KYMENLAAKSON AMMATTIKORKEAKOULU

Ohjelmistotekniikan koulutusohjelma

Mikko Suikkanen

PIC-MIKROKONTROLLERIN HARJOITUSALUSTAN SUUNNITTELU

Insinöörityö 2009

TIIVISTELMÄ

KYMENLAAKSON AMMATTIKORKEAKOULU

Ohjelmistotekniikan koulutusohjelma

SUIKKANEN, MIKKO PIC-mikrokontrollerin harjoitusalustan suunnittelu

Insinöörityö 51 sivua + 5 liitesivua

Työn valvoja DI Jaakko Levomäki

Työn tilaaja Kymenlaakson ammattikorkeakoulu

Marraskuu 2009

Avainsanat mikrosirut, microchip, sulautettu tietotekniikka, ohjelmoitavat logiikat, piirilevyt, suunnittelu

Työn tarkoituksena oli suunnitella harjoitusalusta ohjelmoitavien logiikkapiirien opetuksen avuksi. Alustalle luotiin mahdollisimman monipuolinen valikoima työkaluja eri toimintojen toteuttamiseksi ja niiden käytön harjoittelemiseksi.

Ohjelmoitavat logiikkapiirit eli mikrokontrollerit ovat yhä suuremmassa osassa nykyisessä tekniikkassa, ja ne esiintyvät kaikilla elämän osa-alueilla erilaisina sulautettuina järjestelminä.

Harjoitusalustan tarkoitus on antaa työkalu ohjelmoitavien logiikoiden opetukselle. Siinä käytetään Microchipin PIC 16F874 piiriä. Syynä piirin valintaan on sen monipuolinen moduulivalikoima sekä kohtuullinen muistin määrä.

Työ toteutettiin suunnittelemalla alustan toiminnalliset komponentit valvovan opettajan kanssa, jonka jälkeen piirilevysuunnittelu tehtiin PADS-ohjelmistolla ja ohjelmistoa kehitettiin ja testattiin Microchip:n simulaatio-ohjelmilla.

ABSTRACT

KYMENLAAKSON AMMATTIKORKEAKOULU

University of Applied Sciences

Software Engineering

SUIKKANEN, MIKKO

Designing Practice Board for PIC Microcontroller

Engineering thesis 51 pages + 5 pages of appendices

Supervisor Jaakko Levomäki, MSc

Commissioned by Kymenlaakson Ammattikorkeakoulu

November 2009

Keywords microchip, embedded system, programmable logic, circuit board

The aim of this study was to create a demo board for educational uses. The board is operated with a Microchip PIC 16F874 circuit. The board allows students to operate and test different modules of PIC.

Embedded systems are an ever growing part of every aspect of modern society. Microcontrollers are the very heart of every embedded system. Learning to operate and program microcontrollers is a more and more essential skill.

PIC 16F874 is selected because it has a wide range of different modules and it has a decent amount of memory. Also, programming PIC 16-series circuits is simple thanks to the RISC command set and simple structure.

The work was done in cooperation with the teacher. The designing of the board was made with PADS software and the software was developed and tested with Microchip’s own simulation software.

Sisällysluettelo

Termit ja lyhenteet

AC; Alternating Current; vaihtosähkö. Jännitteen hetkellinen arvo ei ole vakio

A/D-muunnos; Analog-to-Digital Conversion; analogisignaalin arvo muunnetaan digitaaliseen esitysmuotoon

ALU; Arithmetic-Logic Unit; piirin sisäinen moduuli, joka suorittaa laskennalliset ja loogiset operaatiot

Analoginen; Jatkuva signaali, johon vaikuttaa sekä aika että amplitudi, pienetkin muutokset ovat vaikuttavia

Asettaminen; Pakottaa bitin tilaksi 1 (päällä)

Asynkroninen; Erilliskellotettu, laitteiden välinen toiminta tapahtuu ennaltasovitulla nopeudella molempien laitteiden toimiessa omaan tahtiinsa

Baud Rate; Tiedonsiirtonopeus, symbolia sekunnissa; ei pidä sekoittaa termiin ’bittiä sekunnissa’ eli todelliseen nopeuteen

Binääri; 2-lukujärjestelmä [0,1]

Bitti; Binäärijärjestelmän pienin yksittäinen kokonaisuus: 1 tai 0, tosi tai epätosi. Puhuttaessa X-bittisestä järjestelmästä tarkoitetaan käsiteltävän laitteen rekisterin leveyttä (4-bittinen, 8-bittinen…)

BNC; 2-napainen liitin, yleensä käytetään mittalaitteissa

BOR; Brown-Out Reset; jos käyttöjännite putoaa alle sallitun, mutta ei kuitenkaan katkea kokonaan, ajautuu laite epämääräiseen tilaan, jossa sen toimintaa ei ole taattua. Tällöin suoritetaan Brown-Out –resetointi, jossa laite sammutetaan, kunnes käyttöjännite on jälleen vakaa.

CISC; Complex Instruction Set Computer; Yleisnimitys arkkitehtuureille, joissa konekielen käskyt ovat rakenteeltaan monimutkaisia ja yksi käsky voi sisältää useita perusoperaatioita

D9-liitin; 9-napainen liitin, yleensä käytetään sarjaväylälle

DC; Direct Current, tasavirta; Jännitteen arvo pysyy vakiona ajan kaikilla hetkillä

Demultipleksaaminen; DeMUX on kytkin, joka antaa ulostulona tiedon siitä, minkä arvoinen sisääntulo on kytkennän suorittanut. Demultipleksaaminen numeroi nipun signaalijohtimia ja muuntaa yksittäisen johtimen arvon sitä vastaavaksi binääriarvoksi. Näin pystytään vastaanottamaan pienemmällä määrällä laitteen signaalijohtimia suurempi määrä signaaleja

Desimaali; 10-lukujärjestelmä [0-9]

Digitaalinen; Epäjatkuva ja hetkellinen signaali, jaettu tasakokoisiin osiin, esitetään yleensä binäärimuodossa

DSP; Digitaalinen Signaaliprosessori; analogisignaalin käsittelyyn erikoistunut prosessori, myös Digital Signal Processing, digitaalinen signaalin käsittely

Duplex; Tiedonsiirtomuoto;

- Half tilassa laitteet voivat lähettää yksi kerrallaan, tällöin muut vastaanottavat
- Full tilassa jokaisella laitteella on oma lähetys- sekä vastaanottokanava

Fosc; Kellopiiri, voi olla joko ulkoinen tai sisäinen värähtelijä

Heksadesimaali; 16-lukujärjestelmä [0-9, A-F]

Hz; Hertsi, värähdystä sekunnissa

(i); Invertoitu l. 0-aktiivinen; linja on aktiivisena 0-tilassa ja toisinpäin

I/O; Input/Output; datan sisään- ja ulostulo laitteelle

Kellojakso; Yksi värähdys, ei pidä sekoittaa työjaksoon, joka koostuu prosessorista riippuen yhdestä tai useammasta kellojaksosta

Keskeytys; Sisäinen kutsu, jolla prosessointi annetaan keskeytyksen määrittämälle tehtävälle

LED; Light Emitting Diode; Hohtodiodi tai ledi on puolijohdekomponentti, joka säteilee valoa, kun sen läpi johdetaan sähkövirta. Kutsutaan arkikielessä myös valodiodiksi.

Lippu; Rekisterissä oleva bitti, jolla ilmaistaan jonkin asian tapahtuneen, esim. zero-flag eli nollalippu

Multipleksaaminen; MUX on kytkin, joka kytkee sen ulostulon, jonka arvo on annettu sisääntulona MUX:lle. Multipleksaaminen numeroi nipun signaalijohtimia, ja muuntaa binääriarvon sitä vastaavaksi yksittäiseksi johtimeksi. Näin pystytään lähettämään pienemmällä määrällä laitteen signaalijohtimia suurempi määrä signaaleja

Nollaaminen; Pakottaa bitin tilaksi 0

Oktaali; 8-lukujärjestelmä [0-7]

PC; Program Counter; ohjelmalaskuri

PCL, Program Counter Latch; ohjelmalaskurin alempi tavu

PCLATH, Program Counter LATch High; ohjelmalaskurin ylemmät 5 bittiä

PLC; Programmable Logic Controller; ohjelmoitava logiikkayksikkö

Pollaaminen; Polling, kiertokysely; tasaisin väliajoin suoritettava jonkin asian lukeminen tai tarkistaminen

PORTx; Portin x käsittelyrekisteri, kirjoitusasu on joko PORT (yleisesti) tai PORTx (joku tietty portti) esim. PORTA. Jos porteista puhutaan yleisesti tai tarkoitetaan kokonaisuutta, johon kuuluu myös ohjausrekisteri on kirjoitusasu PortX, esim. PortA

PWM; Pulse Width Modulation; pulssinleveysmodulaatio

RAM; Random Access Memory; työmuisti

RC; Vastuksen (Resistor) ja kondensaattorin (Condensator) muodostama piiri, jota käytetään suodattamaan signaalia tai muodostamaan värähtelijäpiiri

Rekisteri; Käyttömuistissa sijaitseva tavu, joka on ennalta määritelty erityiskäyttöön tai on vapaasti käytettävissä

Resoluutio; Erottelukyky, kertoo kuinka moneen osaan jokin asia on jaoteltu, esim. 10-bittinen resoluutio = 1024 osaa

RISC; Reduced Instruction Set Computer; arkkitehtuuri, joka käyttää vakiomittaisia konekielisiä komentoja

RJ-45; 8-napainen parikaapeliliitin, käytetään yleensä tietoverkoilla

ROM; Read-Only Memory; ohjelmamuisti. Erilaisia ROM-muisteja:

PROM	Ohjelmoitava muisti, ei-tyhjennettävä (ns. OTP)
EPROM	UV-valolla tyhjennettävä ohjelmamuisti
EEPROM	Sähköisesti tyhjennettävä ohjelmamuisti
Flash-ROM	Sähköisesti lohkoittain tyhjennettävä ohjelmamuisti

RX#; Yksittäinen I/O-nasta, jossa X on porttikokonaisuuden tunnus ja # on kyseisen I/O-nastan numero [0-7], esim. RA4 on PortA:n 5. I/O-nasta

Synkroninen; Yhteiskellotettu, laitteet toimivat saman kellon tahdissa

Tavu; 8 bitin kokonaisuus

TRISx; Portin X ohjausrekisteri, kirjoitusasu on joko TRIS (yleisesti) tai TRISx (joku tietty portti) esim. TRISA

USART; Universal Synchronous Asynchronous Receiver Transmitter; Ohjelmoitava sarjaväylä

USB; Universal Serial Bus; eräs sarjaväylästandardi

Väylä; Väylä on yleisnimitys joukolle johtimia, joilla välitetään tietoa. Ulkoisia väyliä käytetään tiedonsiirtoon toisten laitteiden kanssa.

> Sarjaväylä; Sarjaväylässä tieto lähetetään johdinparia käyttäen, jolloin voidaan lähettää yksi bitti kerrallaan. RS-232 on jo vanhentunut standardi, joka on korvattu usb:lla tai firewirellä
>
> Rinnakkaisväylä; Rinnakkaisväylässä tietoa siirretään useammalla johtimella samanaikaisesti (4, 8, 16…) ja se vaatii sekä lähetys- että vastaanottopäässä yhtä monta tähän tehtävään tarkoitettua nastaa. Rinnakkaista tiedonsiirtoa ei enää ulkoisille laitteille juuri käytetä, vaan sitä käytetään lähinnä laitteistojen sisäisissä tehtävissä.

W; Työrekisteri; ainoa rekisteri, johon voidaan suoraan kirjoittaa luku, muihin rekistereihin on tieto siirrettävä W:n kautta

1. JOHDANTO

1.1. Alkusanat

Ajatus harjoitusalustasta PIC-mikrokontrollerille sai alkunsa jo ammattikoulussa, jossa opettaja oli valmistanut tässä työssä sunniteltua laitetta vastaavan laitteen opetuksen avuksi. Ammattikorkeakoulussa vastaava opetus suoritettiin täysin teoreettisesti ilman tällaista laitetta. Projektia alettiin suunnittelemaan oppilaslähtöisesti tämän tarpeen täyttämiseksi. Tämä dokumentti on tuloksena tästä suunnittelusta.

Haluan kiittää opettajiani ja läheisiäni tuesta ja kannustuksesta tämän työn tekemisen aikana. Ilman teitä ja tukeanne ei tämä työ olisi koskaan valmistunut.

1.2. Microchip

Microchip on perustettu vuonna 1987 ja se on yksi johtavista mikrokontrollereiden ja analogisten puolijohteiden valmstaja. Se syntyi General Instrumentin eriyttäessä mikroelektroniikan alansa omaksi tytäryhtiökseen. Microchip valmistaa niin muisti- kuin salauspiirejä, digitaalisia signaalin käsittelypiirejä kuin mikrokontrollereja. Yksi sen myydyimmistä tuotteista on keskiluokan eli 14-bittiset mikrokontrollerit.

1.3. Työn tarkoitus

Mikrokontrolleri on tietokone rakennettuna yhdelle piirille. Niillä on käyttöä niin teollisuudessa ja kotitalouksissa kuin missä tahansa muualla, missä käytetään monimutkaisempia elektronisia järjestelmiä.

Harjoitusalustaprojekti aloitettiin syksyllä 2008 opiskelijan ehdotuksesta. Ehdotuksen mukaisesti tarkoitus oli valmistaa alusta Microchip:n PIC-mikrokontrollerille niin, että alustaa hyväksi käyttämällä pystyttäisiin opiskelemaan mikrokontrollerin ohjelmointia, ja sen eri ominaisuuksien käyttöä.

1.4. Työn sisältö

Luvussa 2 käsitellään automaation ja mikrokontrollereiden taustaa. Luvussa 3 käydään läpi harjoitusalustan keskeiset komponentit, luvussa 4 16F874-piirin vastaavien ominaisuuksien toiminta sekä yleisesti MPLAB-ympäristöä ja ohjelmointia käsitteleviä asioita. Luvussa 5 käydään läpi työtä itseään ja luvussa 6 on lyhyt analyysi projektin suunnittelusta ja toteuttamisesta.

2. TAUSTAA

2.1. Ohjaus- ja säätöjärjestelmät yleisesti

Ohjaus- ja säätöjärjestelmiä on ollut aina teollisen aikakauden alusta asti. Ne ovat laitteen tai laitekokonaisuuden osia, joilla ohjataan laitteen toiminnan tiettyä osa-aluetta, joko käyttäjän suoralla vaikutuksella tai automaattisesti antureiden antamien syötteiden avulla.

Ohjattava laite voi olla niinkin yksinkertainen kuin vesijohtoverkostoon liitetty hana tai monimutkainen, kuten auton moottori. Anturi voi lukea esimerkiksi painetta tai virtausnopeutta tai se voi olla ajastin tai rajakytkin. Aluksi tällainen anturiohjattu säätöjärjestelmä on ollut ihminen, joka on lukenut höyrynpaineen mittaria ja mukauttanut koneen toimintaa sen mukaisesti. Teollistumisen voidaan laskea alkaneen Kehruu-Jennystä.

2.1.1. Kehruu-Jenny

John Kay keksi vuonna 1733 pikasukkulan, joka nopeutti huomattavasti kutomista. Tämä asetti vaatimuksen myös kutomisen raaka-aineen eli langan nopeammalle valmistamiselle. Ratkaisun toi Kehruu-Jenny, jonka Hargreaves kehitti 1760-luvulla. Kehruu-Jennyssä oli rukin tavoin yksi pyörä mutta rukista poiketen kahdeksan värttinää, joille lanka kelautui samanaikaisesti. (1)

2.1.2. Spinning Mule

Spinning Mule oli mekaaninen rukki, joka valmisti korkealuokkaista lankaa lyhyessä ajassa. Sen loi 1779 Samuel Crompton, ja se oli yhdistelmä vesivoimalla toimivasta rukista sekä Kehruu-Jennystä. Näiden yhdistäminen tarkoitti sitä, että voitiin sekä kehrätä useampaa kuin yhtä lankaa kerrallaan ja se tehtiin ilman ihmisvoimaa. Myöhemmin vesivoima korvattiin moottorilla. Mule oli ensimmäinen askel kehityksessä kohti teollisuuslaitoksia, sillä se oli kotioloihin aivan liian suuri.

Keksinnöt kehräämisen alalta mahdollistivat suuren tuotannon kasvun tekstiilituotannon, erityisesti puuvillan, alalla. Sekä puuvilla että rauta olivat johtavia aloja teollisessa vallankumouksessa, ja molemmat kävivät läpi suuren tuotannon nousun samoihin aikoihin. Tätä pidetään yleisesti teollisen vallankumouksen alkuna. (2)

2.2. Mikroprosessorit ja säätöjärjestelmät

Aina 1970-luvun alkuun saakka säätö- ja ohjaustoiminnoista vastasi pääasiassa ihminen. 15. marraskuuta 1971 toi Intel markkinoille 4-bittisen mikroprosessorin 4004:n, jota pian seurasivat 8-bittiset 8008 ja 8080 sekä Motorolan 6800. Nämä ovat ns. yleisprosessoreita, eli niitä ei ole suunniteltu mihinkään tiettyyn käyttöön, vaan niistä valmistetaan käyttöön sopiva laitteisto.

Mikroprosessori, lyhyemmin prosessori, on matemaattinen yksikkö, joka tarvitsee ulkoiset työ- ja ohjelmamuistit (RAM ja ROM). Tämän lisäksi siihen yleensä liitetään jokin syötelaite, yleensä näppäimistö, sekä näyttö. Näiden avulla prosessorin toimintaa voidaan ohjata ja valvoa. Mikroprosessorin ympärille voidaan rakentaa laite, jota nykyään kutsutaan ohjelmoitavaksi logiikaksi (Programmable Logic Controller, PLC). Tällaisia valmistavat esimerkiksi Omron ja Siemens. Ohjelmoitava logiikka on mikroprosessori-pohjainen laite, jossa on joko modulaarisia tai integroituja tulo- ja lähtöportteja. Näihin kytketään prosessissa olevia antureita (paine-, lämpötila- jne.) sekä ohjauslaitteet (releet, merkkivalot, venttiilit, moottorinohjausyksiköitä jne.). Logiikka ohjaa prosessia käyttäjän laatiman ohjelman sekä antureiden antamien tietojen mukaisesti. (3)

Ohjelmoitavia logiikoita käytetään ohjaamaan säätö- ja ohjausjärjestelmiä useissa paikoissa, kuten tehtaissa ja hisseissä. Myös muita ohjausjärjestelmiä on käytössä eri käyttökohteissa, kuten auton automaattivaihteisto. Tällaisen ohjausjärjestelmän suurin etu on se, että ihmistä ei tarvita enää valvomaan ja ohjaamaan jotain tiettyä järjestelmää, vaan valvonta voidaan keskittää yleisesti koko prosessin matkalle, ja ihmistä tarvitsee käyttää prosessissa vain tarpeen vaatiessa, prosessin valvonnassa sekä huoltotoimenpiteitä suorittamaan.

2.3. Tietokoneet ja säätöjärjestelmät

Tietokone on mikroprosessorin ympärille rakennettu laite, jossa on sekä ulkoinen työ- että ohjelmamuisti ja laitteisto sekä tiedon syöttämistä että lukemista varten. Tietokoneessa on myös yksi tai useampi järjestelmästä ulos johtava väylä ulkoisten laitteiden kanssa käytävää kommunikointia varten. Tällaisia väyliä ovat erilaiset sarja- ja rinnakkaisväylät. Ennen näitä edustivat RS-232 ja LPT eli tulostinportti. Nykyään nämä on korvannut usb ja firewire sekä erilaiset langattomat ratkaisut.

Tietokoneen etu ohjelmoitavaan logiikkaan nähden on se, että tietokone voi suorittaa useampia ja monimutkaisempia tehtäviä kuin logiikka. Logiikka on suunniteltu tietynlaisen laitteiston kanssa suoritettaviin tehtäviin, kun taas tietokone on yleiskäyttöinen, jonka käyttö määräytyy pitkälti käytettävissä olevasta laitteistosta ja ohjelmistosta. Ohjelmoitavan logiikan etuna taas on huomattavasti halvempi laitteisto ja asennus, ja ohjelmoitava logiikka on jo valmiiksi suunniteltu kyseisen prosessin ohjaamiseen.

Yleensä tietokone ja ohjelmoitava logiikka eivät kilpaile käytöstä prosessiteollisuudessa, vaan ne toimivat rinnakkain, kumpikin hoitaen omia tehtäviään.

2.4. Mikrokontrollerit yleisesti

Mikrokontrolleri on ohjelmoitava mikropiiri, johon on sisällytetty aritmeettis-looginen yksikkö (ALU), ROM (ohjelmamuisti) ja RAM (työmuisti) sekä nykyisin yhä useammin joitain erikoistoimintoja, kuten analogi-digitaalimuunnin, sisään rakennettu väyläajuri, kaappaus ja vertailu - moduuli, PWM-generaattori ym.

Alkujaan mikrokontrollerit olivat paljon yksinkertaisempia, ja vaikka niiden fyysinen koko oli useimmissa tapauksissa nykyistä suurempi, oli toiminnallisuus paljon rajatumpi. Ensimmäisissä mikrokontrollereissa oli vain osoite- ja dataväylät sekä pieni sisäinen ohjelma- ja työmuisti. Kehityksen myötä väylistä on tullut yleiskäyttöisempiä, sisäisen muistin määrä on kasvanut, sisään rakennettuja toimintoja on tullut lisää ja niitä on yhä useampia yhdessä piirissä.

Piirien kehitys on näkynyt myös käytettävässä muistissa. Vanhat piirit käyttivät EPROMia, jolloin uudelleenohjelmointia varten piirin muisti piti tyhjentää UV-valon avulla. Nykyisin käytössä on joko EEPROM- tai FLASH-muistilla varustettuja piirejä, joiden muisti tyhjennetään sähköisesti. Massatuotantoa varten tuotetaan PROM-muistilla varustettuja kertaohjelmoitavia piirejä (OTP, One Time Programmable).

Mikrokontrollerin toimintataajuus on yleensä huomattavasti matalampi kuin mikroprosessoreilla yleensä. Mikrokontrollereiden toimintataajuus on yleensä muutamista kilohertseistä aina kymmeniin megahertseihin. Prosessoreilla mitataan nykyään toimintataajuus gigahertseissä, mutta silti mikrokontrolleri on prosessoriin verrattuna hyvin tehokas. Monet toiminnot, joita niillä suoritetaan, tapahtuvat laitteistotasolla, arkkitehtuurina on nk. harvardilainen arkkitehtuuri. Nykyisissä laitteissa käytetään lähes aina RISC-käskykantaa, eli käskykanta on yleensä suppea sekä käskyt lyhyitä. Vaikka kellojaksoja mahtuukin samaan ajanjaksoon yleensä vähemmän kuin prosessorilla, tarvitaan niitä vähemmän asioiden suorittamiseen. Tämän lisäksi mikrokontrollerin koodi on yleensä vain murto-osan siitä, mitä vastaava ohjelma tietokoneella vaatisi. Jo pelkkä käyttöjärjestelmä voi vaatia miljoonia rivejä koodia, kun mikrokontrollerin koko ohjelma voi mahtua alle sataan tavuun. (4)

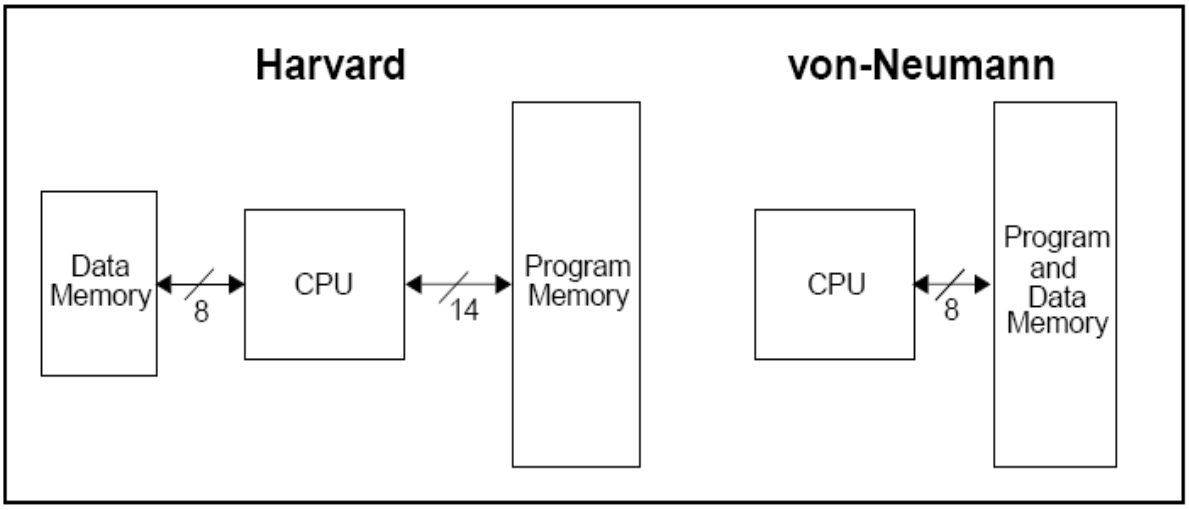
Esimerkki 1.

```
; Perinteinen harjoitus, jossa luetaan painonappia ja sytytetään led.
alku
        btfss       IN          ; IN on aiemmin määritelty painonapin sisääntulo
        goto        off
        bsf         OUT         ; OUT on aiemmin määritelty LEDi ulosmeno
        goto        alku
off
        bcf         OUT
        goto        alku
        end
```

Mikrokontrolleri ei pyri syrjäyttämään mikroprosessoria sen enempää kuin ohjelmoitava logiikka pyrkii syrjäyttämään tietokoneen. Sen sijaan mikrokontrolleri täyttää tarpeen tietokoneen ja ohjelmoitavan logiikan välistä. Mikrokontrollerin käyttökohteita ovat lähes kaikki sulautetut järjestelmät, joita löytyy esimerkiksi moottoreista, säätimistä, puhelimista… lyhyesti ilmaistuna kaikista sellaisista sähkölaitteista, jotka suorittavat mitään monimutkaisempaa toimintaa. Piirien tullessa yhä monimutkaisemmiksi pystytään niiden ominaisuuksia muokkaamaan

yhä enemmän ohjelmallisesti ja näin mahdollistetaan saman yleiskäyttöisen piirin käytön useissa erilaisissa käyttökohteissa.

2.4.1. Harvard vs. von Neumann



Kuva 1. Harvard vs. von Neumann

Harvardin arkkitehtuurissa ohjelma- ja datamuisti ovat erillisiä, niin muistiyksiköiden kuin väylienkin suhteen. Näin sekä käskyt että käyttömuisti voivat olla erimittaisia, niillä voi olla eri ajoitukset, implementaatiot tai muistiosoiterakenteet.

Vastakohtana Harvardin arkkitehtuurille von Neumannin arkkitehtuurissa on yksi säilö sekä ohjelmalle että työmuistille. Prosessori voi joko lukea käskyä tai lukea/kirjoittaa tietoa muistiin, koska sekä käskyt että tieto molemmat käyttävät samaa väylää. Muistirajapinnan vastuulla on erottaa käskyjen lukeminen sekä tiedon välittäminen edestakaisin prosessorin ja sen sisäisten rekisterien välillä.

Voisi vaikuttaa siltä, että muistirajapinta on pullonkaula prosessorin ja muistiavaruuden välissä. Useimmissa von Neumannin arkkitehtuurin ratkaisuissa näin ei ole, sillä käskyn suorittamiseen kulunut aika käytetään seuraavan käskyn hakemiseen. (5)

2.4.2. Nykyaikainen Harvardin arkkitehtuuri

Perinteinen harvardilaisen arkkitehtuurin etu – samanaikainen pääsy useampaan kuin yhteen muistijärjestelmään – on menetetty nykyisten välimuistijärjestelmien takia. Näin paljon joustavampi von Neumannin arkkitehtuuri on suorituskyvyltään vähintään yhtäläinen monissa tapauksissa. Muuteltu harvardilainen arkkitehtuuri pienentää näiden sovellusten välistä kuilua.

Nykyaikanen tehokas prosessorirakenne sisältää piirteitä sekä harvardilaisesta että von Neumannin arkkitehtuurista. Piirillä oleva välimuisti on jaettu ohjelma- ja työvälimuistiin. Harvardilaista arkkitehtuuria käytetään, kun prosessori käsittelee välimuistia. Jos välimuistista ei löydy haettua tietoa, haetaan se ulkoisesta päämuistista, joka taas ei ole jaettu erillisiin ohjelma- ja työmuistiosiin. Ohjelmoijalle arkkitehtuuri näkyy von neumannilaisena ja järjestelmäsovelluksille harvardilaisena arkkitehtuurina.

Mikrokontrollerit voivat käyttää muuteltua Harvardin arkkitehtuuria samasta syystä, kuin se on luotu, eli muistihakujen nopeuttamiseen kohtuullisella monimutkaisuudella sekä työ- ja ohjelmamuistin toteuttamisen sallimiseen toisistaan riippumatta. Mikrokontrollerit yleensä ohjelmoidaan konekielellä, jolloin korkeamman tason ohjelmointikielen tuen puuttuminen on toissijaista, tai järjestelmäkohtaisilla korkeamman tason ohjelmointikielellä, joka käyttää omaa erityistä kääntäjää.

Digitaaliset signaaliprosessorit, DSP, suorittavat yleensä pientä, hyvin loppuun asti hiottua koodia, jossa suorituskyky on tärkein tavoite. Epätavallisen arkkitehtuurin kanssa työskentelyn ongelmat ovat toissijainen huolenaihe suoritusnopeuden rinnalla. Tästä johtuen joillain DSP:llä on useita ohjelmamuistialueita tietyissä osoiteavaruuden osissa, eräänlaisena hyper-harvardilaisena arkkitehtuurina, yksi-käsky-monta-tietoa – prosessointina. (6)

2.4.3. RISC

RISC (Reduced Instruction Set Computer) on tietokoneiden suoritinarkkitehtuurien suunnittelufilosofia, jossa konekielen käskyt on pyritty pitämään yksinkertaisina ja nopeasti vakioajassa suoritettavina. Aiemmin suosittu lähestymistapa oli CISC (Complex Instruction Set Computer), jossa pyritään tekemään käskykannasta mahdollisimman laaja ja monipuolinen ja sitä kautta tehokas.

RISC:stä tuli vallitseva suoritinarkkitehtuurien suunnittelufilosofia 1980-luvun puolivälistä alkaen ja sitä käytetäänkin lähes kaikissa muissa uusissa tietoteknisissä laitteissa paitsi tavallisissa PC- ja Mac-tietokoneissa, joiden suorittimet perustuvat edelleen 1970-luvulla kehitettyyn CISC-tyyppiseen x86-arkkitehtuuriin.

Osa uusimmista x86-suorittimista yrittää hyödyntää RISC-lähestymistavan etuja kääntämällä sisäisesti CISC-komentoja yksinkertaisempaan RISC-tyyliseen muotoon. Tällä tavoitellaan parempaa käskyjen jakautumista rinnakkain suoritettaviin palasiin ja sen kautta nopeampaa suoritusta. Nykyisin käskykannan luonne ei enää ole niin vahvasti sidoksissa suorittimen sisäiseen toimintaan kuin aiemmin, mikä on tehnyt tiukasta jaottelusta RISC- ja CISC-arkkitehtuureihin melko tarpeettoman. (7)

2.5. Sulautetut järjestelmät

Sulautettu järjestelmä (embedded system) on tiettyyn tarkoitukseen tehty tietokonejärjestelmä. Tyypillinen piirre on, että käyttäjän ei tarvitse olla tietoinen laitteen sisällä olevasta tietokoneesta, vaikka sen olemassaolon voi helposti päätellä. Monet järjestelmät ovat suhteellisen pieniä, ja ne on toteutettu mikrokontrollerien avulla.

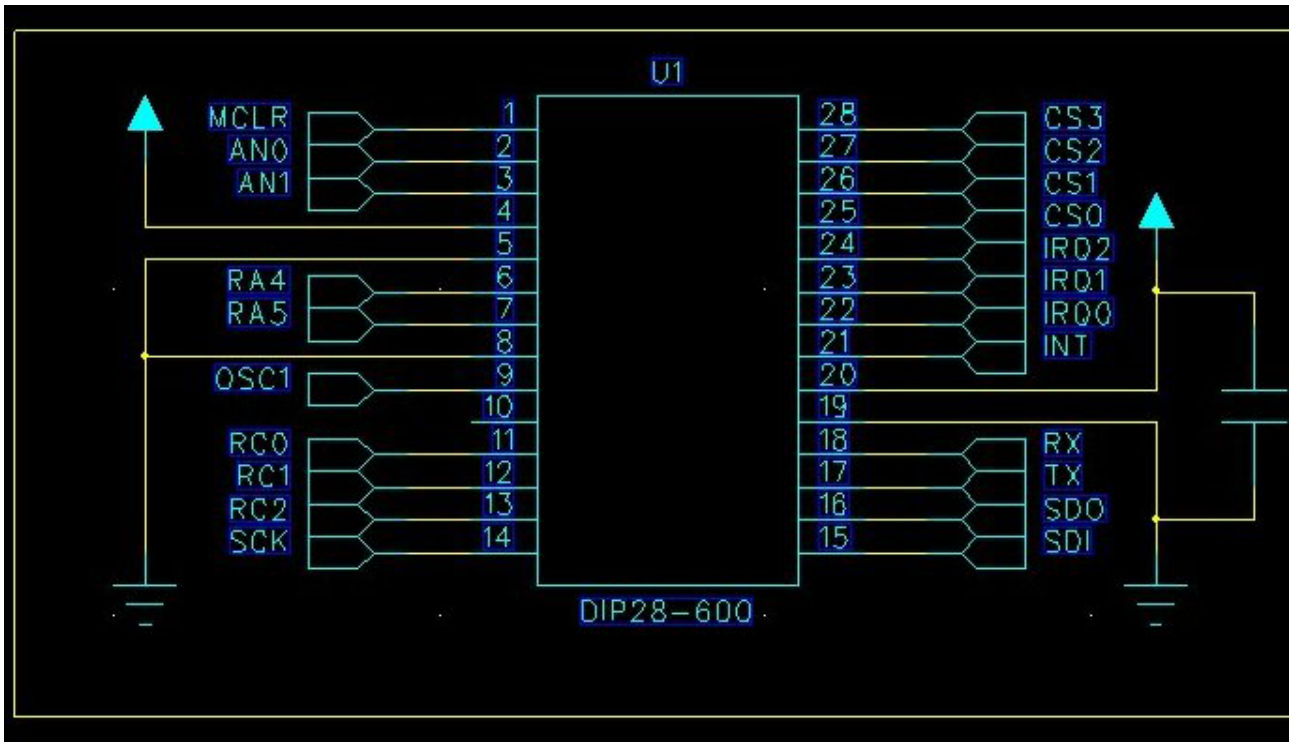
Järjestelmät voivat olla mukana kuljetettavia ja akkukäyttöisiä, mikä asettaa tiukkoja vaatimuksia energian kulutukselle. Vastaavasti suurimmat sulautetut järjestelmät voivat sisältää useita hyvin tehokkaita tietokoneita.

Sulautetuilla järjestelmillä toteutetaan nykyään paljon sellaisia järjestelmiä, jotka toteutettiin aiemmin elektroniikan erilliskomponenteilla. Sulautettuja järjestelmiä on runsaasti automaation eri sovelluksissa (venttiilit, anturit, ohjausjärjestelmät), autoissa (moottorin ohjaus, lukkiintumattomat jarrut, luistonesto, ajotietokone), viihde-elektroniikassa (televisio, CD- ja DVD-soittimet) ja kodinkoneissa (mikroaaltouuni, astian- ja pyykinpesukone). Osa järjestelmistä on sellaisia, joita ei aiemmin ole voitu järkevästi edes toteuttaa (matkapuhelin, GPS-paikannin, pankkiautomaatti).

3. HARJOITUSALUSTA

3.1. Yleistä

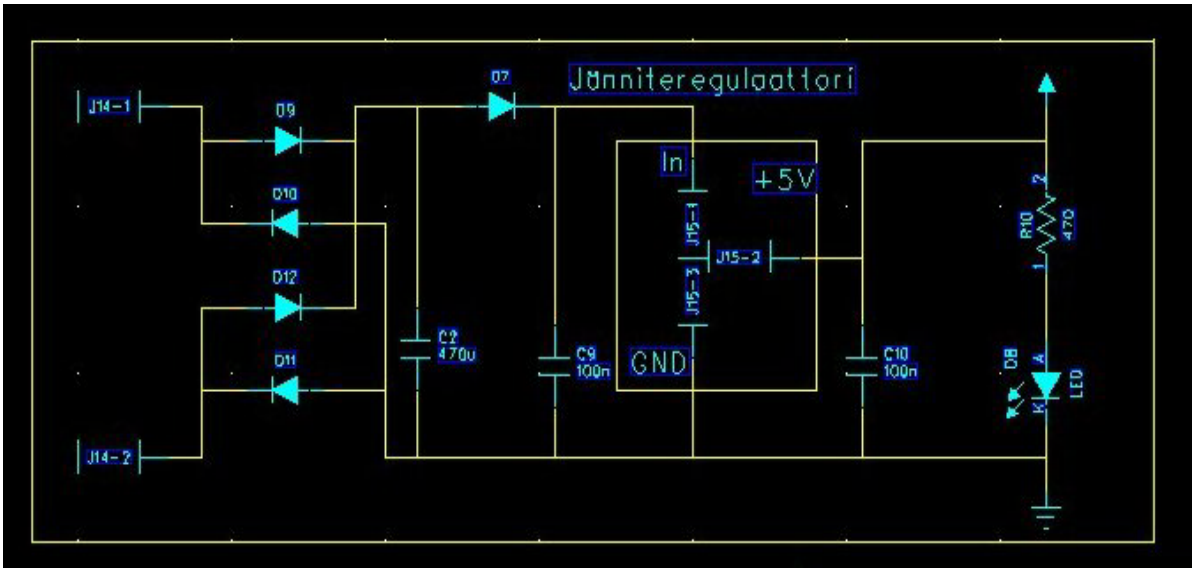
Harjoitusalusta on suunniteltu niin, että se käyttää mahdollisimman monipuolisesti mikrokontrollerin eri ominaisuuksia. Mikrokontrolleriksi on valittu PIC 16F874, monipuolisten ominaisuuksien sekä laajan flash-muistikapasiteetin vuoksi.



Kuva 2. 16F874:n kytkennät

3.2. Käyttöjännite

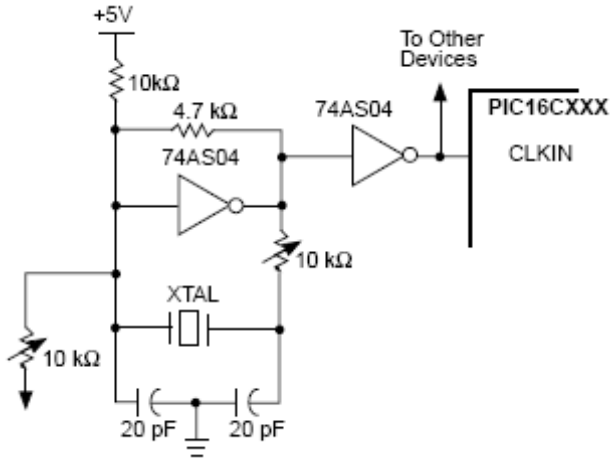
Harjoitusalusta vastaanottaa +9V sekä AC että DC, sillä siinä on sisään rakennettu tasasuuntaus, joka asettaa napaisuuden oikein kaikissa tapauksissa sekä tasoittaa AC:n niin, että siitä voidaan reguloida +5VDC. Harjoitusalustalle ei ole rakennettu ulkoista BOR-piiriä.



Kuva 3. Jänniteregulaattori

3.3. Oskillaattori

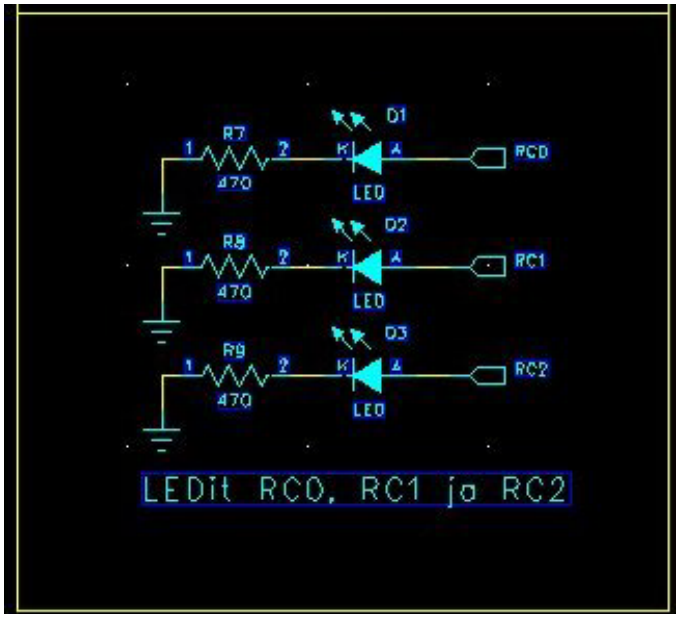
Harjoitusalustalla on rinnakkaisoskillaattoripiiri, jolla kellotetaan harjoitusalustan lisäksi myös alustaan liitetyt lisälaitteet. Jos lisälaite käyttää omaa kellotusta, tulee yhteiskellon jalka jättää kytkemättä lisälaitteella.

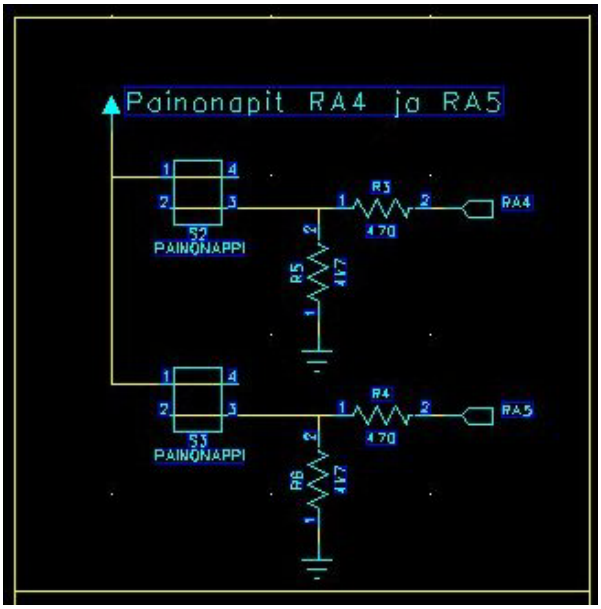


Kuva 4. Sarjaresonaattoripiiri (8)

3.4. Perus-I/O

Harjoitusalustassa on kaksi painonappia ja kolme LEDiä, joilla harjoitellaan perus I/O-toimintojen käyttöä.





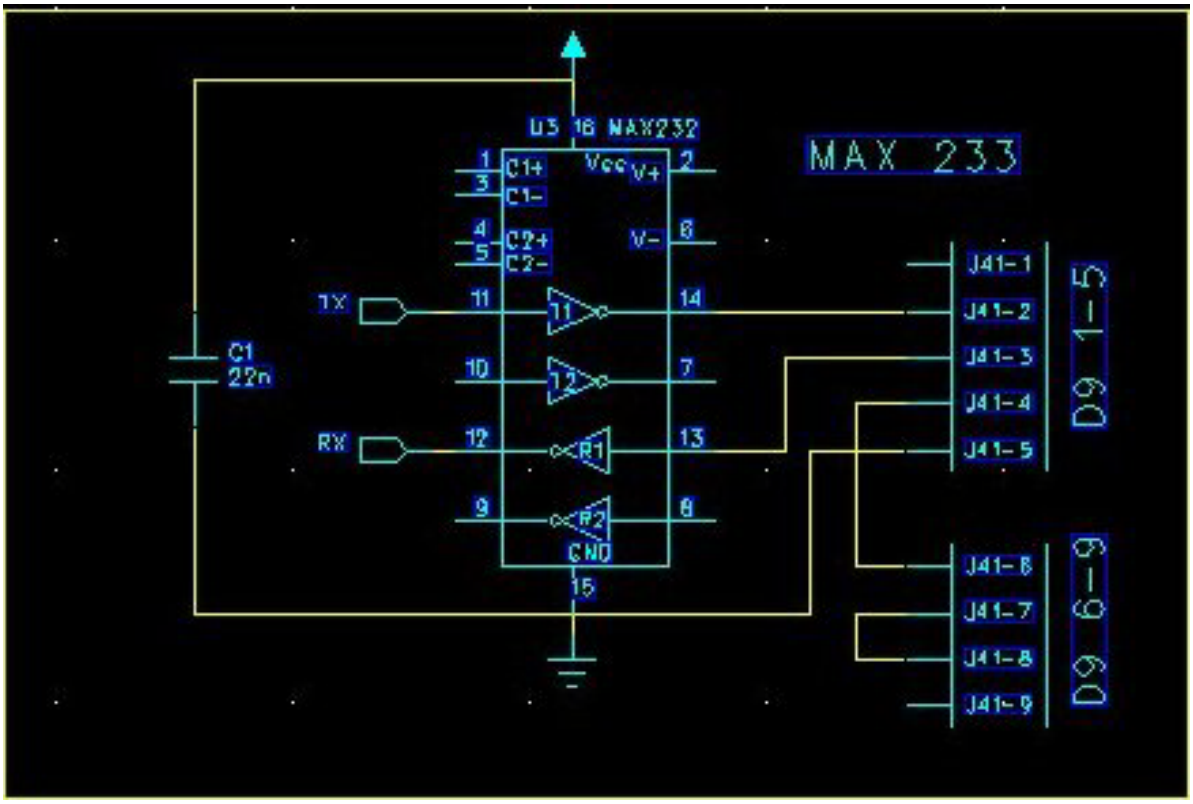
Kuvat 5 ja 6. Painonapit ja LEDit

3.5. Liitännät

Harjoitusalustassa on D9-liitin USARTia varten. Tämän kautta harjoitusalusta voi kommunikoida tietokoneen kanssa. Tämän lisäksi on BNC-liittimet A/D-moduulin ulkoiseksi lähteeksi sekä RJ-45-liittimet lisälaitteita varten.

3.5.1. USART

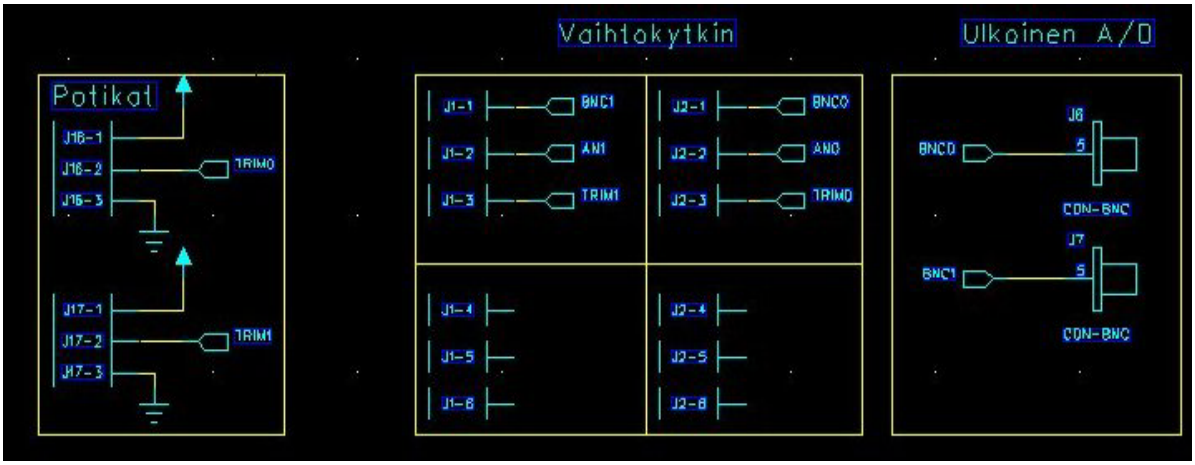
USART käyttää MAX232-väyläajuria, joka ajaa väylän samalle tasolle tietokoneiden käyttämän sarjaväylän kanssa. USART on kytketty niin, että sitä käytetään asynkronisessa tilassa. Tietokoneen sarjaväylän siirtonopeus on 9600 baudia.



Kuva 7. USART

3.5.2. A/D-muunnin

A/D-muunnin käyttää ulkoisena liitäntänä BNC-liitintä, johon voidaan kytkeä tavallinen mittapää, ja käyttää näin A/D-muunnoksen lähteenä mitä tahansa ulkoista lähdettä, jonka jänniteväli ei ylitä +5V. Tämän lisäksi harjoitusalustalla on potentiometri, jota myös voidaan käyttää A/D-muunnokselle lähteenä. Potentiometri on kytketty käyttöjännitteen ja maan välille, jolloin jännitevälin ylitys ei ole mahdollinen. Valinta ulkoisen lähteen ja potentiometrin välillä suoritetaan vaihtokytkimellä.

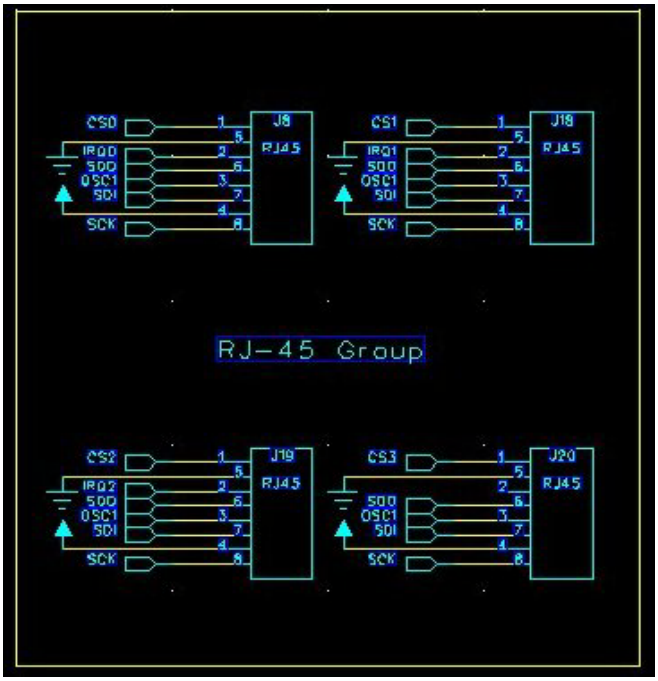


Kuva 8. A/D-muunnin

3.5.3. RJ-45 ja ulkoiset laitteet

RJ-45 kytkee harjoitusalustan lisälaitteisiin, kuten näyttöön tai näppäimistöön. Liitin valittiin kahdesta syystä: Ensinnäkin siinä on tarpeellinen määrä nastoja, jotta kaikki tarvittavat johtimet saadaan kytketyksi ja toiseksi se on yleisesti käytetty liitäntä tietokoneen verkkoyhteydessä, joten kaapeleita on helposti saatavilla, eikä niitä tarvitse itse valmistaa.

RJ-45:llä viedään lisälaitteelle piirinvalinta, keskeytyslinja, käyttöjännite ja maa, yhteiskello sekä SSP-väylä. SSP voidaan kytkeä joko I^2C- tai SPI-tilaan, sillä molemmat käyttävät samoja nastoja vaikkakin eri tarkoitukseen.



Kuva 9. RJ-45

3.6. Näppäimistö

Näppäimistöä ei harjoitusalustaan itseensä liitetty. Jos sen operointia halutaan harjoitella, täytyy se tuoda lisälaitteena. Perinteinen heksanäppäimistö (numerot 0-9, kirjaimet A-F) on helppo toteuttaa yhdellä mikrokontrollerin portilla, josta puolet asetetaan ulostuloiksi ja toinen puoli sisääntuloiksi. Asettamalla yksi linja kerrallaan aktiiviseksi voidaan lukea, onko jotakin, ja mitä, sen rivin näppäintä painettu.

Esimerkki 2.

```
; Näppäimistön luku
; Luetaan heksanäppäimistön painike
; ja lähetetään tieto USART:lla eteenpäin
; TRISB on ’1111 0000’, USART on alustettu
;
main
        clrf            PORTB       ; Tämä tyhjennys suoritetaan ohjelman aluksi
                                    ; Näin varmistutaan siitä, ettei portissa ole ei-toivottuja bittejä
linja_1
        bsf             PORTB, 0    ; Asetetaan 1. linja luettavaksi
        btfsc           PORTB, 4    ; Luetaan ensimmäinen tulo
        call            ruutu11     ; 1. rivin 1. painike
        btfsc           PORTB, 5    ; Luetaan toinen tulo
        call            ruutu12     ; 1. rivin 2. painike
        btfsc           PORTB, 6    ; Kolmas tulo
        call            ruutu13     ; 3. painike
        btfsc           PORTB, 7    ; Ja viimeinen
        call            ruutu14     ; Ja viimeinen
        bcf             PORTB, 0    ; Ja sammutetaan linja 1
linja_2
        bsf             PORTB, 1    ; Asetetaan 2. linja luettavaksi
;
;…toista yllä oleva linja_2:lle, linja_3:lle ja linja_4:lle
;
ruutu11
        movlw           Luku_1      ; Luku_1 on ennalta määritelty luku
        movwf           Out_Buff    ; Out_Buff on ennalta määritelty rekisteri
        goto            transmit    ; siirrytään lähettämään
;
;…toista ruutu12 – ruutu44
;
transmit
        movf            Out_Buff, 0 ; Luetaan lähetettävä luku
        movwf           TXREG       ; ja lähetetään
odota
        btfss           TXIF        ; pollataan lähetyksen valmistumista
        goto            odota
        bcf             TXIF        ; lähetys valmis
        return                      ; palataan pääohjelmaan
;
; return-komento palauttaa pinosta päällimmäisen osoitteen,
; jonka call-komento on sinne sijoittanut
```

3.7. Näyttö

Näyttöä ei ole harjoitusalustaan itseensä liitetty, joten se täytyy tuoda lisälaitteena, jos sellaisen operointia halutaan harjoitella. Perinteinen 7-segmenttinäyttö on helppo toteuttaa yhdellä mikrokontrollerin portilla. Useamman käyttöön tarvitaan lisäksi yksi nasta jokaista 7-segmenttimoduulia kohden. Tätä tarvetta voi pienentää multipleksaamalla moduulinvalinan.

Esimerkki 3.

```
; Kahden 7-segmenttinäytön käyttö
; Näyttöjen sisältö on kirjoitettu rekistereihin Dispdata_1 ja Dispdata_2
; Tiedon vastaanottoon ja käsittelyyn ei tässä esimerkissä puututa
;
; PORTB ohjaa 7-segmenttejä
; PORTA, 0 ja PORTA, 1valitseva näytön, ja ovat nimetty Disp_1 ja Disp_2
;
main  movf    Dispdata_1, 0 ; luetaan ensimmäisen näytön tiedot
      movwf   PORTB
      bsf     Disp_1        ; sytytetään näyttö #1
      call    wait_loop     ; odotusjakso, voidaan tehdä esim. decfsz-goto loopilla
      bcf     Disp_1        ; sammutetaan näyttö
      movf    Dispdata_2, 0 ; luetaan toisen näytön tiedot
      movwf   PORTB
      bsf     Disp_2        ; sytytetään näyttö #2
      call    wait_loop     ; odotusjakso
      bcf     Disp_2        ; sammutetaan näyttö
      goto    main          ; ja aloitetaan alusta
;
; Huomioitavaa on se, että vaikka näyttöjä pidetään päällä vuorotellen
; tapahtuu tämä tuhansia kertoja sekunnissa, ja näytöt näyttävät palavan yhtä aikaa
```

4. 16F874

4.1. Yleistä

Microchipin PIC 16F874 on 40-nastainen mikrokontrolleri, jolla on monipuolinen valikoima erilaisia moduuleja. 16F87X-sarjan mikrokontrollerien erot ovat A/D-moduulien sekä sisäisen muistin määrässä (ks. liite 1). 16F874 valittiin juuri sen takia, että siinä on hieman enemmän nastoja kuin 16F873:ssa, ja muistin määräksi riittää vähempi, kuin 16F876 ja -877–piireissä on. PIC:n mikrokontrollerit perustuvat Harvardin arkkitehtuuriin ja käyttävät RISC-komentokantaa.

4.2. MPLAB

MPLAB on Microchipin oma ympäristö, jolla voidaan kääntää ja simuloida koodia. Ohjelman kääntäjällä on erilaisia ohjelmointia helpottavia työkaluja tarjolla ohjelmoijalle. Tässä käsitellään keskeisimmät niistä.

4.2.1. Määrittelyt

Sekä rekistereitä että yksittäisiä bittejä voidaan käsitellä nimellä osoitemuodon sijaan. Tätä varten ne on nimettävä ennen käyttöä. Rekisteri määritellään seuraavan syntaksin mukaisesti:

```
#define     rekisteri_nimi  rekisteri_osoite
```

Tässä rekisteri_nimi on nimi, jolla rekisteriin viitataan ja rekisteri_osoite on rekisterin konkreettinen osoite muistiavaruudessa, esim.

```
#define     STATUS      0x03
```

Bitti määritellään seuraavan syntaksin mukaisesti:

```
bitti_nimi      equ         rekisteri, bitti
```

Tässä bitti_nimi on nimi, jolla bittiin viitataan, rekisteri on rekisteri, jossa bitti sijaitsee ja bitti on viitattavan bitin numero ko. rekisterissä [0-7], esim.

```
RA0             equ         0x05, 0
```

Bittien määrittelemisessä voidaan myös hyödyntää jo nimettyjä rekistereitä, esim.

```
Z               equ         STATUS, 0
```

Esimerkeissä oletetaan, että rekisterit ja bitit on nimetty dokumenteissa esiintyvällä tavalla ja porteista käytetään myös samoja nimityksiä kuin dokumenteissakin.

4.2.2. Numeroesitys

Numeroesitykset esitetään muodossa $’Syöte’, jossa $ on numeromuodon tunnus ja Syöte on ko. muotoon kirjoitettu numero. Heksadesimaalilla ja desimaalilla on myös vaihtoehtoiset esitysmuodot. Liitteessä 2 on esitetty kaikki esitysmuodot. Kaikkia numeroesitysmuotoja voidaan käyttää kaikissa paikoissa koodia.

4.2.3. Kommentointi

Koodiin voi liittää kommentointeja erottamalla kommentti koodista puolipisteellä (;) esim.

```
movlw       0x23        ; siirretään W:hen 23h
```

Kommentoiminen on yleisen koodin luettavuuden kannalta suositeltavaa, koska se ei kasvata varsinaisen koodin kokoa, ainoastaan lähdetiedoston.

4.2.4. Koodin aloitus ja lopetus

Ohjelman ensimmäinen käsky on:

```
list p=prosessori_tyyppi
```

Siinä prosessori_tyyppi on käytetty mikrokontrolleri, esimerkiksi 16F874. Tämä kertoo kääntäjälle, mille mikrokontrollerille koodi tehdään. Tämän jälkeen tulevat määrittelytiedostot sekä määrittelyt. Sitten asetetaan reset-vektori org-komennolla ja tämän jälkeen seuraa koodi.

Koodi aloitetaan komennolla org, joka asettaa ns. reset-vektorin annettuun osoitteeseen. PCL asetetaan tähän osoitteeseen aina resetin jälkeen. Jos reset-vektoria ei ole asetettu, on se oletusarvoisesti 0x00.

Ohjelmaa kirjoitettaessa on huomioitava, että keskeytysvektori on 0x04, eli jos keskeytykset ovat käytössä ja org on asetettu pienempään arvoon kuin 0x04, viimeistään PCL 0x03 pitää olla goto-komento osoittamassa varsinaiseen ohjelman alkuun, jotta ohjelma toimii halutulla tavalla. Ohjelmakoodin viimeinen komento on aina end. Ohjelma, aliohjelmat sekä keskeytykset on kaikki sijoitettava org- ja end-komentojen väliin.

4.2.5. Rivien nimeäminen sekä goto- ja call-komennot

Rivin syntaksi on seuraavanlainen:

```
Rivin_nimi          komento     ; kommentointi
```

Siinä rivin_nimi on kyseiselle riville asetettu tunniste. Rivien nimeämisestä ei ole itsessään mitään haittaa, sillä ne toimivat kääntäjälle goto- ja call-komentojen linkkeinä. Yletöntä nimeämistä kannattaa välttää, jotta oikeat hyppykohdat erottuvat helposti ja rivien nimeäminen kuvaavasti ei hankaloidu.

Sekä goto- että call-komennot hyppäävät johonkin kohtaa koodia. Tämä kohta on komennon argumenttina oleva rivi. Goto on nimensä mukaisesti (go to = mene paikkaan) ehdoton hyppy ilman paluuta, eli ohjelman suorittamista jatketaan uudesta kohdasta.

Goto-komennolle voidaan antaa myös ”n-hyppyä”-määrä kirjoittamalla se muotoon goto $+n, missä n on hypättävien käskyjen määrä. Goto $ on siis hyppy tälle riville.

Call on”aliohjelmakutsu”, eli hyppy johonkin paikkaan sekä paluuta sieltä joko return- tai retlw-komennolla. Sekä return- että retlw-komennot palaa alkuperäisen koodin, sillä erotuksella, että retlw asettaa samalla W:hen argumenttina olevan arvon. Call-komento sijoittaa nykyisen PCL:n arvon pinoon, josta return- tai retlw-komento sen palauttaa.

4.2.6. Määrittelytiedostot

Määrittelytiedostot ovat tekstitiedostoja, jotka sisältävät rekisterien ja/tai bittien määrittelyt kohdassa 4.2.1 esitetyllä tavalla. Määrittelyt voidaan tehdä myös suoraan ohjelmakoodiin, mutta jos niitä on paljon on ulkoisten tiedostojen käyttö suositeltavaa. Kaikki perusrekisterit sekä -bitit (rekisterit, jotka on dokumenteissa määritelty, sekä niiden määritellyt bitit) kannattaa määritellä dokumentoinnissa käytetyillä nimillä, jolloin niiden käyttö helpottuu ja koodista tulee luettavampaa.

Määrittelytiedosto liitetään varsinaiseen koodiin seuraavan syntaksin mukaisesti:

```
#include        <tiedostonimi>
```

Siinä tiedostonimi on määrittelytiedoston nimi päätteen kanssa, esim.

```
#include        <16f87c.inc>
```

Huom! < ja > merkit tulevat mukaan syntaksiin.

Määrittelytiedostot liitetään ohjelmaan ennen koodia. Niissä määriteltyjä tietoja voidaan käyttää myös toisissa tiedostoissa, esimerkiksi makroissa, jolloin määrittelyt sijoitetaan ennen näiden käyttöä. Muita määrittelytiedostoja voivat olla esimerkiksi omien rekisterien ja/tai bittien määrittelyt tai omaan tiedostoonsa kirjoitetut makrot.

4.2.7. Makrot

Makro on kääntäjän tarjoama työkalu, jolla voidaan korvata pätkä koodia yhdellä komennolla. Makroista on tarkempaa tietoa ”MPASM™ Assembler; MPLINK™ Linker, MPLIB™ Object Librarian User’s Guide” s. 161. Makroja koskee kaikki samat säännöt ja mahdollisuudet kuin itse ohjelmaankin. Makro määritellään seuraavan syntaksin mukaisesti:

```
macron_nimi     macro
                koodia
                koodia
endm
```

Tässä macron_nimi on nimi, jolla makroon viitataan ja endm on makron lopetuskomento. Koodi kirjoitetaan väliin macro – endm (edellä esitetyn ”koodia” riveillä).

Makrot kannattaa kirjoittaa omaan tiedostoon, joka linkitetään koodiin kääntämisvaiheessa (ks. 4.2.6). Huom! makro on kääntäjän työkalu koodin kirjoittamisen helpottamiseksi, ei aliohjelmakutsu. Käännettäessä makro korvataan sen sisältämällä koodilla.

4.2.8. Kääntäminen ja piirin ohjelmointi

Lähdekooditiedosto nimetään asm-päätteiseksi ja käännetään MPLAB-kääntäjällä. Tämä tuottaa hex-päätteisen tiedoston, joka sisältää käännetyn ohjelman. Tämä tiedosto ohjelmoidaan piirille käyttäen MPASM-ohjelmaa. Kääntämisvaihe tuottaa myös muita tiedostoja, joista mainitsemisen arvoinen on err-tiedosto. Tämä sisältää kääntämisvaiheessa havaitut virheet.

4.3. Muisti

Muistia tarvitaan kahteen asiaan: ohjelmakoodille sekä ohjelman muuttujia varten. PIC:n muisti on jaettu kahteen osaan: ohjelmamuisti (ROM) ja datamuisti (RAM). Näistä datamuisti voidaan jakaa vielä kahteen: erityisrekisterit (SFR) sekä yleinen datamuisti.

Ohjelmamuisti on kiinteää muistia, jonka kirjoittamiseen tarvitaan ulkopuolisia työkaluja. Se säilyttää tilansa, kunnes se kirjoitetaan uudelleen. Ohjelmamuisti on sivutettu yhdestä neljään osaan, joista jokaiseen mahtuu 2048 (2K) 14-bittistä käskyä. Näistä aina yhtä 2K sivun sisältöä voidaan osoittaa suoraan PCL:n avulla. Sivujen, eli muistin määrä on laitekohtainen, ja se pitää tarkistaa ko. laitteen datalehdestä (ks. liite 3 ohjelmamuistiorganisaatiokartta).

Pinorekisteri on 8 tasoa syvä, ja sitä käytetään call-käskyn sekä keskeytyksen yhteydessä tallettamaan sen hetkinen PCL. Mitään muita järjestelmärekistereitä ei talleteta pinoon, joten jos niitä käsitellään jonkin keskeytyksen yhteydessä, on syytä ottaa kopio tärkeimmistä rekistereistä. Return, retfie ja retlw palauttavat pinosta edellisen PCL:n.

4.3.1. Käyttömuisti

Käyttömuisti on sekä ohjelman muuttujia että erityisrekistereitä varten varattua tilaa. Käynnistyksen yhteydessä käyttömuisti palaa aina oletustilaan ja kaikki sinne talletettu tieto kadotetaan.

Käyttömuisti on jaettu kahteen osaan:

1. Erityisrekisterit (SFR, Special Function Register)
2. Yleinen datamuisti (GPR, General Purpose Register)

SFR sisältää kaikki laitteen ohjaamiseen liittyvät asetukset ja liput ja GPR on ohjelman käytettävissä oleva muistiavaruus. Vaikka SFR:t voivat toimia periaatteessa myös GPR:nä (esimerkiksi jos moduulia, jolle rekisteri on implementoitu, ei ole laitteessa) ei tämä ole yleisesti hyvä tapa, ja jopa implementoimaton SFR kannattaa jättää käyttämättä laitteen vakaan toiminnan takaamiseksi.

Käyttömuisti on neljässä pankissa, joiden osoittamiseen käytetään 9-bittistä osoitetta. Jokaisessa pankissa on 32 tavua erityisrekistereille ja 96 yleiskäyttöistä muistitavua. Laitteen muistin määrästä riippuen osa pankkien yleiskäyttömuistista voi olla peilattu toisiin pankkeihin. Muistin määrä on laitekohtainen ja tulee tarkistaa ko. laitteen datalehdestä.

Osoitusmuotoja on kaksi: suora ja epäsuora osoitus. Käskyt käyttävät suoraa osoitusta, jossa käskylle annetaan käsiteltävän rekisterin osoite. Tällöin pankin tulee olla valittuna oikein, koska rekisterin osoite koostuu pankinvalinta biteistä sekä käskyn sisältämästä 7-bittisestä osoitteesta: [RP1:RP0:7-bit addr]. Käsiteltävä pankki [0-3] voidaan valita asettamalla bitteihin RP0 ja RP1 pankkia vastaava binääriarvo [00-11].

Esimerkki 4.

```
; Suora osoitusmuoto
        movf        OMA1, 0     ; siirretään työrekisteriin rekisterin OMA1 arvo
        addwf       OMA2, 0     ; lisätään työrekisteriin rekisterin OMA2 arvo,
                                ; tulos tallennetaan työrekisteriin
        movwf       TULOS       ; siirretään työrekisterin sisältö rekisteriin TULOS
```

Epäsuora osoitus käyttää kahta SFR:ää: FSR ja INDF. FSR (File Select Register) sisältää osoitteen, jota voidaan käsitellä INDF (Indirect File) -rekisterin kautta. Molemmat ovat osoitettavissa kaikissa muistipankeissa. Koska FSR on 8-bittinen rekisteri, voidaan epäsuoralla osoittamisella osoittaa kahta pankkia samaan aikaan. Kahden pankin lohko ([0,1] tai [2,3]) voidaan valita muuttamalla IRP-bittiä.

Esimerkkejä 5.

```
; Epäsuora osoitusmuoto
        movf    TAUL_ALKU, 0 ; siirretään työrekisteriin rekisterin TAUL_ALKU arvo
        addwf   OFFSET, 0    ; lisätään työrekisteriin rekisterin OFFSET arvo,
                             ; tulos tallennetaan työrekisteriin
        movwf   FSR          ; siirretään työrekisterin arvo rekisteriin FSR (valitaan kohde)
        movf    INDF, 0      ; siirretään rekisterin INDF arvo työrekisteriin, eli oikeammin
                             ; työrekisteriin siirretään FSR:n osoittaman rekisterin arvo
```

4.3.2. EEPROM

Data EEPROM on pysyvää ohjelmamuistia, jota voidaan käsitellä ainoastaan erikoisrekisterien kautta. Normaalisti ohjelmat eivät tarvitse tällaista muistia, vaan se on ennemminkin asetusten tai mittaustulosten tallentamista varten niin, että niitä päästään käsittelemään käyttöjännitteiden katkettua. EEPROMin käsittelyä varten on neljä rekisteriä:

1. EECON1
2. EECON2
3. EEDATA
4. EEADR

EECON1 ja EECON2 ovat hallintarekisterejä, joilla asetetaan kirjoitussuoja päälle/pois, aloitetaan luku/kirjoitus jne. EEPROMia käsitellään epäsuoralla osoituksella. EEADR on käsiteltävän EEPROM-paikan 8-bittinen muistiosoite ja EEDATA sisältää joko luetun tai kirjoitettavan datan. Laitteiden sisältämän EEPROMin määrä on laitekohtainen, ja se pitää tarkistaa ko. laitteen datalehdestä. Kirjoittaminen EEPROMiin tapahtuu seuraavalla tavalla:

1. Asetetaan WREN-bitti (EECON1, 2)
2. Kirjoitetaan 0x55 EECON2:een
3. Kirjoitetaan 0xAA EECON2:een
4. Kirjoitetaan haluttu data EEDATA:aan
5. Asetetaan WR-bitti (EECON1, 1)

EECON2-rekisterin ainoa käyttö on estää vahingossa EEPROMiin kirjoittaminen. Myös keskeytykset suositellaan poistettavan käytöstä EEPROMin kirjoittamisen ajaksi, jotta kirjoitusvaihe ei häiriinny ja virheiltä vältyttäisiin.

4.4. I/O-rajapinta

PIC:n perusrajapintana toimivat ohjelmoitavat I/O-portit, joiden toiminta määritellään TRIS-rekistereissä. 1 = Input ja 0 = Output, jolloin esimerkiksi ’0000 1111’ määrittelisi puoli porttia (7-4) ulostuloiksi ja toisen puoli (3-0) sisääntuloiksi.

Jos nasta on määritelty sisääntuloksi, määrittää tilan ulkopuolinen laite, kuten painonappi tai toinen mikrokontrolleri. Tila voidaan tarkastaa joko pollaamalla PORT-rekisteri (tai sisääntulon bitti ko. rekisterissä) tai mahdollisesti keskeytyksen avulla. Sisääntulolla halutaan joko valvoa jotakin tilaa tai tuoda jotain muuta informaatiota mikrokontrollerille.

Esimerkki 6.

```
; Sisääntulon tarkistaminen yhtä nastaa pollaten
alku    btfsc       PORTA, 0    ; tarkistetaan onko RA1:een kytkettyä painonappia painettu
        goto        alku        ; painonappi on 0-aktiivinen, eli painallus ajaa tilaksi 0
; Sisääntulon tarkistaminen Z-lipun avulla
        bsf         Z           ; resetoidaan Z-lippu (eli asetetaan se 1:ksi)
alku    movwf       PORTB       ; siirretään PORTB työrekisteriin
        btfsc       Z           ; jos jonkin PORTB:n nastan tila on 1, on Z-lippu 0
        goto        alku        ; jolloin hypätään alkuun palaamisen yli
```

Jos nasta on määritelty ulostuloksi, määrittää mikrokontrolleri tilan. Tällöin PORT-rekisteriin (tai ulostulon bittiin ko. rekisterissä) asetetaan haluttu arvo. Ulostulon tarkoitus on järjestelmän ohjaus, esimerkiksi transistorin avulla ohjataan moottoria, tai tiedonsiirto piirin ulkopuoliselle laitteelle.

Esimerkki 7.

```
; Asetetaan yksittäisen nastan ulostulo
        bsf         RA0
; Asetetaan koko portin ulostulo
        movlw       0x47        ; binäärimuodossa 0100 0111
        movwf       PORTB       ; eli PORTB:n nastat 6, 2, 1 ja 0 ovat aktiivisessa tilassa
```

Tyypillisiä I/O-rajapinnan erityistapauksia ovat esimerkiksi A/D-moduuli sekä USART. Tällöin osa I/O-kannoista muunnetaan kyseisen moduulin käyttöön tarkoitetuiksi nastoiksi, kuten referenssijännitteen sisääntulo tai lähetys/vastaanotto, eikä näitä nastoja voi tässä tapauksessa käyttää perus I/O:n tarpeisiin.

4.5. Sarjaväylä

Sarjaväylä on tiedonsiirtoväylä kahden tai useamman laitteen välillä. Tieto siirretään yhtä johdinta käyttämällä. Tämän lisäksi käytössä voi olla myös kelloväylä yhteiskelloa varten. Lisäksi tiedonsiirto voi tapahtua kahdella johtimella niin, että toisella tietoa lähetetään ja toisella vastaanotetaan.

4.5.1. USART

USART on ohjelmoitava sarjaväylä, joka vastaa PC-tietokoneen sarjaväylää. Se voidaan asettaa ohjelmallisesti joko synkroniseen (yhteiskellotettuun) tai asynkroniseen toimintaan sekä määrittelemään baud rate eli toimintanopeus.

USART:n asetukset ovat seuraavissa rekistereissä: TXSTA, RCSTA ja SPBRG. Ensin valitaan joko synkroninen tai asynkroninen tila. Koska käytettävien nastojen määrä on kaksi, toimii laite joko half- (tiedonsiirto ja kello, synkroninen tila) tai full-duplex (lähetys ja vastaanotto, asynkroninen tila) -tilassa. Synkronisessa tilassa täytyy vielä määrittää isäntä- (master) ja orja (slave) -laitteet. Isäntä antaa kellon tiedonsiirrolle. Asynkronisessa tilassa kaikkien laitteiden täytyy käyttää samaa nopeutta.

Asynkronisessa tilassa on valittavana high- tai low-speed tila. Huolimatta siitä, että näiden ainoana erona on maksiminopeus, saattaa high-speed tilan käyttö olla kannattavaa pienilläkin nopeuksilla. Synkronisessa tilassa vastaavia vaihtoehtoja on vain yksi. Nopeus saadaan kaavalla baud = Fosc / (n(x+1)), missä baud on haluttu nopeus, Fosc on kellotaajuus, n on asetuksista määräytyvä arvo ja x on SPBRG-rekisteriin (Serial Port Baud Rate Generator) kirjoitettu arvo.

Asynkronisessa tilassa on käytettävissä kaksi rekisteriä siirtoa varten: TXREG ja RCREG. TXREG:iin kirjoittamalla lähetetään tietoa ja vastaan otettu tieto on luettavissa RCREG:stä. Sekä lähetyksen tilaa (lähettää/valmis) että vastaanoton tilaa (vastaanotettu / ei vastaanotettu tai vastaanotto käynnissä) voidaan seurata pollaamalla kyseisiä keskeytyslippuja tai ko. keskeytysten avulla.

Synkronisessa tilassa toiminta on edellä kerrotun kaltaista seuraavilla eroilla: lähetystä ja vastaanottoa ei voi käyttää samaan aikaan sekä isäntälaite voi valita joko sanan mittaisen tai jatkuvan vastaanoton väliltä, orjalaitteella valinta tehdään lähettämisen ja vastaanottamisen välillä; vastaanotto on aina jatkuva.

9-bittinen lähetys/vastaanotto on myös mahdollinen tilasta riippumatta. Nämä voidaan valita erikseen niin, että joko lähtö tai vastaanotto tai molemmat ovat 9-bittisiä, jos niin halutaan. Tällöin 9. bitti kirjoitetaan TXSTA- tai RXSTA-rekisteriin.

Esimerkki 8.

```
; Asynkroninen USART
; Alustus
; Kellotus tapahtuu 4MHz kiteellä, baud rate on 9600
        movlw       0x04        ; 0000 0100
        movwf       TXSTA       ; asynkroninen 8-bittinen siirto valittu
        movlw       0x80        ; 1000 0000
        movwf       RCSTA       ; sarjaportti käytössä, ei-jatkuva vastaanotto

; baud rate 9600, n=16 (arvo datalehdestä), kello 4MHz
; 9600 = 4000000/(16(x+1)) => x=25
; tarkka baud rate on 9615, jolloin virhettä on alle 1%

; Vastaanotto
; Piirin asetuksista löytyy seuraava:
        bsf         GIE         ; sallitaan kaikki keskeytykset (rekisterissä INTCON)
        bsf         PEIE        ; sallitaan laitteiston keskeytykset (rekisterissä INTCON)
        bsf         RCIE        ; sallitaan vastaanoton keskeytys (rekisteri laitekohtainen)
; Ohjelmakoodi
alku
        goto        alku        ; odotetaan vastaanotettavaa tietoa
; tapahtuu keskeytys, keskeytyskäsittelijä selvittää että sarjaväylän vastaanotto on tapahtunut
vastaanotto
        movf        RCREG, 0    ; siirretään tieto puskurista työrekisteriin
        movwf       TIETO       ; ja sieltä edelleen tallennuspaikkaan
        bcf         RCIF        ; nollataan keskeytys
        retfie                  ; ja palataan takaisin ohjelmakoodiin
; Keskeytyksen sijaan voi RCIF-lipun pollata, lippu asettuu vaikka keskeytys ei olisikaan sallittu
; Lähetys
; Piirin asetuksista löytyy seuraava:
        bsf         GIE         ; sallitaan kaikki keskeytykset (rekisterissä INTCON)
        bsf         PEIE        ; sallitaan laitteiston keskeytykset (rekisterissä INTCON)
        bsf         TXIE        ; sallitaan lähetyksen keskeytys (rekisteri laitekohtainen)
; Ohjelmakoodi
alku
        goto        alku        ; odotetaan lähetyksen aloittamista
; tapahtuu keskeytys, keskeytyskäsittelijä selvittää, että sarjaväylän lähetys aloitetaan
lahetys
        movf        TIETO, 0    ; siirretään lähetettävä data työrekisteriin
        movwf       TXREG       ; ja sieltä edelleen lähetyspuskuriin
        bsf         TXEN        ; ja käynnistetään lähetys
        retfie                  ; palataan takaisin ohjelmaan
; kun lähetys on valmis, tapahtuu uusi keskeytys, ja TXIF-lippu asettuu
; tiedetään että lähetyspuskuri on tyhjä, ja uuteen lähetykseen ollaan valmiita
```

4.5.2. SSP

Synkroninen sarjaportti (Synchronous Serial Port) on tarkoitettu muiden oheislaitteiden tai mikrokontrollerien kanssa kommunikoimiseen, ja sillä on kaksi toimintatilaa: SPI (Serial Peripheral Interface) tai I^2C (Inter-Integrated Circuit).

SPI on isäntä-orja periaatteella toimiva sarjaväylä. Isäntälaite kellottaa väylää. Jos useampi orja-laite kytketään väylään, on käytettävä jonkinlaista piirinvalintaa. SPI käyttää kolmea linjaa: SDI (Serial Data In), SDO (Serial Data Out) ja SCK (Serial ClocK). Isännän SDO kytketään orjan SDI:hin ja toisinpäin. Isäntä kellottaa SCK-väylällä orjaa. Orjalaitteelle on myös mahdollista määrittää SS(i)-nasta, jota käytetään piirin valintaan. Vaihtoehtoisesti tämän voi myös ohjelmoida esim. ulkoisesta keskeytyksestä.

I^2C on osoitteellinen, moni-isäntäinen kilpavarausverkko. Tämä tarkoittaa sitä, että useampi isäntä-laite voidaan kytkeä samaan verkkoon eikä erillistä piirinvalintajärjestelmää tarvita, sillä jokaisella orja-laitteella on oma yksilöllinen osoitteensa, jolla sitä voidaan kutsua. I^2C käyttää kahta linjaa: SDA (Serial Data) ja SCK (Serial ClocK). Verkko varataan syöttämällä kellosignaalia SCK-väylään. Jos törmäystä ei havaita, aloitetaan tiedon siirto syöttämällä SDA-väylään ensin kohdelaitteen osoite ja sitten data.

SSP:n toimintatila valitaan SSPSTAT-rekisterillä ja sitä hallitaan SSPCON-rekisterillä. SPI-tilassa I/O asetetaan seuraavalla tavalla: SDI sisään, SDO ulos, SCK (isäntä) ulos tai (orja) sisään, SS(i) (orja) sisään. Myös asetetaan kellon polariteetti, sisään tulevan datan näytevaihe, kellonopeus (isäntä) ja SS(i)-tila (orja). I^2C-tilassa sekä SDA että SCK asetetaan sisään. SSPADD-rekisteriin asetetaan laitteen osoite. 10-bittisessä tilassa ylempi tavu on muotoa 1111 0 A9 A8 0, missä A9 ja A8 ovat osoitteen kaksi ylintä bittiä.

SSPBUF-rekisteri (SSP BUFfer) toimii sekä lähetyksessä että vastaanotossa. Sitä ei voi suoraan osoittaa, vaan siihen käytetään SSPSR-rekisteriä (SSP Shift Register). Kummassakin tapauksessa (SPI, I^2C) jos orja haluaa lähettää jotain, on sen saatava lupa isännältä. Tätä tarkoitusta varten voidaan käyttää ulkoista keskeytystä tai isäntä voi pollata orjia.

I^2C tilassa lähetetään ensin joko data tai osoite. Tämä valitaan asetuksista, ja kaikkien tässä väylässä olevien laitteiden täytyy käyttää samaa tilaa. Oletusarvoisesti osoite annetaan ensimmäisenä. Jos osoite täsmää, lähettää laite automaattisesti ACK(i) pulssin. SPI-tilassa orja-laitteet voivat olla joko sarjaan tai rinnan kytkettyjä. Sarjaan kytkettynä orja-laitteet vastaavat yhtä laitetta, rinnan kytkettynä laitteille täytyy olla jokin piirin valinta tai vastaava valintajärjestelmä. Yleensä laitteet kytketään rinnan. Tällöin voidaan kytkeä eri laitteita samaan piiriin.

4.6. Rinnakkaisväylä

Rinnakkaisväylä on kahden tai useamman laitteen välille tarkoitettu tiedonsiirtoväylä, jolla tieto siirretään useita johtimia, yleensä 8, käyttämällä. Tämän lisäksi käytössä on yleensä myös luku- (RD(i)), kirjoitus- (WR(i)) ja piirinvalintaväylät (CS(i)). Rinnakkaisväylässä tieto siirretään väylän levyisinä purskeina.

Rinnakkaisväylä on 8-bittisissä PIC-mikrokontrollerilaitteissa orja-tilassa (PSP; Parallel Slave Port), ja sitä käyttää jokin ulkopuolinen laite. Piiri valitaan ajamalla CS(i)-linja alas, ja tämän jälkeen tieto joko luetaan piiriltä tai uusi tieto syötetään piirille. Rinnakkaisväylä alustetaan asettamalla TRISE-rekisterin PSPMODE (bitti 4).

Laitteelle kirjoittaminen tapahtuu ajamalla CS(i)- ja WR(i)-linjat alas. Kun toinen näistä linjoista nousee taas ylös, asettuu IBF-lippu (Input Buffer Full) ilmaisemaan, että tieto on kirjoitettu puskuriin. Laitteelta lukeminen tapahtuu ajamalla CS(i)- ja RD(i)-linjat alas. OBF nollataan ilmaisemaan, että tieto on luettu puskurista. Sekä luku että kirjoitus käyttävät molemmat PSPIF-lippua ilmaisemaan keskeytystä.

Esimerkki 9.

```
; Alustetaan PSP
        movlw       0x17        ; 0001 0111
        movwf       TRISE       ; RD, WR ja CS sisääntuloiksi, käynnistetään PSP
        bsf         PSPIE       ; sallitaan PSP-keskeytys (rekisteri laitekohtainen)
; Piirin asetuksista löytyy seuraava:
        bsf         GIE         ; sallitaan kaikki keskeytykset (rekisterissä INTCON)
        bsf         PEIE        ; sallitaan laitteiston keskeytykset (rekisterissä INTCON)
; Ohjelmakoodi
alku
        goto        alku        ; odotetaan jotain tapahtuvaksi
; tapahtuu keskeytys, keskeytyskäsittelijä selvittää että väylään on kirjoitettu
vastaanotto
        movf        PORTD, 0    ; siirretään tieto puskurista työrekisteriin
        movwf       TIETO       ; ja sieltä edelleen tallennuspaikkaan
        bcf         PSPIF       ; nollataan keskeytys
        retfie                  ; ja palataan takaisin ohjelmakoodiin
; Keskeytyksen sijaan voi PSPIF-lipun tai IBF-lipun pollata,
; lippu asettuu vaikka keskeytys ei olisikaan sallittu
; kun puskuriin on kirjoitettu, tapahtuu uusi keskeytys, ja PSPIF-lippu ja IBF-lippu asettuvat
; tiedetään että vastaanottopuskurissa on taas tietoa
; Lähetys
; tapahtuu keskeytys, keskeytyskäsittelijä selvittää, että lähetyspuskuri on tyhjä
lahetys
        movf        TIETO, 0    ; siirretään lähetettävä data työrekisteriin
        movwf       PORTD       ; ja sieltä edelleen lähetyspuskuriin
        bcf         PSPIF       ; nollataan keskeytys
        retfie                  ; palataan takaisin ohjelmaan
; kun lähetys on valmis, tapahtuu uusi keskeytys, ja PSPIF-lippu asettuu ja OBF-lippu nollautuu
; tiedetään että lähetyspuskuri on tyhjä, ja uuteen lähetykseen ollaan valmiita
```

4.7. A/D-muunnin

A/D-muunnin (analog-to-digital) muuttaa otetun analoginäytteen tietyllä resoluutiolla digitaaliseen esitysmuotoon. Laitteessa voi olla 8- tai 10-bittinen A/D-muunnin.

A/D-muuntimen asetukset ovat seuraavissa rekistereissä: ADCON0 ja ADCON1. ADCON0:lla hallitaan A/D-muuntimen toimintaa ja ADCON1:llä asetetaan nastojen toimintatila (analogitulo, digitaali I/O tai jännitereferenssi [AN2, AN3]). I/O-nastoista voidaan ohjelmoida 1-8 analogisisääntuloa. Laitteiden sisältämän tulojen määrä ja laatu on laitekohtainen, ja se pitää tarkistaa ko. laitteen datalehdestä.

Analogisisääntulojen valinta tapahtuu taulukoiduista arvoista sekä asettamalla nastat TRIS-rekisteristä sisääntuloiksi. Samalla valitaan, käytetäänkö ulkoista referenssijännitettä vai laitteen käyttöjännitettä. Jos valitaan ulkoinen referenssijännite, muutetaan yksi tai kaksi nastaa referenssinastoiksi. Analogisisääntulot sekä referenssinastat asetetaan sisään.

A/D-muunnos tapahtuu kolmessa osassa:

1. Näytteenotto (Sample & Hold)
2. A/D-muunnos
3. Muunnos valmis, luetaan tulos

Näytteenotto käynnistyy, kun jokin käytössä olevista analogituloista valitaan. Minimiaika näytteenotolle on laskettavissa datalehdessä esitetyn kaavan mukaan. A/D-muunnos käynnistyy asettamalla GO/DONE(i)-bitti.

Kun muunnos on valmis, nollautuu GO/DONE(i)-bitti ja ADIF-lippu asettuu. Näitä voi pollata tai vaihtoehtoisesti käyttää keskeytystä, jotta tiedetään, koska muunnos on valmis. Tämän jälkeen tulos on luettavissa 8-bittisen kohdalla yhdestä (ADRES) tai 10-bittisen kohdalla kahdesta (ADRESL, ADRESH) rekisteristä.

Esimerkki 10.

```
; 10-bittisen A/D-muuntiment käyttö
; Alustetaan A/D
; 4MHz kello, minimi näytteenottoajaksi ”laskettu” 20us
        movlw       0x40        ; 0100 0000
        movwf       ADCON0      ; Asetetaan käännösaika
        movlw       0x25        ; 0010 0101, Alimmat 6 bittiä ADRESL-rekisterissä ovat 0,
        movwf       ADCON1      ; valitaan ulkoinen referenssi+, sekä 2 A/D-sisääntuloa
        bsf         ADIE        ; Asetetaan keskeytys

; Piirin asetuksista löytyy seuraava:
        bsf         GIE         ; sallitaan kaikki keskeytykset (rekisterissä INTCON)
        bsf         PEIE        ; sallitaan laitteiston keskeytykset (rekisterissä INTCON)

; Ohjelmakoodi
alku
        movlw       0xC3        ; asetetaan maski työrekisteriin
        andwf       ADCON0, 1   ; asetetaan AN0, pysäytetään AD-muunnos
        movlw       0x14        ; alustetaan 20 kellojaksoa (20us 4MHz kellolla)
        movwf       LASKURI     ; ja alustetaan laskuri
odotus
        incfsz      LASKURI     ; odotetaan
        goto        odotus      ; ja odotetaan kunnes rekisteri menee ympäri
        bsf         GO          ; ja aloitetaan AD-muunnos
muunnos
        btfsc       GO          ; odotetaan muunnoksen valmistumista
        goto        muunnos     ; kun muunnos on tehty tapahtuu keskeytys
        goto        alku        ; ja kun keskeytyksestä palataan mennään taas alkuun

; tapahtuu keskeytys, keskeytyskäsittelijä selvittää että AD-muunnos on suoritettu
admuunnos
        movf        ADRESH, 0   ; siirretään ylemmät bitit työrekisteriin
        movwf       HI_BIT      ; ja sieltä edelleen tallennuspaikkaan
        movf        ADRESL, 0   ; siirretään alemmat bitit työrekisteriin
        movwf       LO_BIT      ; ja sieltä edelleen tallennuspaikkaan
        bcf         ADIF        ; nollataan keskeytys
        retfie                  ; ja palataan takaisin ohjelmakoodiin
; Keskeytyksen sijaan voi ADIF-lipun tai GO-lipun pollata
; ADIF asettuu vaikka keskeytys ei olisikaan sallittu, GO nollautuu, kun muunnos on valmis
```

4.8. Ajastin-moduuli

Ajastimesta on tärkeää muistaa, että se laskee työjaksoja, ei reaaliaikaa. Ajastimelle yksi työjakso on aina samanmittainen, kellottaa piiriä sitten RC-värähtelijä tai korkeataajuinen kide. Toinen asia muistettava asia on, että työjaksoon kuluu 4 kellojaksoa. Tämä käsitellään tarkemmin kohdassa 4.10.1, oskillaattori. Kaikessa yksinkertaisuudessaan ajastimet ovat rekistereitä, jotka kasvavat PC:n kanssa samaan tahtiin ja mahdollisesti aiheuttavat ylivuodon sattuessa keskeytyksen.

4.8.1. Timer-moduulit

Kaikissa piireissä on yhdestä kolmeen Timer-moduulia. Lohkokaaviokuvat Timer1:stä ja Timer2:sta ovat liiteessä 4

Taulukko 1: Ajastinmoduulit

Ajastin	Koko	Syöte	Skaalaus	Ulostulo	Muuta
Timer0	8-bit	Fosc Ext	1:1–1:256 Huom! WDT:llä sama esiskaalainta!	Int	Ajastin on aina käynnissä
Timer1	16-bit	Fosc Ext	1:1-1:8	Int	Käytetään sekä kaappaus että vertailu –toiminnoissa, toimii valmiustilassa (Ext)
Timer2	8-bit	Fosc	Esi: 1:1, 1:4, 1:16 Jälki: 1:1 – 1:16	SSP Jälkiskaalaus (Int)	Kellottaa SSP:n, Kellottaa PWM-generaattorin

Fosc – oskillaattori, Ext – ulkoinen lähde, Int – Keskeytys, SSP – Synch. Serial Port

Ajastimet käyttävät yhtä rekisteriä asetuksille (OPTION_REG, T1CON, T2CON), yhtä rekisteriä ajastimelle (Timer0 ja 2, kahta Timer1:lle) sekä mahdollisesti muita rekisterejä esimerkiksi keskeytykselle tai keskeytyslipulle.

Esimerkki 11.

```
; Timer0-ajastimen alustaminen
; Huom!
; On suositeltavaa, että TMR0-rekisteri tyhjennetään
; sekä keskeytykset estetään alustamisen ajaksi
; Näin estetään mahdollisen keskeytyksen tapahtuminen ennen aikojaan
; tämä voisi nimittäin sotkea halutun toiminnan
        movlw       0xC3        ; 1100 0011, PortB pull-up estetty,
; keskeytys nousevalla pulssilla
        movwf       OPTION_REG; Timer0 toimii sisäisellä kellolla, esiskaalaus 1:16
;
; jos keskeytystä ei aseteta, joudutaan Timer0 pollaamaan
;
timer0_odotus
        btfss       T0IF        ; tarkastetaan, onko keskeytyslippu asettunut
        goto        timer0_odotus ; jos ei, jatketaan odottamista
;
; ylivuoto on tapahtunut
;
        bcf         T0IF        ; kuitataan keskeytys
```

Ajan kaappaaminen Timer1:n rekistereistä voi tuottaa joskus ongelmia, sillä 8-bittinen rekisteri voi ylivuotaa lukemisen aikana. Kirjoittamisen ajaksi suositellaan ajastimen sammuttamista. Kirjoittaminen on kyllä mahdollista ajastimen käydessä, mutta tällöin arvo saattaa olla jotain muuta, kuin mitä on toivottu. Lukeminen ajastimen käydessä saattaa olla ongelmallista, sillä 8-bittiset rekisterit voivat ylivuotaa lukemisen aikana, kuten seuraavasta esimerkistä käy ilmi.

Taulukko 2: TMR1:n lukemisen ongelma

TMR1	Järjestys 1		Järjestys 2	
	Tapahtuma	TMPH:TMPL	Tapahtuma	TMPH:TMPL
04FFh	Lue TMR1L	xxxxh	Lue TMR1H	xxxxh
0500h	Kirjoita TMPL	xxFFh	Kirjoita TMPH	04xxh
0501h	Lue TMR1H	xxFFh	Lue TMR1L	04xxh
0502h	Kirjoita TMPH	05FFh	Kirjoita TMPL	0401h

TMR1L – Timer1:n alempi tavu, TMR1H – Timer1:n ylempi tavu, TMPL – oma rekisteri alemmalle tavulle, TMPH – oma rekisteri ylemmälle tavulle

Esimerkki 12.

```
; PICmicro Mid-Range MCU Family Reference Manual Exapmle 12-2:
; 16-bittisen käyvän ajastimen lukeminen

; Kaikki keskeytykset on estetty
        movf        TMR1H, 0    ; Ajastimen ylempi tavu työrekisteriin
        movwf       TMPH        ; Kirjoitetaan talteen
        movf        TMR1L, 0    ; Ajastimen alempi tavu työrekisretiin
        movwf       TMPL        ; Kirjoitetaan talteen
        movf        TMR1H, 0    ; Ajastimen ylempi tavu työrekisteriin
        subwf       TMPH, 0     ; Alkup. luku – uusi luku, tulos työrekisteriin
        btfsc       Z           ; Tarkistetaan Z-lippu
        goto        continue    ; Jos tulos 0 ei ylivuotoa tapahtunut, jatketaan
;
; TMR1L on vuotanut yli ylemmän ja alemman tavun luvun aikana
; Nyt suoritetun lukemisen pitäisi tuottaa kelvolliset arvot
;
        movf        TMR1H, 0    ; Ajastimen ylempi tavu työrekisteriin
        movwf       TMPH        ; Kirjoitetaan talteen
        movf        TMR1L, 0    ; Ajastimen alempi tavu työrekisretiin
        movwf       TMPL        ; Kirjoitetaan talteen
; Sallitaan keskeytykset (jos on tarpeen)
continue                        ; Ohjelman ajo jatkuu
```

Esimerkki 13.

```
; Timer2-ajastimen alustaminen
; Huom! On suositeltavaa, että TMR2-rekisteri tyhjennetään ja ajastin sammutetaan
; sekä keskeytykset estetään alustamisen ajaksi
; Näin estetään mahdollisen keskeytyksen tapahtuminen ennen aikojaan
; joka voi sotkea halutun toiminnan
        movlw       0x39        ; 0011 1001, 1:8 jälkiskaalaus, 1:4 esiskaalaus
        movwf       T2CON       ; sekä ajastin pois päältä
        movlw       0x80        ; 1000 000, eli ajastetaan 128 jaksoa
        movwf       PR2         ; jolloin kierron väli on 1024 työjaksoa
                                ; ja keskeytyksen väli 8192 työjaksoa
;
; jos keskeytystä ei aseteta, joudutaan Timer0 pollaamaan
;
tmr2_odotus
        btfss       TMR2IF      ; tarkastetaan, onko keskeytyslippu asettunut
        goto        tmr2_odotus ; jos ei, jatketaan odottamista
;
; ylivuoto on tapahtunut
;
        bcf         TMR2IF      ; kuitataan keskeytys
```

4.8.2. Watchdog Timer ja valmiustila

WDT toimii hieman eri tavalla kuin muut ajastin-yksiköt. Ylivuodon sattuessa ei WDT aiheuta keskeytystä. Normaalin toiminnan aikana mikrokontrolleri resetoidaan (WDT time-out). Jos mikrokontrolleri on valmiustilassa, antaa WDT sille herätteen (WDT wake-up). WDT ja Timer0 käyttävät samaa skaalainta. Jos skaalain siirretään Timer0:n käytöstä WDT:n käyttöön, täytyy se tehdä seuraavalla tavalla (vaikka WDT olisi pois käytöstä):

Esimerkki 14.

```
; PICmicro Mid-Range MCU Family Reference Manual Exapmle 26-1:
; Skaalaimen siirtäminen (Timer0 -> WDT)
; koodissa binääriesityksissä x:t tarkoittavat ei-olennaisia bittejä tämän esimerkin kohdalla
;
        bsf             RP0             ; Pank1
        movlw           b'xx0x0xxx'     ; valitaan kellolähde sekä jälkiskaalaimen arvo
        movwf           OPTION_REG; joka on jotain muuta, kuin 1:1
        bcf             RP0             ; Pank 0
        clrf            TMR0            ; Tyhjennetään TMR0-rekisteri
        bsf             RP0             ; Pank 1
        movwf           b'xxxx1xxx'     ; Valitaan WDT, ei muuteta skaalaimen arvoa
        movwf           OPTION_REG
        clrwdt                          ; Tyhjennetään WDT
        movlw           b'xxxx1###'     ; Valitaan WDT ja uusi skaalaimen arvo (###)
        movwf           OPTION_REG
        bcf             RP0             ; Pank 0
;
; PICmicro Mid-Range MCU Family Reference Manual Exapmle 26-2:
; Skaalaimen siirtäminen (WDT -> Timer0)
; koodissa binääriesityksissä x:t tarkoittavat ei-olennaisia bittejä tämän esimerkin kohdalla
;
        clrwdt                          ; Tyhjennetään WDT ja skaalain
        bsf             RP0             ; Pank 1
        movlw           b'xxxx0###'     ; Valitaan Timer0 ja uusi skaalaimen arvo (###)
        bcf             RP0             ; Pank 0
```

Valmiustilassa mikrokontrollerilla on pienin mahdollinen virrankulutus. Valmiustilaan siirrytään suorittamalla SLEEP-komento. Värähtelijä kytketään pois päältä, jolloin järjestelmällä ei ole kelloa. Portit säilyttävät valmiustilaa edeltäneen tilansa, PD(i) nollataan sekä TO(i) asetetaan. Jotkin virtaa kuluttavat ominaisuudet, kuten BOR-piiri tai WDT, kytketään päälle tai pois mikrokontrollerin asetusten mukaisesti.

Laitteen herättää resetointi, WDT wake-up tai mikä tahansa lisälaitemoduuli. Ensimmäinen vaihtoehto resetoi laitteen, ja ohjelman ajo aloitetaan alusta. Muissa tapauksissa ohjelman ajoa jatketaan siitä, mihin jäätiin. Seuraava suoritettava käsky on haettu muistiin ennen valmiustilaa. Lisälaitemoduuli antaa herätteen, jos sen keskeytys on sallittu. Tämä tapahtuu siitäkin huolimatta, että keskeytykset olisi estetty (GIE bitti on nollattu). Tässä tapauksessa ohjelman suorittamista jatketaan siitä, mihin se oli jäänyt. Jos keskeytykset ovat käytössä (GIE bitti on asetettu), suoritetaan muistissa oleva komento ja hypätään keskeytykseen (0004h). Jos on toivottavaa, ettei mitään käskyä suoriteta heti herätteen jälkeen, kannattaa sijoittaa NOP-komento SLEEP-komennon jälkeen. CLRWDT-komento kannattaa suorittaa ennen SLEEP-komentoa, jotta varmistutaan, että WDT on tyhjä ennen valmiustilaan asettumista.

4.9. Keskeytykset

Keskeytykset ovat joko ulkoisia tai sisäisiä asynkronisia kutsuja, joilla suoritusaika otetaan johonkin toiseen käyttöön. Tyypillinen esimerkki on kellokeskeytys, jolloin tietyn suoritusajan välein käydään suorittamassa jokin asia ilman, että tämä kutsu kirjoitettaisiin koodiin tietyin välein. Laitteella voi olla erilaisia keskeytyksen lähteitä, tyypillisesti yksi jokaista lisämoduulia kohden. Näiden määrä on laitekohtainen, ja se pitää tarkistaa ko. laitteen datalehdestä.

Jokaisella laitteella on rekisteri INTCON (INTerrupt CONtrol), jolla sallitaan tai estetään keskeytysten käyttö ja joka sisältää peruskeskeytykset, jotka löytyvät kaikista laitteista: kellokeskeytys (Timer0-moduulille), ulkoinen keskeytys (RB0) ja PorttiB:n muutos (RB4:RB7). Tämän lisäksi laitteella voi olla yksi tai useampi PIE- (Peripheral Interrupt Enable) ja PIR-rekisteri (Peripheral InterRupt flag), joilla hallitaan lisämoduulien keskeytyksiä. Ks. liite 5 keskeytykset.

Jotta keskeytyksiä voidaan käyttää, on ne sallittava. GIE-bitti (Global Interrupt Enable) sallii tai estää keskeytysten käytön. PEIE-bitti (PEipheral Interrupt Enable) sallii tai estää lisämoduulien keskeytysten käytön. Keskeytysten käyttö itsessään on yksinkertaista: sallitaan keskeytys; odotetaan keskeytyskutsua, jolloin ohjelma hyppää kohtaan 0x04, suoritetaan siellä sijaitseva koodi ja palataan takaisin alkuperäiseen koodiin komennolla retfie (RETurn From IntErrupt). Jotkin keskeytysliput nollautuvat retfie-komennon yhteydessä, mutta eivät kaikki. Tämän takia kannattaa tarkistaa, mitkä liput näin tekevät, ja nollata loput manuaalisesti.

Aivan yksinkertaisimpia ohjelmia lukuun ottamatta keskeytyksiä voi helposti olla useita. Tällöin tarvitaan keskeytyskäsittelijää, joka selvittää, mikä keskeytyksen aiheutti, ja siirtyy suorittamaan tälle tarkoitettu koodia. Tyypillinen keskeytyskäsittelijä voisi olla seuraavanlainen:

```
btfsc   INTCON, 2   ; 0x04, tarkastetaan onko timer0-lippu noussut
call    TimerInt    ; keskeytyksen koodin kutsu
btfsc   INTCON, 1   ; external interrupt lippu (RB0)
call    ExtInt      ; keskeytyksen koodin kutsu
retfie              ; paluu keskeytyksestä
```

Edistyneempi keskeytyskäsittelijä kopioi STATUS- ja W-rekisterit turvaan sekä käyttää kehittyneempiä tapoja löytää keskeytyksen aiheuttaja, varsinkin jos keskeytyksiä on useampia käytössä tai kyseessä on aikakriittinen sovellus.

4.10. Muita huomionarvoisia asioita

4.10.1. Oskillaattori

Oskillaattoriksi on kolme vaihtoehtoa: ulkoinen kello, kide tai muu kellottava piiri, ulkoinen RC-värähtelijä tai sisäinen 4 MHz RC-värähtelijä. Näillä voidaan asettaa piirin kellotaajuus muutamista kymmenistä kHz:stä aina 20 MHz:iin saakka. Yksi työjakso kestää 4 kellojaksoa, ja se koostuu seuraavista osista:

Q1: Käskyn luku (Intruction decode cycle)
Q2: Käskyn haku (Intruction read cycle)
Q3: Tiedon käsittely
Q4: Kirjoitus (Instruction write cycle)

Todellinen toimintanopeus (käskyä/sekunti) saadaan jakamalla kellotaajuus neljällä. Todellinen käskyyn kulutettu aika saadaan 1/toimintanopeus (sekuntia). Esimerkiksi 4 MHz:n kiteellä kellotettu mikrokontrolleri suorittaa miljoona käskyä sekunnissa, ja yhtä käskyä kohti kulunut aika on mikrosekunti.

4.10.2. Ohjelmalaskuri PC

13-bittinen ohjelmalaskuri (PC, Program Counter) koostuu PCL:stä (alemmat 8 bittiä) sekä PCLATH:sta (ylemmät 5 bittiä) ja se pystyy osoittamaan 8K x 14-bittistä ohjelmatavua. Sivuja käsiteltäessä on kirjoitettava PCLATH-nimiseen erityisrekisteriin, PCL:ää käsittelemällä voidaan hyppiä kyseisen sivun ohjelmamuistissa. PCL:ään voidaan suoran kirjoittamisen sijaan lisätä jonkin verran seuraavan esimerkin mukaisesti:

```
movf    PCL, 0      ; kopioidaan PCL => W
addwf   OFFSET, 0   ; W + offset => W
movwf   PCL         ; tulos takaisin PCL:ään
```

PCL:ää kasvatetaan offset-rekisterin verran, ja kun tulos talletetaan takaisin PCL:ään, siirrytään ohjelmassa niin monta pykälää eteenpäin. Tätä kutsutaan laskennalliseksi goto:ksi. Yksi tällaisen käyttö on taulukoitujen arvojen hakeminen; edellisen esimerkin koodi on funktiossa, jonne siirrytään call-komennolla, ja koodin jälkeen on n kpl. retlw-komentoja, jotka palaavat funktiosta tietty luku mukanaan.

4.10.3. Koodin luettavuus

Koska käyttäjä voi nimetä rekisterit ja bitit sekä rivin nimet suhteellisen vapaasti kannattaa käyttää mahdollisimman kuvaavia nimiä. Järjestelmän rekistereihin ja bitteihin suositellaan käytettäväksi dokumenteissa käytettyjä nimiä, jotta sekaannuksilta vältyttäisiin ja koodi olisi mahdollisimman ymmärrettävää. Kommentoinnissa ei myöskään kannata olla säästäväinen. Hyvin kommentoitu koodi on helposti luettavaa koodia. Seuraava esimerkki selventää asiaa.

Esimerkki 15.

```
; Joku ohjelma, joka tekee jotain
; Koettakaa arvata, mitä alla tapahtuu
        org 0x02
        call        a
        goto        p
        goto        q

a       macro_pic16F874_asetukset ; Tämä suorittaa piirin asetusten laiton
        clrf        0x21
        movlw       0x0A
        movwf       0x22
        movlw       0x30
        movwf       0x04
        return
q       decf        0x22
        btfsc       0x03, 0
        bsf         0x23, 0
        movf        0x08, 0
        movwf       0x00
        incf        0x04, 1
        bcf         0x0C, 7
        retfie
p       btfss       0x23, 0
        goto        p
; tästä jatkuu muu ohjelma
        end

; Huolella nimetty koodi
; Tämä koodi on täsmälleen sama, kuin edellä olevassa
; ainoana erotuksena on kommentointi ja nimeäminen

; Vastaus edellä esitettyyn kysymykseen:
; Tämän ohjelmapätkän tarkoitus on kerätä kymmenen arvoa PSP:ltä, ja tallettaa ne taulukkoon
        list p=16F874

#include        <16F874.inc> ; Tämä sisältää kaikki piirin rekisterit, bitit ja nastat
                             ; sekä piirin asetukset sisältävän macron

; Omat vakiot ja rekisterit
#defiene        rivi_pointteri  0x21    ; Tällä osoitetaan taulukon kyseistä riviä
#define         laskuri         0x22    ; Tämä pitää muistissa toistojen määrän
#define         omat_liput      0x23    ; Täällä omat liput ohjaukselle ja merkinannolle
#define         taulukon_alku   0x30    ; Tästä alkaa taulukko
; eli seuraavat kymmenen tavua on varattu tälle
```

```
; Omat bitit
t_valmis        equ     omat_liput, 0   ; Taulukko valmis, 1-valmis

        org 0x02
        call    setup       ; 0x02, alustetaan piiri
        goto    main        ; 0x03, mennään suorittamaan pääohjelmaa
        goto    keskeytys   ; 0x04 – keskeytysvektori, mennään keskeytyksen käsittelijään
setup                       ; Täällä asetetaan piirin toiminta kuntoon

        macro_pic16F874_asetukset ; Tämä suorittaa piirin asetusten laiton

        clrf    rivi_pointteri  ; Alustetaan taulukon osoitin
        movlw   0x0A            ; Taulukon maksimikoko ja toistojen määrä
        movwf   laskuri         ; Alustetaan laskuri

        movlw   taulukon_alku   ; Taulukon aloituskohta työrekisteriin
        movwf   FSR             ; ja asetetaan se epäsuoraan osoitukseen

        return                  ; palataan call-käskyyn

; Keskeytyskäsittelijä jätetty pois esimerkistä
keskeytys
        decf    laskuri         ; pienennetään laskuria yhdellä
        btfsc   Z               ; tarkistetaan onko Z-lippu noussut
        bsf     t_valmis        ; jos on, on laskuria vähennetty kymmenesti
; ja taulukko on tämän kierron jälkeen täysi
        movf    PORTD, 0        ; Siirretään PSP puskuri työrekisteriin
        movwf   INDF            ; ja sieltä epäsuoralla osoituksella talteen
        incf    FSR, 1          ; ja siirretään epäsuora osoitus seuraavaan rekisteriin
        bcf     PSPIF           ; nollataan keskeytys
        retfie                  ; ja palataan ohjelmaan

main
        btfss   t_valmis        ; pääohjelma, odotellaan että taulukko on täysi
        goto    main            ; niin kauan kuin näin ei ole, palataan odottamaan
; tästä jatkuu muu ohjelma
; jossa sitten taulukon arvoja käytetään, kuten käytetään
        end
```

4.10.4. Rekisterien käsittely

Ainoastaan W- eli työrekisteriin voi kirjoittaa suoraan tai suorittaa vähennys- ja yhteenlaskuoperaatioita. Muille rekistereille suoraan suoritettavia toimintoja ovat vieritys vasemmalle tai oikealle, rekisterin tyhjentäminen, arvon pienennys tai kasvatus yhdellä sekä yhden bitin asettaminen tai nollaaminen. Rekisterin tilan muuttaminen osittain on työlästä suoritettavaksi bitti kerrallaan. Tässä hyvänä apuvälineenä toimivat loogiset operaatiot AND, OR ja XOR.

Esimerkki 16.

```
; AND-operaatiolla voidaan pakottaa halutut bitit nollaksi
;
;       rekisteri       1100 1010
;       W               0000 1111
;                       ========
;       AND             0000 1010
;
; OR-operaatiolla voidaan pakottaa halutut bitit ykköseksi
;
;       rekisteri       0000 1010
;       W               1100 0000
;                       ========
;       OR              1100 1010
;
; XOR-operaatiolla voidaan vaihtaa haluttujen bittien tilaa
;
;       rekisteri       1100 1010
;       W               0000 1111
;                       ========
;       XOR             1100 0101
;
; Bittikuviota, jolla operaatio tehdään, kutsutaan maskiksi
; ja biteistä itsestään käytetään usein nimitystä maskibitit.
```

Helpoin tapa tarkistaa onko rekisterin arvo 0, on siirtää rekisteri itseensä MOVF-komennolla, ja pollaamalla Z-lippu.

```
        movf    rekisteri, 1    ; siirretään rekisteri itseensä
        btfss   Z               ; tarkistetaan onko Z-lippu asettunut
        goto    non_zero        ; rekisterin arvo >0
        goto    zero            ; rekisterin arvo =0
;
; Näin tarkistus voidaan tehdä, ilman W-rekisteriä
```

4.10.5. Makrot ja aliohjelmakutsut

Tärkeä asia muistaa makrojen käytössä on, etteivät ne säästä tilaa. Jos makro sisältää kymmenen tavua koodia, ja se sijoitetaan ohjelmaan kymmenen kertaa, kasvaa ohjelman koko sata, ei kymmenen tavua. Makrot ovat erityisen hyödyllisiä korvatessaan pieniä pätkiä koodia selkeällä termillä.

```
; Muistipankin vaihto
;
pank0 macro
        bcf     RP1
        bcf     RP0
endm
;
; Saman voi suoritta pelkällä RP0:n käsittelyllä, jos käytössä on vain kaksi muistipankkia
```

Aliohjelmat hidastavat koodin suorittamista, koska haarautumiskäskyn suorittamiseen kuluu kaksi työjaksoa yhden sijaan. Jos kyseessä on aikakriittinen sovellus ja ohjelmamuistia on käytettävissä riittävästi, kannattaa toistuvan koodin niputtaminen, selkeän luennan vuoksi, suorittaa makroilla aliohjelmien sijaan. Aliohjelman kaksi tärkeintä käyttöä ovat keskeytykset sekä taulukot.

Esimerkki 17.

```
; Taulukoidun tiedon käytöstä
; Tähän kohtaan on saavuttu call-käskyllä
;
; OFFSET-rekisteri sisältää arvon 1-3,
; tämä kertoo mikä taulukon arvo halutaan
;
        movwf   OFFSET, 0
        addwf   PCL, 1          ; Lisätään OFFSET:n verran ohjelmalaskuria
        retlw   0xA0            ; Jos OFFSET on 1, palautetaan arvo A0h
        retlw   0xB0            ; Jos OFFSET on 2, palautetaan arvo B0h
        retlw   0xC0            ; Jos OFFSET on 3, palautetaan arvo C0h
;
; jos halutaan 2-ulotteinen taulukko:
;
        movwf   OFFSETX, 0
        addwf   PCL, 1
        goto    rivi_1
        goto    rivi_2
        goto    rivi_3

rivi_1
        movwf   OFFSETY, 0
        addwf   PCL, 1
        retlw   0x1A
        retlw   0x2A
        retlw   0x3A
;
; rivit 2-5 samalla kaavalla
;
; jos halutaan palauttaa muuttuja
; niin retlw palauttama luku on rekisterin osoite
; tätä osoitetta käytetään epäsuoralla osoituksella
```

5. PROJEKTIN TOTEUTTAMINEN

5.1. Yleisesti

Projektia suunniteltaessa ja toteutettaessa käytettiin hyväksi PADS-suunnitteluohjelmistoa, haettiin esimerkkejä internetistä sekä hyödynnettiin asiantuntijoiden mielipiteitä ja arvioita eri osa-alueiden toteuttamiseksi. Tietoa haettiin myös Microchipin dokumentoinneista, kuten PICmicro Mid-Range MCU Family Reference Manual ja Embedded Control Handbook, volume 1.

5.2. Harjoitusalustan toteuttaminen

Harjoitusalustan suunnittelemisen lähtökohdaksi otettiin mikrokontrolleri. Sen nastat jaoteltiin kytkettäväksi käytettyjen ominaisuuksien mukaan. Tämän lisäksi liittimet tuli sijoittaa reunoille, LEDit näkyvään paikkaan ja painonapit niin, että niitä pystyisi operoimaan. Väylien kanssa pyrittiin noudattamaan yksinkertaista sääntöä, että pystysuuntaiset väylät sijoitetaan toiselle puolelle ja vaakasuuntaiset toiselle puolelle levyä. Näin suuremmilta ongelmilta väylän vedon suhteen vältyttiin.

Piirilevysuunnittelu toteutettiin PADS Logicin avulla ja piirilevyn fyysinen versio piirrettiin PADS Layoutilla. Piirikaavio tulostettiin kalvolle ja piirilevy valmistettiin ensin valottamalla piirikaavio valoherkälle levylle ja sen jälkeen syövyttämällä ylimääräinen kupari. Vaihtoehtoinen tapa olisi ollut käyttää piirilevyjyrsintä. Tällöin piirikaavio olisi viety sähköisessä muodossa jyrsimelle. Se puolestaan jyrsii piirikortin kuparista pois valitut alueet. On huomioitava, että jyrsimelle ja syövyttämiseen on tarkoitettu erilaiset piirilevymateriaalit. Vääränlaisen piirilevyn käyttämistä tulee välttää.

Koska käytössä ei ollut simulaattoriohjelmaa, jolla piirilevyn toiminta olisi voitu testata, pystyi siitä testaamaan vain tietyt osa-alueet eli yhteiskellon sekä virtalähteen. Kumpikin näistä pystyttiin rakentamaan erillisenä ja testaamaan, että toiminta oli toivottua. Tällainen simulaattoriympäristö on esimerkiksi Labcenter Electronicsin Proteus.

Proteuksessa yhdistyy Spicen simulaattoriympäristö MPLABin simulaattoriympäristöön, mikä mahdollistaa samanaikaisesti sekä kytkennän että koodin testaamisen. Proteus sisältää myös piirikortin suunnittelun, jolloin koko projekti suunnittelusta testaamiseen ja siitä varsinaisen laitteen rakentamiseen voidaan toteuttaa yhdellä ohjelmistolla (10).

5.3. Ohjelmiston toteuttaminen

PIC-mikrokontrollerin koodi kirjoitetaan puhtaana tekstitiedostona käyttäen PIC assembler-kieltä. Tämä lähdetiedosto voi koodin lisäksi sisältää kommentteja, muotoiluja ym. koodin lukemista helpottavia asioita.

Microchipin MPLAB tarjoaa kääntäjän lisäksi myös simulaattorin. Simulaattorilla koodi voidaan ajaa läpi ja tarkastella, kuinka rekisterit käyttäytyvät ajon aikana. Simulaattorille voidaan myös luoda stimulus-tiedosto, johon määritellään nastojen tila tietyllä kellojaksolla. Tämä tila on säilyvä, kunnes se jälleen muutetaan. Tämä stimulus-tiedosto simuloi ulkoista vaikutusta. Sillä ei voi kuitenkaan simuloida esimerkiksi A/D-piirin näytteenottoa, koska stimulus on sisällöltään digitaalinen.

Ohjelmisto, joka harjoitusalustan jokaisen komponentin ohjaamiseen tarvittaisiin, on huomattavan laaja sekä hankala toteuttaa. Tästä syystä kannattaa ohjelmisto pilkkoa pieniin osasiin aina sen mukaan, mitä osa-aluetta harjoitellaan. Tällaisia osa-alueita voisivat olla esimerkiksi perus-I/O-rajapinta eli LED ja painonappi tai A/D ja tiedonsiirto käyttäen joko I^2C:tä tai USARTia.

6. Loppupäätelmä

Projektin etenemistä kannattaa valvoa heti alusta alkaen, vaikka aluksi edistystä tuskin tapahtuu. Näin kuitenkin voidaan puuttua epäkohtiin jo ennen kuin niistä muodostuu ongelmia projektin kannalta. Työn etenemisestä, ongelmista sekä muutoksista kannattaa tiedottaa kaikille projektiin liittyville tahoille. Näin asiat eivät tule kenellekään yllätyksenä. Myös sen, mitä kukakin projektissa tekee, olisi hyvä olla kaikkien tiedossa. Eteneminen, muutokset ja ongelmat kannattaa kaikki dokumentoida. Kaikkea ei voi muistaa, joten dokumentointi on ensiarvoisen tärkeää projektiarviointia tehdessä, sekä niistä voi ottaa oppia tulevien projektien kohdalla.

Projektin suunnittelu oli alusta asti heikkoa. Laajuutta ei osattu arvioida oikein, sillä jo harjoitusalustan suunnittelu, kasaaminen ja testaaminen muodostivat hyvin laajan kokonaisuuden. Harjoitusalustan testaamista ei missään vaiheessa edes huomioitu projektin osa-alueeksi. Työmäärä osoittautui arvioitua suuremmaksi, eikä apua työn toteuttamiseen osattu pyytää, ennen kuin projekti oli epäonnistunut.

Projektille ei ollut asetettu mitään kiinteää takarajaa valmistumisen suhteen, mutta silti voidaan sanoa, että aikataulu ei toiminut. Se suunniteltiin summittaisesti, ilman konkreettisia perusteita. Sen noudattaminen oli käytännössä mahdotonta, eikä siinä huomioitu projektin ulkopuolisia asioita, kuten projektin ulkopuolisia töitä, eikä projektille jäänyt sen vaatimaa aikaa. Budjetointi oli projektisuunnitelman ainoa onnistunut kohta, sillä budjetti laadittiin komponenttilistan mukaiseksi.

Mikäli tulevaisuudessa aiotaan tehdä harjoituksia tai töitä, joihin liittyy mikroprosessorilla varustetun piirikortin rakentaminen ja testaaminen, suositellaan hankittavaksi ohjelmisto, jolla voidaan suorittaa piirikortin suunnittelu sekä simuloida piirikorttia ja ohjelmistoa ilman, että korttia valmistetaan.

Kymenlaakson ammattikorkeakoululle tehdyistä oppilastöistä tulisi teettää ensimmäiseksi projektisuunnitelma. Näin kannattaa menetellä jo mahdollisimman varhaisten töiden kohdalla, jotta projektisuunnitelman laatiminen tulisi tutuksi sekä siitä tulisi ilmeinen osa projektia. Suunnitelman merkitystä oikeiden projektien kohdalla ei voi vähätellä. Niinpä sen laatiminen olisi ensiarvoisen tärkeää oppia jo mahdollisimman varhaisessa vaiheessa.

Lähteet

1. Kehruu-Jenny. Wikipedia-artikkeli (Viitattu 17.11.2009)

Saatavissa: http://fi.wikipedia.org/wiki/Kehruu-Jenny

2. Spinnig Mule. Wikipedia-artikkeli (Viitattu 17.11.2009)

Saatavissa: http://en.wikipedia.org/wiki/Spinning_mule

3. Ohjelmoitava logiikka. Wikipedia-artikkeli (Viitattu 17.11.2009)

Saatavissa: http://fi.wikipedia.org/wiki/Ohjelmoitava_logiikka

4. Microcontrollers (Viitattu 17.11.2009)

Saatavissa: www.picvietnam.com/download/Basic%20Conceptions.pdf

5. PIC24F Microcontrollers: Microcontroller Architectures (Viitattu: 17.11.2009)

Saatavissa: http://www.pic24micro.com/harvard_vs_von_neumann.html

6. Modified Harvard architecture. Wikipedia-artikkeli (Viitattu: 17.11.2009)

Saatavissa: http://en.wikipedia.org/wiki/Modified_Harvard_architecture

7. RISC. Wikipedia-artikkeli (Viitattu: 17.11.2009)

Saatavissa: http://fi.wikipedia.org/wiki/RISC

8. Microchip PICmicro™ Mid-Range MCU Family Reference manual

(Viitattu: 17.11.2009)

Saatavissa: http://ww1.microchip.com/downloads/en/DeviceDoc/33023a.pdf

9. MPASM™/MPLINK™ PICmicro® MCU Quick Chart (Viitattu: 17.11.2009)

Saatavissa: http://ww1.microchip.com/downloads/en/DeviceDoc/30400g.pdf

10. Proteus VSM (Viitattu 17.11.2009)

Saatavissa: http://www.labcenter.co.uk/products/vsm_overview.cfm

Liitteet

LIITE 1: PIC16F87X Mikrokontrollerit (8)

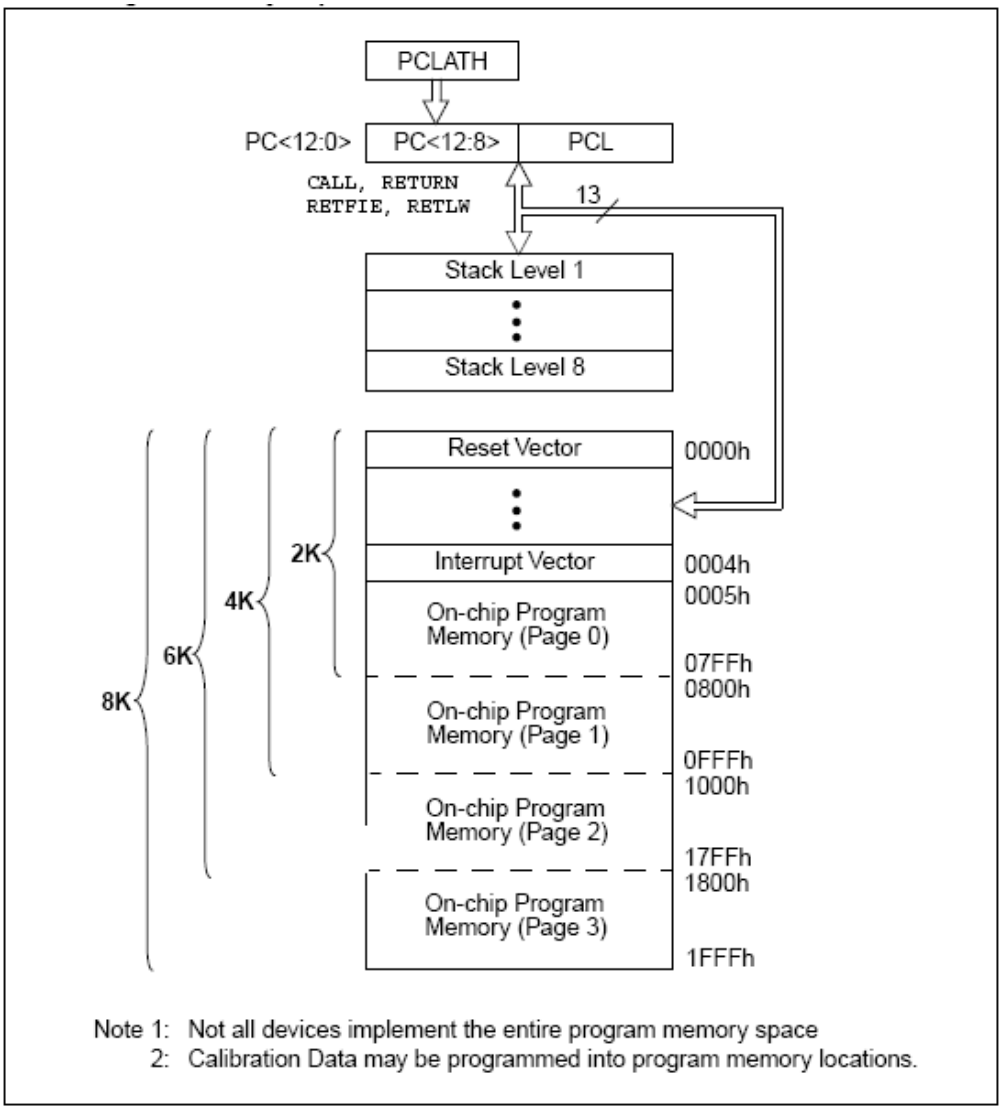
Key Features PICmicro™ Mid-Range Reference Manual (DS33023)	PIC16F873	PIC16F874	PIC16F876	PIC16F877
Operating Frequency	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz
Resets (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
FLASH Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory	128	128	256	256
Interrupts	13	14	13	14
I/O Ports	Ports A,B,C	Ports A,B,C,D,E	Ports A,B,C	Ports A,B,C,D,E
Timers	3	3	3	3
Capture/Compare/PWM modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Instruction Set	35 Instructions	35 Instructions	35 Instructions	35 Instructions

LIITE 2: Kääntäjän tukemat numeraalien esitysmuodot (9)

MPASM Radix Types Supported

Radix	Syntax	Example
Binary	B‘<binary_digits>’	B‘00111001’
Octal	O‘<octal_digits>’	O‘777’
Decimal	D‘<digits>’ .<digits>	D‘100’ .100
Hexadecimal (default)	H‘<hex_digits>’ 0x<hex_digits>	H‘9f’ 0x9f
Character (ASCII)	A‘<Character>’ ‘<character>’	A‘C’ ‘C’

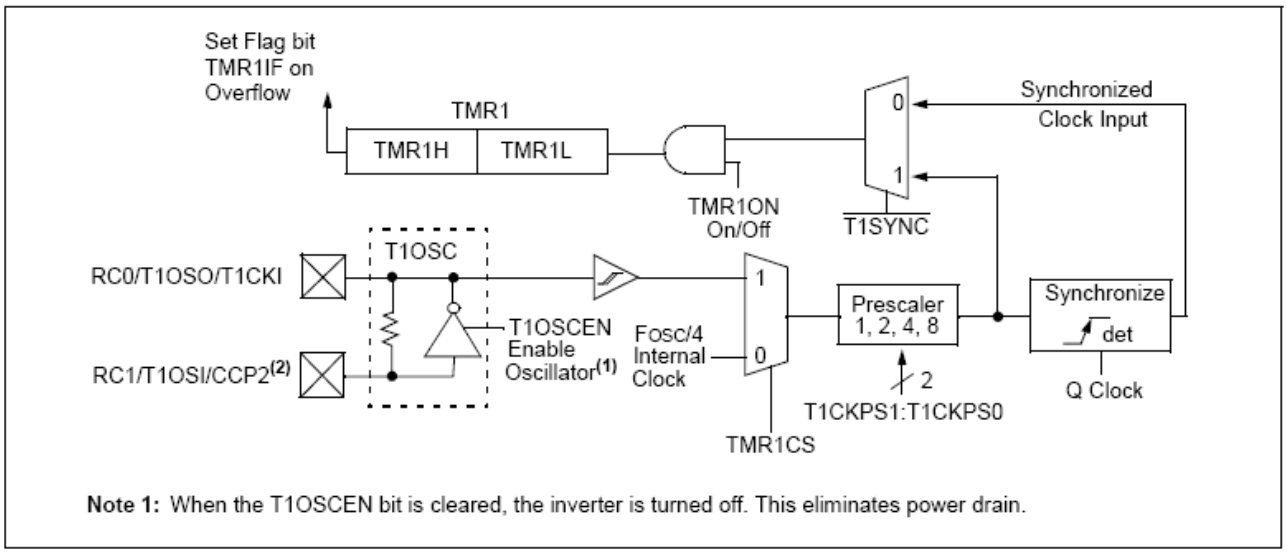
LIITE 3: PIC:n muistiorganisaatio (8)



Note 1: Not all devices implement the entire program memory space
2: Calibration Data may be programmed into program memory locations.

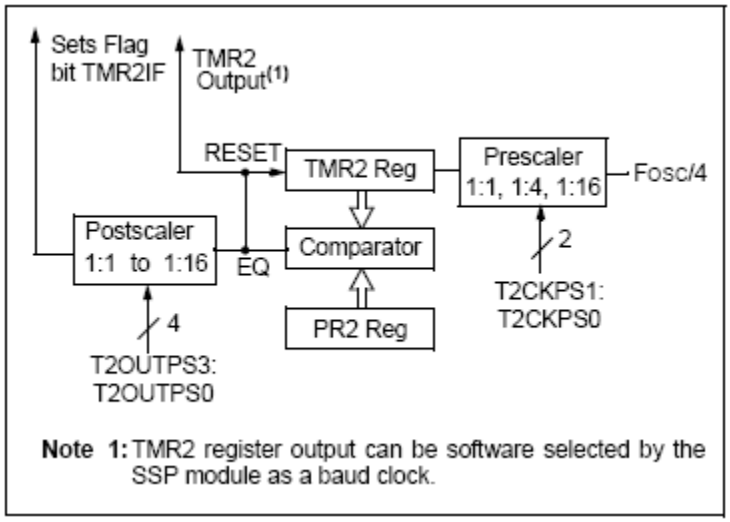
LIITE 4: Timer1-moduulit (8)

Timer1-moduuli



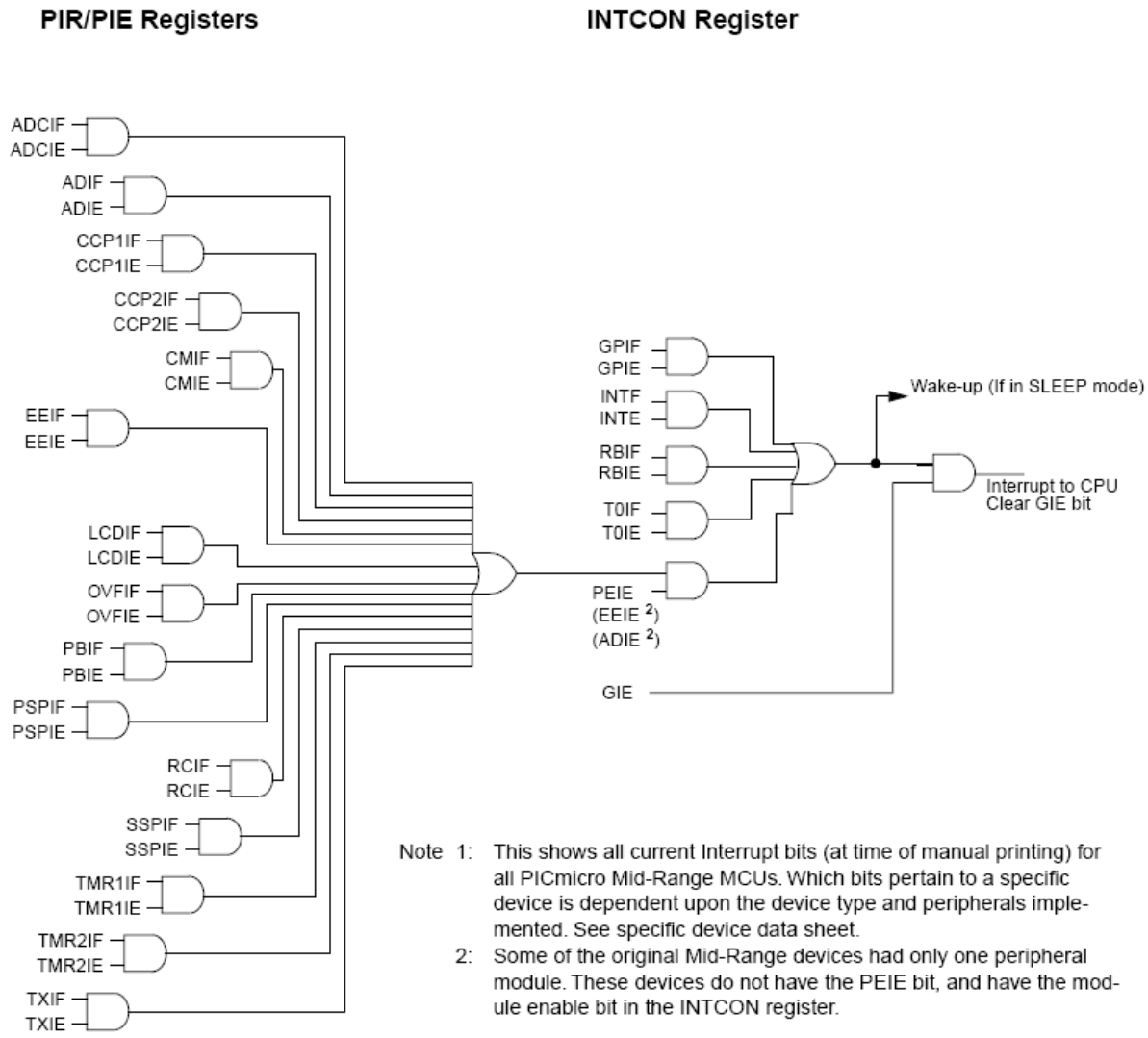
Note 1: When the T1OSCEN bit is cleared, the inverter is turned off. This eliminates power drain.

Timer2-moduuli



Note 1: TMR2 register output can be software selected by the SSP module as a baud clock.

LIITE 5: PIC:n keskeytyskartta (8)



Note 1: This shows all current Interrupt bits (at time of manual printing) for all PICmicro Mid-Range MCUs. Which bits pertain to a specific device is dependent upon the device type and peripherals implemented. See specific device data sheet.

2: Some of the original Mid-Range devices had only one peripheral module. These devices do not have the PEIE bit, and have the module enable bit in the INTCON register.