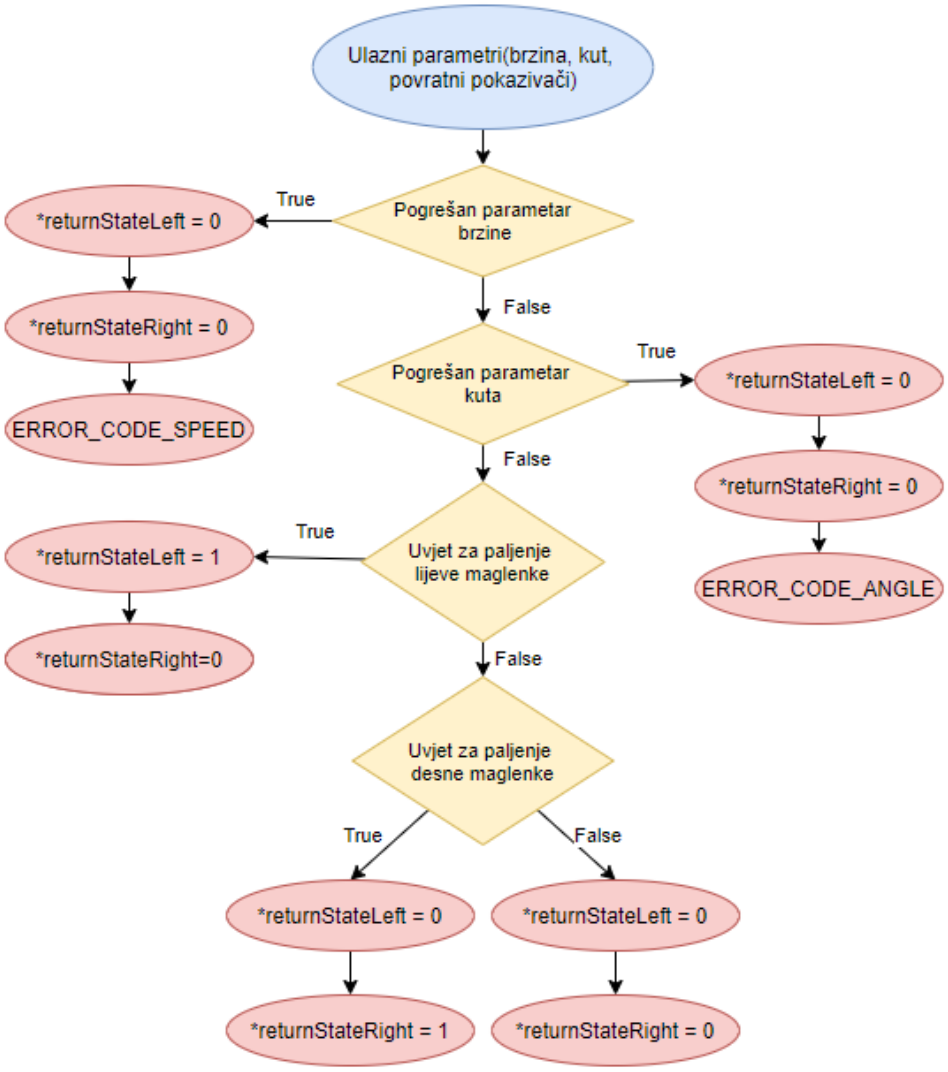
Ova funkcija izračunava stanje (mod) glavnog svjetla. Stanje se mijenja ovisno o brzini kretanja vozila, pa je tako za vozilo u mirovanju svjetlo ugašeno (0), za brzine do 50 km/h upaljeno je kratko svjetlo (1), od 50 do 120 km/h srednje dugo (2), a iznad 120 km/h dugo svjetlo (3). U slučaju primitka neprikladnih vrijednosti funkcija će vratiti kod pogreške „ERROR_CODE_SPEED“. Tok funkcije prikazan je dijagramom na Slika 5.6.



Slika 5.6. Tok funkcije modoviSvjetla

b) stateCornering

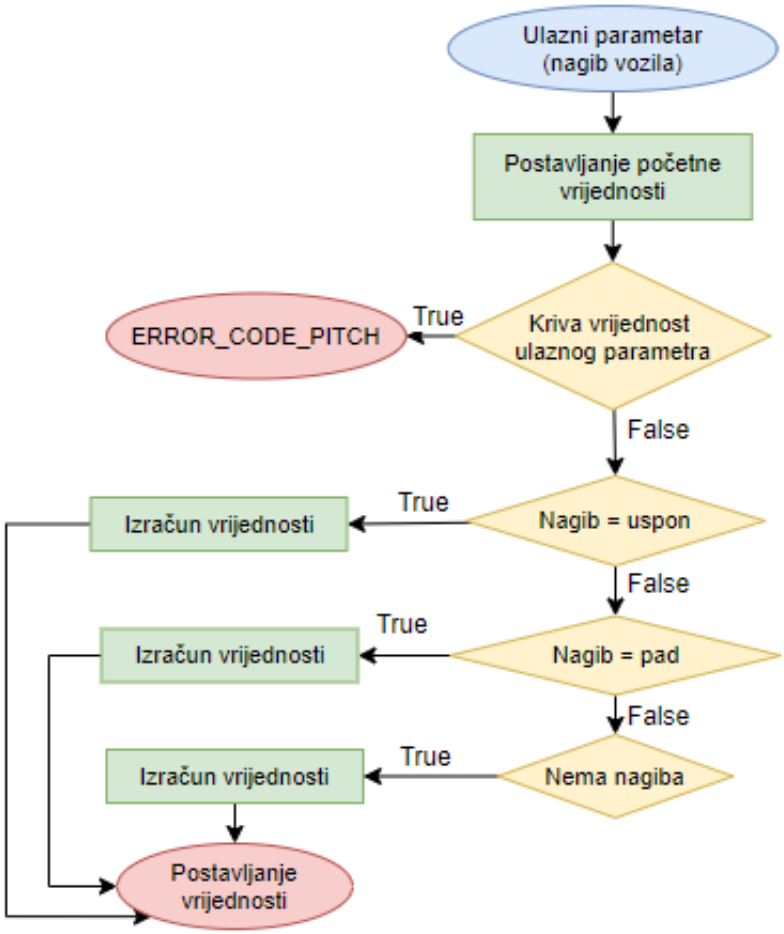
Funkcija *stateCornering* prihvaća parametre brzine i kuta zakretanja kotača, pomoću kojih izračunava vrijednosti parametara za lijevo i desno svjetlo („maglenka“). Samo paljenje maglenki pri skretanju je zamišljeno da radi na brzinama između 2 i 50 km/h, te pri kutu zakretanja kotača od 20 do 40 stupnjeva na bilo koju stranu. Samo svjetlo ima moguća stanja ugašeno (0) i upaljeno (1). Tok funkcije prikazan je na Slika 5.7.



Slika 5.7. Tok funkcije stateCornering

c) pomakVertical

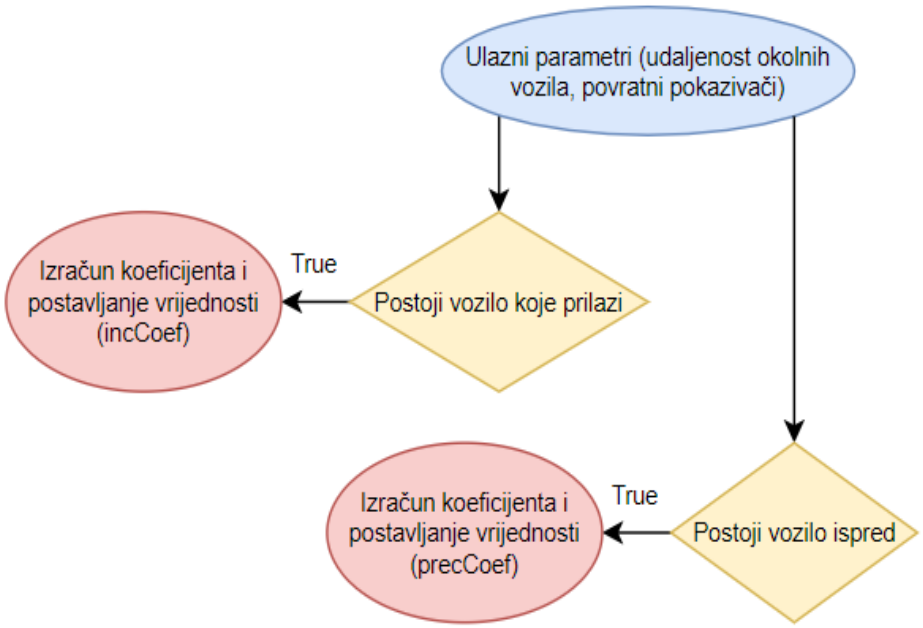
U funkciji *pomakVertical* proračunava se vertikalni pomak glavnog svjetla. Ona kao parametar prima vrijednost nagiba vozila, a u ovisnosti o njemu vraća vrijednost vertikalnog pomaka aktuatora svjetla u stupnjevima. Nagib vozila ovdje podrazumijeva slučaj kada vozilo ide niz ili uz kosinu, odnosno rotira se oko poprečne osi. Predviđene su maksimalne vrijednosti od 20 stupnjeva u oba smjera. Tok funkcije prikazan je na Slika 5.8.



Slika 5.8. Tok funkcije pomakVertical

d) incprecCoef

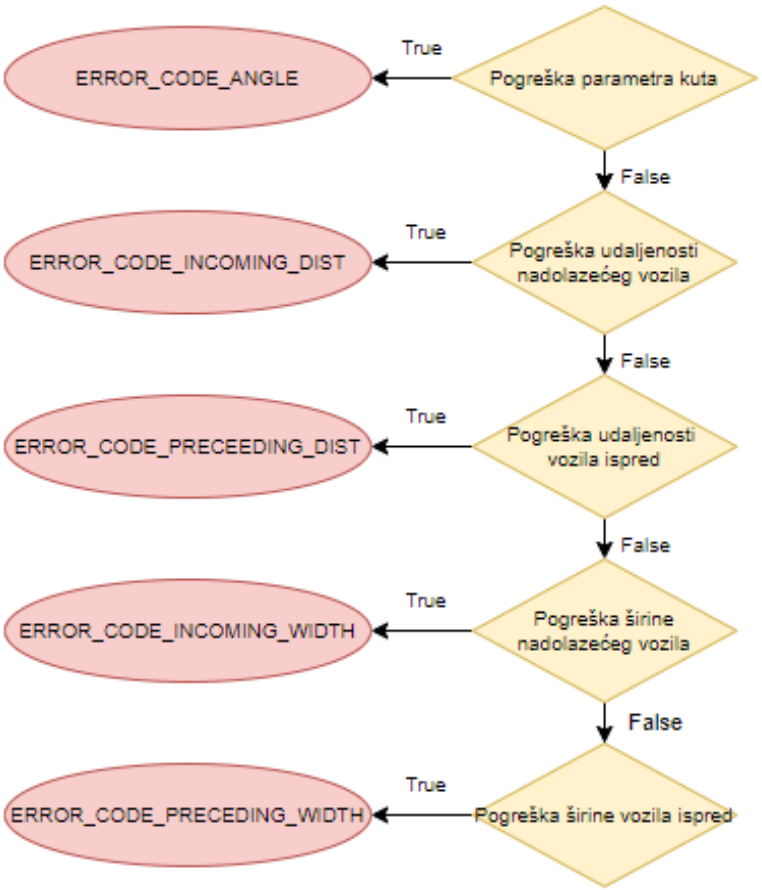
Funkcija *incprecCoef* je pomoćna korisnička funkcija koja prima parametre udaljenosti drugih vozila od referentnog, te udaljenosti najdaljeg dijela tih vozila od sredine ceste. Unutar nje se izračunavaju pomoćni koeficijenti pomoću kojih će se određivati koje vozilo snop svjetla treba „pratiti“. Tok funkcije prikazan je na Slika 5.9.



Slika 5.9. Tok funkcije incprecCoef

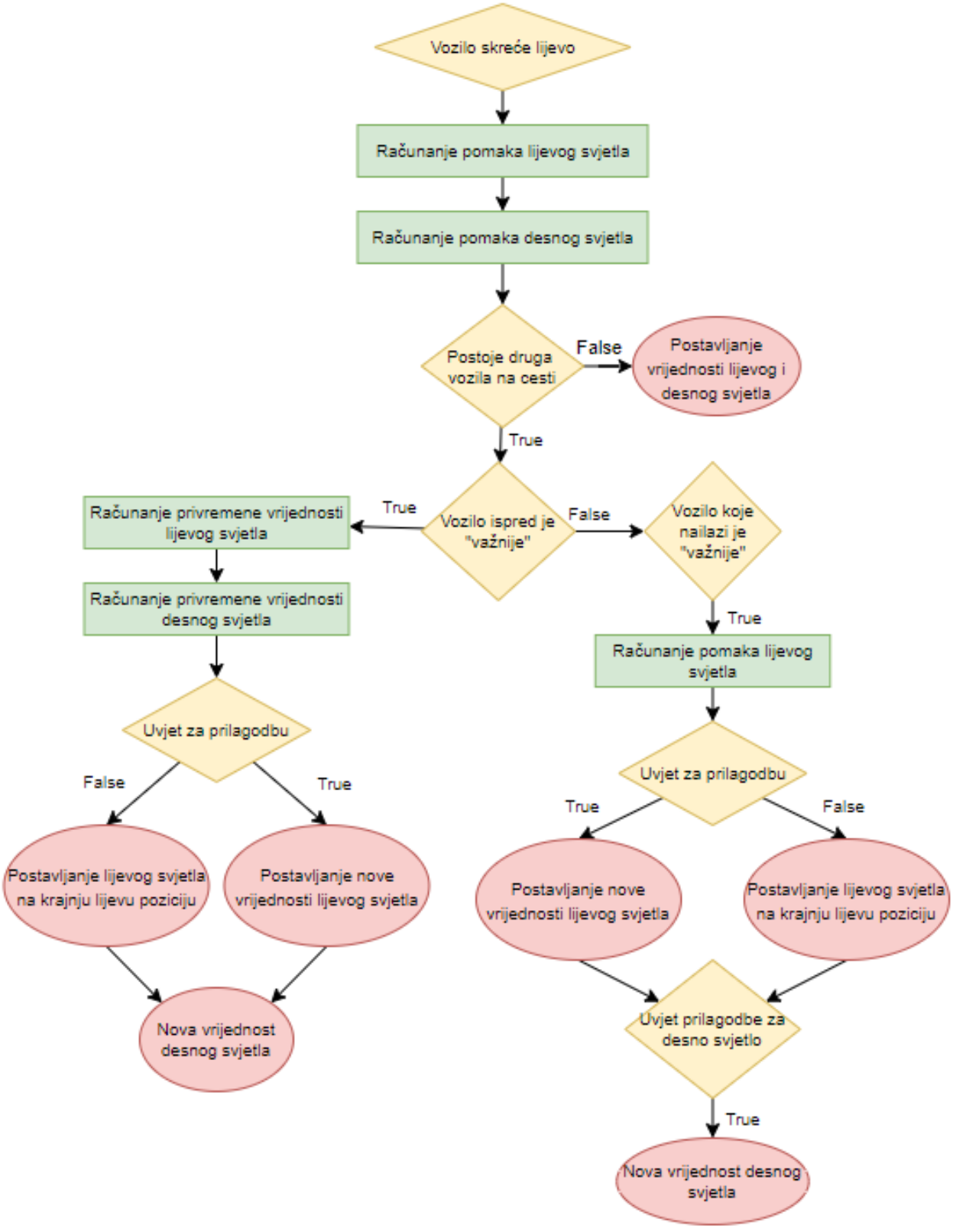
e) pomakHorizontal

Funkcija *pomakHorizontal* najsloženiji je dio ovog sustava. Unutar nje je implementirana logika horizontalnog pomaka aktuatora svjetla s obzirom na kut skretanja, ali i položaj drugih vozila. Funkcija prima parametar kuta zakretanja kotača kao i sve relevantne podatke o ostalim vozilima, a vraća vrijednosti horizontalnog pomaka putem pokazivača *stateHorL* i *stateHorR*. Prije svega, traže se moguće greške u primljenim vrijednostima za sve parametre, a nakon toga kreće izračun parametara pomaka. Uvjeti izračuna pogreške prikazani su na Slika 5.10.



Slika 5.10. Tok podataka za izračun pogreške

Sam izračun izlaznih parametara se može podijeliti na dio za lijevo i desno skretanje. Sustav je izveden tako da pri zakretanju kotača prilagođava kut osvjetljenja ceste horizontalnim pomakom aktuatora svjetla. Ta prilagodba rezultira boljim osvjetljenjem ceste u krivinama, odnosno pokrivanjem veće površine ceste snopovima svjetala. Osim toga, implementirana je funkcionalnost koja prilagođava kut osvjetljenja s obzirom na druga vozila. Svrha te funkcionalnosti je izbjegavanje „zasljepljenja“ drugih vozača snopom svjetlosti, pogotovo kada se radi o „dugom“ svjetlu. Unutar funkcije se tok algoritma grana na slučaj kada vozilo skreće u lijevo i slučaj kada vozilo skreće u desno. Za primjer će biti prikazan tok funkcije za slučaj skretanja u lijevo, dok je za slučaj skretanja u desno algoritam izrađen na analogni način. Tok funkcije za slučaj lijevog skretanja prikazan je na dijagramom toka na Slika 5.11.



Slika 5.11. Tok funkcije pomakHorizontal za lijevo skretanje

Pomak svjetala u odnosu na druga vozila se izračunava unutar pripadajućih blokova koda usporedbom *incC* i *precC* koeficijenata koji se računaju pomoćnom funkcijom *incprecCoef*. Prije toga se detektira ima li uopće drugih vozila. Kada su ti koeficijenti jednaki snop svjetla neće smetati niti jednom od vozila. Sustav najprije prilagođava svjetla ovisno o vozilu ispred sebe, na način da se i lijevo i desno svjetlo pomiču prema van, svako svjetlo na svoju stranu. Taj pomak se izračunava s obzirom na relativnu udaljenost vozila i njegovu širinu. Ukoliko je *incC* veći od *precC* snop svjetla će zaslijepiti nadolazeće vozilo pa je shodno tome potrebno prilagoditi pomak lijevog svjetla. Za slučajeve kada vozilo skreće u lijevo i ima nadolazeće vozilo na maloj udaljenosti (~50 metara) potrebno je prilagoditi izračun kako ne bi došlo do pogrešnog prikaza na simulaciji (blokovi „Uvjeti za prilagodbu“ na Slika 5.11.).

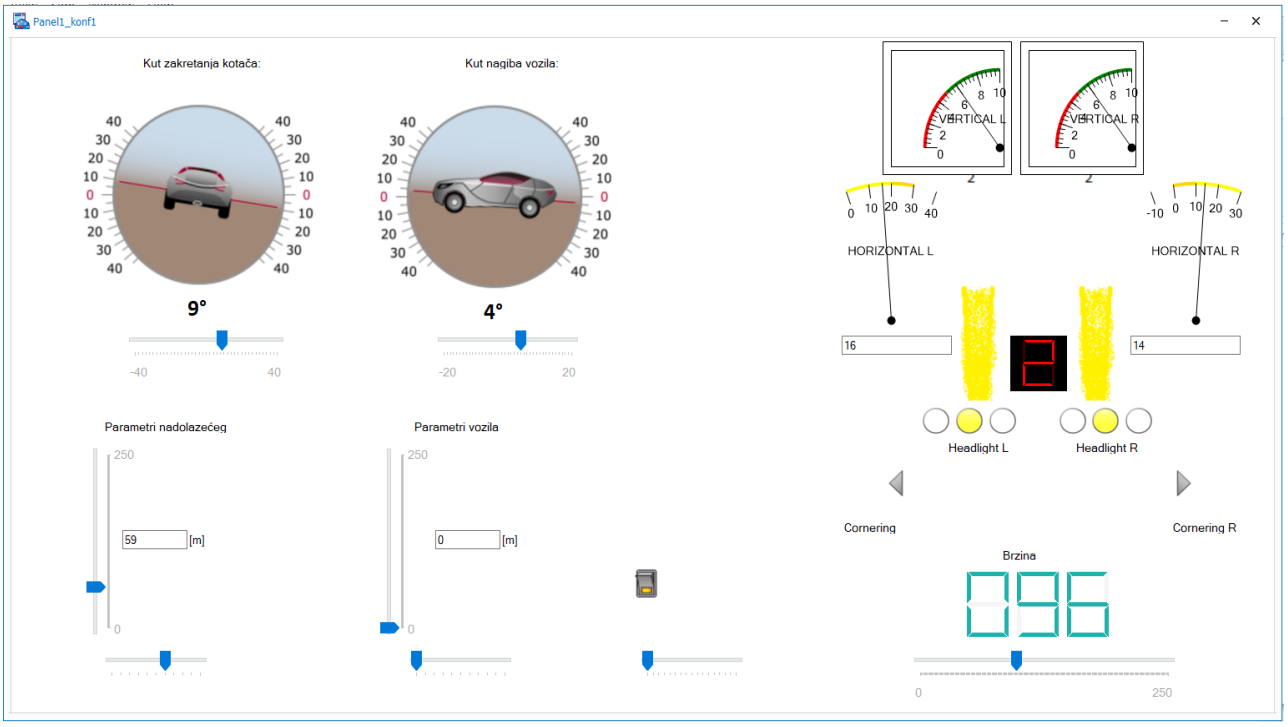
5.3 Simulacija sustava

Nakon što su u model sustava dodane sve njegove funkcionalnosti predviđene zadatkom moguće je demonstrirati njegov rad. Sustav je prikazan pomoću alata CANoe unutar kojeg je moguće simulirati ponašanje mreže kojom se vrši prijenos i upravljanje ulazno/izlaznim parametrima sustava. Simulacija se ovim alatom može izvršavati nad stvarnim sklopovskim komponentama sustava (fizički ECU i senzori/aktuatori) ili nad programskim opisom ponašanja sustava (njegov programski model). Za potrebe ovog rada izvedena je simulacija izrađenog modela sustava, a koji bi u praksi bilo moguće implementirati na sklopovlje te dobiti jednake rezultate.

Prvi korak u izradi simulacije je izrada baza podataka tipa „.dbc“ u CANdb++ alatu. Ove baze daju opis elemenata mreže u obliku mrežnih čvorova, poruka na mreži te varijabli i mrežnih signala. Izrađene su dvije baze („*env.dbc*“ i „*My_Network.dbc*“) prema gotovom CAN predlošku pošto je CAN prethodno odabran kao protokol koji će biti korišten za komunikaciju. U „*env.dbc*“ bazi su definirane varijable okoline (engl „*environment variables*“) koje su apstrakcija stvarnih varijabli sustava, a najvažnija je „*env_DriverComm*“. Ova varijabla će sadržavati vrijednost korisničkog unosa odnosno stanje AFS sustava koje zadaje korisnik (vozač automobila), a može poprimiti vrijednosti 1 (upaljen) i 0 (ugašen). Baza „*My_Network.dbc*“ sadrži mrežne čvorove *MyECU* i *DriverECU* koji predstavljaju ECU AFS sustava i ECU za korisnički unos. Izrađen je signal „sig_AFS_ON“ koji će prenositi vrijednosti 1 (upaljen AFS sustav) ili 0 (ugašen AFS sustav). Taj signal je smješten unutar poruke „*msg_AFS_State*“ koja će ga prenositi kroz mrežu. Poruke i signali su zatim mapirani na mrežne čvorove na način da su dodani u „Tx“ (odašiljanje) kategoriju na čvoru *DriverECU*, odnosno u „Rx“ (primanje) kategoriju na čvoru *MyECu* .

Za potrebe simulacije izrađena je nova CANoe konfiguracijska datoteka pod imenom „*Konfiguracija_1.cfg*“. U nju su dodane prethodno napravljene opisne .dbc baze mrežnih elemenata. Konfiguracijska datoteka je zatim povezana s modelom sustava u *vVIRTUALtarget* alatu. U DaVinci Configuratoru su sada vidljivi signali iz mrežnih datoteka koje je potrebno mapirati na odgovarajuće portove, pa je tako signal „sig_AFS_ON“ mapiran na *PpDriverECU* port. Ovim povezivanjem je simulatoru omogućen pristup varijablama modela, odnosno komunikacija s modelom sustava u svrhu simulacije njegova rada.

Za potrebe grafičkog prikaza izrađeno je sučelje („panel“) na koje su dodani interaktivni prikazi parametara sustava. Senzorski ulazi su predstavljeni klizačima na kojima je moguće podešavati ulazne parametre. Izlazi aktuatora predstavljeni su kazaljkama koje opisuju kut te svjetlosnim signalima. Svakom prikazu senzora/aktuatora dodijeljen je odgovarajući DIO kanal (izrađeni kanali s predznakom „env_“). Prethodno je, u potpoglavlju 4.4, opisana izrada tih kanala a sada su oni povezani sa simulacijskim sučeljem. Putem njih će se prenositi parametri senzora/aktuatora sustava između programskog koda sustava i simulacijskog sučelja. Izgled simulacijskog sučelja prikazan je na Slika 5.12Slika 5.12. Na toj slici je prikazan rad sustava s upaljenim AFS načinom rada gdje su vidljive postavke ulaznih parametara. Kazaljke „HORIZONTAL L“ i „HORIZONTAL R“ prikazuju horizontalni otklon svjetala u stupnjevima pri čemu valja ponoviti da je 20 „nulta“ vrijednost lijeve kazaljke, dok je za desnu to 10. Vertikalni otklon je prikazan kazaljkama „VERTICAL“ gdje je „nulta“ vrijednost jednaka 5. Stanje glavnog svjetla je 2 („srednje dugo“), što odgovara postavljenoj brzini od 96 km/h. Svjetla za skretanje u ovom slučaju nisu upaljena jer nije zadovoljen potreban uvjet.



Slika 5.12 – Grafičko sučelje simulacije

Prikazana simulacija u interaktivnom načinu rada daje rezultate u stvarnom vremenu i ima zadovoljavajući odaziv na unos, odnosno radi „uglađeno“. Isključenjem AFS sklopke gasi se mogućnost horizontalnog pomaka svjetala te se njihove vrijednosti postavljaju na „nulte“ – 20 i 10.

5.4 Testiranje programskog koda

Osim provjere ponašanja sustava pomoću simulacije izvedena je i provjera ispravnosti programskog koda metodom *unit* testiranja. Ta metoda se koristi za provjeru ponašanja programskog koda na način da se uspoređuju stvarne vrijednosti dobivene izvođenjem koda sa očekivanim (pretpostavljenim) vrijednostima za iste ulazne parametre. Testiranje se odvija nad dijelovima koda od interesa, a provodi se pomoću posebnog koda (*unit* test) koji provjerava željene uvjete.

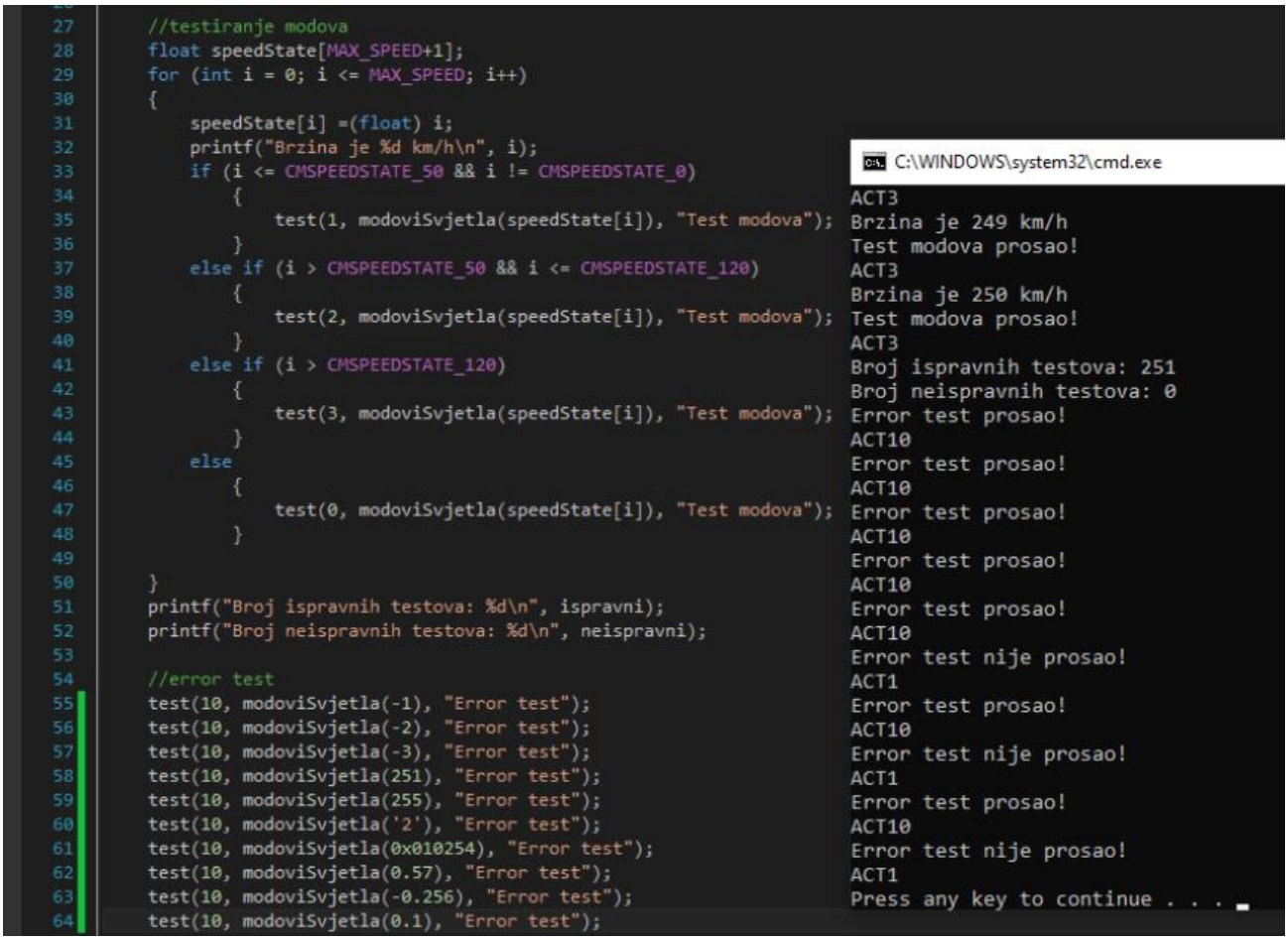
Testiranje je obavljeno nad funkcijama opisanim u potpoglavlju 5.2 koje su korisnički definirane. Za ostale, generirane funkcije, smatra se da su ispravne pa testiranje nije bilo potrebno. Za testiranje funkcija je izrađena datoteka „*UnitTestTest.c*“. U njoj je definirana funkcija *test* kojom su definirani uvjeti *unit* testa. Funkcija *test* (Slika 5.13.) prima očekivanu vrijednost izlaza (*exp*) testirane funkcije te stvarnu vrijednost (*act*). Vrijednosti *exp* i *act* se unutar funkcije uspoređuju i ukoliko su jednake test se smatra uspješnim, a u suprotnom je neuspješan. Za svaki testirani slučaj funkcija izvršava ispis rezultata testa (uspješan ili neuspješan), kao i stvarnu vrijednost testiranog slučaja.

```
1.  void test(int exp, int act, const char* testName)
2.  {
3.      if (exp == act)
4.      {
5.          printf("%s prosao!\n", testName);
6.          printf("ACT%d\n", act);
7.          ispravni++;
8.      }
9.      else
10.     {
11.         printf("%s nije prosao!\n", testName);
12.         printf("ACT%d\n", act);
13.         neispravni++;
14.     }
15. }
```

Slika 5.13. Funkcija *test*

Sve korisničke funkcije su testirane pojedinačno, prilagođeno očekivanim uvjetima svake od njih. Testirani su „rubni slučajevi“ odnosno granične vrijednosti ulaznih parametara jer se u takvim uvjetima često mogu dogoditi pogreške tijekom izvršavanja funkcija. Osim toga, funkcije su testirane normalnim vrijednostima parametara očekivanim u ispravnom radu sustava te pogrešnim

vrijednostima koje se ne bi trebale pojavljivati pri radu sustava. Na Slika 5.14. prikazan je primjer testiranja za funkciju *modoviSvjetla* gdje je ona testirana na normalne i rubne ulazne parametre (0-250). Naknadno su izvršeni testovi na neočekivane ulazne parametre gdje je rezultat uspoređivan s vrijednošću 10, što je predviđena povratna vrijednost ove funkcije u slučaju pogreške (navedeno u 5.2.a).



Slika 5.14 – Primjer testiranja funkcije *unit* testom

Izvedbom *unit* testova na svim korisničkim funkcijama pokazalo se da se sve funkcije ponašaju kako je predviđeno u očekivanim i neočekivanim uvjetima, odnosno da je sustav ispravno postavljen. Sve funkcije po izvedbi vraćaju predviđene vrijednosti za očekivane vrijednosti ulaznih parametara, a u slučaju neočekivanih vrijednosti vraća se vrijednost poruke upozorenja.

6. ZAKLJUČAK

Sustavi za upravljanje svjetlima vozila jedan su od važnijih elemenata sigurnosti u prometu, a u današnje vrijeme sve se više koriste prilagodljivi sustavi svjetala. Takvi sustavi reagiraju na promjene u okolini vozila kako bi pružili optimalnu vidljivost u svim uvjetima. U ovom radu je razvijen sustav prilagodljivih svjetala u koji su implementirane neke od najčešće korištenih funkcionalnosti u sličnim modernim sustavima kako bi se dobio cjelovit sustav, a njegov razvoj je obrađen kroz faze modeliranja, konfiguracije i implementacije funkcija. Sustav je izrađen po AUTOSAR standardu koji je danas *de facto* standard u autoindustriji, koristeći programske alate koji podržavaju takav način rada.

S obzirom na slične primjere iz prakse, kroz poglavlja ovog rada osmišljeno je vlastito rješenje koje implementira često korištene funkcionalnosti AFS-a. Te funkcionalnosti su pomična glavna svjetla vozila i svjetla za oštre zavoje. Za potrebe navedenih funkcionalnosti osmišljena je arhitektura sustava koja se sastoji od aktuatora, upravljačke jedinice i senzora potrebnih za primanje relevantnih parametara iz okoline. U skladu sa zamišljenim funkcionalnostima pomoću alata DaVinci Developer izrađen je model cijelog sustava podijeljen na sastavne dijelove u obliku programskih komponenti. Postavke modela su dodatno podešene unutar DaVinci Configurator alata čime je stvorena osnova za ostvarenje komunikacije komponenata, a nakon toga automatski generirane programske datoteke svih komponenata sustava . Tako stvorene programske datoteke su zatim popunjene funkcijama koje čine programsku logiku ponašanja sustava, odnosno uvjetuju ponašanje aktuatora s obzirom na ulazne parametre. Funkcije su napisane u C programskom jeziku, a obavljaju radnje vezane uz čitanje parametara sa senzora, njihovu obradu te upravljanje aktuatorima. Implementacijom ovih funkcija sustavu je omogućeno da primljene ulazne parametre obradi na odgovarajući način, a zatim na osnovu rezultata obrade ulaznih parametara upravlja aktuatorima. Rad ovako dovršenog modela sustava je demonstriran pomoću simulacije izrađene u CANoe alatu gdje je moguće ručno podešavati ulazne parametre i promatrati izlazne, putem interaktivnog grafičkog sučelja u stvarnom vremenu. Ispravnost rada sustava, odnosno implementiranih funkcija, provjerena je uz pomoć *unit* testova. Rezultati tih testova pokazali su da sustav radi ispravno u svim testnim uvjetima.

LITERATURA

[1] P. M. Dubal,A. N. Shaikh, „Adaptive Head-Light System For Vehicle“, *International Journal of Engineering Research in Computer Science and Engineering,* svez. 5, br. 2, pp. 44-47, 2018.

[2] S. Shreyas, R. Kirthanaa, A. Padmavathy, S. Arun Prasad, G. Devaradjane, „Adaptive Headlight System for Accident Prevention“, *International Conference on Recent Trends in Information Technology,* Chennai, 2014.

[3] S. Pengfei, Z. Yang, W. Xianglong, L. Yufan, „Design and Implementation of the Adaptive Control System for Automotive Headlights Based on CAN/LIN Network“, *Third International Conference on Instrumentation, Measurement, Computer, Communication and Control,* Hefei, 2013.

[4] C. M. Kormanyos, „HID System with Adaptive Vertical Aim Control“, *International Congress and Exposition,* Detroit, 1998.

[5] W. Hendrischk, M. Grimm, F.-J. Kalze, „Adaptive Headlamps“, *ATZ Worldwide,* svez. 104, br. 11, pp. 2-3, 2002.

[6] V. D. Ichake, P. Chipalkatti,G. Kadam, „Design and Implementation of Adaptive Headlight Control System Using CAN-LIN Protocols“, Pune, 2018.

[7] U. Honekamp, M. Wernicke, „Development of Distributed Automotive Software: The DaVinci Methodology“, *Design Methods and Applications for Distributed Embedded Systems*, pp 93-102, 2014.

[8] Gesellschaft für Informatik, ,“www.gi.de“, [Mrežno]. Dostupno na: https://gi.de/fileadmin/GI/Hauptseite/Service/Lexikon/AUTOSAR_Figure2.jpg. [Pristup ostvaren: 03 09 2020].

[9] A. H. ,http://techiscafe.blogspot.com, „Automotive Hub“, [Mrežno]. Dostupno na: http://techiscafe.blogspot.com/p/what-is-autosar-autosar-open-system.html, [Pristup ostvaren: 03 09 2020].

[10] J. Fu, W. Hao, F. B. Bastani, I.-L. Yen, „Model-Driven Development: Where Does the Code Come From?“, *5th IEEE International Conference on Semantic Computing,* Palo Alto, 2011.

[11] M. Rizwan, S. A. Hayat, „The artifacts of component-based development“, 2012. [Mrežno]. Dostupno na:

https://www.researchgate.net/publication/220488245_The_artifacts_of_component-based_development, [Pristup ostvaren: 03 09 2020].

[12] A. Tierno, M. M. Santos, B. Arruda, J. da Rosa, „Open issues for the automotive software testing“, *12th IEEE International Conference on Industry Applications (INDUSCON)*, Curitiba, 2016.

[13] Vector Informatik GmbH, „Testing ECUs and Networks with CANoe“, [Mrežno]. Dostupno na: https://www.vector.com/int/en/products/products-a-z/software/canoe/. [Pristup ostvaren: 03 09 2020].

[14] AUTOSAR GbR, „AUTOSAR_SoftwareComponentTemplate“, 2006. [Mrežno]. Dostupno na: https://www.autosar.org/fileadmin/user_upload/standards/classic/2-0/AUTOSAR_SoftwareComponentTemplate.pdf., [Pristup ostvaren: 03 09 2020].

[15] AUTOSAR, „AUTOSAR_TPS_SoftwareComponentTemplate“, 2015. [Mrežno]. Dostupno na: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-2/AUTOSAR_TPS_SoftwareComponentTemplate.pdf, [Pristup ostvaren: 03 09 2020].

[16] AUTOSAR, „AUTOSAR_EXP_CDDDesignAndIntegrationGuideline“, 2017. [Mrežno]. Dostupno na: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_CDDDesignAndIntegrationGuideline.pdf , [Pristup ostvaren: 03 09 2020].

[17] AUTOSAR, „Specification of DIO Driver“, 2017. [Mrežno]. Dostupno na: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_DIODriver.pdf, [Pristup ostvaren: 03 09 2020].

SAŽETAK

Suvremeni standardi u autoindustriji postavljaju sve više zahtjeve za sigurnost sudionika u prometu. Jedan od važnijih segmenata sigurnosti je zasigurno vidljivost ceste i okolnih objekata. Slabi uvjeti vidljivosti ugrožavaju sigurnost svih sudionika u prometu, a proizvođači taj problem rješavaju implementacijom sve naprednijih sustava svjetala u vozila. Glavna odlika takvih naprednih sustava je njihova interakcija s okolinom vozila i dinamička prilagodba uvjetima. U okviru ovog rada prikazan je proces razvoja prilagodljivog sustava za upravljanje svjetlima vozila kroz nekoliko ključnih faza. Obuhvaćene su faze modeliranja SWC-a, konfiguracija komunikacije, implementacija programskog koda, te simulacija i testiranje gotovog sustava. Sustav je izrađen po AUTOSAR modelu koristeći odgovarajuće metode i alate. Odabrane funkcionalnosti prikazanog rješenja daju pregled nekih od najčešće korištenih rješenja u današnjim sustavima za upravljanje prilagodljivim svjetlima.

Ključne riječi: AFS, AUTOSAR, ECU, SWC, upravljanje svjetlima vozila

ABSTRACT

Modeling and implementation of an adaptive vehicle light management system

Modern standards in the automotive industry place increasingly demanding terms on the safety of road users. One of the most important segments of road safety is surely the visibility of the road and its surroundings. Poor visibility conditions endanger the safety of all traffic participants and the manufacturers are solving this problem by implementing increasingly advanced lightning systems in vehicles. The main feature of such advanced systems is their interaction with the surrounding of the vehicle and its dynamic adaptation to the conditions. This paper presents the process of developing an adaptive light control system of a vehicle via several key phases. Included are the phases of SWC modelling, communication configuration, program code implementation, as well as simulation and testing of the final product. The system is created according to the AUTOSAR standard using appropriate methods and tools. Selected functionalities of the displayed solution provide an overview of some of the most commonly used solutions in today's adaptive light management systems.

Keywords: AFS, AUTOSAR, ECU, SWC, automotive lighting management

ŽIVOTOPIS

Ante Radoš je rođen 21. ožujka 1994. godine u Osijeku. Pohađao je Osnovnu školu Vladimira Nazora u Đakovu, nakon čega upisuje Gimnaziju Antuna Gustava Matoša u Đakovu, prirodoslovno matematički smjer. 2012. godine upisuje se na Građevinski fakultet u Osijeku, s kojeg se ispisuje 2014. godine. 2015. godine upisuje preddiplomski studij Elektrotehnike na Elektrotehničkom fakultetu u Osijeku, današnjem FERIT-u. 2018. sudjeluje na STEM Games-ima u Poreču kao predstavnik FERIT-a u kategoriji Engineering. Preddiplomski sveučilišni studij završava 2018. godine i stječe zvanje sveučilišni prvostupnik (baccalaureus) inženjer elektrotehnike i informacijske tehnologije, a iste godine upisuje i Diplomski sveučilišni studij smjera Mrežne tehnologije na istom fakultetu.

Potpis: _____