Jarmo Kapli

MegaSquirtin rakenne ja Android-sovelluksen kehitys

Opinnäytetyö
Auto- ja kuljetustekniikan koulutusohjelma

Toukokuu 2012

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

KUVAILULEHTI

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Opinnäytetyön päivämäärä** 2.5.2012
Tekijä(t) Jarmo Kapli	**Koulutusohjelma ja suuntautuminen** Auto- ja kuljetustekniikka

Nimeke

MegaSquirtin rakenne ja Android-sovelluksen kehitys

Tiivistelmä

Tässä opinnäytetyössä tutkittiin moottorinohjainlaitteen toimintaa ja kehitettiin sovellus, joka kommunikoi ohjainlaitteen kanssa. Tavoitteena oli selvittää MegaSquirt-moottorinohjainlaitteen rakenne ja kehittää sovellus matkapuhelimelle, joka kommunikoi langattomasti MegaSquirtin kanssa. Sovelluksen tavoite on näyttää ajoneuvon kuljettajalle tietoa moottorinohjainlaitteelta moottorin toiminnasta.

Työssä esitellään MegaSquirt-moottorinohjainlaitetta yleisesti ja tutustutaan tarkemmin yhteen MegaSquirt-version rakenteeseen. Työssä selvitetään myös, kuinka MegaSquirtin piirilevyä täytyy muokata langatonta tiedonsiirtoa varten. Tutustun myös, kuinka Android-sovelluksia kehitetään käytännössä.

Onnistuin työssäni hyvin ja saavutin melkein kaikki työlle asetetut tavoitteet. Sain tutkittua ohjainlaitteen toimintaa hyvin, ja sain tehtyä matkapuhelimelle sovelluksen, joka kommunikoi langattomasti MegaSquirtin kanssa. Sovellus toimii luotettavasti ja kuten suunniteltukin, mutta kuitenkin esimerkiksi sovelluksen käyttöliittymään jäi kehitettävää.

Asiasanat (avainsanat)

Polttomoottorit, ohjausjärjestelmät, elektroniikka, android, java, ohjelmointi

Sivumäärä	**Kieli**	**URN**
30	Suomi	

Huomautus (huomautukset liitteistä)

Ohjaavan opettajan nimi	**Opinnäytetyön toimeksiantaja**
Harri Kosonen	

DESCRIPTION

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Date of the bachelor's thesis** 2.5.2012
Author(s) Jarmo Kapli	**Degree programme and option** Car and transportation technology

Name of the bachelor's thesis

Structure of MegaSquirt and Android-software development

Abstract

The purpose of my bachelor's thesis was to study the structure of MegaSquirt engine management unit and to develop an application for mobile phone that communicates wirelessly with the MegaSquirt. The application was designed to show the current state of the engine for the driver.

In this thesis I describe the MegaSquirt system in general, and explain the circuit board structure of one MegaSquirt version. I also explain how the circuit board must be modified to make wireless communication possible. I tell also about the Android software development.

I succeeded well and achieved almost all of the goals that were set. I learned a lot of engine management systems and managed to develop an application that communicates with the MegaSquirt. The application works fluently and as designed to, but for example the user interface needs to be improved.

Subject headings, (keywords)

Internal combustion engine, electronics, android, java, programming

Pages	**Language**	**URN**
30	Finnish	

Remarks, notes on appendices

Tutor	**Bachelor's thesis assigned by**
Harri Kosonen	

SISÄLTÖ

1 JOHDANTO

Tässä työssä halutaan tutustua polttomoottorin ohjainlaitteen toimintaan ja tehdä sovellus, joka kommunikoi moottorinohjainlaitteen kanssa. Tutkittavana moottorinohjainlaitteena on autoharrastajien suosiossa oleva MegaSquirt.

Työn tavoitteena on oppia tuntemaan MegaSquirtin toiminta. Työssä selvitetään tarkemmin yhden MegaSquirt-version ohjainlaitteen rakenne ja ohjainlaitteen komponenttien toiminta. Osana työtä tehdään sovellus matkapuhelimelle, joka lukee Mega-Squirtilta tietoa ottomoottorin toiminnasta. Sovellus kommunikoi langattomasti moottorinohjainlaitteen kanssa, mikä vaatii myös muutoksia ohjainlaitteen piirilevylle. Sovellus kehitetään Android-alustalle Java-ohjelmointikielellä. Sovelluksen tavoitteena on pyrkiä näyttämään kuljettajalle kaikki tarpeellinen informaatio moottorin sen hetkisestä toiminnasta, jotta kuljettaja tietää polttomoottorin varmasti toimivan oikein.

Aluksi työssä tutustutaan yleisellä tasolla MegaSquirtin toimintaan ja esitellään tarkemmin yhden MegaSquirt-version rakenne ja toiminta. Luvussa kaksi tutkitaan langatonta tiedonsiirtoa ohjainlaitteen ja matkapuhelimen välillä, millaisia muutoksia se vaatii ohjainlaitteen piirilevylle ja miten tiedonsiirto tapahtuu käytännössä. Tämän jälkeen tutkitaan sovelluksen kehittämistä Android-puhelimeen. Luvussa tutustutaan, miten sovellus kommunikoi MegaSquirtin kanssa, miten ohjainlaitteelta saatua tietoa joudutaan käsittelemään ja miten sitä saadaan esitettyä ymmärrettävässä muodossa kuljettajalle. Lopuksi pohditaan aihetta, kuinka työssä onnistuttiin, mitä opittiin ja mitä kehitettävää jäi.

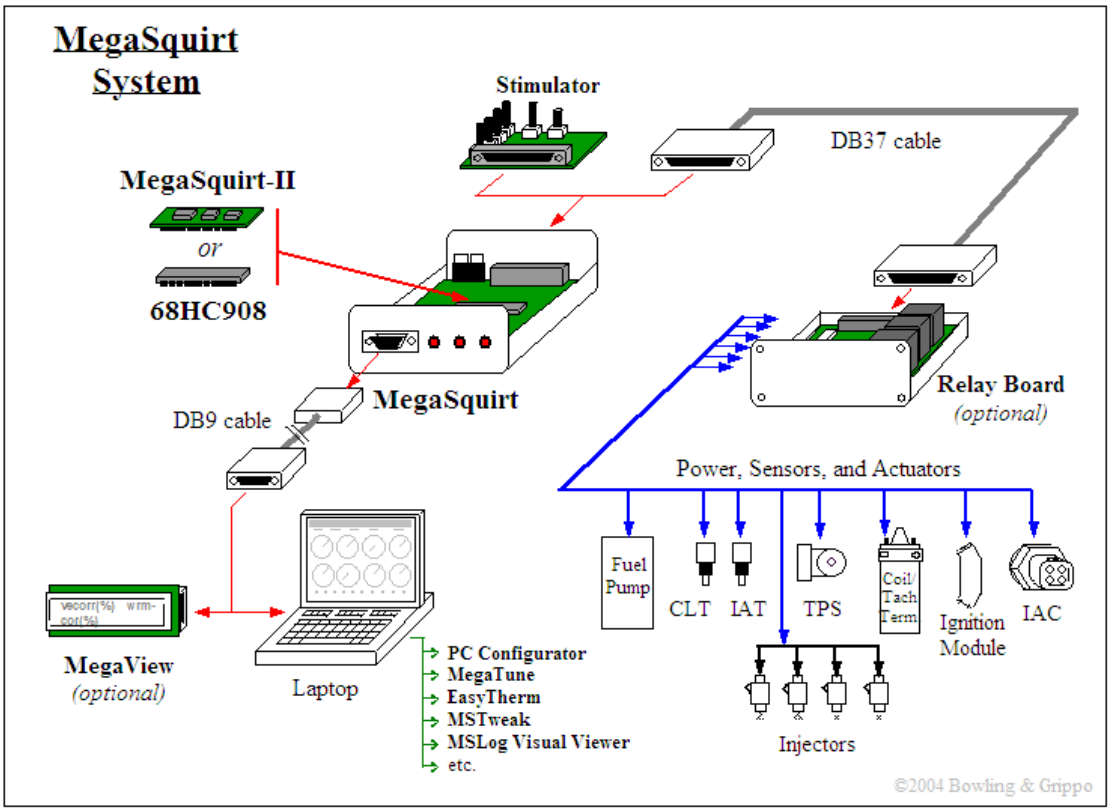
2 MEGASQUIRT

MegaSquirt on ohjelmoitava polttomoottorin ohjainlaite. MegaSquirtin on kehittänyt Bruce Bowling ja Al Grippo autoista, elektroniikasta ja tietotekniikasta kiinnostuneille tee-se-itse-miehille. Jokainen voi ostaa tarvittavat komponentit, kasata ohjainlaitteen itse, asentaa autoon ja säätää sen itse. MegaSquirtin kytkentäkaavio ja ohjainlaitteen lähdekoodi on avoimesti saatavilla, mikä selittääkin sen suosiota autoharrastajien keskuudessa. MegaSquirt asennetaan yleensä ajoneuvoon, jota viritetään suuremman tehon toivossa tai ajoneuvoa modernisoitaessa, kun kaasutin halutaan korvataan elektronisella polttoaineenruiskutuksella. MegaSquirtin voi ostaa rakennussarjana, valmiiksi kasattuna tai vaikka jokaisen komponentin erikseen ja itse valmistaa piirilevyn.

Verrattuna kaupallisiin polttomoottorin ohjainlaitteisiin MegaSquirtin muokattavuus on omaa luokkaansa. Käyttäjän itse rakennettua ohjainlaitteen alusta alkaen ymmärtää hän, miten mikäkin toimii, jolloin ohjainlaitteen muokkaaminen uusien ominaisuuksien toivossa on helppoa. Mahdollisessa vikatilanteessa ohjainlaitteen rakenteen tunteminen helpottaa myös vianetsintää huomattavasti. Toisaalta mitään virallista apua ja tukea ei ole saatavilla, kuten kaupallisiin ohjainlaitteisiin, mutta toisilta harrastajilta saa usein apua esimerkiksi internetin välityksellä.

Ottomoottorin polttoaineen ruiskutuksen ja sytytyksen säätämistä varten MegaSquirtin mikro-ohjaimelle ladataan koodi, joka laskee oikean polttoainemäärän ja sytytyshetken anturitietojen perusteella. Tyhjä mikro-ohjain tarvitsee ensin ohjelmoida PC:llä erityisen ohjelmointilaitteen avulla, minkä jälkeen sen kanssa voidaan kommunikoida suoraan sarjaportin välityksellä. Ostettaessa MegaSquirtin rakennussarjana mikro-ohjain yleensä toimitetaan myyjän toimesta valmiiksi ohjelmoituna niin, että ohjelmointilaitettakaan ei tarvita. Mikro-ohjaimessa on sisäinen EEPROM-muisti, mikä mahdollistaa moottorinohjausparametrien kirjoittamisen ja lukemisen reaaliajassa moottorin käydessä. Valmistaja lupaa EEPROM-muistin olevan uudelleenkirjoitettavissa vähintään 10 000 kertaa, ja tiedon säilyvän vähintään kahdenkymmenen vuoden ajan. /1./

MegaSquirt-järjestelmä koostuu itse ohjainlaitteesta, antureista ja toimilaitteista. Näiden lisäksi tarvitaan tietysti tietokone ohjainlaitteen säätämiseen. Moottorin toimintaan vaikuttavien parametrien säätämiseen on olemassa useita eri ohjelmistoja eri käyttöjärjestelmille, mikä tekee moottorin säätämisestä helppoa ja yksinkertaista. Etenkin aloittelevalle säätäjälle monipuolisesta säätämisohjelmasta on hyötyä, koska kehittyneet ohjelmat osaavat itse säätää esimerkiksi polttoainekartat automaattisesti lambda-tunnistimen avulla, eikä kuljettajan tarvitse kuin ajaa autolla ja seurata tilanteen kehittymistä. Säätämisohjelmista esimerkkinä mainittakoon yleisin käytetyimmät MegaTune ja TunerStudioMS. Kuvassa 1 havainnollistava kaaviokuva MegaSquirt -moottorinohjausjärjestelmästä. /1./



KUVA 1. MegaSquirt-moottorinohjausjärjestelmä kaaviokuvana

MegaSquirtista on useita eri versioita. Tällä hetkellä käytetään kolmea eri mikro-ohjainta, ja piirilevystä on seitsemän eri versiota. MegaSquirtin versio määräytyy käytetyn mikro-ohjaimen mukaan, MS1-versiossa käytetään Motorolan 8-bittistä MC68HC908-mikro-ohjainta, MS2-versiossa käytetään Motorolan 16-bittistä MC9S12C64-mikro-ohjainta ja MS3-versiossa Motorolan 16-bittistä MC9S12XEP100-mikro-ohjainta. MS1-version mikro-ohjaimen väylänopeus on 8

MHz, MS2-versiossa 24 MHz ja MS3-versiossa jo 50 MHz. MS1-versiolla pystytään ohjaamaan ainoastaan polttoaineen ruiskutusta ilman käyttäjän muokkaamista, MS2-versiosta alkaen on suoraan mahdollisuus ohjata samalla ohjainlaitteella sekä polttoaineensyöttöä ja sytytystä. MegaSquirtin versio valitaan käyttötarkoituksen mukaan, tarvitseeko sytytyksen ohjausta ja riittääkö mikro-ohjaimen laskentateho käyttötarkoitusta varten. MS1-version luvataan kuitenkin pystyvän ohjaamaan ottomoottorin polttoaineenruiskutusta moottoreissa. jotka kiertävät 16000 kierrosta minuutissa. Vertailun vuoksi esimerkiksi Boschin Motronic ME7.0 -moottorinohjainlaitteessa on 24 MHz prosessori (C167), mikä on vuodelta 1998. /2; 3, s. 37./

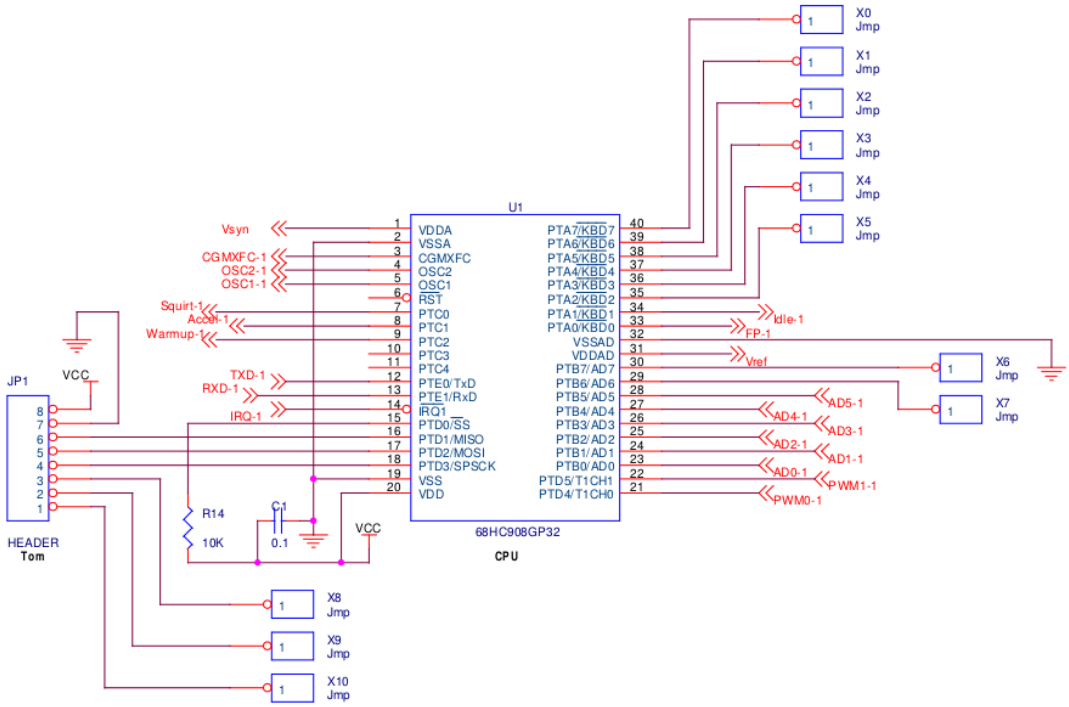
Piirilevyversioita on V1.01 (ei enää saatavilla), V2.2, V3, V3.57, MicroSquirt, MicroSquirt Module ja MS-II Sequencer. Piirilevyt ovat kehittyneet vuosien saatossa tähdäten aina ohjainlaitteen luotettavuuden parantumiseen. Komponenttien käyttölämpötila-alue on kasvanut autokäyttöön soveltuvammaksi, kestävämpi ja laadukkaampi piirilevy sekä useita komponenttien suojauksia vikatilanteita varten on lisätty. Version V3.57 piirilevy on muuten sama kuin V3-piirilevy, mutta käyttää suurimmaksi osaksi pintaliitoskomponentteja, joten se ei ole tarkoitettu itse kasattavaksi vaan toimitetaan valmiiksi kasattuna. V3.57 onkin hyvä valinta sellaiselle, joka ei halua tai pysty itse kasaamaan ohjainlaitetta.

Tässä työssä tutustutaan tarkemmin MS1V2.2 MegaSquirttiin, siis V2.2-piirilevy Motorolan MC68HC908-mikro-ohjaimella. Päädyin kyseiseen versioon, koska ohjainlaite on tarkoituksena asentaa 60-luvun harrasteajoneuvoon kaasuttimen korvaajaksi, ja suurempaa laskentatehoa ei tarvita, koska moottoria tullaan ohjaamaan ns. hukkakipinällä ja polttoaineen ruiskutus tapahtuu imusarjaan ns. jatkuvasuihkutteisesti. Hukkakipinää käyttäessä moottorin asennosta ei tarvitse tietää kuin yksi asento kampiakselin kierroksesta. MS1 on myös edullisempi verrattuna uudempiin versioihin hitaamman mikro-ohjaimen myötä. Uudemmilla versioilla saavutettaisiin tarkempi säätöresoluutio, mutta tässä tapauksessa tälläkin versiolla saavutetaan varmasti paljon paremmin toimiva moottori kuin moottori kaasuttimella varustettuna.

Ostin MegaSquirtin rakennussarjana ja kasasin sen itse. Rakennussarja sisälsi kaikki tarvittavat komponentit, piirilevyn ja valmiiksi ohjelmoidun mikro-ohjaimen. Ohjainlaitteen kasaus oli helppo ja nopea työ selkeitä kasaamisohjeita noudattamalla. MegaSquirtin koodi on päivitetty MS-Extra-koodiin, mikä mahdollistaa myös sytytyksen ohjauksen MS1-versiolla, laajemmat 12x12 polttoainekartat verrattuna alkuperäisiin 8x8 karttoihin sekä muita pienempiä uudistuksia alkuperäiseen koodiin verrattuna. Itse kasattuna ohjainlaite tuli maksamaan vain noin sata euroa.

2.1 MS1V2.2:n rakenne

MegaSquirt-ohjainlaitteen tärkein komponentti on tietysti mikro-ohjain, joka hoitaa kaiken laskentatyön. Mikro-ohjaimen tehtävä on ohjata toimilaitteita sisääntuloportteihin liitetyiden antureilta saadun tiedon perusteella. Tyypillisimpiä moottorinohjaukseen käytettyjä antureita ovat moottorin lämpötila-anturi, imuilman lämpötila-anturi, lambda-anturi, imusarjan paineanturi, kaasuläpän asentoanturi jne. Ohjattavia toimilaitteita ovat polttoaineen ruiskutussuuttimet, sytytyspuolat, joutokäyntimoottori jne. Käytettävät anturit ja toimilaitteet riippuvat tietysti täysin käyttökohteesta.



KUVA 2. MC68HC908 mikro-ohjaimen kytkentäkaavio

Kuvassa 2 näkyy tässä V2.2-piirilevyllä käytettyä MC68HC908-mikro-ohjaimen kytkentäkaavio. Kyseinen mikro-ohjain on 40-porttinen. Porttien merkitys on listattuna taulukossa 1. Kuten taulukosta näkyy, mikro-ohjaimessa on paljon portteja kytkemättä, mitä käyttäjä voi vielä helposti hyödyntää, kuten esimerkiksi ahtopaineen ohjaukseen. Mikro-ohjain tarvitsee toimiakseen viiden voltin vakavoidun jännitteen ja erillisen tahdistuspiirin, jotka on rakennettu piirilevylle erikseen.

TAULUKKO 1. Mikro-ohjaimen portit

#	**Nimi**	**Merkitys**
1	VDDA	Käyttöjännite
2	VSSA	Maa
3	CGMXFC	Kellopulssi
4	OSC2	Kellopulssi
5	OSC1	Kellopulssi
6	RST	Nollaus
7	PTC0	D17-ledin ohjaus (Squirt-1)
8	PTC1	D18-ledin ohjaus (Warmup-1)
9	PTC2	D19-ledin ohjaus (Accel-1)
10	PTC3	Ei kytkettynä
11	PTC4	Ei kytkettynä
12	PTE0/TxD	Sarjaliikenteen lähetys
13	PTE1/RxD	Sarjaliikenteen vastaanotto
14	IRQ1	Sytytyspulssi/Tahdistus
15	PTD0/SS	VSYN
16	PTD1/MISO	Ei kytkettynä
17	PTD2/MOSI	Ei kytkettynä
18	PTD3/SPSCK	Ei kytkettynä
19	VSS	Maa

20	VDD	+5V
21	PTD4/T1CH0	Suuttimien ohjaus PWM
22	PTD5/T1CH1	Suuttimien ohjaus PWM
23	PTB0/AD0	Imusarjanpaine tieto
24	PTB1/AD1	Imusarjan lämpötilatieto
25	PTB2/AD2	Jäähdytinnesteen lämpötilatieto
26	PTB3/AD3	Kaasuläpän asentotieto
27	PTB4/AD4	Akkujännite
28	PTB5/AD5	Lambda-anturi
29	PTB6/AD6	Ei kytkettynä
30	PTB7/AD7	Ei kytkettynä
31	VDDAD	Maadoitus
32	VSSAD	Referenssi jännite (+5V)
33	PTA0/KBD0	Polttoainepumppu
34	PTA1/KBD1	Joutokäyntimoottori
35	PTA2/KBD2	Ei kytkettynä
36	PTA3/KBD3	Ei kytkettynä
37	PTA4/KBD4	Ei kytkettynä
38	PTA5/KBD5	Ei kytkettynä
39	PTA6/KBD6	Ei kytkettynä
40	PTA7/KBD7	Ei kytkettynä

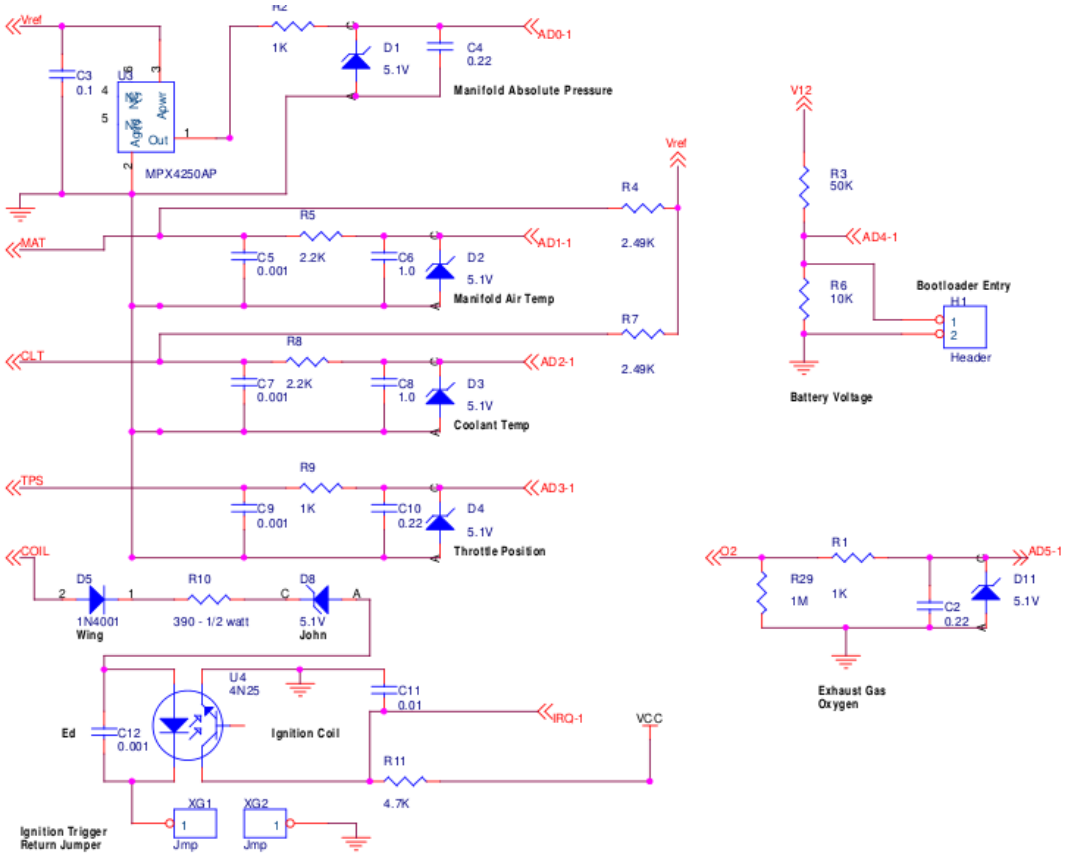
2.2 Sisääntulosignaalit

Mikro-ohjain sisältää A/D-muuntimen (Analogia-digitaalimuunnin) anturitietojen käsittelemiseen, eli anturilta tulevan analogisen jännitteen muuntamiseksi digitaaliseksi lukuarvoksi. Tämä muunnos vaaditaan siksi, että mikro-ohjaimelle

ladattu ohjelma osaa tulkita antureiden arvoja vain binäärilukuina, ja sen perusteella säätää moottorin toimintaa toimilaitteiden avulla.

Mikro-ohjaimessa on kahdeksan sisääntuloporttia A/D-muuntimella (PTB0 – PTB7). Näistä sisääntuloporteista vain kuusi on käytössä tavallisesti. Portteihin on kytketty imusarjan paineanturi, imuilman lämpötila-anturi, jäähdytinnesteen lämpötila-anturi, kaasuläpän asentotunnistin, akun jännite ja pakokaasun jäännöshappianturi, eli lambda-anturi. Kuvassa 3 näkyy sisääntuloporttien kytkentä piirilevyllä.

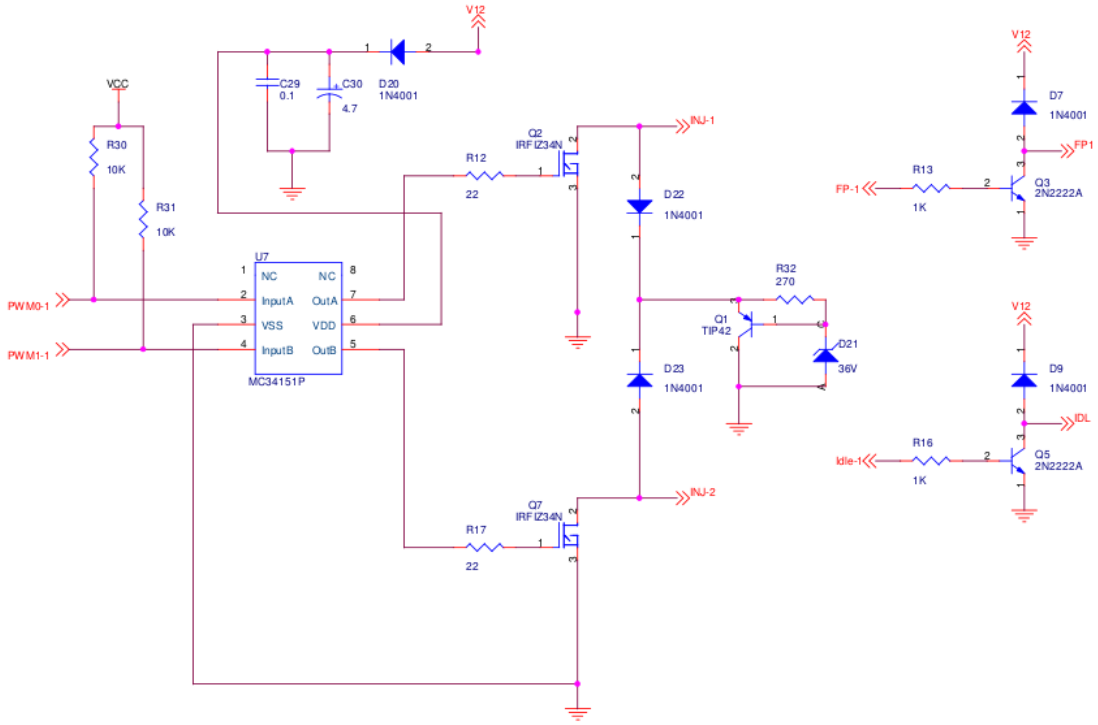
Mikro-ohjaimessa on myös IRQ-portti, eli *interrupt request* portti. IRQ-portti antaa keskeytyssignaalin mikro-ohjaimelle, mikä pakottaa mikro-ohjaimen suorittamaan tietyn toiminnon (keskeytysohjelman). MegaSquirtissa tähän porttiin kytketään polttoaineen ruiskutusta ohjatessa sytytysimpulssitieto, eli yksinkertaisimmillaan sytytyspuolan miinus-napa, minkä ansiosta mikro-ohjain saa keskeytyssignaalin aina sytytysimpulssista. Sytytysimpulsseista MegaSquirt saa laskettua moottorin kierrosnopeuden. Kun halutaan ohjata polttoaineen ruiskutuksen lisäksi myös sytytystä, täytyy IRQ-porttiin kytkeä tahdistinanturi, minkä avulla kierrosnopeuden lisäksi saadaan tietää tarkka moottorin asento tietyssä kohtaa, esimerkiksi kun ensimmäinen sylinteri on yläkuolokohdassaan. Usein tahdistinanturia käytetään kampiakselin hihnapyörään kiinnitetyn sakaralevyn asennon tunnistamiseen. Tahdistin voi sijaita myös esimerkiksi virranjakajan sisällä, nokka-akselin yhteydessä tai vauhtipyörän yhteydessä. Hyvä sijainti anturille on nokka-akselin yhteydessä, koska sen pyörintänopeus on vain puolet moottorin pyörintänopeudesta ja se on näin myös vähemmän häiriöaltis. Tahdistinanturina voidaan käyttää useaa erilaista anturia, kuten esimerkiksi Hall-tahdistinta, induktiotahdistinta tai valotahdistinta. MegaSquirtin koodiin on valmiiksi ohjelmoitu useita eri anturityyppejä, jolloin käyttäjän ei tarvitse kuin valita listalta oikea tahdistin, mitä käytetään. Tahdistussignaali voidaan myös toki ottaa suoraan virranjakajan katkojankärjistä vanhoissa ajoneuvoissa. /4, s. 10./



KUVA 3. Sisääntuloportit

2.3 Ulostulosignaalit

Ulostuloporteista neljä on käytössä. Käytössä olevat ulostuloportit ovat polttoaineen ruiskutusta, sytytystä, polttoainepumpun ja joutokäyntimoottorin ohjausta varten. Kuvassa 4 on ulostuloporttien kytkentä piirilevyllä.



KUVA 4. Ulostuloportit

2.3.1 Polttoaineen ohjaus

Polttoaineen ruiskutusuuttimet on kytkettynä kahteen eri ryhmään, ja niitä ohjataan joko erikseen ryhmittäin tai molempia ryhmiä samanaikaisesti. MegaSquirt ohjaa FET-ohjainta U7, mikä puolestaan ohjaa Q2- ja Q7-transistoreita. Nämä transistorit vastaavasti ohjaavat suoraan suuttimia maadoittaen +12V jännitteen suuttimien käämien läpi, mikä saa suuttimet aukeamaan. MegaSquirt laskee oikean aukioloajan suuttimille moottorin kierrosluvun, imusarjan paineen ja muiden vaikuttavien säätöparametrien mukaan. Suuttimet ruiskuttavat tasaisella virtausnopeudella polttoainetta; suuttimien aukioloajat ovat tyypillisesti muutamia millisekunteja.

Matalan impedanssin (alle 10 Ohmia) suuttimien ohjausvirtaa tarvitsee rajoittaa tai muuten suuttimet lämpeävät liikaa ja rikkoontuvat. Virtaa voidaan rajoittaa kytkemällä etuvastukset ennen suuttimia, tai käytetään MegaSquirtiin ohjelmallisesti rakennettua pulssinleveysmoduloitua virranrajoitusta, eli PWM-virranrajoitusta. PWM katkoo virtaa hyvin nopeassa suhteessa, jolloin suuttimien läpi menevä virta pysyy pienenä.

2.3.2 Sytytyksen ohjaus

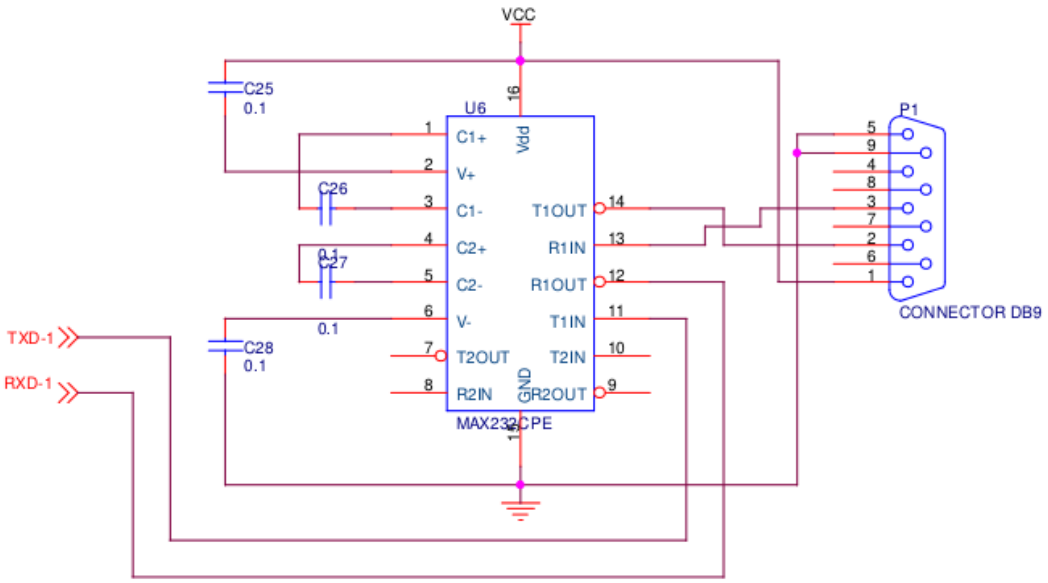
Sytytyksen ohjaus tapahtuu ohjaamalla joko suoraan sytytyspuolia tai ohjainlaitetta, mikä ohjaa sytytyspuolia. Sytytysjärjestelmiä on useita, joten myös sytytysohjauksen signaali voi vaihdella paljon eri järjestelmien kesken. Pääpiirteittäin ohjaus tapahtuu maadoittamalla sytytyspuolat, mikä saa aikaan sytytyspuolaan korkeajännitteen varautumisen. Kun ohjainlaite katkaisee maadoituksen, puolaan varautunut energia vapautuu sytytyspuolasta sytytystulppaan. Mikro-ohjaimen heikko ohjaussignaali on vahvistettava erillisellä transistorilla aina kun jännitettä tarvitaan yli 5V tai virta ylittää 20mA. Käyttäjän tarvitsee asettaa oikein sytytyspuolan vaatima varausaika, jotta MegaSquirt osaa laskea, missä kohdin se rupeaa varaamaan sytytyspuolaa ennen sytytyshetkeä. Sytytyshetki, eli siis usein sytytysennakko, lasketaan tietysti moottorinkierrosluvun, imusarjan paineen ja muiden asiaan vaikuttavien tekijöiden mukaan.

Ajoneuvoissa käytettyjä sytytysjärjestelmiä on useita, mutta MegaSquirtin avulla pystytään ohjaamaan suoraan monia niistä, kun asetukset vain säätää oikein. Sytytyspuoliakin voi olla joka sylinterille omansa, yksi puola kahta sytytystulppaa kohden (ns. hukkakipinä) tai yksi puola kaikkia sytytystulppia varten. Ohjainlaitteen tehtävänä on kuitenkin vain ohjata sytytyspuola varautumaan oikeaan aikaan ja vapauttamaan sytytysenergia sytytystulppaan sytytyshetkellä. Väärin asetetut latausajat tuhoavat helposti sytytyspuolan. Kun MS1-versiolla halutaan ohjata sytytystä, tarvitsee siihen tehdä muutoksia koodin lisäksi myös piirilevylle. Ohjainlaitteen uudemmilla versioilla sytytyksenohjauskin onnistuu suoraan.

2.4 Sarjaliikenneportti

Tietokoneella MegaSquirtin kanssa kommunikointi tapahtuu RS232-sarjaportin välityksellä. Sarjaportti on nykyisin jo hyvin vanhentunut ja hyvää vauhtia poistumassa käytöstä. Sarjaportin käytön poistumista vauhdittaa se, että se vaatii -12V jännitteen. Se onkin ainoa osa PC:ssä, joka tarvitsee negatiivisen jännitteen, ja valmistajat haluaisivatkin päästä siitä eroon, koska se tekee virtalähteistä monimutkaisempia ja kalliimpia. /5./

Mikro-ohjaimen ymmärtääkseen sarjaporttiliikennettä tarvitaan erillinen mikropiiri, joka muuntaa sarjaporttiliikenteen TTL-logiikkaa (Transistor-Transistor Logic) vastaavaksi. Tähän muunnokseen käytetään MAX232-mikropiiriä. Käytännössä siis sarjaportin jännitteet välillä +3 V ja +15 V muutetaan +5 V jännitteeksi, jonka mikro-ohjain tulkitsee loogiselta arvoltaan nollaksi, jännitteet välillä -3 V ja -15 V muutetaan 0V jännitteeksi, jonka mikro-ohjain tulkitsee loogiselta arvoltaan ykköseksi. MAX232-mikropiiriä käytetään tietysti molempiin suuntiin, mikro-ohjaimelta lukiessa ja sinne kirjoittaessa. Kuvassa 5 mikropiirin kytkentä. /6./



KUVA 5. Kytkentäkaavio MAX323 mikropiiristä

3 LANGATON TIEDONSIIRTO

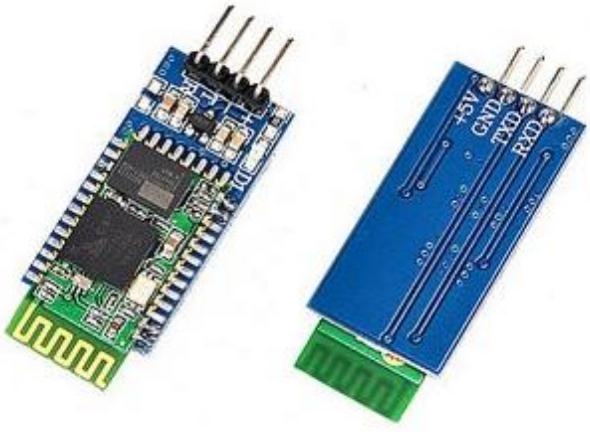
Matkapuhelimen ollessa kannettava laite kommunikointi MegaSquirtin ja puhelimen välillä täytyisi tapahtua langattomasti. Koska vaatimuksena oli langattomuus, tuntui loogiselta käyttää Bluetooth-yhteyttä kommunikoimiseen, mikä nykyisin onneksi löytyy lähes jokaisesta matkapuhelimesta. Bluetooth-tekniikka kehitettiinkin alun perin RS232-sarjaporttiliikenteen korvaajaksi.

Bluetooth on radiotekniikkaan perustuva langaton tiedonsiirtotekniikka. Bluetooth toimii 2,4000 – 2,4835 GHz taajuusalueella. Tiedonsiirtonopeudet Bluetoothilla ovat melko nopeita, esimerkiksi Bluetooth 2.0-versiolla ja EDR-laajennuksella (Enhanced Data Rate) päästään jopa 3,0 Mbps nopeuksiin. Tiedonsiirtonopeudesta ei siis tule

ongelmaa MegaSquirtin kanssa, sillä MS1-versiossa tiedonsiirtonopeutena käytetään 9600kbps nopeutta. Bluetoothin kantomatka riippuu oleellisesti lähettimen lähetystehosta. Tällä hetkellä lähetysteholuokkia on kolme. Luokan yksi laitteissa lähetysteho on 100 mW, jolla saavutetaan n. 100 m kantomatka. Luokan kaksi laitteissa lähetysteho on 2,5 mW mahdollistaen n. 10 m kantomatkan, ja viimeisessä kolmannen luokan laitteissa lähetysteho on 1 mW kantomatkan ollessa n. 5 m. Kantomatkaan vaikuttaa tietysti moni muukin tekijä, kuten radioaaltojen eteneminen väliaineessa, kotelon materiaali, antenni ja jännitelähde. /7./

3.1 MegaSquirtin muokkaaminen

Saadakseen MegaSquirt kommunikoimaan langattomasti Bluetoothin avulla matkapuhelimen kanssa piti MegaSquirtin piirilevylle tehdä muutoksia. Piirilevylle lisättiin Bluetooth-modeemi UART-sarjaliikennepiirillä (Universal Asynchronous Receiver Transmitter), mikä mahdollistaa langattoman asynkronisen sarjaliikenteen MegaSquirtin ja matkapuhelimen välillä. Bluetooth-modeemi pohjautuu CSR:n valmistamaan BlueCore4-mikropiiriin, joka käyttää Bluetooth 2.0-versiota EDR-laajennuksella. Lähetysteholuokaltaan laite kuuluu luokkaan kaksi, eli teoriassa kantomatka tällä laitteella on noin 10 metriä.



KUVA 6. Bluetooth-modeemi

Kytkentä MegaSquirtin piirilevylle oli helppo, koska vain neljä johdinta piti juottaa: virransyöttö +5V, maadoitus, lähetys ja vastaanotto. MegaSquirtista löytyy suoraan viiden voltin jännite useastakin kohtaan, sillä piirilevyltä löytyy viiden voltin regulaattori, koska mikro-ohjain toimii myös viiden voltin jännitteellä. Lähetys(TX) ja

vastaanotto(RX) juotetaan MegaSquirtin mikro-ohjaimelta löytyviin TxD- ja RxD-nastoihin, ristiin kytkettynä.

3.2 Yhteysasetukset

Bluetooth-modeemin oikean kommunikoinin varmistamiseksi MegaSquirtin mikro-ohjaimen kanssa, pitää modeemin sarjaliikenneasetukset määrittää oikeiksi. Oikeat asetukset löytyivät MegaSquirtin kokoamisohjeesta, 9600 Baudia sekunnissa, 8 databittiä, 1 stopbitti ja ei vuonohjausta. Baudi kuvaa elektronisen signaalin muutosnopeutta sekunnissa, eli toisin sanoen tietoa pystytään siirtämään enimmillään 9600 bittiä sekunnissa ohjainlaitteen kanssa. Oikeilla asetuksilla varmistetaan, että mikro-ohjainta ei turhaan rasiteta niin, että polttomoottorin ohjaus häiriintyisi. Modeemissa oli kuitenkin jo valmiiksi sattumalta oikeat asetukset valmistajan toimesta. Asetukset pystyttäisiin kuitenkin helposti muuttamaan lähettämällä modeemille sarjaportin kautta komentoja asetusten muuttamiseksi. Komennot vaihtelevat laitevalmistajan mukaan, enkä siksi tässä tarkemmin käy läpi niitä. Modeemi-piiri saattaa vaatia vielä tietyn jännitetason tiettyyn porttiin, jotta se on suostuvainen ottamaan vastaan säätämiseen liittyviä komentoja.

3.3 Tiedonsiirto teoriassa

Tutkimalla MegaSquirtin mikro-ohjaimelle kirjoitettavaa lähdekoodia sain selville, kuinka ohjainlaitteelta pystyttäisiin lukemaan tietoa sen toiminnasta. Kuten aikaisemmin todettu, mikro-ohjaimelle on ladattu MS-Extra-koodi, versioltaan uusin 029y4a. Lähdekoodin mukana toimitetaan **msns-extra.ini**-tiedosto, joka on tarkoitus avata MegaTune-säätöohjelmalla. Tiedosto kertoo MegaTune-ohjelmalle, mistä ja miten se löytää haluamansa tiedon mikro-ohjaimen muistista. Tiedosto toimitetaan aina mikro-ohjaimen lähdekoodin yhteydessä, koska jokaisessa eri MegaSquirt-versiossa säätöparametrit poikkeavat toisistaan tai ainakin niiden sijainti mikro-ohjaimen muistissa.

Tutkimalla tiedostoa tarkemmin selviää, että *versionInfo = "T"* tarkoittaa, että lähettämällä mikro-ohjaimelle merkin ”T” pitäisi sen vastata lähettämällä takaisin ladatun koodin versionumeron. Mikro-ohjaimelle ladatun koodin versiota tärkeämpi

tieto on tietää, mitä **msns-extra.ini**-tiedoston versiota on käytettävä yhdessä mikro-ohjaimelle ladatun koodin kanssa, eli ns. signature. Signature voi poiketa mikro-ohjaimen koodin versiosta, koska koodiin ei ole välttämättä tehty sellaisia muutoksia, jotka vaikuttaisivat tietokoneen ja MegaSquirtin väliseen kommunikointiin. Komennolla *queryCommand = "S"* saadaan tietää signaturen version, minkä pitäisi vastata **msns-extra.ini**-tiedostosta löytyvää *signature = "MS1/Extra format 029y3 *********"*. Tällä saadaan varmuus siitä, että pystytään käyttämään juuri tällä kyseisellä **msns-extra.ini**-asetustiedostolla olevaa koodia mikro-ohjaimelta saatujen tietojen käsittelyyn.

Tärkein komento sovelluksen kannalta on tietysti se, miten ohjainlaitteelta saadaan luettua anturitedot. Anturitietojen lukeminen on määritetty seuraavasti: *ochGetCommand = "R"*. Havaintoa tukemaan löysin MS-Extra koodin kotisivuilta http://www.msextra.com/doc/ms1extra/COM_RS232.htm täydellisen listan käytetyistä komennoista selitteineen. Kuvassa 7 on kuvakaappaus listatuista komennoista. Lähettämällä merkin ”R” saadaan siis reaaliaikaiset muuttujat ohjainlaitteelta 39:n tavun taulukkona.

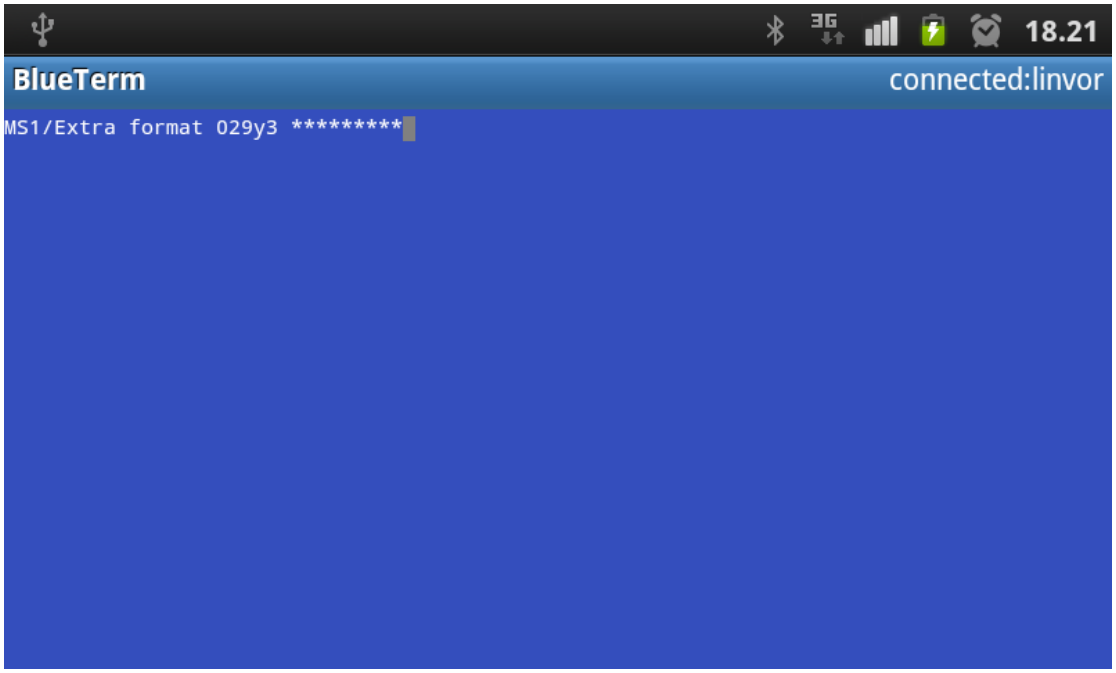
"**A**" = MS1 sends the real time variables as an array of 22 bytes
"**B**" = MS1 jump to flash burner routine and burn VE/constant values in RAM into flash
"**C**" = Test communications - echo back SECL (timer, 0 to 255) as a byte
"**P**"+<page> = MS1 loads page of data from Flash to RAM
"**R**" = MS1 sends the real time variables as an array of 39 bytes
"**S**" = Signature - MS1 sends an identifier string (this is not the version)
"**T**" = Revision - MS1 sends the code version as a string
"**V**" = MS1 sends the VE table and/or constants as an array of bytes
"**W**"+<offset>+<newbyte> = MS1 receives new table or constant byte value and stores in 'offset' location
"**X**"+<offset>+<count>+<newbyte>+<newbyte>.... MS1 receives series of new data bytes

KUVA 7. MegaSquirt MS1 -version kanssa käytetyt komennot selityksineen

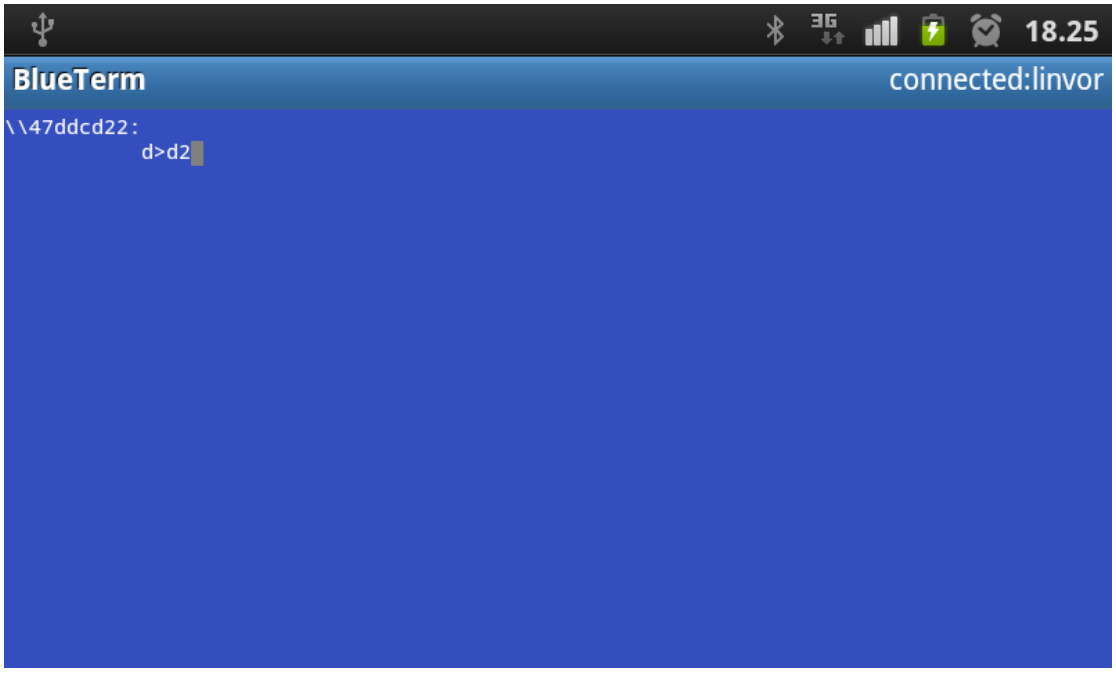
3.4 Yhteyden testaaminen

Komentojen ja Bluetooth-yhteyden testaamiseksi asensin matkapuhelimeen Android-marketista löytyvän BlueTerm-sovelluksen. BlueTerm on pääteohjelma, joka emuloi sarjaliikennettä Bluetooth-yhteydellä. Kuvassa 8 näkyy BlueTerm käytössä, Mega-Squirtin vastaus lähetettyyn signature-kyselyyn. Näytölle ilmestyneen vastauksen myötä pystyttiin päättelemään Bluetooth-modeemin olevan oikein kytketty ja

toimivan oletetulla tavalla. Ensimmäisellä kerralla en saanut MegaSquirtilta minkäänlaista vastausta, sillä olin epähuomiossa juottanut Bluetooth-modeemin RX- ja TX-pinnit suoraan mikro-ohjaimelta löytyviin vastaaviin portteihin, enkä ristiin (RX-TX ja TX-RX), kuten pitäisi. Kuvassa 9 on MegaSquirtin ja puhelimen kommunikoinnin testausta, merkki ”R” on lähetetty ohjainlaitteelle, ja vastauksena näkyy ohjainlaitteen muistista saadut muuttujat.



KUVA 8. BlueTerm-sovelluksella yhteyden testaus



KUVA 9. BlueTerm-sovelluksella muuttujien kysely ohjainlaitteelta

4 SOVELLUS MATKAPUHELIMELLE

MegaSquirtin toiminnan tarkemman tutustumisen lisäksi työssä haluttiin hyödyntää näitä tietoja ja kehittää sovellus, joka pystyisi lukemaan tietoja ohjainlaitteelta. Idea sovelluksen kehittämiseen tuli tarpeesta, missä pitäisi tarkistaa nopeasti, toimiiko moottori oikein. Kannettavaa tietokonetta huomattavasti kätevämpi olisi tarkistaa moottorin toiminta matkapuhelimella. Päätin siis kehittää matkapuhelinta varten oman sovelluksen. Kannettavan tietokoneen jatkuvasti autossa mukana kuljettaminen on myös hieman epäkäytännöllistä.

4.1 Android-sovelluskehitys

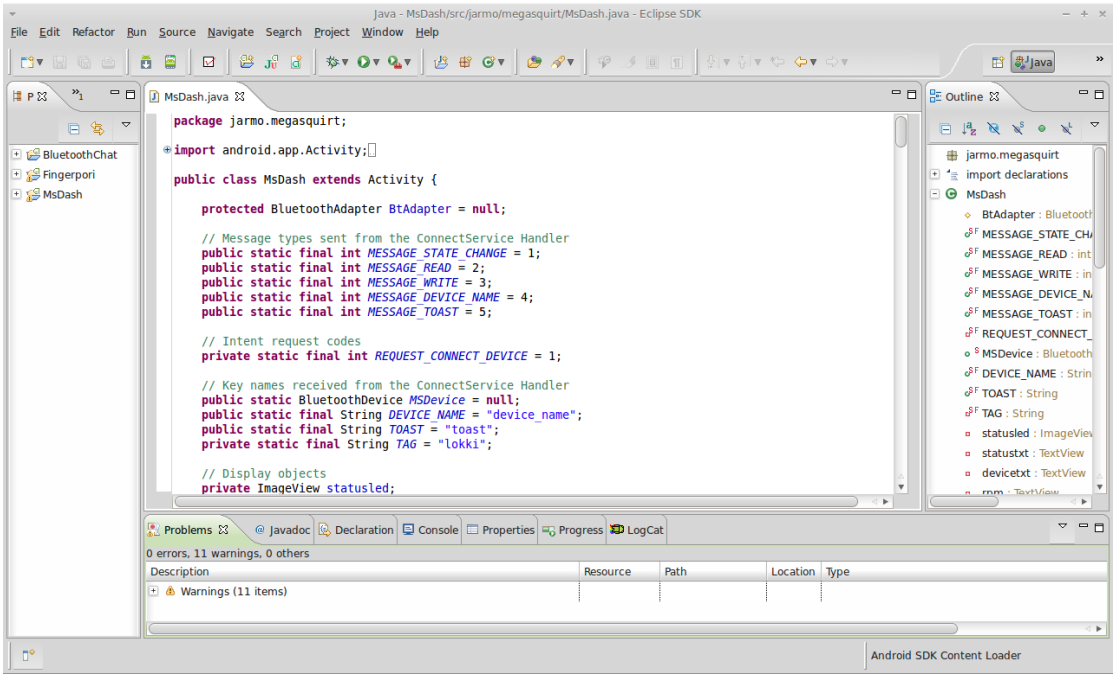
Sovellus päätettiin kehittää Android-alustalle, koska sen on arvioitu olevan käytetyin älypuhelinten käyttöjärjestelmä 52,5 % markkinaosuudellaan vuoden 2011 kolmannella neljänneksellä. Valintaan vaikutti osaltaan myös se, että Androidille ohjelmistokehitys tapahtuu Java-kielellä. Minulla oli jo hieman kokemusta Java-ohjelmoinnista ennakkoon. /8./

Android on avoimen lähdekoodin ohjelmistopino, joka sisältää käyttöjärjestelmän, väliohjelmistoja ja käyttäjän perusohjelmia. Android pohjautuu Linux-käyttöjärjestelmäytimeen. Ohjelmistokehittäjän kannalta lähdekoodin avoimuus on todella hyvä asia, sillä tutkimalla lähdekoodia pystyy oppimaan, miten kaikki toimii, ja tätä tietoa voi hyödyntää ohjelmistokehityksessä.

Android-sovelluskehitys aloitetaan asentamalla Androidin Software Development Kit (SDK). SDK sisältää tarvittavat työkalut sovelluskehittämiseen. SDK:n voi ladata Androidin viralliselta kehittäjille suunnatulta sivulta http://developer.android.com/sdk/index.html. Koska ohjelmointikielenä käytetään Javaa, tarvitaan tietysti myöskin Javasta kehittäjille suunnattu versio, eli JDK (Java Development Kit). JDK sisältää työkalut, joilla lähdekoodi saadaan käännettyä suoritettavaan muotoon.

Sovelluskehityksen helpottamiseksi on vielä hyvä asentaa IDE (Integrated Development Environment) eli integroitu ohjelmointiympäristö.

Ohjelmointiympäristö koostuu useista työkaluista, jotka on tehty avustamaan ohjelmointityössä. Ohjelmointiympäristö ei ole tietenkään pakollinen, koska ohjelmointi onnistuu kyllä käyttämällä pelkkää tekstieditoria. Ohjelmointiympäristö koostuu käyttöliittymästä ja koodin kääntäjästä. Käyttöliittymään liittyy usein toimintoja, jotka helpottavat ohjelmointityötä, kuten syntaksikorostus, automaattitäydennys, automaattisia muotoiluja ja ohjelman käyttöliittymän suunnittelutyökaluja. Koodin kääntäjä avustaa myös käyttäjää esimerkiksi virheiden paikantamisessa ja antamalla korjausehdotuksia. Kuvassa 10 on kuvakaappaus projektissa käytetystä IBM:n kehittämästä *Eclipse*-ohjelmointi-ympäristöstä. /9, s. 21-23./



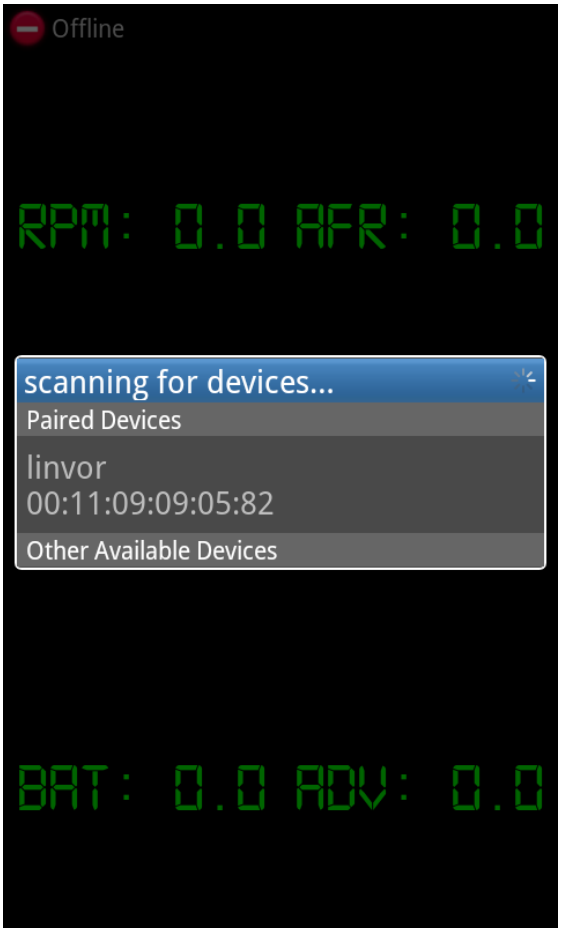
KUVA 10. *Eclipse*-ohjelmointiympäristö

Sovelluksen testaus on mahdollista suoraan puhelimella, joka on liitetty USB-kaapelilla tietokoneeseen. Android-matkapuhelimen asetuksista pitää vain sallia sovellusten asentaminen tuntemattomista lähteistä (**Settings** > **Applications** > **Unknown Sources**), ja USB-virheenkorjaus (**Settings** > **Applications** > **Development** > **USB debugging**) pitää kytkeä päälle. Toinen vaihtoehto on käyttää ns. emulaattoria tietokoneella, jolloin edes puhelinta ei tarvita sovelluskehitykseen. Android SDK:n mukana tulee kaikki tarvittava emulaattoria varten. Tässä projektissa

käytettiin kuitenkin matkapuhelinta testaamiseen, koska Bluetooth-yhteyttä on mahdotonta emuloida tietokoneella.

4.2 Yhteyden muodostus

Ensimmäisenä oli tietysti saatava sovellus hallitsemaan itsenäisesti Bluetooth-yhteys ohjainlaitteen kanssa. Käytin tässä hyväksi Android Developers-sivulta löytyvää esimerkkiprojektia BluetoothChat (http://developer.android.com/resources/samples/BluetoothChat/index.html). Sain hyödynnettyä suoraan tästä esimerkkiprojektista paljon koodia sovellukseeni, kuten uusien Bluetooth-laitteiden etsiminen ja yhteyden muodostus.



KUVA 11. Bluetooth-laitteiden etsiminen ja yhteyden muodostaminen

Yhteyden muodostamisen jälkeen sovellus varmistaa ohjainlaitteen version signature-kyselyllä. Näin varmistetaan, että yhdistetty Bluetooth-laite on MegaSquirt ja versioltaan MS1V2.2. Tällä hetkellä sovellus tehtiin kommunikoimaan vain tämän

kyseisen version kanssa, mutta jatkossa tuki muille versioille on mahdollista lisätä. Eri versioissa eri tiedot sijaitsevat hieman eri muistipaikoissa, siksi version varmistaminen oikeaksi on tärkeää.

4.3 Tiedon kysely ja käsittely

MegaSquirtilta saadaan tieto moottorin toiminnasta 39 tavun taulukkona. Tieto ohjainlaitteelta lähetetään taulukkona, jotta siirrettävän datan määrä pysyisi mahdollisimman pienenä, eikä se näin kuormita liikaa MegaSquirtin mikro-ohjainta ja pystyisi keskittymään tärkeimpään eli moottorin ohjaamiseen. Tietoa kysellään ohjainlaitteelta 50 millisekunnin välein, kuten myös MegaTune-ohjelma tekee.

Taulukko sisältää siis 39 tavua, ja tavun koko on kahdeksan bittiä. Jokainen bitti voi saada joko arvon nolla tai yksi, joten yhteen tavuun voidaan siis tallentaa kokonaisluku väliltä 0-255 (2^8). Ensimmäiseksi pitääkin taulukko jakaa 39 eri muuttujaan. MS-Extra-kotisivuja tutkimalla löysin taulukon, jossa kerrotaan, mitä muuttujaa mikäkin tavu vastaa. Taulukossa 2 on kaikki muuttujat selitteineen suomennettuna.

TAULUKKO 2. MegaSquirtilta luetut 39 muuttujaa

Muuttujan nimi	Sijainti taulukossa	Yksikkö	Selite
secl	0	sekunti	Kellopulssi laskuri, mikä laskee arvoja välillä 0 - 255
squirt	1	bitti	**Bitti Selite, kun ”korkea”-tilassa** 0 inj1 ruiskutetaan 1 inj2 ruiskutetaan 2 aiotaan ruiskuttaa suutinryhmä 1 3 ruiskutetaan suutinryhmää 1 4 aiotaan ruiskuttaa suutinryhmä 2 5 ruiskutetaan suutinryhmää 2 6 ahtopainesäätö pois käytöstä

engine	2	bitti	**Moottorin tila** **Bitti Selite kun ”korkea”-tilassa** 0 käynnissä 1 käynnistetään 2 käynnistyksen jälkeinen rikastus 3 moottorin lämmityskäyttö 4 kaasuläpän asentoon perustuva kiihdytystila 5 moottorijarrutus 6 imusarjan painetietoon perustuva kiihdytystila 7 joutokäynnillä
baroADC	3	ADC	Ympäristössä vallitseva ilmanpaine
mapADC	4	ADC	Imusarjan paine
matADC	5	ADC	Imusarjan lämpötila
cltADC	6	ADC	Jäähdytinnesteen lämpötila
tpsADC	7	ADC	Kaasuläpän asento
batADC	8	ADC	Akun jännite
egoADC	9	ADC	Pakokaasujen jäännöshapen arvo (=Lambda)
egoCorrection	10	%	Lambdasäädön korjausvaikutus (Closed loop tilassa)
airCorrection	11	%	Ilmantiheyden korjausvaikutus
warmupEnrich	12	%	Kylmän moottorin rikastus
rpm100	13	r100	Kierrosta minuutissa jaettuna 100:lla
pulsewidth1	14	ms	Ensimmäisen suutinryhmän pulssin leveys
accelEnrich	15	ms	Kiihdytysrikastuksen pulssisuhteen lisäys
baroCorrection	16	%	Ympäristön ilmanpaineen aiheuttama korjaus ruiskutusaikaan
gammaEnrich	17	%	Kaikkien lisärikastusten yhteisvaikutus seostäytökseen
veCurr1	18	%	Käytössä oleva sylinterin täytössuhteen arvo (Polttoainekartta 1)

pulsewidth2	19	ms	Toisen suutinryhmän pulssin leveys
veCurr2	20	%	Käytössä oleva sylinterin täytössuhteen arvo (Polttoainekartta 2)
idleDC	21	%	Joutokäyntiventtiilin ohjauksen arvo
iTime*	22	s	Väliaika, esim aika miten usein moottorin pyörintänopeus tieto saadaan. Käytetään tarkemman kierroslukutiedon laskentaan.
iTime*	23	s	Väliaika, esim aika miten usein moottorin pyörintänopeus tieto saadaan. Käytetään tarkemman kierroslukutiedon laskentaan.
advance	24	aste	Sytytysennakko
afrTarget	25	ADC	Polttoaineilmaseos arvo mitä MegaSquirt yrittää saavuttaa
fuelADC	26	ADC	piirilevyn X7-liitäntä (esim. toinen happianturi, polttoaineen paineanturi tai ajoneuvon nopeusanturi)
egtADC	27	ADC	piirilevyn X6-liitäntä (esim. pakokaasujen lämpötila-anturi)
CltIatAngle	28	aste	Imuilman lämpötilan ja jäähdytinnesteen lämpötilan vaikutus sytytysennakkoon
KnockAngle	29	aste	Sytytysennakon määrä mitä on vähennetty moottorin nakutuksen ansiosta
egoCorrection2	30	%	Sama kuin egocorrection, mutta toiselle lambda-anturille (jos käytössä)
porta	31	bitti	**Bitti Selite** 0 Polttoainepumppu päällä 1 Joutokäyntiventtiili päällä/tai sytytyksen ohjaus toiminnassa 2 Ulostuloportti 2 (X5) 3 Ulostuloportti 1/Ahtopainesäätö (X4) 4 Ilokaasujärjestelmä/vesiruiskutus (X3)

			5 Tuuletin/vesiruiskutus (X2) 6 Matalaohmisten suutinten ohjaukseen (flyback) 7 Matalaohmisten suutinten ohjaukseen (flyback)
portb	32	bitti	ADC sisääntulot **Bitti Selite** 0 Imusarjanpaine 1 Imuilman lämpötila 2 Jäähdytinnesteen lämpötila 3 Kaasuläpän asento 4 Akun jännite 5 Lambda-anturi 6 X7-liitäntä piirilevyllä 7 X6-liitäntä piirilevyllä
portc	33	bitti	**Bitti Selite** 0 Ruiskutus LED tai sytytyspuola a 1 Kiihdytys LED tai sytytyspuola b tai HEI7 2 Tuuletin/Ulostuloportti 4/sytytyspuola c 3 Vaihtorajoitin tai sytytyspuola e 4 Vaihtovalo tai toinen tahdistin-anturi
portd	34	bitti	**Bitti Selite** 0 käyttämätön tai sytytyspuola d 1 Ilokaasujärjestelmä/Polttoainekartan vaihto 2 Ei nakutusta (Bitti ei-tilassa nakutusta) 3 Lähtörajoitin pois käytöstä 4 Suutin1 5 Suutin2 tai X2 liitäntä tuulettimen ohjaukseen
stackL	35	Tavu	CPU stack
tpsLast	36		Kaasunasento/Imusarjan paine, nykyistä edeltänyt arvo

iTimeX	37	s	Väliaika, esim aika miten usein moottorin pyörintänopeus tieto saadaan. Käytetään tarkemman kierroslukutiedon laskentaan.
bcdc	38	%	Ahtopaineensäätö

Kuten taulukosta 2 nähdään, MegaSquirt lähettää osan muuttujien arvoista suoraan käyttökelpoisina yksikköinä, kuten sekunteina, bitteinä tai prosentteina. Kuitenkin osa arvoista saadaan ohjainlaitteelta ADC-muodossa (*Analog to Digital Converter*), eli MegaSquirtin mikro-ohjain on muuttanut anturilta saadun analogisen jännitteen kahdeksanbittiseksi muuttujaksi, eli kokonaisluvuksi väliltä 0-255. Muuttujien arvot, jotka ohjainlaite lähettää ADC-arvoina, liittyvät sellaisiin antureihin, jotka voivat vaihdella eri käyttösovelluksissa. Esimerkiksi lämpötila-antureita, lambda-antureita ja imusarjan paineantureita voi olla useita erilaisia käyttötarkoituksen mukaan. Niinpä näiden antureiden tieto käsitellään ADC-arvoina, mikä ei sido käyttämään juuri tietynlaisia antureita.

ADC-arvoista sovelluksen täytyy laskea käyttäjälle selkokieliset arvot, eli esimerkiksi cltADC-muuttujasta lasketaan jäähdytinnesteen lämpötila celsius- tai fahrenheit-yksikköinä. Laskentaan käytetään erillisiä ”.inc” -tiedostoja, jotka tulevat MegaSquirtin mikro-ohjaimen lähdekoodin mukana. Näissä asetustiedostoissa on ADC-arvo väliltä 0 – 255, ja sitä vastaava arvo esimerkiksi lämpötila-anturia varten ADC-arvoa vastaava arvo celsius-asteina. Mikäli käytetään General Motorsin lämpötila-antureita, oletus asetustiedostoja ei tarvitse muokata, muussa tapauksessa ne pitää tehdä uusiksi. Lämpötila-antureita varten on olemassa EasyTherm-niminen ohjelma Windows-käyttöjärjestelmälle, minkä avulla voi generoida tiettyä anturityyppiä varten oikean oman asetustiedoston. Ohjelmaan syötetään muutama eri vastusarvo eri lämpötiloissa, ja ohjelma laskee siitä kaikki loputkin arvot ja tekee niistä asetustiedoston.

Android-sovellukseen tuli tehdä koodi joka avaa tämän ”.inc”-tiedoston ja poimii sieltä ADC arvoa vastaavan arvon. Koska erilaisia antureita on useita, tuli käyttäjälle mahdollistaa myös se, että erilaisia antureita varten käyttäjä pystyy käyttämään omaa

asetustiedostoaan. Niinpä sovellus yrittää ensisijaisesti avata matkapuhelimen ulkoiselta muistikortilta asetustiedostoa, minkä mukaan ohjelma skaalaa anturin tiedot. Jos muistikortilta ei löydy asetustiedostoja, sovellus käyttää tietojen skaalaamiseen sovelluksen mukaan sisällytettyjä oletus asetustiedostoja, jotka ovat samat kuin MS-Extra-lähdekoodin mukana toimitettavat asetustiedostot.

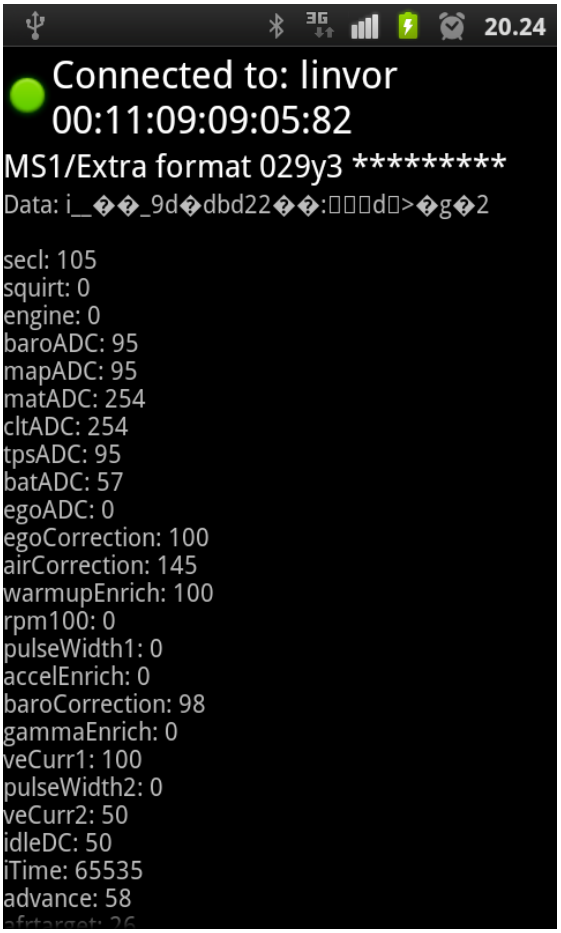
Kun tiedot on saatu talletettua 39 eri muuttujaan, tehdään niille sen jälkeen vielä pieniä tarvittavia laskentatoimia. Kaikkien muuttujien arvojen laskemiseen käytettävät laskukaavat on suoraan lainattu **msns-extra.ini** -tiedostosta, mitä myös MegaTune-ohjelma käyttää asetustiedostonaan. Löysin vastaavasta projektista suoraan java-koodiksi käännettynä nämä **msns-extra.ini** tiedostossa olevat laskukaavat mitä hyödynsin omassa projektissani. Lähdekoodi tälle Megasquirt.java:lle löytyi osoitteesta https://bitbucket.org/scudderfish/mslogger.

4.4 Tiedon esittäminen kuljettajalle

Kun sovellus on saatu luotettavasti hallitsemaan Bluetooth-yhteys, ohjainlaitteelta on saatu luettua tarvittavat tiedot ja käsiteltyä ne, on aika kehittää käyttöliittymä. Loppukäyttäjälle sovelluksen tärkein ja näkyvin osa on tietysti käyttöliittymä. Käyttöliittymää suunnitellessa on hyvä pitää mielessä, että sovellusta tullaan käyttämään liikkuvassa autossa, jolloin siitä täytyy tehdä mahdollisimman yksinkertainen ja helposti käytettävä.

Tavoitteena oli tehdä käyttöliittymästä sellainen, että sillä voitaisiin tarvittaessa korvata kokonaan auton vanha mittaristo. Toimivin ratkaisu olisi näyttää tiedot samanlaisilla viisarimittareilla, kuten autoissa on totuttu näkemään. Mittareiden pitäisi olla selkeästi luettavissa, käyttäjän täysin muokattavissa, kuten mitä tietoja näytetään, mittareiden skaalaus, mahdolliset raja-arvot varoituksille jne. Käyttöliittymän suunnittelu osoittautui kuitenkin haasteellisemmaksi, mitä odotinkaan. Tein käyttöliittymästä useita eri kokeiluja, mutta en saanut siitä kuitenkaan niin hyvää kuin olin suunnitellut. Kuvassa 12 on yksi käyttöliittymäkokeiluista.

Viisarimittareita varten löysin valmiin koodin, jota olisin voinut käyttää tiedon näyttämiseen. Useista yrityksistä huolimatta en saanut mittareiden toimintaa tyydyttävälle tasolle. Ongelmat liittyivät mittareiden skaalaamiseen näytön koon mukaan, mittariasteikoiden skaalaamiseen ja mittareiden sovitteluun näytölle. Suurin ongelma, mikä mittareihin liittyi, oli se, että jokaiselle mittarille pitää erikseen määrittää omat asetukset esim. tiedon skaalaamista varten. Kun jokaiselle mittarille on omat asetukset, sitä tiettyä mittaria ei voida käyttää kuin sen tietyn tiedon näyttämiseen, mistä taas aiheutui se, että käyttäjä ei saanut täysin vapaasti määrittää, mitä tietoa näytetään ja missä mittari sijaitsee näytöllä. Koodi viisarimittarigrafiikkaa varten löytyi osoitteesta: http://code.google.com/p/widgets4androidhmi/.



KUVA 12. Käyttöliittymän testausta

Koska en saanut viisarimittareita tyydyttävästi toimimaan, päädyin näyttämään tiedot yksinkertaisesti numeroina. Idea käyttöliittymän ulkoasuun saatiin jo olemassa olevista mittaristoista 80-luvun urheiluautoista, joissa on käytetty LED-segmenttinäyttöjä mittaristoina. Esimerkiksi Opel Kadett GSI:ssä ja Toyota Suprassa

on käytetty vastaavan näköisiä mittaristoja. Käyttöliittymästä tuli yllättävän selkolukuinen jopa autossa, kun numerot ovat kirkkaan vihreällä mustaa taustaa vasten. Kuvassa 13 on kuvakaappaus valmiista käyttöliittymästä.

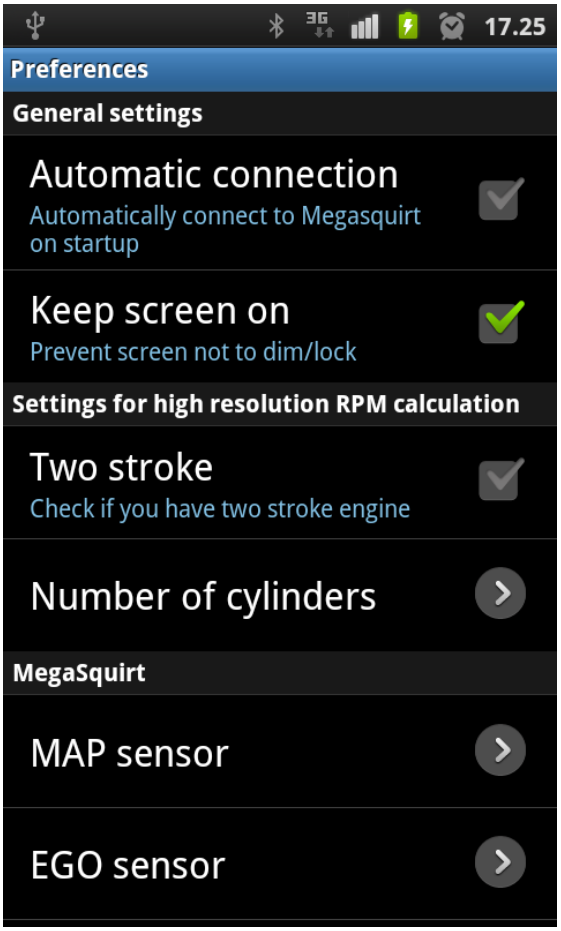
KUVA 13. Valmis käyttöliittymä

Valmiiseen käyttöliittymään jäi kuitenkin useita puutteita, mitkä pitäisi sovellusta jatkokehittäessä ratkaista toisin. Pahimmat puutteet liittyvät muokattavuuden puuttumiseen. Tällä hetkellä käyttäjä ei itse pysty määrittämään, mitä tietoja haluaa nähdä ja miten. Valitsin näytettäviksi tiedoiksi itselleni tärkeät tiedot; kierroslukutieto (kierrosta minuutissa), polttoaine-ilma-seos (1:1), imusarjan painetieto (bar), jäähdytinnesteen lämpötilatieto (°C tai °F), akun jännite (V) ja sytytysennakko (°). Moottoreita ja käyttäjiä on erilaisia, joten käyttöliittymän pitäisi olla täysin käyttäjän muokattavissa, koska toiselle tärkeä tieto voi olla toiselle lähes hyödytöntä tietoa. Kuitenkin omaa käyttöäni varten tämä palvelee nykyisellään todella hyvin.

4.5 Käyttäjän valinnat

Käyttäjäkokemuksen parantamiseksi sovellukseen täytyi tehdä valikko, josta käyttäjä pystyy muuttamaan sovellukseen liittyviä asetuksia tarpeidensa mukaan. Käyttäjän tekemät asetusvalinnat tallentuvat puhelimen muistiin seuraavia käyttökertoja varten. Sovelluksen käynnistyessä asetukset vastaavasti luetaan puhelimen muistista.

Ennen kuin käyttäjä yhdistää ohjainlaitteeseen, täytyy käyttäjän valita muutama asetus MegaSquirtiin liittyen. Käyttäjän tarvitsee valita ainakin ohjainlaitteella käytetyn imusarjan paineanturin tyyppi, lambda-anturin malli, moottorin sylinteriluku ja se, onko moottori kaksi- vai nelitahtinen. Nämä valinnat sovelluksen täytyy tietää laskeakseen oikein näytettävät arvot. Moottorin sylinterilukua ja tahtisuustietoa käytetään laskemaan tarkempi kierroslukutieto, sillä MS1-versiossa kierroslukutieto saadaan ohjainlaitteelta vain sadan kierroksen tarkkuudella. Kuvassa 14 esitellään valikko asetusten muuttamiseen. Käyttäjälle tarjotaan myös mahdollisuus automaattiseen yhdistämiseen ohjainlaitteeseen ja mahdollisuus valita, pidetäänkö puhelimen näyttö jatkuvasti päällä.



KUVA 14. Käyttäjän asetusvalinnat

5 POHDINTA

Tämän insinöörityön tarkoituksena oli tutkia MegaSquirt-polttomoottorinohjausjärjestelmän toimintaa sekä kehittää Android-sovellus, joka pystyy kommunikoimaan MegaSquirtin kanssa. Idea työnaiheelle tuli omasta kiinnostuksesta ohjelmoitavia moottorinohjainlaitteita kohtaan ja omasta ja muiden kanssa harrastajien tarpeesta sovellukselle, joka osaa näyttää tietoa moottorin toiminnasta reaaliajassa.

Tutustumalla MegaSquirtin toimintaan, niin piirilevyn rakenteeseen kuin ohjelmistoon, auttoi oppimaan paljon uutta moottorin ohjainlaitteiden toiminnasta. Nykyaikaisilla ohjainlaitteilla pystytään helposti toteuttamaan useita asioita yhdellä piirilevyllä, kuten esimerkiksi polttoaineen annostelu ja sytytyksen ohjaus.

Oman Android-sovelluksen kehittämisessä onnistuin mielestäni hyvin, kun sain sovelluksen toimimaan luotettavasti. Sovellusta tehdessä opin valtavasti uutta tietoa muun muassa ohjelmoinnista ja digitaalitekniikasta. Työ oli todella mielenkiintoinen, kun siinä yhdistyi ajoneuvotekniikka, elektroniikka ja tietotekniikka. Tulevaisuudessa ajoneuvoihin tulee tietysti vielä entistä enemmän tietotekniikkaa ja sen osaaminen on välttämätöntä alalla työskentelevälle.

Työhön sekä sovellukseen jäi vielä jonkin verran jatkokehitettävää. Työssä olisi voinut käydä tarkemmin läpi myös useita eri MegaSquirtin versioita. Myöskin MegaSquirtin mikro-ohjaimelle ladattua lähdekoodia olisi voinut tutkia ja esitellä tarkemmin työssä. Sovelluksen sain mielestäni toimimaan hyvin, ja sitä on nyt helppo lähteä jatkokehittämään tarvittaessa. Sovelluksen voisi tehdä tukemaan myös useita eri MegaSquirt-versioita. Myös käyttäjälle pitäisi antaa enemmän valinnanvaraa muun muassa siihen, mitä tietoa näytetään ja kuinka. Sovelluksen lähdekoodi on tässä työssä liitteenä, joten jokainen asiasta kiinnostunut voi haluttaessaan tutustua siihen ja kehittää sovellusta tästä eteenpäin.

LÄHTEET

1 Introduction to MegaSquirt. Bruce Bowling, Al Grippo, Lance Gardiner. http://www.megamanual.com/v22manual/mintro.htm Päivitetty 27.7.2011. Luettu 10.2.2012.

2 Wikipedia. MegaSquirt. WWW-dokumentti. http://en.wikipedia.org/wiki/MegaSquirt Päivitetty 27.1.2012. Luettu 10.2.2012

3 Robert Bosch GmbH, Bensiinimoottorin ohjaus Motronic-järjestelmät, Helsinki: Autoalan Koulutuskeskus Oy. 2006.

4 Vahtera Pentti, Mikro-ohjaimen ohjelmointi C-Kielellä, Halikko: Microsalo Oy. 2008

5 Wikipedia. RS-232. WWW-dokumentti. http://fi.wikipedia.org/wiki/Rs232 Päivitetty 26.2.2012. Luettu 30.3.2012

6 Wikipedia. MAX232. WWW-dokumentti. http://en.wikipedia.org/wiki/MAX232 Päivitetty 18.4.2012. Luettu 20.4.2012

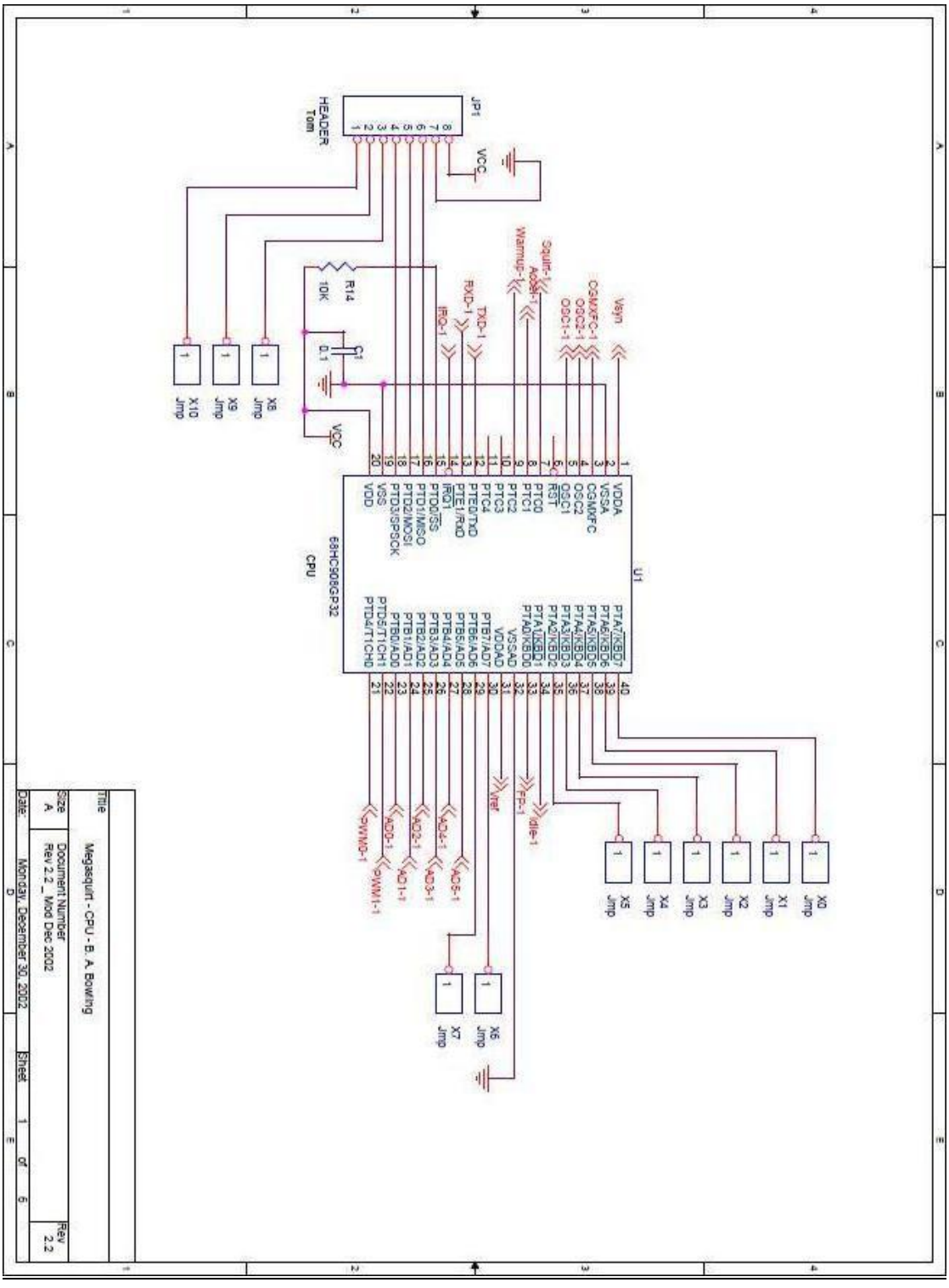
7 Wikipedia. Bluetooth. WWW-dokumentti. http://en.wikipedia.org/wiki/Bluetooth Päivitetty 22.4.2012. Luettu 22.4.2012

8 Wikipedia. Android. WWW-dokumentti. http://fi.wikipedia.org/wiki/Android Päivitetty 18.4.2012. Luettu 22.4.2012

9 Gargenta Marko, Learning Android, O'Reilly Media. 2011.

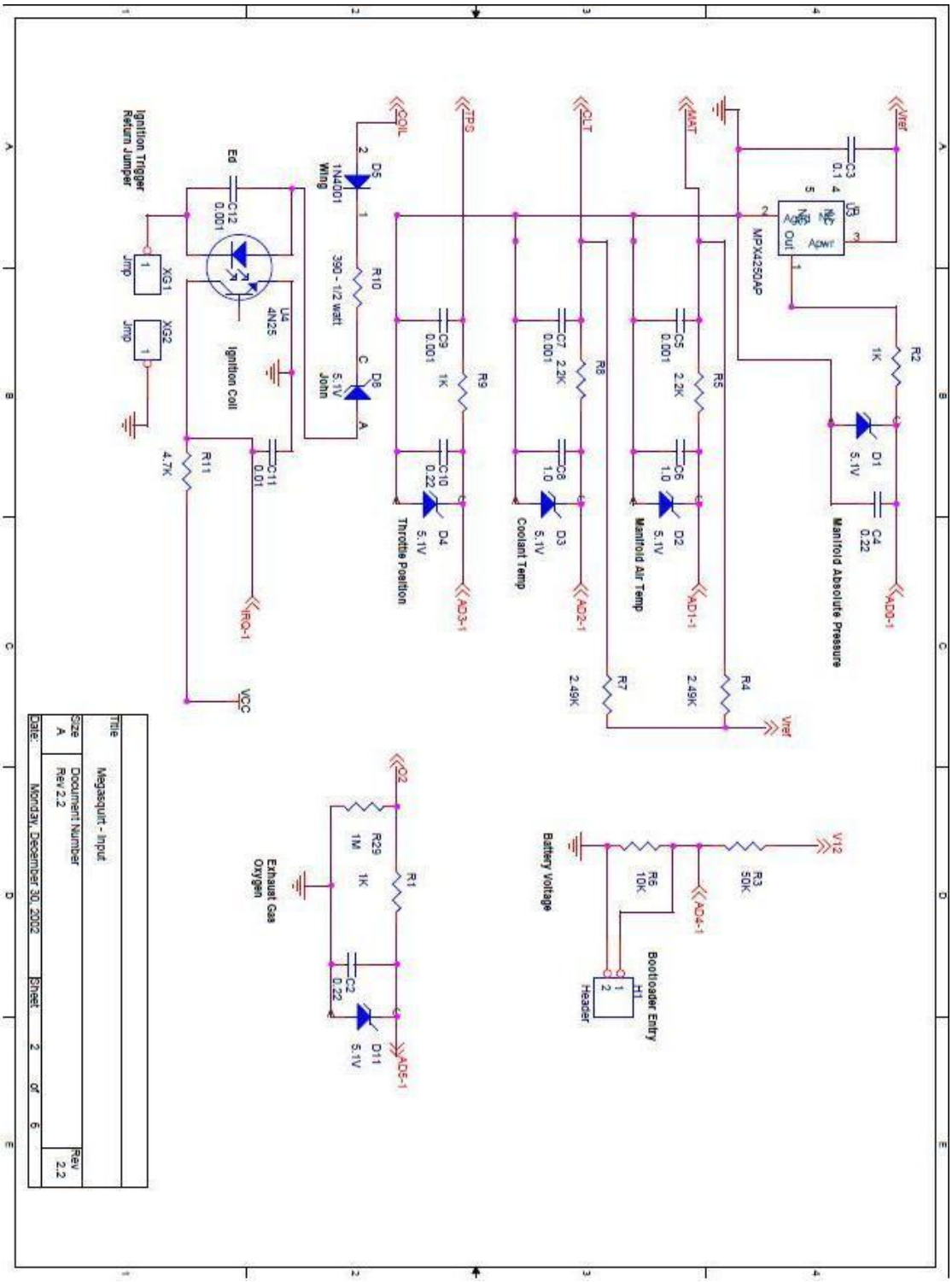
LIITE 1(1).

MegaSquirt V2.2 -kytkentäkaavio



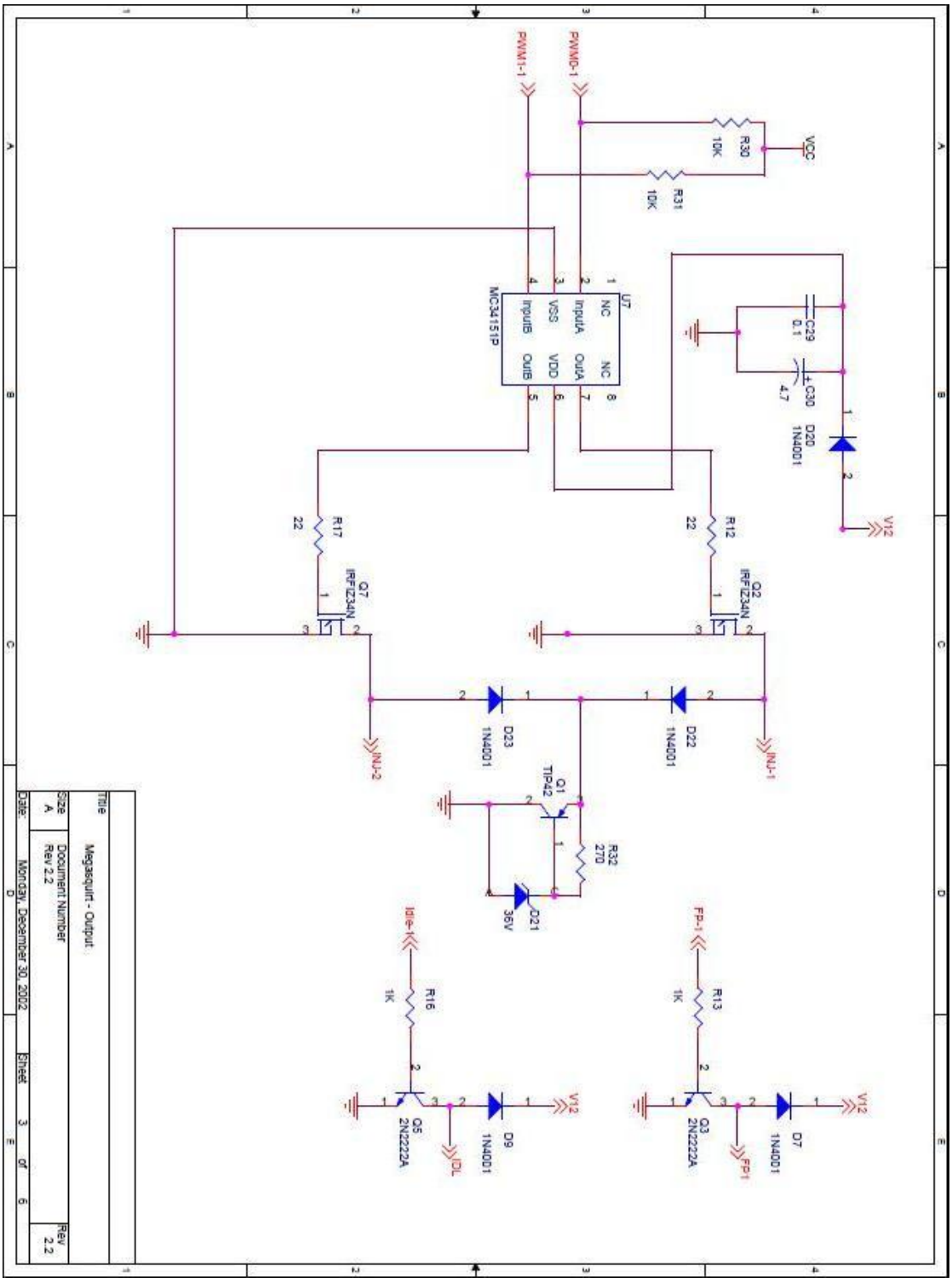
LIITE 1(2).

MegaSquirt V2.2 -kytkentäkaavio



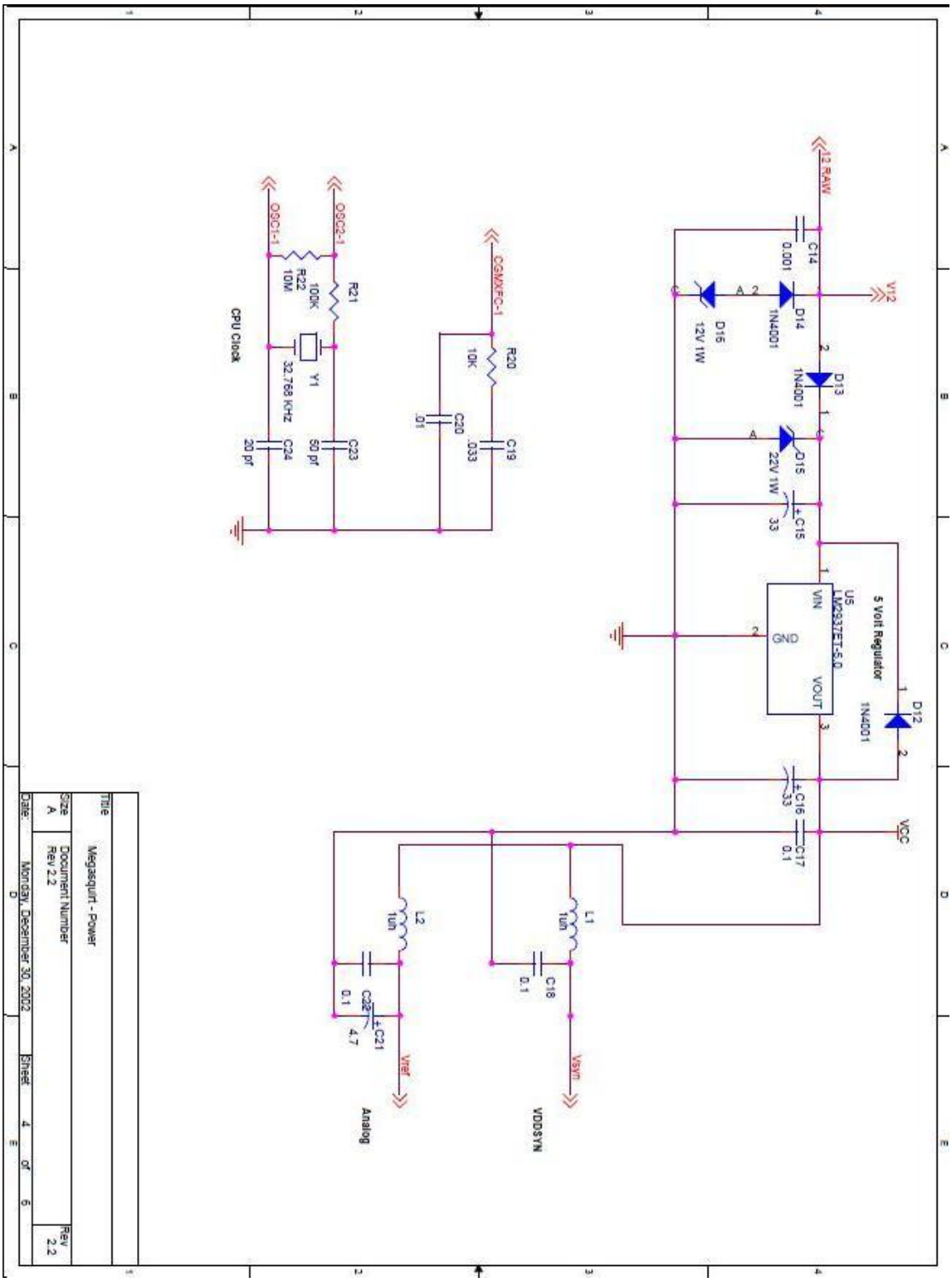
LIITE 1(3).

MegaSquirt V2.2 -kytkentäkaavio



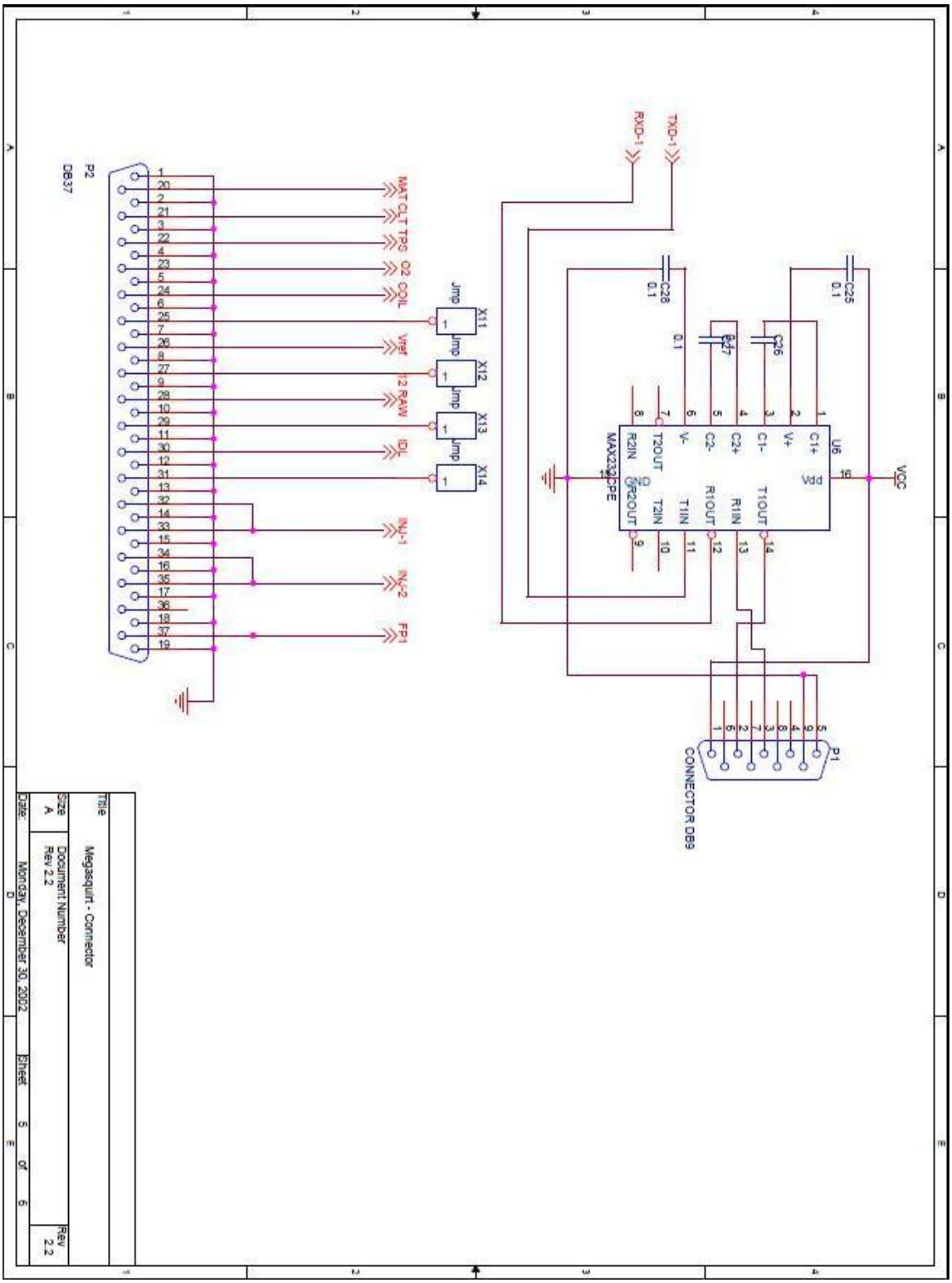
LIITE 1(4).

MegaSquirt V2.2 -kytkentäkaavio



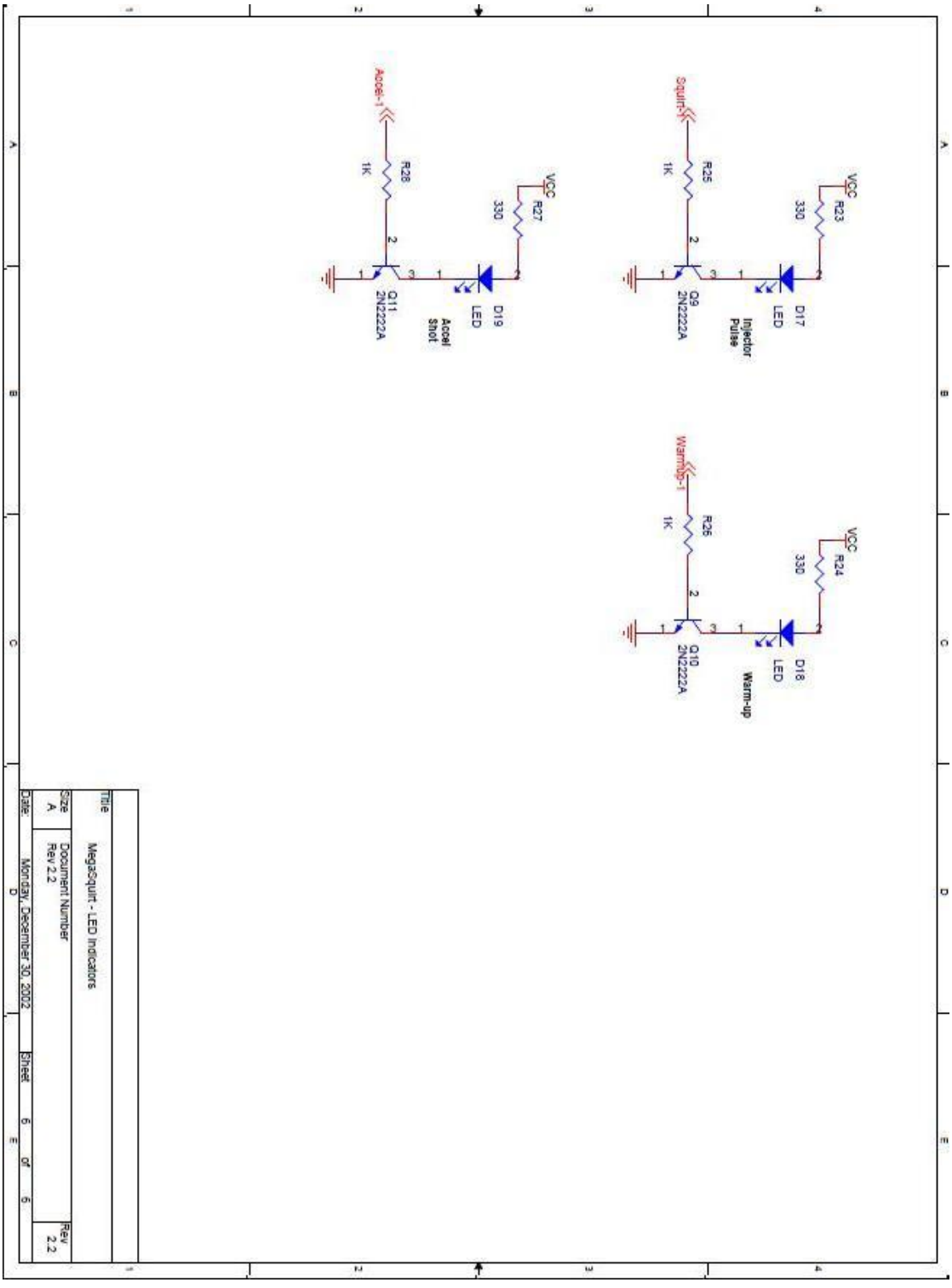
LIITE 1(5).

MegaSquirt V2.2 -kytkentäkaavio



LIITE 1(6).

MegaSquirt V2.2 -kytkentäkaavio



ApplicationSettings.java

```
package jarmo.megasquirt;

import java.io.File;

import android.content.Context;
import android.content.SharedPreferences;
import android.os.Environment;
import android.os.Handler;
import android.preference.PreferenceManager;

public enum ApplicationSettings implements SharedPreferences.OnSharedPreferenceChangeListener
{
   INSTANCE;

   private Context            context;
   private File               dataDir;
   private SharedPreferences   prefs;

   public void initialise(Context context, Handler handler)
   {
      this.context = context;
      prefs = PreferenceManager.getDefaultSharedPreferences(context);
      prefs.registerOnSharedPreferenceChangeListener(this);
      dataDir = new File(Environment.getExternalStorageDirectory(), "MsDash");
      if(!dataDir.canRead())
                  dataDir.mkdirs();
   }

   public File getDataDir()
   {
      return dataDir;
   }

   public Context getContext()
   {
      return context;
   }


                  @Override
                  public void onSharedPreferenceChanged(SharedPreferences sharedPreferences,
                                                        String key) {

                  }

}
```

BluetoothAdapterUtil.java

```
/*
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
```

```
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 */

package jarmo.megasquirt;

import android.bluetooth.BluetoothAdapter;
import android.util.Log;

/**
 *
 * @author Slavomir Hustaty
 *
 */
public class BluetoothAdapterUtil {

  // logger entry
  private final static String LOG_TAG = BluetoothAdapterUtil.class.getSimpleName();

  private static final int LOOP_WAIT_TIME = 500;

  private static final int MAX_REPETITIONS_COUNT = 30;

  public static void startBluetoothAdapter() {
    try {
      waitUntilBluetoothAdapterIsInState(BluetoothAdapter.STATE_ON,
MAX_REPETITIONS_COUNT);
    } catch (Exception e) {
      Log.d(LOG_TAG, e.getMessage());
    }
  }

  public static void stopBluetoothAdapter() {
    try {
      waitUntilBluetoothAdapterIsInState(BluetoothAdapter.STATE_OFF,
MAX_REPETITIONS_COUNT);
    } catch (Exception e) {
      Log.d(LOG_TAG, e.getMessage());
    }
  }

  /**
   * waits until BluetoothAdapter is in required state
   *
   * @param state
   * @throws Exception
   */
  private static void waitUntilBluetoothAdapterIsInState(int state, int remainingLoops) throws Excep-
tion {
    BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

    if(remainingLoops > 0) {
      switch (state) {
      case BluetoothAdapter.STATE_OFF:
```

```
      if (bluetoothAdapter.getState() == BluetoothAdapter.STATE_TURNING_OFF) {
        waitNMillis(LOOP_WAIT_TIME);
        waitUntilBluetoothAdapterIsInState(BluetoothAdapter.STATE_OFF, remainingLoops - 1);
      } else if (bluetoothAdapter.getState() == BluetoothAdapter.STATE_OFF) {
        Log.d(LOG_TAG, "BluetoothAdapter is in state OFF");
        return;
      } else {
        // ensure we're not waiting for Godot ;)
        bluetoothAdapter.disable();
        waitUntilBluetoothAdapterIsInState(BluetoothAdapter.STATE_OFF, remainingLoops - 1);
      }
      break;
    case BluetoothAdapter.STATE_ON:
      if (bluetoothAdapter.getState() == BluetoothAdapter.STATE_TURNING_ON) {
        waitNMillis(LOOP_WAIT_TIME);
        waitUntilBluetoothAdapterIsInState(BluetoothAdapter.STATE_ON, remainingLoops - 1);
      } else if (bluetoothAdapter.getState() == BluetoothAdapter.STATE_ON) {
        Log.d(LOG_TAG, "BluetoothAdapter is in state ON");
        return;
      } else {
        // ensure we're not waiting for Godot ;)
        bluetoothAdapter.enable();
        waitUntilBluetoothAdapterIsInState(BluetoothAdapter.STATE_ON, remainingLoops - 1);
      }
      break;
    default:
      throw new Exception(
          "You can check only final states of BluetoothAdapter(STATE_ON|STATE_OFF).");
    }
  } else {
    Log.e(LOG_TAG, "Error on waiting while BluetoothAdapter changes state to #" + state + ". ");
    return;
  }
 }

 /**
  * delay N milliseconds
  *
  * @param n
  */
 private static void waitNMillis(long n) {
  try {
    Thread.sleep(n);
  } catch (InterruptedException e) {
    Log.e(LOG_TAG, "#waitNMillis() " + e.getMessage());
  }
 }

}
```

ConnectService.java

```
/*
 * Copyright (C) 2009 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
```

```
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package jarmo.megasquirt;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Timer;
import java.util.TimerTask;
import java.util.UUID;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
/**
 * This class does all the work for setting up and managing Bluetooth
 * connections with other devices. It has a thread that listens for
 * incoming connections, a thread for connecting with a device, and a
 * thread for performing data transmissions when connected.
 */
public class ConnectService extends Megasquirt{
    // Unique UUID for this application
                    // Not really a unique, just working one
    private static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-8000-
00805F9B34FB");

    // Member fields
    private final BluetoothAdapter mAdapter;
    private final Handler mHandler;
    private ConnectThread mConnectThread;
    private ConnectedThread mConnectedThread;
    private int mState;

    private byte[]  sigCommand = { 83 }; // 'S'
    private byte[]  ochCommand = { 82 }; // 'R'
    public byte[]  ochBuffer;
                    public String msSig;

    // Constants that indicate the current connection state
    public static final int STATE_NONE = 0;       // we're doing nothing
    public static final int STATE_LISTEN = 1;     // now listening for incoming connections
    public static final int STATE_CONNECTING = 2; // now initiating an outgoing connection
    public static final int STATE_CONNECTED = 3;  // now connected to a remote device
    /**
     * Constructor. Prepares a new BluetoothChat session.
     * @param context  The UI Activity Context
     * @param handler  A Handler to send messages back to the UI Activity
```

```
 */
public ConnectService(Context context, Handler handler) {
    mAdapter = BluetoothAdapter.getDefaultAdapter();
    mState = STATE_NONE;
    mHandler = handler;
    ochBuffer = new byte[39];
}

/**
 * Set the current state of the chat connection
 * @param state  An integer defining the current connection state
 */
private synchronized void setState(int state) {
    mState = state;

    // Give the new state to the Handler so the UI Activity can update
    mHandler.obtainMessage(MsDash.MESSAGE_STATE_CHANGE, state, -1).sendToTarget();
}

/**
 * Return the current connection state. */
public synchronized int getState() {
    return mState;
}


/**
 * Start the ConnectThread to initiate a connection to a remote device.
 * @param device  The BluetoothDevice to connect
 */
public synchronized void connect(BluetoothDevice device) {


    // Cancel any thread attempting to make a connection
    if (mState == STATE_CONNECTING) {
        if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}
    }

    // Cancel any thread currently running a connection
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}

    // Start the thread to connect with the given device
    mConnectThread = new ConnectThread(device);
    mConnectThread.start();
    setState(STATE_CONNECTING);
    sendMessage("Connecting to: "+device.getName() + " "+device.getAddress());

}

public synchronized void connectDevice(Intent data) {
    // Get the device MAC address
    String address = data.getExtras()
        .getString(DeviceListActivity.EXTRA_DEVICE_ADDRESS);
    // Get the BLuetoothDevice object
    BluetoothDevice device = mAdapter.getRemoteDevice(address);
    MsDash.MSDevice = device;

    // Attempt to connect to the device
    this.connect(device);
```

```
    return;
}

/**
 * Start the ConnectedThread to begin managing a Bluetooth connection
 * @param socket  The BluetoothSocket on which the connection was made
 * @param device  The BluetoothDevice that has been connected
 */
public synchronized void connected(BluetoothSocket socket, BluetoothDevice device) {

    // Cancel the thread that completed the connection
    if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}

    // Cancel any thread currently running a connection
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}

    // Start the thread to manage the connection and perform transmissions
    mConnectedThread = new ConnectedThread(socket);
    mConnectedThread.start();

    // Send the name of the connected device back to the UI Activity
    Message msg = mHandler.obtainMessage(MsDash.MESSAGE_DEVICE_NAME);
    Bundle bundle = new Bundle();
    bundle.putString(MsDash.DEVICE_NAME, device.getAddress());
    msg.setData(bundle);
    mHandler.sendMessage(msg);
    //setState(STATE_CONNECTED);
}

/**
 * Stop all threads
 */
public synchronized void stop() {
    if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}
    setState(STATE_NONE);
}

/**
 * Write to the ConnectedThread in an unsynchronized manner
 * @param out The bytes to write
 * @see ConnectedThread#write(byte[])
 */
public void write(byte[] out) {
    // Create temporary object
    ConnectedThread r;
    // Synchronize a copy of the ConnectedThread
    synchronized (this) {
        if (mState != STATE_CONNECTED) return;
        r = mConnectedThread;
    }
    // Perform the write unsynchronized
    r.write(out);
}

protected void sendMessage(String msg)
{
    Message msg1 = mHandler.obtainMessage(MsDash.MESSAGE_TOAST);
    Bundle bundle = new Bundle();
```

```
    bundle.putString(MsDash.TOAST, msg);
    msg1.setData(bundle);
    mHandler.sendMessage(msg1);
}

/**
 * Indicate that the connection attempt failed and notify the UI Activity.
 */
private void connectionFailed() {
    setState(STATE_LISTEN);
    sendMessage("Unable to connect device");
}

/**
 * Indicate that the connection was lost and notify the UI Activity.
 */
private void connectionLost() {
    setState(STATE_LISTEN);
    sendMessage("Device connection was lost");
}

/**
 * This thread runs while attempting to make an outgoing connection
 * with a device. It runs straight through; the connection either
 * succeeds or fails.
 */
private class ConnectThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice mmDevice;

    public ConnectThread(BluetoothDevice device) {
        mmDevice = device;
        BluetoothSocket tmp = null;

        // Get a BluetoothSocket for a connection with the
        // given BluetoothDevice
        try {
            tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
        } catch (IOException e) {
        }
        mmSocket = tmp;
    }

    @Override
                                        public void run() {
        setName("ConnectThread");

        // Always cancel discovery because it will slow down a connection
        mAdapter.cancelDiscovery();

        // Make a connection to the BluetoothSocket
        try {
            // This is a blocking call and will only return on a
            // successful connection or an exception
            mmSocket.connect();
        } catch (IOException e) {
                    connectionFailed();
            // Close the socket
            try {
```

```
                mmSocket.close();
            } catch (IOException e2) {
            }
            // Start the service over to restart listening mode
            //ConnectService.this.start();
            return;
        }

        // Reset the ConnectThread because we're done
        synchronized (ConnectService.this) {
            mConnectThread = null;
        }

        // Start the connected thread
        connected(mmSocket, mmDevice);
    }

    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) {
        }
    }
}

/**
 * This thread runs during a connection with a remote device.
 * It handles all incoming and outgoing transmissions.
 */
private class ConnectedThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;
    Timer                        t = new Timer("IOTimer", true);

    public ConnectedThread(BluetoothSocket socket) {
        mmSocket = socket;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;

        // Get the BluetoothSocket input and output streams
        try {
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) {
        }

        mmInStream = tmpIn;
        mmOutStream = tmpOut;
    }

    @Override
    public void run()
    {
        //Log.i(TAG, "BEGIN mConnectedThread");
        try
        {
            Thread.sleep(500);
        }
```

```
    catch (InterruptedException e1)
    {
       // TODO Auto-generated catch block
       e1.printStackTrace();
    }
    try
    {
       flush();

       if (!verifySignature())
       {
          connectionFailed();
          return;
       }
       loadConstants();
       while (true)
       {
             try {
          getRuntimeVars();
          calculate(ochBuffer);

          } catch (IOException e) {
             connectionLost();
             break;
          }
       }
    }
    catch (IOException e)
    {
       //Log.e(TAG, "disconnected", e);
       connectionLost();
    }
    if (t != null)
    {
       t.cancel();
       t = null;
    }
}

private void flush() throws IOException
{
    mmOutStream.flush();
    try
    {
       Thread.sleep(100);
    }
    catch (InterruptedException e)
    {
       e.printStackTrace();
    }
    while (mmInStream.available() > 0)
    {
       mmInStream.read();
    }
}

private boolean verifySignature() throws IOException
{
    boolean verified = false;
```

```
        //byte[] sigCommand = getSigCommand();
        msSig = getSignature(sigCommand);
        String signature = "MS1/Extra format 029y3 *********";
        verified = signature.equals(msSig);
        if (verified)
        {
                setState(STATE_CONNECTED);
           sendMessage("Verified succefully signature: \n"+msSig);
        }
        else
        {
           sendMessage("Signature error! " + msSig);
        }
        return verified;
     }

     private String getSignature(byte[] sigCommand) throws IOException
     {
        String sig1 = "NoSigReadYet";
        String sig2 = "Not here";
        byte[] buf = new byte[32];
        do
        {
           write(sigCommand);

           sig1 = sig2;
           read(buf);
           sig2 = new String(buf);
           flush();
        }
        while (!sig1.equals(sig2));
        return sig1;
     }

     private void getRuntimeVars() throws IOException
     {
        write(ochCommand);
        read(ochBuffer);
     }

     /**
      * Write to the connected OutStream.
      * @param buffer  The bytes to write
      */
     public void write(byte[] buffer) {
        try {
           mmOutStream.write(buffer);
           mHandler.obtainMessage(MsDash.MESSAGE_WRITE, -1, -1, buffer)
           .sendToTarget();
           try {
                                                Thread.sleep(50);
                                        } catch (InterruptedException e) {
                                                // TODO Auto-
generated catch block
                                                e.printStackTrace();
                                        }
        } catch (IOException e) {
        }
     }
```

```
    private void read(byte[] bytes) throws IOException
    {
       TimerTask cancelTask = new TimerTask()
       {
          @Override
          public void run()
          {
             cancel();
          }
       };

       t.schedule(cancelTask, 1000);

       int nBytes = bytes.length;
       int bytesRead = 0;
       byte[] buffer = new byte[bytes.length];
       while (bytesRead < nBytes)
       {

          int result = mmInStream.read(buffer, bytesRead, nBytes - bytesRead);
          if (result == -1)
             break;

          bytesRead += result;
       }

       synchronized (bytes)
       {
          System.arraycopy(buffer, 0, bytes, 0, bytes.length);
       }
       mHandler.obtainMessage(MsDash.MESSAGE_READ, -1, -1, buffer)
       .sendToTarget();
       cancelTask.cancel();
       t.purge();
    }

    public void cancel() {
       try {
          mmSocket.close();
       } catch (IOException e) {
       }
    }
 }
  public void getPage(byte[] pageBuffer, byte[] pageSelectCommand, byte[] pageReadCommand)
throws IOException
 {
    if (mConnectedThread == null)
    {
       throw new IOException("Not connected!");
    }

    write(pageSelectCommand);
    delay(1000);
    if (pageReadCommand != null)
    {
       write(pageReadCommand);
    }
    delay(1000);
```

```
        mConnectedThread.read(pageBuffer);
    }
    private void delay(long pauseTime)
    {
        try
        {
            Thread.sleep(pauseTime);
        }
        catch (InterruptedException e) {}

    }
}
```

DeviceListActivity.java

```
/*
 * Copyright (C) 2009 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package jarmo.megasquirt;

import java.util.Set;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.Window;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.AdapterView.OnItemClickListener;

/**
 * This Activity appears as a dialog. It lists any paired devices and
 * devices detected in the area after discovery. When a device is chosen
 * by the user, the MAC address of the device is sent back to the parent
```

```
 * Activity in the result Intent.
 */
public class DeviceListActivity extends Activity {
    // Debugging
    private static final String TAG = "DeviceListActivity";
    // Return Intent extra
    public static String EXTRA_DEVICE_ADDRESS = "device_address";

    // Member fields
    private BluetoothAdapter mBtAdapter;
    private ArrayAdapter<String> mPairedDevicesArrayAdapter;
    private ArrayAdapter<String> mNewDevicesArrayAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Setup the window
        requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
        setContentView(R.layout.device_list);

        // Set result CANCELED incase the user backs out
        setResult(Activity.RESULT_CANCELED);


        // Initialize the button to perform device discovery
        Button scanButton = (Button) findViewById(R.id.button_scan);
        scanButton.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                doDiscovery();
                v.setVisibility(View.GONE);
            }
        });



        // Initialize array adapters. One for already paired devices and
        // one for newly discovered devices
        mPairedDevicesArrayAdapter = new ArrayAdapter<String>(this, R.layout.device_name);
        mNewDevicesArrayAdapter = new ArrayAdapter<String>(this, R.layout.device_name);

        // Find and set up the ListView for paired devices
        ListView pairedListView = (ListView) findViewById(R.id.paired_devices);
        pairedListView.setAdapter(mPairedDevicesArrayAdapter);
        pairedListView.setOnItemClickListener(mDeviceClickListener);

        // Find and set up the ListView for newly discovered devices
        ListView newDevicesListView = (ListView) findViewById(R.id.new_devices);
        newDevicesListView.setAdapter(mNewDevicesArrayAdapter);
        newDevicesListView.setOnItemClickListener(mDeviceClickListener);

        // Register for broadcasts when a device is discovered
        IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
        this.registerReceiver(mReceiver, filter);

        // Register for broadcasts when discovery has finished
        filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
        this.registerReceiver(mReceiver, filter);
```

```
    // Get the local Bluetooth adapter
    mBtAdapter = BluetoothAdapter.getDefaultAdapter();

    // Get a set of currently paired devices
    Set<BluetoothDevice> pairedDevices = mBtAdapter.getBondedDevices();

    // If there are paired devices, add each one to the ArrayAdapter
    if (pairedDevices.size() > 0) {
        findViewById(R.id.title_paired_devices).setVisibility(View.VISIBLE);
        for (BluetoothDevice device : pairedDevices) {
            mPairedDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress());
        }
    } else {
        String noDevices = getResources().getText(R.string.none_paired).toString();
        mPairedDevicesArrayAdapter.add(noDevices);
    }
}

@Override
protected void onDestroy() {
    super.onDestroy();

    // Make sure we're not doing discovery anymore
    if (mBtAdapter != null) {
        mBtAdapter.cancelDiscovery();
    }

    // Unregister broadcast listeners
    this.unregisterReceiver(mReceiver);
}

/**
 * Start device discover with the BluetoothAdapter
 */
private void doDiscovery() {
    Log.d(TAG, "doDiscovery()");

    // Indicate scanning in the title
    setProgressBarIndeterminateVisibility(true);
    setTitle(R.string.scanning);

    // Turn on sub-title for new devices
    findViewById(R.id.title_new_devices).setVisibility(View.VISIBLE);

    // If we're already discovering, stop it
    if (mBtAdapter.isDiscovering()) {
        mBtAdapter.cancelDiscovery();
    }

    // Request discover from BluetoothAdapter
    mBtAdapter.startDiscovery();
}

// The on-click listener for all devices in the ListViews
private OnItemClickListener mDeviceClickListener = new OnItemClickListener() {
    public void onItemClick(AdapterView<?> av, View v, int arg2, long arg3) {
        // Cancel discovery because it's costly and we're about to connect
        mBtAdapter.cancelDiscovery();
```

```
            // Get the device MAC address, which is the last 17 chars in the View
            String info = ((TextView) v).getText().toString();
            String address = info.substring(info.length() - 17);

            // Create the result Intent and include the MAC address
            Intent intent = new Intent();
            intent.putExtra(EXTRA_DEVICE_ADDRESS, address);

            // Set result and finish this Activity
            setResult(Activity.RESULT_OK, intent);
            finish();
        }
    };

    // The BroadcastReceiver that listens for discovered devices and
    // changes the title when discovery is finished
    private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();

            // When discovery finds a device
            if (BluetoothDevice.ACTION_FOUND.equals(action)) {
                // Get the BluetoothDevice object from the Intent
                BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                // If it's already paired, skip it, because it's been listed already
                if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
                    mNewDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress());
                }
            // When discovery is finished, change the Activity title
            } else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {
                setProgressBarIndeterminateVisibility(false);
                setTitle(R.string.select_device);
                if (mNewDevicesArrayAdapter.getCount() == 0) {
                    String noDevices = getResources().getText(R.string.none_found).toString();
                    mNewDevicesArrayAdapter.add(noDevices);
                }
            }
        }
    };

}
```

Megasquirt.java

```
package jarmo.megasquirt;

import java.io.IOException;


public class Megasquirt
{
                    // Page constants START
    int     alternate1;
    int     alternate2;
    int     twoStroke1;
    int     nCylinders1;
    int     divider1;
    int     twoStroke2;
```

```
int    nCylinders2;
int    divider2;
// Page constants END

// Runtime vars START
int    secl;
int    squirt;
int    engine;
int    baroADC;
int    mapADC;
int    matADC;
int    cltADC;
int    tpsADC;
int    batADC;
int    egoADC;
int    egoCorrection;
int    airCorrection;
int    warmupEnrich;
int    rpm100;
int    pulseWidth1;
int    accelEnrich;
int    baroCorrection;
int    gammaEnrich;
int    veCurr1;
int    pulseWidth2;
int    veCurr2;
int    idleDC;
int    iTime;
int    advance;
int    afrtarget;
int    fuelADC;
int    egtADC;
int    CltIatAngle;
int    KnockAngle;
int    egoCorrection2;
int    porta;
int    portb;
int    portc;
int    portd;
int    stackL;
int    tpsLast;
int    iTimeX;
int    bcDC;
// Runtime vars END

// Eqns store START
int    injOpen1;
double  boost;
int    accDecEnrich;
double  batteryVoltage;
double  coolant;
double  egoVoltage;
double  ego2Voltage;
double  mat;
int    rpm;
double  time;
double  egttemp;
double  lambda2;
double  afr2;
```

```
int     barometer;
double  map;
int     throttle;
double  advSpark;
int     KnockAng;
int     KnockDeg;
int     CltIatAng;
double  fuelvolt;
double  fuelpress;
int     altDiv1;
int     altDiv2;
double  cycleTime1;
int     nSquirts1;
double  dutyCycle1;
double  cycleTime2;
int     nSquirts2;
double  dutyCycle2;
int     veCurr;
int     pulseWidth;
int     iTimefull;
double  RpmHitmp;
double  RpmHiRes;
double  vacuum;
double  boostVac;
int     floodclear;
byte[]  sigCommand = { 83 }; // 'S'
byte[]  ochCommand = { 82 }; // 'R'

public void calculate(byte[] ochBuffer) throws IOException
{
   setupRuntime(ochBuffer);

   try
   {

      accDecEnrich = ((engine & 32) == 0) ? 100
            : ((pulseWidth1 - injOpen1) / (pulseWidth1 - (accelEnrich / 10) - injOpen1) * 100);
      batteryVoltage = batADC / 255.0 * 30.0;
      coolant = MSUtils.tempCvt(TableManager.INSTANCE.table(cltADC, "thermfactor.inc") - 40);
      egoVoltage = (egoADC / 255.0 * 5.0); // EGO sensor voltage.
      ego2Voltage = (fuelADC / 255.0 * 5.0); // EGO sensor voltage 2.
      mat = (MSUtils.tempCvt(TableManager.INSTANCE.table(matADC, "matfactor.inc") - 40));
      rpm = (rpm100 * 100); // True RPM.

      if (MsDash.CELSIUS.equals("celsius"))
      {
         egttemp = (egtADC * 3.90625); // Setup for converting 0-5V = 0 -
                                       // 1000C
      }
      else
      {
         egttemp = (egtADC * 7.15625);
      }
      // ; Added for second O2 sensor
      if (MsDash.EGO_SENSOR.equals("NARROW_BAND_EGO"))
      {
         afr2 = (TableManager.INSTANCE.table(fuelADC, "NBafr100.inc") / 100.0);
         lambda2 = (afr2 / 14.7);
      }
```

```
else if (MsDash.EGO_SENSOR.equals("ZEITRONIX_NON_LINEAR"))
{
   lambda2 = (TableManager.INSTANCE.table(fuelADC, "WBafr100Zeit.inc") / 100.0);
   afr2 = (lambda2 * 14.7);
}
else if (MsDash.EGO_SENSOR.equals("INNOVATE_LC1_DEFAULT"))
{
   lambda2 = (fuelADC / 255.0 + 0.5);
   afr2 = (lambda2 * 14.7);
}
else
{
   lambda2 = (TableManager.INSTANCE.table(fuelADC, "WBlambda100MOT.inc") / 100.0);
   afr2 = (lambda2 * 14.7);
}

if (MsDash.MAP_SENSOR.equals("MPXH6300A"))
{
   barometer = (int) ((baroADC + 1.53) * 1.213675);
   map = ((mapADC + 1.53) * 1.213675);
}
else if (MsDash.MAP_SENSOR.equals("MPXH6400A"))
{
   barometer = (int) ((baroADC + 2.147) * 1.6197783);
   map = ((mapADC + 2.147) * 1.6197783);

}
else if (MsDash.MAP_SENSOR.equals("MPX4250"))
{
   barometer = (TableManager.INSTANCE.table(baroADC, "kpafactor4250.inc"));
   map = (TableManager.INSTANCE.table(mapADC, "kpafactor4250.inc")); // Manifold
                                                    // pressure
                                                    // in kPa.
}
else
{
   barometer = (TableManager.INSTANCE.table(baroADC, "kpafactor4115.inc"));
   map = (TableManager.INSTANCE.table(mapADC, "kpafactor4115.inc"));

}

throttle = (TableManager.INSTANCE.table(tpsADC, "throttlefactor.inc"));
advSpark = ((advance * 0.352) - 10);
// ; Enhanced Stuff
KnockAng = ((KnockAngle * 90 / 256));
KnockDeg = (-KnockAng);
CltIatAng = (CltIatAngle * 90 / 256);
fuelvolt = (fuelADC < 1 ? 0.0 : fuelADC * (5 / 255) - 0.5);
fuelpress = (fuelADC < 1 ? 0.0 : fuelvolt / 0.04 + 1);
altDiv1 = (alternate1 != 0 ? 2 : 1);
altDiv2 = (alternate2 != 0 ? 2 : 1);

cycleTime1 = (rpm < 100 ? 0 : 60000.0 / rpm * (2.0 - twoStroke1));
nSquirts1 = (nCylinders1 / divider1);
dutyCycle1 = (rpm < 100 ? 0 : 100.0 * nSquirts1 / altDiv1 * pulseWidth1 / cycleTime1);

cycleTime2 = (rpm < 100 ? 0 : 60000.0 / rpm * (2.0 - twoStroke2));

nSquirts2 = (nCylinders2 / divider2);
```

```
    dutyCycle2 = (rpm < 100 ? 0 : 100.0 * nSquirts2 / altDiv2 * pulseWidth2 / cycleTime2);


    // ; These next two are needed to make the runtime dialog look good.
    veCurr = (veCurr1);
    pulseWidth = (pulseWidth1);
    iTimefull = ((iTimeX * 65536) + iTime);
    RpmHitmp = (iTimefull > 0 ? (60000000 * (2.0 - twoStroke1)) / (iTimefull * nCylinders1) : 0);
    // ; get rid of the 1 or 2 rpm display that seems to worry some
    // users
    RpmHiRes = (RpmHitmp > 20 ? RpmHitmp : 0);


    // ; Vacuum and Boost Gauges
    vacuum = ((barometer - map) * 0.2953007);// ; Calculate vacuum in
                                  // in-Hg.
    boost = (map < barometer ? 0.0 : (map - barometer) * 0.1450377);// ;
                                                  // Calculate
                                                  // boost
                                                  // in
                                                  // PSIG.
    boostVac = (map < barometer ? -vacuum : (map - barometer) * 0.1450377);

    floodclear = (tpsADC > 200 ? 1 : 0);// ; For flood clear indicator
                                // on main screen


    // Now change to 2dp
    boost = round(boost);
    batteryVoltage = round(batteryVoltage);
    coolant = round(coolant);
    egoVoltage = round(egoVoltage);
    ego2Voltage = round(ego2Voltage);
    mat = round(mat);
    time = round(time);
    egttemp = round(egttemp);
    lambda2 = round(lambda2);
    afr2 = round(afr2);
    advSpark = round(advSpark);
    fuelvolt = round(fuelvolt);
    fuelpress = round(fuelpress);
    cycleTime1 = round(cycleTime1);
    dutyCycle1 = round(dutyCycle1);
    cycleTime2 = round(cycleTime2);
    dutyCycle2 = round(dutyCycle2);
    RpmHitmp = round(RpmHitmp);
    RpmHiRes = round(RpmHiRes);
    vacuum = round(vacuum);
    boostVac = round(boostVac);

  }
  catch (Exception e)
  {
    // If we've got an arithmetic error, we've probably got duff constants.
    throw new IOException(e.getLocalizedMessage());
  }

}
```

```
public void setupRuntime(byte[] ochBuffer)
{
   secl = MSUtils.getByte(ochBuffer, 0);
   squirt = MSUtils.getByte(ochBuffer, 1);
   engine = MSUtils.getByte(ochBuffer, 2);
   baroADC = MSUtils.getByte(ochBuffer, 3);
   mapADC = MSUtils.getByte(ochBuffer, 4);
   matADC = MSUtils.getByte(ochBuffer, 5);
   cltADC = MSUtils.getByte(ochBuffer, 6);
   tpsADC = MSUtils.getByte(ochBuffer, 7);
   batADC = MSUtils.getByte(ochBuffer, 8);
   egoADC = MSUtils.getByte(ochBuffer, 9);
   egoCorrection = MSUtils.getByte(ochBuffer, 10);
   airCorrection = MSUtils.getByte(ochBuffer, 11);
   warmupEnrich = MSUtils.getByte(ochBuffer, 12);
   rpm100 = MSUtils.getByte(ochBuffer, 13);
   pulseWidth1 = MSUtils.getByte(ochBuffer, 14);
   accelEnrich = MSUtils.getByte(ochBuffer, 15);
   baroCorrection = MSUtils.getByte(ochBuffer, 16);
   gammaEnrich = MSUtils.getByte(ochBuffer, 17);
   veCurr1 = MSUtils.getByte(ochBuffer, 18);
   pulseWidth2 = MSUtils.getByte(ochBuffer, 19);
   veCurr2 = MSUtils.getByte(ochBuffer, 20);
   idleDC = MSUtils.getByte(ochBuffer, 21);
   iTime = MSUtils.getWord(ochBuffer, 22);
   advance = MSUtils.getByte(ochBuffer, 24);
   afrtarget = MSUtils.getByte(ochBuffer, 25);
   fuelADC = MSUtils.getByte(ochBuffer, 26);
   egtADC = MSUtils.getByte(ochBuffer, 27);
   CltIatAngle = MSUtils.getByte(ochBuffer, 28);
   KnockAngle = MSUtils.getByte(ochBuffer, 29);
   egoCorrection2 = MSUtils.getByte(ochBuffer, 30);
   porta = MSUtils.getByte(ochBuffer, 31);
   portb = MSUtils.getByte(ochBuffer, 32);
   portc = MSUtils.getByte(ochBuffer, 33);
   portd = MSUtils.getByte(ochBuffer, 34);
   stackL = MSUtils.getByte(ochBuffer, 35);
   tpsLast = MSUtils.getByte(ochBuffer, 36);
   iTimeX = MSUtils.getByte(ochBuffer, 37);
   bcDC = MSUtils.getByte(ochBuffer, 38);

}

public void loadConstants() throws IOException
{
                                        /*
      byte[] pageBuffer1 = new byte[189];
      byte[] pageBuffer2 = new byte[189];

      byte[] selectPage1 = { 80, 1 };
      byte[] selectPage2 = { 80, 2 };
      byte[] readPage = { 86 };
      .getPage(pageBuffer1, selectPage1, readPage);
      .getPage(pageBuffer2, selectPage2, readPage);

      alternate1 = MSUtils.getBits(pageBuffer1, 150, 0, 0);
      twoStroke1 = MSUtils.getBits(pageBuffer1, 182, 2, 2);
      nCylinders1 = MSUtils.getBits(pageBuffer1, 182, 4, 7) + 1;
      divider1 = pageBuffer1[149];
```

```
        alternate2 = MSUtils.getBits(pageBuffer2, 150, 0, 0);
        twoStroke2 = MSUtils.getBits(pageBuffer2, 182, 2, 2);
        nCylinders2 = MSUtils.getBits(pageBuffer2, 182, 4, 7) + 1;
        divider2 = pageBuffer2[149];
        */
                        alternate1 = 1;
                        if(MsDash.TwoStroke) {
                                twoStroke1 = 1;
                                twoStroke2 = 1;
                        } else {
                                twoStroke1 = 0;
                                twoStroke2 = 0;
                        }
                        nCylinders1 = Integer.valueOf(MsDash.nCylinders);
                        divider1 = 1;

                        alternate2 = 1;

                        nCylinders2 = 8;
                        divider2 = 1;

    }

    public String getDataRow()
    {
       return       "secl: " + secl + "\nsquirt: " + squirt + "\nengine: " + engine + "\nbaroADC: " +
baroADC + "\nmapADC: " + mapADC + "\nmatADC: " + matADC + "\ncltADC: " + cltADC +
"\ntpsADC: " + tpsADC
            + "\nbatADC: " + batADC + "\negoADC: " + egoADC + "\negoCorrection: " + egoCorrec-
tion + "\nairCorrection: " + airCorrection + "\nwarmupEnrich: " + warmupEnrich + "\nrpm100: "
            + rpm100 + "\npulseWidth1: " + pulseWidth1 + "\naccelEnrich: " + accelEnrich +
"\nbaroCorrection: " + baroCorrection + "\ngammaEnrich: " + gammaEnrich + "\nveCurr1: " + veCurr1
            + "\npulseWidth2: " + pulseWidth2 + "\nveCurr2: " + veCurr2 + "\nidleDC: " + idleDC +
"\niTime: " + iTime + "\nadvance: " + advance + "\nafrtarget: " + afrtarget + "\nfuelADC: "
            + fuelADC + "\negtADC: " + egtADC + "\nCltIatAngle: " + CltIatAngle + "\nKnockAngle: "
+ KnockAngle + "\negoCorrection2: " + egoCorrection2 + "\nporta: " + porta + "\nportb: " + portb
            + "\nportc: " + portc + "\nportd: " + portd + "\nstackL: " + stackL + "\ntpsLast: " + tpsLast +
"\niTimeX: " + iTimeX + "\nbcDC: " + bcDC + "\nalternate1: "
            + alternate1 + "\ntwoStroke1: " + twoStroke1 + "\nnCylinders1: " + nCylinders1 +
"\ndivider1: " + divider1 + "\nalternate2: " + alternate2 + "\ntwoStroke2: " + twoStroke2 +
"\nnCylinders2: "
            + nCylinders2 + "\ndivider2: " + divider2 + "\n\nRpmHitmp: " + RpmHitmp + "\nRpmHiRes:
" + RpmHiRes + "\nvacuum: " + vacuum + "\nboostVac: " + boostVac + "\nbarometer: " + barometer
                                + "\ncoolant: " + coolant + "\negoVoltage: " + egoVoltage + "\nmat: "
+ mat + "\nlambda2: " + lambda2 + "\nafr2: " + afr2 + "\nbatteryVoltage: " + batteryVoltage;

    }
    public double round(double v)
    {
       return Math.floor(v * 100 + .5) / 100;
    }



}
```

MsDash.java

```
package jarmo.megasquirt;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Typeface;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.preference.PreferenceManager;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.WindowManager;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

public class MsDash extends Activity {

    protected BluetoothAdapter BtAdapter = null;

    // Message types sent from the ConnectService Handler
    public static final int MESSAGE_STATE_CHANGE = 1;
    public static final int MESSAGE_READ = 2;
    public static final int MESSAGE_WRITE = 3;
    public static final int MESSAGE_DEVICE_NAME = 4;
    public static final int MESSAGE_TOAST = 5;

    // Intent request codes
    private static final int REQUEST_CONNECT_DEVICE = 1;

    // Key names received from the ConnectService Handler
    public static BluetoothDevice MSDevice = null;
    public static final String DEVICE_NAME = "device_name";
    public static final String TOAST = "toast";
    public static final String TAG = MsDash.class.getSimpleName();

                    // Display objects
                    private ImageView statusled;
    private TextView statustxt;
    private TextView devicetxt;
                    private TextView rpm;
    private TextView afr;
    private TextView boost;
    private TextView coolant;
    private TextView batvolt;
    private TextView advance;

    // getPrefs();
    public static boolean AutoConnect;
    public static boolean KeepScreenOn;
    public static boolean TwoStroke;
    public static String nCylinders;
    public static String MAP_SENSOR;
    public static String EGO_SENSOR;
```

```
    public static String CELSIUS = "celsius";

    private ConnectService connectservice = null;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate()");
        setContentView(R.layout.main);
        BtAdapter = BluetoothAdapter.getDefaultAdapter();
        if (BtAdapter == null) {
            // Device does not support Bluetooth
                    Toast.makeText(this, R.string.missing_bluetooth, Toast.LENGTH_LONG).show();
            finish();
            return;
        }
        if (!BtAdapter.isEnabled())
                    BluetoothAdapterUtil.startBluetoothAdapter();
        //Start connectservice with listener to handle messages
        connectservice  = new ConnectService(this, mHandler);
        //For reading ini tables, we don't need listener for this
        ApplicationSettings.INSTANCE.initialise(this, null);

    }

    @Override
                    public void onStart() {
        super.onStart();
        Log.d(TAG, "onStart()");
        getPrefs();
        if(AutoConnect && MSDevice != null && connectservice.getState() != Con-
nectService.STATE_CONNECTED
        && connectservice.getState() != ConnectService.STATE_CONNECTING)
                    connectservice.connect(MSDevice);
     }

    @Override
    public synchronized void onPause() {
        super.onPause();
        Log.d(TAG, "onPause()");
    }

    @Override
                    public void onResume() {
        super.onResume();
        Log.d(TAG, "onResume()");
    }

    @Override
    public void onStop() {
        super.onStop();
        Log.d(TAG, "onStop()");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy()");
```

```
    if(BtAdapter.isEnabled() && connectservice != null) {
                connectservice.stop();
                BluetoothAdapterUtil.stopBluetoothAdapter();
    }
    mHandler.removeMessages(MESSAGE_STATE_CHANGE);
    mHandler.removeMessages(MESSAGE_TOAST);
    finish();
  }

  // The Handler that gets information back from the ConnectService
  private Handler mHandler = new Handler() {

                                    @Override
    public void handleMessage(Message msg) {
      switch (msg.what) {
      case MESSAGE_STATE_CHANGE:
        Log.i(TAG, "MESSAGE_STATE_CHANGE: " + msg.arg1);
        switch (msg.arg1) {
        case ConnectService.STATE_CONNECTED:
                                                sta-
tusled.setImageDrawable(getResources().getDrawable(R.drawable.online));
                                                statustxt.setText(R.string.online);
                                                devicetxt.setText(connectservice.msSig);
                                                break;
        case ConnectService.STATE_CONNECTING:
                                                sta-
tusled.setImageDrawable(getResources().getDrawable(R.drawable.away));
          statustxt.setText("Connecting to: "+MSDevice.getName() +" "+MSDevice.getAddress());
          break;
        case ConnectService.STATE_LISTEN:
                                                sta-
tusled.setImageDrawable(getResources().getDrawable(R.drawable.busy));
          statustxt.setText(R.string.offline);
                                                devicetxt.setText("");
          connectservice.connect(MSDevice);
              break;
        case ConnectService.STATE_NONE:
          break;
        }
        break;
      case MESSAGE_WRITE:
        //byte[] writeBuf = (byte[]) msg.obj;
        // construct a string from the buffer
        //String writeMessage = new String(writeBuf);
        //mConversationArrayAdapter.add("Me:  " + writeMessage);
        //Log.i(TAG, writeMessage);
        break;
      case MESSAGE_READ:
        // construct a string from the valid bytes in the buffer
        //String readMessage = new String(readBuf, 0, msg.arg1);
        //String readMessage = new String(readBuf);
        //Log.i(TAG, readMessage);
        updateView();
        break;
      case MESSAGE_DEVICE_NAME:
        // Save the connected device's name
            MSDevice                                    =                                   BtAdapt-
er.getRemoteDevice(msg.getData().getString(DEVICE_NAME));
        SharedPreferences prefs = PreferenceManager
```

```
                .getDefaultSharedPreferences(getBaseContext());
                        prefs.edit().putString("MSDevice", MSDevice.toString()).commit();
                        break;
            case MESSAGE_TOAST:
                Toast.makeText(getApplicationContext(), msg.getData().getString(TOAST),
                        Toast.LENGTH_SHORT).show();
                break;
            }
        }
    };

    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        Log.d(TAG, "onActivityResult " + resultCode);
        switch (requestCode) {
        case REQUEST_CONNECT_DEVICE:
            // When DeviceListActivity returns with a device to connect
            if (resultCode == Activity.RESULT_OK) {
                connectservice.connectDevice(data);
            }
            break;
        }
    }

    private void getPrefs() {
        // Get the xml/preferences.xml preferences
        SharedPreferences prefs = PreferenceManager
                .getDefaultSharedPreferences(getBaseContext());
        AutoConnect = prefs.getBoolean("autoconnect", false);
        KeepScreenOn = prefs.getBoolean("screenon", false);
        TwoStroke = prefs.getBoolean("twostroke", false);
        nCylinders = prefs.getString("nCylinders", null);
        MAP_SENSOR = prefs.getString("maptype", null);
        EGO_SENSOR = prefs.getString("egotype", null);

        // Read default MS address from memory
                if(prefs.getString("MSDevice", null) != null)
                        MSDevice = BtAdapt-
er.getRemoteDevice(prefs.getString("MSDevice", null));

        // Ask kindly to keep screen on
        if(KeepScreenOn)
                getWin-
dow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        else
                getWin-
dow().clearFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);

        statusled = (ImageView) findViewById(R.id.statusled);
        statustxt = (TextView) findViewById(R.id.statustxt);
        devicetxt = (TextView) findViewById(R.id.devicetxt);

        rpm = (TextView) findViewById(R.id.rpm);
        afr = (TextView) findViewById(R.id.afr);
        boost = (TextView) findViewById(R.id.boost);
        coolant = (TextView) findViewById(R.id.coolant);
        batvolt = (TextView) findViewById(R.id.batvolt);
        advance = (TextView) findViewById(R.id.advance);

        //setup font
```

```
   Typeface font = Typeface.createFromAsset(getAssets(), "fonts/digital.ttf");
   rpm.setTypeface(font);
   afr.setTypeface(font);
   boost.setTypeface(font);
   coolant.setTypeface(font);
   batvolt.setTypeface(font);
   advance.setTypeface(font);

   updateView();
}

public void updateView() {

                //set values
   rpm.setText("RPM: "+connectservice.RpmHiRes);
   afr.setText("AFR: "+connectservice.afr2);
   boost.setText("MAP: "+connectservice.boostVac);
   coolant.setText("CLT: "+connectservice.coolant);
   batvolt.setText("BAT: "+connectservice.batteryVoltage);
   advance.setText("ADV: "+connectservice.advSpark);

   try {
                                        Thread.sleep(50);
                        } catch (InterruptedException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                        }

}

@Override
public boolean onPrepareOptionsMenu(Menu menu)
{
   super.onPrepareOptionsMenu(menu);

   //Disable connect button if everything isn't set up
   MenuItem item = menu.findItem(R.id.connect);
   if(nCylinders == null || MAP_SENSOR == null || EGO_SENSOR == null) {
      Toast.makeText(getApplicationContext(), R.string.setup, Toast.LENGTH_LONG).show();
      item.setEnabled(false);
   } else
                item.setEnabled(true);
   return true;
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
   MenuInflater inflater = getMenuInflater();
   inflater.inflate(R.menu.menu, menu);
                return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
                Intent serverIntent = null;
                switch (item.getItemId()) {
                case R.id.connect:
                                serverIntent = new Intent(this, DeviceListActivity.class);
```

```
                                    startActivityForResult(serverIntent,
REQUEST_CONNECT_DEVICE);
                                    return true;
                        case R.id.preferences:
            Intent settingsActivity = new Intent(this,
                  Preferences.class);
            startActivity(settingsActivity);
                                    return true;
                        }
                        return false;
    }
}
```

MsUtils.java

```
package jarmo.megasquirt;


public class MSUtils
{
   public static int getLong(byte[] ochBuffer, int i)
   {
      return getWord(ochBuffer, i) * 65536 + getWord(ochBuffer, i + 2);
   }

   public static int getWord(byte[] ochBuffer, int i)
   {

      return (getByte(ochBuffer,i) * 256 + getByte(ochBuffer,i+1));
   }

   public static int getByte(byte[] ochBuffer, int i)
   {
      return (int) ochBuffer[i] & 0xFF;
   }

   public static int getSignedLong(byte[] ochBuffer, int i)
   {
      int x = getLong(ochBuffer, i);
      if (x > 2 << 32 - 1)
      {
         x = 2 << 32 - x;
      }
      return x;
   }

   public static int getSignedByte(byte[] ochBuffer, int i)
   {

      int x = getByte(ochBuffer, i);
      if (x > 127)
      {
         x = 256 - x;
      }
      return x;
   }

   public static int getSignedWord(byte[] ochBuffer, int i)
   {
```

```
        int x = getWord(ochBuffer, i);
        if (x > 32767)
        {
           x = 32768 - x;
        }
        return x;
    }

    public static double tempCvt(int t)
    {
        return (t - 32.0) * 5.0 / 9.0;
    }

    public static int getBits(byte[] pageBuffer, int i, int _bitLo, int _bitHi)
    {
        int val = 0;
        byte b = pageBuffer[i];

        long mask = ((1 << (_bitHi - _bitLo + 1)) - 1) << _bitLo;
        val = (int) ((b & mask) >> _bitLo);

        return val;
    }
}
```

Preferences.java

```
package jarmo.megasquirt;

import android.os.Bundle;
import android.preference.PreferenceActivity;

public class Preferences extends PreferenceActivity {
	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		addPreferencesFromResource(R.xml.preferences);
	}
}
```

TableManager.java

```
package jarmo.megasquirt;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import jarmo.megasquirt.ApplicationSettings;
import android.content.res.AssetManager;
```

```
import android.util.Log;

public enum TableManager
{
		INSTANCE;

		private Map<String, List<Integer>>		tables		=		new
HashMap<String, List<Integer>>();

		public synchronized void flushTable(String name)
		{
				tables.remove(name);
		}

		public synchronized int table(int i1, String name)
		{
				List<Integer> table = tables.get(name);
				if (table == null)
				{
						table = new ArrayList<Integer>();
						readTable(name, table);
						tables.put(name, table);
				}
				return table.get(i1);

		}

		private void readTable(String fileName, List<Integer> values)
		{
				values.clear();
				Pattern p = Pattern.compile("\\s*[Dd][BbWw]\\s*(\\d*).*");

				String assetFileName = "tables" + File.separator + fileName;
				File 		override=new
File(ApplicationSettings.INSTANCE.getDataDir(),fileName);
				AssetManager		assetManager		=		ApplicationSet-
tings.INSTANCE.getContext().getResources().getAssets();

				BufferedReader input = null;
				try
				{
						try
						{
								InputStream data = null;
								if(override.canRead())
								{
										data = new FileIn-
putStream(override);

		Log.d(MsDash.TAG,"opening tables from external memory: "+override);
								}
								else
								{

		Log.d(MsDash.TAG,"opening default tables: "+assetFileName);
										data = assetManag-
er.open(assetFileName);
								}
```

```
                                                input = new BufferedReader(new
InputStreamReader(data));
                                                String line;
                                                while ((line = input.readLine()) !=
null)
                                                {
                                                        Matcher matcher =
p.matcher(line);
                                                        if (matcher.matches())
                                                        {
                                                                String
num = matcher.group(1);
                                                                if
(num != null)
                                                                {
                        values.add(Integer.valueOf(num));
                                                                }
                                                        }
                                                }
                                        }
                                        finally
                                        {
                                                if (input != null)
                                                        input.close();
                                        }
                                }
                                catch (IOException e)
                                {
        Log.e(MsDash.TAG,"TableManager.readTable("+fileName+")",e);
                                }
                        }
}
```