OBD2-DATAN SIIRTO MATKAPUHELIMEEN

Hannu Turtiainen

Opinnäytetyö
Marraskuu 2012

Elektroniikan koulutusohjelma
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖN
KUVAILULEHTI

Tekijä(t) TURTIAINEN, Hannu-Tapani	Julkaisun laji Opinnäytetyö	Päivämäärä 12.11.2012
	Sivumäärä 64 (6)	Julkaisun kieli suomi
	Luottamuksellisuus ( ) saakka	Verkkojulkaisulupa myönnetty (X)
Työn nimi OBD2-DATAN SIIRTO MATKAPUHELIMEEN		
Koulutusohjelma Elektroniikka		
Työn ohjaaja(t) PIETIKÄINEN, Kalevi KOTKANSALO, Jouko		
Toimeksiantaja(t) MECTRONIC OY MÄKINEN, Kimmo		

Tiivistelmä

Opinnäytetyön tilaaja oli Mectronic Oy. Työn tarkoituksena oli tutustua ajoneuvojen vikadiagnostiikkaan, rakentaa ajoneuvojen diagnostiikkaa tulkitseva lukulaite ja ohjelmoida matkapuhelimelle sovellus, joka yhdistää matkapuhelimen Bluetoothilla lukulaitteeseen.

Työssä tutkittiin OBD2-järjestelmän toimintaa ja sen yhdistämistä matkapuhelimeen. Prototyyppilaite valmistettiin ja sen toimivuus testattiin. Lukulaitteelle suunniteltiin piirilevy massatuotantoa ajatellen. Android-käyttöjärjestelmällä toimivalle matkapuhelimelle ohjelmoitiin sovellus. Laitteen ja sovelluksen yhteensopivuus testattiin muutamalla tapauksella.

Laite ja sovellus toimivat hyvin yhteen. Sovellus yhdistyy lukulaitteeseen ja näyttää käyttäjälle tietoa ajoneuvon antureista. Käyttäjä voi myös halutessaan lukea ajoneuvon vikakoodit.

Avainsanat (asiasanat)

OBD2, Bluetooth, Android, Java, Piirilevysuunnittelu, Vikakoodi, Vikadiagnostiikka

Muut tiedot

Author(s)
TURTIAINEN, Hannu-Tapani

Type of publication
Bachelor´s Thesis

Date
12.11.2012

Pages
64 (6)

Language
Finnish

Permission for web publication
(X)

Title
OBD2 DATA TRANSFER TO MOBILE PHONE

Degree Programme
Electronics Engineering

Tutor(s)
PIETIKÄINEN, Kalevi
KOTKANSALO, Jouko

Assigned by
MECTRONIC LLC
MÄKINEN, Kimmo

Abstract

The Bachelor´s thesis was assigned by Mectronic LLC. The purpose of the thesis was to study fault diagnostics of modern vehicles, design and build a vehicle diagnostic data interpreter and to program an application for a mobile phone which connects to the interpreter via Bluetooth.

The thesis includes research for fundamentals of OBD2 system and a way to connect to it through a mobile phone. A prototype interpreter was made and tested. A printed circuit board for mass production purposes was also designed for the interpreter. An application was made for Android operating system. Compatibility of the interpreter and the application was tested with a few cases.

The interpreter and the application worked well together. The application connects to the interpreter and presents sensor data from the vehicle to an end user. The end user can also read diagnostic trouble codes from the vehicle.

Keywords
OBD2, Bluetooth, Android, Java, Printed circuit board design, Diagnostic trouble code, Fault diagnostics

Miscellaneous

SISÄLTÖ

LYHENTEET **4**

1 OPINNÄYTETYÖN LÄHTÖKOHTA JA TAVOITE **5**

2 ON-BOARD DIAGNOSTICS **6**

2.1 YLEISTÄ **6**

2.2 PROTOKOLLAT **7**

2.2.1 Yleistä 7

2.2.2 ISO15765-4 (CAN-väylä) 7

2.2.3 ISO14230-4 (KWP2000, Keyword Protocol 2000) 7

2.2.4 ISO9141-2 8

2.2.5 SAE J1850 VPW 8

2.2.6 SAE J1850 PWM 8

2.3 LIITIN **8**

2.4 OBD2-VIESTINNÄN ARKKITEHTUURI **10**

2.4.1 PCI-tavu ja CAN-väylän viestikehykset 11

2.5 DATATAVUT **12**

2.5.1 Moodit 12

2.5.2 PID 13

2.6 VIKAKOODIEN LUKEMINEN **13**

2.7 OBD2-LUKULAITTEEN JA VALMISTAJAKOHTAISEN DIAGNOSTIIKKATYÖKALUN ERO **16**

3 LUKULAITTEISTO **16**

3.1 YLEISTÄ **16**

3.2 PROTOKOLLATULKIT **16**

3.3 PIIRIKAAVIOT **17**

3.3.1 Komponenttien valinta 17

3.3.2 Lukulaitteen piirikaavio, ELM327 18

3.3.3 Lukulaitteen piirikaavio, virransyöttö 19

3.3.4 Lukulaitteen piirikaavio, liitännät 19

3.3.5 Lukulaitteen piirikaavio, Bluetooth 21

3.4 PIIRILEVYT **21**

3.4.1 Yleistä 21

3.4.2 Prototyyppipiirilevy 22
3.4.3 Tuotantopiirilevy 23
3.5 LUKULAITTEEN KONFIGUROINTI 24
3.6 LUKULAITTEEN SUUNNITTELUN YHTEENVETO 24
4 SOVELLUS 25
4.1 SOVELLUKSEN TOIMINTA 25
4.2 OHJELMOINTI 27
4.2.1 Ohjelman käynnistys 27
4.2.2 Bluetooth-laitelista 28
4.2.3 Bluetooth-palvelu 31
4.2.4 Bluetooth-datan vastaanotto 33
4.2.5 Yhteyden luominen ajoneuvon järjestelmän kanssa (bus_init- ja reset_device-metodit) 34
4.2.6 Anturitiedon hakeminen 36
4.2.7 Anturitiedon käyttö 37
4.2.8 Kierroslukumittari 38
4.2.9 Layout 42
4.2.10 Vikakoodien luku ja esitys 43
4.2.11 Parannusehdotuksia 47
4.2.12 Sovelluksen yhteenveto 49
5 OHJELMAN JA LUKULAITTEEN TESTAUS 49
6 POHDINTA 52
LÄHTEET 54
LIITTEET 56
LIITE 1. LUKULAITTEEN PIIRIKAAVIO, ELM327 56
LIITE 2. LUKULAITTEEN PIIRIKAAVIO, VIRRANSYÖTTÖ 57
LIITE 3. LUKULAITTEEN PIIRIKAAVIO, LIITÄNNÄT 58
LIITE 4. LUKULAITTEEN PIIRIKAAVIO, BLUETOOTH 59
LIITE 5. MASSATUOTANTOPIIRILEVY, PÄÄLLYSKERROS 1:1 60
LIITE 6. MASSATUOTANTOPIIRILEVY, POHJAKERROS 1:1 60
LIITE 7. MASSATUOTANTOPIIRILEVY, KASAUSKAAVIO 61

KUVIOT

KUVIO 1. Data Link Connector-liitin, A-tyyppi 9
KUVIO 2. Data Link Connector-liitin, B-tyyppi 9
KUVIO 3. OBD2-viestin rakenne, ei CAN 11
KUVIO 4. OBD2-viestin rakenne, CAN 11
KUVIO 5. Prototyyppi 23
KUVIO 6. Sovellus: Bluetooth lupapyyntö 25
KUVIO 7. Sovellus: Bluetooth-laitelistaus 26
KUVIO 8. Sovellus: Valikko 26
KUVIO 9. Sovellus: Vikakoodien esitys 27
KUVIO 10. Sovellus: Ei vikakoodeja 27
KUVIO 11. Sovellus: Sovelluksen pikakuvake 27
KUVIO 12. Sovellus: Perustoimintatila, kierroslukumittari punarajalla 42
KUVIO 13. Testitapaus 1: Samsung Galaxy S2 ja Audi A4 2002 49
KUVIO 14. Testitapaus 2: Samsung Galaxy S2 ja Audi A5 2011 50
KUVIO 15. Testitapaus 3: Samsung Galaxy S2 ja Audi A4 2008 50
KUVIO 16. Testitapaus 4: HTC Desire HD ja Audi A4 2002 51
KUVIO 17. Testitapaus 5: Samsung Galaxy S3 ja Toyota Avensis 2002 52

TAULUKOT

TAULUKKO 1. Protokollan määrittäminen DLC-liittimen nastoista 9
TAULUKKO 2. DTC 1. merkki, järjestelmä 14
TAULUKKO 3. DTC 2. merkki, alaryhmä 14
TAULUKKO 4. DTC 3. merkki, rakenneryhmä 15

LYHENTEET

AT	Attention, AT-komento
ADT	Android Development Tools, ohjelmointityökalut Android-käyttöjärjestelmälle
CAN	Controller Area Network, CAN-väylä, automaatioväylä
CRC	Cyclic Redundancy Check, tarkisteavaimen luontiin tarkoitettu tiivistealgoritmi
DTC	Diagnostic Trouble Code, vikakoodi
DLC	Data-Link Connector, diagnostiikkajärjestelmän liitin
DP	Density-independent pixel, määrittää objektin koon riippumatta matkapuhelimen näytön suhteellisesta tarkkuudesta
DSUB	D-subminiature, yleismallin d-liitin
ECU	Electronic Control Unit, sähköinen ohjausjärjestelmä
EOBD	European On-board Diagnostics, OBD2:n eurooppalainen versio
ISO	International Standards Organization, standardointijärjestö
JTAG	Join Test Action Group, IEEE 1149.1-standardi, testausportti
MAC	Media Access Control address, verkkosovittimen yksilöity osoite
MIL	Malfunction Indicator Lamp, Check Engine Light, moottorin vikavalo
OBD	On-board Diagnostics, ajoneuvodiagnostiikkajärjestelmä
PCI	Protocol Control Information, PCI-tavu, CAN-väylän aputavu
PIC	Peripheral Interface Controller, mikroprosessorityyppi
PID	Parameter Identification, OBD2-viestinnässä käytettyjä tarkennusosoitteita
PPI	Pixels per Inch, pikseliä/tuuma, näytön suhteellinen tarkkuus
PWM	Pulse Width Modulation, pulssinleveysmodulaatio
RTS	Request to Send, RS232-sarjaliikenteen kontrollointiviesti
SAE	Society of Automotive Engineers, standardointijärjestö
SDK	Software Development Kit, Eclipse SDK, sovelluksen luontiin käytetty ohjelma, kääntäjä
UUID	Universally Unique Identifier, tiedonsiirtoyhteyden muodostuksessa käytetty tunnistusavain
VPW	Variable Pulse Width, muuttuva pulssinleveys

1 OPINNÄYTETYÖN LÄHTÖKOHTA JA TAVOITE

Ajoneuvoissa on nykyään valtavasti elektroniikkaa. Anturit ja laitteet ovat jo niin älykkäitä, että ajoneuvosta saadaan erittäin paljon reaaliaikaista informaatiota sekä vikadiagnostiikkaa. Tämä informaatio on yleensä ollut vain korjaamojen käytettävissä. Nykyisin yksityiskäyttöön on saatavilla monenlaisia lukulaitteita ja sovelluksia on lähes kaikille laitteille ja käyttöjärjestelmille.

Tein työharjoitteluna tutkimuksen markkinoilla olevista OBD2-lukulaitteista, niiden sovelluksista ja vaihtoehdoista, kuinka ajoneuvojen vikadiagnostiikka saadaan ihmisille helpommin lähestyttäväksi. Tutkimuksessa otettiin paljon kantaa alan kannattavuuteen. Opinnäytetyön aihe muodostui tutkimuksen tuloksista.

Opinnäytetyön aiheeksi ja tavoitteeksi asetettiin OBD2-järjestelmään tutustuminen, lukulaitteen prototyypin rakennus sekä matkapuhelinsovelluksen mallin ohjelmointi. Pääpaino opinnäytetyössä oli käytännönsovelluksen toteuttamisessa. Lukulaitteelle suunniteltiin erillinen piirilevy massatuotannon ehdoilla. Sovellus ohjelmoitiin Android-käyttöjärjestelmälle sen suosion takia ja siksi, että sovellus pystyttiin helposti testaamaan. Sovellus ja lukulaite testattiin usealla eri testitapauksella.

Lukulaitepuolella tavoitteena oli tehdä toimiva laite, jolla sovellusta päästäisiin testaamaan. Aluksi päätettiin, että sovelluksen pitää olla helppokäyttöinen ja asiaan perehtymättömälle helposti lähestyttävissä. Haluttiin myös, että muutamia ajoneuvon antureista saatavia tietoja näytettäisiin käyttäjälle ja vikakoodit pystyttäisiin lukemaan.

Opinnäytetyön tilaaja oli jyväskyläläinen Mectronic Oy. Mectronic Oy on Kimmo Mäkisen yritys, joka erikoistuu ajoneuvojen mekatroniikan vikadiagnostiikkaan ja huoltoon. Erityisosaamisena on mainittava VAG-konsernin DSG- ja S-Tronic-robottivaihteistojen huollot ja korjaukset. Mectronic on pieni yritys, mutta sillä on kasvupotentiaalia ajoneuvojen mekatroniikan yleistyessä. Tulevaisuudessa Mectronicin tavoitteena on lisätä autoilijoiden tietoisuutta ajoneuvojen vikadiagnosoinnista.

2 ON-BOARD DIAGNOSTICS

2.1 Yleistä

OBD on ajoneuvojen vikadiagnostiikkaan ja raportointiin kehitetty järjestelmä. OBD kehitettiin alun perin päästöjen tarkkailuun ja niihin liittyvien ongelmien havaitsemiseen. Tarpeeksi vakavan vian ilmaantuessa OBD-järjestelmä antaa ajoneuvon mittaristoon viasta ilmoittavan valon, niin sanotun MIL-valon. Nykyisin käytössä oleva järjestelmä on OBD:n 2. versio. Valmistajien kehitystyön tuloksena OBD2-standardi kattaa nykyisin suuren määrän vikadiagnostiikkaan liittyviä ominaisuuksia ja sen kautta saadaan monenlaista tietoa ajoneuvolta. Myös ajoneuvon sähköjärjestelmät voidaan ohjelmoida nykyisin OBD-liittimen kautta. (Ivan 2006.)

OBD1 kehitettiin jo 1980-luvulla. Järjestelmän käyttöönotossa oli ongelmansa, sillä pakokaasumittauksen diagnostiikkadataa ei ollut vielä standardisoitu, joten järjestelmät olivat lähinnä merkkikohtaisia. (Heikkilä 2007.)

OBD2 julkistettiin vuonna 1994. Siinä oli useita parannuksia vanhaan järjestelmään. OBD2-standardi (SAE- J1979 ja ISO 15031) määrää diagnostiikkaportin ulkomuodon ja sen liitinnastojen järjestyksen, viestien muodon ja signalointiprotokollat. OBD2 poisti myös tarpeen lukulaitteen erilliselle virtalähteelle lisäämällä liittimeen virtanastan, joka on kytkettynä ajoneuvon akkuun. (Heikkilä 2007.)

OBD2-standardi kattaa suuren määrän yleisiä, ajoneuvon valmistajasta riippumattomia, vikakoodeja eli DTC:tä. Ajoneuvon vikaantuessa OBD2-järjestelmä tallentaa vikakoodin muistiin myöhempää tarkastelua varten. Vikakoodin ilmetessä tallentuu myös anturitietoja, jotka voidaan yhdistää tietyn vian ilmenemishetkeen vikakoodeja luettaessa. OBD2-järjestelmän kautta voidaan myös lukea reaaliaikaista tietoa ajoneuvon antureista. Yhdysvalloissa OBD2 tuli pakolliseksi kaikissa ajoneuvoissa vuodesta 1996 eteenpäin. (Heikkilä 2007.)

Euroopassa OBD2:n vastine on EOBD. Järjestelmä on käytännössä sama kuin OBD2. Eroa on päästöstandardeissa. EOBD tuli pakolliseksi kaikissa Euroopassa myydyissä

bensiinikäyttöisissä ajoneuvoissa vuonna 2001 ja dieselkäyttöisissä vuonna 2004. Myös Japanissa ja Australiassa on käytössä omat versionsa. (Heikkilä 2007.)

2.2 Protokollat

2.2.1 Yleistä

Protokolla on käsite, jolla tarkoitetaan kahden järjestelmän välistä kommunikointitapaa. OBD2-järjestelmässä on yleisesti käytössä viisi eri protokollaa. Ajoneuvovalmistajilla on myös käytössä omia protokolliaan, jotka eivät kuulu OBD2-standardiin.

Protokollat nimetään standardointikoodilla. Nimen alussa oleva tunnus viittaa standardointijärjestöön, kuten SAE eli Society of Automotive Engineers ja ISO eli International Standards Organization. Käytössä oleva protokolla määrää muun muassa signaalien jännitetasot ja yhteysnopeuden.

2.2.2 ISO15765-4 (CAN-väylä)

Uusimmat protokollat käyttävät CAN-väylää. Se on ollut pakollinen kaikissa myydyissä ajoneuvoissa vuodesta 2008 eteenpäin. CAN-väylä yhdistää ajoneuvon sähkölaitteet kahden datajohdon avulla. ISO15765-4-standardi sisältää neljä eri vaihtoehtoa tiedonsiirtoon. Erona on nopeus ja ID-bittimäärä kommunikaatioviestien alussa. (Eri Protokollat n.d; OBD-II Protocols n.d.) Alla on esitelty ISO15765-4-protokollan variaatiot.

- 11-bittinen ID, 500 kbaudin nopeus
- 29-bittinen ID, 500 kbaudin nopeus
- 11-bittinen ID, 250 kbaudin nopeus
- 29-bittinen ID, 250 kbaudin nopeus

2.2.3 ISO14230-4 (KWP2000, Keyword Protocol 2000)

ISO14230 on yleinen protokolla vuoden 2003 jälkeen ja ennen vuotta 2008 valmistetuissa ajoneuvoissa. Se käyttää OBD-liittimen K-linjaa. Protokollasta on kaksi eri versiota, jotka eroavat yhteyden muodostustavassa. Viestin maksimipituus on 255 tavua.

(Eri Protokollat n.d; OBD-II Protocols n.d.) Alla on esitelty Keyword Protocol 2000:n variaatiot.

- 5 baudin yhteyden muodostus, 10,4 kbaudin nopeus
- ”Fast Init.”, Nopea yhteyden muodostus, alusta asti käytössä 10,4 kbaudin nopeus

2.2.4 ISO9141-2

ISO9141-2 on ISO14230:aa edeltävä protokolla, joka oli Euroopassa yleisesti käytössä vuosina 2000–2004. Se käyttää OBD-liittimen K-linjaa ja mahdollisesti myös L-linjaa. ISO9141-2 käyttää 5 baudin yhteyden muodostusta, ja liikennöintinopeutena on 10,4 kbaudia. Viestin maksimipituus on 12 tavua. (Eri Protokollat n.d; OBD-II Protocols n.d.)

2.2.5 SAE J1850 VPW

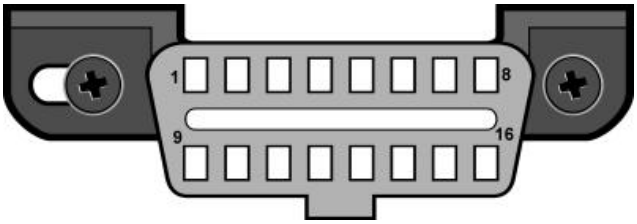
VPW-protokolla viestii yhden johdon välityksellä leveysmoduloitua jännitepulssia. Viestin maksimipituus on 12 tavua ja yhteysnopeus 10,4 kbaudia. (Eri Protokollat n.d; OBD-II Protocols n.d.)

2.2.6 SAE J1850 PWM

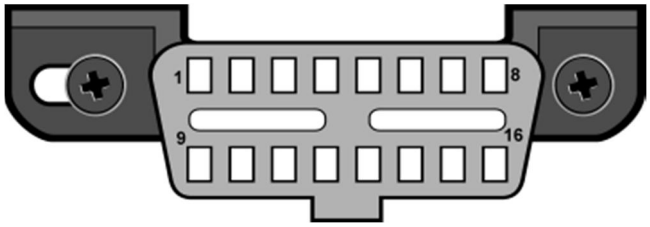
PWM-protokolla käyttää kahta johtoa, ja data saadaan johtojen pulssien erotuksesta. Viestin maksimipituus on 12 tavua ja yhteysnopeus 41,6 kbaudia. (Eri Protokollat n.d; OBD-II Protocols n.d.)

2.3 Liitin

OBD2 varten kehitettiin oma 16-pinninen DLC-liitin. Liitin on standardoitu SAE-J1962 mukaan ja se sijaitsee ajoneuvoissa usein kuljettajan jalkatilassa. Kuviossa 1 on A-mallinen DLC-liitin nastajärjestyksineen. Kuviossa 2 on B-tyypin liitin. Liitinten välillä on eroa vain kohdistuspalkki. (Which OBD-II protocol is supported by my vehicle? 2004.)



KUVIO 1. Data Link Connector-liitin, A-tyyppi (Which OBD-II protocol is supported by my vehicle? 2004.)



KUVIO 2. Data Link Connector-liitin, B-tyyppi (Which OBD-II protocol is supported by my vehicle? 2004.)

Nastat 2, 6, 7, 10, 14, 15 ovat yleisimmin käytössä olevien protokollien liitäntänastat. Valmistajat jättävät yleensä liittimestä pois ne nastat, jotka eivät ole kyseisessä ajoneuvossa käytössä. Tämän perusteella voikin päätellä, taulukon 1 mukaisesti, mikä OBD-protokolla on käytössä. (jfrank 2008.)

TAULUKKO 1. Protokollan määrittäminen DLC-liittimen nastoista

Protokollat	Nasta 2	Nasta 6	Nasta 7	Nasta 10	Nasta 14	Nasta 15
J1850 PWM	käytössä	-	-	käytössä	-	-
J1850 VPW	käytössä	-	-	-	-	-
ISO9141	-	-	käytössä (K)	-	-	mah-dollinen (L)
CAN	-	käytössä	-	-	käytössä	-

Nastat 1, 3 ja 14 ovat vanhempien valmistajakohtaisten järjestelmien liitäntänastoja. Nastassa 16 on akkujännite, nastassa 4 on runkomaa ja nasta 5 on tarkoitettu signaalimaaksi. Nastat 8, 9, 12 ja 13 ovat varalla uusia järjestelmiä varten. (Which OBD-II protocol is supported by my vehicle? 2004.)

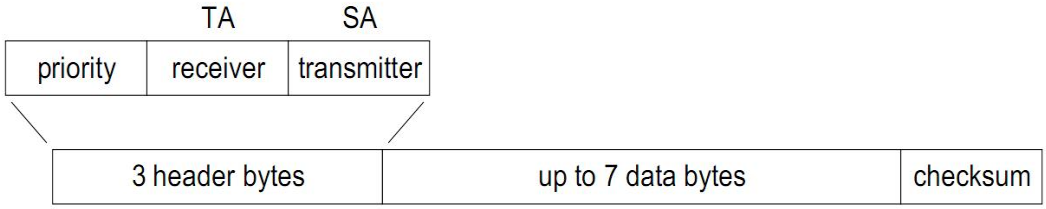
2.4 OBD2-viestinnän arkkitehtuuri

OBD2-järjestelmä on suunniteltu hyvin joustaviksi, sillä monet ajoneuvon laitteet kommunikoivat toisiinsa sen kautta. Tämän seurauksena OBD2 viestintä tarvitsee viestin alkuun osoitetietoa. OBD2 ja sähköiset ohjausyksiköt eli ECU:t viestivät samalla arkkitehtuurilla. Ohjausyksiköltä on aina pyydettävä tietoa, koska se ei lähetä tietoa jatkuvana virtana (streaming). (ELM327 OBD to RS232 Interpreter n.d, 7.)

OBD2-viesti voidaan jakaa kolmeen osaan kuten kuviosta 3 nähdään. Ensiksi viestissä on kolme tunnistetavua (header bytes). Tunnistetavuista ensimmäinen kertoo viestin prioriteetin. Esimerkiksi nokka-akselin asentotiedon välittäminen on tärkeämpää kuin vikakoodien lukeminen. Seuraava tunnistetavu on viestin vastaanottajan osoite ja kolmantena lähettäjän osoite. (ELM327 OBD to RS232 Interpreter n.d, 35-36.)

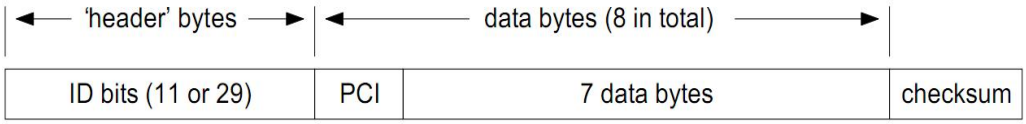
Header-tavujen jälkeen tulee maksimissaan seitsemän datatavua kerralla ennen virheentarkistustavua. Kaikkia tavuja ei tarvitse aina käyttää vaan yksikin tavu riittää usein. Pidemmät viestit lähetetään useamman viestin pakettina. (ELM327 OBD to RS232 Interpreter n.d, 35-36.)

Viimeisenä viestissä on virheentarkistustavu. SAE-protokollat käyttävät CRC-tavua ja ISO-protokollat checksum-tavua. Checksum-virheentarkistus toimii siten, että lähettäjä ja vastaanottaja laskevat kaikkien tavujen arvot yhteen ja vertaavat sitten tulosta. Checksum ei havaitse useita virheitä. CRC tiivistää polynomijaolla datapaketista tarkistusavaimen, jota käytetään virheentarkistukseen. CRC on tarkempi kuin checksum. CRC:llä havaitaan yhdenkin bitin virhe. (Tyson n.d.)



KUVIO 3. OBD2-viestin rakenne, ei CAN (ELM327 OBD to RS232 Interpreter n.d, 35.)

CAN-väylä eroaa edellisestä mallista hieman. Tunnistetavut on jaettu ID-biteiksi. Tuoreimman standardin mukaan ID-bittejä voi olla 11 tai 29 kappaletta. ID-bittien jälkeen tulee PCI-tavu. Se kertoo vastaanottavalle laitteelle tulevan viestikehyksen tyypin ja siihen liittyvää tietoa. PCI-tavun jälkeen tulee muiden protokollien tavoin seitsemän datatavua. Kuviossa 4 on esitetty CAN-väylän viestin rakenne. (ELM327 OBD to RS232 Interpreter n.d, 36.)



KUVIO 4. OBD2-viestin rakenne, CAN (ELM327 OBD to RS232 Interpreter n.d, 36.)

2.4.1 PCI-tavu ja CAN-väylän viestikehykset

CAN-väylän standardi määrittää neljä erilaista viestikehystä, jotka on esitelty alla.

- SF – Single Frame = 0
- FF – First Frame = 1
- CF – Consecutive Frame = 2
- FC – Flow Control Frame = 3

PCI-tavun neljä merkitsevintä bittiä määräävät viestikehyksen tyypin (numero listauksessa kehystyypin perässä).

Single Frame kertoo vastaanottimelle, että tulossa on vain yksi viestikehys eli maksimissaan seitsemän datatavua. PCI-tavun loput bitit kertovat datatavujen määrän. (ELM327 OBD to RS232 Interpreter n.d, 46.)

First Frame on usean viestikehyksen datapaketin ensimmäinen kehys. Usean viestikehyksen paketissa PCI-tavut tulevat peräkkäin. Esimerkiksi kahden viestikehyksen mittaisen paketin alussa on kaksi PCI-tavua, joiden perään tulevat datatavut. Kuten Single Frame-mallissa, kehystyyppi-informaation jälkeen PCI-tavussa ilmoitetaan datatavujen lukumäärä. (Leale n.d.)

Consecutive Frame on usean viestikehyksen paketissa ensimmäistä kehystä seuraavien kehysten tyyppi. Samanlaisessa kahden kehyksen esimerkissä kuin yllä, toinen PCI-tavu on Consecutive Frame-tavu. (Leale n.d.)

Flow Control Frame on vastaus First Frame informaatioon viestikehysten lukumäärästä. Lähettäjä saa sillä tiedon viestin vastaanottajalta, että se on valmis vastaanottamaan viestiä. (Leale n.d.)

2.5 Datatavut

2.5.1 Moodit

OBD2-viestin ensimmäinen datatavu (CAN-väylässä PCI-tavun jälkeen) on moodi (mode). Se kertoo millaista tietoa ohjausyksiköltä pyydetään tai vastaanotetaan. (OBDII Message Structure 2011.) OBD2-standardissa määritellään 10 moodia, mutta ajoneuvovalmistajien ei tarvitse tukea kaikkia. Alla on esitelty kaikki kymmenen moodia.

- Moodi 1: Pyydetään reaaliaikaista tietoa, esimerkiksi kierrosluku.
- Moodi 2: Pyydetään freeze frame tietoa. Samoja tietoja kuin ensimmäiselläkin moodilla, mutta tieto on tallennettu sillä hetkellä, kun vikakoodi on ilmestynyt.
- Moodi 3: Pyydetään kaikki tallennetut vikakoodit
- Moodi 4: Poistetaan kaikki tallennetut vikakoodit
- Moodi 5: Pyydetään dataa lambda-anturilta (happianturi). Ei CAN-väylälle.
- Moodi 6: Pyydetään testituloksia antureilta, joita ei valvota jatkuvasti. Happianturin tulokset haetaan Moodi 6:lla CAN-väyläisessä ajoneuvossa.

- Moodi 7: Pyydetään testiajon aikana tulleet vikakoodit
- Moodi 8: Ohjelmoidaan tai säädetään ajoneuvon laitteistoa
- Moodi 9: Pyydetään ajoneuvon tietoja, esimerkiksi runkonumero.
- Moodi 10 (A heksa): Pyydetään sellaiset pysyvät vikakoodit, jotka poistuvat vain vian korjaamalla.
  (OBDII Message Structure 2011.)

Ohjausyksiköltä tietoa pyydettäessä moodi on ensimmäinen datatavu ja desimaaleina numero yhdestä kymmeneen. Kun yksikkö vastaa, on tähän lukuun lisätty desimaaleina 64. Esimerkiksi, jos pyydetään reaaliaikaista dataa, on mooditavu 1, ja kun yksikkö vastaa, mooditavu on 65. (OBDII Message Structure 2011.)

2.5.2 PID

Toinen datatavu viestissä on PID-tavu. PID-tavu on numero, millä viitataan siihen tietoon, mitä halutaan. PID-tavu on mooditavun tavoin läsnä myös ohjausyksikön vastauksessa, mutta siihen ei lisätä mitään. OBD2-standardi määrittää suuren joukon PID:jä. Ajoneuvovalmistajien ei kuitenkaan tarvitse tukea kaikkia. Valmistajilla voi myös olla omia valmistajakohtaisia PID:jä. Kaikissa moodeissa ei käytetä PID:jä. Esimerkiksi moodit 3 ja 4 (vikakoodien luku ja poisto) eivät käytä PID:jä. Jokaisessa moodissa, jossa käytetään PID:jä, PID 0 kertoo ajoneuvon tuetut PID:t. (OBD-II PIDs 2010.) PID-tavun jälkeiset datatavut ovat vastaanotettua tietoa.

2.6 Vikakoodien lukeminen

Tässä luvussa on kuvattu OBD2-standardin mukaisen vikakoodin tunnistaminen. Muilla ajoneuvon ohjausyksiköillä on omat vikakoodijärjestelmänsä.

Vikakoodi koostuu viidestä merkistä. Yksi vikakoodi on kahden tavun mittainen. Ensimmäisen tavun kaksi ensimmäistä bittiä määräävät vikakoodin ensimmäisen kirjaimen, joka kuvaa sitä ajoneuvon järjestelmää, josta vikakoodi on peräisin. Taulukosta 2. nähdään mahdolliset järjestelmät.

TAULUKKO 2. DTC 1. merkki, järjestelmä (OBD2 Codes Explained n.d.)

Bitti A7 (merkitsevin)	Bitti A6	Järjestelmä
0	0	P – Moottori/ voimansiirto (powertrain)
0	1	C – Alusta (chassis)
1	0	B – Kori (body)
1	1	U – Muut järjestelmät (network)

Loput neljä merkkiä vikakoodissa ovat numeroita. Toinen merkki saadaan ensimmäisen tavun kolmannesta ja neljännestä bitistä, ja se kuvaa alaryhmää taulukon 3. mukaisesti.

TAULUKKO 3. DTC 2. merkki, alaryhmä (OBD2 Codes Explained n.d.)

Bitti A5	Bitti A4	Alaryhmä
0	0	0 – SAE vikakoodi (standardin mukainen)
0	1	1 – Valmistajan vikakoodi
1	0	2 – SAE vikakoodi
1	1	3 – Valmistajan vikakoodi

Loput neljä bittiä ensimmäisestä tavusta kertovat vikakoodin kolmannen merkin taulukon 4. mukaisesti. Kolmas merkki kertoo rakenneryhmän.

TAULUKKO 4. DTC 3. merkki, rakenneryhmä (OBD2 Codes Explained n.d.)

Bitti A3	Bitti A2	Bitti A1	Bitti A0	Rakenneryhmä
0	0	0	0	0 – SAE-varattu
0	0	0	1	1 – Päästöihin liittyvä vikakoodi
0	0	1	0	2 – Polttoaineen suihkutus
0	0	1	1	3 – Sytytysjärjestelmä
0	1	0	0	4 – Päästöihin liittyvä vikakoodi
0	1	0	1	5 – Ajoneuvon nopeus/ joutokäynnin säätö
0	1	1	0	6 – Moottorinohjausyksikkö ja ulostu-losignaalit
0	1	1	1	7 – Vaihteisto
1	0	0	0	8 – Vaihteisto
1	0	0	1	9 – SAE-varattu

Neljäs ja viides merkki saadaan toisen tavun biteistä samoin kuin kolmas merkki. Ne tarkentavat vikakoodin tiettyyn järjestelmäkomponenttiin ja ongelmaan. Jokaisessa rakenneryhmässä voi olla 99 vikakoodia (01-99). (Lehtinen n.d.)

Esimerkiksi vikakoodi P0300 kuuluu moottori/voimansiirto-järjestelmään, SAE-vikakoodi alaryhmään ja sytytysjärjestelmän rakenneryhmään. P0300 on kaikille mer-

keille yhteinen vikakoodi ja se kertoo, että yhdessä tai useammassa sylinterissä on tapahtunut sytytyskatko. (ELM327 OBD to RS232 Interpreter n.d, 31.)

2.7 OBD2-lukulaitteen ja valmistajakohtaisen diagnostiikkatyökalun ero

Kuten mainittua, OBD2 on yleinen standardi ja ajoneuvovalmistajilla voi olla omia järjestelmiään integroituna siihen. OBD2-lukulaite keskustelee vakiona vain moottorinohjausyksikön kanssa. Sillä ei voi yhdistää muihin OBD2-standardin ulkopuolisiin ECU:hin ellei tunnistetavuja viestin alussa muuteta. Valmistajilla on omat osoitetietonsa muihin ECU:hin. Yleismallin lukulaite ei siis suoraan voi olla täysin yleispätevä ajoneuvon vikojen etsinnässä. (What's the difference between VCDS and an OBD-II Scan-Tool? n.d; ELM327 OBD to RS232 Interpreter n.d, 37.)

3 LUKULAITTEISTO

3.1 Yleistä

OBD2-sarjaliikenne ei suoraan ole käyttökelpoista tietoa matkapuhelimelle. Tieto pitää muuttaa esimerkiksi standardinmukaiseksi RS232-sarjaliikennedataksi. Vanhemmilla protokollilla, kuten ISO9141-2, voidaan käyttää yksinkertaista ja analogista optoerottimilla toimivaa niin sanottua tyhmäkaapelia kuten Planetfall-sivuston projektissa on esitetty (Noxon 2009.). Kyseinen laite toimii hyvin useissa vanhemmissa ajoneuvoissa, mutta se ei toimi esimerkiksi CAN-väylän kanssa. Uudempien ajoneuvojen kanssa kyseinen kaapeli on siis hyödytön. Jotta OBD2-lukija toimisi kaikissa ajoneuvoissa, tarvitaan älykäs protokollatulkkipiirillä toimiva lukija.

3.2 Protokollatulkit

Kaikkia protokollia ja niiden variaatioita tukevia valmiita piirejä on muutamia. Valmistajia on muun muassa ELM Electronics, OBD Solutions ja OBDPros. Monipuolisimmat ja uusimmat piirit heiltä ovat ELM 327 v.1.4b (ELM327 v1.4b OBD to RS232

Interpreter 2012.), OBD Solutions STN1110 (STN1110: Multiprotocol OBD to UART Interpreter IC 2012.) ja OBDPro v1.4 (OBD Interpreter IC´s 2012). ELM327-piiri on markkinoilla tunnettu ja paljon käytetty piiri. STN1110 on vastaava, mutta selkeästi edullisempi ja valmistajan mukaan suvereeni vaihtoehto. OBDPro on ELM327:n suora kopio ja yksittäin tilattuna edullisin. Muitakin tulkkeja on olemassa, kuten Freescale MCZ33290EF, mutta kyseinen piiri ei tue esimerkiksi CAN-väylää. Piirit ovat kohtalaisen hyvin integroituja. Oheiskomponentteja tarvitaan melko vähän ja säästöjä komponenteissa voidaan tehdä vähentämällä tuettuja protokollia. Jokainen mainituista piireistä on toteutettu Microchip Technologyn PIC-mikroprosessorilla.

Kyseisillä protokollatulkeilla voidaan siis tehdä yleismallinen OBD-lukija. Lukijoilla päästään käsiksi OBD2-standardin mukaisiin ominaisuuksiin useimmilla ajoneuvoilla ja merkistä riippumatta.

3.3 Piirikaaviot

Opinnäytetyönä valmistettu lukulaite käyttää ELM327-piiriä ja sen piirikaavio on lähes identtinen piirin referenssipiirikaavion kanssa, sillä se varmistaa parhaiten laitteen toimivuuden useiden eri protokollien kanssa. Erona referenssipiirikaavioon on sarjaliikenneportin toteutus. Yhteys lukulaitteeseen luodaan Bluetoothilla, sillä se on selkeästi helpoin ja edullisin vaihtoehto yhdistää lukulaite matkapuhelimeen. Tämä kävi ilmi tutkiessani työharjoitteluna eri liitäntätapoja.

Piirikaaviot ja piirilevyt suunniteltiin Mentor Graphics PADS 9.4-ohjelmistolla. Käytössä oli PADS Logic, PADS Layout sekä PADS Router. Apuna piirilevyn suunnittelussa oli Hannu Tikkasen kirja Piirilevysuunnitteluopas 2.

3.3.1 Komponenttien valinta

Komponenttien valinnassa tärkeintä oli hinta. Kaikki osat pitää saada mahdollisimman halvalla. Hannu Tikkasen (2004) mukaan yksi hyvä keino saada pudotettua tuotteen hintaa on suunnitella laite volyymikomponenteilla. Vastuksien ja kondensaattorien arvot pidetään standardiarvoissa ja resistanssiltaan tai kapasitanssiltaan vastaavia komponentteja käytetään paljon. (Tikkanen 2004, 29.)

Kondensaattorit voitiin valita mahdollisimman halvoiksi, sillä ELM327-piiriä käytetään 4 MHz:n kiteellä. Nopeus on melko hidas, joten kondensaattorien laatu ei vaikuta lukulaitteen toimintaan (Tikkanen 2004, 141.).

Lukulaitteen hintaa vähentää myös se, että kaikki osat ovat matalia ja miltei kaikki niistä voidaan latoa piirilevylle koneellisesti. Lisäksi mikään komponentti ei vaadi jäähdytystä. (Tikkanen 2004, 142-143.)

3.3.2 Lukulaitteen piirikaavio, ELM327

ELM327-piiri on kohtalaisen hyvin integroitu piiri, jolla on paljon ominaisuuksia. Perustapauksissa monia piirin nastoista ei edes tarvita.

Nasta 1 on uudelleenkäynnistyspainike eli reset. Lukijan tapauksessa piirilevylle ei tarvita omaa resetiä vaan uudelleenkäynnistys tehdään aina ohjelmallisesti. Nasta kytketään siis käyttöjännitteeseen. (ELM327 OBD to RS232 Interpreter n.d, 3)

Nasta 2:sta käytetään ajoneuvon akkujännitteen mittauksessa. Jännite skaalataan 0-5 V:n signaaliksi, joten nastaan tarvitaan jännitteenjakoa. Tarvittava vastuksien suhde on vakiona 4,7, mutta se voidaan tarvittaessa muuttaa piiriä konfiguroimalla. Lukulaitteeseen valittiin 10 kΩ:n sekä 47 kΩ:n vastukset. (ELM327 OBD to RS232 Interpreter n.d, 3)

Nasta 5 on muistitoiminnon kytkemisnasta. Kun nasta on kytketty käyttöjännitteeseen, muistaa piiri viimeksi käytetyn protokollan nopeuttaen näin seuraavaa yhdistämistä (ELM327 OBD to RS232 Interpreter n.d, 3). Oletusarvoisesti laitetta käytetään saman ajoneuvon kanssa, joten muistitoiminto pidetään käytössä.

Nasta 6:lla voidaan säätää sarjaliikenneyhteyden nopeutta. Kun nasta on maadoitettu, sarjaliikenteen nopeus on 9,6 kbaudia. Jos nasta on käyttöjännitteessä, nopeus on 38,4 kbaudia tai ohjelmallisesti asetettu. (ELM327 OBD to RS232 Interpreter n.d, 3.) Prototyypissä nasta kytkettiin maahan, mutta massatuotantopiirilevyssä nasta kytketään käyttöjännitteeseen.

Nasta 7 määrittää linefeed-tilan. Kun nasta on käyttöjännitteessä, sarjaliikenneviestin loppuun tulee rivin lopetus- ja seuraava rivi-merkit. Jos nasta on maadoitettu, tulee viestin loppuun vain rivin lopetus-merkki. (ELM327 OBD to RS232 Interpreter n.d, 3.) Lukulaitteessa nasta kytkettiin käyttöjännitteeseen, jolloin viestejä on helpompi lukea Terminal-ohjelmalla. Merkit tulee kuitenkin huomioida sovellusta ohjelmoitaessa.

Nasta 15 on nolla-aktiivinen RTS-sisääntulo. Sen avulla voidaan keskeyttää OBD2-linjan toiminta uutta käskyä varten. Lukulaitteessa nasta on kytketty käyttöjännitteeseen tarpeettomana. (ELM327 OBD to RS232 Interpreter n.d, 4.)

Nastat 25-28 ovat ledien käytössä. Ledien avulla saadaan visuaalista informaatiota siitä, että laite toimii. Nasta 25 on sarjaliikennevastaanotto, nasta 26 sarjaliikennelähetys, nasta 27 OBD2-vastaanotto ja nasta 28 OBD2-lähetys led. (ELM327 OBD to RS232 Interpreter n.d, 5).

Piirikaaviota suunnitellessa päätettiin, että piirin unitilaa ei oteta käyttöön. Tyypillisesti lukulaitetta ei jätetä ajoneuvoon kiinni, vaan se poistetaan aina käytön jälkeen. Lisäksi unitila olisi lisännyt komponenttikustannuksia. Paras komponentti on pois jätetty komponentti. Liitteessä 1 on esitelty mikro-ohjainosio piirikaaviosta.

3.3.3 Lukulaitteen piirikaavio, virransyöttö

Virransyöttö on toteutettu hyvin yksinkertaisesti regulaattoreilla. Akkujännite ohjataan suojausdiodin ja suodatuksen kautta 5 V:n regulaattorille. Regulaattorilla on 1 A:n ulosanti, sillä Bluetooth vie osansa ja CAN-väylä saattaa tarvita virtaa pulsseina. 5 V:n regulaattorin ulostulossa on myös päälläolo-led. 3,3 V:n regulaattoria tarvitaan Bluetooth-moduulia varten. Liiteessä 2 on esitelty virransyöttöosio piirikaaviosta.

3.3.4 Lukulaitteen piirikaavio, liitännät

CAN-väylän signaali muutetaan ELM327-piirille sopivaksi Microchipin MCP2551-tulkkipiirin avulla. Tulkkipiirejä on muitakin, mutta MCP2551 on kohtalaisen edullinen ja hyvin saatavilla. Signaalin muuttaminen onnistuu myös passiivikomponenteilla,

mutta se ei juurikaan tule edullisemmaksi ja tulkkipiiri on huomattavasti varmempi vaihtoehto. (ELM327 OBD to RS232 Interpreter n.d, 65).

ISO-protokollissa on käytössä usein vain yksi signaalijohdin (K-line), mutta harvemmin käytetty kaksijohtiminen vaihtoehto on myös mukana (K-line + L-line).

R15 ja R16 muokkaavat K-linjan jännitetasot piirille sopivaksi. 1-tilassa jännite on suurempi kuin 9,1 V:a ja 0-tilassa pienempi kuin 4,7 V:a. Muun muassa alhainen akkujännite tai kytketyt testilaitteet voivat tuottaa ongelmia ja sitä voidaan kompensoida muuttamalla R16:n resistanssia. (ELM327 OBD to RS232 Interpreter n.d, 65.)

Standardi sanoo R13:n ja R14:n arvoiksi 510 ohmia. Työhön valittiin 560 Ω:n vastukset, sillä ne ovat paljon yleisemmin saatavilla ja käyvät tarkoitukseen hyvin. Pienemmät arvot eivät käy, sillä virta kasvaa silloin liian isoksi. Vastukset ovat 0,5 W:n tehovastuksia, sillä 0-tilassa 14 V:n jännitteellä (ajoneuvo käynnissä, latausjännite) tehohäviöksi vastuksien yli tulee 0,35 W:a. (ELM327 OBD to RS232 Interpreter n.d, 65.)

J1850-standardi sisältää kaksi eri protokollaa. Yksijohtiminen VPW ja kaksijohtiminen PWM. Molemmat toimivat eri jännitteillä, siksi linjassa on säätöregulaattori. VPW tarvitsee noin 8 V:n jännitteen ja PWM noin 5 V:n jännitteen. Piirikaaviossa U5 regulaattorin adjustment-nastan vastuksilla jännitteiksi saadaan noin 7,5 V:a ja 5 V:a, jotka toimivat useimmissa tapauksissa. (ELM327 OBD to RS232 Interpreter n.d, 65.)

VPW käyttää vain BUS+ piuhaa ja sen signaalitasot muutetaan sopiviksi R26 ja R31 vastuksien jännitteenjaolla. 1-tila on yli 4,2 V:a ja 0-tila alle 2,2 V:a. (ELM327 OBD to RS232 Interpreter n.d, 65-66.)

PWM:n erotussignaali pitää muuttaa yhdeksi signaaliksi ELM327-piirille. Q7:aa käytetään tässä tapauksessa erovahvistimena. Q7 ja D8 säätävät linjan alarajan noin 1 V:ksi häiriönsuotosyistä. R27 kuristaa linjan virran ja R30 sulkee Q7-transistorin nopeammin tyhjäämällä hajakapasitanssit. (ELM327 OBD to RS232 Interpreter n.d, 66.)

R25 on valinnainen vastus. Se sulkee Q7-transistorin vielä nopeammin korkean kapasitanssin järjestelmissä. Ilman sitä, näissä tapauksissa saattaisi ilmetä virheitä linjassa. (ELM327 OBD to RS232 Interpreter n.d, 66.)

Nasta 14 ja Q5 ajavat J1850-protokollien miinuslinjan aktiiviseksi. Nasta 4, Q3 ja Q4 ajavat J1850-protokollien pluslinjan aktiiviseksi. (ELM327 OBD to RS232 Interpreter n.d, 66.) Liitäntöjen kytkennät on esitelty liitteessä 3.

3.3.5 Lukulaitteen piirikaavio, Bluetooth

ELM327-piiri toimii 5 V:n jännitteellä ja käytössä oleva Bluetooth-moduuli 3,3 V:n jännitteellä. Tämä logiikkaero on tasoitettava eli jännite on skaalattava. Rajapinnan luonti onnistuu helposti passiivikomponenttien avulla. Nasta 17 ELM327-piirissä on sarjaliikennelähetys nasta. Piirikaaviossa vastukset R34 ja R35 skaalaavat jännitteen-jaolla 0-5 V:n logiikan 0-3,3 V:n järjestelmään. (AN04 - ELM327 and Bluetooth n.d, 1-2.)

Nasta 18 on sarjaliikennevastaanotto. Siihen tarvitaankin hieman enemmän komponentteja. Bluetooth-moduulin lähetysnasta on kytkettynä transistorin emitteriin. Kun tämä nasta on 1-tilassa (yli 2,8 V:a), transistori on pois käytöstä ja noin 5 V:n jännite syötetään vastuksen kautta ELM327-piirin vastaanottoon. Näin saadaan 1-tila. (AN04 - ELM327 and Bluetooth n.d, 1-2.)

Kun Bluetooh-moduulin lähetysnasta on 0-tilassa. Transistori saturoituu ja virta kulkee transistorin kautta maihin ja tällöin ELM327-piirin vastaanottonasta ajetaan 0-tilaan. (AN04 - ELM327 and Bluetooth n.d, 1-2.)

ELM327-piiri operoi 4MHz:n kiteellä. Nopeus ei ole kovin suuri, jolloin transistorin hajakapasitansseista johtuvaa jännitteen laskun hidastuvuutta ei tarvitse huomioida. (AN04 - ELM327 and Bluetooth n.d, 2.) Liitteessä 4 on esitelty Bluetooth-osio piirikaaviosta.

3.4 Piirilevyt

3.4.1 Yleistä

Opinnäytetyön tarkoituksiin suunniteltiin kaksi erilaista piirilevyä eri suunnittelulähtökohdista. Ensimmäinen oli prototyyppipiirilevy, jonka toteutuksessa huomioitiin käsin toteutetun valmistusprosessin vaikeudet. Toinen piirilevy oli

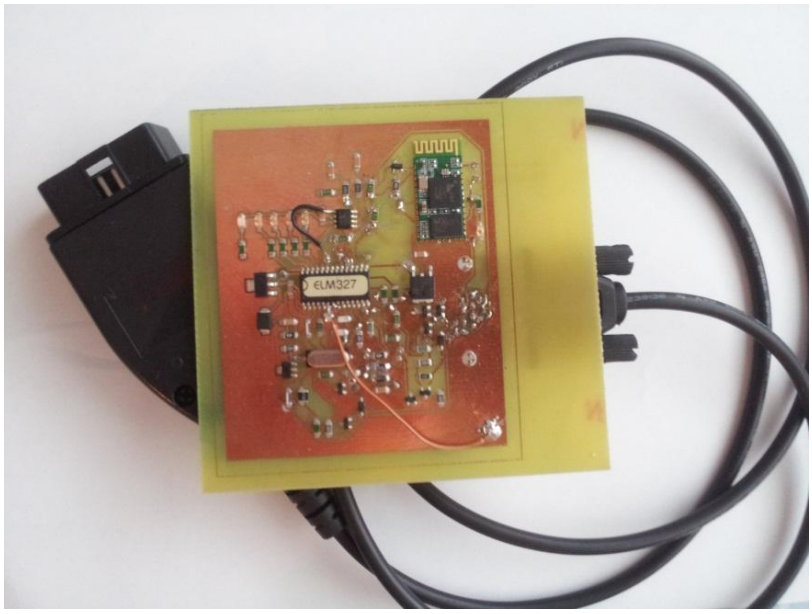
tarkoitettu massatuotantoon, ja pääpainona suunnittelussa oli mahdollisimman pieni hinta ja piirilevyn koko.

Piirilevysuunnittelu alkoi lajittelemalla komponentit ryhmiin kuten Hannu Tikkanen neuvoo kirjassaan (Tikkanen 2004, 21). Osat on helppo kasata pienissä ryhmissä parhaalla mahdollisella tavalla, jolloin vältytään turhan pitkiltä vedoilta. Ryhmät asetellaan piirilevylle ja tärkeimmät vedot tehtiin käsin. Melkein kaikki loput vedot onnistuivat hyvin automaattisesti PADS Router-ohjelmalla. Loppuun tarvittiin vain pientä korjaamista.

3.4.2 Prototyyppipiirilevy

Prototyyppipiirilevyn suunnittelussa johdinleveydet, kuparin välys toisiinsa ja komponenttien etäisyys toisiinsa pidettiin suurempana kuin normaalisti (minimissään 0,5 mm). Näin toimimalla varmistettiin prototyypin toimivuus. Prototyyppipiirilevyssä pyrittiin myös minimoimaan läpivedot (vias), sillä kaikki läpivedot piti tehdä käsin. Ohuiden juotostäplien poraaminen käsin on myös hankalaa, sillä ohut kupari irtoaa helposti levyn pinnasta ja tällöin levy on pilalla. Myös peruskomponentit tilattiin isommalla koteloinnilla (1206, imperial).

Piirilevy on kaksipuoleinen, sillä yksipuoleisen levyn tekeminen useiden yritysten jälkeen, ilman hyppylankoja, tuntui mahdottomalta. Prototyyppiin valittiin DSUB-miniliitin, sillä siihen löytyi valmis DLC-liitinkaapeli ja siten erillistä kaapelia ei tarvinnut ostaa. Tuotantopiirilevyyn DSUB-liitintä ei tule, vaan OBD2-liittimen johdot kytketään suoraan piirilevylle. Levyn mitat ovat 72,1 mm x 85,1 mm. Kuviossa 5 on valmis ja toimiva prototyyppi.



KUVIO 5. Prototyyppi

3.4.3 Tuotantopiirilevy

Massatuotantopiirilevyssä peruskomponentit valittiin pienemmillä koteloinneilla (0603, imperial) piirilevyn koon minimoimiseksi. Vetojen leveydet kavennettiin ja kuparien välystä pienennettiin (minimissään 0,3 mm). Näin saatiin aikaan paljon pienempi piirilevy. Välys olisi voitu asettaa paljon pienemmäksikin, mutta hinta olisi tällöin kasvanut suotta. Suunnittelusäännöt otettiin Prinelin suosituksista (Suunnitteluohjeet n.d.). Mahdollisimman pientä ja mahdollisimman halvalla. Läpivetoja sallittiin huomattavasti enemmän, sillä lukumäärä ei hirveästi lisännyt piirilevyn hintaa, mutta piirilevyn koko pieneni huomattavasti. Läpivedot pidettiin kohtalaisen isoina (0,7 mm), sillä pienemmät reiät olisivat tulleet kalliimmaksi. Komponentit ladottiin vain toiselle puolelle. Näin kustannukset pysyivät mahdollisimman pienenä myös siltä osin. Massatuotantopiirilevyn silkkipainomerkintöjä ja komponenttisijoittelusuunnittelua paranneltiin huomattavasti. Levyyn lisättiin myös kohdistusmerkit komponenttiladontakoneen tarkoituksiin. Levyn mitoiksi tuli 50,9 mm x 69 mm.

Liitteissä 5-7 on esitelty massatuotantopiirilevy. Levystä tehtiin RS-274X-versioiset gerber-tiedostot, joiden avulla levyjä voidaan teettää piirilevyvalmistajilla. Tarvittavia tiedostoja Prinelin ohjeiden mukaan on kuparitasot, juotteenestopinnoitetasot, kompo-

nenttisijoittelutasot, mittapiirustukset ja info-tiedosto, josta käy ilmi muun muassa piirilevyn materiaali (Suunnitteluohjeet n.d.). Myös erillinen excellon poraustiedosto tehtiin varmuuden vuoksi.

3.5 Lukulaitteen konfigurointi

Lukulaitteessa tarvitsee konfiguroida Bluetooth-moduuli sekä ELM327-piiri. Molemmat konfiguroidaan sarjaliikenneportin avulla AT-komennoilla. Helpoiten osat saadaan konfiguroitua erillisissä testipenkeissä.

Ohjelmoitu sovellus itsessään konfiguroi ELM327-piirin suurelta osin. Ainoa ennalta muokattava ominaisuus on sarjaliikenneväylän nopeus. Prototyypissä tyydyttiin 9,6 kbaudin nopeuteen, mikä riittää sovelluksen toimintaan. Mutta tuotantomalliin nopeutta voidaan ja kannattaa nostaa, varsinkin jos sovelluksen ominaisuuksia parannetaan. Sarjaliikennenopeuden nosto on mahdollistettu lukulaitteen nastan 6 avulla (luku 3.3.2). Nopeus voidaan ohjelmoida Bluetooth-moduulin suurimmalle nopeudelle (115,2 kbaud) AT-komennolla PP 0C 23.

Bluetooth-moduulissa on valmiiksi asennettuna Linvor 1.5-ohjelmisto (firmware). Ohjelmisto voidaan päivittää JTAG-liittimen avulla, mutta tarvetta siihen tuskin on. Moduuli toimii vakiona serverinä nimeltä linvor. Vakiona sarjaliikennenopeus on 9,6 kbaudia ja yhteyden muodostamiseen tarvittava pin-koodi on 1234. Tuotantoversioon kannattaisi muuttaa ainakin sarjaliikennenopeus ja laitteen nimi, joka näkyy käyttäjälle. Kun Bluetooth-moduuli on testipenkissä ja sen nasta 34 on yhdistetty käyttöjännitteeseen, on moduuli valmis vastaanottamaan AT-komentoja. Nopeus voidaan muuttaa 115,2 kbaudin nopeuteen komennolla AT+UART=115200,0,0 (nopeus, 1 pysäytysbitti, ei pariteettibittiä) ja nimi komennolla AT+NAME=<nimi>. (CuteDigi BMX Bluetooth to UART/12C/USB Module (GEN II) n.d.)

3.6 Lukulaitteen suunnittelun yhteenveto

Oman lukulaitteen teettämisen tarpeellisuus on hieman ristiriitaista. Onhan selvää, että oman laitteen kanssa voi varmistua siitä, että kaikki ominaisuudet ovat käytössä ja laadunvalvontaa on helpompi suorittaa. Vaikka lukulaite onkin yleispätevä, ei sillä

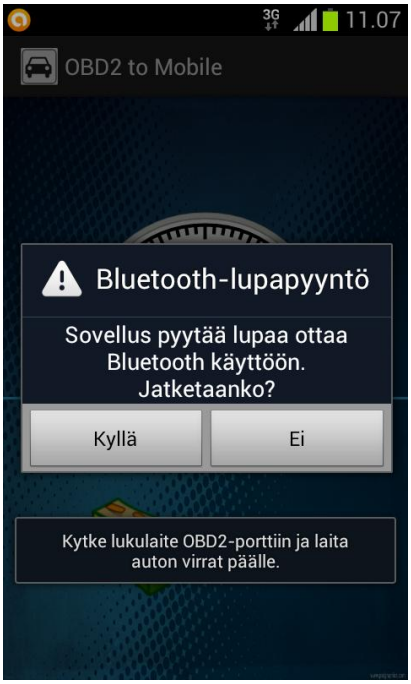
pääse käsiksi kuin standardinmukaisiin osiin ajoneuvon sähköjärjestelmässä. Tällöin iso osa ilmenevistä vioista jää selvittämättä.

Hinnan puolesta oma laite tulee isollakin katteella selvästi edullisemmaksi kuin esimerkiksi Amerikan suosituin lukulaite Scantool, mutta Kiinassa tehtyjen laitteiden kanssa hinnalla ei pysty kilpailemaan. Helpointa olisi varmasti integroida mahdolliseen tuotteeseen jokin hyvä massatuotettu lukija.

4 SOVELLUS

Lukulaitteelle ohjelmoitiin sovellus Android-käyttöjärjestelmälle. Sovellus toimii kaikissa Android-versioissa 2.3:sta alkaen. Sovellus ohjelmoitiin Java-kielellä (Java Run Time Enviroment 7) käyttäen Eclipse SDK 4.2.0 -kääntäjää, johon oli sisällytetty Android ADT-lisäosa. Kaikki sovelluksessa käytetyt kuvat ovat lisenssivapaita Clker.com-sivustolta (http://www.clker.com).

4.1 Sovelluksen toiminta



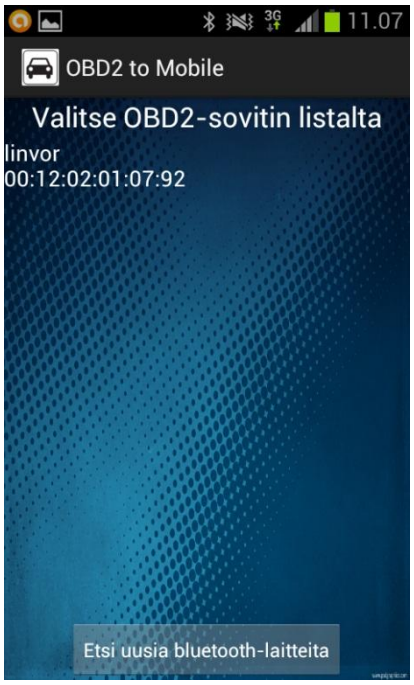
KUVIO 6. Sovellus: Bluetooth lupapyyntö

Kun sovellus käynnistetään, se etsii matkapuhelimesta Bluetooth-ohjainta. Jos sellainen löytyy, sovellus kysyy käyttäjältä lupaa asettaa Bluetooth päälle ellei se ole jo päällä (ks. Kuvio 6). Mikäli käyttäjä vastaa kyllä tai Bluetooth oli jo päällä, sovellus näyttää aikaisemmin matkapuhelimen kanssa paritetut Bluetooth-laitteet listassa (ks. Kuvio 7). ’Etsi Bluetooth lukulaite’-nappia painamalla voidaan etsiä uusia laitteita, jotka listataan mahdollisten vanhojen laitteiden perään. Kun käyttäjä valitsee lukulaitteen listasta, sovellus yhdistää siihen. Tämän jälkeen sovellus yhdistää lukulaitteen ajoneuvon järjestelmään. Yhdistämisen jälkeen sovellus konfiguroi lukulaitteen automaattisesti ja aloittaa kyselyt ajoneuvon moottorin kierrosluvusta ja akun jännitteestä. Tietoja kysytään sovelluksen perusikkunassa jatkuvasti. Kierrosluku esitetään analogisen näköisel-

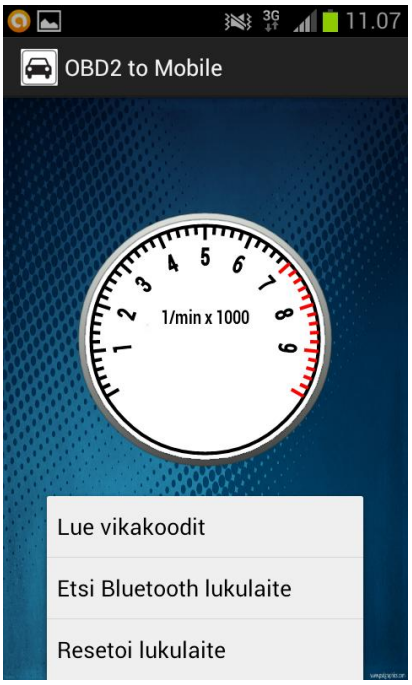
lä mittarilla sovelluksen pääruudussa. Akkujännite ilmoitetaan kierroslukumittarin alla.

Matkapuhelimen asetukset-painikkeen avulla käyttäjä voi lukea ajoneuvon vikakoodit, käynnistää lukulaitteen uudelleen tai etsiä Bluetooth-laitteen uudestaan (ks. Kuvio 8). Lukulaitteen uudelleenkäynnistys ja Bluetooth-lukulaitteen uudelleen etsiminen ovat ongelmatapauksien korjaamiseksi ja niitä ei tarvita normaalitapauksessa.

Kun ’Lue vikakoodit’-painiketta painetaan, kysyy lukulaite ajoneuvolta tallennettujen vikakoodien lukumäärää. Mikäli vastaukseksi saadaan nolla, ilmoittaa ohjelma siitä käyttäjälle Toast-viestillä[1] (ks. Kuvio 10). Jos vikakoodeja löytyy, lopetetaan kierros-luku- ja jännitedatan kysely ja pyydetään itse vikakoodit ajoneuvolta. Luetut vikakoodit näytetään erillisessä ikkunassa (ks. Kuvio 9). Jos moottorin vikavalo on päällä, ikkunan ’CHECK ENGINE’-teksti näytetään punaisena. Mikäli valo ei ole päällä, on teksti harmaa. Kun käyttäjä palaa aloitusikkunaan, jatkuu kierrosluku- ja jännitedatan kysely ajoneuvolta automaattisesti.



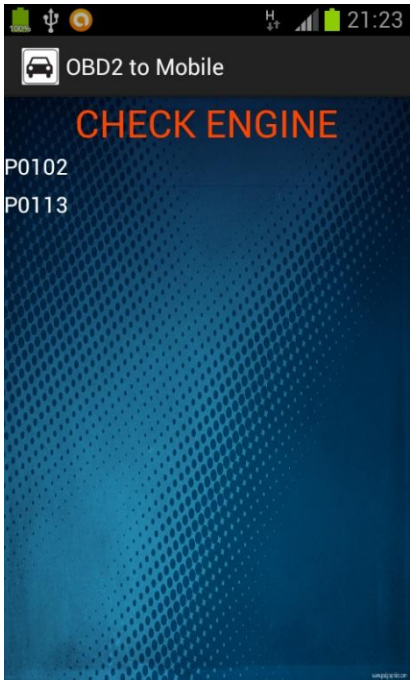
KUVIO 7. Sovellus: Bluetooth-laitelistaus



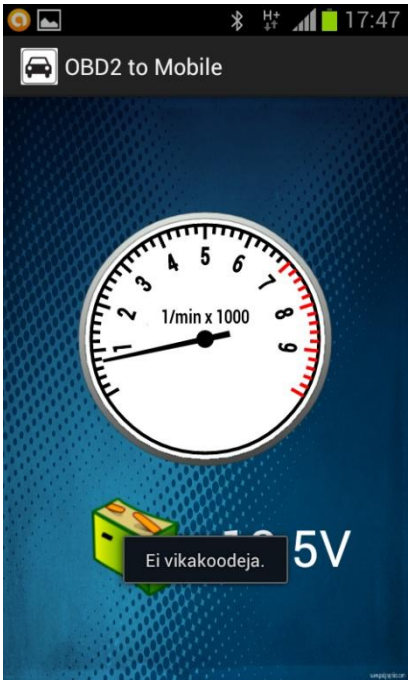
KUVIO 8. Sovellus: Valikko

[1] Toast on Android-käyttöjärjestelmän viestiominaisuus. Toast on viestilaatikko ruudun alalaidassa.

Sovelluksen layoutteja ei ole tehty toimimaan vaakasuunnassa. Android myös käynnistää activityn uudelleen, mikäli ruudun suunta muuttuu. Näiden ongelmien takia sovellus on nykyisellään pakotettu toimimaan vain pystysuunnassa.



KUVIO 9. Sovellus: Vikakoodien esitys



KUVIO 10. Sovellus: Ei vikakoodeja

4.2 Ohjelmointi

4.2.1 Ohjelman käynnistys

Kun sovellus on asennettu Android-käyttöjärjestelmään, luo se pikakuvakkeen matkapuhelimen sovellusvalikkoon (ks. Kuvio 11). Sovelluksen käynnistyessä käyttäjä aloittaa MainActivity-nimisestä pääohjelmasta. Ensiksi käyttäjä saa Toast-viestin, jossa neuvotaan kytkemään lukulaite ajoneuvon OBD2-porttiin ja kytkemään ajoneuvoon virta (ks. kuvio 6). Tämän jälkeen sovellus etsii matkapuhelimen Bluetooth-ohjainta. Mikäli sellaista ei löydy, käyttäjä saa Toast-viestin ilmoittaen, että Bluetooth ei ole käytössä.

KUVIO 11. Sovellus: Sovelluksen pikakuvake

Jos Bluetooth-ohjain löytyy, mutta Bluetooth ei ole päällä, käynnistetään Bluetooth-ohjaimen metodi ACTION_REQUEST_ENABLE. Käyttäjältä pyydetään lupaa laittaa Bluetooth päälle. Mikäli käyttäjä antaa luvan käynnistää Bluetooth-ohjain tai Bluetooth on jo päällä, siirretään käyttäjä Bluetooth-luokan ilmentymään eli olioon, jossa Bluetooth-laitteita voi hakea ja valita. Siirto tehdään startActivityForResult-komennolla, requestCode-muuttujan ollessa CONNECT. Kun aliohjelma päättyy, requestCode-muuttuja kertoo onActivityResult-metodille, mikä aliohjelma on juuri päättynyt. Toiminta on havainnollistettu ohjelmointikielellä alla.

```
intentbt = new Intent(this, Bluetooth.class);
if (aBluetoothAdapter == null) {
   Toast.makeText(context, context.getString(R.string.ei_bt),
     Toast.LENGTH_LONG).show();
}
else {
  if (aBluetoothAdapter.isEnabled()) startActivityForResult(intentbt,CONNECT);
  else {
    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
  }
}
```

4.2.2 Bluetooth-laitelista

Bluetooth-luokassa Bluetooth-laitteet sijoitetaan ArrayAdaptereihin. Tunnetuille ja uusille laitteille on omat adapterinsa. Adapterit näytetään käyttäjälle layoutissa ListViewien avulla. ListView asetetaan kosketusaktiiviseksi. Alla on esimerkki listauksen alustamisesta.

```
pairedArrayAdapter = new ArrayAdapter<String>(Bluetooth.this, R.layout.list_white);
ListView paireddevices = (ListView)findViewById(R.id.paireddevices);
paireddevices.setAdapter(pairedArrayAdapter);
paireddevices.setClickable(true);
paireddevices.setOnItemClickListener(DeviceListClickListener);
```

Vakiona sovellus ei etsi uusia laitteita vaan käyttäjän tulee painaa ruudun alalaidassa olevaa nappia (ks. Kuvio 7). Näin toimimalla säästetään resursseja, sillä laitteiden etsiminen on raskas prosessi ja normaalitapauksessa käyttäjän ei tarvitse etsiä lukulaitetta kuin ensimmäisellä kerralla. Matkapuhelinta ei tarvitse asettaa näkyvyystilaan, sillä lukulaite toimii yhteyden palvelimena ja matkapuhelin löytää sen itse. BluetoothDevice-metodin komento .getBondedDevices listaa kaikki matkapuhelimeen aikaisemmin paritetut Bluetooth-laitteet. Laitteiden nimen ohella listassa näkyy myös

MAC-osoite. Jos yhtään laitetta ei ole valmiiksi paritettu, ohjelma siirtyy suoraan etsimään laitteita. Alla on ohjelmointiesimerkki valmiiksi paritettujen laitteiden lisäämisestä adapteriin.

```
Set<BluetoothDevice> pairedDevices = bluetoothAdapter.getBondedDevices();
if (pairedDevices.size() > 0) {
   for (BluetoothDevice device : pairedDevices)
      pairedArrayAdapter.add(device.getName()
        + "\n" + device.getAddress());
   } else Bluetooth.this.Etsi();
}
```

Etsi-nappulaa painamalla aloitetaan uusien Bluetooth-laitteiden etsintä (ks. Kuvio 7). Jos laite löydetään, rekisteröidään uusi BroadcastReceiver-metodi. BluetoothDevice-metodin .ACTION_FOUND-komento kertoo, että Bluetoothin kantamassa on jokin laite. Metodissa verrataan löytynyttä laitetta aikaisemmin paritettuihin laitteisiin. Jos yhtäläisyyttä ei ole, laitteen tiedot kirjoitetaan uusien laitteiden listaan. Lista on samanlainen kuin aikaisemmin paritettujen laitteiden lista. Uusien laitteiden lista tulee paritettujen laitteiden listan alle näytössä. Mikäli laitteita ei löydetä vieläkään, kysytään käyttäjältä Toast-viestillä, onko lukulaite päällä. BroadcastReceiver-metodi on esitetty alla.

```
private final BroadcastReceiver btReceiver = new BroadcastReceiver()  {
  @Override
  public void onReceive(Context context, Intent intent) {
   String action = intent.getAction();
   if(BluetoothDevice.ACTION_FOUND.equals(action)) {
     BluetoothDevice device =
       intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
     if (device.getBondState() != BluetoothDevice.BOND_BONDED )
       newArrayAdapter.add(device.getName() + "\n" + device.getAddress());
     } else {
       Toast.makeText(context, context.getString(R.string.bluetooth),
         Toast.LENGTH_LONG).show();
       }
  }
};
```

Käyttäjä valitsee oikean laitteen listalta. OnItemClickListener-metodi käynnistyy käyttäjän valinnasta. Metodi lopettaa ensiksi laitteiden etsinnän. Sitten se tallentaa lukulaitteen MAC-osoitteen String-muuttujaan. Bluetooth-olio lopetetaan finish()-komennolla ja samalla lähetetään lukulaitteen MAC-osoite takaisin pääohjelmaan. OnItemClickListener-metodin on esitetty seuraavalla sivulla.

```
private OnItemClickListener DeviceListClickListener
                            = new OnItemClickListener() {
  public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
                                                  long arg3) {
      bluetoothAdapter.cancelDiscovery();
      String info = ((TextView) arg1).getText().toString();
      String address = info.substring(info.length() - 17);
      Intent MAC = new Intent();
      MAC.putExtra(EXTRA_MAC_ADDRESS, address);
      setResult(Activity.RESULT_OK, MAC);
      finish();
      }
};
```

Pääohjelmaan paluu tapahtuu OnActivityResult-metodin kautta. requestCode-muuttujan avulla päätellään, mikä aliohjelma on juuri päättynyt. Yhdistämisprosessissa requestCode-muuttuja on CONNECT. MAC-osoitteen pohjalta luodaan uusi BluetoothDevice-muuttuja. Bluetooth-palvelu käynnistetään ja muuttuja nimeltä device lähetetään sinne. Muuttuja pitää sisällään tiedot yhdistettävästä laitteesta. Mikäli lukulaite ei ole matkapuhelimelle entuudestaan tuttu, täytyy käyttäjän yhdistää laitteet ennalta määrätyn pin-koodin avulla. onActivityResult-metodi on esitelty alla.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
  Context context = getApplicationContext();
  intentbt = new Intent(this, Bluetooth.class);
  switch (requestCode) {
  case REQUEST_ENABLE_BT:
    if (resultCode == 0) {
      Toast.makeText(context, context.getString(R.string.ei_bt),
         Toast.LENGTH_LONG).show();
    } else startActivityForResult(intentbt,CONNECT);
    break;
  case CONNECT:
    if (resultCode == Activity.RESULT_OK) {
      String address = data.getExtras()
                          .getString(Bluetooth.EXTRA_MAC_ADDRESS);
      BluetoothDevice device = aBluetoothAdapter.getRemoteDevice(address);
      mBTService.connect(device);
    }
    break;
  case DTC_DONE:
    if (resultCode == Activity.RESULT_OK) {
      read_dtc = false;
    if (mBTService.getState() == connected && initialized == true)
     new getData().execute();
    }
  }
}
```

onActivityResult-metodi on käytössä jo aikaisemmin Bluetooth-ohjaimen käynnistämisen yhteydessä (case REQUEST_ENABLE_BT) ja myöhemmin vikakoodien esitysaliohjelmasta poistuttaessa (case DTC_DONE).

4.2.3 Bluetooth-palvelu

Pääohjelma käynnistää itsensä rinnalle BTService-nimisen palvelun. Palvelua käytetään Bluetoothin yhdistämiseen, hallinnoimiseen ja tiedonsiirtoon. Palvelu mahdollistaa Bluetoothin toimimisen taustalla. Tällöin Bluetoothia ei ole sidottu mihinkään luokkaan ja se helpottaa sovelluksen jatkokehitystä. Bluetooth-palvelun ohjelmoinnissa on käytetty esimerkkinä kääntäjän ADT-lisäosan mukana tullutta esimerkkisovellusta nimeltä BluetoothChat.

Ensiksi käynnistetään BTServicen connect-metodi. Metodi tarkastaa, onko olemassa olevia yhteyksiä olemassa. Se myös käynnistää ConnectThread-olion nimeltä mConnectThread. connect-metodi on kokonaisuudessaan seuraavaksi.

```
public synchronized void connect(BluetoothDevice device) {
  if (mConnectThread != null) {
    mConnectThread.cancel();
    mConnectThread = null;
  }
  if (mConnectedThread != null) {
    mConnectedThread.cancel();
    mConnectedThread = null;
  }
  mConnectThread = new ConnectThread(device);
  mConnectThread.start();
}
```

Thread eli säie on rinnakkaisesti toimiva luokka. Bluetoothin toiminta ei silloin keskeytä käyttäjän toimintaa esimerkiksi pääohjelmassa. ConnectThread luo Bluetooth-yhteyden avaamalla kannan (socket), mikä yhdistää lukulaitteeseen käyttäen yleistä sarjaliikenteelle määritettyä UUID:tä ja pääohjelmasta saatua device-muuttujaa. Kun kanta on luotu, tuhotaan ConnectedThread-säie ja käynnistetään connected- metodi. Mikäli yhteyttä ei voida muodostaa, suljetaan kanta. ConnectThread on esitetty seuraavalla sivulla.

```
private class ConnectThread extends Thread {
  private final BluetoothSocket btSocket;
  private final BluetoothDevice btDevice;
  public ConnectThread(BluetoothDevice device) {
    btDevice = device;
    BluetoothSocket tmp = null;
    try {
      tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
    } catch (IOException e){}
    btSocket = tmp;
  }

  public void run() {
    aBluetoothAdapter.cancelDiscovery();
    try {
      btSocket.connect();
    } catch (IOException e) {
        try {
            btSocket.close();
        } catch (IOException e2) {}
  return;
  }
  synchronized (BTService.this){
    mConnectThread = null;
    connected(btSocket, btDevice);
  }
```

connected-metodi tarkastaa connect-metodin tavoin olemassa olevat säikeet ja tuhoaa ne tarvittaessa. Metodi lähettää pääohjelmaan käsittelijän (Handler) kautta viestin yhteyden luomisen onnistumisesta ja aloittaa mConnectedThread-olion. Tämä säie ylläpitää yhteyttä. Connected-metodi on hyvin vastaava ohjelmoinniltaan kuin connect-metodi.

ConnectedThread-luokka luo input- ja outputstreamit, joita käytetään tiedon välitykseen Bluetoothin ylitse. Alla on ote ConnectedThread-säikeen alusta.

```
InputStream tmpIn = null;
OutputStream tmpOut = null;
try {
  tmpIn = socket.getInputStream();
  tmpOut = socket.getOutputStream();
} catch (IOException e) {}

InStream = tmpIn;
OutStream = tmpOut;
```

Lukulaitteelle kirjoittaminen tapahtuu yksinkertaisesti lähettämällä sille tavutaulukon OutStream.write-komennolla write-metodin kautta. write-metodi on sisällytetty ConnectedThread-säikeeseen ja toimii asynkronisesti. Metodi on esitelty seuraavaksi.

```
public void write(byte[] buffer) {
  try {
    OutStream.write(buffer);
  } catch (IOException e) {}
```

Lukulaitteelta vastaanottaminen vaatii hieman enemmän ohjelmointia. Bluetoothin kautta saatava inputstream luetaan BufferedReaderin avulla. BufferedReader lukee inputstreamia isommissa erissä ja suorana tekstinä. Jos inputstreamia luettaisiin suoraan muuttujaan, se toimisi vain yksi merkki kerrallaan tavuina. Luku on BufferedReaderin kanssa nopeampaa ja inputstream voidaan suoraan koota käyttäen StringBuilderia.

Sovelluksessa StringBuilder kasaa lukulaitteelta tullutta vastausta siihen asti kunnes löydetään lopetusmerkki. ELM327-piiri on lukulaitteessa säädetty lopettamaan jokainen viesti linefeed- ja carriage return-merkkeihin. Kun merkit löydetään peräkkäin, on viesti valmis ja se lähetetään pääohjelmaan käsittelijän kautta. StringBuilder tuhotaan jokaisen viestin lähettämisen jälkeen muistin vapauttamiseksi. Alla on esitetty inputstreamin luku ja lähetys pääohjelmaan.

```
char[] buffer = new char[128];
int bytes;
String end = "\n\r";
try {
  BufferedReader br = new BufferedReader(new InputStreamReader(InStream));
  StringBuilder Msg = new StringBuilder();
  while ((bytes = br.read(buffer)) != -1) {
    Msg.append(new String(buffer, 0, bytes));
    int endId = Msg.indexOf(end);
    if (endId != -1) {
      String readInput = Msg.substring(0, endId);
      btHandler.obtainMessage(MainActivity.READ, bytes, -1, readInput)
                              .sendToTarget();
      Msg.delete(0, Msg.length());
      readInput = null;
    }
  }
} catch (IOException e){}
```

4.2.4 Bluetooth-datan vastaanotto

Käsittelijälle tulevat viestit lokeroidaan switch/case-rakenteella. Lokerointi tapahtuu .obtainMessage-rakenteen what-parametrin tiedoilla (ensimmäinen parametri). Kun what-parametri on CONNECTED, aloitetaan bus_init-metodi. Bluetooth-palvelu lähettää tämän heti yhteyden luomisen jälkeen. Parametri READ on käytössä, kun Bluetooth-palvelusta tulee inputstreamin kautta viesti.

Ajoneuvolta tuleva vastaus sisältää aina viestin edessä olevat tunnistustavut, joiden avulla viestit voidaan tunnistaa ja lähettää oikeaan metodiin. Esimerkiksi kierrosluvun saa ajoneuvolta lähettämällä sille 010C-viestin. Viesti kysyy ensimmäisen moodin kahdettatoista PID:ä (heksalukuna C). Vastaukseen ajoneuvo lisää mooditavuun desimaalina 64 (40 heksalukuna). Koska ELM327-piiri lähettää heksalukuja, jokaisen kierroslukukyselyn vastauksen alkuun tulee tunnistemerkit 410C. Tätä tietoa käytetään sovelluksessa viestin sisällön tunnistamiseen. Käsittelijä-metodi on esitetty kokonaisuudessaan alla.

```
private final Handler btHandler = new Handler () {
  @Override
  public void handleMessage(Message msg) {
    switch (msg.what) {
    case READ:
      if (msg.arg1 > 0) {
         String tmpString = (String) msg.obj;
         if (tmpString.indexOf("4100") != -1) {
           initialized = true;
           new getData().execute();
         }
         else if (tmpString.indexOf("CAN") != -1) CAN = true;
         else if (tmpString.indexOf("V") != -1) show_voltage(tmpString);
         else if (tmpString.indexOf("410C") != -1) show_rpm(tmpString);
         else if (tmpString.indexOf("4101") != -1) dtc_count(tmpString);
         else if (tmpString.indexOf("43") != -1 && read_dtc == true)
           diagnostic_trouble_codes(tmpString);
         else tmpString = null; break;
         } else break;
    case CONNECTED:
      bus_init();
      break;
    default:
      break;
    }
  }
};
```

4.2.5 Yhteyden luominen ajoneuvon järjestelmän kanssa (bus_init- ja reset_device-metodit)

bus_init- ja reset_device-metodit ovat yksinkertaisia ohjelmanpätkiä, joilla lähetetään tiettyjä parametreja ELM327-piirille.

reset_device-metodi nollaa piirin alkutilaansa ja syöttää sen jälkeen tarvittavat piirin sisäistä toimintaa muuttavat komennot, jotta sovellus toimii oikein. Komennot ovat AT-komentoja. ELM327 käyttää AT-komentoja, sillä ne eroavat ajoneuvon käyttöön tarkoitetuista komennoista (ELM327 OBD to RS232 Interpreter n.d, 7.). Ajoneuvoissa

käytetyissä komennoissa on käytössä vain heksadesimaalimerkit (0-9, A-F). Syötettävät komennot ovat ATD1, ATE0, ATS0, ATSP0 ja ATDP.

ATD1on vain CAN-väylän toimintaan vaikuttava komento. D1 näyttää käyttäjälle CAN-viestin edessä olevan PIC-tavun. (ELM327 OBD to RS232 Interpreter. n.d, 14.) Tavua tarvitaan vikakoodien lukemisessa, mutta anturidatan tulkitsemisessa se ohitetaan.

ATE0 poistaa niin sanotun kaiun (echo) vastausviesteistä (ELM327 OBD to RS232 Interpreter n.d, 15.). Jos kaiku olisi päällä, jokaisessa ELM327-piiriltä tulevassa viestissä olisi alussa komento, joka sille lähetettiin. Tämä toiminto ei ole tarpeen.

ATS0 poistaa välimerkit (spaces) ELM327-piiriltä tulevista viesteistä (ELM327 OBD to RS232 Interpreter n.d, 22.). Koska sovellus tulkitsee ja muokkaa vastaukset itse, tämä viestejä selkeyttävä toiminto on turha ja vie vain turhaan prosessoriaikaa.

ATSP0 asettaa valitun protokollan automaattiasentoon (ELM327 OBD to RS232 Interpreter n.d, 23.). Näin varmistetaan, että sovellus toimii aina, vaikka sitä käytettäisiin useamman eri protokollan kanssa.

ATDP kysyy ajoneuvolta, mikä protokolla on käytössä (ELM327 OBD to RS232 Interpreter n.d, 15.). Komentoa käytetään vain määrittämään onko käytössä CAN-protokolla. Tieto on tarpeellinen vikakoodeja luettaessa.

Komennot on asetettava jokaisen resetin yhteydessä, sillä ATZ-komento asettaa ELM327-piirin alkutilaan (ELM327 OBD to RS232 Interpreter n.d, 7.). Käyttäjä voi myös resetoida lukulaitteen itse sovelluksen pääikkunan valikosta löytyvän painikkeen avulla (ks. kuvio 6). Sovellus nukkuu jokaisen komennon jälkeen. Tällä varmistetaan, että piirillä on aikaa tulkita jokainen komento oikein. Alla on esimerkki siitä, kuinka reset_device-metodissa lähetetään komentoja ELM327-piirille.

```
final String ap = "atsp0"+"\r";
byte[] sp0 = ap.getBytes();
mBTService.write(sp0);
sleep(300L);
```

sleep-komentoa tarvitaan ohjelmassa niin usein, että siitä on tehty oma metodinsa, jotta ohjelmointi helpottuisi. Metodiin lähetetään long-muuttuja, joka kuvaa haluttua nukkumisaikaa millisekunteina. Seuraavaksi on esitelty sleep-metodi.

```
private void sleep(long time) {
  try {
    Thread.sleep(time);
  } catch (InterruptedException e) {}
}
```

bus_init-metodi käynnistyy heti, kun Bluetooth-lukulaite on yhdistetty. bus_init-metodi käy ensin lävitse reset_device-metodin. Näin varmistetaan, että lukulaite on alkutilassaan ja oikein konfiguroitu. Tämän jälkeen lähetetään piirille ensimmäisen moodin 0 PID (komento 0100). Ajoneuvolta odotetaan vastaus ja silloin tiedetään, että ajoneuvo on valmis lähettämään tietoa. Vastaus on muuten täysin yhdentekevä. Tarkoituksena on vain olla varma, että ajoneuvo on valmis, sillä muutamat protokollat vaativat niin sanotun hitaan käynnistyksen.

4.2.6 Anturitiedon hakeminen

Kun käsittelijä (luku 4.2.4) huomaa, että bus_init-metodi on suoritettu oikein ja ajoneuvo on vastannut viestiin, käynnistetään asynkroninen aliohjelma nimeltä getData.

getData-luokka on AsyncTask, jolla lähetetään ajoneuvolle tiedusteluja kierrosluvusta ja akun jännitteestä. getData toimii rinnakkain käyttäjän toimintojen kanssa kuten Threadit. Sovellus ei tällöin jää jumiin. Tämä on elintärkeää, sillä getData on aina päällä, kun yhteys ajoneuvoon on luotu ja ajoneuvolta ei tiedustella vikakooditietoja. Do/while-silmukan lopetusehtona onkin read_dtc-booleanmuuttuja, mikä asetetaan todeksi, kun aloitetaan kysely vikakoodeista. getData-luokka on esitetty kokonaisuudessaan seuraavaksi.

```
private class getData extends AsyncTask<Void,Void,Void> {
  final String voltit = "atrv"+"\r";
  final String rundit = "010C1"+"\r";
  byte[] akku = voltit.getBytes();
  byte[] kiekka = rundit.getBytes();
    @Override
    protected Void doInBackground(Void... params) {
      do {
        for (int d = 0; d < 2; ++d) {
          sleep(300L);
          mBTService.write(kiekka);
        }
        sleep(300L);
        mBTService.write(akku);
      }  while (read_dtc == false);
      return null;
}
```

4.2.7 Anturitiedon käyttö

Kun käsittelijälle tulee vastauksia antureilta, ne tunnistetaan ja siirretään omiin metodeihinsa. Akun jännitetiedon tullessa, aloitetaan show_voltage- metodi ja kierrosluvun tullessa show_rpm-metodi.

show_voltage-metodissa vastauksesta poistetaan turhat tiedot muokkaamalla String-muuttujaa. Muuttujasta etsitään V-merkin (volttia) paikkatieto ja sen avulla muokataan muuttuja uudelleen sisältämään vain ne tiedot, mitä halutaan näyttää (esimerkiksi 12.1V). Sen jälkeen tämä jannite-niminen String-muuttuja näytetään käyttäjälle layoutin TextView-teksti-ikunnassa. show_voltage-metodi on esitetty alla.

```
private void show_voltage(String jannite) {
  int ind = jannite.indexOf("V");
  jannite = jannite.substring(ind-4, ind+1);
  TextView vastaus;
  vastaus = (TextView) findViewById(R.id.voltage);
  vastaus.setText(jannite);
}
```

Kierroslukudataa joudutaan muokkaamaan hieman enemmän. Ensiksi käsittelijältä tuleva String-muuttuja lähetetään edelleen cleanString-metodiin. Tämä metodi poistaa muuttujasta kaikki merkit, jotka eivät ole heksalukuja. Tämä tarkoittaa käytännössä viestin lopetusmerkkejä (new line, carriage return) ja > - merkkiä, minkä ELM327-piiri lisää lopetusmerkkien jälkeen.

Palautuneesta String-muuttujasta etsitään tunnistustavut. Kierroslukukyselyn tapauksessa siis 410C-heksalukusarja. Paikkatiedon avulla tunnistustavut poistetaan.

Kun String-muuttujassa on jäljellä enää kierroslukuvastaus heksalukuna, muutetaan se kymmenlukujärjestelmään integer-muuttujaksi. Integer-muuttuja jaetaan vielä neljällä ja tulos lähetetään float-muuttujana RevCounter-luokkaan. Neljällä jako on tarpeellista, sillä lukulaite lähettää kierrosluvun heksalukuna ¼-kierroksina. cleanString- ja show_rpm-metodit on esitelty seuraavaksi.

```
private String cleanString (String response) {
  response = response.replaceAll("[^0-9a-fA-F]", "");
  return response;
}

private void show_rpm(String kierrokset) {
  kierrokset = cleanString(kierrokset);
  int ind = kierrokset.indexOf("410C");
  kierrokset = kierrokset.substring(ind+4);
  int rounds = Integer.parseInt(kierrokset, 16);
  float revs = (float) rounds/4;
  revcounter.receiveRPM(revs);
}
```

4.2.8 Kierroslukumittari

Kierroslukumittarin ohjelmoinnissa on käytetty esimerkkinä Mind The Robot-sivuston lämpömittariohjelmaa (Memruk 2010.).

RevCounter-luokka piirtää ohjelman pääikkunaan kierroslukumittarin. Jokainen mittarin osa piirretään erikseen ja suhteessa aluksi määritettävään canvakseen (piirtokangas). Canvaksen koko määritetään MainActivityn-layoutissa. Tässä versiossa canvaksen leveydeksi on määritetty 250 dp:tä. DP määrittää objektin koon riippumatta matkapuhelimen näytön suhteellisesta tarkkuudesta eli PPI:stä. Mittari näyttää samalta huolimatta siitä, millainen näyttö matkapuhelimessa on käytössä. RevCounter-luokka hoitaa korkeuden määrittämisen automaattisesti.

Kierroslukumittari koostuu taustasta, reunuksesta, asteikosta, otsikkotekstistä sekä osoittimesta. Canvaksen luonnin jälkeen nämä osat muodostetaan. Kaikille tehdään Paint-muuttuja, mikä määrittää tyylin, värin ja muita ominaisuuksia kuten reunanpehmennyksen. Alla on esitetty esimerkkinä osoittimen Paint-muuttuja.

```
handPaint = new Paint();
handPaint.setAntiAlias(true);
handPaint.setColor(0xff000000);
handPaint.setStyle(Paint.Style.FILL);
```

Osille määritetään myös paikat canvakselle. Paikkatiedot tehdään joko RectF- tai Path-muuttujilla. RectF-muuttujalla luodaan canvakselle nelikulmio, jonka reunoihin voidaan viitata. RectF-muuttujalla luodaan myös ympyrät. Otsikko ja osoitin luodaan Path-muuttujalla, koska paikka voidaan määrittää tarkemmin.

Ensiksi määritetään reunuksen (rim) RectF-muuttuja. Reunus on uloin piirto-osa ja siksi helpoin määrittää. Reunuksen paikkatietoa käytetään mittaritaustan paikan määrittämiseen, jota taas käytetään muiden RectF-muuttujien luomiseen. Osat siis luodaan ulkoa sisäänpäin. Tällöin ne ovat oikein suhteessa toisiinsa. Osoittimen paikka on poikkeus, se tehdään käyttäen apuna canvaksen keskikohtaa. Osoitin piirretään kärki ylöspäin. Osoittimen Path-muuttuja on esitelty alla.

```
handPath = new Path();
handPath.moveTo(0.5f, 0.5f + 0.08f);
handPath.lineTo(0.5f - 0.010f, 0.5f + 0.08f);
handPath.lineTo(0.5f - 0.005f, 0.5f - 0.33f);
handPath.lineTo(0.5f + 0.005f, 0.5f - 0.33f);
handPath.lineTo(0.5f + 0.010f, 0.5f + 0.08f);
handPath.lineTo(0.5f, 0.5f + 0.08f);
```

Asteikko on monimutkaisin piirrettävä. Asteikko on tehty siten, että arvojen muuttaminen käy muuttujia säätämällä. Myöhemmissä sovelluksen versioissa käyttäjä voi muokata asteikon ajoneuvonsa mukaan. Asteikon luomiseen tarvitaan seuraavat parametrit: merkkiviivojen määrä (totalNotches), pitkien ja lyhyiden merkkiviivojen välit kierroslukuina (incrementPerLargeNotch, incrementPerSmallNotch), merkkiviivojen välit asteina (degreesPerNotch) sekä asteikon minimi, maksimi ja keskikohta kierroslukuna (scaleMin/Max/CenterValue).

totalNotches-muuttuja määrittää kuinka monta merkkiviivaa on koko ympyrän alueella. Kierroslukumittarin tapauksessa loput muuttujat määrittävät, että merkkiviivoja ei tulekaan niin paljon kuin totalNotches määrittää. Tällöin asteikko ei täytä koko ympyrää ja näyttää siksi enemmän tavanomaiselta kierroslukumittarilta.

drawScale-metodissa piirretään ensiksi asteikon kaari. Tämän jälkeen siirrytään for-silmukkaan, jossa piirretään merkkiviivat yksi kerrallaan ja asteikon tunnukset isojen merkkiviivojen viereen. Silmukassa annetaan ensin merkkiviivalle todellista kierroslukua vastaava arvo notchToValue-metodissa, joka on esitelty seuraavaksi.

```
private int notchToValue(int value) {
  int rawValue = ((value < totalNotches / 2) ? value : (value - totalNotches)) *
                                                    incrementPerSmallNotch;
  int shiftedValue = rawValue + scaleCenterValue;
  return shiftedValue;
}
```

Silmukassa käydään lävitse totalNotches-muuttujan määrittelemä määrä tapauksia. Jokaisen arvon kohdalla katsotaan kuuluuko notchToValue-metodissa saatu kierroslukuarvo scaleMinValue ja scaleMaxValue määrittelemän asteikon sisään. Jos ei kuulu, siirrytään seuraavaan merkkiviivaan. Jokaisen tapauksen jälkeen canvasta käännetään asteissa degreesPerNotch-muuttujan mukaan. Viivat siis piirretään aina samaan kohtaan, mutta piirtoalustaa käännetään. drawScale-metodi esitellään alla.

```
canvas.drawOval(scaleRect, scalePaint);
canvas.save(Canvas.MATRIX_SAVE_FLAG);
for (int i = 0; i < totalNotches; ++i) {
  float y1 = scaleRect.top;
  float y2 = y1 + 0.028f;
  float y3 = y1 + 0.045f;
  int value = notchToValue(i);
  if (i % (incrementPerLargeNotch/incrementPerSmallNotch) == 0) {
    if (value >= scaleMinValue && value <= scaleMaxValue) {
      if (value < redline) canvas.drawLine(0.5f, y1, 0.5f, y3, scalePaint);
      else canvas.drawLine(0.5f, y1, 0.5f, y3, scalePaint2);
      int tmpValue = value;
      tmpValue = tmpValue/1000;
      String valueString = Integer.toString(tmpValue);
      valueString = valueString.replaceAll("10", "");
      valueString = valueString.replaceAll("0", "");
      canvas.drawText(valueString, 0.5f, y3 + 0.08f, scaleTextPaint);
      }
    }else{
       if (value >= scaleMinValue && value <= scaleMaxValue) {
         if (value < redline) canvas.drawLine(0.5f, y1, 0.5f, y2, scalePaint);
         else canvas.drawLine(0.5f, y1, 0.5f, y2, scalePaint2);
       }
     }
  canvas.rotate(degreesPerNotch, 0.5f, 0.5f);
}
canvas.restore();
```

Kun silmukassa päästään määritellylle asteikolle, aletaan niitä myös piirtää canvakseen. Jos notchToValue-metodista saatu arvo tietylle merkkiviivalle on jokin tasatuhat kierroksista, piirretään pitkä merkkiviiva sekä asteikon tunnus. Asteikon tunnusta muokataan siten, että näyttöön piirretään vain ensimmäinen numero tasatuhansista. Tällöin kierroslukumittari näyttää selkeämmältä. 200 kierroksen välein piirretään vain lyhyt viiva ilman tunnusmerkintää. Kun kierroslukuarvot saavuttavat redline-integermuuttujan, piirretään merkkiviivat punaisena.

Kun tieto kierrosluvusta saadaan, lähetetään se RevCounter-luokan receiveRPM- metodiin, joka asettaa tiedon paikalliseksi muuttujaksi ja lähettää invalidate-komennon. invalidate käskee koko piirron (View) päivittyä. Päivitys tarkoittaa, että onDraw-metodi käydään lävitse.

onDraw-metodi käynnistää drawHand-metodin ja pitää huolen siitä, että kierroslukumittarin piirto ei syö liikaa prosessointitehoa. Se tehdään rajoittamalla ruudunpäivitysnopeutta. Ruudunpäivitysnopeus on määritetty siten, että ennen ja jälkeen piirron otetaan aika. Jos aika on alle 16 ms (noin 60 ruudunpäivitystä per sekunti, frames per second), nukkuu ohjelma jäljellä olevan ajan. onDraw-metodi on esitelty alla.

```
@Override
protected void onDraw(Canvas canvas) {
  long startTime = System.currentTimeMillis();
  drawBackground(canvas);
  float scale = (float) getWidth();
  canvas.save(Canvas.MATRIX_SAVE_FLAG);
  canvas.scale(scale, scale);
  drawHand(canvas);
  canvas.restore();
  long endTime = System.currentTimeMillis();
  long deltaT = endTime - startTime;
  if (deltaT < 16) {
    try {
      Thread.sleep(16 - deltaT);
    } catch (InterruptedException e) {}
  invalidate();
}
```

drawHand-metodi piirtää ensiksi ulkonäkösyistä paikallaan pysyvän ympyrän mittarin keskelle (ks. kuvio 12). revsToAngle-metodi kutsutaan seuraavaksi. Tämä metodi palauttaa float-muuttujan, joka vastaa tämän hetkistä kierroslukua asteina suhteessa asteikon keskiarvoon (scaleCenterValue-muuttuja). Osoitin piirretään aina pystysuoraan (keskikohta). Float-muuttujaa käytetään kääntämään canvasta sen verran, että oikeasti osoitin piirtyykin kierroslukua vastaavaa kohtaan asteikolla, kun canvas palautetaan oikeaan asentoon. drawHand-metodin esittely kokonaisuudessaan seuraavaksi.

```
private void drawHand(Canvas canvas) {
  if (handInitialized) {
    canvas.drawCircle(0.5f, 0.5f, 0.028f, handPaint);
    float handAngle = revsToAngle(handPosition);
    canvas.save(Canvas.MATRIX_SAVE_FLAG);
    canvas.rotate(handAngle, 0.5f, 0.5f);
    if (handPosition < redline) canvas.drawPath(handPath, handPaint);
    else canvas.drawPath(handPath, handPaint2);
    canvas.restore();
  }
}
```

Osoittimen Paint-muuttuja vaihdetaan, jos kierrokset ovat yli redline-muuttujassa asetetun arvon. Osoitin piirretään tällöin punaisena (ks. kuvio 12). Osoitinta ei piirretä ollenkaan ennen kuin yhteys ajoneuvoon on luotu ja ensimmäinen viesti kierrosluvusta on vastaanotettu. Käyttäjä voi myös päätellä osoittimen ilmaantumisesta, että yhteys ajoneuvoon on kunnossa.

KUVIO 12. Sovellus: Perustoimintatila, kierroslukumittari punarajalla

4.2.9 Layout

Layoutit tehdään xml-tiedostoina. MainActivityn layout on lineaarinen layout, jossa on kolme objektia. Jokaiselle objektille annetaan id, jolla niihin voidaan viitata koodissa.

Ylimpänä layoutissa on kierroslukumittari, jolle tarvitsee vain antaa canvaksen leveysarvo, paino (weight) ja keskittää se näyttöön. Paino on arvo, jolla voidaan määrittää objektien suhteellista kokoa näytössä.

Jännite piirretään kierroslukumittarin alle. Jännite kirjoitetaan TextView-objektiin. Koska TextView-objektin viereen haluttiin akun kuva ImageView-objektina, täytyi ne sisällyttää TableLayouttiin. Näin objektit saatiin rinnakkain. Painoarvolla säädettiin kierroslukumittarin ja TableLayoutin suhde toisiinsa. Seuraavalla sivulla on esitetty MainActivityn layout.xml-tiedosto.

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical" >

      <mectronic.fi_obd2tomobile.RevCounter
         android:id="@+id/revcounter"
         android:layout_width="250dp"
         android:layout_height="0dip"
         android:layout_gravity="center"
         android:layout_weight="8"    />

      <TableLayout
        android:layout_width="fill_parent"
        android:layout_height="30dp"
        android:layout_weight="1"
        android:stretchColumns="1"
        android:shrinkColumns="0"   >

      <TableRow >
        <ImageView
          android:id="@+id/image_battery"
          android:src="@drawable/battery"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_gravity="center"
          android:layout_marginLeft="40dp"      />
        <TextView
          android:id="@+id/voltage"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_gravity="center|left"
          android:textColor="@color/white"
          android:textSize="40dp"     />
       </TableRow>
    </TableLayout>
</LinearLayout>
```

Bluetooth-laitteiden listauksessa ja vikakoodien esityksessä on myös omat layoutinsa. Niissä käytetään vain yksinkertaisia Text- ja ListView:ä peräkkäin.

4.2.10 Vikakoodien luku ja esitys

Käyttäjä aloittaa vikakoodien lukemisen valitsemalla pääikkunan valikosta ’Lue vikakoodit’-kohdan (ks. kuvio 8). Tällöin pysäytetään getData-aliohjelma ja lähetetään ajoneuvolle tiedustelu vikakoodien lukumäärästä ja MIL-valon tilasta. Seuraavalla sivulla on ote MainActivityn valikon koodista.

```
case R.id.menu_dtc:
  if (mBTService.getState() == connected && initialized == true) {
    read_dtc = true;
    sleep(1500L);
    final String codes = "0101"+"\r";
    byte[] dtc = codes.getBytes();
    mBTService.write(dtc);
  }
return super.onOptionsItemSelected(item);
```

Ajoneuvon vastaus tulee käsittelijän kautta dtcCount-metodiin. Vastauksesta saadaan vikakoodien lukumäärä tarkastelemalla tiettyä kahta heksalukua, mitkä erotellaan metodissa heti alkuun. Vikakoodien lukumäärä saadaa suoraan muuttamalla heksaluvut kymmenjärjestelmään. Mikäli luku on yli 128, on MIL-valo päällä. Tällöin luvusta vähennetään 128 ja vikakoodien lukumäärä saadaan erotuksena. Integer-muuttuja lines:iin määritellään kuinka monta riviä vastauksia on odotettavissa ajoneuvolta (3 vikakoodia mahtuu yhteen riviin). Tämä tieto nopeuttaa vikakoodien hakemista ajoneuvolta myöhemmässä vaiheessa. Mikäli vikakoodeja ei ole, kerrotaan se käyttäjälle Toast-viestillä ja getData-aliohjelman toiminta aloitetaan uudestaan. Jos vikakoodeja löytyy, kysytään ajoneuvolta tarkat tiedot vikakoodeista. dtcCount-metodi on esiteltynä seuraavaksi.

```
private void dtc_count(String dtcCount) {
  dtcCount = cleanString(dtcCount);
  int ind = dtcCount.indexOf("4101");
  String dtc = dtcCount.substring(ind+4, ind+6);
  troublecodes = Integer.parseInt(dtc, 16);
  if (troublecodes != 0) {
    if (troublecodes > 128) {
      troublecodes = troublecodes - 128;
      MIL = true;
    }
    if (troublecodes > 3) {
      lines = (int) Math.ceil(troublecodes/3.0);
    } else lines = 1;
  final String mode3 = "03"+ lines + "\r";
  byte[] rdtc = mode3.getBytes();
  mBTService.write(rdtc);
  } else {
      Toast.makeText(context, context.getString(R.string.no_dtc),
        Toast.LENGTH_SHORT).show();
      read_dtc = false;
      new getData().execute();
    }
}
```

Mahdollinen vastaus vikakoodeista siirtyy käsittelijän kautta diagnostic_trouble_codes-metodiin. Alkuun luodaan StringBuilder-komponentti. Sillä kasataan String-muuttuja, jossa vikakoodit ovat peräkkäin ilman turhaa informaatiota.

Koska CAN-väylän viestirakenne eroaa muista protokollista, on String-muuttujan kasauksessakin eroja.

Jos ajoneuvo on CAN-väyläinen, viestin alkuun tulee PCI-tavu, mistä saadaan viestin tavujen lukumäärä (luvut 2.5.2 ja 4.2.5). Tällä tiedolla varmistetaan, kuinka monta riviä vikakoodeja vastauksessa on. Muissa protokollissa kuin CAN-väylässä, tunnistustavu ’43’ esiintyy jokaisen rivin alussa. ELM327-piiri hoitaa CAN-väylän viestit hieman erilailla. PCI-tavun jälkeen rivit ovat numeroitu. Ensimmäisen rivin alussa on ’0:’-merkintä, toisen ’1:’ ja niin edelleen. Tunnistustavu ’43’ on vain ensimmäisen rivin alussa. Muissa vikakooditieto jatkuu suoraan kaksoispisteen jälkeen. Sovelluksessa etsitäänkin kaksoispistettä ja lisätään vikakoodit StringBuilderiin sen perusteella. Seuraavaksi on esitelty CAN-väylän vikakoodien luenta diagnostic_trouble_codes-metodissa.

```
int cap = lines * 20;
StringBuilder koodit = new StringBuilder(cap);
if (CAN == true) {
  String  bytecount = dtc_response.substring(0, 3);
  int count = Integer.parseInt(bytecount,16);
  count = (int) Math.ceil(count/14.0);
  String  dcolon =":";
  int c_ind = dtc_response.indexOf("43");
  koodit.append(dtc_response.substring(c_ind+2));
  if (count > 1) {
    for (int i = 1; i < count; ++i) {
      int d_ind = koodit.indexOf(dcolon);
      koodit.delete(d_ind-1, d_ind+1);
    }
}
```

Muissa protokollissa käytetään samaa StringBuilderia, mutta etsitään aina tunnistustavua ’43’. Alla esiteltyä for-silmukkaa käytetään lines-muuttujan lukumäärän verran.

```
for (int u = 1, o = 0; u <= lines; ++u, o+=14) {
  int ind = dtc_response.indexOf("43",o);
  String temp = dtc_response.substring(ind+2,ind+15);
  koodit.append(temp);
}
```

Lopuksi protokollasta välittämättä StringBuilderista tehdään uusi muuttuja ja se lisätään Bundleen. Bundleen laitetaan myös dtcCount-metodista saadut MIL-valon tila sekä vikakoodien lukumäärä. Bundle on muuttujien joukko, jolla voi helposti siirtää muuttujia luokasta toiseen. startActivityForResult-komennolla käynnistetään uusi ali-

ohjelma nimeltä ReadDTC, johon myös kyseinen Bundle lähetetään. Alla on esitelty StringBuilderin ja Bundlen kasaus sekä ReadDTC-luokan kutsu.

```
String faults = koodit.toString();
faults = cleanString(faults);
Intent showDTCs = new Intent(this, ReadDTC.class);
Bundle variables = new Bundle();
variables.putBoolean("vikavalo", MIL);
variables.putInt("dtc", troublecodes);
variables.putString("vikakoodit", faults);
koodit.delete(0, koodit.length());
showDTCs.putExtras(variables);
startActivityForResult(showDTCs,DTC_DONE);
```

ReadDTC-luokassa vikakoodit erotellaan ja näytetään käyttäjälle. Vikakoodit listataan ArrayAdapteriin ja näytetään käyttäjälle ListViewinä samanlailla kuin Bluetooth-laitteet listattiin käyttäjälle lukulaitteen valinnassa. Aluksi Bundle puretaan paikallisiksi muuttujiksi. MIL-valon tila tarkastetaan ja CHECK ENGINE-tekstin väri muutetaan sen mukaan. Seuraavaksi on esitetty toteutus.

```
Intent showDTC = getIntent();
Bundle variables = showDTC.getExtras();
Boolean MIL = variables.getBoolean("vikavalo");
int troublecodes = variables.getInt("dtc");
String vikakoodit = variables.getString("vikakoodit");
if (MIL == true) {
  TextView vikavalo = (TextView) findViewById(R.id.MIL);
  vikavalo.setTextColor(this.getResources().getColor(R.color.orange));
}
```

Tämän jälkeen aloitetaan vikakoodien muokkaus luettavaan muotoon. Koska ELM327-piiri lähettää vikakoodit heksalukuina, on viestin muokkaaminen ymmärrettäviksi vikakoodeiksi hyvin yksinkertaista. Vikakoodit ovat neljän merkin ryppäissä peräkkäin ja vain ensimmäinen merkki tarvitsee muuttaa. Loput kolme ovat suoraan vikakoodin osia.

Vikakoodit muodostetaan jälleen StringBuilderilla ja for-silmukalla. Ensiksi ensimmäinen merkki lähetetään getPrefix-metodiin, joka tunnistaa merkin ja lähettää takaisin oikean etupäätteen vikakoodille. Lopuksi ei tarvitse kuin lisätä tähän päätteeseen loput kolme merkkiä vikakoodista. Näin tehdään jokaiselle vikakoodille. ELM327-piiri täyttää jokaisen vajaaksi jäävän rivin nollilla, joten sellaiset tapaukset pitää poistaa käyttäjälle näkyvästä listauksesta. Vikakoodit kasaavat for-silmukka ja getPrefix-metodi on esitetty seuraavalla sivulla.

```
for (int y = 0, x = 0; y < troublecodes; ++y, x+=4) {
  char firsthex = Character.valueOf(vikakoodit.charAt(x));
  String dtc = new String (vikakoodit.substring(x+1, x+4));
  String prefix = getPrefix(firsthex);
  StringBuilder sb = new StringBuilder(5);
  sb.append(prefix);
  sb.append(dtc);
  String fault = sb.toString();
  sb.delete(0,sb.length());
  if (fault.indexOf("0000") == -1) dtcArrayAdapter.add(fault);
}


private String getPrefix(char value) {
  String prefix = null;
  int a = Character.getNumericValue(value);
  switch (a) {
    case 0: prefix = "P0"; break;
    case 1: prefix = "P1"; break;
    case 2: prefix = "P2"; break;
    case 3: prefix = "P3"; break;
    case 4: prefix = "C0"; break;
    case 5: prefix = "C1"; break;
    case 6: prefix = "C2"; break;
    case 7: prefix = "C3"; break;
    case 8: prefix = "B0"; break;
    case 9: prefix = "B1"; break;
    case 10: prefix = "B2"; break;
    case 11: prefix = "B3"; break;
    case 12: prefix = "U0"; break;
    case 13: prefix = "U1"; break;
    case 14: prefix = "U2"; break;
    case 15: prefix = "U3"; break;
  }
return prefix;
}
```

Kun käyttäjä painaa takaisin-nappia matkapuhelimestaan, asetetaan ReadDTC aliohjelma suoritetuksi ja palataan takaisin MainActivityyn. Palaaminen tapahtuu onActivityResult-metodin kautta requestCode-parametrin ollessa DTC_DONE (luku 4.2.2). Tällöin käynnistetään getData-aliohjelma uudestaan.

4.2.11 Parannusehdotuksia

Ohjelmassa on yksi ratkaisematon ongelma, mikä on käsittelijän roskankeruu. Kääntäjäkin huomioi käsittelijän ohessa, että käsittelijän pitäisi olla static-muuttuja, Muuten roskatietoa saattaa pakkaantua ja se saattaa aiheuttaa muistivuotoa. Avuksi kääntäjä tarjoaa, että käsittelijä sisällytettäisiin ulompaan luokkaan ja siihen luotaisiin heikkoyhteys (WeakReference) . Lukuisista yrityksistä huolimatta tällaista rakennetta ei saatu toimimaan. Käsittelijä on kuitenkin toiminut oikein ilmankin.

Tärkein parannus olisi mielestäni parempi sovelluksen elämänkaari (Android Lifecycle). Kun käyttäjä siirtyy sovelluksesta toiseen tai vastaa esimerkiksi puhelimeen sovelluksen ollessa käynnissä, sovellus siirtyy eri tiloihin. Nämä tilat tulisi tarkasti määrittää, jotta sovellus ei jumiutuisi. Tässä versiossa elämänkaaren ohjelmointi on vielä kohtalaisen vähäistä. Bluetooth-yhteyden katkaisu sovelluksesta poistuttaessa on kuitenkin sisällytetty.

Ajoneuvon sovellukseen yhdistävää metodia (bus_init) voisi parantaa. Tällä hetkellä sovellus lähettää ajoneuvolle komennon ja vain toivoo, että ajoneuvo vastaa. Protokollavalinta pidetään automaattisena. Vähäisten testien perusteella voidaan sanoa, että sovellus toimii hyvin, mutta yhdistämisen voisi mahdollisesti tehdä paremminkin. Esimerkiksi ajoneuvoon voitaisiin yrittää yhdistää käyttäen tiettyä protokollaa ja tulkita piiriltä tulevaa viestiä. ELM327-piiriltä saadaan enemmän tietoa yhdistämisprosessista, kun käytetään tiettyä protokollaa automaattisen valinnan sijaan. Tällainen prosessi kuitenkin hidastaa sovelluksen alkurutiineja.

Kierroslukumittarissa on paljon parantamisen varaa. Tällä hetkellä osoittimen liike ei ole animoitu millään tavalla, joten sovellus piirtää osoittimen aina uudestaan eri kohtaan. Käytännössä osoitin ei liiku, vaan hypähtelee. Käyttäjälle voisi myös antaa mahdollisuuden muokata mittarin asteikkoa.

Sovellus on tällä hetkellä optimoitu lukulaitteen prototyypin rajoittamille nopeuksille. Massatuotantomallissa nopeutta voidaan kasvattaa ja tästä syystä sovellustakin pitäisi muokata. Jos tiedonvastaanoton ohjelmointi tehtäisiin siten, että uusi kysely lähetetään vasta, kun edellinen viesti on vastaanotettu, voitaisiin tehdä sovelluksesta lukulaitteen tiedonsiirtonopeudesta riippumaton.

Sovellus on tietysti vielä lastenkengissä, joten ominaisuudet ovat hyvin vähäiset. Sovellus ei käytä hyväkseen lukulaitteen koko potentiaalia ja lisää ominaisuuksia saataisiin hyvinkin helposti. Tällä hetkellä sovellus on ainoastaan suomenkielinen. Tukea muille kielille olisi hyvä lisätä.

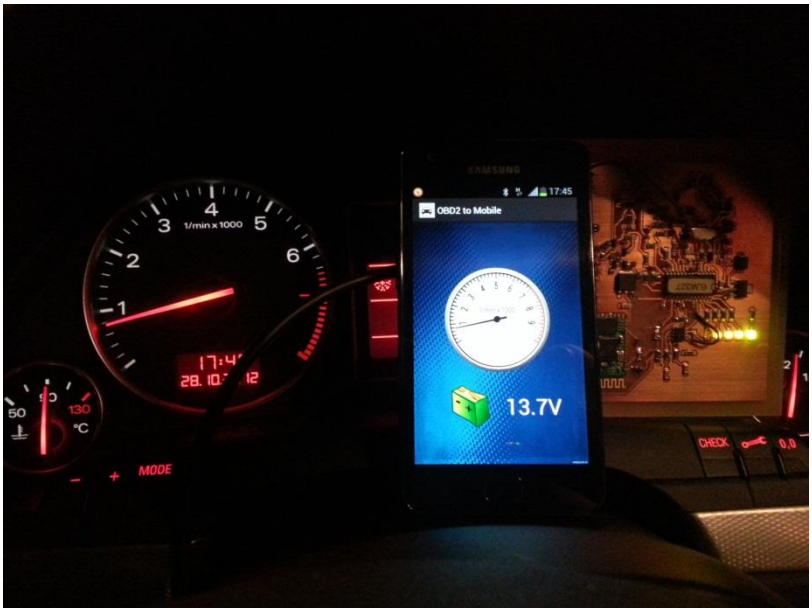
4.2.12 Sovelluksen yhteenveto

Sovelluksen luonti onnistui ilman suurempia ongelmia ja kohtalaisen nopealla tahdilla. Kaikki kriteerit, mitä asetettiin, saatiin myös täytettyä. Loppujen lopuksi täytyy olla tyytyväinen lopputulokseen. Sovellus on hyvä pohja jatkokehitykselle hyvän muunneltavuuden ja jatkokehitystä silmälläpitäen toteutetun ohjelmoinnin takia.

5 OHJELMAN JA LUKULAITTEEN TESTAUS

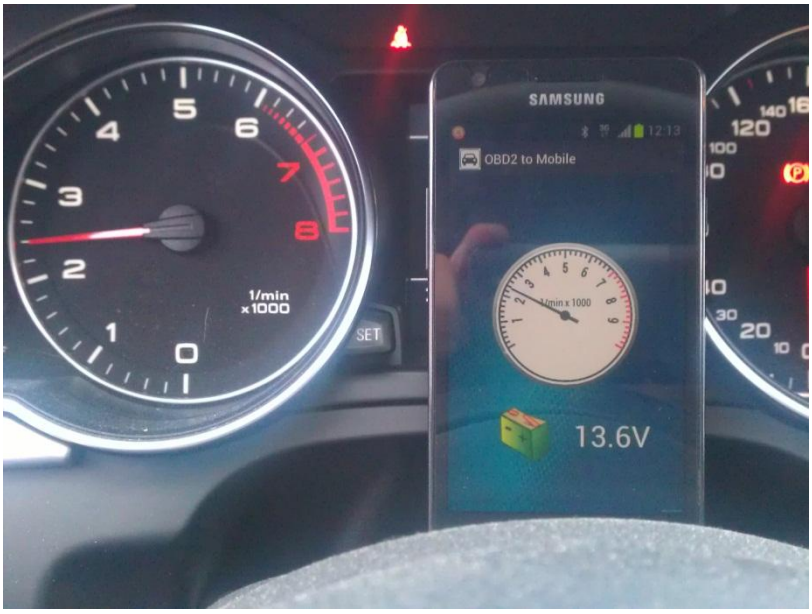
Lukulaite ja sovellus testattiin yhdessä neljässä eri ajoneuvossa ja kolmessa eri matkapuhelimessa.

Ensimmäisessä testitapauksessa käytettiin matkapuhelimena Samsung Galaxy S2 ja ajoneuvona vuoden 2002 bensiinikäyttöistä Audi A4:sta. Tämä testitapaus toimi myös sovelluksen kehitysalustana. Kuviossa 13 on esitetty järjestelmä toiminnassa. Samsung Galaxy S2:ssa oli testin aikana käytössä Android 4.0.4 ja näytön suhteellinen tarkkuus oli noin 219 PPI:tä (näytön koko 4,27" ja resoluutio 800x480). Ajoneuvossa on käytössä ISO14230-4 KWP2000-protokolla.

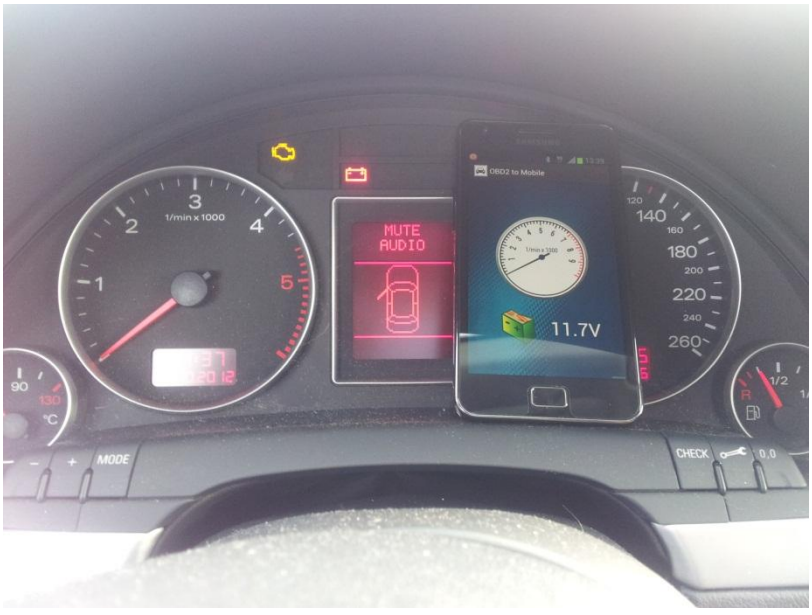


KUVIO 13. Testitapaus 1: Samsung Galaxy S2 ja Audi A4 2002

Toisessa ja kolmannessa testitapauksessa matkapuhelin pysyi samana. Toisessa testitapauksessa ajoneuvona oli bensiinikäyttöinen vuoden 2011 Audi A5 (ks. kuvio 14) ja kolmannessa testissä dieselkäyttöinen 2008 valmistettu Audi A4 (ks. kuvio 15). Testitapausten 2 ja 3 ajoneuvoissa oli käytössä ISO15765-4-protokolla eli CAN-väylä.



KUVIO 14. Testitapaus 2: Samsung Galaxy S2 ja Audi A5 2011



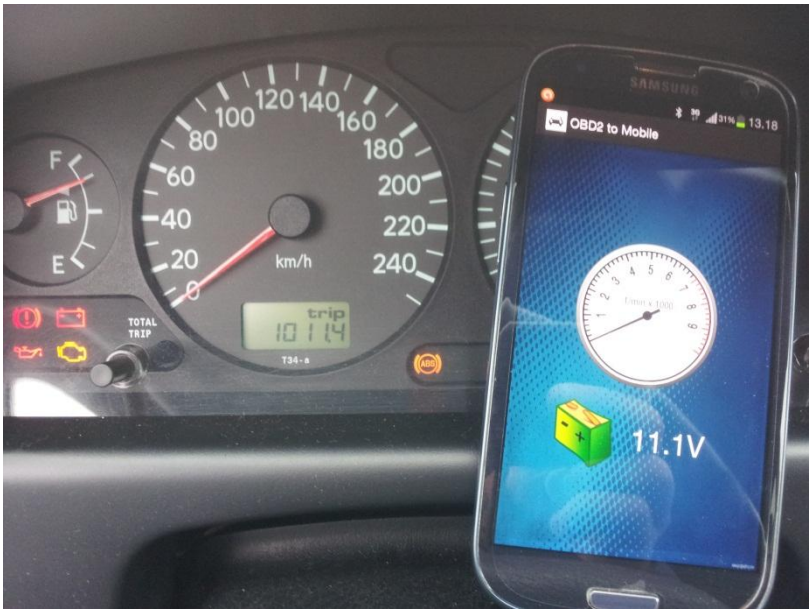
KUVIO 15. Testitapaus 3: Samsung Galaxy S2 ja Audi A4 2008

Sovellus testattiin myös vanhemmalla Android-käyttöjärjestelmällä. Testitapaus 4:ssä on testitapaus 1:stä tuttu vuoden 2002 Audi A4, mutta matkapuhelimena on HTC De-

sire HD. Desire HD:n näytön suhteellinen tarkkuus on noin 217 PPI:tä (näytön koko 4,3” ja resoluution 800x480) ja siinä oli käytössä testihetkellä Android 2.3.5. Tapaus on esitelty kuviossa 16.

KUVIO 16. Testitapaus 4: HTC Desire HD ja Audi A4 2002

Viimeisessä testitapauksessa on vuoden 2002 bensiinikäyttöinen Toyota Avensis, jossa on käytössä ensimmäisen testitapauksen Audi A4:n tavoin ISO14230-4 KWP2000-protokolla. Sovellus on asennettu Samsung Galaxy S3-matkapuhelimeen, jonka näytön suhteellinen tarkkuus on noin 306 PPI:tä (näytön koko 4,8” ja resoluutio 1280x720) ja käyttöjärjestelmänä on Galaxy S2:n tavoin Android 4.0.4. Kuviosta 17 voi nähdä kuinka sovellus toimii hyvin myös eri resoluutioisessa näytössä.



KUVIO 17. Testitapaus 5: Samsung Galaxy S3 ja Toyota Avensis 2002

Kaikki testitapaukset toimivat ongelmitta riippumatta viestintäprotokollasta ja käyttöjärjestelmän versiosta.

6 POHDINTA

Alkuperäinen tavoitteeni oli tutustua OBD2-tiedonsiirtoon, rakentaa lukulaite ja ohjelmoida sitä käyttävä sovellus matkapuhelimelle. Pääsin mielestäni tavoitteisiin erittäin mallikkaasti. Prototyyppi on toimiva ja testattu yleispäteväksi OBD2-lukulaitteeksi. Sovellus on käyttäjäystävällinen ja toimii hyvin yhteen lukulaitteen kanssa, vaikka ominaisuudet eivät ole vielä kovin kattavat. Tutkimusosuus opinnäytetyöstä oli suurilta osin jo valmiina työharjoittelun tuloksena. Hyvä teoriapohja aiheesta auttoi paljon käytännönsovelluksen rakentamisessa. Samalla teoriaosuuden paikkansapitävyyttä testattiin käytännössä.

Opinnäytetyön aikataulu venähti hieman sovelluksen ohjelmoinnin haasteiden takia. Sovelluksen tekemisessä oli paljon uusia asioita. En ollut aikaisemmin tehnyt yhtään sovellusta matkapuhelimelle eli ohjelmointialusta sekä sovelluksen käyttöjärjestelmä olivat minulle tuntemattomia. Lisäksi olin käynyt vain yhden Java-ohjelmointikurssin. Silläkään saralla en siis ollut aivan mukavuusalueellani. Opinnäytetyön tekeminen oli todellakin uusien asioiden oppimistilanne.

Oma oppimistavoitteeni oli perehtyä mahdollisimman hyvin OBD2-järjestelmään. Vaikkakin jotain käyttökokemuksia järjestelmästä olikin, en ollut aikaisemmin tutustunut siihen pintaa syvemmältä. Aihetta oli mielenkiintoista tutkia. Toinen tavoitteeni oli ohjelmoida toimiva sovellus Android-käyttöjärjestelmälle. Ylitin mielestäni itseni tämän tavoitteen kanssa. Annoin aluksi itselleni mielestäni varsin helpon ja yksinkertaisen tavoitteen sovelluksen ohjelmoinnin suhteen. Mutta todellisuudessa oppimista riittikin hyvin paljon. Piirilevysuunnitteluosiota opinnäytetyöstä en pitänyt oppimistavoitteena, vaan aikaisemmin oppimani näyttönä. Ongelmatonta piirilevysuunnittelukaan ei silti ollut, mutta ratkaisut löytyivät paljon helpommin kuin ohjelmoinnissa.

Mielipiteeni opinnäytetyön rajauksesta on hieman ristiriitainen. Koska opinnäytetyö sisältää teoriaa, piirilevysuunnittelua ja kohtalaisen pitkälle vietyä ohjelmointia, voidaan työtä pitää mielestäni erittäin laajana. Ehkä moneen seikkaan olisi voinut perehtyä vielä tarkemmin, jos aiheen rajaus olisi ollut maltillisempi. Toisaalta kaikki osiot ovat tärkeitä, jotta työn otsikon määrittämä tavoite tulisi täytettyä.

Opinnäytetyön tuloksia ei ehkä pystytä hyödyntämään sellaisenaan. Oman lukulaitteen tuotantokustannukset saattavat nousta liian korkeiksi, jotta sellaisia olisi kannattavaa tuottaa. Sovelluskaan ei vielä tässä muodossaan ole valmis tuotteeksi. Ominaisuuksia on liian vähän ja sovelluksen ulkoasu on lähinnä malliesimerkki. Lisäksi sovellusta on testattu hyvin marginaalisesti. Mutta kuten aiemmin tuli todettua, sovelluksessa on jatkokehityspotentiaalia.

Työn aihe oli minulle mieluinen. Lukulaitteen rakentaminen ja sovelluksen ohjelmointi olivat myös erittäin mielenkiintoisia. Sovelluksen viimeistely vei yllättävän paljon aikaa, sillä halusin päästä täydellisesti asetettuihin tavoitteisiin. Opinnäytetyön tekeminen oli mukava kokemus. Aihe oli hyvä ja käytännönläheinen työ sopi minulle erinomaisesti.

LÄHTEET

AN04 - ELM327 and Bluetooth. n.d. ELM Electronics. PDF-tiedosto. Viitattu 10.03.2012. http://www.elmelectronics.com/AppNotes/AppNote04.pdf.

CuteDigi BMX Bluetooth to UART/12C/USB Module (GEN II). n.d. CuteDigi.com-sivusto. Bluetooth-moduulin AT-komennot. PDF-tiedosto. Viitattu 30.9.2012. http://www.cutedigi.com/pub/Bluetooth/BMX_Bluetooth_quanxin.pdf.

ELM327 OBD to RS232 Interpreter. n.d. ELM Electronics. ELM 327 v1.4b -datalehti. PDF-tiedosto. Viitattu 13.12.2011. http://www.elmelectronics.com/DSheets/ELM327DS.pdf.

ELM327 v1.4b OBD to RS232 Interpreter. 2012. ELM Electronics. Viitattu 6.2.2012. http://www.elmelectronics.com/obdic.html#ELM327.

Eri Protokollat. n.d. Elekma.com -sivustolla. Viitattu 8.12.2011. http://www.elekma.com/eri_protokollat.

Heikkilä. J. 2007. EOBD, OBD2 ja valmistajakohtaiset protokollat. OBD.fi -sivustolla. Viitattu 5.12.2011. http://www.obd.fi/index.php?option=com_content&task=view&id=32&Itemid=2.

Ivan, D. 2006. HOWTO Read Your Car´s Mind. Viitattu 5.12.2011. http://www.thinkythings.org/obdii/#references.

jfrank. 2008. Data Link Connector. Weber State University. Viitattu 8.12.2011. http://ocw.weber.edu/automotive-technology/ausv-1320-automotive-electronics/17-serial-data/data-link-connector.

Lehtinen. A. n.d. OBD. Autotieto.net-sivustolla. Viitattu 22.12.2011. http://www.autotieto.net/pakokaasutkurssi/oppimateriaalit/obd_tietoa.htm.

Leale, R. n.d. ISO 15765-2. canbushack: Hack Your Car. Viitattu 18.12.2012. http://www.canbushack.com/blog/index.php?title=iso-15765-2-can-transport-layer-yes-it-can-be-fun&more=1&c=1&tb=1&pb=1.

Memruk. I. 2010. Android Custom UI: Making a Vintage Thermometer. Mind The Robot -sivustolla. Viitattu 15.8.2012. http://mindtherobot.com/blog/272/android-custom-ui-making-a-vintage-thermometer/comment-page-1/#comment-14326.

Noxon. J. 2009. Opendiag OBD-II Schematics & PCB Layout. Planetfall.com -sivustolla. Viitattu 17.2.2012. http://www.planetfall.com/cms/content/opendiag-obd-ii-schematics-pcb-layout.

OBD2 Codes Explained. n.d. OBD-Codes.com -sivustolla. Viitattu 22.12.2011. http://www.obd-codes.com/faq/obd2-codes-explained.php, FAQs.

OBD-II PIDs. 2010. OBD-II Resource -sivustolla. Viitattu 22.12.2011. http://obdcon.sourceforge.net/2010/06/obd-ii-pids/.

OBDII Message Structure. 2011. Obddiagnostics.com -sivustolla. Viitattu 13.12.2011. http://obddiagnostics.com/obdinfo/msg_struct.html.

OBD Interpreter IC´s. 2012. OBD Pros -sivustolla. Viitattu 6.2.2012. http://www.obdpros.com/product_info.php?products_id=139.

STN1110: Multiprotocol OBD to UART Interpreter IC. 2012. Scantool LLC. OBD Solutions -sivustolla. Viitattu 6.2.2012. http://www.obdsol.com/stn1110/.

OBD-II Protocols. n.d. OBDTester.com -sivustolla. Viitattu 8.12.2011. http://www.obdtester.com/obd2_protocols.

Suunnitteluohjeet. n.d. Prinel Piirilevy Oy. n.d. Piirilevyvalmistaja. Piirilevysuunnitte-luohjeet. Viitattu 10.8.2012. http://www.prinel.fi/suunnitteluohjeet-1.

Tikkanen, H. 2004. Piirilevysuunnitteluopas. 2. Jyväskylä: DS-Design Systems.

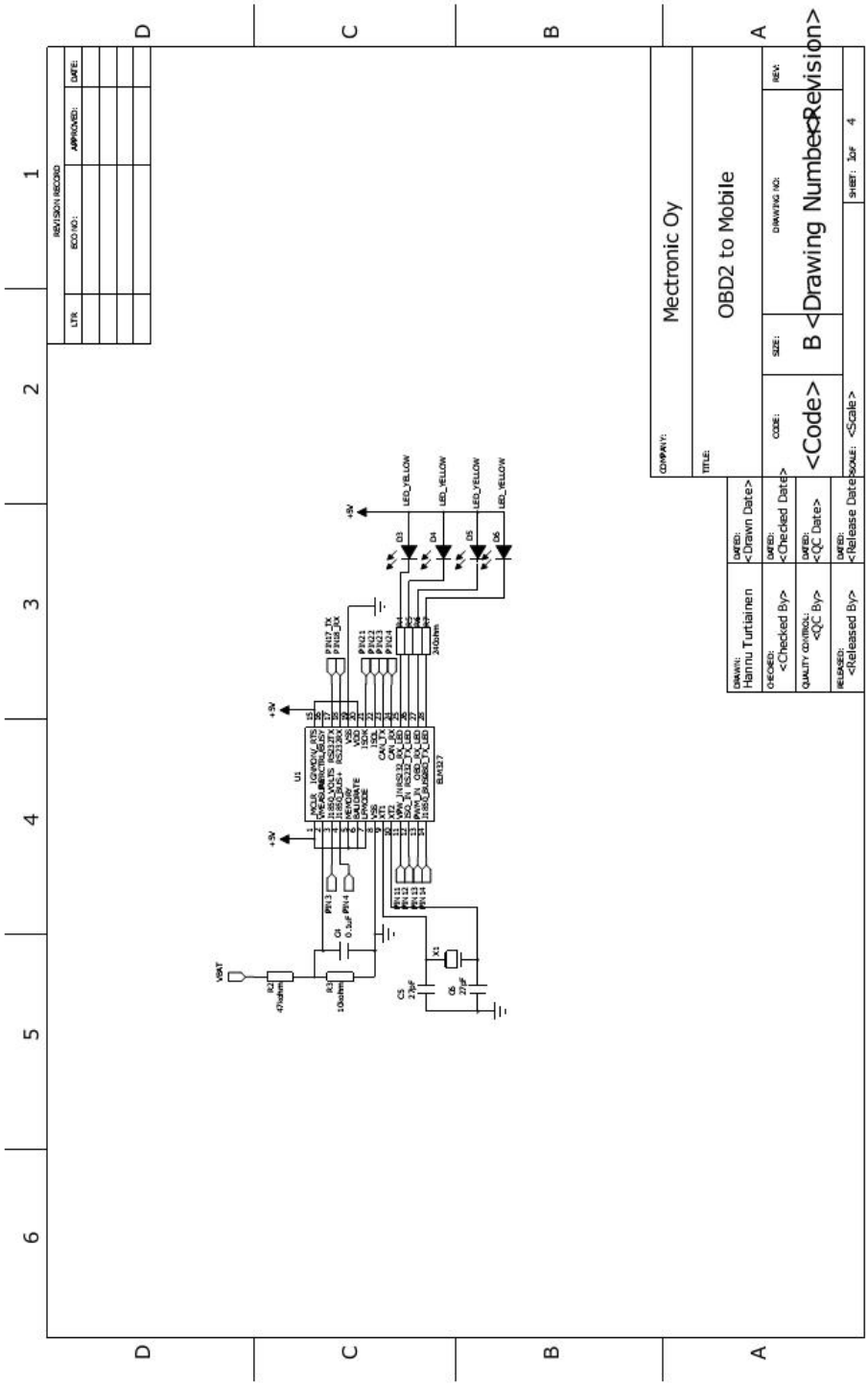
Tyson, J. n.d. How Encryption Works. Viitattu 18.12.2011. http://computer.howstuffworks.com/encryption7.htm.

What's the difference between VCDS and an OBD-II Scan-Tool?. n.d. Ross-Tech.com -sivustolla. Viitattu 22.12.2011. http://ross-tech.com/vag-com/faq_1.html#1.10.

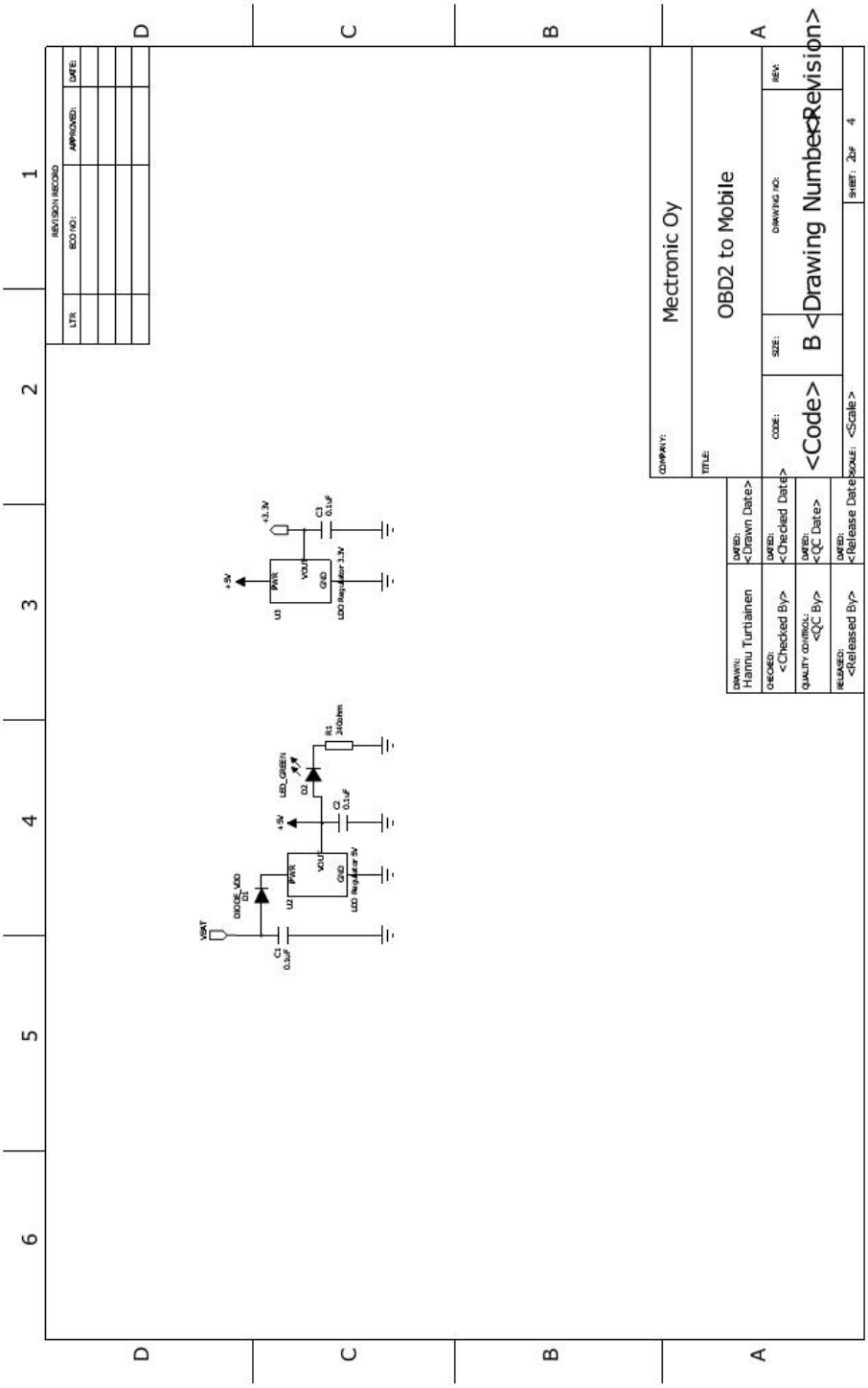
Which OBD-II protocol is supported by my vehicle?. 2004. Scantool.net -sivustolla. Viitattu 8.12.2011. http://www.scantool.net/support/index.php?_m=knowledgebase&_a=viewarticle&kbarticleid=3.

LIITTEET

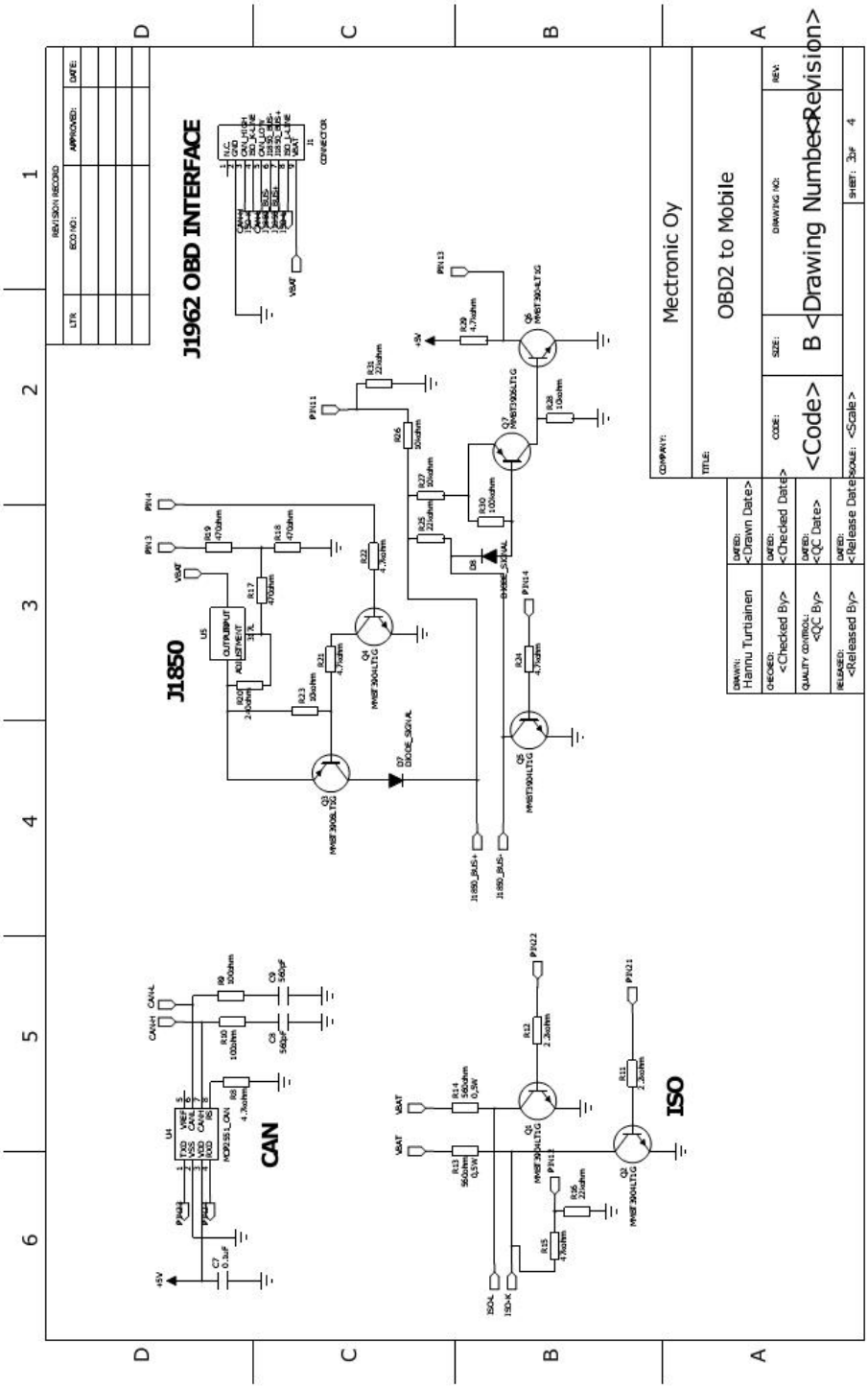
Liite 1. Lukulaitteen piirikaavio, ELM327



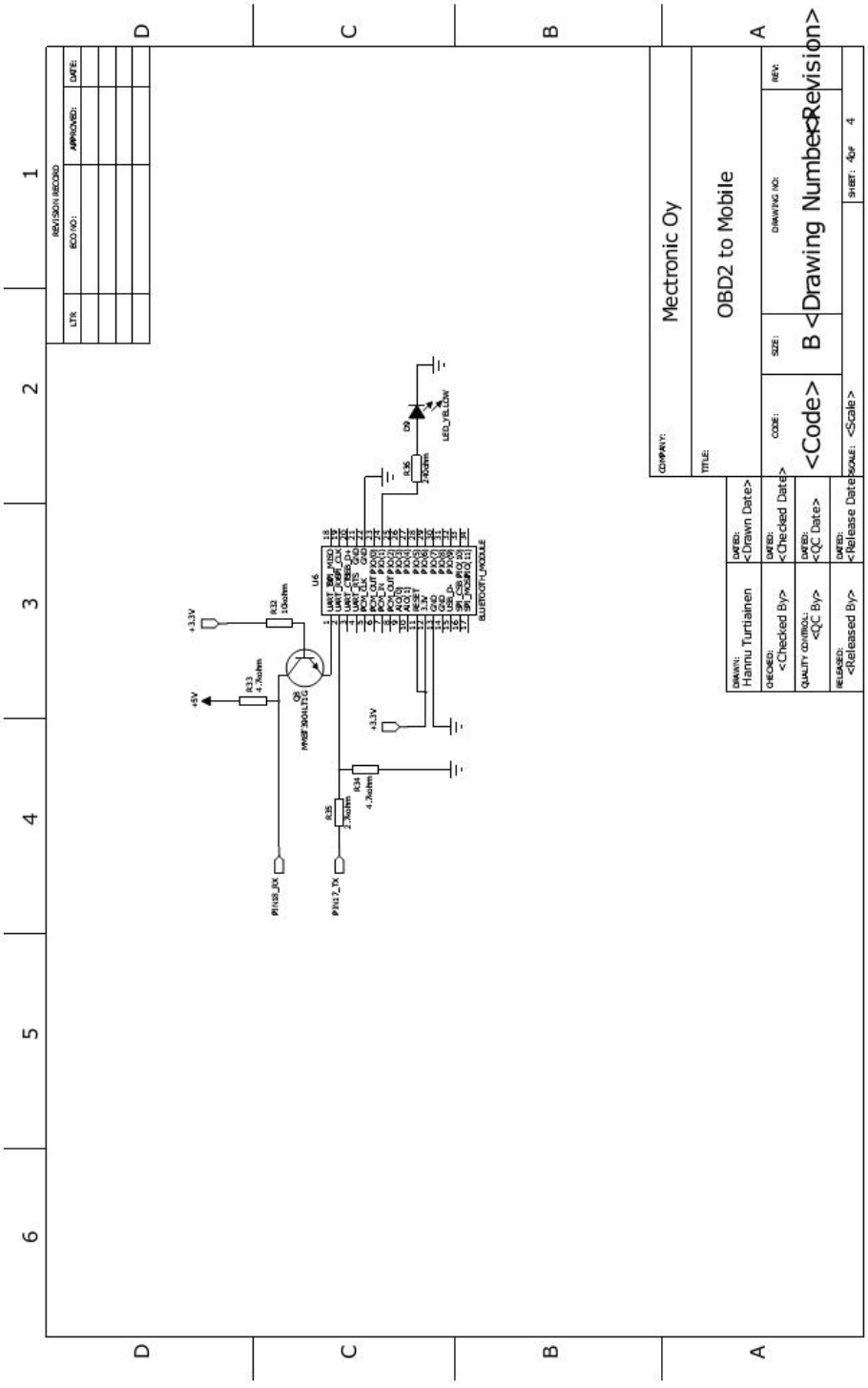
Liite 2. Lukulaitteen piirikaavio, virransyöttö



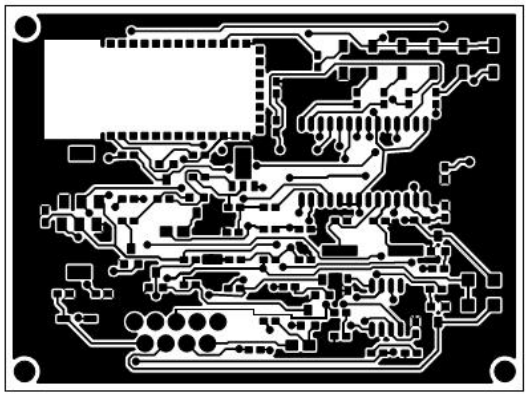
Liite 3. Lukulaitteen piirikaavio, liitännät



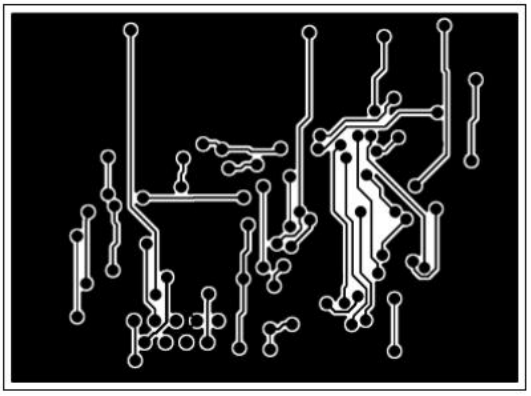
Liite 4. Lukulaitteen piirikaavio, Bluetooth



Liite 5. Massatuotantopiirilevy, päällyskerros 1:1



Liite 6. Massatuotantopiirilevy, pohjakerros 1:1



Liite 7. Massatuotantopiirilevy, kasauskaavio