TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Tietokonetekniikan suuntautumisvaihtoehto

Insinöörityö

Toni Palosaari

GPS-NOPEUSMITTARI

Työn valvoja: Kai Poutanen

Tekijä: Toni Palosaari
Työn nimi: GPS-nopeusmittari

Päivämäärä: 23.4.2007
Sivumäärä: 23 + 4 liitesivua
Hakusanat: AVR, GPS
Koulutusohjelma: Tietotekniikka
Suuntautumisvaihtoehto: Tietokonetekniikka

Työn valvoja: Kai Poutanen

Työssä suunniteltiin ja toteutettiin AVR Butterfly -kokeilukortille ohjelma, jolla luetaan kortille liitetyn GPS-vastaanottimen dataa ja poimitaan sieltä nopeustieto tulostettavaksi näytölle. Kokeilukorttiin tutustuttiin myös muilta osin. Tavoite oli siis tehdä toimiva digitaalinen nopeusmittari, jossa on myös toiminto, joka ottaa muistiin ja näyttää maksiminopeuden. Nopeusmittari on suunniteltu sijoitettavaksi autoon, joten se on otettu työssä huomioon kytkentöjen ja ohjelman osalta. Mikrokontrollerina Butterfly-kortilla on ATMega169. Ohjelman kirjoittamiseen ja kääntämiseen käytetyt ilmaiseksi saatavilla olevat ohjelmat olivat AVR Studio ja AVR-GCC C -kääntäjä. Kokeilukortti liitettiin tietokoneen rinnakkaisporttiin ohjelmointia varten ja sarjaporttiin testauksen ajaksi. Työn tavoite täyttyi ja toimivaa nopeusmittaria päästiin testaamaan, ja se osoittautui erittäin toimivaksi.

Author: Toni Palosaari
Name of the thesis: GPS-Speedometer

Date: 23.4.2007
Number of pages: 23 + 4 attachment pages
Keywords: AVR, GPS
Degree programme: Computer Systems Engineering
Specialization: Embedded Systems

Thesis supervisor: Kai Poutanen

This engineering thesis is about designing and making of a GPS-speedometer. It is based on an evaluation board called AVR Butterfly and a GPS-receiver. The GPS-receiver sends data and the program in the microcontroller picks the speed from the data and shows it on a display. There is also a function which stores maximum speed and it can be also shown on the display and resetted. The speedometer is designed to be installed in a car, and that has been noted while designing electrical connections and the program. The microcontroller on the evaluation board is ATMega169. The applications used to write and compile the program are freely available and they are called AVR Studio and AVR-GCC C -compiler. While programming, the evaluation board was connected to the computer's parallel port and testing was done using the serial port. The goal of this thesis was achieved by testing a working speedometer and it was also proven useful.

ALKUSANAT

Tämän työn tarkoitus on kertoa digitaalisen nopeusmittarin tekemisestä, ja se rakentuu GPS-vastaanottimesta ja mikrokontrollerin kokeilukortista. Näiden laitteiden yhdistäminen oli mahdollista mikrokontrollerille itse suunnitellun ohjelman avulla. Lisäksi käytettyä kokeilukorttia esitellään myös nopeusmittariprojektiin kuulumattomin osin.

Työtä oli mielenkiintoinen tehdä, koska siinä yhdistyi kaksi laitetta uudella ja hyödyllisellä tavalla. Haluankin kiittää kaikkia, jotka ovat olleet mukana edistämässä työn valmistumista.

Tampereella 23.4.2007

_____
Toni Palosaari

SISÄLLYSLUETTELO

LYHENTEET JA SYMBOLIT

GPS	Global Positioning System, satelliittipaikannusjärjestelmä
RMC	Recommended Minimum Specific GPS/TRANSIT Data, suositeltu minimi GPS-data
ISP	In System Programming, ohjelmointitapa
LCD	Liquid Crystal Display, nestekidenäyttö
EEPROM	Electronically Erasable Programmable Read-Only Memory, haihtumaton puolijohdemuisti
SRAM	Static Random Access Memory, haihtuva käyttömuisti
USART	Universal Synchronous/Asynchronous Receiver-Transmitter, synkroninen/asynkroninen lähetin-vastaanotin, sarjaliikenne
DCE	Data Circuit-terminating Equipment, verkkopääte
DTE	Data Terminal Equipment, terminaali

1 JOHDANTO

Navigointilaitteista tuttu tarkka nopeusmittari on hyvä lisä autoon tai mihin tahansa kulkuneuvoon. Varsinkin auton omassa mittarissa on aina mukana mittarivirhettä, jolloin todellisen nopeuden arviointi on hankalaa.

GPS-pohjaisen mittarin hyvänä puolena on sen erinomaisen tarkkuuden lisäksi siirrettävyys, koska kiinteitä, esimerkiksi auton vanteeseen asennettavia antureita ei tarvita. Nyt mittari voidaan tarvittaessa helposti siirtää eri kulkuneuvoihin. Myöskään erikokoiset rengastukset eivät vaikuta mitenkään kalibrointiin.

Käyttötarkoituksia löytyy jokapäiväisen ajon lisäksi myös suorituskyky-mittauksista, sillä huippunopeus voidaan määrittää tarkasti ilman tutkaa. Nopeusmittaria ei ole tarkoitettu ainoaksi nopeusnäytöksi autoon, koska perinteinen viisarimittari puolustaa paikkaansa paremman luettavuutensa takia. Lisäksi oma vaikutuksensa on myös GPS-vastaanottimen antaman nopeustiedon pienellä viiveellä verrattuna reaaliaikaiseen nopeuteen.

Mikrokontrolleri on Atmel AVR Butterfly -kokeilukorttiin sisältyvä ATMega169. Kokeilukortin hyviä puolia tätä työtä ajatellen on siinä valmiiksi oleva LCD-näyttö ja sarjaliikenteen tasomuunnoksen tekevä piiri GPS-laitteen ja mikrokontrollerin välille.

Työn tavoite on siis tehdä GPS-vastaanottimesta ja AVR Butterfly -kokeilukortista toimiva, autoon asennettava nopeusmittari.

2 GPS-VASTAANOTIN JA NMEA-LAUSE

Ensimmäiseksi tutkitaan käytetyn Globalsat BR-304 GPS-vastaanottimen ominaisuuksia. Käyttöjännitteenä se tarvitsee 5 VDC, joka saadaan auton sähköjärjestelmästä reguloimalla. Virrankulutus on 80 mA. Käytetty piirisarja on SiRF Star II/LP. Vastaanottokanavia on 12, mikä on siis maksimi yhtäaikaisesti seurattavien satelliittien määrä. Käyttölämpötila voi olla väliltä -40°C ~ 85°C. Paikannus pitäisi tapahtua yleensä alle 45 sekunnin kuluttua käyttöjännitteen kytkennästä. Nopeuden mittaamisen tarkkuus on 0,1 m/s 95 %. /1/

Vastaanotin on todella pienikokoinen ja hyvin suojattu koteloonsa. Merkkivalo ilmoittaa laitteen tilan. Mikäli laite on sammuksissa, niin merkkivalo ei tietenkään pala. Käyttöjännitteet kytkettynä ja vastaanottimen hakiessa vielä signaalia merkkivalo palaa jatkuvasti ja sen vilkkuminen on merkki siitä, että paikannus on onnistunut. Vastaanotin nähdään kuvassa 1.

Kuva 1 Globalsat BR-304

Vastaanotin lähettää datan kerran sekunnissa NMEA 0183 -standardin mukaisena ja RS232-sarjaliikenneportin kautta. Tämä asettaa rajoituksen nopeusmittarin näytön päivittymisnopeudelle, koska useammin kuin kerran sekunnissa ei saada uutta nopeustietoa. Datan lähetysnopeus ei ole kovin suuri, 4800 bittiä/sekunti. Jokainen lähetettävä merkki on muodossa 8N1, joka tarkoittaa 8 databittiä, pariteettibitti ei ole käytössä ja yksi loppubitti. Seuraavassa GPS-datan lähetysmuodosta esimerkki:

```
$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W
*6A
```

Tätä kutsutaan RMC-lauseeksi. BR-304 lähettää myös muita NMEA-standardin mukaisia lausetyyppejä, mutta tässä työssä keskitytään vain RMC-lauseen

käyttöön, koska se sisältää tarvittavan nopeustiedon. RMC on myös varma valinta, mikäli halutaan käyttää jossain vaiheessa jotain muuta GPS-paikanninta, koska se on yleisin lausetyyppi. Erikoisia lausetyyppejä käytettäessä ei olisi varmuutta, lähettävätkö kaikki laitteet juuri kyseistä lausetta. Lauseessa tiedot on eroteltu omiin kenttiinsä toisistaan pilkuilla. Nopeustieto on 8. kentässä ja se ilmoitetaan solmuina tunnissa. Kilometreiksi tunnissa muuntaminen tapahtuu kertoimella 1,852. Nopeus on saatavissa yhden desimaalin tarkkuudella. Mikäli GPS-signaalissa on katkos tai laite ei ole vielä paikantanut itseään, jolloin nopeuttakaan ei voida ilmoittaa, on nopeuskenttä lauseessa tyhjä. Lauseen päättyminen voidaan havaita rivinvaihtomerkistä.

Muut kentät lauseessa ovat alusta lähtien järjestyksessä; kellonaika, tila A=aktiivinen, leveyskoordinaatti, pituuskoordinaatti, nopeus solmuina, suunta asteina (ei voida määrittää paikallaan ollessa), päiväys, magneettisen ja oikean pohjoissuunnan erotus ja viimeisenä tarkistussumma. /2/

Voidakseen paikantaa itsensä, vastaanottimen tarvitsee vastaanottaa vähintään 4 satelliitin signaalit. Paikannuksen tarkkuus on sitä parempi, mitä enemmän satelliitteja se kuulee. Signaalin taajuussiirtymästä (Doppler) vastaanotin laskee satelliitin ja vastaanottimen nopeuden toistensa suhteen. Tähän perustuu GPS-laitteiden nopeustiedon tarkkuus, joka ei siis riipu paikannuksen tarkkuudesta. /8/

3 AVR BUTTERFLY

AVR Butterfly on Atmelin valmistama sulautettujen järjestelmien kokeilukortti, joka perustuu ATMega169 mikrokontrollerin ympärille. Se on tarkoitettu mikrokontrollerin ohjelmoinnin kokeiluun ja se sopii myös prototyypin rakentamiseen. Jopa lopullinen laite voi perustua siihen, koska kortin virrankulutus on pieni ja se toimii pitkään mukana tulevalla nappiparistolla. Myös ulkoista jännitelähdettä voidaan käyttää.

Kortilla on myös hyvät liitännät. Joystick-ohjain ja LCD-näyttö löytyvät valmiina, ja niitä voi käyttää kokeiltavissa sovelluksissa. Laitteeseen on aluksi valmiina ohjelmoituna ohjelma, joka käyttää hyväksi kortin eri ominaisuuksia ja esittelee toimintoja. Tämän ohjelman C-kielinen lähdekoodi on myös saatavilla, ja siitä voi tutkia, kuinka eri ominaisuuksia käytetään.

3.1 ATMega169-mikrokontrolleri

Atmelin AVR-sarjaan kuuluu kolmenlaisia mikrokontrollereita. Niitä ovat tinyAVR, Classic AVR ja megaAVR ja ne kaikki perustuvat Harvard-arkkitehtuuriin ja ovat RISC-tyyppisiä. Harvard-arkkitehtuuri tarkoittaa rakennetta, jossa data ja ohjelma ovat erillisissä muisteissa. RISC-prosessoreiden hyvä ominaisuus on pieni käskykanta, joista suurin osa voidaan suorittaa yhdellä kellojaksolla. MegaAVR-sarjan mikrokontrollereissa on enemmän muistia ja liitäntäjalkoja verrattuna Classic AVR-sarjaan ja tinyAVR:ään. /3/

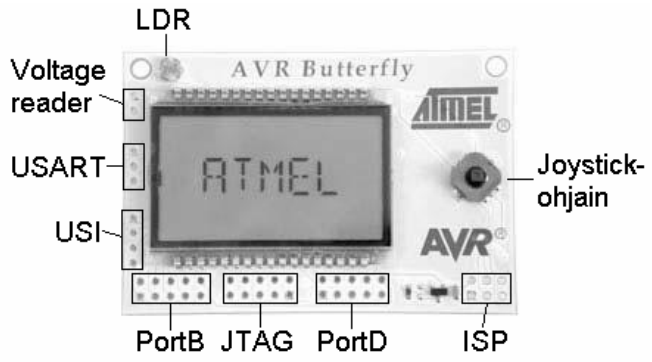
Tärkeimpiä ATMega169:n ominaisuuksia ovat /5/

- kellotaajuus korkeintaan 8 MHz
- 16 kilotavun flash-ohjelmamuisti, 512 tavun EEPROM-datamuisti ja 1 kilotavun SRAM-käyttömuisti
- 8-kanavainen 10-bittinen AD-muunnin
- JTAG-liitäntä debuggaukseen
- USART-sarjaliikenneväylä
- sisäinen ja ulkoinen keskeytys
- virrankulutus on 350 µA kellotaajuudella 1 MHz ja minimikäyttöjännitteellä 1,8 V

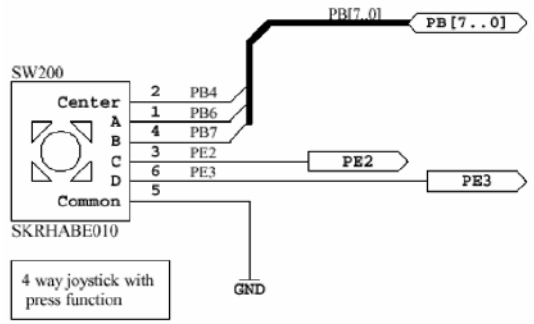
3.2 Liitännät

Kuvassa 2 esitellään AVR Butterflyn liitännät. Nopeusmittarin tekemistä varten niistä tarvitaan USART sarjaliikennettä varten GPS:ltä päin, ISP ohjelmointiin ja PortD:n kaksi liitintä jännitteen syöttöön. 8-bittisiä portteja voidaan käyttää kaksisuuntaisesti, joko sisäänmenoina tai ulostuloina, ja niissä on sisäiset ylösvetovastukset /5/.

Voltage reader on AD-muuntimelle kytketty ja sisäisen jännitteenjaon omaava liitäntä, ja sitä voidaan käyttää 0-5 V:n jännitteiden lukemiseen. LDR-valovastusta voidaan käyttää valoisuuden mittaamiseen. Joystick-ohjaimen pinneistä, jotka nähdään kuvassa 3, osa on kytketty portB-pinneihin ja osa PortE-pinneihin, joten näitä portteja käytettäessä on otettava se huomioon. Lisäksi kortilla on piezokaiutin äänen tuottamiseen ja NTC-vastus lämpötilan mittaamiseen. Näyttö on tyypiltään segmenttinäyttö ja siinä on 120 segmenttiä, joista ATMega voi käyttää 100 segmenttiä. Käytännössä näytölle mahtuu kerralla 6 merkkiä. /6/



Kuva 2 AVR Butterfly liitännät



Kuva 3 Joystick-kytkentä

3.3 Muut ominaisuudet

Datan tallettamiseen on kortille lisätty 4 Mbit:n flash-muistipiiri. Se on kytketty kortin SPI-väylään, jolloin siihen päästään käsiksi ISP-ohjelmoinnilla. /6/

Käyttöjännite kortilla voi olla välillä 3,1 – 4,5 V ja käyttölämpötila 0 - 50 °C. ATMega169 voi käyttää joko sisäistä tai ulkoista kideoskillaattoria. Kortille sijoitettua 32 kHz:n kideoskillaattoria voidaan käyttää referenssinä, kun mikrokontrolleria ajetaan esimerkiksi 1 MHz:n kellotaajuudella. Näin saadaan tarkasti toimiva kellotus, joka mahdollistaa esimerkiksi tarkan ajassa pysyvän kellonajan toteutuksen ja se parantaa myös huomattavasti USART-väylän kautta tapahtuvan sarjaliikenteen luotettavuutta, mikä on tarpeen suurilla tiedonsiirtonopeuksilla. Kellotaajuutta voidaan muuttaa ohjelmallisesti asettamalla sille jakaja. /6/

4 ATMEGA169-OHJELMOINTI

Koska AVR Butterfly on tarkoitettu mikrokontrollerin ohjelmoinnin kokeiluun, on siinä toimitettaessa valmiiksi ohjelmoituna bootloader-ohjelma. Se ottaa ohjelmoitavan HEX-tiedoston vastaan USART-liitännän kautta. Tämä ei ole siis varsinaisesti ohjelmointitapa, koska bootloader hoitaa ohjelmoinnin. /6/

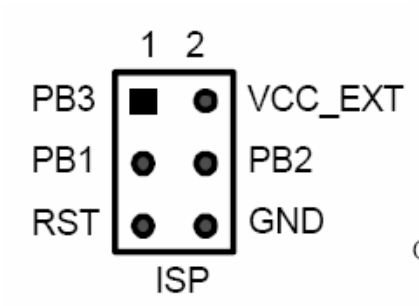
Bootloader asettaa omat ohjelmat flash-muistiin, mutta jättää sinne myös itsensä. Se vastaa myös kontrollerin virranhallinnasta siten, että kytkettäessä käyttöjännite vasta kortilla olevan ohjaimen painallus ylös käynnistää oman ohjelman. Tämä ei ole toivottava käytös tätä työtä ajatellen, joten ohjelmointi suoritettiin ISP-ohjelmoinnilla. Tällä tavalla oma ohjelma ohjelmoidaan ATMEGA169:n flash-muistiin, ja samalla on mahdollisuus haluttaessa ohjelmoida EEPROM-muistia sekä ”Fuse-” ja ”Lock-bittejä”. Toinen hyvä puoli on, että USART-liitäntä vapautuu ja sitä voidaan käyttää testaukseen tietokoneen sarjaliitännän kautta tai liittämällä GPS-vastaanotin siihen. Näin liittimiä ei tarvitse vaihdella testauksen ja ohjelmoinnin välillä.

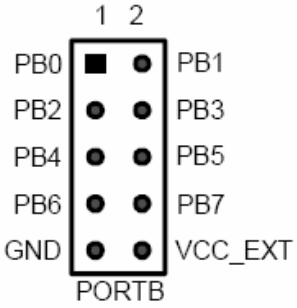
4.1 ISP-ohjelmointi

AVR Butterfly -kortin ISP-liitäntää käytetään liittämällä se tietokoneeseen rinnakkaisportin kautta, ja ohjelmoinnin aikana on myös käyttöjännitteen oltava kytkettynä VCC_EXT-liitäntään. Nopeusmittarin ohjelmoinnissa käyttöjännite oli kytkettynä sekä ISP-liittimeen (kuva 4), että B-portin liittimeen (kuva 5). Taulukossa 1 esitetään käytetyt liitännät kortin ISP-liitännän ja tietokoneen LPT-portin välillä. Avrdude-ohjelmalle oli myös määritettävä nämä samat pinnit asetustiedostoon. LPT-portin jännitetaso on juuri sopiva, joten kytkentä ei tarvitse kuin suorat johtimet LPT-pinneistä ISP-liitäntään.

Taulukko 1 ISP-liitännät

LPT-pinni	ISP-pinni
1	sck/pb1
2	mosi/pb2
11	miso/pb3
16	reset





Kuva 4 Butterfly ISP-liitin Kuva 5 Butterfly PortB-liitin

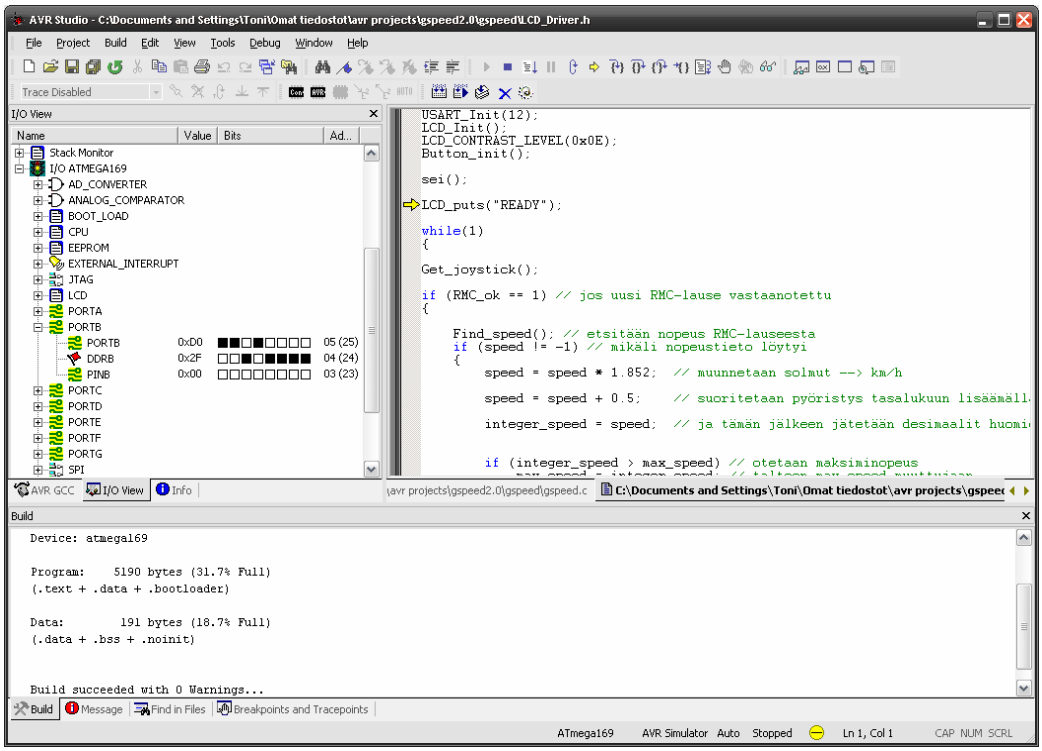
4.2 AVR Studio

ATMEL tarjoaa ilmaista ohjelmointiympäristöä nimeltä AVR Studio. Se toimii ohjelmien kehitystyökaluna assembly- ja C-kielillä ja sisältää simulointi-ominaisuuden, jolla eri rekisterien sisältöjä voidaan tutkia ajamalla ohjelmaa jatkuvasti tai käsky kerrallaan.

Mikäli käytössä on JTAG-kaapeli, niin myös debuggaus on sillä mahdollista, jolloin voidaan simuloinnin sijasta ohjelmaa ajaa mikrokontrollerilla joko jatkuvasti tai käsky kerrallaan ja tarkkailla muuttujien sisältöjä ja rekisterejä. Oletuksena ohjelma sisältää vain assembly-kääntäjän.

Kuvassa 6 esitellään AVR Studion käyttöliittymä. Kuvassa ohjelmalla on juuri käännetty ja ajettu ohjelmakoodia. Vasemmalla olevassa paneelissa voidaan tutkia mikrokontrollerin eri rekisterien sisältöjä ja muita arvoja. Esimerkiksi porttien tilat voidaan tarkistaa kätevästi, kuten on tehty kuvassa juuri PORTB:n

kohdalla. Siitä nähdään, että bitit 7, 6 ja 4 on asetettu sisäänmenoiksi ja ne on asetettu loogiseksi ykköseksi sisäisen ylösvetovastuksen avulla.



Kuva 6 AVR Studion käyttöliittymä

4.3 AVR-GCC C -kääntäjä

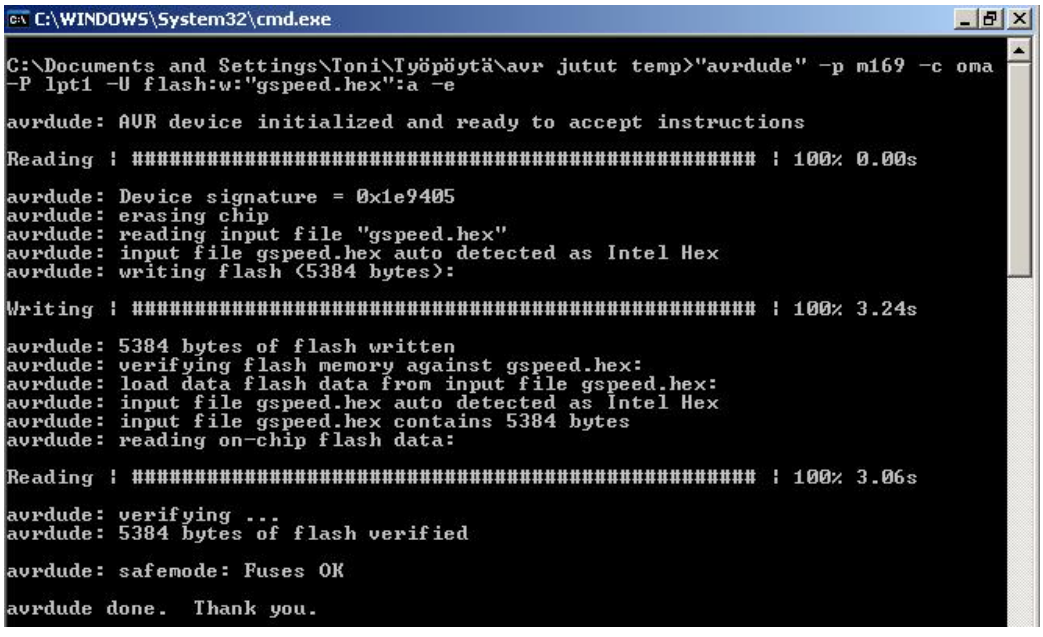
Winavr-ohjelmistopaketti sisältää työkaluja AVR-mikrokontrollerien ohjelmistokehitykseen ja ohjelmointiin. Tärkeimpänä se sisältää ilmaisen AVR-GCC C -kääntäjän, joka integroituu osaksi AVR Studiota. Näin AVR Studiolla voidaan suoraan kirjoittaa ja kääntää C-koodia. AVR-GCC on hyvin dokumentoitu ja siitä on tullut suosittu kääntäjä harrastelijoiden keskuudessa. Muita Winavr-paketin osia ovat:

- avr-libc, C-kääntäjän kirjastotiedot
- GDB, virheenjäljitystyökalu
- avrdude, ohjelmointiohjelma
- avarice, JTAG-liitännän käyttöön liittyvä ohjelma
- programmers notepad, tekstieditori etenkin ohjelmointiin

4.4 Avrdude

Komentorivillä toimiva avrdude on tehokas työkalu AVR-ohjelmointiin, ja sitä on käytettävä ISP-ohjelmoinnissa, koska AVR Studio ei sitä tue. Sille annetaan käytetty ohjelmointimenetelmä, ohjelmoitava hex-tiedosto ja muut asetukset parametreinä. Graafisiin ohjelmointityökaluihin tottuneelle se voi vaikuttaa hankalalta, mutta on silti aivan yhtä hyvä pienen totuttelun jälkeen. Usein toistuvan ohjelmoinnin hoitaa helpoiten komentojonotiedostolla. Kuvassa 7 on esimerkki avrdudella tapahtuvasta ohjelmoinnista, josta nähdään myös annetut parametrit. Ohjelmoitaessa nähdään myös ohjelmoitavan tiedoston koko. Kuvasta nähdään, että siinä ohjelmoitaessa täytetään noin kolmasosa ATMega169:n 16 kilotavun flash-muistista. Koska avrdude ei voi tietää, miten ohjelmoitava piiri on tietokoneeseen liitetty, niin ISP-liitännän pinnit on määritetty avrdude.conf-tiedostoon lisäämällä siihen seuraavanlaiset rivit.

```
programmer
  id    = "oma";
  desc  = "ISP liitäntä";
  type  = par;
  sck   = 1;
  mosi  = 2;
  miso  = 11;
  reset = 16;
;
```



Kuva 7 Avrduden käyttö

5 GPS-NOPEUSMITTARI

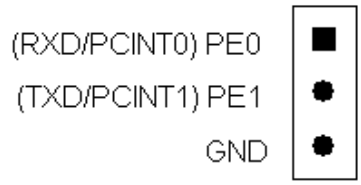
Tässä kappaleessa kuvataan nopeusmittarin rakentamiseen tarvittavat sähköiset kytkennät ja perehdytään muutamiin muihin tärkeisiin seikkoihin.

5.1 RS232-liitäntä

Mikrokontrollerin USART-väylän loogisten bittien jännitetasot ovat välillä 0-5 V. Tietokoneen RS232-liitännän jännitetasot vastaavasti ovat $\pm 5..\pm 12$ V /7/. Välissä tarvitaan siis tasomuunnos, jonka Butterfly-kortilla hoitaa integroitu muunninpiiri /6/.

Näin yhteys USART-RS232 voidaan toteuttaa suoraan normaalilla sarjakaapelilla. Liittimenä käytetään DB9-liitintä. Kuvassa 8 nähdään Butterflyn USART-liitäntä.

GPS-laitteen liittämiseksi kortille tarvitaan väliin kääntävä eli nollamodeemikaapeli, koska GPS on DCE-laite. Normaali suora kaapeli on tarkoitettu DTE-DCE -yhteyksiin, mutta kytkettäessä yhteen GPS ja Butterfly, kyseessä on DCE-DCE. Nollamodeemikaapelissa RX ja TX on kytketty ristiin.

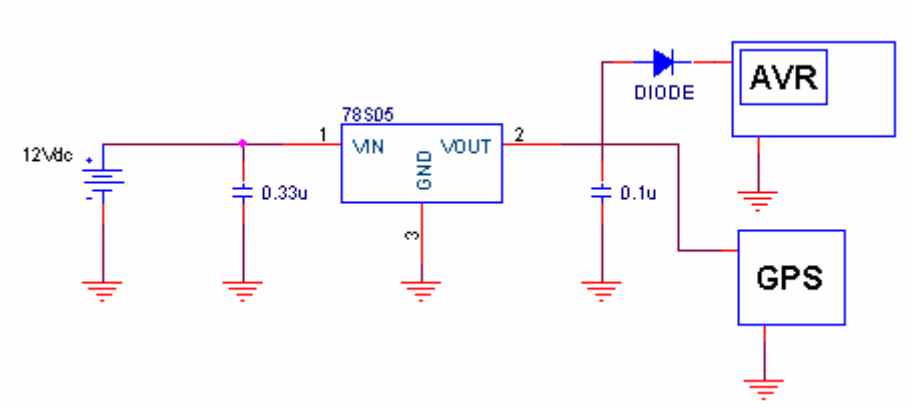


Kuva 8 USART-liitäntä

5.2 Käyttöjännitteen syöttö

Lopullisessa autoon tapahtuvassa asennuksessa käyttöjännitteen syötöstä vastaa 7805-sarjan regulaattori, jonka ulostulosta saadaan 5 VDC. Regulaattorin syötöstä ja ulostulosta on kytkettävä kondensaattorit maahan datalehden osoittamalla tavalla. Suositellut arvot kondensaattoreille ovat 0,33 µF syötössä ja 0,1 µF lähdössä /9/.

Butterflylle jännitettä alennetaan yhdellä diodilla, jotta ei ylitetä maksimiksi suositeltua 4,5 V:n käyttöjännitettä. Kytkentäkaavio käyttöjännitteen syötöstä on kuvassa 9.



Kuva 9 Käyttöjännitteiden kytkentäkaavio

6 OHJELMAN ESITTELY

GPS-datan vastaanottaminen ja siitä tarvittavan nopeustiedon poimiminen ja näytöllä näyttäminen ovat siis ohjelman tärkeimmät tehtävät. Tässä kappaleessa kuvataan ohjelman toiminta ja tutkitaan mahdollisia lisäominaisuuksia. Ohjelma on kirjoitettu aikaisemmin esitellyllä AVR Studiolla ja käännetty AVR-GCC C -kääntäjällä ilman optimointia. Ohjelmakoodissa esiintyvät rekisterit ja niiden bittien merkitykset löytyvät ATMega169:n datalehdestä /5/.

6.1 Ohjelman rakenne

Sulautettujen järjestelmien ohjelmat ovat lähes poikkeuksetta ikuisessa silmukassa pyöriviä, kuten tämäkin. Alkumäärittelyjen jälkeen ohjelmassa mennään ikuiseen silmukkaan. Jotta ei tuhlattaisi mikrokontrollerin suorituskykyä jatkuvaan USART-väylän pollaamiseen, on otettu käyttöön sarjaliikennekeskeytys. Se aiheutuu, kun USART-väylässä havaitaan vastaanotettu merkki. Keskeytysohjelmassa kerätään talteen RMC-lause, jota käsitellään nopeustiedon löytämiseksi omassa aliohjelmassaan. Lopuksi pääohjelmassa nopeustieto muutetaan näytölle sopivaan muotoon ja ohjataan näytölle. Joystick-ohjainta ylöspäin painamalla saadaan näytölle maksiminopeus, ja alaspäin painallus nollaa maksiminopeuden.

6.2 Alustukset

Kaikki ohjelmassa suoritettavat alkuarvojen ja muuttujien sekä asetusten alustukset on koottu tähän kappaleeseen.

6.2.1 Sarjaliikenne

USART_init vastaa nimensä mukaisesti sarjaliikenteen asetuksista asettamalla tarvittavat bitit oikeisiin rekistereihin. Seuraavassa esitellään sarjaliikenteen asetuksiin käytetty koodi.

```
void USART_Init(unsigned int baudrate)
{
    // asetetaan bittinopeus
    UBRRH = (unsigned char)(baudrate>>8);
    UBRRL = (unsigned char)baudrate;

    // sallitaan vastaanotto ja vastaanottokeskeytys
    UCSRB = (1<<RXEN)|(0<<TXEN)|(1<<RXCIE)|(0<<UDRIE);

    // asynkroninen moodi 8N1
    UCSRC = (0<<UMSEL)|(0<<UPM0)|(1<<USBS)|(3<<UCSZ0)|(0<<UCPOL);
}
```

Aliohjelmalle annetaan parametrina numeroarvo, joka lasketaan halutun sarjaliikennenopeuden perusteella. GPS-lähettää dataa 4800 bit/s nopeudella, ja siitä seuraavan kaavan perusteella ja käyttäen kellotaajuutta 1 MHz saadaan tasaluvuksi pyöristettynä 12, ja se asetetaan tiedonsiirtonopeusrekisterin arvoksi.

UBRR = ((kellotaajuus/(16 * bittinopeus)) – 1
= ((1 000 000 / (16 * 4800)) – 1
≈ 12,0208

Seuraavaksi asetetaan UCSRB-rekisteriin RXEN-bitti eli vastaanotto sallituksi, kielletään lähetys TXEN, sallitaan vastaanottokeskeytys RXCIE ja kielletään lähetyspuskurin tyhjenemiskeskeytys UDRIE.

Kehysmuoto on 8 databitistä muodostuva, ilman pariteettibittiä ja 1 loppubitti. Se asetetaan käyttöön UCSRC-rekisteriin. Lisäksi käytetään asynkronista muotoa, koska datassa ei ole mukana kellotussignaalia. UMSEL-bitti asetetaan nollaksi, jotta valitaan asynkroninen moodi. Pariteettibitin käyttö kielletään asettamalla 0<<UPM0. USBS määrää lähetettävien loppubittien määrän, joten se ei vaikuta vastaanottoon. Asettamalla UCSZ0:n arvoksi 3, valitaan 8-bittinen datankentän pituus. UCPOL vaikuttaa vain synkronista tiedonsiirtoa käytettäessä.

6.2.2 Joystick-ohjain

Jotta joystick-ohjaimen tila voidaan lukea, niin täytyy se ensin ottaa käyttöön sopivilla asetuksilla. Seuraavaksi esitellään joystick-ohjaimen alustukseen tarvittava koodi.

```
void Button_init(void)
{
DDRB = 0x2F;            // portin suunta sisään
PORTB |= PINB_MASK;     // asetetaan tarvittavat ylösvetovastukset
}
```

Ohjaimen kolme signaalia (painallus, ylös ja alas) ovat kiinni B-portin biteissä 7, 6 ja 4. Loput suunnat on kytketty eri porttiin, mutta koska ohjelmassa käytetään ohjainta vain kahteen toimintoon, riittää kolme signaalia, joista yksi jää varalle. Aluksi portti on asetettava sisäänmenoksi edellä mainittujen bittien osalta. Seuraavaksi samoille biteille asetetaan ylösvetovastukset bittikuvion avulla, jolloin bitit asettuvat loogiseen ykköseen. Bittikuvio on määritelty pääohjelman

alkumäärityksissä. Ohjaimen painallus kytkee painetun suunnan pinnin PortB:stä maahan, jolloin luettaessa ohjaimen bitit painettua suuntaa vastaava bitti on 0.

6.2.3 Pääohjelmassa tapahtuvat alustukset

Tässä kappaleessa esitellään alustukset, jotka tapahtuvat pääohjelmassa ja lisäksi alussa tarvittavat määrittelyt.

```
#include "LCD_Driver.h"
#include "LCD_Driver.c"
#define F_CPU 1000000 // kellotaajuus
#include "stdio.h" // sprintf
#include <stdlib.h> // atof

ISR(USART0_RX_vect); // sarjaliikennekeskeytys

// aliohjelmien esittelyt
void Find_speed(void);
void USART_Init(unsigned int baudrate);
void Button_init(void);
void Get_joystick(void);

// char-muuttujien alustus
char RMC_string[85], rx;
char output_to_LCD[6];
unsigned char RMC_counter = 0, RMC_place = 0;
unsigned char RMC_ok = 0;
unsigned char joystick;
unsigned char mode = 1;

// muut muuttujat
float speed = 0.0;
int max_speed = 0;
int integer_speed = 0;

#define PINB_MASK ((1<<PINB4)|(1<<PINB6)|(1<<PINB7))
```

Alussa sisällytetään näytönohjauksen aputiedostot. Näytönohjaus tarvitsee myös tiedon käytetystä kellotaajuudesta, joksi on asetettu 1 MHz. Ohjelmaan täytyy sisällyttää otsikkotiedostoista stdio.h, koska merkkijonon käsittelyyn käytetty funktio sprintf on määritelty siinä. Stdlib.h sisällytetään, koska siinä määritellään ascii-merkit float-tyyppiseksi muuttava atof.

Seuraavaksi esitellään aliohjelmat, jotka kaikki ovat void-tyyppisiä, koska ohjelmassa käytetään globaaleja muuttujia paluuarvojen sijaan. Char-muuttujat alustetaan seuraavaksi. RMC_string saa pituudeksi 85 alkiota, jotta se riittää RMC-lauseen vastaanottoon. Näyttöbufferi output_to_LCD alustetaan 6 merkin mittaiseksi, koska näytöllekin mahtuu kerrallaan 6 merkkiä. RMC-lauseen käsittelyssä tarvittavat apumuuttujat määritellään unsigned char -tyyppisiksi, mikä tarkoittaa etumerkitöntä muuttujaa. Char-tyyppi on 8-bittinen muuttuja, joten joystickista luettavat bitit mahtuvat myös siihen. Lopuksi alustetaan näyttömoodin määräävä mode-muuttuja valmiiksi arvoon 1, joka on normaali näyttämä nopeusmittarissa eli nopeusnäyttö. Lopuksi määritetään muut tarvittavat muuttujat, joista tärkeimpänä nopeus eli speed-muuttuja liukulukutyyppiseksi. Maksiminopeus max_speed ja näytölle tulostettava nopeus

integer_speed alustetaan kokonaislukutyyppisiksi. Lopuksi määritetään PINB_MASK vastaamaan bittikuviota, jolla ohjain on kytketty B-porttiin.

Ennen pääohjelmassa olevaa ikuista silmukkaa, on vielä joitain alkumäärityksiä.

```
int main(void)

{
CLKPR = (1<<CLKPCE);     // kellon jakaja käyttöön
                         // asetetaan jakaja = 8, 8Mhz/8 = 1Mhz
CLKPR = (1<<CLKPS1) | (1<<CLKPS0);

USART_Init(12);
LCD_Init();
LCD_CONTRAST_LEVEL(0x0E);
Button_init();

sei();
LCD_puts("READY");
```

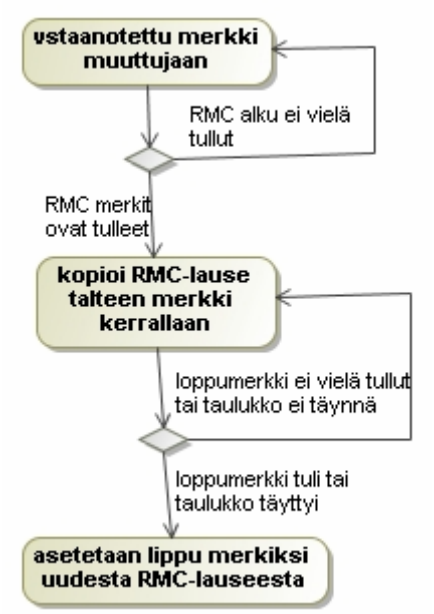
CLKPR on kellotaajuuden jakajarekisteri ja CLKPCE on kellotaajuuden jakajan käyttöönottobitti. Koska ohjelma ei ole kovin vaativa, voidaan ATMega169 asettaa toimimaan normaalia pienemmällä kellotaajuudella. Myös nopeusmittarin käyttö mahdollisesti kannettavana laitteena paristojen varassa onnistuu paremmin, koska pienempi kellotaajuus kuluttaa vähemmän virtaa. Datalehden mukaan asetetaan bitit jakajarekisteriin, jotta saadaan käyttöön 1 MHz kellotaajuus, kun jakaja on 8. Seuraavaksi kutsutaan alustuksia tekeviä aliohjelmia, asetetaan näytön kontrasti ja lopuksi sallitaan keskeytykset sei()-käskyllä. Näytölle tulostetaan READY-teksti merkiksi, että ohjelma on käynnissä ja valmis vastaanottamaan GPS-dataa. Seuraavaksi siirrytään pääohjelman ikuiseen while-silmukkaan, joka esitellään myöhemmin.

6.3 Sarjaliikennekeskeytys

Kun sarjaliikennekeskeytys on asetettu ja keskeytykset sallittu sei()-käskyllä, niin aina merkin ollessa luettavissa USART-väylässä hypätään sarjaliikennekeskeytysohjelmaan. Rivin ISR(USART0_RX_vect) nolla tarkoittaa ensimmäistä USART-väylää, mikäli mikrokontrollerilla on useampia USART-väyliä. Alla on sarjaliikennekeskeytysohjelman koodi sekä toimintakaavio kuvassa 10.

```
ISR(USART0_RX_vect)      // sarjaliikennekeskeytys
{

rx = UDR;                // vastaanotettu merkki (UDR) muuttujaan rx

if (rx == 'R')           // mahdollinen RMC-lauseen alku
 RMC_counter=1;

if (rx == 'M')
 RMC_counter++;

if (rx == 'C')
 RMC_counter++;

if (RMC_counter == 3)    // RMC-lauseen alku on tullut kokonaan
{
```

```
if (rx == '\r' || rx == '\n' || RMC_place > 84) // jos tullut rivinalku
                                                // tai rivinvaihto
{
                                    // niin RMC-lause on tullut loppuun
            RMC_ok = 1;             // tässä lippu merkiksi että RMC-
                                    // lause on vastaanotettu
            RMC_place = 0;          // nollataan käytetyt laskurit
            RMC_counter=0;
}

else        // lause on vielä kesken ja sitä otetaan talteen
{
            RMC_ok = 0;             // RMC ei vielä tullut
                                    // loppumerkkiin asti
            RMC_string[RMC_place] = rx; // tallennetaan vastaanotettu
                                    // merkki RMC_string merkkijonoon
            RMC_place++;            // kasvatetaan muuttujaa jotta
                                    // seuraava merkki menee seuraavaan
                                    // alkioon
```



Kuva 10 Keskeytysohjelman toimintakaavio

Periaatteena on tunnistaa peräkkäiset merkit R, M ja C, jolloin tiedetään että RMC-lauseen alku on tullut, koska niitä ei esiinny peräkkäin muualla NMEA-lauseissa. Seuraavaksi vastaanotettu merkki rx-muuttujassa tallennetaan RMC_string-taulukon ensimmäiseen alkioon ja kasvatetaan RMC_place-muuttujaa yhdellä, jotta seuraava merkki menee seuraavaan alkioon. Näin jatketaan kunnes tulee vastaan rivinvaihtomerkki tai cr eli kursorin siirto alkuun. Samaan ehtolauseeseen on asetettu myös ehdoksi RMC_place > 84, joka estää taulukon ylivuodon, mikäli loppumerkki jostain syystä sattuu menemään ohi ja sitä ei tunnisteta. Molemmat merkit, rivinvaihto ja kursorin siirto ovat mukana siitä syystä, että se helpottaa testausta tekstimuodossa olevalla GPS-datalla. Lopuksi asetetaan RMC_ok-lippu merkiksi, jotta pääohjelmassa tiedetään uuden RMC-lauseen tulleen, ja nollataan vielä käytetyt laskurit.

6.4 RMC-lauseen käsittely

Nopeustiedon löytämiseen ja muuttujaan sijoittamiseen käytettävä aliohjelma on Find_speed(). Sillä käsitellään vastaanotettua RMC-lausetta pääohjelmasta kutsuttuna. Nopeustiedon se asettaa globaaliin float-tyyppiseen muuttujaan nimeltä speed.

```
void Find_speed(void) // hakee nopeustiedon RMC-lauseesta muuttujaan
{
unsigned char X, Y = 0, tmp, comma = 0; // apumuuttujien alustus
char tmp_string[7] = {0,0,0,0,0,0,0};   // merkkijonotaulukko johon
                                        // nopeus luetaan

for (X=0; X<86; X++)     // pyörii silmukassa maksimissaan RMC_string-
                         // taulukon loppuun
{

tmp = RMC_string[X];     // merkki taulukosta tmp-muuttujaan
if (tmp == ',')
            comma++;     // pilkkulaskuri

if (comma == 7)          // kun on löytynyt 7 pilkkua
break;                   // hypätään pois for-silmukasta
}
if (RMC_string[X+1] == ',') // onko seuraavakin merkki pilkku
speed = -1;              // nopeustietoa ei ollut

else
{
do                       // otetaan merkit talteen tmp_string-
                         // merkkijonotaulukkoon kunnes tulee pilkku
{
X++;
tmp_string[Y] = RMC_string[X];
Y++;

if ( X==85 || Y>7 )      // taulukoiden ylivuodon esto
{
X=0;
Y=0;
}

}
while ( RMC_string[X] != ',');

}

speed = atof(tmp_string);           // merkkijonotaulukko -> float
muuttuja

RMC_ok = 0;                         // nollataan RMC_ok jotta samaa
                                    // nopeustietoa ei käsitellä
                                    // uudestaan pääohjelmassa
```

Alussa määritellään apumuuttujat ja nollataan niiden arvot. Nopeustiedon lukemista varten varataan 7 merkkiä pitkä merkkijonotaulukko. Seuraavaksi pyöritetään for-silmukkaa, kunnes RMC-lauseesta löytyy 7 pilkkua jolloin seuraavasta merkistä alkaa nopeustieto. Mikäli seuraavakin merkki pilkun jälkeen on pilkku, niin nopeustietoa ei ollut, koska GPS ei ole vielä löytänyt sijaintiaan eikä osaa täten laskea nopeutta, tai signaalissa voi olla katkos. Tässä tapauksessa nopeudeksi asetetaan arvo -1, joka osataan pääohjelmassa tulkita virheelliseksi nopeustiedoksi ja toimia sen mukaan. Mikäli nopeustieto löytyy,

niin se otetaan merkki kerrallaan talteen tmp_string-merkkijonotaulukkoon kunnes vastaan tulee jälleen pilkku. Tässäkin yhteydessä varaudutaan virhetilanteisiin tarkistamalla taulukoiden indeksit, jotta ne eivät pääse kasvamaan yli määriteltyjen pituuksien. Virhetilanteessa indeksit nollataan. Merkkijonotaulukossa oleva nopeustieto on muutettava liukuluvuksi, jotta sille voidaan suorittaa matemaattisia operaatioita. Muuntaminen tapahtuu käskyllä atof (ascii to float). Lopuksi nollataan RMC_ok-lippu, jotta samaa nopeustietoa ei käsitellä uudelleen pääohjelmassa.

6.5 Näytön ohjaus

Näytön ohjaukseen käytetään valmista ”ajuria” /4/. Butterfly-kortille löytyy useampikin tällainen aputiedosto, jotka sisältävät aliohjelmia merkkien tulostamiseksi sen integroidulle näytölle. Esimerkiksi alkuperäisen esimerkkiohjelman lähdekoodeista löytyvät tiedostot LCD_driver.c ja LCD_driver.h joilla homma hoituu, kunhan ne sisällytetään projektiin includoimalla. Nopeusmittariprojektiin käytetty ajuri on edellä mainitusta hieman yksinkertaistettu versio. Siinä käytetään vain yksinkertaista aliohjelmakutsua LCD_puts(”tekstiä”), tai tulostettaessa merkkijonotaulukko kutsutaan LCD_puts(näyttöbufferi). Esimerkiksi nopeusmuuttujan tulostamiseksi näytölle on se ensin muutettava takaisin merkkijonotaulukoksi ns. näyttöbufferiin. Näytölle mahtuu kerrallaan vain 6 merkkiä, joten ohjelman näytölle tulostukset on mietitty sen mukaan. Yli 6 merkkiset tulostukset onnistuvat myös, mutta silloin niitä rullataan näytöllä ja se huonontaa luettavuutta.

6.6 Joystick-ohjaimen lukeminen

Ohjaimen lukeminen tapahtuu aliohjelmalla Get_joystick(). Siinä luetaan B-portin bitit joystick-muuttujaan. Bittejä luettaessa B-portin bitit käännetään vastakkaisiksi aalto-merkin avulla ja tehdään niiden kanssa and-operaatio ennalta määritetyn bittikuvion avulla. Näin painetun suunnan bitti asettuu ykköseksi joystick-muuttujaan ja siitä päätellään haluttu toiminto. Mikäli painetaan ylös, vaikuttaa se kuudenteen bittiin ja painallus alas vaikuttaa seitsemänteen bittiin. Ehtojen avulla asetetaan globaaliin mode-muuttujaan haluttu toiminto ja mikäli mitään ei painettu, niin pidetään nopeusnäyttö. Koodi esitellään alla.

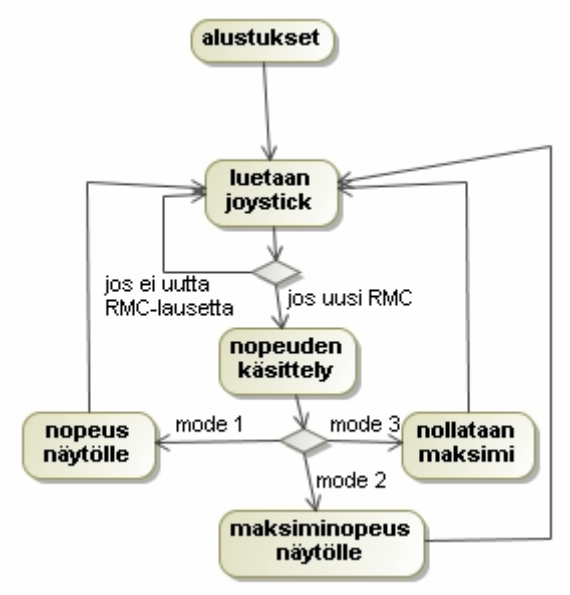
```
void Get_joystick(void)
{
joystick = (~PINB) & PINB_MASK;

if (joystick & (1<<6))
            mode = 2;   // suunta ylös, maksiminopeus

else if (joystick & (1<<7))
            mode = 3;   // suunta alas, maksiminopeuden nollaus

else mode = 1;          // muuten näytetään nopeutta

}
```

6.7 Pääohjelma

Varsinainen pääohjelma alkaa päättymättömästä while-silmukasta. Sulautettujen järjestelmien ohjelmat perustuvat lähes poikkeuksetta päättymättömiin silmukoihin, koska silloin kontrolli ei karkaa ohjelmasta ja tarkoitus on suorittaa haluttua toimintoa niin kauan, kuin käyttöjännite on kytketty laitteelle. Seuraavassa esitellään pääohjelma while-silmukan osalta.

Aluksi luetaan ohjaimen asento. Mikäli uuden RMC-lauseen merkiksi asetettu lippu on 1, etsitään nopeustieto lauseesta. Mikäli lippu on 0, niin pääohjelmassa ei suoriteta muuta. Tällä estetään saman lauseen käsittely useammin kuin kerran. Nopeus muutetaan kilometreiksi tunnissa muuttamisen jälkeen tasaluvuksi. Pyöristys tapahtuu lisäämällä float-tyyppiseen nopeuteen 0,5 ja tämän jälkeen jättämällä desimaaliosa pois kopioimalla nopeus int-tyyppiseen integer_speed-muuttujaan. Näytöllä olisi ollut mahdollisuus esittää myös desimaaliosa, mutta luettavuuden kannalta tasalukuinen nopeus on parempi, ja 1 km/h resoluutio riittää aivan hyvin. Seuraavaksi otetaan nopeus talteen max_speed-muuttujaan, mikäli se oli suurempi kuin edellinen talletettu arvo. Kuvattu ohjelmakoodi on seuraavanlainen. Sen toimintakaavio nähdään kuvassa 11.

```
while(1)
{

Get_joystick();

if (RMC_ok == 1)         // jos uusi RMC-lause vastaanotettu
{

 Find_speed();           // etsitään nopeus RMC-lauseesta
 if (speed != -1)        // mikäli nopeustieto löytyi
 {
             speed = speed * 1.852;  // muunnetaan solmut --> km/h

             speed = speed + 0.5;    // suoritetaan pyöristys tasalukuun
                                     // lisäämällä 0,5

             integer_speed = speed;  // ja tämän jälkeen jätetään
                                     // desimaalit huomioimatta

             if (integer_speed > max_speed) // otetaan maksiminopeus
                         max_speed = integer_speed; // talteen max_speed
                                                    // muuttujaan

}
```



Kuva 11 Pääohjelman toimintakaavio

Tässä kohtaa päätetään näytölle tulostus. Mikäli ohjainta ei paineta mihinkään suuntaan, niin tulostetaan nopeutta näytölle. Ennen tulostusta tarkistetaan että nopeus on väliltä 0-399 eli järkevä arvo. Kokonaislukutyyppinen integer_speed muutetaan merkkijonoksi output_to_LCD-näyttöbufferiin sprintf-apufunktion avulla. Nopeuden perään lisätään KPH (kilometres per hour) osoittamaan nopeuden olevan muodossa km/h. Määritys 3i tasaa nopeuden oikeasta reunasta siten, että kun nopeus muuttuu esimerkiksi 5 -> 15 -> 125, niin viimeinen numero pysyy kokoajan paikallaan. Tämä parantaa huomattavasti nopeusmittarin luettavuutta, kun numerot eivät ”pompi”. Mikäli nopeus oli -1 tai jotain muuta, niin se tulkitaan virheelliseksi tai olemattomaksi nopeustiedoksi ja näytölle tulostetaan NO FIX. Tämä tapahtuu, kun GPS-signaalissa on katkos ja nopeuskenttä jää tyhjäksi, tai heti virran kytkemisen jälkeen GPS-vastaanottimelle ennen kuin se on laskenut sijaintinsa.

Mikäli mode-muuttujan arvo on 2 eli ohjainta painetaan ylöspäin, niin näytölle tulostetaan maksiminopeus samalla tyylillä tasattuna kuin nopeusnäytössäkin, mutta MAX loppuliitteen kera. Ja mikäli ohjainta painetaan alas ja mode saa arvon 3, tulostetaan näytölle RESET ja nollataan nykyinen maksiminopeus. Tämän ominaisuuden avulla voidaan ottaa maksiminopeuksia useista ajoista esimerkiksi autoon tehtyjen säätöjen jälkeen.

```
if (mode == 1) // "normaalimoodi", tulostetaan nopeustieto näytölle

{

if (integer_speed >= 0 && integer_speed < 400)  // tarkistetaan että on
                                                // järkevä nopeus
            {
            sprintf(output_to_LCD, "%3iKPH", integer_speed);
            // näyttöpuskurin päivitys
            LCD_puts(output_to_LCD);
            // näyttöpuskuri näytölle

            }
```

```
            else                    // satelliitteja ei vielä lukittu
            LCD_puts("NO FIX")      // tai signaalissa katkos
}

if (mode == 2)          // maksiminopeus tulostetaan näytölle

{
sprintf(output_to_LCD, "%3iMAX", max_speed);
LCD_puts(output_to_LCD);
}

if (mode == 3)          // maksiminopeus nollataan

{
LCD_puts("RESET");
max_speed = 0;
}
}
}
}
```

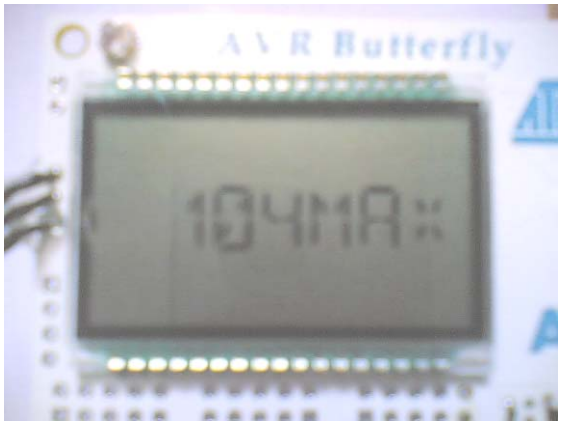
7 TESTAUS

Nopeusmittarin toimintaa on tietenkin testattava. Siihen on olemassa kaksi eri tapaa. Tietokoneen sarjaportin kautta voidaan lähettää tekstitiedostoon tallennettuja NMEA-lauseita, tai sitten kytketään itse GPS-laite ja lähdetään liikkumaan, jotta saadaan nopeustietoa. Koska käytössä ei ollut JTAG-kaapelia, niin testaaminen onnistui vain antamalla syötettä ja seuraamalla tuloksia. Virheiden hakuun JTAG olisi ollut erittäin kätevä.

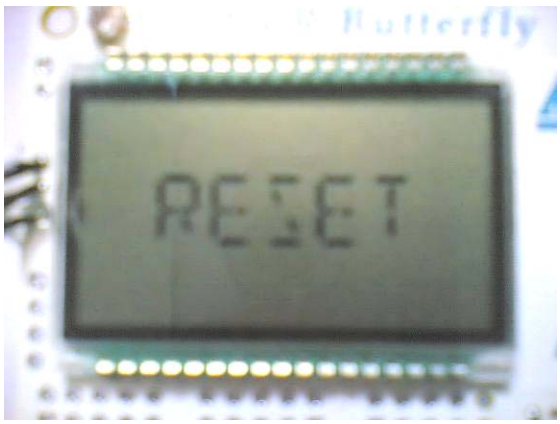
Sarjaportin asetukset täytyi laittaa vastaavaksi kuin GPS-vastaanottimella, jonka jälkeen pystyi tietokoneen terminaaliohjelmalla lähettämään tekstitiedoston sisällön Butterfly-kortille. Alkuvaiheissa ohjelma toimi muuten hyvin, mutta aina tietyssä kohdassa lähetettäviä NMEA-lauseita näyttö sekosi. Syylliseksi paljastui taulukon ylivuoto. RMC-lausetta kerättäessä ei loppumerkkiä jostain syystä havaittu. Nopeustiedoksi meni mitä sattuu merkkejä ja se aiheutti näytön sekoamisen. Lisäämällä tarkistukset kaikkiin käytettyihin taulukoihin ylivuodon varalta, ei ongelmia enää esiintynyt.

Koska ohjelma toimi hyvin jo aiemmilla testeillä, niin se toimi hyvin myös varsinaisen GPS-laitteen kanssa. Mitään virheitä ei esiintynyt käytössä. Ohjaimen painalluksen reagoinnissa olisi hieman parantamisen varaa, koska se vaihtelee riippuen siitä, missä kohti RMC-lauseen vastaanotto ja pääohjelma ovat menossa. Vasteaikaa voisi parantaa ottamalla käyttöön keskeytyksen, joka laukeaa B-portin bittien muutoksista, sekä muuttamalla pääohjelman rakennetta. Kuvissa 12, 13 ja 14 nähdään ohjelman eri näyttötilat.

Kuva 12 Nopeusnäyttö



Kuva 13 Maksiminopeusnäyttö



Kuva 14 Maksiminopeuden nollaus

8 YHTEENVETO

Projektin tärkeimmäksi osuudeksi nousi tietenkin itse ohjelma ja sen testaus vaihe vaiheelta. Nopeusmittarin sähköiset kytkennät ovat varsin yksinkertaiset. Tähän työhön ei kuulunut kovin syvällinen perehtyminen itse GPS-tekniikkaan, koska siitä saisi tehtyä jo oman työnsä. Kirjoitusvaiheessa oli päätettävä, mitä asioita painotetaan eniten. Tämän takia ohjelma vie suurimman osan, sillä onhan se sulautetuissa järjestelmissä muutenkin kaiken ”ydin”. Esitellyistä lähdekoodeista saa aikaan toimivan nopeusmittarin ja niitä voi helposti soveltaa myös muille mikrokontrollereille.

Kaiken kaikkiaan nopeusmittarista tuli erittäin onnistunut ja asetettu tavoite toimivasta nopeusmittarista täyttyi. Testeissä se osoittautui erittäin käteväksi auton lisänäytöksi, josta voi tarkastaa tarkan ajonopeuden. Tämän takia se tullaankin asentamaan kiinteästi vuosimallin 1994 Ford Fiestaan. Auton alkuperäisen kellon paikalle asennus onnistuu huomaamattomasti. GPS-datan mukana lähetettävää kellonaikaakin voisi lukea ohjelmaan ja lisätä siten kellonäytön. Näin alkuperäisen kellon tilalle saisi edelleen näkymään kellonajan, joka on aina oikeassa ajassa kiitos satelliittien atomikellojen. Näytölle tarvitsee lisäksi kehitellä taustavalo, jotta siitä saa selvää hämärässäkin.

Nopeuden päivittymisnopeus 1 kerta/s on sopiva ainakin kaupunkiolosuhteissa, mutta esimerkiksi maantiellä se voisi olla harvemmin päivittyvä. Tätä varten ohjelmaan lisätään mahdollisesti kolmesta viimeisimmästä nopeudesta keskiarvoistus, joka otetaan käyttöön vaikkapa yli 70 km/h nopeuksissa. Tällä tavalla nopeusnäytön päivitystahti pienenisi kolmannekseen, mutta silti säilyttäen tarkkuuden.

On myös paljon muita käyttömahdollisuuksia, joihin mikrokontrollerin ja GPS-laitteen yhdistelmää autossa voitaisiin käyttää. Se voitaisiin liittää esimerkiksi varashälyttimeen tai matkapuhelimeen, jolloin auton paikkatietojen kysely onnistuisi vaikkapa tekstiviestillä. Lisäominaisuuksia tullaan nopeusmittariin tekemään ja edellä mainittu voisi olla yksi niistä.

LÄHTEET

1. Globalsat. [www-sivu]. [viitattu 26.3.2007]
http://www.globalsat.com.tw/eng/product_detail1_00000047.htm

2. Gpsinformation.org [www-sivu]. [viitattu 26.3.2007]
http://www.gpsinformation.org/dale/nmea.htm

3. Wikipedia. [www-sivu]. [viitattu 26.3.2007]
http://en.wikipedia.org/wiki/Atmel_AVR

4. Avrfreaks.net [www-sivu]. [viitattu 26.3.2007]
http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=36234&sid=b5da144ff4815051cd0bc7bdfd6a214f

5. ATMega169-datakirja. [sähköinen dokumentti]. [viitattu 26.3.2007]
http://www.atmel.com/dyn/resources/prod_documents/2514S.pdf

6. AVR Butterfly –datakirja. [sähköinen dokumentti]. [viitattu 26.3.2007]
http://www.atmel.com/dyn/resources/prod_documents/doc4271.pdf

7. Wikipedia. [www-sivu]. [viitattu 26.3.2007]
http://fi.wikipedia.org/wiki/RS232

8. Netlab.tkk.fi [www-sivu]. [viitattu 26.3.2007]
http://www.netlab.tkk.fi/opetus/s38118/s98/htyo/8/yleiskuva.shtml

9. 7805-datakirja. [sähköinen dokumentti]. [viitattu 26.3.2007]
http://www.datasheetarchive.com/datasheet.php?article=623405

LIITTEET

1. GPS-nopeusmittarin lähdekoodi, pituus 4 sivua

```
#include "LCD_Driver.h"
#include "LCD_Driver.c"
#define F_CPU 1000000
#include "stdio.h" // sprintf
#include "stdlib.h" //atof

ISR(USART0_RX_vect);

// aliohjelmien esittelyt
void Find_speed(void);
void USART_Init(unsigned int baudrate);
void Button_init(void);
void Get_joystick(void);

// char-muuttujien alustus
char RMC_string[85], rx;
char output_to_LCD[6];
unsigned char RMC_counter = 0, RMC_place = 0;
unsigned char RMC_ok = 0;
unsigned char joystick;
unsigned char mode = 1;

// muut muuttujat
float speed = 0.0;
int max_speed = 0;
int integer_speed = 0;

#define PINB_MASK ((1<<PINB4)|(1<<PINB6)|(1<<PINB7))


int main(void)

{
CLKPR = (1<<CLKPCE);    // kellon jakaja käyttöön
                        // asetetaan jakaja = 8, sisäinen kide 8Mhz / 8 = 1Mhz
CLKPR = (1<<CLKPS1) | (1<<CLKPS0);

USART_Init(12);
LCD_Init();
LCD_CONTRAST_LEVEL(0x0E);
Button_init();

sei();

LCD_puts("READY");

while(1)
{

Get_joystick();

if (RMC_ok == 1) // jos uusi RMC-lause vastaanotettu
{

            Find_speed();           // etsitään nopeus RMC-lauseesta
            if (speed != -1)        // mikäli nopeustieto löytyi
            {
                        speed = speed * 1.852;  // muunnetaan solmut --> km/h

                        speed = speed + 0.5;    // suoritetaan pyöristys tasalukuun
                                                // lisäämällä 0,5

                        integer_speed = speed;  // ja tämän jälkeen jätetään
                                                // desimaalit huomioimatta


                        if (integer_speed > max_speed) // otetaan maksiminopeus
```

```
                                max_speed = integer_speed; // talteen max_speed
                                                           // muuttujaan


            }

if (mode == 1) // "normaalimoodi", tulostetaan nopeustieto näytölle

            {

            if (integer_speed >= 0 && integer_speed < 400)  // tarkistetaan että on
                                                            // järkevä nopeus
                        {
                        sprintf(output_to_LCD, "%3iKPH", integer_speed);
                        // näyttöpuskurin päivitys
                        LCD_puts(output_to_LCD);
                        // näyttöpuskuri näytölle

                        }

                        else                    // satelliitteja ei vielä lukittu
                        LCD_puts("NO FIX");     // tai signaalissa katkos
            }

if (mode == 2) // maksiminopeus tulostetaan näytölle

            {
            sprintf(output_to_LCD, "%3iMAX", max_speed);
            LCD_puts(output_to_LCD);
            }

if (mode == 3) // maksiminopeus nollataan

            {
            LCD_puts("RESET");
            max_speed = 0;
            }
}
}
}

ISR(USART0_RX_vect)     // sarjaliikennekeskeytys
{

rx = UDR;               // vastaanotettu merkki (UDR) muuttujaan rx

if (rx == 'R')          // mahdollinen RMC-lauseen alku
            RMC_counter=1;

if (rx == 'M')
            RMC_counter++;

if (rx == 'C')
            RMC_counter++;

if (RMC_counter == 3)   // RMC-lauseen alku on tullut kokonaan
{
            if (rx == '\r' || rx == '\n' || RMC_place > 84) // jos tullut rivinalku
                                                            // tai rivinvaihto
            {
                                // niin RMC-lause on tullut loppuun
                        RMC_ok = 1; // lippu merkiksi että RMC-lause vastaanotettu
                        RMC_place = 0; // nollataan käytetyt laskurit
                        RMC_counter=0;
            }

            else        // lause on vielä kesken ja sitä otetaan talteen
            {
```

```
                        RMC_ok = 0; // RMC ei vielä tullut loppumerkkiin asti
                        RMC_string[RMC_place] = rx;
                        // tallennetaan vastaanotettu merkki RMC_string
                        // merkkijonoon

                        RMC_place++;// kasvatetaan muuttujaa jotta seuraava merkki
                                    // menee seuraavaan alkioon

            }
}
}

void Find_speed(void) // hakee nopeustiedon RMC-lauseesta muuttujaan
{
unsigned char X, Y = 0, tmp, comma = 0; // apumuuttujien alustus
char tmp_string[7] = {0,0,0,0,0,0,0}; // merkkijonotaulukko johon nopeus luetaan

for (X=0; X<86; X++)    // pyörii silmukassa maksimissaan RMC_string-taulukon
                        // loppuun
            {           // mahdollistaa myös muiden muuttujien etsimisen lisäämisen
                        // jälkeenpäin

            tmp = RMC_string[X];    // merkki taulukosta tmp-muuttujaan
            if (tmp == ',')
                        comma++;    // pilkkulaskuri

            if (comma == 7)         // kun on löytynyt 7 pilkkua
            break;                  // hypätään pois for-silmukasta
            }
            if (RMC_string[X+1] == ',') // onko seuraavakin merkki pilkku
            speed = -1;             // nopeustietoa ei ollut

            else
            {
            do          // otetaan merkit talteen tmp_string-merkkijonotaulukkoon
                        // kunnes tulee pilkku
            {
            X++;
            tmp_string[Y] = RMC_string[X];
            Y++;

            if ( X==85 || Y>7 )
            {
            X=0;
            Y=0;
            }

            }
            while ( RMC_string[X] != ',');

            }

            speed = atof(tmp_string); // merkkijonotaulukko -> float muuttuja

            RMC_ok = 0; // nollataan RMC_ok jotta samaa nopeustietoa ei käsitellä
                        // uudestaan pääohjelmassa


}



void Button_init(void)
{
            DDRB = 0x2F;
            PORTB |= PINB_MASK;
}
```

```
void Get_joystick(void)
{
            joystick = (~PINB) & PINB_MASK;

            if (joystick & (1<<6))
                        mode = 2;   // suunta ylös, maksiminopeus

            else if (joystick & (1<<7))
                        mode = 3;   // suunta alas, maksiminopeus nollaus

            else mode = 1;          // muuten näytetään nopeutta

}


void USART_Init(unsigned int baudrate)
{
            // asetetaan bittinopeus
            UBRRH = (unsigned char)(baudrate>>8);
            UBRRL = (unsigned char)baudrate;

            // sallitaan vastaanotto ja vastaanottokeskeytys
            UCSRB = (1<<RXEN)|(0<<TXEN)|(1<<RXCIE)|(0<<UDRIE);

            // asynkroninen moodi 8N1
            UCSRC = (0<<UMSEL)|(0<<UPM0)|(1<<USBS)|(3<<UCSZ0)|(0<<UCPOL);
}
```