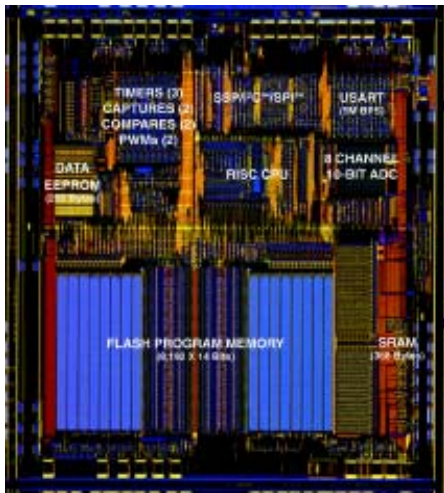


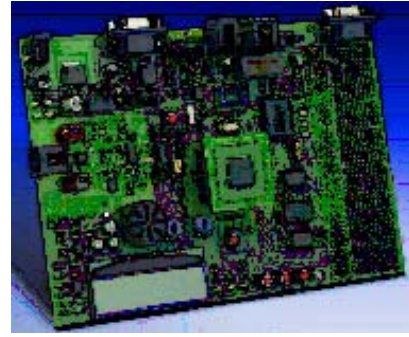
dsPIC kurssi Probyte Pekka Ritamäki



dsPIC kurssi Probyte Pekka Ritamäki	1
1 dsPIC kurssin esittely	3
1. Miksi dsPIC opetetaan vain opetetaan vain C-kielellä, kun assembler on parempi?	4
4 Tiivistetty dsPIC esittely	4
5 Mikä DSP on?	6
6 dsPIC-mallit.....	9
7 Ohjelmointiympäristö.....	10
8 Ohjelmointilaite.....	11
9 C-kääntäjä.....	12
10 Mikä on ohjelma?.....	13
11 Harward-arkkitehtuuri	14
12 Ensimmäinen dsPIC esimerkki	14
13 Ensimmäinen dsPIC-ohjelma	15
14 Linkkeri ja lokaattori	18
15 Hex-tiedostomuunnin	18
16 Pääfunktio.....	19
17 Muuttujat	19
18 Operaattorit.....	21
19 Aritmetiikka.....	22
20 Staattiset muuttujat	23
21 Muuttujien alustus	23
22 Pääohjelma, aliohjelma, moduuli ja funktio	24
23 Ohjausrakenteet	25
24 Ehto-lause	25
25 Valinta-lause.....	25
26 Odota kunnes –lauseke.....	26
27 while (lauseke)	26
28 tee –lauseke	26
29 Vertailuoperaattorit	27
30 Bittioperaattorit	27
31 Esimerkki 1.....	28
4 Merkkijonot	28
32 Osoittimet	30
33 IO-rutiinit.....	31
34 Muotoiltu tulostus <code>printf()</code> -funktioilla	33
35 Input-lauseke	35
36 C-kielen vakiomerkit.....	35
37 dsPIC33F testiohjelma	37
38 Microchip dsPIC demokortit.....	37



40



Lähdeluettelo

1. dsPIC kurssin esittely

Kurssin tarkoituksena on esitellä Microchipin dsPIC perhe, dsp-ohjelmoinnin perusperiaate, dspPIC ohjelmointityökalut, Microchipin demokortit ominaisuudet, C-kielisen ohjelman käyttö MPLAB7 ympäristössä ja tehdä ensimmäinen toimiva ohjelma. C-kielistä dsPIC ohjelmointia opetetaan niin paljon, että henkilöt, jotka eivät ole koskaan sitä opiskelleet ymmärtävät lukea C-kielisiä ohjelmia.

Kurssilaiselta edellytetään sulautettujen järjestelmien perustiedot ja mielellään Microchip PIC prosessorien perustuntemusta. Kirjoitus ei sisällä yksityiskohtaisia ohjeita jokaiseen dsp-työvaiheeseen. Suosittelemme liitteessä olevia kirjoja tietojen syventämiseen. Lopussa esitellään dsPIC demo levyjä ja FIR-suodinohjelma.

2. Miksi dsPIC opetetaan vain opetetaan vain C-kielellä, kun assembler on parempi?

Miksi ei opeteta dsPIC assemblerilla? Kaikki mikroprosessorit perustuvat konekohtaisiin käskyihin, jotka on koodattu binääriluvuiksi. Näille luvuille on annettu lyhenn nimi, assembler-nimi, kuten MOV.

Ihmiselle assemblerin käyttäminen on erittäin epätaloudellista. Ne jotka eivät osaa käyttää C-kieltä, sanovat assemblerin tekevän tehokkaampaa ja nopeampaa koodia. Minä sanon, että C-kielellä tehdään 100 kertaa nopeammin, 10 kertaa tehokkaampia sovelluksia kuin assemblerilla.

CD:llä on myös Microchip dsPIC kehityskurssi assemblerilla
dsPIC_DevTools_101304.pdf.

Esittäjä on Darrel Johansen Development Systems Manager of Documentation, eMedia and Networks



Lisätietoja

<http://techtrain.microchip.com/webseminars/ArchivedDetail.aspx?Active=63>

[Windows media player \(ääni ja esitys \)](#)

Tässä esityksessä on MPLAB/dsPIC debuggaustekniikan

4 Tiivistetty dsPIC esittely

- dsPIC on Arizona Microchip PIC-arkkitehtuuriin perustuva prosessoriperhe johon on lisätty dsp vaatimia laitteita ja käskyjä
- PIC = Periferial Interface Controller eli apuprosessori sopii isomman prosessorin apulaitteeksi. Tämä oli totta 1980-luvulla, nyt nämä laitteet toimivat täysin itsenäisesti.
- dsPIC etuja muihin prosessoreihin verrattuna:

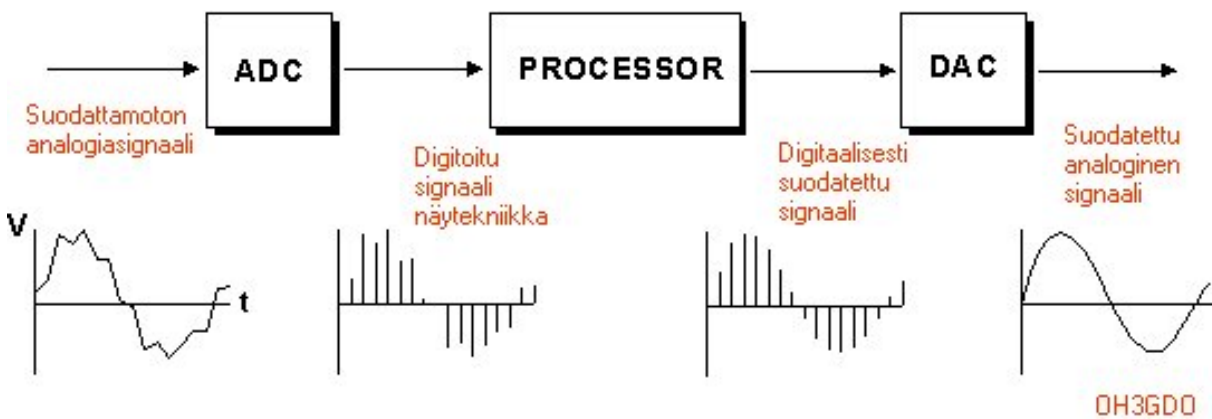
- dsPIC prosessorissa on pitkät 40-bitin dsp-akut, jota tarvitaan DSP-käskyissä
- Sama prosessori toimii yhtä aikaa yleisprosessorina ja DSP-prosessorina.
- Prosessorissa on tarpeeksi RAM ja Flash-muistia isojenkin laitteiden tarpeeseen.
- dsPIC30Fxxx ja dsPIC33Fxxx tuoteperheissä on lukusia eri malleja
- dsPICit ovat edullisia hankkia, alkaen noin 3 euroa/kpl
- Pieni virrankulutus
- RISC-tyyppinen nopea koodin käsittely yhdessä konejaksossa
- Laitteistoläheinen ohjelmointi on helppoa
- Erityisesti nopeat, paljon nopeaa laskentaa vaativat tehtävät sopivat dsPIC:lle
- dsPIC ja tavalliset PIC:t toimivat lähes identtisesti. Ohjelmoijan on helppo siirtyä prosessorista toiseen
- Rinnakkainen, edullisempi, PIC24Fxxx sarja sopii samaan piirilevyyn ja ohjelmaan sovelluksissa, joi ei tarvita DSP-käskyjä
- IO-nastat ovat erittäin käyttökelpoisia ilman erikoisia liitäntälaitteita. Ne pystyvät suoraan ohjaamaan ledejä ja 7-segmenttinäytöjä. Toisaalta tulot ovat erittäin suuri impedanssisia.
- dsPICeissä on laaja, sisäinen lisälaitte joukko, AD-muuntimet, laskurit, PWM, Capture/Compare, UART, SPI, I2C, rinnakkaisportti, Alijännitevalvonta, useita virransäästöominaisuuksia ja kello-oskillaattoreita, vahtikoiria, sarjamuotoinen ohjelmointi vain normaalilla käyttöjännitteellä, Bootloader valmius, hardware matematiikka ym.

Haittoina on:

- Koodin kokorajoitus, mallista riippuen sopii vain pieniin sovelluksiin, noin 8k–128ktavua
- Pieni RAM-muistin koko, 3k –32ktavua
- dsPIC ohjelmointi vaatii uuden oppimista ja uusia työkaluja
- dsPICeissä ei ole kiintolevyä, ei monitoria, ei näppäimistöä. Tämä on aluksi vaikeaa Windows ympäristöön tottuneelle ohjelmoijalle

- Kääntäjät, ohjelmointilaitteet ja emulaattorit eivät ole ilmaisia, mutta ei uusi kannettava PC:kään ole ilmainen ja silti niitä käytetään. Silti pelkällä PC:llä ei voi tehdä omia, itsenäisiä dsp-laitteita. Kaikki ammattityö vaatii oppimista eihän se muuten olisikaan ammattilaisten työtä. Talonmies voisi hoitaa nämä dsp-asiat pihan siivouksen välillä.

5 Mikä DSP on?



3. Analoginen signaalinkäsittely

Analoginen signaalinkäsittely on ollut tunnettua noin sata vuotta. Analogisia signaaleja käsitellään radiossa, audiovahvistimissa, puhelinverkoissa, teollisuuden mittauslaitteissa, lääketieteessä ja antureissa. Perinteisesti on käytetty vahvistimia, suodattimia, analogiakytкимиä jne. Näiden laitteiden suunnittelu on analogiaelektroniikkaa. Tuloksena on piirilevy, johon asennetaan tarpeelliset komponentit. Kun tarvitaan hieman toisenlainen laite rakennetaan taas uusi prototyyppi.

4. Digitaalisessa signaalinkäsittelyssä

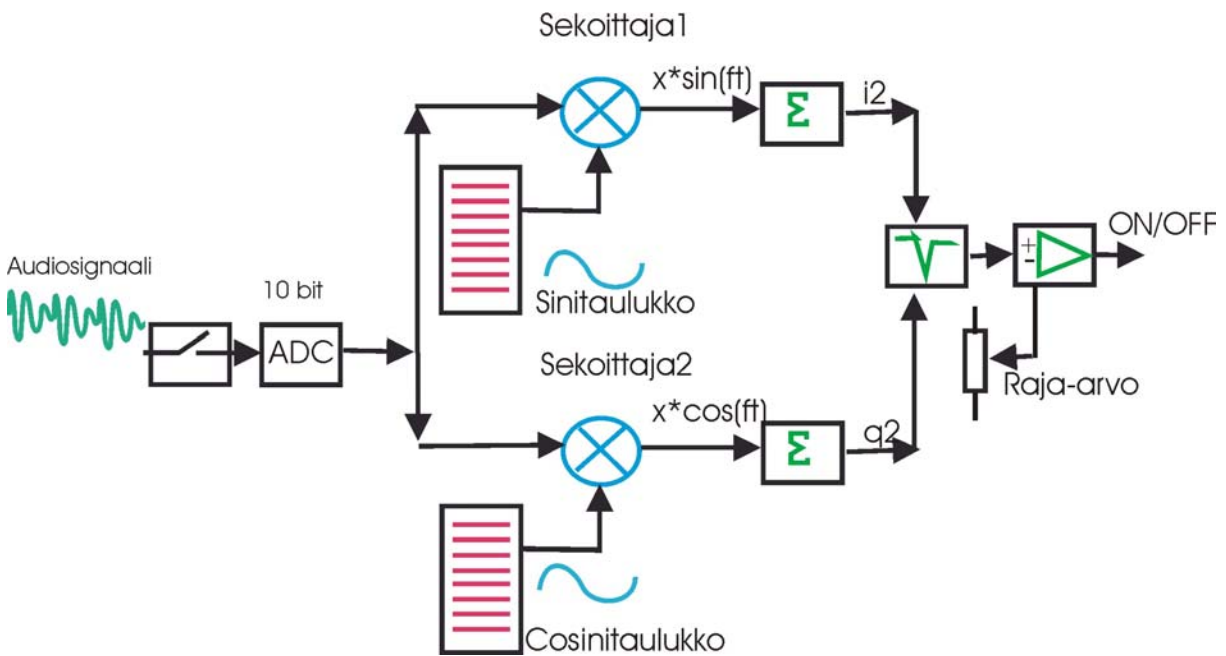
Digitaalisessa signaalinkäsittelyssä signaalin muokkaus tehdään ohjelmallisesti. Kun tarvitaan uusia ominaisuuksia muutokset tehdään ohjelmaan, mutta itse laite saattaa pysyä samana.

Signaali voidaan luoda digitaalisesti tai mitattavaa signaalia voidaan analysoida, muuttaa tai siitä voidaan ottaa näytteitä. Melkein aina tarvitaan analogisen signaalin muokkausta digitaaliseksi tai päinvastoin. Se tehdään analogia digitaalimuuntimella tai digitaali-analogia muuntimella. Tyypillisiä sovelluksia ovat Digitaalinen kuvankäsittely esimerkiksi roboteilla tapahtuva elintarviketuotanto. Tämän alueen tunnen erittäin yksityiskohtaisesti.



Kuvassa1 eräs elintarviketuotantolinja, jossa ylimmäisenä näkyy kameran valot ja viisi nopeaa robottia, vasemmalla on yksi suurempi robotti. Kuva vuodelta 2006-10-23

Puheen tai musiikin tuottaminen prosessorilla on toisen suuntainen dsp-sovellus. Näitä on olen tehnyt moniin suomalaisiin radioamatööriistöisiin.



Kuva2. Radioamatöörisignaalin tunnistaminen DTF-menetelmällä

Signaalin tunnistaminen onnistuu tavallisellakin prosessorilla, mutta jos käytävissä on dsp-prosessorin nopea 16-bittinen kertolaskuyksikkö ja pitkät 40-bittiset summausrekisterit, ohjelmointi onnistuu tavalliseltakin ohjelmoijalta.

Esimerkkejä dsp suodattimien laskemisesta on dsPIC CD:n ScopeFIR suodatin ohjelma, jonka avulla lasketaan suodatin komponentit FIR-suodattimeen,

Suodattimen laskentaan löytyy tietoja jokaisesta dsp-kirjasta. CD:llä on eräs DSP kirja pdf-muodossa. Siinä on 33 kappaletta dsp-tekniikan yksityiskohtia PDF-tiedostoina.

Miksi tarvitaan erillistä dsp-prosessoria?



6 dsPIC-prosesorit

Freescale, Analog Devices, Texas Instruments ovat olleet perinteisiä dsp-prosesorien valmistajia. Ne eivät kuitenkaan sovi yleisprosessoriksi, jossa on monia liitäntöjä ja paljon erilaista muistia, RAM, FLASH ja EEPROM samassa piirissä. Microchip on uudempi tulokas, mutta ylivoimainen monipuolisuudessa.

Microchipin dsp-prosessorit ovat aina täysin nastayhteensopivia saman perheen kesken. dsPIC30F ja dsPIC33-prosessori ovat mekaanisesti täysin yhteensopivia toistensa kanssa. Jopa erinastaiset piirit sopivat kauniisti samalle kohdalle piirilevyssä.

PIC16F ja PIC18F kanssa pitää tehdä aivan pieniä muutoksia nastajärjestykseen tai käyttää välikantaa.

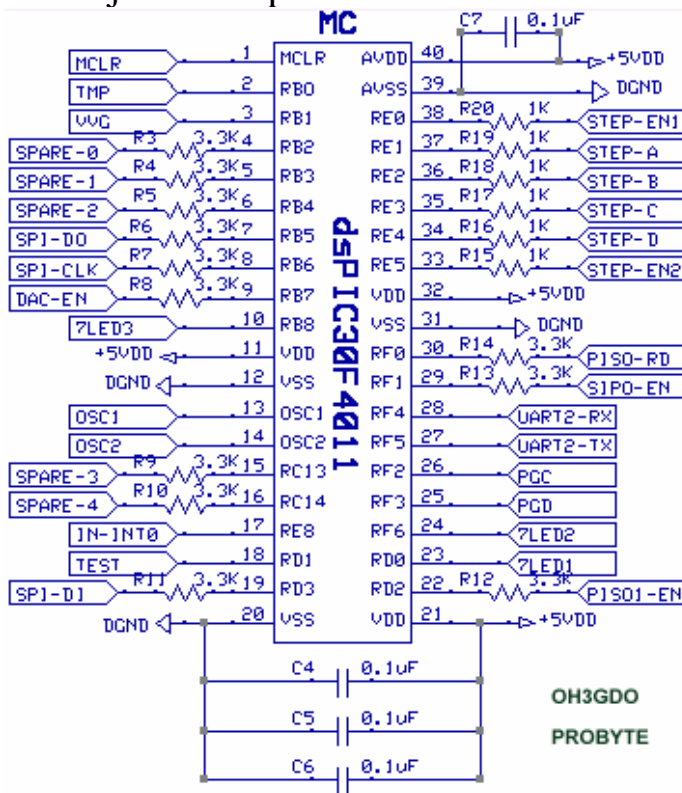
dsPIC30F on alkuperäinen malli ja dsPIC33F on uusin, monipuolisempi dsPIC-sarja. PIC24F on mekaanisesti yhteensopiva dsPIC:n kanssa, mutta

ilman dsp-ominaisuuksia oleva yleisprosessori dsPIC30F2011, dsPIC30F2012, dsPIC30F 2013, dsPIC30F3012, ja dsPIC30F3013 piirejä tehdään 18- 28-, 44-nastaisissa koteloidissa.

Kotelovaihtoehtoja ovat DIP18, DIP28, SOIC18, SOIC28, QFN28 ja QFN44.

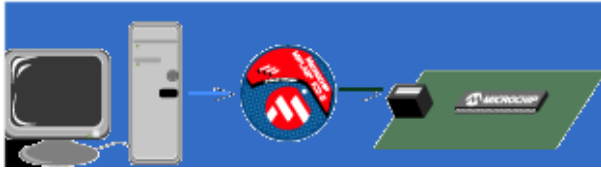
Nämä dsPIC prosessorit toimivat 30MIPSillä ja niissä on kaksi tärkeää 40 bittistä dsp-akkua , 17*17 bitin kertolaskuyksikkö, 16-bitin dataväylä, 21 keskeytyslähdettä, 21 kbyteä Flash-muistia, 2 kilotavua RAM-muistia. 1 kilotavua EEPROM muistia, 10 kanavaa 12-bittistä analogiamuunninta, jotka pystyvät ottaa 200 kilonäytettä sekunnissa.

Microchip jakaa dsPICit kahteen eri ryhmään: anturi dsPIC ja moottori DSPit. Periaatteessa molemmat ovat samanlaisia, mutta niissä on erilaisia PWM ja muita apulaitteita.



Ohessa dsPIC4011 piirikaavio askelmoottori-sovelluksiin

7 Ohjelmointiympäristö



Microchipin oma, dsp-prosessoreille kehitetty kääntäjä C30 (pic30-gcc.exe) perustuu gnu-C-kääntäjään. Tämä kääntäjä on puhdas komentorivikääntäjä ilman editoreita ja vianhakuohjelmia. Kun se integroidaan MPLAB kehitysympäristöön, jossa on editori, simulaattori ja vianhakuympäristö kaikki toimii hienosti.

Kääntäjä käyttää Microchip MPLAB-ympäristöä. Muista ladata viimeisin versio (MPLAB 7.xx) Microchipin sivuilta (www.microchip.com).

8 Ohjelmointilaite

dsp-prosessoreita voi ohjelmoida monilla laitteilla.



- Microchip ICD2(pyöreä laite) on samalla myös emulaattori, jonka avulla voi tehdä keskeytyksiä ohjemaan reaaliaikaisesti
- PM3, Promate ovat kalliimpia tuotanto-ohjelmointilaitteita
- Lisätietoja löytyy http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2035
- PICKIT1 on vanhentunut
- PICSTART ei tue lainkaan dspPIC:jä
- PICKIT2 ei vielä tue dsPIC prosessoreita

MicroEngineering Labsin ohjelmointilaitteet ovat monipuolisia ja tukevat dsPIC:iä. Lisätietoa <http://www.melabs.com/>

Probyte Parpic2 ohjelmointilaitteet <http://www.probyte.fi> tukevat dsPIC piirejä
CCS ohjelmointilaitteet eivät vielä tue dsPIC:iä

9 C-kääntäjä



Microchip C30 ohjelmistotyökalupaketti sisältää 16-bittisen C-kääntäjän on pic30-gcc.exe, assemblerin, pic30-as.exe, linkkerin pic30-ld.exe, ja kirjasto/talletusohjelman pic30-ar.exe.

Käännetty tiedosto käännetään binääriohjelmalla -> Intel Hex tiedosto muuntimella (pic30-bin2hex.exe). Komentorivisimulaattori on sim30.exe.

Kääntäjä tukee seuraavia dsPIC piirejä

dsPIC30 sarja:

30F1010	30F2010	30F3010	30F4011	30F5011	30F6010
	30F2011	30F3011	30F4012	30F5013	30F6010A
	30F2012	30F3012	30F4013	30F5015	30F6011
	30F2020	30F3013		30F5016	30F6011A
	30F2021	30F3014			30F6012
	30F2022				30F6012A
	30F2023				30F6013
					30F6013A
					30F6014
					30F6014A
					30F6015

dsPIC33 sarja

33FJ64GP206	33FJ128GP206	33FJ256GP506
33FJ64GP306	33FJ128GP306	33FJ256GP510
33FJ64GP310	33FJ128GP310	33FJ256GP710
33FJ64GP706	33FJ128GP706	
33FJ64GP708	33FJ128GP708	
33FJ64GP710	33FJ128GP710	
33FJ64MC506	33FJ128MC506	33FJ256MC510
33FJ64MC508	33FJ128MC510	33FJ256MC710
33FJ64MC510	33FJ128MC706	
33FJ64MC706	33FJ128MC708	
33FJ64MC710	33FJ128MC710	



Muilla valmistajilla on myös tulossa C-kääntäjät dsPICeille, Esim. CCS tuote on melkein valmis.

Hitech on jo valmis kääntäjä



Mutta se on kovin puutteellinen ja kallis > 2400 USD.

10 Mikä on ohjelma?

Ohjelmassa on muuttujille varattua muistia, jota käsitellään ohjelman sisäisillä lausekkeilla. Muisti voi olla vakioita tai muuttujia, joihin voi kirjoittaa uusia arvoja.

Kuten kaikki aine, ohjelma voi jakaantua samanlaisiin pienempiin palasiin, joissa on taas nämä samat ominaisuudet. Lopuksi tulevat kuitenkin atomit eli prosessorikohtaiset konekielikäskyt, joissa on taas vain muutama, dsPIC:llä on noin neljäkymmentä assemblerkäskyä, ja kaikkea ohjataan lopuksi kuitenkin assemblerin GOTO käskyillä.

C-kieli piilottaa nämä prosessorikohtaiset käskyt ja antaa ohjelmoijalle yhtenäisen menetelmän jolla prosessoria ohjataan. Tämä on aivan samaa kuin auton ajaminen, siinäkin kuljettaja ei aivan tarkkaan tiedä mitä tapahtuu, kun kuljettaja painaa kaasua. Kun kuljettaja on kerran oppinut ajamaan autoa kaikki autot tottelevat kaasupoljinta samalla periaatteella. Samalla tavalla C-kieli antaa yhtenäisen rajapinnan kaikille prosessoreille niin Ferrareille kuin Ladallekin.

C-kielen tehtävänä on helpottaa ohjelmoijan työtä. Ohjelmoija kirjoittaa hieman ihmisen kieltä muistuttavalla kielellä, jos englantia voidaan kutsua ihmisen kieleksi, lausekkeita. Näitä lauseita C-kielen kääntäjän on kuitenkin suhteellisen helppo tulkita konekoodiksi. Tällä tavalla C-kieli auttaa ihmistä ymmärtämään hieman suurempia kokonaisuuksia kerralla kuin yksittäiset konekielikäskyt.

C-kielestä on kehitetty myös objektorientoituneita C-kääntäjäsovelluksia kuten Imprisen (Borland) C++-Builder, jossa käsitellään vielä laajempia kokonaisuuksia kuin yksittäisiä lauseita. Kieli purkaa nämä objektit C-koodiksi, joka sitten käännetään kuten tavallinen C-kieli ko. konekoodiksi. Osa koodista jää piiloon kääntäjän kirjastoistorutiineihin, mutta tällä tavalla saadaan taas selvempää koodia, kun käyttäjälle näytetään vain ne rutiinit, joihin käyttäjä joutuu tekemään omia muutoksia. Objektorientoitunut PIC C-kääntäjä olisi hieno asia, mutta sen tekeminen vaatii aika paljon työtä, jotta se toimisi yhtä hienosti kuin nykyiset Windows-pohjaiset kääntäjät.

11 Harward-arkkitehtuuri

PIC:ssä on ns. Harward-arkkitehtuuri, jossa koodi ja RAM-muisti ovat toisistaan erikseen. Koodia ei yleensä voi muuttaa muuta kuin erikoislaitteilla (ohjelman polttolaite). PC:ssä kaikki on toisin, siinä käytetään von Neumann arkkitehtuuria, jossa koodi ja RAM-muisti ovat peräkkäin samassa muistiavaruudessa. Monet saattavat ihmetellä miten PIC:n pienellä koodi- ja RAM-määrällä voi lainkaan tehdä ohjelmia, mutta on muistettava, että PIC:ssä ei ole lainkaan näppäimistöä, video-liitäntää, kiintolevyä, hiirtä jne. Oman kokemukseni mukaan C:llä saa aivan yhtä tiivistä koodia kuin assemblerilla, mutta noin 100 kertaa nopeammin. Toisaalta aloitteleva C-ohjelmoija voi täyttää PIC:n koodimuistin parilla sopimattomalla C-rutiinilla.

Samoin C:llä voi tehdä nopeaa tai hidasta koodia. On oikeastaan välttämätöntä, että ohjelmoija tuntee PIC:n rekisterirakenteen ja osaa ohjelmoida ainakin jonkun verran suoraan assembleritasolla.

12 Ensimmäinen dsPIC esimerkki



Kuva1. Ohjelmanluomisen karkea periaate

13 Ensimmäinen dsPIC-ohjelma

Katso CD:n sovellusesimerkit dsPIC_hello_world -hakemistoon ja kopioi sieltä ohjelma MPLAB ympäristöön ja omaan testihakemistoosi .

Ohjelma on tarkoitettu toimivaksi Microchipin demo piirilevyn DV300016, Microchipin ICD2 ohjelmoi n t i l a i t t e e n j a dsPIC30F6012 mikroprosessorin kanssa.

Tee tästä tiedostosta uusi projekti. Hello

```

/*****
*****
* Ensimmäinen ohjelma ICD2/DV300016demolevy/
dsPIC30F6012
* Käytetään Timer1 vilkuttamaan LEDiä RD.4 kun
Kytkiintä S1 ei paineta
* ja vilkuttamaan Lediä RD.5 kun S1 painetaan ,
virheissä RD.7 led palaa
* Pekka Ritamäki Probyte
* Date: 22.10.2006
* File hello.c
*****/
  
```

```

#include "p30F6012.h"
//----- valitaan prosessorityyppi -----
--Configuraatio bitit
_FOSC(CSW_FSCM_OFF & XT_PLL4);
_FWDT(WDT_OFF);
_FBORPOR(PBOR_OFF & BORV_27 & PWRT_16 & MCLR_EN);
_FGS(CODE_PROT_OFF);

//-----
//Kiiteen nopeus määrittely
#define FCY 4000000 //käskeyjen suoritus (Osc x
PLL / 4)
//=====
//Pääohjelmaa alkaa
// alustetaan ajastin timer1 ja ledit

int main(void)
{
int x=99; // testimuuttaja debuggausta varten
int __attribute__((address(0x100))) dsPIC = 1;
// muuttuja liitetään muistiinpaikkaan
LATD = 0xff0f; //alusta ledit OFF tilaan
TRISD = 0xff0f; //Aseta Led-nastat annoiksi
LATDbits.LATD4 = 1; //Laitte RD.4 päälle
T1CON = 0; //Käynnistä timer1
TMR1 = 0; //Aseta Timer1 nolliaksi
PR1 = FCY/256/5; // Aseta periodi rekisteri
1/5:aan
T1CON = 0x8030; //Kytke timeri 1: 256 esijakajaan

while(1) //i kainen luuppi

{
printf("\r dsPIC Hello World, laskuri %d", x++);
printf(__TIME__); // tulosta aika
while(!IFS0bits.T1IF){} //Odota kunnes timer1:n
aika on kulunut
IFS0bits.T1IF = 0; // Nollaa timer1:n lippu
if(PORTCbits.RC13) // Onko S1 painettu (=0)?
{
LATDbits.LATD4 ^= 1; //Vaihda ledin RD.4 tilaa
jos S1 on painettu
LATDbits.LATD5 = 0; // Sammuta LED RD.5
}
}
}

```

```

}
else
{
LATDbits.LATD4 = 0; //LED RD4 pois
LATDbits.LATD5 ^= 1; //vaihdä LED RD5 tilaan kun
S1 on painettu
}
}
} //Main loppu
//=====
//Virheiden sieppaus
//-----
//Oskillosaattorin virheen käsittely
void _ISR_OscillatorFail(void)
{
LATDbits.LATD7 = 0; // LED RD.7 päälle
while(1); //jää tähän
}
//-----
//Osoite virheen käsittely
void _ISR_AddressError(void)
{
LATDbits.LATD7 = 0; // LED RD.7 päälle
while(1); //jää tähän
}
//-----
//Pino virheen käsittely
void _ISR_StackError(void)
{
LATDbits.LATD7 = 0; // LED RD.7 päälle
while(1); //jää tähän
}
//-----
//Matematiikka virhe (Aritmetiikka) virheen
käsittely
void _ISR_MathError(void)
{
LATDbits.LATD7 = 0; // LED RD.7 päälle
while(1); //jää tähän
}
//=====ohjelman loppu=====

```

Yllä on tyypillinen ensimmäinen ”Hello World”-ohjelma. Ohjelma kirjoitetaan MPLAB7 editorilla. Muukin ohjelma käy, jos jossa ei ole ylimääräisiä muotoiluja kuten Microsoftin WORD:ssä. Kääntäminen

tapahtuu painamalla Compile käännöskuvaketta.



Tuloksena on esim. hel | o. hex tiedosto, PIC ohjelman poltto-ohjelmaa varten. Jos MPLAB integroitu joku ohjelmointilaite, ohjelmointi voidaan tehdä samassa ohjelmassa.

14 Linkkeri ja lokaattori

Jotkut C-kääntäjät kuten CCS:n PCW tekee kaikki yhdellä kertaa. Kääntää, linkkaa ja lokatoi. Tosin kahta jälkimmäistä ei tehdä koska tämä kääntäjä on niin nopea, että käännöstä ei tarvitse tehdä useissa erivaiheissa ja sitten yhdistää linkkerillä. Linkkeri yhdistää eri käännöksiä tuottamat objektikoodit yhdeksi objektiksi. Lokaattori liittää objektikoodin prosessorin oikeisiin osoitteisiin ja tekee lopullisen HEX koodin ohjelmointilaitetta varten.

15 Hex-tiedostomuunnin

pic30-bin2hex toimii yksinkertaisesti laittamalla lokatoimiin tuloksena syntyneen tiedoston ohjelman perään. Jos hex- tiedosto halutaan järjestää osoitteen mukaan määrittele **-a** optio
pic30-bin2hex object_file -a
-v optiolla näytetään kaikki optioparametrit
pic30-bin2hex object_file -va

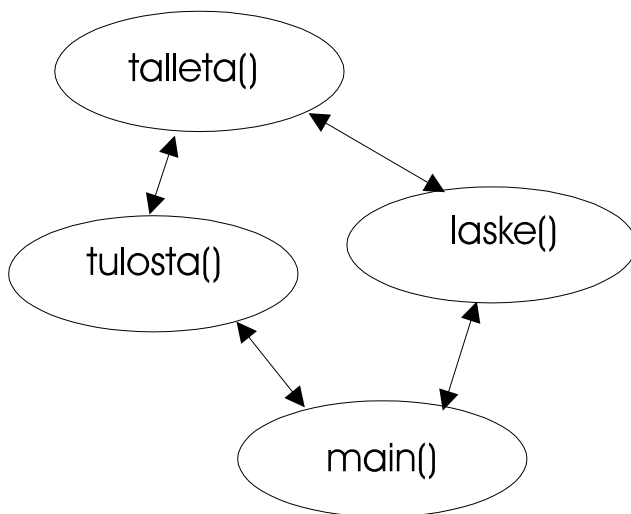
Kääntämisessä voidaan käyttää myös komentojonoja tai ohjelmointiohjelma voidaan käynnistää kääntäjän painikkeesta.

```
/*=====
** Linker Script for 30f6012
   Prosessorille
   file :Linkerscript.gld
   Nape : Pekka Ritamäki /Probyte
   date: 20.10.2005
```

Käytä MPLAB7** projektitiedossa linkkauksessa
*/

```
OUTPUT_FORMAT("coff-pic30")  
OUTPUT_ARCH("pic30")  
EXTERN(__resetPRI)  
EXTERN(__resetALT)  
ENTRY(__reset)
```

16 Pääfunktio



17 Muuttujat

char yksi tavu, useimmiten 8-bittinen, int on 16 bittinen luku ja float desimaalipisteellä varustettu liukuvan pilkun luku .

byte , *int* , *unsigned int*, *char* on kokonaisluku on C30:ssä 8-bittinen luku 0..255

long int, *long* tai *int16* on kaksi kertaa niin pitkä kuin *int* eli dsPIC C30:ssä 32-bittinen luku

float on 32-bittinen liukuvan pilkun luku

double on 64-bittinen eli kaksi kertaa suurempi kuin *float*

Muuttujat voidaan esitellä funktioiden ulkopuolella, jolloin kaikki funktiot näkevät ne tai funktioiden sisällä ennen koodin alkamista, jolloin vain tämä funktio näkee ko. muuttujan.

```
char luku1; // yleinen eli globaali muuttuja  
void main ( void)  
{  
char luku2; // paikallinen eli lokaali muuttuja  
luku2=1;  
luku1=luku2;  
}
```

Uusia muuttuja tyyppejä voidaan luoda entisistä #typedef- kääntäjän ohjeella:

```
#typedef byte char
```

Nyt voidaan käyttää uutta muuttujaa kuten aikaisemmin *char* muuttujaa

```
byte tavu, luku;
```

Monimutkaisempia tyypimäärittelyjä voidaan luoda *struct-rakenteella*:

```
struct {  
byte luku;  
long pitkä luku;  
int32 pisin luku;  
char merkit[20];  
} nimi tyyppi;
```

esim.

```
nimi tyyppi henki lo[10];  
varaa muistia jos on 10 kpl nimi tyyppi nimi stä  
muuttujaa.
```

18 Operaattorit

Operaattorit ovat C-kielen varattuja merkkejä tai merkkiyhdistelmiä, joita käytetään kielessä sovitulla tavalla ja niitä ei saa käyttää muuttujissa merkkeinä. Varatut sanat helpottavat kääntäjän toimintaa. Ilman varattuja merkkiyhdistelmiä kääntäjän olisi hankala erottaa kielioppia ja ohjelman muuttujia toisistaan, varsinkin koska C-kielisiä lauseita voi olla samalla rivillä useita tai sama lause jatkuu monille riveille.

Alla on lueteltu kaikki C-kielen varatut merkit ja niiden yhdistelmät:

() [] ->	.	,	/	*	%			
!	~	++	--	+	-	&		
sizeof	<<	>>	<	<=	>	>=		
==	!=	~	/	&&	//	? :		
=	+=	--	*=	/=	%=	&=	^=	
/=	>>	= >	>=					

Operaattorien merkitys selvitellään erikseen myöhemmin.

Kaikki operaatiot suoritetaan vasemmalta oikealle paitsi sijoitusoperaatiot

(=	+=	--	*=	/=	%=	&=	^=	
/=	>>	= >	>=)					

ja vaihtoehtolauseke (? :). Luottamalla pelkästään kääntäjän oikeamieliseen järjestykseen aloittelija voi kuitenkin tehdä helposti vääriä oletuksia kääntämisjärjestyksestä ja ohjelmaan voi tulla hankalasti löydettäviä vikoja. Jos ohjelmoija on vähänkin epävarma kuinka käänös suoritetaan, kehotan käyttämään sulkuja, jotta arvailulle ei jää aihetta. Esimerkiksi mitä lauseke

```
if ( x & MASK == 0) {  
    tee();  
}
```

oikeastaan tekee? Jos se muutetaan muotoon

```
if ( (x & MASK) == 0) {  
    tee();  
}
```

kaikille on selvä mitä tehdään ensin. Sulkujen sisällä olevat laskutoimitukset tehdään ensin kuten normaali matematiikassa.

19 Aritmetiikka

C-kieli sisältää yksinkertaiset matemaattiset lauseet kuten yhteenlasku (+), vähennyslasku(-), kertolasku(*), jakolasku(/) ja kokonaislukujaon jakojäännös (%).

Esimerkiksi voidaan tehdä lausekkeet

```
a = b + c;  
c = a - b;  
c = a / b;  
b = a % c;
```

%-operaattori tarkoittaa sitä, että tulokseksi tulee kokonaislukujen a ja c jakojäännös, joka sekin on kokonaisluku. Esim.:

```
c = 10 % 3;  
c = 2
```

Vaikka yllä kuvatut esimerkit saattavat näyttää helpolta, miettikää miten tehdään seuraava lasku assemblerilla: 0.00003432342 /32343100000 .000012

Laskutoimituksissa pitää ohjelmoijan itse huolehtia erikoistilanteista kuten nollalla jakamisella (ei saa tapahtua!), ylivuodot ja erikokoisten muuttujien väliset laskutoimitukset. Nämä ovat useimmiten aloittelijoiden ongelmia, joissa epäillään kääntäjän tekemän virheen. Tosiasiassa kääntäjän tekemät virheet ovat yhtä harvinaisia kuin lottovoitto. Loogisten ohjelmavirheiden jälkeen erikokoisten muuttujien aiheuttamat virheet ovat oman kokemukseni mukaan suuri ryhmä.

Kääntäjistä riippuen seuraavan esimerkin tulos on virheellinen tai oikein

```
long int a=1000, b=100;  
char c;
```

```
c = a/b;
```

Jos funktion tulos pakotetaan (= type cast) `char`-muotoon ennen muuttujan siirtämistä erikokoiseen muuttujaan, niin tulos on useimmilla kääntäjillä oikein.

```
c = (char)(a/b);
```

Pakottaminen tarkoittaa, että koska halutaan jakolaskun tuloksen olevan kahdeksanbittinen, niin jakolaskun jälkeen `c`-muuttujaan siirretään eniten merkitsevä tavu. Ilman pakottamista, sinne olisi voinut mennä jakolaskun vähintenmerkitsevä tavu riippuen satunnaisesti prosessorin käyttämästä muistiavaruusjärjestelmästä tai jakolaskufunktion tuloksesta sisäisessä muistissa. C-kieli ei itsestään tee tarkistuksia, mutta tämä pakottaminen on yksi komento kääntäjälle tehdä tämä tarkistus.

20 Staattiset muuttujat

Staattiset muuttujat säilyttävät arvonsa kutsukerrasta toiseen aliohjelmissa. Staattisten muuttujien tila varataan yleiseltä muistialueelta. Staattiset muuttujat eivät kuitenkaan näy aliohjelmista ulos.

Esimerkki:

```
void tulosta_laskuri ( void ) {  
    static int count =0;  
    printf("%d", count++);  
}
```

21 Muuttujien alustus

Kaikki yleiset (globaalit) muuttujat voidaan alustaa, mutta jos niitä ei alusteta ne ovat nollija. Vaikka tämä on C-kieleen liittyvä määritelmä, on parasta asettaa omat muuttujat haluttuihin alkuarvoihin, jotta ei kokisi yllätyksiä, jossakin kääntäjissä tai prosessoriympäristöissä.

```
int alku =1;  
char ch = '\0';  
int taulukko [4][3] = {  
    {1, 233, 3},  
    {3, 2, 33},  
    {6, 3, 34},  
    {5, 4, 35}  
};  
tai  
int taulukko [4][3] = {
```

```
} ;      1, 233, 3, 3, 2, 33, 6, 3, 34, 5, 4, 35
```

Ilman alustusta aliohjelman muuttuja on epämääräinen. Tämä johtuu siitä, että yleensä aliohjelmien muuttujat otetaan pinosta, jolloin niihin on saattanut jäädä edellisen aliohjelman muuttujia. PIC C:ssä ei tätä ongelmaa ole, koska pinoa ei voi käyttää muuttujamuistina. Kuitenkin tämä on pidettävä mielessä, kun käyttää muita prosessoreita C-kielillä.

22 Pääohjelma, aliohjelma, moduuli ja funktio

Kaikki ohjelmat alkavat `main()`-nimisestä pääohjelmafunktiosta.

Pääohjelmasta kutsutaan aliohjelmia, jotka kutsuvat vuorostaan toisia aliohjelmia.

Kaikki aliohjelmat ovat funktioita riippumatta palauttavatko ne tietoja tai eivät palauta (void funktiot). Muissa kielissä erotetaan aliohjelmat ja funktiot toisistaan.

```
int laske ( int luku1, int luku2) {  
    return luku1+luku2;  
}
```

```
void tulosta ( int c) {  
    printf("%d", c);  
}
```

```
void main ( void ) {  
    int a=1, b=2;  
    tulosta ( laske(a, b));  
}
```

Moduuli on ohjelman osa joka voidaan kääntää erikseen objektimuotoon kääntäjän myöhempää linkkausta varten. Moduuli on yleensä `c`-päätteinen tiedosto, josta tulee kääntämisen jälkeen `.obj`-päätteinen binääritiedosto. Kun kaikki käännetyt `.obj`-tiedostot linkitetään yhteen saadaan `.exe`, `.com` tai `.hex`-päätteinen ajettava binääritiedosto. Hex-tiedosto ei tosin ole ajettava vaan ohjelmoitava tiedostomuoto.

23 Ohjausrakenteet

Ohjausrakenteilla määritellään ohjelman siirtymistä eri osiin käyttäjän (PC) tai laitteen (PIC) antamien tietojen perustella. Näitä ohjausrakenteita on erittäin vähän. Ne kannattaa opetella ainakin kerran, jotta niiden käyttö olisi koko elämän ajan sujuvaa.

24 Ehto-lause

```
if (lauseke)
    lause1;
else
    lause2;
```

Esimerkiksi:

```
if (merkki != 'A')
    printf("\r Tuli A");
else
    printf("\r Ei ole A");
```

tai pelkästään

```
if( lause)
    lause1;
```

Esimerkiksi

```
if( merkki != 'A' ) printf("\r Tuli A");
```

25 Valinta-lause

```
switch (merkki) {
    case 'a' : printf("a"); break;
    case 'A' : printf("A"); break;
    case 'b' : printf("b"); break;
    case 66 : printf("B"); break;
default:    printf(" Ei käy"); break;
}
```

Huomaa, että jos ei käytetä `break`-katkaisusanaa, tarkastetaan myös seuraava `case`-lauseke.

26 `Odota kunnes` -lauseke

```
27 while (lauseke)
   lause1;
```

Esimerkiksi vähennetään lukua `kunnes` se on nolla

```
j = 10;
while ( j != 0) {
  j = j - 1;
}
```

```
28 tee -lauseke
do {
  lause1;
} while (lauseke);
```

Esimerkiksi luetaan merkkejä näppäimistöltä, kunnes saadaan `A`-merkki.

```
do {
  scanf("%c", &ch);
} while (ch!='A');
```

`do`- ja `while`-lauseiden ero on se, että `do`-lauseke tekee sen sisällä olevat lauseet ainakin kerran, mutta `while`-lauseke tarkastaa jo ensimmäisellä kerralla ehdon.

```
Laskuri -lause
for ( ehto1; ehto2; ehto3) {
  lauseke1;
}
```

`ehto1` suoritetaan ennen lauseen ajoa, `ehto2` tarkastetaan koska `for`-lause loppuu, `ehto3` suoritetaan joka kierroksen jälkeen.

Esimerkiksi

```
for ( laskuri = 0; laskuri < 10 ; laskuri ++ ) {
```

```
printf ("\raskuri %d", laskuri );  
}
```

exit() -lauseketta, jolla palataan käyttöjärjestelmään ei voi PIC-C:ssä käyttää

return() -lauseella palautetaan aliohjelmasta vain yksi muuttuja, jos aliohjelman eteen on määritelty jokin muuttujan tyyppi, joka on eri kuin void.

29 Vertailuoperaattorit

Pienempi kuin merkki <

Suurempikuin merkki >

Pienempi tai yhtäsuuri merkki <=

Suurempi tai yhtäsuuri merkki >=

Yhtäsuuri merkki ==

Erisuuri merkki !=

Vertailuja voidaan käyttää vain merkeille, luvuille ja osoitteille, ei esimerkiksi merkkijonoille kuten muissa kielissä. C-kielessä on näille omat funktionsa (string.h)

Huomaa myös yhtäsuuri-merkki, joka on kaksi merkkiä peräkkäin (==), tämä on C-kielille erilainen kuin muissa kielissä.

Esimerkkejä

```
if ( a>b) c=1;  
if (ch == 'Z') ch=' ' ;  
if ( a=0) c=4; // vi rre !!  
if ("koe"!="testi ") c=0; // vi rre!!
```

Vertailun tuloksen tulee 0 tai yksi, jota monesti kutsutaan Boolean operaatioksi.

30 Bittiooperaattorit

AND tai JA-operaatio merkitään C-kielessä &

OR tai TAI -operaatio merkitään |

XOR tai poissulkeva-operaatio merkitään ^

SHIFT-LEFT tai siirto vasemmalle-operaatio merkitään <<

SHIFT-RIGHT tai siirto oikealle-operaatio merkitään >>

ONE COMPLEMENT tai yhden komplementti merkitään ~

31 Esimerkki 1

```
char a, b, c;
```

```
a= 0b00001111;  
b= 0b11110000;  
c= a & b; // a ja b muuttujille tehdään biteittäin JA-operaatio  
Tul os c= 0b00000000
```

Esimerkki 2

```
a= 0b00001111 ;  
b= 0b11110000;  
c= a | b ; // tai -operaatio a:lle ja b:lle  
Tul os c= 0b11111111
```

Esimerkki 3

```
a= 0b00001111M  
b= 0b11110000;  
c= a ^ b; // a:lle ja B:lle tehdään poissulkeva tai bittivertailu  
Tul os c= 0b11111111
```

Esimerkki 4

```
a= 0b00001111  
b= a <<1  
Tul os c= 0b00011110
```

Esimerkki 5

```
a= 0b00001111  
c= a >>1  
Tul os c= 0b00000111
```

4 Merkkijonot

Merkkijonot ovat C-kielen eräs vaikeimmista asioista, mutta ilman niitä ei pysty tekemään kuin yksinkertaisia ohjelmia. Merkki (char) on yleensä kahdeksanbittinen tavu, jolla voi ilmaista ASCII-merkkejä ja pieniä numeroita (-127... 0 +128). Merkkimuuttujien varaus on helppoa, ohjelmätiedoston tai ohjelmamoduulin alkuun laitetaan esim. rivi

```
char merkki 1 ;
```

tai merkki voidaan suoraan alustaa seuraavasti:

```
char merkki 1=' A' ;
```

Huomaa, että C-kieli käyttää heittomerkkejä, eikä lainausmerkkejä yhden merkin alustuksessa. Tätä muuttujan automaattista alustusta, ei voi käyttää kaikissa C-kielen versioissa aliohjelman sisällä. Kääntäjät antavat silloin virheen, joten se tulee kyllä esille.

Merkkijono on yksinkertaisesti useita merkkejä peräkkäin, esim. kymmenen merkin jono esitellään:

```
char merki t[10];
```

Koska C-kieli käyttää pääsääntöisesti ASCII 0 (=NULL)-merkkiä lopettamaan merkkijonon, käytettävissä on vain yhdeksän merkkiä omaan ohjelman käyttöön.

Taas merkkijono voidaan esitellä valmiiksi alustettuna.

```
char merki t[10]=" KALLE" ;
```

Tämä onkin sitten ainoa tapa, jolla merkkijonoon voi laittaa dataa yhtäläisyysmerkin avulla. Koska C-kielessä kaikki toimenpiteet paitsi lukujen asetukset ja aritmetiikka tehdään funktiolla, myös merkkijonojen käsittely tehdään omilla erityisfunktioillaan, vaikka ne on erittäin helppo tehdä myös itse. Jos merkkijonoon halutaan laittaa ohjelman aikana vakio dataa, se pitää tehdä merkkijonon kopiointi-funktiolla `strcpy()`, esim.:

```
strcpy ( merki t, "Viile" );
```

Tässä funktiossa on kaksi parametriä; merkit ja vakiomerkkijono "Ville". Tämä funktio sitten siirtää merkit eli tässä tapauksessa kirjaimet yksi kerrallaan muistipaikkaan, joka alkaa kohdasta merkit. Kahden merkkijonon yhdistäminen tapahtuu samalla periaatteella.

```
char merki t1[10]="Hei ";  
char merki t2[10]="vaan";  
strcat (merki t1, merki t2); // tämä on C-kielen ulkoinen kahden  
merkkijonon yhdistämisfunktio, joka on esitelty string.h-tiedostossa C:n  
vakiokirjastohakemistossa.
```

Merkkijono merkit1 sisältää nyt "Hei vaan". Nämä merkkijonoihin liittyvät funktiot ovat kaikki esitelty lähdekielisessä muodossaan string.h tiedostossa. Muita käyttökelpoisia funktioita ovat merkkijonon pituus, merkkijonon etsiminen toisesta merkkijonosta, strncpy() eli kopioidaan vain haluttu määrä merkkejä jne.

32 Osoittimet

Osoitin on muuttuja jonka arvo on osoite tietokoneen muistiin. Osoitin positiivinen luku 8,16 tai 32 bittinen. Pituus riippuu käytetyn tietokoneen muistista. Osoittimella voidaan osoittaa vain yhden tyyppisiä muuttujia. Osoitinta kasvattamalla se osaa hypätä oikean määrän muistipaikkoja eteenpäin riippuen muistinvarauksesta tälle muistityypille. Muuttujan muistipaikan osoite saadaan selville &-merkillä, joka laitetaan muuttujan eteen.

Esim.

```
lukuosoi ti n = &luku;
```

Hyvin harvoin tarvitsee tietää mikä on osoittimen arvo, sitä käytetään vain epäsuorasti muistia käsitettäessä. Ennen kuin osoitin on alustettu se voi osoittaa minne tahansa, yleensä väärään paikkaan. Osoittimilla saa helposti ohjelman sekaisin, jos ei tiedä mitä tekee. Osoittimet eivät ole välttämättömiä, monia kieliä on tehty ilman osoittimia. Osoittimet pitää kuitenkin ymmärtää, jotta saa selville muiden C-ohjelmoijien ohjelmista. Osoittimista on hyötyä eniten kun niillä

osoitetaan merkkijonoja ja struktuureja. Aliohjelmille voidaan välittää useita osoittimia ja saada näin monta muuttujaa takaisin. Normaalisti aliohjelma voi palauttaa vain yhden muuttujan kerrallaan. Väittämällä aliohjelmalle osoittimen struktuuriin, on helppo ja yleinen tapa hoitaa aliohjelmassa suuri joukko muutoksia ilman yleismuuttujien käyttöä.

Osoitin varataan samoin kuin muuttuja, mutta sen eteen laitetaan tähti, esim.:

```
char merkki, *merkki osoitin; // varataan merkki ja sen osoitin

int luku1, luku2;
int *lukuptr; // varataan kokonaisluvun osoitin
int x;

luku1 = 100;
lukuptr = &luku1;
x = *lukuptr; // x on nyt 100
(*lukuptr)++; // lisätään luku1 yhdellä
*(lukuptr++); // lisätään osoitinta yhdellä, lukuptr saattaa nyt osoittaa
minne tahansa, mutta luultavimmin lukuun2. Virhekäskey!!
luku2 = &luku2; // nyt luku sisältää muuttujan osoitteen, joka saattaa
olla esim. 32-bittinen luku. Tätä ei tehdä tällä tavalla normaalisti. Virhe!!
```

33 IO-rutiinit

C-kieleen ei kuulu minkäänlaisia tulostus- eikä lukukäskeyjä, jotta kieli olisi mahdollisimman koneriippumaton. C-kieleen on kuitenkin useimmat valmistajat tehneet kirjastoja, joiden avulla voidaan tehdä ainakin perustoimintoja kuten merkin kirjoitus päätteelle ja merkin luku näppäimistöltä. Nämä valmiit funktiot linkitetään käännöstä tehtäessä lopulliseen ohjelmaan. Nämä rutiinit pitää esitellä kuten kaikki muutkin funktiot, ennenkuin niitä voidaan käyttää. Yleensä kääntäjän valmistaja toimittaa nämä esittelyt standardikirjastohakemistossa ja ne voidaan ottaa yksinkertaisesti käyttöön

```
#include <stdio.h>
```

merkinnällä ennen niitä käytäviä funktioita, yleensä aivan ohjelman alussa.

Periaatteessa siellä pitäisi olla myös printf ja scanf-funktiot, mutta tämä valmistaja on sisällyttänyt ne jo perus C30-kieleen.

getc(), getchar(), getch() ovat samoja merkinluku funktioita ja putc() ja putchar() ovat samoja merkinkirjoitusfunktioita. puts() on merkkijonon kirjoitusfunktio.

Pieni esimerkki näiden funktioiden käytöstä

```
while (1) {  
    putc(getc());  
}
```

Tämä esimerkki kiihdyttää kaikki merkit takaisin sarjaliikenteelle. Ennen kuin tämä ohjelma toimii PIC:llä pitää tietysti määrittellä sarjaliikenneportit, nopeudet, ja kellotaajus.

```
#include "p30F6012.h"
```

```
//Kielen nopeus määrittely  
#define FCY 4000000 //käsytien suoritus (Osc x PLL / 4)
```

```
void main (void) {  
    while (1) {  
        putc(getc());  
    }  
}
```

Tässä onkin mainio esimerkki C-kielen voimasta. Toimiva terminaaliohjelma kahdella rivillä! Saman asian tekeminen assembler-kielellä vie hyvinkin puoli vuotta aloittelijalta.

string.h sisältää merkkijonoihin liittyviä vertailuja ja kopiointeja:

```
char strcat(char *s1, char *s2);  
char strchr(char *s, int c);  
int strcmp(char *s1, char *s2);  
int strncmp(char *s1, char *s2, int n);  
char strcpy(char *s1, char *s2);  
char strncpy(char *s1, char *s2, int n);  
int strcspn(char *s1, char *s2);  
int strspn(char *s1, char *s2);  
int strlen(char *s);
```

```
char strlwr(char *s);
char strtok(char *s1, char *s2);
char strstr(char *s1, char *s2);
```

stdlib.h sisältää lukujen muutoksia numeroiksi ja päinvastoin

```
float atof(char *s)
int abs(signed int i)
long labs(signed long l);
signed int atoi(char *s);
signed long atol(char *s);
```

Muita kirjastoja PCW/PCM:lla on ctype.h, math.h, trig.h ja suuri joukko sovellusesimerkkejä, jotka varsinaisesti eivät kuulu C-kielen peruskirjastoihin.

34 Muotoiltu tulostus printf() -funktiolla

Vakiotulostuslaitteelle tai sarjaliikennetulostus tehdään C-kielessä

```
int printf(char *format, arg1, arg2, ...)
```

-funktiolla. Tämä funktio muotoilee, muuntaa ja tulostaa sen parametrit format muotoilumerkkijonon avulla. Se palauttaa luvun kuinka monta merkkiä se pystyi tulostamaan. Parametrejä voi olla nolla tai niin monta kuin format-lausekkeessa on annettu parametrejä.

Muotoilumerkkijono sisältää tavallisia merkkejä, jota funktio tulostaa sellaisenaan ja muunnosmääritelmiä, joiden mukaan parametrit muotoillaan tai muunnetaan. Format-lauseke on lainausmerkeissä esim. "Terve". Siinä voi olla muunnosmääritelmiä esim. "Luku on %d".

Format-määritelmää seuraa oikea määrä muuttujia Esim.

"Luku on %d", luku

Jokainen muunnosmääritelmä alkaa %-merkillä. Seuraava merkki voi olla, tässä järjestyksessä:

miinus-merkki, joka määrittelee tulostuksen vasemmalle

numero, joka määrittelee minimi kentän pituuden. Jos luku ei mahdu sille varattuun tilaan tilaa suurennetaan, lukua ei katkaista edestä eikä takaa. Jos luku on pienempi kuin määritelty leveys tila täytetään tyhjillä väleillä.

piste erottaa kentän leveyden tarkkuusmääritelmästä

tarkkuusnumero määrittelee montako merkkiä tulostetaan merkkijonosta tai montako merkkiä tulostetaan liukuluvusta desimaalipisteen jälkeen tai kokonaisluvuista minimäärän merkkejä.

h jos luku tulostetaan `short`-muodossa tai `l` jos luku tulostetaan `long`-muodossa

Leveys voidaan antaa *-merkillä, jolloin leveys voidaan määrittellä `int`-parametrilla ja varsinainen tulostusmuuttuja on vasta seuraava. Esim.

```
printf("%.*s", leveys, merkkijono);
```

Seuraava merkki on muunnosmerkki, joka voi olla joku seuraavista:

d,i	<code>int</code> ; kokonaisluku
o	<code>int</code> ; merkitön kokonaisluku oktaalilukuina, ilman etunollia
X,x	<code>int</code> ; merkitön kokonaisluku heksadesimaaliluku, ilman etunollia x= A,B,C jne. X= a,b,c jne.
u	<code>int</code> ; merkitön kokonaisluku
c	<code>int</code> ; yksi merkki
s	<code>char *</code> ; merkkijono, joka tulostetaan kunnes tulee NULL eli <code>'\0'</code>
f	<code>double [-]m.ddddd</code> , mantissa kuudella merkillä, jos ei toisin ole määrätty
E,e	<code>double [-]m.dddddE-+-xx</code> tai <code>[-]m.dddddE-+-xx, ,</code> mantissa kuudella merkillä, jos ei toisin ole määrätty
G,g	<code>double</code> ; käytä e, tai E jos eksponentti on pienempi kuin -4 tai suurempi tai yhtä suuri kuin tarkkuus, muuten käytä %f. Lopussa olevia nollia tai desimaalipistettä ei tulosteta
p	<code>void *</code> ; osoitin tulostusmuoto riippuu sovelluksesta, esim. PC:llä <code>0000:0000 segment:offset</code>
%	tulostetaan %

Esim:

```
: %s:           : Terve maailma:
: %13s:        : Terve maailma:
: %. 10s:      : Terve maailma:
: %-13s:       : Terve maailma:
: %. 15s:      : Terve maailma :
: %-15s:       : Terve maailma :
```

```
: %15.10s:           :         Terve maai :  
: %-15.10s:         : Terve maai       :
```

PIC C kielessä ei ole `doubl e` tyyppiä, mutta koska yllä esitetyt määritelmät ovat yleisiä kaikille C-kielille, ne esitetään tässä täydellisyyden takia. Osoittimia tarvitsee hyvin harvoin tulostaa, mutta vianhaku tilanteessa tästäkin saattaa olla hyötyä.

35 Input-lauseke

Formatoitu input-funktio on nimeltään

```
int scanf (char *format, . . .)
```

Formatoitu input lauseke on melkein samanlainen kuin tulostus-lauseke, `printf()`, mutta tärkeä ero on se, että `scanf()` -lausekkeessa parametrit pitää olla osoittimia vastaaviin muuttujiin.

Esimerkiksi päivämäärän luku numeroina ja kuukauden luku kirjaimina

```
void main ( void) {  
    int paiva, vuosi ;  
    char kuu[20];  
    scanf("%d %s %d", &paiva, kuu, &vuosi );  
}
```

Huomaa, että `&paiva` on muuttujan `paiva` osoite, mutta `kuu` on osoite merkkijonoon `kuu`. `&vuosi` on taas osoite muuttujaan `vuosi`. PCM/PCW-kielessä ei kuitenkaan ole `scanf()`-funktioita, se pitää itse tehdä `getc()`-funktioista.

36 C-kielen vakiomerkit

Erityisesti formatointilauseessa on tarvetta esittää tulostuslaitteelle meneviä merkkejä, joita on vaikea esittää normaalilla tavalla. Esimerkiksi `NULL`, lainausmerkit jne. C-kielen määrittelyssä on sovittu näille merkeille erikoismerkintä takakenon (`\`) ja toisen apumerkin avulla.

Newline = uusi rivi	<code>\n</code>	LF	10
Horisontal tab= vaakatabulointi	<code>\t</code>	HT	9
Vertical tab = pystytbulointi	<code>\v</code>	VT	11
Backspace= peruutusmerkki	<code>\b</code>	BS	8

Carriage return= vaunun palautus	\r	CR	13
Formfeed=lomakkeen siirto	\f	FF	12
Audible alarm=äänimerkki	\a	BEL	7
Backslash= takakeno	\\	\	92
Kysymysmerkki	\?	?	63
Heittomerkki	\'	'	96
Lainausmerkki	\"	"	34

Tässä taulukossa on viimeisenä ASCII taulukon arvo ko. merkille desimaalisena.

PIC C voidaan käyttää myös oktaalilukumerkintää. Tässä on alkumerkkinä \0 ja sen jälkeen kaksi oktaalilukua esim. BEL on \007.

Vielä yksi mahdollisuus on merkitä ASCII-merkki \x00, jossa 00 on ASCII-merkin hex arvo.

Esim.

```
printf("Uusi rivi \r\n jatkuu \b soita kelloa");
```

tai

```
printf("Uusi rivi \x0d\x0a jatkuu \x07 soita kelloa");
```

Sama asia voidaan tehdä myös käyttämällä char-muuttujaa ja laittamalla sille arvoksi sopiva luku. Esim.

```
char cr =10, kello =9;
```

```
printf("Uusi rivi %c jatkuu %c soita kelloa", cr, kello);
```

Samoja merkin korvauksia voidaan käyttää yhden merkin muuttujine asetuksissa esim.

```
char ch= '\n' ;
```

Jotta kääntäjä saisi selvää käyttäjän tarkoituksesta, pitää merkin korvike laittaa heittomerkkeihin. Saman asian voi tehdä myös #define määrittelyllä.

```
#define CR 10
```

```
char ch=CR;
    tai yksinkertaisemmin
char ch=10;
```

37 dsPIC33F testiohjelma

```
/*=====00
Ohjelma vilkuttaa ledejä ja lähettää "Hei, hei Probyte" UArt 2:lla
9600 8n1 MPLAB C30

dsPIC33F ja Microchip Explorer 16 demolevy
dATE: 22.10.2006
FILE; DSPIC33f_UART2DEMO,C
=====*/
#include <libpic30.h> /* uusi header tiedosto */
#include <stdio.h>

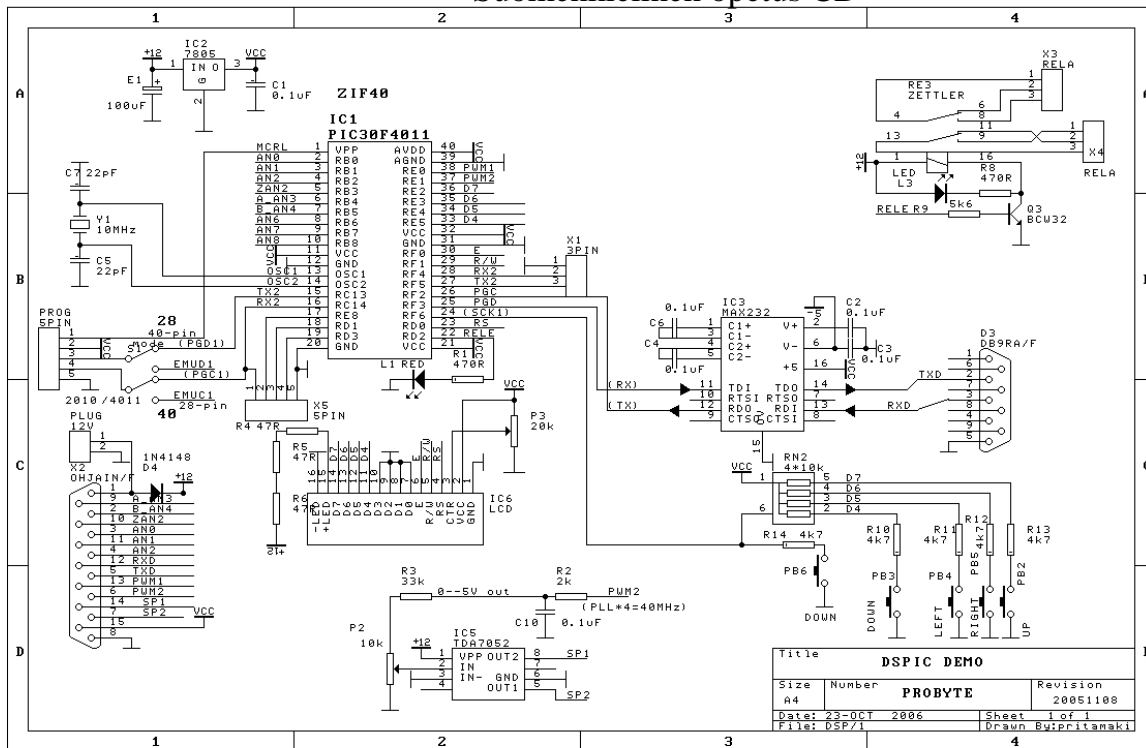
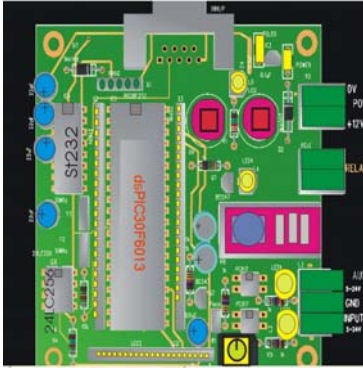
#ifndef __dsPIC33F__
#error vain 33F Cpu ja explorer 16
#endif
#include <p33Fxxxx.h>

_FOSCSEL(FNOSC_PRI );
_FOSC(FCKSM_CSDCMD & OSCIOFNC_OFF & POSCMD_XT);
_FWDT(FWDTEN_OFF);

void main(void ) {
    ODCA = 0;
    TRISAbits.TRISA6 = 0;
    __C30_UART=2;
    U2BRG = 38;
    U2MODEbits.UARTEN = 1;
    while (1) {
        __builtin_btg(&LATA,6);
        printf("Hei, hei Probyte  %d\n",U2BRG);
    }
}
```

38 Microchip dsPIC demokortit

PROBYTE dsPIC demokortti
 sarjaliikenneliitin +MAX232 muunnin
 LCD –liitin, kideoskillaattori
 4*4 näppäimistöliitin
 Potentiometri
 ON-line ohjelmistoliitin PARPIC2:lle
 2 kpl optoa
 1 kpl 10A 230V rele
 EEPROM I2C paikka 1Mbit
 8* 10 bit ADC
 Nappeja ledejä
 +12regulattori +5V varten
 Suomenkielinen opetus CD



DM300014 dsPIC30F6014 on demolevy suodin,
 FIR ja IIR digitaalisia suotimia varten DTMF
 äänen tuottaminen (2* UARTs, SPI™, CAN, RS-
 485), Si3000 äänialueen codec MIC in/speaker
 jacks, 22x32 grafiikka LCD, LED, potentiometri





DM300017 on dsPIC201x sarjan 28-nastaisille dsPIC:elle tehty kehityskortti, dsPIC20010 on tavallisella DIP28 kannalla, sarjaliikenne ja ohjelmointiliitin ICD2:lle



DM300018 on demokortti 28 ja 40-nastaisille moottoriohjain dsPIC:malleille. Myös 18-, 28- ja 40-nastaisia anturi dsPIC voidaan käyttää . dsPIC30F4011 piiri on mukana. RS-232 ja CAN portit. LED lämpötila-anturi 2x16 LCD .

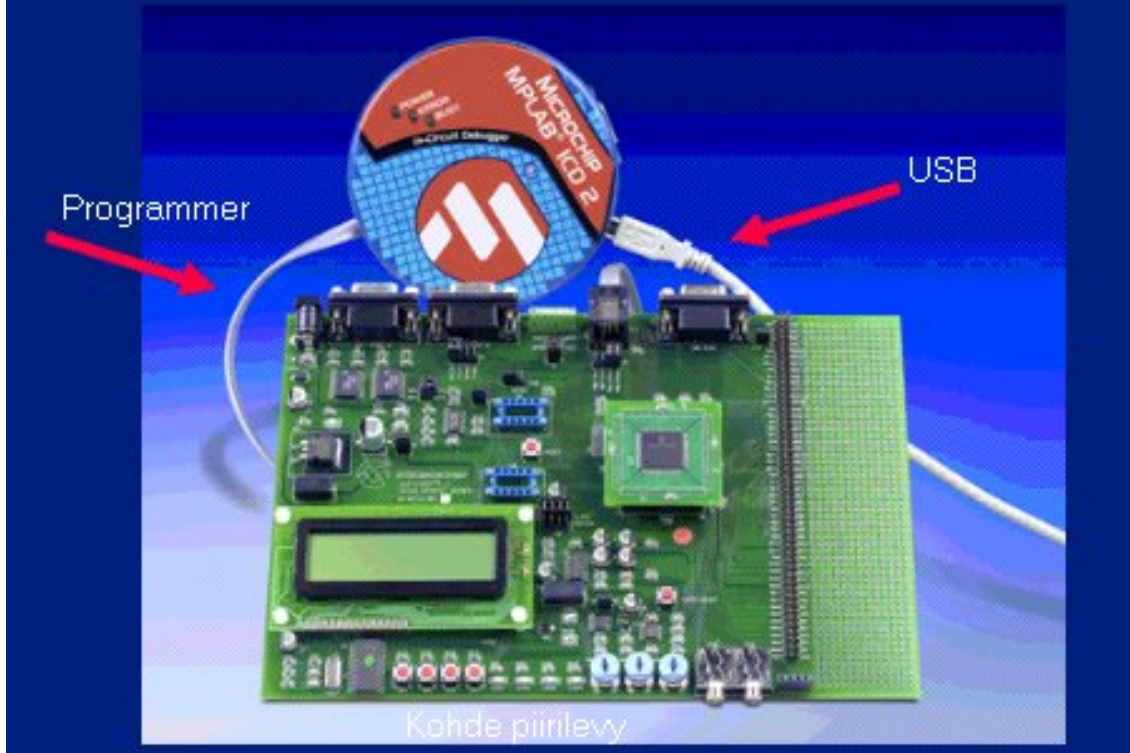


DM300019 demolevyssä on 80-nastaisen dsPIC30F6014 plug in-moduli 5V /3.3V valinta, LEDit, kytkimet ja potentiometri, UART-liitäntä , A/D suodn puheelle, DAC ja suodin



Vasemmalla Microchip Explorer 16 demolevy DSPIC33F sarjalle. Lisälaitekortteja voidaan asentaa kahteen liittimeen, jolloin demokortti on valmis erilaisiin opetustehtäviin nopeasti.

Pyöreä laite on ICD2, jolla voi ohjelmoida suoraan Microchipin demolevyjä



ICD2 ja MPLAB on tehokas työpari ja silti edullinen



MPLAB[®] ICE 4000

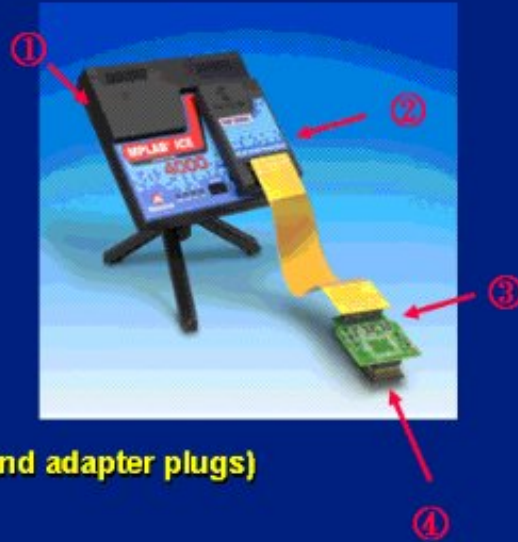
① Emulator Pod Kit

- ♦ MPLAB ICE 4000 pod
- ♦ Power supply
- ♦ USB cable
- ♦ Literature
- ♦ MPLAB CD-ROM
- ♦ Tripod
- ♦ Logic probes
- ♦ Modular flex cable

② Processor Module

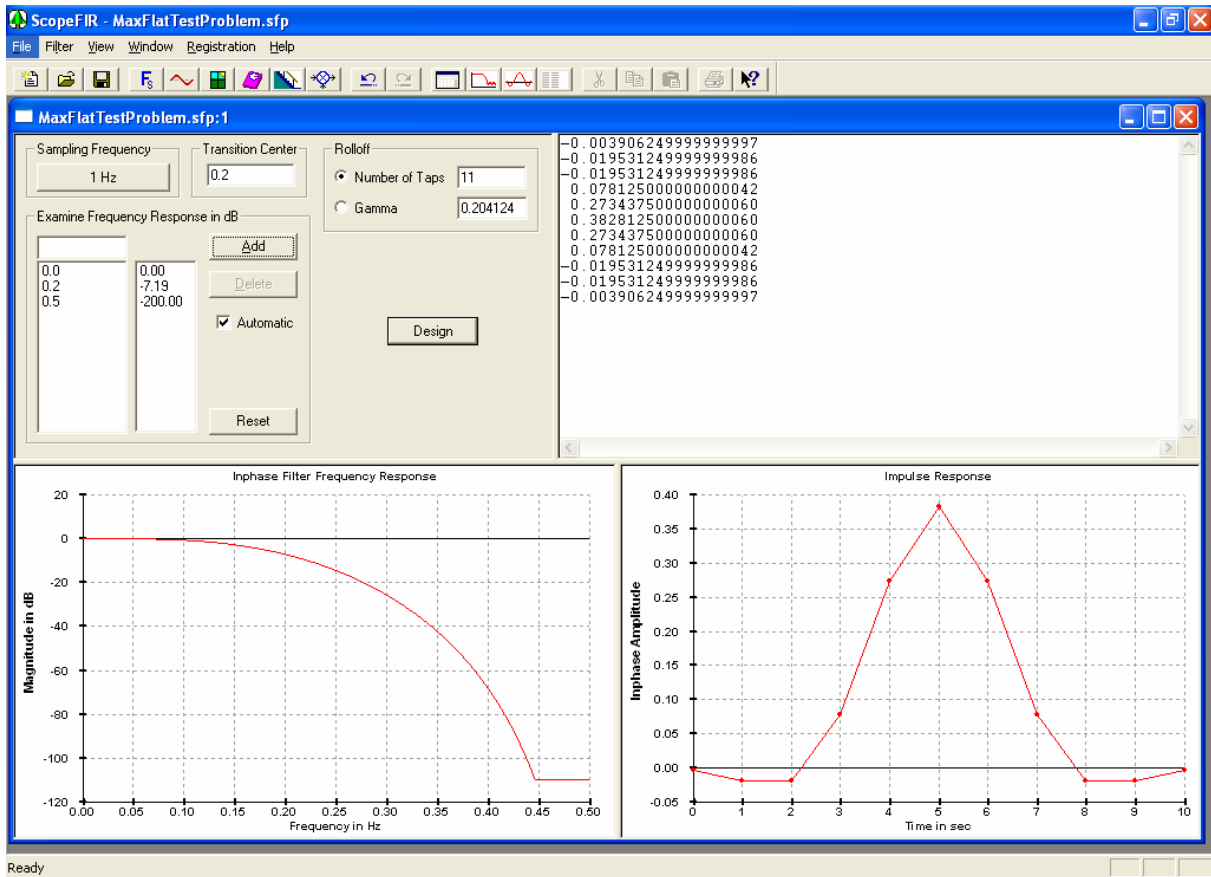
③ Device Adapter (includes board and adapter plugs)

④ Transition Socket (optional)



ICE4000 on Microchipin paras reaaliaikainen emulaattori. Sillä voi testata ohjelmia täydellä nopeudella. Hinta on kuitenkin useimmille liian kallis

39 FIR suodin ohjelman voi käynnistää CD:tä FIR suodin hakemistosta



40 Lähdeluettelo

1 Brian W.Kernichan Dennis M.Ritchie The C programming language. Second Edition Prentice Hall, NJ, USA 1986

2 Digital Signal Processing A practical Approach. Emmanuel C. Ifeachor, Barrie W.Jerves, Addison Wesley 1993

3 Probyte Oy, PIC-CD 2006 www.probyte.fi

4 Datalehdet www.microchip.com

5 dsPIC demolevyjä

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2519¶m=en025319&page=wwwdsPICDSCDemo