

BOOT FILE

BB 200926

JOB: SY35022JOB

ENVIRONMENTS

NETENV
SY35022ENV
DLLENVIR
DOUBLENV
FS35022ENV
MFC1ENV
FD35022ENV
FILEPREFIX
CTENVIR

PAXLIBENV
LIB35022
—
ALFA3502
—
ALFA35022
—
LIBFS3502
"Project 1000" LIB35022

SOURCE

(~~TINITBOOT~~ LIB35022) (~~either BINITBOOT~~ ^{object}
TBOOT — Version 11.1
(~~TDOUBLIB~~ —)
TMFC1 —
THEGADRIVER "Project 1000"

OBJECT FILE

STDPLIB
BGETDRADR
BINITBOOT

ALFA3502
LIB35022
—

16 BIT ← 24 BIT
8000
X = CONVERT PLIB Y

BOOT

900926

220

DBIND

SMS SCUPR USER LIB.AT BOOT

SHYINO

user

~~boot~~

891222.1501

bootcross

bootcross

890518.836

bootfts

890608.1253

~~boot97~~

~~890528~~

Project

boot97

lib35022

881219

binitboot

-

891228

bboot

-

891228

tboot97

-

890413

tinitboot

-

870909

binitboot62

-

881219

tboot

-

891228

cboot97

-

890508

otboot

-

890608

tinitboot62

-

870909

boot111

cboot97

netem

realibem

✓

tinitboot

lib35022

✓

sys35022em

-

✓

sys35022lib

-

✓

tboot

-

✓

faovlelib

-

✓

dllemvir

-

✓

~~alt35022~~

~~downem~~

→ doubleem

alt35022

✓

→ fs3502em

-

✓

mtc1em

-

✓

fd3502em

-

✓

fixprtx

fixprtx

✓

tmtr1

libfs3502

✓

imgadrv

lib35022

ctemvir

lib35022

staplib

lib35022

xtboot

lib35022

otboot

lib35022

tboot

lib35022

shu 1000

boot112

alt35022

boot112

lib35022

boot112

lib35022

boot112

lib35022

boot112

lib35022

11.2

11.1

11.2

scsechur

ny

(reusable)

scsechur

ny

scsechur

ny

xtboot

tboot

delault = 10

= 40

and ((interrupt - sum and DE)

and ((interrupt - sum and DE - DE - DE - DE) = DE do

= DE) do

and ((interrupt - sum and DE - DE - DE - DE) = DE do

→ BB

ALAK 891124

Boot exception

boot ver. 11.1

file limit boot 35024 0000013B ← gages
3 gage

boot exception 000B at E0.82CF

~~> trigger~~ elb. shoe
~~< inc name >~~, line in - ma, code = h h h h
~~trigger~~ ~~trigger~~

Bool

lær 9

if < \$50 sa' 2.1

or \$20

elb. do nothing

→ HEJ

BOOT MA IKKE SURVIVE 21₁₆ 1 Yran[9].

M V H

HCV 901008

Kun debryggeren må skrive her.

^{ID reg.}
switches læses flere gange

↓
 cp CIO

set. fln = 1

check med jens klammorm

anbring cpA og cpa

✓

HLV:

ved fejl når udstøper ikke at komme

Uanset?

ud fordi restant arbejder for tidligt i natthold.

switche rederes kun én gang

ellers tages der ~~fra~~ rum

89.05.25/jocl

Procedure for brænding af RC3502 autoloader PROM'er

Det forudsættes, at der er genereret en crosslinket fil indeholdende initboot og boot, som vi kan kalde bootimage.

Procedure

1. På RC8000 laves en fil i fts-format :
bootfil = convertplib bootimage
2. Overfør bootfil til en Partner tilsluttet ETHER-NET vhja. rr.
F.eks.
rr -s <host> -u <user> -b -f bootfil a:bootfil.dat

Såfremt Partneren også har en brænder tilsluttet, springes næste punkt over.

3. Flyt disketten med bootfil.dat over på en Partner med brænder tilkoblet.
4. bootfil.dat skal nu splittes op.
Hertil bruges programmet **preblast** (MRW) :

preblast a:bootfil.dat

preblast laver to filer (a:bootfil.hi, a:bootfil.lo).

5. Placer en slettet PROM i brænderen.

Kald brændeprogrammet **burn**

(skal som minimum være version 2.0 / 89.10.09
JAHN / KNEH / HLV)

burn læser PROM'ens karakteristika og viser en menu.

Overfør filen, der skal brændes, i Intel-Hex format vhja. kommandoen **g** (burn spørger om filnavnet).

Burn beregner checksum for den overførte fil (papir og blyant).

Tast **b** for få brændt PROM'en.

Dette punkt gentages for den anden fil.

OP.JOCL.234
89.05.23

Nye autoloader PROMer til Norge

Grundet ændrede adresser på kommende OP2 front ends i Norge er der blevet genereret en ny BOOT til RC3502.

Den nye BOOT har flg. **karakteristika** :

Versionsnummer	:	10.7
Dato	:	89.05.23
PROM-numre	:	ROB629/ROB630
ROB629	:	
position	:	102 (MEM209); 90 (MEM206)
checksum	:	183A
ROB630	:	
position	:	91 (MEM209); 73 (MEM206)
checksum	:	2B8F

Ændringsbeskrivelse :

Selve **BOOT-processen** er ændret med hensyn til opkald, hvor adresse-tabellen gennemløbes cyklisk. I tidligere versioner kunne den første adresse, der forsøges opkald til, stamme fra den foranliggende tabel.

Den eksterne procedure **getdteaddr** er ændret med hensyn til adressetabellerne, idet flg. generelle ændringer er indført

110148 er ændret til **150084**
110049 er ændret til **150083**
110150 er ændret til **150082**
110151 er ændret til **150081**

Den flg. side viser det nye indhold i adressetabellerne.

Table 0 : lcg_n 4, lcn 2, lgth 13,
2422-150084-000
2422-150083-000
2422-150082-000
2422-150081-000
Table 1 : lcg_n 4, lcn 2, lgth 13,
2422-150084-000
2422-150083-000
2422-150082-000
Table 2 : lcg_n 4, lcn 2, lgth 13
2422-150081-000
2422-150017-000
Table 3 : lcg_n 4, lcn 2, lgth 13,
2422-150084-000
2422-150083-000
Table 4 : lcg_n 4, lcn 2, lgth 13,
2422-150082-000
2422-150081-000
2422-150017-000
Table 5 : lcg_n 4, lcn 2, lgth 13,
2422-150083-000
2422-150082-000
2422-150081-000
2422-150017-000
Table 6 : lcg_n 4, lcn 2, lgth 13,
2422-150084-000
2422-150083-000
2422-150082-000
2422-150081-000
2422-150017-000
Table 7 : lcg_n 4, lcn 2, lgth 13,
2422-150033-000
2422-150034-000

/*
Kommandoer til denne driver sendes i PI3 buffer til mailbox strøm 0,
driveren sender svar i samme buffer. Datadelen i buffer udpeges ved offset
og top i bufferbeskrivelsen.

Hvor intet andet er nævnt vil kommando og svar se ud som følger:

avn': nr	i kommando:	i svar:	
	u1 = kommando nummer	u1	uændret
			u2
			bytecount

```

read:1  i kommando:
        u3 = loginid
        u4 = connection
        i svar:
        u3      uændret
        u4      uændret
        bytecount
        datadel: antal bytes læ
                  de læste bytes

Læser fra fil til datadelen af kommando buffer.
Status i svar kan antage følgende værdier:
io_ok, io_removed, io_mode, io_end_of_file, io_disk_error,
io_catalog_error

write:2 i kommando:
        u3 = loginid
        u4 = connection
        datadel: bytes der skal skrives
        i svar:
        u3      uændret
        u4      uændret

Skriver fra datadelen i kommando buffer.
Status i svar kan antage følgende værdier:
io_ok, io_removed, io_mode, io_disk_full, io_disk_error

open:4  i kommando:
        u3 = loginid
        u4 = mode
        datadel: navn
        i svar:
        u3      uændret
        u4      connection

Åbner connection til navngiven fil
Mode: read=1, write=2, readtext=5, writetext=6
Status i svar kan antage følgende værdier:
io_ok, io_not_found, io_exhausted, io_drive_not_ready,
io_catalog_error, io_no_memory

close:8 i kommando:
        u3 = loginid
        u4 = connection
        i svar:
        u3      uændret
        u4      no connection

Nedlægger connection til fil
Status i svar kan antage følgende værdier:
io_ok, io_removed

login:28 i kommando:
        datadel: IMC port      12 bytes
                  maxdata      integer
                  user length   1 byte
                  user name     user length bytes
                  password length 1 byte
                  password      password length bytes
        i svar:
        u3 = loginid

Login
Status i svar kan antage følgende værdier:
io_ok

logout:32 i kommando:
        u3 = loginid
        i svar:
        u3 = 0

nextport:36 i kommando:
        i svar:
        datadel: IMC port

Vælger en ny port som ender på de tre første tegn i
IMC port navnet givet i user

Status i svar kan antage følgende værdier:
io_ok, io_not_found

load:40 i kommando:
        u3 = loginid
        datadel: navn
        i svar:
        u3      uændret

Load fra navngiven fil

```

Protocol no.ii, v.jj

EL/880418

Name: Loading LAN Stations Via FTS

Pages: pp

Programmable, loadable units with no private file storage may be attached to RcLAN as separate stations, e.g. communication front-ends or device controllers. Such units, in the following called *target units* may be loadable from special FTS servers, called *load servers*, across RcLAN. This note describes conventions for such use of FTS.

The target unit communicates as an FTS user, giving a user name whereby it is uniquely identified, but no password. The user name will be composed of a type part (RC3502, RC9310, LANMUX etc.) and a unit identifier. The unit identifier may e.g. be the serial number of a cabinet or board, which can typically be read from a PROM or NVM on the target unit. Preferably the same serial number shall be used which occurs in the Ethernet address of the unit.

Note this implies that if the unit identifier is changed, e.g. because of board replacement, the user name must be changed accordingly on the load server.

More specific conventions for individual types of units will be described in a later version of this note.

Often a load process is in stages, where the first stage is performed by a program in PROM and subsequent stages by program(s) loaded in preceding stage(s). In this way the PROM program need only know one object file name. Thus only this name needs to be fixed over the lifetime of the product.

Files read during the load process may include object modules as well as files containing configuration/customization parameters. All files needed to load a given target unit shall be collected in the home directory indicated by the user name of the unit on its load server(s). Normally there shall be exactly one load server for a given unit to avoid duplicates of unit-specific information and general confusion. Fault tolerance may be achieved by using a fault tolerant load server.

If the load server is an RC8000, replace "collected in the home directory" above with "visible from the bases", and note that the association between user name and bases is established by means of the FTS user catalog.

The use of a directory per target unit allows the use of standard file names. To avoid duplication of object files, links can be used. In the RC8000 case, the shared file may be given a wider scope which is visible from all the bases associated with the relevant units.

There is a mask bit for load service in the service and request masks which are part of the RCLLC SAP masks (ref.1). This means that a unit which sets only the load request mask will get links only to those stations which offer load service. This serves to reduce the number of FTS servers visible. A target unit must

attempt to log in to all visible FTS servers in turn, until it finds (the) one which accepts its user name. This is done by means of the getnames/masknames function and the *FTS convention for server port naming.

When not all stages of the load are performed by the PROM program, it is advised that the name of the load server be passed from the PROM program to the program for the next stage (and so on if necessary).

REFERENCES

- (1) RcLAN Protocol No.7, v.2/871026
Configuration of IMC Nodes in RCLLC Networks

Ændringer / ønsker til BOOT.

Der er to generelle ændringsønsker (ny STD-BOOT) :

- 1) Udvidet addr. message
- 2) Ny switch-tolkning.

Derudover er der to ønsker herudover af Speciel og midlertidig karakter - dvs. en overbygning på den nye STD-BOOT.

- 3) Udvidet tolkning af switche, således at der i Norge kan skelnes mellem en OP1-maskine og en OP2-maskine.
- 4) Udskrift af de udpegede adresser ved opstart.

I det flg. gennemgås punkterne mere detaljeret.

Ad 1)

addr-message udvides til :

u1 = BB _H	} som i dag
u2 = 66 _H	
u3 = BB _H	
u4 = 66 _H	

data	addr-lgth.	} som i dag,
	antal adresser	
	own-dte-addr	
	alternativ 1	
	alternativ 2	
	alternativ N.	

Udvidet addr.-message kræver:

- større buffer. (init-addr.-message)
 - indsættelse af adresseleugden for denne adresse-tabel (—)
 - successive kald af 'get-dte-addr' ved initialiseringen. (—)
- NB! De alternative adresser skal alle være $\neq$ own-dte-addr.

Ad 2).

Forslaget er beskrevet i 88.06.10 /jod.

Ændringen i tolkningen af switche skyldes behov for at skifte mellem forskellige adresse-tabeller i BOOT - blot ved at ændre switche.

Det generelle forslag giver mulighed for at have 16 forskellige adresse-tabeller (se dog også under pkt. 3).

En adresse tabel skal som minimum indeholde flg. oplysninger:

- antal adresser i tabellen
- lcn, lch (som BOOT skal bruge i dte-headeren).
- mulige opkaldsadresser.

function 'get-dte-addr' laves elustern og indeholder alle adresse-tabeller.

'get-dte-addr' leverer i dag antal af adresser, på frent den kaldes med parameteren - 1. Ellers leveres adressen med index = parameter-værdi modulo antal adresser.

'get-dte-addr' skal udvides til også at returnere lcn og lch.

NB Man kunne måske også udvide tabellen med et tabel navn, som BOOT skriver ud ved opstart.

F.eks.:

Portugal
eller Norge overgang trin 2.

Speciel Norge-version

Denne version skal bruges i Norge ved overgangen fra OP1 til OP2.

Ad 3)

Den nye STD-BOOT udvides med flg. fortolkning af TABLE-feltet i Switchene:

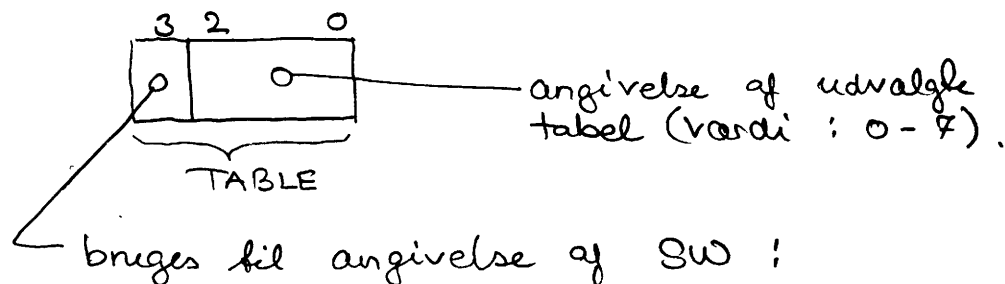


TABLE (3) = 0 :

angiver, at dette er en OP1 (BTI) masline og at 1. HDLC-controller har interrupt-niveau 16 (decimalt)

TABLE (3) = 1

angiver, at dette er en OP2 (RC) masline og at 1. HDLC-controller har (RC-std) interrupt-niveau 72 (decimalt).

Denne facilitet vil gøre det muligt at inddrage de forskellige maskiner til enten OP1 eller OP2 efter som behovet er i overgangsfasen.

Inklusionen af en masline foregår blot ved at switche ændres, uden at straps skal ændres på COM204.

Samtidig bevares muligheden for cyklisk at gennemløbe forskellige adresse-tabeller, som også vil være nyttig i en overgangsfase.

+ Aktuelle adresse-tabeller

Ad 4)

For at teste, at det er den rigtige tabel med de rigtige adresser, der er udvalgt via Switcher, vil det være en god ting at tabellen skrives ud ved opstart af den midlertidige BOOT/Norge-version.

(print-the-addr' kan bruges).

m.v.h.

Joel

88.06.10/jocl

Til : JHN, TL, JSH, PFR, JOCL, EBK, EG, JMI, TG,
 AH, PEB, BIN, BB, TOG, JKK, KHN, MIM.

 ! FORSLAG TIL NY SWITCH-SETTING VED !
 ! AUTOLOAD AF RC3502 VIA **X.25/3** !

I dag fortolkes switchene ved X.25/3-load således :

15	14	12	11	8	7	0

!	!	KIND	!	MODULE	!	!

D/E				Chan A/B		

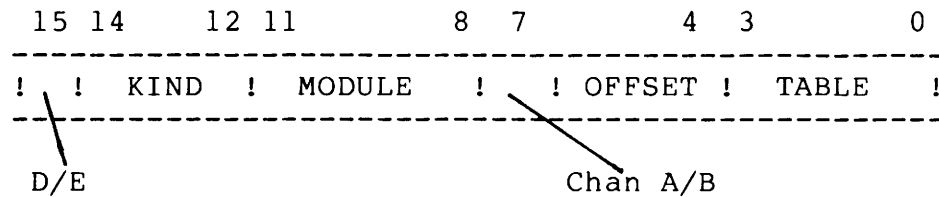
ADDRESS giver det absolutte interrupt-niveau for den controller, der ønskes loadet over.

RC bruger som standard niveau 72 (48H) for den **første** controller.

Dvs. at ADDRESS indeholder overflødig information, idet man kan nøjes med at angive **offset** til 1. controller.

Herved frigives fire bits, som kan bruges til andet - **f.eks.** til udvælgelse af adresse-tabel.

Forslag til ny fortolkning af switche :



D/E, KIND, MODULE, A/B er som hidtil.

OFFSET angiver load-controllerens offset til første controller (level 72).

- Dvs. 1. controller har OFFSET 0
 2. controller har OFFSET 1
 .
 .
 .
 8. controller har OFFSET 7 (max)

TABLE angiver hvilken X.25-adressetabel BOOT skal vælge til at komme ud i landet med sit load-request.

Sammen med BOOT er brændt en række adresse-tabeller.

Dette kunne eksempel-vis være :

tabel 1 : Norge
 tabel 2 : Portugal
 tabel 3 : Norge overgang til OP2 1. trin
 tabel 4 : Norge overgang til OP2 2. trin
 ... etc.

hvor midlertidige adresetabeller (overgangs-ordninger) kan bruges til andet, når de har udtjent deres formål.

Løsningen er ikke den mest geniale. Der skal stadigvæk brændes nye PROM'er, når en ny tabel skal inkluderes.

Der bliver også behov for at lave om på fortolkningen af switche i BOOT, når der skal loades via **LAN kombineret med X.25/3**, idet ADDRESS ved LAN-load forventes at angive interrupt-niveau for LAN.
 Også her har RC en standard for første MFC (level ?).

Dvs. at man her også kunne bruge OFFSET sammen med en konvention om, at LAN altid skal sidde forrest, når der er flere MFC'er.

OPERATING GUIDE skal laves om til at indeholde en generel beskrivelse af X.25/3-load, som suppleres med et lille ark, der fortæller hver kunde, der bruger X.25/3-load, hvorledes switche skal sættes i dette specielle tilfælde.

Den beskrevne løsning (såfremt den er realiserbar) skulle give os et pusterum til at finde en mere køn løsning på X.25-adresseproblemet (måske noget med EEPROM'er).

OBS

Skulle du se nogle problemer i denne løsning, så henvend dig til BB eller JOCL.
Tak.

87.12.07/jocl

MØDE OM NY STANDARD BOOTER TIL RC3502

Til stede : BB, JOCL

Problemet er at få standard boot og Norge-boot smeltet sammen igen.

Det betyder bl.a. at

- 1) X.25/3-autoload skal flettes ind i standard boot
- 2) det skal designes, hvorledes boot forsynes med X25-adresser.

Ad 2

En mulighed er at boot har software til at requeste X.25-adresser fra et tilkoblet device (f.eks. en vifte af RC45-skærme).

I RC45 kunne relevante X.25-adresser være lagt ind i NVM.

Vi kom på mødet ikke nærmere ind på dette.

Ad 1

Der er ved at mangle autoloadkind-bits på switchene. Vi foreslår flg. autoloadkinds:

wd
wd + com204
fpa
gci
com204
com205
lan
lan + X.25

Implementation af LAN + X.25

Boot forsøger først et antal autoloading-requests via LAN med "korrekt navn". Herefter forsøges et antal gange med "fejllindikator navn". Hvis intet har givet resultat, ændres autoloadingkind blødt til "X.25" og boot forsøger med denne autoloading kind, indtil der er "bid".

Da switchene nu ikke er sat til X.25-autoloading men til LAN-autoloading, må boot gå ud fra nogle default-værdier :

Enten

- X.25 adresserne kaldes i et cyklisk gennemløb af adressetabellen (module = 0)

Eller

- en bestemt X.25-adresse i tabellen kaldes permanent (module - 1 angiver index).

Ved skift fra LAN-autoloading til X.25-autoloading sættes module til 0 (permanent opkaldsadresse altså ikke længere muligt).

Endvidere skal altid 1. controllers 1. kanal (ch.A) benyttes. Dvs. startadresse i 90.0000 samt interruptniveau 48Hex.

Uafklarede punkter

- Hvilken autoloaddkind skal vige pladsen for X.25-load via com205 ?
Eller bliver com205 altid X.25/3 ??
- Er det muligt at finde en ubenyttet bit på switchene til udvidelse af antallet af mulige autoloaddkinds ?
I "module" måske ?? Eller måske inddagelse af D/E-bitten ???
- Hvordan får boot fat i X.25-adresser ?

Aktiviteter på kort sigt

✓ **JOCL** markerer X.25-autoload delen i Norge-boot og leverer til BB.

BB fletter standard boot og X.25-autoload sammen og producerer PROM.

OP-gruppen aftester den nye booter.

Tidshorisont : Klar til test medio januar 88

1. INTRODUCTION
 - 1.1 Revision Record
 - 1.2 Terms, Notation, and Abbreviations

2. DATA STRUCTURES
 - 2.1 Semaphores and Pools
 - 2.2 Buffers and Messages
 - 2.3 Tables and Arrays
 - 2.4 Files

3. PROGRAM STRUCTURE
 - 3.1 Coding Principles
 - 3.2 Modes
 - 3.3 Submodules

4. EXTERNAL COMMUNICATION
 - 4.1 Input
 - 4.2 Output

5. SYSTEM GENERATION
 - 5.1 Compile-time Hardware
 - 5.2 Compile-time Software
 - 5.3 Installation

6. PROGRAM EXECUTION
 - 6.1 Run-time Hardware
 - 6.2 Run-time Software
 - 6.3 Error Messages
 - 6.4 Test Facilities
 - 6.5 Test Report

A. REFERENCES.

TABLE OF CONTENTS

PAGE

1.	INTRODUCTION	
1.1	Revision Record	
1.2	Terms, Notation, and Abbreviations	
2.	DATA STRUCTURES	
2.1	Semaphores and Pools	
2.2	Buffers and Messages	
2.3	Tables and Arrays	
2.4	Files	
3.	PROGRAM STRUCTURE	
3.1	Coding Principles	
3.2	Modes	
3.3	Submodules	
4.	EXTERNAL COMMUNICATION	
4.1	Input	
4.2	Output	
5.	SYSTEM GENERATION	
5.1	Compile-time Hardware	
5.2	Compile-time Software	
5.3	Installation	
6.	PROGRAM EXECUTION	
6.1	Run-time Hardware	
6.2	Run-time Software	
6.3	Error Messages	
6.4	Test Facilities	
6.5	Test Report	
A.	REFERENCES	

1. INTRODUCTION

1.

The main purpose of the BOOT module is to take over control of the RC3502 after the micro program of the RC3502 has initialized the machine (Def [1]).

The BOOT module ^{an absolute binary} autoloader ^{program system} optionally from an external device, and includes the programs residing in the EPROM part of the address space.

After autoload and inclusion, a repertoire of basic process incarnations are created and started, whereafter control is passed to the basic program system.

The purpose of this manual is not to give a detailed description of the BOOT module, but to present the general outline of strategy and data structure. For detailed information please study the program list.

re- vision	made by	date	revision concerning
1.01	BB	8410.11	First Edition

revision concerning

First Edition

1.2 Terms, Notation, and Abbreviations

1.2

2. DATA STRUCTURES

bootloadhist

inithist

programcount

buf

currregset

bootmodelino

upper-limit

ram_pointer

save_ram_pointer

pointer-move

cur_adamdata

bootlevel

program_pointer

program_top

sumset

initialisierung

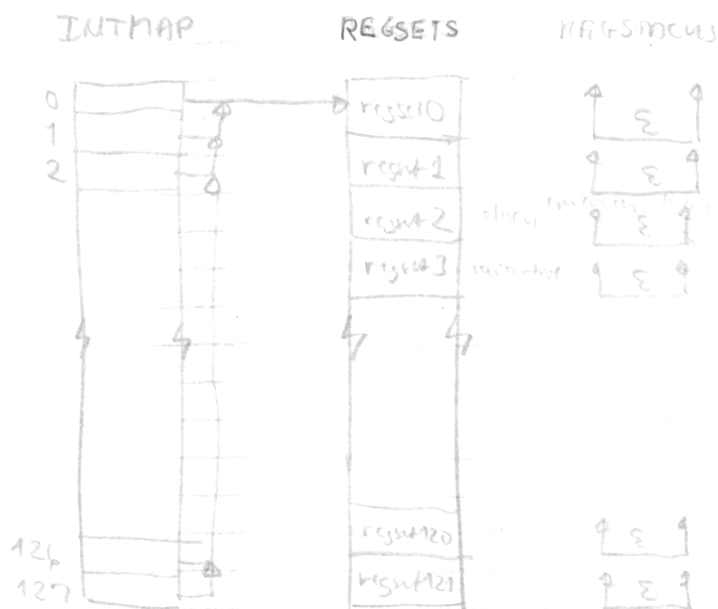
message number 20
out
bitmask pointers
ram pointer
platform

2. DATA STRUCTURES

2.

The global data structures as seen from the BOOT module consist of the context sets, (interruption bit array) and the physical memory modules (RAM and EPROM).

The initial state as passed by the mixer program may be illustrated by the following figure



regset0	ps	+	+	+	+
	sb	+	+	+	+
	gt	+	+	+	+
	lt	+	+	+	+
	lv	+	+	+	+
	lm	+	+	+	+
	ib				
	ic				

membyte (eo.0019)
mem (eo.001a)

reset0 executes the INITBOOT program (see 3.3) and ends up as the 'dummy register', counting interrupts not directed to devices.

reset1 executes the BOOT program

reset2 is used in different contexts: The first is for initialization of the interruption bit table.

The second depends on the switch value^(see 3.2). If kind is COM204, 'reset2' executes the COM204 interruption service program.

If kind is WD, 'reset2' executes the clock program.

reset3

executes the SAIZ01 control program, when the switch kind is WD.

reset4121, reset4120, ..., reset4116

are allocated the six basic process incarnations in a simple 'bottom up' strategy. (see PROCEDURE bootcreate).

After autoloading from an external device,
a RAM module has the following layout

$$\{ \langle \text{program} \rangle \}_1^n \{ \langle \text{RAM sumcatalog} \rangle \}_0^1$$

$$\langle \text{program} \rangle = \langle \text{descriptor segment} \rangle \langle \text{code} \rangle$$

EPROM modules have the layout

$$\{ \langle \text{program} \rangle \}_1^n \langle \text{FFFF word} \rangle \langle \text{EPROM sumcatalog} \rangle$$

A ^{memory} module conforms to the conventions:

1. The first program always starts in the second word (byte 2) of the module. ^(except CO, CC)
2. Memory allocation for autoloaded programs and incarnation descriptors is always from CO.0100 and onward. ^{So in module CO, the first program starts in CO.0100}
3. The autoloaded programs must be consecutive memorywise in the 0, 02, 04, range CO, C2, C4, , de, starting in CO.

All allocated objects

- programs
- basic incarnation descriptors and stacks
- stop message header
- RAM sum catalogs

all prefixed (or contain) a link record (or subrecord) of 8 bytes

descriptor-length	-
no-of-pages	-
page size	-
last-page-length	-

The length of an allocated object (including the link record itself) is calculated after the general formula (unsigned arithmetic operations)

$$\text{length} := \text{descriptor-length} + (\text{no-of-pages} - 1) \times \text{page size} + \text{last-page-length}$$

Chaining through a module ^{takes place} via the link records and a module is terminated by a link record, where 'descriptor-length ≤ 0 '.

Program:

A program is

PROGRAM =
link:

RECORD

```

    descriptor-length , (* length of link record *)
    no_of_pages      ,
    pagesize         ,
    last_page_length ,
    kind              : integer; (* = 1, 2, 3 *)
    name              : alpha;
    entry_point       : addr;
    exception_point   : addr;
    default_appetite  :
    last_param_offset :
    ? ? ? ?           : integer;
    no_of_params      : integer;
    params             : ARRAY (1..no_of_params) OF
                        RECORD
                          kind
                          | size : integer;
                        END;
    END;
    Code:
    : ARRAY (1..(no_of_pages-1)*pagesize
              + last_page_length)
      OF byte;
  
```

END;

The program records are allocated by BOOT, but the contents is generated by the crosslink program.

Incarnation Stack:

incarnation_stack =

RECORD

link: RECORD

```

    descriptor-length , (* = 8 *)
    no_of_pages      , (* = 1 *)
    pagesize         , (* = 0 *)
    last_page_length : integer;
  
```

END;

stack: ARRAY (1..last_page_length) OF byte;

END;

The 'stack' part contains the incarnation descriptor, actual parameters, and room for declared objects and routine activation records. The incarnation stacks inclusive the link record is ^{and initialized} allocated by BOOT.

Stop Message Header

```

stop_message_header =
  RECORD
    link : RECORD
      descriptor-length , (* = 8 *)
      no_of_pages       , (* = 1 *)
      pagesize          , (* = 0 *)
      last-page-length : integer; (* = 28 *)
    END;
  header : message_header;
END;

```

The 'header' part is signalled to ALLOCATOR after unblock and initialization passing the top address of RAM allocated by BOOT. The Stop Message Header is allocated by BOOT and initialized.

RAM Sum Catalog

```

ram_sum_catalog =
  RECORD
    link :
      RECORD
        descriptor-length , (* = 10 *)
        no_of_pages       , (* = 1 *)
        pagesize          , (* = 2048 *)
        last-page-length , (* = 68 *)
        kind              : integer; (* = 0 *)
      END;
    date : coded-date; (* like page 1 in a program *)
    time : coded-time;  (* like page 1 in a program *)
    sum : ARRAY (1..32) OF integer;
  END;

```

The ^{total} sum catalog is generated by the CROSSLINK program. (inclusive the link record)

A sum word covers 'pagesize' consecutive bytes in this module.

Bigger pagesizes may be specified.

The contents of the RAM Sum Catalogs are checked by the BOOT program and deallocated in the modules that contain programs only, - not incarnation stubs etc.

EPROM Sum Catalog

This format is unknown by the BOOT module.
The check is performed by the LINKER
on request from BOOT.

BOOTSTACK, STACK1, STACK2

The INITBOOT subprogram moves and
initializes the stack of BOOT to the last
RAM module in the ^{autoload} range c0, c2, ..., de, in
order to give ^(maximum) room for autoload from c0, c2,
and so on.

Depending on the autoload kind, the BOOT module
creates 0, 1, or 2 individual channel programs.
These incarnation descriptors and stacks
are initialized in the arrays STACK1 and
STACK2, which therefore are deallocated
together with the BOOTSTACK, when BOOT
dies and ALLOCATOR takes over control
of the RAM memory.

2.1 Semaphore and Reference Variables

2.1 Semaphores and References

2.1

channel_msg

program_msg

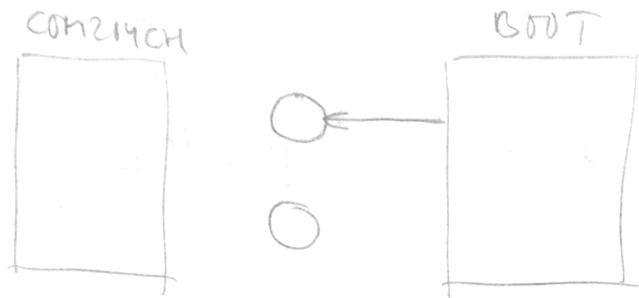
buffer_msg

r1

secret_vector

2.1.1 Semaphores

2.1.1.



2.1 Semaphore and Reference Variables

2.1

A number of semaphore and reference variables are declared and used for internal communications between BOOT and the channel subprograms, and between BOOT and some of the basic process incarnations.

Besides a global semaphore pointer array is initialized for use by the Run Time System, the so-called SECRET_VECTOR.

2.2 Buttons and Messages

2.2

init m.wahlenden

2.3 Tables and Arrays

base_stack

stack1

stack2

initialising

2.3

6.4 Test facilities

6.4

A. REFERENCES

A.

1. RC3502/2 Reference Manual.

Jens Kristian Kjergård, October 1983

RCSL No: 52-AA1192.

2. RC3502/2 Operating Guide

Bo Bagger, August 1984

RCSL No: 99_0_00771

3. RC3502/2 Real-Time Pascal, Reference Manual

Bo Bagger Larsen, October 1983

RCSL No: 52-AA1167

4. RC-FS File System Nucleus, Reference Manual

Jens Kristian Kjergård, June 1984

RCSL No: 99_0_00761

5. RC35 SAI Driver, Maintenance Manual

Jens Kristian Kjergård,

RCSL No: ?

6. RC3546 HDLC Driver, Maintenance Manual

Per Mondrup,

RCSL No:

7. PAXNET, Down Line Load, RC3502, HDLC Down Line Load

Tom Skovgaard, October 1982

RCSL No.: