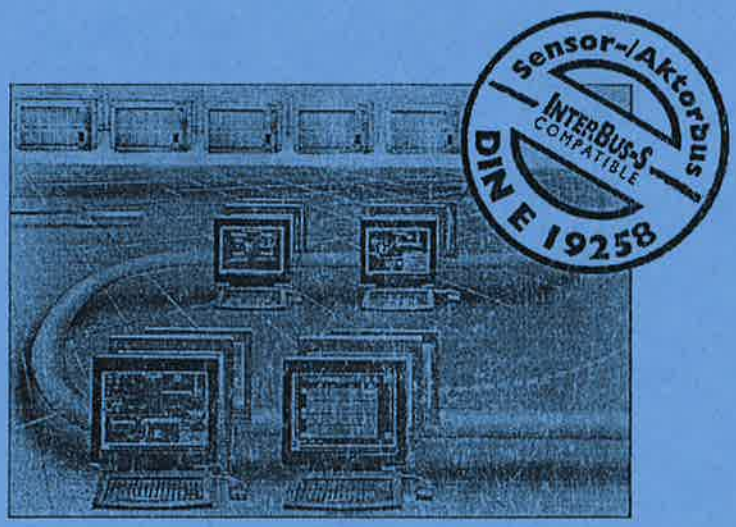
ISSN 0280–5316
ISRN LUTFD2/TFRT--5533--SE

# Kommunikation mot InterBus-S fältbuss under Windows NT




Per Persson

Institutionen för Reglerteknik
Lunds Tekniska Högskola
Augusti 1995

**Department of Automatic Control**
**Lund Institute of Technology**
P.O. Box 118
S-221 00 Lund Sweden

*Document name*
MASTER THESIS

*Date of issue*
August 1995

*Document Number*
ISRN LUTFD2/TFRT--5533--SE

*Author(s)*
Per Persson

*Supervisor*
Knut Mårtensson, Thomas Bergström, Alfa Laval Automation, Karl-Erik Årzén, LTH

*Sponsoring organisation*

*Title and subtitle*
Kommunikation mot InterBus-S fältbuss under Windows NT. (Communication between InterBus-S fieldbus and Windows NT.)

*Abstract*

The market for fieldbuses is new and very expansive. Different types of fieldbuses constantly evolve and only a few of them reach the status of international or de facto standard. One of the runners for becoming an international standard is InterBus-S, a fieldbus with optimized performance for sensors and actuators.

In this thesis the possibilities for communication between InterBus-S and an application running under Windows NT are analyzed. The thesis describes the theory of InterBus-S including, e.g. topology and protocol. The development of a device driver for Windows NT and the integration of InterBus-S in an application running under Windows NT are also described. Finally the control exercise "Ball and beam" shows an example of an application using InterBus-S for communication.

*Key words*

*Classification system and/or index terms (if any)*

*Supplementary bibliographical information*

*ISSN and key title*
0280–5316

*ISBN*

*Language*
Swedish

*Number of pages*
36

*Recipient's notes*

*Security classification*

# Innehållsförteckning

# 1 Inledning

Detta examensarbete behandlar hur man skulle kunna använda PC-datorer som kör Windows NT i styr- och reglersammanhang, det vill säga ansluta givare och ställdon direkt till en PC. Det vanligaste sättet att ansluta givare och ställdon har hittills varit att ansluta dem till speciellt dedicerade styrsystem tillverkade av automationsföretag som till exempel Alfa Laval Automation AB. Dessa styrsystem består av specialdesignad hårdvara som varje automationsföretag själv måste utveckla.

Det är här examensarbetet kommer in som ett exempel på hur man i stället skulle kunna använda standardhårdvaran i en PC för styrning och reglering inom industrin. Det finns egentligen två olika alternativ att utnyttja PC:n på: antingen som ett PC-baserat styrsystem placerat ute i fabriken, eller som en kombinerad operatörstation och styrsystem stående i kontrollrummet.

Uppgiften i detta examensarbetet bestod i att koppla en applikation i Windows NT-miljö till I/O-moduler på InterBus-S. För att göra denna koppling användes ett InterBus-S-instickskort för PC och applikationen att koppla sig mot var SattLine. SattLine är ett objektorienterat, distribuerat automationssystem utvecklat av Alfa Laval Automation AB.

Arbetet har varit uppdelat i fyra olika delar: Undersökning av InterBus-S och dess möjligheter, utveckling av drivrutin för Windows NT, framtagande av biblioteksmoduler för InterBus-S i SattLine samt test och utvärdering av examensarbetets tidigare delar med hjälp av laborationsprocessen "Bommen".

Kapitel 3 innehåller en allmän introduktion av fältbussar samt djupare information om InterBus-S. Där behandlas även historia, egenskaper och prestanda för InterBus-S.

I kapitel 4 beskrivs arbetet med att få hårdvaran för InterBus-S att kommunicera med en applikation Windows NT. Detta innefattar hur drivrutinen utvecklades och de implementerings- och designval som gjordes under utvecklingen.

Hur InterBus-S kan användas i SattLine och hur denna integrering gjordes med hjälp av biblioteksmoduler finns beskriven i kapitel 5.

Som ett exempel på tillämpning av den utveckling som beskrivits i de två föregående kapitlen ges i kapitel 6 en beskrivning av laborationsprocessen "Bommen". I detta kapitel presenteras även resultaten från detta exempel.

I kapitel 7 ges resultat och utvärdering av hela projektet att kommunicera mot InterBus-S i Windows NT. Detta kapitel innehåller resultaten från examensarbetet samt idéer och funderingar om framtida utvecklingsmöjligheter för det som har framkommit under examensarbetet.

Slutligen finns i appendix en förklaring av den interna signaladresseringen i SattLine, samt en handledning till hur man sätter upp en utvecklingsmiljö för drivrutiner till Windows NT.

# 2 Definitioner och akronymer

| | |
|---|---|
| *ANSI* | American National Standards Institute. |
| *ASI* | Actuator Sensor Interface. Fältbuss speciellt lämpad för givare och ställdon. |
| *CAN* | Controller Area Network. Fältbuss konstruerad av Bosch för bilindustrin. |
| *CCITT* | Consultative Committee on International Telegraphy and Telephony. |
| *ControlNet* | Fältbuss utvecklad av Allen-Bradley. |
| *CRC* | Cyclic Redundancy Check. En kontrollsumma som används vid bitorienterade protokoll. Beräknas som resten efter division mellan dataordet och ett generatorpolynom. |
| *DIN* | Deutsche Industri Norm. |
| *DEK* | Deutsche Elektroteschnische Kommision. |
| *DoD* | Deparment of Defence, USA. |
| *FF* | Fieldbus Foundation. Bildad september 1994. Arbetar för en internationell fältbusstandard. Har cirka 120 anslutna automationsföretag. |
| *FIP* | Factory Instrumentation Protocol. Fältbuss som är en utmanare till PROFIBUS. |
| *FMS* | Fieldbus Message Specification. |
| *Gateway* | Utrustning med enda uppgiften att koppla samman två system, speciellt om systemen använder sig av olika protokoll. |
| *IEC* | International Electrotechnical Commission. Internationellt standardiseringsorgan med flera tekniska kommittéer (TC) för olika branscher. |
| *IEEE* | Institute of Electrical and Electronics Engineers. |
| *InterBus-S* | Fältbuss ursprungligen utvecklad av Phoenix Contact GmbH & Co. Speciellt anpassad för givare och ställdon. |
| *I/O* | Generell beteckning för in- och utsignaler och den utrustning som används för att förmedla dem. |
| *I/O-RAM* | Utrymme i datorns eget minne för mellanlagring av data till och från MPM. |
| *IP 65* | Skyddsgrad av elektrisk utrustning enligt franska normen NFC 20-010 och tyska DIN 40050. IP 65 innebär fullständigt skydd mot damm och sköljning av vatten. |
| *ISA* | Industry Standard Architecture. Buss lanserad av IBM till deras AT-datorer. Det finns idag enorma mängder av passande instickskort till standarden. |
| *ISO* | International Standards Organisation. |
| *ISP* | Interoperable Systems Project. Har tillsammans med World FIP bildat FF för att skapa en världsstandard för fältbussar. |
| *MMS* | Manufacturing Message Specification. |
| *MPM* | Multiport minne. Ett fysiskt RAM-minne med flera portar vilket ger möjlighet för flera användare att adressera minnet samtidigt. |
| *OSI* | Open Systems Interconnect. En modell för datakommunikation skapad av ISO som delar upp kommunikationen i sju lager. |
| *PC* | Personal Computer. I denna rapport menas med PC de Intel-baserade persondatorer som ofta kallas IBM-kompatibla. |
| *PCP* | Peripherals Communication Protocol. |
| *PMS* | Peripherals Message Specification. |
| *POSIX* | Portable Operating System Interface. Standard för UNIX-system definierad av ANSI och IEEE. |
| *PROFIBUS* | Process Field Bus. Fältbuss standardiserad enligt DIN 19245. Finns i olika varianter (DP, FMS osv.) vilka är optimerade för olika tillämpningar. |
| *TTL* | Transistor Transistor Logic. Digitala kretsar som arbetar med 0 och 5V. |
| *World FIP* | Organisation för FIP-bussen. Den nordamerikanska delen av World FIP har med ISP bildat FF. |

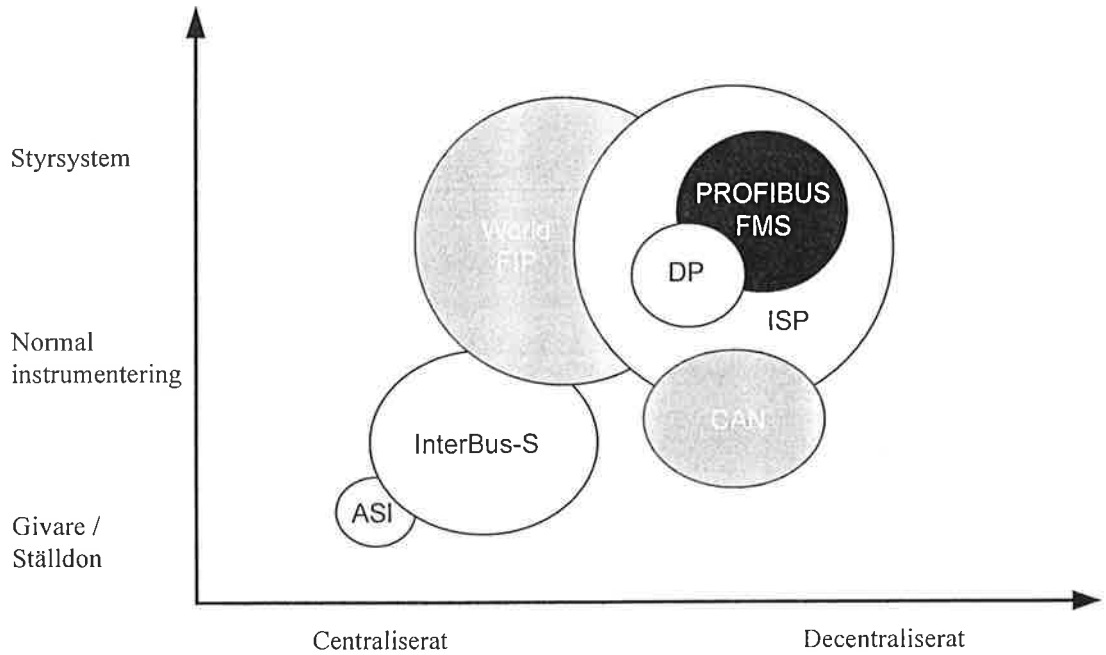
# 3 Beskrivning av InterBus-S

## 3.1 Allmänt om fältbussar

Inom industriell automation har det länge varit vanligt att alla signaler man vill styra och mäta dras i enskilda ledningar till sitt styrsystem. Detta leder till enorma kabelstammar om processen som skall automatiseras är någorlunda stor. Även vid felsökning och underhåll kommer dessa mängder av kablar att med stor sannolikhet vålla problem. För att undvika detta har man tagit fram konceptet med fältbussar. En fältbuss innebär att man knyter samman de ställen i processen där signaler mäts och styrs med en enda bussledning som löper genom hela fabriken. Med detta förfarande minskar man dramatiskt mängden kabel som måste dras. Nu är det tyvärr inte så enkelt, utan det man vinner i minskad kabeldragning har fått betalas i en ökad komplexitet hos kommunikationen i den nu enda kabeln.

Från att ha transporterat en enda signal per kabel måste kabeln nu rymma hela processens alla signaler. Detta kan lösas på en mängd olika sätt som till exempel tidsmultiplex eller frekvensmultiplex. Som en följd av fältbussens uppbyggnad måste även ett annat problem med den lösas, nämligen att kommunikationen kan bli väldigt sårbar om den enda kabeln blir störd eller skadad. För att lösa de problem en ensam bussledning medför, måste kommunikationen säkras med hjälp av protokoll med felkorrektion, feldetektering eller omsändningar.

Med införandet av fältbussar får man förutom enklare kabeldragning även ökad funktionalitet. Fältbussen gör det möjligt att ha decentraliserad logik i systemen. Detta kallas ofta intelligenta moduler, vilket innebär att ute vid givare och sensorer i fabriken sitter även en del av styrningen som till exempel skalning av analoga värden och motorstyrning.



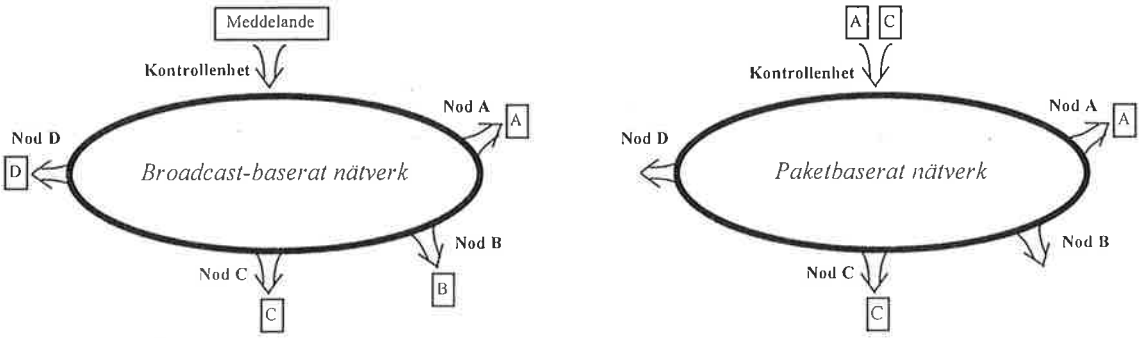

**Figur 1: Illustration av några olika fältbussars användningsområden.**

Det finns idag ett flertal olika implementationer av fältbussar, så som till exempel PROFIBUS-DP, CAN, InterBus-S, ControlNet och ASI. Alla dessa fältbussar har specialiserat sig mot olika typer av användningar (se några exempel i Figur 1 ovan), eftersom det är praktiskt omöjligt att tillverka en generell fältbuss som skulle passa i alla lägen. Detta examensarbete är inriktat på

InterBus-S, vilket är en buss speciellt lämpad för givare och ställdon på grund av dess egenskaper med snabb cykeltid, effektivt protokoll och små datamängder.

## 3.2 Bakgrund, historia och framtid

InterBus-S utvecklades från början av Phoenix Contact GmbH & Co. i Blomberg, Tyskland och lanserades som en produkt 1987. InterBus-S har fått ett starkt fäste inom tysk industri, speciellt inom bilindustrin. För att inte låsa sig i en nisch har Phoenix och andra intressenter på senare tid bildat en InterBus-S-förening med uppgift att standardisera och styra utvecklingen av InterBus-S. Ett resultat av detta samarbete är InterBus Sensor Loop, vilket är en fältbuss specialiserad på billig installation av enkla analoga och digitala givare och ställdon. InterBus Sensor Loop är gjord för enkel samverkan med existerande InterBus-S-system, där priset per anslutning till bussen måste vara extremt lågt.




**Figur 2: Förmedlingssätt hos broadcast- och paketbaserade nätverk.**

Det finns huvudsakligen två olika förmedlingssätt hos fältbussar för att transportera information mellan kontrollenheten och anslutna noder. Figur 2 ovan illustrerar dessa två förmedlingssätt. Vid ett broadcast-baserat nätverk kommer vid varje sändning av data alla noder i nätverket att få ta emot data. Det krävs således lika mycket sändningskapacitet att sända till en nod som att sända till alla noder i nätet. Fördelen är att man bara behöver extremt lite adresseringsinformation i den nyttiga datan eftersom alla noder alltid skall ta emot information.

Vid ett paketbaserat nätverk sänds bara information mellan kontrollenheten och de noder som har någon information att förmedla. Detta gör att ingen onödig data skickas till och från noder som inte har något att förmedla, men nackdelen är att man måste inkludera en hel del adresseringsinformation med datan så att den når rätt mottagare i nätverket.

Fältbussen InterBus-S är ett broadcast-baserat nätverk, där alla moduler på bussen tar emot och sänder data samtidigt. En beskrivning av hur protokollet som sköter detta fungerar finns i avsnitt 3.4 nedan.

InterBus-S har blivit godtagen av DEK som en tysk standard (DIN 19258) i maj 1994, men siktar mot en internationell IEC-standard inom något år. Kampen om att få just sin fältbuss antagen som en internationell standard är mycket hård, vilket leder till att det kan dröja innan man vet vilken eller vilka bussar som skall få status som internationella standarder. Det internationella standardiseringsarbetet leds av kommittén CLC/TC65CX och InterBus-S-föreningen siktar på att InterBus-S skall bli antagen i standarden CLC/TC65CX(SEC)23: "High efficiency communication subsystem for small data packages". Produkter som är testade för kompatibilitet mot InterBus-S får märkas med en logotyp enligt Figur 3 nedan.


**Figur 3: Logotyper för InterBus-S DIN-standard (t.v.) och för märkning av godkända moduler (t.h.).**

DIN-standarden stöds idag av över 500 produktleverantörer. För att underlätta utvecklingen har InterBus-S-föreningen sedan 1990 uppmuntrat bildandet av arbetsgrupper med inriktningar mot olika tillämpningar för InterBus-S. Idag finns det åtta arbetsgrupper med olika inriktningar som till exempel DRIVECOM (frekvensstyrningar och servomotorer), ENCOM (avkodare), MMICOM (operatörs- och presentationsutrustningar) och WELDCOM (svetsstyrningar). Med InterBus-S har det hittills installerats mer än 30.000 applikationer världen över med mer än 500.000 sammankopplade fältenheter.

Hur framtiden ser ut för InterBus-S kan vara svårt att sia, men det har gjorts stora investeringar i befintliga anläggningar och intresset för fältbussar ökar allt eftersom tekniken blir mer mogen och spridd. När det visar sig att man kan uppnå tillräckligt hög tillförlitlighet hos systemen, så är det troligt att även de mer försiktiga branscherna, som till exempel den svenska pappersindustrin, kommer att vilja börja använda fältbussar.

En annan utveckling inom fältbussar som verkar kunna bli till en fördel för InterBus-S är att det utvecklas gateways mellan olika bussar. Detta innebär att det kanske räcker att bara ha speciell anpassning för vissa fältbussar i sin tillämpning och sedan kan man ansluta andra fältbussar via gateways. För InterBus-S finns det redan nu gateways för Sensor Loop och ASI. För mer teknisk information om InterBus-S hänvisas till följande avsnitt i kapitel 3 och referens [1].

## 3.3 Topologi

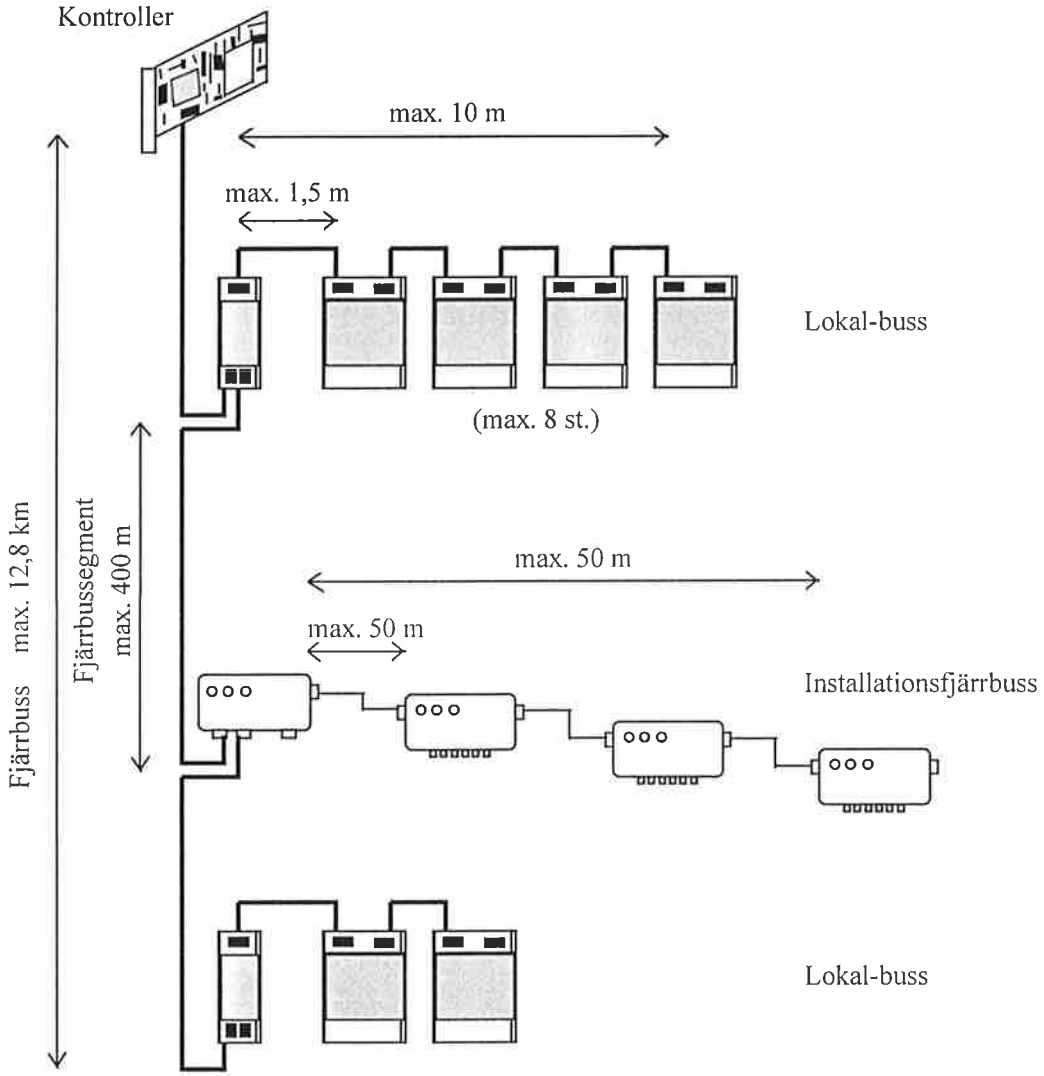
Ett InterBus-S-nät byggs som ett träd med avstickande grenar. Vid roten sitter en kontrollenhet och styr trafiken i hela nätet. Delarna i trädet byggs upp av tre typer av segment: fjärrbussar, installationsfjärrbussar och lokalbussar, alla tre med olika användningsområden och egenskaper.

Figur 4 nedan illustrerar hur de olika typerna av bussegment används för att bygga upp ett InterBus-S-nät samt vissa av de begränsningar som finns angående avstånd och antal.

Fjärrbussar används för att överbrygga långa avstånd i fabriken och bär endast med sig den dubbelriktade kommunikationen. Den totala längden av fjärrbussen för hela nätet kan vara maximalt 12,8 km (från kontrollenheten till sista anslutna fjärrbussmodul). Den del av fjärrbussen som går mellan två anslutna fjärrbussmoduler delar in hela fjärrbussen i ett antal segment. Varje sådant fjärrbussegment kan vara maximalt 400 m långt. Man kan maximalt ansluta 256 moduler till fjärrbussen, men eventuellt färre (32 eller 64 moduler) om man inte har en fullt implementerad InterBus-S-kontrollenhet.

Kommunikationen i fjärrbussar sker normalt via ett RS-485 interface på kopparkabel, men kan även göras via optisk fiber, vågledare eller IR-kommunikation om bussen skall användas i extremt störande eller känslig miljö. Modulerna på fjärrbussen har som funktion att regenerera

inkommande datatrafik på utgående ledning, samt att elektriskt skilja de olika segmenten från varandra, för att minimera spridningen av störningar och skador som kan uppstå av till exempel överspänningar. Fjärrbussmodulerna kan även koppla bort bussegment (fjärrsegment eller lokalsegment), antingen på uppdrag av kontrollenheten eller på grund av ett fel i någon efterföljande modul som omöjliggör vidare kommunikation.




**Figur 4: Topologi för InterBus-S-nät med de olika typerna av bussegment.**

Installationsfjärrbussar utgör förgreningar från fjärrbussen och är avsedda för medellånga avstånd. De bär förutom kommunikationen även med sig spänningsmatning till de anslutna modulerna. Det är bara möjligt att ansluta installationsfjärrbussar till den ursprungliga fjärrbussen, vilket gör att vidare förgreningar inte är möjliga. Denna typ av segment är speciellt anpassad för moduler av skyddsklass IP 65 eller liknande för att slippa ha separat spänningsförsörjning till varje modul i krävande miljöer. Den maximala längden på hela installationsfjärrbussen är 50 m, vilket också är det maximala avståndet mellan två efterföljande moduler på denna typ av segment. Det finns ytterligare en fysisk begränsning genom att strömmatningen längs bussen är maximerad till 4,5 A.

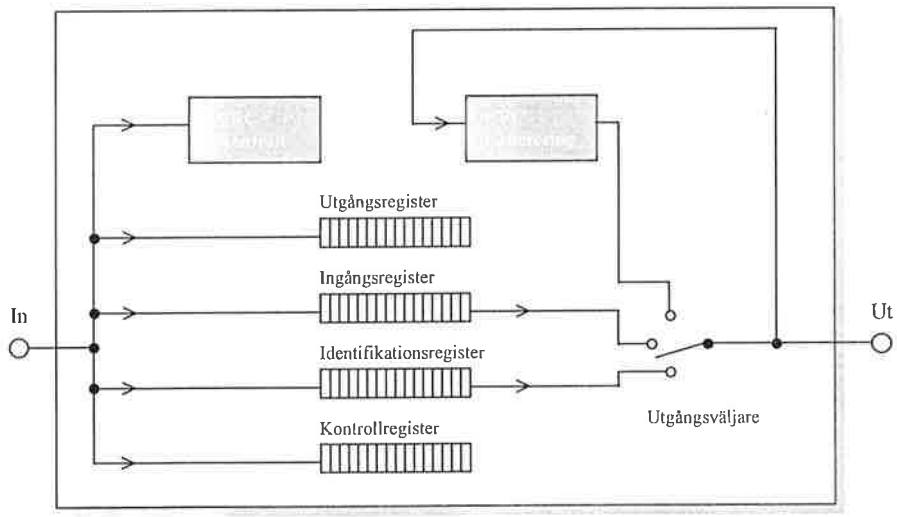
Lokalbussarna slutligen är endast avsedda för korta avstånd inom till exempel rackar, skåp och mindre maskiner och bär precis som installationsfjärrbussen både kommunikation och matningsspänning, men med en enklare, mindre störningsimmun kommunikation (TTL nivåer).

Detta ger möjligheten att göra billigare och enklare moduler för en kostnadseffektiv installation i jämförelse med modulerna på fjärrbussarna. Lokalbussgrenen kan i likhet med installations-fjärrbussen inte grenas i nya lokalbussar och kan maximalt innehålla 8 moduler. Vidare är den begränsad av att hela lokalbussegmentets längd inte får överskrida 10 m och med maximalt 1,5 m mellan varje lokalbussmodul.

## 3.4 Protokoll

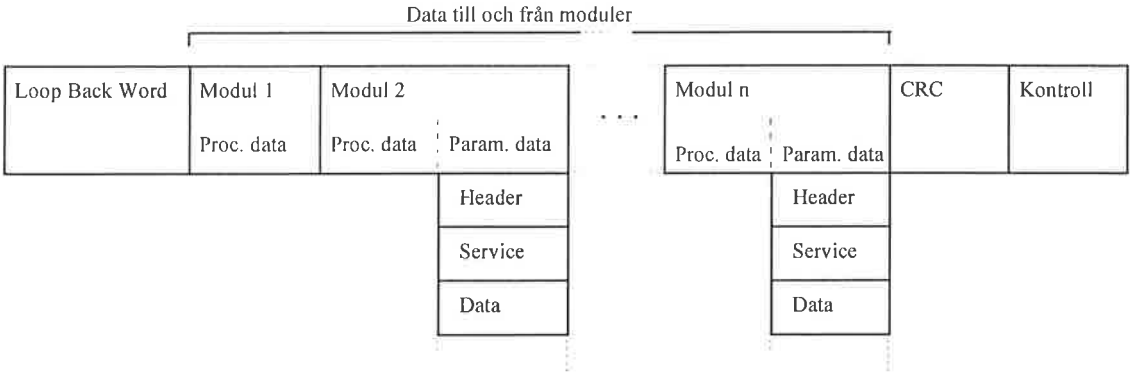
### 3.4.1 Allmänt

Även om noderna i InterBus-S-nätet ser ut att vara kopplade i ett träd, så är de logiskt kopplade i en ring. Att alla noder är kopplade i en ring är en av grundidéerna med InterBus-S. Hela bussen fungerar som ett distribuerat skiftregister med ett master/slave-förhållande. Som master används en kontrollenhet (ofta kallad InterBus-S Master), vilken är placerad i ett styrsystem eller en PC. Kontrollenheten sköter klockningen av data runt i skiftregistret och skiftar på så sätt ut data från det överliggande styrsystemet eller PC:n och skiftar samtidigt in data från processen. Noderna på InterBus-S-nätet fungerar som slavar, vilka bara kommunicerar på initiativ av kontrollenheten.




**Figur 5: Schematisk intern uppbyggnad av slavmodul**

Den interna uppbyggnaden av varje slavmodul på InterBus-S (beskriven i Figur 5 ovan) består schematiskt av fyra register, en CRC-generator, en modul för CRC-kontroll och en utgångsväljare. De fyra registerna har följande funktioner: Ett ingångsregister, ett utgångsregister, ett identifikationsregister och ett kontrollregister. Om en modul bara är ingångs- eller utgångsmodul finns trots det alltid de fyra registerna, men då är ett av in- eller utgångsregisterna ett dummy-register. In- och utgångsregisterna är av olika längd för olika moduler (men samma för båda registerna), beroende på hur mycket data modulen i fråga behöver. De andra två registerna är alltid 16 bitar vardera. Hur registerna används finns beskrivet i sektion 3.4.2 och 3.4.3 nedan.




**Figur 6: Uppbyggnad av meddelande enligt summa-ram protokoll**

Protokollet som används på bussen är ett summa-ram protokoll med meddelande enligtFigur 6 ovan. Ett summa-ram protokoll innebär att det skickas ett meddelande på bussen per cykel. Detta meddelande innehåller all data ut till modulerna samt fel- och kontrollinformation. Samtidigt skiftas det föregående utsända meddelandet in, vilket nu innehåller indata från samtliga moduler samt fel- och kontrollinformation på samma sätt som vid sändning av ett meddelande. Längden på paketen är konstant under drift och bara beroende av antalet anslutna moduler till bussen och typerna hos respektive moduler.

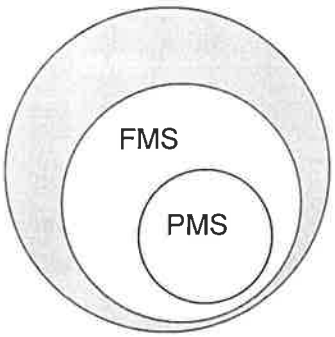
| Signal | Funktion |
|---|---|
| Klockfrekvens (CK) | Anger bittakten (normalt 500 kbps). |
| Kontrollsignal (CR) | Anger om meddelandet är i data- eller CRC-delen. |
| Statussignal (SL) | Anger om ID- eller datacykel pågår |
| Resetsignal (RC) | Används för att återställa alla anslutna moduler |

**Figur 7: Signaler som kan tas fram ur kommunikationen på bussen.**

Informationen överförs som seriell två-trådskommunikation mellan modulerna. Den seriella signalen innehåller förutom själva datan även de fyra signaler som presenteras i Figur 7 ovan. Varje modul kan genom att nollställa CR-signalen under CRC-delen meddela övriga moduler att ett fel har inträffat vid kontrollen av CRC, vilket innebär att hela datacykeln förkastas.

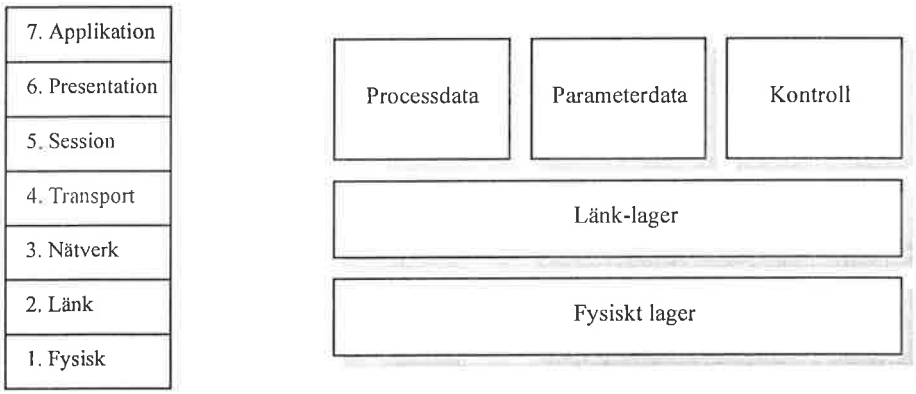
Det transporteras två typer av data oberoende av varandra på bussen: Process- och parameterdata. Processdata är den tidskritiska datan som skall transporteras från givare till kontrollenheten och till ställdon från kontrollenheten. Denna data transporteras som en direkt avbild av vad som finns i modulernas in- och utgångsregister. Processdata kommer därför att uppdateras en gång per datacykel, vilket sker med en deterministisk frekvens. Parameterdata däremot transporteras enligt ett speciellt protokoll (PCP) i de luckor som de moduler vilka behöver parameterdata har reserverat (se Figur 6 ovan) för parameterdata. Dessa meddelanden enligt PCP överförs under flera datacykler, med endast en del per cykel, och protokollet sköter uppdelningen och sammansättningen av de större meddelandena hos kontrollenhet och respektive moduler. Typisk information som transporteras som parameterdata är parametrar till regulatorer, motorskydd och så vidare.

Gränssnittet mot användaren är för processdata endast den obehandlade rådatan, men för parameterdata använder InterBus-S protokollet PMS vilket är en delmängd av FMS och MMS (se Figur 8 nedan). För parameterdata ger PMS exakt samma funktionalitet som FMS. Se referens [1] för mer ingående information.




**Figur 8: PMS som delmängd av FMS och MMS**

Vid designen av InterBus-S har man följt OSI-modellen (ISO 7498) och delat upp kommunikationen enligt de sju nivåer som OSI-modellen föreskriver. Dock har man av prestandaskäl valt att inte implementera nivå 3 till 6 explicit då dessa nivåer inte fyller någon viktig funktion vid kommunikationen på InterBus-S. Figur 9 nedan illustrerar placeringen av InterBus-S i OSI-modellen och hur process- och parameterdata transporteras som kanaler oberoende av varandra. Det finns även en tredje kanal för nätverkskontroll där styr- och kontrollinformation till InterBus-S transporteras.




**Figur 9: OSI-modellen och dess implementering i InterBus-S**

För att garantera korrekt operation av bussen och korrekt överföring av data använder InterBus-S två olika mekanismer: Loop Back Word och CRC-kod. Loop Back Word sänds ut först på bussen och används för att kontrollera att ett meddelande har kommit ut till modulerna på bussen och att hela föregående paket har skiftats in igen. Loop Back Word består av ett slumpmässigt taget 16-bitars ord, vilket är entydigt skilt från ID-koderna vilka beskrivs nedan. När Loop Back Word kommer in igen efter ett varv i skiftregistret, så kontrolleras det mot det som tidigare sändes ut. Om dessa två inte är lika kommer all information som gick med cykeln in och ut att förkastas utan någon mer kontroll.

Den CRC-kod som används är 16-bitar lång och har som funktion att kontrollera korrektheten hos sänd och mottagen data. Denna CRC-kod kontrolleras och genereras på nytt hos varje modul på nätet. Om ett fel skulle upptäckas, så förkastas data i aktuellt meddelande och modulen väntar på nästa meddelande. För att skapa koden används följande generatorpolynom av grad 16, vilket är fastställt av CCITT:

$$G(x) = x^{16} + x^{12} + x^5 + 1 \qquad (1)$$

För mer allmän information om CRC hänvisas till referens [5]. Den här använda CRC-koden

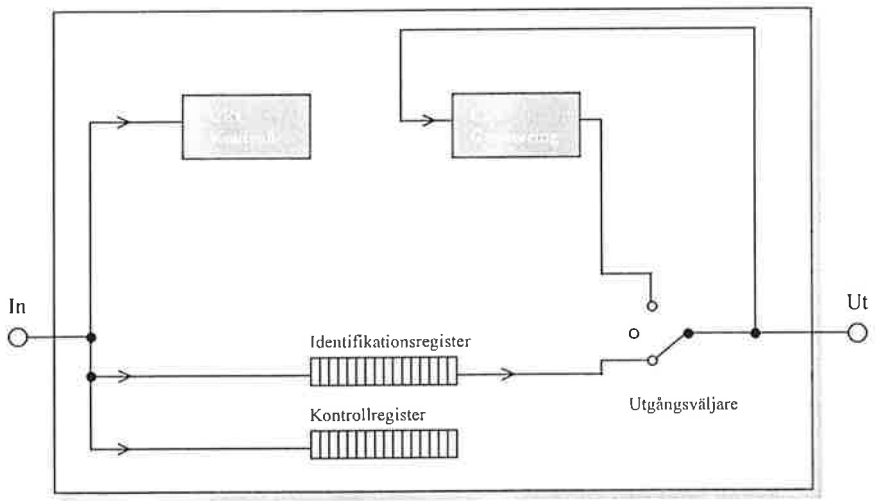
ger ett Hamming-avstånd på 4 och möjlighet att detektera alla enkel- och dubbelfel samt alla udda antal bitfel. Dessutom detekteras alla skurfel med en längd kortare än 16 bitar och längre skurfel detekteras med en sannolikhet av 99,9%.

### 3.4.2 Konfigurering

ID-koden är ett 16-bitars ord som identifierar varje typ av modul. Ordet innehåller två 8-bitars delar som innehåller en längdkod respektive en gruppkod. Längdkoden innehåller information om hur mycket plats i datameddelandet modulen tar och gruppkoden anger vilken typ av modul det är frågan om. Olika modultyper kan till exempel vara: analoga fjärrbussmoduler, digitala lokal-bussmoduler och så vidare. Giltiga värden på dessa koder bestäms av InterBus-S-föreningen och därför har moduler med samma datamängd och samma typ, men olika tillverkare, samma ID-kod.

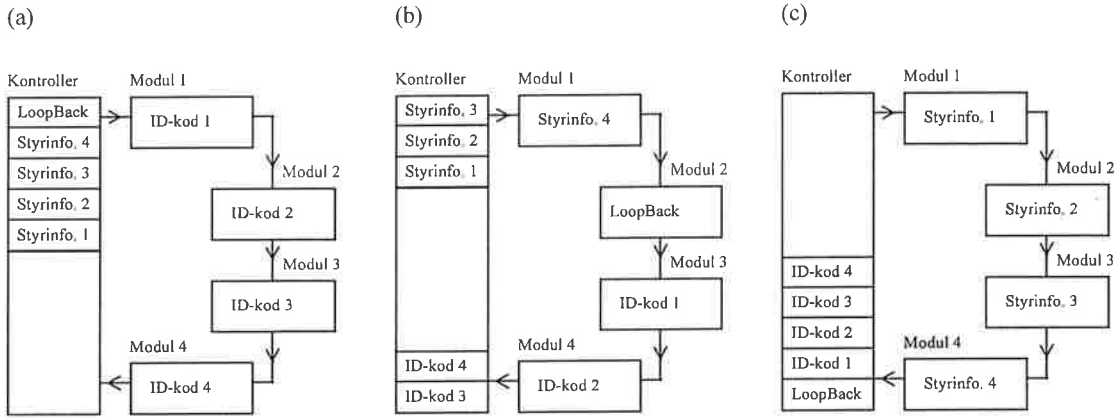
Konfigureringen av ett InterBus-S-nät sker genom att en ID-cykel samlar upp information om de anslutna modulerna till bussen. Denna information kan sedan användas i styrsystemet och kontrollenheten för att detektera eventuella förändringar i konfigurationen. Precis som beskrivits ovan kontrolleras all datatrafik under identifikationscykeln av Loop Back Word och CRC.

Samtidigt med insamling av identifikationskoder sker utmatning av styrinformation till modulerna, enligt principen med skiftregister. Figur 10 nedan visar hur de interna registerna i modulen används för att skicka in identifikationskod till kontrollenheten samtidigt som styr- och kontrollinformation tas emot från kontrollenheten. Med hjälp av styrinformationen kan kontrollenheten ge reset-signal eller beordra avstängning av vissa bussegment till enskilda moduler oberoende av om andra moduler är anslutna till bussen.




**Figur 10: Intern funktion i moduler under identifikationscykel.**

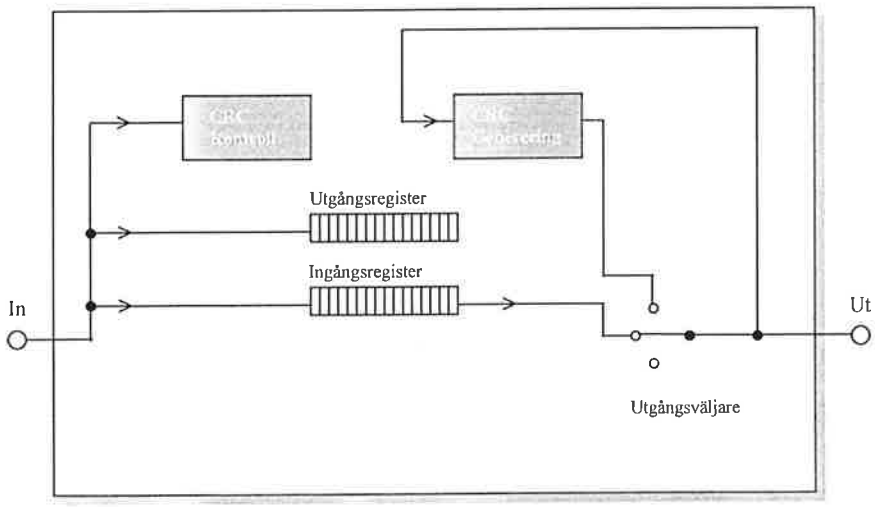
Själva ID-cykeln går till så att kontrollenheten har i sitt interna minne lagt upp en bild av det yttre skiftregistret. Där ligger nu all styrinformation till nätet ordnat i den följd modulerna sitter på bussen, se Figur 11 (a). När ID-cykeln börjar skiftas Loop Back Word och styrinformationen ut till modulerna, se Figur 11 (b). När hela ID-cykeln har genomförts har styrinformationen kommit ut till modulerna och kontroll av CRC anger om överföringen har skett korrekt. I kontrollenheten finns nu identifikationskoder för de anslutna modulerna och ordningen mellan koderna tillsammans med gruppkoderna anger hur trädstrukturen ser ut, se Figur 11 (c).




**Figur 11: Funktionaliteten hos kommunikationen vid ID-cykel.**

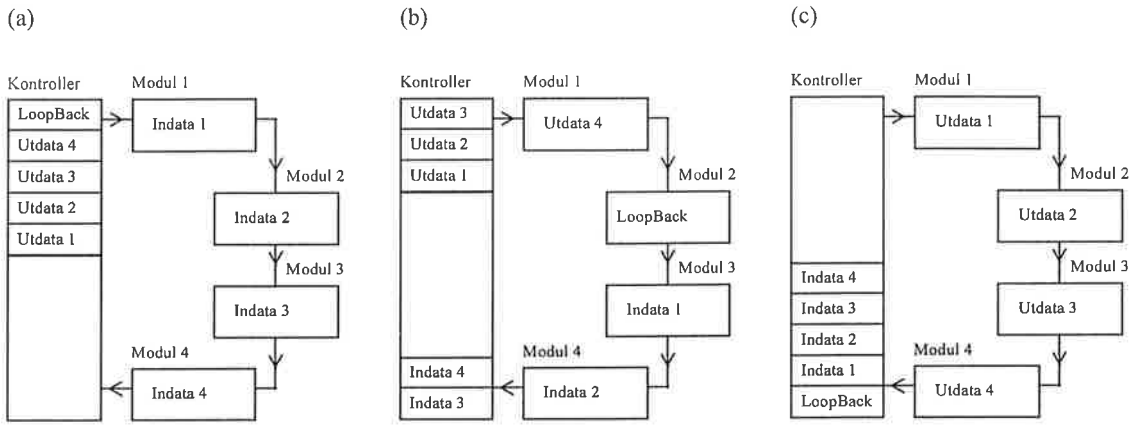
### 3.4.3 Datatrafik

Under datacykeln används bara in- och utgångsregisterna enligt Figur 12 nedan. Före starten av datacykeln innehåller ingångsregistret den data som modulen har läst av från sina ingångar. Om modulen saknar ingångar kommer registret vara ett dummy-register av samma längd som utgångsregistret. All den seriella datan som kommer från föregående modul skiftas in i både in- och utgångsregistret. Genom ingångsregistret passerar datan igenom modulen till nästa modul och när datacykeln är färdig kommer utgångsregistret att innehålla utdata till just den här modulen. Innan utgångsdatan kopieras till modulens utgångar, så kontrolleras CRC och CR-signalen så att inga överföringsfel har skett. Om inget fel har detekterats av någon modul på bussen, är allt som det skall och då kommer utgångsregistret att kopieras till sina respektive utgångar.




**Figur 12: Intern funktion i moduler under datacykel.**

Figur 13 nedan beskriver hur modulerna är kopplade till varandra som ett distribuerat skiftregister och hur data transporteras till och från processen under en datacykel. I delfigur (a) ligger Loop Back Word och utdatan redo att skickas ut till modulerna. Datacykeln har hunnit en bit på väg i delfigur (b) och i delfigur (c) har Loop Back Word kommit tillbaka till kontrollenheten. Denna innebär att datacykeln är komplett och modulerna kontrollerar sina kontrollsummor.




**Figur 13: Funktionaliteten hos kommunikationen vid datacykel.**

## 3.5 Prestanda

Protokollet på InterBus-S ger en deterministisk svarstid för all processdata. Att det blir så enkelt beror på att all in- och utdata förmedlas under varje datacykel av protokollet enligt ovan. Detta protokoll ger således både för- och nackdelar med tanke på determinism respektive onödig transport av data.

Fördelarna är som nämnts en deterministisk svarstid samt att ingen adressering är nödvändig under normal drift. Nackdelarna är att även data som inte behöver uppdateras så ofta som cykeltiden, kommer att uppdateras i varje varv av datacykeln. Det enkla protokollet gör det enkelt att beräkna cykeltiden för uppdatering av in- och utdata.

Cykeltiden kan beräknas med följande linjära ekvation:

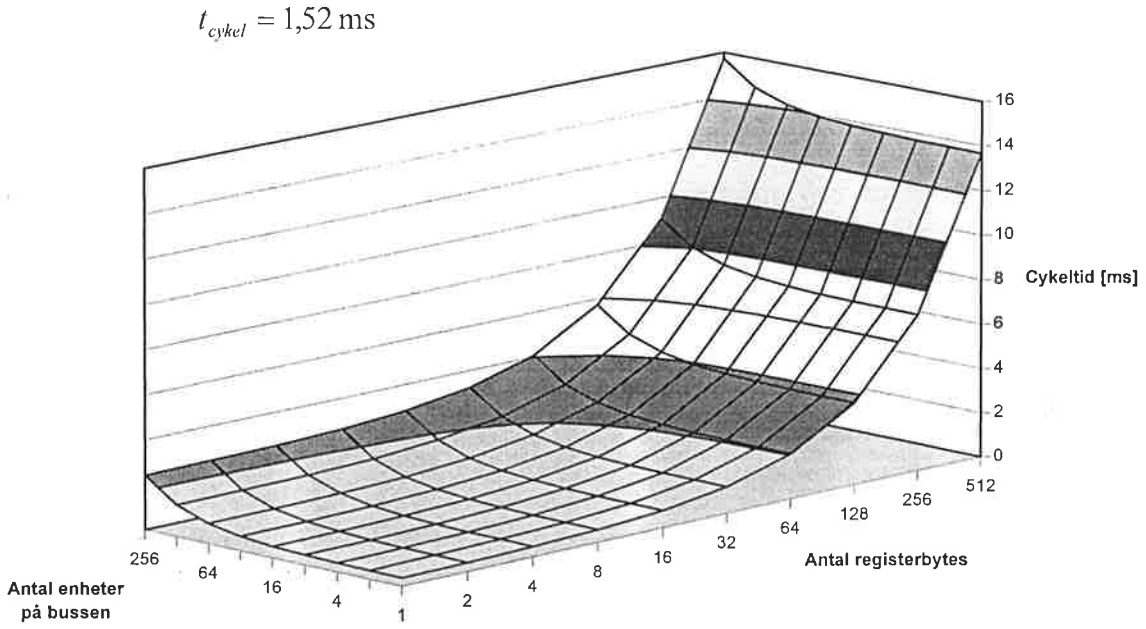
$$t_{cykel} = \left(13 \cdot \left(6+n\right) + 4 \cdot m\right) \cdot t_{bit} + t_{SW} + 2 \cdot l \cdot t_{ut} \qquad (2)$$

där
- $t_{cykel}$ Tid för en komplett datacykel.
- $n$ Totalt antal registerbytes för alla moduler på bussen.
- $m$ Antal enheter på bussen.
- $t_{bit}$ Tid för sändning av en bit ( = 0,002 ms vid 500 kbps).
- $t_{SW}$ Tid för körning av intern programvara ( = 0,2 ms).
- $l$ Kabelns längd i km för fjärrbuss.
- $t_{ut}$ Utbredningstid per kilometer i aktuellt medium.

Denna formel ger med vilket tidsintervall all I/O-data kommer att uppdateras. Figur 14 nedan visar cykeltiden för olika InterBus-S-nät och för att få ett begrepp om vilka tider det rör sig om för en typisk installation, följer här ett litet exempel:

**Exempel:** Antag att fjärrbussen har en längd av 500 m och att vi har 7 moduler på bussen. I vårt fall antar vi att dessa 7 moduler skulle detta ge en total registerlängd av 42 bytes. Antag dessutom att vi använder kopparkablar ($t_{ut}$ = 0,016 ms / km) och RS-485. Vi får då tiden för en datacykel:

$$t_{cykel} = \left(13 \cdot \left(6+42\right) + 4 \cdot 7\right) \cdot 0{,}002 + 0{,}2 + 2 \cdot 0{,}5 \cdot 0{,}016 \Rightarrow \qquad (3)$$




**Figur 14: Cykeltiden för InterBus-S vid 500 kbps.**

När man undersöker prestandan för olika protokoll är det viktigt att ta hänsyn till effektiviteten på protokollet. För att bestämma effektiviteten hos protokoll kan man definiera följande kvot:

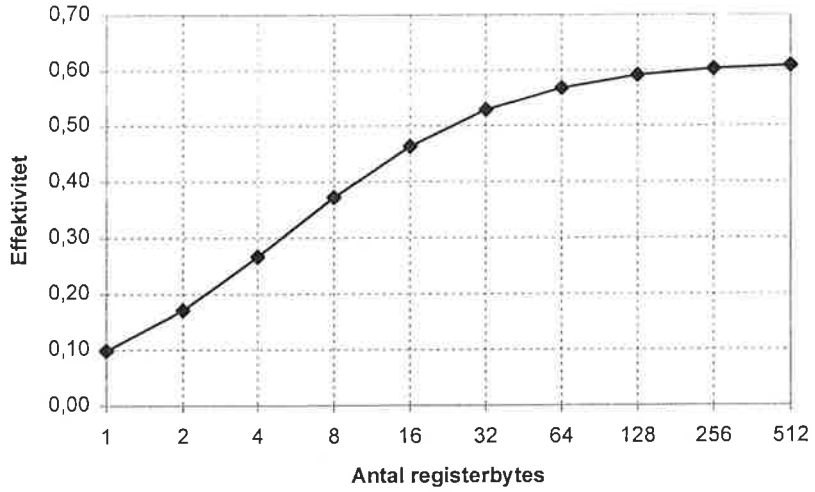
$$\text{Effektivitet} = \frac{\text{Nyttig information}}{\text{Nyttig information} + \text{Kontrollinformation}} \tag{4}$$

Om man i fallet med InterBus-S gör det förenklande antagandet att all överförd data varje cykel bara består av nyttig information, så får man följande uttryck för effektiviteten på InterBus-S:

$$\text{Effektivitet} = \frac{R \cdot 8}{R \cdot 8 + \left(16 + 32 + \left(R + 4\right) \cdot 5\right)} \tag{5}$$

där R är det totala antalet registerbytes för alla moduler på bussen.

Man ser att effektiviteten blir starkt beroende av antalet anslutna enheter på bussen. Detta beror på att all in- och utdata uppdateras av ett gemensamt meddelande under varje cykel på bussen (enligt principen för summa-ram protokoll beskrivet i avsnitt 3.4.1 ovan). Detta meddelande innehåller en fix del som är oberoende av antalet anslutna enheter vilket ger dålig effektivitet vid få anslutna enheter. I Figur 15 nedan presenteras effektiviteten för InterBus-S som funktion av antalet registerbytes för alla moduler på bussen.




**Figur 15: Effektiviteten för InterBus-S som funktion av antalet registerbytes.**

Protokollet har en asymptotisk effektivitet på 0,6154 som i stort sett uppnås redan vid 100 registerbytes. Man skall dock inte stirra sig blind på protokollets effektivitet, utan även tänka på att cykeltiden förlängs med fler ansluta moduler. I slutändan är det cykeltiden som bestämmer hur snabb den process som skall styras eller övervakas kan vara.

# 4 Drivrutin till Windows NT

## 4.1 Allmänt

Windows NT är ett mikro-kernel baserat operativsystem gjort av Microsoft. Det lanserades 1993 (som version 3.1) och har idag nått version 3.5. Målet med Windows NT var att skapa ett operativsystem som uppfyllde ett antal krav:

- **Portabel** - Hårdvara utvecklas snabbare än mjukvara och operativsystemet måste ha möjlighet att flyttas till nya arkitekturer allt eftersom utvecklingen går vidare.
- **Skalbar** - Operativsystemet skall ha inbyggt stöd för en ständigt växande hårdvara som till exempel fler processorer, mer minne och större hårddiskar.
- **Delad data** - Allt eftersom kommunikation mellan datorer blir allt vanligare måste operativsystemet kunna utnyttja detta med till exempel distribuerat filsystem och delad hårdvara.
- **POSIX-stöd** - Att kunna stödja det standardiserade (IEEE 1003), UNIX-baserade, gränssnittet för operativsystem är viktigt för att till exempel kunna sälja system till amerikanska staten.
- **Säkerhet** - Att ge säkerhet enligt klass C2 (beskrivet i den amerikanska försvarsstandarden DoD 5200.28-STD). Detta innebär i korthet att all data har en ägare som kan kontrollera vem som får accessa den.

Som ett led i att uppfylla kraven ovan har man valt att implementera operativsystemet modulärt, med placering av alla moduler så att de kör i något av de två olika lägen som finns i operativsystemet. Det ena av dessa två lägen kallas *Kernel mode* och innehåller de delar av operativsystemet som sällan ändras och måste vara stabila i alla lägen. Här placeras systemtjänster som till exempel säkerhet, drivrutiner för fysiska enheter och själva kärnan. I det andra läget, *User mode*, finns förutom tillämpningsprogrammen (som till exempel Word och

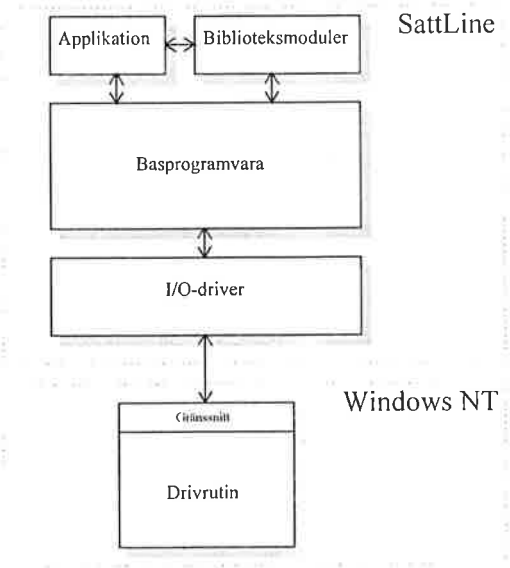
Excel) även undersystem för att kunna köra applikationer från andra operativsystem. Windows NT kan köra applikationer från till exempel Windows, POSIX och äldre OS/2.

Som en viktig del i detta examensarbete ingår en drivrutin för Windows NT. Detta är en drivrutin som direkt hanterar hårdvaran på instickskortet för InterBus-S, vilket leder till att drivrutinen måste skrivas i Kernel mode. Att en drivrutin skrivs i Kernel mode leder till en något speciell situation eftersom en drivrutin av det slaget, som havererar, tar också med sig hela systemet ner. Problemet är att hinna se vad som gick snett och lokalisera var felet fanns. För detta ändamål finns det framtaget en Kernel debugger vilken möjliggör att debugga drivrutinen på källkodsnivå. För att genomföra detta krävs att man har tillgång till två datorer vilka båda kör Windows NT. Det praktiska förfarandet att sätta upp en miljö för Kernel debugging finns beskrivet i appendix 9.2.

## 4.2 Gränssnitt i minnet mellan InterBus-S och tillämpningar

### 4.2.1 Förutsättningar och problem

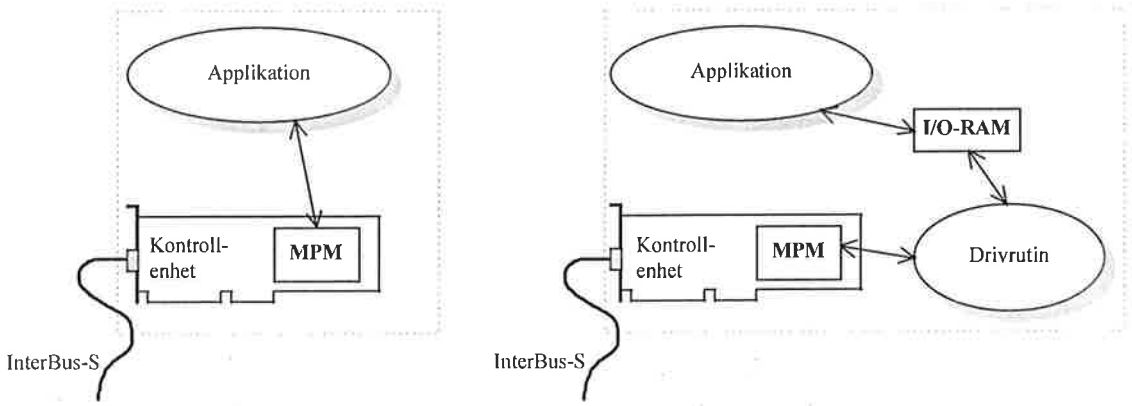
En förutsättning för examensarbetet var att använda ett PC-kort för InterBus-S utvecklat av Phoenix Contact GmbH & Co. Detta kort är utrustat med ett standard kontrollerkort för InterBus-S (baserat på en processor av Motorola-typ), vilket är av samma typ som man kan hitta i de flesta InterBus-S-kontrollenheter på marknaden (till exempel i den för Siemens Simatic S5). Detta speciella InterBus-S-kort sitter i vårt fall på ett PC-instickskort, vilket innehåller interface-, busskontroll- och övervakningslogik. För användaren presenterar kortet ett multiportsminne (MPM) där kommandon och meddelanden kommuniceras mellan kontrollenheten och PC:n. I MPM lagras även in- och utdata till och från processen. Detta MPM är 16 kbytes stort, men kan på grund av hårdvaran på kortet bara adresseras i 4 kbytes åt gången. För att kunna adressera hela minnet finns det ett minnesfönster på 4 kbytes, vilket kan flyttas runt i hela minnet för att möjliggöra läsning eller skrivning på önskad adress. Fönstret flyttas genom adressering via portar, vilka även används för initiering och kontroll av PC-kortet.




**Figur 16: Drivrutinens placering i förhållande till SattLine.**

En för examensarbetet viktig frågeställning var hur hårdvaran för InterBus-S skulle presenteras och kontrolleras av användaren. Figur 16 ovan visar drivrutinens placering i förhållande till användaren (det vill säga SattLine). Det finns egentligen bara två huvudalternativ att göra

gränssnittet mot användaren på. Det första alternativet är att bara presentera MPM precis som det är och låta användaren ta hand om de egenheter kortet har med tanke på annan processorarkitektur och minnesfönster. Alternativ två är att införa ett I/O-RAM, vilket är en spegling av de intressanta minnesareorna från kortet som har behandlats av drivrutinen och lagts i PC:ns eget minne. Frågan är nu vilket man bör välja?




**Figur 17: Alternativ för gränssnitt mot InterBus-S instickskort.**

Figur 17 ovan illustrerar hur informationen mellan InterBus-S och en applikation transporteras i fallet med direkt tillgång till MPM, respektive fallet med ett mellanliggande I/O-RAM.

För att lättare kunna fatta ett riktigt beslut finns det nedan en lista med olika för- och nackdelar. Det är inget självklart val att välja en av metoderna, utan beslutet grundar sig på övervägande av de olika för- och nackdelar.

**Det som talar för införandet av ett I/O-RAM är:**

- InterBus-S-kortet har en Motorola-processor och således omvänd byte-ordning jämfört med den Intel-processor som finns i PC:n. Detta kan döljas med I/O-RAM genom att drivrutinen vänder rätt på byte-ordningen.

- För att få tillgång till hela MPM på InterBus-S-kortet måste man flytta ett 4 kbytes stort fönster fram och tillbaka vid access till olika delar av minnet. Även detta kan döljas med I/O-RAM genom att drivrutinen sköter flyttning av fönster vid behov.

- Man kan genom att välja I/O-RAM gardera för framtiden när det kanske kommer ett PC-kort med Intel byte-ordning eller utan minnes-fönster. Då slipper man att ändra i allt utom drivrutinen och då kan man utnyttja direkt tillgång till MPM.

**Det som talar mot införandet av I/O-RAM är:**

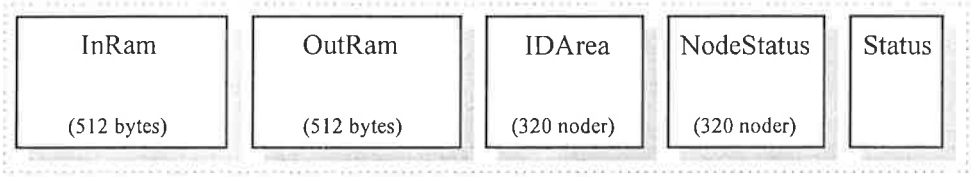
- Eventuella prestandaförluster eftersom man måste kopiera in hela I/O-RAM även om bara en signal bland alla andra signaler på InterBus-S är eftersökt.

- Besvärligt att garantera konsistens vid kopiering av signaler (av olika längder) från MPM till I/O-RAM och tillbaka. Speciellt inom moduler med stor registerlängd.

### 4.2.2 Lösning och implementation

Att bara presentera det gränssnitt som hårdvaran ger verkade inte vara ett bra alternativ,

eftersom mycket arbete då måste läggas in i SattLine för att tolka den tillgängliga informationen från kortet, samt för att kunna styra kortet. Under det här examensarbetet har valet därför fallit på I/O-RAM i stället för på MPM-access. Detta ger oss kanske något sämre prestanda, men genom att införa I/O-RAM har man skaffat en mer framtidssäker lösning genom att dölja för användaren de egenheter kortet har.




**Figur 18: Gränssnittet mot drivrutinen med dess fem minnesareor.**

Figur 18 ovan beskriver det gränssnitt som drivrutinen presenterar mot användaren. Areorna InRam och OutRam är de som tidigare kallats I/O-RAM, medan övriga areor har tillkommit för att på ett enkelt sätt för användaren presentera den information som kortet kan ge om det anslutna InterBus-S-nätet.

Arean för NodeStatus innehåller ett boolskt värde per möjlig ansluten modul som indikerar om modulen fungerar korrekt. IDArea innehåller ID- och längdkoder för de anslutna modulerna och slutligen arean Status, vilken innehåller gemensam information för InterBus-S. I fallet med InterBus-S är informationen denna: om bussen kör, om det har inträffat något fel, antalet anslutna noder samt hur många bytes dessa noder använder av in- och ut-areorna. För mer information om implementeringen hänvisas till avsnitt 4.4 nedan samt referens [4]. Om man tittar tillbaka på Figur 16, kan man se att gränssnittet mot SattLine i drivrutinen är en del av det som beskrivs i Figur 18 ovan. Dessutom tillkommer i gränssnittet de funktioner som är beskrivna i avsnitt 4.3 nedan.

En viktig konsekvens av dessa areor, vilka drivrutinen ger som gränssnitt, är att man skulle kunna tänka sig att använda dessa för andra liknande fältbussar (eller olika typer av I/O-kort), eftersom snittet är så pass generellt att det skulle kunna passa även andra bussar och kort.

## 4.3 Funktionsgränssnitt mellan InterBus-S och tillämpningar

### 4.3.1 Krav och önskemål på funktionalitet

När det gäller vilken funktionalitet man bör erbjuda användaren blir den nog ofta en kompromiss mellan egna önskemål och det som verkligen kortet och InterBus-S har stöd för. Inom detta examensarbete har följande funktionalitet hos drivrutinen ansetts som rimlig för att kunna styra och övervaka InterBus-S från SattLine:

- **Konfigurering**. Detta innebär att kunna ta reda på vilka typer av moduler som är anslutna till bussen och hur många de är. Man måste dessutom kunna ta reda på i vilken ordning som modulerna sitter på bussen för att kunna räkna ut var varje moduls in- och utgångsregister kommer att hamna i MPM och I/O-RAM. Denna funktion bygger helt på körandet av en ID-cykel, vilken är beskriven i avsnitt 3.4.2 ovan.

- **Start och stopp**. Det är ganska självklart att man måste ha funktionalitet för att starta (och eventuellt stoppa) datatrafiken på bussen. Start av datacykeln, beskriven i avsnitt 3.4.3, bör ske med ett kommando, men stopp av bussen skulle kunna ske när man stänger drivrutinen.

- **Felkontroll och felhantering**. Det borde finnas automatisk detektering av fel som uppkommer på InterBus-S. Dessa fel skulle sedan kunna förmedlas till användaren genom de areor som är beskrivna i avsnitt 4.2.2 ovan. Dessutom borde det finnas funktioner för att själv kontrollera om något fel har inträffat, samt möjlighet att kvittera dessa fel allt eftersom man har reparerat dem ute på InterBus-S-nätet.

- **Information om PC-kort**. Detta kan ge viktig information om den anslutna hårdvaran för att identifiera att rätt kort sitter i datorn och vilken version kortet har.

Dessa generella önskemål på drivrutinen måste nu passas in mot de krav på hur en drivrutin för Windows NT bör implementeras.

### 4.3.2 Lösning och implementation

Drivrutiner till Windows NT måste följa en standard för vilka anrop som kan göras till drivrutinen. Dessa anrop kallas "Major functions" och är nio till antalet och av dessa är vissa obligatoriska för alla drivrutiner och vissa valfria. Antalet funktioner är som synes starkt begränsat, men man skall ändå kunna stödja all tänkbar hårdvara till Windows NT. Hur går det ihop? För att man skall kunna skriva funktioner till drivrutiner som inte passar in under något standardanrop, finns det i ett av anropen, DeviceIoControl, möjlighet att tolka ett valfritt antal egendefinierade kommandon som implementeras av den som skriver drivrutinen. På så vis klarar man av olika typer av hårdvara.

För denna InterBus-S-drivrutin blir lösningen att implementera nedanstående fem "Major functions" och fem kommandon till DeviceIoControl. Av dessa är CreateFile och CloseHandle obligatoriska för alla drivrutiner.

| | |
|---|---|
| **CreateFile** | Öppnar kommunikationen mot drivrutinen för InterBus-S. Denna funktion kontrollerar till exempel att det verkligen sitter ett InterBus-S-kort i datorn och ger en återställningssignal till kortet. |
| **CloseHandle** | Stänger kommunikationen mot InterBus-S. Denna funktion får också vara ansvarig för att stanna datacykeln på bussen innan all kommunikation mot InterBus-S stängs. |
| **WriteFile** | Skriver ner en minnesarea från I/O-RAM till MPM. För att kompensera för den prestandaförlust som det innebär att kopiera till I/O-RAM istället för att direkt läsa och skriva i MPM, så har Read- och Write-kommandona räknat ut hur stor plats den anslutna bussen tar i minnet. Sedan kopierar dessa kommandon på så sätt minimal mängd data trots att in- och ut-areorna är 512 bytes vardera. |
| **ReadFile** | Kopierar upp en minnesarea från MPM till I/O-RAM. Har i övrigt samma funktionalitet som Write-funktionen ovan. |
| **DeviceIoControl** | Anropas för att ge kommandon till och hämta data från kortet som kontrollerar InterBus-S. Funktionen skulle även kunna användas för att i framtiden överföra parameterdata till moduler på bussen. |

De fem kommandona har implementerats enligt nedan och uppfyller de krav och önskemål som diskuterades i avsnitt 4.3.1.

**START_BUS_REQUEST:**

Utför tre operationer mot InterBus-S i sekvens. Först görs en ID-cykel som samlar in information och de anslutna modulerna. Därefter hämtas denna konfiguration upp från kortet och lagras i IDArean enligt avsnitt 4.2.2 ovan. Slutligen startas datacykeln, vilken fortsätter att gå tills drivrutinen stängs av CloseHandle

**SEND_SOFTWARE_REVISION_REQUEST:**

Hämtar information om PC-kortet, vilken innehåller bland annat namnet på tillverkaren, versionsnummer på kortets programvara och dess datum.

**GET_CONNECTION_REQUEST:**

Öppnar en förbindelse till I/O-RAM genom att returnera fem pekare till de olika areorna som utgör gränssnittet i minnet mot tillämpningsprogrammen.

**ACTIVATE_WATCHDOG_REQUEST:**

Detta kommando har två funktioner. Första gången det anropas så aktiveras en watchdog-funktion som sedan måste anropas med maximalt 146 ms mellanrum för att inte InterBus-S-kortets processor skall tro att PC:n har dött och därför nollställa alla utgångar på bussen.

**UPDATE_CONNECTIONS:**

Försöker återanslutna moduler på bussen som tidigare har drabbats av fel. Kommandot upptäcker även om nya moduler skulle ha blivit felaktiga sedan senaste anropet. För att indikera vilka noder som har fel och vilka som fungerar korrekt, uppdateras minnesarean för NodeStatus.

För mer ingående information om funktioner och kommandon hänvisas till referens [4].

## 4.4 Implementering

### 4.4.1 Språk och verktyg

Vid utvecklingen av drivrutinen har det huvudsakligen använts tre verktyg från Microsoft:

- Visual C++ v2.0
- Device Development Kit for Windows NT 3.5 (DDK)
- Win32 Software Development Kit For Microsoft Windows NT Version 3.5 (SDK)

Visual C++ är en integrerad miljö för programmering, kompilering och debugging. I detta verktyg har all källkod producerats och med hjälp av så kallade projekt har man kunnat gruppera ihop alla källkodsfiler som ingår i drivrutinen. Detta ger fördelen att man enkelt kan hoppa mellan olika källkodsfiler och rader i dem utan att själv behöva öppna och stänga filer. Trots att kompilatorn är gjord för C++ är drivrutinen skriven i vanlig C-kod, eftersom hela Windows NT är skrivet i C och inte tar nytta av några objektorienterade konstruktioner från C++ i den interna koden.

Som ett speciellt stöd för att utveckla drivrutiner till Windows NT, har Microsoft tagit fram DDK. Detta är ett paket med exempel, hjälpfiler, hjälpprogram och include-filer till C-kompilatorer speciellt ämnade för utveckling av drivrutiner. Bland hjälpprogrammen finns det bland annat speciella kommandon för att kompilera och länka drivrutiner. När man utvecklar programvara till Windows NT kompilerar man fram två olika versioner av drivrutinen: free och

checked build. Free build innebär att drivrutinen saknar debuginformation och extra kontroller och är den version som slutligen skall användas i det färdiga systemet. Checked build däremot innehåller debuginformation och används bara vid testningen av programvaran.

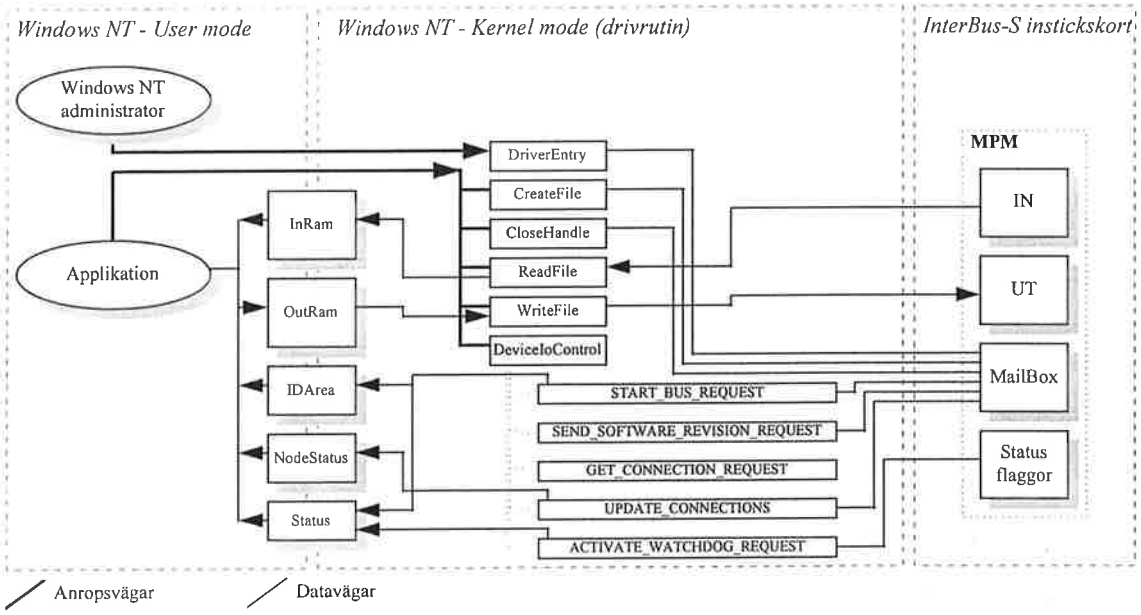
SDK är ett paket framtaget för allmän programutveckling till Microsofts operativsystem. Den version som har använts här är speciell för 32-bitars Windows (Windows NT och Windows 95) och innehåller även den exempel, hjälpfiler och hjälpprogram för utveckling av programvara. Bland hjälpprogrammen finns det debugprogram som kan debugga drivrutinen när den är installerad i systemet och kör i Kernel mode. I appendix 9.2 finns det beskrivet mer i detalj hur det går till att debugga en drivrutin i Kernel mode.

### 4.4.2 Design av drivrutinen

Figur 19 nedan beskriver förenklat hur kommunikationen mellan applikationer i Windows NT och InterBus-S instickskort går till. Man kan i figuren se att dess främsta uppgift är att förvandla det gränssnitt mot hårdvaran på InterBus-S-kortet till ett mer användarvänligt gränssnitt mot applikationerna. Figuren beskriver också den interna designen av drivrutinen i ett antal funktioner. Vad som inte syns i figuren är den interruptstyrda mailbox-hanteringen, vilken kommunicerar med InterBus-S-kortet enligt ett speciellt mailbox-protokoll specificerat av Phoenix Contact.

Det första som händer när en användare, som måste ha rättigheter som administrator, vill starta drivrutinen är att en funktion i drivrutinen kallad *DriverEntry* blir anropad. Varje drivrutin i Windows NT har en DriverEntry-funktion och dess uppgift är enligt följande:

- Kontrollera att rätt hårdvara är installerad och att den fungerar som den skall.
- Reservera de resurser i Windows NT som drivrutinen kommer att behöva senare under drift. För InterBus-S innebär detta bland annat en interruptsignal för mailbox-hanteringen, två semaforer för intern synkronisation och portar för kommunikation med kortet.
- Avbilda den fysiska minnesarea som MPM belägger till en plats i det virtuella minnet i Windows NT. Här är det viktigt att tala om för operativsystemet att MPM inte får lov att hamna i datorns cacheminne, vilket skulle hindra korrekt uppdatering av minnesarean. De minnesareor som drivrutinen sedan allokerar för applikationen kan däremot fritt efter behov läggas i cacheminnet för att få ökad prestanda.




**Figur 19: Kommunikation till och från drivrutinen.**

När DriverEntry är anropad går det bra att använda drivrutinen från applikationen. Det sker först genom att man anropar *CreateFile* och anger att man vill öppna kommunikation mot InterBus-S. Därefter bör man anropa *DeviceIoControl* med kommandot GET_CONNECTION_ REQUEST, vilket som tidigare beskrivits allokerar de minnesareor som bildar gränssnittet mellan drivrutin och applikation.

För mer detaljerad information om implementeringen hänvisas till referens [4] och den källkod som är skriven enligt funtionsspecifikationen.

## 4.5 Resultat

Som ett resultat av minnesareorna i gränssnittet har man fått införa ett nytt sätt att adressera signalerna i SattLine. SattLine använder i vanliga fall internt en fyrställig adress av typen XX.YY.ZZ.VV för att adressera I/O-signaler, med viss betydelse för varje del av adressen. I fallet med InterBus-S har den fyrställiga adressen fått en annan tolkning, vilken är beskriven i appendix 9.1. Detta är gjort för att klara boolska-, heltals- och analoga signaler för denna typ av I/O.

Det viktigaste resultatet av arbetet med drivrutinen till Windows NT var det generella gränssnittet i minnet. Med detta gränssnitt skulle man kunna ansluta andra fältbussar, I/O-kort, labbkort och så vidare. Ja, alla typer av I/O som kan fås att passa in i beskrivningen av minnesareor för data och status skulle gå att ansluta. Eftersom mycket av arbetet med integrationen i SattLine är gjort, så krävs det bara en för det aktuella kortet anpassad drivrutin och en uppsättning biblioteksrutiner för att kunna använda en ny fältbuss eller ett labbkort tillsammans med SattLine.

# 5 Integration i SattLine

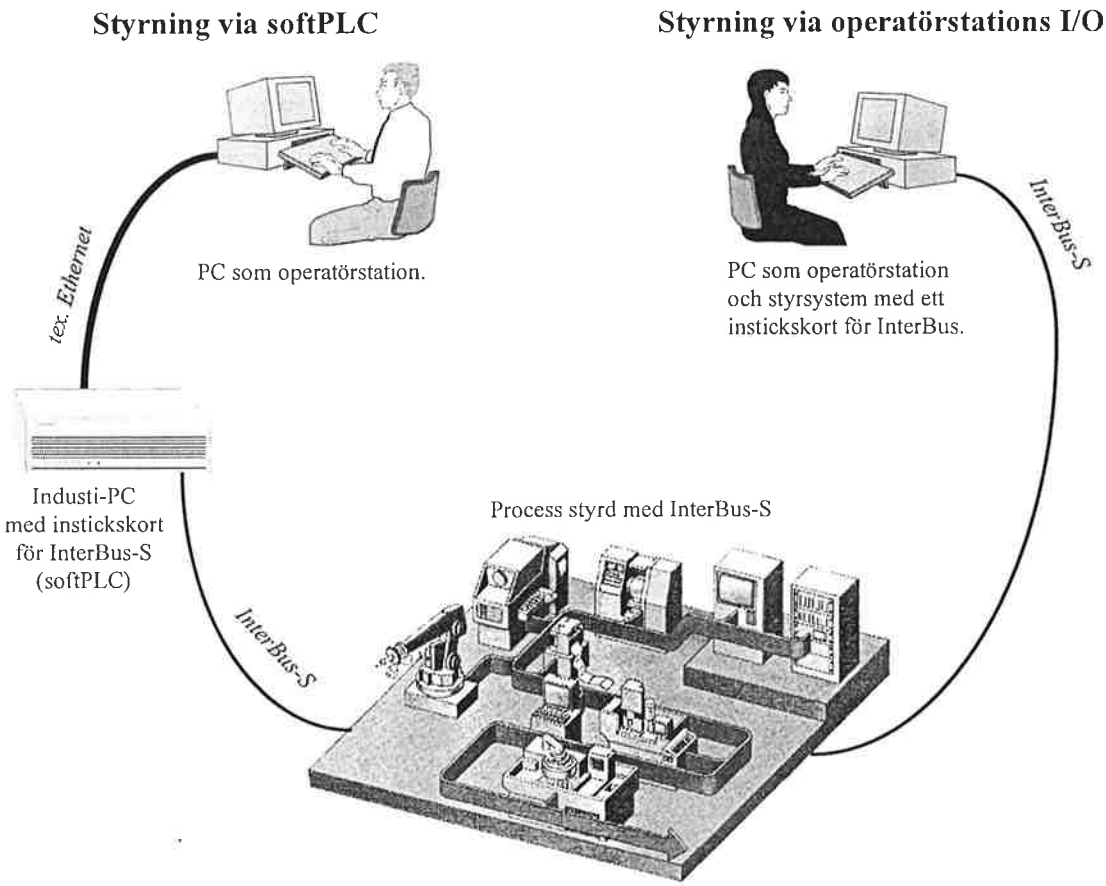
## 5.1 Operatörstations I/O eller softPLC

Detta examensarbete baserar sig på ett instickskort för PC-datorer, vilket gör det möjligt att

ansluta I/O direkt till PC:n. Detta är inte det sätt på vilket I/O vanligen har anslutits till processtyrningen fram till idag. Det vanligaste sättet har varit att ansluta I/O till speciellt dedicerade styrsystem tillverkade av automationsföretag som till exempel Alfa Laval Automation AB. Dessa styrsystem är specialbyggda av respektive företag och kan oftast inte använda några typer av standardiserade instickskort för till exempel ISA-bussen.

Vad man skulle kunna vinna med att använda standardiserad hårdvara, typ PC, är att kunna ta del av den enorma prestandaökning som sker på området med persondatorer. Kostnaden för denna utveckling delas av mängder av användare inom olika områden. Detta leder till att man kan få stora prestandaökningar bara genom att köpa ny hårdvara då och då. Att köpa sådan hårdvara, vars utvecklingskostnader delas av många, är mycket billigare än att som tillverkare av hårdvara själv utveckla och testa sina egna produkter tills de når en sådan prestanda.

Nu är det inte så enkelt att standardhårdvara skulle kunna användas överallt utan problem. Det finns aspekter som till exempel driftsäkerhet och framtidskompabilitet som man som användare av standardhårdvara inte har någon kontroll över. Vad man däremot skulle kunna använda standardhårdvara till (kompletterad med utrustning för driftsäkerhet) är att få de applikationer som har extremt höga prestandakrav ekonomiskt lönsamma. I de fallen är det troligen inte ekonomiskt försvarbart att försöka utveckla egen hårdvara, utan man skulle istället kunna köpa det bästa som finns på marknaden och integrera sina produkter med denna hårdvara.




**Figur 20: PC i automationssystem som softPLC eller operatörstations I/O.**

Det är här examensarbetet kommer in som ett exempel på hur man skulle kunna använda en PC i styrsammanhang. Det finns egentligen två olika alternativ att utnyttja PC:n på: som ett PC-

baserat styrsystem placerat ute i fabriken (softPLC), eller som en kombinerad operatörstation och styrsystem, vilket kallas operatörstations I/O. Figur 20 ovan illustrerar de olika användningarna av PC med instickskort för InterBus-S som del i ett automationssystem.

En annan komplikation med att använda en PC som styrsystem är att finna ett operativsystem som lämpar sig för uppgiften. Windows NT, som har använts för detta examensarbete, är inte specialbyggt för realtidsapplikationer, vilket kan vålla problem i speciellt tidskritiska applikationer. Den högsta tidsupplösning som Windows NT kan ge vid körningen av olika processer är idag 10 ms och detta sätter en gräns för vilken typ av reglering som är möjlig i en PC med Windows NT. Framtiden får utvisa om Windows NT kan förbättras i detta avseende, eller om man bör välja ett annat operativsystem.

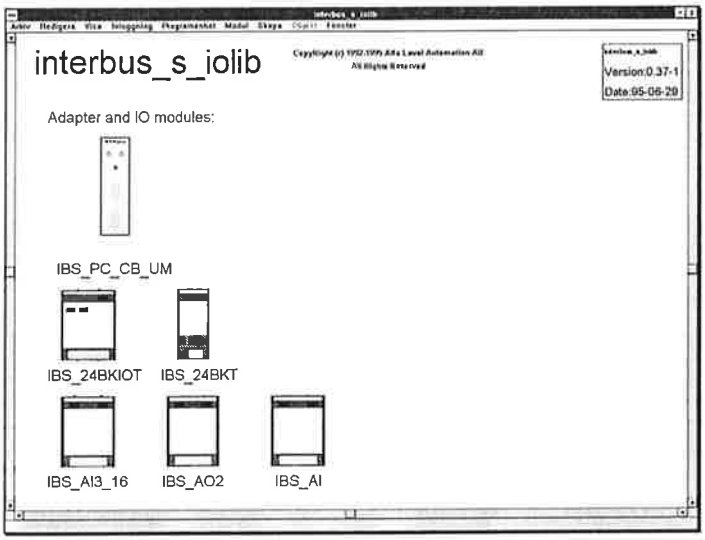
## 5.2 Införandet av biblioteksmoduler

Föregående avsnitt motiverar att det skulle gå att använda en PC i automationssystem, men hur kommer det att se ut för användaren? Kommer han att känna igen sig med InterBus-S i SattLine, eller kommer denna fältbuss att vara något nytt och udda?

Lösningen på problemet hur man skall integrera InterBus-S i SattLine är att använda sig av biblioteksmoduler. SattLine är objektorienterat i sin representation och en modul i ett bibliotek kan ses som en klass. Av denna klass kan man skapa ett godtyckligt antal instanser. För att integrera InterBus-S i SattLine behövs alltså ett bibliotek med moduler som representerar alla de InterBus-S-moduler man vill kunna använda samt en modul för kontrollenheten (i detta fall instickskortet för PC:n).

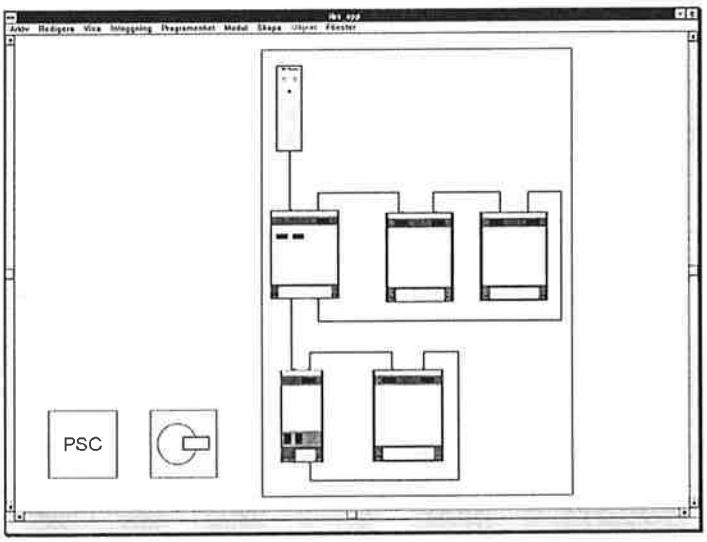
Figur 21 nedan illustrerar det bibliotek som gjordes under examensarbetet. En medveten begränsning av examensarbetet är att endast de InterBus-S-moduler som fanns tillgängliga på Alfa Laval Automation samt instickskortet för PC:n (kontrollenheten) har implementerats i biblioteket. Det är dock enkelt att lägga till nya InterBus-S-moduler med hjälp av vanlig SattLine-programmering. Detta görs genom att kopiera befintliga moduler och sedan införa de ändringar som är specifika för den nya modultypen.

Biblioteket i figuren innehåller förutom kontrollenheten (översta raden) två fjärrbussmoduler (andra raden), varav en med 16 digitala in- och utgångar och en utan in- eller utgångar. Det finns även tre lokalbussmoduler (understa raden) med följande data: en med 4 analoga ingångar, en med 16 multiplexade analoga ingångar och en med 4 analoga utgångar.




**Figur 21: Bibliotek med InterBus-S-moduler i SattLine.**

Dessa biblioteksmoduler sammankopplas med så kallade grafiska kopplingar i SattLine på ett liknande sätt som InterBus-S-moduler kopplas samman i verkligheten. Figur 22 nedan visar ett exempel på en uppkoppling med InterBus-S-moduler. Längst upp i bilden finns modulen för kontrollenheten. Denna är i sin tur kopplad med en grafisk koppling till en fjärrbussmodul (i detta exempel en modul med 16 digitala in- och utgångar). Från fjärrbussmodulen går en avstickande lokalbussledning åt höger med två anslutna moduler. Fjärrbussen fortsätter sedan till en ny modul utan in- eller utgångar som i sin tur har en lokalbussgren med en ansluten lokalbussmodul.



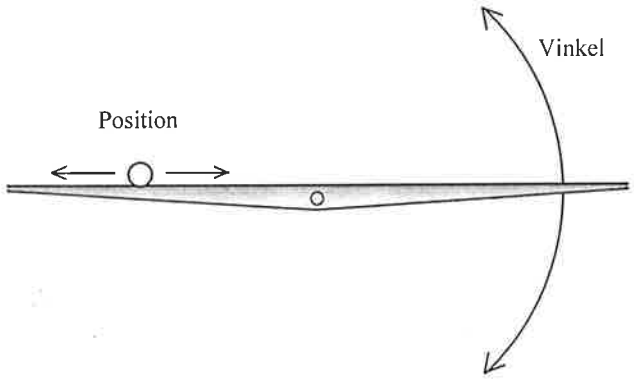

**Figur 22: Exempel på uppkoppling med InterBus-S-moduler i SattLine.**

Anledningen till att lokalbussmodulerna är kopplade i en slinga med grafiska kopplingar, är att kunna få varje fjärrbussmodul att detektera hur många lokalbussmoduler som är anslutna till fjärrbussmodulen.

# 6 Tillämpning: Bommen

## 6.1 Processen

Processen som skall regleras i denna tillämpning består av en vridbar horisontell bom med en drivmotor. Mellan två metallskenor utmed med bommens ovansida kan en kula rulla fritt och uppgiften går ut på att reglera kulans position på bommen.




**Figur 23: Mätbara storheter från processen "Bommen".**

Figur 23 ovan illustrerar de mätvärden man kan få från processen. Vinkeln mot horisontalplanet ($\phi$) mäts med en vridbar potentiometer fästad vid bommens rotationsaxel. Läget ($x$) mäts genom att en av de båda metallskenorna på bommens ovansida har hög resistivitet och när metallkulan befinner sig vid olika lägen så kortsluts olika stora delar av den resistiva skenan mot den andra skenan. Detta ger en spänning som utsignal, vilken är proportionell mot kulans position. Motorn som styr bommen är internt återkopplad med en takometer, vilket gör att den roterar med en hastighet som är proportionell mot dess styrspänning.

## 6.2 En modell av processen

En modell av bommen kan delas upp i en modell av vinkelprocessen, $G_\phi(s)$, och en modell av hur kulans läge påverkas av bommens vinkel, $G_x(s)$. Den totala överföringsfunktionen från motorns inspänning till spänningen som indikerar kulans position blir då $G_\phi(s)G_x(s)$.

### 6.2.1 Vinkelprocessen

Tack vare den interna återkopplingen av motorn blir bommens vinkelhastighet proportionell mot inspänningen $u(t)$, vilket ger följande enkla samband:

$$\frac{d\phi}{dt} = k_u u(t). \tag{6}$$

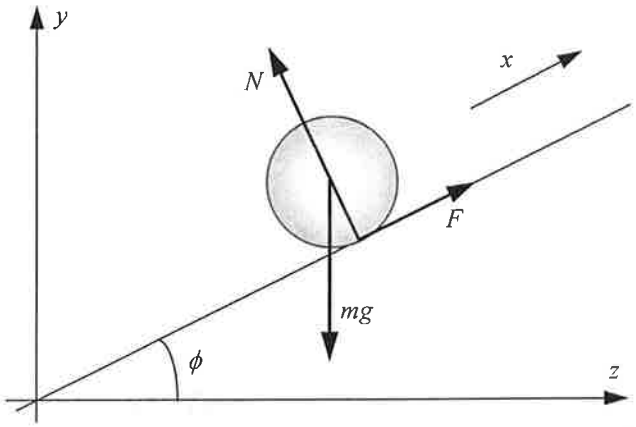
Detta ger överföringsfunktionen:

$$G_\phi(s) = \frac{k_u}{s} \tag{7}$$

Enligt experiment på reglertekniska institutionen, LTH, kan man uppskatta att $k_u \approx 4{,}4$.

### 6.2.2 Kulans läge

Dynamiken för kulans läge på bommen kan bestämmas med klassisk mekanik. Antag att kulans läge är $x$, bommens vinkel är $\phi$, kulans radie är $r$ och att kulans bana går genom centrum för bommens vridning. Figur 24 nedan inför de koordinater och krafter som behövs för modellen.




**Figur 24: Krafter och koordinater för modell av processen.**

Kraftekvationen ger följande samband

$$\begin{cases} m\dfrac{d^2y}{dt^2} = -mg + N\cos\phi + F\sin\phi \\ m\dfrac{d^2z}{dt^2} = -N\sin\phi + F\cos\phi \end{cases} \tag{8}$$

Multiplicera med $\sin\phi$ respektive $\cos\phi$ och addera, vilket ger:

$$m\left(\frac{d^2y}{dt^2}\sin\phi + \frac{d^2z}{dt^2}\cos\phi\right) = -mg\sin\phi + F \tag{9}$$

För en friktionsfritt rullande kula gäller att $Fr = -J/r \cdot d^2x/dt^2$ . För ett klot är $J = 2mr^2/5 \Rightarrow$ $F = -2m/5 \cdot d^2x/dt^2$ . Vidare gäller att $z = x\cos\phi$ och $y = x\sin\phi$ . Derivera dessa samband två gånger, multiplicera med $\cos\phi$ respektive $\sin\phi$ och addera så erhålles:

$$\frac{d^2y}{dt^2}\sin\phi + \frac{d^2z}{dt^2}\cos\phi = \frac{d^2x}{dt^2} - x\left(\frac{d\phi}{dt}\right)^2 \tag{10}$$

Sätt in detta uttryck i ekvation (9). Då erhålles:

$$m\left(\frac{d^2x}{dt^2} - x\left(\frac{d\phi}{dt}\right)^2\right) = -mg\sin\phi - \frac{2}{5}m\frac{d^2x}{dt^2} \tag{11}$$

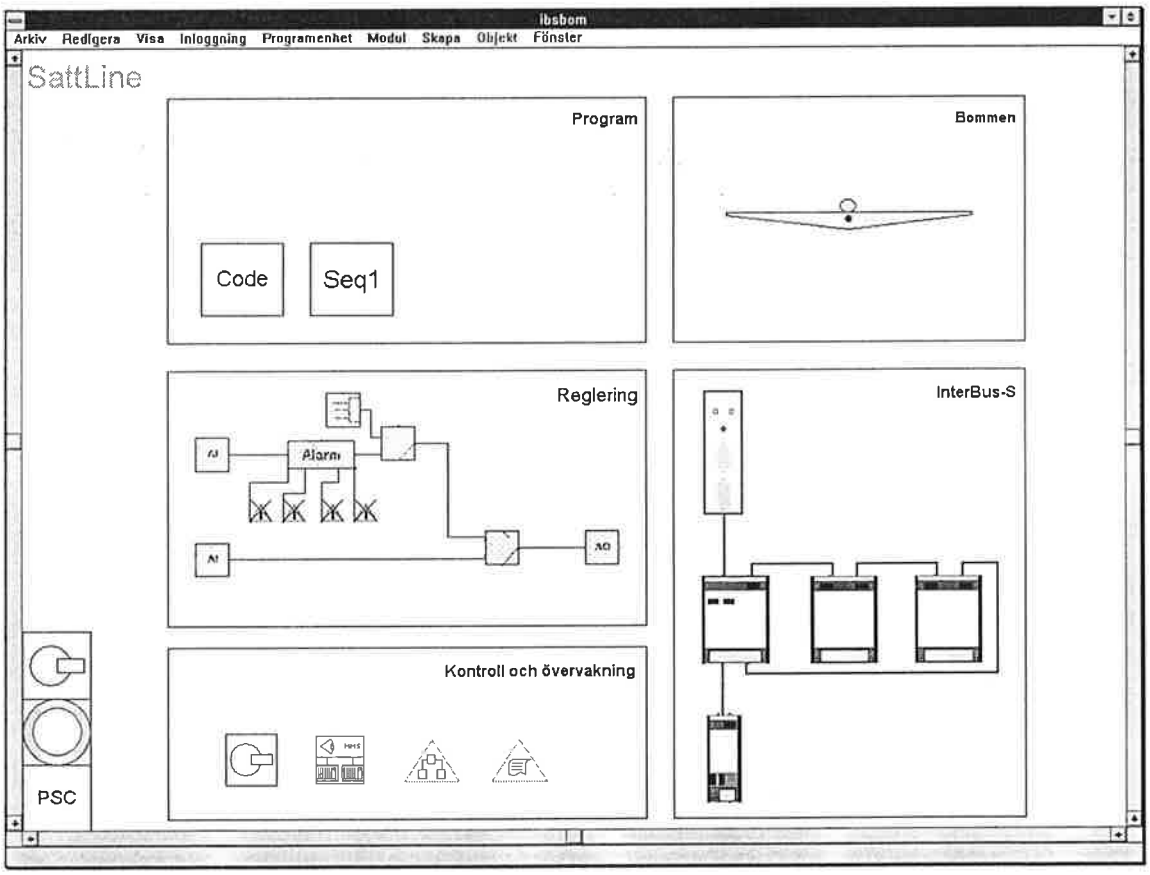
Om vi antar att $\phi$ och $d\phi/dt$ är små erhålles slutligen:

$$G_x(s) = -\frac{5g}{7s^2} \approx -\frac{7}{s^2} \tag{12}$$

Antagandet att $\phi$ och $d\phi/dt$ är små är rimligt vid reglering av kulans läge, eftersom relativt små ändringar av vinkeln sätter bra fart på kulan. Om man däremot skulle använda regulatorn för att till exempel kasta iväg kulan, skulle dessa antaganden inte längre vara rimliga.

## 6.3 Programmering i SattLine

Figur 25 nedan innehåller resultatet av hela examensarbetet. Bakom denna bild som presenteras för användaren vid operatörsstationen, kör instickskortet de data- och ID-cyklar som beskrevs i kapitel 3. För att kommunikationen mot SattLine skall fungera används drivrutinen beskriven i kapitel 4 och slutligen används biblioteksmodulerna från kapitel 5 för att göra inkopplingarna av signaler i SattLine. På så vis kommer alla delar av examensarbetet in i detta exempel.




**Figur 25: Reglering av processen "Bommen" med InterBus-S.**

Bilden är uppdelad i fem sektioner beroende på funktionalitet. Nedan följer en kort beskrivning av innehållet i varje sektion:

**Program** Här finns två små program som dels flyttat kulan mellan två lägen på bommen och dels blinkar med 16 lysdioder på den digitala in- och utgångsmodulen.

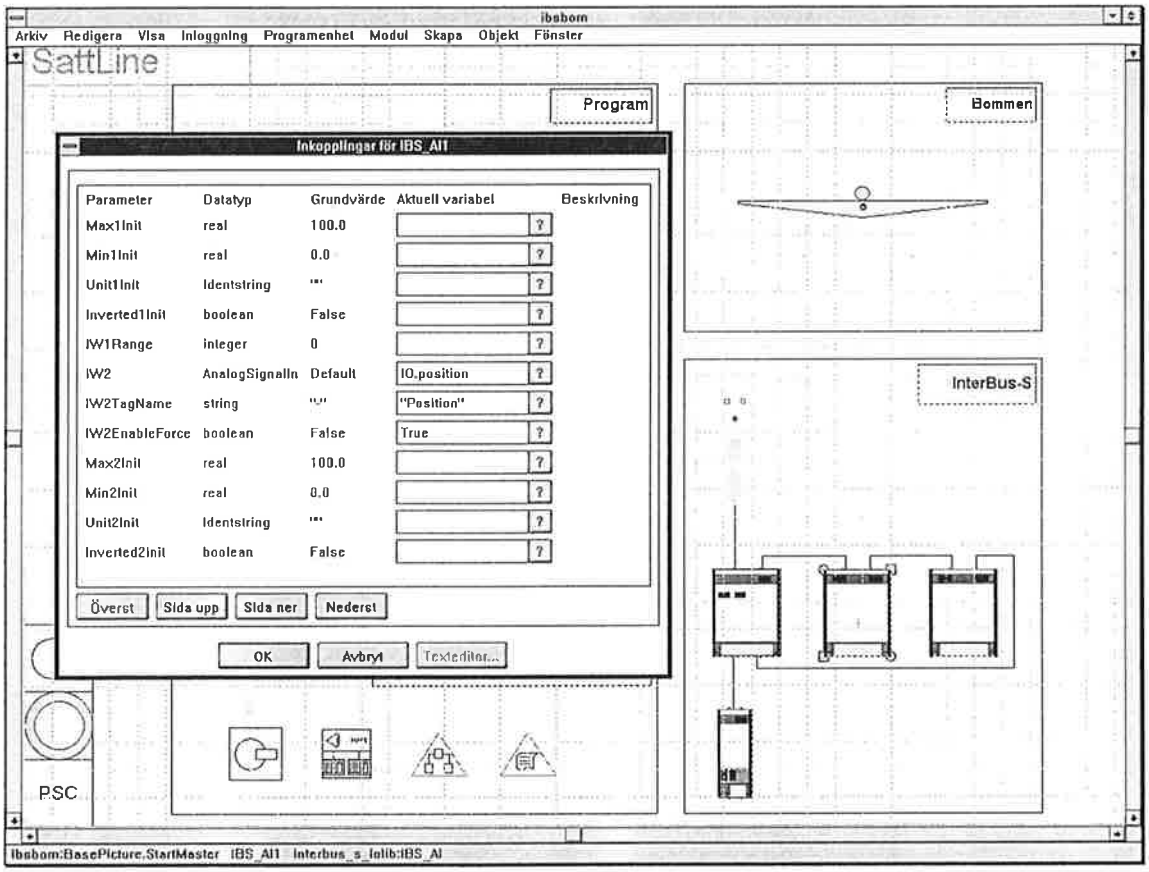
**Bommen** Detta är en animation som visar kulans position och bommens vridning efter de mätvärden som SattLine hämtar från processen.

**Reglering** Här sköts regleringen av bommen med hjälp av två kaskadkopplade PID-regulatorer. En regulator används för vinkelreglering och en för positions-reglering, enligt modellen som beskrevs i avsnitt 6.2 ovan.

**InterBus-S** I denna sektion kopplas InterBus-S-modulerna samman enligt det nät man har anslutit till datorn. Här kopplas också signaler till och från processen in.

**Kontroll och övervakning** I den sista sektionen finns några olika funktioner som till exempel kontroll av kommunikationen.

Dessutom finns ute i vänsterkanten ett par funktioner gemensamma för hela bilden. För mer information om dessa funktioner och om SattLine-programmering i allmänhet, hänvisas till referens [3].




**Figur 26: Inkoppling av variabler i SattLine till InterBus-S.**

Figur 26 ovan visar hur signaler i SattLine kopplas till InterBus-S via biblioteksmodulerna. I figuren visas inkopplingen av den analoga signalen som mäter kulans position på bommen. Signalen IO.position är här inkopplad till den analoga ingången IW2 på InterBus-S modulen IBS_AI1, vilket är en modul med fyra analoga ingångar. Dialogrutan för att mata in dessa inkopplingar får man enkelt genom att välja modulen (markeras med gröna hörn på modulen) och ur menyn välja funktionen *Inkopplingar*.

# 7 Utvärdering

## 7.1 Resultat

Resultatet av examensarbetet visar att det går bra att integrera kommunikation mot InterBus-S i en applikation under Windows NT. Det framkommer också att det efter detta examensarbete

skulle vara ganska enkelt att anslutna nya fältbussar eller andra I/O-kort, om man bara kunde få informationen från dem att passa in i de minnesareor som har tagits fram här. Om man dessutom höjer blicken något inser man att andra liknande bussar skulle gå att koppla till InterBus-S genom att koppla samman dem via en gateway. Redan idag finns det möjlighet att ansluta ASI och Sensor Loop till InterBus-S på detta sätt.

Att integrera InterBus-S i SattLine via biblioteksmoduler gör att I/O-signaler ser ut som användaren av SattLine är van vid. Detta är viktigt för att nya typer av I/O skall kunna införas med så liten inlärningströskel för användaren som möjligt. I fallet med InterBus-S i SattLine finns det dock två typer av gamla användare: de som tidigare jobbat med SattLine och de som tidigare jobbat med InterBus-S.

Den gamla InterBus-S-användaren är van vid att ha väldigt lite stöd av programvaran han använder, utan får själv hålla reda på vilka bitar eller bytes som skall adresseras för att få tag i rätt signaler. Så som adresseringen av signaler har lösts i detta examensarbete göms adresseringen med bitar och bytes för användaren och inkopplingarna görs grafiskt i biblioteksmoduler. Detta gör att en grupp av användare får lära om och det är min åsikt att man borde låta de gamla InterBus-S-användarna lära om sig till förmån för det mer användarvänliga sättet som används i SattLine för att kommunicera mot InterBus-S.

## 7.2 Framtida utvecklingar

Det var ej syftet med detta examensarbete att göra en produkt som var färdig att sälja. Det finns därför funktionalitet som ännu inte är implementerad och det finns idéer om framtida utvecklingar och saker som kan undersökas. I detta avsnitt finns dessa tankar samlade för att underlätta framtida arbete och underhåll av det som kommit fram under examensarbetet.

1. Att införa kommunikation av parameterdata till och från InterBus-S-moduler med hjälp av PMS skulle man ganska enkelt kunna implementera genom att ge drivrutinen nya kommandon som ger möjlighet att skicka, buffra och ta emot PMS-meddelande. Om dessa kommandon kan göras lika de som finns för FMS i PROFIBUS, skulle integrationen i SattLine bli ganska enkel.
2. På instickskortet för InterBus-S finns det plats för en extra processor av 386-typ. Denna processor kan köra program oberoende av processorn i PC:n och har eget minne och egen kommunikation mot kontrollenheten. Denna extra beräkningskraft skulle man kanske kunna ta nytta av genom att låta en del av drivrutinen köra på den extra processorn. Det som skulle kunna köras här är till exempel uppdatering av I/O-RAM, hantering av felmeddelanden och återanslutning av moduler efter kommunikationsproblem.
3. Kommandona ACTIVATE_WATCHDOG_REQUEST och UPDATE_CONNECTIONS till drivrutinen är inte implementerade.
4. Kopieringsrutinerna som omvandlar värden mellan AD/DA-omvandlarna till värden som SattLine kan hantera är bara skrivna för att hantera moduler som ger 0-10 V och 0-20 mA. För att kunna hantera de andra två vanliga signaltyperna 4-20 mA och +/-10 V, behövs det en mindre ändring i kopieringsrutinerna.

# 8 Referens- och litteraturlista


[1] Baginski Alfredo & Müller Martin, "InterBus-S, Grundlagen und Praxis", Hütig Buch Verlag GmbH, Heidelberg, 1994

[2] Custer Helen, "Inside Windows NT", Microsoft Press, Washington, 1993

[3] Johannesson Göran, "Objektorienterad processautomation med SattLine", Studentlitteratur, Lund, 1994

[4] Persson Per, "Funktionsspecifikation, NT-drivrutin för InterBus-S", ADL-9505-111 (02), Alfa Laval Automation AB, Malmö, 1995

[5] Stallings William, "Data and Computer Communications", 4th edition, MacMillan Publishing Company, New York, 1994

[6] "Date Up 1995 Automotive Special", Phoenix Contact GmbH & Co., Blomberg, 1995 (tidskrift)

[7] "Kernel-mode Device Driver Guide", Microsoft Windows NT 3.5 Device Driver Kit, April 1995 (manual)

[8] "User Manual for InterBus-S IBS Controller Board for IBM-Compatible PCs", Phoenix Contact GmbH & Co., Blomberg, 1993 (manual)

# 9 Appendix

## 9.1 Adressering av signaler i SattLine

De minnesareor som drivrutinen för InterBus-S gör tillgängliga för SattLine att knyta variabler i, är indelade i fem områden: InRam, OutRam, IDArea, NodeStatus och Status. Dessa areor har fått nummer från 0 till 4 enligt nedan:

| Area | Areanummer |
|---|---|
| InRam | 0 |
| OutRam | 1 |
| IDArea | 2 |
| NodeStatus | 3 |
| Status | 4 |

SattLine använder sig av en fyrställig adressering av IO-signaler, av typen XX.YY.ZZ.VV. De ingående delarna har följande betydelse i normala fall och vid InterBus-S:

| Position | Normal betydelse | Betydelse vid InterBus-S |
|---|---|---|
| XX | Modulnummer | Modulnummer |
| YY | Nodnummer | Nodnummer |
| ZZ | IO-modulnummer | Minnesarea och ByteOffset |
| VV | Kanalnummer | Bit eller upplösning och område |

Fältet för Minnesarea och ByteOffset kodas enligt följande:

$$Värde = Areanummer*65536 + ByteOffset$$

Fältet för "bit eller upplösning och område" innehåller för digitala signaler den bit som adresseras och för analoga signaler ett kodat 32-bitars tal, där 16 bitar anger det antal bitar som AD/DA-omvandlaren använder och 16 bitar anger området för signalen. I övriga fall saknar fältet innebörd.

Värdet för upplösning och område beräknas enligt följande:

$$Värde = Område*65536 + Upplösning$$

Koden för område har följande tolkningar:

| Område | Tolkning |
|---|---|
| 0 | In- eller utgång avstängd |
| 1 | 0-10 V eller 0-20 mA |
| 2 | 4-20 mA |
| 3 | +/- 10 V |

***Exempel:*** Antag att man vill adressera en 12-bitars analog utgångssignal, 0-10 V, som finns på nod nummer 3 och denna nod har ByteOffset 8. Detta ger, om man antar att instickskortet för InterBus-S har modulnummer 14, adressen: 14.3.65544. 65548.

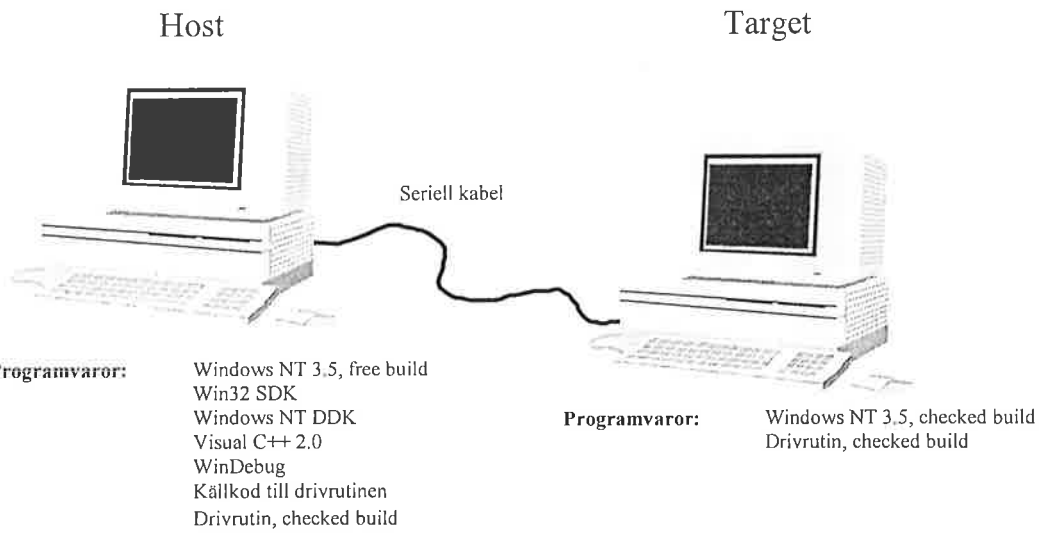
## 9.2 Kernel debugging i Windows NT

### 9.2.1 Inledning

Detta dokument beskriver hur man kan göra debugging på källkodsnivå i en Kernel mode driver under Windows NT 3.5. För att kunna göra detta måste man ha tillgång till viss utrustning och vissa produkter. Det som behövs är:

- Två datorer där den ena har relativt mycket ledig plats på hårddisken (cirka 150 Mbytes).
- En seriell nollmodemskabel.
- Win32 SDK for Windows NT 3.5
- Windows NT 3.5 DDK
- Microsoft Visual C++ 2.0

Figuren nedan ger en förenklad översikt på hur en färdig felsökningsmiljö kan se ut. Om man har tillgång till olika kraftfulla datorer, bör man välja att ha den mest kraftfulla som Host. Detta eftersom de mest krävande tillämpningarna kommer att köra på Host-maskinen (Visual C++ och build), medan Target-maskinen kommer att stå stilla under långa perioder.




### 9.2.2 Att sätta upp en Host-maskin

Saker att göra:

1. Installera en free-build version av Windows NT 3.5 (det vill säga en helt vanlig version).
2. Installera SDK, DDK och Visual C++ med versionsnummer enligt ovan (eller nyare).
3. Gör *checked-build* på din drivrutin med hjälp av verktygen som kom från DDK-skivan och kopiera .sys-filen till ett lämpligt katalog på Host-datorn. Denna katalog kommer att användas för Host-maskinens filer med debuginformation om drivrutinen och Windows NT, så att Host-maskinen kan följa programmen och drivrutinerna som exekveras på Target-maskinen.

4. Kopiera filerna ntoskrnl.dbg (katalog: \\support\debug\i386\symbols\exe\ ) och hal.dbg (katalog: \\support\debug\i386\symbols\dll\ ) från DDK-skivan till samma katalog som du kopierade din egen drivrutin till på Host-datorn.

5. Anslut den seriella kabeln till COM1-porten.

### 9.2.3 Att sätta upp en Target-maskin

Saker att göra:

1. Installera en checked-build av Windows NT 3.5 (se avsnitt 9.2.6 för detaljer).

2. Kopiera .sys-filen från *checked-build* (punkt 3 i avsnitt 9.2.2 ovan) till Target-datorns c:\nt35chk\system32\drivers\ (eller motsvarande katalog).

3. Registrera drivrutinen med till exempel: REGINI <drivername.INI>. För exempel på .ini-fil se bilaga i avsnitt 6.2. Observera att detta bara behöver göras första gången man skall börja debugga en drivrutin. I fortsättningen behöver man endast uppdatera den ändrade .sys-filen.

4. Editera boot.ini som ligger i rotkatalogen (se bilaga i avsnitt 9.2.8).

5. Anslut den seriella kabeln till vald COM-port.

6. Boota om datorn och välj versionen av Windows NT med Kernel debugging aktiverad i den valbara booten.

### 9.2.4 Debugging

Här följer ett exempel hur man kan börja att debugga sin källkod i drivrutinen:

1. Starta WinDebug på Host-maskinen. Välj "Options" i menyn och alternativet "User Dlls...". Mata in i rutan "Symbol Search Path" den katalog som du valde att lägga filerna från avsnitt 9.2.2, punkt 3 och 4 i.

2. Välj sedan "Debug"-alternativet från "Options"-menyn. Mata där in sökvägen till dina källkodsfiler i rutan "Source Search Path".

3. Välj åter "Options" från menyn och alternativet "Kernel Debugger...". Kryssa i rutorna för "Enable Kernel Debugging" och "Go on Exit". Välj också samma kommunikationshastighet som valdes i boot.ini-finen i Target-maskinen, se avsnitt 9.2.8 nedan. Välj också COM1 och låt övriga alternativ vara.

4. Öppna nu källkoden till drivrutinen som skall testas. Detta görs med "File" - "Open..." i menyn. Det går nu bra att sätta en brytpunkt i källkoden (t.ex. DriverEntry).

5. Nu är det ett bra tillfälle att spara de inställningar som gjorts. Välj"Program" - "Save As..." i menyn och mata in ett bra namn. Dessa inställningar är det bara att hämta in nästa gång drivrutinen skall provköras.

6. Aktivera debuggern genom att skriva "GO" i "Command"-fönstret. Den ställer sig nu och väntar på ett avbrott från Target-maskinen. Tryck kontroll-C på Host-maskinen och

kommunikationen skall vara igång. Tryck på "play-knappen" i verktygsfältet för att gå vidare".

7. På Target-maskinen kan man starta "Control Panel", "Drivers" och leta upp sin drivrutin. Välj den och tryck på "Start". Nu skall Host-maskinen hänga sig och brytpunkten som sattes i DriverEntry ovan kommer att aktiveras.

8. Nu är det bara att debugga som vanligt med WinDebug och Target-maskinen rör sig bara framåt på kommando från Host-maskinen. Det fungerar precis som vanligt att inspektera variabler m.m..

9. **Lycka till!**

### 9.2.5 Mer information

Mer information om att avlusa en Kernel mode driver kan man hitta i hjälpfilen Getting Started som installeras med DDK. Ytterligare hjälp finns i "DDK Help" kapitel 4 och då speciellt avsnitt 4.1.

Mer information om WinDebug hittas lättast med hjälp av programmets egen hjälpfunktion.

### 9.2.6 Installation av Windows NT 3.5 checked build

Det enklaste sättet att installera Windows NT 3.5 checked build är att ansluta en CD-ROM läsare direkt till datorn man vill installera NT på. Detta kräver att man har tillgång till de tre boot-disketterna för Windows NT 3.5.

1. Anslut CD-ROM läsaren till datorn (lägg till nödvändiga drivrutiner).

2. Mata in Windows NT Boot-disk i enhet A.

3. Starta om datorn.

4. När setup-programmet frågar efter de olika disketterna, matas de in.

5. När programmet frågar efter Windows NT CD:n, laddas CD-ROM läsaren med DDK-skivan (där checked build finns).

6. Sedan är det bara att följa instruktionerna. Det går bra att göra en minimal installation om man har ont om plats i Target-maskinen.

**OBS!** Glöm inte att installera TCP/IP-stacken (med connectivity utilities) om du vill kunna dela enheter över nätet med Target-maskinen. Detta är praktiskt då man kan uppgradera sin nyutvecklade drivrutin direkt över nätet från Host-maskinen utan att använda disketter.

För att kunna få access till Target-maskinens hårddisk från Host-maskinen över nätet måste man göra två saker:

1. I Target-maskinen går man in i Filhanteraren. Man markerar rotkatalogen på den lokala hårddisken och väljer i menyn "Disk" alternativet "Share As...". Där trycker man på "New Share..." och matar in ett lämpligt "Shared name". Sedan väljer man "Permissions" och sätter där enklast "Everyone- Full Control".

2. I Host-maskinen går man också in i filhanteraren. Där väljer man under "Disk" i menyn "Connect Network Drive". Man letar upp datorn man vill koppla sig till (i domänen eller i Workgroup, beroende på hur NT-installationen är gjord). Därefter skall den delade enheten visa sig under vald dator och det är bara att ansluta sig. Skulle den inte dyka upp automatiskt, kan man mata in "Path" själv eftersom man känner den från punkt 1 ovan.

### 9.2.7 Exempel på .INI-fil för egen drivrutin

Denna exempelfil används till IbsDrv.sys:

\registry\machine\system\currentcontrolset\services\IbsDrv
Type = REG_DWORD 0x00000001
Start = REG_DWORD 0x00000003
Group = Extended base
ErrorControl = REG_DWORD 0x00000001

(raden "Start = REG_DWORD 0x00000003" medför att drivrutinen får manuell start)

### 9.2.8 Editering av boot.ini

- Skapa ett DOS-fönster och gå till roten.
- Ändra filens attribut: `attrib -r -h -s boot.ini`
- Editera filen som ser ut t.ex. så här innan editering:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINNT35
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINNT35="Windows NT Workstation Version 3.5"
multi(0)disk(0)rdisk(0)partition(1)\WINNT35="Windows NT Workstation Version 3.5 [VGA mode]" /basevideo
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Windows NT Workstation Version 3.5"
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Windows NT Workstation Version 3.5 [VGA mode]" /basevideo
```

till följande utseende (fetstil markerar vilka delas som editerats):

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINNT35
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINNT35="Windows NT Workstation V 3.5 (chk)" /DEBUG
/DEBUGPORT=COM2 /BAUDRATE=56000
multi(0)disk(0)rdisk(0)partition(1)\WINNT35="Windows NT Workstation V 3.5 [VGA mode] (chk)" /basevideo
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Windows NT Workstation Version 3.5"
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Windows NT Workstation Version 3.5 [VGA mode]" /basevideo
```

Genom att ändra i textsträngen "Windows NT Works..." så ser man lätt vilken version som är för Kernel debugging. Tilläggen efter identifikationssträngen aktiverar Kernel debugging samt ställer in vilken seriell port och vilken hastighet som skall användas vid kommunikationen med Host-maskinen.

- Ändra tillbaka attributen: `attrib +r +h +s boot.ini`