UDC: 004.41:338.47 Info M str. 18-26

JEZIK SPECIFIČAN ZA DOMEN MODELOVANJA MCAL SLOJA U AUTOSAR SIL SIMULACIJI
A DOMAIN-SPECIFIC LANGUAGE FOR MCAL LAYER MOCKUP IN AUTOSAR SIL SIMULATION

Nikola Stojkov, Igor Dejanovic
Fakultet Tehnickih Nauka, Univerzitet u Novom Sadu
Faculty of Technical Science, University of Novi Sad

REZIME: AUTOSAR standard obezbeđuje specifikaciju za razvoj softverskih arhitektura u automobilskoj industriji. Jedan od ključnih aspekata AUTOSAR standarda je nezavisnost aplikativnog sloja od mikrokontrolera. U ovom radu, istražujemo načine za testiranje aplikativnog softvera sa zamenskim modulima mikrokontrolerskog sloja nezavisno od hardvera, pokazaćemo kako se jezik specifični za domen (JSD, eng. Domain-Specific Language - DSL) može koristiti za generisanje ovakvog koda za AUTOSAR mikrokontrolerski sloj za simulaciju softvera u petlji (eng. Software in the loop - SIL). Pokazaćemo primer JSD-a koji omogućava korisniku da definiše parametre za memorijski drajver FLS (eng. **Flas**h memory driver), generiše kod za inicijalizaciju drajvera, brisanje, programiranje i čitanje podataka. Biće pokazano kako se ovaj kod može koristiti za povezivanje FLS modula sa FEE (eng. Flash EEPROM Emulation) modulom u skladu sa AUTOSAR standardom. Takođe, raspravljamo o prednostima i ograničenjima korišćenja softverskog modela FLS modula u svrhu testiranja i simulacije, bez oslanjanja na hardver. Pokazujemo kako se softverski model FLS modula može koristiti za pomoć pri testiranju i simulaciji gornjeg softverskog sloja, koji nije zavisan od hardvera.
KLJUČNE REČI: AUTOSAR, JSD (Jezik specifičan za Domen), mock, simulacija, testiranje
ABSTRACT: The AUTOSAR standard provides a set of specifications for developing software architectures for automotive electronics systems. One key aspect of AUTOSAR standard is that application layer is not dependent on the microcontroller. In this paper, we explore the possibilities of testing application part of the AUTOSAR software with mockup modules for microcontroller layer and demonstrate how a Domain-Specific Language (DSL) can be used to generate hardware independent code for the AUTOSAR microcontroller layer for software in the loop (SIL) simulation. We provide an example DSL that allows the user to define the parameters for the FLS (Flash Memory Driver) module, generate code for driver initialization, erasing, programming data, and reading data from the module. We show how this code can be used to connect the FLS module to the FEE (Flash EEPROM Emulation) module, enabling operations on the FLS memory in accordance with the AUTOSAR standard. We also discuss the benefits and limitations of using a software mockup of the FLS module for testing and simulation purposes, without relying on the hardware. We will show how the software mockup of the FLS module can be used to help the testing and simulation of the upper software layer, which is not hardware dependent.
KEY WORDS: AUTOSAR, DSL (Domain Specific Languages), mock, simulation, testing

1. UVOD

U automobilskoj industriji uobičajeno je da se softver piše pre nego što hardver postane dostupan, jer se razvoj softvera i hardvera radi u paraleli. Ovakav način razvoja softvera može dovesti do problema pri integraciji softvera sa hardverom. Takođe se otežava razvoj i stabilnost bezbednosnih mehanizama u softveru (ako je hardver namenjen za rad u okruženju gde je bezbednost bitna) jer se problemi obično otkrivaju tek kasnije u procesu razvoja ili tek kada hardver bude dostupan. Rešavanje ovih problema u kasnijim fazama razvoja može biti izuzetno skupo i vremenski zahtevno. Korišćenje SIL (softver u petlji) metodologije omogućava proveru softvera u simuliranom okruženju pre nego što se softver stavi u produkciju [1, 2].

Ovakav razvoj softvera omogućava softverskim dizajnerima da testiraju softver u situacijama koje bi inače bile teško ponoviti u stvarnom svetu. Na primer, dizajneri softvera za automobilsku industriju mogu simulirati različite uslove upravljanja vozilom, uključujući i bezbednosno kočenje, ili testiranje softvera energetske elektronike gde naponi prelaze uveliko preko bezbednosnih vrednosti za čoveka.

Iako korišćenje SIL metodologije ima brojne prednosti, postoje i izazovi u njenoj primeni. Jedan od glavnih problema je vremenski i finansijski trošak koji je povezan sa razvojem simuliranog hardvera i softvera. Ipak, ovi troškovi su često opravdani u odnosu na troškove popravke nakon što se softver stavi u produkciju. U ovom radu, pokazaćemo načine kako se može ubrzati razvoj okruženja za simuliranje softvera u petlji, korišćenjem JSD metodologije i time smanjiti trošak ukupnog razvoja softvera.

2. AUTOSAR ARHITEKTURA

Arhitektura AUTOSAR (**AUT**omotive **O**pen **S**ystem **AR**chitecture) je standardizovani pristup razvoju softvera u automobilskoj industriji [3]. Korišćenje standardizovanih interfejsa i komponenti u skladu sa AUTOSAR arhitekturom smanjuje se rizik od grešaka u dizajnu sistema i olakšava uočavanje i rešavanje problema u toku razvoja.

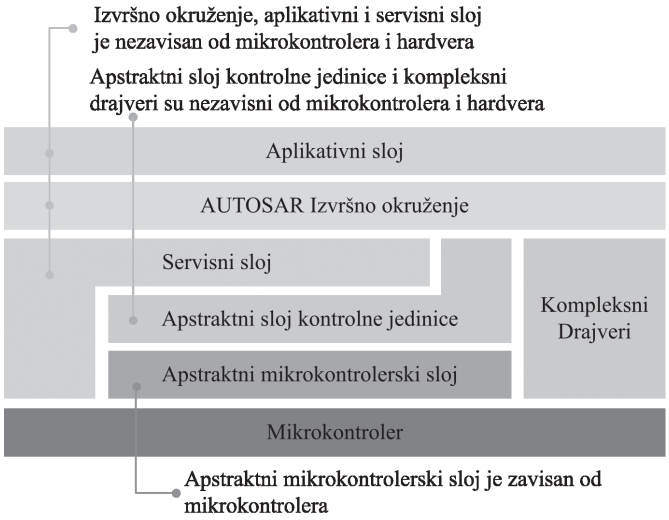
Arhitektura je podeljena na tri sloja: aplikativni, servisni i mikrokontrolerski sloj (*Ilustracija 1)* [4].

Aplikativni sloj sadrži softver za konkretne funkcije koje se izvršavaju na automobilu, kao što su upravljanje motorom, klima uređajem, osvetljenjem, navigacijom, i sl.

Servisni sloj pruža infrastrukturu za komunikaciju i razmenu podataka između aplikacija i mikrokontrolera, kao i podršku za dijagnostiku i održavanje.

Mikrokontrolerski sloj sadrži softver specifičan za mikrokontroler na samom hardveru, i obavlja niske nivoe funkcija, kao što su upravljanje memorijom, rad sa prekidima, itd. Implementacija softvera na sloju mikrokontrolera zavisi od konkretnog mikrokontrolera koji se koristi. Ipak, aplikativni i servisni sloj su dizajnirani da budu nezavisni od konkretnog hardvera koji se koristi, što omogućava da se isti softver može koristiti na različitim mikrokontrolerima.

Testiranje softvera koji zavisi od hardvera, tj. sloja mikrokontrolera (eng. Microcontroller Abstraction Layer - MCAL) može biti izazovno ako hardver nije dostupan. U takvim situacijama, može se koristiti simulator hardvera ili virtualni uređaj koji emulira hardver i omogućava testiranje softvera bez fizičkog hardvera.



Ilustracija 1: AUTOSAR Arhitektura [4]

3. SOFTVER U PETLJI

Softver u petlji (SIL) je tehnika koja se koristi za testiranje softvera u simuliranoj okolini umesto na fizičkom hardveru. SIL se obično koristi tokom razvoja softvera kako bi se proverilo da li softver ispunjava funkcionalne zahteve i specifikacije pre nego što se implementira na fizičkom hardveru.

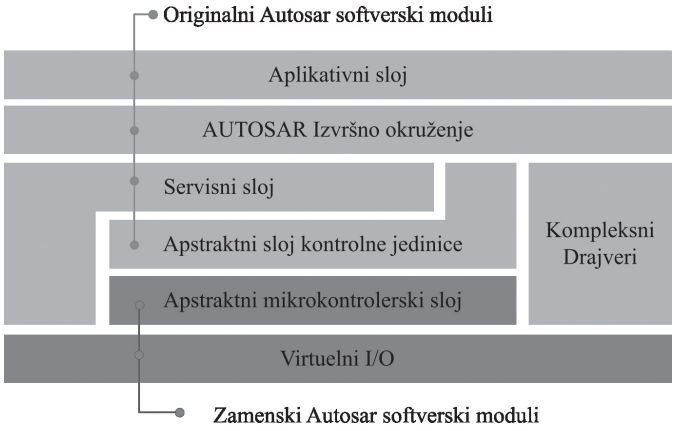
U kontekstu AUTOSAR-a, SIL se koristi za testiranje softvera u ranoj fazi razvoja, kada fizički hardver još nije dostupan, skup je za upotrebu ili pogrešno upravljanje hardverom može dovesti do velike štete. Ovakvo testiranje omogućava dizajnerima softvera testiranje u simuliranom okruženju i identifikaciju potencijalnih problema ili grešaka pre integracije sa fizičkim hardverom.

Jedan od problema koji se može pojaviti pri korišćenju SIL tehnike za testiranje softvera u AUTOSAR-u je zavisnost MCAL sloja i ponekad zavisnost operativnog sistema (OS) od fizičkog hardvera. To znači da je simulacija OS-a i MCAL sloja u SIL okruženju teža i da može biti ograničena. Na primer, neke funkcionalnosti koje su dostupne na fizičkom hardveru mogu biti nedostupne u simuliranom okruženju, što može uticati na tačnost testova i pouzdanost softvera.

Kako bi se rešio ovaj problem, u SIL okruženju se obično koriste simulirane verzije OS-a i MCAL sloja. Ove simulirane verzije omogućavaju dizajnerima softvera da testiraju softver u simuliranom okruženju i identifikuju potencijalne probleme ili greške pre implementacije na fizičkom hardveru.

Jedan od načina za simuliranje MCAL-a je korišćenje simulatora hardvera koji emulira hardver mikrokontrolera. Ovakvi simulatori omogućavaju izvršavanje softvera na simuliranom hardveru i pružaju virtuelni interfejs koji se može koristiti u softveru koji zavisi od MCAL-a [5].

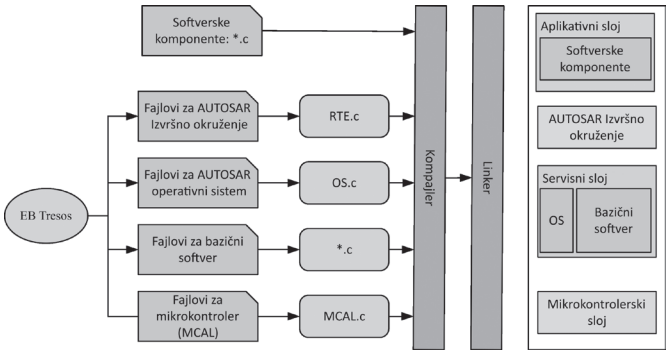
Drugi način za simuliranje MCAL-a je korišćenje zamenskih funkcija koje implementiraju interfejs MCAL-a i koje se koriste umesto stvarnog MCAL-a u softverskom okruženju što je i tema ovog rada. Ove zamenske funkcije (*Ilustracija 2)* pružaju isti interfejs kao i MCAL, ali ne zavise od fizičkog hardvera i mogu biti implementirane kao obične funkcije u softveru. Ove zamenske funkcije se mogu koristiti u SIL okruženju za testiranje softvera koji zavisi od MCAL-a. Zamenske funkcije se mogu konfigurisati tako da simuliraju različita scenarija i uslove rada hardvera, što omogućava testiranje softvera u različitim uslovima rada.



Ilustracija 2: Originalni i zamenski moduli

4. PRIMENA JSD-A ZA "APSTRAKTNI MIKROKONTROLERSKI SLOJ"

Za razvoj softvera u AUTOSAR-u se često koriste alati, među kojim je EB Tresos kompanije Electrobit [6]. EB Tresos je alat za razvoj softvera koji pruža različite funkcionalnosti za razvoj softvera, uključujući i generisanje koda, integraciju softverskih modula kao i validaciju softvera. Proces razvoja softvera u EB Tresos-u obično uključuje nekoliko koraka. Prvi korak je definisanje arhitekture sistema. Ovo uključuje izbor komponenti koje će biti korišćene, definisanje njihovih interfejsa i konfigurisanje sistema (*.xdm, *.arxml fajlovi) kako bi se zadovoljile specifične potrebe projekta. Nakon definisanja arhitekture, softverski dizajneri implementiraju softverske komponente koje su definisane u skladu sa AUTOSAR standardom.

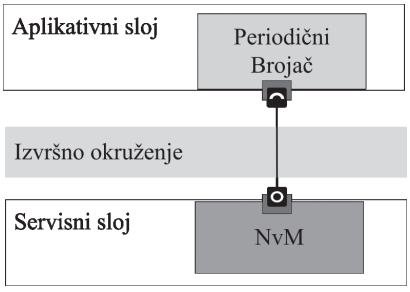


Ilustracija 3: Razvoj softvera u AUTOSAR-u

Datoteke kao što su XDMap (XML Data Mapping) fajlovi (*.xdm) se koriste za konfigurisanje modula u AUTOSAR arhitekturi. Ovi fajlovi definišu parametre i konfiguracije modula, što omogućava Tresos-u da generiše kod za module u skladu sa specifikacijama koje su definisane u fajlovima. Konkretno, *.xdm fajlovi se koriste za konfigurisanje i generisanje koda za AUTOSAR jezgro (izvršno okruženje, servisni sloj i apstraktni sloj kontrolne jedinice - *Ilustracija 1*), što su osnovni moduli koji pružaju zajedničke funkcije koje koriste ostali moduli u AUTOSAR arhitekturi.

Fajlovi kao što su ARXML (AUTOSAR XML) (*.arxml) se koriste za integraciju softverskih modula i komponenti u AUTOSAR arhitekturi. Ovi fajlovi definišu interfejse između softverskih modula i komponenti, i omogućavaju EB Tresos-u da generiše kod koji implementira te interfejse i omogućava komunikaciju između softverskih modula i komponenti. ARXML fajlovi se koriste za konfigurisanje i generisanje koda za aplikativne softverske module i komponente, kako bi se mogli uvezivati u donje slojeve kao što su na primer servisni sloj bazičnih AUTOSAR modula [7].

U ovom radu je posmatran primer kompozicije (*Ilustracija 4*) softverskih komponenti za pisanje i čitanje podatka iz memorije od aplikativnog softverskog modula do NVM (eng. Non Volatile Memory) modula u servisnom sloju, putem klijent-server principa.



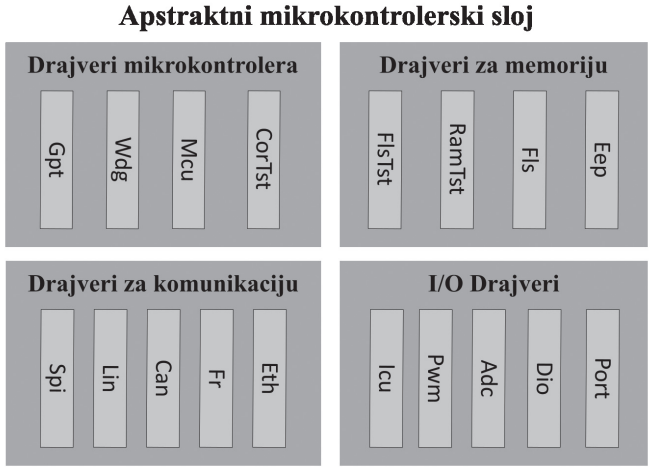
Ilustracija 4: Softverska kompozicija komponenti

Kada se radi sa SIL testovima, često se postavlja pitanje da li se može koristiti MCAL sloj generisan od strane EB Tresos-a. MCAL sloj predstavlja sloj apstrakcije hardvera koji omogućava softverskim komponentama da rade sa hardverskim modulima koji su specifični za odabrani mikrokontroler. U SIL testovima se ne koristi stvarni hardver, već se simulira u softveru. Zbog toga se MCAL sloj koji je generisan u EB Tresos-u ne može direktno koristiti u SIL testovima.

Postoje komercijalna rešenja simulacije MCAL sloja, kao što su vVirtualTarget [8], Synopsis Silver [5] i drugi alati koji pružaju određene generičke komponente MCAL sloja, dok komponente specifične za hardverski sistem korisnik mora da integriše sam.

U ovom radu ćemo se bazirati na JSD kao rešenju otvorenog koda, ali isto tako integracija sa komercijalnim rešenjem je moguća.

Zamena MCAL sloja u okviru SIL okruženja može biti izuzetno složen proces. Kada se MCAL sloj ručno piše, postoji rizik od grešaka i nepotpunog pokrivanja svih funkcionalnosti. Ovo može dovesti do toga da softverski testovi ne budu dovoljno precizni i pouzdani. Ukoliko bi koristili jezik specifičan za domen opisa i generisanja MCAL sloja, mogli bismo značajno pomoći u rešavanju ovog problema. Jezik specifičan za domen je programski jezik koji je prilagođen za rešavanje specifičnih problema iz domena aplikacije. U slučaju AUTOSAR arhitekture, JSD bi mogao biti korišćen za generisanje MCAL sloja. Na primer, softverski dizajner bi mogao da definiše konfiguraciju i specifikacije hardverskih funkcionalnosti kao i komunikacione protokole u JSD-u, nakon čega bi se MCAL sloj automatski generisao na osnovu tih specifikacija i tako napravio MCAL sa AUTOSAR standardizovanim interfejsima. Ovo bi omogućilo da se fokus usmeri na sam softver i testiranje, umesto da se troši vreme na ručno podešavanje MCAL sloja za SIL okruženje. Takođe bi se smanjio rizik od grešaka i olakšalo održavanje softvera.

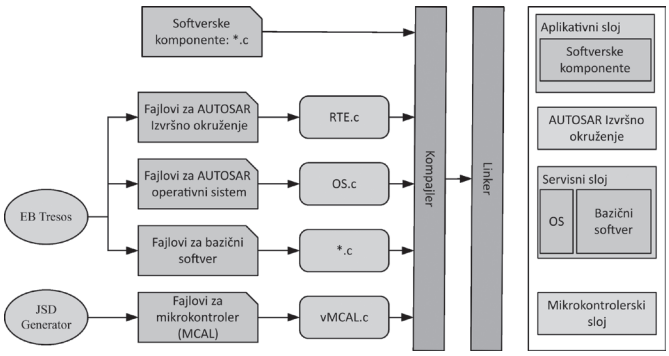


Ilustracija 5: Mikrokontrolerski sloj

U okviru SIL okruženja, virtuelni I/O (Input/Output) se koristi za simuliranje hardverskih ulaza i izlaza, kao i za prikupljanje podataka iz softverske simulacije.

U slučaju automobilske kontrolne jedinice (ECU), virtuelni I/O (*Ilustracija 2*) se može koristiti za simuliranje ulaza senzora i izlaza aktuatora. Ovo omogućava testiranje i validaciju softvera bez potrebe za fizičkom implementacijom senzora i aktuatora u stvarnom okruženju. Korišćenje virtuelnog I/O-a

omogućava softverskim dizajnerima da testiraju svoj softver u kontrolisanom okruženju, bez potrebe za stvarnim hardverom. Takođe se omogućava brže i efikasnije testiranje za različita scenarija bez potrebe za ručnim podešavanjem hardvera. SIL okruženje sa virtuelnim I/O-om može se koristiti zajedno sa JSD-om za generisanje MCAL sloja, kako bi se automatizovalo generisanje koda za simulaciju hardverskih ulaza i izlaza.



Ilustracija 6: JSD generator za MCAL sloj

5. REALIZACIJA JSD-A ZA FLS MODUL

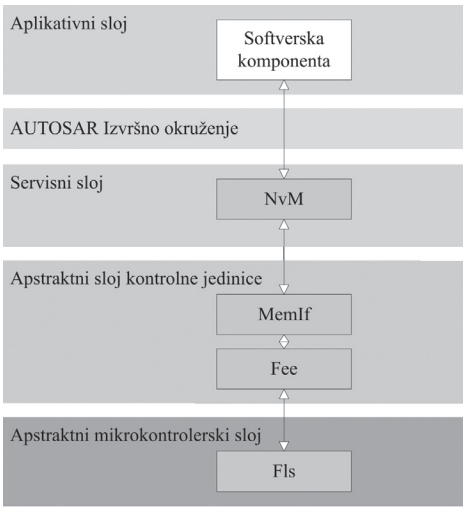
Tema ovog rada je bazirana na primeru jednog od softverskih modula unutar MCAL sloja. Izabrani modul koji će biti pokazan u domenu JSD-a je FLS softverski modul.

Realizacija JSD-a za FLS modul podrazumeva implementaciju specifičnih jezičkih konstrukcija i pravila za generisanje koda koji je optimizovan za FLS modul. JSD bi trebalo da pruži jezičke konstrukcije za upravljanje memorijom i operacijama sa memorijom, poput opisa same konstrukcije memorije kao i čitanje i pisanje podataka u memoriju. Ovo bi omogućilo softverskim dizajnerima da jednostavno definišu i koriste ove operacije u kodu, bez potrebe za detaljnim poznavanjem karakteristika FLS modula. Pored toga, JSD bi trebalo da pruži i specifična pravila za generisanje koda koji je optimizovan za FLS modul.

Kada se radi o realizaciji JSD-a za FLS modul, jedan od pristupa bi mogao biti korišćenje postojećih programskih jezika poput C ili C++, sa dodatkom domenskih jezičkih konstrukcija i pravila. Alternativno, mogu se koristiti i specijalizovani jezici poput MATLAB-a.

Uzmimo za primer memorijski servis (NVM) koji se nalazi u servisnom sloju (što je i tema ovog rada) i omogućava rad sa ne-volatilnom memorijom (NVM) u automobilskim aplikacijama. FLS (Flash) drajver predstavlja jedan od modula koji se koriste za realizaciju memorijskog servisa.

Gornji slojevi NVM servisa rade sa FLS drajverom kroz MCAL sloj. Time, gornji slojevi NVM servisa ne moraju direktno da komuniciraju sa FLS drajverom, već koriste standardne interfejse koje pruža MCAL sloj kao što su čitanje i pisanje podataka u memoriju. MCAL sloj poziva odgovarajuće interfejse u FLS drajveru da bi obavio konkretne operacije na memoriji.

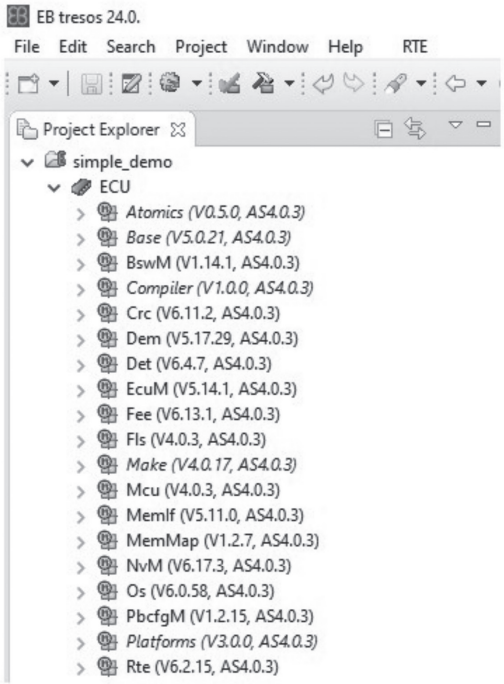


Ilustracija 7: Memorijski servis

5.1. *EB TRESOS KONFIGURACIJA AUTOSAR SOFTVERA*

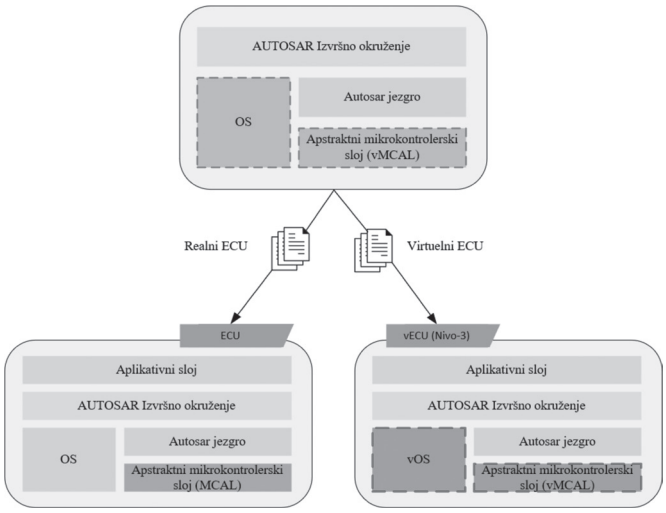
EB Tresos [6] omogućava integraciju AUTOSAR jezgra (izvršno okruženje, servisni sloj i apstraktni sloj kontrolne jedinice), a podržava različite operativne sisteme za različite platforme, uključujući AUTOSAR OS, OSEK/VDX, Linux i Windows.

Ilustracija 8 je primer jednog integrisanog AUTOSAR jezgra u projektu nazvanog „simple_demo“. Treba naglasiti da ovakva konfiguracija (*Ilustracija 9*) AUTOSAR jezgra ne zavisi od toga da li se generisani softver koristi za SIL ili za pravi hardver. EB Tresos konfiguracija se moze lako sačuvati kako bi se mogla kasnije koristiti za druge projekte ili hardverske jedinice.



Ilustracija 8: Primer Electrobit Tresos projekta

Ilustracija 9 je primer jednog virtuelnog sistema. Operativni sistem i MCAL sloj je zamenjen sa modulima za simulaciju. Operativni sistem vOS je zamenski modul dobijen od strane Electrobit-ovog AUTOSAR jezgra za Windows operativni sistem, jer u konkretnom slučaju OS modul za realan sistem zavisi od tajmerskih jedinica koje se nalaze na realnom mikrokontroleru. Da bi se ovo prevazišlo OS je prepravljen da tajmeri dolaze od Windows ili Linux operativnog sistem (vOS).



Ilustracija 9: Generisanje programskih fajlova

5.2. JSD SPECIFIKACIJA ZA FSL MODUL

U ovom radu predstavljen je JSD koji uključuje osnovne parametre potrebne za FLS modul ali i dodatne parametre radi simulacije softvera u petlji.

```
MODULE vFLS {
    size = 1048576
    start_address = 0
    otp = [1048320, 256]
    EVENTS {
        EVENT Init {
            CYCLE 0 ms {
                DELAY 1000 ms
            }
        }
        EVENT Result {
            CYCLE 0 ms {
                DELAY 100 ms
            }
            CYCLE 1000 ms {
                ERROR "Get status result error" for 5000 ms
            }
        }
        EVENT Write {
            CYCLE 0 ms {
                DELAY 100 ms
            }
            CYCLE 1000 ms {
                ERROR "Failed access" for 5000 ms
            }
        }
    }
}
```

Ilustracija 10: JSD za vMCAL

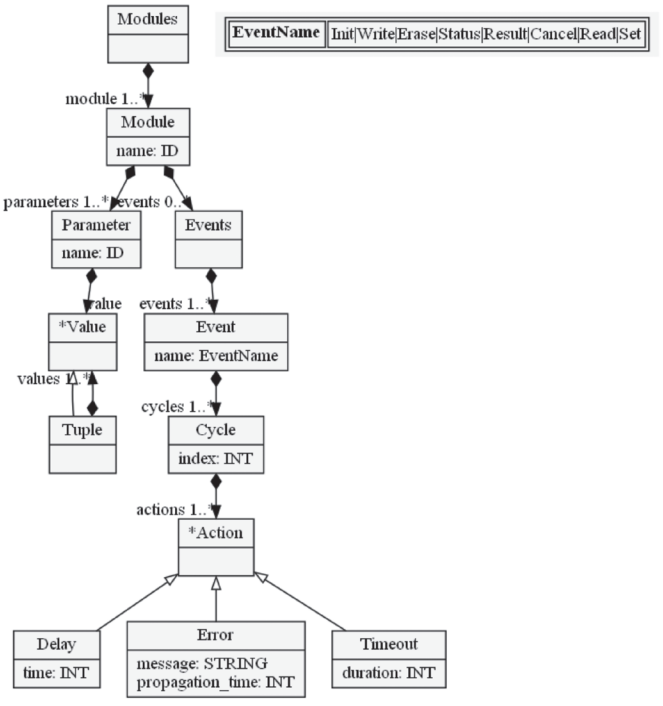
Specifikacija je data u tekstualnom formatu. Treba napomenuti da je JSD konfiguracija (*Ilustracija 10)* generalizovana za dodavanje novih parametara kao i opisa drugih modula unutar MCAL sloja. Videti GitHub projekat [9].

Opis modula počinje sa ključnom reči MODULE i zatim se piše ime modula, u ovom slučaju vFLS. Za potrebe FLS modula data je definicija parametara za veličinu i početnu adresu memorijskog prostora, kao i gde se nalazi deo memorije koji je jednom moguće upisati (eng. One Time Programmable - OTP).

Nakon definisanja FLS parametara, zadaju se parametri simulacije koji će uticati na izvršavanje softvera u petlji. To se postiže korišćenjem ključne reči EVENTS, nako koje korisnik zadaje niz događaja putem ključne reči EVENT (*Ilustracija 10*). Ovi EVENT blokovi definišu trenutak u izvršavanju simulacije gde će se aplicirati zadati EVENT. Ključna reč CYCLE govori u kom vremenskom periodu simulacije treba da se izvrši zadati EVENT blok.

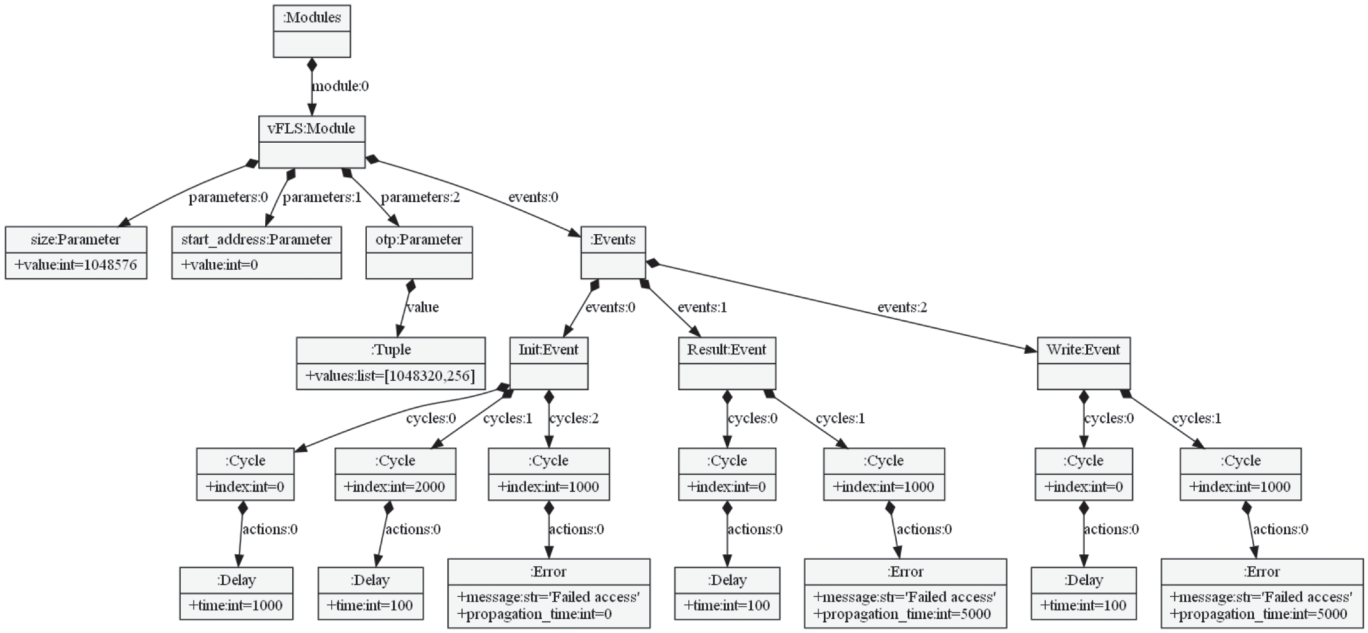
U primeru (*Ilustracija 10*) definisana su tri EVENT bloka. U prvom EVENT bloku je definisano da će u prvom ciklusu operacija sa FLS modulom će trajati jednu sekundu, što omogućava da se proveri reakcija softvera na veliko kašnjenje u, na primer, inicijalizaciji FLS modula. Drugi i treći EVENT blok opisuju da u prvom ciklusu operacija sa FLS modulom traje 100 milisekundi, a za 1 sekundu kasnije, operacija sa FLS modulom rezultuje greškom i ostaje aktivna 5 sekundi, što omogućava da se proveri reakcija softvera na grešku.

Ilustracija 11 predstavlja meta-model JSD-a.



Ilustracija 11:JSD Meta - model

Ilustracija 12 predstavlja parsirani JSD sa primera koristeći meta-model (*Ilustracija 11)*.



Ilustracija 12: Model vMCAL specifikacije

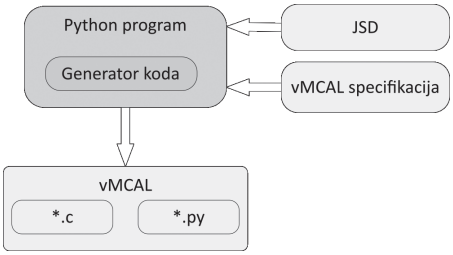
5.3. PARSIRANJE JSD SPECIFIKACIJE

Za definisanje JSD-a korišćen je textX meta-jezik [10] koji omogućava softverskim dizajnerima da definišu jezik u jednostavnom tekstualnom formatu i uz pomoć Python jezika implementiraju parsiranje tog jezika. Takođe omogućava proveru i obradu semantičkih pravila za parsirane strukture.

Posle parsiranja JSD-a uz pomoć textX modula, naredni korak je generisanje GPL (eng. General purpose language) koda, tj koda za jezike široke upotrebe (C, Python itd.).

U ovom radu za generisanje C i Python koda, korišćena je Jinja biblioteka [11].

Generisani C kod će se koristiti kao zamenski kod za FLS modul, dok će se Python kod koristiti za simulaciju softvera u petlji. Videti GitHub projekat za detalje [9].



Ilustracija 13: Generisanje koda za GPL

```
import jinja2
from textx.metamodel import metamodel_from_file

mcal_metamodel = metamodel_from_file('mcal.tx', debug=False)

# Create model for vFLS
mcal_model = mcal_metamodel.model_from_file('mcal.config')
# Generate code
for module in mcal_model.module:
# generate C code first
    # Load C template
    template = jinja_env.get_template('%s.c_template'% module.name)
     # For each entity generate C file
    with open(os.path.join('..', 'src_gen',
                  "%s.c" % module.name), 'w') as f:
        f.write(template.render(module=module))
# generate python code
    # Load the Py template
    template = jinja_env.get_template('%s.p_template'% module.name)
     # For each entity generate python file
    with open(os.path.join('..', 'src_gen',
                  "%s.py" % module.name), 'w') as f:
        f.write(template.render(module=module))
```

Ilustracija 14: Python generator za GPL kod

5.4. KOMPAJLIRANJE SOFTVERA SA ZAMENSKIM FLS MODULOM

Kada je C i Python kod (vFLS) generisan za FLS modul, može se dalje kompajlirati softver za simulaciju. Zamenom Fls.c fajla, koji dolazi sa standardnim softverom, sa generisanim vFLS.c fajlom omogućujemo kompajliranje na Windows ili drugim operativnim sistemom (*Ilustracija 9*).

U generisanom kodu, unutar vFLC.c fajla, definisani su potrebni atributi kompajlera kako bi se funkcije FLS modula videle za spoljne pozive kroz DLL fajl, kao i mapiranje na Python vFLS modul za simulaciju.

Mapiranje ovih funkcija je odrađeno automatski na nivou generisanja koda i korisnik ne mora da dodatno podešava FLS modul.

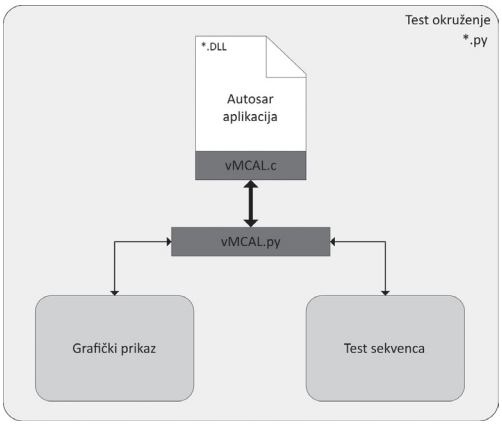
U ovom radu korišćen je alat Make [12] i MinGW kompajler [13] kako bi se dobio kod za izvršavanje na Windows operativnom sistemu.

Pošto će se kompajlirana AUTOSAR aplikacija koristiti u simulaciji softvera u petlji, krajnji proizvod će biti dinamička biblioteka, u Windows okruženju DLL (eng. Dynamic Linking Library) fajl.

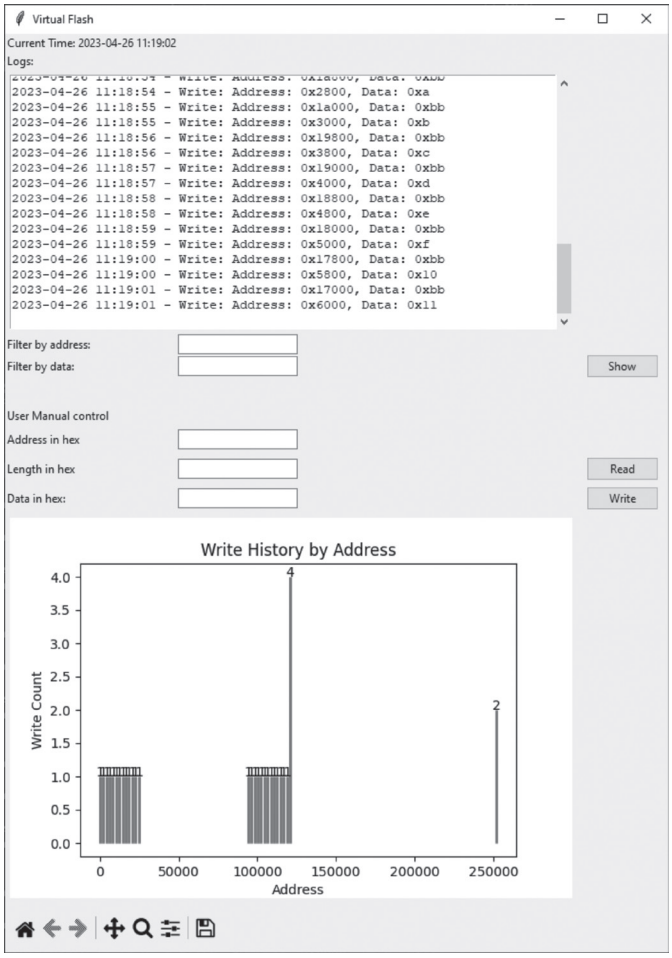
5.5. IZVRŠAVANJE SOFTVERA NA WINDOWS OPERATIVNOM SISTEMU

Softver u petlji se sastoji od Python okruženja i AUTOSAR aplikacije (DLL fajla). Ilustracija 15 prikazuje okruženje za simulaciju softvera u petlji. Komponenta vMCAL predstavlja zamenski MCAL sloj koji se sastoji od simuliranih MCAL komponenti, u našem primeru od vFLS modula.

Pored simuliranog FLS modula, napravljen je grafički prikaz FLS operacija (*Ilustracija 16*) i test scenarija po ciklusu simulacije.



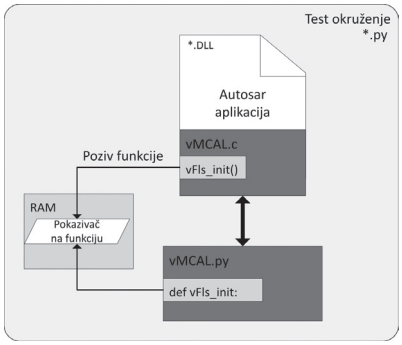
Ilustracija 15: Test okruženje



Ilustracija 16: Grafički prikaz FLS operacija

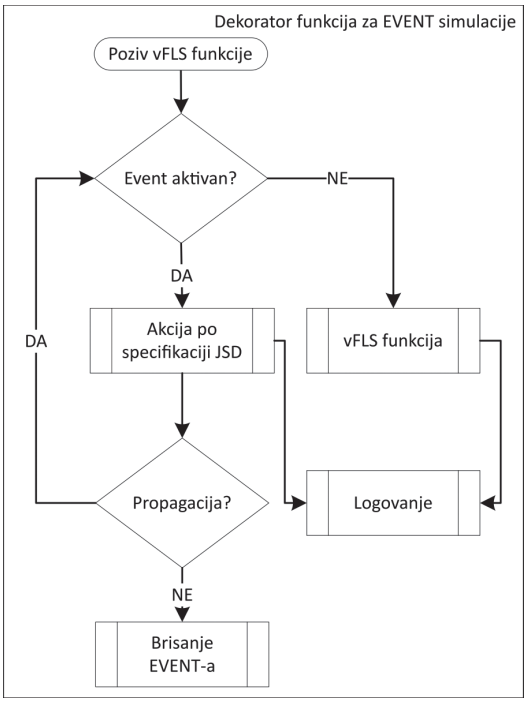
Korisnik može kroz grafički prikaz posmatrati FLS operacije, filtrirati i sačuvati istoriju samih operacija, kao i manuelno upisivati podatke u memoriju. Simulacija softvera u petlji je realizovana na način da je sva apstrakcija kao i simulacija FLS modula smeštena u Python jeziku. Ovo omogućava korisniku veliku fleksibilnost kao i brzu implementaciju ponašanja samog FLS modula. AUTOSAR aplikacija (DLL) poziva Python funkcije (*Ilustracija 17*) i time vMCAL predstavlja sponu između C i Python jezika za FLS funkcionalnost (vMCAL.c i vMCAL.py).

Prilikom pokretanja Python test okruženja, FLS funkcije napisane unutar vMCAL.py modula su prosleđene AUTOSAR aplikaciji (vMCAL.c) kroz pokazivač na funkciju. Kada se simulacija pokrene, AUTOSAR aplikacija poziva vFLS funkcije i time direktno izvršava Python vFLS funkcije.



Ilustracija 17: vFLS sprega između C i Python koda

Kako bi se korisnički definisani EVENT blokovi aplicirali, napisan je dekorator funkcije koji će u zavisnosti od vremenskog perioda simulacije i same akcije opisane u EVENT bloku, izvršiti dodatne funkcije (*Ilustracija 18*).



Ilustracija 18:Dekorator vMCAL funkcija

6. REZULTATI IMPLEMENTACIJE

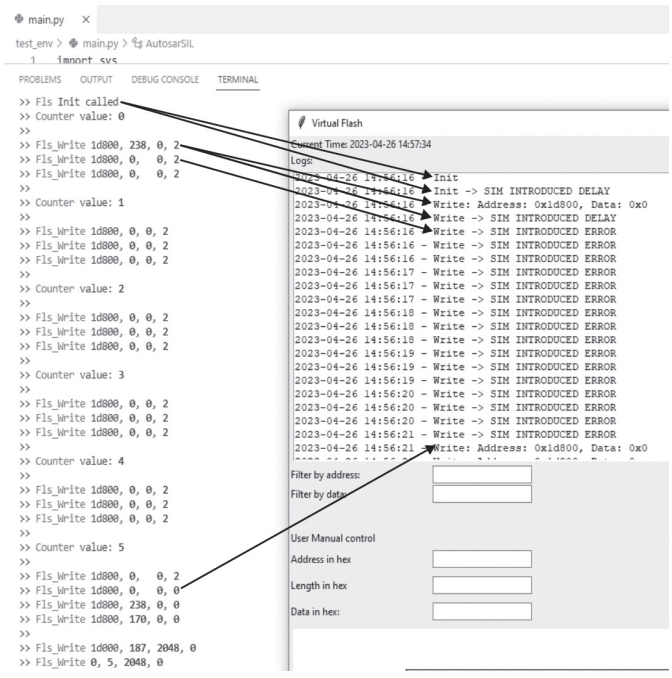
Primer (Ilustracija 4) na kome je softver simuliran u petlji, odnosi se na vrednost brojača koji se upisuje u memoriju i uvećava svake sekunde. Pokretanjem simulacije, vidimo da zahtevi od AUTOSAR aplikacije dolaze ka vMCAL sloju. Prva operacija koja se može videti posmatrajući prozor u terminalu u kom se nalazi ispis poruka od AUTOSAR aplikacije, je Init operacija. Kako je specificirano kroz JSD (*Ilustracija 10*), inicijalizacija vFLS-a treba da traje 1 sekundu što možemo i videti na grafičkom prikazu Virtuelnog Flash prozora (*Ilustracija 19*).

Sledeći zahtev je upis brojača kroz "Write" zahtev. Po JSD specifikaciji, zahtev u prvom pozivu treba da traje 100 milisekundi, što možemo i potvrditi na grafičkom prikazu Virtuelnog Flash prozora (*Ilustracija 19*). Zatim po specifikaciji JSD-a upis u memoriju treba da rezultuje greškom koja traje 5 sekundi. Na prozoru Virtuelnog Flash-a možemo videti kako zahtevi pristižu u vremenskom periodu od 5 sekundi, i svaki zahtev rezultuje greškom.

AUTOSAR aplikacija detektuje grešku i ponovno pokreće upis u memoriju u sledećem ciklusu.

Ono što se može videti jeste da aplikativni sloj AUTOSAR aplikacije, simulirane u petlji, nastavlja da uvećava brojač iako je nastala greška prilikom upisa. Ovo nam govori da se aplikacija mora popraviti da uzima u obzir rezultat NVM operacije ako se svaka vrednost brojača mora upisati.

Vidimo da tek vrednost brojača 5 biva upisana u memoriju (*Ilustracija 19*), pošto se posle 5 sekundi simularina greška uklanja sa "Write" operacije, dok je period upisa brojača u AUTOSAR aplikaciji podešen na jednu sekundu.



Ilustracija 19: Rezultat testiranja

7. STANJE U OBLASTI

Najveći uticaj na ovo istraživanje imali su radovi koji se bave simulacijom i ko-simulacijom AUTOSAR softvera. U radu [14], autori prikazuju upotrebu FMI (eng. Functional Mock-up Interface) za virtuelnu validaciju softvera za ECU. U radu [15] i [16] autori prikazuju upotrebu Synopsys Silver alata za virtualnu integraciju hardverskih i softverskih komponenti u automobilskoj industriji.

Napomenuti radovi koriste poznate standarde ili alate za simulaciju AUTOSAR softvera na nivou ECU-a i ko-simulacije za razmenu podataka između drugih alata, dok se u ovom radu sagledavaju neki od ključnih aspekata vezanih za izradu virtuelnog MCAL sloja u Python jeziku i integraciju njegove funkcionalnosti direktno sa C AUTOSAR aplikacijom.

8. ZAKLJUČAK

U ovom radu je predstavljen je model JSD-a pomoću kojeg se mogu generisati komponente virtuelnog MCAL sloja (vMCAL). Sav kod je slobodno dostupan i nalazi se na GitHub projektu [9]. Primeri su pokazani sa memorijskom komponentom (NVM) i operacijama sa fleš memorijom (FLS) ali je pristup veoma sličan i za druge komponente, tako da je JSD predstavljen veoma prilagodljiv za generisanje i drugih komponenti MCAL sloja. U radu je prikazan primer AUTOSAR aplikacije u DLL formatu gde je izvršavanje SIL okruženja bilo na Windows operativnom sistemu. Sam JSD model je nezavisan od platforme i od GPL jezika, može se lako nadograditi da SIL okruženje podržava izvršavanje i na Linux operativnom sistemu uz korišćenje dinamičke biblioteke za AUTOSAR aplikaciju prilagođena platformi na kojoj se izvršava kod.

Dalji pravci razvoja bi podrazumevali nadogradnju JSD modela za ostale komponente MCAL sloja kao i mogućnost generisanja izveštaja za greške nastale prilikom simulacije da se ukaže na eventualne probleme sa AUTOSAR aplikacijom.

LITERATURA

[1] S. Jeong, Y. Kwak i W. Lee, „Software-in-the-Loop simulation for early-stage testing of AUTOSAR software component.," u *Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, Vienna, Austria, 2016.

[2] M. Tatar i J. Mauss, „Systematic Test and Validation of Complex Embedded Systems," u *Embedded Real Time Software and Systems*, Toulouse, France, 2014.

[3] AUTOSAR, „Standards," [Na mreži]. Available: https://www.autosar.org/standards. [Poslednji pristup 25 4 2023].

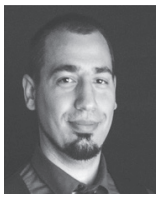
[4] AUTOSAR, „Classic Platform," 2014. [Na mreži]. Available: https://www.autosar.org/standards/classic-platform. [Poslednji pristup 25 04 2023].

[5] Synopsys, „Silver," Synopsys, [Na mreži]. Available: https://www.synopsys.com/verification/virtual-prototyping/virtual-ecu/silver.html. [Poslednji pristup 25 04 2023].

[6] Electrobit, „EB Tresos," Electrobit, [Na mreži]. Available: https://www.elektrobit.com/products/ecu/eb-tresos/. [Poslednji pristup 25 04 2023].

[7] AUTOSAR, *AUTOSAR Basic Software Module, AUTOSAR Specification Release 4.2.1,* Munich, Germany: AUTOSAR, 2014.

[8] Vector, „vVirtualTarget,“ [Na mreži]. Available: https://www.vector.com/at/de/produkte/produkte-a-z/software/vvirtualtarget/. [Poslednji pristup 25 04 2023].
[9] N. Stojkov, „vMCAL,“ 2023. [Na mreži]. Available: https://github.com/nikola-winmaker/vMCAL. [Poslednji pristup 25 04 2023].
[10] I. Dejanović, R. Vaderna, G. Milosavljević i Ž. Vuković, „textX: A Python tool for Domain-Specific Languages,“ *Knowledge-based systems,* br. 115, pp. 1-4, 2017.
[11] Pallets, „Jinja,“ [Na mreži]. Available: https://jinja.palletsprojects.com/en/3.1.x/. [Poslednji pristup 25 04 2023].
[12] GNU, „GNU Make,“ [Na mreži]. Available: https://www.gnu.org/software/make/. [Poslednji pristup 25 04 2023].
[13] „MinGW-w64,“ [Na mreži]. Available: https://www.mingw-w64.org/. [Poslednji pristup 25 04 2023].
[14] L. Mikelsons i R. Samlaus, „Towards Virtual Validation of ECU Software using FMI,“ u *International Modelica Conference*, Prague, Czech Republic, 2017.
[15] A. Junghanns, „Virtual integration of Automotive Hard-and Software with Silver,“ QTronic GmbH, Berlin, 2010.
[16] A. Junghanns, R. Serway, T. Liebezeit i M. Bonin, „Building virtual ECUs quickly and economically,“ *ATZelektronik worldwide,* t. 7, br. 3, pp. 48-51, 2012.



Nikola Stojkov je PhD student na Departmanu za računarstvo i automatiku, Fakulteta tehničkih nauka Univerziteta u Novom Sadu.
Kontakt: nikola.stojkov@outlook.com
Oblast interesovanja: Softverska arhitektura, jezici specifični za domen, Autosar, simulacija softvera, izvršavanje softvera u realnom vremenu.

Igor Dejanović je redovni profesor na Departmanu za računarstvo i automatiku, Fakulteta tehničkih nauka Univerziteta u Novom Sadu.
Kontakt: igord@uns.ac.rs
Oblast interesovanja: Inženjerstvo softverskih jezika, jezici specifični za domen, parsiranje, programski prevodioci, upravljanje konfiguracijom softvera.