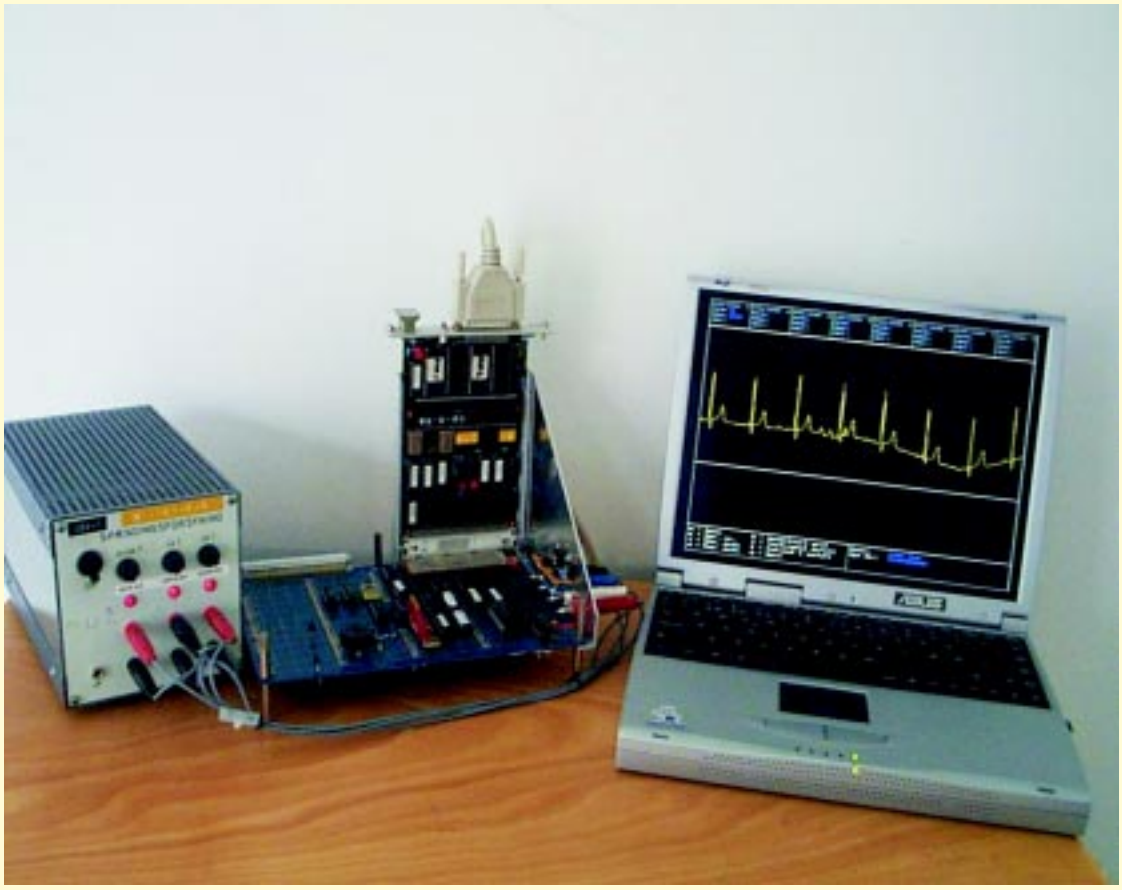
EKG-2000

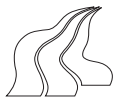
Group 412

EKG 2000



Aalborg Universitet
Institut for Elektroniske Systemer
Gruppe 412
4. semester 2000

AALBORG UNIVERSITY
INSTITUTE OF ELECTRONIC SYSTEMS, I8
DEPARTMENT OF COMMUNICATION TECHNOLOGY
Fredrik Bajers Vej 7 • DK-9220 Aalborg Ø • www.ies.auc.dk



Title:
EKG 2000

Topic:
Microprocessors

Project period:
4th semester.
2000.02.01 - 2000.05.31

Project group:
E4-412

Group members:
Lars Klitgaard Jakobsen
Christian Corfits Jensen
Kenneth Kristensen
Peter Ilsøe Nielsen
Jan Ozimek
Thomas Søhus

Supervisor:
Preben Blichert-Toft

Number printed: 9

Number of pages: 165

Abstract:
This report documents the design of an ECG monitor that is able to display up to eight persons ECG signals.
Parts of the system are not able to process signals from more than one person, due to limits in the hardware.
The system is based on a Motorola 68000 microprocessor, which uses the collected data to calculate the connected patients pulses. The calculated pulses are monitored, and if they exceed some predefined limits, an alarm is given. To warn the medical personnel, the alarm is both visual and audible.
The report starts with the construction of an amplifier, which amplifies the signal from the patient to a level that the rest of the system is able to process.
Next a unit to convert the amplified analog signals to digital signals is constructed. Moreover this unit has to give both a visual and audible alarm if the limits are exceeded.
Part of the operating system and applications to calculate the pulse, is written for the microcomputer. These programs are written in assembler and in C.
The user interface consists of a PC with an application, that displays the relevant information. The PC program is programmed to show the ECG signals from the patient, and to send pulse limits to the microprocessor. The PC program is written in C.
The microcomputer is connected to the PC via a RS232C connection.

AALBORG UNIVERSITET
INSTITUT FOR ELEKTRONISKE SYSTEMER, I8
AFDELINGEN FOR KOMMUNIKATIONSTEKNOLOGI
Fredrik Bajers Vej 7 • DK-9220 Aalborg Ø • www.ies.auc.dk

Titel:
EKG 2000.

Tema:
Mikrodatamatsystemer.

Projektperiode:
4. Semester.
01.02.2000 - 31.05.2000

Projektgruppe:
E4-412

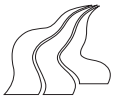
Gruppemedlemmer:
Lars Klitgaard Jakobsen
Christian Corfits Jensen
Kenneth Kristensen
Peter Ilsøe Nielsen
Jan Ozimek
Thomas Søhus

Vejleder:
Preben Blichert-Toft

Oplag: 9

Sideantal: 165

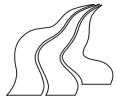
Synopsis:
Rapporten omhandler konstruktionen af et system til monitorering af op til otte personers EKG. Dele af systemet er dog, pga. hardwarens begrænsede ydeevne, kun i stand til at behandle data fra een patient.
Systemet er opbygget omkring en Motorola 68000 processor, som foretager beregninger på de opsamlede data, med henblik på at bestemme de tilsluttede patienters puls. Den beregnede puls overvåges for at undersøge, om den overskrider nogle fastlagte grænseværdier. Hvis dette er tilfældet, afgives et visuelt og auditivt signal, for at alarmere det medicinske personel.
Der konstrueres først en forstærker, som forstærker de opsamlede signaler til et niveau, som kan behandles af resten af systemet.
Dernæst opbygges en enhed, der omsætter signalerne til digitale værdier, som sendes videre til mikroprocessorkortet. Desuden har denne enhed til opgave at afgive alarmsignalet.
Til mikrocomputeren udvikles en del af operativsystemet, samt programmer til at håndtere de indkommende data for at beregne patientens puls. Disse programmer udvikles dels i programmeringssproget C og dels i assembler.
Som brugerflade til systemet benyttes en PC, som forbindes til mikroprocessorkortet vha. en RS232C-forbindelse. På PC'en udvikles software til at vise det opsamlede EKG, samt til at sende de indtastede pulstærskelværdier videre til mikrocomputeren. PC-programmet skrives i programmeringssproget C.



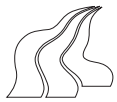
Forord

Denne rapport er udarbejdet på Aalborg Universitet, Institut for elektroniske systemer, som projekt på 4. semester i perioden 1. februar til 31. maj 2000, med temaet "Mikrodatamatsystemer".
Rapporten henvender sig til studerende på Aalborg Universitet og øvrige, som måtte have interesse for de, i rapporten, behandlede emner.
Kildehenvisninger er angivet ved forfatterens efternavn, udgivelsesår og evt. side i firkantede parenteser som vist: [efternavn, udg. år, side]. Kilderne er nærmere angivet i litteraturlisten.
Henvisninger til appendiks er angivet ved et stort bogstav, som henviser til appendiksbogstavet.
Figurer, tabeller og grafer er nummereret efter hvilket afsnit, de er placeret i, hvor de første tal angiver hvilket hovedafsnit de tilhører, og det sidste tal viser figurens individuelle nummer.
Da der i rapporten anvendes en lang række forkortelser, findes der i appendiks H en symbolliste, som forklarer disse.
Teksten og layoutet er lavet i Corel WordPerfect. Brødteksten er sat i Times New Roman. Figurer og lignende er lavet i CorelDRAW, Corel Photo-Paint og MicroSim OrCAD.
Der er bagerst i rapporten en CD-ROM, som indeholder datablade på de anvendte komponenter, de, i rapporten, dokumenterede programmer samt den samlede rapport i PDF-format. CD'en vil endvidere indeholde øvrige projektrelevante data.
I appendiks G findes en brugervejledning til det færdige apparat.

Lars Klitgaard Jakobsen	Peter Ilsøe Nielsen
Christian Corfits Jensen	Jan Ozimek
Kenneth Kristensen	Thomas Søhus



Indholdsfortegnelse

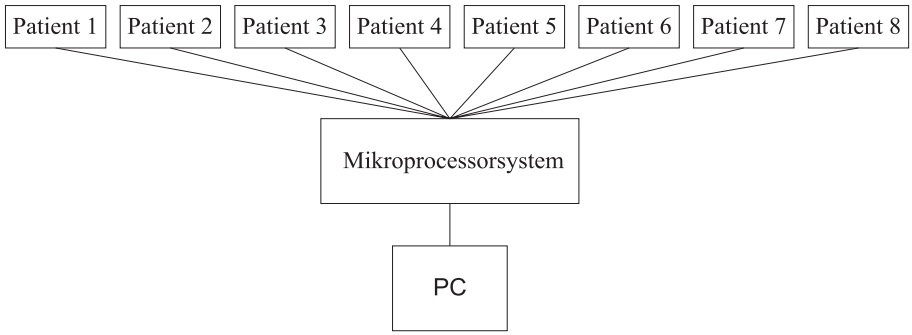


1 Indledning

Dette afsnit omfatter en formålserklæring for projektet samt en gennemgang af hjertets opbygning. Efter denne gennemgang følger to synsvinkler på hjertets funktionsprincip; en fysiologisk og en elektrofysiologisk.
Herefter opdeles EKG-signalet i forskellige stadier, som beskrives med hensyn til spændingsniveauer og varighed. Til sidst beskrives, hvordan EKG-signalet afledes fra kroppen.

1.1 Formålserklæring

Formålet med dette projekt er, at konstruere en elektrokardiogram-monitor (EKG-monitor), der kan vise en patients EKG-signal. Dette signal skal en læge kunne bruge ved diagnosticering. Apparatet skal være i stand til at overvåge otte patienter, vise deres puls og give alarm hvis alarmgrænserne overskrides. Der skal dog kun vises een patients EKG-signal på skærmen af gangen.



Figur 1.1 Oversigt over systemet.

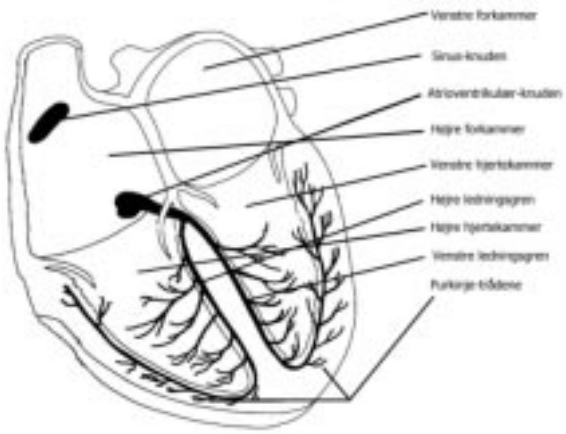
Der skal således udvikles et system, der opsamler data fra de otte patienter, behandler dem og viser resultaterne på en skærm. For at kunne opsamle data fra patienterne, er det nødvendigt at vide noget om disse signalers elektriske egenskaber.

1.2 Hjertet

Idet signalet, der måles på, er patientens hjerterytme, er hjertets funktionsprincip og signaltype specielt interessant i denne situation. I de følgende afsnit forefindes en beskrivelse af hjertets opbygning samt en beskrivelse af dets funktionsprincip. Endvidere bliver EKG-signalet opsplittet i små intervaller, som analyseres.

1.3 Hjertets opbygning

Hjertet er et meget vitalt organ, der bruges til at pumpe blod rundt i kroppen. Det er derfor et hult organ, som er designet til at udføre en pumpefunktion. [Backer, 1995]



Figur 1.2 Hjertets opbygning.

Hjertet er en muskel på størrelse med ejerens knyttede næve, og det omslutter et hulrum. Dette hulrum er opdelt i en højre og venstre halvdel, som hver især er opdelt i henholdsvis forkammer og hjertekammer. Desuden består hjertet af to vævsknuder; sinusknuden og atrioventrikulærknuden, som er en del af det autonome nervesystem.
Purkinjetrådene samt højre og venstre ledningsgren anvendes til at lede elektriske signaler.
Hjertet transporterer, for en mand på 70 kg, 5 liter blod rundt i årerne pr. minut i hviletilstand, men kan pumpe op mod 30 liter rundt i minuttet under hårdt arbejde.
Omregnet til hjerteslag (Et hjerteslag er defineret som een rytmisk kontraktion) svarer det til henholdsvis 70 hjerteslag pr. minut i hvile og omkring 220 hjerteslag per minut under hårdt arbejde. Placeringsmæssigt er hjertet, i de fleste tilfælde, at finde lidt til venstre for kroppens median og cirka midt i brystet, set i profil.

1.4 Hjertets funktionsprincip

En af blodets vigtigste funktioner er at transportere ilt rundt i kroppen. Derfor må blodet tilføres ilt. Dette sker ved at alveolesækkene, som sidder i lungerne optager ilt og blander det med blodet, hvorefter hjertet pumper det rundt.

1.4.1 Fysiologisk funktionsprincip

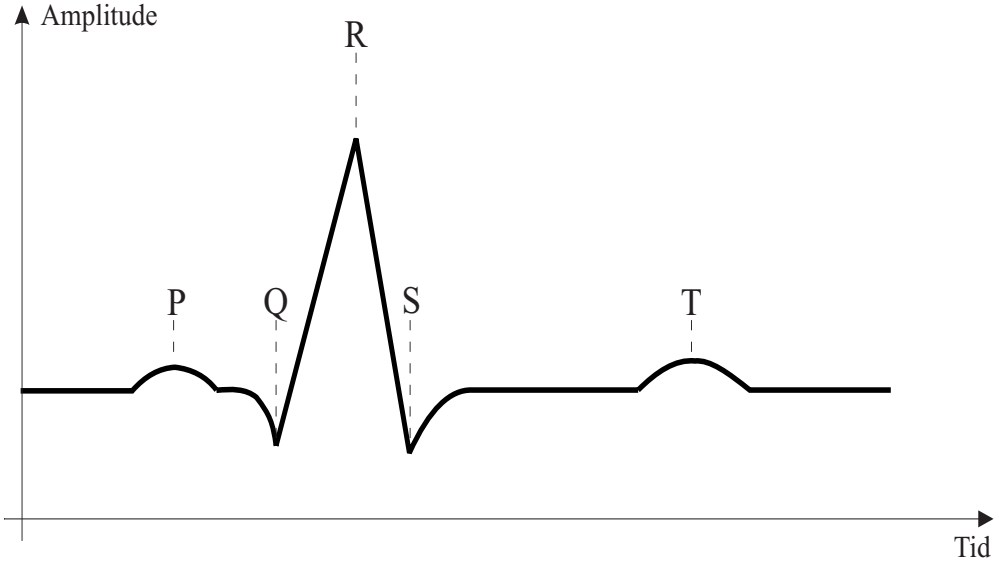
Selve pumpefunktionen fungerer ved, at det iltede blod løber ind i venstre forkammer, mens det iltfattige blod, som kommer fra kroppen, løber ind i højre forkammer. Når forkamrene er fyldte, åbnes hjerteklapperne til hjertekamrene, som umiddelbart efter fyldes, hvorefter en impuls fra det autonome nervesystem (Sinusknuden udsender en impuls til venstre hjertedel og atrioventrikulærknuden udsender en impuls til højre hjertedel) får hjertemuskulaturen til at kontrahere. Idet dette sker, lukker hjerteklapperne øjeblikkeligt, hvorved blodet presses mod klapperne til den store kropsarterie og lungearterien. Disse klapper åbnes, og blodet pumpes ud. Derefter slapper hjertet atter af, og klapperne til den store kropsarterie og lungearterien lukkes. [Backer, 1995]

1.4.2 Elektrofysiologisk funktionsprincip

Kontraktion af hjertemuskulaturen foregår ved at den skiftevis depolariseres (aflades) og repolariseres (oplades). Det er en impuls, der får hjertet til at kontrahere, og denne opstår i sinusknuden, som sidder i højre forkammer. Denne impuls får en elektrisk strøm, irritationsbølgen, til at fordele sig over hjertemusklen i begge arterier, som derved stimuleres til kontraktion. Derefter starter højre forkammer sin kontraktion, og umiddelbart efter kontraherer venstre forkammer. Impulsen påvirker også atrioventrikulærknuden og føres videre herfra via højre og venstre ledningsgren til purkinjetrådene. Purkinjetrådene leder den elektriske strøm hen til hjertekammermuskulaturen, og får derved hjertekamrene til at kontrahere samtidig. Det er irritationsbølgen, som spreder sig gennem hjertemuskulaturen, og ledsages af elektriske forandringer, som får hjertet til at slå. Det er denne elektriske spænding, EKG-monitoren registrerer. [Lunau, 1970]

1.5 Opsplitning af EKG-signalet

På figur 1.3 ses en skitse af een periode af et EKG-signal.



Figur 1.3 EKG-signal med bogstavbetegnelser på de forskellige faser.

P-takken

Denne tak repræsenterer den elektriske depolarisation gennem de to forkamre.
Eftersom muskelmassen og dermed muskelkontraktionen er forholdsvis lille, er den elektriske potentialeændring også lille. P-takken varer normalt mindre end 120 ms og har en maksimal amplitude på 0,3 mV.

PQ-intervallet

Dette interval repræsenterer den tid, det tager den elektriske puls at nå fra forkamrene til hjertekamrene. Varigheden er ca. 120 til 220 ms.

QRS-komplekset

QRS-komplekset afbilleder spredningen af elektrisk depolarisation gennem hjertekammer-muskulaturen. Q-takken er første negative udslag, R-takken er det positive udslag og S-takken er det negative udslag som følger R-takken. Hele QRS-komplekset har normalt en varighed på 120 ms og en typisk spændingsforskel fra spids til spids på ca. 1,5 mV, hvor den øverste spids ligger på ca. 1 mV.

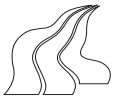
T-takken
Denne tak repræsenterer hjertekamrenes repolarisation. ST-stykket repræsenterer tidsperioden fra afslutningen af hjertekamrenes depolarisation (slutningen af QRS-komplekset) til begyndelsen af hjertekamrenes repolarisation (begyndelsen af T-takken). I denne periode løber der ingen elektrisk strøm gennem hjertemuskulaturen. T-takken har en maksimal amplitudestørrelse på 0,3 mV.

QT-intervallet
QT-intervallet er den samlede tid, der benyttes til ventriklernes depolarisation og repolarisation.

1.6 Afledning af EKG-signalet

For at kunne aflede signalet fra patienten, skal der placeres nogle elektroder på kroppen.
Disse elektroder kan placeres mange forskellige steder på kroppen, hvilket resulterer i forskellige udseender af signalet. I dette projekt anvendes en standardmetode, som er enkel at udføre. Den mest simple form for afledning er Einthovens trekant, eller blot refereret til som standard-afledningen. [Dremstrup, 2000]
Afledningen foretages, ved at placere en elektrode på hvert af patientens håndled, nær den største arterie, og en på foden, nær den største arterie. Der opdeles i tre kategorier: **I**, **II** og **III**. **I** er potentialeforskellen mellem højre og venstre hånd. **II** er forskellen mellem højre hånd og foden, og **III** er potentialeforskellen mellem venstre hånd og foden.

I projektet måles potentialeforskellen mellem venstre og højre hånd, og elektroden på foden anvendes som reference.



2 Kravsspecifikation

Her udspecificeres de krav, som på forhånd stilles til apparatet.
Ved accepttesten i kapitel 8, kontrolleres det, om systemet lever op til kravene.

2.1 Indledning

2.1.1 Formål

At lave et system som, ved hjælp af simpel databehandling, kan monitorere patienters EKG-signaler, og derved hjælpe ved diagnosticering.
Da der er tale om et indlæringsforløb, er det tilladt at ændre i kravsspecifikationen undervejs. Eventuelle ændringer vedtages på møder, hvor projektgruppen og vejlederen deltager.

2.1.2 Referencer

Ved udarbejdelsen af denne kravsspecifikation, er bogen "Håndbog i Struktureret Program-Udvikling" benyttet. [SPU, 1998]

2.1.3 Læsevejledning

Denne kravsspecifikation er baseret på den model som er beskrevet i bogen "Håndbog i Struktureret Program-Udvikling", og nummereringskonventionen fra denne er overholdt, for at gøre det let at sammenligne med modellen. (2.1.2 svarer f.eks. til 1.2 i modellen).
Det forudsættes således, at læseren er bekendt med omtalte model.

2.2 Generel beskrivelse

2.2.1 Systembeskrivelse

Det samlede system skal kunne vise en patients EKG på en computerskærm. For at udføre denne funktion, skal systemet kunne optage de elektriske impulser fra hjertet på patientens hud, og omsætte dem til en graf på skærmen. Der benyttes en PC med tilhørende skærm, men beregningerne på EKG-signalet foretages af en mikroprocessor. Dette sker for at undgå flaskehalsproblemer ved overførsel af data til PC, samt for at sikre at vitale dele af systemet stadig kører hvis PC'en er ude af drift.
Den hardware, der skal konstrueres, er alt fra måleelektroder til kommunikation med mikroprocessor. Desuden skal der udvikles software til mikroprocessorkort og PC.

2.2.2 Systemets funktioner

Følgende funktioner opdateres løbende for alle tilsluttede patienter:

- ✓ Puls: Patienternes puls skal løbende beregnes og vises på skærmen.
- ✓ Alarmer: Der kontrolleres løbende om patienternes alarmgrænser overskrides.

Endvidere opdateres følgende funktioner for den aktive patient:

- ✓ EKG: Der skal vises en graf over forløbet af patientens EKG.
- ✓ Pulstrend: Tilsvarende skal der vises en graf over forløbet af patientens puls over et givet tidsrum.

2.2.3 Systemets begrænsninger

Der skal ikke være indbygget nogen form for autodiagnosticering i apparatet.

2.2.4 Systemets fremtid

Af hensyn til fremtidssikring, skal det være muligt at opdatere softwaren til systemet, således at der er mulighed for senere at implementere auto-diagnosticering.

2.2.5 Brugerprofil

Der er tre grupper af brugere, som vil komme i kontakt med det færdige produkt:

Medicinsk personel:

Denne gruppe er den egentlige brugergruppe, da de bruger systemet til diagnosticering. Til denne gruppe skal der være en simpel brugerflade, således at systemet ikke tager opmærksomheden fra selve diagnosticeringen. Fremover vil disse være betegnet "brugere".

Patienter:

Patienterne behandles af det medicinske personel, og anvender derfor ikke systemet aktivt. Fremover vil disse være betegnet "patienter".

Teknisk personel:

Der er her tale om den gruppe der monterer og vedligeholder systemet. De skal evt. have adgang til mere avancerede funktioner, som f.eks. opdatering af software. Fremover vil disse være betegnet "teknikere".

2.2.6 Krav til udviklingsforløbet

Da der er tale om en indlæringsproces, stiller studieordningen og de øvrige studiemæssige rammer følgende krav til udviklingsprocessen:

- ✓ Der skal anvendes "struktureret programudvikling".
- ✓ Programmeringen skal dels foregå i sproget C og dels i assembler.
- ✓ Mikroprocessorkortet skal være af typen VMPM 68KA-2.
- ✓ Der skal anvendes VMEbus internt i systemet.
- ✓ Der skal anvendes RS232C internt i systemet.

Desuden skal der udvikles en brugervejledning til systemet.

2.2.7 Omfang af kundeleverance

Den 31.05.2000 skal det færdige produkt stå klar til installering, og med udførlig dokumentation. Der vil dog kun blive leveret et system, som kan monitorere 1 patient. (se pkt. 2.8. dellevering)

2.2.8 Forudsætninger

Da PC'en ikke leveres som en del af apparatet, forventes kunden at have en til rådighed. Kravene til denne er som følger:

- ✓ RS232C kommunikationsport, som er i stand til at kommunikere med 9.600 Baud.
- ✓ Operativsystemet DOS, eller kompatibelt installeret.
- ✓ 500 kB ledig harddiskplads.

2.3 De specifikke krav

2.3.1 Definitioner

Der vil igennem afsnittet blive benyttet nogle termer, som kræver en nærmere forklaring. Denne vil blive givet her, for at undgå gentagelser. Dette gælder nogle data om den enkelte patient, der defineres som følger:

- ✓ Navn: 1 til 23 ASCII-tegn.
- ✓ CPR.nr.: 10 cifre, mellem 0 og 9.
- ✓ Min. puls: Hvis patientens puls bliver lavere end denne værdi, skal alarmen aktiveres. Angives med et tal fra 0-255.
- ✓ Max. puls: Hvis patientens puls bliver højere end denne værdi, skal alarmen aktiveres. Angives med et tal fra 0-255. Max. puls skal være større end min. puls.

2.3.2 Funktionelle krav

Her angives de funktionelle krav, som skal implementeres vha. systemet. I de følgende punkter gives en detaljeret beskrivelse af funktionerne fra pkt. 2.2.2.

EKG-plot

Kravene til EKG-plottet er som følger:

- ✓ Der skal vises een persons EKG på skærmen ad gangen.
- ✓ EKG-plottet skal kunne fryses.
- ✓ Der skal altid være mindst 10 sekunders billede på skærmen; dvs. at der skal gå 10 sekunder fra, et punkt er optegnet, til det overskrives igen.
- ✓ EKG-plottet skal kunne aflæses i en afstand af 1 m.
- ✓ Det skal være af en sådan kvalitet, at det er muligt at identificere PQRT-faserne.
- ✓ Hele skærmens bredde skal udnyttes.

Pulsmåling

For at lette aflæsningen af patientens puls, skal der foruden EKG-plottet være en eksplicit pulsangivelse. Pulsangivelsen skal opfylde nedenstående krav:

- ✓ Pulsen skal udlæses i en heltalsværdi. [Bpm]
- ✓ Pulsen skal mindst opdateres een gang hvert femte sekund.
- ✓ Mulighed for indstilling af min. puls og max. puls.

✓ Alarm ved overskridelse af min. puls eller max. puls.
✓ Alle tilsluttede patienters puls skal vises på skærmen samtidigt.

Pulstrend
Pulstrend skal ligeledes vises på skærmen, hvorved det vil være muligt at følge den enkelte patients pulsudvikling over et givet tidsrum. Kravene til denne måling er som følger:

✓ Det skal være muligt at lagre de sidste 24 timers pulstrend på harddisk i PC.
✓ Hele skærmens bredde skal udnyttes.
✓ Plottet skal vises samtidigt med den aktive patients EKG.

Alarm
For at advare brugeren om patientens tilstand, hvis denne er kritisk, skal der kunne gives en alarm. Det vil være hensigtsmæssigt at alarmen indgår som en del af den konstruerede hardware, idet der herved vil kunne gives en alarm, selvom den tilkoblede PC er ude af drift. Kravene til alarmen fremgår af nedenstående:

✓ Ved overskridelse af min. eller max. puls skal der forekomme både visuel og auditiv alarm på PC og på dataopsamlingsenheden.
✓ Alarmen skal, både på PC og dataopsamlingsenhed, vise, fra hvilken patient alarmen kommer.

2.4 Eksterne grænsefladekrav

Grænsefladerne opdeles i følgende afsnit:

2.4.1 Brugergrænseflader

- på PC
Brugergrænsefladen skal præsenteres i et program, som installeres på PC. Programmet skal kunne vise alle data på een patient samtidigt på skærmen; dette værende EKG-plot, puls, pulstrend samt de tidligere omtalte personlige oplysninger. Endvidere skal brugergrænsefladen opfylde nedenstående krav:

✓ Funktionerne skal kunne aktiveres ved enkelte tastetryk.
✓ Skærmen skal være i "graphics-mode" (640 × 480).
✓ Sprog: Dansk.

- på dataopsamlingsenhed
På dataopsamlingsenheden skal der indikeres visuelt, hvilke patienter, der evt. er alarm ved. Desuden skal der her kunne gives en auditiv alarm, når en alarm aktiveres.

2.4.1 Hardwaregrænseflade
Hardwaregrænsefladerne til systemet er primært elektroderne på patientens hud. Desuden er der en grænseflade ved systemets spændingsforsyning.

✓ EKG-signal opsamles vha. 3 elektroder pr. patient.
✓ Signalniveau mellem -0,5 mV og 1 mV.

2.4.3 Kommunikationsgrænseflade

Da systemet ikke skal kommunikere med andre systemer, er dette punkt ikke specificeret.

2.4.4 Softwaregrænseflader

På PC'en ligger der en softwaregrænseflade mellem applikationen og operativsystemet. Denne er defineret af DOS.

2.5 Krav til systemets ydelse

Signalerne fra de overvågede patienter skal kunne behandles i realtime; EKG'et på skærmen må maximalt være forsinket med 1 sekund.
Der må højst gå 1 sekund fra, at en alarm registreres af systemet, til den bliver tilkendegivet af apparatet som beskrevet under punkt 2.3.2.

2.6 Kvalitetsfaktorer

I skemaet ses kvalitetsfaktorer med en tilhørende vurdering af deres prioritet. Punkterne er taget fra "Håndbog i Struktureret Programudvikling". [SPU, 1998]

Prioritering af kvaliteter	
Egenskab	Prioritering
Pålidelighed	Særdeles vigtig. Systemet skal kunne fortsætte, hvis kommunikationen med PC'en afbrydes.
Vedligeholdelsesvenlighed	Vigtig / Meget vigtig. Det skal, af hensyn til hygiejnen, være let at rengøre apparatet.
Udvidelsesvenlighed	Vigtig. Af hensyn til konkurrenceevne.
Brugervenlighed	Meget vigtig. Det er meget vigtigt at informationerne er pålidelige og at de ikke kan misforstås.
Genbrugbarhed	Vigtig. Instrumentforstærkeren skal kunne genbruges, og produceres derfor som en separat enhed.
Integritet	Vigtig / Meget vigtig. PC og mikrocomputer skal kunne fortsætte ved strømsvigt. Dette er dog ikke et krav til selve systemet, men til dets spændingsforsyning.
Effektivitet	Ikke særlig vigtig. Apparatet skal kun være så hurtigt, at det netop kan udføre funktionerne i punkt 2.3.2.

2.7 Andre krav

Apparatet skal være let transportabelt. Dvs. at det i nedpakket tilstand skal kunne bæres af een person. Vægten må derfor ikke overstige 5 kg. (Eksklusiv PC).
Da apparatet skal anvendes i et hospitalsmiljø, skal det være let at rengøre, og bør derfor monteres i en kasse, der som minimum er stænktæt. Det skal dermed være i kapslingsklasse IP34.

2.8 Dellevering

Den 31.05.2000 leveres en prototype, som er i stand til at overvåge een patient. På længere sigt skal der kunne leveres en udvidelse, der giver mulighed for tilslutning af flere patienter. Hardware på dataopsamlingsenheden skal kunne håndtere op til otte patienter. Software på PC'en designes således, at den let kan modificeres til at håndtere otte patienter.
Mikrocomputeren, samt dennes software, vil kun være i stand til at behandle data fra een patient ad gangen.
Reultatet er dermed, at det færdige system kun vil kunne overvåge een patient ad gangen.
Da den udleverede hardware skal kunne genbruges af andre projektgrupper, er der ikke mulighed for at montere systemet i en kasse, som beskrevet under punkt 2.7.

3 Systemdesign

Dette afsnit er baseret på den fremgangsmåde for programdesign, som er beskrevet i "Håndbog I Struktureret Program-Udvikling".
Trinene T1-T6 er blevet fulgt, og dette har resulteret i den følgende dokumentation.

3.1 Valg af designmetode

Først bestemmes systemets eksterne grænseflader. Dernæst inddeles systemet i en række processer, og disses eksterne grænseflader bestemmes.

3.2 Systemets eksterne grænseflader

Systemets eksterne grænseflader går ved brugerfladen til hhv. patient og det medicinske personnel.

3.2.1 Patient

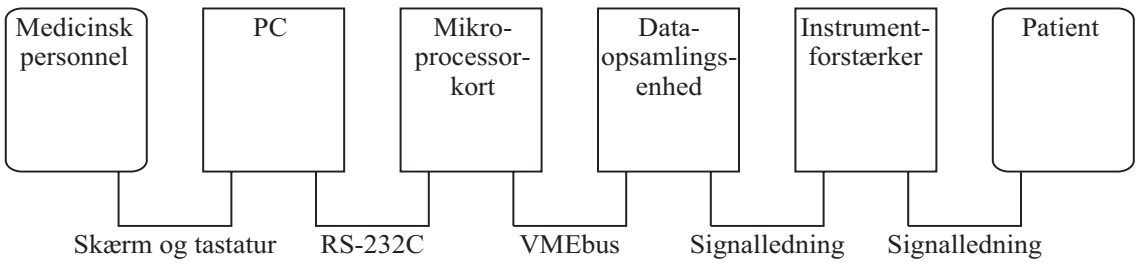
Grænsefladen til patienten lægges mellem patientens hud og elektroderne på denne. Elektroderne regnes dermed som en del af systemet. Det forventes at spændingsniveauerne på patientens hud vil ligge i intervallet fra -0,5 mV til 1 mV. Det er vigtigt at patientens hud rengøres, og evt. pudses let med fint sandpapir for at give den optimale forbindelse til elektroderne. Elektrodernes placering er beskrevet i afsnit 1.5 i indledningen.

3.2.2 Bruger

Grænsefladen til brugeren er den tilsluttede PC. Kravene til programmet på denne er opstillet i kravsspecifikationen, afsnit 2.4.1.

3.3 Opdeling i processer

Systemet opdeles i en række processer hvor imellem, der kan trækkes klare grænseflader:



3.3.1 Instrumentforstærker

Der placeres en instrumentforstærker ved hver patient. Denne har to primære funktioner:

- At forstærke signalet fra elektroderne. Som beskrevet ved fastlæggelsen af systemets eksterne grænseflader (afsnit 2.2.1), forventes signalniveauer fra -0,5 mV til 1 mV.
- At sikre galvanisk adskillelse mellem patienten og den del af systemet, som forsynes af en spændingsforsyning, der er tilsluttet lysnettet. Dette er for at sikre, at patienten ikke får stød ved en evt. fejl i apparatet.

3.3.2 Dataopsamlingsenhed

I denne del af systemet omsættes det forstærkede, analoge signal fra patienten til et digitalt signal. For at sikre en tilstrækkelig kvalitet af signalet, samples der med 500 Hz ved en opløsning på 8 bit. Der skal være otte indgange, som der skal multiplexes mellem på denne enhed.
Hvis der opstår en alarm, skal dette resultere i en visuel markering af, hvilken patient der er tale om, samt et auditivt signal.

3.3.3 Mikroprocessorkort

Signalet fra dataopsamlingsenheden behandles her med henblik på at bestemme patientens puls. EKG-signalet og pulsen sendes videre til PC'en. Endvidere kontrollerer mikroprocessorkortet, om grænserne for pulsen er overskredet, og hvis dette er tilfældet, sendes en alarm til PC og dataopsamlingsenhed.

3.3.4 PC

Den aktiveredes patients EKG, puls og pulstrend vises på skærmen. Data fra pulstrend lagres på PC i 24 timer.
Alle kommandoer til systemet gives via PC. En alarm skal ligeledes være både visuel og auditiv ved PC.

3.4 Grænseflader mellem processerne

3.4.1 Instrumentforstærker / Dataopsamlingsenhed

Der føres een linie, bestående af en fælles stelforbindelse og en signalforbindelse, mellem hver instrumentforstærker og dataopsamlingsenheden. Signalniveauet på denne linie vil være mellem 0 V og 5 V.

3.4.2 Dataopsamlingsenhed / Mikroprocessorkort

Til forbindelsen mellem dataopsamlingsenheden og mikroprocessorkortet benyttes VMEbus. Da denne er beskrevet i appendiks B, angives her kun, hvordan de enkelte ben på bussen anvendes i dette konkrete tilfælde. Angivelsen er inddelt i de forskellige cycles.

Fælles for alle cycles:

Kontrol:	CLK	Clocken fra VMEbussen overføres til dataopsamlingsenheden og benyttes til at drive de forskellige enheder på denne. Den divideres ned efter behov lokalt på dataopsamlingsenheden.
	BERR*	Ved udeblivende DTACK*, asserteres dette signal af mikroprocessorkortet.

Adresse:	DS0*-DS1*	Begge er aktiverede, hvilket angiver, at der læses hele words, dvs. 16 bit.

Interruptcycle:

Kontrol:	IRQ5*	Dataopsamlingsenheden asserterer dette ben for at blive serviceret.

IACK-cycle:

Kontrol:	IACK*	Angiver at mikroprocessorkortet er klar til at modtage en interruptvektor.
	AS*	Genereres af mikroprocessorkortet for at sikre at adressen er stabil på bussen, før de læses af dataopsamlingsenheden.
Adresse:	A_{01} - A_{03}	Mikroprocessorkortet udsender et femtal på disse tre ben, dvs. kombinatioen (1,0,1). Dette angiver at interruptniveau 5 er accepteret.

Readcycle:

Data:	D_{00}-D_{07}	Data om signalniveau fra ADC.
	D_{08}-D_{10}	Angivelse af nummeret på patienten til det pågældende sample. D_{10} er den mest betydende bit.
	D_{11}-D_{15}	Resten af statusregisteret, som er ubenyttet. Dvs. at der sendes nuller.
Kontrol:	DTACK*	Genereres på dataopsamlingsenheden for at indikere, at data er stabile på bussen, før de læses af mikroprocessorkortet.
	AS*	Genereres for at indikere, at adressen er stabil på bussen.
Adresse:	A_{01}-A_{23}	Mikroprocessoren angiver hvilken adresse den læser fra.

Writecycle:

Data:	D_{00}-D_{07}	Aktivering af måling på patienter.
	D_{08}-D_{15}	Alarm tænd eller sluk på de pågældende patienter.
Kontrol:	DTACK*	Genereres for at indikere at data er læst fra bussen.
	AS*	Genereres for at indikere at adressen er stabil på bussen.
Adresse:	A_{01}-A_{23}	Angiver adressen på dataopsamlingsenhedens kontrolregister.

Dataopsamlingsenheden skal generere en interrupt request, hver gang et sample er klart til at blive leveret til mikroprocessorkortet. Når den modtager en IACK*, sendes en interruptvektor ud på bussen, og efter et passende delay aktiveres DTACK*.
Dataopsamlingsenhedens kontrolregister tildeles adressen $F90000 og statusregisteret tildeles $FA0000. Til interrupten benyttes vektornummer 64.

3.4.3 Mikroprocessorkort / PC

Til kommunikationen mellem mikroprocessorkort og PC benyttes RS232C. (Se appendiks E)
Den benyttede protokol er defineret herunder, hvor hver kasse svarer til 1 byte sendt data.

Fra mikroprocessor til PC:

#	E	10 x	Værdi	$	\n	\r

"Værdi" repræsenterer et EKG-sample, som er et tal mellem 0 og 255.
Bemærk at EKG-signalerne sendes i pakker med 10 samples.

#	P	Værdi	$	\n	\r

"Værdi" repræsenterer pulsen, som er et tal mellem 0 og 255.

#	A	Værdi	$	\n	\r

"Værdi" angiver om alarmen skal aktiveres eller deaktiveres.
1 aktiverer. 0 deaktiverer.

Fra PC til mikroprocessor:

#	V	Værdi	$	\n	\r

"Værdi" angiver om patienten skal aktiveres eller deaktiveres.
1 aktiverer. 0 deaktiverer.

#	N	Værdi	$	\n	\r

"Værdi" repræsenterer min. puls, som er et tal mellem 0 og 255.

#	X	Værdi	$	\n	\r

"Værdi" repræsenterer max. puls, som er et tal mellem 0 og 255.

3.5 Kontrol af systemdesign

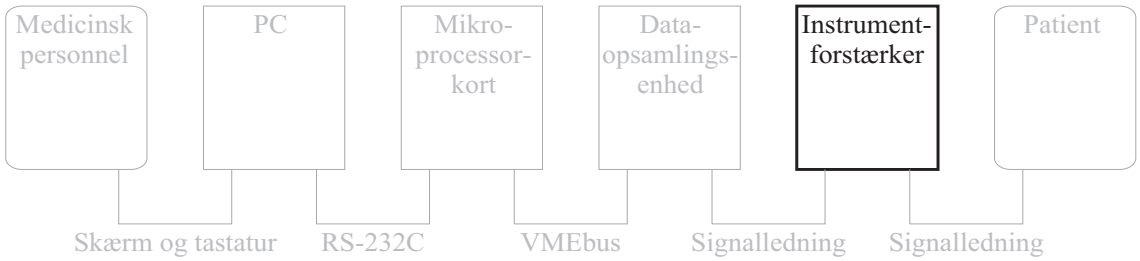
For at kontrollere om systemdesignet lever op til de, i kravsspecifikationen, stillede krav, opstilles et skema, som angiver hvilke enheder, der udfører de ønskede funktioner.

Kontrolskema	
Funktion i kravsspecifikation	Proces som varetager denne
Lagring af patientdata som CPR.nr. og navn	PC
Visning af EKG	PC
Lagring af pulstrend i 24 timer	PC
Beregning af puls	MP
Hukommelse med min. og max. puls	MP, PC
Alarm ved for høj eller lav puls	MP⇒PC, MP⇒DE
Auditiv og visuel alarmsignal	DE, PC
Indikation af den pågældende patient ved alarm	DE, PC
Generering og visning af testsignal	PC

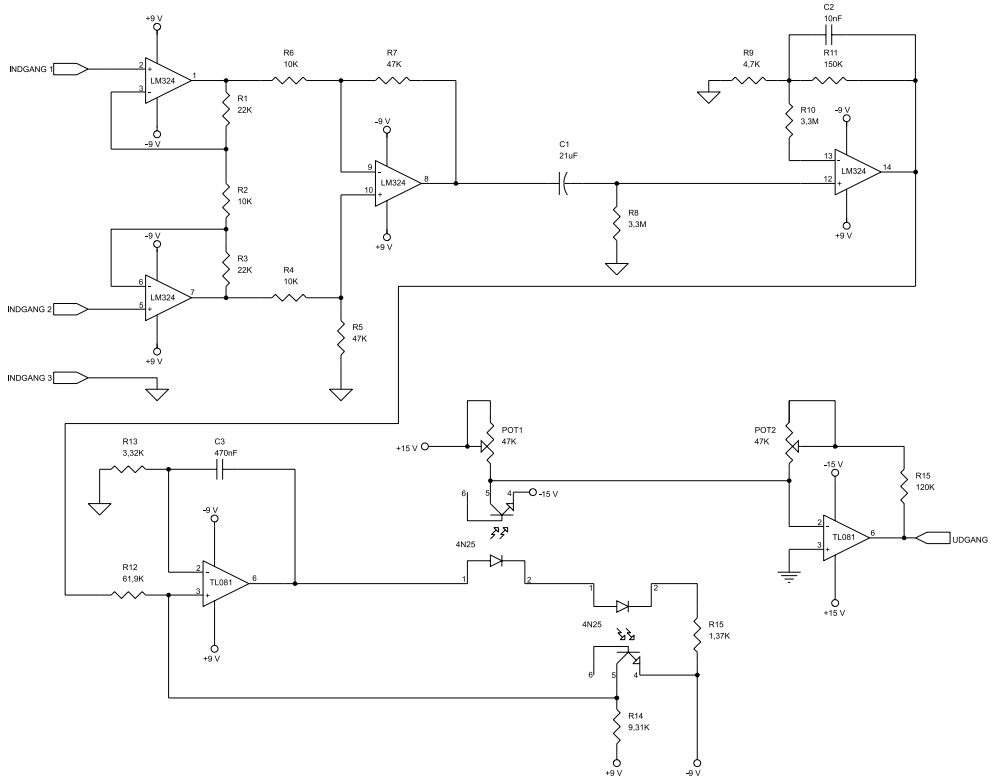
DE: Dataopsamlingsenhed
MP: Mikroprocessorkort
PC: Personlig computer
⇒: Angiver at den første enhed giver besked til den sidste.

4 Instrumentforstærker

I dette afsnit foretages en analyse af en instrumentforstærker.
Formålet med at lave en instrumentforstærker er primært at forstærke EKG-signalet i et bestemt frekvensbånd, men også at opnå galvanisk adskillelse mellem patienten og den del af systemet, der benytter sig af stærkstrøm.
Forstærkeren opdeles i fire moduler, der analyseres hver for sig.
Efter analysen af kredsløbet, er det opbygget og testet.



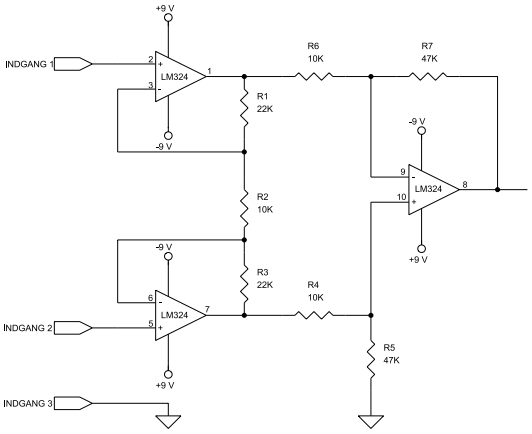
Der tages udgangspunkt i et eksisterende kredsløb, som nedbrydes i moduler, hvorefter disse moduler analyseres. Kredsløbet, som ses på figur 4.1, inddeles således i de fire moduler: differensforstærker, højpasfilter, lavpasfilter og galvanisk adskillelse.
For at lette analysen af kredsløbet, antages det, at alle operationsforstærkere er ideelle.



Figur 4.1 Samlet diagram over instrumentforstærkeren.

4.1 Differensforstærker

En differensforstærker forstærker forskellen mellem de to indgange 1 og 2. Indgang 3 bruges som reference. En almindelig differensforstærker har imidlertid den ulempe, at indgangsmodstanden er lav, og for at modvirke dette opsættes et bufferkredsløb, hvorved kredsløbet kommer til at se ud som på figur 4.2.



Figur 4.2 Differensforstærker med bufferkredsløb.

Kredsløbets virkemåde er følgende:
Der tilsluttes signaler på terminalerne INDGANG 1 og INDGANG 2, og grundet de virtuelle kortslutninger i operationsforstærkerne, vil disse signaler også forefindes på hver side af modstanden, R_2. Derved bliver spændingen over R_2:

$$v_{R2} = v_{IN1} - v_{IN2} \tag{4.1}$$

Dette medfører, at der løber en strøm:

$$I_{R2} = \frac{v_{IN1} - v_{IN2}}{R_2} \tag{4.2}$$

Denne strøm vil imidlertid også løbe gennem modstandene R_1 og R_3, hvorved der opstår en spændingsforskel mellem indgangsbenene på differensforstærkeren, som bliver:

$$v_{o1} - v_{o2} = \left(1 + \frac{R_1 + R_3}{R_2}\right)\left(v_{IN1} - v_{IN2}\right) \tag{4.3}$$

Dette medfører, at udgangsspændingen bliver:

$$v_o = -\frac{R_7}{R_6}\cdot\left(v_{o1} - v_{o2}\right) = \frac{R_7}{R_6}\cdot\left(1 + \frac{R_1 + R_3}{R_2}\right)\left(v_{IN2} - v_{IN1}\right) \tag{4.4}$$

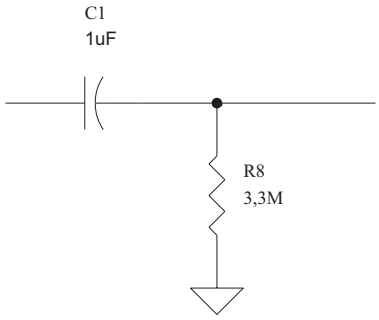
Forstærkningen kan opskrives som følger:

$$A_d = \frac{v_o}{v_{IN2} - v_{IN1}} = \frac{R_7}{R_6} \cdot \left(1 + \frac{R_1 + R_3}{R_2}\right) = \frac{47k\Omega}{10k\Omega} \cdot \left(1 + \frac{22k\Omega + 22k\Omega}{10k\Omega}\right) \approx 25{,}4\frac{V}{V} \quad \textbf{(4.5)}$$

[S&S, 1998, 89]

4.2 Højpasfilter

Idet man kun ønsker at forstærke EKG-signalerne, foretages en filtrering. For at sortere de lave frekvenser fra, er der opsat et passivt førsteordens højpasfilter som det, der ses på figur 4.3.



Figur 4.3 Førsteordens højpasfilter.

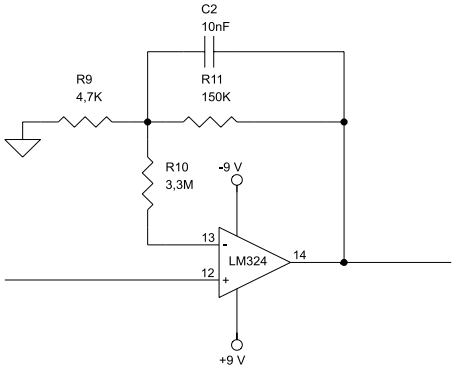
Filteret har en 3 dB knækfrekvens ved:

$$f_{HP} = \frac{1}{2 \cdot \pi \cdot R \cdot C} = \frac{1}{2 \cdot \pi \cdot 3{,}3M\Omega \cdot 1\mu F} \approx 0{,}05Hz \quad \textbf{(4.6)}$$

Idet knækfrekvensen er så lav, har højpasfilteret således kun til opgave at adskille kredsløbet DC-mæssigt.

4.3 Lavpasfilter

Lavpasfilteret er opbygget over en ikke-inverterende operationsforstærker. Lavpasfilteret kan ses på figur 4.4.



Figur 4.4 Lavpasfilter.

Modstanden på 3,3 MΩ, er placeret på dette sted, for at sikre at operationsforstærkerens indgange "ser" ind i den samme modstand. Idet der ikke løber nogen strøm gennem denne modstand, kan der ses bort fra denne. Forstærkningen bliver dermed:

$$A = \frac{Z_1 + Z_2}{Z_1} \tag{4.7}$$

Hvor Z_1 = R9 = 4,7 kΩ og

$$\frac{1}{Z_2} = \frac{1}{R_{11}} + j\omega C_2 \Rightarrow \tag{4.8}$$

$$Z_2 = \frac{R_{11}}{1 + j\omega C_2 R_{11}} \tag{4.9}$$

Ved indsættelse i udtrykket for forstærkningen bliver forstærkningen, idet vi regner på billedkredsløbet:

$$A = \frac{Z_2}{R_9} + 1 = \frac{\dfrac{R_{11} \cdot \dfrac{1}{s \cdot C_2}}{R_{11} + \dfrac{1}{s \cdot C_2}}}{R_9} + 1 = \frac{\dfrac{R_{11}}{s \cdot C_2 \cdot R_{11} + 1}}{R_9} + 1 \Rightarrow \tag{4.10}$$

$$A = \frac{\dfrac{R_{11}}{s \cdot C_2 \cdot R_{11} + 1}}{R_9} + \frac{R_9}{R_9} = \frac{\dfrac{R_{11}}{s \cdot C_2 \cdot R_{11} + 1} + R_9}{R_9} = \frac{\dfrac{R_{11} + s \cdot C_2 \cdot R_{11} \cdot R_9 + R_9}{s \cdot C_2 \cdot R_{11} + 1}}{R_9} \Rightarrow \tag{4.11}$$

$$A = \frac{\dfrac{R_{11}}{R_9} + 1 + s \cdot C_2 \cdot R_{11}}{s \cdot C_2 \cdot R_{11} + 1} = \frac{R_{11} \left(\dfrac{1}{R_9} + \dfrac{1}{R_{11}} + s \cdot C_2 \right)}{s \cdot C_2 \cdot R_{11} + 1} = \frac{R_{11} \left(\dfrac{R_9 + R_{11}}{R_9 \cdot R_{11}} + s \cdot C_2 \right)}{s \cdot C_2 \cdot R_{11} + 1} \Rightarrow \tag{4.12}$$

$$A = \frac{\dfrac{R_{11} \cdot (R_9 + R_{11})}{R_9 \cdot R_{11}} \cdot \left(1 + \dfrac{s \cdot C_2}{\dfrac{R_9 + R_{11}}{R_9 \cdot R_{11}}} \right)}{s \cdot C_2 \cdot R_{11} + 1} = \frac{R_9 + R_{11}}{R_9} \cdot \frac{1 + \dfrac{s \cdot C_2 \cdot R_9 \cdot R_{11}}{R_9 + R_{11}}}{s \cdot C_2 \cdot R_{11} + 1} \Rightarrow \tag{4.13}$$

$$A = \frac{R9 + R11}{R9} \cdot \frac{1 + (R9 \| R11) \cdot s \cdot C_2}{1 + s \cdot C_2 \cdot R_{11}} \tag{4.14}$$

Idet den første del af udtrykket bestemmer forstærkningen i pasbåndet, og den anden del knækfrekvensen, bliver forstærkningen i pasbåndet:

$$A = \frac{R_9 + R_{11}}{R_9} = \frac{150k\Omega + 4{,}7k\Omega}{4{,}7k\Omega} = 32{,}9\frac{V}{V} \quad \textbf{(4.15)}$$

Knækfrekvenserne findes ud fra polen og nulpunktet i den anden del af udtrykket.
Først findes knækfrekvensen forårsaget af polen:

$$\omega_1 = \frac{1}{C_2 \cdot R_{11}} = \frac{1}{0{,}01\mu F \cdot 150k\Omega} = 666{,}7rad/s = 106{,}1Hz \quad \textbf{(4.16)}$$

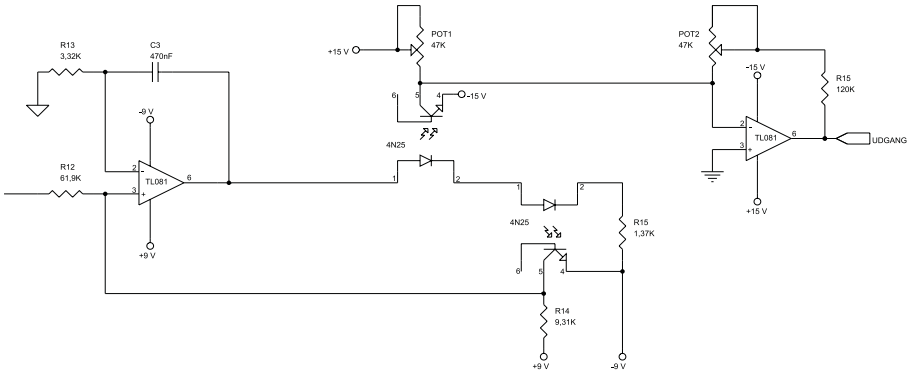
Dernæst knækfrekvensen forårsaget af nulpunktet:

$$\omega_2 = \frac{1}{C_2 \cdot \left(R_9 \| R_{11}\right)} = \frac{1}{0{,}01\mu F \cdot \left(4{,}7k\Omega \| 150k\Omega\right)} = 21943{,}3rad/s = 3{,}5kHz \quad \textbf{(4.17)}$$

Dette medfører at frekvenser mellem 106,1 Hz og 3,5 kHz dæmpes 20 dB/dekade, og at forstærkningen ved frekvenser over 3,5 kHz, går mod 1.

4.4 Galvanisk adskillelse

Den galvaniske adskillelse foretages ved hjælp af en optokobler, som er en kreds, der består af en lysdiode og en fototransistor. Ved at sende strøm gennem lysdioden, kan lyset fra denne påvirke fototransistoren. Idet der ikke er elektrisk forbindelse mellem lysdioden og fototransistoren, opnås der galvanisk adskillelse. I kredsløbet, der ses på figur 4.5, benyttes to optokoblere, hvis lysdioder er koblet i serie. Dette medfører at strømmen gennem de to fototransistorer, vil være ens.



Figur 4.5 Galvanisk adskillelse.

Idet inputstrømmen ikke kan løbe ind i operationsforstærkeren og fototransistoren ikke leder, vil der opstå en spændingsforskel mellem inputbenene på operationsforstærkeren. Dette medfører at operationsforstærkeren regulerer strømmen ud gennem dens udgang, således at der løber en strøm gennem dioderne. Denne strøm får fototransistoren til at lede, og inputstrømmen vil således løbe gennem fototransistoren. Idet strømmen gennem de to dioder er ens, vil strømmen gennem den anden fototransistor blive lige så stor som inputstrømmen. Denne strøm

kan beregnes ved:

$$i = \frac{v_i}{R_{12}} \tag{4.18}$$

Dette medfører at udgangsspændingen bliver:

$$v_o = \frac{v_i}{R_{12}} \cdot \left(POT_2 + R_{15} \right) \tag{4.19}$$

Ud fra ovenstående formel og forskellige indstillinger af POT_2, kan den minimale og den maksimale forstærkning beregnes:

$$A_{\min} = \frac{v_o}{v_i} = \frac{POT_2 + R_{15}}{R_{12}} = \frac{0k\Omega + 120k\Omega}{61{,}9k\Omega} = 1{,}94\frac{V}{V} \tag{4.20}$$

$$A_{\max} = \frac{v_o}{v_i} = \frac{POT_2 + R_{15}}{R_{12}} = \frac{47k\Omega + 120k\Omega}{61{,}9k\Omega} = 2{,}70\frac{V}{V} \tag{4.21}$$

Potentiometeret POT_2 indstilles til en forstærkning på 2,45 V/V, hvilket opnås ved en indstilling på 31,9 kΩ.

4.5 Samlet kredsløb

Efter sammenkobling af alle modulerne kan den samlede forstærkning findes til:

$$A = 25{,}4\frac{V}{V} \cdot 32{,}4\frac{V}{V} \cdot 2{,}45\frac{V}{V} = 2016\frac{V}{V} \tag{4.22}$$

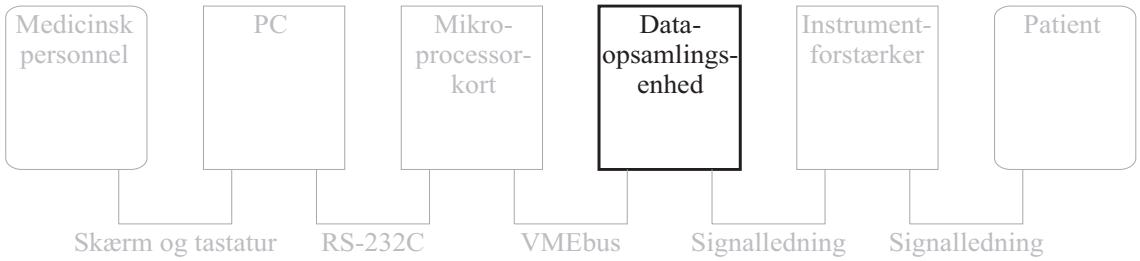
Idet spændingen fra patienten ligger mellem -0,5 mV og 1 mV, ligger udgangssignalerne fra instrumentforstærkeren, mellem -1,008 V og 2,016 V. Da ADC'en ikke kan arbejde med bipolære signaler, foretages en signaltilpasning med en kondensator og to modstande, således at signalet kommer til at svinge omkring 2,5 V.

4.6 Modultest

Efter analysen af kredsløbet, er det opbygget i laboratoriet. Ved afprøvning på en testperson, fandt vi ud af, at der var problemer med støj på signalet. Problemet var at støjen i laboratoriet, havde en amplitude, der var tilnærmelsesvis lige så stor, som det signal vi skulle måle. Problemet blev til dels løst ved at placere kredsløbet i en metalkasse, og forbinde denne til testpersonens referenceledning. Efter denne løsning på problemet blev kredsløbet testet ved at tilkoble en sinusgenerator på indgangen. Dette resulterede i et udgangssignal, som var af en acceptabel kvalitet.

5 Dataopsamlingsenhed

I dette kapitel er proceduren for procesdesign i "Håndbog I Struktureret Program-Udvikling" anvendt. Dermed er de anviste trin [SPU, 1998, 136] brugt som retningslinie.
Modulerne er beskrevet i den rækkefølge, de bliver nødvendige i den samlede dataopsamlingsenhed.
En grundigere beskrivelse følger senere i kapitlet, under udarbejdelsen af de enkelte moduler.



Denne enhed har til opgave at omforme signalerne fra forstærkerne således at mikroprocessoren kan behandle dem. Desuden skal den kunne give de, i kravsspecifikationen, nænvte alarmsignaler.
Designmetoden "mest kritisk først" under "Funktionsorienteret design" er anvendt. Ifølge denne model skal man beskrive de primære dele af dataopsamlingsenheden først og derefter beskrive de sekundære elementer, som er nødvendige, for at få de primære dele til at fungere.

VMEbus interface
De enkelte moduler på dataopsamlingsenheden kan ikke forbindes direkte til VMEbussen. Derfor indsættes en række drivere, som skal sikre, at der f.eks. ikke trækkes for meget strøm fra VMEbussen. Desuden muliggør de, at der kan kobles flere enheder på bussen, uden at der opstår konflikter mellem dem.

ADC/MUX
Formålet med dataopsamlingsenheden er, at konvertere 8 parallelle, analoge signaler til 8 serielle, digitale signaler. Kravene til konverteringen er, at opløsningen skal være 8 bit og samplingsfrekvensen skal være på 500 Hz pr. kanal. [Dremstrup, 2000]. ADC-modulet skal afgive et patientnummer, sammen med EKG-signalet, således at det fremgår, hvilken patient signalet tilhører.

Clock divider
Til ADC/MUX-modulet skal der benyttes en clockpuls til at styre, hvilken patients signal, der skal konverteres. Der skal bruges en clockpuls til at starte konverteringen, og en intern clock puls til at styre selve ADC'en. For at generere disse pulser, deles systemclocken på VMEbussen fra 16 MHz ned til hhv. 1 MHz og 4 kHz.

IRQ-generator
Når konverteringen fra analog til digital er færdig, skal data afhentes fra ADC'en, hvilket foregår ved hjælp af interrupt. Da det ikke nødvendigvis er alle patienter, der skal opsamles data fra, skal IRQ-generatoren sørge for, at der kun sendes et IRQ-signal ved de patienter, som er registreret i kontrolregisteret. Desuden skal det sikres, at IRQ-signalet har den rette længde.

IRQ-dekoder
Når mikroprocessoren har accepteret interruptet, udsender denne IRQ-nummeret på de tre laveste adresseben. IRQ-dekoderen skal kontrollere dette signal sammen med en række kontrolsignaler (AS* og IACK*), og hvis disse er aktive, afsende IRQ-vektoren.

Adressedekoder
Dette modul dekoder de, af microprocessorkortet, udsendte adresser, og aktiverer de moduler, som skal udføre den ønskede opgave. De, i systemdesignet, valgte adresser for hhv. in- og output anvendes.

Statusregister
I registeret skal der stå, hvilken patients EKG-signal mikroprocessoren modtager. Registeret skal holde på denne information fra den kommer fra ADC/MUX-modulet og indtil mikroprocessoren har læst det.

Kontrolregister
I dette register står informationerne om, hvilke patienter der er tilsluttet og om eventuelle alarmer. Disse data kommer fra mikroprocessorkortet under en write-cycle. Registeret skal holde på disse data, således at de altid er tilgængelige for andre moduler på dataopsamlingsenheden.

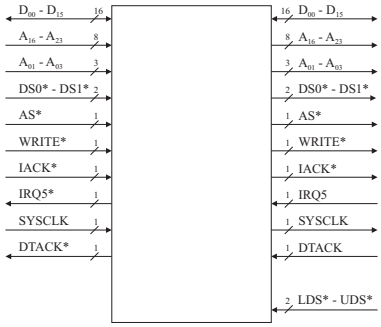
Alarm
Systemet er designet således, at mikroprocessorkortet skal kunne udsende et alarmsignal, hvis EKG-signalets form tyder på en kritisk situation. Derfor skal der, på dataopsamlingsenheden, laves en auditiv og visuel alarmanordning. Der skal være en visuel indikation af hvilke patienter der er alarm ved. Hvis der er alarm ved mindst een patient, skal der samtidig være en auditiv indikation.

DTACK-generator
Mikroprocessorkortet skal have et signal, som styrer timingen af data til og fra dataopsamlingsenheden. Dette signal skal afgives når data er klar på databussen. For at sikre stabile data, skal delayet på timingen beregnes ved "worst case".

5.1 Grænseflader mellem modulerne

I det følgende er grænsefladerne til de enkelte moduler beskrevet modul for modul. Der er lavet en figur til hvert modul, hvor alle in- og output er vist. Input er placeret i venstre side, og output er placeret i højre side. VMEbusinterfacet er dog en undtagelse, da det, som det eneste modul, har tovejskommunikation gennem de samme forbindelser. Her er siden mod VMEbussen placeret til venstre, og siden mod resten af dataopsamlingsenheden til højre.
I appendiks I kan den samlede dataopsamlingsenheds moduler ses.

VMEbusinterface



D_{00} - D_{15}	Databussen.
A_{16} - A_{23}	De øverste 8 ben fra adressebussen.
A_{01} - A_{03}	Interruptnummeret fra mikroprocessoren.
DS0* - DS1*	Aktivering af henholdsvis de 8 nederste, de 8 øverste eller alle 16 databit.
AS*	Bliver sat, når data er stabile på adressebussen.
WRITE*	Angiver om mikroprocessoren læser eller skriver på databussen.
IRQ5*	Interruptsignal fra dataopsamlingsenheden, som inverteres af interfacet.
IACK*	Signal som mikroprocessoren afgiver når den har accepteret et interrupt.
SYSCLK	Systemclock fra VMEbussen (16 MHz).
DTACK*	Signal der afgives af dataopsamlingsenheden, når dens data er klar. Inverteres af interfacet.
LDS* - UDS*	Benyttes til intern selektering af databussen. Signalerne LDS* og UDS* har samme funktion som henholdsvis DS0* og DS1*.

ADC/MUX

A_SIG0 - 7	Otte analoge signaler fra instrumentforstærkeren.
CONV_CLK	Konverteringsclock på ADC'en.
ADC_CLK	Startsignal til ADC.
COUNT_CLK	Startsignal til tæller.
ADC_ENA	Aktiverer data på udgang af ADC'en.
D_{00} - D_{07}	8 bit EKG-signal.
EOC	Konvertering færdig (End of Conversion).
PID1-PID3	Nummeret på den valgte patient.

Clock divider

SYSCLK	Systemclock fra VMEbussen (16 MHz).
CONV_CLK	Konverteringsclock til ADC'en.
COUNT_CLK	Tællersignal til styring af multiplexer.
ADC_CLK	Startsignal til ADC'en.

IRQ-generator

EOC	Konvertering færdig (End of Conversion).
PS0 - PS7	Repræsenterer de tilsluttede patienter.
PID1 - PID3	Angiver fra hvilken patient, signalet stammer.
IACK*	Signal fra mikrocomputeren om at stoppe IRQ5, da den ér accepteret.
IRQ5	Et interruptsignal fra dataopsamlingsenheden.

IRQ-dekoder

A_{01} - A_{03}	Interruptnummeret fra mikroprocessor.
AS*	Bliver sat, når data er stabile på adressebussen.
IACK*	Indikerer at mikrocomputeren har accepteret interruptet.
A_LDS*	Lower datastrobe fra adressedekoder.
D_{00} - D_{07}	IRQ-vektoren skal repræsenteres på disse databen.
I_DTACK*	Signal til chipselect af DTACK-generatoren.
LDS*	Aktiverer interfacets 8 nederste databen.

Adressedekoder

DS0* - DS1*	Aktivering af henholdsvis de 8 nederste, de 8 øverste eller alle 16 databit.
AS*	Bliver sat, når data er stabile på adressebussen.
A_{16} - A_{23}	De øverste 8 ben fra adressebussen.
ADC_ENA	Aktiverer data på udgang af ADC'en
STA_ENA*	Enable til statusregistret.
CON_ENA*	Enable til kontrolregistret.
A_DTACK*	Et chipselect til DTACK-generatoren.
A_LDS*- UDS*	Benyttes til intern selektering af databussen. Har samme funktion som henholdsvis DS0* og DS1*.

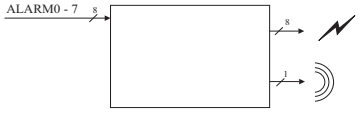
Statusregister

PID1 - PID3	Patient ID fra ADC/MUX-modulet.
STA_ENA*	Enable til statusregistret.
PID1 - PID3	Det samme signal, som kom ind i modulet, sendes videre til D_{08} - D_{10} på VMEbussen.

Kontrolregister

D_{00}-D_{07}	Angiver tilsluttede patienter.
D_{08}-D_{15}	Angiver eventuelle alarmer.
CON_ENA*	Enable til kontrolregistret.
ALARM0 - 7	Angiver patienter med alarm.
PS0 - PS7	Disse signaler repræsenterer de tilsluttede patienter.

Alarm



ALARM0 - 7	Angiver patienter med alarm.
LED0 - 7	Hver patient er repræsenteret ved en lysdiode.
SPEAKER	Bruges til at aktivere en højttaler.

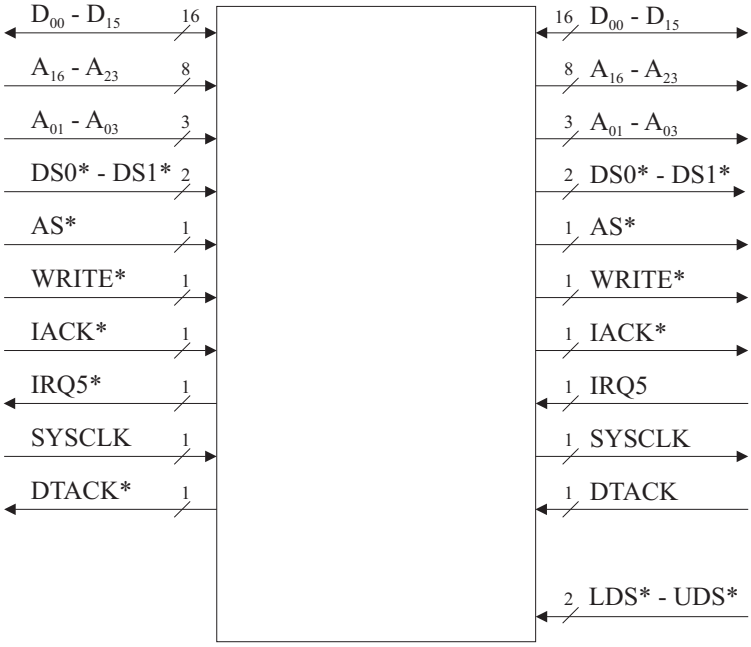
DTACK generator

A_DTACK*	Dette signal kommer fra adressedekoderen når en DTACK skal afsendes.
I_DTACK*	Dette signal kommer fra IRQ-dekoderen når en DTACK skal afsendes.
SYSCLK	Systemclocken fra VMEbussen på 16 MHz.

DTACK — Det forsinkede signal, som fortæller mikroprocessorkortet, at data er blevet sendt eller modtaget. Signalet inverteres af VMEbusinterfacet, således at mikroprocessoren modtager DTACK*.

5.2 Interface til VMEbussen

Interfacet er placeret mellem resten af dataopsamlingsenheden og VMEbussen. Det sikrer kommunikationen mellem den interne databus på dataopsamlingsenheden og VMEbussen.



Figur 5.1 Interfacets grænseflader mellem dataopsamlingsenhedens interne bus og VMEbussen. Venstre side af figuren udgør grænsefladen til VMEbussen fra mikroprocessoren og højre side grænsefladen til den interne bus på dataopsamlingsenheden.

Input

Grænsefladen til VMEbussen betragtes her som input, selvom kommunikationen i nogle tilfælde går i begge retninger.

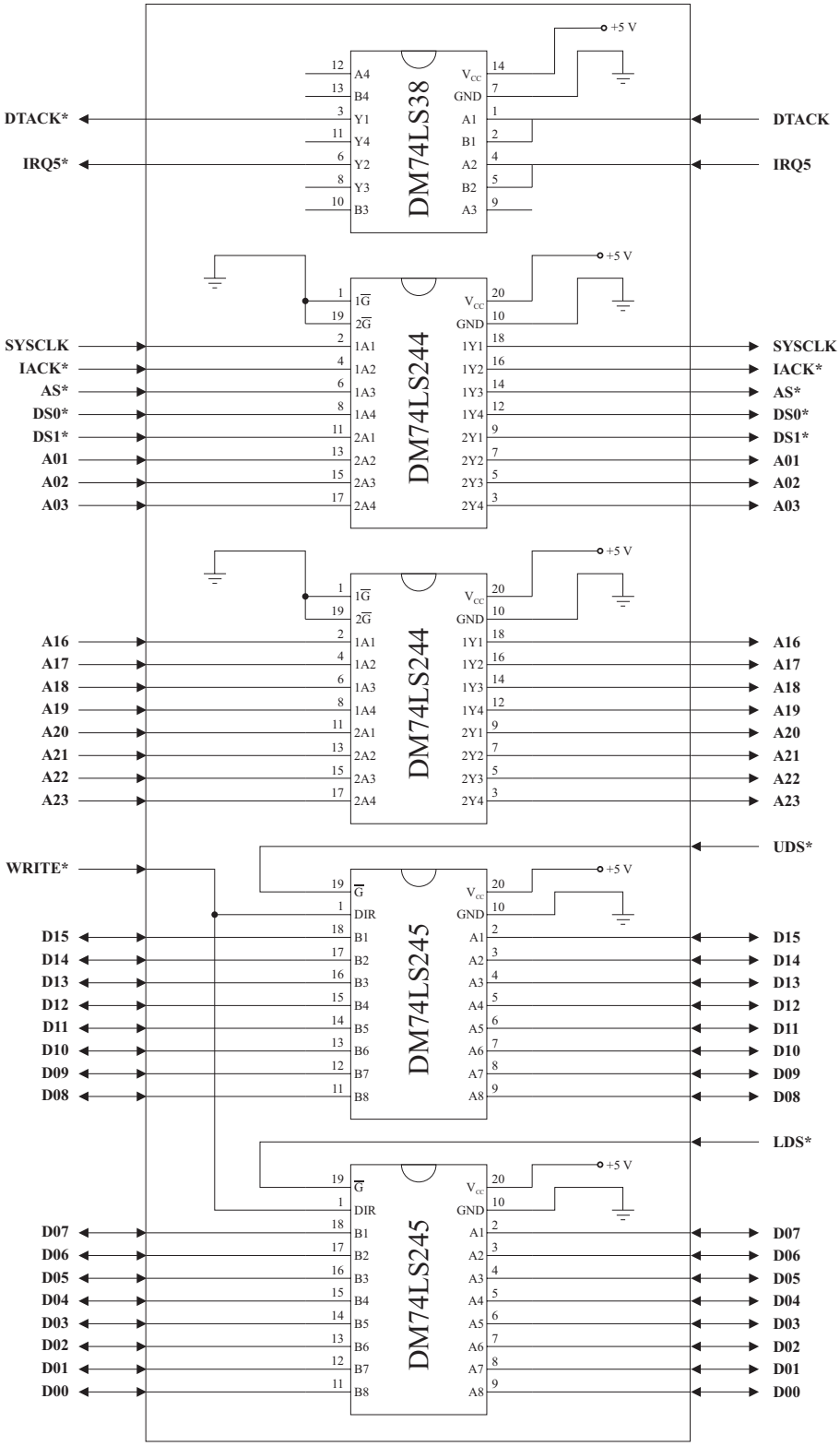
D_{00} - D_{15}	Databussen.
A_{16} - A_{23}	De øverste 8 ben fra adressebussen.
A_{01} - A_{03}	Interruptnummeret fra mikroprocessoren.
DS0* - DS1*	Aktivering af henholdsvis de 8 nederste og de 8 øverste databit.
AS*	Aktiveres når adressen er stabil på bussen.
WRITE*	Angiver om mikroprocessoren læser eller skriver på databussen.
IRQ5*	Interruptsignal fra dataopsamlingsenheden, når den ønsker at blive serviceret. Inverteres i interfacet.
IACK*	Et signal som mikroprocessoren afgiver, når den har accepteret et interrupt.
SYSCLK	Systemclocken fra VMEbussen (16 MHz).
DTACK*	Afgives af dataopsamlingsenheden når den tilsigtede operation er udført. Inverteres i interfacet.

Output
Grænsefladen til dataopsamlingsenhedens interne bus betragtes her som output, selvom kommunikationen i nogle tilfælde går i begge retninger. På denne side optræder de samme signaler som på VMEbussiden. Derfor vil der kun blive nævnt de nye signaler som optræder her.

LDS* - UDS*	Aktivering af henholdsvis de 8 nederste og de 8 øverste databit til VMEbussen.
DTACK	Afgives af dataopsamlingsenheden når den tilsigtede operation er udført. Inverteres i interfacet.
IRQ5	Interruptsignal fra dataopsamlingsenheden, når den ønsker at blive serviceret. Inverteres i VMEbusinterfacet.

5.2.1 Design
Som det ses på figuren, er interfacets primære funktion at virke som en buffer mellem den interne og eksterne bus. Disse buffere skal leve op til nogle veldefinerede elektriske krav mht. spændingsniveauer, strømniveauer og lignende.
Endvidere introduceres der to ekstra signaler på interfacets grænseflade til den interne bus. Disse to signaler, LDS* og UDS* benyttes som chipselect til databussens buffere. Disse chipselect er nødvendige, idet dataopsamlingsenheden, i nogle tilfælde, skal skrive til databussen, og i andre tilfælde læse fra databussen. I modsætning hertil skal de buffere, som modtager adresser og kontrolsignaler ikke bruge et chipselect, da de altid skal læse på adressebussen for at modtage dataopsamlingsenhedens interne adresser til status- og kontrolregistret samt diverse kontrolsignaler.

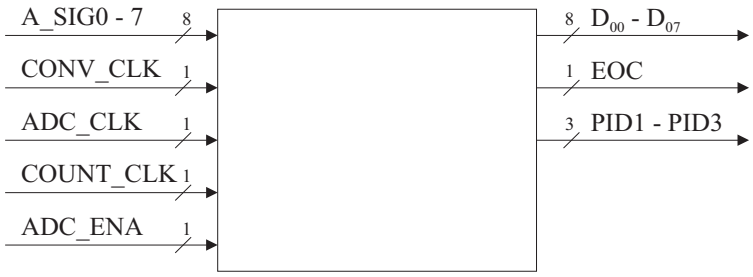
5.2.2 Implementering
Der benyttes buffere fra 74LS serien. De lever godt nok ikke op til VMEbusstandardens elektriske krav, men erfaringen viser at der ikke opstår problemer. [Clements, 1997, 807]
Til håndtering af databussen, D_{00} - D_{15} vælges to 74LS245 8 bit transceivere, idet kommunikationen på databussen skal gå i begge retninger. WRITE*-signalet benyttes til at styre retningen på de to transceivere; altså hvorvidt der skal hentes data fra VMEbussen, eller der skal placeres data på VMEbussen. Til adressebussen A_{16} - A_{23} , A_{01} - A_{03}, AS*, DS0* - DS1* samt signalerne IACK* og SYSCLK vælges to 74LS244 8 bit receivere, idet disse signaler kun skal modtages fra VMEbussen.
Signalerne DTACK og IRQ5, som skal styres fra dataopsamlingsenheden, interfaces til VMEbussen gennem en 74LS38, som er en NAND-gate med "open collector"-udgang. Denne kreds vælges, da der derved kan tilsluttes flere enheder til disse ben på VMEbussen, uden at der opstår problemer. Da kredsen er opdelt i fire NAND-gates, benyttes den ene til at drive IRQ5-signalet, den anden til DTACK-signalet og de to sidste bliver ikke brugt. Signalerne IRQ5 og DTACK forbindes til begge inputben på hver sin NAND-gate. Derved fungerer denne som en simpel inverter, hvorved der fås signalerne IRQ5* og DTACK* på VMEbussen.
På baggrund af dette er der opstillet et diagram over dette modul, som kan ses på figur 5.2.



Figur 5.2 Det samlede diagram over interfacemodulet, hvor der er benyttet kredse fra 74LS-serien.

5.3 ADC/MUX

Modulet skal konvertere otte analoge signaler til eet digitalt signal pr. indgang.



Figur 5.3 Input- Outputrelationen for modulet.

Input

A_SIG0 - 7	Otte analoge signaler fra instrumentforstærkerne.
CONV_CLK	Konverteringsclocken på ADC'en.
ADC_CLK	Startsignal til ADC.
COUNT_CLK	Startsignal til tæller.
ADC_ENA	Aktiverer data på udgang.

Output

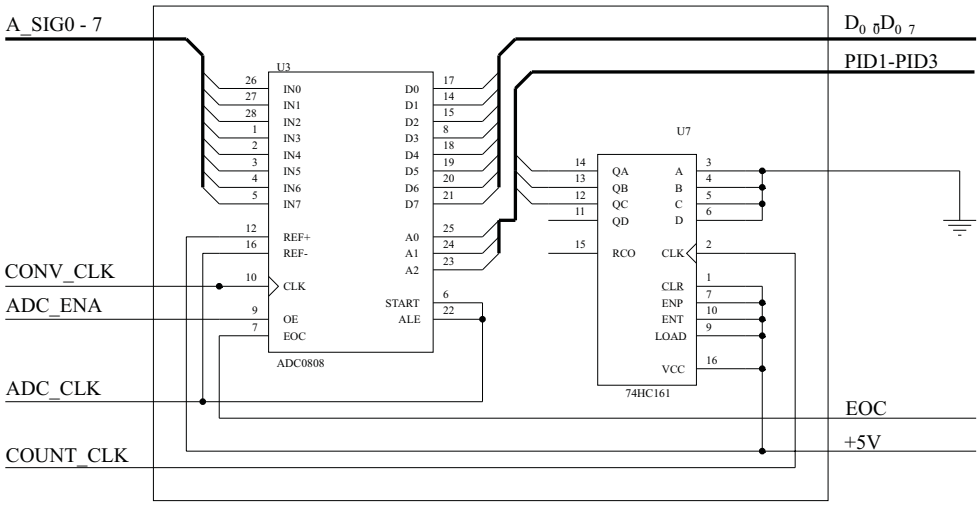
D_{00} - D_{07}	8 bit digitalt signal.
EOC	Konvertering færdig.
PID1 - PID3	Nummeret på den valgte patient.

5.3.1 Design

Modulet består af tre dele: En tæller, en multiplexer og en ADC.
Tælleren skal holde styr på, hvilken patients signal der bliver konverteret, ved at sende en 3 bit kode (PID1 - PID3) videre til multiplexeren og IRQ-generatoren.
Multiplexeren skal skifte mellem de otte analoge indgange og sende den valgte videre til ADC'en.
ADC'en skal konvertere det valgte analoge signal til et 8 bit digitalt signal (D_{00} - D_{07}), hvorefter den sender et EOC (End Of Conversion) til IRQ-generatoren.

5.3.2 Implementering

Der er valgt en 74HC161 kreds til tælleren. Til multiplexer og ADC er valgt en kombineret kreds: ADC0808CCN.

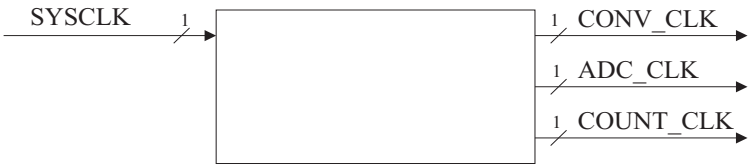


Figur 5.4 Diagram over modulet.

Konverteringscycle
Det første der sker i en konverteringscycle er, at tælleren bliver trigget af clockdivideren (COUNT_CLK). Den sætter så de tre PID-bit på udgangen, så multiplexeren ved, hvilken analog indgang den skal sende videre. 50 ns efter bliver ADC'en selekteret fra clockdivideren ved ADC_CLK. Grunden til denne forsinkelse er tællerkredsens delay, som typisk er 19 ns (max 30 ns). Konverteringen tager typisk 100 µs, hvorefter EOC asserteres. IRQ-rutinen går i gang, og når mikroprocessorkortet er klar, asserteres ADC_ENA via adressedekoderen. Derefter bliver data placeret på bussen (D_{00} - D_{07}).

5.4 Clockdivider

For at generere clockpulserne til ADC'en, deles systemclocken (SYSCLK), fra VMEbussen, til de ønskede frekvenser.



Figur 5.5 Input- Outputrelationen for modulet.

Input

SYSCLK	Systemclock fra VMEbussen på 16 MHz

Output

CONV_CLK	Konverteringsclock til ADC'en på 1 MHz.
COUNT_CLK	Tællersignal til styring af multiplexer på 4 kHz.
ADC_CLK	Startsignal til ADC'en på 4 kHz, forsinket 50 ns i forhold til COUNT_CLK.

5.4.1 Design

Da clockdelingerne skal implementeres med logiske kredse, vil det være hensigtsmæssigt, at udtrykke dem med 2^n.

For at lave konverteringsclocken (CONV_CLK) skal systemclocken divideres med:

$$k = \frac{f_{SYSCLK}}{f_{CONV\_CLK}} = \frac{16\,MHz}{1\,MHz} = 16 \qquad \textbf{(5.1)}$$

det vil sige at SYSCLK skal deles 4 gange da $2^4 = 16$.

Til COUNT_CLK og ADC_CLK skal systemclocken divideres med:

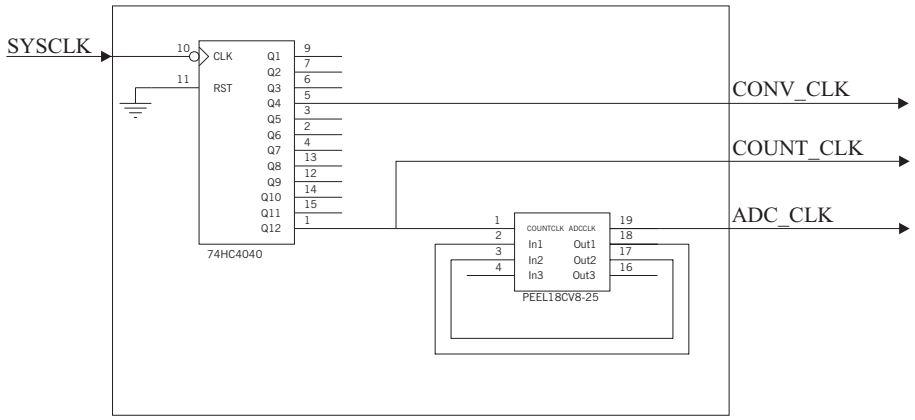
$$k = \frac{f_{SYSCLK}}{f_{COUNT\_CLK}} = \frac{16\,MHz}{4\,kHz} = 4000 \qquad \textbf{(5.2)}$$

$$n = \frac{\log k}{\log 2} = \frac{\log 4000}{\log 2} = 11{,}96 \qquad \textbf{(5.3)}$$

Da $4000 = 2^{11,96}$ er uhensigtsmæssigt at regne med, divideres COUNT_CLK og ADC_CLK i stedet med 2^{12}, hvilket giver en clock på 3906 Hz. Denne ændring vil ikke få indflydelse på funktionen af dataopsamlingsenheden, men der skal tages hensyn til det i softwaren.

5.4.2 Implementering

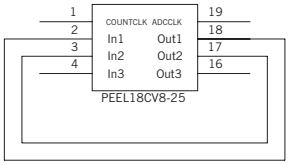
Til implementering af clockdivideren benyttes en tællerkreds, af typen 74HC4040A.
Til at udføre tidsforsinkelsen programmeres en PEEL-kreds, af typen 18CV8-25. Idet denne kreds har et propagationdelay på 25 ns, skal signalet blot "loopes" gennem kredsen to gange.
ABEL-programmet til PEEL-kredsen kan ses i appendiks A.



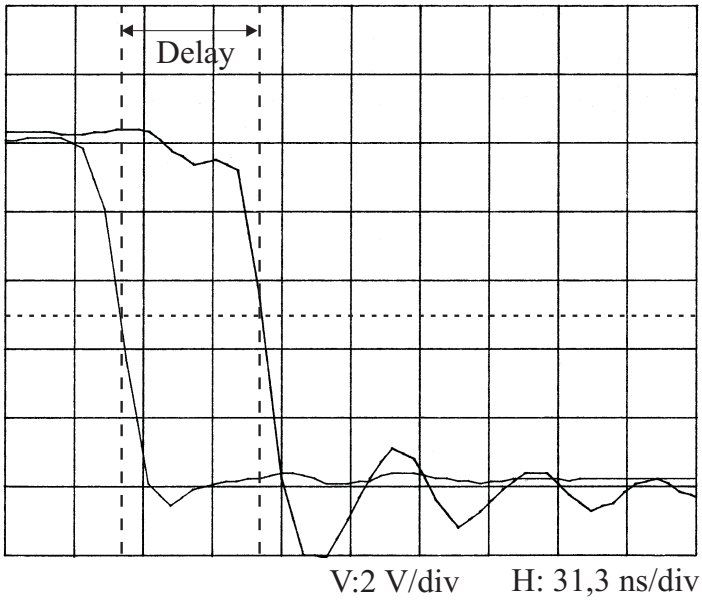
Figur 5.6 Diagram over modulet.

5.4.3 Modultest

For at undersøge om det ønskede delay på 50 ns er overholdt, testes PEEL-kredsen i en måleopstilling. På figur 5.7 er opstillingen til PEEL-kredsen vist. Ved at sætte en puls ind på COUNT_CLK, kan det måles, hvilket delay der er på ADC_CLK.



Figur 5.7 PEEL-kreds til at generere delay.

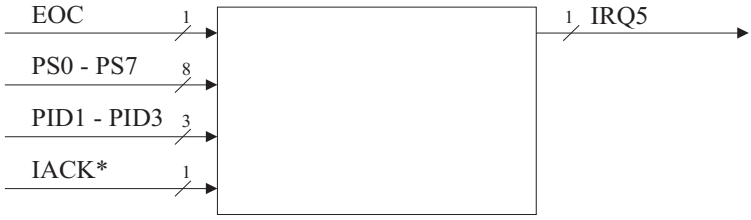


Figur 5.8 Oscilloskopplot der viser det delay PEEL-kredsen genererer.

Det ses på figuren, at det ønskede delay på 50 ns, er opnået, og dermed anses modulet for at virke.

5.5 IRQ-generator

IRQ-generatoren har til formål at afgive et IRQ-signal til mikrocomputeren, når ADC'en er færdig med en konvertering, og skal serviceres. Signalet til mikrocomputeren kunne blot have været ADC'ens EOC-signal (End Of Convertion), men for at undgå at mikrocomputeren forstyrres mere end højst nødvendigt, indsættes en IRQ-generator, der kontrollerer, om EOC-signalet må sendes videre. IRQ-signalet skal kun sendes til mikroprocessoren ved de patienter, som skal overvåges. Kontrolregisteret indeholder information om hvilke patienter, der er tilkoblet, og ved at sammenligne disse informationer med patient ID, kan det bestemmes, om der skal afsendes en IRQ5. For at sikre at IRQ-signalet ikke bliver stående for længe, og derved bliver opfattet som et nyt IRQ-signal, skal det fjernes igen når der kommer en IACK* fra mikroprocessoren.



Figur 5.9 Input- Outputrelationen for modulet.

Input

EOC	Kommer fra ADC'en, når den er færdig med at konvertere.
PS0 - PS7	Repræsenterer de tilsluttede patienter.
PID1 - PID3	Angiver hvilken patient signalet stammer fra.
IACK*	Når dette signal kommer fra mikrocomputeren, skal IRQ5 deaktiveres.

Output

IRQ5	Skal afgives når EOC bliver høj, og patient ID og de tilsluttede patienter stemmer overens.

5.5.1 Design

Designet opdeles i to, hvor den første del kontrollerer om de nødvendige signaler er aktive, og den anden del sikrer at IRQ5-pulsen har den rigtige længde.

5.5.2 Implementering

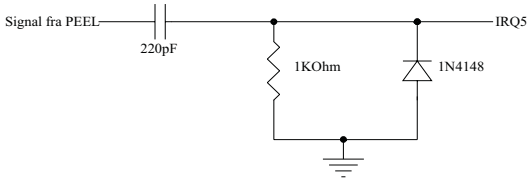
Implementeringen af IRQ-generatoren opdeles ligeledes i to:

Sammenligning af EOC, PID og PS:

Denne del implementeres med en PEEL18CV8 kreds, som programmeres således, at den kun tillader, at EOC-signalet sendes videre, hvis signalerne fra kontrolregisteret og patient ID er ens, når der korrigeres for kodning.
ABEL-programmet til kredsen ses i appendiks A.

Kontrol af længden på IRQ5:

For at fjerne IRQ5-signalet når mikroprocessoren har godkendt det, indsættes en flip-flop der sættes af signalet fra ovenstående PEEL-kreds og resettes af IACK*. Da dette EOC-signalet vil være højt indtil starten af næste AD-konvertering og derved sætte flip-flopp'en igen, er det nødvendigt at lave det om til en kort puls. Til at generere denne korte puls indsættes et højpasfilter, som kun slipper signalets flanker igennem. For at undgå negative spændinger, ved nedadgående flanker, indsættes en diode.



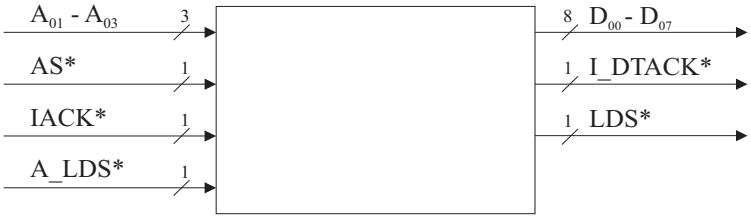
Figur 5.10 Højpasfilter til generering af kort puls.

5.5.3 Modultest

Efter programmering af PEEL-kredsen, blev der, vha. en PEEL-tester, foretaget en kontrol af denne; dvs. at de, i ABEL-programmet, nævnte testvektorer blev efterprøvet på den programmerede kreds. Længden på IRQ5-signalet blev kontrolleret vha. en logic analyzer.

5.6 IRQ-dekoder

IRQ-dekoderen har til formål at afsende en IRQ-vektor til mikrocomputeren, således at denne ved, hvilken interruptroutine, der skal udføres.
Når ADC'en er færdig med konverteringen, afsender den et EOC-signal (End Of Convertion), som kontrolleres af IRQ-generatoren, og hvis dette bliver godkendt, afsendes et IRQ5-signal til interfacet. Interfacet inverterer signalet således at mikrocomputeren modtager et IRQ5*-signal. Når denne har accepteret IRQ5*-signalet afsender den interruptlevel 5 på A_{01} - A_{03} (A_{04} - A_{23} er logisk høje), og sætter IACK* og AS* lave. Dette skulle så få IRQ-dekoderen til at lægge en IRQ-vektor ud på databussens ben D_{00} - D_{07}, hvorefter dataopsamlingsenheden sætter DTACK* lav. Mikrocomputeren læser herefter vektoren på databussen, og sætter AS* høj. Dette skal få dataopsamlingsenheden til at sætte DTACK* høj, hvorefter mikrocomputeren udfører interruptroutinen.



Figur 5.11 Input- Outputrelationen for modulet.

Input

A_{01} - A_{03}	IRQ-nummeret repræsenteres på de tre laveste adresseben.
AS*	Dette signal bliver aktivt, når adressen er stabil på bussen.
IACK*	Dette signal bliver aktivt, når mikrocomputeren har accepteret interruptet.
A_LDS*	Lower datastrobe fra adressedekoder.

Output

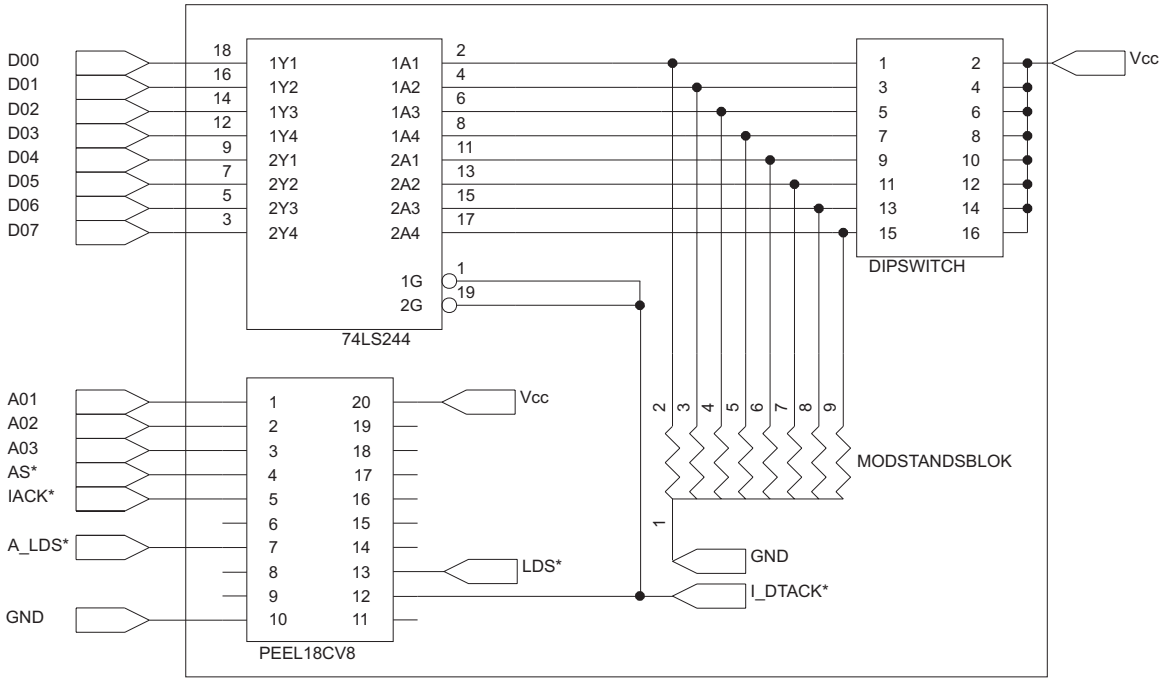
D_{00} - D_{07}	IRQ-vektoren skal repræsenteres på disse databen.
I_DTACK*	Dette signal skal afsendes til DTACK-generatoren, som tidsforsinker det, således at data når at blive stabile på databussen.
LDS*	Dette signal skal aktivere interfacets otte nederste databen.

5.6.1 Design

Da dette modul skal lagre den vektor, der sendes til mikroprocessoren, er der brug for en 8 bit hukommelse. Denne laves fysisk med kontakter, således at det er nemt at ændre den, når systemet er opbygget.

5.6.2 Implementering

Implementeringen af IRQ-dekoderen foretages med en dipswitch, en modstandsblok med otte modstande på 10 kΩ, en tristate line driver type 74LS244, samt en PEEL18CV8 kreds.
Kredsene forbindes som vist på figur 5.12.



Figur 5.12 Diagram over IRQ-dekoder

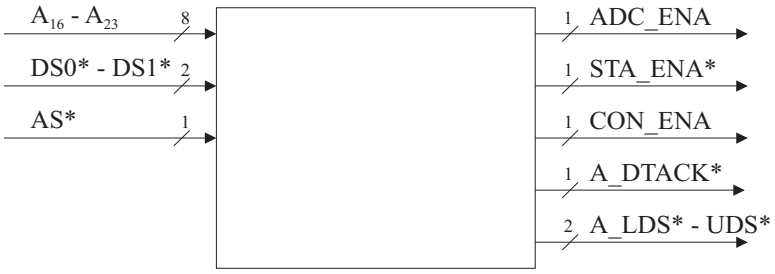
På den ene side af dipswitchen tilsluttes V_{cc}, og på den anden tilsluttes line driveren. Modstandsblokken sættes ind mellem dipswitchen og line driveren, således at modstandenes ene side er forbundet til stel. Dette medfører at, hvis dipswitchen er tændt, så vil der stå logisk "1" på line driverens ben, og hvis den er slukket vil der stå logisk "0". Dipswitchen, modstandsblokken og line driverens sammenkobling gør det muligt at indstille et vektornummer på dipswitchen, og når line driveren aktiveres, bliver denne vektor lagt ud på databussen. I dette tilfælde er dipswitchen indstillet således, at den repræsenterer vektornummeret 64.
PEEL-kredsen programmeres således, at den aktiverer line driveren, når interruptet er godkendt, dvs. når A_{01} - A_{03} indeholder den binære værdi for 5, og når AS* og IACK* er lave.
Denne aktivering foregår med signalet I_DTACK*, som også føres ud til DTACK-generatoren. Endvidere skal signalet LDS* sendes, for at aktivere interfacets otte laveste databen. LDS*-signalet skal også afsendes hvis A_LDS* bliver aktiv. Grunden til at A_LDS*-signalet er ført gennem IRQ-dekoderen er, at der ville opstå problemer med LDS*-signalet, hvis både adressedekoderen og IRQ-dekoderen forsøgte at styre dette signal samtidigt.
ABEL-programmet til PEEL-kredsen findes i appendiks A.

5.6.3 Modultest

Efter programmering af PEEL-kredsen, blev alle kredsene til modulet forbundet. Det blev kontrolleret at den indstillede vektor blev udsendt på databussen, når de korrekte input var tilstede.

5.7 Adressedekoder

Adressedekoderens opgave er at dekode de adresser, der lægges ud på VMEbussen af mikroprocessorkortet. På baggrund af disse input, genereres en række chip select-signaler til andre moduler på dataopsamlingsenheden.



Figur 5.13 Input- Outputrelationen for modulet.

Input

DS0* - DS1*	Signalerne fra mikroprocessorkortet, der styrer hvorvidt der læses fra / skrives til den øverste eller nederste databyte. Hvis begge signaler er aktiveret samtidigt, læses eller skrives der words (16 bit).
AS*	Skal være aktivt før adressen på A_{16} - A_{23} læses.
A_{16} - A_{23}	Der anvendes de 8 øverste adresseben fra VMEbussen til dekodning af de interne adresser på dataopsamlingsenheden.

Output

ADC_ENA	Bruges til chip select af udgangen på ADC'en.
STA_ENA*	Chip select til statusregisteret.
CON_ENA	Chip select til kontrolregisteret.
A_DTACK*	Chip select til DTACK-generatoren.
A_LDS*- UDS*	Benyttes til intern selektering af databussen. Har samme funktion som henholdsvis DS0* og DS1*.

Adressedekoderen skal modtage adresserne FAXXXXh og F9XXXXh på adressebenene A_{23} - A_{16} til henholdsvis statusregisteret og kontrolregisteret. Disse signaler bliver dekodet til interne chip select, hvis inputbenene AS*, DS0* og DS1* samtidigt er aktiveret (logisk 0).
På timingsdiagrammet i appendiks C ses det, at chip select-signalet, som adressedekoderen aktiverer kontrolregisteret med (CON_ENA), skal forsinkes med mindst 57 ns. Idet forsinkelsen kun må virke når kontrolregisteret aktiveres, skal deaktiveringen ske uden delay.

5.7.1 Design

Modulet opdeles i to blokke. Den første sammenligner de forskellige input som beskrevet ovenfor, og den anden forsinker CON_ENA.

5.7.2 Implementering

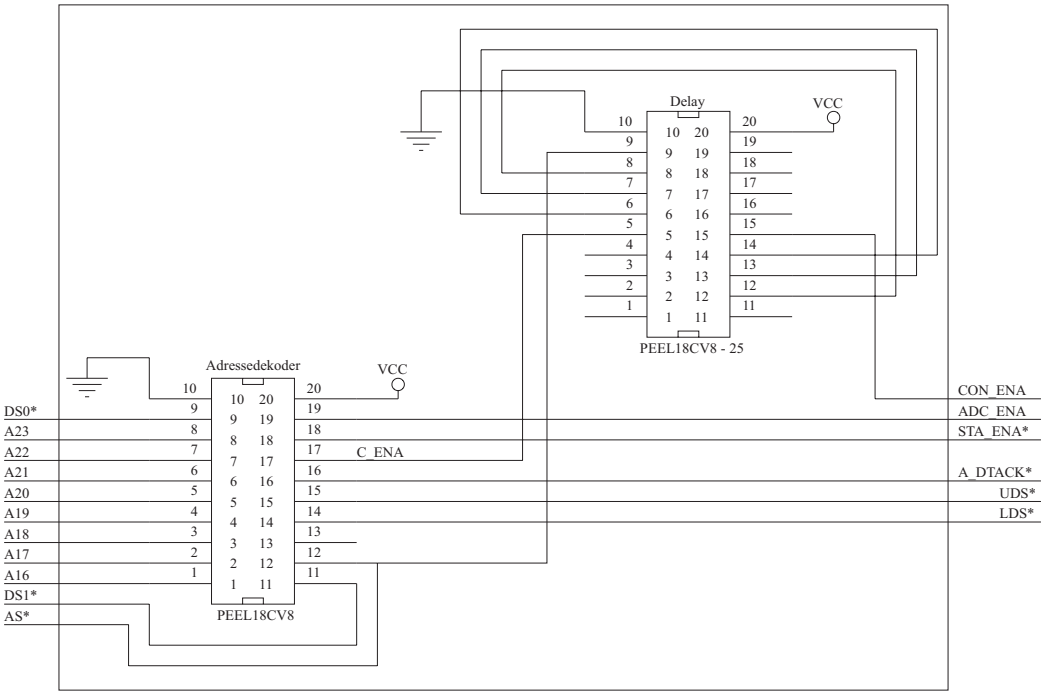
Implementeringen opdeles ligeledes i to dele: Adressedekoder og delay.

Adressedekoder:

I adressedekoderen benyttes de øverste otte adresseben. Det er kun nødvendigt, at tilgå to adresser på dataopsamlingsenheden; een adresse til statusregisteret og een adresse til kontrolregisteret.
Adressedekoderen implementeres ved anvendelse af en PLD, P18CV8. De to nævnte adresser omsættes til binære værdier, hvorved det ses, at STA_ENA* og CON_ENA skal asserteres, når der står hhv. 11111010b og 11111001b på indgangen, samtidig med at AS*, DS0* og DS1* er asserterede.

Delay af CON_ENA:

Ved at bruge en PEEL af typen 18CV8-25, som forsinker outputsignalerne med 25 ns i forhold til inputsignalerne, kan der ved tre gennemløb, opnås en forsinkelse på 75 ns, hvilket er tilfredsstillende. For at deaktivere CON_ENA uden delay, føres det forsinkede signal og AS* ind i en NOR gate, hvilket medfører at forsinkelsen kun sker ved aktivering.
Denne PEEL-kreds implementeres i den anden delay-kreds og programmet forefindes i appendiks A.



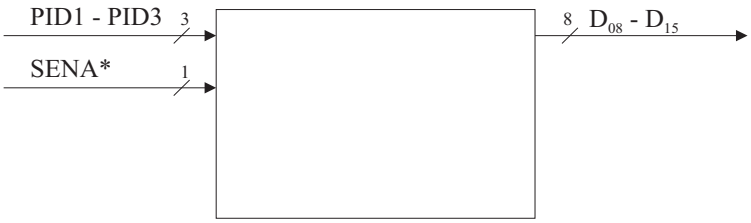
Figur 5.14 Diagram over modulet.

5.7.3 Modultest

Adressedekoderen er testet i en PEEL-tester, hvor de forskellige kombinationer er afprøvet.
For kontrol af delay, se afsnit 5.4.3. (Modultest af clockdivider).

5.8 Statusregister

Statusregisteret er en buffer, der holder på data, som skal sendes fra dataopsamlingsenheden til mikroprocessorkortet. Registerets primære opgave er at lagre data i det tidsrum, der går fra at de er beregnet af dataopsamlingsenheden, til mikroprocessorkortet er klar til at modtage dem.



Figur 5.15 Input- Outputrelationen for modulet.

Input

PID1 - PID3: Patient ID er en 3 bit kode fra ADC/MUX-modulet, som angiver hvilken patient det pågældende sample kommer fra.

STA_ENA*: Status Enable er et signal, som får modulet til at sende data til VMEbusinterfacet.

Output

PID1 - PID3: Det samme signal, som kom ind i modulet, sendes videre til D_{08} - D_{10} på VMEbusinterfacet. D_{11} - D_{15} lægges til stel, hvorved der sættes nuller på udgangen.

5.8.1 Design

Da modulet blot består af en buffer, er der ikke nogen egentlig designopgave forbundet med dette modul.

5.8.2 Implementering

For at udføre dette moduls funktion kræves een enkelt kreds; SN74LS541. Det er en 8 bit buffer med mulighed for at tri-state udgangen.

Forbindelser

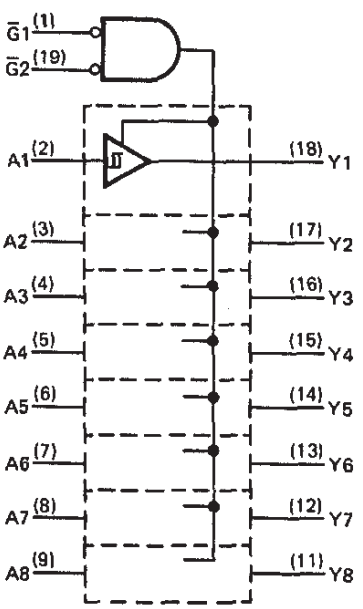
G1* - G2* Som det fremgår af figur 5.16, er outputtet tristatet, hvis enten G1* eller G2* er høj. Derfor holdes G2* permanent lav, således at STA_ENA* forbindes til G1*.

V_{CC}: +5 V forsyning

A1-A3 PID1 - PID3.

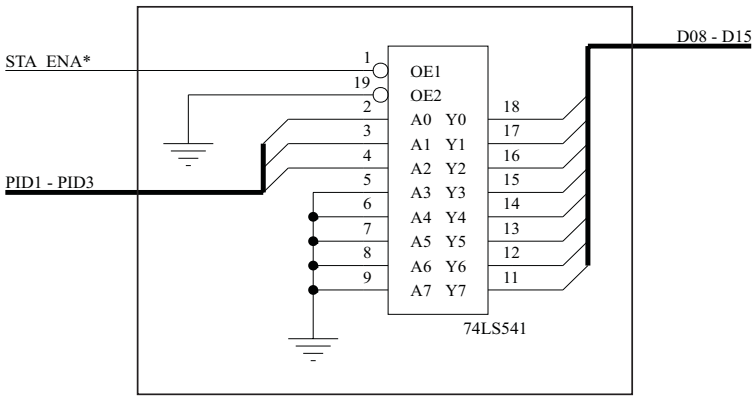
A4-A8 Stel, 0 V

Y1-Y8 D_{08}-D_{15} på VMEbusinterfacet.



Figur 5.16

Modulet implementeres efter dette diagram:



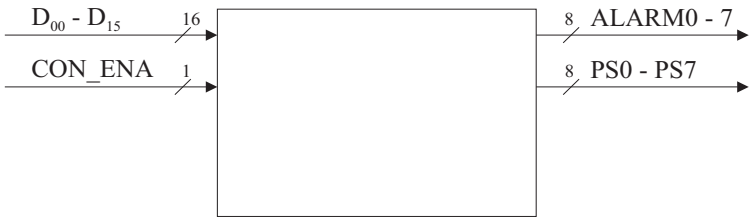
Figur 5.17 Diagram over modulet.

5.8.3 Modultest

Modulet kontrolleres ved at opkoble kredsen i laboratoriet, sætte forskellige bit-kombinationer ind på indgangen, og konstatere at de kunne måles på udgangen.

5.9 Kontrolregister

Kontrolregisterets opgave er at huske de indstillinger, der kommer fra mikrocomputeren, således at IRQ-generatoren og alarmen hele tiden kan kontrollere, hvilke patienter der er tilsluttet, og hvilke der er alarm ved.



Figur 5.18 Input- Outputrelationen for modulet

Input

D_{00}-D_{07}	Angiver tilsluttede patienter.
D_{08}-D_{15}	Angiver eventuelle alarmer.
CON_ENA	Får registeret til at latche data på D_{00}-D_{15}.

Output

Alarm0 - 7	Angiver patienter med alarm.
PS0 - PS7	Fortæller IRQ-generatoren hvilke patienter, der er tilsluttet.

5.9.1 Implementering

Der vælges to 8 bit registre af typen 74LS373, som har den egenskab, at der er mulighed for at disable indgangen, hvorved de otte latches beholder deres værdi uafhængigt af, hvad der står på indgangen. Der vælges eet register til hver af de to sæt outputsignaler.

Forbindelser

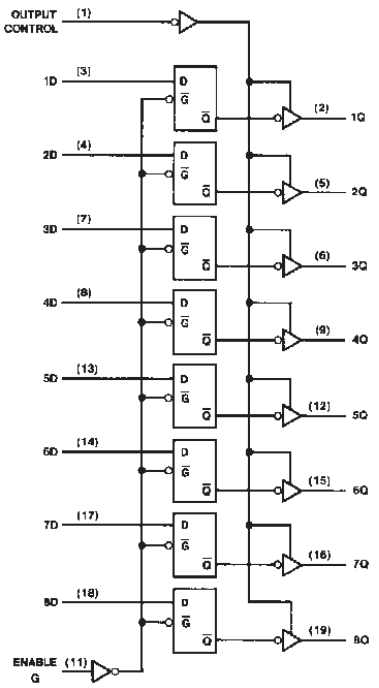
Her angives hvordan de enkelte ben på kredsene forbindes.

Kreds nr. 1 (Patienter tilsluttet):

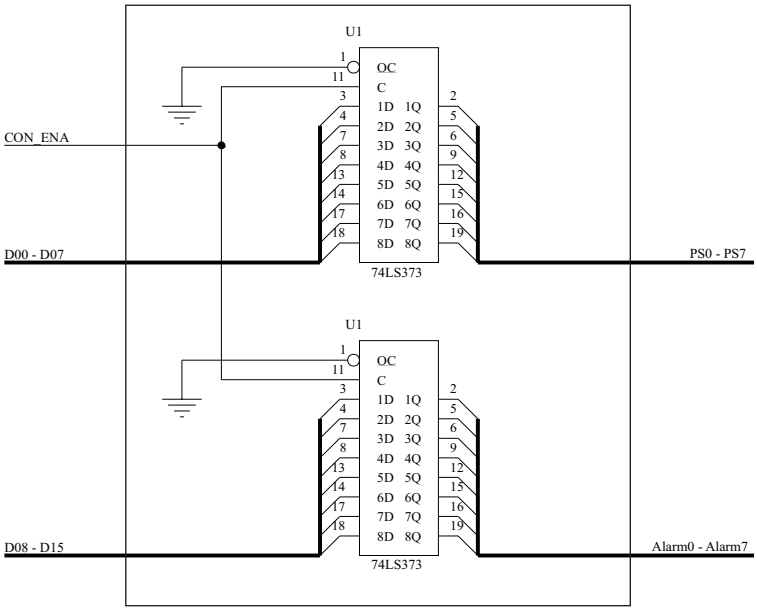
Output Control: Permanent lav, da outputtet altid skal være tilgængeligt for IRQ-generatoren.
1D-8D: D_{00}-D_{07}
Enable G: CON_ENA fra adressedekoder.
1Q-8Q: IRQ-generator.

Kreds nr. 2 (Aktuelle alarmer):

Output Control: Permanent lav, da outputtet altid skal være tilgængeligt for alarmen.
1D-8D: D_{08}-D_{15}
Enable G: CON_ENA fra adressedekoder.
1Q-8Q: Alarmmodul.



Figur 5.19 Logisk diagram over 74LS373.



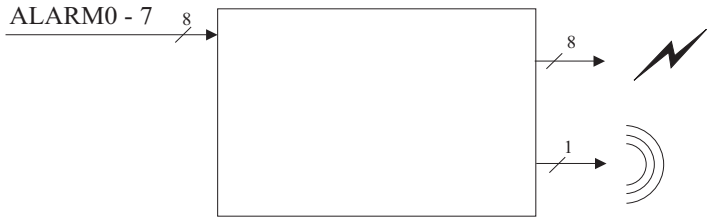
Figur 5.20 Diagram over modulet.

5.9.2 Modultest

Modulet blev kontrolleret ved at opkoble kredsen i laboratoriet, sætte forskellige bit-kombinationer ind på indgangen, og konstatere at de kunne måles på udgangen.

5.10 Alarmmodul

Dette modul modtager udgangssignalet fra den del af kontrolregisteret, som indeholder information om alarmstatus, og aktiverer evt. en alarm.



Figur 5.21 Input- Outputrelationen for modulet.

På figuren er der symboliseret et auditivt og et visuelt output. Dette laves med lysdioder og en højttaler. Signalerne tildeles følgende navne.

Input

ALARM0 - 7	Disse input til alarmmodulet kommer fra kontrolregisteret, og repræsenterer otte patienter. Der benyttes 8 databit, for at kunne aktivere alarm for flere patienter af gangen.

Output

LED0 - 7	Lys der indikerer alarmer ved de respektive patienter.
SPEAKER	Lydsignal som indikerer, at der er alarm ved een af de tilsluttede patienter.

5.10.1 Design

Designet opdeles i 3 dele: Alarmgenerator, lysgenerator og højttaler.

- Alarmgeneratoren skal, hvis et eller flere af signalerne ALARM0 - 7 er høje, generere et signal til den eller de lysdioder, som repræsenterer de pågældende patienter. Endvidere skal der udsendes et signal, der aktiverer højtaleren. Den sidste opgave kan varetages af en simpel OR-funktion.
- Lysgeneratoren laves med otte lysdioder, som placeres på en række, så det tydeligt fremgår, hvilken patient der er tale om.

Lysdioderne indsættes på alarmgeneratorens udgangsben, LED 0 - 7 i lederetningen, og serieforbindes med en modstand.
SPEAKER-signalet forbindes til en højttaler. For ikke at skulle generere et sinussignal til denne højttaler, bruges der en piezo-elektrisk højttaler. Denne højttaler har et indbygget båndpasfilter, således at den kun udsender lydbølger med frekvensen 4 kHz ± 500 Hz.

5.10.2 Implementering

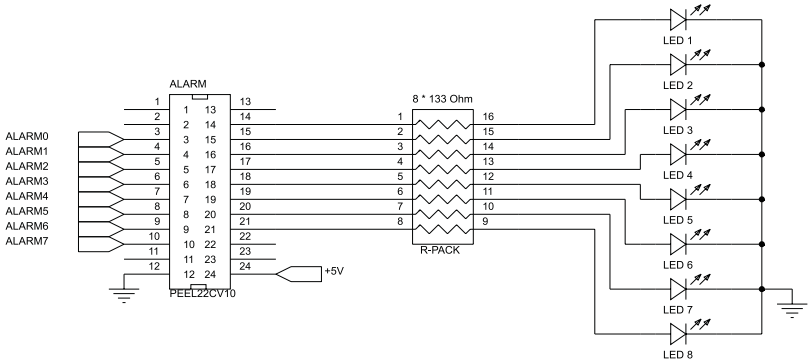
Implementeringen er ligeledes opdelt i tre dele.

Alarmgenerator:
På baggrund af funktionskravene til alarmgeneratoren, indsættes en PEEL-kreds af typen PEEL22CV10A. Denne programmeres således, at otte outputben følger de otte input. Disse sendes videre til lysgeneratoren.
På det sidste outputben ligger resultatet af en OR-funktion af alle inputbenene, som sendes videre til højttaleren.

Lysindikering:
Da der er anvendt standard TTL-logik i dataopsamlingsenheden, forudsættes det, at spændingsniveauet på udgangen af alarmmodulet, når det er aktivt, ligger på 2,8V - 5 V. Derfor regnes der her med 3,5 V. I databladet for lysdioderne oplyses det, at de afgiver et let synligt lys ved en strøm på 25 mA. Derfor vælges modstanden herefter.
I følge databladet har lysdioden, ved de pågældende spændinger og strømme, en indre modstand på ca. 5 Ω.

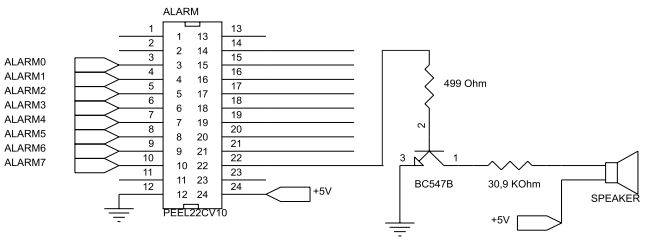
$$25mA = \frac{3{,}5V}{5\Omega + R} \Leftrightarrow R = \frac{3{,}5V - 0{,}125V}{0{,}025A} = 135\Omega \qquad \textbf{(5.4)}$$

Der vælges en modstand på 133 Ω til serieforbindelse med lysdioden.



Figur 5.22 Diagram over lysgeneratoren.

Højtaler:
Signalet fra alarmgeneratoren skal tilpasses højtaleren, og derfor indsættes en CE (Common Emitter) transistorkobling som driver.
Højtaleren indsættes på collector-benet af transistoren i serie med en modstand på 30,9 kΩ, for ikke at få et for højt lydsignal. På transistorens basis indsættes en modstand på 499 Ω.

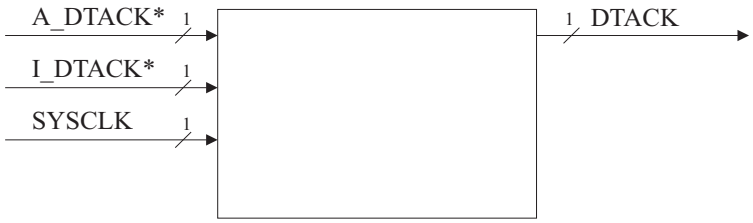


Figur 5.23 Diagram over højtalerens opkobling.

5.10.3 Modultest
Alarmmodulet blev testet ved at sætte forskellige signaler ind på PEEL-kredsens indgangsben; pin 3-10 og betragte udgangssignalet.

5.11 DTACK-generator

DTACK-generatoren skal varetage timing mellem mikroprocessorkortet og dataopsamlingsenheden. Dvs. at dataopsamlingsenheden skal afgive et DTACK-signal, når den er klar til enten at afgive eller modtage data.



Figur 5.24 Input- Outputrelationen for modulet.

Input

A_DTACK*	Kommer fra adressedekoderen når en DTACK skal afsendes. Signalet er aktivt indtil AS* eller DS0* og DS1* på adressedekoderen deaktiveres.
I_DTACK*	Kommer fra IRQ-dekoderen når en DTACK skal afsendes. Signalet er aktivt indtil AS* på IRQ-dekoderen deaktiveres.
SYSCLK	Systemclocken fra VMEbussen på 16 MHz. Denne benyttes til at forsinke signalerne A_DTACK* og I_DTACK*.

Output

DTACK	Fortæller mikroprocessorkortet, at data er blevet sendt eller modtaget. Signalet inverteres af VMEbusinterfacet, således at mikroprocessoren modtager DTACK*.

5.11.1 Design

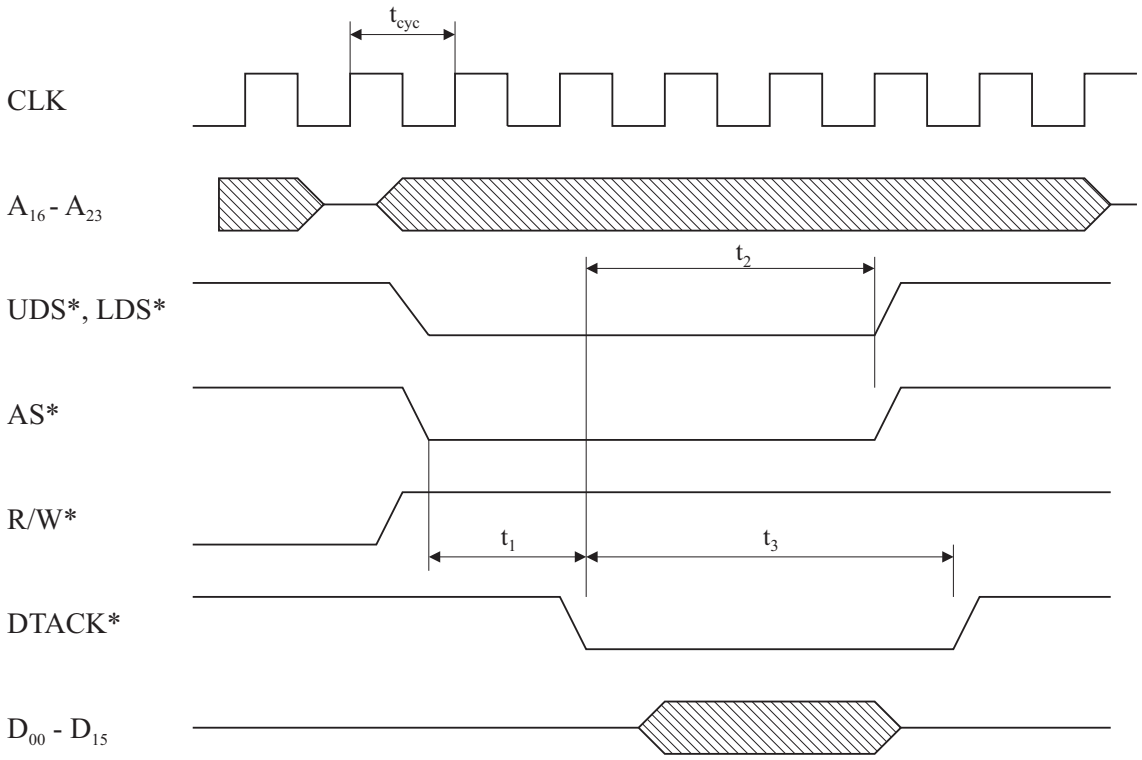
Før DTACK-generatoren kan implementeres, er man nødt til at vide, hvordan den interne timing er på dataopsamlingsenheden, lige såvel som man skal vide, hvordan timingen er i en read-, write- og interruptcycle fra mikroprocessorkortet. For at få dette overblik, gennemgås de tre forskellige cycles.

Read cycle:

Ved betragtning af de signaler, i en read cycle, som har betydning for DTACK-signalets udseende, kan timingen af DTACK-signalet fastlægges. Det er dog væsentligt at undersøge, hvorvidt DTACK-signalet lever op til de øvrige timingkrav i henholdsvis write- og interrupt cyclen.

På figur 5.25 er vist de væsentlige signaler i en read cycle fra mikroprocessoren. De angivne tider, er de tider, som har betydning for det DTACK-signal, som dataopsamlingsenheden skal afgive.

Tiden t_{cyc}, som er cycle tiden for mikroprocessorens interne clock, er 125 ns, da den kører med en frekvens på 8 MHz. Hvor lang tiden t_1 er, afhænger af den interne timing på dataopsamlingsenheden. Ved betragtning af timingdiagrammet for dataopsamlingsenheden i appendiks C ses det, at den maximale tid der går, fra at AS* bliver aktiveret på adressedekoderen, til at data er klar på bussen, er 262 ns. Det vil sige, at DTACK-signalet skal forsinkes med minimum 262 ns i forhold til AS*.



Figur 5.25 De væsentligste signaler og tider i en read cycle fra mikroprocessoren hvad angår DTACK signalet.

Tiden t_2 er specificeret i [Clements, 1997] til max 215 ns, hvorved tiden t_3 skal vælges til en større værdi, så DTACK-signalet ikke afsendes to gange. På baggrund af disse betragtninger kan der opstilles følgende krav til DTACK-pulsen:

- Delay (t_1): min 262 ns.
- Pulsbredde (t_3): min 215 ns.

Write cycle:
Kravene til DTACK-pulsen, ved en write cycle, findes ligeledes ved at betragte timing-diagrammet for dataopsamlingsenheden i appendiks C. Derved fås, at DTACK-signalet skal forsinkes minimum 75 ns i forhold til signalet A_DTACK* (kommer samtidigt med A_LDS* og UDS* på dataopsamlingsenhedens interne databus). Den minimale pulsbredde af DTACK-signalet kan findes til 215 ns ved at betragte mikroprocessorens write cycle. Der kan derved opstilles følgende krav til DTACK-pulsen:

- Delay (t_1): min 75 ns.
- Pulsbredde (t_3): min 215 ns.

Interrupt cycle:
Specifikationerne for denne cycle findes på samme måde som for de øvrige cycles, og derved fås følgende krav til DTACK-pulsen:

- Delay (t_1): min 64 ns.
- Pulsbredde (t_3): min 215 ns.

DTACK-pulsens egenskaber:
På baggrund af ovenstående betragtninger af de forskellige cycles, kan der opstilles nogle endelige krav til DTACK-generatoren. For kun at skulle afgive een type DTACK-puls, skal der vælges en DTACK, som lever op til alle timingbetingelserne. Dvs. et delay på min. 262 ns og en pulsbredde på min. 215 ns. Da DTACK-pulsen skal forsinkes på baggrund af inputtet SYSCLK, vil det være en fordel at vælge et delay og en pulsbredde, som er et multiplum af SYSCLK'ens periodetid. SYSCLK har en frekvens på 16 MHz, hvilket svarer til en periodetid på 62,5 ns. Der vælges følgende specifikationer:

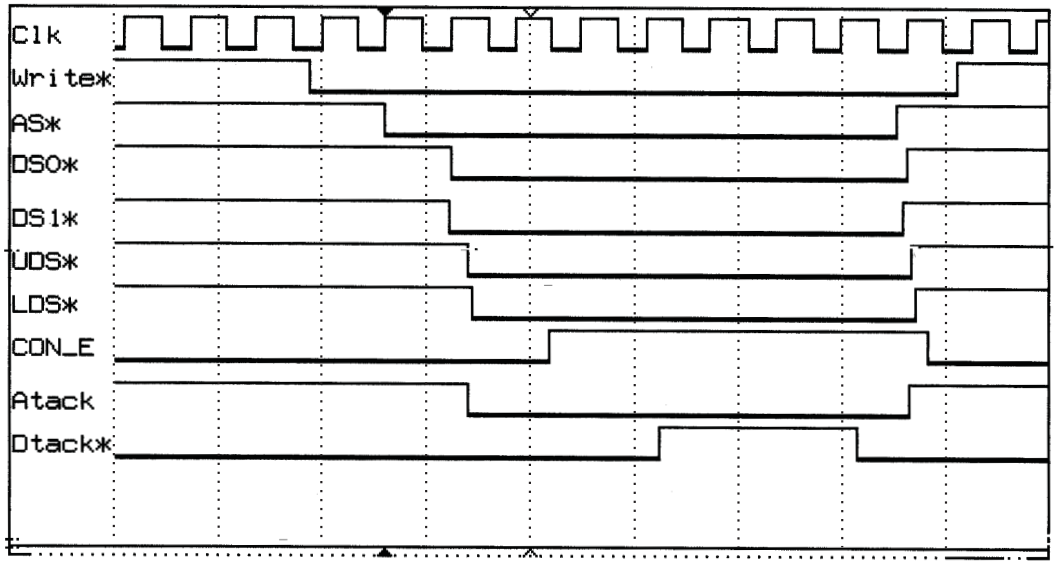
- Delay: 312,5 ns.
- Pulsbredde: 250 ns.

5.11.2 Implementering
Modulet kan nu implementeres, så det opfylder de to ovenstående krav. Dette gøres ved at programmere en PEEL-kreds af typen P18CV8. Kredsen programmeres således at den forsinker pulsen med 5 clock cycles og gør den 4 clock cycles bred.
Programkoden til denne PEEL-kreds kan findes i appendiks A.

5.12 Modulintegration og procestest

For at teste dataopsamlingsenheden, programmeres et lille program, som skriver til kontrolregisteret og læser fra statusregisteret. Ved at måle med en logic analyzer, vil det være muligt at undersøge kontrolsignalerne for hver cycle. I appendiks C er der en beskrivelse af en write, read og IRQ cycle.

5.12.1 Write cycle
Ved at skrive forskellige værdier til kontrolregisteret og måle hvad der står i dette, kan det konstateres, om mikroprocessoren kan kommunikere med kontrolregisteret.



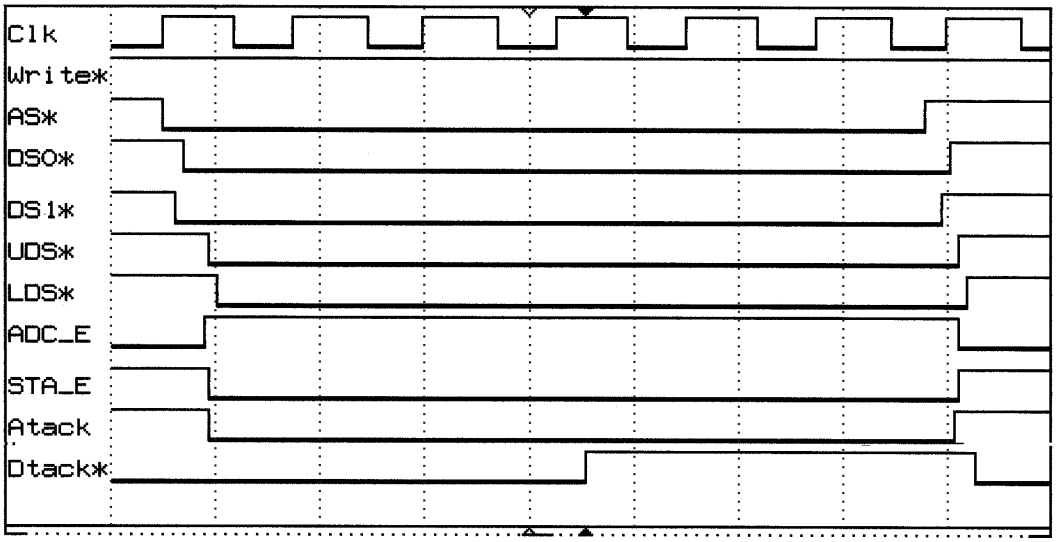
Figur 5.26 Kontrolsignaler fra en write cycle, målt med en logic analyzer. (100 ns / div)

Måling på intern timing på dataopsamlingsenheden i en write cycle			
Parameterbeskrivelse	Symbol	Beregnet	Målt
Tid fra DS0* - DS1* på VMEbus til UDS*, A_DTACK* på intern bus.	t_1	<28 ns	18 ns
Tid fra UDS* til LDS*.	t_2	<10 ns	4 ns
Tid fra UDS* til CON_ENA.	t_3	>57 ns	78 ns
Tid fra CON_ENA til DTACK*.	t_4	>21 ns	106 ns
Tid fra DTACK* aktiveret på VMEbussen til AS* deaktiveres på VMEbussen.	t_5	<215 ns	210 ns
Tid fra AS* deaktiveret til CON_ENA deaktiveret.	t_6	<40 ns	30 ns

Idet det lykkedes at skrive til kontrolregisteret og måle de forventede værdier, må det konstateres at denne kommunikation virker. Endvidere kan det på figur 5.26 ses at kontrolsignalerne stemmer overens med de forventede fra appendiks C.

5.12.2 Read cycle

Denne cycle kontrolleres ved at læse data fra statusregisteret og ADC'en. Der tilsluttes otte fastlagte spændinger på ADC'ens analoge input, hvorefter programmet gemmer data fra ADC og statusregister i et bestemt adresseområde. Herefter kan det undersøges hvorvidt data fra et bestemt input på ADC'en, er gemt sammen med det tilhørende patient ID.



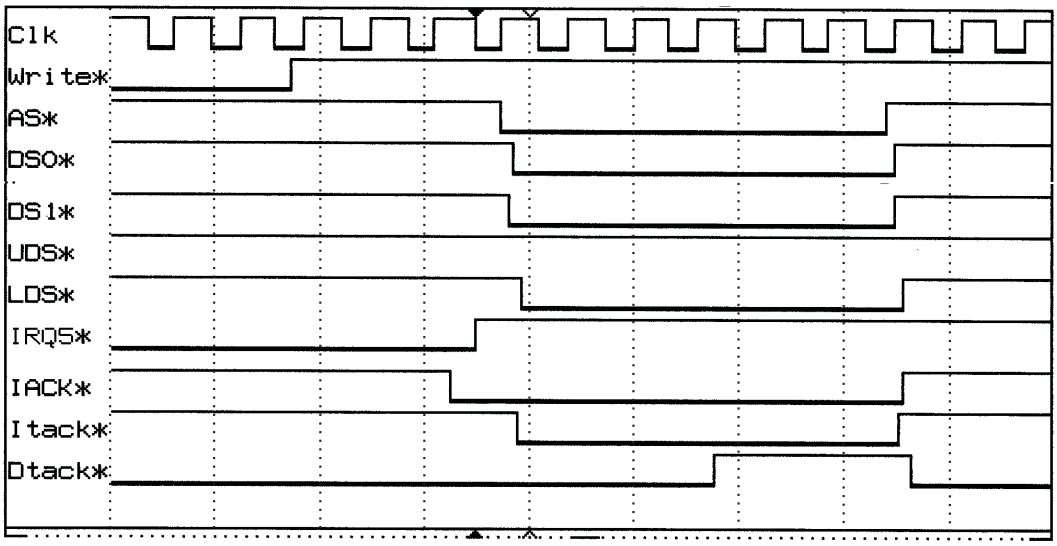
Figur 5.27 Kontrolsignaler fra en read cycle. (50 ns / div)

Måling på intern timing på dataopsamlingsenheden i en read cycle			
Parameterbeskrivelse	Symbol	Beregnet	Målt
AS* på VMEbus til UDS*, ADC_ENA, STA_ENA*, A_DTACK* på intern bus.	t_1	<28 ns	10 ns
Tid fra UDS* til LDS*	t_2	<10 ns	6 ns
ADC_ENA til data er klar på VMEbussen	t_3	<262 ns	2 ns
Delay fra A_DTACK* til DTACK*	t_5	>262 ns	310 ns
DTACK* pulsbredde	t_7	>215 ns	240 ns

Idet det lykkedes at gemme de ønskede værdier fra statusregisteret og ADC'en, må det konkluderes, at denne kommunikation virker. Det blev dog konstateret, at værdierne fra ADC'en og statusregisteret ikke altid passede sammen. Denne fejl skyldes at læsningen ved testen ikke er IRQ-styret, og derfor kan data være læst under en AD-konvertering. Dette betyder, at der læses gamle data sammen med nyt patient ID. Dette problem opstår ikke i det færdige system hvor læsningen er IRQ-styret.

5.12.3 Interrupt cycle

For at teste denne cycle sættes mikroprocessoren til at modtage interrupt, og ved at måle en interrupt cycle, med en logic analyzer, kan det undersøges om delay'ene er af passende længde.



Figur 5.28 Kontrolsignaler fra en interrupt cycle. (100 ns / div)

Målinger på intern timing på dataopsamlingsenheden i en interrupt cycle			
Parameterbeskrivelse	Symbol	Beregnet	Målt
Tid fra AS* til LDS* på intern bus.	t_1	<28 ns	10 ns
Tid fra I_DTACK* til DTACK*.	t_3	>30 ns	310 ns
DTACK* aktiveret på VMEbussen til AS* deaktiveres på VMEbussen.	t_4	<215 ns	164 ns
AS* deaktiveret til I_DTACK* er deaktiveret.	t_6	<28 ns	12 ns

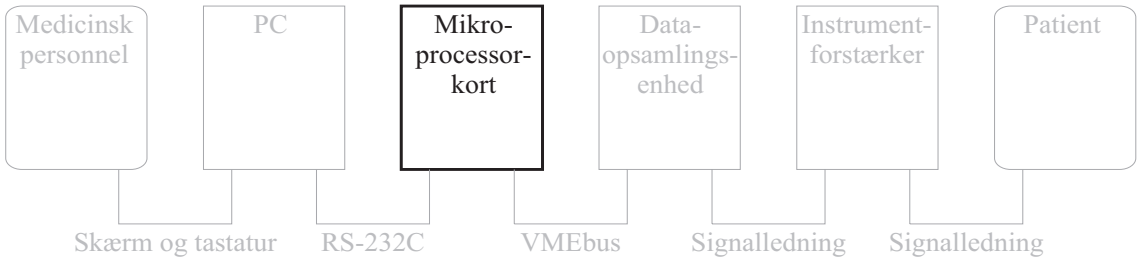
Som det fremgår af tabellen er alle delay overholdt, og derfor antages det at interrupt cyclen er korrekt. For at kontrollere om det var muligt at styre en read cycle ved hjælp af interrupt, blev read cycle programmet omprogrammeret, således at det blev interruptstyret. Efter denne omprogrammering, blev der igen indlæst data fra ADC og statusregister, og det blev konstateret at de indlæste data altid var synkroniseret.

5.12.4 Konklusion

Idet det lykkedes at læse fra statusregisteret og ADC'en ved hjælp af interrupt, samt at skrive til kontrolregisteret, må det konstateres at dataopsamlingsenheden lever op til kravene.

6 Mikroprocessorprogram

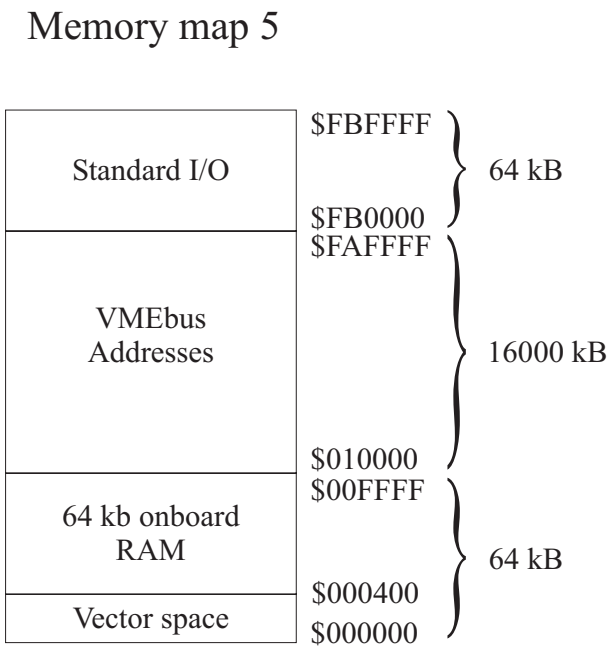
Denne proces omfatter den software, der skal afvikles på Motorola mikroprocessorkortet. Der skal designes software der muliggør at mikroprocessorkortet kan kommunikere med dataopsamlingsenheden, samt software som udfører de funktioner, der er beskrevet i kravsspecifikationen. Endvidere skal programmet kommunikere med en PC via RS232C.



Programmerne bliver designet efter princippet “funktionsorienteret top-down”.
Idet softwaren på mikroprocessorkortet kan inddeles i flere overordnede funktioner, er den delt i flere delprogrammer. Disse delprogrammer er inddelt efter deres overordnede funktion og disse er; et operativsystem, et applikationsprogram samt en hardwaredefinition (RAM definition). Operativsystemets hovedfunktioner er at varetage kommunikationen mellem hardware og operativsystem. Dvs. at de hardwarenære ting implementeres i operativsystemet. Applikationens hovedfunktion er at bearbejde de data, der kommer fra dataopsamlingsenheden, gennem operativsystemet, og sende de relevante data videre til PC ved et kald til operativsystemet.
Endvidere er der lavet en softwaremæssig hardwaredefinition, der skal sørge for, at de enkelte dele i programmerne placeres de rigtige steder. Før der gås i detaljer med, hvad de enkelte programdele skal udføre, og hvordan grænsefladerne mellem dem skal være, vil hardwaredefinitionen blive lavet.

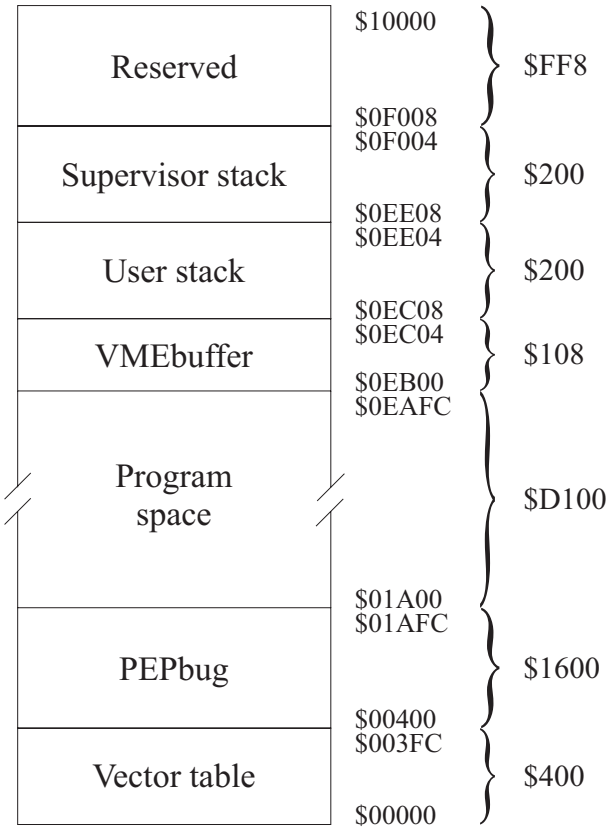
6.1 Hardwaredefinition

På det udleverede Motorola mikroprocessorkort er der integreret et VMEbusinterface samt en RS232C kompatibel kommunikationsport. Endvidere er der på kortet 64 kB RAM og 64 kB ROM. I kortets ROM ligger et preinstalleret operativsystem, PEPbug, som også benytter et område af RAM. På nedenstående figur ses det memory map, som er standard på Motorola kortet.



Figur 6.1 Adresseområdet som Motorola mikroprocessorkortet kan adressere når det er indstillet til memory map 5. Dette gøres vha. af jumpere på kortet.

Det øverste adresseområde "Standard I/O" benyttes internt på mikroprocessorkortet, til bl.a. at adressere UART'en, hvis adresseområde ligger mellem FEFFE0h og FEFFFFh.
Det næste adresseområde, som er det største, er VMEbus adresseområdet. Dette område benyttes til at adressere de ydre enheder, som tilkobles mikroprocessorkortet. Dataopsamlingsenheden benytter to adresser i dette område til henholdvis kontrol- og statusregister. Idet der ikke benyttes fuld adressedekodning, bliver et større adresseområde dog optaget. Dette anses ikke som værende et problem, idet der stadig er meget adresserum ledigt i dette område.
De nederste 64 kB af adresseområdet benyttes af de 64 kB RAM, der sidder på mikroprocessorkortet. Det er, som tidligere nævnt, ikke muligt at benytte hele dette område til eget programmel. De nederste 400h benyttes til en vektortabel, som bliver "mirrored" fra mikroprocessorkortets ROM, hver gang den resettes. Endvidere benyttes de næste 1600h af PEPbug, hvilket betyder at området fra 000000h - 001A00h ikke kan benyttes. Det giver et arbejdsområde fra 001A00h til 010000h, til den software der skal udarbejdes.
Softwaren er delt op i to hoveddele; et operativsystem og en applikation. For at holde styr på, hvor de forskellige dele af softwaren ligger, laves der en hardwaredefinition (RAM definition). Hvordan RAM-området er opdelt kan ses på figur 6.2.



Figur 6.2 Fordeling af de 64 kB RAM som sidder på mikroprocessorkortet VMPM 68 KA-2. De forskellige områder i RAM'en repræsenterer det software der skal ligge der.

Det ses, at de nederste 400h benyttes til en vektortabel, som er en tabel, der indeholder pointere til de adresser, som benyttes til exeption handlers og interrupt handlers. I PEPbug benyttes TRAP #14 vektoren, som ligger på adresse B8h. Idet der benyttes interrupt på dataopsamlingsenheden, skal der ligeledes vælges en pointer til en adresse, hvor der skal ligge en interrupt handler til denne. Denne pointer er placeret i 100h.
De øverste ca. 4 kB RAM "Reserved" er reserveret til eventuelle testprogrammer, som man kunne få brug for. Der er afsat 200h til henholdsvis en User stack og en Supervisor stack som benyttes af applikationen og operativsystemet. User stack benyttes ikke, da der i programmet altid køres i supervisor mode. Dette betyder, at kun Supervisor stack benyttes, men for en sikkerheds skyld er der afsat plads til en User stack. Dette gør, at programmet ikke "crasher" hvis det, på trods af forventningerne, skulle komme i User mode.
Der er afsat 108h til VMEbuffer, som er en ringbuffer, der skal indholde samples fra dataopsamlingsenheden. Endvidere er der afsat ca. 52 kB RAM til applikationssoftwaren og operativsystemet.

6.2 Opdeling i Moduler

Da softwaren, som tidligere nævnt, er inddelt i to delprogrammer, vil opdelingen i moduler blive gjort selvstændigt for de to delprogrammer. Der følger først en beskrivelse af de funktioner, som det enkelte program skal udføre, hvorefter der følger en egentlig modulopdeling.

6.2.1 Applikation

Denne del af softwaren, som varetager de mere overordnede opgaver, skrives i programmeringssproget C, idet det giver den største overskuelighed. Det giver til gengæld en maskinkode, der ikke er så kompakt, som hvis den var skrevet direkte i assembler. Opgaverne, som denne applikation skal varetage, beskrives kort nedenfor.

Initialize
Denne opgave består i at initialisere de globale variable, der benyttes i programmerne. Der skal endvidere foretages nogle hardwaremæssige initialiseringer, som skal varetages ved et kald til en initialiseringsprocedure i operativsystemet.

Test data
Her testes de data, som kommer fra PC'en. Dvs. at der skal undersøges, om protokollen mellem PC og mikroprocessorkortet, som er defineret i systemdesign, bliver overholdt.

Min/Max puls
Disse to funktioner kaldes, hvis min eller max puls skal sættes. Funktionerne kaldes hvis PC har sendt en "min/max puls" til mikroprocessorkortet.

Set Patient
Denne funktion kaldes, hvis en patient skal tilsluttes eller frakobles på dataopsamlingsenheden.

Decode
Her dekodes de data, som kommer fra PC. Dvs. at de tegn, som er benyttet til "indpakning" af data, pga. RS232 protokollen, fjernes, og data videresendes til den relevante funktion.

Data from PC
Denne funktion skal undersøge om der er data fra PC. Der skal foretages et kald til operativsystemet, som kommunikerer med UART'en på mikroprocessorkortet.

Encode / Encode EKG
Funktionerne encode benyttes til at "pakke data ind" i nogle tegn som defineret i RS232 protokollen. Herefter skal der foretages et kald til operativsystemet, som kommunikerer med UART'en på mikroprocessorkortet, hvorved data kan sendes til PC.

Puls calculation
I denne funktion foretages den egentlige databehandling. Funktionen skal på baggrund af EKG-signalet, som ligger i VMEbuffer i operativsystemet, beregne en puls. Hvis pulsen er under et minimum eller over et maximum, skal der kaldes en funktion i operativsystemet, som starter alarmen. Der skal endvidere sendes data til funktionen "encode", som sender en alarm til PC.

EKG reduction
Da EKG-signalet, som ligger i VMEbuffer, er samplet med 500 Hz, er det en fordel kun at videresende nogle af dataene, da alle disse samples alligevel ikke kan vises på skærmen. Denne funktion udtager nogle af de samples, som ligger i VMEbuffer og kalder derefter en funktion i operativsystemet, som sender data over RS232C.

På baggrund af de ovenstående opgaver, kan applikationen opdeles i nogle moduler. Modulerne benævnes med APP_NAVN (Ene store bogstaver) for nemmere at kunne identificere, hvor de

enkelte moduler hører hjemme. Opgaver bliver varetaget af en række C-funktioner, som er navngivet således, at det fremgår hvilke opgaver de varetager. Funktionsnavnene er angivet i parentes i den følgende liste. Modulerne i applikationen er:

APP_INIT: Varetager opgaven Initialize. (Init).

APP_DECODE: Varetager opgaverne Data from Pc, Test data, Decode, Min/Max puls og Set patient. (DataFromPc, TestData, Decode, MinPuls, MaxPuls, PatientON_OFF).

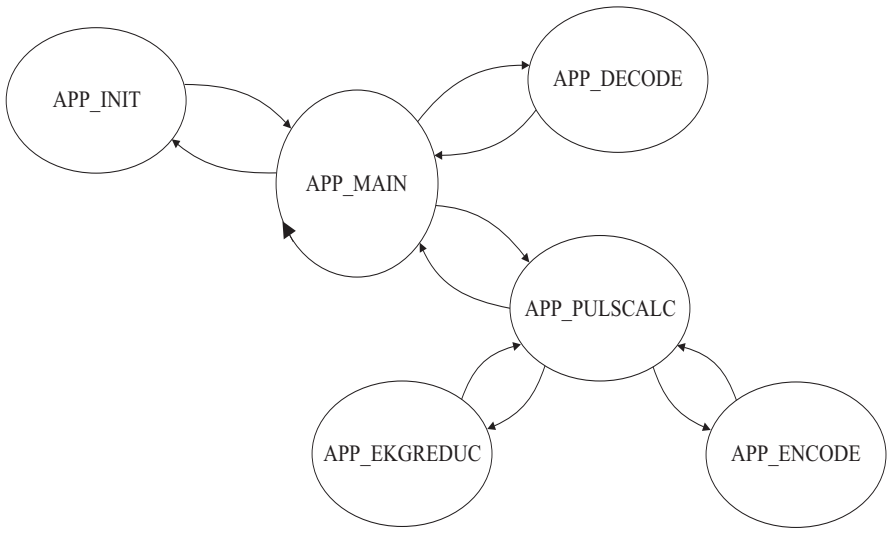
APP_ENCODE: Varetager opgaven Encode og Encode EKG. (Encode, EncodeEKG).

APP_EKGREDUC: Varetager opgaven EKG reduction. (EKGReduction).

APP_PULSCALC: Varetager opgaven Puls calculation. (PulsCalculation).

APP_MAIN: Indeholder mainløkken, som kalder de øvrige funktioner. (main).

På nedenstående figur ses et statediagram, som viser, hvordan modulerne i applikationen kaldes fra APP_MAIN. De kald, som foretages til operativsystemet fra de enkelte moduler, er ikke angivet.



Figur 6.3 Statediagram for applikationen på Motorola mikroprocessorkortet. Funktionskald til operativsystemet er ikke vist på dette diagram. Det bør bemærkes at modulet APP_INIT kun kaldes een gang.

6.2.2 Operativsystem

På samme måde som for applikationen, kan de opgaver, som operativsystemet skal udføre, opstilles. Der følger derfor en beskrivelse af disse opgaver.

Initialize

Denne funktion skal initialisere systemet før applikationen køres. Dvs. at initialisere stackpointere, statusregistre, vektortabel og lignende.

Warmstart / Coldstart
Disse funktioner implementeres i operativsystemet for at kunne resette systemet.

Mask / Unmask IRQ
Funktionerne benyttes til at maske IRQ'en ind og ud, alt efter hvornår man ønsker at IRQ skal komme igennem.

Start / Stop alarm
Starter og stopper alarmen ved at skrive til den øverste byte i kontrolregistret på data-opsamlingsenheden.

Set patient
Tilkobler eller frakobler en patient på dataopsamlingsenheden. Dette gøres ved at skrive til den nederste byte i kontrolregistret på dataopsamlingsenheden. Det er dog kun muligt at at tilkoble eller frakoble den første patient pga. softwareafgrænsningen i kravsspecifikationen.

UART test
Denne funktion skal undersøge, om der er data i UART'en.

Read / Write RS232
Disse funktioner sørger for at bringe data fra PC til mikroprocessorkort og omvendt.

Read / Write / Init buffer
Funktionerne læser og skriver i VMEbuffer, som er en ringbuffer, der indholder EKG-samples fra dataopsamlingsenheden. Init buffer benyttes til at initialisere bufferen, dvs. read- og writepointer.

IRQ routine
Denne rutine kaldes når dataopsamlingsenheden afgiver en IRQ. Rutinen kalder WriteBuffer som skriver et sample ind i VMEbuffer. Denne rutine kaldes automatisk pga. pointeren der ligger i vektortabellen.

Disse opgaver kan nu deles ud i nogle moduler. Modulerne benævnes med OS_NAVN for at angive hvor modulerne hører til.

OS_INIT:	Varetager opgaverne Initialize, Warmstart og Coldstart. (_Initialize, _Warmstart, _Coldstart).
OS_MASKIRQ:	Varetager opgaverne Mask IRQ og Unmask IRQ. (_MaskIrq, _UnmaskIrq).
OS_DAOE:	Varetager opgaverne Start alarm, Stop alarm og Set patient. (_StartAlarm, _StopAlarm, _SetPatient). DAOE er en forkortelse for dataopsamlingsenhed.
OS_RS232:	Varetager opgaverne UART test, Read RS232 og Write RS232. (_UARTTest, _ReadRs232, _WriteRs232).

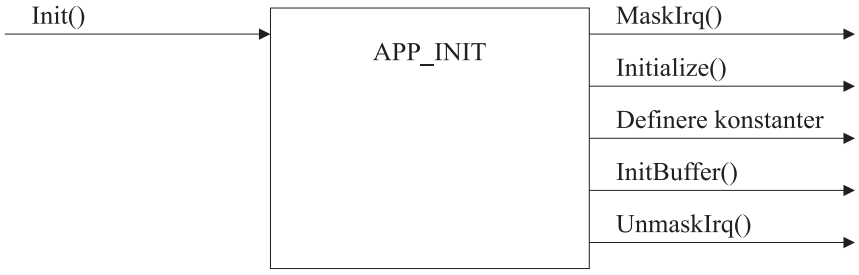
OS_BUFFER: Varetager opgaverne Init buffer, Read buffer og Write buffer. (_InitBuffer, _ReadBuffer, _WriteBuffer).

OS_IRQROUTINE: Varetager opgaven IRQ routine. (IrqRoutine).

Der kan ikke opstilles et statediagram for operativsystemet, da det består af en række uafhængige rutiner, som kaldes fra applikationen.
Det følgende vil bestå af et design af de enkelte moduler, startende med applikationen og efterfulgt af operativsystemet.

6.3 APP_INIT

Modulet skal initialisere de globale variable, User stack, vektortabel og VMEbuffer.



Figur 6.4 Input- Outputrelationen for modulet.

Input

Init( )	Kaldes fra Main, for at initialisere programmet.

Output

MaskIrq( )	Masker IRQ ud, mens der initialiseres.
Initialize( )	Initialiserer User stack og lægger adressen på IRQ-rutinen i vektor-tabellen.
Definere konstanter	Initialiserer de globale variable i applikation.
InitBuffer( )	Initialiserer VMEbuffer, dvs. ReadPointer og WritePointer.
UnMaskIrq( )	Masker IRQ ind, så IRQ'en kan slippe igennem.

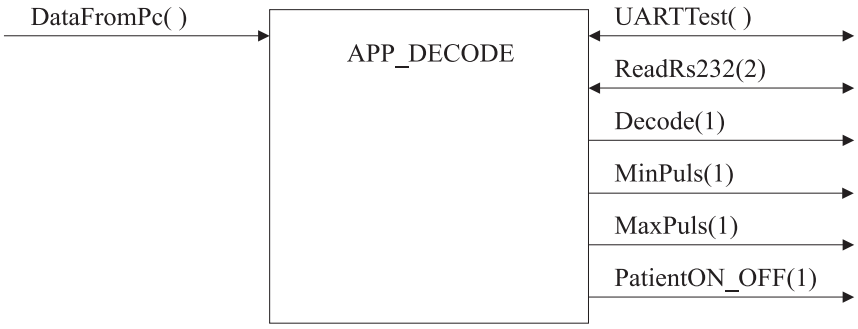
6.3.1 Design

Modulet designes således at IRQ'en først maskes ud mens der initialiseres. Derefter skal Initialize( ) kaldes i operativsystemet, som initialiserer User stack og vektortabellen. De globale variable skal dernæst sættes til de ønskede værdier. MinPuls sættes pr. default til 0 og MaxPuls til 255. Efter dette skal VMEbufferen initialiseres. Dette sker ved at kalde funktionen InitBuffer( ) i operativsystemet, som initialiserer ReadPointer og WritePointer. Til sidst maskes IRQ'en ind igen.

6.4 APP_DECODE

Modulet skal teste om der er data i UART, og hvis dette er tilfældet, skal den teste om det overholder protokollen. Derefter skal den decode data jfr. protokollen og undersøge, om det er en Min. puls, Max. puls eller Patient On/Off, der er sendt fra PC. Det ønskede kald skal så

udføres.



Figur 6.5 Input- Outputrelationen for modulet.

Input

DataFromPc( )	Kaldes fra Main, for at undersøge om der er kommet data fra PC.

Output

UARTTest( )	Sender 1 hvis der er data i UART. Parameteren overføres gennem dataregister 0.
ReadRs232(2)	Kaldes med to parametre: længden og en pointer til datastrengen.
Decode(1)	Dekoder datastrengen og aktiverer en af nedenstående funktioner.
MinPuls(1)	Sætter MinPuls til værdien, der er modtaget fra PC.
MaxPuls(1)	Sætter MaxPuls til værdien, der er modtaget fra PC.
PatientON_OFF(1)	Aktiverer eller deaktiverer en patient.

6.4.1 Design

Modulet designes således, at funktionen DataFromPC bliver kaldt fra APP_MAIN. I DataFromPC testes om der er data i UART'en ved hjælp af funktionen UARTTest. Denne returnerer et "1" hvis der er data, og et "0" hvis der ikke er data. Strengen indlæses ved hjælp af funktionen ReadRS232. Derefter bliver strengen undersøgt af TestData for at undersøge om protokollen er overholdt. Er protokollen overholdt, dekodes strengen.
Hvis den anden plads er et:

"N" sættes MinPuls vha. funktionen MinPuls.
"X" sættes MaxPuls vha. funktionen MaxPuls.
"V" sættes patient til og fra vha. funktionen PatientON_OFF.

6.5 APP_ENCODE

Dette modul skal encode de data, som skal videresendes til PC-programmet.

Figur 6.6 Input- Outputrelationen for modulet.

Input

Encode(2)	Kaldes med to parametre; en kommando (enten A eller P), samt en værdi.
EncodeEKG(2)	Kaldes med to parametre; en kommando "E" samt en pointer til et array med ti samples.

Output

WriteRS232(2)	Kaldes med to parametre; længden på arrayet samt en pointer til dette array. Arrayet indeholder de data, som skal sendes.

6.5.1 Design

Modulet designes således, at Encode() bliver kaldt fra modulet APP_PULSCALC, med to parametre. En kommando "A" hvis en alarm skal sendes, "P" hvis en puls skal sendes. Den anden parameter har værdien "1", hvis alarmen skal startes og "0", hvis den skal slukkes. Hvis det er en puls, der skal sendes, er det ASCII-værdien af pulsen, der bliver sendt. EncodeEKG(2) bliver kaldt fra modulet APP_EKGREDUC. Den første parameter er "E", og den næste er en pointer til et array med ti samples. Både Encode(2) og EncodeEKG(2) skal "pakke strengen ind" med først et "#", derefter data, og til sidst strengen "$\n\r\0".

6.6 APP_EKGREDUC

Da EKG-signalet, som skal vises på PC, ikke skal vises med samme opløsning, som det er samplet med, er det unødvendigt at overføre hele signalet til PC. Derfor skal EKG-signalet reduceres, således at det kun er de signaler, som skal bruges af PC'en, der overføres.

Input

EKGReduction	Funktionen EKGReduction kaldes, indeholdende det fulde EKG-signal.

Ouput

EncodeEKG	Encoder det reducerede EKG-signal, så det er klar til at blive sendt til PC.

6.6.1 Design

Ved EKG reduktion, skal kun nogle af EKG-samplingerne, sendes videre til PC. Antallet af samplinger, som skal sendes videre, beregnes ved at finde antallet af punkter, der skal plottes på skærmen pr. sekund (pixelfrekvensen):

(6.1)

$P_{skærmbredde}$ = Skærmbredden i pixels.
$t_{visning}$ = Tidsrummet som hver pixel skal blive på skærmen, før den overskrives.

Efter pixelfrekvensen er beregnet, kan det beregnes, hvor ofte der skal sendes et EKG-signal til PC:

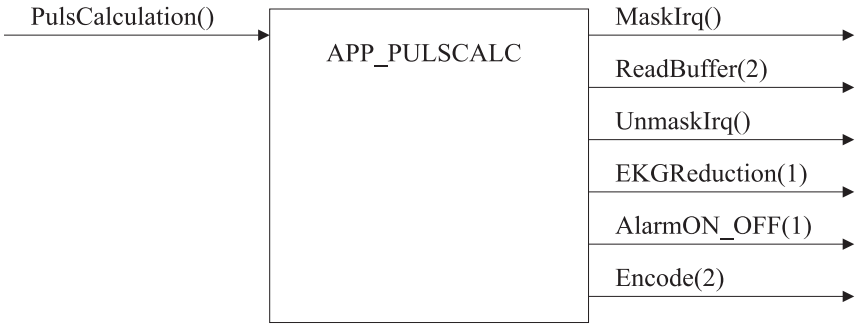
(6.2)

f_{EKG} = Samplefrekvensen for EKG-signalet.

Dette vil sige, at hvert ottende sample skal sendes videre til PC. Da RS232C-protokollen overfører 10 EKG-signaler af gangen, skal modulet først sende signalerne, når der er opsamlet 10 samples.

6.7 APP_PULSCALC

Dette modul skal beregne pulsen for den valgte patient, samt give alarm, hvis pulsen bliver større eller mindre end den indstillede max.- og min. puls.



Figur 6.8 Input- Outputrelationen for modulet.

Input

PulsCalculation( ) Bliver kaldt fra Main.

Output

MaskIRQ( )	Masker IRQ ud, mens der læses i bufferen.
ReadBuffer(2)	Læser i bufferen. Har to parametre; VmeDataIn og Buffer-UnderRun.
UnMaskIrq( )	Udmasker IRQ efter der er læst i bufferen.
EKGReduction(1)	Sender VmeDataIn videre til EKGReduction.
AlarmON_OFF(1)	Starter og stopper alarmen.
Encode(2)	Encode kaldes med to parametre, som starter/stopper alarmen eller indeholder pulsen.

6.7.1 Design

Modulet designes således, at IRQ maskes ud, så længe der læses i bufferen, hvorefter den maskes ind igen.
ReadBuffer indlæser et sample fra VMEbuffer og lægger det ind i et array, der har størrelsen 10. Dette sample sendes videre til et array i EKGReduction.
Når dette array er fyldt, udregnes en gennemsnitshældning på de 10 samples. Hvis hældningen (Diff) er større end 2 for 10 samples, registreres dette som en puls.
Denne beregning er foretaget på baggrund af kendskabet til R-takkens varighed.
En R-tak varer typisk 120 ms, hvorfor det antages, at peaken nås efter 60 ms. Da der samples med 500 Hz, udtages der ca. 30 samples på den opadgående flanke. EKG-signalet har værdien 127, hvis ingen patienter er tilsluttet, og derfor regnes en R-tak for mindst at nå værdien 200. Hældningen på mindst 2 er bestemt ud fra disse kriterier:

(6.3)

2,43 er gennemsnitshældningen på den opadgående flanke af R-takken, men denne afrundes til 2. Dette betyder at kortere samt lavere R-takker også registreres.

6.8 APP_MAIN

Dette modul styrer hvilke hovedfunktioner der kaldes. Det første som kaldes er initialiseringsfunktionen, som sørger for, at alle parametre indstilles. Herefter startes en uendelig løkke, der sørger for, at mikroprocessoren kontrollerer for IRQ-signaler. Endvidere køres funktionerne DataFromPC() og PulsCalculation().

Output

Init	Initialiserer mikroprocessorkortet samt de globale variable ved opstart.
DataFromPC	Undersøger om der er kommet data fra PC.
PulsCalcuation	Beregner pulsen vha. R-tak.

6.8.1 Design

I Main startes der med at kalde initialiseringsrutinen, som sørger for at alle parametre er i orden. Herefter startes en uendelig løkke, som kalder de funktioner, mikroprocessorkortet skal udføre. Først kaldes funktionen DataFromPC(), der undersøger om der er data fra PC. Dernæst kaldes funktionen PulsCalculation(), som beregner pulsen og kalder EKGReduction() samt øvrige funktioner, som det er beskrevet i modulet APP_PULSCALC. Når dette er udført, startes der forfra i løkken.
Det er væsentligt, at processen i mainløkken generelt ikke tager længere tid, end tiden der går mellem to IRQ. Hvis dette ikke var tilfældet, ville samples ophobe sig i bufferen.

6.9 OS_INITIALIZE

Ved opstart skal operativsystemet initialiseres, således at alle registre, så som stack pointer og kontrolregister, har de ønskede værdier.

Input

Initialize()	Funktion som initialiserer operativsystemet.

6.9.1 Design

Ved initialisering af operativsystemet sættes StackPointer og vektor til IRQ-rutine til de værdier, som er angivet i hardwaredefinitionen. Herefter skal registeret til BufferError nulstilles.

6.10 OS_MASKIRQ

Ved initialisering og læsning fra EKG-bufferen, er det ikke hensigtsmæssigt at modtage en IRQ. Derfor skal udmaskning af IRQ være muligt.

Input

MaskIrq()	Masker IRQ ud.
UnmaskIrq()	Fjerner udmaskning.

6.10.1 Design

Statusregisteret på mikroprocessoren styrer hvilket IRQ-nummer, der bliver godkendt af mikroprocessoren.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T		S			I_2	I_1	I_0				X	N	Z	V	C

Statusregisteret på mikroprocessoren.[Clements, 1997, 24]

Som det ses i tabellen er det bit 8 til bit 10 som styrer hvilke IRQ-numre, der bliver udmasket og bit 13 styrer om mikroprocessoren er i supervisor- eller user mode.
Ved "MaskIrq" skal bit 13 sættes høj, bit 8 til bit 10 skal have værdien 7, og resten skal ikke ændres. For ikke at ændre resten af statusregisteret skal der derfor udføres en OR-funktion med 2700h.
Ved "UnmaskIrq" skal der udføres en AND-funktion med 20FFh, hvilket medfører at alle interrupt levels accepteres.

6.11 OS_DAOE

Modulet skal kommunikere fra mikroprocessoren til dataopsamlingsenheden. De data, som skal sendes til denne enhed, er Alarm On/Off og patient til- og fra kobling.

Input

StartAlarm	Starter alarmen på dataopsamlingsenheden.
StopAlarm	Stopper alarmen.
SetPatient	Til- og frakobling af patienter.

Output

Kontrolregisteret på dataopsamlingsenheden.

6.11.1 Design

Kontrolregisteret på dataopsamlingsenheden styrer, hvilke patienter der er tilkoblet, og hvilke patienter der evt. er alarm ved.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0
Alarmregister								Patientregister							

Kontrolregister på dataopsamlingsenheden.

Da dataopsamlingsenheden er designet til at behandle otte patienter, og mikroprocessoren kun skal designes til at behandle een patient, skal kun patient 0 aktiveres. Dette medfører, at bit 0

skal sættes for at aktivere patienten, hvilket svarer til xx01h. For at starte alarmen på patient 0, skal bit 8 sættes, hvilket svarer 01xxh. Tilsvarende skal hhv. bit 0 og bit 8 slukkes, når patienten skal frakobles og alarmen stoppes.

6.12 OS_RS232

Kommunikationen mellem mikroprocessor og PC skal foregå over en seriel RS232 forbindelse. Dette modul skal varetage kommunikationen mellem applikationen og hardwaren til denne forbindelse.

Input

WriteRS232	Skriver til RS232-driveren.

Output

ReadRS232	Læser fra RS232-driveren,

6.12.1 Design

For at kommunikere med RS232-driveren benyttes Trap #14, som er en indbygget funktion i PEPbug. Ved at kalde Trap #14 med kommandoen F5h, vil de data, som ligger mellem det adresseregister 5 og 6 peger på, blive sendt over til PC'en. Det samme gælder når Trap #14 kaldes med kommandoen F3h. Da vil de data, som bliver sendt fra PC'en blive lagret hvor adresseregister 6 peger.

6.13 OS_BUFFER

Formålet med dette modul er at læse, skrive og initialisere en ringbuffer til EKG-signaler fra dataopsamlingsenheden.

Input

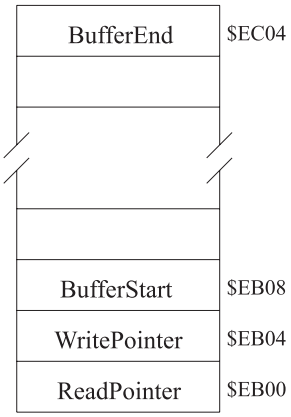
InitBuffer	Opretter EKGbufferens Read- og WritePointer
WriteBuffer	Skriver til bufferen og forøger værdien af WritePointer til næste skrivning.

Output

ReadBuffer	Læser fra bufferen og forøger værdien af ReadPointer til næste læsning.

6.13.1 Design

Ved opstart skal bufferen initialiseres, således at Read- og Write pointerne kommer til at pege på den første dataplads i bufferen. De to første pladser i bufferen benyttes til at lagre Read- og Writepointer.
Ved en læsecyklus skal det undersøges om Read- og WritePointer peger på det samme, hvis dette er tilfældet er bufferen tom. Er bufferen ikke tom skal dataen læses og derefter skal ReadPointeren rykkes en plads frem. Skrivecyklusen skriver først data i bufferen og rykker så WritePointeren frem til næste skrivning. Efter at pointeren er flyttet, undersøges det om Read- og WritePointerne peger på det samme, og er dette tilfældet, betragtes bufferen for at være fuld.
Da bufferen er en ringbuffer, skal det undersøges, om de to pointere peger på toppen af bufferen, og hvis dette er tilfældet, skal de flyttes tilbage til starten af bufferen.



Figur 6.15 EKG-bufferens placering i RAM.

6.14 OS_IRQROUTINE

Ved modtagelse af en IRQ fra dataopsamlingsenheden, skal mikroprocessoren hente data fra statusregisteret og lægge det i en ringbuffer.

Input

IRQ	Afgives fra dataopsamlingsenheden når en AD-konvertering er færdig.

Output

WriteBuffer(1)	Skriver til ringbufferen.

6.14.1 Design

Data i statusregister og ADC på dataopsamlingsenheden læses, hvorefter WriteBuffer kaldes med de hentede data. Statusregistret på dataopsamlingsenheden er opbygget som vist nedefor.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	P_2	P_1	P_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Patientregister								Dataregister							

Statusregister på dataopsamlingsenheden.

6.15 Modulintegration og procestest

De enkelte moduler er testet løbende, så vidt det var muligt at afvikle dem selvstændigt.
Den eneste mulighed for at teste mikroprocessorprogrammet (processen) ved integrationen, var ved metoden “big bang”. Da modulerne var meget afhængige af hinanden, var det ikke muligt at foretage en trinvis integration.
Procestesten foregik ved at køre hele programmet igennem i trace mode og kontrollere på PC-skærmen at programmet gjorde som forventet.

7 PC-program

Denne proces omfatter det program der skal køre på PC'en. Det skal præsentere de data som mikroprocessoren beregner og sender til PC'en, samt sende kommandoer fra brugeren til mikroprocessoren.
Designet af dette program er baseret på proceduren for procesdesign i "Håndbog I Struktureret Program Udvikling" [SPU, 1998].

Programmet er primært designet ved hjælp af metoden "funktionsorienteret top-down".
I kravsspecifikationen er der stillet en række krav til dette program. Disse krav samt tekniske omstændigheder har resulteret i den følgende liste af opgaver, som programmet skal varetage:

- Starte / stoppe program.
- Vise EKG på skærm.
- Vise pulstrend på skærm.
- Skrive aktuel puls på skærm.
- Gemme puls på disk een gang pr. minut i op til 24 timer.
- Vise alarm hvis grænser overskrides.
- Afgive lyd ved alarm.
- Give brugeren mulighed for indtastning af følgende:
    - Patientens navn.
    - Patientens CPR.nr.
    - Max. puls.
    - Min. puls.
- Sende alarmgrænser til mikroprocessorkort.
- Vise testsignal.
- Holde signal.
- Tilslutte / frakoble patient.
- Give mikroprocessorkortet besked om tilslutning / frakobling af patienter.
- Modtage data fra mikroprocessorkortet.
- Tolke data fra mikroprocessorkortet.
- Styre kommunikation med RS232.
- Starte / stoppe RS232.

- Starte / stoppe grafikdrivere.
- Åbne / lukke datafiler til trendlog og testsignal.

For at give overblik over disse opgaver, samles de i grupper, der organiseres i et hierarki som vist på nedenstående figur:

7.1 Opdeling i moduler

Diagrammet på figur 7.1 resulterer i følgende opdeling i og navngivning af moduler:

Programmet starter i modulet Main, hvorefter modulet Init kaldes. Dette modul skal initialisere grafik, RS232 samt klargøre filer til læsning og skrivning. Derefter går programmet ind i en løkke, som checker for tastaturtryk og data fra mikrocomputeren. Hvis een af disse ting forekommer, kaldes hhv. UserInput eller ReadRS232 til at håndtere de indkommende data.
Modulet ReadRS232 kontrollerer, om der er kommet data fra mikrocomputeren, og hvis dette er tilfældet, dekodes data og sendes til det rette modul. Hvis det er EKG-samples, der er modtaget, kaldes modulet EkgPlot, som optegner signalet på skærmen. Tilsvarende kaldes PulsPlot eller Alarm, hvis det er henholdsvis Puls og Alarm der modtages.
Hvis brugeren trykker på en taste, kaldes modulet UserInput, som giver brugeren mulighed for

indtastning af Navn osv. Dette modul skal endvidere sørge for at sende data til mikrocomputeren, således at denne ved, hvorledes eksempelvis alarmgrænserne er.
Når programmet er i modulet UserInput, skal det, for at undgå at optegningen af EKG-signalet går i stå, hele tiden kontrollere, om der er kommet data fra mikrocomputeren. Dette medfører at modulet ReadRS232 kaldes fra modulet UserInput, og hvis der er modtaget data, kaldes modulerne som beskrevet tidligere. Trykker brugeren på tasten 'q', skal programmet afsluttes.
Efter denne beskrivelse kan der opstilles en oversigt over hvilke moduler, der kalder hinanden.
Oversigten ses på figur 7.3.
Modulerne skal opbygges således, at deres features aktiveres ved at kalde en funktion med samme navn som modulet.
Det er nu muligt at designe de enkelte moduler, samt at fastlægge deres grænseflader. De følgende afsnit indeholder moduldesignet af disse. Punktet "implementering" er udeladt, da det består af programkoden, som er placeret i appendiks F. Der er kommentarer i koden, som forklarer hvordan de enkelte funktioner fungerer.

- Main
    - Init
        - (RS232)
    - ReadRS232
        - PlotPuls
        - PlotEkg
        - Alarm
        - (RS232)
    - UserInput
        - ReadRS232
            - PlotPuls
            - PlotEkg
            - Alarm
            - (RS232)

Figur 7.3 Oversigt over hvilke moduler, der kalder hvilke.

7.2 Main

Dette modul skal være det første, der indlæses når programmet startes, og skal således kalde de underliggende funktioner efter behov.

Input
Forskellige tastetryk som sendes videre til modulet UserInput.
Hvis der trykkes på tasten "q", skal modulet afbrydes, hvorved programmet stoppes.

Output

Init(1)	Udfører initialiseringsrutiner.
Init(0)	Udfører afslutningsrutiner.
UserInput("Tastetryk")	Udfører inputrutiner afhængig af tastetryk.
ReadRS232()	Kontrollerer om der er data fra RS232.

7.2.1 Design
Modulet designes således, at det, ved programopstart, kalder Init med værdien "1".
Herefter skal modulet starte en løkke, som kalder ReadRS232 og, hvis der trykkes på en taste kaldes UserInput. Hvis der trykkes på tasten "q", skal modulet kalde funktionen Init med værdien "0" og afslutte.

7.3 Init

Dette modul kaldes som det første, når programmet startes, og som det sidste, når programmet afsluttes. Det har til opgave at initialisere forskellige ting, som er nødvendige for resten af programmet, samt at afbryde dem når programmet afsluttes. Ved opstart skal grundskærmen optegnes, dvs. de faste rammer og tekst.

Input
Modulets features aktiveres ved at kalde funktionen Init() med følgende værdier:

Init(1)	Et funktionskald med denne værdi angiver, at der skal initialiseres.
Init(0)	Angiver at de initialiserede funktioner skal afbrydes.

Output
Grundskærmen optegnes.

7.3.1 Design

Modulet opbygges af en række funktioner, som kaldes fra modulets hovedfunktion Init. Disse funktioner er som følger:

DrawBase
Denne funktion optegner grundskærmen ved opstart. (Se fig. 7.6).

FadeOut
Skal få skærmbilledet til at fade ud. Dette gøres for at vise brugeren, at programmet er afsluttet på korrekt vis, og ikke afbrudt pga. en programfejl.

7.4 UserInput

Dette modul har til formål at fortolke tastetryk fra brugeren, og derefter udføre den tilhørende funktion, hvilket kan ske internt i modulet eller ved at kalde andre moduler.

Input
Input til modulet er tastetryk, der er relateret til funktioner, som beskrevet under output.

Output
Følgende tastetryk som input resulterer i det beskrevne output:

- n Aktiverer en prompt, hvorved brugeren kan indtaste navn, som derefter vises på skærmen.
- c Aktiverer en prompt, hvorved brugeren kan indtaste CPR.nr., som derefter vises på skærmen.
- i Aktiverer en prompt, hvorved brugeren kan indtaste min. puls, som derefter vises på skærmen og videresendes til mikroprocessorkortet.
- a Aktiverer en prompt, hvorved brugeren kan indtaste max. puls, som derefter vises på skærmen og videresendes til mikroprocessorkortet.
- t Skifter mellem visning af det rigtige signal fra mikroprocessoren og et testsignal fra PC'en.
- h Fryser signalet. Virker som toggle.
- 1 Tilkobler patient nr. 1. Virker som toggle.

7.4.1 Design

Modulet opdeles således at der laves en funktion til hvert af de forskellige input:

InputName()	Indtastning af navn.
InputCpr()	Indtastning af CPR.nr.
SendFrame()	Varetager afsendelsen af frames til modulet RS232. Først skal framen formateres, hvorefter den sendes.
Min_Puls()	Indtastning af min. puls, som derefter sendes vha. funktionen SendFrame.
Max_puls()	Indtastning af max. puls, som derefter sendes vha. funktionen SendFrame.

Patient_On_Off()	Aktiverer/deaktiverer patient nr. 1 og skriver teksten ved denne med hvid/grå. Der sendes besked til mikroprocessoren vha. funktionen SendFrame.

Modulet bygges op omkring en funktion, som vælger mellem de ovennævnte funktioner afhængigt af, hvad der er indtastet.

7.5 RS232

Dette modul er en driver, der skal kommunikere mellem UART-chippen og softwaren.
Modulet består af en samling funktioner, der er i stand at kommunikere med UART-chippen.
Endvidere er der implementeret en ringbuffer i denne driver.
Modulet er programmeret af Jens F.D.Nielsen fra I8, Aalborg Universitet, men gruppen har foretaget nogle mindre kosmetiske ændringer.
Da modulet ikke er programmeret af gruppen, bliver der ikke tale om et moduldesign, men i stedet en modulanalyse.

Input

UART-chip	Sender serielle signaler til mikroprocessoren.
InitCom()	Initialiserer UART'ens portindstillinger.
ClearComInBuf()	Sletter ringbufferen i driveren.
WriteStrCom()	Skriver til UART.
StartCom()	Nulstiller og starter UART.
StopCom()	Stopper UART.

Output

ChInBuffer	Returnerer antallet af tegn i UART buffer.
CRInBuffer	Returnerer antallet af "\r" i UART buffer.
ReadCom()	Læser fra ringbufferen.

Modulet skal blot inkluderes i programmet, hvorved det er muligt at kalde de funktioner det indeholder.

7.6 ReadRS232

Dette modul skal kontrollere, om der er kommet data fra mikrocomputeren, og hvis dette er tilfældet, dekode data og kalde de respektive moduler.

Input

ChInBuffer()	Funktion der returnerer antallet af bytes i bufferen.
CRInBuffer()	Funktion der returnerer antallet af "\r" i bufferen.
ReadCom()	Funktion der læser en byte i bufferen.
Testsignal	Global variabel som angiver, om der skal plottes rigtigt signal eller testsignal.

Output

EkgPlot("Rigtigt signal")	Sender 10 rigtige sampels til modulet EkgPlot.
EkgPlot("Testsignal")	Sender 10 testsampels til modulet EkgPlot.
PulsPlot(Puls)	Sender en puls til modulet PulsPlot.
Alarm("værdi")	Kalder modulet Alarm med "værdi".

7.6.1 Design

Modulet gør brug af nogle funktioner, som ligger i modulet RS232.
Modulet designes således, at det kontrollerer, om der er en "\r" i bufferen, idet dette betyder, at der er modtaget en frame. Funktionen CRInBuffer() benyttes til at undersøge om dette er tilfældet. Herefter læses fra ringbufferen i COM-portdriveren, ved hjælp af funktionen ReadCom(), indtil der findes en "#". Efter dette læses der bytes ind i et array indtil "\n\r" eller indtil der ikke er flere bytes i bufferen. Antallet af bytes i bufferen fås fra funktionen ChInBuffer(). Der skal læses indtil "\n\r" findes, da alle frames afsluttes med disse karakterer. Arrayet dekodes og en af funktionerne EkgPlot, PulsPlot og Alarm kaldes. Valget af funktion er afhængigt af framens indhold som beskrevet i afsnit 3.4.3.

7.7 EkgPlot

Dette modul har til opgave at plotte EKG'et på skærmen, efterhånden som data kommer fra ReadRS232. Desuden skal der være mulighed for at udskrive et testsignal, som ligger på harddisken.

Input

EkgPlot("10 samples")	Funktionen, som dette modul består af, kaldes med en array, som indeholder 10 samples. Disse samples kan enten komme fra det målte signal eller fra testsignalet.
Hold	Global variabel som angiver, om skærmbilledet skal fryses.
Testsignal	Global variabel som angiver, om der er tale om test eller rigtigt signal.

Output

Grafen optegnes på skærmen. Hvis der er tale om det rigtige signal, er grafen grøn, og hvis der er tale om testsignalet, er grafen gul.

7.7.1 Design

Der laves en løkke, som gennemløbes een gang for hvert sample i den pågældende data-array. Her udvælges det pågældende sample, som plottes på skærmen. For at undgå, at grafen bliver en række prikker, trækkes en linie mellem det nye punkt og det foregående.

7.8 PulsPlot

Dette modul har til opgave at skrive den aktuelle pulsværdi samt at plotte pulstrenden på skærmen, efterhånden som data kommer fra ReadRS232. Pulstrenden skal desuden gemmes på harddisken i en tekstfil.

Input

PlotPuls("Aktuel Puls")	Funktionen kaldes med en variabel, som angiver den aktuelle puls.
TimeNow	En struct som angiver det aktuelle tidspunkt.

Output

Grafen optegnes på skærmen og pulsen skrives som tal i det dertil beregnede felt. Desuden gemmes den aktuelle pulsværdi een gang pr. minut i en tekstfil kaldet "trend.log".

7.8.1 Design

Modulet består af een funktion, og denne opdeles i tre dele:

Skriv puls:
Skriver den aktuelle puls på skærmen.

Plot trendgraf:
Tilføjer et punkt på trendgrafen. Denne graf vises som punkter, da den ville blive svær at aflæse hvis punkterne blev forbundet.

Gem trend-log:
Den aktuelle puls gemmes i filen "Trend.log" een gang hvert minut.

7.9 Alarm

Dette modul har til opgave at generere det alarmsignal PC'en skal udsende. Dette gøres ved at ændre farven på rammen omkring EKG-plottet, og give en lyd vha. PC-speakeren.

Input

Alarm(1)	Hvis funktionen kaldes med denne værdi, bliver alarmen på PC'en aktiveret.
Alarm(0)	Denne værdi deaktiverer alarmen.

Output
Farven på rammen ændres afhængigt af om alarmen aktiveres eller deaktiveres:

Aktiver:	Rammen bliver rød.
Deaktiver:	Rammen bliver hvid igen.

Der afgives en lyd vha. PC-speakeren, når alarmen er aktiv.

7.9.1 Design

Modulet består kun af funktionen Alarm, hvori der laves en IF-sætning, som afhænger af, hvad funktionen blev kaldt med.

7.10 Modul integration og procestest

Programmet er testet ved "top-down"-metoden. Dvs. at modulet Main blev kompileret og testet først. Når modulerne var kompileret uden fejl, blev de tilføjet et efter et til Main-programmet. De var dog modificeret således, at de ikke kaldte underliggende funktioner, som ikke var integreret endnu. Modulerne blev tilføjet i nedenstående rækkefølge:

- Main
- Init
- RS232
- UserInput
- ReadRS232
- Alarm
- PlotEkg
- PlotPuls

For at kontrollere kommunikationen mellem mikrocomputeren og PC'en, blev PC'en forbundet til en test-PC, som kørte et testprogram. Dette testprogram sendte de definerede frames med EKG-samples, pulser og alarmbeskeder til PC'en. Desuden viste det de signaler, der kom fra PC'en på test-PC'ens skærm.
Det kunne konstateres, at EKG-programmet virkede efter hensigten, når det modtog de korrekte frames. Kildekoden til testprogrammet kan ses i appendiks F.

8 Accepttest og konklusion

Dette kapitel indledes med en diskussion, hvor de enkelte processer behandles. Dernæst beskrives den afsluttende accepttest af systemet, hvilken leder frem til den endelige konklusion på projektet. Accepttesten er foretaget for at bestemme, om den færdige prototype opfylder de, i kravsspecifikationen, opstillede krav.

8.1 Diskussion

Her vil den konstruerede prototype gennemgås, og der vil blive diskuteret, hvor vellykket de enkelte processer er implementeret. Processerne vil blive gennemgået enkeltvis, startende med instrumentforstærkeren. Det er ikke systemet som en helhed der diskuteres, men snarere hvordan designet af processerne er forløbet.

8.1.1 Instrumentforstærker

Det viste sig, at den benyttede instrumentforstærker, ikke levede helt op til forventningerne. Dette betød, at der var meget støj på udgangsignalet fra denne forstærker. Projektgruppen valgte dog at nedpriotere løsningen af dette problem, idet konstruktionen af en instrumentforstærker ikke er det væsentligste for pensum på dette semester.

8.1.2 Dataopsamlingsenhed

Denne del, som udgør den konstruerede hardware, i dette projekt viste sig at fungere efter hensigten. Der var imidlertid nogle ting, som kunne have været lavet på en bedre måde. Eksempelvis ville det være en fordel, at kunne aflæse kontrolregistret på dataopsamlingsenheden. Derved ville det være muligt, at aflæse hvilken tilstand enheden er i. Dette er dog implementeret softwaremæssigt, ved at et register på mikroprocessorkortet indholder de data, som skrives til dataopsamlingsenheden. Registeret vil dog ikke indholde de rigtige data, hvis der sker en fejl i overførslen til dataopsamlingsenheden.
Endvidere er der benyttet nogle kredse i interfacet, som ikke kunne levere så meget strøm, som VME-standarden foreskriver. Dette gav dog ikke nogle problemer. Der kunne istedet benyttes kredsen 74LS645, som tranceiver og kredsen 74LS541, som henholdsvis driver og receiver. Derved ville problemet med hensyn til VME-standarden være løst. Envidere kan det nævnes, at disse kredse er pinkompatible med de benyttede kredse, hvorved denne udskiftning kan foretages uden at "wrappe" om.
Dataopsamlingsenheden er implementeret på således at den opfylder kravene, men der er stadig dele der kunne forbedres.

8.1.3 Mikroprocessorprogram

Der er i dette program benyttet to forskellige programmeringssprog; C og assembler. Applikationen, som blev skrevet i C, kunne muligvis optimeres, hvis den var skrevet i

assembler. Der er i den forbindelse bemærket, at programmer som er lavet vha. C-kompileren laver nolge unødigt komplicerede operationer, som kunne være undgået, hvis koden var skrevet direkte i assembler. For eksempel laves en hel række "jump"-funktioner, hver gang der skal laves eet "jump". At skrive programmet i assembler ville dog kræve mange arbejdstimer i forhold til den optimering, man ville opnå.
Operativsystemet, som er skrevet i assembler, kunne formentlig optimeres ved at benytte alternative instruktioner i visse tilfælde. Projektgruppen valgte dog ikke at foretage disse ændringer, idet programmerne virkede efter forventningerne, og der ikke var pladsproblemer.

8.1.4 PC-program

Til dette program blev der benyttet programeringssproget C++. Dette sprog blev benyttet, idet det giver mulighed for at vise grafik. Programmet blev lavet i Borland C++, som indeholder et grafikbibliotek, der giver mulighed for at lave dosgrafik. Denne dosgrafik blev benyttet, da den levede op til de stillede krav, og samtidig var forholdvis simpel at bruge.

8.2 Accepttest

I dette afsnit konkluderes på det samlede system, for samtidig at undersøge hvorvidt de, i kravsspecifikationen, stillede krav, er opfyldt.
Da der, som nævnt i kravsspecifikationen, kun er tale om en prototype, vil kun de krav, som stilles til denne, blive testet.

8.2.1 Funktionelle krav

Først undersøges det om prototypen lever op til de funktionelle krav til de tre funktioner, EKG-plot, pulsmåling og pulstrend. Testen foregår ved at starte systemet op og tilslutte en patient, for derefter at gå listen af funktionelle krav igennem punkt for punkt.

EKG-plot

- ✓ Der vises een persons EKG på skærmen ad gangen.
- ✓ EKG-plottet kan fryses.
- ✓ Der vises ca. ti sekunders billede på skærmen.
- ✓ På en 14" skærm er grafen er af en sådan størrelse, at det er muligt af aflæse den i 1 meters afstand.
- ✗ På grund af støj i instrumentforstærkeren, var signalet af en sådan kvalitet, at det var vanskeligt at identificere de enkelte faser. Ved at tilslutte en tonegenerator direkte til ADC'ens indgang, kunne det konstateres, at problemet lå i instrumentforstærkeren, da signalet i dette tilfælde stod tydeligt på skærmen.
- ✓ Bortset fra rammen, som fylder een pixel i hver side af skærmen, blev hele skærmens bredde udnyttet til grafen.

Pulsmåling

- ✓ Pulsen bliver skrevet som heltalsværdi.
- ✓ Pulsen opdateres mindst hvert fjerde sekund, da det tolkes som en alarm hvis der går mere end 2000 samples mellem to pulsslag.
- ✓ Det er muligt at indstille min./max. pulser for alarmen.
- ✓ Både dataopsamlingsenheden og PC'en giver visuel og auditiv alarm hvis grænseværdierne for pulsen overskrides.
- ✗ Da der er tale om en prototype, som kun kan behandle data fra een patient, bliver alle tilsluttede patienters puls ikke vist på skærmen.

Pulstrend

✓ Der bliver lagret op til 24 timers pulstrend på PC'ens harddisk, hvorefter værdierne i trendfilen bliver overskrevet fra starten af med nye værdier.
✓ Bortset fra rammen, som fylder een pixel i hver side af skærmen, blev hele skærmens bredde udnyttet til grafen.
✓ Både pulstrenden og patientens EKG vises på skærmen samtidigt.

Alarm

✓ Både PC og dataopsamlingsenhed afgiver visuel og auditiv alarm, hvis pulsgrænserne overskrides.
✓ På dataopsamlingsenheden markerer en lysdiode, for hver patient, eventuelle alarmer.
✗ I PC-programmet markeres det ikke hvilken patient en alarm stammer fra, da det blot er rammen omkring EKG-plottet der skifter farve. En sådan indikering kunne dog let tilføjes, ved at ændre farven på rammen omkring den pågældende patients navn.

8.2.2 Kvalitetsfaktorer

Kontrol af kvaliteter		
Egenskab	Prioritering	Accept
Pålidelighed	Særdeles vigtig. Da systemet kan fortsætte uden kommunikation med PC, hvis min. og max. puls er indtastet, lever det op til dette krav.	✓
Vedligeholdelsesvenlighed	Vigtig / Meget vigtig. Da der er tale om en prototype, er apparatet ikke monteret i en kasse, og derfor er det ikke nemt at rengøre.	✗
Udvidelsesvenlighed	Vigtig. Det er relativt simpelt at opdatere softwaren på PC og mikroprocessorkort. Hvis systemet skal udvides til at kunne behandle data fra alle otte patienter, er mikroprocessorkortet den eneste hardware, som skal udskiftes. Der kræves blot, at den nye hardware er kompatibel med grænsefladerne til dataopsamlingsenhed og PC.	✓
Brugervenlighed	Meget vigtig. Da PC-programmet er meget simpelt, og da genvejene er vist på skærmen, er denne del af brugerfladen brugervenlig. Da hardwaren ikke er monteret i en kasse, lever den ikke op til kravet om brugervenlighed. Dette vil dog ikke være noget problem for det færdige apparat.	✓

Genbrugbarhed	Vigtig. Da PC-programmet er forberedt for udvidelse til otte patienter, vil store dele af dette kunne genbruges. Dataopsamlingsenheden er fuldt forberedt for otte patienter, og vil derfor kunne genbruges i sin helhed. Det samme gælder instrumentforstærkeren, da denne blot skulle produceres i otte eksemplarer.	✓
Integritet	Vigtig / Meget vigtig. PC og mikrocomputer skal kunne fortsætte ved strømsvigt. Dette er dog ikke et krav til selve systemet, men til dets strømforsyning.	✓
Effektivitet	Ikke særlig vigtig. Apparatet er ikke i stand til at monitorere otte patienter, men da dette ikke er et krav til prototypen, leves der op til kravene til denne.	✓

Som det fremgår af skemaet, lever det fremstillede system i rimelig grad op til kravene til prototypen. Der er dog stadig en række krav til det færdige system, som ikke er opfyldt. Da der har lagt en begrænsning i ydeevnen for den udleverede hardware, er dette acceptabelt.

8.3 Konklusion

Den fremstillede EKG-monitor er ikke ment som nogen revolutionerende nyskabelse indenfor EKG-monitorering, men som en øvelse i design af hardware samt software.
Som det fremgår af accepttesten, lever systemet op til langt de fleste funktionelle krav. Der er dog problemer med at leve op til kravene om at kunne monitorere otte patienter. Da det i kravsspecifikationen blev besluttet, at prototypen kun skulle kunne monitorere een patient, skyldtes det begrænsninger i den udleverede hardware. Det var altså omstændigheder, som var uden for projektgruppens indflydelse, der gav disse problemer.
Et andet stort problem i systemet var instrumentforstærkeren, som indtroducerede så meget støj, at EKG-signalet var svært at genkende på skærmen. Dette anses dog ikke for at være nogen stor svaghed for systemet, da det er meget simpelt at udskifte denne forstærker; der kræves blot en bedre forstærker, som lever op til grænsefladekravene.
Når det gælder kvalitetsfaktorerne, er der flere steder, hvor systemet ikke opfylder kravene til fulde. Dette skyldes dels, at der har været tale om et uddannelsesforløb, og dels de begrænsede ressourcer, der har været til rådighed; det havde for eksempel været relativt simpelt at montere mikroprocessorkortet i en kasse, hvorved kravet om vedligeholdelsesvenlighed var blevet opfyldt.
Da det er lykkedes at fremstille et system, som i stor udstrækning lever op til kravsspecifikationen og formålserklæringen, må det konkluderes, at projektet som en helhed har været en succes.

9 Litteraturliste

[Forkortelse, År, (Side)] Titel, udgivelses år, forfatter navn, forlag, udgave nr., ISBN

[Backer, 1995] Politikens store Lægebog, 1995, Paul Backer, Politikens Forlag]

[Clements, 1997] Microprocessor Systems Design, 1997 Third Edition, Alan Clements, PWS Publishing Company, 0-534-9822-7

[Dremstrup, 2000] Forelæsning af Kim Dremstrup Nielsen i foråret år 2000

[Gøtzsche, 1967] Elektrocardiografisk atlas, 1967, Henning Gøtzsche, F.A.D.L.s Forlag.

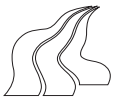
[Lange, 1997] Vejledning i EKG mønstre, 1997, Christian Lange, Medicotest, 87-984226-3-4

[Lunau, 1975] Fysiologisk Atlas, 1975, I. Lunau og P. Lund Mathiasen, Medicinsk Forlag.

[Mathiasen, 1970] Fysiologisk Atlas - en studiehåndbog, 1970, Lunau og Lund Mathiasen, Medicisk Forlag A/S, 1. oplag

[SPU, 1998] Håndbog i Struktureret Programudvikling, 1998, Stephen Biering-Sørensen, Finn Overgaard Hansen, Susanne Klim og Preben Thalund Madsen, Teknisk Forlag A/S, 1. udg. 8. oplag, 87-571-1046-8

[VMEbus, 1982] VMEbus Specification Manual, 1982, VMEbus Manufacturers Group]



Appendiks

A ABEL-programmer

I dette appendiks er der en beskrivelse af PLD'er og programmeringssproget ABEL. Endvidere er programkoderne til PEEL kredsene, i de forskellige moduler på dataopsamlingsenheden, angivet. Overskrifterne A3 -A8 beskriver i hvilket de enkelte PEEL-kredse er placeret.

A.1 Generelt om PLD

PLD (Programmable Logic Device) er et effektivt værktøj i design af et hardwaresystem. Den største force ved PLD er, at de kan programmeres til at udføre mange forskellige funktioner. Endvidere kan der foretages ændringer internt i kredsen, uden at det er nødvendigt at "wrappe om". I dette projekt bruges PLD'er af typen PEEL (Programmable Electrically Erasable Logic), fordi de er de mest økonomiske under et udviklingsforløb og tilpas avancerede til formålet.

A.2 Programmering af PLD

Sædvanligvis udgiver producenten af PLD'er et programmeringsværktøj, som passer til netop deres PLD'er. Det betyder at der findes utallige værktøjer til programmering af PLD-kredse. Eftersom der findes mange forskellige producenter, kan der nemt opstå problemer med understøttelse af PLD-typer. Der er dog visse programmer, som understøtter stort set alle PLD-typer. Et af disse er programmeringsværktøjet, ABEL (Advanced Boolean Expression Language) fra PLD-producenten "DATA I/O". Dette program er valgt til programmeringen af de anvendte PEEL-kredse på følgende grundlag:

- Programmet er gratis og tilgængeligt her på stedet (AAU).
- Der er blevet undervist i brugen af dette program.
- Det er forholdsvist enkelt at programmere i, og funktionskravene til disse PLD'er kan forholdsvist let implementeres.
- Programmet understøtter de fleste kredse, som overholder PLD og PAL standarderne.

A.3 DTACK-generator

```
MODULE   DTACKGEN
TITLE   'DTACK generator'
DTACKG   DEVICE  'P18CV8';

"AUTHOR:
"                    2000GR412,
"                    Aalborg Universitet,
"                    d. 25/04-2000 kl. 15.00
"
"                      ___  ___
"                     |   \/   |
"      SYSCLK -----> | 1   20 | --> Vcc
"      _A_DTACK ---> | 2   19 | --> DTACK
"      _I_DTACK ---> | 3   18 | --> HOLD_PULS
"                    | 4   17 | --> STOP_PULS
"                    | 5   16 | --> Num1
"                    | 6   15 | --> Num2
"                    | 7   14 | --> Num3
"                    | 8   13 | --> Num4
"                    | 9   12 | --> Num5
"      Ground -----> | 10  11 |
"                    | ______ |
"
"                  _ = Active low


"************************ INPUT ***************************************

SYSCLK        PIN  1;        "Clock input, 16 MHz.
_A_DTACK      PIN  2;        "DTACK signal fra adressedekoderen.
_I_DTACK      PIN  3;        "DTACK signal fra interuptdekoderen.


"************************ OUTPUT **************************************

DTACK         PIN 19;       "DTACK signal.


"************************ Pins for internal use ***********************

HOLD_PULS     PIN 18;
STOP_PULS     PIN 17;
NUM1,NUM2,NUM3,NUM4,NUM5  PIN  16,15,14,13,12;


"************************ Constants ***********************************

x = .x.;
C = .C.;
Pulswidth = 4;
Delay = 5;
Num = [NUM5,NUM4,NUM3,NUM2,NUM1];


"************************ DTACK generating ****************************

EQUATIONS

HOLD_PULS = (HOLD_PULS # !_A_DTACK) & !STOP_PULS;
HOLD_PULS = (HOLD_PULS # !_I_DTACK) & !STOP_PULS;
Num := (Num +1) & HOLD_PULS;
WHEN (Num <= Delay) # (Num > (Delay + Pulswidth)) THEN DTACK = 0;
WHEN (Num > Delay) & (Num <= (Delay + Pulswidth)) THEN DTACK = 1;
WHEN (Num >= (Delay + Pulswidth)) THEN STOP_PULS = 1;
```

```
"************************ Module test **********************************

TEST_VECTORS

( [SYSCLK,_A_DTACK,_I_DTACK] -> [DTACK] )

  [C,1,1] -> [0];
  [C,0,1] -> [0];
  [C,1,1] -> [0];
  [C,1,1] -> [0];
  [C,1,1] -> [0];
  [C,1,1] -> [0];
  [C,1,1] -> [1];
  [C,1,1] -> [1];
  [C,1,1] -> [1];
  [C,1,1] -> [1];
  [C,1,1] -> [0];
TEST_VECTORS

( [SYSCLK,_I_DTACK,_A_DTACK] -> [DTACK] )

  [C,1,1] -> [0];
  [C,0,1] -> [0];
  [C,1,1] -> [0];
  [C,1,1] -> [0];
  [C,1,1] -> [0];
  [C,1,1] -> [0];
  [C,1,1] -> [1];
  [C,1,1] -> [1];
  [C,1,1] -> [1];
  [C,1,1] -> [1];
  [C,1,1] -> [0];

END DTACKGEN
```

A.4 Adressedekoder

```
MODULE   ADR_DEK
TITLE   '11 input address decoder'
ADR_DEK  DEVICE 'P18CV8';

"AUTHOR:
"                    2000GR412,
"                    Aalborg Universitet,
"                d. 25/04-2000 kl. 15.00
"
"                      ___  ___
"                     |   \/   |
"    Address 16 ---> | 1   20  | --> Vcc
"    Address 17 ---> | 2   19  | --> ADC data enable
"    Address 18 ---> | 3   18  | --> _Statusregister enable
"    Address 19 ---> | 4   17  | --> Controlregister enable
"    Address 20 ---> | 5   16  | --> _DTACK generator select
"    Address 21 ---> | 6   15  | --> _Upper datastrobe
"    Address 22 ---> | 7   14  | --> _Lower datastrobe to IRQ-decoder
"    Address 23 ---> | 8   13  |
"    Datastrobe 0--> | 9   12  | <-- _Adressstrobe
"    Ground -------> | 10  11  | <-- _Datastrobe 1
"                    | _______ |
"
"                  _ = active low


"************************** INPUT ************************************

A16,A17,A18,A19,A20,A21,A22,A23    PIN    1,2,3,4,5,6,7,8;
_DS0,_DS1,_AS                      PIN    9,11,12;
```

```
"***************************** OUTPUT ************************************

A_LDS,_UDS,_DTACKSEL,CONTROL,_STATUS,ADCENA    PIN    14,15,16,17,18,19;

"***************************** CONSTANTS**********************************

X = .X.

"***************************** ADDRESS DECODING **************************

EQUATIONS

ADCENA     = !A16 & A17 & !A18 & A19 & A20 & A21 & A22 & A23 & !_DS0 &
             !_DS1 & !_AS;

!_STATUS   = !A16 & A17 & !A18 & A19 & A20 & A21 & A22 & A23 & !_DS0 &
             !_DS1 & !_AS;

CONTROL    = A16 & !A17 & !A18 & A19 & A20 & A21 & A22 & A23 & !_DS0 &
             !_DS1 & !_AS;

!_DTACKSEL = (!A16 & A17 & !A18 & A19 & A20 & A21 & A22 & A23 &
              !_DS0     & !_DS1 & !_AS) # (A16 & !A17 & !A18 & A19 & A20
              & A21 & A22 & A23 & !_DS0 & !_DS1 & !_AS);

!_UDS      = (!A16 & A17 & !A18 & A19 & A20 & A21 & A22 & A23 & !_DS0 &
             !_DS1 & !_AS) # (A16 & !A17 & !A18 & A19 & A20 & A21 &
             A22 & A23 & !_DS0 & !_DS1 & !_AS);

!A_LDS     = (!A16 & A17 & !A18 & A19 & A20 & A21 & A22 & A23 & !_DS0 &
             !_DS1 & !_AS) # (A16 & !A17 & !A18 & A19 & A20 & A21 &
             A22 & A23 & !_DS0 & !_DS1 & !_AS);

"***************************** MODULE TEST *******************************

TEST_VECTORS

( [A16,A17,A18,A19,A20,A21,A22,A23,_DS0,_DS1,_AS] ->
  [ADCENA,_STATUS,CONTROL,_DTACKSEL,_UDS,A_LDS] )

  [0,1,0,1,1,1,1,1,0,0,0] -> [1,0,0,0,0,0];
  [1,0,0,1,1,1,1,1,0,0,0] -> [0,1,1,0,0,0];
  [0,0,0,0,0,0,0,0,0,0,0] -> [0,1,0,1,1,1];
  [1,1,1,1,1,1,1,1,1,1,1] -> [0,1,0,1,1,1];
  [1,1,1,1,1,0,0,0,0,0,0] -> [0,1,0,1,1,1];
  [0,0,0,0,0,1,1,1,1,1,1] -> [0,1,0,1,1,1];
  [1,0,1,0,1,0,1,0,1,0,1] -> [0,1,0,1,1,1];
  [0,1,0,1,0,1,0,1,0,1,0] -> [0,1,0,1,1,1];

END ADR_DEK
```

A.5 IRQ-dekoder

```
MODULE   IRQ_DECODER
TITLE   'IRQ DECODER'
IRQDEC   DEVICE  'P18CV8';

"AUTHOR:
"                    2000GR412
"                    Aalborg University
"                    d. 04/05-2000 09.25
"
"
"                      ___  ___
"                     |   \/   |
"      Address 01 -> | 1   20 | --> Vcc
"      Address 02 -> | 2   19 |
"      Address 03 -> | 3   18 |
"      _AS --------> | 4   17 |
"      _IACK ------> | 5   16 |
"                    | 6   15 |
"      _ALDS ------> | 7   14 |
"                    | 8   13 | --> _LDS
"                    | 9   12 | --> _I_DTACK
"      Ground -----> | 10  11 |
"                    | ______ |
"
"                  _ = active low

"************************* INPUT ***************************************

A01,A02,A03,_AS,_IACK,_ALDS        PIN 1,2,3,4,5,7;

"************************* OUTPUT **************************************

_I_DTACK,_LDS                      PIN 12,13;

"************************* Constants ***********************************

X = .x.;

"************************* TRUTH TABLE *********************************

TRUTH_TABLE
( [A03,A02,A01,_AS,_IACK] -> [_I_DTACK] )
  [ 1, 0, 1, 0, 0]        -> [0];
  [ X, X, X, X, X]        -> [1];

EQUATIONS

LDS = _ALDS & _I_DTACK;


************************* TEST VECTORES *********************************

TEST_VECTORS

( [A03,A02,A01,_AS,_IACK,_ALDS] -> [_I_DTACK, _LDS] )

  [ 1, 0, 1, 0, 0, 1 ]      ->  [0, 0];
  [ X, X, X, X, X, 1 ]      ->  [1, 1];
  [ 1, 1, 1, 0, 0, 1 ]      ->  [1, 1];
  [ X, X, X, X, X, 0 ]      ->  [1, 0];

END IRQ_DECODER
```

A.6 Alarmgenerator

```
MODULE   ALARM
TITLE    'Alarmmodulets programkode'
ALARMM    DEVICE 'PEEL22CV10';

"AUTHOR:
"               2000GR412,
"               Aalborg Universitet,
"               d. 22/04-2000 kl. 12.00
"
"                  ___  ___
"                 |   \/   |
"                 | 1   24 | --> Vcc
"                 | 2   23 |
"      ALARM0 --> | 3   22 | --> SPEAKER
"      ALARM1 --> | 4   21 | --> LED7
"      ALARM2 --> | 5   20 | --> LED6
"      ALARM3 --> | 6   19 | --> LED5
"      ALARM4 --> | 7   18 | --> LED4
"      ALARM5 --> | 8   17 | --> LED3
"      ALARM6 --> | 9   16 | --> LED2
"      ALARM7 --> | 10  15 | --> LED1
"                 | 11  14 | --> LED0
"      Ground --> | 12  13 |
"                 | _____  |

"************************ INPUT ****************************************

ALARM0,ALARM1,ALARM2,ALARM3,ALARM4,ALARM5,ALARM6,ALARM7 PIN
3,4,5,6,7,8,9,10;


"************************ OUTPUT ***************************************

LED0,LED1,LED2,LED3,LED4,LED5,LED6,LED7,SPEAKER PIN
14,15,16,17,18,19,20,21,22;


"************************ EQUATIONS ************************************

EQUATIONS

LED0 == ALARM0;
LED1 == ALARM1;
LED2 == ALARM2;
LED3 == ALARM3;
LED4 == ALARM4;
LED5 == ALARM5;
LED6 == ALARM6;
LED7 == ALARM7;
Speaker == ALARM0 # ALARM1 # ALARM2 # ALARM3 # ALARM4 # ALARM5 # ALARM6 #
           ALARM7;

"************************ TEST VECTORES ********************************

TEST_VECTORS

([ALARM0,ALARM1,ALARM2,ALARM3,ALARM4,ALARM5,ALARM6,ALARM7]->
 [LED0,LED1,LED2,LED3,LED4,LED5,LED6,LED7,Speaker])

[0,0,0,0,0,0,0,0] -> [0,0,0,0,0,0,0,0,0];
[0,1,0,0,0,0,0,0] -> [0,1,0,0,0,0,0,0,1];
[1,0,1,0,0,0,0,0] -> [1,0,1,0,0,0,0,0,1];
[1,0,0,0,1,0,1,0] -> [1,0,0,0,1,0,1,0,1];
END ALARM
```

A.7 Delaymodul

```
MODULE   DELAYMODULE
TITLE   'Delay module for ADC og kontrolregister'
DELAY    DEVICE  'P18CV8';

"AUTHOR:
"                   2000GR412,
"                   Aalborg University,
"                   21/04-2000 12.14
"
"
"                     ___  ___
"                    |   \/   |
"      COUNT_CLK --> |  1  20 | --> Vcc
"      IN1 --------> |  2  19 | --> ADC_CLK
"      IN2 --------> |  3  18 | --> OUT1
"      IN3 --------> |  4  17 | --> OUT2
"      _CON_ENAI --> |  5  16 | --> OUT3
"      INA --------> |  6  15 | --> _CON_ENAO
"      INB --------> |  7  14 | --> OUTA
"      INC --------> |  8  13 | --> OUTB
"                    |  9  12 | --> OUTC
"      Ground -----> | 10  11 |
"                    | ______ |
"
"                   _ = Active low
" Eksternt er PIN IN1,PIN IN2,..  forbundet til PIN OUT1,PIN OUT2,..


"************************ INPUT ****************************************

 COUNT_CLK    PIN 1;        "Start Counter"
_CON_ENAI     PIN 5;


ADC_CLK       PIN 19;       "Start ADC"
_CON_ENAO     PIN 15;

"************************ Internal declarations *************************

IN1,IN2,IN3   PIN 2,3,4;
INA,INB,INC   PIN 6,7,8;

OUT1,OUT2,OUT3  PIN 18,17,16;
OUTA,OUTB,OUTC  PIN 14,13,12;


"************************ Delay ****************************************

OUT1    = COUNT_CLK;
OUT2    = IN1;
ADC_CLK = IN2;

OUTA      = _CON_ENAI;
OUTB      = INA;
OUTC      = INB;
_CON_ENAO = INC;

END DELAYMODULE
```

A.8 IRQ-generator

```
MODULE   IRQ_GEN
TITLE   'Control module for IRQ signal
IRQGEN   DEVICE  'P18CV8';

"Author
"                    2000GR412
"                    Aalborg University
"                    04 Maj 2000 09:26
"
"
"                      ___  ___
"                     |   \/   |
"      EOC (CLK) -->  |  1  20 |  --> Vcc
"      PS1 -------->  |  2  19 |  --> O1
"      PS2 -------->  |  3  18 |  --> O2
"      PS3 -------->  |  4  17 |  --> O3
"      PS4 -------->  |  5  16 |  --> O4
"      PS5 -------->  |  6  15 |  --> O5
"      PS6 -------->  |  7  14 |  --> IRQ5
"      PS7 -------->  |  8  13 |  --> Patient ID 3
"      PS0 -------->  |  9  12 |  --> Patient ID 2
"      Ground ----->  | 10  11 |  --> Patient ID 1
"                     | ______ |
"
"                   _ = active low


"************************ INPUTS ****************************************

PS0,PS1,PS2,PS3,PS4,PS5,PS6,PS7 PIN 9,2,3,4,5,6,7,8;
EOC                              PIN 1;
PID1,PID2,PID3                   PIN 11,12,13;


"************************ OUTPUTS ***************************************

IRQ5                             PIN 14;


"************************ Define special constants **********************

X = .x.;
C = .c.;


"************************ EQUATIONS *************************************

EQUATIONS

IRQ5 := PS0 & !PID3 & !PID2 & !PID1 #
        PS1 & !PID3 & !PID2 &  PID1 #
        PS2 & !PID3 &  PID2 & !PID1 #
        PS3 & !PID3 &  PID2 &  PID1 #
        PS4 &  PID3 & !PID2 & !PID1 #
        PS5 &  PID3 & !PID2 &  PID1 #
        PS6 &  PID3 &  PID2 & !PID1 #
        PS7 &  PID3 &  PID2 &  PID1;
```

```
"************************ TEST VECTORES *********************************

TEST_VECTORS

( [EOC,PS0,PS1,PS2,PS3,PS4,PS5,PS6,PS7,PID3,PID2,PID1] -> IRQ5 )

  [ C, 1, X, X, X, X, X, X, X, 0, 0, 0 ] ->  1;
  [ C, X, 1, X, X, X, X, X, X, 0, 0, 1 ] ->  1;
  [ C, X, X, 1, X, X, X, X, X, 0, 1, 0 ] ->  1;
  [ C, X, X, X, 1, X, X, X, X, 0, 1, 1 ] ->  1;
  [ C, X, X, X, X, 1, X, X, X, 1, 0, 0 ] ->  1;
  [ C, X, X, X, X, X, 1, X, X, 1, 0, 1 ] ->  1;
  [ C, X, X, X, X, X, X, 1, X, 1, 1, 0 ] ->  1;
  [ C, X, X, X, X, X, X, X, 1, 1, 1, 1 ] ->  1;
  [ C, X, X, X, X, X, X, X, X, X, X, X ] ->  0;
  [ C, 0, 0, 0, 0, 0, 0, 0, 0, X, X, X ] ->  0;
  [ C, 0, 0, 0, 0, 0, 0, 0, 0, X, X, X ] ->  0;
  [ C, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0 ] ->  1;
  [ C, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1 ] ->  0;
  [ C, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0 ] ->  1;
  [ C, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1 ] ->  0;
  [ C, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0 ] ->  0;
  [ C, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1 ] ->  1;
  [ C, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0 ] ->  1;
  [ C, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1 ] ->  1;

END IRQ_GEN
```

B Generelt om VMEbus

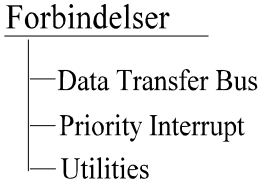
VMEbus (Versa Module Europe) er en industristandard, oprindeligt designet til brug med Motorola 68000, men den er nu blevet en verdensstandard, og indgår også i systemer med andre processorer.
Bussens benyttes i dette projekt for at muliggøre kommunikation mellem mikroprocessorkortet og dataopsamlingsenheden. Endvidere specificerer VMEbusstandarden nogle protokoller, der nøjagtigt angiver samspillet mellem VMEbus og de systemer, der er interfacet til den.
Desuden specificerer bussen den elektriske og mekaniske karakteristik for interfacet på de systemer, som skal kommunikere gennem VMEbussen.

B.1 Mekanisk

Standarden for VMEbussens mekaniske opbygning er ikke interresant i denne sammenhæng, da det udleverede mikroprocessorkort ikke er wrappet efter denne standard.
Bagest i dette afsnit ses, hvordan mikroprocessorkortet er wrappet med hensyn til VMEbus forbindelser.

B.2 Forbindelser

VMEbussens forbindelser kan, for nemmere anskuelighed, deles op i tre underafsnit som illustreret på figuren. I det følgende er det kun de relevante forbindelser, for dette projekt, der er nævnt**.**



Figur B.1 Opdeling af VMEbussen i tre hovedgrupper.

B.2.1 Data Transfer Bus (DTB)

Disse forbindelser bruges til at overføre data mellem mikroprocessorkortet og dataopsamlingsenheden (master og slave). Forbindelserne består af en databus og en adressebus samt kontrolsignale r til disse.
Adressebussen (A_{01} - A_{23}) bruges til de signaler, som bestemmer, hvorfra der læses og hvortil der skrives.
Databussen (D_{00} - D_{15}) er kommunikationsleddet for dataoverførslerne mellem dataopsamlingsenheden og mikroprocessorkortet.

Data strobe 0 og Data strobe 1 (DS0*, DS1*) bestemmer størrelsen på datastrengen. Altså, hvorvidt der skal sendes bytes eller words over databussen. Disse svarer til henholdsvis LDS* og UDS* på M68k.
READ/WRITE (WRITE*) bestemmer om masteren (mikroprocessorkortet) læser eller skriver. Denne forbindelse skal bruges for at kunne kommunikere begge veje mellem enhederne.
Data Transfer Acknowledge (DTACK*) er det handshake-signal slaven sender til masteren når data er klar på busses.

B.2.2 Priority Interrrupt

Disse forbindelser anvendes, hvis slaven ønsker at blive serviceret af masteren ved interrupt. Priority interrupt forbindelserne består af syv forskellige forbindelser (IRQ1* - IRQ7*), så 7 forskellige moduler på dataopsamlingsenheden kan tildeles forskellig prioritet, og altså hver sin interrupt.

IRQ5* benyttes af dataopsamlingsenheden, når den ønsker at blive serviceret af mikroprocessorkortet.
Interrrupt acknowledge (IACK*) benyttes af mikroprocessorkortet, når det interrupt dataopsamlingsenheden udsender er modtaget og mikroprocessorkortet er klar til at servicere dataopsamlingsenheden.

B.2.3 Utilities

System clock 16 MHz (SYSCLK) er et konstant 16 MHz clock signal, som er uafhængigt af processorens hastighed. Det kan bruges til at time signalhastighederne på og fra data-opsamlingsenheden

B.3 Elektrisk specifikation

VMEbussen er en standardbus, og dens signalniveauer er fastlagt til at være kompatible med TTL logik.
På nedenstående figur ses de spændingsniveauer, som svarer til et logisk 1 og et logisk 0 på hver side af bussen.

Værdierne, som er angivet i figuren, er minimumsværdier for et logisk 1 og maksimalværdier for det logiske 0. En anden måde at udtrykke dette på ses i nedenstående erklæringer:

0 V	<	Driver low output level	<	0.6 V
0 V	<	Receiver low input level	<	0.8 V
2.4 V	<	Driver high output level	<	5.0 V
2.0 V	<	Receiver high input level	<	5.0 V

[Clements, 1997]

Spændingsværdierne 0 V og 5 V er henholdsvis stel og forsyningsspænding, og gør systemet kompatibelt med TTL logik.
Der er ligeledes nogle krav til strømmen, der løber på VMEbussen. Disse er som følger:

Driver low level output current > 64 mA
Driver high level output current > 3 mA
Driver output low level current when tristatet < 450 µA
Driver output high level current when tristatet < 100 µA
[Clements, 1997]

Ifølge [Clements, 1997] er 74LS-serien hensigtsmæssig at implementere som driver, tranceiver og receiver, hvorfor kredse fra denne serie benyttes til interfacet på dataopsamlingsenheden.

B.4 Protokol

For at bestemme, hvordan der kommunikeres over VMEbussen, er det nødvendigt, at anvende en protokol. En protokol er et regelsæt, som bestemmer, hvordan enheder kommunikerer.
I det pågældende tilfælde er der brug for følgende protokoller:

- READ & WRITE protokol.
- IRQ protokol.

I disse protokoller er visse forbindelser udeladt, da de ikke har betydning for det aktuelle system. Disse protokoller findes i manualen til VMEbussen [VMEbus, 1982].
Protokollerne definerer hvordan timingen skal være i de forskellige cycles.

B.5 VMEbus forbindelser

VMEbus forbindelser								
WW-pin	DIN connector	Signal mnemonic	WW-pin	DIN connector	Signal mnemonic	WW-pin	DIN connector	Signal mnemonic
A 1	1 A	D00	B 1	30 A	A01	C 1	30 B	IRQ1*
A 2	2 A	D01	B 2	29 A	A02	C 2	29 B	IRQ2*
A 3	3 A	D02	B 3	28 A	A03	C 3	28 B	IRQ3*
A 4	4 A	D03	B 4	27 A	A04	C 4	27 B	IRQ4*
A 5	5 A	D04	B 5	26 A	A05	C 5	26 B	IRQ5*
A 6	6 A	D05	B 6	25 A	A06	C 6	25 B	IRQ6*
A 7	7 A	D06	B 7	24 A	A07	C 7	24 B	IRQ7*
A 8	8 A	D07	B 8	30 C	A08	C 8	20 A	IACK*
A 9	1 C	D08	B 9	29 C	A09	C 9	12 B	BR0*
A 10	2 C	D09	B 10	28 C	A10	C 10	13 B	BR1*
A 11	3 C	D10	B 11	27 C	A11	C 11	14 B	BR2*
A 12	4 C	D11	B 12	26 C	A12	C 12	15 B	BR3*
A 13	5 C	D12	B 13	25 C	A13	C 13	10 A	SYSCLK
A 14	6 C	D13	B 14	24 C	A14	C 14	12 C	SYSRESET*
A 15	7 C	D14	B 15	23 C	A15	C 15	10 C	SYSFAIL*
A 16	8 C	D15	B 16	22 C	A16	C 16	3 B	ACFAIL*
A 17	13 A	DS0*	B 17	21 C	A17	C 17	16 B	AM0
A 18	12 A	DS1*	B 18	20 C	A18	C 18	17 B	AM1
A 19	14 A	WRITE*	B 19	19 C	A19	C 19	18 B	AM2
A 20	13 C	LWORD*	B 20	18 C	A20	C 20	19 B	AM3
A 21	16 A	DTACK*	B 21	17 C	A21	C 21	23 A	AM4
A 22	11 C	BERR*	B 22	16 C	A22	C 22	14 C	AM5
A 23	1 B	BBSY*	B 23	15 C	A23	C 23	21 B	SERCLK
A 24	2 B	BCLR*	B 24	18 A	AS*	C 24	22 B	SERDAT*

C Timingdiagrammer

Dette appendiks indeholder timingdiagrammer for dataopsamlingsenheden. Disse skal illustrere dataflowet i enheden og dokumentere de tider, som er anvendt gennem rapporten.
Da denne enhed kan tilgås på tre forskellige måder, vil dette afsnit ligeledes være delt op i tre tilfælde; read, write og en interruptcycle. Det vil først blive vist, hvordan timingdiagrammet ser ud, hvorefter der følger en beskrivelse af de forskellige tider, som er angivet. Der vil kun blive betragtet de tider, som er væsentlige på dataopsamlingsenheden, hvilket betyder at M68k-timinger ikke angives eller forklares.

Resten af denne side er blank for at sikre,
at timingdiagrammerne står overfor de tilhørende skemaer.

C.1 Read cycle

Intern timing på dataopsamlingsenheden i en read cycle			
Parameterbeskrivelse	Symbol	Minimum	Maximum
AS* på VMEbus til UDS*, ADC_ENA, STA_ENA*, A_DTACK* på intern bus	t_1	-	28 ns
Tid fra UDS* til LDS*	t_2	-	10 ns
ADC_ENA til data er klar på VMEbussen	t_3	-	262 ns
STA_ENA* til data er klar på VMEbussen	t_4	-	50 ns
Delay fra A_DTACK* til DTACK	t_5	262 ns (1)	-
Delaytid for DTACK til DTACK*		-	22 ns
DTACK* aktiveret på VMEbussen til AS* deaktiveres på VMEbussen	t_6	-	215 ns
DTACK* pulsbredde	t_7	215 ns (2)	-
STA_ENA* deaktiveret til data er fjernet fra VMEbussen	t_8	-	25 ns
ADC_ENA deaktiveret til data er fjernet fra VMEbussen	t_9	-	35 ns
UDS* deaktiveret til LDS* deaktiveret	t_{10}	-	10 ns
AS*, DS0* - DS1* deaktiveret på VMEbussen til UDS*, ADC_ENA, STA_ENA* og A_DTACK* er deaktiveret på den interne bus	t_{11}	-	28 ns
Periodetiden for SYSCLK fra VMEbussen	t_{cyc}	62,5 ns	

(1) Denne tid skal mindst være lige så lang som t_3. Denne er valgt til 312,5 ns i afsnit 5.11.
(2) Denne tid skal mindst være lige så lang som t_6. Denne er valgt til 250 ns i afsnit 5.11

C.2 Write cycle

Intern timing på dataopsamlingsenheden i en write cycle			
Parameterbeskrivelse	Symbol	Minimum	Maximum
Tid fra DS0* - DS1* på VMEbus til UDS*, A_DTACK* på intern bus	t_1	-	28 ns
Tid fra UDS* til LDS*	t_2	-	10 ns
Tid fra UDS* til CON_ENA	t_3	57 ns (1)	-
Tid fra CON_ENA til DTACK*	t_4	21 ns	-
Delaytid for DTACK til DTACK*	-	-	22 ns
Tid fra DTACK* aktiveret på VMEbussen til AS* deaktiveres på VMEbussen	t_5	-	215 ns
Tid fra AS* deaktiveret til CON_ENA deaktiveret	t_6	-	40 ns (2)
Tid fra AS* deaktiveret til data er fjernet fra VMEbussen	t_7	40 ns	-
Tid fra AS* deaktiveret til DTACK* deaktiveret	t_8	-	240 ns (3)
Periodetiden for SYSCLK fra VMEbussen	t_{cyc}	62,5 ns	

(1) Denne tid skal mindst være tiden det tager at aktivere interfacet.
(2) Denne tid må højest være tiden fra AS* er deaktiveret, til data er fjernet fra VMEbussen; altså tiden t_7.
(3) Den maximale tid der må gå inden DTACK* deaktiveres.

C.3 IRQ cycle

Intern timing på dataopsamlingsenheden i en interrupt cycle			
Parameterbeskrivelse	Symbol	Minimum	Maximum
Tid fra AS* på VMEbussen til LDS* på den interne bus	t_1	-	38 ns
Tid fra IACK* til data er stabil på VMEbussen	t_2	-	92 ns (1)
Tid fra I_DTACK* til DTACK	t_3	64 ns	-
Delaytid for DTACK til DTACK*	-	-	22 ns
DTACK* aktiveret på VMEbussen til AS* deaktiveres på VMEbussen	t_4	-	215 ns
AS* deaktiveret til data er fjernet fra VMEbussen	t_5	-	53 ns
AS* deaktiveret til I_DTACK* er deaktiveret	t_6	-	28 ns
AS* deaktiveret til DTACK* deaktiveret	t_7	-	240 ns (2)
Periodetiden for SYSCLK fra VMEbussen	t_{cyc}	62,5 ns	

(1) Tiden inkluderer aktivering af line driver til IRQ-vektoren.
(2) Den maximale tid, der må gå inden DTACK*, skal være væk.

D Kode, mikroprocessorprogram

Dette afsnit indeholder kildekoden til de programmer, som er skrevet til mikroprocessorkortet. Kildekoden er delt i tre dele, som det blev beskrevet i afsnittet Mikroprocessorprogram. Disse er en applikation “APPm68.c”, et operativsystem “OS412.m68” samt en hardwaredefinition “hwdef.m68”. Den første del er “hwdef.m68”, den næste “OS412.m68” og derefter “APPm68.c”.
Herefter er der inkluderet kildekoden til den linkfil, som skal benyttes til at linke de øvrige filer sammen. Filnavnet på denne linkfil er “ekg200.lnk”.

D.1 Hardwaredefinition (hwdef.m68)

```
hwdef idnt 1,3

* Module:            Hwdef
* File:              Hwdef.m68
* Description:       Definerer ram på mikroprocessorkortet.
* Author:            Gruppe 412.
* Date:              31.05.2000.

************* Definition af Memorymap 5 på VMPM68KA-2 **************

RamStart      equ    $0
RamSize       equ    $10000-$1000
RamEnd        equ    RamSize
VMEStart      equ    RamEnd+$1000
VMEEnd        equ    $FB0000
UARTStart     equ    $FEFFE0
UARTEnd       equ    $FEFFFF


*************** Definition af Dataopsamlingsenheden ****************

ContReg       equ    $F90000
StatReg       equ    $FA0000


*********************** Standard benævnelser ***********************

RSR           equ    $1b


*************************** Ram områder ****************************

VectorSpace           equ    $400
PEPbugReserved        equ    $600
IOReserved            equ    $1000
SStackSize            equ    $200
UStackSize            equ    $200
VMEBufferSize         equ    $100
```

```
********************** Ram opbygning *******************************

              org    RamStart
              ds.b   VectorSpace
              ds.b   IOReserved+PEPbugReserved
ProgramStart  equ    *
              org    RamEnd-(SStackSize+UStackSize+VMEBufferSize)
ProgramEnd    equ    *
ProgramSize   equ    ProgramEnd-ProgramStart
ReadPointer   equ    *
              ds.l   1
WritePointer  equ    *
              ds.l   1
BufferStart   equ    *
              ds.b   VMEBufferSize
BufferEnd     equ    *

UStackEnd     equ    *
              ds.b   UStackSize
UStackStart   equ    *

SStackEnd     equ    *
              ds.b   SStackSize
SStackStart   equ    *
BufferError   equ    *
              ds.l   1
ControlStatus equ    *
              ds.l   1

********************** Til eksterne referencer ***********************

       xdef   RamStart,RamSize,RamEnd
       xdef   VMEStart,VMEEnd
       xdef   UARTStart,UARTEnd,RSR
       xdef   ContReg,StatReg,ControlStatus
       xdef   SStackSize,UStackSize
       xdef   ProgramStart,ProgramEnd,ProgramSize
       xdef   VMEBufferSize,BufferStart,BufferEnd,ReadPointer
       xdef   WritePointer,BufferError
       xdef   UStackStart,UStackEnd
       xdef   SStackStart,SStackEnd
```

D.2 Operativsystem (OS412.m68)

```
* Module:            OS412
* File:              OS412.m68
* Description:       Definerer de koder som ligger i operativsystemet.
* Author:            Gruppe 412
* Date:              31.05.2000

os412 idnt 1,0

              OPT    p=68000,BRB

/******* Definerer eksterne og interne funktioner ********/


              xref   StatReg,ContReg,ControlStatus
              xref   BufferStart,BufferEnd,ReadPointer,WritePointer
              xref   UStackStart,SStackStart,UARTStart,RSR,BufferError
              xdef   _Initialize,_Warmstart,_Coldstart
              xdef   _MaskIrq,_UnmaskIrq,_StartAlarm
              xdef   _StopAlarm,_SetPatient,_UARTTest
              xdef   _ReadRs232,_WriteRs232
              xdef   _WriteBuffer,_ReadBuffer,_InitBuffer
```

```
/**************** Initialiseringsfunktion ****************/
/*                                                       */
/* Denne subrutine initialiserer vektortabellen på 100h. */
/* User stacken initialiseres og adressen buffererror    */
/* nulstilles. ControlStatus nulstilles for at kunne     */
/* holde styr på hvad der står i kontrolregistret på     */
/* dataopsamlingsenheden. Til sidst initialiseres        */
/* kontrolregistret og ControlStatus opdateres           */
/*                                                       */
/*******************************************************/

_Initialize


              movem.l       a0-a1,-(sp)
              lea.l         IrqRoutine,a1
              move.l        a1,$100
              lea.l         UStackStart,a0
              move.l        a0,USP
              clr.l         BufferError
              clr.l         ControlStatus
              lea.l         ContReg,a1
              move.w        #$0000,ControlStatus
              move.w        ControlStatus,(a1)
              movem.l       (sp)+,a0-a1
              rts

/****************** Varmstart af M68k ******************/
/*                                                     */
/* Denne subrutine genstarter mikroprocessoren, uden at  */
/* resette vektortabellen.                             */
/*                                                     */
/*******************************************************/

_Warmstart

              move.l        d7,-(sp)
              move.b        #$e8,d7
              trap          #14
              move.l        (sp)+,d7
              rts

/****************** Koldstart af M68k ******************/
/*                                                     */
/* Denne subrutine genstarter mikroprocessoren, og       */
/* resette vektortabellen og registre                  */
/*                                                     */
/*******************************************************/

_Coldstart

              move.l        d7,-(sp)
              move.b        #$e9,d7
              trap          #14
              move.l        (sp)+,d7
              rts

/****************** Frakobling af IRQ ******************/
/*                                                     */
/* Denne subrutine sætter IRQ masken i M68k’s status     */
/* register til 7, uden at overskrive CCR.             */
/*                                                     */
/*******************************************************/

_MaskIrq

              move.l        d1,-(sp)
              move.w        sr,d1
```

```
              ori.w         #$2700,d1
              move.w        d1,sr
              move.l        (sp)+,d1
              rts

/****************** Tilkobling af IRQ ******************/
/*                                                     */
/* Denne subrutine sætter IRQ masken i M68k's status   */
/* register til 0, uden at overskrive CCR.             */
/*                                                     */
/*******************************************************/

_UnmaskIrq

              move.l        d1,-(sp)
              move.w        sr,d1
              andi.w        #$20ff,d1
              move.w        d1,sr
              move.l        (sp)+,d1
              rts

/*************** Starter alarmen på DAOE  ***************/
/*                                                     */
/* Denne subrutine skriver en alarm ud i kontrolregistret*/
/* på dataopsamlingsenheden. Værdien tages fra stakken.  */
/* ControlStatus adressen opdateres.                   */
/*                                                     */
/*******************************************************/

_StartAlarm

              movem.l       d1/a1,-(sp)
              move.l        12(sp),d1
              or.w          ControlStatus,d1
              move.w        d1,ControlStatus
              lea.l         ContReg,a1
              move.w        ControlStatus,(a1)
              movem.l       (sp)+,d1/a1
              rts

/*************** Stopper alarmen på DAOE  ***************/
/*                                                     */
/* Denne subrutine skriver nuller i de øverste 8 bit i   */
/* kontrolregistret for at stoppe alle alarmer.        */
/* ControlStatus adressen opdateres.                   */
/*                                                     */
/*******************************************************/

_StopAlarm

              movem.l       d1/a1,-(sp)
              move.w        #$00ff,d1
              and.w         ControlStatus,d1
              move.w        d1,ControlStatus
              lea.l         ContReg,a1
              move.w        ControlStatus,(a1)
              movem.l       (sp)+,d1/a1
              rts



/********* Tilslutter/frakobler patient på DAOE **********/
/*                                                     */
/* Denne subrutine skriver i de nederste 8 bit i       */
/* kontrolregistret for at tilslutte/frakoble en patient.*/
/* ControlStatus adressen opdateres.                   */
/*                                                     */
/*******************************************************/
```

```
_SetPatient

              movem.l       d1-d2/a1,-(sp)
              move.l        16(sp),d1
              add.w         #$FFFE,d1
              move.w        ControlStatus,d2
              or.w          #1,d2
              and.w         d1,d2
              move.w        d2,ControlStatus
              lea.l         ContReg,a1
              move.w        ControlStatus,(a1)
              movem.l       (sp)+,d1-d2/a1
              rts

/*********** Tester om der kommet data fra PC ************/
/*                                                       */
/* Denne subrutine undersøger om bit 7 i UART'ens        */
/* receiver status register. Hvis bitten er sat høj      */
/* returneres et. Ellers returneres nul over D0.         */
/*                                                       */
/*******************************************************/

_UARTTest

              move.l        a0,-(sp)
              lea           UARTStart,a0
              clr.l         d0
              btst          #7,RSR(a0)
              beq           Return
              moveq         #1,d0
Return        move.l        (sp)+,a0
              rts

/**************** Læser data fra UART'en *****************/
/*                                                       */
/* Denne subrutine læser data fra mikroprocessorkortets  */
/* UART ved kald af TRAP #14. Startadressen hvor den skal*/
/* placere data ligges i A6.                             */
/*                                                       */
/*******************************************************/

_ReadRs232

              movem.l       d7/a5-a6,-(sp)
              move.l        16(sp),d0
              move.l        20(sp),a6
              move.l        a6,a5
              move.b        #$f3,d7
              trap          #14
              clr.b         (a6)
              clr.l         d0
              movem.l       (sp)+,d7/a5-a6
              rts

/*************** Skriver data til UART'en **************/
/*                                                       */
/* Denne subrutine skriver data til mikroprocessorkortets*/
/* UART ved kald af TRAP #14. Startadressen, hvor den    */
/* skal læse, placere i A5 og slutadressen i A6.         */
/*                                                       */
/*******************************************************/

_WriteRs232

              movem.l       d6-d7/a5-a6,-(sp)
              move.l        20(sp),d6
              move.l        24(sp),a5
              move.l        a5,a6
              add.l         d6,a6
```

```
                move.b          #$f5,d7
                trap            #14
                clr.b           (a6)
                movem.l         (sp)+,d6-d7/a5-a6
                rts

/****************** Læser i VMEbuffer ******************/
/*                                                     */
/* Denne subrutine læser et word fra ringbufferen. Der */
/* undersøges om readpointer står øverst i bufferen og */
/* hvis den gør det flyttes readpointer til starten.   */
/* Der undersøges ligeledes om ReadPointer er lig med  */
/* WritePointer, i hvilket tilfælde der springes til   */
/* UnderRun.                                           */
/*                                                     */
/*******************************************************/

_ReadBuffer

                movem.l         d1-d2/a1-a2,-(sp)
                clr.l           d1
                move.l          WritePointer,d2
                cmp.l           ReadPointer,d2
                beq             Underrun
                move.l          20(sp),a1
                move.l          ReadPointer,a2
                move.l          (a2),d1
                addi.l          #2,ReadPointer
                andi.l          #$000000ff,d1
                move.l          d1,(a1)
                move.l          24(sp),a1
                move.l          #0,d1
                move.l          d1,(a1)
                move.l          #BufferEnd,d2
                subi.l          #2,d2
                cmp.l           ReadPointer,d2
                beq             MoveRP
                movem.l         (sp)+,d1-d2/a1-a2
                rts
MoveRP          move.l          #BufferStart,d1
                move.l          d1,ReadPointer
                movem.l         (sp)+,d1-d2/a1-a2
                rts
Underrun        move.l          24(sp),a1
                move.l          #1,d1
                move.l          d1,(a1)
                movem.l         (sp)+,d1-d2/a1-a2
                rts

/***************** Skriver i VMEbuffer ******************/
/*                                                     */
/* Denne subrutine skriver et word i ringbufferen. Der */
/* undersøges om WritePointer er lim med ReadPointer og */
/* hvis dette er tilfældet tælles BufferError en op.   */
/* Der undersøges om WritePointer står øverst i bufferen */
/* og hvis den gør det, flyttes WritePointer til starten.*/
/*                                                     */
/*******************************************************/

_WriteBuffer

                movem.l         d1/a0,-(sp)
                move.w          12(sp),d1
                move.l          WritePointer,a0
                move.w          d1,(a0)
                addi.l          #2,WritePointer
                move.l          ReadPointer,a0
                cmpa.l          WritePointer,a0
                beq             BufferFull
```

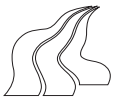
```
              move.l        #BufferEnd,d1
              cmp.l         WritePointer,d1
              beq           MoveWP
              movem.l       (sp)+,d1/a0
              rts
MoveWP        lea.l         BufferStart,a0
              move.l        a0,WritePointer
              move.l        ReadPointer,a0
              cmpa.l        WritePointer,a0
              beq           BufferFull
              movem.l       (sp)+,d1/a0
              rts
BufferFull    addi.l        #1,BufferError
              movem.l       (sp)+,d1/a0
              rts

/************* Initialisering af VMEbuffer ***************/
/*                                                       */
/* Denne subrutine initialiserer ringbufferen. Dette     */
/* gøres ved at skrive BufferStart ud i WritePointer og  */
/* ReadPointer adressen.                                 */
/*                                                       */
/*******************************************************/

_InitBuffer

              movem.l       a0-a1,-(sp)
              lea.l         BufferStart,a0
              move.l        a0,WritePointer
              lea.l         BufferStart,a1
              move.l        a1,ReadPointer
              movem.l       (sp)+,a0-a1
              rts

/********************** IRQ rutine ***********************/
/*                                                       */
/* Denne subrutine kaldes af mikroprocessoren når data-  */
/* opsamlingsenheden udsender et IRQ. IRQ rutinen kalder */
/* _WriteBuffer hvorved data skrives til bufferen        */
/*                                                       */
/*******************************************************/

IrqRoutine

              movem.l       d2/a1,-(sp)
              lea.l         StatReg,a1
              move.w        (a1),d2
              andi.l        #$000001ff,d2
              move.w        d2,-(sp)
              jsr           _WriteBuffer
              add.l         #2,sp
              movem.l       (sp)+,d2/a1
              rte
```

D.3 Applikation (APPm68.c)

```
/********* Include filer som benyttet i appm68 ***********/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*********************************************************/


/************ Eksterne funktioner fra os412 **************/


extern int Initialize();
extern int Warmstart();
extern int Coldstart();
extern int MaskIrq();
extern int UnmaskIrq();
extern int StartAlarm();
extern int StopAlarm();
extern int SetPatient();
extern int UARTTest();
extern int ReadRs232();
extern int WriteRs232();
extern int ReadBuffer();
extern int InitBuffer();

/*********************************************************/


/****************** Global Variables *********************/


int Min_Puls, Max_Puls, PickSample;
int RSTak, Space1, Sample, Count10;
unsigned char EKGArray[10];
unsigned char PulsArray[10];

/*********************************************************/


/********** Header til de benyttede funktioner ***********/


void Init();
int  TestData();
void MinPuls();
void MaxPuls();
void PatientON_OFF();
void Decode();
void Encode();
void EncodeEKG();
void DataFromPc();
void EKGReduction();
void PulsCalculation();

/*********************************************************/


/******************** Main programmet ********************/
/*                                                       */
/* Mainproceduren hvorfra alle de øvrige funktioner      */
/* kaldes. Funktionerne kaldes ikke direkte, da funktione*/
/* n DataFromPc og PulsCalculation kalder nogle andre    */
/* funktioner.                                           */
```

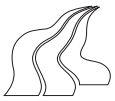
```
/*                                                          */
/************************************************************/


void main (void)
{
#asm
    lea.l      SStackStart,a0
    move.l     a0,a7
    move.w     sr,d1
    ori.w      #$2700,d1
    move.w     d1,sr
#endasm

   Init();

   for(;;)
   {
      DataFromPc();
      PulsCalculation();
   }

}

/*************** Initialiseringsfunktion ***************/


void Init()
{
   MaskIrq();
   Initialize();
   Min_Puls=0;
   Max_Puls=255;
   RSTak=0;
   Sample=0;
   Space1=0;
   PickSample=0;
   Count10=0;
   InitBuffer();
   UnmaskIrq();

}

/************************************************************/


/************** Funktion der tester data *****************/
/*                                                          */
/* Denne Procedure undersøger om det data, der er blevet */
/* hentet fra Pc, har det rigtige format. Der undersøges */
/* hvorvidt den første plads i framen er en # og derefter*/
/* undersøges det om der følger en '$' indenfor de næste */
/* 10 pladser. Funktionen returnere et 0 hvis beting-    */
/* elserne ikke er opfyldt. Der returneres 1  hvis testen*/
/* er bestået.                                              */
/*                                                          */
/************************************************************/

int TestData(Data)
char *Data;
{
   int Count=0, Space=0;
   for (Space=0;Space<10;Space++)
   {
      if (Data[Space]=='#')
         Count=Space+1;
      if (Data[Space]=='$' && (Count))
      {
         return 1;
```

```
      }
   }

   Data[0]=0;
   return 0;
}

/*********************************************************/


/*********** Funktioner der sætter variable **************/
/*                                                       */
/* Disse procedurer indlæser en pointer til arrayet Data-*/
/* String der indeholder den modtagne frame fra Pc.      */
/* Derved bliver de globale variabel sat til værdien     */
/* på plads 2 i arrayet.                                 */
/*                                                       */
/*********************************************************/


void MinPuls(Data4)
char *Data4;
{
   Min_Puls=Data4[2];

}


void MaxPuls(Data4)
char *Data4;
{
   Max_Puls=Data4[2];

}


void PatientON_OFF(Data4)
char *Data4;
{
   SetPatient(Data4[2]);

}

/*********************************************************/


/**************** Dekoder til data fra PC ****************/
/*                                                       */
/* Denne procedure indlæser en pointer til arrayet Data- */
/* String. Afhængigt af hvad der står på 2. plads i dette*/
/* array kaldes en af funktionerne MinPuls, MaxPuls eller*/
/* PatientON_OFF.                                        */
/* Denne funktion kaldes fra "DataFromPc"                */
/*                                                       */
/*********************************************************/


void Decode(Data3)
char *Data3;
{
   switch(Data3[1])
   {
      case 'N':   MinPuls(Data3);
                  break;

      case 'X':   MaxPuls(Data3);
                  break;

      case 'V':   PatientON_OFF(Data3);
```



```
                  break;
   }
}

/*********************************************************/


/***************** Encoder data til PC ******************/
/*                                                       */
/* Denne procedure indlæser to pointere. Den første "Com"*/
/* indlæser kommandoen som skal overføres via Rs232      */
/* protokollen. Kommandoen er et'P' hvis det er pulsen   */
/* som skal overføres eller et 'A' hvis det er alarmen.  */
/* Den anden pointer 'Data5' er en pointer til indholdet */
/* af puls eller alarm. Der tilføjes desuden et '$','\n',*/
/* '\r' og et '0' da Rs232 driveren på Pc bruger disse   */
/* tegn. Til slut kaldes funktionen WriteRs232 i operativ*/
/* systemet, der skriver til Pc via Rs232.               */
/* Funktionen kaldes fra "PulsCalculation".              */
/*                                                       */
/*********************************************************/


void Encode(Com, Data5)
unsigned char Com, Data5;
{
   unsigned char Frame1[7];

   Frame1[0]='#';
   Frame1[1]=Com;
   Frame1[2]=Data5;
   Frame1[3]='$';
   Frame1[4]='\n';
   Frame1[5]='\r';
   Frame1[6]=0;
   WriteRs232(7, Frame1);
}

/*********************************************************/


/***************** Encoder EKGsignalet ******************/
/*                                                       */
/* Denne procedure indlæser to pointere.Den første "Com1"*/
/* indlæser kommandoen som skal overføres via Rs232      */
/* protokollen. Kommandoen er et'E' da der skal sendes   */
/* EKG signaler.                                         */
/* Den anden pointer der indlæses er en pointer til et   */
/* et array der indeholder 10 EKG signaler. Der tilføjes */
/* desuden et '$','\n','\r' og et '0' da Rs232 driveren  */
/* på Pc bruger disse tegn. Til slut kaldes funktionen   */
/* WriteRs232 i operativsystemet, der skriver til Pc via */
/* Rs232.                                                */
/* Funktionen kaldes fra "EKGReduction".                 */
/*                                                       */
/*********************************************************/


void EncodeEKG(Com1, Data6)
unsigned char Com1, *Data6;
{
   unsigned char Frame2[16];

   Frame2[0]='#';
   Frame2[1]=Com1;
   Frame2[2]=Data6[0];
   Frame2[3]=Data6[1];
   Frame2[4]=Data6[2];
   Frame2[5]=Data6[3];
```

```
   Frame2[6]=Data6[4];
   Frame2[7]=Data6[5];
   Frame2[8]=Data6[6];
   Frame2[9]=Data6[7];
   Frame2[10]=Data6[8];
   Frame2[11]=Data6[9];
   Frame2[12]='$';
   Frame2[13]='\n';
   Frame2[14]='\r';
   Frame2[15]=0;
   WriteRs232(15, Frame2);
}

/*********************************************************/


/*********** Undersøger, og læser data fra PC ************/
/*                                                       */
/* Denne procedure undersøger om der er kommet data fra  */
/* PC. Hvis der er kommet data kaldes ReadRs232 i        */
/* operativsystemet som læser en streng i UART'en.       */
/* Dernæst undersøges det om de modtagne data er gyldige */
/* vha. funktionen TestData. Tilsidst kaldes funktionen  */
/* Decode som dekoder data fra Pc.                       */
/* Funktionen kaldes fra "Main".                         */
/*                                                       */
/*********************************************************/


void DataFromPc()
{
   unsigned char DataString[10];
   int Length=10;

   if (UARTTest())
   {
      ReadRs232(Length,DataString);
      if (TestData(DataString))
      {
         Decode(DataString);
      }

   }
}

/*********************************************************/


/**** Reducerer antallet af samples der sendes til PC  ***/
/*                                                       */
/* Denne procedure henter hvert 8. EKGsample ind i EKG-  */
/* Array. Når der er 10 samples i arrayet kaldes "Encode-*/
/* EKG", som sender arrayet videre til PC via Rs232.     */
/* Dette gøres for at reducere datamængden der sendes    */
/* over Rs232, da PC alligevel ikke kan vise alle        */
/* samples.                                              */
/* Funktionen kaldes fra PulsCalculation.                */
/*                                                       */
/*********************************************************/


void EKGReduction(EKGData)
unsigned char EKGData;
{
   unsigned char Command2;

   PickSample++;
   if (PickSample==8)
   {
```



```
      EKGArray[Count10]=EKGData;
      Count10++;
      if (Count10==10)
      {
         Count10=0;
         Command2='E';
         EncodeEKG(Command2, EKGArray);
      }
    PickSample=0;
   }

}

/*********************************************************/


/****************** Beregner pulsen *********************/
/*                                                       */
/* Denne procedure starter med at hente et sample fra    */
/* ringbufferen der ligger i operativsystemet. Derefter  */
/* skrives den indhentede værdi ind i arrayet "PulsArray"*/
/* Derefter kaldes funktionen EKGReduction der sender EKG*/
/* signalet til Pc via Rs232. Når der er skrevet 10 tal  */
/* ind i PulsArrayet beregnes 9 hældninger (a,b,c,d,e,f,g*/
/* h,i). Diff indholder et gennemsnit af disse beregnede */
/* hældningskoefficienter. Hvis den beregnede Diff er    */
/* stor nok er en QRS-tak detekteret. Derved startes en  */
/* optælling af samples indtil den næste QRS-tak         */
/* detekteres. Pulsen udregnes herfter på baggrund af det*/
/* antal af samples der er forekommet mellem de seneste  */
/* to QRS-komplekser. Da der muligvis kan detekteres en  */
/* stor hældningskoefficient to gange i løbet af en QRS- */
/* tak, undersøges der ligeledes for en negativ hældning */
/* inden den næste QRS-tak kan detekteres. Hvis Pulsen   */
/* er mindre en 'Min_Puls' eller større end 'Max_Puls'   */
/* kaldes funktionen StartAlarm fra operativsystemet der */
/* sender alarmen til dataopsamlingsenheden. Endvidere   */
/* sendes alarmen til Pc vha. funktionen 'Encode'.       */
/* Til slut kaldes funktionen 'Encode' der sender pulsen */
/* til Pc via Rs232.                                     */
/* Funktionen kaldes fra 'main'.                         */
/*                                                       */
/*********************************************************/


void PulsCalculation()
{
   unsigned int VmeDataIn=0, BufferUnderRun=0;
   int a,b,c,d,e,f,g,h,i,SampleTime,Puls,Diff;
   unsigned char Command, Command1;
   SampleTime=2;
   MaskIrq();
   ReadBuffer(&VmeDataIn,&BufferUnderRun);
   UnmaskIrq();
   if (!BufferUnderRun)
   {
      PulsArray[Space1]=VmeDataIn;
      EKGReduction(VmeDataIn);
      Space1++;
      Sample++;
      if (Space1==10)
      {
         a=(PulsArray[1]-PulsArray[0]);
         b=(PulsArray[2]-PulsArray[1]);
         c=(PulsArray[3]-PulsArray[2]);
         d=(PulsArray[4]-PulsArray[3]);
         e=(PulsArray[5]-PulsArray[4]);
         f=(PulsArray[6]-PulsArray[5]);
         g=(PulsArray[7]-PulsArray[6]);
```

```
            h=(PulsArray[8]-PulsArray[7]);
            i=(PulsArray[9]-PulsArray[8]);
            Diff=((a+b+c+d+e+f+g+h+i)/9);

            if (Sample>2000)
            {
               StartAlarm(256);
               Encode('A',1);
               Encode('P', 0);
               Sample=0;
            }

            if ((Diff>2)&&(RSTak))
            {
               Puls=(60000/(Sample*SampleTime));
               RSTak=0;
               if ((Puls<Min_Puls) || (Puls>Max_Puls))
               {
                  StartAlarm(256);
                  Command1='A';
                  Command='P';
                  Encode(Command,Puls);
                  Encode(Command1,1);
                  Sample=0;
               }
               else
               {
                  StopAlarm();
                  Command1='A';
                  Command='P';
                  Encode(Command1,0);
                  Encode(Command,Puls);
                  Sample=0;
               }

            }
            if (Diff<(-2)) RSTak=1;

            Space1=0;
         }
      }
   }
```

D.4 Linkfil (ekg2000.lnk)

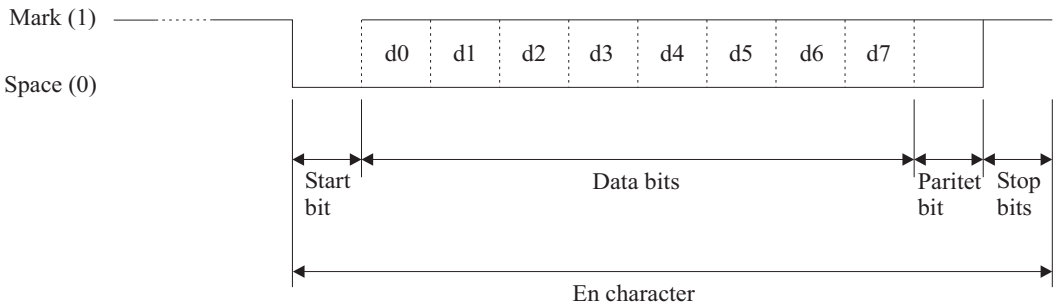
```
app41214 idnt 1,2

   link APPm68,OS412,hwdef
   link c:\Avcase68\lib\68klibc.lib()
   org ProgramStart
```

E RS232C

I dette appendiks belyses RS232C-standarden for at give et overblik over, hvordan mikrocomputer og PC kommunikerer serielt.

E.1 Funktionsprincip

RS232C-standarden beskriver, hvordan den serielle kommunikation mellem to enheder kan foregå. Standarden var oprindeligt ment som en standard mellem DTE (Data Terminal Equipment) og DCE (Data Communication Equipment), eller med andre ord mellem en terminal og et modem, og benyttes i stor udstrækning mellem terminaler. Grunden til, at den serielle kommunikation er blevet så populær, er de lave omkostninger, der er forbundet med at implementere denne teknologi, samt simpliciteten i forhold til parallel kommunikation, som til gengæld er hurtigere.
RS232C benytter sig af en asynkron seriel datatransmission, hvilket betyder, at transmitter og receiver ikke bliver synkroniseret over en tidsperiode. Dette stiller store krav til receiverens og transmitterens interne clock, da en for stor uoverensstemmelse mellem disse vil betyde, at de transmitterede data bliver misfortolket.
Til transmission af data over en seriel forbindelse benyttes en UART-chip på begge sider af transmissionsledningen. En UART er karakterorienteret, idet den sender ASCII-kodet data på 7 -



Figur E.1 En byte data, som bliver sendt serielt ved benyttelse af standarden RS232C. Signalet er vist som det ser ud inde i UART'en, da signalet har omvendt polaritet, når det sendes på transmissions-linien.

8 bit af gangen (serielt) ved en given clockfrekvens. For at disse databit kan tolkes som data, benyttes endvidere nogle kontrolbit. Disse kontrolbit består af 1 startbit, 1 - 2 stopbit samt eventuelt 1 paritetsbit til hjælp ved fejlfinding (se figur E.2). Transmissionshastigheden over en seriel forbindelse angives ofte i baud, som angiver, hvor mange bit der bliver transmitteret pr. sekund (databit og kontrolbit). Denne hastighed kan ofte indstilles på den enkelte UART og ligger typisk mellem 9.600 og 115.200 baud (bit/s).
Kommunikationen over den serielle transmissionslinie foregår ved "handshake". Dette betyder, at en transmitter først beder om lov til at sende data, hvorefter modtageren svarer. Alt efter hvad modtageren svarer, kan den egentlige dataoverførsel begynde. UART'en benytter signalerne, RTS (Request To Send) og CTS (Clear To Send) som "handshake".

E.2 Elektrisk specifikation

RS232C-standarden kan inddeles i en mekanisk specifikation og i en elektrisk specifikation. I dette afsnit vil kun den elektriske specifikation blive beskrevet. For oplysninger om den mekaniske specifikation henvises til [Clements, 1997].
Den elektriske specifikation stiller hovedsageligt krav til signalniveauerne på transmissionsledningen, men der stilles også krav til kommunikationshastighed, kabellængder, termineringer osv.
De væsentligste elektriske krav til kommunikationsgrænsefladen kan ses i nedenstående figur.

Elektriske specifikationer for RS232C

Karakteristika	**Værdi**
Max. kabellængde	15 m
Max. overførselshastighed	20.000 baud (bit/s)
Max. outputspænding fra driver	$-25\ V < V_o < +25\ V$
Min. outputspænding fra driver	$-25\ V < V_o < -5\ V$ eller $+5\ V < V_o < +25\ V$
Min. outputmodstand fra driver	300 Ω
Max. outputstrøm fra driver	500 mA
Max. slew rate fra driver	30 V/µs
Indgangsmodstand på receiver	3 - 7 kΩ
Indgangsspænding på receiver	$-25\ V < V_i < +25\ V$
Max. spændingstærskel på receivers input	$-3\ V < V_t < +3\ V$

Det ses i skemaet, at den maksimale kabellængde er 15 meter ved datahastigheder på op til 20 kbaud. I de fleste tilfælde vil der kunne benyttes meget længere kabler (100 m - 1000 m) uden problemer, idet der er en stor margin mellem signalniveauerne på input og output [Clements, 1997, s. 748]. RS232C stiller i øvrigt ingen krav til, at termineringsmodstandene skal tilpasses den karakteristiske impedans for transmissionsledningen.

F Kildekode til PC-program

I dette appendiks findes kildekoden til PC-programmet.
Kildekoden er delt op i moduler, som beskrevet i procesdesign for PC-programmet.
Til sidst findes kildekoden for det testprogram, der blev brugt til at kontrollere kommunikationen mellem PC og test-PC.

F.1 Main

```
#include <stdio.h>
#include <stdlib.h>
#include <bios.h>
#include <dos.h>

int MinPuls=0, MaxPuls=255, AktPuls=0, Hold=0, XTrend=0, XEkg=0, YLast;
int PatientOnOff=0,Testsignal=0;
float TrendTime;
struct time TimeNow;

extern void Init(int);
extern void UserInput(char);
extern void ReadRS232(void);
extern char *int_char(int);

/*****************************************************************************/
/* Funktion der konverterer en integer til en streng.                        */
/*****************************************************************************/

char *int_char(int value)
{
   int  div, ndig, dec, sign;
   char *string=0;
   char *outstring=0;
   ndig  = 0;

   div=abs(value);
   while (div>0)
   {
      ndig++;
      div=div/10;
   }
   string = ecvt(value, ndig, &dec, &sign);
   outstring=string;
   return(outstring);
}

void main (void)

/*****************************************************************************/
/* Hovedprogrammet som loopes indtil en taste er aktiveret, eller der er     */
/* modtaget data fra RS232. Hvis der trykkes på tasten 'q' afsluttes         */
/* programmet.                                                               */
/*****************************************************************************/

{
   char key=0;
```

```
   Init(1);
   while (key != 'q')
   {
      key=0;
      if (bioskey(1))
      {
         key=bioskey(0);
         UserInput(key);
      }
      ReadRS232();
   }
   Init(0);
}
```

F.2 Init

```
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <dos.h>
#include "RS232.h"

#define black 0
#define green 2
#define red 4
#define grey 7
#define blue 9
#define yellow 14
#define white 15

FILE *TestFile,*Trend;

extern void Init(int);
extern int InitCom(int,int,int,int,int);
extern void StartCom();
extern void ClearComInBuf();
extern void StopCom();
extern char *int_char(int);
extern float TrendTime;
extern struct time TimeNow;

/*****************************************************************************/
/* Funktion der laver en fadeout på skærmen, således at brugeren kan se      */
/* at programmet er korrekt afsluttet.                                       */
/*****************************************************************************/

int FadeOut(int color)
{
   int cls;
   setcolor(color);
   for (cls=0; cls<320; cls++)
   {
      line(320-cls,240+cls,320+cls,240+cls);
      line(320-cls,240-cls,320+cls,240-cls);
      line(320+cls,240-cls,320+cls,240+cls);
      line(320-cls,240-cls,320-cls,240+cls);
   }
   return(0);
}
```

F.3 UserInput

```
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <bios.h>
#include <dos.h>
#include "RS232.h"
```

```
#define black 0
#define green 2
#define red 4
#define grey 7
#define blue 9
#define yellow 14
#define white 15

extern int MinPuls, MaxPuls, AktPuls, Hold, PatientOnOff, Testsignal;
extern float TrendTime;
extern struct time TimeNow;

extern void UserInput(char);
extern void ClearComInBuf(void);
extern int WriteStrCom(const char *,int);
extern void ReadRS232(void);
extern void Alarm(int);

void InputName()

/*****************************************************************************/
/* Funktionen giver brugeren mulighed for indtastning af navn.               */
/*****************************************************************************/

{
   char key=0,Navn[30]="";
   int x=0;
   setcolor(white);
   outtextxy(325,465,"Indtast Navn:");
   while ((key != 13)&&(x != 24))   /* Kontrollerer for indtastning af enter  */
   {                                /* eller maksimal længde af navn.         */
      if (bioskey(1))               /* Der kontrolleres for tryk på en taste. */
      {
         key=bioskey(0);
         if (key == 8)              /* Kontrollerer for backspace.            */
         {
            if (x>0)
            {
               x--;
               Navn[x]=' ';         /* Sletter indtastede bogstav.            */
            }
         }
         else
         {
            Navn[x]=key;            /* Gem indtastede bogstav i navn.         */
            x++;
         }
         setviewport(450,460,637,475,1);
         clearviewport();
         setviewport(0,0,639,479,1);
         setcolor(green);
         outtextxy(450,465,Navn);   /* Genoptegn navnet.                      */
      }
      ReadRS232();                  /* Kontrollerer for data fra RS232.       */
   }
   setviewport(323,460,637,475,1);  /* Slet prompt og gammelt navn.           */
   clearviewport();
   setviewport(400,433,637,443,1);
   clearviewport();
   setviewport(0,0,639,479,1);
   setcolor(blue);
   x--;
   Navn[x]=0;
   outtextxy(400,435,Navn);         /* Genoptegn navnet.                      */
}


void InputCpr()

/*****************************************************************************/
/* Funktionen giver brugeren mulighed for indtastning af CPR. nr.            */
/*****************************************************************************/
```

```
{
   char key=0,Cpr[11]="";
   int x=0;
   setcolor(white);
   outtextxy(325,465,"Indtast Cpr.Nr:");
   while ((x != 10))                /* Kontrollerer for maksimal længde.      */
   {
      if (bioskey(1))               /* Der kontrolleres for tryk på en taste. */
      {
         key=bioskey(0);
         if (key == 8)              /* Kontrollerer for backspace.            */
         {
            if (x>0)
            {
               x--;
               Cpr[x]=' ';          /* Sletter indtastede tal.                */
            }
         }
         else
         {
            if ((key >='0') && (key <='9')) /* Kontrollerer om det er et tal. */
            {
               Cpr[x]=key;          /* Gem indtastede tal i Cpr.              */
               x++;
            }
         }
         setviewport(450,460,637,475,1);
         clearviewport();
         setviewport(0,0,639,479,1);
         setcolor(green);
         outtextxy(450,465,Cpr);    /* Genoptegn Cpr. Nr.                     */
      }
      ReadRS232();                  /* Kontrollerer for data fra RS232.       */
   }
   setviewport(323,460,637,475,1);  /* Slet prompt og gammelt Cpr. Nr.        */
   clearviewport();
   setviewport(400,443,637,453,1);
   clearviewport();
   setviewport(0,0,639,479,1);
   setcolor(blue);
   outtextxy(400,445,Cpr);          /* Genoptegn Cpr. Nr.                     */
}


void SendFrame(char c,int p)

/*************************************************************************/
/* Sørger for afsendelse af frame via RS232 til Motorola 68000.          */
/* En Frame kan eksempelvis se således ud: #NA$\n\r\0, hvilket svarer til at  */
/* sende beskeden: MaxPuls = 65, idet N betyder MaxPuls og A har værdien 65   */
/*************************************************************************/

{
   unsigned char str[10];
   str[0]='#';                      /* Framen pakkes, med de modtagne værdier */
   str[1]=c;                        /* og de definerede kontrolsignaler.      */
   str[2]=p;
   str[3]='$';
   str[4]='\n';
   str[5]='\r';
   str[6]='\0';
   ClearComInBuf();
   WriteStrCom(str,6);              /* Framen afsendes.                       */
}


void Min_Puls()

/*************************************************************************/
/* Funktionen giver brugeren mulighed for indtastning af min. puls.      */
/* Når pulsen er godkendt sendes denne via RS232 til Motorola 68000.     */
/*************************************************************************/
```

```
{
   char key=0,temp[10]="";
   int a,x=0,valid=0;
   MinPuls=0;
   setcolor(white);
   outtextxy(325,465,"Indtast Min Puls:");
   while (valid != 1)
   {
      while ((key != 13)&&(x != 3)) /* Kontrollerer for indtastning af enter  */
      {                             /* eller maksimal længde af MinPuls.      */
         if (bioskey(1))            /* Der kontrolleres for tryk på en taste. */
         {
            key=bioskey(0);
            if (key == 8)           /* Kontrollerer for backspace.            */
            {
               if (x>0)
               {
                  x--;
                  temp[x]=' ';      /* Sletter indtastede tal.                */
               }
            }
            else
            if ((key >='0')&&(key <='9')) /* Kontrollerer om det er et tal.   */
            {
               temp[x]=key;         /* Gem indtastede tal i temp.             */
               x++;
            }
            setviewport(458,460,637,475,1);
            clearviewport();
            setviewport(0,0,639,479,1);
            setcolor(green);
            outtextxy(460,465,temp);      /* Genoptegn MinPuls.               */
         }
         ReadRS232();               /* Kontrollerer for data fra RS232.       */
      }
      MinPuls=atoi(temp);
      if ((MinPuls >= 0)&&(MinPuls < MaxPuls)) /* Kontrollerer om ok.         */
         valid=1;
      else
      {
         for (a=0;a<8;a++)          /* Slet tallet hvis det ikke er ok.       */
            temp[a]=' ';
         x=0;
         key=0;
         setviewport(458,460,637,475,1);       /* Slet Prompt.               */
         clearviewport();
         setviewport(0,0,639,479,1);
      }
   }
   setviewport(323,460,637,475,1);  /* Slet prompt og gammelt tal.            */
   clearviewport();
   setviewport(38,13,79,23,1);
   clearviewport();
   setviewport(0,0,639,479,1);
   setcolor(blue);
   outtextxy(40,15,temp);           /* Genoptegn MinPuls.                     */
   SendFrame('N',MinPuls);          /* Send data.                             */
}


void Max_Puls()

/*************************************************************************/
/* Funktionen giver brugeren mulighed for indtastning af max. puls.      */
/* Når pulsen er godkendt sendes denne via RS232 til Motorola 68000.     */
/*************************************************************************/

{
   char key=0,temp[10]="";
   int a,x=0,valid=0;
   MaxPuls=255;
   setcolor(white);
```

```
   outtextxy(325,465,"Indtast Max Puls:");
   while (valid != 1)
   {
      while ((key != 13)&&(x != 3)) /* Kontrollerer for indtastning af enter  */
      {                             /* eller maksimal længde af MaxPuls.      */
         if (bioskey(1))            /* Der kontrolleres for tryk på en taste. */
         {
            key=bioskey(0);
            if (key == 8)           /* Kontrollerer for backspace.            */
            {
               if (x>0)
               {
                  x--;
                  temp[x]=' ';      /* Sletter indtastede tal.                */
               }
            }
            else
            if ((key >='0')&&(key <='9')) /* Kontrollerer om det er et tal.   */
            {
               temp[x]=key;         /* Gem indtastede tal i temp.             */
               x++;
            }
            setviewport(458,460,637,475,1);   /* Slet Prompt.                 */
            clearviewport();
            setviewport(0,0,639,479,1);
            setcolor(green);
            outtextxy(460,465,temp);      /* Genoptegn MaxPuls.               */
         }
         ReadRS232();               /* Kontrollerer for data fra RS232.       */
      }
      MaxPuls=atoi(temp);
      if ((MaxPuls > MinPuls)&&(MaxPuls <= 255)) /* Kontrollerer om ok.       */
         valid=1;
      else
      {
         for (a=0;a<8;a++)          /* Slet tallet hvis det ikke er ok.       */
            temp[a]=' ';
         x=0;
         key=0;
         setviewport(458,460,637,475,1);  /* Slet Prompt.                     */
         clearviewport();
         setviewport(0,0,639,479,1);
      }
   }
   setviewport(323,460,637,475,1);  /* Slet prompt og gammelt tal.            */
   clearviewport();
   setviewport(38,23,79,33,1);
   clearviewport();
   setviewport(0,0,639,479,1);
   setcolor(blue);
   outtextxy(40,25,temp);           /* Genoptegn MaxPuls.                     */
   SendFrame('X',MaxPuls);          /* Send data.                             */
}

void Patient_On_Off(int On)

/*************************************************************************/
/* Hvis patienten tilsluttes, afsendes besked via RS232 til Motorola 68000    */
/* og teksten skrives med hvid.                                          */
/* Tilsvarende afsendes besked via RS232 til Motorola 68000, og teksten       */
/* skrives med gråt, hvis patienten frakobles.                           */
/*************************************************************************/

{
   if (On)
   {
      PatientOnOff=1;
      SendFrame('V',1);
      setcolor(white);
      outtextxy(5,5,"PATIENT");
      outtextxy(70,5,"1");
      outtextxy(5,15,"MIN:");
```

```
      outtextxy(5,25,"MAX:");
      outtextxy(5,35,"AKT:");
      gettime(&TimeNow);            /* Henter PC´ens systemtid.              */
      TrendTime = TimeNow.ti_min;   /* TrendTime sættes lig med minuttallet.  */
   }
   else
   {
      PatientOnOff=0;
      SendFrame('V',0);
      Alarm(0);
      setcolor(grey);
      outtextxy(5,5,"PATIENT");
      outtextxy(70,5,"1");
      outtextxy(5,15,"MIN:");
      outtextxy(5,25,"MAX:");
      outtextxy(5,35,"AKT:");
    }
}

void UserInput (char key)

/*****************************************************************************/
/* Denne funktion kalder de forskellige funktioner afhængig af værdien key    */
/*****************************************************************************/

{
   switch (key)
   {
      case 'n' :    InputName();break;
      case 'c' :    InputCpr();break;
      case 'h' :    if (Hold==0)
                     Hold=1;
                  else
                     Hold=0;
                  break;
      case 'i' :    Min_Puls();break;
      case 'a' :    Max_Puls();break;
      case '1' :   if (PatientOnOff==0)
                  {
                     Patient_On_Off(1);
                     Testsignal=0;
                  }
                  else
                     Patient_On_Off(0);
                  break;
      case 't'   :   if (Testsignal==0)
                  {
                     Testsignal=1;
                     PatientOnOff=0;
                     Patient_On_Off(0);
                  }
                  else Testsignal=0;
                  break;
      default   :    break;
   }
}
```

F.4 RS232.h

```
/*****************************************************************************/
/* Filename: rs232.h                                                         */
/*                                                                           */
/* 051294/OB                                                                 */
/* Minor mods by Group 412-2000                                              */
/*****************************************************************************/

#define Err             int
#define TRUE            1
#define FALSE           0
#define COM1            0x3f8
#define COM2            0x2f8
```

```
#define ComIntrNr1       4
#define ComIntrNr2       3
#define COM             COM1           /* define COM used here              */
#define TXBuf           COM+0
#define RXBuf           COM+0
#define DivLatchMSB     COM+1
#define DivLatchLSB     COM+0
#define LineControlReg  COM+3
#define LineStatusReg   COM+5
#define ModemControlReg COM+4
#define ModemStatusReg  COM+6
#define IntrEnableReg   COM+1
#if (COM==COM1)
#define ComIntrNr       ComIntrNr1
#endif
#if (COM==COM2)
#define ComIntrNr       ComIntrNr2
#endif
#define IntrComxx       ComIntrNr+8
#define IntrController  0x20
#define EOI             0x20

#define MAXTIME         100000L         /* round 1 sec                      */
#define BufSize         200
```

F.5 RS232

```
/*****************************************************************************/
/* Driver for serial ports (COM1/COM2) in an IBM-compatible PC.              */
/* Author: Jens F.D.Nielsen I8 Aalborg University, Denmark.                  */
/* Not to be used outside AUC, and no use without proper reference to author. */
/* Minor mods by Group 412-2000                                              */
/*****************************************************************************/

#include   <conio.h>
#include   <stdio.h>
#include   <dos.h>
#include   <math.h>
#include   "rs232.h"


void MaskHWInterrupt(char intnr, int IntrEnable,int *AlreadyEnabled);
/*void (_interrupt _far Comxx)(void);*/


#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

void interrupt (*pOldComIntr)(__CPPARGS);

static unsigned int BufferWriterPointer,BufferReaderPointer,BytesInBuffer;
static unsigned char OldModemControlReg;
static unsigned char LineStatusRegister;
static unsigned char BufAr[BufSize];
static unsigned char CRreceivedInRS232;
static int AlreadyEnabled;
static int ComActive;

/*****************************************************************************/
/*   MaskHWInterrupt                                                         */
/*   Used to mask in/out interrupts on the PC interruptcontrollers(8259A)    */
/*                                                                           */
/*               controller nr                                               */
/*               1(master)  2(slave)                                         */
/*   intr req    0              timer outportbut 0                           */
/*               1              keyboard (outportbut buffer full)            */
/*               2 <--|         interrupt from slave controller             */
/*                    |--  8    realtime clock interrupt                     */
```

```
/*                      |    9     software rediretcted to int 0Ahex(irq2)     */
/*                      |   10     reserved                                    */
/*                      |   11     reserved                                    */
/*                      |   12     reserved                                    */
/*                      |   13     math coprocessor                            */
/*                      |   14     hard disk controller+                       */
/*                      |-- 15     reserved                                    */
/*                3               serial port 2                                */
/*                4               serial port 1                                */
/*                5               parallel port 2                              */
/*                6               floppy disk controller                       */
/*                7               parallel port 1                              */
/*                                                                             */
/* REMARK: ONLY THE "AT-PC" TYPE HAS TO INTRCONTROLLERS                        */
/*                                                                             */
/* A interrupt is enabled if the corresponding bit is zero...                  */
/*******************************************************************************/

void MaskHWInterrupt(char intrnr, int IntrEnable, int *AlreadyEnabled)
{
   unsigned char b,mask1,mask2;
   unsigned int contadr;
   int fail;

   fail = TRUE;

   if (intrnr <= 7 && (intrnr >= 0 )) {
      contadr = 0x21;          /* MasterIntrController. */
      mask1 = 1 << intrnr;
      mask2 = 0xFF - mask1;
      fail = FALSE;
   }

   if (intrnr >= 8 && intrnr <= 15) {
      contadr = 0xa1;           /* Slavecontroller. */
      mask1 = 1 << (intrnr - 8);
      mask2 = 0xFF - mask1;
      fail = FALSE;
   }
   if (!fail)
   {

/*******************************************************************************/
/*    Mask on intrcontroller.                                                  */
/*******************************************************************************/

      b = inportb(contadr);
      if ((b && mask1) == 0)
         *AlreadyEnabled = TRUE;
      else
         *AlreadyEnabled = FALSE;

      if (IntrEnable) {
         b = b & mask2;
         disable();
         outportb(contadr,b);
         enable();
      } else {
         b = b | mask1;
         disable();
         outportb(contadr,b);
         enable();
      }
   }
}

/*******************************************************************************/
/* Comxx: Reads the incomming char and puts it in buffer BufAr.                */
/*******************************************************************************/

void interrupt Comxx(__CPPARGS)
{
```

```
   unsigned char b0,b1;

/*****************************************************************************/
/*    UART linestatus register:                                              */
/*      bit 0: DATA READY                                                    */
/*         1 if incomming char ready                                         */
/*                                                                           */
/*      bit 1: OVERRUN ERROR                                                 */
/*         1 if overrun on incomming chars                                   */
/*                                                                           */
/*      bit 2: PARITY ERROR                                                  */
/*         1 if parity error (no valid parity bit acc to uart setup)         */
/*                                                                           */
/*      bit 3: FRAMING ERROR                                                 */
/*         1 if framing error (no valid stopbit in received char)            */
/*                                                                           */
/*      bit 4: BREAK INTERRUPT                                               */
/*         1 if break interrupt (if inportbut pin is hold in logic 0 state   */
/*                        for longer than receive time of a char)            */
/*                                                                           */
/*      bit 5: TRANSMITTER HOLDING REGISTER EMPTY                            */
/*         1 if transmit register is empty (so you can send one more)        */
/*                                                                           */
/*      bit 6: TRANSMITTER EMPTY                                             */
/*         1 if transmit register AND shiftregister is empty                 */
/*                                                                           */
/*      bit 7: not us'd                                                      */
/*****************************************************************************/

   b0=LineStatusRegister = inportb(LineStatusReg);

/*****************************************************************************/
/*    Char in UART.                                                          */
/*****************************************************************************/

   if ((b0 & 0x01))
      b1 = inportb(RXBuf);

/*****************************************************************************/
/*    Reset intr controller.                                                 */
/*****************************************************************************/

   outportb(IntrController,EOI);

   if (!(b0 & 0x1E))
   {

/*****************************************************************************/
/*    OK: no framing parity etc error.                                       */
/*    put char in buf if space.                                              */
/*****************************************************************************/

       if (b1=='\r') CRreceivedInRS232++;
       if (BytesInBuffer < BufSize)
        {
         BufAr[BufferWriterPointer] = b1;
         if (BufSize <= ++BufferWriterPointer) BufferWriterPointer = 0;
         BytesInBuffer++;
         }
   }
}

/*****************************************************************************/
/* ChInBuffer: return number of char in buffer BufAr.                        */
/*****************************************************************************/

int ChInBuffer()
{
  return (BytesInBuffer);
}

/*****************************************************************************/
```

```
/* CRInBuffer: return number of \r char in buffer BufAr.                      */
/*****************************************************************************/

int CRInBuffer()
{
  return (CRreceivedInRS232);
}

/*****************************************************************************/
/* ReadCom: Gets char from interrupt buffer.                                 */
/*****************************************************************************/

Err ReadCom(char *dest)
{
   *dest= BufAr[BufferReaderPointer];
   if ( BufSize <= (++BufferReaderPointer)) BufferReaderPointer = 0;
   disable();
   BytesInBuffer--;
   CRreceivedInRS232=0;         /* clear \r counter. */
   enable();
   return 0;
}

/*****************************************************************************/
/* ReadStrCom: Get string from interrupt buffer.                             */
/*****************************************************************************/

void ReadStrCom(char *dest, int lgt)
{
   unsigned int index;

   for (index = 0; index < (unsigned int)lgt; index++)
   {
      if (BytesInBuffer)
      {
       dest[index] = BufAr[BufferReaderPointer];
       if ( BufSize <= (++BufferReaderPointer)) BufferReaderPointer = 0;
       disable();BytesInBuffer--;enable();
      }
      else break;
   }
   dest[index]=0;
   CRreceivedInRS232=0;         /* clear \r counter. */

}

/*****************************************************************************/
/* ReadUART: Gets char from UART. If no chars waiting, user waits here until  */
/* char has been read but time_out.                                          */
/*****************************************************************************/

Err ReadUART(char *dest)
{
unsigned char b;
long timer=MAXTIME;
do
{
 b=inportb(LineStatusReg);
 LineStatusRegister |=b;
} while ((!(0x01&b)) && (timer--));
if (timer <=0) return 10;
*dest=inportb(RXBuf);
return 0;
}

Err WriteStrCom(const char *s, int lgt)
{
   unsigned int Index;
   unsigned char b;
   long int timer=MAXTIME;

   for (Index=0; Index < (unsigned int)lgt; Index++)
```

```
   {
      timer=MAXTIME;
      do
      {
         b = inportb(LineStatusReg);
         LineStatusRegister = (b | LineStatusRegister);

/*****************************************************************************/
/*       Poll on ready uart.                                                 */
/*****************************************************************************/

       } while ((!(0x20 & b)) && (timer--));
       if (timer <=0) return 10;
      outportb(TXBuf, s[Index]);
      delay(40);
   }
   return 0;
}

Err WriteCom(unsigned char b_out)
{
   unsigned char b;
   long timer=MAXTIME;
   do
    {
      b = inportb(LineStatusReg);
      LineStatusRegister = (b | LineStatusRegister);
    } while ((!(0x20 & b)) && (timer--));
   if (timer <=0) return 10;
   outportb(TXBuf, b_out);
   delay(40);
   return 0;
}

void ClearComInBuf(void)
{
   disable();
   BufferReaderPointer = BufferWriterPointer = BytesInBuffer = 0;
   CRreceivedInRS232=0;         /* clear \r counter.   */
   enable();
}

/*****************************************************************************/
/* Init the 8250 communication controller to the specific baudrate ...       */
/*                                                                           */
/*   baudrate : 300,600,1200,2400,4800,9600,19200,38400(1),57600(2),115200(3) */
/*   nrofdatabits: 5,6,7,8                                                   */
/*   nrofstopbits: 1,2                                                       */
/*   parity :      false,true                                                */
/*   evenparity:   false,true ( if parity!)                                  */
/*   spacing :     false,true (true if space wanted btw char when sending)   */
/*   NrOF100mikrosec: nr of 100 mikrosec space btw char if spacing true      */
/*                                                                           */
/* Returns with a boolean that indicate well init or not...                  */
/*****************************************************************************/

Err InitCom(int baudrate, int nrbits, int nrofstopbits,
               int parity, int evenparity)

{
unsigned char msb,lsb,linecontrol;

/*****************************************************************************/
/* Line Control Register:                                                    */
/*   bit 0:                                                                  */
/*       1: word length 00: 5 bit                                            */
/*                      01: 6 bit                                            */
/*                      10: 7 bit                                            */
/*                      11: 8 bit                                            */
/*         2: stop bit     0: 1 stop bit                                     */
/*                         1: 2 stop bit  (1.5 stop bit if 5 bit wordlength) */
/*         3: parity enable if 1                                             */
```

```
/*         4: even parity if 1. odd parity if 0                                */
/*         5: stick parity default 0                                           */
/*         6: break control. if 0 rs232 outportbut is forced to 0. default 0. */
/*         7: divisor latch access bit. Must be 1 for r/w to divisor registers*/
/*                                                                             */
/* Baudrate is calculated as XTAL-freq / (msb<<8)+lsb (115kHz)                 */
/*******************************************************************************/

   msb = 0;

   switch (baudrate)
   {
      case 300:msb = 1;lsb = 0x80;break;
      case 600:lsb = 0xc0;break;
      case 1200:lsb = 0x60;break;
      case 2400:lsb = 0x30;break;
      case 4800:lsb = 0x18;break;
      case 9600:lsb = 0x0c;break;
      case 19200:lsb = 0x06;break;
      case 1:lsb = 0x03;break;
      case 2:lsb = 0x02;break;
      case 3:lsb = 0x01;break;
      default:return 10;
   }

      if (nrbits >= 5 && nrbits <= 8) linecontrol = nrbits - 5;
      else return 10;

      switch (nrofstopbits)
      {
         case 1: break;
         case 2: linecontrol = (linecontrol | 0x04);break;
         default: return 10;
      }

      if (parity)
      {
         linecontrol = (linecontrol | 0x08);
         if (evenparity) linecontrol = (linecontrol | 0x10);
      }

/*******************************************************************************/
/*    Do baudrate programming                                                  */
/*******************************************************************************/

      outportb(LineControlReg,0x80);
      outportb(DivLatchMSB,msb);
      outportb(DivLatchLSB,lsb);

/*******************************************************************************/
/*    Do word length etc. programming                                          */
/*******************************************************************************/

      outportb(LineControlReg,linecontrol);

   return 0;
}

void StartCom(void)
{
   unsigned char  b;

   if (!ComActive)
   {
      LineStatusRegister = 0;
      disable();
      pOldComIntr = getvect(IntrComxx);
      setvect(IntrComxx,Comxx);
      enable();
      b = inportb(LineStatusReg);

      if ((b & 1) == 1)
```

```
         b = inportb(RXBuf);

      b = 0x08;
      b = b | 0x01;
      b = b | 0x02;

      OldModemControlReg = (unsigned char)inportb(ModemControlReg);

      outportb(ModemControlReg,b);

      outportb(IntrEnableReg,0x04 + 0x01);

      MaskHWInterrupt(ComIntrNr,TRUE,&AlreadyEnabled);
      disable();
      BufferReaderPointer =BufferWriterPointer = 0;
      BytesInBuffer = 0;
      CRreceivedInRS232=0; /*clear \r counter*/
      ComActive = TRUE;
      puts("Com port started");
      enable();
   }
}


void StopCom()
{
   int b1;
   if (ComActive)
   {
      ComActive = 0;b1=IntrComxx;
      disable();
      outportb(IntrEnableReg,OldModemControlReg);
      enable();
      setvect(IntrComxx,pOldComIntr);
      MaskHWInterrupt(ComIntrNr,AlreadyEnabled,&b1);
      printf("Com port stopped.");
   }
}
```

F.6 ReadRS232

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

extern void ReadRS232(void);
extern int ChInBuffer();
extern int CRInBuffer();
extern int ReadCom(char*);
extern void EkgPlot(int*);
extern void PulsPlot(int);
extern void Alarm(int);
extern char *int_char(int);
extern int Testsignal;
extern FILE *TestFile;

void ReadRS232(void)

/*************************************************************************/
/* Denne funktion kontrollerer om der er modtaget data fra RS232.        */
/* Hvis der er modtaget data, dekodes disse og sendes til den funktion   */
/* som data refererer til.                                               */
/* p og q er lokale pointere, som sættes til at pege på starten af frame[]  */
/*************************************************************************/

{
   unsigned char frame[18],*I;
   unsigned char *p,*q;
   int SendArray[11]={0};
   int FrameOK=0,NotFinish=1,x;
   p=q=frame;
```

```
/*****************************************************************************/
/* Hvis det ikke er testsignalet, der vises, og der er en streng i bufferen  */
/* (kontrolleres med CRInBuffer()), læses karakterer med ReadCom(), indtil   */
/* der findes en '#'.                                                        */
/* p sættes herefter til at pege på '#', og der refereres nu ud fra dette    */
/* punkt. (p+1),(p+2) osv. For at sikre, at der ikke testes på en gammel     */
/* værdi i frame sættes pladsen efter '#' til \0.                            */
/* Den næste karakter læses herefter ind, og der kontrolleres for en gyldig  */
/* kommando.                                                                 */
/* q opdateres for hver læst karakter, således at der læses en karakter ind i */
/* hver plads i frame[], indtil '\n\r' findes, eller der ikke er flere       */
/* karakterer i bufferen.                                                    */
/*****************************************************************************/

   if (Testsignal==0)
   {
      if (CRInBuffer())
      {
         while (ChInBuffer())
         {
            ReadCom(q);
            if((*q=='#')&&(NotFinish))
            {
               p=q;
               *(p+1)=0;
               NotFinish=0;
            }
            if(((*p)=='#')&&(((*(p+1))=='E')||((*(p+1))=='A')||((*(p+1))=='P')))
            {
               FrameOK=1;
            }
            if ((FrameOK)&&(*q=='\r')&&(*(q-1)='\n'))
            {
               *(q-1)=0;
               NotFinish=1;
               break;
            }
            q++;
         }
      }

/*****************************************************************************/
/* De modtagne frames dekodes og sendes til de respektive funktioner.        */
/* Hvis kommandoen er 'E' og der er en '$' på plads 12, er der modtaget en   */
/* frame med Ekgsampels. Start og slut karakterer fjernes og pakken sendes   */
/* til EkgPlot.                                                              */
/* Tilsvarende kaldes henholdsvis Alarm og PulsPlot, hvis kommandoen er      */
/* henholdsvis 'A' og 'P'.                                                   */
/*****************************************************************************/

      if (((*(p+1))=='E') && ((*(p+12))=='$'))
         {
            *(p+12)=0;
            for (x=2;x<=12;x++)
            {
               SendArray[x-2]=*(p+x);
            }
            EkgPlot(SendArray);
         }
      if (((*(p+1))=='A') && ((*(p+3))=='$'))
         Alarm(*(p+2));
      if (((*(p+1))=='P') && ((*(p+3))=='$'))
         PulsPlot(*(p+2));
      *(p+1)=0;
   }

/*****************************************************************************/
/* Hvis det er Testsignalet, der skal vises, hentes 10 sampels fra en fil, og */
/* disse sendes til EkgPlot.                                                 */
/* Når EOF i filen nås, startes på ny fra filens start, således at signalet  */
/* aldrig går i stå.                                                         */
```

```
/*****************************************************************************/

   else
   {
      for (x=0;x<=10;x++)
      {
         I=fgets(frame,5,TestFile);
         if (I == NULL) fseek(TestFile, 0, SEEK_SET);
         SendArray[x]=atoi(frame);
      }
      EkgPlot(SendArray);
      delay(150);
   }
}
```

F.7 EkgPlot

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

#define black 0
#define green 2
#define red 4
#define grey 7
#define blue 9
#define yellow 14
#define white 15

extern char *int_char(int);
extern int XEkg, YLast, Hold, Testsignal;

void EkgPlot(int EkgArray[])

/*****************************************************************************/
/* Funktionen gennemløbes kun, hvis der ikke er valgt hold af signal.        */
/*****************************************************************************/

{
   if (!Hold)
   {
      int EkgNow;               /* Aktuel EKG-amplitude.                    */
      int YPlot;                /* Den tilpassede værdi af pulsen til trend. */
      int SampleNr=0;           /* Angiver aktivt sample i EkgArray.        */

/*****************************************************************************/
/* Funktionen gennemløbes 10 gange, en for hvert sample i EkgArray[].        */
/* Der foretages en tilpasning af samplet, således at det plottes på den     */
/* rigtige plads                                                             */
/*****************************************************************************/


      while (SampleNr < 10)
      {
         XEkg++;
         EkgNow = EkgArray[SampleNr];
         YPlot = (307 - EkgNow);

/*****************************************************************************/
/* Hvis EkgPlottet er nået til højre side af skærmen, startes i venstre side. */
/*****************************************************************************/

         if ((XEkg == 639) || (XEkg == 1))
         {
            XEkg = 1;
            setcolor(black);
            line(XEkg,51,XEkg,309);

/*****************************************************************************/
/* Hvis der er tale om et testsignal fra disk plottes grafen med gult, og    */
/* hvis det er et signal fra en patient, plottes det med grønt.              */
```

```
/*****************************************************************************/

            if (Testsignal)
            {
               setcolor(yellow);
               putpixel(XEkg,YPlot,green);
            }
            else
            {
               setcolor(green);
               putpixel(XEkg,YPlot,green);
            }
         }

/*****************************************************************************/
/* Hvis EkgPlottet ikke er nået til højre side af skærmen, tegnes en streg    */
/* fra sidste punkt og til aktuelle punkt. Hvis der er tale om et testsignal, */
/* tegnes med gult og ellers med grønt.                                       */
/*****************************************************************************/

         else
         {
            setcolor (black);
            line(XEkg,51,XEkg,309);
            if (Testsignal)
               setcolor(yellow);
            else
               setcolor(green);
            line(XEkg-1,YLast,XEkg,YPlot);
         }
         SampleNr++;
         YLast=YPlot;
      }
   }
}
```

F.8 PulsPlot

```
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <dos.h>

#define black 0
#define green 2
#define red 4
#define grey 7
#define blue 9
#define yellow 14
#define white 15

int YPlot;

extern char *int_char(int);
extern int XTrend;
extern void PulsPlot(int);
extern float TrendTime;
extern FILE *Trend;
extern struct time TimeNow;

void PulsPlot(int PulsNow)

/*****************************************************************************/
/* Funktionen plotter trend og skriver den aktuelle puls på skærmen og        */
/* gemmer pulsen i filen trend.log en gang i minuttet.                        */
/*****************************************************************************/

{
   int TrendInterval=1, TrendLength=((24*60)/TrendInterval);

/*****************************************************************************/
```

```
/* Sletter og skriver den aktuelle puls på skærmen.                          */
/****************************************************************************/

   setviewport(40,35,76,45,1);
   clearviewport();
   setviewport(0,0,639,479,1);
   setcolor(green);
   outtextxy(40,35,int_char(PulsNow));

/****************************************************************************/
/* Tilpasser y-værdi og plotter pulsværdi.                                  */
/****************************************************************************/

   XTrend++;
   if (XTrend == 638)
      XTrend = 1;
   YPlot = 427 - PulsNow / 2.2;
   setcolor(black);
   line(XTrend,331,XTrend,429);
   setcolor(green);
   putpixel(XTrend,YPlot,green);

/****************************************************************************/
/* Henter PC´ens systemtid, kontrollerer om tiden siden sidste skrivning til  */
/* fil er gået, og hvis dette er tilfældet, skrives til filen.              */
/* Filen er opbygget som en ringbuffer således at den ikke bare vokser, men   */
/* altid indeholder trend fra det sidste døgn.                              */
/****************************************************************************/

   gettime(&TimeNow);
   if (TimeNow.ti_min >= TrendTime)
   {
      if (ftell(Trend) == TrendLength) /* Disse to linier bruges til at styre */
         fseek(Trend,0,0);             /* ringbufferen i trendfilen.          */
      fputc(PulsNow,Trend);            /* Gemmer en værdi i trendfilen.       */
      TrendTime = TrendTime + TrendInterval; /* Styrer TrendIntervallet.      */
   }
}
```

F.9 Alarm

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

#define black 0
#define green 2
#define red 4
#define grey 7
#define blue 9
#define yellow 14
#define white 15

extern void Alarm(int);

void Alarm (int AlarmOnOff)

/****************************************************************************/
/* Hvis alarmen er høj, optegner programmet en rød streg omkring EKG-signalet */
/* og der udsendes en 1 kHz vedvarende tone. Tilsvarende tegnes stregen med   */
/* hvidt og lyden stoppes, hvis alarmen er lav.                             */
/****************************************************************************/

{
  if (AlarmOnOff)
   {
      sound(1000);
      setcolor(red);
      line(0,50,639,50);
      line(0,310,639,310);
      line(0,50,0,310);
```

```
      line(639,50,639,310);
   }
  else
   {
      nosound();
      setcolor(white);
      line(0,50,639,50);
      line(0,310,639,310);
      line(0,50,0,310);
      line(639,50,639,310);
   }
}
```

F.10 Testprogram

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <bios.h>
#include "RS232.h"

extern int InitCom(int,int,int,int,int);
extern void  StartCom();
extern void  ClearComInBuf();
extern void  StopCom();
extern int ChInBuffer();
extern int CRInBuffer();
extern int WriteStrCom(const char *,int);
extern int ReadCom(char*);


void InitRS232(void)

/*****************************************************************************/
/* Initialisering af RS232                                                   */
/*****************************************************************************/

{
   InitCom(19200,8,1,FALSE,0);
   StartCom();
   ClearComInBuf();
}

void GetFrame(void)

/*****************************************************************************/
/* Denne funktion kontrollerer om der er modtaget data fra RS232.            */
/* Hvis der er modtaget data, dekodes disse og sendes til den funktion       */
/* som data refererer til.                                                   */
/* p og q er lokale pointere, som sættes til at pege på starten af frame[]   */
/*****************************************************************************/

{
   int FrameOK=0;
   unsigned char frame[50];
   unsigned char *p,*q;
   int found=0,ch=0,x;
   p=q=frame;

/*****************************************************************************/
/* Hvis der er en streng i bufferen (kontrolleres med CRInBuffer()), læses   */
/* karakterer med ReadCom(), indtil der findes en '#'.                       */
/* p sættes herefter til at pege på '#', og der refereres nu ud fra dette    */
/* punkt. (p+1),(p+2) osv. For at sikre, at der ikke testes på en gammel     */
/* værdi i frame sættes pladsen efter '#' til \0.                            */
/* Den næste karakter læses herefter ind, og der kontrolleres for en gyldig  */
/* kommando.                                                                 */
/* q opdateres for hver læst karakter, således at der læses en karakter ind i */
/* hver plads i frame[], indtil '\n\r' findes, eller der ikke er flere       */
/* karakterer i bufferen.                                                    */
```

```
/*****************************************************************************/

      if (CRInBuffer())                /* Kontrollerer om der er en frame i   */
      {                                /* bufferen.                           */
         while (ChInBuffer())          /* Denne løkke eksekveres så længe der */
         {                             /* er tegn i bufferen.                 */
            ReadCom(q);                /* Der læses en karakter fra bufferen. */
            if(*q=='#')
            {                          /* p peger på det første '#' i frame.  */
               p=q;                    /* q flyttes for hver læst karakter.   */
               *(p+1)=0;
            }
            if(((*p)=='#')&&(((*(p+1))=='N')||((*(p+1))=='X')||((*(p+1))=='V')))
            {                          /* Der kontrolleres for en korrekt     */
               FrameOK=1;              /* frame.                              */
            }
            if ((FrameOK)&&(*q=='\r')&&(*(q-1)='\n'))
            {                           /* Der læses karakterer indtil '\n\r' */
               *(q-1)=0;                /* findes i framen, hvorefter der     */
               break;                   /* springes ud af løkken.             */
            }
            q++;
         }
      }

/*****************************************************************************/
/* Framen dekodes og der skrives, hvilken frame det var, og hvad værdien var. */
/*****************************************************************************/

   *(p+3)=0;
   if ((*(p+1))=='N') printf("Min puls: %i\n",*(p+2));
   if ((*(p+1))=='X') printf("Max puls: %i\n",*(p+2));
   if ((*(p+1))=='V') printf("Patient: %i\n",*(p+2));
   *(p+1)=0;
}

void SendFrame(char c,int p[10])

/*****************************************************************************/
/* Sender Ekg-sampels til PC.                                                */
/*****************************************************************************/

{
   unsigned char str[16];                /* Ekg-sampels gøres klar til      */

   str[0]='#';                           /* afsendelse.                     */
   str[1]=c;
   str[2]=p[0];
   str[3]=p[1];
   str[4]=p[2];
   str[5]=p[3];
   str[6]=p[4];
   str[7]=p[5];
   str[8]=p[6];
   str[9]=p[7];
   str[10]=p[8];
   str[11]=p[9];
   str[12]='$';
   str[13]='\n';
   str[14]='\r';
   str[15]=0;
   ClearComInBuf();                      /* Tømmer bufferen                 */
   WriteStrCom(str,16);                  /* Sender data.                    */
   printf("%s \n",str);                  /* skriver afsendte data på skærm. */
}

void SendCom (char c,int p)

/*****************************************************************************/
/* Sender kommando til PC. (Puls eller Alarm)                                */
/*****************************************************************************/
```

```
{
   unsigned char str[16];                  /* Kommando gøres klar til         */
   str[0]='#';                             /* afsendelse.                     */
   str[1]=c;
   str[2]=p;
   str[3]='$';
   str[4]='\n';
   str[5]='\r';
   str[6]=0;
   ClearComInBuf();                        /* Tømmer bufferen                 */
   WriteStrCom(str,7);                     /* Sender data.                    */
   printf("%s \n",str);                    /* skriver afsendte data på skærm. */
}

void main(void)

/*************************************************************************/
/* Hovedprogrammet åbner en fil med Ekg-signal, henter 10 sampels og sender   */
/* disse til PC. Hvis brugeren trykker på 'A', 'a', 'P' eller 'p' sendes      */
/* henholdsvis Alarm on, Alarm off, Puls=60 og Puls=80.                       */
/* Trykker brugeren på 'q' stopper programmet.                                */
/*************************************************************************/

{
   FILE *InFile;
   char key=0,C='E',I[20];
   int P,ekg[10],n;
   InFile = fopen("EkgTest.ekg", "r");     /* Åbner fil med Ekg-signal.       */
   fseek(InFile, 0 , SEEK_SET);
   InitRS232();                            /* Initialiserer RS232             */
   system("cls");
   while (key!='q')
   {
      for(n=0;n<=9;n++)                    /* Henter 10 sampels.              */
      {
         ekg[n]=atoi(fgets(I, 10, InFile));
      }
      GetFrame();                          /* Checker om der er data fra PC.  */
      SendFrame(C,ekg);                    /* Sender Ekg-sampels til PC.      */
      if (bioskey(1))
      {
         key=bioskey(0);
         switch (key)
         {
            case 'A' : SendCom('A',1);break;  /* Send Alarm On              */
            case 'a' : SendCom('A',0);break;  /* Send Alarm Off             */
            case 'P' : SendCom('P',60);break; /* Send Puls=60               */
            case 'p' : SendCom('P',80);break; /* Send Puls=80               */
         }
      }
   }
   fclose(InFile);                         /* lukker fil med Ekg-signal.      */
   StopCom();                              /* Stopper Com-porten.             */
   system("cls");
}
```

G Brugervejledning

Denne brugervejledning omhandler installation og betjening af EKG-2000. Først står en liste over de ting, som følger med til systemet. Dernæst angives kravene til den PC, som programmet skal køre på.
Der afsluttes med en brugsvejledning til programmet, som beskriver, hvordan det installeres og anvendes.

G.1 Systemets dele

EKG-2000 består af følgende fysiske enheder:

- Instrumentforstærker.
- RS232C serielkabel.
- Motorola VMPM KA-2 kort + Dataopsamlingsenhed.
- Et sæt elektroder.
- Spændingsforsyning.
- CD-ROM med programmer.
- 2 stk. 9V batterier.

G.2 Systemkrav

For at anvende EKG-2000 kræves en PC med disse specifikationer:

- IBM kompatibel PC, af typen 80286 eller højere.
- 3.5" diskettedrev.
- Mindst 200 kB fri harddiskplads.
- Mindst een ledig COM-port.
- Operativsystemet DOS.

G.3 Installation af software

På den medfølgende CD-ROM ligger den software, som er nødvendig for at få EKG-2000 til at virke. For at installere programmet, sættes CD-ROM'en i PC'ens CD-ROM-drev, og filen "install.bat" eksekveres.
Som default installeres programmet i stien "c:\ekg-2000\". Ønskes en anden placering, kan installationsfilen køres med denne sti som parameter. Denne sti skal formateres på følgende måde:

install <drev:\bibliotek\[bibliotek\]>

G.4 Brugervejledning til programmet

For at starte programmet, eksekveres filen “ekg-2000.exe”.

Derefter starter programmet op, og følgende skærmbillede vises:

Som det ses på figuren, har følgende taster funktionerne:

n: Indtastning af patientens navn.
c: Indtastning af patientens CPR-nr.
i: Indtastning af patientens minimale puls før alarm.
a: Indtastning af patientens maksimale puls før alarm.
h: Frys/start EKG-billede. (Toggle)
t: EKG-testsignal startes/stoppes. (Toggle)
1: Aktivér/Deaktivér patient. (Toggle)
q: Afslut program.

Hvis en patient er aktiveret, er den pågældende patients tekst hvid. Ellers er den grå.
Hvad angår EKG-signalets farve, gælder følgende:

- Det rigtige signal er grønt.
- Testsignalet er gult.

Programmet er indrettet således, at der for indtastning gælder:

- Navn kan maksimalt fylde 23 karakterer. Afsluttes ved at trykke på <Enter>.
- CPR.nr. fylder 10 cifre og afsluttes automatisk, når dette antal nås.
- Min. og max. puls indtastes og afsluttes ved at trykke på <Enter> eller automatisk når det tredje ciffer indtastes.
- Ved indtastning af min. puls skal den være mindre end max. puls, ellers accepteres indtastningen ikke, og indtastningsprompten starter forfra.
- Tilsvarende gælder for max. puls.

Pulstrenden gemmes i tekstfilen “trend.log”.

Når programmet afsluttes korrekt fader billedet ud.

H Symbolliste

I dette appendiks forklares de forskellige symboler, forkortelser og betegnelser, som er anvendt i projektet. De er opdelt efter deres placering i rapporten og derunder i alfabetisk orden.
Dette appendiks er beregnet til opslag, og skal derfor ikke læses som en fortløbende tekst.

H.1 Dataopsamlingsenhed

A_DTACK*	Signal fra adressedekoderen når DTACK* kan afsendes.
A_LDS*	Lower data strobe fra adressedekoder til IRQ-dekoder.
A_SIG0 - 7	Analoge indgangssignaler fra instrumentforstærker til ADC/MUX.
ABEL	Advanced Boolean Expression Language.
ADC	Analog til Digital Converter.
ADC_CLK	Startsignal til ADC'en på 4 kHz.
ADC_ENA	Aktiverer data på udgang af ADC.
ALARM0 - 7	Chip select til hver af alarmerne på dataopsamlingsenheden.
AS*	Address Strobe.
CON_ENA*	Chip select til kontrolregisteret.
CONV_CLK	Konverteringsclock til ADC'en på 1 MHz.
COUNT_CLK	Tællersignal, på 4 kHz, til styring af multiplexer.
D_{00} - D_{15}	Databus.
DS0*	Data Strobe 0.
DS1*	Data Strobe 1.
DTACK*	Data Transfer ACKnowledge.
EOC	End Of Convertion.
I_DTACK*	Signal fra IRQ dekoderen som bekræfter at DTACK* kan afsendes.
IRQ	Interrupt ReQuest.
LED0 - 7	Signal til de 8 alarm-LED'ere.
MUX	Multiplexer.
PAL	Programmable Array Logic.
PEEL	Programmable Electrically Erasable Logic.
PID0 - 3	Encodet Patient Identifikation til statusregister.
PLD	Programmable Logical Device.
PS0 - 7	Patient select 0 - 7 til IRQ-generator.
Speaker	Signal til piezo-elektrisk højttaler.
STA_ENA*	Chip select til statusregister.
SYS_CLK	System clock.
UDS*	Upper Data Strobe til VME interface.

H.2 PC-program

1 “tastetryk”	Tilkobler/frakobler patient nr. 1. Virker som toggle.
a “tastetryk”	Aktiverer en prompt, hvorved brugeren kan indtaste max. puls.
Alarm(1)	Aktivering af alarmen på PC.
Alarm(0)	Deaktiverer af alarmen på PC.
Alarm(“værdi”)	Kalder modulet Alarm med “værdi”.
c “tastetryk”	Aktiverer en prompt, hvorved brugeren kan indtaste CPR-nr.
ChInBuffer	Returnerer antallet af tegn i UART-buffer.
ClearComInBuf()	Sletter ringbufferen i RS232-driveren.
CRInBuffer	Returnerer antallet af “\r” i UART-buffer.
DrawBase	Denne funktion optegner grundskærmen ved opstart.
EkgPlot(“10 samples”)	Funktionen kaldes med et array, som indeholder 10 samples.
FadeOut	Får skærmbilledet til at fade ud.
h “tastetryk”	Fryser signalet. Virker som toggle.
Hold	Global variabel, som angiver, om skærmbilledet skal fryses.
i “tastetryk”	Aktiverer en prompt, hvorved brugeren kan indtaste min. puls.
Init(1)	Et funktionskald med denne værdi angiver, at der skal initialiseres.
Init(0)	Et funktionskald med denne værdi angiver at de initialiserede funktioner skal afbrydes.
InitCom()	Initialiserer UART’ens portindstillinger.
InputName()	Indtastning af navn.
InputCpr()	Indtastning af CPR.nr.
Min_Puls()	Indtastning af min. puls.
Max_puls()	Indtastning af max. puls.
n (tastetryk)	Aktiverer en prompt, hvorved brugeren kan indtaste navn.
Patient_On_Off()	Aktiverer/deaktiverer patient nr.1.
PlotPuls(“Aktuel Puls”)	Funktion der kaldes med en variabel, som angiver den aktuelle puls.
ReadCom()	Funktion der læser en byte i bufferen.
ReadRS232()	Kontrollerer om der er data fra RS232C.
SendFrame()	Varetager afsendelsen af frames til modulet RS232.
StartCom()	Nulstiller og starter UART.
StopCom()	Stopper UART.
t “tastetryk”	Skifter mellem visning af det rigtige signal og testsignal.
Testsignal	Global variabel.
TimeNow	En struct som angiver det aktuelle tidspunkt.
UART-chip	Sender serielle signaler til mikroprocessoren.
UserInput	Udfører inputrutiner afhængig af tastetryk.
WriteStrCom()	Skriver til UART.

H.3 M68k program

AlarmON_OFF(1)	Starter og stopper alarmen.
APP_INIT	Modul, som indeholder funktionen Initialize.
APP_DECOD	Modul, som indeholder funktionerne Data from PC, Test data, Decode Min/Max. puls og Set patient.
APP_ENCODE	Modul, som indeholder funktionen Encode og Encode EKG.
APP_EKGREDUC	Modul, som indeholder funktionen EKGReduction.
APP_PULSCALC	Modul, som indeholder funktionen PulsCalculation.

APP_MAIN	Modul, som indeholder mainløkken, som kalder de øvrige funktioner.
DataFromPc( )	Kaldes fra Main for at undersøge, om der er kommet data fra PC.
Decode(1)	Dekoder datastrengen og aktiverer en passende funktion.
EKGReduction	Funktionen EKGReduction kaldes, indeholdende det fulde EKG.
EKGReduction(1)	Sender VmeDatain videre til EKGReduction.
EncodeEKG	Encoder det reducerede EKG-signal.
EncodeEKG(2)	Kaldes med to parametre; "E" samt en pointer til array med samples.
EnCode(2)	Sender to parametre videre til EnCode enten start/stop alarm eller pulsen.
Init( )	Bliver kaldt fra Main, for at initialisere programmet.
InitBuffer( )	Initialiserer VMEbuffer, dvs. ReadPointer og WritePointer.
Initielize()	Funktion som initialiserer operativsystemet.
IRQPuls	Afgives når en AD-konvertering er færdig.
M68k	Motorola 68000 mikroprocessor.
MaskIrq( )	Masker IRQ ud, mens der initialiseres og læses i bufferen.
MaxPuls(1)	Sætter max. puls til værdien der er modtaget fra PC.
MinPuls(1)	Sætter min. puls til værdien der er modtaget fra PC.
OS_INIT:	Modul, som indeholder funktionerne Initialize, Warmstart og Coldstart.
OS_MASKIRQ	Modul som indeholder funktionerne Mask IRQ og Unmask IRQ.
OS_DAOE	Modul som indeholder funktionerne Start alarm, Stop alarm og Set patient.
OS_RS232	Modul som indeholder funktionerne UART test, Read RS232 og Write RS232.
OS_BUFFER	Modul som indeholder funktionerne InitBuffer, ReadBuffer og WriteBuffer.
OS_IRQROUTINE	Modul som indeholder funktionen IRQ-routine.
PatientON_OFF	Til- og frakobling af patienter.
PatientON_OFF(1)	Aktiverer eller deaktiverer en patient.
PulsCalculation( )	Beregner pulsen.
ReadBuffer	Læser fra bufferen og forøger ReadPointer til næste læsning.
ReadBuffer(2)	Læser i bufferen og kaldes med to parametre VmeDataIn og Buffer-UnderRun.
ReadRS232	Læser fra RS232-driveren.
ReadRs232(2)	Kaldes med to parametre; længden og en pointer til datastrengen.
UARTTest( )	Returnerer 1 hvis der er data i UART.
UnMaskIrq( )	Udmasker IRQ'en efter at der er læst i bufferen.
WriteBuffer	Skriver til bufferen og forøger WritePointer til næste skrivning.
WriteBuffer(1)	Skriver til ringbufferen.
WriteRS232	Skriver til RS232-driveren.
WriteRS232(2)	Kaldes med parametrene: længden på arrayet samt en pointer til dette.

I Diagram over dataopsamlingsenhed

I dette appendiks findes diagrammer, som viser hvordan data-opsamlingenheden er opbygget. På denne side er en oversigt over komponentplaceringen. På A3-siderne herefter følger et diagram over modulerne, samt forbindelserne på dataopsamlingsenheden.