Datavetenskap

Jonas Larsson och Andreas Agorander

Mätdon för felsökning av hydraulventiler

Examensarbete

2004:18

Mätdon för felsökning av hydraulventiler

Jonas Larsson och Andreas Agorander

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är vårt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Jonas Larsson

Andreas Agorander

Godkänd, 2004-06-03

Handledare: Per Strömgren

Examinator: Stefan Alfredsson

Sammanfattning

Ångpanneföreningen (ÅF) har utvecklat system för styrning av hydraulventiler med hjälp av pulsbreddsmodulerade styrsignaler. När systemet som styrs på detta vis fallerar mäts denna styrsignal som en del i felsökningsprocessen. För detta ändamål har ÅF sedan tidigare ett enkelt mätverktyg som mäter medelspänning och ström. Denna information anses dock inte längre tillräcklig och därför utvecklar man ett nytt instrument med ytterliggare funktionalitet för felsökning. Examensarbetet behandlar utvecklingen av mjukvaran till detta nya instrument.

Resultatet är ett väl användbart system som löser de problem och önskemål som funnits.

Tack till:

Anders Lönnstam, vår handledare vid ÅF.

Kalle Dahlbom, ÅF.

Per Strömgren, vår handledare vid Karlstads universitet.

Abstract

Ångpanneföreningen (ÅF) has been developing systems for controlling hydraulic vents using pulse width modulated signals. Measuring the PWM-signals is a part of the debugging procedure when these systems fail. ÅF is already in possession of a simple tool which measures the average voltage and electric current, but this information is no longer regarded as sufficient. They are therefor developing a new tool which provides extended functionality for debugging. This report deals with the development of the software for this new tool.

The work resulted in a useful system which solves the problems and implements the desired features that were requested during the project.

Innehåll

Figurer

Tabeller

1 Inledning

Detta examensarbete omfattar 20 poäng och har utförts av Andreas Agorander och Jonas Larsson på Ångpanneföreningen System & Industri AB i Karlstad. Det påbörjades 2004-01-22. Via gemensamma kontakter fick vi reda på att ÅF eventuellt hade ett intressant exjobb involverandes hårdvarunära programmering. Uppgiften går ut på att skriva styrprogrammet för ett mätinstrument som skall användas för felsökning av vissa datorstyrda system.

I kapitel 2 går vi igenom bakgrunden till arbetet och presenterar det sammanhang instrumentet vi programmerat ingår i och det problem det skall åtgärda. Även det resultat som efterfrågas presenteras. I kapitel 3 går vi igenom utvecklingsmetodiken, dvs det arbetssätt vi använde för att komma till det resultat vi kom fram till. I kapitel 4 redovisas resultatet av arbetet. Dessa resultat diskuteras i kapitel 5. Projektberättelsen i kapitel 6 behandlar hela utvecklingsprocessen.

2 Bakgrund, problem, syfte och mål

Detta kapitel beskriver bakgrunden till uppgiften och det resultat som efterfrågades. Det börjar med en enkel beskrivning av den situation som ger upphov till uppgiften, och fortsätter med de alternativ som finns idag och varför de inte betraktas som tillräckliga. Slutligen utmynnar detta i en "önskelista" för vad som skall åstadkommas i konstruktionsdelen av uppgiften.

2.1 Bakgrund

ÅF har för avsikt att utveckla ett nytt mät- och felsökningsinstrument som vi deltagit i att utveckla.

2.1.1 Målgrupp

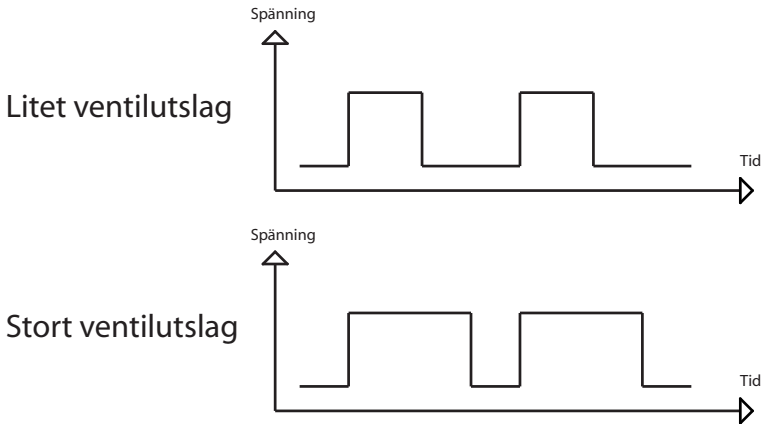
För systemet finns det två huvudsakliga problem som skall lösas, och då också två olika målgrupper, vars behov måste tas i beaktande vid utformningen av systemet.

Det första är felsökning i produktionssammanhang. Det kan vara en skogsmaskin som slutar fungera i fält. Det andra fallet är felsökning i konstruktionssammanhang, dvs när konstruktören av ett industrisystem i sin utveckling behöver information om hur systemet beter sig.

2.1.2 Hydraulventiler och PWM

Det problem som ligger som specifik bakgrund för uppgiften har att göra med styrning av hydraulventiler via PWM-signaler. Instrumentet som utvecklats kommer kunna användas till andra mätningar av PWM-signaler också men "ventilfallet" presenteras som det huvudsakliga problemet.

Styrningen sker genom att en elektrisk signal med en fast frekvens (i det här fallet 120hz) genereras, som anger vilket läge ventilen skall ha. Signalen är en fyrkantvåg med två nivåer (hög och låg) där läget på ventilen bestäms av hur stor andel av perioden som vågen är i "högt" läge.



Figur 2.1: PWM-signal

Detta är samma styrsätt som används i t ex servosystem till radiostyrda bilar/flygplan och dylikt. Om ett system som styrs på detta vis fallerar är det av intresse att undersöka styrsignalen som en del av felsökningen.

2.1.3 Problem med existerande felsökningsmetoder

Det finns ett par alternativ som används vid felsökning idag. Det ena är att koppla in ett oscilloskop till ventilerna som skall felsökas. Problemet är att det är otympligt att dra med ett oscilloskop ut i fält.

Som smidigare alternativ har ÅF sedan tidigare utvecklat ett enkelt instrument som ger enkel information (medelspänning och medelström) som kan användas till viss felsökning. Detta instrument är smidigare men ger ingen detaljerad information vilket kan vara nödvändig.

Problemet är alltså att antingen är existerande lösning för otymplig, eller så ger den för lite information.

2.2 Syfte och mål med examensarbetet

Det nya instrumentet skall för att lösa ovanstående problem förutom att vara i ett lätthanterligt format dels:

- Visa den information (medelspänning och medelström) som det existerande instrumentet visar.

- Visa vågformerna för ström och spänning under perioden.

Dessa behov löser i stora drag det första användarfallet, felsökning i fält.

För att på ett smidigt sätt tillhandahålla konstruktörer den data de behöver så specificerades att konstruktören skall kunna få relevant mätinformation till sin egen dator. Det innebär att instrumentet behöver nätverksfunktionalitet, och till systemet behövs ett tillhörande program så att persondatorer kan kommunicera med instrumentet.

Arbetet går huvudsakligen ut på att utveckla mjukvaran. Hårdvaran tillhandahålls av uppdragsgivaren. Någon mer detaljerad kravspecifikation gavs inte vilket gav oss stor frihet i utvecklingen av systemet. De yttre begränsningarna som fanns gavs av den hårdvara som uppdragsgivaren hade specificerat som tillgänglig för uppgiften.

Följande deluppgifter identifierades för att lösa uppgiften:

A Inläsning av mätvärden via analog till digital konvertering.

B Presentation av mätvärden på display

C Styrfunktionalitet för display med hjälp av knappar

D Kommunikationsmöjlighet över nätverk

E Windowsprogram som kan kommunicera med instrumentet

I uppgift A ingår att sampla (dvs mäta värdet vid många jämnt fördelade tidpunkter) de två mätvärdena i en sådan takt att det går att presentera resultatet som en lämplig kurva.

I uppgift B ingår två delar. Dels en grundläggande drivrutin för displayen. Displayen ifråga har inte använts tidigare av uppdragsgivaren och därför måste grundläggande styrkod (drivrutin) för denna skapas. Intressant för uppdragsgivaren är att denna blir så generell och modulär som möjligt så att drivrutinen kan återanvändas. Utöver detta består uppgift B av den kod som gör de specifika uppritningarna för att presentera resultatet vi har fått från mätningarna.

Uppgift C är lite luddig då det inte är helt specificerat exakt vilken styrfunktionalitet som är efterfrågad. Men den består åtminstonde av en lågnivås drivrutin för att hantera knappar, och användande av denna för att växla mellan olika visningslägen (kurvor, siffror).

Även uppgift D består av en lågnivås drivrutin för nätverkshårdvarumodulen och en programmodul på högre nivå.

Uppgift E består dels av en generell programdel för att kommunicera med hårdvarumodulen och dels ett skal mot detta som ger motsvarande visnings och styrfunktionalitet som finns

på själva hårdvarumodulen.

3 Arbetssätt

I denna sektion beskrivs de metoder och verktyg, samt det arbetssätt vi använde för att planera och lösa uppgiftens olika delar. För att illustrera processen nämns vissa händelser under arbetets gång snabbt här. Mer detaljerad beskrivning händelser fås i projektberättelsen.

3.1 Inläsning

Nödvändig inläsning bestod av tillverkarnas dokumentation till hårdvarumoduler, RFC:er för kommunikationsprotokoll, samt forum på tillverkarens hemsida vid behov, i de fall dokumentationen ej var fullständig. För tillverkaroberoende implementationsproblem användes en uppsättning böcker (se bibliografi). Ingen dedikerad inläsningsfas användes, utan dokumentation användes som uppslagsverk vid behov.

3.2 Använda verktyg

Programmeringen av instrumentet genomfördes i C med en kompilator specialanpassad för den mikroprocessor som användes (PIC). Med hjälp av en ICD (In Circuit Designer) kunde programmering och testning ske mot färdig uppkoppling.

Perl har använts för att skapa kodgenererande script för att automatisera viss kodning, speciellt datatabeller för ingående grafik. För PC-styrprogrammet användes C++ i Visual Studio .NET.

Ethereal, ett program för analys av nätverkstrafik, användes för felsökning av nätverksfunktionalitet.

3.3 Planering

Den planering som genomfördes var väldigt grov och bestod av tidsallokering av de deluppgifter som identifierats i sektion "Syfte och mål". En sådan planering gjordes innan arbetet påbörjades. Mot slutet av implementationsfasen tillverkades en reviderad version då tidsplanen inte höll.

Inom varje deluppgift försökte vi skapa oss en bild av ungefär vilken funktionalitet som var nödvändig i diskussion med uppdragsgivare och genom snabb analys av funktionalitetens plats i systemet. Denna ursprungliga bild av levererad funktionalitet stämde dock inte helt överens med det som visade sig behövas under implementationens gång. Så till stor del blev arbetet ad-hoc med avseende på tillägg av funktionalitet.

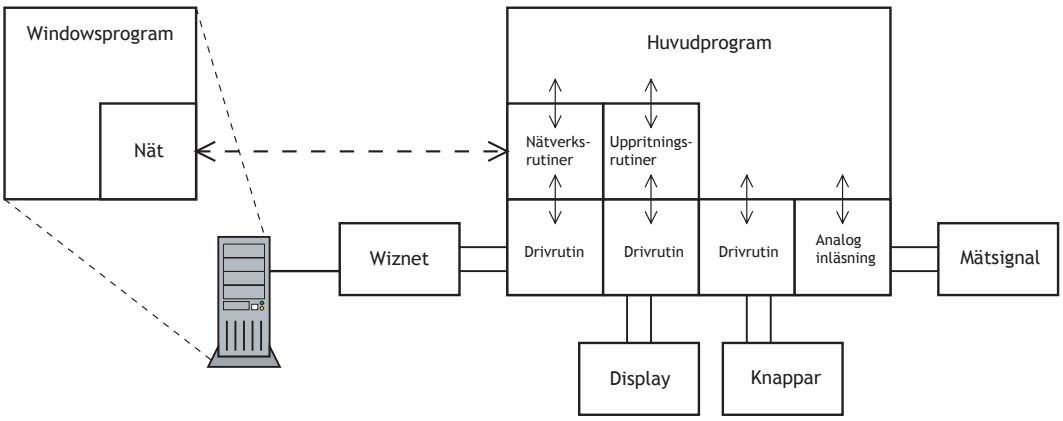
Som hjälp innan första funktionalitetsspecifikationerna tillverkade vi en grafisk bild av hur instrumentet skulle kunna se ut och vilken funktionalitet det skulle kunna tillhandahålla. Resultatet innehöll en del av denna funktionalitet.

3.4 Implementation

Här beskrivs implementationsprocessen för olika kategorier av funktionalitet. Systemets olika komponenter syns i figure 3.1

3.4.1 Drivrutiner

Inom uppgift A till D som listats i "syfte och mål" ingår det en varsin hårdvarunära drivrutin. För implementation av dessa specificerades först en grov lista av nödvändig funktionalitet som drivrutinen skall leverera till det övriga programmet. En snabb översikt över tillverkarens dokumentation över hårdvaran låg till grund för den övergripande metodiken för hur implementationen skulle ske. Efter detta implementerades drivrutinen steg för steg. Det var dock svårt att åtskilja implementation av drivrutin från implementation av de rutiner som använde drivrutinerna. Ändringar i kraven på högre nivå ledde ofta till att ingrepp



Figur 3.1: Blockschema

i drivrutinerna blev nödvändiga.

3.4.2 Funktionalitet mot användare

För att leverera funktionalitet till användaren programmerades rutiner som huvudsakligen använder sig av drivrutinerna. Ingen detaljerad specifikation över önskad funktionalitet gavs. Istället utgick vi ifrån den översiktliga specifikation uppdragsgivaren hade försett oss med, och kombinerade detta med att ofta demonstrera eller redovisa vad som tillverkats för uppdragsgivaren. På detta sätt blev funktionaliteten under implementationens gång allt mer specificerad.

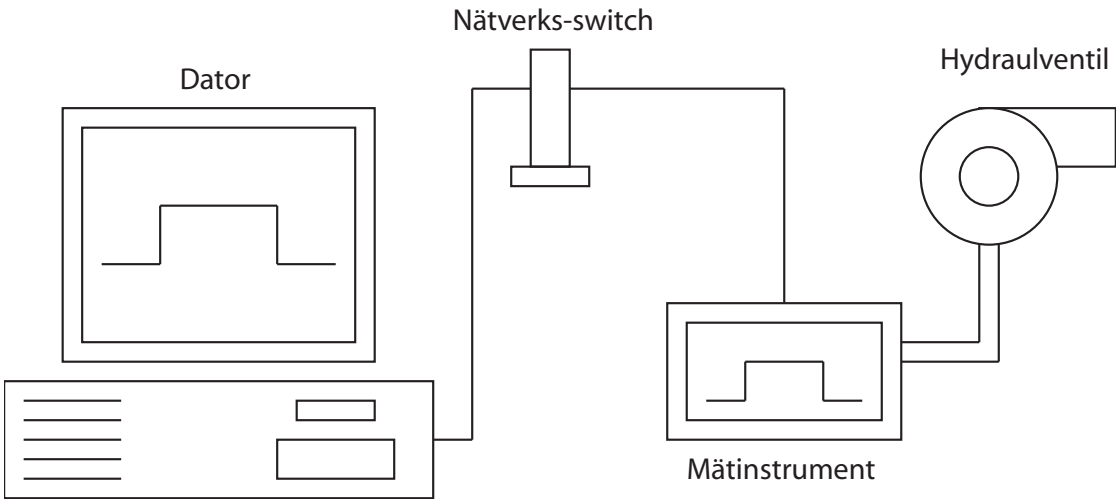
Inga hårda kriterier ställdes upp för att betrakta en viss funktion som färdig. Implementationen fortgick istället tills levererad funktionalitet något subjektivt bedömdes som motsvarande de specificerat nödvändiga eller efterfrågade.

4 Resultat av vårt arbete

Här beskrivs det resultat som vårt arbete utmynnat i. Först beskrivs systemet som det ter sig för en användare. Den tekniska beskrivningen är översiktlig. För mer detaljerad information se bilaga B.

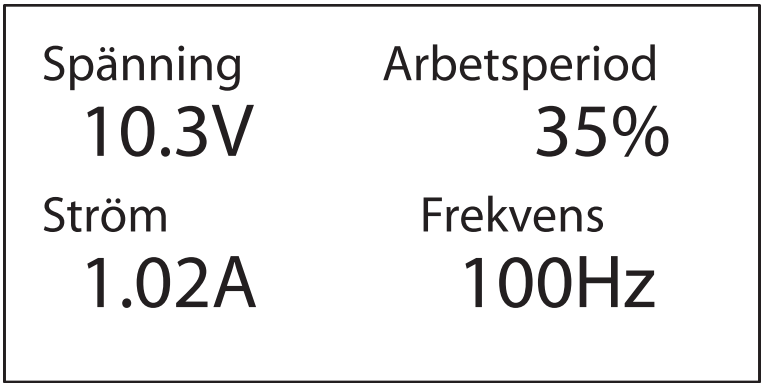
4.1 Systemets vy för användare

Det finns två vyer in i systemet. Det ena är instrumentinterfacet med knappar och LCD-display. Det andra är ett windowsprogram som används för att komma åt instrumentet över lokalt nätverk. Systemet ser alltså ut så här:

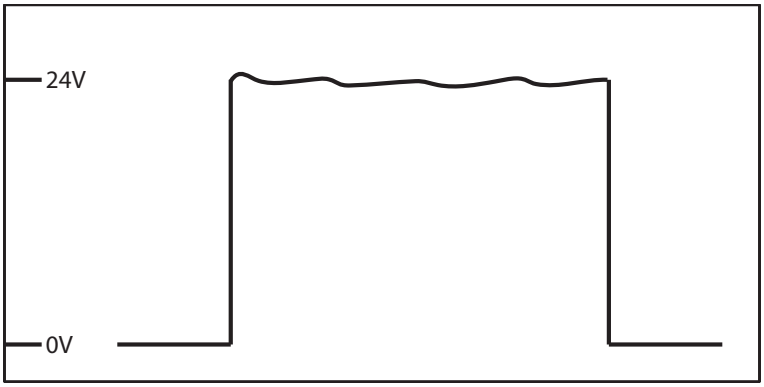


Figur 4.1: Systemsammansättning

Obligatorisk del för att ha någon funktionalitet alls är instrumentdelen. Mätinstrumentet kopplas in mot den enhet som skall mätas. Mätresultaten visas på displayenheten på olika sätt. Styrning av instrument tillhandahålls genom två knappar. Den ena knappen växlar presentationsläget för mätdatan. Lägen som finns är data i form av text, där medelspänning, medelström, frekvens och arbetsperiod presenteras. Två andra presentationslägen är Spänningskurvans respektive strömkurvans utseende.



Figur 4.2: Presentation på display, textläge

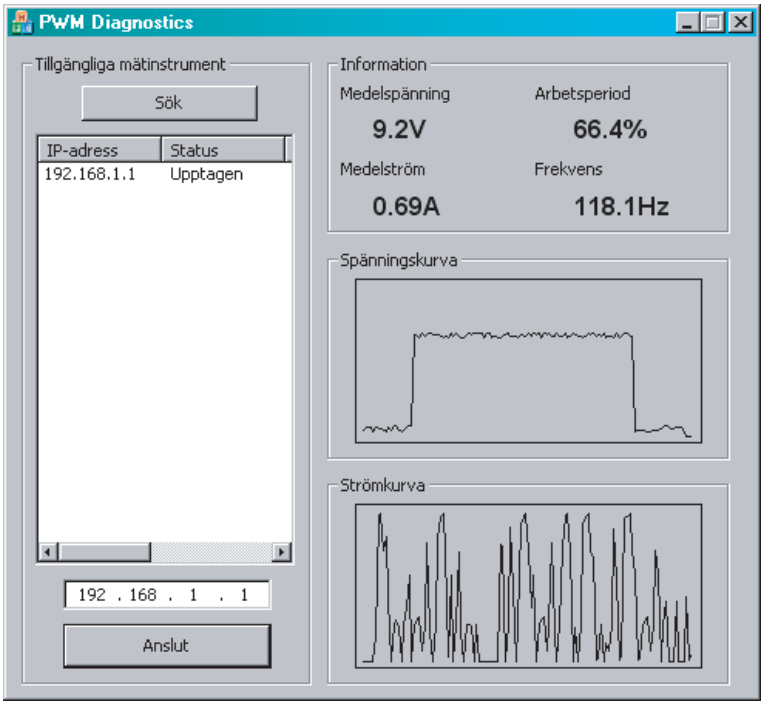


Figur 4.3: Presentation på display, grafläge

Den andra knappen fryser visningsläget så att inga nya data presenteras förrän frysningen hävs.

Istället för att behöva gå till instrumentet och läsa av mätvärden på display kan man få sina mätvärden levererade till en dator. För att åstadkomma detta kopplas instrumentet in på samma lokala nätverk som den dator man vill få mätdata levererad till. Det windowsprogram som skapats används sedan för att söka efter, eventuellt konfigurera, kommunicera med och slutligen presentera mätdata på datorskärmen.

Ett annat sätt att få mätinformation till datorn är att ansluta till instrumentet via en standardwebbläsare (t ex Internet Explorer eller Mozilla), då mätdata presenteras som



Figur 4.4: Skärmdump av windowsprogrammet

text och i form av två kurvor i en bild.

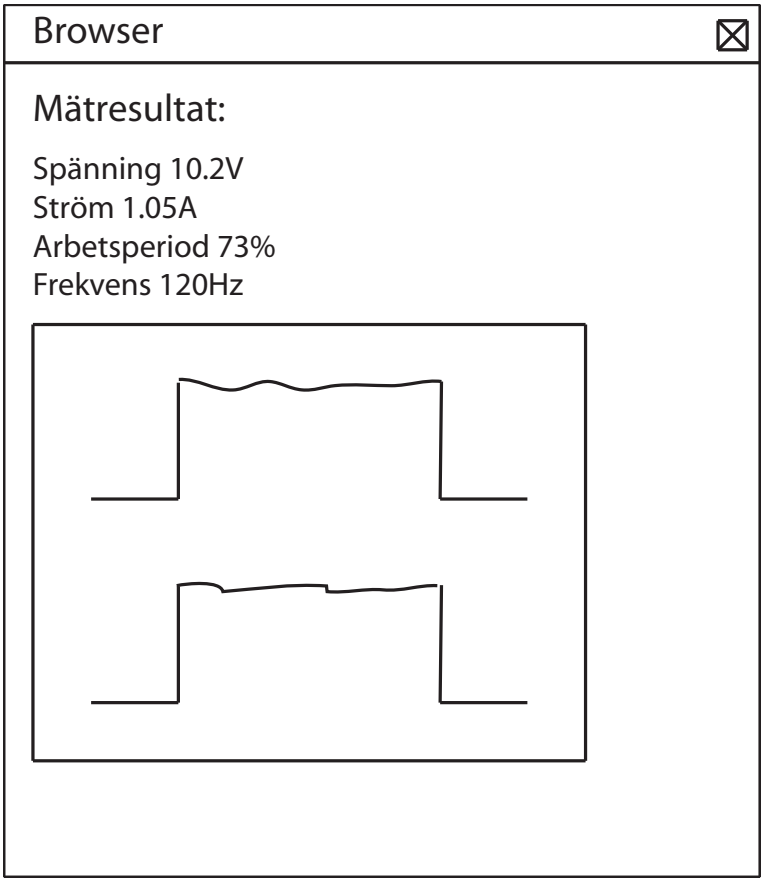
Den konfiguration som kan behöva göras av användare innan nätverksöverföring fungerar är tilldelning av ipnummer. Detta görs via windowsprogrammet.

4.2 Teknisk funktionalitet hos systemet

Här beskrivs den funktionalitet vi implementerat i systemet uppdelat i olika övergripande funktionalitetskategorier, huvudsakligen efter de deluppgifter som identifierats i tidigare sektioner.

4.2.1 Mätning och presentation

Den första delfunktionaliteten är inläsning av mätdata på ett sådant sätt att all nödvändig information kan räknas ut och presenteras. Systemet tar in och mäter två analoga sig-



Figur 4.5: Skiss över webbsvaret

naler, som via inkoppling har sin källa i den signal som mäts. Den ena är spänningen hos signalen som mäts och den andra är strömmen på samma signal. Uppdelningen av signalen till två som kan läsas av programmet sker via en hårdvarukoppling som designats av uppdragsgivaren.

Systemet samplar en period av signalen i 128 mätpunkter. För att kunna räkna ut i vilken takt och när dessa mätningar skall genomföras så har systemet följande funktionalitet: detektering av frekvens (mellan 50-150Hz) och centrering av mätsignal. Den första funktionaliteten räknar ut hur ofta dessa mätningar skall genomföras, och den andra när mätning skall påbörjas. Med hjälp av dessa mätvärden kan kurvor, medelvärden, frekvens

och arbetsperiod utrönas och presenteras.

När data räknats ut visas den på display. Hela processen går så pass snabbt att systemet klarar av mer än 15 mätnings och presentationscykler per sekund. Vid presentation i textformat har processen saktats ner så att siffrorna inte uppdateras mer än ca 2 ggr per sekund.

4.2.2 Nätverksfunktionalitet

Nätverksfunktionaliteten är implementerad som fyra delfunktioner.

Den första funktionen är DHCP. Med hjälp av denna funktionalitet efterfrågar instrumentet konfigureringsparametrar för nätverket, som ipnummer, nätmask och router. Om det finns en DHCP-server på det nätverk som instrumentet kopplas in på så kan instrumentet med hjälp av detta automatiskt konfigurera sina nätverksinställningar utan ytterligare inbladning av användaren. Om konfiguration med DHCP misslyckas sätts instrumentet i ett läge där användaren via windowsprogrammet kan konfigurera nätverket för hand.

Den andra funktionen är upptäckts och konfigureringsfunktionalitet. Det går ut på att användare via PC-programmet kan söka efter alla instrument som finns inkopplade på nätverket, och konfigurera de som ännu inte är konfigurerade. Instrumentet lyssnar hela tiden efter sådana sökpaket, och svarar på dessa. Okonfigurerade instrument lyssnar även de efter konfigurationspaket. Windowsprogrammet använder denna mekanism för att presentera en lista på alla instrument som finns tillgängliga på nätverket och ge användaren möjlighet att välja vilken av dem som skall avläsas.

Tredje delfunktionen är i form av en strömmande uppkoppling. När ett program kopplar upp sig mot denna funktionalitet hos instrumentet börjar instrumentet strömma ut mätdata ca en gång i sekunden. Windowsprogrammet tar emot denna data och presenterar den på skärmen för användaren i form av siffror och kurvor.

Den fjärde delfunktionen är i form av en webbserver. Vid uppkoppling mot denna

skickas en HTML-sida med mätdata ut som kan ses i standardwebbläsare. Generering av denna sida tar ca tio sekunder.

För detaljer om kommunikationsprotokoll se C.1 och C.2.

4.3 Återanvändbara moduler

Vissa av modulerna i projektet är inte begränsade i sitt användningsområde till bara detta projekt. Delar av den hårdvara vi fick tillgänglig för uppgiften har inte tidigare använts av uppdragsgivaren. Därför behövde nya drivrutiner skrivas, drivrutiner som är generella nog att användas i andra projekt. Det är två av dessa moduler som är intressanta här.

4.3.1 Display

För presentation av mätdata tillhandahölls en grafisk LCD-display med en upplösning av 128x64 pixlar. Drivrutinen för denna display tillhandahåller initialiseringar, pixelvis uppritning och utmatning av olika former av text. För detaljer se B.4.6.

4.3.2 Wiznet TCP/IP-modul

För kommunikation över nätverk tillhandahölls en Nätverksmodul av fabrikat Wiznet. Denna tillhandahåler möjligheter för ethernet, IP, TCP och UDP- kommunikation. Genom att använda denna modul minskas bördan för vår huvudprocessor, och den mängd kod vi måste skriva, dramatiskt. Men det behövs ändå någon form av drivrutiner för denna. Vår drivrutin implementerar initialisering av hårdvaran och dubbelriktad kommunikation både via UDP och TCP. Se B.4.7 för detaljer.

4.4 Dokumentation

5 Diskussion och slutsatser

I detta kapitel återanknyts lösningen med ursprungsproblemet. Hur väl uppgiften lösts och vad det finns för alternativ till vår lösning diskuteras.

5.1 Lösningen och kraven

Här gås de specificerade kraven igenom en och en och huruvida vårt system motsvarar kraven diskuteras.

Ett krav är att systemet skall kunna presentera den data som deras tidigare version av instrumentet klarar av. Detta innebär enkelt kontrollerbara krav som att systemet kan leverera medelspänning och medelström i sifferform, vilket vår lösning klarar av.

Förutom denna funktionalitet skall systemet kunna visa kurvor som presenterar spänningen och strömmen i signalen över tid. För att svara upp mot detta krav visar instrumentet kurvorna för en signalperiod.

Specifikationen sade även att systemet skulle kunna fjärrstyras över nätverk. Några avancerade styrmöjligheter för instrumentet finns inte så vi tolkade kravet som att all mätdata skall kunna föras över via nätverk till lämplig mottagare på en vanlig PC. Detta löses genom nätverkskommunikation mellan instrumentet och det windowsprogram som ingår i systemet. All data som användare kan få från instrumentet kan avläsas via nätverkskommunikation. Begränsningen är här endast i uppdateringsfrekvens, då uppdateringarna över nätverk sker mer sällan än på instrumentets egen display.

5.2 Hur väl löser systemet uppgiften?

Med de formella kraven uppfyllda kan det vara upplysande att se hur väl systemet löser uppgiften från en användares synvinkel. Som beskrivet i delkapitlet målgrupp (2.1.1) finns

det två grupper av användare att ta hänsyn till. Utvecklare av system som skall styra ventiler, och användare av system där ventilstyrning krånglar i produktionssammanhang.

Om vi först riktar uppmärksamheten mot en användare av ventilsystem i produktionssammanhang så är det huvudsakliga kravet att instrumentet kan göras kompakt och enkelt att använda. Vi har inte sett hårdvaran i slutmonterat skick, men baserat på den hårdvara som ingår i systemet räknar vi med att instrumentet kan tillverkas i en storlek motsvarande en normal räknedosa. Detta instrument blir då ett instrument bland andra standardinstrument man kan använda vid felsökning av elektriskt styrda maskiner. Vi gör bedömningen att detta för denna grupp användare ökar smidigheten betydligt jämfört med de alternativ som fanns tidigare (oscilloskop).

Om vi betraktar användargruppen utvecklare av ventilstyrningssystem så är vinsten för dem att de via nätverksfunktionalitet kan få upp mätdata på samma skärm som de utvecklar systemet på. Instrumentets storlek är dock inte av samma vikt för denna användargrupp, och hur stor praktisk effekt denna lösning skulle ha över den information utvecklare kan få genom att läsa av ett oscilloscop i närheten kan vi inte uttala oss om. Möjligen skulle det kunna finnas situationer där systemet är inkopplat på annan plats än utvecklarens arbetsrum, och i det fallet så skulle systemet kunna vara användbart. Men vi har ingen erfarenhet av hur sådan utveckling går till.

5.3 Begränsningar i lösning och alternativa lösningar

I anslutning till systemets nätverksfunktionalitet finns det flera lösningar som var och en har sina begränsningar. De begränsningar som finns och olika alternativ för implementation diskuteras här.

5.3.1 Allockering av nätverksresurser

Systemet är begränsat av den hårdvara som används för att tillhandahålla nätverksfunktionalitet till ett maximum av fyra samtidiga kanaler. Det vill säga endast fyra uppkopplingar kan

göras samtidigt. Dessa fyra kanaler har vi allockerat en var till vardera nätverksfunktionalitet identifierad i kapitlet nätverksfunktionalitet. Att endast en kanal är avsatt till vardera funktionalitet innebär att endast en klient i taget kan hämta data från instrumentet. För att komma runt denna begränsning skulle man kunna tilldela fler kanaler till den funktionaliteten, med en design som minskar behovet av andra kanaler, alternativt byta från en strömmande (TCP) uppkoppling till en datagramuppkoppling där instrumentet håller reda på alla det skall skicka mätdata till.

5.3.2 System för överföring och presentation av mätdata

Det finns flera sätt att lösa problemet med att föra över och presentera data på en dator. Vår lösning använder ett specialgjort windowsprogram som kommunicerar med instrumentet. Problemet med denna lösning är att användare måste hålla sig med detta program på alla platser där de vill använda denna funktionalitet. Användaren kan inte ta med sig enbart instrumentet till annan plats och ha tillgång till denna funktionalitet. Som alternativ diskuterades att ha applikationen i form av en java-applet i instrumentet som kunde överföras av instrumentet till måldatorn.

Även en sådan lösning skulle ha problem. Det windowsprogram som skapats tar även hand om konfiguration av instrumentets nätverksinställningar. En javaapplet-lösning skulle kräva att detta konfigurationsproblem antingen löstes på annat sätt (t ex genom att utöka kontrollerna på själva instrumentet för att mata in inställningarna där), eller genom att begränsa instrumentets funktionalitet till endast nätverk med DHCP-server.

Vårt val avgjordes av den bedömning vi gjorde av tidsåtgången för att implementera de olika lösningarna. Det skulle dock kunna vara intressant att som vidareutveckling ta bort behovet på ett separat program från systemet.

6 Projektberättelse

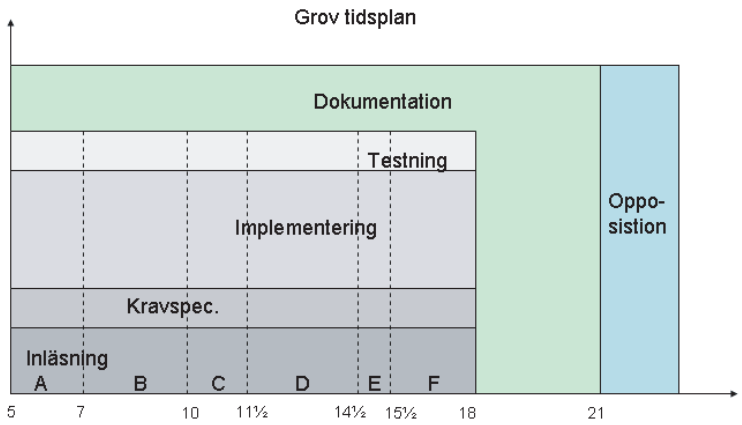
Här beskrivs arbetets förlopp uppdelat i stort sett i kronologisk ordning.

6.1 Grovplanering

Uppgiften var inte specificerad i detalj, så vi hade lite svängrum i implementeringen av den specificerade funktionaliteten. Vi identifierade i stort de grundläggande uppgifterna:

- Inläsning av mätdata
- Presentation av mätdata på display
- Kontroll med hjälp av knappar
- Möjlighet till nätverkskommunikation för instrument
- Windowskod för kommunikation med instrument
- Windowsskal för presentation av mätdata och kontroll av instrument

Vi gjorde en grov tidsplan uppdelad för på dessa uppgifter (och dokumentation):



Figur 6.1: Ursprunglig tidsplan

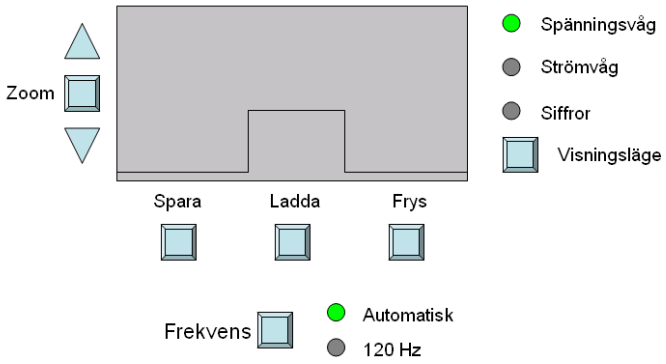
Planeringen är alltså uppdelad relativt den funktionalitet som skall levereras. I varje del ingår aktiviteter som inläsning på hårdvaruspecifikationer, specificering av nödvändig funktionalitet som programmodulen skall leverera och sedan implementation.

Utförandet blev inte helt enligt ursprunglig plan då vi från uppdragsgivare fick tillgång till lite olika hårdvara då och då och inte i samma ordning som vi planerat.

6.2 Inledande undersökning

Den första veckan ägnade vi oss åt att bekanta oss med arbetsverktygen och att sätta oss in i vad själva uppgiften gick ut på. Vi kollade upp hur utvecklingskortet fungerade och så skapade vi några enkla testprogram för att bekanta oss med utvecklingsmiljön. Testprogrammen vi gjorde var en föreberedelse inför första uppgiften och gick ut på att vi läste in analoga värden, A/D omvandlade dessa och visade upp resultatet med hjälp av lysdioder. Vi gjorde också så att programmet samplar några värden i en serie och spelar sedan upp denna serie på lysdioderna.

Vi gjorde också ett förslag till möjlig funktionalitet på instrumentet, dock blev endast en del av dessa, här är det första utkastet av förslaget.



Figur 6.2: Första förslag till instrumentfunktionalitet

6.3 Nätverkskommunikation

Den första hårdvara som vi fick tillgång till var WizNet TCP/IP modul, den kommunicerade med PIC-processorn genom I2C-kommunikation. Vi lyckades ganska snabbt med att få igång I2C-kommunikationen och initiera WizNet modulen med ett IP-nummer så att den svarade på ping. Vi hade dock problem med överförning av data, vi jobbade länge på att lösa problemet men vi kunde inte finna något fel. Vi läste dokumentation, exempelkod och inlägg på WizNet's forum. Efter ca två veckor gav vi upp med att försöka få igång nätverkskommunikationen.

6.4 Mätdata och indata

Efter att vi misslyckats med att få igång TCP/IP modulen började vi skriva kod för styrning via knappar, denna uppgift var enkel att lösa. Vi gjorde också en funktionsgenerator så att vi kunde testa inläsningen av PWM-signal, men dagen därpå fick vi en riktig hydraulventil inkopplad så vi kunde verfiera att denna kod fungerade. Implementationen för inmatning via knappar färdigställdes.

6.5 Grafisk display

Fem veckor in i projektet fick vi tillgång till en grafisk 128x64 pixlars display och vi kunde redan efter två dagar börja rita upp saker på den, vi hade dock problem med att det blev underliga uppritningsfel som vi inte kunde förstå. Vi fixade också teckensnitt och drivrutiner för att rita upp dessa. Två veckor jobbade vi på att fixa uppritningsfelen men vi kunde inte hitta något fel, vi läste dokumentation, sökte på internet och vår uppdragsgivare ringde och pratade med leverantören. Efter två veckor utan framsteg fick vi en alternativ hårdvaruinkoppling av displayen och då fungerade allt felfritt, det visade sig att programkoden hade varit rätt hela tiden och att det var hårdvaruinkopplingen som strulade. Det kändes riktigt bra att vi äntligen fick igång någon extern hårdvara efter flera

veckors misslyckanden med både nätverkskommunikation och display.

6.6 Nätverkskommunikation, andra försöket

Vi gjorde ett andra försök med att få igång TCP/IP modulen och efter en vecka lyckades vi lösa våra tidigare problem och nätverkskommunikationen fungerade. Därefter delade vi upp arbetsuppgifterna så att Andreas implementerade bland annat DHCP protokol och Jonas gjorde ett windowsprogram som skulle kommunicera med mätinstrumentet.

6.7 Funktionalitet

Den faktiska funktionalitet av mätinstrumentet gick väldigt snabbt att implementera. När vi väl fått våra hårdvarudrivrutiner att fungera så behövde vi bara koppla ihop alla delar och vi hade en färdig prototyp av mätinstrumentet. Den sista mars var sista dagen vi var på ÅF och då provkörde vi mätinstrumentet och windowsprogrammet. Allt fungerade som det skulle.

6.8 Dokumentation

De första dokument som producerades var planeringsdokument och den exjobbsspecifikation som låg till grund för vår planering.

Löpande under hela projektets gång har en dagbok förts. I den har en kortfattad beskrivning av varje dags aktiviteter noterats tillsammans med tid spenderad. Dagboken har legat till grund för stora delar av denna rapport. Rapporten påbörjades efter ungefär sju veckor in i arbetet, men den allra största delen av dokumentationen skrevs efter att implementationen var färdig.

6.9 Reflektion

Vi är nöjda med det vi kan leverera som ett resultat av vårt examensarbete. Vägen fram har inte varit perfekt med flera tekniska problem som tagit mycket tid. Problemen löstes dock och systemet löser problemet på ett väl användbart sätt.

Under hela arbetets gång hade vi tillgång till uppdragsgivaren som vi kunde kontakta när det gällde problem eller för att kontrollera huruvida vår lösning var på väg åt rätt håll. Detta avspeglades i vår arbetsprocess på så vis att det var ingen noggrann planering i början som avstämdes mot uppdragsgivaren utan resultaten presenterades allteftersom de blev klara. Synpunkter på lösningen togs löpande emot och beaktades i det fortsatta arbetet. Hade vi inte haft samma tillgänglighet till uppdragsgivaren hade en noggrannare planering i arbetets startskede varit nödvändigt.

Bibliografi

[1] Nigel Gardner, *PICmicro MCU C: An introduction to Programming the Microchip PIC in CCS C*, Bluebird Electronics, 19 Augusti 2002, ISBN: 0972418105.

[2] W. Richard Stevens, *UNIX network programming; Volume 1, 2nd edition*, Prentice Hall, 1998, ISBN 0-13-490012-X.

[3] Herbert Schildt och Frank Crockett, *MFC Programming from the Ground Up*, McGraw-Hill Osborne Media, Augusti 1998, ISBN 0078825733.

[4] Microsoft, *MSDN - MFC dokumentation*, `http://msdn.microsoft.com`; läst 2004-03-30.

[5] Wiznet, *Exempel kod och dokumentation*, 2003-04-14, `http://www.wiznet.co.kr/e_iinchip/technical_download.htm`; hämtad 2004-02-12.

[6] R. Droms, *RFC2131 - Dynamic Host Configuration Protocol*, Mars 1997, `http://www.faqs.org/rfcs/rfc2131.html`; hämtad 2004-03-30.

[7] S. Alexander, Silicon Graphics Inc. och R. Droms, *RFC2132 - Dynamic Host Configuration Protocol*, Mars 1997, `http://www.faqs.org/rfcs/rfc2132.html`; hämtad 2004-03-30.

[8] Tianma Microelectronics Co., ltd, *Dokumentation för display*, `http://www.tianma.com/spec_sheets/TM12864NCCG%20SPEC.pdf`; hämtad 2004-02-26.

[9] Microchip Technology Inc., *PIC18FXX8 Data Sheet*, 2003, `http://ww1.microchip.com/downloads/en/DeviceDoc/41159c.pdf`; hämtad 2004-02-26.

A Terminologi och begrepp

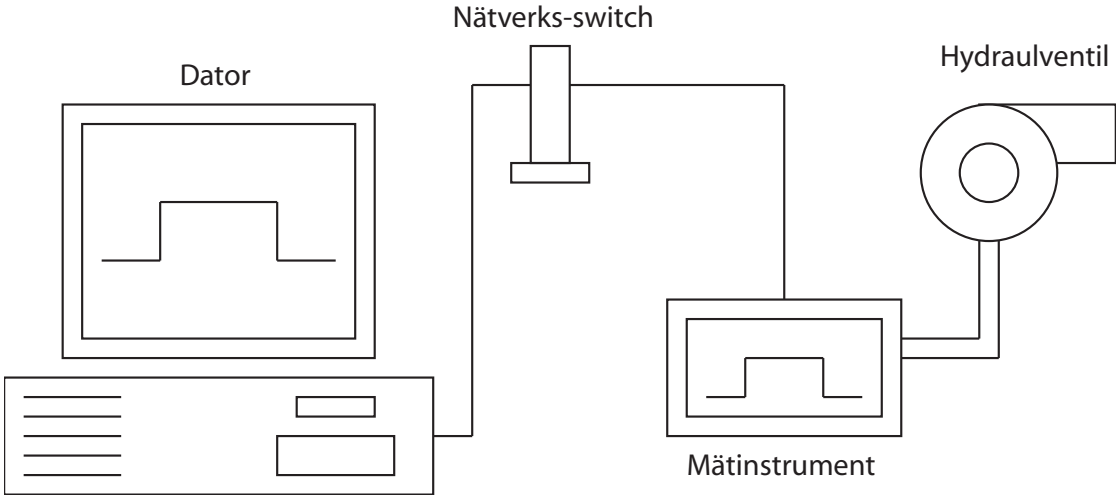
- **A/D omvandlare** - En elektrisk krets som konverterar en analog signal till bitar så att den kan representeras digitalt.
- **Arbetsperiod** - Arbetsperioden är den andel av en signal som är hög. Om t ex en PWM-signal är hög i en ms och låg i tre ms så är arbetsperioden 25%.
- **Broadcast** är ett utskick av UDP/IP data till en speciell ip-adress (broadcast adress). Paktetet som skickas tas emot av alla klienter som befinner sig på samma lokala nät som sändaren.
- **DHCP** står för Dynamic Host Configuration Protocol och är ett protokol som används för automatisk konfigurering av bland annat ip-nummer på klienter som ansluter sig till ett nätverk.
- **Drivrutin** är den del av program som tillhanda håller styrning av hårdvara och fungerar som ett gränssnitt mellan hårdvaran och övrig mjukvara.
- **FIFO** Benämning på kösystem,First In First Out. Det som kommer in först i kön hämtas först när kön läses.
- **Funktionsgenerator** - Ett instrument som kan generera elektriska signaler med ett visst utseende som går att ställa in, t ex sinuskurva, fyrkantsvåg. Man kan också ange t ex frekvens och spänning.
- **Hydraulventil** Ventil vars läge styrs hydrauliskt (t ex genom trycksättning av vätska).
- **Hårdvarunära programmering** - Tillverkandet av program som skall köras direkt mot hårdvara, utan något mellanliggande operativsystem.
- **I2C** - Ett standardiserat protokoll för seriell kommunikation.

- **Kodgenererande script** - Ett script som skapar källkod. Används med fördel till redundanta uppgifter så som att konvertera binär data till en array som kan användas i C kod.
- **Mätdata** - Information som innehåller resultatet från någon typ av mätning.
- **Oscilloskop** - Ett instrument som kan mäta bland annat ström och spänning och rita upp kurvorna för dessa på en display.
- **PIC-processor** - En processor speciellt utvecklad för att användas i inbyggda system. Finns i många modeller och storlekar.
- **PWM-signal** - PWM står för Pulse Width Modulator, plusbreddsmodulator på svenska. Signalen är en fyrkantvåg med två nivåer (hög och låg) där alla pulser kommer med ett fast intervall. Tidslängden eller bredden på pulsen representerar själva informationen som skickas.
- **Script** - Program som kan köras utan att först kompileras, koden interpreteras under körning.

B Teknisk beskrivning

Här följer teknisk dokumentation för projektet. Första delkapitlet går igenom systemet i stort och hur de olika delarna sitter samman. I systemöversikten tas också kommunikation mot omvärlden utanför systemet upp. Andra delkapitlet går igenom det som hårdvarudesignern behöver veta om styrprogrammet för att konstruera ett komplett instrument. Tredje delkapitlet dokumenterar de enskilda modulerna i systemet.

B.1 Systemöversikt



Figur B.1: Systemsammansättning

Bilden visar det kompletta systemet i översiktsform. Det finns en styrsignal, i det här fallet kopplad från en ventil, som mäts. Den kopplas in till mottagare i mätinstrumentet. Resultatet visas på display. Instrumentet kopplas in till lokalt nätverk och kan på så vis föra över mätinformation till en vanlig PC.

B.2 Beröringspunkter med omvärlden

Det finns flera delar i systemet som kommunicerar med omvärlden:

- Instrumentet tar in en signal som skall mätas.
- Instrumentet visar upp mätresultat på display.
- Användaren påverkar uppvisningen med hjälp av knappar.
- Mätdata visas på skärmen på en PC, med påverkan av vilket instruments data som visas.
- Vid uppstart försöker instrumentet konfigurera sin nätverksinformation med hjälp av DHCP.

De fyra första punkterna berör en användares interaktion med systemet och tas upp separat. Det som en användare inte har direkt kontroll över är när instrumentet initialiserar sig innan det påbörjar sina mätningar.

B.2.1 Initiering med DHCP

När instrumentet startas upp skickar det ut ett DHCPDISCOVER-meddelande med broadcast. Instrumentet väntar i upp till fem sekunder på att få ett DHCPOFFER-meddelande som svar från en DHCP-server. Får instrumentet ett sådant svar skickar det ett DHCPREQUEST-meddelande för att reservera informationen. Instrumentet väntar på ett DHCPACK-meddelande i upp till fem sekunder. När detta meddelande mottagits konfigurerar instrumentet sina parametrar enligt den information som DHCP-servern skickat. Misslyckas DHCP-initieringen på något vis under initieringsfasen initierar instrumentet sin IP-adress till 0.0.0.0 och sätter en flagga som noterar att initiering krävs.

Instrumentet begär en lease-tid på ca 12000 sekunder för sitt tilldelade ipnummer. När halva denna tid gått försöker instrumentet förlänga sin leasetid med en ny omgång av DHCP-konfigurering.

B.2.2 Initiering av ethernet-adress

Alla nätverksenheter behöver ha en unik ethernet-adress, så även vårt instrument. Ethernetadressen skall programmeras in i enheten, och måste då vara ett korrekt unikt nummer. Ethernetadresser delas ut av IETF antingen i block om 16.7 miljoner (Organisationally Unique Identifier, OUI) eller i block om 4096 adresser (Individually Adress Block). Uppdragsgivaren har varken något OUI eller IAB-block. Om instrumentet skall tillverkas i den form som det utvecklats av oss så måste Antingen ett OUI- eller IAB-block införskaffas. Konstanterna ETHADR1 till ETHADR4 skall ställas till de fyra första bytevärdena i använd ethernetadressrymd. De två sista byten läses från en fil (serial.num) som skall innehålla ett tal i klartext i hexadecimal följt av CR/LF (dvs nyrad). För varje ny tillverkad enhet skall då innehållet i serial.num ändras innan mjukvara laddas in i enheten.

B.3 Hårdvaruinkoppling

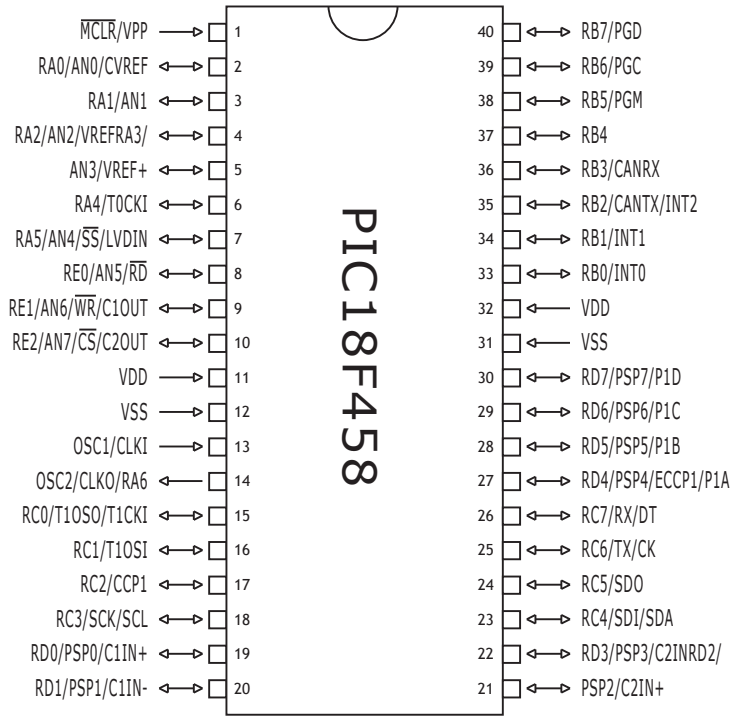
Här dokumenteras de beslut i design av programmet för instrumentet som påverkar hårdvaruinkoppling och design.

B.3.1 Hårdvaruval

Programmet är skrivet för Microchip Technology Inc.:s processor PIC18F458. De tidskänsliga funktionerna är anpassade för att denna enhet skall köras med en frekvens av 20.0 Mhz. Nätverksdrivrutinen är anpassad för hårdvaruenheten Wiznet IIM7010A. Displaydrivrutinen är anpassad för KS0724 displaycontroller till display TIANMA12864NCCG eller liknande.Drivrutin för knappinmatning är anpassad för en knapp som ger låg signal när den är intryckt och hög signal annars.

B.3.2 Inkoppling

I de följande delkapitlen beskrivs inkopplingarna som behöver göras hårdvarumässigt från processor till övrig hårdvara pinne för pinne.



Figur B.2: PIC18F458

B.3.3 Inläsning

Signalen som skall mätas skall kopplas till två pinnar på processorn: Spänningsdelen av signalen skall skall kopplas mot RA0/AN0, strömdelen översatt till spänning skall kopplas in mot pinne RA1/AN1.

B.3.4 Knappar

Två knappar ingår i designen. Knapp för växling av presentationsläge på display skall kopplas in på pinne RD1. Knapp för frysning av visad data skall kopplas in mot pinne RD0.

B.3.5 Display

Displayenheten skall kopplas in med fem pinnar:

- Register Select skall kopplas in på RC0.
- Chip Select (CS1B/CS1 på display) skall kopplas in på RC1.
- Hårdvarureset skall kopplas in på RC2.
- SCLK (seriell klocka) skall kopplas in på RC5.
- SID (seriell data) skall kopplas in på RC6.

Dessa inkopplingar kan ändras med hjälp av konstanter i lcd_driver.h

B.3.6 Wiznet TCP/IP-modul

Wiznet-modulen skall kopplas så att den används i i2c-läge. Kopplingarna mot processor är som följer:

- SDA (seriell data) på pinne RC4.
- SCL (seriell klocka) på pinne RC3
- Reset (aktiv hög) på pinne RD5.
- Reset (aktiv låg) på pinne RD6.
- Interrupt (avbrott) på pinne RB1/INT1

De två resetinkopplingarna är inte nödvändiga om reset erhålls på annat sätt. Reset måste i sådana fall genomföras omedelbart efter uppstart av instrumentet.

B.3.7 Övriga inkopplingar

Huruvida enheten försöker konfigurera sig med hjälp av DHCP eller inte avgörs av nivån på pinne RD7. Om man önskar att enheten skall försöka konfigurera sig med hjälp av DHCP skall pinne D7 ges hög nivå vid uppstart. Om man inte önskar DHCP-funktionalitet skall D7 hållas låg vid uppstart av instrumentet.

B.4 Komponentdokumentation

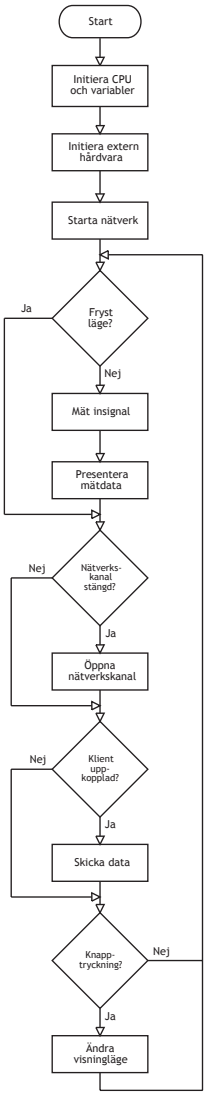
Här beskrivs varje modul programmet består av var för sig. Modulens arbetsgång och eventuella gränssnitt utåt beskrivs, samt eventuella detaljer viktiga att veta för underhåll och förändringar i modulen.

B.4.1 Huvudprogram

Figuren visar huvudprogrammets flöde. Programmet börjar med initieringar av processor och inkopplad hårdvara. I programmets huvudloop anropas först funktioner för inläsning av mätdata. Dessa presenteras sedan på display. Nätverkskanalerna kontrolleras, och eventuella stängda kanaler öppnas. Sedan kontrolleras om några klienter är uppkopplade mot någon av nätverkskanalerna och om detta är fallet så skickas nödvändiga svar ut. Sist kontrolleras om det finns någon knappinmatning på kö. Om detta finns ändras presentationsinställningarna på motsvarande sätt.

B.4.2 Inläsning av analog signal

Inläsningsmodulen exporterar funktionen measure. Denna funktion har ett flöde som motsvarar figur.



Figur B.3: Flödesschema

Först anropas en funktion som mäter insignalens frekvens. Denna frekvens räknas om till värden som motsvarar vilken takt signalen skall samplas i. Efter detta synkroniseras inläsning så att signalens intressanta del (högperioden) är centrerad över mätningen. Mätsignalen samplas i jämn takt 2x128 gånger. Dvs först samplas spänningsdelen 128 gånger. Direkt efteråt samplas strömdelen av signalen 128 gånger.

Detektering av frekvens och centrering av mätsignal fungerar för signaler mellan 50-150Hz. Misslyckas detektering av frekvens genomförs mätningarna som om signalen hade en frekvens på 120 Hz.

För att få korrekt mätdata är det viktigt att rutinen inte avbryts under någon längre period. Av denna anledning används samma klocka i processorn som styr de regelbundna tidsavbrotten till att mäta tid för signalen. Genom den nollställning saom genomförs av denna signal garanteras att inga tidsavbrott sker under tiden som frekvensdetektering och sampling pågår.

B.4.3 Utskriftsrutiner (för display)

Presentationsrutinerna är de som använder lcd-drivrutinen till att skicka ut data på display. All kalkylering av hur mätdata skall transformeras till kurvor, och hur siffror skall skrivas ut genomförs i dessa rutiner. Presentdata exporterar funktionen:

void present_data(char frozen)

Funktionen kontrollerar vilket presentationsläge programmet är i och anropar rätt utskriftsrutiner för det läget.

B.4.4 Nätverksrutiner

Nätverksmodulen exporterar funktioner för initiering av nätverksenhet och Initiering av nätverkskanaler. De olika former av nätverkskommunikation som fins har implementerats i funktioner i denna modul. Funktioner för att läsa HTTP och DHCP-paket, samt funktioner för de egendefinierade protokoll som används för kommunikation mellan instrument och

PC finns definierade här.

De funktioner som används externt är:

- start_network(flag) Initierar nätverksinställningar, (ipadress etc.). flag är antingen FIRST_INIT om det är första gången systemets nätverksinställningar initieras i denna körning. Om det inte är första gången skall flag vara av värde RENEW_INIT.
- init_tcp_sock() startar en nätverkskanal för kommunikation med PC-klient.
- init_web_sock() startar nätverkskanal för kommunikation med webbläsare.
- analyze_udp() Funktion för kommunikation med dator. Den del av kommunikationen där datorn söker efter och konfigurerar instrument på nätverk är implementerad här.
- web_recv_and_send() Funktion för kommunikation med webbläsare med hjälp av HTTP-protokoll.

Systemet använder fyra olika nätverkskanaler:

- DHCP-kanalen. Denna är inställd i UDP-läge och lyssnar på port 68. Kanalen används endast till att erhålla konfigurationsinformation från eventuella DHCP-servrar.
- Konfigurationskanal. Denna kanal är inställd i UDP-läge och lyssnar på port 8738. Den används dels till att svara på sökanrop från andra program (när andra enheter på nätverket letar efter instrument) och för överföring av konfigurationsinformation från dator, oftast via användare.
- Datakanal. Denna kanal är inställd i TCP-läge och lyssnar på port 8738. När något externt program har en uppkoppling mot denna kanal överför instrumentet med jämna mellanrum (ca 1 gång i sekunden) mätinformation. Överföringarna fortsätter tills klienten stänger ner uppkopplingen eller instrumentet stängs av.

- HTTP-kanal. Denna kanal är inställd i TCP-läge och lyssnar på port 80. Instrumentet lyssnar på denna kanal efter HTTP GET förfrågningar och genererar som svar en HTML-sida som innehåller mätvärden samt en BMP-bild med kurvformer. Generering av sidan och bilden tar ca 10 sekunder, och under denna tid är instrumentet "fryst".

Datan överförd på två av dessa kanaler har egendefinierade format som beskrivs i de två nästföljande avsnitt.

B.4.5 Drivrutin för knappar

Drivrutinen för knapparna sköter inmatning från upp till åtta knappar. Antalet som används ställs in i en konstant i buttons.h. Drivrutinen lagrar knappinmatningar i en FIFO kö som kan läsas av utifrån. läsningsrutinen genomför "debounce" på knappar, så hårdvaran behöver inte vara studsfri. Denna studshantering tar dock 10ms vilket innebär att rutinen som kontrollerar knapparna inte skall anropas i situationer där minimal tidsåtgång är nödvändig. Drivrutinen håller reda på när en knapp är nedtryckt och när den släpps upp igen och registrerar bara en inmatning per nedtryckning av en knapp.

Drivrutinen använder de lägsta bitarna i port D, dvs med de två knappar som instrumentdesignen har så använder drivrutinen pinnar D0 och D1.

Drivrutinen exporterar följande funktioner:

- init_buttons() Nollställer inmatningskö och statusinformation för knappar, måste anropas innan någon annan knapphanteringsfunktion.

- check_ctrl() Kontrollerar knapparnas status och lägger till knapptryckningar i inmatningskön. Skall anropas ungefär 50 ggr per sekund för att få en acceptabel knapphantering.

- getkey() Hämtar en knapptryckning från inmatningskön. Returnerar noll om kön är tom.

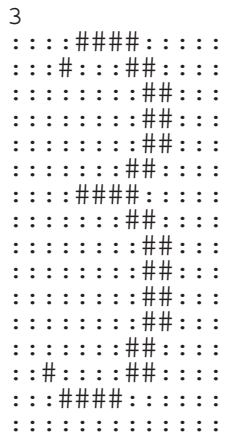
B.4.6 Drivrutin för display

Drivrutinen tillhandahåller funktioner för att initiera display, rita pixelgrafik, samt lägga ut text. Drivrutinen ritar huvudsakligen i en framebuffer som är lokal för processorn. Hela denna buffer skickas ut till display vid anrop till update_lcd. Detta ställer krav på att processorn skall ha åtminstonde 1024 byte tillgängligt för en sådan buffer. De viktigaste exporterade funktionerna är:

- disp_init() Initierar hårdvaran. Måste anropas före någon annan funktion i drivrutinen.
- disp_clear() Rensar framebuffern. Dvs inga pixlar satta.
- disp_update() Skickar ut innehåll i framebuffer till display.
- dwrite(char page, char col, char data) Skriver åtta pixlar direkt till display (inte till framebuffer). Åtta pixlar på höjden skrivs, där värdet på page avgör vilka åtta pixlar det är. Page har värde 0-7. Col är vilken kolumn som skall skrivas. Värdet på col kan vara 0-127. Data är en byte sär varje bit motsvarar en pixel på display.
- disp_plot(x, y, c) Ställer in en pixel i framebuffer. x är kolumn (0-127), y är rad (0-63) och c är vad pixeln skall sättas till. Om c är noll rensas pixel. Om c är skilt från noll sätts pixeln.
- putlogo() Ritar ut en bild som är 64x64 pixlar stor. Bilden är definierad i filen logo.h

Se headerfile lcd_driver.h för fullständig lista.

I drivrutinen ingår rutiner för att skriva ut tecken i olika storlekar. För att underlätta hantering och ändring av den delen i drivrutinen har ett speciellt system utvecklats. En uppsättning Perlscript har skrivits som omvandlar vanliga textfiler bestående av block av : # som beskriver tecknen till siffertabeller som passar till fontutritningsrutinerna. T ex ser definitionen av 3 ut såhär i font16.txt:



Figur B.4: Exempel på fontdefinition

B.4.7 Drivrutin för nätverksmodul

Drivrutinen är skriven för Wiznet IIM7010A hårdvarumodul och kräver att den är i läge för kommunikation med processor med hjälp av i2c-protokollet. Drivrutinen använder en avbrottsfunktion kopplad mot avbrott 1 på processorn. För korrekt funktion av denna avbrottsrutin måste processorn konfigureras av huvudprogrammet så att interrupt ges vid övergång från hög till låg nivå.

Drivrutinen tillhandahåller funktioner för initiering av wiznetmodulen, konfigurering av kanaler i TCP eller UDP-läge samt funktioner för att skicka och ta emot data i båda dessa lägen. Det fattas funktion för att själv initiera en TCP-uppkoppling för att drivrutinen skall ha komplett funktionalitet.

Det som exporteras från drivrutin är följande: Global vektor char socket_state[] med fyra platser (en för varje kanal) ur vilken kanalens läge kan utläsas. SOCK_CLOSED innebär att kanalen är stängd och behöver initieras igen. SOCK_ESTABLISHED innebär att uppkoppling inkommit etc. Se headerfil wiznet.h för lista på konstanter.En annan global vektor socket_data håller reda på antalet väntande datapaket som finns att läsa på en datakanal.

Funktioner:

- wiznet_reset() , genomför hårdvaru- och mjukvarureset. Måste vara den första funktion som anropas i drivrutin.

- wiznet_init_addr(smr, shar, sipr, gar), initierar modulens adressinställningar med Subnet mask SMR, etyhernet adress shar, ip adress sipr och router gar. Samtliga variabler är pekare till char-vektorer. Denna funktion skall anropas efter reset och före initieringar av kommunikationskanaler.

- wiznet_init_sock(channel, protocol, myport). Öppnar en kommunikationskanal. channel skall vara ett värde från 0 till 3, och vara en unik kanalnyckel, dvs två kanaler kan inte ha samma nummer. Protocol är antingen PROTO_TCP eller PROTO_UDP. Myport är vilken port som kanalen skall vara kopplad till. myport är en pekare till en vektor med två byte som tillsammans utgör 16-bitarstalet som är portnumret. Numret är i nätverksordning, dvs tvärtom vad processorn arbetar i.

- wiznet_send_udp_data(channel, destip, destport, data, len). Skickar upp till len byte pekad ut av data till destport på destinationsplats destip. Begränsad till att skicka max 2kilobyte data i taget.

- wiznet_send_tcp_data(channel, data, len). Skickar len byte utpekad av data över en TCP-kanal. Begränsad till att skicka 2kilobyte data i taget.

- wiznet_read_udp_data(channel, buf, len). Läser upp till len byte från kanal till utrymme utpekat av buf. Returnerar antalet byte som lästs.

- wiznet_read_tcp_data(channel, buf, len). Läser upp till len byte från kanal till utrymme utpekat av buf. Returnerar antalet byte som lästs.

- wiznet_listen(channel). Sätter en TCP-kanal i läge för att lyssna efter inkommande uppkopplingar.

- wiznet_close(channel). Stänger en kanal, och därmed eventuell uppkoppling mot klient.

B.4.8 Windowsprogram

Windowsprogrammet har skapats i C++ med Visual Studio. Funktionaliteten är byggd ovanpå MFC (Microsoft Foundation Classes) med hjälp av den kod som automatiskt genereras av Visual Studio.

C Protokollbeskrivning

C.1 Protokoll för datakanal

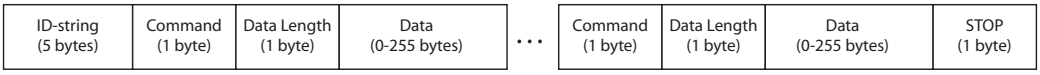
Först kommer en tre bytes signatur som består av tre %-tecken. Sedan följer 128 byte som innehåller 128 samples från mätning av spänning. Varje sample är mellan 0 och 255. Nästa 128 byte är motsvarande data för mätningar av strömnivåerna på signalen. Sist ligger fyra byte som är varsin konstant över hur instrumentet är inställt. Dessa motsvarar vilka spännings respektive strömvärden instrumentet behandlar extremvärdena 0 respektive 255. Samtliga konstanter kan ha värden mellan -128 till 127. Dessa konstanter är i ordning:

- low_v_const. Hur många volt motsvarar mätvärde 0.
- high_v_const. Hur många volt motsvarar mätvärde 255.
- low_i_const. Hur många tiotal milliampere motsvarar mätvärde 0.
- high_i_const. Hur många tiotal milliampere motsvarar mätvärde 255.

Detta innebär att om ändringar genomförs i instrumentets hårdvara i omvandlingen av mätsignal till de 0-5V som processorn kan läsa så skall inställningarna endast ändras i instrumentets mjukvara. Eventuella program som läser från instrumentet skall använda dessa konstanter som överförs vid beräkning av faktiska mätvärden från de uppmätta.

C.2 Protokoll för kommunikationskanal

Formatet i överföringarna till och från konfigurationskanalen visas i figur C.1.



Figur C.1: Överförningsprotokoll

Alla paket till och från instrumentet följer detta format, och ett paket kan innehålla ett eller flera kommandon. Mätinstrumentet och datorn har olika kommandon som är tillåtna. Tabell C.1 visar vilka kommadon som är tillåtna. “ID-String” innehåller alltid texten “AFPWM”.

Sändare	Namn	Kommando	Datastorlek
PC	DISCOVER	0x01	0
PC	SET_IP	0x02	18
Instrument	VERSION	0x03	2
Instrument	IP_PORT	0x04	6
Instrument	HW	0x05	6
Instrument	STATUS	0x06	1
Båda	STOP	0xFF	-

Tabell C.1: Kommandolista för kommunikationsprotokollet

(Observera att kommandot STOP har inget fält för storlek eller data.)

Alla paket från PC ska broadcastas och alla paket från instrumentet skall skickas endast vid mottagande av broadcast. Paketen från instrumentet skickas till broadcastarens IP-adress och source port.

C.2.1 DISCOVER kommandot

Broadcastas från PC, paktetet beskrivs i figur C.2.

ID	DISCOVER	Datastorlek	STOP
AFPWM	0x01	0	0xFF

Figur C.2: DISCOVER kommandot

C.2.2 SET_IP kommandot

Broadcastas från PC för att konfiguera IP-adress för ett mätinstrument som har en viss MAC-adress. Innehåller 18 bytes data. Om vi antar att värdena i tabell C.2 gäller.

MAC-adress:	0xAABBCCDDEEFF
IP-adress:	192.168.0.9
Nätmask:	255.255.255.0
Gateway:	192.168.0.1

Tabell C.2: Exempelvärden för SET_IP kommandot

Så ser paketet ut som i figur C.3.

ID	SET_IP	Datastorlek	MAC-adress	IP-adress	Nätmask	Gateway	STOP
AFPWM	0x02	18	0xAABBCCDDEEFF	0xC0A80009	0xFFFFFF00	0xC0A80001	0xFF

Figur C.3: Exempel på SET_IP kommandot

C.2.3 Svar på DISCOVER

När mätinstrumentet tagit emot en DISCOVER svarar den med ett paket som innehåller följande kommadon: VERSION, HW, IP_PORT och STATUS.

C.2.4 HW kommandot

Innehåller instrumentets MAC-adress. Datalängden är 6 bytes.

C.2.5 IP_PORT kommandot

6 bytes data som innehåller instrumentets IP-adress och port nummer. Om vi antar att instrumentet har IP-adress “192.168.1.4” och lyssnar på port “8738” så ser kommandot ut som i figur C.4.

Kommando	Längd	Data	Data
0x04	6	0xC0A80104	0x2222

Figur C.4: Exempel på IP_PORT kommandot

C.2.6 STATUS kommandot

En byte data som kan vara något av värdena i tabell C.3.

Namn	Kommando
FREE_FOR_CONNECTION	0x00
NET_TAKEN	0x01
NEED_IP_CONF	0x02

Tabell C.3: Giltiga datavärden för STATUS kommandot

C.2.7 VERSION kommandot

Datan som innehåller versionsnumret är ett 16-bits heltal från 0 till 65536. Datan består av två delar, vi kan kalla dom *ver* och *spec* här. Följande C-kod visar hur *ver* och *spec* beräknas från 16-bitar numret.

```
ver = 16bit_data % 10000;
spec = 16bit_data / 10000;
```

ver är alltså ett värde mellan 0 och 9999 och *spec* är ett värde mellan 0 och 6.

- Om t ex *ver* är “0124” så blir versionen “1.24”
- Om t ex *ver* är “3780” så blir versionen “37.8”

För *spec* gäller att:

- Om *spec* är “0” så händer inget speciellt
- Om *spec* är “1” och *ver* är “0124” så blir versionen “1.24a”
- Om *spec* är “2” och *ver* är “0124” så blir versionen “1.24b”
- *spec* 3,4,5 och 6 är reserverade.

D Källkod för mätinstrument

D.1 analog.h

```
/***************************************************************************
                      analog.h  -  Read analog PWM signal
                             -------------------
    begin                : 2004-03-19
    copyright            : (C) 2004 ÃF
    written by           : Andreas Agorander, Jonas Larsson
    email                : bluefire@bluefire.nu, nospam@jonaslarsson.com
 ***************************************************************************/

unsigned char freq_detect_mode; /*Either FREQ_AUTO_DETECT or FREQ_MANUAL*/
unsigned char freq_value; /*Current autodetected freq, or manually set freq */
unsigned char freq_flag;

#define FREQ_DETECTED 0
#define FREQ_NOT_DETECTED 1

/* measure
**
** Reads in 2*128 samples of input sources
** and saves thos to the arrays
** read_v_data and read_i_data
**
** Tries to autodetect frequency
**
** Waits for half the time the volt-signal is low before meausring starts
** (so that signal is centered)
*/
void measure( void );

/* Convert a number representing the amount of ticks to a number
** representing corresponding hz number
** |error| < 0.996hz
*/
unsigned char ticks_to_hz( unsigned int16 );

/* find_period_ticks
**
```

```
** Try to find the frequency of input signal
**
** Check number of RTCC timer ticks between two
** rising edges. (Calibrated for 2.5MHz RTCC timer)
** This number of ticks is saved into global var ticks_per_period
**
** Half the amount of time that signal is low is saved into global var
** waitamount. (Used for centering signal capture)
*/
void find_period_ticks( void );
```

D.2 buttons.h

```
/***************************************************************************
                  buttons.h -  Queue for button events
                         -------------------
    begin              : 2004-03-19
    copyright          : (C) 2004 ÅF
    written by         : Andreas Agorander, Jonas Larsson
    email              : bluefire@bluefire.nu, nospam@jonaslarsson.com
 ***************************************************************************/

#define KEYPRESS_QUEUE_SIZE 5
#define NUM_OF_BUTTONS 2

/* Initializing button handling functions
** Must be called before any other button functions are called
*/
void init_buttons( void );

/* check_ctrl() catches the keypresses and stores them on a queue
** must be called often to catch all events
**
** This function needs to be called regularly. It handles the checking
** of keypress status, and adds keypresses to the keyqueue
** About 50 times per second should be fine.
** Debounce code causes this function to run for 10ms when
** a keypress is detected
**
** It treats the NUM_OF_BUTTONS lowest bits of port d as it's buttons.
*/
void check_ctrl( void );
```

```
/* get the next key from the queue
**
** Returns the first button in the keypress queue (or 0 if no button
** in keyqueue) and removes it from the queue
*/
unsigned char getkey( void );
```

D.3 i2c.h

```
/***************************************************************************
            i2c.h -  Serial communication with WizNet module
                          -------------------
   begin              : 2004-03-25
   copyright          : (C) 2004 ÅF
   written by         : Andreas Agorander, Jonas Larsson
   email              : bluefire@bluefire.nu, nospam@jonaslarsson.com
 ***************************************************************************/

#define WIZNET_WRITE_ADRESS 0
#define WIZNET_READ_ADRESS 1

/* Reads one byte from wiznet module
** addr2 is MSB of adress and addr1 LSB
*/
char i2c_read_byte( char addr2, char addr1 );

/* Writes many bytes to wiznet module
** addr2 is MSB of adress and addr1 LSB
*/
void i2c_write_n_bytes( char addr2, char addr1, char *data, char len );

/* Writes one byte to Wiznet module
** addr2 is MSB of adress and addr1 LSB
*/
void i2c_write_byte( char addr2, char addr1, char data );
```

D.4 language.h

```
/***************************************************************************
          language.h -  All text-string constants for the project
                          -------------------
   begin              : 2004-02-26
   copyright          : (C) 2004 ÅF
   written by         : Andreas Agorander, Jonas Larsson
```

```
   email              : bluefire@bluefire.nu, nospam@jonaslarsson.com
 ***************************************************************************/

/* put text strings here */

/*
 *   Å = [
 *   Ä = \
 *   Ö = ]
 */

#define STR_CURRENT "STR]M"
#define STR_VOLTAGE "SP\\NNING"
#define STR_DUTY "ARB.PERIOD"
#define STR_FREQ "FREKVENS"
```

D.5 lcd_driver.h

```
/***************************************************************************
                      lcd_driver.c - Driver for KS0724
                          -------------------
   begin              : 2004-02-26
   copyright          : (C) 2004 ÅF
   written by         : Andreas Agorander, Jonas Larsson
   email              : bluefire@bluefire.nu, nospam@jonaslarsson.com
 ***************************************************************************/

// IMPORTANT:
// To use this driver please set the following constants
// to the correct PIN:s for your design .

#ifndef LCD_DRIVER_H
#define LCD_DRIVER_H

#define CS PIN_C1  //Chip Select (CS1B)
#define SID PIN_C6 //Data in pin
#define SCLK PIN_C5 //Clock pin
#define RS PIN_C0  //Register Select pin
#define RESET PIN_C2 //Hardware reset pin

/*
** Puts a 64x64 pixels picture in the middle of the framebuffer.
*/
void putlogo();
```

```
/*
** Puts 16-pixel text (one character) on framebuffer, positioned
** on 8-pixel boundary on height
** row = 0 - 6
*/
void put_text16(unsigned int16 x, unsigned int16 row, unsigned int16 num);


/* Puts eight pixel text on displaybuffer, row=0-7
*/
void put_text8( unsigned char x, unsigned int16 row, unsigned char c );


/* Puts 5*3 pixel text on displaybuffer
** unlike other textfunctions no alignment on 8-pixel boundary on height
*/
void put_text5( unsigned char x, unsigned int16 row, unsigned char c );

/* Prints a string as 8 pixel text
** 0 <= row <= 7
*/
void print8( unsigned char x, unsigned char row, char *c );

/* Prints a number as 16 pixel text
** 0 <= row <= 6
*/
void print16num( unsigned char x, unsigned char row, signed int16 num );

/* Sets or clears a single pixel at position x,y
** 0 <= x <= 127, 0 <= y <= 63 , c==0 > clear pixel c==1 > set pixel
*/
void disp_plot( unsigned char x, unsigned char y, unsigned char c );

/* dwrite, write direct to display (not framebuffer)
** page 0-8, (translated to B0-B8)
** col 0-127
** Writes one byte (eight pixels) at a time
*/
void dwrite( char page, char col, char data );

void disp_init();
```

```
/* Clear display buffer */
void disp_clear();

/* Sends contents of 1024 byte framebuffer to LCD */
void disp_update();

unsigned char fbuffer[1024]; //Framebuffer

#endif
```

D.6 network.h

```
/************************************************************************
                      network.h -  Network functions
                           -------------------
   begin              : 2004-03-25
   copyright          : (C) 2004 ÅF
   written by         : Andreas Agorander, Jonas Larsson
   email              : bluefire@bluefire.nu, nospam@jonaslarsson.com
 ************************************************************************/

#ifndef NETWORK_H
#define NETWORK_H

#define FREE_FOR_CONNECTION 0
#define NET_TAKEN           1
#define NEED_IP_CONF        2

int16 dhcplease=0;
char free_for_net = NEED_IP_CONF;
char lease_status = 0;
char ethernet_adr[6];

/*
** Sets up and runs sock init and listen on channel 3
** and port 80, this is for the webbserver part delivering
** web pages containing measuring data to browsers
*/
void init_web_sock( );

/* Interprets reponse from dhcp server
*/
char decode_dhcp_message( char *msg, char check_value, char *answ );
```

```
/*
** Constructs a skeleton DHCP package
*/
void dhcp_create_basic( char *buf );

/*
** Network initializing function for application
**
** Is meant to perform SYS_INIT on network module
** and then setup channel 1 as UDP socket and then
** negotiate IP-settings with DHCP-server
**
** Then (whether dhcp was successful or not) setup
** channel 0 as a listener for incomingbroadcasts searching
** for this instrument.
**
** This function reads hardware pin D7. If that pin is low no DHCP
** setup is attempted, high means dhcp setup attempted
**
** returns 1 (success) or 0 (no success)
*/
char start_network( char flag );

/* Function for analyzing incoming UDP packets, and possibly responding
** to them (For communication with windows program)
*/
void analyze_udp( );

/* Adds the calculated voltage, current, duty and frequency values
** to out http response
*/
int16 create_base_http_packet( char *buf );

/* Generates a graph of our data as a bmp-image, and sends
** it to the client
*/
void sendbmpdata( );

/* Receives data from port 80
** Constructs and sends response
*/
void web_recv_and_send( );
```

```
/*
** Sets up and runs sock init and listen on channel 0
** port 0x2222, this is for the application protocol
*/
void init_tcp_sock( );

#endif
```

D.7 presentdata.h

```
/***************************************************************************
                      presentdata.h - description
                          -------------------
    begin                : 2004-03-19
    copyright            : (C) 2004 ÅF
    written by           : Andreas Agorander, Jonas Larsson
    email                : bluefire@bluefire.nu, nospam@jonaslarsson.com
 ***************************************************************************/

#ifndef PRESENTDATA_H
#define PRESENTDATA_H

#define FREQ_AUTO_DETECT 0
#define FREQ_MANUAL 1

#define LOW_CUTOFF 64
#define HIGH_CUTOFF 100

#define HIGH_VOLTAGE 32
#define LOW_VOLTAGE 0

#define HIGH_CURRENT 125 //1.25 A
#define LOW_CURRENT 0

#define VMARKER_1 0   // 0 V
#define VMARKER_2 24  // 24 V

#define AMARKER_1 100 // 1.00 A
#define AMARKER_2 50  // 0.50 A
#define AMARKER_3 0   // 0.00 A

/*
** Show measured values on the display, as volt curve, current curve
** or numbers
```

```
**
** If the argument frozen is not equal to 0 then a small image of a key
** is drawn in the top right corner of the display.
*/
void present_data( char );

#endif
```

D.8 wiznet.h

```
/*************************************************************************
                      wiznet.h -  Driver for IM7010A
                          -------------------
   begin              : 2004-03-25
   copyright          : (C) 2004 ÅF
   written by         : Andreas Agorander, Jonas Larsson
   email              : bluefire@bluefire.nu, nospam@jonaslarsson.com
 *************************************************************************/

#define NUM_CHANNELS 4
#define MAX_CHANNELS 4

//Constants for commands in Cx_CR registers
#define SYS_INIT  1
#define SOCK_INIT 2
#define CONNECT   4
#define LISTEN    8
#define CLOSE    16
#define SEND     32
#define RECV     64
#define WRESET  128
#define MEMTEST 128

/* Clears a selected interrupt flag */
void int_clr_flag( unsigned char channel, unsigned char flag );

/* Init function for sys_init of wiznet module */
void wiznet_init_addr( char *smr, char *shar, char *sipr, char *gar );

/* Resets wiznet hardware, both hardware and software
*/
void wiznet_reset( );

/* Init function for sock_init (TCP or UDP)
```

```
**
** protocol : PROTO_TCP or PROTO_UDP
*/
int1 wiznet_init_sock( char channel, char protocol, char *myport );

/* Send to destip host port destport len bytes from buffer array on UDP
** channel
*/
int1 wiznet_send_udp_data( char channel, char *destip, char *destport, char *
    array, unsigned int16 len );

/* Send len bytes of data over tcp connection from array
** Channel must be initialized and connected
*/
int1 wiznet_send_tcp_data( char channel, char *array, unsigned int16 len );

/* Read up to maxlen data from tcp channel into buf
** must be connected
** actual number of bytes read returned
*/
int16 wiznet_read_tcp_data( char channel, char *buf, int16 maxlen );

/* Reads upto maxlen chars of one UDP-packet into buf,
**
** Returns number of characters read
**
** Read data is of format: 4 chars source IP, 2 chars source port
** len-6 chars data
*/
int16 wiznet_read_udp_data( char channel, char *buf, int16 maxlen );

/* Set channel in listen mode */
void wiznet_listen( char channel );

/* Close channel*/
void wiznet_close( char channel );

char socket_state[MAX_CHANNELS];
char interupt_state[MAX_CHANNELS];

/* Number of data packets available for UDP */
char socket_data[MAX_CHANNELS]={ 0,0,0,0 };
```