UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Vlado Dimitrieski

Vzpostavitev vozlišča CANopen na vgrajenem sistemu

diplomsko delo na univerzitetnem študiju

mentor: izr. prof. dr. Uroš Lotrič

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01669/2010
Datum: 05.04.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **VLADO DIMITRIESKI**

Naslov: **VZPOSTAVITEV VOZLIŠČA CANOPEN NA VGRAJENEM SISTEMU**

IMPLEMENTATION OF A CANOPEN NODE IN AN EMBEDDED SYSTEM

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Danes, ko vgrajene sisteme najdemo že skoraj v vsaki napravi, je njihov razvoj, s poudarkom na integraciji z ostalimi sistemi, vse pomembnejši. Za komunikacijo v realnem času se danes zelo veliko uporablja par protokolov CAN in CANopen.
Na obstoječem vgrajenem sistemu vzpostavite popolnoma funkcionalno vozlišče CANopen. Vgrajeni sistem v ta namen nadgradite z vso potrebno strojno in programsko opremo. Nove zmožnosti vgrajenega sistema preizkusite v realnem omrežju CAN.

Mentor:

prof. dr. Uroš Lotrič

Dekan:

prof. dr. Franc Solina

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Vlado Dimitrieski,

z vpisno številko 63040406,

sem avtor diplomskega dela z naslovom:

Vzpostavitev vozlišča CANopen na vgrajenem sistemu

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom

  izr. prof. dr. Uroša Lotriča

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 27.09.2010 Podpis avtorja:

Najprej bi se rad zahvalil mentorju, prof.dr. Urošu Lotriču. Izdelava diplomskega dela ne bi bila možna brez njegovih napotkov in usmeritev. Potem velja zahvala vsem zaposlenim v podjetju Evo-Teh, d.o.o, posebej Mateju in Davidu, ki sta me usmerjala in vedno pomagala. Na koncu bi se zahvalil svoji družini, ki me je podpirala in mi omogočila študij.

Povzetek

Protokola CAN in CANopen sta zelo primerna za uporabo v sodobnih omrežjih vgrajenih sistemov, saj sta načrtovana za delovanje v realnem času z visoko stopnjo zanesljivosti. Protokol CAN je serijska, robustna, hitra in zelo zanesljiva omrežna tehnologija, ki se uporablja za izmenjavo kratkih sporočil med mikrokrmilniki. Zaradi omenjenih lastnosti ga najpogosteje srečamo v avtomobilski industriji, v zadnjem času pa se vedno več uporablja tudi za komunikacijo med najrazličnejšimi vgrajenimi sistemi. Protokol CANopen najde svoje mesto v bolj kompleksnih omrežjih, v katerih je potreba po bolj abstraktnih načinih komunikacije z določeno stopnjo varnosti. Med drugim skuša z uvajanjem standardiziranih profilov naprav podati določeno stopnjo neodvisnosti od proizvajalcev naprav.

Vozlišče CANopen predstavlja potrebna strojna oprema, na kateri teče ustrezna programska oprema. Vozlišče mora z ustreznim krmilnikom, oddajno-sprejemno enoto CAN in ustreznimi gonilniki za vso strojno opremo zagotoviti nemoteno uporabo v omrežjih CAN.

V pričujočem delu so predstavljene osnove protokolov CAN in CANopen, ki so nujne za razvoj novega vozlišča CANopen. Nadalje je opisana uporaba ogrodja CanFestival in vse potrebne namestitve na operacijskem sistemu Linux za uporabo družine gonilnikov SocketCan. Predstavljena je sodobna platforma LUX, njene strojne komponente in potrebne nastavitve za programsko opremo, kot je operacijski sistem in gonilniki za brezhibno integracijo vseh delov sistema. Na koncu je predstavljen realen sistem, kjer novo izdelano vozlišče gospodar nadomešča komercialno vozlišče. Slednje predstavlja dokaz, da so vse naštete prednosti protokolov CAN in CANopen uspešno izvedene in imajo dejansko uporabo v praktičnem svetu.

Ključne besede:

- vgrajeni sistemi,
- CAN,
- CANopen,
- CanFestival,
- SocketCan,
- Linux.

Abstract

The use of the CAN and CANopen protocols is very common in embedded systems today, since they were designed to work in real time systems where high reliability is imperative. The CAN protocol is serial, robust, fast and very reliably network technology, used for interconnection of micro-controllers. As a result of the above, the CAN protocol is typically used in automobile industry. Lately we are witnesses of the expansion in the use of CAN in myriad embedded systems. The CANopen protocol on the other hand, alleviates the communication in more complex networks, where the necessity of more complex paradigms of communication, with respect to higher level of security, is inevitable. Moreover, with the introduction of its standardized device profiles, CANopen circumvents the vendor dependency of similar devices.

The typical CANopen node consists of adequate hardware equipment running the suitable software on top of it. The CANopen node, using a micro-controller, a CAN transceiver and proper drivers for the hardware, has to guarantee faultless communication over the CAN network.

The presented work provides the fundamentals of CAN and CANopen protocols, needed when developing a new CANopen node. Followed by the introduction of the open source framework CanFestival, the reader is provided with enough tools for development of a new node. Next we present the modern embedded platform LUX, its hardware components and all the necessary settings of the software(the Linux operating system and the SocketCan drivers), with one goal in our minds, integration of all the components in one flawless system. At the end follows a thorough presentation of the developed node, which replaces a commercial CANopen master node. That itself is a sufficient proof that a minimal set of CANopen functionalities were successfully implemented in an actual system with applications in the real world.

Keywords:

- embedded systems,
- CAN(Controller Area Network),
- CANopen,
- CanFestival,
- SocketCan,
- Linux.

Seznam uporabljenih kratic

CAN – *ang.* Controller Area Network, omrežje krmilnikov

PDO – *ang.* Process Data Object, procesni komunikacijski objekt

RPDO – *ang.* Receive Process Data Object, podatkovno procesni objekt za sprejemanje

TPDO – *ang.* Transmit Process Data Object, podatkovno procesni objekt za oddajanje

SDO – *ang.* Service Data Object, podatkovno servisni objekt

SSDO – *ang.* Server SDO, strežnik podatkovno servisnih objektov

CSDO – *ang.* Client SDO, odjemalec podatkovno servisnih objektov

SYNC – *ang.* Synchronization Object, sinhronizacijski objekt

EMCY – *ang.* Emergency Object, urgentni objekt

OD – *ang.* Object Dictionary, slovar objektov

LUX – ploščica LUX-SFX9 podjetja Evo-Teh, d.o.o.

Kazalo

1 Uvod

Vgrajeni sistemi(*ang. embedded systems*) v sodobnem slogu življenja so postali vseprisotni. Razlog za njihovo vse bolj pogosto uporabo je v nizki ceni zmogljive elektronike in velikih serijah izdelave. Pametne hiše, pametni mobilni telefoni, popolnoma avtomatizirani avtomobili s pametnimi sistemi so le del tega, kjer je možno najti vgrajene sisteme. Od najpreprostejših 8 bitnih mikrokrmilnikov za prosti čas do gigaherčnih modernih procesorjev v mobilnih telefonih. Dvoma ni, vgrajeni sistemi so našli svojo uporabo v vsakem kotičku sodobnega življenja in njihov čas šele prihaja.

Današnji trendi razvoja tehnologije zahtevajo vedno večje povpraševanje po čim večji integraciji visoko specializiranih sistemov. Slednje implicira potrebo po naprednih komunikacijskih rešitvah. Zlasti v sistemih, kjer je potrebna komunikacija v realnem času, na primer v vgrajenih sistemih, je potreben pameten pristop k integriranju več različnih komunikacijskih protokolov.

Tema pričujočega dela je praktična uporaba protokola CAN v povezavi z višjim protokolom CANopen na modernem vgrajenem sistemu, na katerem teče sodoben operacijski sistem. Opisana je izdelava in primer uporabe sistema, ki poleg komunikacije po protokolu CANopen omogoča, da se hkrati izvaja še množica najrazličnejših opravil za potrebe ciljnega uporabnika.

Rezultat je platforma, ki odpira vrata k popolnoma novemu načinu uporabe drugače nezdružljivih tehnologij. Z integracijo protokolov CAN in CANopen obravnavani vgrajeni sistem omogoča povezovanje konvencionalnih načinov komunikacije, na primer internetne povezave, z bolj specializiranimi povezavami, ki se množično uporabljajo v industriji. Če na takem sistemu teče internetni strežnik, lahko preko spleta nadzorujemo in upravljamo omrežje CAN oziroma CANopen. Ko imamo enkrat na voljo tak vgrajeni sistem, je nabor končnih aplikacij omejen samo še z idejami izdelovalcev.

V naslednjem poglavju so predstavljene osnove protokola CAN, njegove posebnosti in razlogi za množično uporabo v vgrajenih sistemih, posebej v avtomobilski industriji.

Tretje poglavje skuša podati odgovore na predhodno zastavljena vprašanja o potrebi po višjem protokolu ter okvirno opisuje posebnosti protokola CANopen. Najnujnejše znanje o uporabi in najpomembnejše zahteve protokola CANopen se nahajajo v tem poglavju.

Implementacija specifikacij protokola CANopen v okviru ogrodja CanFestival je opisana v četrtem poglavju. Na kratko je predstavljeno ogrodje z opisi najosnovnejših funkcij za gradnjo vozlišča CANopen.

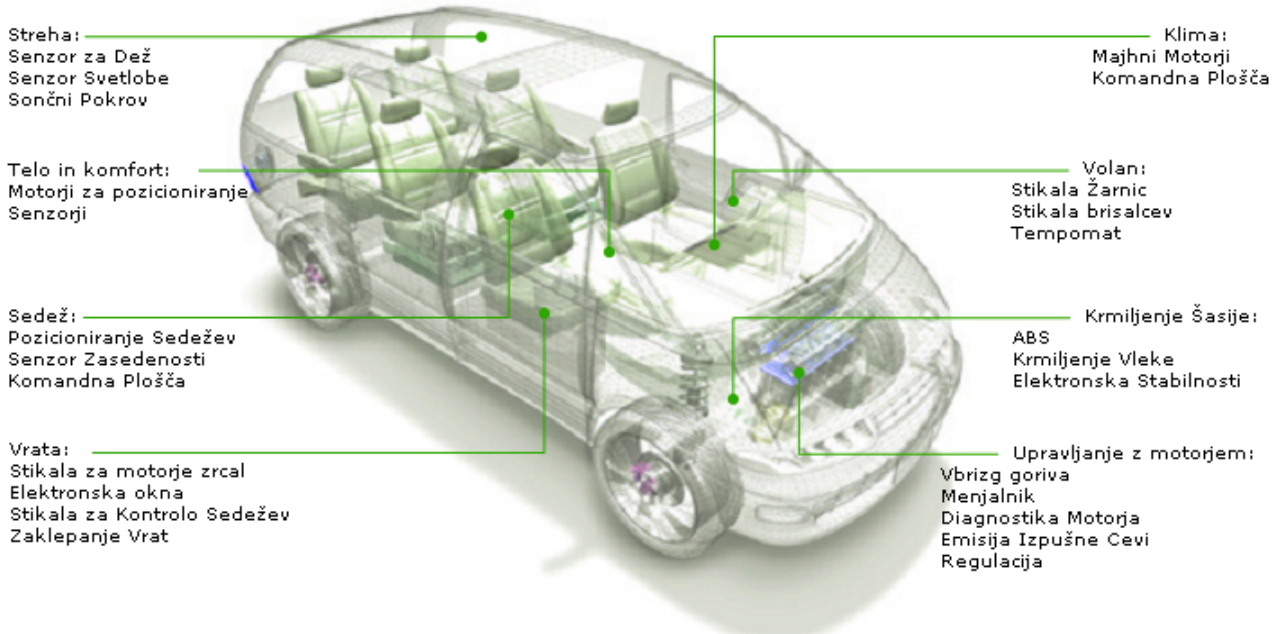
V petem poglavju nato sledi opis konkretne izdelave vozlišča na izbrani platformi, ki je v

šestem poglavju izvedena na realnem sistemu. Podrobneje je predstavljeno delovanje tipičnega vozlišča gospodar v omrežju CAN. Nazadnje so v zaključku zbrane najpomembnejše ugotovitve skupaj z idejami za nadaljnje delo.

2 Protokol CAN

2.1 Uvod

Omrežje krmilnikov (*ang. controller area network*), v nadaljevanju omrežje CAN, je temelj pričujočega dela, na katerega se korak za korakom nadgrajuje znanje za izdelavo celotnega sistema za visokotehnološko komunikacijo. Omrežje CAN je našlo svojo tržno nišo na globalnem trgu kot popularno sredstvo za povezovanje najrazličnejših elektronskih naprav v industriji. Kaj točno je omrežje CAN? Katere so njegove lastnosti? Katere so njegove glavne prednosti in pomanjkljivosti? Jasne odgovore na vprašanja bomo skušali podati v nadaljevanju. Tipično uporabo omrežja CAN v sodobnih avtomobilih prikazuje Slika 1.



Slika 1: Omrežje CAN v sodobnem avtomobilu

2.2 Potreba po enostavnem protokolu

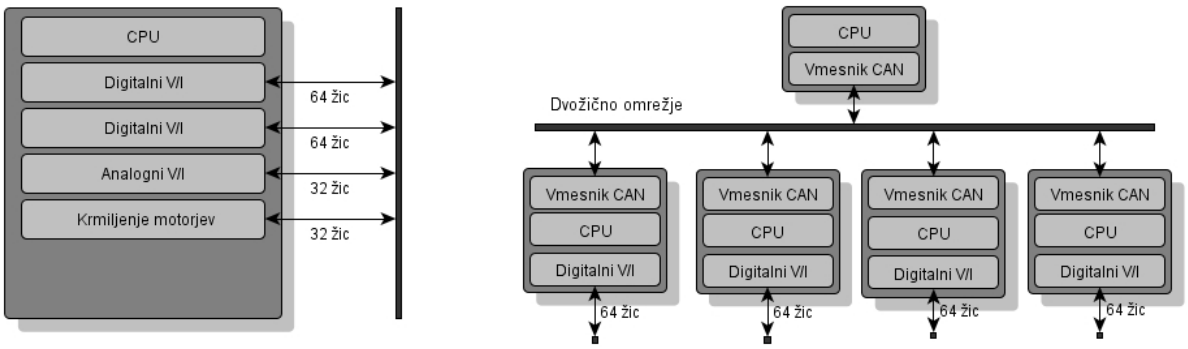
Do potrebe po novem omrežju je prišlo zaradi hitrega vdora elektronike v avtomobilsko industrijo. Obstoječi protokoli niso ustrezali zahtevi po zanesljivem, hitrem, robustnem ter preprostem načinu komunikacije. To je pripeljalo do iskanja nove, sveže rešitve, katere rezultat je bilo omrežje CAN. Najrazličnejši sistemi za avtomatizacijo, regulacijo in obveščanje so nekaj navadnega v sodobnih avtomobilih. Slika 1 prikazuje del elektronskih sistemov avtomobila.

Standard za omrežje CAN je bil razvit v Nemčiji, v podjetju Robert Bosch GmbH leta 1986 za potrebe podjetja Marcedes Benz. Komunikacija naj bi potekala v realnem času, med tremi enotami za krmiljenje motorja (*ang. engine control unit*), v nadaljevanju ECU. Po več neuspešnih poskusih vzpostavitve zanesljive komunikacije z uporabo povezave RS232 je namreč postalo jasno, da je neizbežen razvoj novega protokola. Poleg visoke stopnje zanesljivosti naj bi novi protokol omogočal, da je v omrežju hkrati prisotnih več gospodarjev (*ang. multimaster*) in obenem zmanjšal količino žic. Leto dni pozneje so v podjetju Intel že proizvedli prvi čip za omrežje CAN.

Razvijalci so takoj opazili velik potencial omrežja CAN in ga začeli uporabljati kot hrbtenico v komunikaciji med merilnimi in izvršnimi členi v avtomobilih. Od tedaj je omrežje CAN našlo svoje mesto tudi v industrijski avtomatizaciji, medicini, vojski, navtiki in povsod, kjer se uporabljajo vgrajeni sistemi. Leta 1993 so protokol CAN za omrežje CAN standardizirali. Danes je protokol CAN del standarda ISO 11898.

2.3 Kratek opis protokola CAN

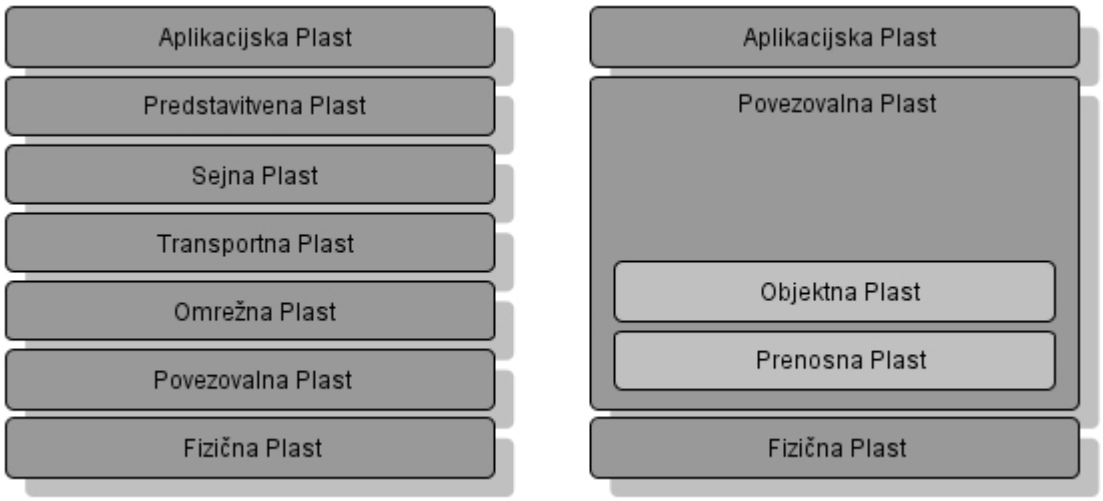
Protokol CAN je serijski komunikacijski protokol, s katerim je mogoče učinkovito porazdeljeno krmiljenje v realnem času. Razlika med porazdeljenim in centralnim krmiljenjem je prikazana na Sliki 2.



Slika 2: Primerjava centraliziranega in porazdeljenega protokola

Komunikacija običajno poteka po dveh žicah, je dvosmerna, vendar poteka le v eno smer naenkrat (*ang. halfduplex*) in je primerna za uporabo povsod, kjer aplikacije zahtevajo hiter prenos kratkih sporočil. Ponuja robustne mehanizme za odkrivanje in odpravljanje napak ter brezhibno deluje v kritičnih okoljih. Zagotavlja hitrost prenosa do 1Mb/s na razdalji do 40 m ter povezavo do 110 naprav. V primeru, da vsaka naprava uporablja le eno identifikacijsko številko, je na vodilo teoretično možno priklopiti do 2032 naprav. Omejitev na 110 naprav izhaja iz strojnega nivoja.

V protokolu CAN sta izvedeni spodnji dve plasti ISO/OSI modela, fizična plast ter povezovalna plast (*ang. data link layer*). Slednja je sestavljena iz plasti LLC (*ang. logical link control*) in plasti MAC (*ang. medium access control*). V verziji A[1] standarda CAN je plast LLC objektna plast, plast MAC pa prenosna plast. Obe plasti sta prikazani na sliki 3.



Slika 3: Primerjava modela OSI in strukture CAN. Pozor, povezovalna plast protokola CAN ne nadomešča vmesne plasti modela OSI, saj jih pri protoklu CAN preprosto ni

Naloge objektne plasti so določanje sporočil za pošiljanje po prenosni plasti, branje sporočil, sprejetih preko prenosne plasti, hkrati pa služi tudi kot vmesnik za povezovanje aplikacije in strojne opreme. V prenosni plasti je izveden prenosni protokol, v katerega na primer sodijo mehanizmi za kontrolo okvirov, opravljanje arbitraže, preverjanje in sporočanje napak. Tukaj se preverja, ali je vodilo prosto za oddajanje podatkov oziroma katero drugo vozlišče vodi prenos. V sami prenosni plasti ni možnosti za spremembe protokola.

Po drugi strani je za fizično plast, po kateri poteka dejanski prenos podatkov med vozlišči, značilna visoka stopnja fleksibilnosti. Vseeno pa morajo za naprave oziroma za vsa vozlišča v istem omrežju veljati ista pravila. Na primer, v istem omrežju morajo vsa vozlišča uporabljati isto bitno hitrost prenosa podatkov.

V nadaljevanju je predstavljenih nekaj osnovnih pojmov in značilnosti protokola CAN.

1 Protokola CAN je standardiziran v dveh verzijah, A in B. Verzija A uporablja za identifikacijsko številko naprave 11 bitov, verzija B, ki je nazaj združljiva z verzijo A, pa 29 bitov. V nadaljevanju se bomo omejili na verzijo A.

- Bitna hitrost mora biti v danem omrežju homogena.
- Z identifikacijsko številko CAN ID je določena prioriteta naprave. Naprave z nižjo številko imajo večjo prioriteto.
- Vsaka naprava je gospodar vodila (*ang. multimaster*). Ko je vodilo prosto, vsaka naprava lahko začne oddajati sporočilo. Da na vodilu ne pride do trkov, je potreben ustrezen mehanizem za arbitražo.
- Naprava sporočilo na vodilo pošilja v podatkovnem okviru, ki ima spremenljivo dolžino, vendar v naprej podano obliko. Takoj ko je vodilo prosto, vsaka naprava lahko začne oddajati novo sporočilo.
- Z zahtevo za oddaljene podatke (*ang. Remote Data Request, RDR) vozlišče* dostopa do podatkov nekega drugega vozlišča. V tem primeru imata tuji okvir (*ang. remote frame*) in podatkovni okvir enako identifikacijsko številko.
- Arbitraža med napravami na vodilu je deterministična. Med napravami, ki želijo oddajati sporočilo, dobi prioriteto tista z najnižjo identifikacijsko številko CAN ID, vse ostale naprave pa takoj začnejo sprejemati sporočila. Mehanizem arbitraže je izveden na nivoju bitov, dominanti bit ima večjo prioriteto kot recesivni bit[2]. Vse naprave neprestano spremljajo dogajanje na vodilu. Če je naprava poslala recesivni bit, medtem ko je prebrala dominantni bit, pomeni, da je izgubila arbitražo. V tem primeru takoj preneha z oddajanjem preostalih bitov v sporočilu in začne sprejemati tuji okvir. Tako se ne izgublja ne časa ne podatkov. Podatkovni okvir ima vedno prednost pred tujim okvirom z isto identifikacijsko številko.
- Za detekcijo napak skrbi več mehanizmov, med drugim primerjanje oddanih bitov z biti na vodilu, preverjanje pravilnosti okvirov, vključno s cikličnim redundančnim kodiranjem (*ang. cyclic redundancy check, CRC*), in vstavljanje neinformacijskih bitov v sporočilo (*ang. bit stuffing*).
- Vsi sprejemniki preverjajo pravilnost vsakega sprejetega sporočila, ki jo tudi potrdijo.
- Do signalizacije napak pride, ko katerakoli naprava zazna napako v sporočilu. Sporočilo, pri katerem je bila zaznana napaka, postane neveljavno. Čas okrevanja (*ang. recovery time*) je v najslabšem primeru enak času, potrebnemu za pošiljanje sporočila dolžine 29 bitov.

V omrežjih CAN vozlišča nimajo ne naslovov in ne informacije o konfiguraciji sistema, kar implicira nekatere pomembne posledice:

- fleksibilnost sistema – stare naprave se lahko izklapljajo, nove pa priklapljajo v omrežje brez posegov v programsko ali strojno opremo,
- usmerjanje sporočil – vsebina sporočila je določena z identifikacijsko številko –

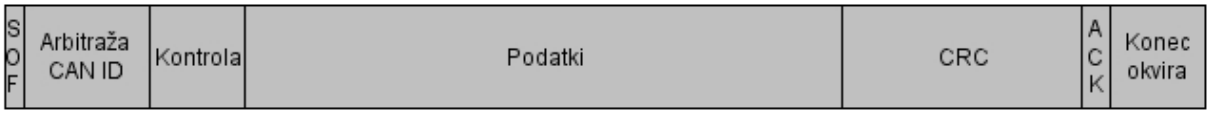
2 O dominantnih in recesivnih bitov več v poglavju 2.4

slednja ne določa lokacije oziroma cilja sporočila, ampak zgolj opisuje pomen podatkov, s čimer se odpira možnost filtriranja sporočil; vsako vozlišče odloča, ali je sporočilo zanj uporabno ali ne,

- oddajanje sporočila več sprejemnikom (*ang. multicast*[3]) ali oddajanje multicast – skupina vozlišč lahko istočasno sprejme in uporabi isto sporočilo,
- konsitenčnost podatkov – protokol CAN zagotavlja konsistenčnost sporočil; vse naprave bodisi potrdijo sprejem sporočila ali ga zavrnejo – če vsaj ena naprava ugotovi napako, potem vsa vozlišča razveljavijo trenutno sporočilo.

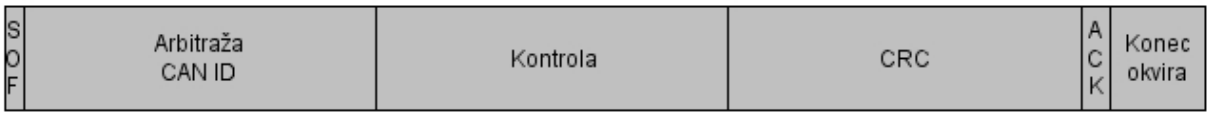
Standard uporablja štirje različne tipe okvirov. Vsak tip ima svoj natančno določen namen. Standard CAN pozna štiri tipe okvirov:

- Podatkovni okvir je prikazan na sliki 4 in je osnovni okvir za prenos podatkov v omrežjih CAN.



Slika 4: Format podatkovnega okvira

- Tuji okvir na sliki 5 je mehanizem, s katerim naprava zahteva prenos podatkov iz druge naprave.



Slika 5: Format tujega okvira

- okvir napake prikazan na sliki 6**,** ga pošlje katerakoli enota v primeru, da na vodilu odkrije napako.

Slika 6: Format okvira napake

3 Multicast je oddajanje enega toka podatkov več gostiteljem. Razlika med oddajanjem multicast in oddajanjem boradcast je v tem, da pri oddajanju multicast podatke sprejemajo samo izbrani sprejemniki, pri oddajnju broadcast pa vsi.

- Okvir za preobremenitev (*ang. Overload*) na sliki 7 pomaga pri ustvarjanju dodatne zakasnitve med zaporednimi okviri.

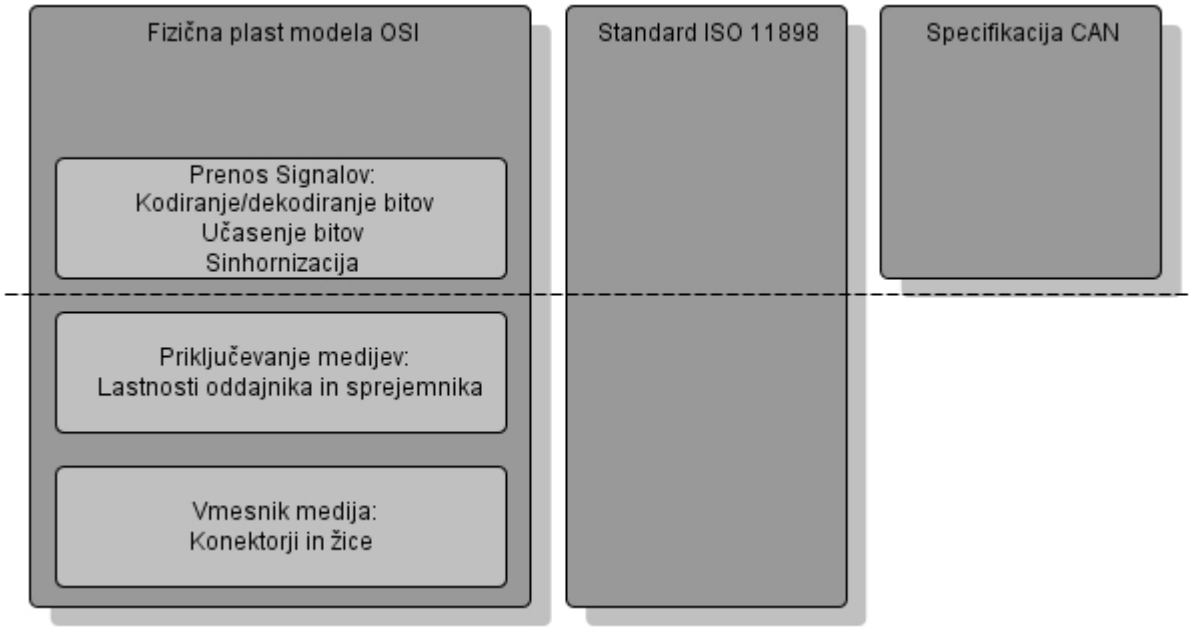
Slika 7: Format okvira za preobremenitev

Zahtevnejši bralec najde podrobni opis vsakega okvira v [1].

2.4 Fizična plast Protokola CAN in ISO 11898

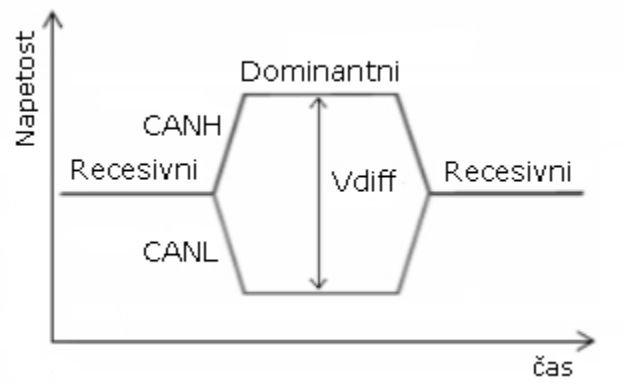
Sam protokol CAN ne definira celotne fizične plasti po modelu OSI, vendar le del tega. Ostanek fizične plasti ni del specifikacij CAN, ampak je del standarda ISO 11898. Odgovor na vprašanju do kam sega protokol CAN oziroma česa ne definira, je razviden iz slike 8. Kot je bilo omenjeno je mednarodna organizacija za standardizacijo postavila standard ISO 11898 katerega del je specifikacija CAN. Natančneje, specifikacija CAN je definirana v sklopu ISO 11898-1, ostale podplasti fizične plasti modela OSI pa so definirane v sklopu ISO 11898-2. Standard ISO 11898 je bil postavljen za hitro komunikacijo v avtomobilih z uporabo protokola CAN. Standard ISO 11898 definira celotno fizično plast skladno z modelom OSI z enim samim razlogom, zagotoviti kompatibilnost med različne oddajno-sprejemne enote CAN.

ISO 11898-2 dopolnjuje specifikacijo CAN z definicijo manjkajočih podplasti – podplast priključevanja medijev(*ang. Physical Medium Attachment*) in podplast vmesnika medija(*ang. Medium Dependent Interface*). Običajno je vsa specifikacija CAN implementirana v strojni opremi tipičnega krmilnika CAN.



Slika 8: Primerjava implementacije fizične plasti modela OSI med specifikacijo CAN in standardom ISO 11898

Protokol CAN določa dva logična nivoja, recesivni in dominantni nivo prikazana na sliki 9. ISO 11898 dodatno predpisuje predstavitev logičnih nivojev na dveh vodnikih s pomočjo protifazne napetosti. Dominantni bit povozi recesivnega, s čimer se doseže arbitraža na vodilu.

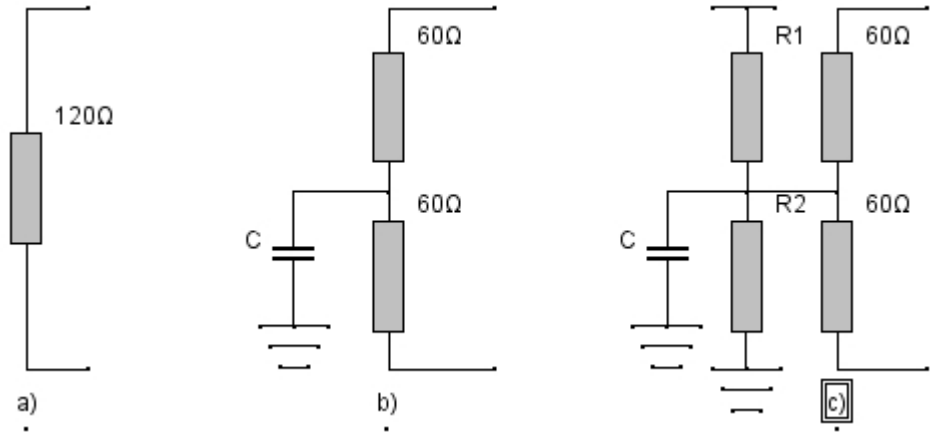


Slika 9: Fizična predstavitev logičnih nivojev signala ob predpostavki, da so naprave povezane z dvema vodnikoma (CANH in CANL)

Konektorjev in mehanskih žic standard ISO 11898 ne definira, zahteva le izpolnjevanje

ustreznih električnih specifikacij.

Zaključevanje(*ang. terminating*) vodila je potrebno za zmanjševanje odboja na vodilu. Po predpisih standarda ISO-11898 mora imeti vodilo v omrežjih CAN nominalno impedanco 120Ω. Zato je vrednost zaključnega upornika 120Ω. Obstaja več različnih načinov zaključevanja, nekateri so predstavljeni na sliki 10.



Slika 10: Zaključevanje vodila a) standardna zaključitev b)zaključitev z razcepom in c) pristranska zaključitev z razcepom

Vsak način pripomore k izboljševanju elektromagnetne združljivosti (*ang. electromagnetic compatibility*):

- standardna zaključitev, en upornik 120Ω, povezan na vsak konec vodila,
- zaključitev z razcepom je spremenjena standardna zaključitev, kjer sta namesto enega upornika 120Ω,dva upornika po 60Ω, povezana na vsak konec vodila, in kondenzator povezan na ozemljitev,
- zaključitev z razcepom, ki opravlja tudi funkcijo napetostnega delilnika tako, da med obema upornikoma doseže polovično napajalno napetost Vdd/2.

2.5 Prednosti protokola CAN

Prednosti je več, zlasti v primerjavi s protokolom TCP/IP oziroma povezavo RS232. Protokol CAN je zasnovan tako, da zagotavlja delovanje v realnem času, kar se najbolje izkaže pri kratkih sporočilih, s pametno arbitražo in močnimi mehanizmi za detekcijo napak. Protokol CAN je kljub nizki hitrosti prenosa podatkov 1Mb/s v primerjavi s 100 Mb/s pri protokolu Ethernet superioren, če se upoštevajo zahteve, kot so kratek odzivni čas, pravočasno odkrivanje napak, kratek čas okrevanja po napaki in popravljanje napak. Lastnosti protokola, ki so obenem njegove največje prednosti, so:

- definiranje prioritete sporočil in sistem za deterministično arbitražo,
- zagotovljena časovna zakasnitev,
- sočasno sprejemanje podatkov s sinhronizacijo oziroma oddajanje multicast,
- konsistenčnost podatkov na nivoju sistema,
- vse naprave v omrežju lahko nastopajo kot gospodarji,
- zanesljiva detekcija napak in njihovo sporočanje, oziroma
- samodejno ponovno oddajanje sporočil, oddanih z napakami,
- razlikovanje med začasnimi napakami in trajnimi odpovedmi naprav ter samodejno izklapljanje nedelujočih naprav.

Z ekonomskega vidika je protokol CAN eden najbolj gospodarnih protokolov, vključujoč tudi protokola TCP/IP in povezavo RS232. To je tudi eden od razlogov za njegovo popularnost. Na sliki 11 je prikazana prodaja čipov, ki podpirajo protokol CAN, po letih.

2.6 Pomanjkljivosti in zakaj je potreben višji protokol

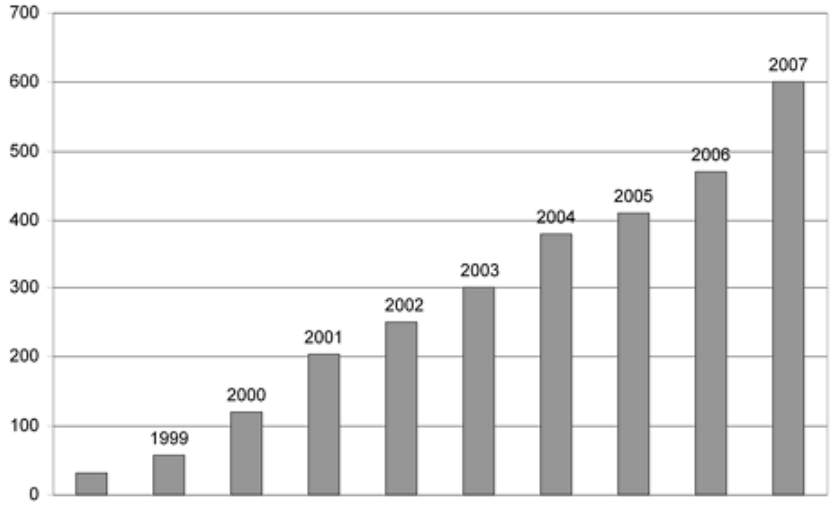
Čeprav je protokol CAN zelo lepo zasnovan, v standardu način povezovanja med povezovalno in aplikacijsko plastjo ni natančno podan. Vmesnih nivojev ni definiranih in jih vsak posameznih implementira sam po potrebi, iz tega razloga se je razvilo veliko število višjih protokolov. Pomemben razlog za višji protokol, zlasti v avtomatiki, je tudi implementacija razmerja gospodar / suženj. Sam protokol CAN je izredno učinkovit v avtomobilih in manjših aplikacijah, vendar za uporabo v bolj zapletenih sistemih, kjer je potreben nadzor in konfiguracija omrežja ter večji pretok podatkov, niti od daleč ni primeren. Eden izmed višjih protokolov, ki bazira na standardu CAN in se veliko uporablja v sodobnih sistemih, je zagotovo protokol CANopen. Poleg višjega protokola CANopen obstajajo še bolj specializirani protokoli, kot so DeviceNet za industrijsko avtomatizacijo, J1939 za avtobuse in tovornjake, Isobus za gozdarske in kmetijske stroje, MiLCan za vojaška vozila itd. Organizacija, ki razvija in podpira CANopen in tudi del zgoraj navedenih protokolov, je CiA (*ang. CAN in Automation*).

Razlogov za višji protokol je zagotovo več:

- ni mehanizmov za konfiguracijo vozlišč,
- ni načinov za upravljanje vozlišč,
- vozlišče kot koncept z vidika protokola CAN ne obstaja,

- velikost posameznega sporočila je lahko največ 8B.

Bistvo protokola CAN predstavlja preprosto, vendar hitro, zmogljivo in zanesljivo prenašanje podatkov. Zato so v standardu definirana le sporočila (okviri) in identifikacijske številke za potrebe arbitraže pri komunikaciji po vodilu. Definicije objektov, vozlišč in vseh drugih bolj zapletenih konceptov prepušča višjim protokolom, kot je na primer protokol CANopen.



Slika 11: Prodaja čipov CAN v milijonih, CiA

Takoj ko začnemo razmišljati o pomenu posameznega sporočila in identifikacijskih številk, se dejansko postavljajo temelji višjega protokola. Bolj logično se zdi izbrati že obstoječi protokol kot začeti razvijati novega. Za naše potrebe smo se odločili za protokol CANopen, ki je odprt protokol, specifikacije so brezplačne in jih vsakdo lahko dobi na internetni strani CiA [5]. Prav zaradi odprtosti protokola CANopen lahko optimiziramo aplikacije s tem, da vključimo samo funkcije, ki jih potrebujemo, ostale, ki niso potrebne, pa zanemarimo. V primerjavi s protokolom DeviceNet je protokol CANopen bolj fleksibilen z vidika fizične plasti. Tudi dovoljeno število vozlišč v enem samem omrežju je 127, kar je dvakrat toliko kot pri protokolu DeviceNet. Poleg tega je protokol CANopen že uveljavljen, zrel, stabilen protokol, ki ga uporablja velika množica ponudnikov opreme na globalnem trgu.

3 CANopen

3.1 Uvod

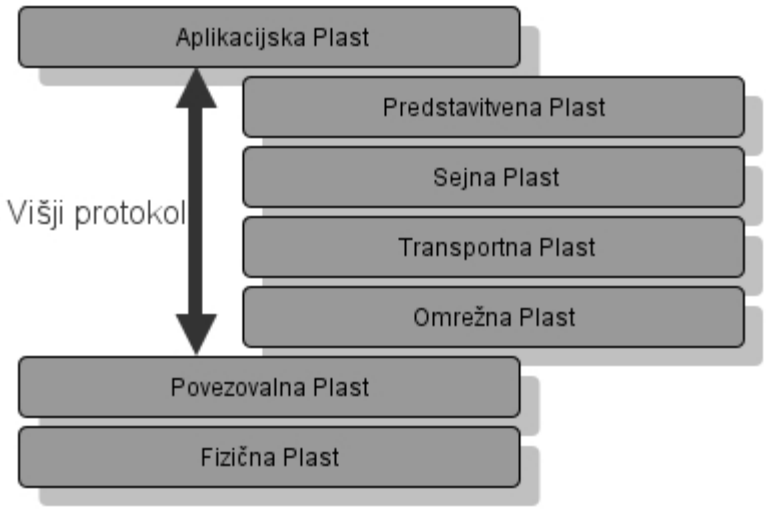
CANopen standardizira način, kako se podatki organizirajo in izmenjujejo med vozliči v omrežju. Na eleganten način premošča razdaljo med aplikacijo in omrežjem CAN[4]. Podpira gradnjo in uporabo serijskih modulov, ne da bi omejeval izdelave specializiranih produktov. Slednje omogoča proizvajalcem naprav z vmesniki CANopen izkoristiti že izdelane module in še vedno izboljšati sistem za optimalno ceno glede na zmogljivost. To seveda pomeni večjo konkurenčnost in hitrejši razvoj izdelka. Na naslednjih straneh so predstavljene bistvene značilnosti protokola CANopen.

3.2 CANopen kot višji protokol

V predhodnem poglavju je bilo pokazano, da samo mehanizmi protokola CAN niso zadostni za uporabo v sodobnih sistemih. Potreben je višji protokol, ki dopolnjuje osnovne mehanizme in storitve protokola CAN, ter uvaja še robustnejše in zanesljivejše koncepte. Protokol CANopen kot eden izmed višje ležečih protokolov v omrežju CAN je že uveljavljena rešitev. Slika 12 prikazuje njegovo mesto glede modela OSI. Je odprt in od proizvajalcev neodvisen standard, ki omogoča interoperabilnost med zelo različnimi napravami. Protokol CANopen od protokola CAN podeduje in nadgradi delovanje v realnem času z velikimi hitrostmi, kar so zaželene lastnosti vsakega dobro definiranega in odprtega standarda. Princip uporabi, kar potrebuješ, in odprta narava standarda prispevata k visokemu nivoju modularnosti.

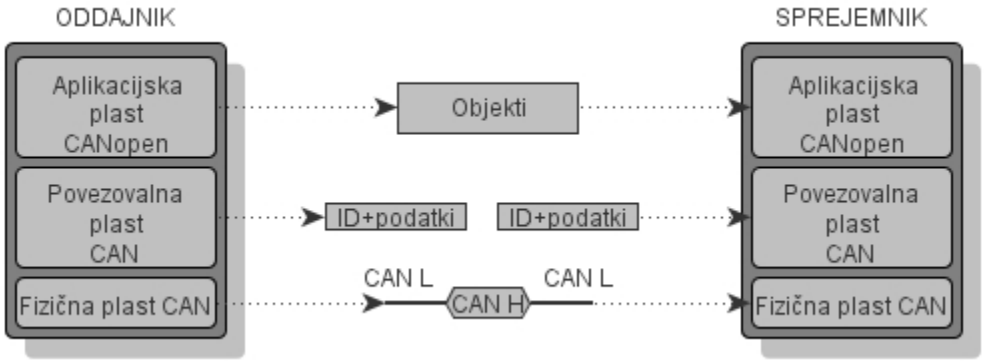
Standard ponuja mehanizme za sinhronizacijo in konfiguracijo omrežja s preprostim dostopom do parametrov ter s tem zagotavlja zanesljivo in deterministično obnašanje omrežja, kjer skalabilnost sistema ne implicira nepričakovane težave. Slika 13 grafično opisuje povezave med plastmi dveh vozlišč CANopen.

4 CANopen kot višji protokol ni omejen le na omrežjih CAN. Lahko se uporabi tudi v omrežjih EtherCAT, Ethernet Powerlink itd.



Slika 12: Referenčni model OSI in kje standard CANopen, se nahaja kot višji protokol

Zelo močno orodje protokola CANopen je koncept profila naprave, kar predstavlja dodaten razlog za izbiro standarda pred ostalimi konkurenčnimi. Profili predstavljajo način standardizacije podobnih naprav različnih proizvajalcev.



Slika 13: Interakcija med plastmi standarda CANopen

3.3 CAN v avtomatizaciji in standardi

Protokol CANopen predstavlja standardizirano izvedbo specifikacije CAL (ang. *CAN application layer*), ki so ga razvili člani združenja CiA (ang. Can in Automation) leta 1995. CiA je neprofitna organizacija pod sponzorstvom številnih podjetij in korporacij z različnih delov sveta. Prva objava standarda je bila sestavljena iz dveh delov, katerega prvi del predstavlja načrt

standarda DS 301, ki definira aplikacijsko plast,drugi del pa je predlog načrta standarda za ogrodja programabilnih naprav DSP 302. Oba načrta delno, vendar zadostno, pokrivata manjkajoče plasti OSI referenčnega modela, kar je prikazano na sliki 14.

Slika 14: Standard CANopen pokriva plasti 3-6 referenčnega modela OSI

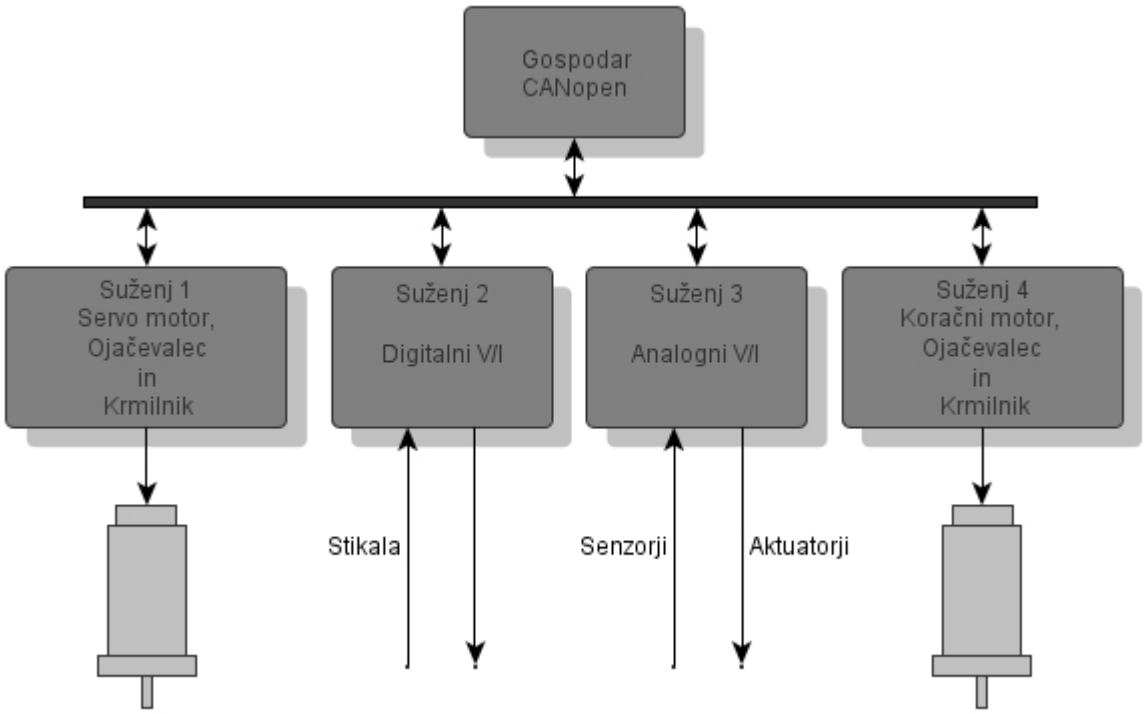
3.4 Profili naprav CANopen

Dodatki k standardu, ki ponujajo visok nivo standardizacije podobnih naprav, so profili naprav. Profil naprave določa podatke in komunikacijske modele, ki jih podpirajo različni moduli iste narave. Objekti, definirani v okviru profilov, so lahko procesni podatki, procesni ukazi ali izvršitvene funkcije, ki močno olajšajo delo uporabnikov standarda, kajti ni več potrebno prilagajati vmesnikov med napravami različnih proizvajalcev. V nadaljevanju je naveden le del profilov naprav, ki jih določa organizacija CiA:

- DS 401 – generični V/I,
- DSP 402 – pogoni in kontrola gibanja,
- DS 406 – profil naprave za enkoderje,
- DS 408 – profil naprave za ventile in prenosno hidravliko,
- DS 410 – profil naprave za klinometre,
- DS 412 – profil naprave za medicinske naprave,
- DS 414 – profil naprave za tkalske stroje,

- DS 418 – profil naprave za baterijske module,
- DS 419 – profil naprave za baterijske polnilce.

Zaradi novih potreb in razvoja različnih naprav se obstoječi profili nenehno dopolnjujejo ter dodajajo novi. Primer omrežja CANopen je predstavljen na sliki 15.



Slika 15: Primer omrežja CANopen

3.5 Povezava med standardoma CAN in CANopen

Protokol CANopen uporablja identifikacijske številke protokola CAN ter uvaja pojem vozlišča s pomočjo identifikacije komunikacijskega objekta (ang. Communication object identification, v nadaljevanju COB-ID). COB-ID je vsota identifikacijske številke okvira za določen komunikacijski objekt in identifikacijske številke vozlišča. V slovarju je vnaprej definirana identifikacija posameznega komunikacijskega objekta. To omogoča nedvoumno vključevanje identifikacije vozlišča ter identifikacije okvirja v eno samo sporočilo CAN. Na

primer, če je identifikacija vozlišča 02h in ima okvir prvega procesnega objekta za sprejem po specifikaciji DS 401 vrednost 180h, dobi okvir CAN objekta RPDO za drugo vozlišče identifikacijsko številko 1180h + 02h = 182h.

3.6 Osnovni modeli standarda CANopen

Za lažje in nemoteno razumevanje uporabe standarda CANopen so predstavljeni osnovni koncepti specifikacije. Cilj pričujočega besedila ni prikaz natančnega opisa protokola, temveč opisati temeljne mehanizme, s katerimi ta razpolaga. Radovedni bralci najdejo več preciznih informacij v uradnih specifikacijah [4]. V nadaljevanju so predstavljeni različni modeli, ki se uporabljajo v omrežjih CAN.

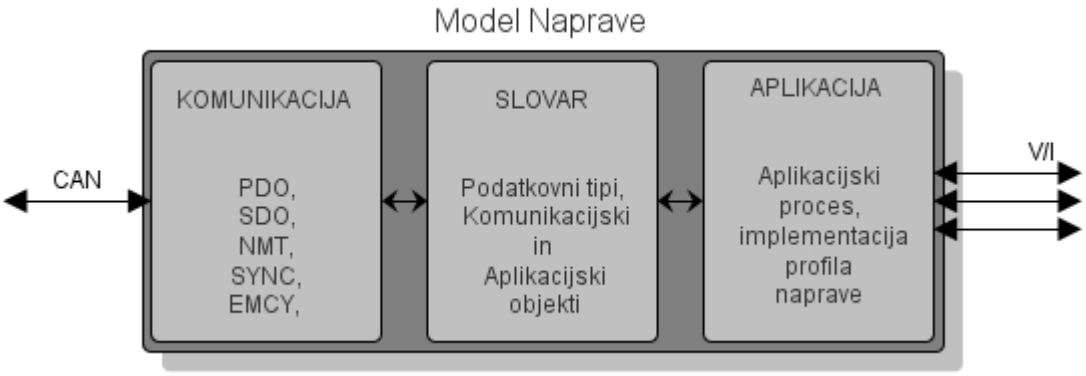
3.6.1 Referenčni model

Referenčni model je opisan v standardu ISO 11898 (CAN). Aplikacijska plast skrbi za konfiguracijo, komunikacijo in sinhronizacijo podatkov v realnem času. Funkcionalnosti, ki jih omogoča aplikacijska plast, so logično porazdeljene čez različne storitvene objekte. Torej objekt ponuja določeno funkcionalnost standarda in vključuje vse storitve, povezane s to funkcionalnostjo. Aplikacije, ki želijo med sabo komunicirati, to počnejo z uporabo različnih storitev, ki jih ponujajo objekti. Za uspešno izvedbo takšne komunikacije preko omrežja CAN mora celota potekati v skladu z vnaprej določenimi protokoli.

Storitve in njihovi protokoli so natančno opisani v [4], namen tega dela je naučiti bralca, kako uporabljati te storitve in ne ponuditi celotne razlage standarda. Zato so storitve in protokoli zgolj našteti v tabelah.

3.6.1 Model naprave

Enoto za komunikacijo predstavljajo komunikacijski objekti in ustrezni mehanizmi za prenos podatkov v omrežju, ki jih sestavlja slovar objektov (*ang. object dictionary, OD*), v katerem so navedeni vsi podatki, ki vplivajo na obnašanje aplikacijskih in komunikacijskih objektov ter končnega avtomata naprave. Aplikacija je dejanska funkcionalnost naprave v stiku z okoljem procesa, katere slovar objektov predstavlja vmesnik med aplikacijo in komunikacijo. Slika 16 grafično prikazuje opisani model naprave.



Slika 16: Model naprave standarda CANopen

3.6.1 Slovar objektov

Slovar objektov predstavlja jedro vsakega vozlišča CANopen in je točka, kjer so navedeni vsi parametri, ki določajo način komunikacije. Zajema opis, podatkovni tip in strukturo vsakega parametra. Sestavljen je iz več odsekov oziroma logičnih delov: komunikacijskega profila, podatkovnega profila, profila naprave ter posebnega profila za proizvajalce. Podroben ogled organizacije slovarja je predstavljen v tabeli 1.

Indeks(heksadecimalno)	Objekt
0000	Rezervirano
0001-001F	Statični podatkovni tipi
0020-003F	Kompleksni podatkovni tipi
0040-005F	Podatkovni tipi po meri proizvajalca
0060-007F	Specifični statični podatkovni tip v okvirju profila naprav
0080-009F	Specifični kompleksni podatkovni tip v okvirju profila naprav
00A0-0FFF	Rezervirano za kasnejšo uporabo
1000-1FFF	Komunikacijski profil
2000-5FFF	Profil proizvajalca
6000-9FFF	Standardiziran profil naprav
A000-FFFF	Rezervirano za kasnejšo uporabo

Tabela 1: Slovar objektov

Slovar predstavlja seznam objektov, ki so dostopni preko standardiziranih metod v omrežju CAN. Za naslavljanje objektov v slovarju se uporablja 16-bitni indeks in 8-bitni

podindeks. Večja optimalnost naprav je posledica dejstva, da se od proizvajalcev ne zahteva realizacija vseh objektov, vendar samo minimalna implementacija obveznih mehanizmov. Obvezni objekti zagotavljajo določeno obnašanje vozlišč istega tipa, ki je v skladu s standardom CANopen.

Za slovar je značilno, da podpira veliko različnih podatkovnih tipov, od preprostejših bitnih vrednosti do zapletenih struktur, kot so servisni objekti. Statični podatkovni tipi (boolean, integer, string) so definirani na naslovih 0001h-001Fh, kompleksnejši pa na naslovih 0020h-003F. Podatkovni tipi po meri proizvajalcev zasedajo naslove 0040h-005Fh. Omenjeni vnosi v OD niso le za referenco, spremembe niso možne, razen pri podatkovnih tipih proizvajalcev.

Komunikacijski profil obsega naslove 1000h-1FFFh. Na teh naslovih so parametri, ki določajo potek komunikacije v omrežju. Vnosi so enaki za vse naprave, edina razlika je morebiti prisotnost oziroma odsotnost neobveznih objektov. V tabeli 2 so primeri vnosov nekaterih komunikacijskih parametrov.

Indeks	Objektna koda	Ime	Podatkovni tip	Atribut[5]	O/N[6]
1000	VAR	Tip naprave	UNSIGNED32	b	O
1001	VAR	Register napak	UNSIGNED8	b	N
1003	ARRAY	Polje napak(prednast.)	UNSIGNED32	b	N
1200	RECORD	1 parameter SSDO	SDO Parameter(22h)	bp	O/N
1400	RECORD	1 parameter RPDO	PDO CommPar(20h)	bp	O/N
1600	RECORD	1 preslikava RPDO	PDO Mapping(21h)	bp	O/N

Tabela 2: Format vnosov v slovarju objektov

Indeksi igrajo pomembno vlogo pri definiranju objektov. V primeru preprostega tipa objekta je ta naslovljen kar neposredno z indeksom, ki pri zapisih in poljih kaže na naslov celotne strukture. Možnost naslavljanja posameznih elementov v sklopu struktur, lokalno ali preko omrežja, omogočajo podindeksi. Pri primitivnih podatkovnih tipih je podindeks nič, pri strukturah pa kaže na naslov določenega elementa strukture. Primer organizacije podatkov tipičnega procesnega objekta je podan v tabeli 3.

.

5 Atribut določa dostop do podatkovnega objekta. Branje, pisanje itd.
6 Obvezni/neobvezni(ang. *Mandatory/Optional*) objekt. Drugo ime za ta stolpec je Kategorija

Podindeks	Opis	O/N	Atribut	Preslikava PDO	Razpon vrednosti	Privzeta vrednost
0h	Največji podindeks	O	b	Ne	2-5	×
1h	COB-ID od PDO	O	b	Ne	UNSIGNED32	200+NodeID
2h	Tip oddaje	O	b	Ne	UNSIGNED8	Odvisno
3h*	Inhibirni čas	N	bp	Ne	UNSIGNED16	×
4h*	Kompatibilnost	N	bp	Ne	UNSIGNED8	×
5h*	Časovnik dogodka	N	bp	Ne	UNSIGNED16	×
*Se ne uporablja za oddajne procesne objekte(RPDO)						

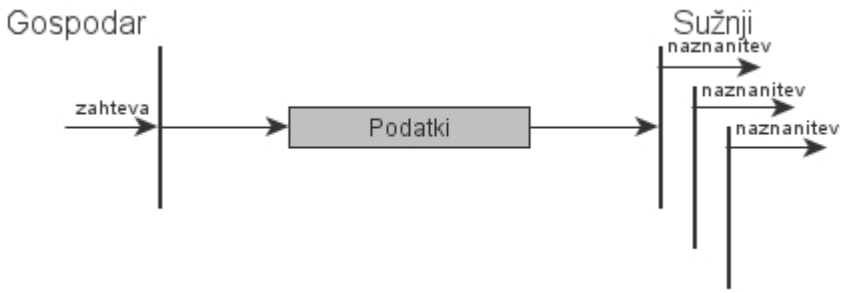
Tabela 3: Struktura procesnega objekta za oddajo(RPDO) na naslovu 1400h

3.6.1 Model komunikacije

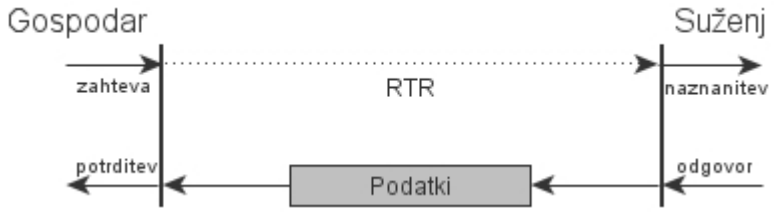
Model komunikacije omogoča različne načine komunikacije. Vsak način je ustrezen v določenih primerih, saj ponuja specifične objekte in storitve. Da ne pride do sestradanja objektov(*ang. starvation*) z nizkimi prioritetami, obstaja možnost dodelitve t.i. inhibirnih časov. Inhibirni čas definira najmanjši interval med dvema zaporednima klicema oddaje istega objekta. Sledi predstavitev različnih modelov komunikacije ki jih ponuja CANopen.

Način komunikacije gospodar/suženj

Za določene funkcionalnosti omrežja je potrebno eno samo vozlišče gospodar, ki to funkcionalnost ponuja ostalim vozliščem. Pogosto vse funkcionalnosti implementiramo kar v eno samo vozlišče. Seveda je možno različne funkcionalnosti razdeliti med različna vozlišča, vendar je to rahlo nepregledno. Gospodar dodeljuje zahteve sužnjem, ki glede na vrsto storitve odgovorijo brez potrditve (slika 17) ali pa z zahtevano potrditvijo, ki jo prikazuje slika 18.



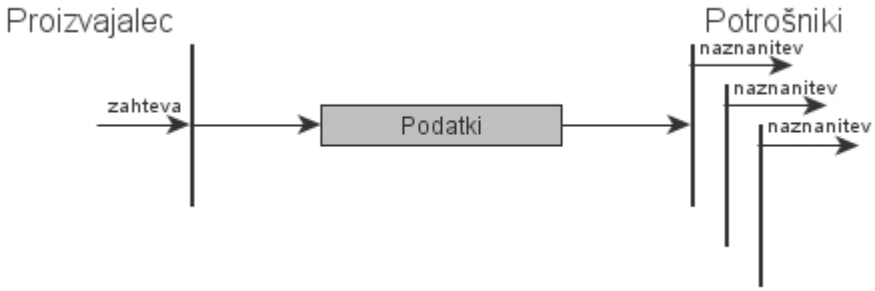
Slika 17: Komunikacija gospodar suženj brez potrditve



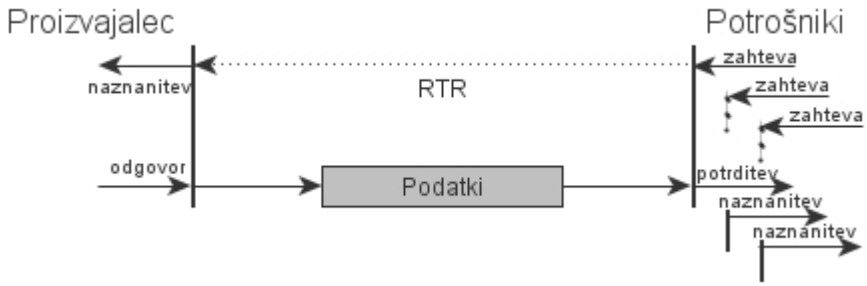
Slika 18: Komunikacija gospodar suženj s potrjevanjem

Način komunikacije proizvajalec/potrošnik

Vozlišču, ki izstavlja zahteve, pravimo proizvajalec, potrošniki pa so vozlišča, ki poslušajo. Pri modelu tipa potisni (ang. push) na sliki 19 potrošniki zahteve ne potrdijo. Pri modelu tipa povleci (ang. pull) na sliki 20 je potrošnik tisti, ki zahteva storitev, in ne proizvajalec. Storitev je s potrditvijo.



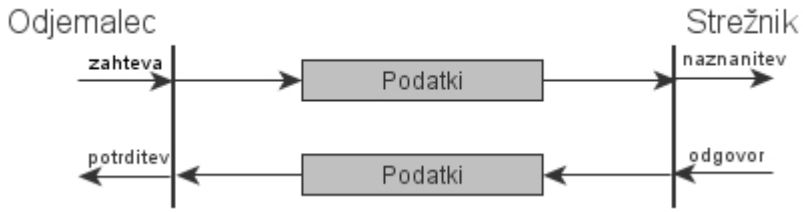
Slika 19: Komunikacija proizvajalec/potrošnik, model potisni



Slika 20: Komunikacija proizvajalec/potrošnik, model povleci

Način komunikacije odjemalec/strežnik

Pri komunikaciji odjemalec strežnik se dve vozlišči pogovarjata med seboj[7]. Odjemalec pošlje zahtevo strežniku za določeno nalogo, slednji po izvršitvi naloge pošlje odgovor nazaj. Komunikacijo odjemalec/strežnik grafično opisuje slika 21.



Slika 21: Komunikacija odjemalec/strežnik

3.7 Aplikacijska plast standarda CANopen

Če se bralec sprašuje, zakaj CANopen ne implementira druge plasti, razen aplikacijske, je odgovor naslednji: CANopen je višje ležeči protokol, ki za nižje ležeče plasti uporablja druge protokole, kot je CAN. Vsi objekti in storitve, ki te protokole uporabljajo, so na nivoju aplikacijske plasti. Slednja skrbi za abstrakcijo podatkov nižjih protokolov. Aplikacijska plast skuša vpeljati smisel okvirom nižjih protokolov na transparenten način. Zapletene strukture in protokoli izmenjave informacij morajo ostati skriti, toda način uporabe le-teh pa mora biti dovolj jasen in enostaven za uporabo in razumevanje razvijalcu. V naslednjih odstavkih so predstavljeni tisti objekti, ki jih opisuje standard CANopen in so potrebni za nemoteno praktično uporabo standarda.

3.7.1 Komunikacijski objekti

Komunikacija, ki poteka med različnimi vozlišči, je pravzaprav izmenjava komunikacijskih objektov. Za vsako funkcionalnost protokola obstaja določen komunikacijski objekt, s katerim je dosežena izvedba te funkcionalnosti. Storitve in protokoli opisujejo komunikacijske objekte standarda. Storitev, ki lahko zahteva potrditev objekta, ali pa tudi ne, predpostavlja, da ne pride do okvar na povezovalni plasti omrežja CAN. Sledi opis najpomembnejših komunikacijskih objektov standarda CANopen in obveznih storitev, ki jih mora vsako vozlišče implementirati.

7 Komunikacija v omrežju CAN je multicast, dejstvo, da se dve napravi pogovarjata med seboj, pomeni, da ostale naprave ignorirajo pakete, ki njim niso namenjeni.

Objekti za upravljanje z omrežjem

Protokol CAN ne pozna mehanizmov za upravljanje z omrežjem (*ang. network management*). Kot je bilo že omenjeno v drugem poglavju, ne pozna koncepta vozlišča, ampak sporočila. Protokol CANopen ponuja eleganten način za upravljanje omrežja s pomočjo objektov za upravljanje omrežja (v nadaljevanju objekt NMT). Tako lahko rečemo, da vozlišče, ki predstavlja gospodarja preko storitev in objektov NMT, upravlja z ostalimi vozlišči, ki jim pravimo sužnji. Za ta namen ima na voljo funkcije za: signalizacijo prehoda vozlišča v poljubno stanje (STOPPED, OPERATIONAL, PREOPERATIONAL), ponovni zagon vozlišča, ponovni zagon komunikacije itd. Tabela 4 prikazuje storitve v okviru modula za upravljanje z omrežjem.

V okviru objekta NMT so še t. im. storitve za kontrolo napak (ang. *error control*), s katerimi se zaznajo odpovedi v omrežju. Kontrola napak se izvaja s preprostimi, vendar zanesljivimi mehanizmi, ki bazirajo na periodičnemu pošiljanju sporočil. Obstajata dva mehanizma za kontrolo napak: varovanje vozlišč (*ang. node guarding*) ter srčni utrip (*ang. heartbeat*). Pri prvem gospodar preverja stanje ostalih vozlišč, pri drugem pa proizvajalci srčnega utripa ciklično pošiljajo sporočilo ostalim vozliščem. Implementacija enega izmed teh dveh protokolov je obvezna. V praksi se najpogosteje uporablja izraz srčni utrip.

Tretji in zadnji tip storitve upravljanja omrežja je zagonska storitev(*ang. boot up*). Sužnji obveščajo gospodarja, da so pripravljeni za priklop na omrežje s tem mehanizmom. Gospodar takoj, ko sprejeme objekt te storitve, spremeni stanje ustreznega sužnja v stanje OPERATIONAL. Tudi ta storitev je obvezna, kajti vozlišča nikoli ne bi drugače prešla v stanje OPERATIONAL. To pomeni, da vozlišča nikoli ne začnejo sodelovati v komunikaciji omrežja.

Storitev	Parametri	Potrditev	Protkol	O/N	
Start Remote Node	Node ID	×	Gosporar/Suženj	O	Storitve za upravljanje z vozlišči
Stop Remote Node	Node ID	×	Gosporar/Suženj	O	
Enter Preoperational	Node ID	×	Gosporar/Suženj	O	
Reset Node	Node ID	×	Gosporar/Suženj	O	
Reset Commnication	Node ID	×	Gosporar/Suženj	O	
Node Guarding Event	Node ID	×	Gosporar/Suženj	N	Storitve za kontrolo napak
Life Guarding Event	Node ID	×	Gosporar/Suženj	N	
Heartbeat Event	Node ID	×	Proizvajalec/Potrošnik	N	
Bootup Event	×	×	Gosporar/Suženj	O	Zagonska storitev

Tabela 4: Storitve za upravljanje z omrežjem

Objekt za sinhornizacijo

Proizvajalec oddaja objekt za sinhronizacijo(v nadaljevanju SYNC) vsem ostalim vozliščem. Proizvajalec objekta SYNC ni nujno vozlišče gospodar, čeprav je pogosto tako. Objekt SYNC je neke vrste ura omrežja. Čas med dvema ponovitvama objekta SYNC se imenuje komunikacijski cikel (*ang. comumunication cycle period)*, katerega naslov v slovarju je 1006h. Konfiguracija objekta SYNC poteka tako, da se nastavi ustrezni čas ter ustrezno identifikacijsko številko objekta SYNC (ta se nahaja na naslovu 1005h v slovarju). Logično je določiti identifikacijsko številko s čim višjo prioriteto, kar implicira pravočasno pojavitev objekta na vodilu. Poleg sinhronizacije naprav ta storitev služi tudi za sinhrono pošiljanje procesnih objektov. Storitev je brez potrditve in sledi modelu tipa potisni pri načinu komunikacije proizvajalec/potrošnik.

Urgentni objekt

Urgentni objekt(*ang.* e*mergency object, EMCY*) se sproži v izrednih, nujnih primerih kot na primer notranja okvara naprave. Objekt pošlje proizvajalec ostalim potrošnikom in je primeren za pošiljanje opozorila napak. Sproži se enkrat pri eni napaki, in dokler ni novih napak v napravi, tudi novih oddajanj urgentnega objekta ni. Protokol CANopen ne določa, kaj ukrepati v primeru okvar in odpovedi naprav. Dokumentacijo vseh zastavic in registrov napak bralec najde v [4].

Podatkovno servisni objekt

Edini način dostopa do vnosov slovarja preko omrežja je z uporabo podatkovno servisnih objektov (*ang.* s*ervice data object, SDO*). Komunikacija s servisnimi objekti[8] predstavlja zelo močan mehanizem za diagnostiko in konfiguracijo vozlišč. Sledi principu komunikacije odjemalec / strežnik. Vnosi v slovarju so različne velikosti in vsebujejo različne podatkovne tipe. Zato servisni objekti ponujajo prenos več množic (skupin) podatkov različnih velikosti. Vsebina in naslov posamezne množice podatkov sta vnaprej določena v slovarju.

Transfer servisnega objekta je v osnovi prenos zaporedja podatkovnih segmentov. Pred začetkom **segmentnega prenosa** odjemalec in strežnik določita pogoje prenosa v okviru faze inicializacije. Tukaj obstaja možnost za prenos podatkov velikosti 4B, temu pravimo **ekspeditivni prenos** in je najhitrejši od vseh, vendar je omejen na 4B. Če je potrebno prenašati ogromne količine podatkov, je na voljo še tretji način, **blokovni prenos**. Podobno kot pri segmentem prenosu je blokovni prenos zaporedje blokov, kjer je en sam blok lahko sestavljen iz največ 127 segmentov. Zadnji način nikakor ni primeren za prenos majhnih količin podatkov, vendar je nujen za prenos velikih količin podatkov. Odjemalec kot tudi strežnik imata možnost prekiniti transfer podatkov ne glede na načina prenašanja, vendar pa se vsak transfer začne izključno na zahtevo odjemalca.

Po navadi ima vsaka naprava svoj servisni objekt, lahko tudi več. Struktura servisnega objekta je podana na naslovu 22h v slovarju. Komunikacijski parameter servisnega objekta

8 V nadaljevanju se uporablja pojem servisni objekt namesto podatkovno servisni objekt.

opisuje zmožnosti servisnega objekta odjemalca oziroma strežnika. Sledijo naslovi vnosov v slovarju:

- SSDO (Strežnik servisnega objekta) indeks komunikacijskega parametra: 1200h + SSDO-številka – 1
- CSDO (Odjemalec servisnega objekta) indeks komunikacijskega parametra: 1280h + CSDO-številka – 1

kjer številka predstavlja zaporedno številko strežnika oziroma odjemalca. Komunikacijski parametri so obvezni za vsak servisni objekt.

Podrobnosti vseh protokolov servisnega objekta in storitev, ki jih ponujajo, so izven razpona tega dela. Tabela 5 prikazuje vse storitve servisnega objekta. Implementacija ekspeditivnega prenosa je obvezna. Implementacija segmentnega prenosa je obvezna v primeru transferjev objektov, večjih od 4B, pri čemer pa je implementacija blokovnega prenosa neobvezna.

Uporaba servisnih objektov za konfiguracijo vozlišč je preprosta, kompliciran mehanizmi so že implementirani v sami programski opremi komunikacijske enote[9]. Razvijalec le kliče ustrezne funkcije za izvršitev želenega transfera servisnega objekta. Zahtevnejši bralec lahko zadovolji svojo potrebo po celotnemu razumevanju komunikacije preko servisnih objektov definiranih v [4].

Storitev	Sestavne storitve	Potrditev	Protkol	O/N
SDO Download	Initiate SDO Download	✓	Odjemalec/ Strežnik	O
	Download SDO Segment	✓		
SDO Upload	Initiate SDO Download	✓	Odjemalec/ Strežnik	O
	Upload SDO Segment	✓		
SDO Block Downlaod	Initiate Block Download	✓	Odjemalec/ Strežnik	N
	Download Block	✓		
	End Block Download	✓		
SDO Block Upload	Initiate Block Upload	✓	Odjemalec/ Strežnik	N
	Upload Block	✓		
	End Block Upload	✓		
Abort SDO Transfer	Abort SDO Transfer	×	Odjemalec/ Strežnik	O

Tabela 5: Storitve servisnega objekta

Podatkovno procesni objekt

9 To pomeni da mehanizmi so del ogrodja ki implementira standarda CANopen.

Vsi doslej opisani mehanizmi so postavljeni z enim samim razlogom, da omogočijo brezhibni prenos podatkov v realnem času oziroma izmenjavo podatkovno procesnih objektov(*ang.* p*rocess data object, PDO*). Kot vsi objekti so tudi procesni objekti[10] vneseni v slovar in predstavljajo vmesnik do aplikacijskih objektov. Preslikava aplikacijskega objekta v procesnega in dodelitev podatkovnega tipa določa privzeta struktura procesnega objekta v slovarju. Če je možno poljubne spremenljivke preslikati v procesne objekte, se to naredi na začetku z mehanizmi servisnih objektov. Število in dolžina procesnih objektov je različno od aplikacije do aplikacije in se mora določiti v okviru profila naprave.

Obstajata dve vrsti procesnih objektov, procesni objekt za sprejem podatkov (*ang.* r*eceive process data object, RPDO*) in procesni objekti za oddajo podatkov (*ang.* t*ransmit process data object, TPDO*). Naprave, ki podpirajo oddajo procesnih objektov so proizvajalci, naprave, ki pa podpirajo sprejem procesnih objektov, so potrošniki. Parametra za procesni objekt sta dva, komunikacijski parameter je definiran na naslovu 20h, parameter preslikave pa na naslovu 21h. Komunikacijski parameter določa zmožnosti procesnega objekta, parameter preslikave pa neposredno povezuje spremenljivke aplikacije z vnosi procesnih objektov v slovarju. V nadaljevanju so predstavljeni vnosi v slovarju, s katerimi se lahko manipulira:

- indeks komunikacijskega parametra za sprejem procesnih objektov (RPDO): 1400h + RPDO-številka – 1,
- indeks komunikacijskega parametra za oddajo procesnih objektov(TPDO): 1800h + TPDO-številka – 1,
- indeks parametra preslikave za sprejem procesnih objektov (RPDO): 1600h + RPDO-številka – 1,
- indeks parametra preslikave za oddajo procesnih objektov (TPDO): 1A00 + TPDO-številka – 1,

kjer je številka zaporedna številka objekta TPDO/RPDO. Za vsak procesni objekt sta oba parametra nujna, saj tvorita povezavo med aplikacijo in komunikacijo.

3.7.1 Načini prenosa in sprožitveni mehanizmi procesnih objektov

Na voljo sta dva načina prenosa procesnih objektov, sinhroni in asinhroni. Sinhroni način uporablja objekt SYNC tako, da se takoj po pojavitvi le-tega procesni objekt pošlje s tekočimi podatki. Izbira želenega načina prenosa ter sprožitvenega načina se izvede preko parametra tipa prenosa. Obstajajo trije tipi prenosa:

- **tip 0** – sporočilo se odda po pojavitvi objekta SYNC, vendar aciklično (če se je le zgodila sprememba),
- **tip 1** – sporočilo se odda z vsakim objektom SYNC,

10 V nadaljevanju se uporablja pojem procesni objekt namesto podatkovno procesni objekt.

- **tip n** – sporočilo se odda z vsakim n-tim objektom SYNC.

Proizvajalci pošiljajo sinhrone procesne objekte za oddajo na enega izmed naštetih načinov. Tukaj se odpira možnost določanja hitrosti za oddajanje sinhronih procesnih objektov. Hitrost oddajanja je proporcionalna s hitrostjo oddajanja objekta SYNC. Asinhroni procesni objekti za oddajo se pošiljajo ne glede na pojavitev objekta SYNC.

Podatki sinhronih procesnih objektov za sprejem, prejetih po pojavitvi objekta SYNC, se posredujejo aplikaciji istočasno ob naslednji pojavitvi objekta SYNC, ne glede na način oddajanja. Podatki asinhronih procesnih objektov za sprejem se posredujejo direktno aplikaciji. Na koncu so prikazani še sprožitveni mehanizmi procesnega objekta:

- **dogodkovni** – oddajo povzroča dogodek pojavitve določenega objekta v napravi,
- **časovni –** oddaja je rezultat dogodka v sami napravi; če ni prišlo do dogodka po izteku vnaprej določenega časa, se oddaja vseeno izvede,
- **na zahtevo –** oddaja asinhronega procesnega objekta je posledica zahteve oddaljene naprave oziroma potrošnika procesnega objekta.

Storitve procesnega objekta so podane v Tabeli 6.

Storitev	Parametri	Potrditev	Protkol	O/N
Write PDO	Številka PDO, podatki	×	Proizvajalec/Potrošnik(push)	O
Read PDO	Številka PDO, podatki	✓	Proizvajalec/Potrošnik(pull)	O

Tabela 6: Storitve podatkovno procesnega objekta

3.7.1 Zagon sistema

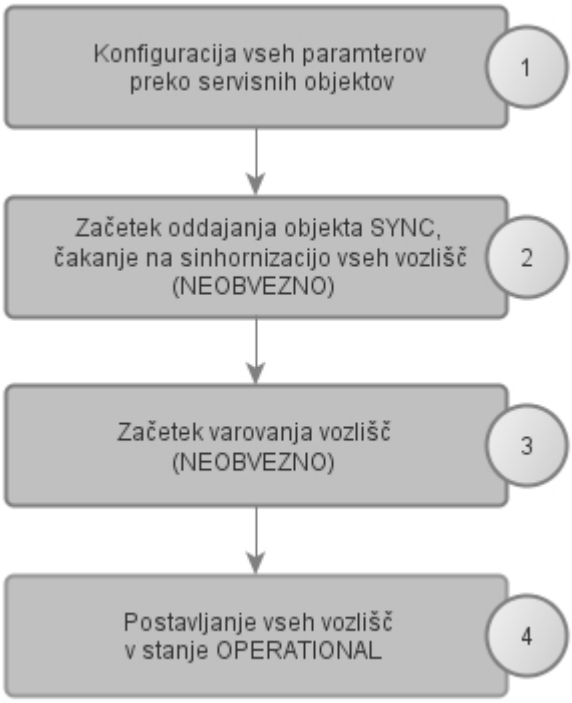
Preden se začne komunikacija med vozlišči, je potrebna inicializacija omrežja, kar narekuje inicializacijo vseh vozlišč. Na sliki 22 je prikazan potek inicializacije, ki ga vodi gospodar ali druga aplikacija za konfiguracijo.

V prvem koraku vsa vozlišča iz stanja INITIALISATION[11] samodejno preidejo v stanje PRE-OPERATIONAL. Stanje PREOPERATIONAL pomeni, da je slovar objektov vsakega vozlišča dostopen preko servisnih objektov. Preko orodij za konfiguracijo ali vozlišč gospodarjev je možno konfigurirati vsako vozlišče v omrežju. Da bi bilo slednje možno, mora vsako vozlišče gostiti strežnik servisnega objekta. Konfiguracija parametrov vseh vozlišč torej poteka v prvem koraku. V mnogih primerih konfiguracija ni potrebna, ker imajo naprave že privzete nastavitve in so pripravljene za takojšnjo (*ang*. plug and play) uporabo.

Kot je razvidno s slike 22, se drugi korak uporablja, če je potrebna sinhronizacija vozlišč in ni obvezen. Začne se oddajati objekt SYNC pred izvršitvijo naslednjega, tretjega, koraka.

11 Po mnenju avtorja angleška imena stanj so nujna zaradi konsistenčnosti s standardom

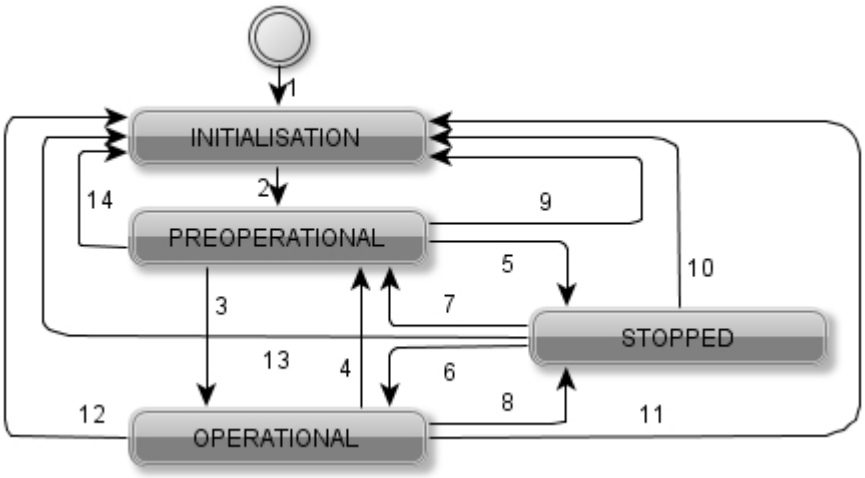
Slednji omogoča vklop mehanizmov za kontrolo napak. Tudi ta je neobvezen. Na koncu, v četrtem koraku, ko je vse nastavljeno, gospodar spremeni stanja vseh vozlišč v OPERATIONAL, kar pomeni začetek komunikacije v realnem času.



Slika 22: Potek inicializacije omrežja

Končni avtomat naprave

Diagram prehajanja stanj je prikazan na sliki 23. Stanje PREOPERATIONAL se doseže po začetni inicializaciji naprave, takoj po zagonu. Dokler je naprava v stanju PREOPERATIONAL, je možna sprememba parametrov preko servisnih objektov, potem pa lahko naprava preide v stanje OPERATIONAL.



Slika 23: Končni avtomat vozlišča CANopen

Končni avtomat naprave določa način delovanja komunikacijske enote. Minimalni zagon je sestavljen iz sporočila Start_Remote_Node.

Možna stanja v življenjskem ciklu naprave so naslednja:

- INITIALISATION, ki je sestavljeno iz podstanj INITISALISING, RESET APPLICATION in RESET COMMUNICATION,
- PREOPERATIONAL,
- OPERATIONAL,
- STOPPED.

Tabela 7 prikazuje dostopnost objektov, pogojenih s stanjem naprave.

Objekti	INITIALISATION	PREOPERATIONAL	OPERATIONAL	STOPPED
PDO			✓	
SDO		✓	✓	
SYNC		✓	✓	
EMCY		✓	✓	
BootUp	✓			
NMT		✓	✓	✓

Tabela 7: Stanje naprave in dostopnost objektov

4 Ogrodje CanFestival

4.1 Uvod

Eno izmed ogrodij (*ang. framework*), ki implementira standard CANopen, je CanFestival. Realizacija praktičnega dela te diplomske naloge temelji na tem ogrodju. V sklopu tega poglavja je razloženo, zakaj je bilo izbrano prav to ogrodje, katere so njegove prednosti in katere so njegove pomanjkljivosti. Potrebno je opozoriti, da to poglavje predstavlja zgolj opis ogrodja CanFestival in povzetek njegovih najpomembnejših funkcij, ki so potrebne za izdelavo tipičnega vozlišča CANopen. Uporaba teh funkcij je predstavljena v poglavju 5.

4.2 Kaj je CanFestival

Preprosto povedano je CanFestival ogrodje (*ang. framework*), ki omogoča razvoj in implementacijo vozlišč, temelječih na standardu CANopen. Lahko pa gledamo nanj tudi kot na orodje za razvijalce, skupek knjižnic in modulov, ki omogočajo hiter in brezplačen način za ustvarjanje vozlišča CANopen. Avtor tega dela ne bo predstavljal novih definicij in bo uporabil preproste definicije ogrodja CanFestival, zapisanega na strani [12]:

CanFestival je odprtokodno ogrodje standarda CANopen, dostopno pod licencama GPLv2 in LGPLv2.

4.3 Kaj omogoča CanFestival

CanFestival implementira omrežni sklad (*ang. network stack*) v jeziku ANSI-C, zanj pa je značilna visoka stopnja neodvisnosti od sodobnih platform. Končno vozlišče, ki je produkt uporabe CanFestival ogrodja, je bodisi gospodar bodisi suženj, ki teče na računalnikih,

mikrokrmilnikih ali celo v ogrodjih za medprocesno komunikacijo, ki zahtevajo delovanje v realnem času.

Praktične možnosti ogrodja CanFestival:

- implementacija vozlišča CANopen na marsikaterem mikrokrmilniku in računalniku,
- ustvarjanje in urejanje slovarja objektov,
- uporaba marsikaterega vmesnika CAN,
- povezava kode s komercialno(lastniško) kodo,
- brezplačno in lažje poučevanje CANopen.

4.4 Katere platforme podpira CanFestival

Vprašanje, ki je precej vplivalo na odločitev, katero ogrodje in orodje izbrati za izdelavo praktičnega dela diplomske naloge, je naslednje: ali bo ogrodje kompatibilno s ciljno platformo[12]? Več o izbrani platformi sledi v naslednjem poglavju, področje tega poglavja obsega posebnosti ogrodja CanFestival.

Kot se je pozneje izkazalo, CanFestival lepo in brez težav teče na ciljni platformi. Dejstvo, da je ogrodje CanFestival odprtokodno, je zgolj potrdilo odločitev za uporabo le tega. Tabela 1 prikazuje podprte platforme ogrodja CanFestival

4.5 Prednosti in slabosti

Težko je govoriti o prednosti in slabosti tako mladega ogrodja brez komercialnega ozadja. Med največjimi prednostmi so zagotovo odprta koda in možnost uporabe na operacijskih sistemih Windows in Linux. Ogrodje CanFestival je napisano tako, da se lahko uporablja tudi na sistemih, kjer ni operacijskega sistema. Po eni strani je to velika prednost, medtem ko po drugi tak način implementacije predstavlja zmogljivostni problem. Še ena prednost, vredna omembe, je grafično okolje, napisano v programskem jeziku *Python,* s pomočjo katerega se generira koda C za uporabo knjižnice CanFestival. Čeprav je ogrodje zaradi številnih pomanjkljivost včasih nerodno za uporabo, vseeno precej olajšuje delo in pospešuje razhroščevanje.

Največja slabost ogrodja CanFestival je pomanjkanje uporabne dokumentacije, kar je mogoče posledica nekomercialne narave projekta. Koristna uporaba je pogojena s celovitim razumevanjem standarda CANopen ter njegovih posebnosti. Predpostavlja se, da se na spletu

12 O ciljni platformi več v naslednjem poglavju, tukaj je pomembno, da na platformi teče jedro Linux in gonilnik SocketCan

najde vse, vendar za CanFestival skorajda ni uporabnih primerov, razen teh, ki so uradno vključeni v ogrodje. To seveda predstavlja še eno prepreko pri uporabi ogrodja CanFestival.

CAN naprava oziroma gonillnik	Linux			Windows			µC
	Vanilla	Xenomai	RTAI	VC++	Mingw	Cygwin	
SocketCan	✓	✓					×
Peak System	✓	✓	✓	✓	✓	✓	×
can4linux	✓						×
CAN-mVCCM				✓			×
LinCan	✓						×
IXXT				✓			×
VScom	✓			✓			×
Anagate	✓			✓	✓	✓	×
Kvazer				z napakami			×
UNIX pipes	✓					✓	×
Serial	✓						×
Motorola HCS12	×	×	×	×	×	×	z napakami
AVR	×	×	×	×	×	×	✓

✓ brez programske napake od zadnje uspešne potrditve
× neprimerno za uporabo

Tabela 8: Kaj podpira ogrodje CanFestival

Pri tehtanju slabosti in prednosti ni težko ugotoviti, da je, tako kot pri vsaki odločitvi, potrebno skleniti kompromise. Če bi se kdo lotil implementacije vozlišča po standardu CANopen, ima na voljo dve poti. Investirati čas z uporabo CanFestival in tako privarčevati sredstva, ali pa investirati v že obstoječi komercialni produkt in privarčevati čas. Seveda obstaja še tretja možnost – izdelava lastne implementacijo standarda CANopen. Na začetku se mogoče ideja zdi vabljiva in logična in je celo mogoča pri majhnih sistemih s preprostimi funkcionalnostmi. Vendar takoj, ko se pojavi potreba po nadgradnji sistema in vpeljavi novih funkcionalnosti standarda CANopen, postane jasno, da izbrani način ni gospodaren. Na koncu je vse odvisno od dobrin, s katerimi nekdo razpolaga. Če je imperativ čim krajši čas za razvoj izdelka, potem je komercialna rešitev prava izbira. Tipična komercialna rešitev obsega potrebna ogrodja za razvoj, podporo uporabnikov, različne učne pripomočke ter spodobno dokumentacijo. Nekatere komercialno dostopne rešitve so: CANbedded proizvajalca Vector, programska oprema CANopen proizvajalca IXXAT, MicroCANopen proizvajalca Embedded Systems Academy, knjižnice CANopen proizvajalca Datalink Engineering.

4.6 Obseg implementacije standarda CANopen

Ogrodje CanFestival ne implementira v celoti standarda CANopen, vendar samo bistvene funkcije v skladu s specifikacijami standarda. Implementacija vključuje DS 301 verzija 4.02 (ang. CANopen application layer and communication profile), zgoščen DFC iz DSP 302 ter storitve LSS iz DS 305.
Implementirane funkcionalnosti DS 301 so:

- gospodar in suženj NMT,
- srčni utrip za proizvajalca in za potrošnika,
- varovanje vozlišč brez sledenja,
- sinhronizacija z objektom SYNC,
- odjemalec in strežnik SDO, segmentni in ekspeditivni načni prenosa,
- PDO: TPDO in RPDO z vsemi tipi prenosa,
- EMCY: oddajanje, sprejem in sledenje urgentnih objektov,
- podatkovni tipi: 8- do 64-bitne vrednosti, znakovna zaporedja s fiksnimi dolžinami.

Kljub temu da velik del specifikacij CANopen ni implementiran, ogrodje CanFestival ponuja konkretno izhodišče za izdelavo vsakega profila standarda CANopen[13].

4.7 Zgradba ogrodja

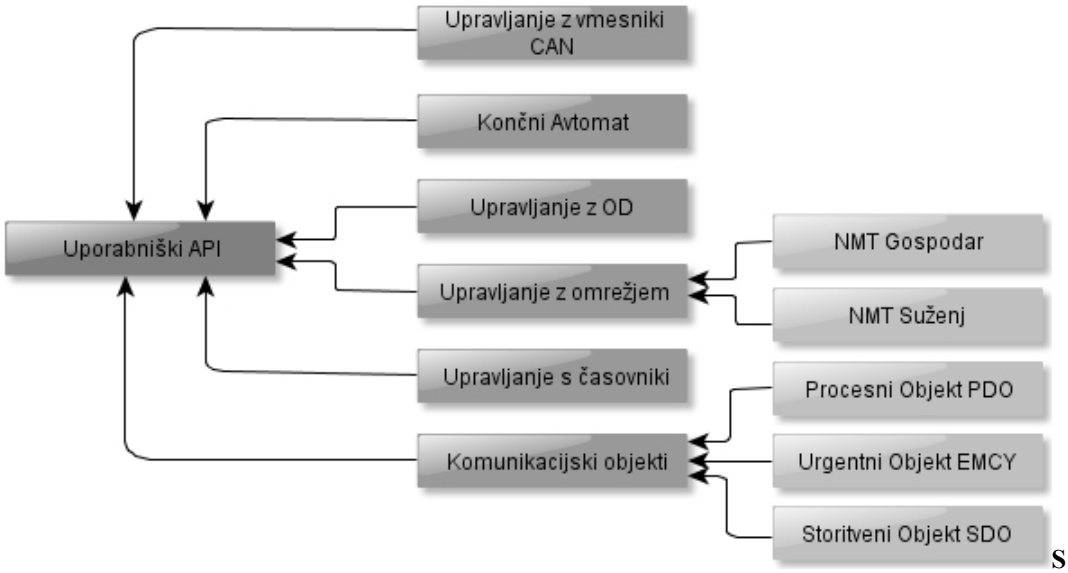
Funkcionalnosti ogrodja so na voljo preko programskega vmesnika (*ang. API*[14]). Uporabljajoč podane funkcije in strukture, se lahko zgradi modularno zasnovan sistem. V konkretnem primeru je to vozlišče CANopen.

Na Sliki 24 je predstavljen diagram sodelovanja med objekti programskega vmesnika

13 Jedro standarda CANopen je specifikacija DS 301, ki je v veliki meri implemenitrana. Manjka implementacija najrazlišnejših profilov CANopen

14 *ang. aplication programming interface*

CanFestival. Razvidna je organizacija ogrodja po modulih, kar je logična implikacija razdelitve storitev standarda CANopen.



S

lika 24: Ogrodje CanFestival

Kako uporabiti do sedaj predstavljene funkcije, je tema naslednjega poglavja. V tem poglavju je podan seznam s kratkim opisom najbolj pomembnih funkcij ogrodja CanFestival.

4.7.1 Osnovne strukture

Ogrodje ne razpolaga z velikim številom struktur. Ravno nasprotno, vsebuje le štiri strukture in jih uporablja precej varčno. Sledi pregled struktur s kratkim opisom:

- **MMessage,** vsebuje COB-ID, tip sporočila, dolžino in podatke,
- **struct_CO_Data,** vsi kritični podatki za definicijo vozlišča CANopen so v tej strukturi,
- **struct_s_BOARD,** konfiguracija naprave(vmesnika) CAN. Hrani ime vodila ter bitno hitrost,
- **struct_s_PDO_status,** hrani zadnjo sporočilo s statusom procesnega objekta za oddajo.

4.7.1 Upravljanje z vmesniki CAN

Modul uporablja strukturo Message ter strukturo struct_s_BOARD. Slednja poskrbi za povezavo med aplikacijo in ustreznim vmesnikom CAN. Naslednje funkcije omogočajo manipulacijo z vmesniki CAN:

- **canOpen()** se uporablja na začetku aplikacije za povezavo med aplikacijo in gonilnikom,
- **canChangeBaudRate()** poskrbi za spremembo bitne hitrosti naprave, kadar gonilnik to podpira,
- **canClose(),** zapre povezavo in sprosti uporabljene vire,
- **LoadCanDriver(),** nalaganje ustreznega gonilnika,
- **UnLoadCanDriver(),** sproščanje gonilnika.

4.7.1 Končni avtomat vozlišča

Sprehajanje med različnimi stanji je samodejno in skrito pred aplikacijo. Lahko pa razvijalec stanje vozlišča spremeni eksplicitno. Na primer pri inicializaciji je pogosto potrebno spremeniti nekatere vrednosti elementov v slovarju objektov. Slednje je možno doseči prav z uporabo prve od naslednjih funkcij, ki se sprožijo ob spremembi stanja:

- **void _initialisation()**
- **void _preOperational()**
- **void _operational()**
- **void _stopped()**
- **void _SlaveBootup()**

Pravzaprav je zadnja funkcija malo drugačna od ostalih. Razlikuje se v tem, da se uporablja izključno s strani vozlišč gospodarjev pri spreminjanju stanja iz stanja PREOPERATIONAL v stanje OPERATIONAL oziroma pri zagonu vozlišč sužnjev. Sledi seznam ostalih procedur za manipulacijo s stanji:

- **getNodeId(),** branje identifikacije vozlišča,
- **setNodeId(),** postavljanje identifikacije vozlišča,
- **getState()** in **setState()** za eksplicitno upravljanje stanj.

4.7.1 Upravljanje s slovarjem objektov

Doslej je bilo veliko povedano o slovarju objektov, najpomembnejši del standarda CANopen, kjer živijo vsi komunikacijski in aplikacijski objekti. Elementi slovarja se manipulirajo, s ciljem doseči želeno funkcionalnost vozlišča CANopen. Za orientacijo in sprehajanje po slovarju se uporabljata indeks in podindeks. Vnosi so dostopni za branje, če struktura dovoli tudi za pisanje.

Za dostope sta na voljo dva tipa funkcij, za dostop do lokalnega ter za dostope do oddaljenega slovarja. Za razliko od prvega tipa drugi tip upošteva pravilo debelega in tankega konca podatkov. Pregled funkcij:

- **getODentry(),** branje vnosa z upoštevanjem debelega in tankega konca
- **setODentry(),** branje vnosa z upoštevanjem debelega in tankega konca iz omrežja v obliko zapisa na napravi
- **readLocalDict(),** brez upoštevanja debelega in tankega konca
- **writeLocalDict(),** z upoštevanjem debelega in tankega konca

4.7.1 Upravljanje z omrežjem

Mehanizmi, s katerimi se uvaja red v omrežju, sodijo v ta modul. Sestavljata ga dva modula, gospodar NMT in suženj NMT. Gospodar NMT ponuja mehanizme za kontrolo in nadzor omrežja. Tega doseže preko manipulacije stanj in obnašanj posameznih vozlišč. Funkcije:

- **masterSendNMTstateChange(),** spremeni stanje določenega vozlišča,
- **masterSendNMTnodeguard(),** pošiljanje sporočila za varovanje vozlišč,
- **masterRequestNodeState(),** branje stanja v katerem se nahaja oddaljeno vozlišče.

Za sužnja NMT razvijalcu ni potrebno skrbeti, saj se ustrezne procedure izvedejo samodejno.

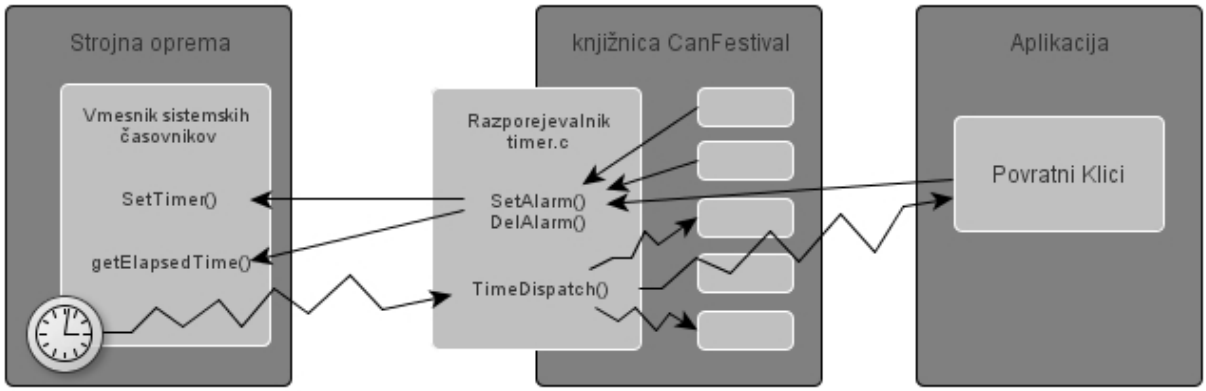
4.7.1 Upravljanje s časovniki

Modul ponuja vse potrebne mehanizme za uporabo časovnikov[15]. Če se bralec sprašuje, zakaj je uporaba časovnikov nujna, je odgovor, da mora vsako vozlišče CANopen podpirati uporabo zakasnjenih ukrepov(akcij). Torej, periodični SYNC, generiranje srčnega utripa ali časovna omejitev servisnega objekta postavijo svoje alarme. Po izteku določenega časa se alarmi

15 CanFestival ima le enega, vendar z njim oponaša več časovnikov.

sprožijo in izvršijo določena opravila. Mikrokrmilniki po navadi nimajo dovolj prostih časovnikov za direktno obravnavo zahtev standarda CANopen, zato ima CanFestival implementirane svoje programske časovnike, ki zasedejo le enega sistemskega. Časovnik je ene vrste mikro-razporejevalnik, njegova odgovornost je upravljanje tabele alarmov in ustrezno kronološko izvrševanje vsakega alarma [6].

Na Sliki 25 je prikazana interakcija časovnika z določenimi procedurami aplikacije.



Slika 25: Uporaba časovnika za izvajanje določenih funkcij

Seznam procedur v okvirju tega modula:

- **void TimerInit(),**inicializacija časovnika,
- **void TimerCleanup(),** pospravljanje časovnika,
- **void StartTimerLoop(),** zagon opravila časovnika,
- **void StopTimerLoop(),** ustavljanje opravila časovnika,
- **TIMER_HANDLE SetAlarm(),** zagon povratnega klica ob izteku alarma,
- **TIMER_HANDLE DelAlarm(),** brisanje alarma, preden se sproži,
- **void SetTimer(),** postavljanje časovnika.

4.7.1 Komunikacijski objekti

O komunikacijskih objektih je bilo dovolj povedano v prejšnjem poglavju, v tem pa se bolj osredotočamo na način njihove implementacije. Komunikacija v realnem času se doseže s procesnimi objekti (PDO), obveščanje o napačnem delovanju ter kakršnihkoli napak se izvede preko urgentih objektov (EMCY). Preden začne potekati komunikacija s procesnimi in urgentnimi objekti, je potrebno omrežje konfigurirati. Konfiguracija in nastavitve potekajo s

pomočjo servisnih objektov (SDO). Na tržišču so vozlišča s privzetimi nastavitvami, kjer konfiguracija ni potrebna. To so naprave tipa vklopi in uporabljaj (*ang.* plug and play).

Najprej so predstavljeni servisni objekti, ki so po mnenju avtorja najbolj zapleten koncept standarda CANopen. Po standardu komunikacija preko servisnih objektov poteka na tri načine. Ekspeditivno, segmentno in blokovno, od katerih ogrodje CanFestival implementira le prva dva. Število strežnikov in odjemalcev v enem vozlišču ni omejeno. Sledi jedrnat seznam procedur:

- **writeNetworkDict(),** pisanje v slovar oddaljenega vozlišča z identifikacijo nodeId, na lokaciji, ki je določena z indeksom in podindeksom,
- **writeNetworkDictCallBack(),** procedura, podobna zgornji, z uporabo povratnega klica(ang. c*allback*)
- **writeNetworkDictCallBackAl(),**procedura podobna zgornji z upoštevanjem debelega in tankega konca,
- **readNetworkDict(),** branje iz slovarja oddaljenega vozlišča z identifikacijo nodeId, iz lokacije, ki je določena z indeksom in podindeksom,
- **readNetworkDictCallBack(),** procedura podobna zgornji, ki uvaja uporabo povratnega klica,
- **readNetworkDictCallBackAl(),** procedura, podobna zgornji, ki upošteva debeli in tanki konec,
- **getReadResultNetworkDict(),** branje rezultatov procedure readNetworkDict,
- **getWriteResultNetworkDict(),** branje rezultatov procedure writeNetworkDict.

Procedura, ki se sproži ob pojavitvi napake, se prekrije. Kaj točno je napaka, standard CANopen ne definira. Torej je pojem napake od proizvajalca do proizvajalca različen.

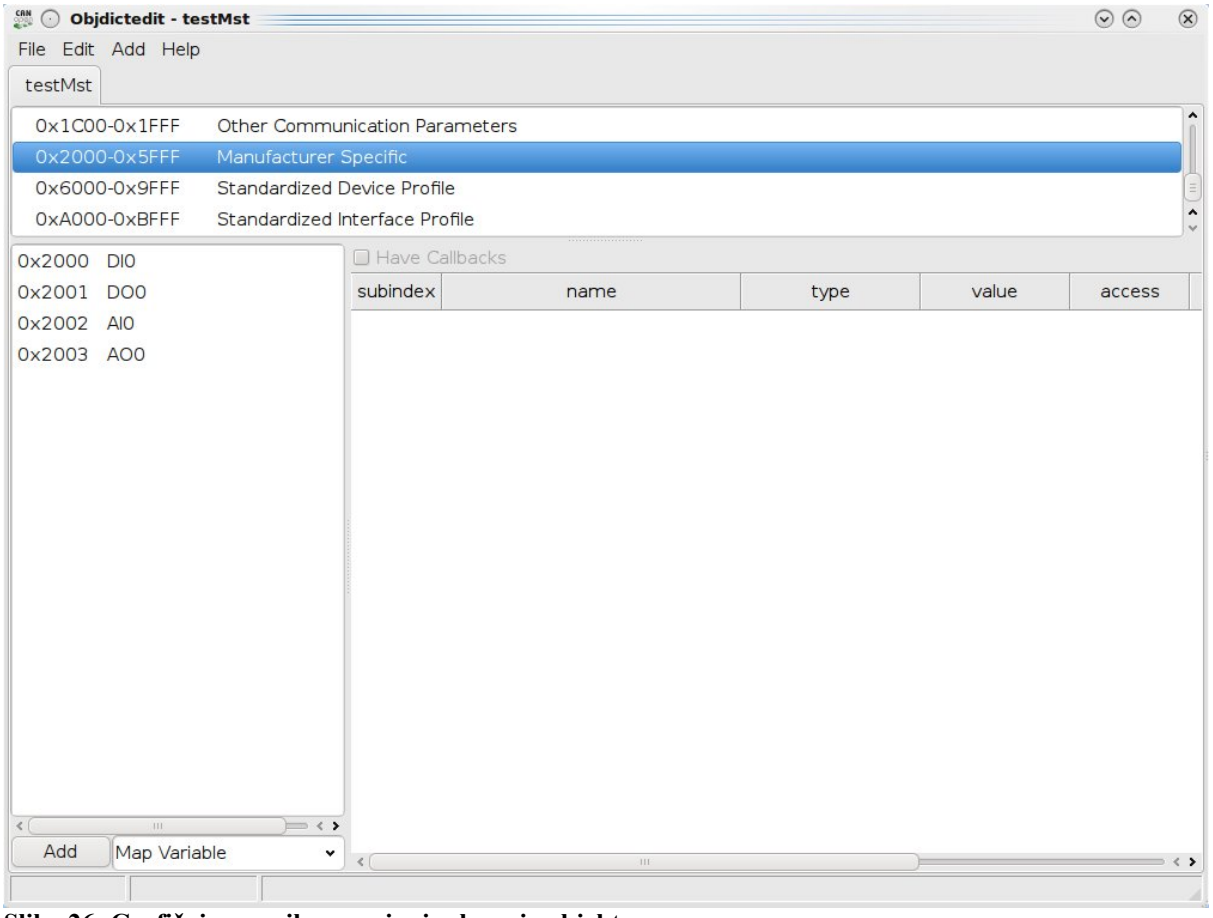
Procesni objekti so mehanizem za izmenjavo podatkov v realnem času. Komunikacijski parameter procesnega objekta in parameter preslikave procesnega objekta definirata procesni objekt.

Seznam procedur:

- **UNS8 sendPDOrequest(),** oddaja zahteve sužnju za procesni objekt,
- **UNS8 _sendPDOevent(),** procedura iterira čez vse procesne objekte za oddajoTPDO in pošlje samo tiste, pri katerih je prišlo do spremembe; oddaja je pogojena s tipom oddaje,
- **void PDOEventTimerAlarm(),** postavitev časovnika za dogodek procesnega objekta,
- **void PDOInhibitTimerAlarm(),** inhibitni časovnik za dogodek procesnega objekta.

4.8 Grafični vmesnik za izdelavo slovarja objektov

Običajno je logično uporabiti že obstoječe slovarje, vključene v ogrodju. Tudi če se ne razvija novega vozlišča, je potreben dostop do konfiguracije slovarja. Preprosti način je spremeniti kodo C glede na zahteve sistema. S tem pristopom ni nič narobe, vendar je slabša preglednost nad vsemi vnosi v slovarju. Tudi za najmanjšo spremembo je potrebno spreminjati veliko vrstic. Z drugimi besedami, fleksibilnost slovarja je precej na nizkem nivoju. Zato ogrodje CanFestival ponuja možnost uporabe grafičnega vmesnika, s katerim v nekaj korakih ustvarimo nova in hitro spremenimo že obstoječa vozlišča. Grafični vmesnik je prikazan na sliki 26.



Slika 26: Grafični vmesnik za urejanje slovarja objektov

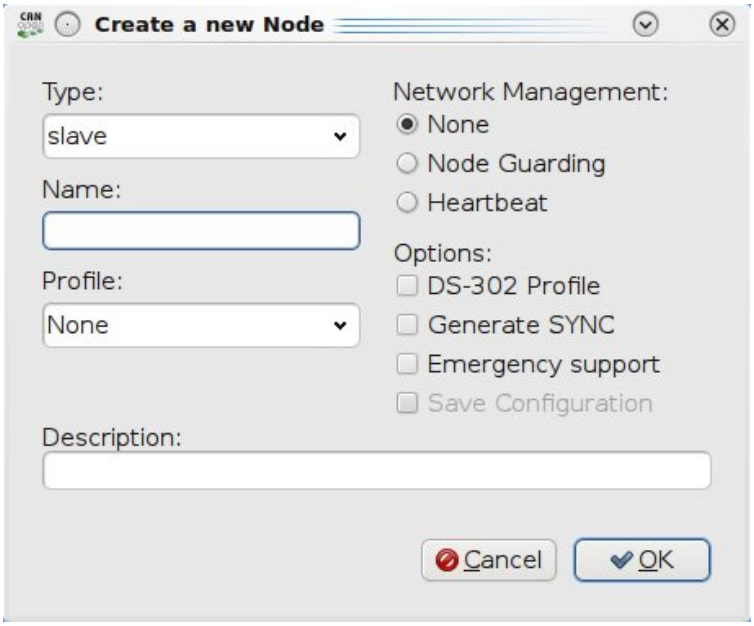
Zmogljivost grafičnega vmesnika je v njegovi sposobnosti, da sam generira datoteke C v skladu s podanimi zahtevami. Sledi predstavitev grafičnega vmesnika preko primera ustvarjanja novega vozlišča.

Izdelava novega vozlišča se začne s postavitvijo želenih zmožnosti, s katerimi bo vozlišče

razpolagalo. Enkrat, ko so lastnosti vozlišča določene, se ustrezne zahteve realizirajo preko grafičnega vmesnika. Grafični vmesnik je aplikacija, napisana z uporabo ogrodja WxPython in je preprosta za uporabo. Namestitev potrebnih orodij (WxPython, Gnosis XML utils) je predstavljena v [6]. Generiranje kode poteka bodisi preko grafičnega vmesnika s pritiskom File->Build Dictionary, ali preko ukazne vrstice:

python objdictgen.py XMLFilePath CfilePath

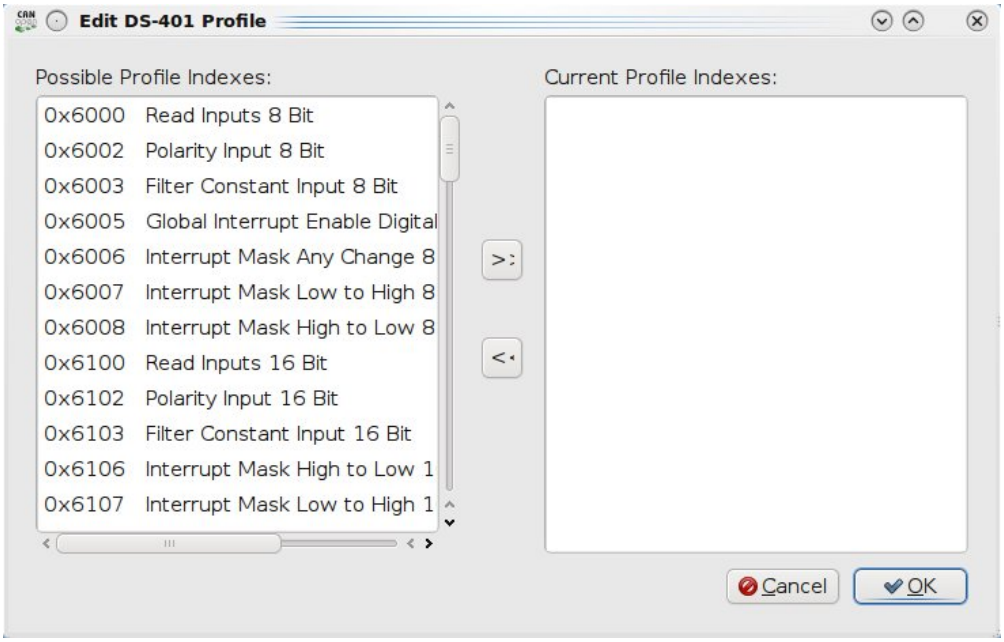
Datoteka XML s končnico .od je rezultat shranjevanja datotek iz grafičnega vmesnika. Uporabnik, z izbiro File->New ustvari novo vozlišče. Bistvene lastnosti vozlišča se izberejo v pojavnem oknu, prikazanem na sliki 27. Določi se tip vozlišča (gospodar ali suženj), proizvajanje objekta SYNC, podporo urgentnega objekta, način upravljanja z omrežjem in profil naprave.



Slika 27: Okno za ustvarjanje novega vozlišča

Vozlišče se ustvari ob pritisku na gumb OK. Uporabnik spreminja mehanizme standarda CANopen preko profila DS 301, ki ga prikazuje slika 26. Določanje procesnih, servisnih in urgentnih objektov poteka s spreminjanjem ustreznih vnosov v slovarju objektov. Urejanje profila DS 401 za generične V/I naprave prikazuje slika 28. Kot se vidi iz slike 26, slovar ni nič drugega kot tabela, ki zaseda največji del grafičnega vmesnika. Dodajanje strežnikov in odjemalcev, nastavitev ustreznih identifikacij in še veliko več se enostavno doseže s spreminjanjem vnosov v tabelo.

Zahtevnejši bralec si za globlje razumevanje ogrodja lahko prebere priročnik ter pogleda celotno kodo. V naslednjem poglavju je predstavljena implementacija standarda CANopen preko ogrodja CanFestival na platformi Lux9.



Slika 28: Profil DS 401 za generične vhodno-izhodne naprave

5 Implementacija in namestitev okolja

5.1 Uvod

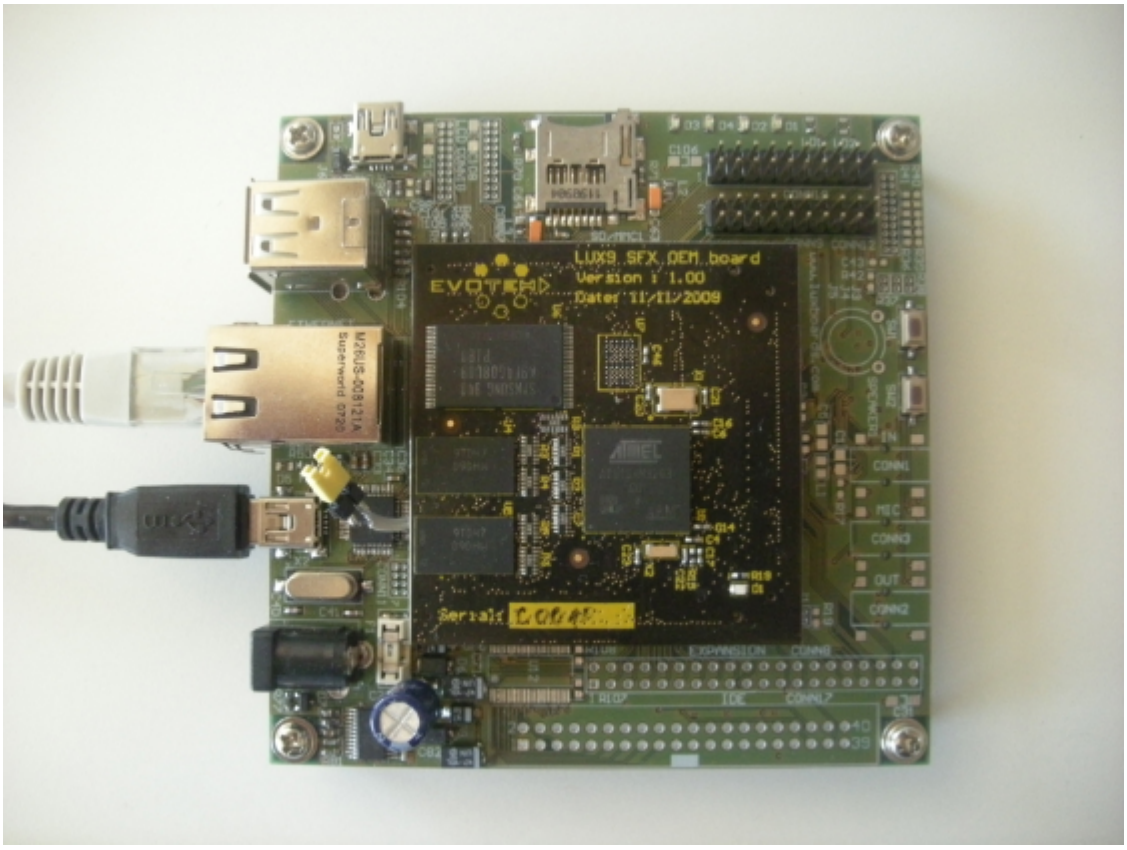
Naslednje besedilo v sklopu poglavja 5 predstavlja integracija vseh posameznih podsistemov za izdelavo vozlišča CANopen. Sledi opis razvojne platforme oziroma strojne opreme, ki je bila uporabljena za praktični del diplomskega dela. Poleg potrebne strojne opreme, kot je krmilnik CAN, oddajno-sprejemna enota oziroma čip CAN, je potrebno konfigurirati in inštalirati programsko opremo – operacijski sistem in potrebne gonilnike za krmilnik CAN. Nazadnje postavljena računalniška oprema predstavlja platformo za razvoj načrtovanega vozlišča CAN. Za uspešno realizacijo vozlišča je potrebno uporabiti izbrano ogrodje in vse ostale priložene vire.

V nadaljevanju najprej sledi predstavitev strojne opreme in njenih zmožnosti ter nato predstavitev programske opreme. V tem podpoglavju je prikazana nastavitev in inštalacija operacijskega sistema, postavljanje gonilnikov SocketCan za strojno opremo oziroma krmilnik CAN ter nalaganje in manipuliranje z moduli in ustrezna uporaba določenih pripomočkov za uporabo vmesnikov CAN. Nazadnje je prikazan agilen način uporabe ogrodja CanFestival za ustvarjanje vozlišča CANopen.

5.2 Strojna oprema

Razvojna platforma, s katero je realiziran praktični del te diplomske naloge, je prispevek podjetja Evo-Teh, d.o.o., in je prikazana na sliki 29. Ploščico LUX-SFX9 (v nadaljevanju LUX) poganja Atmelov mikrokrmilnik AT91SAM9263-CU, s frekvenco ure 200 MHz. Glavni pomnilnik je sinhroni dinamični pomnilnik SDRAM s kapaciteto 64 MB. Povezava med glavnim pomnilnikom in mikrokontrolerjem poteka preko 32-bitnega čelnega sistemskega vodila (*ang. front side bus*) s frekvenco 100 MHz. Shranjevanje podatkov je omogočeno na več različnih

načinov in predstavlja veliko prednost sistema. Dva priključka za pomnilniške kartice MMC/SD, dva priključka USB 2.0, 512 MB pomnilnika NAND flash ter krmilnik CF/IDE. Tudi komunikacijskih vmesnikov je več kot dovolj: 2 priključka USB 2.0, Ethernet 100Mb, serijski vmesniki UART, I2C in SPI. Za interakcijo z uporabnikom skrbi periferija, audio AC'97 in zaslon LCD TFT 320x240, občutljiv na dotik.



Slika 29: Ploščica LUX podjetja Evo-Teh d.o.o

5.2.1 Mikrokrmilnik AT91SAM9263

AT91SAM9263 je Atmelov sistem na čipu (*ang. system on chip, SOC*) s procesorjem 200 MIPS ARM926EJ-S. Arhitektura paralelnih vodil, ki vključuje celo 27 kanalov DMA[16], povzroča zmanjševanje podatkovnih ozkih grl sistema. Predvsem pri podatkovno preobremenjenih sistemih, ki uporabljajo operacijske sisteme, zaslone in internetne ali GPS povezave. Hitrost prenosa podatkov na nivoju čipa je 41,6 Gb, hitrost med čipom in periferijo je višja od 1Gb.

Število vmesnikov za interakcijo z uporabnikom je veliko. Vmesnik za kamero, krmilnik za zaslone TFT/STN LCD, 6 kanalni vmesnik za avdio(AC97) in vmesnik I2S in pomožni procesor za 2D grafiko, ki razbremenjujeta glavni procesor z avdio in grafičnim pospeševanjem. Periferija za komunikacijo vključuje 12 Mb gostitelj/naprava podporo za USB, povezavo 10/100

16 *ang. Direct Memory Access*

Ethernet in 1Mb omrežje krmilnikov CAN. Poleg tega so še štirje USART, 2 povezavi SPI po 50 Mb, CompactFlash, SDIO podpora ter TWI(I2C) dvožični serijaki vmesnik.

Z uporabo dveh vmesnikov za dostop do vodila je rešen problem deljenja pomnilnika med procesorjem in krmilnikom za zaslon LCD. S tem se doseže povečevanje procesorskega MIPS za 20 % do 40 %, vendar pa je potrebno dodati tudi to, da drugo vodilo, ki je namenjeno pospeševanju grafike, deli signale s povezavo Ethernet. S tem je onemogočena uporaba grafičnega pospeševalnika in potezave Ethernet istočasno.

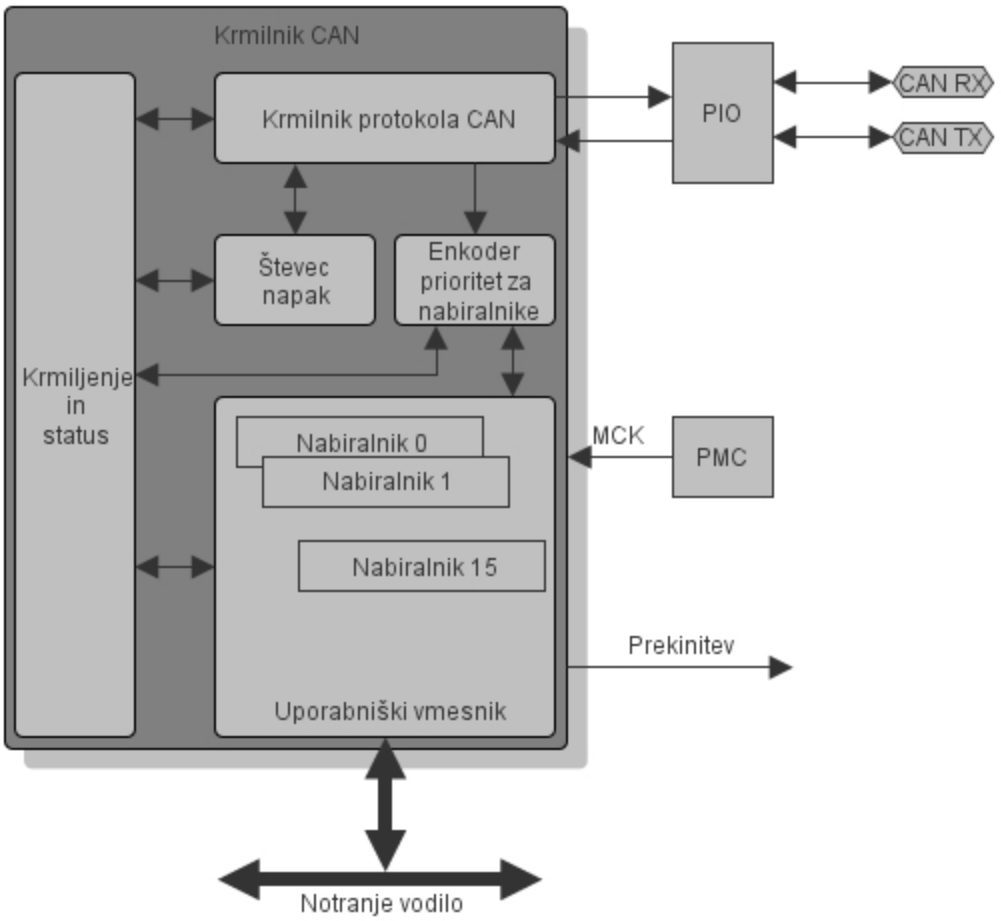
Sledi opis krmilnika CAN, informacije o ostalih karakteristikah mikrokrmilnika pa se nahajajo v [14].

5.2.1 Krmilnik CAN

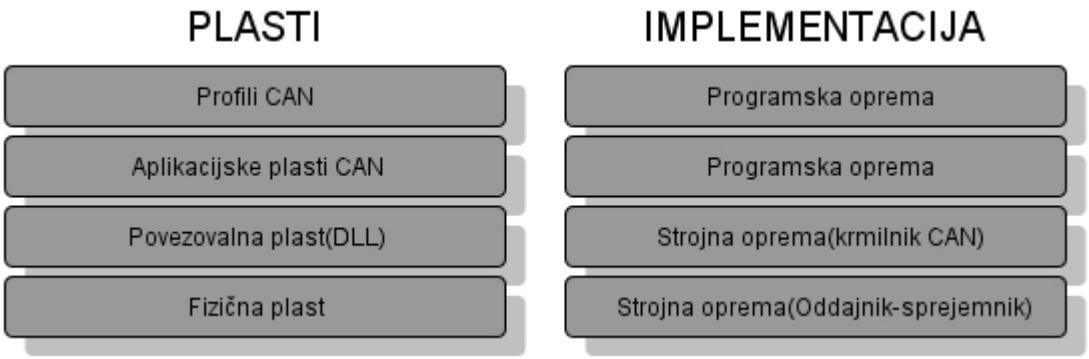
Krmilnik CAN v sklopu mikrokrmilnika AT91SAM9263-CU je skladen s standardom ISO/11898A za visoke hitrosti. Implementira vse značilnosti protokola CAN vključujoč uporabo vseh tipov okvirov (podatkovnih, oddaljenih, za napake, za prekrivanje) pri hitrosti do 1 MB/s. Dostop do krmilnika je možen preko konfiguracijskih registrov. Za komunikacijo v omrežju CAN je na voljo 16 neodvisnih medpomnilnikov (nabiralnikov) za oddajanje in sprejemanje sporočila. Blokovna shema krmilnika je prikazana na sliki 30.

Nabiralniki imajo možnost nastavitve za sprejemanje ali oddajanje sporočil. Sprejemanje sporočil poteka preko izbranih nabiralnikov. S povezovanjem več nabiralnikov je možno narediti še večji medpomnilnik, kjer ni nujno, da so nabiralniki zaporedni. Nastavitev maske nabiralnika pomeni sprejemanje določenih sporočil, pri čemer se ostali ignorirajo. Kadar se medpomnilnik napolni, se sproži prekinitev. Glede na nastavitev nabiralnika se bodisi čaka na potrditev na prvo sprejeto sporočilo bodisi se vsebina razveljavi ob sprejemu novega sporočila. Oddajanje sporočil poteka skoraj identično kot sprejemanje. Nastavitev več različnih nabiralnikov za oddajanje kakor tudi prioriteta je opcija ki se lahko določi za posamezni nabiralnik.

Krmilnik CAN v sklopu mikrokrmilnika AT91SAM9263 je zasnovan v skladu s potrebami za strojne rešitve v omrežjih CAN. Kot smo omenili, omogoča poljubno ustvarjanje medpomnilnikov za oddajanje in sprejemanje sporočil z veliko različnimi maskami pri vsakem nabiralniku oziroma medpomnilniku. To je zelo zaželeno v preprostejših sistemih, vendar gonilniki operacijskih sistemov ne izkoriščajo teh strojnih rešitev zaradi kompatibilnosti s čim večjim številom krmilnikov različnih proizvajalcev oziroma zaradi ohranjanja principa generičnosti gonilnika. To se dogaja tudi pri gonilniku, ki se uporablja na ciljni platformi, zato je opis posebnosti krmilnika CAN zanemarjen. Tehnologija implementacije posamezne plasti je prikazana na sliki 31.



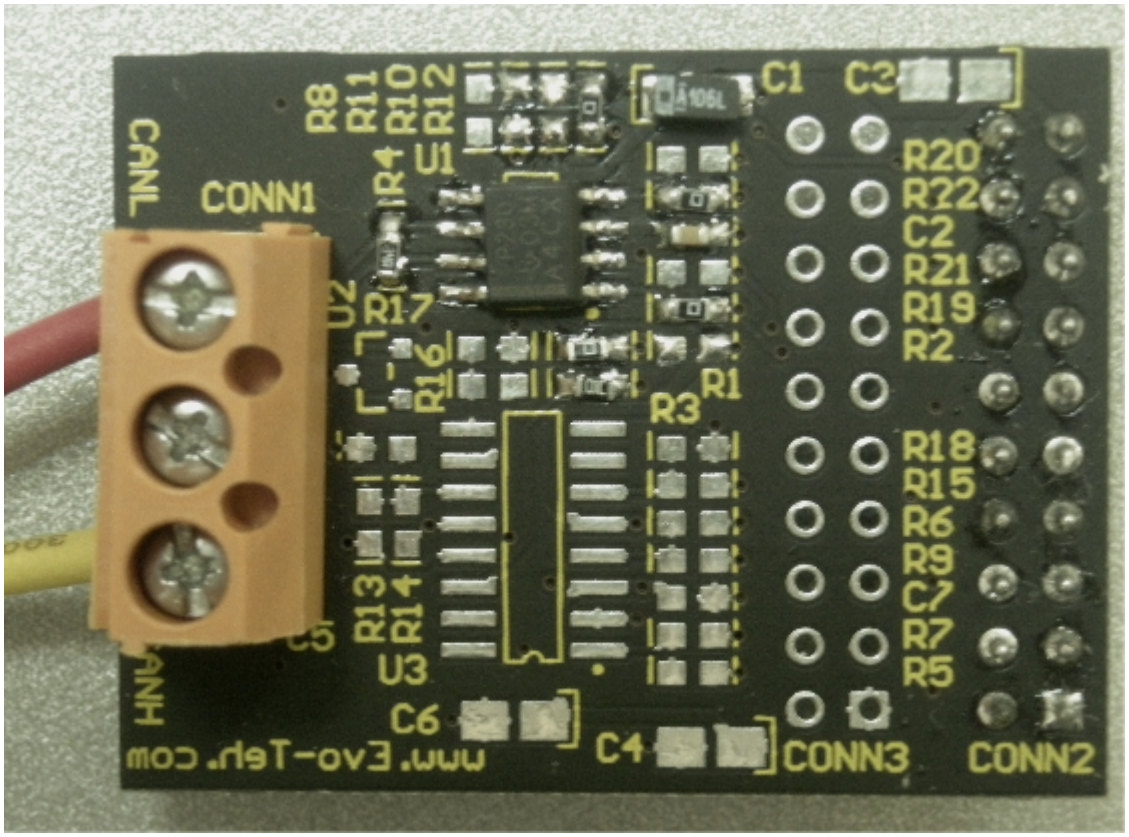
Slika 30: Blokovna shema krmilnika CAN v sklopu mikrokrmilnika AT91SAM9263-CU



Slika 31: Blokovna shema aplikacije

5.2.1 Čip Texas Instruments SN65HVD230

Za uspešno komunikacijo v omrežjih CAN je poleg krmilnika potrebno priklopiti napravo za preslikavo in pošiljanje signalov fizičnega nivoja. Tukaj je uporabljen čip SN65HVD230, proizvajalca Texas Instruments. Čip omogoča diferencialno povezavo za sprejemanje in oddajanje podatkov v omrežjih CAN, s hitrostjo največ 1MB/s. Čip je po specifikacijah proizvajalca izredno zanesljiv, zahvaljujoč naslednjim funkcijam: zaščita crosswire, zaščita pred izpadom zemlje, prenapetostna zaščita in temperaturna zaščita. Izdelana ploščica s čipom SN65HVD230 je prikazana na Sliki 32.



Slika 32: Ploščica s čipom SN65HVD230

Za delovanje uporablja napajalno napetost med 2V in 7V, brez težav pa prenese napetosti tudi do 25 V. Lastnosti krmilnika CAN:

- deluje z napajanjem 3,3 V,
- zaščita vodila/nožic pred elektrostatično razelektritvijo. Testirano do 16 kV HBM (*ang. human body model*),
- visoka vhodna impedanca omogoča uporabo do 120 vozlišč na enem samem

vodilu,

- vozlišče brez napajanja ne moti delovanja vodila,
- kompatibilnost z zahtevami standarda ISO 11898,
- nizki tokovi pri načinu delovanja standby/sleep, tipično 370 μA/40 nA,
- zaščita pred pregrevanjem,
- varna odpoved sistema v primeru odprtih sponk,
- vklop in izklop sistema brez motenj (*ang. glitch*).

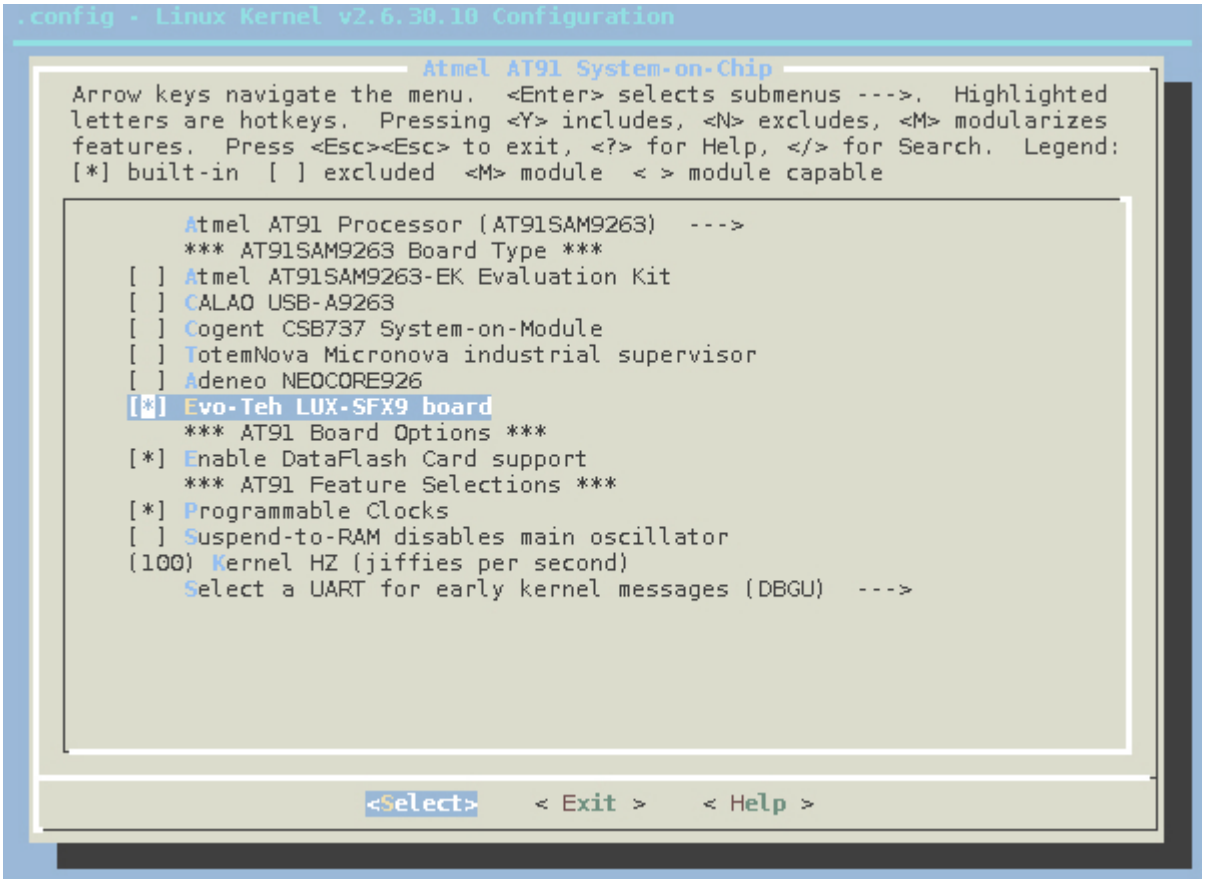
5.3 Programska oprema

Na sistemu teče jedro Linux z distribucijo Ångström. Za nalaganje operacijskega sistema skrbi tristopenjski nalagalnik. Komunikacijo računalnik-ploščica LUX je možno vzpostaviti na več načinov. Najpreprostejša je serijska komunikacija preko priključka USB z uporabo terminalskega emulatorja minicom. Oddaljeni dostop do ploščice LUX se lahko naredi preko lupine ssh ali NFS.

5.3.1 Nastavitve in prevajanje jedra Linux

Jedro Linux je najprej treba ustrezno konfigurirati in prevesti. Šele potem ga je možno shraniti v enem od pomnilnikov na ploščici LUX, ali pa prenesti v pomnilnik SDRAM in zagnati. Za konfiguracijo jedra in izbiro želenih modulov je zelo uporaben program ncurses, ki omogoča grafično in pregledno nastavljanje parametrov jedra Linux.

Po konfiguraciji se jedro prevede in je pripravljeno za prenos na ploščico. Eden izmed načinov prenašanja jedra iz razvojne postaje na ploščico je preko omrežja z uporabo programa tftp. Takrat se jedro naloži v pomnilnik SDRAM in s pomočjo ukaza bootm se začne nalaganje in zagon jedra. Postopek nalaganja je opisan v naslednjem odseku. Sliki 33 in 34 prikazujeta del konfiguracije jedra in potrebnih modulov.



Slika 33: Konfiguracija operacijskega sistema pred prevajanjem

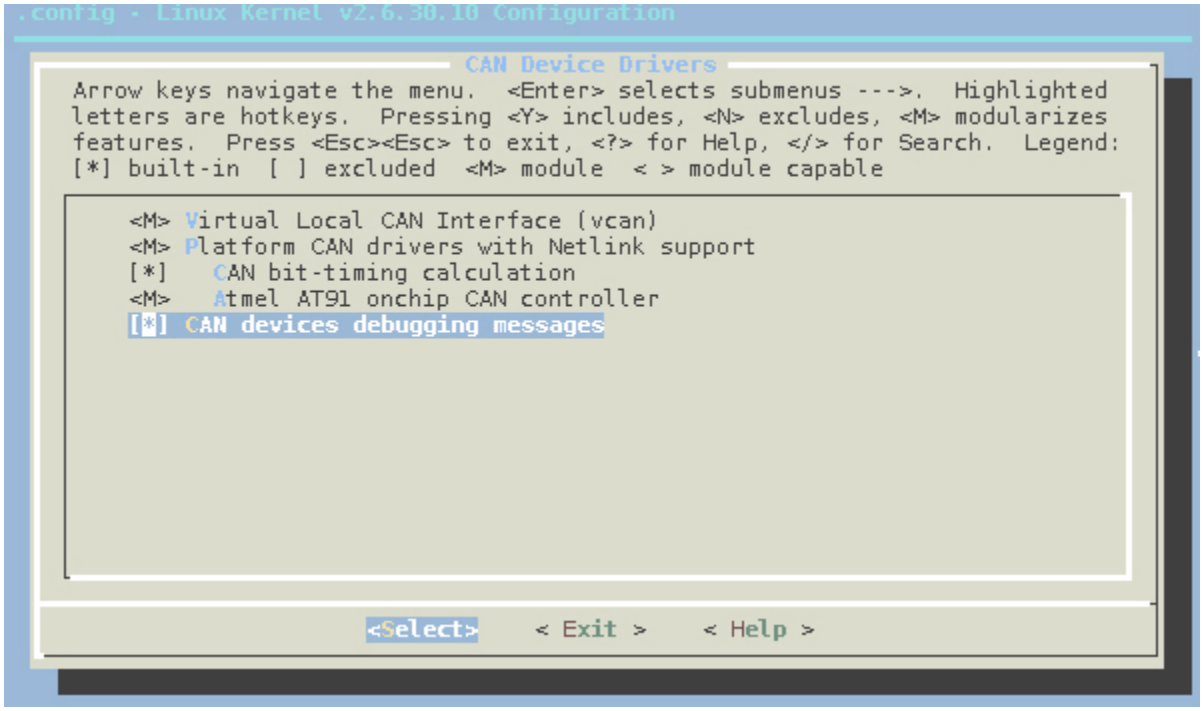
5.3.1 Boot, uBoot, Linux, Ångström

Na razvojni ploščici teče jedro Linux, verzija 2.6.30.10. Uporabljena distribucija Ångström je primerna za vgrajene sisteme. Po besedah ustvarjalcev je distribucija vsestranska v pravem smislu in bo mogoče nekoč tekla celo na zelo preprosti strojni opremi, kot je opekač. Preden pa začne teči operacijski sistem, ga je potrebno naložiti.

Mikrokrmilnik AT91SAM9263 uporablja t.i. tristopenjsko nalaganje. V prvi stopnji se zagonski nalagalnik samodejno naloži iz pomnilnika ROM v statični pomnilnik SRAM. Nalagalnik NVMboot poskrbi za inicializacijo kristala in potem se odvisno od stanja vhoda BMS zgodi naslednje:

- če je BMS=0, zažene kodo iz pomnilnika NOR na zunanjem vodilu EBI0,
- če je BMS=1, gre naprej.

Prva lokacija s katere skuša naložiti nalagalnik, je kartica SD na vodilu MCI1. V primeru prisotnosti nalagalnika AT91Bootstrap ga naloži, sicer pogleda prvi sektor v pomnilniku NAND. Če tudi tukaj ne najde nalagalnika AT91Bootstrap, pogleda v serijski pomnilnik DataFlash in ga zažene. V primeru odsotnosti nalagalnika AT91Bootstrap v serijskem pomnilniku zagonski nalagalnik NVMboot čaka na nalaganje preko programa SAM-BA, ki teče na osebnem računalniku in se poveže z mikrokontrolerjem preko povezave USB.



Slika 34: Izbira želenih modulov, v tem primeru moduli CAN

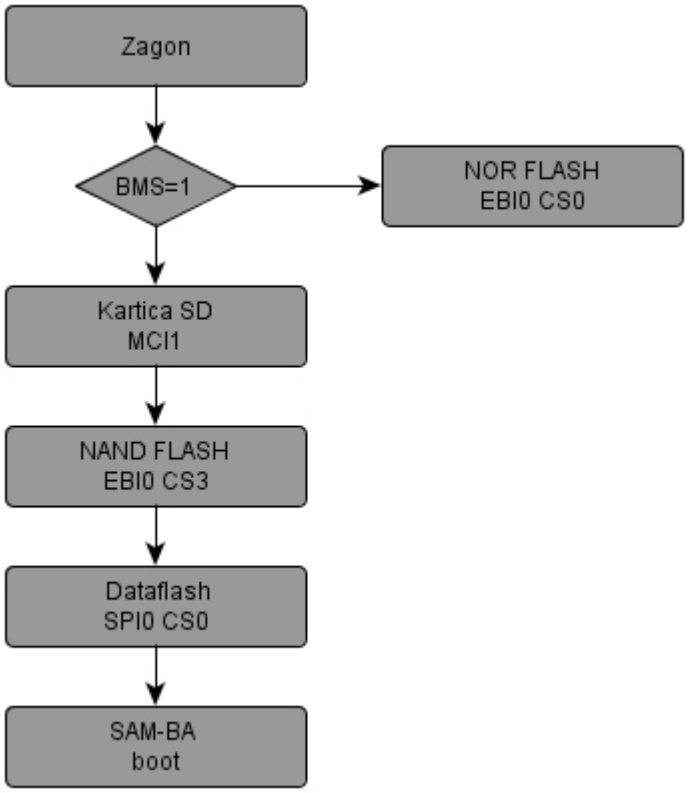
Po uspešnem kopiranju nalagalnika druge stopnje AT91Bootstrap v pomnilnik SRAM se ta zažene. Nalagalnik AT91Bootstrap poskrbi za inicializacijo strojne opreme. Tukaj poteka nastavitev splošnih V/I nožic, ure in ostalih zunanjih komponent, na primer pomnilnika SDRAM. Takoj ko je nastavljen glavni pomnilnik, zagonski nalagalnik prenese prvo aplikacijo iz izbranega pomnilnika FLASH v glavni pomnilnik ter jo zažene. V našem primeru je omenjena aplikacija nalagalnik uBoot iz pomnilnika DataFlash. Slika 35 opisuje diagram poteka nalaganje aplikacije iz različnih medijev.

Delo nalagalnika uBoot je naslednje:

- inicializacija ostale periferije,
- priprava procesorja za nalaganje jedra Linux.

Možnosti za nalaganje jedra je več. Jedro se lahko nahaja v pomnilniku DataFlash,

NAND, na kartici SD, celo na oddaljenem računalniku in se s pomočjo programa TFTP brez posebnih težav naloži. Jedro se naloži v glavni pomnilnik SDRAM na naslovu 0x20000000 ter se zažene. Od tega trenutka naprej ima glavno vlogo jedro Linux, ki hrani datotečni sistem v pomnilniku FLASH, program U-boot pa se umakne.



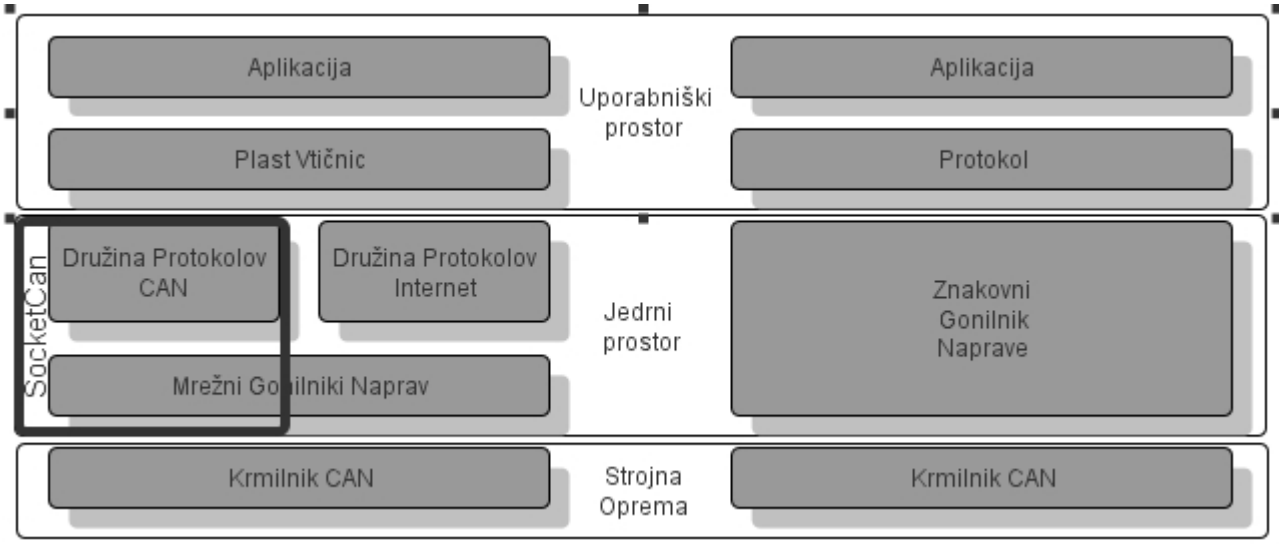
Slika 35: Zaporedje nalaganja

Do sedaj je uspešno nastavljen sistem, na katerem teče jedro Linux, vmesnik za komunikacijo v omrežjih CAN in ogrodje za implementacijo protokola CANopen. Potrebno je še izbrati ustrezni način, kako povezati vse sestavne komponente skupaj. Kako povezati vmesnik CAN z jedrom Linux tako, da je uporaba ogrodja CanFestival možna? V prejšnjem poglavju so v tabeli 8 prikazani vsi gonilniki, ki jih podpira ogrodje CanFestival, odgovor na vprašanje, zakaj je bil izbran gonilnik SocketCan, se nahaja v naslednjem odseku. Pomembno je še povedati, da je SocketCan trenutno privzeti gonilnik v jedru Linux.

5.3.1 Družina gonilnikov SocketCan

SocketCan predstavlja množico odprtokodnih gonilnikov za protokol CAN. SocketCan je uradni gonilnik za naprave CAN na operacijskem sistemu Linux. Uporabljajoč berkeleyske vtičnice (*ang. Berkeley sockets*) in Linuxov omrežni sklad ustvarja omrežni vmesnik do vsake naprave CAN. Za razliko od drugih gonilnikov za Linux, ki so tipično zankovni gonilniki, SocketCan predstavlja omrežni gonilnik. Razliko med družino gonilnikov SocketCan in ostalimi tipičnimi gonilniki prikazuje slika 36. Glavni cilj gonilnika SocketCan je ponuditi vmesnik za vtičnice v okviru omrežne plasti jedra Linux. V omrežjih CAN vozlišča uporabljajo način pošiljanja multicast in nimajo naslovov, kot je to narejeno v omrežjih Ethernet. Identifikacijska številka sporočila CAN za arbitražo je edino sredstvo za ugotavljanje naslova vozlišča, če se previdno uporablja.

V primeru omrežnega gonilnika dostopa več aplikacij do istih sporočil. Jedro gonilnika SocketCan ponuja več seznamov, ki omogočajo transparente dostope aplikacij do želenih podatkov. Naročanje in preklic sprejemanja določenih identifikacijskih številk sporočila CAN poteka preko modula CAN RAW. Uporabniška aplikacija ustvari novo vtičnico, modul RAW pa samodejno zahteva razpon identifikacijski številk od jedra CAN, ki ustreza uporabniški zahtevi. Za optimizacijo izvajalnega časa so seznami za sprejemanje s filtri ustrezno razdeljeni.



Slika 36: Družina gonilnikov CAN na levi strani v primerjavi s tipičnimi gonilniki CAN

Zahteva transparentnosti izmenjave podatkov ne glede na lokacije aplikacije (na enem ali na več različnih računalnikih) je izpolnjena s pomočjo implementacije povratne zanke (*ang. loopback*). Po privzeti nastavitvi je povratna zanka vklopljena, obstaja pa tudi možnost izklopa. SocketCan ponuja mehanizme za sprejemanje lokalnih okvire, upoštevajoč identifikacijske številke CAN, kar pomeni da ni pomembno kje se nahaja aplikacija.

SocketCan omogoča uporabo t.i. okvirov napake, ki se lahko uporabljajo za diagnostiko strojne opreme. Kadarkoli gonilnik CAN zazna napako na fizični plasti ali na plasti MAC, ustvari ustrezno sporočilo napake, ki ga potem posreduje uporabniški aplikaciji. Aplikacija se na drugi

strani lahko odloči, ali je napaka zanimiva zanjo ali ne. Sprejemanje okvirov napake je onemogočeno po privzeti nastavitvi.

Prednosti družine gonilnikov SocketCan je več in vse so posledice dejstva, da je SocketCan omrežni gonilnik in ne znakovni. To pomeni, da lahko do naprave dostopa več procesov, kar v primeru znakovnega gonilnika ni možno. Gonilnik naprave se registrira kot mrežni gonilnik, kar pomeni, da se okviri CAN posredujejo med gonilnikom in mrežno plastjo jedra Linux.

Poleg modula CAN CORE je potrebno naložiti vsaj še en modul protokola. Nalaganje modulov protokola poteka dinamično in naloženih je lahko več protokolov hkrati. Možno je odpiranje več vtičnic, ki sodelujejo bodisi v okviru enega protokola ali pa z več različnimi. Aplikacija se pri odpiranju vtičnic zgolj odloči, kateri protokol bo uporabljala (npr. ISO-TP), potem pa brez težav sodeluje v komunikaciji, ne da bi se ukvarjala z okviri, identifikacijskimi številkami ali podrobnosti protokola CAN. Sledi povzetek prednosti družine protokolov CAN (SocketCan):

- **lažja uporaba:** namesto zapletenih sistemskih ukazov (ioctl) se razvijalec ukvarja z bolj prijaznimi, navadnimi ukazi vtičnic,
- **čistejša koda:** v primerjavi z znakovnimi gonilniki ni podvojevanja kode, ker gonilniki uporabljajo že obstoječo kodo jedra Linux za upravljanje s čakalnimi vrstami,
- **abstrakcija:** abstrakcijska plast ponuja aplikaciji poenoten vmesnik do naprav na eni strani in vmesnik za točno določene gonilnike za strojno opremo na drugi strani.

Slabosti družine protokolov CAN ni veliko. Mogoče je največja slabost generična narava samega gonilnika, ki onemogoča uporabo strojnih mehanizmov krmilnikov CAN. To je obenem tudi prednost, saj omogoča uporabo velikega spektra različnih naprav. Uporaba vseh specifičnih funkcij krmilnika CAN v okviru AT91SAM9263-CU je torej onemogočena. Nabiralniki so razdeljeni v dveh skupinah, za pošiljanje in sprejemanje sporočil. SocketCan ne uporablja strojnih filtrov, saj ima uporaba enega filtra vpliv na celotni vmesnik CAN. Če bi ena od aplikacij postavila svoje strojne filtre, bi se posledice čutile tudi pri drugih aplikacijah in uporabnikih (v večuporabniških sistemih).

5.3.1 Uporaba družino gonilnikov SocketCan

Na sistemih z operacijskim sistemom Linux se družino protokolov CAN omogoči z nalaganjem modulov. Potrebno je naložiti jedro CAN in vsaj še en transportni protokol (can, vcan itd). Vsi uporabljeni ukazi v nadaljevanju se kličejo iz lupine operacijskega sistema Linux, za večino pa potrebujemo ustrezne uporabniške pravice. Nalaganje poteka z naslednjim ukazom:

modprobe can
modprobe vcan .

Sproščanje naloženih modulov se naredi z:

modeprobe -r can .

Naslednji ukaz izpiše naložene module:

lsmod .

Po nalaganju modula, se lahko ustvari poljubno število vmesnikov. Na voljo je tudi navidezni vmesnik CAN, ki za delovanje ne zahteva fizične naprave CAN. Ustvarjanje in brisanje vmesnikov CAN poteka preko vmesnika netlink:

ip link add type canX
ip link add type vcanX
ip link add dev can23 type can
ip link del can23 .

Prikaz vseh omrežnih vmesnikov:

ip link show .

Postavljanje bitne hitrosti:

ip link set canX type can bitrate 125000 .

Branje lastnosti in statistike določenega vmesnika:

ip -details -statistics link show canX .

5.3.1 Pripomočki canutils v sklopu družine gonilnikov SocketCan

SocketCan ponuja pripomočke, ki precej olajšajo nadzor omrežja CAN. Obenem so omenjeni pripomočki zaradi njihove odprte narave popoln primer uporabe družine gonilnikov SocketCan za izdelavo nižje-nivojskih aplikacij.

Omenili bomo le bolj koristne pripomočke. Pred začetkom uporabe pripomočkov je potrebno postaviti celotno omrežje CAN, da bi se lahko začelo s testiranjem. V prejšnjem poglavju je bilo povedano, da SocketCan omogoča ustvarjanje t. i. navideznih naprav CAN. Število navideznih naprav je praktično neomejeno, saj je zmogljivost sistema edina omejitev. Izkušnje, pridobljene z izdelavo tega diplomskega dela, kažejo, da je zadosten sistem za testiranje simulacija dveh naprav z vmesnikom CAN. Z drugimi besedami – dovolj je ustvariti dve navidezni naprave, ki komunicirata med sabo. Naslednji sklop ukazov ustvari dve napravi z imenoma vcan1 in vcan2:

ip link add type vcan

ip link add type vcan

ip link set vcan0 up

ip link set vcan1 up .

Da bi bili zgornji ukazi uspešno izvedeni, je potrebno poskrbeti za nalaganje ustreznih modulov (glej prejšnje poglavje). Za ustvarjanje prometa na želeni napravi obstaja več orodij, najbolj uporabna pa sta po mnenju avtorja **cansend** in **cangen**. **Cansend** pošlje določeni napravi eno sporočilo. Uporaba je prikazana s spodnjo vrstico:

./cansend vcan0 5AA# .

Format sporočila je ID#DATA in je zelo fleksibilen. Dolžina identifikacijske številke je 3 ali 8 heksadecimalnih znakov, ki se lahko ločijo s pikami. Za oddaljeno sporočilo je format naslednji: ID#R.

Pripomoček cansend je dovolj za začetna testiranja, vendar za testiranje v realnem času ni primeren. Razvijalci družine gonilnikov SocketCan so rešili tudi to zahtevo s pripomočkom **cangen**. Pripomoček se enkrat zažene in glede na vhodne argumente nenehno ustvarja sporočila na izbrani napravi. Na voljo so navadne in razširjene identifikacijske številke, sporočila napake, izbira časa med zaporednimi sporočili, povečevanje vrednosti podatkov z vsakim naslednjim sporočilom, naključni podatki v sporočilih in še več. Tipični zagon pripomočka cangen zgleda takole:

./cangen vcan0 4 -I 5FF -L 1 -D i -v –v .

Za nadzor prometa na napravah se uporabljata dva pripomočka, **candump** in **cansniffer**. Uporaba pripomočka **cansniffer** je preprosta, saj kot parameter sprejema ime naprave in še nekatera dodatna stikala. Med njimi je stikalo f, ki postavlja filter sporočil glede na identifikacijske številke CAN-ID. Spodnji ukaz posluša na napravi vcan0 in izpiše vsako sporočilo:

./cansniffer *vcan0 -f 000* .

Cansniffer omogoča uporabo ukazov v času izvajanja, radovedni bralec pa dobi vse informacije o pripomočku cansniffer s klicanjem pripomočka brez dodatnih parametrov.

Pripomoček **candump** je zmogljivejši v primerjavi s cansniffer in ponuja dodatne funkcionalnosti. Z njim lahko se promet izpisuje v določeno datoteko, promet iz več naprav se usmeri na zaslon ali v datoteko, celo iz ene naprave na drugo. Prikaz uporabe pripomočka candump:

./candump [možnosti] naprava .

Na koncu sledi ukaz, ki omogoča postavitev sistema, s katerim se pokaže, da omrežni sklad deluje pravilno preko povezave dveh navideznih naprav. Pripomoček candump ustvari most med obema napravama in pošlje vsa sporočila iz vcan0 na vcan1 z naslednjim ukazom:

./candump vcan0 -b vcan1 .

Diskriminacija določenih identifikacij sporočila, ki se pojavijo pri prvi napravi, usmeri na drugo napravo:

./candump can0,400:700 -b vcan1 .

5.3.1 Uporaba ogrodja CanFestival

Na začetku zahteva uporaba ogrodja CanFestival od razvijalca precej truda, vendar primeri implementacije, ki so vključeni v kodi, občutno olajšajo delo. Izbrana metoda implementacije sledi naslednjim principom:

- Čim večja uporaba že obstoječe kode (tukaj so mišljeni že obstoječi primeri),
- spremembe obstoječe kode namesto pisanja nove,
- slog programiranja, ki je čim bližji slogu ogrodja CanFestival.

Razvijalec zastavi želene funkcije vozlišča in jih implementira v slovarju objektov s pomočjo grafičnega vmesnika ter na koncu priredi ustrezni primer iz priložene kode.

5.3.1 Uporaba primerov v sklopu ogrodja CanFestival

Na voljo so primeri za različne platforme in različne tipe vozlišč. Od gospodarjev za mikorokrmilnike Atmel do sužnjev za bolj sofisticirane sisteme z operacijskim sistemom. Za izvedbo praktičnega dela te diplomske naloge je koda vozlišča gospodar za operacijski sistem Linux ustrezno adaptirana. Zastavljen je bil naslednji cilj: zamenjati gospodarja CANopen omrežja, ki teče na industrijskem programabilnem krmilniku Unitronics V570 s ploščico LUX, na kateri teče koda gospodarja. Gospodar na krmilniku Unitronics V570 ne uporablja nikakršnega mehanizma varovanja vozlišč (čeprav mora biti po standardu CANopen vsaj eden) in uporablja ekspeditivni način konfiguracije preko servisnih objektov. Zna vklapljati naprave, kadar sporočijo, da so pripravljene za vklop na vodilo, in bere ustrezne procesne objekte v realnem času.

5.4 Izdelava vozlišča CANopen z ogrodjem CanFestival

Za opis izdelave vozlišča CANopen je uporabljena koda, napisana pri izdelavi praktičnega dela diplomske naloge. To poglavje predstavlja lep primer hitre izdelave tipičnega vozlišča CANopen z najbolj osnovnimi funkcionalnostmi.

5.4.1 Skelet kode tipičnega vozlišča

Struktura celotnega vozlišča je naslednja:

- datoteka .od z definicijo slovarja objektov,
- datoteka .c slovar objektov,
- datoteka .c definicija vozlišča,

- datoteka .c s funkcijo main, izvedba vozlišča.

Začne se z ustvarjanjem datoteke s pomočjo grafičnega vmesnika wxWidget. Datoteka predstavlja definicijo izbrane funkcionalnosti slovarja objektov. Iz te datoteke samodejno (pri prevajanju) nastane datoteka .c, ki abstraktne funkcionalnosti prevede v ustrezno kodo v jeziku C. Za boljšo modularnost je v eni datoteki definicija ustreznih funkcij vozlišča, druga datoteka pa vsebuje kodo za inicializacijo vozlišča v okviru glavne aplikacije, kjer se nahaja povezovanje med vozliščem in programskim ogrodjem CanFestival.

Razvijalec torej določi želene lastnosti vozlišča z uporabo grafičnega vmesnika, ki se posledično pretvorijo v kodo C. Lastnosti vozlišča so v bistvu protokoli, ki jih vozlišče pozna, oziroma načini komunikacije v omrežju. Naslednji korak je omogočiti določene funkcionalnosti, značilne za posamezno vozlišče. Skratka, cilj drugega koraka je definicija vozlišča in funkcij, ki se uporabljajo pri določenih protokolih. Zadnji korak je vključevanje zasnovanega vozlišča v aplikacijo. Tukaj razvijalec poskrbi za ustvarjanje vozlišča, povezovanje z mehanizmi ogrodja, zagon vozlišča v novi niti in na koncu uničevanje in sproščanje resursov, ki jih je vozlišče zasedlo.

Uporaba grafičnega vmesnika za ustvarjanje datotek s specifikacijami slovarja objektov kot prvi korak je opisana v prejšnjem poglavju. Sledi podroben opis drugega in tretjega koraka izvedbe vozlišča z ogrodjem CanFestival.

5.4.1 Definicija funkcije za določeno funkcionalnost protokola CANopen

Zaradi boljše preglednosti in lepšega načina programiranja je potrebno definicijo funkcij vozlišča spraviti v posebno datoteko. Kot primer je prikazana implementacija proizvajanja objekta za sinhronizacijo.

```
void testMst_post_sync(CO_Data* d)
{
        AO0=(DO0*7 + DO0*27 + DO0*101)%9999;

        DO0++;

        eprintf("Master:%d: Post Sync!\n",init_step);
        eprintf("Master DO0 :  %2.2x \n",DO0);

        eprintf("Turck: DI0: %2.2x \n",DI0);
```

```
        eprintf("Turck: AI0: %x\n",AI0);
        eprintf("Turck: AO0  %x\n",AO0);
}
```

Funkcija se izvede na intervalu, ki je bil določen v slovarju objektov. Poleg generiranja ustreznega komunikacijskega objekta(COB-ID=0x80) funkcija izpiše vrednosti določenih spremenljivk.

5.4.1 Definicija naprave in povezovanje funkcij v glavni aplikaciji

Vsaka funkcionalnost standarda CANopen, ki je bila izbrana pri definiciji vozlišča, se preslika v funkcijo, ki se potem samodejno izvede ob določenih pogojih skladno s standardom CANopen. Po generiranju kode C iz slovarja objektov se ime strukture, ki predstavlja vmesnik do slovarja objektov, nahaja v zadnji vrstici datoteke .c. Če je bilo izbrano ime za slovar objektov testMst, se koda C nahaja v datoteki testMst.c, struktura slovarja objektov pa je definirana v zadnji vrstici:

```
CO_Data testMst_Data = CANOPEN_NODE_DATA_INITIALIZER(testMst);
```

V glavni aplikaciji je najprej potrebno definirati vozlišče z ukazom s_board():

```
s_BOARD masterBoard={"0","125K"}
```

Nato sledi definicija funkcij, ki vsaka zase predstavljajo most med aplikacijo in skladom CanFestival, oziroma mehanizmi protokola CANopen. Torej če vozlišče implementira funkcije: initialisation (se sproži ob inicializaciji vozlišča), post_sync (generiranje objekta za sinhronizacijo) in post_TPDO (oddajanje procesnega objekta), je potrebno definirati funkcije in potem v glavni aplikaciji povezati te funkcije z ustreznimi funkcijami vozlišča. Primer:

```
testMst_Data.initialisation = testMst_initialisation;
testMst_Data.post_sync = testMst_post_sync;
testMst_Data.post_TPDO = testMst_post_TPDO;
```

Zgornja koda zagotavlja, da se ob določenih pogojih izvede ustrezna funkcija. Za tem sledi odpiranje naprave CAN oziroma povezava med napravo in slovarjem objektov:

```
if(!canOpen(&masterBoard,&testMst_Data)){
   eprintf("Cannot open Master Board\n");
   goto fail_master;
  }
```

V primeru uspešne povezave med napravo in slovarjem sledi zagon naprave. Naprava se bo, ob pravilni definiciji funkcije uspešno pogovarjala skladno s protokolom CANopen.

```
StartTimerLoop(&InitNodes);
```

Ob izklopu naprave iz vodila, je potrebno zapreti uporabljen časovnik in sporočiti ostalim napravam v omrežju o prenehanju komuniciranja. Slednja zahteva je zlasti pomembna pri pomembnejših vozliščih kot sta na primer gospodar vodila in proizvajalec objekta za sinhronizacijo.

```
StopTimerLoop(&Exit);
```

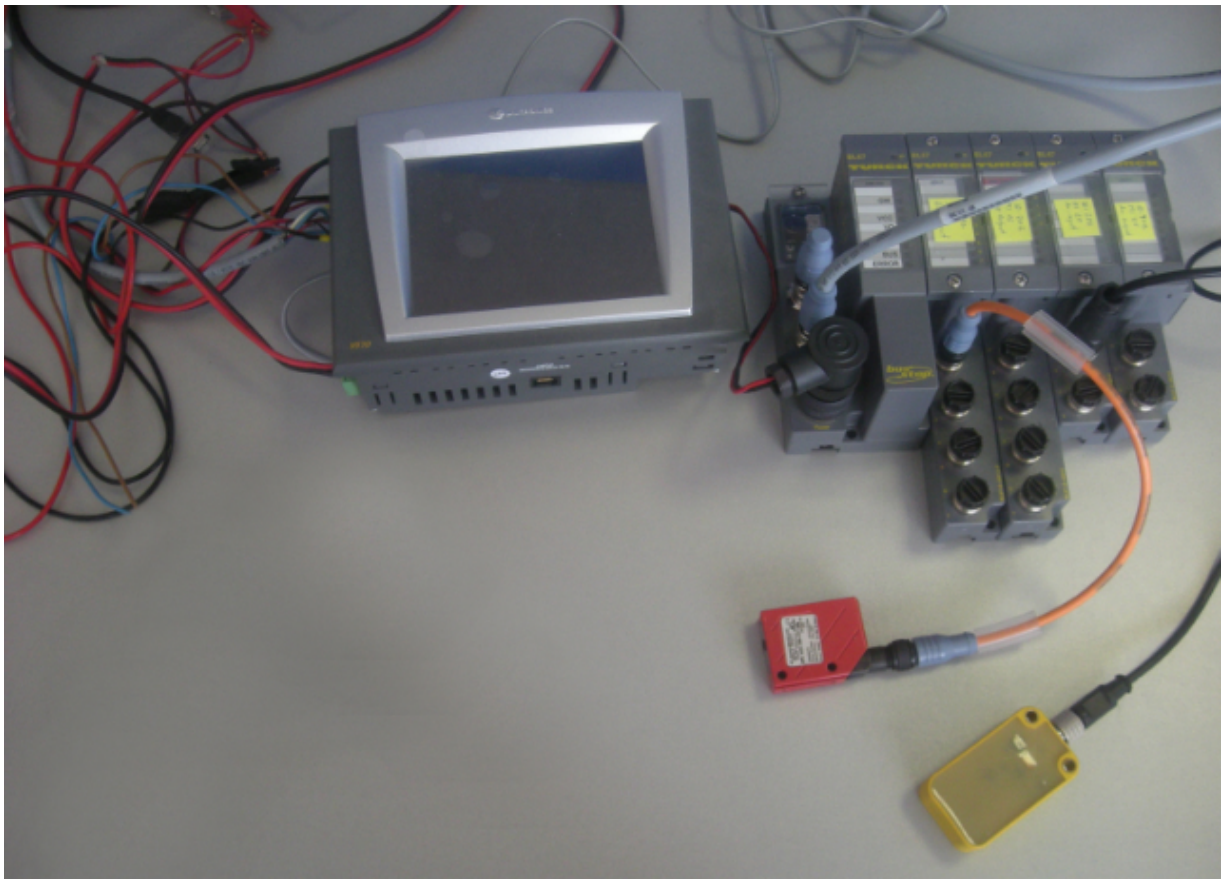
Inicializacija vozlišča (postavljanje identifikacijske številke[17], spreminjanje stanja) se doseže s funkcijo InitNodes. Ob izklopu naprave iz vodila s pomočjo funkcije Exit je doseženo obveščanje in spreminjanje stanja končnega avtomata vozlišča. Preden se aplikacija konča, poskrbi še za časovnike:

```
TimerCleanup();
```

17 Identifikacijska številka vozlišča je različno od identifikacijskih števil komunikacijskih objektov COB-ID. COB-ID predstavlja vsota identifikacijske številke “splošnega” komunikacijskega objekta in identifikacijske številke vozlišča.

6 Primer uporabe

Končna aplikacija predstavlja implementacijo gospodarja vozlišča, ki upravlja z omrežjem CAN z uporabo protokola CANopen. Vozlišče, ki teče na ploščici LUX, nadomešča programabilni logični krmilnik V570 podjetja Unitronics, ki je bil prvotni gospodar[18] omrežja.



Slika 37: Omrežje CAN z vozlišči CANopen. Na levi krmilnik Unitronics V570, na desni Türck BL67.

18 Previden bralec se ne bi strinjal z zgornjim stavkom. Namreč gospodar lahko vsako vozlišče, ki uporablja ta način komunikacije, zato je v enem samem omrežju lahko več vozlišč gospodarjev. V tem kontekstu z besedo gospodar je mišljeno na vozlišče, ki skrbi za konfiguracijo in združuje več funkcionalnosti CANopen za upravljanje z omrežjem.

Poleg ostalih vmesnikov krmilnik vsebuje tudi vmesnik CANopen, preko katerega je povezan z V/I enoto, na katero je možno priklopiti digitalne in analogne senzorje ter aktuatorje. Krmilnik preko V/I enote komunicira z analognim kapacitivnim senzorjem za merjenje razdalje ter z digitalno fotocelico, ki zaznava prisotnost objektov. Omrežje je navadno omrežje CAN, nad katerim teče protokol CANopen. Ob zagonu sistema je krmilnik V570 nastavljen tako, da poskrbi za konfiguracijo V/I enote, šele potem pa se začne komunikacija v realnem času. Prvotni sistem s krmilnikom V570 je prikazan na sliki 37.

6.1 Modularna V/I enota Türck BL67

Türckova V/I enota na sliki 39 je vmesnik med senzorji in aktuatorji na eni strani in napravami CANopen na drugi. Visoka raven modularnosti omogoča uporabo različnih mrežnih vmesnikov(ProfiNet, DeviceNet, CANopen itd) ter različnih tipov V/I naprav, analognih ali digitalnih. V/I enota je sestavljena iz dveh delov: komunikacijskega vmesnika in V/I vmesnika. V našem primeru, je komunikacijski vmesnik tipa CANopen, za V/I sta uporabljena dva vhoda – en analogni, drugi pa digitalni. Komunikacijski vmesnik predstavlja prehod(*ang. gateway*) med omrežjem in V/I napravami. Module je možno poljubno odstranjevati in dodajati do največ 32. V našem primeru so bili uporabljeni štirje moduli, od katerih sta dva analogna dva digitalna, dva vhoda oziroma izhoda.

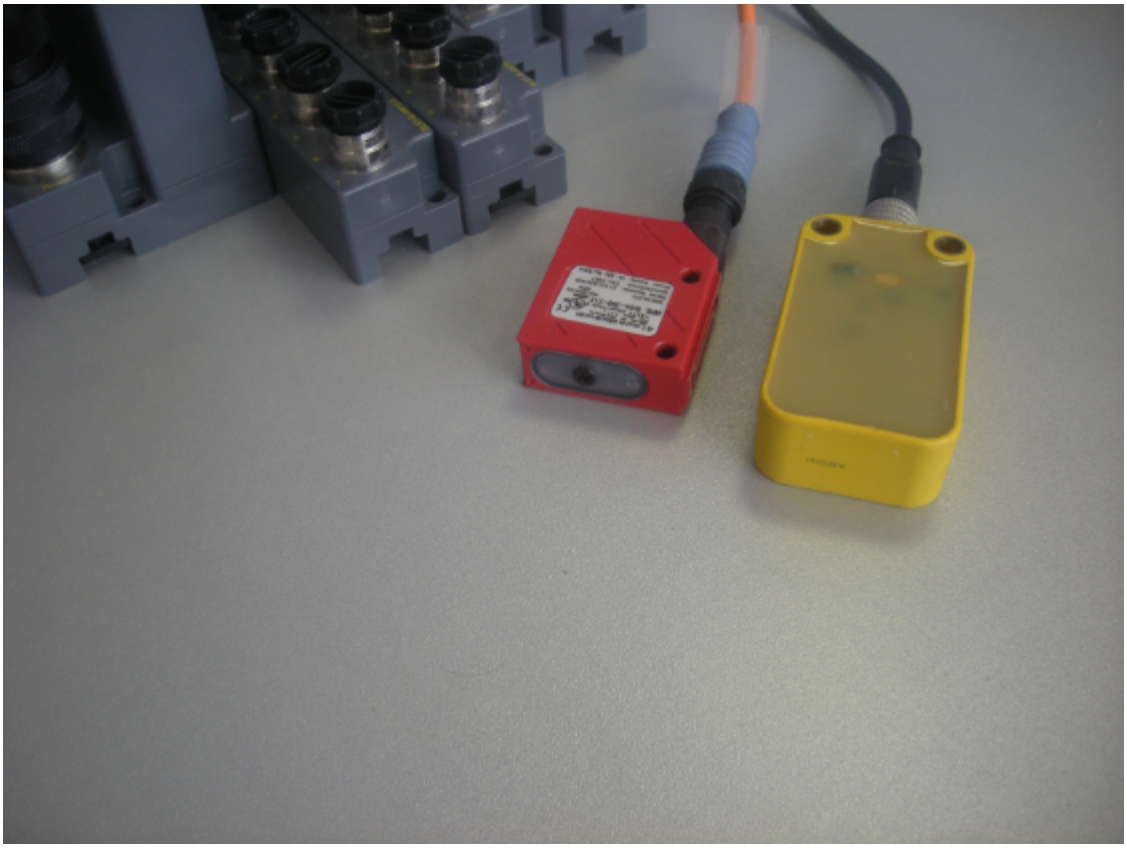
V/I enota ima odstranljiv terminator 120Ω za omrežje CAN, poleg tega ima tudi rotacijsko stikalo, s katerim se izbira bitna hitrost, ter dva enkoderja, s katerima se določi naslov oziroma identifikacijska številka naprave. Razpon identifikacijske številke je od 0 do 63, v našem primeru je identifikacijska številka V/I enote 0x02.

Slika 38: Distribuirana modularna V/I enota Türck z vmesnikom CANopen

6.2 Analogni induktivni senzor Bi15-Q20-LI2-H1141

Analogni induktivni senzor za merjenje razdalje Bi15-Q20-LI2-H1141 proizvajalca Türck ima tokovni izhod, ki je povezan na analogni vhod V/I enote. Slednja uporablja procesni objekt PDO3, ki zasede komunikacijsko identifikacijsko številko COB-ID=280[19] in uporablja 32-bitni podatkovni del. Gospodar je nastavljen za sprejemanje procesnega podatkovnega objekta z ustrezno komunikacijsko številko. Vrednost 32-bitnega števila se izpiše ob generiranju komunikacijskega objekta za sinhronizacijo. Predstavljen je na sliki 39 skupaj s fotocelico.

19 Tej vrednosti se doda še identifikacijska številka naprave, v tem primeru ima V/I enota ID=2, kar pomeni COB-ID=282



Slika 39: Senzorja za zaznavanje objektov, senzor rumene barve je analogni induktivni senzor Bi15-Q20-LI2-H1141, senzor rdeče barve pa digitalna fotocelica HRTL8/24-350-S12

6.3 Digitalna fotocelica HRTL 8/24-350-S12

Fotocelica je sestavljena iz oddajnika (laserja rdeče svetlobe) in sprejemnika. Po specifikacijah je domet fotocelice med 5 in 400 milimetrov. Na ohišju se nahaja majhna dioda rumene barve, ki opozarja na detekcijo objekta. Fotocelica je povezana s V/I enoto preko enega izmed digitalnih senzorskih vhodov. Komunikacija poteka preko PDO1 s komunikacijsko identifikacijsko številko COB-ID=180[20] in uporablja 8-bitni podatkovni del. Tudi tukaj se vrednost izpiše ob generiranju komunikacijskega objekta za sinhronizacijo.

20 Podobno kot pri prejšnjem primeru, COB-ID=180+nodeID=182

6.4 Povezovanje ploščice LUX z V/I enoto preko protokola CANopen

Kot je prikazano na sliki 37, je v prvotnem sistemu gospodar omrežja vozlišče, ki ga predstavlja krmilnik V570. Njegova naloga je naslednja:

- po inicializaciji strojne in programske opreme preiti v stanje OPERATIONAL,
- v sklopu inicializacije konfigurirati vse procesne objekte za sprejemanje podatkov, ki jih uporablja gospodar,
- po ugotovitvi, da so sužnji končali z inicializacijo, je potrebno inicializirati še želene CANopen funkcionalnosti preko servisnih objektov – običajno je to inicializacija procesnih objektov za oddajanje,
- spremeniti stanje vseh sužnjev v OPERATIONAL,
- brati in pisati vrednosti senzorjev in aktuatorjev na V/I enoti s pomočjo procesnih objektov.

Spremljanje dogajanja v omrežju CAN oziroma njegov nadzor omogoča pripomoček cansniffer. Njegovo uporaba je bila razložena v poglavju 5. Pripomoček posluša določene vmesnike in izpisuje vse okvire, ki se pojavljajo v omrežju. Tako je možno natančno določiti, kaj točno se dogaja v prvotnem omrežju, v katerem je gospodar krmilnik V570 in potem natančno posneti njegovo obnašanje.

Kot je bilo omenjeno, se najprej inicializira strojna in programska oprema krmilnika. V okviru tega je potrebno inicializirati tudi V/I enoto. Uporablja se ekspeditivni način za konfiguracijo preko servisnih objektov. Pri uporabi senzorjev je potrebno konfigurirati dva procesna objekta za pošiljanje. Čeprav ni bilo nobenih aktuatorjev, sta uporabljena dva procesna objekta za sprejemanje na prvem analognem in prvem digitalnem izhodu. Pozor, omenjena objekta za oddajanje sta na strani gospodarja objekta za sprejemanje. In obratno, objekti, ki so oddajajo s strani gospodarja za V/I enoto, so objekti za sprejemanje.

Skladno s programom krmilnika v570 se V/I enota konfigurira preko servisnih objektov z uporabo ekspeditivnega načina. V/I enota uporablja štiri procesne objekte:

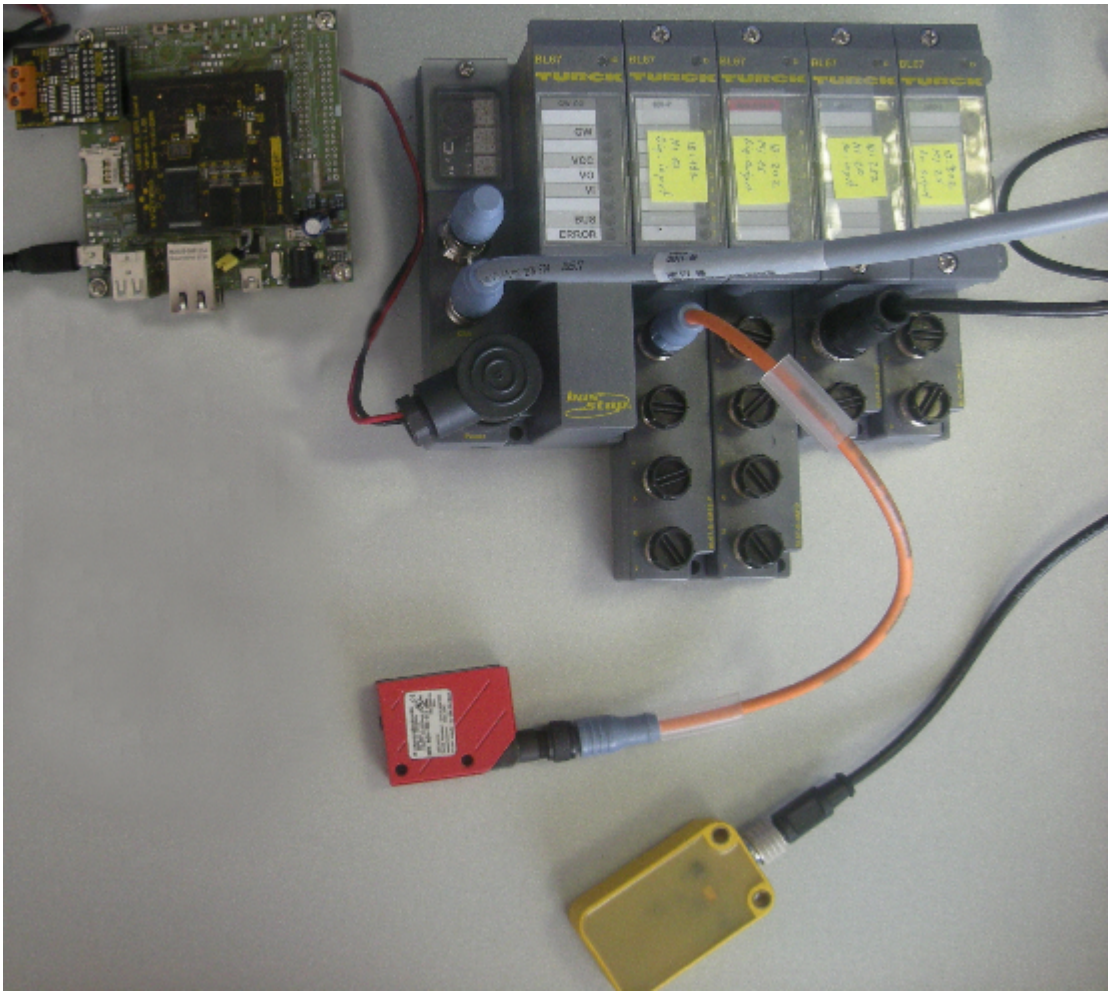
- digitalni izhod, katerega podatki se pošiljajo preko procesnega objekta s COB-ID 0x182.
- analogni izhod procesnega objekta s COB-ID 0x280,
- digitalni vhod procesnega objekta s COB-ID 0x202,
- analogni vhod procesnega objekta s COB-ID 0x302.

Vsi procesni objekti so sinhronizirani s sinhornizacijskim objektom SYNC. To pomeni, da se ob pojavitvi sinhronizacijskega objekta (okvir z identifikacijo 0x80) izvede oddajanje oziroma sprejemanje procesnih objektov.

V prvotnem sistemu so vrednosti senzorjev prikazane na zaslonu krmilnika. V sistemu s ploščico LUX se vrednosti izpisujejo v lupini ob vsaki pojavitvi sinhronizacijskega objekta.

6.5 Končni sistem

Z opisanimi postopki in pristopi je uspešno realizirano načrtovano vozlišče, ki je v celoti nadomestilo vlogo krmilnika V570. Podatki se izmenjujejo v realnem času in se v skladu z intervalom generiranja komunikacijskega objekta za sinhronizacijo izpisujejo v konzoli na razvojni ploščici LUX. Slika 40 prikazuje končni sistem.



Slika 40: Končni sistem

7 Zaključek

Prihodnost protokolov CAN in CANopen[21] je, vsaj kratkoročno, še zelo obetavna. Pojavljajo se vedno novi komunikacijski protokoli, vendar sta protokola CAN in CANopen našla svojo nišo in težko jima bo konkurirati vsaj še naslednjih 20 let. Najresnejši naslednik protokola CAN je protokol FlexRay, najresnejši konkurent protokolu CANopen pa najrazličnejše verzije industrijskega Etherneta.

V današnjem svetu, ko vgrajene sisteme najdemo že skoraj v vsaki napravi, postaja njihov razvoj s poudarkom na medsebojni integraciji in integraciji z ostalimi sistemi vse pomembnejši. Sama strojna oprema, ki predstavlja osnovo vgrajenega sistema, je brez ustrezne programske opreme vredna zelo malo, saj je za končne uporabnike skoraj neuporabna. Dodana vrednost, ki jo prinaša dobra kombinacija strojne in programske opreme, je za uporabnika neprecenljiva.

Končni cilj diplomskega dela je bil na ploščici LUX razviti sistem oziroma vozlišče, ki deluje v skladu s standardom CANopen. V delu je opisan postopek integracije programske opreme in orodij za protokola CAN in CANopen na strojno opremo ploščice LUX. Za potrditev pravilnosti koncepta in nove funkcionalnosti ploščice LUX kot vozlišča CAN smo Unitronicsov programabilni logični krmilnik V570 nadomestili s ploščico LUX. Rezultati testiranj so pokazali, da naš sistem pravilno, brez kakršnih koli težav, komunicira z vsemi ostalimi napravami v omrežju.

Ena od možnosti za nadaljnje delo je razviti popolnoma namenski sistem oziroma vozlišče CANopen, ki bi našlo svojo uporabo v avtomatiki. Toda to ni tako trivialno, saj poleg potrebe po času in znanju ter znatnih finančnih virov vključuje zelo izčrpna testiranja za pridobitev uradne licence CANopen.

Od samega začetka izdelave praktičnega dela sem se srečal z veliko izzivi, ki se jih pred tem nisem zavedal, oziroma sem jih imel za nepomembne. Od samega spajkanja posameznih komponent ploščice, preko namestitve in konfiguracije operacijskega sistema Linux do uporabe popolnoma novega ogrodja je minilo kar nekaj časa. Na koncu se je vse izplačalo, saj sem si pridobil mnoga nova znanja in izkušenj.

21 CAN kot protokol nižjih plasti in CANopen kot protokol višjih sta tesno povezana in zato ju obravnavam kot celoto

8 Literatura

[1] Robert Bosch GmbH. (1991) “CAN Specification“ Version 2, Stuttgart, 1991.

[2] W. Voss, “CANopen-Higher layer protocol based on controller area network(CAN) supports device profiles for I/O modules, Motion control“.

[3] W. Voss, “The Future of CAN/CANopen and the Industrial Ethernet Challenge“.

[4] CAN in Automation, „CiA Draft Standard 301: CANopen Application layer and Communicaiton Profile“ Version 40.2, 2002.

[5] CAN in Automation, “CiA Draft Standard 401: Device profile for generic I/O modules“ Version 3.0, 2008.

[6] “The CanFestival CANopen stack manual“, dostopno na : www.canfestival.org

[7] F. Dupin, “CANopen Memento“ Version 1.5, 2009

[8] U. Hagström, “CANopen“, 2009, Dostopno na: www.datalink.se/index6.php

[10] www.canopen.us

[11]www.can-cia.org

[12]www.canfestival.org

[13]www.luxboards.com/foswiki