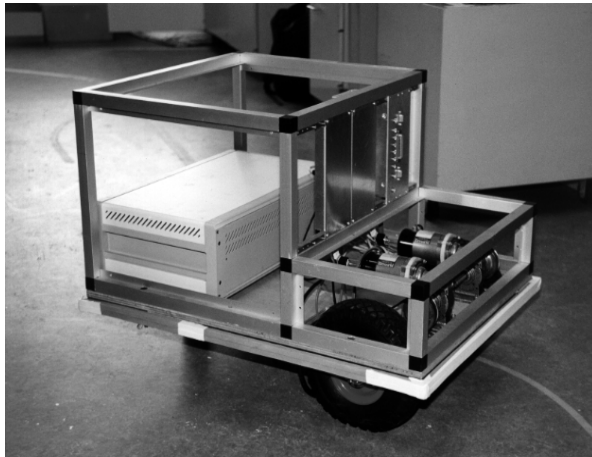


Praktisk AGV konstruktion og sensorintegration



Speciale i datateknologi, af:

Anders Stengaard Sørensen

16. november 1998

Vejledere:

Edmund Christiansen
Institut for Matematik og Datalogi
Odense Universitet

Jørn Fly Hansen
Det elektrotekniske retningsområde
Ingeniørhøjskolen Odense Teknikum



Indhold

Forord	ix
Om uddannelsen i Datateknologi	ix
Om AGV-teknologi på Odense Universitet	ix
Tak til	x
 I Rapportdel	 1
 1 Indledning	 3
1.1 Forhistorie	3
1.2 Projektforløb	4
1.3 Rapporten	5
 2 AGV-teknologi — En emneoversigt	 7
2.1 AGV'er	7
2.2 Navigation	8
2.3 Sensorer	10
 3 Overordnet koncept	 13
3.1 De forhåndenværende ”søm“	14
3.2 Datamat	14
3.3 Mekanisk skelet	15
3.4 Motorer og motorstyring	15
3.5 Strømforsyning	16
3.6 Sensorer	16
3.7 Opsummering	18

4	Motorstyring	21
4.1	Motorer	21
4.2	Regulering	22
4.3	Feedback	23
4.4	Motordriver	24
5	VME-bus slave moduler	31
5.1	VME-bussen	31
5.2	Overordnede designbeslutninger	32
5.3	Overordnet struktur	32
5.4	Gefion	33
5.5	Heimdal	36
5.6	Implementation	37
5.7	Afprøvning	38
5.8	Testresultater	39
5.9	Konklusion	41
6	VME-modul til motorstyring	43
6.1	LM629	43
6.2	Testopstilling	45
6.3	Struktur af VME-modul	46
6.4	Implementation	48
6.5	Funktionsbibliotek	50
6.6	Afprøvning	51
6.7	Konklusion	53
7	VME sonar- interface	55
7.1	POLAROID's sonar-ranging system	55
7.2	Struktur af Heimdal	57
7.3	Implementation	60
7.4	Afprøvning	62
7.5	Konklusion	63
8	Udvikling af software-interface til Heimdal	65
8.1	Indledende overvejelser om SW-interface	65
8.2	Struktur af SW-interface	66
8.3	Integration med OS-9	67
8.4	Design	68

8.5	Implementation	69
8.6	Afprøvning	74
8.7	Dokumentation	78
8.8	Konklusion	78
9	Liniesensor	79
9.1	Sensorprincip	79
9.2	Elektronik	80
9.3	Optik og mekanik	81
9.4	Afprøvning	81
9.5	James	83
9.6	Konklusion	83
10	Tykfilmbaseret motordriver	85
10.1	Kredsløb	85
10.2	Tykfilmteknik	86
10.3	Tyke lederbaner	87
10.4	Flerlags trykktetik	88
10.5	Fremstilling	89
10.6	Testresultater	91
10.7	Konklusion	91
11	Styring af Scorbot	93
11.1	Elektronik	93
11.2	Direkte kinematik	94
11.3	Invers kinematik	97
11.4	Software	100
11.5	Konklusion	100
12	Integration	101
12.1	Strømforsyning	102
12.2	Datamat	103
12.3	Motorstyring	103
12.4	Sensorer	105
12.5	Software	105
12.6	Problemer og erfaringer	106
12.7	Konklusion	107

13 Konklusion	109
13.1 Motorstyring	109
13.2 Sensorer	110
13.3 Navigation	111
13.4 Dokumentation	111
13.5 Genbrug af materialer og udstyr	111
14 Fremtidig udvikling	113
14.1 Navigation	113
14.2 Sensorer	113
14.3 Effektelektronik	114
14.4 Projektideer	114
15 Forhandlerliste	117
Litteraturliste	119
 II Teknisk dokumentation	 127
16 Dokumentation af Cato	129
16.1 Mekanik	129
16.2 Strømforsyning	131
16.3 Bump sensorer og interface	136
16.4 Motorer og enkodere	139
16.5 Motor I/O box	140
16.6 H-broer	147
16.7 Computer	152
17 Dokumentation af James	155
17.1 Mekanik	156
17.2 Strømforsyning	157
17.3 Motorer og enkodere	157
17.4 Interconnect print	159
17.5 Motor-drivere	164
17.6 Computer	172

18 Dokumentation af Gefion	173
18.1 Funktionel beskrivelse	173
18.2 Hardware setup	176
18.3 Tilslutning	177
18.4 Adressering	181
18.5 Programmering	183
18.6 Diagrammer og printudlæg	188
18.7 Programmerbar logik	196
18.8 Kildetekst til motor testprogram	213
18.9 Kildetekster til MCI bibliotek	215
19 Dokumentation af Heimdal	231
19.1 Generel beskrivelse af Heimdal	231
19.2 Funktionel beskrivelse	231
19.3 Sonar-kontrol	233
19.4 Hardware setup	234
19.5 Tilslutning	235
19.6 Adressering	235
19.7 Programmering	237
19.8 Initialisering	240
19.9 Grundlæggende programmering	241
19.10PI/T timer	245
19.11BUS forsats	246
19.12Device driver	249
19.13Diagrammer og printudlæg	253
19.14Programerbar logik	258
19.15Kildetekster til device driver	267
20 Programmel til Scorbot	281
20.1 Struktur	281
20.2 Funktionsbibliotek til Scorbot	282
20.3 Testprogram med invers kinematik	284
20.4 Datafil til testprogram	284
20.5 Kildetekster	284

21 Dokumentation af liniesensor	293
21.1 Elektriske forbindelser	293
21.2 Diagram	294
21.3 Printlayout	295
22 Fremstilling af Rungner	297
22.1 Elektronik	297
22.2 Fremstilling	301
22.3 Tykfilmtest	308
22.4 Kredsløbstest	311
22.5 Effekt og virkningsgrad	321
III Noter	325
23 Opbygning og brug af DC-motorer	327
23.1 DC-motoren	327
23.2 Effektførstærker	330
24 Programmering af VMPMKC2 CPU-kort under OS-9	331
24.1 Godt begyndt	331
24.2 Udviklingssystemet	335
24.3 Hardwarenær programmering	337
24.4 Device drivers og file managers	341
24.5 Filoverførsel	346

Forord

Om uddannelsen i Datateknologi

Datateknologiuddannelsen blev oprettet i 1986 som et samarbejde mellem Odense Universitet og Odense Teknikum. Uddannelsen blev til på baggrund af et ønske fra det lokale erhvervsliv om en lokal uddannelse på civilingeniørniveau inden for praktisk anvendelse af informationsteknologi.

Fundamentet i uddannelsen er en række grundlæggende fag inden for hovedområderne: Matematik, fysik, datalogi, elektronik og datateknik; som de studerende følger sammen med studerende fra hhv. den naturvidenskabelige kandidatuddannelse på Odense Universitet og de Diplomingeniørstuderende på Odense Teknikum. De grundlæggende fag udbydes hovedsageligt af Institut for Matematik og Datalogi (IMADA) og Fysisk institut på Odense Universitet samt af det elektrotekniske retningsområde på Odense Teknikum.

Uddannelsen rummer desuden en række højt specialiserede fag, nogle obligatoriske, nogle valgfri, der udbydes af ovennævnte institutter samt af det nyoprettede *Mærsk Mc-Kinney Møller Institut for produktionsteknologi (MIP)* på Odense Universitet.

MIP, der blev oprettet som *Datateknologisk institut* 1. Jan. 1996 og skiftede navn til *MIP* 1. september. 1997, er Odense Universitets første rendyrkede teknisk-naturvidenskabelige institut. Med datateknologiuddannelsen som en af hjørnestenene skal MIP forske i intelligente produktionssystemer — bl.a. robotteknologi — og uddanne civilingeniører i bl.a. datateknologi.

Om AGV-teknologi på Odense Universitet

Forkortelsen AGV står for *Autonomous Guided Vehicle*. Betegnelsen kan bruges om alle tænkelige selvstyrende fartøjer — til lands, til vands, og i luften — men bruges i regelen kun om selvstyrende køretøjer.

Interessen for AGVteknologi holdt sit indtog på Odense Universitet med forskningsprojektet: *Autonomous Multiple Robot Operation in Structured Environments* også kaldet AMROSE¹. Projektet anvender avancerede simuleringstekniker fra molekyledynamik til at styre redundante robotsystemer i omgivelser med forhindringer. En af deltagerne i AMROSE projektet — Niels Jul Jacobsen — afholdt i August 1992 et sommerkursus i AGVteknologi, på Institut for matematik og datalogi (IMADA). Formålet med kurset var dels at få erfaring med vejfinding, sensorteknologi, styring m.m. og dels at skabe interesse for robotteknologi generelt. Jeg deltog sammen med ca. 15 andre studerende i kurset, men kun 3-4 af os holdt fast i interessen for AGVteknologi.

Efterfølgende har undertegnede, Peter Favrholt og Lene Monrad Petersen lavet hver vores bachelorprojekt i AGV-teknologi. Samtidigt med at jeg byggede en AGV, med computer, sensorer etc. lavede Peter et udviklingssystem og en multiprogramkerne til den. Senere udviklede Lene programmet, der gjorde AGV'en i stand til at foretage en simpel kortlægning af sine omgivelser. Projektrækken forløb over halvandet år i

¹Forskningsprojektet AMROSE har i mellemtiden udviklet sig til firmaet AMROSE A/S med 30 fuldtidsansatte medarbejdere, herunder mig selv.

løbet af 1993 og 1994. Oprindeligt var det meningen, at der skulle laves flere projekter med den anvendte AGV, men det viste sig, at den mekaniske holdbarhed og dokumentationen ikke var god nok til, at der kunne arbejdes på den uden Peters og min assistance.

Jeg valgte at fortsætte arbejdet med AGV-teknologi i håb om, at det ville blive starten på en ny projektrække, hvor en større, mere robust og bedre dokumenteret AGV kunne blive platform for en lang række eksperimenter og forskning i sensorteknologi, styring, vejfinding, kortlægning etc.

I 1997 udskrev Institut for automation (IAU), ved Danmarks Tekniske Universitet, Danmarksmesterskab i kørsel med robotbiler efter en hvid linie i gulvet. Hardware interessegruppen på Odense Universitet² besluttede sig for at deltage med et køretøj baseret på den teknologi, jeg havde udviklet i løbet af dette projekt.

Resultatet af Hardwaregruppens anstrengelser over to måneder blev James. En ombygget hobby-modelbil, der bortset fra mekanikken, liniesensoren og softwaren, er en ægte delmængde af teknologien i dette projekt. James var en stor succes, idet den vandt klassen for ikke-DTU køretøjer og samtidigt gennemførte banen næste dobbelt så hurtigt som det bedste DTU køretøj.

I august 1997 var jeg medunderviser i et nyt AGV sommerkursus, afholdt i samarbejde mellem MIP og AMROSE A/S. Kurset blev baseret på færdige singlechip datamater i byggesæt og billige fjernstyrede legetøjsbiler. Der deltog 18 studerende fordelt på 4 hold. Kurset blev afsluttet med en demonstration/konkurrence, hvor de 4 biler skulle følge en oval bane markeret med linier på underlaget. Kurset var en succes og blev gentaget i samarbejde med Odense Teknikum i 1998.

Hvor interessen for AGV-teknologi døde meget hurtigt ud efter kurset i 1992, ser det ud til at aktiviteterne omkring James og sommerkurset i 1997/1998, har fået interessen til at bide sig bedre fast. Interessen har imidlertid flyttet sig fra IMADA til MIP. MIP interesserer sig i begrænset omfang for AGV teknologi som forskningsobjekt, men har med udgangspunkt i James projektet og 1997 sommerkurset, en stor interesse for AGVprojekter i forbindelse med undervisning.

Tak til

Jeg vil gerne rette en tak til alle der har været behjælpelige med dette projekt, men især til **Einar Hougs** (IMADA) og **Niels Jul Jacobsen** (AMROSE A/S) — der har ydet værdifuld hjælp og støtte gennem hele projektet. Desuden en tak til **Jytte Hansen, Michael Djørby, Peter Favrholdt, Thorbjørn Ravn Andersen, Anders Elkær Tønnesen, Ole Benny Hansen, Peder Thusgaard Ruhoff** samt alle de studerende der har arbejdet med James og Cato.

²En selvbestaltet gruppe, startet af Anders Elkær Tønnesen og undertegnede. Den består hovedsageligt af datateknologistuderende med interesse for elektronik

Del I

Rapportdel

Kapitel 1

Indledning

Dette projekt er den foreløbige kulmination i forsøget på at opbygge en projektrække omkring selvstændige robotkøretøjer — AGV'er — ved Odense Universitet. Formålet med projektet er, med udgangspunkt i materialer fra tidligere projekter, at udvikle et selvstændigt robotkøretøj der kan bruges som platform for forskning i og udvikling af sensorbaseret robotstyring. For at sikre at projektet kan bruges som springbrædt for efterfølgende projekter, lægges der vægt på robusthed, dokumentation, samt muligheder og visioner for videre udvikling.

1.1 Forhistorie

Alle AGV-projekterne er tilknyttet forskningsprojektet: *Autonomous Multiple Robot Operation in Structured Environments* — AMROSE. Det er en central del af AMROSE's koncept at have en matematisk model af det miljø, en robot skal bevæge sig i. Denne model kan opbygges på baggrund af en statisk geometrisk model (CAD), den kan opbygges dynamisk vha. sensorer, eller der kan anvendes en kombination af de to metoder.

Folkene bag AMROSE lancerede i august 1992 sommerkurset: *Konstruktion og styring af AGV'er*, der havde til formål at fremme interessen for kontrolsystemer, sensorer, og andre emner med relation til robotstyring. Kurset resulterede ikke i en fungerende AGV, men det gav deltagerne nyttig viden og erfaring inden for området. Jeg deltog selv i kurset og fik her styrket min interesse for samspillet mellem elektronik, programmer og den *virkelige verden*, samtidigt med at jeg fattede interesse for AGV'er som test- og udviklingsplatform for dette samspil.

Mit eget bachelorprojekt i 1993 bragte erfaringerne fra sommerkurset til anvendelse i form af en lille *skildpadde*-AGV, som blev udstyret med 3 POLAROID sonar-afstandsmålere og en passiv infrarød detektor. Mit projekt blev umiddelbart fulgt op af et andet bachelorprojekt, hvor der blev udviklet en multi-programkerne og et funktionsbibliotek til skildpadden [2]. Senere blev den anvendt i et tredje projekt, der brugte skildpaddens sonarsensorer til at opmåle og kortlægge et lokale [5].

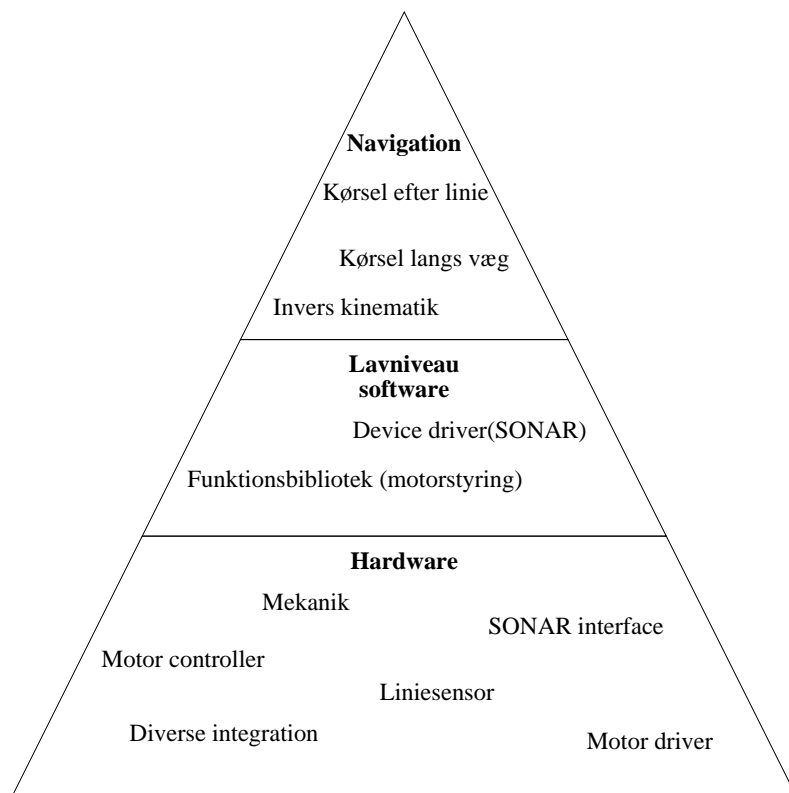
Det største problem i sommerkurset var at integrere styredatamaten og dens mikroprocessorstyrede I/O moduler med den virkelige verden. Problemerne med at udvikle og tilpasse elektronik og hardwarenære programmer blev kraftigt undervurderet og blev projektets akilleshæl.

1.2 Projektforløb

Erfaringerne fra tidligere projekter viser entydigt vigtigheden af en velfungerende og veldokumenteret hard- og softwaremæssig platform og jeg har vægtet min arbejdsindsats derefter.

Langt den største del af projektet er gået med udvikling af hardware. En mindre del er gået med udvikling af lavniveau software til styring af hardwaren og en lille del er gået med at studere og afprøve konkrete teknikker til navigation. Projektet har derved antaget pyramideform.

Efter at have foretaget nogle overordnede betragtninger og designvalg er jeg så at sige startet fra bunden. Nedenstående liste beskriver i grove træk det praktiske arbejde, jeg har lavet i projektet og afspejler også i nogen grad rækkefølgen.



Figur 1.1: Projektindhold

- Fundamentet for at kunne styre en AGV er en passende motorstyring. Jeg har udviklet en 4 akset VME-bus baseret motorcontroller, kaldet **Gefion**. For hver akse implementerer Gefion en PID regulering af en motors position med et dynamisk sætpunkt, der genereres af en programmerbar S-kurve generator.
- Styresignalet fra motorcontrolleren skal *forstærkes* af et passende effektttrin/motordriver. I løbet af projektet har jeg arbejdet med forskellige kredsløb til dette og udviklet en teknik til fremstilling af højeffekt tykfilm prototyper.
- For at opbygge et hensigtsmæssigt og stabilt mekanisk fundament for AGV'en har jeg brugt en smule tid på design og lidt mere tid på metalsløjde. AGV'en opbygget over det resulterende skelet kalder jeg **Cato**.
- Undervejs i projektet har Cato's strømforsyning udviklet sig fra testopstilling til prototype.

- For at kunne kalibrere AGV'en udviklede jeg en sensor, der sætter AGV'en i stand til at følge en streg i gulvet med stor nøjagtighed. Ved at følge en lige streg kan skævheder i hjul m.m. måles og udkompenseres.
- Med udgangspunkt i sonarinterfacet i mit bachelorprojekt udviklede jeg et mere avanceret VME-bus-baseret interface til POLAROID's sonarafstandsmålere. Interfacet, som jeg kalder **Heimdal**, muliggør tilkobling af op til 16 sonarafstandsmålere med mulighed for at måle på fire ad gangen.
- For at få adgang til Gefions funktioner fra C har jeg udviklet et funktionsbibliotek, der implementerer en række kommandoer til styring af motorerne.
- For at udnytte POLAROIDs sonar-afstandsmålere og Heimdal fuldt ud har jeg udviklet en interrupt-drevet OS-9 device-driver, der integrerer sonar-målingerne med OS-9's I/O system.
- Jeg har koblet en 5 akset robotarm, beregnet til undervisning, til to Gefion motorkontrollere, opmålt robotten og opstillet en invers kinematik for den. Dernæst har jeg implementeret et funktionsbibliotek til styring af den i joint-, såvel som kartesiske-koordinater.
- Da AGV konkurrencen *DTU robocup 1997* blev annonceret i foråret 1997, var jeg foregangsmand og koordinator, for 10-15 datateknologer og andre, der byggede og programmerede AGV'en **James** til at repræsentere Datateknologiuddannelsen i konkurrencen. James er i alt væsentligt en nedskalering af Cato baseret på en fjernstyret modelbil.
- For at sætte Cato i stand til at detektere kollisioner har jeg udstyret den med bump-sensorer.
- For at afprøve sonarafstandsmålerne som en del af et simpelt navigationsprincip har jeg udviklet et testprogram, der sætter James i stand til at køre parallelt med en ubrudt væg.

1.3 Rapporten

Denne rapport dokumenterer det arbejde, jeg har udført i forbindelse med specialeprojektet. Projektet har været meget stort og rummet en stor del praktisk arbejde. Ud over at være bedømmelsesgrundlag for mine vejledere og censor, har jeg lagt vægt på at rapporten skal sætte andre studerende i stand til at forsætte mit arbejde med de emner jeg har berørt i projektet. For at tilgodese begge formål har jeg delt rapporten op i tre dele:

Rapportdelen giver en overordnet beskrivelse af mit arbejde. De første tre og det sidste kapitel: *Indledning, emneoversigt, overordnet koncept og konklusion*; forudsætter ikke detailkendskab til AGV-relaterede emner og er henvendt til alle læsere. De resterende kapitler i rapportdelen beskriver udviklingen af specifikke dele af projektet og forudsætter forhåndskendskab til de beskrevne emner.

Dokumentationsdelen rummer detaljeret dokumentation af mit arbejde på et niveau, der skal sætte næste generation speciale- og bachelorstuderende i stand til at anvende, udbygge og kopiere det arbejde, jeg har lavet. Dokumentationsdelen er ikke beregnet til gennemlæsning men til opslag efter behov.

Notedelen har til formål at videregive vigtig viden og erfaring om dele af projektet, jeg ikke selv har udviklet. Noterne er ikke færdige og jeg ønsker ikke, at lade dem indgå i bedømmelsesgrundlaget. Når de alligevel er med, skyldes det, at de allerede i deres nuværende form har vist sig at være en hjælp for studerende, der vil arbejde videre med de beskrevne emner.

For at gøre det nemmere at anvende dele af mit arbejde i andre sammenhænge, har jeg anvendt engelske tekster i de fleste af mine figurer.

En PostScript-udgave af rapporten findes sammen med alle relevante software-kildetekster og hardware-designfiler samlet på en CD-ROM.

Kapitel 2

AGV-teknologi — En emneoversigt

2.1 AGV'er

Forkortelsen *AGV* står for *Autonomous Guided Vehicle*. Betegnelsen kan bruges om alle tænkelige selvstyrende fartøjer — til lands, til vands, i luften og i rummet, men den bruges normalt om køretøjer. Der er stort sammenfald mellem AGV-begrebet og begrebet *autonom robot*, fordi en autonom robot kan opfattes som en AGV, der er beregnet til at påvirke/manipulere sine omgivelser frem for blot at reagere på dem.

Anvendelsen af AGV'er er hovedsageligt fordelt mellem industriel automation, servicerobotter, rumforskning, oceanografi og forskning i forskellige emner relateret til robotter, sensorer og kunstig intelligens. Hvis AGV-begrebet anvendes i bredest mulig forstand, er det nødvendigt at inkludere våbentechnologi, hvor selvstyrende missiler, torpedoer mv. repræsenterer et stort anvendelsesområde.

2.1.1 Industriel automation, service og overvågning

AGV'er har længe været benyttet i industrien til transport mellem forskellige processer og lagre i en produktion. Her er der typisk tale om meget simple AGV'er¹, der har meget begrænsede muligheder for at opfatte deres omgivelser, og som derfor færdes i et miljø der er specielt tilpasset til at lette AGV'ens navigation. [84] [85] [87]

Der er et meget stort potentielt marked i at udvikle AGV'er og autonome robotter, der kan løse opgaver i et miljø, der ikke er specielt tilpasset AGV'ens behov. F.eks. vil autonome husholdningsmaskiner, der masseproduceres til fornuftige priser, formentligt udkonkurrere konventionelle støvsugere, græsslåmaskiner etc., på samme måde som vaskemaskiner har udkonkurreret vaskebrætter. [70] [71] [78]

Mere konkret har der gennem de sidste år eksisteret AGV'er som er beregnet til at løse transport og overvågningsopgaver i et umodificeret (kontor-) miljø. [46] [47] [81] [86] [89] [91]

2.1.2 Rumfart, oceanografi mv.

En del AGV-forskning drejer sig om semiautonome anvendelser i rumfart og oceanografi, hvor omstændighederne ikke tillader direkte fjernstyring af de anvendte ubemandede fartøjer pga. tidsforsinkelser, begrænset båndbredde, eller på anden måde forringede kommunikationsmuligheder. Semiautonom opførsel tillader ubemandede sonder at løse opgaver, der traditionelt kun har kunnet løses bemandet.

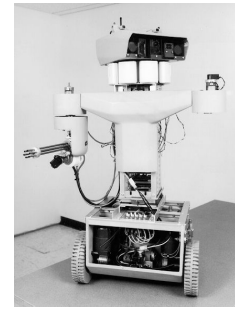
¹ Ofte kaldet *Automated Guided Vehicles*, i stedet for *Autonomous*



(a) HelpMate



(b) Nomadic Technologies



(c) Robart III

Figur 2.1: AGV'er / Autonome robotter til service og overvågnings opgaver

2.1.3 Forskning

AGV'er er meget populære indenfor forskning i robotter, sensorer og kunstig intelligens. En del af forskningen går direkte på at udvikle bedre AGV'er, men AGV'er bruges ofte blot som en testplatform for grundforskning i forskellige former for kunstig intelligens, sensortechnologi, adfærdsforskning mv. [88]. AGV'er er meget velegnede som testplatform, fordi kompleksiteten af de opgaver der skal løses, kan varieres fra det banale til det meget svære ved simple ændringer i AGV'ens omgivelser. Netop fordi AGV'er er så velegnede som platform for forskning og udvikling inden for så mange felter, er det ved at blive almindeligt at stimulere udviklingen ved at afholde AGV-konkurrencer som f.eks. det internationale robotfoldboldmesterskab, RoboCup [43], *The Aerial Robotic Competition* [74], det mere lokale CMU Mobot race [69] og det danske DTU-RoboCup [79].

2.2 Navigation

Det der gør en AGV selvstyrende, er dens evne til at navigere. I dette afsnit kommer jeg ind på forskellige aspekter af AGV navigation.

Al navigation består af to elementer: **Positionsbestemmelse** og **ruteplanlægning**, der afhængigt af sammenhængen begge er mere eller mindre afhængige af **kendskab til omgivelserne**.

2.2.1 Relativ positionsbestemmelse.

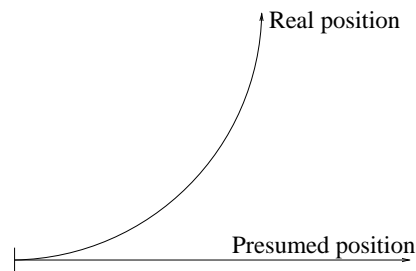
Ved relativ positionsbestemmelse tages der udgangspunkt i en kendt begyndelsesposition, hvorefter AGV'ens positionsændringer løbende registreres og integreres for at give den nuværende position i forhold til udgangspunktet. Metoden har været anvendt til søs i århundreder og går også under navnet bestiknavigation eller *dead reckoning*.

Normalt bruges kontrolmatematens kendskab til AGV'ens bevægelser til at foretage en løbende fremskrivning af de formodede værdier for kørt distance, retning og position. Der vil altid være uoverensstemmelser mellem formodet og virkelig distance og retning [35]. En rimelig model for fejlen ved et køretøj er skitseret ved (2.1), hvor \mathbf{e} er fejlen på formodet distance og retning, d er kørt distance, \mathbf{a} er en koefficient, der afhænger af mekaniske skævheder og mistilpasninger ved AGV'en og \mathbf{n} er en støjfunktion, der repræsenterer ujævnheder, snavs etc. ved underlaget og AGV'ens hjul.

$$\mathbf{e}(d) = \begin{bmatrix} e_{d'} \\ e_{\phi'} \end{bmatrix} = \int_0^d (\mathbf{a} + \mathbf{n}(z)) dz = \mathbf{a} d + \mathbf{N}(d) \quad (2.1)$$

Fejlen på selve positionsbestemmelsen vokser med e og d . En voksende fejl i bestemmelsen af kørselsretningen får positionsfejlen til at vokse eksplosivt (figur 2.2) og metoden kan forbedres væsentligt, hvis retningsfejlen kan begrænses, f.eks. ved kalibrering [4] [62] eller helt elimineres.

En simpel og effektiv metode til eliminering af retningsfejl er at lade AGV'en følge fastlagte spor i form af linier, ledninger, metaltråde eller lignende i gulvet. I disse tilfælde degenererer positionsbestemmelse til et endimensionalt problem. [54]



Figur 2.2: Udvikling af positionsfejl

Relativ positionsbestemmelse uden drift er mulig, hvis der i AGV'ens omgivelser findes regelmæssige kendetegn, der kan detekteres med fuldstændig sikkerhed. F.eks. fuger i et flisegulv, regelmæssigt anbragte armeringsjern i et betongulv eller regelmæssigt anbragte loftslamper. Hvis AGV'en kører efter faste spor, kan kendetegn integreres med sporet.

2.2.2 Absolut positionsbestemmelse

De to mest kendte teknikker til absolut positionsbestemmelse i stor skala DECCA og GPS, kan normalt bestemme modtagerens position inden for 10-30 m. Vha. forskellige tricks kan der opnås bedre nøjagtighed for GPS. Begge systemer fungerer kun udendørs.

Der findes ikke generelle systemer til absolut positionsbestemmelse indendørs. Et væsentligt problem er, at radio- og lydbølger reflekteres og absorberes af vægge og inventar og dermed er uegnede som bæremedier ved en nedskalering af konventionelle udendørs navigationssystemer.

En driftsikker men kostbar metode er at lægge et gitter af ledninger i underlaget, hvor AGV'en skal færdes. Ved at sende vekselstrøm gennem ledningerne i en kendt sekvens kan positionen af en modtager bestemmes ret nøjagtigt ud fra magnetfelterne omkring ledningerne.

En billigere relativt simpel men mindre driftsikker metode går ud på at anvende kortlagte **kodede** kendemærker, der kan registreres på afstand. De registrerede placeringer kan entydigt sammenlignes med de kortlagte placeringer og derved give AGV'ens position [38] [39] [63].

En vanskelig men i mange sammenhænge attraktiv metode, er at sammenligne ukodede kendemærker eller naturlige kendetegn med et *kort*. Hvor svært det er at anvende denne metode, afhænger i høj grad af miljøet og de oplysninger, der er til rådighed om det via kort og sensorer. Det vil typisk være svært eller umuligt at bestemme positionen entydigt overalt, hvilket kan afhjælpes ved at kombinere metoden med relativ positionsbestemmelse [33] [41] [48] [57] [58] [64].

2.2.3 Kendskab til omgivelserne

Kendskab til omgivelserne kan komme fra forhåndskendskab i form af en model, fra målinger på de virkelige omgivelser vha. sensorer eller som en kombination af de to. Afhængigt af omgivelsernes beskaffenhed og den opgave AGV'en skal løse, kan behovet for kendskab til omgivelserne variere fra det banale til det meget avancerede.

Hvilken metode, der anvendes til at modellere AGV'ens omgivelser, afhænger ligeledes af omstændighederne. Modellen kan være statisk og opbygget på forhånd, den kan være dynamisk og opbygges undervejs [5] [51] [53] [56] [66] eller den kan rumme både statiske og dynamiske elementer. [52]

Ved mange former for navigation er det nødvendigt at skaffe oplysninger om de virkelige omgivelser vha. sensorer, dels for at foretage en positionsbestemmelse og dels for at foretage en form for dynamisk

kortlægning — herunder registrering af uforudsete forhindringer. Hvilke typer sensorer, der er relevante, afhænger af omstændighederne.

2.3 Sensorer

Mange forskellige sensorer kan være relevante for AGV'er i forskellige sammenhænge. Sensorer kan bruges alene eller i samarbejde med andre sensorer.

2.3.1 Interne målinger

Der kan være behov for at foretage en lang række målinger af interne værdier i en AGV f.eks. måling af: temperaturer, strømforbrug, batterispændinger etc. Af særlig interesse er måling af AGV'ens bevægelse (*odometri*), der er vital for anvendelsen af relativ positionsbestemmelse. Typisk baseres odometrien på de samme omdrejningssensorer, der anvendes i motorernes servosystem evt. kombineret med gyro- eller magnetkompass for at eliminere retningsfejl. På ujævnt underlag, kan der med fordel suppleres med gyroskoper og/eller accelerometre, der måler vinkelændringer/-hastigheder og accelerationer [50] [55]

2.3.2 Detektion af Berøring

Af indlysende årsager er det interessant at kunne detektere fysiske kollisioner. Kollisioner kan anvendes som middel til måling på omgivelserne, men som regel tilstræbes det at undgå dem ved at bruge andre sensorer. Skulle en kollision alligevel forekomme, er det vigtigt at registrere og reagere på den, dels for at begrænse evt. materiel skade og dels fordi kollisioner kan bringe AGV'en ud af kurs. Forskellige former for kontakter med påmonterede *kofangere* eller *følehorn* er en af de mest anvendte sensorer til detektion af sammenstød, men alternative principper — fra overvågning af motorernes energiforbrug, til efterligninger af den menneskelige hud — er mangfoldige.

2.3.3 Detektion af nærhed

For at minimere chancen for kollision kan der anvendes sensorer, der kan registrere, om der er forhindringer tæt på AGV'en. Hvad der defineres som *tæt på* afhænger af sammenhængen. Dette område er meget kaotisk, idet der findes mange forskellige sensorprincipper, der kan opsamle mere eller mindre nøjagtige data om tilstedeværelse og bevægelse af objekter i nærheden. Typiske nærhedsfølere baserer sig på mekaniske *følehorn* eller målinger af omgivelsernes påvirkning af et udstrålet elektrisk, magnetisk, elektromagnetisk eller akustisk felt. [37] [46] [61]

2.3.4 Afstandsmåling

Grænsen mellem nærhedsmåling og afstandsmåling er flydende og afhænger helt af, hvad der opfattes som *nært*. En afstandsmåler kan fungere som nærhedssensor, hvis den kan foretage målinger inden for AGV'ens *nære* omgivelser. Udover detektion af *nærhed* bruges forskellige former for berøringsløse afstandsmålere til at skaffe AGV'en geometriske oplysninger om omgivelserne, til kortlægning, lokalisering, vejfinding mv.

Til indenedørs AGV-teknologi er sonar det mest populære princip til afstandsmåling, fordi det er billigt, simpelt, nøjagtigt og driftsikkert. De to største ulemper ved sonar er den langsomme målefrekvens pga. lydens hastighed og den relativt dårlige retningsbestemthed pga. lydens spredning. [8, afsnit 10.6.3.2] [36] [18] [25] [42] [31] [60] [59] [46] [81] [83] [89] [61]

Laser-afstandsmålere baseret på *time of flight* (LIDAR²) vinder stærkt frem som konkurrent eller supplement til sonar, efterhånden som der udvikles billig elektronik og optik, der er hurtigt nok til nøjagtig måling af de meget små tidsforsinkelser, der er mellem udsendelse og modtagelse af en lyspuls, der reflekteres af en forhindring indendørs (6.5 ns/m). LIDAR afstandsmålere kan blive næsten vilkårligt retningsbestemte, kan anvende meget høje målefrekvenser og måleretningen kan styres hurtigt og nøjagtigt med bevægelige spejle. Den største ulempe ved denne type sensor er de store datamængder, der genereres ved scanning af omgivelserne. [44] [90] [82]

Et tredje princip der anvendes i AGV-teknologi er triangulering i et system, hvor en fokuseret lyskilde, et oplyst punkt på måleobjektet og en optisk vinkelmåler (1 eller 2 dimensionalt kameraelement) danner hjørnerne i en trekant, hvor afstanden mellem lyskilde og et oplyst punkt kan beregnes ud fra den målte vinkel. Rækkevidde og opløsning afhænger af sensorens geometri og opløsningen på det anvendte kameraelement. Opdateringsfrekvensen afhænger af kameraelementet. Ved at anvende et 2D kameraelement og lade lyskilden udsende lys i et plan opnås en relativt billig 3D scanner. [8, afsnit 10.6.3.1] [46]

Radar-teknologi er efterhånden ved at udvikle sig til at kunne lave nøjagtige afstandsmålinger på korte afstande, hvilket bl.a. udnyttes til niveaumålere i tankanlæg. Radar-afstandsmålere anvendes i udendørs AGV-teknologi, men er på trods af den langt højere målefrekvens endnu ikke begyndt at erstatte sonar-afstandsmålere til indendørs brug. [49] [72] [80]

2.3.5 Datamatsyn

Datamatsyn eller *Computer vision* er en af de mest populære sensorteknologier i forbindelse med forskning i AGV'er og mobile robotter. Til gengæld bruges teknologien kun sparsomt i kommercielle AGV-produkter. Det overordnede problem med datamatsyn er dels de enorme mængder information et kamera genererer koblet med den ofte meget beregningskrævende billedanalyse. Med udviklingen af stadigt hurtigere processorer og bedre algoritmer er datamatsyn ved at vinde indpas i stadigt mere avancerede industrielle anvendelser og i universitetsmiljøerne findes en lang række AGV'er, der styres helt eller delvist af datamatsyn. [32] [34] [65]

2.3.6 Specialiserede sensorer

AGV'er og mobile robotter kan betjene sig af en lang række specialiserede sensorer i forbindelse med navigation. Typiske eksempler er forskellige former for elektriske, magnetiske, optiske eller kemiske sensorer til at identificere og følge spor i eller på gulvet. Ligeledes kan der anvendes sensorer, der er dedikeret til at identificere bestemte kendemærker i omgivelserne. [45] [63] [67]

²LIght Detection And Ranging

Kapitel 3

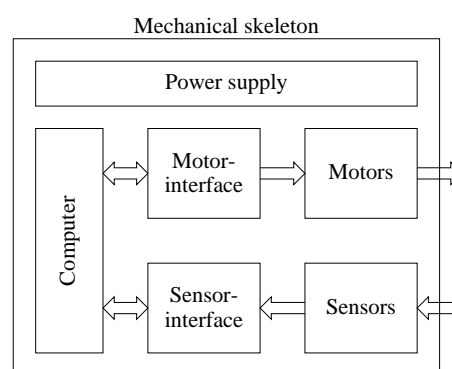
Overordnet koncept

Konceptet for AGV-projekteterne ved datateknologiuddannelsen er stort set forblevet uændret siden sommerkurset i 1992 og kan beskrives som følger:

- Hovedformålet med AGV'erne er at fungere som platform for afprøvning af sensorer og algoritmer til behandling af sensordata. Desuden skal de fremme interessen for robotrelaterede emner blandt studerende.¹
- Da AGV-teknologi kun er et lille forskningsområde ved OU, kan der ikke bindes store ressourcer til hvert enkelt projekt. Projekterne holdes på lave budgetter, bl.a. ved så vidt muligt at basere dem på eksisterende udstyr og teknologier.
- Da der ikke rådes over større laboratoriefaciliteter, skal AGV'erne kunne færdes i kontormiljøet på Campus.
- For at opnå en vis udvikling er det vigtigt, at AGV'erne er robuste og veldokumenterede nok til, at de kan indgå i længere projektrækker.

En AGV består typisk af følgende komponenter:

1. Et mekanisk fundament eller ramme.
2. En eller flere motorer med tilhørende energiforsyning og motorstyring, der giver køretøjet de ønskede køre og styreegenskaber.
3. Hvis køretøjet skal interagere med omgivelserne, skal den udstyres med en række sensorer, der kan opsamle den information, der er behov for.
4. For at få en *fornuftig* interaktion med omgivelserne skal der ske en avanceret bearbejdning af data fra sensorerne, som resulterer i en påvirkning af motorerne. Denne bearbejdning udføres naturligvis af en datamat.



Figur 3.1: Typisk AGV struktur

I dette kapitel gennemgår jeg de overordnede designvalg, jeg har foretaget mht. ovenstående 4 punkter.

¹Siden **DTU RoboCup** i 1997 bliver der lagt stadig større vægt på AGV projekternes rolle i profileringen af datateknologiuddannelsen.

3.1 De forhåndenværende ”søm“

En af de vigtigste aspekter af det overordnede design er de materialer og midler, jeg har til rådighed. Ved projektets begyndelse havde jeg følgende at gøre godt med:

- To identiske rektangulære 60×80 cm. stålrammer med et trillebørhjul foran i hver side og et fritløbende hjul — som dem på indkøbsvogne — bagerst i midten.
- To 12 V, 80W DC-motorer med snekegear, der med tandremme kan drive de to trillebørhjul.
- 9 sonarafstandsmålere fra POLAROID.
- En doppler-radar bevægelses-detektor.
- Et budget på 15.000 kr., der skal dække hele projektet bortset fra motorcontrolleren. Motorcontrolleren betragtes økonomisk som et separat projekt, idet den har mange anvendelser ud over AGV-styring. Budgettet dækker heller ikke udgifter til fremstilling af tykfilm-kredsløb, der dækkes af driftsbudgettet på Odense Teknikums tykfilmlaboratorium.
- Ud over budgettet, stiller Institut for Matematik og Datalogi — IMADA eksisterende komponenter og udstyr til rådighed i det omfang, der er behov og mulighed for det.

Ud over rammen, sonarsensorerne og motorerne er der en mængde grej til rådighed fra sommerkurset i 1992. Langt det meste må betragtes som skrot pga. manglende vedligehold, manglende dokumentation eller for dårlig kvalitet.

3.2 Datamat

Budgettet åbner ikke mulighed for at anskaffe en ny datamat specielt til dette projekt, hvorfor jeg er henvist til at anvende eksisterende udstyr. Mine valgmuligheder er:

- | | |
|---|--|
| • Et Inmos-T800 baserede transputer CPU-kort fra sommerkurset i 1992. | • En IBM-kompatibel PC med 8086 eller 80286 processor. |
| • En 68008 baseret datamat fra mit bachelorprojekt. | • En VME-bus baseret industricomputer med 68020 CPU-modul fra PEP modular computers. |
| • En Sun eller Next UNIX-arbejdsstation. | |

Transputeren fungerede mildest talt ustabil under 1992-sommerkurset og selv om en af de andre deltagere — Michael Djørby, senere har lagt et stort arbejde i at forbedre den, vil jeg stadig ikke betragte den som egnet. Dertil kommer at transputerteknologien allerede fra projektets start er ved at forsvinde ud af IMADA's interesseområde.

Datamaten fra mit bachelor-projekt fungerer udmærket, men har næppe regnekraft, lagerplads, eller I/O-faciliteter nok til at fungere tilfredsstillende i større AGV-projekter.

UNIX arbejdsstationerne lider af mangel på I/O-faciliteter, er ikke særligt mobile og er desuden i flittigt brug i IMADA's terminalrum og kontorer.

IMADA rådede ved projektets start kun over små og relativt gamle IBM-PC'er med lille regnekraft. PC'en er interessant i kraft af det store udbud af billige I/O kort, men IMADA rådede ikke over en egnet multi-programkerne eller operativsystem til PC arkitekturen (QNX, OS-9000 e.lign.)

Mit valg er faldet på PEP datamaterne, som IMADA har 8 af. PEP'erne er nærmest ideelle til formålet. De er baseret på den industrielle VME-bus og modulært opbygget i et 19" rack. De eksisterende 68020 baserede CPU-kort har en passende regnekraft og kan udskiftes om nødvendigt. IMADA råder over en række digitale og analoge I/O moduler. Enkelte af maskinerne er udstyret med 20MB harddisk og realtime operativsystemet OS-9 med tilhørende udviklingssystem.

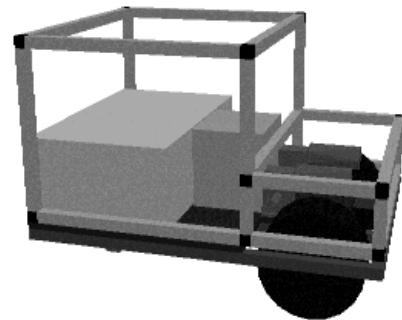
3.3 Mekanisk skelet

De eksisterende stålrammer med hjul er en udmærket platform for opbygning af et AGV-skelet. Deres største handicap er, at forhjulene har pneumatiske ringe, hvilket både introducerer en væsentlig usikkerhed på forholdet mellem størrelsen af de to hjul, og forholdet mellem hjulomdrejning og kørt distance. Denne usikkerhed vil påvirkes af mængden og fordelingen af den masse, der placeres på platformen. Der kan i høj grad kompenseres for denne ulempe ved at måle hjulstørrelserne jævnlige[4]. Når AGV'en er kalibreret, kan de pneumatiske ringe vise sig fordelagtige frem for hårde hjul, idet AGV'en kan køre over små ujævnheder uden at miste kontakten med gulvet.

I sommerkurset var stålrammerne forsynet med en 10 mm tyk spånplade som dæk. Spånpladen var ikke velegnet for montage af beslag m.m. og jeg har derfor udskiftet den med en 15 mm tyk krydsfinerplade.

I sommerkurset blev AGV'ens overbygning skruet sammen af L-profiler, der normalt er beregnet til industrielle stålreoler. Jeg ønskede en anden løsning, idet stålprofilerne var for tunge og klodsede. Jeg valgte at basere overbygningen på 25 mm brede kvadratiske aluminiums profiler (PORS system) og designede en passende opbygning vha. et 3D modelprogram (Sculpt 3D). Overbygningen er designet ud fra følgende krav:

- Den skal flugte omridset af den oprindelige stålramme.
- Den skal kunne bære en last gerne en person (100 kg).
- IMADA's robotarm — SCORBOT — skal kunne monteres ovenpå.
- Der skal kunne monteres sidepaneler, hvori der kan monteres sensorer.
- Der skal være plads til motorer, batteri og et 6U, eller to 3U 19" racks.



Figur 3.2: 3D model af Cato

3.4 Motorer og motorstyring

De få kørsler, det lykkedes at foretage under sommerkurset, viste at de oprindelige motorer og deres gearing passede fint til AGV'en. De giver en tophastighed på ca. 1 m/s og kan trække AGV'en på et plant gulv med en 100 kg tung last.

De inkrementalencoder, der blev anvendt under sommerkurset, fungerer godt. Under sommerkurset blev de tilkoblet motorernes drivakslar — efter gearingen — vha. kilerebbe. For at optimere opløsningen og udelukke problemer med slør ønsker jeg at montere enkoderne direkte på motorenes rotor før gearingen.

I sommerkurset skulle reguleringen af motorerne varetages i software af en separat 68HC05 mikroprocessor udstyret med passende I/O enheder og koblet til den centrale *transputer* vha. en *transputerlinkadapter*. I mellemtiden har jeg fundet frem til en dedikeret signalprocessor, der varetager de beregningskrævende aspekter af motorstyring — regulering og kurvegenerering — i hardware. Signalprocessoren LM628/LM629 har et almindeligt Intel-8086 kompatibelt 8-bits synkront businterface, der med lidt tilpasning kan kobles til den asynkrone VME-bus. Jeg har i løbet af projektet udviklet VME-modulet **Gefion**, der er bestykket med fire LM629 og kan styre fire uafhængige motorer.

Jeg har tidligere arbejdet på en prototype af en lavspændings *switched-mode* motordriver, der er baseret på tykfilm. Prototypen fungerer, men har blandt andre fejl for stor indre modstand til at kunne bruges i dette projekt. Tykfilmteknikken er velegnet til seriefremstilling, og giver samtidigt mulighed for at fremstille kompakte effektkredsløb med attraktive monterings og køleegenskaber. Da jeg allerede har en del erfaring

med tykfilm, og skal bruge mindst 8 motordrivere, finder jeg det passende at gå videre med udviklingen som en del af projektet.

3.5 Strømforsyning

Et autonomt køretøj skal medbringe sin egen energiforsyning, der i dette tilfælde baseres på opladelige batterier, af praktiske årsager.

AGV'en bør have energi nok til at køre mindst en time på en opladning. Batteriet skal nemt kunne udskiftes med et nyopladet og AGV'en skal, mens den holder stille, kunne forsynes fra en ekstern forsyning.

Motorenes lave indre modstand i sammenhæng med en *switched-mode* motordriver kan trække AC-strømme på op til ca. $30A_{p-p}$ fra batteriet, hvilket kan give interferensproblemer og frister til at anvende separate batterier til motorer og resten af elektronikken. Erfaringerne fra mit bachelorprojekt taler imidlertid imod en opsplittet forsyning, idet batterierne aldrig af- eller op-lades lige hurtigt, hvilket giver anledning til begrænsninger og problemer med brug af AGV'en. Jeg ønsker følgelig at opbygge strømforsyningen omkring et enkelt batteri og løse interferensproblematikken på anden vis.

3.6 Sensorer

Navigation af en AGV baseres typisk på bestiknavigtion, der udvides med korrektioner ud fra et statisk kort og senere dynamisk kortlægning. Mit valg af sensorer er betinget af dette hierarki men i høj grad også af økonomi og tid. I dette afsnit gennemgår jeg egenskaberne af de sensorer jeg har valgt og grundene til, at jeg har valgt dem. Desuden gennemgår jeg også nogle af de sensorer, jeg har overvejet at bruge men valgt fra.

3.6.1 Motorstyringen som sensor

Motorstyringen med positionsfeedback fra enkodere er den mest grundlæggende sensor i AGV'en. Ved kontinuerligt at overvåge kørslen af begge motorer kan datamaten hele tiden holde rede på AGV'ens position i forhold til udgangspunktet. Denne bestiknavigation eller *dead reckoning* er naturligvis behæftet med en fejl (drift), der vokser efterhånden som AGV'en kører. Hvor hurtigt den anslåede position afviger fra den virkelige afhænger primært af AGV'ens og underlagets mekaniske egenskaber.

Udover bestiknavigation kan PID-regulatoren i motorsyningen være en kilde til oplysninger om omgivelserne, idet positionsfejlen og evt. integralet af denne kan fortælle om belastningen af den enkelte motor. Dermed kan datamaten detektere kollisioner, hældninger og større ujævnheder.

3.6.2 Liniesensor

En simpel måde at optimere bestiknavigation på er ved at kalibrere AGV'en ved at få den til at følge en lige streg på gulvet.

Tage Søndergård har i sit bachelorprojekt [4] foretaget en sådan kalibrering vha. en primitiv optisk sensor. Sensoren bestod af fire digitale refleksionsfølere i en korskonfiguration. Sensoren var meget følsom overfor baggrundsbelysning og krævede meget stor kontrast (metaltape). Pga. sensorens digitale natur kunne AGV'en ikke følge afvigelser fra linien dynamisk men opdagede først fejlen, når den var et par cm væk fra linien. På trods af sensorens mangler blev der opnået rimelige resultater, der formentligt kan blive endnu bedre med en sensor, der kan følge linien dynamisk.

Da nøjagtig kørsel er altafgørende for kvaliteten af bestiknavigation, udvikler jeg en sensor, der kan måle afvigelsen fra en linie dynamisk. Af praktiske årsager, skal sensoren fungere i normal belysning og kunne arbejde med almindeligt sort eller hvidt PVC tape.

3.6.3 Kollisionssensor

For at detektere kollisioner, der kan bringe AGV'en ud af kurs og ødelægge bestiknavigationen, udstyres AGV'en med kollisionssensorer.

I sommerkurset 1992 blev der brugt hængslede kofangere med kontakter, men der var vanskeligheder med at få mekanikken til at fungere fejlfrit. I stedet for elektromekaniske komponenter anvender jeg trykfølsomme modstande, der fås som 1 mm tykke, 25 mm brede selvklæbende strimler i forskellige længder. Modstandene er simplere og mere pålidelige end kontakter og kofangere. De giver mulighed for at måle størrelsen af den kraft der påvirker dem, og ved at dele modstandene op i forskudte overlappende sektioner kan det afgøres hvor en berøring finder sted.

3.6.4 Sonarafstandsmålere

For at kunne korrigere den anslåede position vha. et kort og for at kunne kortlægge dynamisk er det nødvendigt med sensorer, der kan medvirke til at danne et billede af AGV'ens nære og fjerne omgivelser. Flere typer sensorer er i stand til dette, men den mest nærliggende er POLAROID's sonar-afstandsmåler, som er blevet brugt i sommerkurset 1992 samt i mit og Lene Monrad Petersen's bachelorprojekter.

Hver sensor har en synsvinkel på ca. 30°, hvori den kan måle afstande på 0.2 – 10 m, med en nøjagtighed på 1%. Disse egenskaber gør sensoren velegnet til at registrere fixpunkter som døre, fremspring i vægge, møbler etc., som kan bruges i forbindelse med kortnavigation. Dens evne til at se forbi objekter der kun udfylder en del af synsvinklen, gør den velegnet til kortlægning af møblerede lokaler. [31] [36] og [42] er eksempler på kortlægning og navigation vha. POLAROID's sensorer.

Jeg har gode erfaringer med POLAROID's sonarafstandsmålere fra mit bachelorprojekt, hvor jeg imidlertid ikke udnyttede deres fulde potentiale. Skildpadde AGV'en kunne maksimalt udstyres med fire afstandsmålere, og der kunne kun måles med en ad gangen. Jeg ønsker at anvende så mange sensorer at AGV'en kan se hele vejen rundt (min 12 stk.), at kunne bruge flere samtidigt, og at kunne registrere alle detekterede ekkoer.

3.6.5 laser afstandsmåler

POLAROID sensorernes største handicap er deres dårlige retningsbestemthed, der skyldes transducernes akustiske egenskaber. En laser afstandsmåler har derimod en næsten vilkårligt god retningsbestemthed.

Oprindeligt havde jeg tænkt mig at integrere en 3D laserscanner i mit projekt. Ivar Balslev havde på Fysisk Institut vejledt en række bachelorprojekter, hvor en opstilling med en liniegenererende laser og et CCD kamera gjorde det muligt at måle afstande til alle punkter, der blev skåret af laser-linien vha triangulering. Med et snævert optisk filter foran kameraet og et interface, der detekterede placeringen af max værdien på hver linie i videosignalet, kunne måleresultaterne aflæses uden *frame grabber* og beregningstunge algoritmer. Med en vandret laser-linie og ved at montere opstillingen på en styret vippeplatform, kan et stort område foran AGV'en skannes for forhindringer med stor opløsning.

Indledende tests antydede at systemet vil få problemer med normal baggrundsbelysning, og det snævre budget og tidsmangel fik mig til at skrinlægge ideen.

I løbet af projektet er firmaet SICK, gået på markedet med en laserscanner, der vha. *time of flight* måler afstande fra 0.1 – 100m, med en nøjagtighed på 2 mm. Scanneren anvender et roterende spejl og scanner

et synsfelt på 180° 20 gange i sekundet, med en 1/4° opløsning. SICK's scanner til 40.000 kr. syntes ideel til AGV formål, men ligger uden for rammerne af mit budget.

3.6.6 Doppler radar bevægelses detektor

Under sommerkurset brugte jeg en del tid på at arbejde med en doppler-radar bevægelsesdetektor. Sensorens senderenhed udsender en konstant 10GHz bærebølge. I modtageren mixes en del af det returnerede signal med bærebølgen. Hvis et punkt, der reflekterer en del af bærebølgen, bevæger sig i forhold til sensoren opstår der pga. dopplereffekten et lavfrekvent signal med en frekvens, der er proportional med bevægelseshastigheden projiceret på sensorens længdeakse. Sensoren kan således bruges til at afgøre om, der er noget eller nogen i nærheden, der bevæger sig og bruges bl.a. i automatiske døråbnere og tyverialarmer. Det dominerende problem ved at anvende sensoren på AGV'en er, at AGV'en selv bevæger sig. AGV'ens egenbevægelse vil, i et lukket lokale, give anledning til et spektrum af signaler fra DC til den frekvens der svarer til AGV'ens hastighed. Hvis hele dette spektrum blev filtreret fra af et skarpt filter eller en dynamisk fourieranalyse, kunne sensoren principielt bruges til at registrere bevægelse nær AGV'en. Jeg tager ikke sensoren med i dette projekt, men finder den meget spændende. At anvende sensoren på en AGV, vil formentligt være et passende bachelorprojekt.

3.6.7 Datamatsyn

Selv om datamatsyn er en meget lovende sensorteknologi, har jeg valgt at se bort fra den i dette projekt. Selv om det anvendte CPU-modul har en rimelig regnekraft (svarende til en 80386 baseret PC) er den langt fra tilstrækkelig til at løse de forventede billedbehandlingsopgaver dynamisk under kørsel.

Hvis AGV'en skal benytte datamatsyn, bør den udstyres med et kraftigere CPU-modul. Eftersom VME er en multiprocessor-bus, kan styredatamaten udstyres med et eller flere separate CPU-moduler, der kan arbejde med billedbehandling særskilt. For at undgå at belaste bussen med dataoverførsel mellem *frame-grabber* og CPU, kan der anvendes et CPU-modul der tillader tilkobling af I/O enheder direkte på CPU-modulets interne bus (f.eks. efter *Industry-Pack* standarden).

3.7 Opsummering

På baggrund af projektoplæget, mine ressourcer og erfaringer, udvikler jeg AGV'en **Cato** med følgende ydre karakteristika:

- Jeg genbruger en 80×60 cm. platform med hjul og motorer, som jeg forsyner med en robust letvægtsoverbygning, der afspejler AGV'ens ønskede anvendelse, som sensorplatform med plads til udvidelser og ændringer.
- Til styring af AGV'en anvendes en VME-bus baseret datamat med 68020 CPU-kort.
- Jeg udvikler et VME-modul til styring af fire uafhængige motorer.
- Jeg udvikler en sensor, der sætter AGV'en i stand til at følge en linie i gulvet med henblik på kalibrering af AGV'en til bestiknavigation.
- Jeg udvikler et VME-modul til måling med POLAROID's sonar afstandsmålere.
- IMADA's 5-aksede robotarm **SCORBOT** placeres ovenpå AGV'en og integreres med styredatamaten.
- AGV'ens energiforsyning opbygges over et enkelt opladeligt batteri.

Jeg udvikler lavniveau programmel til styring af AGV'ens komponenter med et omfang og niveau, som jeg skønner nødvendigt for at sikre en effektiv grænseflade til højniveauprogrammel i C under OS-9.

Jeg lægger vægt på, at dokumentation og brugervejledning til AGV'ens komponenter er på et niveau, der gør det muligt for andre at anvende og vedligeholde dem uden min hjælp.

3.7.1 Resten af rapporten

Resten af denne rapport beskriver udviklingen af AGV'en og dens delkomponenter. Resten af rapporten, bortset fra konklusionen, er teknisk orienteret og forudsætter forhåndskendskab til de anvendte teknologier.

3.7.2 James

Selv om jeg kun skulle udvikle en enkelt AGV, har jeg deltaget meget aktivt i udviklingen af AGV'en **James**. James er i alt væsentligt en kopi af Cato, der er nedskaleret og bygget ind i karosseriet til en fjernstyret modelbil. Der er enkelte aspekter af hardwaren til James der adskiller sig fra Cato, hvilket er kommenteret i rapporten, hvor det er relevant.

Kapitel 4

Motorstyring

Ved sommerkurset i 1992 var et af de største praktiske problemer at styre AGV'ens to DC-motorer. Problemerne bestod dels i at fremstille en stabil effektudgang til at trække motorstrømmene på op til 15 ampere, og dels i at implementere et stabilt reguleringssystem til at styre dem med.

I mit bachelor projekt (skildpadde agv), undgik jeg problematikken ved at anvende step-motorer, der kan kontrolleres nøjagtigt uden at indgå i et reguleringssystem. Motorerne var så små at de kunne drives vha. almindelige småsignal transistorer. Stepmotorer er bedst egnede til positioneringsopgaver ved små belastninger, og er egentligt uegnede til at trække køretøjer. Når det kunne lade sig gøre i skildpadden, skyldtes det skildpaddens ringe størrelse og dens lave friktion mod underlaget.

Jeg har anvendt Cato's oprindelige DC-motorer, og løst problemerne omkring effektudgange, feedback, og kontrolsystem. Det meste af motorstyringen fra Cato er genanvendt i James, der også anvender DC-motorer.

4.1 Motorer

Jeg har *arvet* to 80W 12V DC-motorer, fra AGV-kurset i 1992. DC-motorer er særdeles velegnede til AGV brug, idet de er nemme at energiforsyne, og har et stort drejningsmoment ved lave hastigheder. En af DC-motorernes største ulemper, er at de anvender kommutatorer (børster), hvilket gør regelmæssigt elektrodeskift nødvendigt.

Hvis driftsikkerhed og levetid havde været prioriteret meget højt, burde børsteløse DC-motorer, eller AC-servomotorer have været under overvejelse, idet de ikke har problemet med slid på elektroder. Til gængæld kræver børsteløse den type motorer en mere sofistikeret styring af energitilførslen.

Tabel 4.1 viser en oversigt over de motorer der anvendes i projektet, og deres egenskaber. Bemærk at forsyningsspændingen i James, af hensyn til elektronikken, er dobbelt så høj som drivmotorernes nominelle polspænding.

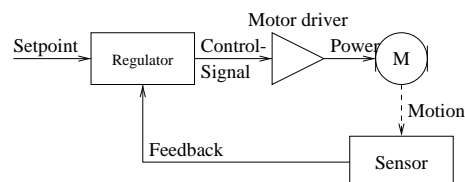
Motor	Cato's drivmotorer	James drivmotorer	Scorbot 1	Scorbot 2
Type	DC-motor	DC-motor	DC-motor	DC-motor
Nominel polspænding	12 V	7.2 V	12 V	12 V
Nominel effekt	80 W	70 W	20 W	30 W
Nominel strøm	8 A	12 A	2 A	3 A
Indre modstand	0.7 Ω	0.3 Ω	3 Ω	2 Ω
Egenskaber i opstillingen				
Forsyningsspænding	12-14 V DC	12-14 V DC*	12-14 V DC	12-14 V DC
Peakstrøm	20 A	47 A*	5 A	7 A
Peakeffekt	280 W	650 W*	65 W	100 W

* : Motorens spændingsforsyning er dobbelt så høj som den nominelle polspænding!

Tabel 4.1: Oversigt over motoregenskaber

4.2 Regulering

Da omdrejningshastigheden af en DC-motor er en funktion af polspænding, strøm, belastning, og tid (se kapitel 23), er det nødvendigt at anvende regulering for at styre kørslen af dem. Figur 4.1 viser strukturen af et typisk kontrolsystem, hvor en regulator forsøger at styre energitilførslen til motoren, så den målte feedback parameter hele tiden styres mod et setpunkt.

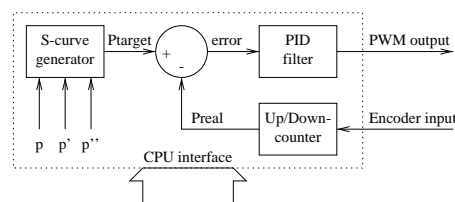


Figur 4.1: Kontrolsystem

I Sommerkurset 1992, blev det forsøgt at implementere en hastighedsregulering, vha. en simpel proportionalregulering. Motorens hastighed blev målt vha. en diskret differentiering af signalet fra en positionssensor. Positionssignalet havde imidlertid en relativt lav opløsning, der medvirkede til at gøre differentieringen enten langsom eller unøjagtig, og regulatoren kom aldrig til at virke tilfredsstillende.

Til regulering af motorer i dette projekt, anvender jeg et integreret motor kontrol system, i form af en LM629 *motion control IC* fra National Semiconductors. LM629 er en specialiseret signalprocessor, der implementerer en positionsregulering vha. en programmerbar PID regulator. PID regulatoren får sit setpunkt fra en programmerbar S-kurve generator, der også er integreret i ICen, sammen med diverse andre faciliteter. LM629 har en indbygget op/ned-tæller, og bruger inkrementalencoder-signaler til positionsfeedback. Kontrolsignalet fra LM629 er et pulsviddemoduleret digitalt signal, der let kan omsættes til et analogt signal, men som også kan bruges til at styre en *switched mode* motordriver direkte.

Selv om LM629 regulerer positionen, kan både hastighed og acceleration styres, ved at programmere S-kurve generatoren til at generere et positionssetpunkt der udvikler sig med en bestemt hastighed og acceleration. LM629 har et kommando-orienteret mikroprocessor interface, og lægger minimalt beslag på processoren i værtsdatamaten.



Figur 4.2: LM629 *motion controller*

PID filteret implementerer (4.2), der er en diskretisering af (4.1). $u(t)$ og $u(n)$ er udgangssignalet til tiden t hhv. måling n . På samme måde er $e(t)$ og $e(n)$, fejlen til tiden t hhv. måling n . k_p , k_i , og k_d kaldes for hhv. Proportional-, integral-, og differential-konstanten. Det er disse konstanter der fastlægger filterets

opførsel, og de er alle programmerbare. (4.2) er i virkeligheden en simplificering af PID filterets virkemåde, idet LM629 også rummer en programmerbar begrænsning af integralets størrelse, og en programmerbar målefrekvens for de målinger der ligger til grund for beregningen af fejltilvæksten.

$$u(t) = k_p e(t) + k_i \int e(t) dt + k_d \frac{de(t)}{dt} \quad (4.1)$$

$$u(n) = k_p e(n) + k_i \sum_{N=0}^n e(N) + k_d [e(n) - e(n-1)] \quad (4.2)$$

S-kurve generatoren, programmeres med en ønsket slutposition, en ønsket tophastighed, og en ønsket acceleration, ud fra hvilke den genererer et kontinuert, differentiabelt positionssignal (S kurve) som funktion af tiden. En kommando til generatoren fører til et positionssignal der først udvikler sig med den ønskede konstante acceleration, indtil den ønskede tophastighed opnås. Derefter udvikler signalet sig med konstant hastighed indtil det er på tide at mindske hastigheden med den ønskede konstante acceleration. Tidspunkterne for skift mellem acceleration, jævn hastighed, og negativ acceleration, er fastlagt så det genererede positionssignal ender med at være lig den ønskede slutposition. Det genererede positionssignal bruges som setpunkt for reguleringssystemet, der efter bedste evne får motoren til at køre synkront med det positionssignal der genereres.

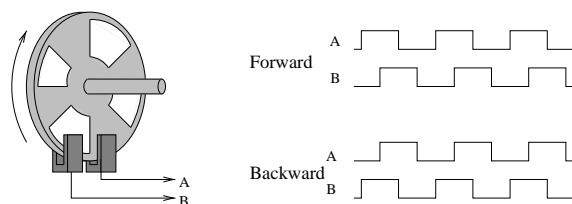
Jeg har afprøvet at motorerne kunne styres tilfredsstillende med en LM629, og derefter udviklet et VME-bus slave-modul, med 4 uafhængige LM629er. Jeg har fremstillet to fungerende kopier af dette modul, så jeg kan styre Cato's to, og Scorbotens 6 motorer samtidigt. Beskrivelsen af dette findes i kapitel 6

4.3 Feedback

Motorerne i Scorboten har indbyggede inkrementalencodere, der kan bruges til positionsfeedback. Motorerne i Cato og James har ingen indbygget mulighed for feedback. I sommerkurset i 1992 blev der brugt eksterne inkrementalencodere, der blev tilsluttet drivakslerne vha. kileremme. Inkrementalencodere er meget velegnede til at give feedback, og jeg fortsætter med at anvende dem i al motorstyringen i dette projekt.

4.3.1 Inkrementalencodere

Princippet i en inkrementalencoder er vist i figur 4.3. En op/ned tæller, koblet på enkoderens A og B udgange, kan holde rede på enkoderens position i forhold til udgangspositionen, med en nøjagtighed der afhænger af hvor mange pulser enkoderen giver pr. omdrejning, og et dynamikområde der afhænger af hvor langt tælleren kan tælle. Ved at måle frekvensen, pulsbredden, eller på anden vis differentiere signalet, kan enkodersignalerne bruges til hastighedsfeedback.



Figur 4.3: Inkremental-encoder

Inkrementalencodere kan implementeres med forskellige teknologier, men i praksis anvendes næsten altid optik i form af en skive med huller. Optiske inkrementalencodere er simple, billige, driftsikre, og med moderne optik kan der, opnås næsten ubegrænset nøjagtighed.

Inkrementalencoderens væsentligste ulempe er at den kun måler den relative vinkel. På AGV'ens drivmotorer er den absolutte vinkel irrelevant, men i en robot som Scorboten er den essentiel. På Scorboten har producenten løst problemet, ved at placere en ekstern sensor (kontakt), der aktiveres ved en helt bestemt

vinkel, på hvert led. Ved at finde dette punkt vha. en *homing*-procedure når systemet initialiseres, kan den absolutte vinkel derefter bestemmes ud fra den relative.

Homing-procedurer er upraktiske, tidskrævende, og kræver et frit arbejdsrum. Derfor ser man ofte inkrementalencodere erstattet eller suppleret af sensorer der kan måle absolutte vinkler eller positioner.

4.3.2 Montering

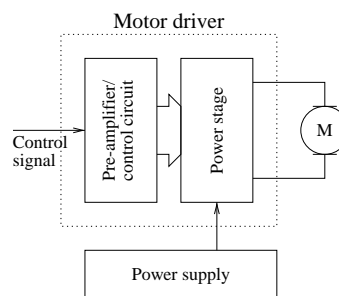
At forbinde enkoderne til drivakslen (efter evt. gearing) med en kilerem, er en nem men dårlig løsning. Kileremmen kan skride, og der introduceres *svingninger* og evt. en forsinkelse hvis kileremmen er elastisk.

På Cato har jeg forbedret monteringen ved at forbinde enkoderens aksel direkte til motorens rotor. For at realisere dette, var det nødvendigt at udbore rotorakslen, og lime en studs i, der vha. et stykke silikone-slange trækker enkoderakslen. Se figur 16.11. Den direkte forbindelse til rotoren, giver den mest pålidelige måling, og øger samtidigt opløsningen, idet bevægelsen ikke længere er gearet ned. I James er der ikke mulighed for direkte forbindelse til rotorerne. Her har man koblet enkoderne til drivakslene (efter gearing) vha. tandhjul. Cato's mekanik, og enkodernes opløsning giver en total opløsning på ca. 12000 enkoderpulser/m, mens tallet for James er ca. 1000 enkoderpulser/m.

4.4 Motordriver

Motordriveren er essentielt en effektforstærker, der tilfører motoren energi, styret af signaler fra regulatoren.

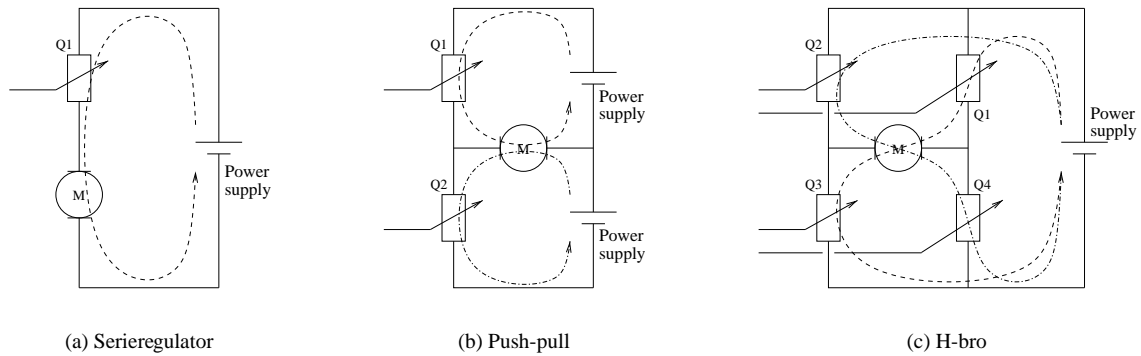
En motordriver består af et effekttrin der rummer de effektkomponenter der styrer energitilførslen, og en forforstærker der styrer effektkomponenterne ud fra det eksterne kontrolsignal. Designet af effekttrinnet afhænger af strømforsyningen, motoren, og den applikation hvori de indgår. Designet af forforstærkeren afhænger af effekttrinnet og kontrolsignalet.



Figur 4.4: Motordriver

4.4.1 Struktur af effekttrin

Alle anvendte motorer, er DC-motorer, der forsynes fra et batteri med en polspænding på 12-14V. Begrænsning af energiforsyningen sker ved at indskyde en variabel komponent i serie med motoren, som vist på figur 4.5. Figuren viser tre almindelige konfigurationer af et udgangstrin: Serieregulatoren, der kun kan sende strøm en vej gennem motoren, *push-pull* trinnet, der er bipolart men kræver dobbelt strømforsyning, og H-broen der er bipolar, men kun kræver en enkelt strømforsyning. H-broen, der egentligt er to brokoblede *push-pull* trin, bruges til energiforsyning af alle motorer i dette projekt.



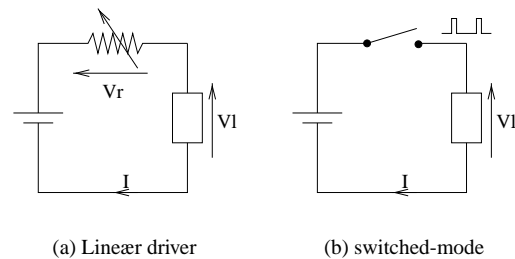
Figur 4.5: Forskellige konfigurationer af udgangstrin

Uanset hvilke komponenter og koblinger af disse der anvendes i en H-bro, kan den klassificeres som lineær, eller *switched mode*. I en lineær opstilling fungerer de energibegrænsende komponenter som variable *modstande* i serie med motoren. Energiførslen begrænses, men en del af den forbrugte effekt spildes som varmeafsætning i de begrænsende komponenter. Hvor stor en effekt der således går tabt som varme varierer mellem 0 ... 25% af *peak effekten*, afhængigt af energiførsel og belastning. Spildeffekten kan dels give problemer mht. køling, og dels udgøre et problematisk energispild i en batteridrevet opstilling. I Cato vil spildeffekten i en lineær driver udgøre op til 70W, og i James op til 160W pr. motor

$$P_{spild,max} = \frac{V_f^2}{4R_m} \quad (4.3)$$

Den lineære H-bro er en uacceptabel løsning, både pga. batteriøkonomi og køleproblemer. I stedet anvendes en *switched mode* H-bro, hvor de energibegrænsende komponenter fungerer som kontakter der enten er slukkede eller tændte. Ved at tænde og slukke med en fast frekvens, og variere pulsbredden, kan energiførslen til motoren varieres med minimal effektafsættelse i de styrende komponenter.

Et firkantsignal der veksler mellem 0% og 100%, med en frekvens f og pulsbredde d er givet ved (4.4). Som det ses er DC komponenten af signalet givet ved pulsbredden.



Figur 4.6: Driverprincipper

$$y(t) = d + \frac{2}{\pi} \sum_{n=1}^{\infty} \frac{\sin[2nd\pi] \cos[4n\pi ft]}{2n} - \frac{2}{\pi} \sum_{n=0}^{\infty} \frac{\sin[(2n+1)d\pi] \cos[(4n+2)\pi ft]}{2n+1} \quad (4.4)$$

Elektromotorer fungerer som et 2. ordens lavpasfilter, dels pga. ankerviklingens selvinduktion og indre modstand, og dels pga. motorens inertie og friktion. Ved at lægge *switch frekvensen* tilpas højt i forhold til motorens filteregenskaber, bortfiltreres grundfrekvensen og de harmoniske, så kun DC komponenten bliver tilbage.

LM629 *motion control* IC'en der anvendes i reguleringssystemet, leverer et pulsvidde moduleret styresignal med en frekvens på op til 12kHz. Da alle anvendte motorer ifølge min vurdering har knækfrekvenser på under 10Hz, vil en switch frekvens på 12kHz blive dæmpet 60 ... 120dB, hvilket er rigeligt. Følgelig vælger jeg de 12kHz som switchfrekvens.

4.4.2 Komponenter i effekttrin

Ved den givne frekvens og forsyningsspænding, kan *kontakterne* i H-broen implementeres med flere forskellige typer transistorer, hvoraf de tre mest interessante er den almindeligt kendte *bipolar junction transistor* (BJT), *Metal Oxide Semiconductor Field Effect Transistor* (MOS-FET), og *Insulated Gate Bipolar Transistor* (IGBT). BJTen og MOS-FETen er vidt forskellige, mens IGBTen nærmest er en hybrid mellem de to, med MOS-FETens indgangs- og den BJTens udgangs-egenskaber. De ydre forskelle mellem typerne går mest på deres frekvensegenskaber, effektafsættelsen i dem, og deres spændingsområde. For alle tre teknologier gælder at der findes forskellige typer, der er optimeret til forskelligt brug. I Lavspændingsområdet ved 12kHz, er de væsentligste sammenligningskriterier effekttab spændingsfald og pris.

Effekttabet i en BJT kommer primært fra spændingsfaldet over den PN overgang strømmen passerer i transistoren. Spændingsfaldet er næsten uafhængigt af strømmen og er på ca. 0.6V. Effektafsættelsen i en BJT eller IGBT bliver derfor ca. $0.6V \times I$.

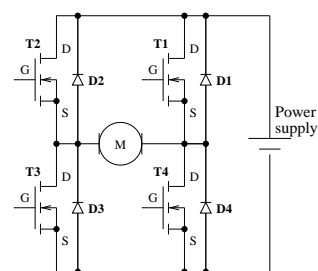
Effekttabet i en MOS-FET kommer primært fra den indre modstand i et lavt doteret område af transistorens halvledermateriale. En effekt MOS-FET er opbygget af tusindvis af mikroskopiske transistorer (*devices*) der er parallelkoblet, og den indre modstand ($R_{DS,on}$) afhænger af størrelsen og antallet af parallelkoblede *devices* i transistoren. Spændingsfaldet er givet ved $R_{DS,on} \times I$, og effektafsættelsen ved $R_{DS,on} \times I^2$.

Grænsen for hvornår en BJT/IGBT bliver en MOS-FET overlegen mht. spændingsfald og effekttab er givet ved den strøm hvor spændingsfaldet over de to typer er ens. Teoretisk kan modstanden i en MOS-FET reduceres i det uendelige, men grænsen for det økonomisk og praktisk fornuftige ligger for tiden ved ca. $5m\Omega$. En tommelfingerregel for skellelsens placering er givet ved:

$$I \times 0.6V = I^2 \times 5m\Omega \Rightarrow I = \frac{0.6V}{5m\Omega} = 120A$$

BJT/IGBTens spændingsfald på 0.6V vil i en H-bro give et totalt spændingsfald på 1.2V, eller 10% af min forsyningsspænding, hvilket vil reducere motorens maksimale drejningsmoment og hastighed, samt give anledning til et konstant effekttab på 10%. I denne sammenhæng er MOS-FET væsentligt bedre egnede, og jeg baserer derfor min H-bro på MOS-FET teknologi.

Det elektriske felt der styrer MOS-FETen genereres af spændingsforskellen mellem *Source* elektroden og *Gate* elektroden. For at anvende transistoren som *kontakt*, drives den i mætning ved at hæve *gate-source* spændingen (V_{GS}) til over den tærskelværdi producenten angiver. De fleste MOS-FET transistorerne i handelen, inklusive dem jeg anvender, har $V_{GS,on} = 10V$, og $V_{GS,max} = \pm 20V$. Som andre transistortyper, findes MOS-FET i to komplimentære varianter. N-kanal, der arbejder med positiv V_{DS} og V_{GS} , samt P-kanal, der arbejder med negative ditto. Pga. den ringere mobilitet af *huller* fremfor elektroner i halvledersubstratet, har P-kanals transistorer væsentligt større indre modstand end tilsvarende N-kanals.



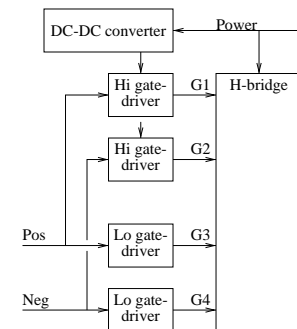
Figur 4.7: MOS-FET H-bro

Alle motordrivere i dette projekt opbygges efter modellen i figur 4.7, med 4 identiske N-kanals effekt MOS-FET transistorer. Jeg anvender forskellige transistorer i forskellige opstillinger, men deres elektriske egenskaber er stort set identiske. Med on-modstand på ca. $12m\Omega$ har de lavere spændingsfald og effekttab end BJT/IGBT ved strømme op til 50A. Da en motor udgør en induktiv belastning, er det nødvendigt at placere *freewheeling* dioder parallelt med transistorerne, for at lede selvinducerede strømme uden om slukkede transistorer. Transistorerne indeholder selv, pga. deres interne struktur, en sådan diode, der i mange tilfælde overflødiggør en ekstern monteret.

4.4.3 Forforstærker

I en MOS-FET *switch-mode* motordriver, er forforstærkerens opgave at styre de fire transistorers *gate* indgange, så motoren får tilført den ønskede firkantsspænding. Eftersom styresignalet til motordriveren i forvejen er et digitalt pulsviddemoduleret signal, reduceres problemet til at tænde og slukke for de rigtige transistorer i takt med indgangssignalet.

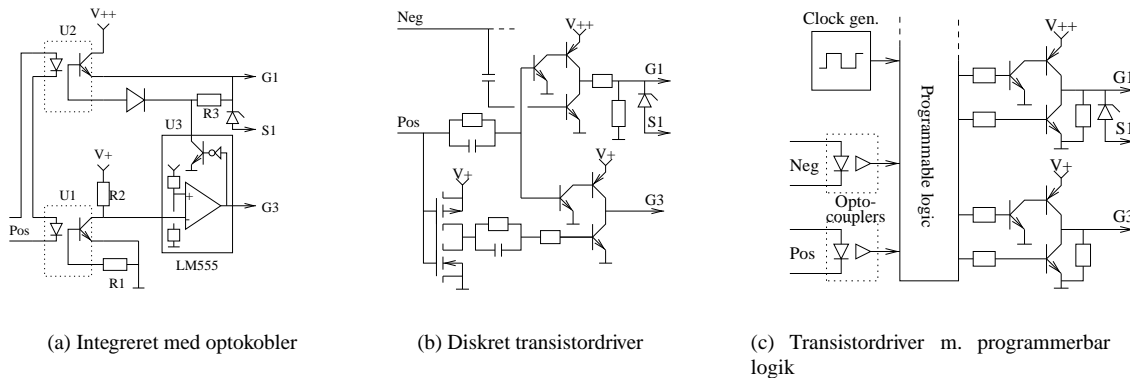
For at gøre motordriveren så simpel som muligt, har jeg designet udgangen fra motorcontrolleren, så der er to pulsviddemodulerede digitale signaler for hver motor. Det ene signal er aktivt hvis motoren skal energiforsynes den ene vej (positiv omløbsretning), og den anden hvis motoren skal energiforsynes den anden vej. I princippet opbygges forforstærkeren som vist på figur 4.8, med et udgangstrin til at drive hver *gate* elektrode. Der skelnes mellem *gate* på T1 og T2 (kaldet de høje transistorer), og T3 og T4 (kaldet de lave), fordi *source*- og dermed *gate*-spændingen for de høje transistorer offsetforskydes med forsyningsspændingen når de er tændt. Dette stiller andre krav til *gate-driveren* end T3 og T4, hvis *source* er fast forbundet til nul.



Figur 4.8: Forforstærker

I løbet af projektet har jeg arbejdet med tre forskellige typer forforstærkere, skitseret i figur 4.9.

Figuren viser forforstærkertrinnet til den *høje* og *lave* transistor i H-broen, der skal tændes for polarisere motorterminalerne positivt (T1 og T3). Alle forforstærkerne arbejder med to *gate*-spændinger. $V+$ der er mellem 12 og 20V, og $V++$ der er H-broens forsyningsspænding plus 12-20V. Alle forforstærkerne forhindrer de høje transistorers V_{GS} i at overskride $V_{GS,max}$ vha. en zenerdiode.



(a) Integreret med optokobler

(b) Diskret transistordriver

(c) Transistordriver m. programmerbar logik

Figur 4.9: Principper for forforstærkere

Opstillingen i figur 4.9-a kombinerer på elegant vis de anvendte komponenters egenskaber. Kredsløbet tænder og slukker begge relevante transistorer i H-broen synkront, hvilket kan give anledning til en ulineær sammenhæng mellem pulsbredden og spændingen over motoren (se afsnit 4.4.4).

Opstillingen i figur 4.9-b, viser princippet for forforstærkeren i motordriveren **Rungner**, som jeg har brugt en del tid på at implementere i tykfilm. Indgangssignalet bruges til at tænde både den høje og lave effekt-transistor samtidigt, men slukker kun for den lave. Den høje transistor slukkes først efter en periode uden aktivitet, eller hvis H-broen skal polariseres i den anden retning.

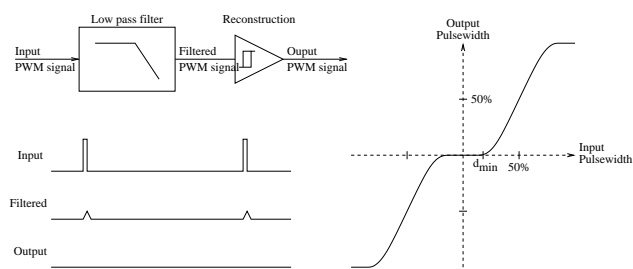
Pga. vanskeligheder og uheld med fremstillingen af tykfilmkredsløbene, havde jeg kun fået fremstillet en enkelt fungerende prototype af Rungner, da der opstod et akut behov for to motordrivere til AGV'en

James. James stiller krav til funktion og mekanisk udformning, der gør det relevant at opbygge en tredje motordriver med konventionel printteknologi. Forforstærkeren, der er skitseret i figur 4.9-c, bruger en programmerbar logikkreds til at styre de *push-pull* transistorer, der trækker MOS-FET transistorernes *gate*-elektroder. Logikken er programmeret så transistorerne i H-broen styres på samme måde som i Rungner. James forsyningsspænding er dobbelt så høj som motorens nominelle polspænding, derfor betinges/multipliceres indgangssignalet med et firkantsignal, genereret internt i den programmerbare logik. Den effektive polspænding over motoren er forsyningsspændingen multipliceret med pulsbredden af indgangssignalet og pulsbredden af det interne firkantsignal. Ved at sætte pulsbredden af det interne signal til 50% kompenseres der for den fordoblede forsyningsspænding i James¹.

4.4.4 Ulineariteter

Jeg er stødt på to fænomener ved *switched mode* motordrivere, der kan give anledning til ulinearitet i sammenhængen mellem styresignalets pulsbredde, og motorens effektive polspænding. Det drejer sig om båndbredde begrænsning af det pulsviddemodulerede signal, og motorens selvinduktion.

Båndbredde-begrænsningen, der opstår i f.eks. optokoblere, giver anledning til en ulinearitet i signaloverførselen, idet signaler med små pulsbredder ikke slipper igennem, og opfattes som permanent inaktive. Omvendt opfattes signaler med store pulsbredder som permanent aktive. Fænomenet er skitseret på figur 4.10. Kurvens præcise form afhænger af kredsløbets egenskaber, men det antages at kurvens *knæpunkter* er symmetriske omkring 50%, med det lave knæpunkt ved en pulsbredde, svarende til en pulstid på $2/BW$:



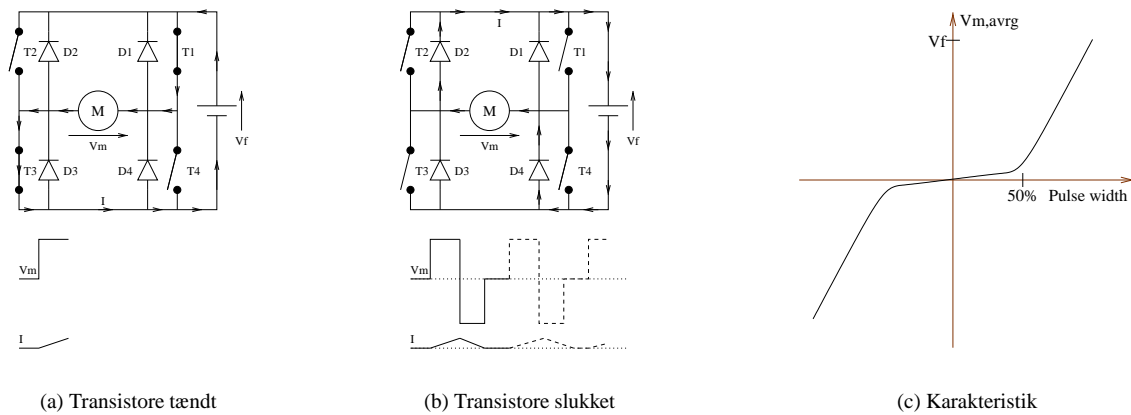
Figur 4.10: Skitse af lavpasfiltrerings virkning på PWM signal

$$d_{min} \simeq \frac{f_{pwm}}{2BW} \quad (4.5)$$

Hvor BW er båndbredden af det begrænsende element, og f_{pwm} er frekvensen af det pulsviddemodulerede styresignal.

Motorens selvinduktion, kan give ulineariteter i H-broer, hvor de høje og lave transistorer styres synkront (illustreret på figur 4.11).

¹For at øge motorens ydelse, er pulsbredden blevet øget til 62.5%

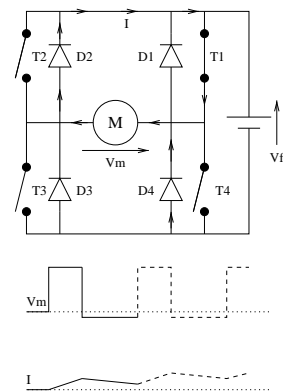


Figur 4.11: Skitse af ulinearitet pga. motorens selvinduktion

Når T1 og T3 begge er tændt, påtrykkes motoren en polspænding svarende til forsyningsspændingen, og der opbygges en strøm gennem motoren. Slukkes begge transistorer samtidigt, vil strømmen fortsætte med at løbe gennem D2, D4 og spændingsforsyningen. I denne periode svarer polspændingen over motoren til den negative forsyningsspænding plus spændingsfaldet over de to dioder, og strømmen dør ud lidt hurtigere end den blev opbygget. Hvis motorens elektriske tidskonstant er væsentligt større end periodetiden for det pulsvide-modulerede styresignal, vil der ikke kunne opbygges en væsentlig motorstrøm og den gennemsnitlige polspænding vil være tæt på nul for pulsbredder på under 50%. Ved pulsbredder over 50% vokser den gennemsnitlige polspænding lineært med forøgelsen i pulsbredde. Karakteristikkens nøjagtige form afhænger af motorens egenskaber og *switch*-frekvensen.

Hvis T1 holdes tændt hele tiden (så længe der ikke skiftes retning), vil polspændingen over motoren veksle mellem forsyningsspændingen og spændingsfaldet over D2, som vist i figur 4.12, hvilket stort set eliminerer ulineariteten.

Begge former for ulinearitet giver anledning til skabelsen af et dødbånd omkring 0% pulsbredde, så omsætningen af styresignal til effektiv polspænding forvrænges. Den praktiske betydning af denne *crossover*-forvrængning er meget afhængig af sammenhængen. I mange tilfælde vil det reguleringssystem som motordriveren er en del af, være i stand til at kompensere for en del *crossover*-forvrængning uden problemer.



Figur 4.12: T1 holdes tændt

Kapitel 5

VME-bus slave moduler

Der skelnes essentielt mellem to forskellige typer VMEbus-moduler: Slave- og mastermoduler. Slavemoduler kan ikke selv tilgå bussen, men adresseres af mastermoduler. Mastermoduler kan tilgå bussen og dermed adressere slavemoduler.

I forbindelse med dette projekt har jeg udviklet og fremstillet 2 forskellige VME-bus slavemoduler:

Gefion er en 4 akset motor styring, bygget op om 4 LM629 *precision motion controllers*. Jeg har udviklet tre forskellige versioner af Gefion. Prototypen V. 1.0 fungerede for dårligt til at det kunne betale sig at bibeholde den. V. 1.1 fungerer fint i praksis, men har et par enkelte principielle fejl, der overskrider VME-bus specifikationen. Jeg har fremstillet og bibeholdt to stk. Gefion V. 1.1. På V. 1.2 er fejlene fra V. 1.1 rettet, og der er tilføjet enkelte nye funktioner. Jeg har fremstillet et enkelt print til V. 1.2, men ikke nået at montere komponenter på det.

Heimdal er et interface til POLAROID's sonar-afstandsmålere. Heimdal findes kun i et eksemplar, prototypen V 1.0. Prototypen havde ganske vist et par småfejl, der dog alle kunne rettes tilfredsstillende vha. små indgreb på printet.

Selv om de to VME-moduler er vidt forskelligt opbygget, så er den del af dem der interfacer til VME bussen bygget over de samme koncepter. Jeg har derfor valgt af beskrive denne del af de to moduler i det samme kapitel.

5.1 VME-bussen

VMEbus er en ikke multiplexet, asynkron, parallel bus. Den understøtter 8, 16, og 32 bits CPU-, hukommelses- og periferi-enheder. VMEbus er udviklet af Motorola, og støttet af Philips, Thompson m.fl. VME er en anerkendt industristandard (IEC821BUS / IEEE P1014), der er anvendt verden over.

VMEbus er en afledning af, eller overbygning til bussen på Motorolas MC68xxx processorer. VME er derfor funktionelt meget lig 68000 bussen. De største signalmæssige forskelle ligger i busimpedanserne, bus arbitringen, og interrupt systemet.

5.1.1 VMEbus specifikation

Den fulde specifikation af VMEbus (Rev. C. 1.) [24] er et dokument på lidt over 250 sider, der bl.a. kan skaffes via **Danelec** [77]. Specifikationen rummer både den fulde tekniske beskrivelse af VMEbus, men også nyttige designregler og designforslag.

5.1.2 PEP computer

IMADA råder over 8 VMEbus baserede computere, fra *PEP modular computers*. De er opbygget i et 3U 19" rack, med et 9-slots J1 backplane. Se [26]. og kapitel 24.

5.2 Overordnede designbeslutninger

De væsentlige komponenter i ethvert I/O modul, er de periferienheder, der danner grænsefladen mellem bussen og den ydre verden. Gefion er bygget op over enhederne: MC68230 og LM629. Heimdal er bygget op over MC68230 og 74LS374. Sidstnævnte er koblet parvist, så hvert par danner en 16-bits periferienhed.

Som udgangspunkt er businterfacet designet ud fra tre grundlæggende krav:

1. Overholdelse af VMEbus specifikationen.
2. Overholdelse af periferienhedernes specifikationer.
3. Så god og fleksibel udnyttelse af VMEbussens og periferienhedernes muligheder som praktisk muligt.

Jeg har arbejdet ud fra følgende konkrete designkrav.

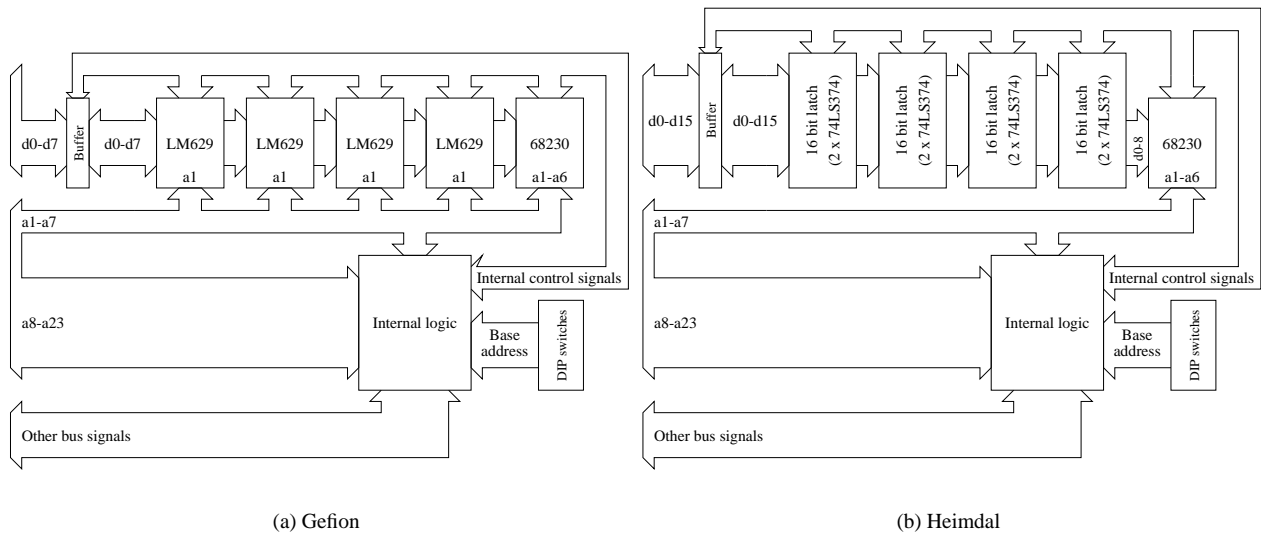
- Modulerne skal kunne adresseres med *Standard Supervisory Data Access* og *Standard Non-Privileged Data Access* (24 bits adressering, *address modifier* \$3D og \$39).
- 8-bits registre — MC68230 og LM629 periferienhederne — skal kunne adresseres som *D08(O)* (ulige byte read).
- 16-bits registre — 74LS374 periferienhederne — skal kunne adresseres som *D16* (word read)
- Modulerne må maksimalt optage et 256 bytes stort vindue i adresserummet.
- Modulernes basisadresse skal kunne konfigureres inden for hele 24-bits adresserummet, i spring på max. 256 bytes.
- Modulerne skal understøtte begge *vectored interrupts* fra MC68230 periferienheden. Interruptniveauet bør være konfigurerbart, med så mange konfigurationsmuligheder inden for området 1 . . . 6 som muligt.

5.3 Overordnet struktur

Mht. businterface er periferienhederne på Gefion og Heimdal meget lig hinanden, og den måde jeg har organiseret dem på er derfor næsten ens. Som det fremgår af figur 5.1, er den væsentligste forskel i modulernes overordnede struktur, at Gefion kun bruger de 8 laveste datalinier, mens Heimdal har koblet 74LS374 kredse i par, som 16 bits registre.

Enhed	MC68230	LM629	74LS374
Type	Paralel interface / timer (PIT)	Precision Motion Controller	Octal TRI-state flip-flop
Klasse	Motorola 68000 type periferi	Intel 8086 type periferi	standard TTL logik
Producent	Motorola, Philips	National semiconductors	National, Philips oma.
Datalinier	8	8	16 (parvist koblede)
Adresseliner	5	1	0
Registre	23 Read/Write	2 Read, 2 Write	1 Read
Bus interface	Asynkront	Synkront	Synkront
DTACK	Ja	NEJ	NEJ
Interrupts	2 (1 timer, 1 parallel)	1	0
Vectored interrupts	Ja, programmerbare vectorer	Nej	Ingen interrupts

Tabel 5.1: Egenskaber ved de anvendte periferienheder



Figur 5.1: Overordnet struktur af Gefion og Heimdal

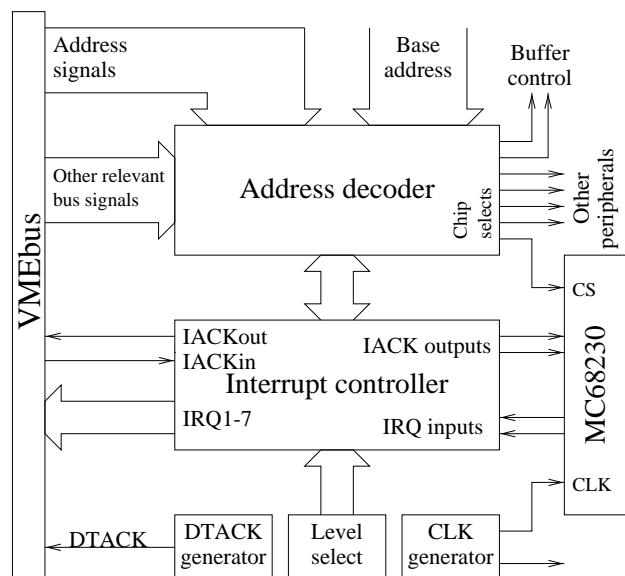
Netop fordi strukturen af modulerne er næsten ens, er funktionen af den interne logik også, stort set, den samme. Figur 5.2 viser den overordnede struktur af den interne logik på begge moduler. Funktionen af de enkelte blokke er:

Adresse dekoderen Dekoder adresse-, adresse modifier, og relevante kontrol-signaler, for at adressere periferienhederne, styre bufferen på datalinierne, og foretage andre relevante operationer når modulet adresseres korrekt.

Interrupt controlleren Danner grænsefladen mellem MC68230's og VMEbus' interruptdel. Den samarbejder med adresse dekoderen om at adressere MC68230 under en *interrupt acknowledge cycle*, hvor bussen læser en interrupt vektor fra MC68230.

DTACK generator De fire periferienheder med synkront businterface der er på hvert modul genererer ikke selv DTACK signaler ved endt dataoverførsel, hvilket så varetages af et eksternt kredsløb.

CLK generator MC68230 og LM629 skal forsynes med clock signaler.



Figur 5.2: Overordnet struktur af intern buslogik på Gefion og Heimdal

5.4 Gefion

Gefion er mit første forsøg på at udvikle et VMEbus modul. Design og implementation af bus interfacet på Gefion er i høj grad inspireret af de digitale I/O moduler fra **PEP modular computers**, VDIN [27] og VDOUT [28].

I dette afsnit beskrives udviklingen af de forskellige dele af businterfacet. Den tekniske dokumentation af businterfacet, såvel som resten af Gefion findes i kapitel 18, der indeholder diagrammer, printlayout og anden relevant dokumentation. I dette afsnit henvises ofte implicit til denne dokumentation.

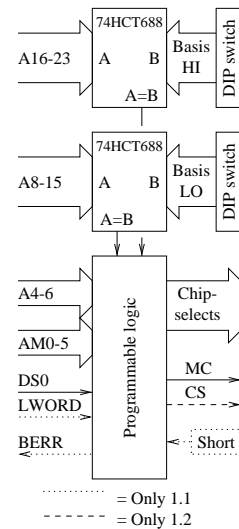
5.4.1 Adressedekoder

Adressedekoderen er opbygget meget lig adressedekoderen i PEP's digitale I/O moduler.

De 8 højeste og 8 mellemste adresselinier sammenlignes af **U1** og **U2** med indstillingerne fra 2×8 DIP switche. Hvis alle 8 datalinier i en af de to grupper matcher indstillingen på DIPswitchen signaleres dette til en programmerbar logikkreds **U3** (kildetekst i afsnit 18.7.1 og 18.7.3). Den programmerbare logikkreds overvåger også alle 5 *address modifiers* samt *data strobe 0* (**DS0**), hvilket gør den i stand til at registrere når bussen adresserer en adresse inden for et 256 bytes stort område, som en ulige byte, eller et word. På V.1.1 er det muligt at skifte mellem standard (24-bits) adressering og short (16-bits) adressering, vha. en jumper. V.1.2 anvender udelukkende standard adressering.

For at bryde 256 bytes adresserummet ned i mindre enheder overvåger den programmerbare logikkreds også $A_4 \dots A_6$, som dekodes og bruges til at generere *chip select* signaler til de 5 periferienheder. Opdelingen af adresseområdet er vist i afsnit 18.4.

På V.1.2 findes signalet **CS** der aktiveres hvis blot en af de 5 periferienheder adresseres. På alle versioner findes **MC** der aktiveres hvis en af de 4 LM629 periferienheder aktiveres. Begge dele bruges til styring af databuffere og diverse *timere*. V.1.1 overvåger *longword* signalet fra bussen, og kan selv generere en *bus error*. Begge dele er overflødige, og er ikke medtaget i V.1.2.



5.4.2 CLK generator

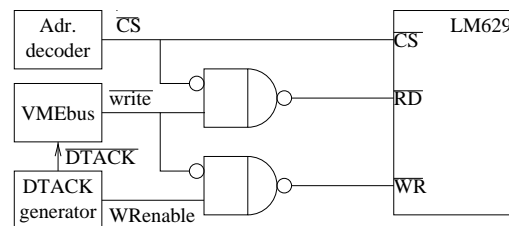
MC68230 fødes med et 8MHz clock signal, der opnås ved at dele VMEbus' 16MHz signal med 2 (**U12:A**). LM629 fødes med et 6MHz clock signal, der genereres af en selvstændig krystal oscillator (**X1**, **U5:D** **U12:B**).

5.4.3 DTACK generator og LM629 timing

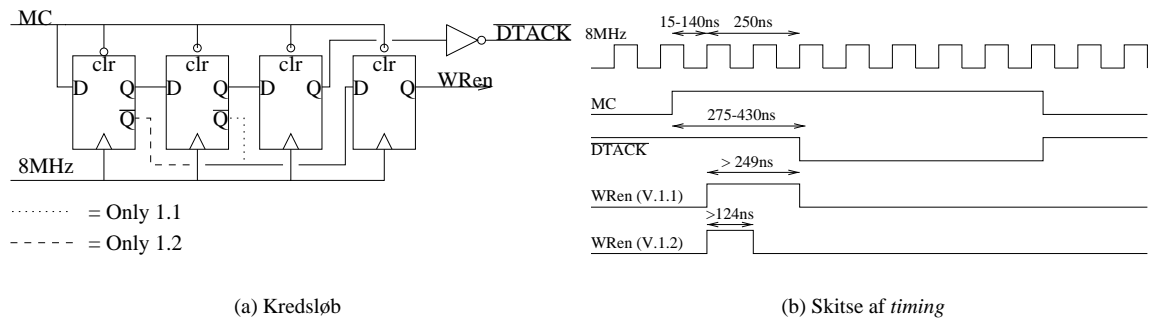
LM629 er ikke beregnet til en 68000 lignende bus, hvorfor det er nødvendigt med et eksternt kredsløb for at tilpasse den til VMEbus. Specifikationerne for LM629's businterface findes i [20]

LM629 har separate *read* og *write* signaler. Ved læsning må RD ikke aktiveres før CS, hvorfor den betinges af denne og aktiveres samtidigt (**U5:A**). Efter max. 180ns er data til rådighed fra LM629, hvorefter DTACK må aktiveres.

Ved skrivning må WR ikke aktiveres før CS, WR skal være aktiv i min. 100ns, data skal være til rådighed min. 50ns før, og min. 120ns efter WR deaktiveres. For at imødekomme ovenstående, betinges WR af et passende WRenable signal, der genereres af DTACK generatoren (**U5:B-C**).



Figur 5.3: Tilpasning af LM629 til VMEbus



Figur 5.4: DTACK generatorens opbygning og funktion

Vha. 8MHz clock signalet, og en række D flip-flops forsinkes MC signalet fra adressedekoderen — der aktiveres samtidigt med at en LM629 adresseres — i 275...430ns¹(U11), hvorefter DTACK aktiveres. DTACK generatoren genererer også det signal der betinger aktivering af WR indgangen på en LM629. Timingen på V.1.1 giver ikke den pause på 120ns mellem deaktivering af WR og aktivering af DTACK, der er nødvendig for at overholde LM629's specifikationer. Det har vist sig at fejlen ikke har praktisk betydning, men den udgør en principiel overskridelse af specifikationerne, og er rettet i V.1.2.

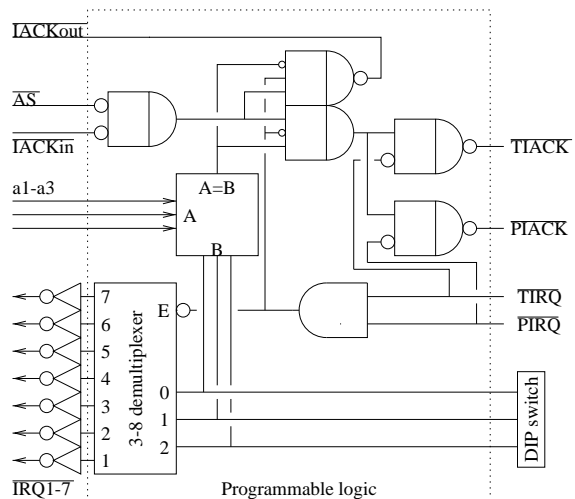
5.4.4 Interrupt controller

Interruptcontrolleren understøtter alle VMEbussens 7 interruptniveauer. Hvilket af de 7 niveauer der anvendes konfigureres som binært tal vha. 3 DIP switche. Interrupts slås fra ved at vælge 0.

Det meste af interruptcontrolleren er implementeret som en programmerbar logikkreds U4 (kildetekst i afsnit 18.7.2), hvis funktion er skitseret som blokdiagram i figur 5.5.

Hvis blot en af de to *interrupt request* indgange fra MC68230 er aktive, aktiveres det IRQ signal der er valgt vha. DIP switchen. Signalet forbliver aktivt indtil begge *interrupt request* indgange deaktiveres.

Når VMEbussen foretager en *interrupt acknowledge cycle*, signalerer $a_1 \dots a_3$ hvilket interruptniveau bussen reagerer på, og IACK daisy chainet aktiveres. $a_1 \dots a_3$ sammenlignes med DIP switchene, og hvis der er overensstemmelse samtidigt med at en *interrupt request* indgang er aktiv, og IACKin registreres, aktiveres MC68230's IACK indgang (PIACK har prioritet over TIACK) Ellers forsættes bussens daisy-chain, ved at aktivere IACKout.



Figur 5.5: Skitse af interruptcontroller

Som vist aflæser interrupt controlleren også *address strobe* (AS) hvilket har vist sig at være ganske overflødigt, eftersom AS altid aktiveres før IACK daisychainet. Jeg har bibeholdt denne *uhensigtsmæssighed* i alle versionerne, for at kunne nøjes med en enkelt udgave af den programmerbare IC der implementerer interrupt controlleren. I den første udgave af interruptcontroller ICen (U4 REV. 1.1) havde jeg byttet rundt på

¹Komponenternes *setup times* og *propagation delays* er medregnet

IRQ_n signalerne, så kun IRQ_4 kunne anvendes. Fejlen blev nemt rettet ved at ændre programmet (U4 REV. 1.1-B).

5.4.5 Databuffer

Hverken LM629 eller MC68230's dataudgange overholder specifikationerne for at drive VMEbussens datalinier [24, afsnit 6.4.2.2]. For at imødekomme specifikationen er der indsat en passende bidirektional buffer, eller tranceiver, (U22) mellem bussen og periferienhederne. Bufferen er koblet så den er aktiveret konstant. Bufferens retning styres af VMEbussens *write* signal, sammen med adressedekoderen, så data overføres fra bussen til periferienhederne, med mindre en periferienhed adresseres for læsning.

Jeg har desværre lavet en del grove fejl mht. databufferen. I første udgave af Gefion (V.1.0) havde jeg helt overset behovet for den, med det resultat at LM629 enhederne ikke kunne aflæses. I næste version (V.1.1) har jeg indsat bufferen foran LM629 enhederne, men ikke foran MC68230, der fortsat er forbundet direkte til bussen. I det konkrete tilfælde har det vist sig at læsning fra MC68230 fungerer udmærket på trods af den manglende buffer, men der er tale om en graverende overskridelse af specifikationen, hvorfor fejlen er rettet på V.1.2.

5.5 Heimdal

Tidsmæssigt er Heimdal designet og udviklet imellem Gefion V.1.1 og V.1.2. Heimdals businterface er funktionelt meget lig Gefions, men der er markante forskelle i implementationen, der dels skyldes min større erfaring med VME, men især at jeg i mellemtiden havde fået adgang til en ny type programmerbare logikkredse, der er væsentlig mere avancerede, end den type jeg havde adgang til ved Gefion.

Hele Heimdals businterface er opbygget vha. en enkelt ispLSI 1016 (U20), tre bufferkredse, og to 8-bits digitale komparatore. Programmet til U20 er gengivet i afsnit 19.14.1. Funktionelle diagrammer af U20's bus kontrollogik findes i afsnit 19.14.1.

5.5.1 Adressedekoder

Som på Gefion sammenlignes de højeste 16 adresselinier med indstillingen af basisadressen på 16 DIP-switches (U18-U19). Hvor Gefion dekode de 8 højeste og 8 mellemste adresselinier hver for sig, med henblik på mulighed for A16 adressering, dekodes de 16 adresselinier samlet, så Heimdal kun kan bruges ved A24 adressering. Denne indskrænkning i fleksibilitet er foretaget pga. mangel på indgange på den programmerbare logikkreds.

Den kombinatoriske logik i U20, overvåger adresse modifiers, de lave adresselinier, begge *data strobe* signaler, og *write* signalet, for at bryde det 256 bytes store adresseområde ned i mindre blokke. Heimdals *memory map*, gennemgået i afsnit 19.6, er i struktureret på samme måde som Gefions, med den undtagelse at de lige adresser nu også er til rådighed, og der kan skelnes mellem læsning og skrivning. MC68230 periferienheden er, som på Gefion, tildelt de ulige adresser i området 0...\$3F, mens de fire resterende periferienheder, okkuperer både de lige og ulige adresser i området fra \$40...\$7F. De adresseres imidlertid kun ved *word* adgang, og kun ved læsning.

5.5.2 CLK generator

Som ved Gefion genereres et 8MHz clock signal til MC68230, ved at dele VMEbus clock med 2.

5.5.3 DTACK generator

De fire 16 bits registre genererer ikke selv DTACK signaler ved dataoverførsler, men har heller ikke brug for så sofistikeret timing som LM629erne på Gefion. U20 genererer vha. et par interne flip-flops et DTACK signal 125. . .200ns efter *chip select* til disse registre, hvilket er tilstrækkeligt for registre og databuffer til at drive VMEbussens datalinier.

Af hensyn til interruptcontrolleren genererer DTACK generatoren også DTACK ved *byte* adressering af de ubenyttede lige adresser over for MC68230's ulige adresser, hvilket tillader VMEbussen at adressere dette *tomme* område.

5.5.4 Interrupt controller

Interrupt controlleren er implementeret i samme kreds som reseten af kontrol logikken til businterfacet, men er fuldstændig ækvivalent med Gefions interruptcontroller, bortset fra to ting.

1. Pga. mangel på ledige forbindelser til U20, har jeg undladt muligheden for at generere interrupts på niveau 7.
2. Interruptniveauet er programmerbart.

Hvis jeg havde anvendt en programmerbar logikkreds med flere I/O forbindelser, kunne den uden problemer have fungeret som en simpel periferikreds, med dataregistre til konfiguration af interruptniveau mv. Det har af praktiske årsager ikke været muligt i dette projekt, men jeg har anvendt en anden, og mere primitiv metode til konfiguration af interruptniveauet.

I lighed med Gefion bestemmes interruptniveauet af et 3-bits binært tal. På Gefion kom de tre bits fra en DIPswitch, men på Heimdal genereres de internt i U20, vha. en 3-bits binær tæller. Da både interruptcontroller og adressedekoder er implementeret i samme kreds, var det nemt at implementere tælleren så den nulstiller hhv. tæller op, ved skrivning hhv. læsning til de ubenyttede lige adresser over for MC68230's ulige.

5.5.5 Databuffer

Som på Gefion er der behov for en buffer på datalinierne mellem bussen og MC68230. 16-bits registre kan implementeres så de kan trække bussens datalinier direkte, men da der under alle omstændigheder skal anvendes en buffer, og da det er noget billigere at implementere registre med almindelige udgangstrin, har jeg valgt at bruge bidirektionale buffere til alle 16 datalinier (**U16-U17**).

Bufferne aktiveres når bussen adresserer det område der okkuperes af MC68230 for læsning eller skrivning, eller hvis bussen adresserer området der okkuperes af 16-bits registre, for læsning. Bufferens retning styres direkte af bussens *write* signal.

For at tage hensyn til *propagation delay* i bufferen, forsinkes *chip select* signalet til MC68230 i 125. . .200ns i forhold til aktivering af bufferen.

5.6 Implementation

Begge VME interfaces er implementeret som en integreret del af Gefion og Heimdal. Udviklingen af disse moduler er hver for sig beskrevet i kapitel 4 og 7. Den tekniske dokumentation over de to moduler, med diagrammer printlayout, programudskrifter m.m. findes i hhv. kapitel 18 og 19

5.7 Afprøvning

Jeg har foretaget en meget simpel afprøvning af VME interfacet på Gefion og Heimdal. Afprøvningen har haft til formål at overbevise mig om at modulet fungerer tilfredsstillende, eller afdække eventuelle fejl og problemer med modulet. Den udførte afprøvning er overfladisk, og kan ikke sammenlignes med den grundige og systematiske afprøvning der forventes af et professionelt industrielt produkt.

5.7.1 Instrumenter

Til den del af afprøvningen der indbefatter målinger direkte på elektronikken, er et af de vigtigste instrumenter en logikanalysator. IMADA råder over en 100MHz (10 ns opløsning) logikanalysator, hvilket er fuldt tilstrækkeligt til måling på VME bussens signaler. Logikanalysatoren har fire indgange ved 100MHz sample frekvens, 8 indgange ved 50MHz samplefrekvens, og mulighed for op til 32 indgange ved 12.5MHz sample frekvens. Analysatoren kan sættes til at *trigge* ved forskellige simple kombinationer af logikniveauer på indgangene.

Selv om VME-modulet består af ren digital elektronik, er det alligevel nødvendigt at anvende et oscilloskop, for at teste spændingsniveauer, afdække uønskede svingninger og pulser på strømforsynings og signalledere, etc. IMADA råder over et 50MHz storage oscilloskop, hvilket er tilstrækkeligt til de mest almindelige målinger. Nogle af de anvendte komponenter kan arbejde med signaler på op til 80MHz, og der kan principielt opstå problemer med signaler og pulser der er for højfrekvente/hurtige til at kunne detekteres med det anvendte måleudstyr.

For at foretage afprøvningen monteres VME-modulet i en VMEbus computer. For at lette tilkoblingen af måleprober etc. har jeg lavet et print i 3U format, der fungerer som 96 polet forlænger, så signalerne på backplanet i en PEP computer, er til rådighed ved frontet. Det er vigtigt at bemærke at forlængerens har begrænset anvendelse, da der ikke er foretaget nogen form for impedanstilpasning af forlængerens til *backplanet*, hvilket kan medføre refleksioner og forvrængninger af signalerne, der ikke ville opstå under normale omstændigheder. Det er muligt at købe professionelle forlængere til VMEbus, der er impedanstilpassede, har nemt tilgængelige målepunkter på alle signaler, og har mulighed for at afbryde/tilslutte VME signalerne enkeltvis vha. jumpere eller kontakter.

VME computeren skal kunne adressere modulet, med de adresseringsmetoder de er designet til. Dette foregår typisk fra PEP computerens indbyggede debugger (PEPBUG), eller fra et simpelt C program under PEPernes operativsystem OS-9.

5.7.2 Indledende målinger

Inden der monteres komponenter på et print, ses det efter for utilsigtede kortslutninger og afbrydelser.

Som noget af det første testes det at spændingsforsyningen til modulet er i orden, der testes for DC-niveauer, og for AC-variationer. Da AC-variationer typisk vil optræde synkront med busoperationer, måles spændingsforsyningen til kortet med et oscilloskop, der sættes til at trigge på forskellige bussignaler, typisk: DS0, DS1, AS, og DTACK.

Det undersøges om VME-modulets clocksignaler er i orden.

Det sikres at VME-modulet under test, ikke er sat til et adresseområde der allerede bruges af testcomputeren. Dernæst undersøges om testcomputeren fungerer normalt, med VME-modulet monteret.

5.7.3 Adressedekoder

Føst undersøges om det kredsløb der dekode de øverste 16 adresselinier² fungerer. Logikanalysator eller oscilloskop forbindes til $\overline{A=B}$ udgangene på de to 74xx688 ICer der står for den del af dekodningen. Det testes at udgangen aktiveres når en adresse i modules 256 bytes adresseområde adresseres. Der testes med modulet sat til forskellige basisadresser, sådan at alle bits i basisadressen har været både 0 og 1 under en test. $\overline{A=B}$ signalet bruges som *trigger* for de fleste efterfølgende målinger af modules funktion, idet signalet er en forudsætning for andre aktiviteter i addressedekoderen, og fordi signalet genereres af et selvstændigt kredsløb der ikke påvirkes af modules interne signaler.

Som det næste testes det at *chip select* signalerne til MC68230, og de fire andre periferienheder genereres korrekt. Samtidigt testes styresignalerne til bufferen på datalinierne, og på Gefion testes *trigger signalet* til DTACK generatoren (MC). Testen udføres med alle mulige adresseringsmetoder.

5.7.4 Periferienheder

Det testes at MC68230 periferienhederne kvitterer adressering ved at aktivere deres \overline{DTACK} udgang, og dette medfører at VME bussens \overline{DTACK} signal aktiveres.

På Gefion testes det at \overline{CS} , \overline{RD} , og \overline{WR} signalerne fungerer som de skal, og det testes om DTACK generatoren fungerer, og aktiverer \overline{DTACK} på det rigtige tidspunkt. På Heimdal testes timingen mellem 74LM324 kredsenes \overline{OE} , og \overline{DTACK} , tilsvarende.

Det testes at bufferne på datalinierne fungerer, ved at udføre skrive, og læse operationer til VME-modulet. Dataoverførslen checkes vha. logikanalysator.

Periferienhederne testes på et højere niveau, vha. simple testprogrammer, der foretager en endeløs række dataoverførsler til periferienhederne, og hele tiden undersøger om de aflæste værdi er som forventet. Disse tests kører i mange timer, og kan afsløre en lang række problemer med støj, interferens, forkert timing etc. der ikke viser sig ved hver eneste dataoverførsel. Sådanne tests suppleres med målinger af spændingsforsyningen og signaler til/fra forskellige komponenter på modulet, vha. oscilloskop, for at sikre at spændinger og signalniveauer holder sig inden for acceptable tolerancer.

Al yderligere afprøvning af dataoverførsel til VME-modulet kræver at modules øvrige funktionalitet tages i betragtning, og er derfor en integreret del af afprøvningen af modules øvrige funktionalitet.

5.7.5 Interrupt controller

Jeg har foretaget en lavniveau test af interrupt controllerne ved — undervejs i software udviklingen — at få et simpelt testprogram under OS-9, til at sætte VME-modulet op til at generere et interrupt. Forløbet af interruptsignal og *interrupt acknowledge cycle* har jeg overvåget vha. logikanalysatoren. Således har jeg testet alle mulige interruptniveauer, og alle interrupt kilder, på både Gefion og Heimdal.

5.8 Testresultater

I stedet for en minutiøs gennemgang af testforløbet vil jeg nøjes med at opsummere de problemer som afprøvningen af VME-delen af modulet gav anledning til.

²De midterste 8 ved *short* adressering

5.8.1 Elektromekaniske problemer

Der var enkelte tilfælde af glemte lodninger på komponentsiden af printene. De fleste gav anledning til banale fejl, og var lette at finde. En enkelt, på Heimdal, gav anledning til en periodisk fejl, og blev først fundet under afprøvningen af softwareinterfacet til Heimdal.

Nogle få steder havde fejl i den fotografiske fremstillingsproces bevirket afbrydelser eller kortslutninger af printbaner. Fejlene blev alle fundet og rettet inden komponentmontage. I et enkelt tilfælde blev to printbaner på Heimdal kortsluttet af en klat tabt loddetin.

5.8.2 Adressedekoder

Adressedekodere og al anden logik der styrer VME bussens adgang til periferienhederne fungerer som det skal, med undtagelse af et interferensproblem på Heimdal og en *timing* fejl på Gefion³.

På trods af at Heimdals adressedekoder tilsyneladende fungerede som den skulle, resulterede ca. hver anden adressering af MC68230 periferienheden i en *bus error*, hvilket indikerer at MC68230 ikke har genereret et **DTACK** signal. Monterede jeg Heimdal direkte i VME backplanet, i stedet for at bruge min forlænger opstod fejlen kun en gang ud af ca. 100 adresseringer. Pålodning af 4.7 μ F tantal kondensatore over spændingsforsyningen tæt på MC68230 og den ispLSI1016 kreds der implementerer adressedekoder m.m. reducerede fejlraten yderligere med en faktor 10. Målinger med oscilloskop bekræftede at forsyningsspændingen til de to ICer ikke holdt sig inden for acceptable tolerancer ved adressering af MC68230. Problemet viste sig at skyldes for store impedanser i forsyningsvejen til ICerne, der dels får forsyningsspænding til den enkelte IC til at falde, og dels får potentialet på stelforbindelsen til at stige, hvilket kan give problemer for alle kredsløb der benytter sig af samme stelforbindelse. Problemet blev løst ved at afbryde de to ICer fra de printbaner der forsynede dem med 0 og 5V, og i stedet lave disse forbindelser vha. parsnoede ledninger trukket i lige linie fra ICerne til VME connectoren.

5.8.3 Databuffere

På Gefion V.1.0 kunne jeg ikke aflæse de forventede værdier fra LM629 periferienhedens registre, i stedet aflæste jeg mere eller mindre tilfældige tal. Årsagen til fejlen var at jeg havde overset behovet for en buffer på datalinierne mellem VME bussen og LM629. Fejlen blev løst midlertidigt ved at lave en speciel sokkel med indbygget buffer. Soklen var for bred til at alle fire LM629 kunne udstyres med den, og jeg valgte at kassere V.1.0, til fordel for V.1.1 hvor kredsløbet fra soklen var integreret med det oprindelige kredsløb.

På Heimdal opstod problemer med aflæsning af registre i MC68230. Problemerne skyldtes at indgangene til bufferne på de øverste 8 interne databits *svævede* ved aflæsning af MC68230's 8 bits registre. De svævende indgange opsamlede højfrekvente signaler, der fik bufferene til at svinge med. Den højfrekvente støj forplantede sig til de 8 lave datalinier, sandsynligvis via galvanisk kobling gennem buffernes strømforsyning. Problemet blev løst ved at montere 4.7k Ω *pulldown* modstande på de 8 *svævende* indgange.

5.8.4 Interrupt controller

Under afprøvning af interruptcontrolleren på Gefion viste det sig at kun interrupts på niveau 4 fungerede. Årsagen til fejlen var at jeg havde skiftet bufferen til $\overline{\text{IRQ}}$ signallerne ud med en anden type, og i den forbindelse havde forbundet $\overline{\text{IRQ}}$ linierne i en anden rækkefølge, hvor kun $\overline{\text{IRQ4}}$ var forbundet som på V.1.0. Jeg havde ikke foretaget den tilsvarende ombytning af udgangene på den PEEL22CV10 programmerbare logikkreds der implementerer interrupt controller logikken, og fejlen blev rettet ved at foretage disse ændringer i programmet til kredsen.

³Fejlen på Gefion er gennemgået i afsnit 5.8.5

5.8.5 Fejl uden symptomer

I Gefion V.1.1 er der to fejl der ikke giver anledning til symptomer, men som er blevet opdaget ved hhv. test af timing vha. logikanalysator, og almindelig gennemgang af designet. Begge fejl er rettet på V.1.2

Som beskrevet i afsnit 5.4.3, er pausen mellem deaktivering af $\overline{\text{WRen}}$ til LM629 enhederne og aktivering af $\overline{\text{DTACK}}$ signalet for kort i forhold til LM629's specifikationer. Denne fejl kunne give anledning til skrivning af forkerte værdier i LM629's registre.

Som beskrevet i afsnit 5.4.5, er MC68230 periferienhedens datalinier er forbundet direkte til VME bussen og ikke via den databuffer der også bruges til LM629 enhederne. Denne fejl kunne give anledning til forkert aflæsning af MC68230's registre.

5.9 Konklusion

I dette kapitel har jeg beskrevet udviklingen af VME interfacet på to forskellige VME slave moduler. Udviklingen af de to moduler som helhed er beskrevet i hhv. kapitel 4 og 7.

Slutprodukterne: Interfacet på Gefion V.1.1 og Heimdal V.1.0 repræsenterer perfekt fungerende prototyper, med nogle få kendte problemer. Problemerne med Gefion er rettet på V.1.2, men jeg har ikke prioriteret samling og afprøvning af V.1.2 højt nok til at nå at få det med i projektet.

Den underliggende årsag til samtlige problemer, er den overfladiske designfase, der var nødvendig for at få tid til at udvikle VME-modulerne overhovedet. Havde projektet primært drejet sig om VME-moduler, havde der været mulighed for en dybtgående analyse og design og afprøvning af både de digitale og analoge aspekter af VME interfacet, hvilket havde forebygget alle de registrerede designfejl.

Jeg mener at begge VME interfaces har et fundamentalt godt design, og at jeg har udnyttet den teknologi jeg har haft til rådighed godt. VME interfacet på begge moduler ville kunne bruges direkte i professionelle VME produkter, hvis printudlæget blev ændret til et EMC rigtigt design — sandsynligvis med 4-lags print.

Kapitel 6

VME-modul til motorstyring

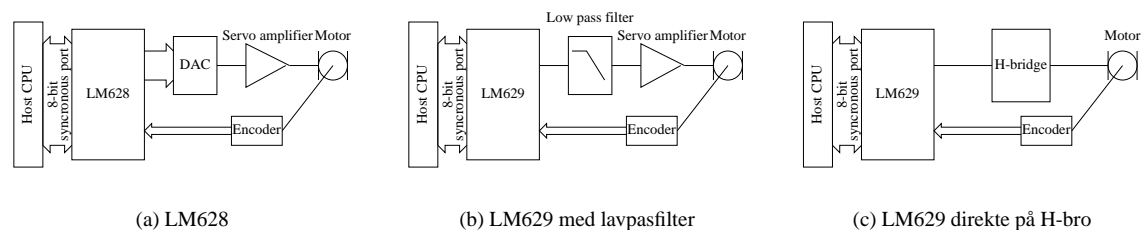
I dette kapitel beskrives udviklingen af Gefion — et fire akset motorstyringsmodul til VME-bus — fra testopstilling til fungerende VME-bus modul.

6.1 LM629

National Semiconductors LM628 og LM629 *Precision Motion Controller* er en dedikeret signalprocessor, der kan varetage reguleringsdelen i forskellige servostyringer, der anvender inkrementalencodere som feedback-sensor.

LM628/629 fungerer som periferikreds, og kommunikerer med værtsdatamaten via en synkron (INTEL 8086 kompatibel) 8-bits port[11]. LM628/629 kobles direkte på A/B, og evt INDEX signalerne fra en kvadratur/inkrementalencoder.

LM628 er beregnet til at levere et analogt styresignal til en servoforstærker, via en ekstern D/A omsætter. LM629 leverer et digitalt pulsbreddemoduleret signal, der kan omsættes til en analog værdi via et lavpasfilter, eller styre transistorerne i en *switch-mode* H-bro direkte, hvilket sparer det analoge indgangstrin i en normal servoforstærker.



Figur 6.1: LM628/629 konfigurationer

LM629's reguleringsmæssige aspekter er omtalt i afsnit 4.2 og dokumenteret i [20].

6.1.1 CPU grænseflade

LM629 har en CPU grænseflade, der gør den kompatibel med Intels periferikredse, så den med et minimum af ekstern logik kan anvendes som I/O kreds i en IBM-PC, eller anden INTEL baseret maskine.

Hardwaremæssigt består grænsefladen af 8 bidirektionale datalinier $d_0 \dots d_7$, en \overline{CS} (*chip select*) indgang, separate \overline{READ} og \overline{WRITE} indgange, en interrupt udgang, og en enkelt adresseindgang.

LM629 har med sine otte datalinier og ene adreselinie, to byte-registre for læsning, og to for skrivning.

Adresse	Læsning	Skrivning
0	Status register	kommando register
1	Data register	Data register

Tabel 6.1: *Memory map* for LM629

Softwaremæssigt, har LM629, et kommandoorienteret grænseflade, der fungerer ved at skrive en kommando-byte i dens kommandoregister, og derefter overføre kommandoens eventuelle parametre, en byte ad gangen, til eller fra dens data register. LM629 styres af 21 forskellige kommandoer, der har mellem 0 og 14 bytes parametre, struktureret som 8, 16, og 32 bits heltal og fastkomma-tal. LM629's status register kan aflæses til enhver tid, uden at forstyrre en igangværende dataoverførsel. Statusregisteret indeholder bl.a. et *busy* flag, der bruges til at teste om LM629 er klar til at modtage kommandoer, eller overføre parametre.

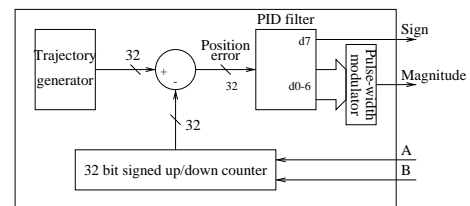
LM629 er i stand til at generere interrupts, ved 6 forskellige hændelser (*breakpoint*, *trajectory complete*, *position error*, etc.) der gør det muligt at implementere højniveau motorstyring, med en minimal belastning af CPUen.

6.1.2 Servo grænseflade

LM629 indgår som en integreret del af reguleringssløjfen i en positionsregulering, der får feedback fra en kvadraturenkoder. Positionen af kvadraturenkoderen registreres løbende af en 32 bits op/ned-tæller, der kobles direkte på A/B kvadratursignalerne fra enkoderen. Pga. de 90° faseforskydning mellem de to kvadratursignaler, måles positionen i *inkremitter*, hvor hver enkoderpuls opløses i fire inkremitter.

Den målte position fra tælleren trækkes fra den øjeblikkeligt ønskede position, fra banegeneratoren, og den resulterende positionsfejl behandles i et PID filter.

PID filteret har en 8 bits udgang, hvor fortegnet — bit 7 — er forbundet direkte til *sign* udgangen, og de resterende 7 bits, bruges til at generere et pulsviddemoduleret digitalt signal, der er forbundet til *magnitude* udgangen. *Magnitude* udgangen er moduleret med en fast frekvens, på en 512. del af LM629's clock frekvens. Pulsbredden af *magnitude* signalet varierer fra 0 til 100%, med 128 diskrete værdier, hvilket giver signalet en effektiv *samplefrekvens* på en fjerdedel af LM629's clockfrekvens.



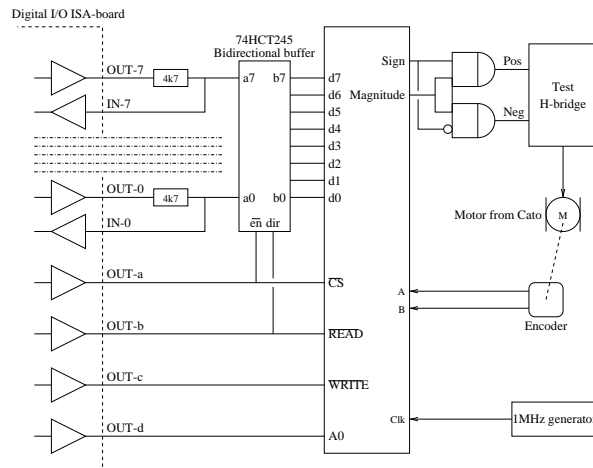
Figur 6.2: Servodelen i LM629

6.2 Testopstilling

Inden jeg påbegyndte udviklingen af et LM629 baseret VME-modul, opbyggede jeg en testopstilling med en enkelt LM629. I stedet for at forbinde den direkte til bussen i en datamat, genererede og aflæste jeg alle relevante signaler i LM629's bus-interface, vha. en simpel digital I/O port, i en IBM-PC.

Det anvendte I/O kort, havde ikke bidirektionale I/O linier, et problem der blev løst, ved at benytte 8 indgange, og 8 udgange, koblet med modstande, som vist i figur 6.3. Den bidirektionale buffer (74HCT245) er ikke nødvendig, men er medtaget for at beskytte, den ret kostbare, LM629 mod fejltilslutning, over-spændinger mv.

Jeg anvendte den lavest mulige clockfrekvens (1MHz) i testopstillingen, hvilket gav *magnitude* signalet en frekvens på ca. 2kHz.



Figur 6.3: Skitse af LM629 testopstilling

Ved at betinge *magnitude* signalet med *sign* hhv. \overline{sign} , fås to *magnitude* signaler (*pos* og *neg*), der er aktive når motoren skal køre i positiv hhv. negativ omløbsretning. *Pos* og *neg* kobles til en testopstilling af en MOS-FET baseret H-bro, med en optokoblerbaseret forforstærker som vist på figur 4.9-a. H-broen trækker en af motorene på Cato. Motoren er koblet til en enkoder, og enkodersignalerne er koblet tilbage til LM629.

6.2.1 Erfaringer og overvejelser mht. testopstilling

LM629 og testopstillingen fungerer perfekt. Der kan foretages en stabil regulering af motorens position, hastighed og acceleration, både ubelastet, og når motoren trækker Cato rundt i cirkler på gulvet med forskellige belastninger.

Når motorene kobles direkte på en *switched-mode* H-bro, opstår der mikroskopiske svingninger i motorerne i takt med skifte-frekvensen — motorene fungerer som højttalere. Svingningerne påvirker ikke motorens makroskopiske bevægelse, men den udsendte 2kHz hyletone er stærkt generende. Problemet løses stort set, ved at øge clockfrekvensen til de maksimale 6MHz. Ved 6MHz kommer skiftefrekvensen op på ca. 12 kHz, hvor lydniveauet er lavere pga. motorens mekaniske og elektriske egenskaber som lavpasfilter, samtidigt med at det menneskelige øre er mindre følsomt for 12kHz. Problemet kan mindskes yderligere ved at anvende 8 MHz versionen af LM629.

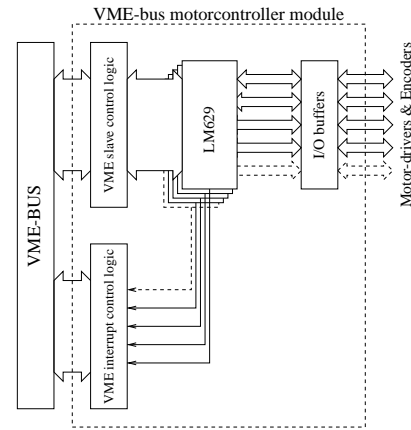
Den optokoblerbaserede forforstærker i H-broen er baseret på standard *low-cost* optokoblere, der giver opstillingen en båndbredebegrænsning på 50...100kHz, hvilket er noget lavere end den effektive samplefrekvens af *magnitude* signalet. At testopstillingen fungerer lige godt både ved $f_{clk} = 1MHz$ og $f_{clk} = 6MHz$, selvom motordriveren er markant unlineær ved 12kHz, skyldes at reguleringen formår at kompensere for de opståede ulineariteter i systemet.

6.3 Struktur af VME-modul

LM629's opbygning, som periferikreds, gør strukturen af et LM629 baseret VME-bus modul relativt enkel. Jeg tager udgangspunkt i følgende *ønsker* i designet af Gefion.

- Gefion opbygges som VME-bus slave modul, hvor CPU'en har direkte adgang til LM629 registrene via VME-bussen.
- VME-modulet skal tillade fuld udnyttelse af LM629's muligheder, inklusive interrupts.
- For at beskytte LM629 mod fejltilslutning etc. skal alle signaler mellem LM629 og motordriver/enkodere passere en *buffer*.
- VME-modulet skal rumme så mange LM629ere som praktisk muligt.

Figur 6.4 afspejler denne grundlæggende struktur.



Figur 6.4: Grundlæggende design

6.3.1 VME-bus slave interface

LM629's opbygning som periferikreds, gør at den kan kobles direkte til VME-bussen. LM629 benytter sig af en synkron databus, mens VME-bussen er asynkron, et problem der løses med lidt ekstra kontrollogik.

Detalldesignet af VME-bus slave interfacet, er beskrevet i kapitel 5.

Vilkårligt mange LM629ere kan kobles parallelt til VME-bus slave interfacet. Hvor mange LM629 der kommer på et modul, afhænger derfor primært af den plads der er til rådighed på printet, når andre nødvendige komponenter er medtaget. I praksis har der vist sig at være plads til 4.

6.3.2 VME-bus interrupt interface

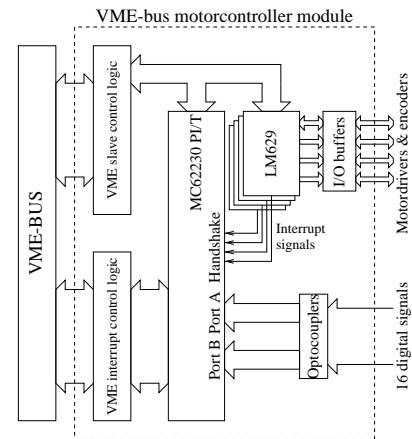
Hvor det er ret simpelt at tilpasse LM629's INTEL 8086 kompatible synkrone databus til VME-bussen, er det anderledes problematisk at integrere LM629's interrupt system med VME-bussen.

Når LM629 kommer i en interrupt tilstand, aktiverer den sin *Host interrupt (HI)* udgang, der forbliver aktiveret indtil LM629 via databussen får en *Reset Interrupts* kommando der deaktiverer den. Denne fremgangsmåde harmonerer med 8086 familien, og andre CPU'er med primitive interrupt systemer, men kan ikke direkte bruges i VME-bussen.

VME-bussens interruptsystem er væsentligt mere avanceret, idet CPUen og den *interruptende* periferienhed automatisk udveksler en *interrupt-vektor* der fortæller CPUen hvilken *interrupt service rutine* der skal kaldes pga. interruptet. [24]

I stedet for at designe kontrollogik til at udveksle en vektor med CPUen, har jeg valgt at lade en Motorola 68000 kompatibel periferikreds — der understøtter *vectored interrupts* — danne grænsefladen mellem interruptsystemet, og LM629. Jeg har valgt en MC68230 *Parallel Interface/Timer (PI/T)* [19], der kan generere *vectored interrupts*, når en af dens fire *handshake* indgange skifter tilstand. MC68230 er meget velegnet som interface mellem LM629 og VME-bussens interruptsystem, og dens øvrige funktion som digitalt I/O interface og timer, er en attraktiv tilføjelse til et motorstyringsmodul.

Detalldesignet af interrupt systemet og VME—MC68230 interfacet er beskrevet i kapitel 5.



Figur 6.5: Design med PI/T

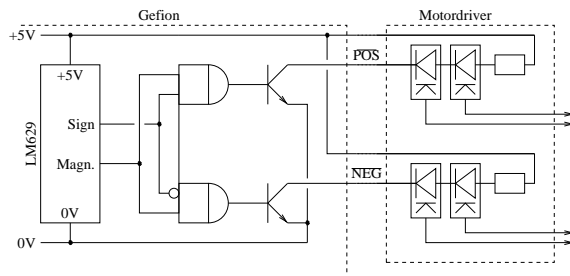
6.3.3 Digitale indgange

Med tilføjelsen af en MC68230 PI/T til designet, er der mulighed for at supplere motorstyringen med digital I/O. Jeg har valgt at bruge MC68230's port A og B, til aflæsning af 16 digitale indgange, der kan anvendes til aflæsning af digitale signaler fra endestop-følere, kalibrerings-følere, start og stop knapper etc.

6.3.4 PWM udgange

Koblingen af LM629's PWM udgange er vist i figur 6.6. Af pladsmæssige hensyn, er det ikke muligt at fortage en galvanisk isolering af PWM signalerne direkte på Gefion, og i stedet anvendes *open collector* udgange der, som vist, kan drive en eller to eksterne optokoblere, placeret på en motordriver, eller på et *interconnect board*.

Jeg fortsætter med at anvende princippet fra testopstillingen, hvor *sign* og *magnitude* ændres til et *pos* og et *neg* signal, idet det simplificerer designet af den efterfølgende motordriver.



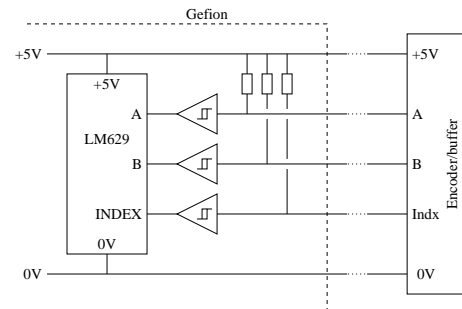
Figur 6.6: PWM udgange

På V.1.2 af Gefion bruges et mere avanceret udgangstrin, hvor det er muligt at konfigurere modulets udgange på flere måder (se afsnit 18.2.3).

6.3.5 Enkoderindgange

Der er ikke plads til de nødvendige optokoblere på Gefion, til at foretage en galvanisk adskilt overførsel af signaler fra enkoderne. I stedet anvendes almindelige bufferkredse, for at foretage en vis beskyttelse af LM629.

Indgangene er koblet med *pullup* modstande, så der kan anvendes enkodere eller eksterne buffere/optokoblere med *open collector* udgange. Enkoderne er kun mekanisk koblet til motorerne, og galvanisk adskillelse kan opretholdes ved at forsyne dem med strøm fra Gefion, som vist.



Figur 6.7: Enkoderindgange

6.3.6 Clock generator

Jeg foretrækker at give LM629erne et clocksignal der er synkroniseret med VME-bussens 16MHz clock, så flere Gefion moduler anvendt i samme VME maskine, vil køre synkront. Samtidigt ønsker jeg at anvende så høj clockfrekvens som muligt, primært for at begrænse støjgenerne fra motorerne mest muligt. Det er muligt at generere de ønskede 6MHz ud fra bussens 16MHz signal med et *phased locked loop*, men der er ikke plads til de fornødne komponenter på printet. Alternativt kan 16MHz signalet nemt deles ned til et 4MHz signal, men erfaringerne med støjgener fra testopstillingen gør at jeg vælger at droppe ønsket om synkronisering, og i stedet genererer et 6MHz signal med en krystalstyret oscillator.

6.3.7 Watch-dog

Hvis det program der styrer motorerne via LM629 standser, uden at standse motorerne — f.eks. pga. en programfejl; fortsætter motorerne den sidst programmerede bevægelse, hvilket kan have uheldige konsekvenser. På V.1.2 af Gefion har jeg inkluderet en *watch-dog timer*, der afbryder alle Gefions PWM udgange, med mindre den modtager et reset signal via PI/Ten med jævne mellemrum. Hvis softwaren opbygges så det program der styrer motorerne genererer reset signalet, vil motorerne standse hvis programmet *går ned*.

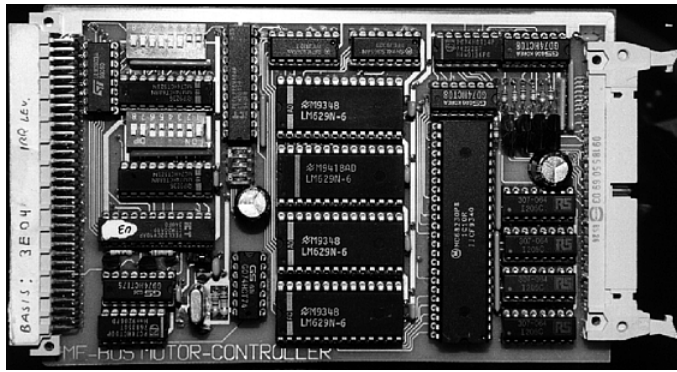
6.4 Implementation

Gefion er et 3U VME slave modul. Den fulde tekniske dokumentation af Gefion findes i kapitel 18, sammen med brugervejledning og kildetekster til softwareinterfacet.

En komplet gennemgang af alle aspekter omkring implementationen af Gefion vil føre for vidt, og i dette afsnit gives kun en kortfattet beskrivelse.

6.4.1 Kredsløbsdesign

Hele kredsløbet er digitalt, og det meste af kredsløbsdesignet har gået på at identificere og sammensætte de bedst egnede komponenter til at udføre de ret simple kombinatoriske, og sekventielle logikfunktioner der er nødvendige i VME-bus interfacet, og resten af kredsløbet.



Gefion sagdes at være Skjolds kone. Skjold var en af de gamle konger der bragte landet velstand. I sin søgen efter mere land gik hun til svenskernes konge, Gylfe, der tilbød hende så meget land, som hun var i stand til at pløje. Hun besøgte en jætte og fik fire sønner med ham, som hun skabte om til store okser, og med dem pløjede hun Sjælland ud af Sverige, så det blev til en ø, og hullet til søen Mälaren.

(Nordisk myte)

Figur 6.8: Gefion

Da jeg ikke råder over kredsløbs CAD programmer til IMADA's UNIX maskiner, eller min egen Amiga, er kredsløbet designet og tegnet manuelt. Da jeg senere fik mulighed for at låne en IBM kompartibel PC, med CAD programmet Protel, er designet overført til denne.

6.4.2 Programmerbar logik

Til at varetage de mest komplekse logikfunktioner i VME slave interfacet, og interrupt interfacet, anvender jeg programmerbar logik, i form af 2 PEEL 22CV10 kredse.

Programmeringsudstyret til PEEL kredsene befinder sig på IOT, hvor jeg har begrænset adgang til det via Søren Peder Jensen, og Jørgen Jeppe. PEEL kredsene er ikke specielt avancerede, og deres muligheder er yderligere begrænset af det faktum at IOT's programmeringssoftware ikke er i stand til at udnytte deres faciliteter fuldt ud, men kun kan bruge dem til at simulere de noget simplere PAL kredse. På trods af deres begrænsede muligheder kan PEEL/PAL kredsene implementere kombinatorisk og sekventiel logik, der havde krævet opimod 10-20 standard TTL logikkredse at implementere, og de bliver dermed en forudsætning for at kredsløbet kan implementeres på et 3U modul overhovedet, samtidigt med at de letter fejlretning og printdesign.

På grund af den lidt besværlige adgang til programmeringsudstyret har jeg kun anvendt PEEL kredse hvor det var absolut nødvendigt, og brugt standard TTL logik for resten.

6.4.3 Printteknologi

Budgettet levner ikke mulighed for at få fremstillet print professionelt, og da min erfaring fra sommerkurset-92, og mit bachelorprojekt, siger mig at kvaliteten af mine hjemmelavede print er på højde med, eller bedre, end kvaliteten af print fra IOT's og OU's værksteder, vælger jeg selv at stå for printfremstillingen.

Jeg har mulighed for at fremstille dobbeltsidede print uden gennempletteringer. Printene designes vha. Amiga printlayout-programmet **NewIO**, der er pålideligt og let at bruge, men som ikke understøtter *netlister*, *autorouting*, og *design-rule check*.

De færdige printmønstre til over- og underside overføres til klar film, og overføres ved kontaktfoto til et dobbeltsidet print, der er forbehandlet med en lysfølsom lak. Efter fremkaldning fremstår printlayoutet i den lak der sidder tilbage på printets kobberfolie, og efter ætsning fremstår printlayoutet som rene kobberbaner.

Det bedste resultat opnås ved at anvende en fotosætter, der fremstiller en målfast film direkte fra f.eks. en PostScript fil. Har man ikke adgang til en fotosætter, kan der opnås næsten lige så gode resultater ved at udskrive layoutet, i overstørrelse, på papir, og nedfotografere det med reprojekamera. Der bør ikke anvendes

laser-printer, da opvarmningen og fremføringen af papiret kan deformere udskrifterne i en grad der gør dem uanvendelige.

Det er vanskeligt at få layoutet på over- og underside til at passe nøjagtigt overens, men jeg har opnået gode resultater ($\leq 0.3mm$) ved at lægge filmen til over- og underside overens, og derefter bore små styrehuller, i to diagonale hjørner, gennem film og print samtidigt. Efterfølgende kan filmenes position genoprettes på hver sin side af printet, vha. nåle der passer i hullerne, og fastholdes vha. tape.

Når printet ikke er gennempletteret er det nødvendigt at lodde komponentben på både under og overside for at skabe de nødvendige forbindelser. Pga. den høje komponenttæthed er det ikke altid muligt at føre loddekolben ned til loddesteder på oversiden uden at beskadige nabokomponenter. Problemet løses ved at lade loddekolben varme det pågældende komponentben op fra undersiden, og foretage lodningen på oversiden ved hjælp af den varme der ledes gennem komponentbenet. Hvis der anvendes IC sokler, er det nødvendigt at bruge en type med tykke massive ben for at opnå tilfredsstillende varmeledning, f.eks. tulipansokler.

6.4.4 Printdesign

Printdesignet afspejler den mekaniske opbygning af VME, med Et 96-polet DIN stik der danner forbindelsen til VME-bussen i den ene ende, og et 50-polet IDC stik, der danner forbindelsen til den ydre verden i den anden. Komponenterne der implementerer VME interfacet sidder tæt på VME-connectoren, mens komponenterne til udgangslogik, udgangsbuffer, og den galvaniske isolering af indgangene sidder tæt på det 50-polede IDC stik. Pereferikredsene LM629 og MC68230 sidder i midten.

+5V og 0V er ført med brede baner overfor hinanden på hhv. komponent- og loddeside, for at minimere selvinduktionen i forsyningsvejen, samtidigt med at den distribuerede kapacitet mellem dem maksimeres. Forsyningsbanerne går langs hele den ene kant af printet, med vinkelrette sidebaner ind til komponentgrupperne.

Der er placeret afkoblingskondensatore mellem 0 og +5V, strategiske steder på printet, for at minimere transiente ændringer i komponenternes forsyningsspænding, ved ujævnt strømforbrug. Det er forsøgt at minimere modstand og selvinduktion af forbindelserne mellem komponenter, VME-bus, og afkoblingskondensatore.

6.5 Funktionsbibliotek

For at give adgang til at styre Gefion fra et højniveau program under OS-9, har jeg udviklet funktionsbiblioteket MCI.l, hvis funktioner styrer Gefions ressourcer, og danner en simpel grænseflade mellem et C program, og LM629's indbyggede kommandoer. MCI.l giver en nem adgang til LM629's kommandoer, men understøtter ikke interrupts, og giver derfor ikke mulighed for at udnytte Gefions fulde potentiale. Kildeteksterne til MCI.l er gengivet i afsnit 18.9. Kildeteksterne er kommenterede og fungerer sammen med [20] som dokumentation for biblioteket.

6.5.1 Grundlæggende struktur

MCI.l er i det store hele en samling funktioner der danner en *C front end* til LM629's egne kommandoer, og Gefions 16 digitale indgange.

Parameteroverførslen til LM629 foregår en byte ad gangen, og der er mulighed for at operativsystemet afbryder MCI funktionerne midt i en parameteroverførsel på flere bytes. For at undgå konflikter mellem flere processer, der ønsker adgang til samme LM629, implementerer funktionsbiblioteket en semaforbeskyttelse af hver enkelt LM629.

6.5.2 Identifikation og beskyttelse af ressourcer

Hvert Gefion modul rummer op til fem ressourcer: en PI/T, og op til fire LM629. Hver resurce identificeres ved dens startadresse i VME-hukommelsen, og beskyttes af en semafor, implementeret som et OS-9 *event*.

Til håndtering af identifikation og beskyttelse benytter MCI.I sig af en datastruktur — en nøgle, der rummer alle relevante oplysninger om hver enkelt ressource — startadresse, *event* navn, *event* ID etc. Programmer der bruger MCI.I opbevarer de fornødne *nøgler*, der initialiseres af MCI.I, og derefter bruges som identifikation af ressourcer overfor MCI.I, på samme måde som en *filepointer* bruges ved I/O kald til operativsystemer.

Nøgler initialiseres af funktionen **MCLLink**, der kaldes med basisadressen for det pågældende Gefion modul, undersøger om der findes et modul på adressen, initialiserer modulet, og fem nøgler der identificerer PI/Ten, og de fire LM629 på modulet.

Hver nøgle deinitialiseres efter brug af funktionen **MCLReleaseKey**, der deinitialiserer nøglen, den LM629 eller PI/T nøglen identificerer, og frigiver det OS-9 *event* der blev brugt til semaforbeskyttelse.

6.5.3 Styling af LM629

MCI.I rummer 24 funktioner, der bruges til at give LM629 kommandoer. De fleste kommandoer er blot en *C front end* til LM629's egne kommandoer (f.eks. **MCLResetMotor** og **MCLLoadFilter**), andre er makrofunktioner, der bruger kalder en sekvens af LM629 kommandoer, for at implementere en logisk funktion (f.eks. **MCLStopMotor**). Resten er boolske funktioner der undersøger forskellige flag i LM629's status register (f.eks. **MCLTrajectoryComplete** og **MCLBreakpointReached**).

Alle disse funktioner bruger den ovennævnte *nøgle* til at identificere hvilken LM629 der arbejdes på.

6.5.4 Styling af PI/T

MCI.I rummer to funktioner til initialisering hhv. aflæsning af PI/Tens parallelle indgange; samt tre uimplementerede funktioner til styling af timeren.

6.6 Afprøvning

I dette afsnit beskrives afprøvningen af den ikke VME-specifikke del af Gefion. Afprøvningen af Gefions VME-interface er beskrevet i afsnit 5.7.

Med et fungerende VME-interface, kan resten af afprøvningen af Gefion, foretages ved hjælp af testprogrammer som **motor**, der er gengivet i afsnit 18.8. Programmet spørger efter hvilken LM629 der anvendes, nulstiller den, spørger efter P, I, og D filtreparametre, og ønsket position, hastighed og acceleration, hvorefter den ønskede bevægelse startes, og programmet afsluttes.

6.6.1 Åben sløjfe

Motorstyringen testes først mens motoren og enkoderen er adskilt.

Enkoderen tilsluttes, og med et oscilloskop testes det at enkoderen genererer de forventede pulser på A og B udgangen, når akslen drejes. Det testes at enkodersignalerne når gennem Gefions *buffer*, til enkoderindgangene på LM629.

Gefions *Pos* og *Neg* PWM udgange kobles via $1k\Omega$ *pullup*-modstande til +5V, og med et tokanals oscilloskop overvåges udgangene parvist. Det testes at alle *Pos* og *Neg* indgange er inaktive (+5V) umiddelbart efter datamaten med Gefion tændes. Vha. testprogrammet **motor**, testes en LM629 derefter en ad gangen.

I **motor** sættes P til en lav værdi (f.eks. 10), mens I, D, position, hastighed, og acceleration alle sættes til 0. Den genererede pulsbredde vil nu være proportional med enkoderens bevægelse fra udgangspositionen, med P værdien som proportionalitetsfaktor. Det testes at pulsbredden vokser når enkoderen drejes, og at fortegnet skifter omkring enkoderens udgangsposition.

Testen gentages med motordriver og motor tilkoblet, og det testes at motorens hastighed/moment vokser når enkoderen drejes fra sin udgangsposition, og at motorens bevægelse altid vil dreje enkoderen tilbage mod sit udgangspunkt hvis de kobles sammen.

6.6.2 Lukket sløjfe

Motoren og enkoderen kobles sammen, og servosystemet som helhed afprøves vha. testprogrammet **motor**.

Ud fra kendskab til enkoderopløsning og motoregenskaber, beregnes positions og hastighedsparameter der svarer til at motoren vil bevæge sig med ca. 50% af tophastighed i ca. 30 sekunder. For Cato's motorer: ca. 450000 inkremitter med 5.0 inkremitter/sampleperiode. Sammen med en accelerationsparameter der er urealistisk høj (> 1.0 for Cato's motorer), vil parametrene generere et rampeinput til LM629's positions-servo.

Parametrene for position, hastighed, og acceleration, bruges ved stigende værdier for P. For små værdier af P, skal motoren starte langsomt, køre i ca. 30 sekunder, og bremse langsomt ned. Værdien for P øges indtil motoren starter voldsomt, og falder til ro ved en jævn hastighed, efter nogle hastighedsmæssige svingninger (ringning). Herefter justeres accelerationsparameteren ned indtil ringningerne forsvinder.

Gefion er blevet afprøvet i lukket sløjfe, med motorer fra Cato, James, og Scorbotten, og jeg har foretaget en eksperimentel tilpasning af reguleringsparametrene for de forskellige typer.

6.6.3 Digitale indgange

Der sendes en strøm på 10mA gennem hver enkelt af de 16 optoisolerede indgange, og det testes at den tilsvarende bit i MC68230's registre for port A og B, aflæses som 0, mens resten aflæses som 1

6.6.4 Testresultater

Gefions I/O del fungerer som ventet, og på nær et par glemte lodninger på komponentsiden, blev der ikke fundet fejl eller uhensigtsmæssigheder.

De reguleringsparametre jeg har fundet bedst egnede for de forskellige motortyper er gengivet i table 6.2

Motor	P	I	D	I-limit	D sample interval	acceleration
Cato	100	5	2	20000	255	0.02
James	500	20	5	10000	100	0.01
Scorbot (base)	2000	200	100	20000	255	0.0004
Scorbot (Shoulder)	2000	200	100	20000	255	0.0004
Scorbot (Elbow)	2000	200	100	20000	255	0.0004
Scorbor (Wrist1)	5000	50	200	20000	255	0.0005
Scorbor (Wrist2)	5000	50	200	20000	255	0.0005
Scorbot (Gripper)	2000	0	0	N/A	N/A	0.0005

Tabel 6.2: Reguleringsparametre

6.7 Konklusion

Det er lykkedes at udvikle et velfungerende 4-akset VME-bus motorstyringsmodul — Gefion, der er velegnet til brug i eksperimenter, testopstillinger, mv. i undervisning og projekter. Gefion har været brugt til styring af Cato, Scorbot, James, og en løbekran til demonstration af tidsoptimal kontrol.

Gefion er veldokumenteret, og er i forbindelse med DTU-RoboCup 1997 og 1998 blevet anvendt af andre studerende, med minimal hjælp fra min side.

Ved projektets afslutning eksisterer der to eksemplarer af Gefion Version 1.1. Version 1.1 har nogle fejl og uhensigtsmæssigheder i VME-bus interfacet, der ikke har praktisk betydning — se kapitel 5. Fejlene er rettet i version 1.2, men da version 1.1 fungerer fint i praksis, har jeg prioriteret fremstilling af V.1.2 lavere end andre dele af projektet, hvorfor der ikke foreligger et samlet eksemplar.

Design og afprøvning af Gefion må betragtes som overfladisk i forhold til de krav der stilles til professionelle VME produkter, og modulet er derfor kun egnet til brug i opstillinger der ikke stiller krav til person- og driftssikkerhed. Modulets arkitektur, med direkte adgang til LM629's registre, gør imidlertid Gefions LM629 delkompatibelt med en række LM628/LM629 baserede industrielle produkter. Gefion kan relativt nemt udskiftes med et professionelt produkt, hvis et projekt udvikler sig i industriel retning.

6.7.1 Software

Funktionsbiblioteket MCI.1 er et udmærket *C front-end* til Gefion, der understøtter hele Gefions funktionalitet, bortset fra interrupts og MC68230's *timer*. Den manglende understøttelse af interrupts har ikke givet problemer i dette projekt, men vil givetvis blive et problem for mere avancerede anvendelser.

MCI.1 har dækket mine egne behov, og selv om jeg erkender fordelene ved en interrupt baseret OS-9 devicedriver, har jeg prioriteret udvikling af en sådan, lavt i forhold til andre dele af projektet.

6.7.2 Videre udvikling

Gefion V.1.2 vil — efter samling og test af en prototype — kunne fremstilles som dobbeltsidet gennempletteret print, hvilket vil afhjælpe den besværlige loddeprocedure. Med gennempletteret print, vil et Gefion modul kunne samles på fire timer, til en materialepris på ca. 4000 kr. Da materialeprisen svarer til ca. 1000 kr. pr. styret motor, er der et rimeligt incitament til at anvende Gefion til undervisning, frem for tilsvarende professionelle produkter, der koster ca. 3000 kr. pr. styret motor.

Med lidt forhåndskendskab til OS-9, 68000 maskinkode, og med udgangspunkt/støtte i kapitel 8, afsnit 19.12, kapitel 5, kapitel 18, samt [13], [15], [16], [19], [20], og [21]; vil det være relativt nemt at udvikle

en interrupt baseret device driver til Gefion. Mht. overordnet design af en sådan device driver, vil man med fordel kunne søge inspiration i softwareinterfacet til andre motorstyringer¹

Design og udvikling af en device driver til Gefion, kan formentligt ske som en del af et speciale eller afgangprojekt, som et bachelorprojekt, eller som selvstændigt studieprojekt.

¹Danelec[77] forhandler flere forskellige LM628/629 baserede motorstyringsmoduler, og skaffer gerne dokumentation

Kapitel 7

VME sonar- interface

POLAROID's system til berøringsfri måling af afstand til faste objekter vha. ultralydsbaseret sonar, har en ganske enestående kombination af egenskaber der gør det velegnet til brug i dette projekt.

Jeg har udviklet et VMEbus I/O modul der er specielt beregnet på at tilkoble op til 16 af POLAROID's sensorer, med fuld udnyttelse af sensorernes muligheder. Jeg har valgt at kalde dette VME-modul for Heimdal.

7.1 POLAROID's sonar-ranging system

POLAROID sælger et system til sonar-baseret afstandsmåling. Systemet har en rækkevidde på ca. 11m, og er beregnet til autofokusering af POLAROID's kameraer. Systemet er effektivt, pålideligt, nemt at bruge, billigt, og relativt småt, hvilket har gjort det særdeles populært som sensor indenfor robotteknologi i almindelighed, og AGV-teknologi i særdeleshed.

7.1.1 Beskrivelse

Hjertet i sonar-systemet er et såkaldt *sonar-ranging module*, et 45×60 mm print, der rummer to specialfremstillede IC'er, oscillator, transformatorer, mv. til at drive en lille elektrostatisk transducer. Strømforsyning og alle signaler til/fra *ranging* modulet, føres i et 9-polet striplinekabel.

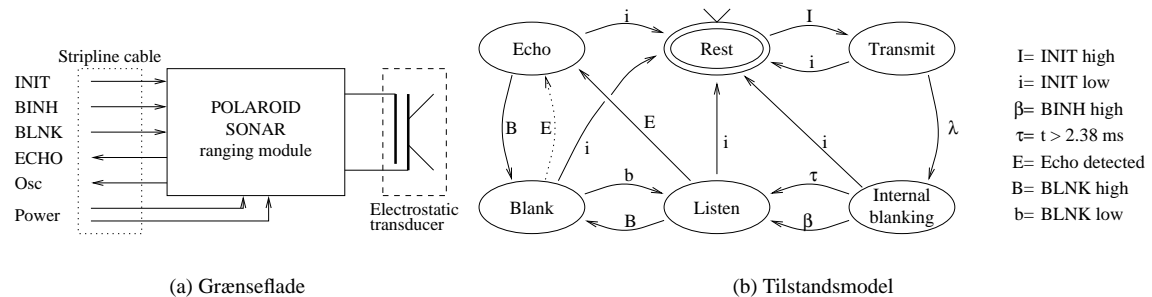
Den elektrostatiske transducer har form af en 10mm tyk skive, med en diameter på 43mm. Den består af en membran af metalliseret kunststof, der er spændt op overfor en bagplade af metal, og beskyttet af en perforeret forplade. Transducere har to tilslutninger (membran og bagplade), der forbindes til *ranging* modulet vha. små kabelsko. Der findes forskellige varianter af transducere, men de er alle kompatible, og afvigelserne går mere på montering end elektriske og akustiske egenskaber.

Når der skal foretages en måling driver *ranging* modulet transducere med 16 svingninger af et $400V_{p-p}$ signal, med en DC-offset på 200V, og en frekvens på 50kHz, hvorved membranen bringes i svingninger og frembringer en lydimpuls. Når membranen falder til ro, bruges transducere som elektrostatisk mikrofon, idet DC-spændingen opretholdes og fungerer som forspænding.

Ranging modulet sælges for ca. 250kr, og transducere for ca. 50kr.

7.1.2 Model

Dokumentationen af *ranging* modulet er ikke så detaljeret som jeg kunne tænke mig, og den er også be- hæftet med et par vitale fejl, eller udeladelser som jeg vender tilbage til senere. Ud fra dokumentationen og mine tidligere erfaringer med sensoren, har jeg opstillet følgende model for sensorens funktion:



Figur 7.1: Model af POLAROID's sonar-ranging module

Figur 7.1-a viser det elektriske interface til *ranging* modulet, der består af strømforsyning, tre digitale indgange og to digitale udgange. Det er vigtigt at bemærke at **ECHO** udgangen er en *NPN open collector* udgang, selv om dokumentationen leder en til at tro at det drejer sig om en normal *TTL totem-pole* udgang, ved at kalde modulet "TTL kompartibelt."

OSC udgangen er blot et udtag fra modulets interne clockgenerator, og kan bruges som tidsreference ved målinger, men har ellers ingen relation til modulets funktion.

Figur 7.1-b viser funktionen af *ranging* modulet, som en tilstandsmaskine, hvis tilstande og funktion beskrives herunder.

- Når indgangen **INIT** er lav, holdes modulet i en inaktiv *hviletilstand*. Uanset hvornår **INIT** sættes lav, skiftes øjeblikkeligt til denne tilstand.
- En måling påbegyndes ved at sætte **INIT** høj. Modulet befinder sig i *transmit* og *intern blanking* tilstanden samtidigt. *Transmit* tilstanden varer indtil modulet har afgivet sine 16 pulser til transduceren.
- *Intern blanking* tilstanden varer indtil der er forløbet 2.38ms fra **INIT** gik høj, eller indtil **BINH** (*Blanking INHibit*) sættes høj. I begge tilfælde skiftes til *lytte* tilstanden.
- Modulet bliver i *lytte* tilstanden indtil der registreres et ekko, hvorefter der skiftes til *ekko* tilstanden. Det er muligt at skifte til *blank* tilstanden ved at aktivere **BLNK**, men denne mulighed har ingen praktisk anvendelse.
- *Ekko* tilstanden er den eneste tilstand hvor modulets **ECHO** udgang er høj. I alle andre tilstande er udgangen lav. Modulet bliver i *ekko* tilstanden indtil målingen afsluttes ved deaktivering af **INIT**, eller indtil **BLNK** (*BLaNK*) indgangen aktiveres, hvorefter der skiftes til *Blank* tilstanden.
- Modulet bliver i *blank* tilstanden indtil **BLNK** indgangen deaktiveres, hvorefter modulet vender tilbage til *lytte* tilstanden. Selv om det ikke fremgår af dokumentationen, har det vist sig at modulet kan udsende kortvarige sporadiske høje pulser på **ECHO** udgangen mens **BLNK** er aktiveret. Dette optræder når **BLNK** aktiveres før alle svingningerne af ekkoet har nået sensoren. Dette passes ind i modellen ved at antage at de resterende svingninger får modulet til, kortvarigt, at vende tilbage til *ekko* tilstanden, der forlades med det samme, da **BLNK** stadig er aktiv.

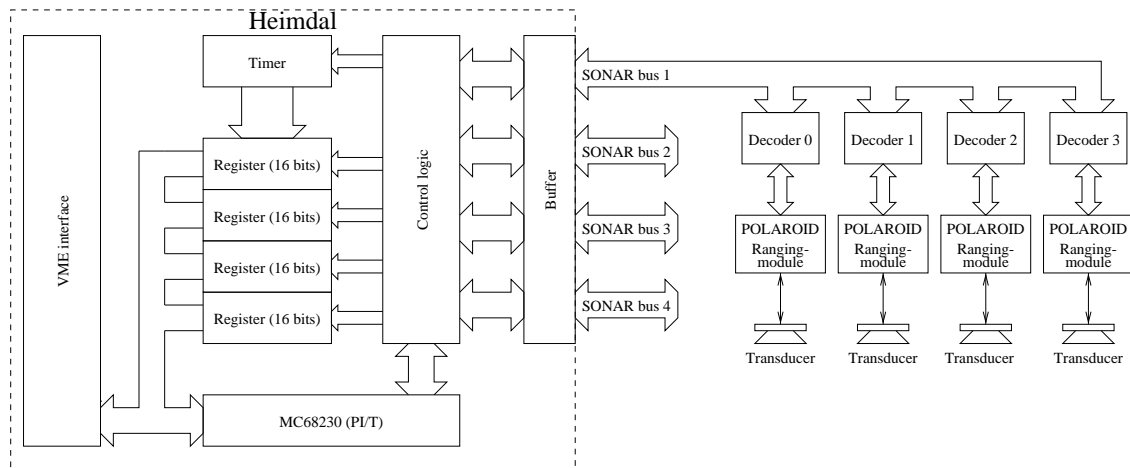
Systemet kan anvendes til en meget simpel afstandsmåling ved at aktivere **INIT**, og måle hvor lang tid der går før **ECHO** aktiveres. Denne metode kan kun registrere det første ekko. Vil man detektere flere ekkoer, aktiveres **BLNK** for hver modtaget ekko, hvorefter sensoren kan registrere et nyt.

For at udelukke at eftersvingninger i transduceren registreres som et ekko, går systemet normalt ikke i *lytte* tilstanden før 2.38ms efter **INIT** aktiveres. Vil man registrere ekkoer før dette tidspunkt, kan modulet tvinges i *lytte* tilstanden, ved at aktivere **BINH** indgagen.

For at kompensere for dæmpningen og spredningen af lyden, arbejder modtagedelen af *ranging* modulet med progressiv forstærkning. I løbet af de første 38ms efter **INIT** aktiveres stiger forstærkningen, og dermed modtagefølsomheden ca. med en faktor 40.

7.2 Struktur af Heimdal

Det nøjagtige design af Heimdal er dokumenteret og beskrevet i kapitel 19, hvor der også findes brugervejledning programmeringseksempler mv. I dette afsnit har jeg forsøgt at give en kortfattet gennemgang af de designbeslutninger der ligger til grund for det færdige resultat. Jeg har tilstræbt ikke at få for stort sammenfald mellem dette afsnit og kapitel 19, hvilket gør at det kan være nødvendigt at støtte sig til kapitel 19 ved læsning af dette afsnit.



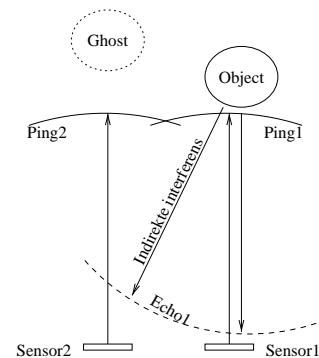
Figur 7.2: Skitse af Heimdals struktur, samt eksempel på tilkobling af sensorer

Jeg har designet Heimdal til at kunne interface til 16 POLAROID sonar-ranging moduler. Tilkoblingen af modulerne sker via fire separate sonar-busser, implementeret som fire 10-leder fladkabler. Hver sonar-bus rummer de nødvendige signaler til at styre/aflæse et enkelt ranging modul, samt to adresse signaler, så der ved hjælp af simple dekodere kan kobles op til fire ranging moduler på hver sonar-bus.

7.2.1 Sensor konfiguration

Strukturen med 16 sensorer i en 4×4 konfigurationen skyldes et kompromis mellem en række forskellige faktorer:

- Jo flere sensorer der er mulighed for at tilkoble jo bedre opløsning kan opnås vha. fastmonterede sensorer.
- Transducerne udsender mest lyd, og er mest følsomme inden for en rumvinkel på 30° , så omgivelserne kan registreres hele vejen rundt vha. 12 sensorer.
- Jo flere sensorer der kan foretage målinger samtidigt, jo hurtigere kan der skaffes oplysninger om omgivelserne.
- Hvis flere sensorer bruges samtidigt kan der opstå direkte interferens, hvor et *ping* fra en sensor opfattes som et ekko af en anden sensor. Desuden kan der opstå indirekte interferens, hvor ekkoet fra en sensors *ping* pga. lydens spredning når frem til, og registreres, af en anden sensor.
- Jo flere sensorer der skal kunne tilkobles, jo mere fyldig og kompliceret bliver elektronikken.



Figur 7.3: Indirekte interferens

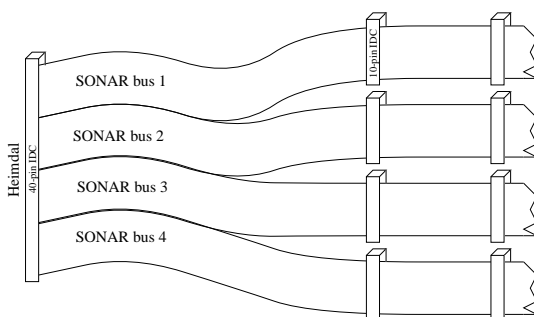
Antallet af separate målegrupper (sonarbusser), er primært begrænset af stik og komponenters pladsforbrug. Med den anvendte teknologi, har jeg fået plads til fire målegrupper. Ved at anvende en mere avanceret teknologi kunne jeg sandsynligvis have fået plads til 6. . 8 målegrupper på et 3U VME board.

Jeg har valgt at stille mig tilfreds med fire målegrupper, dels pga. den ekstra arbejdsindsats det havde krævet at bruge mere avancerede teknologier, og dels fordi jeg vurderer at fordelene ved flere samtidige målinger vil blive opvejet af problemerne med indirekte interferens når der anvendes flere aktive sensorer, og vinklen mellem dem derfor bliver mindre.

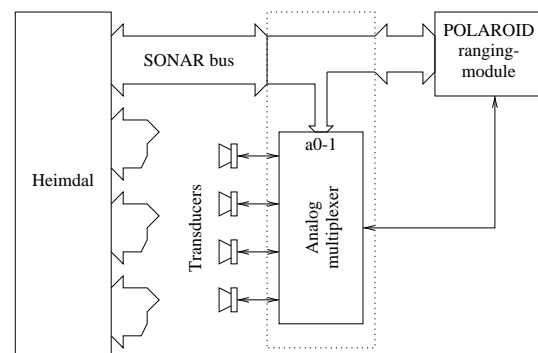
Antallet af mulige sensorer i hver målegruppe er begrænset af hvor mange adressesignaler det har været muligt at tilknytte hver sonar-bus. Jeg har lagt vægt på at lade adressesignalerne i hver bus være uafhængige, for at få maksimal fleksibilitet i valg af aktive sensorer. Som elektronikken er opbygget var der potentielt mulighed for ialt 12 uafhængige adresselinier, men med den anvendte print- og komponent-teknologi har jeg kun fået plads til at føre to adresselinier ud i hver sonar-bus.

7.2.2 Tilkobling og adressering af ranging moduler

Jeg har valgt sonarbus strukturen fordi det er upraktisk at have individuelle kabler mellem Heimdal, og alle de sensorer der kan adresseres. Bus strukturen tillader flere sensorer at deles om de samme forbindelser. Busserne er implementeret som fire 10-polede fladkabler, som vist i figur 7.4. Pga. pladsbesparelsen forbindes alle fire kabler til heimdal via et fælles stik.



Figur 7.4: De fire sonar-busser



Figur 7.5: Analog multiplexning

Der er flere forskellige måder at tilkoble og multiplexe sensorer på.

- Ignorerer adresselinierne, kan der tilkobles et enkelt ranging modul direkte til hver sonar-bus. Denne løsning er velegnet hvor pladsen betyder mere end antallet af sensorer.
- Op til fire ranging moduler kan tilkobles hver bus, via hver sin adressedekoder (skitseret på figur 7.2). Transducer ranging modul og dekoder kan passende indbygges i en fælles kasse. Denne løsning er den mest fleksible, idet sensorerne nemt kan flyttes rundt uafhængigt af hinanden.
- Op til fire ranging moduler kan forbindes til en fælles multiplexer tilkoblet en enkelt bus. Denne løsning er mindre fleksibel end at have separate dekodere, men den kan spare lidt plads, hvis de fire sensorer skal anbringes tæt på hinanden.
- Et enkelt ranging modul kan kobles direkte til en sonar-bus, mens der multiplexes mellem op til fire transducere vha. de to adresselinier (figur 7.5). Denne teknik kræver en DC-koblet analog multiplexer der både kan håndtere de op til $400V_{p-p}$ der udsendes ved et *ping*, og gengive de signaler i mV området transduceren modtager ved et ekko (f.eks. fire små relæer). Denne teknik sparer op til tre ranging moduler pr. bus, hvilket gør den velegnet hvor der er brug for mange sensorer under små forhold. Modtagekredsløbet i ranging modulerne er temmelig støjfølsomme, så der skal udvises stor omtanke mht. kobling, længde og føring af de kabler der bærer transducersignalet.

7.2.3 Tidstagning

Da ranging modulernes modtager har progressiv forstærkning, er sansynligheden for interferens mindst hvis målingerne startes samtidigt. Heimdal er følgelig designet med en enkelt digital timer, der startes når et sæt målinger påbegyndes. For at få nøjagtige målinger har jeg udstyret Heimdal med fire registre, et for hver sonar-gruppe, til at registrere tiden for et modtaget ekko. Med denne løsning kan hver gruppe kun huske et ekko ad gangen, men ved at tillade Heimdal at generere et interrupt når et ekko registreres, kan CPU'en nemt aflæse det pågældende register inden der er mulighed for at modtage næste ekko, idet de 16 svingninger i et ping¹ varer $320\mu s$.

Heimdals fire ekko registre, registrerer tiden med en opløsning på 16 bits. Registrerne aflæses direkte fra VME-bussen vha. et *word read*. Aflæsning af et register, er signal til Heimdal om at registeret kan bruges til at registrere tiden for næste ekko.

Polaroid opgiver sensorens nøjagtighed til 1% af den målte afstand. Normalt forhindrer ranging modulets indbyggede 2.38ms *blanking* modtagelse af ekkoer fra kilder tættere på end 41 cm. Tidligere erfaringer med ranging modulet viser at blanking intervallet, afhængigt af transducerens montering, kan sættes ned til lidt over 1ms, uden at eftersvingninger i transduceren registreres som et ekko. Et blanking interval på 1ms svarer til en minimumsafstand på ca. 17cm. Den bedst opnåelige målenøjagtighed for sensoren er følgelig 1.7m.m.

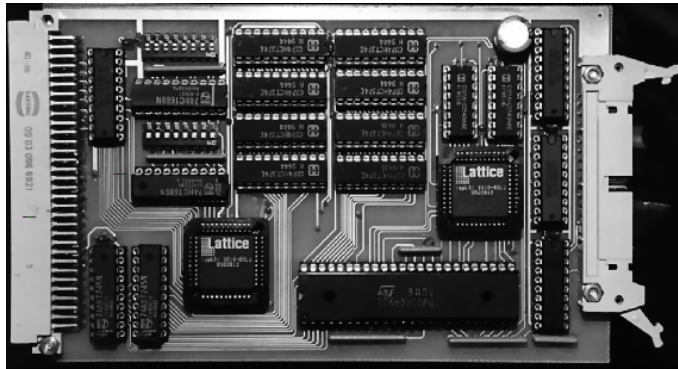
For at kunne udnytte sensorens nøjagtighed fuldt ud, er Heimdal designet så ekko registrens LSB svarer til $8\mu s$, svarende til ca. 1.4m.m. MSB's betydning bliver da $262144\mu s$ svarende til ca. 41m.

For at undgå problemer hvis der ikke registreres et ekko, bruges MSB til at indikere *timeout*. Når MSB bliver sat, opfører Heimdal sig nøjagtigt som hvis et ekko var modtaget, bortset fra at uret standses.

7.2.4 Styling af måling

Jeg ønsker at gøre Heimdal's belastning af CPU'en så lille som praktisk muligt, hvilket betyder at Heimdal selv skal styre timingen af de signaler der skal udveksles med ranging modulerne, samt at Heimdal skal kunne generere interrupts når det er nødvendigt for CPU'en at aflæse ekko registre.

¹Pga. eftersvingninger i transduceren varer et ping snarere ca. $450\mu s$



Figur 7.6: Heimdal

Der hvor regnbuen Bifrost ender i Asgård, bor Heimdal på gården Himmelbjerg, for det er ham, som vogter adgangen til gudernes bolig. Han behøver mindre søvn end en fugl; han ser lige godt ved nat som ved dag, og det hundrede mile væk! Hans hørelse er så fin, at han kan høre græsset gro, og ulden på fårenes rygge og alt, hvad der lyder højere end det! Mærker han nogen farer, vil han blæse i luren Gjallarhorn, som kan høres i alle verdener.

(Nordisk Myte)

Til styring af alle signaler vedrørende ranging moduleerne, timer og registre, har jeg implementeret et passende logikkredsløb (sonar-kontrollogik) i en programmerbar logikkreds (ispLSI 1016). Mine overordnede overvejelser mht. sonar-kontrollogikken er:

- Pga. muligheden for interferens mellem sensorerne skal der være mulighed for at vælge hvilken kombination af de fire sensorgrupper der deltager i en måling.
- En måling startes ved at aktivere et INIT signal, der sendes til de grupper der deltager i målingen, og også bruges til at starte og stoppe/nulstille Heimdals ur.
- Når tiden for et registreret ekko, indlæses i det tilsvarende register, skal der genereres et interrupt.
- Heimdal skal kunne bruges til registrering af mange ekkoer fra hver sonargruppe. Derfor skal hver enkelt gruppe *nulstilles* (BLANK) efter hvert modtaget ekko.
- En ny tid må ikke indlæses i et (ekko) register før CPU'en har aflæst det. Efter en enkelt aflæsning kan ekko registeret bruges til at aflæse tiden for næstkommende ekko.
- For at undgå at registrere det samme ekko mere end en gang, må der ikke kunne registreres ekkoer tættere på hinanden end $450\mu s$.

Som interface mellem sonar-kontrollogikken og VME-bussen har jeg valgt en MC68230 PI/T periferikreds. PI/Ten har de fornødne digitale ind og udgange til adressering af sensorer på sonar-busserne, og til at styre sonar-kontrol logikken. PI/Ten's fire *handshake* indgange er koblet så der kan genereres separate *vectored interrupts* ved modtagelse af ekkoer fra hver enkelt sonar-bus. Jeg bruger ikke PI/Tens indbyggede 24-bits timer direkte i målekredsløbet, idet den kun kan aflæses softwaremæssigt. Den kan frit bruges som enkeltstående timer i software.

7.3 Implementation

Som nævnt er Heimdal implementeret som et 3U VME slave modul. Den fulde dokumentation af Heimdal findes i kapitel 19. Det vil føre for vidt at give en detaljeret beskrivelse af aspekterne omkring Heimdals opbygning, hvorfor jeg i dette afsnit giver en kortfattet beskrivelse af de mest interessante aspekter.

7.3.1 Kredsløbsdesign

Kredsløbsdesignet er en del simplere end det var tilfældet med Gefion, fordi en langt større del af funktionaliteten er implementeret i programmerbar logik, og fordi der ikke anvendes periferienheder der stiller specielle krav om timing i forhold til VME-bussens.

7.3.2 Programmerbar logik

Hvor jeg i Gefion anvendte den ret simple PAL teknologi, er jeg i Heimdal gået over til de væsentligt mere avancerede ispLSI² kredse fra Lattice semiconductors, nemlig ispLSI-1016. Selvom ispLSI-1016 er en af de simpleste kredse fra ispLSI serien, svarer dens funktionalitet groft sagt til hvad man kan opnå ved en sammekobling af 5-10 af de PAL kredse der anvendes i Gefion. Kredsen har 39 ben der kan anvendes til I/O, clock signaler mv, og den rummer 64 flip-flops, med rig mulighed for intern feedback, hvilket gør den meget velegnet til implementation af mindre tilstandsmaskiner. ispLSI seriens største kredse rummer flip-flops nok til at implementere små CPU'er.

Jeg anvender en ispLSI 1016 til at implementere kontrollogikken til VME interfacet, og en til at implementere sonar-kontrollogikken. I begge tilfælde udnytter jeg ca. 75% af kredsens funktionalitet. og 100 hhv. 85% af kredsens I/O forbindelser.

Jeg har anvendt softwaren fra Lattice semiconductors evalueringskit til at programmere kredsene med. Kittet giver kun adgang til ispLSI 1016 og dens *lillebror* ispLSI 2032. Programmeringen foregår direkte i Booleske ligninger³, hvilket er et tilbageskridt i forhold til PAL-assemblerens pascal-lignende syntax. Til implementation af tilstandsmaskiner har jeg brugt PAL-assembleren til at oversætte fra højniveau udtryk til tilstandsligninger, som jeg derefter har kopieret ind i Lattice udviklingsprogrammet.

Programmerbar logik af ispLSI's kaliber har været en stor fordel i designet af Heimdal, og har gjort det muligt at gøre Heimdal meget selvstændig mht. at håndtere måleprocessen. Havde jeg haft adgang — og tid — til at anvende større udgaver af ispLSI familien, kunne jeg have lagt en større del af funktionaliteten ind i en sådan kreds. En enkelt ispLSI kreds med 70-80 I/O ben, og et par hundrede interne flip-flops, ville kunne erstatte samtlige komponenter på Heimdal, bortset fra bufferkredse, samtidigt med at Heimdals funktionalitet, f.eks. antallet af sensorgrupper, ville kunne udvides.

7.3.3 Printteknologi

Som ved fremstilling af Gefion, fremstiller jeg selv de nødvendige print, og har derfor kun mulighed for at anvende almindeligt dobbeltsidet print, uden gennempletteringer.

De anvendte designregler for Heimdal er vist i tabel 7.1. Mine erfaringer med fremstilling af print til Gefion er så gode at jeg tør anvende lidt tyndere baner, og lidt mindre afstande ved fremstillingen af Heimdal. På trods af denne forfining, har jeg stadig kun mulighed for at føre en enkelt bane mellem to ben på en standard IC. Jeg har mulighed for at anvende SMD komponenter med en benafstand på 50 mils, men har ikke mulighed for at føre baner mellem benene på en sådan.

Værdi	mils	mm
min. hulstørrelse:	25	0.6
min. banebredde:	12	0.3
min. afstand:	10	0.25
min. loddeø (0.6mm hul):	40 Ø	1.0 Ø
min. loddeø (0.8mm hul):	50 Ø	1.3 Ø
min. loddeø (1.0mm hul):	60 Ø	1.5 Ø

Tabel 7.1: Designregler

Ved fremstillingen af Heimdal, er jeg nået til grænsen for hvad jeg formår med den printteknologi jeg råder over. Den høje koncentration af loddeøer der i særdeleshed er forbundet med brug af PLCC sokler, og med den høje komponenttæthed i almindelighed, gør det vanskeligt at route printet tilfredsstillende. Den direkte konsekvens har været at jeg er gået på kompromis med tykkelsen og føringen af spændingsforsyningen til flere IC'er på printet. Et kompromis der under afprøvningen viste sig at være en dårlig ide. Det er sandsynligt at problemerne med spændingsforsyningen kan fjernes ved at udføre endnu en design-iteration, men hvis jeg fremover skal fremstille kredsløb med lignende, eller større komponentkoncentration vil jeg foretrække at anvende en teknologi med mulighed for mere end to lag lederbaner.

²In system programmeble Large Scale Integration

³I nyere versioner af evalueringskittet er designsoftwaren blevet væsentligt mere fleksibelt, avanceret og brugervenligt

Da jeg ikke har mulighed for at anvende gennempletterede loddeøer, skaber jeg forbindelse mellem baner på de to sider, ved at lodde komponentben på både over og underside. På komponentsiden er det mange steder umuligt at føre spidsen af en loddekolbe ned til loddestedet, hvorfor jeg i stedet varmer på loddensiden, og gennemfører lodningen på komponentsiden vha. komponentbenets varmeledningsevne. Denne metode fungerer fint med tulipansokler, hvor benene består af massivt kobber med en diameter på 0.8mm. På PLCC sokler, hvor benene består af foldet kobberfolie er varmeledningsevnen ikke helt så stor, hvilket giver problemer med at lodde på komponentsiden. Lodningerne blev gennemført med stort besvær, og beskadigelse af enkelte loddeøer til følge. Mht. PLCC sokler vil jeg kun anbefale loddemetoden i mangel af bedre, og kun til den yderste række ben.

7.3.4 Printdesign

I dette afsnit gives en kort gennemgang af de vigtigste aspekter af printdesignet.

+5V og 0V er ført med brede baner, på hver sin side af printet, langs den ene kant. Derfra fører vinkelrette *sidebaner* ind til de komponentgrupper der skal forsynes med 5V. Ved at holde arealet omsluttet af forsyningsbanerne minimalt, minimeres selvinduktionen i forsyningsvejen.

Alle bufferkredse til at drive VME signaler sidder så tæt på VME connectoren som praktisk muligt, mens de tre restenrende IC'er der indgår i VME interfacet sidder så tæt på som bufferkredsene og nødvendige printbaner tillader. Bufferkredsene til sonar-busserne sidder så tæt på sonar-connectoren som praktisk muligt. Forbindelserne til sonar-busserne laves så alle signaler i fladkablet er nabo til en stelforbindelse ⁴ (0V), hvilket minimerer selvinduktionen, og muligheden for induktivt koblet støj og interferens.

Det var muligt at afkoble MC68230 og de to ispLSI 1016 kredse med hver sin 100nF SMD kondensator, monteret under komponenten, så looparealet af *sløjfen* gennem komponent, printbaner, og kondensator minimeres. De tre bufferkredse til sonar-bussen afkobles både med alm. afkoblingskondensatorer, og en 100µF elektrolytkondensator. Alle bufferkredse til VME bussen afkobles med standard afkoblingskondensatorer, placeret lige op ad komponenten, med så lille loopareal af printbanerne som muligt.

Under designet af printet, var der kun lige akkurat plads til de nødvendige printbaner, hvilket har bevirket at jeg måtte gå på kompromis med føringen af banerne til spændingsforsyning, så banerne blev længere og tyndere end først planlagt. Desuden ligger +5V og 0V ikke overalt umiddelbart ved siden af — eller ovenpå — hinanden.

7.4 Afprøvning

Afprøvningen af Heimdals ikke VME-specifikke del er fortrinsvis sket parallelt med udvikling og afprøvning af softwareinterfacet til Heimdals (se kapitel 8). Afprøvningen af den VME-specifikke del af Heimdals er beskrevet i afsnit 5.7.

Afprøvningen af sonar-delen har bestået i at koble Heimdals sammen med POLAROID's afstandsmålere i forskellige konfigurationer og teste funktionen af Heimdals i samspil med afstandsmålerne vha. logikanalysator oscilloskop og den udviklede testsoftware. Afprøvningen har begrænset sig til at afdække problemer med Heimdals og samspillet mellem Heimdals og afstandsmålerne. Jeg har ikke foretaget en egentlig afprøvning af afstandsmålernes akustiske egenskaber.

7.4.1 Testresultater

Der er blevet afdækket enkelte problemer i den VME-specifikke del (beskrevet i afsnit 5.8) og en glemt lodning i sonar-delen (beskrevet i afsnit 8.6). Desuden har afprøvningen vist nogle mindre problemer med

⁴Undtaget een adresselinie, der ligger yderst i kablet

POLAROID's *ranging moduler*. Bortset fra disse mindre problemer fungerer Heimdal fuldstændigt efter hensigten.

7.4.2 Problemer med POLAROID's *ranging modules*

Ud over de udokumenterede egenskaber/problemer med modulerne jeg har beskrevet i afsnit 7.1.2 har modulerne yderligere to problemer. Nogle af modulerne har en tendens til at rapportere et ekko umiddelbart efter de skifter til lyttetilstanden. Problemet skyldes en ustabil spændingsforsyning pga. det store strømforbrug ved udsendelsen af lydimpulsen og det kan afhjælpes ved at montere en $47\mu\text{F}$ tantalkondensator mellem 0 og + terminalerne på bagsiden af modulet.

Når modulet er i lyttetilstanden ligger der ca. 200V DC over den elektrostatiske transducer. Spændingen opretholdes udelukkende af kapaciteten mellem transducerens bagplade og membran. Den høje impedans og følsomhed af modulets indgang gør det særdeles følsomt overfor det elektriske brumfelt der findes i bygninger med AC-installationer. Når Heimdals værtsdatamat er koblet til lysnettet giver denne følsomhed ikke problemer, men ved batteridrift er det umuligt at bruge sensorerne. Problemet løses fuldstændigt ved at indkapsle *ranging modulet* og den elektrostatiske transducer i et Faradaybur der forhindrer eksterne elektriske felter i at påvirke transducerens membranelektrode og dens tilledning. En almindelig metalkasse med hul til transduceren er en udmærket løsning. Den ene version af transduceren har en metalgitter foran membranen der effektivt skærmer den af hvis det forbindes til metalkassen. Den anden version har plasticgitter, der om nødvendigt kan males med et tyndt lag elektrisk ledende maling for at opnå en skærmende effekt. Problemet mindskes tilsyneladende når tilledningerne til *ranging modulerne* gøres kortere og modulerne har fungeret upåklageligt uden indkapsling på James hvor afstanden mellem Heimdal og modulerne er væsentligt kortere end på Cato.

7.5 Konklusion

Det er lykkedes at udvikle et velfungerende VMEbus slave modul — Heimdal, til styring af og dataopsamling fra 16 POLAROID *sonar-ranging modules*.

Heimdal sætter brugeren i stand til at foretage samtidige målinger med op til fire sonar-moduler, der vælges dynamisk ud af ialt 16 moduler. Med en måleopløsning på $8\mu\text{s}$ og en maksimal måletid på 262ms er det POLAROID's sensorer, med en nøjagtighed på ca. 1.7mm og måleafstand på ca. 11m der begrænser målingerne. Heimdal er i stand til at registrere ekkoer med en afstand på $450\mu\text{s}$, og er således i stand til at registrere op til ca. 130 ekkoer i hver måling. Heimdal er designet med henblik på interruptdrevet software og kan foretage målinger med minimal belastning af CPUen i værtsdatamaten.

Heimdal udfører en funktion der er enestående i forhold til eksisterende produkter. Jeg kunne have opnået en tilsvarende funktion og nøjagtighed ved at bruge en kombination af kommercielt tilgængelige *timer moduler*, digital I/O og software, men systemet ville blive mindre effektivt, mere kompliceret, dyrere og ville formentligt optage flere pladser på VMEbussen i styredatamaten.

I lighed med Gefion gælder at design og afprøvning er overfladisk i forhold til hvad der forventes af professionelle produkter og modulet egner sig derfor kun til brug i undervisning, forskning og andre steder hvor der ikke stilles væsentlige krav til driftssikkerhed.

Den øgede brug af programmerbar logik i form af ispLSI 1016 kredse, har gjort det muligt at øge kompleksiteten af modulet i forhold til Gefion, samtidigt med at kredsløbsdesignet er blevet simplere. Brugen af ispLSI kredse stiller imidlertid større krav til printlayoutet, dels pga. den større benthed og dels fordi kredsenes højere arbejdhastighed (80 MHz) introducerer EMC-problemer i et højere frekvensområde end der er tale om ved ældre teknologier.

Kapitel 8

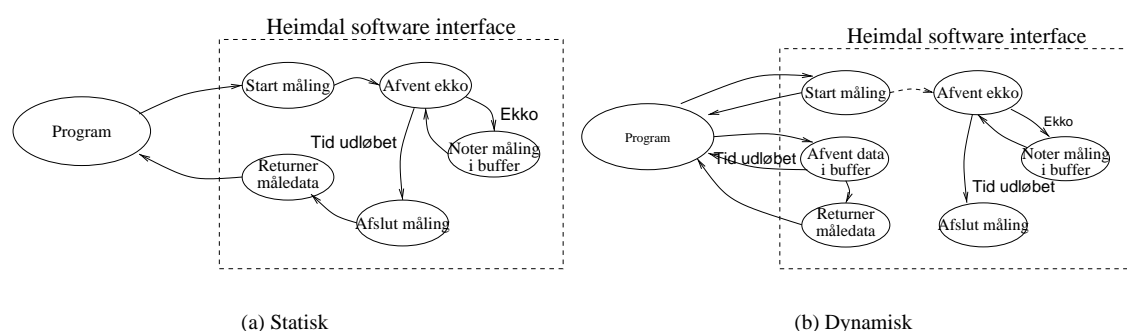
Udvikling af software-interface til Heimdal

Heimdal er designet til at kunne fungere med et minimalt software-interface. Hvis man vil nøjes med at registrere det første ekko fra hver måling, og er tilfreds med POLAROID sonar-sensorernes indbyggede pause mellem målingens start, og mulighed for registrering af første ekko (2.38 ms). Kan Heimdal programmeres uden brug af polling, og uden brug af interrupts.

Hvis ingen ekkoer skal overses, skal et ekko register aflæses senest $448\mu\text{s}$ efter Heimdal registrer ekkoet. Det er muligt, omend upraktisk, at opnå sådanne responstider vha. *polling*, men eftersom Heimdal er designet til at generere interrupts ved modtagelse af ekkoer, er interrupts den eneste fornuftige måde af udnytte Heimdals *multi ekko* muligheder.

8.1 Indledende overvejelser om SW-interface

Med mulighed for detektion af flere ekkoer fra hver sensor, og med generering af et interrupt ved hvert modtaget ekko, er der principielt to måder at lave software interfacet på, som skitseret i figur 8.1



Figur 8.1: To forskellige muligheder for software interface

Statisk måling: Brugerprocessen kalder målerutinen, der starter målingen, og returnerer samtlige måledata på en gang, efter det tidsrum der er afsat til målingen.

Fordelen ved denne metode, er at den er meget simpel. Ulempen er at alle måledata gemmes til målingen er slut, så informationer om objekter tæt på sensoren udsættes for en unødvendig forsinkelse.

Dynamisk måling: Brugerprocessen kalder en rutine der starter målingen. Derefter kalder brugerprocessen en rutine der returnerer måledata efterhånden som de registreres. Software interfacet lagrer måledata i en buffer, så de ikke går tabt hvis brugerprocessen ikke aftager data så hurtigt som de kommer.

Fordelen ved denne metode er at ekkoer kan aflæses løbende som de modtages. Ulempen er en mere kompliceret struktur.

Hvis sensoren og måleobjektet bevæger sig i forhold til hinanden, mens lyden er undervejs, vil der under alle omstændigheder opstå en målefejl. Den ekstra forsinkelse forbundet med statisk måling vil øge målefejlen. Hvis den relative hastighed mellem sensor og måleobjekt er kendt, kan afstanden ved målingens afslutning dog beregnes, ved begge typer måling.

Hvis sensorerne bruges i et miljø med bevægelige objekter (mennesker), hvis hastigheder ikke kendes, er det klart at der vil være større usikkerhed ved at bruge den statiske målemetode.

Sonar-modulernes maksimale rækkevidde er opgivet til 11m, hvor lyden tilbagelægger 22m i løbet af ca. 64ms. Det giver ingen mening at lade en måling vare længere end denne tid. Ved normal gåhastighed på 1m/s, vil tidsforsinkelsen give anledning til målefejl på op til 6.4 cm.

Selv om den statiske målemetode kan give anledning til større målefejl end den dynamiske, vælger jeg at basere mig på den alligevel, da den er lettere at implementere, og lettere at overskue for senere brugere.

$$d_{dyn} = t_m \frac{v_l - v}{2} \quad (8.1)$$

$$d_{stat} = t_m \frac{v_l - v}{2} - v(T - t_m) \quad (8.2)$$

d_{dyn} : Afstand til måleobjekt ved dynamisk måling

d_{stat} : Afstand til måleobjekt ved statisk måling

v_l : Lydens hastighed

v : Relativ hastighed mellem sensor og måleobjekt

t_m : Den målte tid for modtagelse af ekko

T : Den totale måletid ved statisk måling

8.2 Struktur af SW-interface

Inden en måling påbegyndes skal målerutinen have følgende oplysninger.

- Hvilke sonar-moduler deltager aktivt i målingen?
- Hvor lang tid skal der gå fra udsendelse af lydimpuls, til mulighed for registrering af ekkoer?
- Hvor længe skal målingen vare?
- Hvor mange ekkoer fra hver aktiv sensor ønskes registreret?

Målerutinen skal:

1. Adressere de valgte sensorgrupper, og adressere den valgte sensor i hver gruppe, samt aktivere interrupt ved modtagelse af ekko.
2. Aktivere de adresserede sonar-moduler, så de udsender en lydimpuls.
3. Sætte Heimdals timer til at generere et interrupt når sonar-modulerne skal begynde at lytte efter et ekko.
4. Vente på et signal fra interrupt service rutinen, om at målingen er slut.
5. Returnere de måleværdier interrupt service rutinen har opsamlet.

Interrupt service rutinen skal:

1. Hvis interruptet blev genereret af Heimdals timer, og sonar-modulerne endnu ikke har fået besked på at lytte, skal sonar-modulerne have besked på at lytte. Timeren skal derefter sættes til at generere et interrupt når målingen skal afsluttes.
2. Hvis interruptet blev genereret af timeren, og sonar-modulerne har fået besked på at lytte, skal sonar-modulerne deaktiveres, og målerutinen skal signaleres/vækkes.
3. Hvis interruptet skyldes et ekko, skal det relevante EKKO register aflæses, og resultatet lægges i blokken af måldata, med mindre det ønskede antal ekkoer fra den pågældende gruppe allerede er registreret.

8.3 Integration med OS-9

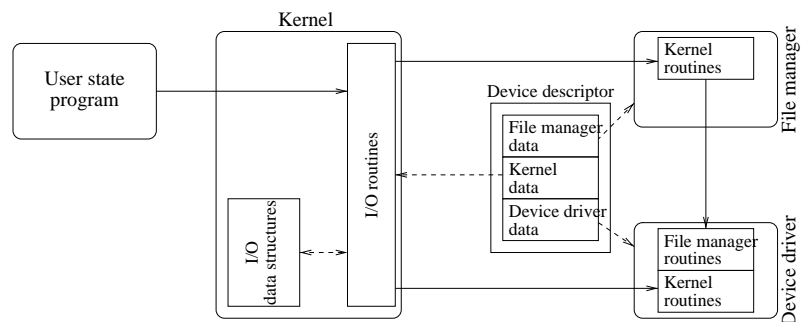
Den bedste måde at håndtere hardware i et OS-9 system, er ved at integrere hardwaren med OS-9's I/O system. Det er muligt at adressere hardware, uden om I/O systemet — en metode der kan være udmærket til simpel registeradgang uden brug af interrupts. Integration med I/O systemet er imidlertid den pæneste og mest generelle metode til håndtering af hardware, og samtidigt den nemmeste måde at håndtere interrupts på.

Oplysninger om OS-9's I/O system kan findes i [21, kap 5-12] og [16, kap 7]

8.3.1 De nødvendige komponenter

For at integrere en hardware enhed i I/O systemet, skal der som minimum udvikles en *device driver*, der danner grænsefladen mellem en *file manager* og hardwaren.

Hvis det er umuligt, eller upraktisk at bruge en eksisterende file manager, skal der udvikles en ny. Der skal desuden fremstilles et eller flere *device descriptor* moduler, der definerer enhedens egenskaber overfor kernen, file manager og device driver, samt kan indeholde opsætningsparametre til hardwaren.



Figur 8.2: Skitse af komponenter i et I/O sub system

8.3.2 Valg af file manager

To af OS-9's file managers kan komme på tale som interface til Heimdals device driver:

Sequential character file manager (SCF) er beregnet til at håndtere Serielporte, og lignende enheder der håndterer en byte ad gangen. Heimdals måldata består af 16-bits tal, der kan lagres i en buffer, og aflæses af SCF en byte ad gangen via Read rutinen. Jeg vil ikke betegne SCF som velegnet, men med lidt god vilje kan der udvikles en passende driver til den.

Sequential block file manager (SBF) er beregnet til at håndtere båndstationer og andre enheder der håndterer en blok ad gangen. I modsætning til *Random block file manager (RBF)*, der implementerer et filsystem, modificerer SBF hverken struktur eller indhold på de data den formidler. SBF er perfekt som file manager i et system der implementerer statisk måling, hvor alle måldata returneres i en samlet

blok. Den er også velegnet til dynamisk måling, hvor måledata returneres ad flere omgange i små blokke.

OS-9 sælges i mange forskellige konfigurationer, der ikke alle inkluderer SBF manageren. SCF manageren er den eneste der er inkluderet i alle konfigurationer, hvorfor den ofte bruges som interface til hardware den ikke nødvendigvis er velegnet til at håndtere.

SBF er inkluderet i alle diskbaserede OS-9 distributioner, og dermed også i IMADA's. Jeg regner ikke med at Heimdal nogensinde skal bruges på et ikke diskbaseret (embedded) OS-9 system, og skulle det endelig blive aktuelt, kan SBF købes i *løs vægt*. Jeg vælger derfor at basere interfacet til Heimdal på SBF.

8.4 Design

Jeg går efter et meget simpelt software interface, så Heimdal kan betjenes via OS-9 kaldene: **I\$Open**, **I\$Close**, **I\$Read**, og **I\$Write**. Kaldene svarer til C rutinerne: **open**, **close**, **read**, og **write**.

8.4.1 I\$Open

I\$Open åbner en sti til enheden. Hvis enheden ikke allerede er initialiseret, kalder kernen automatisk I\$Attach.

8.4.2 I\$Attach

Et kald til I\$Attach skal initialisere Heimdal, såvel som Heimdals device driver. Default værdier for timing, og hvilke sensorer der anvendes, skal læses fra device descriptor modulet, og opbevares/opdateres af device driveren, indtil enheden nedlægges med I\$Close. Der må kun findes en sti til enheden ad gangen.

8.4.3 I\$Close

Et Kald til I\$Close lukker en sti til enheden. Hvis kernen registrerer at I\$Close kaldet medfører at enheden ikke længere er i brug, kalder kernen automatisk I\$Detatch.

8.4.4 I\$Detatch

I\$Detatch skal deinitialisere Heimdal, og nedlægge OS-9 enheden til styring af den.

8.4.5 I\$Read

Et kald til I\$Read starter en måling. Den kaldende proces deaktiveres indtil målingen er færdig. Ved endt måling genaktiveres processen, og måleresultaterne ligger i den buffer processen gav I\$Read.

Hvilke sensorer der deltager i målingen er specificeret af en sensorliste som device driveren opbevarer. Sensorlisten er på fire elementer, et for hver sensorgruppe. Hvert element specificerer om sensorgruppen er aktiv under målingen, og hvilken sensor i gruppen der i givet fald skal anvendes. Elementet kan også specificere at der måles cyklisk på de fire sensorer i gruppen, så næste måling automatisk foretages på næste sensor i rækken.

Læsebufferen er, som vist på figur 8.1 struktureret som et todimensionalt array af *words*, med fire søjler, en for hver sensorgruppe. Den første række indeholder efter kaldet til `I$Read` numrene på den sensor der blev anvendt i hver gruppe. De efterfølgende rækker indeholder de målte tider for hvert ekko, som nulterminerede lister.

Device driveren afgør det maksimale antal ekkoer der ønskes aflæst, ud fra størrelsen af læsebufferen, der er givet ved:

$$8 \times (1 + \text{antal ekkoer})$$

Hvis læsebufferen er mindre end 16 bytes, eller hvis størrelsen ikke er et multiplum af 8, foretages ingen måling.

WORD EchoBuf[1+NEcho][4]

index	0	1	2	3
index	Gruppe 1	Gruppe 2	Gruppe 3	Gruppe 4
0	Sensor nr.	Sensor nr.	Sensor nr.	Sensor nr.
1	Ekko tid	Ekko tid	Ekko tid	Ekko tid
2	Ekko tid	Ekko tid	Ekko tid	Ekko tid
3	Ekko tid	Ekko tid	Ekko tid	Ekko tid
4	Ekko tid	Ekko tid	Ekko tid	0
5	Ekko tid	Ekko tid	Ekko tid	Udefineret
6	Ekko tid	0	Ekko tid	Udefineret
7	Ekko tid	Udefineret	0	Udefineret
...
...
...
...

Tabel 8.1: Struktur af læsebuffer

8.4.6 I\$Write

`I$Write` bruges til at opdatere device driverens sensorliste, og dermed bestemme hvilke sensorer der vil være aktive i næste måling. `I$Write` bruges også til at ændre de to timing parametre.

8.5 Implementation

Al udvikling af systemmoduler til OS-9, foregår på assemblerniveau. Det er muligt at udvikle device driver og file manager rutiner i C, men der skal stadig bruges et *skelet* i assemblerkode, og for den relativt lille applikation der er på tale her, kan det ikke betale sig at begynde på dette.

Jeg har på forhånd en smule erfaring med MC68000 maskinkode, fra den computer jeg udviklede i mit bachelorprojekt, og fra min tid som aktiv Amiga bruger. Jeg er dog lang fra ekspert i denne form for programmering, hvilket den udviklede kode bærer tydeligt præg af. Jeg anvender kun en lille delmængde af processorens instruktioner, og er klar over at programmet kunne optimeres en del både mht. størrelse og eksekveringstid af en erfaren maskinkode programmør.

En oplagt metode til at optimere koden, er at skive den i C, og oversætte den til assemblerkode med en compiler der genererer bedre maskinkode end jeg selv — f.eks. GNU. Som nævnt er applikationen ikke specielt stor, og jeg regner ikke med at min uoptimale maskinkode giver et reelt ydelsesproblem, derfor holder jeg mig til min — lidt naive — maskinkode.

8.5.1 Udviklingsmiljø

På en PEP computer, har jeg oprettet en *konto* kaldet HEIMDAL, til brug for udvikling af Heimdals softwareinterface. Kontoen er oprettet i brugergruppe 0, hvilket giver superbruger rettigheder. Hjemmekataloget ligger i `/dd/USR/HEIMDAL`. Udover en `.login` fil, ligger der tre underkataloger: `SOURCE`, `RELS` og `EXE`, der indeholder hhv. kildetekster, relokerbare (ikke linket) kode, og færdige (linkede) moduler.

Til programudviklingen bruges følgende programmer fra Microwares udviklinkssystem:

umacs Editoren Micro Emacs.

r68 Assembleren, bruges til at assemblere device driver, device descriptor, og et definitionsbibliotek.

l68 Linkeren bruges til at linke device descriptor og device driver.

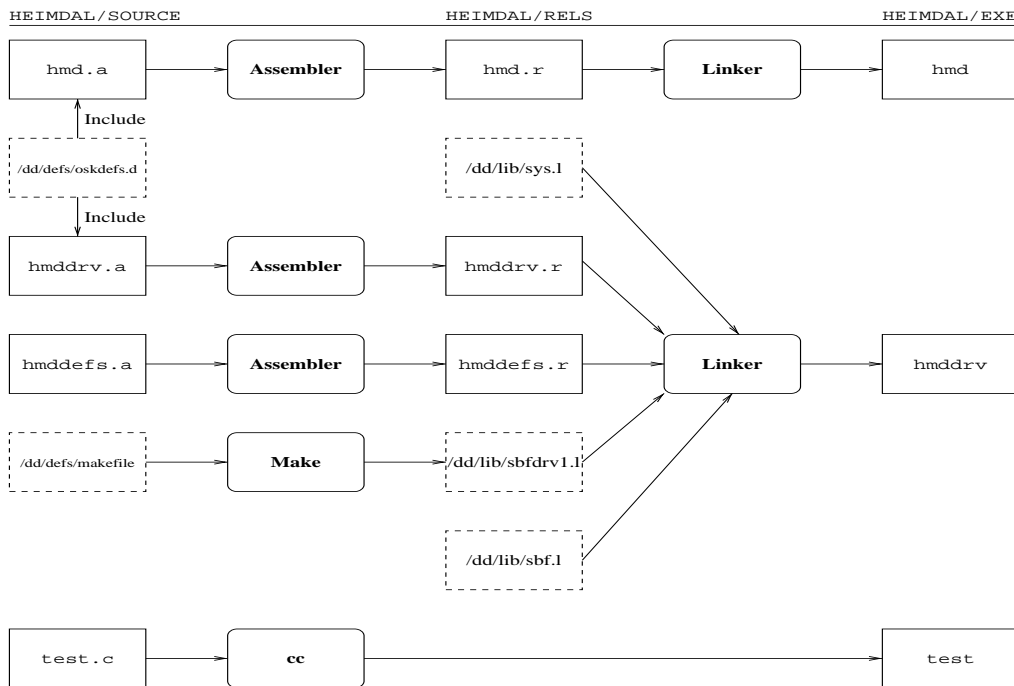
cc C compileren bruges til at kompilere testprogrammet der er skrevet i C. **cc** kalder selv assembler og linker.

make Microwares make program.

Jeg har også haft stor glæde af programmerne **dump**, der udskriver indholdet af en fil eller et modul, og **rdump**, der udskriver oplysninger om objektfiler og biblioteker.

8.5.2 Filer

Ud over selve SBF file manageren, der ligger i `/dd/SYSSRC/BOOTOBS/sbf`, består softwareinterfacet til Heimdal af device descriptor `hmd`, og device driveren `hmdrv`. Figur 8.3 viser disse to filer, testprogrammet `test`, samt de kildetekster, objektfiler, og biblioteker, der bruges til at opbygge dem (systemfiler er vist som punkterede kasser).



Figur 8.3: Oversigt over filer til softwareinterface

Ud over de viste filer, indeholder `HEIMDAL/SOURCE` også makefilen `makefile`, der bruges til at bygge de færdige moduler med. `makefile` afspejler den struktur der er vist i figur 8.3.

8.5.3 File manager

Det største problem i implementationsfasen har været SBF file manageren selv, og dokumentationen til den. SBF er ikke så godt integreret i PEP's udviklingssystem som SCF og RBF, og er heller ikke så vel-dokumenteret, eller beskrevet i den litteratur jeg har haft til rådighed. Dokumentationen i [21], bærer præg af at være på brugerniveau, snarere end udviklingsniveau. På enkelte punkter har der endog vist sig at

være fejl i dokumentationen (i [21, figur 22, side 9-4] er der angivet forkerte offset adresser til felter i *path descriptor*en).

Undervejs i forløbet overvejede jeg at lave min egen file manager, i stedet for at bruge SBF. Baseret på eksemplet i [16, kapitel 13.17] er det ikke svært at lave en manager med den nødvendige funktionalitet. At jeg ikke gennemførte denne ide, skyldes udelukkende at jeg i `/dd/SYSSRC/DRIVER` fandt kildetekst eksempler på en SBF driver: `sbVADI`, beregnet til en A/D converter med input buffer. Selv om der også er fejl i `sbVADI`¹, gjorde den, sammen med den eksisterende dokumentation, i stand til at udvikle en SBF driver.

Der er stadig aspekter af SBF jeg ikke har fundet dokumentation for, og dermed ikke har en fuld forståelse af, bl.a. dens mulighed for at oprette en process til at håndtere I/O. På de steder dokumentationen om SBF har været mangelfuld, er jeg faldet tilbage på `sbVADI` eksemplet.

Jeg har senere drøftet problemet med OS-9 ekspert Ole Benny Hansen fra Danelec. Han kender problemet, og bruger aldrig selv SBF. I stedet bruger han en række file managers fra tredieparts leverandører, der er beregnet på den direkte form for I/O der anvendes her, og som er dokumenteret på udviklingsniveau.

8.5.4 Device descriptor

En device descriptor er et modul der ikke indeholder programmer, men kun data om den enhed det beskriver. Device descriptor har et område med data til kernen, et område med data til file manageren, og evt. et område med data til device driveren. Kernens del er dokumenteret i [21, side 6-4], SBF's del i [21, side 9-3]. Som tidligere nævnt er der fejl i offset adresserne i figur 21 og 22. De rigtige offset adresser kan udledes fra biblioteksfilen `/dd/LIB/sbf.l` vha kommandoen: `rdump /dd/lib/sbf.l -g` Yderligere information om device descriptor moduler, findes i [16, afsn. 1.8, 7.3, og 7.8]

For at lette fremstillingen af descriptor moduler til eksisterende file managers, findes der i `/dd/DEFS` definitionsfiler, der indeholder makroer og passende default parametre til SCF, RBF, og SBF. For SBF's vedkommende er definitionsfilen: `/dd/DEFS/sbfdesc.d`. I [21, side B-5], er vist et eksempel på fremstilling af en SBF descriptor vha. denne metode.

Hvis det er relevant at fremstille en række forskellige descriptormoduler, med små variationer i parametrene, er ovenstående metode en overskuelig måde at gøre det på. Jeg har imidlertid undladt at bruge metoden, og holdt al koden til mit descriptormodul i een kildefil, da det gav mig det bedste overblik over udviklingen af modulet.

Filen: `/dd/USR/HEIMDAL/SOURCE/hmd.a` rummer kildeteksten til device descriptor. Den inkluderer filen: `/dd/DEFS/oskdefs.d` for at importere system relevante konstanter. Filen assembleres og linkes, for at ende som `/dd/USR/HEIMDAL/EXE/hmd`.

Som nævnt er alle kerne og SBF relevante felter dokumenteret i [21]. Herunder knytter jeg en kommentar til de felter der har haft særlig interesse under udviklingen af Heimdals software interface.

¹`sbVADI` bruger en udokumenteret *feature* ved SBF, idet den udnytter at `(a1)` peger på path descriptor ved kald til bl.a. `Read` og `Write`, selv om dette ikke er en del af det dokumenterede interface mellem SBF og driver. ([16, side 307] advarer om denne *særhed* ved SBF)

Kerne felter

M\$Port Basisadressen for Heimdal, i CPUens adresserum: \$87xxxx00, hvor xxxx er Heimdals VME-bus adresse, der sættes vha. 16 dip-switches på Heimdal.

M\$Vector Den interruptgenererende enhed på Heimdal, en MC68230 PI/T, bruger *vectored interrupts*. De øverste 6 bits af MC68230's interrupt vektor er programmerbar, mens de nederste to afhænger af kilden til interruptet. MC68230 bruger således fire efterfølgende interrupt vektorer. Dette felt rummer *basis vektoren*, der skal være delelig med 4. Det er driverens opgave at programmere Heimdal til at bruge vektoren i dette felt.

M\$IRQLvl Normalt er *interrupt level* for et VME kort bestemt af hardware, enten som et fast tal, eller konfigurerbart vha. *jumpers*. Heimdals interrupt level er programmerbar mellem 1 og 6, og det er driverens opgave at programmere Heimdal til det interruptniveau der gives i dette felt.

M\$Mode Heimdal anvendes som læse/skrive enhed, uden mulighed for delt tilgang.

M\$DevCon Offset til det område af descriptoren der definerer Heimdal relevante configurations parametre.

SBF felter

PD.DTP *Device class*, sat til 3, der koder for SBF.

PD.NumBlk *Number of buffers/blocks used for buffering*, er sat til 0 for at tvinge SBF til at anvende *unbuffered I/O*

Driver (Heimdal) felter

HMD_BINH Antal mikrosekunder fra Heimdal ud-sender en lyd, til den begynder at lytte efter ekkoer.

HMD.TIME Målingens længde i mikrosekunder efter BINH.

HMD_SENSOR Sensorlisten på fire elementer, hvor elementerne styrer sensorgruppe 1 til 4.

8.5.5 Device driver

Device driveren indeholder subrutiner der interfacer mellem file manager og den fysiske enhed, samt evt. en eller flere interrupt service rutiner. I SBF's tilfælde drejer det sig om rutinerne: *Init*, *Read*, *Write*, *GetStat*, *SetStat*, *Term*, og evt. *Trap*. Disse rutiner, samt deres interface til SBF er dokumenteret i [21, kap. 9].

Kildeteksten til hele programdelen af device driveren, ligger i filen `/dd/USR/HEIMDAL/hmddrv.a`, der inkluderer `/dd/DEFS/oskdefs.d` for at importere system relevante konstanter. Kildeteksten indeholder en enkelt variabel sektion (*vsect*), der reserverer plads til driverens egne variable. Kildeteksten assebleres til objektfilen `/dd/USR/HEIMDAL/RELS/hmddrv.r`. Objektfilen linkes sammen med en række filer, for at blive til den endelige device driver `/dd/USR/HEIMDAL/EXE/hmddrv`. De filer der linkes sammen med beskrives herunder.

/dd/LIB/sbf.l rummer hverken programkode eller datasektioner, men bruges udelukkende til at definere eksterne konstanter der er relevante for SBF. Kildeteksterne til at generere `sbf.l` findes ikke på PEPen.

/dd/LIB/sbfdrv1.l Rummer ingen kode, og ingen eksterne definitioner, men udelukkende to variabelsektioner der henholdsvis reserverer 128 bytes *static storage* til brug for file manageren, og 64 bytes *static storage*, til en drev tabel. Det er vigtigt at der linkes med denne fil, før objektfilen `hmddrv.r`, for at få variabelsegmeneterne allokeret i rigtig rækkefølge. Kildeteksten til `sbfdrv1.l` ligger sammen med `makefil` til at fremstille den, i `/dd/DEFS(make_sbfdrv1.l)`

hmddefs.r Rummer hverken program eller variabelområder, men derimod eksterne definitioner der er relevante for Heimdal, og device driveren til den. Alle hardware adresse offsets er defineret her, sammen med offset adresserne på de Heimdal relevante felter i device descriptor modulet. Kildeteksten til denne objektfil ligger i `/dd/USR/HEIMDAL/SOURCE/hmddefs.a`

/dd/LIB/sys.l Rummer ingen program eller variabelområder, men rummer alle systemrelevante eksterne definitioner. Kildeteksterne og `makefil` til at fremstille denne biblioteksfil ligger i `/dd/DEFS(make_sys.l)`

Programkoden til selve driveren, ligger som nævnt i `/dd/USR/HEIMDAL/SOURCE/hmddrv.a`. Kilde-teksten er fyldigt kommenteret, og jeg opfatter den som selvforklarende. Jeg vil dog herunder knytte nogle kommentarer til hver del af driveren:

Init begynder med at initialisere *static storage*. Det mest interessante ved dette, er SBF felterne **SBF_NDrv** og **SBF_DPrc**. **SBF_NDrv** initialiseres til 1, for at indikere at denne driver understøtter et drev². Dette harmonerer med at der blev linket med `/dd/LIB/sbfdrv1.l` der reserverer plads til netop en drevtabel. **SBF_DPrc** (*Driver process pointer*) initialiseres til `$ff`, hvilket forhindrer SBF i at oprette en proces til håndtering af enheden. Jeg har ikke fundet nogle henvisninger til dette i den dokumentation jeg har til rådighed, men taget det fra RBF driver eksemplet `/dd/SYSSRC/DRIVER/sbVADI.a`.

Dernæst kopieres de Heimdal relevante felter fra descriptor modulet til driverens eget variabel-område, og der installeres fire interrupt service rutiner, der koresponderer til de fire interrupt vektorer Heimdal anvender. Hvis installationen af en service rutine fejler, afinstalleres de allerede indstillerede. Denne sidste mekanisme, har jeg senere opdaget, er overflødig, idet kernen automatisk kalder *Term* rutinen hvis *Init* rutinen fejler.

Hvis service rutinerne blev installeret korrekt, initialiseres Heimdals Hardware. Først initialiseres MC68230 PI/Ten, og dernæst programmeres interrupt controlleren til det interruptniveau der er specificeret i descriptor modulet.

Term Deaktiverer Heimdal, og afinstallerer interrupt service rutinerne.

Read Starter med at undersøge om forudsætningerne for at starte en måling er i orden, og returnerer en passende fejlkode hvis ikke. Dernæst beregner den, ud fra størrelsen på læsebufferen, hvor mange ekkoer der maksimalt ønskes detekteret.

Som det næste, undersøges de fire elementer i sensorlisten. De grupper der skal anvendes under målingen aktiveres. Den sensor der skal anvendes i hver aktiv gruppe, adresseres. Hvis en gruppe er sat til cyklisk måling tælles det pågældende element i sensorlisten en op, så der anvendes en anden sensor ved næste måling. Numrene på de anvendte sensorer kopieres ned i de første fire elementer i læsebufferen, og de næste fire elementer sættes til 0 (nulterminerede lister).

Heimdals timer sættes til at generere et interrupt efter den ønskede **BINH** pause er udløbet, målingen startes, og timerens *preload register* programmeres med den resterende måletid.

Den kaldende proces sættes i ventekøen på ubestemt tid, med et kald til `F$SLEEP`. Når den aktiveres igen testes det om aktiveringen skyldes et signal fra interrupt service rutinen. Hvis det ikke er tilfældet kaldes `F$SLEEP` igen. Hvis aktiveringen skyldes et signal fra interrupt service rutinen returneres til den kaldende proces.

Write Starter med at sætte bit 1 i SBF's static storage variable **SBF_DFIfg** til 0. Jeg kan ikke finde nogen henvisninger til dette i den dokumentation jeg har til rådighed, men har taget det fra `/dd/SYSSRC/DRIVER/sbVADI.a` eksemplet, hvor det er kommenteret med: "clear WRITE flag, set by SBF, necessary to bypass SBF activities" Driveren virker ikke uden denne detalje!

Dernæst undersøges størrelsen af skrivebufferen. Er den ulig fra 4 eller 8 returneres en fejl. Er den lig fire, betragtes skrivebufferen som en sensorliste, der kopieres til driverens interne sensorliste, og dermed opdaterer denne. Er størrelsen lig 8, betragtes bufferen som to 32 bits tal, der koder for hhv. **BINH** pausen, og tiden for den resterende måling i mikrosekunder. driverens interne tidsvariable ændres til disse to værdier.

SetStat Er ikke implementeret, og returnerer blot en fejlkode.

GetStat Er ikke implementeret, og returnerer blot en fejlkode.

Interrupt service rutiner De fire service rutiner kaldes ved registrering af et ekko på en af de fire sensorgrupper. Den ene rutine kaldes også ved timer interrupt. Alle fire rutiner læser nummeret på den

²SBF er beregnet til båndstationer

sensorgruppe der er potentielt ansvarlig for interruptet, ind i et register, og hopper til en fælles ekko service rutine. Den rutine der kan kaldes pga. både timer og ekko, undersøger først om interruptet skyldes timeren, og hopper til en timer service rutine hvis det er tilfældet.

Den fælles ekko service rutine undersøger om interruptet overhovedet kom fra Heimdal, og returnerer til kernen hvis ikke. Hvis interruptet kom fra Heimdal, slettes Heimdals interrupt tilstand, og hvis der er plads til flere ekko registreringer i den pågældende søjle i læsebufferen aflæses det pågældende ekko register. Er der plads til det i læsebufferen, skrives et nul i næste række, så søjlerne fortsat er nulterminerede.

Timer service rutinen undersøger om **BINH** allerede er aktiveret. Hvis ikke aktiveres **BINH**, interrupt tilsat den slettes, og der returneres til kernen. Fordi Read rutinen har lagt den restrende måletid ind i timerens preload register, vil timeren generere et nyt interrupt når den resterende måletid er udløbet.

Hvis **BINH** allerede er sat, slettes interrupt tilstanden, timeren standes, målingen afbrydes, og der sendes et signal til den sovende måleproces, før der returneres til kernen.

8.6 Afprøvning

Den egentlige afprøvning af device driveren er foregået løbende under udviklingen, men jeg har naturligvis foretaget en endelig test af systemet som helhed.

8.6.1 Løbende afprøvning

Eftersom jeg er nybegynder mht. OS-9, og ikke er ekspert i 68000 maskinkode, har jeg lavet en masse programfejl undevejs i udviklingen. Som konsekvens heraf har jeg udviklet driveren i små skridt, og foretaget en grundig afprøvning/debugning for hvert skridt.

OS-9's *system state debugger* (**sysdbg**) er ikke til rådighed på PEPen, så jeg har været henvist til en mere primitiv metode til debugning/afprøvning.

Det vigtigste redskab til afprøvning undervejs i udviklingen har været et VDOUT, 16 bits digitalt *output* VME-board, fra PEP, der vha. 16 lysdioder viser tilstanden af sine udgange. VDOUT er en bekvem måde at udlæse statusinformation, parametre, variable etc. fra et *system state* modul. Jeg har brugt VDOUT, til at checke parameteroverførsel, betingede hop i programmet, udførelsen af løkker, kald af interrupt service rutiner etc.

8.6.2 Test vha. dump utility

Efterhånden som device driveren tog form, kom det på tale at teste systemet som en enhed. Til det formål fremstillede jeg en device descriptor, der giver en **BINH** pause på 1 ms, og en måletid, efter **BINH** på 64ms. De fire elementer i sensorlisten sættes så alle fire grupper foretager cyklisk måling, startende med sensor nummer 1.

Den nemmeste metode til at foretage målinger, er at bruge shell-kommandoen **dump**, der normalt bruges til at udskrive indholdet af filer eller moduler, på hexadecimal og tekstform. **dump** læser 16 bytes ad gangen, og udskriver dem i en linie, i 8 kolonner, med et word i hver kolonne.

Ved at give kommandoen: **dump /hmd** efter **sbfb**, **hmd**, og **hmddrv** er læst ind i hukommelsen, åbner **dump** enheden (**I\$Open**), hvilket medfører at enheden initialiseres (**I\$Attach**), så device driverens **Init** rutine kaldes. Derefter læser **dump** 16 bytes ad gangen (**I\$Read**), via **SBF** og driverens **Read** rutine. Resultatet er at driveren foretager en måling for hver linie **dump** udskriver. Den første linie måler vha. sensor 1 på alle fire grupper, den næste vha. sensor 2, etc. De første fire *word*-kolonner i udskriften fortæller hvilken

sensor i hver af de fire grupper, der blev anvendt i målingen. De næste fire *word*-kolonner, indeholder et mål for afstanden til første ekko.

Figur 8.4 viser forløbet i en test vha. **dump**. Testen er foretaget med kun to sensorer tilkoblet. Sensorerne sidder på gruppe 1 og 3, som nummer 3 hhv. 1.

Figuren viser at de fire første kolonner tæller gennem sekvensen 1, 2, 3, 4, 1, 2, 3, 4, 1, ... som forventet. I kolonne 5 og 7 ses et gyldigt måleresultat i hver fjerde linie, når sensor nummer 3 hhv. 1 anvendes. Alle andre tal i kolonne 5 til 8 er udtryk for at der ikke er adresseret nogen sensor i den pågældende gruppe, og at *pullup* modstanden i Heimdals ekko indgange, får Heimdal til at tro at et ekko er registreret.

```
> load /dd/SYSSRC/BOOTOBJS/sbf -d
> load /dd/USR/HEIMDAL/EXE/hmd -d
> load /dd/USR/HEIMDAL/EXE/hmddrv -d
> dump /hmd
```

Addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	2	4	6	8	A	C	E
0000	0001	0001	0001	0001	0001	007c	007c	065c	007c	007c	007c	007c	007c	007c	007c	007c
0010	0002	0002	0002	0002	0002	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c
0020	0003	0003	0003	0003	0003	063d	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c	=
0030	0004	0004	0004	0004	0004	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c
0040	0001	0001	0001	0001	0001	007c	007c	065c	007c	007c	007c	007c	007c	007c	007c	007c
0050	0002	0002	0002	0002	0002	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c
0060	0003	0003	0003	0003	0003	0641	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c	A
0070	0004	0004	0004	0004	0004	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c
0080	0001	0001	0001	0001	0001	007c	007c	065a	007c	007c	007c	007c	007c	007c	007c	007c
0090	0002	0002	0002	0002	0002	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c
00a0	0003	0003	0003	0003	0003	063f	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c	?
00b0	0004	0004	0004	0004	0004	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c
00c0	0001	0001	0001	0001	0001	007c	007c	065c	007c	007c	007c	007c	007c	007c	007c	007c
00d0	0002	0002	0002	0002	0002	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c
00e0	0003	0003	0003	0003	0003	0640	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c	@
00f0	0004	0004	0004	0004	0004	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c	007c

Figur 8.4: Test af Heimdals device driver, vha **dump**

De gyldige ekkoer giver værdier på ca. \$630, hvilket svarer til de ca. 2 meter der er fra sensoren til det loft der reflekterer lyden.

8.6.3 Afprøvning vha. testprogram

Afprøvning vha. **dump** er for så vidt udmærket, men giver hverken mulighed for at teste modtagelse af mere end et ekko, eller dynamisk ændring af parametre vha. *I\$Write*.

For at teste disse ting, har jeg lavet et menustyret testprogram i C (*test.c*). Programmet giver mulighed for at teste begge disse aspekter af device driveren. Testprogrammet bruger altid den maksimale bufferstørrelse, der giver mulighed for registrering af op til 98 ekkoer. Jeg har inkluderet en primitiv grafisk udlæsning af de modtagne ekkoer, vha almindelige tegn. På en linie med plads til 80 tegn, svarer hvert felt til et 10cm. interval. Hvis der modtages et eller flere ekkoer inden for et interval, markeres antallet af ekkoer i intervallet med et ciffer. Hvis det ønskes kan måledata også vises i tabelform.

Jeg starter med at lave en måling hvor alle fire grupper bruges, men hvor der ikke er tilsluttet nogle sensorer (eller Bifrost moduler). Ekko indgangene vil i dette tilfælde være konstant aktive, og alle fire grupper skal derfor registrere en række ekkoer med et mellemrum på 64 *counts*, eller 512 μ s, svarende til Heimdals indbyggede pause mellem registrering af ekkoer.

```

ECHO DISPLAY

Group 1: Sensor 1
Group 2: Sensor 1
Group 3: Sensor 1
Group 4: Sensor 1

-----1-----2-----3-----4-----5-----6-----7-----> meters
11121111211112111121111211112111121111211112111121111211112111121111
11121111211112111121111211112111121111211112111121111211112111121111
11121111211112111121111211112111121111211112111121111211112111121111
11121111211112111121111211112111121111211112111121111211112111121111
-----
Press M for menu, R to examine raw data, other key to measure again
RAW DATA BLOCK
.....
N: 0 | 1 1 1 1
N: 1 | 124 124 124 124
N: 2 | 188 188 188 188
N: 3 | 252 252 252 252
N: 4 | 316 316 316 316
N: 5 | 380 380 380 380
.....
N: 93 | 6012 6012 6012 6012
N: 94 | 6076 6076 6076 6076
N: 95 | 6140 6140 6140 6140
N: 96 | 6204 6204 6204 6204
N: 97 | 6268 6268 6268 6268
N: 98 | 6332 6332 6332 6332
.....

```

Figur 8.5: Måling uden sensorer

Som det fremgår af figur 8.5, giver denne test det forventede resultat. Før jeg nåede så langt, afslørede testen imidlertid, dels en programfejl i device driveren, og dels en glemt lodning, på U12's ben 9 (komponentside), idet bit 12 altid blev aflæst som 0, i gruppe 3.

Som det næste testes at device driveren aktiverer de rigtige sensorgrupper, adresserer den korrekte sensor i hver gruppe, og at sensorlisten kan opdateres vha. testprogrammet. Denne test blev foretaget manuelt, og gav ikke anledning til kommentarer.

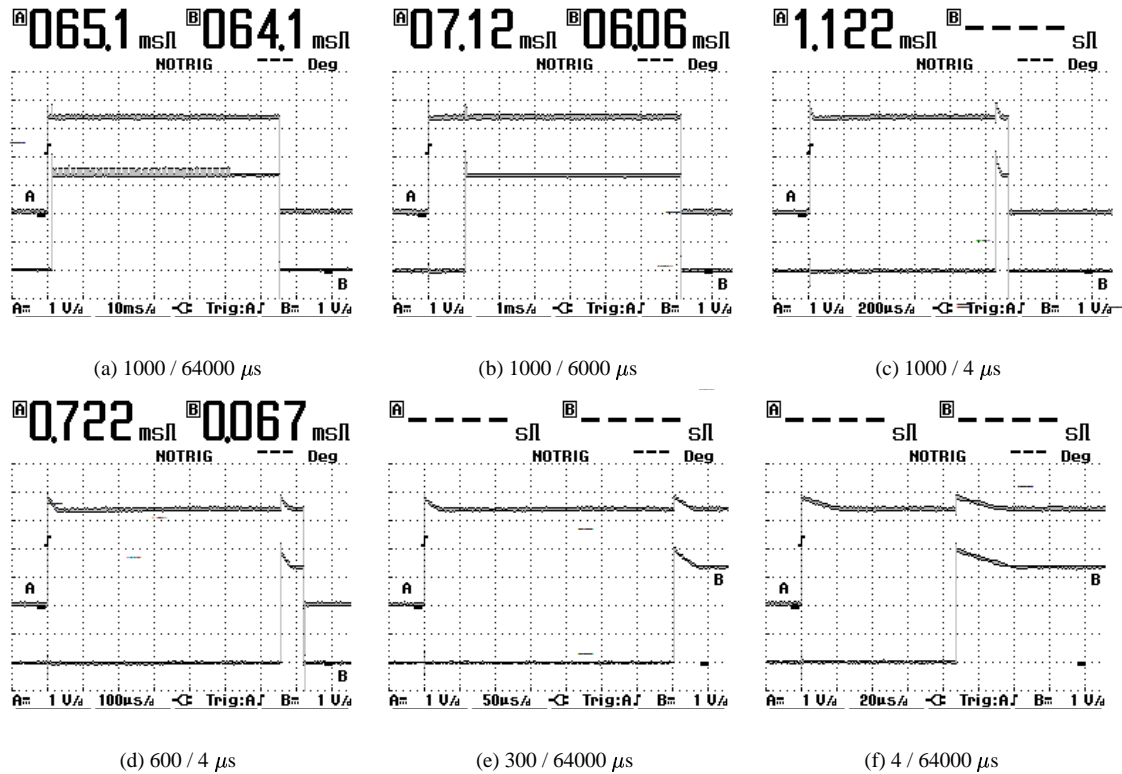
For at teste at device driveren håndterer de programmerbare tidsintervaller korrekt, bruges et oscilloskop til at måle om tiden mellem **INIT_on**, **BINH**, og **INIT_off**, passer med forskellige indprogrammerede tider. En del af resultaterne ses i figur 8.6. Ved hver delfigur er angivet de programmerede tider for **BINH** perioden, og resten af måleperioden³.

Testen viser at begge tidsintervallerne passer med de angivne parametre, på nær en konstant fejl, på ca. $60\mu s$. Fejlen skyldes tidsforsinkelsen fra timeren genererer et interrupt, og til interrupt service rutinen reagerer. Af samme årsag, og pga. andre forsinkelser i device driveren, kan **BINH** intervallet ikke blive kortere end ca. $90\mu s$, og intervallet for resterende måletid, ikke kortere end ca. $70\mu s$.

Begrænsningen af hvor korte tidsintervaller der kan anvendes har ingen praktisk betydning, og den konstante forlængelse af intervallerne, kan i princippet kompenseres ved at lade device driveren trække $60\mu s$ fra de angivne parametre. Jeg har valgt at lade device drivren forblive som den er, da forlængelsen af tidsintervallerne er for små til at have praktisk betydning⁴, og iøvrigt ikke påvirker selve afstandsmålingen.

³ Af hardwaremæssige grunde kan der ikke anvendes parametre på under $4\mu s$

⁴ En tidsfejl på $60\mu s$ svarer til en længdefejl på ca. 1cm

Figur 8.6: Måling af *timing* (A=INIT, B=BINH)

Figur 8.7 viser resultatet af en måling med en enkelt sensor vha. testprogrammet. Sensoren ligger fladt på et bord (fyldt med computerudstyr), og peger op mod et gipsplade loft.

Sensoren registrerer 3 efterfølgende ekkoer, lidt over 2 meter væk. Disse tre ekkoer er i virkeligheden et enkelt ekko, der er strukket tidsmæssigt ud, fordi lyden udbreder sig sfærisk, men reflekteres af en ru plan flade.

Sensoren registrerer en række ekkoer fra 4 meter og udefter. Disse ekkoer er i virkeligheden den lyd der oprindeligt blev reflekteret fra loftet, der har ramt en computermonitor, bordpladen, stole, eller gulvet, og er blevet reflekteret tilbage til loftet, og videre tilbage til sensoren.

```

ECHO DISPLAY

Group 1:           Disabled
Group 2:           Disabled
Group 3: Sensor 1
Group 4:           Disabled

                                                    meters
-----1-----2-----3-----4-----5-----6-----7----->

111                111 1 111 1 11 1 1 1 1 11 111 11 11 1 1

```

Figur 8.7: Test af Heimdals device driver vha. `test.c`

```

ECHO DISPLAY

Group 1:          Disabled
Group 2:          Disabled
Group 3: Sensor 1
Group 4:          Disabled

-----1-----2-----3-----4-----5-----6-----7-----> meters

      1              11  1              1 11 111 1111 1

-----

```

Figur 8.8: Test med objekt 1/2 m fra sensor.

8.7 Dokumentation

Dokumentation på brugerniveau, dvs. installation, og brug af driveren, findes i den tekniske dokumentationsdel, i afsnit 19.12.

Dokumentation på udviklingsniveau, består af grundigt kommenterede kildetekster, til alle dele af softwareinterfacet, gengivet i afsnit 19.15, samt oplysningerne i dette kapitel. Især afsnit 8.5 vil være interessant for udviklere.

Jeg har ikke fundet det nødvendigt at gøre mere ud af dokumentationen, idet alle dele af software interfacet ligger inden for meget faste rammer, defineret af OS-9's I/O system, der er dokumenteret i [21].

8.8 Konklusion

Jeg har lagt et stort arbejde i udviklingen af Heimdals softwareinterface, hvilket overvejende skyldes at jeg skulle sætte mig ind i OS-9's I/O system, som jeg undervejs er blevet meget begejstret for. Bortset fra den mangelfulde dokumentation af SBF file manageren, er OS-9 et meget behageligt, velfungerende, og veldokumenteret operativsystem at arbejde med på I/O niveau.

Det største problem i udviklingen har som tidligere nævnt, været dokumentationen af SBF file manageren. Hvis jeg senere for brug for at udvikle andre OS-9 drivere, med sekventiel blokoverførsel, vil jeg seriøst overveje at finde en anden filemanager, eller evt. udvikle en selv.

Bortset fra det faktum at MC68230 PI/T kredsen er både avanceret og kompliceret at arbejde med, har det været ganske let at programmere Heimdals hardware. Alle de programmeringsmæssige problemer jeg har haft undervejs, har eneten skyldtes manglende kendskab til SBF file manageren, eller min manglende rutine med 68000 assemblerkode.

Det udviklede softwareinterface har, så vidt jeg ved, ingen deciderede fejl, men det kunne forbedres en smule ved at kompensere for de forlængelser af tidsintervallerne der blev påvist i afsnit 8.6.

Den eneste begrænsning softwareinterfacet sætter for brugen af Heimdals, er forsinkelsen af måledata, som omtalt i afsnit 8.1. Jeg tvivler meget på at denne begrænsning vil få praktisk betydning for brugen af Heimdals, og føler mig overbevist om at Heimdals med dette softwareinterface vil kunne imødekomme behovet for et interface til POLAROID's *sonar-ranging modules* for denne og kommende AGV'er.

Kapitel 9

Liniesensor

Tage Søndergaard Larsen har i sit bachelorprojekt arbejdet med at kalibrere en AGV vha. kørsel efter en ret linie ([4]).

I [4] blev der brugt en sensor, opbygget af tre digitale reflektive optiske følere, monteret på tværs af kørselsretningen. Sensoren havde to store problemer:

- Afvigelser fra linien blev først detekteret når AGV'en i realiteten allerede var *kørt af sporet*.
- Sensoren var særdeles følsom overfor baggrundsbelysning, og fungerede kun ved dæmpet belysning.

Sensorens dårlige opløsning betød at AGV'en ikke kørte på striben under kalibreringen, men nærmere zig-zaggede hen over den, hvilket øgede måleusikkerheden.

På trods af den dårlige sensor lykkedes det at foretage en kalibrering der forbedrede kørslen en del, men jeg er overbevist om at en sensor der kan få AGV'en til at blive på linien under kalibreringen vil give langt bedre resultater.

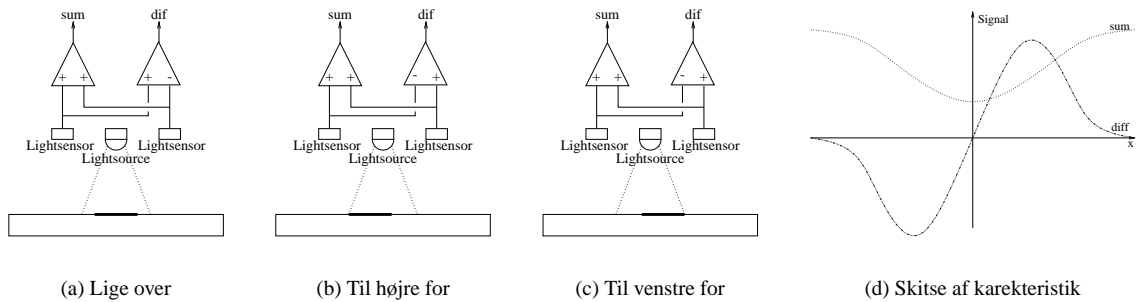
9.1 Sensorprincip

Jeg ønsker at konstruere en sensor der sætter AGV'en i stand til at følge linien uden at forlade den. For at realisere dette, under fornuftige forhold, skal sensoren:

- Sætte datamaten i stand til at bedømme afvigelsen fra linien kontinuert, så der kan foretages en dynamisk kompensering for afdrift.
- For at undgå tilsmudsning og beskadigelse af sensoren skal den fungere i et par cm. afstand fra gulvet.
- Kunne fungere med almindeligt farvet tape, på ensfarvede linoliumsgulve.
- Fungere uafhængigt af den almindelige belysning fra solskin og lamper.

På baggrund af ovenstående har jeg udviklet en sensor, der kan beskrives som en **variabel differentiell optokobler**. Sensorprincippet er inspireret af LVDT¹, der er en induktiv sensor til nøjagtig måling af små afstande. Princippet for sensoren er vist i figur 9.1.

¹liniær variabel differentiell transformator



Figur 9.1: Princippet for variabel differentiell optokobler

Sensoren består af en symmetrisk lyskilde, monteret midt mellem to lyssensorer. Når opstillingen er placeret midt over en streg, som i figur 9.1-a, vil de to sensorer modtage lige meget lys, og differenssignalet er nul. Forskydes linien en smule sidelæns i forhold til opstillingen (figur 9.1-b og c), vil den ene sensor modtage mere lys end den anden, og differenssignalet vil afvige fra 0. Figur 9.1-d viser en skitse af sensorens karakteristisk. Inden for et begrænset område vil differenssignalet være en, om ikke lineær så lineariserbar, funktion af afvigelsen. Uden for dette område aftager differenssignalet, for at falde til 0 når linien er ude af syne. Summen af modtaget lys kan bruges til at afgøre om linien er inden for det *lineære* område.

I figur 9.1 er striben mørkere end underlaget. Sensoren fungerer lige godt i det modsatte tilfælde, differenssignalet skifter blot fortegn, og sumsignalet får maksimum midt over linien.

9.2 Elektronik

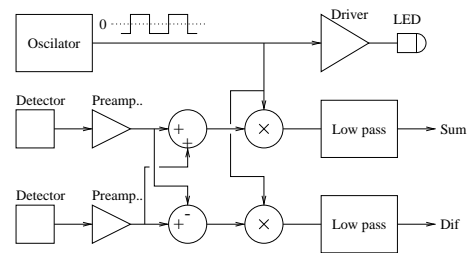
Efter jeg fik ideen til sensoren opbyggede jeg forskellige testopstillinger, for at afprøve princippet, og for at afprøve forskellige optiske komponenter og elektroniske kredsløb. Efterhånden konvergerede testkredsløbene mod den funktionalitet der er vist på figur 9.2. Den færdige linesensor er dokumenteret i kapitel 21

For at gøre sensoren uafhængig af baggrundsbelysningen, er det nødvendigt at anvende moduleret lys. Jeg anvender en modulationsfrekvens på ca. 4kHz. Signalet fra begge lysdetektorer forstærkes af en forforstærker, hvorefter summen og differencen genereres.

Der foretages en fasefølsom detektion, eller synkronisering, af Sum og differenssignalerne, ved at multiplicere dem med udgangssignalet, og derefter lavpasfiltrere dem. Den fasefølsomme detektion undertrykker alle signaler der ikke er i fase med udgangssignalet, og dermed bortfiltreres bidragene fra al modtaget lys der ikke er udsendt af sensoren selv.

Sensoren anvender en kraftig infrarød lysdiode som lyskilde, og to almindelige planare fotodioder som lyssensorer. Fotodioderne er koblet i spærretretningen, hvor deres lækstrøm er en lineær funktion af modtaget lys.

Alle forstærkere og filtre er opbygget vha. almindelige operationsforstærkere, og multiplikationen foretages af en analog multiplekser.



Figur 9.2: Blokdiagram over sensor

9.3 Optik og mekanik

Sensorens karakteristik afhænger af de optiske komponenters egenskaber, deres indbyrdes placering, afstanden til underlaget, underlagets og sribens refleksivitet, samt sribens bredde. Jeg har brugt en blanding af intuition og eksperimenter for at nå frem til en fungerende prototype, men hvis sensoren skal videreudvikles vil det være en fordel at kunne simulere opstillingerne frem for at bygge dem.

9.4 Afprøvning

Efter at have konstateret at sensorens elektronik fungerer som forventet, undersøges dens egenskaber, og der optages en karakteristik af den.

Liniesensorens karakteristik opmåles vha. en 18mm bred, 290mm lang sort streg, udskrevet med laser printer på almindeligt hvidt papir. Sensoren anbringes parallelt med papiret, så sensorens to fotodioder begge er 30mm fra papiret. Sensoren parallelforskydes i forhold til linien, mens differens- og sum-signalerne måles.

Sensoren afprøves under forskellige belysningsforhold.

Ved at integrere sensoren med Cato, afprøves det at sensoren sætter Cato i stand til at følge 18mm brede tapestrimler på gulvet, vha. en simpel reguleringsalgoritme.

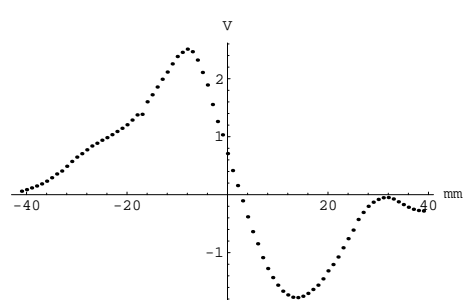
9.4.1 Resultater

Sensoren viste sig at have et 1.5V stort *offset* på den differentielle udgang. Problemet skyldes et ubetænksomt printlayout og ledningsføring, hvor det magnetfelt der dannes af tilledningerne til lysdioden, inducerer en spænding i printbanerne til den ene fotodiode. Den inducerede spænding er i fase med det udsendte lys, og filtreres ikke bort i synkroniseretningen. Ved at sno tilledningerne til lysdioden, bringes *offset* spændingen ned til ca. 0.4V. Den resterende *Offset* spænding, svarer til en afvigelse fra linien på 1-2mm, har næppe praktisk betydning og datamaten kan om nødvendigt tage højde for den.

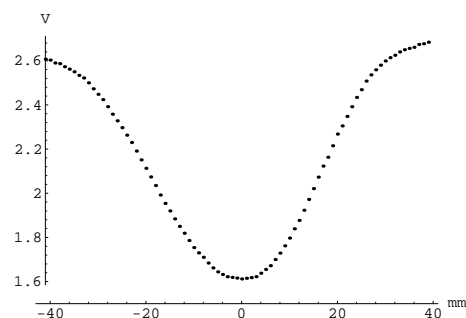
Afstanden mellem sensoren og linien skal være mindst 35mm, og den optimale afstand er ca. 40m.m.

Figur 9.3 viser sensorens karakteristik ved en afstand til linien på 40m.m. Karakteristikken er målt over et 80mm bredt område, med en mm mellem hvert målepunkt. Figur 9.4 Viser samme karakteristik, for et 9mm bredt måleområde, med 0.1mm mellem hvert målepunkt².

²Sensoren var monteret på en mekanisme med mikrometerskrue

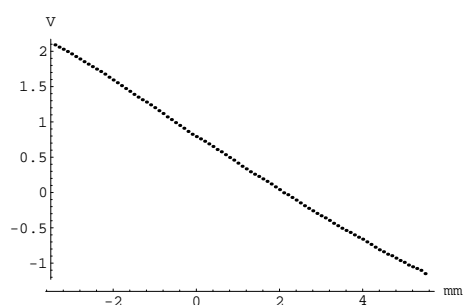


(a) Differenssignal

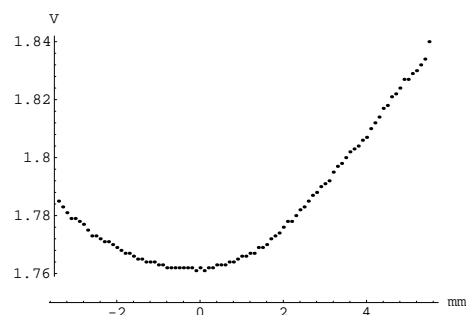


(b) Sumsignal

Figur 9.3: Opmålt karakteristikk, med 1 mm intervaller



(a) Differenssignal



(b) Sumsignal

Figur 9.4: Udsnit af karakteristikk, med 0.1 mm intervaller

Karakteristikken af differenssignalet er tydeligt *offset*forskudt, hvilket til dels skyldes den tidligere nævnte interferens mellem printbaner, men en ubalance mellem fotodiodernes følsomhed, der er for stor til at kunne bortjusteres med balance potentiometeret, bidrager også. Karakteristikens asymmetri skyldes til dels samme ubalance, der dog ikke kan forklare *knækket* mellem -17 og -40 mm, der muligvis opstår pga. en ujævn ud- eller ind-strålings karakteristikk af de optiske komponenter.

Karakteristikken af differensudgangen er praktisk taget lineær over et område på ca. 100 mm, hvilket gør sensoren meget velegnet til at indgå i et liniefølgende kontrolsystem.

Karakteristikkenes toppunkter og hældningskoefficient er afhængige af kontrasten mellem underlag og linie, et forhold der skal tages i betragtning hvis AGV'en skal følge linier på forskellige underlag. Hvis liniens refleksivitet er kendt, kan sumsignalet bruges til at måle baggrundens refleksivitet, når differenssignalet

viser at sensoren befinder sig på midten af linien.

Hvis kontrasten mellem linie og baggrund er kendt, giver sumudgangen mulighed for at afgøre om sensoren befinder sig i det lineære område.

Sensoren er ikke helt uafhængig af baggrundsbelysningen, idet kraftig belysning, giver anledning til en *offset*forskydning af både sum og differenssignal. Følsomheden for baggrundsbelysning er dog for lille til at have praktisk betydning (en 40W pære i 30cm afstand giver anledning til en fejl på ca. 0.2V).

9.5 James

Den variable differentiable optokobler kan ikke detektere opsplitninger i en linie, og kunne derfor ikke bruges til AGV'en James. I stedet blev der fremstillet en bred sensor med 8 lyskilder og 7 detektorer. Detektorerne aflæses analogt, og databehandlingen foregår i software. Der anvendes moduleret lys i James sensor, men i stedet for fasefølsom detektion anvendes der en simplere metode baseret på et båndpasfilter og en AM³ detektor.

9.6 Konklusion

Den fremstillede linesensor fungerer som forventet. Den har en tilnærmelsesvis lineær karakteristik, inden for et område på mindst $\pm 5\text{mm}$, hvilket gør den meget velegnet når en AGV skal kalibreres efter en linie i gulvet.

Sensorens afhængighed af kontrast, dens snævre synsfelt, og manglende evne til at registrere forgreninger på en linie, gør den uinteressant i forbindelse med navigation efter linier. Dens styrke ligger i dens gode nøjagtighed ved AGV kalibrering under kendte forhold.

9.6.1 Videre Udvikling

Selv om sensoren utvivlsomt kan forbedres elektrisk, optisk, og mekanisk, fungerer den godt nok til at varetage sin opgave i Cato, og indtil videre ser jeg ingen grund til at videreudvikle den til brug i AGV-projekter.

I en bredere sammenhæng kan princippet i den differentielle optokobler være interessant som erstatning eller supplement til LVDT ved måle og positioneringsopgaver i mm og μm området. At forfine teknologien til et stade hvor den kan konkurrere med LVDT'er, er en større udviklingsopgave, der bedst kan udføres af, eller i samarbejde med en organisation der i forvejen arbejder med opto-elektronik.

³Amplitude modulation

Kapitel 10

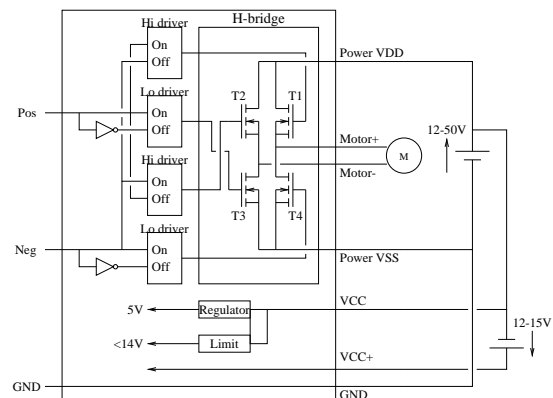
Tykfilmbaseret motordriver

Tykfilm er en teknologi til implementation af kredsløb, og adskiller sig fra almindelige printplader ved at leder- og modstands baner trykkes direkte på et keramisk bæremedie. Teknologien har en række egenskaber der gør den attraktiv i forhold til konventionel printteknologi. I forbindelse med effektelektronik er det især faktorer som bæremediets gode varmeledningsevne, stor komponenttæthed, og god mekanisk stabilitet, der gør det muligt at udvikle små kompakte og driftsikre effekt-kredsløb der er velegnede til brug, hvor der stilles store krav til pladsforbrug, ydelse og driftsikkerhed. Tykfilm er endvidere velegnet til seriefremstilling, og da jeg skal bruge 8 motordrivere i projektet (2 til Cato, og 6 til Scorbotten), er tykfilm på flere måder en attraktiv teknologi.

10.1 Kredsløb

Det kredsløb der implementeres, er bygget over en MOS-FET baseret H-bro, med en forforstærker der er opbygget af diskrete komponenter. Jeg har lagt vægt på at motordriveren skal kunne anvendes i andre sammenhænge end dette projekt, og designet kredsløbet til brug i spændingsområdet fra 12V til 50V, i frekvensområdet fra 0 til 100kHz.

Forforstærkeren er opbygget efter princippet vist i figur 4.8, og et blokdiagram for hele motordriveren, med tilkobling, er vist i figur 10.1. Kredsløbsdiagrammet og en nærmere beskrivelse, befinder sig sammen med resten af dokumentationen af Rungner i kapitel 22



Figur 10.1: Blokdiagram af Rungner

Analyse og dimensionering af kredsløbets dynamiske egenskaber, i et bredt temperaturområde ligger uden for sigtet med denne del af projektet, og vil kunne optage et selvstændigt speciale eller afgangspj. Det eksisterende kredsløb er designet ud fra statiske betragtninger, og de vigtigste dynamiske egenskaber er derefter justeret eksperimentelt vha. en testopstilling.

Bortset fra spændingsregulatorerne, arbejder alle småsignal transistorer i opstillingen som kontakter, hvilket har gjort det muligt at arbejde med ret store tolerancer på de fleste modstande i opstillingen. De store tolerancer gør det muligt at undgå designarbejdet og problemerne, der normalt er forbundet med at fremstille,

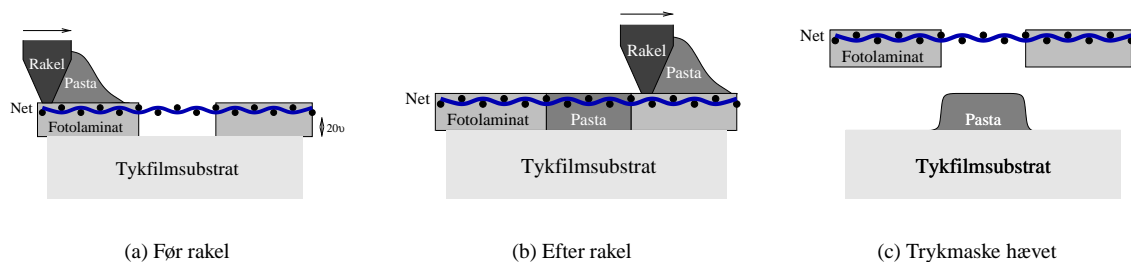
og evt. justere, nøjagtige modstande i tykfilm.

10.2 Tykfilmteknik

I forbindelse med et valgfrit kursus, har jeg tidligere arbejdet med effektkredsløb i tykfilm, og derved dannet mig et overblik over teknologiens muligheder og problemer.

For at realisere et kredsløb påføres bæremediet mønstre af forskellige materialer, der giver de ønskede elektriske og mekaniske egenskaber. Typisk anvendes lederbaner, modstands baner, og områder med isolerende materialer, der tillader ledende baner at krydse hinanden uden at danne forbindelse. De forskellige materialer silketrykkes på bæremediet som en pasta, der består af bindemiddel, opløsningsmiddel, og opslemmede partikler af de materialer der giver området dets egenskaber. Efter tørring af pastaen brændes den ved høj temperatur så bindemidlet forbrændes og de resterende partikler sintres med hinanden og bæremediet.

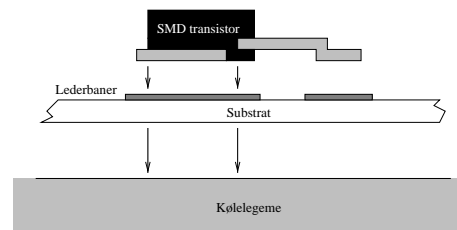
Figur 10.2 skitserer trykkeprocessen, hvor en rakel skubber en bræmme af pasta foran sig på trykkemasken. Når rakel passerer et hul i trykkemasken fyldes det op med pasta, der bliver siddende på substratet når masken hæves.



Figur 10.2: Trykning af lederbane

10.2.1 Varmeledning og montage

En af tykfilms største fordele er at effektkomponenter, på grund af bæremediet lave termiske modstand, kan monteres direkte i kredsløbet, side om side med de småsignal komponenter der styrer dem. Der anvendes SMD komponenter, hvis egne køleplader loddes direkte på de lederbaner der er påført bæremediet. Varme ledes effektivt væk fra komponenterne ved at montere et passende kølelegeme på bæremediet bag-side.



Figur 10.3: Eksempel på montering

Ved at samle alle komponenter på et enkelt bæremedie spares der plads samtidigt med at opstillingen bliver mekanisk simpel. Bæremediet kan skrues eller limes fast til et kølelegeme/køleplade, der samtidigt kan være en del af indkapslingen.

10.2.2 Lederbaner

Lederbaner består af områder belagt med sintrede metalpartikler. Typisk anvendes blandinger af palladium og guld eller sølv. Den normale trykketykkelse er $20\mu\text{m}$, hvilket efterlader et lag på $10 \dots 15\mu\text{m}$ efter brænding. For dette lag, opgiver fabrikanterne flademodstanden¹ af lederbaner til at være $20 \dots 40\text{m}\Omega/\square$ for sølv-palladium blandingerne, der er de mest anvendte, og $2 \dots 4\text{m}\Omega/\square$ for rent sølv, der er det bedst ledende materiale.

10.2.3 Effektmæssig begrænsning

Det største problem med effektelektronik i tykfilm, er at lederbaner normalt kun kan fremstilles i en tykkelse på $15 \dots 20\mu\text{m}$, hvilket giver lederbanerne en u hensigtsmæssig høj modstand med stort effekt-tab og varmeudvikling til følge.

Selv om lederbanerne trykkes i sølv, og holdes så korte og brede som muligt, er det meget vanskeligt at implementere et kredsløb så den totale modstand i tilledningerne til transistorerne i en H-bro, holdes under ca. $40\text{m}\Omega$. Et problem der gør det nødvendigt med tykkere lederbaner.

10.3 Tykke lederbaner

Det er tykkelsen af fotolaminatet, der afgør hvor tykt et pastalag der afsættes på substratet. Ved egne eksperimenter med at lægge fotolaminatet dobbelt og tredobbelt (40 og $60\mu\text{m}$), opstod der imidlertid to alvorlige trykkesproblemer. Pasten hæftede ikke tilfredsstillende til substratet, og banernes kanter flød ud.

I elektronikindustrien fremstilles tykke lederbaner til effektelektronik, ved at laminere baner af fast metal på substratet. IOT har selv deltaget i udviklingen af en industriel proces, hvor fast kobber hæftes til et almindeligt substrat af aluminiumoxid. Lamineringsprocessen skal foregå i en iltfri atmosfære, og IOT's brændeovn egner sig ikke til brug af denne proces.

Som alternativ til den kendte lamineringsproces, har jeg udtænkt to alternativer:

- Laminering af metalbaner ved lodning.
- En flerlags trykkesproce til gradvis opbygning af tykke baner.

10.3.1 Laminering vha. lodning

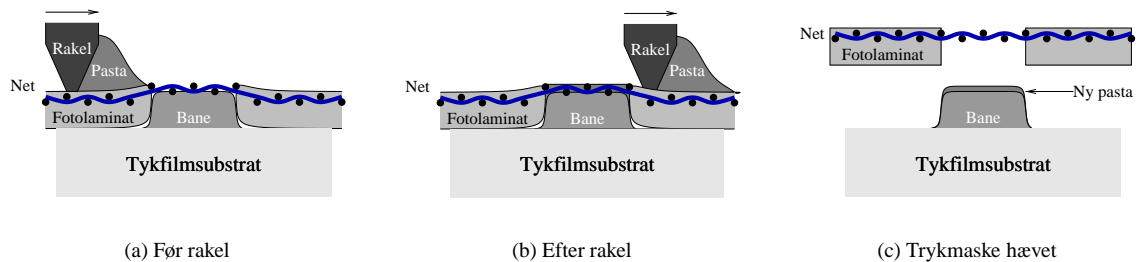
Som alternativ til laminering ved brænding, forestiller jeg mig at anvende en lignende proces, hvor kobberfolie loddet ovenpå almindelige lederbaner med en tin med højt smeltepunkt. Kobberfolien kan loddet på i et stykke, og kobberet mellem lederbaner kan derefter fjernes med de kendte ætseprocesser fra almindelig printfremstilling. Afsluttende kan alle komponenter loddet på med en tin med lavt smeltepunkt, uden at kobberbanerne falder af.

Metoden forekommer mig brugbar men meget besværlig, og jeg har foretrukket at koncentrere mig om udviklingen af et trykkesproce alternativ.

¹Modstanden i en homogen lederbane er proportional med forholdet mellem længde og bredde, og måles i Ω/\square , der angiver modstanden mellem to modsatte sider i et kvadratisk udsnit

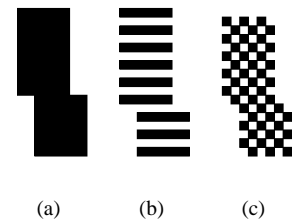
10.4 Flerlags trykketeknik

Det kan ikke lade sig gøre at hæve lagtykkelsen væsentligt ved blot at trykke frisk pasta oven i en eksisterende lederbane. Den eksisterende lederbane vil blot fylde trykmasken op nedefra, så nyt pasta ikke kan komme til (figur 10.4).



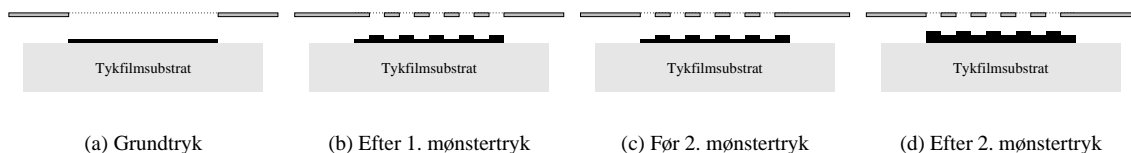
Figur 10.4: Gentaget trykning af lederbane

I modsætning til rene signalbaner, har effektbærende baner mere form af et område end en streg, hvor det er muligt at anvende overfladen af et tryk-område som anlægsflade for et tryk med mindre dimensioner, f.eks. det oprindelige område overlejret med et regelmæssigt mønster af linier eller tern. (figur 10.5).



Figur 10.5:

Figur 10.6 Viser princippet i den gradvise opbygning af en bane ved forskudt trykning. Først trykkes lederbanerne som vanligt i et grundtryk (figur 10.6-a). Efter en tørring trykkes med en maske der rummer de effektbærende baner overlejret med et mønster, der blokerer halvdelen af det åbne areal (figur 10.6-b). Alle de områder i trykmasken der er blokeret, danner en anlægsflade mod den pasta der er trykt i forvejen, så de åbne tern trykkes i en tykkelse på $20\ \mu\text{m}$. Efter endnu en tørring forskydes trykmasken med en afstand svarende til sidelængden på et tern/linie. De udfyldte områder i trykmasken danner nu anlægsflade mod de områder der blev trykt forrige gang (figur 10.6-c), og ved at lade rakelens passere to-tre gange, fyldes de $40\ \mu\text{m}$ dybe huller op, og giver nye tern/linier der ligger $20\ \mu\text{m}$ højere end de forrige (figur 10.6-d).



Figur 10.6: Trykning i skaktern

Jeg har foretaget eksperimenter med både linier, og skaktern, hvor inspektion med mikroskop, og kontrolmålinger af flademodstanden viser at skaktern giver det bedste resultat. Der opstår dybe revner hvis der anvendes linier.

En sidelængde på $25\ \text{mils}^2$ er den mindste, der med IOT's trykkeudstyr kan give et rimeligt veldefineret

²1 mil = (1/1000)''

kvadrat. Jeg har eksperimenteret med skakmønstre hvor ternene har sidelængder på 25 hhv. 50 mils, uden at kunne konstatere kvalitetsmæssige forskelle. For at få maksimal trykkemæssig opløsning anvendes en sidelængde på 25 mils.

Teknikken kan gentages igen og igen, ved at trykmasken forskydes skiftevis den ene og den anden vej. Fordi det ikke er nødvendigt at brænde substraterne, eller skifte masken mellem hvert tryk, er teknikken både hurtig og nem at anvende.

10.5 Fremstilling

Kredsløbet implementeres i tykfilm vha. ni forskellige trykmasker, i en proces der er nærmere beskrevet i kapitel 22. Det mest markante træk ved fremstillingen er opbygningen af tykke lederbaner vha. skaktrykningen, der har givet anledning til en del overvejelser og eksperimenter. Jeg er imidlertid også stødt ind i generelle trykstekniske problemer.

10.5.1 Mulige problemer med anvendelse af rent sølv

Selv om det er dyrere, anvendes normalt en blanding af sølv og palladium til lederbaner. Årsagen er dels at normalt tin/bly loddeing opløser sølv, samt at sølvioner under bestemte betingelser kan trænge ind i de isolerende lag, og med tiden danne kortslutninger mellem to baner der krydser hinanden (sølvmigration). I begge tilfælde løses problemet ved at anvende sølv-palladium, hvor palladium binder sølvet.

For at undgå problemer ved lodning, forsøgte jeg at trykke et lag sølv-palladium oven på de tykke sølvbaner. Resultatet var at sølvbanerne mistede hæftningen til substratet, og bøjede eller krøllede opad under brænding. Sølv-palladium overfladen blev desuden glasagtig og umulig at lodde på. Fænomenet skyldes formentlig et kemisk reaktion under brændingen, kombineret med en bimetaleffekt mellem sølv og sølv-palladium. Det potentielle loddeproblem løses i stedet ved at anvende sølvholdigt loddetin.

Ved tidligere projekter, har jeg observeret en markant blæredannelse på sølv-palladium baner, der vha. et lag dielektrikum krydser over en bane af rent sølv. Ifølge [40] opstår fænomenet fordi det anvendte dielektrikum ved brændetemperatur fungerer som elektrolyt, og effektivt danner et *batteri* med de to krydsende baner af forskellige materialer. Pga. denne såkaldte *batterieffekt* opstår kemiske reaktioner der udskiller ilt under topelektroden, hvorved der opstår blærer.

Batterieffekten kan undgås ved at anvende identiske metaller ved krydsninger, ved at lave midlertidige kortslutninger mellem krydsende baner, eller ved at anvende et dielektrikum der hæmmer mobiliteten af ioner under brænding. Ifølge [40] hæmmer Heraeus IP9117 *multilayer dielectric* pastaen ionmobiliteten så meget at der hverken forekommer batterieffekt eller sølvmigration under brænding.

De to lederbaner der passerer under et område med rent sølv, er det eneste sted der kan opstå problemer med sølvmigration og batterieffekt, og i denne prototype har jeg ikke gjort andet for at forebygge dette end at anvende Heraeus IP9117 pasta.

Efter fremstillingen, blev kredsløbene inspiceret og testet meget nøje, men ingen af de fremstillede kredsløb viser tegn på problemer med batterieffekt eller sølvmigration.

10.5.2 Modstande

Kredsløbet er tolerant overfor store variationer i modstandsværdierne. Derfor kan jeg tillade mig at minimere antallet af trykninger, ved at gå på kompromis med kvaliteten af de trykte modstande.

For at realisere alle trykte modstande med en enkelt pastatype ($1k\Omega/\square$), anvendes både ekstremt smalle og ekstremt korte modstande, med længde/bredde forhold på 0.1 til 10. De små dimensioner giver store usikkerheder på modstandsværdierne.

10.5.3 Trykketeknik

Jeg har haft en del problemer af trykketeknisk karakter, der har kostet en del tid at få løst. Efter mange eksperimenter med måling og justering af pastaernes viskositeter, med trykkemaskinen, og trykkeparametrene; viste det sig at trykkemaskinens rakelkraft, der var justeret til tryk på $1 \times 1''$ substrater, var for lavt til trykning på $2 \times 2''$ substrater, idet kraften fordeles på et større areal. Ved justering af den effektive rakelkraft, til 10. . . 15N, blev alle trykketekniske problemer løst.

Sammenhængen mellem rakelkraft og trykkekvalitet er kompliceret. De forskellige pastatyper har forskellig rheologi, og deres krav til minimalt rakeltryk er forskellige. Trådspændingen i de anvendte trykmasker kan variere. Rakelen udsættes for slid, og ændrer med tiden form.

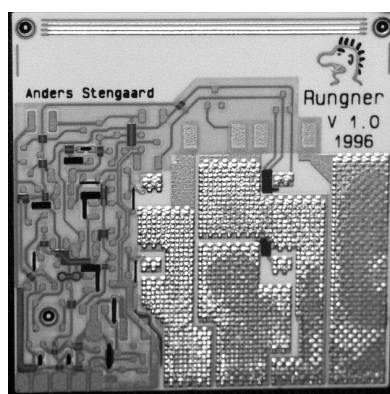
10.5.4 Komponentmontering og håndtering

Jeg anvender loddepasta til pålodning af komponenter. Pastaen silketrykkes på substratet, komponenterne monteres med pincet, og loddes fast vha. *reflow lodning* hvor substratet føres hen over et varmelegeme der smelter det påførte tin.

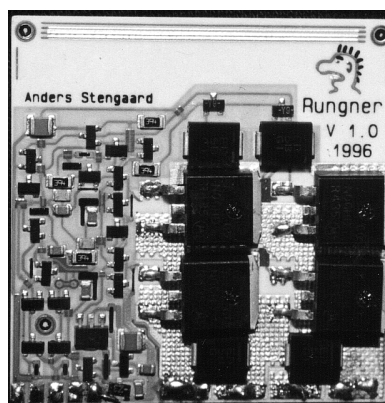
Da de anvendte MOS-FET transistorer er ret kostbare, har jeg i vid udstrækning anvendt transistorer fra et tidligere projekt. Ca. 30% af de flyttede transistorer har imidlertid vist sig at være defekte på trods af at de har fungeret upåklageligt i de oprindelige opstillinger. Jeg har ikke foretaget en egentlig undersøgelse af årsagerne til dette, men det kan skyldes for kraftig opvarmning ved aflodning, udladning af statisk elektricitet, eller begge dele.

Metoden til aflodning af komponenter, består i at placere tykfilmssubstratet på en varmeplade, vente på at loddetinnet smelter, og derefter afmontere transistorerne enkeltvis med en pincet, eller samlet ved at ryste/slå substratet. Loddetin smelter ved ca. 180°C , og det er sandsynligt at den maksimale temperatur for halvlederen (ca. 150°C) overskrides i processen.

I IOT's tykfilmlaboratoriet opbygges meget nemt statisk elektricitet, til et niveau hvor der ved smertefulde udladninger kan trækkes flere mm lange gnister mellem ens fingre og f.eks. dørhåndtag³, og der findes ingen muligheder for potentialeudledning⁴. En MOS-FET transistor kan kun tåle en *gate-souce* spænding i størrelsesordenen 20V, og da det er vanskeligt at undgå at berøre transistorens elektroder ved afmontering, er det sandsynligt at en del er blevet beskadiget på denne måde.



(a) Uden komponenter



(b) Med komponenter

Figur 10.7: Rungner (naturlig størrelse)

Rungner var den største og stærkeste blandt jætter. Hans hjerte var af hård sten, kantet og med ekstra takker på; hans hoved var også af sten, og det samme gjaldt hans skjold, der var både stort og tykt.

Så stod han da klar og ventede på Tor ved Grotunagård; skjoldet holdt han op foran sig, og sit våben, slibestenen, holdt han løftet højt op over skulderen — han var ikke god at komme nær....

(Nordisk myte)

³Luftens dielektriske styrke er ca. 3kV/mm

⁴Gulvbelægningen er senere blevet skiftet, hvilket har afhjulpet problemet

10.6 Testresultater

Rungner testes dels for at undersøge fremstillingsmetoden til tykke lederbaner, og dels for at undersøge det elektriske kredsløb. Selve afprøvningen er beskrevet i kapitel 22, her gives blot en opsummering af de vigtigste resultater.

10.6.1 Tykfilm

Tykkelsesmålingen viser at tykkelsen vokser linært med antallet af lag, og selv om målingen af flademodstanden ikke er særligt nøjagtig, er der rimeligt belæg for at fladeledningsevnen ligeledes vokser linært med antallet af lag.

For hvert lag vokser tykkelsen ca. $15\mu\text{m}$, og fladeledningsevnen vokser $375 \dots 800 S/\square$, svarende til at hvert lag har en flademodstand på $1.3 \dots 2.7 m\Omega/\square$.

10.6.2 Kredsløb

Kredsløbet fungerer efter hensigten, men kombinationen af MOS-FET transistorer, og separate forbindelser af strømforsyningen til effektdel og signaldel, gør det meget skrøbeligt overfor potentialeforskelle mellem effekt- og signal-terminalerne, ved af- og påmontering.

10.7 Konklusion

I løbet af denne del af projektet, har jeg med held udviklet en metode til fremstilling af tykke lederbaner til tykfilmbaseret effektelektronik. Metoden kan næppe betale sig til masseproduktion, men da den er let og hurtig at bruge, mener jeg at den er interessant for laboratorier og firmaer, der ønsker at arbejde med tykfilmbaseret effektelektronik, til prototyper, eller små serier.

Ved hjælp af tykfilmteknik har jeg opbygget et switched-mode udgangstrin, der kan styres direkte af digitale pulsviddemodulerede signaler. Udgangstrinnet er opbygget ret kompakt, med god udnyttelse af tykfilmteknologiens muligheder. Udgangstrinnet fungerer udmærket, og har et stort arbejdsområde. Alligevel er der mulighed for forbedringer af kredsløbet. Især skrøbeligheden ved håndtering og montering bør afhjælpes.

10.7.1 Forslag til videre udvikling

Selv om mit arbejde foreløbigt standser her, ser jeg et stort potentiale for både skaktern-teknikken, og tykfilmbaseret effektelektronik.

Skaktern-teknikken

Skal kredsløb baseret på skaktern-teknikken anvendes i professionel sammenhæng, er det vigtigt af få afklaret en række spørgsmål. F.eks:

- Hvordan er skakternbanernes mekaniske egenskaber? Hvor godt sidder de fast på substratet? Hvordan reagerer de overfor vibrationer, temperatursvingninger, fugt etc?
- Hvordan føres signalbaner bedst på tværs af skakternbaner? Kan der opstå batterieffekt, sølvmigration og kortslutninger mellem sølvbanerne og underliggende baner? Hvordan er de mekaniske egenskaber af materialerne der hvor baner krydser?

- Hvor tykke baner kan der opbygges? Hvordan optimeres trykketeknikken?
- Hvor stor termisk og elektrisk modstand er der mellem komponenter og de kuperede lederbaner?
- Hvordan er økonomien i metoden i forhold til andre teknikker?

Det kunne være spændende at få afklaret disse spørgsmål, og videreudviklet teknikken; f.eks. gennem efterfølgende afgangsprojekter eller specialer.

Tykfilmbaseret effektelektronik

Udviklingen af mit udgangstrin har demonstreret at IOT's tykfilmlaboratorium er i stand til at udvikle tykfilmbaseret effektelektronik, der er både effektivt og kompakt. Kompakte effektive effekt-kredsløb er meget interessante inden for bl.a. robotteknologi, hvor der er mange fordele ved at integrere elektronik og mekanik.

Mit kredsløb er ikke tilstrækkeligt gennemarbejdet til professionelt brug, og det næppe er interessant at lave en direkte videreudvikling af det. Jeg ser derimod interessante perspektiver i at udvikle et høj-effekt udgangstrin, med integreret mikroprocessor. Kun halvdelen af Rungners 25cm² bruges til effektkomponenter, og ved anvendelse af integrerede MOS-FET drivere, bør der kunne blive plads til en *single-chip* processor i SMD hus.

Ved at integrere en programmerbar mikroprocessor med udgangstrinnet, kan brugeren selv konfigurere udgangstrinnet til det ønskede brug. Processoren kan have indgange til forskellige former for feedback, så den kan varetage reguleringsopgaver direkte, og den kan have forskellige former for digitale og analoge grænseflader, så udgangstrinnet kan indgå et et distribueret kontrolsystem med central styring.

Generel trykketeknik

Undervejs i denne del af mit projekt har jeg stiftet bekendtskab med trykketekniske problemer, der alle blev løst ved at øge rakelkraften. Jeg mener derfor, at det er oplagt at foretage en nærmere undersøgelse af sammenhængen mellem rakelkraft, og trykkekvalitet for de forskellige pastatyper.

IOT har stået fast på at trykkemaskinen ikke må justeres midt i et semester, for at give modstandstrykkene en ensartet kvalitet. Derfor er det især interessant at undersøge modstandsfremstillingens følsomhed for variationer i rakelkraften, med henblik på at gøre det muligt for afgang- eller speciale-studerende at arbejde frit med trykkeparametrene i deres projekter.

Kapitel 11

Styring af Scorbot

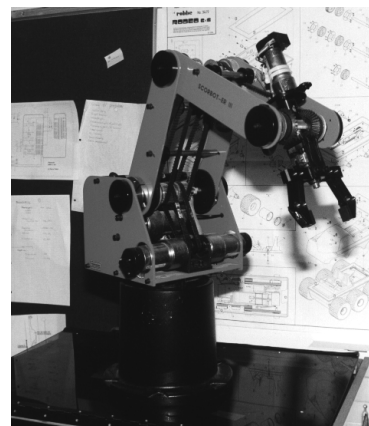
Scorboten er en simpel robotarm beregnet til undervisning. Den har 6 DC motorer, med inkremental-
enkodere, der trækker dens gribeklo og fem rotationsakser. Robotens mekanik er baseret på billige tand-
hjulsgear og remtræk, hvilket giver en stor friktion og en del slør og unøjagtighed.

Selve robotten rummer ingen elektronik, men motortilslutninger, enkodersignaler, og signaler fra *homing* kontakter er samlet i et kabel der vha. et 50 polet stik kan tilsluttes en medfølgende styreenhed.

Selv om Scorboten, mekanisk set, ikke er særligt overbevisende, vil den medvirke til at gøre AGV'en mere funktionel og interessant, hvis den kan integreres med denne.

For at integrere Scorboten med AGV'en, har jeg:

- Interfacet dens motorer til to Gefion motorstyringsmoduler.
- Opmålt robotten
- Opstillet en direkte- og derefter en invers-kinematik for den.
- Skrevet et C funktionsbibliotek til den, der bygger ovenpå funktionsbiblioteket til Gefion.
- Lavet et testprogram der får den til at åbne en 1/2 l sodavandsflaske, og skænke indholdet i to glas.



Figur 11.1: Scorbot

11.1 Elektronik

Scorbotens originale styreenhed rummer en primitiv datamat, der implementerer en positionsstyring for hver motor. Kommunikation med styreenheden foregår via. et RS232 baseret kommandointerface. Styreenhedens største problem er at det ikke giver mulighed for nogen form for styring eller synkronisering af motorernes hastigheder, hvilket begrænser robottens anvendelse væsentligt.

I stedet for at bruge den oprindelige styreenhed, har jeg brugt 6 tidlige prototyper af de motordrivere jeg har fremstillet, og via dem forbundet robottens 6 motorer, til to Gefion moduler. Motorernes indbyggede inkremental-
enkodere kan forbindes umiddelbart til Gefions enkoderindgange, og *homing*-kontakterne kunne ligeledes forbindes direkte til Gefions digitale indgange. Sammenkoblingen af Scorbot, Motordrivere, og Gefion er vist i afsnit 16.5.

11.2 Direkte kinematik

Formålet med den direkte kinematik, er at beregne tool-spidsens position og orientering, som funktion af de bevægelige led.

Beregningen af den direkte kinematik følger nøje [8, kapitel 2], og der anvendes samme notation.

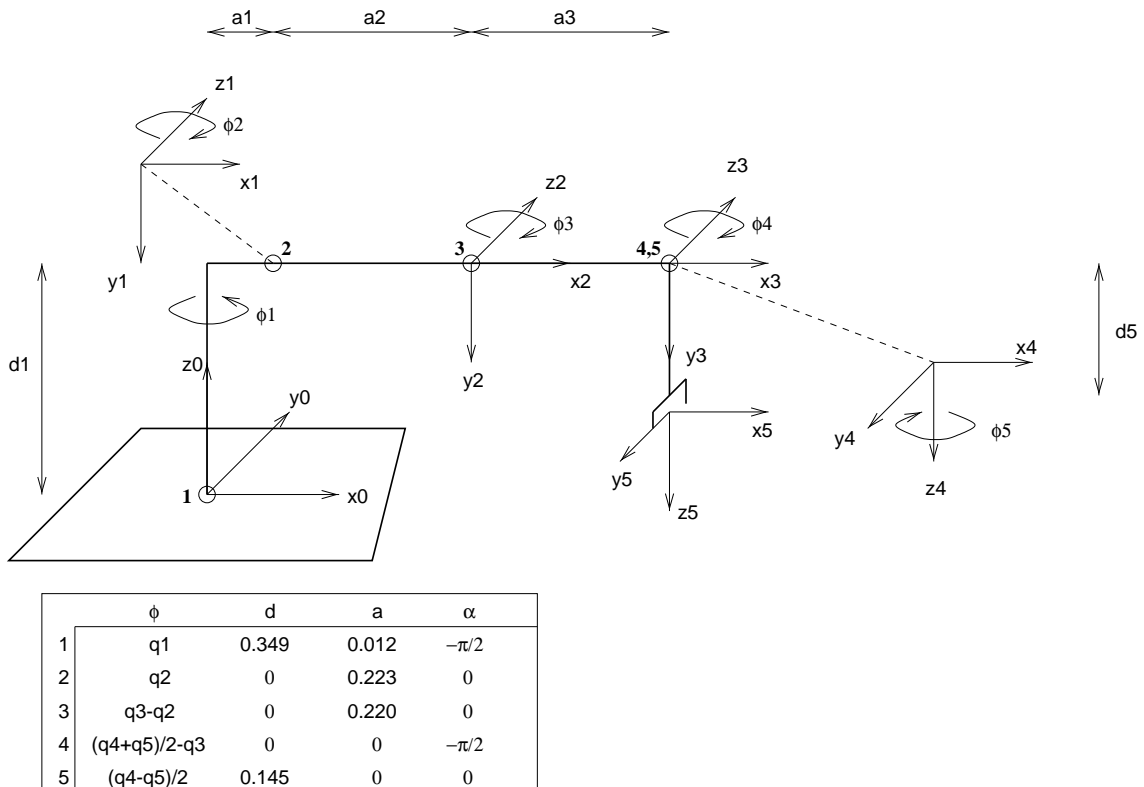
Scorboten har 5 rotationsled, der styres af 5 motorer. Desuden har den en gribe funktion der styres af en 6. motor.

Positionen af de 5 motorer der kontrollerer rotationsledene, repræsenteres som vektoren: $\mathbf{q} \in R^5$. Positionen er ikke defineret på selve motorens aksel, men derimod efter den gearing der sætter motorakslen i forbindelse med robotens led.

11.2.1 Denavit-Hartenberg repræsentation

Det første skridt i analysen af robotten, er at tildele hvert led et koordinatsystem. Til det formål anvendes Denavit-Hartenberg metoden [8, Algorithm 2-5-1]. Resultatet fremgår af figur 11.2.1 og tabel 11.2.1.

Bemærk at tegningen viser robotten i positionen hvor $\phi = 0$, hvilket ikke svarer til robotens normale *home position*.



Figur 11.2: Denavit-Hartenberg repræsentation af Scorboten

	ϕ	d	a	α
1	q_1	0.349m	0.012m	$-\pi/2$
2	q_2	0	0.223m	0
3	$q_3 - q_2$	0	0.220m	0
4	$(q_4 + q_5)/2 - q_3$	0	0	$-\pi/2$
5	$(q_4 - q_5)/2$	0.145m	0	0

Tabel 11.1: Data til Denavit-Hartenberg repræsentationen

11.2.2 Arm matricen

Arm matricen transformerer koordinater i *tool-spids* koordinatsystemet over i verdenskoordinater. Arm matricen findes ud fra Denavit-Hartenberg repræsentationen, [8, Proposition 2-6-1 og equation 2-6-4].

Arm matricen er givet ved 11.1. Bemærk at der ikke er anvendt den korte notation fra [8], men at udtrykket er skrevet fuldt ud.

$$\mathbf{T}_{base}^{tool} = \begin{bmatrix} \frac{\cos(q_1)(\cos(q_4)+\cos(q_5))}{2} + \sin(q_1) \sin(\frac{q_4-q_5}{2}) & \cos(\frac{q_4-q_5}{2}) \sin(q_1) + \frac{\cos(q_1)(-\sin(q_4)+\sin(q_5))}{2} \\ \frac{(\cos(q_4)+\cos(q_5)) \sin(q_1)}{2} - \cos(q_1) \sin(\frac{q_4-q_5}{2}) & -(\cos(q_1) \cos(\frac{q_4-q_5}{2})) + \frac{\sin(q_1)(-\sin(q_4)+\sin(q_5))}{2} \\ -\frac{\sin(q_4)}{2} - \frac{\sin(q_5)}{2} & \frac{-\cos(q_4)+\cos(q_5)}{2} \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -(\cos(q_1) \sin(\frac{q_4+q_5}{2})) & \cos(q_1) (a_1 + a_2 \cos(q_2) + a_3 \cos(q_3) - d_5 \sin(\frac{q_4+q_5}{2})) \\ -(\sin(q_1) \sin(\frac{q_4+q_5}{2})) & \sin(q_1) (a_1 + a_2 \cos(q_2) + a_3 \cos(q_3) - d_5 \sin(\frac{q_4+q_5}{2})) \\ -\cos(\frac{q_4+q_5}{2}) & d_1 - d_5 \cos(\frac{q_4+q_5}{2}) - a_2 \sin(q_2) - a_3 \sin(q_3) \\ 0 & 1 \end{bmatrix} \quad (11.1)$$

11.2.3 Tool konfigurations vektoren

Konfigurations vektoren er beskrevet i [8, Definition 3-3-1]. Det er en søjlevektor med 6 rækker. De 3 første rækker danner en vektor, der angiver *tool-spidsens* position i basiskoordinatsystemet. De sidste 3 rækker danner en vektor, der er parallel med *tool spidsens* retningsvektor, og hvis længde er en funktion af tool spidsens rotation om sin egen z-akse.

Scorbotens tool konfigurations vektor er givet ved (11.2)

$$\mathbf{w} = \begin{bmatrix} \cos(q_1) (a_1 + a_2 \cos(q_2) + a_3 \cos(q_3) - d_5 \sin(\frac{q_4+q_5}{2})) \\ \sin(q_1) (a_1 + a_2 \cos(q_2) + a_3 \cos(q_3) - d_5 \sin(\frac{q_4+q_5}{2})) \\ d_1 - d_5 \cos(\frac{q_4+q_5}{2}) - a_2 \sin(q_2) - a_3 \sin(q_3) \\ -\left(e^{\frac{q_4-q_5}{2\pi}} \cos(q_1) \sin(\frac{q_4+q_5}{2})\right) \\ -\left(e^{\frac{q_4-q_5}{2\pi}} \sin(q_1) \sin(\frac{q_4+q_5}{2})\right) \\ -\left(e^{\frac{q_4-q_5}{2\pi}} \cos(\frac{q_4+q_5}{2})\right) \end{bmatrix} \quad (11.2)$$

11.2.4 Mekaniske overvejelser

En vigtig konsekvens af at Scorboten kun har 5 frihedsgrader, er at *tool spidsen* er begrænset til at bevæge sig i x_1 - y_1 planet¹.

Scorboten kan ikke bevæge sig ubegrænset. Dels er den forsynet med stopklodser, for at forhindre kabler i at blive snoede, og dels støder den ind i sig selv. Desuden er det begrænset hvor hurtigt Scorboten kan bevæge sig.

De fundamentale mekaniske begrænsninger er angivet i tabel 11.2. Vær opmærksom på at tabellen ikke tager højde for at *tool spidsen*, og den påmonterede motor, kan kollideres med underlaget, roboten selv, eller andre forhindringer. De angivne vinkelhastigheder er fra [23, Side 4–22]. De er behæftet med stor usikkerhed, og skal opfattes som vejledende.

I forbindelse med kollisionscheck, kan det være nyttigt at kunne beregne tool spidsens z_0 koordinat, der er givet ved (11.3), og tool spidsens afstand fra z_0 akse i x_0 - y_0 planet, der er givet ved (11.4). Begge udtryk er udledt fra positionsdelen af (11.2)

$$d1 - d5 \cos\left(\frac{q_4 + q_5}{2}\right) - a2 \sin(q_2) - a3 \sin(q_3) \quad (11.3)$$

$$\left| a1 + a2 \cos(q_2) + a3 \cos(q_3) - d5 \sin\left(\frac{q_4 + q_5}{2}\right) \right| \quad (11.4)$$

-170°	\leq	ϕ_1	\leq	170°	$ d\phi_1/dt $	$<$	$21^\circ/s$
-115°	\leq	ϕ_2	\leq	55°	$ d\phi_2/dt $	$<$	$34^\circ/s$
-150°	\leq	ϕ_3	\leq	150°	$ d\phi_3/dt $	$<$	$34^\circ/s$
-150°	\leq	ϕ_4	\leq	150°	$ d\phi_4/dt $	$<$	$120^\circ/s$
$-\infty$	$<$	ϕ_5	$<$	∞	$ d\phi_5/dt $	$<$	$85^\circ/s$
$0m$	\leq	<i>griber</i>	\leq	$0.065m$			

Tabel 11.2: Mekaniske begrænsninger

q_1	$7725/\pi$	<i>inkr/rad</i>
q_2	$6028/\pi$	<i>inkr/rad</i>
q_3	$6028/\pi$	<i>inkr/rad</i>
q_4	$3014/\pi$	<i>inkr/rad</i>
q_5	$3014/\pi$	<i>inkr/rad</i>
Griber	9820	<i>inkr/m</i>

Tabel 11.3: Gearforhold for robotens motorer

11.2.5 Gearings forhold

Motorcontrolleren måler motorakslens position i *inkrementer*. Hvor mange inkrementer der er på en omdrejning afhænger af hvor findelt den anvendte enkoder-skive² er opdelt. Hvor mange omdrejninger af motoraksen der skal til for at dreje det givne led på roboten et vist stykke afhænger endvidere af gearforholdet mellem motorakslens og robotens led.

I tabel 11.3 er det angivet hvor mange inkrementer der går til at dreje en radian om en given akse, eller øge afstanden mellem gribekløerne en meter.

¹Se figur 11.2.1

²Se [23, Side 4 - 15]

11.3 Invers kinematik

Formålet med den inverse kinematik, er at beregne motorernes positioner, som funktion af tool konfigurationen.

Den inverse kinematik realiseres ved numerisk eller symbolsk analyse af tool konfigurations vektoren \mathbf{w} .

Numerisk analyse kan anvendes generelt, uafhængigt af robotens mekaniske konstruktion. Numerisk analyse kan være beregningskrævende, og vanskelig at realisere i *real-time*.

Symbolsk analyse giver symbolske udtryk for motor-positionerne, som funktion af tool konfigurationen. Det er ikke specielt beregningskrævende at evaluere de symbolske udtryk. Den symbolske analyse er fuldstændig afhængig af robotens mekaniske konstruktion, og den kan være ret vanskelig at gennemføre.

11.3.1 Beregning af motorkonfigurationen

Scorbotten er en relativt simpel robot, hvorfor der anvendes symbolsk analyse.

Scorboten er en 5 akset vertikalt artikuleret robot, der minder meget om den der er gennemgået i [8, afsnit 3-4]. Eksemplet adskiller sig ved at $a_1 = 0$ og $a_4 \neq 0$ for RHINO XR-3, og ved at [8] definerer motorpositionerne \mathbf{q} som $\mathbf{q} = \phi$. Med forbehold for disse forskelle, er det analysen i [8] der ligger til grund for analysen af Scorboten.

Det sværeste problem er at isolere q_2 og q_3 . Nøglen til problemet er at betragte den trekant der dannes mellem led 2,3, og 4 (skulder, albue, og håndled). Alle 3 sidelængder kan findes ud fra robotens mål, tool spidsens koordinater (w_1, w_2 , og w_3), samt tool spidsens *pitch*.

Cosinus relationen bruges til at finde cosinus til albuens vinkel. Det kan ikke afgøres om vinklen er positiv eller negativ, men Scorboten tillader begge løsninger (med forbehold for mekaniske begrænsninger).

Cosinus og Sinus til q_2 findes med de samme omskrivninger³ som i [8, (3-4-10) og (3-4-11)], og q_2 findes med $\arctan2$ (se [8, Tabel 3-1]). Herefter giver q_3 sig selv.

Tabel 11.4 angiver de symbolske udtryk for scorbotens inverse kinematik. Når ligningerne anvendes i praksis er det nødvendigt at sikre sig imod tool konfigurationer der ikke kan realiseres.

Figur 11.3 viser et program til Mathematica, der kan bruges til at evaluere udtrykkene med.

Der er to problemer med det symbolske udtryk:

³Samme omskrivninger, ikke samme udtryk

1. Undtaget tilfældet hvor armen er strakt helt ud, er der — teoretisk — to måder at realisere enhver tool konfiguration på. Med positiv eller negativ albuevinkel.

Den eneste simple måde at vælge mellem løsningerne på, er ved at undersøge om den ene løsning er ugyldig pga. mekaniske begrænsninger. Hvis der stadig er to løsninger kan man undersøge hvor hensigtsmæssig hver af dem er, i forhold til den overordnede bevægelse, eller simpelthen altid foretrække den ene frem for den anden.

2. Når toolspidsen kommer tæt på z_0 akse vil numerisk unøjagtighed begynde at få indflydelse på (11.11). Hvis tool spidsen placeres på z_0 akse vil (11.11) evaluere til vinklen 0. Tool spidsen kan derfor ikke passere gennem z_0 akse, uden at hele robotten vil dreje med. Med andre ord tillader den foreslåede løsning ikke robotten at række bagud, selv om der ikke er noget mekanisk til hinder for det.

Det betyder ikke at der er punkter robotten ikke kan nå, men det betyder at den ikke altid vil vælge den optimale bevægelse, hvis der arbejdes tæt på z_0 akse.

11.3.2 Invers kinematik algoritme

I dette afsnit defineres en algoritme til at beregne invers kinematik

1. Undersøg vha. w_1, w_2 , og w_3 om toolspidsen skal placeres over underlaget, og uden for Scorboten selv — Afbryd hvis ikke.
2. Undersøg om tool orienteringen (w_4, w_5, w_6) ligger i $x_1 - y_1$ planet — Afbryd hvis ikke.
3. Beregn q_1 vha (11.11)
4. Undersøg om q_1 ligger i det tilladte interval — Afbryd hvis ikke.
5. Beregn θ (toolvinkel i forhold til lodret) og ϕ_5 vha. (11.5) og (11.6)
6. Beregn α vha. (11.9)
7. Undersøg om $-1 \leq \alpha \leq 1$ — Afbryd hvis ikke.
8. Beregn q_2 , for både den positive og negative albueløsning, vha. (11.12)
9. Undersøg, for begge albueløsninger, om q_2 ligger i det tilladte interval.
10. Beregn q_3 , for begge albue løsninger, vha (11.13)
11. Undersøg, for begge albueløsninger, om ϕ_3 ligger i det tilladte interval.
12. Undersøg, for begge albueløsninger, om ϕ_4 ligger i det tilladte interval.
13. Afbryd hvis ingen af albueløsningerne tilfredsstiller de mekaniske krav til q_2, ϕ_3 , og ϕ_4
14. Beslut om den positive eller negative albueløsning skal anvendes.
15. Beregn q_4 og q_5 vha. (11.14) og (11.15)

$$\theta = \arctan2(-\sqrt{w_4^2 + w_5^2}, w_6) \quad (11.5)$$

$$\phi_5 = \frac{\pi \log(w_4^2 + w_5^2 + w_6^2)}{2} \quad (11.6)$$

$$r = -a_1 + \sqrt{w_1^2 + w_2^2} + d_5 \sin(\theta) \quad (11.7)$$

$$z = -d_1 + w_3 + d_5 \cos(\theta) \quad (11.8)$$

$$\alpha = \frac{a_2^2 + a_3^2 - r^2 - z^2}{2 a_2 a_3} \quad (11.9)$$

$$\phi_3 = \pi \pm \arccos(\alpha) \quad (11.10)$$

$$q_1 = \arctan2(w_2, w_1) \quad (11.11)$$

$$q_2 = \arctan2[-(a_2 + a_3 \cos(\phi_3))z - a_3 \sin(\phi_3)r, (a_2 + a_3 \cos(\phi_3))r - a_3 \sin(\phi_3)z] \quad (11.12)$$

$$q_3 = \phi_3 + q_2 \quad (11.13)$$

$$q_4 = \theta + \phi_5 \quad (11.14)$$

$$q_5 = \theta - \phi_5 \quad (11.15)$$

$$(11.16)$$

Tabel 11.4: Ligninger til invers kinematik

```

Pos[x_] := (Sign[x] + 1) Abs[Sign[x]] / 2
ArcTan2[y_, 0_] := Pi / 2 Sign[y]
ArcTan2[y_, x_] := Pos[x] ArcTan[y/x] + (1 - Pos[x]) (ArcTan[y/x] + Sign[y] Pi)

d1 = .349
a1 = 0.012
a2 = 0.223
a3 = 0.220
d5 = 0.145

fi5 := Pi Log[w4^2 + w5^2 + w6^2] / 2
fi4 := ArcTan2[-Sqrt[w4^2 + w5^2], w6]
r := Sqrt[w1^2 + w2^2] + d5 Sin[fi4] - a1
z := w3 + d5 Cos[fi4] - d1
alpha := (a2^2 + a3^2 - r^2 - z^2) / (2 a2 a3) (* Cos[2Pi - (q3 - q2)] *)
fi3 := Pi - (Sign[elbow] ArcCos[alpha])
Cfi3 := - alpha
ASfi3 := Sqrt[1 - Cfi3^2] (* Abs[Sin[q3 - q2]] *)
Sfi3 := Sign[elbow] ASfi3 (* Sin[q3 - q2] *)

q1 := ArcTan2[w2, w1]
q2 := ArcTan2[-(a2 + a3 Cfi3)z - a3 Sfi3 r, (a2 + a3 Cfi3)r - a3 Sfi3 z]
q3 := fi3 + q2
q4 := fi4 + fi5
q5 := fi4 - fi5

w angives som w1, w2 ... w5
elbow skal være positiv hvis de løsninger hvor albuevinklen er negativ (albuen vender opad) skal benyttes.
Bemærk definitionen af sin(φ3) (ASfi3 og Sfi3)

```

Figur 11.3: Mathematica program til beregning af invers kinematik

11.4 Software

De programmer jeg har udviklet til Scorboten, er bygget ovenpå funktionsbiblioteket til Gefion, og består af tre lag:

1. Et funktionsbibliotek til Scorboten, der rummer funktioner til:
 - Initialisering af hardware.
 - En søgealgoritme der bringer robotten i *home*-position.
 - Styling af en eller flere motorers position og hastighed.
 - Aflæsning af en eller flere motorers position.
 - Test af om alle motorer er nået frem til deres ønskede position.
2. Et testprogram der:
 - Indlæser toolvektorer, griber-positioner, og tidsintervaller fra en fil.
 - Omsætter de indlæste værdier til joint-koordinater vha. algoritmen til invers kinematik.
 - Kører robotten hen i den angivne konfiguration i løbet af den angivne tid.
3. En håndediteret fil der indeholder en sekvens af toolvektorer, griberpositioner, og tidsintervaller; til at åbne en flaske med skruelåg, og skænke indholdet i to glas.

Programmerne er dokumenteret i kapitel 20

11.5 Konklusion

Scorboten fungerer langt bedre med min styreelektronik, end med den oprindelige, hvilket primært skyldes at Gefion kan styre motorenes hastigheder.

Det har vist sig at ledningerne fra Scorbotens motorer til dens stik er alt for tynde, idet der kan opstå flere volts spændingsfald over dem når motorerne belastes, hvilket begrænser motorenes ydelse. I første omgang har jeg hævet forsyningsspændingen til 20V for at kompensere for modstanden i ledningerne, men ledningerne bør udskiftes.

Scorboten er plaget af mekanisk slør og elasticitet, der gør at spidsen kan flyttes manuelt flere cm. uden at motorerne bevæges. Tyngdekraften virker imidlertid altid samme vej på de fleste af dens led, så sløret altid er til samme side. Robottens gentagenøjagtighed bliver derved overraskende god ($\leq 3mm$).

Den inverse kinenatik fungerer udmærket, idet den bringer roboten i den rigtige position inden for et par cm. Positioneringsfejlen skyldes dels robottens slør og elasticitet, men målefejl fra min opmåling yder givetvis også et bidrag. En kalibrering for slør og elasticitet, kombineret med en bedre opmåling vil, ifølge min vurdering, reducere positioneringsfejlen væsentligt.

Interfacet mellem Scorbot og Gefion har fungeret som testopstilling i en lang periode, hvor Scorboten, ved en enkelt lejlighed, har fungeret sammen med Cato. En fejltilslutning af motordriverne til en demonstration ødelagde imidlertid både Scorboten's og Cato's motordrivere. Jeg har ikke haft tid til at genopbygge interfacet mellem Gefion og Scorbot, og Scorboten er derfor ikke blevet endeligt integreret med Cato.

Kapitel 12

Integration

For at få motorer, sensorer, data-mater, I/O moduler etc. til at danne en helhed, har det været nødvendigt at lægge en del arbejde i montering, sammenkobling strømforsyning mv.

Figur 12.1 skitserer den ønskede integration af Cato's delkomponenter. Samme struktur gør sig gældende for James, blot uden Scorbot, joystick, sonar og overvågning af strømforsyning.

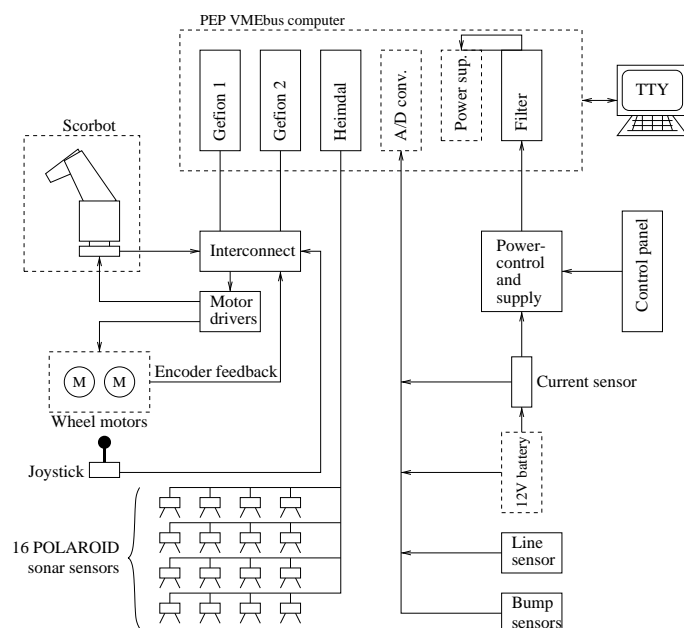
Dette kapitel giver en overordnet gennemgang af ideer, beslutninger mv. mht. integrationen af Cato og James. Cato og James er dokumenteret i hhv. kapitel 16 og 17.

Integrationen af Cato's delkomponenter er sket a' to omgange. Først blev de enkelte delkomponenter testet i midlertidige testopstillinger, der gradvist blev integreret mere og mere med hinanden, til et niveau

hvor Scorbot'en og Cato fungerede som en enhed, med joystickstyring og mulighed for at følge linier i gulvet.

De midlertidige testopstillinger var ikke robuste og brugervenlige nok til anvendelse på langt sigt. Efter et uheld¹ der ødelagde otte motordrivere og et A/D-convertermodul, blev de midlertidige opstillinger kasseret.

De delkomponenter jeg har bidraget med til James er integreret og dokumenteret til et færdigt niveau, mens integrationen af delkomponenterne på Cato ikke er blevet prioriteret så højt som James og dermed ikke er fuldstændigt integrerede endnu.



Figur 12.1: Integration af Cato's komponenter

¹Fejltilslutning af spændingsforsyning ved en demonstration.

12.1 Strømforsyning

I både Cato og James, skal strømforsyningen levere energi til:

- 1 Styredatamat á (5V/5A, +12V/0.5A, -12V/0.5A)
- 2 Motorer á (Cato: 12V/0-17A James: 7.2V/0-25A)
- Anden elektronik (5V, ± 12 -15V, og 12V)

Den eneste relevante energikilde i forbindelse med AGV'er af Cato's og James beskaffenhed, er opladelige batterier, der kan bruges mange gange og som findes i udgaver der kan levere den nødvendige effekt.

Opbygningen af Cato's og James strømforsyninger er dokumenteret i afsnit 16.2 og 17.2.

12.1.1 Batterier

Frem for at splitte strømforsyningen op på separate batterier til motorer, datamater osv. foretrækker jeg at samle energiforsyningen i et enkelt batteri for at undgå problemer med uens op/afladetid, levetid etc. og for at gøre batteriskift nemt og hurtigt.

I Cato, anvendes en 12V 38Ah blyakkumulator (vægt 10kg), der giver en aktionstid på 2-6 timer, afhængigt af hvor meget og hvordan der køres. Jeg anvender en lukket type med fast elektrolyt, der tåler total afladning og opbevaring i afladet tilstand.

Egenskab	Cato	James
Antal batterier	1	1
Polspænding	12...15V	12...15V
Strømforbrug, ved stilstand	4A	3.5A
Strømforbrug, nominel last	15A	15A
Strømforbrug, max last	32A	50A
Mindste acceptable aktionstid	1 time	3 min
Max batteristørrelse	40 × 16 × 16 cm	10 × 10 × 16 cm
Max batterivægt	30 kg	1 kg

Tabel 12.1: Krav til batterier

I James anvendes 12 seriekoblede 1.3Ah

NiCd elementer (ialt 0.5kg), der giver en aktionstid på 3-15 minutter.

12.1.2 Spændingskonvertering

For at imødekomme de forskellige delkomponenters krav til forsyningsspændingen anvendes *switched-mode* DC-DC convertere til at forsyne datamater og følsom elektronik. I James er batterispændingen dobbelt så stor som motorernes nominelle polspænding. For at forhindre overbelastning af motorerne opbygges motordriveren i James så den effektive polspænding begrænses til lidt over 50% af batterispændingen.

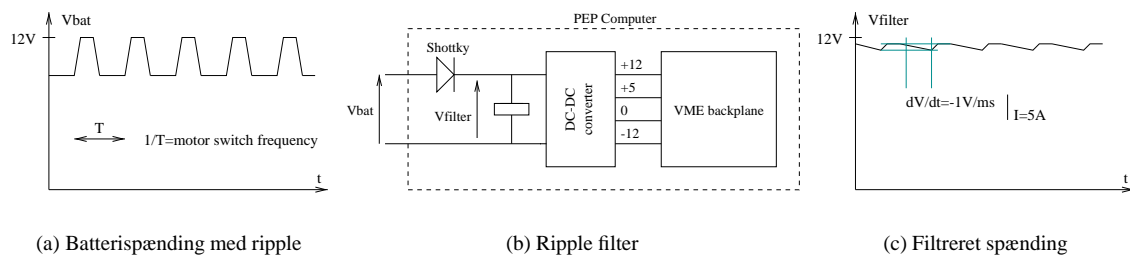
12.1.3 Op-/afladning

Opladning af Cato's batteri sker med et almindeligt ladeapparat til bilbatterier eller en reguleret strømforsyning (14.8V, strømbegrænset til max. 9A). Jeg har forsynet Cato med 3 m Ω shuntmodstand i serie med batteritilslutningen, for at muliggøre overvågning af strømforbrug og lade strøm, men ikke nået at udvikle elektronik og software til aflæsning af strømmen og batterispændingen.

James batterier frakobles James, og oplades med en ekstern NiCd lader, eller med en reguleret strømforsyning (16V strømbegrænset til 130mA i 14 timer — tåler overladning, eller 16V strømbegrænset til 1.5A i 1 time — tåler ikke overladning og nedsætter levetiden.) For at undgå *memoryeffekt* bør batterierne aflades helt før de genoplades.

12.1.4 Filtrering af ripple støj

Når AGV'ens motorer er hårdt belastet (ved acceleration), introducerer *switched-mode* motordrivers uens strømforbrug en del *ripple* på Cato's 12V batterispænding. Batterispændingen kan komme under 9V i nogle få mikrosekunder ad gangen, hvilket får de anvendte DC-DC convertere til at afbryde strømforsyningen af bl.a. datamaten. For at opretholde en indgangsspænding på over 9V til converterne foretages en ensretning og udglatning af spændingen fra batteriet. Der anvendes en $4700\mu F$ HF elektrolytkondensator, der ved 5A forbrug, begrænser spændingsfaldet til ca. 1V/ms. For at minimere spændingsfaldet i filteret anvendes en *shottky barrier diode*.



Figur 12.2: Skitse af ripple og filtrering i strømforsyning

12.2 Datamat

Både Cato og James styres af en VME-bus datamat fra *PEP modular computers*, der er bestykket med et VMPM68KC2 68020 baseret CPU-kort.

12.2.1 Mekanik

PEP datamaterne består af et VME-bus *S1 backplane* med plads til 9 moduler, der sammen med en strømforsyning, en 3.5" diskette station, og en 3.5" harddisk er monteret i et 3U 19" rack. CPU-modulet, *discontroller*-modulet, og alle nødvendige I/O moduler monteres lodret i racket.

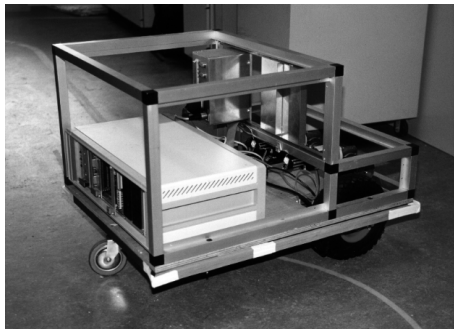
Cato er designet så der netop er plads til to 3U, eller et 6U 19" rack, mens der under ingen omstændigheder er plads til et rack på James.

På James er der lige akkurat plads til at placere et 7-moduls J1 VME *backplane* på tværs af dækket. *Backplanet* monteres på en afkortet 19" *subrack* ramme, der giver opstillingen mekanisk stabilitet, men ingen mekanisk/elektrisk indkapsling. Strømforsyning og diskettestation monteres uden for rammen (se figur 12.3-b).

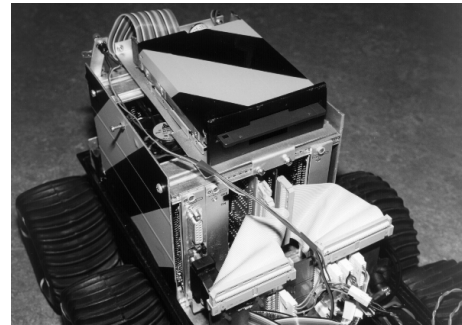
I både James og cato, beslaglægges 5 pladser i *backplanet* af CPU modul (fylder to moduler), *discontroller*, et *A/D converter* modul, og et Gefion motorstyringsmodul. I Cato bruges yderligere et Heimdal sonarinterface modul, og et ekstra Gefion modul hvis Scorboten anvendes.

12.3 Motorstyring

Med Scorboten koblet sammen med Cato, er der behov for styring af 8 motorer. Til det formål anvendes to Gefion motorstyrings moduler, der monteres i PEP styredatamanten. På James, er der kun behov for at styre to motorer, hvilket klares med et enkelt Gefion modul.



(a) Cato



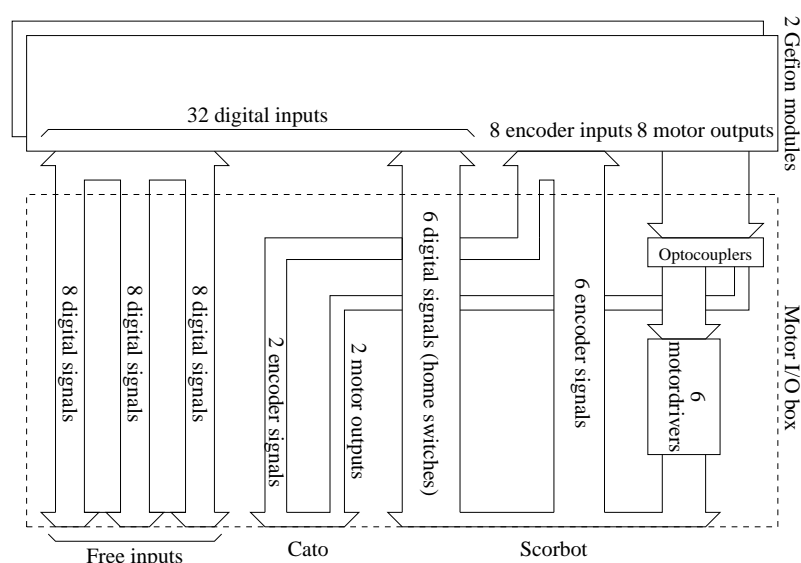
(b) James

Figur 12.3: PEP VME-bus datamat på Cato og James

Gefion har af pladshensyn alle sine I/O forbindelser samlet i et enkelt 50 polet IDC stik. For at dele Gefions I/O forbindelser op i funktionelle grupper, har jeg fremstillet to passende *interconnect* print til Cato og James.

Interconnect printet til Cato, er udviklet i forbindelse med en større opstilling — motor I/O box, der samler alle motorrelaterede forbindelser i et enkelt kabinet, med tilslutning direkte til Scorboten, og Cato's egne motordrivere. Motor I/O boxen er dokumenteret i afsnit 16.5. Motor I/O boxen et pt. ikke helt færdiggjort idet jeg mangler at opbygge og montere 6 nye motordrivere til erstatning for dem der tidligere blev ødelagt ved en fejltilslutning. De 6 manglende motordrivere skal bruges til at drive scorbotens motorer.

På James er alle I/O signaler til motorstyring og A/D-convertere samlet på et enkelt interconnectboard, der er dokumenteret i afsnit 17.4.



Figur 12.4: Blokdiagram over motor I/O box

12.4 Sensorer

Cato er udstyret med 500 mm lange, 25 mm brede, og 1 mm tykke strimler af trykfølsomme modstande, der fungerer som kollisionss og berøringssensorer. Strimlerne er limet på kanterne af det mekaniske skelet, så hver side deles ind i tre zoner vha. to forskudte strimler². Der limes en gummiliste uden på modstandene for at beskytte dem og fordele trykket fra et sammenstød. Trykket på modstandene omsættes til en spænding ved at lade hver modstand indgå i en halvbro. Bumpsensorerne er dokumenteret i afsnit 16.3.

Liniesensoren (kapitel 9) monteres fast under Cato, midt mellem drivhjulene.

Siden Cato's A/D-convertermodul blev ødelagt ved en fejltilslutning har jeg ikke rådet over analoge indgange på Cato's styredatamat. For at færdiggøre integrationen af bump- og linesensorer skal Cato udstyres med et nyt A/D modul, der skal forbindes med sensorerne.

Elektrisk og funktionelt er POLAROID's sonarafstandsmålerne fuldt integreret med Cato's og James styredatamater, samt alle andre VMEbus datamater, vha. VMEbus-modulet Heimdal. Det er min tanke at afstandsmålerne skal indbygges i små individuelle kasser, der kan monteres på beslag på Cato, James, eller i andre opstillinger efter behov. Indbygning af sensorerne har dog ikke været prioriteret højt og indtil videre monteres de vha. primitive plasticholdere og skruer eller tape.

Den eneste eksterne sensor på James (1997) er en speciel linesensor med 7 analoge udgange, der forbindes til et *Analog Devices RTI 600* A/D modul, via et interconnectboard. Liniesensoren på James er udviklet af andre studerende og er ikke behandlet i denne rapport. I 1998 blev James udstyret med to digitale *følehorn* der blev forbundet til to digitale indgange på Gefion via. det oprindelige interconnectboard.

12.5 Software

Det meste af den softwaremæssige integration jeg har foretaget ligger i funktionsbiblioteket til motorcontrolleren Gefion (afsnit 6.5 og 18.9), device driveren til sonarinterfacet Heimdal (kapitel 8 og afsnit 19.15) og funktionsbiblioteket til Scorbot'en (kapitel 20) Integration af A/D-converter er banal og kræver ikke device drivers eller funktionsbiblioteker.

For at afprøve og demonstrere AGV'erne som samlede systemer har jeg lavet nogle primitive programmer der integrerer motorstyringen med sensorinput.

12.5.1 Kørsel med joystick

Ved at koble et standard digitalt (commodore C64 kompatibelt) joystick til Gefions digitale indgange kan AGV'en styres manuelt ved at skrive et simpelt program der justerer motorhastighederne efter joystickens konfiguration.

Programmet er en uendelig løkke der aflæser de fem digitale signaler fra joysticket (frem, bagud, højre, venstre og knap) og øger eller sænker hastigheden på højre og venstre hjul i små skridt afhængigt af styrepindens retning. Knappen bruges som nødstop.

12.5.2 Kørsel efter linie

Ved at aflæse differenssignalet fra linesensoren (kapitel 9) kan AGV'ens afstand fra en linie på gulvet vurderes. Vha. det primitive program i figur 12.5³ kan AGV'en følge linien hvis reguleringsparameteren og hastigheden vælges rigtigt og linien ikke har for små krumningsradier. Programmet justerer krumningen

²Budgettet tillod kun indkøb af fem modstande, der er monteret foran og på siderne.

³C-lignende pseudokode

af AGV'ens bane så den er proportional med afvigelsen fra linien, en metode der meget nemt bliver ustabil. Metoden kan forbedres væsentligt ved at tage afvigelsens 1. afledede med i betragtning.

```
#define STEP ??
#define P ??

init_motors();
speed=0;
while(read_joystick()!=knap) {
    if (read_joystick()==frem) speed=speed+STEP;
    if (read_joystick()==bak) speed=speed-STEP;
    x=read_linesensor(); /* Positiv når vi er til højre for linien */
    x=x*P;                /* P er reguleringsparameteren */
    setspeed_rightwheel(speed+x);
    setspeed_leftwheel (speed-x);
}
setspeed_rigthwheel(0);
setspeed_leftwheel(0);
```

Figur 12.5: Simpelt program til at følge en linie

12.5.3 Kørsel langs væg

Ved at montere en sonarafstandsmåler på siden af Cato, kan der udvikles et program der får Cato til at køre parallelt med en væg. Metoden er vist i figur 12.6 og baserer sig på samme princip som programmet til at følge linier med. I denne udgave af programmet anvender jeg både afstanden til væggen og den 1. afledede af afstanden, til at styre krumningen med. Metoden fungerer ganske overbevisende i praksis, omend der skal tages højde for åbne døre når der køres langs en væg. Et åbenlyst problem ved metoden er at afstandsmåleren ikke måler den korteste afstand til væggen, men afstanden vinkelret på AGV'ens bevægelsesretning. Hvis AGV'en drejer for skarpt ind mod væggen vil afstandsmåleren måle en større og større afstand efterhånden som AGV'ens vinkel til væggen bliver større, hvilket får AGV'en til at dreje hurtigere og resultatet bliver ofte en frontal kollision med væggen.

12.5.4 DTU-Robocup

Mine testprogrammer er banale og har kun til formål at vise at Cato's enkelte dele fungerer sammen. De hidtil bedste eksempler på softwaremæssig integration af AGV-komponenter i denne forbindelse, er foretaget af de studerende der stod bag programmeringen af James til DTU-Robocup konkurrencen i 1997 og 1998. Jeg har ikke selv deltaget i programmeringen af James.

12.6 Problemer og erfaringer

Det største problem mht. integration har været de mekaniske aspekter. Især mht. indbygning og montering af elektronik, stik mv. har det været meget svært at opnå et fornuftigt niveau af robusthed med de faciliteter jeg har haft adgang til og min egen begrænsede erfaring som mekaniker. Selve det mekaniske fundament for Cato er særdeles robust og fungerer udmærket. Cato kan uden problemer køre på et plant underlag med en last på 100kg.

Mht. effektelektronikken har jeg haft nogle EMC-problemer der skyldes en dårligt implementation af det kredsløb der står for styring af Cato's H-bro. Kredsløbet er opbygget på et prototype-hulprint (det såkaldte


```

#define STEP ??
#define P ??
#define D ??

init_motors();
init_sonar();
speed=0;
while(read_joystick()!=knap) {
    if (read_joystick()==frem) speed=speed+STEP;
    if (read_joystick()==bak) speed=speed-STEP;
    x=read_distance_right(); /* Sensoren sidder i højre side */
    x=x-desired_distance; /* Vi er interesserede i afstandsfejlen */
    dx=calculate_dx_from_stored_x_values(x);

    x=x*P; /* P er proportionalparameteren */
    dx=dx*D; /* D er differntialparameteren */
    setspeed_rightwheel(speed+x+dx);
    setspeed_leftwheel (speed-x-dx);
}
setspeed_righthwheel(0);
setspeed_leftwheel(0);

```

Figur 12.6: Simpelt program til at følge en væg vha. sonar

power-board — afsnit 16.2.1) , hvor forbindelserne dannes ved at lodde ledninger på undersiden af hulprintet. Problemerne skyldtes støj fra H-broerne koblet via stelledninger og blev løst ved at ændre føringen af kredsløbets stelforbindelser.

Der er stadig nogle få montagemæssige problemer med Cato, der primært går på at fremstille og montere diverse beslag kasser og dækplader. Problemerne er imidlertid små og med hjælp fra et af universitetets mekaniske værksteder kan de relativt let løses.

James er simplere og mindre end Cato, samtidigt med at den er bygget over en modelbil til fjernstyring. Disse faktorer har gjort det nemmere at integrere komponenterne på James til en robust enhed, der til gengæld ikke har de store muligheder for udvidelser. De væsentligste integrationsmæssige problemer med James er det åbne VME-rack og den sårbare placering af disktestationen (figur 12.3-b).

Pga. et uheld der ødelagde en del af mit udstyr er Scorbot, linie- og bumpsensorer endnu ikke endeligt integreret med Cato. Før uheldet var Cato integreret med både Scorbot og linesensor, og den endelige integration er blot et spørgsmål om at genetablere 6 motordrivere og et A/D-converter modul.

12.7 Konklusion

Både Cato og James er integreret til fungerende enheder hvor der via. software kan opnås et samspil mellem sensorer og motorer. Begge AGV'er egnede til videre brug af andre studerende. Begge AGV'er har behov for forbedringer mht. montage og indkapsling, men har ikke væsentlige problemer. Jeg har dokumenteret integrationsaspektet på samme niveau som resten af projektet og hvis der er interesse for det, bør dokumentationen gøre det muligt for andre at fortsætte integrationen ud fra oplysningerne i kapitel 16 og resten af denne rapports dokumentationsdel.

Kapitel 13

Konklusion

I løbet af dette projekt har jeg udviklet et selvstændigt robotkøretøj, en AGV¹, kaldet Cato. Cato er blevet til ved at videreudvikle og integrere eksisterende mekanik, datamat mv. med sensorer, motorstyring og en række andre komponenter, jeg selv har udviklet. En stor del af komponenterne og erfaringerne fra Cato indgår i AGV'en James, som jeg byggede sammen med en række andre studerende til konkurrencen DTU-RoboCup 1997. James er i store træk en nedskaleret udgave af Cato og det er rimeligt at betragte dem som to eksempler på integration af samme teknologi.

Mit arbejde gennem projektet har koncentreret sig om fremstilling af relevante og pålidelige delkomponenter til AGV'erne, der sammen med min dokumentation og det udviklede lavniveauprogrammel gør det nemt at anvende AGV'ernes faciliteter på et højt abstraktionsniveau. Valget og udformningen af de enkelte delkomponenter og strukturen af AGV'en som helhed (se kapitel 3) er sket på baggrund af mit kendskab til og undersøgelse af AGV'er, AGV-navigation mv. (se kapitel 2). Mange vigtige delkomponenter er udviklet helt fra idéstadiet til en fungerende reproducerbar prototype. Som regel fordi noget tilsvarende ikke fandtes i handlen, men i enkelte tilfælde fordi budgettet ikke tillod at indkøbe tilsvarende udstyr.

Tabel 13.1 opsummerer indholdet i projektet, der kan inddeles i hovedgrupperne: motorstyring, sensorer, navigation og andet. Tabellen viser emnerne i hver hovedgruppe og henviser til de steder i rapporten, der er relevante for hvert emne.

Med Cato og James er de delkomponenter jeg har udviklet integreret i to AGV'er, der kan bruges som platforme for eksperimenter og demonstrationer af samspillet mellem sensorer og styring. Begge AGV'er er robuste og veldokumenterede nok til at indgå i længere projektrækker. Med mindre der skal inddrages nye sensorer, er den hardwaremæssige udvikling af Cato og James færdiggjort til et niveau, hvor der er tale om vedligehold og forbedringer. Projektet har dermed opfyldt sit formål og bragt AGV-teknologien ved Odense Universitet til et stadie hvor den fremtidige udvikling primært ligger i højniveauprogrammel og integration af nye sensorer.

13.1 Motorstyring

Til styring af motorerne i Cato, James, og den lille robotarm: **Scorbot**, har jeg udviklet et avanceret motorstyringsmodul, der er i stand til at styre op til fire motorer med minimal belastning af CPU'en i værtsdatamaten (se kapitel 6). Modulet er generelt og kan anvendes til styring af elektromotorer og andre aktuatorer i kontrolsystemer, der anvender kvadraturenkodere som positionsgivere. Der er fremstillet to eksemplarer af motorstyringsmodulet, der bruges til styring af Cato, James, Scorbot og en øvelsesopstilling² til et kursus i kontrolteori. Der kan fremstilles flere moduler efter behov. Det fremstillede motorstyringsmodul tåler

¹ Autonomous Guided Vehicle

² En lineær kran der flytter et hængende lod

Overordnet	Emne	Kapitel	Delkonklusion Afsnit	Dokumentation Kapitel
Motorstyring	Motorstyring (overordnet)	4		^a 23
	VME-modul til motorstyring	6	6.7	18
	Tykfilmbaseret motordriver	10	10.7	22
Sensorer	VME-modul til sonarafstandsmåling	7	7.5	19
	Device driver til sonarafstandsmåling	8	8.8	19
	Liniesensor	9	9.6	21
	Berørings-/kollisionssensorer			16
Navigation	Emneoversigt	2		
	Styring af Scorbot	11	11.5	20
Andet	Udvikling af VME-bus slavemoduler	5	5.9	18 , 19
	Udvikling af tykfilmt teknik ^b	10	10.7	22
	Integration	12	12.7	16 , 17
	Programmering under OS-9			^a 24

^a ufuldstændig note.

^b Gennemgået som en del af emnet: *Tykfilmbaseret motordriver*.

Tabel 13.1: Projektindhold

sammenligning med tilsvarende kommercielle produkter mht. funktionalitet og størrelse. Modulet kunne i princippet udvikles til et kommercielt produkt.

Jeg har udviklet et C funktionsbibliotek, der bl.a. muliggør styring af motorernes position, hastighed og acceleration, på et højt abstraktionsniveau.

For at gøre motorstyringsmodulet i stand til at regulere de anvendte motorer, er der fremstillet flere forskellige typer motordrivere, der kan styre energitilførselen som funktion af motorstyringsmodulets udgangssignaler. Bl.a. er der fremstillet en meget kompakt driver baseret på tykfilmt teknologi (se kapitel 10). I forbindelse med tykfilm-motordriveren har jeg udviklet en lovende metode til fremstilling af tykke effektbærende lederbaner, der kan bruges af laboratorier og virksomheder, der ikke råder over udstyr til de komplicerede fremstillingsmetoder, der normalt anvendes i tykfilmbaseret effektelektronik. Der er ikke lagt vægt på grundig efterprøvning af fremstillingsmetoden, men med de muligheder den åbner for mindre laboratorier, vil det være interessant at undersøge dens holdbarhed, anvendelighed og nyhedsværdi nærmere.

13.2 Sensorer

Ud over enkoderne i motorstyringen, har jeg arbejdet med sonarafstandsmålere, berørings-/kollisions-følere og en sensor til at følge en linie på gulvet.

Til afstandsmåling med sonar er der udviklet et modul, der fungerer som grænseflade mellem styredatamaten og de velkendte sonar-afstandsmålere fra POLAROID (se kapitel 7). Modulet giver mulighed for tilslutning af 16 POLAROID afstandsmålere, 4 samtidige målinger og detektion af over 100 ekkoe i hver måling. Jeg har udviklet en *device driver*, der integrerer styringen af sonarmodulet med styredatamatens operativsystem, så modulet kan anvendes via operativsystemets normale I/O rutiner. Muligheden for samtidig brug af flere POLAROID sensorer og detektion af flere ekkoe i hver måling åbner interessante muligheder for kortlægning af sensorernes omgivelser vha. avanceret databehandling. Sammenholdt med den gennemgåede litteratur og mit øvrige kendskab til området, udgør modulet et af de mest avancerede systemer til sonarafstandsmåling med POLAROID's, eller tilsvarende, sensorer.

Til registrering af berøring og sammenstød, har jeg forsynet Cato med strimmelformede tryksensorer langs kanterne. Sensorerne dækker kanten rundt om Cato og omsætter kraftpåvirkninger til en spænding, der er egnet til aflæsning vha. styredatamatens A/D omsætter.

Jeg har udviklet en analog optisk sensor, der gør AGV'en i stand til at følge en linie på gulvet (se kapitel 9). Sensoren, der fungerer under normale belysningsforhold, er i stand til at skelne afvigelser på under $100\mu\text{m}$. Sensoren kan bruges til nøjagtig kørsel efter en linie i forbindelse med kalibrering af AGV'ens mekanik, kørsel gennem snævre passager og lignende.

13.3 Navigation

Jeg har sat mig ind i forskellige principper for sensorbaseret AGV-navigation og valgt at arbejde med sensorer, der understøtter bestiknavigation, kørsel efter streger på gulvet og geometrisk kortlægning af omgivelserne.

Forskellige kombinationer af bestiknavigation og geometrisk kortlægning er meget udbredte og godt dokumenteret (se kapitel 2). Kortlægning vha. sonar er tidligere blevet behandlet ved Odense Universitet [5] og metoderne herfra kan overføres til Cato.

Holdet bag James gav en overbevisende demonstration af kørsel efter linier kombineret med bestiknavigation, da James gennemførte DTU-RoboCup 97 på rekordtid ([92] og [93]). Selv har jeg demonstreret brugen af Cato's liniesensor og sonarafstandsmålere ved henholdsvis at følge en tapestrimmel på gulvet og køre parallelt med en væg.

13.4 Dokumentation

Jeg har lagt vægt på at dokumentere mit arbejde med henblik på at sætte andre studerende i stand til at bruge og videreudvikle det. Denne rapports del II og III rummer lidt over 200 siders teknisk dokumentation, brugervejledning og noter. Dele af dokumentationen har allerede været brugt af andre studerende i forbindelse med: Udvikling af OS-9 device-drivere, brug af POLAROID's sonar-afstandsmålere, sensorudvikling og programmering af James til DTU-RoboCup i 1998, udvikling af MOS-FET baseret motordriver, brug af PEP datamat til styring af hydraulisk aktuator og udvikling af VMEbus moduler.

13.5 Genbrug af materialer og udstyr

For at projektet kunne realiseres med et beskedent budget, har jeg genbrugt en del materialer og udstyr fra tidligere projekter. Selve Cato's platform og motorer, samt 8 POLAROID afstandsmålere stammer fra et sommerkursus i 1992. VME-datamaterne, der bruges i begge AGV'er, har tidligere været brugt i praktisk orienterede projekter, men har pga. manglende aktiviteter været opmagasineret.

Jeg har integreret den lille robotarm **Scorbot** med to af mine motorstyringsmoduler. I modsætning til den oprindelige styreenhed, kan mine moduler styre dens 6 motorer synkront (se kapitel 11). Jeg har desuden opstillet en invers kinematik for Scorboten, og skrevet et C funktionsbibliotek der sætter en bruger i stand til at styre Scorboten i både kartesiske- og joint-koordinater.

Kapitel 14

Fremtidig udvikling

14.1 Navigation

Skal Cato eller James kunne bevæge sig rundt uden linier på gulvet, skal de sættes i stand til at kombinere bestiknavigation med en eller flere metoder til positionsbestemmelse og ruteplanlægning.

Bestiknavigation kan udvikles ved at opmåle AGV'ens køreegenskaber og udvikle forskellige programstumper, der kan få AGV'en til at bevæge sig ligeud og foretage forskellige fundamentale sving og manøvrer med god nøjagtighed. Ved at bruge linesensoren i forbindelse med forskellige testbaner, kan opmålingen automatiseres med henblik på jævnlig kalibrering. Udvikling af bestiknavigation kunne være temaet for et bachelorprojekt eller indgå i et speciale- eller afgangsprøveprojekt.

Når AGV'en kan foretage en sekvens af *standardmanøvrer* og holde rede på sin relative position vha. bestiknavigation, mener jeg, at det vil være interessant at kombinere bestiknavigation med positionsbestemmelse, ruteplanlægning og sonar-kortlægning. Et eller flere specialeprojekter i dette emne vil formentligt kunne drage nytte af ekspertisen indenfor billedbehandling, computergrafik, grafteori og robotstyring, der findes i og omkring datateknologimiljøet.

14.2 Sensorer

De sensorer, jeg har arbejdet med i dette projekt, er ifølge den gennemgåede litteratur tilstrækkeligt grundlag for at sætte AGV'erne i stand til at navigere i et indendørsmiljø med gange og åbne rum. For at opnå dette skal der imidlertid udvikles programmet til at behandle, analysere og sammenholde sensordata med kort over omgivelserne.

For at lette og forbedre positionsbestemmelse, kortlægning og undgåelse af kollisioner, kan det være interessant at anvende eller udvikle andre sensorer som supplement til dem jeg har valgt.

Absolut positionsbestemmelse vil blive vanskeligt med de nuværende sensorer. Ved at supplere med kunstige kodede eller ukodede kendmærker i omgivelserne kan opgaven gøres betydeligt lettere. Der findes en lang række forskellige principper for at implementere aktive og passive kendmærker og tilhørende sensorer (se kapitel 2). En litteraturgennemgang og sammenlignende afprøvning af forskellige metoder vil være interessant.

POLAROID's sonarafstandsmålere dækker afstande fra 0.15 - 10m med god nøjagtighed, men er ganske uanvendelige til registrering af de helt nære omgivelser. For at lette navigation i snævre passager, dørkarme og møblerede rum er det relevant at arbejde med forskellige former for nærhedssensorer. Principperne

beskrevet i artiklen: *Multi-functional optical proximity sensor by using phase information*[37] kunne være et interessant udgangspunkt for f.eks. et bachelorprojekt eller lignende indenfor dette felt.

Muligheden for at registrere mere end det første ekko fra POLAROID's sonarafstandsmålere åbner måske bedre muligheder for kortlægning og analyse af omgivelserne idet det er muligt at se forbi forhindringer. POLAROID's afstandsmålere registrerer imidlertid kun når styrken af det reflekterede signal overskider en vis grænseværdi. Der vil kunne indhentes langt mere information om omgivelserne ved at registrere og analysere det reflekterede signal direkte. Et eller flere projekter der undersøger og afprøver mulighederne for anvendelse af sonar i luft, vil være meget interessante i forbindelse med kortlægning af robotters omgivelser. Både Biologisk institut og Institut for teknik og fysik ved Odense Universitet er interessante samarbejdspartnere med deres ekspertise i ultralyd, akustik og signalbehandling, der stammer fra forskning i hhv. biosonar¹ og akustik.

Dopler radarsensoren, som jeg omtalte i afsnit 3.6.6, kan anvendes i robotteknologi på to måder. Hvis den udstyres med almindelig hornantenne fylder den nærmest et rum op med et felt af stående bølger, så sensoren kan anvendes som langtrækkende bevægelsessensor. Anvendes en utæt *waveguide* (et metalrør med huller) som antenne spredes energien, så sensoren kan bruges til at registrere bevægelse i nærheden af waveguiden. Begge metoder kræver at der tages højde for signaler, der skyldes sensorens egenbevægelse og begge metoder er interessante i AGVteknologi såvel som i industriroboter til registrering af bevægelse og nærhed. Det vil være interessant at foretage en nærmere analyse/afprøvning af muligheder og problemer ved anvendelse af dopplerradar i AGV- og robotteknologi.

Lawrence Livermore National Laboratories har udviklet en radarafstandsmåler baseret på *time of flight*[72], der på mange måder minder om POLAROID's sonarafstandsmåler, men med langt højere målefrekvens. Radarsensoren er et interessant alternativ til sonar bl.a. fordi den kan indkapsles bedre og gøres mere robust. Det vil være interessant at skaffe en eller flere radarsensorer og undersøge dens muligheder som erstatning eller supplement til sonar.

14.3 Effektelektronik

Det arbejde, jeg har udført indenfor effektelektronik og tykfilm, har kun indirekte interesse for AGV- og robotteknologi, men er særdeles interessant i sig selv. Det vil være spændende at fortsætte udviklingen af min metode til trykning af tykke lederbaner og mht. robotteknologi vil det være særdeles interessant at udvikle et lille og effektivt *switched-mode* udgangstrin med integreret CPU. (Se kapitel 10 og afsnit 10.7.1.) Det omtalte udgangstrin vil ikke kun være interessant i forbindelse med elektromotorer, men også i forbindelse med styring af ventiler til hydrauliske og pneumatiske aktuatorer indenfor robotteknologi [30], hvor mineaturisering og modularitet er vigtige egenskaber.

14.4 Projektideer

Blandt alle ideerne til videre udvikling er der en del der kan udkrystalliseres i projekter der kan påbegyndes med det samme f.eks:

1. **Udvikling af devicedriver til Gefion:** Det funktionsbibliotek jeg har udviklet til Gefion, er blot en *front-end* til de funktioner der understøttes af hardwaren. Ved at udnytte Gefions evne til at generere interrupts når hver motor har kørt en forudprogrammeret strækning vil det være muligt at udvikle en OS-9 device-driver der tilbyder en langt mere avanceret grænseflade til motorerne.
2. **Udvikling af programmel til bestiknavigation og standardmanøvrer:** Ved at bruge linesensoren til at kalibrere AGV'en på forskellige testbaner kan dens køreegenskaber opmåles med god nøjagtighed. Med kendskab til AGV'ens køreegenskaber er det muligt at lave et funktionsbibliotek

¹Bla. flagermus brug af sonar

der tilbyder forskellige standardmanøvrer, hvor der løbende kan holdes styr på AGV'ens position og orientering. Denne opgave kan med fordel kombineres med forslag 1

3. **Udvikling af metoder og programmel til analyse af SONAR-målinger:** Vil man bruge mere end en sonarafstandsmåler ad gangen, eller bruge mere end et ekko, opstår der komplikationer med indirekte interferens fordi lydpuiserne reflekteres og kastes frem og tilbage mellem objekter i omgivelserne før de når tilbage til sensorerne. Der ligger en udfordring i at finde eller udvikle metoder til at analysere sonarmålinger for at udnytte sensorerne optimalt.
4. **Analyse af muligheder for anvendelse af dopplerradar som nærhedssensor:** Små og billige halvlederbaserede dopplerradarmoduler anvendes i vid udstrækning til registrering af bevægelse og nærhed i tyverialarmer og automatiske døråbnere. At anvende en simpel dopplerradar som bevægelses/nærhedssensor på en AGV eller robot stiller krav til signalbehandlingen om at bortfiltrere signaler der skyldes sensorens egenbevægelse. Det kunne være spændende at få undersøgt om det er muligt at fremstille en praktisk anvendelig bevægelses/nærhedssensor til brug i robotteknologi.
5. **Rekonstruktion og forbedring af SCORBOT styring:** De 6 motordrivere jeg brugte i min styring af Scorboten blev desværre ødelagt ved en fejltilslutning under en demonstration. For at genetablere integrationen mellem Scorbot og Cato, skal der indbygges 6 nye motordrivere i Cato's motor-I/O-box. Ved samme lejlighed bør de tynde ledninger til Scorbotens motorer udskiftes. Når Scorboten igen kan bevæges, vil det være relevant at udbygge eller erstatte min kinematik for den, så der bliver taget højde for Scorbotens slør og elasticitet.

Hvis Cato på et tidspunkt bliver i stand til at køre målbevidst rundt på egen hånd vil det være oplagt at fastmontere Scorboten på Cato og f.eks. lave en mobil udgave af min demonstration med udskænkning af sodavand.

6. Forbedring og udvikling af Cato:

Selv om Cato fungerer udmærket, er der en række aspekter der kan forbedres og videreudvikles.

- Den bør udstyres med dækplader.
- Ledningsføringen bør udføres bedre.
- Elektronikken monteret under den bør indkapsles for at give mekanisk beskyttelse.
- Alle sonarafstandsmålerne skal monteres i metalkasser sammen med de nødvendige multiplexere.
- Liniesensor og bumpsensorer skal forbindes permanent med en A/D omsætter i styredatamaten.
- Styredatamaten bør opgraderes med et af universitetets nye VMEbus CPU-moduler.
- *Power-boardet* bør implementeres som trykt kredsløb med bedre mekaniske og EMC-egenskaber end det nuværende prototypeprint.

Forbedringer af Cato egner sig ikke til at stå alene, men vil kunne integreres i projekter der alligevel bygger videre på Cato.

Kapitel 15

Forhandlerliste

I dette kapitel giver jeg en kortfattet oversigt over nogle af de forhandlere jeg har benyttet mig af i projektet. Listen er ikke udtømmende, men giver information nok til at finde frem til nogle af de mere eksotiske produkter jeg har anvendt.

Farnell electronic components

Naverland 31
2600 Glostrup
Tlf: 44 53 66 44
Fax: 44 53 66 06

Har leveret: Akkumulator, og div. elektroniske komponenter.

Fyneca

V. Arne Madsen
Skolevej 23 Espe
5750 Ringe
Tlf: 62 66 13 95

Har leveret: Genopslebne printbor og prototypeprint.

Lautronic

Lyngbygårdsvej 126B
2800 Lyngby
Tlf: 45 93 51 52

Har leveret: SENO fotoprint, fremkalder, finætsekystal, tinbad, og diverse hjælpemidler til printfremstilling.

Polaroid A/S

Toftebakken 2G
Postbox 9
3460 Birkerød
Tlf: 35 25 82 00
Kontakt: Linius

Har leveret: *sonar-ranging modules*, samt elektrostatiske transducere.

RS / RadioParts A/S

Glentevej 57
Postbox 977
2400 København NV
Tlf: 38 16 99 00
Fax: 38 33 33 10

Har leveret: DC-DC konvertere, og diverse elektroniske komponenter.

Top Tronic

Bakkegårdsvej 405
3050 Humlebæk
Tlf: 49 19 25 00

Forhandler Piezo resistive trykfølere (trykfølsomme modstande), brugt til *bump sensorer*.

Litteratur

Litteraturlisten er opdelt efter litteratortype og artikelgrupperinger. Inden for hver gruppe er henvisningerne sorteret alfabetisk efter titel. I afsnittene *Manualer datablade mv.* og *Diverse artikler* forekommer henvisninger til enkelte værker, i min eller Odense Universitets besiddelse, hvor der ikke forekommer brugbare oplysninger om oprindelsen.

Henvisningerne til data på internet er sidst verificeret i September 1998

Andre projekter ved Odense Universitet

- [1] **Anvendelse af PEP Modular Computer til processovervågning og -styring**
Lars Dahl-Hansen
Institut for Matematik og Datalogi, Odense Universitet
DM07 opgave, 21 februar 1991.
- [2] **Design og implementering af hjælpefunktioner samt operativsystem til skildpadde-AGV**
Peter Aagård Favrholt, Institut for Matematik og Datalogi, Odense Universitet
December 1993.
- [3] **Design, udvikling, og afprøvning af skildpadde AGV**
Anders Stengaard Sørensen
Bachelorprojekt ved Institut for matematik og datalogi, Odense Universitet
1993
- [4] **Kalibrering af AGV**
Tage Søndergaard
Bachelorprojekt Institut for Matematik og Datalogi, Odense Universitet
1994
- [5] **Kortlægningsstrategi for skildpadderobot**
Lene Monrad Petersen, Institut for Matematik og Datalogi, Odense Universitet.
September 1995

Bøger, noter mv.

- [6] **Control System Design and simulation**
Jack Golten & Andy Verwer
McGraw-Hill book company. ISBN 0-07-707412-2
- [7] **Electronic devices, discrete and integrated**
Stephen R. Fleeman
Prentice Hall. ISBN 0-13-338120-X

- [8] **Fundamentals of robotics — Analysis & control**
Robert J. Schilling
ISBN 0-13-344433-3
- [9] **Handbook of transducers**
Harry N. Norton.
Prentice Hall ISBN 0-13-382599-X
- [10] **Introduction to Robotics**
Phillip John McKerrow, University of Wollongong — Australia
Addison-Wesley publishing company.
- [11] **Microcomputer systems: The 8086/8088 family** Architecture, programming, and design.
Yu-Cheng Liu & Glenn A. Gibson
Prentice-Hall International Editions ISBN: 0-13-581851-6
- [12] **Noter til REG1, Januar 1992**
Daniel Durand & Ejvind Jeppesen, IOT
Odense Tekniske boghandel
- [13] **OS-9 insights — an advanced programmers guide**
Peter Dibble
Microware ISBN 0-918035-01-5
- [14] **Power FETS and their applications**
Edwin S. Oxner
Prentice-Hall ISBN 0-13-686923-8
- [15] **Programming the M68000 — second Edition**
Tim King & Brian Knight
The Benjamin / Cummings Publishing Company ISBN 0-8053-5550-2
- [16] **The OS-9 guru — 1 - The facts**
Paul S. Dayan
Galactic Industrial. ISBN 0 9519228 0 7
- [17] **Tykfilm, Hybrid mikroelektronik** 5. udgave rev. 15.01.91
J. Papsøe
Odense Tekniske Boghandel

Manualer, datablade mv.

- [18] **Datablad til POLAROID 6500 series sonar ranging module (PID #615077)**
POLAROID Corporation, Ultrasonic Components Group
POLAROID DK (tlf: 42 81 75 00)
- [19] **Motorolas datablad for MC68230 Paralel interface / Timer**
Motorola
- [20] **Nationals datablad for LM628/LM629 Precision Motion Controller**
National
- [21] **OS-9/68000 Operating system technical manual** Revision G, 1986 (OS-9 V 2.0)
Microware Systems Corporation
1866 N.W. 114th Street
Des Moines, Iowa 50322
- [22] **RTI-600, RTI-602 VMEbus compatible Analog input/output boards Users manual**
Analog devices

- [23] **Scorbot ER-III — Users manual, fifth edition**
Eshed Robotec
- [24] **The VMEbus specification. Revision C.1**
Micrology pbt, Inc. / VITA
PRINTEX Publishing, Inc.
- [25] **Ultrasonic ranging system** Description, operation and use information for conducting tests and experiments with Polaroid's Ultrasonic Ranging System components.
POLAROID Corporation, Ultrasonic Components Group
POLAROID DK (tlf: 42 81 75 00)
- [26] **VBP-9/15/20 VME Backplane User's manual**
PEP Modular Computers.
- [27] **VDIN 16 Channel optoisolated digital input module for VMEbus User's manual**
PEP Modular computers.
- [28] **VDOUT 16 Channel optoisolated digital output module for VMEbus User's manual**
PEP Modular computers.
- [29] **VMPM 68KC2 VME CPU board User's manual**
PEP Modular Computers.

Diverse Artikler

- [30] **Design of Double-Octahedral VGT manipulators**
Ole Grå Jacobsen S.A. Larsen Jacob Steineke, Niels Jul Jacobsen og Anders S. Sørensen — IOT & AMROSE A/S
Neue Maschinenkonzepte mit parallelen Strukturen für Handhabung und Produktion. November 1998
VDI Bericht 1427 **ISBN: 3-18-091427-0**
- [31] **A Bayesian Approach to Real-Time Obstacle Avoidance for a Mobile Robot**
Housheng Hu and Michael Brady, Dpt. of Engineering Science — University of Oxford
Kluwer Academic Publishers, Boston
- [32] **A robustness Analysis of Triangulation-Based Robot Self-Positioning**
Claus B. Madsen, Claus S. Andersen & Jens S. Sørensen
Laboratory of Image Analysis, Aalborg University, Denmark
Proceedings of the 5th International Symposium on Intelligent Robotic Systems, 1997.
- [33] **Absolute Positioning for Indoor Mobile Robots Guidance**
A.B. Martínez, J. Climent, J.M. Asenio, J. Batlle
Automatic Control and Computer Engineering Department.
Universitat Politècnica de Catalunya, Barcelona, Spain.
Proceedings of the 23. International Symposium on Industrial Robots, 1992
- [34] **Apperance Based Processes for Visual Navigation**
Claus Andersen, Stephen D. Jones & James L. Crowley
Project PRIMA-IMAG
Institut National Polytechnique de Grenoble, France.
Proceedings of the 5th International Symposium on Intelligent Robotic Systems, 1997.
- [35] **Estimation of Position and its Uncertainty in the Dead Reckoning System for the Wheeled Mobile Robot**
Yutaka Watanabe & Shin'ichi Yuta
Intelligent Robot Lab. Institute of Information Science and Electronics

University of Tsukuba, Japan

Proceedings of the 20. International Symposium on Industrial Robots, 1989

[36] **Histogramic In-Motion Mapping for Mobile Robot Obstacle Avoidance**

Johann Borenstein and Yoram Koren

IEEE transactions on robotics and automation, vol 7, no. 4, August 1991

[37] **Multi-functional optical proximity sensor by using phase information**

Ryosuke Masuda

Department of Electrical Engineering, Tokai University, Japan

Proceedings of the 1985 International Conference on Advanced Robotics

[38] **Optical and Ultrasonic Systems for Absolute Localization of Mobile Robots**

Claude Pagard & Mustapha Mouaddib

Laboratoire des Systèmes Automatisés, Université de Picardie, Amiens, France.

Proceedings of the 24th International Symposium on Industrial Robots, 1993

[39] **Positioning System for Indoor Mobile Robots Using LED Landmarks and PSD**

Ken Sasaki, Hideo Takahashi, & Masahura Takano

Department of Precision Machinery Engineering, The University of Tokyo, Japan

Proceedings of the 24. International Symposium on Industrial Robots, 1993

[40] **Reliable Multilayers for High Volume Applications**

Heraeus GmbH, Produktbereich Dickfilm

PDF/Dr.Ke/KI 23.09.92

[41] **Sensor System of a Guideless Autonomous Vehicle in a Flexible Manufacturing System**

E. Nakano, N. Koyachi Y. Agari & S. Hirooka

Mechanical Engineering Laboratory, AIST, MITI Japan

Proceedings of the 15. International Symposium on Industrial Robots

[42] **Using Occupancy Grids for Mobile Robot Perception and navigation**

Alberto Elfes

Carnegie Mellon University

RoboCup Papers at ICRA-98 and DARS-98

RoboCup Federation Spring 1998,

www.robocup.org/RoboCup/

[43] **The RoboCup Challenge**

Minoru Asada & Hiroaki Kitano.

**22. International Symposium on Industrial Robots.
1991**

[44] **Development of a Practical Scanning Laser Radar Sensor for Robotic Bin-Picking, Autonomous Guidance, and Other Difficult Image analysis Tasks**

Gary Johnson, Executive Vice President

Perceptron, Farmington Hills, Michigan, USA.

[45] **Development of a Wireless Ranging AGVGS Which Uses Ultrasonic Transponders**

Ooi Kok Chan, Ng KoK Loon, Koh Kok Hwee, Koh Liang Mong & Au Ann San

Nanyang Technological University, Singapore.

[46] **HelpMateTM Delivery Robot Operates Safely Amongst the General Public**

Gordon I. Robertson, Vice President. Engineering Transitions Research Corporation. Danbury, CT

- [47] **Navigation and Control Requirements for Security Robots, A Customers Perspective**
John M. Holland, President, Cybermotion Inc. Roanoke, VA

————— **Proceedings of the 1996 IEEE international conference
on Robotics and automation** —————

- [48] **A Method of Indoor Mobile Robot Navigation Using Acoustic Landmarks**
Joong Hyup Ko, Seung Do Kim, & Myung Jin Chung
Department of Electrical Engineering, Korea Advanced Institute of Science and Technology.
- [49] **An Integrated MMW Radar System for Outdoor Navigation**
Dirk Langer
The Robotics Institute, Carnegie Mellon University, Pittsburgh USA.
- [50] **Dead Reckoning for a Lunar Rover on Uneven Terrain**
Yasutaka Fuke & Eric Krotkov
The Robotics Institute, Carnegie Mellon University, Pittsburgh.
- [51] **Decoupling Odometry and Exteroceptive Perception in Building A Global World Map of a Mobile Robot: The Use of Local Maps**
P. Hébert, S. Betgé-Brezetz & R. Chatila
LAAS-CNRS, Toulouse France.
- [52] **Dynamic Modeling of an Indoor Environment**
Oliver Habert & Alain Pruski
Laboratoire d'Automatique et d'Electronique Industrielles (L.A.E.I.), France
- [53] **Exact Dynamic Map Building for a Mobile Robot using Geometrical Primitives Produced by a 2D Range Finder**
J. Vandorpe, H. Van Brussel & H. Xu
Division PMA, Department of Mechanical Engineering, Faculty of Engineering
Katholieke Iniversiteit Leuven, Belgium
- [54] **Fuzzy-logic Control for Differential-wheel-drive AGVs Using Linear Opto-sensor Arrays**
S.K. Tso, Centre for Intelligent Design, Automation and Manufacturing, City University of Hong Kong.
Y.H. Fung & Y.P. Cheung, Department of Electrical and Electronic Engineering, The University of Hong Kong.
- [55] **Gyrodometry: A New Method for Combining Data from Gyros and Odometry in Mobile Robots**
J. Borenstein & L. Feng, University of Michigan, Advanced Technologies Lab.
- [56] **Incremental Construction of a Landmark-based and Topological Model of Indoor Environments by a mobile robot**
Hanna Bulata & Michel Devy
L.A.A.S. - C.N.R.S, France.
- [57] **Mobile Robot Localization in Dynamic Environments Using Dead Reckoning and Evidence Grids**
Brian Yamauchi
Center for the study of Language and Information, Stanford University
- [58] **Non-Uniform Dead-Reckoning Position Estimate Updates**
Don Krantz, MTS Systems Corporation &
Maria Gini, Dept. of Computer Science, University of Minnesota.

[59] **Sonar Sensing Strategies**

Thomas C. Henderson, Beat Bruderlin, Mohammed Dekhil, Larry Schenkat & Larry Veigel
Department of Computer Science, University of Utah.

**Proceedings of the 1995 IEEE/RJS international
conference on Intelligent robots and systems**

[60] **A Comparison of two Methods for Fusing Information from a Linear Array of sonar Sensors for Obstacle Localization**

Orhan Arikan & Billur Barshan
Department of electrical Engineering, Bilkent University, Ankara, Turkey.

[61] **A Low-Cost, Composite Sensor Array Combining Ultrasonic and Infrared Proximity Sensors**

Angelo M. Sabatini, Vincenzo Genovese, Eugenio Guglielmelli, Anselmo Mantuano, Giovannini Ratti, Paolo Dario
Advanced Robotics Technology & Systems (ARTS Lab), Scuola Superiore Sant' Anna, Pisa Italy

[62] **Correction of systematic Odometry Errors in mobile robots**

Johann Borenstein and Liqiang Feng, The University of Michigan

[63] **Multi Level Navigation Using Active Localization Systems**

F. Giuffrida, C. Massucco, P. Morasso, G. Vercelli, R. Zaccaria
DIST, University of Genova.

[64] **Position Estimation for Mobile Robot Using Sensor Fusion**

Daehee Kang, Hideki Hashimoto, & Fumio Harashima
Institute of Industrial Science, University of Tokyo.

[65] **Sensor-Based Homing Using Omnidirectional Range and Intensity Sensing System for Indoor Mobile Robot Navigation**

Seok Won Bang, Wonpil Yu, & Myung Jin Chung
Dept. of Electrical Engineering, Korea Advanced Institute of Science and Technology.

[66] **Sonar Feature Based Exploration**

R. Bauer & W.D. Rencken
Siemens AG, Corporate Research and Development, München, Germany

[67] **Steering of a Mobile Robot Using Insect Antennae**

Yoshiko Kuwana, Isao Shimoyama, and Hirofumi Miura
Department of Mechano-Informatics, The University of Tokyo, Japan.

[68] **The Ultrasonic Range Finder for Outdoor Mobile Robots**

Tsutomu Tanzawa, Noriaki Kiyohiro, Shinji Kotani, Hideo Mori
Department of Electrical Engineering and Computer Science, Yamanashi University, Japan.

**WWW henvisninger (med forbehold for ændrede og
nedlagte adresser)**

[69] **<http://almond.srv.cs.cmu.edu/afs/cs.cmu.edu/copetas/www/public/mobot/index.html>**

CMU Mobot race 1998.

[70] **<http://diwww.epfl.ch/lami/robots/K-family/vaccum.html>**

The Koala Mobile Robot as Vacuum Cleaner

[71] **<http://international.husqvarna.com/products/robotic/solarmower.html>**

Husqvarna SolarMower
Autonom græsslåmaskine

- [72] <http://lasers.llnl.gov/lasers/idp/mir/overview.html>
Lawrence Livermore National Laboratories — Micropower Impulse Radar
- [73] <http://piglet.cs.umass.edu:4321/robotics.html>
Robotics Internet Resources Page
- [74] <http://www.avdil.gtri.gatech.edu/AUVS/IARCLaunchPoint.html>
Association for Unmanned Vehicle Systems, International's Aerial Robotics Competition.
- [75] <http://www.ai.mit.edu/projects/mobile-robots.html>
MIT AI Lab: The mobot group.
- [76] <http://www.amrose.dk>
AMROSE A/S
- [77] <http://www.danelec.dk>
Danelec A/S
Den danske forhandler af OS-9, og af diverse industricomputre bl.a. PEP og Motorola. Siden rummer primært links til VME og OS-9 relevante firmaer og organisationer.
- [78] <http://www.electrolux.se/corporate/pressnotes/RobotPressroom.htm>
Robot Vacuum Cleaner Releases
- [79] <http://www.iau.dtu.dk/robocup/robocup.html>
DTU RoboCup — DTU's årlige konkurrence for selvstyrende køretøjer.
- [80] <http://www.llnl.gov/IPandC/op96/10/10o-mic.htm>
Lawrence Livermore National Laboratories — Micropower Impulse Radar
- [81] <http://www.ntplx.net/helpmate>
Serviceroboten HelpMate.
- [82] http://www.ntplx.net/helpmate/RC_components/RC_lightranger.html
LightRanger
Time of flight laser afstandsmåler fra HelpMate
- [83] http://www.ntplx.net/helpmate/RC_components/RC_sonaranger.html
SonaRanger
Sonar ranger fra HelpMate
- [84] <http://www.helix.com/robot.htm>
Helix Systems, Inc. Robotics System Page
Industrielle AGV'er
- [85] <http://www.mentoragvs.com/>
Mentor AGV's Industrielle AGV'er
- [86] <http://www.nosc.mil/robots/land/robart/spie96.html>
SPAWAR Systems Center San Diego.
A Third Generation Security Robot
H.R. Everett, Douglas W. Gage
- [87] <http://www.translogic-corp.com/AGV.htm>
Translogic Corp
Industrielle AGV'er
- [88] <http://www-robotics.usc.edu/robots.html>
Robots at the USC Robotics Research Lab
- [89] <http://www.robots.com>
Nomadic Technologies
Fremstiller flere forskellige typer AGV'er, beregnet som platforme til servicerobotter.

- [90] <http://www.sickoptik.com/laser.htm>
Sick Optic
LIDAR laser scanner

Diverse

- [91] **Ny medarbejder på batterier**
(Om afprøvning af en HelpMate servicerobot på Odense Universitetshospital)
Fyens Stiftstidende, onsdag 17. januar 1996
- [92] **Nørdernes robotkamp**
Politikken, torsdag 8. maj 1997.
- [93] **Vi kom, vi så, vi sejrede**
(De datateknologistuderendes udlægning af DTU Robocup 1997)
FACTS, Juni 1997 — FACTS er medlemsbladet for Foreningen Af Datateknologistuderede i Odense (FADSO)

Del II

Teknisk dokumentation

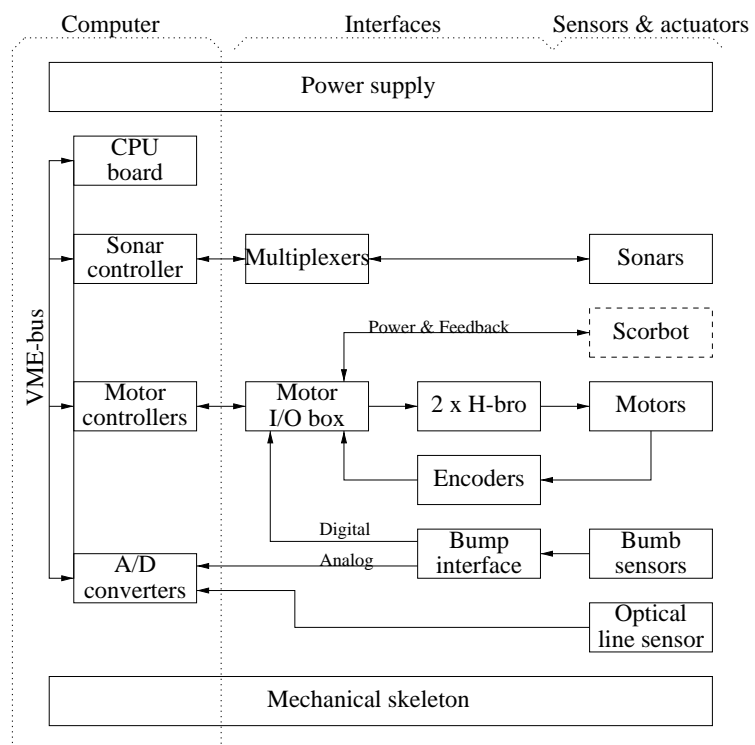
Kapitel 16

Dokumentation af Cato

Heri dokumenteres:

- Det mekaniske skelet
- Strømforsyning
- Bump sensorer og interface.
- Motorer og enkodere
- Motor I/O box
- H-broer
- Computer

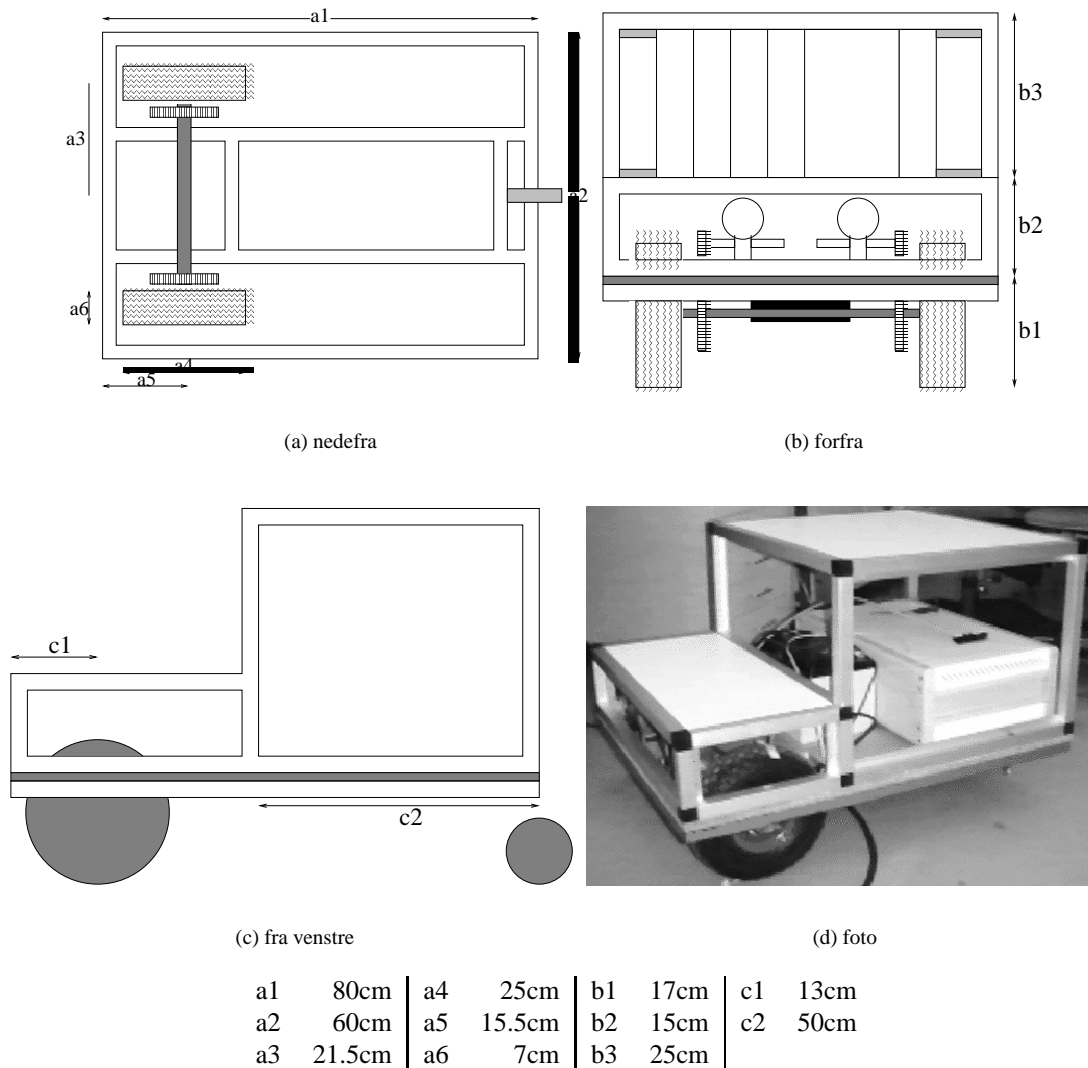
Resten af komponenterne er enten dokumenteret i andre kapitler (sonar controller, sonar-multiplexere, motorcontroller, og liniesensor), eller i eksterne manualer (A/D converter, CPU-kort, VME-bus, og Scorbot)



Figur 16.1: Blokdiagram over Cato's hardware

16.1 Mekanik

Figur 16.2 viser en skitse over cato's mekaniske opbygning.



Figur 16.2: Cato's form

16.1.1 Skelet

Cato er bygget over en rektangulær stålramme. Foran er monteret to ikke drejelige gummihjul, med pneumatiske ringe. Bagerst er monteret et frit drejeligt nylonhjul. På hver hjulaksel er monteret et tandhjul, der kan drives af en tandrem.

Oven på stålrammen er monteret et dæk, i form af en 15mm krydsfinerplade med udskæringer til hjulene. Pladen holdes fast på stålrammen, dels af de bolte der holder motorerne, dels af de bolte der holder overbygningen.

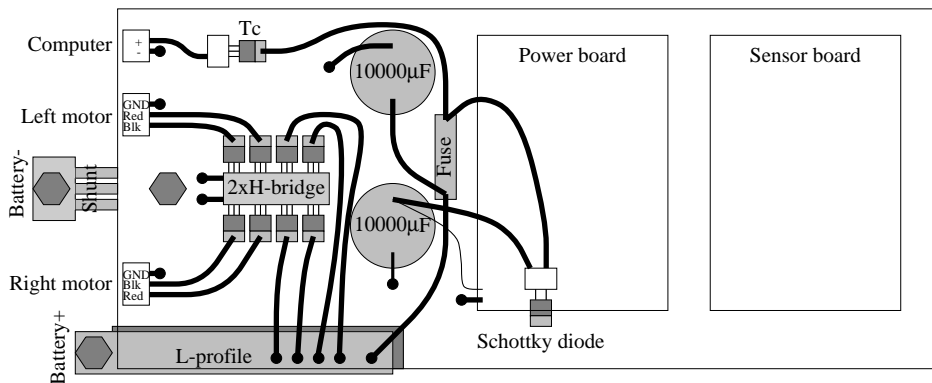
De to motorer der driver Cato, er monteret oven på dækket, forrest på Cato.

Overbygningen er opbygget af 25mm kvadratiske aluminiums profiler (Porsa system). Overbygningen er konstrueret i to niveauer, med et lavt fordæk, og et højt agterdæk. Begge dæk på overbygningen udgøres af 20mm melaninbelagt spånplade. Der er flanger til isætning af dækplader i alle åbninger i overbygningen, undtaget bagtil. I trinnet mellem for- og agter-dæk, er isat messingprofiler med M3 gevindhuller, for

montage af 200mm lange forplader. Forpladerne skal have en modulbredde på 55 m.m. Skruehullerne i forpladerne placeres i hjørnerne, 5mm fra hver kant.

16.1.2 Elektronik monteringsplade

På undersiden af den finerplade der udgør Cato's *dæk* er der monteret en aluminiumsplade, med en godstykkelse på 4m.m. Pladen tjener som montageplatform, stelplan og køleplade for strømforsyning, motor-drivere, og andet elektronik der med fordel kan placeres under Cato.



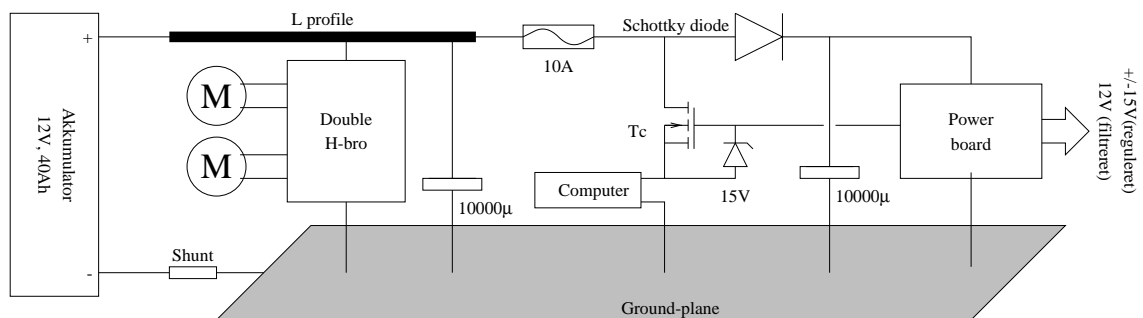
Figur 16.3: Skitse af monteringsplade under Cato

Ud over diverse diskrete komponenter, er der vha 25mm afstandsstykker, monteret to 10cm × 16cm hulprint på monteringspladen. Printene kaldes hhv. *power board* og *sensor board*.

På pladens forkant sidder en 2-polet og to 3-polede stikdele, af adskillige klemrækker. Den 2-polede er til tilslutning af computerens strømforsyning, mens de 3-polede er for tilslutning af motorer.

16.2 Strømforsyning

Figur 16.4 viser diagrammet for den overordnede del af Cato's strømforsyning.



Figur 16.4: Strømforsyning

Stelplanet udgøres af montagepladen under Cato's dæk. Shuntmodstanden på $3m\Omega$ er skruet direkte på stelplanet. Langs stelplanets ene side er monteret en 2mm L-profil af aluminium, der fungerer som påskrukningsterminal for +12V forsyningen. Shuntmodstanden og L-profilen er forbundet til akkumulatoren vha. $10mm^2$ kabler. Kablerne er begge sorte, men +kablet er tydeligt markeret med rød tape i enderne.

Transistorerne i den dobbelte H-bro er forbundet til forsyningen før sikringen.

MOS-FET transistoren T_c er monteret direkte på stelplanet, med passende elektrisk isolation. 15V zen-dioden er monteret sammen med transistoren. De to $10000\mu F$ kondensatorer er begge skruet fast på stelplanet. Shottky dioden i et isoleret TO-220 hus er også skruet fast på stelplanet, under *power boardet*.

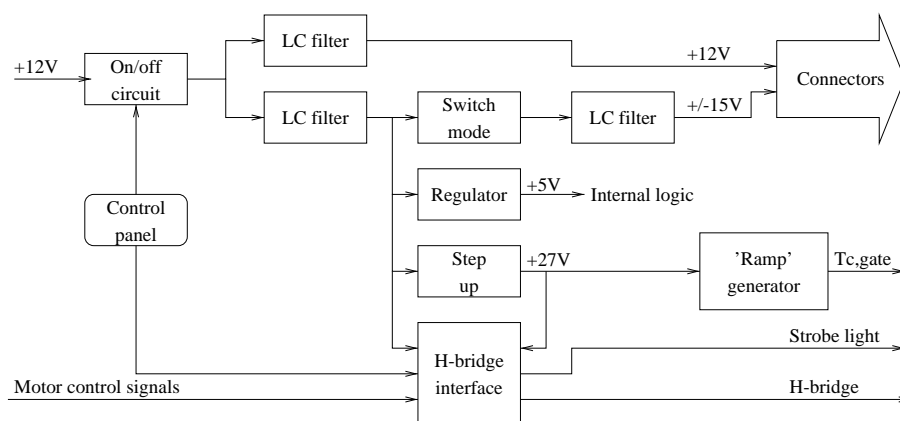
16.2.1 Power boardet

Power boardet er opbygget på et $16 \times 10\text{cm}$ prototype print (Fyneca: TY 209). Printet varetager følgende funktioner:

- Tænd/sluk funktion
- Filtrering af spændingsforsyning
- Generering af forsyningsspændinger til analog elektronik.
- Generering af forsyningsspændinger til styring af MOS-FET transistore.
- Interface mellem motor controller og transistorerne i de to H-broer til drivmotorerne.

Figur 16.5 viser et blokdiagram over power boardet, figur 16.7 viser kredsløbsdiagrammet, figur 16.6 viser placeringen af stik, og større komponenter på printet, og tabel 16.2 gengiver komponentlisten for boardet. Interfacet til H-broen er beskrevet separat i afsnit 16.6.1

Powerboardet styres af et lille kontrolpanel med 4 knapper, der forbindes vha. et kabel og et 10-polet molexstik. Kontrolpanelet har funktionerne: *On*, *Off*, *Panic* og *Don't panic*. *On* og *Off* tænder hhv. slukker for strømforsyningen til al elektronikken inklusiv datamaten, mens *Panic* og *Don't panic* de- hhv. reaktiverer energitilførslen til cato's to motorer.



Figur 16.5: Blokdiagram af power-board

Indgangsspænding 11V – 15V

Filtreret udgang (12V) Samme spænding som på indgangen. Max 3A forbrug.

+15V ud Reguleret udgang, max 200mA

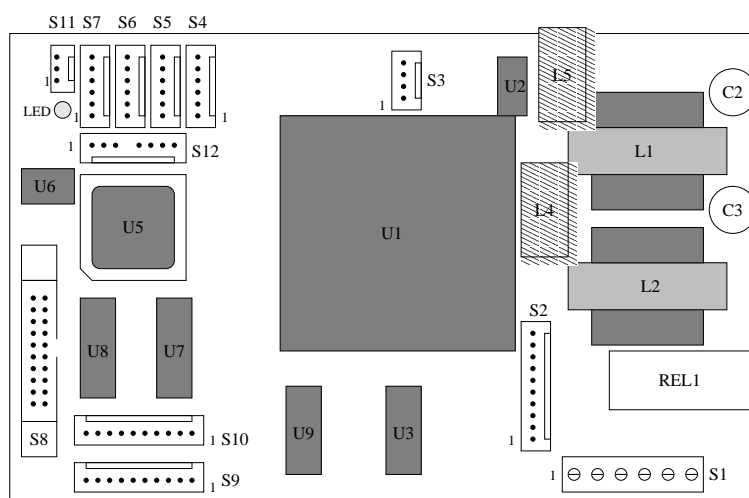
-15V ud Reguleret udgang, max 200mA

Gate-C Højimpedant rampegenererende styresignal til MOS-FET transistoren der tænder computeren.

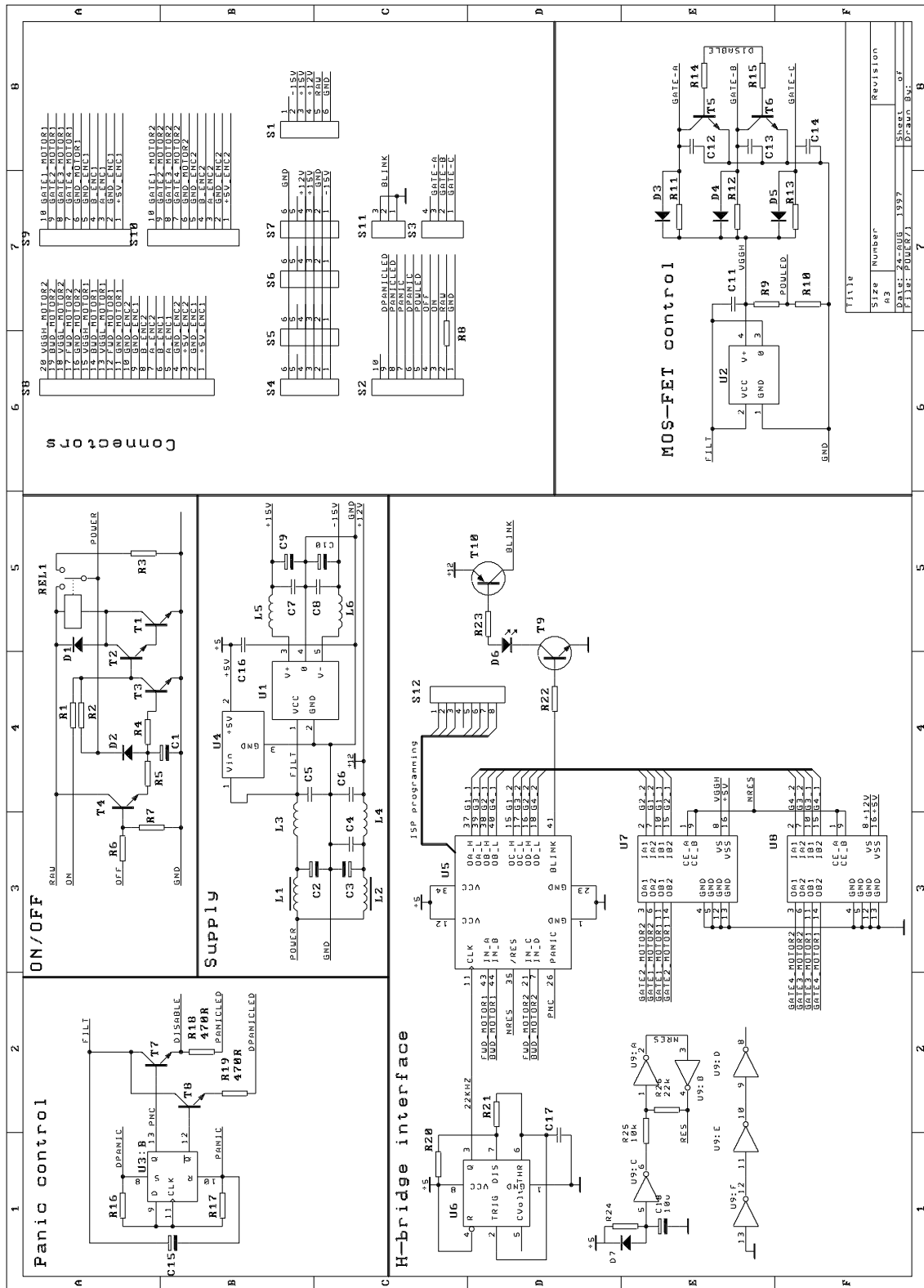
$$V_{high} \geq 25\text{V}, R_{out} = 100\text{k}\Omega, C_{out} = 100\text{nF}, t_r = 10\text{ms}.$$

Pin nr.	S1 Strømforsyning ind/ud	S2 Kontrolpanel	S4-S7 Strømforsyning ud	S3 MOS-FET kontrol
1	NC	GND	-15V	Gate-A (bruges ikke)
2	-15V ud	Signal power (12V)	GND	Gate-B (bruges ikke)
3	+15V ud	ON	+15V	Gate-C (styrer Tc)
4	+12V ud (filtreret)	OFF	+12V	
5	+12V ind	Power LED	NC	
6	GND	Don't panic	GND	
7		Panic		
8		Panic LED		
9		Don't panic LED		
10		NC		

Tabel 16.1: Benforbindelser til strømforsyningsdelen



Figur 16.6: Skitse over komponentplacering på power-board



Figur 16.7: Diagram over power-board

R1	10k Ω	$\pm 5\%$	1/4W	S1	6way	Screw terminal	D1	1N4001	1A diode
R2	47k Ω	$\pm 5\%$	1/4W	S2	10way	MOLEX	D2	1N4148	small signal
R3	10 Ω	$\pm 5\%$	1W	S3	4way	MOLEX	D3	1N4148	small signal
R4	100k Ω	$\pm 5\%$	1/4W	S4	6way	MOLEX	D4	1N4148	small signal
R5	120 Ω	$\pm 5\%$	1/4W	S5	6way	MOLEX	D5	1N4148	small signal
R6	1.8k Ω	$\pm 5\%$	1/4W	S6	6way	MOLEX	D6	3mm LED	yellow, 20mA
R7	10k Ω	$\pm 5\%$	1/4W	S7	6way	MOLEX	D7	1N4001	1A diode
R8	1k Ω	$\pm 5\%$	1/4W	S8	20way	IDC			
R9	270 Ω	$\pm 5\%$	1/4W	S9	10way	MOLEX	T1	BD135	Power NPN
R10	1.5k Ω	$\pm 5\%$	1/4W	S10	10way	MOLEX	T2	BC547B	NPN
R11	100k Ω	$\pm 5\%$	1/4W	S11	3way	MOLEX	T3	BC547B	NPN
R12	100k Ω	$\pm 5\%$	1/4W	S12	8way	MOLEX	T4	BC547B	NPN
R13	100k Ω	$\pm 5\%$	1/4W				T5	BC547B	NPN
R14	10k Ω	$\pm 5\%$	1/4W	C1	4.7 μ F	Tantal	T6	BC547B	NPN
R15	10k Ω	$\pm 5\%$	1/4W	C2	1000 μ	Elektrolyt	T7	BC547B	NPN
R16	100k Ω	$\pm 5\%$	1/4W	C3	1000 μ	Elektrolyt	T8	BC547B	NPN
R17	100k Ω	$\pm 5\%$	1/4W	C4	100nF	Sibatit	T9	BC547B	NPN
R18	470 Ω	$\pm 5\%$	1/4W	C5	100nF	Sibatit	T10	BD438	Power PNP
R19	470 Ω	$\pm 5\%$	1/4W	C6	100nF	Sibatit			
R20	2.2k Ω	$\pm 5\%$	1/4W	C7	100nF	Sibatit	U1	NMXD1215	5W DC-DC konv
R21	2.2k Ω	$\pm 5\%$	1/4W	C8	100nF	Sibatit	U2	NME1215	1W DC-DC
R22	2.2k Ω	$\pm 5\%$	1/4W	C9	10 μ F	Elektrolyt	U3	4013	2 D flipflops
R23	470 Ω	$\pm 5\%$	1/4W	C10	10 μ F	Elektrolyt	U4	L7805CV	Volt. reg.
R24	471 Ω	$\pm 5\%$	1/4W	C11	100nF	Sibatit	U5	ispLSI-2032	Prog. logic
R25	10k Ω	$\pm 5\%$	1/4W	C12	100nF	Sibatit	U6	NE555	Timer
R26	22k Ω	$\pm 5\%$	1/4W	C13	100nF	Sibatit	U7	L293D	quad drivers
				C14	100nF	Sibatit	U8	L293D	quad drivers
L1	250mH	Jernkerne	5A	C15	1 μ F	Tantal	U9	74HC14	hex inverters
L2	250mH	Jernkerne	5A	C16	100nF	Sibatit			
L3	50mH	Ferritkerne	5A	C17	10nF	Polyester			
L4	50mH	Ferritkerne	5A	C18	10 μ F	Tantal			
L5	0.47mH	Ferritkerne	500mA						
L6	0.47mH	Ferritkerne	500mA						

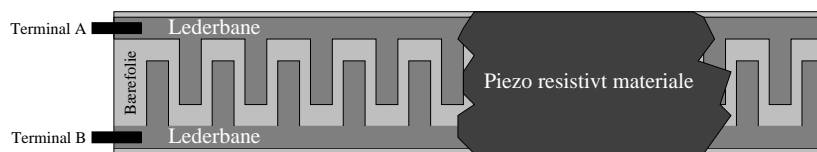
Tabel 16.2: komponentliste til power-board

16.3 Bump sensorer og interface

For at kunne detektere sammenstød med omgivelserne, benytter Cato en række trykfølsomme modstande, der er monteret på yderkanterne af understellet, under en beskyttende gummiliste.

16.3.1 Sensorelementerne

Figur 16.8 viser en skitse af en trykfølsom modstand. Modstanden består af en isolerende bærefolie (mylar), hvorpå der er lagt to lederbaner parallelt i et fiskebensmønster. De to lederbaner har ikke kontakt med hinanden. Hele arrangementet er lamineret sammen med et lag piezo resistivt materiale, hvis modstandsværdi falder når der trykkes på det. Hver enkelt sensor er 500 mm lang, 12mm bred, og ca. 1mm tyk.



Figur 16.8: Lagdelt skitse af trykfølsom modstand

16.3.2 Placering og tilkobling

Sensorerne er monteret som vist i figur 16.9. De trykfølsomme modstande er selvklæbende og monteret direkte på metalrammen. Gummilisterne er limet uden på modstandene vha. kontaktlim.

Fra hvert sensorelements to terminaler er forbundet til et fælles 20-polet stik, der kobles til *sensor boardet* på monteringspladen under Cato. Terminalerne er forbundet til stikket vha. 0.25mm² monteringsledning, der holdes fast på Cato's stålskelet af TESA stoftape.

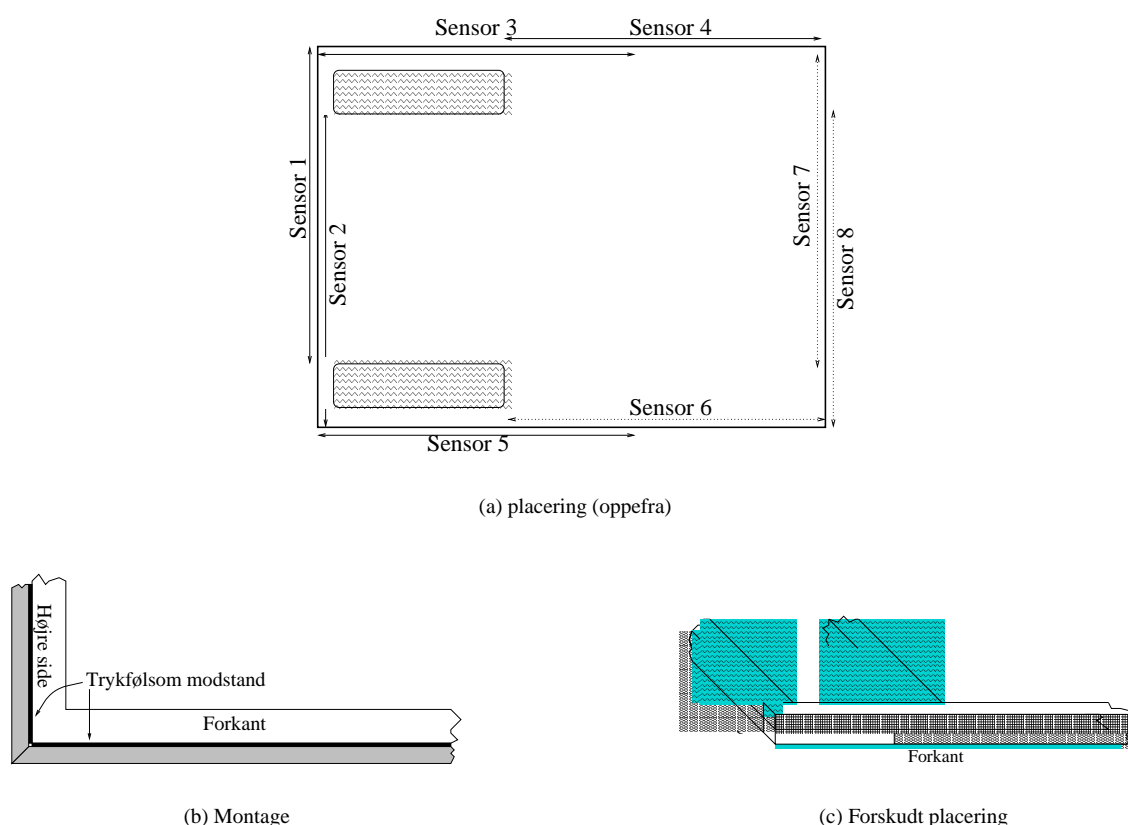
Pin nr.		Pin nr.	
1	sensor 1, terminal A	11	sensor 1, terminal B
2	sensor 2, terminal A	12	sensor 2, terminal B
3	sensor 3, terminal A	13	sensor 3, terminal B
4	sensor 4, terminal A	14	sensor 4, terminal B
5	sensor 5, terminal A	15	sensor 5, terminal B
6	reserveret til sensor 6, terminal A	16	reserveret til sensor 6, terminal B
7	reserveret til sensor 7, terminal A	17	reserveret til sensor 7, terminal B
8	reserveret til sensor 8, terminal A	18	reserveret til sensor 8, terminal B
9		19	
10		20	Blokeret (polarisering af stik)

Tabel 16.3: Stikforbindelser til bump sensorer

16.3.3 Sensor board

Sensorsignalerne bearbejdes af *sensor-boardet*, der sidder på metalpladen under Cato, ved siden af *power-boardet*. Sensorboardet er opbygget på et 10×16cm prototypeprint.

Figur 16.10 viser diagrammet over boardet. Hver af de trykfølsomme modstande indgår som den høje del af en spændingsdel, hvorved dens modstandsværdi omsættes til en spænding. Hver af de 8 spændinger

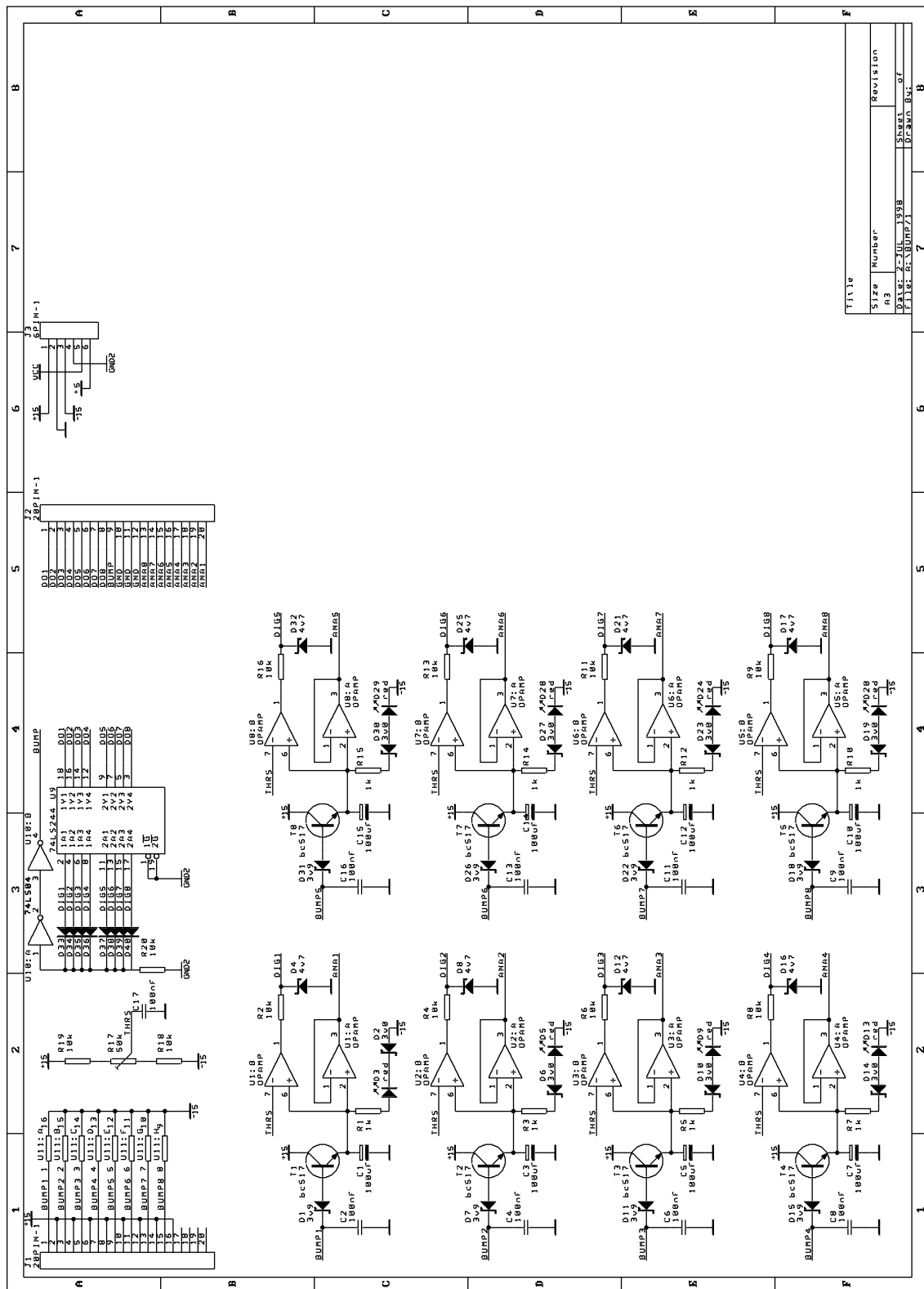


Figur 16.9: Placering af bump sensorer

bearbejdes af et analogt kredsløb der fungerer som AM detektor (med en tidskonstant på 0.1s), begrænser spændingen til -10V...10V, foretager en impedanstilpasning, og driver en lysdiode der indikerer trykket. Udgangsspændingen fra hvert kredsløb er en voksende funktion af trykket på den koresponderende modstand. Spændingen fra hver analoge udgang sammenlignes med en fælles tærskelværdi af en komparator der aktiverer en digital udgang hvis spændingen overstiger tærskelværdien. Et diode-OR kredsløb aktiverer en fælles digital udgang hvis spændingen fra et enkelt kredsløb overstiger tærskelværdien.¹

Sensorene og boardets udgange er koblet til hvert sit 20-polede IDC stik. For at kende forskel, og undgå at vende sensorstikket forkert, mangler indgangsstikkets ben 20, og sensorstikkets ben 20 er blokeret. Sensorboardet strømforsynes via et 6-polet MOLEX stik, hvis benforbindelser fremgår af figur 16.10

¹P.t. er komponenterne til den digitale del ikke monteret endnu



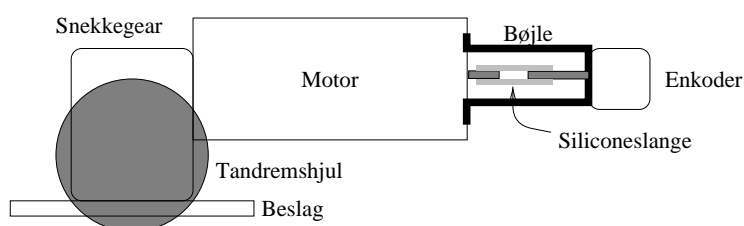
Figur 16.10: Diagram over sensor-board

16.4 Motorer og enkodere

Cato drives af to identiske 12V 80W standard DC-motorer, der vha. et snækkegear og en tandrem trækker hvert sit hjul. Motorerne er normeret til 4000 omdrejninger pr. minut, ved 12V uden belastning. Snækkegearet har en udveksling på 1 til 40. Tandremshjulene på motorerne har en omkreds på ca. 19cm, mens tandremshjulet på hjulaksen har en omkreds på ca. 35cm.

For at få feedback fra motorerne er der boret et 3.5mm hul i hver motoraksel, hvori der er limet en 3mm studs. Studsen er vha. et kort stykke siliconeslange forbundet til akslen på en optisk inkrementalenkoder monteret på en bøjle bag motoren.

De to polledninger på motoren, er sammen med jordledningen skruet i stikdelen af en 3-polet adskillelig klemrække. De tilhørende to fatningsdele er monteret i forreste højre og venstre hjørner af monteringspladen under Cato's dæk. polledningerne forbindes derved til H-broerne, og jordledningerne til monteringspladen/stelplanet.

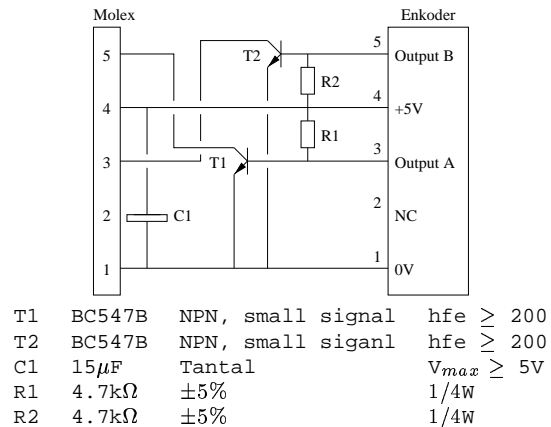


Figur 16.11: Skitse af motor

16.4.1 Enkodere

Der anvendes en optisk enkoder fra Bourns (Farnell best. nr. 107-102). Enkoderen afgiver 128 pulser pr. omdrejning.

Som vist i figur: 16.12 er der påmonteret et bufferkredsløb på enkoderen, for at beskytte dens udgange mod fejltilslutning etc. Bufferen er inverterende, med *open collector* udgange. For at kompensere for den 180° fasedrejning inverteringen giver anledning til, byttes kanal A og B. Benforbindelserne og signalkonventionen for enkoderen er dermed upåvirket af bufferen. Enkoderne er forbundet til S9 og S10 på *power boardet*, vha et 8 leder IDC kabel, med fire ledere til hver enkoder.

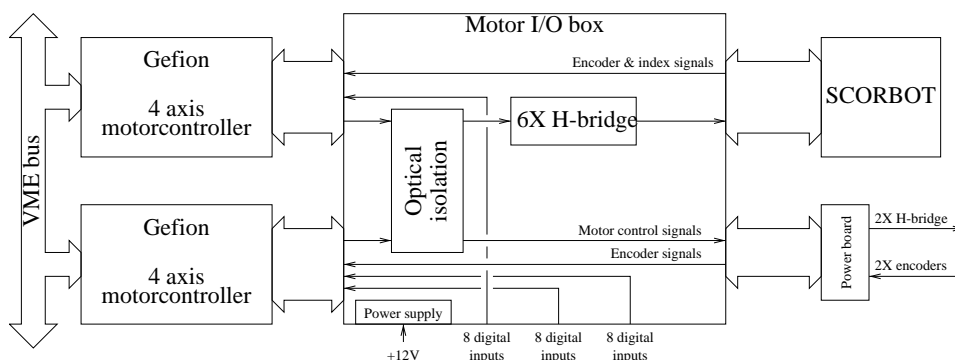


Figur 16.12: Diagram over enkoderbuffer

16.5 Motor I/O box

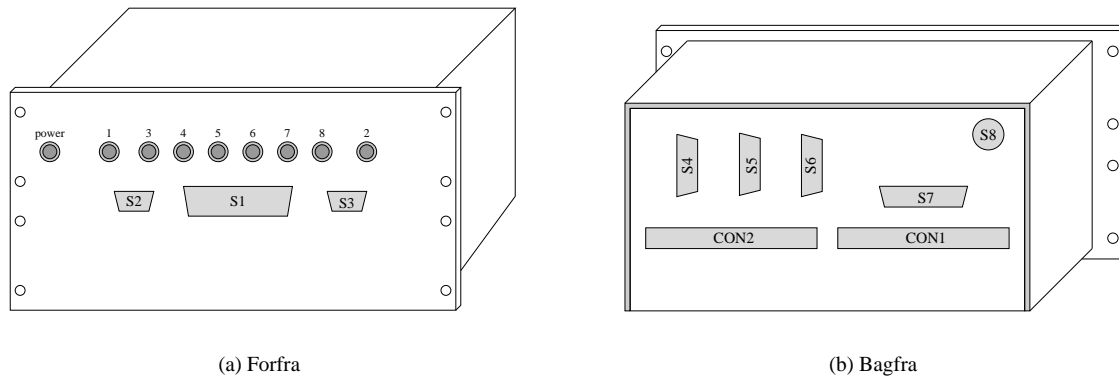
Motor I/O boxen, danner grænseflade mellem en eller to Gefion motorcontrollere, Cato, og en Scorbott robotarm. I/O boxen har 4 funktioner:

- Galvanisk isolering af Gefions udgange.
- H broer til Scorbottens 6 motorer.
- Forbindelser mellem Gefions enkoderindgange, og Cato's hhv. Scorbottens enkodere.
- Forbindelser og formodstande til Gefions optisk isolerede digitale indgange, og Scorbottens *home switches*, og digitale signaler fra Cato.



Figur 16.13: Blokdiagram over Motor I/O box

Motor I/O boxen er opbygget i en selvstændig kasse, med en forplade der passer i Cato's forplade modul system. Hvis man ønsker at bruge Scorbotten, uden Cato, kan kassen afmonteres og bruges på et bord.



Figur 16.14: Motor I/O box

De to Gefion motorcontrollere forbindes til stikkene CON1 hhv. CON2 bagpå motor I/O boxen. De fire akser styret af Gefion motorcontrolleren, tilkoblet CON1, nummeres 1 til 4, mens de fire akser fra den anden controller nummeres 5 til 8. Motor I/O boxen er forsynet med 8 tofarvede (grøn/røde) lysdioder der indikerer retningen og størrelsen af energitilførslen til hver enkelt motor.

Axe 1 og 2 bruges til Cato's venstre hhv. højre drivmotor, mens 2 til 8 bruges til Scorbotens base, skulder, albue, håndled-a, håndled-b, og griber; i den nævnte rækkefølge.

De 16 optisk isolerede digitale indgange på de to Gefion motorcontrollere, benævnes PA0... PA7, PB0... PB7 for controlleren tilsluttet CON1, og PC0... PC7, PD0... PD7, for controlleren tilsluttet CON2. De 3×8 signaler: PAX, PBX, og PCX, er ført ud på separate stik: S4, S5 og S6, på motor I/O boxen, mens PD0... PD4 er koblet til Scorbotens base, skulder, albue, håndled-a og håndled-b *homing switches*. PD5... PD7 bruges ikke.

Hvis Scorboten og PC0... PC7 ikke bruges, er den kun nødvendigt med en enkelt Gefion motorcontroller, koblet til CON1, for at bruge Cato.

16.5.1 Stikforbindelser

Motor I/O boxen har 3 stik foran, 7 stik bagpå:

S1 Forbindelse til Scorbot.

S2 Bruges ikke.

S3 Bruges ikke.

S4 PA0... PA7 digitale indgange.

S5 PB0... PB7 digitale indgange.

S6 PC0... PC7 digitale indgange.

S7 Motorstyresignaler ud, og enkodersignaler ind fra Cato's drivmotorer.

S8 Filtret og ufiltreret 12V ind.

CON1 Forbindelse til Gefion motorcontroller.

CON2 Forbindelse til Gefion motorcontroller.

OBS! Pga. et uheld der har ødelagt de Rungner motordrivere, der skulle have været indbygget i motor-I/O-boxen, er den endnu ikke korrekt bestykket (9. december 1998) og nogle stik er forbundet midlertidigt. Der henvises til [23] for oplysninger om benforbindelser i Scorbotens stik. Mht. benforbindelserne til S4, S5, S6 og S8 anbefales det at undersøge den konkrete konfiguration af stik og benforbindelser inden disse stik anvendes.

S1					
pin	Signal	pin	Signal	pin	Signal
1		18		35	
2		19		36	
3		20		37	
4		21		38	
5		22		39	
6		23		40	
7		24		41	
8		25		42	
9		26		43	
10		27		44	
11		28		45	
12		29		46	
13		30		47	
14		31		48	
15		32		49	
16		33		50	
17		34			

(a) Stik til Scorbot

S4			
pin	Signal	pin	Signal
1	PA0	9	COMA03
2	PA1	10	GND
3	PA2	11	+5V
4	PA3	12	
5	PA4	13	+5V
6	PA5	14	GND
7	PA6	15	COMA47
8	PA7		

S5			
pin	Signal	pin	Signal
1	PB0	9	COMB03
2	PB1	10	GND
3	PB2	11	+5V
4	PB3	12	
5	PB4	13	+5V
6	PB5	14	GND
7	PB6	15	COMB47
8	PB7		

S6			
pin	Signal	pin	Signal
1	PC0	9	COMC03
2	PC1	10	GND
3	PC2	11	+5V
4	PC3	12	
5	PC4	13	+5V
6	PC5	14	GND
7	PC6	15	COMC47
8	PC7		

(b) Optisk isolerede digitale indgange

S7			
pin	Signal	pin	Signal
1		14	Enkoder2 +5V
2	Enkoder1 +5V	15	Enkoder2 GND
3	Enkoder1 GND	16	Enkoder2 A
4	Enkoder1 A	17	Enkoder2 B
5	Enkoder1 B	18	Enkoder2 GND
6	Enkoder1 GND	19	Motor2 sgnl-GND
7	Motor1 sgnl-GND	20	Motor2 FWD
8	Motor1 FWD	21	Motor2 BWD
9	Motor1 BWD	22	
10		23	Motor2 sgnl-GND
11	Motor1 sgnl-GND	24	Motor2 Vggl
12	Motor1 Vggl	25	Motor2 Vggh
13	Motor1 Vggh		

S8	
pin	tilslutning
1	
2	
3	
4	

(c) Forbindelser til motordrivere og enkodere for motor 1 og 2

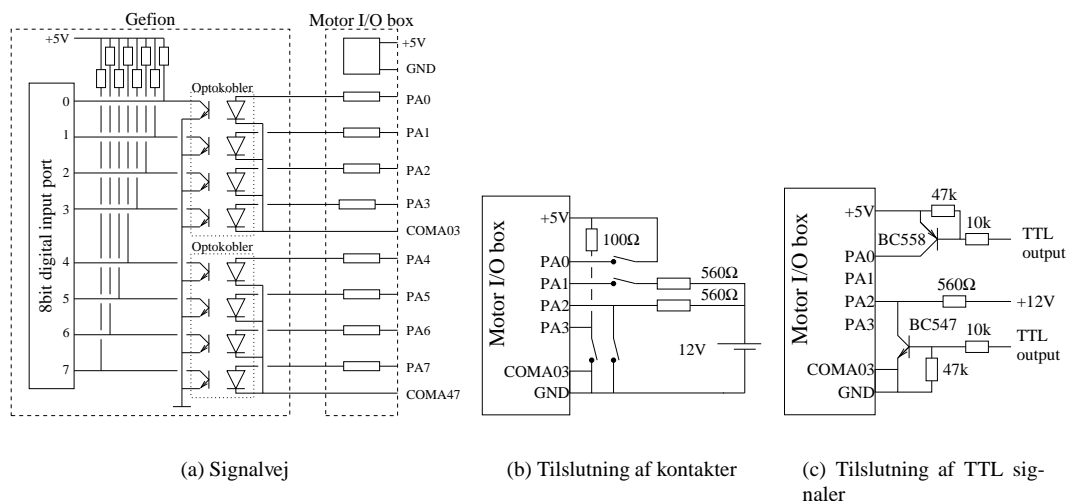
(d) Tilslutning af strømforsyning^b

Figur 16.15: Stikforbindelser for motor I/O box

16.5.2 Tilslutning til digitale indgange

De 16 digitale indgange på hver Gefion motorcontroller, er optisk isoleret, som vist på figur 16.16(a). De 16 indgange aflæses som to separate *bytes*, hvor hver indgang repræsenterer en *bit*. Hvis der løber en strøm på 10...20mA i optokoblerens lysdiode, aflæses den *bit* der repræsenterer den pågældende indgang som 0, hvis der ikke løber en strøm aflæses *bit*en som 1.

De digitale indgange er grupperet som 4 grupper a 4 indgange. Hver gruppe har fælles katode.



Figur 16.16: Tilslutning til digitale indgange

Motor I/O boxen's eneste funktion mht. de digitale indgange, er at sætte en modstand i serie med hver anode på optokoblerne, så optokoblerne kan trækkes direkte af en 5V forsyning. Motor I/O boxen stiller også en 5V forsyning til rådighed, med pinout i hvert af de tre tilkoblingsstik. **Bemærk at det samlede strømforbrug fra Motor I/O boxens +5V forsyning ikke må overstige 500mA!**

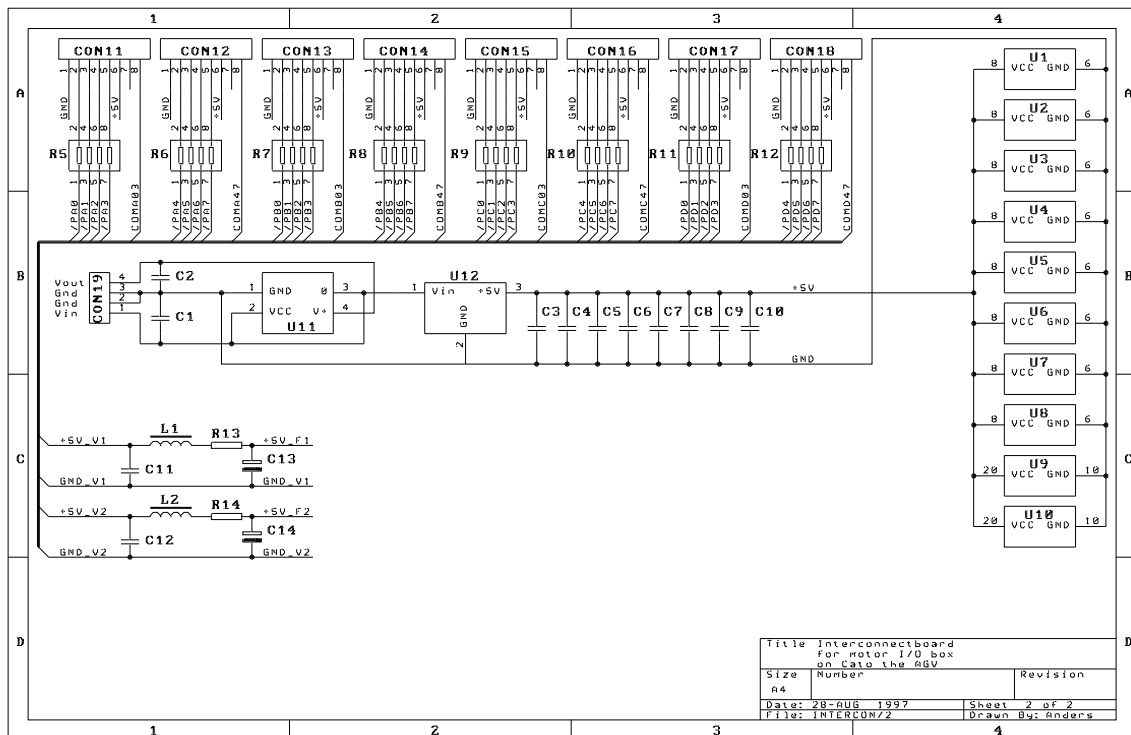
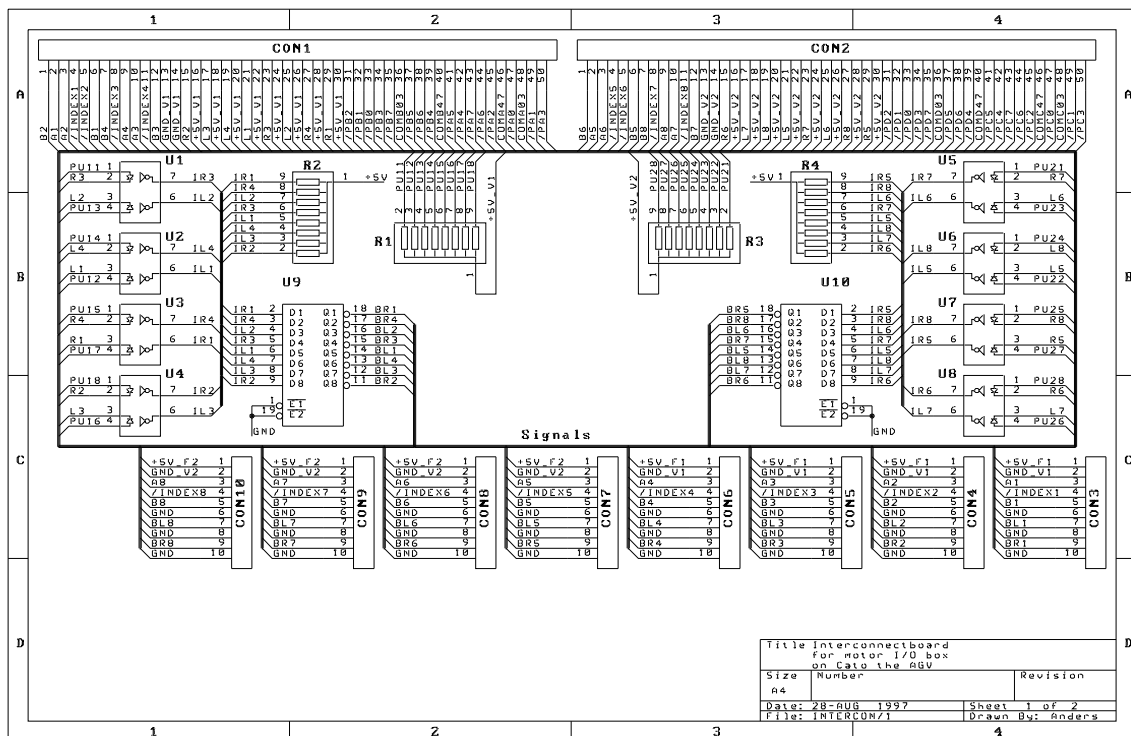
Figur 16.16(b) viser eksempler på tilslutning af kontakter, trykknapper eller relæer. Tilkoblingen til PA0 på figur 16.16(b) er den simpleste. Ønsker man at bruge et signal med højere spænding end 5V, indsættes en seriemodstand i kredsløbet, som vist ved tilkoblingen til PA1. Tilslutningen til PA2 og PA3 viser hvordan en kontakt med den ene pol til stel kan tilsluttes. Denne form for tilslutning bør kun anvendes hvor der ikke er andre muligheder, da der altid vil gå en strøm i kredsløbet. Enten gennem lysdioden i optokobleren, eller gennem kontakten. Især tilslutningen til PA3, spilder meget strøm.

Figur 16.16(c) viser to eksempler på tilslutning af TTL udgange. Begge metoder bruger en *open collector* opstilling i stedet for en kontakt. Begge opstillinger lader en strøm gå gennem optokobleren når TTL udgangen er *lav*². Som ved tilslutningen af PA2 og PA3 på figur 16.16(b), gælder det at tilslutningen til PA2 spilder energi, ved at lade en strøm løbe konstant.

16.5.3 Interconnect print

Motor I/O boxen består af 6 Rungner H-broer, og et enkelt print — kaldet interconnect printet, der varetager alle interconnect funktionerne, samt den galvaniske isolation af motorstyrings signalerne.

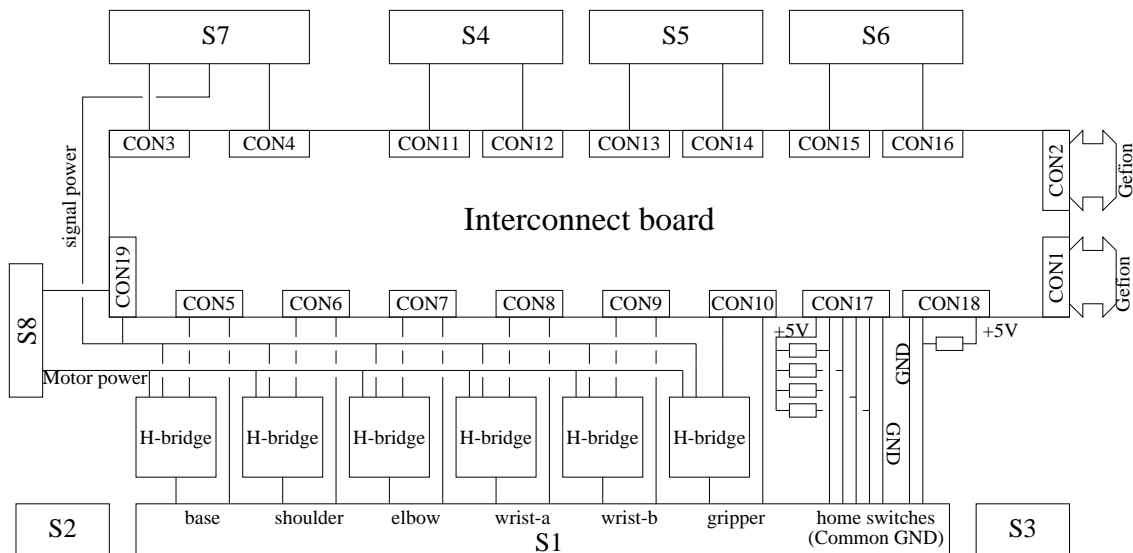
²Tilslutningen til PA0 på figur 16.16(c), kan også bruges om strøm-økonomisk tilslutning af en kontakt eller lignende, der har den ene pol til stel



Figur 16.17: Diagram over interconnect print

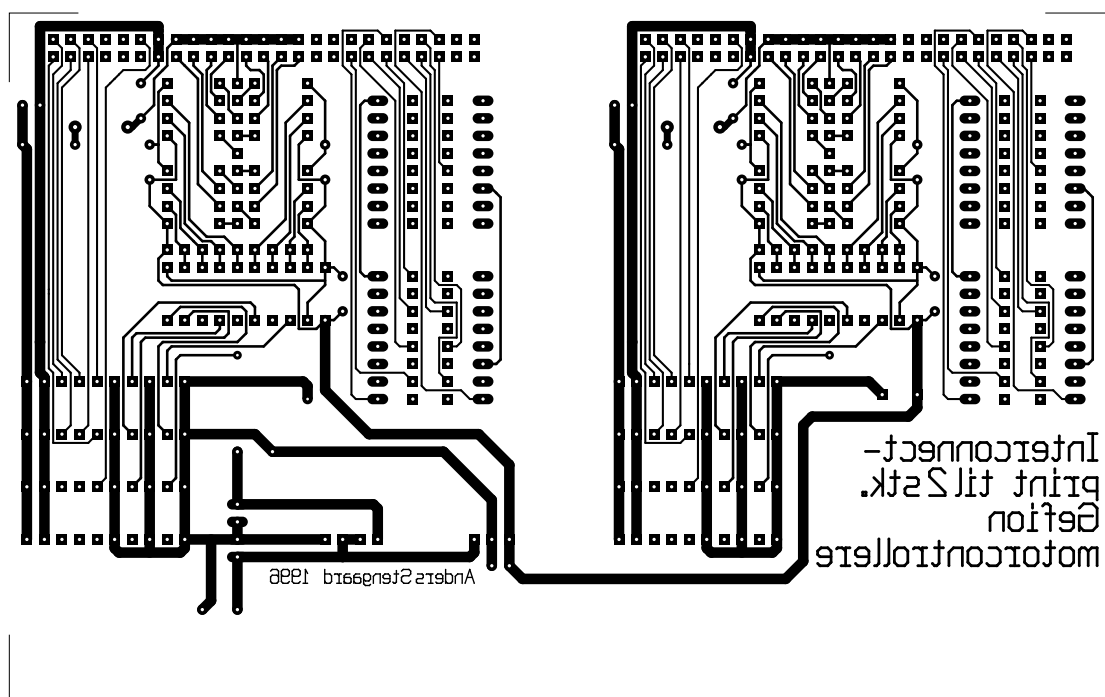
R1	8×	9 pin SIL		CON1	50 pol IDC	
R2	8×	9 pin SIL		CON2	50 pol IDC	
R3	8×	9 pin SIL		CON3	10 pol MOLEX	
R4	8×	9 pin SIL		CON4	10 pol MOLEX	
R5	4×	8 pin SIL		CON5	10 pol MOLEX	
R6	4×	8 pin SIL		CON6	10 pol MOLEX	
R7	4×	8 pin SIL		CON7	10 pol MOLEX	
R8	4×	8 pin SIL		CON8	10 pol MOLEX	
R9	4×	8 pin SIL		CON9	10 pol MOLEX	
R10	4×	8 pin SIL		CON10	10 pol MOLEX	
R11	4×	8 pin SIL		CON11	8 pol MOLEX	
R12	4×	8 pin SIL		CON12	8 pol MOLEX	
R13	1Ω	±5%	1/4W	CON13	8 pol MOLEX	
R14	1Ω	±5%	1/4W	CON14	8 pol MOLEX	
C1	100nF	Sibatit	> 15V	CON15	8 pol MOLEX	
C2	100nF	Sibatit	> 30V	CON16	8 pol MOLEX	
C3	100nF	Sibatit	> 5V	CON17	8 pol MOLEX	
C4	100nF	Sibatit	> 5V	CON18	8 pol MOLEX	
C5	100nF	Sibatit	> 5V	CON19	4 pol MOLEX	
C6	100nF	Sibatit	> 5V	U1	HCPL2630	Dual logic output optcoupler
C7	100nF	Sibatit	> 5V	U2	HCPL2630	Dual logic output optcoupler
C8	100nF	Sibatit	> 5V	U3	HCPL2630	Dual logic output optcoupler
C9	100nF	Sibatit	> 5V	U4	HCPL2630	Dual logic output optcoupler
C10	100nF	Sibatit	> 5V	U5	HCPL2630	Dual logic output optcoupler
C11	100nF	Sibatit	> 5V	U6	HCPL2630	Dual logic output optcoupler
C12	100nF	Sibatit	> 5V	U7	HCPL2630	Dual logic output optcoupler
C13		Tantal	> 5V	U8	HCPL2630	Dual logic output optcoupler
C14		Tantal	> 5V	U9	74HC540	Octal buffers
L1	30μH	3A		U10	74HC540	Octal buffers
L2	30μH	3A		U11	NME1215	1W DC-DC converter
				U12	MC7805CT	5V voltage regulator

Tabel 16.4: Komponentliste til interconnectboard

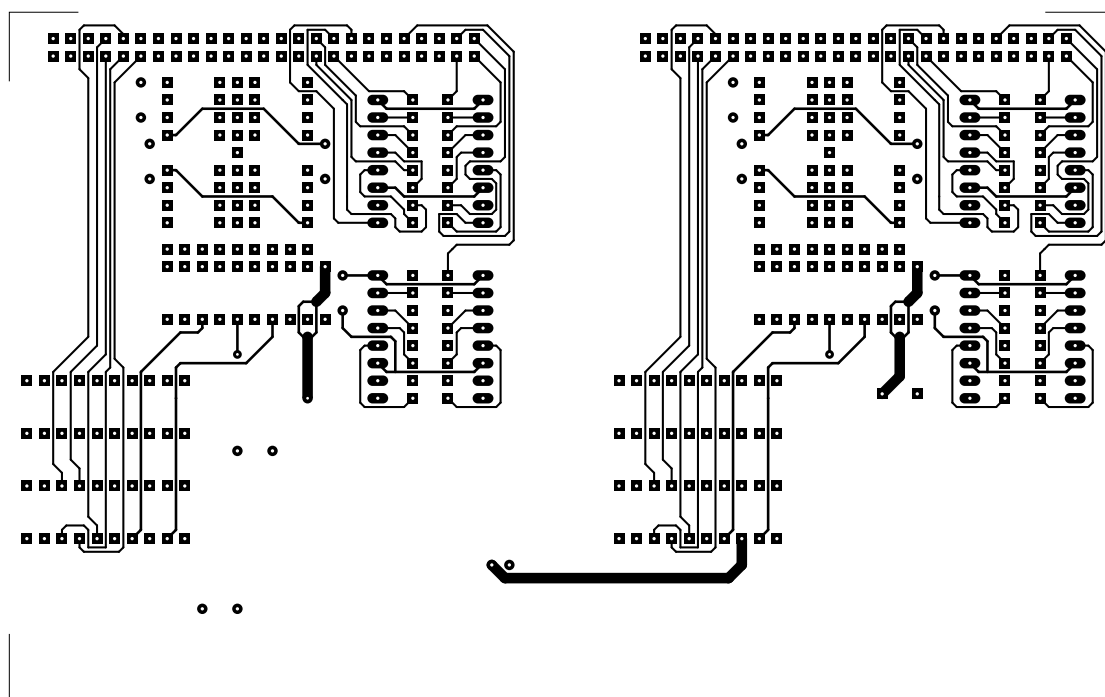


Figur 16.18: Skitse over interne forbindelser i motor I/O box

Figur 16.18 skitserer hvordan stikkene på interconnect printet er forbundet til de ydre stik i I/O boxen. Bemærk at Scorbotens *homing switches* har en fælles forbindelse til stel, og at de digitale indgange der aflæser dem derfor er koblet som hvilestrøms kredsløb vha. 100Ω modstande. Se figur 16.16(b).

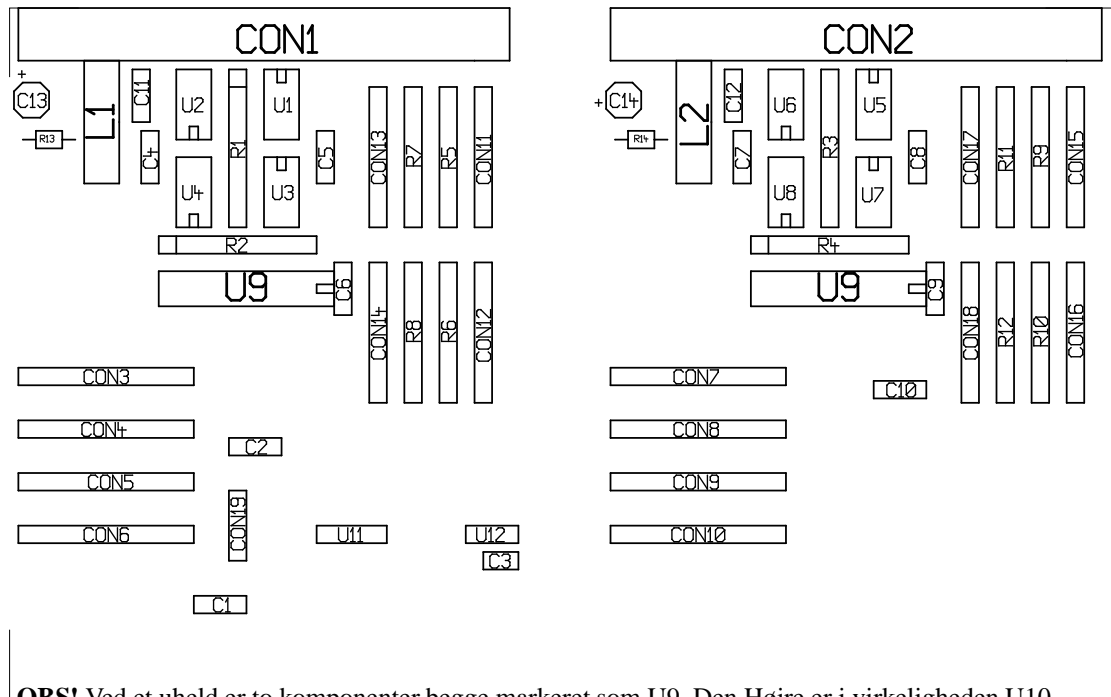


(a) Loddeside



(b) Komponentside

Figur 16.19: Printudlæg af interconnectboard (160mm×100mm)



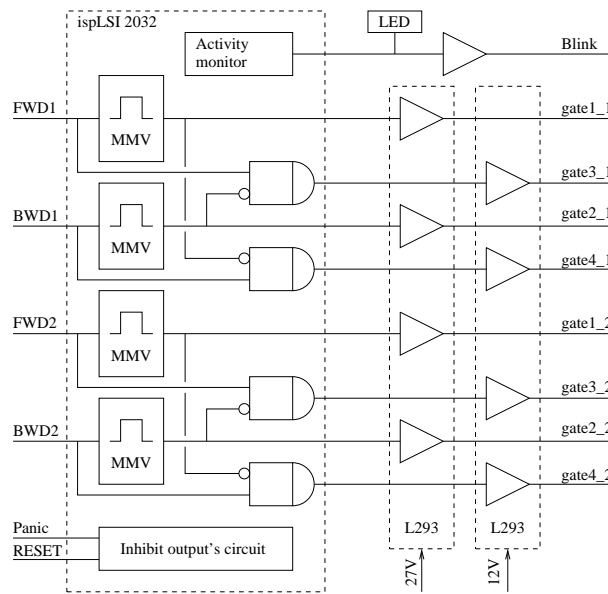
Figur 16.20: Komponentplacering på interconnectboard(160mm×100mm)

16.6 H-broer

Gefion motorcontroller kortet genererer to (12kHz) pulsvidde modulerede signaler for hver motor, et der signalerer kørsel i den ene retning, og et der signalerer kørsel i den anden. Signalerne for de to retninger er gensidigt udelukkende, og kan ikke være aktive samtidigt. Gefions motorstyrings udgange er *open collector* udgange. Udgangene bliver via optokoblere og buffere i motor I/O boxen, omdannet til TTL kompatible signaler, der sendes videre til H-bro interfacet på *power boardet*. H-bro interfacet omdanner TTL signalerne for **fremad** hhv. **bagud** til, spændings og tidsmæssigt, passende styrespændinger til transistorerne i H-broen.

16.6.1 H-bro interface

H-bro interfacet er forbundet til motor I/O boxen, vha et S8, et 20 polet IDC stik. Forbindelsen til H-broer, og enkodere, sker via S9 og S10 — 10 polede MOLEX stik. Forsyningsspændingen til, og signalerne fra enkodere passerer direkte fra S8 til S9 og S10. FWD og BWD signalerne fra motor I/O boxen, om sættes til gate spændinger til transistorerne i H-broen, af et logiknetværk, implementeret i en ispLSI2032 programmerbar logikkreds, og niveautilpasningen sker i to L293 driver kredse.



Figur 16.21: Blokdiagram af H-bro interface

FWD og BWD, 1 og 2 TTL kompatible indgange.

Gate 1 og 2 Push/pull udgange fra L293. $V_{low} \leq 1V$, $V_{high} \geq 24V$

Gate 3 og 4 Push/pull udgange fra L293. $V_{low} \leq 1V$, $V_{high} \geq 10V$

Pin nr	S8			Pin nr	S9/S10
	Forbindelse til motor I/O box				Enkoder og H-bro Venstre/højre side
1	+5V til enkoder 1	11	Signal GND til H-bro 1	1	+5V til enkoder 1/2
2	GND til enkoder 1	12	FWD (fremad) signal til H-bro 1	2	GND til enkoder 1/2
3	+5V til enkoder 2	13	Vggl (12V) til H-bro 1 (bruges ikke)	3	Ch-a fra encode 1/2
4	GND til enkoder 2	14	BWD (bagud) signal til H-bro 1	4	Ch-b fra encode 1/2
5	Ch-A fra enkoder 1	15	Vggh (27V) til H-bro 1 (bruges ikke)	5	GND til enkoder 1/2
6	Ch-B fra enkoder 1	16	Signal GND til H-bro 1	6	Signal GND til H-bro
7	Ch-A fra enkoder 2	17	FWD (fremad) signal til H-bro 2	7	Gate 4/8 til H-bro
8	Ch-B fra enkoder 2	18	Vggl (12V) til H-bro 2 (bruges ikke)	8	Gate 3/7 til H-bro
9	GND til enkoder 1	19	BWD (bagud) signal til H-bro 2	9	Gate 2/6 til H-bro
10	GND til enkoder 2	20	Vggh (27V) til H-bro 2 (bruges ikke)	10	Gate 1/5 til H-bro

Tabel 16.5: Benforbindelser til H-bro interface

Ligninger i ispLSI 2032 kredsen

LDE Report

Part: ispLSI2032-80LJ44

Pin Number	Pad Name	Pin Type	Pullup	Pin Number	Pad Name	Pin Type	Pullup
1	GND	Gnd		23	GND	Gnd	
7	XIN_D	Input	No	26	XPANIC	Input	Yes
11	XCLK	Input	No	34	VCC	Vcc	
12	VCC	Vcc		37	XOAH	Output	No
15	XOCH	Output	No	38	XOBH	Output	No

16	XODH	Output	No	39	XOAL	Output	No
17	XOCL	Output	No	40	XOBL	Output	No
18	XODL	Output	No	41	XBLINK	Output	No
21	XIN_C	Input	No	43	XIN_A	Input	No
				44	XIN_B	Input	No

 STATISTICS FOR GLB A0, USERNAME: MMV_A, ROUTED LOCATION: A0

GLB Input List:	GLB Output List:
I0 : QA2	O0 : QA2
I1 : QA1	O1 : QA1
I2 : QA0	O2 : QA0
I3 : IN_A	O3 : QA_D
CLK0 : CLK	
RESET : !RESET	

CELL EQUATIONS:

```

QA0.CLK = CLK;   QA_D.RE = IN_A;   QA0.D = (QA2.Q & QA1.Q & QA0.Q) # (!QA0.Q);
QA1.CLK = CLK;   QA0.RE = IN_A;   QA1.D = !((!QA2.Q & QA1.Q & QA0.Q) # (!QA1.Q & !QA0.Q));
QA2.CLK = CLK;   QA1.RE = IN_A;   QA2.D = !((!QA2.Q & !QA1.Q) # (!QA2.Q & !QA0.Q));
QA_D.CLK = CLK;  QA2.RE = IN_A;   QA_D.D = QA2.Q & QA1.Q & QA0.Q;

```

 STATISTICS FOR GLB A1, USERNAME: MMV_B, ROUTED LOCATION: A1

GLB Input List:	GLB Output List:
I0 : QB2	O0 : QB2
I1 : QB1	O1 : QB1
I2 : QB0	O2 : QB0
I3 : IN_B	O3 : QB_D
CLK0 : CLK	
RESET : !RESET	

CELL EQUATIONS:

```

QB0.CLK = CLK;  QB0.RE = IN_B;  QB0.D = (QB2.Q & QB1.Q & QB0.Q) # (!QB0.Q);
QB1.CLK = CLK;  QB1.RE = IN_B;  QB1.D = !((!QB2.Q & QB1.Q & QB0.Q) # (!QB1.Q & !QB0.Q));
QB2.CLK = CLK;  QB2.RE = IN_B;  QB2.D = !((!QB2.Q & !QB1.Q) # (!QB2.Q & !QB0.Q));
QB_D.CLK = CLK; QB_D.RE = IN_B; QB_D.D = QB2.Q & QB1.Q & QB0.Q;

```

 STATISTICS FOR GLB A2, USERNAME: MMV_C, ROUTED LOCATION: A2

GLB Input List:	GLB Output List:
I0 : QC2	O0 : QC2
I1 : QC1	O1 : QC1
I2 : QC0	O2 : QC0
I3 : IN_C	O3 : QC_D
CLK0 : CLK	
RESET : !RESET	

CELL EQUATIONS:

```

QC_D.CLK = CLK;  QC_D.RE = IN_C;  QC0.D = (QC2.Q & QC1.Q & QC0.Q) # (!QC0.Q);
QC0.CLK = CLK;  QC0.RE = IN_C;  QC1.D = !((!QC2.Q & QC1.Q & QC0.Q) # (!QC1.Q & !QC0.Q));
QC1.CLK = CLK;  QC1.RE = IN_C;  QC2.D = !((!QC2.Q & !QC1.Q) # (!QC2.Q & !QC0.Q));
QC2.CLK = CLK;  QC2.RE = IN_C;  QC_D.D = QC2.Q & QC1.Q & QC0.Q;

```

 STATISTICS FOR GLB A3, USERNAME: MMV_D, ROUTED LOCATION: A3

GLB Input List:	GLB Output List:
I0 : QD2	O0 : QD2
I1 : QD1	O1 : QD1
I2 : QD0	O2 : QD0

```

I3      : IN_D          O3      : QD_D
CLK0    : CLK
RESET   : !RESET

```

CELL EQUATIONS:

```

-----
QD0.CLK = CLK;  QD_D.RE = IN_D;  QD0.D = (QD2.Q & QD1.Q & QD0.Q) # (!QD0.Q);
QD1.CLK = CLK;  QD0.RE = IN_D;  QD1.D = (!((QD2.Q & QD1.Q & QD0.Q) # (!QD1.Q & !QD0.Q)));
QD2.CLK = CLK;  QD1.RE = IN_D;  QD2.D = (!((QD2.Q & !QD1.Q) # (!QD2.Q & !QD0.Q)));
QD_D.CLK = CLK;  QD2.RE = IN_D;  QD_D.D = QD2.Q & QD1.Q & QD0.Q;

```

```

*****
STATISTICS FOR GLB A4, ROUTED LOCATION: A4
*****

```

GLB Input List:	GLB Output List:
I0 : R2	O0 : R0
I1 : R1	O1 : R2
I2 : R0	O2 : R1
I3 : R3	O3 : R3
CLK0 : CLK	
RESET : !RESET	

CELL EQUATIONS:

```

-----
R0.CLK = CLK;  R0.D = (R2.Q & R1.Q & R0.Q & R3.Q) # (!R0.Q);
R1.CLK = CLK;  R1.D = (R2.Q & R1.Q & R0.Q & R3.Q) # (!R1.Q & R0.Q) # (R1.Q & !R0.Q);
R2.CLK = CLK;  R2.D = (!((R2.Q & !R1.Q) # (!R2.Q & !R0.Q) # (R2.Q & R1.Q & R0.Q & !R3.Q)));
R3.CLK = CLK;  R3.D = (!((R2.Q & !R3.Q) # (!R1.Q & !R3.Q) # (!R0.Q & !R3.Q)));

```

```

*****
STATISTICS FOR GLB A5, ROUTED LOCATION: A5
*****

```

GLB Input List:	GLB Output List:
I0 : QD_D	O2 : BLINK
I1 : QC_D	O3 : ENABLE
I2 : QB_D	
I3 : QA_D	
I4 : PANIC	
I5 : R3	
I6 : R2	
I7 : R1	
I8 : R0	

CELL EQUATIONS:

```

-----
ENABLE = R3 & R2 & R1 & R0;
BLINK = (!QD_D & !PANIC & R3 & R2 & R1 & R0) # (!QC_D & !PANIC & R3 & R2 & R1 & R0) #
        (!QB_D & !PANIC & R3 & R2 & R1 & R0) # (!QA_D & !PANIC & R3 & R2 & R1 & R0);

```

```

*****
STATISTICS FOR GLB A6, ROUTED LOCATION: A6
*****

```

GLB Input List:	GLB Output List:
I0 : IN_D	O0 : OCH
I1 : QC_D	O1 : OCL
I2 : QD2	O2 : ODH
I3 : IN_C	O3 : ODL
I4 : PANIC	
I5 : ENABLE	
I6 : QD_D	
I7 : QC2	

CELL EQUATIONS:

```

ODL = IN_D & QC_D & !PANIC & ENABLE;
ODH = QC_D & !QD2 & !PANIC & ENABLE;
OCL = IN_C & !PANIC & ENABLE & QD_D;
OCH = !PANIC & ENABLE & QD_D & !QC2;

```

```
*****
STATISTICS FOR GLB A7, ROUTED LOCATION: A7
*****
```

```
GLB Input List:          GLB Output List:
  I0      : IN_B          O0      : OAH
  I1      : QA_D          O1      : OAL
  I2      : QB2           O2      : OBH
  I3      : IN_A          O3      : OBL
  I4      : PANIC
  I5      : ENABLE
  I6      : QB_D
  I7      : QA2
```

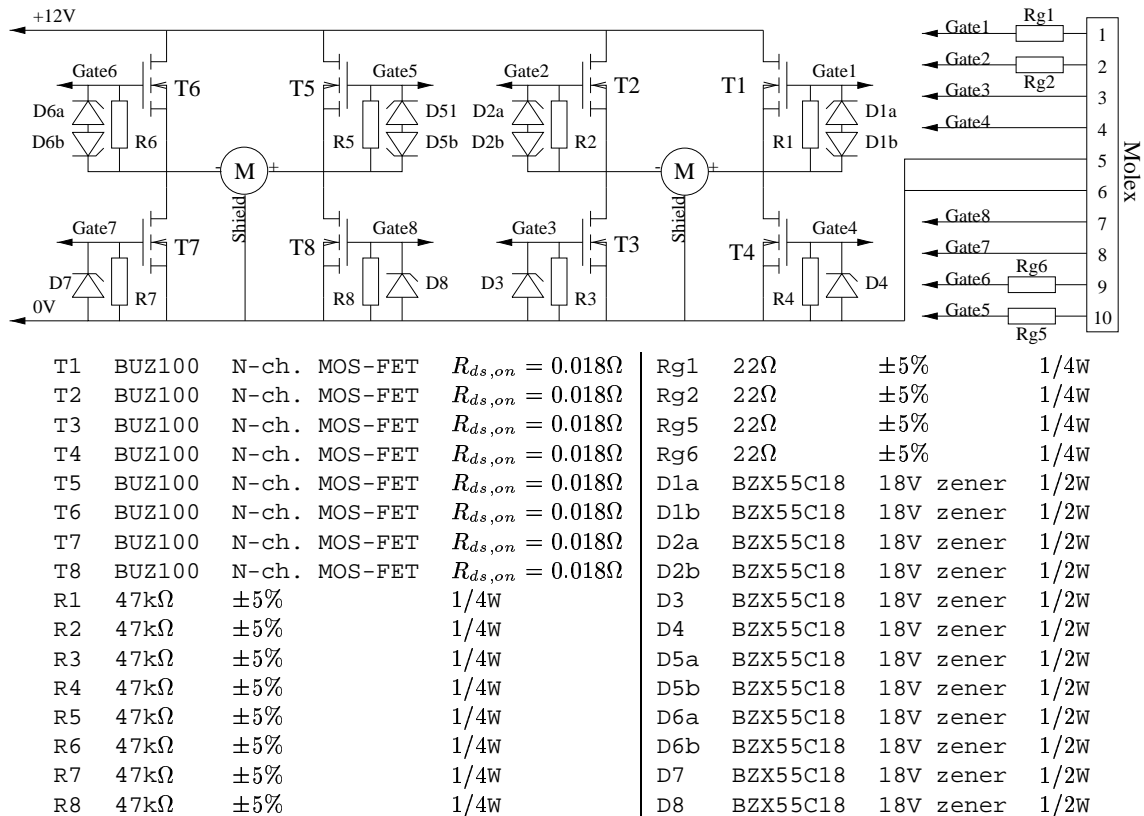
```
CELL EQUATIONS:
OBL = IN_B & QA_D & !PANIC & ENABLE;
OBH = QA_D & !QB2 & !PANIC & ENABLE;
OAL = IN_A & !PANIC & ENABLE & QB_D;
OAH = !PANIC & ENABLE & QB_D & !QA2;
```

```
*****
STATISTICS FOR I/O CELLS
*****
```

I/O cell	User name	Input List:	Output List:	CELL EQUATIONS:
IO0	A	PAD : XIN_A	OUT : IN_A	XPIN IO XIN_A LOCK 43; IB11(IN_A, XIN_A);
IO1	B	PAD : XIN_B	OUT : IN_B	XPIN IO XIN_B LOCK 44; IB11(IN_B, XIN_B);
IO2	C	PAD : XIN_C	OUT : IN_C	XPIN IO XIN_C LOCK 21; IB11(IN_C, XIN_C);
IO3	D	PAD : XIN_D	OUT : IN_D	XPIN IO XIN_D LOCK 7; IB11(IN_D, XIN_D);
IO15	Panic	PAD : XPANIC	OUT : PANIC	XPIN IO XPANIC LOCK 26 PULLUP; IB11(PANIC, XPANIC);
IO16		IMUX : BLINK	PAD : XBLINK	XPIN IO XBLINK LOCK 41; OB11(XBLINK, BLINK);
IO24	DL	IMUX : ODL	PAD : XODL	XPIN IO XODL LOCK 18; OB11(XODL, ODL);
IO25	DH	IMUX : ODH	PAD : XODH	XPIN IO XODH LOCK 16; OB11(XODH, ODH);
IO26	CL	IMUX : OCL	PAD : XOCL	XPIN IO XOCL LOCK 17; OB11(XOCL, OCL);
IO27	CH	IMUX : OCH	PAD : XOCH	XPIN IO XOCH LOCK 15; OB11(XOCH, OCH);
IO28	BL	IMUX : OBL	PAD : XOBL	XPIN IO XOBL LOCK 40; OB11(XOBL, OBL);
IO29	BH	IMUX : OBH	PAD : XOBH	XPIN IO XOBH LOCK 38; OB11(XOBH, OBH);
IO30	AL	IMUX : OAL	PAD : XOAL	XPIN IO XOAL LOCK 39; OB11(XOAL, OAL);
IO31	AH	IMUX : OAH	PAD : XOAH	XPIN IO XOAH LOCK 37; OB11(XOAH, OAH);
Y0		PAD : XCLK	OUT : CLK	XPIN CLK XCLK LOCK 11; IB11(CLK, XCLK);

16.6.2 H-bro

De to H-broer til drivmotorerne er opbygget vha. 8 stk BUZ100 N-kanal MOS-FET transistorer. Transistorerne er skruet direkte på stelplanet med M3 nylonskruer, og elektrisk isoleret fra stelplanet vha. varmeledende silikoneskiver. T1,T2,T5,og T6's drain's er via TO-220 husenes køleplader, og påskruede kabelsko, forbundet individuelt til +12V forsyningen. Tilsvarende er T3, T4, T7, og T8's drain's forbundet individuelt til motorstikkene.



Figur 16.22: Diagram over dobbelt H-bro

16.7 Computer

Som styrecomputer anvendes en 16/24 bits VME computer, opbygget i et standard 19" rack. Computeren er opbygget af følgende komponenter:

Backplane: 9 slots S1 VME-backplane fra PEP. Se [26]

CPU modul: VMPPM KC2, 68020 baseret CPU fra PEP. Se [29]

Disk controller: VMSC VME disk controller fra PEP.

Floppy drev: 720Kb standard 3.5" floppy drev.

Harddisk: 20MB 3.5" MFM harddisk.

Analog indgange: Analog Devices RTI-600, med 16 12 bits analoge indgange (multiplexede), fra Analog devices. Se [22]

Strømforsyning: 60W 12V til +5V, +12V, og -12V Switch mode forsyning fra VERO.

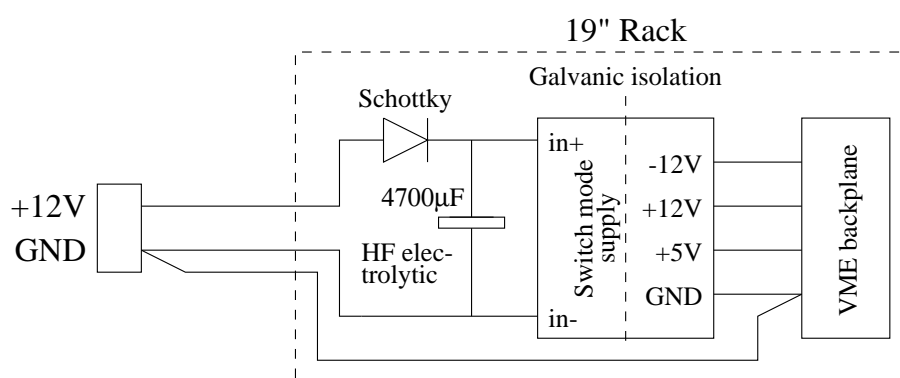
16.7.1 Konfiguration af backplane

Da der kun er en *master* på bussen (CPU kortet), skal *bus request/ bus grant* jumperne alle være isat.

Alle *interrupt acknowledge* skal sidde i, undtaget ved de VME kort der anvender interrupts. Dvs. de skal være fjernet ud for de *slots* der rummer: Heimdal modulet, og diskcontrolleren.

16.7.2 Strømforsyning

Figur 16.23 viser opbygningen af computerens strømforsyning.



Figur 16.23: VME computerens strømforsyning

Computeren strømforsynes vha. en rød (+12V) og en sort (GND) 4mm² ledning, der vha. en 2 polet adskillelig klemrække, er forbundet til strømforsyningen på Cato's elektronik-monteringsplade.

Dioden og kondensatoren bortfiltrerer kortvarige spændingsfald, der ellers kunne have fået switch mode forsyningen til fejle, med *reset* af computeren til følge. Dioden sikrer også mod evt. forkert polarisering af forsyningsledningerne.

Computerens GND holdes på samme potentiale som Cato's GND, vha. en separat 0.5mm² ledning, der også er koblet til den adskillelige klemrække.

Kapitel 17

Dokumentation af James

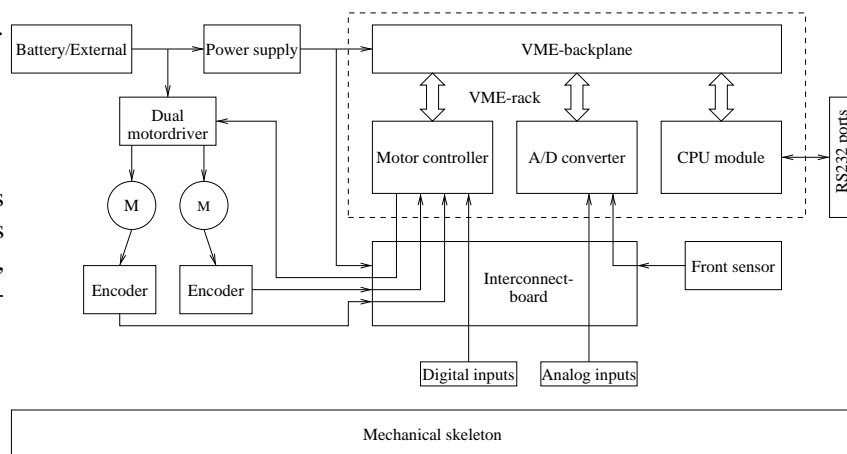
James er den AGV som hardware-interessegruppen for datateknologistuderende deltog — og vandt — med, i konkurrencen *DTU robocup 1997*, hvor AGV'er skulle følge en hvid linie i gulvet.

James er i meget høj grad baseret på mit specialeprojekt, men en lang række mennesker har bidraget til udviklingen af programmel mekanik, og elektronik til denne specifikke inkarnation af speciale.

I dette afsnit dokumenteres de dele af James som jeg selv har stået for, nemlig:

- Overordnet struktur.
- Strømforsyning.
- Interconnect print.
- H-broer.

Motorstyringen på James er det ene motorstyrings modul (Gefion) fra Cato, og er dokumenteret andetsteds i denne rapport.



Figur 17.1: Blokdiagram over James

A/D converteren, CPU modulet er professionelle VME-moduler fremstillet af hhv. *Analog Devices* og *PEP modular computers*. De er dokumenteret i manualer derfra.

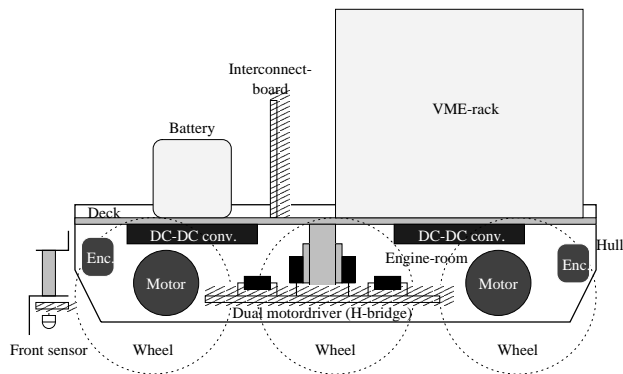
17.1 Mekanik

Dette afsnit giver ikke en fuldstændig dokumentation af mekanikken på James, men giver det overblik der er nødvendigt i forbindelse med dokumentationen af elektronikken.

Figur 17.2 viser en tværsnitsskitse af James.

James er baseret på byggesættet: *Rodeo 6X6* fra modelløsefirmaet *Robbe*. Byggesættet var oprindeligt et amfibiekøretøj beregnet til fjernstyring.

Al mekanikken er bygget op omkring skroget, der har facon af en skotøjsæske. De 6 hjulaksler er vha. kuglelejer og *stævnør* ført ud gennem skrogets sider. Motorer og gearkasser er skruet fast i skrogets bund. Ved hjælp af tandremme trækker motorerne hver især på tre hjul, så køretøjet nærmest styres som et bæltekøretøj.



Figur 17.2: Tværsnit af James

Vi har monteret en optisk inkremental enkoder på trækket fra hver motor. Enkoderne er monteret på for- hhv. bag-enden af skroget vha. et beslag, og er forbundet til transmissionen vha. et tandhjul.

For at have en god monteringsplatform har vi placeret en 4mm aluminiumsplade som låg over skroget. Rummet under pladen kaldes *maskinrummet*. Maskinrummet rummer:

- Motorer med gearkasser.
- Enkoder.
- Strømforsyning.
- Motordriver (H-broer)

Batteriet, eller en ekstern strømforsyning, er forbundet til elektronikken i maskinrummet vha. et 4mm² to-leder kabel, der går gennem et hul i dækkets forende.

VME-racket er forbundet til strømforsyningen i maskinrummet vha. fire enkelte 1mm² ledninger der går gennem et hul i dækkets bagende.

Alle andre elektriske forbindelser til maskinrummet foregår vha. et 25 polet sub-D stik monteret i dækket, foran VME-racket.

De elektriske forbindelser gennem sub-D stikket i dækket, skal fordeles til forskellige kilder/modtagere. For at undgå ledningsvirvar, er der fremstillet et interconnect print med de fornødne stik til at fordele signalerne til/fra: Maskinrum, A/D konverter, motorstyring, og sensorer. Printet er forsynet med et 25 polet sub-D stik, der passer til stikket i dækket, således at printet spændes fast til stikket i dækket, i lodret position.

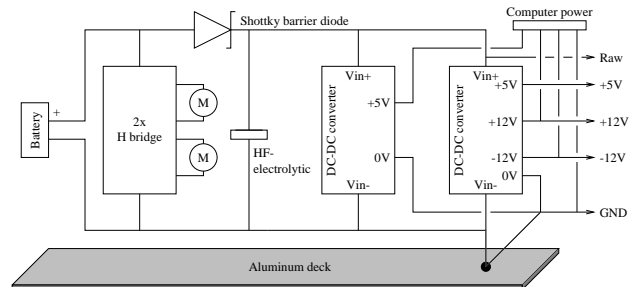
VME-racket er bygget over et 19 tommer subrack, der er kortet af for at passe til skrogets bredde. Bagest i racket sidder et 7-slots VME-backplane fra *VERO electronics*. VME-backplanen forbinder et VMPM-KC2 68020 baseret CPU modul fra *PEP modular computers* med et Gefion motorstyrings modul, et A/D convertermodul fra *Analog devices*, og en floppy/harddisk controller fra *PEP*. Ovenpå racket er monteret et 880KB 3.5 tomme floppydrev fra *Teak*.

17.2 Strømforsyning

Figur 17.3 viser diagrammet over James strømforsyning.

James energiforsynes fra en NiCd-, Bly-akkumulator, eller en anden kilde der leverer 12-15V, ved strømme på 5-40A afhængigt af James forbrug.

Selv gode akkumulatorer har en indre modstand der bevirker at variationer i belastningen medfører variationer i polspændingen. Motordriveren i James fungerer i *switch mode*, ved frekvenser på både ca. 12kHz og 500Hz.



Figur 17.3: James strømforsyning

Ved store belastninger giver switch mode motordriveren anledning til en 500Hz og 12kHz *ripple* spænding over batteriet på op til flere volt. For at komme denne ripple støj til livs ensrettes og udglattes batterispændingen af en diode og et sæt HF elektrolyt kondensatorer.

Den filtrerede batterispænding bruges til at drive et sæt 25W DC-DC convertere, der genererer +5V, ved op til 5A, til VME-computeren, samt +12V (0.8A) -12V (0.8A) og +5V (1A) til VME-computeren og andre kredsløb på james.

James strømforsyning er opbygget med stjernestel, hvor James 4mm tykke aluminiumsdæk udgør det fælles stelpunkt.

James tændes og slukkes ved at tilslutte hhv. afbryde batteriet.

17.3 Motorer og enkodere

De mekaniske aspekter omkring motorer og enkodere blev varetaget af andre deltagere i James projektet¹, og bliver derfor ikke dokumenteret her, men blot overfladisk beskrevet.

17.3.1 Motorer

De anvendte motorer fulgte med det oprindelige byggesæt James er lavet over. Der er tale om almindelige DC-motorer, med en nominel maximal polspænding på 7.5V, og et nominelt maximalt effektforbrug på 100W. Motoren er bygget op om et tredelt anker, med 8 viklinger på hvert anker. Viklingerne har en indre modstand på ca. 0.7Ω. I spidsbelastninger kan Motorene forbruge 20A ved en polspænding 7.5V.

¹Primært Anders Tønnesen og Einar Hougs

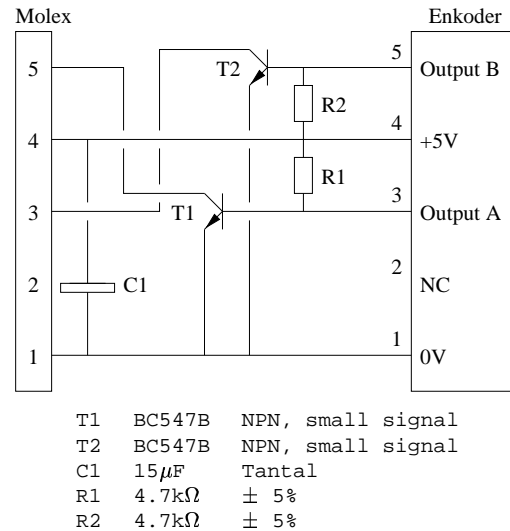
17.3.2 Enkodere

Der anvendes de samme enkodere i James og Cato. Enkoderne afgiver en to-bits graykode, der gentager sig 128 gange på en omdrejning af enkoderakslen, hvilket giver mulighed for aflæsning af 512 diskrete positioner på en omdrejning.

Enkoderen er via et tandhjul forbundet til samme hjulaksel som motoren trækker på. Tandhjulstrækket mellem enkoder og hjulaksel, samt tandhjulsgæret mellem hjulakslen og motorens drivaksel, giver anledning til en del slør mellem motor og enkoder. Sløret forringer stabiliteten af det reguleringssystem der skal regulere motorerne, og betyder at der i nogen grad må gives afkald på precision og responstid for at undgå at systemet går i selvsving.

Enkoderne forsynes med +5V direkte fra Gefion motorcontrolleren. Enkoderens to udgange er *open collector*. For at beskytte enkoderne mod forkert tilslutning mv. er der sat et lille print med bassende buffere på hver enkoder.

Bufferprintet er lavet så det har samme tilslutninger som enkoderen oprindelgt havde, og dermed virker *transperant*. De enkelte buffere på bufferprintet inverterer ganske vist signaler, men bufferprintet bytter samtidigt A og B signalet, hvilket ophæver inverteringen.



Figur 17.4: Diagram over enkoderbuffer

17.4 Interconnect print

Interconnect printets opgave er at forbinde de relevante signaler til og fra Gefion VME-motorcontrolleren, VME A/D-converteren, Motordrivere, enkodere, strømforsyning, sensorer, og evt. andre enheder.

Interconnect printet er fastmonteret på James' dæk, via et 25 polet sub-D stik, der danner forbindelsen til elektronikken i maskionrummet. Tabel: 17.1 viser forbindelserne i dette stik. Printet står på højkant i forhold til dækket, og sub-D stikket er placeret på printetst nederste kant.

På Printets øverste kant, sidder to 50 polede IDC hanstik, der forbinder printet til hhv. motorcontrolleren (Gefion), og A/D convertren i VME-bus computeren. **Begge enheder forbindes vha. et 50 polet fladkabel, så der skal udvises omhyggelighed!** Forbindelserne til Gefion og A/D converteren er gengivet i deres respektive manualer.

ben	signal	Beskrivelse	fra	til
1	MC +5V	Spændingsforsyning	VME-motorcontroller	Enkoder 1
2	MC GND	Spændingsforsyning	VME-motorcontroller	Enkoder 1
3	A1	Enkodersignal	Enkoder 1	VME-motorcontroller
4	B1	Enkodersignal	Enkoder 1	VME-motorcontroller
5	MC +5V	Spændingsforsyning	VME-motorcontroller	Optokobler på motordriver
6	MC +5V	Spændingsforsyning	VME-motorcontroller	Optokobler på motordriver
7	MC +5V	Spændingsforsyning	VME-motorcontroller	Optokobler på motordriver
8	MC +5V	Spændingsforsyning	VME-motorcontroller	Optokobler på motordriver
9	+12V	Spændingsforsyning	DC-DC converter	Sensorer mv.
10	+5V	Spændingsforsyning	DC-DC converter	Sensorer mv.
11	GND	Spændingsforsyning	DC-DC converter	Sensorer mv.
12	-12V	Spændingsforsyning	DC-DC converter	Sensorer mv.
13				
14	B2	Enkodersignal	Enkoder 2	VME-motorcontroller
15	A2	Enkodersignal	Enkoder 2	VME motorcontroller
16	MC GND	Spændingsforsyning	VME-motorcontroller	Enkoder 2
17	MC +5	Spændingsforsyning	VME-motorcontroller	Enkoder 2
18	OA2	Motorstyringssignal	VME-motorcontroller	Optokobler på motordriver
19	OB1	Motorstyringssignal	VME-motorcontroller	Optokobler på motordriver
20	OB2	Motorstyringssignal	VME-motorcontroller	Optokobler på motordriver
21	OA1	Motorstyringssignal	VME-motorcontroller	Optokobler på motordriver
22				
23				
24	Power GND	Spændingsforsyning	Batterifilter	Effektdel af sensorer mv.
25	Power +12V	Spændingsforsyning	Batterifilter	Effektdel af sensorer mv.

Tabel 17.1: Forbindelser i stikket gennem dækket

Gefion motorcontrolleren styrer James to motorer vha. fire digitale pulsviddemodulerede signaler, to til hver motor. For hver motor koder det ene signal for 'energitilførslen' til at drive James fremad hhv. bagud. Signalerne sendes videre til motordriveren i maskinrummet, men er også koblet til fire lysdioder på interconnectboardet (D1 – D4). D1 og D2 koder for motor 1, mens D3 og D4 koder for motor 2. D1 og D3 er røde og koder for 'baglæns', mens D2 og D4 er grønne og koder for 'fremad'. Da Gefion anvender pulsvidde modulerede signaler, med en grundfrekvens på 12kHz, vil lysdioderne se ud til at lyse jævnt, med en lysstyrke der indikerer spændingen over motorene.

For at kunne forbinde James sensorer til spændingsforsyning og relevante A/D indgange, bruges 4 10-polede MOLEX hanstik (J4 – J7). Alle fire stik har forbindelser til 0V, +12V, og -12V på de samme ben,

og har derudover op til 7 forbindelser til indgange på A/D converteren. J4 og J5 er beregnet til tilknytning af differentielle optiske sensorer, eller andre enheder med to analoge udgange. J6 er beregnet til en 7-punkts reflektions sensor, eller anden enhed med 7 analoge udgange. J7 er ikke reserveret til noget bestemt formål, men danner forbindelser til de resterende 5 analoge indgange på A/D converteren. Tabel 17.2 viser forbindelserne på de fire stik.

J4		J5		J6		J7	
ben	signal	ben	signal	ben	signal	ben	signal
1		1		1	A/D channel 10	1	
2		2		2	A/D channel 9	2	
3		3		3	A/D channel 8	3	A/D channel 15
4	+12V	4	+12V	4	+12V	4	+12V
5	-12V	5	-12V	5	-12V	5	-12V
6	GND	6	GND	6	GND	6	GND
7		7		7	A/D channel 7	7	A/D channel 14
8		8		8	A/D channel 6	8	A/D channel 13
9	A/D channel 1	9	A/D channel 3	9	A/D channel 5	9	A/D channel 12
10	A/D channel 0	10	A/D channel 2	10	A/D channel 4	10	A/D channel 11

Tabel 17.2: Forbindelser til 'sensor' stik

Gefion motorcontroller modulet har 16 optisk isolerede digitale indgange. Interconnectboardet har ført 8 af disse ud på et stik (J8), sammen med 5V spændingsforsyningen fra Gefion. Indgangene kan bruges til tilkobling af diverse digitale signaler, f.eks. kontakter, trykknapper, joysticks etc.

Interconnectboardet har indbygget formodstande til optokoblerne, så en indgang skal blot lægges til +5V, for at aktivere den. En aktiv indgang aflæses som logisk 0 af Gefions I/O port, en passiv som logisk 1.

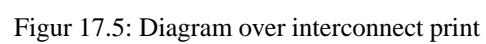
J8	
ben	signal
1	Port A, bit 0
2	Port A, bit 1
3	Port A, bit 2
4	Port A, bit 3
5	Port A, bit 4
6	Port A, bit 5
7	Port A, bit 6
8	Port A, bit 7
9	Gefion GND
10	Gefion +5V

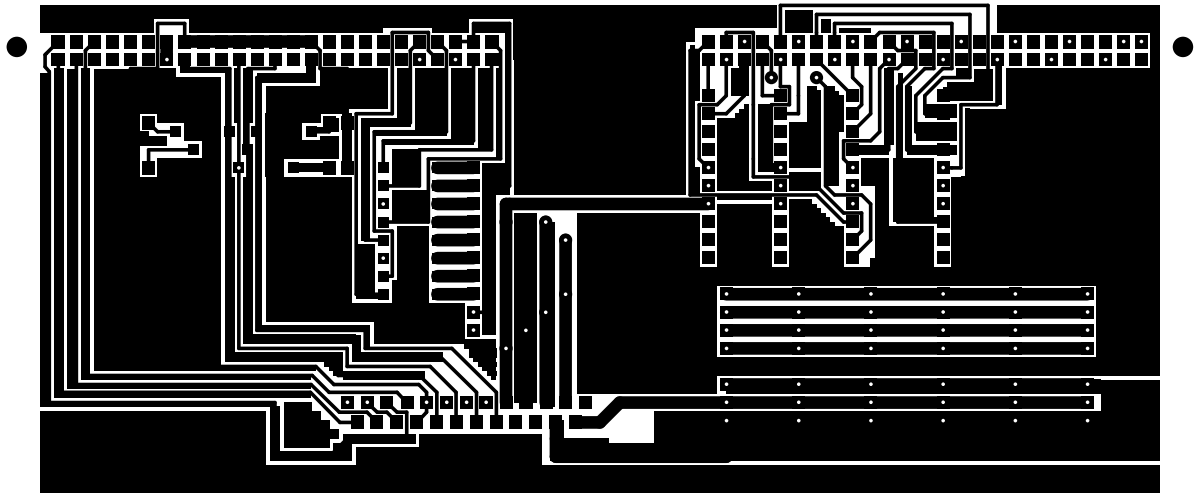
Tabel 17.3: Forbindelser til digitale indgange

For at kunne forsyne sensorer og andre enheder med energi, er interconnectboardet forsynet med en række 3 polede MOLEX stik (J9 – J14), der giver adgang til den filtrerede spænding fra batteriet, og en række 4 polede MOLEX stik (J15 – J20) der giver adgang til +5V, +12V, og -12V fra DC-DC converteren i maskinrummet. Den filtrerede batterispænding kan bruges til kredsløb der skal bruge meget strøm, men ikke er kritiske mht. stabil spænding og galvanisk koblet støj, f.eks. til drift af lysdioder i optiske sensorer. Spændingsforsyningen fra DC-DC converteren kan bruges til diverse analoge og digitale kredsløb.

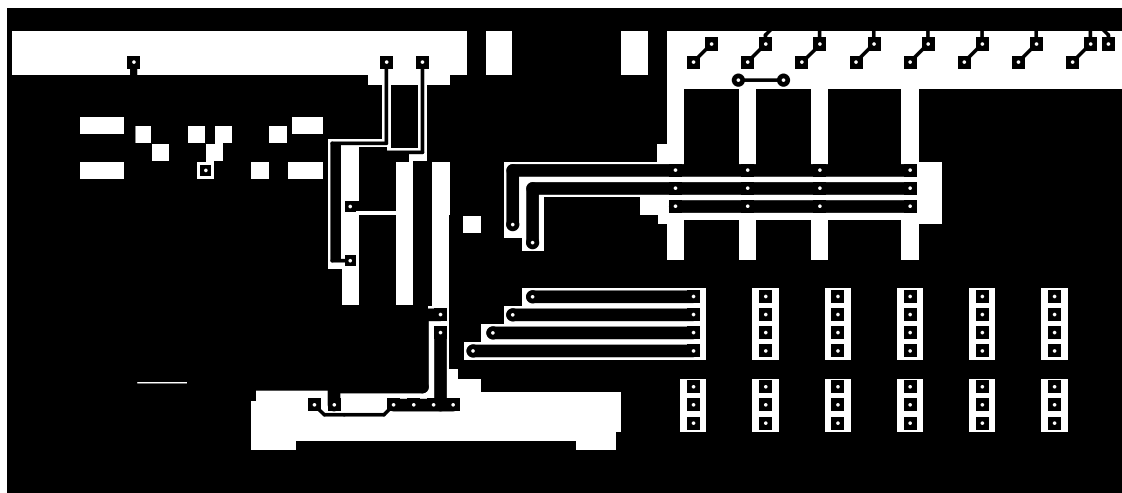
J9 – J14	
ben	signal
1	Power GND
2	Power +12V
3	Power GND
J15 – J20	
1	+12V
2	+5V
3	GND
4	-12V

Tabel 17.4: Forbindelser til spændingsforsynings stik



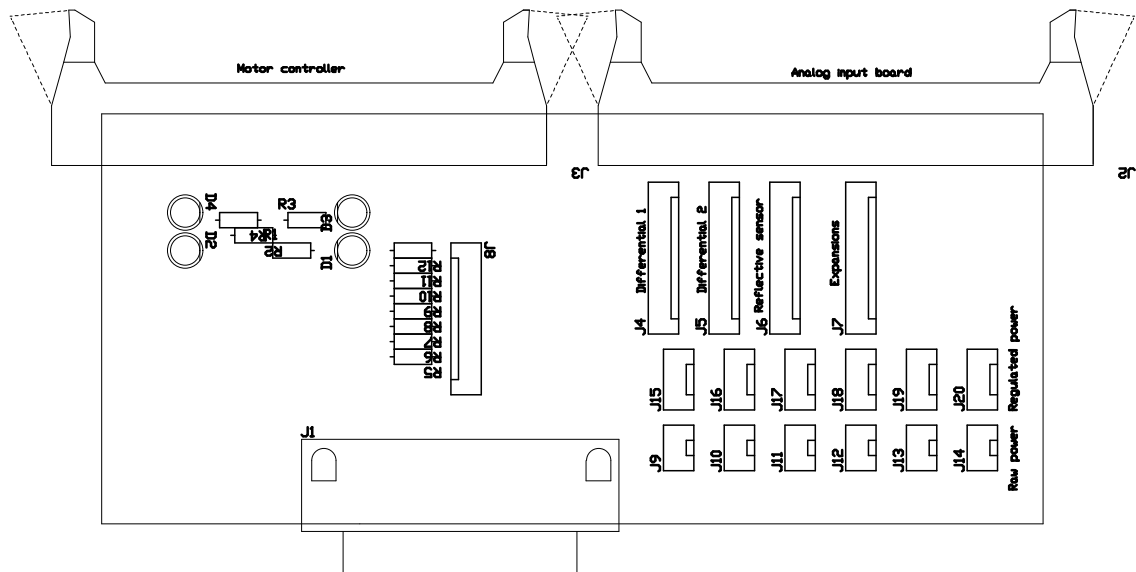


(a) Loddesside



(b) Komponentside

Figur 17.6: Printudlæg til interconnectprint



Figur 17.7: Komponentplacering

R1	470Ω	1/4W	J7	10 ben	MOLEX han
R2	470Ω	1/4W	J8	10 ben	MOLEX han
R3	470Ω	1/4W	J9	3 ben	MOLEX han
R4	470Ω	1/4W	J10	3 ben	MOLEX han
R5	470Ω	1/4W	J11	3 ben	MOLEX han
R6	470Ω	1/4W	J12	3 ben	MOLEX han
R7	470Ω	1/4W	J13	3 ben	MOLEX han
R8	470Ω	1/4W	J14	3 ben	MOLEX han
R9	470Ω	1/4W	J15	4 ben	MOLEX han
R10	470Ω	1/4W	J16	4 ben	MOLEX han
R11	470Ω	1/4W	J17	4 ben	MOLEX han
R12	470Ω	1/4W	J18	4 ben	MOLEX han
J1	25 ben	SUB-D han 90°	J19	4 ben	MOLEX han
J2	50 ben	IDC han 90°	J20	4 ben	MOLEX han
J3	50 ben	IDC han 90°	D1	3mm rød	standard LED
J4	10 ben	MOLEX han	D2	3mm rød	standard LED
J5	10 ben	MOLEX han	D3	3mm grøn	standard LED
J6	10 ben	MOLEX han	D4	3mm grøn	standard LED

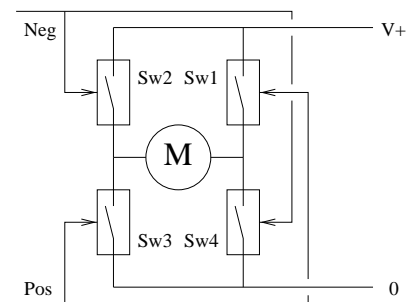
Tabel 17.5: Komponentliste

17.5 Motor-drivere

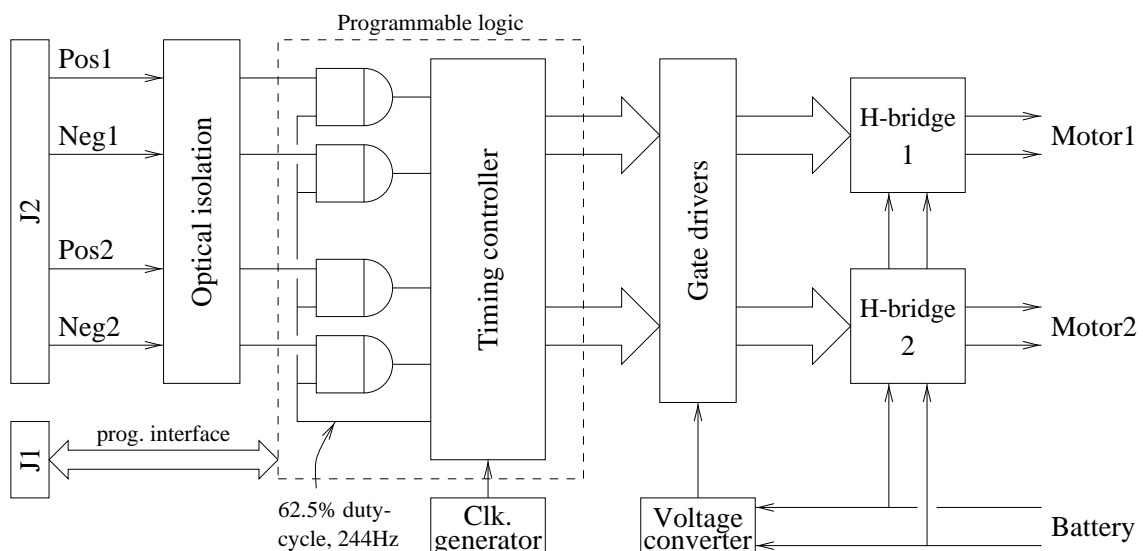
James motordrivere er opbygget omkring to switch-mode H-broer, som skitseret i figur 17.8. Tændes Sw1 og Sw3 samtidigt påtrykkes motoren den fulde forsyningsspænding. Hvis mindst tre kontakter er slukkede påtrykkes motoren ingen spænding. De fire kontakter styres parvist. Tændes Sw2 og Sw4, påtrykkes motoren den fulde forsyningsspænding med modsat fortegn (negativ). Tændes Sw1 og Sw4, eller Sw2 og Sw3 samtidigt opstår der en kortslutning.

Motordriverens opgave er at tænde og slukke de rette 'kontakter' i sine H-broer, i den konfiguration der signaleres med de fire optisk isolerede digitale signaler fra motor controlleren.

Da en motor pga. interti, selvinduktion og indre modstand fungerer som et lavpasfilter, kan energitilførslen reguleres trinløst ved at styre et kontaktsæt med et firkantsignal hvis grundfrekvens er højere end motorens knækfrekvens. Den effektive spænding over motoren svarer til produktet af den anvendte forsyningsspænding, og firkantsignalets *duty cycle*.



Figur 17.8: Princippet i en switch-mode H-bro



Figur 17.9: Blokdigram over James motordriver

Ud over de to H-broer, der er implementeret vha. MOS-FET transistorer, indeholder motordriveren 8 små udgangstrin, der bruges til at drive *gaten* på de 8 transistorer, en spændings forsyning der genererer de nødvendige styrespændinger til MOS-FET transistorerne, en optisk isolering mellem styresignalerne fra motor controlleren (Gefion), samt en programmerbar kreds der sørger for at de rigtige MOS-FET transistorer er tændt/slukket på de rigtige tidspunkter.

Motorerne i James er beregnet til drift med 7.2V batterier, mens James bruger et 14.4V batteri. For at reducere den effektive spænding motordriveren kan levere, multipliceres/choppes de pulsvide modulerede signaler fra motorcontrolleren, med et digitalt signal der har en *duty cycle* på 62.5%. Dette reducerer den effektive spænding over motoren til $62.5\%^2$ af hvad den ellers havde været. Chopper signalet har en frekvens på ca. 244Hz

²Pga. James forøgede vægt var der brug for lidt ekstra kraft i forhold til de ideelle 50%

17.5.1 Tilslutning

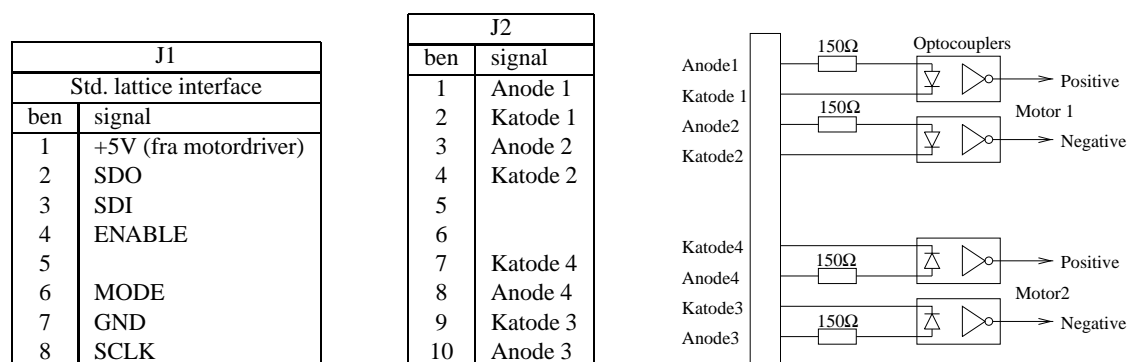
For at spare plads, er spændingsforsyningen og motortilslutningen ikke ført ud på et stik. I stedet loddes ledningerne til dette brug, direkte på printbanerne, der for at kunne lede de nødvendige strømme er så brede at denne monterings teknik ikke volder problemer. Den bedste forbindelse opnås ved at afisolere ca. 5mm af ledningen, skille kobbertrådene i ledningen op i en vifte, forfinne viften, og presse viften helt ned til den forfinede printbane med loddekolben indtil al tinnet omkring viften er helt smeltet. Ledningerne bør aflastes et stykke fra lodningen, for at forhindre at den knækker ved loddestedet.

Figur 17.12 viser printlayoutet for hhv. lodde-/under-side, og komponent-/over-side. Udgangstransistorerne er monteret i to rækker. De høje transistorer — dem der trækker spændingen opad — T1, T2, T5, og T6 i den ene række, og de lave — dem der trækker spændingen nedad — T3, T4, T7, og T8 i den anden række. På komponentsiden løber to brede printbaner imellem de to rækker transistorer. De to baner distribuerer V+ (VDD) og 0 (VSS) til hhv. de høje og lave transistorer. Flere steder er banerne forbundet til store 'ører' på printets loddesside. Spændingsforsyningen kan passende loddess på to af øerne på loddessiden.

På printets loddesside går fire brede printbaner på tværs mellem de to rækker. De forbinder T1 med T4, T2 med T3, T5 med T8, og T6 med T7. Disse fire baner er tilslutningerne til motorerne. Motor 1 forbindes til banerne mellem T1/T4 og T2/T3. Motor 2 forbindes til banerne mellem T5/T8 og T6/T7.

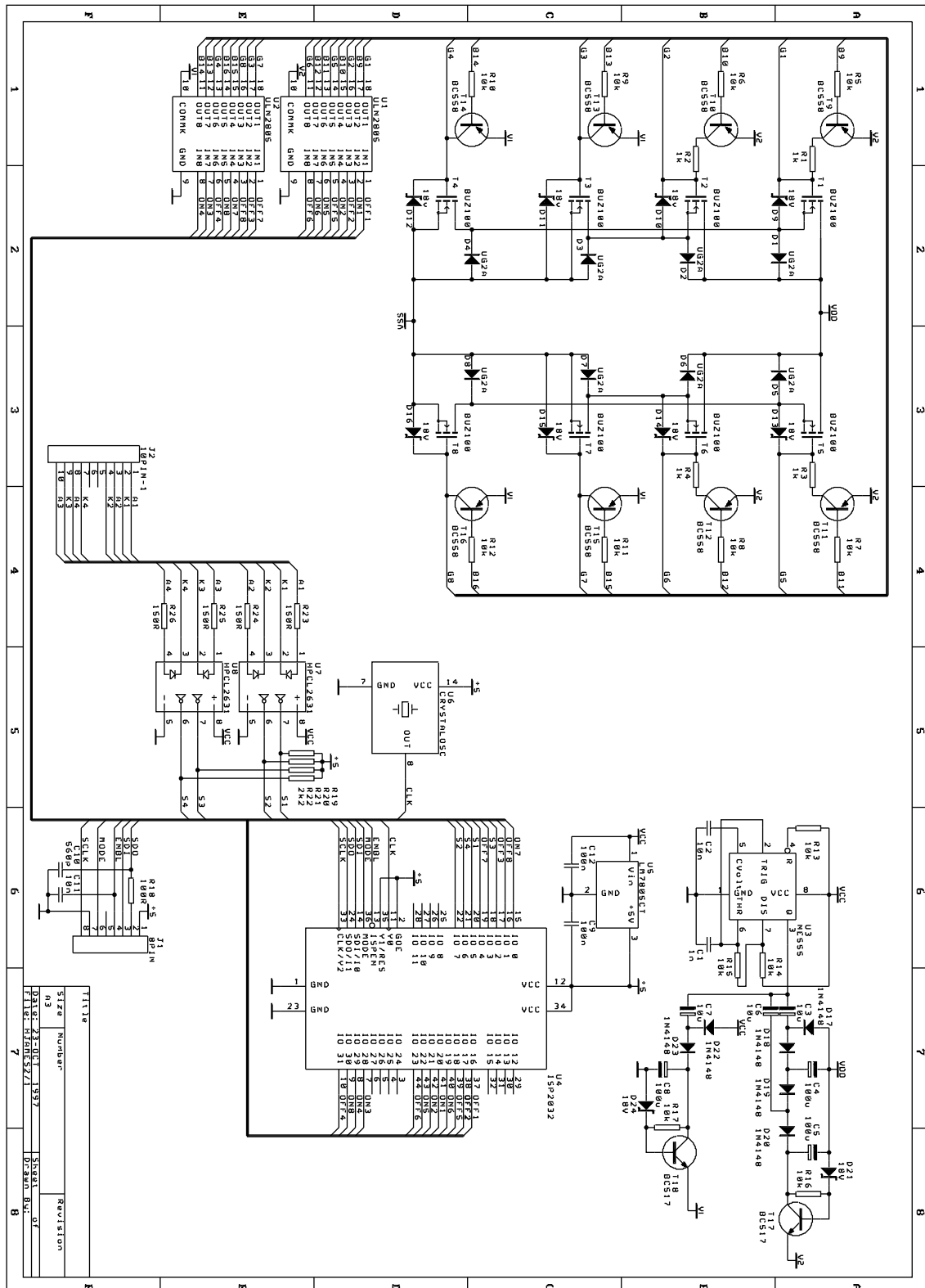
Forbindelsen til motor controller, varetages af et 10-polet MOLEX stik, J2.

Den programmerbare logikkreds ispLSI 1024 kan programmeres mens den sidder i opstillingen, via stikket J1. **Ved programmering af kredsen må KUN U4 spændingsforsynes. Spændingsforsyning til andre dele af kresløbet vil ødelægge MOS-FET transistorerne, eller gate-driverne**

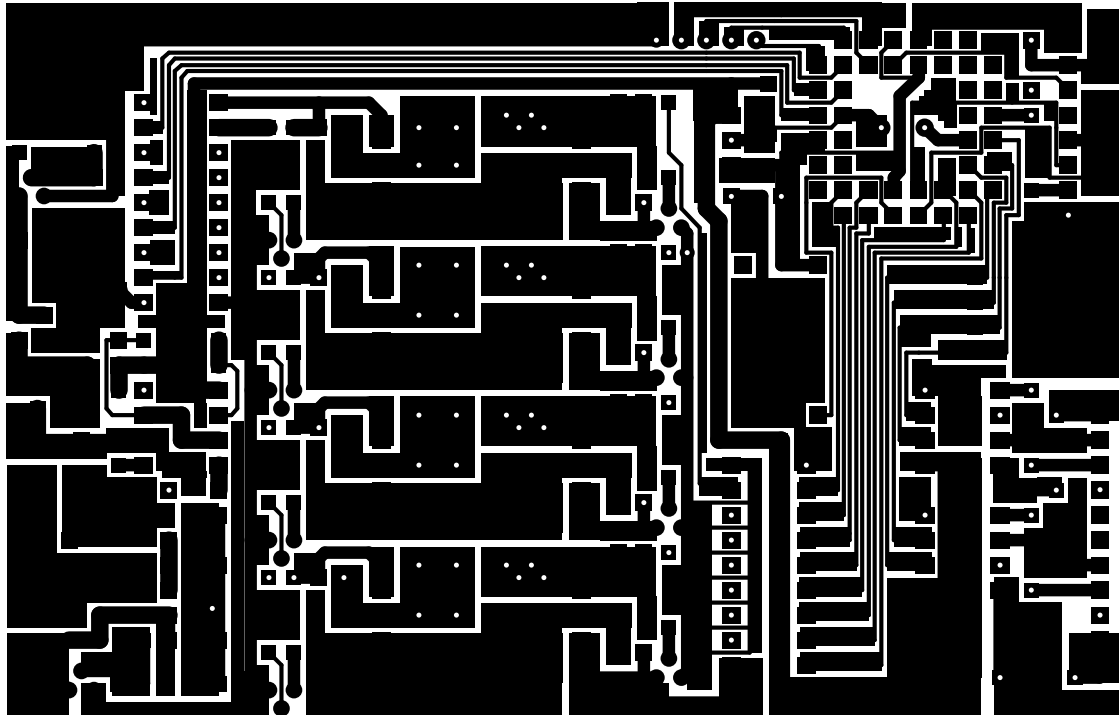


Figur 17.10: Stikforbindelser på James motordriver

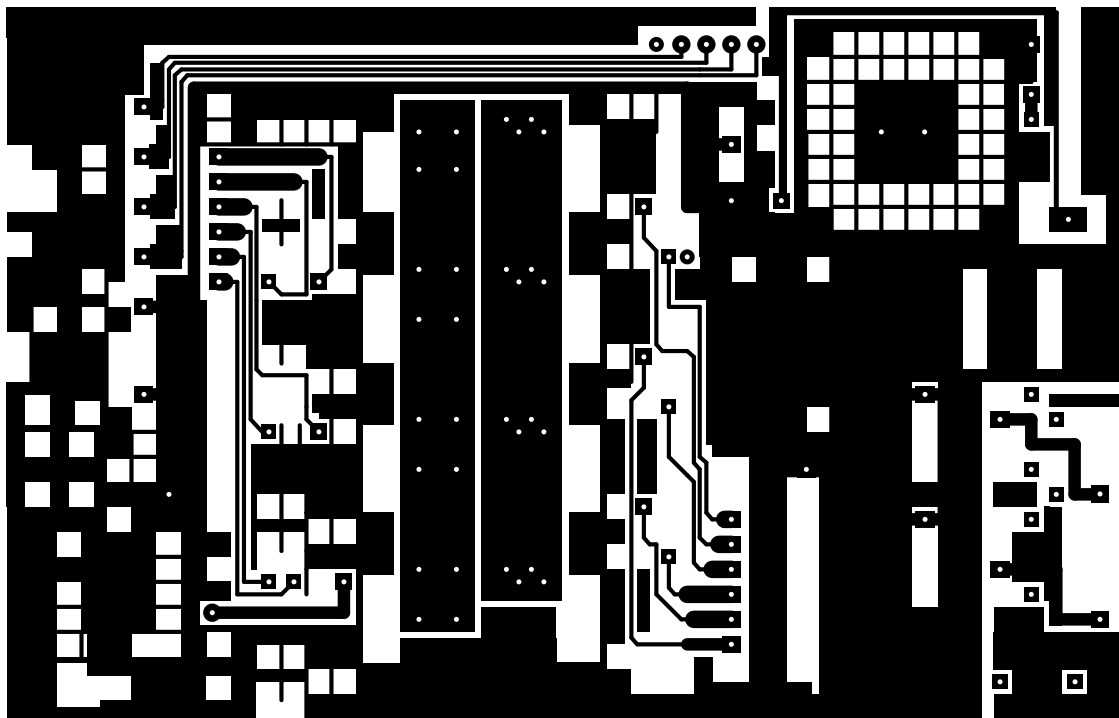
17.5.2 Diagram, printlayout, m.m.



Figur 17.11: Diagram over motordriver

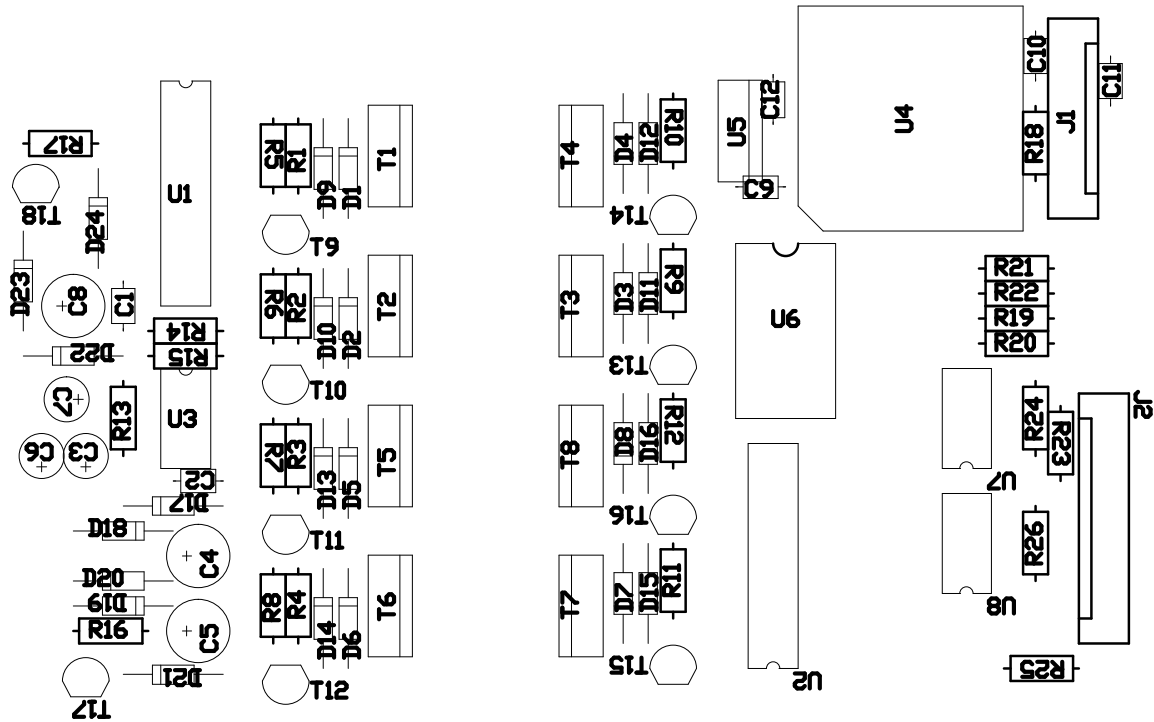


(a) Loddeside



(b) Komponentside

Figur 17.12: Printlayout til motordriver
167

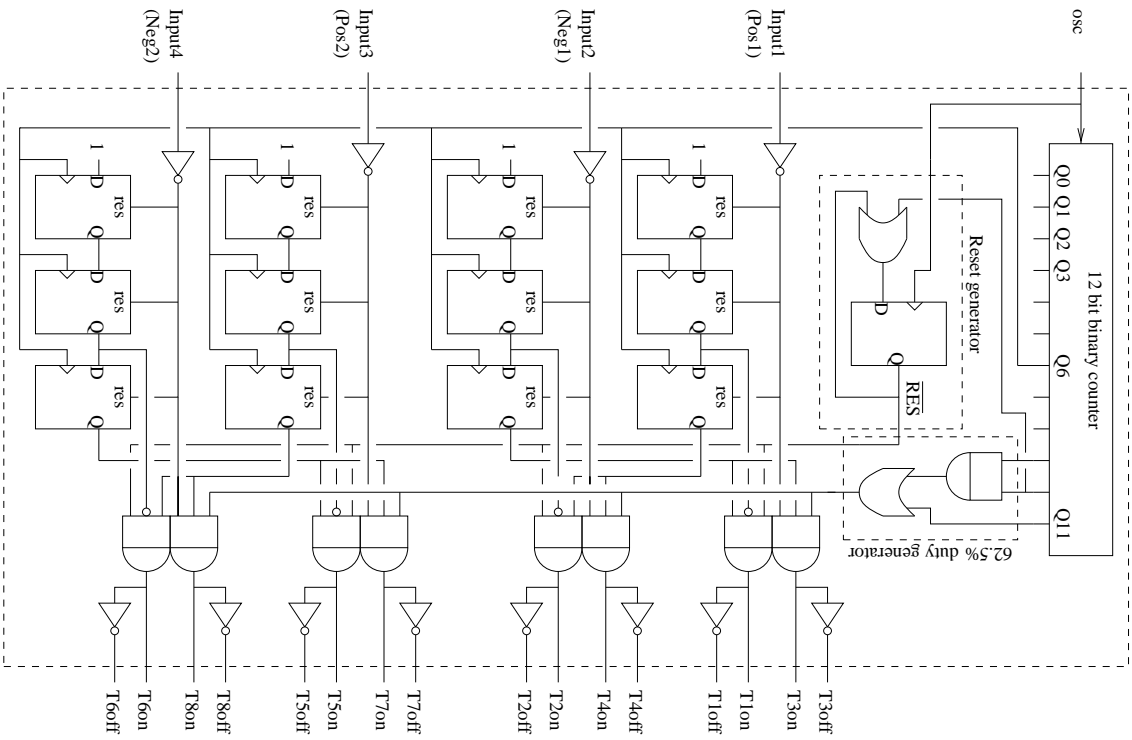


Figur 17.13: Komponentplacering på motordriver

J1	8 pol	MOLEX	C3	10μF	tantal	D21	18V Zener	1/2 W
J2	10 pol	MOLEX	C4	100μF	elektrolyt	D22	1N4148	small signal diode
R1	1kΩ	1/4 W	C5	100μF	elektrolyt	D23	1N4148	small signal diode
R2	1kΩ	1/4 W	C6	10μF	tantal	D24	18V Zener	1/2 W
R3	1kΩ	1/4 W	C7	10μF	tantal	T1	BUZ100	N-channel power MOS-FET
R4	1kΩ	1/4 W	C8	100μF	elektrolyt	T2	BUZ100	N-channel power MOS-FET
R5	10kΩ	1/4 W	C9	100nF	sibatit	T3	BUZ100	N-channel power MOS-FET
R6	10kΩ	1/4 W	C10	560pF	keramisk	T4	BUZ100	N-channel power MOS-FET
R7	10kΩ	1/4 W	C11	10nF	sibatit	T5	BUZ100	N-channel power MOS-FET
R8	10kΩ	1/4 W	C12	100nF	sibatit	T6	BUZ100	N-channel power MOS-FET
R9	10kΩ	1/4 W	D1	UG2A	2A ultra fast	T7	BUZ100	N-channel power MOS-FET
R10	10kΩ	1/4 W	D2	UG2A	2A ultra fast	T8	BUZ100	N-channel power MOS-FET
R11	10kΩ	1/4 W	D3	UG2A	2A ultra fast	T9	BC558	PNP small signal
R12	10kΩ	1/4 W	D4	UG2A	2A ultra fast	T10	BC558	PNP small signal
R13	10kΩ	1/4 W	D5	UG2A	2A ultra fast	T11	BC558	PNP small signal
R14	10kΩ	1/4 W	D6	UG2A	2A ultra fast	T12	BC558	PNP small signal
R15	10kΩ	1/4 W	D7	UG2A	2A ultra fast	T13	BC558	PNP small signal
R16	10kΩ	1/4 W	D8	UG2A	2A ultra fast	T14	BC558	PNP small signal
R17	10kΩ	1/4 W	D9	18V Zener	1/2 W	T15	BC558	PNP small signal
R18	100Ω	1/4 W	D10	18V Zener	1/2 W	T16	BC558	PNP small signal
R19	2.2kΩ	1/4 W	D11	18V Zener	1/2 W	T17	BC547	NPN small signal
R20	2.2kΩ	1/4 W	D12	18V Zener	1/2 W	T18	BC547	NPN small signal
R21	2.2kΩ	1/4 W	D13	18V Zener	1/2 W	U1	ULN2805	octal open collector driver
R22	2.2kΩ	1/4 W	D14	18V Zener	1/2 W	U2	ULN2805	octal open collector driver
R23	150Ω	1/4 W	D15	18V Zener	1/2 W	U3	NE555	general purpose timer
R24	150Ω	1/4 W	D16	18V Zener	1/2 W	U4	ispLSI1024	programmerbar logik
R25	150Ω	1/4 W	D17	1N4148	small signal	U5	LM7805	5V spændingsregulator
R26	150Ω	1/4 W	D18	1N4148	small signal	U6	1MHz osc	integreret krystaloscillator
C1	1nF	keramisk	D19	1N4148	small signal	U7	HPCL2631	dual high speed optocoupler
C2	10nF	sibatit	D20	1N4148	small signal	U8	HPCL2631	dual high speed optocoupler

Tabel 17.6: Komponentliste til motordriver

17.5.3 Programmerbar logik



Figur 17.14: Kredsløb i programmerbar logik

Ligninger

A:\HJTEST2.RPT generated using Lattice pDS Version 2.61

**** Håndredigeret for bedre læselighed ****

Part: ispLSI1016-80LJ44
Design Name: hjtest1
Design Revision: 0.1
Author: Anders Stengaard Sørensen
Project Name: James
Description: Styling af transistore i HBRO til James
Verify Status: Complete
Route Status: Complete
Data Used: Post-Route

External I/O Pin Report				
Pin Number	Pad Name	Fixed Before Route	Pin Type	Pullup
1	GND	Yes	Gnd	
7	XON3	Yes	Output	No
8	XON4	Yes	Output	No
9	XON8	Yes	Output	No
10	XOFF4	Yes	Output	Yes
11	XOSC	Yes	Input	No
12	VCC	Yes	Vcc	
15	XON7	Yes	Output	No
16	XOFF8	Yes	Output	Yes
17	XOFF3	Yes	Output	Yes
18	XINPUT3	Yes	Input	No
19	XOFF7	Yes	Output	Yes
20	XINPUT1	Yes	Input	No
21	XINPUT4	Yes	Input	No
22	XINPUT2	Yes	Input	No
23	GND	Yes	Gnd	
34	VCC	Yes	Vcc	
35	XRESET	No	Input	No
37	XOFF1	Yes	Output	Yes
38	XOFF2	Yes	Output	Yes
39	XOFF5	Yes	Output	Yes
40	XON6	Yes	Output	No

41	XON1	Yes	Output	No
42	XON2	Yes	Output	No
43	XON5	Yes	Output	No
44	XOFF6	Yes	Output	Yes

LDE Report

STATISTICS FOR GLB A0, USERNAME: CNT_A, ROUTED LOCATION: A7

GLB Input List:	GLB Output List:
I12 : Q3	O0 : Q3
I13 : Q2	O1 : Q2
I14 : Q1	O2 : Q1
I15 : Q0	O3 : Q0
CLK0 : OSC	
RESET : !RESET	

CELL EQUATIONS:

Q0.CLK = OSC; Q0.D = !Q0.Q;
Q1.CLK = OSC; Q1.D = (Q1.Q) \$\$ (Q0.Q);
Q2.CLK = OSC; Q2.D = (Q2.Q) \$\$ (Q1.Q & Q0.Q);
Q3.CLK = OSC; Q3.D = (Q3.Q) \$\$ (Q2.Q & Q1.Q & Q0.Q);

STATISTICS FOR GLB A1, USERNAME: CNT_B, ROUTED LOCATION: B0

GLB Input List:	GLB Output List:
I0 : Q0	O0 : Q6
I1 : Q1	O1 : Q7
I2 : Q2	O2 : Q4
I3 : Q3	O3 : Q5
I12 : Q5	
I13 : Q4	
I14 : Q7	
I15 : Q6	
CLK0 : OSC	
RESET : !RESET	

CELL EQUATIONS:

Q4.CLK = OSC; Q4.D = (Q4.Q) \$\$ (Q3 & Q2 & Q1 & Q0);
Q5.CLK = OSC; Q5.D = (Q5.Q) \$\$ (Q4.Q & Q3 & Q2 & Q1 & Q0);
Q6.CLK = OSC; Q6.D = (Q6.Q) \$\$ (Q5.Q & Q4.Q & Q3 & Q2 & Q1 & Q0);
Q7.CLK = OSC; Q7.D = (Q7.Q) \$\$ (Q6.Q & Q5.Q & Q4.Q & Q3 & Q2 & Q1 & Q0);

STATISTICS FOR GLB A2, USERNAME: CNT_C, ROUTED LOCATION: A6

GLB Input List:	GLB Output List:
I0 : Q6	O0 : Q11
I1 : Q7	O1 : Q10
I2 : Q4	O2 : Q9
I3 : Q5	O3 : Q8
I8 : Q11	
I9 : Q10	
I10 : Q9	
I11 : Q8	
I12 : Q3	
I13 : Q2	
I14 : Q1	
I15 : Q0	
CLK0 : OSC	
RESET : !RESET	

CELL EQUATIONS:

Q8.CLK = OSC; Q8.D = (Q8.Q) \$\$ (Q7 & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0);
Q9.CLK = OSC; Q9.D = (Q9.Q) \$\$ (Q8.Q & Q7 & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0);
Q10.CLK = OSC; Q10.D = (Q10.Q) \$\$ (Q9.Q & Q8.Q & Q7 & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0);
Q11.CLK = OSC; Q11.D = (Q11.Q) \$\$ (Q10.Q & Q9.Q & Q8.Q & Q7 & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0);

STATISTICS FOR GLB A3, USERNAME: RESET, ROUTED LOCATION: B5

GLB Input List:	GLB Output List:
I5 : Q9	O0 : NRESET
I6 : Q10	O1 : CHOP
I7 : Q11	O2 : CHOP_1
I11 : NRESET	
CLK0 : OSC	
RESET : !RESET	

CELL EQUATIONS:

CHOP_1.CLK = OSC; CHOP_1.D = (Q9 & Q10) # (Q11);
CHOP.CLK = OSC; CHOP.D = (Q9 & Q10) # (Q11);
NRESET.CLK = OSC; NRESET.D = !(NRESET.Q & !Q10);

STATISTICS FOR GLB A4, ROUTED LOCATION: B4

GLB Input List:	GLB Output List:
I6 : MMV1_C	O0 : T3
I8 : INPUT2	O1 : T4
I9 : CHOP_1	O3 : T4_1
I10 : INPUT1	
I12 : MMV2_C	

CELL EQUATIONS:

T4_1 = MMV1_C & !INPUT2 & CHOP_1;
T4 = MMV1_C & !INPUT2 & CHOP_1;
T3 = MMV2_C & CHOP_1 & !INPUT1;

```

STATISTICS FOR GLB A5, ROUTED LOCATION: A1
*****
GLB Input List:      GLB Output List:
I3   : INPUT3        O0   : T7
I5   : CHOP          O1   : T8ECHO
I6   : INPUT4
I11  : MMV3_C
I14  : MMV4_C

CELL EQUATIONS:
-----
T8ECHO = MMV3_C & !INPUT4 & CHOP;
T7 = MMV4_C & CHOP & !INPUT3;

*****
STATISTICS FOR GLB A4_part1, ROUTED LOCATION: A4
*****
GLB Input List:      GLB Output List:
I3   : MMV2_C        O2   : T3ECHO
I5   : INPUT1
I6   : CHOP_1

CELL EQUATIONS:
-----
T3ECHO = MMV2_C & CHOP_1 & !INPUT1;

*****
STATISTICS FOR GLB A5_part1, ROUTED LOCATION: B7
*****
GLB Input List:      GLB Output List:
I4   : MMV3_C        O2   : T8
I9   : INPUT4
I10  : CHOP

CELL EQUATIONS:
-----
T8 = MMV3_C & !INPUT4 & CHOP;

*****
STATISTICS FOR GLB B4, USERNAME: MMV4, ROUTED LOCATION: B3
*****
GLB Input List:      GLB Output List:
I2   : MMV4_B        O0   : MMV4_A
I3   : MMV4_A        O1   : MMV4_B
I4   : MMV3_C        O2   : MMV4_C
I9   : INPUT4        O3   : T6
I11  : NRESET
CLK1 : Q6
RESET : !RESET

CELL EQUATIONS:
-----
MMV4_A.CLK = Q6;      MMV4_A.D = VCC;
MMV4_B.CLK = Q6;      MMV4_B.D = MMV4_A.Q;
MMV4_C.CLK = Q6;      MMV4_C.D = MMV4_B.Q;
MMV4_A.RE = !INPUT4;  T6 = !MMV4_B.Q & MMV3_C & NRESET;
MMV4_B.RE = !INPUT4;
MMV4_C.RE = !INPUT4;

*****
STATISTICS FOR GLB B5, USERNAME: MMV3, ROUTED LOCATION: B6
*****
GLB Input List:      GLB Output List:
I1   : MMV4_C        O0   : MMV3_A
I6   : MMV3_B        O1   : MMV3_B
I7   : MMV3_A        O2   : T5
I11  : NRESET        O3   : MMV3_C
I12  : INPUT3
CLK1 : Q6
RESET : !RESET

CELL EQUATIONS:
-----
MMV3_A.CLK = Q6;      MMV3_A.D = VCC;
MMV3_B.CLK = Q6;      MMV3_B.D = MMV3_A.Q;
MMV3_C.CLK = Q6;      MMV3_C.D = MMV3_B.Q;
MMV3_A.RE = !INPUT3;  T5 = !MMV3_B.Q & MMV4_C & NRESET;
MMV3_B.RE = !INPUT3;
MMV3_C.RE = !INPUT3;

*****
STATISTICS FOR GLB B6, USERNAME: MMV2, ROUTED LOCATION: B1
*****
GLB Input List:      GLB Output List:
I6   : MMV1_C        O1   : T2
I8   : INPUT2        O2   : MMV2_B
I9   : MMV2_B
I11  : NRESET
I15  : MMV2_A
CLK1 : Q6
RESET : !RESET

CELL EQUATIONS:
-----
MMV2_B.CLK = Q6;      MMV2_B.D = MMV2_A;
MMV2_B.RE = !INPUT2;  T2 = !MMV2_B.Q & MMV1_C & NRESET;

*****
STATISTICS FOR GLB B7, USERNAME: MMV1, ROUTED LOCATION: B2
*****
GLB Input List:      GLB Output List:
I4   : MMV1_A        O0   : T1
I5   : MMV1_B        O1   : MMV1_C
I10  : INPUT1        O2   : MMV1_B
I11  : NRESET        O3   : MMV1_A
I12  : MMV2_C
CLK1 : Q6

```

```

RESET : !RESET

CELL EQUATIONS:
-----
MMV1_A.CLK = Q6;          MMV1_A.D = VCC;
MMV1_B.CLK = Q6;          MMV1_B.D = MMV1_A.Q;
MMV1_C.CLK = Q6;          MMV1_C.D = MMV1_B.Q;
MMV1_A.RE = !INPUT1;      T1 = !MMV1_B.Q & MMV2_C & NRESET;
MMV1_B.RE = !INPUT1;
MMV1_C.RE = !INPUT1;

*****
STATISTICS FOR GLB B6_part1, USERNAME: MMV2, ROUTED LOCATION: A0
*****
GLB Input List:          GLB Output List:
I6 : MMV2_B              O0 : MMV2_A
I7 : INPUT2              O3 : MMV2_C
CLK1 : Q6
RESET : !RESET

CELL EQUATIONS:
-----
MMV2_C.CLK = Q6;          MMV2_A.D = VCC;
MMV2_A.CLK = Q6;          MMV2_C.D = MMV2_B;
MMV2_C.RE = !INPUT2;
MMV2_A.RE = !INPUT2;

*****
STATISTICS FOR IO CELLS
*****
I/O      User
cell     name
Input List:  Output List:      CELL EQUATIONS:
-----
I00      PAD : XINPUT1      OUT : INPUT1      XPIN IO XINPUT1 LOCK 20;      IB11(INPUT1, XINPUT1);
I01      PAD : XINPUT2      OUT : INPUT2      XPIN IO XINPUT2 LOCK 22;      IB11(INPUT2, XINPUT2);
I02      PAD : XINPUT3      OUT : INPUT3      XPIN IO XINPUT3 LOCK 18;      IB11(INPUT3, XINPUT3);
I03      PAD : XINPUT4      OUT : INPUT4      XPIN IO XINPUT4 LOCK 21;      IB11(INPUT4, XINPUT4);
I016     on8 IMUX : T8       PAD : XON8       XPIN IO XON8 LOCK 9;          OB11(XON8, T8);
I017     off8 IMUX : T8ECHO  PAD : XOFF8     XPIN IO XOFF8 LOCK 16 PULLUP; OB21(XOFF8, T8ECHO);
I018     on7 IMUX : T7       PAD : XON7       XPIN IO XON7 LOCK 15;          OB11(XON7, T7);
I019     off7 IMUX : T7       PAD : XOFF7     XPIN IO XOFF7 LOCK 19 PULLUP; OB21(XOFF7, T7);
I020     on6 IMUX : T6       PAD : XON6       XPIN IO XON6 LOCK 40;          OB11(XON6, T6);
I021     off6 IMUX : T6       PAD : XOFF6     XPIN IO XOFF6 LOCK 44 PULLUP; OB21(XOFF6, T6);
I022     on5 IMUX : T5       PAD : XON5       XPIN IO XON5 LOCK 43;          OB11(XON5, T5);
I023     off5 IMUX : T5       PAD : XOFF5     XPIN IO XOFF5 LOCK 39 PULLUP; OB21(XOFF5, T5);
I024     on4 IMUX : T4       PAD : XON4       XPIN IO XON4 LOCK 8;          OB11(XON4, T4);
I025     off4 IMUX : T4_1    PAD : XOFF4     XPIN IO XOFF4 LOCK 10 PULLUP; OB21(XOFF4, T4_1);
I026     on3 IMUX : T3       PAD : XON3       XPIN IO XON3 LOCK 7;          OB11(XON3, T3);
I027     off3 IMUX : T3ECHO  PAD : XOFF3     XPIN IO XOFF3 LOCK 17 PULLUP; OB21(XOFF3, T3ECHO);
I028     on2 IMUX : T2       PAD : XON2       XPIN IO XON2 LOCK 42;          OB11(XON2, T2);
I029     off2 IMUX : T2       PAD : XOFF2     XPIN IO XOFF2 LOCK 38 PULLUP; OB21(XOFF2, T2);
I030     on1 IMUX : T1       PAD : XON1       XPIN IO XON1 LOCK 41;          OB11(XON1, T1);
I031     off1 IMUX : T1       PAD : XOFF1     XPIN IO XOFF1 LOCK 37 PULLUP; OB21(XOFF1, T1);
RESET    PAD : !XRESET      OUT : !XRESET    XPIN RST !XRESET LOCK 35;      IB11(!RESET, !XRESET);
Y0       PAD : XOSC          OUT : OSC        XPIN CLK XOSC LOCK 11;        IB11(OSC, XOSC);

```

17.6 Computer

Som styrecomputer anvendes en 16/24 bits VME computer, opbygget i et åbent, afkortet 19" subrack.

Backplane: 7 slots S1 VME-backplane fra VERO.

CPU modul: VMPPM KC2, 68020 baseret CPU fra PEP. Se [29]

Disk controller: VMSC VME disk controller fra PEP.

Floppy drev: 720Kb standard 3.5" floppy drev.

Analog indgange: Analog Devices RTI-600, med 16 12 bits analoge indgange (multiplexede), fra Analog devices. Se [22]

Srømforsyning: +5V, +12V, og -12V, fra james strømforsyning.

17.6.1 Konfiguration af backplane

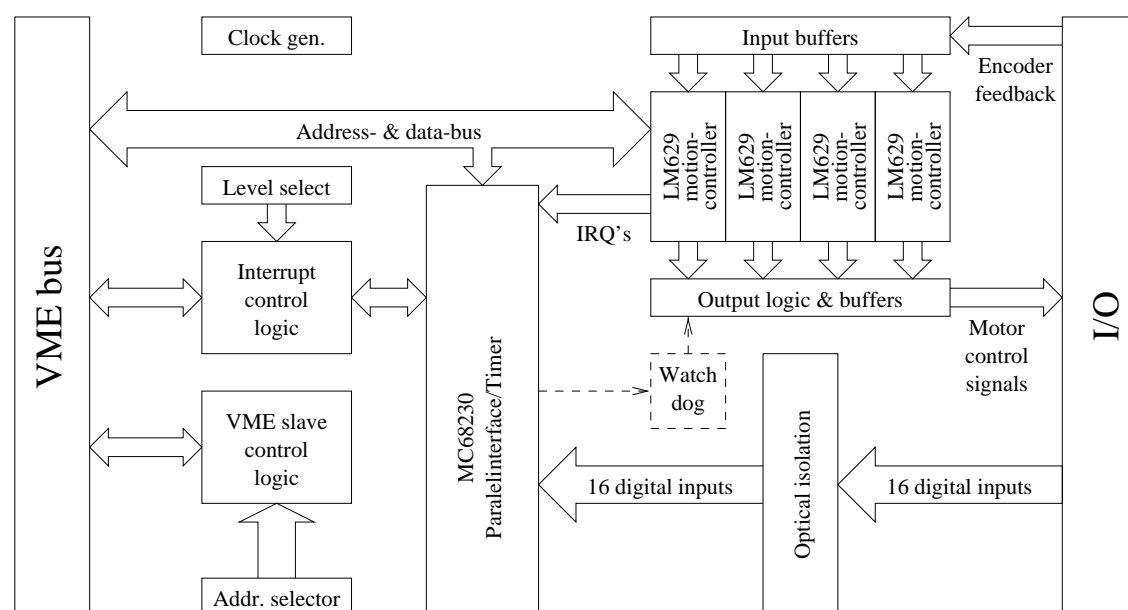
Da der kun er en *master* på bussen (CPU kortet), skal *bus request/ bus grant* jumperne alle være isat.

Alle *interrupt acknowledge* skal sidde i, undtaget ved de VME kort der anvender interrupts. Dvs. de skal være fjernet ud for det *slot* der rummer diskcontrolleren.

Kapitel 18

Dokumentation af Gefion

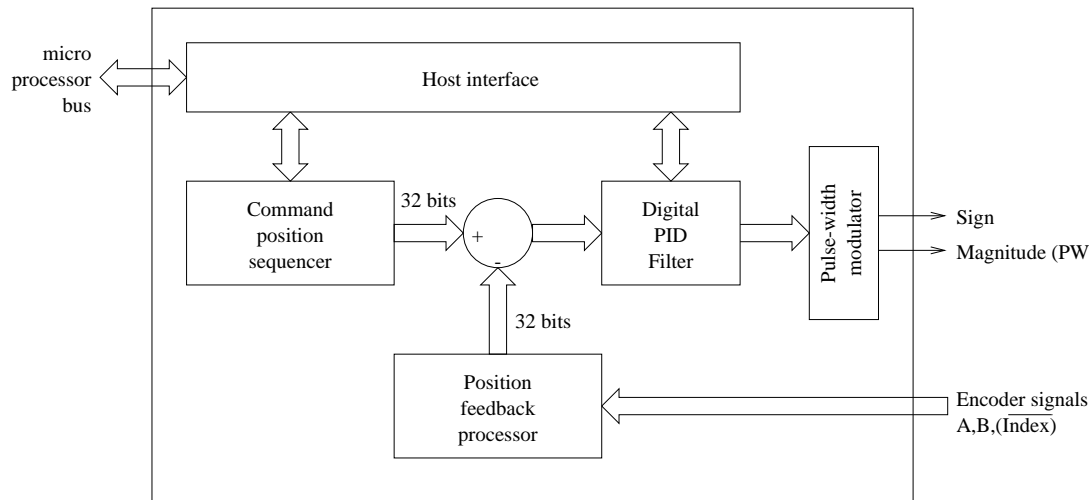
18.1 Funktionel beskrivelse



Figur 18.1: Blokdiagram over Gefion

18.1.1 LM629 motion controller

LM629 er en dedikeret motion-control processor, designet til brug med forskellige DC-motorer, og andre servomekanismer med inkrementale kvadratur enkoder signaler som feedback.



Figur 18.2: Blokdiagram over LM629

LM629 interfacer via sit *host interface* til en synkron 8-bits INTEL kompatibel bus. Via *host interfacet* kan man programmere PID filteret, og *command position sequenceren*. *Position feedback processoren* holder vha. signaler fra en inkrementalenkoder rede på motorens position. *Command position controlleren* er en S-kurve generator, der genererer kontinuerte, 2 gange differentiable positionskurver, som funktion af tiden. Kurverne genereres ud fra oplysninger om endelig position, tophastighed, og acceleration. PID filteret fødes kontinuert med forskellen mellem den øjeblikkeligt ønskede position, og den målte funktion. Outputtet fra filteret er givet ved:

$$u(n) = kp \cdot e(n) + ki \sum_{N=0}^n e(n) + kd \cdot (e(n') - e(n' - 1)) \quad (18.1)$$

Hvor $u(n)$ er output fra filteret ved *sample* n , $e(n)$ er fejlen ved *sample* n , $e(n')$ er fejlen ved *sample* n' , og $e(n' - 1)$ er fejlen ved forrige *sample*. Til brug i differentiationsleddet kan fejlen samples ved en lavere frekvens end den resten af systemet anvender. Disse sampleværdier symboliseres med $e(n')$, $e(n' - 1)$ etc.

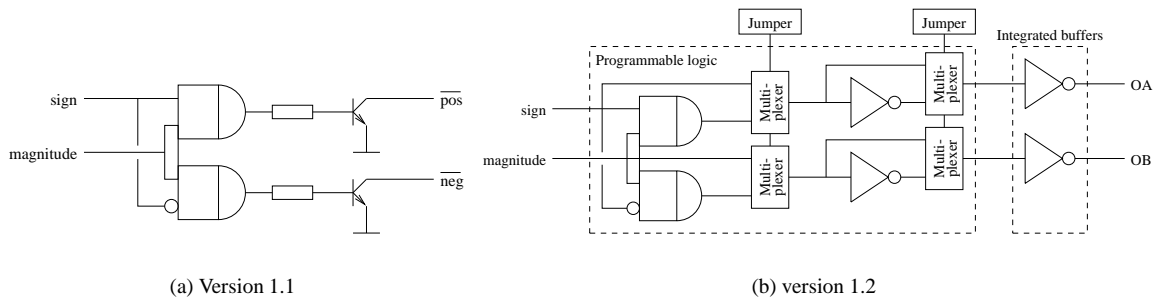
Outputtet fra filteret er et 8 bits tal med fortegn. Fortegnet sendes ud på en separat udgang, mens de resterende 7 bits, omsættes til et pulsbredde moduleret digitalt signal. Frekvensen for signalet er $f_{clk}/512$, altså ca. 12kHz for et 6MHz clock signal. Eftersom det pulsviddemodulerede signal genereres ud fra et 7 bits signal, er der tale om 128 diskrete pulsbredder, hvilket giver signalet en båndbredde på $f_{clk}/4$, eller 1.5MHz, ved en clock frekvens på 6MHz.

18.1.2 Input buffere

Feedback signalerne fra inkrementalenkoderne er ikke galvanisk adskilt fra Gefions kredsløb, men passerer en alm. digital buffer, der i nogen grad beskytter LM629 kredsene mod fejltilslutning m.m. af enkoderne.

18.1.3 Output logik & buffere

Det digitale pulsvidde modulerede signal **magnitude** og det digitale retningssignal **sign** omsættes til to gensidigt udelukkende pulsviddemodulerede signaler, der koder for positiv og negativ omløbsretning af motoren. (På V. 1.2 kan outputlogikken omkonfigureres så signalerne passerer uændret). Signalerne passerer derefter en buffer, beregnet på at drive optisk isolerede indgange.



Figur 18.3: Skitse af outputlogik og buffere

18.1.4 Watch dog

V. 1.2 er udstyret med en *watch-dog* der skal aktiveres af en udgang fra MC68230 kredsen med jævne mellemrum, for at aktivere motorudgangene.

18.1.5 MC68230 PI/T

Gefion er udstyret med en MC68230 Parallel interface/Timer kreds, der anvendes som interrupt controller for de fire LM629 kredse, og hvis parallelporte anvendes til at give Gefion 16 digitale indgange. På V. 1.2 bruges MC68230eren også til at aktivere *watch-dog* timeren.

18.1.6 Optisk isolering

Gefion har 16 digitale indgange, der er galvanisk isoleret fra signalkilden vha. optokoblere placeret på selve Gefion.

18.1.7 Adressevælger

Gefion har et adresserum på 256 bytes, der vha. 16 dip-switches kan placeres overalt i VME bussens 16 MB store område, med spring på 256 bytes.

18.1.8 VME slave kontrollogik

Gefion er et standard VME slave modul. Slave kontrollogikken danner interfacet mellem VME-bussen, og Gefions periferikredse. Slave kontrol logikken rummer buffere, adressedekoder, DTACK generator mv. Slave kontrol logikken er hovedsageligt implementeret vha. programmerbar logik.

18.1.9 Niveauvælger

Gefion kan generere interrupts på alle VME bussens 7 niveauer. Hvilket niveau der anvendes, kan indstilles vha. 3 dip-switches.

18.1.10 Interrupt kontrollogik

Interrupt kontrollogikken danner interfacet mellem MC68230 PI/T kredsens *interrupt request* udgange, dens *interrupt acknowledge* indgange, og VME-bussen. Interrupt kontrollogikken er hovedsageligt implementeret vha. programmerbar logik.

18.1.11 Clock generator

De anvendte LM629 motorcontrollere, er 6MHZ versionen, der kan anvende klokfrekvenser fra 1MHz til 6MHz. Gefion er udstyret med en separat 12MHz krystaloscillator, med efterfølgende frekvensdeler, der genererer en 50% duty-cycle 6MHz frekvens. Hvis andre frekvenser end 6MHz ønskes, kan krystallet udskiftes.

18.2 Hardware setup

En række aspekter af Gefions funktioner skal indstilles hardwaremæssigt. **Bemærk at der er markante forskelle mellem V. 1.1 og V. 1.2!**

18.2.1 Adresseringsmetode og basisadresse

V. 1.1 kan adresseres vha. 16 adressebits (short) eller 24 adressebits (standard), mens V. 1.2 kun kan adresseres vha. 24 adressebits (standard).

På V. 1.1 vælges *short* ved at kortslutte ben 1 og 2 på S3, mens *standard* vælges ved at kortslutte ben 2 og 3. **OBS! pga. en fejl i printlayoutet til V. 1.1 er der to komponenter der hedder S3. En 3 pins jumper, og en 4 pols DIP switch. *short/standard* vælges vha. 3 pins jumperen**

Uanset om der anvendes *short* eller *standard* adressering, fylder begge versioner af Gefion 256 bytes i adresserummet, og okkuperer således adresserne fra $hhl00_{16}$ til $hhlFF_{16}$ ($ll00_{16}$ til $llFF_{16}$ ved *short*) på VME bussen. Basisadressen: hhl_{16} vælges vha S1 og S2, hvor S1 vælger de 8 bits i *ll* og S2 de 8 bits i *hh*. Ved *short* adressering anvendes S2 ikke. På V. 1.1 svarer en kontakt i S1 og S2 til at den pågældende bit er sat til 0, mens en tændt kontakt på V. 1.2 svarer til en bit sat til 1.

På PEP VMPM68KC CPU modulet, mappes VME bussens adresseområde ind i adresseområdet: $87000000_{16} \dots 87FFFF_{16}$ ved *standard* adressering, og $85000000_{16} \dots 8500FFFF_{16}$ ved *short*.

18.2.2 Interruptniveau

Gefion kan generere interrupts på alle VME-bussens 7 niveauer. Interruptniveauet indstilles som binært tal, på DIP-switchen S3's kontakt 1,2, og 3, der svarer til hhv. bit 0,1, og 2. Niveauet stilles fra 0 til 7. Ved 0 er interrupts afbrudt. På V. 1.1 sættes en bit til 0, vha. en tændt kontakt. På V. 1.2 sættes en bit til 0 ved en slukket kontakt.

Kontakt nr. 4 leder VME-bussens IACK signal uden om Gefions interrupt kontrol logik, og skal være slukket hvis der anvendes interrupts.

18.2.3 Output konfiguration

På V. 1.1 er det ikke muligt at ændre output konfigurationen. V. 1.1's udgange er altid konfigureret som to aktivt lave, open collector udgange til hver motor. En der signalerer positiv omløbsretning, og en der

signalerer negativ omløbsretning. Begge signaler er pulsvidde modulerede, hvor pulsvidden koder for energitilførslen til motoren i den givne omløbsretning.

Som standard er V. 1.2 konfigureret som V. 1.1. På V. 1.2 er udgangsbufferene implementeret vha. en enkelt 74LS641-1 IC monteret i sokkel. Der findes en lang række benkompatible bufferkredse i 74 serien, og kredsen kan derfor udskiftes med en kreds der har den ønskede type udgange.

Polariteten af udgangene kan konfigureres vha. jumperen J1. Kortsluttes J1, er udgangene aktivt høje, afbrydes den er de aktivt lave.

Betydningen af udgangene vælges vha jumperen J2. Er jumperen afbrudt, betyder de to pulsvidde modulerede udgange for hver motor hhv. positiv og negativ omløbsretning. Kortsluttes jumperen koder den ene udgang (A) for motorens retning, mens den anden udgang B er et pulsvidde moduleret signal, der koder for energitilførslen til motoren.

18.2.4 Clock frekvens

På begge versioner kan clock frekvensen til LM629 kredsene konfigureres ved at skifte krystallet på kortet. Clockfrekvensen til LM629'erne er det halve af krystallets påtrykte frekvens. Med LM629-6 kan der anvendes krystaller i området: $2 \dots 12 MHz$.

Clockfrekvensen til MC68230 er fast på 8MHZ, der afledes fra VME-bussens clockfrekvens.

18.3 Tilslutning

Tilslutningen til VME bussen sker ved at montere Gefion i et 19 tommer rack med VME backplane. Hvis Gefions interrupts skal anvendes skal man huske at fjerne IACK jumperen fra den slot i backplanet Gefion monteres i.

Tilslutningen til motorer, enkodere, og digitale signaler sker via det 50 polede IDC stik CON2.

pin	signal	pin	signal
1	B2	2	A1
3	A2	4	$\overline{\text{INDEX1}}$
5	$\overline{\text{INDEX2}}$	6	B1
7	B4	8	$\overline{\text{INDEX3}}$
9	A4	10	A3
11	$\overline{\text{INDEX4}}$	12	B3
13	GND	14	GND
15	$\overline{\text{POS2}}$	16	+5V
17	$\overline{\text{NEG3}}$	18	+5V
19	$\overline{\text{NEG4}}$	20	+5V
21	$\overline{\text{NEG1}}$	22	+5V
23	$\overline{\text{POS3}}$	24	+5V
25	$\overline{\text{NEG2}}$	26	+5V
27	$\overline{\text{POS4}}$	28	+5V
29	$\overline{\text{POS1}}$	30	+5V
31	$\overline{\text{IPB2}}$	32	$\overline{\text{IPB1}}$
33	$\overline{\text{IPB0}}$	34	$\overline{\text{IPB3}}$
35	$\overline{\text{IPB7}}$	36	COMB 0-3
37	$\overline{\text{IPB5}}$	38	$\overline{\text{IPB6}}$
39	$\overline{\text{IPB4}}$	40	COMB4-7
41	$\overline{\text{IPA5}}$	42	$\overline{\text{IPA4}}$
43	$\overline{\text{IPA7}}$	44	$\overline{\text{IPA6}}$
45	$\overline{\text{IPA2}}$	46	$\overline{\text{IPA0}}$
47	$\overline{\text{IPA0}}$	48	COMA 0-3
49	$\overline{\text{IPA1}}$	50	$\overline{\text{IPA3}}$

(a) V.1.1

pin	signal	pin	signal
1	B2	2	A1
3	A2	4	$\overline{\text{INDEX1}}$
5	$\overline{\text{INDEX2}}$	6	B1
7	B4	8	$\overline{\text{INDEX3}}$
9	A4	10	A3
11	$\overline{\text{INDEX4}}$	12	B3
13	GND	14	GND
15	OA2	16	+5V
17	OB3	18	+5V
19	OB4	20	+5V
21	OB1	22	+5V
23	OA3	24	+5V
25	OB2	26	+5V
27	OA4	28	+5V
29	OA1	30	+5V
31	$\overline{\text{IPB2}}$	32	$\overline{\text{IPB1}}$
33	$\overline{\text{IPB0}}$	34	$\overline{\text{IPB3}}$
35	$\overline{\text{IPB7}}$	36	COMB 0-3
37	$\overline{\text{IPB5}}$	38	$\overline{\text{IPB6}}$
39	$\overline{\text{IPB4}}$	40	COMB4-7
41	$\overline{\text{IPA5}}$	42	$\overline{\text{IPA4}}$
43	$\overline{\text{IPA7}}$	44	$\overline{\text{IPA6}}$
45	$\overline{\text{IPA2}}$	46	$\overline{\text{IPA0}}$
47	$\overline{\text{IPA0}}$	48	COMA 0-3
49	$\overline{\text{IPA1}}$	50	$\overline{\text{IPA3}}$

(b) V.1.2

Figur 18.4: Tilslutning til Gefion (CON2)

Signalerne kan opdeles i tre grupper:

Enkodere: Forbindes til ben 1 ... 12

Motordrivere: Forbindes til de ulige ben 15 ... 29

Digitale signaler: Forbindes til ben 31 ... 50

Det anbefales at lade et kort 50 polet fladkabel forbinde Gefion med et *interconnect board*, der via passende stik forbindes til motordrivere, enkodere, etc. Interconnectboardet kan også rumme evt. formodstande, buffere, optokoblere, etc.

18.3.1 Forskelle mellem V.1.1 og V. 1.2

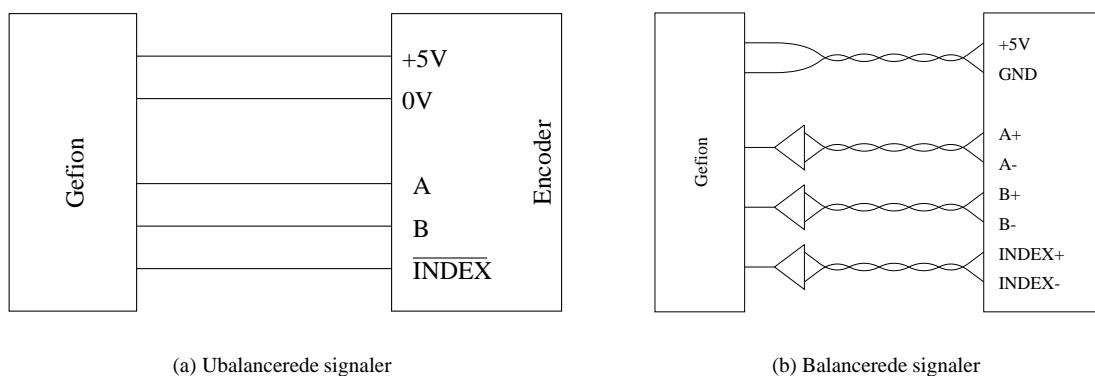
Den eneste tilslutningsmæssige forskel mellem V. 1.1 og V. 1.2 er motorudgangene (ulige ben: 15-29). På V. 1.1 har udgangene er fast funktion og polaritet, og er navngivet derefter. På V. 1.2 kan polaritet og funktion konfigureres vha. jumpers. Derfor er udgangene navngivet anderledes. Som udgangspunkt er V. 1.2's konfiguration identisk med V. 1.1's, hvorfor tilslutningen til de to versioner som udgangspunkt også er identiske.

18.3.2 Tilslutning af enkodere

For hver af de 4 akser Gefion kan styre, er der 3 enkoderindgange: **A**, **B**, og **INDEX**. Alle indgangene er TTL kompatible, og har en 4.7 k Ω pullup-modstand, hvorfor de kan drives af både TTL kompatible udgange, og NPN open-collector udgange.

Hvis den anvendte enkoder ikke har en **INDEX** udgang, eller hvis den ikke benyttes, skal **INDEX** indgangen på Gefion være uforbundet, eller tøjret til +5V via en pullup modstand.

Hvis den anvendte enkoder er forsynet med **INDEX** udgang, så vær sikker på at index funktionen for enkoderen svarer overens med LM629's specifikationer.



Figur 18.5: Forslag til tilslutning af enkodere

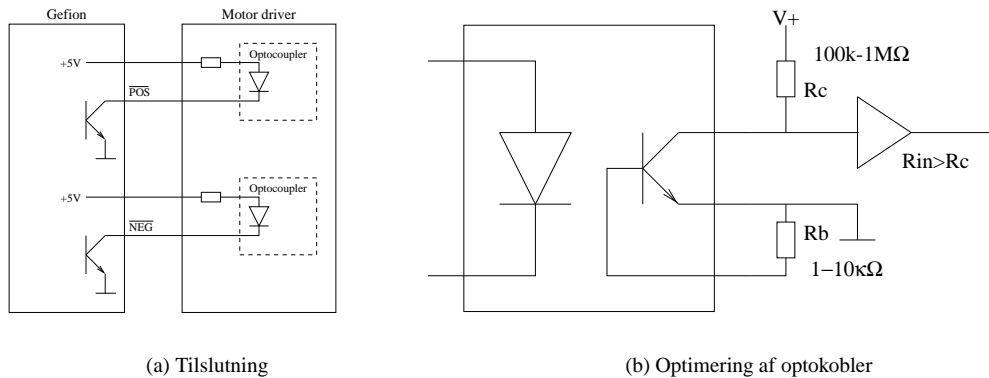
Hvis enkoderen forsynes med 5V, kan forsyningen tages fra VME bussen, via Gefion, hvilket giver den fordel at Gefion, og bussen kan holdes galvanisk adskilt fra andre kredsløb, hvis dette ønskes.

Figur: 18.5 viser to forslag til tilslutning af enkodere. Delfigur a, viser tilkobling af en enkoder med ubalancerede udgange. Denne metode er udmærket hvor der ikke stilles store krav til støjimmunitet. Arbejdes der i miljøer med store elektromagnetiske støjniveauer, vil det være en fordel at anvende enkodere med balancerede udgange, som vist i delfigur b. Signalerne fra enkoderen sendes via parsnoet kabel, til et sæt balancerede modtagere, der omsætter de balancerede signaler til TTL signaler der kan bruges af Gefion. De balancerede modtagere skal placeres tæt ved Gefion, for at undgå indkobling af støj på de ubalancerede siagneler fra de balancerede modtagere til Gefion.

Arbejdes der i miljøer med meget elektromagnetisk støj, er det ikke nødvendigvis en god ide' at forsyne enkoderen med strøm fra Gefion, og ovenstående figur skal kun opfattes som vejledende.

18.3.3 Tilslutning af motor-drivere

For at kunne holde Gefion og VME bussen galvanisk adskilt fra effektkredsløbene i en motordriver, er de otte motorudgange beregnet til at drive lysdioderne i optokoblere. Udgangene er alle *open collector*, der kan trække op til 50mA. Ved at bruge +5V fra Gefion, kan udgangene trække en enkelt eller evt. to seriekoblede lysdioder, med passende formodstande.



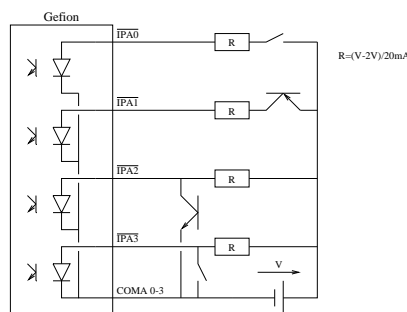
Figur 18.6: Forslag til tilslutning af motordrivere

Som standard er Gefion udstyret med et 12MHz krystal, hvilket giver de pulsvide modulerede signaler en frekvens på ca. 12kHz. LM629 kan generere 128 diskrete pulsbrejder, så den reelle båndbredde af signalet er ca. 1.5MHz. Passerer de pulsvidemodulerede signaler gennem kredsløb med lavere båndbredde, f.eks. langsomme optokoblere, kan det give anledning til reduceret opløsning, og ulineær overførsel af pulsbredden.

Figur 18.6(a) viser et eksempel på tilslutning af motordriver. Figur 18.6(b) skitserer hvordan en *low-cost* optokobler med typisk båndbredde på et par kHz, kan optimeres til en båndbredde op til flere hundrede gange større. Figuren er vejledende, og kredsløbet skal dimensioneres efter den optokobler der anvendes. Alternativt kan anvendes en decideret *high speed* optokobler.

18.3.4 Tilslutning af digitale signaler

Gefion har 16 optisk isolerede digitale indgange. Indgangene er grupperet i 4 grupper a' fire indgange. Hver gruppe har fælles katode.



Figur 18.7: Forslag til tilslutning af digitale indgange

Figur 18.7 viser fire forslag til hvordan indgangene kan tilsluttes, med kontakter og *open collector* NPN eller PNP koblinger.

18.4 Adressering

Tabel 18.1 viser blokopdelingen af Heimdals adresserum. Bemærk at bit 7 ikke bruges, så der opstår to identiske spejlinger i hhv. adresserummet 00...7F og 80...FF.

Adresse (Hex)	lige	ulige	funktion
00 — 3F	—	PI/T	Read/write
40 — 4F	—	LM629 1	Read/write
50 — 5F	—	LM629 2	Read/write
60 — 6F	—	LM629 3	Read/write
70 — 7F	—	LM629 4	Read/write
80 — BF	—	PI/T	Read/write
C0 — CF	—	LM629 1	Read/write
D0 — DF	—	LM629 2	Read/write
E0 — EF	—	LM629 3	Read/write
F0 — FF	—	LM629 4	Read/write

Tabel 18.1: Gefions memory-map

18.4.1 PI/T

MC68230 PI/Ten har et adresserum på 32 bytes, der ligger på fortløbende ulige adresser i Gefions adresserum (1,3,5,...). Memorymappet for de ulige adresser i PI/Tens adresseblok er gengivet i tabel 18.2

Adr.	Register	Forkortelse	Adr.	Register	Forkortelse
01	Port General Control Register	PGCR	21	Timer Control Register	TCR
03	Port Service Request Register	PSRR	23	Timer Interrupt Vector Register	TIVR
05	Port A Data Direction Register	PADDR	25	— Intet register —	
07	Port B Data Direction Register	PBDDR	27	Counter Preload Register, High	CPRH
09	Port C Data Direction Register	PCDDR	29	Counter Preload Register, Mid	CPRM
0B	Port Interrupt Vector register	PIVR	2B	Counter Preload Register, Low	CPRL
0D	Port A Control Register	PACR	2D	— Intet register —	
0F	Port B Control Register	PBCR	2F	Count Register, High	CRH
11	Port A Data Register	PADR	31	Count Register, Mid	CRM
13	Port B Data Register	PBDR	33	Count Register, Low	CRL
15	Port A Alternate Register	PAAR	35	Timer Status Register	TSR
17	Port B Alternate Register	PBAR	37	— Intet register —	
19	Port C Data Register	PCDR	39	— Intet register —	
1B	Port Status Register	PSR	3B	— Intet register —	
1D	— Intet register —		3D	— Intet register —	
1F	— Intet register —		3F	— Intet register —	

Tabel 18.2: Memory map for PI/T områderne

18.4.2 LM629 motion controller

LM629 motion control ICerne optræder som 16 bytes store blokke i Gefions adresserum, men har hver især kun et internt adresserum på 2 bytes. De to register optræder skiftevis på de ulige adresser. Tabel 18.3 gengiver memorymappet for de ulige adresser i hver af LM629ernes 16 bytes blokke.

Adr	Register	Forkortelse	Adr	Register	forkortelse
1	Control register	CR	9	Control register	CR
3	Data/command register	DCR	B	Data/command register	DCR
5	Control register	CR	D	Control register	CR
7	Data/command register	DCR	F	Data/command register	DCR

Tabel 18.3: Memory map for LM629 områderne

18.4.3 Adressering fra MC68000 Assembler program

Der er mange måder at foretage adressering af Gefions registre på, via 68000 maskinkodeinstruktioner. Gefions konstruktion lægger op til at læse og skrive i registrene vha. byte orienterede kommandoer som *move.b*, men word orienterede kommandoer er også mulige.

Figur 18.8 viser et simpelt eksempel på adressering af Gefions forskellige registre fra maskinkode.

Alle MC68000's adresseringsmetoder kan naturligvis anvendes. Så længe Gefions registre adresseres korrekt.

```

BASIS          equ      $873E0000      ; Gefions basisadresse er sat til
                                           ; 3E0000 via dipswitches S1 og S2
                                           ; CPU modulet mapper VME I/O adresser
                                           ; til adresseområdet 87xxxxxx

PGCR           equ      BASIS + 1       ; Port General Control Register i PI/T
PSRR           equ      BASIS + 3       ; Port Service Request Register i PI/T
PADDR          equ      BASIS + 5       ; Port A Data Direction Register
; etc, etc.

CSR1           equ      BASIS + $41     ; Control/Status register i LM629 1
DR1            equ      BASIS + $43     ; Data register i LM629 1
CSR2           equ      BASIS + $51     ; Control/Status register i LM629 2
DR2            equ      BASIS + $53     ; Data register i LM629 2
;
; etc etc
;
; Udsnit af program
;

                move.b   CSR1,d0        ; Læs CSR1 over i data register 0's 8 laveste bits
                move.b   #$00,CSR1      ; Giv kommando med værdi 0 (RESET) til LM629 1
                move.b   #$3f,PGCR      ; læg værdien 0 i PI/Tens PGCR register
                move.b   PADR,d1        ; Læs de 8 bits på PI/Tens port A over i data register 1

```

Figur 18.8: Eksempel på adressering fra maskinkode

18.4.4 Adressering fra C program

Heimdals registre kan adresseres direkte fra C, uden brug af inline maskinkode:

```

/* C arbejder ikke direkte med typen bytes, så følgende definition
af denne type, er afhængig af hvilken C compiler der anvendes. For
Microwares OS-9/68k C ver. 3.0, der anvendes i forbindelse med OS-9 på Cato's
PEP VME computer gælder følgende definition: */
#define BYTE unsigned char

/* Følgende linie skriver byte-værdien VALUE i registeret på adresse ADR */
*(BYTE *)ADR = VALUE;

```

```
/* Følgende læser indholdet af byte registeret på adresse ADR, ind i variabelen byte */
byte = *(BYTE *)ADR;
```

Figur 18.9 viser et eksempel på C kode, med samme funktion som assemblerkoden i figur 18.8.

Anvendes metoden sammen med andre compilere, er det vigtigt at sikre sig at adresseringen stadig foretages korrekt. F.eks. ved at studere den maskinkode compileren genererer.

```
#define BYTE unsigned char

#define BASIS 0x873e0000

#define PGCR          *(BYTE *) (BASIS + 1)
#define PSRR          *(BYTE *) (BASIS + 3)
#define PADDR         *(BYTE *) (BASIS + 5)
/* etc. etc. */
#define CSR1           *(BYTE *) (BASIS + 0x41)
#define DR1            *(BYTE *) (BASIS + 0x43)
#define CSR2           *(BYTE *) (BASIS + 0x51)
#define DR1            *(BYTE *) (BASIS + 0x53)

/* etc etc */

/* Udsnit af program */

BYTE status;           /* Variabel definitioner */
BYTE port_a;

status = CSR1;          /* Aflæs status for LM629 1 */
CSR1 = 0;               /* Giv kommando med værdi 0 (RESET) til LM629 1 */
PGCR = 0x3F;           /* Skriv 3F_hex til PGCR */
port_a = PADDR          /* Aflæs de 8 bits på PI/Tens port A */
/* etc etc */

range[0] = sonar0;      /* Aflæs sonar register 0 */
range[1] = sonar1;      /* Aflæs sonar register 1 */
```

Figur 18.9: Eksempler på adressering i C

18.5 Programmering

Programmering af Gefion baserer sig på forståelse af såvel LM629 motion control IC'en, og MC68230 PI/Ten, som Gefions konstruktion. I dette afsnit gives kun en introduktion, og der henvises til [20] og [19], samt diagrammerne i afsnit 18.6, for yderligere information.

Memory mappet for Gefion, såvel som for MC68230 og LM629 findes i afsnit 18.4.

18.5.1 LM629 registeroversigt

LM629 programmeres via 2 8-bits registre: **Command/status** og **Data**. Command hhv. status funktionen af det første register opstår ved skrivning hhv. læsning.

LM629's kommandosæt består af 22 forskellige kommandoer, repræsenteret ved hver sin *byte* værdi. En kommando gives ved at skrive den pågældende byteværdi i **command** registeret. Hvis der overføres parametre med kommandoen læses eller skrives parameterværdierne sekventielt, byte for byte, via **data** registeret.

De 8 bits der aflæses ved at læse i **status** registeret koder hver især for forskellige tilstande internt i LM629. Inledningsvist er bit 0 den mest interessante, idet den koder for **busy**

18.5.2 Aflæsning af LM629 status

Status registeret aflæses vha. en *byte read cycle*, til Gefions basisadresse plus 41,52,63, eller 73 (hex), afhængigt af om det er LM629 1,2,3, eller 4 der anvendes.

18.5.3 Kommandoer til LM629

En kommando gives til LM629 ved at skrive byte værdien for kommandoen til **command** registeret. Efterfølgende skrives eller læses eventuelle data bytes til/fra **data** registeret. Det tager tid for *host interface* at udføre kommandoer, og flytte data mellem interne registre. I disse tidsrum må der ikke gives nye kommandoer, eller skrives/læses i data registeret. LM629 indikerer disse travle perioder med *busy* bittene i **status** registeret. De travle perioder bør udnyttes til noget fornuftigt af CPU'en, men man kommer næppe uden om at *poll* statusregisteret for at checke busy bittene før en kommando gives.

18.5.4 Parameteroverførsel til LM629

En del kommandoer overfører parametre til eller fra LM629. Parametrene er enten i form af 8, 16, eller 32 bits tal (bytes, words, og long-words). To kommandoer: **Load Filter Parameters**, og **Load Trajectory** overfører mere end en parameter.

Uanset hvilke, og hvor mange parametre, overføres de sekventielt, en byte ad gangen, med mest betydende bytes først. Efter hver kommando ventes på busy bittene, før parameteroverførslen begynder. Derefter ventes på busy, for hver 2 bytes der overføres.

Nedenfor er gengivet et sæt makroer og subrutiner i C til overførsel af bytes, words, og longwords. Bemærk at makroerne ikke venter på busy, og at busy derfor skal testes før makroerne kaldes. Subrutinerne til overførsel af longwords tester busy før makroerne kaldes.

```
#define WAIT1 do { } while (STATUS1 & 0x01)

#define READ_BYTE1      DATA1
#define WRITE_BYTE1(B)  DATA1 = ( B )

#define READ_WORD1      READ_BYTE1<<8 | READ_BYTE1
#define WRITE_WORD(B)   WRITE_BYTE1(( B )/0x100); WRITE_BYTE1(( B )%0x100)
.
.
.
unsigned long read_long1()
{
    unsigned short buffer;

    WAIT1;
    buffer = READ_WORD1;
    WAIT1;
    return buffer<<16 | READ_WORD1;
}

void write_long(num)
```

```

unsigned long num;
{
    WAIT1;
    WRITE_WORD1(num / 0x10000);
    WAIT1;
    WRITE_WORD1(num % 0x10000);
}

```

18.5.5 Asynkront processkift

Eftersom kommando og parameteroverførsler til LM629 ikke foregår vha. en udelelig maskininstruktion, kan et interrupt afbryde overførslen af parametre til/fra LM629. Dette er kun et problem hvis andre program-/system-dele begynder at skrive/læse til LM629 før den programdel der var igang med parameteroverførslen får lov at fortsætte og færdiggøre overførslen.

Hvis der arbejdes under et operativsystem med tvungen tidsdeling (f.eks. OS-9), eller hvis interruptrutiner anvender LM629en, er det vigtigt at sikre at en igangværende parameteroverførsel ikke afbrydes af nye kommandoer til LM629.

Hvis mere end een proces i et multitasking system anvender LM629, bør LM629 betragtes som kritisk ressource, og semaforbeskyttes el.lign.

Hvis En interruptrutine anvender LM629, bør det pågældende interrupt disables mens andre program-/system-dele overfører data til/fra LM629.

18.5.6 MC68230, uden interrupts

MC68230 er en ret avanceret periferikreds, der er meget logisk opbygget. Hvert register i kredsen har en unik adresse, og hvert register har kun en funktion.

Hvis der ikke anvendes interrupts, er MC68230's eneste funktion af fungere som 2 8-bits input porte, til aflæsning af de 16 digitale indgange. På V. 1.2. anvendes MC68230 også til jævnligt at nulstille *watch-dog* timeren, for at holde Gefions motorudgange aktive.

Initialisering

I alle tilfælde kan MC68230 initialiseres på følgende måde.

```

#define BASIS 0x873e0000 /* Gefions basisadresse */
#define PGCR *(unsigned char *) (BASIS + 0x01)
#define PSRR *(unsigned char *) (BASIS + 0x03)
#define PADDR *(unsigned char *) (BASIS + 0x04)
#define PBDDR *(unsigned char *) (BASIS + 0x07)
#define PCDDR *(unsigned char *) (BASIS + 0x09)
#define PIVR *(unsigned char *) (BASIS + 0x0B)
#define PACR *(unsigned char *) (BASIS + 0x0D)
#define PBCR *(unsigned char *) (BASIS + 0x0F)
#define PADR *(unsigned char *) (BASIS + 0x11)
#define PBDR *(unsigned char *) (BASIS + 0x13)
#define PAAR *(unsigned char *) (BASIS + 0x15)
#define PBAR *(unsigned char *) (BASIS + 0x17)
#define PCDR *(unsigned char *) (BASIS + 0x19)
#define PSR *(unsigned char *) (BASIS + 0x1B)

```

```

#define TCR      *(unsigned char *) (BASIS + 0x21)
#define TIVR     *(unsigned char *) (BASIS + 0x23)
#define CPRH     *(unsigned char *) (BASIS + 0x27)
#define CPRM     *(unsigned char *) (BASIS + 0x29)
#define CPRL     *(unsigned char *) (BASIS + 0x2B)
#define CNTRH    *(unsigned char *) (BASIS + 0x2F)
#define CNTRM    *(unsigned char *) (BASIS + 0x31)
#define CNTRL    *(unsigned char *) (BASIS + 0x33)
#define TSR      *(unsigned char *) (BASIS + 0x35)
.
.
.
void init_pit()
{
    PGCR = 0x3F; /* mode 0 (unidirectional 8 bit), all Hx inputs active, All
                  Hx flip flops triggers on rising flanks */
    PSRR = 0x18; /* No DMA, port C carries interrupt functions */
    PACR = 0x80; /* Submode 1x, H2/H4 are inputs, no interrupts */
    PBCR = 0x80; /* Submode 1x, H1/H3 are inputs, no interrupts */
    PCDR = 0x00; /* Reset all bits in output buffer for port C */
    PADDR = 0x00; /* All bits in port A are inputs */
    PBDDR = 0x00; /* All bits in port B are inputs */
    PCDDR = 0x01; /* Bit 0 in port C is output, the rest are inputs */
}

```

18.5.7 Aflæsning af digitale indgange

Efter initialisering foregår aflæsning af de 16 digitale indgange, ved at aflæse port A og port B i MC68230. Hvis en indgang sættes høj, så den pågældende optokobler aktiveres, aflæses den koresponderende bit som 0.

Nedenfor defineres en makro der aflæser alle 16 bits, med port A som MSB.

```
define DIGIN (PADR<<8|PBDR)
```

18.5.8 Watch-dog

På V. 1.2 er der integreret en *watch-dog* timer, der deaktiverer motorudgangene på Gefion, hvis den ikke periodisk nulstilles. Nulstillingen forgår ved at pulse bit 0 i port C på MC68230.

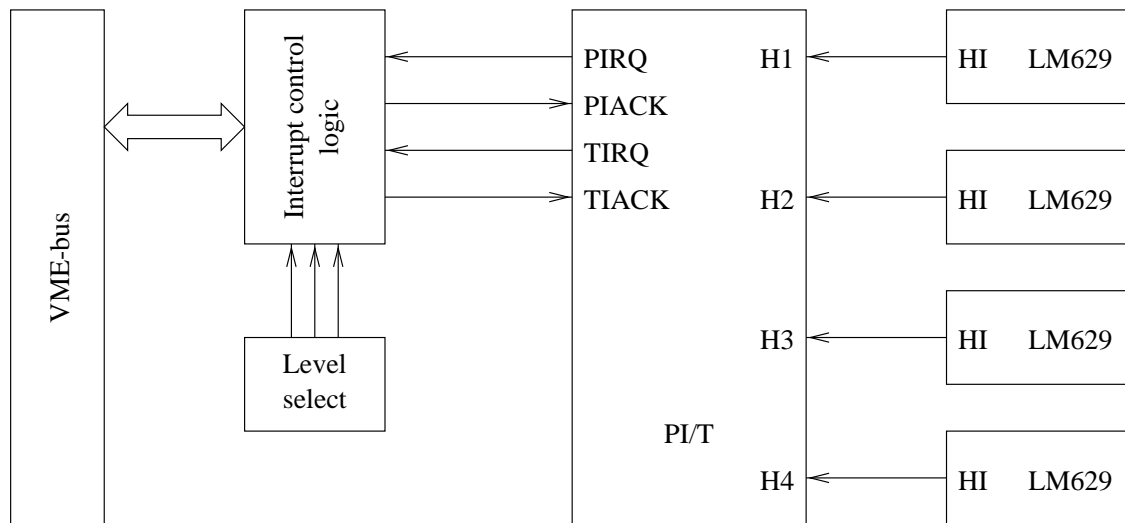
Nulstillingen kan varetages af nedenstående macro:

```
#define RESETWATCHDOG PCDR=0x01;PCDR=0x00
```

18.5.9 MC68230 og interrupts

MC68230 kan sættes op til at generere interrupts i to uafhængige tilfælde: Når dens indbyggede timer løber ud, eller når en af dens fire handshake indgange skifter tilstand.

Timeren anvendes ikke til noget i Gefion, og er for såvidt uinteressant i denne sammenhæng. Brugere af Gefion kan frit bruge timeren og dens interrupt til hvad de har lyst til.

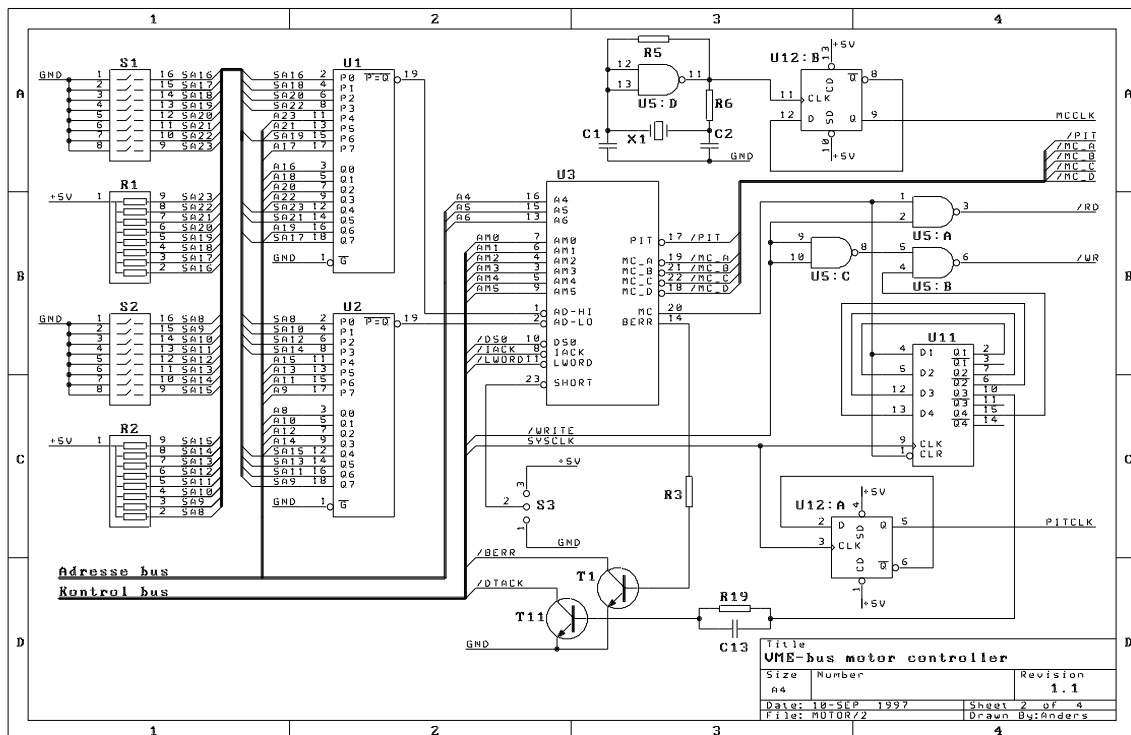
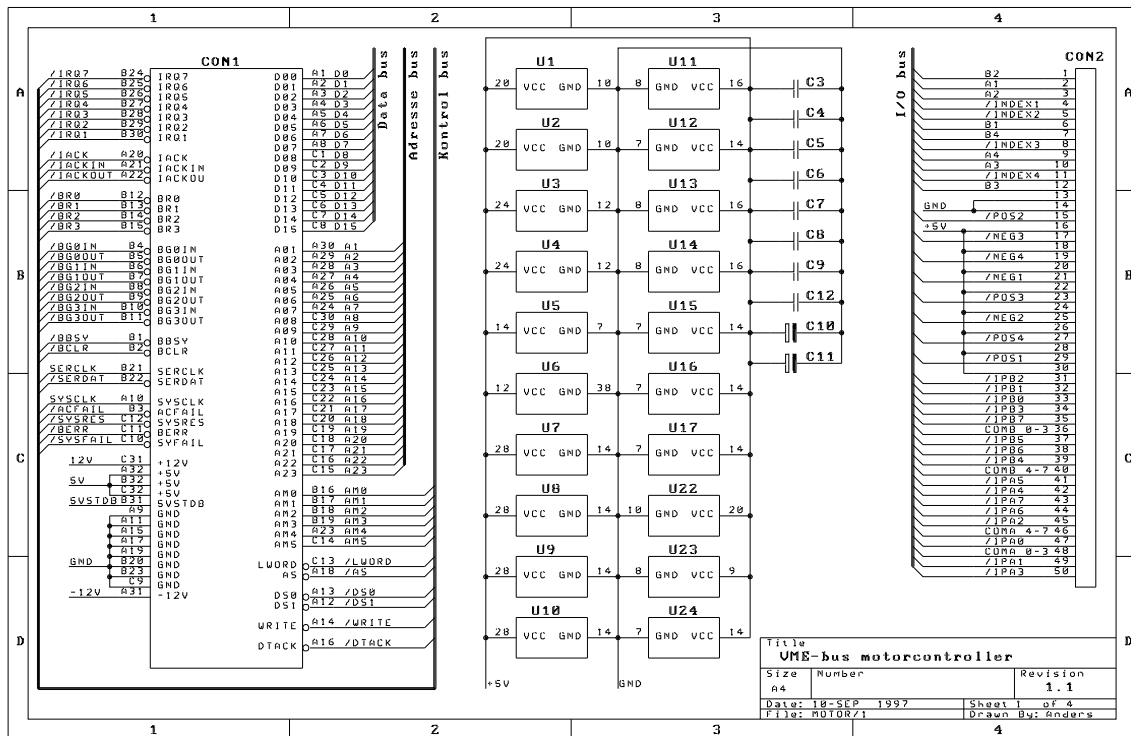


Figur 18.10: Gefions interrupt system

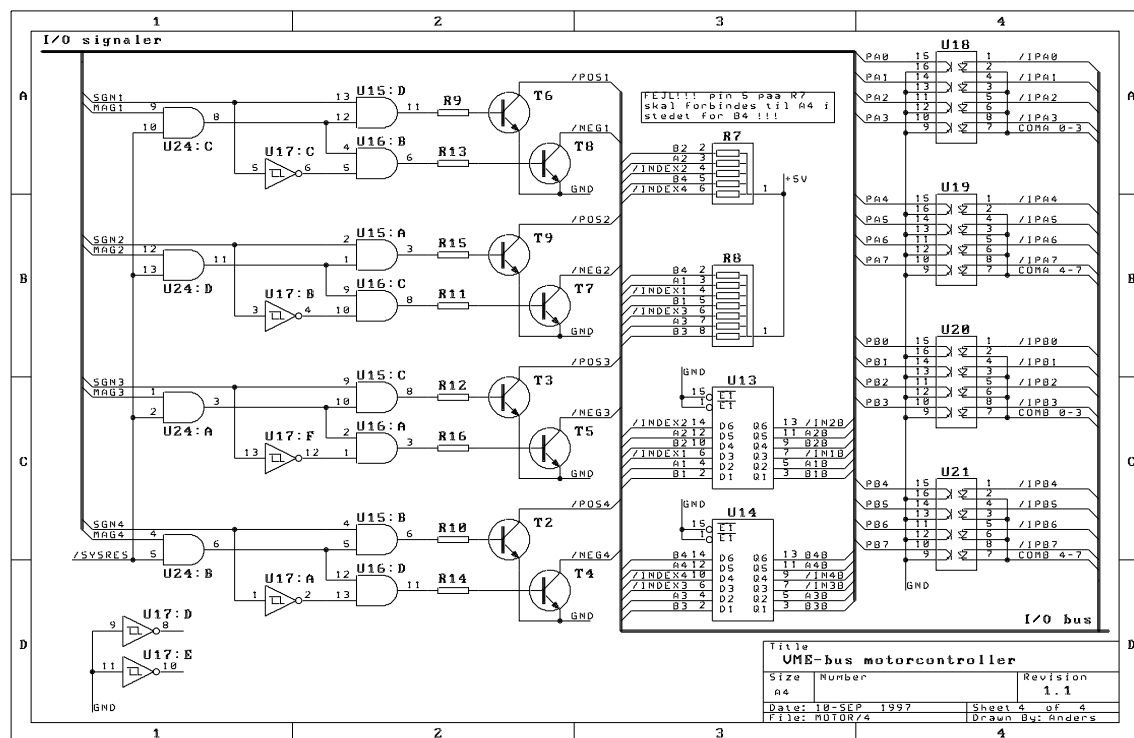
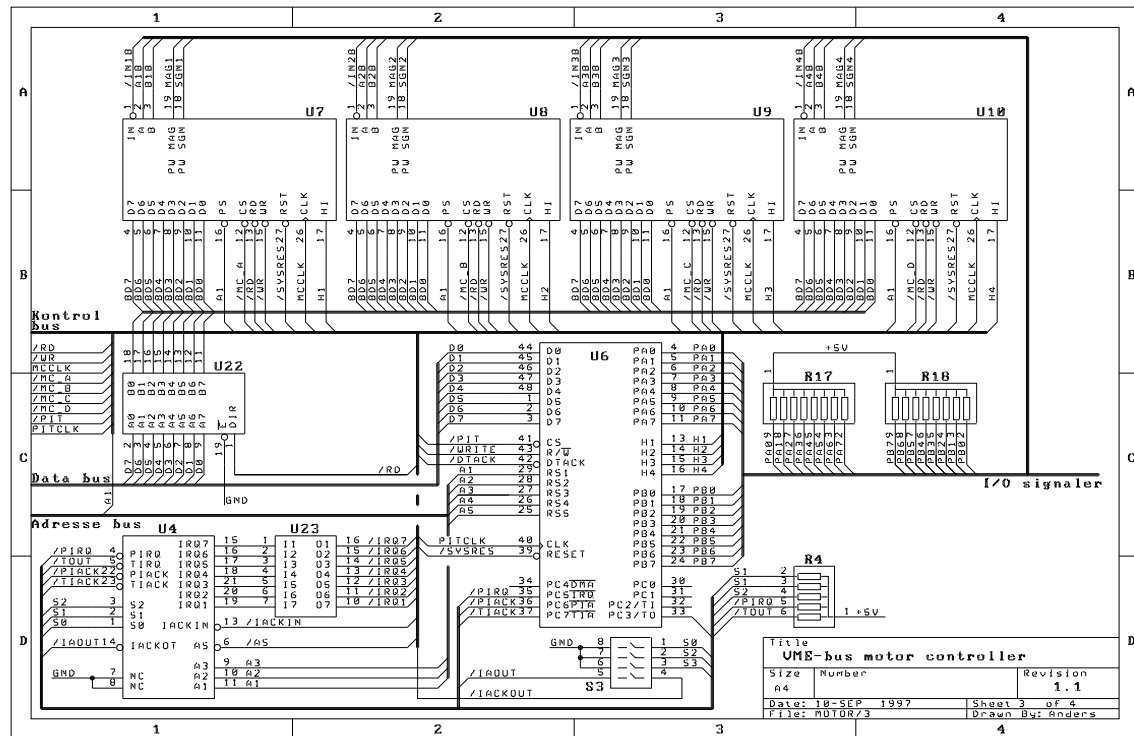
Som det ses på figur 18.10 har LM629 ernes *Handshake interrupt (HI)* udgange ikke direkte forbindelse til Gefions interrupt kontrollogik, men er forbundet til MC68230's handshake indgange. For at bruge LM629ernes interrupt faciliteter, skal MC68230 først sættes op til at generere interrupts ved opadgående flanker på sine handshake indgange. Tilsvarende skal en interrupt service rutine der skal tage sig af interrupts fra LM629erne, både adressere den pågældende LM629, men også MC68230, for at håndtere et interrupt med oprindelse i en LM629.

Til de fleste formål er det unødvendigt at anvende interrupts i forbindelse med styring af motorer vha. LM629, hvorfor jeg ikke yderligere vil uddybe anvendelsen af interrupts sammen med Gefion. Skulle det blive aktuelt at anvende interrupts, giver [19] og [20] de oplysninger om MC68230 og LM629's interrupt faciliteter der er nødvendige.

18.6 Diagrammer og printudlæg



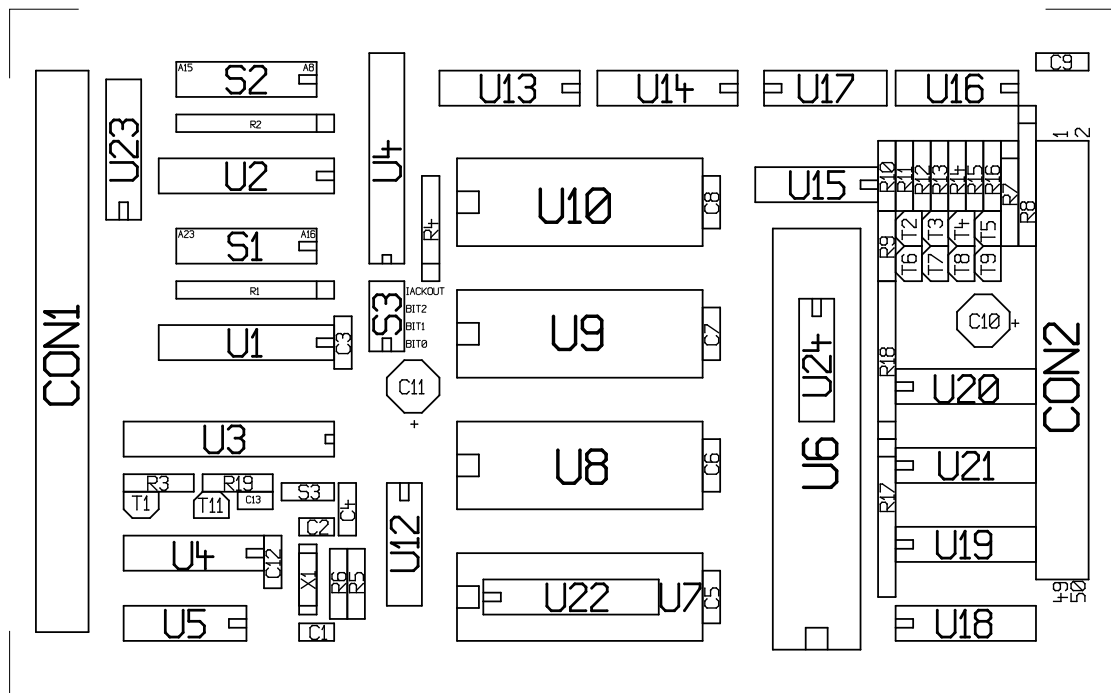
Figur 18.11: Diagram over Gefion V. 1.1 (1 og 2 af 4)



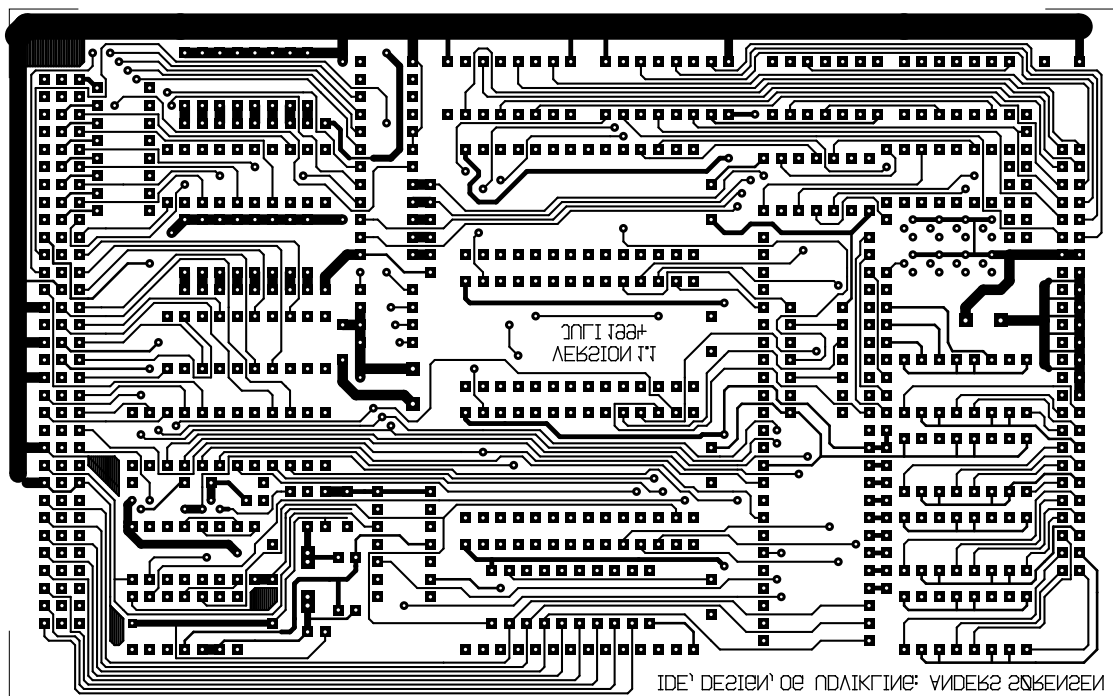
Figur 18.12: Diagram over Gefion V. 1.1 (3 og 4 af 4)

R1	8 * 4.7k	9 pin SIL	C7	100nF	sibatit	U1	74HCT688
R2	8 * 4.7k	9 pin SIL	C8	100nF	sibatit	U2	74HCT688
R3	4.7k		C9	100nF	sibatit	U3	PAL22V10
R4	5 * 4.7k	6 pin SIL	C10	470uF	elektrolyt	U4	PAL22V10
R5	1M		C11	470uF	elektrolyt	U5	74HCT00
R6	330		C12	100nF	sibatit	U6	MC68230-8
R7	5 * 4.7k	6 pin SIL	C13	47pF	keramisk	U7	LM629
R8	7 * 4.7k	8 pin SIL	CON1	DIN41612-C	96 pols	U8	LM629
R9	1k		CON2	IDC 90 °	50 pols	U9	LM629
R10	1k		S1	DIL-switch	8 pols	U10	LM629
R11	1k		S2	DIL-switch	8 pols	U11	74HC175
R12	1k		S3	jumper	3 pols	U12	74HCT74
R13	1k		S4	DIL-switch	4 pols	U13	74LS365A
R14	1k		X1	krystal	12MHz	U14	74LS365A
R16	1k		T1	BC547c		U15	74HCT08
R17	8 * 4.7k	9 pin SIL	T2	BC547c		U16	74HCT08
R18	8 * 4.7K	9 pin SIL	T3	BC547c		U17	74HCT14
R19	27k		T4	BC547c		U18	ISQ74
C1	5pF	keramisk	T5	BC547c		U19	ISQ74
C2	15pF	keramisk	T6	BC547c		U20	ISQ74
C3	100nF	sibatit	T8	BC547c		U21	ISQ74
C4	100nf	sibatit	T9	BC547c		U22	74ABT245
C5	100nF	sibatit	T10	BC547c		U23	ULN2003A
C6	100nF	sibatit	T11	BC547c		U24	74HCT08

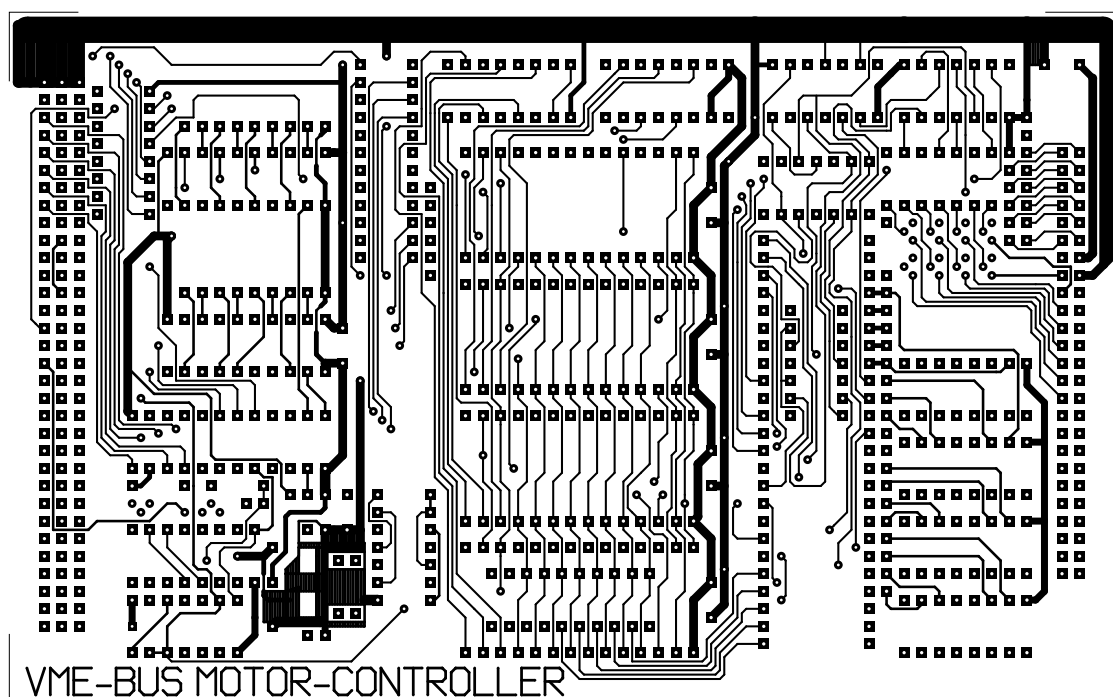
Tabel 18.4: Komponentliste til Gefion V. 1.1



Figur 18.13: Komponentplacering på Gefion V. 1.1 160mm x 100mm

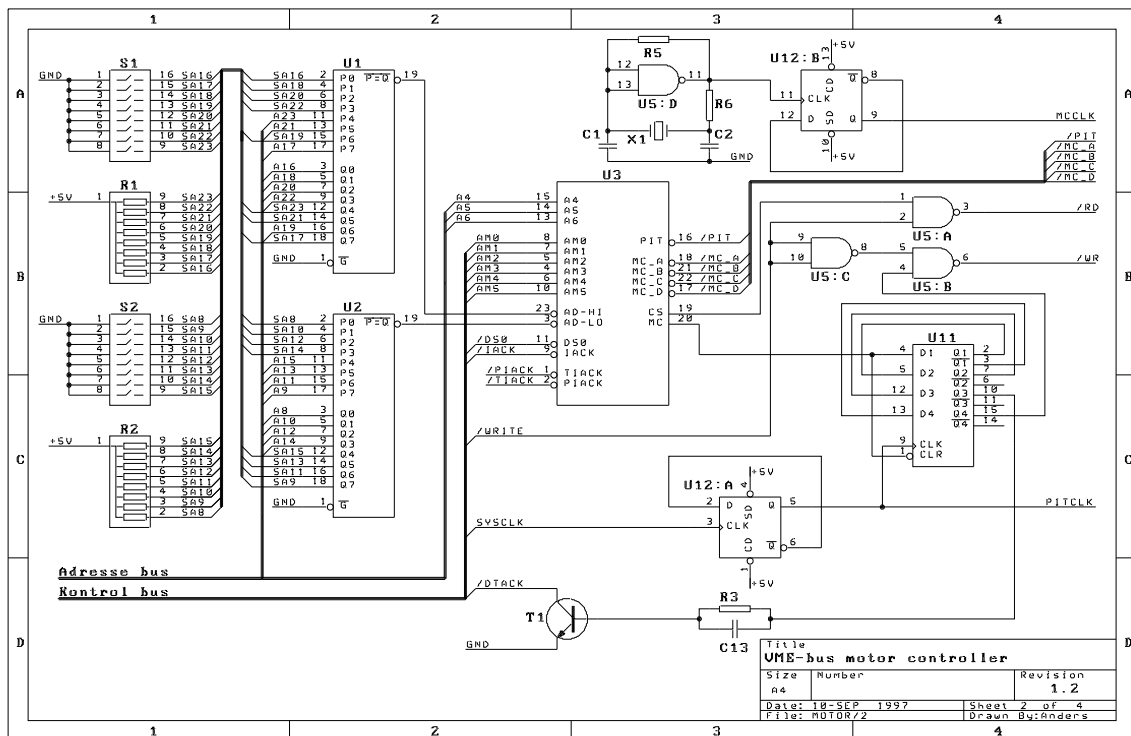
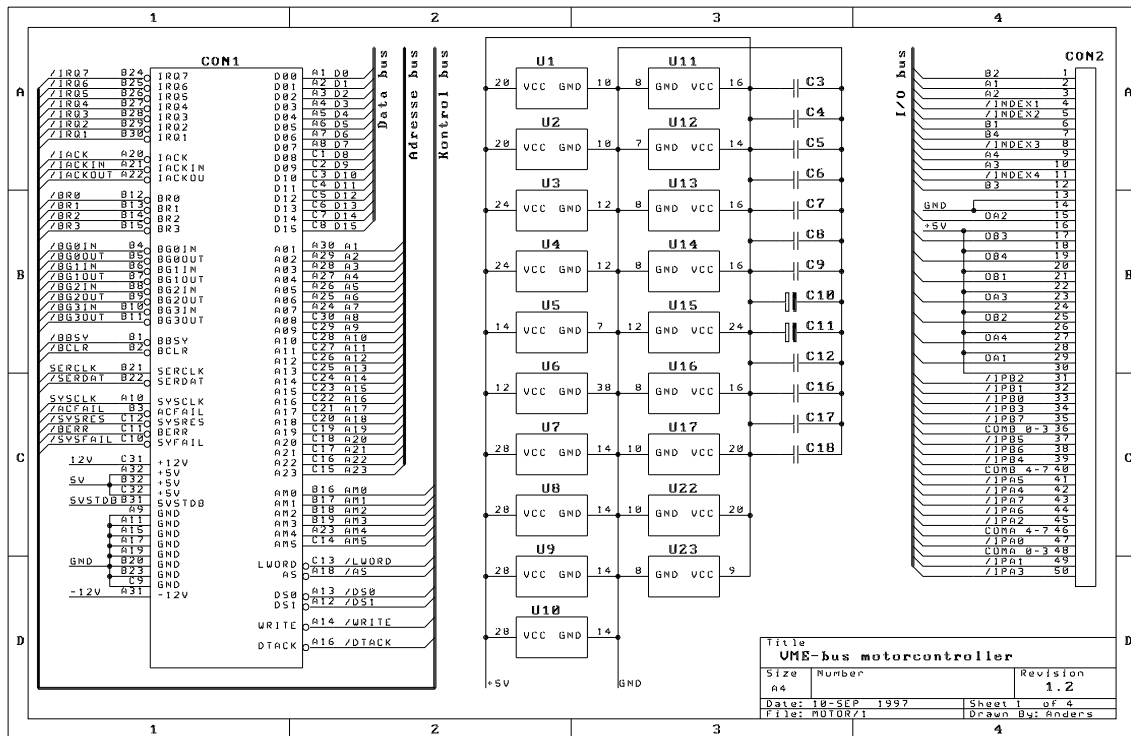


(a) Loddesside

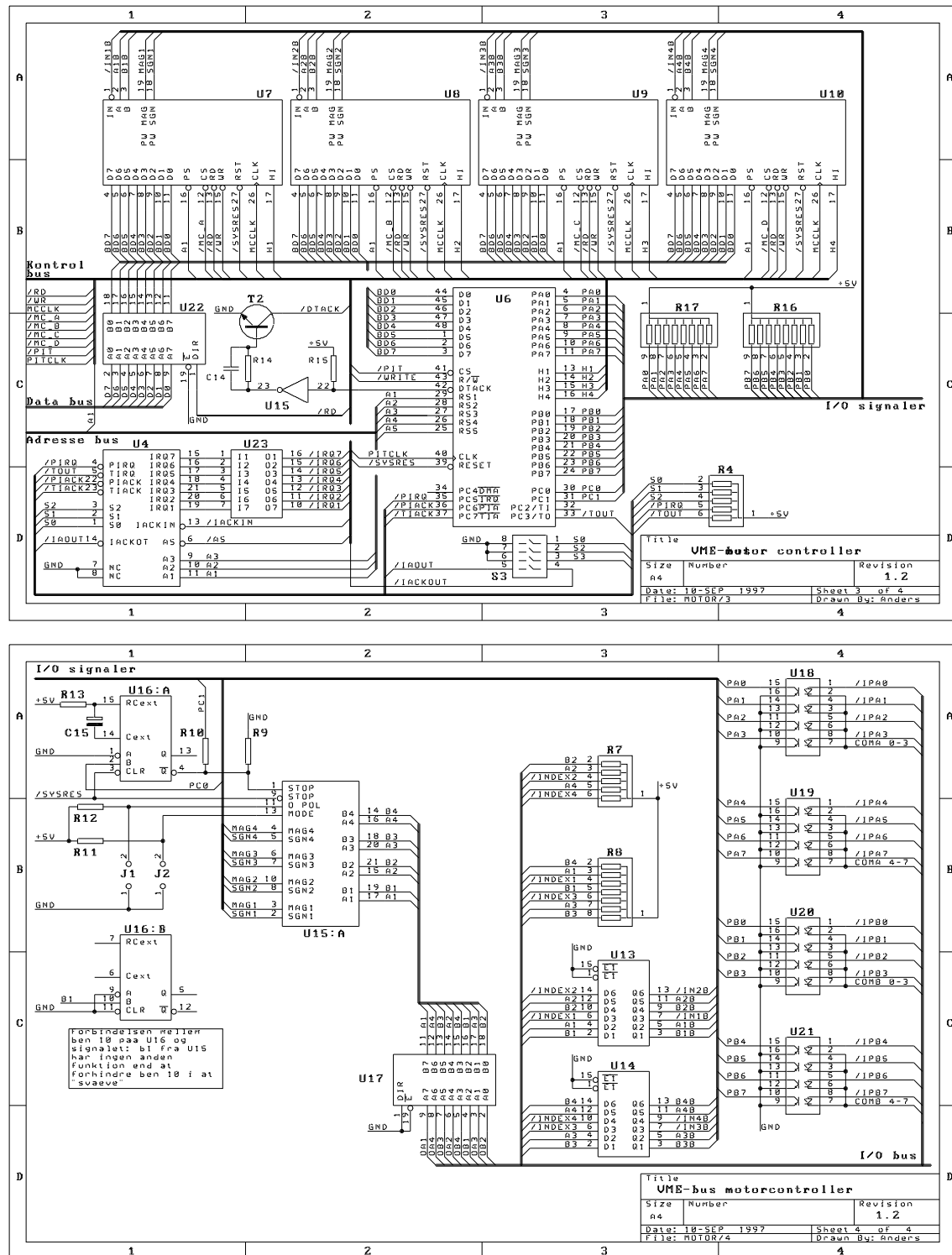


(b) Komponentside

Figur 18.14: Printudlæg af Gefion V. 1.1 160mm × 100mm



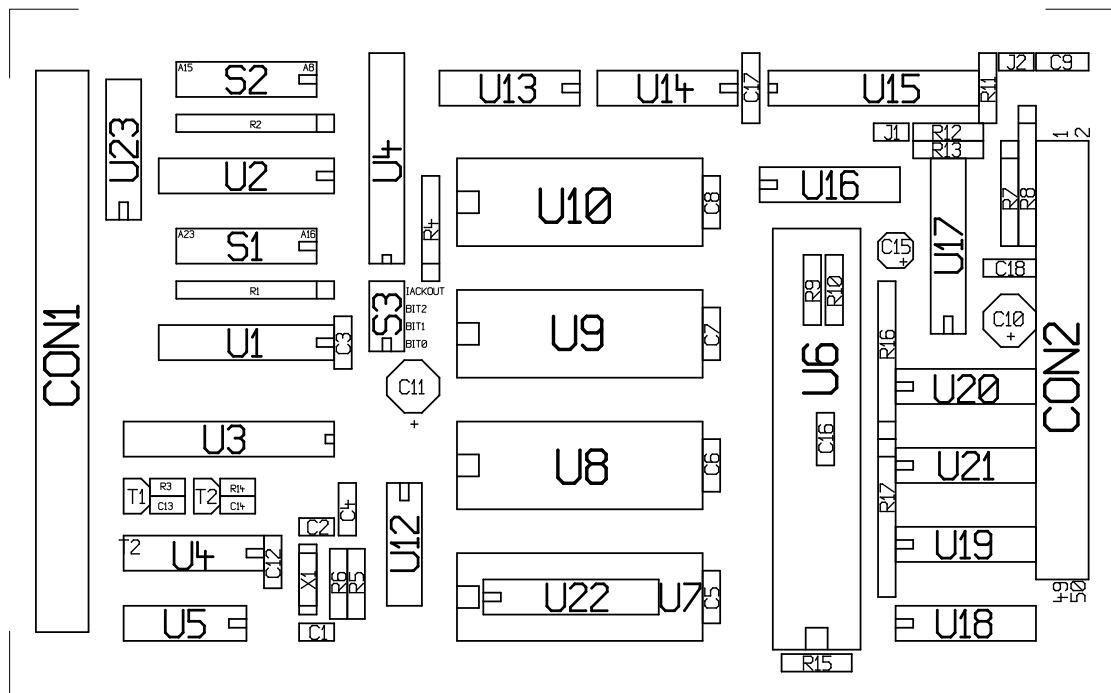
Figur 18.15: Diagram over Geflon V. 1.2 (1 og 2 af 4)



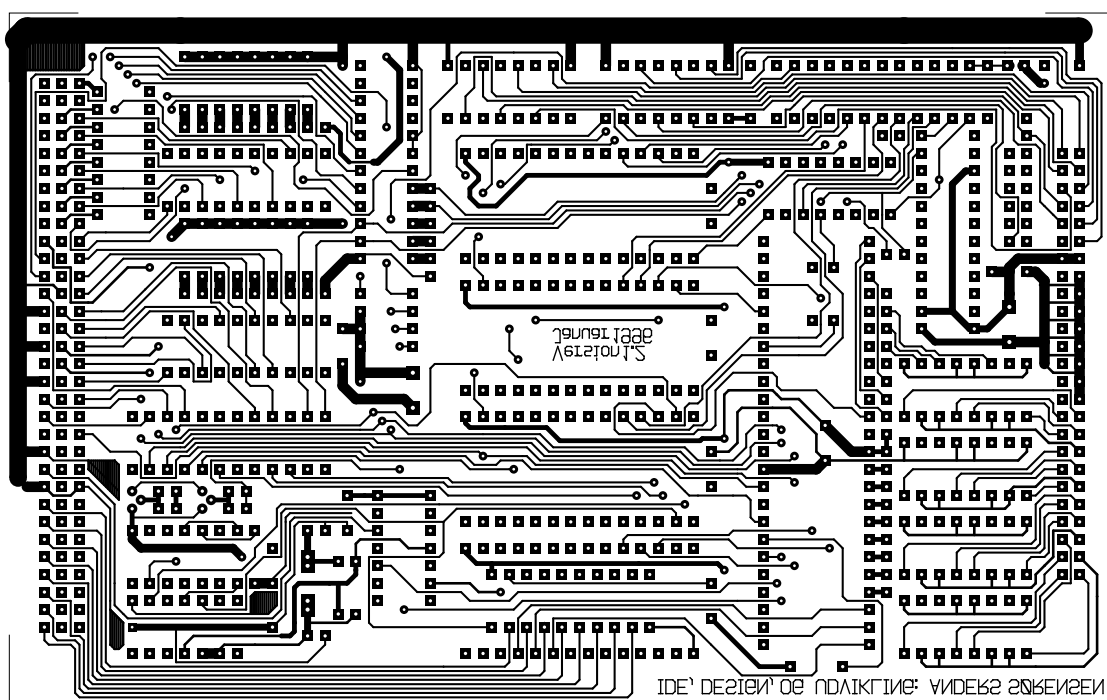
Figur 18.16: Diagram over Gefion V.1.2 (3 og 4 af 4)

R1	8 * 4.7k	9 pin SIL	C7	100nF	sibatit	U1	74HCT688
R2	8 * 4.7k	9 pin SIL	C8	100nF	sibatit	U2	74HCT688
R3	27k		C9	100nF	sibatit	U3	PAL22V10
R4	5 * 4.7k	6 pin SIL	C10	470uF	elektrolyt	U4	PAL22V10
R5	1M		C11	470uF	elektrolyt	U5	74HCT00
R6	330		C12	100nF	sibatit	U6	MC68230-8
R7	5 * 4.7k	6 pin SIL	C13	47pF	keramisk	U7	LM629
R8	7 * 4.7k	8 pin SIL	C14	47pF	keramisk	U8	LM629
R9	1k		C15	0.5 - 4.7μF	?F 25V	U9	LM629
R10	1k		C16	100nF	sibatit	U10	LM629
R11	4.7k		C17	100nF	sibatit	U11	74HC175
R12	4.7k		C18	100nF	sibatit	U12	74HCT74
R13	47k		CON1	DIN41612-C	96 pol	U13	74LS365A
R14	27k		CON2	IDC 90°	50 pol	U14	74LS365A
R15	1k		S1	DIL-switch	8 pol	U15	PAL22V10
R16	8 * 4.7k	9 pin SIL	S2	DIL-switch	8 pol	U16	74HCT123
R17	8 * 4.7k	9 pin SIL	S3	DIL-switch	4 pol	U17	74LS641-1
C1	5pF	keramisk	J1	jumper	2 pol	U18	ISQ74
C2	15pF	keramisk	J2	jumper	2 pol	U19	ISQ74
C3	100nF	sibatit	X1	krystal	12MHz	U20	ISQ74
C4	100nf	sibatit	T1	BC547c		U21	ISQ74
C5	100nF	sibatit	T2	BC547c		U22	74ABT245
C6	100nF	sibatit				U23	ULN2003A

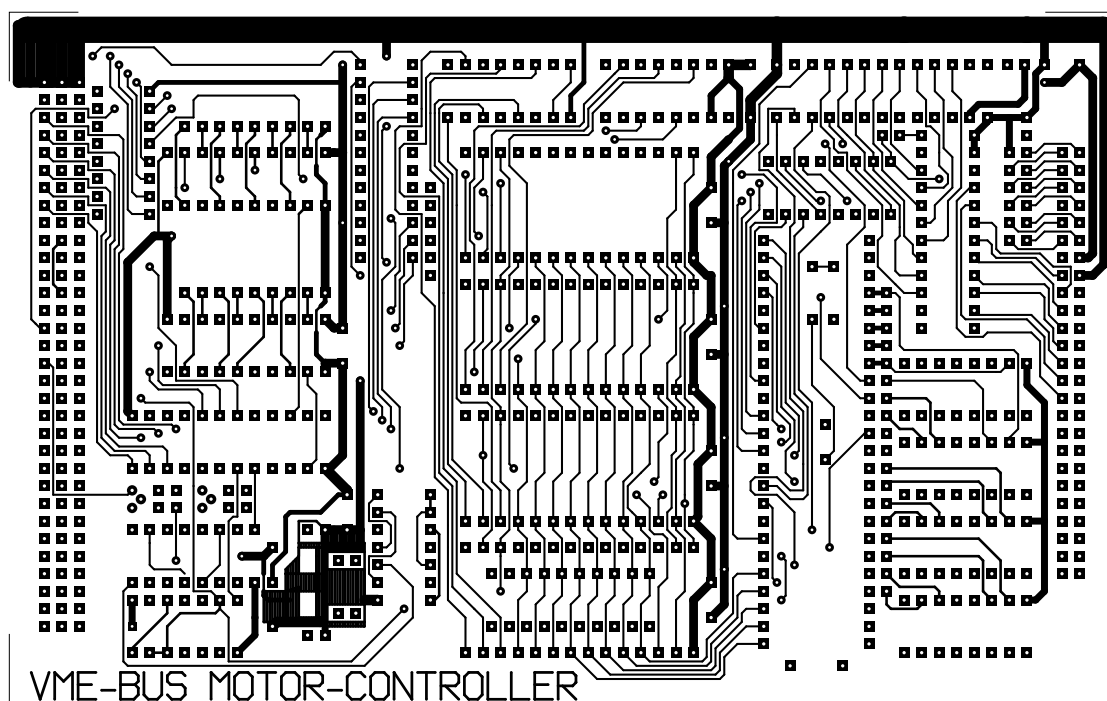
Tabel 18.5: Komponentliste til Gefion V. 1.2



Figur 18.17: Komponentplacering på Gefion V. 1.2 160mm×100mm



(a) Loddesside



(b) Komponentside

Figur 18.18: Printudlæg af Gefion V. 1.2 160mm × 100mm

18.7 Programmerbar logik

Alle programmerbare logikkredse på Gefion, er implementeret vha **PAL22V10**. Programmerne til dem er udviklet vha programmet **PALASM**, der tillader at udtrykke kredsens funktion i et PASCAL lignende højniveausprog, der så omsættes til tilstandsligninger. Hvert delafsnit herunder gengiver først højniveau kildeteksten, og derefter de resulterende tilstandsligninger.

18.7.1 U3 på V1.1

Kildetekst

```
;PALASM Design Description

;----- Declaration Segment -----
TITLE      ADRESSE DEKODER TIL VME-BUS MOTORCONTROLLER
PATTERN
REVISION 1.0
AUTHOR    ANDERS STENGAARD SOERENSEN
COMPANY    ODENSE UNIVERSITET, IMADA
DATE       07/09/94

CHIP _U3 PAL22V10

;----- PIN Declarations -----
PIN 1      /AD_HIGH      COMBINATORIAL ; INPUT
PIN 2      /AD_LOW       COMBINATORIAL ; INPUT
PIN 3      AM[3]         COMBINATORIAL ; INPUT
PIN 4      AM[2]         COMBINATORIAL ; INPUT
PIN 5      AM[4]         COMBINATORIAL ; INPUT
PIN 6      AM[1]         COMBINATORIAL ; INPUT
PIN 7      AM[0]         COMBINATORIAL ; INPUT
PIN 8      /IACK         COMBINATORIAL ; INPUT
PIN 9      AM[5]         COMBINATORIAL ; INPUT
PIN 10     /DS0          COMBINATORIAL ; INPUT
PIN 11     /LWORD        COMBINATORIAL ; INPUT
PIN 13     A[6]          COMBINATORIAL ; INPUT
PIN 15     A[5]          COMBINATORIAL ; INPUT
PIN 16     A[4]          COMBINATORIAL ; INPUT
PIN 17     /PIT          COMBINATORIAL ; OUTPUT
PIN 14     BERR          COMBINATORIAL ; OUTPUT
PIN 18     /MC[4]        COMBINATORIAL ; OUTPUT
PIN 19     /MC[1]        COMBINATORIAL ; OUTPUT
PIN 20     MC[0]         COMBINATORIAL ; OUTPUT
PIN 21     /MC[2]        COMBINATORIAL ; OUTPUT
PIN 22     /MC[3]        COMBINATORIAL ; OUTPUT
PIN 23     /SHORT       COMBINATORIAL ; INPUT

;----- Boolean Equation Segment -----
EQUATIONS
IF DS0 * /IACK * (SHORT * AD_LOW + /SHORT * AD_LOW * AD_HIGH) THEN BEGIN
  IF SHORT THEN BEGIN
    IF ((AM[5..0] = #h29) + (AM[5..0] = #h2D)) * /LWORD THEN BEGIN
      BERR=0
      CASE A[6..4] BEGIN
        0..3 : BEGIN
          PIT = 1
          MC[4..0] = 0
        END
        4 : BEGIN
          PIT = 0
          MC[0] = 1
          MC[1] = 1
          MC[4..2] = 0
        END
        5 : BEGIN
          PIT = 0
          MC[0] = 1
          MC[1] = 0
          MC[2] = 1
          MC[4..3] = 0
        END
        6 : BEGIN
          PIT = 0
          MC[0] = 1
          MC[2..1]=0
          MC[3] = 1
        END
      END
    END
  END
END
```

```

        MC[4] = 0
    END
    7 : BEGIN
        PIT = 0
        MC[0] = 1
        MC[3..1] = 0
        MC[4] = 1
    END
END
ELSE BEGIN
    BERR = 1
    PIT=0
    MC[4..0] = 0
END
END
ELSE BEGIN ; (IF /SHORT)
    IF ((AM[5..0] = #h39) + (AM[5..0] = #h3D) + (AM[5..0] = #h00)) * /LWORD THEN BEGIN
        BERR = 0
        CASE A[6..4] BEGIN
            0..3 : BEGIN
                PIT = 1
                MC[4..0] = 0
            END
            4 : BEGIN
                PIT = 0
                MC[0] = 1
                MC[1] = 1
                MC[4..2] = 0
            END
            5 : BEGIN
                PIT = 0
                MC[0] = 1
                MC[1] = 0
                MC[2] = 1
                MC[4..3] = 0
            END
            6 : BEGIN
                PIT = 0
                MC[0] = 1
                MC[2..1]=0
                MC[3] = 1
                MC[4] = 0
            END
            7 : BEGIN
                PIT = 0
                MC[0] = 1
                MC[3..1] = 0
                MC[4] = 1
            END
        END
    END
    ELSE BEGIN
        BERR = 1
        PIT = 0
        MC[4..0] = 0
    END
END
END
ELSE BEGIN
    PIT = 0
    MC[0..4] = 0
    BERR = 0
END

```

```

;----- Simulation Segment -----
SIMULATION

```

Tilstandsligninger

TITLE	ADRESSE DEKODER TIL VME-BUS MOTORCONTROLLER
PATTERN	
REVISION	1.0
AUTHOR	ANDERS STENGAARD SOERENSEN
COMPANY	ODENSE UNIVERSITET, IMADA
DATE	07/09/94

CHIP _U3 PAL22V10

```

PIN 1 /AD_HIGH COMB
PIN 2 /AD_LOW COMB
PIN 3 AM[3] COMB
PIN 4 AM[2] COMB
PIN 5 AM[4] COMB
PIN 6 AM[1] COMB
PIN 7 AM[0] COMB
PIN 8 /IACK COMB
PIN 9 AM[5] COMB
PIN 10 /DS0 COMB
PIN 11 /LWORD COMB
PIN 12 GND
PIN 13 A[6] COMB
PIN 14 BERR COMB
PIN 15 A[5] COMB
PIN 16 A[4] COMB
PIN 17 /PIT COMB
PIN 18 /MC[4] COMB
PIN 19 /MC[1] COMB
PIN 20 MC[0] COMB
PIN 21 /MC[2] COMB
PIN 22 /MC[3] COMB
PIN 23 /SHORT COMB
PIN 24 VCC

```

EQUATIONS

```

/BERR = /(DS0 * /IACK * (SHORT * AD_LOW + /SHORT * AD_LOW
      * AD_HIGH))
      + (AM[0] * /AM[1] * /AM[2] * AM[3] * AM[4] * AM[5]
      + AM[0] * /AM[1] * AM[2] * AM[3] * AM[4] * AM[5]
      + /AM[0] * /AM[1] * /AM[2] * /AM[3] *
      /AM[4] * /AM[5]) * /LWORD * /SHORT * DS0 *
      /IACK * (SHORT * AD_LOW + /SHORT * AD_LOW * AD_HIGH
      )
      + (AM[0] * /AM[1] * /AM[2] * AM[3] * /AM[4] * AM[5]
      + AM[0] * /AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]
      ) * /LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW
      + /SHORT * AD_LOW * AD_HIGH)
PIT = (/A[6] + /A[5] * /A[6]) * (AM[0] * /AM[1] *
      /AM[2] * AM[3] * AM[4] * AM[5] + AM[0] *
      /AM[1] * AM[2] * AM[3] * AM[4] * AM[5] +
      /AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *
      /AM[5]) * /LWORD * /SHORT * DS0 * /IACK * (SHORT
      * AD_LOW + /SHORT * AD_LOW * AD_HIGH)
      + (/A[6] + /A[5] * /A[6]) * (AM[0] * /AM[1] *
      /AM[2] * AM[3] * /AM[4] * AM[5] + AM[0] *
      /AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
      /LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
      /SHORT * AD_LOW * AD_HIGH)
/MC[4] = /(DS0 * /IACK * (SHORT * AD_LOW + /SHORT * AD_LOW
      * AD_HIGH))
      +
      /((AM[0] * /AM[1] * /AM[2] * AM[3] * AM[4] * AM[5]
      + AM[0] * /AM[1] * AM[2] * AM[3] * AM[4] * AM[5]
      + /AM[0] * /AM[1] * /AM[2] * /AM[3] *
      /AM[4] * /AM[5]) * /LWORD) * /SHORT * DS0 *
      /IACK * (SHORT * AD_LOW + /SHORT * AD_LOW * AD_HIGH
      )
      + /A[4] * A[5] * A[6] * (AM[0] * /AM[1] *
      /AM[2] * AM[3] * AM[4] * AM[5] + AM[0] *
      /AM[1] * AM[2] * AM[3] * AM[4] * AM[5] +
      /AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *
      /AM[5]) * /LWORD * /SHORT * DS0 * /IACK * (SHORT
      * AD_LOW + /SHORT * AD_LOW * AD_HIGH)
      + A[4] * /A[5] * A[6] * (AM[0] * /AM[1] *
      /AM[2] * AM[3] * AM[4] * AM[5] + AM[0] *
      /AM[1] * AM[2] * AM[3] * AM[4] * AM[5] +
      /AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *
      /AM[5]) * /LWORD * /SHORT * DS0 * /IACK * (SHORT
      * AD_LOW + /SHORT * AD_LOW * AD_HIGH)
      + /A[4] * /A[5] * A[6] * (AM[0] * /AM[1] *
      /AM[2] * AM[3] * AM[4] * AM[5] + AM[0] *
      /AM[1] * AM[2] * AM[3] * AM[4] * AM[5] +
      /AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *

```

```

/AM[5]) * /LWORD * /SHORT * DS0 * /IACK * (SHORT
* AD_LOW + /SHORT * AD_LOW * AD_HIGH)
+ ((/A[6] + /A[5] * /A[6]) * (AM[0] *
/AM[1] * /AM[2] * AM[3] * AM[4] * AM[5] + AM[0]
* /AM[1] * AM[2] * AM[3] * AM[4] * AM[5] +
/AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *
/AM[5]) * /LWORD * /SHORT * DS0 * /IACK * (SHORT
* AD_LOW + /SHORT * AD_LOW * AD_HIGH)
+
/((AM[0] * /AM[1] * /AM[2] * AM[3] * /AM[4] * AM[5]
+ AM[0] * /AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]
) * /LWORD) * SHORT * DS0 * /IACK * (SHORT * AD_LOW
+ /SHORT * AD_LOW * AD_HIGH)
+ /A[4] * A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * /AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)
+ A[4] * /A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * /AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)
+ /A[4] * /A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * /AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)
+ ((/A[6] + /A[5] * /A[6]) * (AM[0] *
/AM[1] * /AM[2] * AM[3] * /AM[4] * AM[5] + AM[0]
* /AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)
/MC[3] = /(DS0 * /IACK * (SHORT * AD_LOW + /SHORT * AD_LOW
* AD_HIGH))
+
/((AM[0] * /AM[1] * /AM[2] * AM[3] * AM[4] * AM[5]
+ AM[0] * /AM[1] * AM[2] * AM[3] * AM[4] * AM[5]
+ /AM[0] * /AM[1] * /AM[2] * /AM[3] *
/AM[4] * /AM[5]) * /LWORD) * /SHORT * DS0 *
/IACK * (SHORT * AD_LOW + /SHORT * AD_LOW * AD_HIGH)
+
A[4] * A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * AM[4] * AM[5] +
/AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *
/AM[5]) * /LWORD * /SHORT * DS0 * /IACK * (SHORT
* AD_LOW + /SHORT * AD_LOW * AD_HIGH)
+ A[4] * /A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * AM[4] * AM[5] +
/AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *
/AM[5]) * /LWORD * /SHORT * DS0 * /IACK * (SHORT
* AD_LOW + /SHORT * AD_LOW * AD_HIGH)
+ ((/A[6] + /A[5] * /A[6]) * (AM[0] *
/AM[1] * /AM[2] * AM[3] * AM[4] * AM[5] + AM[0]
* /AM[1] * AM[2] * AM[3] * AM[4] * AM[5] +
/AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *
/AM[5]) * /LWORD * /SHORT * DS0 * /IACK * (SHORT
* AD_LOW + /SHORT * AD_LOW * AD_HIGH)
+
/((AM[0] * /AM[1] * /AM[2] * AM[3] * /AM[4] * AM[5]
+ AM[0] * /AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]
) * /LWORD) * SHORT * DS0 * /IACK * (SHORT * AD_LOW
+ /SHORT * AD_LOW * AD_HIGH)
+ A[4] * A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * /AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)
+ A[4] * /A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * /AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +

```

```

/SHORT * AD_LOW * AD_HIGH)
+ /A[4] * /A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * /AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)
+ (/A[6] + /A[5] * /A[6]) * (AM[0] *
/AM[1] * /AM[2] * AM[3] * /AM[4] * AM[5] + AM[0]
* /AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)
/MC[2] = /(DS0 * /IACK * (SHORT * AD_LOW + /SHORT * AD_LOW
* AD_HIGH))
+
/((AM[0] * /AM[1] * /AM[2] * AM[3] * AM[4] * AM[5]
+ AM[0] * /AM[1] * AM[2] * AM[3] * AM[4] * AM[5]
+ /AM[0] * /AM[1] * /AM[2] * /AM[3] *
/AM[4] * /AM[5]) * /LWORD) * /SHORT * DS0 *
/IACK * (SHORT * AD_LOW + /SHORT * AD_LOW * AD_HIGH
)
+ A[4] * A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * AM[4] * AM[5] +
/AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *
/AM[5]) * /LWORD * /SHORT * DS0 * /IACK * (SHORT
* AD_LOW + /SHORT * AD_LOW * AD_HIGH)
+ /A[4] * A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * AM[4] * AM[5] +
/AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *
/AM[5]) * /LWORD * /SHORT * DS0 * /IACK * (SHORT
* AD_LOW + /SHORT * AD_LOW * AD_HIGH)
+ (/A[6] + /A[5] * /A[6]) * (AM[0] *
/AM[1] * /AM[2] * AM[3] * AM[4] * AM[5] + AM[0]
* /AM[1] * AM[2] * AM[3] * AM[4] * AM[5] +
/AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *
/AM[5]) * /LWORD * /SHORT * DS0 * /IACK * (SHORT
* AD_LOW + /SHORT * AD_LOW * AD_HIGH)
+
/((AM[0] * /AM[1] * /AM[2] * AM[3] * /AM[4] * AM[5]
+ AM[0] * /AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]
) * /LWORD) * SHORT * DS0 * /IACK * (SHORT * AD_LOW
+ /SHORT * AD_LOW * AD_HIGH)
+ A[4] * A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * /AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)
+ /A[4] * A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * /AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)
+ (/A[6] + /A[5] * /A[6]) * (AM[0] *
/AM[1] * /AM[2] * AM[3] * /AM[4] * AM[5] + AM[0]
* /AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)
/MC[1] = /(DS0 * /IACK * (SHORT * AD_LOW + /SHORT * AD_LOW
* AD_HIGH))
+
/((AM[0] * /AM[1] * /AM[2] * AM[3] * AM[4] * AM[5]
+ AM[0] * /AM[1] * AM[2] * AM[3] * AM[4] * AM[5]
+ /AM[0] * /AM[1] * /AM[2] * /AM[3] *
/AM[4] * /AM[5]) * /LWORD) * /SHORT * DS0 *
/IACK * (SHORT * AD_LOW + /SHORT * AD_LOW * AD_HIGH
)
+ A[4] * A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * AM[4] * AM[5] + AM[0] *

```



```

/AM[2] * AM[3] * /AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)
+ A[4] * /A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * /AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)
MC[1] = /A[4] * /A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * AM[4] * AM[5] +
/AM[0] * /AM[1] * /AM[2] * /AM[3] *
/AM[4] * /AM[5]) * /LWORD * /SHORT * DS0 *
/IACK * (SHORT * AD_LOW + /SHORT * AD_LOW * AD_HIGH
)
+ /A[4] * /A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * /AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)
MC[2] = A[4] * /A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * AM[4] * AM[5] +
/AM[0] * /AM[1] * /AM[2] * /AM[3] *
/AM[4] * /AM[5]) * /LWORD * /SHORT * DS0 *
/IACK * (SHORT * AD_LOW + /SHORT * AD_LOW * AD_HIGH
)
+ A[4] * /A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * /AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)
MC[3] = /A[4] * A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * AM[4] * AM[5] +
/AM[0] * /AM[1] * /AM[2] * /AM[3] *
/AM[4] * /AM[5]) * /LWORD * /SHORT * DS0 *
/IACK * (SHORT * AD_LOW + /SHORT * AD_LOW * AD_HIGH
)
+ /A[4] * A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * /AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)
MC[4] = A[4] * A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * AM[4] * AM[5] +
/AM[0] * /AM[1] * /AM[2] * /AM[3] *
/AM[4] * /AM[5]) * /LWORD * /SHORT * DS0 *
/IACK * (SHORT * AD_LOW + /SHORT * AD_LOW * AD_HIGH
)
+ A[4] * A[5] * A[6] * (AM[0] * /AM[1] *
/AM[2] * AM[3] * /AM[4] * AM[5] + AM[0] *
/AM[1] * AM[2] * AM[3] * /AM[4] * AM[5]) *
/LWORD * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)
BERR =
/((AM[0] * /AM[1] * /AM[2] * AM[3] * AM[4] * AM[5]
+ AM[0] * /AM[1] * AM[2] * AM[3] * AM[4] * AM[5]
+ /AM[0] * /AM[1] * /AM[2] * /AM[3] *
/AM[4] * /AM[5]) * /LWORD) * /SHORT * DS0 *
/IACK * (SHORT * AD_LOW + /SHORT * AD_LOW * AD_HIGH
)
+
/((AM[0] * /AM[1] * /AM[2] * AM[3] * /AM[4] * AM[5]
+ AM[0] * /AM[1] * AM[2] * AM[3] * /AM[4] * AM[5])
* /LWORD) * SHORT * DS0 * /IACK * (SHORT * AD_LOW +
/SHORT * AD_LOW * AD_HIGH)

```

18.7.2 U4 på V1.1 og V1.2

Kildetekst

```
;PALASM Design Description

;----- Declaration Segment -----
TITLE      INTERRUPT CONTROLLER TIL VME-BUS MOTORCONTROLLER
PATTERN
REVISION 1.1-B (1.1 havde ombyttet nogle af IRQ[n] signalerne)
AUTHOR   ANDERS STENGAARD SOERENSEN
COMPANY  ODENSE UNIVERSITET, IMADA
DATE     7/1/96

CHIP _U4 PAL22V10

;----- PIN Declarations -----
PIN 1      S[0]      COMBINATORIAL ; INPUT
PIN 2      S[1]      COMBINATORIAL ; INPUT
PIN 3      S[2]      COMBINATORIAL ; INPUT
PIN 4      /PIRQ     COMBINATORIAL ; INPUT
PIN 5      /TIRQ     COMBINATORIAL ; INPUT
PIN 6      /AS       COMBINATORIAL ; INPUT
PIN 7      NC1       COMBINATORIAL ; INPUT
PIN 8      NC2       COMBINATORIAL ; INPUT
PIN 9      A[3]      COMBINATORIAL ; INPUT
PIN 10     A[2]      COMBINATORIAL ; INPUT
PIN 11     A[1]      COMBINATORIAL ; INPUT
PIN 13     /IACK_IN  COMBINATORIAL ; INPUT
PIN 14     /IACK_OUT COMBINATORIAL ; OUTPUT
PIN 15     IRQ[7]    COMBINATORIAL ; OUTPUT
PIN 16     IRQ[6]    COMBINATORIAL ; OUTPUT
PIN 17     IRQ[5]    COMBINATORIAL ; OUTPUT
PIN 18     IRQ[4]    COMBINATORIAL ; OUTPUT
PIN 19     IRQ[1]    COMBINATORIAL ; OUTPUT
PIN 20     IRQ[2]    COMBINATORIAL ; OUTPUT
PIN 21     IRQ[3]    COMBINATORIAL ; OUTPUT
PIN 22     /PIACK    COMBINATORIAL ; OUTPUT
PIN 23     /TIACK    COMBINATORIAL ; OUTPUT

;----- Boolean Equation Segment -----
EQUATIONS

IF PIRQ + TIRQ THEN BEGIN
CASE S[2..0] BEGIN
0 : BEGIN
    IRQ[7..1] = 0
END
1 : BEGIN
    IRQ[1] = 1
    IRQ[7..2] = 0
END
2 : BEGIN
    IRQ[1] = 0
    IRQ[2] = 1
    IRQ[7..3] = 0
END
3 : BEGIN
    IRQ[2..1] = 0
    IRQ[3] = 1
    IRQ[7..4] = 0
END
4 : BEGIN
    IRQ[3..1] = 0
    IRQ[4] = 1
    IRQ[7..5] = 0
END
5 : BEGIN
    IRQ[4..1] = 0
    IRQ[5] = 1
    IRQ[7..6] = 0
END
6 : BEGIN
    IRQ[5..1] = 0
    IRQ[6] = 1
    IRQ[7] = 0
END
7 : BEGIN
    IRQ[6..1] = 0
    IRQ[7] = 1
END
```

```

        END
    END
END
ELSE BEGIN
    IRQ[7..1] = 0
END

IF IACK_IN * AS THEN BEGIN
    IF (PIRQ + TIRQ) * (A[3] **: S[2]) * (A[2] **: S[1]) * (A[1] **: S[0]) THEN BEGIN
        IF PIRQ THEN BEGIN
            IACK_OUT = 0
            PIACK = 1
            TIACK = 0
        END
        ELSE BEGIN
            IACK_OUT = 0
            PIACK = 0
            TIACK = 1
        END
    END
    ELSE BEGIN
        IACK_OUT = 1
        PIACK = 0
        TIACK = 0
    END
END
ELSE BEGIN
    IACK_OUT = 0
    PIACK = 0
    TIACK = 0
END
END
;----- Simulation Segment -----
SIMULATION
;-----

```

Tilstandsligninger

TITLE	INTERRUPT CONTROLLER TIL VME-BUS MOTORCONTROLLER
PATTERN	
REVISION	1.1-B (1.1 havde ombyttet nogle af IRQ[n] signalerne)
AUTHOR	ANDERS STENGAARD SOERENSEN
COMPANY	ODENSE UNIVERSITET, IMADA
DATE	7/1/96

CHIP _U4 PAL22V10

```

PIN 1 S[0] COMB
PIN 2 S[1] COMB
PIN 3 S[2] COMB
PIN 4 /PIRQ COMB
PIN 5 /TIRQ COMB
PIN 6 /AS COMB
PIN 7 NC1 COMB
PIN 8 NC2 COMB
PIN 9 A[3] COMB
PIN 10 A[2] COMB
PIN 11 A[1] COMB
PIN 12 GND
PIN 13 /IACK_IN COMB
PIN 14 /IACK_OUT COMB
PIN 15 IRQ[7] COMB
PIN 16 IRQ[6] COMB
PIN 17 IRQ[5] COMB
PIN 18 IRQ[4] COMB
PIN 19 IRQ[1] COMB
PIN 20 IRQ[2] COMB
PIN 21 IRQ[3] COMB
PIN 22 /PIACK COMB
PIN 23 /TIACK COMB
PIN 24 VCC

```

EQUATIONS

```

/IRQ[7] = /(PIRQ + TIRQ)

```

```

+ /S[0] * S[1] * S[2] * (PIRQ + TIRQ)
+ S[0] * /S[1] * S[2] * (PIRQ + TIRQ)
+ /S[0] * /S[1] * S[2] * (PIRQ + TIRQ)
+ S[0] * S[1] * /S[2] * (PIRQ + TIRQ)
+ /S[0] * S[1] * /S[2] * (PIRQ + TIRQ)
+ S[0] * /S[1] * /S[2] * (PIRQ + TIRQ)
+ /S[0] * /S[1] * /S[2] * (PIRQ + TIRQ)
/IRQ[6] = /(PIRQ + TIRQ)
+ S[0] * S[1] * S[2] * (PIRQ + TIRQ)
+ S[0] * /S[1] * S[2] * (PIRQ + TIRQ)
+ /S[0] * /S[1] * S[2] * (PIRQ + TIRQ)
+ S[0] * S[1] * /S[2] * (PIRQ + TIRQ)
+ /S[0] * S[1] * /S[2] * (PIRQ + TIRQ)
+ S[0] * /S[1] * /S[2] * (PIRQ + TIRQ)
+ /S[0] * /S[1] * /S[2] * (PIRQ + TIRQ)
/IRQ[5] = /(PIRQ + TIRQ)
+ S[0] * S[1] * S[2] * (PIRQ + TIRQ)
+ /S[0] * S[1] * S[2] * (PIRQ + TIRQ)
+ /S[0] * /S[1] * S[2] * (PIRQ + TIRQ)
+ S[0] * S[1] * /S[2] * (PIRQ + TIRQ)
+ /S[0] * S[1] * /S[2] * (PIRQ + TIRQ)
+ S[0] * /S[1] * /S[2] * (PIRQ + TIRQ)
+ /S[0] * /S[1] * /S[2] * (PIRQ + TIRQ)
/IRQ[4] = /(PIRQ + TIRQ)
+ S[0] * S[1] * S[2] * (PIRQ + TIRQ)
+ /S[0] * S[1] * S[2] * (PIRQ + TIRQ)
+ S[0] * /S[1] * S[2] * (PIRQ + TIRQ)
+ S[0] * S[1] * /S[2] * (PIRQ + TIRQ)
+ /S[0] * S[1] * /S[2] * (PIRQ + TIRQ)
+ S[0] * /S[1] * /S[2] * (PIRQ + TIRQ)
+ /S[0] * /S[1] * /S[2] * (PIRQ + TIRQ)
/IRQ[3] = /(PIRQ + TIRQ)
+ S[0] * S[1] * S[2] * (PIRQ + TIRQ)
+ /S[0] * S[1] * S[2] * (PIRQ + TIRQ)
+ S[0] * /S[1] * S[2] * (PIRQ + TIRQ)
+ /S[0] * /S[1] * S[2] * (PIRQ + TIRQ)
+ /S[0] * S[1] * /S[2] * (PIRQ + TIRQ)
+ S[0] * /S[1] * /S[2] * (PIRQ + TIRQ)
+ /S[0] * /S[1] * /S[2] * (PIRQ + TIRQ)
/IRQ[2] = /(PIRQ + TIRQ)
+ S[0] * S[1] * S[2] * (PIRQ + TIRQ)
+ /S[0] * S[1] * S[2] * (PIRQ + TIRQ)
+ S[0] * /S[1] * S[2] * (PIRQ + TIRQ)
+ /S[0] * /S[1] * S[2] * (PIRQ + TIRQ)
+ S[0] * S[1] * /S[2] * (PIRQ + TIRQ)
+ S[0] * /S[1] * /S[2] * (PIRQ + TIRQ)
+ /S[0] * /S[1] * /S[2] * (PIRQ + TIRQ)
/IRQ[1] = /(PIRQ + TIRQ)
+ S[0] * S[1] * S[2] * (PIRQ + TIRQ)
+ /S[0] * S[1] * S[2] * (PIRQ + TIRQ)
+ S[0] * /S[1] * S[2] * (PIRQ + TIRQ)
+ /S[0] * /S[1] * S[2] * (PIRQ + TIRQ)
+ S[0] * S[1] * /S[2] * (PIRQ + TIRQ)
+ /S[0] * S[1] * /S[2] * (PIRQ + TIRQ)
+ /S[0] * /S[1] * /S[2] * (PIRQ + TIRQ)
IRQ[1] = S[0] * /S[1] * /S[2] * (PIRQ + TIRQ)
IRQ[2] = /S[0] * S[1] * /S[2] * (PIRQ + TIRQ)
IRQ[3] = S[0] * S[1] * /S[2] * (PIRQ + TIRQ)
IRQ[4] = /S[0] * /S[1] * S[2] * (PIRQ + TIRQ)
IRQ[5] = S[0] * /S[1] * S[2] * (PIRQ + TIRQ)
IRQ[6] = /S[0] * S[1] * S[2] * (PIRQ + TIRQ)
IRQ[7] = S[0] * S[1] * S[2] * (PIRQ + TIRQ)
/IACK_OUT = /(IACK_IN * AS)
+ /PIRQ * (PIRQ + TIRQ) * (A[3] :: S[2]) *
  (A[2] :: S[1]) * (A[1] :: S[0]) * IACK_IN * AS
+ PIRQ * (PIRQ + TIRQ) * (A[3] :: S[2]) * (A[2]
  :: S[1]) * (A[1] :: S[0]) * IACK_IN * AS
PIACK = PIRQ * (PIRQ + TIRQ) * (A[3] :: S[2]) * (A[2]
  :: S[1]) * (A[1] :: S[0]) * IACK_IN * AS
/TIACK = /(IACK_IN * AS)
+ /((PIRQ + TIRQ) * (A[3] :: S[2]) * (A[2] :: S[1])
  * (A[1] :: S[0])) * IACK_IN * AS
+ PIRQ * (PIRQ + TIRQ) * (A[3] :: S[2]) * (A[2]
  :: S[1]) * (A[1] :: S[0]) * IACK_IN * AS
/PIACK = /(IACK_IN * AS)
+ /((PIRQ + TIRQ) * (A[3] :: S[2]) * (A[2] :: S[1])
  * (A[1] :: S[0])) * IACK_IN * AS
+ /PIRQ * (PIRQ + TIRQ) * (A[3] :: S[2]) * (A[2]
  :: S[1]) * (A[1] :: S[0]) * IACK_IN * AS
TIACK = /PIRQ * (PIRQ + TIRQ) * (A[3] :: S[2]) * (A[2]

```

```

      **: S[1]) * (A[1] **: S[0]) * IACK_IN * AS
IACK_OUT = /((PIRQ + TIRQ) * (A[3] **: S[2]) * (A[2] **: S[1]
           ) * (A[1] **: S[0])) * IACK_IN * AS

```

18.7.3 U3 på V1.2

Kildetekst

```
;PALASM Design Description
```

```
;----- Declaration Segment -----
```

```
TITLE      ADRESSE DEKODER TIL VME-BUS MOTORCONTROLLER
```

```
PATTERN
```

```
REVISION 1.2-L (LONG ADRESSES)
```

```
AUTHOR    ANDERS STENGAARD SOERENSEN
```

```
COMPANY   ODENSE UNIVERSITET, IMADA
```

```
DATE      09/01/94
```

```
CHIP      _U3  PAL22V10
```

```
;----- PIN Declarations -----
```

```

PIN 1      /TIACK      COMBINATORIAL ; INPUT
PIN 2      /PIACK      COMBINATORIAL ; INPUT
PIN 3      /AD_LOW     COMBINATORIAL ; INPUT
PIN 4      AM[3]        COMBINATORIAL ; INPUT
PIN 5      AM[2]        COMBINATORIAL ; INPUT
PIN 6      AM[4]        COMBINATORIAL ; INPUT
PIN 7      AM[1]        COMBINATORIAL ; INPUT
PIN 8      AM[0]        COMBINATORIAL ; INPUT
PIN 9      /IACK       COMBINATORIAL ; INPUT
PIN 10     AM[5]        COMBINATORIAL ; INPUT
PIN 11     /DS0        COMBINATORIAL ; INPUT
PIN 13     A[6]         COMBINATORIAL ; INPUT
PIN 15     A[4]         COMBINATORIAL ; INPUT
PIN 16     /PIT        COMBINATORIAL ; OUTPUT
PIN 17     /MC[4]       COMBINATORIAL ; OUTPUT
PIN 14     A[5]         COMBINATORIAL ; INPUT
PIN 18     /MC[1]       COMBINATORIAL ; OUTPUT
PIN 19     CS          COMBINATORIAL ; OUTPUT
PIN 20     MC[0]        COMBINATORIAL ; OUTPUT
PIN 21     /MC[2]       COMBINATORIAL ; OUTPUT
PIN 22     /MC[3]       COMBINATORIAL ; OUTPUT
PIN 23     /AD_HIGH    COMBINATORIAL ; INPUT

```

```
;----- Boolean Equation Segment -----
```

```
EQUATIONS
```

```
IF DS0 * /IACK * AD_LOW * AD_HIGH * ((AM[5..0] = #h39) + (AM[5..0] = #h3D) + (AM[5..0] = #h00)) THEN BEGIN
```

```
  CS = 1
```

```
  CASE A[6..4] BEGIN
```

```
    0..3 : BEGIN
```

```
      PIT = 1
```

```
      MC[4..0] = 0
```

```
    END
```

```
    4 : BEGIN
```

```
      PIT = 0
```

```
      MC[0] = 1
```

```
      MC[1] = 1
```

```
      MC[4..2] = 0
```

```
    END
```

```
    5 : BEGIN
```

```
      PIT = 0
```

```
      MC[0] = 1
```

```
      MC[1] = 0
```

```
      MC[2] = 1
```

```
      MC[4..3] = 0
```

```
    END
```

```
    6 : BEGIN
```

```
      PIT = 0
```

```
      MC[0] = 1
```

```
      MC[2..1]=0
```

```
      MC[3] = 1
```

```
      MC[4] = 0
```

```
    END
```

```
    7 : BEGIN
```

```
      PIT = 0
```

```
      MC[0] = 1
```

```
      MC[3..1] = 0
```

```
      MC[4] = 1
```

```
  END
```

```

END
END
ELSE BEGIN
    PIT = 0
    MC[4..0] = 0
    CS = TIACK + PIACK
END
;----- Simulation Segment -----
SIMULATION
;-----

```

Tilstandsligninger

```

TITLE      ADRESSE DEKODER TIL VME-BUS MOTORCONTROLLER
PATTERN
REVISION   1.2-L (LONG ADDRESSES)
AUTHOR     ANDERS STENGAARD SOERENSEN
COMPANY    ODENSE UNIVERSITET, IMADA
DATE       09/01/94

```

CHIP _U3 PAL22V10

```

PIN 1 /TIACK COMB
PIN 2 /PIACK COMB
PIN 3 /AD_LOW COMB
PIN 4 AM[3] COMB
PIN 5 AM[2] COMB
PIN 6 AM[4] COMB
PIN 7 AM[1] COMB
PIN 8 AM[0] COMB
PIN 9 /IACK COMB
PIN 10 AM[5] COMB
PIN 11 /DS0 COMB
PIN 12 GND
PIN 13 A[6] COMB
PIN 14 A[5] COMB
PIN 15 A[4] COMB
PIN 16 /PIT COMB
PIN 17 /MC[4] COMB
PIN 18 /MC[1] COMB
PIN 19 CS COMB
PIN 20 MC[0] COMB
PIN 21 /MC[2] COMB
PIN 22 /MC[3] COMB
PIN 23 /AD_HIGH COMB
PIN 24 VCC

```

EQUATIONS

```

MC[3] = /A[4] * A[5] * A[6] * DS0 * /IACK * AD_LOW * AD_HIGH
        * AM[0] * /AM[1] * AM[3] * AM[4] * AM[5]
        + /A[4] * A[5] * A[6] * DS0 * /IACK * AD_LOW * AD_HIGH
        * /AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *
        /AM[5]
MC[2] = A[4] * /A[5] * A[6] * DS0 * /IACK * AD_LOW * AD_HIGH
        * AM[0] * /AM[1] * AM[3] * AM[4] * AM[5]
        + A[4] * /A[5] * A[6] * DS0 * /IACK * AD_LOW * AD_HIGH
        * /AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *
        /AM[5]
MC[0] = A[6] * DS0 * /IACK * AD_LOW * AD_HIGH * AM[0] *
        /AM[1] * AM[3] * AM[4] * AM[5]
        + A[6] * DS0 * /IACK * AD_LOW * AD_HIGH *
        /AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *
        /AM[5]
CS = TIACK
    + PIACK
    + DS0 * /IACK * AD_LOW * AD_HIGH * AM[0] * /AM[1] * AM[3]
    * AM[4] * AM[5]
    + DS0 * /IACK * AD_LOW * AD_HIGH * /AM[0] *
    /AM[1] * /AM[2] * /AM[3] * /AM[4] * /AM[5]
MC[1] = /A[4] * /A[5] * A[6] * DS0 * /IACK * AD_LOW * AD_HIGH
        * AM[0] * /AM[1] * AM[3] * AM[4] * AM[5]
        + /A[4] * /A[5] * A[6] * DS0 * /IACK * AD_LOW * AD_HIGH
        * /AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *

```

```

      /AM[5]
MC[4] = A[4] * A[5] * A[6] * DS0 * /IACK * AD_LOW * AD_HIGH
      * AM[0] * /AM[1] * AM[3] * AM[4] * AM[5]
      + A[4] * A[5] * A[6] * DS0 * /IACK * AD_LOW * AD_HIGH
      * /AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *
      /AM[5]
PIT = /A[6] * DS0 * /IACK * AD_LOW * AD_HIGH * AM[0] *
      /AM[1] * AM[3] * AM[4] * AM[5]
      + /A[6] * DS0 * /IACK * AD_LOW * AD_HIGH *
      /AM[0] * /AM[1] * /AM[2] * /AM[3] * /AM[4] *
      /AM[5]

```

18.7.4 U15 på V1.2

Kildetekst

```
;PALASM Design Description
```

```
;----- Declaration Segment -----
```

```
TITLE TEST 1
```

```
PATTERN
```

```
REVISION
```

```
AUTHOR
```

```
COMPANY
```

```
DATE 01/18/96
```

```
CHIP _test1 PAL22V10
```

```
;----- PIN Declarations -----
```

PIN 1	stop	COMBINATORIAL ; INPUT
PIN 2	sgn1	COMBINATORIAL ; INPUT
PIN 3	mag1	COMBINATORIAL ; INPUT
PIN 4	mag4	COMBINATORIAL ; INPUT
PIN 5	sgn4	COMBINATORIAL ; INPUT
PIN 6	mag3	COMBINATORIAL ; INPUT
PIN 7	sgn3	COMBINATORIAL ; INPUT
PIN 8	sgn2	COMBINATORIAL ; INPUT
PIN 10	mag2	COMBINATORIAL ; INPUT
PIN 11	/polarity	COMBINATORIAL ; INPUT
PIN 9	/reset	COMBINATORIAL ; INPUT
PIN 13	mode	COMBINATORIAL ; INPUT
PIN 14	b4	COMBINATORIAL ; OUTPUT
PIN 15	a2	COMBINATORIAL ; OUTPUT
PIN 16	a4	COMBINATORIAL ; OUTPUT
PIN 17	a1	COMBINATORIAL ; OUTPUT
PIN 18	b3	COMBINATORIAL ; OUTPUT
PIN 19	b1	COMBINATORIAL ; OUTPUT
PIN 20	a3	COMBINATORIAL ; OUTPUT
PIN 21	b2	COMBINATORIAL ; OUTPUT
PIN 22	/dtackin	COMBINATORIAL ; INPUT
PIN 23	dtackout	COMBINATORIAL ; OUTPUT

```
;----- Boolean Equation Segment -----
```

```
EQUATIONS
```

```
IF (stop + reset) THEN BEGIN ; (Hvis stop eller reset er aktiv)
```

```
  a1 = polarity ; alle udgange inaktive under stop
```

```
  a2 = polarity
```

```
  a3 = polarity
```

```
  a4 = polarity
```

```
  b1 = polarity
```

```
  b2 = polarity
```

```
  b3 = polarity
```

```
  b4 = polarity
```

```
END ; (IF)
```

```
ELSE BEGIN ; (Hvis stop er inaktiv)
```

```
  IF /polarity THEN BEGIN ; (hvis der nskes aktivt hje udgange)
```

```
    IF mode THEN BEGIN ; (Hvis POS/NEG udgange nskes)
```

```
      a1 = mag1 * sgn1 ; a1 er aktiv ved positivt fortegn
```

```
      a2 = mag2 * sgn2
```

```
      a3 = mag3 * sgn3
```

```
      a4 = mag4 * sgn4
```

```
      b1 = mag1 * /sgn1 ; b1 er aktiv ved negativt fortegn
```

```
      b2 = mag2 * /sgn2
```

```
      b3 = mag3 * /sgn3
```

```
      b4 = mag4 * /sgn4
```

```
    END
```

```
  ELSE BEGIN ; (IF NOT mode -- Hvis SGN/MAG udgange nskes)
```

```
    a1 = sgn1 ; Alle 8 signaler fres videre uden behandling
```

```

a2 = sgn2
a3 = sgn3
a4 = sgn4
b1 = mag1
b2 = mag2
b3 = mag3
b4 = mag4
END ; (ELSE)
END ; (IF polarity)
ELSE BEGIN
  IF mode THEN BEGIN ; (Hvis POS/NEG udgange nskes)
    a1 = /(mag1 * sgn1) ; a1 er aktiv ved positivt fortegn
    a2 = /(mag2 * sgn2)
    a3 = /(mag3 * sgn3)
    a4 = /(mag4 * sgn4)
    b1 = /(mag1 * /sgn1) ; b1 er aktiv ved negativt fortegn
    b2 = /(mag2 * /sgn2)
    b3 = /(mag3 * /sgn3)
    b4 = /(mag4 * /sgn4)
  END
  ELSE BEGIN ; (IF NOT mode -- Hvis SGN/MAG udgange nskes)
    a1 = /sgn1 ; Alle 8 signaler fres videre uden behandling
    a2 = /sgn2
    a3 = /sgn3
    a4 = /sgn4
    b1 = /mag1
    b2 = /mag2
    b3 = /mag3
    b4 = /mag4
  END ; (ELSE)
END ; (ELSE)
END ; (ELSE)

dtackout = dtackin
;----- Simulation Segment -----
SIMULATION
;-----

```

Tilstandsligninger

```

TITLE      TEST 1
PATTERN
REVISION
AUTHOR
COMPANY
DATE       01/18/96

```

```
CHIP _TEST1 PAL22V10
```

```

PIN 1 STOP COMB
PIN 2 SGN1 COMB
PIN 3 MAG1 COMB
PIN 4 MAG4 COMB
PIN 5 SGN4 COMB
PIN 6 MAG3 COMB
PIN 7 SGN3 COMB
PIN 8 SGN2 COMB
PIN 9 /RESET COMB
PIN 10 MAG2 COMB
PIN 11 /POLARITY COMB
PIN 12 GND
PIN 13 MODE COMB
PIN 14 B4 COMB
PIN 15 A2 COMB
PIN 16 A4 COMB
PIN 17 A1 COMB
PIN 18 B3 COMB
PIN 19 B1 COMB
PIN 20 A3 COMB
PIN 21 B2 COMB
PIN 22 /DTACKIN COMB
PIN 23 DTACKOUT COMB
PIN 24 VCC

```

```
EQUATIONS
```



```

A1 = /SGN1 * /MODE * POLARITY * /(STOP + RESET)
+ /(MAG1 * SGN1) * MODE * POLARITY * /(STOP + RESET)
)
+ SGN1 * /MODE * /POLARITY * /(STOP + RESET)
+ MAG1 * SGN1 * MODE * /POLARITY * /(STOP + RESET)
+ POLARITY * (STOP + RESET)
A2 = /SGN2 * /MODE * POLARITY * /(STOP + RESET)
+ /(MAG2 * SGN2) * MODE * POLARITY * /(STOP + RESET)
)
+ SGN2 * /MODE * /POLARITY * /(STOP + RESET)
+ MAG2 * SGN2 * MODE * /POLARITY * /(STOP + RESET)
+ POLARITY * (STOP + RESET)
A3 = /SGN3 * /MODE * POLARITY * /(STOP + RESET)
+ /(MAG3 * SGN3) * MODE * POLARITY * /(STOP + RESET)
)
+ SGN3 * /MODE * /POLARITY * /(STOP + RESET)
+ MAG3 * SGN3 * MODE * /POLARITY * /(STOP + RESET)
+ POLARITY * (STOP + RESET)
A4 = /SGN4 * /MODE * POLARITY * /(STOP + RESET)
+ /(MAG4 * SGN4) * MODE * POLARITY * /(STOP + RESET)
)
+ SGN4 * /MODE * /POLARITY * /(STOP + RESET)
+ MAG4 * SGN4 * MODE * /POLARITY * /(STOP + RESET)
+ POLARITY * (STOP + RESET)
B1 = /MAG1 * /MODE * POLARITY * /(STOP + RESET)
+ /(MAG1 * /SGN1) * MODE * POLARITY * /(STOP + RESET)
)
+ MAG1 * /MODE * /POLARITY * /(STOP + RESET)
+ MAG1 * /SGN1 * MODE * /POLARITY * /(STOP + RESET)
)
+ POLARITY * (STOP + RESET)
B2 = /MAG2 * /MODE * POLARITY * /(STOP + RESET)
+ /(MAG2 * /SGN2) * MODE * POLARITY * /(STOP + RESET)
)
+ MAG2 * /MODE * /POLARITY * /(STOP + RESET)
+ MAG2 * /SGN2 * MODE * /POLARITY * /(STOP + RESET)
)
+ POLARITY * (STOP + RESET)
B3 = /MAG3 * /MODE * POLARITY * /(STOP + RESET)
+ /(MAG3 * /SGN3) * MODE * POLARITY * /(STOP + RESET)
)
+ MAG3 * /MODE * /POLARITY * /(STOP + RESET)
+ MAG3 * /SGN3 * MODE * /POLARITY * /(STOP + RESET)
)
+ POLARITY * (STOP + RESET)
B4 = /MAG4 * /MODE * POLARITY * /(STOP + RESET)
+ /(MAG4 * /SGN4) * MODE * POLARITY * /(STOP + RESET)
)
+ MAG4 * /MODE * /POLARITY * /(STOP + RESET)
+ MAG4 * /SGN4 * MODE * /POLARITY * /(STOP + RESET)
)
+ POLARITY * (STOP + RESET)
/B4 = MAG4 * /MODE * POLARITY * /(STOP + RESET)
+ MAG4 * /SGN4 * MODE * POLARITY * /(STOP + RESET)
)
+ /MAG4 * /MODE * /POLARITY * /(STOP + RESET)
+ /(MAG4 * /SGN4) * MODE * /POLARITY * /(STOP + RESET)
)
+ /POLARITY * (STOP + RESET)
/B3 = MAG3 * /MODE * POLARITY * /(STOP + RESET)
+ MAG3 * /SGN3 * MODE * POLARITY * /(STOP + RESET)
)
+ /MAG3 * /MODE * /POLARITY * /(STOP + RESET)
+ /(MAG3 * /SGN3) * MODE * /POLARITY * /(STOP + RESET)
)
+ /POLARITY * (STOP + RESET)
/B2 = MAG2 * /MODE * POLARITY * /(STOP + RESET)
+ MAG2 * /SGN2 * MODE * POLARITY * /(STOP + RESET)
)
+ /MAG2 * /MODE * /POLARITY * /(STOP + RESET)
+ /(MAG2 * /SGN2) * MODE * /POLARITY * /(STOP + RESET)
)
+ /POLARITY * (STOP + RESET)
/B1 = MAG1 * /MODE * POLARITY * /(STOP + RESET)
+ MAG1 * /SGN1 * MODE * POLARITY * /(STOP + RESET)
)
+ /MAG1 * /MODE * /POLARITY * /(STOP + RESET)
+ /(MAG1 * /SGN1) * MODE * /POLARITY * /(STOP + RESET)
)

```

```

+ /POLARITY * (STOP + RESET)
/A4 = SGN4 * /MODE * POLARITY * /(STOP + RESET)
+ MAG4 * SGN4 * MODE * POLARITY * /(STOP + RESET)
+ /SGN4 * /MODE * /POLARITY * /(STOP + RESET)
+ /(MAG4 * SGN4) * MODE * /POLARITY * /(STOP + RESET)
)
+ /POLARITY * (STOP + RESET)
/A3 = SGN3 * /MODE * POLARITY * /(STOP + RESET)
+ MAG3 * SGN3 * MODE * POLARITY * /(STOP + RESET)
+ /SGN3 * /MODE * /POLARITY * /(STOP + RESET)
+ /(MAG3 * SGN3) * MODE * /POLARITY * /(STOP + RESET)
)
+ /POLARITY * (STOP + RESET)
/A2 = SGN2 * /MODE * POLARITY * /(STOP + RESET)
+ MAG2 * SGN2 * MODE * POLARITY * /(STOP + RESET)
+ /SGN2 * /MODE * /POLARITY * /(STOP + RESET)
+ /(MAG2 * SGN2) * MODE * /POLARITY * /(STOP + RESET)
)
+ /POLARITY * (STOP + RESET)
/A1 = SGN1 * /MODE * POLARITY * /(STOP + RESET)
+ MAG1 * SGN1 * MODE * POLARITY * /(STOP + RESET)
+ /SGN1 * /MODE * /POLARITY * /(STOP + RESET)
+ /(MAG1 * SGN1) * MODE * /POLARITY * /(STOP + RESET)
)
+ /POLARITY * (STOP + RESET)
DTACKOUT = DTACKIN

```

18.8 Kildetekst til motor testprogram

I dette afsnit gengives kildeteksten til testprogrammet motor, der blev brugt ved afprøvningen af Gefion. Programmet kompiles fra kildeteksten **constants.h**, **motor.h**, og **motor.c** vha. kommandoen: `cc motor.c`

18.8.1 constants.h

```
#define BASIS_A 0x873e0400
```

18.8.2 motor.h

```
#include "constants.h"

#define PIT_A BASIS_A

#define A1 BASIS_A + 0x41
#define A2 BASIS_A + 0x51
#define A3 BASIS_A + 0x61
#define A4 BASIS_A + 0x71

#define STATUS(A)    *(unsigned char *) (A)
#define COMMAND(A)  *(unsigned char *) (A)
#define DATA(A)     *(unsigned char *) (A + 0x02)

#define WAIT(A) do {} while (STATUS(A) & 0x01)

#define READ_BYTE(A)    DATA(A)
#define WRITE_BYTE(A,B) DATA(A) = ( B )

#define READ_WORD(A)    READ_BYTE(A)<<8 | READ_BYTE(A)
#define WRITE_WORD(A,B) WRITE_BYTE(A,( B )/0x100); WRITE_BYTE(A,( B )%0x100)

/* Kommandoer */
#define RESET    0x00
#define PORT8    0x05
#define PORT12   0x06
#define DFH      0x02
#define SIP      0x03
#define LPEI     0x1B
#define LPES     0x1A
#define SBPA     0x20
#define SBPR     0x21
#define MSKI     0x1C
#define RSTI     0x1D
#define LFIL     0x1E
#define UDF      0x04
#define LTRJ     0x1F
#define STT      0x01
#define RDSIGS   0x0C
#define RDIP     0x09
#define RDDP     0x08
#define RDRP     0x0A
#define RDDV     0x07
#define RDRV     0x0B
#define RDSUM    0x0D

/* Adresser i MC68230 PI/T */
#define PGCR(P)  *(unsigned char *) (P + 0x01)
#define PSRR(P)  *(unsigned char *) (P + 0x03)
#define PADDR(P) *(unsigned char *) (P + 0x04)
#define PBDDR(P) *(unsigned char *) (P + 0x07)
#define PCDDR(P) *(unsigned char *) (P + 0x09)
#define PIVR(P)  *(unsigned char *) (P + 0x0B)
#define PACR(P)  *(unsigned char *) (P + 0x0D)
#define PBCR(P)  *(unsigned char *) (P + 0x0F)
#define PADR(P)  *(unsigned char *) (P + 0x11)
#define PBDR(P)  *(unsigned char *) (P + 0x13)
#define PAAR(P)  *(unsigned char *) (P + 0x15)
#define PBAR(P)  *(unsigned char *) (P + 0x17)
#define PCDR(P)  *(unsigned char *) (P + 0x19)
#define PSR(P)   *(unsigned char *) (P + 0x1B)
#define TCR(P)   *(unsigned char *) (P + 0x21)
```

```
#define TIVR(P) *(unsigned char *)(P + 0x23)
#define CPRH(P) *(unsigned char *)(P + 0x27)
#define CPRM(P) *(unsigned char *)(P + 0x29)
#define CPRL(P) *(unsigned char *)(P + 0x2B)
#define CNTRH(P) *(unsigned char *)(P + 0x2F)
#define CNTRM(P) *(unsigned char *)(P + 0x31)
#define CNTRL(P) *(unsigned char *)(P + 0x33)
#define TSR(P) *(unsigned char *)(P + 0x35)
```

18.8.3 motor.c

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

void test(ADR);

void reset(ADR)
unsigned long ADR;
{
    WAIT(ADR);
    COMMAND(ADR) = RESET;
}

void load_filter(ADR,interval,p,i,d,limit)
unsigned long ADR;
unsigned short interval,p,i,d,limit;
{
    WAIT(ADR);
    COMMAND(ADR) = LFIL;
    WAIT(ADR);
    WRITE_WORD(ADR,interval * 256 + 15);
    WAIT(ADR);
    WRITE_WORD(ADR,p);
    WAIT(ADR);
    WRITE_WORD(ADR,i);
    WAIT(ADR);
    WRITE_WORD(ADR,d);
    WAIT(ADR);
    WRITE_WORD(ADR,limit);
}

unsigned long read_long(ADR)
unsigned long ADR;
{
    unsigned short buffer;

    WAIT(ADR);
    buffer = READ_WORD(ADR);
    WAIT(ADR);
    return buffer<<16 | READ_WORD(ADR);
}

void write_long(ADR,num)
unsigned long ADR,num;
{
    WAIT(ADR);
    WRITE_WORD(ADR,num / 0x10000);
    WAIT(ADR);
    WRITE_WORD(ADR,num % 0x10000);
}

void init(ADR)
unsigned long ADR;
{
    reset(ADR);
    WAIT(ADR);
    COMMAND(ADR) = LPES;
    WAIT(ADR);
    WRITE_WORD(ADR,1000);
    WAIT(ADR);
    COMMAND(ADR) = RSTI;
    WAIT(ADR);
    WRITE_WORD(ADR,0);
    WAIT(A1);
    COMMAND(ADR) = RDSIGS;
    WAIT(ADR);
    printf("INIT status: %x \n",READ_WORD(ADR));
}
```

```

}

void main()
{
    int pos,n;
    unsigned int p,i,d;
    float spd,acc;
    unsigned long adr;

    printf(" Hvilken LM629 (1,2,3,4)\n");
    scanf("%d",&n);
    if (n==4) adr=A4;
    if (n==3) adr=A3;
    if (n==2) adr=A2;
    else adr=A1;

    test(adr);
    init(adr);

    printf(" Indtast P, I, og D parametre \n");
    scanf("%d %d %d",&p,&i,&d);
    load_filter(adr,1,p,i,d,30000);
    WAIT(adr);
    COMMAND(adr) = UDF;
    printf(" Indtast position, Hastighed, og acceleration \n");
    scanf("%d %f %f",&pos,&spd,&acc);
    WAIT(adr);
    COMMAND(adr) = LTRJ;
    WAIT(adr);
    WRITE_WORD(adr,0x2a);
    write_long(adr,(unsigned long)(acc*0x10000));
    write_long(adr,(unsigned long)(spd*0x10000));
    write_long(adr,pos);
    WAIT(adr);
    COMMAND(adr)=STT;
}

```

18.9 Kildetekster til MCI bibliotek

I dette afsnit gengives kildeteksterne til MCI funktionsbiblioteket, og testprogrammet **test**, samt en makefil, til at bygge begge dele med.

18.9.1 makefile

Denne makefil er brugt til at bygge funktionsbiblioteket: **MCI**, der udgør software interfacet til Gefion. Den har været anvendt af bruger: **zaphod**, med hjemmekatalog i /dd/USR/ZAPHOD. Den skal bruge underkatalogerne: MCI_SOURCE, RELS, EXE, og LIB; og ligger selv i MCI_SOURCE sammen med kildeteksterne.

```

MYDIR = /dd/usr/zaphod

ODIR = $(MYDIR)/EXE
SDIR = $(MYDIR)/MCI_SOURCE
RDIR = $(MYDIR)/RELS

LDIR = $(MYDIR)/LIB

CFLAGS =
RFLAGS =
LFLAGS =

#
# MCI
#

test: test.r $(LDIR)/MCI.l
    cc $(RDIR)/$*.r -l=$(LDIR)/MCI.l -f=$(ODIR)/$*

test.r: test.c

```

```

MCI:                $(LDIR)/MCI.l
                    rdump -l $(LDIR)/MCI.l

$(LDIR)/MCI.l:      MCI.r MCIhardware.r MCIOs9.r
                    -del $(LDIR)/MCI.l
                    merge $(RDIR)/mci.r $(RDIR)/mcihardware.r $(RDIR)/mcios9.r >$(LDIR)/MCI.l

MCI.r:             MCI.c MCI.h MCIhardware.h
                    cc $(SDIR)/MCI.c -k2 -r=$(RDIR)

MCIhardware.r:     MCIhardware.c MCIhardware.h
                    cc $(SDIR)/MCIhardware.c -k2 -r=$(RDIR)

MCIOs9.r:          MCIOs9.c MCIOs9.h
                    cc $(SDIR)/MCIOs9.c -k2 -r=$(RDIR)

#
# motor
#

motor:             motor.r anders.r
                    cc $(RDIR)/$.r anders.r -f=$(ODIR)/$*

motor.r:           motor.c motor.h constants.h
                    cc $(SDIR)/motor.c -k2 -r=$(RDIR)

```

18.9.2 MCIhardware.h

Definitionsmodul til den hardwarespecifikke del af koden i MCI biblioteket.

```

/*****
----- MCHhardware.h -----

Definitions modul til hardware specifikke rutiner

---- Denne fil indeholder prototyper til ----

+ MCI_BusError
+ MCI_MotorControllerPresent

*****/

#define BUSY 0x01

#define STATUS(m)  *(unsigned char *) (m)
#define COMMAND(m) *(unsigned char *) (m)
#define DATA(m)   *(unsigned char *) (m + 0x02)

#define AWAIT_NOT_BUSY(m) do {} while (STATUS(m) & BUSY)

#define READ_BYTE(m)      DATA(m)
#define WRITE_BYTE(m,val) DATA(m) = (val)
#define WAIT_READ_BYTE(m,val) AWAIT_NOT_BUSY(m); \
                               val = READ_BYTE(m)
#define WAIT_WRITE_BYTE   AWAIT_NOT_BUSY(m); \
                               WRITE_BYTE(m,val)

#define READ_WORD(m)      (READ_BYTE(m) << 8 | READ_BYTE(m))
#define WRITE_WORD(m,val) WRITE_BYTE(m, (val)>>8); \
                               WRITE_BYTE(m, (val) & 0xff);
#define WAIT_READ_WORD(m,val) AWAIT_NOT_BUSY(m); \
                               val = READ_WORD(m)
#define WAIT_WRITE_WORD(m,val) AWAIT_NOT_BUSY(m); \
                               WRITE_WORD(m,val)

#define READ_LONG(m)      MCI_ReadLong(m)
#define WRITE_LONG(m,val) MCI_WriteLong(m,val)
#define WAIT_READ_LONG(m,val) AWAIT_NOT_BUSY(m); \
                               val = READ_LONG(m)

```

```

#define WAIT_WRITE_LONG(m,val)  AWAIT_NOT_BUSY(m);\
                                WRITE_LONG(m,val)

/* LM629 kommandoer */
#define RESET    0x00
#define DFH      0x02
#define SIP      0x03
#define LPEI     0x1B
#define LPES     0x1A
#define SBPA     0x20
#define SBPR     0x21
#define MSKI     0x1C
#define RSTI     0x1D
#define LFIL     0x1E
#define UDF      0x04
#define LTRJ     0x1F
#define STT      0x01
#define RDSIGS   0x0C
#define RDIP     0x09
#define RDDP     0x08
#define RDRP     0x0A
#define RDDV     0x07
#define RDRV     0x0B
#define RDSUM    0x0D

/* Adresser i MC68230 PI/T */
#define PGCR(pit)  *(unsigned char *) (pit + 0x00)
#define PSRR(pit)  *(unsigned char *) (pit + 0x02)
#define PADDR(pit) *(unsigned char *) (pit + 0x04)
#define PBDDR(pit) *(unsigned char *) (pit + 0x06)
#define PCDDR(pit) *(unsigned char *) (pit + 0x08)
#define PIVR(pit)  *(unsigned char *) (pit + 0x0A)
#define PACR(pit)  *(unsigned char *) (pit + 0x0C)
#define PBCR(pit)  *(unsigned char *) (pit + 0x0E)
#define PADR(pit)  *(unsigned char *) (pit + 0x10)
#define PBDR(pit)  *(unsigned char *) (pit + 0x12)
#define PAAR(pit)  *(unsigned char *) (pit + 0x14)
#define PBAR(pit)  *(unsigned char *) (pit + 0x16)
#define PCDR(pit)  *(unsigned char *) (pit + 0x18)
#define PSR(pit)   *(unsigned char *) (pit + 0x1A)
#define TCR(pit)   *(unsigned char *) (pit + 0x20)
#define TIVR(pit)  *(unsigned char *) (pit + 0x22)
#define CPRH(pit)  *(unsigned char *) (pit + 0x26)
#define CPRM(pit)  *(unsigned char *) (pit + 0x28)
#define CPRL(pit)  *(unsigned char *) (pit + 0x2A)
#define CNTRH(pit) *(unsigned char *) (pit + 0x2E)
#define CNTRM(pit) *(unsigned char *) (pit + 0x30)
#define CNTRL(pit) *(unsigned char *) (pit + 0x32)
#define TSR(pit)   *(unsigned char *) (pit + 0x34)

int MCI_BusError(addr);
/* Tester om der opstår en BUS ERROR ved læsning fra adressen: addr.
   Returnerer 1 hvis der opstår en BUS ERROR
   Returnerer 0 hvis der ikke opstår en BUS ERROR */

int MCI_MotorControllerPresent(addr);
/* Tester om der er placeret en LM 628/629 på adressen: addr.
   Returnerer 1 hvis der gør.
   Returnerer 0 hvis der ikke gør */

unsigned long MCI_ReadLong(addr);
/* Læser et 'longword' fra en LM629 motorcontroller IC */

void MCI_WriteLong(addr,val);
/* Skriver 'longwordet' val til en LM629 motorcontroller IC */

```

18.9.3 MCJos9.h

Definitionsmodul til den OS-9 specifikke del af koden i MCI biblioteket.

```

/*****
----- MCJos9.h -----

```

```

Definitions modul til OS-9 specifikke funktioner

---- Denne fil indeholder prototyper til ----

+ MCI_LinkSemaphore
+ MCI_UnlinkSemaphore
+ MCI_OpenSemaphore
+ MCI_CloseSemaphore

*****/

int MCI_LinkSemaphore(name);
/* Hvis der findes en semafor med navnet: name; linkes der til den, og dens
   ID returneres.
   Hvis der ikke findes en, oprettes der en nulstillet semafor med
   navnet: name, og dens ID returneres. */

void MCI_UnlinkSemaphore(semid);
/* Der unlinks fra semaforen med ID: semid */

void MCI_OpenSemaphore(semid);
/* Semaforen med ID: semid åbnes */

void MCI_CloseSemaphore(semid);
/* Der ventes til semaforen med ID: semid - er åben, hvorefter den lukkes. */

```

18.9.4 MCI.h

Definitionsmodul til de funktioner biblioteket stiller til rådighed.

```

/*****
----- MCI.h -----

Definitions modul til inteface rutiner til Gefion motorcontroller

---- Denne fil indeholder prototyper til ----

+ MCI_Link
+ MCI_ReleaseKey
+ MCI_InitParalelInput
+ MCI_ParalelInput
+ MCI_SeizeTimer
+ MCI_ReleaseTimer
+ MCI_InitTimer
+ MCI_ResetMotor
+ MCI_LoadFilter
+ MCI_StopMotor
+ MCI_PositionMode
+ MCI_VelocityMode
+ MCI_LoadPosError
+ MCI_DefineHome
+ MCI_SetIndexPos
+ MCI_SetBreakpoint
+ MCI_BreakpointReached
+ MCI_TrajectoryComplete
+ MCI_PositionError
+ MCI_IndexPulse
+ MCI_MaskInt
+ MCI_ResetInt
+ MCI_UpdateFilter
+ MCI_StartMotion
+ MCI_ReadStatus
+ MCI_ReadSignalsReg
+ MCI_ReadIndexPos
+ MCI_ReadDesiredPos
+ MCI_ReadRealPos
+ MCI_ReadDesiredVelocity
+ MCI_ReadRealVelocity

*****/

#define BYTE unsigned char

```



```

#define WORD unsigned short
#define LONGWORD unsigned long

#define MCI_KEYTYPE struct MCI_key

struct MCI_key
{
    LONGWORD address;
    char semname[10];
    int semID;
    int acces;
};

#define NOMOTOR 0

#define ACC_ABS 0x0020
#define VEL_ABS 0x0008
#define POS_ABS 0x0002
#define ACC_REL 0x0030
#define VEL_REL 0x000C
#define POS_REL 0x0003

#define BACKWARD 0x1000
#define FORWARD 0x0000

#define BPI 0x0040
#define PEI 0x0020
#define WAI 0x0010
#define IPI 0x0008
#define TCI 0x0004
#define CEI 0x0002
#define ALLI 0x007D

#define INT 0x1001
#define STOP 0x1002

#define REL 0x1003
#define ABS 0x1004

#define SMOOTH 0x0400
#define ABRUPT 0x0200
#define OFF 0x0100

void MCI_Link(basis_adr,pit,mc1,mc2,mc3,mc4);

/* unsigned long      basis_adr
   MCI_KEYTYPE*       pit
   MCI_KEYTYPE*       mc1,mc2,mc3,mc4

basis_adr      Controller kortets basisadresse
pit           Pointer til den nøgle der skal identificere kortets MC68230 PIT
mc1..mc4      Pointer til den nøgle der skal identificere hver af de 4 LM629 motorcontrollere

Funktionen undersøger, overfladisk, om der findes et kort på den angivne
basisadresse. Hvis det er tilfældet initialiseres *pit.

Derefter undersøges kortets bestykning af LM629 motorcontroller ICer, og
*mc1..*mc4 initialiseres.

Funktionen lænker den kaldende proces sammen med 1..5 OS-9 events, der
anvendes til semaforbeskyttelse af adgangen til motorcontrollere og timer.
Når processen ikke længere ønsker adgang til controller kortet, bør den
frigive de aktuelle OS-9 events, ved at kalde MCI_ReleaseKey, med adressen
på hver af de anvendte nøgler.

- Typisk anvendelse -

MCI_Link(0x873e0300,&pit,&motor1,&motor2,&motor3,&motor4);

-----O----- */

void MCI_ReleaseKey(key);

/* MCI_KEYTYPE* key

key      Pointer til nøgle

```

Funktionen frigør processen fra det OS-9 event, der implementerer semafor-beskyttelse af den ressource der repræsenteres af nøglen: *key
 Nøglen: *key 'ødelægges', så den ikke længere giver adgang til interface rutinerne.

- Typisk anvendelse -

```
MCI_ReleaseKey(&motor1);
```

```
-----O----- */
```

```
void MCI_InitParalelInput(pitkey);
```

```
/* MCI_KEYTYPE* pitkey
```

```
pitkey Pointer til den nøgle der giver adgang til den ønskede MC68230 PIT
```

```
Funktionen initialiserer parallel IO delen af den givne MC68230 PIT
```

```
-----O----- */
```

```
unsigned short MCI_ParalelInput(pitkey);
```

```
/* MCI_KEYTYPE* pitkey
```

```
pitkey Pointer til den nøgle der giver adgang til den ønskede MC68230 PIT
```

```
De 16 bits i funktionens returnværdi korresponderer med de 16 digitale indgange på motorcontrolleren. En lav bit svarer til en upåvirket indgang.
```

```
-----O----- */
```

```
void MCI_SeizeTimer(pitkey);
```

```
/* MCI_KEYTYPE* pitkey
```

```
pitkey Pointer til den nøgle der giver adgang til den ønskede MC68230 PIT
```

```
Funktionen venter til den semafor der beskytter timer-delen af den pågældende MC68230 PIT er 'åben', hvorefter den 'lukker' semaforen, og sætter et flag i *pitkey, der signalerer at *pitkey giver adgang til at skrive til timer registre.
```

```
-----O----- */
```

```
void MCI_ReleaseTimer(pitkey);
```

```
/* MCI_KEYTYPE* pitkey
```

```
pitkey pointer til den nøgle der giver adgang til den ønskede MC68230 PIT
```

```
Funktionen undersøger om *pitkey giver adgang til at skrive til timer registre i den pågældende MC68230 PIT.
```

```
Hvis det er tilfældet 'åbner' den den semafor der beskytter timer-delen af MC68230 PITen, og nulstiller det flag i *pitkey der signalerer at nøglen giver adgang til at skrive i timer registre.
```

```
Hvis det ikke er tilfældet afbrydes programafviklingen med en fejlmeddelelse.
```

```
-----O----- */
```

```
void MCI_InitTimer(pitkey);
```

```
/* MCI_KEYTYPE* pitkey
```

```
pitkey pointer til den nøgle der giver adgang til den ønskede MC68230 PIT
```

```
Funktionen undersøger om *pitkey giver adgang til at skrive til timer registre i den pågældende MC68230 PIT.
```

```
Hvis det er tilfældet initialiseres timer delen i den pågældende MC68230
```

```
Hvis det ikke er tilfældet afbrydes programmet med en fejlmeddelelse.
```

```
-----O----- */
```

```
void MCI_ResetMotor(mcikey);
```

```
void MCI_LoadFilter(mcikey, sample_int, Kp, Ki, Kd, il);
```

```
void MCI_StopMotor(mcikey, stop_profile);
```

```

void MCI_PositionMode(mcikey, control_word, acc, vel, pos);
void MCI_VelocityMode(mcikey, control_word, vel, acc);
void MCI_LoadPosError(mcikey, error, action);
void MCI_DefineHome(mcikey);
void MCI_SetIndexPos(mcikey);
void MCI_SetBreakpoint(mcikey, breakpoint, breakpoint_type);
int MCI_BreakPointReached(mcikey);
int MCI_TrajectoryComplete(mcikey);
int MCI_PositionError(mcikey);
int MCI_IndexPulse(mcikey);
void MCI_MaskInt(mcikey, interrupts);
void MCI_ResetInt(mcikey, interrupts);
void MCI_UpdateFilter(mcikey);
void MCI_StartMotion(mcikey);
BYTE MCI_ReadStatus(mcikey);
WORD MCI_ReadSignalsReg(mcikey);
long MCI_ReadIndexPos(mcikey);
long MCI_ReadDesiredPos(mcikey);
long MCI_ReadRealPos(mcikey);
double MCI_ReadDesiredVelocity(mcikey);
short MCI_ReadRealVelocity(mcikey);
WORD MCI_ReadIntegrationSum(mcikey);

```

18.9.5 MCIhardware.c

Implementationsmodul til den hardware-specifikke del af koden i MCI biblioteket.

```

/*****
----- MCHhardware.c -----

Implementations modul til hardware specifikke rutiner

---- Denne fil indeholder kildekode til ----

+ MCI_BusError
+ f_strap          (assembler)
+ f_rtrap          (assembler)
+ probe_byte       (assembler)
+ MCI_MotorControllerPresent

*****/

#include "MCIhardware.h"

#define u_int unsigned int      /* bruges i MACHINE/reg.h */
#define u_short unsigned short
#define u_char unsigned char

#include <MACHINE/reg.h>        /* 68000 register struktur */
#include <errno.h>

#define ERROR (-1)

REGISTERS stack_frame;        /* structure for stack frame */

/* Prototyper til maskinkodefunktioner brugt af MCI_BusError */
int f_strap(), f_rtrap(), probe_byte();

int MCI_BusError(addr)
void* addr;
{
    int val;
    if (f_strap(&stack_frame) == ERROR) /* install trap handler */
        exit(_errmsg(errno, "Cant install handler\n"));

    val = probe_byte(addr); /* test the address */

    if (f_rtrap(&stack_frame) == ERROR) /* remove trap handler */
        exit(_errmsg(errno, "Cant remove handler\n"));

    return val;
}

int MCI_MotorControllerPresent(addr)
{

```

```

    return 1;
}

unsigned long MCI_ReadLong(adr)
char* adr;
{
    unsigned long temp;
    temp = READ_WORD(adr) << 16;
    AWAIT_NOT_BUSY(adr);
    return (temp | READ_WORD(adr));
}

void MCI_WriteLong(adr,val)
char* adr;
unsigned long val;
{
    WRITE_WORD(adr,val >> 16);
    AWAIT_NOT_BUSY(adr);
    WRITE_WORD(adr,val & 0xffff);
}

/* --- Assembler koden er kopieret fra The OS-9 Guru-1 --- */

#asm
f_strap:      movem.l d1/a0-a1,-(a7)    save registers
              lea      ExcpTbl(pc),a1  point at table of handlers
              tst.l    d0              any stack given?
              beq.s     f_strap10      ..no
              addi.l    #R$Size-2,d0   convert to pointer to top of stack
f_strap10     movea.l    d0,a0          copy top of stack address
              os9       F$STrap        make the system call
              bcs.s     f_strap20      ..error
              moveq     #0,d0          show no error
              bra.s     f_strap30      ..and return
f_strap20     move.l     d1,errno(a6)   save errorcode
              moveq     #-1,d0         show error occurred
f_strap30     movem.l    (a7)+,d1/a0-a1 retrieve registers
              rts                    return

f_rtrap:      movem.l d1/a0-a1,-(a7)    save registers
              lea      NoHand(pc),a1   point at table of handlers
              tst.l    d0              any stack given?
              beq.s     f_rtrap10      ..no
              addi.l    #R$Size-2,d0   convert to pointer to top of stack
f_rtrap10     movea.l    d0,a0          copy top of stack address
              os9       F$STrap        make the system call
              bcs.s     f_rtrap20      ..error
              moveq     #0,d0          show no error
              bra.s     f_rtrap30      ..and return
f_rtrap20     move.l     d1,errno(a6)   save errorcode
              moveq     #-1,d0         show error occurred
f_rtrap30     movem.l    (a7)+,d1/a0-a1 retrieve registers
              rts                    return

probe_byte:   move.l     a0,-(a7)       save register
              move.l     d0,a0          copy address to use
              move.b     (a0),d0        read byte

              moveq     #0,d0           show no bus error

probe_byte10  movea.l    (a7)+,a0       retrieve register
              rts                    return

bus_hand:     movea.l    a1,a7          restore stack pointer
              movem.l    (a5),d0-d7/a0-a4
              movea.l    R$a5(a5),a5    restore a5 from stack frame
              moveq     #1,d0          show bus error occurred
              bra.s     probe_byte10    ..finish off

/*Table of execeptions to handle, and handler offsets:

ExcpTbl       dc.w      T_BusErr,bus_hand-4
              dc.w      -1              end of table marker

NoHand        dc.w      T_BusErr,0
              dc.w      -1

#endasm

```

18.9.6 MCIOs9.c

Implementationsmodul til den OS-9 specifikke del af koden i MCI biblioteket.

```

/*****

        ---- MCIOs9.c ----

        Implementations modul til OS-9 specifikke funktioner

        ---- Denne fil indeholder kildekode til ----

+ MCI_CreateSemaphore

*****/

#include <events.h>
#include <errno.h>

#define ERROR (-1)

char errortext[80];

int MCI_LinkSemaphore(name)
char *name;
{
    int eventid;
    eventid = _ev_creat(0,1,-1,name);
    if (eventid == ERROR)          /* Hvis det ikke kunne oprettes */
    {
        eventid = _ev_link(name);
        if (eventid == ERROR)
        {
            sprintf(errortext,"MCI_LinkSemaphore: Couldn't create - or link to - %9s\n",name);
            exit(_errmsg(1,errortext));
        }
    }
    return eventid;
}

void MCI_UnlinkSemaphore(semid)
int semid;
{
    if (_ev_unlink(semid)==ERROR)
    {
        sprintf(errortext,"MCI_UnlinkSemaphore: Couldn't unlink event: %x",semid);
        exit(_errmsg(1,errortext));
    }
}

void MCI_OpenSemaphore(semid)
int semid;
{
    if (_ev_signal(semid,0) == ERROR)
    {
        sprintf(errortext,"MCI_OpenSemaphore: Couldn't signal event with ID %x\n",semid);
        exit(_errmsg(errno,errortext));
    }
}

void MCI_CloseSemaphore(semid)
int semid;
{
    if (_ev_wait(semid,0,0) == ERROR)
    {
        sprintf(errortext,"MCI_CloseSemaphore: Couldn't wait for event with ID %x\n",semid);
        exit(_errmsg(errno,errortext));
    }
}

```

18.9.7 MCI.c

Implementationsmodul til de funktioner biblioteket stiller til rådighed.

```

/*****

```

```

----- MCI.c -----

Implementation modul til interface rutiner til Gefion motorcontroller

----- Denne fil indeholder kildetekst til -----

+ MCI_Link
+ MCI_ReleaseKey
+ MCI_InitParalelInput
+ MCI_ParalelInput
+ MCI_SeizeTimer
+ MCI_ReleaseTimer
- MCI_InitTimer
+ MCI_ResetMotor
+ MCI_LoadFilter
+ MCI_StopMotor
+ MCI_PositionMode
+ MCI_VelocityMode
+ MCI_LoadPosError
+ MCI_DefineHome
+ MCI_SetIndexPos
+ MCI_SetBreakpoint
+ MCI_BreakpointReached
+ MCI_TrajectoryComplete
+ MCI_PositionError
+ MCI_IndexPulse
+ MCI_MaskInt
+ MCI_ResetInt
+ MCI_UpdateFilter
+ MCI_StartMotion
+ MCI_ReadStatus
+ MCI_ReadSignalsReg
+ MCI_ReadIndexPos
+ MCI_ReadDesiredPos
+ MCI_ReadRealPos
+ MCI_ReadDesiredVelocity
+ MCI_ReadRealVelocity

*****/

#include "MCI.h"
#include "MCIhardware.h"
#include "MCIOs9.h"

#define KEYDEF                MCI_KEYTYPE* mcikey

#define DEF_ADDRESS            unsigned long address;\
                                address = mcikey->address

#define VALIDATE(key,text)    if (key->address == NOMOTOR)\
                                exit(_errmsg(1,"text Key not valid!\n"));

#define COMMAND_HEAD(cmd,errtxt)  VALIDATE(mcikey,errtxt);\
                                MCI_CloseSemafore(mcikey->semID);\
                                AWAIT_NOT_BUSY(mcikey->address);\
                                COMMAND(mcikey->address) = cmd

#define COMMAND_TAIL          MCI_OpenSemafore(mcikey->semID)

#define TEST_BODY(flag)        KEYDEF;\
                                {\
                                VALIDATE(mcikey,MCI_PositionError);\
                                return (STATUS(mcikey->address) & flag) != 0;\
                                }

void MCI_Link(basis_adr,pit,mc1,mc2,mc3,mc4)
LONGWORD basis_adr;
MCI_KEYTYPE *pit,*mc1,*mc2,*mc3,*mc4;
{
    int n;
    MCI_KEYTYPE *m[5];
    char errortext[80];

    m[0] = pit; m[1] = mc1; m[2] = mc2; m[3] = mc3; m[4] = mc4;

    if (MCI_BusError(basis_adr + 0x41))
    {
        sprintf(errortext,"MCI_Link: No response at basis: %08x hex  (BUS ERROR detected!!)\n",basis_adr);
    }
}

```

```

    exit(_errmsg(1, errortext));
}
for (n=0; n<5; n++)
{
    m[n]->acces = 0;
    if (m[n] != NOMOTOR)
    {
        if (n!=0)    m[n]->address = basis_adr+0x31+0x10*n;
        else        m[n]->address = basis_adr+1;

        if (MCI_MotorControllerPresent(m[n]->address))
        {
            sprintf(m[n]->semname, "mci_%05x", ((m[n]->address)&0xfffff0)>>4);
            m[n]->semID = MCI_LinkSemaphore(m[n]->semname);
        }
        else
            m[n]->address = NOMOTOR;
    }
}

void MCI_ReleaseKey(key)
MCI_KEYTYPE* key;
{
    MCI_UnlinkSemaphore(key->semID);
    key->address = NOMOTOR;
    sprintf(key->semname, "NOMOTOR!!");
    key->semID = -1;
    key->acces = 0;
}

void MCI_InitParalelInput(pitkey)
MCI_KEYTYPE* pitkey;
{
    LONGWORD basis = pitkey->address;

    PGCR(basis) = 0x00;    /* Mode 0, handshake disabled */
    PSRR(basis) = 0x00;    /*
    PACR(basis) = 0x80;    /* Submode 1X
    PBCR(basis) = 0x80;    /* Submode 1X
    PADDR(basis) = 0x00;    /* All input
    PBDDR(basis) = 0x00;    /* All input

}

WORD MCI_ParalelInput(pitkey)
MCI_KEYTYPE* pitkey;
{
    LONGWORD basis = pitkey->address;

    return (PBDR(basis) << 8 | PADR(basis));
}

void MCI_SeizeTimer(pitkey)
MCI_KEYTYPE* pitkey;
{
    MCI_CloseSemaphore(pitkey->semID);
    pitkey->acces = 1;
}

void MCI_ReleaseTimer(pitkey)
MCI_KEYTYPE* pitkey;
{
    MCI_OpenSemaphore(pitkey->semID);
    pitkey->acces = 0;
}

void MCI_InitTimer(pitkey)
MCI_KEYTYPE* pitkey;
{
}

void MCI_ResetMotor(mcikey)
KEYDEF;
{
    COMMAND_HEAD(RESET, MCI_ResetMotor:);
    COMMAND_TAIL;
}

void MCI_LoadFilter(mcikey, sample_int, Kp, Ki, Kd, il)

```

```

KEYDEF;
WORD sample_int,Kp,Ki,Kd,il;
{
    DEF_ADDRESS;
    COMMAND_HEAD(LFIL,MCI_LoadFilter:);
    WAIT_WRITE_WORD(address,(sample_int<<8)+0x0F);
    WAIT_WRITE_WORD(address,Kp);
    WAIT_WRITE_WORD(address,Ki);
    WAIT_WRITE_WORD(address,Kd);
    WAIT_WRITE_WORD(address,il);
    COMMAND_TAIL;
}

void MCI_StopMotor(mcikey,stop_profile)
KEYDEF;
WORD stop_profile;
{
    DEF_ADDRESS;
    COMMAND_HEAD(LTRJ,MCI_StopMotor:);
    WAIT_WRITE_WORD(address,stop_profile);
    AWAIT_NOT_BUSY(address);
    COMMAND(address) = STT;
    COMMAND_TAIL;
}

void MCI_PositionMode(mcikey, control_word,acc, vel, pos)
KEYDEF;
WORD control_word;
double acc, vel;
long pos;
{
    DEF_ADDRESS;
    COMMAND_HEAD(LTRJ,MCI_PositionMode:);
    WAIT_WRITE_WORD(address,control_word);
    if ((control_word & ACC_ABS) != 0) {WAIT_WRITE_LONG(address,(LONGWORD)(acc*0x10000));}
    if ((control_word & VEL_ABS) != 0) {WAIT_WRITE_LONG(address,(LONGWORD)(vel*0x10000));}
    if ((control_word & POS_ABS) != 0) {WAIT_WRITE_LONG(address,pos);}
    COMMAND_TAIL;
}

void MCI_VelocityMode(mcikey, control_word, vel, acc)
KEYDEF;
WORD control_word;
double vel, acc;
{
    DEF_ADDRESS;
    if (vel>=0) {control_word=control_word|0x1000;}
    if (vel<0) {vel=-vel;}
    COMMAND_HEAD(LTRJ,MCI_VelocityMode:);
    WAIT_WRITE_WORD(address,control_word | 0x800);
    if ((control_word & ACC_ABS) != 0) {WAIT_WRITE_LONG(address,(LONGWORD)(acc*0x10000));}
    if ((control_word & VEL_ABS) != 0) {WAIT_WRITE_LONG(address,(LONGWORD)(vel*0x10000));}
    COMMAND_TAIL;
}

void MCI_LoadPosError(mcikey,error,action)
KEYDEF;
WORD error;
int action;
{
    DEF_ADDRESS;
    if (action == STOP) {COMMAND_HEAD(LPES,MCI_PosError:);}
    else {COMMAND_HEAD(LPEI,MCI_PosError:);}
    WAIT_WRITE_WORD(address,error);
    COMMAND_TAIL;
}

void MCI_DefineHome(mcikey)
KEYDEF;
{
    COMMAND_HEAD(DFH,MCI_DefineHome:);
    COMMAND_TAIL;
}

void MCI_SetIndexPos(mcikey)
KEYDEF;
{
    COMMAND_HEAD(SIP,MCI_SetIndexPos:);
    COMMAND_TAIL;
}

```



```

void MCI_SetBreakpoint(mcikey, breakpoint, breakpoint_type)
KEYDEF;
long breakpoint;
int breakpoint_type;
{
    DEF_ADDRESS;
    if (breakpoint_type == ABS)    {COMMAND_HEAD(SBPA,MCI_SetBreakpoint:);}
    else                          {COMMAND_HEAD(SBPR,MCI_SetBreakpoint:);}
    WAIT_WRITE_LONG(address, breakpoint);
    COMMAND_TAIL;
}

int MCI_BreakPointReached(mcikey)
TEST_BODY(0x40)

int MCI_TrajectoryComplete(mcikey)
TEST_BODY(0x04)

int MCI_PositionError(mcikey)
TEST_BODY(0x20)

int MCI_IndexPulse(mcikey)
TEST_BODY(0x08)

void MCI_MaskInt(mcikey, interrupts)
KEYDEF;
WORD interrupts;
{
    DEF_ADDRESS;
    COMMAND_HEAD(MSKI,MCI_MaskInt:);
    WAIT_WRITE_WORD(address, 0xffff - interrupts);
    COMMAND_TAIL;
}

void MCI_ResetInt(mcikey, interrupts)
KEYDEF;
WORD interrupts;
{
    DEF_ADDRESS;
    COMMAND_HEAD(RSTI,MCI_ResetInt:);
    WAIT_WRITE_WORD(address, 0xffff - interrupts);
    COMMAND_TAIL;
}

void MCI_UpdateFilter(mcikey)
KEYDEF;
{
    COMMAND_HEAD(UDF,MCI_UpdateFilter:);
    COMMAND_TAIL;
}

void MCI_StartMotion(mcikey)
KEYDEF;
{
    COMMAND_HEAD(STT,MCI_StartMotion:);
    COMMAND_TAIL;
}

BYTE MCI_ReadStatus(mcikey)
KEYDEF;
{
    return STATUS(mcikey->address);
}

WORD MCI_ReadSignalsReg(mcikey)
KEYDEF;
{
    WORD temp;
    COMMAND_HEAD(RDSIGS,MCI_ReadSignals:);
    WAIT_READ_WORD(mcikey->address, temp);
    COMMAND_TAIL;
    return temp;
}

long MCI_ReadIndexPos(mcikey)
KEYDEF;
{
    long temp;
    COMMAND_HEAD(RDIP,MCI_ReadIndexPos:);

```

```

    WAIT_READ_LONG(mcikey->address,temp);
    COMMAND_TAIL;
    return temp;
}

long MCI_ReadDesiredPos(mcikey)
KEYDEF;
{
    long temp;
    COMMAND_HEAD(RDDP,MCI_ReadDesiredPos:);
    WAIT_READ_LONG(mcikey->address,temp);
    COMMAND_TAIL;
    return temp;
}

long MCI_ReadRealPos(mcikey)
KEYDEF;
{
    long temp;
    COMMAND_HEAD(RDRP,MCI_ReadRealPos:);
    WAIT_READ_LONG(mcikey->address,temp);
    COMMAND_TAIL;
    return temp;
}

double MCI_ReadDesiredVelocity(mcikey)
KEYDEF;
{
    LONGWORD temp;
    COMMAND_HEAD(RDDV,MCI_ReadDesiredVelocity:);
    WAIT_READ_LONG(mcikey->address,temp);
    COMMAND_TAIL;
    return ((double)temp)/0x10000;
}

short MCI_ReadRealVelocity(mcikey)
KEYDEF;
{
    short temp;
    COMMAND_HEAD(RDRV,MCI_ReadRealVelocity:);
    WAIT_READ_WORD(mcikey->address,temp);
    COMMAND_TAIL;
    return temp;
}

WORD MCI_ReadIntegrationSum(mcikey)
KEYDEF;
{
    WORD temp;
    COMMAND_HEAD(RDSUM,MCI_ReadIntegrationSum:);
    WAIT_READ_WORD(mcikey->address,temp);
    COMMAND_TAIL;
    return temp;
}

```

18.9.8 test.c

Simpelt testprogram

```

#include "MCI.h"

void main()
{
    int n;
    MCI_KEYTYPE pit1;
    MCI_KEYTYPE motor[4];

    MCI_Link(0x873e0300,&pit1,&motor[0],&motor[1],&motor[2],&motor[3]);

    MCI_ResetMotor(&motor[0]);
    MCI_LoadFilter(&motor[0], 255,1000,0,0,0);
    MCI_UpdateFilter(&motor[0]);
    MCI_PositionMode(&motor[0],(ACC_ABS|VEL_ABS|POS_ABS),1.0,0.1,-10000);
    MCI_StartMotion(&motor[0]);

    printf("PIT semafornavn: %s SemaforID: %x\n\n",pit1.semname,pit1.semID);
}

```

```
MCI_ReleaseKey(&pit1);
for (n=0; n<4; n++)
{
    printf("Motor: %d Semafornavn: %s SemaforID: %x\n",n+1,motor[n].semname,motor[n].semID);
    MCI_ReleaseKey(&motor[n]);
}
}
```


Kapitel 19

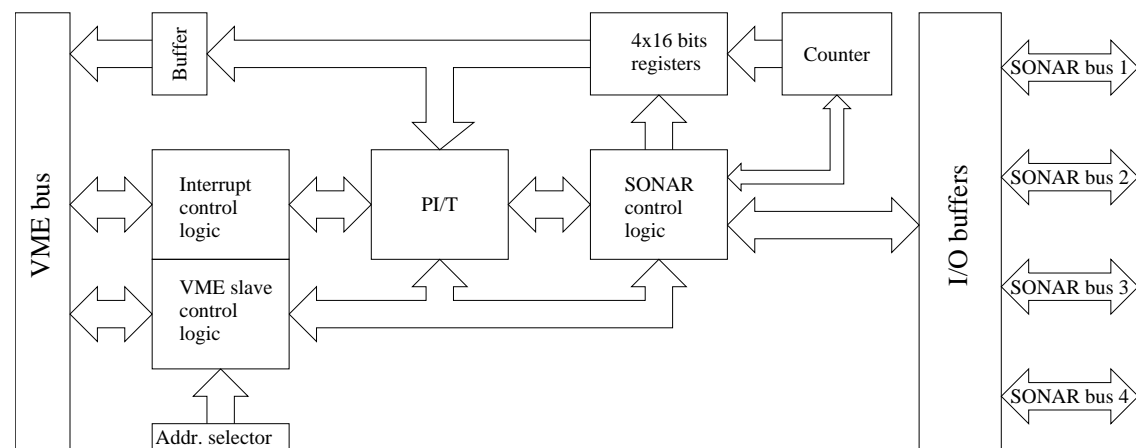
Dokumentation af Heimdal

19.1 Generel beskrivelse af Heimdal

Heimdal kan forbinde op til 16 *ranging modules* til VME bussen. Om nødvendigt kan flere Heimdal moduler forbindes til VME-bussen.

Heimdal foretager op til fire målinger samtidigt, idet sensorerne er opdelt i fire individuelle grupper. Inden for hver gruppe, kan der *eksternt* multiplexes mellem fire sensorer, idet kontrolsignalerne til hver gruppe indeholder to adresselinier.

19.2 Funktionel beskrivelse



Figur 19.1: Blokdiagram over Heimdal

19.2.1 I/O Buffers

Der er *ikke* anvendt galvanisk adskillelse, men alle Heimdals I/O signaler, passerer en *buffer*, der til en vis grad beskytter Heimdals øvgige kredsløb. Alle indgange fra *ranging* modulerne er via *pullup* modstande forbundet til +5V.

19.2.2 Adressevælger

Heimdals *memory map* fylder 256 adresser, og der anvendes 24 bits adressering. De øverste 16 bits af heimdals adresse vælges via. 16 *dip switches*.

19.2.3 Slave kontrollogik

Heimdals er konfigureret som et VME-slave I/O modul. Slave kontrollogikken, overvåger VME bussens adresse og kontrolsignaler, og dekode disse til interne kontrolsignaler, der styrer dataoverførsel, og andre funktioner.

19.2.4 Interrupt kontrollogik

Heimdals kan generere interrupts:

- Når en sensor registrer et ekko.
- Hvis en intern timer løber ud før der modtages et ekko (timeout)
- Via. en uafhængig programmerbar timer.

Alle interrupts optræder på samme *IRQ level*, der kan programmeres fra 1 til 6. Der genereres *vectored* interrupts, på fem forskellige programmerbare vektorer. En vektor for ekko eller *timeout*, for hver sensor, og en vektor for den uafhængige programmerbare timer.

19.2.5 Paralel Interface / Timer (PI/T)

De fleste af Heimdals funktioner styres og overvåges via. digitale ind og udgange på en MC68230 PI/T.

PI/T ligger som en 32 byte blok af lige adresser, i Heimdals *memory map*, og der er direkte adgang til dens registre via. *byte read* og *byte write bus cycles*, til disse adresser.

MC68230's faciliteter som interrupt controller, udnyttes til fulde, således at PI/T kan generere et *vectored interrupt*, for hver sensorgruppe, når en sensor i gruppen registrerer et ekko, eller ved *timeout*.

MC68230's timer del benyttes ikke af Heimdals, men kan bruges som uafhængig timer. Timeren kan programmeres til at generere selvstændige interrupts.

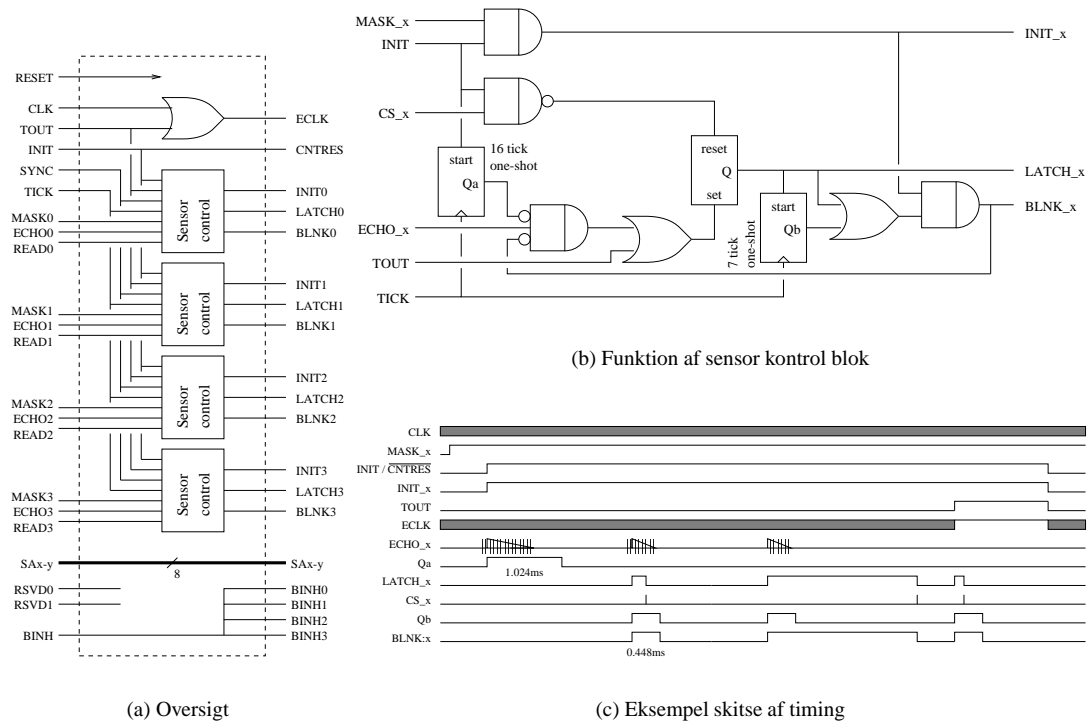
19.2.6 Tæller

Når en afstandsmåling påbegyndes, nulstilles tælleren, der herefter fødes med en fast frekvens, og dermed fungerer som ur. Tælleren standses igen når målingen afbrydes, eller efter ca. 265 ms (timeout).

19.2.7 Ekko registre

Heimdals rummer 4 16 bits ekko registre, et for hver sensorgruppe. Når der registreres et ekko fra en af sensorgrupperne, huskes tællerens tilstand i det register der svarer til den pågældende gruppe. Der registreres ikke flere ekkoer, før registeret aflæses, eller målingen afbrydes.

De fire registre har hver deres adresse i Heimdals *memory map*, og aflæses vha. en *word read cycle*. Efter et ekko, bør registeret kun aflæses en gang, idet aflæsningen tillader at et nyt ekko kan registreres i registeret.



Figur 19.2: Sonar-kontrollogik

19.3 Sonar-kontrol

Sensorerne kontrolleres af Heimdals sonar-kontrollogik, der igen styres af digitale signaler fra VME-bussen, via PI/Ten. I dette afsnit gennemgås funktionen af sonar-kontrollogikken.

Figur 19.2 viser et **simplificeret** diagram over sonar-kontrollogikken, og tabel 19.1 giver en oversigt over signalerne til og fra den.

19.3.1 Funktionel beskrivelse

Sonar-kontrollogikken består af et relativt komplekst logikkredsløb, der indeholder kombinatorisk logik, såvel som adskillige tilstandsmaskiner. Det meste af sonar-kontrollogikken er implementeret som programmerbar logik.

Så længe INIT ikke er aktiveret (mellem målinger) holdes tælleren nulstillet. Så snart INIT aktiveres, begynder tælleren at tælle ECLK pulser (8MHz). Efter 2^{21} pulser ($262144 \mu s$) Aktiverer tælleren TOUT signalet, hvilket afbryder ECLK signalet, så tælleren standser. Uanset hvornår INIT deaktiveres, sættes tælleren igen i nulstillet tilstand, og processen kan startes forfra.

Når INIT aktiveres, startes samtidigt en *one-shot*, der forhindrer kredsløbet i at registrere ekko signaler, i en periode svarende til 16 TICK pulser (1.024 ms). Som det fremgår af figur 19.2, registrerer kredsløbet ikke ekko signaler, i denne periode, eller når BLNK-X udgangen for den pågældende sensorgruppe er aktiv. Begge disse foranstaltninger er foretaget, for at imødekomme problemer med POLAROID's *ranging modules* interne blanking kredsløb, der ikke fungerer efter hensigten.

Når et (ægte) ekko registreres, eller når timeren aktiverer TOUT signalet, registreres og huskes dette, af en *set-reset flip flop*. Flip floppens udgang, aktiveres, hvilket får det pågældende sonar-register, til at huske

Signal(er)	antal	til/fra	Funktion	Aktiv
RESET	1	VME-bus	Nulstiller kontrollogik ved SYSRESET	lav
CLK	1	SLAVE kontrollogik	8MHz clock signal	N/A
RESCNT	1	Tæller	Nulstiller tælleren og holder den nulstillet	lav
ECLK	1	Tæller	8MHz clock signal til tælleren der måler tiden	N/A
SYNC	1	Tæller	ECLK delt med 64 (125kHz), til synkronisering og interne timers	N/A
TICK	1	Tæller	ECLK delt med 512 (15625Hz), til interne timers	N/A
TOUT	1	Tæller	<i>timeout</i> signal aktiveres efter 262144 μ s (ECLK delt med 4194304)	høj
INIT	1	PI/T	Indleder en afstandsmåling	høj
MASK 0-3	3	PI/T	Tillader at INIT sendes videre til sensorgrupperne	høj
SA0, 0-1	2	PI/T	Adressen på aktiv sensor i gruppe 0	høj
SA1, 0-1	2	PI/T	Adressen på aktiv sensor i gruppe 1	høj
SA2, 0-1	2	PI/T	Adressen på aktiv sensor i gruppe 2	høj
SA3, 0-1	2	PI/T	Adressen på aktiv sensor i gruppe 3	høj
INIT 0-3	4	Sensorgrupper	INIT signaler til sensorgrupperne	høj
ECHO 0-3	4	Sensorgrupper	ECHO signaler fra sensorgrupperne	høj
LATCH 0-3	4	Ekko registre	<i>latcher</i> tællerens tilstand ind i ekko register	høj
BLNK 0-3	4	Sensorgrupper og PI/T	BLNK signal til sensorgrupperne	høj
BINH	1	PI/T	Signal der afbryder sensorenes interne <i>blanking</i> interval	høj
BINH 0-3	4	Sensorgrupper & PI/T	BINH signal til de enkelte sensorgrupper	høj
RSVD 0-1	2	PI/T	Reserveret til fremtidig brug	N/A
CS 0-3	4	SLAVE kontrollogik	Signalerer at et ekko register aflæses	lav

Tabel 19.1: Oversigt over signaler til/fra sonar-kontrollogikken

tilstanden af tællerens bit 6-21, hvilket giver sonar-registret en opløsning på 8 μ s, svarende til en afstand-sopløsning på:

$$\frac{v_{lyd} \times 8\mu s / count}{2} \simeq 1.376 \times 10^{-3} m / count, v_{lyd} \simeq 344 m/s \quad (19.1)$$

Sonar-registret indlæser tællerens tilstand ved en opadgående flanke, så det har ingen betydning for aflæsningen at LATCH-X signalet forbliver aktivt. *flip flop*en nulstilles når INIT deaktiveres, eller hvis sonar registret aflæses (CS-X).

Når LATCH signalet aktiveres, startes en *one shot*, hvis udgang forbliver aktiv under 7 TICK pulser (448 μ s). Forudsat at INIT-X er aktiveret (svarende til at sensorgruppen anvendes under målingen), er BLNK-X signalet aktivt, så længe *one shot*en er kører. Når *one shot*en udløber, er BLNK-X i samme tilstand som LATCH-X. Udover at nulstille *ranging* modulet, efter et ekko, går BLNK signalet også til en *handshake* indgang på PI/Ten, der kan aflæses både som almindeligt register (polling), og generere et interrupt.

19.4 Hardware setup

Den eneste hardware setup der er nødvendig er indstilling af Heimdals basis adresse.

19.4.1 Basisadresse

3U VME bus anvender 23 adresselinier ($a_1 \dots a_{23}$) til adressering af 8192K words, eller 16M bytes hukommelse, i adresserummet $000000_{hex} \dots FFFFFFF_{hex}$. Heimdals adresserum består af 128 words, hvis basisadresse bestemmes af 2×8 dipswitches (S1 — placeret yderst mod modulets kant, og S2 — placeret længere inde på modulet). S2 vælger de 8 mest betydende bits af basisadressen ($bit_{16} \dots bit_{23}$), og S1 de 8 mindst betydende bits ($bit_8 \dots bit_{15}$).

Nummereringen af kontakterne på S1 og S2, svarer til adresse bittene, således at S2's kontakt 8, svarer til bit_{23} , og S1's kontakt 1 svarer til bit_8 .

En kontakt i ON positionen svarer til en sat bit.

ben	signal	ben	signal	ben	signal	ben	signal	funktion
1	SA0-0	11	SA0-1	21	SA0-2	31	SA0-3	Adresse ₀
2	SA1-0	12	SA1-1	22	SA1-2	32	SA1-3	Adresse ₁
3	GND	13	GND	23	GND	33	GND	Reference
4	INIT-0	14	INIT-1	24	INIT-2	34	INIT-3	Init
5	GND	15	GND	25	GND	35	GND	Reference
6	BINH-0	16	BINH-1	26	BINH-2	36	BINH-3	Blanking inhibit
7	BLNK-0	17	BLNK-1	27	BLNK-2	37	BLNK-3	Blank
8	GND	18	GND	28	GND	38	GND	Reference
9	ECHO-0	19	ECHO-1	29	ECHO-2	39	ECHO-3	Echo
10	GND	20	GND	30	GND	40	GND	Reference

Tabel 19.2: Benforbindelser til CON 2

Sættes de 16 kontakter til at repræsentere tallet $yyyy_{hex}$, ligger Heimdal i adresserummet: $yyyy00_{hex} \dots yyyyFF_{hex}$

Bemærk at VME CPU moduler ofte *mapper* VME-bussens adresseområde til et lokalt område i et 32-bits adresserum. Det VMPM-68KC2 CPU modul der anvendes i Cato, *mapper* VME adresserummet til området: $87xxxxxx_{hex}$, således at der skal adderes 87000000_{hex} til alle VME adresser.

19.4.2 Yderligere konfiguration

Al yderligere konfiguration af Heimdal sker via. software, som gennemgås i kapitel 19.7

19.5 Tilslutning

Heimdal tilsluttes J1¹ connectoren i VME-bussen, via. det 96 Polede Europa-stik (CON 1). Heimdal er et standard europa størrelse (100 × 160mm) modul, så det passer i skinnerne i VME racket. For benforbindelserne på CON 1 stikket henvises til diagrammet i appendix 19.13, eller til [24].

De fire sonar-grupper, tilsluttes via et enkelt 40 polet fladkabelstik (CON 2) i den modsatte ende af modulet. Tabel 19.2 viser benforbindelserne til stikket. Som det ses, er der afsat 10 ben til hver sonar-gruppe, så fladkablet kan splittes op i fire uafhængige kabler, der hver især kan anvendes som *bus*, for op til fire *ranging modules*.

19.6 Adressering

Tabel 19.3 Viser blokopdelingen af Heimdals adresserum. Bemærk at adressebit 7 ikke anvendes, og at der derfor opstår to identiske *spejlinger* (00 — 7F og 80 — FF).

19.6.1 PI/T

Tabel 19.4 viser Hvilke Heimdal-adresser, der svarer til registre i PI/Ten. PI/Ten er en 8-bits enhed, forbundet til VME-bussens 8 lave datalinier (D0 — D7), hvorfor PI/Ten forekommer på de ulige byte adresser.

PI/Ten er en relativt avanceret periferikreds, og jeg henviser til [19], for yderligere information.

¹PEP computerne har kun J1 connectorer.

Adresse (Hex)	lige	ulige	Funktion
00 — 3F	IRQ level control	PI/T	Read/write
40 — 4F	Echo register 0, Hi byte	Echo register 0, low byte	Read only
50 — 5F	Echo register 1, Hi byte	Echo register 1, low byte	Read only
60 — 6F	Echo register 2, Hi byte	Echo register 2, low byte	Read only
70 — 7F	Echo register 3, Hi byte	Echo register 3, low byte	Read only
80 — AF	IRQ level control	PI/T	Read/write
C0 — CF	Echo register 0, Hi byte	Echo register 0, low byte	Read only
D0 — DF	Echo register 1, Hi byte	Echo register 1, low byte	Read only
E0 — EF	Echo register 2, Hi byte	Echo register 2, low byte	Read only
F0 — FF	Echo register 3, Hi byte	Echo register 3, low byte	Read only

Tabel 19.3: Heimdals *memory map*

Adr.	Register	Forkortelse	Adr.	Register	Forkortelse
01	Port General Control Register	PGCR	21	Timer Control Register	TCR
03	Port Service Request Register	PSRR	23	Timer Interrupt Vector Register	TIVR
05	Port A Data Direction Register	PADDR	25	— Intet register —	
07	Port B Data Direction Register	PBDDR	27	Counter Preload Register, High	CPRH
09	Port C Data Direction Register	PCDDR	29	Counter Preload Register, Mid	CPRM
0B	Port Interrupt Vector register	PIVR	2B	Counter Preload Register, Low	CPRL
0D	Port A Control Register	PACR	2D	— Intet register —	
0F	Port B Control Register	PBCR	2F	Count Register, High	CRH
11	Port A Data Register	PADR	31	Count Register, Mid	CRM
13	Port B Data Register	PBDR	33	Count Register, Low	CRL
15	Port A Alternate Register	PAAR	35	Timer Status Register	TSR
17	Port B Alternate Register	PBAR	37	— Intet register —	
19	Port C Data Register	PCDR	39	— Intet register —	
1B	Port Status Register	PSR	3B	— Intet register —	
1D	— Intet register —		3D	— Intet register —	
1F	— Intet register —		3F	— Intet register —	

Tabel 19.4: Memory map for PI/T

19.6.2 IRQ level kontrol

Dette område af adresserummet (00, 02, 04 . . . 3C, 3E), svarer ikke til registre i nogen enheder. Alligevel kan dette område adresseres med *byte read* og *byte write* kommandoer.

Skrivning og læsning i området, bruges til at vælge IRQ-level for *interrupt* kontrollogikken. En *byte write cycle*, bevirker at *IRQ-level* sættes til 0 (LEVEL-RESET), svarende til at *interrupts* er slået fra. En *byte read cycle* (LEVEL-INCREMENT), bevirker at *IRQ-level* tælles en op. *IRQ-level* tælleren er en tre-bits binær tæller, der tæller fra 0 til 7, og derefter kammer over til 0 igen. Kun værdierne 1 til og med 6 giver anledning til generering af interrupts, på den tilsvarende *IRQ level*. Værdierne 0 og 7 svarer til at *interrupts* er slået fra.

Det er uden betydning hvilken byte værdi der skrives, ved LEVEL-RESET, ligesom det ikke er defineret hvilken byte værdi der aflæses ved LEVEL-INCREMENT.

19.6.3 Ekko registre

Adresse signalerne A1, A2, og A3 bruges ikke ved adresseringen af ekko registre, hvorfor hvert register er gentaget i 8 efterfølgende word-adresser. Hvilken af de 8 *spejlinger* der anvendes er uden betydning.

Ekko registrene er 16 bits *read only* registre, der aflæses vha. *word read cycles*.

19.6.4 Adressering fra MC68000 Assembler program

Der er mange måder at foretage adressering af Heimdals registre på, via 68000 maskinkodeinstruktioner. Generelt skal PI/Ten og *IRQ level* kontrol området adresseres vha. *move.b*, eller tilsvarende byte orienterede instruktioner, mens ekko registre skal adresseres vha. *move.w* eller tilsvarende word orienterede kommandoer.

Figur 19.3 viser et simpelt eksempel på adressering af Heimdals forskellige registre fra maskinkode.

Alle MC68000's adresseringsmetoder kan naturligvis anvendes. Sålænge Heimdals registre adresseres korrekt, mht. lige / ulige adresser, og byte/word adgang.

19.6.5 Adressering fra C program

Heimdals registre kan adresseres direkte fra C, uden brug af inline maskinkode:

```
/* C arbejder ikke direkte med typerne bytes og words, så følgende definition
af disse typer, er afhængige af hvilken C compiler der anvendes. For
Microwares OS-9/68k C ver. 3.0, der anvendes i forbindelse med OS-9 på Cato's
PEP VME computer gælder følgende definitioner: */
#define BYTE unsigned char
#define WORD unsigned short

/* Følgende linie skriver byte-værdien VALUE i registeret på adresse ADR */
*(BYTE *)ADR = VALUE;

/* Følgende læser indholdet af byte registeret på adresse ADR, ind i variabelen byte */
byte = *(BYTE *)ADR;

/* Følgende læser indholdet af word registeret på adresse ADR, ind i variabelen word */
word = *(WORD *)ADR;
```

Figur 19.4 viser et eksempel på C kode, med samme funktion som assemblerkoden i figur 19.3.

Anvendes metoden sammen med andre compilere, er det vigtigt at sikre sig at adresseringen stadig foretages korrekt. F.eks. ved at studere den maskinkode compileren genererer.

19.7 Programmering

I dette kapitel beskrives hvordan Heimdals programmeres. Eksemplerne gives ikke i noget bestemt programmeringssprog, men i *pseudo-kode*.

19.7.1 Adgang til registre

Adresseringen af Heimdals registre i MC68000 assembler og C, er beskrevet i afsnit 19.6. I dette afsnit beskrives adresseringen af Heimdals registre, med pseudo kommandoerne:

Byte-Read(*adr*) En funktion der foretager en *byte read cycle*, og antager værdien af byte-registeret på adressen: *adr*. *Byte-Read(ad)* kan også anvendes som procedure, idet dens returværdi så ignoreres.

Byte-Write(*adr,val*) En procedure der foretager en *byte write cycle*, for at skrive værdien: *val*, i byte-registeret, på adressen: *adr*.

Word-Read(*adr*) Som *Byte-Read*, blot med en *word read cycle* i stedet.

```

BASIS          equ      $873E0000      ; Heimdals basisadresse er sat til
                                           ; 3E0000 via dipswitches S1 og S2
                                           ; CPU modulet mapper VME I/O adresser
                                           ; til adresseområdet 87xxxxxx

LEVEL_CONTROL  equ      BASIS + 0      ; Adresse i IRQ level control området

PGCR           equ      BASIS + 1      ; Port General Control Register i PI/T
PSRR           equ      BASIS + 3      ; Port Service Request Register i PI/T
PADDR         equ      BASIS + 5      ; Port A Data Direction Register
; etc, etc.

sonar0         equ      BASIS + $40    ; Adresse på sonar register 0
sonar1         equ      BASIS + $50    ; Adresse på sonar register 1
sonar2         equ      BASIS + $60    ; ...
sonar3         equ      BASIS + $70    ;

;
; Udsnit af instruktioner
;

        clr.b LEVEL_CONTROL          ; skriv til LEVEL CONTROL, for at
                                           ; sætte IRQ level til 0.
        tst.b LEVEL_CONTROL          ; læs fra LEVEL CONTROL, for at tælle
                                           ; IRQ level op til 1.
        tst.b LEVEL_CONTROL          ; Igen, for at få IRQ level op på 2.

        move.b #$3F,PGCR             ; skriv 3F_hex i PGCR i PI/T
        move.b #$20,PSRR             ; skriv 20_hex i PSRR i PI/T
;
; etc, etc.
;

        move.w sonar0,d0              ; aflæs sonar 0 registeret
        move.w sonar1,d1              ; aflæs sonar 1 registeret

```

Figur 19.3: Eksempel på adressering fra maskinkode

```

#define BYTE unsigned char
#define WORD unsigned short

#define BASIS 0x873e0000

#define LEVEL_CONTROL *(BYTE *)(BASIS + 0)
#define PGCR          *(BYTE *)(BASIS + 1)
#define PSRR          *(BYTE *)(BASIS + 3)
#define PADDR         *(BYTE *)(BASIS + 5)

#define sonar0         *(WORD *)(BASIS + 0x40)
#define sonar1         *(WORD *)(BASIS + 0x50)
#define sonar2         *(WORD *)(BASIS + 0x60)
#define sonar3         *(WORD *)(BASIS + 0x70)

/* Udsnit af program */

BYTE dummy;          /* Variabel definitioner */
WORD range[4];

LEVEL_CONTROL = 0;    /* Skriv til LEVEL_CONTROL, for at sætte IRQ levet til 0 */
dummy = LEVEL_CONTROL; /* Læs fra LEVEL_CONTROL, for at tælle IRQ level 1 op */
dummy = LEVEL_CONTROL; /* Igen, for at komme op på 2 */

PGCR = 0x3F;          /* Skriv 3F_hex til PGCR */
PSRR = 0x20;          /* Skriv 20_hex til PSRR */
/* etc etc */

range[0] = sonar0;    /* Aflæs sonar register 0 */
range[1] = sonar1;    /* Aflæs sonar register 1 */

```

Figur 19.4: Eksempler på adressering i C

Word-Write(adr,val) Som Byte-Write, blot med en *word write cycle* i stedet.

Adresserne der anvendes i dette kapitel, er adresserne på Heimdals registre, uden *offset* pga. basisadresse, eller *mapping*. Alle adresser gives som hexadecimale tal, med registerets navn i en kommentar bagefter.

19.7.2 Styresignaler

Alle styresignaler til sonar-logikken, på nær fire, genereres af PI/Ten, der også overvåger de fire BLNK signaler, der indikerer at et ekko er registreret. De fire styresignaler CS-0 ... CS-3, genereres af SLAVE kontrollogikken, når det respektive sonar-register aflæses.

Tabel 19.5, giver en oversigt over signalerne.

Gruppe	idx	signal	Gruppe	idx	signal	Gruppe	idx	signal
PI/T Port A	0	MASK-0	PI/T Port B	0	SA0-0	PI/T Port C	0	INIT
	1	MASK-1		1	SA1-0		1	BINH
	2	MASK-2		2	SA0-1		2	NC
	3	MASK-3		3	SA1-1		3	NC
	4	Reserved 0		4	SA0-2		4	NC
	5	Reserved 1		5	SA1-2		5	NC
	6	NC		6	SA0-3		6	NC
	7	NC		7	SA1-3		7	NC
PI/T Handshake	H1	BLNK-0				Læsning fra sonar register	0	CS-0
	H2	BLNK-1					1	CS-1
	H3	BLNK-2					2	CS-2
	H4	BLNK-3					3	CS-3

Tabel 19.5: Oversigt over kontrolsignaler til/fra sonar logik

19.8 Initialisering

I dette afsnit beskrives hvordan Heimdal initialiseres

19.8.1 Interruptniveau

Heimdal har programmerbar *interrupt level* fra 1 til 6. *Level 7 (non maskable interrupts)* er ikke implementeret, da det i forbindelse med operativsystemer ikke giver mening at anvende interrupts processoren ikke kan slå fra.

Interruptniveauet styres vha. en tæller, implementeret i Heimdals SLAVE kontrollogik. Tælleren er en 3 bits binær tæller, der kan nulstilles og tælles et skridt op. Tælleren gennemløber sekvensen 0,1,2,3,4,5,6,7,0,1,..., hvor tilstandene 1...6 koresponderer med interruptniveauet 1...6. Tilstanden 0 og 7 svarer til at interrupts er slået fra.

Tælleren nulstilles ikke når Heimdal nulstilles vha. SYSRESET fra VME-bussen, men skal nulstilles via software, for at komme i en kendt tilstand.

Nedenfor vises hvordan IRQ niveauet kan initialiseres.

```
PROCEDURE Init-IRQlevel(int level)
  Byte-Write(0x00,0x00)      /* nulstil tæller ved at skrive en tilf. byte i IRQlevel control */
  WHILE (level > 0) DO        /* Løkken kører level gange! */
    Byte-Read(0x00)           /* tæl IRQ level tæller en op! */
    level=level-1
  END WHILE
END PROCEDURE
```

19.8.2 PI/T (uden interrupts)

MC68230 er dokumenteret i [19], og jeg henviser hertil for en tilbundsgående beskrivelse af den.

Hvordan den initialiseres i Heimdal afhænger af om der skal anvendes interrupts eller ej. I dette afsnit gives et eksempel på initialisering til brug uden interrupts. I dette tilfælde er det iøvrigt unødvendigt at initialisere IRQ niveauet, som omtalt tidligere.

```
PROCEDURE Init-PIT-basic()
  Byte-Write(0x01,0x3F)      /* PGCR = 00111111_b */
  Byte-Write(0x03,0x18)      /* PSRR = 00011000_b */
  Byte-Write(0x0D,0x80)      /* PACR = 10000000_b */
  Byte-Write(0x0F,0x80)      /* PBCR = 10000000_b */
  Byte-Write(0x11,0x00)      /* PADR = 00000000_b */
  Byte-Write(0x13,0x00)      /* PBDR = 00000000_b */
  Byte-Write(0x19,0x00)      /* PCDR = 00000000_b */
  Byte-Write(0x05,0xFF)      /* PADDR = 11111111_b */
  Byte-Write(0x07,0xFF)      /* PBDDR = 11111111_b */
  Byte-Write(0x09,0x03)      /* PCDDR = 00000011_b */
END PROCEDURE
```

PI/Ten anvendes i *mode 0 (unidirectional 8-Bit mode)*, hvorfor bit 7 og 6 i PGCR sættes til 0. Alle fire *handshake* slås til ved at sætte bit 4 og 5 lig 1. PI/Tens fire indbyggede *set reset flip flops* sættes alle til at *trigge* på opadgående flanker på de respektive *handshake* indgange, ved at sætte bit 0-3 lig 1.

PSRR har intet bit-7. Der anvendes ikke DMA, hvorfor bit 5 og 6 sættes til 0. Selv om der ikke anvendes interrupts, er PI/Ten hardware-mæssigt koblet til interrupt logik, hvorfor bit 3 og 4 sættes til 1. Da der ikke anvendes interrupts, er tilstanden af bit 0 til 2 uden betydning.

Både port A og B anvendes i *submode 1X*, hvorfor PACR og PBCR's bit 7 sættes til 1. Bit 6 er uden betydning når bit 7 er 1. H2/H4 anvendes som input ben, til statusaflysning, hvorfor bit 5 sættes til 0. Når bit 5 er 0, er 3 og 4 uden betydning. Der anvendes ikke interrupts, hvorfor bit 1 og 2 sættes til 0. I submode 1X er bit 0 uden betydning.

Både port A, B, og C nulstilles, for at sætte alle udgange lave fra starten.

Hele port B, bit 0-3 i port A, og bit 0 og 1 i port C, anvendes som digitale udgange, hvorfor de tilsvarende bits i PADDR, PBDDR, og PCDDR, sættes til 1. Bit 4 og 5 i port A, er forbundet til sonar-kontrollogikken, for at have mulighed for at udvide dennes funktionalitet. Bit 6 og 7 er ikke forbundet til noget.

19.8.3 PI/T (med interrupts)

Skal der anvendes interrupts, ser initialiseringen således ud:

```
PROCEDURE Init-PIT(BYTE IRQ-vector)
  Byte-Write(0x01,0x3F)      /* PGCR  = 00111111_b */
  Byte-Write(0x03,0x18)      /* PSRR = 00011000_b */
  Byte-Write(0x0B,IRQ-vector) /* PIVR  = IRQ-vecor  */
  Byte-Write(0x0D,0x86)      /* PACR  = 10000110_b */
  Byte-Write(0x0F,0x86)      /* PBCR  = 10000110_b */
  Byte-Write(0x11,0x00)      /* PADR  = 00000000_b */
  Byte-Write(0x13,0x00)      /* PBDR  = 00000000_b */
  Byte-Write(0x19,0x00)      /* PCDR  = 00000000_b */
  Byte-Write(0x05,0xFF)      /* PADDR = 11111111_b */
  Byte-Write(0x07,0xFF)      /* PBDDR = 11111111_b */
  Byte-Write(0x09,0x03)      /* PCDDR = 00000011_b */
END PROCEDURE
```

Da der anvendes interrupts, skal PI/Ten programmeres med en *IRQ vecor*, der skrives til PIVR registeret.

I både PACR og PBCR, sættes bit 1 og 2, for at få PI/Ten til at generere interrupts, ved aktivering af alle fire *handshake* indgange.

Resten af initialiseringen svarer til tilfældet uden brug af interrupts.

19.9 Grundlæggende programmering

I dette afsnit gennemgås grundbegreberne i at programere Heimdal. Der tages udgangspunkt i den simplest mulige anvendelse af Heimdal, uden interrupts og uden polling. Programmeringen gennemgås i pseudokode, og der gives et fungerende programeksempel til Microware C, på PEP VMPM 68KC2 computeren.

```
/* *****
 * Forudsætningen for at nedenstående procedure kan anvendes, er at
 * funktionen: Init_PIT_basic() har været kaldt.
 *
 * Get Distances tager fire parametre (S0-S2), en for hver sensorgruppe.
 * Hvis en sensorgruppe ikke skal anvendes under målingen, sættes den
 * tilsvarende parameter lig -1.
 *
 * Skal gruppen anvendes, indeholder parameteren nummeret på den sensor i
 * gruppen, der skal bruges (0-3).
 *
 * Proceduren returnerer de målte afstande i meter, i parametrene D0 - D3.
 * Ugyldige målinger (TIMEOUT, eller hvis sensoren ikke anvendes) returneres
 * som negative tal (-1.0)
 * ***** */

PROCEDURE Get_Distances(INTEGER S0,S1,S2,S3;REF FLOAT D0,D1,D2,D3)
  BYTE mask, addresses      /* variabel deklARATIONER */
  WORD dist[4]

/* Initialiser Dx variablene */
  D0=0.0; D1=0.0; D2=0.0; D3=0.0

/* Vælg hvilke sensorgrupper der deltager i målingen! */
  mask = 0x0F
  IF S0 < 0 THEN mask = mask-1; S0=0; D0=-1.0 /* Alle fire sensorgrupper! */
  /* slet bit 0 hvis S0 ikke bruges */
```

```

IF S1 < 0 THEN mask = mask-2; S1=0; D1=-1.0 /* slet bit 1 hvis S1 ikke bruges */
IF S2 < 0 THEN mask = mask-4; S2=0; D2=-1.0 /* slet bit 2 hvis S2 ikke bruges */
IF S3 < 0 THEN mask = mask-8; S3=0; D3=-1.0 /* slet bit 3 hvis S3 ikke bruges */
Byte_Write(0x11,mask) /* skriv mask til PADR i PI/T */

/* Set alle grupper to adressesignaler som S0-S3 indikerer */
addresses = S0 + 4*S1 + 16*S2 + 64*S3 /* adresserne "stables" */
Byte_Write(0x13,addresses) /* Skriv adresserne til PBDR i PI/T */

/* Start målingen */
Byte_Write(0x19,0x01) /* PCDR, bit 0 (INIT) aktiveres */
WAIT(0.27) /* vent i (mindst) 0.27 sekund */

/* Aflæs registre een gang! */
dist[0] = Word_Read(0x40) /* Aflæs sonar register 0 */
dist[1] = Word_Read(0x50)
dist[2] = Word_Read(0x60)
dist[3] = Word_Read(0x70)

/* Afbyd målingen */
Byte_Write(0x19,0x00) /* Sæt INIT lav igen */

/* Check for "timeout" */
IF dist[0] >= 0x8000 THEN D0=-1.0 /* Timeout indikeres af bit 15 */
IF dist[1] >= 0x8000 THEN D1=-1.0 /* i sonar registeret */
IF dist[2] >= 0x8000 THEN D2=-1.0
IF dist[3] >= 0x8000 THEN D3=-1.0

/* Beregn afstand i meter (konstanten K = 0.001376 meter/count) */
IF D0 > -1.0 THEN D0 = CAST(dist[0],FLOAT)*K
IF D1 > -1.0 THEN D1 = CAST(dist[1],FLOAT)*K
IF D2 > -1.0 THEN D2 = CAST(dist[2],FLOAT)*K
IF D3 > -1.0 THEN D3 = CAST(dist[3],FLOAT)*K
END PROCEDURE

```

Proceduren bruger Dx variablene internt, så de initialiseres til nul, i starten af proceduren.

Hver enkelt Sx parameter chekes for at undersøge hvilke sensorgrupper der skal anvendes. Hvis en sensorgruppe ikke anvendes, fjernes det tilsvarende bit, fra en bitmaske. For at få Sx til at give en gyldig to bits adresse, sættes Sx i dette tilfælde til 0. Bruges en sensorgruppe ikke, sættes dens returparameter til -1.0, for at afspejle en ugyldig værdi. Når alle fire parametre er checket, skrives bitmasken til PI/Tens Port A, og dermed til sonar kontrollogikken.

De fire Sx parametre, sættes sammen til en Byte, der skrives til PI/Tens Port B, og dermed til de fire sensorgrupper.

Målingen startes ved at sætte INIT høj. BINH bruges ikke i dette eksempel, og holdes lav gennem hele målingen.

Der ventes i 0.27 sekund. Hvis en sensorgruppe ikke registrerer et ekko i dette tidsrum, vil dets sonar-registre registrere *timeout* efter ca. 0.262 sekund.

Alle fire registre aflæses. De kopieres over i fire variable, så de kun skal aflæses en gang.

Målingen afbrydes, ved at sætte INIT lav igen (Port C bit 0).

Det undersøges om der er *timeout* på nogen af grupperne, og målingen registreres som ugyldig, hvis det er tilfældet.

Hvis en måling er gyldig konverteres aflæsningen af sonar-registre til meter. sonar-registre måler tiden med en opløsning på 8 mikrosekunder. Lydens hastighed er 344 m/s, og lyden skal bevæge sig både frem og tilbage, hvilket giver en lydhastighed på 0.001376 m/count.

19.9.1 Brug af BINH

Hvis det ønskes at sætte sensorerne i *lytte tilstanden* før deres indbyggede 2.38 ms *timer* gør det, sættes BINH (Port C bit 1) højt, efter den ønskede pause. Det sted i programmet hvor målingen startes kommer så til at se således ud:

```
/* Start målingen */
Byte_Write(0x19,0x01)          /* PCDR, bit 0 (INIT) aktiveres          */
WAIT(BINH_pause)                /* Vent indtil BINH skal aktiveres      */
Byte_Write(0x19,0x03)          /* Både INIT og BINH høje              */
WAIT(0.27-BINH_pause)          /* vent i resten af 0.27 sekund intervallet */
```

Målingen afsluttes stadig ved at skrive 0 til Port C, for at sætte INIT og BINH lave igen.

19.9.2 Hurtigere måling

Sensorerne har en rækkevidde på ca. 11 meter, svarende til ca. 64 ms. Det er derfor muligt at foretage helt op til ca. 15 målinger i sekundet, uden at få problemer med akustisk interferens mellem de enkelte målinger².

En hurtigere målefrekvens kan opnås, ved at sætte pausen ned fra 0.27 sekund, til f.eks. de nævnte 64 ms. Hvis der imidlertid ikke registreres et ekko i dette tidsinterval, er indholdet i sonar-registret er ikke defineret.

For at imødekomme dette problem, kan PI/Tens *handshake* indgange (H1 - H4), aflæses for at se om et ekko er registreret. Hx indgangen niveau kan aflæses direkte i PI/Tens PSR register, som bits 4 til 7. Bit 4 svarer til BLNK-0, bit 5 til BLNK-1 osv. PSR registerets bit 0-3 afspejler ikke direkte niveauet af H1-H4, og dermed BLNK-0 til BLNK-3. De fire nederste bits i PSR, svarer til udgangene fra fire individuelle *set reset flip-flops*. Hver *flip-flop* nulstilles, ved at skrive en *maske* i PSR registeret, hvor den korresponderende bit er sat. *Flip-floppen* sættes, ved at den koresponderende *handshake* indgang aktiveres. I initialiseringen blev det defineret at alle fire *flip-flops* bliver sat, ved opadgående flanker på de respektive Hx indgange.

Hvis der kun skal måles et enkelt ekko, for hver sensorgruppe, er det et fedt om niveauerne undersøges direkte (bit 4-7) eller om der anvendes *flip-flops* (bit 0-3), idet ingen af BLNK signalerne nulstilles igen, før målingen afsluttes. *flip-flop* faciliteten er imidlertid nyttig ved detektering af flere ekkoer, og jeg vælger at introducere den allerede her.

Måledelen af programmet kommer til at se således ud:

```
/* Start målingen */
Byte_Write(0x1B,0x0F)          /* Skriv 00001111 til PSR, for at nulstil alle */
                                /* fire Handshake flip flops              */
Byte_Write(0x19,0x01)          /* PCDR, bit 0 (INIT) aktiveres          */
WAIT(BINH_pause)                /* Vent indtil BINH skal aktiveres      */
Byte_Write(0x19,0x03)          /* Både INIT og BINH høje              */
WAIT(0.064-BINH_pause)         /* vent i resten af 0.064 sekund intervallet */

/* Check for "timeout" før registre aflæses!!!! */
/*      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ */
status=Byte_Read(0x1B)          /* Aflæs PSR i PI/T                      */
IF (status BIT_AND 0x01) = 0 THEN D0=-1.0 /* Hvis et ekko er reistreret          */
IF (status BIT_AND 0x02) = 0 THEN D1=-1.0 /* for en sensorgruppe, er den         */
IF (status BIT_AND 0x04) = 0 THEN D2=-1.0 /* korresponderende bit sat             */
IF (status BIT_AND 0x08) = 0 THEN D3=-1.0 /* Ellers ikke                          */

/* Aflæs registre een gang! */
dist[0] = Word_Read(0x40)       /* Aflæs sonar register 0                */
dist[1] = Word_Read(0x50)       /* Aflæs sonar register 1                */
dist[2] = Word_Read(0x60)       /* Aflæs sonar register 2                */
dist[3] = Word_Read(0x70)       /* Aflæs sonar register 3                */
```

²Interferens mellem efterfølgende målinger, kan opstå hvis der måles for ofte, idet et ekko fra den forrige måling kommer tilbage til sensoren, mens den lytter efter ekkoet fra en ny måling

```

/* Afbyd målingen */
Byte_Write(0x19,0x00)                /* Sæt INIT og BINH lave igen */

/* Beregn afstand i meter (konstanten V = 0.001376 meter/count) */
IF D0 > -1.0 THEN D0 = CAST(dist[0],FLOAT)*V
IF D1 > -1.0 THEN D1 = CAST(dist[1],FLOAT)*V
IF D2 > -1.0 THEN D2 = CAST(dist[2],FLOAT)*V
IF D3 > -1.0 THEN D3 = CAST(dist[3],FLOAT)*V
END PROCEDURE

```

Det er vigtigt at aflæse *Handshake* indgangene, før EKKO registrene, idet det ellers risikeres at aflæse EKKO registeret **før** et ekko modtages, og BLNK, **efter** et ekko modtages.

19.9.3 Multi ekko programmering

Heimdal er designet til at registrere mere end det første ekko. For at udnytte denne facilitet, er det imidlertid nødvendigt for at ens program registrerer og aflæse et ekko umiddelbart når det registreres. Derfor er det nødvendigt at anvende interrupts, eller at *polle*, PI/Tens *handshake* indgange ret ofte.

Uanset hvilken af de to metoder der anvendes, ser strukturen af programmet ud som følger:

1. Initialiser Heimdal
2. Nulstil alle fire *handshake flip-flops*.
3. Sæt INIT høj (begynd måling)
4. Hvis BINH anvendes, så vent til BINH skal aktiveres, og aktiver den.
5. Vent til en *handshake flip-flop* bliver sat (der er registreret et ekko), eller til der er gået over t_{max} ³.
6. Hvis en *handshake flip-flop* er sat (der er registreret et ekko), så nulstil den pågældende *handshake flip-flop*, og aflæs det tilsvarende EKKO register.
7. Er der gået over t_{max} , så hop til 9
8. Gentag fra 5
9. Sæt INIT og BINH lave (afslut måling)-

I det øjeblik et ekko registreres, aktiverer sonar-kontrollogikken, automatisk den pågældende sensorgruppes BLNK signal, der holdes aktivt i et minimumsinterval på 384 til 448 μs . Aktiveringen af BLNK signalet, sætter samtidigt en *handshake flip-flop* i PI/Ten. Hvis minimums tidsintervallet er gået når gruppens ekko register aflæses, deaktiveres gruppens BLNK signal med det samme. Hvis minimums intervallet endnu ikke er gået ved aflæsningen, deaktiveres BLNK, så snart intervallet er slut. Når gruppens BLNK signal er deaktiveret igen, er sensoren klar til at registrere endnu et ekko.

Hver *handshake flip-flop* kan sættes til at generere et interrupt, eller de kan *polles*. Uanset hvordan det registreres at en *flip-flop* er sat, nulstilles den, og det koresponderende ekko register aflæses. Aflæsningen vil nulstille BLNK signalet, der aktiverede *flip-flopen*, enten med det samme eller senere. *Flip-flopen* forbliver imidlertid nulstillet, indtil næste ekko registreres, og BLNK atter går høj.

Der er intet i vejen for at flere *flip-flops* er sat samtidigt, idet de nulstilles individuelt.

Tidsintervallet på 384-448 μs svarer til ultralyds-pulsens længde, og denne pause er nødvendig for at forhindre det samme ekko fra et objekt, i at blive registreret som flere ekkoer. For at skelne to objekter fra hinanden, skal ekkoet fra dem være forskudt mere end 448 μs , svarende til at de er mindst 8 cm fra hinanden, i lydens bevægelsesretning.

³ t_{max} kan være de førnævnte 64 ms, et andet tidsinterval. Evt kan Heimdals egen *timeout* funktion benyttes

Hvor tætte objekter der kan skelnes, ud over de 8 cm, afhænger af programmets reaktionstid, idet et nyt ekko ikke kan registreres før det forrige er aflæst. Vil det sikres at alle ekkoer registreres, skal reaktionstiden være under 448 μ s!

Hvis softwaren kører under et *multi tasking* operativsystem, kan sådanne reaktionstider, kun opnås ved at anvende *time slices* på under 448 μ s, eller ved at anvende interrupts. De korte *time slices* give det relativt langsomme PEP VMPM 68KC CPU modul et problem, idet bogholderiet ved processkift, vil komme til at opage en stor del af dets kapacitet. Til sammenligning, anvender OS-9 til dette modul, typisk *timeslices* på 5-10 ms.

Så længe der arbejdes med multitasking på VMPM 68KC, eller tilsvarende langsomme CPU moduler, bør multi ekko målinger foretages vha. interrupt rutiner (device drivers). Dels fordi responstiden på interrupts er meget lavere end 448 μ s, og dels fordi interruptrutiner optager et minimum af CPUens kapacitet, i forhold til polling.

19.10 PI/T timer

Den indbyggede 24 bits timer i PI/Ten, anvendes ikke direkte i styringen af Heimdal, men det er bekvemt at anvende den som ur, i forbindelse med programmeringen af Heimdal. Timeren kan anvendes som *one shot*, eller periodisk, og den kan sættes op til at generere interrupts.

Der er mange måder at konfigurere timeren på, og her gennemgås kun *one-shot* med og uden interrupt generering. For yderligere information henvises til [19]

19.10.1 One shot uden interrupts

```

PROCEDURE Start_Timer(LONGWORD time)
    Byte_Write(0x21,0x90)          /* TCR = 10010000_b          */
    Byte_Write(0x2B,time BIT_AND 0xFF) /* CPRL = nederste 8 bits af time */
    time = time / 0x100             /* Fjern nederste 8 bits af time */
    Byte_Write(0x29,time BIT_AND 0xFF) /* CPRM = næste 8 bits af den orig. time */
    time = time / 0x100             /* Fjern nederste 8 bits af time */
    Byte_Write(0x27,time BIT_AND 0xFF) /* CPRH = næste 8 bits          */
    Byte_Write(0x21,0x91)          /* TCR = 10010001_b          */
END PROCEDURE

FUNCTION Check_Timeout
    IF Byte_Read(0x35) = 0 THEN
        RETURN FALSE
    ELSE
        RETURN TRUE
    ENDIF
END FUNCTION

```

Hvordan timeren fungerer, bestemmes af *Timer Control Register* (TCR). Ved initialiseringen sættes TCR til 10010000_{bin} = 90_{hex}.

De øverste tre bits (5-7) konfigurerer timeren som en *vectored interrupt* genererende enhed, med interrupts slået fra. Bit 4 får timeren til at kamme over til FFFFFFF_{hex} når den passerer 0, i stedet for at indlæse værdien fra *counter preload* registre. Bit 3 har ingen funktion. Bit 1 og 2 konfigurerer timeren til at anvende PI/Tens 8MHz clock signal, delt med 2⁵ — altså 250Khz — som tællefrekvens, hvilket giver timeren en opløsning på 4 μ s. Bit 0 sat til 0 standser timeren.

Tiden har enhed af 4 μ s intervaller, og indlæses af tre gange. Rækkefølgen af den lave, middel og høje del er underordnet.

Timeren startes, ved at sætte bit 0 i TCR høj. Værdien i *preload* registeret indlæses i *counter* registeret — der ikke kan aflæses pålideligt når timeren kører — og counteren begynder at tælle nedad. Når den

passerer fra 000000_{hex} til $FFFFFF_{hex}$ sættes bit 0 i TSR, og den forbliver sat indtil timeren standses, eller der skrives en maske til TSR, med bit 0 sat høj (*direct reset method*).

19.10.2 Timer med interrupts

Dette tilfælde er identisk med brug uden interrupts, bortset fra tre ting:

Ved initialiseringen skal der skrives en *interrupt vector* til PI/Tens *Timer interrupt vector register* (TIVR).

Bit 5 i TCR skal være sat højt, for at slå interrupts til. Der skal altså skrives $B0_{hex}$ hhv. $B1_{hex}$ for at standse/starte timeren.

I det øjeblik timeren kammer over, sættes bit 0 i TSR højt, hvilket nu bevirker at der genereres et interrupt. Interrupt service rutinen, skal standse timeren (skrive $B0_{hex}$ til TCR). Dels for at nulstille bit 0 i TSR — der koresponderer direkte med PI/Tens *interrupt request* signal — og dels for at forhindre at timeren tæller videre fra $FFFFFF_{hex}$, og dermed genererer et nyt interrupt efter ca. 67 sekunder.

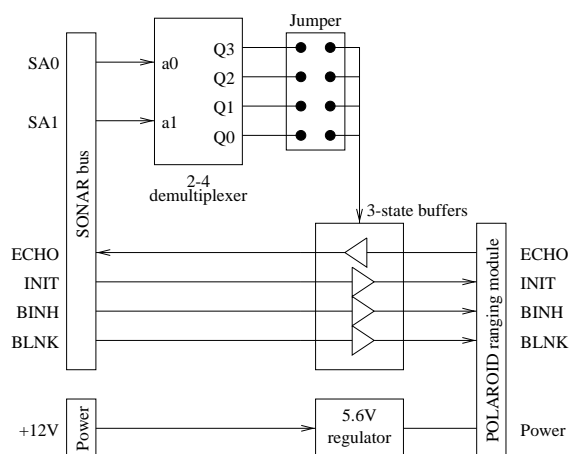
Det er ikke trivielt at skrive interrupt rutiner, og det anbefales kraftigt at sætte sig grundigt ind i MC68230's funktionalitet, samt i hvordan det anvendte CPU modul og operativsystem håndterer interrupts.

19.11 BUS forsats

Polaroids *ranging modules*, kan ikke umiddelbart kobles på en fælles bus. For at blive i stand til at anvende bus teknikken, er det nødvendigt at anvende en *forsats*, der rummer det nødvendige adressedekodning og buffere.

Den forsats jeg har udviklet, har jeg valgt at kalde **Bifrost**

19.11.1 Generel beskrivelse



Figur 19.5: Blokdiagram over bifrost

Bifrost er et 2×5 cm print, med et 10 polet fladkabestik (S2), for tilslutning til sonar-bussen, et 9 polet stripline-stik (S3), for tilslutning til et *ranging modul*, og et to-polet Molex stik (S1), for tilslutning af spændingsforsyning.

Figur 19.5 viser et blokdiagram over Bifrost.

Hjertet i Bifrost, er de fire *three-state buffers*, der kobler eller afbryder *ranging* modulets fire signaler til/fra den fælles bus.

Bussen rummer to adresseliner: SA₀ og SA₁, der dekodes af en 2 til 4 demultiplexer. Adressen på hvert enkelt Bifrost modul, indstilles ved at indsætte en *jumper*, på en ud af fire mulige positioner. Hvis adressen på SA₀₋₁ svarer til jumperens position, tilslutter det pågældende Bifrost modul, *ranging* modulet til den fælles bus. **På hver enkelt sonar-bus, må der ikke være flere Bifrost moduler med samme adresse.**

Hvert Bifrost modul forsynes med 9 — 24V, og rummer spændingsregulatorer, til at forsyne både sig selv og *ranging* modulet med de korrekte spændinger.

19.11.2 Konfiguration og tilslutning

Adressen på Bifrost modulet, vælges ved at placere en *jumper*, på en af de fire positioner J1-J4 J1 svarer til adressen 0, j2 til 1 osv. Der må kun placeres en jumper!

Polaroids *ranging module* (model: PID # 615077) forbindes med et specielt striplinekabel, til Det dertil indrettede stik: S3.

Sonar-bussen består typisk af et 10 leder 0.05 tomme fladkabel, der forbindes til Bifrost via. det 10-polede IDC stik: S2.

Strømforsyningen forbindes til stikket: S1.

Hvis forsyningsspændingen er højere end 12V, eller hvis Bifrost skal anvendes ved temperaturer over 50°C, skal den termiske modstand mellem regultortransistoren: T1, og omgivelserne reduceres til maksimalt 25 K/W.

Stik	ben	signal	Stik	ben	signal	stik	ben	signal
S1	1	GND	S2	1	SA ₀	S3	1	GND
	2	+12V		2	SA ₁		2	_BLNK
				3	GND		3	NC
				4	INIT		4	_INIT
				5	GND		5	NC
				6	BINH		6	NC
				7	BLNK		7	_ECHO
				8	GND		8	_BINH
				9	ECHO		9	V+ (5.6V)
				10	GND			

Tabel 19.6: Forbindelser til Bifrost

19.11.3 Specifikationer

Temperaturområde	
Med køling af regulatortransistor (max. 25K/W)	0 — 70°C
Uden køling, og ($V_f \leq 12V$)	0 — 50°C
Strømforsyning	
V_f forsyningsspænding uden køling	9 — 12V
V_f forsyningsspænding med køling (25K/W)	9 — 24V
I_f strømforbrug	max. 40mA (uden <i>ranging module</i>)
V_+ (Spænding til <i>ranging module</i>)	5.5 — 6.0 V
I_{V+} (Strøm til <i>ranging module</i>)	max. 150 mA midlet over 1 sekund
$I_{V+,peak}$	max. 5A.
Indgange	(strøm positiv ind i indgangen)
V_{IH} høj indgangsspænding	+2.0 til +5.0 V
V_{IL} lav indgangsspænding	0.0 til +0.8 V
I_{IH} indgangsstrøm ved højt signl.	max. 0.5 mA
I_{IL} indgangsstrøm ved lavt signl.	max. -0.5 mA
Udgange	(strøm positiv ud ad udgangen)
V_{OH} høj udgangsspænding	min. 2.0V
V_{OH} hvis $I_{OH} \leq 3mA$	min. 2.4V — typ. 3.4V
V_{OL} lav udgangsspænding	max. 0.5V
I_{IH} belastning ved udg. høj	max. 10mA
I_{IL} belastning ved udg. lav	max. -10mA
Skifte tid	
t_s Tid fra adresseskift, til til-/fra-kobling	max. 100ns

19.12 Device driver

For at udnytte Heimdals faciliteter fuldt ud, under OS-9, er der udviklet en device driver. I dette afsnit beskrives det hvordan driveren bruges, og hvordan man ændrer default parametre i device desriptoren.

19.12.1 Installation

Alle filer med relation til device driveren, ligger på en DS, DD 3.5diskette, mærket: *PEP filer til Specialeprojekt, Anders Stengaard Sørensen*, i biblioteket: HEIMDAL/ .

Ønskes det kun at bruge driveren, er det tilstrækkeligt at anvende filerne i /d0/HEIMDAL/EXE. Skal driveren være tilgængelige for alle brugere på en PEP, lægges filerne: **hmd**, **hmddrv** og **test** i et fælles bibliotek, f.eks. /dd/HEIMDAL. Skal driveren kun bruges af en selv, kan filerne lægges i ens private **EXE** bibliotek.

For at fungere skal device driveren **hmddrv** og device desriptoren **hmd**, samt file manageren **/dd/SYSSRC/BOOTOBJS/sbf** læses ind i hukommelsen vha. **load** kommandoen. Skal Heimdals bruges regelmæssigt, kan dette gøres vha. ens egen .login fil.

```
User name?: anders
Password:
Process #05 logged on      98/01/07 17:43:25
Welcome!

Du er i: /dd/USR/ANDERS
anders> makdir EXE
anders> copy /d0/HEIMDAL/EXE/* -w=EXE
copying /d0/HEIMDAL/EXE/hmd to EXE/hmd
copying /d0/HEIMDAL/EXE/hmddrv to EXE/hmddrv
copying /d0/HEIMDAL/EXE/test to EXE/test
anders> load EXE/hmd -d
anders> load EXE/hmddrv -d
anders> load /dd/SYSSRC/BOOTOBJS/sbf -d
```

Figur 19.6: Installation af device driver

19.12.2 Device descriptor og default parametre

Alle hardware mæssige parametre der relaterer til driveren, opbevares i device desriptoren. Flere forskellige device desriptorer kan relatere til samme driver, med forskellige parametre, og optræde under forskellige navne.

Der er kun en device descripter blandt filerne, **hmd**. **hmd** rummer en mængde hard- og software relevante parametre, hvor de vigtigste for Heimdals er følgende:

Navn	beskrivelse	værdi
M\$Port	Heimdals basisadresse	\$873e0000
M\$Vector	Basis interrupt vector	200
M\$IRQLvl	Interrupt level	4
PD_BINH	BINH tidsinterval	1000 (μ s)
PD_TIME	<i>Lytte</i> tidsinterval	64000 (μ s)
PD_SENSOR	Sensorliste	\$81,\$81,\$81,\$81

De 4 midterste cifre i basisadressen, skal korrespondere med Heimdals adressevælger kontakter.

Basis interrupt vektoren, skal være inden for området af lovlige OS-9 interrupt vektorer, og iøvrigt være et multiplum af 4.

Interrupt level skal være i intervallet: 1 . . . 6. Driveren programmerer selv Heimdals til det valgte niveau.

Sensorlisten består af fire tal, der koder for hvilken sensor i hver af de fire sensorgrupper, der anvendes ved en måling. En værdi på 0, deaktiverer den pågældende gruppe. Hvis bit 7 er sat, betyder det at driveren anvender de fire sensorer cyklisk, i stigende rækkefølge, startende med den sensor der er angivet af de lavere bits.

19.12.3 Måling vha. driveren

Driveren understøtter måling, med detektion af op til 98 ekkoer. For at foretage en måling, anvendes de almindelige operativsystemkald til at åbne og læse fra en fil. For at lave en måling med detektion af et enkelt ekko, på alle fire grupper, kan man anvende følgende programstump i C.

```
#include <stdio.h>
#include <modes.h>
#define N_ECHO 1
void main()
{
    int fd,n;
    unsigned short buffer[N_ECHO+1][4];

    fd=open("/hmd",S_IREAD+S_IWRITE);
    read(fd,buffer,8*(N_ECHO+1));

    for(n=0;n<4;n++) {
        printf("Gruppe %d, sensor: %d, Measured value: %d\n",n+1,buffer[0][n],buffer[1][n]);
    }
    close(fd);
    exit(0);
}
```

Device driveren fungerer, over for brugeren, som en almindelig fil, hvis navn er `"/"` efterfulgt af navnet på device descriptor (**hmd**). Når driveren er *åbnet*, foretages en måling ved at læse fra den vha. **read** funktionen. Bufferen der bruges ved læsning er et todimensionelt array af 16 bits heltal, der altid har fire søjler, og 2...99 rækker. Hver søjle repræsenterer måleværdier fra en sensor-gruppe.

index	0	1	2	3
index	Gruppe 1	Gruppe 2	Gruppe 3	Gruppe 4
0	Sensor nr.	Sensor nr.	Sensor nr.	Sensor nr.
1	Ekko tid	Ekko tid	Ekko tid	Ekko tid
2	Ekko tid	Ekko tid	Ekko tid	Ekko tid
3	Ekko tid	Ekko tid	Ekko tid	Ekko tid
4	Ekko tid	Ekko tid	Ekko tid	0
5	Ekko tid	Ekko tid	Ekko tid	Udefineret
6	Ekko tid	0	Ekko tid	Udefineret
7	Ekko tid	Udefineret	0	Udefineret
...
...
...
...

Tabel 19.7: Struktur af læsebuffer

I det øjeblik **read** kaldes, startes en måling. Målerutinen er fuldt interrupt drevet, og den kaldende bruger-proces suspenderes indtil målingen er færdig. Målingen tager sammenlagt $PD_BINH + PD_TIME$, altså 65 ms med de parametre der er sat op i den medfølgende device descriptor.

Efter en måling, indeholder den første række numrene på de sensorer der blev brugt af hver grupe. Række 1 og fremefter er, for hver søjle, en nultermineret liste⁴ af ekko aflæsninger. Ud fra den bufferstørrelse der angives til **read** funktionen, afgør driveren hvor mange ekkoer der skal aflæses. Bufferstørrelsen skal være et multiplum af 8 (4 words/ række), og ligge i intervallet: 16...792. Er bufferstørrelsen 16 aflæses max. 1 ekko, er den 24 aflæses max 2, etc.

Som en kuriøsitet kan OS-9 utiliten **dump** bruges til at foretage målinger med, idet **dump** netop udskriver en fil 16 bytes ad gangen. Kommandoen:

```
> dump /hmd
```

starter en serie af målinger, der kan afsluttes med \hat{C} .

⁴Hvis alle pladser i listen bliver brugt, er den ikke nultermineret

19.12.4 Dynamisk ændring af parametre

Det er ikke altid ønskeligt at skifte cyklisk mellem sensorerne i hver gruppe, i stedet ønskes en direkte styring af hvilke sensorer der anvendes ved hver måling. I nogle tilfælde kan det være ønskeligt at kunne ændre driverens to tidsparametre på samme måde.

Både tidsparametrene og sensorlisten opdateres ved at skrive til driveren vha. **write**. For at opdatere sensorlisten skrives netop fire bytes, og for at opdatere tidsparametrene skrives netop 8 bytes. Alle andre bufferstørrelser end 4 og 8 er ulovlige ved skrivning til driveren.

Sensorlisten er organiseret som et array af fire bytes (`unsigned char`), der repræsenterer gruppe 1,2,3, og 4. Formatet af hver byte er beskrevet i afsnit 19.12.2.

Tidsparametrene er organiseret som et array af to 32 bits heltal (`unsigned long int`). Det første koder for **BINH** intervallet, og det andet for det tidsinterval hvor der lyttes efter ekkoer. Begge intervaller angives i μ s. Af hardwaremæssige grunde skal begge tal være i intervallet 4...67108860, men for **BINH** giver det kun mening med tider i intervallet fra ca. 500 til 2380 μ s. Mht. *lytte* intervallet ligger tider på over 64000 μ s uden for sensorens angivne rækkevidde på 11m.

Vil man f.eks. opdatere sensorlisten, så gruppe 1 er slået fra, gruppe 2 altid anvender sensor 1, og gruppe 3 og 4 skifter cyklisk mellem sensorerne, startende med sensor 4 hhv. 2, kan nedenstående programstup sættes ind i ovenstående program.

```
unsigned char      sensorlist[4];
.
.
sensorlist[0]=0;sensorlist[1]=1;sensorlist[2]=0x84;sensorlist[3]=0x82;
write(fd,sensorlist,4);
```

Vil man ændre tidsintervallerne til 700 hhv. 32000 μ s, kan nedenstående programstup anvendes.

```
unsigned long int  timers[2];
.
.
timers[0]=700;timers[1]=32000;
write(fd,timers,8);
```

Driveren *glemmer* de nye værdier af parametrene, når den deinitialiseres. Det sker når den ikke længere bruges. Følges ovenstående eksempler vil den blive initialiseret hver gang programmet startes, og kalder **open**, og deinitialiseres hver gang programmet afsluttes, og dermed kalder **close**. Næste gang driveren initialiseres hentes parametrene på ny fra descriptoren.

Hvis driveren skal huske parametrene mellem hvert kald til programmet, eller huske hvor langt den er nået mht. cyklisk måling, skal driveren specifikt initialiseres, f.eks. vha. shell kommandoen:

```
> in iz hmd
```

Driveren kan så igen deinitialiseres vha

```
> de in iz hmd
```

19.12.5 Assemblering af kildetekster

Vil man lave device descriptorer med andre parametre end den medfølgende, eller vil man forbedre device driveren, ligger samtlige kildetekster også på den ovennævnte diskette, i biblioteket: HEIMDAL/SOURCE. Kopieres hele biblioteket til brugerens hjemmekatalog, kan kildeteksterne nemt ændres og reassembleres, under forudsætning af at brugeren er i brugergruppe 0 (superbruger).

Hvis stiangivelserne i den medfølgende makefil tilpasses brugerens hjemmekatalog, kan den uden videre bruges til at genopbygge descriptor (`make hmd`), driver (`make hmddrv`) og det medfølgende testprogram (`make test`).

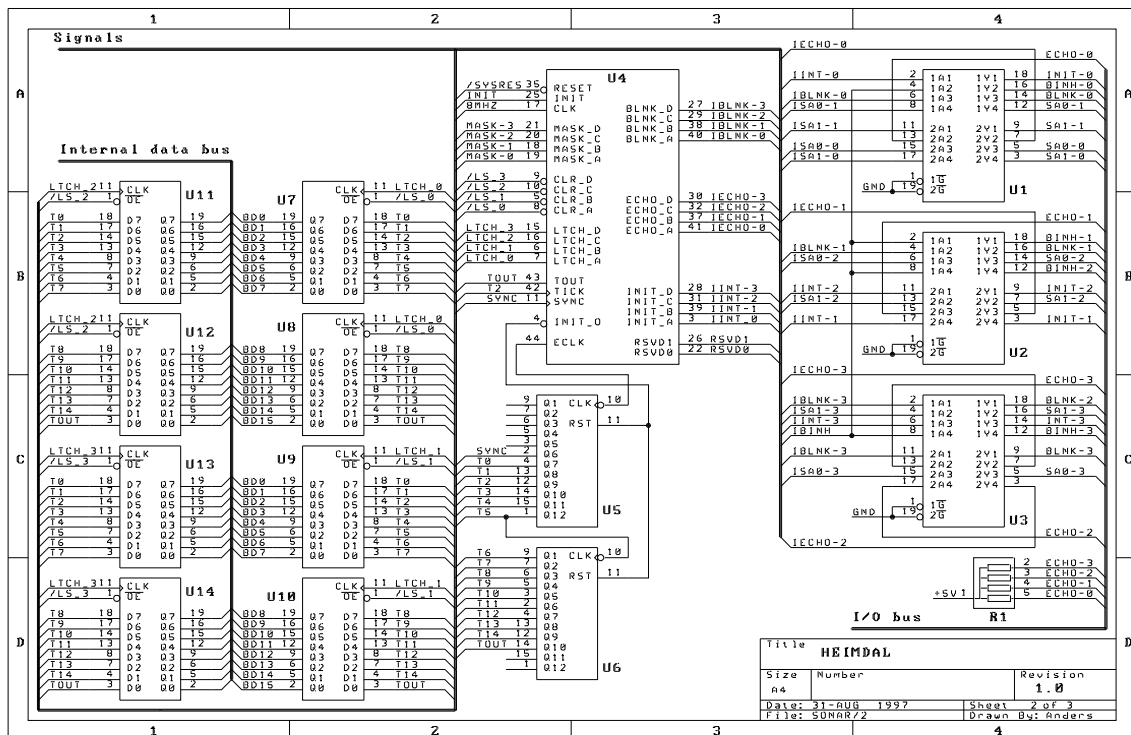
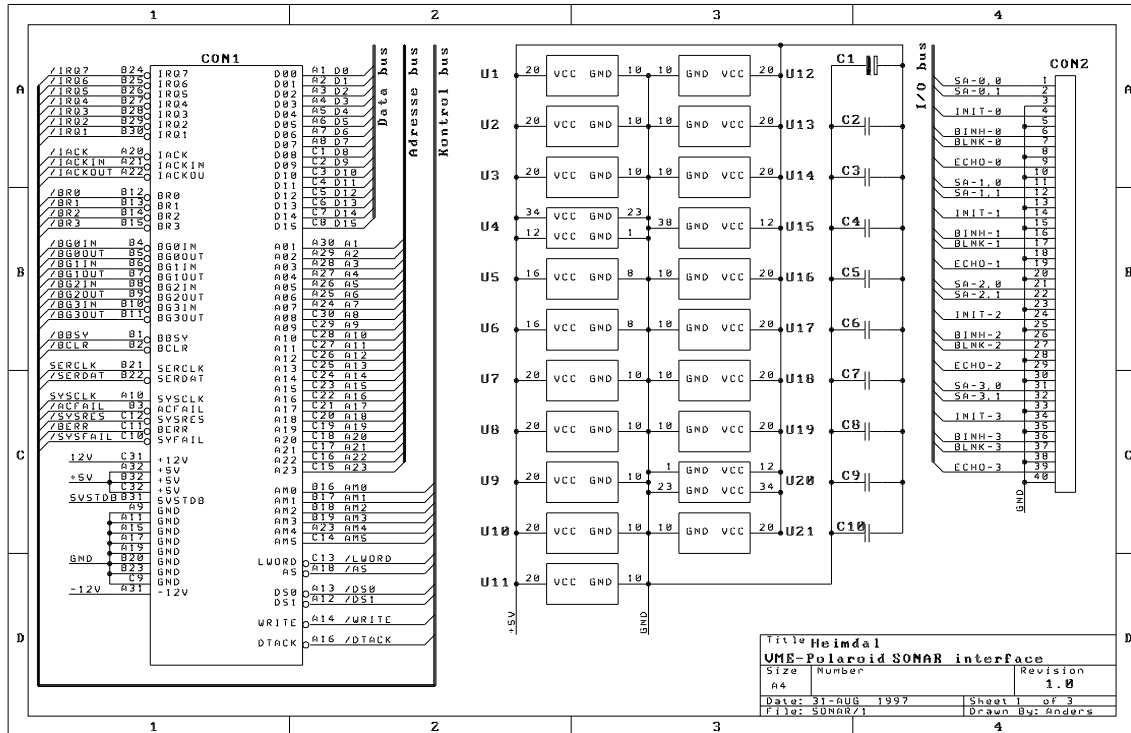
19.12.6 Yderligere information

For yderligere informationer om device driveren, henvises til kapitlet: **Udvikling af softwareinterface til Heimdall** i min specialerapport. Kapitlet beskiver designet og udviklingen af device driveren, og beskriver de filer der indgår i opbygningen af driver, descriptor, og testprogram.

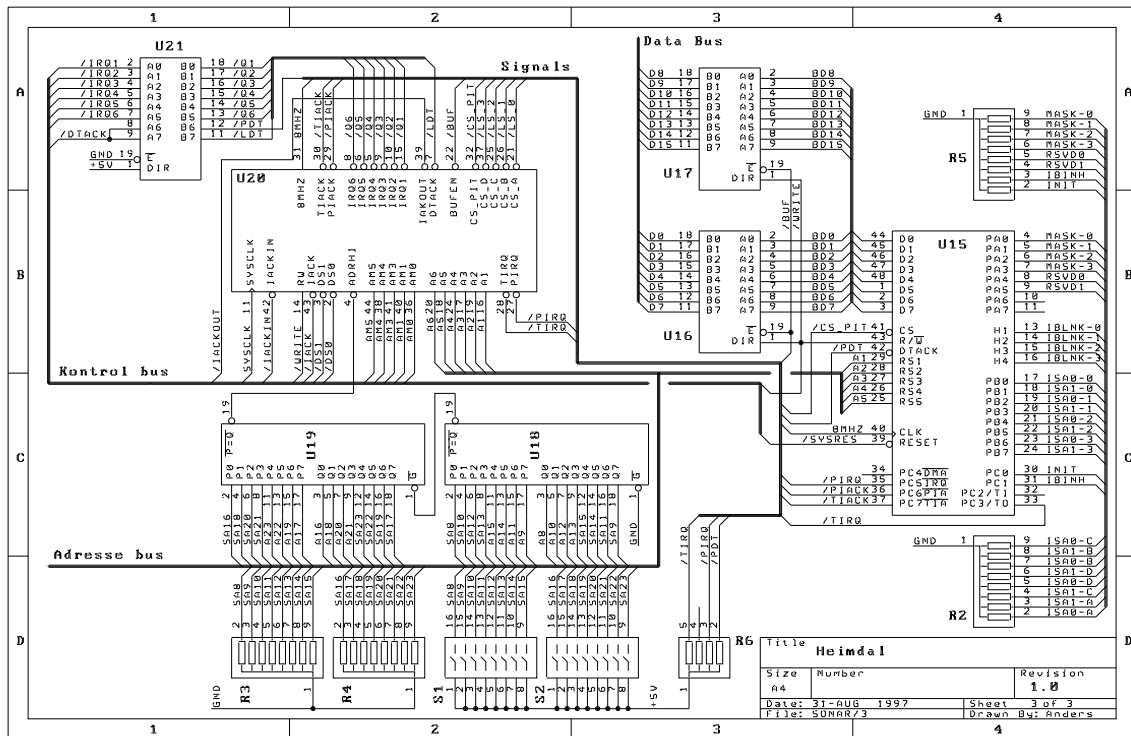
For informationer om OS-9's I/O system, device drivers og file managers generelt, henvises til [16], [13], og [21, Kap. 5,6, og 17].

For informationer om RBF file mangeren henvises til [21, Kap. 9].

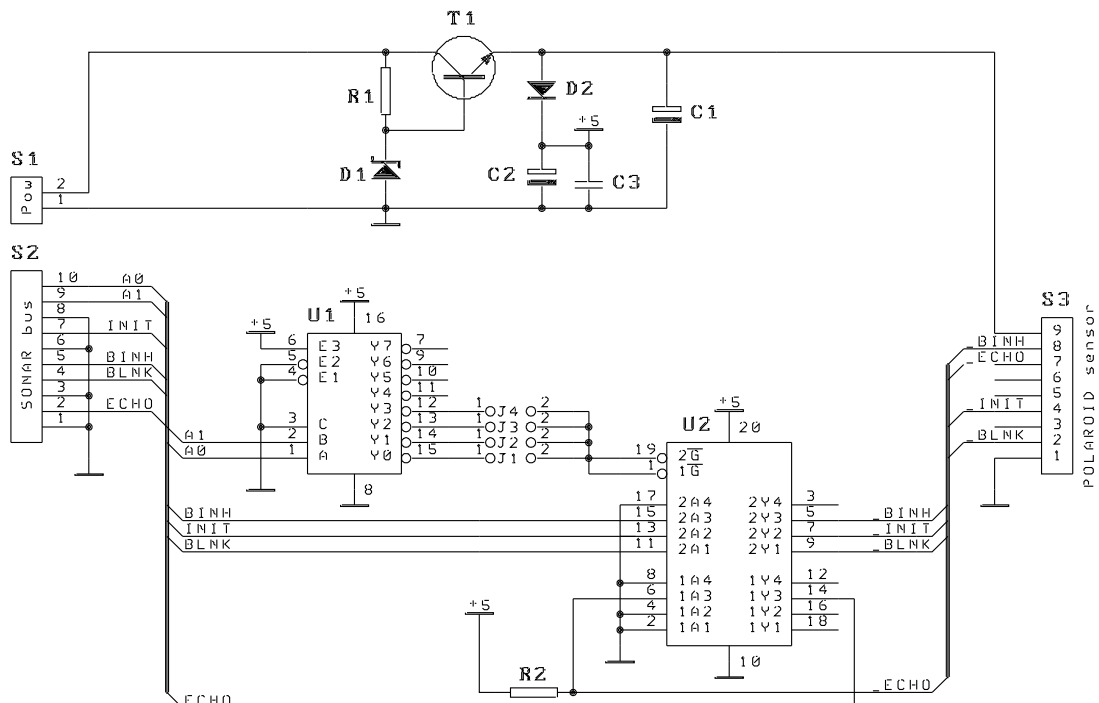
19.13 Diagrammer og printudlæg



Figur 19.7: Diagram over Heimdal (1 og 2 af 3)



Figur 19.8: Diagram over Heimdal (3 af 3)



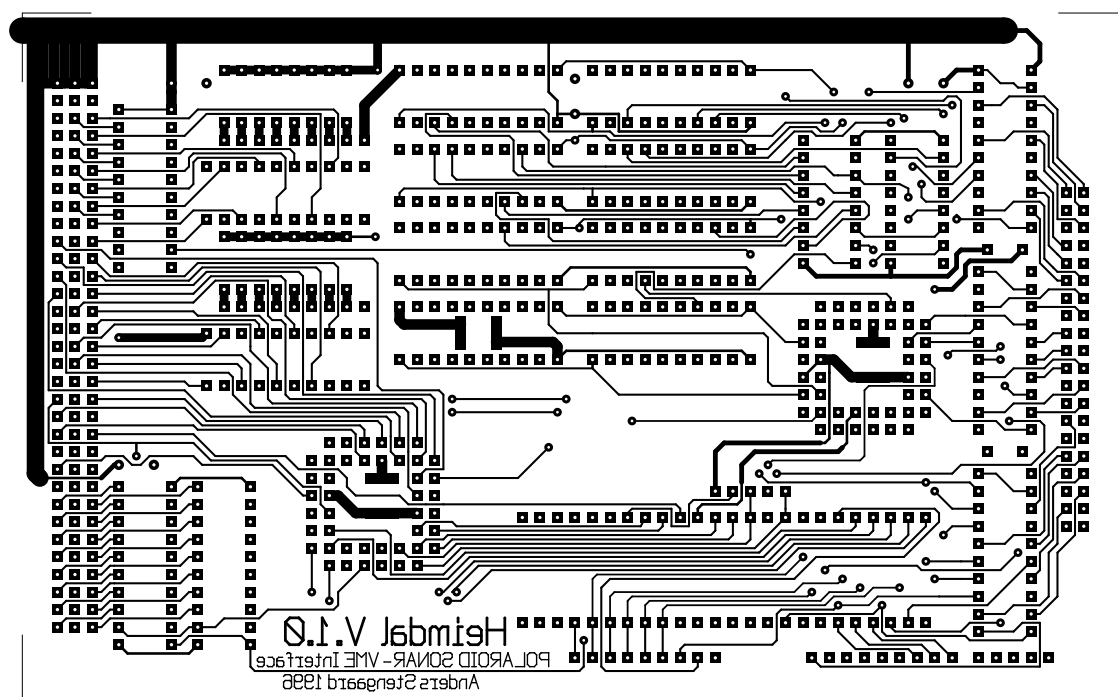
Figur 19.9: Diagram over Bifrost (1 af 1)

R1	4×	SIL-5		U1	74LS244		
R2	8×	SIL-9		U2	74LS244		
R3	8×	SIL-9		U3	74LS244		
R4	8×	SIL-9		U4	ispLSI1016	prog. logik	sonar control
R5	8×	SIL-9		U5	74HCT4040	12 bits tæller	
R6	4×	SIL-5		U6	74HCT4040	12 bits tæller	
C1		Elektrolyt	$\geq 5V$	U7	74LS374		
C2	100nF	Sibatit	$\geq 5V$	U8	74LS374		
C3	100nF	Sibatit	$\geq 5V$	U9	74LS374		
C4	100nF	Keramisk SMD	$\geq 5V$	U10	74LS374		
C5	100nF	Keramisk SMD	$\geq 5V$	U11	74LS374		
C6	100nF	Sibatit	$\geq 5V$	U12	74LS374		
C7	100nF	Sibatit	$\geq 5V$	U13	74LS374		
C8	100nF	Sibatit	$\geq 5V$	U14	74LS374		
C9	100nF	Sibatit	$\geq 5V$	U15	MC68230	PI/T	
C10	100nF	Keramisk SMD	$\geq 5V$	U16	74LS245		
CON1	DIN41612-C	96 pol		U17	74LS245		
CON2	IDC 90°	40 pol		U18	74LS688		
S1	DIL switch	8 pol		U19	74LS688		
S2	DIL switch	8 pol		U20	ispLSI 1016	prog. logik	VME control
				U21	74LS641		

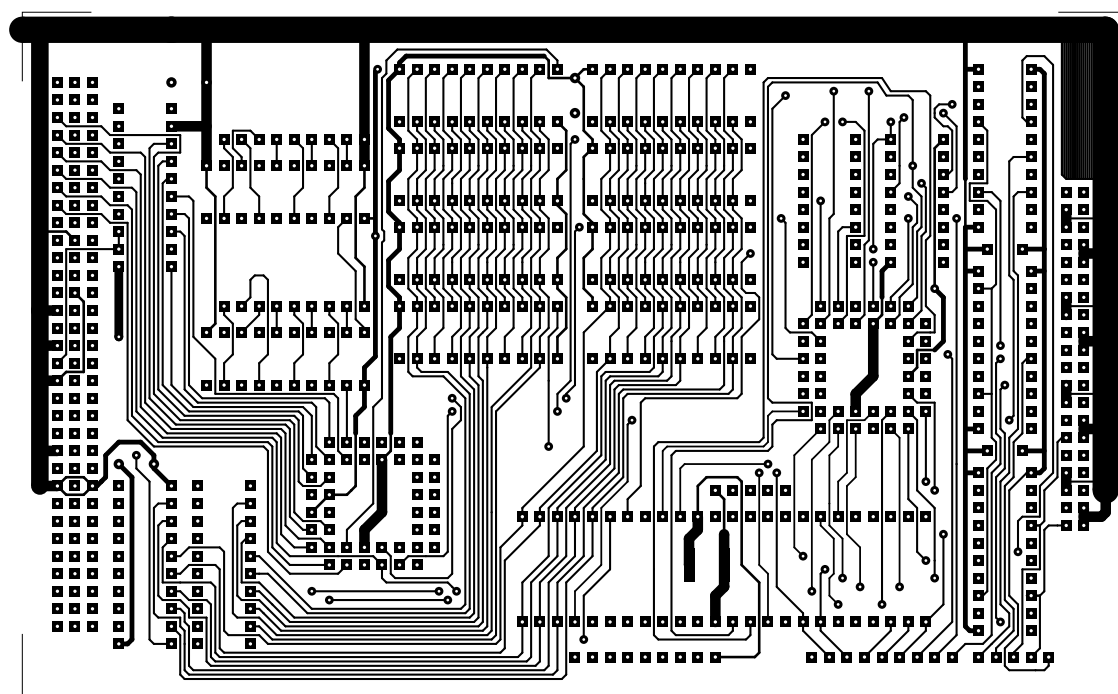
Tabel 19.8: Komponentliste til Heimdal

R1	10k Ω	$\pm 5\%$	1/4W	J1	Jumper	2 pol	
R2	10k Ω	$\pm 5\%$	1/4W	J2	Jumper	2 pol	
C1	1000 μF	Elektrolyt	$\geq 6V$	J3	Jumper	2 pol	
C2	47 μF	Elektrolyt	$\geq 6V$	J4	Jumper	2 pol	
C3	100nF	Sibatit	$\geq 6V$	D1	Zenerdiode	6.8V	400mW
S1	Molex	2 pol		D2	1N4001	Ensrette diode	
S2	IDC	10 pol		T1	BD645	NPN effekt	
S3	Stripline	9 pol	Polaroid	U1	74LS138		SMD
				U2	74LS244		SMD

Tabel 19.9: Komponentliste til Bifrost

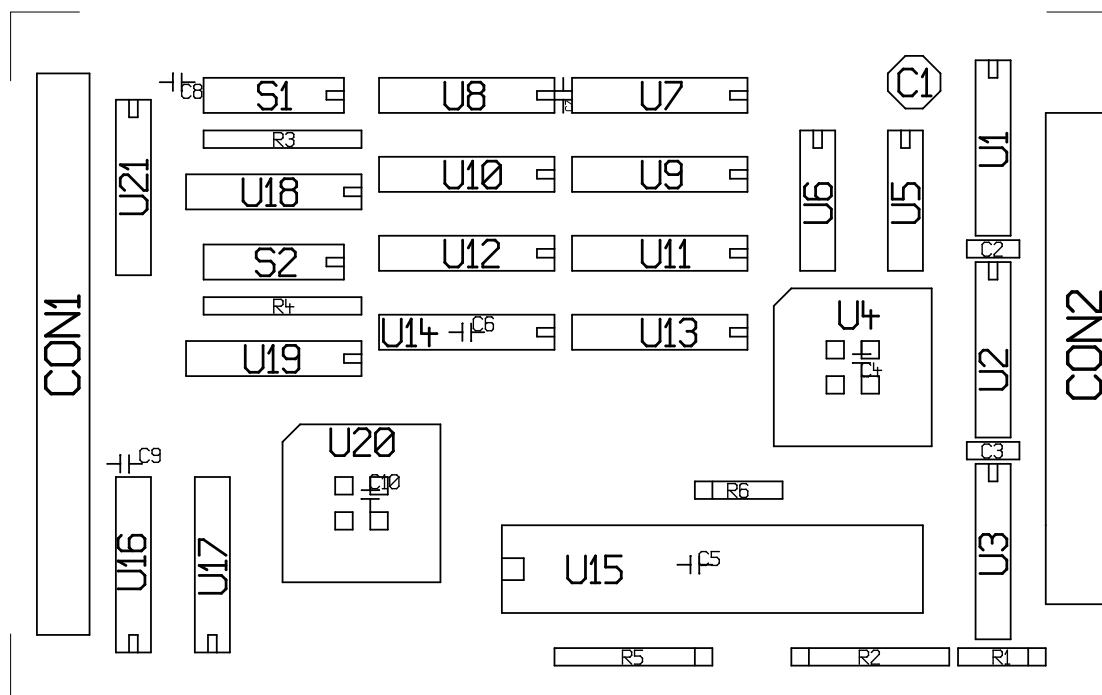


(a) Loddeside

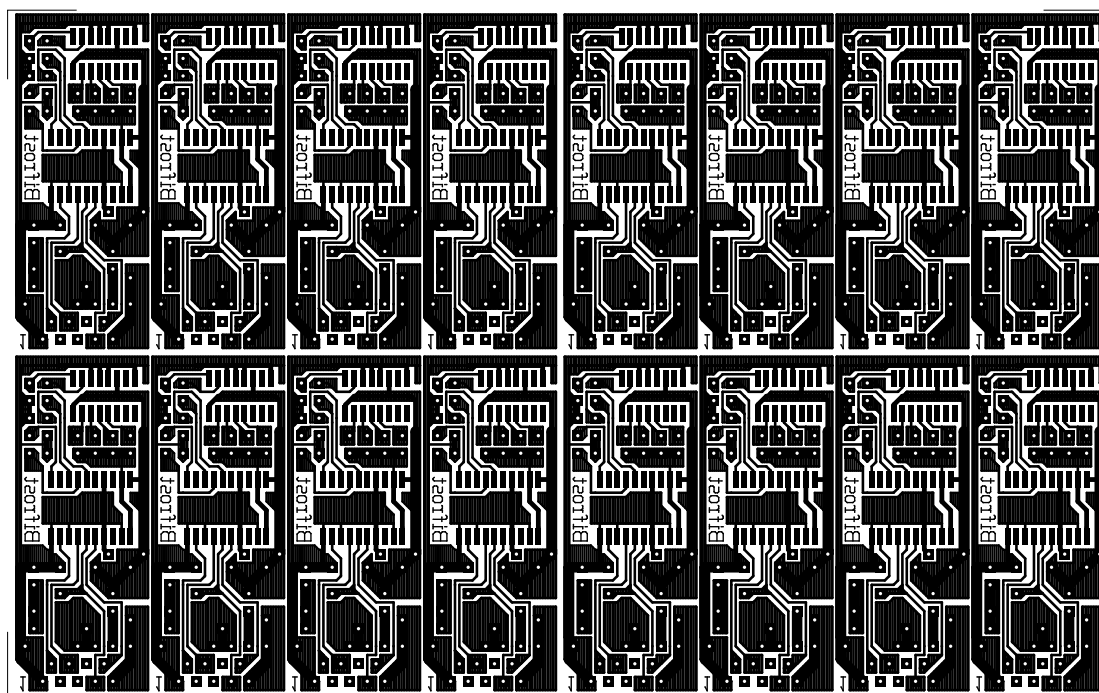


(b) Komponentside

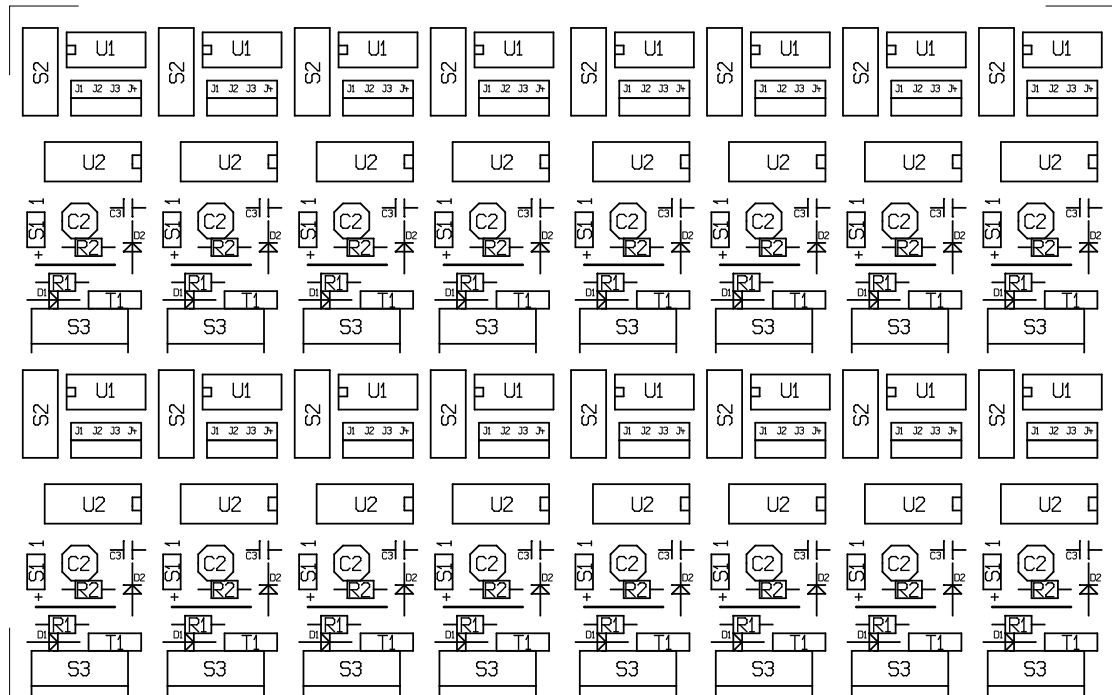
Figur 19.10: Printudlæg af Heimdal 160mm × 100mm



Figur 19.11: Komponentplacering på Heimdal 160mm×100mm



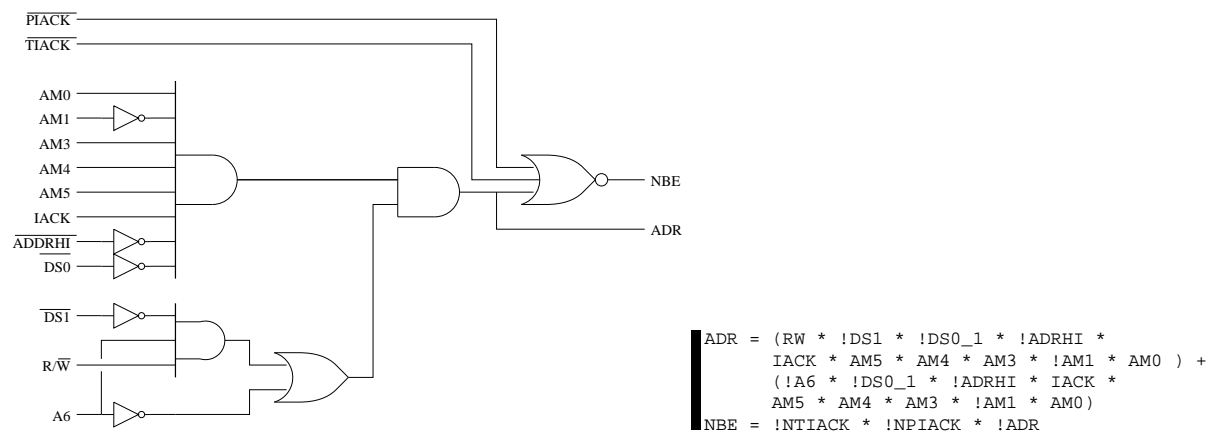
Figur 19.12: Printudlæg af Bifrost. 160mm×100mm enkeltsidet (16 identiske kredsløb)



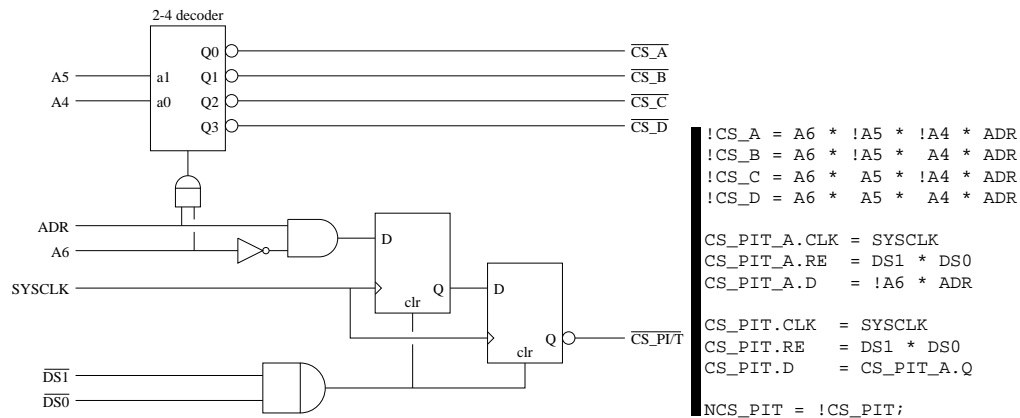
Figur 19.13: Komponentplacering på Bifrost 160mm×100mm (16 identiske kredsløb)

19.14 Programmerbar logik

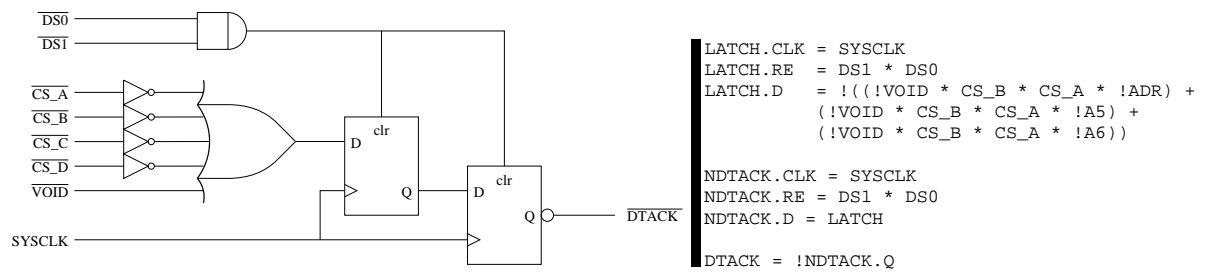
19.14.1 BUS kontrollogik



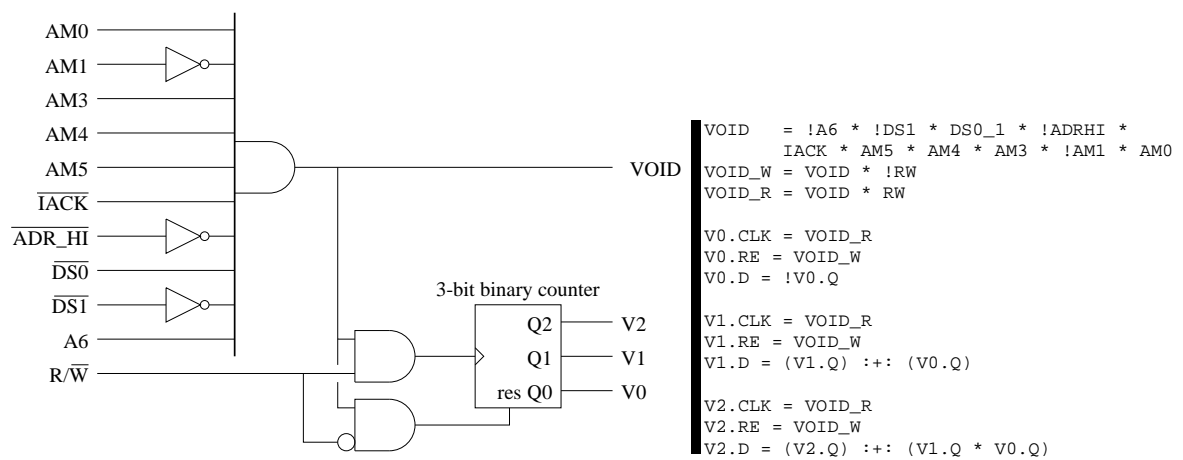
Figur 19.14: Adgangs dekoder



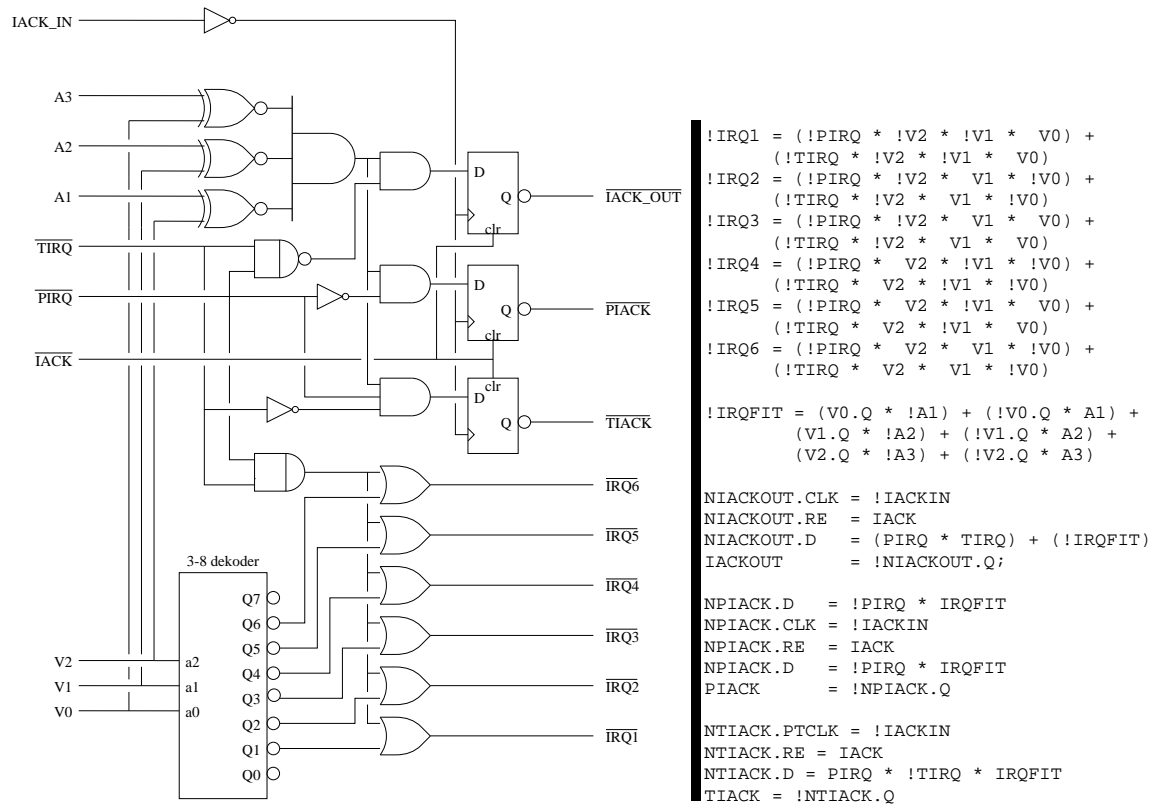
Figur 19.15: Chip vælger



Figur 19.16: DTACK generator



Figur 19.17: IRQ niveau kontrol



Figur 19.18: Interrupt kontrol

Programlistning

OBS! listningen er håndredigeret for at fjerne overflødig information, og fremme overskueligheden.

Mon Sep 01 17:47:37 1997
ADRDEK4.rpt generated using Lattice pDS Version 2.61

External I/O Pin Report

Part: ispLSI1016-60LJ44
Design Name: Adr dekod
Design Revision: 0.1
Author: Anders Stengaard Sørensen
Project Name: Heimdal
Description:
Verify Status: Complete
Route Status: Complete
Data Used: Post-Route

Pin Number	Pad Name	Pin Type	Pullup	Pin Number	Pad Name	Pin Type	Pullup
1	GND	Gnd		23	GND	Gnd	
2	_DS0	Input	Yes	24	_A4	Input	Yes
3	_DS1	Input	Yes	25	_CS_C	Output	Yes
4	_ADRHI	Input	Yes	26	_CS_B	Output	Yes
5	_IRQ4	Output	Yes	27	_PIRQ	Input	Yes
6	_IRQ5	Output	Yes	28	_TIRQ	Input	Yes
7	_DTACK	Output	Yes	29	_PIACK	Output	Yes
8	_IRQ6	Output	Yes	30	_TIACK	Output	Yes
9	_IRQ3	Output	Yes	31	_MHZ8	Output	Yes
10	_IRQ2	Output	Yes	32	_CS_PIT	Output	Yes
11	_SYSCLK	Input	Yes	34	VCC	Vcc	
12	VCC	Vcc	35	XRESET	Input	No	
14	_RW	Input	Yes	36	_AM0	Input	Yes
15	_IRQ1	Output	Yes	37	_CS_D	Output	Yes
16	_A1	Input	Yes	38	_AM4	Input	Yes
17	_A3	Input	Yes	39	_IACKOUT	Output	Yes
18	_A5	Input	Yes	40	_AM1	Input	Yes
19	_A2	Input	Yes	41	_AM3	Input	Yes
20	_A6	Input	Yes	42	_IACKIN	Input	Yes
21	_CS_A	Output	Yes	43	_IACK	Input	Yes
22	_BE	Output	Yes	44	_AM5	Input	Yes

```

*****
STATISTICS FOR GLB A0, ROUTED LOCATION: A1
*****
GLB Input List:          GLB Output List:
I1   : A1                O0   : !IRQFIT
I2   : A3                O1   : V2
I4   : A2                O2   : V1
I5   : V2                O3   : V0
I6   : V1
I7   : V0
I13  : VOID_R
I14  : VOID_W
RESET : !RESET

CELL EQUATIONS:
-----
V0.PTCLK = VOID_R;      V0.RE = VOID_W;
V1.PTCLK = VOID_R;      V1.RE = VOID_W;
V2.PTCLK = VOID_R;      V2.RE = VOID_W;

V0.D = !V0.Q;
V1.D = (V1.Q) $$ (V0.Q);
V2.D = (V2.Q) $$ (V1.Q & V0.Q);
!IRQFIT = (V0.Q & !A1) # (!V0.Q & A1) # (V1.Q & !A2) # (!V1.Q & A2) # (V2.Q & !A3) # (!V2.Q & A3);

*****
STATISTICS FOR GLB A1, ROUTED LOCATION: A7
*****
GLB Input List:          GLB Output List:
I0   : AM0              O0   : ADR
I1   : AM4              O1   : VOID_R
I2   : DS0_1            O2   : VOID_W
I3   : AM1              O3   : VOID
I4   : AM3
I5   : A6
I6   : IACK
I7   : AM5
I8   : DS1
I9   : ADRHI
I12  : RW

CELL EQUATIONS:
-----
VOID = !A6 & !DS1 & DS0_1 & !ADRHI & IACK & AM5 & AM4 & AM3 & !AM1 & AM0;
VOID_W = !RW & !A6 & !DS1 & DS0_1 & !ADRHI & IACK & AM5 & AM4 & AM3 & !AM1 & AM0;
VOID_R = RW & !A6 & !DS1 & DS0_1 & !ADRHI & IACK & AM5 & AM4 & AM3 & !AM1 & AM0;
ADR = (RW & !DS1 & !DS0_1 & !ADRHI & IACK & AM5 & AM4 & AM3 & !AM1 & AM0) #
      (!A6 & !DS0_1 & !ADRHI & IACK & AM5 & AM4 & AM3 & !AM1 & AM0);

*****
STATISTICS FOR GLB A6, ROUTED LOCATION: B0
*****
GLB Input List:          GLB Output List:
IN0  : AAM0             O0   : AM0
IN1  : DDS0             O1   : DS0
O2   : DS0_1

CELL EQUATIONS:
-----
DS0_1 = DDS0;
AM0 = AAM0;
DS0 = DDS0;

*****
STATISTICS FOR GLB B0, ROUTED LOCATION: A6
*****
GLB Input List:          GLB Output List:
I6   : NPIACK           O0   : PIACK
I7   : NTIACK           O1   : TIACK
I10  : MHZ8             O2   : MHZ8
CLK0 : SYSCLK
RESET : !RESET

CELL EQUATIONS:
-----
MHZ8.CLK = SYSCLK;
MHZ8.D = !MHZ8.Q;
TIACK = !NTIACK;
PIACK = !NPIACK;

*****
STATISTICS FOR GLB B1, ROUTED LOCATION: B1
*****
GLB Input List:          GLB Output List:
I4   : TIRQ             O1   : NIACKOUT
I5   : PIRQ             O2   : NPIACK
I9   : IACK             O3   : NTIACK
I10  : IACKIN
I11  : !IRQFIT
RESET : !RESET

CELL EQUATIONS:
-----
NTIACK.PTCLK = !IACKIN;   NTIACK.RE = IACK;
NPIACK.PTCLK = !IACKIN;   NPIACK.RE = IACK;
NIACKOUT.PTCLK = !IACKIN; NIACKOUT.RE = IACK;

NTIACK.D = PIRQ & !TIRQ & IRQFIT;
NPIACK.D = !PIRQ & IRQFIT;
NIACKOUT.D = (PIRQ & TIRQ) # (!IRQFIT);

*****
STATISTICS FOR GLB B2, ROUTED LOCATION: B2
*****
GLB Input List:          GLB Output List:
I2   : LATCH            O0   : DTACK
I3   : ADR              O1   : NDTACK

```

```

I4      : NCS_PIT_A      O2      : NCS_PIT
I6      : NDTACK         O3      : NCS_PIT_A
I7      : DS1
I10     : A6
I14     : DS0
CLK0    : SYSCLK
RESET   : !RESET

CELL EQUATIONS:
-----
NCS_PIT_A.CLK = SYSCLK;      NCS_PIT_A.RE = DS1 & DS0;
NCS_PIT.CLK   = SYSCLK;      NCS_PIT.RE   = DS1 & DS0;
NDTACK.CLK    = SYSCLK;      NDTACK.RE    = DS1 & DS0;

NCS_PIT_A.D = !A6 & ADR;
NCS_PIT.D   = NCS_PIT_A.Q;
NDTACK.D     = LATCH;
DTACK        = !NDTACK.Q;

*****
STATISTICS FOR GLB B3, ROUTED LOCATION: A0
*****
GLB Input List:      GLB Output List:
I3      : A5         O0      : !CS_C
I5      : A6         O3      : CS_PIT
I10     : NCS_PIT
I12     : ADR
I13     : A4

CELL EQUATIONS:
-----
CS_PIT = !NCS_PIT;
!CS_C  = A6 & A5 & !A4 & ADR;

*****
STATISTICS FOR GLB B4, ROUTED LOCATION: A2
*****
GLB Input List:      GLB Output List:
I3      : A5         O1      : !CS_B
I5      : A6         O2      : !CS_A
I6      : NPIACK     O3      : NBE
I7      : NPIACK
I12     : ADR
I13     : A4

CELL EQUATIONS:
-----
NBE = !NTIACK & !NPIACK & !ADR;
!CS_B = A6 & !A5 & A4 & ADR;
!CS_A = A6 & !A5 & !A4 & ADR;

*****
STATISTICS FOR GLB B5, ROUTED LOCATION: A3
*****
GLB Input List:      GLB Output List:
IN0     : READWRITE  O0      : RW
IN1     : AA4        O1      : A4

CELL EQUATIONS:
-----
A4 = AA4;
RW = READWRITE;

*****
STATISTICS FOR GLB B6, ROUTED LOCATION: B6
*****
GLB Input List:      GLB Output List:
I4      : TIRQ       O1      : !IRQ6
I5      : PIRQ       O3      : !IRQ5
I8      : V0
I9      : V1
I10     : V2

CELL EQUATIONS:
-----
!IRQ6 = (!PIRQ & V2 & V1 & !V0) # (!TIRQ & V2 & V1 & !V0);
!IRQ5 = (!PIRQ & V2 & !V1 & V0) # (!TIRQ & V2 & !V1 & V0);

*****
STATISTICS FOR GLB B7, ROUTED LOCATION: B
*****
GLB Input List:      GLB Output List:
I4      : TIRQ       O2      : !IRQ4
I5      : PIRQ       O3      : !IRQ2
I8      : V0
I9      : V1
I10     : V2

CELL EQUATIONS:
-----
!IRQ4 = (!PIRQ & V2 & !V1 & !V0) # (!TIRQ & V2 & !V1 & !V0);
!IRQ2 = (!PIRQ & !V2 & V1 & !V0) # (!TIRQ & !V2 & V1 & !V0);

*****
STATISTICS FOR GLB B0_part1, ROUTED LOCATION: B5
*****
GLB Input List:      GLB Output List:
I10     : NIACKOUT   O2      : IACKOUT

CELL EQUATIONS:
-----
IACKOUT = !NIACKOUT;

*****
STATISTICS FOR GLB B3_part1, ROUTED LOCATION: B7
*****
GLB Input List:      GLB Output List:

```

```

I0      : VOID          00      : !CS_D
I2      : A4            01      : LATCH
I3      : ADDR
I5      : !CS_A
I6      : !CS_B
I7      : DS1
I10     : A6
I12     : A5
I14     : DS0
CLK0    : SYSCLK
RESET   : !RESET

CELL EQUATIONS:
-----
LATCH.CLK = SYSCLK;
LATCH.RE  = DS1 & DS0;
LATCH.D   = !((!VOID & CS_B & CS_A & !ADR) # (!VOID & CS_B & CS_A & !A5) #
              (!VOID & CS_B & CS_A & !A6));
!CS_D     = A6 & A5 & A4 & ADR;

*****
STATISTICS FOR GLB B7_part1, ROUTED LOCATION: A4
*****
GLB Input List:          GLB Output List:
I5      : V2            00      : !IRQ1
I6      : V1
I7      : V0
I10     : PIRQ
I11     : TIRQ

CELL EQUATIONS:
-----
!IRQ1 = (!PIRQ & !V2 & !V1 & V0) # (!TIRQ & !V2 & !V1 & V0);

*****
STATISTICS FOR GLB B7_part2, ROUTED LOCATION: B3
*****
GLB Input List:          GLB Output List:
I4      : TIRQ          02      : !IRQ3
I5      : PIRQ
I8      : V0
I9      : V1
I10     : V2

CELL EQUATIONS:
-----
!IRQ3 = (!PIRQ & !V2 & V1 & V0) # (!TIRQ & !V2 & V1 & V0);

I/O
cell  User      Input list:      Output list:      CELL EQUATIONS:
-----
I0    RW        PAD      : _RW      OUT      : READWRITE    XPIN I _RW LOCK 14 PULLUP;      IB11(READWRITE, _RW);
I1    DS0       PAD      : _DS0     OUT      : DDS0          XPIN I _DS0 LOCK 2 PULLUP;      IB11(DDS0, _DS0);
I2    A4        PAD      : _A4      OUT      : AA4           XPIN I _A4 LOCK 24 PULLUP;      IB11(AA4, _A4);
I3    AM0       PAD      : _AM0     OUT      : AAM0          XPIN I _AM0 LOCK 36 PULLUP;      IB11(AAM0, _AM0);
I00   A1        PAD      : _A1      OUT      : A1            XPIN IO _A1 LOCK 16 PULLUP;      IB11(A1, _A1);
I01   A2        PAD      : _A2      OUT      : A2            XPIN IO _A2 LOCK 19 PULLUP;      IB11(A2, _A2);
I02   A3        PAD      : _A3      OUT      : A3            XPIN IO _A3 LOCK 17 PULLUP;      IB11(A3, _A3);
I03   TQ        PAD      : _TIRQ     OUT      : TIRQ          XPIN IO _TIRQ LOCK 28 PULLUP;      IB11(TIRQ, _TIRQ);
I04   A5        PAD      : _A5      OUT      : A5            XPIN IO _A5 LOCK 18 PULLUP;      IB11(A5, _A5);
I05   PQ        PAD      : _PIRQ     OUT      : PIRQ          XPIN IO _PIRQ LOCK 27 PULLUP;      IB11(PIRQ, _PIRQ);
I06   AM1       PAD      : _AM1     OUT      : AM1           XPIN IO _AM1 LOCK 40 PULLUP;      IB11(AM1, _AM1);
I07   A6        PAD      : _A6      OUT      : A6            XPIN IO _A6 LOCK 20 PULLUP;      IB11(A6, _A6);
I08   AM3       PAD      : _AM3     OUT      : AM3           XPIN IO _AM3 LOCK 41 PULLUP;      IB11(AM3, _AM3);
I09   AM4       PAD      : _AM4     OUT      : AM4           XPIN IO _AM4 LOCK 38 PULLUP;      IB11(AM4, _AM4);
I010  IACK      PAD      : _IACK     OUT      : IACK          XPIN IO _IACK LOCK 43 PULLUP;      IB11(IACK, _IACK);
I011  AM5       PAD      : _AM5     OUT      : AM5           XPIN IO _AM5 LOCK 44 PULLUP;      IB11(AM5, _AM5);
I012  DS1       PAD      : _DS1     OUT      : DS1           XPIN IO _DS1 LOCK 3 PULLUP;      IB11(DS1, _DS1);
I013  BE        IMUX     : NBE      PAD      : BE           XPIN IO _BE LOCK 22 PULLUP;      OB11(_BE, NBE);
I014  AHI       PAD      : _ADRHI    OUT      : ADRHI         XPIN IO _ADRHI LOCK 4 PULLUP;      IB11(ADRHI, _ADRHI);
I015  I1N       PAD      : _IACKIN   OUT      : IACKIN        XPIN IO _IACKIN LOCK 42 PULLUP;      IB11(IACKIN, _IACKIN);
I016  8MHZ      IMUX     : MHZ8      PAD      : MHZ8         XPIN IO _MHZ8 LOCK 31 PULLUP;      OB11(_MHZ8, MHZ8);
I017  IACKOUT   IMUX     : IACKOUT   PAD      : _IACKOUT     XPIN IO _IACKOUT LOCK 39 PULLUP;      OB11(_IACKOUT, IACKOUT);
I018  PIACK     IMUX     : PIACK     PAD      : PIACK         XPIN IO _PIACK LOCK 29 PULLUP;      OB11(_PIACK, PIACK);
I019  TIACK     IMUX     : TIACK     PAD      : TIACK         XPIN IO _TIACK LOCK 30 PULLUP;      OB11(_TIACK, TIACK);
I020  DTACK     IMUX     : DTACK     PAD      : DTACK         XPIN IO _DTACK LOCK 7 PULLUP;      OB11(_DTACK, DTACK);
I021  PIT       IMUX     : CS_PIT    PAD      : _CS_PIT      XPIN IO _CS_PIT LOCK 32 PULLUP;      OB11(_CS_PIT, CS_PIT);
I022  CSD       IMUX     : !CS_D     PAD      : _CS_D        XPIN IO _CS_D LOCK 37 PULLUP;      OB21(_CS_D, !CS_D);
I023  CSC       IMUX     : !CS_C     PAD      : _CS_C        XPIN IO _CS_C LOCK 25 PULLUP;      OB21(_CS_C, !CS_C);
I024  CSB       IMUX     : !CS_B     PAD      : _CS_B        XPIN IO _CS_B LOCK 26 PULLUP;      OB21(_CS_B, !CS_B);
I025  CSA       IMUX     : !CS_A     PAD      : _CS_A        XPIN IO _CS_A LOCK 21 PULLUP;      OB21(_CS_A, !CS_A);
I026  Q6        IMUX     : !IRQ6     PAD      : _IRQ6        XPIN IO _IRQ6 LOCK 8 PULLUP;      OB21(_IRQ6, !IRQ6);
I027  Q5        IMUX     : !IRQ5     PAD      : _IRQ5        XPIN IO _IRQ5 LOCK 6 PULLUP;      OB21(_IRQ5, !IRQ5);
I028  Q4        IMUX     : !IRQ4     PAD      : _IRQ4        XPIN IO _IRQ4 LOCK 5 PULLUP;      OB21(_IRQ4, !IRQ4);
I029  Q3        IMUX     : !IRQ3     PAD      : _IRQ3        XPIN IO _IRQ3 LOCK 9 PULLUP;      OB21(_IRQ3, !IRQ3);
I030  Q2        IMUX     : !IRQ2     PAD      : _IRQ2        XPIN IO _IRQ2 LOCK 10 PULLUP;      OB21(_IRQ2, !IRQ2);
I031  Q1        IMUX     : !IRQ1     PAD      : _IRQ1        XPIN IO _IRQ1 LOCK 15 PULLUP;      OB21(_IRQ1, !IRQ1);
RESET  SYSGEN   PAD      : !XRESET    OUT      : !RESET       XPIN RST !XRESET LOCK 35;      IB11(!RESET, !XRESET);
Y0     CLK      PAD      : _SYSCLK   OUT      : SYSCLK       XPIN CLK _SYSCLK LOCK 11 PULLUP;      IB11(SYSCLK, _SYSCLK);

```

19.14.2 Sonar-kontrollogik

Programlistning

OBS! listningen er håndediteret for at fjerne overflødig information, og fremme overskueligheden.

Mon Sep 01 17:48:04 1997
sonar.rpt generated using Lattice pDS Version 2.61

External I/O Pin Report

Part: ispLSI1016-80LJ44
 Design Name: Quad sonar controller
 Design Revision: 0.1
 Author: Anders Stengaard
 Project Name: Heimdal
 Description: Part of a VME-bus interface for POLAROID's 50KHz sonar
 Verify Status: Complete
 Route Status: Complete
 Data Used: Post-Route

Pin Number	Pad Name	Pin Type	Pullup	Pin Number	Pad Name	Pin Type	Pullup
1	GND	Gnd		23	GND	Gnd	
3	_INIT_A	Output	No	25	_INIT	Input	Yes
4	_N_INIT	Output	No	27	_BLNK_D	Output	No
5	_N_CS_B	Input	Yes	28	_INIT_D	Output	No
6	_LATCH_B	Output	No	29	_BLNK_C	Output	No
7	_LATCH_A	Output	No	30	_ECHO_D	Input	Yes
8	_N_CS_A	Input	Yes	31	_INIT_C	Output	No
9	_N_CS_D	Input	Yes	32	_ECHO_C	Input	Yes
10	_N_CS_C	Input	Yes	34	VCC	Vcc	
11	_SYNC	Input	Yes	35	XRESET	Input	No
12	VCC	Vcc		37	_ECHO_B	Input	Yes
15	_LATCH_D	Output	No	38	_BLNK_B	Output	No
16	_LATCH_C	Output	No	39	_INIT_B	Output	No
17	_CLK	Input	Yes	40	_BLNK_A	Output	No
18	_INITMASK_B	Input	No	41	_ECHO_A	Input	Yes
19	_INITMASK_A	Input	No	42	_TICK	Input	Yes
20	_INITMASK_C	Input	No	43	_TOUT	Input	Yes
21	_INITMASK_D	Input	No	44	_ECLK	Output	No

 STATISTICS FOR GLB A0, ROUTED LOCATION: B1

 GLB Input List: GLB Output List:
 I4 : BLNK_A O0 : LATCH_A
 I6 : !CLR_A
 I8 : TIMER
 I9 : TOUT
 I11 : ECHO_A
 I16 : LATCH_A
 CLK0 : SYNC
 RESET : !RESET

CELL EQUATIONS:

 LATCH_A.CLK = SYNC;
 LATCH_A.RE = CLR_A;
 LATCH_A.D = (TIMER & !BLNK_A & ECHO_A) # (TOUT) # (LATCH_A.Q);

 STATISTICS FOR GLB A1, ROUTED LOCATION: B0

 GLB Input List: GLB Output List:
 I0 : N_CS_C O0 : INIT_A
 I3 : !CLR_B O1 : !CLR_C
 I7 : INIT O2 : INIT_B
 I8 : TIMER O3 : LATCH_B
 I9 : TOUT
 I11 : INITMASK_A
 I12 : INITMASK_B
 I14 : BLNK_B
 I15 : ECHO_B
 I17 : LATCH_B
 CLK0 : SYNC
 RESET : !RESET

CELL EQUATIONS:

 LATCH_B.CLK = SYNC;
 LATCH_B.RE = CLR_B;
 LATCH_B.D = (TIMER & !BLNK_B & ECHO_B) # (TOUT) # (LATCH_B.Q);
 INIT_B = INITMASK_B & INIT;
 INIT_A = INITMASK_A & INIT;
 !CLR_C = N_CS_C & INIT;

 STATISTICS FOR GLB A2, ROUTED LOCATION: A2

 GLB Input List: GLB Output List:
 I5 : INITMASK_C O2 : INIT_C
 I6 : INITMASK_D O3 : INIT_D
 I8 : INIT

CELL EQUATIONS:

 INIT_D = INITMASK_D & INIT;
 INIT_C = INITMASK_C & INIT;

 STATISTICS FOR GLB A3, ROUTED LOCATION: A6

 GLB Input List: GLB Output List:
 I8 : INIT O1 : !CLR_A
 I13 : N_CS_A

CELL EQUATIONS:

 !CLR_A = N_CS_A & INIT;

 STATISTICS FOR GLB A4, ROUTED LOCATION: A3

```

GLB Input List:          GLB Output List:
I0   : LATCH_D           O0   : DQ0
I5   : TICK              O1   : DQ1
I8   : INIT              O2   : BLNK_D
I11  : INIT_D           O3   : DQ2
I12  : DQ0
I13  : DQ1
I15  : DQ2
RESET : !RESET

CELL EQUATIONS:
-----
DQ0.PTCLK = TICK;      DQ0.RE = !INIT;
DQ1.PTCLK = TICK;      DQ1.RE = !INIT;
DQ2.PTCLK = TICK;      DQ2.RE = !INIT;
BLNK_D.PTCLK = TICK;    BLNK_D.RE = !INIT;

DQ0.D = (DQ2.Q & !DQ0.Q) # (DQ2.Q & DQ1.Q & LATCH_D) # (!DQ0.Q & LATCH_D) # (DQ1.Q & !DQ0.Q);
DQ1.D = (DQ2.Q & DQ0.Q & LATCH_D) # (DQ1.Q & !DQ0.Q) # (!DQ1.Q & DQ0.Q);
DQ2.D = !((DQ2.Q & DQ1.Q & DQ0.Q & !LATCH_D) # (!DQ2.Q & !DQ0.Q) # (!DQ2.Q & !DQ1.Q));
BLNK_D.D = !((!DQ2.Q & !DQ1.Q & !DQ0.Q & !LATCH_D) # (DQ2.Q & DQ1.Q & DQ0.Q & !LATCH_D) # (!INIT_D));

*****
STATISTICS FOR GLB A5, ROUTED LOCATION: A4
*****
GLB Input List:          GLB Output List:
I1   : CQ2              O0   : BLNK_C
I2   : CQ1              O1   : CQ2
I3   : CQ0              O2   : CQ1
I5   : TICK             O3   : CQ0
I8   : INIT
I10  : INIT_C
I13  : LATCH_C
RESET : !RESET

CELL EQUATIONS:
-----
CQ0.PTCLK = TICK;      CQ0.RE = !INIT;
CQ1.PTCLK = TICK;      CQ1.RE = !INIT;
CQ2.PTCLK = TICK;      CQ2.RE = !INIT;
BLNK_C.PTCLK = TICK;    BLNK_C.RE = !INIT;

CQ0.D = (CQ2.Q & !CQ0.Q) # (CQ2.Q & CQ1.Q & LATCH_C) # (!CQ0.Q & LATCH_C) # (CQ1.Q & !CQ0.Q);
CQ1.D = (CQ2.Q & CQ0.Q & LATCH_C) # (CQ1.Q & !CQ0.Q) # (!CQ1.Q & CQ0.Q);
CQ2.D = !((CQ2.Q & CQ1.Q & CQ0.Q & !LATCH_C) # (!CQ2.Q & !CQ0.Q) # (!CQ2.Q & !CQ1.Q));
BLNK_C.D = !((!CQ2.Q & !CQ1.Q & !CQ0.Q & !LATCH_C) # (CQ2.Q & CQ1.Q & CQ0.Q & !LATCH_C) # (!INIT_C));

*****
STATISTICS FOR GLB A6, ROUTED LOCATION: B4
*****
GLB Input List:          GLB Output List:
I7   : INIT            O0   : BQ0
I10  : TICK            O1   : BLNK_B
I12  : LATCH_B         O2   : BQ1
I13  : INIT_B          O3   : BQ2
I15  : BQ0
I16  : BQ1
I17  : BQ2
RESET : !RESET

CELL EQUATIONS:
-----
BQ0.PTCLK = TICK;      BQ0.RE = !INIT;
BQ1.PTCLK = TICK;      BQ1.RE = !INIT;
BQ2.PTCLK = TICK;      BQ2.RE = !INIT;
BLNK_B.PTCLK = TICK;    BLNK_B.RE = !INIT;

BQ0.D = (BQ2.Q & !BQ0.Q) # (BQ2.Q & BQ1.Q & LATCH_B) # (!BQ0.Q & LATCH_B) # (BQ1.Q & !BQ0.Q);
BQ1.D = (BQ2.Q & BQ0.Q & LATCH_B) # (BQ1.Q & !BQ0.Q) # (!BQ1.Q & BQ0.Q);
BQ2.D = !((BQ2.Q & BQ1.Q & BQ0.Q & !LATCH_B) # (!BQ2.Q & !BQ0.Q) # (!BQ2.Q & !BQ1.Q));
BLNK_B.D = !((!BQ2.Q & !BQ1.Q & !BQ0.Q & !LATCH_B) # (BQ2.Q & BQ1.Q & BQ0.Q & !LATCH_B) # (!INIT_B));

*****
STATISTICS FOR GLB A7, ROUTED LOCATION: B6
*****
GLB Input List:          GLB Output List:
I5   : AQ2             O0   : AQ0
I6   : AQ1             O1   : AQ1
I7   : INIT            O2   : AQ2
I10  : TICK            O3   : BLNK_A
I11  : LATCH_A
I15  : INIT_A
I16  : AQ0
RESET : !RESET

CELL EQUATIONS:
-----
AQ0.PTCLK = TICK;      AQ0.RE = !INIT;
AQ1.PTCLK = TICK;      AQ1.RE = !INIT;
AQ2.PTCLK = TICK;      AQ2.RE = !INIT;
BLNK_A.PTCLK = TICK;    BLNK_A.RE = !INIT;

AQ0.D = (AQ2.Q & !AQ0.Q) # (AQ2.Q & AQ1.Q & LATCH_A) # (!AQ0.Q & LATCH_A) # (AQ1.Q & !AQ0.Q);
AQ1.D = (AQ2.Q & AQ0.Q & LATCH_A) # (AQ1.Q & !AQ0.Q) # (!AQ1.Q & AQ0.Q);
AQ2.D = !((AQ2.Q & AQ1.Q & AQ0.Q & !LATCH_A) # (!AQ2.Q & !AQ0.Q) # (!AQ2.Q & !AQ1.Q));
BLNK_A.D = !((!AQ2.Q & !AQ1.Q & !AQ0.Q & !LATCH_A) # (AQ2.Q & AQ1.Q & AQ0.Q & !LATCH_A) # (!INIT_A));

*****
STATISTICS FOR GLB A0_part1, ROUTED LOCATION: B2
*****
GLB Input List:          GLB Output List:
I1   : N_CS_D          O1   : N_INIT
I2   : LATCH_C          O2   : !CLR_D
I7   : INIT            O3   : ECLK
I9   : TOUT
I11  : LATCH_A
I12  : LATCH_B

```

```

I13 : CLK
I15 : LATCH_D

CELL EQUATIONS:
-----
ECLK = (LATCH_A & LATCH_D & LATCH_C & LATCH_B & TOUT) # (CLK);
!CLR_D = N_CS_D & INIT;
N_INIT = !INIT;

*****
STATISTICS FOR GLB A2_part1, ROUTED LOCATION: A7
*****
GLB Input List:          GLB Output List:
I0 : BLNK_C              00 : !CLR_B
I1 : !CLR_C              01 : LATCH_C
I6 : TOUT
I7 : TIMER
I8 : INIT
I10 : N_CS_B
I13 : LATCH_C
I15 : ECHO_C
CLK0 : SYNC
RESET : !RESET

CELL EQUATIONS:
-----
LATCH_C.CLK = SYNC;
LATCH_C.RE = CLR_C;

LATCH_C.D = (TIMER & !BLNK_C & ECHO_C) # (TOUT) # (LATCH_C.Q);
!CLR_B = N_CS_B & INIT;

*****
STATISTICS FOR GLB A3_part1, ROUTED LOCATION: A0
*****
GLB Input List:          GLB Output List:
I0 : LATCH_D              00 : LATCH_D
I6 : TOUT
I7 : TIMER
I10 : !CLR_D
I13 : ECHO_D
I14 : BLNK_D
CLK0 : SYNC
RESET : !RESET

CELL EQUATIONS:
-----
LATCH_D.CLK = SYNC;
LATCH_D.RE = CLR_D;
LATCH_D.D = (TIMER & !BLNK_D & ECHO_D) # (TOUT) # (LATCH_D.Q);

*****
STATISTICS FOR GLB B6, ROUTED LOCATION: A1
*****
GLB Input List:          GLB Output List:
I5 : TICK                 03 : TIMER
I7 : TIMER
I8 : INIT
I12 : T3
I13 : T2
I14 : T1
I15 : T0
RESET : !RESET

CELL EQUATIONS:
-----
TIMER.PTCLK = TICK;
TIMER.RE = !INIT;
TIMER.D = (T3 & T2 & T1 & T0) # (TIMER.Q);

*****
STATISTICS FOR GLB B7, ROUTED LOCATION: B3
*****
GLB Input List:          GLB Output List:
I0 : T0                   00 : T3
I1 : T1                   01 : T2
I2 : T2                   02 : T1
I3 : T3                   03 : T0
I7 : INIT
I10 : TICK
RESET : !RESET

CELL EQUATIONS:
-----
T0.PTCLK = TICK;          T0.RE = !INIT;
T1.PTCLK = TICK;          T1.RE = !INIT;
T2.PTCLK = TICK;          T2.RE = !INIT;
T3.PTCLK = TICK;          T3.RE = !INIT;

T0.D = !T0.Q;
T1.D = (T1.Q) $$ (T0.Q);
T2.D = (T2.Q) $$ (T1.Q & T0.Q);
T3.D = (T3.Q) $$ (T2.Q & T1.Q & T0.Q);

*****
STATISTICS FOR I/O CELLS
*****
I/O User
cell name Input List:      Output list: Cell equations
-----
I00      PAD : _CLK         OUT : CLK          XPIN IO _CLK LOCK 17 PULLUP;  IB11(CLK, _CLK);
I01      PAD : _TOUT        OUT : TOUT         XPIN IO _TOUT LOCK 43 PULLUP; IB11(TOUT, _TOUT);
I02      PAD : _INIT        OUT : INIT         XPIN IO _INIT LOCK 25 PULLUP; IB11(INIT, _INIT);
I03      IMUX : ECLK        PAD : _ECLK        XPIN IO _ECLK LOCK 44;      OB11(_ECLK, ECLK);
I04      IMUX : N_INIT      PAD : _N_INIT      XPIN IO _N_INIT LOCK 4;     OB11(_N_INIT, N_INIT);
I05      PAD : _ECHO_A      OUT : ECHO_A       XPIN IO _ECHO_A LOCK 41 PULLUP; IB11(ECHO_A, _ECHO_A);
I06      PAD : _ECHO_B      OUT : ECHO_B       XPIN IO _ECHO_B LOCK 37 PULLUP; IB11(ECHO_B, _ECHO_B);

```



```

I07      PAD : _ECHO_C      OUT : ECHO_C      XPIN IO _ECHO_C LOCK 32 PULLUP; IB11(ECHO_C, _ECHO_C);
I08      PAD : _ECHO_D      OUT : ECHO_D      XPIN IO _ECHO_D LOCK 30 PULLUP; IB11(ECHO_D, _ECHO_D);
I09      IMUX : LATCH_A      PAD : _LATCH_A      XPIN IO _LATCH_A LOCK 7; OB11(_LATCH_A, LATCH_A);
I010     IMUX : LATCH_B      PAD : _LATCH_B      XPIN IO _LATCH_B LOCK 6; OB11(_LATCH_B, LATCH_B);
I011     IMUX : LATCH_C      PAD : _LATCH_C      XPIN IO _LATCH_C LOCK 16; OB11(_LATCH_C, LATCH_C);
I012     IMUX : LATCH_D      PAD : _LATCH_D      XPIN IO _LATCH_D LOCK 15; OB11(_LATCH_D, LATCH_D);
I013     PAD : _N_CS_A      OUT : N_CS_A      XPIN IO _N_CS_A LOCK 8 PULLUP; IB11(N_CS_A, _N_CS_A);
I014     PAD : _N_CS_B      OUT : N_CS_B      XPIN IO _N_CS_B LOCK 5 PULLUP; IB11(N_CS_B, _N_CS_B);
I015     PAD : _N_CS_C      OUT : N_CS_C      XPIN IO _N_CS_C LOCK 10 PULLUP; IB11(N_CS_C, _N_CS_C);
I018     IMUX : INIT_D      PAD : _INIT_D      XPIN IO _INIT_D LOCK 28; OB11(_INIT_D, INIT_D);
I019     IMUX : INIT_C      PAD : _INIT_C      XPIN IO _INIT_C LOCK 31; OB11(_INIT_C, INIT_C);
I020     IMUX : INIT_B      PAD : _INIT_B      XPIN IO _INIT_B LOCK 39; OB11(_INIT_B, INIT_B);
I021     IMUX : INIT_A      PAD : _INIT_A      XPIN IO _INIT_A LOCK 3; OB11(_INIT_A, INIT_A);
I022     PAD : _INITMASK_D  OUT : INITMASK_D  XPIN IO _INITMASK_D LOCK 21; IB11(INITMASK_D, _INITMASK_D);
I023     PAD : _INITMASK_C  OUT : INITMASK_C  XPIN IO _INITMASK_C LOCK 20; IB11(INITMASK_C, _INITMASK_C);
I024     PAD : _INITMASK_B  OUT : INITMASK_B  XPIN IO _INITMASK_B LOCK 18; IB11(INITMASK_B, _INITMASK_B);
I025     PAD : _INITMASK_A  OUT : INITMASK_A  XPIN IO _INITMASK_A LOCK 19; IB11(INITMASK_A, _INITMASK_A);
I026     PAD : _TICK        OUT : TICK        XPIN IO _TICK LOCK 42 PULLUP; IB11(TICK, _TICK);
I027     IMUX : BLNK_D      PAD : _BLNK_D      XPIN IO _BLNK_D LOCK 27; OB11(_BLNK_D, BLNK_D);
I028     IMUX : BLNK_C      PAD : _BLNK_C      XPIN IO _BLNK_C LOCK 29; OB11(_BLNK_C, BLNK_C);
I029     IMUX : BLNK_B      PAD : _BLNK_B      XPIN IO _BLNK_B LOCK 38; OB11(_BLNK_B, BLNK_B);
I030     IMUX : BLNK_A      PAD : _BLNK_A      XPIN IO _BLNK_A LOCK 40; OB11(_BLNK_A, BLNK_A);
I031     PAD : _N_CS_D      OUT : N_CS_D      XPIN IO _N_CS_D LOCK 9 PULLUP; IB11(N_CS_D, _N_CS_D);
RESET    SYSGEN PAD : !XRESET OUT : !RESET    XPIN RST !XRESET LOCK 35; IB11(!RESET, !XRESET);
Y0        SYNC PAD : _SYNC    OUT : SYNC      XPIN CLK _SYNC LOCK 11 PULLUP; IB11(SYNC, _SYNC);

```

19.15 Kildetekster til device driver

I dette afsnit gengives kildeteksterne til device descriptoren **hmd**, device driveren **hmddrv** og testprogrammet **test**, samt en makefil, til at bygge alle tre dele med.

19.15.1 makefile

Denne make fil er brugt til at bygge alle komponenter af software interfacet til Heimdal. Den har været anvendt af bruger: **heimdal**, med hjemmekatalog i /dd/USR/HEIMDAL. Den skal bruge underkatalogerne: **SOURCE**, **RELS**, og **EXE**, og ligger selv i **SOURCE** sammen med kildeteksterne.

```

MYDIR = /dd/USR/HEIMDAL

ODIR = $(MYDIR)/EXE
SDIR = $(MYDIR)/SOURCE
RDIR = $(MYDIR)/RELS

RFLAGS = -Q

LDIR = /dd/LIB

HMDDRV_OBJS = $(LDIR)/sbfdrv1.l $(LDIR)/sbfdrv1.l $(RDIR)/hmddefs.r $(RDIR)/hmddrv.r

#
# MAKE all
#
all:    hmddrv hmd
        touch make.date

#
# driver
#

hmddrv:    $(LDIR)/sbfdrv1.l hmddrv.r hmddefs.r
           168 $(HMDDRV_OBJS) -l=$(LDIR)/sys.1 -O=$(ODIR)/hmddrv

hmddrv.r:    hmddrv.a
           r68 hmddrv.a -o=$(RDIR)/hmddrv.r $(RFLAGS)

#
# device descriptor
#

hmd:        hmd.r
           168 $(RDIR)/hmd.r -O=$(ODIR)/hmd

hmd.r:      hmd.a
           r68 hmd.a -o=$(RDIR)/hmd.r $(RFLAGS)

#
# Definitions

```

```
#
hmddefs.r: hmddefs.a
           r68 hmddefs.a -o=$(RDIR)/hmddefs.r $(RFLAGS)

#
# Static storage
#

/dd/LIB/sbdrv1.l: /dd/DEFS/makefile
                (chd /dd/DEFS; make ../LIB/sbdrv1.l)
```

19.15.2 hmd.a

Kildeteksten til device descriptoren. Den eneste eksterne fil der inkluderes er `/dd/DEFS/oskdefs.d`, der definerer nogle modul relevante konstanter der skal være kendt på assemble niveau.

```
*****
* Edition History
*
*      date      comments      by
*  --  -
* 01    1 dec 1997 Created      Anders
* 01    18 dec 1997 Completed   Anders

        nam hmd
        ttl heimdal device descriptor module

Edition      equ 1      current edition number

TypeLang set (Devic<<8)+0
Attr_Rev set (ReEnt<<8)+0

psect hmd,TypeLang,Attr_Rev,Edition,0,0

dc.l $873e0000      Heimdals BASE address in the CPU's address space
dc.b 200            Interrupt vector to be used (must be multiple of 4)
dc.b 4              Interrupt level. (1..6)
dc.b 1              OS-9 IRQ polling priority
dc.b Updat_+PUpdat_+Share_ Device mode capabilities (R/W, non share)
dc.w FileMgr        file manager name offset
dc.w DevDrv         device driver name offset
dc.w DevCon         Offset to driver configuration block (Heimdal parameters)
dc.w 0,0,0,0        reserved
dc.w OptSize        option byte count

Options
* This is the beginning of the options field. The field is divided into two parts.
* The required part contains fields that are required of all SBF descriptors.
* The Heimdal part contains parameters used by the device driver for Heimdal (hmddrv)
*
* This whole options field is located from address $48 in descriptor module.
* When a path to this device is created, these values are copied to the path descriptor
* from address $80
*

*****
* Required part:
* The names and >>path descriptor offsets<< of the required parameters are
* contained in the library: /dd/lib/sbf.l
*
*      val name      comment
*  ---
dc.b 3      PD_DTP      Device type (3=SBF device)
dc.b 0      PD_TDrv     Drive number (must be zero for this driver)
dc.b 0      PD_SBF      (don't know what this field is... Zero in /dd/defs/vadidesc.d)
dc.b 0      PD_NumBlk   Number of blocks (must be zero for this driver)
dc.l 0      PD_BlzSiz   Block size
dc.w 0      PD_Prior    driver process priority
dc.b 0      PD_Flags    SBF path flags (This field is also known as PD_SBFflags in later OS-9 versions)

* ----> The following fields are defined in OS-9 3.0, but not 2.0 I don't know
* if they are used by 2.2 (this version), but have included them,
* just in case.
```

```

dc.b    0    PD_DrivFlag This field can be used by the driver
dc.w    0    PD_DMAMode
dc.b    0    PD_ScsiID
dc.b    0    PD_ScsiLUN
dc.l    0    PD_ScsiOpts

DevCon
*****
HEIMDAL_PARAMETERS:
*
* Parameters for Heimdal device driver (hmddrv)
* The names and >>path descriptor offsets<< of the required parameters are
* contained in the object file: hmdopts.r, which is made from hmdopts.a
*
*value  name      comment
*-----
dc.l    1000     PD_BINH      BINH time = 1ms
dc.l    64000    PD_TIME      Time to listen after BINH=64ms
dc.b    $81,$81 PD_SENSOR    List that tells driver which sensors to use in the 4 groups.
dc.b    $81,$81          Bit 7 tells driver to cycle through the 4 sensors in the group.

OptSize equ *-Options

*****
* Strings are kept here

FileMgr dc.b "SBF",0    Sequential Block File manager
DevDrv  dc.b "hmddrv",0 Device driver

        use <oskdefs.d>
ends

```

19.15.3 hmddefs.a

Kildetekst til definition af diverse hard- og software relevante værdier, af betydning for device driveren. Heri er defineret offset adresser til driverens variable, og til Heimdals registre. Denne kildetekst assembleres til hmddefs.r, der senere linkes med hmddrv.r

```

*****
*** hmddefs.a          ***
*** Definitions for Heimdal device driver ***
*****
* Edition history:
* date            action            by
*-----
* Nov 11, 1997    Created            by    Anders

        psect hmdconst,0,0,0,0,0
*****
*Below are the offsets for Heimdal parameters
*in the device descriptor module
*****
        org      0

HMD_BINH:  do.l    1      The BINH time (measured in us) is a 32 bit integer
HMD_TIME:  do.l    1      The active 'listen' time (in us) is a 32 bit integer
HMD_SENSOR: do.b    4      The address of the active sensor in each of the 4 groups

*****
*Below are the definitions for Heimdal's memory map!
*****
* name            type      offset  comment
*-----
        org      0

* ----> This is the address of a bogus register in the VME-SLAVE controller logic (U4)
* The value read/written to this register doesn't matter, only the action
* of reading/writing. The interrupt level is controlled by reading/writing to the address.

HMD_ICTRL: do.b    1      $00      for IRQ level control

* ----> This section defines registers in the MC 68230 PIT (U20)

```

```

HMD_PGCR:  do.b  2  $01  port general control register
HMD_PSRR:  do.b  2  $03  port service request register
HMD_PADDR: do.b  2  $05  port A data direction register
HMD_PBDDR: do.b  2  $07  port B data direction register
HMD_PCDDR: do.b  2  $09  port C data direction register
HMD_PIVR:  do.b  2  $0B  port interrupt vector register
HMD_PACR:  do.b  2  $0D  port A control register
HMD_PBCR:  do.b  2  $0F  port B control register
HMD_PADR:  do.b  2  $11  port A data register
HMD_PBDR:  do.b  2  $13  port B data register
HMD_PAAR:  do.b  2  $15  port A alternate register
HMD_PBAR:  do.b  2  $17  port B alternate register
HMD_PCDR:  do.b  2  $19  port C data register
HMD_PSR:   do.b  6  $1B  port status register

HMD_TCR:   do.b  2  $21  timer control register
HMD_TIVR:  do.b  2  $23  timer interrupt vector register
HMD_CPR:   do.b  2  $25  counter preload register offset, for movep.l
HMD_CPRH:  do.b  2  $27  counter preload register (high)
HMD_CPRM:  do.b  2  $29  counter preload register (mid)
HMD_CPRL:  do.b  2  $2B  counter preload register (low)
HMD_CR:    do.b  2  $2D  counter register offset, for movep.l
HMD_CRH:   do.b  2  $2F  counter register (high)
HMD_CRM:   do.b  2  $31  counter register (mid)
HMD_CRL:   do.b  2  $33  counter register (low)
HMD_TSR:   do.b  11 $35  timer status register

```

* ----> Addresses for the echo registers (located in U7..U14)

```

HMD_REG1:  do.w  8  $40  echo register for group 1
HMD_REG2:  do.w  8  $50  echo register for group 2
HMD_REG3:  do.w  8  $60  echo register for group 3
HMD_REG4:  do.w  8  $70  echo register for group 4

```

ends

19.15.4 hmddrv.a

Kildetekst til selve device driveren. Denne fil indeholder al koden til driveren. Den assembleres, og linkes sammen med hmddefs.r, samt systembibliotekerne /dd/LIB/sbf.l og /dd/LIB/sbfdrv1.l Det er vigtigt at der linkes med /dd/LIB/sbfdrv1.l før hmddrv.r, for at linke variabelområderne (vsect) i rigtig rækkefølge.)

```

* test device driver for Heimdal sonar interface
Typ_Lang    set (Drivr<<8)+Objct
Att_Revs    set ((ReEnt+SupStat)<<8)+0
*
Edition set 1

    psect    hmddrv,Typ_Lang,Att_Revs,Edition,0,EntryTable
    use <oskdefs.d>

* Routine offset table
EntryTable  dc.w    Init
            dc.w    Read
            dc.w    Write
            dc.w    GetStat
            dc.w    SetStat
            dc.w    Term
            dc.w    0

* Static storage definitions
    vsect
V_MAXECHO   ds.w    1    Maximum number of echos to receive
V_ECHOS     ds.b    4    How many echos have been received from group 0,1,2, and 3
V_DAT       ds.l    4    Points to where the next echos will be stored

V_BINH      ds.l    1    Local copy of the BINH option in dev. descriptor
V_TIME      ds.l    1    Local copy of the TIME option in dev. dexriptor
V_SENSOR    ds.b    4    Local copy of the SENSOR list in dev. descriptor

ends

*****
Init

```

```

*
* Initialize device
*
* Input:
*   (a1)    address of the device descriptor module
*   (a2)    address of the device static storage
*   (a6)    System global data pointer
*
* Output: none
*
* Error:
*   cc      = carry bit set
*   dl.w    = error code
*
* Initialisation of Heimdal, and this device driver
*
*****

* ----> Initialize device static storage

        move.b  #1,SBF_NDrv(a2)    This device can handle one "drive"
        move.l  #$ff,SBF_DPr(a2)  This prevents SBF from creating a process
*                                     during Create/Open

* ----> Copy parameters from dev. descriptor to static storage
        move.w  M$DevCon(a1),d1    Get offset to device configuration data, within module
        move.l  HMD_BINH(a1,d1.w),d0
        asr.l   #2,d0              divide by 4 to get time in 4us intervals
        move.l  d0,V_BINH(a2)
        move.l  HMD_TIME(a1,d1.w),d0
        asr.l   #2,d0              divide by 4 to get time in 4us intervals
        move.l  d0,V_TIME(a2)
        move.l  HMD_SENSOR(a1,d1.w),V_SENSOR(a2)

* ----> Set up interrupt service routines (Heimdals MC68230 uses 4 consecutive
*      interrupt vectors, which makes it necessary to install 4 service routines)

        move.b  M$Vector(a1),d0    Get interrupt vector from descriptor module
        btst    #0,d0              Test if bit 0 in vector is 0
        bne     Err_Param           Error if not
        btst    #1,d0              Test if bit 1 in vector is 0
        bne     Err_Param           Error if not
        move.b  M$Prior(a1),d1     Get interrupt polling priority from descriptor module
        lea     PIRQ_H1(pcr),a0    Get address of IRQ handler for H1
        os9     F$IRQ              Install IRQ handler
        bcs     Err_exit           If error, exit (errno will be in d1)
        lea     PIRQ_H2(pcr),a0    Get address of IRQ handler for H2
        addq.b  #1,d0              H2 uses next vector
        os9     F$IRQ              Install IRQ handler
        bcc.s   Init_4             if OK, jump

Init_e1    movea.l #0,a0             I want to remove-
        subq.b  #1,d0              previously installed-
        os9     F$IRQ              interrupt handler,-
Init_e0    ori  #Carry,ccr          signal error,-
        rts                      and exit.

Init_4     lea     PIRQ_H3(pcr),a0  Get address of IRQ handler for H3
        addq.b  #1,d0              H2 uses next vector
        os9     F$IRQ              Install IRQ handler
        bcc.s   Init_5             if OK, jump

Init_e2     movea.l #0,a0             I want to remove-
        subq.b  #1,d0              previously installed-
        os9     F$IRQ              interrupt handler,-
        bra.s   Init_e1            as well as the one before that.

Init_5     lea     PIRQ_H4(pcr),a0  Get address of IRQ handler for H4
        addq.b  #1,d0              H2 uses next vector
        os9     F$IRQ              Install IRQ handler
        bcc.s   Init_3             If all went well, jump!

Init_e3     movea.l #0,a0             I want to remove-
        subq.b  #1,d0              previously installed-
        os9     F$IRQ              interrupt handler,-
        bra.s   Init_e2            as well as the one before that.

* ----> Initialize Heimdals MC68230 PI/T

```

```

Init_3      movea.l V_PORT(a2),a3      Get port address

            and.b    #11111100,d0      Restore d0, to contain orgiginal vector.
            move.b    #3F,HMD_PGCR(a3)
            move.b    #18,HMD_PSR(a3)
            move.b    d0,HMD_PIVR(a3)   Interrupt vector for port
            move.b    d0,HMD_TIVR(a3)   Interrupt vector for timer
            move.b    #86,HMD_PACR(a3)
            move.b    #86,HMD_PBCR(a3)
            move.b    #00,HMD_PADR(a3)
            move.b    #00,HMD_PBDR(a3)
            move.b    #00,HMD_PCDR(a3)
            move.b    #FF,HMD_PADDR(a3)
            move.b    #FF,HMD_PBDDR(a3)
            move.b    #03,HMD_PCDDR(a3)

* ----> Initialize Heimdals interrupt controller

            move.b    d0,HMD_ICTRL(a3)   Reset IRQ level (any byte write will do)
            move.b    M$IRQLvl(a1),d1    Get interrupt level
Init_1      tst.b     d1                 Is it 0 ?
            beq.s     Init_2             Get out of loop!
            move.b    HMD_ICTRL(a3),d0    Increment IRQ level (any byte read will do)
            subq.b    #1,d1
            bra.s     Init_1             Repeat

Init_2      tst.w     d0                 clear carry (no error)
            rts

*****
* IRQ service routine
*
*****

* ----> This part sorts out, if Heimdal created the interrupt

PIRQ_H4     move.l    #3,d0              We get here by an echo on group 4
            bra.s     PIRQ_0
PIRQ_H3     move.l    #2,d0              We get here by an echo on group 3
            bra.s     PIRQ_0
PIRQ_H2     move.l    #1,d0              We get here by an echo on group 2
            bra.s     PIRQ_0
PIRQ_H1     *
            *                          We get here by an echo on group 1, or
            *                          By a timer interrupt

            movea.l    V_PORT(a2),a3      Get the port address from static storage
            btst.b    #0,HMD_TSR(a3)      Test if the timer caused the interrupt
            bne.s     TIRQ_0              Jump if it did
            move.l    #0,d0              We get here by an echo on group 1

PIRQ_0      movea.l    V_PORT(a2),a3
            btst.b    d0,HMD_PSR(a3)      Test if IRQ was caused by 68230 handshake
            bne.s     PIRQ_1              Jump if it was
            ori    #Carry,ccr             Set carry if it wasn't
            rts                          and get out of here

* ----> This part handles interrupts from Heimdal, caused by echos

PIRQ_1      move.l    d2,-(SP)            store d2 on stack
            move.l    a1,-(SP)            store a1 on stack

            move.b    #1,d1
            asl.b     d0,d1
            move.b    d1,HMD_PSR(a3)      Reset the PSR Hx flip-flop in question (clears interrupt condition)
            move.w    V_MAXECHO(a2),d1    What is the maximum amount of echos?
            move.l    a2,a1
            add.l     d0,a1
            move.l    a1,d2              remember the address
            cmp.b     V_ECHOS(a1),d1      Have we got out quota ?
            ble.s     PIRQ_2              Skip to exit if we have
            addq.b    #1,V_ECHOS(a1)      increment echo counter for this group
            lsl.w     #2,d0              multiply group number (0..3) by 4
            move.l    a2,a1
            add.l     d0,a1
            movea.l    V_DAT(a1),a0       get address of data field to write to
            lsl.w     #2,d0              multiply group number by further 2
            move.w    HMD_REG1(a3,d0.w),(a0) read sensor
            addq.l    #8,V_DAT(a1)        Increment addresss to ptr

```

```

        move.l    d2,a1
        cmp.b    V_ECHOS(a1),d1      Test if we already have enough echos
        ble.s    PIRQ_2              Skip to exit if there was
        move.w    #0,8(a0)           write a zero to next entry

PIRQ_2    move.l    (SP)+,a1          restore a1 from stack
        move.l    (SP)+,d2          restore a2 from stack
        tst      d0                  Clear carry
        rts

* ----> This part handles interrupts from Heimdal, caused by the timer

TIRQ_0    move.b    #%10110000,HMD_TCR(a3)  Stop timer
        move.b    #1,HMD_TSR(a3)          Clear interrupt condition
        btst.b    #1,HMD_PCDR(a3)         Is BINH already asserted?
        bne.s     TIRQ_1                  jump if it is

        move.b    #3,HMD_PCDR(a3)         Assert BINH
        move.b    V_TIME+3(a2),HMD_CPRL(a3)  measure time, low byte
        move.b    V_TIME+2(a2),HMD_CPRM(a3)  measure time, mid byte
        move.b    V_TIME+1(a2),HMD_CPRH(a3)  measure time, high byte
        move.b    #%10110001,HMD_TCR(a3)  Start timer
        tst      d0                      Clear carry
        rts

TIRQ_1    move.w    V_WAKE(a2),d0          Get ID of process to wake
        move.b    #$0,HMD_PCDR(a3)        negate INIT
        moveq     $$Wake,d1              Send special wakeup signal
        os9       F$Send
        clr.w     V_WAKE(a2)             The process is no longer waiting
        tst      d0
        rts

Term      movea.l    V_PORT(a2),a3        Get port address
        move.b    #$00,HMD_PCDR(a3)
        move.b    #$00,HMD_PADR(a3)
        move.b    #$00,HMD_PBDR(a3)
        move.b    #$86,HMD_PACR(a3)      Disable IRQ's from port A
        move.b    #$86,HMD_PBCR(a3)      Disable IRQ's from port B
        move.b    #$00,HMD_TCR(a3)      Disable IRQ's from timer
        move.b    d0,HMD_ICTRL(a3)       Reset IRQ level (any byte write will do)
        move.b    $FFF,HMD_PSR(a3)       Reset Hx flip-flops
        move.b    $FFF,HMD_TSR(a3)       Reset timer zero detect

        movea.l    #0,a0                  I want to remove IRQ handler
        move.b    M$Vector(a1),d0        Get interrupt vector
        ori.b     #00000011,d0          Start with H4 IRQ handler
        os9       F$IRQ                  Remove IRQ handler
        bcs.s     Term_err               Jump if error
        subq.b    #1,d0                  Go on with H3 IRQ handler
        os9       F$IRQ                  Remove IRQ handler
        bcs.s     Term_err               Jump if error
        subq.b    #1,d0                  Go on with H2 IRQ handler
        os9       F$IRQ                  Remove IRQ handler
        bcs.s     Term_err               Jump if error
        subq.b    #1,d0                  Go on with H1 IRQ handler
        os9       F$IRQ                  Remove IRQ handler
        bcs.s     Term_err               Jump if error
        tst.w     d0                      Clear carry (no error)
        rts

Term_err   ori      #Carry,ccr            Set Carry (error)
        rts

*****
Read
*
* Passed:   (a0) Address of buffer
*           (a2) Device static storage
*           (a3) Drive table
*           (a4) Process descriptor of current process
*           (a6) System globals
*           d0.l Number of bytes to read
*
* Returns   d1.l Number of bytes read
*
*****

* ----> Verify the drive number (required of all SBF drivers)

```

```

*          move.b  PD_TDrv(a1),d1          Get drive number
*          cmp.b   SBF_NDrv(a2),d1        is drive number in range?
*          bge     Err_Unit                no! ...exit

* ----> Check if there is a suspended process, using the device

          tst.w    V_WAKE(a2)              Any process waiting ?
          bne      Err_DevBsy              yes! ...exit

* ----> Check if the supplied buffer is as specified

          move.l    a0,d1                  copy buffer address to data register
          btst      #0,d1                  buffer address on word boundary ?
          bne      Err_BPAddr              no! ...exit

          move.l    d0,d1                  copy buffer length to d1
          cmp.l     #16,d1                  is buffer too short?
          blt       Err_BPAddr              yes! ...exit

          cmp.l     #800,d1                 is buffer longer than needed for 99 echos?
          bgt       Err_BPAddr              yes! ...exit

          andi.b    #7,d1                  is buffer length multiple of 8?
          bne      Err_BPAddr              no! ...exit

* ----> Calculate maximum number of echos to be detected

          move.b    d0,d1                  restore d1.b
          asr.l     #3,d0                  divide by 8 ...
          subq.l    #1,d0                  and subtract 1, ...
          move.w    d0,V_MAXECHO(a2)        Store it in static storage

* ----> This part examines what sensors are to be active in this measurement.
*          It puts the numbers in the appropriate fields in the Read buffer.
*          It enables the sensor groups with active sensors.
*          It sets the sensor address lines to the appropriate values.

          movea.l   V_PORT(a2),a3          Get port address
          clr.b     HMD_PADR(a3)           Disable all sonar groups
          clr.b     HMD_PBDR(a3)           Set all sensor address lines to 0
          clr.l     (a0)                   Clear all 4 sensornumbers in ...
          clr.l     4(a0)                  read buffer

          move.b     3+V_SENSOR(a2),d0
          beq.s      Read_1
          bset.b     #3,HMD_PADR(a3)       Enable group 4
          subq.b     #1,d0                  move number from (1..) to (0..) domain
          btst       #7,d0                  Is increment bit set?
          beq.s      Rd_a                  If no, jump
          andi.b     #$83,3+V_SENSOR(a2)   Remove all bits but 0,1, and 7
          addq.b     #1,3+V_SENSOR(a2)     increment by one
          andi.b     #3,d0                  remove increment bit
Rd_a      move.b     d0,7(a0)               put sensornumber in read buffer
          addq.b     #1,7(a0)               restore to (1..) domain
          lsl.b      #6,d0                  move number 6 bits left
          or.b       d0,HMD_PBDR(a3)       set appropriate sensor address lines

Read_1     move.b     2+V_SENSOR(a2),d0
          beq.s      Read_2
          bset.b     #2,HMD_PADR(a3)
          subq.b     #1,d0
          btst       #7,d0                  Is increment bit set?
          beq.s      Rd_b                  If no, jump
          andi.b     #$83,2+V_SENSOR(a2)
          addq.b     #1,2+V_SENSOR(a2)
          andi.b     #3,d0
Rd_b      move.b     d0,5(a0)
          addq.b     #1,5(a0)
          lsl.b      #4,d0
          or.b       d0,HMD_PBDR(a3)

Read_2     move.b     1+V_SENSOR(a2),d0
          beq.s      Read_3
          bset.b     #1,HMD_PADR(a3)
          subq.b     #1,d0
          btst       #7,d0                  Is increment bit set?
          beq.s      Rd_c                  If no, jump
          andi.b     #$83,1+V_SENSOR(a2)
          addq.b     #1,1+V_SENSOR(a2)
          andi.b     #3,d0

```



```

Rd_c      move.b  d0,3(a0)
          addq.b  #1,3(a0)
          lsl.b   #2,d0
          or.b    d0,HMD_PBDR(a3)

Read_3     move.b  V_SENSOR(a2),d0
          beq.s    Read_4
          bset.b   #0,HMD_PADR(a3)
          subq.b   #1,d0
          btst     #7,d0
          beq.s    Rd_d
          andi.b   #$83,V_SENSOR(a2)
          addq.b   #1,V_SENSOR(a2)
          andi.b   #3,d0
          Is increment bit set?
          If no, jump

Rd_d       move.b  d0,1(a0)
          addq.b   #1,1(a0)
          or.b     d0,HMD_PBDR(a3)

Read_4     move.l   #0,V_ECHOS(a2)
          Set number of echos received to 0 for all groups
          move.l   a0,d0
          addq.l   #8,d0
          move.l   d0,V_DAT(a2)
          addq.l   #2,d0
          move.l   d0,4+V_DAT(a2)
          addq.l   #2,d0
          move.l   d0,8+V_DAT(a2)
          addq.l   #2,d0
          move.l   d0,12+V_DAT(a2)
          clr.l    8(a0)
          Set first 4 echo fields to 0
          clr.l    12(a0)

          move.w    V_BUSY(a2),V_WAKE(a2)
          move.b    #%10110000,HMD_TCR(a3)
          Prepare to use timer
          move.b    V_BINH+3(a2),HMD_CPRL(a3)
          BINH time, low byte
          move.b    V_BINH+2(a2),HMD_CPRM(a3)
          BINH time, mid byte
          move.b    V_BINH+1(a2),HMD_CPRH(a3)
          BINH time, high byte
          move.b    #$01,HMD_PCDR(a3)
          assert INIT
          move.b    #%10110001,HMD_TCR(a3)
          Start timer

Read_5     move.l   #00,d0
          os9       F$Sleep
          sleep indefinitely
          tst.w     V_WAKE(a2)
          Sleep until woken
          bne.s     Read_5
          woken by interrupt handler?
          If not, go back to sleep

          tst.w     d0
          Clear Carry
          rts       Return

*****
Write
*
* Write block to device
*
* Input:
*   d0.l    =   buffer size
*   (a0)    =   address of buffer
*   (a2)    =   address of the device static storage area
*   (a3)    =   drive table
*   (a4)    =   process descriptor pointer
*   (a6)    =   system global data storage pointer
*
* Output: NONE
* Error Output:
*   cc      =   carry bit set
*   dl.w    =   Appropriate error code
*
*****

          bclr     #1,SBF_DFlg(a3)
          necessary to bypass SBF activities (MAGIC?)

* ----> Check if the supplied buffer is as specified

          move.l   a0,d1
          copy buffer address to data register
          btst     #0,d1
          buffer address on word boundary ?
          bne.s    Err_BPAddr
          no! ...exit

          move.l   d0,d1
          copy buffer length to d1
          cmpi.l   #4,d1
          is buffer 4 bytes long?

```

```
        beq.s    Write_1                yes! jump
        cmpi.l   #8,d1                 is buffer 8 bytes long?
        bne.s    Err_BPAddr            no! ... exit

        move.l   (a0),d0                get BINH time
        asr.l    #2,d0                  divide by 4 to get time in 4us intervals
        move.l   d0,V_BINH(a2)          update static storage
        move.l   4(a0),d0                get LISTEN time
        asr.l    #2,d0                  divide by 4 to get time in 4us intervals
        move.l   d0,V_TIME(a2)          update static storage
        rts

Write_1    move.l   (a0),V_SENSOR(a2)    copy sensor list to static storage
        rts                               return

GetStat    move.w   #E$UnkSvc,d1
        ori      #Carry,ccr
        rts

SetStat

        move.w   #E$UnkSvc,d1
        ori      #Carry,ccr
        rts

Err_DevBsy move.w   #E$DevBsy,d1
        bra.s    Err_exit
Err_BMode  move.w   #E$BMode,d1
        bra.s    Err_exit
Err_BPAddr move.w   #E$BPAddr,d1
        bra.s    Err_exit
Err_Param  move.w   #E$Param,d1
        bra.s    Err_exit
Err_Unit  move.w   #E$Unit,d1
Err_exit   ori      #Carry,ccr
        rts
ends
```

19.15.5 test.c

Kildetekst til testprogram. Testprogrammet er menustyret, og selvforklarende.

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

#define STEP 76
#define N_ECHO 99

/*****
Definitions for longword, word, and byte
*****/

typedef unsigned long    uint_32;
typedef unsigned short   uint_16;
typedef unsigned char    uint_8;

/*****
Prototypes
*****/

void display(d);
int get_sensorlist(sensorlist);
int get_timing(time);
char menu();

/*****
Main program
*****/

void main()
{
    int    i,n,path;
    char   c;
    uint_16 d[N_ECHO][4];
```

```

uint_8  sensorlist[4];
uint_32 time[2];

path=open("/hmd",3);
if (path!=-1) exit(errno);

while (1) {
    switch (menu()) {
        case 'a' :
            do {
                n=read(path,d,8*N_ECHO);
                if (n==-1) exit(errno);
                display(d);
                printf("-----\nPress M for menu, R to examine raw data, other key to measure again\n");
                while (read(1,&c,1)==-1) tsleep(2); /* wait to press key */
                if ((c=='r') || (c=='R')) {
                    printf("\fRAW DATA BLOCK\n\n");
                    for (n=0;n<N_ECHO;n++) printf("N: %2d | %5d %5d %5d %5d\n",n,d[n][0],d[n][1],d[n][2],d[n][3]);
                    printf("\nPress key!\n");
                    while (read(1,&c,1)==-1) tsleep(2); /* wait to press key */
                }
            } while ((c!='m') && (c!='M'));
            break;
        case 'b' :
            if (!get_sensorlist(sensorlist)) break;
            n=write(path,sensorlist,4);
            if (n==-1) exit(errno);
            break;
        case 'c' :
            if (!get_timing(time)) break;
            n=write(path,time,8);
            if (n==-1) exit(errno);
            printf("\nTiming updated with: %d %d\n",time[0],time[1]);
            while (read(1,&c,1)==-1) tsleep(2); /* wait to press key */
            break;
        case 'q' :
            printf("\n\nGoodbye!\n");
            close(path);
            exit(0);
    }
}

void display(d)
uint_16 d[N_ECHO][4];
{
    int i,j,k,n;
    char c;

    printf("\f ECHO DISPLAY\n\n");
    for (i=0;i<4;i++) {
        printf("Group %d: ",i+1);
        if (d[0][i]!=0) printf("Sensor %d\n",d[0][i]);
        else printf("          Disabled\n");
    }
    printf("
-----1-----2-----3-----4-----5-----6-----7----->\n");
    for (i=0;i<4;i++) {
        if (d[0][i]!=0) {
            n=1;j=0;k=0;
            while (d[n][i]!=0) {
                while (j<d[n][i]) {
                    j=j+STEP;
                    if (j/STEP<80) if (k==0) printf(" ");
                    k=0;
                }
                k=0;
                while ((d[n][i]<=j) && (d[n][i]!=0)){
                    k++;n++;
                }
                if ((d[n][i]!=0) && (j/STEP<80)) {
                    if (k>9) printf("***");
                    else printf("%d",k);
                }
            }
        }
    }
    printf("\n");
}

```

```
}

int get_sensorlist(sensorlist)
uint_8 *sensorlist;
{
    int n,i;
    char c;

    printf("\f SENSOR LIST\n\n");
    for (i=0;i<4;i++) {
        printf("\nWhich sensor for group %d ? (0..4 - 0 = Group off) ",i+1);
        scanf("%d",&n);
        if ((n<0) || (n>4)) {
            printf("Number out of range!\n");
            while (read(1,&c,1)==-1) tsleep(2); /* wait to press key */
            return 0;
        }
        if (n!=0) {
            printf("Auto increment? (Y/N)\n");
            while (read(1,&c,1)==-1) tsleep(2); /* wait to press key */
            if ((c=='Y') || (c=='y')) n=n+128; /* set bit 7 */
        }
        sensorlist[i]=(uint_8)n;
    }
}

int get_timing(time)
uint_32 *time;
{
    char c;
    int n;

    printf("\f SET TIMING\n\n");
    printf("Enter number of microseconds from sound transmission,\n");
    printf("to beginning of 'listen mode' : ");
    scanf("%d",&time[0]);
    printf("\n");
    if (time[0]<4) {
        printf("The time interval is too short for the hardware to handle!\n");
        printf("ACTION CANCELED!\n");
        while (read(1,&c,1)==-1) tsleep(2); /* wait to press key */
        return 0;
    }
    if (time[0]<350) {
        printf("The time interval is shorter than the duration of the sonar ping\n");
        printf("But OK, you are the boss :-)\n\n");
    }
    if (time[0]<1000) {
        printf("If you begin 'listen mode' too soon, there is a chance that the transducer\n");
        printf("has not stabilized yet, which can result in false echo detections\n\n");
    }
    if (time[0]>2380) {
        printf("The sensor automatically switches to 'listen mode' after 2.38 ms.\n");
        printf("BINH intervals longer than this, has no effect, but does no harm!\n\n");
    }
    if (time[0]>64000) {
        printf("The BINH interval is longer than the 64ms life expectancy of the sonar ping!\n");
        printf("You are beeing silly!\n");
        printf("ACTION CANCELED!\n");
        while (read(1,&c,1)==-1) tsleep(2); /* wait to press key */
        return 0;
    }
    printf("Enter duration of 'listen mode' in microseconds : ");
    scanf("%d",&time[1]);
    if (time[1]<4) {
        printf("The time interval is too short for the hardware to handle!\n");
        printf("ACTION CANCELED!\n");
        while (read(1,&c,1)==-1) tsleep(2); /* wait to press key */
        return 0;
    }
    if (time[1]>64000) {
        printf("The life expectancy of the sonar ping is 64ms\n\n");
    }
    if (time[1]>128000) {
        printf("The BINH interval is far longer than the 64ms life expectancy of the sonar ping!\n");
        printf("You are beeing silly!\n");
        printf("ACTION CANCELED!\n");
        while (read(1,&c,1)==-1) tsleep(2); /* wait to press key */
        return 0;
    }
}
```

```
    return 1;
}

char menu()
{
    char c;
    while (1) {
        printf("\f WELCOME TO THE TESTPROGRAM FOR HEIMDAL\n\n");
        printf("A: Make measurement\n\n");
        printf("B: Enter list of active sensors\n\n");
        printf("C: Change sensor timing\n\n");
        printf("Q: Quit\n\n");
        printf("Make Your choice!\n");

        while (read(1,&c,1)==-1) tsleep(2);    /* Try to read one char from stdin
                                                until there is a char to be read.
                                                Sleep in between, to save CPU time */

        switch (c) {
            case 'a':
            case 'b':
            case 'c':
            case 'q': return c;                /* Return the char if it is a,b,c, or q */
        }
        /* Otherwise, repeat the menu */
    }
}
```


Kapitel 20

Programmel til Scrobot

Programmellet til Scroboten består af et funktionsbibliotek til Scroboten, et testprogram der indeholder en algoritme til invers kinematik, og en datafil til testprogrammet.

Al programmellet er udviklet til OS-9 V.2.2, på IMADA's PEP VME-bus datamater. På nær datafilen er alle programmer skrevet til Microware C.

20.1 Struktur

Til at styre Scroboten bruges i alt 6 ud af 8 akser på to Gefion motorcontrollere. Funktionsbiblioteket til Scroboten anvender derfor funktionsbiblioteket til Gefion.

Scrobotens funktionsbibliotek stiller nogle få funktioner til rådighed, der anvendes af testprogrammet til Scroboten.

Testprogrammet indlæser en datafil, der i hver linie rummer en toolspids-vektor, en griberposition, og en kørselstid.

20.1.1 Makefil

Følgende *makefil* anvendes ved kompileringen af programmerne, og illustrerer deres indbyrdes afhængigheder.

```
MYDIR = /dd/usr/zaphod

ODIR = $(MYDIR)/EXE
SDIR = $(MYDIR)/CSOURCE
RDIR = $(MYDIR)/RELS

LDIR = $(MYDIR)/LIB

CFLAGS = -g -k2F

#
# scrobot
#

scrobot:      $(LDIR)/scrobot.l
              rdump -l $(LDIR)/scrobot.l

$(LDIR)/scrobot.l:  scrobot.r $(LDIR)/MCI.l
                  -del $(LDIR)/scrobot.l
                  @merge $(RDIR)/scrobot.r $(LDIR)/MCI.l >$(LDIR)/scrobot.l
```

```

scorbot.r:      scorbot.c scorbot.h mechanics.h
               @cc $(SDIR)/scorbot.c $(CFLAGS) -r=$(RDIR)

#
# test
#

test:   test.r $(LDIR)/scorbot.l
        @cc $(RDIR)/$*.r -l=$(LDIR)/scorbot.l -f=$(ODIR)/$*

test.r: test.c scorbot.h mechanics.h
        @cc $(SDIR)/$*.c $(CFLAGS) -r=$(RDIR)

```

20.2 Funktionsbibliotek til Scorbot

Funktionsbiblioteket sammensættes af filerne `scorbot.h`, `mechanics.h` og `scorbot.c` til biblioteket `scorbot.l` som andre programmer kan linkes sammen med.

mechanics.h Indeholder definitioner af Denavit-Hartenberg parametre, mekaniske begrænsninger, reguleringsparametre, og hastighedsbegrænsninger.

scorbot.h Indeholder relevante hardwaremæssige definitioner, en datastruktur der skal bruges som *nøgle* til alle funktionskald, samt prototyper på de funktioner der stilles til rådighed.

scorbot.c Rummer implementationen af alle de funktioner der stilles til rådighed, samt diverse hjælpefunktioner.

Funktionsbiblioteket stiller følgende funktioner til rådighed:

20.2.1 void SB_Init(adr1,adr2,keyptr)

adr1, adr2 (unsigned long) Basisadresserne på de to anvendte Gefion motorcontrollere.

keyptr (SB_KEYTYPE *) Pointer til den *nøgle* der skal anvendes ved alle funktionskald.

Funktionen initialiserer nødvendige hard- og software-komponenter, og initialiserer den givne *nøgle*, så den giver adgang til de øvrige funktionskald.

20.2.2 void SB_Cleanup(keyptr)

keyptr (SB_KEYTYPE *) Pointer til *nøgle*.

Funktionen deinitialiserer relevante hard- og software-komponenter, deinitialiserer den givne *nøgle*, så den ikke længere giver adgang til andre funktionskald end SB_Init.

20.2.3 void SB_LoadJointMovement(keyptr, joint, speed, pos)

keyptr (SB_KEYTYPE *) Pointer til *nøgle*.

joint (int) Nummeret på det Joint der skal bevæges.

speed (double) Jointhastigheden i rad/s (m/s for griberen).

pos (double) Den nye jointposition i rad (meter for griberen).

Funktionen indlæser en bevægelseskommmando i kommandobufferen for et enkelt led. Hastigheden opgives numerisk (positiv), fortegnet beregnes fra positionen.

20.2.4 void SB_LoadMultiMovement(keyptr,speedarray,posarray)

keyptr (SB_KEYTYPE *) Pointer til *nøgle*.

speedarray (double[6]) Array med 6 jointhastigheder. (rad/s el. m/s)

posarray (double[6]) Array med 6 jointpositioner. (rad el. meter)

Funktionen indlæser en bevægelseskommmando i kommandobufferen for alle 6 led. Hastigheder opgives numerisk (positiv), fortegnene beregnes fra positionerne.

20.2.5 void SB_StartJointMovement(keyptr, joint)

keyptr (SB_KEYTYPE *) Pointer til *nøgle*.

joint (int) Nummeret på det led der skal bevæges.

Funktionen aktiverer den bevægelseskommmando der ligger i kommandobufferen for det pågældende led.

20.2.6 void SB_StartMultiMovement(keyptr)

keyptr (SB_KEYTYPE *) Pointer til *nøgle*.

Funktionen aktiverer bevægelseskommandoerne i kommandobufferene for alle 6 led.

20.2.7 void SB_KillMotor(keyptr, joint)

keyptr (SB_KEYTYPE *) Pointer til *nøgle*.

joint (int) Nummeret på det led der skal standses.

Funktionen deaktiverer motoren for det pågældende led.

20.2.8 void SB_KillAllMotors(keyptr)

keyptr (SB_KEYTYPE *) Pointer til *nøgle*.

Funktionen deaktiverer motoren for alle led.

20.2.9 void SB_FindHome(keyptr)

keyptr (SB_KEYTYPE *) Pointer til *nøgle*.

Funktionen udfører en *homing*-procedure, hvor robotens led køres frem og tilbage i en sekvens, mens der ledes efter de vinkler hvor robotens *home*-kontakter aktiveres. Robotten efterlades i *home*-positionen

20.2.10 void SB_GetJointPositions(keyptr,posarray)

keyptr (SB_KEYTYPE *) Pointer til *nøgle*.

posarray (double[6]) Array af positioner. (rad el. m)

Funktionen aflæser positionen af alle 6 led, og lægger dem i posarray.

20.2.11 int SB_AllMotionsComplete(keyptr)

keyptr (SB_KEYTYPE *) Pointer til *nøgle*.

Funktionen returnerer værdien 1 hvis alle led er nået frem til deres ønskede positioner.

20.3 Testprogram med invers kinematik

Testprogrammet `test.c` Beder om et filnavn, åbner filen, og indlæser toolspidsvektorer, griberåbninger, og tidsintervaller fra filen.

For hvert (vektor, griber, tid) felt der indlæses anvender programmet en invers kinematik algoritme, til at omsætte toolspids-vektoren til 5 jointvinkler. Hvis det lykkedes køres de fem rotationsled og griberen hen til den ønskede konfiguration, på den tid der var angivet i filen, og næste felt i filen indlæses.

Den eneste interessante funktion i programmet er:

20.3.1 int invers(w,q0,q1)

w (float[6]) toolkonfigurationsvektoren, der rummer x,y,z koordinater for toolspidsens placering, og x,y,z koordinater for toolspidsens retningsvektor. Alle mål er i meter.

q0,q1 (double[5]) To arrays af jointvinkler. Når funktionen returnerer rummer de to arrays jointvinklerne for de op til to mulige løsninger.

Funktionen forsøger at finde begge løsninger til det invers-kinematiske problem. Jointvinklerne til den ene løsning lægges i q0, og vinklerne til den anden løsning lægges i q1. Funktionen returnerer værdien 1 hvis den ene løsning er fundet, 2 hvis den anden er fundet, og 3 hvis begge er fundet. Kunne ingen løsninger findes returneres 0.

20.4 Datafil til testprogram

Datafilerne til testprogrammet er almindelige *flade filer*, der for hver linie indeholder 8 tal. De første 6 er toolspids-vektoren, det næste er griberens åbning, og det sidste er et tidsinterval. Alle mål er i meter eller sekunder. Toolspidsvektoren består af (x, y, z, x', y', z') der er x,y,og z koordinaten for placeringen af robotens værktøj, og x,y, og z koordinaten for retningsvektoren af robotens værktøj.

20.5 Kildetekster

20.5.1 scorbot.h

/******

```

----- scorbot.h -----

Definitions modul til interface rutiner til SCORBOT

----- Denne fil indeholder prototyper til -----

+ SB_Init
+ SB_Cleanup
+ SB_LoadJointMovement
+ SB_LoadMultiMovement
+ SB_StartJointMovement
+ SB_StartMultiMovement
+ SB_KillMotor
+ SB_KillAllMotors
+ SB_FindHome
+ SB_GetJointPositions
+ SB_AllMotionsComplete

*****/
#ifndef MCI
#define MCI
#include "../DEFS/MCI.h"
#endif

#include "mechanics.h"

#define CRYSTALFREQ 12000000 /* Krystalfrekvensen */
#define SAMPLEFREQ (CRYSTALFREQ/4096) /* Sample frekvensen */

#define AXES 6

#define BASE 0
#define SHLD 1
#define ELBOW 2
#define WRIST_A 3
#define WRIST_B 4
#define GRIPPER 5

#ifndef PI
#define PI 3.1415
#endif

#define SB_KEYTYPE struct SB_KEY

struct SB_KEY
{
    MCI_KEYTYPE PIT[2];
    MCI_KEYTYPE motor[6];
};

void SB_Init(adrl,adr2,keyptr);
void SB_Cleanup(keyptr);
void SB_LoadJointMovement(keyptr, joint, speed, pos);
void SB_LoadMultiMovement(keyptr, speedarray, posarray);
void SB_StartJointMovement(keyptr, joint);
void SB_StartMultiMovement(keyptr);
void SB_KillMotor(keyptr, joint);
void SB_KillAllMotors(keyptr);
void SB_FindHome(keyptr);
void SB_GetJointPositions(keyptr, posarray);
int SB_AllMotionsComplete(keyptr);

```

20.5.2 mechanics.h

```

/***** Parametre til kinematik *****/

#define DH_D { 0.349, 0.000, 0.000, 0.000, 0.145 } /* d iflg. Denavit Hartenberg */
#define DH_A { 0.016, 0.223, 0.220, 0.000, 0.000 } /* a iflg. Denavit Hartenberg */

#define CONSTRAINTS_MAX { 2.670, 0.580, 2.800, 0.380, 99999, 0.065 } /* i radianer (gripper i meter) */
#define CONSTRAINTS_MIN { -2.670, -2.227, -2.270, -3.600, -99999, 0.000 } /* i radianer (gripper i meter) */

#define SOFT_HOME { 705, 4115, 925, 103, 377, 0 } /* offset til det matematiske nulpunkt */

/***** Parametre til regulering *****/

#define REG_INTERVAL { 255, 255, 255, 255, 255, 255 } /* Sample interval for differentiation */
#define REG_P { 2000, 2000, 2000, 2000, 5000, 5000, 2000 } /* P led i reguleringen */
#define REG_I { 200, 200, 200, 50, 50, 0 } /* I led i reguleringen */
#define REG_D { 100, 100, 100, 200, 200, 0 } /* D led i reguleringen */
#define REG_LIMIT { 20000, 20000, 20000, 20000, 20000, 20000 } /* Oevre graense for integralet */

#define POS_ERROR { 50, 50, 50, 100, 100, 50 } /* Stoerst tilladelige positionsfejl */

#define ACCELERATION { 0.0004, 0.0004, 0.0004, 0.0005, 0.0005, 0.0005 } /* Accelerationer */

/***** Parametre til 'home search' *****/

#define TOP_SPEED { 0.3, 0.25, 0.3, 0.3, 0.3, 0.3 } /* Top hastighed (incrementer / sampleinterval) */
#define FULL_TURN { 15450, 12056, 12056, 6028, 6028, 61700 } /* Incrementer for en hel omdrejning af en akse */
/* gripper: incrementer per 2 PI meter */

```

20.5.3 scorbot.c

```

/*****
----- scorbot.c -----

Implementations modul til interface rutiner til Scorbot

----- Denne fil indeholder funktionerne -----

+ SB_Init
+ SB_Cleanup
+ SB_LoadJointMovement
+ SB_LoadMultiMovement
+ SB_StartJointMovement
+ SB_StartMultiMovement
+ SB_KillMotor
+ SB_KillAllMotors

og deres hjælpefunktioner

*****/
#include "scorbot.h"

#define MKEY(N) keyptr->motor[N]
#define PITKEY(N) keyptr->PIT[N]
#define MOVE_MOTOR(axis,vel,pos) MCI_PositionMode(&MKEY(axis), VEL_ABS|POS_ABS, 0.0, vel, pos); \
MCI_StartMotion(&MKEY(axis))
#define MOVE_MOTOR_REL(axis,vel,pos) MCI_PositionMode(&MKEY(axis), VEL_ABS|POS_REL, 0.0, vel, pos); \
MCI_StartMotion(&MKEY(axis))

#define STOP_MOTOR(axis,action) MCI_StopMotor(&MKEY(axis),action)

#define MOTOR_POSITION(axis) MCI_ReadRealPos(&MKEY(axis))
#define POS_ERR(axis) MCI_PositionError(&MKEY(axis))
#define HOMESWITCH(axis) SB_Homeswitch(keyptr,axis)

#define WAIT_POS_ERR(axis) while (!POS_ERR(axis)) {}
#define WAIT_TRAJ_CMPLT(axis) while (!MCI_TrajectoryComplete(&MKEY(axis))) {}
#define RESET_POS_ERR(axis) MCI_ResetInt(&MKEY(axis),PEI)
#define RESET_TRAJ_CMPLT(axis) MCI_ResetInt(&MKEY(axis),TCI)

#define WAIT_HOMESWITCH(axis) while (!HOMESWITCH(axis)) {}
#define WAIT_NOT_HOMESWITCH(axis) while (HOMESWITCH(axis)) {}

#define INCREMENTS(angle,axis) ((angle * (double)full_turn[axis])/(2.0*PI))
#define ANGLE(increments,axis) (((double)increments) * 2.0*PI)/((double)full_turn[axis])

/***** 'interne' prototyper *****/

void SB_FindBaseHome(keyptr);
void SB_FindShoulderHome(keyptr);
void SB_FindAxisHome(keyptr,axis);
void SB_FindPitchHome(keyptr);
void SB_FindRollHome(keyptr);
void SB_FindGripperHome(keyptr);
void SB_FindHome(keyptr);
int SB_HomeSwitch(keyptr,axis);

/***** Konstant definitioner *****/

static unsigned int reg_p[AXES] = REG_P;
static unsigned int reg_i[AXES] = REG_I;
static unsigned int reg_d[AXES] = REG_D;
static unsigned int reg_interval[AXES] = REG_INTERVAL;
static unsigned int reg_limit[AXES] = REG_LIMIT;

static unsigned int pos_error[AXES] = POS_ERROR;

static double acceleration[AXES] = ACCELERATION;

static double top_speed[AXES] = TOP_SPEED;
static unsigned long full_turn[AXES] = FULL_TURN;
static long soft_home[AXES] = SOFT_HOME;

/***** Funktioner *****/

void SB_Init(adr1,adr2,keyptr)
LONGWORD adr1, adr2;
SB_KEYTYPE *keyptr;
{
    int n;

    MCI_Link(adr1,&PITKEY(0),&MKEY(0),&MKEY(1),&MKEY(2),&MKEY(3));
    MCI_Link(adr2,&PITKEY(1),&MKEY(4),&MKEY(5),NOMOTOR,NOMOTOR);
    MCI_InitParallelInput(&PITKEY(0));
    MCI_InitParallelInput(&PITKEY(1));

    for (n=0; n<AXES; n++)
    {
        MCI_ResetMotor(&MKEY(n));
        MCI_LoadFilter(&MKEY(n),reg_interval[n],reg_p[n],reg_i[n],reg_d[n],reg_limit[n]); /* Nulstil motorer */
        MCI_UpdateFilter(&MKEY(n)); /* Indstil PID filtre */
        MCI_LoadPosError(&MKEY(n),pos_error[n],STOP); /* Opdater PID filtre */
        MCI_PositionMode(&MKEY(n),(ACC_ABS | OFF),acceleration[n],0.0,0); /* Indstil pos fejl */
        MCI_StartMotion(&MKEY(n)); /* Indstil acceleration */
        /* Opdater acceleration */
    }
    SB_FindHome(keyptr);
}

void SB_Cleanup(keyptr)
SB_KEYTYPE *keyptr;
{
    int n;

```

```

MCI_ReleaseKey(&PITKEY(0));
MCI_ReleaseKey(&PITKEY(1));
for (n=0; n<6; n++) MCI_ReleaseKey(&MKEY(n));
}

void SB_FindBaseHome(keyptr)
SB_KEYTYPE *keyptr;
{
    MOVE_MOTOR(BASE, top_speed[BASE], full_turn[BASE]);
    while (!POS_ERR(BASE) & !HOMESWITCH(BASE)) {};
    if (POS_ERR(BASE))
    {
        RESET_POS_ERR(BASE);
    }
    else
    {
        MOVE_MOTOR(BASE, top_speed[BASE], MOTOR_POSITION(BASE) + full_turn[BASE]/50);
        WAIT_TRAJ_CMPLT(BASE);
    }
    MOVE_MOTOR(BASE, top_speed[BASE], -full_turn[BASE]);
    WAIT_HOMESWITCH(BASE);
    RESET_TRAJ_CMPLT(BASE);
    MOVE_MOTOR(BASE, top_speed[BASE]/5, MOTOR_POSITION(BASE) + full_turn[BASE]/180);
    WAIT_TRAJ_CMPLT(BASE);
    MOVE_MOTOR(BASE, top_speed[BASE]/10, -full_turn[BASE]);
    WAIT_HOMESWITCH(BASE);
    STOP_MOTOR(BASE, ABRUPT);
}

void SB_FindShoulderHome(keyptr)
SB_KEYTYPE *keyptr;
{
    MOVE_MOTOR(SHLD, top_speed[SHLD], -full_turn[SHLD]);
    MOVE_MOTOR(ELBOW, top_speed[ELBOW]/1.5, -full_turn[ELBOW]);
    WAIT_HOMESWITCH(SHLD);
    STOP_MOTOR(SHLD, ABRUPT);
    STOP_MOTOR(ELBOW, OFF);

    RESET_TRAJ_CMPLT(SHLD);
    MOVE_MOTOR_REL(SHLD, top_speed[SHLD]/5, full_turn[SHLD]);
    WAIT_NOT_HOMESWITCH(SHLD);
    MOVE_MOTOR(SHLD, top_speed[SHLD]/5, MOTOR_POSITION(SHLD)+full_turn[SHLD]/180);
    WAIT_TRAJ_CMPLT(SHLD);
    MOVE_MOTOR(SHLD, top_speed[SHLD]/10, -full_turn[SHLD]);
    WAIT_HOMESWITCH(SHLD);
    STOP_MOTOR(SHLD, ABRUPT);
}

void SB_FindElbowHome(keyptr)
SB_KEYTYPE *keyptr;
{
    MOVE_MOTOR(ELBOW, top_speed[ELBOW], full_turn[ELBOW]);
    while (!POS_ERR(ELBOW) & !HOMESWITCH(ELBOW)) {};
    if (POS_ERR(ELBOW))
    {
        RESET_POS_ERR(ELBOW);
    }
    else
    {
        MOVE_MOTOR(ELBOW, top_speed[ELBOW], MOTOR_POSITION(ELBOW) + full_turn[ELBOW]/10);
        WAIT_TRAJ_CMPLT(ELBOW);
    }
    MOVE_MOTOR(ELBOW, top_speed[ELBOW], -full_turn[ELBOW]);
    WAIT_HOMESWITCH(ELBOW);
    RESET_TRAJ_CMPLT(ELBOW);
    MOVE_MOTOR(ELBOW, top_speed[ELBOW]/5, MOTOR_POSITION(ELBOW) + full_turn[ELBOW]/180);
    WAIT_TRAJ_CMPLT(ELBOW);
    MOVE_MOTOR(ELBOW, top_speed[ELBOW]/10, -full_turn[ELBOW]);
    WAIT_HOMESWITCH(ELBOW);
    STOP_MOTOR(ELBOW, ABRUPT);
}

void SB_FindPitchHome(keyptr)
SB_KEYTYPE *keyptr;
{
    double spd;

    if (top_speed[WRIST_A] > top_speed[WRIST_B]) spd = top_speed[WRIST_A]/2;
    else spd = top_speed[WRIST_B]/2;

    MOVE_MOTOR(WRIST_A, spd, full_turn[WRIST_A]);
    MOVE_MOTOR(WRIST_B, spd, full_turn[WRIST_B]);
    while (!POS_ERR(WRIST_A) & !POS_ERR(WRIST_B) & !HOMESWITCH(WRIST_A)) {};
    if (POS_ERR(WRIST_A) | POS_ERR(WRIST_B))
    {
        STOP_MOTOR(WRIST_A, ABRUPT);
        STOP_MOTOR(WRIST_B, ABRUPT);
        RESET_POS_ERR(WRIST_A);
        RESET_POS_ERR(WRIST_B);
    }
    else
    {
        MOVE_MOTOR(WRIST_A, spd, MOTOR_POSITION(WRIST_A) + full_turn[WRIST_A]/10);
        MOVE_MOTOR(WRIST_B, spd, MOTOR_POSITION(WRIST_B) + full_turn[WRIST_B]/10);
        WAIT_TRAJ_CMPLT(WRIST_A);
        WAIT_TRAJ_CMPLT(WRIST_B);
    }
    MOVE_MOTOR(WRIST_A, spd, -full_turn[WRIST_A]);
    MOVE_MOTOR(WRIST_B, spd, -full_turn[WRIST_B]);
    WAIT_HOMESWITCH(WRIST_A);
    RESET_TRAJ_CMPLT(WRIST_A);
    RESET_TRAJ_CMPLT(WRIST_B);
    MOVE_MOTOR(WRIST_A, spd/5, MOTOR_POSITION(WRIST_A) + full_turn[WRIST_A]/80);
    MOVE_MOTOR(WRIST_B, spd/5, MOTOR_POSITION(WRIST_B) + full_turn[WRIST_B]/80);
}

```

```

    WAIT_TRAJ_CMPLT(WRIST_A);
    WAIT_TRAJ_CMPLT(WRIST_B);
    MOVE_MOTOR(WRIST_A,spd/10,-full_turn[WRIST_A]);
    MOVE_MOTOR(WRIST_B,spd/10,-full_turn[WRIST_B]);
    WAIT_HOMESWITCH(WRIST_A);
    STOP_MOTOR(WRIST_A,ABRUPT);
    STOP_MOTOR(WRIST_B,ABRUPT);
}

void SB_FindRollHome(keyptr)
SB_KEYTYPE *keyptr;
{
    double spd;

    if (top_speed[WRIST_A] > top_speed[WRIST_B]) spd = top_speed[WRIST_A];
    else spd = top_speed[WRIST_B];

    MOVE_MOTOR(WRIST_A,spd/2, full_turn[WRIST_A]*2);
    MOVE_MOTOR(WRIST_B,spd/2,-full_turn[WRIST_B]*2);
    WAIT_HOMESWITCH(WRIST_B);
    MOVE_MOTOR(WRIST_A,spd/5,MOTOR_POSITION(WRIST_A) - full_turn[WRIST_A]/30);
    MOVE_MOTOR(WRIST_B,spd/5,MOTOR_POSITION(WRIST_B) + full_turn[WRIST_B]/30);
    WAIT_TRAJ_CMPLT(WRIST_A);
    WAIT_TRAJ_CMPLT(WRIST_B);
    MOVE_MOTOR(WRIST_A,spd/10, full_turn[WRIST_A]*4);
    MOVE_MOTOR(WRIST_B,spd/10,-full_turn[WRIST_B]*4);
    WAIT_HOMESWITCH(WRIST_B);
    STOP_MOTOR(WRIST_A,ABRUPT);
    STOP_MOTOR(WRIST_B,ABRUPT);
}

void SB_FindGripperHome(keyptr)
SB_KEYTYPE *keyptr;
{
    MOVE_MOTOR(GRIPPER, top_speed[GRIPPER], -full_turn[GRIPPER]);
    WAIT_POS_ERR(GRIPPER);
    RESET_POS_ERR(GRIPPER);
}

void SB_FindHome(keyptr)
SB_KEYTYPE *keyptr;
{
    int n;

    SB_FindShoulderHome(keyptr);
    SB_FindElbowHome(keyptr);
    SB_FindPitchHome(keyptr);
    SB_FindRollHome(keyptr);
    SB_FindGripperHome(keyptr);
    SB_FindBaseHome(keyptr);
    SB_FindPitchHome(keyptr);
    for (n=0; n<AXES; n++) MCI_DefineHome(&MKEY(n));
}

int SB_Homeswitch(keyptr,axis)
SB_KEYTYPE *keyptr;
int axis;
{
    switch (axis)
    {
        case 0: case 1: case 2: case 3:
            return ((MCI_ParallelInput(&PITKEY(0)) & (0x0001 << (4*axis))) !=0);
            break;
        case 4:
            return ((MCI_ParallelInput(&PITKEY(1)) & 0x0001) != 0);
            break;
        default:
            exit(_errmsg(1,"SB_Homeswitch: axis not in range 0..4"));
            break;
    }
}

void SB_LoadJointMovement(keyptr, joint, speed, pos)
SB_KEYTYPE *keyptr;
int joint;
double speed;
double pos;
{
    MCI_PositionMode(&MKEY(joint),VEL_ABS|POS_ABS,0.0,INCREMENTS(speed,joint)/SAMPLEFREQ,((long)INCREMENTS(pos,joint)) + soft_home[joint]);
}

void SB_StartJointMovement(keyptr, joint)
SB_KEYTYPE *keyptr;
int joint;
{
    MCI_StartMotion(&MKEY(joint));
}

void SB_LoadMultiMovement(keyptr, speedarray, posarray)
SB_KEYTYPE *keyptr;
double *speedarray, *posarray;
{
    int n;
    for (n=0; n<AXES; n++)
    {
        MCI_PositionMode(&MKEY(n),VEL_ABS|POS_ABS,0.0,INCREMENTS(speedarray[n],n)/SAMPLEFREQ,(long)INCREMENTS(posarray[n],n) + soft_home[n]);
    }
}

void SB_StartMultiMovement(keyptr)
SB_KEYTYPE *keyptr;
{
    int n;
    for (n=0; n<AXES; n++) MCI_StartMotion(&MKEY(n));
}

```

```

void SB_KillMotor(keyptr, joint)
SB_KEYTYPE *keyptr;
int joint;
{
    MCI_StopMotor(&MKEY(joint), OFF);
}

void SB_KillAllMotors(keyptr)
SB_KEYTYPE *keyptr;
{
    int n;
    for (n=0; n<AXES; n++) MCI_StopMotor(&MKEY(n), OFF);
}

void SB_GetJointPositions(keyptr, posarray)
SB_KEYTYPE *keyptr;
double *posarray;
{
    int n;

    for (n=0; n<AXES; n++)
    {
        posarray[n] = ANGLE(MCI_ReadRealPos(&MKEY(n)) - soft_home[n], n);
    }
}

int SB_AllMotionsComplete(keyptr)
SB_KEYTYPE *keyptr;
{
    int n, r;

    r=1;
    for (n=0; n<AXES; n++)
    {
        r = r*MCI_TrajectoryComplete(&MKEY(n));
    }
    return r;
}

```

20.5.4 test.c

```

#include <stdio.h>
#include <math.h>
#include "scorbot.h"

#define EPSILON 0.0001

static double d[5] = DH_D;
static double a[5] = DH_A;
static double constraints_max[6] = CONSTRAINTS_MAX;
static double constraints_min[6] = CONSTRAINTS_MIN;

double atan2(y,x);
void calcspeeds(q,old,dq);
int invers(w,q);
double sign(x);
int fati(dptr,min,max); /* Fit Angle To Interval */

int fati(dptr,min,max)
double *dptr, min, max;
{
    while (*dptr < min) *dptr = *dptr+2*PI;
    while (*dptr > max) *dptr = *dptr-2*PI;
    if (*dptr < min) return 0;
    else return 1;
}

double sign(x)
double x;
{
    if (x == 0.0) return 0.0;
    if (x > 0.0) return 1.0;
    if (x < 0.0) return -1.0;
}

double atan2(y,x)
double x,y;
{
    if (x > 0.0) return atan(y/x);
    if (x == 0.0) return sign(y)*PI/2;
    if (x < 0.0) return atan(y/x) + PI;
}

void calcspeeds(q,old,dq,time)
double *q,*old,*dq,time;
{
    double max,dist[6],maxspeed;
    int n;

    for (n=0; n<6; n++)
    {
        dist[n] = fabs(q[n]-old[n]);
        if (dist[n] > max) max=dist[n];
    }
    printf("Max: %f\n",max);
    maxspeed=max/time;
    for (n=0; n<6; n++) dq[n] = maxspeed*(dist[n]/max);
}

int invers(w,q0,q1)
float *w;

```

```

double *q0,*q1;
{
    double fi[5], r, z, alpha,dot,dist,theta;
    int solution[2];

    solution[0] = 1;
    solution[1] = 1;

    dist = sqrt(w[0]*w[0]+w[1]*w[1]);
    if (dist != 0) dot = w[0]*w[3]+w[1]*w[4];
    else dot = w[3];
    if (w[2] < 0.0)
    if ((w[2] < 0.2) && (dist < .08))
    if ((w[2] < 0.4) && (w[2] >= 0.2) && (dist < .1))
    if ((fabs(fabs(dot)-dist*(sqrt(w[3]*w[3]+w[4]*w[4])))>EPSILON) && (dist!=0))
    {printf("Under underlaget !\n");return 0;}
    {printf("Inden i soklen!\n"); return 0;}
    {printf("Inden i scorboten!\n");return 0;}
    {printf("tool vektor ikke i x-y planet!\n");return 0;}

    theta = atan2(-sign(dot)*sqrt(w[3]*w[3]+w[4]*w[4]),-w[5]); /* theta er toolvinkelen if forhold til lodret */

    fi[4] = (PI*log(w[3]*w[3]+w[4]*w[4]+w[5]*w[5]))/2.0; /* Udtræk rotation fra længden af retningsvektoren */
    if (!fati(&fi[4],constraints_min[4],constraints_max[4]))
    {printf("Fi[4]: %f uden for interval!\n",fi[4]); return 0;}

    r = -a[0] + sqrt(w[0]*w[0] + w[1]*w[1]) + d[4]*sin(theta); /* Vandret afstand mellem skulder og håndled */
    z = -d[0] + w[2] + d[4]*cos(theta); /* Lodret afstand mellem skulder og håndled */

    alpha = (a[1]*a[1] + a[2]*a[2] - r*r - z*z)/(2*a[1]*a[2]);
    if ((alpha < -1.0) || (alpha > 1.0))
    {printf("alpha: %f uden for interval!\n",alpha); return 0;}

    fi[2] = PI-acos(alpha); /* Den ene mulighed er med albuen opad */
    if (!fati(&fi[2],constraints_min[2],constraints_max[2]))
    {printf("Fi[2]: %f uden for interval!\n",fi[2]); solution[0]=0;}
    else printf("Loesning 0: fi[2] = %f\n",fi[2]);

    if (solution[0])
    {
        q0[0] = (double)(atan2(w[1],w[0]));
        if (!fati(&q0[0],constraints_min[0],constraints_max[0]))
        {printf("q0[0]: %f uden for interval!\n",q0[0]); solution[0]=0;}
    }

    if (solution[0])
    {
        q0[1] = (double)(atan2(-(a[1]+a[2]*cos(fi[2]))*z - a[2]*sin(fi[2])*r,(a[1]+a[2]*cos(fi[2]))*r-a[2]*sin(fi[2])*z));
        if (!fati(&q0[1],constraints_min[1],constraints_max[1]))
        {printf("q0[1]: %f uden for interval!\n",q0[1]);solution[0]=0;}
    }

    if (solution[0])
    {
        q0[2] = (double)(fi[2] + q0[1]);
        fi[3] = theta-q0[2];
        if (!fati(&fi[3],constraints_min[3],constraints_max[3]))
        {printf("fi[3]: %f Uden for interval!\n",fi[3]); solution[0]=0;}
        theta = fi[3]+q0[2];
        q0[3] = (double)((theta + fi[4])/2.0);
        q0[4] = (double)((theta - fi[4])/2.0);
    }

    fi[2] = PI+acos(alpha); /* Den anden mulighed er albuen nedad */
    if (!fati(&fi[2],constraints_min[2],constraints_max[2]))
    {printf("Fi[2]: %f uden for interval!\n",fi[2]); solution[1]=0;}
    else printf("Loesning 1: fi[2] = %f\n",fi[2]);

    if (solution[1])
    {
        q1[0] = (double)(atan2(w[1],w[0]));
        if (!fati(&q1[0],constraints_min[0],constraints_max[0]))
        {printf("q1[0]: %f uden for interval!\n",q1[0]); solution[1]=0;}
    }

    if (solution[1])
    {
        q1[1] = (double)(atan2(-(a[1]+a[2]*cos(fi[2]))*z - a[2]*sin(fi[2])*r,(a[1]+a[2]*cos(fi[2]))*r-a[2]*sin(fi[2])*z));
        if (!fati(&q1[1],constraints_min[1],constraints_max[1]))
        {printf("q1[1]: %f uden for interval!\n",q1[1]);solution[1]=0;}
    }

    if (solution[1])
    {
        q1[2] = (double)(fi[2] + q1[1]);
        fi[3] = theta-q1[2];
        if (!fati(&fi[3],constraints_min[3],constraints_max[3]))
        {printf("fi[3]: %f Uden for interval!\n",fi[3]); solution[1]=0;}
        theta = fi[3]+q1[2];
        q1[3] = (double)((theta + fi[4])/2.0);
        q1[4] = (double)((theta - fi[4])/2.0);
    }

    return solution[0]+2*solution[1];
}
int moveto(keyptr,x,y,z,a,b,c,g,t);

int moveto(keyptr,x,y,z,a,b,c,g,t)
SB_KEYTYPE *keyptr;
float x,y,z,a,b,c;
double g,t;
{
    float w[6];
    double q1[6],q2[6],old[6],dq[6];
    int n;

    w[0] = x;
    w[1] = y;
    w[2] = z;
    w[3] = a;
    w[4] = b;
    w[5] = c;
    n = invers(w,q1,q2);
    if (n==0) return 0;
    if (n==2)
    {
        q2[5] = g;

```


0.2	0.15	0.01	0	0	-1	0.06	1	Slip_laaget
0.2	0.15	0.3	0	0	-1	0.06	3	op
0.4	0	0.3	1	0	0	0.06	2	over_flasken
0.4	0	0.215	1	0	0	0.06	2	ved_halsen
0.4	0	0.215	1	0	0	0	1	luk_griber
0.4	0	0.25	1	0	0	0	1	Loeft_flasken
0.4	-0.1	0.20	1.4067	-0.3517	0	0	2	Drej_og_saenk_flasken
0.4	-0.1	0.18	1.5134	-0.3784	0	0	0.5	Drej_lidt_mere
0.4	-0.1	0.15	1.6124	-0.4031	0	0	9.6	Hæld_af_flasken
0.4	-0.1	0.25	1.4067	-0.3517	0	0	2	Flasken_tilbage
0.4	0.1	0.2	0.6219	0.1555	0	0	6	Flasken_til_den_anden_side
0.4	0.1	0.175	0.5837	0.1459	0	0	2	Drej_lidt_mere
0.4	0.12	0.15	0.4789	0.1437	0	0	10	Toem_flasken
0.4	0.1	0.25	0.5837	0.1459	0	0	1	Flasken_tilbage
0.4	0	0.25	1	0	0	0	2	Flasken_lodret
0.4	0	0.210	1	0	0	0	1	Sæt_flasken
0.4	0	0.210	1	0	0	0.06	1	Slip_flasken
0.4	0	0.3	1	0	0	0.06	1	Op
0.3	-0.15	0.3	0	0	-1	0	4	Hjem

Kapitel 21

Dokumentation af liniesensor

Liniesensoren er beregnet til at måle den vandrette afstand til centrum af en linie på gulvet. Det er en forudsætning at linien er vinkelret på sensoren, inden for en margin på ca. $\pm 20^\circ$.

Sensoren er beregnet til at fungere med almindelig 19mm sort eller hvid PVC-tape på et lyst eller mørkt ensfarvet underlag, med en afstand til underlaget på 3 ... 5cm.

Sensoren er monteret i en plastkasse med indbygget DIN-stik, beregnet på montage i en sokkel. Hvis kassen åbnes er der mulighed for at justere på tre parametre:

Balance For at kompensere for elektroniske, optiske, eller mekaniske skævheder i balancen mellem forstærkningen af signalet i højre eller venstre side, kan balancen justeres på trimmeren R1.

Differensforstærkning Forstærkningen af differenssignalet kan justeres med trimmeren R21.

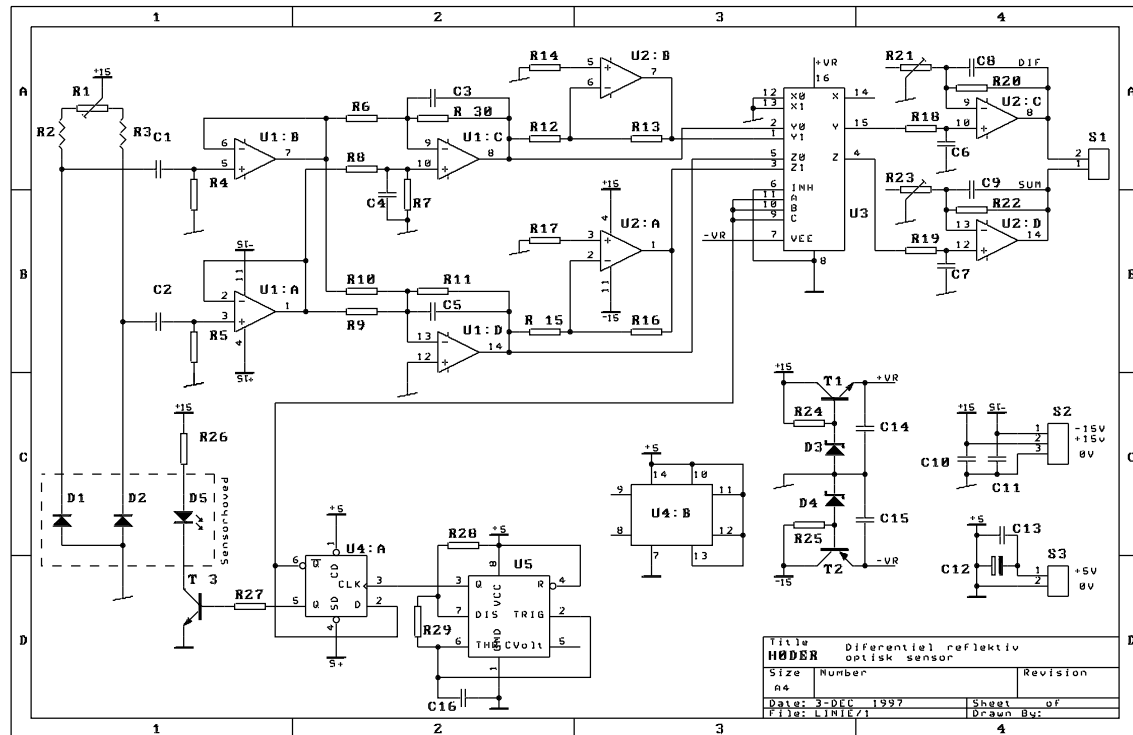
Sumforstærkning Forstærkningen af sumsignalet kan justeres med trimmeren R23

21.1 Elektriske forbindelser

ben	signal	ben	signal
1	Analog 0V	2	Sum udgang
3	Differens udgang	4	
5	+12V	6	Analog 0V
7	Analog 0V	8	-12V
9		10	Digital 0V
11	+5V		

Tabel 21.1: Elektriske forbindelser til liniesensor

21.2 Diagram

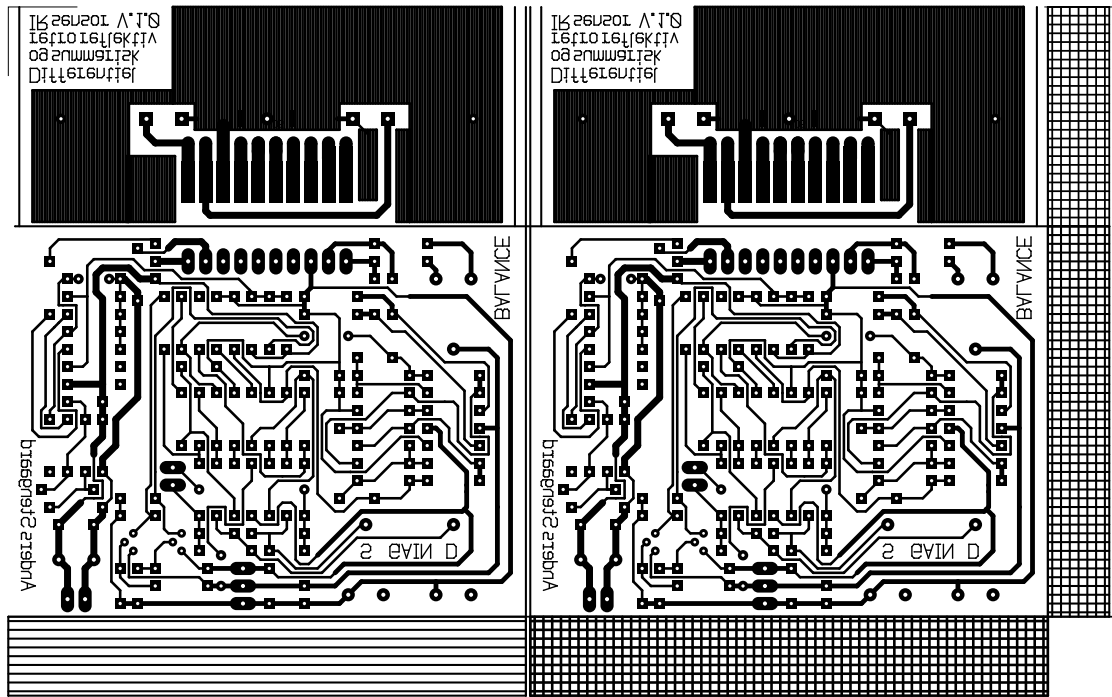


Figur 21.1: Diagram over linesensor

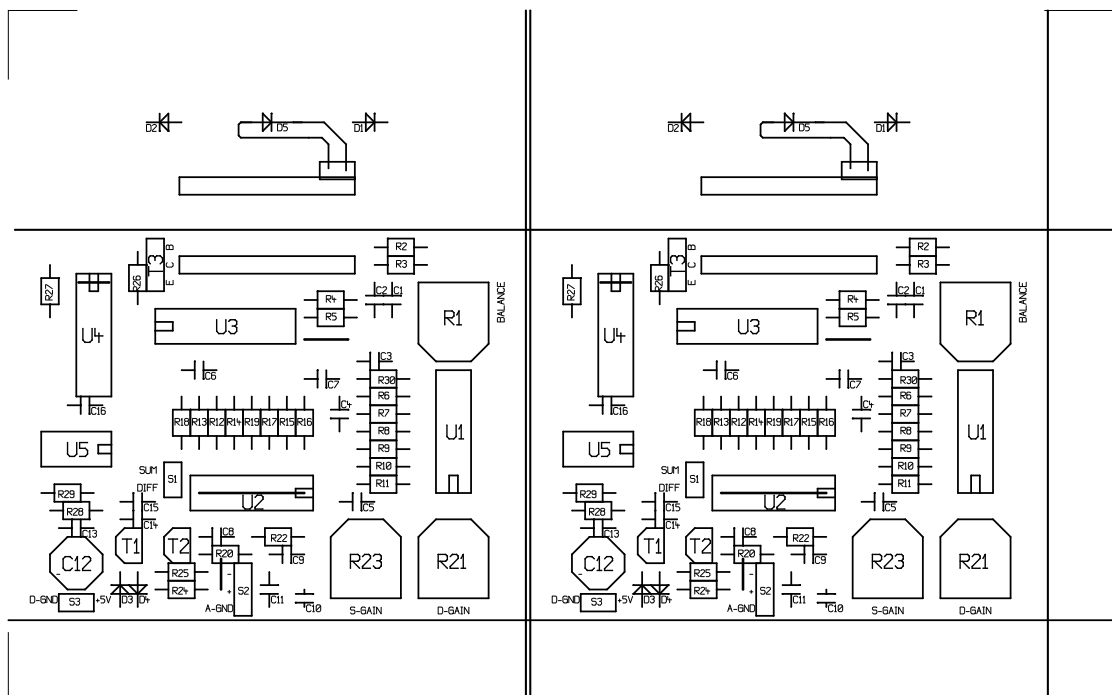
R1	10k Ω	Trim	R19	10k Ω		C10	10nF	keramisk
R2	47k Ω		R20	10k Ω		C11	10nF	keramisk
R3	47k Ω		R21	10k Ω	trim	C12	15 μ F	tantal
R4	270k Ω		R22	10k Ω		C13	100nF	keramisk
R5	270k Ω		R23	10k Ω	trim	C14	10nF	keramisk
R6	1k Ω		R24	47k Ω		C15	10nF	keramisk
R7	47k Ω		R25	47k Ω		C16	1nF	keramisk
R8	1k Ω		R26	39 Ω		D1	BPW 41	fotodiode
R9	1k Ω		R27	1k Ω		D2	BPW 41	fotodiode
R10	1k Ω		C1	1nF	keramisk	D3	7.5V	zener
R11	10k Ω		C2	1nF	keramisk	D4	7.5V	zener
R12	10k Ω		C3	220pF	keramisk	D5	OD8811	IR lysdiode
R13	12k Ω		C4	0	ikke monteret	U1	LM324	quad OP-AMP
R14	4.7k Ω		C5	0	ikke monteret	U2	LM324	quad OP-AMP
R15	10k Ω		C6	10nF	keramisk	U3	4053	triple analog multiplexer
R16	10k Ω		C7	10nF	keramisk	U4	7474	dual flip-flop
R17	4.7k Ω		C8	10nF	keramisk	U5	LM555	timer
R18	100k Ω		C9	10nF	keramisk			

Tabel 21.2: Komponentliste

21.3 Printlayout



(a) loddeside



(b) komponentplacering

Figur 21.2: Printlayout til liniesensorer

Kapitel 22

Fremstilling af Rungner

Dette kapitel rummer information om design og fremstilling af den tykfilm-baserede motordriver, jeg har givet navnet Rungner.

22.1 Elektronik

22.1.1 Kravspecifikation

I forbindelse med Cato, har jeg ud fra mit kendskab til motorer, strømforsyning, motorcontroller, og plads; stillet følgende krav til en motordriver:

Driftspænding 11 - 15 V
Driftsstrøm 15 A
Switch frekvens DC - 12 kHz

Switch tid max 500ns
Virkningsgrad min 85% ved maksimal last.
Størrelse max $20 \times 20 \times 5$ cm.

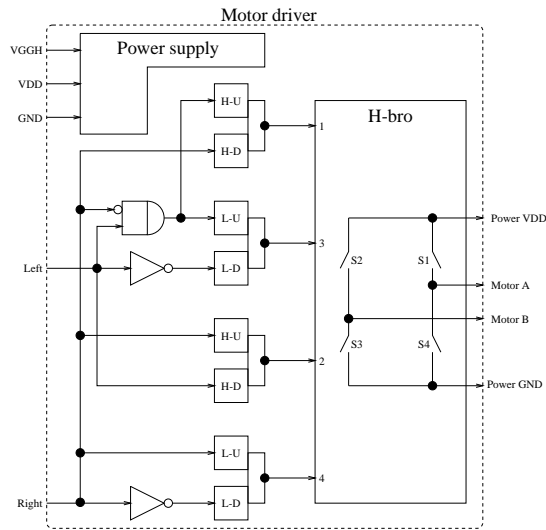
Jeg vil gerne udvikle en opstilling der kan bruges til andre formål end Cato, og jeg har et ønske om at afprøve anvendeligheden af tykfilm teknologi, til høj - effekt applikationer. Med mit kendskab til tilgængelige komponenter og teknologier, mener jeg at følgende ambitioner er — om ikke realistiske, så værd at sigte efter:

Driftspænding 11 - 50 V.
Driftsstrøm 30A.
Switch frekvens DC - 20kHz.

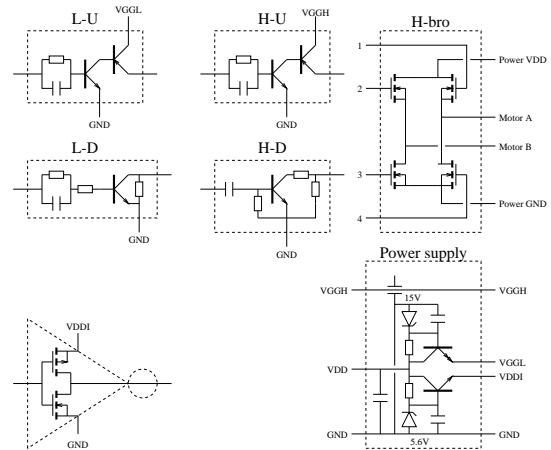
Switch tid < 200 ns.
Virkningsgrad $\geq 95\%$ ved maksimal last.
Størrelse $2'' \times 2''$ tykfilm.

22.1.2 Kredsløb

Figur 22.1 viser blokdiagrammet over Rungner, og figur 22.2 viser implementationen af de vigtigste blokke. Figur 22.3 viser hele kredsløbet.



Figur 22.1: Blokdiagram over Rungner



Figur 22.2: Implementation af blokke

Som det fremgår af figur 22.2, er selve H-broen opbygget vha. fire N-kanal MOS-FET transistorer. For at åbne helt, skal de to transistorer der er forbundet til Power VDD¹ have en gate spænding, der er min. $V_{GS,ON}$ højere end Power VDD, mens de to lave transistorer skal have en gatespænding på min. $V_{GS,ON}$ i forhold til Power GND.

For at opnå hurtige skiftetider, styres hver af H-broens gates, af et push-pull kredsløb, der på blokdiagrammet er vist som separate push-up og pull-down blokke. Der anvendes forskellige kredsløb til de høje og lave transistorer. push/pull blokkene er navngivet således:

H-U High transistor push Up
H-D High transistor pull Down

L-U Low transistor push Up
L-D Low transistor pull Down

Bemærk at begge typer pull-down kredsløb indeholder en pull-down modstand, der altid vil trække de respektive gates lave, hvis push-up kredsløbet ikke er aktivt. Bemærk også at H-D kredsløbene er AC-koblede, og derfor kun — aktivt — vil trække de pågældende gates lave i en puls efter kredsløbet aktiveres, hvorefter gaten holdes lav af pulldown transistoren. (Pulse pulldown)

Tabel 22.1 viser hvordan de enkelte gates i H-broen styres, som funktion af de to indgange: Left og Right.

Left	Right	Gate 1	Gate 2	Gate 3	Gate 4
0	0	Resistor pulldown	Resistor pulldown	Active pulldown	Active pulldown
0	1	Pulse pulldown	Active pushup	Active pulldown	Active pushup
1	0	Active pushup	Pulse pulldown	Active pushup	Active pulldown
1	1	Pulse pulldown	Active pushup (+ Pulse pulldown)	Resistor pulldown	Active pushup

Tabel 22.1: Styring af H-bro

Den sidste tilstand er ulovlig, men en enkel AND funktion, implementeret vha. T23 og R17 Beskytter imod den., ved at prioritere Right højere end Left

¹De høje transistorer

22.1.3 H-U

Disse kredsløb er opbygget som common emitter udgangstrin (T13/ T16), styret af et common emitter indgangstrin (T14, R4, C3 / T15, R13, og C7), som gør indgangsniveauet TTL/CMOS kompartibelt, og optimerer skiftetiderne.

H-U kredsløbet skal spændingsforsynes med en spænding der er 11 - 15 V højere end Power VDD.

22.1.4 H-D

Disse kredsløb er opbygget som AC-koblede common emitter udgangstrin (T12 R7 C5 / T17 R10 C6) med seriekoblede strømbegrænsende modstande (R8 / R9). Desuden sidder der faste pulldown modstande (R16 / R15).

For at forhindre V_{GS} for den pågældende power MOS-FET i at kommer under -20 V, er der indsat zenerdioder (D8 / D7), der forhindrer V_{GS} i at komme uden for intervallet: [-0.7;18] V. (R8 / R9) begrænser strømmen gennem diode såvel som transistor, så max værdierne ikke overskides. AC-koblingen af kredsløbet forhindrer at langtids effektafsættelsen i transistor, diode, og modstand overskrides.

De faste pulldown modstande er ret store (ca. 400 k Ω), hvilket betyder at den tilhørende power MOS-FET ikke slukkes med det samme dens H-U driver deaktiveres. De anvendte power MOS-FET's har $C_{GS} \simeq 1nF$, så systemet har en tidskonstant på ca. 400 μs . Med H-D driveren aktiv er tidskonstanten til sammenligning ca. 100ns. Kredsløbet er med vilje designet sådant, for at få de høje transistorer til at forblive åbne, efter at de lave transistorer er lukket — med mindre udgangen skifter polaritet.

22.1.5 L-U

Disse kredsløb er stort set identiske med H-U kredsløbene, og til en vis grad sammenbygget med dem.

Transistorerne i udgangstrinnet (T11 / T18) har lavere V_{CE} og højere $I_{C,peak}$ end (T13 / T16), men transistorerne i indgangstrinnene (T10 / T19) er af samme type som (T14 / T15).

H-D kredsløbene deler basiskredsløb (R4 C3 / R13 C7) med HU kredsløbene.

22.1.6 L-D

Disse kredsløb er opbygget som common emitter udgangstrin (T9 / T20). Baiskredsløbene (R5 C4 / R12 C6) optimerer switchtiderne, mens (R6 / R11) forsinket switch tidspunktet. Forsinkelsen er nødvendig for at forhindre L-U og L-D kredsløbene i at være tændt samtidigt.

22.1.7 inverter

For at sikre hurtige switch-tider for L-D kredsløbene, er det nødvendigt at anvende en inverter, der ikke blot er hurtig, men som også har et lavimpedant udgangstrin.

For at imødekomme begge krav, og samtidigt spare på pladsen / komponentantallet, har jeg valgt at implementere inverterne som C-MOS invertere, vha småsignal MOS-FET transistorer. (T7 T8 / T21 T22)

22.1.8 AND-gate

Den AND-gate der er vist på blokdiagrammet, skal forhinde at alle fire transistorer i H-broen tændes samtidigt, hvis begge indgange ved et uheld aktiveres samtidigt.

AND-gaten er implementeret som et common emitter udgangstrin (T23 og R17), der forhindrer at H-U (left) og L-U (left) aktiveres hvis Right indgangen er aktiveret.

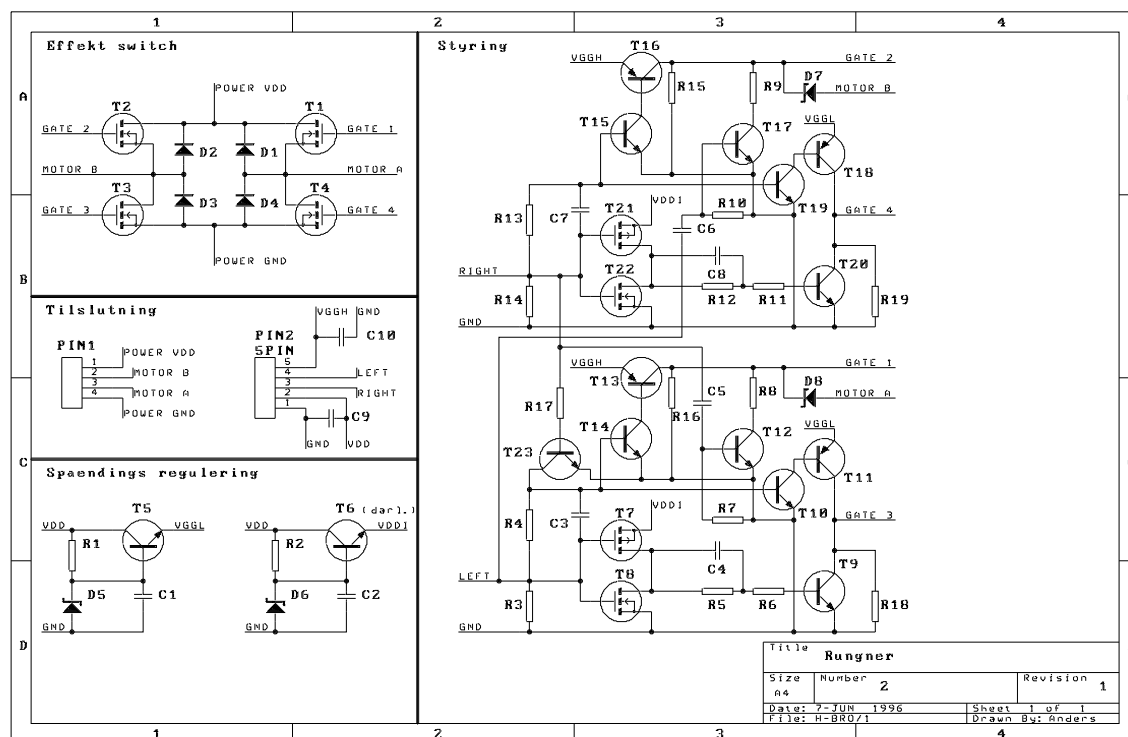
22.1.9 Power supply

Rungner skal kunne bruges med en forsyningsspænding på 11 - 50 V, og det er nødvendigt at anvende to spændingsregulatorer.

Til at forsyne L-U trinnene, skal der bruges en spænding på 10.7 - 20 V. (T5, R1, D5, og C1) realiserer en spændingsregulator der giver 13.5 - 14V ud, ved indgangsspændinger over 15V. Regulatorens spænding er valgt så lavt i intervallet for at mindske sandsynligheden for at støj og transienter på forsyningsspændingen skal få udgangsspændingen til at overskride $V_{GS,max}$. Da T5 har meget høj strømforstærkning ($H_{FE} > 10000$) kan Rungner forsynes med spændinger helt ned til 12V, uden problemer. Hvis forsyningsspændingen til Rungner imidlertid, ligger mellem 11 og 12 V, er det nødvendigt at kortslutte emitter og collector på T5.

C-MOS inverterene skal forsynes med ca. 5V. (T6, D6, R2, og C2) realiserer en spændingsregulator der giver 4.9 - 5.1 V ud, ved forsyningsspændinger på over 6V.

22.1.10 Diagram mv.



Figur 22.3: Diagram over Rungner

T1	IRF1010S	N-ch. power MOS-FET	D1	ES3B	Ultrafast recovery rectifier
T2	IRF1010S	N-ch. power MOS-FET	D2	ES3B	Ultrafast recovery rectifier
T3	IRF1010S	N-ch. power MOS-FET	D3	ES3B	Ultrafast recovery rectifier
T4	IRF1010S	N-ch. power MOS-FET	D4	ES3B	Ultrafast recovery rectifier
T5	BCV 49	NPN darl.	D5	BCX 15V0	15V Zener diode
T6	BC 817	NPN small signal	D6	BCX 5V6	5.6V Zener diode
T7	BSS 84	P-ch. small signal MOS-FET	D7	BCX 16V0	16V Zener diode
T8	BSS 138	N-ch. small signal MOS-FET	D8	BCX 16V0	16V Zener diode
T9	BC 817	NPN small signal	C1	15nF	Keramisk kondensator
T10	BC 846	NPN small signal	C2	15nF	Keramisk kondensator
T11	BC 807	PNP small signal	C3	68pF	Keramisk kondensator
T12	BC 846	NPN small signal	C4	4.7nF	Keramisk kondensator
T13	BC 856	PNP small signal	C5	10nF	Keramisk kondensator
T14	BC 846	NPN small signal	C6	10nF	Keramisk kondensator
T15	BC 846	NPN small signal	C7	68pF	Keramisk kondensator
T16	BC 856	PNP small signal	C8	4.7nF	Keramisk kondensator
T17	BC 846	NPN small signal	C9	100nF	Keramisk kondensator
T18	BC 807	PNP small signal	C10	100nF	Keramisk kondensator
T19	BC 846	NPN small signal			
T20	BC 817	NPN small signal	R4	390k Ω	1/4W, 5% modstand
T21	BSS 84	P-ch. small signal MOS-FET	R13	390k Ω	1/4W, 5% modstand
T22	BSS 138	N-ch. small signal MOS-FET	R15	390k Ω	1/4W, 5% modstand
T23	BC 846	NPN small signal	R16	390k Ω	1/4W, 5% modstand

Tabel 22.2: Diskrete komponenter

Modstand	Min. værdi [k Ω]	Max. værdi [k Ω]	effekt [mW]
R1	1.5	100	600
R2	1.5	10	350
R3	0.5	1.5	50
R5	2.5	5	< 1
R6	0.05	0.15	< 1
R7	5	10	< 1
R8	0.05	0.15	250
R9	0.05	0.15	250
R10	5	10	< 1
R11	0.05	0.15	< 1
R12	2.5	5	< 1
R14	0.5	1.5	50
R17	0.5	1.5	50
R18	2	50	100
R19	2	50	100

Tabel 22.3: Tykfilm modstande

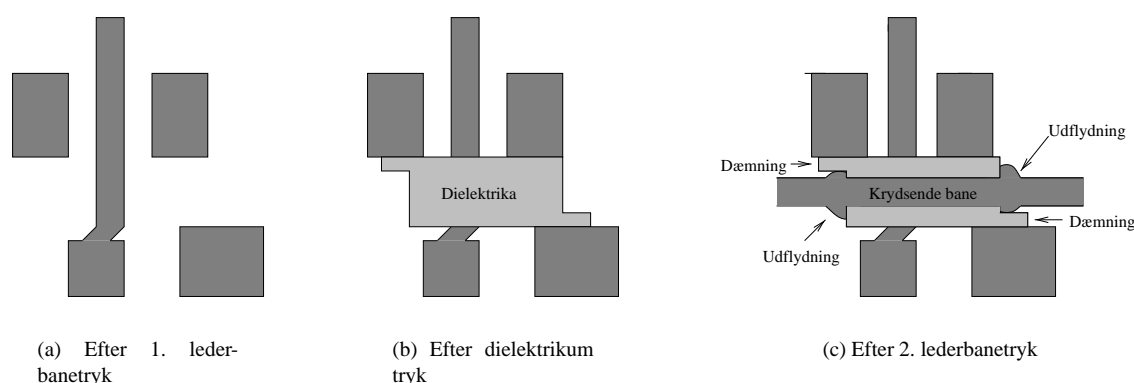
22.2 Fremstilling

22.2.1 Design af trykkemønstre

Kredsløbet designes med print-layout programmet PCAD, der er beregnet til multilags-print, og dermed er brugbart til tykfilmdesign.

Placeringen af effektkomponenter, og lederbanerne til dem er kritisk for designet, og behandles som det første. For at fremstille et kompakt kredsløb med brede, korte lederbaner, har jeg valgt transistorer med hus-typen: SMD TO-220, hvor det er muligt at føre en bred bane under transistoren, uden at danne forbindelse til transistorens elektroder.

Efter at have placeret effektkomponenterne, designses resten af kredsløbet rundt om dem. Lederbaner til småsignaldelen designses som et normalt to-lags print, og alle steder hvor krydsninger er nødvendige placeres et område med dielektrikum. Mange steder er dielektrikaet aktivt brugt som dæmning, for at kontrollere den udflydning af lederbanepasta, der forekommer når der trykkes på et ujævnt underlag (figur 22.4).



Figur 22.4: Brug af dielektrika til dæmning

Alle modstande i kredsløbet, undtaget 4, ligger i området 100Ω til $10k\Omega$, og ved at anvende $1k\Omega/\square$ pasta, kan de alle fremstilles i en enkelt trykning. Af de resterende fire modstande, stiller to af dem større krav til tolerance end der normalt kan opnås uden trimning. Det kan ikke betale sig at lave en trykning for to modstandes skyld, og de sidste fire modstande implementeres derfor alle som diskrete komponenter. Et enkelt sted har det, af pladshensyn, været nødvendigt at lægge en modstand oven på et lag dielektrikum, så der skal anvendes et dielektrikum der er egnet som substrat for modstande.

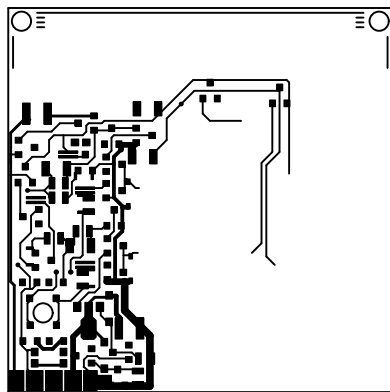
Modstand	Min. værdi [k Ω]	Max. værdi [k Ω]	effekt [mW]	pasta	længde [mm]	bredde [mm]	l/b \square	effekt [mW]
R1	1.5	100	600	Dupont 8031	2.5	0.5	5	780
R2	1.5	10	350	Dupont 8031	2.5	0.5	5	780
R3	0.5	1.5	50	Dupont 8031	0.5	0.5	1	150
R5	2.5	5	< 1	Dupont 8031	1.9	0.51	3.7	600
R6	0.05	0.15	< 1	Dupont 8031	0.3	2.5	0.12	460
R7	5	10	< 1	Dupont 8031	1.9	0.25	7.6	290
R8	0.05	0.15	250	Dupont 8031	0.3	2.5	0.12	460
R9	0.05	0.15	250	Dupont 8031	0.3	2.5	0.12	460
R10	5	10	< 1	Dupont 8031	1.9	0.25	7.6	290
R11	0.05	0.15	< 1	Dupont 8031	0.3	2.5	0.12	460
R12	2.5	5	< 1	Dupont 8031	1.9	0.51	3.7	600
R14	0.5	1.5	50	Dupont 8031	0.5	0.5	1	150
R17	0.5	1.5	50	Dupont 8031	0.5	0.5	1	150
R18	2	50	100	Dupont 8031	2.5	0.25	10	390
R19	2	50	100	Dupont 8031	2.5	0.25	10	390

Tabel 22.4: Modstande

Når de tykke sølvbaner er trykt, er det pga. højdeforskellene ikke muligt at foretage flere trykninger. I stedet for at bruge en normal *resistor overglaze* som loddestopmaske, påtrykkes loddestopmasken derfor med et almindeligt dielektrikum, før trykning af sølvbanerne. Dielektrikaet trykkes ikke oven på modstandene, der må efterlades ubeskyttede, og effektkomponenterne må undvære loddestopmaske. I en prototype har den manglende beskyttelse af modstandene ingen betydning, og i industriel sammenhæng kan beskyttelsen af modstandene varetages ved f.eks. indstøbing. Den manglende loddestopmaske på sølvbanerne har kun kosmetisk betydning, og hvis kredsløbene behandles med loddelak e.lign. bliver det ubeskyttede sølv ikke anløbet og sort.

22.2.2 AgPd 1

I det første tryk fremstilles flertallet af lederbaner og loddeøer til småsignal delen af Rungner. Figur 22.5 viser trykmasken retvendt i 1:1. Trykretningen er fra venstre mod højre.



Figur 22.5: AgPd 1

På den øverste del af dette tryk findes diverse styrelinier og cirkler. Desuden findes en styrecirkel i 3. kvadrant. De lod- og vandrette styrelinier, bruges sammen med de firkantede loddeøer nederst til venstre til at positionere dette 1. tryk på substratet. De korte vandrette styrelinier og cirklerne, bruges til at positionere efterfølgende tryk i forhold til dette første tryk.

efter dette tryk tørres og brændes substratet.

Pasta Hereaus C 1214 AgPd

Maske 100T

Rakelhastighed 0.085 m/s

Rakletryk ca. 15 N

22.2.3 Kryds 1

I det 2. og 3. tryk fremstilles isolerende områder, der tillader efterfølgende lederbanetryk at krydse det første lederbanetryk, elektrisk isoleret fra dette. Visse steder bruges dette isolerende tryk, som en form for ”dæmning” der forhindrer efterfølgende tryk i at flyde sammen med andre tryk.

De 3 styrecirkler i dette tryk skal være koncentriske med styrecirklerne fra 1. tryk.

Der trykkes to gange med masken vist i figur 22.6, trykretningen er fra venstre mod højre. Efter hver trykning tørres og brændes substratet. Selv om det



Figur 22.6: kryds 1

er tidskrævende er det dobbelte tryk og brænding nødvendig, idet indlejrede støvpartikler i det første tryk, brændes væk i første brænding, og efterlader mikroskopiske huller, der udfyldes i andet tryk.

Pasta Hereaus IP 9117 Multilayer dielectric.

Maske 100T

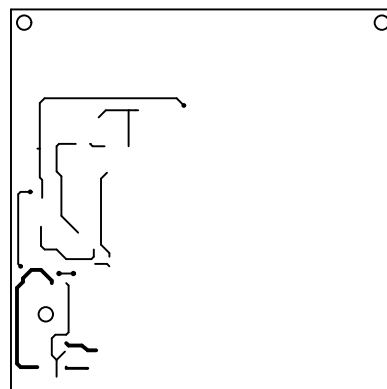
Rakelhastighed 0.085 m/s

Rakletryk ca. 15 N

22.2.4 AgPd 2

I det 4. tryk fremstilles de resterende lederbaner til småsignal delen af Rungner. Alle baner fremstillet i denne trykning skal passere over et område med dielektrikum, og skal danne elektrisk forbindelse til baner fra 1. tryk.

De 3 styrecirkler i dette tryk skal være koncentriske med styrecirklerne fra 1. tryk. Trykkemaskinen skal indstilles meget omhyggeligt for at sikre at der ikke opstår kortslutninger / afbrydelser i kredsløbet.



Figur 22.7: AgPd 2

Der trykkes med masken vist i figur 22.7. Trykretningen er fra venstre mod højre. Efter trykket tørres substratet.

Pasta Hereaus C 1214 AgPd.

Maske 100T

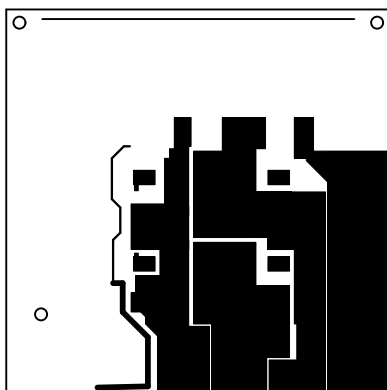
Rakelhastighed 0.085 m/s

Rakletryk ca. 15 N

22.2.5 Ag 1

I det 5. tryk Fremstilles lederbaner og loddeøer til effekt delen af Rungner. På trods af potentielle problemer med sølv-migration, anvendes der rent sølv, for at mindske modstanden i lederbanerne.

Den vandrette styrelinie i dette tryk skal passe sammen med det øverste sæt korte styrelinier fra 1. tryk, og de 3 styrecirkler skal være koncentriske med styrecirklerne fra 1. tryk.



Figur 22.8: Ag 1

Der trykkes med masken vist i figur 22.8. Trykkeretningen er fra venstre mod højre. Efter trykningen tørres og brændes substratet.

Pasta Hereaus Ag

Maske 100T

Rakelhastighed 0.085 m/s

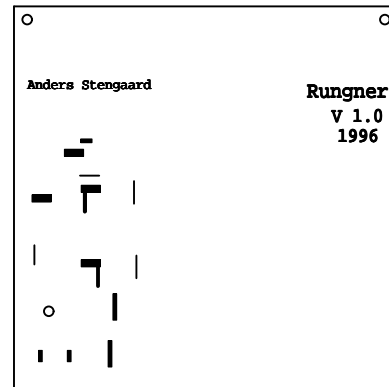
Rakletryk ca. 15 N

22.2.6 R1k

I det 6. tryk fremstilles samtlige tykfilm baserede modstande. Modstandende er dimensioneret med sideforhold (*aspect ratio*) fra ca. 0.1 til 10, for at fremstille modstande med værdier i et bredt interval, med

kun eet tryk. Dette bevirker imidlertid at de største modstande får store tolerancer.

De 3 styrecirkler i dette tryk skal være koncentriske med styrecirklerne fra 1. tryk.



Figur 22.9: Modstande (1 kΩ / □)

Der trykkes med masken vist i figur 22.9.

Trykkeretningen er fra venstre mod højre. Efter trykningen tørres substratet.

Pasta DuPont 8031 950 Ω / □

Maske 100T

Rakelhastighed 0.085 m/s

Rakletryk ca. 15 N

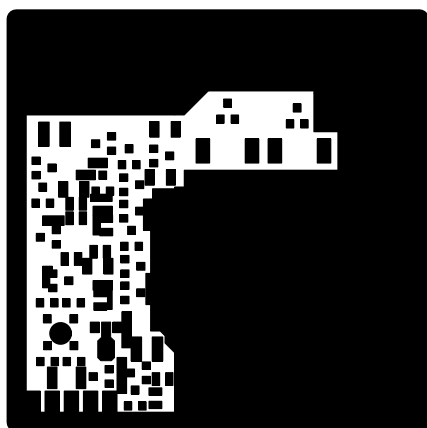
22.2.7 Lodde- stopmaske

På grund af det tykke sølvlag der lægges på senere i processen, er der ikke muligt at afslutte fremstillingen af substratet med *resistor overglaze* idet det tykke lederbaner umuliggør et jævnt silketryk. Overglazens funktion er dels at beskytte modstandende under trimning, og mod miljøpåvirkninger; og dels at fungere som lodde-stopmaske. Da modstandende ikke skal trimmes, og miljøbeskyttelsen kan varetages på andre måder, er det kun nødvendigt at fremstille en lodde-stopmaske.

Det er ikke nødvendigt med loddestopmaske til komponenterne i effekt delen, da deres størrelse gør at loddeøerne er placeret med så stor afstand at sammenflydning af loddetin ikke er aktuel. Geometrien af loddeøerne til effektkomponenterne er endvidere sådan at loddetinnets overfladespænding (i smeltet tilstand) vil fastholde komponenterne på deres plads under *reflow* lodning.

Følgelig anvendes en loddestopmaske der kun dækker områderne uden for de tykke sølvbaner.

Der er kun en styrecirkel i dette tryk, så der må også sigtes efter loddeøer ved indstilling af trykkemaskinen.



Figur 22.10: Lodde-stopmaske (**negativ**)

Til dette 7. tryk, anvendes masken vist i figur 22.10.

I princippet kan alle typer dielektrikum-pasta anvendes til lodde-stopmasken. Den jeg anvender er valgt fordi den er halvgennemsigtig (så man kan se de underliggende lag), grøn (en farve der normalt anvendes til dæklag i elektronik), og ikke bliver brugt til andet længere.

Trykkeretningen er fra venstre mod højre.

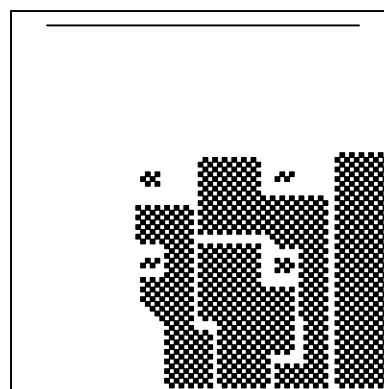
Efter trykningen tørres substratet.

Pasta ESL crossover dielectric.

Maske 100T

Rakelhastighed 0.085 m/s

Rakletryk ca. 15 N



Figur 22.11: Ag 2

Disse lag skal trykkes med så lavt rakeltryk som muligt, for at forhindre udflydninger med resulterende kortslutninger. Det kan være nødvendigt at føre substratet ind i silketryksmaskinen to gange ved hvert tryk, for at sikre at hullerne fra forrige tryk fyldes helt op med pasta.

Hvert tryk forøger lagtykkelsen med ca. 20 μm (før brænding), og substratet tørres mellem hvert tryk.

Der trykkes 8 gange med denne maske.

Efter tørringen af det sidste tryk, brændes substratet.

Pasta Hereaus Ag.

Maske 100T

Rakelhastighed 0.085 m/s

Rakletryk ca. 10 N

22.2.8 Ag 2

Formålet med disse 8. til 8+2n. tryk er at forøge tykkelsen af lederbanerne i effektdelen af Rungner.

Der trykkes efter masken vist i figur 22.11. Trykkeretningen er fra venstre mod højre.

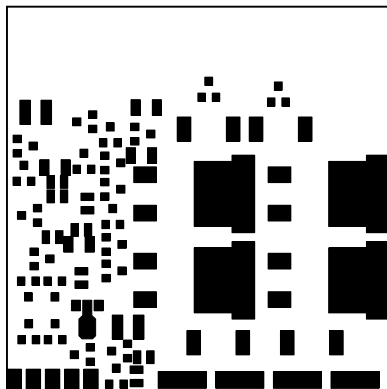
Som tidligere nævnt består disse tryk af en serie trykninger, med den samme — skakternede — maske, men ved hvert andet tryk forskydes masken 25 mils / 635 μm *nedad*, svarende til bredden af ternene.

Styrelinien øverst i dette lag, skal falde skiftevis sammen med det næstnederste, og det nederste sæt markeringslinier i AgPd 1 laget.

22.2.9 Loddetin

Jeg silketrykker en loddepasta på substratet, frem for at anvende dype-fortining.

For at forhindre at silketryksmaskinen forurenes med bly, skal loddetin trykkes på med en mere besværlig og unøjagtig hånd-trykke metode, men fordelene frem for dypefortining overskygger langt dette besvær.



Figur 22.12: Loddetin

Der trykkes med masken vist i figur 22.12. Trykkeretningen er fra op til ned.

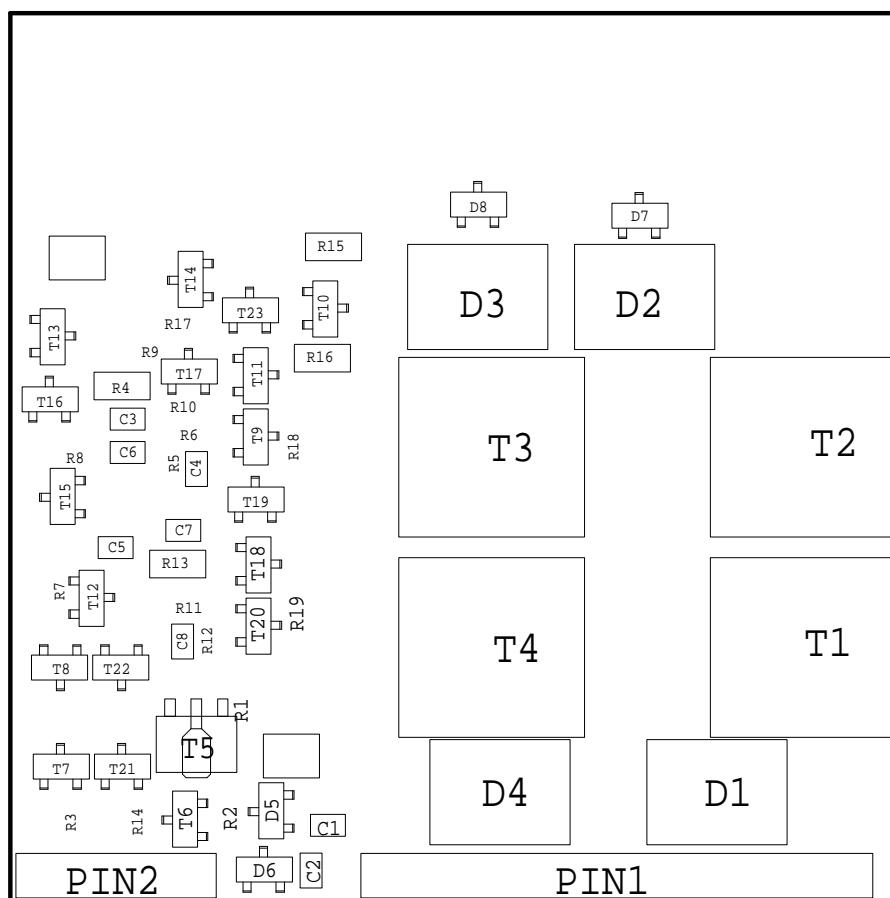
Pasta SbPbAg loddepasta.

Maske 66T

Rakelhastighed Manuel trykning.

Rakletryk Manuel trykning.

22.2.10 Komponentplacering



Placeringen af tykfilmmodstande er vist uden omrids. Se evt figur 22.9
De to rektangulære komponenter uden påskrift er C9 (nederst i midten) og C10 (øverst til venstre).

Figur 22.13: Komponentplacering

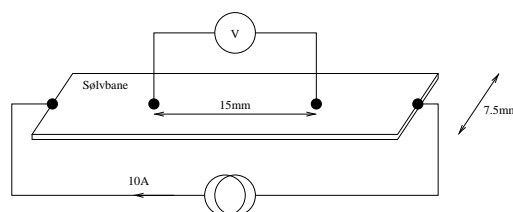
22.3 Tykfilmtest

Efter fremstillingen, undersøges tykfilmkredsløbene, med henblik på at klarlægge brugbarheden af skakternmetoden, og godkende de trykte modstande.

22.3.1 Tykke lederbaner

Tykkelsen af de tykke lederbaner måles med mikrometerskrue. Visuel inspektion med og uden mikroskop, viser hverken revner, sprækker, blæredannelse, misfarvning, ujævnheder eller andre problemer med de tykke lederbaner.

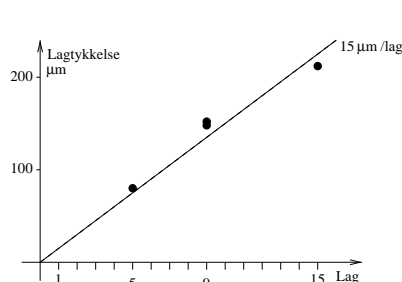
Flademodstanden måles for lederbanen yderst til højre, idet det er den eneste der har en regelmæssig bredde (7.5mm). På de substrater der testes, sendes en strøm på 10A gennem den yderste sølvbane, samtidigt med at spændingsfaldet over en 15mm lang strækning midt på banen måles med et voltmeter. Strømtætheden midt på banen antages at være homogen, og flademodstanden (ved stuetemperatur) bestemmes som:



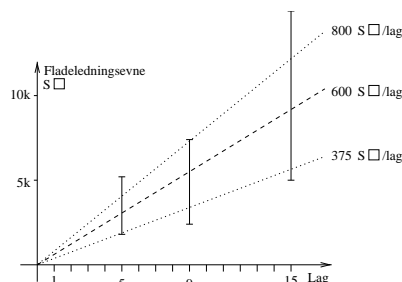
Figur 22.14: Måling af flademodstand

$$R_f = \frac{V \times 7.5mm}{10A \times 15mm} \quad (22.1)$$

Lag:	Tykkelse	Δ	Flademodstand	Δ	Ledningsevne
9	150 μm	$\pm 5 \mu m$	0.2 m Ω/\square	$\pm 50\%$	5 kS \square
9	150 μm	$\pm 5 \mu m$	0.2 m Ω/\square	$\pm 50\%$	5 kS \square
9	140 μm	$\pm 5 \mu m$	0.2 m Ω/\square	$\pm 50\%$	5 kS \square
5	80 μm	$\pm 5 \mu m$	0.3 m Ω/\square	$\pm 50\%$	3.5 kS \square
15	210 μm	$\pm 5 \mu m$	0.1 m Ω/\square	$\pm 50\%$	10 kS \square



(a) Tykkelse



(b) Fladeledningsevne

Figur 22.15: Målinger på tykke lederbaner

Tykkelsesmålingen viser at tykkelsen vokser lineært med antallet af lag, og selv om målingen af flademodstanden ikke er særligt nøjagtig, er der rimeligt belæg for at fladeledningsevnen ligeledes vokser lineært med antallet af lag.

For hvert lag vokser tykkelsen ca. 15 μm , og fladeledningsevnen vokser 375 ... 800 S \square , svarende til at hvert lag har en flademodstand på 1.3 ... 2.7 m Ω/\square .

22.3.2 Tykfilmmodstande

Med et ohmmeter testes det at de fremstillede tykfilmmodstande ligger inden for de ønskede rammer:

Modstande	længde mm	bredde mm	l/b □	måledata Ω	måledata Ω	måledata Ω	middel Ω	std. afv. %	max. afv. %
R1 ¹	2.5	0.5	5	6k6	6k2	6k0	6k3	5	6
R2	2.5	0.5	5	4k5	4k5	5k1	4k7	8	9
R3,14	0.5	0.5	1	740,680	760,730	940,820	780	12	21
R5,12	1.9	0.51	3.7	4k6,3k5	4k6,3k7	5k0,3k7	4k2	15	20
R6,8,9,11	0.3	2.5	0.12	72,79,76,78	77,86,74,83	82,86,85,82	80	7	10
R7	1.9	0.25	7.6	6k3	7k0	8k1	7k1	13	14
R10 ²	1.9	0.25	7.6	7k9	7k9	9k5	8k4	11	13
R17 ²	0.5	0.5	1	790	900	930	870	9	10
R18,19	2.5	0.25	10	15k,13k	15k,14k	19k, 13k	14k8	15	30

1) Trykt på dielektrikum. 2) Trykt på langs af trykkeretning.

Tabel 22.5: Måling af tykfilmmodstande

Selv om testen kun tager udgangspunkt i tre stikprøver, og må betragtes som overfladisk, afslører den vigtig information.

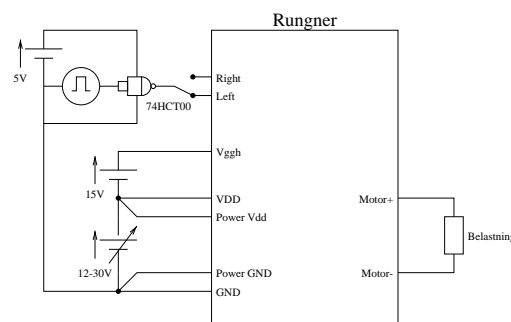
Alle modstande ligger inden for de ønskede intervaller, med afvigelser indenfor det acceptable. Det eneste potentielle problem er R5, der ligger ved yderkanten af det ønskede interval. En nøjere visuel inspektion afslører at R5 på alle substrater har en skæv kant, der gør den svagt trapezformet, og dermed nedsætter gennemsnitsbredden. Formentligt har trykkemasken ikke været vasket ordentligt ud. Teorien underbygges af at sporingen mellem par af identiske modstande er dårligere end gentagenøjagtigheden for tryk af enkeltstående modstande.

22.4 Kredsløbstest

Afprøvningen af Rungner foretages ved at anvende prøveopstillingen vist i figur 22.16.

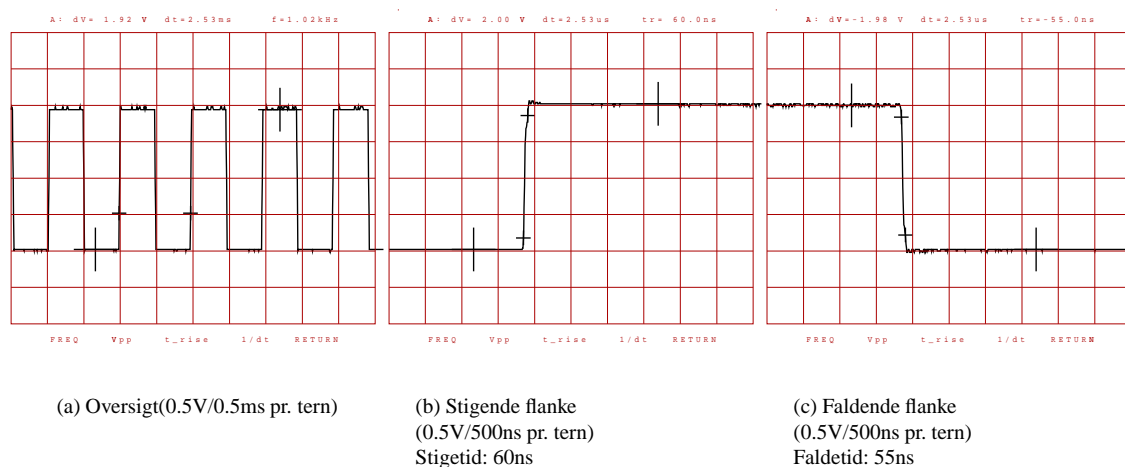
Rungner forsynes primært af en variabel laboratoriestrømforsyning, der kan levere 10A i spændingsområdet 0—30V. For at stabilisere spændingen, og absorbere *induktivt kickback* parallelforbinder forsyningen med en 10mF (10.000 μ F) kondensator. Forsyningen forbindes til effekt og signaldelen i en stjerne-konfiguration. Spændingen til de høje transistorer leveres af en 15V laboratorieforsyning koblet i serie med effektforsyningen.

Rungner styres af et signal fra et CMOS udgangstrin (74HCT00), der styres af en firkantgenerator. Udgangssignalet fra firkantgeneratoren er vist på figur 22.17



Figur 22.16: Måleopstilling

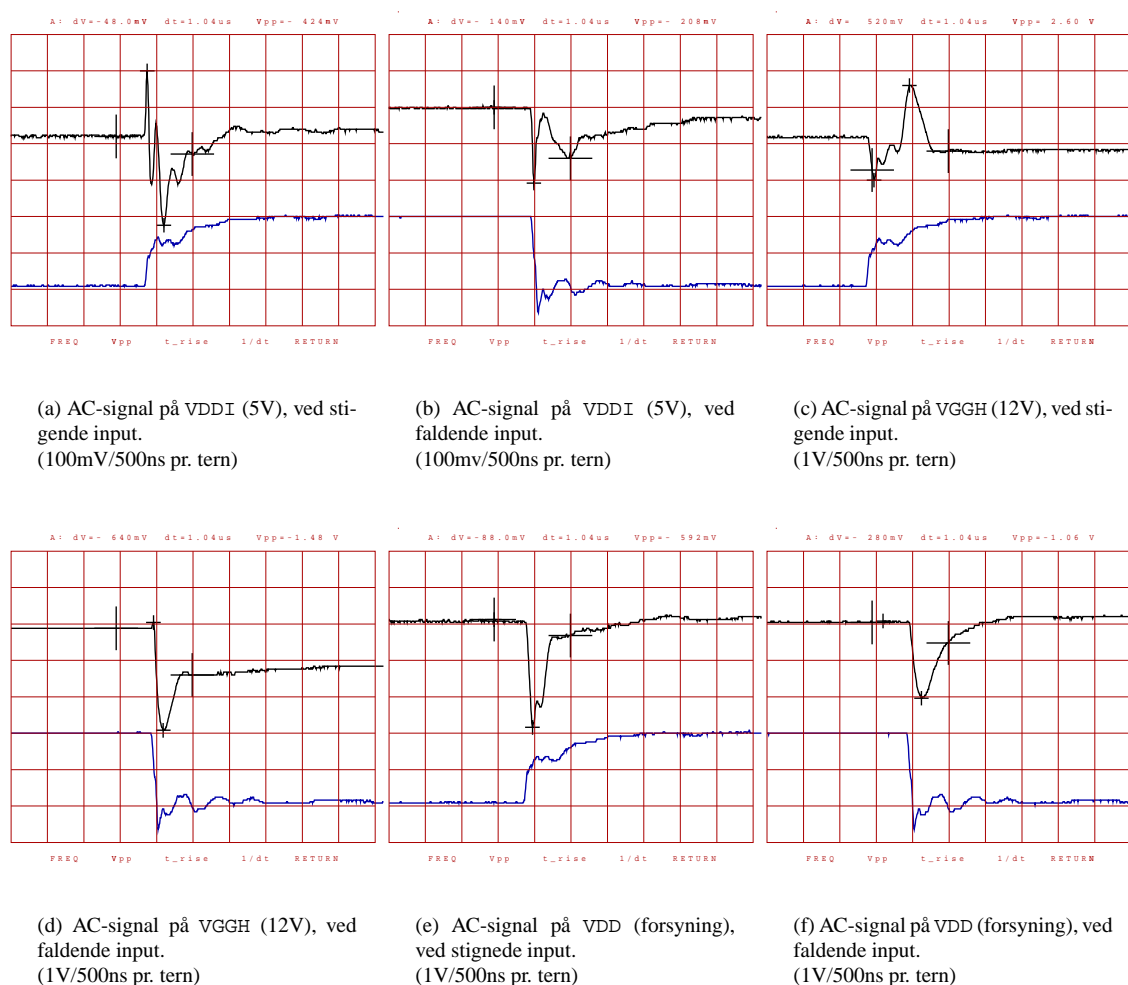
Undervejs i afprøvningen kobles Rungner til forskellige belastninger. Der anvendes en 47 Ω effektmotstand, en af Cato's motorer — ubelastet (ca. 1A forbrug) og belastet (ca. 8A forbrug). Der foretages målinger for input på både *right* hhv. *left* indgangen.



Figur 22.17: Signal fra firkantgenerator (0-2V, 1kHz)

22.4.1 Spændingsforsyning

Det testes af Rungners to interne spændingsregulatorer fungerer tilfredsstillende under alle driftsforhold. Dels testes det at de leverer en tilfredsstillende DC-spænding (5V hhv. 10.7—14V), men deres dynamiske opførsel testes også. Figur 22.18 viser eksempler på de dynamiske forhold til de tidspunkter hvor indgangssignalet skifter. Indgangssignalet ses nederst på alle 6 delfigurer (2V/tern).

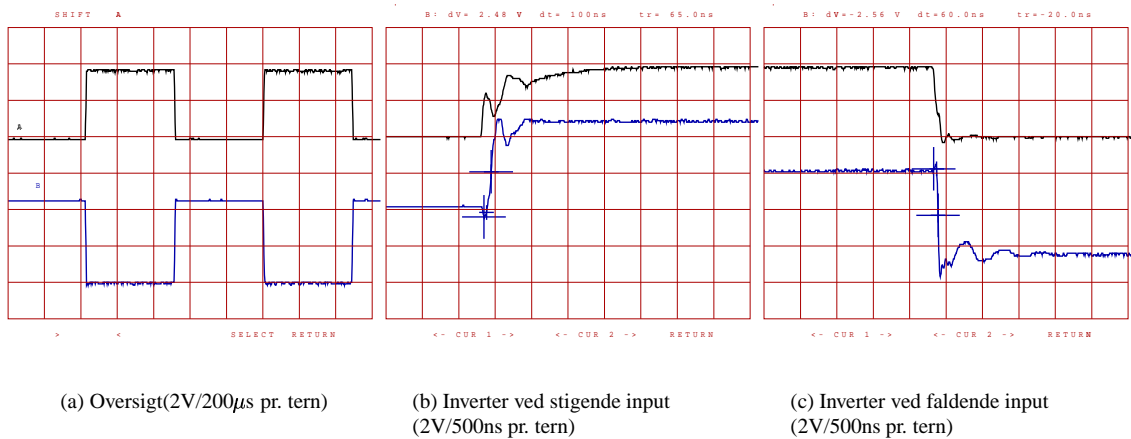


Figur 22.18: Dynamiske forhold ved spændingsforsyning

Alle spændinger holder sig inden for det acceptable, statisk, såvel som dynamisk. VGGH falder ganske vist 3V i 200ns efter indgangen skifter fra aktiv til inaktiv, men da de transistorer der skal styres vha, VGGH netop skal slukkes i dette tilfælde, udgør spændingsfaldet intet problem.

22.4.2 Invertere

Funktionen af de to CMOS invertere testes ved at sammenligne indgangssignalerne (left og right) med udgangen fra inverterne (drain på T8 / T22). Figur 22.19 viser sammenhængen mellem indgangssignalet (øverst) og udgangssignalet (nederst). På delfigur b og c er udgangssignalet vist inverteret for bedre sammenligning.



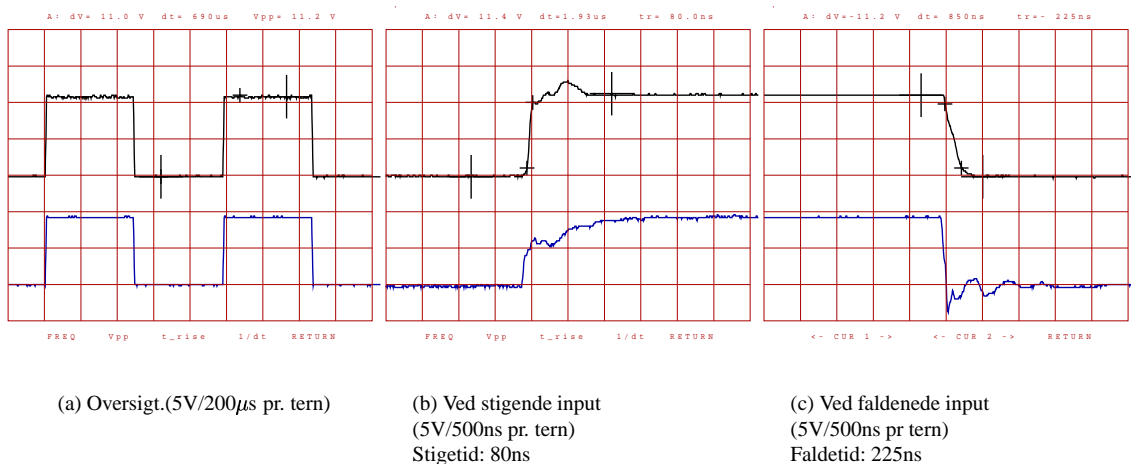
Figur 22.19: Test af inverter

Inverterne fungerer meget fint. De har et *propagation delay* på mindre end 100ns, og meget veldefinerede flanker, med stige- faldetider i størrelsesordenen: 50ns.

22.4.3 Gate spændinger

Gate spændingen på de fire effekttransistore måles under forskellige forhold.

Figur22.20 Viser *gate-source* spændingen på de lave transistorer (T3/T4). *gate-source* spændingerne er vist øverst, og indgangssignalet (2V/tern) nederst, som reference.

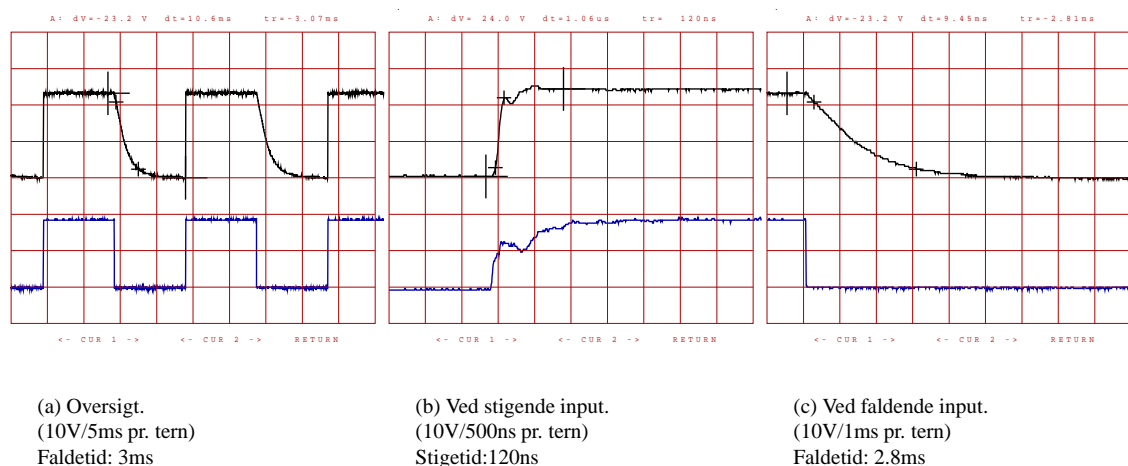


Figur 22.20: Gate-source spænding på T3/T4

Gate spændingen til T3/T4 vise at driver-trinnet til disse transistorer fungerer tilfredsstillende. Forsinkelsen mellem indgangssignal og gatespændinger er under 100ns. Ved opadgående flanker skifter gatespændingen fra ca. 0V til over 10V imponerende hurtigt. Ved nedadgående flanker ses en noget større skiftetid, der formentligt skyldes basismodstanden R6/R11.

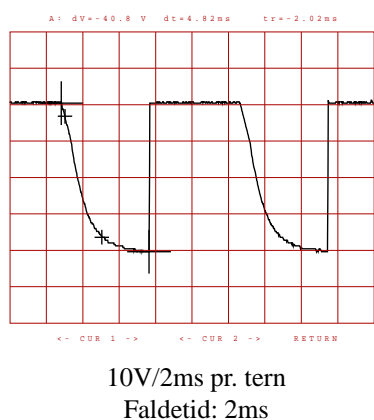
Da styrespændingen til de lave transistore kommer fra en regulator, er den uafhængig af forsyningsspændingen. Styrespændingen til de høje transistorer, skal være 10-20V større end forsyningsspændingen.

Figur 22.21 viser *gate* spændingen i forhold til GND på de høje transistorer (T1/T2), ved en forsyningsspænding på 12V. *gate* spændingen er vist øverst, og indgangssignalet (2V/tern) nederst, som reference.



Figur 22.21: Gate spænding på T1/T2 i forhold til GND, ved 12V forsyningsspænding

Figur 22.22 Viser *gate* spændingen på T1/T2 i forhold til GND, ved en forsyningsspænding på 30V.

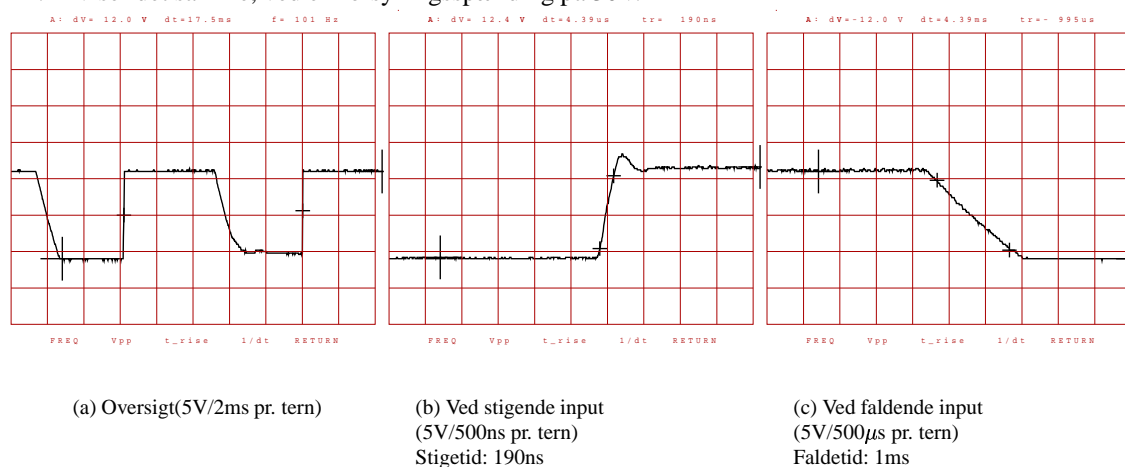


Figur 22.22: Gate spænding på T1/T2 i forhold til GND, ved 30V forsyningsspænding

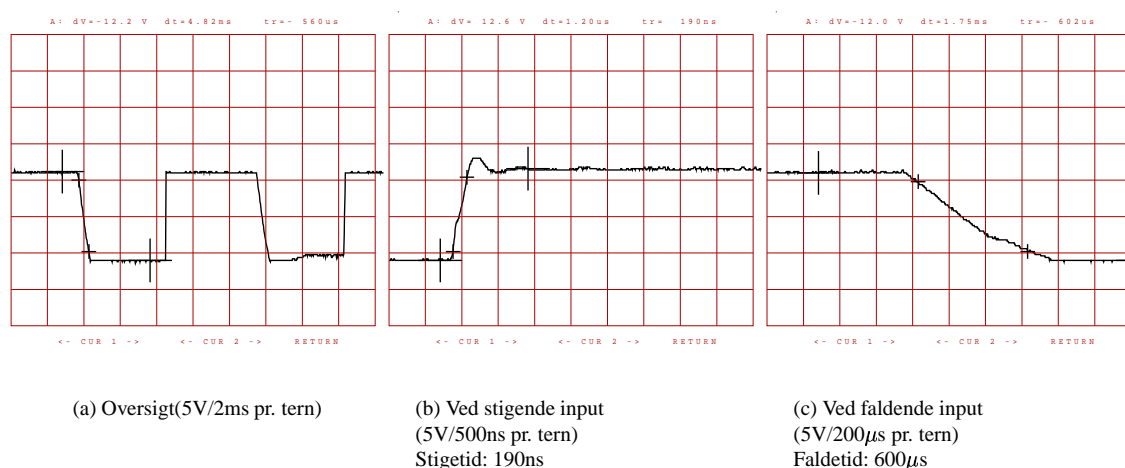
De to ovenstående figurer viser at drivertrinnet til T1/T2 overordnet fungerer som det skal. Ved opadgående flanker stiger spændingen hurtigt til det forventede niveau, og ved nedadgående flanker, falder spændingen med den karakteristiske RC afladningskurve, der forventes, idet den parasitiske *gate-source* capacitor, aflades til GND gennem R15/R16.

For at godkende drivertrinnet er det imidlertid nødvendigt at kigge på *gate-source* spændingen, da det er denne der styrer transistoren.

Figur 22.23 viser *gate-source* spændingen på transistor T1/T2, ved en forsyningsspænding på 12V. Figur 22.24 viser det samme, ved en forsyningsspænding på 30V.



Figur 22.23: Gate-source spænding på T1/T2, ved 12V forsyningsspænding



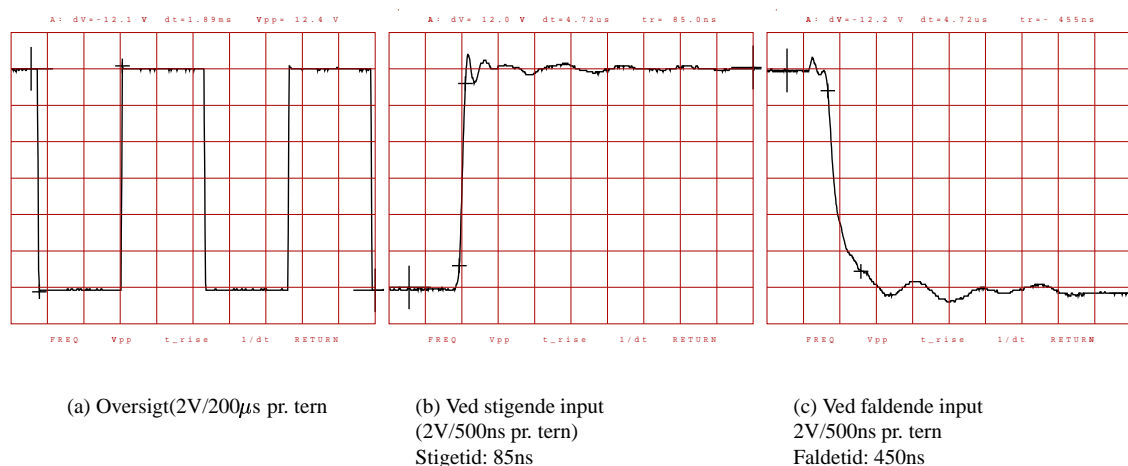
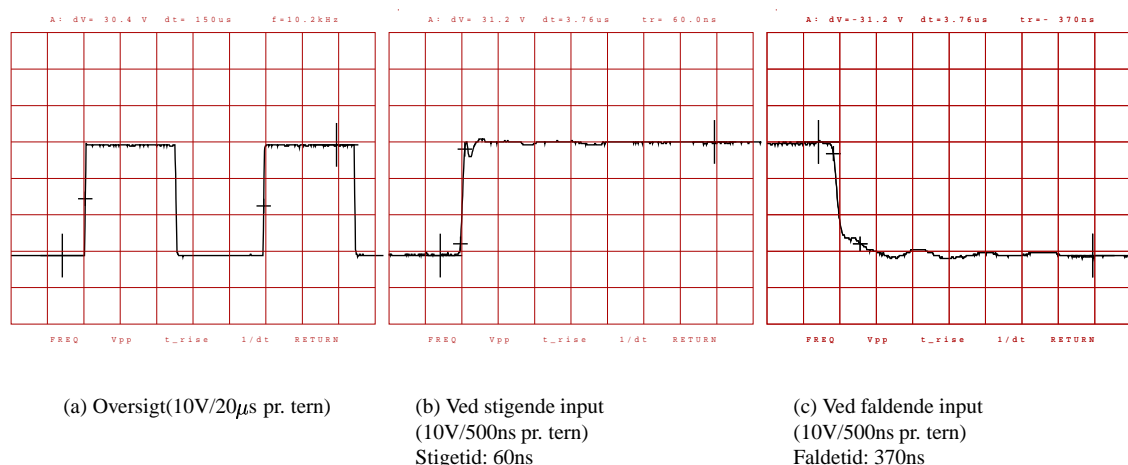
Figur 22.24: Gate-source spænding på T1/T2, ved 30V forsyningsspænding

Med de sidste målinger, kan det konkluderes at drivertrinnet til de høje transistorer fungerer som det skal. *gate-source* spændingen stiger til over 10V, i løbet af ca. 200ns, og falder tilbage igen i løbet af 1 ms. Spændingen holdes over 10V ($V_{gs,on}$) i ca. 100µs, efter indgangen deaktiveres. Derefter befinder transistorerne sig i princippet i det lineære område. Så længe der anvendes *switch* frekvenser på over 10kHz, vil de høje transistorer ikke nå at slukke mellem to aktive perioder. Så længe transistorerne ikke når at slukke mellem de aktive perioder, betyder det intet at tændetiden er længere end for T3/T4.

Hvis der skal anvendes *switch* frekvenser på under 10kHz, bør det overvejes at udskifte R15/R16, med nogle større værdier, for at lade transistorerne stå åbent tilsvarende længere.

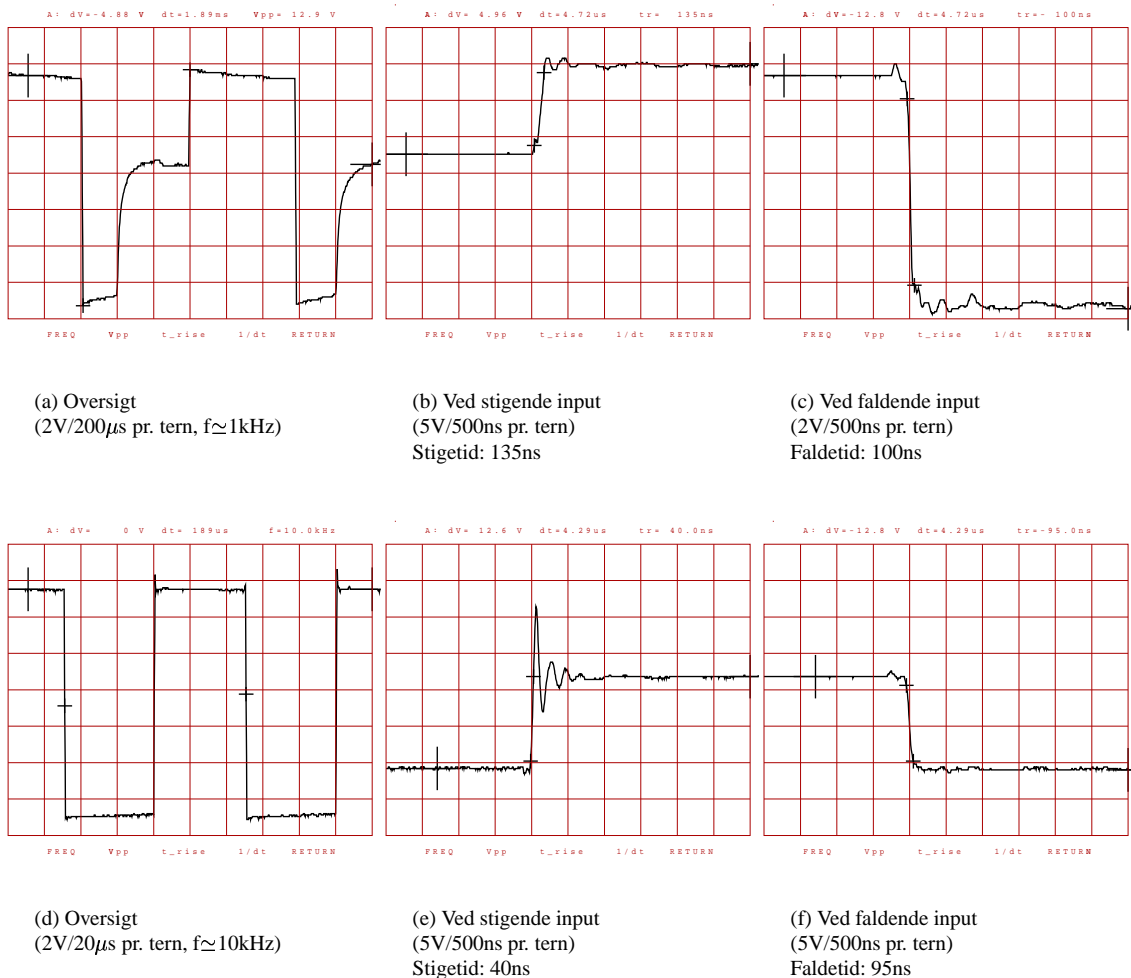
22.4.4 Udgang

For at undersøge udgangskarakteristikken, måles spændingen på Rungners udgang, under forskellige forhold. Som det første belastes Rungner beskedent med en 47Ω modstand, ved 12V og 30V.

Figur 22.25: Spænding over en 47 Ω belastning, ved 12V forsyning.Figur 22.26: Spænding over en 47 Ω belastning, ved 30V forsyning.

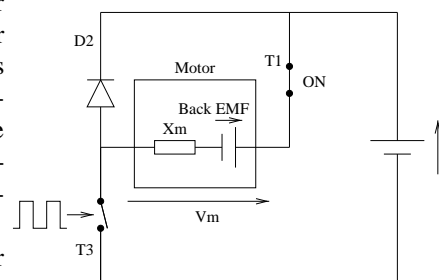
Målingerne viser det forventede resultat. Når indgangen er aktiv ligger hele forsyningsspændingen over modstanden, og ellers falder spændingen til 0V. Stigetiden er imponerende, mens faldetiden er lidt større end jeg havde ventet. Den lange faldetid, kan imidlertid føres tilbage til den tilsvarende lange faldetid for T3/T4's *gate-source* spænding. Selv om faldetiden er længere end ventet, finder jeg den ikke problematisk.

Som det næste belastes Rungner med en af motorene fra Cato. Motoren anvendes i en ubelastet konfiguration, hvor den ved en 12V forsyning, falder til ro ved et strømforbrug på ca. 1A, når den når sin tophastighed. Der testes ved en forsyningsspænding på 12V, med en switch-frekvens på 1kHz, og på 10kHz.



Figur 22.27: Spænding over ubelastet motor fra Cato, ved 12V forsyning.

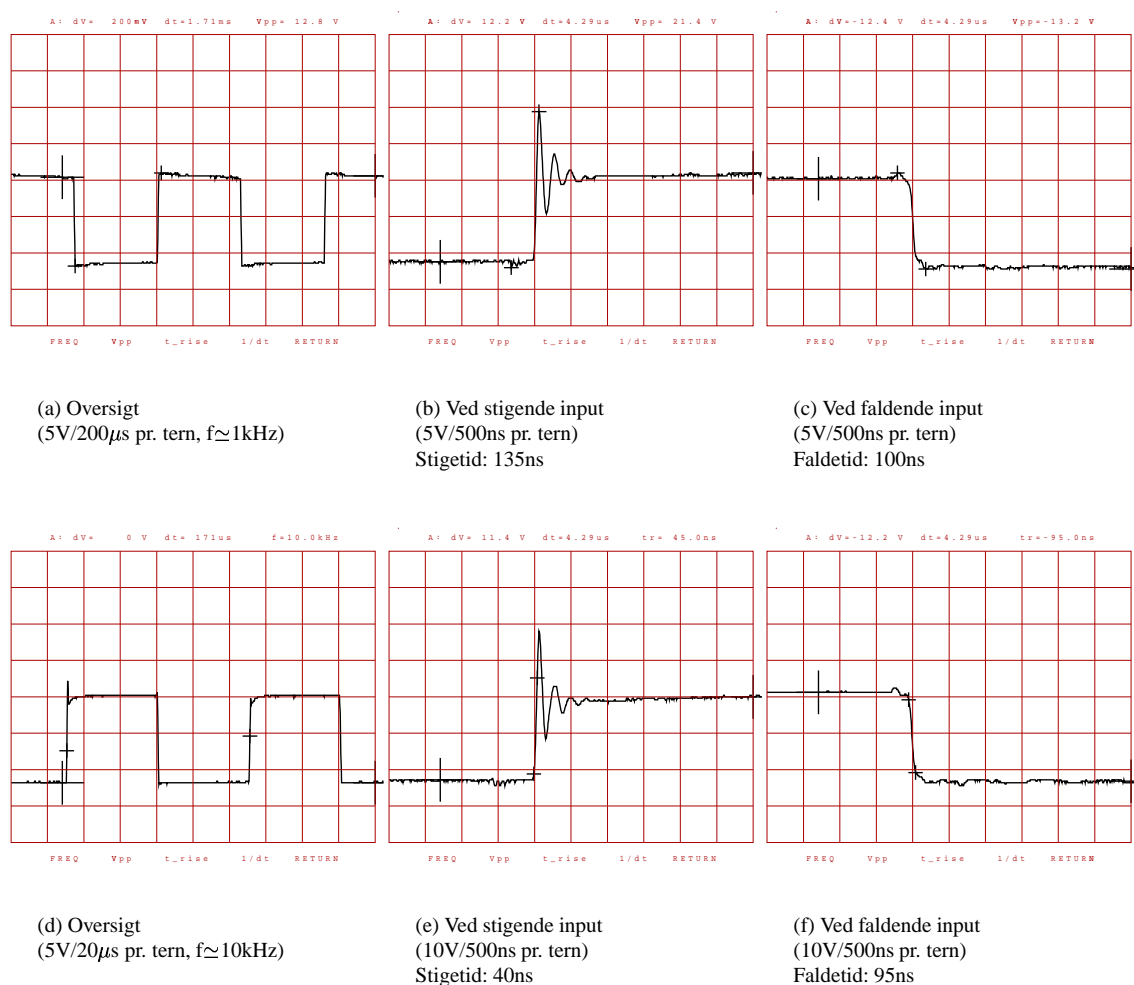
Figureerne forstås bedst ved at skele til ekvivalensdiagrammet for motor og H-bro, i figur 22.28. Når både T1 og T3 er tændt, er motoren påtrykt hele forsyningsspændingen, og der opbygges en strøm gennem motoren. Når T3 slukkes vil den, nu selvinducerede, strøm gennem motorviklingen, fortsætte med at løbe gennem D2 og T1, hvilket kræver en beskeden negativ motorspænding, for at overvinde D2's diodespænding. Denne spænding genereres af fluxændringen gennem motorens spole. Ifølge figur 22.27-a, dør den selvinducerede strøm ud efter ca. 200 μ s, hvorefter motorspændingen stabiliserer sig på den spænding der genereres af motorspolens bevægelse i motorens magnetfelt (back EMF).



Figur 22.28: Ekvivalensdiagram

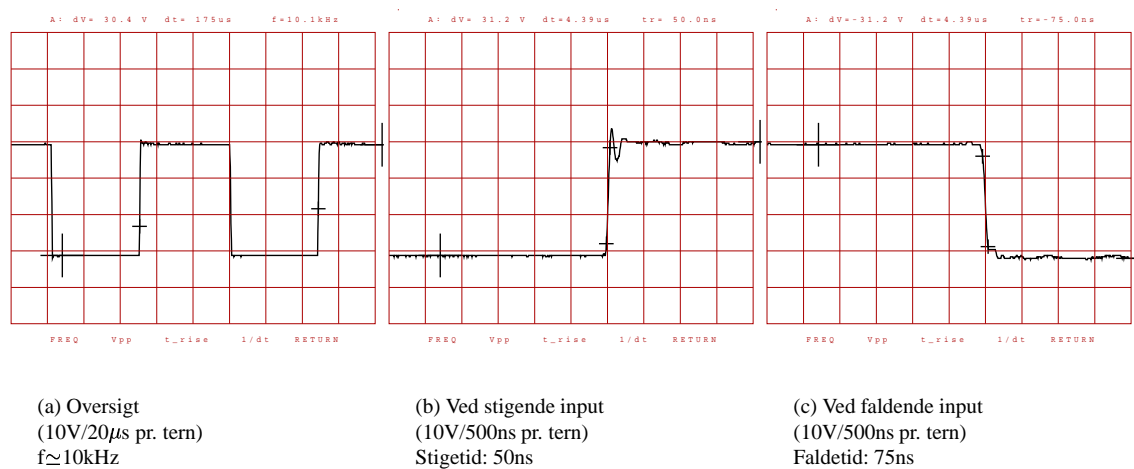
Når T3/T4 ikke når at være slukket længe nok til at motorstrømmen kan dø ud, skifter motorspændingen direkte mellem 0 og forsyningsspændingen, i takt med at T3/T4 slukkes og tændes. De meget hurtige skiftetider opstår pga. motorens selvinduktion.

Den ringning der opstår når T3/T4 tændes, før motorstrømmen er uddød, repræsenterer strømforsyningsens step-respons. Samme indsvingningsforløb ses i afsnit 22.4.5, når T3/T4 slukkes.

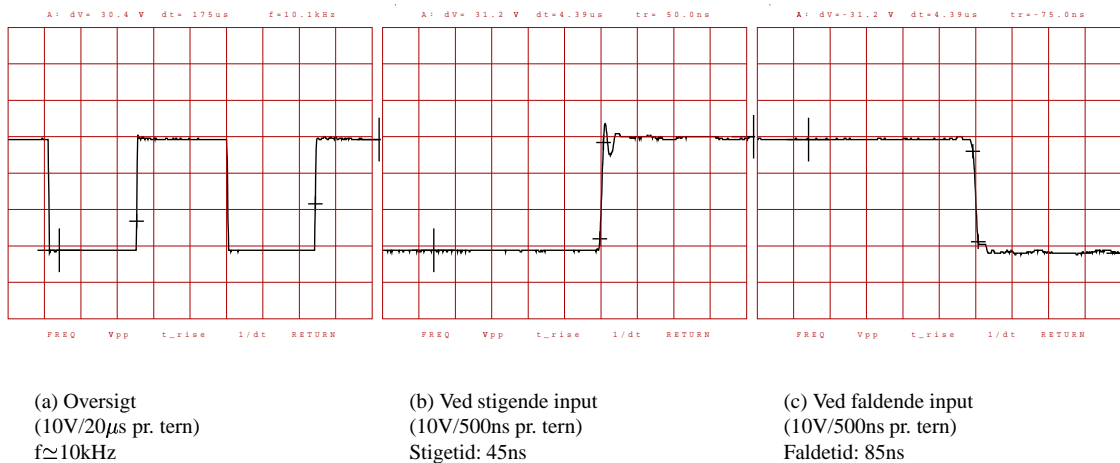


Figur 22.29: Spænding over belastet motor fra Cato, ved 12V forsyning.

Gentages målingerne for en belastet motor, ses det tydeligt at motorstrømmen er vokset, idet strømmen ikke længere kan nå at dø ud, i løbet af de ca. 500 μ s T3/T4 er slukkede, ved 1kHz switchfrekvens.



Figur 22.30: Spænding over en ubelastet motor, ved 30V forsyning.

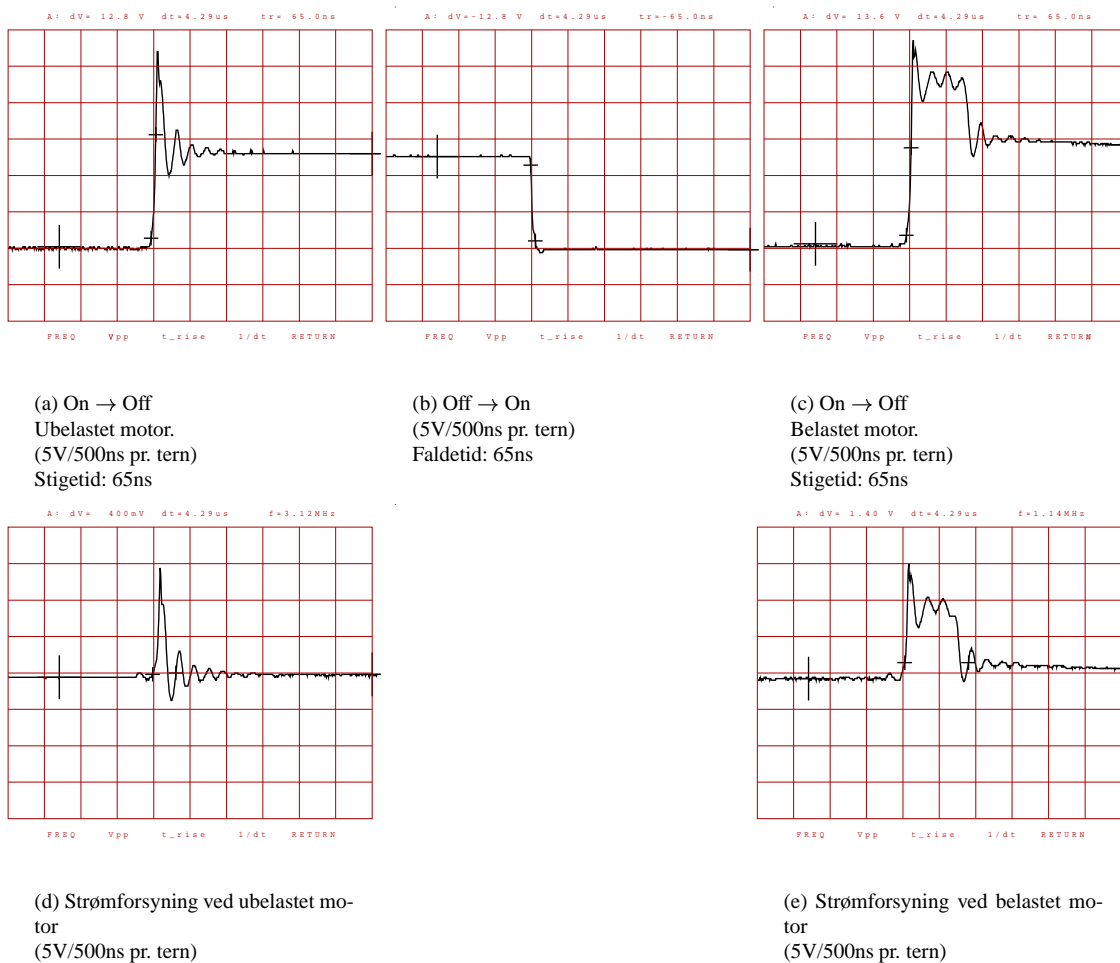


Figur 22.31: Spænding over en belastet motor, ved 30V forsyning.

Målinger ved 30V forsyningsspænding, viser at Rungner fungerer upåklageligt, også ved 30V.

22.4.5 Spænding over T3/T4

For at fuldende billedet af T3/T4's funktion, måles spændingen mellem deres *drain* og *source* terminal.



Figur 22.32: Spænding over T3/T4 og strømforsyning (12V), ved drift af motor

Målingen viser den forventede firkantkurve, med flanker der svarer til flankerne på motorspændingen. Ved drift af ubelastet motor, ses den samme karakteristiske ringning som der er på motorspændingen når T3/T4 tændes. Ved belastet motor, anes samme ringning, overlejret på en form for firkant.

Ingen af fænomenerne kan iagttages på motorspændingen, men ses helt identiske på Rungners terminaler til (effekt) strømforsyning (figur 22.32-d og -e). Ringningen er tilsyneladende strømforsyningen's (inklusive tilledninger m.m.) respons på en *stepformet* ændring i strømforbrug. Firkantspændingen der kan iagttages både over T3/T4 og på strømforsyningen, skyldes formentligt strømforsyningens regulators respons på et pludseligt faldende strømforbrug.

22.4.6 Strømforbrug til signaldel

Ved dimensionering af DC-DC konvertere, filtre m.m. til drift af Rungners signaldel, er det vigtigt at kende strømforbruget. Jeg har foretaget nogle få målinger af forbruget fra de to nødvendige spændingskilder, under forskellige driftsforhold. Under testen var Rungner belastet med en 47Ω modstand, og V_{ggh} var 15V under hele testen. Spændingen til forsyning af selve H-broen, og resten af signaldelen (V_{dd} og power- V_{dd}) blev varieret mellem 15V og 30V. Indgangssignalet til Rungner var enten inaktivt, eller bestod af et firkantsignal med 50% duty-cycle, ved forskellige frekvenser.

Indgang:	Inaktiv		10kHz, 50%		50kHz, 50%		100kHz, 50%	
V_{dd}	I_{Vdd}	I_{Vggh}	I_{Vdd}	I_{Vggh}	I_{Vdd}	I_{Vggh}	I_{Vdd}	I_{Vggh}
15 V	2.7 mA	0 mA	7.5 mA	0.9 mA				
20 V	4.5 mA	0 mA						
25 V	6.3 mA	0 mA						
30 V	8.2 mA	0 mA	12.8 mA	1.2 mA	32 mA	1 mA	69 mA	1 mA

Tabel 22.6: Måling af strømforbrug til signaldel

Måleresultaterne kan opsummeres i følgende tommelfingerregel:

$$I_{Vdd} \leq 2 \times 10^{-4} \Omega^{-1} \times V_{dd} + 6 \times 10^{-7} As \times f_{switch} \quad (22.2)$$

$$I_{Vggh} \leq 1.5 mA \quad (22.3)$$

22.4.7 Montering og håndtering

Alle de otte kopier jeg oprindeligt havde monteret komponenter på, var defekte ved afprøvningen, eller blev det undervejs. På trods af vanskelighederne ved at fejlfinde et tykfilmbaseret kredsløb, fandt jeg frem til at defekten i alle tilfælde skyldtes en eller flere defekte MOS-FET effekt transistorer. En del af fejlene kan henføres til at jeg har anvendt brugte komponenter, der kan have taget skade ved udlodningen. For at undersøge denne mulighed, opbyggede jeg to nye kopier, med helt nye komponenter, hvor jeg tog følgende forholdsregler for at undgå at beskadige komponenterne:

- Arbejdsbordet blev dækket af en tynd aluminiumsplade, som jeg, vha. et ESD-beskyttelses armbånd, forbandt mig selv til under montagearbejdet.
- Jeg arbejdede barfodet, for at eliminere opbygningen af ladning ved friktion mellem kunstofskosåler og gulvbelægning.
- MOS-FET transistorerne blev monteret til sidst, og jeg sikrede mig at jeg ikke kom i elektrisk kontakt med transistorerne eller resten af kredsløbet under montagen af disse.

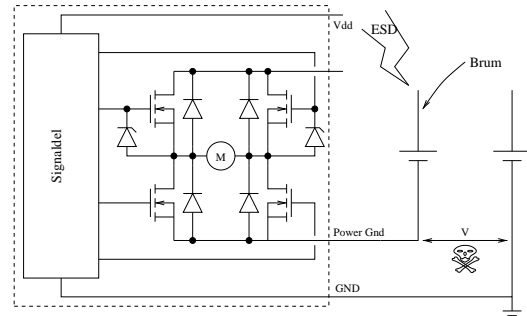
- Under håndtering af det færdige kredsløb sikrede jeg mig at jeg ikke kom i elektrisk kontakt med kredsløbet under håndtering, og at al loddearbejde blev foretaget på et potentialeudlignet underlag, med potentialeudlignet loddekolbe.
- Kredsløbene blev opbevaret i en kunststofkasse, foret med et ledende skummateriale.

Begge kredsløb fungerede upåklageligt, og det er dem der ligger til grund for målingerne i dette afsnit.

Da de første målinger havde vist at begge kredsløb fungerede, ville jeg teste hvordan Rungner fungerede under hårde belastninger, integreret i AGV'en Cato. Efter at have monteret begge kredsløb i Cato, virkede ingen af dem, og en nærmere undersøgelse viste at det igen drejede sig om defekte MOS-FET effektransistorer.

Undervejs i monteringsarbejdet, har Signaldelen og effektdelen af Rungner været forbundet til hhv. laboratorieforsyning med jordforbindelse, og en blyakkumulator der en overgang ikke havde potentialeudligning med laboratorieforsyningen. En sandsynlig årsag til ødelæggelsen af motordriverne, er at potentialeforskellen mellem signaldel og effektdel har overskredet transistorernes $V_{gs_{max}}$ på 20V, og dermed beskadiget det få nm tykke SiO lag der skiller gate og source i transistorerne.

En voldsom potentialeforskel kan netop opstå, ved ESD, eller indkoblet brum, fordi den eneste forbindelse mellem Rungners signal- og effektdel, er den parasitiske capacitor mellem MOS-FET transistorernes *drain/source* og *gate*. T1/T2 er beskyttede af en 18V zenerdiode (D7/D8), men T3/T4 er ubeskyttede. Hvis kredsløbet forsøges anvendt med hhv. T3 eller T4 defekt (kortslettet), vil Strømmen løbe direkte gennem den defekte transistor og dens tilsvarende *høje* transistor (T2 hhv. T1), når denne tændes. Er kredsløbet koblet til en ubegrænset spændingsforsyning (batteri), overskrider den høje transistors I_{max} øjeblikkeligt.



Figur 22.33: Fare for spændingsforskel mellem effekt- og signaldel ved montering

22.5 Effekt og virkningsgrad

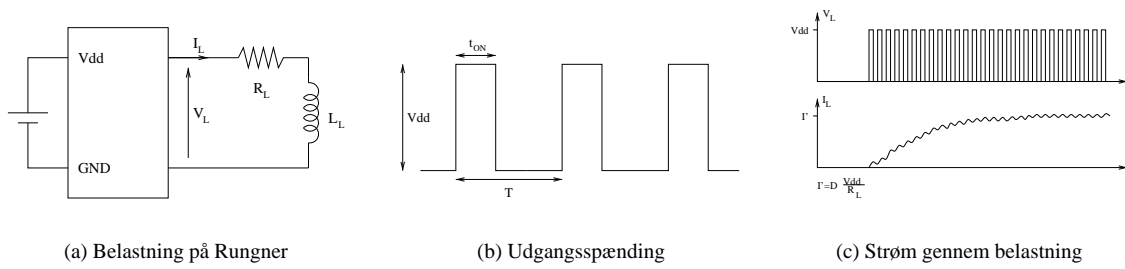
Selv om der sker en drastisk reduktion af spilleffekten ved at anvende en *switch-mode* H-bro, vil der stadig være et tab i H-broen. Tabet stammer primært fra:

- Den ohmske modstand i lederbaner, lodninger, og komponenter.
- Effektafsættelsen i transistorer der passerer gennem deres lineære områder.
- Spændingsfaldet over P-N overgangene i afkoblingsdioderne.

Jeg har ikke nået at foretage de ønskede elektriske og kalorimetriske målinger af effekttabet i Rungner, men kun opstillet en simpel matematisk model af det forventede tab. Modellen gennemgås herunder.

22.5.1 Belastningen

Jeg antager at Rungner belastes af en induktiv belastning X_L , der kan modelleres som en modstand R_L i serie med en spole L_L . Jeg antager endvidere at tidskonstanten $\tau_L = L_L/R_L$ er væsentligt større end perioden for den anvendte *switch*-frekvens f_s (jvf. afsnit 22.4.4 og figur 22.27).



Figur 22.34: Model af belastning

Når X_L udsættes for en firkantspænding mellem 0V og V_{dd} , med periodetid $T = 1/f_s$, og en pulsbredde $D = t_{ON}/T$, vil X_L fungere som lavpasfilter, og strømmen vil konvergere mod gennemsnitsværdien:

$$I_L = D \frac{V_{dd}}{R_L} \quad (22.4)$$

22.5.2 Forventede tab

Den tilnærmelsesvis konstante motorstrøm løber to forskellige veje gennem H-broen, afhængigt af om H-Broen er tændt eller slukket.

Uanset hvilken vej motorstrømmen løber gennem Rungner, passerer den ca. 15 kvadrater effektlederbane, hvis modstand repræsenteres som $R_c = 15R_f$, hvor R_f er flademodstanden af lederbanerne. Tabet til lederbanerne er givet ved:

$$P_C = I_L^2 \cdot R_c \quad (22.5)$$

Når H-broen er tændt, passerer motorstrømmen to tændte MOS-FET transistorer, der hver især har en indre modstand: $R_{DS,ON}$. Når H-broen er slukket, passerer strømmen kun den ene. Tabet i transistorernes indre modstande er givet ved:

$$P_T = (1 + D) \cdot I_L^2 \cdot R_{DS,ON} \quad (22.6)$$

Når H-broen er slukket, løber motorstrømmen gennem en afkoblingsdiode, der har et spændingsfald over P-N overgangen på V_D . Tabet i denne diode er givet ved:

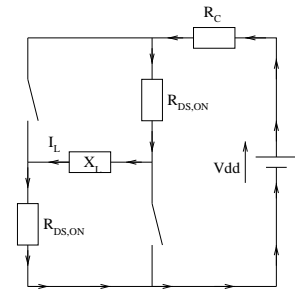
$$P_D = (1 - D) \cdot I_L \cdot V_D \quad (22.7)$$

Når en transistor, ved et skift, befinder sig i en tilstand mellem tændt og slukket, kan der afsættes en større effekt i den. Ifølge afsnit 22.4.5, befinder den skiftende transistor sig maksimalt 65ns i en sådan mellemtilstand ved hvert skift. Hvor meget energi der tabes i transistoren under et skift, afhænger af mange faktorer jeg ikke vil fordybe mig i, men en øvre grænse for den gennemsnitlige effektafsættelse pga. skift kan gives ved:

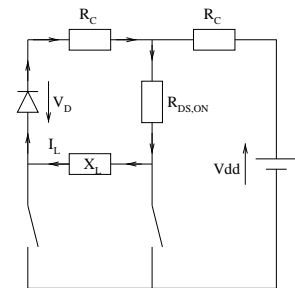
$$P_S \leq I_L \cdot V_{dd} \cdot 2 \cdot 65ns \cdot f_s \quad (22.8)$$

En model for det totale effekttab i Rungner er givet ved (22.4) og (22.9).

$$P_w = I_L(V_D(1 - D) + I_L(R_c + (1 + D)R_{DS,ON}) + V_{dd} \cdot 130ns \cdot f_s) \quad (22.9)$$



(a) Tændt H-bro



(b) Slukket H-bro

Figur 22.35: Strømveje

Den samlede forbrugte effekt fra strømforsyningen beregnes som:

$$P = I_L^2 \cdot R_L + P_w \quad (22.10)$$

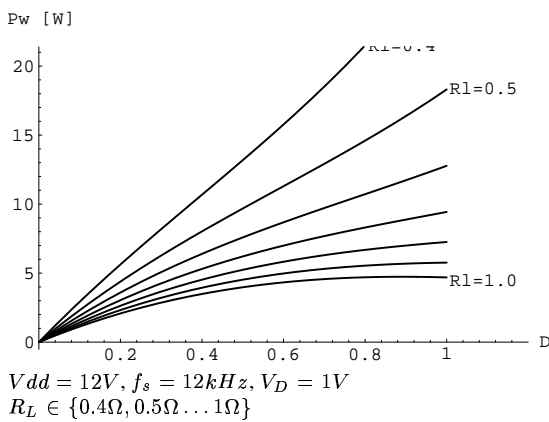
Virkningsgraden er givet ved:

$$Q = 1 - \frac{P_w}{P} \quad (22.11)$$

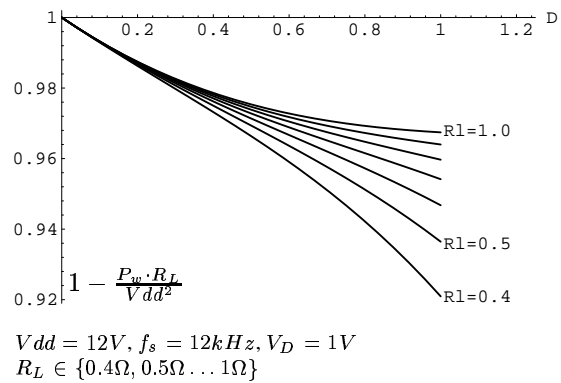
Rungner		Applikation	
		Cato	James
R_c	3m Ω	R_L	0.7 Ω 0.4 Ω
$R_{DS,ON}$	14m Ω	V_{dd}	12V 14V
V_D	1V	f_s	12kHz 12kHz

Tabel 22.7: Værdier til effektberegning

For at illustrere modellens forudsigelser af effekttab, har jeg taget udgangspunkt i driftsbetingelserne i Cato og James. Figur 22.36, viser effekttabet i Rungner, som funktion af den anvendte pulsbredde, ved forskellige belastninger. Figur 22.37, viser det samme, men sætter effekttabet i forhold til den maksimalt opnåelige belastning.

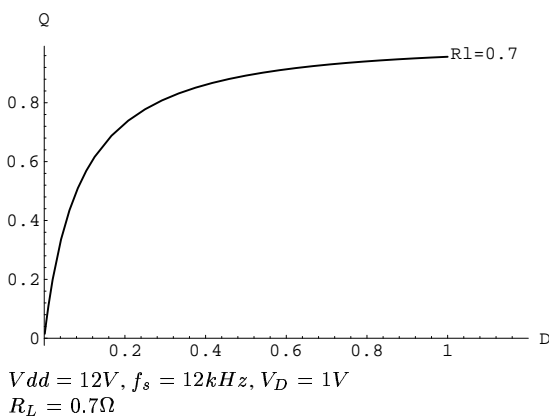


Figur 22.36: Spildeffekt

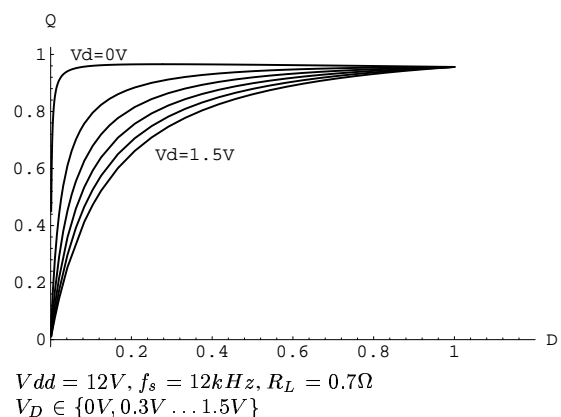


Figur 22.37: Falsk virkningsgrad

Figur 22.38 viser Rungners virkningsgrad, som funktion af pulsbredden, ved en belastning $R_L = 0.7\Omega$. Ved 100% pulsbredde, er virkningsgraden den samme som vist i figur 22.37, men ved lave pulsbredder falder virkningsgraden markant. Den lave virkningsgrad ved små pulsbredder skyldes primært tabet i afkoblingsdioden. Figur 22.39 viser samme graf for forskellige diodespændinger.



Figur 22.38: Virkningsgrad



Figur 22.39: V_D 's betydning for virkningsgrad

Del III

Noter

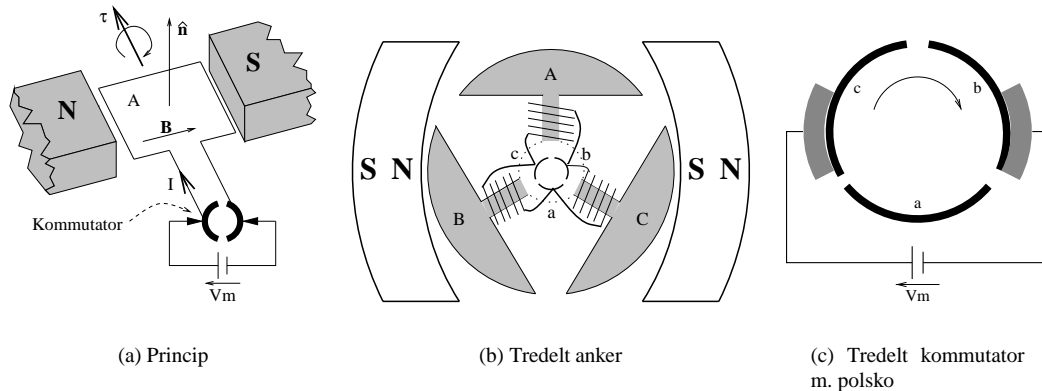
Kapitel 23

Opbygning og brug af DC-motorer

23.1 DC-motoren

I dette afsnit giver jeg en kort introduktion til opbygningen af en DC-motor, samt opstiller en simpel matematisk model for den, til senere brug.

23.1.1 Opbygning



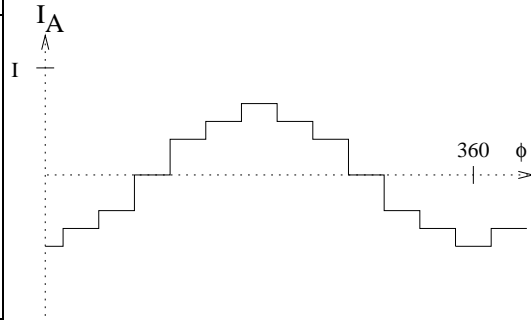
Figur 23.1: skitse af DC-motor

Figur 23.1.1(a) viser princippet i en DC-motor. Motorens bevægelige del — ankeret — befinder sig i et magnetfelt: \mathbf{B} , der både kan genereres vha. permanente magneter og elektromagneter. Når der sendes en strøm gennem ankerviklingen, vil ankeret udsættes for et drejningsmoment: $\tau = I A \hat{\mathbf{n}} \times \mathbf{B}$, hvor I er strømmen gennem ankerviklingen, A er ankerviklingens areal, $\hat{\mathbf{n}}$ er enheds normalen til A , og \mathbf{B} er magnetfeltet. Drejningsmomentet får ankeret til at dreje rundt. Når ankeret har drejet 90° , vil spændingen over ankeret skifte polaritet, pga. kommutatorens bevægelse. Kommutatoren sørger altså for at drejningsmomentet altid har samme retning. For den skitserede motor er drejningsmomentet (ved konstant strøm), proportionalt med cosinus til vinklen mellem $\hat{\mathbf{n}}$ og \mathbf{B} , hvilket giver en uensigtsmæssigt ujævn kørsel.

I praksis opbygges ankeret i DC-motorer af en række ankerspoler, der er drejet i forhold til hinanden. Som minimum anvendes 3 ankerspoler, som vist i figur 23.1.1(b). Som indikeret vikles spolerne ofte om et ferromagnetisk materiale.

Figur 23.1.1(c) Viser en forstørrelse af motorens tredelte kommutator, sammen med de to polsko. Kommutatoren og polskoene giver 12 forskellige koblinger af de tre ankerviklinger, som vist i tabel 23.1. Afhængigt af hvor optimalt kommutatoren og polskoene er konstrueret, bevirker motorens konstruktion at vinklen mellem den effektive \hat{n} , og \mathbf{B} , aldrig overstiger $15 - 30^\circ$. Drejningsmomentet for en motor med tredelt anker varierer derfor mindre end $1 - \cos 30^\circ \simeq 14\%$, i værste tilfælde. En optimalt konstrueret tredelt motor vil have en variation på mindst $1 - \cos 15^\circ \simeq 3.5\%$.

Polsko +	Polsko -	I_A/I	I_B/I	I_C/I	Vinkel
c	b	-2/3	1/3	1/3	0
c+a	b	-1/2	0	1/2	30
a	b	-1/3	-1/3	2/3	60
a	b+c	0	-1/2	1/2	90
a	c	1/3	-2/3	1/3	120
a+b	c	1/2	-1/2	0	150
b	c	2/3	-1/3	-1/3	180
b	c+a	1/2	0	-1/2	210
b	a	1/3	1/3	-1/3	240
b+c	a	0	1/2	-1/2	270
c	a	-1/3	2/3	-1/3	300
c	a+b	-1/2	1/2	0	330



(a) Ankerstrømme

(b) Vikling A

Tabel 23.1: Strømme i ankerviklingerne på en tredelt motor

I praksis er motorens mekaniske opbygning mindre interessant, idet alle forhold omkring opbygningen sammenfattes til en række makroskopiske egenskaber ved motoren, såsom: Nominel spænding, strøm, omløbstal, moment, motorkonstant etc.

23.1.2 Matematisk model

For de fleste praktiske tilfælde kan det uden problemer antages at DC motorens drejningsmoment er uafhængigt af ankerets vinkel.

Er magnetfeltet konstant, kan en DC-motors drejningsmoment udtrykkes som:

$$M_m = K_m I_m \quad (23.1)$$

Hvor I_m er strømmen gennem ankerviklingerne, og *motorkonstanten* K_m , bestemmes af $|\mathbf{B}|$, antallet af viklinger, ankerets opbygning, og andre konstante faktorer ved motorens konstruktion.

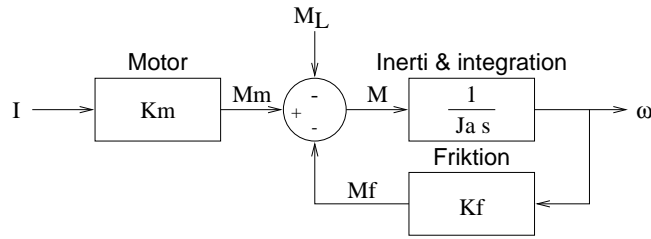
Samenhængen mellem moment omdrejningshastighed kan tilnærmes med:

$$\begin{aligned} m_m(t) &= m_f(t) + m(t) + m_L(t) \\ m_f(t) &= K_f \omega(t) \\ \omega'(t) &= m(t) / J_a \end{aligned} \quad (23.2)$$

Hvor J_a er motorens intertimoment, $\omega(t)$ er ankerets vinkelhastighed, $m_f(t)$ er det moment der skal til for at overvinde friktionen i motoren (idet det antages at $m_f(t)$ er proportional med $\omega(t)$), og $m_L(t)$ er det moment der går til at trække en evt. belastning af motoren.

(23.2) laplacetransformeres til (23.3), der kan udtrykkes med blokdiagrammet i figur 23.2.

$$\begin{aligned} M_m(s) &= M_f(s) + M(s) + M_L(s) \\ M_f(s) &= K_f(s) \omega(s) \\ s \omega(s) &= M(s) / J_a \end{aligned} \quad (23.3)$$

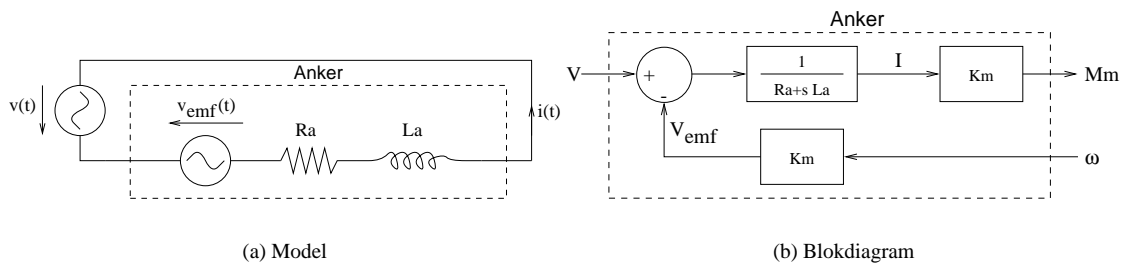


Figur 23.2: Mekaniske faktorer

Er motoren ubelastet, er overføringsfunktionen for motoren givet ved (23.4).

$$\frac{\omega(s)}{I(s)} = \frac{K_m}{J_a s + K_f} \quad (23.4)$$

(23.4) viser — som forventet — at motorens mekanik fungerer som 1. ordens lavpasfilter, med knækfrekvens: $f_{-3dB} = \frac{K_f}{2\pi J_a}$



Figur 23.3: Elektriske faktorer

For at forstå motorens respons som funktion af den påtrykte spænding, opstilles i en simpel model for motorens elektriske del. Figur 23.3(a) viser en model af motorens anker. Ankerviklingen har en selvinduktion: L_a , en modstand: R_a , og når ankeret bevæger sig i motorens magnetfelt, induceres en elektromotorisk kraft: $v_{emf}(t) = \omega(t)K_m$ i ankerviklingen. (23.5) giver sammenhængen mellem ankerstrøm og ankerspænding, som funktion af t , mens den laplace transformerede (23.6), udtrykker sammenhængen som funktion af s . Figur 23.3(b) viser modellen som blokdiagram.

$$\begin{aligned} v(t) &= v_{emf}(t) + R_a i(t) + L_a \frac{di(t)}{dt} \\ v_{emf}(t) &= \omega(t)K_m \end{aligned} \quad (23.5)$$

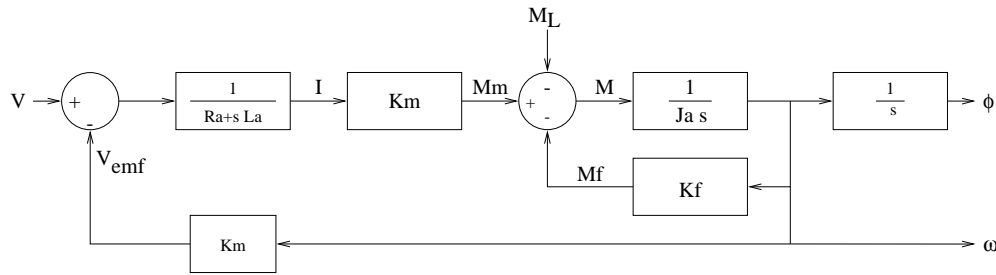
$$\begin{aligned} V(s) &= V_{emf}(s) + R_a I(s) + s L_a I(s) \\ V_{emf}(s) &= \omega(s) K_m \end{aligned} \quad (23.6)$$

Blokdiagramet i figur 23.4 viser sammenhængen mellem polspændingen V , ankerstrømmen I , momenter, vinkelhastigheden ω , og ankervinklen ϕ , der — hvis man ser bort fra evt. belastning — også kan udtrykkes som:

$$\frac{\omega(s)}{V(s)} = \frac{K_m}{(sL_a + R_a)(sJ_a + K_f) + K_m^2} \quad (23.7)$$

$$\frac{\phi(s)}{V(s)} = \frac{K_m}{s((sL_a + R_a)(sJ_a + K_f) + K_m^2)} \quad (23.8)$$

$$\frac{I(s)}{V(s)} = \frac{sJ_a + K_f}{(sL_a + R_a)(sJ_a + K_f) + K_m^2} \quad (23.9)$$

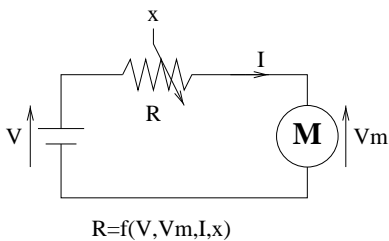


Figur 23.4: Simpel model af DC-motor

(23.7) og (23.9) klassificeres begge som 2. ordens type 0 systemer, mens (23.8) pga. integrationen af ω , klassificeres som et 3. ordens type 1 system. DC-motoren fungerer altså som 2. ordens lavpasfilter mht. både strøm og vinkelhastighed, som funktion af polspændingen. Motorens *mekaniske* knæfrekvens opstår pga. ankerets intertimitmoment, og den *elektriske* opstår pga. selvinduktionen i ankerviklingen. Knæfrekvensernes individuelle placering afhænger af motorens konstruktion, og den tilkoblede belastning, men den elektriske knæfrekvens vil typisk være langt højere end den mekaniske.

De udledte overføringsfunktioner gælder for en ubelastet motor, men såfremt motoren skal drive 1. ordens systemer af formen: $M_L = \omega(sa + b)$ [inerti og friktion], vil systemets samlede overføringsfunktion have samme form, blot med andre koeficienter end J_a , og K_f . Skal motoren drive systemer der ikke domineres af inertie og friktion, må overføringsfunktionerne udledes fra en model der medtager hele det nye system.

23.2 Effektforstærker



Figur 23.5: Princip i effektforstærker

En DC-motor der forsynes med energi fra en konstant spændingskilde, reguleres ved at indsætte aktive komponenter — transistorer e.lign. — i serie med motoren; de aktive komponenter indeholder ikke selv strøm eller spændingskilder, og modeleres derfor som variable modstande. Afhængigt af hvilke typer komponenter der anvendes, og hvordan de kobles, kan forskellige parametre af motorens energiforsyning reguleres, som funktion af et styresignal. Figur 23.5 viser dette princip, som henholdsvis mono- og bi-polar forstærker. Den bipolare forstærker går også under betegnelsen: *H-bro*, pga. konfigurationens lighed med bogstavet **H**.

23.2.1 Variabel spænding

23.2.2 Variabel strøm

23.2.3 Switch mode

Kapitel 24

Programmering af VMPMKC2 CPU-kort under OS-9

24.1 Godt begyndt

24.1.1 Isenkram

VMPM68KC2 er et 68020 baseret CPU modul til 16 bit VMEbus. Modulet består af: 68020 CPU, sokkel til 68881 FPU, op til 512Kb ROM, 1Mb RAM, og 2 serielporte. Modulet er opbygget af to 3U EUROPA kort, i en piggyback konstruktion.

Selv om kortet i princippet kan fungere på egen hånd, sker strømforsyning og adgang til andre enheder via VME bussen, hvorfor kortet er monteret i et kortmagasin, med strømforsyning, diskcontroller, floppy- og hard-disk, og hvad man ellers ønsker af I/O.

På fronten af modulet er de to serielporte ført ud vha. hvert sit 15 polede sub-D stik. Mellem stikkene sidder en rød RESET knap, og en sort ABORT knap. BREAK knappen genererer et *non maskable interrupt* (niveau 7). Mellem knapperne sidder en rød lysdiode der indikerer hvis CPUen går i HALT mode.

Al kommunikation med modulet foregår via RS-232. For at komme igang skal man bruge en terminal, eller en computer med RS-232 port og et passende terminalprogram. Selv bruger jeg en PC under Linux, med minicom, under MS-DOS med procom, eller under Windows med hyperterminal.

Terminalen forbindes til Modulets port 0, der er placeret nederst. Det er tilstrækkeligt med et 3 leder nulmodem kabel, forbundet til GND, TxD, og RxD; der er placeret på hhv. pin 1, 2 og 3 i det 15 polede stik. Der kommunikeres ved 9600 bps, med 8 databits, ingen paritet, og 1 stopbit.

Hvis kun RxD og Txd føres over, skal terminalens/computerens hardware handshake slås fra.

Hvis alt fungerer som det skal, skal der dukke en opstarts meddelelse op når PEPen tændes.

24.1.2 Login

CPU modulet starter altid op i en debugger/monitor (PEPbug). Ved at give kommandoen **help** fås en oversigt over kommandoer. Har man koblet en terminal eller computer til den anden serielport, kan kommandoen **TM** (transparent mode) være nyttig, idet den kopierer input på den ene port til output på den anden, og effektivt gør PEPen transparent.

Operativsystemet ligger på harddisken, og startes op med kommandoen **OS**

24.1. GODT BEGYNDELSE KAPITEL 24. PROGRAMMERING AF VM68K CPU-KORT UNDER OS-9

```
V      V M      M P P P P P M      M      K      K C C C C
V      V M M      M M P      P M M      M M      K      K C
V      V M      M M P P P P P M      M      M      K K      C
V V      M      M P      P      M      M      K      K C
V      M      M P      P      M      M      K      K C C C C
```

VM68K Debugger/Diagnostics Version 1.4 - 20/FEB/89

CPU passed test
SCC passed test
TIC passed test
No FPC detected
RTC passed test

ONBOARD RAM found : \$00000000 - \$000FFFFF
VME-BUS RAM found : \$87000000 - \$870FFFFF

M PEPbug>

Figur 24.1: Opstarts meddelelse

M PEPbug>os
OS-9/68K System Bootstrap
sysgo: WELCOME to Professional V2.2
Hit RETURN to continue

OS-9/68000 V2.2 PEP VME Series - 68020 97/07/25 12:31:01

User name?:

Er man ny bruger på systemet skal man oprettes som bruger, og skal derfor gå ind som system administrator. Normalt har administratoren brugernavn **super** og password **user**.

Administratoren skal gennem en lang login sekvens. Normalt viser operativsystemet kun en side ad gangen, og der trykkes SPACE for at komme videre i outputtet.

Password:
Process #05 logged on 97/07/25 12:35:02
Welcome!

```
***** .login file *****
.
. >>>> DET MESTE ER IKKE GENGIVET <<<<
.
***** eof .login *****
-l -nt
su>
```

24.1.3 De første trin

Når login sekvensen er færdig, står man i roden af filsystemet, der har stien **/dd**. For at snuse rundt bruges kommandoerne:

dir Svarer til ls under unix, eller DIR under MS-DOS. Med option **-e** fås en udvidet liste, svarende til ls -l. Med option **-a** vises også filer der starter med . , svarende til ls -a.

chd *Change data directory* Svarer til cd under unix eller MS-DOS.

list Viser indholdet af en tekstfil.

dump Viser indeholdet af en fil, som hex og tekst.

pd Viser det nuværende data directory. Option **-x** viser execution directory.

-e Slår tekstbaserede fejlmeddelelser til.

-ne Slår tekstbaserede fejlmeddelelser fra.

OS-9 skelner ikke mellem store og små bogstaver i stinavne, men der er konvention for at filer navngives med små bogstaver, og directories med store.

De vigtigste directories i første omgang er: CMDS, der indeholder alle værktøjer og kommandoer; SYS, der bl.a. indeholder password filen; og USR, der indeholder brugernes directories.

```
su> chd sys
su> dir
                                Directory of . 13:00:42
errmsg                errmsg.short    password                termcap                termset
umacs.hlp
su> list password
***** password file *****
super,user,0.0,130,/dd/CMDS,.,shell
applic,user,0.1,130,/dd/CMDS,/dd/APPLIC,shell
*
zaphod,,1.1,100,/dd/CMDS,/dd/USR/ZAPHOD,shell
***** eof *****
su>
```

24.1.4 Editering

For at kunne bruge skærbaserede programmer, som f.eks. editorer, skal systemet vide hvilken terminal/terminalemulering der bruges. Filen: **/dd/SYS/termcap** definerer egenskaberne for en række forskellige.

Navnet på den anvendte terminalemulering lægges i systemvariablen TERM, vha. kommandoen **setenv**. Til IMADA's 6110 og 6310 grønne terminaler anvendes **abm85**. I de fleste andre tilfælde anvendes **vt100**

```
su> setenv TERM vt100
```

Editoren på systemet er *mikro emacs*, der startes med kommandoen **umacs**. Mikro emacs er meget lig den almindelige emacs, men der er små forskelle. F.eks. bruges aldrig C-S (Ctrl sammen med S), da OS-9 opfatter dette som flow-control tegnet XOFF, og fryser output fra serielporten. Kommer man alligevel til at bruge C-S, ophæves effekten ved at bruge C-Q, der modsvarer tegnet XON. I mange tilfælde hvor PEPen tilsyneladende er *gået ned*, viser det sig at den venter på et XON tegn. I stedet for at gemme filer vha C-X C-S, bruges C-X S. Markøren styres ikke vha. piletasterne, men i stedet vha. C-P, C-N, C-B, og C-F, der står for hhv. *previous line*, *next line*, *backward*, og *forward*. I stedet for at bruge DEL til at slette fremad, bruges C-D. For at forlade umacs bruges i lighed med emacs C-X C-C. Bruges C-X C, suspenderes umacs, og brugeren får en ny *shell*. Forlades den nye *shell*, med logout, eller ved at trykke ESC, vendes tilbage til umacs. Denne egenskab er nyttig til at kompilere kildetekster uden at forlade editoren. Filen: **/dd/SYS/umacs.hlp** Indeholder en liste over disse og andre kommandoer.

24.1.5 Oprettelse af bruger

Første trin i oprettelse af brugere, er at tilføje en linie i password filen.

```
su> chd /dd/SYS
su> copy password pasword.bak
su> umacs password
```

Filen tilføjes linien:

```
brugernavn,password,1.2,100,/dd/USR/BRUGERNAVN/EXE,/dd/USR/BRUGERNAVN,shell -p="@brugernavn> "
```

Brugernavn og password giver sig selv. Glemmer man sit password kan man blot kigge i password filen som superbruger. Decimaltallet efter password, angiver brugergruppen før punktummet, og bruger ID inden for gruppen, efter punktummet. Brugergrupperne bruges til at administrere filrettigheder m.m. Brugergruppe 0 er superbrugerne, der kan administrere systemet. Det næste tal angiver default procesprioriteterne for de processer brugeren starter. Det næste argument angiver stien til brugerens *execution* directory. Derefter følger stien til brugerens hjemme directory, og derefter kommandoen til at starte en *shell*.

Efter at have tilføjet mig selv som bruger: **anders** ser filen således ud:

```
***** password file *****
super,user,0.0,130,/dd/CMDS,.,shell
applic,user,0.1,130,/dd/CMDS,/dd/APPLIC,shell
*
zaphod,,1.1,100,/dd/CMDS,/dd/USR/ZAPHOD,shell
anders,user,1.2,100,/dd/USR/ANDERS/EXE,/dd/USR/ANDERS,shell -p="@anders> "
***** eof *****
```

Tilbage for superbrugeren er nu blot at oprette hjemme- og execution-directoriet for den nye bruger.

```
su> chd /dd/USR
su> mkdir ANDERS
su> mkdir ANDERS/EXE
su> logout
```

Logger man ind som sit nye alter ego ser man følgende:

```
User name?: anders
Password:
Process #05 logged on      97/07/25 23:20:06
Welcome!
```

```
@anders>
```

Det er praktisk at oprette en login fil, der initialiserer systemet til ens egne behov.

```
@anders> setenv TERM vt100
@anders> umacs .login
```

Login filen kan f.eks. se sådan ud:

```
*Set terminal mode
setenv TERM vt100
*Nulstil variabel der holder styr paa shellnumre
setenv _sh 0
*Denne shell maa kun kunne forlades vha. logout kommandoen
-l
*Hvis der ikke oenskes scrollpauser fjernes * i starten af naeste linie
*tmode nopause
```

```
*Slå tekstuelle fejlmeddelelser til
-e

* Udskriv nuværende directory
echo "Du er i: " -r
pd
```

24.2 Udviklingssystemet

Ud over editoren umacs, består udviklingssystemet af C compiler, assembler, linker, og et make program.

24.2.1 Hello world!

Når man står i sit hjemmekatalog, indtastes kildeteksten: **hello.c** vha. umacs

```
#include <stdio.h>

void main()
{
    printf("Hello world\n");
}
```

Programmet compiles med kommandoen :

```
cc hello.c
```

cc Samler compiler, assembler, og linker under en hat. når oversættelsen af programmet er færdig, kan man sikre sig at den eksekverbare fil ligger i ens execute-directory, med kommandoen:

```
dir -x.
```

Programmet startes som vanligt, ved at give programnavnet som kommando:

```
hello
```

Når man skriver hardware nære programmer, kan det være en stor hjælp at kigge compileren lidt i kortene, på assemblerniveau. option -a til cc, standser oversættelsen før assamblering, og filen **hello.a** dukker op i ens hjemmedirectory.

```
psect hello_c,0,0,0,0,0
    nam hello_c
    ttl main
main:  link a5,#0
       movem.l #_1!1,-(sp)
       move.l #_3,d0 :6
       bsr _stkcheck
       lea _5(pc),a0
       move.l a0,d0 :2
       bsr printf
_4:
       movem.l -4(a5),#_1
       unlk a5
       rts :2
_3 equ 0xffffffffbc :0
_1 equ 0x00000100 :0
_2 equ 0x00000010 :0
_5 dc.b "Hello world!", $d,$0
ends
```

Man kan ændre i maskinkoden — f.eks. strengen: *Hello world!*, og derefter assamlere maskinkoden vha kommandoen:

```
cc hello.a
```

option -? til **cc** udskriver en kort beskrivelse af **cc**'s muligheder.

24.2.2 Funktionsdeklerationer

Det er en gammel C compiler der anvendes, og den variation af C der anvendes har nogle særheder i forhold til de fremherskende i dag.

En af de vigtigste faldgruber er måden variable deklarerer på i funktionshoveder. Herunder er vist et eksempel på hvordan det skal gøres.

```
#include <stdio.h>

int sum(tal1,tal2);

void main()
{
    printf("Summen af %d og %d er %d\n",28,14,sum(28,14));
}

int sum(tal1,tal2)
int tal1;
int tal2;
{
    return tal1+tal2;
}
```

24.2.3 Projekter

Ved større projekter, bliver det hurtigt upraktisk at arbejde ved at kalde **cc** direkte. I stedet bruges **make**.

Der er kotyme for at oprette separate direktories til kildetekster, objektfiler, og binære filer. Normalt kaldes de hhv. **SOURCE**, **RELS**, og **EXE**. Hvis man bygger *libraries* lægges de typisk i et **LIB** directory.

Make programmet læser filen: **makefile**, der indeholder instruktioner om hvordan forskellige kommandoer skal udføres. Herunder er vist en **makefile**, der indeholder instruktioner til at bygge et *library* (**MCI**) af mange forskellige kildetekster; og et program der bruger det (**mcitest**).

```
MYDIR = /dd/USR/ANDERS

ODIR = $(MYDIR)/EXE
SDIR = $(MYDIR)/SOURCE
RDIR = $(MYDIR)/RELS

LDIR = $(MYDIR)/LIB

CFLAGS =
RFLAGS =
LFLAGS =

#
# MCI
#
```

```

MCI:      $(LDIR)/MCI.l
          rdump -l $(LDIR)/MCI.l

$(LDIR)/MCI.l:  MCI.r MCIhardware.r MCios9.r
               -del $(LDIR)/MCI.l
               merge $(RDIR)/mci.r $(RDIR)/mcihardware.r $(RDIR)/mcios9.r>$(LDIR)/MCI$

MCI.r:  MCI.c MCI.h MCIhardware.h
        cc $(SDIR)/MCI.c -k2 -r=$(RDIR)

MCIhardware.r:  MCIhardware.c MCIhardware.h
               cc $(SDIR)/MCIhardware.c -k2 -r=$(RDIR)

MCios9.r:  MCios9.c MCios9.h
           cc $(SDIR)/MCios9.c -k2 -r=$(RDIR)

#
# mcitest
#

mcitest:  mcitest.r
          @cc $(RDIR)/$.r -l=$(LDIR)/MCI.l -f=$(ODIR)/$*

mcitest.r:  mcitest.c
           @cc $(SDIR)/mcitest.c -k2 -r=$(RDIR)

```

Biblioteket og programmet bygges med kommandoerne: `make MCI` og `make mcitest` henholdsvis.

24.3 Hardwarenær programmering

I dette afsnit gennemgår jeg de grundlæggende koncepter vedrørende adgang til VME bussen, fra både maskinkodeprogrammer og C programmer.

24.3.1 Adressering

Det 68020 baserede CPU modul arbejder med en intern 16/32¹ bus. VME bussen er 16/24, med udvidelsesmulighed til 32/32. PEPen bruger en 16/24 VMEbus.

CPU modulet kan adressere VME bussen i *standard adress mode*, hvor alle adresselinier anvendes, eller i *short address mode*, hvor kun de nederste 16 adresselinier anvendes. Afhængigt af hvilken metode der anvendes, mappes VME bussens adresseområde ind på CPU modulets 4GB adresseområde, som adresserne: $87000000_{16} \dots 87FFFFFF_{16}$, for standard, og $85000000_{16} \dots 8500FFFF_{16}$ for short. Hvilken adresseringsmetode der skal anvendes til en bestemt VME enhed, afhænger af enheden og dens konfiguration. Ud over standard/short adressering, kan VME enheder skelne mellem om CPU modulet arbejder i *user state* eller *supervisor state*.

Den anvendte VME bus har som nævnt 16 databits, og data kan overføres 16 eller 8 bits ad gangen. Adresseringen er byte orienteret, hvilket vil sige at det første 16-bits register har adressen 0, det næste adressen 2 etc. 16-bits-, også kaldet word-adgang kan derfor kun ske til lige adresser. De høje og lave 8

¹ 16 databits og 32 adressebits

bits i et word-register, kan adresseres separat, vha. byte-overførsler. En byte overførsel til en lige adresse adresserer de højeste 8 bits i registeret, mens en overførsel til en ulige adresse adresserer de nederste 8 bits.

24.3.2 Maskinkode

Eftersom VME adresser mappes ind som almindelige adresser i CPUens adresseområde, kan alle 68000 instruktionssættets adresseringsmetoder anvendes til VME adgang.

Herunder gives et par eksempler på hvordan data overføres til/fra VME bussen til processoren.

move.w #0,\$87001000 Skriver værdien 0, til word-registeret på adresse 1000_{16} , vha. standard adressering. Alle 16 bits sættes til 0.

move.w #\$FF,\$87001000 Skriver værdien $FF_{16} = 11111111_2$ til word-registeret på adresse 1000_{16} , vha. standard adressering. De øverste 8 bits sættes til 0, og de nederste 8 sættes til 1.

move.w #\$AAFF,\$85001000 Skriver værdien $AAFF_{16} = 1010101011111111_2$ til word registeret på adresse 1000_{16} vha. *short* adressering. De øverste 8 bits sættes til 10101010, mens de nederste alle sættes til 1.

move.b #0,\$87001000 Skriver værdien 0 til de øverste 8 bits i wordregisteret på adresse 1000_{16} , mens de nederste 8 bits forbliver uforandrede. Der anvendes standard adressering.

move.b #0,\$87001001 Skriver værdien 0 til de nederste 8 bits i wordregisteret på adresse 1000_{16} , mens de øverste 8 bits forbliver uforandrede. Der anvendes standard adressering.

move.b d0,\$85001001 Kopierer de nederste 8 bits i CPU register d0, til de nederste 8 bits i wordregisteret på adresse 1000_{16} , med *short* adressering.

move.w \$873e0300,d0 Kopierer værdien fra wordregisteret på adresse $3e0300_{16}$ til de nederste 16 bits i CPU register d0, med standard adressering.

move.b \$85002304,d0 Kopierer de øverste 8 bits fra wordregisteret på adresse 2304_{16} til de nederste 8 bits i CPU register d0, med *short* adressering.

move.b \$85002305,d0 Kopierer de nederste 8 bits fra wordregisteret på adresse 2304_{16} til de nederste 8 bits i CPU register d0, med *short* adressering.

24.3.3 C

Skal man læse eller skrive direkte til specifikke hukommelsesregistre i et C program, anvendes en pointer til den pågældende adresse. Pointertypen afgør hvilken type (byte,word,longword) overførsel der anvendes. Typerne byte, word, og longword findes ikke som standard i C. Hvilke C typer der svarer til bytes, words og longwords er afhængige af hvilken C compiler der anvendes.

I PEPernes C compiler, svarer bytes, words, og longwords til henholdsvis: **unsigned char**, **unsigned short int**, og **unsigned long int**. For at undgå problemer med et evt. skift til en anden compiler, bør disse typer defineres i en includefil, så der kun skal rettes eet sted. F.eks. med **#define** som herunder, eller vha. typedef.

```
#define BYTE unsigned char
#define WORD unsigned short int
```

Vil man overføre en byte- hhv word-værdi til VME adresse 300_{16} , vha. standard adressering, kan det se således ud:

```
* (BYTE *)0x87000300=bytevalue;
* (WORD *)0x87000300=wordvalue;
```


Adressen på registeret der skal skrives i, typekonverteres til en pointer til hhv en byte eller et word. Pointeren bruges så til at skrive i hukommelsen med. Skal der læses fra samme adresse, kan det se således ud.

```
bytevalue=* (BYTE *)0x87000300;
wordvalue=* (WORD *)0x87000300;
```

Naturligvis bør adressen på registeret kun være defineret et sted, så den er nem at rette. Herunder er et eksempel på et C program der læser og skriver en byte og et word fra adresse 300₁₆ på VME bussen, med standard adressering.

```
#define BYTE unsigned char
#define WORD unsigned short int

#define MY_BYTE * (BYTE *)0x87000300
#define MY_WORD * (WORD *)0x87000300

void main()
{
    BYTE b;
    WORD w;

    b=MY_BYTE; /* Læs byteværdi fra memory ind i b */
    MY_BYTE=b; /* Læs byteværdien b over i memory */
    w=MY_WORD; /* Læs wordværdi fra memory ind i w */
    MY_WORD=w; /* Læs wordværdien w over i memory */
}
```

Compileren oversætter ovenstående til nedenstående assemblerkode. Tallet : -2030042368 er rent faktisk 87000300_{16} , udskrevet som `signed long int`.

```

psect test_c,0,0,0,0,0
    nam test_c
    ttl main

main:  link a5,#0
       movem.l #_l11,-(sp)
       move.l #_3,d0 :6
       bsr _stkcheck
       subq.l #4,sp :2
       move.b -2030042368,3(sp) :8
       move.b 3(sp),-2030042368 :8
       move.w -2030042368,(sp) :6
       move.w (sp),-2030042368 :6
       addq.l #4,sp :2
_4
    unlk a5
    rts :2
_3 equ 0xffffffffbc :0
_1 equ 0x00000000 :0
_2 equ 0x0000000c :0
ends

```

Der er mange variationer over måden at adresserer fysiske hukommelsesceller fra C. Ofte arbejdes med enheder der har mange forskellige registre, på fortløbende adresser. I sådant et tilfælde er det muligt at lave en elegant implementation vha. en struct.

Herunder er gengivet uddrag af kildeteksten til et C program der adresserer registre i en netværks controller. Bemærk at alle registre i controlleren er 8 bits, og ligger på de lave 8 bits i hvert word register på VME bussen, og derfor ligger på ulige adresser. De tilsvarende lige adresser er i structen repræsenteret ved posterne: `space . .`

```

#define BYTE unsigned char
#define BASIS_ADDRESS 0x87002000

typedef struct _ARCREGS {
    BYTE    space00;
    BYTE    status_imask;
    BYTE    space01;
    BYTE    dstatus_command;
    BYTE    space02;
    BYTE    hiaddr;
    BYTE    space03;
    BYTE    loadaddr;
    BYTE    space04;
    BYTE    data;
    BYTE    space05;
    BYTE    reserved;
    BYTE    space06;
    BYTE    config;
    BYTE    space07;
    BYTE    setup_id;
    BYTE    space1[55];
    BYTE    vector_resoff;
    BYTE    space2[7];
    BYTE    id_reson;
} ARCREGS;

ARCREGS* device=(ARCREGS *)BASIS_ADDRESS;

void main()
{
    device->id_reson=0x00;
    device->vector_resoff=0xaa;
    device->dstatus_command=0x05;
    device->config=0x80;
    device->config=0x19;
}

```

Kigger man på assemblerkoden der produceres (herunder), afsløres det at compileren laver korrekt, men ikke just optimal kode. Compileren bruger a0 som register til at opbevare basisadressen, men kopierer basisadressen til a0 hver gang enheden adresseres. Til forsvar for compileren skal siges at den efterfølgende kører en optimeringsalgoritme på den producerede assemblerkode.

```

psect test_c,0,0,0,0,0
    nam test_c
    vsect
device:  dc.l -2030034944
    ends
    ttl main
main:   link a5,#0
    movem.l #_l!1,-(sp)
    move.l #_3,d0 :6
    bsr _stkcheck
    movea.l device(a6),a0
    clr.b 79(a0)
    movea.l device(a6),a0
    move.b #170,71(a0) :6
    movea.l device(a6),a0
    move.b #5,3(a0) :6
    movea.l device(a6),a0

```

```

move.b #128,13(a0) :6
movea.l device(a6),a0
move.b #25,13(a0) :6
_4
movem.l -4(a5),#_1
unlk a5
rts :2
_3 equ 0xffffffffc0 :0
_1 equ 0x00000100 :0
_2 equ 0x00000010 :0
ends

```

Ovenstående eksempel viser at det er relativt nemt at skrive hardwarenær kode vha. PEPens C compiler, men at der er fordele ved at skrive sådanne programmer i maskinkode.

24.4 Device drivers og file managers

24.4.1 OS-9's I/O system

Alle brugerprogrammer der benytter I/O under OS-9 benytter kernekald som: **I\$Open**, **I\$Read**, **I\$GetStt** etc. Der er et helt fast sæt kernekald til håndtering af I/O, og de har alle en fast defineret grænseflade.

Når en ny I/O kanal åbnes, parser kernen stinavnet, for at afgøre hvilket *device* der ønskes åbnet en I/O kanal til. *Devicet* angives først i stinavnet. F.eks. hentyder **/term** til den serielle port der er tilkoblet terminalen, og **/d0** henviser til floppy drevet.

Når kernen har afgjort hvilket *device* der skal anvendes leder den i hukommelsen efter et *device descriptor* modul, for den pågældende enhed. (Kommandoen **mdir** udskriver en oversigt over moduler i hukommelsen).

En *device descriptor* er et datamodul, der indeholder relevante oplysninger om et *device*. De vigtigste er navnet på den *file manager* og den *device driver* der administrerer *devicet*. *Device descriptor*, *file manager*, og *device driver*, kaldes tilsammen et *I/O subsystem*.

En *file managers* opgave er at håndtere den logiske struktur, med dertil hørende databehandling, for en klasse af beslægtede enheder. De tre mest benyttede *filemanagers* under OS-9, er:

SCF Der håndterer enheder med sekventielt dataflow. F.eks. serielle porte.

RBF Der håndterer enheder med blokstruktur. F.eks. (hard-)diske.

PIPEMAN Der håndterer *pipes*. *Pipes* er en virtuel sekventiel datakanal, der kan åbnes mellem processer i OS-9.

Når kernen vha. *device descriptoren* har fået navnet på *filemanageren*, forsøger den at åbne et programmodul af det navn i hukommelsen. Grænsefladen mellem kerne og *filemanager* er fast defineret, og består af 13 subrutiner i *filemanageren*, som kernen kan kalde. De fleste I/O kald fra brugerprogrammer til kernen, sendes i virkeligheden ubehandlet videre til *file manageren*. Kernens vigtigste opgave i den forbindelse er at afgøre hvilken *filemanager*.

Device driveren er interfacet mellem *file manageren* og den fysiske hardware. I lighed med *file manageren*, er *device driveren* et programmodul der ligger i hukommelsen. Interfacet mellem *file manager* og *device driver*, udgøres af en række funktioner i *device driver* modulet som *file manageren* kan kalde. Hvilke funktioner en *device driver* skal indeholde, og hvilke parametre de har, er bestemt af forfatteren til *file manageren*. SCF *file manageren* foreskriver funktionerne:

Init Initialiserer hardware, datastrukturer m.m.

Read Læser et tegn fra den fysiske enhed.

Write Skriver et tegn til den fysiske enhed.

Term Afslutter kommunikation med enheden (tøm buffere, af-initialiser hardware etc.)

GetStat/SetStat Overfør information der ikke har med de transmitterede data at gøre. F.eks. statusinformation, kommandoer om ænding af bufferstørrelse, transmissionshastighed m.m.

24.4.2 Udvikling af device descriptor

Device descriptoren er det datamodul der binder et I/O subsystem sammen. Ud over navnene på I/O subsystemets file manager og device driver, indeholder device descriptoren ofte information til både file manager og device driver, om f.eks. adresser, interrupt vectorer, konfiguration, etc.

På PEPerne skrives device descriptore i assemblerkode. Herunder viser jeg hvordan man laver en device descriptor til SCF file manageren. Jeg kalder device descriptoren for **aa**, hvilket også bliver navnet på det logiske device den kommer til at repræsentere.

SCF file manageren, og dens tilhørende device drivere forventer at device descriptoren rummer en masse information om hvordan den pågældende serielle kanal er konfigureret. F.eks: Baud rate, duplex, om output skal pauses for hver side der udskrives, hvor mange linier der er på en side etc.

For at gøre det nemmere for folk der vil lave device descriptore til SCF, ligger der en include fil: **/dd/DEFS/scfdesc.d**, som rummer næsten alle relevante poster, med tilhørende default værdier. Filen vises herunder:

```
*****
* Edition History
* #   date      comments                                     by
* --  -
* 00 09-28-83 Converted to 68000 from 6809 source             rfd
* 00 04-06-84 Added use of TrmDrNam macro for driver name     WGP
* 01 10-12-84 Added IRQ Level & resersted bytes.             rfd
* 02 10-24-84 Changed to "use" file format.                   rfd
* 03 11-05-84 Inserted macro for descriptor generation.       rfd
* 04 06-27-85 Added mode byte.                                rfd

Edition equ 4 current edition number

TypeLang set (Devic<<8)+0
Attr_Rev set (ReEnt<<8)+0
psect ScfDesc,TypeLang,Attr_Rev,Edition,0,0

dc.l Port port address
dc.b Vector auto-vector trap assignment
dc.b IRQLevel IRQ hardware interrupt level
dc.b Priority irq polling priority
dc.b Mode Device mode capabilities
dc.w FileMgr file manager name offset
dc.w DevDrv device driver name offset
dc.w 0 DevCon (reserved)
dc.w 0,0,0,0 reserved
dc.w OptSiz option byte count

* Default Parameters
Options
*
*      name      function                                     default
*      -
*      -
```

```

dc.b DT_SCF    device type          SCF
dc.b upclock   upcase lock          OFF
dc.b bsb       backspace=BS,SP,BS   ON
dc.b linedel   line del/bsp line     OFF
dc.b autoecho  full duplex          ON
dc.b autolf    auto line feed        ON
dc.b eolnulls  null count            0
dc.b pagpause  end of page pause     OFF
dc.b pagsize   lines per page        24
dc.b C$Bsp     backspace char        ^H
dc.b C$Del     delete line char      ^X
dc.b C$CR      end of record char     <return>
dc.b C$EOF     end of file char       ESC
dc.b C$Rprt    reprint line char     ^D
dc.b C$Rpet    dup last line char    ^A
dc.b C$Paus    pause char            ^W
dc.b C$Intr    Keyboard Interrupt char ^C
dc.b C$Quit    Keyboard Quit char    ^E
dc.b C$Bsp     backspace echo char   ^H
dc.b C$Bell    line overflow char    ^G
dc.b Parity    stop bits and parity   none
dc.b BaudRate  bits/char and baud rate none
dc.w EchoNam   offset of echo device  none
dc.b C$XOn     Transmit Enable char  ^Q
dc.b C$XOff    Transmit Disable char ^S
dc.b C$Tab     tab character          ^I
dc.b tabsize   tab column size        4
OptSiz equ *-Options

FileMgr dc.b "Scf",0  file manager

* Macro to generate main features of device descriptor
SCFDesc macro
    ifeq \#-8          EchoNam specified by macro
EchoNam dc.b "\8",0
    else
    ifeq \#-7          EchoNam equ bname
EchoNam equ bname
    else
    FAIL SCFDesc: must specify all 7 arguments
    endc
    endc

Port      equ \1 Port address
Vector    equ \2 autovector number
IRQLevel  equ \3 hardware interrupt level
Priority   equ \4 polling priority
Parity    equ \5 parity, stop bits
BaudRate  equ \6 baud rate
DevDrv    dc.b "\7",0 driver module name
    endm

Mode set ISize_+Updat_ default device mode capabilities

```

I ovenstående fil defineres alle felter i device descriptoren, på nær: Port address, autovector number, hardware interrupt level, polling priority, parity, baud rate, og navnet på device driveren.

Som det ses defineres der i bunden af filen en macro, der tager alle disse 7 oplysninger som parametre. Ved at bruge denne macro i en anden fil — som inkluderer ovenstående fil — kan man meget simpelt fremstille en device descriptor til SCF enheder. Herunder er vist et eksempel på en device descriptor der bruger ovenstående fil:

```
*
* Device descriptor module for dummy device: aa
*

        nam AA
        ttl AA device descriptor module

AA      macro
port          set 0
vector        set 27
IRQLev        set 3
IRQPri        set 5
parity        set $20
baud          set 14
        SCFDesc port,vector,IRQLev,IRQPri,parity,baud,driver
pagepause     equ OFF
DevCon        equ 0
        endm

        use defsfile
        use <scfdesc.d>
AA
        ends
```

Jeg kalder denne device descriptor for **aa**, så ovenstående fil lægges i mit SOURCE directory under navnet **aa.a**. Filen inkluderer udover **scfdesc.d** også en fil kaldet **defsfile**. Den ser således ud:

```
* this file determines the system to be built
opt        -l
use        <oskdefs.d>
use        <systype.d>
opt        1
```

Herefter kan device descriptoren **aa** assembleres vha:

```
r68 aa.a -o=../RELS/aa.r
```

Og linkes vha:

```
l68 ../RELS/aa.r -l=/dd/LIB/sys.l -o=../EXE/aa
```

24.4.3 Udvikling af device driver

En device driver er interfacet mellem file manager og hardware. Interfacet mellem file manager og device driver, består af en række funktioner i device driveren, som file manageren kan kalde. Hvilke funktioner, og hvilke parametre de bruger, bestemmes af file manageren.

En device driver til SCF file manageren, skal indeholde funktionerne: Init, Read, Write, GetStat, SetStat, og Term.

På PEPerne skrives device drivere i assembler, men på mere moderne OS-9 udviklingssystemer kan det meste af koden til en device driver udvikles i C. Nedenfor er kildeteksten til en devicedriver, jeg kalder **driver**. Kildeteksten lægges i SOURCE directoryet, under navnet: **driver.a**. Driveren er en tom skal. Alle funktionerne returnerer uden at gøre noget, bortset fra **Read**, der returnerer talkoden for tegnet **A**. Det

betyder at der altid vil læses en uendelig række A'er fra device driveren. Bortset fra detaljen med at returnere A'er, er eksemplet stort set identisk med eksempel fra [16, side 306], men med færre kommentarer for at spare plads.

```
* skeleton device driver
Typ_Lang      set      (Drivr<<8)+Objct
Att_Revs      set      ((ReEnt+SupStat)<<8)+0
*
Edition set    1
              psect    skeldrv,Typ_Lang,Att_Revs,Edition,0,EntryTable
              use      /dd/DEFS/oskdefs.d

* Static storage definitions
              vsect
IRQMask        ds.w    1
              ends

* Routine offset table
EntryTable     dc.w     Init
              dc.w     Read
              dc.w     Write
              dc.w     GetStat
              dc.w     SetStat
              dc.w     Term
              dc.w     0

Init           tst.w    d0          clear carry (no error)
              rts          Exit

Term           tst.w    d0
              rts

Read           move.b   #65,d0      Return 'A'
              tst.w    d0
              rts

Write          tst.w    d0
              rts

GetStat move.w   #E$UnkSvc,d1
              ori      #Carry,ccr
              rts

SetStat move.w   #E$UnkSvc,d1
              ori      #Carry,ccr
              rts

              ends
```

Eksemplet assemblres med:

```
r68 driver.a -o=../RELS/driver.r
```

Og linkes med:

```
../RELS/driver.r -l=/dd/LIB/sys.l -o=../EXE/driver
```

Det eneste der nu mangler for at kunne bruge I/O subsystemet beskrevet af device descriptoren: **aa**, er at sikre at alle elementerne ligge i hukommelsen. Som det første skal det sikres at de nye moduler/filer har

execute rettigheder:

```
attr -e ../EXE/*
```

SCF file manageren bruges af operativ systemet, og ligger i forvejen i hukommelsen. Device descriptor'en og device driveren kopieres til hukommelsen med henholdsvis:

```
load aa
load driver
```

Laver man ændringer i driver eller descriptor, skal man fjerne de gamle kopier fra hukommelsen, før de nye kopieres til den. Det gøres ved at gentage kommandoen:

`unlink driver` indtil der ikke er flere links til modulet, hvorefter kernen fjerner modulet fra hukommelsen. Kommandoen **mdir** giver en oversigt over moduler i hukommelsen.

Driveren kan f.eks. afprøves ved at give kommandoen:

```
dump -c /aa
```

Den endeløse strøm af A'er afbrydes igen med C-C.

Ovenstående device driver er en god start for eksperimenter med device drivere. For at lette eksperimenterne/udviklingen, er det en fordel at montere et VDOUT (16 digitale udgange) modul i PEPen. Modulet har 16 lysdioder i displayet, der kan styres vha. byte eller word *writes* til modulets basisadresse. Ved at skrive forskellige statuskoder til modulet, forskellige steder i koden, bliver det nemmere at debugge sin kode.

Modificeres linien:

```
port set 0
```

I **aa.a** til:

```
port set $873e0300
```

Hvor \$873e0300 er basisadressen for VDOUT modulet. Kan device driveren hente basisadressen til VDOUT fra device descriptor'en, i stedet for at lade basisadressen kode ind i kildeteksten til device driveren. Nedenfor er angivet hvordan Init funktionen i devicedriveren kan modificeres til at skrive word værdien $AAAA_{16}$, der tænder hver anden lysdiode, til porten når device driveren initialiseres.

Init	movea.l V_PORT(a2),a3	Get port address
	move.w #\$aaaa,(a3)	Set every other bit of port address
	tst.w d0	clear carry (no error)
	rts	Exit

24.4.4 Udvikling af file manager

24.5 Filoverførsel