CODEN: LUTFD2/(TFRT-5335)/1-57/(1985)

En drivrutin för kommunikation mellan ONSPEC och CLUSTER controller

Per-Anders Nilsson
Stefan Nilsson

Institutionen för Reglerteknik
Lunds Tekniska Högskola
September 1985

LUND INSTITUTE OF TECHNOLOGY
DEPARTMENT OF AUTOMATIC CONTROL
Box 118
S 221 00 Lund Sweden

Document name
Master thesis

Date of issue
September 1985

Document number
CODEN:LUTFD2/(TFRT-5335)/1-057/(1985)

Author(s)
Nilsson Per-Anders
Nilsson Stefan

Supervisor
Björn Wittenmark, Claes Rytoft

Sponsoring organization

Title and subtitle
En drivrutin för kommunikation mellan ONSPEC och CLUSTER controller.
(A driver for communication between ONSPEC and CLUSTER controller.)

Abstract

The intention of this thesis was to build a driver between a Cluster Controller, built by ASEA ILKL, and a IBM PC/XT. The Cluster Controller acts as a cross communication master in the communication with one or more of ASEA MP100 process computers, and as a slave in the communication with the IBM PC/XT. The communication protocol that the Cluster Controller was implemented for in the communication with the PC was a subset of Excom which is described in appendix 2.

We also had at our disposal a program package for making process pictures which is called Onspec and is built by Heuristics Inc. ,California. This program also connects certain values on the process picture to certain places in special data tables. It is possible to access this data tables from another task and that was one thing our driver was supposed to do. The driver was also supposed to convert all the information that is received and sent on the line, into understandable types for Onspec and the Cluster Controller.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

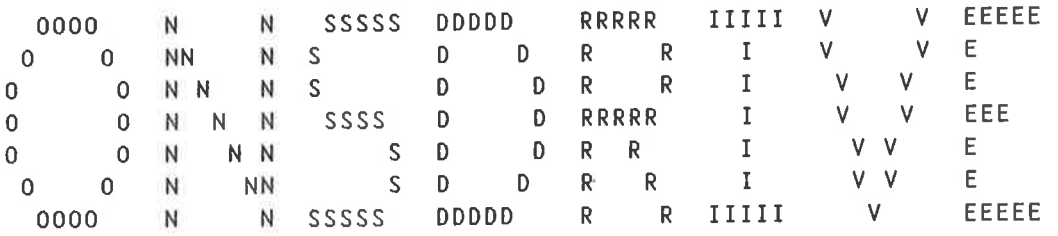
ISBN

Language
Swedish

Number of pages
57

Recipient's notes

Security classification



En drivrutin för kommunikation mellan ONSPEC och Cluster Controller

AV

Per-Anders Nilsson

Stefan Nilsson

HANDLEDARE

Claes Rytoft

Björn Wittenmark

FÖRORD.

Vi skulle gärna vilja tacka de anställda på ASEA ILKL för att de,trots att det vi sysslat med varit nytt för dem,gett oss många goda råd och konstruktiv kritik.

Lund 1985-09-05

Stefan Nilsson

Per-Anders Nilsson

INEHÅLLSFÖRTECKNING.

1. INLEDNING.

Bakgrunden till vårt examensarbete är den att ASEA's Master-view-system för små företag ter sig ganska dyrt.Tanken var då att till en persondator koppla ett programpaket som understödjer byggandet av processbilder samt inehåller alarmfunktioner, trendfunktioner och historik.Kommunikationen mellan detta programpaket och yttervärlden skulle då skötas av en drivrutin. Det var denna som vårt examensarbete gick ut på att skriva.

Yttervärlden bestod av en Cluster Controller byggd av ASEA. Denna kan kort beskrivas som en busshanterare för insamling av mätvärden till/från ett antal processdatorer MP100. För ytterligare information om Cluster Controllern hänvisas till kapitel 3.1.

Kommunikationsprotokollet som denna busshanterare använde sig av var en delmängd av EXCOM protokollet se bilaga 2.

Program paketet som inköptes var gjort av Heuristics Inc, Kalifornien och heter ONSPEC. ONSPEC uppfyllde alla de önskemål som ovan nämnts. För en beskrivning av ONSPEC hänvisas till kapitel 4 .

Detta program är gjort för att köras mot operativsystemet Concurrent PC/DOS. Det är ett multitasksystem som möjliggör att fyra program kan köras "parallellt" enligt round robin principen. Detta för att kunna köra ONSPEC i en task och drivrutinen i en annan. Ytterligare information om Concurrent PC/DOS finns i kapitel 5.

Vårt arbete var att skriva ovannämnda drivrutin för att erhålla ett komplett system, så att man kan avläsa och ändra värden i processdatorerna från en externdator.

2. UPPSTÄLLDA KRAV PÅ DRIVRUTINEN.

De krav på drivrutinen som vi till en början hade att rätta oss efter var följande:

* Baudrate: 2400 Baud
* Drivrutinen skall kunna på ett snyggt sätt ta omhand felmeddelanden
* Timeoutfunktion
* Fast koppling mellan datasetnummer och platser i Onspec's tabeller
* Att vid tre omsändningar av ett skrivmeddelande dumpa detsamma
* Xon/Xoff, det vill säga att vid mottaget tecken=Xoff så skall drivrutinen sluta sända tills man fått Xon
* Valbar uppdateringstid av dataset från clustern
* Alla meddelanden skall innehålla checksumma
* 1 stoppbit
* Udda paritet
* Inga startbitar

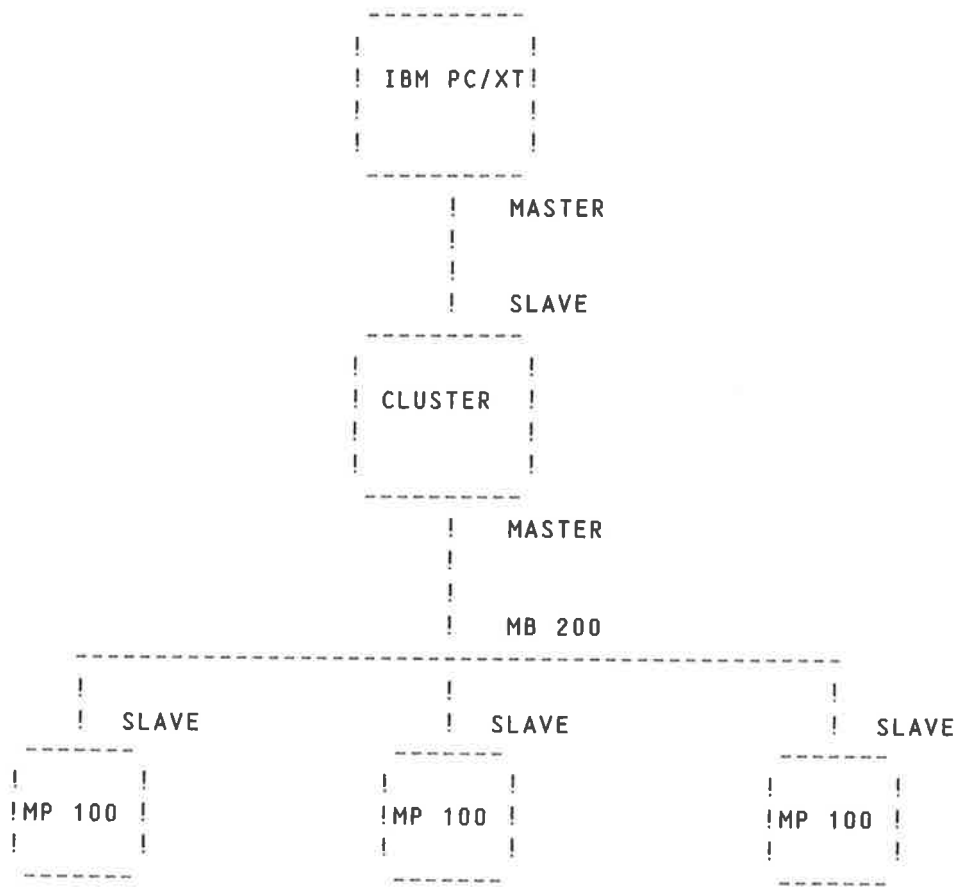
Efter hand har framkommit önskemål om valbar baudrate.Detta har vi inte implementerat.

3.1 CLUSTER CONTROLLER.

Cluster controllern är en korskommunikationsmaster för MP100 systemet. Enheten sammanbinder ett flertal processdatorer MP100 via en MB200 bus till ett komplett system där cluster controllern är master mot MP100-miljön. Kommunikationen mellan noderna administreras genom att polla varje MP100 och beroende på svaret från slavnoden skicka det vidare till destinationsnoden.

Data som mottagits från en slavnod kommer ,om den är addreserad till Cluster Controllern,att lagras i dennas interna dataarea. Dessa data kan sedan nås från en externenhet via en delmängd av EXCOM-protokollet, se bilaga 2. Det är just inom detta sista användningsområde som vi har arbetat med Cluster Controllern.

Hela systemet får då följande utseende:



3.2 KOMMUNIKATION MELLAN CLUSTER CONTROLLERN OCH EN EXTERN DATOR.

Kommunikationen sker med hjälp av dataset där de efterfrågade dataseten sänds ut från respektive MP100 och speglas in i cluster controllerns interna dataarea. Dessa dataset kan i sin tur nås frånen extern dator via en delmängd av EXCOM-protokollet, se bilaga 2.

Dataarean rymmer totalt 128 dataset och dessa har följande uppdelning:

1- 40	32 bit boolean
41- 90	real
91-128	32 bit integer

Varje dataset innehåller fem värden (record 1-5). För real och integer innebär detta att ett dataset innehåller 5 värden medan för boolean ett dataset innehåller 5 * 32 = 160 st logiska värden.

Syntaxen för kommunikation med cluster controllern är den samma för såväl kommando- som svarsmeddelanden. Ett meddelande innehåller enbart ASCII-tecken och inga små bokstäver är tillåtna. Som styrtecken används LF i början och CR i slutet av ett meddelande.

Varje svarsmeddelande innehåller information om hur överföringen från den externa datorn till cluster controllern lyckades.

Det finns fem olika typer av kommunikationsinformation:

* OK Överföringen gick bra parametrarna överensstämde.
* WG Varning alla data var ej konverterbara.
* US Odefinierat dataset i cluster controllerns data area.
* IC Ej tillåtet kommando.
* IF Felaktigt format på utsänd data.

BEGRÄNSNINGAR.

* Cluster controllern kan enbart användas i lokalt nätverk.
* Maximalt antal noder är 10.
* Endast en extern dator kan kopplas in.

För vidare information om Cluster Controllern hänvisas till K.Westling ASEA avd ILKL.

4. BESKRIVNING AV ONSPEC.

Onspec är ett programvarupaket framtaget av Heuristics Inc, Kalifornien.Programvaran har till uppgift att understödja byggandet av processbilder,vilka schematiskt skall beskriva en process.Bilderna byggs upp med semigrafik,det vill säga man ritar med tecken till skillnad från full grafik där man använder pixel.Till programvaran medföljer ett speciellt teckenprom som innehåller ett ökat antal tecken för att man skall ha större valfrihet när man bygger bilder.

Programvaran understödjer också att man till bilderna kan knyta värden ur speciella datatabeller som Onspec bildar.De tabeller som vi har använt oss av har följande utseende:

```
EUR  :  array (1..512) of real;

EUI  :  array (1..512) of integer;

DII  :  array (1..512) of char;

DOO  :  array (1..512) of char;
```

Med integer menas i detta sammanhang tvåbytes integer.I tabellerna DII och DOO lagras logiska värden.

Onspec's interface utåt är två "systemköer" ,genom vilka program i andra tasks kan kommunicera med tabellerna.Dessa kallas för READQUE och WRITEQUE.Köerna är till utseendet exakt lika och innehåller poster med följande relevanta information:

```
QUETYPE = record
            .
            .
            PREFS:char;
            INDEXS:integer;
            VALU:real;
            VALUS:char;
            .
            .
          end;
```

PREFS anger vilken kö värdet kommer ifrån:

```
EUR =>  PREFS= chr(1)

EUI =>  PREFS= chr(2)

DII =>  PREFS= chr(6)

DOO =>  PREFS= chr(7)
```

INDEXS anger vilket index i respektive tabell som värdet ligger på.

VALU anger värdet.Även heltal representeras som decimaltal i posten.

VALUS: Om PREFS är antingen chr(6) eller chr(7) så anger VALUS om den logiska variabeln på positionen INDEXS är ON eller OFF.ON representeras som chr(0) och OFF som chr(2).

När operatören gör en ändring av t ex ett börvärde i en process, så lägger Onspec börvärdet,index i tabellen samt ett värde som anger vilken tabell det är frågan om i en post som i sin tur läggs i WRITEQUE.

Motsvarande:Om det finns något i READQUE så tar Onspec och lägger in värdet på på rätt plats i rätt tabell.

Onspec har också alarmfunktion,trendrepresentation samt viss historik implementerad.För en mer ingående beskrivning av dessa, och hur man bygger bilder hänvisas till Onspec User Reference Manual.

5. KORT BESKRIVNING AV CONCURRENT PC/DOS.

Det operativsystem som Onspec är skrivet för heter Concurrent PC/DOS.Det är uppbyggt så att man kan köra fyra tasks samtidigt enligt round robin principen.Operativsystemet skall vara helt kompatibelt med vanlig PC/DOS.Vi har dock inte testat om det verkligen gäller. Concurrent PC/DOS har en viktig skillnad från vanlig PC/DOS nämligen att det finns ett utökat antal DOS-funktioner. Dessa behandlar de systemköer som används för att sköta intertaskkommunikationen.Det går dock inte att hitta dessa funktioner i manualen, utan man får gå till Concurrent CP/M manualen för att få reda på vad de olika funktionerna utför!

För användaren ter sig Concurrent PC/DOS vid första anblicken ganska tjusigt.Man får olika menyer att välja på och sedan är det bara att med hjälp av funktionstangenter förflytta sig inom menyn och utföra vissa kommando.De menyer som står till buds är:

* CARDFILE
* DREDIX
* DRTALK
* FILE MANAGER
* PRINTER MANAGER

CARDFILE är ett program med vars hjälp man kan lägga upp ett kortbibliotek med elektroniska kort.

DREDIX är en vanlig skärmeditor.

DRTALK understödjer kommunikation med andra datorer.

FILE MANAGER lägger upp de vanligaste kommandona i en meny. Man kan sedan utföra dessa kommandon genom att trycka på funktionstangenter.Exempel på vanliga kommandon är DREDIX,RUN,COPY.

PRINTER MANAGER understödjer att man kan skriva ut en eller flera filer på en eller flera printrar.

Efter hand som man använder operativsystemet så blir man mer och mer irriterad på att det inte går att förflytta sig mellan olika directories med hjälp av ett vanligt kommando.Man måste använda FILE MANAGER. Detta gör att det tar mycket längre tid att förflytta sig i systemet för det är ganska stora fördröjningar innan de nya menyerna kommit upp.

6. ARBETSGÅNG OCH ÖVERVÄGANDEN VID KONSTRUKTION AV DRIVRUTINEN.

Vi inledde vårt exjobb med att studera operativsystemet Concurrent PC/DOS. Detta operativsystem är ett multitasksystem där man kan köra fyra tasks "parallellt". Tilldelningen av CPU tid sker enligt round robin principen. Se kapitel 5.

Manualen som medföljer systemet är helt användarinriktad och är inte till någon nytta vid programmeringsarbete.

Persondatorn som vi använt är en IBM PC/XT som har ett primärminne på 640 kilobyte och en winchester på 10 megabyte.Datorn har två asynkrona kommunikationskort för extern kommunikation som är programmerbara, se kapitel 8. Beskrivningen av hur man programmerar dessa studerade vi speciellt på grund av att detta var en central del av konstruktionsarbetet. En beskrivning av korten finns i bilaga 4 .

Vid studier av Onspecs manualer fann vi egentligen ingenting om hur man sköter kommunikationen till Onspecs datatabeller från ett annat program.Den enda dokumentation som fanns var ett skelett till en drivrutin .Detta skelett finns i bilaga 1.

När vi studerade Onspecs källkod fann vi två program som kommunicerade med varandra.De använde sig av Bdoskommandon vilka är ett slags fördefinierade funktioner som är inbakade i operativsystemet.En del av dessa funktioner utför intertaskkommunikation via något som kallas systemköer. Onspec använder två sådana köer READQUE och WRITEQUE.

Vår taktik var nu att försöka modifiera ett av programmen så att man ifrån det kunde ändra i Onspecs tabeller.Vi fortsatte sedan med att läsa de ändringar som Onspec lade i WRITEQUE.

Nu började vi studera den externa kommunikationen,det vill säga kommunikationen med Cluster Controllern.Vi gjorde små avbrottsrutiner för att lära oss programmera de asynkrona kommunikationskorten.Den viktiga komponenten i dessa kort är ett asynkront kommunikationselement INS8250.(se bilaga 4) För att kunna testa avbrottsrutinerna så kopplade vi en Tandberg VT100 terminal till korten.Vi har gjort avbrottsrutinen sådan att man skickar ut ett tecken per cykel av programmet.Detta för att man snabbare skall kunna detektera tecknet Xoff som anger när Cluster Controllern inte kan ta emot fler tecken.Då stoppas våran utmatning tills Clustern skickat tecknet Xon.Inläsning går alltså alltid före utmatning.

När vi kopplade ihop avbrottsrutinen med det program som skötte kommunikationen med Onspec uppstod problem på grund av att alla variabler som användes i avbrottsrutinen måste vara absolutdeklarerade,och vi kunde inte veta var i minnet som Onspec lade sina variabler.Vi fick systematiskt genomsöka minnet tills vi hittade ledig plats.

Nu började vi göra upp programstrukturen för den "riktiga" drivrutinen.Subrutinernas funktion definierades strikt och vi delade upp dem mellan oss.Programkoden är skriven i Pascal.

När vi kodat,kompilerat och länkat samman ONSDRIVE började vi testköra mot Cluster Controllern.Efter en del modifieringar fungerar nu ONSDRIVE enligt alla krav.

Sammanfattningsvis kan man säga att det största problemet i konstruktionsarbetet har utan tvekan varit den bristfälliga dokumentationen av Onspec och av operativsystemet.

7.1 BESKRIVNING AV FUNKTIONEN HOS PROCEDURERNA I ONSDRIVE.

INFORM: Proceduren ger användaren information på bildskärmen om hur ONSDRIVE skall köras.

SHOW : Proceduren läser av filen TEMPFILE som innehåller de från förra gången definierade uppdateringarna. Dessa värden läggs in i CHECKQUE för att möjliggöra ändring av de förut definierade värdena samt för att proceduren CHECKTOSEND skall veta vilka datasetnummer som är definierade för uppdatering.
Dessutom visar proceduren på monitorn de i CHECKQUE lagrade datasetnummren med tillhörannde nodnummer detta för att operatören skall kunna se vilka dataset som definierats.

INTCHQ: Proceduren är en ren initieringsprocedur som anropas varje gång Onsdrive uppstartas. Dess uppgift är att möjliggöra för operatören att göra ändringar i CHECKQUE om så önskas. Information om hur detta går till skrivs ut på monitorn. Då operatören är nöjd lagras den aktuella CHECKQUE i TEMPFILE för att underlätta en uppstart av systemet.

CHECK-
TOSEND: Proceduren går igenom CHECKQUE och formaterar de datasetnummer som är definierade av proceduren INTCHQ till korrekta EXCOM meddelande. Dessa meddelande som är läskommando läggs sedan in i SENDQUE som är den kö där meddelande för utsändning lagras. Till SENDQUE finns en pekare READREQ som pekar ut var i SENDQUE som det sista läskommandot ligger lagrat, detta återkommer vi till senare.

BUFFHAND: Proceduren går igenom bufferten RECBUFF i vilken avbrottsrutinen lägger in alla tecken som den tar emot. BUFFHAND letar igenom bufferten efter ett helt meddelande enligt EXCOM protokollet. Då ett sådant hittats läggs det in på första lediga plats i RECQUE och proceduren letar efter ytterligare kompletta meddelande i RECBUFF tills inga fler finns.

REC-
EIVE : Då det finns något meddelande i RECQUE tar proceduren det första och kontrollerar dess checksumma om denna är fel kastas meddelandet. I annat fall undersöks vilken typ av meddelande det rör sig om, läsdataset-svarsmeddelande eller skrivdataset-svarsmeddelande.
Vid ett läsdataset går man in i SENDQUE och undersöker vilket meddelande innan READREQ vars identifierare överensstämmer med läsdatasetets. Därmed har det mottagna meddelandets address bestämts. Genom att konvertera från dataset nummer till index nummer i ONSPEC's data tabeller får man reda på var det mottagna värdet skall läggas in. Därefter formatteras värdet för att passa systemköerna och läggs in i READQUE.
För ett skrivdataset jämförs det mottagna meddelandets identifierare med identifierarna för de meddelande som sparats i SAVE. Då identifierarna överensstämmer plockas det aktuella meddelandet bort från SAVE.
Detta förfarande fortgår tills inga fler meddelande lagrats i RECQUE.

WQTO-
SENDQ : Om det finns någon post i WRITEQUE så tar WQTOSEND denna, formaterar den till ett korrekt EXCOM meddelande och lägger detta på första lediga plats efter READREQ i SENDQUE.

ACTIVATE-
SEND : Aktiverar avbrottsrutinen för sändning då något av följande villkor är uppfyllda:

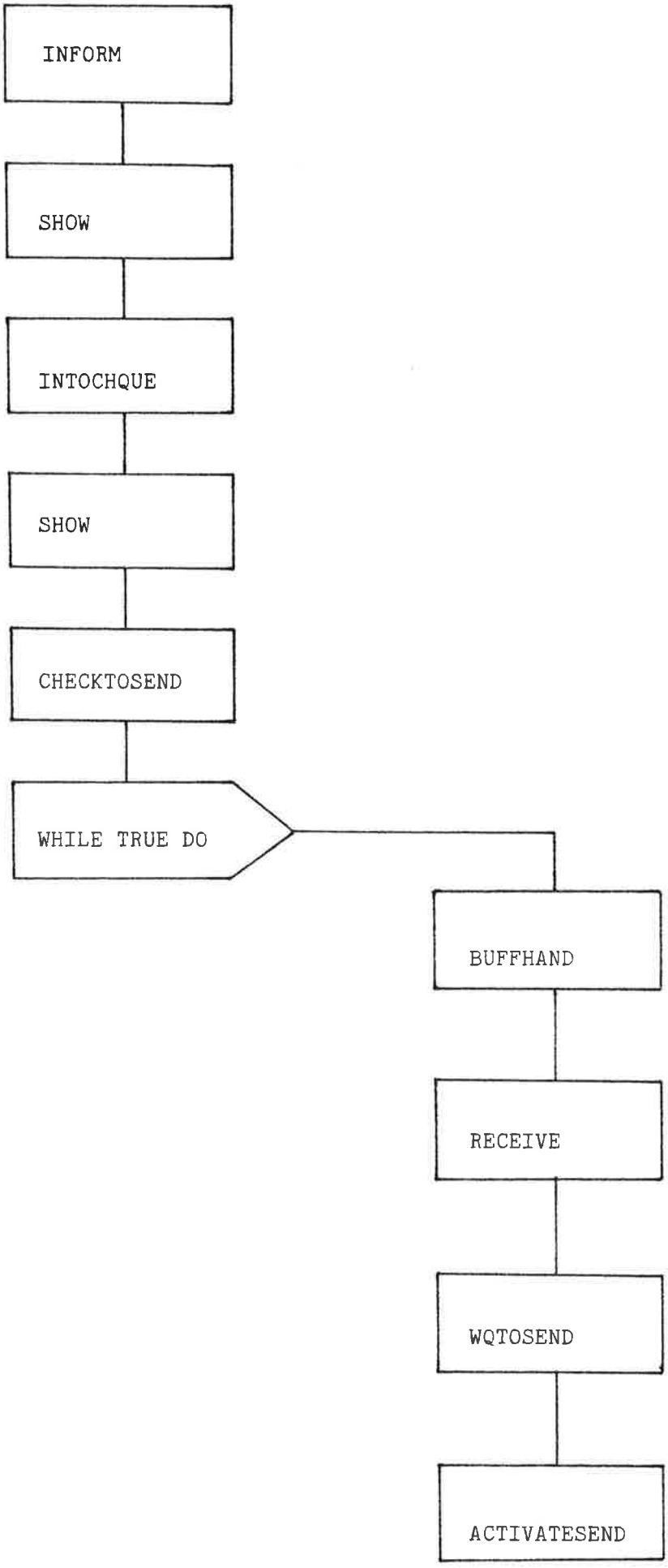
* Ett meddelande håller på att sändas ut.
Avbrottsrutinen har blivit aktiverad på grund av att något av nedanstående villkor är uppfylld och ett eller flera tecken har redan sänts ut.

* Det har blivit timeout på ett meddelande i SAVE

* Det är dags att sända läsförfrågan på de dataset som operatören valt att uppdatera

* Det finns ett eller flera skrivmeddelande i SENDQUE. ACTIVATESEND lagrar alla skrivmeddelande som sänds ut i SAVE.

Varje varv av programmet sänds ett tecken ut, efter det att ett tecken sänts stängs avbrottsrutinen av för utsändning och aktiveras nästa gång om något av ovanstående villkor är uppfyllda.


INFORM
SHOW
INTOCHQUE
SHOW
CHECKTOSEND
WHILE TRUE DO
BUFFHAND
RECEIVE
WQTOSEND
ACTIVATESEND

8. INITIERING AV DE ASYNKRONA KOMMUNIKATIONSKORTEN.

En av de mest centrala delarna i vår drivrutin är avbrottsrutinen. Det är mycket viktigt att man initierar det asynkrona kommunikationskortet som man väljer på rätt sätt,samt att man i avbrottsrutinen kontrollerar vad som orsakat avbrottet.

Alla de register som i fortsättningen nämns finns detaljerat beskrivna i bilaga 4.

Det första man gör är att välja vilket asynkront kort man vill aktivera. Genom att nollställa speciella bitar i ett register på adress 21H görs detta val. Om man som vi vill aktivera det andra asynkrona kortet så nollställer man bit 3 i registret.

För att få rätt teckenlängd,rätt paritet,rätt antal stoppbitar osv så finns ett Line Status Register (LSR) tillgängligt för programmeraren på adress 02FBH.

Överföringshastigheten bestämmer man genom att initiera registerna Divisor Latch MSB och Divisor Latch LSB. Dessa ligger på adresserna 02F9H oh 02F8H. Man kan välja hastighet från 50 Baud till 9600 Baud.
OBS! För att kunna sätta önskad överföringshastighet så måste bit 7 i Line Control Register vara ettställd. Alltid annars skall bit 7 vara nollställd.

Man initierar och sätter igång avbrotten genom att sätta olika bitar i Interrupt Enable Register på adress 02F9.De bitar som är viktigast är bit 0 och bit 1.Om man ettställer bit 0 så får man ett avbrott när ett tecken kommer på linjen. Detta avbrott har vi alltid aktiverat på grund av att vi inte vill missa några tecken från Cluster Controllern.Om man ettställer bit 1 så får man ett avbrott när sändbufferten på det asynkrona kortet är tom.Detta gör att man bara kan initiera "sändavbrott" när man är absolut säker på att det finns något att sända, annars kommer man kontinuerligt att ha sändavbrott på grund av att sändbufferten alltid är tom.

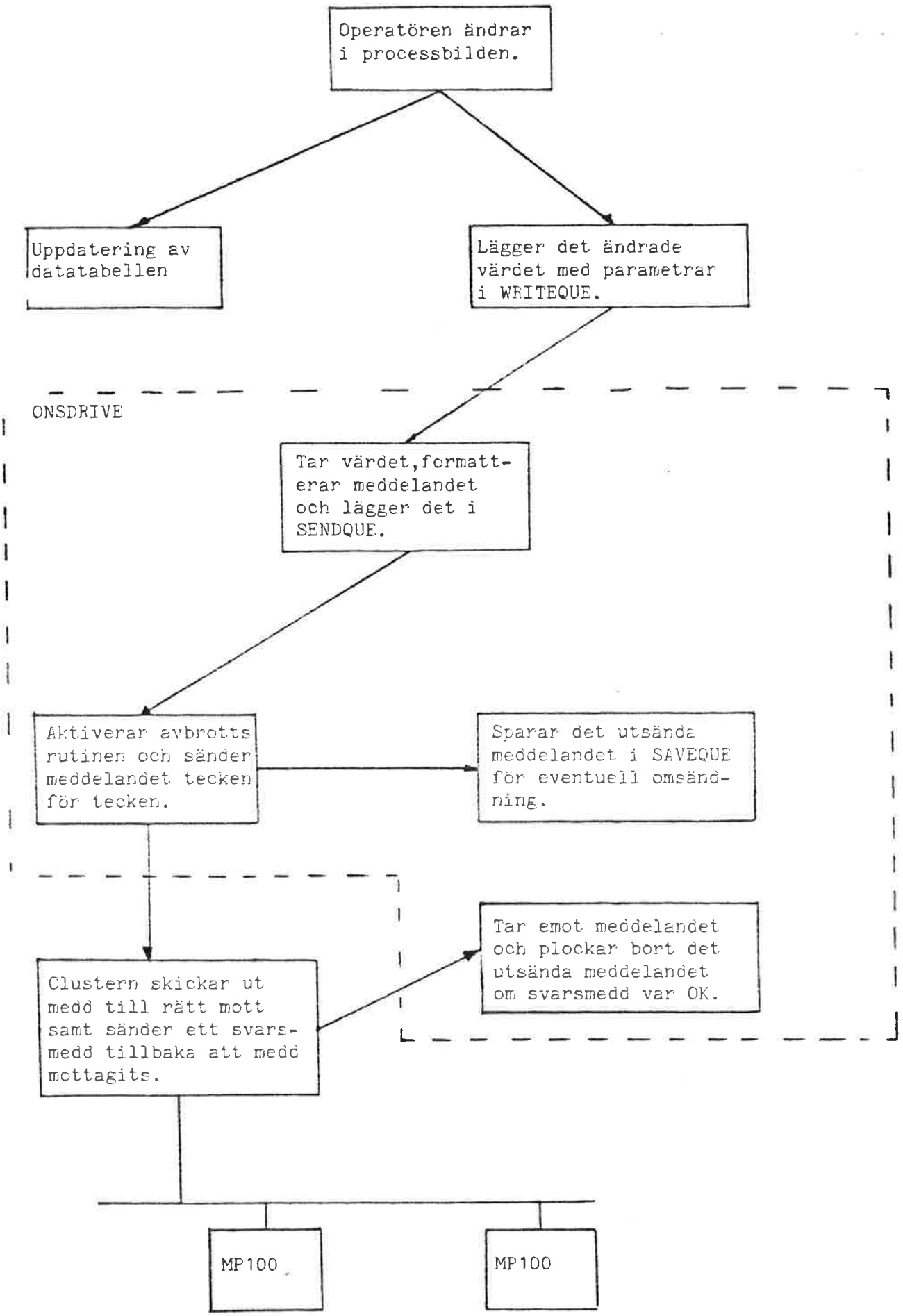
9.1 KOMMENTARER TILL BLOCKSCHEMORNA I KAP 9.2 OCH 9.3 .

De blockschema som finns i detta kapitel symboliserar händelseförloppet vid två olika händelser i drivrutinen. Då man ändrar i processbilden ,kap 9.2, samt då det är dags att läsa av de olika mätvärdena ute i processen ,kap 9.3.

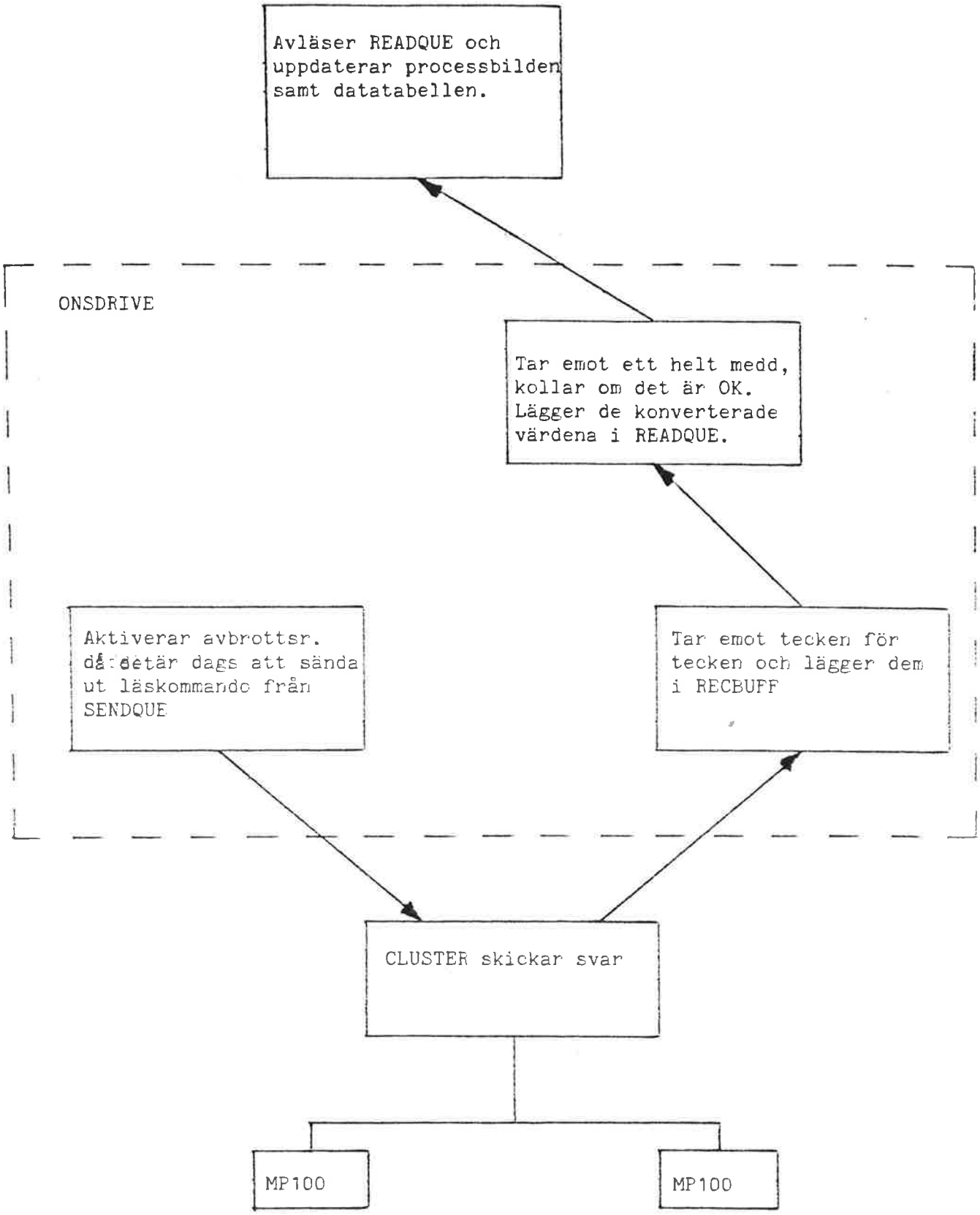
Kapitel 9.2 behandlar det fall då operatören ändrar något värde i processbilden på bildskärmen och detta värde skall skickas ut till processen.
Följande moment gås då igenom i drivrutinen ONSDRIVE. Proceduren WQTOSEND läser av det nya värdet som då ligger i systemkön WRITEQUE och formaterar detta till ett korrekt EXCOM meddelande. Informationen om mottagarens adress hämtas från CHECKQUE och läggs till huvudet i EXCOM meddelandet. Därefter läggs meddelandet i SENDQUE tills dess att proceduren ACTIVATESEND upptäcker det och aktiverar avbrottsrutinen för utskickning av tecken. Denna rutin sparar också meddelandet i SAVE, läser av systemklockan och lagrar denna tillsammans med meddelandet för att hålla reda på omsändningstiden, samt nollställer en räknare för att begränsa antalet omsändningar till tre. Sedan sänds ett tecken ut för varje cykel av programmet.
Då ett helt meddelande sänts ut skickar Cluster Controllern svarsmeddelande som upplysning om att den tagit emot ett meddelande. Detta meddelande mottages tecken för tecken av avbrottsrutionen för inläsning och lagras i RECBUFF för vidare hantering. Proceduren BUFFHAND letar igenom bufferten och söker efter slutet på ett meddelande. Då detta hittats kopieras meddelandet över till RECQUE. Denna kö behandlas av RECEIVE ,som då ett meddelande läggs in i kön kontrollerar dess checksumma och vilken typ av meddelande det är. I detta fallet ett skrivsvarsmeddelande ,och om detta är OK ,kommer den sparade kopian att plockas bort ur SAVE. I annat fall kommer inget att inträffa utan man sänder om meddelandet efter en viss tidsperiod.

Kapitel 9.3 behandlar det fall då en tidsperiod har förflutit och det är dags att sända ut ett läskommando. Den rutin som administrerar detta är ACTIVATESEND. Avbrottsrutinen för utskickning aktiveras på ett liknande sätt som i föregående exempel med den skillnaden att istället för att bara sända ut det senaste meddelandet i SENDQUE så skickas alla meddelandena ut. Den parameter som pekar på gränsen mellan läskommando och skrivmeddelande i SENQUE kallas READREQ.Utsändning och mottagning av tecken sker på samma sätt som förut men nu är de mottagna meddelandena lässvarsmeddelande. Proceduren RECEIVE behandlar dem på liknande sätt som förut, men tar nu och lägger in de mottagna mätvärdena i systemkön READQUE vilken sedan ONSPEC avläser.

9.2 HÄNDELSEFÖRLOPP VID ÄNDRING AV PROCESSBILDEN



9.3 HÄNDELSEFÖRLOPP VID UPPDATERING FRÅN CLUSTERN.



10.1 ATT TÄNKA PÅ VID BILDBYGGNAD.

Vid byggandet av en bild är det viktigt att men har klart försig hur ONSPECs datatabeller är knutna till cluster controllerns datasets. Vi har knutit sammman dessa båda viktiga parametrar på följande sätt.

CLUSTER CONTROLLER			ONSPEC	
DATASET	1	REC 1	DII	1- 11
	1	2		12- 22
	1	3		:
	1	5		45- 55
	2	1		56- 66
	:	:		:
	2	5		100-110
	:	:		:
	:	:		:
	9	5		485-495
DATASET	10	REC 1	DOO	1- 11
	:	:		:
	10	5		45- 55
	:	:		:
	:	:		:
	18	5		485-495
DATASET	41	REC 1-5	EUR	1- 5
	42	1-5		6- 10
	:	:		:
	90	1-5		246-250
DATASET	91	REC 1-5	EUI	150-154
	92	1-5		155-159
	:	:		:
	128	1-5		335-339

För logiska dataset dvs nummer 1-18 gäller följande:
De 11 minst signifikanta bitarna i varje record är aktiva, där den minst signifikanta biten är matchad mot det lägsta tabellelementet innom respektive record. De övriga bitarna är matchade i stigande ordning tills den mest signifikanta biten av de 11 tillhör det högsta tabellelementet.

Vad som gäller för analoga dataset framgår klart och tydligt ur tabellen ovan. För ytterligare information om ONSPEC's datatabeller se kapitel 4. beskrivning av ONSPEC. Begreppet dataset förklaras i kapitel 3.2 .

10.2 ALARM VID KOMMUNIKATIONSFEL

Det är möjligt att erhålla alarminformation från drivern om något har gått fel i kommunikationen med cluster controllern. Genom att då ONSPEC och ONSDRIVE är startade gå över i den task där ONSPEC går mata in kommandot ALMLOD för att på så sätt ladda in alarmet. I och med detta erhålles ett alarm varje gång drivern tar emot ett felmeddelande från cluster controllern. Detta sker genom att då man befinner sig i ONSPEC's task kommer en statusrad att visas i botten av skärmen med texten

'WARNING COMMUNICATION ERROR CHANGE'

Detta innebär att man skall byta task till den där ONSDRIVE kör för där får man mer information om vad som gått fel. Observera att statusraden inte försvinner då man noterat vad som var fel, utan

den skrivs bara om på nytt då ett nytt alarm inträffar. Vill man att en ljudsignal skall signalera då ett alarm inträffar är det bara att i den tredje tasken starta programmet ALTCON.

OBS!

Det finns en display och ett alarmnummer som är upptagna av ONSDRIVE. Det är alarmnummer 256 och display 96.Dessa används för att åstadkomma alarm vid kommunikationsfel.

10.3 KÖRNING AV ONSDRIVE.

Onsdrive förklarar i princip själv hur man skall köra.Det är dock några saker man bör tänka på.

På frågan "DO YOU WANT TO UPDATE THE SAME DATASETS AS LAST TIME Y/N ?" så MÅSTE man svara "Y" om man vill ha kvar några dataset från förra körningen.Svarar man "N" så försvinner hela uppdateringsfilen och man får skriva in alla uppdateringarna igen!

Om man vill ändra skriver man alltså "Y" och sedan tar man bort respektive lägger till önskade datasetnummer och nodnummer.

Man startar drivern genom att skriva "ONSDRIVE" och man stänger av den genom att skriva "CTRL-C".

OBS!

Man måste starta om Onspec varje gång man vill starta drivern! Detta beror på att när Onspec startas så lägger Onspec in alla adresserna till tabellerna i en speciell kö.Det första Onsdrive gör när den startas är att beta av den kön.Om Onspec inte startats upp innan drivern så kan alltså inte Onsdrive få adresserna och följdaktligen inte heller bearbeta tabellerna.

11. SLUTSATSER

När vårt examensarbete nu är färdigt kan man ge synpunkter på det.

För det första har vi operativsystemet Concurrent PC/DOS som vi är ganska missnöjda med. Till att börja med är det CPU tilldelningen som sker enligt round robin principen vilket omöjliggör prioritering. Dessutom är det besvärligt att jobba med på grund av menyhanteringen, all filhantering går via menyer och detta blir ganska irriterande i längden.

Därefter har vi programpaketet ONSPEC som innehåller alla de funktioner som krävs för processtyrning. Dock är programmet klumpigt att jobba med när man byggt en bild och skall knyta denna till datatabellerna. Det är en ansenlig mängd parametrar som skall fyllas i. Grafiken som programmet genererar är semigrafik vilket gör att upplösningen inte blir vad man skulle kunna önska sig. En annan svaghet i programmet är uppdateringsttiden som har ett minsta intervall på 10 sekunder.

Vad som återstår är vår drivrutin ONSDRIVE som fungerar enligt önskemålen och uppfyller de krav som uppställts i kapitel 2.

De som är intresserade av att fördjupa sig i ONSDRIVES källkod eller är intresserade av Cluster Controllern hänvisas till Claes Rytoft ASEA avd ILKL.

BILAGA 1.

Skelett till en drivrutin hämtat ur ONSPEC's manualer.

As a guide consider the following procedures. The OUT command has the address output to to the right of the assign (:=) and the data to be output to the left in the brackets [ ]. Reference the Pascal and IBM Tech Manual as necessary.

```
PROGRAM PROCIO; (* SKELETON SERIAL DRIVER *)
CONST
     SECOND = $0B; (* SOFTWARE INTERRUPT FOR SECONDARY ASYNCH *)
(*$I B:PROCIO.LIB *) (* DEFINES GLOBALS ETC *)
VAR
      VECOB : ABSOLUTE [0:$02C] IPTR;

PROCEDURE UNCOM2; (* PROCEDURE TO UNMASK SECONDARY INTERRUPT *)
BEGIN
      IN21 := INP [ ($21) ]; (* INTERRUPT 8259A PORT *)
      IF TSTBIT (IN21,3) THEN BEGIN
         CLRBIT (IN21,3); OUT [(IN21)] := $21; END;
END;

PROCEDURE MASCOM2; (* PROCEDURE TO MASK SECONDARY INTERRUPT *)
BEGIN
                  INLINE ($FA); (* DISABLES INTERRUPTS *)
                  IN21 := INP[ ($21) ];
                  SETBIT (IN21,3);
                  OUT [(IN21)] := $21;
                  INLINE ($FB);  (* ENABLES INTERRUPTS *)
END;

PROCEDURE INTERRUPT [ SECOND ] INT2;
BEGIN

(*** IMPORTANT ALL VARIABLES USED MUST BE ABSOLUTE ***)
(*** SEE CPASCAL.LIB FOR EXAMPLE OF THESE ***)
(*** MEMORY ABOVE 752K IN AMDEK OR OTHER COLOR CARD MAKES
     A GOOD ABSOLUTE MEMORY SCRATCH AREA                ***)

(* A COINCIDENCE THAT $20 IS THE ACK DATA FOR $20 TO THE 8259A *)
    OUT[($20)] := $20; (* ACKNOWLEDGE INTERRUPT TO 8259A *)
    INTBYTE := INP [($02F8)]; (* BYTE IN *)
    INTSTAT  := INP [($02FD)]; (* STAT IN *)
    ITNEXT := KBIT +1;
    IF  ITNEXT > 15 THEN ITNEXT := 1;  (* CIRCULAR BUFFER 15 CHARACTERS
                                          LONG. LONGER FOR MOST PROCESS I/O *)
    IF ITNEXT <  1 THEN ITNEXT := 1;
    IF  ITNEXT <> KBITP THEN
      BEGIN
      ININT[ITNEXT] := INTBYTE;
      KBIT := ITNEXT;
      END;

END;
```

```
PROCEDURE INSISC;
VAR
   PTRTD : IPTR;
   IDELAY, IDELAYR : INTEGER;
BEGIN
IF NOT ( KBIT  IN[1..15]) THEN KBIFLG := TRUE;
IF NOT (KBITP  IN[1..15]) THEN KBIFLG := TRUE;
IF KBIFLG THEN
BEGIN
(* KBIFLG USED TO INITIALIZE INTERRUPT AND COMM2 PORT *)
  KBIFLG := FALSE;
  INLINE ($FA);
  VECOB := ADDR (INT2);
  (* UNMASK ASYNCH INTERRUPT *)
  UNCOM2; (* NOT OBVIOUS FROM IBM DOCUMENTATION *)
  (* ENABLE INTERRUPT *)
  KBITP := 1;KBIT := 1;
  INLINE ($FB);
(* SECONDARY SERIAL PORT INITIALIZATION *)

  OUT [($01)] := $02F9; (* INITIALIZE 8250 ASYNCH COMM ELEMENT *)
  OUT[($08)] := $02FC;  (* INTIIALIZE 8250 ASYNCH COMM ELEMENT *)
END;
(* INITIALIZE EACH TIME INSISC  *)
  INSS := '';  IB:=0; WRAP := FALSE; (* USE YOUR OWN MESSAGE VAR *)
IDELAY := 15;  (* 60 PER SECOND; USE YOUR OWN TIMING *)
REPEAT
  MOVE(IDELAY,PTRTD,2);
  IDELAYR := @BDOS86(141,PTRTD)
(* TIME OUT LOGIC AND EXIT HERE *)
  UNTIL  KBIT <> KBITP ;
(* SUSPEND INTERRUPTS WHILE RECOMPUTING CIRCULAR BUFFER  POINTERS *)
  INLINE  ($FA);
  KBITP := KBITP +1 ;
  IF KBITP > 15 THEN KBITP := 1;
  IF KBITP < 1  THEN KBITP := 1;
  INBYTE := ININT[KBITP];
  INLINE ($FB);
  LOOKATIT (* CONSTRUCT MESSAGE COMING IN *)
(* UNTIL EG: CARRIAGE RETURN,  LENGTH OF 10 CHARS, OR LOGICAL END *)
  UNTIL (INBYTE = C13) OR (LENGTH(INSS)>= 9) OR WRAP;
(* THIS WAY WHEN MESSAGE IS BUILT *)
  INLIN := INSS; (* STRING WITH MESSAGE; USE YOUR OWN VAR *)
END;
BEGIN (* PROCESS I/O MAINLINE *)
(* MAINLINE FOR DRIVER *)
(* PROCESS MESSAGES PUTTING DATA INTO ONSPEC TABLES *)
(* IMPLEMENT PROTOCOL FOR YOUR PROCESS I/O
(* CONSOLE TAKES DATA FROM ABSOLUTE LOCATIONS DEFINED IN DRIVER
   THAT CORRESPOND TO ONSPEC TABLES  *)
END.  (* PROCESS I/O MAINLINE *)
```

An alternative to the above approach is to use the serial queues supplied with the operating system to obtain the messages coming in and going out.

BILAGA 2.

Funktionsskiss över den del av EXCOM protokollet som använts vid kommunikationen.

FUNKTIONSSKISS EXCOM

1 ALLMÄNT

EXCOM i clustercontrollern möjliggör kommunikation med externa datorer. Kommunikationen har följande karakteristik:

* Master/slave förhållande med den externa datorn som master.
* Datautbyte i form av kommandon och svar på kommandon.
* Asynkron seriell kommunikation (RS232, V24).
* Full duplex.
* Dataöverföring i ASCII-kod.

Följande kommandon är möjliga:

* Läs dataset
* Skriv dataset

Endast en extern dator kan anslutas till clustercontrollern.

2 DATA

2.1 Syntax

Samma syntax används i kommando- och svarsmeddelandet. Meddelandet får bara innehålla ASCII-tecken. Små bokstäver är ej tillåtna.

Frånsett meddelandehuvudet måste alla data vara åtskillda av blanktecken. Meddelandet skall börja med ett radmatningstecken <LF> och sluta med vagnretur <CR>.

2.2 Meddelandehuvud

Huvudet startar i position 2 i meddelandet direkt efter <LF>-tecknet. Huvudet har en fix storlek på nio tecken.

position

1 : riktning '<' indikerar kommandomeddelande (från extern dator till EXCOM-funktionen)
'>' indikerar svarsmeddelande (från EXCOM-funktionen till den externa datorn)

2,3	: ww	'DD'	DS 101 nätverknummer. Anger vilket nätverk som dataset skall läsas från eller skrivas till. (Alltid noll, lokalt nätverk)
4,5	: nn	'01'..'99'	DS 101 nodnummer. Anger vilken nod som dataset skall läsas från eller skrivas till.
6,7	: id	2 ASCII	meddelandeidentifierare, 2 ASCII-tecken med värden i intervallet 20(H) till 50(H). Detta är användarens identifierare som skickas tillbaks oförändrad i svarsmeddelandet.
8,9	: kommando	2 ASCII	Två stora bokstäver, 'RD' eller 'WR', som svarar mot kommandot läs resp. skriv dataset.
	eller		
	tillstkod	2 ASCII	Två stora bokstäver som indikerar resultatet av givet kommando i svarsmeddelandet. (se resp. kommando)

Exempel huvud:

<DDD317RD	Läs i nod 3 i anslutet nätverk. Meddelandeidentitet = 17
>DDD2ZKOK	Svarsmeddelande från nod 2 med tillståndskod = 'OK' Meddelandeidentitet = ZK

2.3 Meddelande med checksumma

I den externa datorn finns möjligheten att utöka testfaciliteterna med en checksummekontroll av meddelandet.

Om den externa datorn sänder ett kommandomeddelande med teckensträngen &xy<CR>, kommer EXCOM att utföra en checksummekontroll av det mottagna meddelandet. Om meddelandet är korrekt accepteras det, annars ignoreras det.

Kommandon innehållande denna kontroll genererar ett svar från EXCOM med en checksumma. Svaret avslutas enligt följande: &xy<CR>.

Teckenförklaring:

&	Indikerar att checksumma är bifogad.
xy	2 ASCII-tecken 'DD'..'FF', checksumman.
<CR>	Vagnretur = 0D(H).

Beräkningen av checksumman börjar med det första tecknet i huvudet ('<') och slutar med tecknet '&'. Se exemplet nedan.
Beräkningen utförs som modulo 256 addition av teckens numeriska (hex-) värden. Checksumman är de 8 minst signifikanta bitarna av den totala teckensumman.

Checksumman konverteras till två hexadecimala ASCII-tecken, 'DD'..'FF'.

Exempel checksummeberäkning

<000233RD -44&D5 Läs dataset 44. Checksum = D5. Checksumman är korrekt

tecken	hex
'<'	3C
'0'	30
'0'	30
'0'	30
'2'	32
'3'	33
'3'	33
'R'	52
'D'	44
' '	20
'-'	2D
'4'	34
'4'	34
'8'	26
SUM	2D5

2.4 Dataformat

* 32 BOOLEANS — representeras vid datautbyte av ASCII-koden för motsvarande hexadecimala värde. De inledande nollorna trunkeras inte av EXCOM men den externa datorn kan utesluta dem i ett skrivkommando. Datat skall alltid inledas av ett '#'-tecken.
  Exempel '#F34C34AC'

* INTEGERLONG — representeras vid datautbyte av motsvarande ASCII-siffror. EXCOM trunkerar inledande nollor. Negativa värden skall inledas med tecken, positiva kan inledas med tecken.

* REAL — består av 24 bitar tecken och signifikand samt 8 bitar exponent. Syntaxen skall följa reglerna för standard-pascal.
  Vid datautbyte är antalet signifikanta siffror begränsat till 6. Datat får endast vara representerat av 12 ASCII-tecken.

  +/-__<heltalsdel>__'.'__<decimaldel>__'E'__+/-__<exponent>__

2.5 Datasetformat

I EXCOM får ett dataset endast bestå av en datatyp. Datatyperna är knutna till datasetnummer enligt följande:

* -41 - -90 — REAL
* -1 - -40 — 32 BOOLEANS

* -91 - -128 INTEGERLONG

Det maximala antalet datasets som kan knytas till cluster controllern är 128. Se kapitel 8 Begränsningar.

3 ADRESSERING

* NETW — Nätverk specifieras i huvudet och anges alltid till noll. Se Konfigurering (kap 6).

* NODE — Nodnummer specifieras i huvudet och anger i vilken nod som önskat dataset befinner sig.

* DATASET — Datasetnummer anges vid läs- och skrivkommandon och är alltid negativt ('-128'..'-1'). Minustecken skall alltså ingå i meddelandet.

OBS! — Observera att det negativa datasetnummer som anges i kommandot från den externa datorn, refererar till motsvarande positiva elementnummer i PC-elementen EXT-1 och EXT-0.

4 KOMMUNIKATION

4.1 Sub protokoll

Kommunikationen kan ses som uppdelad i två delar:

A) Datautbyte EXTERN DATOR <--> CLUSTER CONTR

B) Datautbyte CLUSTER CONTR <--> MP100

EXCOM arbetar mot en i cluster controllern uppspänd dataarea.
Denna innehåller de (max 128) datasets som adresserats till clusternoden.
Dataseten uppdateras med fördefinierad cyklisitet.
Vid skrivkommando från den externa datorn lägger EXCOM det mottagna datasetet på "skrivplats" i dataarean. Clustern distribuerar det sedan vidare till angiven adress i nätverket.

4.2 Kommunikationskarakteristik

Kommunikationen har följande karakteristik:

- Konstant master/slave förhållande med den externa datorn som master.
- Datautbyte genom kommandon och svar på kommandon.
- Asynkron, seriell kommunikation (RS232, V24).

- Full duplex
- Dataöverföring med ASCII-tecken.
- Överföringshastighet 2400 baud.
- Meddelandet börjar med ASCII-tecknet <LF> (0A(H)) och slutar med ASCII-tecknet <CR> (0D(H)).
- Felhantering genom att ignorera mottagna meddelanden med detekterade fel (ex. checksummefel). Se 5.3 Felhantering.

Svarsmeddelanden kommer inte i någon definierad ordning. Det är därför lämpligt att öka värdet på meddelandeidentifieraren för att möjliggöra identifikation av meddelandena. Se 2.2 Meddelandehuvud.

5 KOMMANDO- OCH SVARSMEDDELANDE

5.1.1 RD - Läskommando

- Läser dataset

Data läses och konverteras enligt datasetformatet. Se 2.5 Datasetformat.

Om datat ej är konverterbart kommer dess plats att ersättas av ett antal ASCII-stjärnor '**********' i svarsmeddelandet.

Läskommandots syntax:

<LF>__<wwnndiRD>__' '__<datasetnummer>__<CR>

<LF>	=	Line Feed 0A(H)
<wwnndiRD>	=	Huvud
' '	=	Blanktecken
<datasetnummer>	=	('-128'..'-1')
<CR>	=	Carriage Return 0D(H)

Exempel:

Läs dataset 3

Kommando: <000182RD -3
Svar : >000182WG 1.23168E2 1.78345E10 **********

Svaret indikerar att ett värde ej var konverterbart.

5.1.2 RD - Svarsmeddelande

Ok:
<LF>__>wwnndiOK__' '___<data>____<CR>

```
WARNING:                '********'
<LF>__>wwnndiWG__' '___<data>____<CR>


Error message:
<LF>__>wwnndicc___<CR>
```

Möjliga tillståndskoder i svarsmeddelandet:

```
'OK'        = Felfritt svarsmeddelande
'WG'        = Warning, alla data var ej konverterbara.
'US'        = Undefined set. Angivet dataset är ej knutet till
              clustercontrollern.
'IC'        = Illegal command. Syntaxfel i kommandot.
<timeout>   = Se kapitel 5.3 Felhantering.
```

5.2.1 WR - Skrivkommando

- Skriver data i angivet dataset

Det är endast möjligt att skriva till de datasets som dessutom är knutna till cluster controllern för läsning.

Värdena konverteras enligt datasetformatet. Detta medför att använt format måste överensstämma med det specifierade (se 2 Dataformat). Om inte, indikeras ett feltillstånd i svarsmedmeddelandet.

Antalet parametrar skall överensstamma med formatet för motsvarande dataset. Om inte, indikeras ett feltillstånd i svarsmeddelandet.

Om ett feltillstånd inträffar kommer ingen av parametrarna i datasetet att uppdateras.

Skrivkommandots syntax:

```
<LF>___<wwnndiWR___' '___<datasetnummer>___' '___<data>___<CR>

<LF>              = Line Feed 0A(H)
<wwnnidWR         = Huvud
' '               = Blanktecken
<datasetnummer>   = ('-128'..'-1')
<data>            = data som skrives i angivet dataset
<CR>              = Carriage Return 0D(H)
```

5.2.2 WR - Svarsmeddelande

```
<LF>___>wwnndicc___<CR>
```

Möjliga tillståndskoder i svarsmeddelandet:

```
'OK'        = Felfritt meddelande. (OBS! Indikerar endast att datasetet placerats
              i cluster controllerns interna dataarea för vidare distribuering)
'US'        = Undifined set. Angivet dataset är ej knutet till
```

cluster controllern.
'IF' = Illegal format. Felaktigt antal parametrar eller formatet överensstämmer ej med det specifierade (2.5 Datasetformat)
'IC' = Illegal command. Syntaxfel i kommandot.
<timeout> = Se kapitel 5.3 Felhantering

Exempel:

Skriv data i dataset 112, meddelandeidentitet 'ZK'

Kommando: <0004ZKWR -112 12 0 11
Svar : >0004ZKOK

5.3 Sammanfattning, felhantering

EXCOM ignorerar meddelandet om följande fel föreligger:

- Checksumman är felaktig (om bifogad).
- Nätverk- och nodtecken i huvudet är inte ASCII-siffror.
- Felaktig meddelandelängd.
- Felindikering från drivrutinen.

Ovanstående fel förväntas detekteras som "internal timeout" av den externa datorn.

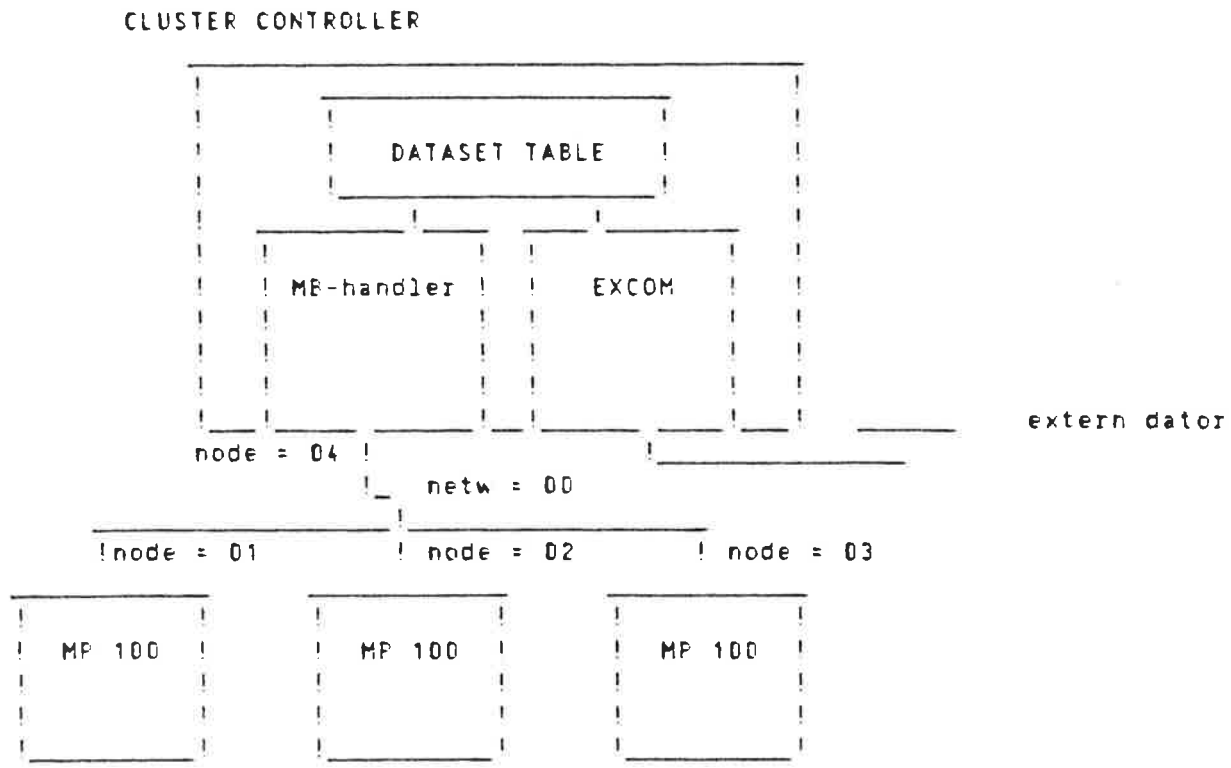
EXCOM returnerar ett svarsmeddelande med felstatus om följande fel föreligger:

- Om kommandokoden inte är någon av 'RD' och 'WR'.
- Om antalet parametrar ej överensstämmer med syntaxen.
- Om meddelandeformatet — " " —
- Om angivet dataset ej är knutet till clustercontrollern

EXCOM returnerar ett svarsmeddelande med varningstatus om fel föreligger:

- Om ett data ej är konverterbart.

6 KONFIGURERING



7. SNITT EXCOM / BUS HANDLER

I cluster controllerns RWM-minne läggs en dataset-tabell upp med plats för 128 dataset samt en "sök"-tabell med uppgift om på vilken adress i dataset-tabellen man kan finna önskat dataset. Till varje dataset bifogas en byte med uppgift om antalet parametrar i datasetet. I denna byte ingår även en statusbit. Då denna är satt får buss-hanteraren ej uppdatera datasetet.

Då ett dataset skickas från en MP100 med clustern som mottagare uppdateras (ev. utvidgas) dataset-tabellen.

Då ett skrivkommando från en extern dator ges uppdateras tabellen på motsvarande sätt och excom-rutinen ger kommandot "sänd" till buss-hanteraren som i sin tur hämtar aktuellt dataset i tabellen.

Då ett läskommando från en extern dator ges hämtar excom-rutinen aktuellt dataset i tabellen.

8. BEGRÄNSNINGAR

Följande restriktioner gäller vid kommando från extern dator:

- Antalet datasets som kan knytas till cluster controllern är maximalt 128. (Fel markeras med lysdiod på clusterns front)
- Antalet parametrar skall överensstämma med formatet för motsvarande dataset. Max antal parametrar = 5.
- Ett dataset får endast bestå av en datatyp. Datatyperna är knutna till datasetnummer enligt kapitel 2.5.
- Det är endast möjligt att skriva i ett dataset som dessutom är knutet till clustern för läsning.
- Meddelandelängden får ej överstiga 79 bytes (= maximal längd för ett kommando).

Beräkning av största möjliga meddelandelängd:

```
<LF>+<HUVUD>+<MAX ANT PARAM>*(<SPACE>+<REAL>)+<CHECKSUM>+<CR>
  1 +   9   +        5       *(   1   +  12  )+    3     +  1

  10 + 5*13 + 4 = 79
```

BILAGA 3.

Definition av dom kötyper som använts i drivrutinen.

```
type STR2= string[2];
     STR4= string[4];
     STR15= string[15];
     STR30= string[30];
     STR83= string[83];
     PTR= ^integer;
     POINTER= ^STR83;
     RECTYPE= record
                 QUE: array[1..RECMAX] of POINTER;
                 BORJAN,SLUT: integer;
              end;
     SENDTYPE= record
                  QUE: array[1..SENDMAX] of POINTER;
                  BORJAN,SLUT: integer;
               end;
     SAVETYPE= record
                  QUE: array[1..SAVEMAX] of POINTER;
                  NUM_OF_RES: array[1..SAVEMAX] of integer;
                  BORJAN,SLUT: integer;
               end;
     TIMETYPE= record
                  QUE: array[1..TIMEMAX] of INTEGER;
                  BORJAN,SLUT: integer;
               end;
     VEKTOR1=array[1..512] of real;
     VEKTOR2=array[1..512] of integer;
     VEKTOR6=array[1..512] of char;
     VEKTOR7=array[1..512] of char;
     TABLETYPE= record
                   TAB1_PTR:record case integer of
                              0:(EUR_PTR:^INTEGER);
                              1:(YOURPTR:^VEKTOR1);
                            end;
                   TAB2_PTR:record case integer of
                              0:(EUI_PTR:^INTEGER);
                              1:(YOURPTR:^VEKTOR2);
                            end;
                   TAB6_PTR:record case integer of
                              0:(DII_PTR:^INTEGER);
                              1:(YOURPTR:^VEKTOR6);
                            end;
                   TAB7_PTR:record case integer of
                              0:(DOO_PTR:^INTEGER);
                              1:(YOURPTR:^VEKTOR7);
                            end;
                end;
    CHQUETYPE=array[1..128] of record
                                 LAS:boolean;
                                 NODNR:STR2;
                               end;
```

BILAGA 4.

IBM's asynkrona kommunikationskort med initierings register.

IBM Asynchronous Communications Adapter

The asynchronous communications adapter system control signals and voltage requirements are provided through a 2 by 31 position card edge tab. Two jumper modules are provided on the adapter. One jumper module selects either RS-232C or current-loop operation. The other jumper module selects one of two addresses for the adapter, so two adapters may be used in one system.

The adapter is fully programmable and supports asynchronous communciations only. It will add and remove start bits, stop bits, and parity bits. A programmable baud rate generator allows operation from 50 baud to 9600 baud. Five, six, seven or eight bit characters with 1, 1-1/2, or 2 stop bits are supported. A fully prioritized interrupt system controls transmit, receive, error, line status, and data set interrupts. Diagnostic capabilities provide loopback functions of transmit/receive and input/output signals.

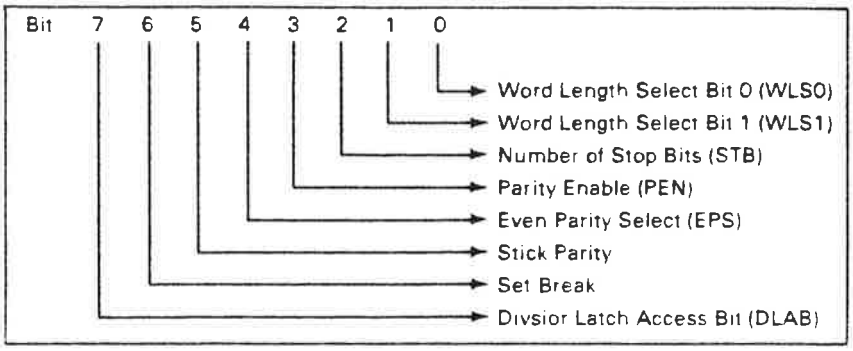
The heart of the adapter is a INS8250 LSI chip or functional equivalent. Features in addition to those listed above are:

- Full double buffering eliminates need for precise synchronization.
- Independent receiver clock input.
- Modem control functions: clear to send (CTS), request to send (RTS), data set ready (DSR), data terminal ready (DTR), ring indicator (RI), and carrier detect.
- False-start bit detection.
- Line-break generation and detection.

All communications protocol is a function of the system microcode and must be loaded before the adapter is operational. All pacing of the interface and control signal status must be handled by the system software. The figure below is a block diagram of the asynchronous communications adapter.

Line-Control Register

The system programmer specifies the format of the asynchronous data communications exchange through the line-control register. In addition to controlling the format, the programmer may retrieve the contents of the line-control register for inspection. This feature simplifies system programming and eliminates the need for separate storage in system memory of the line characteristics. The contents of the line-control register are indicated and described below.



Line-Control Register (LCR)

Bits 0 and 1: These two bits specify the number of bits in each transmitted or received serial character. The encoding of bits 0 and 1 is as follows:

Bit 1	Bit 0	Word Length
0	0	5 Bits
0	1	6 Bits
1	0	7 Bits
1	1	8 Bits

Bit 2: This bit specifies the number of stop bits in each transmitted or received serial character. If bit 2 is a logical 0, one stop bit is generated or checked in the transmit or receive data, respectively. If bit 2 is logical 1 when a 5-bit word length is selected through bits 0 and 1, 1-1/2 stop bits are generated or checked. If bit 2 is logical 1 when either a 6-, 7-, or 8-bit word length is selected, two stop bits are generated or checked.

Bit 3: This bit is the parity enable bit. When bit 3 is a logical 1, a parity bit is generated (transmit data) or checked (receive data) between the last data word bit and stop bit of the serial data. (The parity bit is used to produce an even or odd number of 1's when the data word bits and the parity bit are summed.)

Bit 4: This bit is the even parity select bit. When bit 3 is a logical 1 and bit 4 is a logical 0, an odd number of logical 1's is transmitted or checked in the data word bits and parity bit. When bit 3 is a logical 1 and bit 4 is a logical 1, an even number of bits is transmitted or checked.

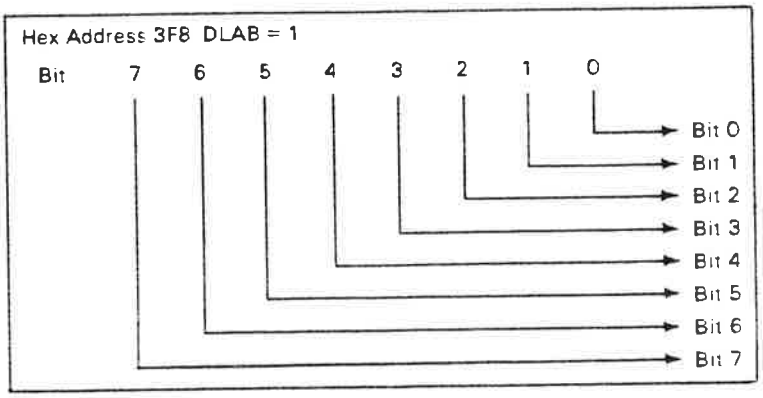
Bit 5: This bit is the stick parity bit. When bit 3 is a logical 1 and bit 5 is a logical 1, the parity bit is transmitted and then detected by the receiver as a logical 0 if bit 4 is a logical 1, or as a logical 1 if bit 4 is a logical 0.

Bit 6: This bit is the set break control bit. When bit 6 is a logical 1, the serial output (SOUT) is forced to the spacing (logical 0) state and remains there regardless of other transmitter activity. The set break is disabled by setting bit 6 to a logical 0. This feature enables the processor to alert a terminal in a computer communications system.

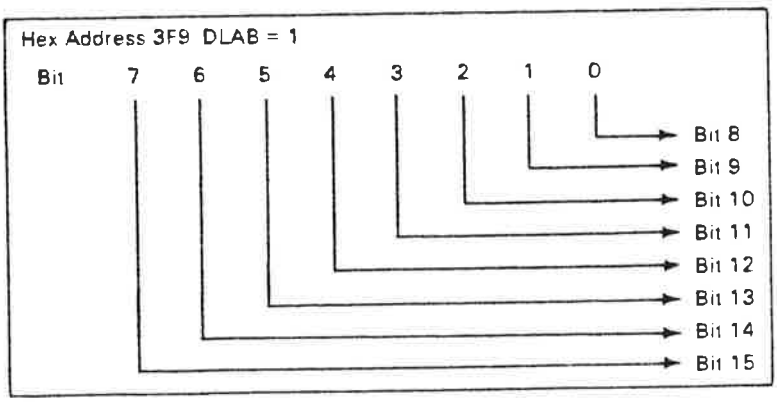
Bit 7: This bit is the divisor latch access bit (DLAB). It must be set high (logical 1) to access the divisor latches of the baud rate generator during a read or write operation. It must be set low (logical 0) to access the receiver buffer, the transmitter holding register, or the interrupt enable register.

Programmable Baud Rate Generator

The INS8250 contains a programmable baud rate generator that is capable of taking the clock input (1.8432 MHz) and dividing it by any divisor from 1 to (2^{16}-1). The output frequency of the baud generator is 16 x the baud rate [divisor # = (frequency input)/(baud rate x 16)]. Two 8-bit latches store the divisor in a 16-bit binary format. These divisor latches must be loaded during initialization in order to ensure desired operation of the baud rate generator. Upon loading either of the divisor latches, a 16-bit baud counter is immediately loaded. This prevents long counts on initial load.



Divisor Latch Least Significant Bit (DLL)



Divisor Latch Most Significant Bit (DLM)

The following figure illustrates the use of the baud rate generator with a frequency of 1.8432 MHz. For baud rates of 9600 and below, the error obtained is minimal.

Note: The maximum operating frequency of the baud generator is 3.1 MHz. In no case should the data rate be greater than 9600 baud.

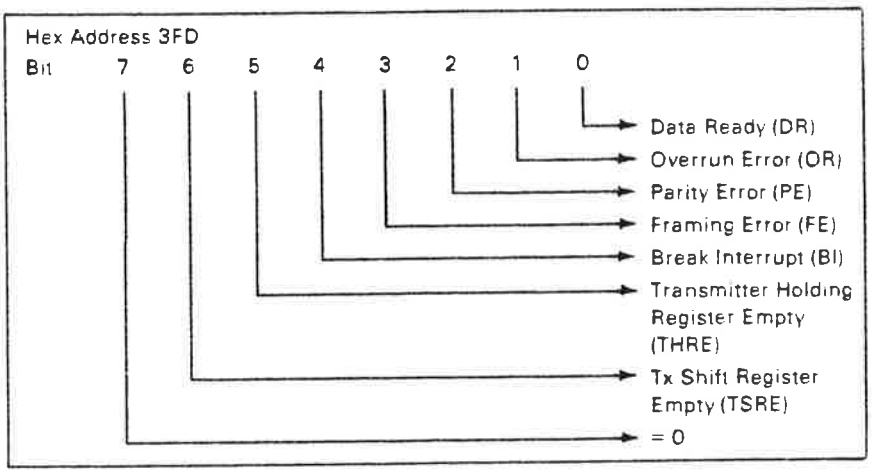
Desired Baud Rate	Divisor Used to Generate 16x Clock (Decimal)	(Hex)	Percent Error Difference Between Desired and Actual
50	2304	900	—
75	1536	600	—
110	1047	417	0.026
134.5	857	359	0.058
150	768	300	—
300	384	180	—
600	192	0C0	—
1200	96	060	—
1800	64	040	—
2000	58	03A	0.69
2400	48	030	—
3600	32	020	—
4800	24	018	—
7200	16	010	—
9600	12	00C	—

Baud Rate at 1.843 MHz

1-200 Asynchronous Adapter

Line Status Register

This 8-bit register provides status information on the processor concerning the data transfer. The contents of the line status register are indicated and described below:



Line Status Register (LSR)

Bit 0: This bit is the receiver data ready (DR) indicator. Bit 0 is set to a logical 1 whenever a complete incoming character has been received and transferred into the receiver buffer register. Bit 0 may be reset to a logical 0 either by the processor reading the data in the receiver buffer register or by writing a logical 0 into it from the processor.

Bit 1: This bit is the overrun error (OE) indicator. Bit 1 indicates that data in the receiver buffer register was not read by the processor before the next character was transferred into the receiver buffer register, thereby destroying the previous character. The OE indicator is reset whenever the processor reads the contents of the line status register.

Bit 2: This bit is the parity error (PE) indicator. Bit 2 indicates that the received data character does not have the correct even or odd parity, as selected by the even parity-select bit. The PE bit is set to a logical 1 upon detection of a parity error and is reset to a logical 0 whenever the processor reads the contents of the line status register.

Bit 3: This bit is the framing error (FE) indicator. Bit 3 indicates that the received character did not have a valid stop bit. Bit 3 is set to a logical 1 whenever the stop bit following the last data bit or parity is detected as a zero bit (spacing level).

Bit 4: This bit is the break interrupt (BI) indicator. Bit 4 is set to a logical 1 whenever the received data input is held in the spacing (logical 0) state for longer than a full word transmission time (that is, the total time of start bit + data bits + parity + stop bits).

Note: Bits 1 through 4 are the error conditions that produce a receiver line status interrupt whenever any of the corresponding conditions are detected.

Bit 5: This bit is the transmitter holding register empty (THRE) indicator. Bit 5 indicates that the INS8250 is ready to accept a new character for transmission. In addition, this bit causes the INS8250 to issue an interrupt to the processor when the transmit holding register empty interrupt enable is set high. The THRE bit is set to a logical 1 when a character is transferred from the transmitter holding register into the transmitter shift register. The bit is reset to logical 0 concurrently with the loading of the transmitter holding register by the processor.

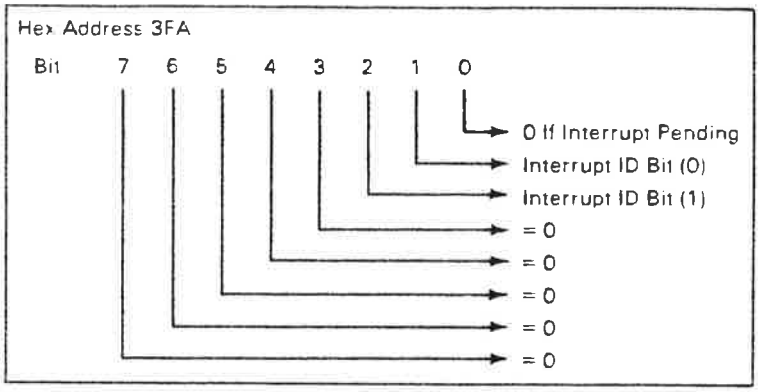
Bit 6: This bit is the transmitter shift register empty (TSRE) indicator. Bit 6 is set to a logical 1 whenever the transmitter shift register is idle. It is reset to logical 0 upon a data transfer from the transmitter holding register to the transmitter shift register. Bit 6 is a read-only bit.

Bit 7: This bit is permanently set to logical 0.

Interrupt Identification Register

The INS8250 has an on-chip interrupt capability that allows for complete flexibility in interfacing to all the popular microprocessors presently available. In order to provide minimum software overhead during data character transfers, the INS8250 prioritizes interrupts into four levels: receiver line status (priority 1), received data ready (priority 2), transmitter holding register empty (priority 3), and modem status (priority 4).

Information indicating that a prioritized interrupt is pending and the type of prioritized interrupt is stored in the interrupt identification register. Refer to the "Interrupt Control Functions" table. The interrupt identification register (IIR), when addressed during chip-select time, freezes the highest priority interrupt pending, and no other interrupts are acknowledged until that particular interrupt is serviced by the processor. The contents of the IIR are indicated and described below.



Interrupt Identification Register (IIR)

Bit 0: This bit can be used in either a hard-wired prioritized or polled environment to indicate whether an interrupt is pending and the IIR contents may be used as a pointer to the appropriate interrupt service routine. When bit 0 is a logical 1, no interrupt is pending and polling (if used) is continued.

Bits 1 and 2: These two bits of the IIR are used to identify the highest priority interrupt pending as indicated in the "Interrupt Control Functions" table.

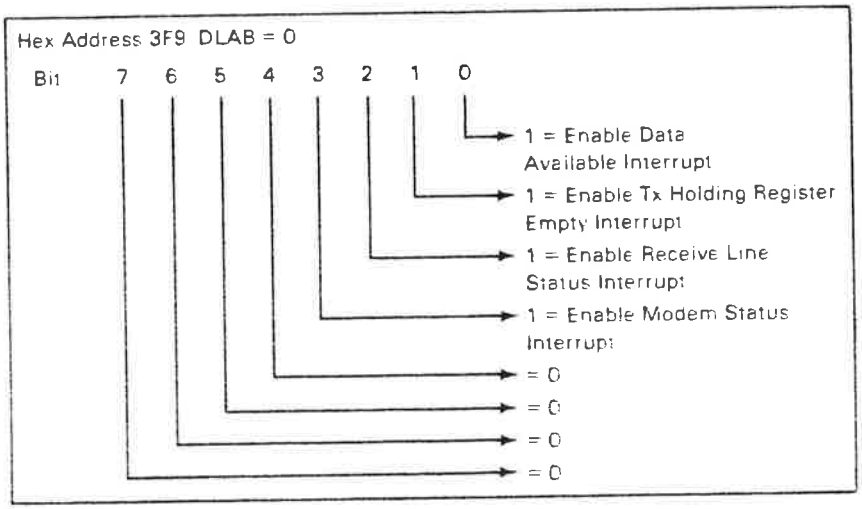
Bits 3 through 7: These five bits of the IIR are always logical 0.

Interrupt ID Register				Interrupt Set and Reset Functions		
Bit 2	Bit 1	Bit 0	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Control
0	0	1		None	None	
1	1	0	Highest	Receiver Line Status	Overrun Error or Parity Error or Framing Error or Break Interrupt	Reading the Line Status Register
1	0	0	Second	Received Data Available	Receiver Data Available	Reading the Receiver Buffer Register
0	1	0	Third	Transmitter Holding Register Empty	Transmitter Holding Register Empty	Reading the IIR Register (if source of interrupt) or Writing into the Transmitter Holding Register
0	0	0	Fourth	Modem Status	Clear to Send or Data Set Ready or Ring Indicator or Received Line Signal Detect	Reading the Modem Status Register

Interrupt Control Functions

Interrupt Enable Register

This eight-bit register enables the four types of interrupt of the INS8250 to separately activate the chip interrupt (INTRPT) output signal. It is possible to totally disable the interrupt system by resetting bits 0 through 3 of the interrupt enable register. Similarly, by setting the appropriate bits of this register to a logical 1, selected interrupts can be enabled. Disabling the interrupt system inhibits the interrupt identification register and the active (high) INTRPT output from the chip. All other system functions operate in their normal manner, including the setting of the line status and modem status registers. The contents of the interrupt enable register are indicated and described below:



Interrupt Enable Register (IER)

Bit 0: This bit enables the received data available interrupt when set to logical 1.

Bit 1: This bit enables the transmitter holding register empty interrupt when set to logical 1.

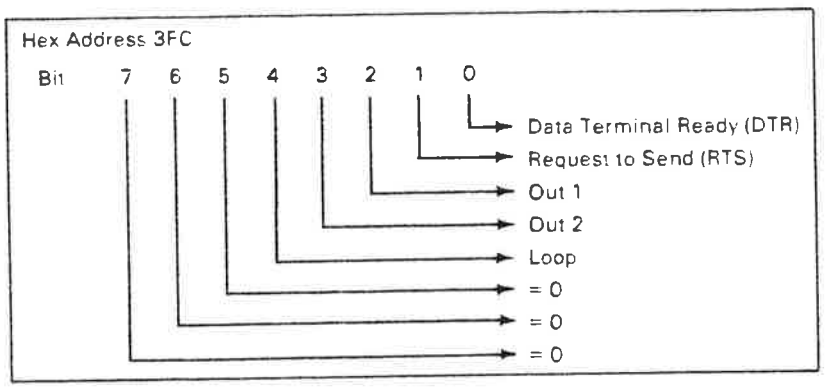
Bit 2: This bit enables the receiver line status interrupt when set to logical 1.

Bit 3: This bit enables the modem status interrupt when set to logical 1.

Bits 4 through 7: These four bits are always logical 0.

Modem Control Register

This eight-bit register controls the interface with the modem or data set (or a peripheral device emulating a modem). The contents of the modem control register are indicated and described below:



Modem Control Register (MCR)

Bit 0: This bit controls the data terminal ready ($\overline{DTR}$) output. When bit 0 is set to a logical 1, the $\overline{DTR}$ output is forced to a logical 0. When bit 0 is reset to a logical 0, the $\overline{DTR}$ output is forced to a logical 1.

Note: The $\overline{DTR}$ output of the INS8250 may be applied to an EIA inverting line driver (such as the DS1488) to obtain the proper polarity input at the succeeding modem or data set.

Bit 1: This bit controls the request to send ($\overline{RTS}$) output. Bit 1 affects the $\overline{RTS}$ output in a manner identical to that described above for bit 0.

Bit 2: This bit controls the output 1 ($\overline{OUT\ 1}$) signal, which is an auxiliary user-designated output. Bit 2 affects the $\overline{OUT\ 1}$ output in a manner identical to that described above for bit 0.

Bit 3: This bit controls the output 2 ($\overline{OUT\ 2}$) signal, which is an auxiliary user-designated output. Bit 3 affects the $\overline{OUT\ 2}$ output in a manner identical to that described above for bit 0.

Bit 4: This bit provides a loopback feature for diagnostic testing of the INS8250. When bit 4 is set to logical 1, the following occurs: the transmitter serial output (SOUT) is set to the marking (logical 1) state; the receiver serial input (SIN) is disconnected; the output of the transmitter shift register is "looped back" into the receiver shift register input; the four modem control inputs ($\overline{CTS}$, $\overline{DSR}$, $\overline{RLSD}$, AND $\overline{RI}$) are disconnected; and the four modem control outputs ($\overline{DTR}$, $\overline{RTS}$, $\overline{OUT\ 1}$, and $\overline{OUT\ 2}$) are internally connected to the four modem control inputs. In the diagnostic mode, data that is transmitted is immediately received. This feature allows the processor to verify the transmit- and receive-data paths of the INS8250.

In the diagnostic mode, the receiver and transmitter interrupts are fully operational. The modem control interrupts are also operational but the interrupts' sources are now the lower four bits of the modem control register instead of the four modem control inputs. The interrupts are still controlled by the interrupt enable register.

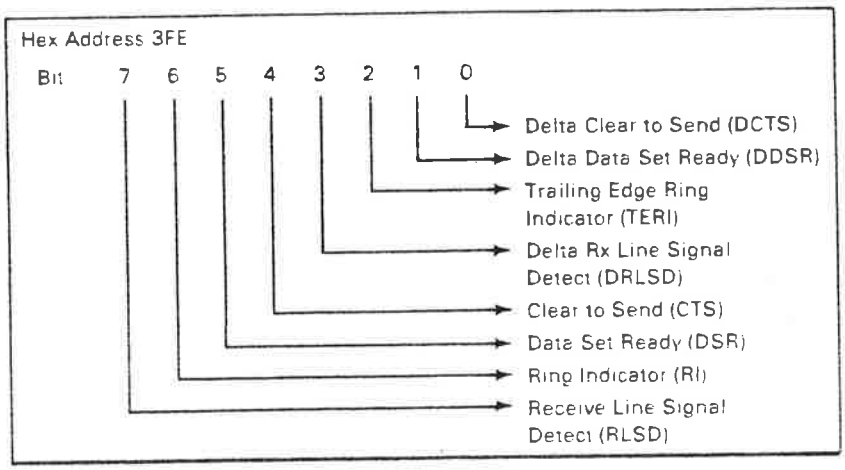
The INS8250 interrupt system can be tested by writing into the lower four bits of the modem status register. Setting any of these bits to a logical 1 generates the appropriate interrupt (if enabled). The resetting of these interrupts is the same as in normal INS8250 operation. To return to normal operation, the registers must be reprogrammed for normal operation and then bit 4 of the modem control register must be reset to logical 0.

Bits 5 through 7: These bits are permanently set to logical 0.

Modem Status Register

This eight-bit register provides the current state of the control lines from the modem (or peripheral device) to the processor. In addition to this current-state information, four bits of the modem status register provide change information. These bits are set to a logical 1 whenever a control input from the modem changes state. They are reset to logical 0 whenever the processor reads the modem status register.

The content of the modem status register are indicated and described below:



Modem Status Register (MSR)

Bit 0: This bit is the delta clear to send (DCTS) indicator. Bit 0 indicates that the $\overline{CTS}$ input to the chip has changed state since the last time it was read by the processor.

Bit 1: This bit is the delta data set ready (DDSR) indicator. Bit 1 indicates that the $\overline{DSR}$ input to the chip has changed state since the last time it was read by the processor.

Bit 2: This bit is the trailing edge of ring indicator (TERI) detector. Bit 2 indicates that the $\overline{RI}$ input to the chip has changed from an On (logical 1) to an Off (logical 0) condition.

Bit 3: This bit is the delta received line signal detector (DRLSD) indicator. Bit 3 indicates that the $\overline{RLSD}$ input to the chip has changed state.

Note: Whenever bit 0, 1, 2, or 3 is set to a logical 1, a modem status interrupt is generated.

Bit 4: This bit is the complement of the clear to send ($\overline{CTS}$) input. If bit 4 (loop) of the MCR is set to a logical 1, this is equivalent to RTS in the MCR.

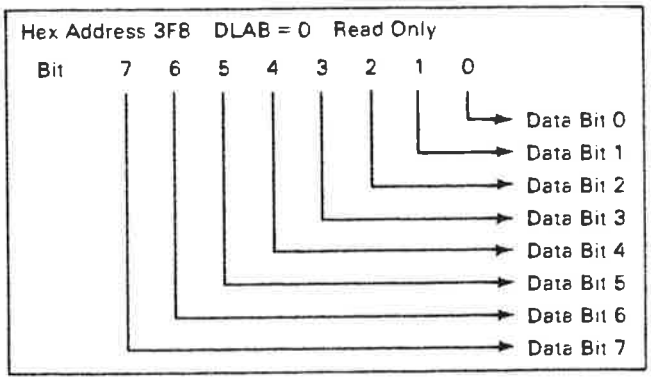
Bit 5: This bit is the complement of the data set ready ($\overline{DSR}$) input. If bit 4 of the MCR is set to a logical 1, this bit is equivalent to DTR in the MCR.

Bit 6: This bit is the complement of the ring indicator ($\overline{RI}$) input. If bit 4 of the MCR is set to a logical 1, this bit is equivalent to OUT 1 in the MCR.

Bit 7: This bit is the complement of the received line signal detect ($\overline{RLSD}$) input. If bit 4 of the MCR is set to a logical 1, this bit is equivalent to OUT 2 of the MCR.

Receiver Buffer Register

The receiver buffer register contains the received character as defined below:

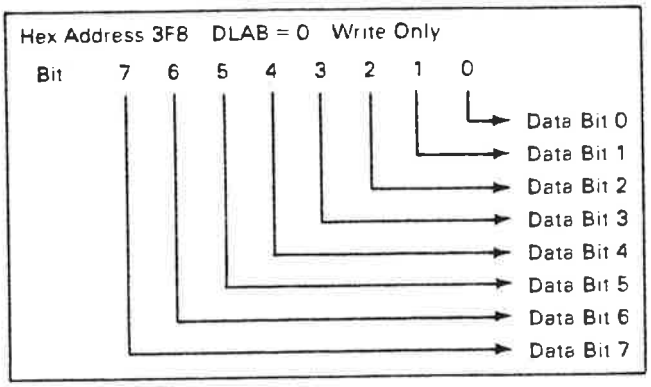


Receiver Buffer Register (RBR)

Bit 0 is the least significant bit and is the first bit serially received.

Transmitter Holding Register

The transmitter holding register contains the character to be serially transmitted and is defined below:



Transmitter Holding Register (THR)

Bit 0 is the least significant bit and is the first bit serially transmitted.

REFERENSER

Concurrent CP/M manual

Concurrent PC/DOS manual

ONSPEC manualer

IBM hårdvarumanual för persondator XT