Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jurij Cernatič

Sistem za avtomobilsko diagnostiko

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana 2012

Rezultati diplomskega dela so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00019/2012

Datum: 10.04.2012

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

Kandidat: **JURIJ CERNATIČ**

Naslov: **SISTEM ZA AVTOMOBILSKO DIAGNOSTIKO**
AN AUTOMOTIVE DIAGNOSTIC SCAN TOOL

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Načrtujete vgrajen sistem, ki omogoča avtomobilsko diagnostiko. Pri tem preučite delovanje vodila CAN (Controller Area Network) ter standard OBD (On-Board Diagnostics), ki se uporablja za diagnostiko posamičnih podsistemov v avtomobilih. Načrtovani sistem naj omogoča komunikacijo po vodilu CAN, pošiljanje zahtev OBD in obdelavo parametrov OBD. Za implementacijo vgrajenega sistema uporabite mikrokrmilnik STM32F407VGT6, ki vsebuje 32-bitno procesorsko jedro ARM Cortex-M4F in vmesnik za CAN. Za povezavo na fizični nivo vodila CAN uporabite enega od adapterjev (t.i. CAN Transceiver), ki jih je možno dobiti na trgu. Programsko opremo načrtujte v programskem jeziku C v okolju, ki omogoča razvoj programske opreme za mikroprocesorje ARM Cortex (npr. IAR Embedded Workbench for ARM, Atollic TrueSTUDIO, Keil).

Mentor:

prof. dr. Patricio Bulić

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jurij Cernatič, z vpisno številko **63080061**, sem avtor diplomskega dela z naslovom:

Sistem za avtomobilsko diagnostiko

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Patricia Bulića,
- 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 24. septembra 2012 Podpis avtorja:

Zahvaljujem se izr. prof. dr. Patriciu Buliću za napotke in nasvete pri izdelavi diplomske naloge.

Iskreno se zahvaljujem tudi svojim staršem in starim staršem, ki so me vzpodbujali in podpirali tekom študija.

Staršem, ki so mi s pridnim delom omogočili študij.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Splošno o omrežjih in vodilih **3**

2.1 Skupni dostop do medija in arbitraža 3

2.2 Skalabilnost omrežja 4

2.3 Nadzor nad napakami 5

3 Controller Area Network (CAN) **7**

3.1 Zgodovina 7

3.2 Definicija protokola CAN 8

3.3 Protokol CAN in ISO/OSI referenčni model 9

3.4 Predstavitev logičnih stanj 11

3.5 Promet v omrežju CAN 12

3.6 Kodiranje bitov 15

3.7 Vrivanje bitov 16

3.8 Okviri CAN 16

3.9 Napake v omrežju CAN 22

3.10 Implementacija na fizičnem nivoju 26

4 On-Board Diagnostics (OBD) **33**

4.1 Zgodovina 33

4.2 Značilnosti OBD-II 34
4.3 OBD-II in omrežje CAN 38

5 Orodje za diagnostiko vozil z omrežjem CAN 43

5.1 Motivacija 43
5.2 Strojni del 44
5.3 Programski del 54
5.4 Preizkus delovanja 73

6 Sklep 77

Seznam uporabljenih kratic in simbolov

ABS	Anti-lock Braking System
ACK	Acknowledge
AHB	Advanced High-performance Bus
ALDL	Assembly Line Diagnostic Link
ASCII	American Standard Code for Information Interchange
CAN	Controller Area Network
CARB	California Air Resources Board
CDI	Common rail Diesel Injection
CRC	Cyclic Redundancy Code
CSMA	Carrier Sense Multiple Access
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DMA	Direct Memory Access
DTC	Diagnostic Trouble Code
EPA	Environmental Protection Agency
ESC	Electronic Stability Control
FIFO	First In, First Out
GPIO	General Purpose Input/Output
ISO	International Organization for Standardization
JOBD	Japan OBD
KWP2000	Keyword Protocol 2000

LCD	Liquid Crystal Display
LLC	Logical Link Control
MAC	Medium Access Control
MDI	Medium Dependent Interface
NRZ	Non-Return-to-Zero
OBD	On Board Diagnostics
OSI	Open Systems Interconnection
PID	Parameter ID
PLS	Physical Signalling
PMA	Physical Medium Attachment
PWM	Pulse Width Modulation
RTR	Remote Transmission Request
SAE	Society of Automotive Engineers
SoF	Start of Frame
SOIC	Small Outline Integrated Circuit
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
TDI	Turbo Direct Injection
TT-CAN	Time Triggered CAN
UART	Universal Asynchronous Receiver/Transmitter
USART	Universal Synchronous Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VTi	Variable Valve Lift and Timing injection

Povzetek

Sodobna motorna vozila so opremljena z množico elektronskih krmilnih sistemov, ki so med seboj povezani v omrežja in komunicirajo z uporabo različnih komunikacijskih protokolov. V okviru diplomske naloge smo preučili delovanje najbolj zastopanega komunikacijskega protokola – Controller Area Network (CAN). Preučili smo tudi delovanje standarda On Board Diagnostics (OBD), ki se uporablja pri diagnostiki krmilnih sistemov. Zasnovali in izdelali smo cenovno ugodno orodje, ki s pomočjo pošiljanja zahtev po standardu OBD-II na omrežje CAN omogoča diagnostiko krmilnih sistemov. Orodje sestavljata mikrokrmilnika z arhitekturo ARM oziroma AVR, oddajno/sprejemni modul CAN ter zaslon LCD. Za orodje smo razvili programsko opremo, ki uporabniku omogoča izbiro in prikaz različnih diagnostičnih parametrov na zaslonu LCD, dodatno pa s pomočjo osebnega računalnika omogoča tudi spremljanje prometa v omrežju CAN. Na koncu naloge smo podali predloge za izboljšavo diagnostičnega orodja.

Ključne besede:

Controller Area Network, On Board Diagnostics, avtomobilska diagnostika, mikrokrmilnik, vgrajen sistem

Abstract

Modern motor vehicles are equipped with a variety of electronic control systems connected into networks and communicating using various different communication protocols. For the purpose of this thesis we studied the functioning of the most commonly used protocol, the Controller Area Network (CAN). We also studied the On Board Diagnostics standard (OBD), which is used for diagnostics of these control systems. We have designed and built an affordable tool, which enables diagnostics by sending requests adhering to the OBD-II standard to the CAN network. The tool consists of two different (ARM, AVR) microcontrollers, a CAN transceiver and an LCD display. We have also designed software for the tool, which lets the user choose and display different diagnostic parameters. The software also supports traffic monitoring on the CAN network. In the end we have provided suggestions on how to improve the functioning of the tool.

Keywords:

Controller Area Network, On Board Diagnostics, vehicle diagnostics, microcontroller, embedded system

Poglavje 1

Uvod

Živimo v informacijski dobi, v kateri računalništvo predstavlja del našega vsakdana. Ni naključje, da ga srečamo tudi pri enemu izmed najbolj razširjenih prevoznih sredstev – avtomobilu. Za delovanje slednjega so danes namreč zaslužni prav dovršeni elektronski in računalniški sistemi. Pri današnjih avtomobilih delovanje različnih sistemov omogoča vrsta elektronskih krmilnih enot. Združene skupaj pravzaprav predstavljajo porazdeljen računalniški sistem, povezan v omrežje. Pri delovanju tovrstnih sistemov se pojavljajo tudi okvare, ki jih odkrivamo s pomočjo posebnih diagnostičnih orodij.

V diplomski nalogi si bomo najprej ogledali nekaj splošnih lastnosti omrežij in vodil. To nam bo pomagalo pri kasnejšem razumevanju enega izmed temeljnih komunikacijskih protokolov v avtomobilski industriji – protokola CAN. Slednjemu bomo namenili več pozornosti – najprej bomo spoznali, kako je, zaradi zgodovinskih razlogov, prišlo do njegovega nastanka. Sledil bo podroben opis njegovih značilnosti in delovanja, na koncu pa si bomo ogledali še nekaj različnih implementacij v praksi. Teoretični del bomo zaključili z opisom standarda OBD, ki se uporablja pri diagnostiki elektronskih krmilnih enot v avtomobilih.

V drugem delu diplomske naloge bomo opisali razvoj orodja za diagnostiko vozil z omrežjem CAN. Navedli bomo, zakaj smo se za razvoj sploh odločili, in utemeljili izbiro strojnih komponent, ki orodje sestavljajo.

Preučili bomo lastnosti in zmožnosti izbranih komponent in razložili, kako smo jih med seboj povezali. Nadaljevali bomo z opisom postopka razvoja programske opreme – od izbire razvojnih okolij, preko konfiguracije različnih vhodno/izhodnih naprav, do združitve programskih odsekov v zaključeno celoto. Za konec bomo ocenili razvito diagnostično orodje in podali predloge za njegovo izboljšavo.

Poglavje 2

Splošno o omrežjih in vodilih

Preden si pobližje ogledamo protokol Controller Area Network (v nadaljevanju CAN), se velja seznaniti z nekaj bistvenimi lastnostmi omrežij oziroma natančneje vodil ter s težavami pri načrtovanju le-teh. Kljub raznolikosti omrežij in omrežnih protokolov so težave, ki jih moramo razrešiti, med seboj zelo podobne.

2.1 Skupni dostop do medija in arbitraža

Omrežje običajno sestavlja množica naprav, ki med seboj komunicirajo preko nekega medija. S pojmom "vodilo" običajno opišemo fizično povezavo za prenos podatkov med večimi napravami, pri čemer so vse naprave povezane na isto (in hkrati tudi edino) povezavo. V praksi se vodilo izkaže za zelo dobrodošlo vrsto povezovanja med napravami. Odpravlja namreč slabost povezovanja "vsak z vsakim", kjer ima vsaka naprava v omrežju ločeno fizično povezavo z vsako izmed preostalih naprav v omrežju. Na drugi strani se pomanjkljivost vodila kaže v tem, da lahko v nekem časovnem obdobju podatke oddaja le ena izmed naprav, ki so nanj priključene. V primeru, ko preko vodila več naprav v istem času oddaja podatke, se zgodi trk podatkov in prenos je potrebno ponoviti. Trki podatkov predstavljajo izgubo časa in posledično tudi nižajo teoretično pasovno širino omrežja, zato so nezaželjeni.

Mehanizmu, ki določa, kdaj lahko katera izmed naprav uporablja prenosni medij ter tako prepreči trke, pravimo arbitraža. V sodobnih omrežjih se za ta namen uporablja predvsem protokol CSMA (Carrier Sense Multiple Access), pri katerem pošiljatelj pred pošiljanjem posluša dogajanje na mediju (npr. vodilu) in podatke pošlje le v primeru, da je medij prost. V splošnem obstajata dve podvrsti protokola CSMA [1]:

- CSMA/CD (Carrier Sense Multiple Access with Collision Detection): V primeru istočasnega začetka oddajanja dveh ali večih naprav se ugotovi trk in vse naprave prenehajo z oddajanjem podatkov. Po določenem času, ki je različen za vsako izmed naprav, poskuša vsaka izmed naprav ponovno dostopiti do omrežja.

- CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance): Pri tej podvrsti protokola CSMA skušajo člani omrežja poskrbeti, da do trkov sploh ne pride. Načinov za doseganje tega cilja je več (posebni paketi, dodatni signali ...), pri čemer protokol CAN uporablja t. i. "bitwise contention" [1], ki si ga bomo ogledali nekoliko kasneje.

2.2 Skalabilnost omrežja

Omrežja v praksi uporabljamo v zelo različnih konfiguracijah. Njihove arhitekture in topologije ne prilagajamo le med različnimi aplikacijami, ampak vedno pogosteje tudi znotraj določene aplikacije. Za lažje razumevanje lahko vzamemo kar analogijo s proizvodno linijo, ki navzven morda izgleda ves čas enaka, pa vendar potrebuje občasne spremembe in popravke glede na proizvod, ki se na njej proizvaja. Omrežja, pri katerih sprememba v arhitekturi ali topologiji ne predstavlja večje težave, imenujemo skalabilna ali elastična omrežja. Pod spremembami v omrežju so mišljene predvsem fizične spremembe v povezavah med člani omrežja ter spremembe v številu članov omrežja. Pri vodilih fizičnih sprememb v povezavah med člani omrežja seveda ni, pri spremembah v številu članov omrežja pa ostaja problem z naslavljanjem med napravami. Očitno je, da običajen pristop, pri katerem imamo na

eni strani izvorni naslov, na drugi strani pa ponorni naslov, neugodno vpliva na skalabilnost omrežja. Pri dodajanju nove naprave v omrežje ta lastnost namreč povroča težave pri usmerjanju paketov. Veliko ugodneje je, če je za naslavljanje poskrbljeno kar v vsebini samega paketa, kar implicira dve dejstvi. Prvo je to, da mora vsak paket priti do vseh ostalih članov omrežja (to običajno imenujemo razpošiljanje ali broadcasting), kar je pri vodilu seveda trivialno. Drugo pa je to, da se mora vsak član omrežja posebej odločiti, kateri paketi so zanj pomembni in kateri ne. Za ta namen se uporablja filtriranje, pri katerem se nepomembna sporočila preprosto prezre, pomembna pa uporabi. Več o skalabilnosti omrežij izvemo v [1].

2.3 Nadzor nad napakami

Pri vodilu, kjer so vse naprave v omrežju povezane s skupno fizično povezavo, je nadzor nad napakami zelo pomemben. Če kateri izmed članov omrežja povroča napake med oddajanjem, ali še huje, med sprejemanjem paketov, bi to pravzaprav na dolgi rok povročilo izpad celotnega omrežja in s tem prekinitev vsakršne možne komunikacije med ostalimi (sicer pravilno delujočimi) člani omrežja. V takem primeru želimo člana, ki povzroča napake, izolirati od omrežja, kar lahko naredimo z relativno preprosto strategijo. Vsakemu članu dodelimo po dva ločena števca, pri čemer prvi beleži število napak pri oddajanju paketov, drugi pa število napak pri sprejemanju paketov. Števca se povečujeta v odvisnosti od vrste napake in razmer, v katerih je do napake prišlo. Pri neki meji član preide v stanje, ko preneha z oddajanjem sporočil in le še sprejema pakete od drugi članov omrežja. Če se število napak še vedno veča in doseže novo, višjo mejo, pa član preide v tako imenovano pasivno stanje, pri čemer se odklopi od vodila in s tem prepreči izpad omrežja. Odklop ("bus off") izvede tako, da svoje vhode postavi v visoko impedančno stanje in to pravzaprav za ostale člane deluje, kot da bi se fizično odklopil od vodila [2]. Prednost odklopa z visokim impedančnim stanjem pred dejanskim fizičnim odklopom je v tem, da je spremljanje signalov, ki potujejo po

omrežju, še vedno mogoče. V tem stanju član ostane določeno število časa, med katerim se števci zmanjšujejo in ko dosežejo nek prag, se vhodi ponovno postavijo v običajno stanje, pri katerem lahko sprejema pakete od drugih članov v omrežju.

Poglavje 3

Controller Area Network (CAN)

3.1 Zgodovina

Za lažje razumevanje vloge in pomena protokola CAN si je dobro ogledati zgodovinsko ozadje, ki je pripeljalo do njegove iznajdbe in uporabe.

V zgodnjih osemdesetih letih so se v avtomobilih začeli vedno bolj uveljavljati elektronski sistemi. Če so na začetku komponente delovale ločeno vsaka zase, se je hitro pojavila potreba po komunikaciji v realnem času, predvsem med krmnilniki motorne elektronike in samodejnega menjalnika ter sistemom proti blokiranju koles med zaviranjem (ABS). Število elektronskih komponent je postajalo vedno večje, komunikacija med njimi pa je potekala izključno po načelu vsak z vsakim. V praksi je to predstavljalo problem, saj je to hkrati pomenilo tudi veliko dodatne električne napeljave. Slednja je predstavljala prostorski problem, finančni strošek ter nenazadnje tudi odvečno težo. Poleg tega so se protokoli, ki so bili takrat uporabljeni za komunikacijo med komponentami (v veliki večini je to vlogo opravljal kar univerzalni asinhroni sprejemnik in oddajnik – UART), izkazali za neprimerne. Težavo je predstavljala predvsem slaba robustnost, saj avtomobil predstavlja okolje z veliko

viri elektromagnetnih motenj [3].

Leta 1983 so se zato v nemškem podjetju Robert Bosch odločili zasnovati komunikacijski protokol, ki bo namenjem porazdeljenim sistemom in bo omogočal komunikacijo v realnem času, pri čemer bo dosegel vse zahteve, ki jih potrebuje za delovanje v okolju avtomobila [3]. Na začetku so pri razvoju sodelovali s tehnično univerzo Braunschweig, kasneje pa še z ameriškim podjetjem Intel ter nizozemskim podjetjem Philips SemiConductors. Tekom razvoja se je nato pridružilo še veliko drugih podjetij, na primer Siemens, Motorola, Texas Instruments, Temic, Atmel ...

Spomladi leta 1986 je bil kot plod razvoja na konferenci združenja avtomobilskih inženirjev (SAE) v Detroitu predstavljen protokol CAN. Še istega leta se je začela standardizacija protokola na mednarodni organizaciji za standardizacijo (ISO), leto kasneje pa so bili nared prvi delujoči prototipi elektronskih krmilnikov s podporo protokolu CAN. V dejansko uporabo je protokol CAN stopil leta 1991, prvo vozilo, ki ga je uporabljalo, pa je bil Mercedesov avtomobil razreda S [5]. Opremljen je bil s petimi elektronskimi krmilniki, ki so preko vodila CAN med seboj komunicirali s hitrostjo 500 kbit/s. Od tedaj naprej se je uporaba protokola CAN razširila ne samo med avtomobilsko industrijo, ampak na splošno med celotno industrijo. Čeprav je bil CAN prvotno zasnovan kot komunikacijski protokol avtomobilskih elektronskih sistemov, ga danes najdemo tudi pri železniškem prometu, letalski industriji ter celo pri medicinski tehnologiji [4].

3.2 Definicija protokola CAN

Protokol CAN je bil najprej definiran s strani podjetja Robert Bosch, kasneje pa še standardiziran v dokumentu ISO 11898-1 [6]. Trenutna različica standarda (2.0) [7] je razdeljena na dva dela, A in B:

- Del A definira običajen okvir CAN ("base frame format") z dolžino

oznake okvira 11 bitov.

- Del B definiran razširjen okvir CAN ("extended frame format") z dolžino oznake okvira 29 bitov.

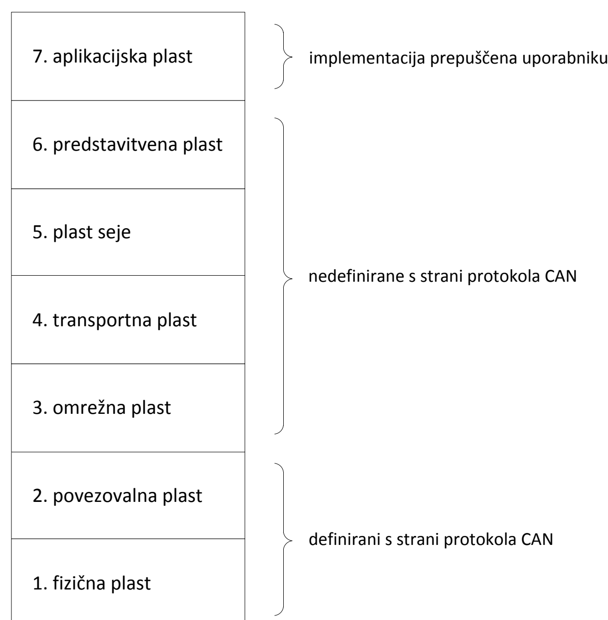
Na splošno se v celotni industriji (vključno z avtomobilsko) v veliki večini uporablja različica 2.0A, zato bomo slednji posvetili več pozornosti. Različica 2.0B se uporablja skoraj izključno v tovornem in transportnem prometu (tovornjaki, avtobusi ...) [8].

3.3 Protokol CAN in ISO/OSI referenčni model

Po običajnem modelu ISO/OSI (International Standardization Organization/Open Systems Interconnection) je vsak omrežni protokol razdeljen na sedem delov, kot lahko vidimo na sliki 3.1. Protokol CAN pokriva v celoti le drugo (povezovalno) plast, v veliki meri pa tudi prvo (fizično) plast. Ostale plasti so nedefinirane, razen sedme (aplikacijske) plasti, katere implementacija je v celoti prepuščena uporabniku. Prvi dve plasti opravljata več nalog, zato si je koristno podrobneje ogledati njuno zgradbo [9].

Plast 1 (fizična plast) ISO/OSI modela:

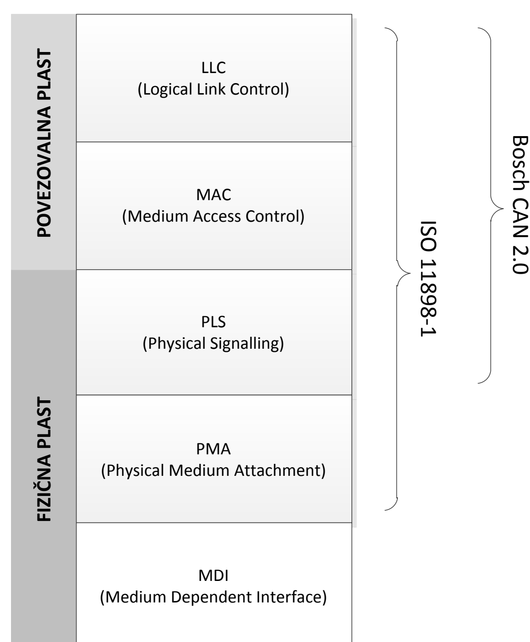
- MDI (Medium Dependent Interface) definira:
  - lastnosti fizičnih povezav,
  - lastnosti priključkov (konektorjev).
- PMA (Physical Medium Attachment) definira:
  - električne lastnosti oddajno/sprejemnih modulov.
- PLS (Physical Signalling) skrbi za frekvenčno predstavitev bitov:
  - kodiranje in dekodiranje,
  - časovno usklajevanje,
  - sinhronizacija.



Slika 3.1: Model ISO/OSI in protokol CAN.

Na tem mestu velja omeniti, da se standard, podan s strani podjetja Bosch, nekoliko razlikuje od standarda ISO 11898-1. Prvi vsebuje namreč le podplast PLS, pri čemer sta ostali dve podplasti popolnoma nedefinirani. Namen tega je omogočiti optimizacijo protokola glede na različna področja uporabe. Razliko lahko na nazorno vidimo na sliki 3.2. Plast 2 (povezovalna plast) ISO/OSI modela:

- MAC (Medium Access Control) predstavlja jedro protokola CAN in skrbi za:
  - okvirjanje sporočil,
  - potrjevanje prejetih sporočil,
  - zaznavanje napak,
  - obveščanje o napakah,
  - arbitražo.



Slika 3.2: Podrobnejša delitev prvih dveh plasti in primerjava standardov Bosch CAN 2.0 ter ISO 11898-1.

- LLC (Logical Link Control) skrbi za:
  - filtriranje sporočil,
  - okrevanje po napakah,
  - obveščanje o preobremenitvah.

3.4 Predstavitev logičnih stanj

Na vodilu CAN je predstavitev logičnih stanj nekoliko neobičajna. Navadno imamo pri digitalnih sistemih opravka z logičnima stanjema, definiranima kot "1" ali "0". Pri vodilu CAN pa logični stanji označujemo kot dominantno oziroma recesivno, posledično imamo zato dominantne in recesivne bite [10]. Pri hkratnem prenosu dominantnega in recesivnega bita vedno prevlada dominanten bit. Razlog za odklon od običajne predstavitve logičnih

stanj tiči v dejstvu, da CAN protokol nikjer ne definira fizičnega prenosnega medija. Običajno vlogo fizičnega prenosnega medija opravlja bakrena parica, pri kateri lahko (ni pa nujno) dominantno stanje opredelimo kot napetost 2 V, recesivno stanje pa kot napetost 0 V. V primeru, da za fizični prenosni medij izberemo optični vodnik, prisotnost svetlobe predstavlja dominantno, odsotnost svetlobe pa recesivno stanje.

3.5 Promet v omrežju CAN

V omrežju CAN nima nobeden izmed članov svojega naslova, kar povroči vrsto pomembnih posledic:

1. Prilagodljivost sistema: dodajanje novih članov oziroma odstranjevanje obstoječih članov omrežja ne zahteva programskih in/ali strojnih sprememb pri ostalih članih omrežja, ravno tako ne zahteva sprememb na aplikacijskem nivoju.

2. Usmerjanje sporočil: vsebina sporočil je označena z oznako okvira, ki opisuje pomen podatkov.

3. Oznaka okvira (identifier): oznaka okvira ne označuje ciljnega naslova sporočila, ampak opisuje vsebino in pomen sporočila. Posledično to pomeni, da mora biti vsak član v omrežju sposoben presoje, ali je sporočilo zanj pomembno ali ne, kar stori z ustreznim filtriranjem.

4. Difuzija podatkov: vsa sporočila, poslana na vodilo CAN, dospejo do vseh članov omrežja hkrati. Ker pa je od nastavitve filtrov posameznega člana odvisno, ali bo sporočilo tudi sprejel, govorimo o prometu tipa multicast.

5. Konsistentnost podatkov: zagotovljena je zaradi t. i. multicast tipa prometa ter s pomočjo mehanizmov za zazanavanje in odpravljanje napak.

6. "Multimaster" delovanje: pri prostem vodilu lahko vsak izmed članov poskuša prevzeti nadzor nad vodilom in začeti prenašati podatke. Član, katerega sporočilo ima najvišjo prioriteto, dobi nadzor nad vodilo in začne s prenosom podatkov.

7. Arbitraža: pri istočasnem dostopu dveh članov omrežja do komunikacijskega medija (v našem primeru vodila) se spor razrešuje s postopkom arbitraže. Spor na vodilu se rešuje z bitno primerjavo oznak okvirov [10]. Bitna primerjava je nedestruktivna vrsta arbitraže, saj član, ki je dobil nadzor nad vodilom, preprosto nadaljuje z oddajanjem vsebine sporočila. Med arbitražnim postopkom (prikazan na sliki 3.3) vsak pošiljatelj preverja logični nivo bita, ki ga pošilja, z logičnim nivojem bita, ki se dejansko pojavi na vodilu. Če naprava v nekem trenutku na vodilo pošlje recesiven bit, v resnici pa se na vodilu pojavi dominanten bit, je to znak, da je izgubila arbitražo in zato takoj preneha z nadaljnim oddajanjem. V primeru, da se logična nivoja ujemata, naprava nadaljuje z oddajanjem.

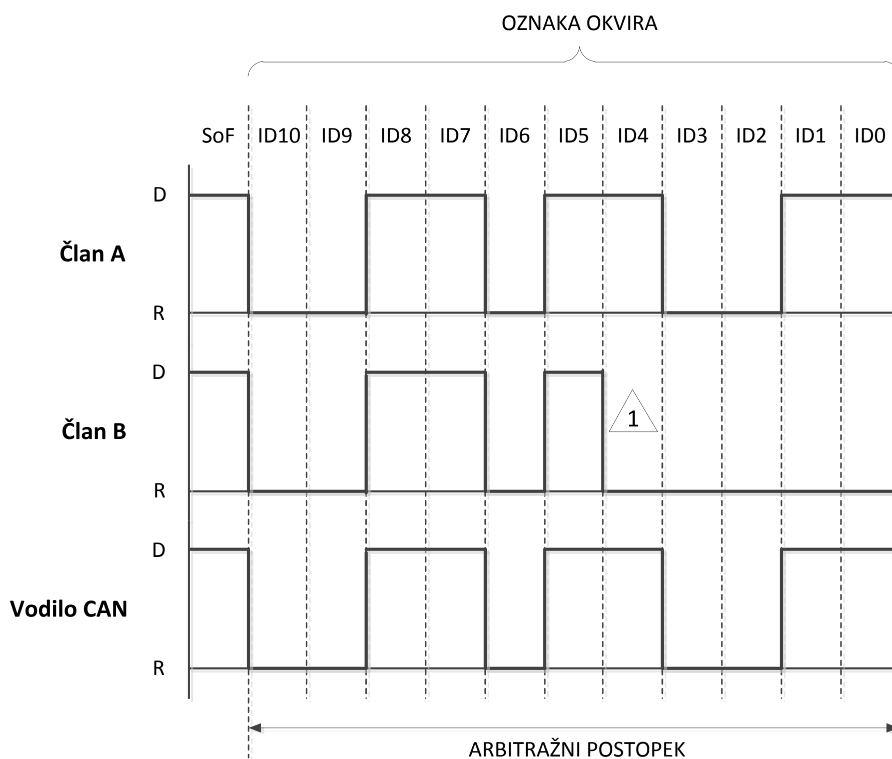
8. Varnost prenosa: protokol CAN definira različne ukrepe za odkrivanje napak, obveščanje o napakah in samokontrolo. Z njimi lahko zaznamo do pet naključno razporejenih napak v posameznem sporočilu. Mednje sodijo:

   - monitoring vodila: pošiljatelj preverja željeni električni nivo s tistim, ki se dejansko pojavi na vodilu,
   - preverjanje okvirov sporočil s ciklično redundančno kodo (CRC),
   - postopek varnostnega vrivanja in brisanja bitov ("bit stuffing"), ki je opisan v nadaljevanju.

9. Potrjevanje prenosa: vsi prejemniki preverjajo skladnost prejetega sporočila. V primeru, da je sprejeto sporočilo brez napak, pošljejo potrditveno sporočilo ("acknowledgement"), v nasprotnem primeru pa o napaki obvestijo s posebnimi zastavicami.

10. Število naprav: na posamezno vodilo CAN lahko teoretično priključimo neomejeno število naprav, saj nismo omejeni z naslovnim prostorom. V praksi običajno največje število naprav omejuje zakasnitev na vodilu.

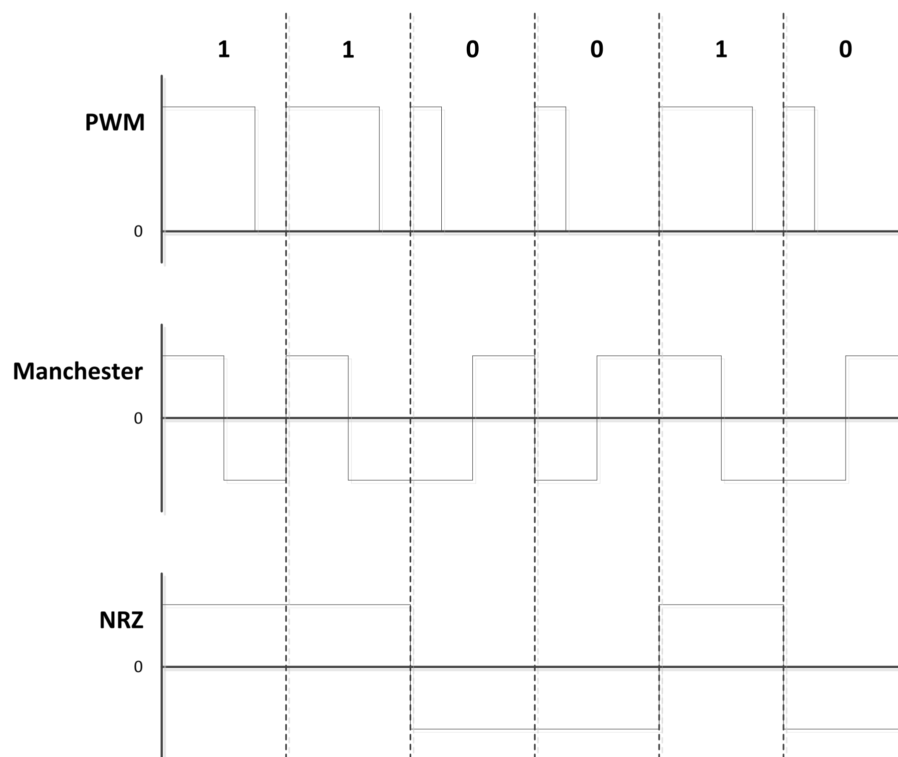
11. Nadzor nad porabo energije: z namenom zmanjševanja porabe električne energije lahko vsak izmed članov preide v tako imenovano stanje spanja ("sleep mode"), pri katerem izključi oddajno/sprejemni modul.



Slika 3.3: Arbitraža na vodilu CAN. Črka D označuje dominantno stanje, črka R pa recesivno. V točki, ki je na sliki označena s številko 1, član B izgubi v arbitražnem postopku in zato preneha z oddajanjem sporočila. Od tu naprej član A prevzame nadzor nad vodilom in nadaljuje z oddajanjem sporočila.

3.6 Kodiranje bitov

Pri kodiranju bitov gledamo trajanje posameznega bita in kako je njegova vrednost v tem času kodirana. V splošnem poznamo več vrst različnih kodiranj [11], na primer kodiranje s pomočjo pulzno-širinske modulacije (PWM), kodiranje s pomočjo spreminjanja faze (Manchester) ali NRZ (non-return-to-zero) kodiranje. Protokol CAN uporablja slednje. Za NRZ tip kodiranja je značilno, da se med samim trajanjem bita logični nivo ne spreminja. Primerjavo med različnimi kodiranji si lahko ogledamo na sliki 3.4.



Slika 3.4: Primerjava treh različnih vrst kodiranja bitov.

3.7 Vrivanje bitov

Pri komunikaciji velikokrat pride do situacije, ko sporočilo vsebuje bite z enako vrednostjo zaporedoma. Zaradi uporabe NRZ kodiranja bi to pomenilo, da se ves ta čas logična vrednost na vodilu ne bi spremenila. Za ostale člane omrežja bi to lahko izgledalo, kot da je vodilo ta čas prosto oziroma da je prišlo do napake. Za razbitje te monotonosti je poskrbljeno s postopkom vrivanja bitov (bit stuffing) [10]. Ta deluje tako, da za vsakim zaporedjem petih bitov z enako vrednostjo vrine en bit z nasprotno (komplementarno) vrednostjo. Seveda morata ta postopek obvladati tako pošiljatelj, kot tudi prejemnik. Slednji mora pri prejemu prepoznati vrinjene bite in jih odstraniti, da dobi veljavno sporočilo. V praksi ta postopek nekoliko podaljša čas prenosa sporočila, vendar igra pomembno vlogo pri varovanju njegove vsebine med prenosom.

3.8 Okviri CAN

Protokol CAN definira štiri različne tipe okvirov – podatkovnega, daljinskega, prekoračitvenega ter okvir napaka [10]. Podrobneje si bomo ogledali najpogostejšega izmed njih, to je podatkovni okvir, na koncu pa bomo navedli še lastnosti, ki so specifične za vsakega izmed ostalih okvirov. Za vse tipe okvirov velja, da so sestavljeni iz različnih polj, ta pa so sestavljena iz bitov z različnimi pomeni. Podatki tega podpoglavja veljajo za običajne CAN 2.0A okvire z 11-bitno dolžino oznake okvira.

3.8.1 Podatkovni okvir

Podatkovni okvir je najpogosteje uporabljeni tip okvira. Njegova naloga je, kot že ime samo pove, prenos podatkov med člani omrežja. Na sliki 3.5 lahko vidimo, da je sestavljen iz sedmih polj:

1. **SoF (Start of Frame): 1 bit**

Polje SoF je sestavljeno iz enega samega dominantnega bita, ki ostalim članom omrežja sporoča začetek prenosa okvira. Prenos se lahko prične samo, če je vodilo do tega trenutka prosto, vse naprave pa morajo biti med seboj sinhronizirane.

2. Arbitražno polje: 12 bitov

To polje je pomembno za postopek arbitraže in je sestavljeno iz dveh delov:

1. Oznaka okvira – 11 bitov: služi kot identifikator celotnega sporočila in uvaja prioritete. Biti so označeni z oznakami od ID10 do ID0 in so v tem vrstnem redu tudi preneseni, pri čemer ima bit z oznako ID10 največjo težo. Zaradi skladnosti s starejšimi vezji velja dogovor, da prvih 7 najpomembnejših bitov ne sme biti hkrati v recesivnem stanju, kar nam da največje število identifikatorjev kot $2^{11} - 2^4 = 2048 - 16 = 2032$ kombinacij.

2. RTR (Remote Transmission Request) – 1 bit: ta bit ločuje podatkovni okvir od daljinskega. V podatkovnem paketu je vedno v dominantnem stanju.

3. Kontrolno polje: 6 bitov

Sestavljeno je iz dveh delov, pri čemer sta prva dva bita (v 2.0A okvirih sta dominantna) rezervirana za kasnejšo uporabo in zagotavljata bodočo skladnost z novejšimi okviri (na primer 2.0B). Naslednji štirje biti navajajo dolžino podatkov v bajtih, ki se prenašajo v naslednjem, podatkovnem polju.

4. Podatkovno polje: 0–64 bitov

Podatkovno polje prenaša uporabne informacije med člani v omrežju. Zavzame lahko vse dolžine med 0 in 8 bajti. To nam da 9 različnih vrednosti, zaradi česar potrebujemo v kontrolnem polju 4 bite za določanje dolžine podatkov namesto 3.

5. Polje CRC: 16 bitov

CRC je okrajšava za Cyclic Redundancy Code oziroma ciklično redundančno kodo. Pomembna je pri zagotavljanju veljavnosti in integritete prenešenega sporočila. Polje je razdeljeno na dva dela:

1. Zaporedje CRC – 15 bitov: podatki, ki jih nosi sporočilo, so zaščiteni z zaporedjem CRC. Zaporedje izračuna in ga v sporočilo doda pošiljatelj, prejemnik pa glede na prejete podatke ponovno izračuna zaporedje CRC in ga primerja s tistim, ki ga je izračunal pošiljatelj. V primeru ujemanja zaporedij se je sporočilo preneslo brez napak in ga zato prejemnik potrdi kot pravilnega, v nasprotnem primeru pa pošiljatelja obvesti o napaki. Zaporedje se izračuna s pomočjo BCH (Bose-Chaudhuri-Hocquenghem) kodiranja in polinomom (3.1) [12], s pomočjo katerega lahko odkrijemo do pet neodvisnih napak v sporočilu.

$$g(X) = X^{15} + X^{14} + X^{10} + X^{8} + X^{7} + X^{4} + X^{3} + 1 \tag{3.1}$$

2. Razmejilnik CRC – 1 bit: zaporedje CRC in naslednje polje ločuje razmejilnik CRC, ki je sestavljen iz enega samega recesivnega bita.

6. Polje ACK: 2 bita

Polje ACK služi kot potrditveno polje za potrjevanje pravilnosti sprejetega sporočila. Sestavljeno je iz dveh delov:

1. Predalček ACK – 1 bit: pri pošiljanju je predalček ACK vedno postavljen v recesivno stanje. Če se je sporočilo preneslo brez napak (vključno s preverbo CRC), prejemniki postavijo predalček v dominantno stanje. Na tem mestu je pomembno opozoriti na dejstvo, da dominantno stanje predalčka ACK pomeni zgolj to, da je vsaj eden izmed prejemnikov uspešno prejel sporočilo brez napak, saj je dominantno stanje seveda nemogoče "preglasiti". Prejmenik, ki je prejel sporočilo z napakami, mora pošiljatelja obvestiti s posebnim okvirom (okvir napaka).

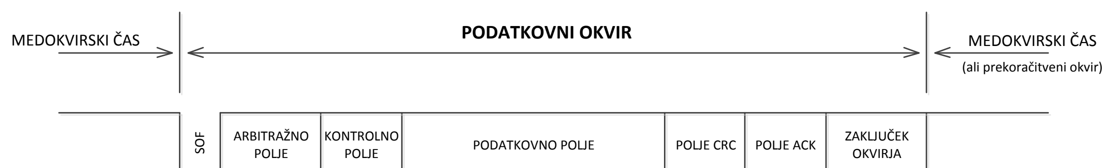
2. Razmejilnik ACK – 1 bit: predstavlja ločitev med predalčkom ACK in zaključkom okvira. Je vedno v recesivnem stanju.

7. Zaključek okvira: 7 bitov

Pošiljatelj vsak okvir zaključi s pošiljanjem sedmih zaporednih recesivnih bitov, med katerimi je vrivanje bitov onemogočeno. Ta lastnost omogoča ločevanje med okviri, saj se drugače na vodilu nikoli ne pojavi več kot pet zaporednih bitov z enako vrednostjo (razen v primeru napake).

8. Medokvirski čas: najmanj 3 bite

Medokvirski čas ni del okvira, ampak predstavlja najmanjši možen čas med oddajanji dveh okvirov.



Slika 3.5: Sestava podatkovnega okvira CAN 2.0A.

3.8.2 Daljinski okvir

Poleg podatkovnega okvira, ki neposredno prenaša podatke, definira protokol CAN tudi daljinski okvir, s katerim pošiljatelj zaprosi za določene podatke. Odgovor na daljinski okvir je podatkovni okvir, ki ima enako številko okvira kot daljinski in prenaša zahtevane podatke. Sestava daljinskega okvira (slika 3.6) je zelo podobna podatkovnemu – sestavljen je iz podobnih polj, manjka mu le podatkovno, saj ni namenjen prenosu podatkov med člani omrežja. Poleg odsotnosti podatkovnega polja sta tu še dve dodatni razliki, ki ju velja omeniti.

Arbitražno polje: bit RTR (Remote Transmission Request) je pri daljinskem okviru vedno v recesivnem stanju, kar ga najbolj jasno ločuje od podatkovnega. Pri situaciji, ko se v istem trenutku začneta prenašati tako podatkovni, kot tudi daljinski okvir in imata oba enako številko okvira, arbitražo dobi podatkovni okvir. To je tudi edino smiselno, saj podatkovni okvir že nosi podatke, ki bi jih daljinski šele zahteval.

Kontrolno polje: zadnji štirje biti ne definirajo števila podatkov (v bajtih), ki jih okvir nosi, ampak število podatkov, ki jih prejemnik pričakuje v odgovoru.

Slika 3.6: Sestava daljinskega okvira CAN 2.0A.

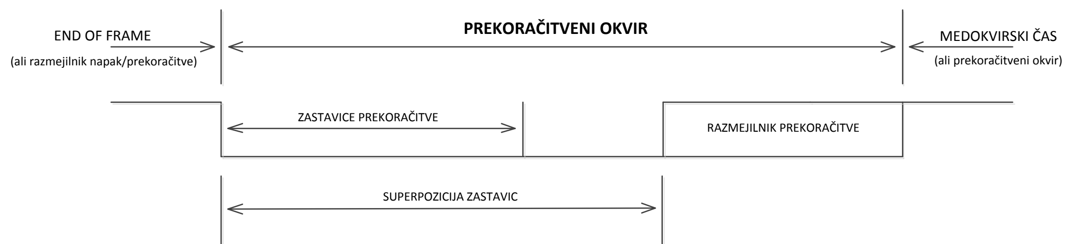
3.8.3 Prekoračitevni okvir

Namen prekoračitvenega okvira je sporočanje preobremenitve enega izmed članov omrežja. Z njim prejemnik sporočila sporoči, da potrebuje dodaten čas pred sprejemanjem novega (podatkovnega ali daljinskega) okvira. Pošilja se lahko takoj po zaključku podatkovnega ali daljinskega okvira oziroma po prenosu razmejilnika napake, brez čakanja na medokvirski čas. Skupno je celoten prekoračitveni okvir lahko dolg največ 20 bitov, njegova sestava (slika 3.7) pa je preprosta, saj ima le dva dela:

1. Zastavice prekoračitve – 6 bitov: predstavljene so s šestimi zaporednimi dominantnimi biti. Vsi ostali člani, ki zaznajo pošiljanje prekoračitvenega okvira, začnejo tudi sami oddajati prekoračitveni okvir in posledično

tudi svoje zastavice prekoračitve. Slednje lahko trajajo še največ 6 dodatnih bitov.

2. Razmejilnik prekoračitve – 8 bitov: zaporedje osmih zaporednih recesivnih bitov loči prekoračitveni okvir od medokvirskega časa oziroma naslednjega prekoračitvenega okvira.



Slika 3.7: Sestava prekoračitvenega okvira CAN 2.0A.

3.8.4 Okvir napaka

Okvir napaka sporoča različne vrste napak pri prenosu podatkov. Naprave z njim sprožijo zahtevo za ponoven prenos sporočila. Na sliki 3.8 lahko vidimo, da je njegova sestava zelo podobna tisti od prekoračitvenega okvira, saj ima le dve polji. Prvo polje sestavljajo zastavice napake in je zaradi načela superpozicije zastavic lahko dolgo med 6 in 12 biti, pri čemer so zastavice vedno dolžine 6 bitov. Obstajata dva tipa zastavic:

1. Zastavice aktivne napake: sestavlja jih 6 zaporednih dominantnih bitov.

2. Zastavice pasivne napake: sestavlja jih 6 zaporednih recesivnih bitov.

Polju zastavic sledi še razmejilnik napak, to je polje, predstavljeno z 8 zaporednimi recesivnimi biti. Skupno je tako okvir lahko dolg največ 20 bitov, kar je enako kot pri prekoračitvenem okviru. Takoj lahko ugotovimo, da je sestava okvira napake z zastavicami aktivne napake identična sestavi prekoračitvenega okvira. Kako lahko potem ločimo med tema dvema okviroma?

Razlika je v tem, da je začetek pošiljanja prekoračitvenega okvira mogoč samo po zaključku (podatkovnega ali daljinskega) okvira oziroma razmejilnika napak/prekoračitve, medtem kot se pošiljanje okvira napaka začne že med prenosom podatkovnega okvira. Podrobnosti o napakah si bomo ogledali v naslednjem poglavju, saj si zaslužijo posebno obravnavo.

Slika 3.8: Sestava okvira napaka CAN 2.0A.

3.9 Napake v omrežju CAN

Ena izmed ključnih lastnosti protokola CAN je njegova robustnost. Za zagotavljanje učinkovitega nadzora in detekcije napak uporablja več učinkovitih mehanizmov in to že na strojnem nivoju.

3.9.1 Vrste napak

Za začetek si najprej velja ogledati tipe napak [13], ki se lahko pojavijo. V splošnem jih razdelimo v dve skupini, odvisno od nivoja, na katerem se pojavijo:

1. Napake na fizičnem nivoju:

   (a) bitna napaka: vrednost bita, ki se pojavi na vodilu, se razlikuje od vrednosti poslanega bita, kar se interpretira kot napaka. Obstajata pa dve izjemi. Prva izjema je pošiljanje pošiljanje potrditvenega bita v predalčku ACK, ki je s strani pošiljatelja poslan v recesivnem stanju, prejemniki pa ga pri potrditvi prejema sporočila

postavijo v dominantno stanje. Drugo izjemo predstavlja situacija, ko pošiljatelj pošilja zastavice pasivne napake (zaporedje 6 recesivnih bitov), na vodilu pa se pojavijo dominantni biti. V obeh primerih pošiljatelj ne sme opredeliti stanja kot bitno napako.

(b) napaka pri vrivanju bitov: med poljem SoF in razmejilnikom CRC se pojavi zaporedje 6 ali več bitov z enako logično vrednostjo.

2. Napake na nivoju okvira:

   (a) napaka pri prenosu podatkov: podatki, ki so bili poslani, niso enaki prejetim. To vrsto napake odkrijmo s pomočjo ciklične redundančne kode (CRC).

   (b) napaka potrditve (ACK): pri prenosu podatkovnega okvira v predalčku ACK ne zaznamo dominantnega stanja, kar pomeni, da nihče izmed članov omrežja ni potrdil prejema sporočila.

   (c) napaka zaključka okvira: podatkovnemu oziroma daljinskemu okviru ne sledi 7 zaporednih recesivnih bitov.

   (d) napaka razmejilnika CRC

   (e) napaka razmejilnika napake

   (f) napaka razmejilnika prekoračitve

3.9.2 Soočanje z napakami

Napake, ne glede na vrsto, zmanjšujejo prepustnost omrežja in povečujejo zakasnitvene čase. Če bi jih prepustili same sebi, bi se število napak lahko nenadzorovano povečevalo, kar bi v končni fazi privedlo do situacije, ko bi bilo omrežje blokirano in posledično komunikacija nemogoča. Nujno je, da napake nadzorujemo in se nanje čim hitreje odzivamo, tako da minimizirano njihov vpliv na običajne aktivnosti v omrežju.

V ta namen imajo vsi mikrokrmilniki, ki so skladni s protokolom CAN, dva ločena notranja števca napak – števec napak pri pošiljanju ("transmit error

counter") ter števec napak pri sprejemanju ("receive error counter") [13]. Števca imata nalogo beleženja napak med pošiljanjem oziroma sprejemanjem sporočil:

- če je sporočilo pravilno poslano oziroma sprejeto, se ustrezen števec zmanjša (oziroma se ne spremeni, če je bila njegova vrednost predhodno enaka 0),
- če med pošiljanjem oziroma sprejemanjem sporočila pride do napake, se ustrezen števec poveča.

Vrednost, za katero se števec zmanjša oziroma poveča glede na uspešnost prenosa, ni proporcionalna. Vrednost, za katero se ustrezen števec pri uspešnem prenosu sporočila zmanjša, je manjša, kot tista, za katero se ustrezen števec pri zaznani napaki poveča. V daljšem časovnem obdobju to povzroči, da števec napak raste kljub dejstvu, da je število uspešno prenesenih sporočil večje od števila napak pri prenosih. V tem primeru predstavlja vrednost ustreznega števca relativno frekvenco napak, ki se dogajajo v omrežju. Razmerje med vrednostima, s katerima se ustrezen števec zmanjšuje oziroma povečuje, se stalno prilagaja razmerju med uspešno prenešenimi sporočili in tistimi, pri katerih je prišlo do napake. Protokol ima sicer kot privzeto vrednost število 8.

Glede na trenutne vrednosti števcev lahko delovanje člana omrežja razdelimo v tri skupine, pri čemer vedno vzamemo višjo vrednost od obeh števcev:

1. **Vrednost med 0 in vključno 127:**

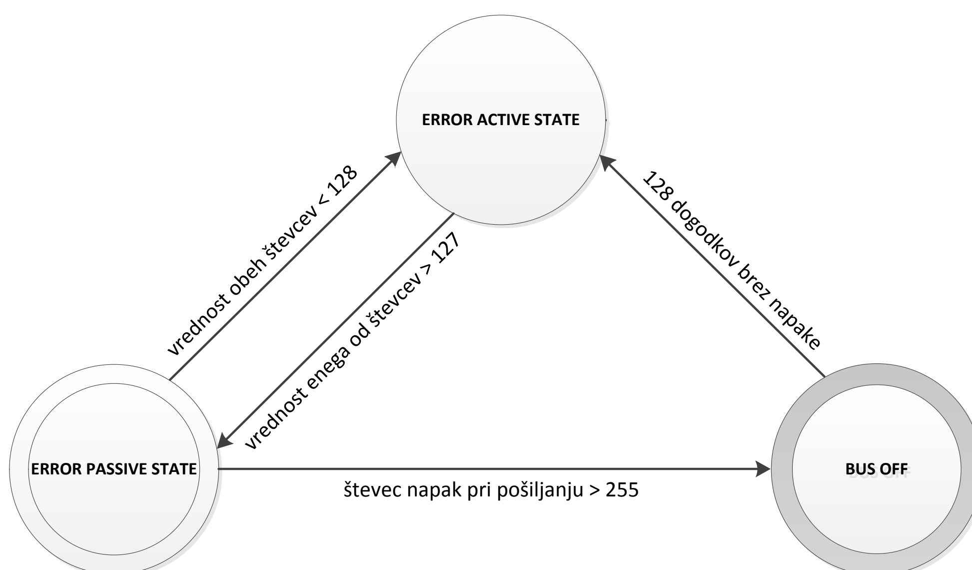
Pri vrednosti števcev med 0 in 127 deluje član omrežja v tako imenovanem stanju aktivne napake ("error active state"). V tem stanju lahko normalno pošilja in sprejema sporočila, pri napaki pa pošlje okvir napako, ki vsebuje zastavice aktivne napake.

2. Vrednost med 128 in vključno 255:

Če vrednost enega izmed števcev pride v območje med 128 in 255, član omrežja preide v stanje pasivne napake ("error passive state"). V tem stanju lahko šte vedno normalno pošilja in sprejema sporočila, vendar pri napaki pošlje okvir napako z zastavicami pasivne napake.

3. Vrednost nad 255:

Če vrednost katerega koli od števcev preseže vrednost 255, se član omrežja galvansko odklopi od vodila (visoko impedančno stanje) in preide v stanje "bus off". To pomeni, da član ni bil zmožen uspešno pošiljati oziroma prejemati sporočila in se je z namenom sprostitve omrežja odklopil od vodila. Protokol CAN določa, da mora član v tem stanju ostati dokler ne preteče 128 dogodkov (pošiljanj/sprejemov) na vodilu brez napake. Po tem času preide član v stanje aktivne napake, njegova števca pa se ponastavita na 0. Prehajanje med stanji prikazuje slika 3.9.



Slika 3.9: Prehajanje med različnimi stanji glede na vrednosti števcev napak.

3.10 Implementacija na fizičnem nivoju

Sedaj, ko smo protokol CAN dodobra spoznali na podatkovnem in logičnem nivoju, si oglejmo še njegovo praktično implementacijo na fizičnem nivoju. V splošnem obstaja več rešitev [14], mi pa se bomo v skladu s temo diplomskega dela omejili predvsem na rešitve, ki se uporabljajo v avtomobilski industriji.

3.10.1 Prenosni medij

Funkcijo prenosnega medija vodila CAN v vozilih opravljajo bakreni vodniki in sicer v treh izvedbah [15]:

1. izvedba z enim vodnikom,
2. izvedba z dvema vzporednima vodnikoma,
3. izvedba s prepletenim parom ("twisted pair").

Prva rešitev je cenovno najugodnejša, vendar je precej občutljiva na elektromagnetne motnje iz okolice. Druga rešitev rešuje problem občutljivosti na motnje, vendar zaradi vzporedne strukture tudi sama povzroča elektromagnetno onesnaževanje med prenosom sporočil. Slednjo težavo rešuje izvedba s prepletenim parom, pri čemer sta vodnika ovita drug okrog drugega in tako med prenosom sporočil vzajemno zmanjšujeta elektromagnetno onesnaževanje.

3.10.2 Električne lastnosti

Že na začetku smo omenili, da sam protokol CAN na fizični plasti ne definira podplasti PMA, katera opisuje električne lastnosti oddajno/sprejemnih modulov. Vendar pa so se kasneje pojavile potrebe tudi po standardizaciji električnih lastnosti in se tako danes v avtomobilski industriji uporabljajo trije različni standardi – dva izmed njih je postavila že omenjena organizacija ISO, enega pa združenje avtomobilskih inženirjev SAE (Society of Automotive Engineers):

1. **ISO 11898-2: High-Speed CAN**

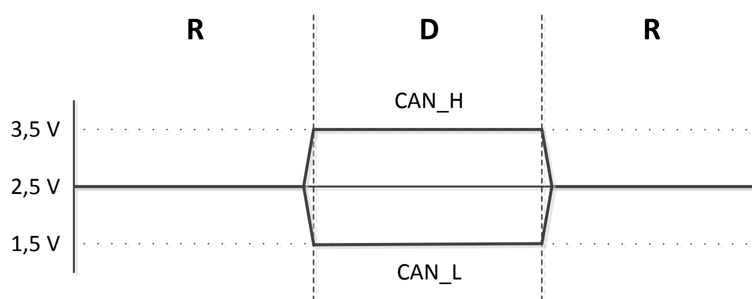
CAN visoke hitrosti [16] je najpogostejša implementacija fizičnega nivoja tako pri aplikacijah v motornih vozilih, kot tudi na splošno v celotni industriji. Organizacija ISO je pravzaprav le formalizirala obstoječe električne lastnosti, ki so se uporabljale že od začetka uporabe protokola CAN v praksi.

Glavne lastnosti:

- hitrost: od 125 $kbit/s$ do 1 $Mbit/s$,
- fizična dolžina vodila: do 40 m pri največji hitrosti 1 $Mbit/s$,
- število članov omrežja: od 2 do 30,
- prenosni medij: diferenčni prepleteni par,
- karakteristična upornost linije: 120 Ohm,
- upornost vodnika: 70 $\frac{mOhm}{m}$,
- nominalni čas širjenja signala na vodilu: 5 $\frac{ns}{m}$,
- izhodni tok oddajnika: 25 mA,
- zaščita vodila pred kratkimi stiki: od -3 V do $+16$ V,
- napajalna napetost: 5 V.

Na sliki 3.10 so prikazani napetostni nivoji po standardu ISO 11898-2. Vodnika v prepletenem paru sta poimenovana CAN_L ter CAN_H, v recesivnem stanju imata oba napetost okoli $+2,5$ V. V dominantnem stanju ima CAN_L napetost okoli $+1,5$ V, CAN_H pa okoli $+3,5$ V. Z odštevanjem napetostnih nivojev ($CAN_H - CAN_L$) dobimo diferenčno napetost, ki pri dominantnem stanju znaša okoli $+2$ V, pri recesivnem stanju pa okoli 0 V. Za pretvarjanje (odštevanje) napetostnih nivojev skrbi oddajno/sprejemni modul CAN. Tu velja omeniti, da je razlog za različne napetostne nivoje na

vodnikih CAN_L in CAN_H doseganje visoke odpornosti na motnje iz okolice. Recimo, da se v nekem trenutku po vodilu prenaša dominanten bit, hkrati pa se pojavi napetostni šum v velikosti $2\ V$. Napetost na vodniku CAN_L bo v tem primeru narastla na $3,5\ V$, na vodniku CAN_H pa na $5,5\ V$. Oddajno/sprejemni moduli bodo napetostni stanji pretvorili v diferenčno napetost: $CAN_H - CAN_L = 5,5\ V\ -\ 3,5\ V = 2\ V$. V končni fazi bomo tako dobili pravilno diferenčno napetost za dominantno stanje ($2\ V$) in zato motnja ne bo povzročila napake.



Slika 3.10: Napetostni nivoji ISO 11898-2. S črko R je označeno recesivno stanje, s črko D pa dominantno.

V vozilih se CAN visoke hitrosti uporablja za komunikacijo med ključnimi krmilnimi enotami, ki skrbijo za delovanje, upravljanje in aktivno varnost vozila, saj so tu potrebe predvsem po hitri komunikaciji. Mednje sodijo:

- krmilna enota motorne elektronike,
- krmilna enota samodejnega menjalnika,
- krmilna enota sistema proti blokiranju koles (ABS),
- krmilna enota za nadzor stabilnosti (ESC),
- krmilna enota elektromehanske delovne zavore,
- krmilna enota elektromehanskega servovolana.

2. ISO 11898-3: Fault Tolerant Low Speed CAN

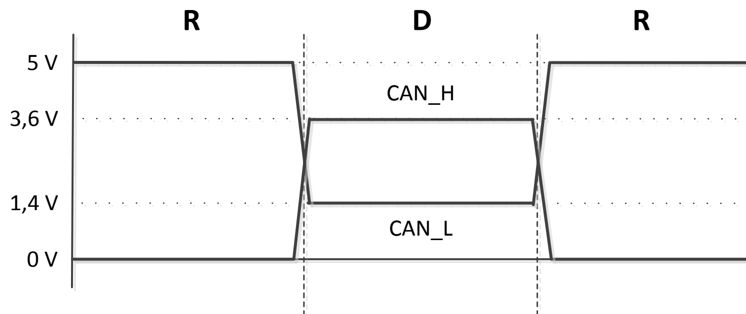
Novejši standard ISO 11898-3 [17] se zadnje čase uveljavlja predvsem za komunikacijo med krmilnimi enotami, kjer hitrost ni ključnega pomena ali tam, kjer je potreba po večji robustnosti komunikacije. Zaradi drugačnih električnih lastnosti ni kompatibilen s standardom ISO 11898-2, kar pomeni da v primeru sobivanja deluje na fizično ločenem vodilu.

Glavne lastnosti:

- hitrost: do 125 $kbit/s$,
- fizična dolžina vodila: omejena samo s kapacitivno obremenitvijo vodila,
- število članov omrežja: od 2 do 20,
- prenosni medij: diferenčni prepleteni par,
- izhodni tok oddajnika: 1 mA,
- zaščita vodila pred kratkimi stiki: od -6 V do $+16$ V,
- napajalna napetost: 5 V,
- nizka stopnja elektromagnetnega onesnaževanja.

Na sliki 3.11 lahko vidimo, da so napetostni nivoji pri standardu ISO 11898-3 (CAN nizke hitrosti) drugačni od tistih pri omrežju CAN visoke hitrosti. Dominanten bit je na vodniku CAN_H predstavljen z napetostjo $3,6$ V, na vodniku CAN_L pa z napetostjo $1,4$ V. Recesiven bit je na vodniku CAN_H predstavljen z napetostjo 0 V, na vodniku CAN_L pa z napetostjo 5 V. Z odštevanjem napetostnih nivojev ponovno pridemo do diferenčne napetosti, ki za dominanten bit znaša $2,2$ V, za recesiven bit pa 5 V. Poleg nižje hitrosti ima CAN nizke hitrosti še eno prednost – zmanjšano občutljivost na napake ("fault tolerant"). V praksi to pomeni, da pri napaki na enem izmed vodnikov (prekinitev tokokroga, kratek stik z napajalno napetostjo ...)

komunikacija ne izostane, ampak se z zmanjšano hitrostjo (in občutljivostjo na motnje) nadaljuje le z uporabo preostalega vodnika.



Slika 3.11: Napetostni nivoji ISO 11898-3. S črko R je označeno recesivno stanje, s črko D pa dominantno.

CAN po standardu ISO 11898-3 se v vozilih čedalje bolj uporablja za komunikacijo med manj kritičnimi krmilnimi enotami, na primer med tistimi, ki skrbijo za signalizacijo, udobje in dostop. Mednje sodijo:

- krmilna enota razsvetljave,
- krmilna enota merilnikov,
- krmilna enota brisalnikov,
- krmilna enota dežnega in svetlobnega senzorja,
- krmilna enota za nadzor nad napajanjem,
- krmilna enota za nadzor tlaka v pnevmatikah,
- krmilna enota elektronske blokade motorja,
- krmilne enota zračnih blazin,
- krmilna enota klimatske naprave,
- krmilna enota panoramske strehe,

- krmilna enota parkirnih senzorjev,
- krmilna enota za upravljanje sedežev,
- krmilna enota vratnih modulov.

Za konec je potrebno omeniti, da vse prej naštete krmilne enote v starejših vozilih, opremljenih z vodilom CAN, komunicirajo preko omrežja CAN po standardu ISO 11898-2, vendar z zmanjšano hitrostjo (običajno 125 $kbit/s$). Razlog je običajno ravno v električni nekompatibilnosti standarda ISO 11898-3 s standardom ISO 11898-2. V novejših vozilih pa vedno pogosteje srečujemo tudi vodilo CAN po standardu ISO 11898-3, med drugim tudi zaradi manjše porabe električne energije v mirovanju. Zaradi vedno večjega števila krmilnih enot postaja nizka poraba čedalje bolj pomembna.

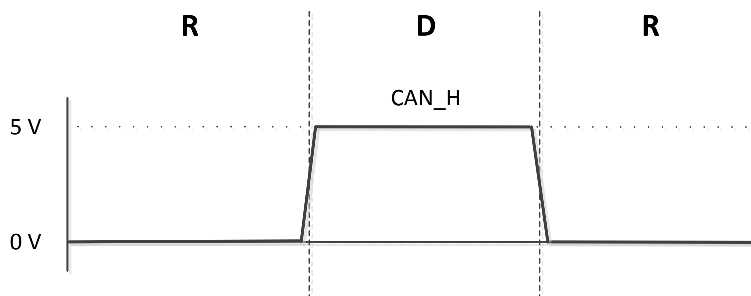
3. SAE J2411: Single Wire CAN

Že iz imena lahko sklepamo, da je bistvena razlika standarda SAE J2411 v primerjavi s prejšnjima standardoma v tem, da je komunikacijsko vodilo sestavljeno (poleg vodnika za maso) le iz enega vodnika [17]. Glavni razlog za to je ekonomski, saj je tako omrežje cenejše od tistega s prepletenim parom. Po drugi strani pa je ta rešitev bistveno manj robustna, saj ob prekinitvi edinega komunikacijskega vodnika pride do popolnega izpada omrežja. Poleg tega dosega tudi bistveno nižje hitrosti. Iz napisanega lahko sklepamo, da se omrežja CAN po tem standardu uporabljajo le za komunikacijo med nekritičnimi krmilnimi enotami.

Glavne lastnosti:

- hitrost: $33,3\ kbit/s$ ali $83,3\ kbit/s$,
- število članov omrežja: od 2 do 32,
- prenosni medij: enojni vodnik,
- napajalna napetost: $12\ V$.

Napetostni nivoji po standardu J2411 (na sliki 3.12) so bistveno drugačni od obeh protokolov ISO. Ker imamo samo en vodnik, je recesivno stanje predstavljeno z napetostjo 0 V, dominantno stanje pa z napetostjo 5 V.



Slika 3.12: Napetostni nivoji J2411. S črko R je označeno recesivno stanje, s črko D pa dominantno.

Zanimivo je dejstvo, da so omrežja CAN po standardu J2411 v avtomobilski industriji prava redkost. Uporablja jih namreč le ameriški proizvajalec General Motors, ki v Evropi trži znamke Chevrolet, Opel in Vauxhall. Uporablja ga za komunikacijo med naslednjimi krmilnimi enotami:

- krmilna enota klimatske naprave,
- krmilna enota radija,
- krmilna enota elektromehanske nastavitve sedežev in ogledal,
- krmilna enota ogrevanja sedežev,
- krmilna enota vlečne prikolice,
- krmilna enota vmesnika za telefon,
- krmilna enota sistema za opozarjanje na spremembo voznega pasu,
- krmilna enota za samodejno upravljanje z bleščečimi lučmi,
- krmilna enota parkirnih senzorjev.

Poglavje 4

On-Board Diagnostics (OBD)

On-Board Diagnostics (OBD) predstavlja sistem pravil in priporočil za diagnostiko vozil, opremljenih z elektronskimi sistemi. Sistem uporabnikom sporoča morebitne težave pri delovanju vozila, serviserjem pa z uporabo ustreznih naprav omogoča vpogled v trenutno stanje vozila in jim s tem pomaga pri odkrivanju vzrokov težav in okvar. Osredotoča se predvsem na parametre, ki vplivajo na onesnaževanje okolja.

4.1 Zgodovina

Za razumevanje razlogov, ki so pripeljali do uvedbe sistema OBD, je potrebno skočiti nekaj desetletij nazaj. V šestdesetih letih prejšnjega stoletja je v Združenih državah Amerike prodaja osebnih vozil začela skokovito naraščati. Zaradi vedno večjega obsega prometa v mestih so se tam začele pojavljati težave zaradi onesnaževanja zraka z izpušnimi plini, kar je privedlo do vedno pogostejšega smoga. Vladne oblasti so zato leta 1968 izdale regulativo, ki je zahtevala uvedbo sistemov za nadzor nad emisijami vozil [18]. Dve leti kasneje je bila ustanovljena tudi agencija za zaščito okolja (Evironmental Protection Agency), ki je izdala še nekaj novih zahtev in ukrepov za nadzor izpustov vozil.

Vsi ti ukrepi so proizvajalce vozil pravzaprav prisilili v uporabo elektronsko krmiljenih sistemov za upravljanje in nadzor motornih agregatov, saj z mehanskimi rešitvami niso mogli dosegati predpisanih standardov. Motorji so dobili vrsto elektronskih senzorjev za merjenje parametrov med delovanjem, s pomočjo katerih je elektronska krmilna enota prilagajala delovanje motorja ter tako skušala zagotoviti kaj najmanjšo možno stopnjo onesnaževanja.

Z uvedbo elektronskih sistemov se je pojavila tudi potreba po diagnostiki njihovega delovanja v primeru okvar. Na začetku so proizvajalci ta problem reševali vsak po svoje z uvedbo lastnih načinov in standardov za diagnostiko. Zametke sodobnega sistema za diagnostiko je postavilo podjetje General Motors z uvedbo diagnostičnega komunikacijskega protokola ALDL (Assembly Line Diagnostic Link) [19]. Ta je omogočal diagnostiko s uporabo tako imenovanih "utripnih kod". Pri stiku dveh pinov v diagnostičnem priključku je posebna lučka "Check Engine" na števcu z utripanjem v ustreznem zaporedju sporočala vrsto napake. Leta 1988 je združenje avtomobilskih inženirjev (SAE) predlagalo vrsto standardiziranih diagnostičnih signalov, poleg tega pa tudi standardiziran diagnostični priključek [20]. Tri leta kasneje je organizacija CARB (California Air Resources Board) objavila zahtevo, da morajo vsa nova vozila omogočati osnovno diagnostiko elektronskih naprav po standardu, ki ga danes poznamo kot OBD-I. Leta 1994 je nato ista organizacija izdala zahtevo po uvedbi standarda, ki ga uporabljamo še danes: OBD-II. Slednji je upošteval večino predlogov združenja avtomobilskih inženirjev iz leta 1988, za razliko od OBD-I pa je dodatno standardiziral tudi diagnostični priključek ter njegovo mesto vgradnje. Kasneje so pojavile še nekatere dopolnitve standarda, na primer v Evropi (EOBD) ter na Japonskem (JOBD).

4.2 Značilnosti OBD-II

Standard OBD-II, ki je v veljavi danes, kot že rečeno prinaša številne dodatne izboljšave in standardizacije v primerjavi z OBD-I. Poleg poenotene oblike

diagnostičnega priključka prinaša tudi standardizirano razporeditev pinov, še pomembneje pa je dejstvo, da standardizira signalne protokole ter obliko sporočil. Serviser lahko tako z uporabo enega samega orodja diagnosticira katerokoli vozilo, ki podpira OBD-II standard: orodje pošilja poizvedbe ter tako pridobi podatke o trenutnih informacijah ali o morebitnih zabeleženih napakah.

4.2.1 Diagnostični priključek

Priključek OBD-II je trapezaste oblike in ima 16 pinov (slika 4.1). Standard določa, da mora biti vgrajen v potniškem prostoru v bližino volanskega obroča, biti pa mora tudi enostavno dostopen [21]. Razporeditev pinov je definirana v dokumentu SAE J1962 [22]:

1: odvisno od proizvajalca

2: pozitivna linija (SAE-J1850 PWM / VPW)

3: odvisno od proizvajalca

4: masa šasije avtomobila

5: masa signala

6: CAN_H (ISO 15765-4 / SAE J2284)

7: linija K (ISO 9141-2 / ISO 14230-4)

8: odvisno od proizvajalca

9: odvisno od proizvajalca

10: negativna linija (samo SAE-J1850 PWM)

11: odvisno od proizvajalca

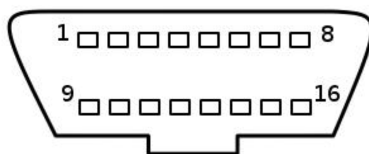
12: odvisno od proizvajalca

13: odvisno od proizvajalca

14: CAN_L (ISO 15765-4 / SAE J2284)

15: linija L (ISO 9141-2 / ISO 14230-4)

16: pozitivna napetost baterije (+12V)



Slika 4.1: Diagnostični priključek OBD-II. (Vir: [23])

4.2.2 Signalni protokoli

V okviru protokola OBD-II je definiranih pet signalnih (komunikacijskih) protokolov [24]:

- SAE J1850 PWM,
- SAE J1850 VPW,
- ISO 9141-2,
- ISO 14230 (KWP2000),
- ISO 15765 (CAN).

V okviru diplomske naloge se bomo osredotočili le na zadnjega, ki za komunikacijo med elektronskimi krmilniki in diagnostično napravo koristi kar omrežje CAN. Pin številka 6 je priključen na vodnik CAN_H, pin številka 16 pa na vodnik CAN_L. Maso najdemo na pinu številka 5. Na tem mestu je pomembno omeniti dejstvo, da prisotnost omrežja CAN na vozilu še ne pomeni, da lahko preko njega vzpostavimo diagnostično komunikacijo preko standarda OBD-II. Veliko vozil (za orientacijo lahko vzamemo vozila, narejena pred letom 2005), pri katerih kontrolne enote med samo sicer komunicirajo preko omrežja CAN, za namen diagnostike po standardu OBD-II uporablja namreč enega izmed ostalih štirih signalnih protokolov. Vozila, ki standarda ISO 15765 ne podpirajo, običajno prepoznamo po odsotnosti pinov 6 ter 14 v diagnostičnem priključku (prisotnost teh pinov pa še ne pomeni podpore temu standardu).

4.2.3 Načini delovanja

Standard OBD-II definira 10 različnih načinov delovanja [25] (predpona 0x označuje šestnajstiške vrednosti):

- način 0x01: prikaz trenutnih podatkov,
- način 0x02: prikaz podatkov v "zmrznjenem" okviru,
- način 0x03: prikaz zabeleženih diagnostičnih napak (DTC),
- način 0x04: brisanje zabeleženih diagnostičnih napak (DTC),
- način 0x05: rezultati preizkusa delovanja tipala kisika (vozila brez omrežja CAN),
- način 0x06: rezultati preizkusa drugih tipal oziroma delovanja tipala kisika, pri vozilih z omrežjem CAN,
- način 0x07: prikaz diagnostičnih napak, zabeleženih v trenutnem voznem ciklu,
- način 0x08: aktivacija sprožil z uporabo diagnostične naprave,
- način 0x09: prikaz informacij o vozilu,
- način 0x0A: prikaz stalnih napak (DTC).

Proizvajalci niso dolžni implementirati vseh načinov, lahko pa dodajo tudi svoje.

4.2.4 Parametri

Informacije, ki jih dobivamo preko protokola OBD-II, so v obliki parametrov (PID-i) [26]. V tabeli 4.1 najdemo nekaj parametrov, ki se v praksi najbolj uporabljajo.

4.3 OBD-II in omrežje CAN

V prejšnjem poglavju smo si ogledali značilnosti protokola OBD-II, ki so skupne ne glede na signalni protokol. Oblika sporočil in realizacija komunikacije pa je od protokola do protokola različna. Oglejmo si, kako poizvedbe in odgovori izgledajo v omrežju CAN.

4.3.1 Poizvedbe

Diagnostična naprava običajno pošlje okvir CAN z oznako 0x7DF [27], ki si jo lahko predstavljamo kot naslov za razpošiljanje. Na okvir s to oznako se namreč odzovejo vse krmilne enote v omrežju CAN, ki podpirajo standard OBD-II . Za pošiljanje poizvedbe do točno določene krmilne enote se uporabljajo naslovi med 0x7E0 in 0x7E7 (poizvedbe lahko torej pošiljamo največ osmim različnim krmilnim enotam). V večini primerov se na okvir z oznako 0x7E0 odziva glavna krmilna enota, ki skrbi za upravljanje z motorno elektroniko. Čeprav naj bi standard OBD-II podpirale različne krmilne enote, ga v praksi skoraj vedno podpira le krmilna enota motorne elektronike.

Poizvedbo poleg oznake okvira seveda sestavljajo tudi podatki, ki so odvisni od vrste parametra, po katerem poizvedujemo. Uporabljeni so samo prvi trije bajti okvira CAN, njihov pomen pa lahko vidimo v tabeli 4.2. Zadnjih pet bajtov nima definiranega pomena, zato lahko imajo poljubno vrednost.

4.3.2 Odgovori

Krmilna enota (oziroma več izmed njih) se na poizvedbo odzove s pošiljanjem okvira CAN z zahtevanimi podatki. Oznaka okvira odgovora je običajno za 8 večja od tiste pri zahtevi [28]. Krmilna enota motorne elektronike se tako odzove z okvirom CAN, ki ima oznako 0x7E8. Kot lahko vidimo v tabeli 4.3, uporabne podatke nosi največ prvih 7 bajtov okvira CAN.

4.3.3 Izračun vrednosti parametra

Bajti 3–6 v okviru CAN, ki ga kot odgovor pošlje krmilna enota, vsebujejo delne vrednosti parametra. Končno vrednost parametra diagnostična naprava izračuna z uporabo enačb (tabela 4.4) [29] oziroma z ustreznim dekodiranjem bitov. Delne vrednosti (bajti 3–6) so označene s črkami od A do D. Velika večina delnih vrednosti zasede le prva dva bajta.

Tabela 4.1: Seznam nekaterih najpogostejših OBD-II PID-ov.

način	PID	opis	$vrednost_{min}$	$vrednost_{max}$	enote
0x01	0x04	obremenitev motorja	0	100	%
0x01	0x05	temp. hladilne tekočine	-40	215	°C
0x01	0x0C	vrtljaji motorja	0	16383,75	rpm
0x01	0x0D	hitrost vozila	0	255	km/h
0x01	0x0F	temp. vstopnega zraka	-40	215	°C
0x01	0x10	masni pretok zraka	0	655,35	g/s
0x01	0x11	položaj pedala za plin	0	100	%
0x01	0x14	napetost kisikove sonde 1	0	1,275	V
0x01	0x15	napetost kisikove sonde 2	0	1,275	V
0x01	0x23	tlak goriva (skupni vod)	0	655350	kPa
0x09	0x02	številka šasije (VIN)	-	-	-

Tabela 4.2: Sestava poizvedbe OBD-II v omrežju CAN.

bajt 0	bajt 1	bajt 2	bajti 3–7
število dodatnih podatkovnih bajtov	način delovanja	PID	brez pomena
vedno 0x02	primer: 0x01 (prikaz trenutnih podatkov)	primer: 0x0D (hitrost vozila)	poljubna vrednost

Tabela 4.3: Sestava odgovora OBD-II v omrežju CAN.

bajt 0	bajt 1	bajt 2	bajti 3–6	bajt 7
število dodatnih podatkovnih bajtov	način delovanja z odmikom 0x40	PID	delne vrednosti parametra	brez pomena
med 0x03 in 0x06	primer: 0x49 (prikaz informacij o vozilu)	primer: 0x02 (številka šasije)	primer: podatki šasije vozila	poljubna vrednost

Tabela 4.4: Enačbe za izračun končnih vrednosti nekaterih parametrov OBD-II.

PID	opis	enačba za izračun
0x04	obremenitev motorja	$A * 100/255$
0x05	temp. hladilne tekočine	$A - 40$
0x0C	vrtljaji motorja	$((A * 256) + B)/4$
0x0D	hitrost vozila	A
0x0F	temp. vstopnega zraka	$A - 40$
0x10	masni pretok zraka	$((A * 256) + B)/100$
0x11	položaj pedala za plin	$A * 100/255$
0x14	napetost kisikove sonde 1	$A/200$
0x15	napetost kisikove sonde 2	$A/200$
0x23	tlak goriva (skupni vod)	$((A * 256) + B) * 10$

Poglavje 5

Orodje za diagnostiko vozil z omrežjem CAN

V prejšnjih poglavjih smo se podrobneje spoznali z delovanjem protokola CAN ter standarda OBD-II, sedaj pa želimo to znanje s pridom uporabiti tudi v praksi. Razviti želimo preprosto orodje za diagnostiko vozil, opremljenih z elektronskimi sistemi, ki so med seboj povezani prek omrežja CAN. Z njim bo lahko serviser pregledoval parametre OBD-II, kakor tudi spremljal promet v omrežju CAN.

5.1 Motivacija

Na tržišču obstaja kar nekaj diagnostičnih orodij s podobnimi funkcionalnostmi kot to, ki ga želimo razviti. Obstoječe rešitve lahko v grobem razdelimo v dve skupini. Prvo predstavljajo cenena diagnostična orodja, ki v večini primerov izvirajo iz Kitajske. Problem teh je predvsem pomanjkanje funkcionalnosti, saj omogočajo le spremljanje osnovnih parametrov OBD-II, ne pa tudi prometa v omrežju CAN. Poleg tega so ta orodja običajno precej toga in ne omogočajo nadgradenj programske opreme. Nenazadnje predstavlja slabost tudi precej nezanesljivo delovanje. Edina njihova prednost tako ostaja cena.

V drugo skupino spadajo profesionalna diagnostična orodja. Ta imajo širok nabor funkcionalnosti in serviserjem omogočajo izvajanje temeljite diagnostike elektronskih sistemov v vozilih. Orodja iz te skupine so poleg zanesljivega delovanja tudi bistveno prilagodljivejša, saj njihovi proizvajalci nenehno skrbijo za posodobitve programske opreme. Naštete lastnosti terjajo svoj davek – slabost teh orodij se praviloma kaže v njihovi visoki ceni. Kljub vsemu pa v tej skupini zelo težko najdemo orodje, ki bi omogočalo spremljanje prometa CAN. Za to nalogo je običajno potrebno kupiti namensko orodje, kar pomeni dodaten finančni strošek. Zaradi kompleksnosti so ta orodja tudi nekoliko zahtevnejša za uporabo in zato v nekaterih primerih nekoliko okorna pri izvajanju hitrega diagnostičnega pregleda.

Z našim orodjem želimo zapolniti vrzel med tema dvema skupinama. Pravzaprav želimo ostati cenovno blizu orodjem iz prej opisane prve skupine, pri čemer želimo kar največ lastnosti tistih iz druge skupine. Poleg prikaza parametrov OBD-II želimo implementirati tudi spremljanje prometa CAN. Ena izmed poglavitnih prednosti našega orodja bo tudi njegova prilagodljivost, saj si želimo dodajanja novih funkcionalnosti v prihodnosti. Čeprav se zavedamo, da bo naše orodje težko konkuriralo tistim iz druge skupine, profesionalnim, želimo odpraviti še eno njihovo slabost – zamudno uporabo. V ta namen bo zato pomembno enostavno rokovanje ter preprost in intuitiven uporabniški vmesnik.

5.2 Strojni del

Naše diagnostično orodje bo, tako kot tudi vsa druga tovrstna orodja, sestavljeno iz strojnega in programskega dela. Po kratkem premisleku ugotovimo, da so naloge strojnega dela v grobem razdeljene na 3 dele:

1. pridobivanje podatkov,

2. obdelava podatkov,

3. prikaz informacij.

Iščemo torej takšno množico komponent, ki bodo te tri naloge kar najbolj učinkovito opravljale, pri čemer pomemben faktor predstavljata enostavnost razvoja ter seveda cena. Omenjene zahteve kar kličejo po izbiri tako imenovanih razvojnih ploščic. Slednje so pravzaprav miniaturni sistemi z mikroprocesorjem, okoli katerega najdemo veliko število raznovrstnih vhodno/izhodnih naprav. Njihovo veliko prednost predstavlja zelo ugodno razmerje med ceno in zmogljivostjo, saj za malo denarja ponujajo veliko funkcionalnosti. Proizvajalci teh ploščic so običajno proizvajalci mikroprocesorjev. Z razvojnimi ploščicami želijo predstaviti zmogljivosti njihovih na novo razvitih procesorjev in zato načrtno postavljajo nizke cene. Dodatno prednost predstavlja dejstvo, da gre običajno za velike proizvajalce, ki navadno omogočajo uporabo širokega nabora razvojnih okolij. Nenazadnje pa pomemben faktor predstavlja tudi dobra podpora (dokumentacija, tehnične informacije, sheme, primeri, knjižnice ...), kar bistveno olajša razvoj na teh razvojnih ploščicah. Ni naključje, da smo se za eno izmed tovrstnih ploščic odločili tudi v našem primeru.

5.2.1 Razvojna plošča STM32F4Discovery

Za osrčje našega diagnostičnega orodja smo izbrali razvojno ploščo STM32F4-Discovery (slika 5.1), izdelek podjetja STMicroelectronics. Zgrajena je okoli zelo sodobnega mikrokrmilnika STM32 serije F4, baziranega na arhitekturi ARM Cortex-M4. Oglejmo si le nekaj izmed značilnosti tega mikrokrmilnika z oznako STM32F407VGT6 [30]:

- jedro: 32-bit ARM Cortex-M4F s frekvenco 168 MHz,
- pomnilnik flash: 1 MB,
- pomnilnik SRAM: 192+4 KB,
- 3 x analogno/digitalni pretvornik,

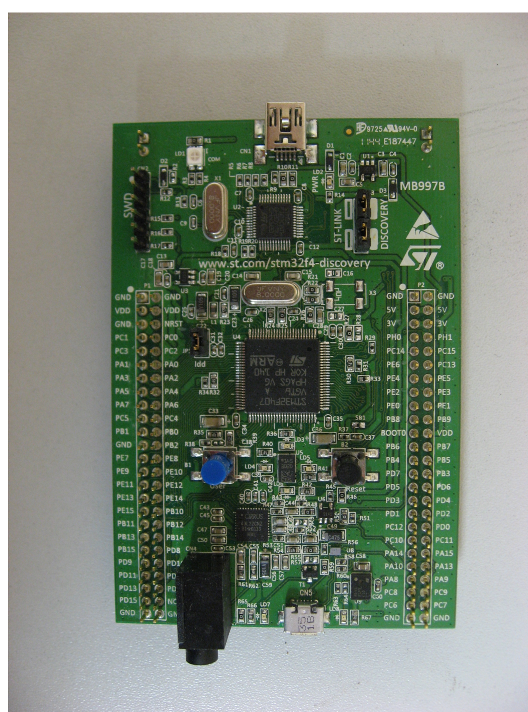
- splošno namenski DMA,
- 17 x časovnik,
- 4 x vmesnik USART / 2 x vmesnik UART,
- 3 x vmesnik SPI,
- **2 x vmesnik CAN**,
- 2 x vmesnik USB 2.0.

Poglejmo si še dodatne funkcionalnosti, ki jih prinaša razvojna ploščica STM32F4Discovery:

- enota za razhroščevanje ST-LINK/V2,
- napajanje preko priključka USB oziroma zunanje napajanje 5 V,
- pospeškometer,
- mikrofon,
- generator zvočnega signala,
- 8 x svetleča dioda (LED),
- 2 x gumb,
- fizični priključki za vse vhodno/izhodne naprave mikroprocesorja.

Iz seznamov lahko razberemo, da razvojna plošča omogoča zelo veliko funkcionalnosti in v večih pogledih seveda presega naše potrebe. Ključno pa je, da vsebuje vmesnik CAN. Po branju dokumentacije [31] se sicer izkaže, da pravzaprav ne gre za popoln vmesnik CAN, ampak le za njegov del – krmilnik. Za povezavo na dejansko omrežje CAN bo torej potrebno na ploščo priključiti še oddajno/sprejemni modul CAN, ki bo poskrbel za fizično plast protokola. Razvoj nam v veliki meri olajša tudi programska knjižnica za vmesnik CAN, ki nam, kot bomo kasneje videli, prihrani ogromno dela pri razvoju programske opreme.

Prvi dve nalogi strojne opreme smo torej skoraj v celoti (z izjemo obvladovanja fizične plasti protokola CAN) pokrili z uporabo razvojne plošče STM32F4Discovery. Kako pa je s tretjo – prikazom informacij? Na žalost proizvajalec ni razvil programske knjižnice za katerega izmed zaslonov (segmentni, LCD ...). Ocenili smo, da bo enostavneje in hitreje uporabiti dodatno razvojno ploščo, ki tako knjižnico ima, čeprav bo to pomenilo dodaten strošek. Posledično bo potrebna tudi komunikacija med razvojnima ploščama, kar pa ne predstavlja posebnega problema. STM32F4Discovery ima namreč tudi komunikacijski vmesnik UART ter programsko knjižnico za njegovo uporabo.



Slika 5.1: Razvojna plošča STMicroelectronics STM32F4Discovery.

5.2.2 Microchip MCP2551 (oddajno/sprejemni modul CAN)

Oddajno/sprejemni modul CAN, MCP2551, skrbi za fizično plast komunikacije našega diagnostičnega orodja z omrežjem CAN. Služi torej kot vmesnik med fizičnim vodilom ter krmilnikom CAN, ki je vgrajen v procesor STM32F407VGT6. Gre za preprost element v ohišju SOIC z osmimi nožicami. Med glavnimi lastnostmi najdemo [32]:

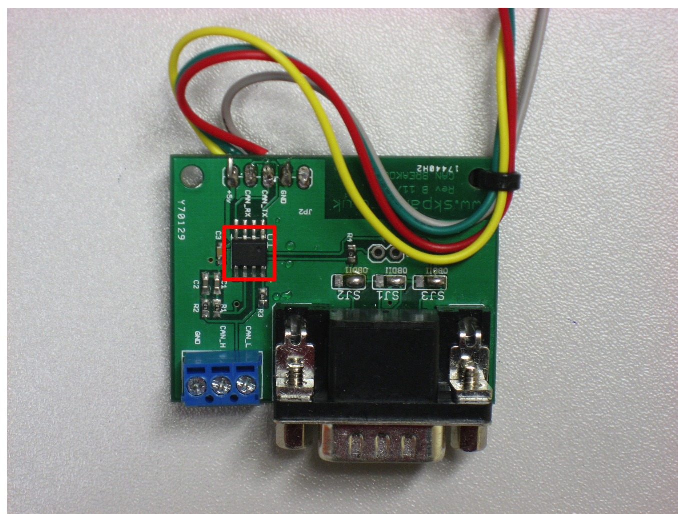
- implementira standard ISO-11898,
- podpira hitrosti do 1 Mbit/s,
- primeren za 12 V in 24 V sisteme,
- omogoča komunikacijo z do 112 člani omrežja.

Za uporabo modula MCP2551 moramo poskrbeti z ustreznim priklopom signalov. S signali CANL in CANH komunicira z vodilom CAN, s signali TXD in RXD pa s krmilnikom CAN. Poleg tega potrebuje še napajanje V_{dd} (+5 V) ter maso V_{SS}.

Za delovanje potrebuje še nekaj dodatnih analognih elektronskih elementov, zato smo se odločili, da poiščemo izdelek, ki poleg oddajno/sprejemnega modula vsebuje tudi te elemente. Rešitev ponujajo pri podjetju SK Pang Electronics in se imenuje CAN-Bus Breakout Board (slika 5.2). Gre za zelo enostavno ploščo tiskanega vezja, na kateri poleg modula MCP2551 in analognih elementov najdemo tudi moški priključek DE-9. Slednji nam olajša povezavo plošče tiskanega vezja s priključkom OBD-II.

5.2.3 Razvojna plošča Arduino Mega 2560

Arduino Mega 2560 je razvojna plošča, zgrajena okoli mikroprocesorja ATmega2560. Tako kot vsi ostali izdelki Arduino je tudi ta odprtokodnega značaja. Plošča je sicer bistveno manj zmogljiva kot STM32F4Discovery,



Slika 5.2: Can-Bus Breakout Board. Oddajno/sprejemni modul MCP2551 je označen z rdečo barvo.

vendar našim potrebam še vedno popolnoma zadostuje. Izbrali smo jo namreč za posredovanje podatkov med STM32F4Discovery in zaslonom LCD oziroma osebnim računalnikom. Zaradi odprtokodnosti je za razvojne plošče Arduino na voljo ogromno programskih knjižnic, med katerimi najdemo tudi knjižnice za uporabo različnih zaslonov LCD. Slednje najdemo tudi v posebej prilagojenih različicah za platformo Arduino, t. i. "ščitih". Tovrstne "ščite" lahko preprosto (brez spajkanja) priključimo na razvojne plošče Arduino. Poglejmo si glavne lastnosti razvojne plošče Arduino Mega 2560 [33]:

- glavni procesor: 8-bit ATmega2560 s frekvenco 16 MHz,
- dodatni procesor: 8-bit ATmega16U2 s frekvenco 16 MHz,
- 54 x digitalni vhodno/izhodni priključek,
- 16 x analogni vhod,
- 4 x vmesnik UART,
- 1 x priključek USB.

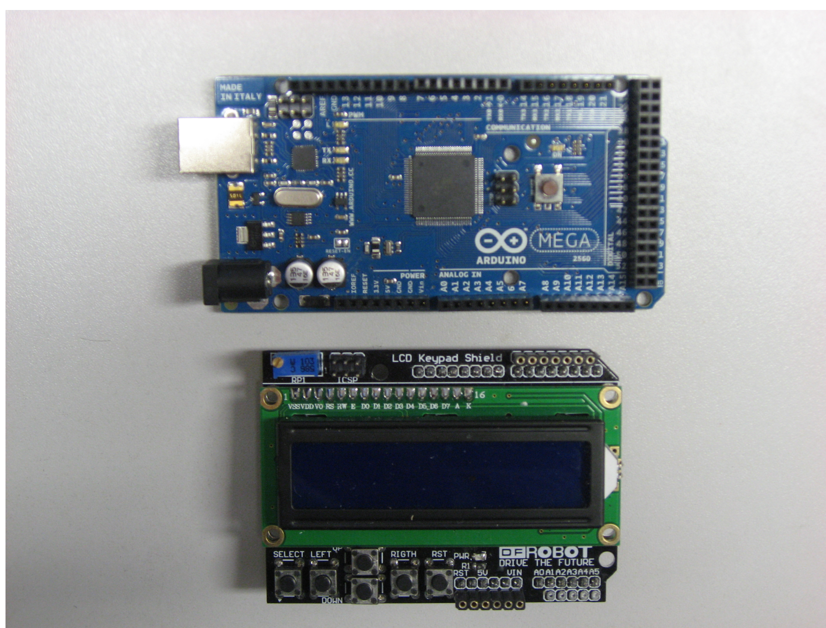
Mega 2560 ima ravno tako kot STM32F4Discovery komunikacijski vmesnik UART ter programsko knjižnico zanj. Uporabili ga bomo za vzpostavitev komunikacije med ploščama. Za priključitev zaslona LCD bomo uporabili kar splošne digitalne vhodno/izhodne priključke. Razvojna plošča Mega 2560 pa ima še eno, za nas zelo uporabno lastnost. Njen dodatni procesor, ATmega16U2, je tovarniško sprogramiran, da opravlja nalogo pretvornika med serijsko in USB povezavo [33]. Osebni računalnik pri priključitvi razvojne plošče preko povezave USB tako ustvari navidezni serijski priključek COM, ki služi za komunikacijo med ploščo in računalnikom. To lastnost bomo s pridom uporabili – če bomo želeli podatke iz razvojne plošče STM32F4Discovery pošiljati na osebni računalnik, bomo preprosto zaobšli glavni procesor in uporabili dodatni procesor. Ta bo vse sprejete podatke preprosto preposlal na osebni računalnik, brez potrebe po dodatnih konfiguracijah oziroma programiranju. Ploščo lahko vidimo na sliki 5.3.

5.2.4 LCD Keypad Shield for Arduino

Za dejanski prikaz informacij smo uporabili "ščit" LCD Keypad Shield for Arduino (slika 5.3) podjetja DFRobot. Na njem najdemo dvovrstični, 16-segmentni zaslon LCD, 6 gumbov ter potenciometer za nastavitev kontrasta zaslona. Poleg nizke cene in enostavnega priklopa ("ščit" preprosto z zgornje strani vstavimo na ploščo Arduino) obstaja zanj tudi več programskih knjižnic. Zaslon namreč uporablja zelo popularen krmilnik Hitachi HD44780 [34]. Koristno bomo uporabili tudi gumbe, saj bo s pomočjo njih uporabnik diagnostične naprave lahko izbiral, katero izmed informacij želi imeti prikazano.

5.2.5 Vezava in priklop komponent

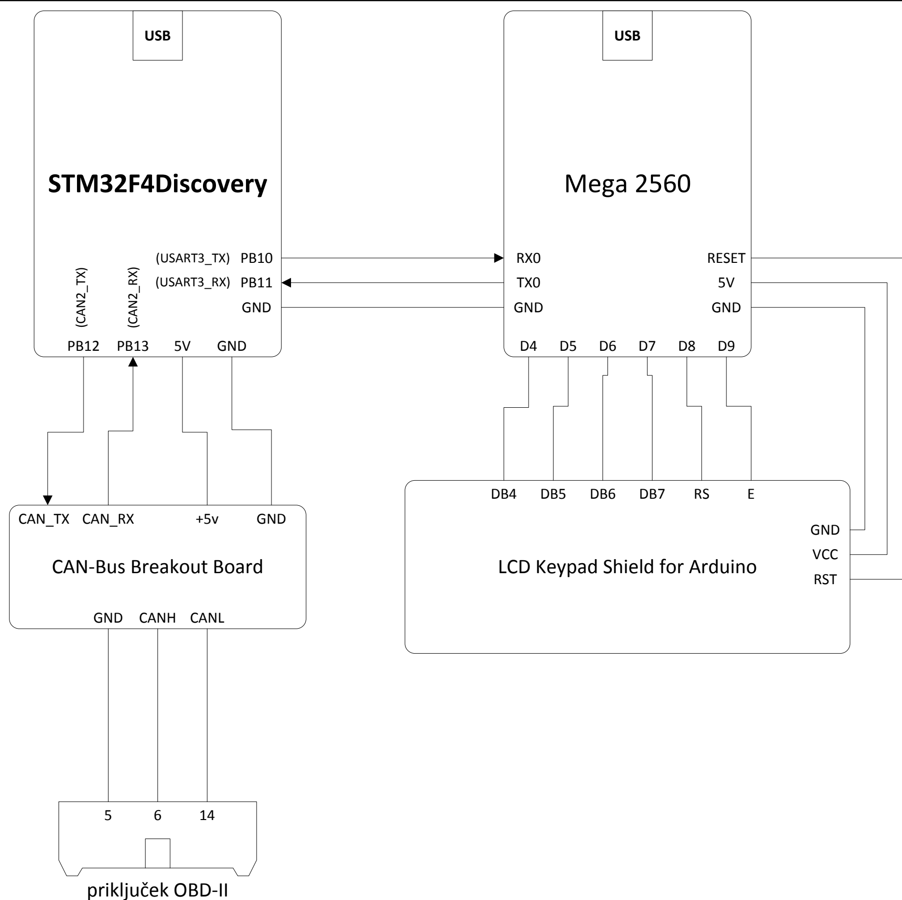
Izbrane imamo vse potrebne komponente, sedaj jih moramo le še ustrezno povezati med seboj. Začnemo z najenostavnejšim delom – priklopom 'ščita' LCD Keypad Shield for Arduino s ploščo Arduino Mega 2560. Ta korak je



Slika 5.3: Zgoraj razvojna plošča Arduino Mega 2560, spodaj "ščit" LCD Keypad Shield for Arduino.

enostaven, saj "ščit" preprosto z zgornje strani vstavimo v ploščo. Za vezavo nam torej ni potrebno skrbeti.

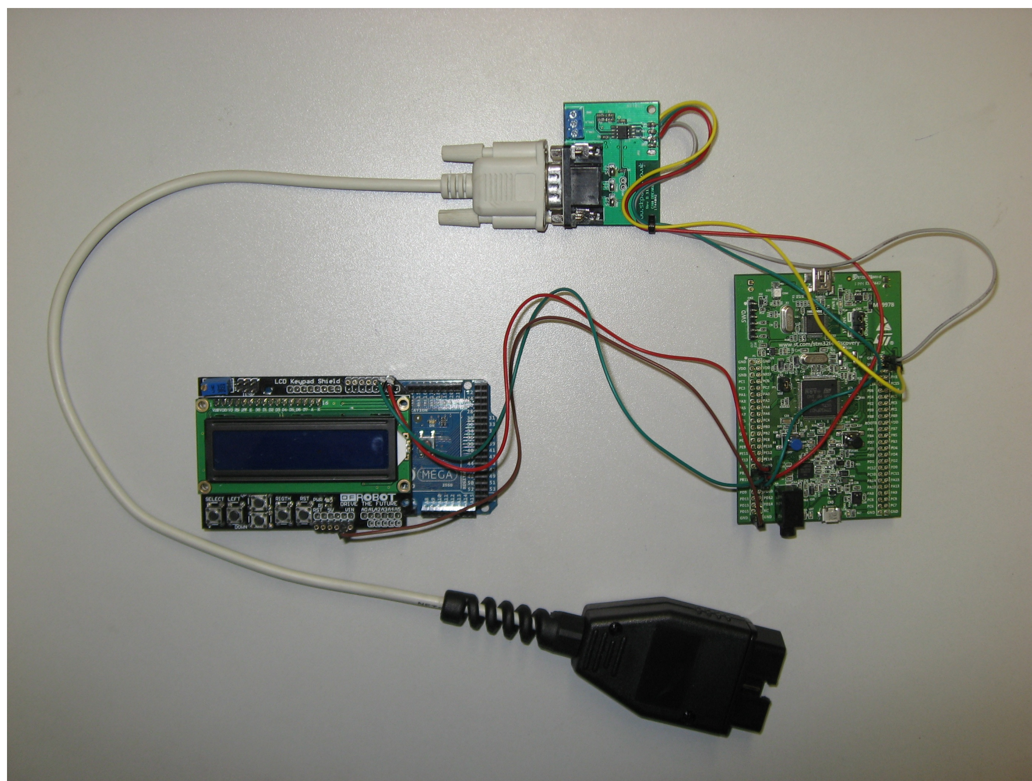
Plošči STM32F4Discovery ter Mega 2560 med seboj komunicirata s pomočjo protokola UART. Slednji zahteva tri signale – maso, signal za pošiljanje podatkov (TxD) ter signal za prejemanje podatkov (RxD). Na plošči STM32F4-Discovery bomo uporabili pina `PB10` ter `PB11`, ki ju bomo kasneje programsko nastavili kot signal za pošiljanje oziroma sprejemanje podatkov. Plošča Mega 2560 ima signal za pošiljanje podatkov označen kot `TX0`, signal za sprejemanje podatkov pa kot `RX0`. Signala za pošiljanje in prejemanje podatkov je potrebno vezati križno, zato pin `PB10` vežemo na pin `RX0`, pin `PB11` pa na pin `TX0`. Masa je na obeh ploščah označena kot `GND`, zato je njena vezava trivialna. Fizični priklop med pini izvedemo kar s t. i. mostički, ki se uporabljajo za povezovanje elementov na testnih ploščah ("breadboards"). STM32F4Discovery ima moški tip, Mega 2560 pa ženski tip priključkov. Za



Slika 5.4: Shema vezave diagnostičnega orodja.

priklop torej potrebujemo mostičke z različnima tipoma priključkov na vsakem izmed koncev.

Nadalje je potrebno ploščo STM32F4Discovery povezati z oddajno/sprejemnim modulom CAN-Bus Breakout Board. Slednji za delovanje potrebuje napajanje, maso ter dva komunikacijska signala – za pošiljanje (CAN TX) ter sprejemanje (CAN RX) podatkov. Napajanje je na plošči STM32F4Discovery označeno kot `5V`, na oddajno/sprejemnem modulu pa kot `+5v`. Masa je na obeh komponentah označena kot `GND`. Signal za pošiljanje podatkov bomo na plošči STM32F4Discovery s pomočjo programske konfiguracije pripeljali



Slika 5.5: Komponente orodja za diagnostiko, priključene med seboj. Na spodnjem delu lahko vidimo diagnostični priključek OBD-II črne barve.

do pina PB12, signal za sprejemanje podatkov pa do pina PB13. Na oddajno/sprejemnem modulu sta ta dva pina označena kot CAN_TX oziroma CAN_RX. Podatkovnih signalov v tem primeru ne smemo križati, zato pin PB12 povežemo s pinom CAN_TX, pin PB13 pa s pinom CAN_RX. Fizični priklop izvedemo z manjšo razliko v primerjavi s priklopom med ploščama STM32F4Discovery ter Mega 2560. CAN-Bus Breakout board namreč nima priključkov, zato je potrebno moški del na mostičkih odrezati ter vodnike zaspajkati neposredno na ploščo tiskanega vezja.

Preostane nam le še zadnji del. Povezati in priklopiti je potrebno moški priključek OBD-II z oddajno/sprejemnim modulom CAN-Bus Breakout Board. Slednji ima komunikacijska signala CAN_L in CAN_H ter maso povezane

v moški priključek DE-9. CAN_L je povezan na pin 5, CAN_H na pin 3, masa pa na pin 2. Signale v priključku OBD-II povežemo na sledeč način: CAN_L na pin 14, CAN_H na pin 6, maso pa na pin 5. Priklop izvedemo s kablom, ki ima na eni strani ženski priključek DB-9, na drugo stran pa zaspajkamo vodnike na ustrezna mesta v priključku OBD-II. Ne smemo pa pozabiti še na napajanje razvojnih plošč. Obe se napajata preko priključkov USB. Končno shemo vezave lahko vidimo na sliki 5.4, med seboj priključene komponente pa na sliki 5.5.

5.3 Programski del

Programski del našega diagnostičnega orodja je zaradi uporabe dveh razvojnih plošč razdeljen na dva dela. Bistveno zahtevnejši in pomembnejši del predstavlja razvoj programske opreme za ploščo STM32F4Discovery. Obsega namreč konfiguracijo večih vhodno/izhodnih naprav, med katerimi ima seveda največjo težo krmilnik CAN. Po drugi strani je razvoj programske opreme za ploščo Mega 2560 enostavnejši predvsem zaradi dveh razlogov. Prvi je ta, da Mega 2560 v našem primeru opravlja manj obsežen nabor nalog, drugi razlog pa predstavlja dejstvo, da je celotna platforma Arduino usmerjena predvsem k enostavni uporabi (programiranju). V skladu s to filozofijo je poskrbljeno, da konfiguracij čim manj ter da so te karseda enostavne.

5.3.1 Razvoj programske opreme za STM32F4Discovery

Izbira razvojnega okolja

Proizvajalec STMicroelectronics za svojo razvojno ploščo STM32F4Discovery nudi uradno podporo za štiri različna razvojna okolja [35]: Altium TASKING VX Toolset for ARM, IAR EWARM, Keil MDK-ARM ter Atollic TrueSTUDIO. Odločili smo se za slednjega, saj je osnovan na dobro znanem in popularnem okolju Eclipse. V praksi tako ni bilo potrebno privajanje na uporabniški

vmesnik, kar se je izkazalo kot prihranek časa.

TrueSTUDIO, razvit v podjetju Atollic, je v osnovi sicer plačljivo razvojno okolje (različica Pro) [36]. Kot alternativa je na voljo še različica Lite, ki je v zameno za določene omejitve na voljo brezplačno. Med najbolj očitnimi omejitvami tako najdemo odsotnost podpore za programski jezik C++ (podprta sta le zbirni jezik in programski jezik C) ter omejitev dolžine izvorne kode na 32 kB. Seveda je odveč omeniti, da za verzijo Lite tehnična podpora ni na voljo. Pri razvoju programske opreme za naše diagnostično orodje ni nobena izmed omejitev predstavljala posebne težave.

Konfiguracija urinih signalov

Za uporabo vhodno/izhodnih (v nadaljevanju V/I) naprav je najprej potrebno omogočiti urin signal na njihovih vhodih. Ta je privzeto izključen z namenom prihranka na električni energiji, saj običajno ne potrebujemo vseh V/I naprav, ki jih nek mikroprocesor ponuja.

V našem primeru je urin signal potrebno omogočiti štirim napravam. Mikroprocesor STM32F407VGT6 ima 2 krmilnika CAN, `bxCAN1` ter `bxCAN2`. Odločili smo se za uporabo slednjega, v tem primeru pa je urin signal potrebno pripeljati do obeh. Naslednja V/I naprava, ki jo potrebujemo, je krmilnik USART. Naš mikroprocesor ima sicer na voljo 6 krmilnikov U(S)ART, odločili pa smo se za `USART3`. Krmilnika `bxCAN2` in `USART3` sta povezana na notranje vodilo `APB1` s frekvenco 42 MHz. Kot zadnja so tu splošno namenska V/I vrata (GPIO), s pomočjo katerih lahko notranje signale pripeljemo do različnih pinov, ki so del teh vrat. Iz množice splošno namenskih V/I vrat smo izbrali vrata `GPIOB`, saj omogočajo, da preko njih do zunanjih pinov pripeljemo tako signale krmilnika `bxCAN2`, kot tudi krmilnika `USART3`. S pazljivim izborom teh dveh krmilnikov smo torej dosegli, da lahko njune signale pripeljemo do pinov z uporabo le enih splošno namenskih V/I vrat. `GPIOB` je za razliko od prejšnjih dveh krmilnikov povezan na notranje vodilo

AHB1 s frekvenco 150 MHz.

```
1 /* ENABLE CLOCKS */
2 /* CAN1 clock enable */
3 RCC_APB1PeriphClockCmd(RCC_APB1Periph_CAN1, ENABLE);
4 /* CAN2 clock enable */
5 RCC_APB1PeriphClockCmd(RCC_APB1Periph_CAN2, ENABLE);
6 /* USART3 clock enable */
7 RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);
8 /* GPIOB clock enable */
9 RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
```

Konfiguracija splošno namenskih V/I vrat

V naslednjem koraku je potrebno pripeljati podatkovne signale krmilnikov do zunanjih pinov, tako da jih bomo lahko kasneje fizično povezali do ustreznih naprav. To storimo z uporabo že prej omenjenih splošno namenskih V/I vrat GPIOB.

Najprej moramo določiti, na katere pine bomo pripeljali signale. Oznaka pinov, ki so na plošči STM32F4Discovery povezani s splošno namenskimi V/I vrati, se začne s črko P [37]. Sledi ji črka, ki označuje, na katera izmed vrat je določen pin povezan, tej pa še zaporedna številka pina. Glede na to, da uporabljamo vrata GPIOB, bomo podatkovne signale krmilnikov morali pripeljati do pinov z oznako PBx, pri čemer x predstavlja zaporedno številko določenega pina. Vsak izmed obeh krmilnikov ima po dva podatkovna signala – krmilnik bxCAN2 ima tako CAN2_TX in CAN2_RX, krmilnik USART3 pa USART3_TX ter USART3_RX. Iz uporabniška priročnika UM1472 [37] lahko razberemo, da je signal CAN2_TX mogoče povezati na pin PB6 oziroma PB13, signal CAN2_RX pa na pin PB5 ali PB12. Pri krmilniku USART3 izbire pravzaprav nimamo, saj lahko signal USART3_TX vežemo le na pin PB10, signal USART3_RX pa na pin PB11. Iz čisto praktičnih razlogov smo se odločili, da pri krmilniku bxCAN2 signal CAN2_TX povežemo na pin PB13, signal CAN2_RX pa na pin PB12. V

tem primeru imamo namreč vse potrebne podakovne signale na eni strani strani razvojne plošče, kar nam nekoliko olajša fizičen priklop.

Določiti je potrebno še nekatere lastnosti pinov. S parametrom `GPIO_Mode_AF` povemo, da gre za digitalen V/I pin. Izhode s parametrom `GPIO_OType_PP` nastavimo kot t. i. "Push-pull output", vendar kasneje s parametrom `GPIO_Pu Pd_NOPULL` onemogočimo t. i. "Pull-up" oziroma "Pull-down" delovanje. Za konec s parametrom `GPIO_Speed_50MHz` nastavimo frekvenco izhodov na 50 MHz.

Končna konfiguracija splošno namenskih V/I vrat je za oba krmilnika podobna, poglejmo si primer za krmilnik `USART3`:

```
1  /* GPIO USART3 Configuration */
2  GPIO_InitStructureUSART.GPIO_Pin = GPIO_Pin_10 | GPIO_Pin_11;
3  GPIO_InitStructureUSART.GPIO_Mode = GPIO_Mode_AF;
4  GPIO_InitStructureUSART.GPIO_OType = GPIO_OType_PP;
5  GPIO_InitStructureUSART.GPIO_PuPd = GPIO_PuPd_NOPULL;
6  GPIO_InitStructureUSART.GPIO_Speed = GPIO_Speed_50MHz;
7  GPIO_Init(GPIOB, &GPIO_InitStructureUSART);
8
9  /* Connect USART3 to AF; USART_TX = PB10, USART_RX = PB11 */
10 GPIO_PinAFConfig(GPIOB, GPIO_PinSource10, GPIO_AF_USART3);
11 GPIO_PinAFConfig(GPIOB, GPIO_PinSource11, GPIO_AF_USART3);
```

Na tem mestu poskrbimo še za inicializacijo svetlečih diod LED ter gumba `BUTTON_USER` na plošči STM32F4Discovery. Prve bomo uporabili za obveščanje uporabnika o stanju naprave, z drugim pa bo uporabnik pri vklopu oziroma resetiranju naprave določil njen režim delovanja. Funkcija `STM_EVAL_LEDInit` poskrbi za inicializacijo svetlečih diod, `STM_EVAL_LEDOn` pa za njihov vklop. Inicializirali bomo modro (`LED6`), oranžno (`LED3`) ter zeleno (`LED4`) svetlečo diodo. Podobno je pri gumbu `BUTTON_USER` – najprej ga moramo inicializi-

rati s funkcijo STM_EVAL_PBInit, potem pa lahko njegovo stanje prebiramo s funkcijo STM_EVAL_PBGetState. Slednja vrne število 1 v primeru, da je gumb trenutno pritisnjen, v nasprotnem primeru pa vrne števillo 0.

Izkoristili bomo dejstvo, da se ta del programske kode izvede takoj na začetku (ob priklopu napajanja oziroma takoj za resetiranjem) in z branjem stanja gumba BUTTON_USER določili režim delovanja diagnostičnega orodja. V primeru, da gumb tedaj ne bo pritisnjen, bo orodje delovalo v navadnem režimu delovanja, kjer bo pošiljalo ter prejemalo zahteve OBD-II. V nasprotnem primeru bomo orodje postavili v t. i. tihi režim delovanja, pri katerem bo sprejemalo ves promet v omrežju CAN ter ga preko vmesnika UART pošiljalo na osebni računalnik. Izbor navadnega režima bomo uporabniku sporočili s prižigom modre, izbor tihega režima pa s prižigom oranžne svetleče diode. Zeleno svetlečo diodo bomo uporabili kasneje.

```
1  /* Initialize LEDs */
2  STM_EVAL_LEDInit(LED6); // initialize BLUE led
3  STM_EVAL_LEDInit(LED3); // initialize ORANGE led
4  STM_EVAL_LEDInit(LED4); // initialize GREEN led
5
6  /* Initialize BUTTON_USER and read its state */
7  STM_EVAL_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
8  if (STM_EVAL_PBGetState(BUTTON_USER)) {
9     silentMode = 1;  // silent mode selected
10    STM_EVAL_LEDOn(LED3); // turn on orange LED
11 }
12 else { // normal mode selected
13    STM_EVAL_LEDOn(LED6); // turn on green LED
14 }
```

Konfiguracija vmesnika UART

Za ustrezno delovanje vmesnika USART3 je potrebna pravilna konfiguracija njegovih parametrov, ki pa je sicer precej enostavna. Najprej s parametrom USART_BaudRate izberemo hitrost delovanja v baudih, v našem primeru smo jo nastavili na 256000 baud. S parametrom USART_WordLength določimo dolžino besede, ki se naenkrat prenese. Kot običajno, smo tudi v našem primeru za dolžino besede vzeli 8 bitov. Parameter USART_StopBits določa število bitov, ki označujejo konec prenosa posamezne besede. Nastavili smo ga na 1, kar je najmanjša možna izbira. Pariteto, ki služi kot nekakšna zaščita pri prenosu podatkov, nastavimo s parametrom USART_Parity. V našem primeru smo jo izključili. Ravno tako smo s parametrom USART_HardwareFlow Control izključili strojni nadzor prenosa podatkov, ki sicer potrebuje še dva dodatna signala. Posledično smo tako pravzaprav vmesnik USART3 nastavili na asinhronski način delovanja in ga tako spremenili v UART. Preostala nam je tako le še nastavitev parametra USART_Mode, ki določa smer prenosa podatkov. Izbrali smo dvosmerno komunikacijo.

```
1  /* USART3 configuration */
2  /* 256000 baud,  8bit word, 1 stop bit */
3  /* no parity/hw control, rx/tx enabled */
4  USART_InitStructure.USART_BaudRate = 256000;
5  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
6  USART_InitStructure.USART_StopBits = USART_StopBits_1;
7  USART_InitStructure.USART_Parity = USART_Parity_No;
8  USART_InitStructure.USART_HardwareFlowControl =
9  USART_HardwareFlowControl_None;
10 USART_InitStructure.USART_Mode =
11 USART_Mode_Rx | USART_Mode_Tx;
12 USART_Init(USART3, &USART_InitStructure);
```

Konfiguracija vmesnika CAN

Preostane nam še konfiguracija zadnjega vmesnika, vmesnika `bxCAN2`, ki je sestavljena iz dveh delov. Najprej moramo nastaviti nekaj splošnih parametrov ter parametre za časovno sinhronizacijo z vodilom CAN. Drugi del konfiguracije predstavlja nastavitev filtra za vmesnik `bxCAN2`, s katerim bomo glede na potrebe omejili sprejem sporočil CAN.

Prvih šest parametrov opisuje funkcije krmilnika `bxCAN2`, ki so bodisi vključene bodisi izključene. Parameter `CAN_TTCM` omogoča vklop časovno proženega protokola CAN (TT-CAN). S parametrom `CAN_ABOM` lahko vključimo samodejni odklop od vodila v primeru napak, medtem ko parameter `CAN_AWUM` omogoča samodejno prebujanje krmilnika CAN. Kaj naj se zgodi v primeru neuspelega pošiljanja sporočila CAN, določimo s parametrom `CAN_NART`. Če je slednji vključen, bo krmilnik v tem primeru skušal ponovno poslati sporočilo. Naslednja dva parametra se tičeta čakalnih vrst FIFO. S prvim, `CAN_RFLM`, lahko v primeru polne sprejemne vrste le-to zaklenemo in nova sporočila zavržemo. Z drugim, `CAN_TXFP`, lahko nastavimo, da imajo prednost pri pošiljanju sporočila, ki so že dlje časa v vrsti za pošiljanje. V našem primeru se vsi do sedaj omenjeni parametri izključeni, ker nekaterih izmed funkcionalnosti ne potrebujemo, drugih pa ne želimo. Tipičen primer, ki spada v drugo skupino, je parameter `CAN_RFLM`. Pri poizvedbah OBD-II želimo vedno imeti na voljo karseda aktualne podatke, zato ne želimo teh podatkov pri polni sprejemni vrsti zavreči, temveč jih v njo umestiti namesto starih.

Način delovanja krmilnika `bxCAN2` določimo s parametrom `CAN_Mode`. Vrednost tega parametra je seveda odvisna od uporabnikove izbire režima delovanja diagnostičnega orodja. V primeru, da je uporabnik izbral normalen režim delovanja diagnostičnega orodja, krmilnik `bxCAN2` nastavimo na normalen način delovanja. Analogno se odzovemo v primeru izbire tihega načina delovanja. Pri tihem načinu delovanja je signal `CAN_TX` galvansko odklopljen, kar pomeni, da krmilnik `bxCAN2` ne more pošiljati sporočil CAN, kakor tudi ne more potrjevati prejetih paketov (ACK). Ta način je torej pri-

meren za pasivno spremljanje dogajanja v omrežju CAN, saj ne moremo vplivati na ostale člane omrežja.

Nastaviti moramo tudi parametre za časovno sinhronizacijo krmilnika `bxCAN2`. Naše diagnostično orodje bo vedno povezano z omrežjem CAN hitrosti 500 kbps. Že v prejšnjih poglavjih smo namreč povedali, da so krmilne enote za nadzor motorne elektronike vedno povezane na omrežje CAN visoke hitrosti (500 kbps), običajno pa le te podpirajo standard OBD-II. Vrednost sinhronizacijskega segmenta določimo s parametrom `CAN_SJW` in je običajno (in zato tudi v našem primeru) vedno 1. Točko vzorčenja določimo s parametrom `CAN_BS1` in je v rangu med 1 in 16, točko prenosa pa s parametrom `CAN_BS2`. Zaloga vrednosti slednje je med 1 in 6. Parameter `CAN_Prescaler` določa vrednost preddelilnika, za hitrost 500 kbps moramo izbrati vrednost 4. Za določitev vrednosti parametrov `CAN_BS1` ter `CAN_BS2` si pomagamo s tehničnim priročnikom RM0090 [31]. Najprej moramo frekvenco vodila AHB1 (42 Mhz), na katero je priključen krmilnik `bxCAN2` deliti z željeno hitrostjo vodila CAN:

$$A = \frac{f_{AHB1}}{R_{CAN}} = \frac{42\,\mathrm{MHz}}{500\,\mathrm{kbps}} = 84 \tag{5.1}$$

V naslednjem koraku dobljeno vrednost vstavimo v enačbo (5.2):

$$CAN_{BS1} + CAN_{BS2} = \frac{A}{CAN_{Prescaler}} - 1 = \frac{84}{4} - 1 = 20 \tag{5.2}$$

Izračunali smo, da je vsota vrednosti točke vzorčenja ter točke prenosa enaka 20. V skladu s standardom ISO 11898 imamo tako na voljo dve rešitvi. Za prvo rešitev izberemo vrednost točke vzorčenja 16, vrednost točke prenosa pa 4. Pri drugi rešitvi lahko izberemo vrednost točke vzorčenja 15 in vrednost točke prenosa 5. Obe izmed rešitev delujeta, v našem primeru smo se odločili za prvo. Vrednost parametra `CAN_BS1` smo torej nastavili na 16, vrednost parametra `CAN_BS2` pa na 4. Na koncu še preverimo, ali se je krmilnik CAN pravilno inicializiral. Za uspešno inicializacijo je poleg pravilne konfiguracije

potreben še dodaten pogoj: ustrezno priključen oddajno/sprejemni modul. Uporabniku zato uspešno inicializacijo krmilnika CAN sporočimo s prižigom zelene svetleče diode.

```
1  /* bxCAN2 configuration */
2  CAN_InitStructure.CAN_TTCM = DISABLE;
3  CAN_InitStructure.CAN_ABOM = DISABLE;
4  CAN_InitStructure.CAN_AWUM = DISABLE;
5  CAN_InitStructure.CAN_NART = DISABLE;
6  CAN_InitStructure.CAN_RFLM = DISABLE;
7  CAN_InitStructure.CAN_TXFP = DISABLE;
8  if (silentMode) { // silent mode
9     CAN_InitStructure.CAN_Mode = CAN_Mode_Silent;
10 else {  // normal mode
11    CAN_InitStructure.CAN_Mode = CAN_Mode_Normal;
12 }
13 CAN_InitStructure.CAN_SJW = CAN_SJW_1tq;
14 CAN_InitStructure.CAN_BS1 = CAN_BS1_14tq;
15 CAN_InitStructure.CAN_BS2 = CAN_BS2_6tq;
16 CAN_InitStructure.CAN_Prescaler = 4;
17
18 if (CAN_Init(CAN2, &CAN_InitStructure)) {
19    STM_EVAL_LEDOn(LED4); // turn on green LED
20 }
```

V drugem delu konfiguracije krmilnika `bxCAN2` poskrbimo za nastavitev filtriranja pri sprejemu sporočil CAN. V normalnem režimu delovanja diagnostičnega orodja želimo sprejemati le odgovore na naše poizvedbe OBD-II. Sporočila, ki nosijo druge informacije, nas v tem primeru ne zanimajo in zato želimo, da jih krmilnik `bxCAN2` zavrže. Kako to doseči? Spomnimo se, da imajo sporočila CAN, ki nosijo odgovor na poizvedbe OBD-II, oznake okvira od 0x7E8 naprej. Filtriranje bomo torej izvedli na podlagi oznake okvira. Odločili smo se za nekoliko ohlapnejše pravilo in dopustili sprejem sporočil z

oznako okvira med 0x700 ter 0x7FF.

Krmilnik `bxCAN2` ima na voljo 14 različnih filtrov, z oznakami med 14 ter 27. S parametrom `CAN_FilterNumber` določimo, kateri filter želimo nastaviti. Izbrali smo kar zadnjega, torej tistega z oznako 27. Način, s katerim dosežemo filtriranje sporočil, določimo s parametrom `CAN_FilterMode`. V našem primeru smo se odločili za izbor filtriranja na podlagi mask. Obseg filtra nastavimo s parametrom `CAN_FilterScale`. Izbrali smo filter z obsegom 32 bitov. Parametra `CAN_FilterIdHigh` ter `CAN_FilterMaskIdHigh` določata, kako naj se filter obnaša. S prvim nastavimo začetno oznako okvira, sporočila z nižjo oznako od te se zavržejo, drugi pa predstavlja t. i. masko, ki na podlagi ujemanja bitov ustrezno filtrira sporočila. Bit z vrednostjo 1 na določenem mestu označuje, da se mora dohodno sporočilo CAN na tem bitu ujemati z bitom, ki je nastavljen v začetni oznaki okvira. Nasprotno pa bit z vrednostjo 0 na določenem mestu pove, da vrednost bita dohodnega sporočila na tem mestu ni pomembna. Z nastavitvijo obeh parametrov, `CAN_FilterIdHigh` ter `CAN_FilterMaskIdHigh` na vrednost 0x0700 smo torej dosegli, da krmilnik `bxCAN2` sprejme le sporočila, ki imajo oznako okvira med 0x700 ter 0x7FF. Potreben je še logični pomik te vrednosti za pet mest v levo zaradi filtriranja sporočil po standardu CAN 2.0A. V primeru, ko naše orodje deluje v tihem režimu, oba parametra preprosto nastavimo na vrednost 0x0000 in tako dovolimo sprejem vseh sporočil. Konfiguracijo zaključimo z nastavitvijo dveh dodatnih parametrov. Prvi, `CAN_FilterFIFOAssignment`, določa v katero izmed treh dohodnih vrst FIFO naj prihajajo sporočila. V našem primeru smo se odločili kar za prvo dohodno vrsto. Filter moramo nato le še vključiti z nastavitvijo parametra `CAN_FilterActivation`.

```
/* bxCAN2 configuration */
/* CAN2 filter configuration */
CAN_FilterInitStructure.CAN_FilterNumber = 27;
CAN_FilterInitStructure.CAN_FilterMode = CAN_FilterMode_IdMask;
CAN_FilterInitStructure.CAN_FilterScale = CAN_FilterScale_32bit;
if (silentMode) { // accept all traffic
   CAN_FilterInitStructure.CAN_FilterIdHigh = 0x0000;
   CAN_FilterInitStructure.CAN_FilterMaskIdHigh = 0x0000;
}
else { // accept only OBD-II traffic
   CAN_FilterInitStructure.CAN_FilterIdHigh = 0x0700 << 5;
   CAN_FilterInitStructure.CAN_FilterMaskIdHigh = 0x0700 << 5;
}
CAN_FilterInitStructure.CAN_FilterFIFOAssignment = CAN_FIFO0;
CAN_FilterInitStructure.CAN_FilterActivation = ENABLE;
```

Pošiljanje in sprejemanje sporočil CAN

Opravili smo vse potrebne konfiguracije, ki so pogoj za pošiljanje in sprejemanje sporočil CAN. Preden to tudi storimo, si moramo ogledati parametre, ki določajo zgradbo sporočila CAN.

Za pošiljanje sporočila CAN različice 2.0A je potrebno definirati 12 parametrov. Najprej s parametrom `StdId` določimo oznako okvira sporočila CAN. Že iz imena parametra lahko sklepamo, da določamo oznako običajnega (2.0A) okvira. V našem primeru smo oznako okvira nastavili na 0x7E0, torej na oznako, na katero se odziva krmilna enota motorne elektronike. S parametrom `RTR` krmilniku `bxCAN2` povemo, ali pošiljamo podatkovni ali daljinski okvir. Zahteve OBD-II vedno pošiljamo kot podatkovne okvire CAN. Parameter `IDE` določimo različico (2.0A oziroma 2.0B) oznake okvira sporočila CAN, ki ga želimo poslati. Naše diagnostično orodje (in tudi standard OBD-II) uporablja le običajne (2.0A) oznake okvira sporočila CAN. Določiti moramo tudi število podatkovnih bajtov, ki jih želimo poslati. To storimo z

nastavitvijo parametra `DLC`. Standard OBD-II od nas zahteva, da je ta parameter nastavljen na 8, kar je tudi največje število podatkovnih bajtov, ki jih lahko pošljemo z enim sporočilom CAN. S preostalimi osmimi parametri določimo dejansko vsebino sporočila CAN. Ti parametri so oblike `Data[x]`, pri čemer je vrednost `x` med 0 in 7, vsak izmed njih pa predstavlja po 1 bajt podatkov. Vsebina teh podatkovnih bajtov je pri poizvedbah OBD-II natančno določena in smo jo že navedli v tabeli 4.2. Parameter `Data[0]` je tako nastavljen na 0x02, parameter `Data[1]` na 0x01, vrednost parametra `Data[2]` pa je odvisna od vrste PID. Vrednosti ostalih parametrov so nastavljene na 0x00.

Sprejemanje sporočil CAN je sicer ravno nasprotna operacija pošiljanju, vendar je obenem tudi zelo podobna. Oznako okvira prejetega sporočila izvemo z uporabo parametra `StdId`, število podatkovnih bajtov, ki jih prejeto sporočilo nosi, pa s pomočjo parametra `DLC`. Do podatkovnih bajtov sporočila CAN ravno tako dostopamo z že znanimi parametri `Data[x]`. Zaradi nasprotne operacije vrednosti omenjenih parametrov ne nastavljamo, ampak shranjujemo.

Pošiljanje in sprejemanje sporočil UART

Vsebino sporočil CAN, ki jih kot odgovor na zahteve OBD-II prejmemo v krmilnik `bxCAN2`, moramo z uporabo protokola UART poslati na razvojno ploščo Arduino Mega 2560, kjer s pomočjo zaslona LCD nato prikažemo vrednosti parametrov OBD-II. Z iste razvojne plošče pa ravno tako preko protokola UART sprejemamo informacije o tem, kateri parameter OBD-II želi uporabnik imeti prikazan na zaslonu LCD.

Tako pošiljanje kot tudi prejemanje sta pri protokolu UART zelo enostavna. Pošiljanje izvršimo z uporabo funkcije `USART_SendData()`, pri čemer moramo slednji podati dva parametra. S prvim določimo, kateri krmilnik USART naj mikroprocesor STM32F407VGT6 uporabi pri pošiljanju, zato

smo ga podali kot `USART3`. Kot drugi parameter pa podamo sporočilo, ki ga želimo poslati. Sporočilo, ki se prenese po povezavi UART, je dolgo 1 B (8 bitov) in je predstavljeno s kodo ASCII. Sporočila CAN so bistveno daljša od enega bajta, zato jih prenašamo v več korakih. Vrednosti sporočila CAN (oznaka okvira, podatkovni bajti) najprej razdelimo na posamezne števke, nato jih spremenimo v njihovo vrednost ASCII ter vsako posebej pošljemo. Med posameznimi sporočili pošiljemo vrednost 0x0A, ki po standardu ASCII označuje novo vrstico, v našem primeru pa služi kot delilnik med posameznimi sporočili.

Prejemanje sporočil je v našem primeru še enostavnejše. Funkcija `USART_ReceiveData()` nam vrne sprejeti bajt podatkov. Potrebuje le en parameter, s katerim določimo, kateri krmilnik USART naj mikroprocesor STM32F407VGT6 uporabi pri sprejemu (v našem primeru torej ponovno `USART3`). Ker je en bajt dolg 8 bitov, lahko z njim določimo $2^8 = 256$ različnih vrednosti, kar bistveno presega število različnih parametrov OBD-II, ki jih naša naprava lahko prikazuje.

Končni program

Na koncu moramo vse opisane korake združiti v celoto, tako da bo razvojna plošča STM32F4Discovery periodično pošiljala poizvedbe OBD-II in nato pridobljene podatke posredovala plošči Mega 2560.

Program, ki je naložen na plošči STM32F4Discovery, se začne izvajati takoj po priklopu napajanja. Najprej se izvedejo konfiguracije: konfiguracija urinih signalov, konfiguracija splošno namenskih V/I vrat, konfiguracija vmesnika UART, konfiguracija vmesnika CAN ter konfiguracija filtrov vmesnika CAN. Nato program vstopi v neskočno zanko, ki je sestavljena iz štirih funkcij. S funkcijo `CAN_TxMessage()` najprej preverimo, če smo prek krmilnika UART prejeli zahtevo za spremembo prikaza parametra OBD-II, takoj zatem pa pošljemo sporočilo CAN s poizvedbo OBD-II. V naslednjem koraku

z uporabo funkcije `CAN_RxMessage()` sprejmemo sporočilo CAN, ki vsebuje odgovor na našo poizvedbo OBD-II. S funkcijo `USART_RxCANMessage()` nato prejete podatke le še pretvorimo v ustrezno obliko in preko vmesnika UART pošljemo do plošče Mega 2560. Preden ponovno vstopimo v zanko, pokličemo še funkcijo `Delay()`, ki poskrbi za zakasnitev med posameznimi poizvedbami.

```
1  int main(void) {
2
3     /* Initialize Clocks */
4     RCC_Configuration();
5     /* Initialize GPIO */
6     GPIO_Configuration();
7     /* Initialize USART */
8     USART_Configuration();
9     /* Initialize CAN */
10    CAN_Configuration();
11    /* Initialize CAN Reception Filter */
12    CAN_FilterConfiguration();
13
14    /* Query and answer in a loop */
15    while(1) {
16       /* Transfer CAN message */
17       CAN_TxMessage();
18       /* Receive CAN message */
19       CAN_RxMessage();
20       /* Send contents of CAN message over USART */
21       USART_RxCANMessage();
22       /* Delay between requests */
23       Delay(2000000);
24    }
25
26 }
```

5.3.2 Razvoj programske opreme za Mega 2560

Izbira razvojnega okolja

Za platformo Arduino je (poleg nekaterih drugih) na voljo kar istoimensko razvojno okolje [38], ki je daleč najbolj popularno in smo ga uporabili tudi v našem primeru. Gre za razvojno okolje, izpeljano iz tistega, ki je bilo razvito za odprtokodno platformo za vizualizacijo podatkov Processing. Okolje je bilo osnovano predvsem z mislijo na enostavno uporabo in kot tako ne omogoča naprednih funkcij, kot smo jih vajeni iz okolja TrueSTUDIO. Tako recimo ne vsebuje orodja za razhroščevanje, kar v neki meri otežuje razvoj programske opreme, vendar je to pomanjkljivost mogoče premostiti z uporabo nekaterih prijemov pri pisanju programske kode. Omogočena je podpora za jezika C in C++, pri čemer se razvojno okolje opira na programsko knjižnico Wiring. Težav z okoljem Arduino pri razvoju programske opreme za ploščo Mega 2560 zaradi relativne enostavnosti nalog, ki jih plošča opravlja, nismo imeli.

Konfiguracija vmesnika UART in zaslona LCD

Konfiguriranje V/I naprav je na plošči (in hkrati tudi na celoti platformi Arduino) zelo enostavno. Za konfiguracijo vmesnika UART je potrebno poklicati le eno samo funkcijo, `Serial.begin()`, ter ji kot parameter podati željeno hitrost povezave v baudih. Vrednost tega parametra v našem primeru znaša torej 256000.

Za konfiguracijo zaslona LCD s krmilnikom Hitachi HD44780 poskrbimo z dvema korakoma. S prvim korakom mikroprocesorju ATmega2560 povemo, na katere pine smo zaslon LCD priključili. To s storimo s klicem funkcije `LiquidCrystal()`, ki ji podamo 6 parametrov, ta pa nam nato ustvari spremenljivko tipa `LiquidCrystal`. Prva dva parametra določata, na katerih mestih sta signala `RS` in `E`, ostali štirje pa kje so signali `DB4`, `DB5`, `DB6` in `DB7`. Signal `RS` našega zaslona LCD je priključen na pin `D8`, signal `E` pa na pin `D9`.

Signali od DB4 do DB7 so priključeni na pine med D4 in D7. V drugem koraku je potrebno določiti velikost zaslona LCD. Določimo jo s klicem funkcije `lcd.begin()`, pri čemer se sklicujemo na spremenljivko tipa `LiquidCrystal`, ki smo jo ustvarili v prejšnjem koraku. Funkcija potrebuje dva parametra – prvi določa število stolpcev, drugi pa število vrstic, ki jih je zaslon LCD zmožen prikazati. V našem primeru uporabljamo zaslon LCD s 16 stolpci ter dvema vrsticama.

```
1 LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
2
3 void setup() {
4   Serial.begin(256000);
5   lcd.begin(16, 2);
6 }
```

Pošiljanje in sprejemanje sporočil UART

Tudi ta korak je zelo enostaven. Sporočila UART pošiljamo s pomočjo funkcije `Serial.write()`, ki ji kot parameter podamo željeno sporočilo. Preko protokola UART pošiljamo kar kodo PID, ki je dolga 1 bajt, ter izberemo željen parameter OBD-II. Sprejemanje sporočil UART poteka s pomočjo funkcije `Serial.read()`. Funkcija ne potrebuje parametrov, vrača pa nam posamezen sprejet bajt. Posamezno sporočilo, ki ga preko vmesnika UART sprejemamo od razvojne plošče STM32F4Discovery, je dolgo 19 bajtov, ki jim sledi še dodaten bajt z vrednostjo 0x0A. Slednji, kot že vemo, služi kot razmejilnik med sporočili. Podatke zato v zanki sprejemamo dokler ne sprejmemo natanko 19 bajtov, med katerimi ni bajta z vrednostjo 0x0A. Funkcija `Serial.available()` nam vrača število bajtov v medpomnilniku (bajtov, ki jih nismo še sprejeli). Iz podatkov, ki jih sprejmemo, izračunamo parameter PID. Implementirali smo izračun vseh parametrov PID načina 0x01 iz tabele 4.1.

```
1  void receiveCANData() {
2     index = 0;
3        while(index < 19) {
4           while(Serial.available() > 0) {
5              char ASCII = Serial.read();
6
7              if (ASCII == '\n' && index < 19) {
8              index = 0;
9           }
10          else {
11             data[index] = ASCII;
12             index++;
13             if (index == 19) {
14                break;
15             }
16          }
17       }
18    }
19 }
```

Izpis podatkov na zaslonu LCD

Pri izpisu podatkov na zaslon LCD si pomagamo s tremi funkcijami. Funkcija `lcd.print()` na zaslonu LCD izpiše niz znakov, ki ji ga podamo kot parameter. S funkcijo `lcd.setCursor()` določimo položaj kurzorja na zaslonu. Ta določa, kje na zaslonu naj se niz znakov začne izpisovati. Funkcija je določena z dvema parametroma, pri čemer prvi predstavlja številko stolpca, drugi pa številko vrstice. Tretja funkcija, `lcd.clear()`, izbriše prikazane podatke na zaslonu LCD in kurzor premakne v levi zgornji kot (v točko [0,0]).

```
1 case 0x0D: // 0x0D = vehicle speed
2     lcd.clear();
3     lcd.print("Hitrost vozila:");
4     lcd.setCursor(0,1);
5     lcd.print(PIDvalue);
6     lcd.setCursor(4,1);
7     lcd.print("km/h");
8     delay(150);
9     break;
```

Določanje stanja gumbov

Na "ščitu" LCD Keypad Shield for Arduino je poleg dvovrstičnega zaslona LCD tudi 6 gumbov. Vsak izmed njih je po principu deljenja napetosti preko različnih uporov povezan na analogni pin `A0` na plošči Mega 2560. S pomočjo funkcije `analogRead()`, ki ji kot parameter podamo številko pina (v našem primeru torej 0), izvemo, ali je kateri izmed gumbov pritisnjen. Funkcija vrača vrednost med 0 in 1023, pri čemer vrednost 0 predstavlja napetost 0 V, vrednost 1023 pa napetost 5 V. Za naše potrebe smo uporabili le gumba `UP` ter `DOWN`. Prvi je s pinom `A0` povezan preko 330 Ω upora, drugi pa preko 620 Ω upora. Funkcija `analogRead(A0)` zato ob pritisku gumba `UP` vrne vrednost okoli 145, ob pritisku gumba `DOWN` pa vrednost okoli 330. Z zaznavanjem sprememb teh vrednosti lahko spreminjamo med prikazovanjem različnih parametrov OBD-II.

```
BUTTONvalue = analogRead(0);

if ((BUTTONvalue > 120) && (BUTTONvalue < 160)) { // button UP
   if (BUTTONindex == 9) {
      BUTTONindex = 0;
   }
   else {
      BUTTONindex++;
   }
}
if ((BUTTONvalue > 310) && (BUTTONvalue < 340)) { // button DOWN
   if (BUTTONindex == 0) {
      BUTTONindex = 9;
   }
   else {
      BUTTONindex--;
   }
}
```

Končni program

Zadnjo nalogo spet predstavlja združevanje opisanih delov v celoto. Kot pri plošči STM32F4Discovery se tudi pri plošči Mega 2560 program začne izvajati takoj po priklopu napetosti. Najprej se izvedeta konfiguraciji vmesnika UART ter zaslona LCD, nato po vstopimo v neskočno zanko. Če je medpomnilnik krmilnika CAN prazen, najprej v zanki počakamo, da v njega prispe vsaj en podatek (bajt). Tik za tem izvedemo pet zaporednih funkcij. S funkcijo `receiveCANData()` začnemo sprejemati podatke preko vmesnika UART. Po končanem sprejemu jih s pomočjo funkcije `convertData()` pretvorimo v prvotno obliko, torej v obliko, v kateri so bili, preden smo jih poslali s plošče STM32F4Discovery. Končno vrednost parametra OBD-II izračunamo s klicem funkcije `calculatePID()` in jo nato z uporabo funkcije `displayPID()` izpišemo na zaslonu LCD. Za konec le še s funkcijo `selectPID()` preverimo,

ali je uporabnik z ustreznimi gumbi želel spremeniti parameter OBD-II, ki se prikazuje, zatem pa se vrnemo na začetek zanke.

```
void loop() {
   while(Serial.available() == 0){
      // wait for data
   }
   receiveCANData();
   convertData();
   calculatePID();
   displayPID();
   selectPID();
}
```

5.4 Preizkus delovanja

Potem, ko smo zaključili z razvojem programske opreme, smo preizkusili delovanje orodja za diagnostiko vozil. Preizkus je potekal v dveh fazah – naprej smo preverili delovanje orodja v normalnem načinu, kasneje pa še v tihem.

5.4.1 Normalni način

Delovanje orodja v normalnem načinu smo preizkusili na treh osebnih vozilih različnih proizvajalcev:

- Citroen C3 1.4 VTi (leto izdelave 2010),
- Mercedes A-Class A160 CDI (leto izdelave 2008),
- Volkswagen Passat 2.0 TDI (leto izdelave 2006).

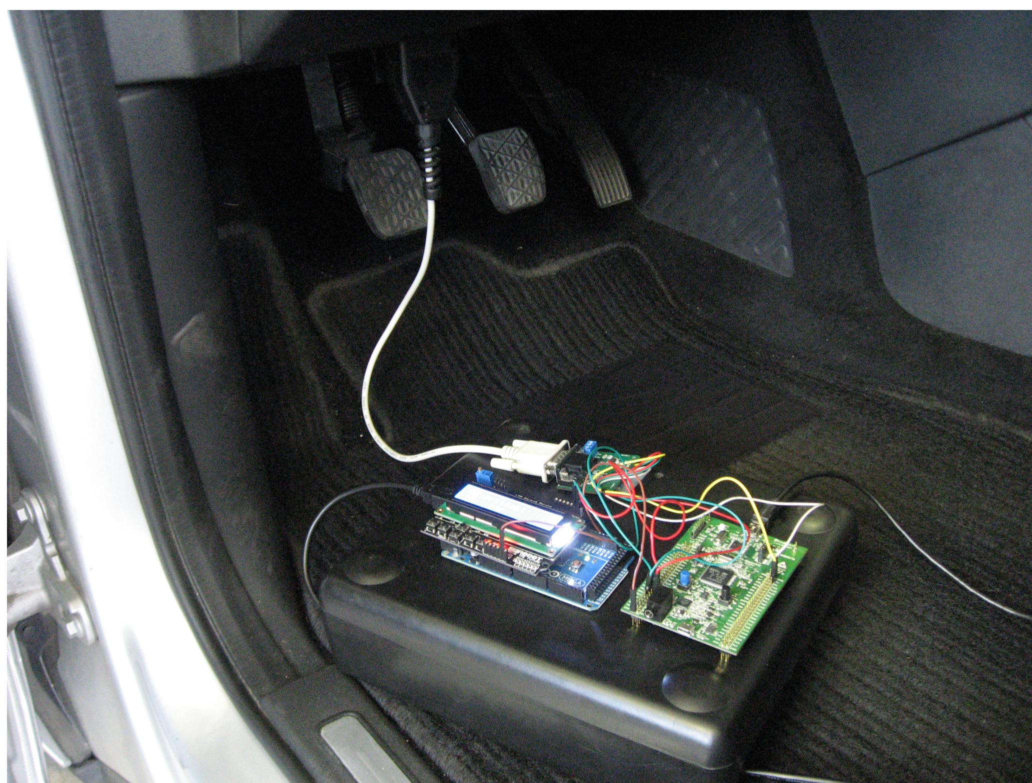
Vsa našteta vozila seveda podpirajo standard OBD-II in uporabljajo signalni protokol ISO 15765 (CAN). Prvo vozilo (Citroen C3) je opremljeno

z bencinskim motorjem. Po priklopu orodja za diagnostiko je slednje takoj (kot je tudi privzeto nastavljeno) prikazalo podatek o številu vrtljajev motorja. Brezhibno je deloval tudi prikaz podatkov o obremenitvi motorja, hitrosti vozila, temperaturi vstopnega zraka, temperaturi hladilne tekočine, masnem pretoku zraka, položaju pedala za plin ter o napetosti obeh kisikovih sond. Prikaz podatka o tlaku goriva v skupnem vodu ni deloval, kar pa ni presenetljivo, saj vozilo ne uporablja tehnologije skupnega voda.

Ostali dve vozili sta opremljeni z dizelskim motorjem. Pri prvem, Mercedes A-Class, je bilo mogoče spremljati podatke o številu vrtljajev motorja, obremenitvi motorja, temperaturi hladilne tekočine, masnem pretoku zraka ter tlaku goriva v skupnem vodu. Ostalih parametrov ni bilo mogoče spremljati. Podobno je bilo pri vozilu Volkswagen Passat, a je bilo za razliko od prejšnjega mogoče dodatno spremljati tudi podatke o položaju pedala za plin in temperaturi vstopnega zraka. Čemu razlika? Omenili smo že, da proizvajalci vozil niso dolžni implementirati vseh načinov delovanja oziroma parametrov OBD-II. Vseeno pa so se podatki, ki so bili na voljo, izkazali za točne. Primerjali smo jih namreč s podatki, pridobljenimi z uporabo nekaterih profesionalnih diagnostičnih orodij, in ugotovili, da odstopanja ni. Preizkus delovanja v normalnem načinu je prikazan na sliki 5.6.

5.4.2 Tihi način

Preizkus delovanja tihega načina smo izvedli na vozilu Fiat Punto (leto izdelave 2004). Vozilo sicer za diagnostiko OBD-II ne uporablja signalnega protokola ISO 15765 (CAN), vendar to v tem primeru niti ni pomembno. Bistveno je dejstvo, da ima nekatere krmilne enote, ki med seboj komunicirajo preko vodila CAN. Med njimi tako najdemo tudi krmilno enoto motorne elektronike in krmilno enoto merilnikov. Signale vodila CAN najdemo kar na običajnih mestih v priključku OBD-II, kar se je izkazalo za zelo dobrodošlo lastnost, saj ni bilo potrebe po spremembi oziroma izdelavi novega priključka. Za preizkus smo morali orodje za diagnostiko povezati še z osebnim računalnikom, kamor je orodje z uporabo povezave UART pošiljalo podatke. Za spremljanje podat-



Slika 5.6: Preizkus diagnostičnega orodja v vozilu Mercedes A-Class.

kov smo uporabili orodje HyperTerminal, ki je vgrajeno v operacijski sistem Windows. Preizkus se je izkazal za uspešnega, saj smo prejemali sporočila z omrežja CAN tekoče in brez težav, seveda pa si z njimi nismo mogli kaj prida pomagati. Njihov pomen je bil popolna neznanka, saj je definiran s strani proizvajalca in ne sledi nobenemu izmed znanih standardov. V praksi je tako tihi način seveda uporaben le v primerih, ko vemo, kaj podatki, ki potujejo po vodilu CAN, sploh pomenijo.

Poglavje 6

Sklep

V diplomski nalogi smo temeljito preučili delovanje protokola CAN. Spoznali smo njegove značilnosti in posebnosti, kakor tudi njegovo aplikativno uporabo. Ogledali smo si tudi standard za diagnostiko vozil OBD-II in opisali, kakšne informacije lahko z njegovo uporabo pridobimo. S poznavanjem protokola CAN in standarda OBD-II smo si ustvarili jasno sliko o delovanju, komunikaciji in diagnostiki elektronskih krmilnih enot v sodobnih vozilih. Zasnovali smo cenovno ugodno orodje za diagnostiko krmilnih enot. Na primeru snovanja strojnega dela orodja za diagnostiko smo se pravzaprav naučili, iz kakšnih komponent je sestavljen običajen član v omrežju CAN ter kako je nanj priključen. Pri razvoju programske opreme smo se uspešno spoprijeli s konfiguracijo krmilnika CAN. Slednja predstavlja bistveni del pri razvoju programske opreme za poljubno aplikacijo krmilnika CAN. Skupek teh korakov nas je pripeljal do enostavnega in učinkovitega orodja za diagnostiko vozil.

Razvito orodje deluje dobro in je uporabno tudi v praksi. Nič pa ni tako dobro, da ne bi moglo biti še boljše. Pravzaprav smo v skladu s to mislijo razvijali diagnostično orodje že od samega začetka. Manevrskega prostora za izboljšave je zato relativno veliko. Če začnemo na strojnem nivoju, je najbolj očiten korak v smeri izdelave ohišja za diagnostično orodje, kar bi izboljšalo

njeno prenosljivost in olajšalo uporabo. Dodatno izboljšavo bi predstavljala sprememba načina napajanja razvojnih plošč. V priključku OBD-II je namreč nepretrgoma na voljo napetost 12 V, ki jo lahko z uporabo dodatne komponente pretvorimo v napetost 5 V. Tako bi odpravili (sicer zamudno) potrebo po priklapljanju napajanja preko priključkov USB. Šli bi lahko celo bistveno dlje in ukinili eno izmed najdražjih komponent – razvojno ploščo Mega 2560. Ta korak bi zahteval minimalne posege na strojnem nivoju, potrebno bi bilo le priključiti zaslon LCD neposredno na razvojno ploščo STM32F4Discovery, kar bi imelo dve posledici na programskem nivoju. Prva (pravzaprav ugodna) je ta, da bi odpadla potreba po komunikaciji preko povezave UART v normalnem načinu delovanja. Nekoliko manj dobrodošla bi bila druga posledica, saj bi zahtevala razvoj programske knjižnice za uporabo zaslona LCD.

Tudi na programskem nivoju je možna vrsta izboljšav. Lahko bi omogočili branje in izbris diagnostičnih kod napak (DTC), kar bi pravzaprav vzelo le malo dodatnega truda. Dodali bi lahko tudi način za periodično beleženje parametrov, ki bi serviserju omogočil kasnejšo analizo delovanja krmilnih enot. Nenazadnje bi lahko implementirali tudi standarde za diagnostiko, ki so lastni posameznim proizvajalcem vozil. Omejitev pri tem koraku nam v resnici predstavlja le dostop do podatkov o teh standardih. Omenjene izboljšave bi lahko izvedli tako z dodatnim razvojem obstoječe programske opreme za razvojno ploščo STM32F4Discovery, kot tudi z razvojem namenske aplikacije za osebni računalnik. Potencial orodja je zelo velik – lahko bi dejali, da nas pri nadaljnem razvoju omejuje le domišljija.

Literatura

[1] D. Paret, "Concepts of bus access and arbitration," v *Multiplexed Networks for Embedded Systems: CAN, LIN, FlexRay, Safe-by-Wire...* Chichester: J. Wiley & Sons, 2007.

[2] D. Paret, "Error processing and management," v *Multiplexed Networks for Embedded Systems: CAN, LIN, FlexRay, Safe-by-Wire...* Chichester: J. Wiley & Sons, 2007.

[3] D. Paret, "Historical context of CAN," v *Multiplexed Networks for Embedded Systems: CAN, LIN, FlexRay, Safe-by-Wire...* Chichester: J. Wiley & Sons, 2007.

[4] (2012) Controller Area Network (CAN) Overview: CAN Applications. Dostopno na: `http://www.ni.com/white-paper/2732/en`

[5] (2012) Vehicle Networks: Controller Area Network (CAN). Dostopno na: `http://www.sti-innsbruck.at/fileadmin/documents/vn-ws0809/02-VN-CAN.pdf`

[6] (2012) ISO 11898-1:2003: Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling. Dostopno na: `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=33422`

[7] (2012) CAN Specification: Version 2.0. Dostopno na: `http://www.semiconductors.bosch.de/media/pdf/canliteratur/can2spec.pdf`

[8] D. Paret, "CAN 2.0B," v *Multiplexed Networks for Embedded Systems: CAN, LIN, FlexRay, Safe-by-Wire...* Chichester: J. Wiley & Sons, 2007.

[9] D. Paret, "Definitions of the CAN Protocol: 'ISO 11898-1'," v *Multiplexed Networks for Embedded Systems: CAN, LIN, FlexRay, Safe-by-Wire...* Chichester: J. Wiley & Sons, 2007.

[10] D. Paret, "Introduction," v *Multiplexed Networks for Embedded Systems: CAN, LIN, FlexRay, Safe-by-Wire...* Chichester: J. Wiley & Sons, 2007.

[11] (2012) Communication in Distributed Embedded Systems: Bit Encoding. Dostopno na `http://ti.tuwien.ac.at/rts/teaching/courses/esevobachelor/unterlagen/vortragsfolien-ws10/ese_6_communication_in_DES.pdf`

[12] W. Voss, "Error detection," v *A Comprehensible Guide to Controller Area Network.* Amherst: Copperhill Technologies Corporation, 2005.

[13] D. Paret, "Errors: Their Intrinsic Properties, Detection and Processing," v *Multiplexed Networks for Embedded Systems: CAN, LIN, FlexRay, Safe-by-Wire...* Chichester: J. Wiley & Sons, 2007.

[14] D. Paret, "Medium, implementation and physical layers in CAN," v *Multiplexed Networks for Embedded Systems: CAN, LIN, FlexRay, Safe-by-Wire...* Chichester: J. Wiley & Sons, 2007.

[15] D. Paret, "The range of media and the types of coupling to the network," v *Multiplexed Networks for Embedded Systems: CAN, LIN, FlexRay, Safe-by-Wire...* Chichester: J. Wiley & Sons, 2007.

[16] D. Paret, "High speed CAN, from 125 kbit s^{-1} to 1 Mbit s^{-1}: ISO 11898-2," v *Multiplexed Networks for Embedded Systems: CAN, LIN, FlexRay, Safe-by-Wire...* Chichester: J. Wiley & Sons, 2007.

[17] D. Paret, "Low speed CAN, from 10 to 125 kbit s^{-1}," v *Multiplexed Networks for Embedded Systems: CAN, LIN, FlexRay, Safe-by-Wire...* Chichester: J. Wiley & Sons, 2007.

[18] (2012) The History of On-Board Diagnostics. Dostopno na: `http://www.nicoclub.com/archives/the-history-of-on-board-diagnostics.html`

[19] (2012) General Motors Computerized Vehicle Control Systems: A Short History. Dostopno na: `http://www.tomboynton.com/GMnetworks.pdf`

[20] (2012) On-Board Diagnostics: History. Dostopno na: `http://en.wikipedia.org/wiki/On-board_diagnostics#History`

[21] (2012) On-Board Diagnostics: OBD-II diagnostic connector. Dostopno na: `http://en.wikipedia.org/wiki/On-board_diagnostics#OBD-II_diagnostic_connector`

[22] (2012) Diagnostic Connector Equivalent to ISO/DIS 15031-3:December 14, 2001. Dostopno na: `http://standards.sae.org/j1962_200204/`

[23] (2012) OBD-connector-pinout. Dostopno na: `http://upload.wikimedia.org/wikipedia/commons/3/38/OBD-connector-pinout.png`

[24] W. Zimmerman, R. Schmidgall, "On-Board-Diagnose OBD nach ISO 15031 / SAE J1979," v *Bussysteme in der Fahrzeugtechnik : Protokolle und Standards.* Wiesbaden: Vieweg, 2006.

[25] W. Zimmerman, R. Schmidgall, "Überblick OBD Diagnosedienste," v *Bussysteme in der Fahrzeugtechnik : Protokolle und Standards.* Wiesbaden: Vieweg, 2006.

[26] W. Zimmerman, R. Schmidgall, "Auslesen des Fehlerspeichers und vor Steuergerätewerten," v *Bussysteme in der Fahrzeugtechnik : Protokolle und Standards.* Wiesbaden: Vieweg, 2006.

[27] (2012) OBD-II PIDs: CAN (11-bit) Bus format - Query. Dostopno na: http://en.wikipedia.org/wiki/OBD-II_PIDs#Query

[28] (2012) OBD-II PIDs: CAN (11-bit) Bus format - Response. Dostopno na: http://en.wikipedia.org/wiki/OBD-II_PIDs#Response

[29] (2012) OBD-II PIDs: CAN (11-bit) Bus format - Standard PIDs. Dostopno na: http://en.wikipedia.org/wiki/OBD-II_PIDs#Standard_PIDs

[30] (2012) STM32F407VG: High-performance and DSP with FPU, ARM Cortex-M4 MCU with 1 Mbyte Flash, 168 MHz CPU, Art Accelerator, Ethernet. Dostopno na: http://www.st.com/internet/mcu/product/252140.jsp

[31] (2012) RM0090 Reference manual: Controller area network (bxCAN). Dostopno na: http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/REFERENCE_MANUAL/DM00031020.pdf

[32] (2012) MCP2551: High Speed CAN Transceiver. Dostopno na: http://ww1.microchip.com/downloads/en/devicedoc/21667d.pdf

[33] (2012) Arduino Mega 2560. Dostopno na: http://www.arduino.cc/en/Main/ArduinoBoardMega2560

[34] (2012) Arduino LCD Keypad Shield. Dostopno na: http://www.droboticsonline.com/index.php/arduino-lcd-keypad-shield.html

[35] (2012) STM32F4DISCOVERY: STM32F4 high-performance discovery board. Dostopno na: http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATA_BRIEF/DM00037955.pdf

[36] (2012) Atollic TrueSTUDIO. Dostopno na: http://www.atollic.com/index.php/truestudio

[37] (2012) STM32F4DISCOVERY: User Manual UM 1472. Dostopno na: http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/USER_MANUAL/DM00039084.pdf

[38] (2012) Arduino Development Environment. Dostopno na: http://arduino.cc/en/Guide/Environment